

Combinational logic design case studies

- General design procedure
- Examples
 - calendar subsystem
 - BCD to 7-segment display controller
 - process line controller
 - logical function unit
- Arithmetic
 - integer representations
 - addition/subtraction
 - arithmetic/logic units

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 1

General design procedure for combinational logic

- Understand the problem
 - what is the circuit supposed to do?
 - write down inputs (data, control) and outputs
 - draw block diagram or other picture
- Formulate the problem using a suitable design representation
 - truth table or waveform diagram are typical
 - may require encoding of symbolic inputs and outputs
- Choose implementation target
 - ROM, PAL, PLA
 - mux, decoder and OR-gate
 - discrete gates
- Follow implementation procedure
 - K-maps for two-level, multi-level
 - design tools and hardware description language (e.g., Verilog)

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 2

Calendar subsystem

- Determine number of days in a month (to control watch display)
 - used in controlling the display of a wrist-watch LCD screen

- inputs: month, leap year flag
- outputs: number of days

- Use software implementation to help understand the problem

```

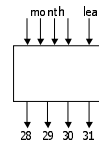
integers number_of_days ( month, leap_year_flag ) {
    switch (month) {
        case 1: return (31);
        case 2: if (leap_year_flag == 1)
                then return (29)
                else return (28);
        case 3: return (31);
        case 4: return (30);
        case 5: return (31);
        case 6: return (30);
        case 7: return (31);
        case 8: return (31);
        case 9: return (30);
        case 10: return (31);
        case 11: return (30);
        case 12: return (31);
        default: return (0);
    }
}
    
```

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 3

Formalize the problem

- Encoding:
 - binary number for month: 4 bits
 - 4 wires for 28, 29, 30, and 31
 - one-hot – only one true at any time
- Block diagram:



month	leap	28	29	30	31
0000	-	-	-	-	-
0001	-	0	0	0	1
0010	0	1	0	0	0
0011	1	0	1	0	0
0011	-	0	0	0	1
0100	-	0	0	1	0
0101	-	0	0	0	1
0110	-	0	0	1	0
0111	-	0	0	0	1
1000	-	0	0	0	1
1001	-	0	0	1	0
1010	-	0	0	0	1
1011	-	0	0	1	0
1100	-	0	0	0	1
1101	-	-	-	-	-
111-	-	-	-	-	-

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 4

Choose implementation target and perform mapping

- Discrete gates

28 = $m8' m4' m2 m1' leap'$

29 = $m8' m4' m2 m1' leap$

30 = $m8' m4 m1' + m8 m1$

31 = $m8' m1 + m8 m1'$

- Can translate to S-o-P or P-o-S

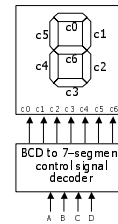
month	leap	28	29	30	31
0000	-	-	-	-	-
0001	-	0	0	0	1
0010	0	1	0	0	0
0011	1	0	1	0	0
0011	-	0	0	0	1
0100	-	0	0	1	0
0101	-	0	0	0	1
0110	-	0	0	1	0
0111	-	0	0	0	1
1000	-	0	0	0	1
1001	-	0	0	1	0
1010	-	0	0	0	1
1011	-	0	0	1	0
1100	-	0	0	0	1
1101	-	-	-	-	-
111-	-	-	-	-	-

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 5

BCD to 7-segment display controller

- Understanding the problem
 - input is a 4 bit bcd digit (A, B, C, D)
 - output is the control signals for the display (7 outputs C0 – C6)
- Block diagram



4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 6

Formalize the problem

- Truth table
 - show don't cares
- Choose implementation target
 - if ROM, we are done
 - don't cares imply PAL/PLA may be attractive
- Follow implementation procedure
 - minimization using K-maps

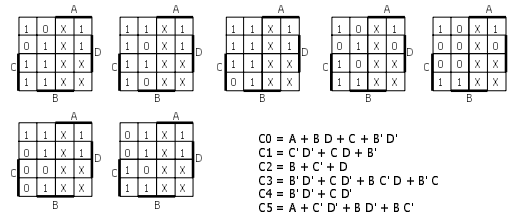
A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	0	0	0	1
0	1	0	0	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	1	1	1	0	0	1	1	1
1	1	0	1	-	-	-	-	-	-	-
1	1	1	1	-	-	-	-	-	-	-

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 7

Implementation as minimized sum-of-products

- 15 unique product terms when minimized individually

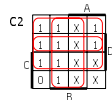


4/25/01

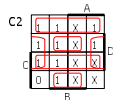
CSE 370 - Spring 2001 - Combinational Examples - 8

Implementation as minimized S-o-P (cont'd)

- Can do better
 - 9 unique product terms (instead of 15)
 - share terms among outputs
 - each output not necessarily in minimized form



$C0 = A + B D + C + B' D'$
 $C1 = C' D' + C D + B'$
 $C2 = B + C' + D$
 $C3 = B' D' + C D' + B C' D + B' C$
 $C4 = B' D' + C D'$
 $C5 = A + C' D' + B D' + B C'$
 $C6 = A + C' D' + B C' + B' C$

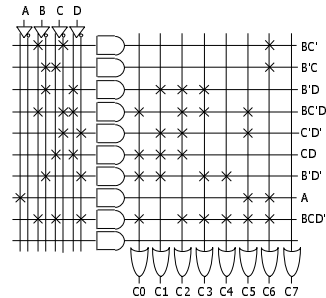


$C0 = B C' D + C D + B' D' + B C D' + A$
 $C1 = B' D + C' D' + C D + B' D'$
 $C2 = B' D + B C' D + C' D' + C D + B C D'$
 $C3 = B C' D + B' D' + B' D' + B C D'$
 $C4 = B' D' + B C D'$
 $C5 = B C' D + C' D' + A + B C D'$
 $C6 = B' C + B C' + B C D' + A$

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 9

PLA implementation



4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 10

PAL implementation

- Limit of 4 product terms per output
 - decomposition of functions with larger number of terms
 - do not share terms in PAL anyway (although there are some with some shared terms)
 - $C2 = B + C' + D$
 - $C2 = B' D + B C' D + C' D' + C D + B C D'$
 - $C2 = B' D + B C' D + C' D' + W$ \leftarrow need another input and another output
 - $W = C D + B C D'$ \leftarrow
 - decompose into multi-level logic (hopefully with CAD support)
 - find common sub-expressions among functions
 - $C0 = C3 + A' B X' + A D Y$
 - $C1 = Y + A' C5' + C' D' C6$
 - $C2 = C5 + A' B' D + A' C D$
 - $C3 = C4 + B D C5 + A' B' X'$
 - $C4 = D' Y + A' C D'$
 - $C5 = C' C4 + A Y + A' B X$
 - $C6 = A C4 + C C5 + C4' C5 + A' B' C$
 - $X = C' + D'$
 - $Y = B' C'$

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 11

Production line control

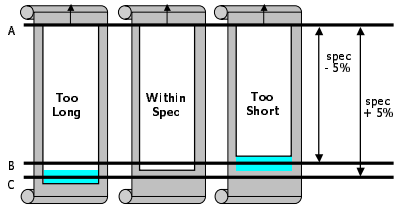
- Rods of varying length (+/-10%) travel on conveyor belt
 - mechanical arm pushes rods within spec (+/-5%) to one side
 - second arm pushes rods too long to other side
 - rods that are too short stay on belt
 - 3 light barriers (light source + photocell) as sensors
 - design combinational logic to activate the arms
- Understanding the problem
 - inputs are three sensors
 - outputs are two arm control signals
 - assume sensor reads "1" when tripped, "0" otherwise
 - call sensors A, B, C

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 12

Sketch of problem

- Position of sensors
 - A to B distance = specification - 5%
 - A to C distance = specification + 5%



4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 13

Formalize the problem

- Truth table
 - show don't cares

A	B	C	Function
0	0	0	do nothing
0	0	1	do nothing
0	1	0	do nothing
0	1	1	do nothing
1	0	0	too short
1	0	1	don't care
1	1	0	in spec
1	1	1	too long

logic implementation now straightforward
just use three 3-input AND gates

"too short" = $AB'C$
(only first sensor tripped)

"in spec" = ABC'
(first two sensors tripped)

"too long" = ABC
(all three sensors tripped)

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 14

Logical function unit

- Multi-purpose function block
 - 3 control inputs to specify operation to perform on operands
 - 2 data inputs for operands
 - 1 output of the same bit-width as operands

C0	C1	C2	Function	Comments
0	0	0	1	always 1
0	0	1	$A + B$	logical OR
0	1	0	$(A + B)'$	logical NAND
0	1	1	$A \text{ xor } B$	logical xor
1	0	0	$A \text{ xnor } B$	logical xnor
1	0	1	$A \cdot B$	logical AND
1	1	0	$(A + B)'$	logical NOR
1	1	1	0	always 0

3 control inputs: C0, C1, C2
2 data inputs: A, B
1 output: F

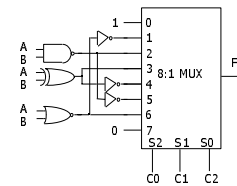
4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 15

Formalize the problem

choose implementation technology
5-variable K-map to discrete gates
multiplexor implementation

C0	C1	C2	A	B	F
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0



4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 16

Arithmetic circuits

- Excellent examples of combinational logic design
- Time vs. space trade-offs
 - doing things fast may require more logic and thus more space
 - example: carry lookahead logic
- Arithmetic and logic units
 - general-purpose building blocks
 - critical components of processor datapaths
 - used within most computer instructions

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 17

Number systems

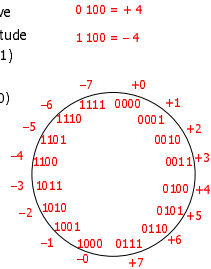
- Representation of positive numbers is the same in most systems
- Major differences are in how negative numbers are represented
- Representation of negative numbers come in three major schemes
 - sign and magnitude
 - 1s complement
 - 2s complement
- Assumptions
 - we'll assume a 4 bit machine word
 - 16 different values can be represented
 - roughly half are positive, half are negative

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 18

Sign and magnitude

- One bit dedicate to sign (positive or negative)
 - sign: 0 = positive (or zero), 1 = negative
- Rest represent the absolute value or magnitude
 - three low order bits: 0 (000) thru 7 (111)
- Range for n bits
 - $\pm(2^{n-1} - 1)$ (two representations for 0)
- Cumbersome addition/subtraction
 - must compare magnitudes to determine sign of result



4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 19

1s complement

- If N is a positive number, then the negative of N (its 1s complement or N') is $N' = (2^n - 1) - N$
 - example: 1s complement of 7

$$\begin{array}{r} 2^4 = 10000 \\ 1 = 00001 \\ 2^4 - 1 = 1111 \\ 7 = 0111 \\ \hline 1000 = -7 \text{ in 1s complement form} \end{array}$$

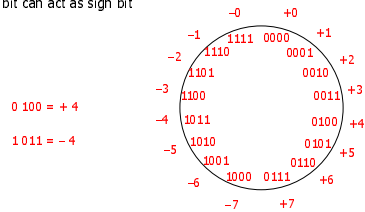
- shortcut: simply compute bit-wise complement (0111 \rightarrow 1000)

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 20

1s complement (cont'd)

- Subtraction implemented by 1s complement and then addition
- Two representations of 0
 - causes some complexities in addition
- High-order bit can act as sign bit



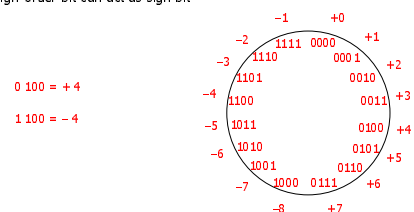
$$\begin{array}{l} 0100 = +4 \\ 1011 = -4 \end{array}$$

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 21

2s complement

- 1s complement with negative numbers shifted one position clockwise
 - only one representation for 0
 - one more negative number than positive number
 - high-order bit can act as sign bit



$$\begin{array}{l} 0100 = +4 \\ 1000 = -4 \end{array}$$

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 22

2s complement (cont'd)

- If N is a positive number, then the negative of N (its 2s complement or N^*) is $N^* = 2^n - N$
 - example: 2s complement of 7

$$\begin{array}{r} 2^4 = 10000 \\ \text{subtract } 7 = 0111 \\ \hline 1001 = \text{repr. of } -7 \end{array}$$

- example: 2s complement of -7

$$\begin{array}{r} 2^4 = 10000 \\ \text{subtract } -7 = 1001 \\ \hline 0111 = \text{repr. of } 7 \end{array}$$

- shortcut: 2s complement = bit-wise complement + 1
 - 0111 \rightarrow 1000 + 1 \rightarrow 1001 (representation of -7)
 - 1001 \rightarrow 0110 + 1 \rightarrow 0111 (representation of 7)

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 23

2s complement addition and subtraction

- Simple addition and subtraction
 - simple scheme makes 2s complement the virtually unanimous choice for integer number systems in computers

$$\begin{array}{r} 4 \quad 0100 \\ + 3 \quad 0011 \\ \hline 7 \quad 0111 \end{array} \quad \begin{array}{r} -4 \quad 1100 \\ + (-3) \quad 1101 \\ \hline -7 \quad 1001 \end{array}$$

$$\begin{array}{r} 4 \quad 0100 \\ - 3 \quad 1101 \\ \hline 1 \quad 1001 \end{array} \quad \begin{array}{r} -4 \quad 1100 \\ + 3 \quad 0011 \\ \hline -1 \quad 1111 \end{array}$$

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 24

Why can the carry-out be ignored?

- Can't ignore it completely
 - needed to check for overflow (see next two slides)
- When there is no overflow, carry-out may be true but can be ignored
 - M + N when N > M:

$$M^* + N = (2^n - M) + N = 2^n + (N - M)$$

ignoring carry-out is just like subtracting 2n

$$-M + -N \text{ where } N + M \leq 2^n - 1$$

$$(-M) + (-N) = M^* + N^* = (2^n - M) + (2^n - N) = 2^n - (M + N) + 2^n$$

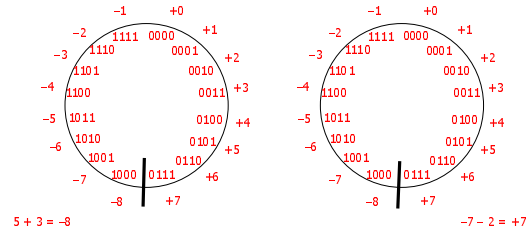
ignoring the carry, it is just the 2s complement representation for -(M + N)

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 25

Overflow in 2s complement addition/subtraction

- Overflow conditions
 - add two positive numbers to get a negative number
 - add two negative numbers to get a positive number

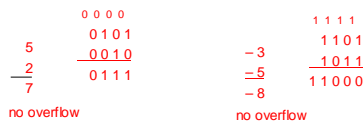
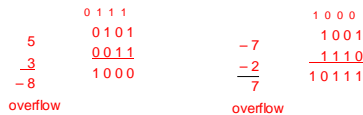


4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 26

Overflow conditions

- Overflow when carry into sign bit position is not equal to carry-out



4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 27

Circuits for binary addition

- Half adder (add 2 1-bit numbers)
 - Sum = A' B + A B' = A xor B
 - Cout = A B
- Full adder (carry-in to cascade for multi-bit adders)
 - Sum = C_i xor A xor B
 - Cout = B C_i + A C_i + A B = C_i (A + B) + A B

A _i	B _i	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

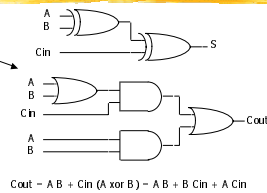
A _i	B _i	C _i	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

4/25/01

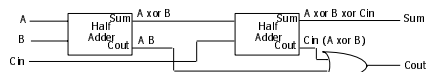
CSE 370 - Spring 2001 - Combinational Examples - 28

Full adder implementations

- Standard approach
 - 6 gates
 - 2 XORs, 2 ANDs, 2 ORs



- Alternative implementation
 - 5 gates
 - half adder is an XOR gate and AND gate
 - 2 XORs, 2 ANDs, 1 OR

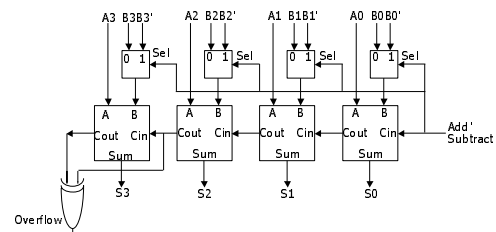


4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 29

Adder/subtractor

- Use an adder to do subtraction thanks to 2s complement representation
 - A - B = A + (-B) = A + B' + 1
 - control signal selects B or 2s complement of B

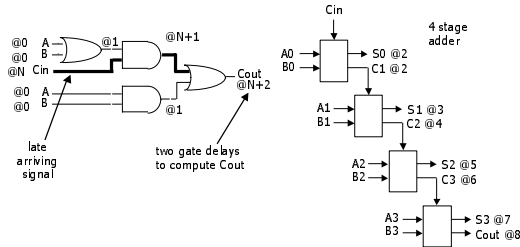


4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 30

Ripple-carry adders

- Critical delay
 - the propagation of carry from low to high order stages

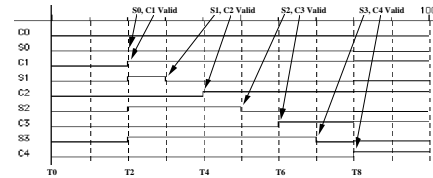


4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 31

Ripple-carry adders (cont'd)

- Critical delay
 - the propagation of carry from low to high order stages
 - 1111 + 0001 is the worst case addition
 - carry must propagate through all bits



4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 32

Carry-lookahead logic

- Carry generate: $G_i = A_i B_i$
 - must generate carry when $A = B = 1$
- Carry propagate: $P_i = A_i \oplus B_i$
 - carry-in will equal carry-out here
- Sum and Cout can be re-expressed in terms of generate/propagate:
 - $S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$
 - $C_{i+1} = A_i B_i + A_i C_i + B_i C_i = A_i B_i + C_i (A_i + B_i) = A_i B_i + C_i (A_i \oplus B_i) = G_i + C_i P_i$

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 33

Carry-lookahead logic (cont'd)

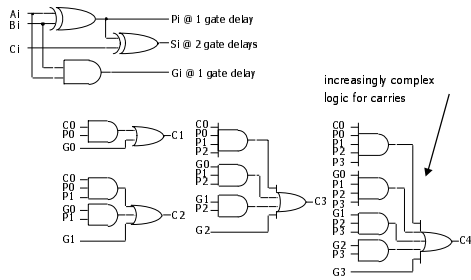
- Re-express the carry logic as follows:
 - $C_1 = G_0 + P_0 C_0$
 - $C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$
 - $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$
 - $C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$
- Each of the carry equations can be implemented with two-level logic
 - all inputs are now directly derived from data inputs and not from intermediate carries
 - this allows computation of all sum outputs to proceed in parallel

4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 34

Carry-lookahead implementation

- Adder with propagate and generate outputs

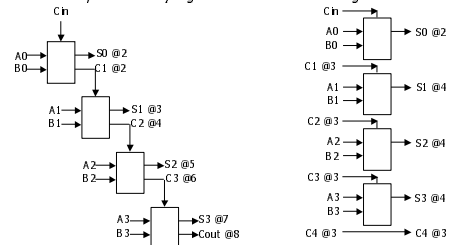


4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 35

Carry-lookahead implementation (cont'd)

- Carry-lookahead logic generates individual carries
 - sums computed much more quickly in parallel
 - however, cost of carry logic increases with more stages

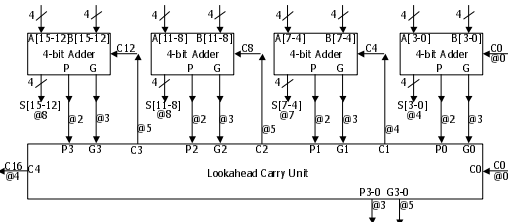


4/25/01

CSE 370 - Spring 2001 - Combinational Examples - 36

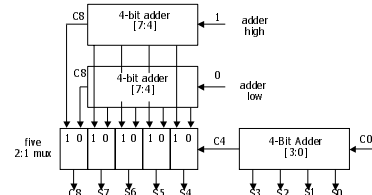
Carry-lookahead adder with cascaded carry-lookahead logic

- Carry-lookahead adder
 - 4 four-bit adders with internal carry lookahead
 - second level carry lookahead unit extends lookahead to 16 bits



Carry-select adder

- Redundant hardware to make carry calculation go faster
 - compute two high-order sums in parallel while waiting for carry-in
 - one assuming carry-in is 0 and another assuming carry-in is 1
 - select correct result once carry-in is finally computed



Arithmetic logic unit design specification

M = 0, logical bitwise operations

S1 S0	Function	Comment
0 0	$F_i = A_i$	input A_i transferred to output
0 1	$F_i = \text{not } A_i$	complement of A_i transferred to output
1 0	$F_i = A_i \text{ xor } B_i$	compute XOR of A_i, B_i
1 1	$F_i = A_i \text{ xnor } B_i$	compute XNOR of A_i, B_i

M = 1, C0 = 0, arithmetic operations

S1 S0	Function	Comment
0 0	$F = A$	input A passed to output
0 1	$F = \text{not } A$	complement of A passed to output
1 0	$F = A \text{ plus } B$	sum of A and B
1 1	$F = (\text{not } A) \text{ plus } B$	sum of B and complement of A

M = 1, C0 = 1, arithmetic operations

S1 S0	Function	Comment
0 0	$F = A \text{ plus } 1$	increment A
0 1	$F = (\text{not } A) \text{ plus } 1$	twos complement of A
1 0	$F = A \text{ plus } B \text{ plus } 1$	increment sum of A and B
1 1	$F = (\text{not } A) \text{ plus } B \text{ plus } 1$	B minus A

logical and arithmetic operations
not all operations appear useful, but "fall out" of internal logic

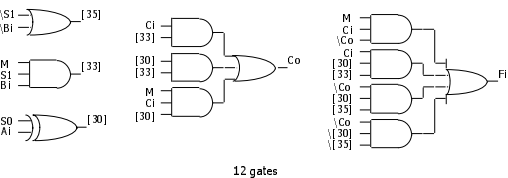
Arithmetic logic unit design (cont'd)

- Sample ALU - truth table

M	S1	S0	C1	A1	B1	F1	C1+1
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	x	x	x
0	0	0	0	1	0	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	x	x	x
0	0	0	1	0	0	0	1
0	0	0	1	0	1	0	1
0	0	0	1	0	x	x	x
0	0	0	1	1	0	0	1
0	0	0	1	1	1	0	1
0	0	0	1	1	x	x	x
0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	x	x	x
0	0	1	0	1	0	0	0
0	0	1	0	1	1	0	0
0	0	1	0	1	x	x	x
0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	1
0	0	1	1	0	x	x	x
0	0	1	1	1	0	0	1
0	0	1	1	1	1	0	1
0	0	1	1	1	x	x	x
1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	x	x	x
1	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0
1	0	0	0	1	x	x	x
1	0	0	1	0	0	0	1
1	0	0	1	0	1	0	1
1	0	0	1	0	x	x	x
1	0	0	1	1	0	0	1
1	0	0	1	1	1	0	1
1	0	0	1	1	x	x	x
1	1	0	0	0	0	0	0
1	1	0	0	0	1	0	0
1	1	0	0	0	x	x	x
1	1	0	0	1	0	0	0
1	1	0	0	1	1	0	0
1	1	0	0	1	x	x	x
1	1	0	1	0	0	0	1
1	1	0	1	0	1	0	1
1	1	0	1	0	x	x	x
1	1	0	1	1	0	0	1
1	1	0	1	1	1	0	1
1	1	0	1	1	x	x	x
1	1	1	0	0	0	0	0
1	1	1	0	0	1	0	0
1	1	1	0	0	x	x	x
1	1	1	0	1	0	0	0
1	1	1	0	1	1	0	0
1	1	1	0	1	x	x	x
1	1	1	1	0	0	0	1
1	1	1	1	0	1	0	1
1	1	1	1	0	x	x	x
1	1	1	1	1	0	0	1
1	1	1	1	1	1	0	1
1	1	1	1	1	x	x	x

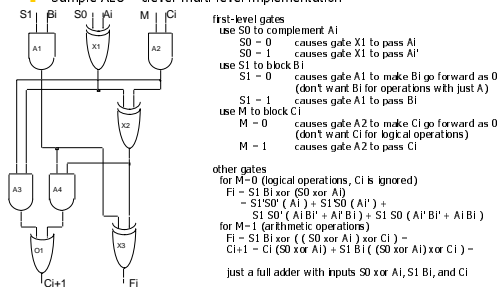
Arithmetic logic unit design (cont'd)

- Sample ALU - multi-level discrete gate logic implementation



Arithmetic logic unit design (cont'd)

- Sample ALU - clever multi-level implementation



Summary for examples of combinational logic

- Combinational logic design process
 - formalize problem: encodings, truth-table, equations
 - choose implementation technology (ROM, PAL, PLA, discrete gates)
 - implement by following the design procedure for that technology
- Binary number representation
 - positive numbers the same
 - difference is in how negative numbers are represented
 - 2s complement easiest to handle: one representation for zero, slightly complicated complementation, simple addition
- Circuits for binary addition
 - basic half-adder and full-adder
 - carry lookahead logic
 - carry-select
- ALU Design
 - specification, implementation