Final project
ECE 471

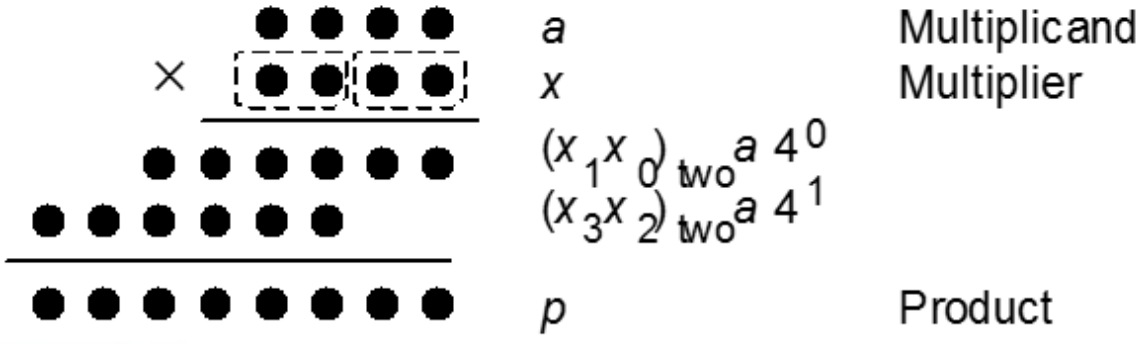
4 bit multiplier

Author: Chauncey Yan

Project Descriptions

1. Design structure: Booth Multiplier

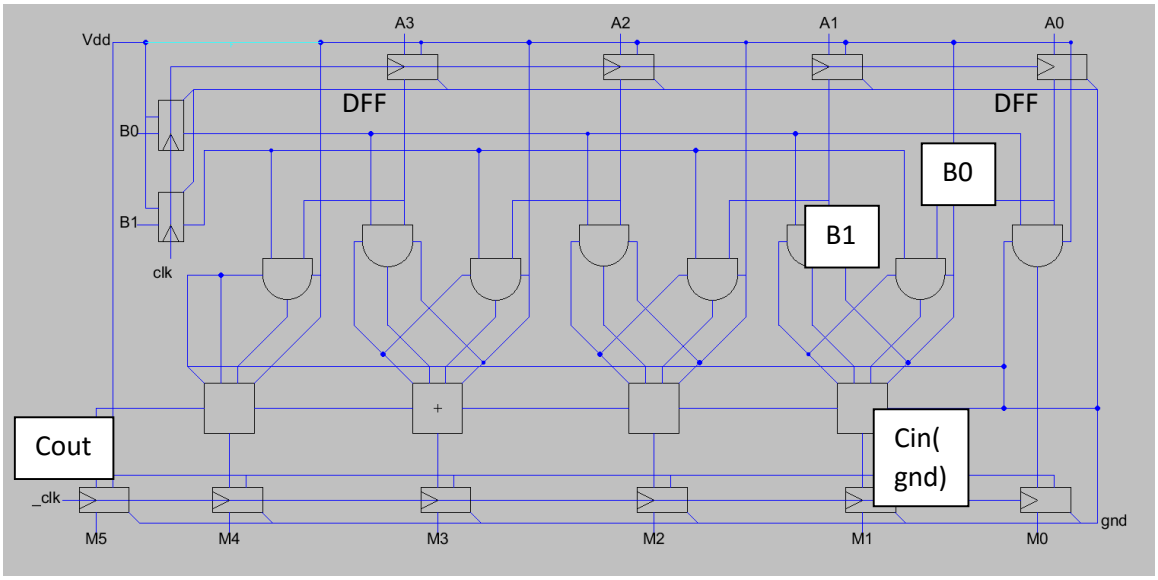
4 bit Booth Multiplier in dot notation



First stage:
Preform two 2-bit X 4-bit multiplications concurrently.
Second Stage:
Sum the products of the first stage.

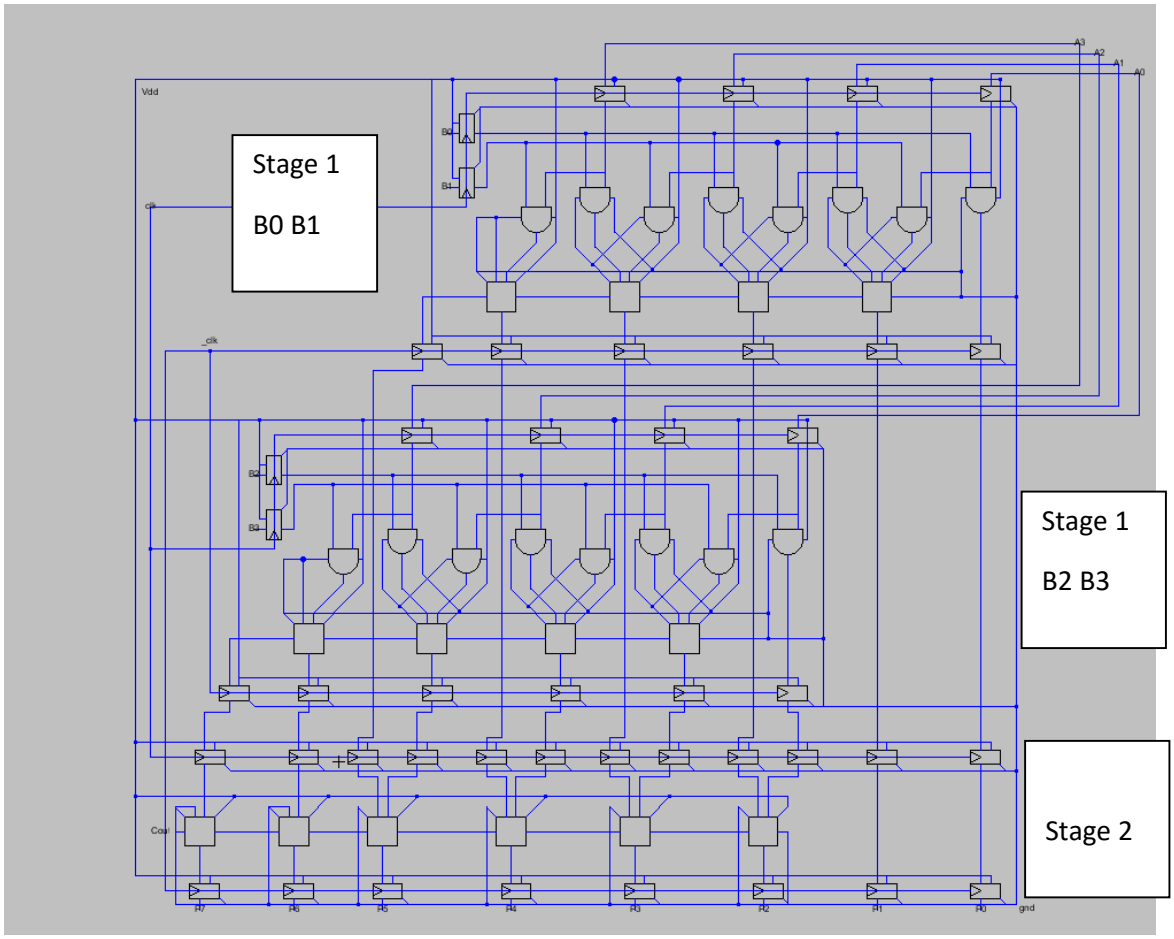
My design to do 2-bit X 4-bit

If x0 is 1 then pass a as first input to add.
If x0 is 0 then pass 0 as first input to add.
If x1 is 1 then pass a as second input to add.
If x1 is 0 then pass 0 as second input to add.
This If statement is easily to construct with an AND2 gate.
Then add the result of the 2 AND2 gate with a fulladder.



I added a row of d type of flip flop at the input and another row of d type flip flop at the output. Input DFF controlled by clk, output DFF controlled by clkbar.

After we have 2 set of M0-M5, we need to shift the second set of M0-M5 to the left by two bits. Then add these two numbers we get the 8bit product.



Advantages:
Combine the calculations to reduce number of stages in multiplication.
Performs two bits of multiplication at once—requires half of the stages.

Disadvantages:
Each stage is slightly more complex than simple multiplier.

Alterations/simplifications you made and why you made them.
I firstly use the array multiplier configuration, and put one single clk controlled flipflop in between each stage. However, in order to pipeline the inputs I need 4 clocks to controlling the operations. So I change the design to the above.

2. Design decisions:

Beta = 2 as the common value for this simulation.

Transistor size as 80nm for the gate, 405nm for the invertor NMOS, 810nm for the invertor PMOS. **Total NAND gate size** is

$$18 * 45nm + 18 * 45\ nm = 1620\ nm.$$

I had this size similar to my midterm selection of transistor size. Since we are not aiming at better performance in this course, a big size is ok choice for learning basics and layout. Made the transistor slightly smaller to make my layout easier.

3. Test sources selections for multiplication:

To able to simulate the worst case scenario, the max 4-bit multipliers as the second set of number was chosen. First set multipliers have the product as the complementary of the second set product.

First set multipliers a = 0101 b = 0110

Then the operands transition to a = 1111 and b = 1111

Back to a = 0101 b = 0110

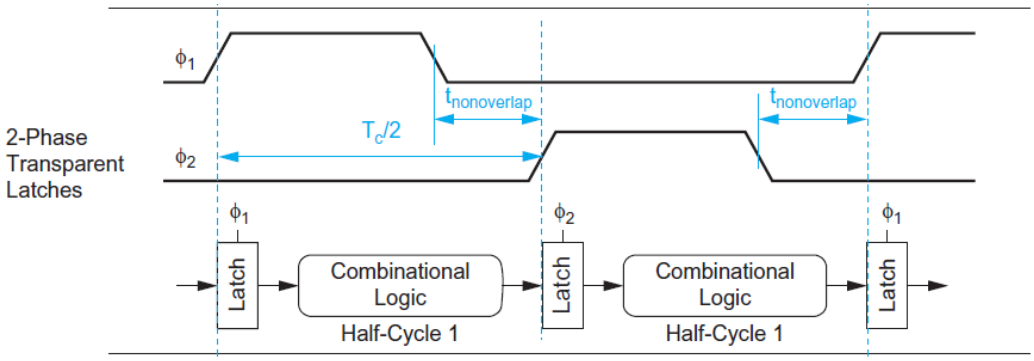
Then looping continues.

There are 2 reasons behind why I use those test numbers. First, in order to find the longest delay path, it needs to be carry in and out on most of the full adder before presenting the final result. 4 bit maximum number 1111 multiply 1111 is a good testing selection. Second, for showing the changes of the result, it need to be flipped every time inputs change.

4. Maximus clock rate to achieve the correct result

I used this clk(φ 1) controled D-type flip-flop instead of latches 1 in the following graph.
I used this clkbar(φ 2) controlled D-type flip-flop instead of latches 2 in the following graph.

Clock cycle implementation:



My clock source split into two clocks, clk and _clk. Clk in charge of turning on first and third D-FF, _clk in charge of turning on second D-FF. For my design, t nonoverlap ~= 0. In this design, one set of data can be load and calculated between 1 full period of clock cycle and be presented at the third D-FF when the second clock cycle kicks in. The minimum interval for each set of data will be half cycle of clk period. We can maximum the performance by reducing the clock period to have _clk rising edge right after when cout, the longest logic path in adder, is calculated.

5. Monte Carlo transistor mismatch

Procedures:

Find the 50ps increment where your circuit goes from no failures to one or more failures.

My minimum clkperiod without failure is 550ps.

Record the number of failures as you push 50ps, 100ps, and 150ps below the no failures clock period.

550ps score a 100% pass

499ps score a 20% pass

450 has no pass

Write some measure statement to help you to detect how frequently the multiplier fails.

```
.meas product5_3to35 avg v(p5) from='3*clkperiod' to='3.5*clkperiod'
.meas product5_4to45 avg v(p5) from='4*clkperiod' to='4.5*clkperiod'
```

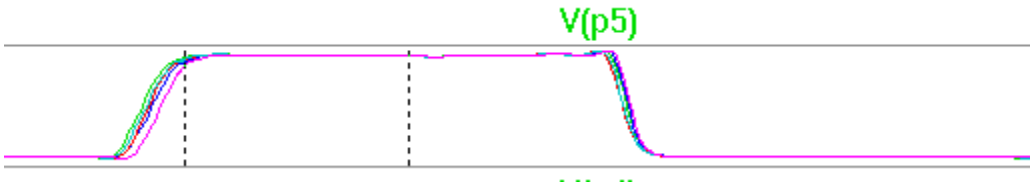
The reason I use p5 to do the measurements is because it is longest path in this booth multiplier.

To be honest, I expected p6 as the longest path and the result show off last. However, during the test, P5 is always the first one to break. I think it could have something to do with my test number 1111 X 1111.

Results:

550ps is all of the test pass.

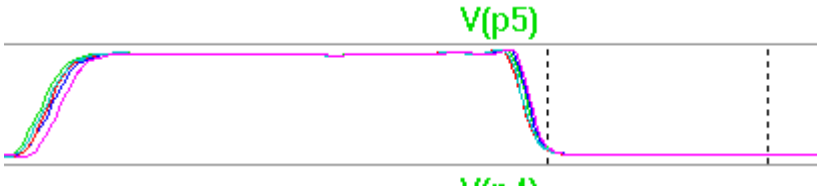
Logic 1 state




```
Measurement: product5 3clk to 3.5clk
step   AVG(v(p5))   FROM   TO
1      0.960824     1.65e-009    1.925e-009
2      0.931669     1.65e-009    1.925e-009
3      0.94258      1.65e-009    1.925e-009
4      0.950476     1.65e-009    1.925e-009
5      0.899354     1.65e-009    1.925e-009

1-5 run averages are all close to 1v.
```

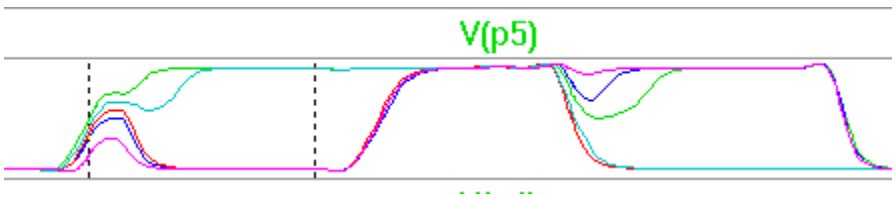
Logic 0 state




```
Measurement: product5 4clk to 4.5clk
step   AVG(v(p5))   FROM   TO
1      0.125645     2.2e-009     2.475e-009
2      0.132381     2.2e-009     2.475e-009
3      0.107574     2.2e-009     2.475e-009
4      0.11073      2.2e-009     2.475e-009
5      0.140736     2.2e-009     2.475e-009
```

499ps as 550ps reduced by 50ps. (500ps run too slow)

Logic 1 state

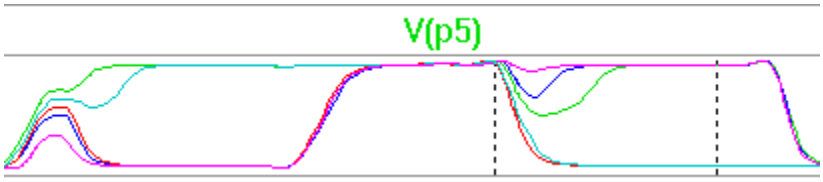



```
Measurement: product5 3clk to 3.5clk
step   AVG(v(p5))   FROM   TO
1      0.745357     1.497e-009   1.7465e-009
```

```
2      0.112802    1.497e-009  1.7465e-009
3      0.136227    1.497e-009  1.7465e-009
4      0.646549    1.497e-009  1.7465e-009
5      0.0521894   1.497e-009  1.7465e-009
```

1 and 4 are close to 1 but the other are close to 0.

Logic 0 state



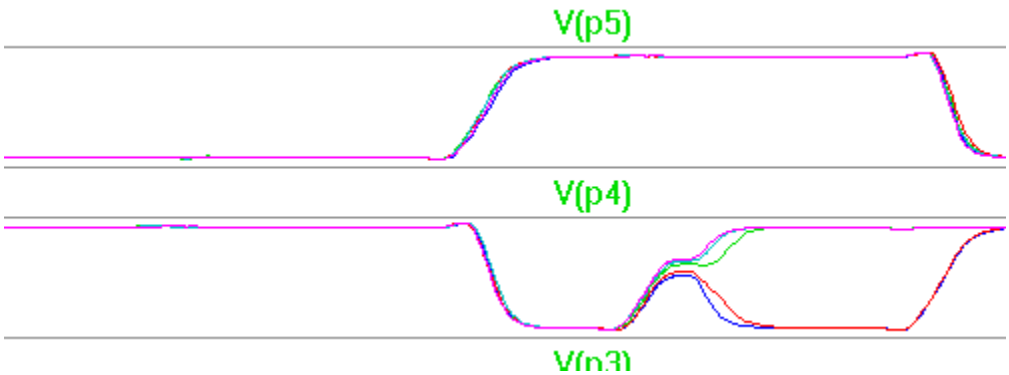

```
Measurement: product5 4clk to 4.5clk
step   AVG(v(p5))   FROM   TO
1      0.84899     1.996e-009  2.2455e-009
2      0.952593    1.996e-009  2.2455e-009
3      0.328637    1.996e-009  2.2455e-009
4      0.344647    1.996e-009  2.2455e-009
5      0.996322    1.996e-009  2.2455e-009
```

3 and 4 are close to 0 but the other are close to 1.

Only 4 are close to the correct logic.

Error rate is 80%. Under 499ps

450ps as 550ps reduced by 100ps.



P5 should be 10 but it is 01

P4 should be 01 but it is 1*

Complete incorrect logic. 100% failures.

6. Source citations

Behrooz Parhami's Computer Arithmetic Textbook -- Multiplications

Pennsylvania State University CMPEN 411 VLSI Digital Circuits Spring 2012