

Chapter 3

COMBINATIONAL CIRCUITS LARGE DESIGNS

Digital Logic Design and Computer Organization with Computer Architecture for Security

1

In this Chapter

- Design methodologies for large combinational circuits
 - Bit-parallel
 - Bit-serial
- Integer arithmetic as examples
 - Add, subtract, multiply, and divide as four basic arithmetic operations
- IEEE floating-point number standards
 - Floating-point Data Space
- Floating-point arithmetic
 - Floating-point unit (FPU)

Digital Logic Design and Computer Organization with Computer Architecture for Security

2

Top-Down Design Methodology

- **Bit-parallel**
 - Partition n-bit design problem into smaller n-bit design problems
 - E.g., 8-bit ALU designed using 8-bit adder/subtractor and 8-bit bit-wise logic
- **Bit-serial**
 - Partition n-bit design problem into a fewer-bit design problem (called slice)
 - E.g., 8-bit ALU designed using eight 1-bit ALU modules
- **Hybrid**
 - Design uses bit-parallel and bit-serial modules

Digital Logic Design and Computer Organization with Computer Architecture for Security

3

Carry Propagate Adder (A bit-serial adder)

- Use FA slices
- Carry bits generated sequentially, one at a time
- Propagation delay proportional to number of carry bits
- Assuming SOP expressions for sum and carry bits and 0.1 ns delay for NANDs determine:
 - $\Delta CPA(8)$
 - $\Delta CPA(32)$
- CPA is the slowest

 $\Delta CPA(8) = 1.7 \text{ ns}$
 $\Delta CPA(32) = 6.5 \text{ ns, too slow}$
Digital Logic Design and Computer Organization with Computer Architecture for Security

4

Carry Look-Ahead (CLA) Adder

- Goal: Generate carry bits in parallel
- Let's examine FA expressions
 - Easy to generate p and g bits in parallel
 - Carry bits are dependent, but can substitute carry expressions to break dependency
 - Once carry bits are known, easy to generate sum bits in parallel
 - $\Delta CLA(8) = ?$

0.8 ns
FA expression from Ch2:

$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

$$c_i = (a_i \oplus b_i) c_{i-1} + a_i b_i$$

Let,

$$p_i = a_i \oplus b_i$$

$$g_i = a_i b_i$$

$$s_i = p_i \oplus c_{i-1}$$

$$c_i = g_i + p_i c_{i-1}$$

Digital Logic Design and Computer Organization with Computer Architecture for Security

5

Observations

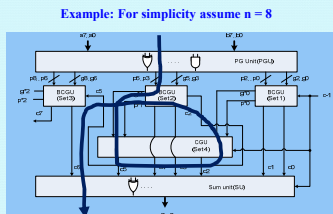
- If keep substituting previous carry expression in next carry expression will run into Fan-in and fan-out problems
- Solution: Generate some carry bits sequentially and some in parallel (next slide)

Digital Logic Design and Computer Organization with Computer Architecture for Security

6

Large CLA Adder

1. Group carry bits into equal sized sets with sets resulting in no fan-in or fan-out problem
 2. Generate carry bits in two steps
- What is the longest signal path from inputs to outputs?
 - Determine $ACLA(32) = 1.2 \text{ ns}$



$ACLA(32) = 1.2 \text{ ns}$

Digital Logic Design and Computer Organization with Computer Architecture for Security

7

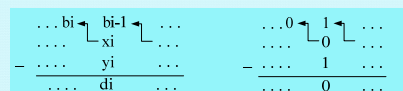
Subtractor

- Similar expressions as FA:

$$d_i = x_i \oplus y_i \oplus b_{i-1}$$

$$b_i = (x_i \oplus y_i) b_{i-1} + \bar{x}_i y_i$$

- Can use adder to do subtraction if both are needed
 - 2's complement adder/subtractor



Digital Logic Design and Computer Organization with Computer Architecture for Security

8

Twos Complement Adder/Subtractor

- $A - B$ can be viewed as adding A with $-B$
- Circuits for $A + B$ and $A + (-B)$ are similar except second input is negated when subtracting
- What do we know about converting negative numbers to 2's complement representation?
 1. Flip bits
 - Can be done with NOT gates
 2. Add 1
- Addition:
 - Do not flip B bits
 - Set carry-in to 0
 - Carry-out not part of the result
- Subtraction:
 - Flip B bits
 - Set carry-in to 1
 - Carry-out not part of the result
- How to combine into one circuit? Use a control bit m for mode.
 - Add when $m = 0$
 - Subtract when $m = 1$
 - Need an inverter circuit controlled by m
- Potential problem?
 - Result can overflow and become incorrect
 - Need overflow detection logic

Digital Logic Design and Computer Organization with Computer Architecture for Security

9

Arithmetic Overflow

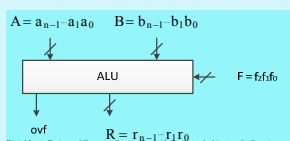
- When sum of two positive numbers is negative
 - I.e., Sign of result becomes 1
 - Applies to subtraction too
 - $A - B$ when $A > 0$ and $B < 0$
- When sum of two negative numbers is positive
 - Sign of result becomes 0
 - Applies to subtraction too
 - $A - B$ when $A < 0$ and $B > 0$
- A simple rule to detect overflow
 - Overflow when carry-in to sign bit position \neq carry-out from sign bit position

Digital Logic Design and Computer Organization with Computer Architecture for Security

10

Arithmetic Logic Unit (ALU)

- Performs arithmetic or bit-wise logic functions
 - A function code specifies which operation to perform
 - A complex combinational circuit
- Need to use bit-parallel or bit-serial design methodology
- Overflow flag (OVF) can only be active when performing arithmetic operations
 - Must be masked otherwise



Digital Logic Design and Computer Organization with Computer Architecture for Security

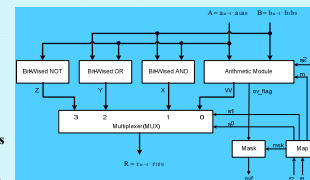
11

Example

f2	f1	f0	Function
0	0	0	Add
0	0	1	Sub
0	1	0	Increment
0	1	1	Decrement
1	0	0	Bitwise AND
1	0	1	Bitwise OR
1	1	0	Bitwise NOT
1	1	1	Not Defined

ALU Bit-Parallel Design

- Identify different types of operations
 - n-bit Arithmetic
 - Add, subtract, increment, decrement
 - Can combine into one 2's complement adder/subtractor
 - n-bit Bit-wise operators
 - NOT, AND, OR
- Assume you have these modules draw a data path
 - Use MUX to select only one output
 - Include other necessary circuits
 - Circuit to convert input F into internal data path signals
 - Circuit to mask OVF during bit-wise operations
- Design modules and assemble



Digital Logic Design and Computer Organization with Computer Architecture for Security

12

Example ALU Modules

- Design Arithmetic module
 - Use n-bit 2's complement adder/subtractor
 - Left input always A
 - Right input either B or 1
 - Need a circuit that outputs B if add/sub or 1 if inc/dec.
 - Use known modules when possible
- Design n-bit 4-to-1 MUX
 - Bit-parallel: Design using n-bit 2-to-1 MUXs (bit-parallel)
 - Bit-serial: Design using 1-bit 4-to-1 MUX slices
- Design Map and Mask circuits
 - Create truth tables
 - Find minimal SOP/POS expressions

Digital Logic Design and Computer Organization with Computer Architecture for Security

13

ALU Bit-Serial Design

- Consider n copies of 1-bit ALU slices
 - Create truth table for 1-bit ALU slice
 - Use Espresso
- May use larger slices
 - 2-bit or 4-bit, for example
 - Larger slices may be designed bit-parallel
- Can be slow for large n

f2	f1	f0	a	b	ci	co	r	Function
0	0	0	0	0	0	0	0	
0	0	0	0	1	0	0	1	
0	0	0	1	0	0	1	1	
0	0	0	1	1	0	1	0	
0	0	1	0	0	0	1	0	
0	0	1	0	1	0	1	1	
0	0	1	1	0	0	1	1	
0	0	1	1	1	0	1	0	
0	1	0	0	0	0	0	0	Add
0	1	0	0	1	0	0	1	
0	1	0	1	0	0	0	1	
0	1	0	1	1	0	0	0	
0	1	1	0	0	0	0	0	
0	1	1	0	1	0	0	1	
0	1	1	1	0	0	0	1	
0	1	1	1	1	0	0	0	
1	0	0	0	0	1	1	1	Sub
1	0	0	0	1	1	1	0	
1	0	0	1	0	1	1	0	
1	0	0	1	1	1	1	1	
1	0	1	0	0	1	1	1	
1	0	1	0	1	1	1	0	
1	0	1	1	0	1	1	0	
1	0	1	1	1	1	1	1	
1	1	0	0	0	1	0	1	Increment
1	1	0	0	1	1	0	1	
1	1	0	1	0	1	0	1	
1	1	0	1	1	1	0	1	
1	1	1	0	0	1	0	0	Decrement
1	1	1	0	1	1	0	0	
1	1	1	1	0	1	0	0	Bit-wise AND
1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	0	Bit-wise OR
1	1	1	1	1	1	0	0	Bit-wise NOT
1	1	1	1	1	1	0	1	Not Defined

Truth Table for 1-bit ALU slice

14

Digital Logic Design and Computer Organization with Computer Architecture for Security

Other design examples (unsigned multiplier)

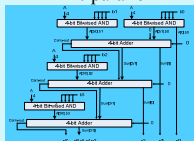
Algorithm

```

  1 0 0 1  A
  * 1 0 1 1  B
  -----
  1 0 0 1
  0 0 0 0
  1 0 0 1
  * 1 0 0 1
  -----
  1 1 0 0 0 1 1  P = B * A
  
```

- Bit-parallel
 - Addends are added one at a time after adding 1st two
 - Less concurrency in the data path
 - Slower, longer propagation delay
- Bit-serial
 - Addend bits are added vertically, the way numbers are added by hand
 - More concurrency in the data path
 - Faster, shorter propagation delay

Bit-parallel



Bit-serial



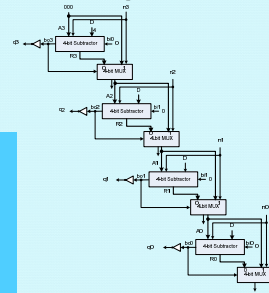
Digital Logic Design and Computer Organization with Computer Architecture for Security

15

Unsigned Divider (restoring)

- Similar to how we divide by hand
 - In each step, remainder can be + or -
 - If remainder positive, use in the next step
 - Ehe, restore
 - Concatenate next numerator bit and repeat
- Bit-parallel
 - Requires subtractor and MUX modules
- Bit-serial
 - Can use 1-bit combined subtractor/MUX slices
 - See Exercise section

Bit-parallel



Digital Logic Design and Computer Organization with Computer Architecture for Security

16

Real Number Arithmetic

IEEE 754 FP number Standards

- Single, 32 bits
 - 1-bit sign, 8-bit biased exponent (bias = 127), 23-bit fraction
 - Stored as a 32-bit number in memory
- double, 64 bits
 - 1-bit sign, 11-bit biased exponent (bias = 1023), 52-bit fraction
 - Stored as a 64-bit number in memory
- Extended, 80 bits
 - 1-bit sign, 15-bit bias exponent (bias = 16383), 64-bit fraction
 - Stored in 80-bit registers only (no memory representation)

Digital Logic Design and Computer Organization with Computer Architecture for Security

17

FP number Data Space (assume 32-bit FP numbers)

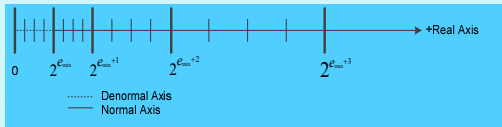
- Normal
 - $1 \leq \text{Biased exponent} \leq 254$
- Denormal
 - Biased exponent = 0 and fraction $\neq 0$
- Zero
 - Biased exponent = 0 and fraction = 0
- Infinity
 - Biased exponent = 255 and fraction = 0
- E.g., $\frac{1}{0}$
- Not-a-number (Nan)
 - Biased exponent = 255 and fraction $\neq 0$
 - E.g., $\sqrt{-1}$

Digital Logic Design and Computer Organization with Computer Architecture for Security

18

Data Space Illustration (1-Dimensional)

- Bold and thin lines indicate real numbers stored as FP numbers in computer
- More fraction bits implies more thin lines
- More exponent bits implies more bold lines

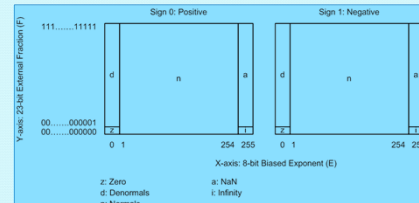


Digital Logic Design and Computer Organization with Computer Architecture for Security

19

Two-Dimensional Illustration

- Easier to identify data space regions
- Easier to mark specific FP numbers or domain or range of a function
 - Eg. The largest FP number
 - E.g., for test generation purposes
 - region identified by $(-1, 1)$ or $[-1, 1]$, for example



Digital Logic Design and Computer Organization with Computer Architecture for Security

20

FP Arithmetic

- Requires integer arithmetic
 - Operates on exponent and fraction numbers independently
 - Typically combinational arithmetic circuits
- Requires shift operations
 - Typically combinational shifter circuits
 - Used to line up implicit decimal points
 - E.g., during FP add
 - Used for normalizing results
 - Result converted to standard format
 - Used for rounding results
 - 64-bit fraction in register is converted to 23 or 52 bits format for storage
 - "float" data type: 23-bit fraction
 - "double" data type: 52-bit fraction
 - The resultant fraction is rounded
 - Based on the value of the bits lost
 - May require another normalization step

Digital Logic Design and Computer Organization with Computer Architecture for Security

21

FP Add (e.g., $S = A + B$)

1. Switch operands (if necessary)
 - For $S = A + B$, $|A|$ must be $\geq |B|$
 2. Align decimal points and compute result $R.F = A.F + B.F$
 3. Normalize $R.F$
 4. Round $R.F$ to produce $S.F$
- Example

Digital Logic Design and Computer Organization with Computer Architecture for Security

22

FP subtract, multiply, divide

- Subtraction
 - Lineup decimal points
 - Compute $A - B$ if $A.s = B.s$ or $A + B$ if $A.s \neq B.s$.
- Multiplication
 - Integer multiply fractions
 - Add exponents
 - XOR the sign bits
- Division
 - Integer divide fractions
 - Subtract exponents
 - XOR the sign bits
- The rounding and normalization steps are the same as in FP add

Digital Logic Design and Computer Organization with Computer Architecture for Security

23