

ECE 274 Digital Logic – Fall 2008

Datapath Components – Adders and Incrementers

Digital Design 4.3, 4.6



Digital Design

Chapter 4: Datapath Components

Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
<http://www.ddvahid.com>



Copyright © 2007 Frank Vahid

Instructors of courses requiring Vahid's Digital Design textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as [unannotated pdf versions](#) on publicly-accessible course websites, PowerPoint source (or pdf with animations) may [not](#) be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.AhVahid.com> for information.

Datapath Components

Adders

- Adds two N-bit binary numbers
 - 2-bit adder: adds two 2-bit numbers, outputs 3-bit result
 - e.g., 01 + 11 = 100 (1 + 3 = 4)
- Can design using combinational logic design process, but doesn't work well for reasonable-size N
 - *Why not?*

| Inputs | | | | Outputs | | |
|--------|----|----|----|---------|----|----|
| a1 | a0 | b1 | b0 | c | s1 | s0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |

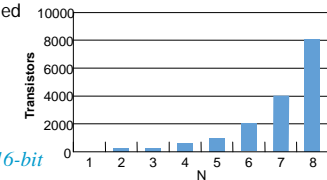
3

Datapath Components

Why Not Use Standard Combinational Design Process

- Truth table too big
 - 2-bit adder's truth table shown
 - Has $2^{(2+2)} = 16$ rows
 - 8-bit adder: $2^{(8+8)} = 65,536$ rows
 - 16-bit adder: $2^{(16+16)} = \sim 4$ billion rows
 - 32-bit adder: ...
- Big truth table with numerous 1s/0s yields big logic
 - Plot shows number of transistors for N-bit adders, using state-of-the-art automated combinational design tool

| Inputs | | | | Outputs | | |
|--------|----|----|----|---------|----|----|
| a1 | a0 | b1 | b0 | c | s1 | s0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |



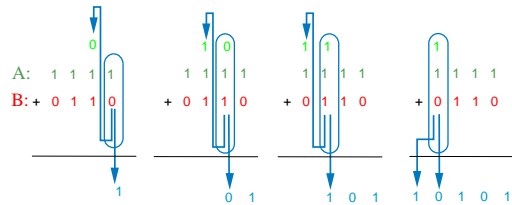
How many of transistors are needed for a 16-bit adder?

4

Datapath Components

Alternative Method: Imitate Adding by Hand

- Alternative Adder Design Method
 - Mimic how people do addition by hand
 - One column at a time
 - Compute sum, add carry to next column

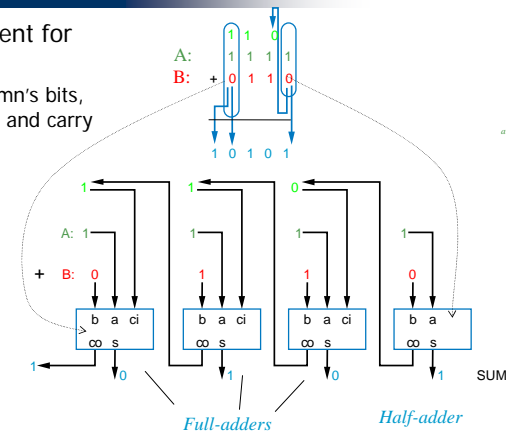


5

Datapath Components

Alternative Method: Imitate Adding by Hand

- Create component for each column
 - Adds that column's bits, generates sum and carry bits



6

Datapath Components

Half-Adder

- Half-adder:** Adds 2 bits, generates sum and carry
- Design using combinational design process from Ch 2

| Inputs | | Outputs | |
|--------|---|---------|---|
| a | b | co | s |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

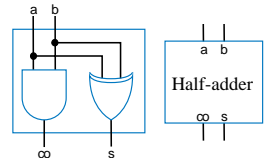
Step 1: Capture the function

Step 2: Convert to equations

$$co = ab$$

$$s = a'b + ab' \quad (\text{same as } s = a \text{ xor } b)$$

Step 3: Create the circuit

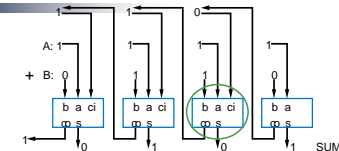


7

Datapath Components

Full-Adder

- Full-adder:** Adds 3 bits, generates sum and carry
- Design using combinational design process from Ch 2



Step 1: Capture the function

| Inputs | | | Outputs | |
|--------|---|----|---------|---|
| a | b | ci | co | s |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Step 2: Convert to equations

$$co = a'bc + ab'c + abc' + abc$$

$$co = a'bc + abc + ab'c + abc + abc' + abc$$

$$co = (a'+a)bc + (b'+b)ac + (c'+c)ab$$

$$co = bc + ac + ab$$

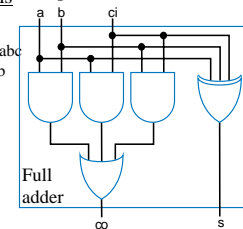
$$s = a'b'c + a'bc' + ab'c' + abc$$

$$s = a'(b'c + bc') + a(b'c' + bc)$$

$$s = a'(b \text{ xor } c) + a(b \text{ xor } c)$$

$$s = a \text{ xor } b \text{ xor } c$$

Step 3: Create the circuit

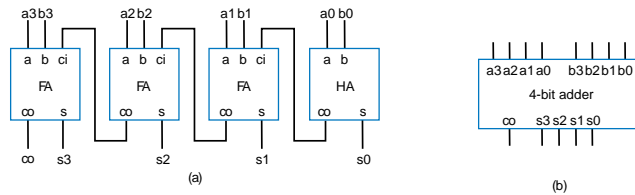


8

Datapath Components

Carry-Ripple Adder

- Using half-adder and full-adders, we can build adder that adds like we would by hand
- Called a **carry-ripple adder**
 - 4-bit adder shown: Adds two 4-bit numbers, generates 5-bit output
 - 5-bit output can be considered 4-bit "sum" plus 1-bit "carry out"
 - Can easily build any size adder

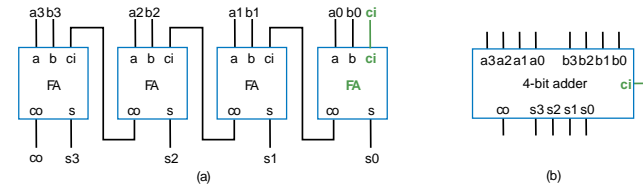


9

Datapath Components

Carry-Ripple Adder

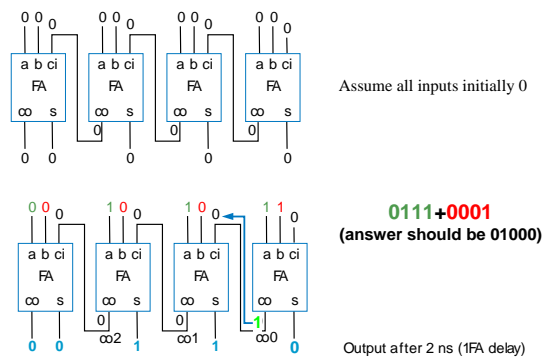
- Using full-adder instead of half-adder for first bit, we can include a "carry in" bit in the addition
 - Will be useful later when we connect smaller adders to form bigger adders



10

Datapath Components

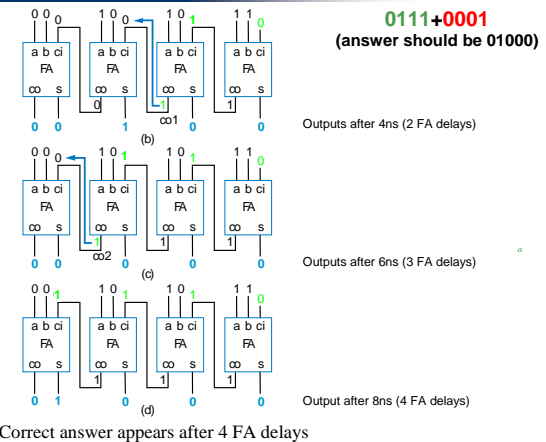
Carry-Ripple Adder's Behavior



11

Datapath Components

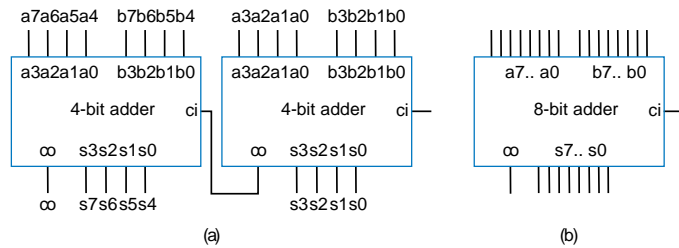
Carry-Ripple Adder's Behavior



12

Datapath Components

Cascading Adders

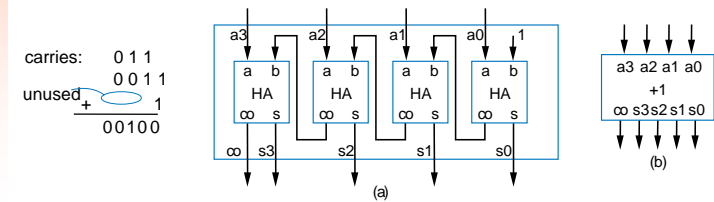


13

Datapath Components

Incrementer

- Counter design used incrementer
- Incrementer design
 - Could use carry-ripple adder with B input set to 00...001
 - But when adding 00...001 to another number, the leading 0's obviously don't need to be considered -- so just two bits being added per column
 - Use half-adders (adds two bits) rather than full-adders (adds three bits)



14