UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Digital Computer Arithmetic
ECE 666

Part 6c
High-Speed Multiplication - III

Israel Koren

Array Multipliers

♦ The two basic operations - generation and summation of partial products - can be merged, avoiding overhead and speeding up multiplication

♦ Iterative array multipliers (or array multipliers) consist of identical cells, each forming a new partial product and adding it to previously accumulated partial product

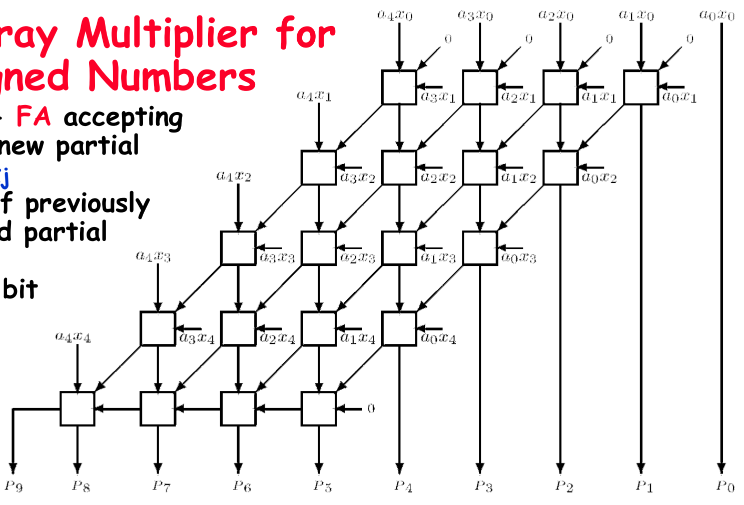
  * Gain in speed obtained at expense of extra hardware
  * Can be implemented so as to support a high rate of pipelining

Illustration - 5 x 5 Multiplication

					a_4	a_3	a_2	a_1	a_0
				$\times$	x_4	x_3	x_2	x_1	x_0
					$a_4 \cdot x_0$	$a_3 \cdot x_0$	$a_2 \cdot x_0$	$a_1 \cdot x_0$	$a_0 \cdot x_0$
				$a_4 \cdot x_1$	$a_3 \cdot x_1$	$a_2 \cdot x_1$	$a_1 \cdot x_1$	$a_0 \cdot x_1$	
			$a_4 \cdot x_2$	$a_3 \cdot x_2$	$a_2 \cdot x_2$	$a_1 \cdot x_2$	$a_0 \cdot x_2$		
		$a_4 \cdot x_3$	$a_3 \cdot x_3$	$a_2 \cdot x_3$	$a_1 \cdot x_3$	$a_0 \cdot x_3$			
	$a_4 \cdot x_4$	$a_3 \cdot x_4$	$a_2 \cdot x_4$	$a_1 \cdot x_4$	$a_0 \cdot x_4$				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

- Straightforward implementation -
  - Add first 2 partial products (a_4x_0, a_3x_0,…,a_0 x_0 and a_4x_1, a_3x_1,…,a_0x_1) in row 1 after proper alignment
  - The results of row 1 are then added to a_4x_2, a_3x_2,…,a_0x_2 in row 2, and so on

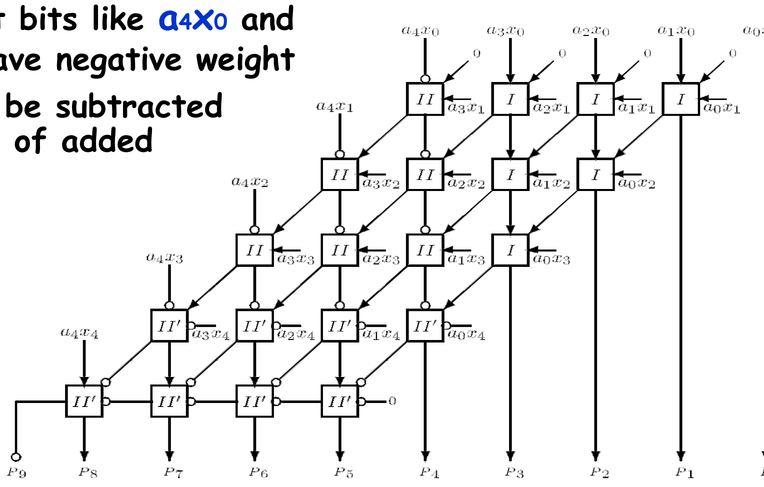
5 x 5 Array Multiplier for Unsigned Numbers



- Basic cell - FA accepting one bit of new partial product a_ix_j + one bit of previously accumulated partial product + carry-in bit
- No horizontal carry propagation in first 4 rows - carry-save type addition - accumulated partial product consists of intermediate sum and carry bits
- Last row is a ripple-carry adder - can be replaced by a fast 2-operand adder (e.g., carry-look-ahead adder)

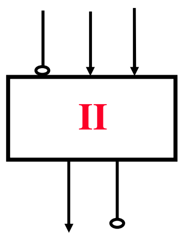
Array Multiplier for Two's Complement Numbers

- ♦ Product bits like a_4x_0 and a_0x_4 have negative weight
- ♦ Should be subtracted instead of added



Type I and II Cells

- ♦ Type I cells: 3 positive inputs - ordinary FAs
- ♦ Type II cells: 1 negative and 2 positive inputs
- ♦ Sum of 3 inputs of type II cell can vary from -1 to 2
  - * c output has weight +2
  - * s output has weight -1
- ♦ Arithmetic operation of type II cell -



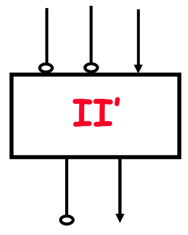
$$x + y - z = 2c - s$$

- ♦ s and c outputs given by

$$s = (x + y - z) \bmod 2 \qquad c = \frac{(x + y - z) + s}{2}$$

Type I' and II' Cells

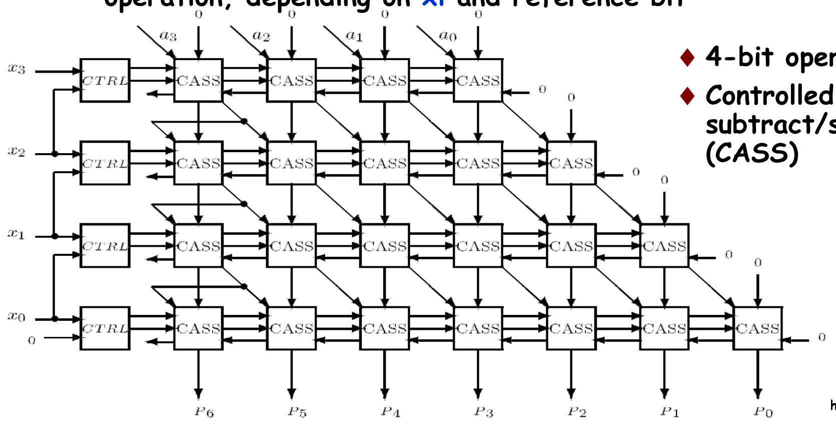
- Type II' cells: 2 negative inputs and 1 positive
- Sum of inputs varies from -2 to 1
  - * c output has weight -2
  - * s output has weight +1



- Type I' cell: all negative inputs - has negatively weighted c and s outputs
- Counts number of -1's at its inputs - represents this number through c and s outputs
- Same logic operation as type I cell - same gate implementation
- Similarly - types II and II' have the same gate implementation

Booth's Algorithm Array Multiplier

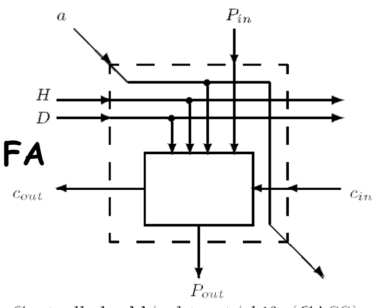
- For two's complement operands
- n rows of basic cells - each row capable of adding or subtracting a properly aligned multiplicand to previously accumulated partial product
  - * Cells in row i perform an add, subtract or transfer-only operation, depending on xi and reference bit



- 4-bit operands
- Controlled add/ subtract/shift (CASS)

Controlled add/subtract/shift - CASS

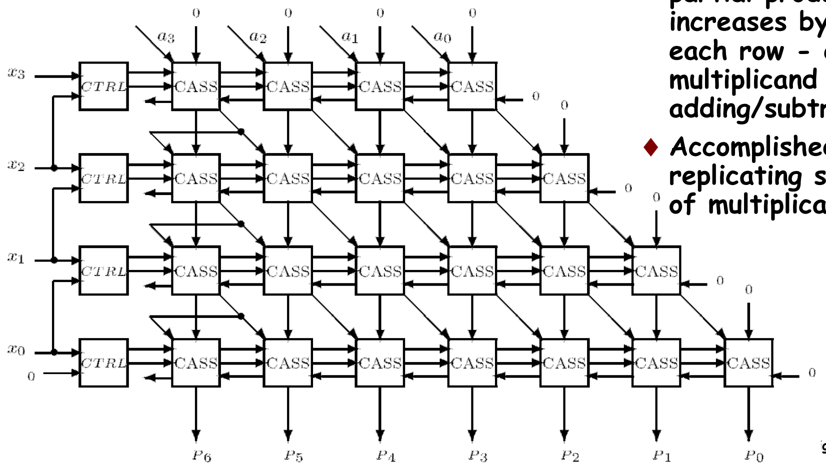
- H and D: control signals indicating type of operation
- H=0: no arithmetic operation done
- H=1: arithmetic operation performed - new Pout
    - Type of arithmetic operation indicated by D signal
    - D=0: multiplicand bit, a, added to Pin with cin as incoming carry - generating Pout and cout as outgoing carry
    - D=1: multiplicand bit, a, subtracted from Pin with incoming borrow and outgoing borrow
- Pout=Pin⊕(a H)⊕(cin H)
  cout=(Pin⊕D)(a+cin) + a cin
- Alternative: combination of multiplexer (0, +a and -a) and FA
- H and D generated by CTRL - based on xi and reference bit x{i-1}



Controlled add/subtract/shift (CASS) cell

Booth's Algorithm Array Multiplier - details

- First row - most significant bit of multiplier
- Resulting partial product need be shifted left before adding/subtracting next multiple of multiplicand
- A new cell with input Pin=0 is added
- Number of bits in partial product increases by one each row - expand multiplicand before adding/subtracting it
- Accomplished by replicating sign bit of multiplicand



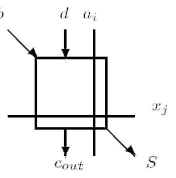
Properties and Delay

- Cannot take advantage of strings of 0's or 1's - cannot eliminate or skip rows
- Only advantage: ability to multiply negative numbers in two's complement with no need for correction
- Operation in row i need not be delayed until all upper (i-1) rows have completed their operation
- P0, generated after one CASS delay (plus delay of CTRL), P1 generated after two CASS delays, and P{2n-2}, generated after (2n-1) CASS delays
- Similarly can implement higher-radix multiplication requiring less rows
- Building block: multiplexer-adder circuit that selects correct multiple of multiplicand A and adds it to previously accumulated partial product

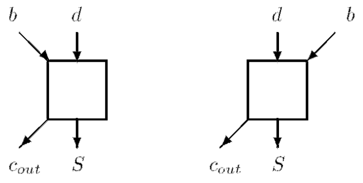
Pipelining

- Important characteristic of array multipliers - allow pipelining
- Execution of separate multiplications overlaps
- The long delay of carry-propagating addition must be minimized
- Achieved by replacing CPA with several additional rows - allow carry propagation of only one position between consecutive rows
- To support pipelining, all cells must include latches - each row handles a separate multiplier-multiplicand pair
- Registers needed to propagate multiplier bits to their destination, and propagate completed product bits

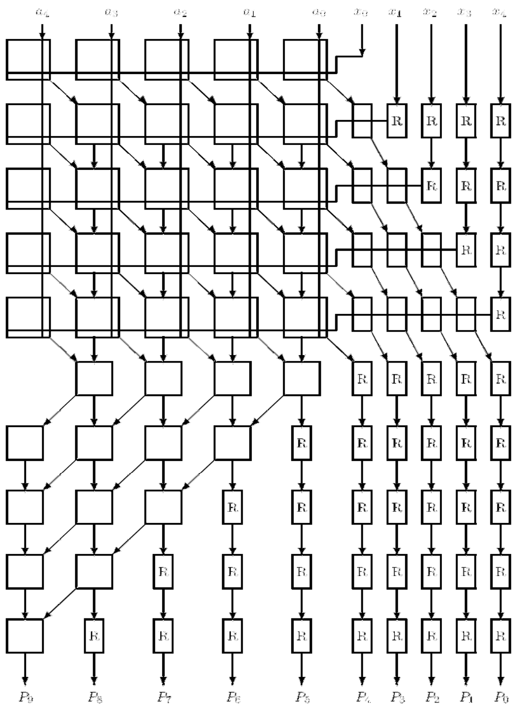
Pipelined Array Multiplier



Latched full adder with an AND gate.



Latched half adders.


a_4
a_3
a_2
a_1
a_0
x_0
x_1
x_2
x_3
x_4
R
P_9
P_8
P_7
P_6
P_5
P_4
P_3
P_2
P_1
P_0