Arithmetic Circuits

(Part II)

Randy H. Katz
University of California, Berkeley

Spring 2007

Overview

- *BCD Circuits*
- *Combinational Multiplier Circuit*
- *Design Case Study: 8 Bit Multiplier*
- *Sequential Multiplier Circuit*

BCD Addition

BCD Number Representation

Decimal digits 0 thru 9 represented as 0000 thru 1001 in binary

Addition:

```
5 = 0101          5 = 0101
3 = 0011          8 = 1000
    ----              ----
    1000 = 8          1101 = 13!
```

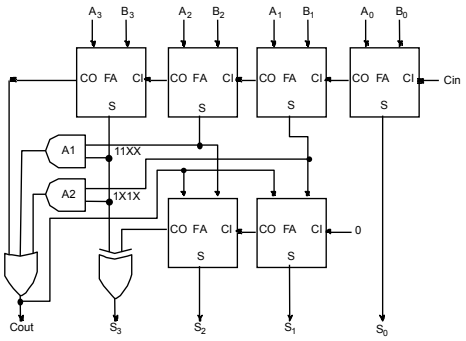
Problem when digit sum exceeds 9

Solution: add 6 (0110) if sum exceeds 9!

```
5 = 0101          9 = 1001
8 = 1000          7 = 0111
    ----              ----
    1101              1 0000 = 16 in binary
6 = 0110          6 = 0110
  ~~1 0011~~ = 1 3 in BCD   ~~1 0110~~ = 1 6 in BCD
```


BCD Addition

Adder Design



Add 0110 to sum whenever it exceeds 1001 (11XX or 1X1X)

Combinational Multiplier

Basic Concept

```
multiplicand        1101  (13)
multiplier        * 1011  (11)
                    ----
                    1101
                   1101
Partial products  0000
                 1101
                 -------
                10001111  (143)
```

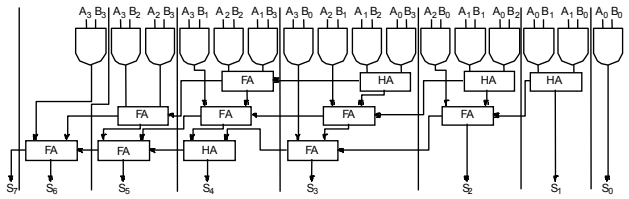
product of 2 4-bit numbers is an 8-bit number

Combinational Multiplier

Partial Product Accumulation

				A3	A2	A1	A0
				B3	B2	B1	B0
				A3 B0	A2 B0	A1 B0	A0 B0
			A3 B1	A2 B1	A1 B1	A0 B1	
		A3 B2	A2 B2	A1 B2	A0 B2		
	A3 B3	A2 B3	A1 B3	A0 B3			
S7	S6	S5	S4	S3	S2	S1	S0

Combinational Multiplier
Partial Product Accumulation



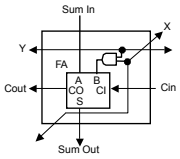
Note use of parallel carry-outs to form higher order sums

12 Adders, if full adders, this is 6 gates each = 72 gates

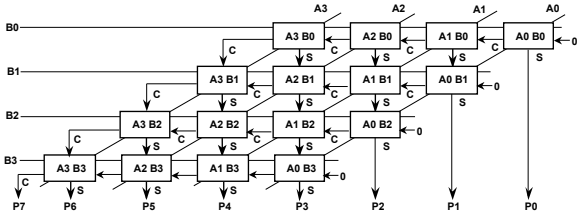
16 gates form the partial products

total = 88 gates!

Combinational Multiplier
Another Representation of the Circuit

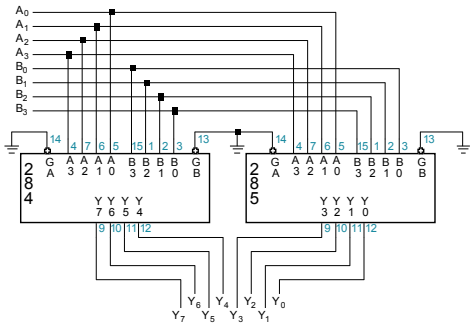


Building block: full adder + and



4 x 4 array of building blocks

Case Study: 8 x 8 Multiplier
TTL Multipliers



Two chip implementation of 4 x 4 multipler

Case Study: 8 x 8 Multiplier
Problem Decomposition

How to implement 8 x 8 multiply in terms of 4 x 4 multiplies?

A7-4 A3-0
* B7-4 B3-0

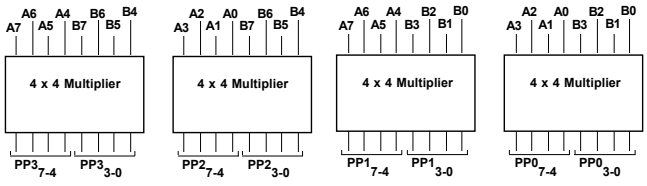
8 bit products

A3-0 * B3-0 = PP0
A7-4 * B3-0 = PP1
A3-0 * B7-4 = PP2
A7-4 * B7-4 = PP3

P15-12 P11-8 P7-4 P3-0

P3-0 = PP0 3-0
P7-4 = PP0 3-0 + PP1 + PP2 + Carry-in
P11-8 = PP1 3-0 + PP2 3-0 + PP3 3-0 + Carry-in
P15-12 = PP3 7-4 7-4 3-0 + Carry-in
7-4

Case Study: 8 x 8 Multiplier
Calculation of Partial Products

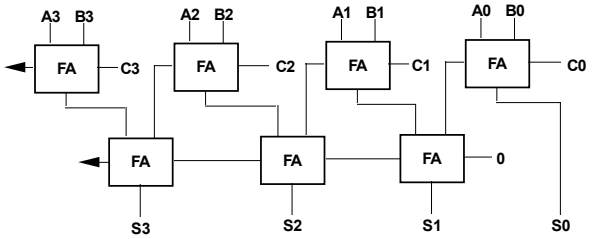


Use 4 4x4 multipliers to create the 4 partial products

Case Study: 8 x 8 Multiplier
Three-At-A-Time Adder

Clever use of the Carry Inputs

Sum A[3-0], B[3-0], C[3-0]:

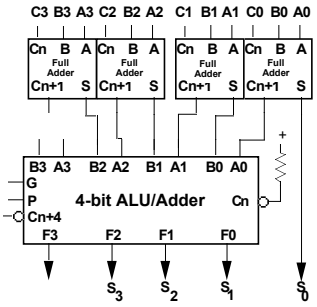


Two Level Full Adder Circuit

Note: Carry lookahead schemes also possible!

Case Study: 8 x 8 Multiplier

Three-At-A-Time Adder with TTL Components



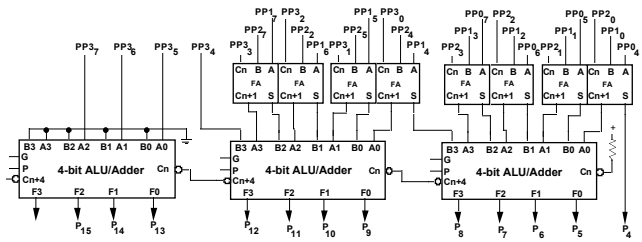
Full Adders
(2 per package)

Standard ALU configured as 4-bit cascaded adder
(with internal carry lookahead)

Note the off-set in the outputs

Case Study: 8 x 8 Multiplier

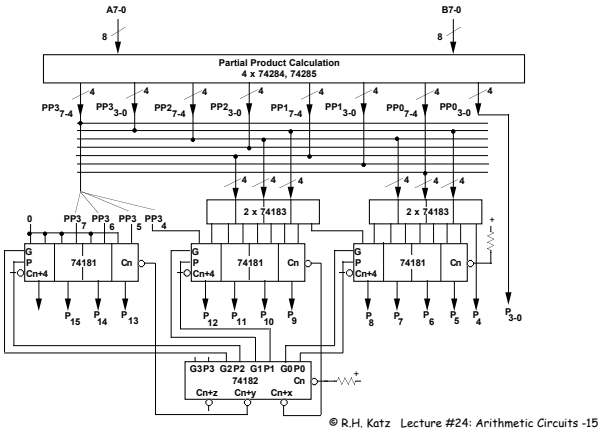
Accumulation of Partial Products



Just a case of cascaded three-at-a-time adders!

Case Study: 8 x 8 Multiplier

The Complete System (Actual TTL packages)



Case Study: 8 x 8 Multiplier

Package Count and Performance

4 74284/74285 pairs = 8 packages
4 74183, 3 74181, 1 74182 = 8 packages
16 packages total

Partial product calculation (74284/285) = 40 ns typ, 60 ns max

Intermediate sums (74183) = 9 ns/20ns = 15 ns average, 33 ns max

Second stage sums w/carry lookahead

- 74LS181: carry G and P = 20 ns typ, 30 ns max
- 74182: second level carries = 13 ns typ, 22 ns max
- 74LS181: formations of sums = 15 ns typ, 26 ns max

103 ns typ, 171 ns max

Sequential Multiplier

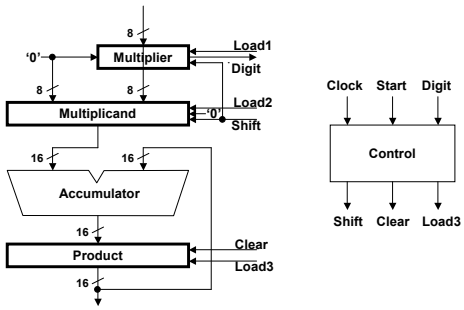
- 4-Bit Multiplier Example: 3 x 4 = 12
  - Four cycles to completion

Cycle	Multiplier	Multiplicand	Product
Initialize	0011	0000 0100	0000 0000
Cycle 0, Multiplier[0]=1	0001	0000 1000	0000 0100
Cycle 1, Multiplier[0]=1	0000	0001 0000	0000 1100
Cycle 2, Multiplier[0]=0	0000	0010 0000	0000 1100
Cycle 3, Multiplier[0]=0	0000	0100 0000	0000 1100

```
Product = 0
For i = 0 to 3 do
     If Multiplier[0] = 1 then Product = Product + Multiplicand
     Shift right the Multiplier
     Shift left the Multiplicand
```

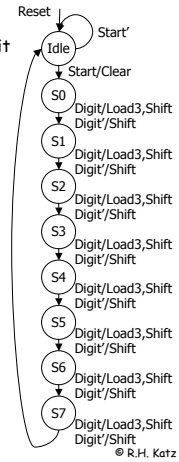

Sequential Multiplier

- Datapath and Control Block Diagram
  - External load signals for Multiplier and Multiplicand
  - Low order bit of multiplier: if 0, shift; if 1, shift and add



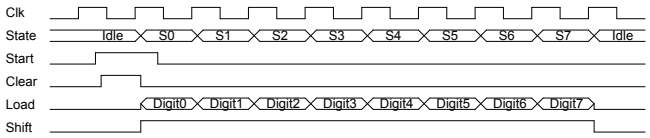
Sequential Multiplier

- State Diagram
  - One state for each bit



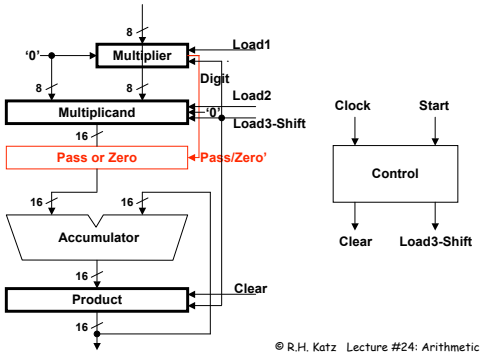
Sequential Multiplier

- Control Timing Behavior



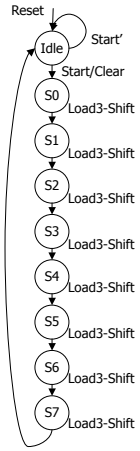
Sequential Multiplier

- Slightly Revised Block Diagram
  - If multiplier low order bit is 0 than assert 0 into accumulator
  - Else pass multiplicand through to accumulator
  - Simplifies the control!



Sequential Multiplier

- Revised State Diagram
  - Fewer control outputs!



Sequential Multiplier

- Symbolic/Encoded State Transition Table
  - State assignment chosen as follows:
    - » Idle state set to 0000, i.e., START resets the state FFs
    - » States S0 to S7 set to 1000 to 1111, easy to implement as a counter

Current State		Start	Next State		Clear	Load3-Shift
Idle	0000	0	0000	Idle	0	0
Idle	0000	1	1000	S0	1	0
S0	1000	-	1001	S1	0	1
S1	1001	-	1010	S2	0	1
S2	1010	-	1011	S3	0	1
S3	1011	-	1100	S4	0	1
S4	1100	-	1101	S5	0	1
S5	1101	-	1110	S6	0	1
S6	1110	-	1111	S7	0	1
S7	1111	-	0000	Idle	0	1

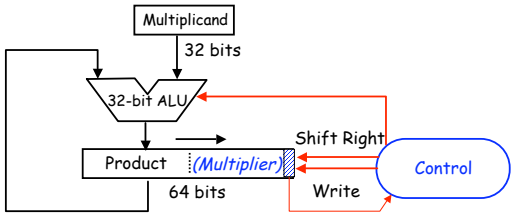
Booth Multiplier

Searching for ways to speed up the basic multiply step

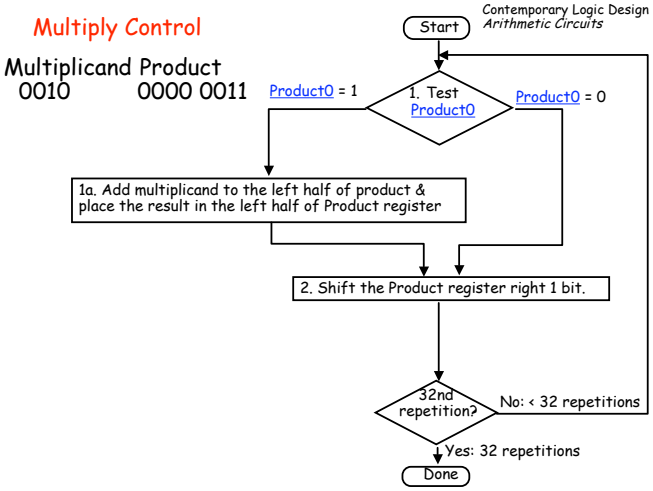
- Tricky encoding scheme to reduce the number of stages in a binary multiplier
- Considers two bits at a time rather than one—this cuts the number of multiplier steps in half
- Each step is slightly more complex compared to the simple multiplier, but is almost as fast as the basic multiplier stage that it replaces

Alternative Multiply Hardware

- 32-bit Multiplicand reg, 32 -bit ALU, 64-bit Product reg, (0-bit Multiplier reg)
- Rather than shift multiplier right and multiplicand left, we can shift BOTH the product and the multiplier to the right …



Multiply Control



Motivation for Booth's Algorithm

- Example 2 x 6 = 0010 x 0110:

```
            0010
      x     0110
      +     0000  shift (0 in multiplier)
      +     0010  add (1 in multiplier)
      +    0010   add (1 in multiplier)
      +   0000    shift (0 in multiplier)
          00001100
```

- If ALU can subtract as well as add, get same result as follows:

```
6     = – 2 + 8
0110  = – 00010 + 01000 = 11110 + 01000
```

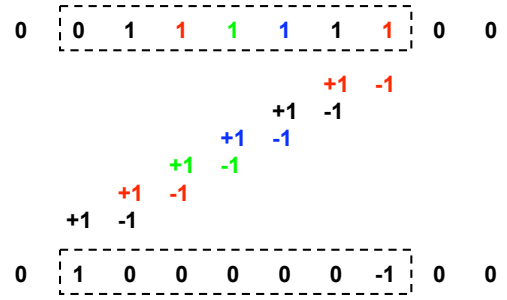
- For example

```
            0010
      x     0110
            0000  shift (0 in multiplier)
      –     0010  sub(first 1 in multiplier)
           0000   shift (mid string of 1s)
      +   0010    add (prior step had last 1)
          00001100
```

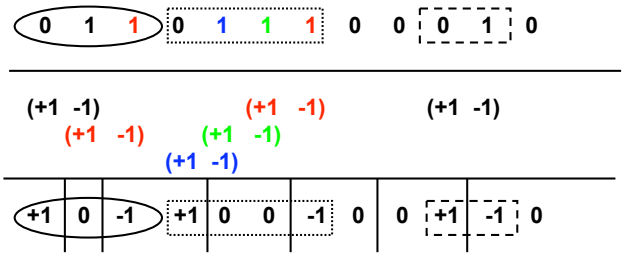
Just add 1111110
(twos complement of 2)

Booth Multiplier: an Introduction

- Recode each 1 in multiplier as "+2-1"
  - Converts sequences of 1 to 10…0(-1)
  - Might reduce the number of 1's



Recoding (Encoding) Example



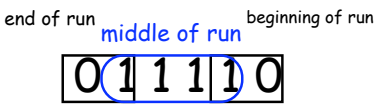
- If you use the last row in multiplication, you should get exactly the same result as using the first row (after all, they represent the same number!)

Booth Multiplication Example

Sign extension

```
                0 0 1 1 0     6x
                0 1 1 1 0     14
              +1 0 0 -1 0
              0 0 0 0 0
1 1 1 1 1 0 1 0               (-6)
      0 0 0 0 0
    0 0 0 0 0
  0 0 1 1 0
  0 0 1 0 1 0 1 0 0           84
```

Booth's Algorithm: Implementation Approach



Current Bit	Bit to the Right	Explanation	Example	Op
1	0	Begins run of 1s	0001111000	sub
1	1	Middle of run of 1s	0001111000	none
0	1	End of run of 1s	0001111000	add
0	0	Middle of run of 0s	0001111000	none

Originally for Speed (when shift is faster than add, it is advantageous to replace adds and subs with shifts)

Basic idea: replace a string of 1s in multiplier with an initial subtract for rightmost 1 in a run of 1's, then later add back a 1 for the bit to the left of the last 1 in the run

```
   -1
+ 10000
  01111
```


Booth's Example (2 x 7)

Operation	Multiplicand	Product	next?
0. initial value	0010	0000 0111 0	10 -> sub
1a. P = P - m	1110	+ 1110 1110 0111 0	shift P (sign ext)
1b.	0010	1111 0011 1	11 -> nop, shift
2.	0010	1111 1001 1	11 -> nop, shift
3.	0010	1111 1100 1	01 -> add
4a.	0010	+0010 0001 1100 1	shift
4b.	0010	0000 1110 0	done

Booth's Example (2 x -3)

Operation	Multiplicand	Product	next?
0. initial value	0010	0000 1101 0	10 -> sub
1a. P = P - m	1110	+1110 1110 1101 0	shift P (sign ext)
1b.	0010	1111 0110 1 + 0010	01 -> add
2a.		0001 0110 1	shift P
2b.	0010	0000 1011 0 + 1110	10 -> sub
3a.	0010	1110 1011 0	shift
3b.	0010	1111 0101 1	11 -> nop
4a		1111 0101 1	shift
4b.	0010	1111 1010 1	done

Lecture Review

We have covered:

- *BCD Adders*
  Simple extension of binary adders
- *Multipliers*
  4 x 4 multiplier: partial product accumulation
  extension to 8 x 8 case
- *Sequential Multipliers*
- *Booth Multiply Step*
  Recoding to speed up the calculation