Multiplication

Introduction

There are tons of papers written on multiplication. Unfortunately it is rare that the paper talks about the logic and circuit issues together. There are many ‘logic’ papers with describe clever ways of adding the partial products, but they rarely address the cost of the wiring. Other circuit papers talk about fast ways to build the basic adder cells. We will start with the multiplier organization, and look at both the logic and circuit issues.

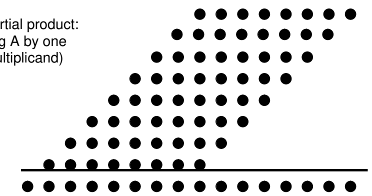
Multiplication

Basic idea is pretty simple

- A * B = C

Done by generating a number of partial products, and adding them together:

Row of dots is a partial product: Result of multiplying A by one bit of B (A is the multiplicand)

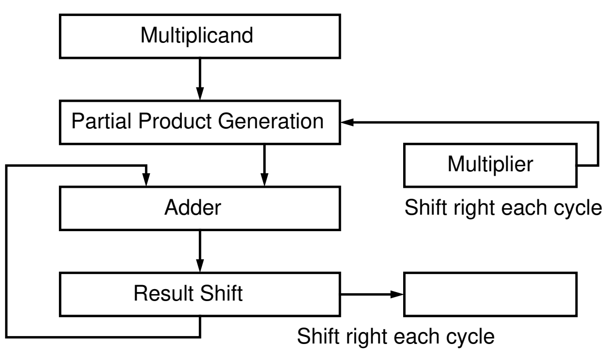


- Result is 2n bits
- For integer operations want LSB n bits, for FP want MSB

  FP 1.XXXX * 1.XXX

Simple Multiplier

Takes n cycles, and generates and adds one partial product each cycles:



Main Issues

1. Create the partial products
2. Sum them together

For a fast multiplier

- Reduce the number of partial products
- Fast Adders
- Tree adders

Talk about this stuff next

Partial Product Generation

In the simple form it is very easy:

0 * A = 0 1 * A = A

- This is a simple AND gate at each bit
- But generated a PP for each multiplier bit

We can generate a PP for every two multiplier bits

0 0 0

0 1 1*

1 0 2* (this is easy to generate by a shift)

1 1 3* (yech -- but can make easy 4 -1, and delay the 4 into the next group

Modified Booth Coding

Make it easy

- When the MSB of the 2bits is 1 add 4 to next group up (2 = 4 -2)
- Booth recoder looks at 3 bits

  The two bits it is recoding, and the MSB of previous group

  To know when it needs to add one for the previous group

Table 1:

MSB	LSB	Prev	Out
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

3 Bit Booth

Can recode 3 multiplier bits at a time

- Generates 1/3 of the partial products
- But you end up with needing 3*Multiplicand

  This takes an adder

  So this is generally not done

Table 2:

MSB		LSB	Out
0	0	0	0x
0	0	1	1x
0	1	0	2x
0	1	1	3x
1	0	0	-4x
1	0	1	-3x
1	1	0	-2x
1	1	1	-1x

Need to look at MSB of previous group and add 1 if it is 1. This still keeps the numbers between -4 and 4. The only hard one to generate is 3x

Adding Partial Products

If you want to add more partial products in a cycle you could use multiple adders

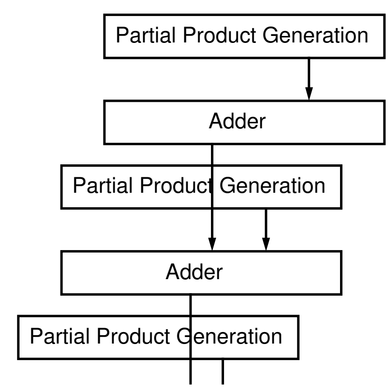
This solution has two problems:

- Even fast adders are pretty slow

  7+ FO4 delays

- Fast adders are pretty large

  Expensive in area



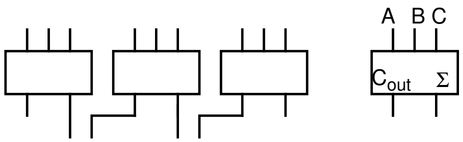
Carry Save Adder

To get a faster adder we will use a trick:

- Represent the result of adder (sum) in a redundant form

  $Sum = Sum_1 + Sum_2$

  That is when you add Sum_1 and Sum_2 together you get the right result. This is nice since we now don't need to propagate the carry in the adder.



Instead of rippling the carry, shift it left by one, and use it as the 'other' part of the sum.

Carry Save Adder

Each adder can add one partial product to a redundant sum

- Redundant sum takes two inputs, partial product takes the third

Each adder is small and fast

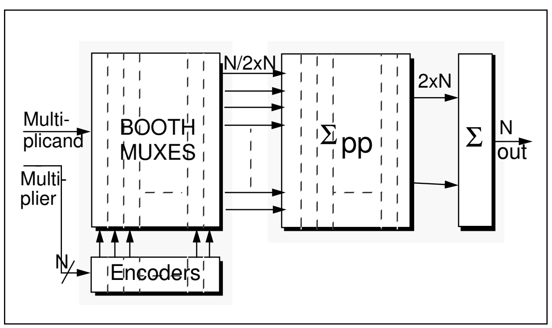
- Domino CSA is around 1.5 FO
- CSA is pretty simple, it is just a full adder

The problem is the output is really in a usable form

- At the end of the array you need to add two parts of redundant number together
- This take a fast adder, but you only need one at the end of multiplier, not one for partial product

Multiplier Overview

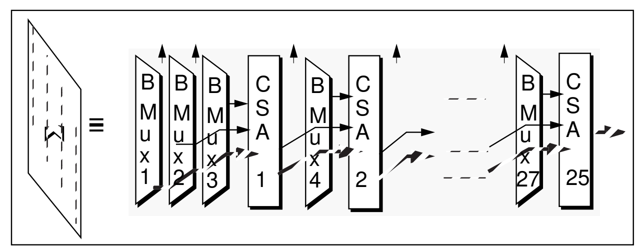
Block diagram of multiplier:



The Σ array is in carry save adders, and final sum is a normal adder

Array Multiplier

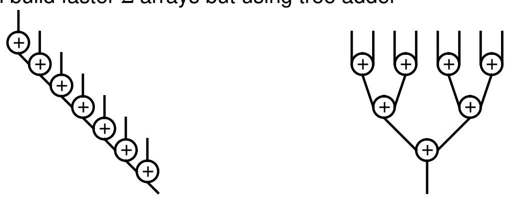
For floating point numbers you have 53 bit numbers, which give 27PP



- 25 CSAs in datapath, all in critical path (First adder adds 3 PP)
- 56 bits tall regular datapath
- Short wires

Tree Multipliers

Can build faster Σ arrays but using tree adder



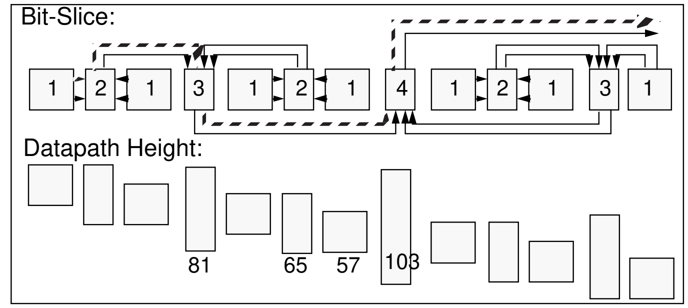
Need to be careful since:

- Trees have longer wires (each adder is a row of adders
- Drawn a binary tree, but CSA are 3 input 2 output devices

  Can build a 4:2 combiner using 2 CSA

Tree Layout

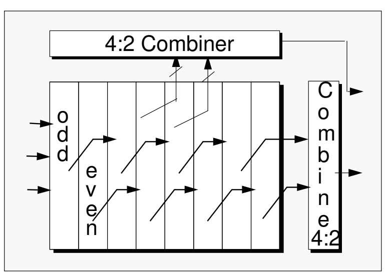
Full (4:2) Trees:



- 26 CSAs in datapath, 8 in critical path
- Increased datapath height
- Long wires, many cross-overs, wiring channels

Interleaved Arrays

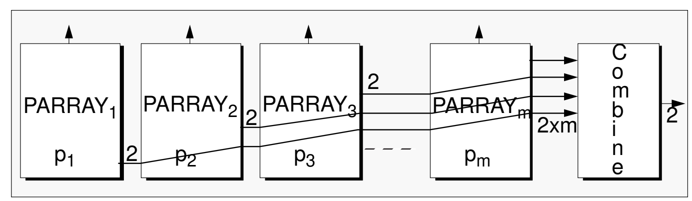
Compromise solution



- Full-array critical path reduced by half for small area overhead
- Longer wires within the array
- Can we do even better?

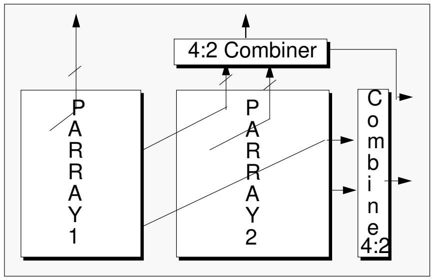
Array-of-Arrays

General Idea:



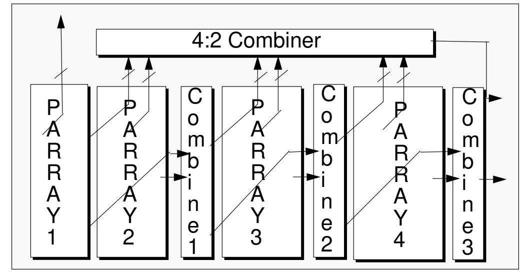
- More partial arrays: higher parallelism, wiring complexity, larger combining network
- Chose M depending on problem size

Two Partial Arrays



- Regular wiring, small number of cross-overs
- Maximum effective wire loading ~1.5xCSA delays
- For optimal parallelism, p1=13 + p2=14.
  ~14.5 CSA delays in critical path

Four Partial Arrays



- Tree combiner: Wiring, area overhead
- Matching Array Delays: 5+5+7+10 pp.10 CSAs
- Optimal Parallelism: 4+4+8+11 pp. 11 CSAs
- More Partial arrays?: marginal returns

Details

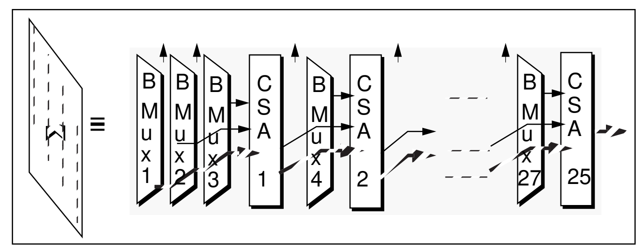
There are two details that we still need to cover

- Dealing with the bits that fall off the end of the array
  - In the array organization the datapath width remain constant
  - The 'extra' (LSB) bits fall off the top of the array
  - But these bits are still in redundant form
  - For rounding these bits need to be added too
- Dealing with negative partial products
  - The problem is with sign extension
  - Need to extend the sign bits the full 2N bits
    - This is N additional bits for the first PP

Look at these issues next

LSBs Falling Off the Array

Speed through the array is one CSA delay



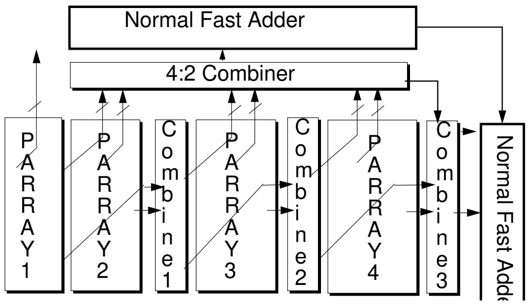
- But since the PP are booth coded, two bit positions fall off each stage
- Need to do a two bit carry propagate add in on CSA delay
  - Build a small carry select adder
  - Carry in to carry out is only a mux delay

Faster Structures

Do more adds in parallel, making dealing with the end bits harder

- In tree adder, all the bits fall off the end at roughly the same time
- Array of arrays has a similar problem

Need to build a complete carry propagate adder for these bits



Sign Extension

When the PP is negative, need to sign extend

- This is bad since the fanout can be large

Use a trick instead

- Pre-add the triangle of 1's

```
1 1 1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1
1 1
1 0 1 0 1 0 1 1
```

- To clear out 1's add 1 to the row

1 1 1 1 1 1 1 1
$\overline{S}$

Sign Extension

So you only need to add few bits:

$\overline{S}$ S S
1 S
1 $\overline{S}$
1 $\overline{S}$
1 0 1 0 1 0 1 1

Adding these bits together is the same as complete sign extension