Arithmetic Circuit Implementation

Haibo Wang
ECE Department
Southern Illinois University
Carbondale, IL 62901

Number Representation

- Binary Number

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

- Signed-magnitude number

-3	-2	-1	0	0	1	2	3
111	110	101	100	000	001	010	011

- Two’s complement number

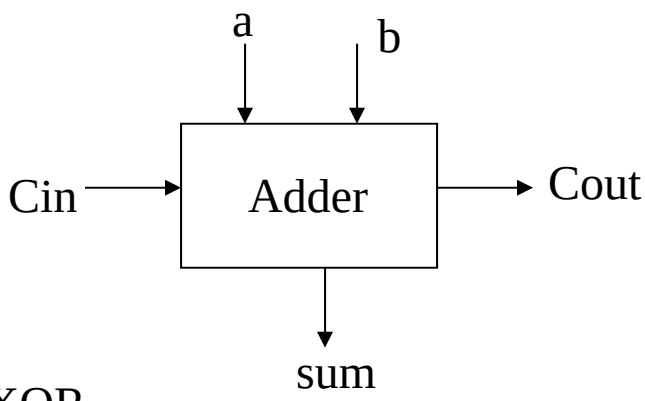
-4	-3	-2	-1	0	1	2	3
100	101	110	111	000	001	010	011

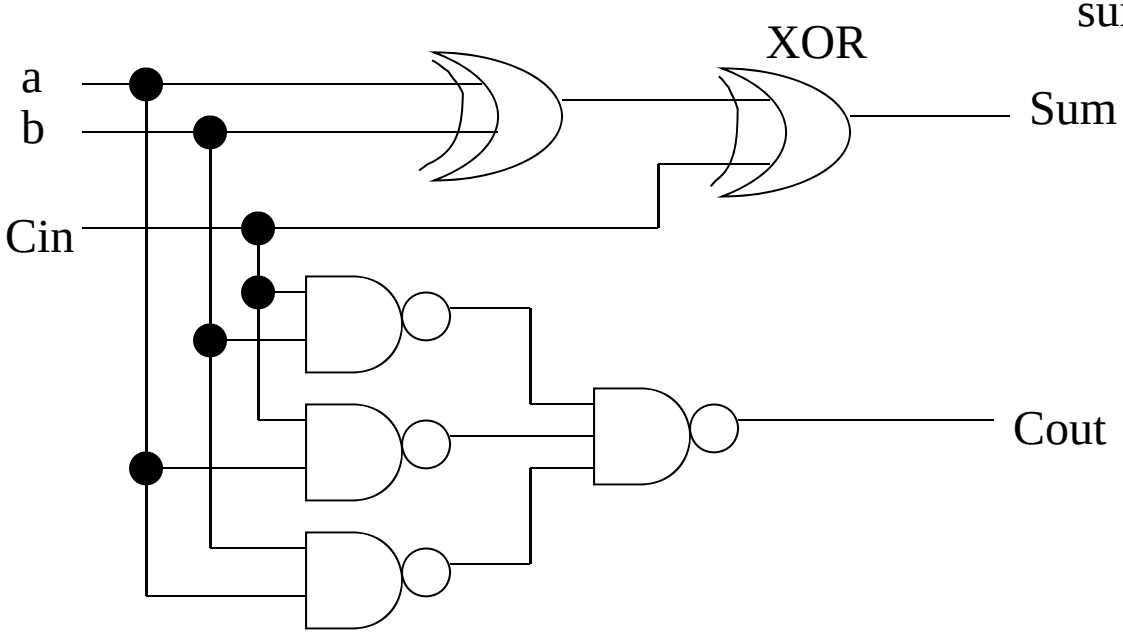
Addition Operation

- Full adder

Sum = a ⊗ b ⊗ Cin

Cout = a•b + a•Cin + b•Cin

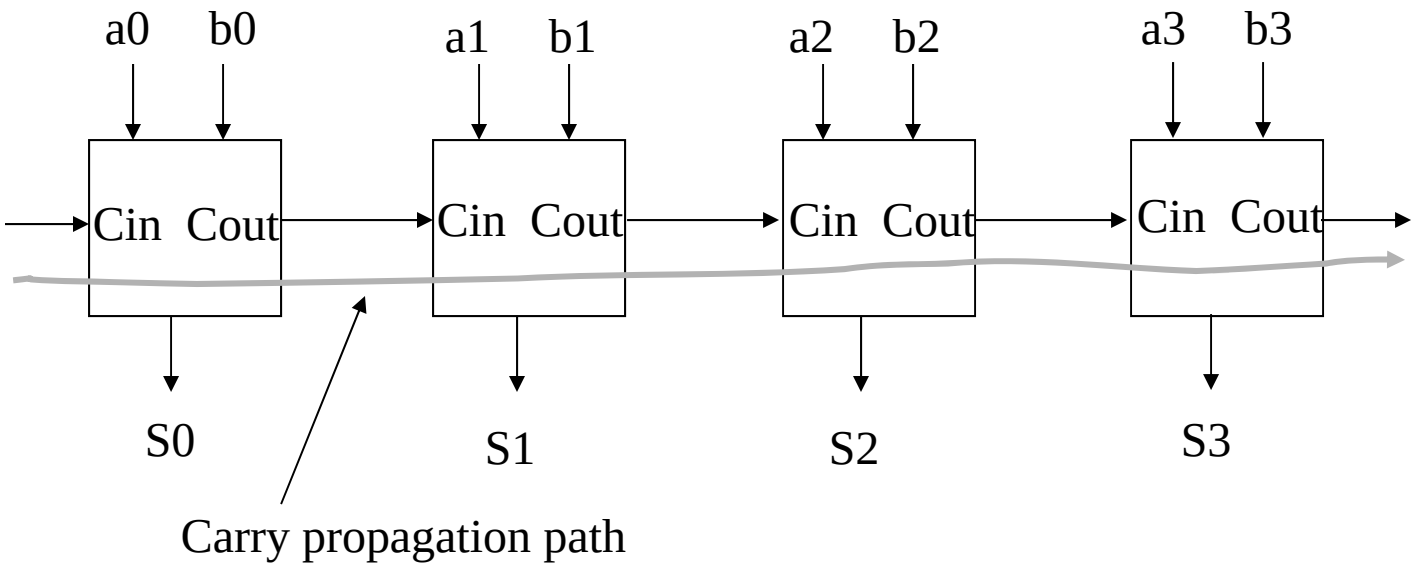




- It works only for binary and 2's complement numbers

Addition Operation

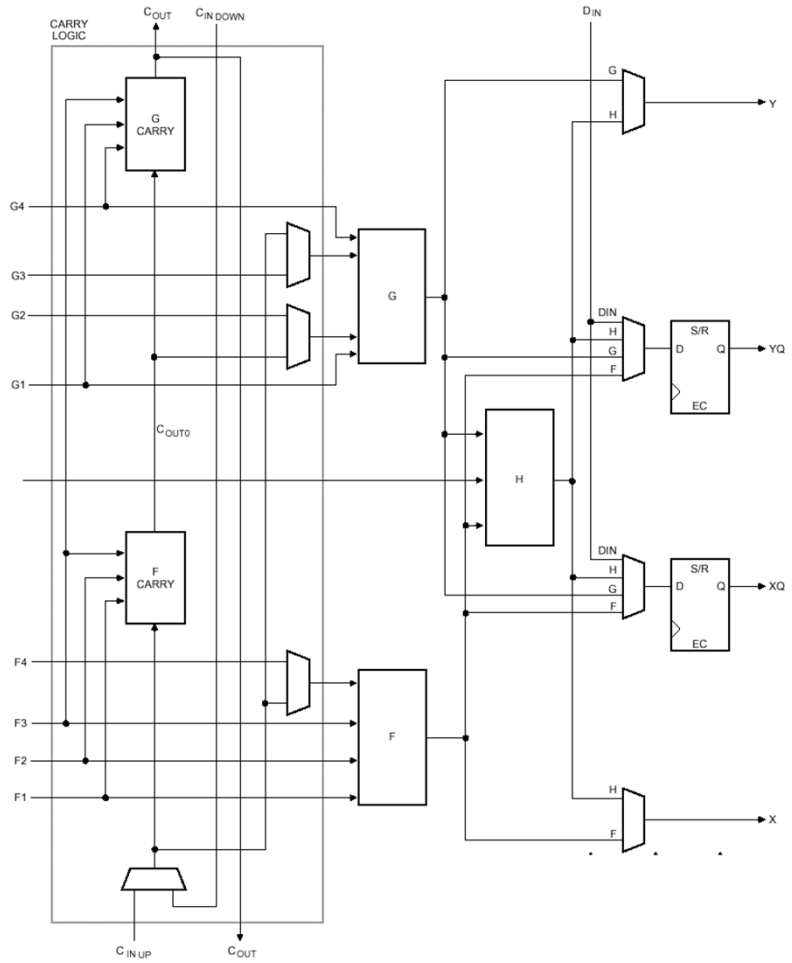
- Four-bit ripple-carry adder



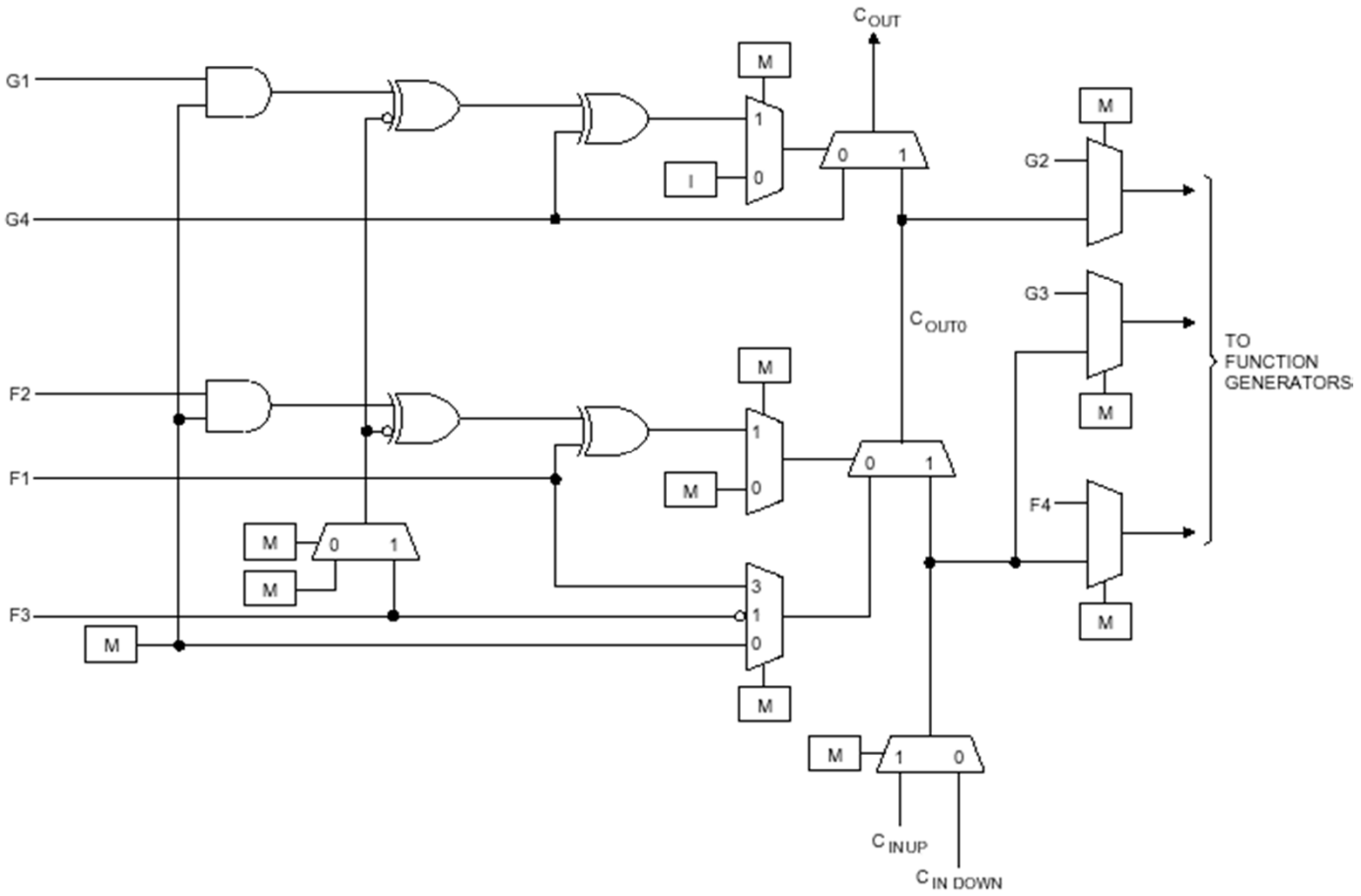
— The large propagation delay on carry propagation path make ripple-carry adder slow.

— Fast adder implementations include carry-bypass adder, carry-select adder and carry look-ahead adder.

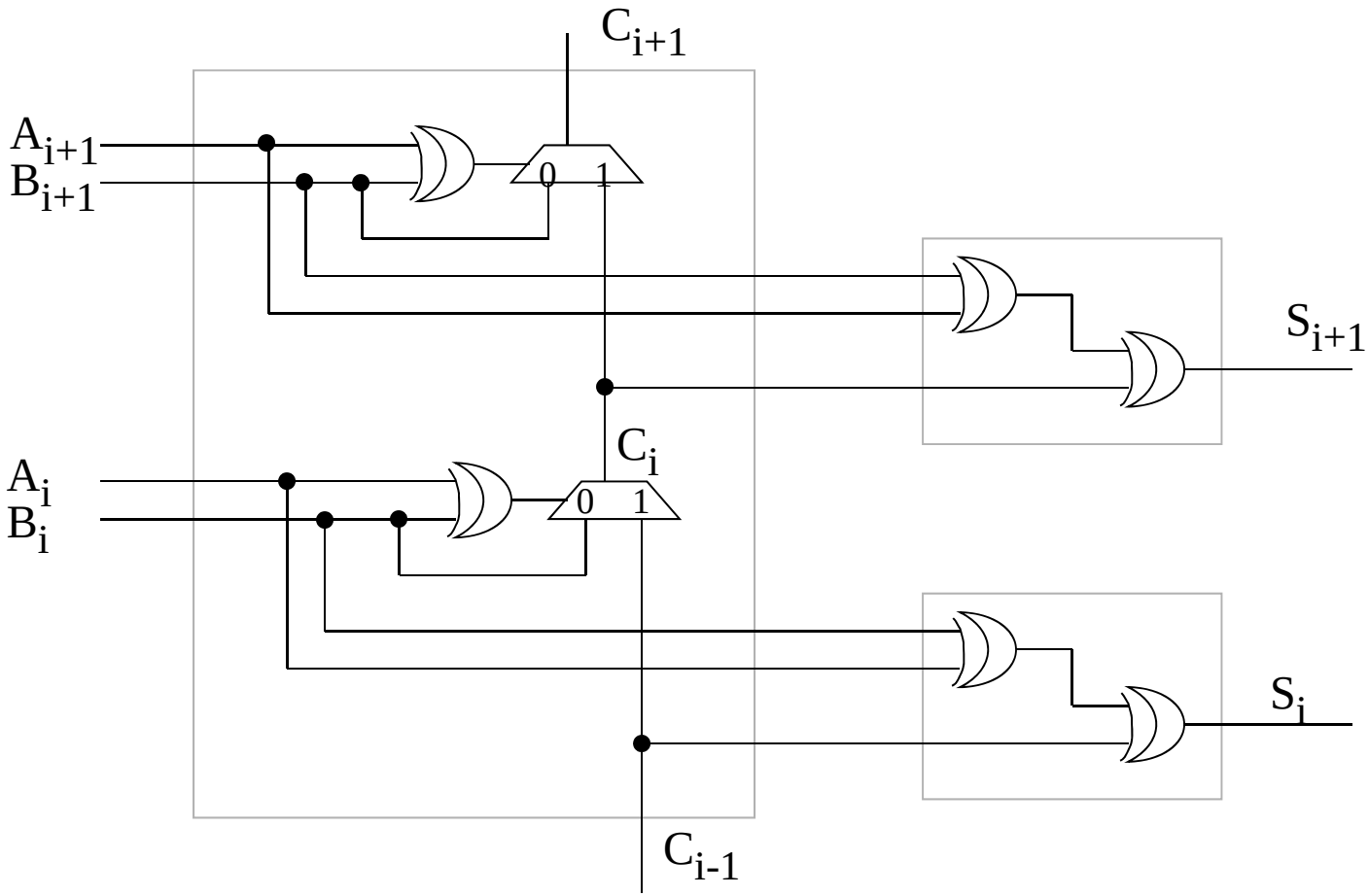
Dedicated Carry Logic in Xilinx XC4000 FPGAs



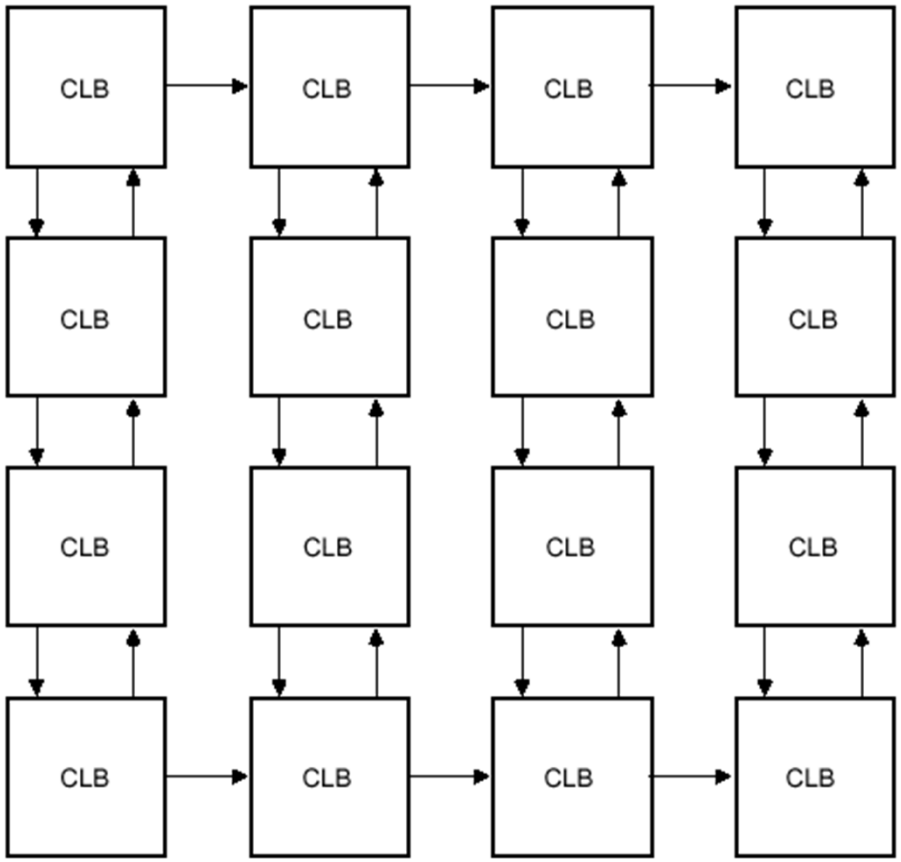
Detailed Carry Logic



A Typical Adder Implementation



Carry Propagation Path

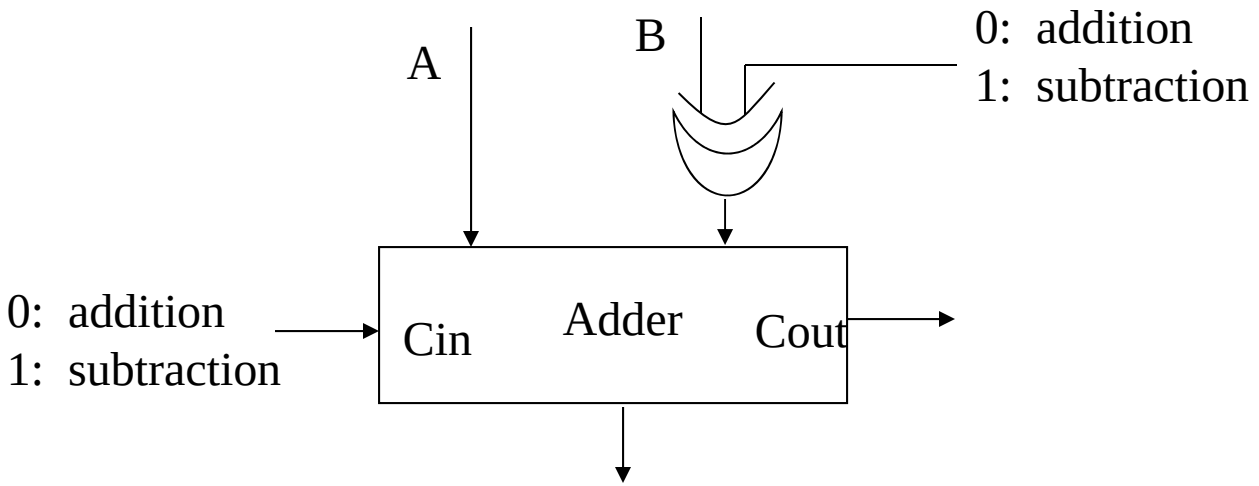


Adder/Subtracter Implementation

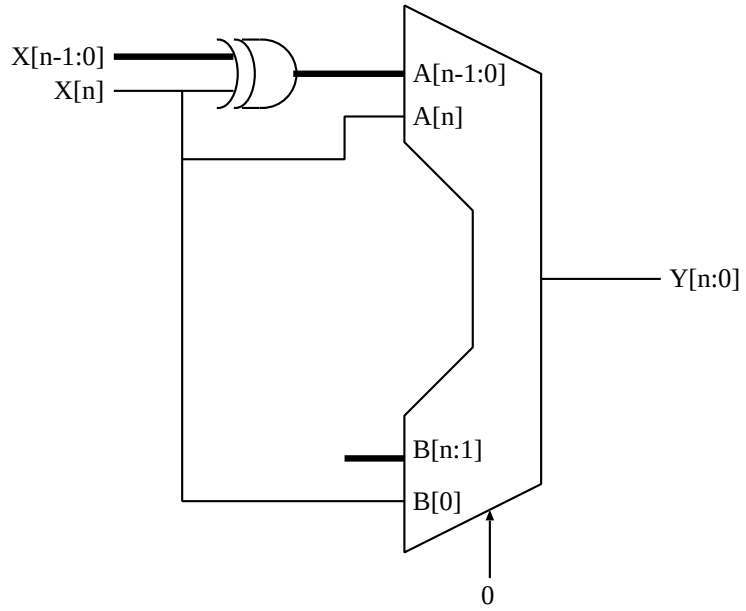
- Subtraction

$$A - B = A + (-B) = A + \overline{B} + 1$$

- Adder/Subtracter implementation



Conversion between 2's complement and signed-magnitude numbers

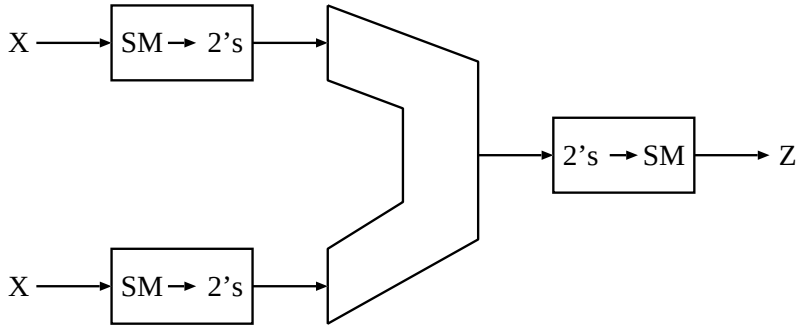


Signed-magnitude	2's complement
2's complement	Signed-magnitude

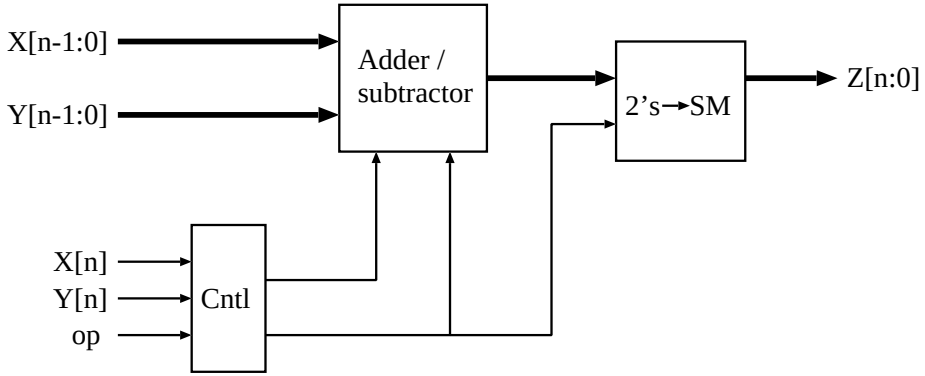
Exception: 100…0

Addition/Subtraction for signed-magnitude numbers

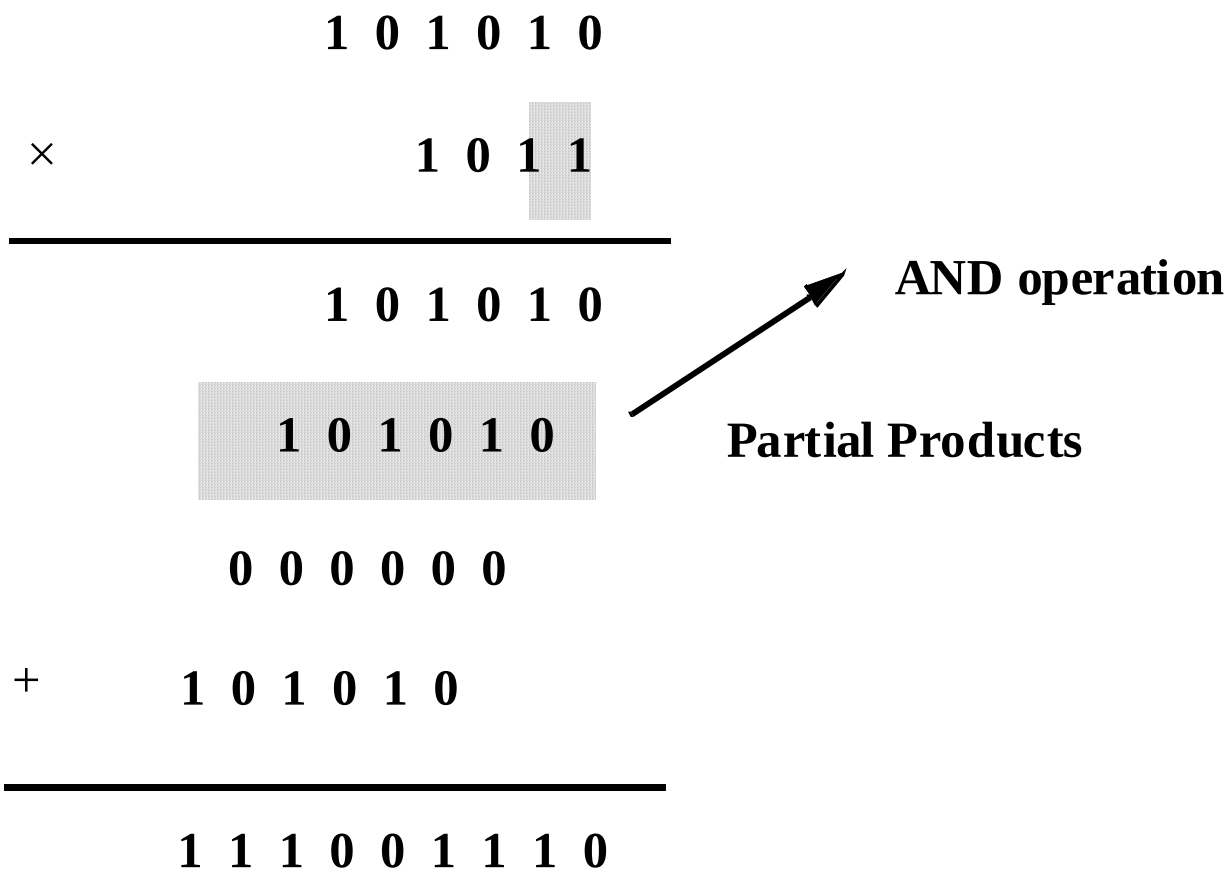
- Method 1



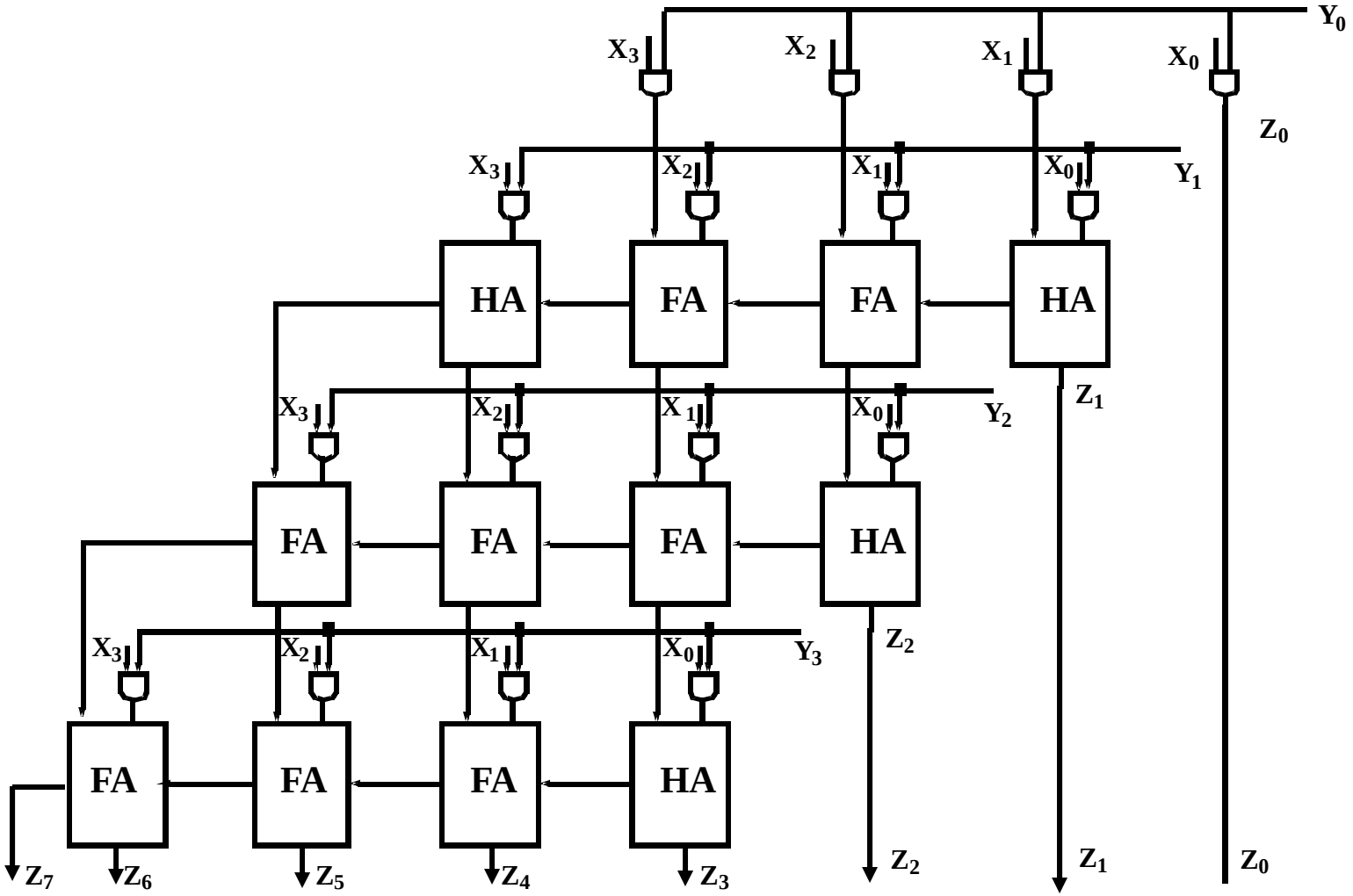
- Method 2



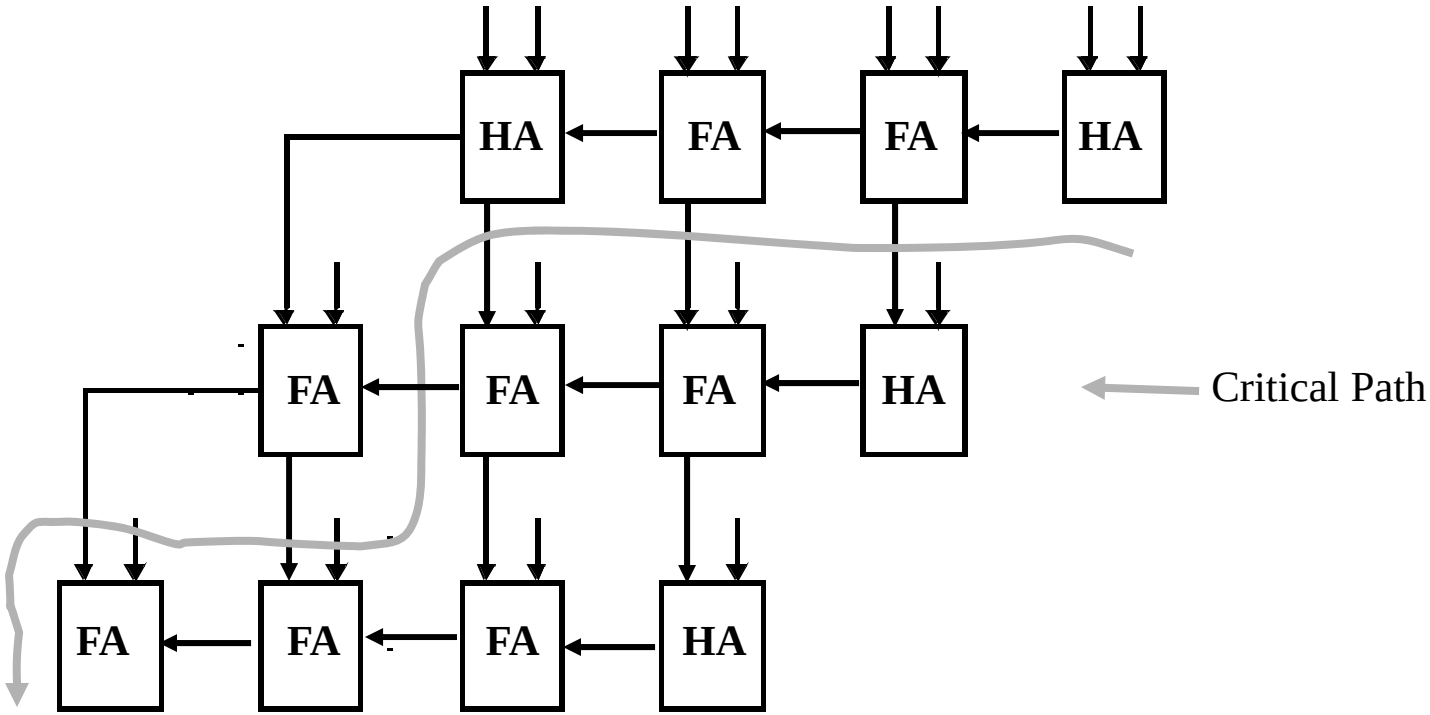
Binary Number Multiplication



Array Multiplier

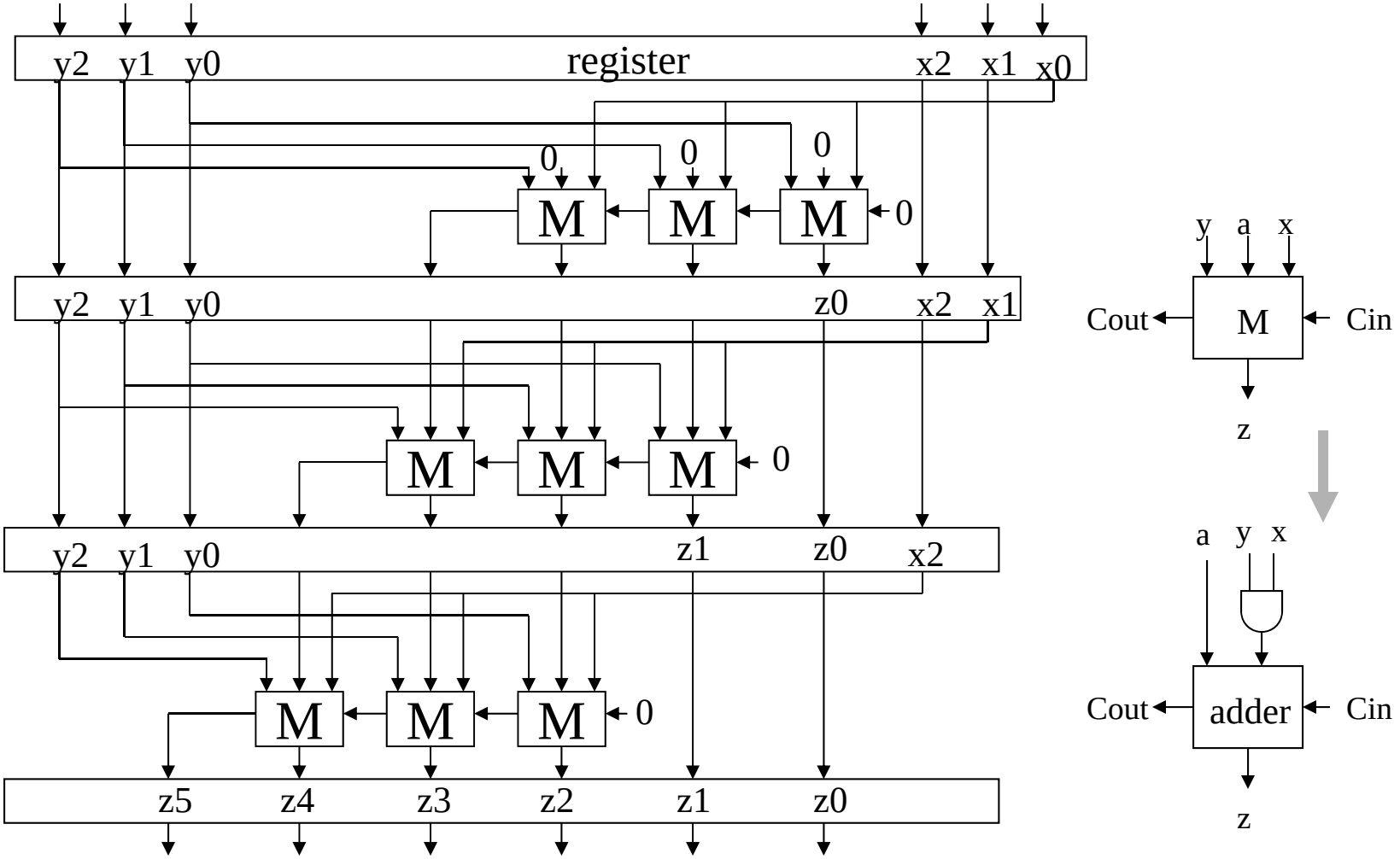


Delay in Array Multiplier

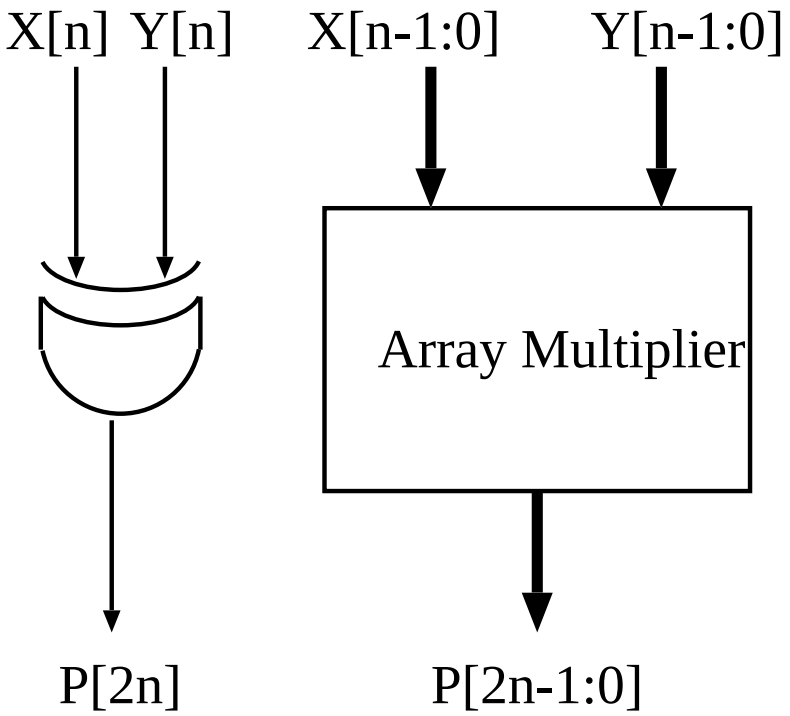


- Fast Multiplier Implementation
  - — Booth multiplier, carry-save multiplier, etc
  - — Pipelined implementation

Pipelined Array Multiplier



Signed-Magnitude Number Multiplication



Baugh-Wooley Multiplier

- The value of 2's complement number $x_{n-1}x_{n-2}\bullet\bullet\bullet\bullet\bullet x_1x_0$ is:

$$X = -x_{n-1} \bullet 2^{n-1} + \sum_{i=0}^{n-2} x_i \bullet 2^i$$

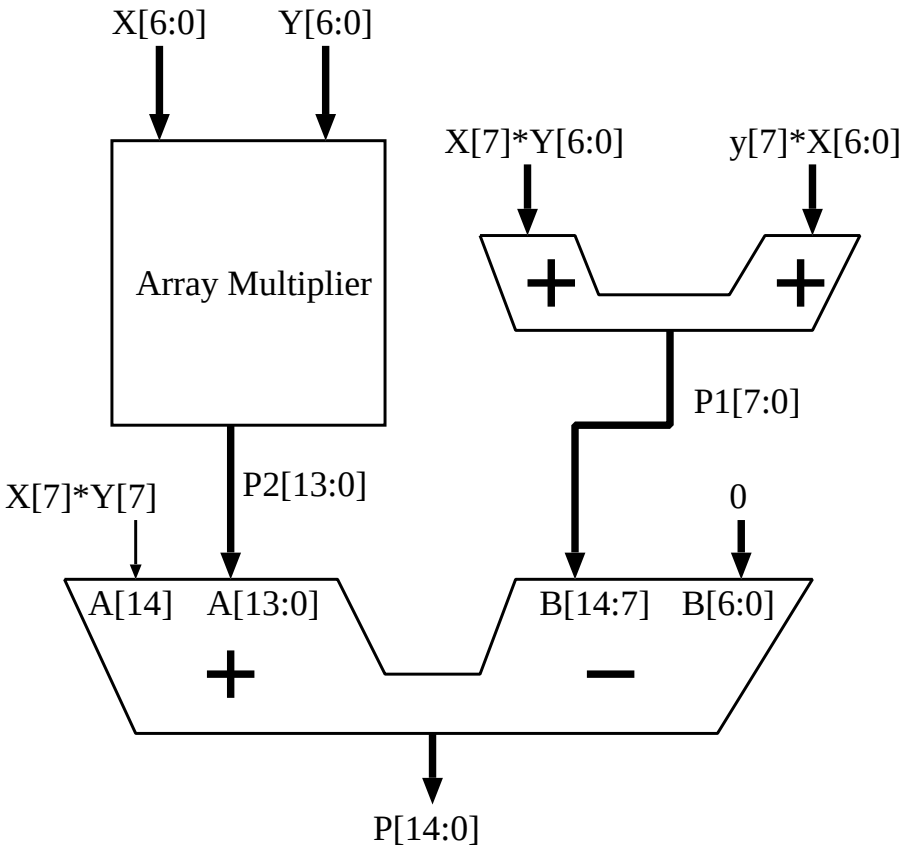
- So, P=X*Y can be calculated by:

$$P = x_{n-1} \bullet y_{n-1} \bullet 2^{2n-2} + (\sum_{i=0}^{n-2} x_i \bullet 2^i) \bullet (\sum_{j=0}^{n-2} y_j \bullet 2^j)$$

$$- 2^{n-1} \bullet (x_{n-1} \bullet \sum_{j=0}^{n-2} y_j \bullet 2^j + y_{n-1} \bullet \sum_{i=0}^{n-2} x_i \bullet 2^i)$$

Baugh-Wooley Multiplier

- Circuit Implementation



Booth Encoding

- The value of 2's complement number $x_{n-1}x_{n-2}\bullet\bullet\bullet\bullet\bullet\ x_1x_0$ is:

$$X = -2^{n-1} \bullet x_{n-1} + 2^{n-2} \bullet x_{n-2} + 2^{n-3} \bullet x_{n-3} + \cdots\cdots + 2^0 \bullet x_0 \qquad (1)$$

- Also: $2^m = 2^{m+1} - 2^m$ (2)

- Substitute Eq. (2) into Eq. (1)

$$X = 2^{n-1} \bullet (x_{n-2} - x_{n-1}) + 2^{n-2} \bullet (x_{n-3} - x_{n-2}) + 2^{n-3} \bullet (x_{n-4} - x_{n-3}) \cdots\cdots \qquad (3)$$

- Let: $B_m = 2 \bullet (x_m - x_{m+1}) + (x_{m-1} - x_m)$ (4)

- Eq. (3) can be written as:

$$X = 2^{n-2} \bullet B_{n-2} + 2^{n-4} \bullet B_{n-4} + 2^{n-6} \bullet B_{n-6} \cdots\cdots \qquad (5)$$

Booth Encoding for data with even number of bits

- Example: a 6-bit data $x_5x_4x_3x_2\ x_1x_0$

$$X = \left\{2^5 \bullet (x_4 - x_5) + 2^4 \bullet (x_3 - x_4)\right\} + \left\{2^3 \bullet (x_2 - x_3) + 2^2 \bullet (x_1 - x_2)\right\} + \left\{2^1 \bullet (x_0 - x_1) + 2^0 \bullet (x_{-1} - x_0)\right\}$$

- The last term (in red color) is needed. Otherwise, the contribution of x_0 will be $2x_0$. And $x_{-1}=0$
- The expression in each {} bracket corresponds to a B_m in Booth encoding
- Hence, the bits are grouped as follows for Booth encoding

$$\overbrace{x_5\ x_4}^{B_4}\ \overbrace{x_3\ x_2}^{B_2}\ \overbrace{x_1\ x_0\ 0}^{B_0}$$

Booth Encoding for data with odd number of bits

- Example: a 5-bit data $x_4x_3x_2\ x_1x_0$

$$X = \left\{2^4 \bullet (x_3 - x_4) + 2^3 \bullet (x_2 - x_3)\right\} + \left\{2^2 \bullet (x_1 - x_2) + 2^1 \bullet (x_0 - x_1)\right\} + \left\{2^0 \bullet (x_{-1} - x_0) + 2^{-1} \bullet (x_{-2} - x_{-1})\right\}$$

- The last term (in red color) is needed. Otherwise, the contribution of x_0 will be $2x_0$. And $x_{-1}=0$, $x_{-2}=0$
- The expression in each {} bracket corresponds to a B_m in Booth encoding
- Hence, the bits are grouped as follows for Booth encoding

$$\overbrace{x_4\ x_3}^{B_3}\ \overbrace{x_2\ x_1}^{B_1}\ \overbrace{x_0\ 0\ 0}^{B_{-1}}$$

Booth Encoding

- P=X*Y can be calculated by:

$$P = 2^{n-2} \bullet B_{n-2} \bullet Y + 2^{n-4} \bullet B_{n-4} \bullet Y + 2^{n-6} \bullet B_{n-6} \bullet Y \cdots\cdots$$

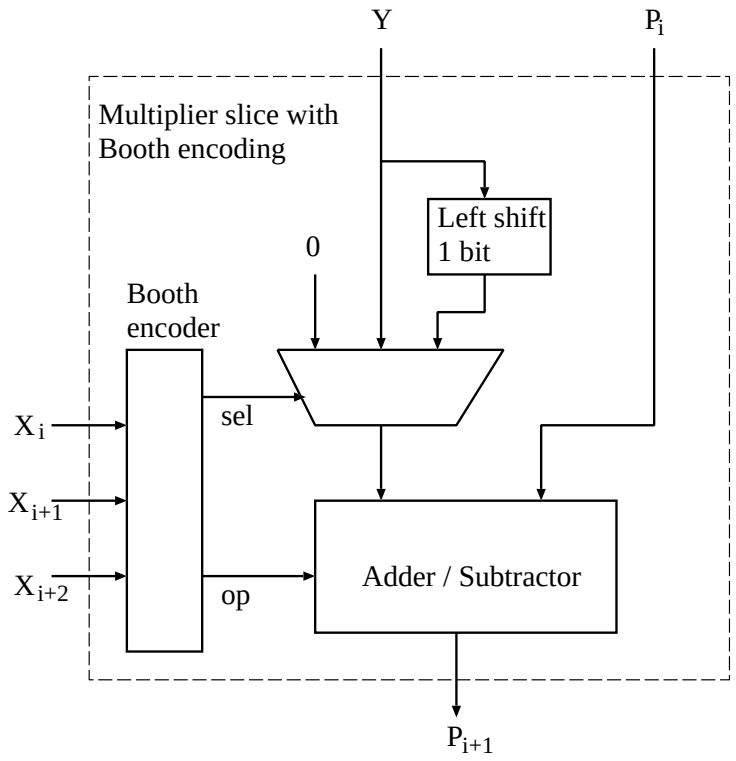
- Possible B_m values are:

x_{m+1}	x_m	x_{m-1}	B_m
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

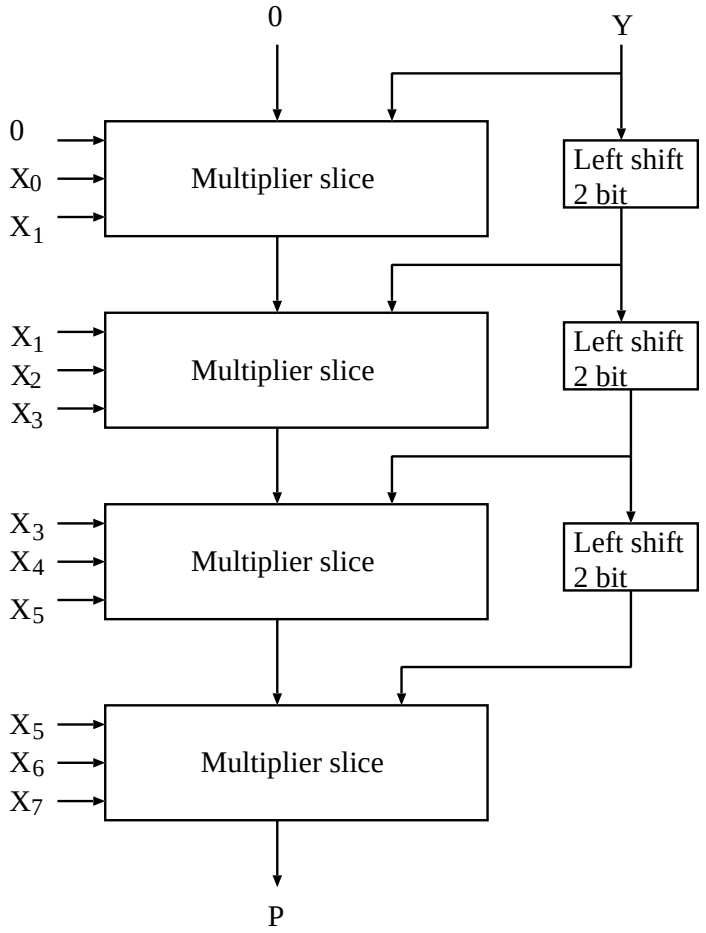
$$B_m = 2 \bullet (x_m - x_{m+1}) + (x_{m-1} - x_m)$$

8-bit Multiplier with Booth encoding

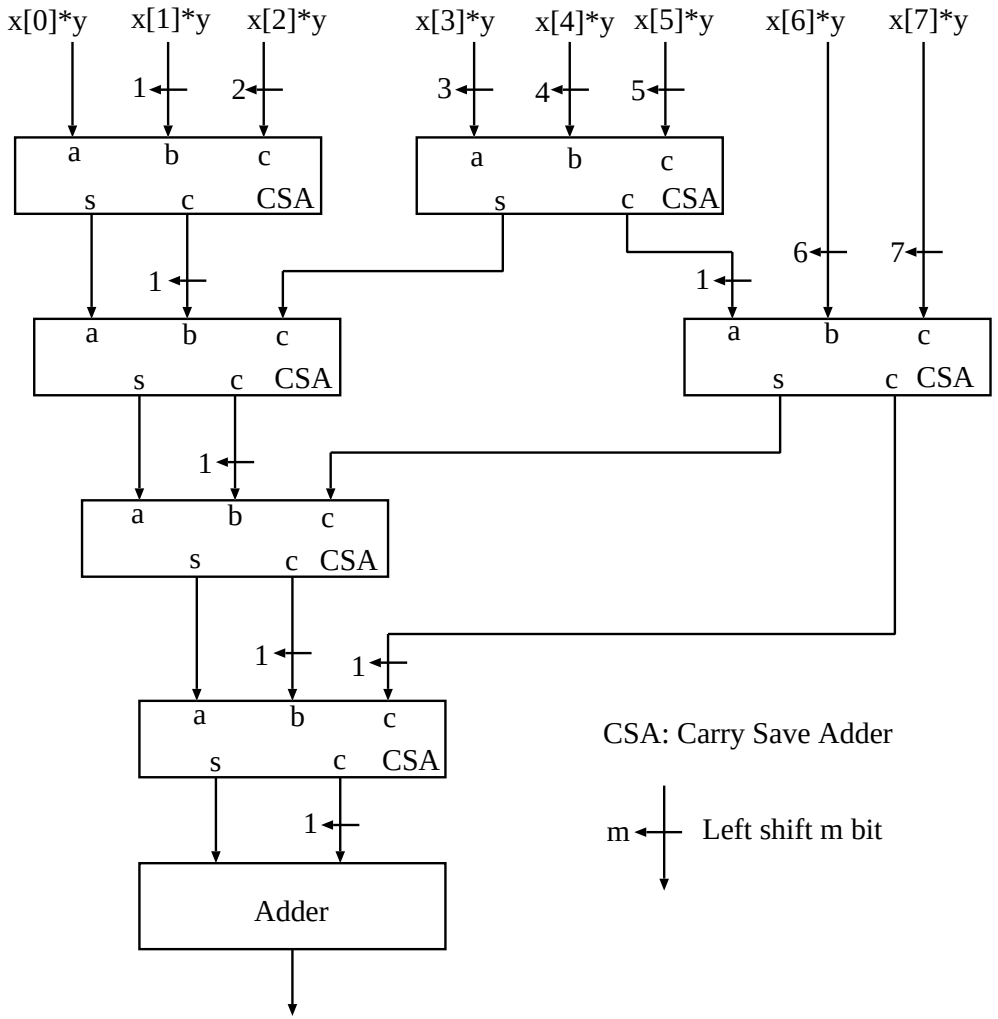
❑ Multiplier Slice



❑ Multiplier Circuit

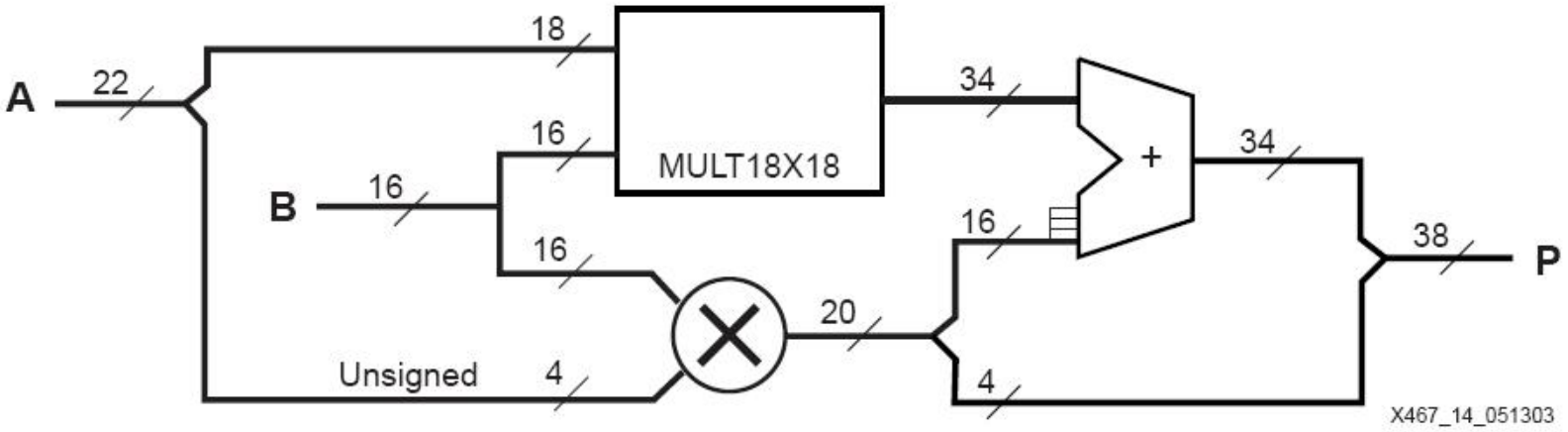


Wallace Tree Multiplier



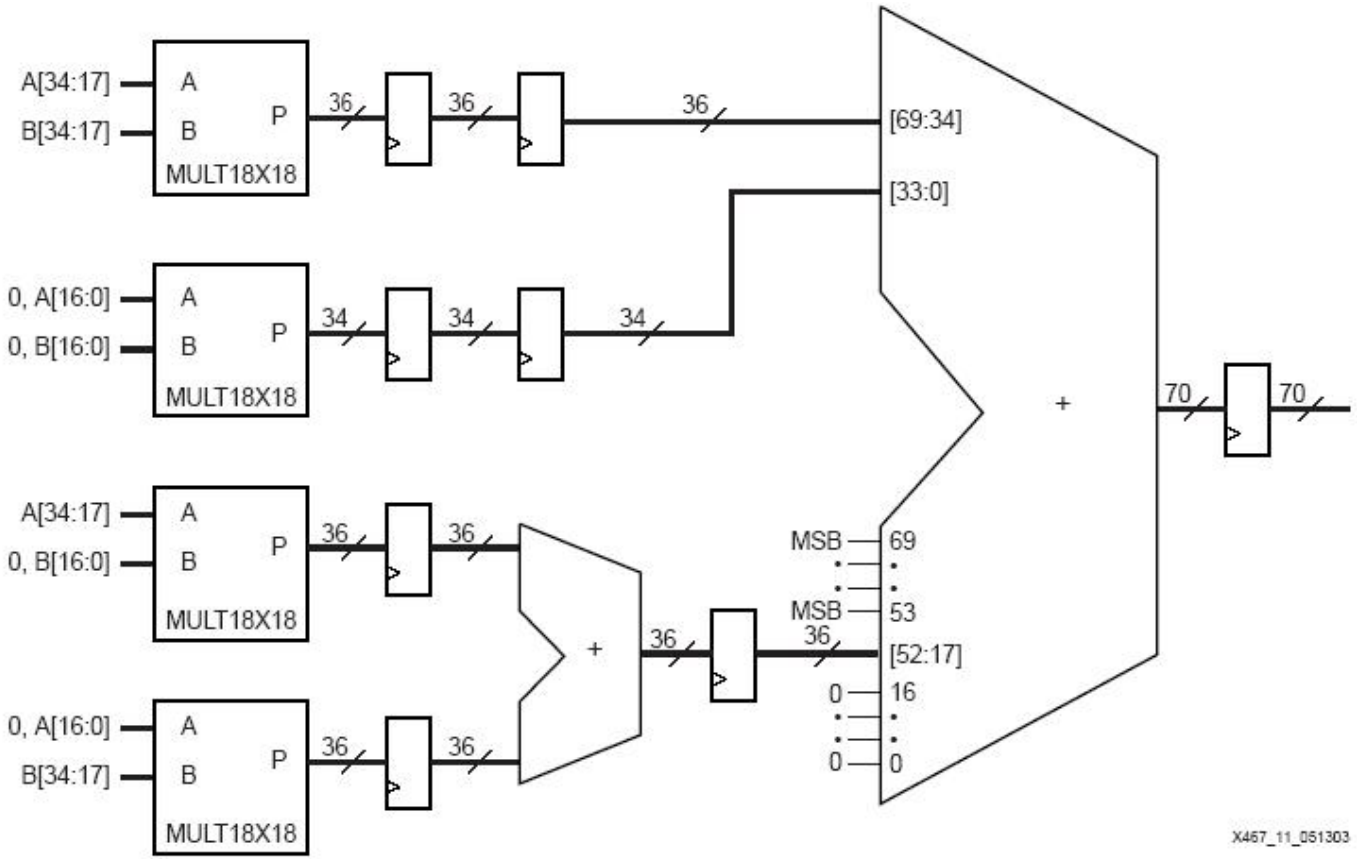
Building Large Multiplier

- Modern FPGAs contain embedded multipliers for high performance DSP applications.
- Embedded multipliers can be used to design different size multipliers
- 18x16-bit signed multiplier



Building Large Multiplier

- 35x35-bit signed multiplier



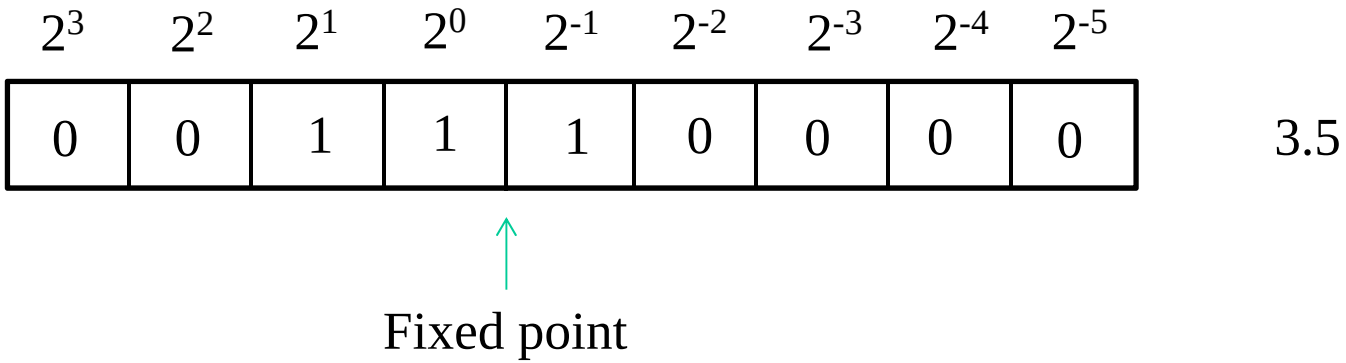
Division

- The results of addition, subtraction, and multiplication are integer numbers. However, division often leads to fraction number results.
- Assume we use fixed-point format to represent the fraction numbers.

Fixed point

$$b_n\, b_{n-1}\, b_{n-2} \bullet\bullet\bullet b_0 \bullet b_{-1}\, b_{-2} \bullet\bullet\bullet b_{-m}$$

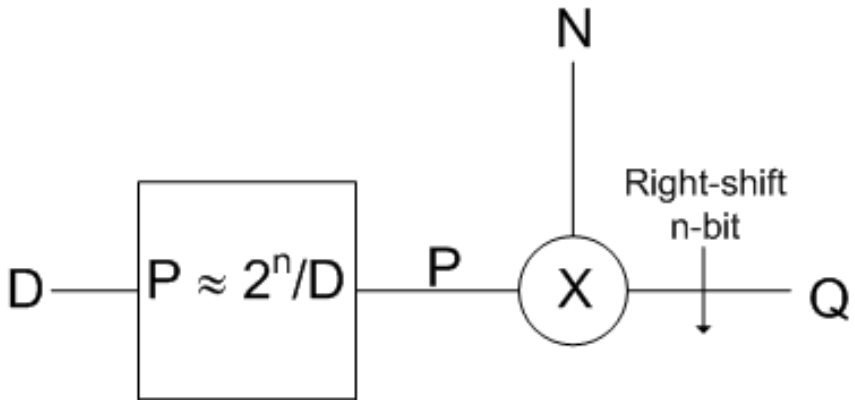
$$= \sum_{i=1}^{n} b_i \bullet 2^i + \sum_{j=1}^{m} b_j \bullet 2^j$$



Division by multiplication and shifting

$$Q = \frac{N}{D} = N \times \frac{2^n}{D} \div 2^n$$

- If 2^n is large enough, $2^n/D$ can be approximated by an integer number P. Then, $N \times 2^n/D$ can be approximated by $N \times P$. In FPGA implementation, A LUT table can be used to store the P values for D
- Divided by 2^n can be performed by an n-bit right-shifting operation.



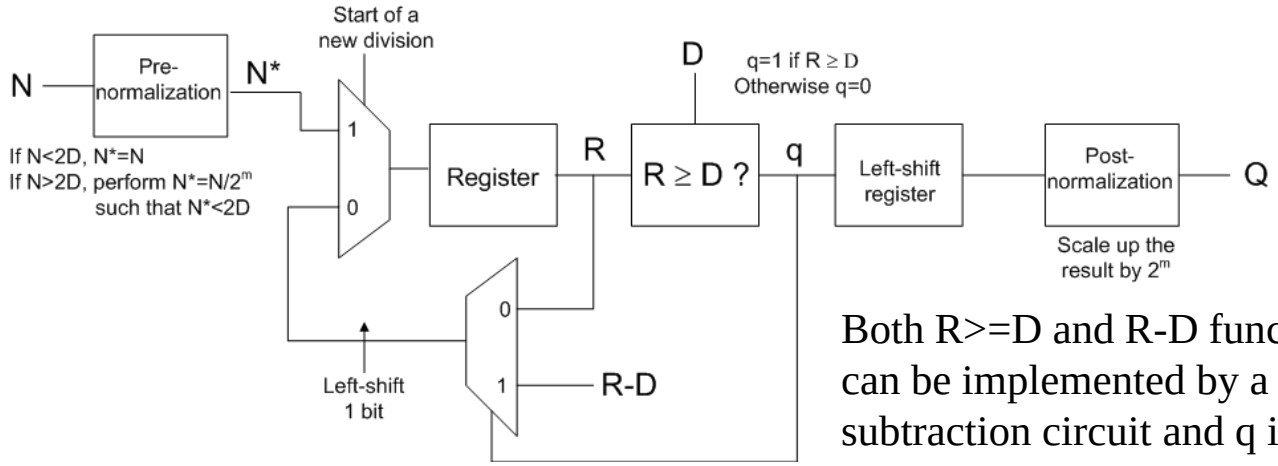
- Advantage: Fast
- Drawback: low precision.

Division by iterative subtraction

- Similar to the "pencil and paper" method:
  - If N>D, find the maximum number q such that N-q*D<D. q is the quotient of the current bit.
  - If N<D, the quotient of the current bit is 0.

```
          0. 1 0 1 0 1 0 1
   1 1  ) 1 0
          0 0
          1 0 0
            1 1
              1 . . .
```

- Since we are working on binary numbers (either 1 or 0), the pre-requirement of N<2D will lead to more regular operations
- Block diagram of a iterative divider (assume N>0, D>0)



Both R>=D and R-D functions can be implemented by a single subtraction circuit and q is the carry out of the circuit

Division by Goldschmidt method

Calculate N/D (assume N ≥1 and D<2)

1: *Let L_1 be an approximation of 1/D (L_1 is provided by an LUT)*

2: $Q_1=L_1*N$, $e_1=L_1*D$

3: $L_2=2-e_1$, $Q_2=Q_1*L_2$, $e_2=e_1*L_2$

4: *continuously perform similar steps as Step 3*
$L_i=2-e_{i-1}$, $Q_i=Q_{i-1}*L_i$, $e_i=e_{i-1}*L_i$

- Since L_1 is an approximation of 1/D, we have:

$$L_1 = \frac{1}{D} + \Delta \quad (\Delta \text{ is a small error term})$$

Division by Goldschmidt method

$$Q_1 = N * L_1 = N * (\frac{1}{D} + \Delta) = \frac{N}{D} + N\Delta$$

$$e_1 = D * (\frac{1}{D} + \Delta) = 1 + D\Delta$$

$$L_2 = 2 - e_1 = 1 - D\Delta$$

$$Q_2 = Q_1 L_2 = (\frac{N}{D} + N\Delta)(1 - D\Delta) = \frac{N}{D} - ND\Delta^2$$

$$e_2 = e_1 L_2 = (1 + D\Delta)(1 - D\Delta) = 1 - D^2\Delta^2$$

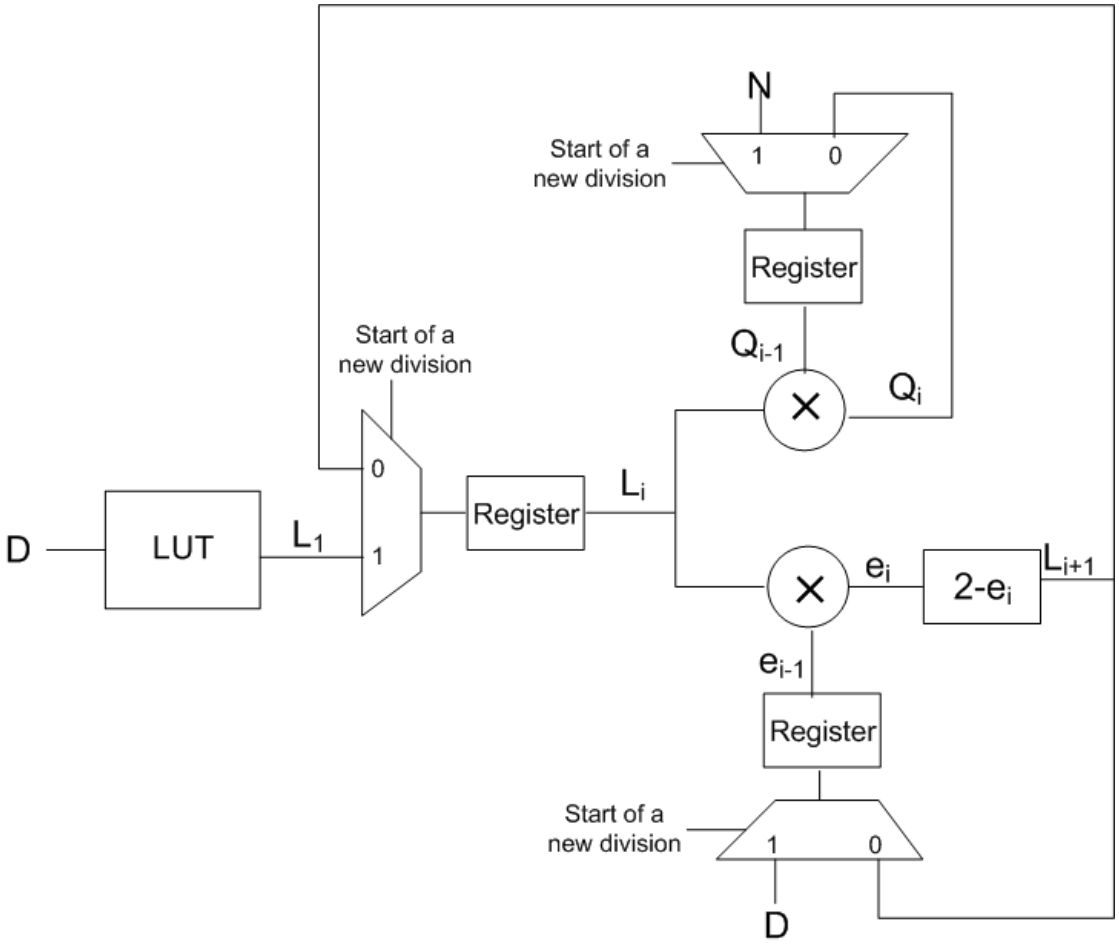
$$L_3 = 2 - e_2 = 1 + D^2\Delta^2$$

$$Q_3 = Q_2 L_3 = (\frac{N}{D} - ND\Delta^2)(1 + D^2\Delta^2) = \frac{N}{D} - ND^3\Delta^4$$

$$e_3 = e_2 L_3 = (1 - D^2\Delta^2)(1 + D^2\Delta^2) = 1 - D^4\Delta^4$$

- In Q_i expression, the first term N/D is the accurate result, the second term is the error term.
- After several iteration, the error becomes significantly small

Division by Goldschmidt method



- Since $e_i < 2$, $2-e_i$ is the same as the 2's complement of e_i. Hence, $2-e_i$ is also written as $-e_i$.

Modulator Operation

- Modular arithmetic circuits are also frequently implemented on FPGAs. One of their applications is cryptography
- A simple modular operation is : A mod B

Calculate A Mod B (assume A>0, B>0)

1: While A>B

2: { A=A-B}

3: Output A

- Examples:
  16 Mod 3 = 1
  7 Mod 5 =2

- A simple implementation