Lect 14

Binary Multiplier

CS221: Digital Design

Dr. A. Sahu
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

Outline

- Array Binary Multiplier
- Sequential Multiplier
- High Radix Multiplier
- Booth Multiplier

- Programmable logic Device (PLD)
  – PLA, PAL, ROM, GAL, CPLD, CLB
  – Software …..HDLs

Delay of Adder

- Ripple Carry Adder (RCA) $= N * T_c = \alpha N$
- Machester RCA $= N * T_m = \alpha N$
- Carry Skip Adder
  Total Delay $= p\,(N/m)\,T_s + (p-1) * (N/m) * m * T_c$
  $T = N * (\,p/m * T_s + (p-1)T_c) = \alpha N$
- Carry Select Adder = Independent of Data
  Delay of select = Ts
  $T = (\,N/m - 1)\,T_s + m\,T_c$
  $T = N * T_s/m + (\,mT_c - T_s) = \alpha N + c$
- CLA : $\log_4 N$, Area: $O(2N)$

Efficient Multiplier Design

Multiplication: paper - pencil method

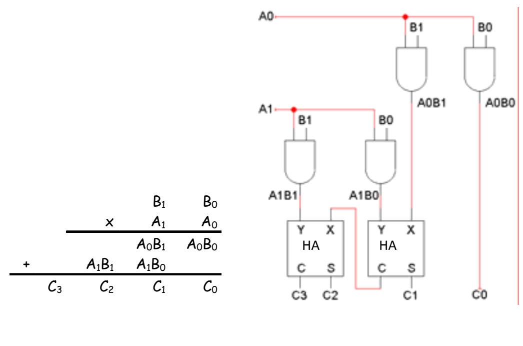
```
A          0 1 1
B          1 0 1
           0 1 1    0 0 0 0 0
         0 0 0 x    0 0 0 1 1
       0 1 1 x x    0 0 0 1 1
AxB    0 1 1 1 1    0 1 1 1 1
```

Binary Multiplier: 2 Bits

```
                 A1  A0
            X    B1  B0
         ----------------
             | B0A1 | B0A0 |     Partial Products
      | B1A1 | B1A0 |
     ------------------
  C3    C2     C1     C0
```

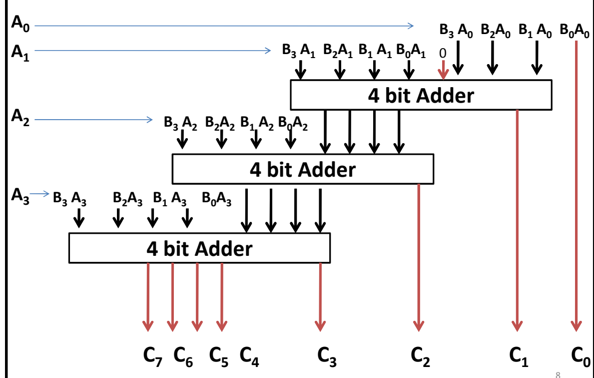
$C_0 = B_0A_0$
$C_1 = B_0A_1 + B_1A_0$
$C_2 = B_1A_1 + \text{Carry of } C_1$
$C_3 = \text{Carry of } C_2$

Binary Multiplier: 2 Bits



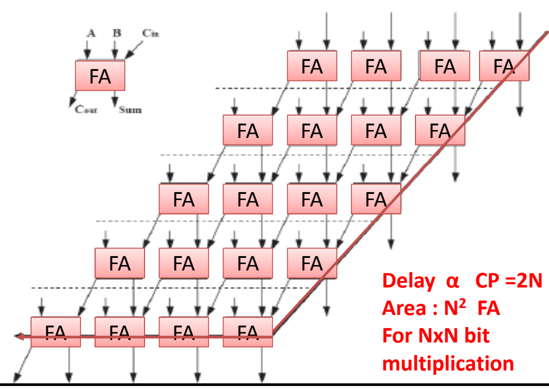
7

Binary Multiplier: 4 Bits



8

Area & Delay (Critical Path): Multiplier



Delay α CP =2N
Area : N^2 FA
For NxN bit multiplication

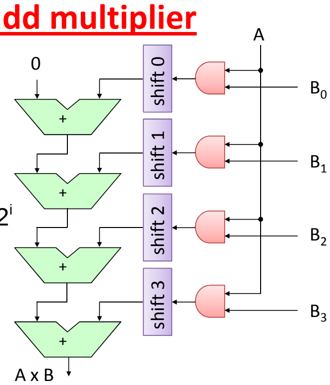
Multiply: Shift & Add

- Decimal number : 15x20=300, 10x20+5x20 =300
- Binary number: 1111 X 10100
  - 1000X**10100** + 100X**10100** + 10X**10100** + 1X**10100**
  - **Sft3(10100) + sft2(10100) + sft1(10100)+sft0(10100)**
  - 1111X10000 + 1111X100
  - Sft5(1111)+sft2(1111)
- Multiplication of N bit number, N shift, N Add, if bit is zero don't add
- Special addition

10

Shift add multiplier

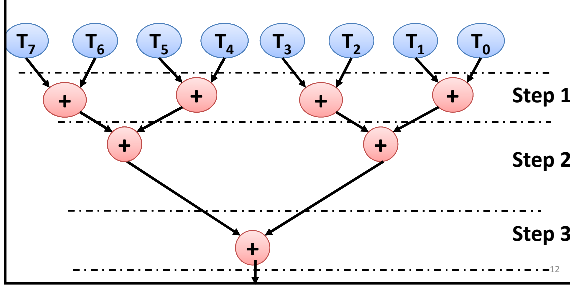
$$A \times B = \sum_{i=0}^{n-1} A \cdot B_i \times 2^i$$



Simple Speeding up

$$A \times B = \sum_{i=0}^{n-1} A \cdot B_i \times 2^i = \sum_{i=0}^{n-1} T_i$$

$$= A.B_0 \times 2^0 + A.B_1 \times 2^1 + .. + A.B_{n-1} \times 2^{n-1}$$



12

Simple Speeding up

$$A \times B = \sum_{i=0}^{n-1} A \cdot B_i \times 2^i = \sum_{i=0}^{n-1} T_i$$

$$= A.B_0 \times 2^0 + A.B_1 \times 2^1 + .. + A.B_{n-1} * 2^{n-1}$$

- **Assumption: Generate All the term in parallel**
- N Addition can be done in parallel in Log(N) steps using N/2 Adder.
- **Adder complexity is Linear O(N) using RCA**
  - Area : Number of Adder*Area Per Adder = N/2 * N
  - Delay : Delay per Adder* Steps = N. lg N
- **Adder complexity CLA Log (N)**
  - Area : Number of Adder*Area Per Adder = N/2 * 2N
  - Delay : Delay per Adder* Steps = lg N. lg N = $(\lg N)^2$

Algorithm Serial Multiplication: D & C

- To multiply two n-digit integers:
  - Multiply **four** ½n-digit integers.
  - Add two ½n-digit integers, and shift to obtain result.

$$\begin{aligned} x &= 2^{n/2}\cdot x_1 + x_0 \\ y &= 2^{n/2}\cdot y_1 + y_0 \\ xy &= \left(2^{n/2}\cdot x_1 + x_0\right)\left(2^{n/2}\cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2}\cdot\left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 \end{aligned}$$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Improved: Karatsuba Multiplication

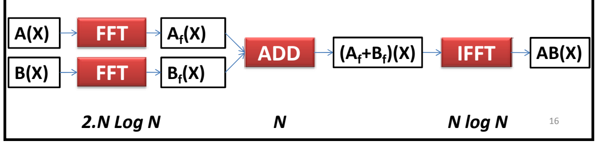
- To multiply two n-digit integers:
  - Add two ½n digit integers.
  - Multiply three ½n-digit integers. (Re use of Term)

$$\begin{aligned} x &= 2^{n/2}\cdot x_1 + x_0 \\ y &= 2^{n/2}\cdot y_1 + y_0 \\ xy &= 2^n \cdot x_1 y_1 + 2^{n/2}\cdot\left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 \\ &= 2^n \cdot x_1 y_1 + 2^{n/2}\cdot\left((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0\right) + x_0 y_0 \end{aligned}$$

$$T(n) = \underbrace{3T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58})$$

N bit Multiplication: FFT Method

- Idea: 1024*16 $=2^{10}+2^4=2^{10+6}=2^{16}=65536$
- FFT based multiplication
  - N Bit binary numbers $A(X)=A_{n-1}2^{n-1}+A_{n-2}2^{n-1}+..+A_0 1$
  - Polynomial multiplication A(X) * B (X)
  - A(X) * B(X) = IFFT (FFT(A(X))+FFT(B(X)));
  - Complexity : 2n *lgn* + n + n *lgn* = O(n *lgn*)



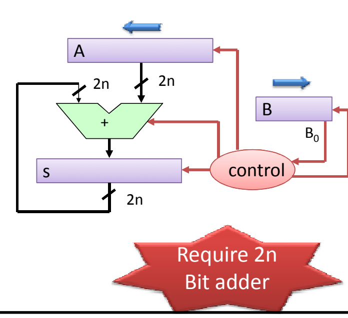
16

Shift add multiplier (sequential)

$$A \times B = \sum_{i=0}^{n-1} A \cdot B_i \times 2^i$$

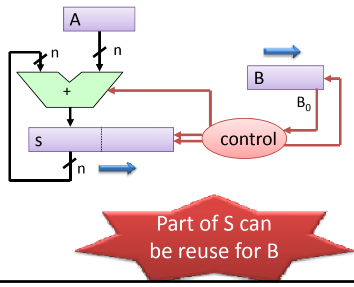
step1: i=0; s=0	step1: i=0; s=0	step1: i=0; s=0
do {	do {	do {
step2:	step2:	step2:
s += $A \cdot B_i$ x 2^i	if (B_i) s += A	if (B_0) s += A
i ++	A = 2xA	A = 2xA
} while (i < n)	i ++	B = B/2; i ++
	} while (i < n)	} while (i < n)

Sequential shift add multiplier 1



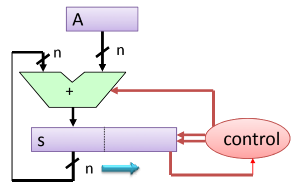
step1: i=0; s=0
do {
step2:
if (B_0) s += A
A = 2xA
B = B/2; i ++
} while (i < n)

Sequential shift add multiplier 2




```
step1: i=0; s=0
do {
  step2:
    if (B0)sH += A
  step3: s = s/2
    B = B/2; i ++
} while (i < n)
```

Sequential shift add multiplier 3



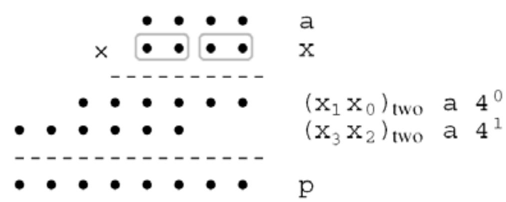

```
step1: i=0; s=0|B
do {
  step2:
    if (s0) sH += A
  step3:
    s = s/2; i ++
} while (i < n)
```

Look Ahead Ref: Sec 8.7, Mano Book , RTL design of Multiplier with FSM/ASM chart.

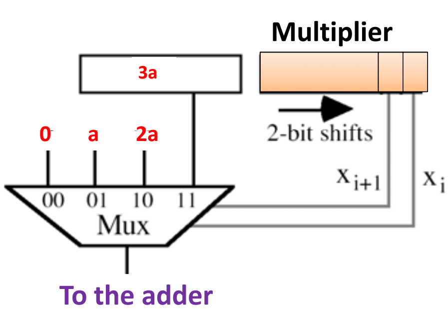
Higher Radix Multiplication

21

Radix-4, or two-bit-at-a-time, multiplication in dot notation



Multiple generation part of a radix-4 multiplier with precomputation of 3*a*



Higher Radix Multiplication

- In radix-8, one must precompute 3a, 5a, 7a
  - Overhead becomes prohibitive and does not help

Higher Radix Multiplication Booth Encoding

Radix-2 Booth Recoding

		j+1 j i	
	1 0 0 1	1 1 0 1 1 0 1 0 1 1 1 0	Operand x
(1)	-1 0 1 0	0 -1 1 0 -1 1 -1 1 0 0 -1 0	Recoded version y

Justification

$$2^j + 2^{j-1} + \cdots + 2^{i+1} + 2^i = 2^{j+1} - 2^i$$

Radix-2 Booth Recoding

x_i	x_{i-1}	y_i	Explanation
0	0	0	No string of 1s in sight
0	1	1	End of string of 1s in x
1	0	-1	Beginning of string of 1s in x
1	1	0	Continuation of string of 1s in x

	1 0 0 1 1 1 0 1 1 0 1 0 1 1 1 0	Operand x
(1)	-1 0 1 0 0 -1 1 0 -1 1 -1 1 0 0 -1 0	Recoded version y

$$y_i = -x_i + x_{i-1}$$

Radix-2 Booth Multiplier Basic Step