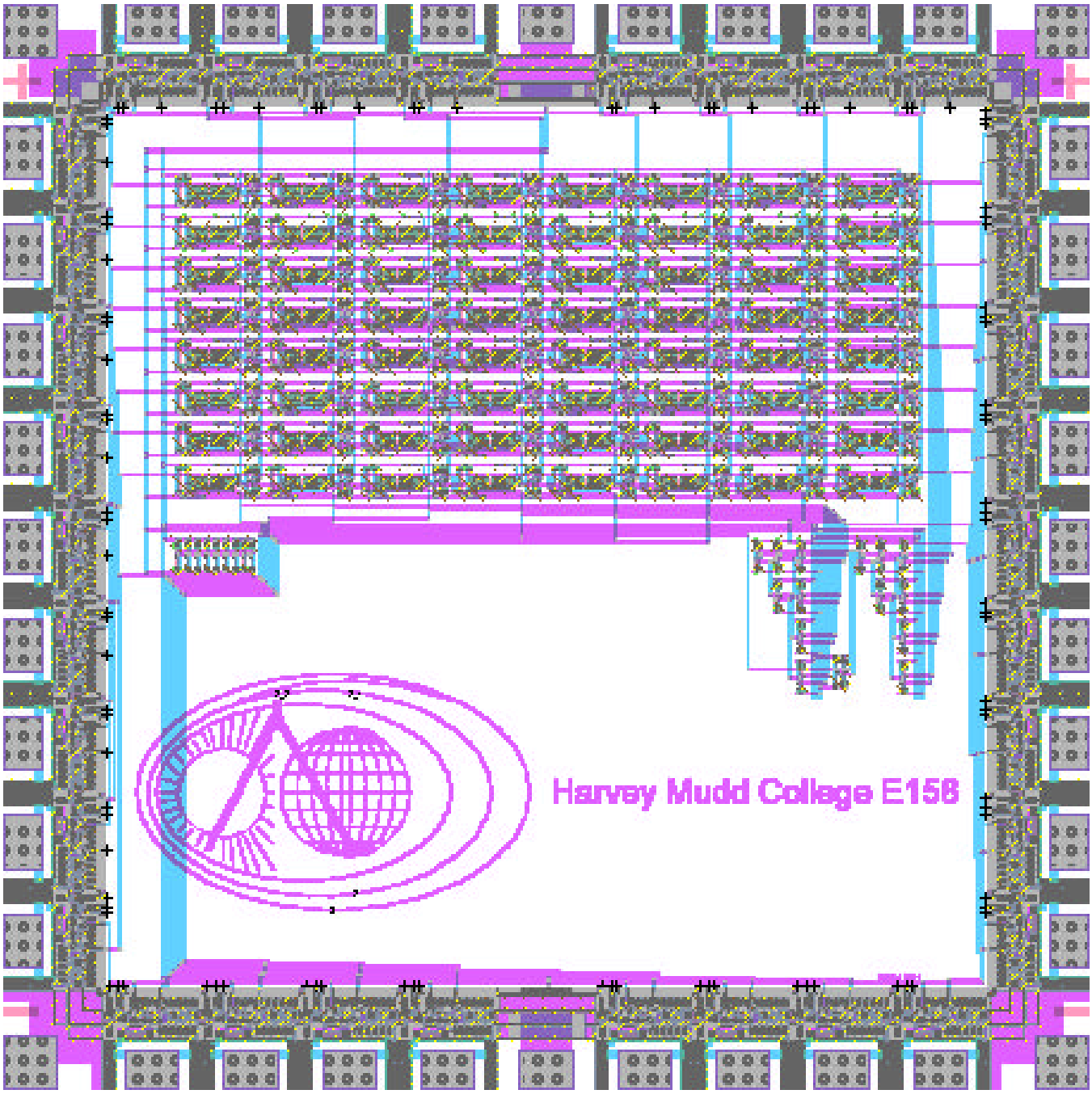
8-BIT UNSIGNED MULTIPLIER

Charles Hastings
David Hopkins



Charles Hastings

David Hopkins

E158

04/11/01

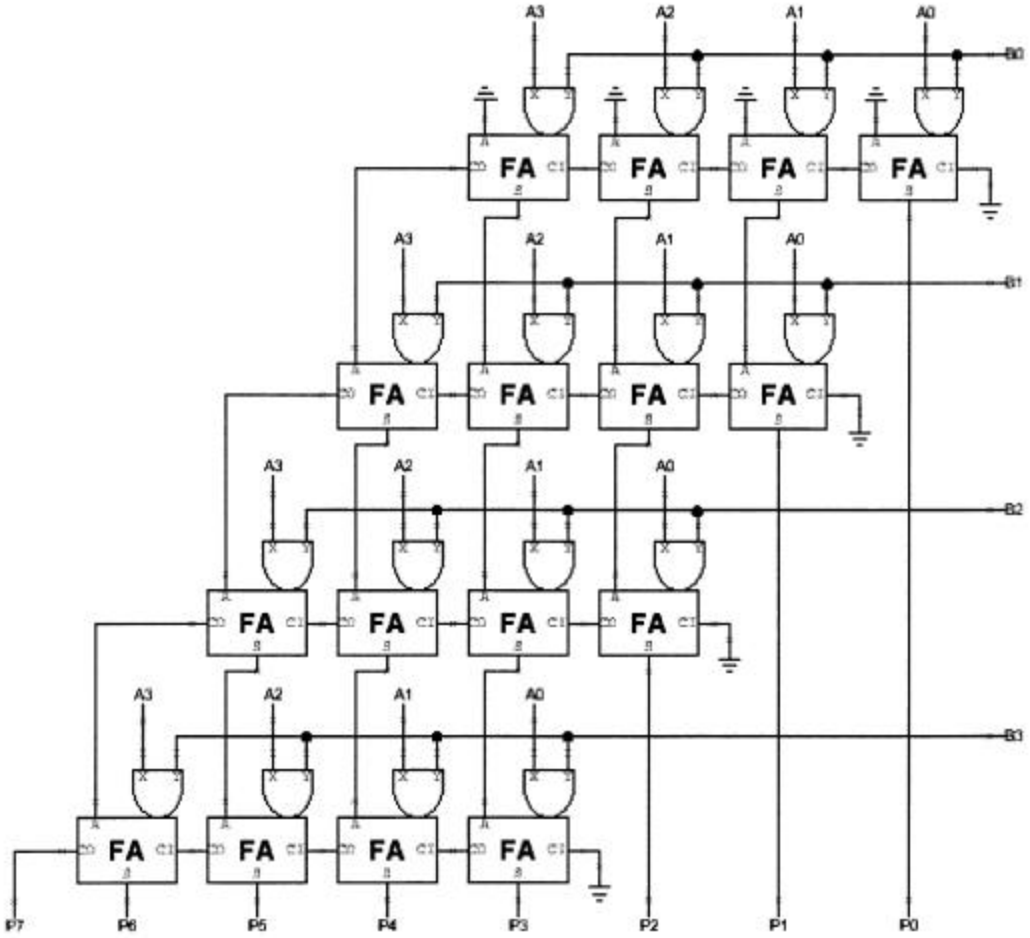
UNSIGNED 8-BIT MULTIPLIER

Overview

We designed an unsigned 8-bit by 8-bit combinational array multiplier. The chip takes two 8-bit unsigned values, multiplies them, and outputs a 16-bit result. The chip also includes zero-detect logic.

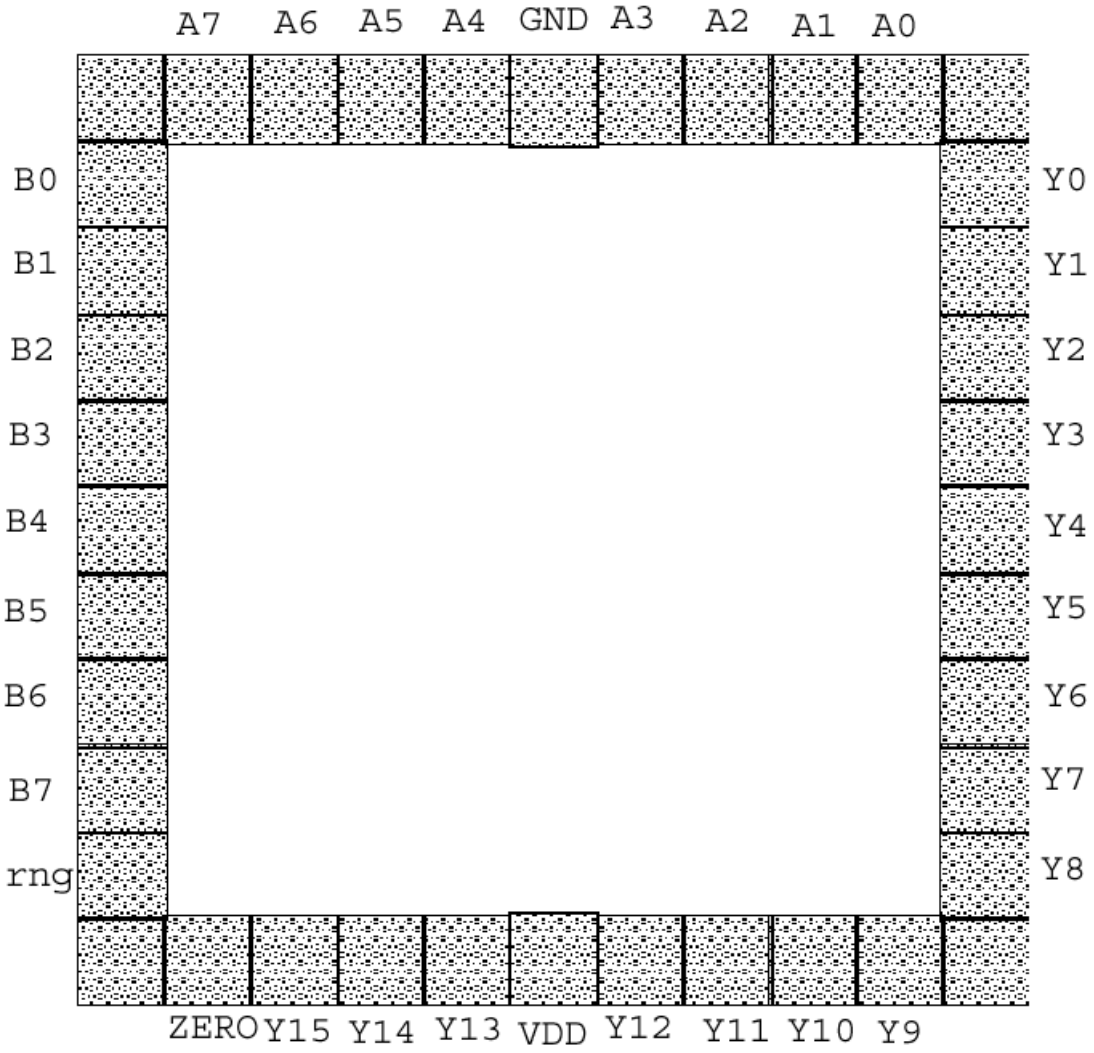
The multiplier design requires 64 full adders along with 64 AND gates. These can be organized in an array fashion, with the *n*th output from one row of full adders leading to input *n-1* in the next row. When an output is at the beginning of a row, it becomes a member of the product.

This array of full adders is intertwined with an array of AND gates. The gates serve to compute the partial products of the multiplication, which are then summed by the network of full adders. We chose to integrate the AND gate with the fuller adder layout into a cell called mulcell. This allowed for easy organization of the mulcell into an array of 64. The array was broken down into eight rows (called MulRow), each of which contained eight mulcells in a linear array. A 4 by 4 array multiplier is shown below. This design was used as a model for our implementation.



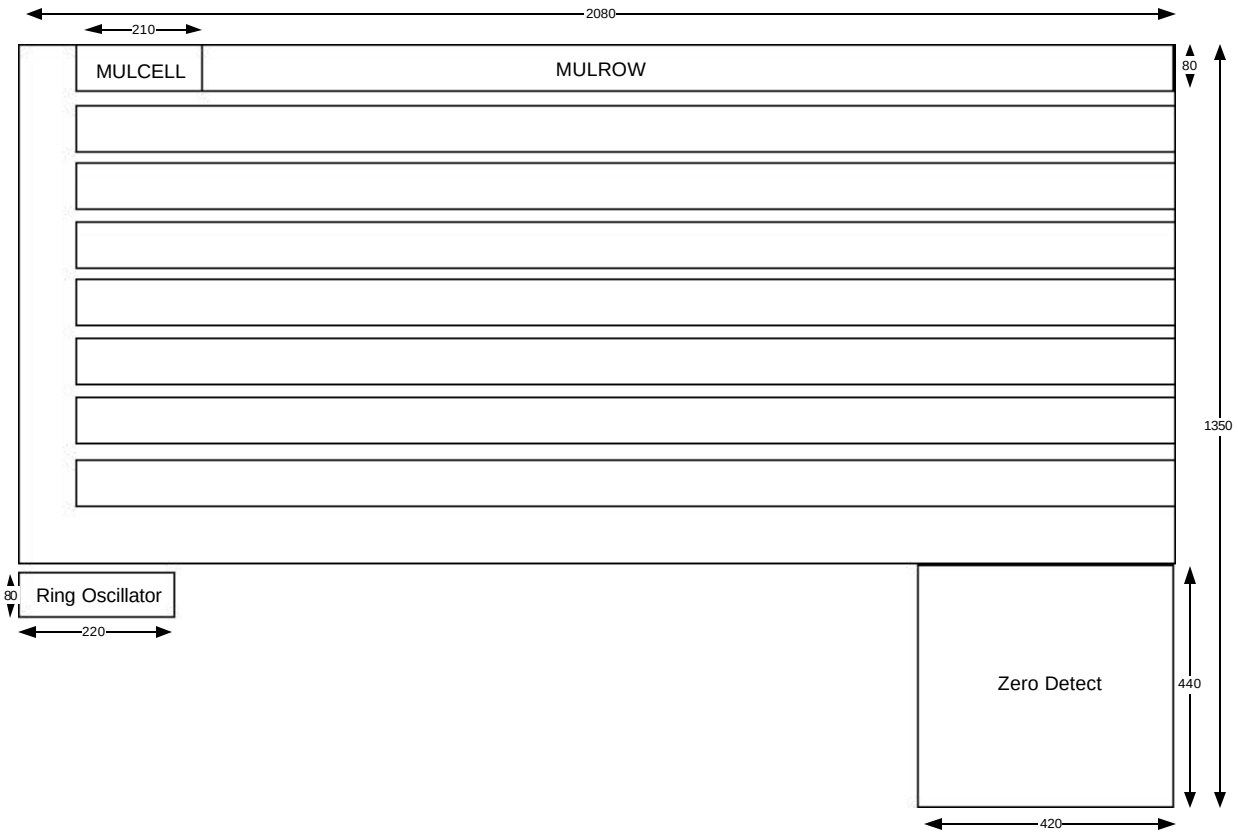
4-Bit Array Multiplier
From <http://6004.lcs.mit.edu/Spring00/handouts/L09.pdf

Zero detect was accomplished by comparing all output bits of the product. If all are zero, zero is high, or else zero is low. The zero detect logic used is a tree of 2-input logic gates. The lowest level of the tree is 8 NOR gates, followed by 4 NAND gates, 2 NOR gates, and finally a single AND gate.



Padframe for design

Pin Configuration	Pin	I/O	Description
	A[7:0]	Input	Multiplicand
	B[7:0]	Input	Multiplier
	Y[15:0]	Output	Product
	ZERO	Output	Zero detect
	rng	Output	Ring Oscillator



Floor Plan for unsigned 8-bit multiplier

Area Estimates

Shown below is a table listing each cell that the design required, the estimated size of the cell, and the actual size. Note that the overall size is somewhat larger than estimated because the area is measure as a rectangle that surrounds the entire design, and much of the space within that rectangle does not contain any useful components. Otherwise, most of the cells were only a bit bigger than expected, excluding the zerodetect, which was not optimized for area.

Cell	Est. Width	Est. Height	Est. Area	Width	Height	Area
mulcell	210	80	16800	220	80	17600
MulRow	1640	80	131200	1900	80	152000
multiplier	1640	960	1574400	2000	900	1800000
zerodetect	730	80	58400	440	420	184800
ring osc.	270	80	21600	220	80	17600
core	1640	1040	1705600	2080	1350	2808000

Design Time Data

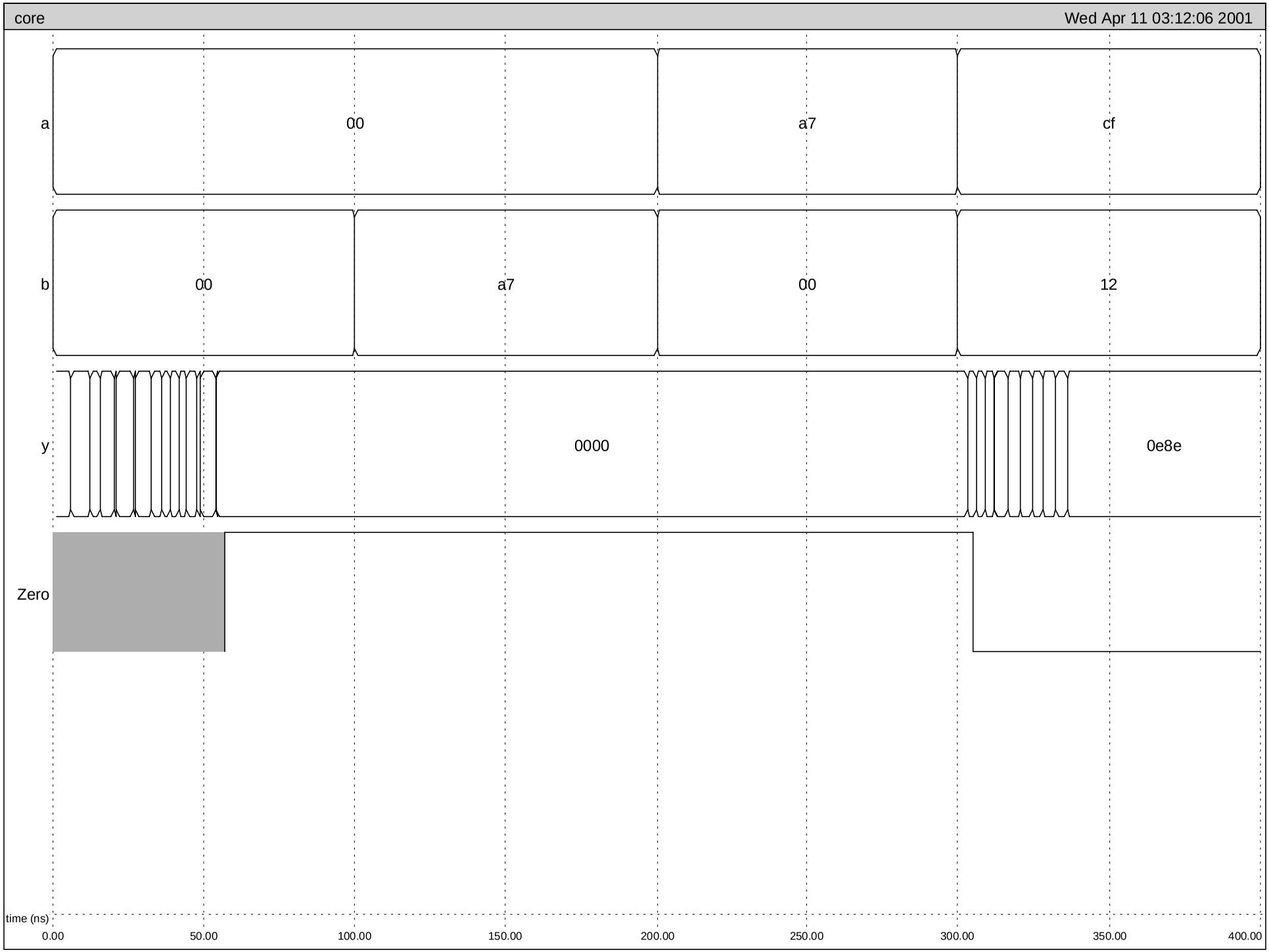
Shown below is a table listing each cell that the design required, and the design time spent on schematics, layout, and design verification for each of the cells. The total design time spent by the team was approximately 73 hours. Also shown is the number of transistors in each cell, and a time per transistor in the far right column. This number may be a bit misleading, as most of the transistors after the first level of the design come from multiple instantiations of the lower level, not new layouts. The time spent on the higher levels was primarily focused on routing of interconnections and placement of cells.

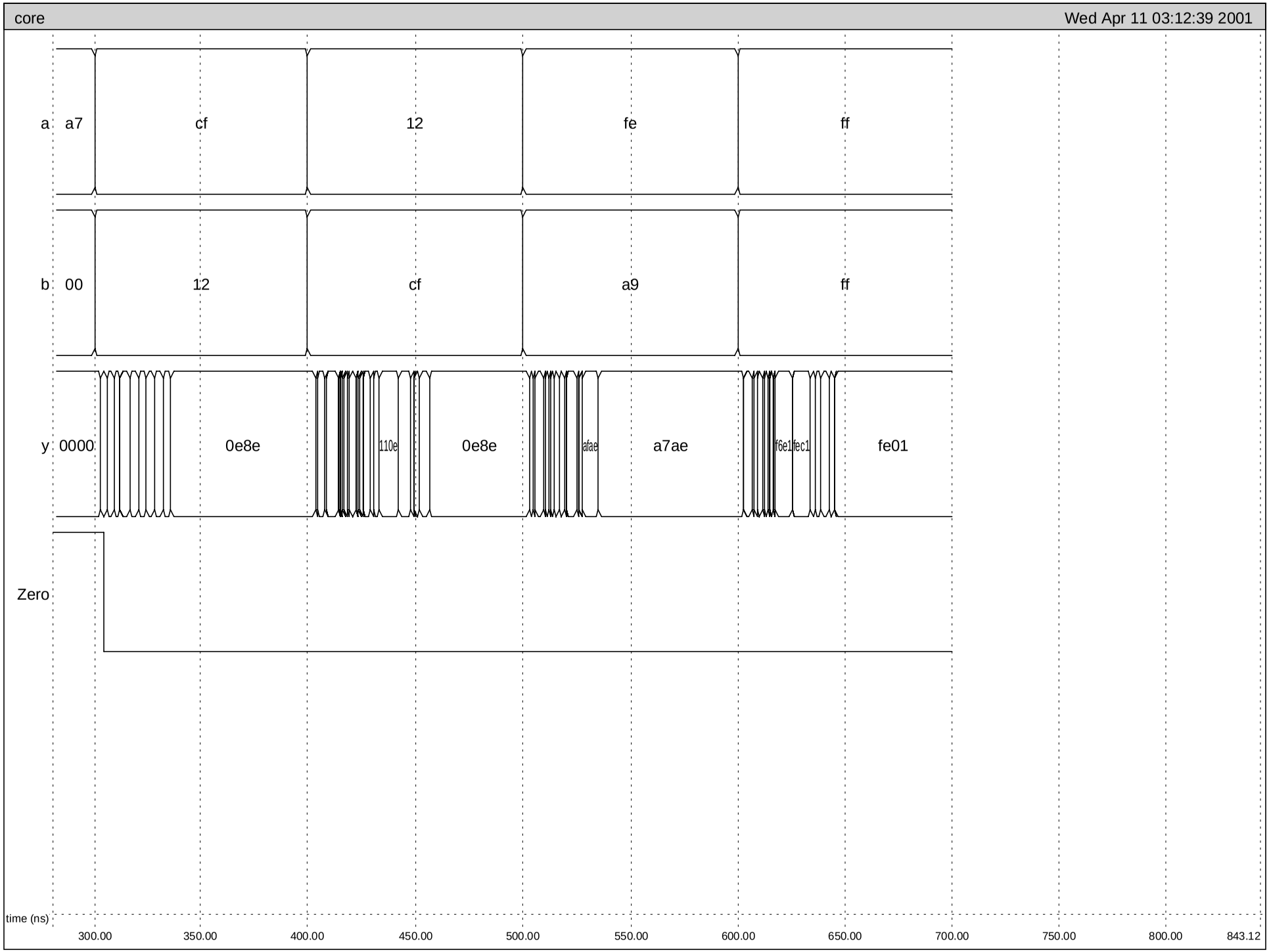
Cell	Schematic	Layout	NCC/ERC/DRC	# transistors	total time/transistor
mulcell	1	3	1	34	0.147058824
MulRow	1.5	9.5	2.5	272	0.049632353
multiplier	3	13	3.25	2176	0.008846507
zerodetect	1	4	1.5	62	0.10483871
ring osc	0.5	1	0.25	10	0.175
core	1.25	14	4.5	2248	0.008785587
top	N/A	7	N/A	2248	0.003113879
	total time:	72.75			

Simulation Results

Shown on the following two pages are the simulation waveforms for a few selected cases. We felt that these “corner cases” give substantial proof that the system is operating correctly.

In addition to these few cases, test scripts were written to facilitate the testing of our design for all possible inputs (256 * 256 cases). These test scripts are in appendix A.


core
Wed Apr 11 03:12:06 2001
a
00
a7
cf
b
00
a7
00
12
y
0000
0e8e
Zero
time (ns)
0.00
50.00
100.00
150.00
200.00
250.00
300.00
350.00
400.00


core
Wed Apr 11 03:12:39 2001
a
a7
cf
12
fe
ff
b
00
12
cf
a9
ff
y
0000
0e8e
110e
0e8e
afae
a7ae
f6e1
fec1
fe01
Zero
time (ns)
300.00
350.00
400.00
450.00
500.00
550.00
600.00
650.00
700.00
750.00
800.00
843.12

Verification Results

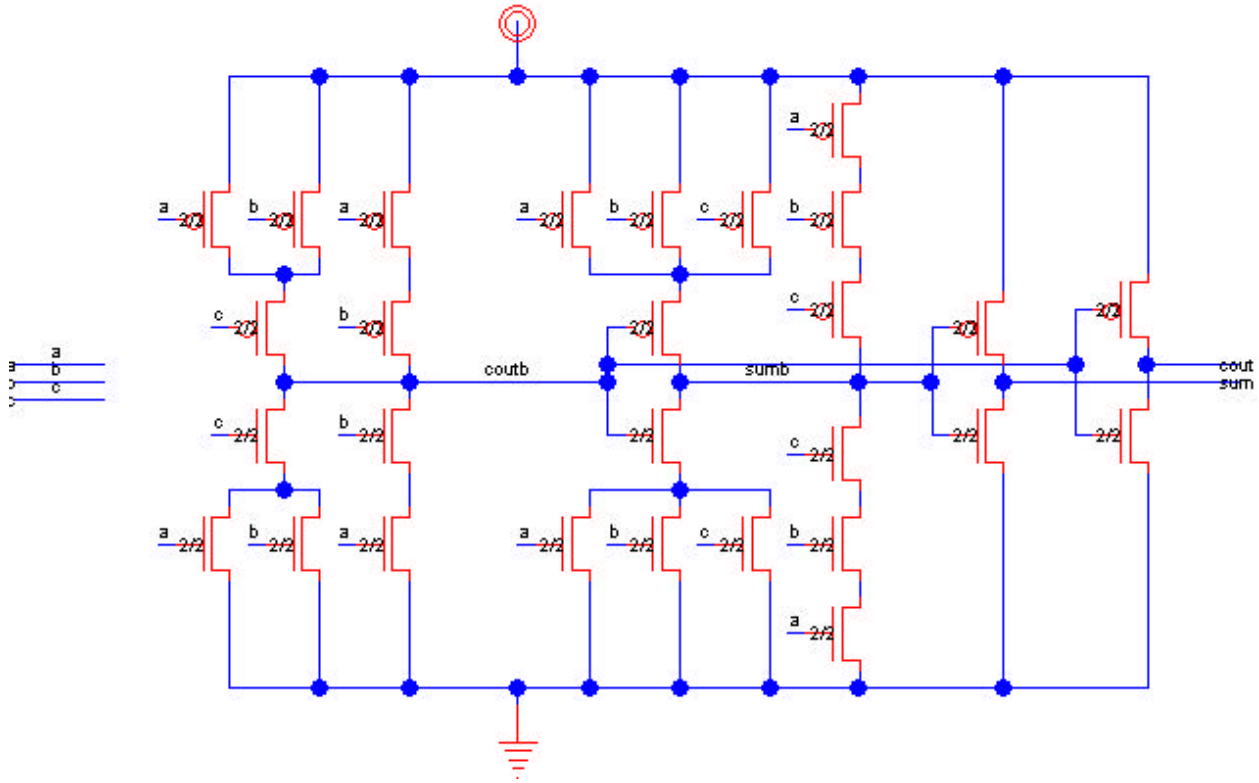
The entire design, including all leaf cells and high-level cells (excluding the pad frame), passed DRC, ERC, and NCC. For DRC and ERC, Electric was used, while Gemini was utilized to do a network comparison.

Postfabrication Test Plan

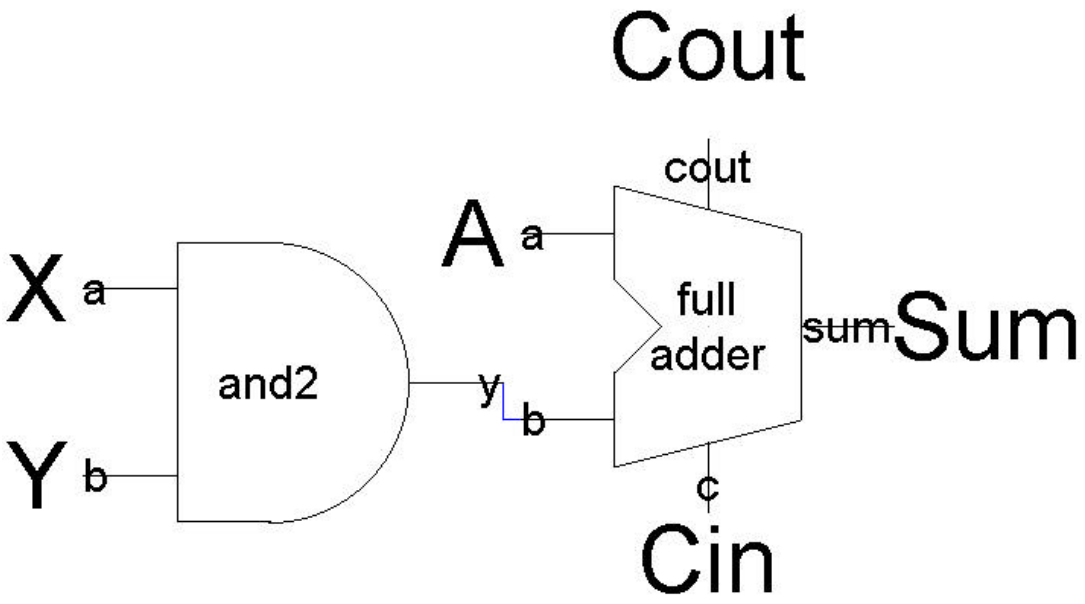
Once the chip is returned for testing we plan to initially hook up the power and ground pins, and verify with an oscilloscope that the ring oscillator is functioning. Given that the output of our chip is only dependent on the current inputs, the testing will be much easier than a system whose output is dependent upon previous inputs. We will verify the correct operation of the chip using our 'corner cases'. These will be supplemented by a series of random tests, utilizing random inputs.

Schematics and Layout

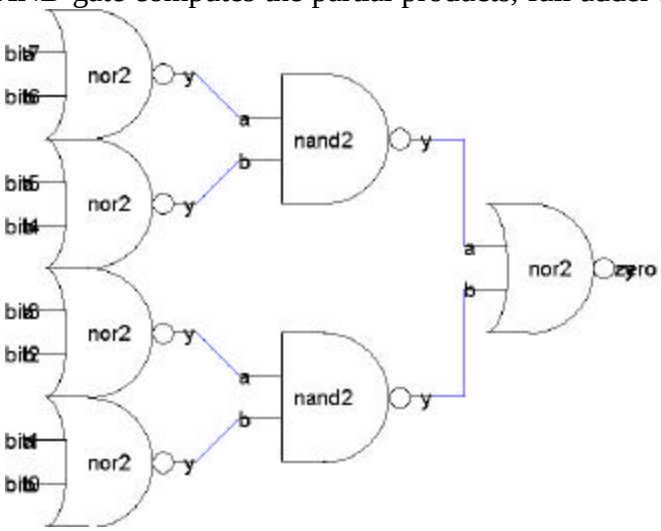
The next few pages contain schematics and layouts for all of the cells used in the design.



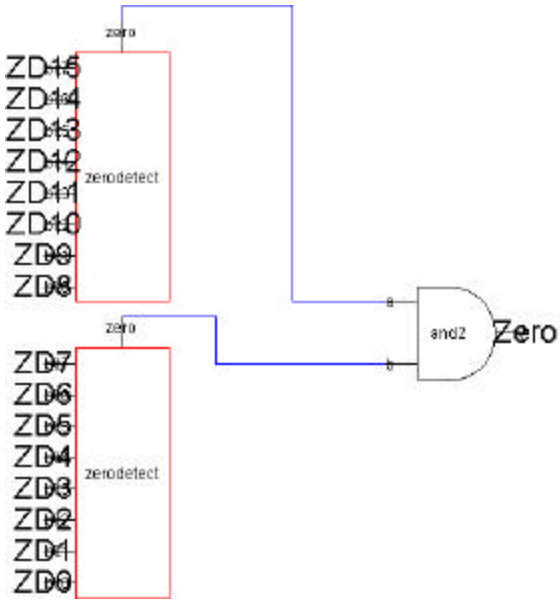
fulladder: a standard full adder schematic



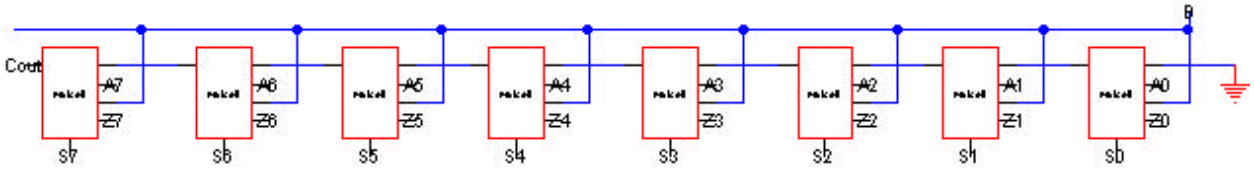
mulcell: AND gate computes the partial products, full adder sums them



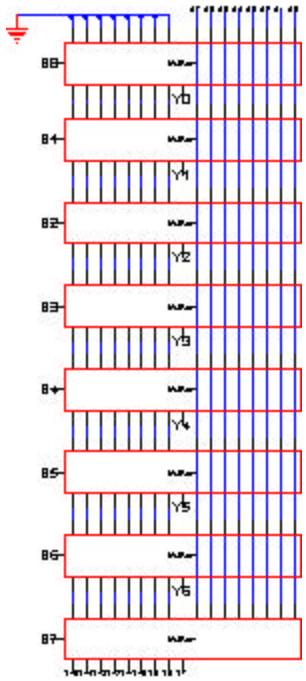
Eight bit zero-detect



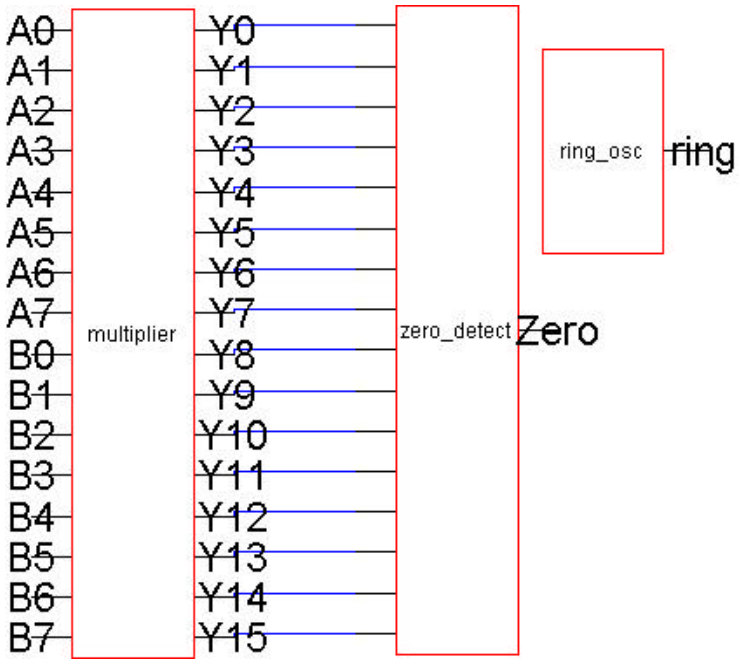
16 bit zero-detect



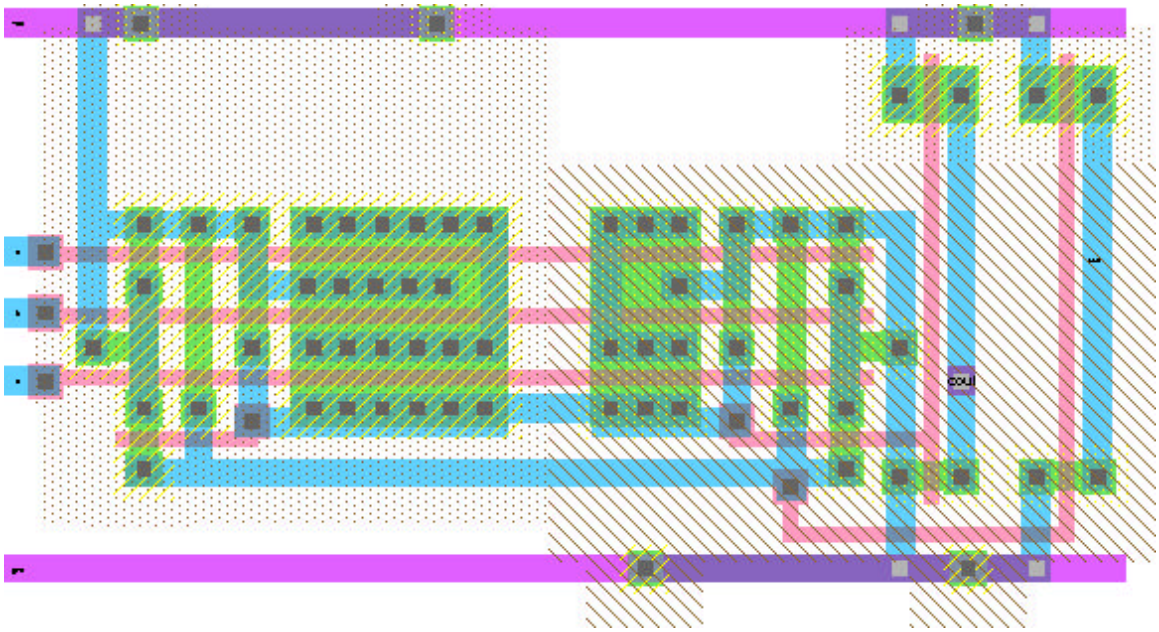
MulRow: A single row in the array, containing 8 mulcells in a linear array



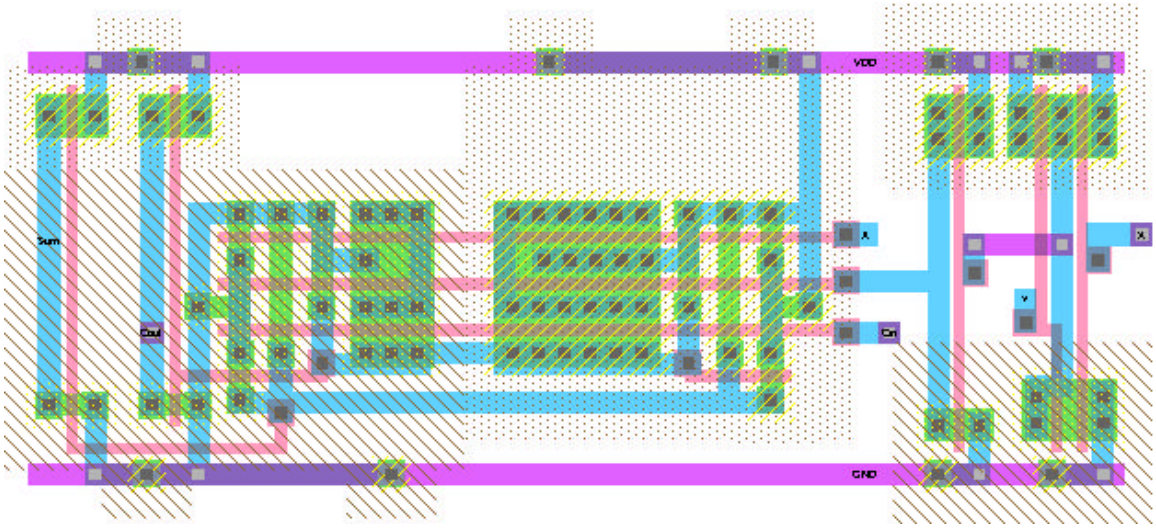
Multiplier: A column of 8 MulRow's forms the multiplier cell



Core: Top level schematic containing multiplier, zero detect, and a ring oscillator

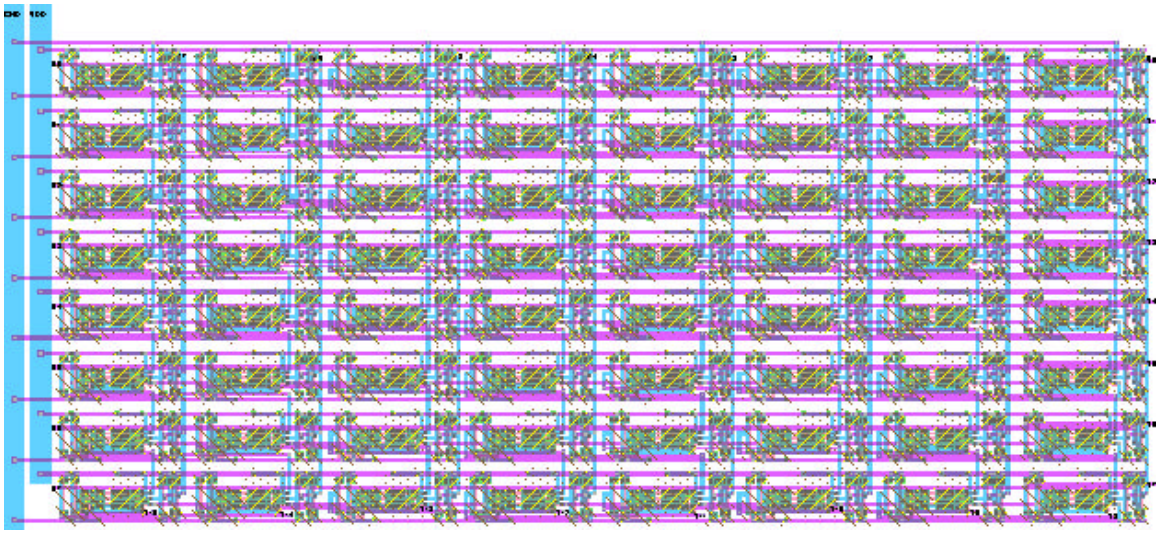


Layout of Fulladder cell

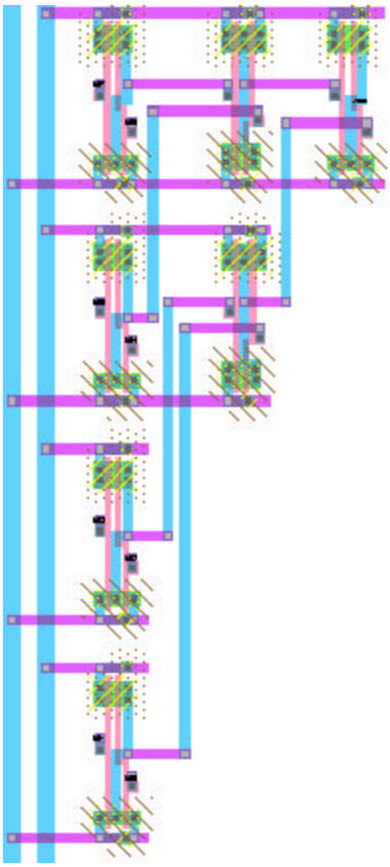


Layout of mulcell: fulladder and AND gate

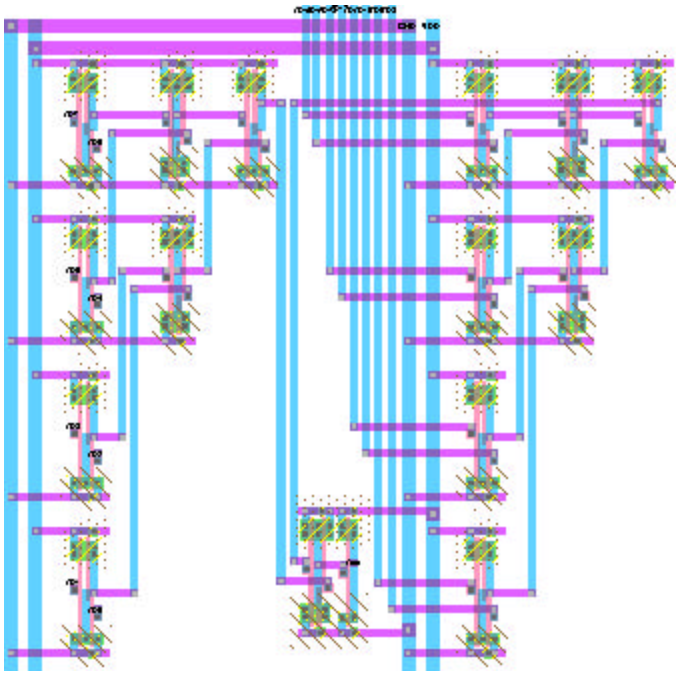
Layout of MulRow: 8 mulcell's in a linear array



Layout of multiplier: 8 MulRow cells in a vertical array



Layout of zerodetect: 8 bit zero detect cell



Layout of zero_detect: 16 bit zero-detection using 2 – 8 bit detectors

Layout of core: all components excluding pad frame

Appendix A: Test Scripts

```
#include <iostream.h>
#include <string>
#include <math.h>

string dec2bin(int x);

//
// Generates binary products from 0x00 * 0x00 to 0xff * 0xff.
//

void main() {
      for (int a=0 ; a<=255 ; a++) {
            for(int b=0 ; b<=255 ; b++) {
                  cout << dec2bin(a).substr(8)
                        << " " << dec2bin(b).substr(8) << " "
                        << dec2bin(a*b) << endl;
            }
      }
}



//
// Function returns 16-byte long STL string of 1's and 0's which
// represent the argument dec in binary.
//
// dec2bin(0xff) returns 1111111111111111.
//


string dec2bin(int dec) {
      string bin="0000000000000000";


      for(int x=15;x>=0;x--) {

            if(dec-int(pow(2,x)) >= 0) {
                  dec-=int(pow(2,x));
                  bin[15-x]='1';
            }
      }

      return bin;
}
```

```
#!/usr/bin/perl

#
# Creates IRSIM .cmd-formatted file to test all possible 8-bit binary
products.
#
# Charles Hastings, 2001
#

open(FILE, "binary.out");

while( $temp=<FILE> ) {
      $temp=~s/\n//g;

      my ($a, $b, $result)=split(" ", $temp);

      print "set a $a\n";
      print "set b $b\n";
      print "s\n";
      print "d y\n";

}


#!/usr/bin/perl

#
# Compares IRSIM output file to known list of products.  Reports
anomalies.
#
# Charles Hastings, 2001
#


open(REF, "binary.out");
open(TEST, "test_results.out");

while( $test=<TEST> ) {
      $test=~s/\n//g;

      if( substr($test, 0, 2) eq "y=" ) {
            $ref=<REF>;
            $ref=~s/\n//g;

            my ($a, $b, $prod)=split(" ", $ref);

            $prodTest=substr($test, 2, 16);

            if( $prod ne $prodTest ) {
                  print "Error!  $a * $b != $prodTest (should be
$prod)\n";
            }

      }

}
```