TSTE18 Digital Arithmetic
Seminar 4

Oscar Gustafsson

Multiplication

- **Partial product generation**
- **Partial product accumulation**
- **Final adder**

- Squarers
- Constant multiplication

Multiplication

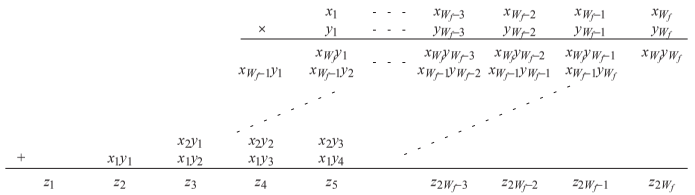
- Multiplication can typically be separated into three sub-problems
  - Generating partial products
  - Adding the partial products using a redundant number representation
  - Converting the redundant number representation to non-redundant form (commonly using a vector merging adder, VMA)

Partial product generation

- Multiplying two unsigned binary numbers X and Y can be written as

$$Z = XY = \sum_{i=-L}^{M-1} x_i 2^i \sum_{j=-L}^{M} y_j 2^j = \sum_{i=-L}^{M} \sum_{i=-L}^{M} x_i y_j 2^{i+j}$$

- The resulting partial product array is



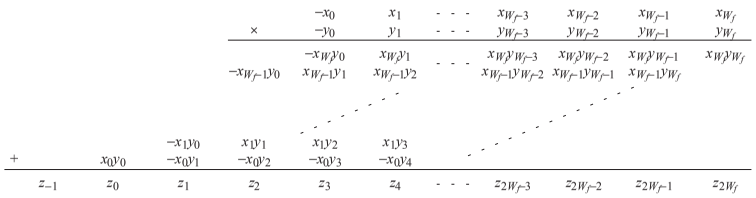
Partial product generation

- For two's complement we can, by using that the sign-bit should be subtracted, write

$$\begin{aligned} Z &= XY \\ &= (-x_{M-1}2^{M-1} + \sum_{i=-L}^{M-2} x_i 2^i)(-y_{M-1}2^{M-1} + \sum_{j=-L}^{M-2} y_j 2^j) \\ &= x_{M-1}y_{M-1}2^{2(M-1)} - x_{M-1}2^{M-1}\sum_{j=-L}^{M-2} y_j 2^j - \\ & \quad y_{M-1}2^{M-1}\sum_{i=-L}^{M-2} x_i 2^i + \sum_{i=-L}^{M-1}\sum_{j=-L}^{M-1} x_i y_j 2^{i+j} \end{aligned}$$

Partial product generation

- The resulting partial product array is



Partial product generation

- To add the partial products we must make all rows have the same length (or add them sequentially)
  - Zero extend at LSB side
  - Sign extend at MSB side
- Adding zeros can be ignored, but additional circuits are required to deal with the sign-bits
- Instead, use the identity $-p = \bar{p} - 1$ (modified Baugh-Wooley)

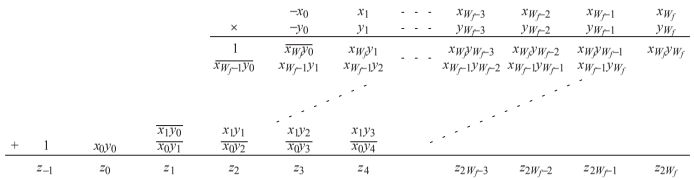
Partial product generation

- Replace all negative partial products by their inverse

$$\begin{aligned} Z &= XY \\ &= x_{M-1}y_{M-1}2^{2(M-1)} + \sum_{j=-L}^{M-2}(\overline{x_{M-1}y_j} - 1)2^{j+M-1} + \sum_{i=-L}^{M-2}(\overline{y_{M-1}x_i} - 1)2^{i+M-1} \\ &\quad + \sum_{i=-L}^{M-2}\sum_{j=-L}^{M-2} x_i y_j 2^{i+j} \\ &= x_{M-1}y_{M-1}2^{2(M-1)} + \sum_{j=-L}^{M-2}\overline{x_{M-1}y_j}2^{j+M-1} + \sum_{i=-L}^{M-2}\overline{y_{M-1}x_i}2^{i+M-1} + \\ &\quad \sum_{i=-L}^{M-2}\sum_{j=-L}^{M-2} x_i y_j 2^{-i-j} - 2\left(2^{2(M-1)} - 2^{M-L-1}\right) \end{aligned}$$

Partial product generation

- Possible to add partial products with positive weights and then add the compensation $2\left(2^{2(M-1)} - 2^{M-L-1}\right) = 1000\ldots00100\ldots$
- The resulting partial product array is



Signed-digit representations

- We only need to add non-zero partial products
- Representation with more zeros?
- The binary signed-digit representation, i.e., $r = 2$, $x_i \in \{-1, 0, 1\}$ is a redundant number representation
  - $7 = 0111 = 100\bar{1}$, where $\bar{1} = -1$
  - $115 = 01110011 = 100\bar{1}010\bar{1} = 1000\bar{1}\bar{1}0\bar{1} = \ldots$
- An SD representation with minimum number of non-zero digits is called a minimum SD (MSD) representation
- In particular, if $x_i x_{i+1} = 0$, i.e., no two adjacent digits are both non-zero, the representation is minimum
- This particular MSD is called canonic (=unique, i.e., no longer redundant) SD (CSD) representation
- For CSD the average number of non-zeros is $\frac{M+L}{3}$ instead of $\frac{M+L}{2}$ for binary
- More importantly, the maximum number of non-zeros is $\frac{M+L}{2}$ instead of $M + L$ for binary

Partial product generation

- A standard multiplier would typically have $M + L$ rows to generate and add
- If we can change to a signed-digit representation we know that there can be quite a bit of zeros
- However, the CSD representation require a conversion with "carry-propagation"
- Another encoding, Modified Booth encoding, have a fully parallel conversion
- The modified Booth encoding has at most $\frac{M+L}{2}$ non-zeros, but on average it has more digits compared to minimum signed-digit representations like CSD

Partial product generation

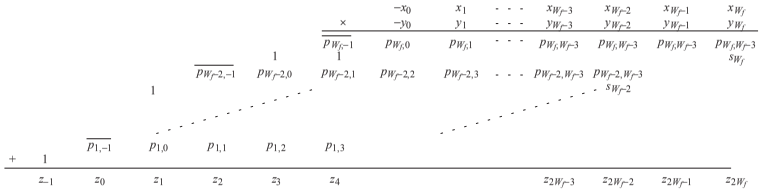
- The modified Booth encoding can be seen as a radix-4 signed-digit representation with the recoded digits $x_i \in \{-2, -1, 0, 1, 2\}$
- The encoding is based on three bits, but it only replaces two

x_{2k+1}	x_{2k}	x_{2k-1}	r_k	$d_{2k+1}d_{2k}$	Description
0	0	0	0	00	String of zeros
0	0	1	1	01	End of ones
0	1	0	1	01	Single one
0	1	1	2	10	End of ones
1	0	0	-2	$\bar{1}0$	Start of ones
1	0	1	-1	$0\bar{1}$	Start and end of ones
1	1	0	-1	$0\bar{1}$	Start of ones
1	1	1	0	00	String of ones

- Example: recode 371

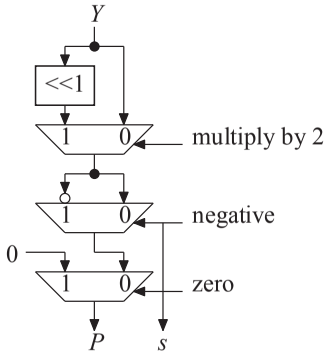
Partial product generation

- Only half the number of partial product rows are now required
- Must be one bit wider to allow shifts
- Also, to possibly subtract a row, a partial product for adding the LSB must be included
- Resulting partial products array



Partial product generation

- One possible realization of the modified radix-4 Booth encoding partial product generation



Partial product generation

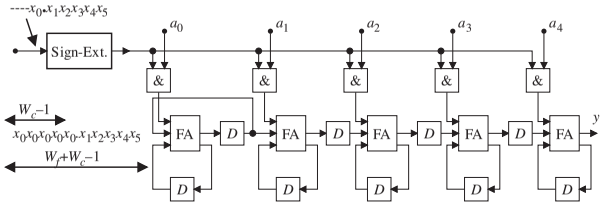
- Possible to define Booth encoding for higher radices
- For radix-8 the digits are $x_i \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$
- Multiplication by 3 require one adder in the pre-computation part
- The number of rows are now $\frac{M+L}{3}$ instead of $\frac{M+L}{2}$ for radix-4
- Example: derive a truth table for Radix-8 Booth encoding

Partial product accumulation

- Three general ways of adding the partial products
  - Sequential – some of the partial products are added in each cycle
  - Array – regular structure
  - Tree – smallest logic depth but irregular structure

Partial product accumulation – Sequential

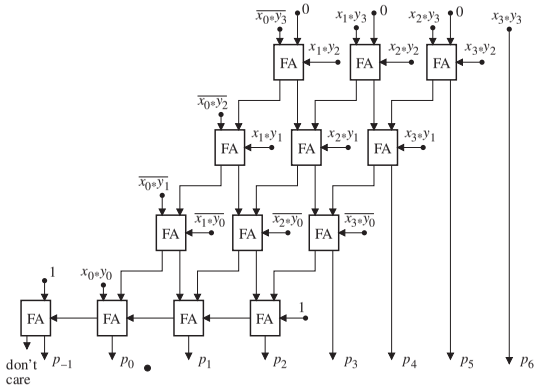
- Add one row or column in each cycle, shift the result and add one more row or column
- Five-bits row-wise sequential accumulator



- Commonly used structure for bit-serial/parallel multipliers

Partial product accumulation – Array

- Use an array of almost identical cells for generation and accumulation of the partial products
- Array multiplier with multiplication time proportional to $2W$

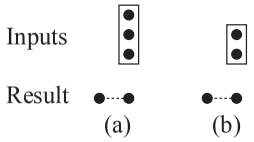


Partial product accumulation – Tree

- The partial product accumulation tree should add a number of bits
- Convenient to represent these using dot diagrams
- Example of partial products from a 6×6 multiplication

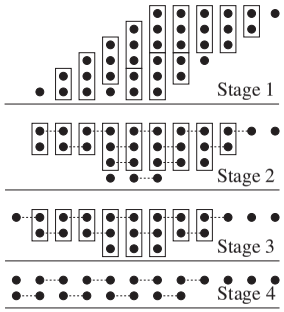
Partial product accumulation – Tree

- Array accumulation gives high regularity, but at the same time the delay increase linearly with the wordlength
- Instead, we can just add all the partial products using simple building blocks such as half and full adders
- A full adder will take three partial products and output two partial products
- A full adder will take two partial products and output two partial products



Partial product accumulation – Tree

- The approach to add all partial products as fast as possible by first adding as many full adders as possible and then as many half adders as possible is known as the Wallace multiplier
- The drawback of the Wallace multiplier is the excessive use of half adders
  - Half adders does not decrease the number of partial products, it only moves them to different columns



Partial product accumulation – Tree

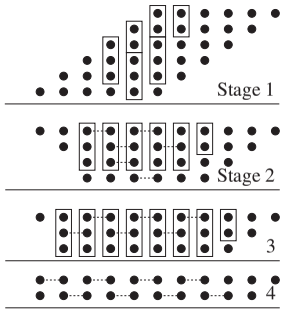
- An alternative approach is to try to balance the height of the columns first to later reduce as much as possible
- This approach is called the Dadda multiplier
- The sequence of wanted number of partial products is the same as the maximum number of partial products that can be reduced using a given number of levels

Levels	Partial products
1	3
2	4
3	6
4	9
5	13
6	19
7	28
8	42

- The drawback is that it hardly use any half adders and, hence, the carry-propagation may become longer

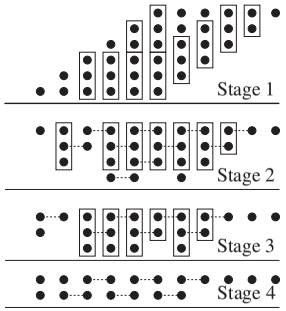
Partial product accumulation – Tree

- Dadda tree



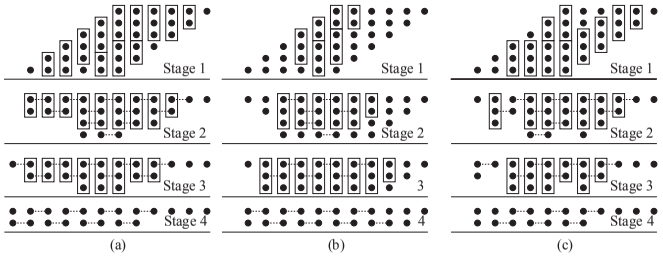
Partial product accumulation – Tree

- The reduced area tree tries to use the best of both algorithms, fastest possible reduction, but considering half adders
- Works as follows
  1. Use as many full adders as possible for each level
  2. Use half adders if (a) will result in a number of bits equal to the Dadda sequence or (b) to reduce the right-most column containing exactly two bits



Partial product accumulation – Tree

- Example reduction steps of the discussed algorithms: (a) Wallace, (b) Dadda, and (c) Reduced area



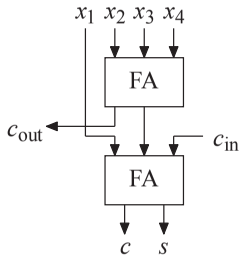
Tree structure	Full adders	Half adders	VMA length
Wallace	16	13	8
Dadda	15	5	10
Reduced area	18	5	7

Counters and compressors

- A full adder counts three bits and produce a result with two bits
- Hence, this can be called a 3:2-counter
- Similarly, a half adder can be called a 2:2-counter
- This is useful when looking at alternative cells to accumulate partial products
- A $n:k$ counter adds n bits and produce a result with k bits
- Clearly $k \geq \log_2(n+1)$

Counters and compressors

- A compressor does not produce a valid binary count of the number of input bits
- Instead, it reduces the number of bits while having several input and output carries
- However, the output carries does not depend on the input carries
- A commonly used compressor is the 4:2 compressor as illustrated below



Counters and compressors

- The 4:2 compressor implemented using full adders may not have any significant advantages
- However, there are other realizations possible
- These should satisfy

$$x_1 + x_2 + x_3 + x_4 + c_{in} = s + 2c + 2c_{out} \quad (1)$$

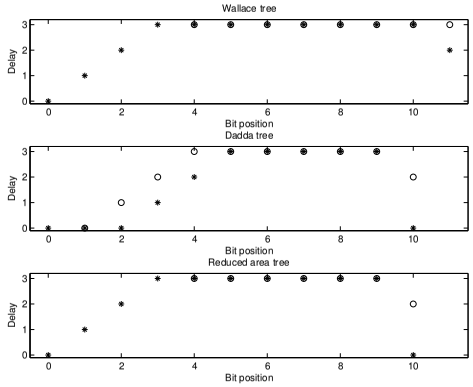
  with c_{out} independent of c_{in}
- Example: determine Boolean equations for a 4:2 compressor

Vector merging adder

- Most partial products accumulation schemes results in a redundant representation
- This must be converted to non-redundant form, typically by addition
- All carry-propagation schemes from earlier seminars can be used

Vector merging adder

- Often, the arrival time of the bits differ
- Shown below are the arrival times for the bits of the earlier accumulation schemes assuming unity delay for full adders



- Can use this information to optimize the vector merging adder

Fixed-width multiplication

- Multiplying two N-bit numbers results in a $2N$-bit number
- Often, only N bits are of interest
- This can be utilized to simplify the multiplication
- Remove less significant partial products and possibly compensate for the error
- Commonly referred to as a fixed-width multiplication
- Example: Design a 4×4-bit unsigned fixed-width multiplier with 5-bit output and static compensation