Combinational Logic Review
(Class 1.2 – 1/21/2021)

CSE 4357/CSE 5357 – Advanced Digital Logic Design

Spring 2021

Instructor – Bill Carroll, PhD, PE, Professor

UNIVERSITY OF TEXAS ARLINGTON
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Mandatory Face Covering Policy

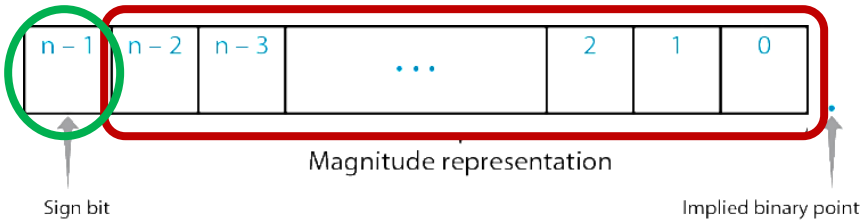
All students and instructional staff are required to wear facial coverings while they are on campus, inside buildings and classrooms. Students that fail to comply with the facial covering requirement will be asked to leave the class session. If students need masks, they may obtain them at the Central Library, the E.H. Hereford University Center's front desk or in their department. Students who refuse to wear a facial covering in class will be asked to leave the session by the instructor, and, if the student refuses to leave, they may be reported to UTA's Office of Student Conduct.

Today's Topics

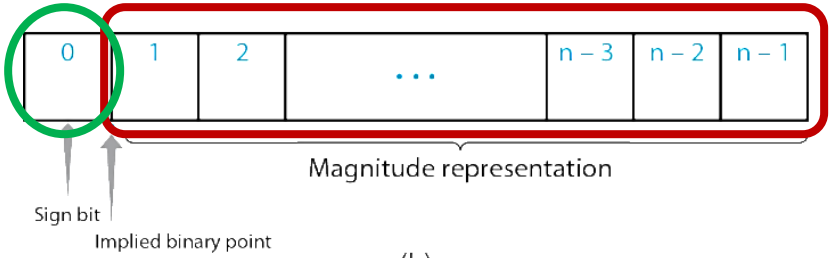
- Signed-number representation
  - Sign-magnitude
  - One's-complement
  - Two's complement
  - Overflow
- Ripple-carry Adder/Subtractor
  - Ripple-carry adder revisited
  - Two's complement subtractor
  - Two's complement adder/subtractor
  - Verilog model
- Overflow detection hardware
- More Verilog review
- Propagation delay
- Add times using basic adders

Fixed-Point Number Representation

- Typical formats for integers and fractions



(a)



(b)

- Signed-number representation
  - ✓ Sign-magnitude
  - ✓ 2's complement
  - ✓ 1's complement

Sign-Magnitude Method

- $N = \pm (a_{n-2} \ldots a_0)_2$ is represented as

  $N = (s a_{n-2} \ldots a_0)_{2sm}$,

  where $s = 0$ if N is positive and $s = 1$ if N is negative.

  – *If* $N = +(101)_2$, then $N = (0101)_{2sm}$

  – *If* $N = -(101)_2$, then $N = (1101)_{2sm}$

- *Range* $2^{n-1} - 1 \geq N \geq -2^{n-1} + 1$ $(011..11 \geq N \geq 111..11)$

- *Pros and cons*

  – Simple and easy for humans to understand

  – Complicates computer arithmetic circuits

    - Two representations of zero, *+0* and *-0*
    - Need both an adder and a subtractor for arithmetic

One's Complement Method

- Let $N = (a_{n-2} \ldots a_0)_2$
  - If $N \geq 0$, it is represented by $(0a_{n-2} \ldots a_0)_2$
  - If $N < 0$, it is represented by $[0a_{n-2} \ldots A_0]_1$
- ***1's complement*** $[N]_1 = 2^n - (N)_2 - 1 = [N]_2 - 1$ where n is the number of bits in $[N]_1$
- Examples ($n = 4$, $2^n = 10000$)
  - $+(110)_2 = (0110)_2$
  - $-(110)_2 = [0110]_1 = (10000\text{-}0110\text{-}1)_2 = (1001)_2$
- *Range* $2^{n-1} - 1 \geq N \geq -2^{n-1} + 1$ $(011\ldots11 \geq N \geq 100\ldots00)$
- *Pros and cons*
  - Easy to derive (bit-wise complement)
  - Harder to perform arithmetic operations (end-around carry correction) than two's complement
  - Two representations of 0
  - Same range as sign-magnitude
  - Not intuitive for humans

Two's Complement Method

- Let $N = (a_{n-2} \ldots a_0)_2$
  - If $N \geq 0$, it is represented by $(0a_{n-2} \ldots a_0)_2$
  - If $N < 0$, it is represented by $[0a_{n-2} \ldots a_0]_2$
- ***2's complement*** $[N]_2 = 2^n - (N)_2$, where n is the number of digits in $(N)_2$.
- Example ($n = 4$, $2^n = 10000$)
  - $+(110)_2 = (0110)_2$
  - $-(110)_2 = [0110]_2 = (10000\text{-}0110)_2 = (1010)_2$
- *Range* $2^{n-1} - 1 \geq N \geq -2^{n-1}$ $(011\ldots11 \geq N \geq 100\ldots00)$
- *Pros and Cons*
  - *Easy to implement in hardware*
  - *One representation of 0*
  - *Wider negative range*
  - *Not intuitive for humans*

Signed Binary Numbers

Decimal	Sign-Magnitude	Two's Complement	One's Complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
0	0000, 1000	0000	0000, 1111
-1	1001	1111	1110
-2	1010	1110	1101
-3	1011	1101	1100
-4	1100	1100	1011
-5	1101	1011	1010
-6	1110	1010	1001
-7	1111	1001	1000
-8	NA	1000	NA

Deriving the Two's Complement

- **Derive $[N]_2$ given $(N)_2$**
- **Use the definition** $[N]_2 = 2^n - (N)_2$, where n is the number of digits in $(N)_2$.
- ***Use Algorithm 1.4***
  - Copy the bits of N, beginning with the LSB and proceeding toward the MSB until the first 1 bit is reached.
  - Leave this 1 as is.
  - Replace each remaining digit a_j of N by its complement, i.e., replace each 1 with 0 and each 0 with 1.
- ***Example***: 2's complement of $(\mathbf{1010})_2$ is $(\mathbf{0110})_2$.

Deriving the Two's Complement

- **Derive $[N]_2$ given $(N)_2$**
- ***Algorithm 1.5***
  - Complement each digit of N and add 1
  - This follows from the equation $[N]_1 = 2^n - (N)_2 - 1 = [N]_2 - 1$, i.e., $[N]_2 = [N]_1 + 1$
- ***Example***: Derive the 2's complement of $N = (0100)_2$.

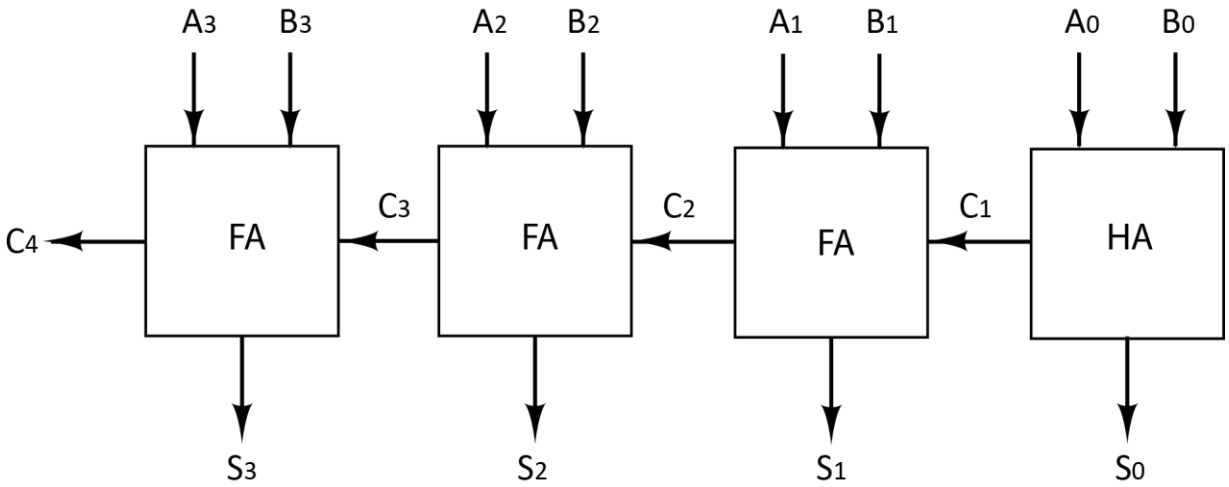
$N = 0100$

1011 complement the bits

+1 add 1

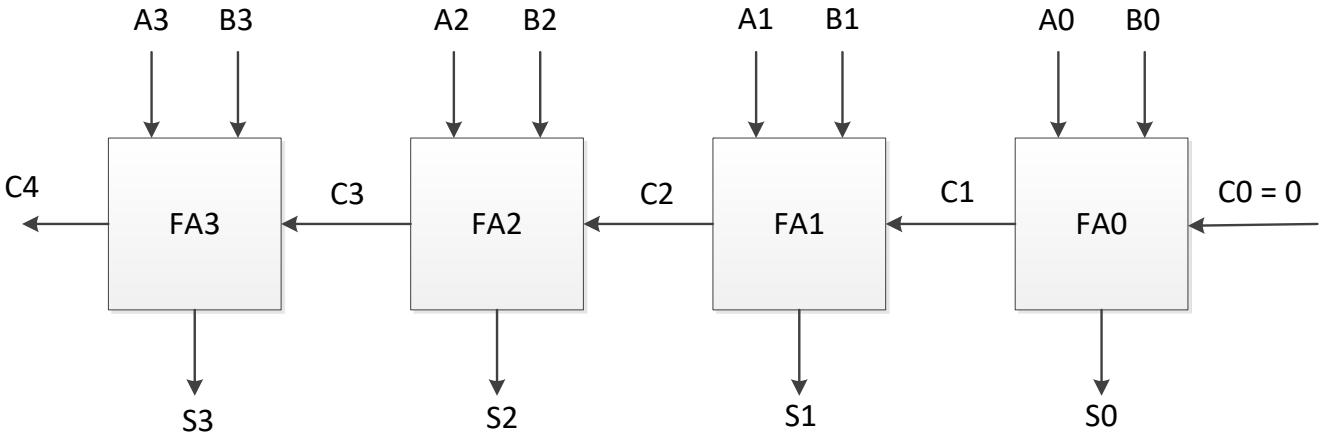
$[N]_2 = (1100)_2$

Ripple-Carry Adder (4-bit)



$$(C_4S_3S_2S_1S_0)_2 = (A_3A_2A_1A_0)_2 + (B_3B_2B_1B_0)_2$$

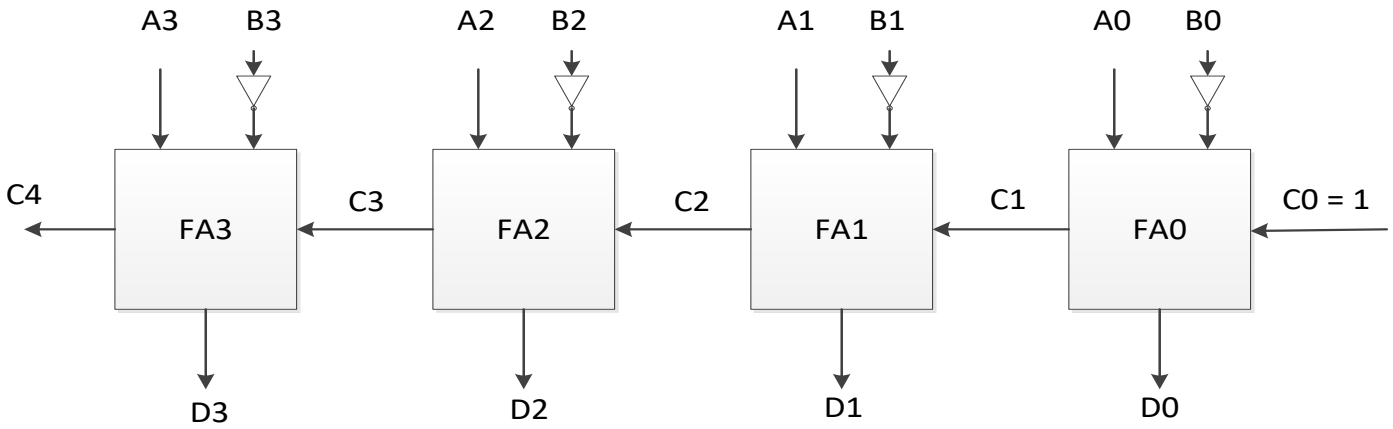
Ripple-Carry Adder



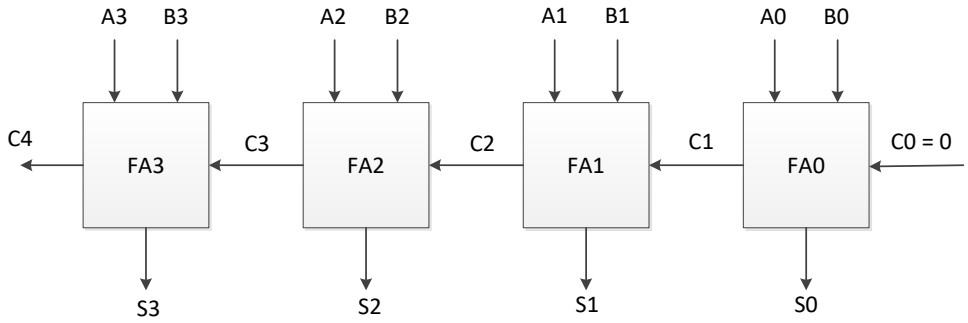
$$(C_4S_3S_2S_1S_0)_2 = (A_3A_2A_1A_0)_2 + (B_3B_2B_1B_0)_2$$

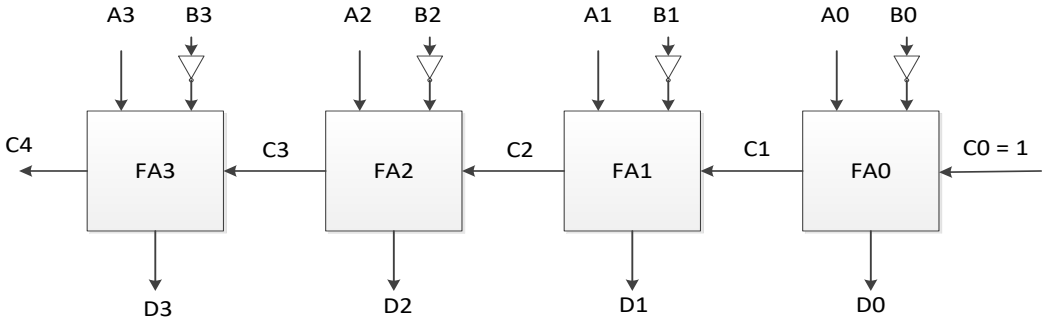
Using A Ripple-Carry Adder to Subtract

$$(C_4 D_3 D_2 D_1 D_0)_2 = (A_3 A_2 A_1 A_0)_2 - (B_3 B_2 B_1 B_0)_2$$
$$= (A_3 A_2 A_1 A_0)_2 + [B_3 B_2 B_1 B_0]_{2cns}$$
$$= (A_3 A_2 A_1 A_0)_2 + [B_3 B_2 B_1 B_0]_{1cns} + 1$$

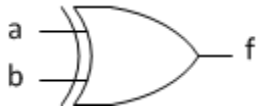


Ripple-Carry Adder/2's-Comp Subtractor





XOR-Gate Revisited

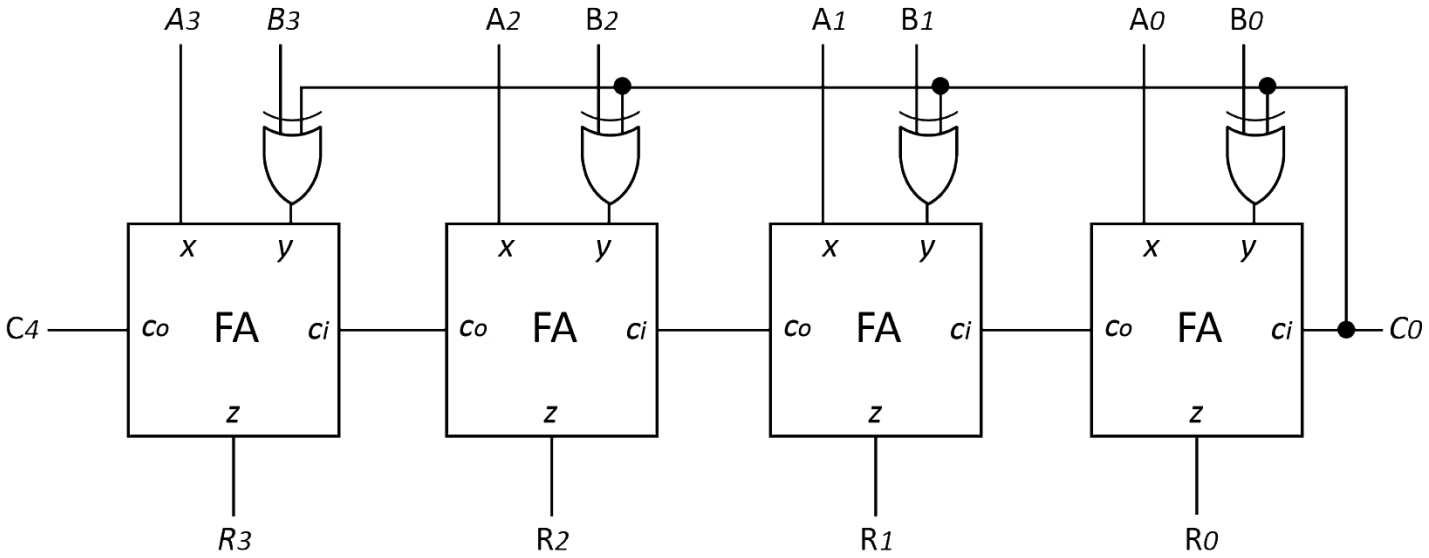


a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

If b = 0, then f = a.

If b = 1, then f = a'.

A Two's Complement Adder/Subtractor



If C0 = 0, then $(C_4R_3R_2R_1R_0)_2 = (A_3A_2A_1A_0)_2 + (B_3B_2B_1B_0)_2$. Therefore, R = A + B.

If C0 = 1, then $(C_4R_3R_2R_1R_0)_2 = (A_3A_2A_1A_0)_2 + [B_3B_2B_1B_0]_{1ns} + 1$. Therefore, R = A – B.

Ripple-Carry Adder/Subtractor Verilog Model

```
//Ripple Carry Adder Subtractor Structural Model
module RCAddSub (
    input AddSub,      //select add (AddSub=0) or subtract (AddSub=1) operation
    input [3:0] A, B,              //declare input ports
    output [3:0] S,                //declare output ports for sum
    output Cout);                  //declare carry-out port
    wire [3:0] Bb;                 //declare internal nets
    wire [4:0] C;
    //complement B input for subtraction
    assign Bb[3:0] = {AddSub^B[3],AddSub^B[2],AddSub^B[1],AddSub^B[0]};
    assign C[0] = AddSub;          //Add 0 for addition, 1 for subtraction
    assign Cout = C[4];            //rename carry out port
    //instantiate the full adder module for each stage of the ripple carry adder
    FAbehave s0 (S[0], C[1], A[0], Bb[0], C[0]);          //stage 0
    FAbehave s1 (S[1], C[2], A[1], Bb[1], C[1]);          //stage 1
    FAbehave s2 (S[2], C[3], A[2], Bb[2], C[2]);          //stage 2
    FAbehave s3 (S[3], C[4], A[3], Bb[3], C[3]);          //stage 3
endmodule
```

Two's Complement Arithmetic

- Two's complement number systems are used in computer systems since this reduces hardware requirements (only adders are needed).
  - $A + B = (A)_2 + (B)_2$
  - $A - B = (A)_2 + (-B)_2 = (A)_2 + [B]_2$ (add *2*'s complement of *B* to *A*)
- If the result of either operation falls outside the range, an ***overflow condition*** is said to occur and the result is not valid.
- Range of numbers in two's complement number system, where *n* is the number of bits.
  - $2^{n-1} - 1 = (0, 11 \ldots 1)_{2cns}$ and $-2^{n-1} = (1, 00 \ldots 0)_{2cns}$

Two's Complement Arithmetic

- ***Consider*** $A = B + C$ where B and C are two n-bit positive integers.
  - $(A)_2 = (B)_2 + (C)_2$
  - If $A > 2^{n-1} - 1$ (***overflow***), it is detected by the nth bit, which is set to 1.
- ***Example***: $(2)_{10} + (4)_{10} = ?$ using 4-bit two's complement arithmetic.
  - $+ (2)_{10} = +(010)_2 = (0010)_{2cns}$
  - $+ (4)_{10} = +(100)_2 = (0100)_{2cns}$
  - $(0010)_{2cns} + (0100)_{2cns} = (0110)_{2cns} = +(110)_2 = +(6)_{10}$
  - No overflow.

Two's Complement Arithmetic

- ***Example***: $(5)_{10} + (4)_{10} = ?$
  - $+ (5)_{10} = +(101)_2 = (0101)_{2cns}$
  - $+ (4)_{10} = +(100)_2 = (0100)_{2cns}$
  - $(\mathbf{0}101)_{2cns} + (\mathbf{0}100)_{2cns} = (\mathbf{1}001)_{2cns}$ (***overflow has occurred***)

Two's Complement Arithmetic

- ***Consider*** $A = B - C$ where B and C are two n-bit positive integers.
- $A = (B)_2 + (-(C)_2) = (B)_2 + [C]_2 = (B)_2 + 2^n - (C)_2 = 2^n + (B - C)_2$
  - Overflow cannot occur for this case
  - If $B \geq C$, then $A \geq 2^n$, discarding the carry, 2^n, leaves $(B - C)_2$.
  - So, $(A)_2 = (B)_2 + [C]_2|_{\text{carry discarded}}$
  - If $B < C$, then $A = 2^n - (C - B)_2 = [C - B]_2$ or $A = -(C - B)_2$ (no carry in this case).

- ***Example***: $(7)_{10} - (2)_{10} = ?$
  - Perform $(7)_{10} + (-(2)_{10})$
  - $(7)_{10} = +(111)_2 = (0111)_{2cns}$
  - $-(2)_{10} = -(010)_2 = (1110)_{2cns}$
  - $(7)_{10} - (2)_{10} = (0111)_{2cns} + (1110)_{2cns} = (0101)_{2cns} + \cancel{carry}$
    $= +(101)_2 = +(5)_{10}$

Two's Complement Arithmetic

- ***Example***: $(2)_{10} - (3)_{10} = ?$
  - Perform $(2)_{10} + (-(3)_{10})$
  - $(2)_{10} = +(10)_2 = (0010)_{2cns}$
  - $-(3)_{10} = -(11)_2 = (1101)_{2cns}$
  - $(2)_{10} - (3)_{10} = (0010)_{2cns} + (1101)_{2cns} = (1111)_{2cns}$
    $= -(0001)_2 = -(1)_{10}$

Two's Complement Arithmetic

- ***Consider*** $A = -B - C$ where B and C are *n*-bit positive integers
  - $A = [B]_2 + [C]_2 = 2^n - (B)_2 + 2^n - (C)_2 = 2^n + 2^n - (B + C)_2 = 2^n - (B + C)_2 = [B+C]_2$
  - An overflow can occur, and is indicated by a sign bit of 0.
  - The carry bit (2^n) is discarded leaving the 2's complement of B+C.
- ***Example***: $-(3)_{10} - (4)_{10} = ?$
  - Perform $(-(3)_{10}) + (-(4)_{10})$
  - $-(3)_{10} = -(11)_2 = (1101)_{2cns}$, $-(4)_{10} = -(100)_2 = (1100)_{2cns}$
  - $-(3)_{10} - (4)_{10} = (1101)_{2cns} + (1100)_{2cns} = (1001)_{2cns} + \cancel{carry}$
    $= -(0111)_2 = -(7)_{10}$
- ***Example***: $-(4)_{10} - (5)_{10} = ?$
  - Perform $(-(4)_{10}) + (-(5)_{10})$
  - $-(4)_{10} = -(100)_2 = (1100)_{2cns}$, $-(5)_{10} = -(101)_2 = (1011)_{2cns}$
  - $-(4)_{10} - (5)_{10} = (\mathbf{1}1100)_{2cns} + (\mathbf{1}1011)_{2cns} = (\mathbf{0}0111)_{2cns} + \cancel{carry}$
  - ***An overflow is indicated by the sign bit of 0.***

Two's Complement Arithmetic

Case Summary

Case	*Carry*	*Sign of A*	*Condition*	*Overflow?*
$A = B + C$	0	0	$(B + C) \leq 2^{n-1} - 1$	No
	0	1	$(B + C) > 2^{n-1} - 1$	Yes
$A = B - C$	1	0	$B \leq C$	No
	0	1	$B > C$	No
$A = -B - C$	1	1	$-(B + C) \geq -2^{n-1}$	No
	1	0	$-(B + C) < -2^{n-1}$	Yes

Where *B* and *C* are *n*-bit positive integers.

Two's Complement Arithmetic

Overflow Detection for Addition

Operation	*Sign of X*	*Sign of Y*	*Sign of Z*	*Overflow?*
$Z = X + Y$	0	0	0	No
	0	**0**	**1**	**Yes**
	0	1	0	No
	0	1	1	No
	1	0	0	No
	1	0	1	No
	1	**1**	**0**	**Yes**
	1	1	1	No

Design an Overflow Detection Circuit

Recall

(+) + (+) = (+): **0**110 + **0**001 = **0**111 √

: **0**110 + **0**010 = **1**000 × **OVERFLOW**

(-) + (-) = (-): **1**010 + **1**111 = ~~1~~ **1**001 √

: **1**010 + **1**110 = ~~1~~ **1**000 √

: **1**010 + **1**101 = ~~1~~ **0**111 × **OVERFLOW**

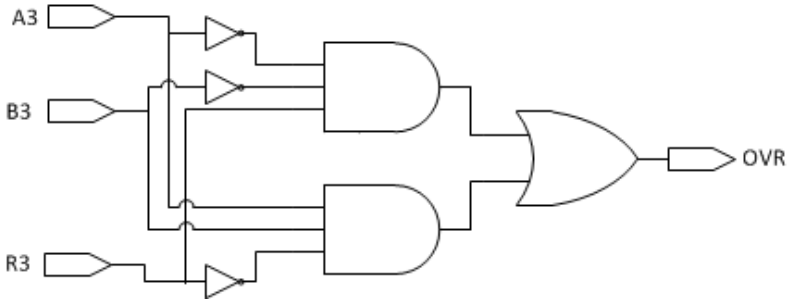
Design an Overflow Detection Circuit

So $(A_3A_2A_1A_0) + (B_3B_2B_1B_0) = C_4R_3R_2R_1R_0$

0111 + 0001 = 1000 OVERFLOW

1111 + 1000 = ~~1~~0111 OVERFLOW

R_3 \ A_3B_3	00	01	11	10
0			1	
1	1			



$OVR = \bar{A}_3\bar{B}_3R_3 + A_3B_3\bar{R}_3$

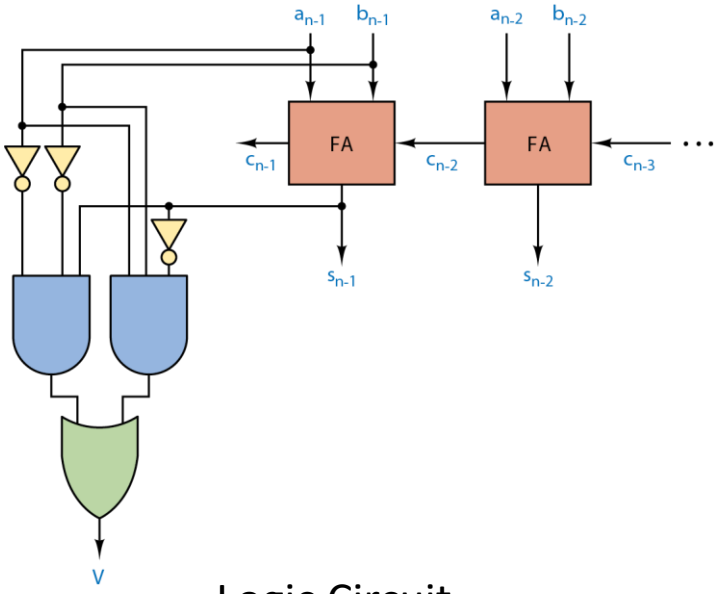
Two's Complement Overflow Detection

Adder Inputs			Adder Outputs		Overflow
a_{n-1}	b_{n-1}	c_{n-2}	c_{n-1}	s_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Truth Table

$$V = \overline{a}_{n-1}\overline{b}_{n-1}s_{n-1} + a_{n-1}b_{n-1}\overline{s}_{n-1}$$

Logic Equation



Logic Circuit

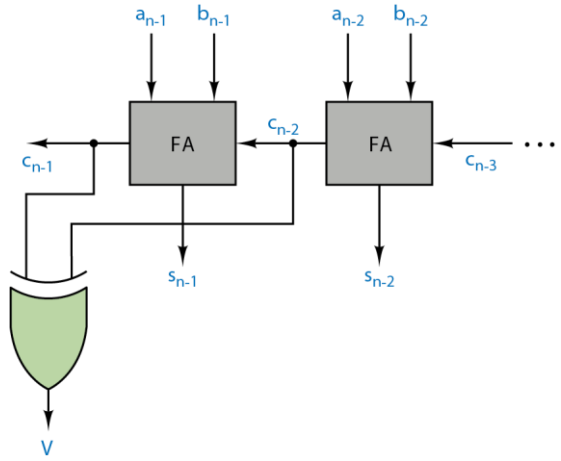
Two's Complement Overflow Detection

Adder Inputs			Adder Outputs		Overflow
a_{n-1}	b_{n-1}	c_{n-2}	c_{n-1}	s_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Truth Table

$$V = c_{n-2} \oplus c_{n-1}$$

Logic Equation

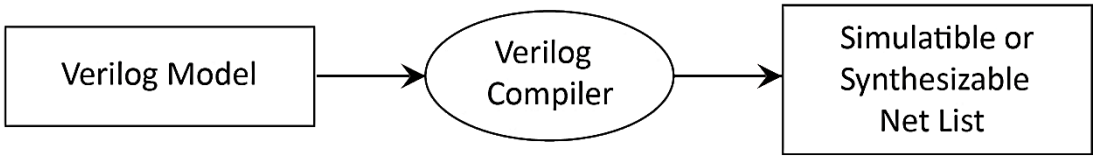


Logic Circuit

The Verilog HDL

- Verilog is a widely used Hardware Description Language (HDL)
- C-like syntax, but not a programming language

(a) C Programming Language Compilation



(b) Verilog Hardware Description Language Compilation

- Used to capture existing designs for simulation purposes – design verification or analysis
- Used to capture new designs for synthesis or analysis

When to Use Verilog Models

- In general – anytime you want to realize a design with programmable logic (ASIC, CPLD, FPGA, SoC)
- Structural – capture design (structure) of an existing circuit
  - Simulation
  - Synthesis
- Dataflow – capture functionality of simple combinational logic
  - Simulation
  - Synthesis
- Behavioral – capture functionality (behavior) of combinational and/or sequential logic
  - Simulation
  - Synthesis

Verilog Module Structure

module *name (port list);*

[port declarations]

[data type declarations]

[functionality statements]

[timing specifications]

endmodule

Case Statements

case *(expression)*

condition1: statement1;

condition2: statement2;

condition3: statement3;

…

…

default*: default_statement;*

endcase

Data Types

- *Net* datatypes – represent physical connections, must be driven, value can change at any time, declared by

  wire d,e,f,g;

  tri x,y,z;

- *Variable* datatypes – abstractions of storage elements, hold values from one assignment to the next, declared by

  reg si, couti; //unsigned, length specified

  integer I; //signed 32-bit variable

  time t; //unsigned integer

- Outputs of behavioral models must be declared as variable datatypes.

Number Representation

- General format

  <*size*><*base*><*value*>

  3’b101 – three-bit binary number 101

  8’d101 – eight-bit decimal number 101

  8’h9e – eight-bit hexadecimal number 9E

Procedural Blocks

- Procedural blocks – represent processes and execute concurrently, may be named.
- **initial** Blocks – execute once, typically used to initialize variables in a test bench.
- **always** Blocks – execute continuously, used to represent the functionality of hardware.

Assignment Statements

- Continuous assignment statements -- execute continuously, used outside of procedural blocks.

  assign si **=** ai **^** bi **^** cini;

- Procedural assignment statements – used to derive or update variable datatypes, LHS must be a variable datatype, RHS may be any valid Verilog statement, used inside procedural blocks.

  si = 1'b0;

  begin si = 1'b0; couti = 1'b0; **end**

Concatenation

- Concatenation symbol – {}

  if ai = 0, bi = 1, cini = 0, then {a1,bi,cini} = 010

Events

- Event – a change in the value of a variable or on a net, may be used to control execution of *always blocks*.

@ *expression* where the expression defines the event

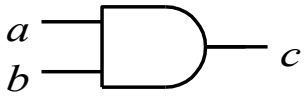
```
always @ (ai,bi,cini);
```

if … else Statements

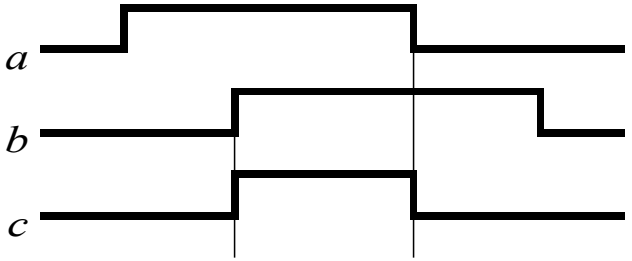
```
if (condition1) statement1;
        else if (condition2) statement2;
        else if (condition3) statement3;
                ...
                ...
        else defaultstatement;
```

Propagation Delay

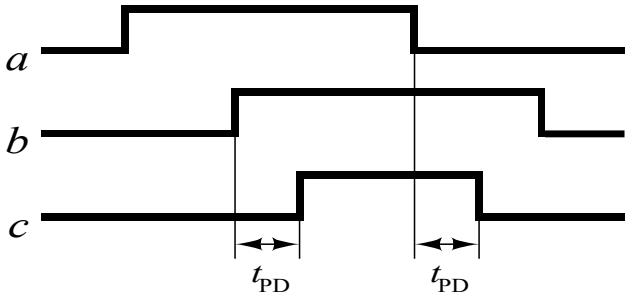
- Propagation delay through a logic gate



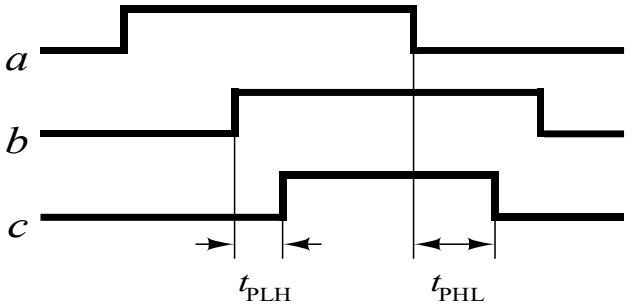
(a) Two-input AND gate



(b) Ideal (zero) delay



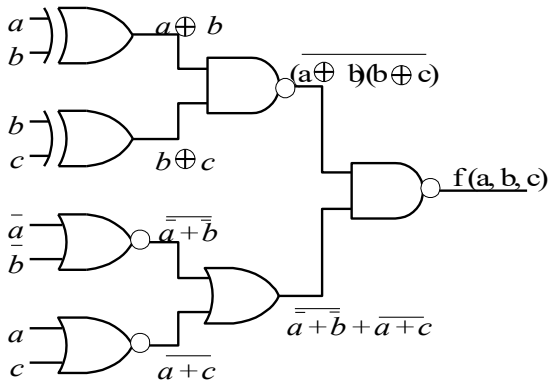
(c) $t_{PD} = t_{PLH} = t_{PHL}$



(d) $t_{PLH} < t_{PHL}$

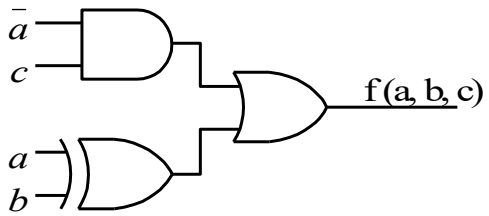
Circuit Propagation Delay

Let t_g represent the propagation delay for all basic gates and t_f the circuit delay.



Given circuit

$t_f = 3t_g$



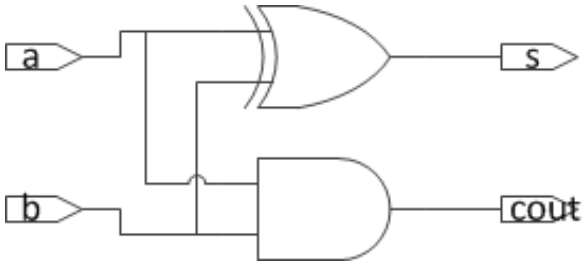
Simplified circuit

$t_f = 2t_g$

Circuit delay = sum of the gate delays through the longest path from an input to an output.

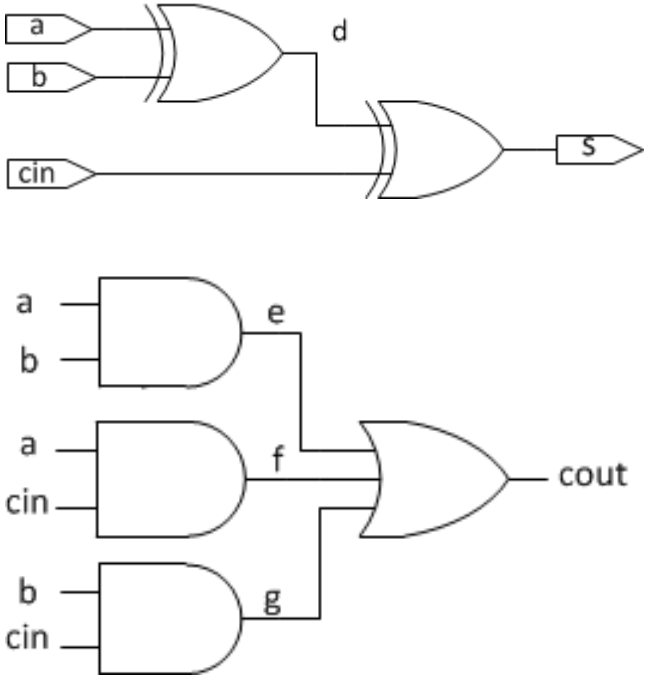
Add Times for Basic Adders

Half-Adder



t_g

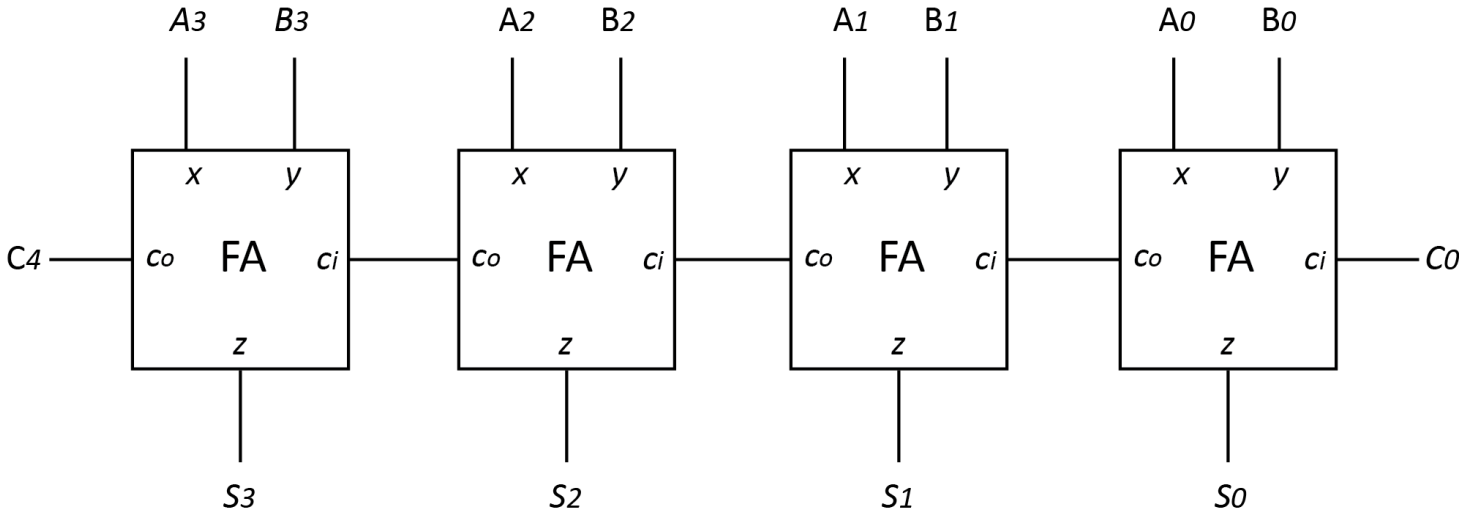
Full-Adder



$2t_g$

Add Time for Ripple-Carry Adders

Assume all inputs are applied simultaneously and $t_{fa} = 2t_g$.



$$2t_g + 2t_g + 2t_g + 2t_g = 4(2t_g)$$

For an n-bit ripple-carry adder, the add time is $2nt_g$.

Add Time Implications

$$f_{MAX} = 1/t_{add} = 1/2nt_g$$

t_g \ n	16	32	64	128
10 ns	320 ns	640 ns	1,280 ns	2,560 ns
	3.125 MHz	1.5625 MHz	0.78125 MHz	0.390625 MHz
1 ns	32 ns	64 ns	128 ns	256 ns
	6.25 MHz	3.125 MHz	1.5625 MHz	0.78125 MHz
0.1 ns	3.2 ns	6.4 ns	12.8 ns	25.6 ns
	12.5 MHz	6.25 MHz	3.125 MHz	1.5625 MHz
0.01 ns	0.32 ns	0.64 ns	1.28 ns	2.56 ns
	25 MHz	12.5 MHz	6.25 MHz	3.125 MHz
0.001 ns	0.032 ns	0.064 ns	0.128 ns	0.256 ns
	50 MHz	25 MHz	12.5 MHz	6.25 MHz

t_{add}, f_{MAX}

How Fast Do We Need to Add?

- Assume one add operation per clock cycle, $f = 1/t_{add}$
- Then $t_{add\text{-}max} = 1/f$

f (GHz)	1.5	1.8	2.0	2.4	2.8	3.0	3.6	3.9
$t_{add\text{-}max}$ (ps)	666	555	500	417	357	333	277	256

Q & A