

Course Kickoff

(Class 1.1 – 1/19/2021)

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

- Syllabus overview
 - Course learning objectives
 - Course organization and topics
 - Course grading
 - Technology requirements
 - Textbook
 - Reference materials
- Course resources
 - DE1-SoC lab kits
 - Lab kit check out
 - Quartus Prime Lite (version 18.1) software
 - CpE labs (ERB 126 and 127)
- FPGAs and SoCs
 - Technology
 - Design flow
 - HDLs
- Review
 - Combinational logic
 - Verilog



Syllabus Overview

Instructor: Bill Carroll, Ph.D., P.E.

Office Number: ERB 521

Office Telephone Number: +1 817-272-3785 (CSE Department)

Email Address: carroll@uta.edu

Faculty Profile: <https://mentis.uta.edu/explore/profile/bill-carroll>

Office Hours: E-mail and virtual Q&A sessions on Teams.

MTuWTh 3:30 to 5:30 PM, or by appointment or chance.

Communication:

My primary communication link is email carroll@uta.edu. Course materials will be posted on Canvas.

Time and Place of Class Meetings:

Synchronous online in Teams (TTh 5:30 to 6:50 PM). Recordings posted on Canvas.

Exams: On campus, February 23, April 6, 2-3:20pm, locations TBA.

Laboratory: ERB 126 and 127 can be used for testing of assignments if necessary. COVID-19 occupancy policies will be in effect. Students in scheduled labs have precedence for seats.

Students will checkout a DE1-SoC board for the semester.



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Syllabus Overview

Textbook and Other Course Materials:

Required:

1. Michael D. Ciletti, *Advanced Digital Design with the Verilog HDL, Second Edition*. New York: Pearson, 2011.

Recommended:

2. Sunggu Lee, *Advanced Digital Logic Design*. Toronto: Nelson, 2006.

3. Victor P. Nelson, Bill D. Carroll, H. Troy Nagle, and J. David Irwin, *Digital Logic Circuit Analysis and Design, Second Edition*. New York: Pearson, 2021. Pearson eText, ISBN 9780135305706 or 9780135297070.

Major Assignments and Examinations:

Labs: Various assignments will be made during the semester.

Exam 1: Tuesday, February 23

Exam 2: Tuesday, April 6

Project: Tuesday, May 4



Course Content and Objectives

CSE 4357/5357 Advanced Digital Logic Design -- Hierarchical organization, design, simulation, implementation, and testing of digital systems. Industrial standard computer-aided design tools including hardware description languages (HDLs), field-programmable gate arrays (FPGAs), and other prototyping hardware and software will be employed. Design of arithmetic and other algorithmic processes will be covered. A term project will be required. Prerequisite: C or better in [CSE 3442](#).

COURSE OBJECTIVES -- You will learn the concepts, methods, and technologies needed to analyze, specify, design, build, and test advanced digital logic circuits using programmable logic devices. You will use your knowledge and skills to design the components of a high-speed, four-function (add, subtract, multiply, and divide), fixed-point arithmetic logic unit (ALU). Finally, you will complete a term project that integrates the components to realize the ALU on a DE1-SoC development board.



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Organization and Topics*

1/19,21: Course overview, programmable logic, review of combinational logic. Review of Verilog.
1/26,28: Review of Quartus Prime 18.1. High-speed adders.
2/2,4: Review of sequential logic. Review of Verilog models of sequential machines.
2/9,11: Test benches. ModelSim.
2/16,18: FPGA organization and architecture. Quartus Prime processes.
2/23: **Examination 1.**
2/25: Quartus Prime analysis tools.
3/2,4: Basic multipliers.
3/9,11: High-speed multipliers.
3/16,18: *Spring Break.*
3/23,25: Pipelined functional units.
3/30,4/1: Dividers.
4/6: **Examination 2.**
4/8: Post synthesis design tasks.
4/13,15: Timing analysis.
4/20,22: ALU design.
4/27,29: Project work.
5/4: Project work.

*** Subject to change as necessary to maintain effective pedagogical pace and topical coverage.**



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Grading

Grading:

- Grade scale: A (90-100), B (80-89), C (70-79), D (60-69), and F (0-59)
- Grade calculation: Exam 1 (25%), Exam 2 (25%), Assignments (25%), Term Project (25%)

Students not completing one or more of these requirements may receive an Incomplete (I) or F in the course. Students not completing the term project will receive a D or an F in the course. Late assignments will generally not be accepted.

Exams:

- Exams will be open-book, open-notes, calculators allowed.

Assignments:

- Assignments are individual in nature. Discussing assignment topics is allowed, but the submissions must be unique. Sharing of designs is not allowed.

Make-up Exams and Assignments

Make up of missed examinations and assignments will be handled case-by-case and, generally, be approved only if sufficient justification can be made and documented. Requests for make-up must be made to the instructor within one week of the missed work's due date.

Project:

- Design, implement, test, demonstrate, and document a high-speed, four-function, fixed-point ALU.
- Due date: May 4, 2021, 11:59 PM.



UNIVERSITY OF
TEXAS
ARLINGTON

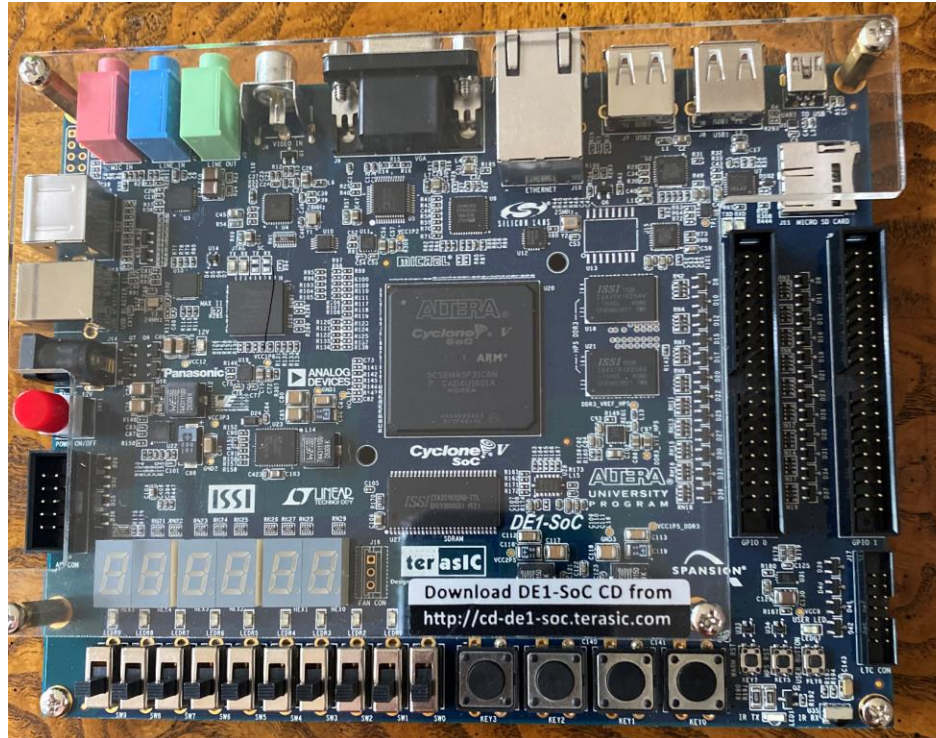
DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Technology Requirements

- MS Teams
- Canvas
- Webcam
- Camera for photo and video documentation
- Camera stand
- Printer
- Scanner
- Laptop for running Windows applications
- Intel Quartus Prime Lite, version 18.1, CAD SW
<https://fpgasoftware.intel.com/18.1/?edition=lite>
- ModelSim – Intel FPGA Starter Edition 10.5b



DE1-SoC Lab Kits



FPGA side

- Intel Cyclone V SE 5CSEMA5F31C6
- 64MB SDRAM (16-bit data bus)
- And much more

HPS side

- 800MHz Dual-core ARM Cortex-A9
- 1GB DDR3 SDRAM (32-bit data bus)
- And much more



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Lab Equipment Check Out

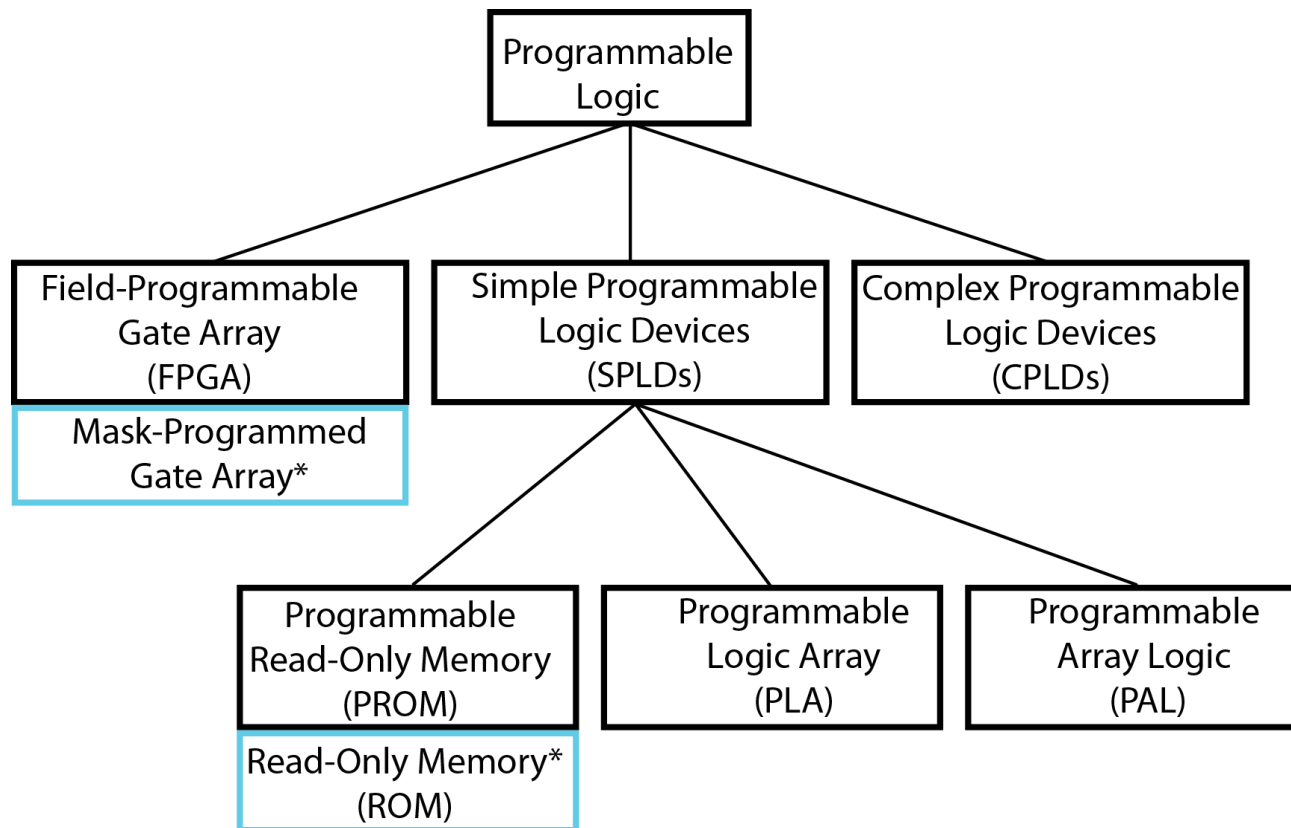
- Check-out date: Friday, January 22, 2021, 1-2pm, ERB 127.
- Return date: Wednesday, May 5, 2021, 1-2pm, ERB 127.
- Please printout, complete, and sign the checkout form prior to arriving at the lab. Please leave the kit number blank. You will receive the signed form back when you return the kit at the end of the semester as your return receipt.
- You will checkout only a DE1-SoC kit at this time.
- The kit must be returned by date above or an incomplete grade will be assigned.
- Failure to return the kit with all components in good condition (except normal wear-and-tear) can result in your being charged up to \$249 to replace the missing or damaged components.



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Taxonomy of Programmable Devices



* Factory Programmed



UNIVERSITY OF
TEXAS
ARLINGTON

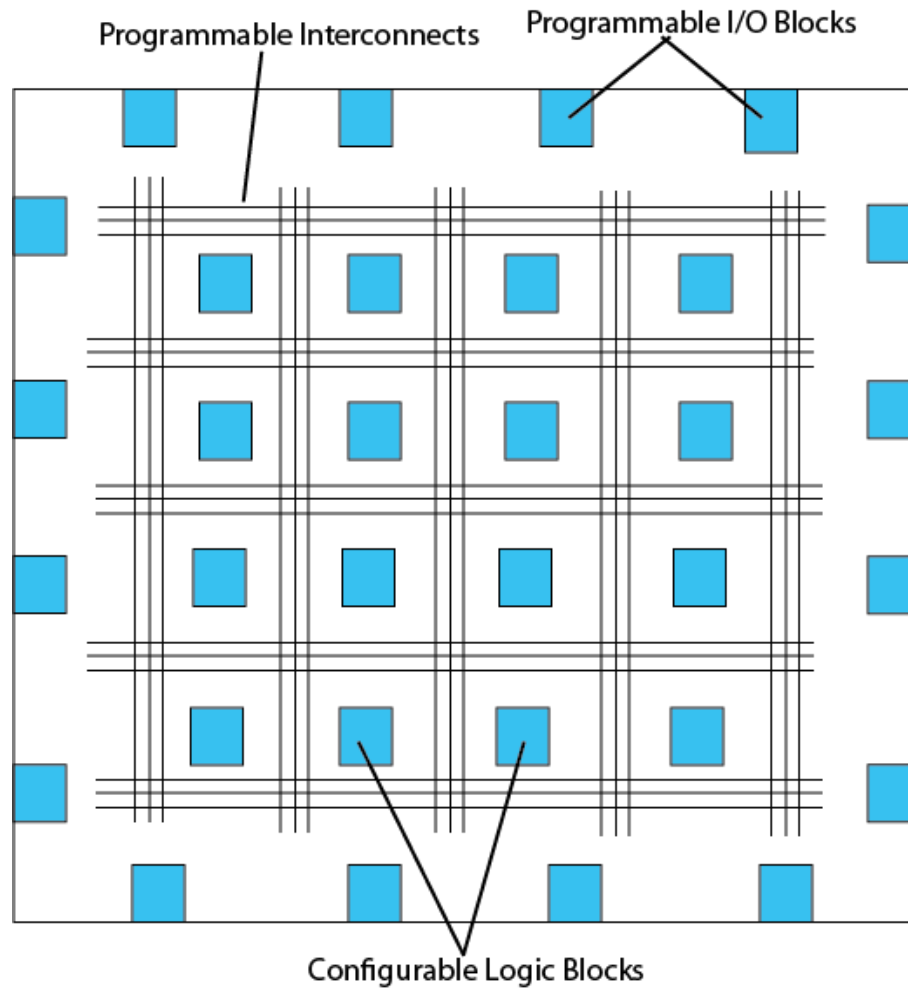
DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Other Programmable “Logic” Devices

- System on Chip (SoC)
 - FPGA + analog I/O (A to D, D to A) + processor
 - Example devices – Intel Arria 10, Xilinx Zynq 7000, Cypress PSoC 4, Intel Cyclone V-SoC
- Micro controller – small computer (processor, memory, digital I/O, analog I/O) on a chip.
 - Embedded system applications
 - Example devices – MT PIC32, Freescale ColdFire, TI C2000, Intel 8051



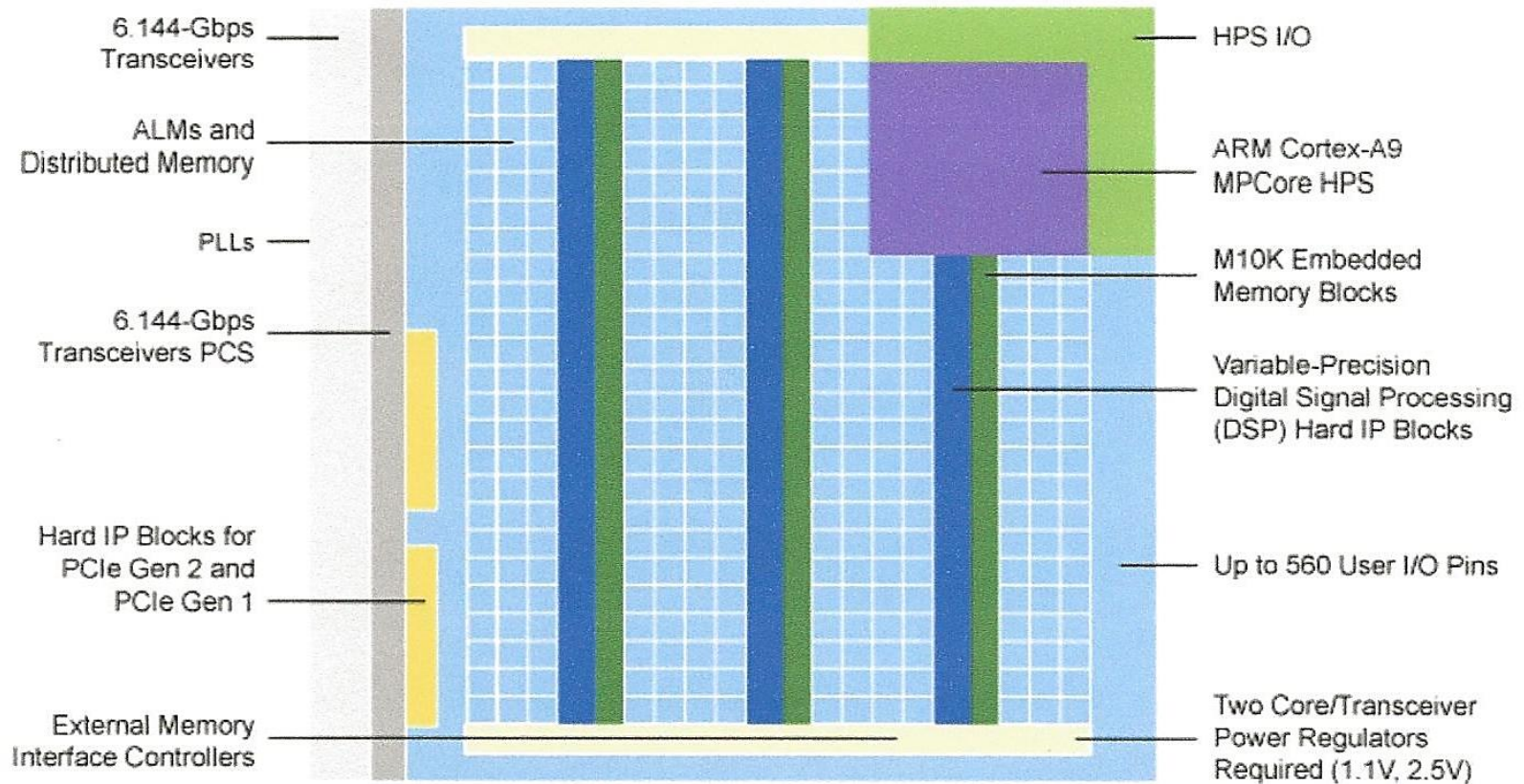
FPGA Logic Array Structure



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Cyclone V FPGA Architecture

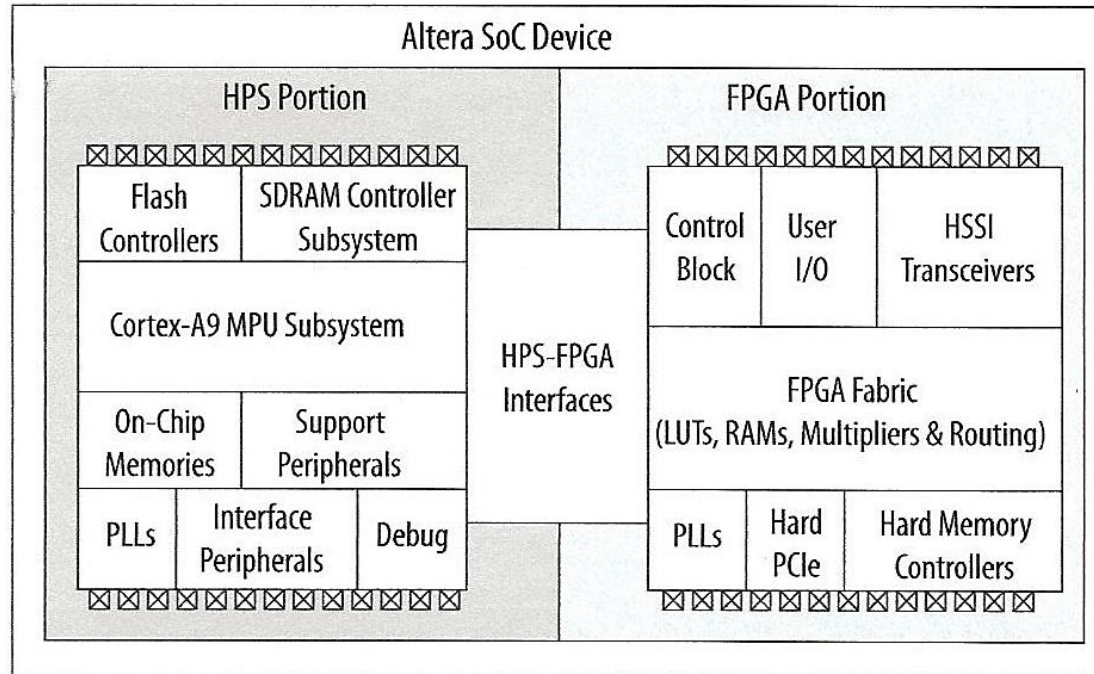


UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

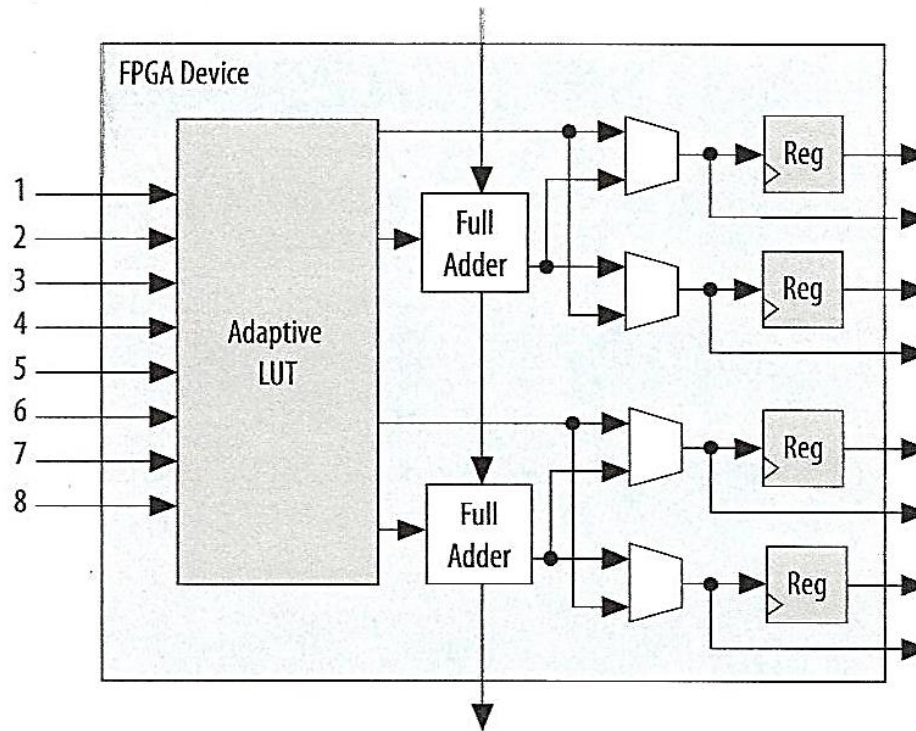
Cyclone V FPGA Architecture

Intel SoC Device Block Diagram



Cyclone V FPGA Architecture

ALM for Cyclone V Devices



FPGA Design Flow

Front End

1. Determine the required *functionality* of the digital system. This can be expressed in the form of truth tables for combinational logic, state diagrams for sequential logic, and/or RTL designs for more complex systems.
2. *Model* the system behavior with an HDL (VHDL, Verilog, or other design language supported by the CAD tools.)
3. Conduct a *behavioral simulation* of the HDL model to verify that it produces correct outputs and states for all expected input sequences.

Back End

4. Implement the design for the desired target FPGA using the vendor's CAD tools. This typically requires the following steps.
 - a. *Synthesize* a digital circuit from the HDL model.
 - b. *Map* the digital circuit onto primitive components (LUTs, flip-flops, etc.) of the target FPGA.
 - c. *Place* the primitive components of the circuit by assigning each to a specific component within the array of FPGA resources.
 - d. *Route* interconnections between the placed components and the external pins.
5. Conduct a *timing simulation* of the post place and route circuit to verify its functionality and timing characteristics. The latter is possible since this circuit comprises technology-specific primitive components for the target FPGA that includes timing parameters.
6. Generate a *configuration file*, i.e. a set of configuration bits to program each primitive component and interconnection in the FPGA.
7. *Download* the configuration file to the FPGA to implement and test the design.



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Quartus Prime software

- Download and install from the Intel FPGA website
 - Quartus Prime Lite, version 18.1
 - ModelSim – Intel FPGA Starter Edition 10.5b
- You'll have to register with Intel before you can download but it's free and can be done the first time you access the site.
<https://fpgasoftware.intel.com/18.1/?edition=lite>
- Quartus Prime is large, so a fast connection is recommended.
- You'll also need to install the USB-Blaster-II driver on your computer in order to connect to the DE1-SoC. This can be done during the above software install.
- You will need this software to complete your lab exercises this semester.

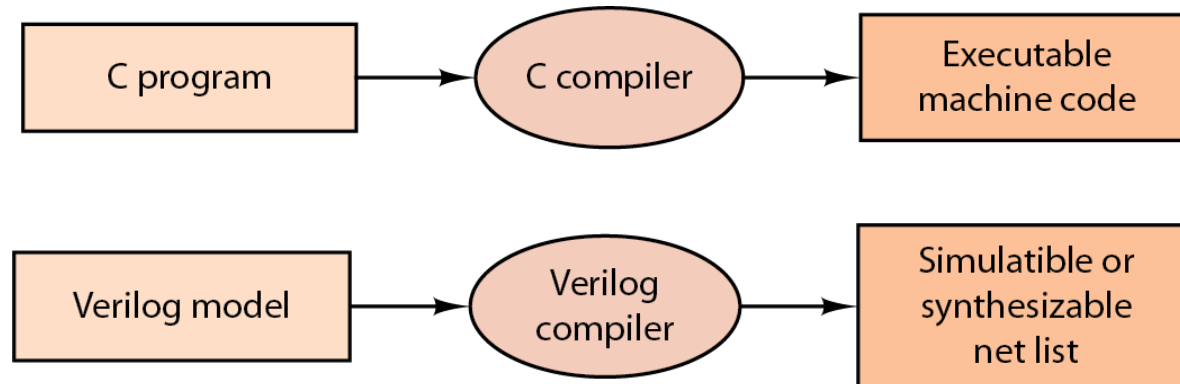


UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Hardware Description Languages (HDLs)

- Verilog – C-like syntax



- VHDL – Ada-like syntax
- Others



REVIEW OF COMBINATIONAL LOGIC AND VERILOG



UNIVERSITY OF
TEXAS
ARLINGTON

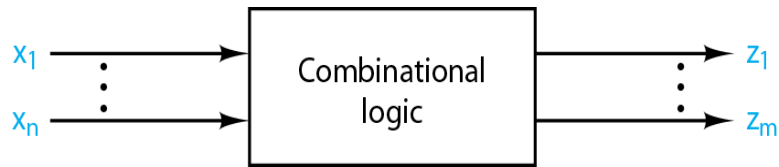
DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Digital Logic Circuit Taxonomy

- Combinational Circuits
 - Primary characteristic – memoryless
 - Primary building blocks – logic gates
- Sequential circuits
 - Primary characteristic – memory
 - Primary building blocks – logic gates, flip-flops
 - Types
 - Synchronous (clocked)
 - Asynchronous (unclocked)

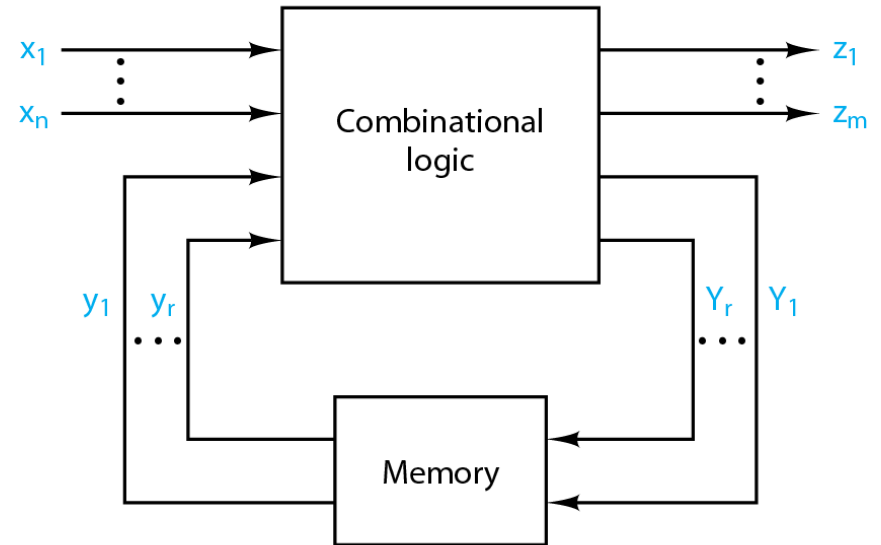


Logic Circuit Models



$$z_i = f_i(x_1, \dots, x_n) \text{ for } i = 1, \dots, m$$

(a) Combinational Logic Model



$$z_i = g_i(x_1, \dots, x_n, y_1, \dots, y_r) \text{ for } i = 1, \dots, m$$

$$Y_i = h_i(x_1, \dots, x_n, y_1, \dots, y_r) \text{ for } i = 1, \dots, r$$

(b) Sequential Logic Model




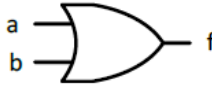





Boolean Algebra Postulates and Theorems

(Table 2.2 of *Nelson, et.al.*)

Expression	Dual
P2(a): $a + 0 = a$	P2(b): $a \cdot 1 = a$
P3(a): $a + b = b + a$	P3(b): $a \cdot b = b \cdot a$
P4(a): $a + (b + c) = (a + b) + c$	P4(b): $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
P5(a): $a + bc = (a + b)(a + c)$	P5(b): $a \cdot (b + c) = a \cdot b + a \cdot c$
P6(a): $a + \bar{a} = 1$	P6(b): $a \cdot \bar{a} = 0$
T1(a): $a + a = a$	T1(b): $a \cdot a = a$
T2(a): $a + 1 = 1$	T2(b): $a \cdot 0 = 0$
T3(a): $\overline{(\bar{a})} = a$	T3(b): $\overline{(\bar{a})} = a$
T4(a): $a + ab = a$	T4(b): $a(a + b) = a$
T5(a): $a + \bar{a}b = a + b$	T5(b): $a(\bar{a} + b) = a \cdot b$
T6(a): $ab + a\bar{b} = a$	T6(b): $(a + b)(a + \bar{b}) = a$
T7(a): $ab + \bar{a}c + bc = ab + \bar{a}c$	T7(b): $(a + b)(\bar{a} + c)(b + c) = (a + b)(\bar{a} + c)$
T8(a): $\overline{(\bar{a} + \bar{b})} = \bar{a}\bar{b}$	T8(b): $\overline{(\bar{a}\bar{b})} = \bar{a} + \bar{b}$



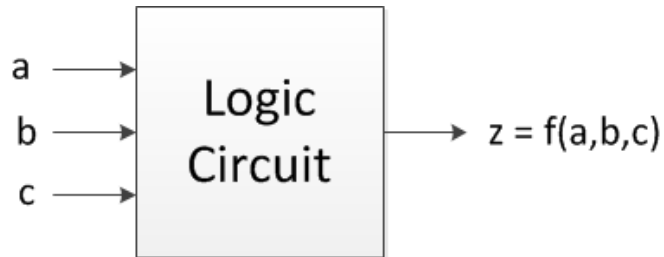
Basic Logic Gates

Gate Type	Logic Symbol	Truth Table	Boolean Equation	Verilog Statement	IC Part Number															
AND		<table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	a	b	f	0	0	0	0	1	0	1	0	0	1	1	1	$f = ab$	and (f,a,b)	SN7408
a	b	f																		
0	0	0																		
0	1	0																		
1	0	0																		
1	1	1																		
OR		<table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	a	b	f	0	0	0	0	1	1	1	0	1	1	1	1	$f = a + b$	or (f,a,b)	SN7432
a	b	f																		
0	0	0																		
0	1	1																		
1	0	1																		
1	1	1																		
NOT		<table><tr><th>a</th><th>f</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	a	f	0	1	1	0	$f = a'$	not (f,a)	SN7404									
a	f																			
0	1																			
1	0																			
NAND		<table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	f	0	0	1	0	1	1	1	0	1	1	1	0	$f = (ab)'$	nand (f,a,b)	SN7400
a	b	f																		
0	0	1																		
0	1	1																		
1	0	1																		
1	1	0																		
NOR		<table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	f	0	0	1	0	1	0	1	0	0	1	1	0	$f = (a + b)'$	nor (f,a,b)	SN7402
a	b	f																		
0	0	1																		
0	1	0																		
1	0	0																		
1	1	0																		
XOR		<table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	f	0	0	0	0	1	1	1	0	1	1	1	0	$f = a \oplus b$	xor (f,a,b)	SN7486
a	b	f																		
0	0	0																		
0	1	1																		
1	0	1																		
1	1	0																		
XNOR		<table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	a	b	f	0	0	1	0	1	0	1	0	0	1	1	1	$f = (a \oplus b)'$	xnor (f,a,b)	None
a	b	f																		
0	0	1																		
0	1	0																		
1	0	0																		
1	1	1																		



Combinational Logic Circuits and Truth Tables

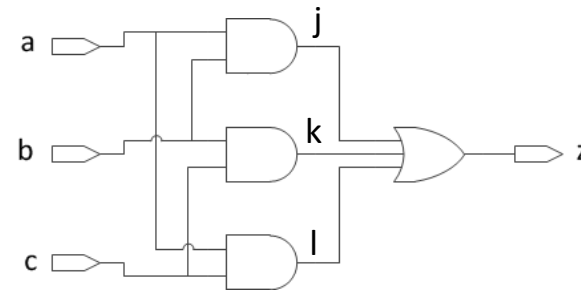
Block Diagram



Truth Table

a	b	c	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Circuit Diagram



Logic Equation

$$z = ab + ac + bc$$

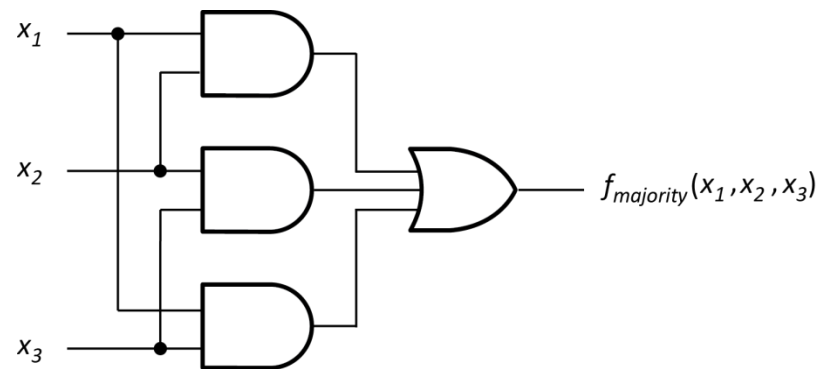
Verilog HDL Code

```
module LogicCircuit (a,b,c,z);  
  input a,b,c;  
  output z;  
  wire j,k,l;  
  and (j,a,b);  
  and (k,b,c);  
  and (l,a,c);  
  or (z,j,k,l);  
endmodule
```



Majority Functions

x_3	x_2	x_1	f_{majority}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

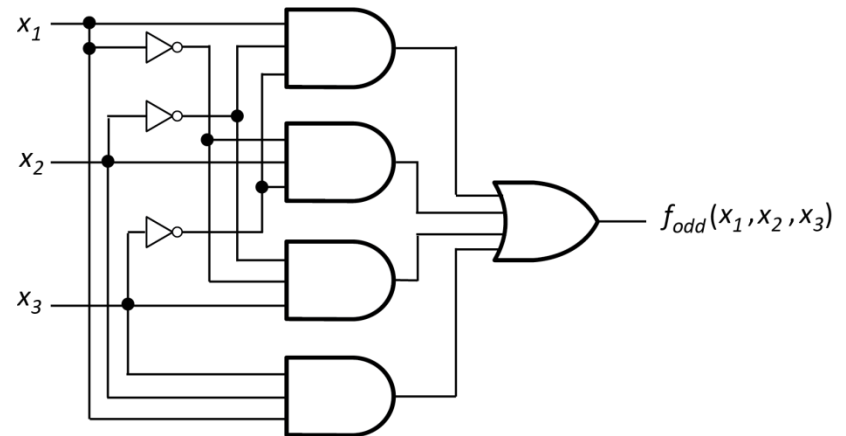


$$f_{\text{majority}}(x_1, x_2, x_3) = \bar{x}_3 x_2 x_1 + x_3 \bar{x}_2 x_1 + x_3 x_2 \bar{x}_1 + x_3 x_2 x_1 = x_1 x_2 + x_1 x_3 + x_2 x_3$$



Parity Functions

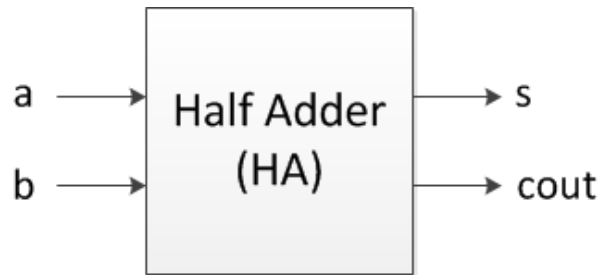
x_3	x_2	x_1	f_{odd}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$$f_{odd}(x_1, x_2, x_3) = x_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + x_1 x_2 x_3$$



Half Adder

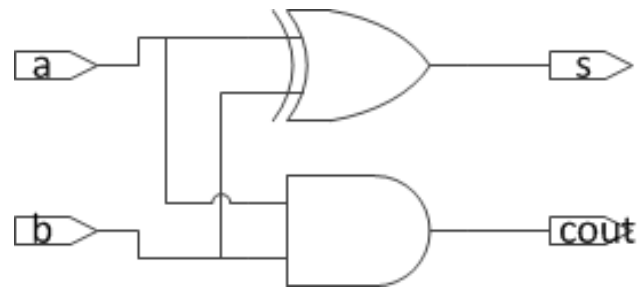


Output logic equations

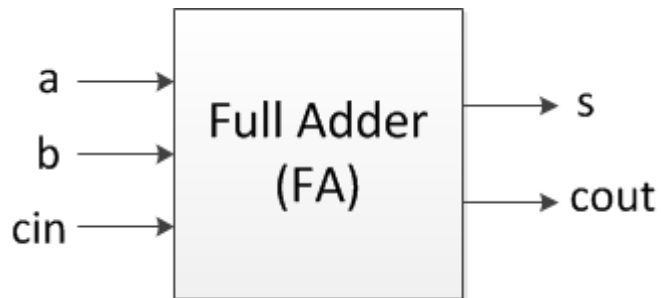
$$s = a \oplus b$$

$$c_{out} = a \cdot b$$

a	b	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Full Adder



a	b	c_{in}	c_{out}	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



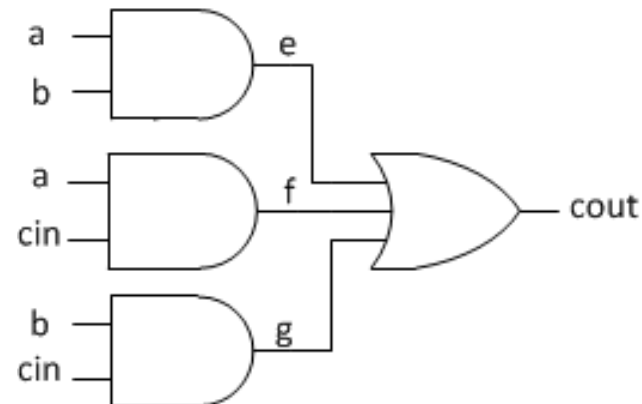
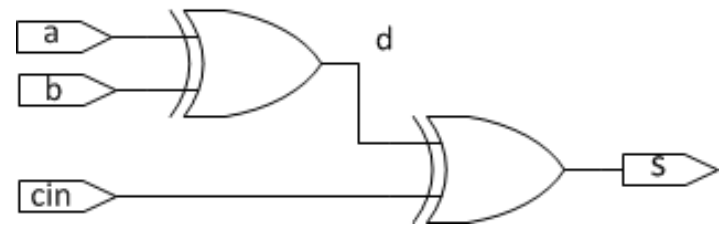
Full Adder Realization

Output logic equations

$$\begin{aligned}s &= a \oplus b \oplus c_{in} \\ &= \bar{a}\bar{b}c_{in} + \bar{a}b\bar{c}_{in} \\ &\quad + a\bar{b}\bar{c}_{in} + abc_{in}\end{aligned}$$

$$\begin{aligned}c_{out} &= \bar{a}bc_{in} + a\bar{b}c_{in} \\ &\quad + ab\bar{c}_{in} + abc_{in} \\ &= ab + ac_{in} + bc_{in} \\ &= \text{majority}(a, b, c_{in})\end{aligned}$$

Logic Circuit Diagram



HA and FA Verilog Structural Models

```
module halfadder (s, cout, a, b);  
  input a, b;  
  output s, cout;  
  and (cout, a, b);  
  xor (s, a, b);  
endmodule
```

```
module fulladder (s, cout, a, b, cin);  
  input a, b, cin;  
  output s, cout;  
  wire d, e, f, g;  
  xor (d, a, b);  
  xor (s, d, cin);  
  and (e, a, b);  
  and (f, a, cin);  
  and (g, b, cin);  
  or (cout, e, f, g);  
endmodule
```



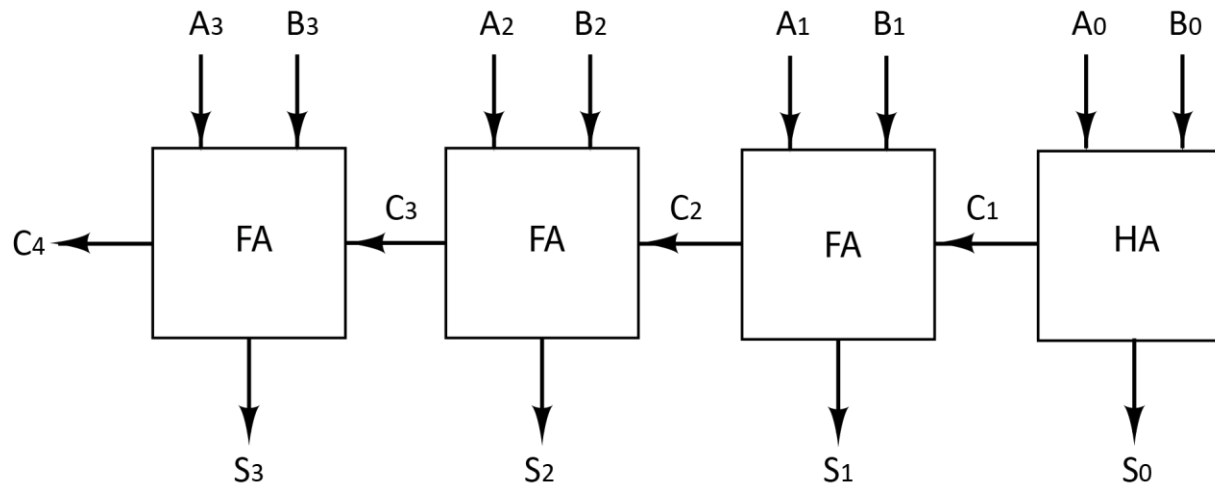
Full Adder Verilog Behavioral Models

```
module FAdf (s, cout, a, b, cin);  
  input a, b, cin;  
  output s, cout;  
  assign s = a ^ b ^ cin;  
  assign cout = a&b|a&cin|b&cin;  
endmodule
```

```
module FAbehave (s, cout, a, b, cin);  
  input a, b, cin;  
  output s, cout;  
  always @ (a, b, cin)  
    {cout,s} = a + b + cin;  
endmodule
```



Ripple-Carry Adder (4-bit)



$$(C_4 S_3 S_2 S_1 S_0)_2 = (A_3 A_2 A_1 A_0)_2 + (B_3 B_2 B_1 B_0)_2$$



Ripple-Carry Adder Verilog Model

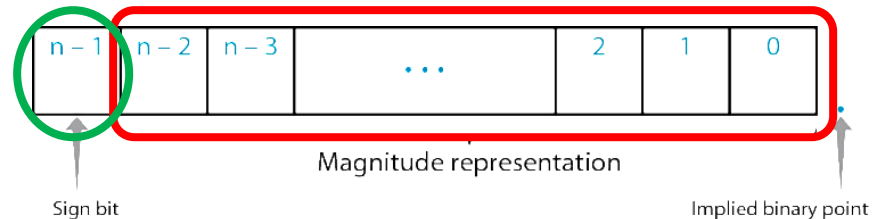
```
//Ripple Carry Adder Structural Model
module RippleCarryAdderStructural (
    input [3:0] A, B,                //declare input ports
    output [3:0] S,                  //declare output ports for sum
    output Cout;                     //declare carry out port
    wire [4:0] C;                    //declare internal nets

    //instantiate the full adder module for each stage of the ripple carry adder
    fulladder s0 (S[0], C[1], A[0], B[0], C[0]);    //stage 0
    fulladder s1 (S[1], C[2], A[1], B[1], C[1]);    //stage 1
    fulladder s2 (S[2], C[3], A[2], B[2], C[2]);    //stage 2
    fulladder s3 (S[3], C[4], A[3], B[3], C[3]);    //stage 3
    assign C[0] = 1'b0;                          //assign 0 to least significant carry-in
    assign Cout = C[4];                          //rename carry out port
endmodule
```

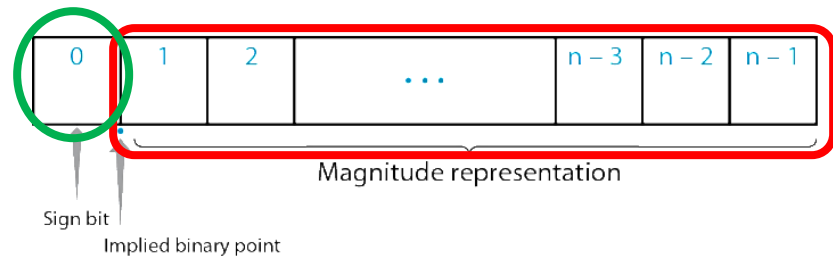


Fixed-Point Number Representation

- Typical formats for integers and fractions



(a)



(b)

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



Sign-Magnitude Method

- $N = \pm (a_{n-2} \dots a_0)_2$ is represented as
$$N = (sa_{n-2} \dots a_0)_{2sm},$$
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 $N = (a_{n-2} \dots a_0)_2$
 - If $N \geq 0$, it is represented by $(0a_{n-2} \dots a_0)_2$
 - If $N < 0$, it is represented by $[0a_{n-2} \dots A_0]_1$
- **1's complement** $[N]_1 = 2^n - (N)_2 - 1 = [N]_2 - 1$ where n is the number of bits in $[N]_1$
- Examples ($n = 4, 2^n = 10000$)
 - $+(110)_2 = (0110)_2$
 - $-(110)_2 = [0110]_1 = (10000 - 0110 - 1)_2 = (1001)_2$
- *Range* $2^{n-1} - 1 \geq N \geq -2^{n-1} + 1$ ($011\dots 11 \geq N \geq 100\dots 00$)
- *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 $N = (a_{n-2} \dots a_0)_2$
 - If $N \geq 0$, it is represented by $(0a_{n-2} \dots a_0)_2$
 - If $N < 0$, it is represented by $[0a_{n-2} \dots a_0]_2$
- **2's complement** $[N]_2 = 2^n - (N)_2$, where n is the number of digits in $(N)_2$.
- Example ($n = 4, 2^n = 10000$)
 - $+(110)_2 = (0110)_2$
 - $-(110)_2 = [0110]_2 = (10000 - 0110)_2 = (1010)_2$
- *Range* $2^{n-1} - 1 \geq N \geq -2^{n-1}$ ($011\dots 11 \geq N \geq 100\dots 00$)
- *Pros and Cons*
 - *Easy to implement in hardware*
 - *One representation of 0*
 - *Wider negative range*
 - *Not intuitive for humans*

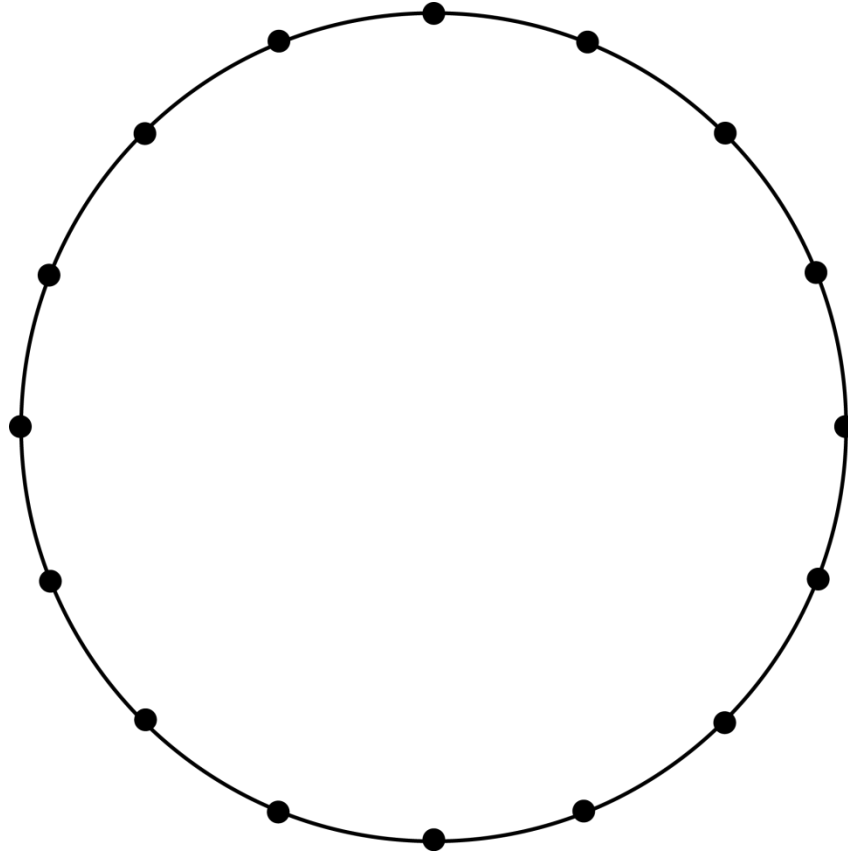


Signed Binary Numbers

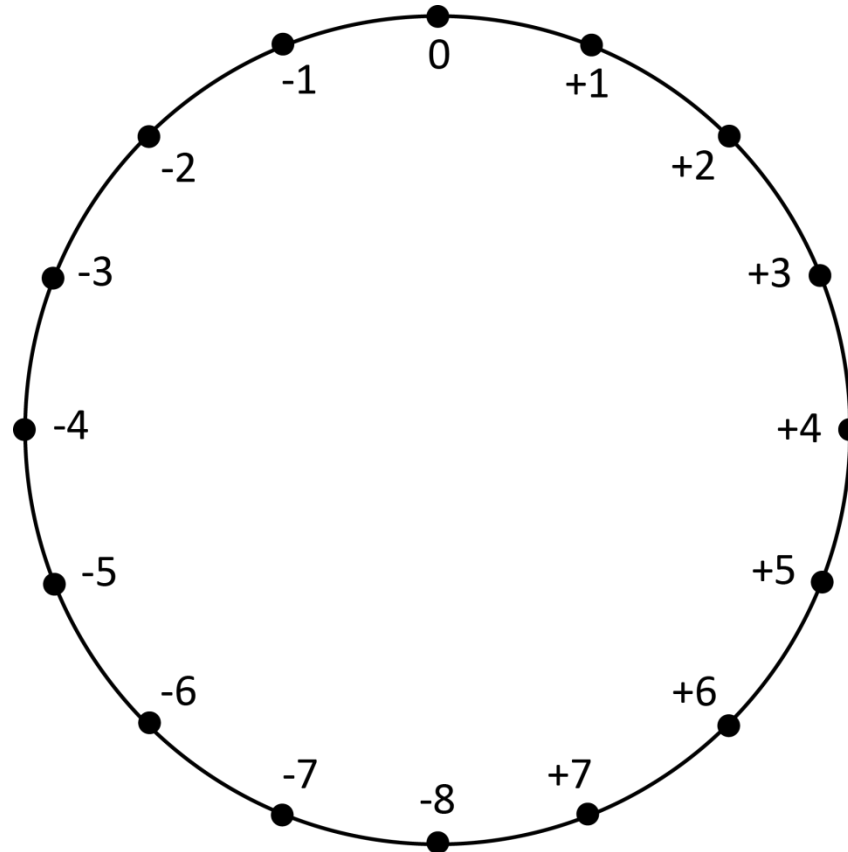
Decimal	Sign-Magnitude	Two's Complement	One's Complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
0	0000, 1000	0000	0000, 1111
-1	1001	1111	1110
-2	1010	1110	1101
-3	1011	1101	1100
-4	1100	1100	1011
-5	1101	1011	1010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



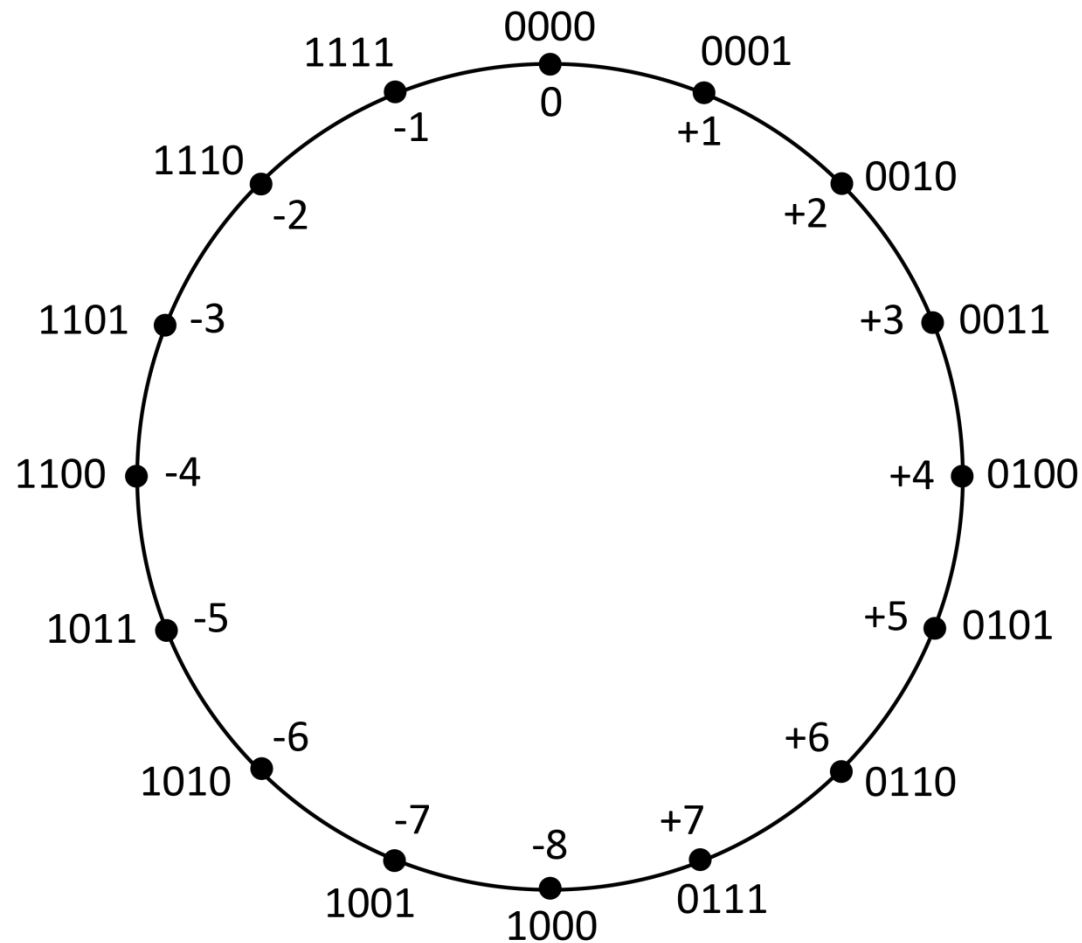
Two's Complement Numbers



Two's Complement Numbers



Two's Complement Numbers



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 $(1010)_2$ is $(0110)_2$.



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

$$[N]_2 = (1100)_2$$



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.
 - $2^{n-1} - 1 = (0, 11 \dots 1)_{2cns}$ and $-2^{n-1} = (1, 00 \dots 0)_{2cns}$

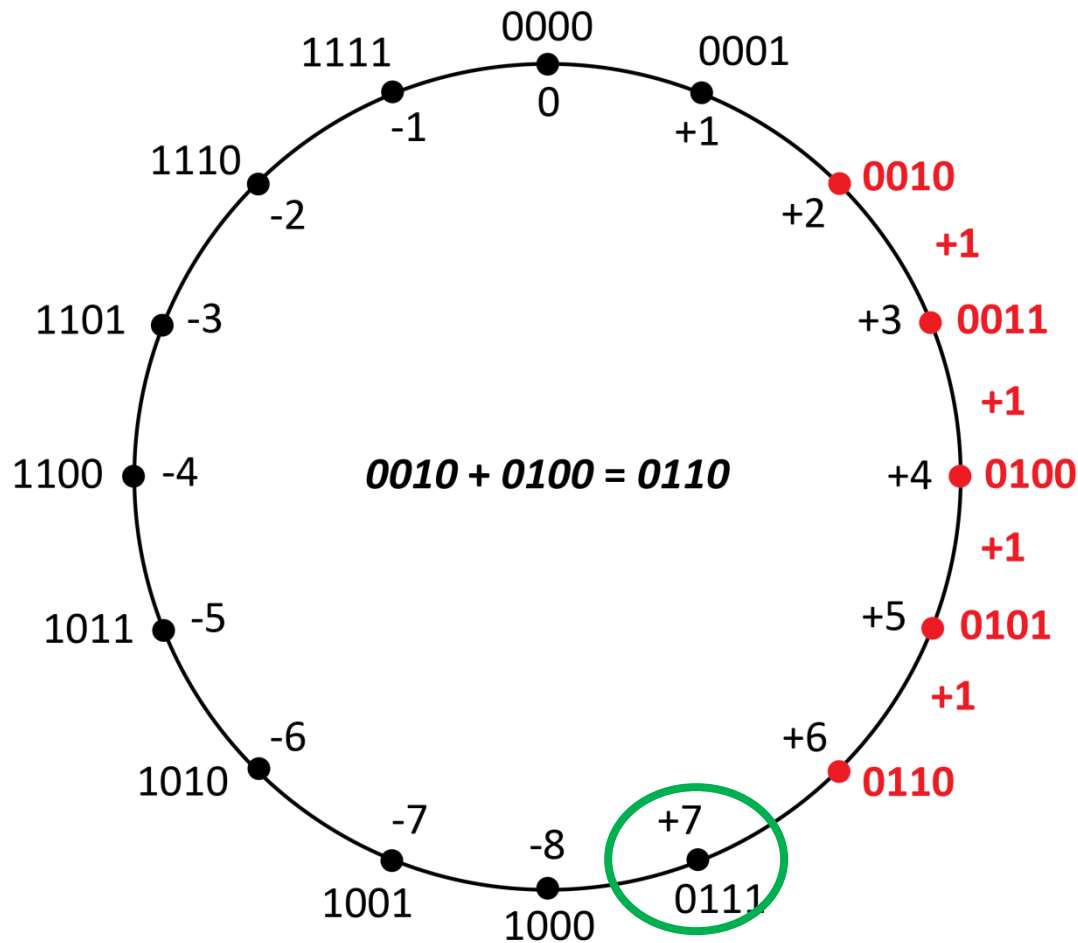


Two's Complement Arithmetic

- **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 n th 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

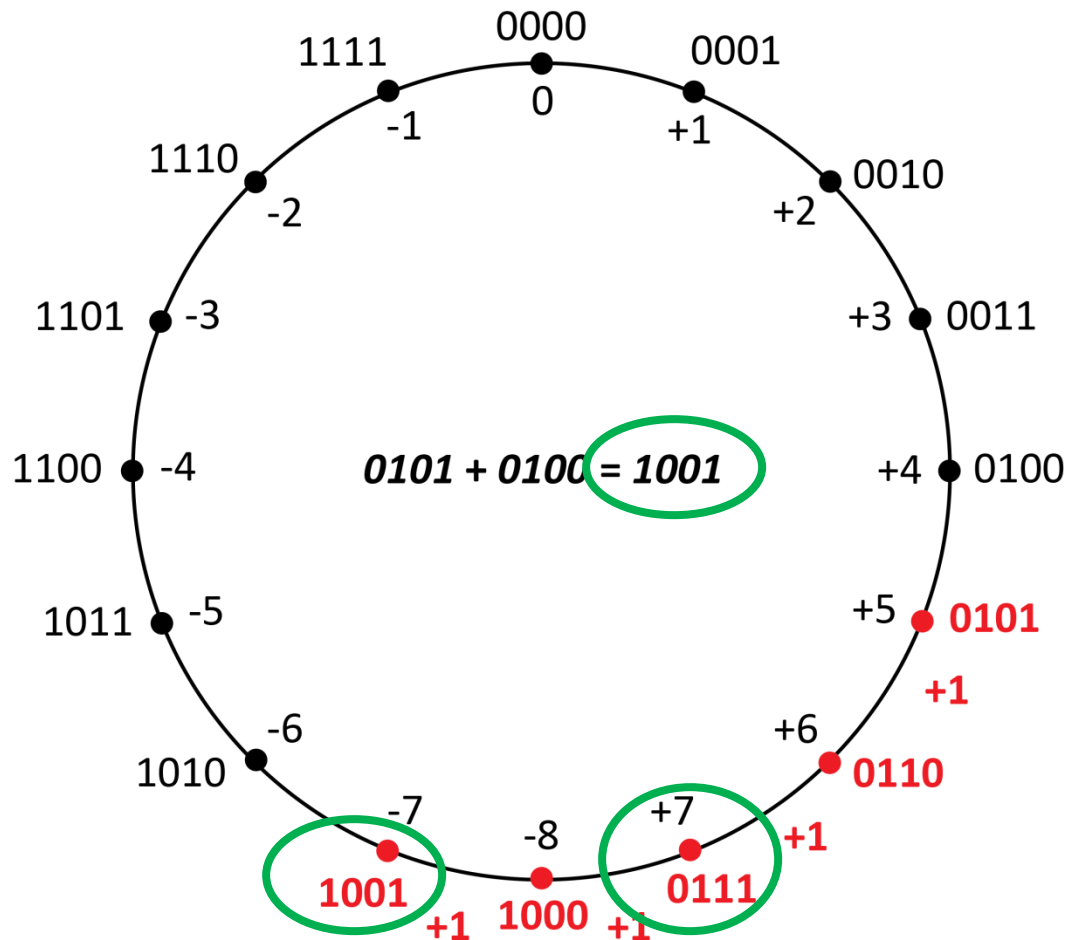


Two's Complement Arithmetic

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



Two's Complement Arithmetic



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, $(A)_2 = (B)_2 + [C]_2$ | 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 $(7)_{10} + (-(2)_{10})$
 - $(7)_{10} = +(111)_2 = (0111)_{2cns}$
 - $-(2)_{10} = -(010)_2 = (1110)_{2cns}$
 - $(7)_{10} - (2)_{10} = (0111)_{2cns} + (1110)_{2cns} = (0101)_{2cns} + \text{carry}$
 $= +(101)_2 = +(5)_{10}$



Two's Complement Arithmetic

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

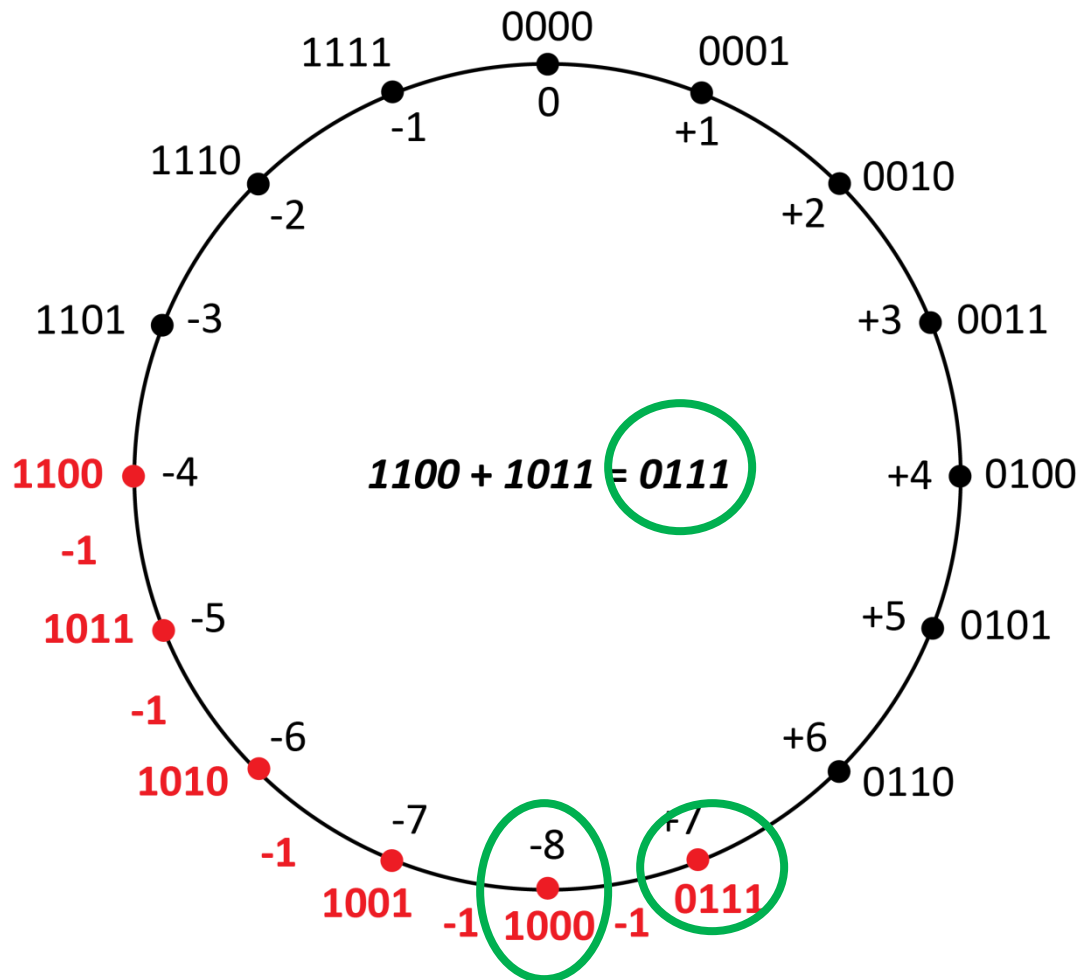


Two's Complement Arithmetic

- **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 $-(3)_{10} + (-(4)_{10})$
 - $-(3)_{10} = -(11)_2 = (1101)_{2cns}$, $-(4)_{10} = -(100)_2 = (1100)_{2cns}$
 - $-(3)_{10} - (4)_{10} = (1101)_{2cns} + (1100)_{2cns} = (1001)_{2cns} + \text{carry}$
 $= -(0111)_2 = -(7)_{10}$
- **Example:** $-(4)_{10} - (5)_{10} = ?$
 - Perform $-(4)_{10} + (-(5)_{10})$
 - $-(4)_{10} = -(100)_2 = (1100)_{2cns}$, $-(5)_{10} = -(101)_2 = (1011)_{2cns}$
 - $-(4)_{10} - (5)_{10} = (\textcolor{red}{1}100)_{2cns} + (\textcolor{red}{1}011)_{2cns} = (\textcolor{red}{0}111)_{2cns} + \text{carry}$
 - **An overflow is indicated by the sign bit of 0.**



Two's Complement Arithmetic



Two's Complement Arithmetic

Case Summary

<i>Case</i>	<i>Carry</i>	<i>Sign of A</i>	<i>Condition</i>	<i>Overflow?</i>
$A = B + C$	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

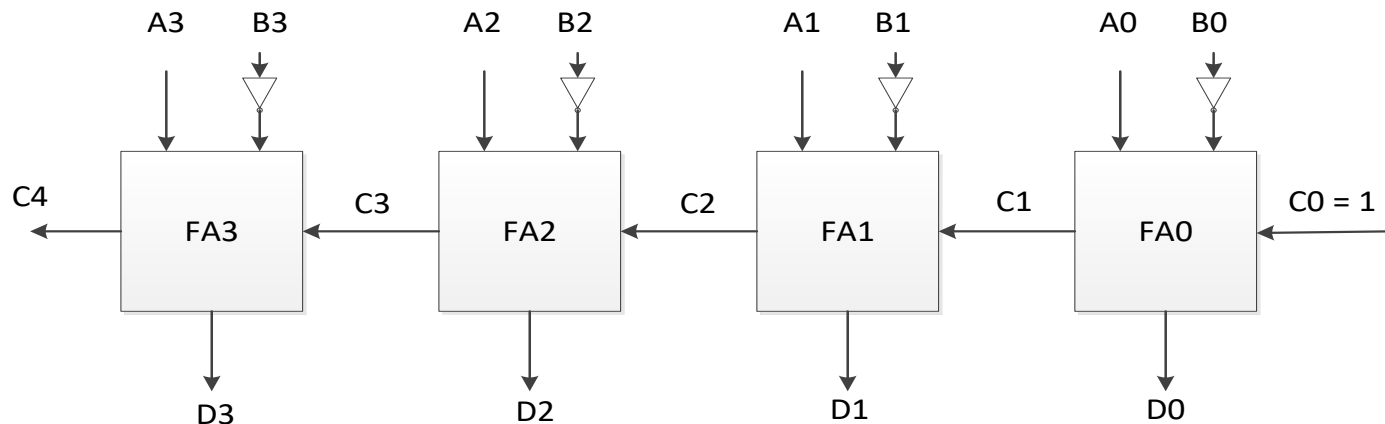
Overflow Detection for Addition

<i>Operation</i>	<i>Sign of X</i>	<i>Sign of Y</i>	<i>Sign of Z</i>	<i>Overflow?</i>
$Z = X + Y$	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

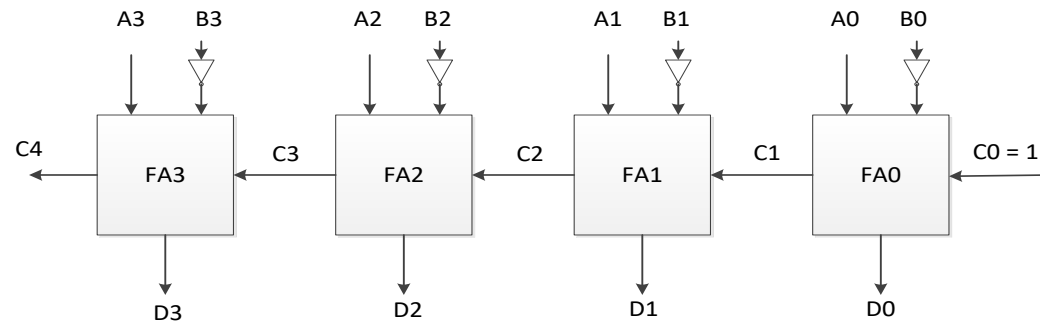
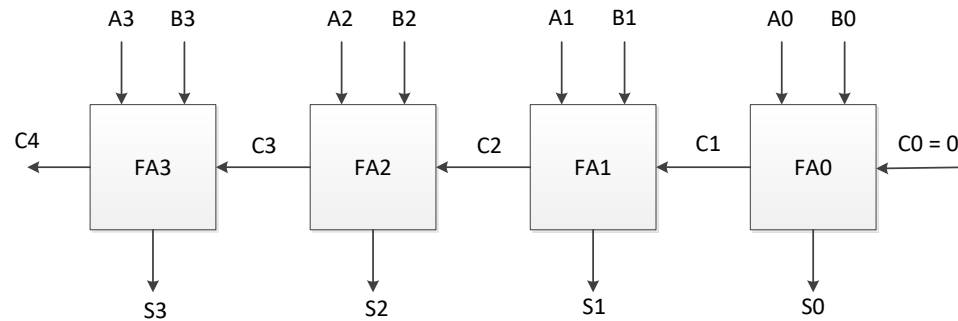


Using A Ripple-Carry Adder to Subtract

$$\begin{aligned}(C_4D_3D_2D_1D_0)_2 &= (A_3A_2A_1A_0)_2 - (B_3B_2B_1B_0)_2 \\ &= (A_3A_2A_1A_0)_2 + [B_3B_2B_1B_0]_{2\text{cns}} \\ &= (A_3A_2A_1A_0)_2 + [B_3B_2B_1B_0]_{1\text{cns}} + 1\end{aligned}$$



Ripple-Carry Adder/2's-Comp Subtractor



XOR-Gate Revisited



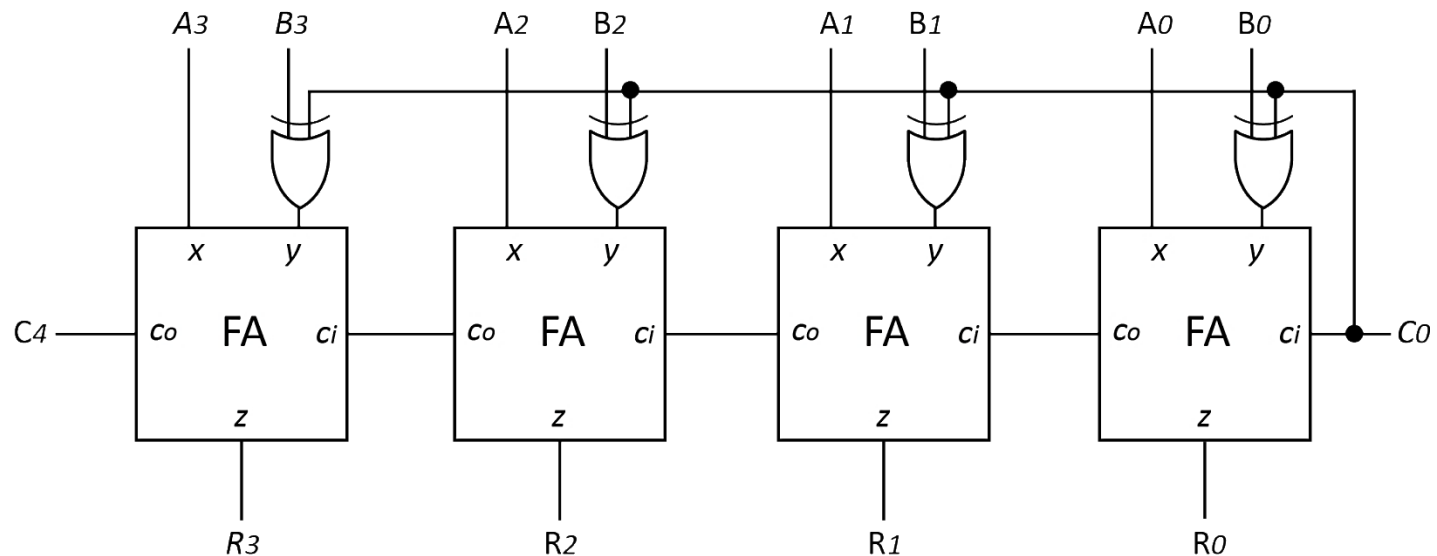
a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

If $b = 0$, then $f = a$.

If $b = 1$, then $f = a'$.



A Two's Complement Adder/Subtractor



If $C0 = 0$, then $(C_4 R_3 R_2 R_1 R_0)_2 = (A_3 A_2 A_1 A_0)_2 + (B_3 B_2 B_1 B_0)_2$. Therefore, $R = A + B$.

If $C0 = 1$, then $(C_4 R_3 R_2 R_1 R_0)_2 = (A_3 A_2 A_1 A_0)_2 + [B_3 B_2 B_1 B_0]_{1ns} + 1$. Therefore, $R = A - B$.



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Ripple-Carry Adder/Subtractor Verilog Model

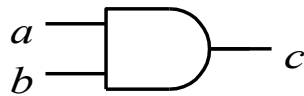
//Ripple Carry Adder Subtractor Structural Model

```
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
```

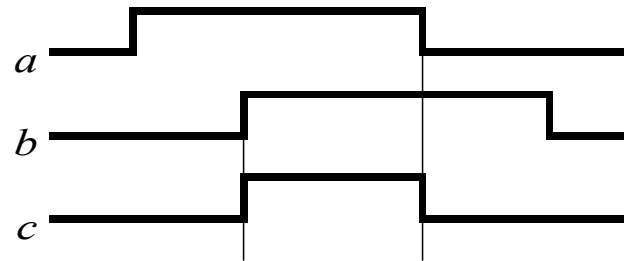


Propagation Delay

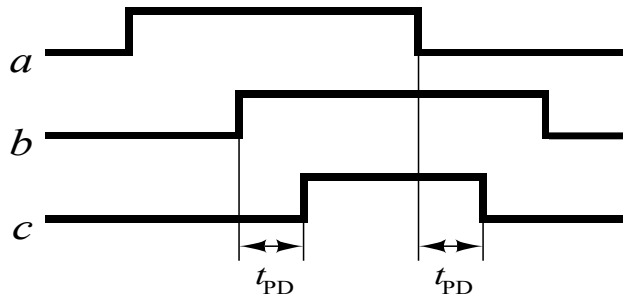
- Propagation delay through a logic gate



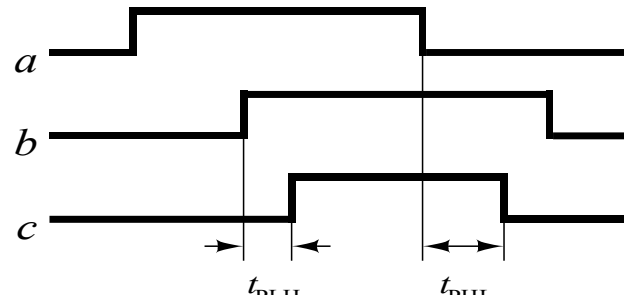
(a) Two-input AND gate



(b) Ideal (zero) delay



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

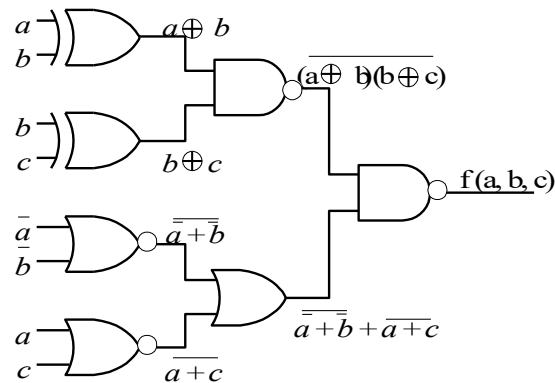


(d) $t_{PLH} < t_{PHL}$



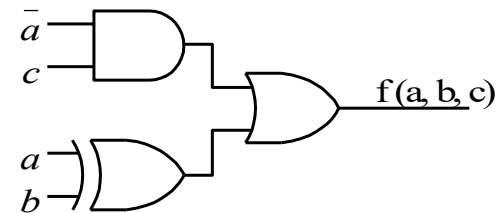
Circuit Propagation Delay

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

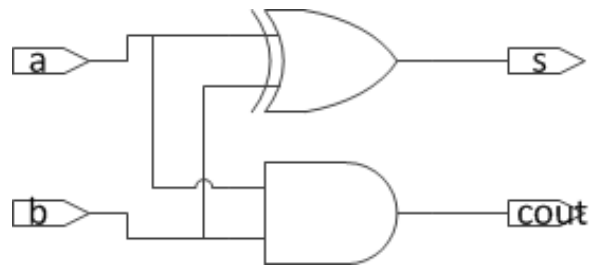
$$t_f = 2t_g$$

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



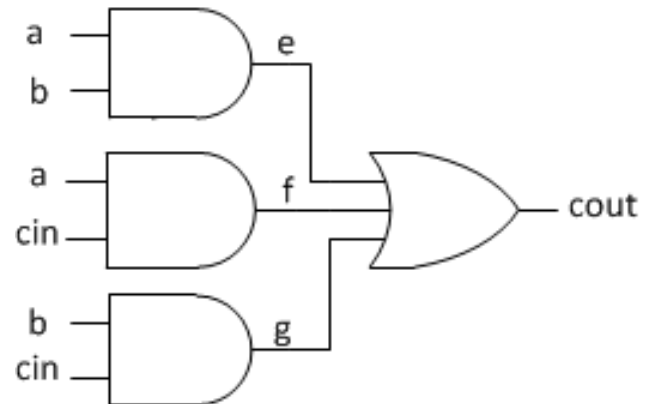
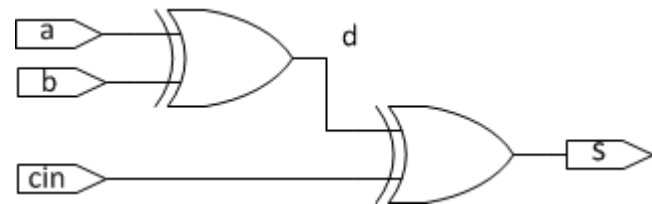
Add Times for Basic Adders

Half-Adder



t_g

Full-Adder

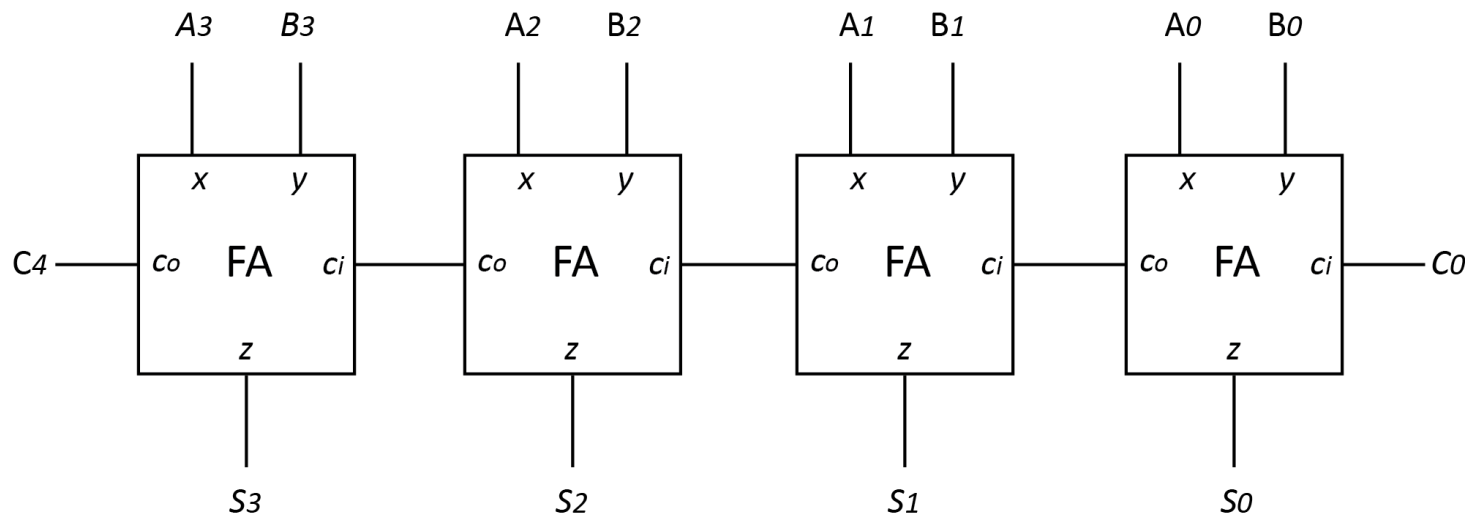


$2t_g$



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 $2nt_g$.



UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Add Time Implications

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

$t_g \backslash n$	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

$t_{\text{add}}, f_{\text{MAX}}$



How Fast Do We Need to Add?

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

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



Two's Complement Arithmetic

Case Summary

<i>Case</i>	<i>Carry</i>	<i>Sign of A</i>	<i>Condition</i>	<i>Overflow?</i>
$A = B + C$	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

<i>Operation</i>	<i>Sign of X</i>	<i>Sign of Y</i>	<i>Sign of Z</i>	<i>Overflow?</i>
$Z = X + Y$	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

$$\begin{aligned} (+) + (+) = (+): & \quad 0110 + 0001 = 0111 \quad \checkmark \\ & : \quad 0110 + 0010 = 1000 \quad \times \quad \text{OVERFLOW} \end{aligned}$$

$$\begin{aligned} (-) + (-) = (-): & \quad 1010 + 1111 = \pm 1001 \quad \checkmark \\ & : \quad 1010 + 1110 = \pm 1000 \quad \checkmark \\ & : \quad 1010 + 1101 = \pm 0111 \quad \times \quad \text{OVERFLOW} \end{aligned}$$



Design an Overflow Detection Circuit

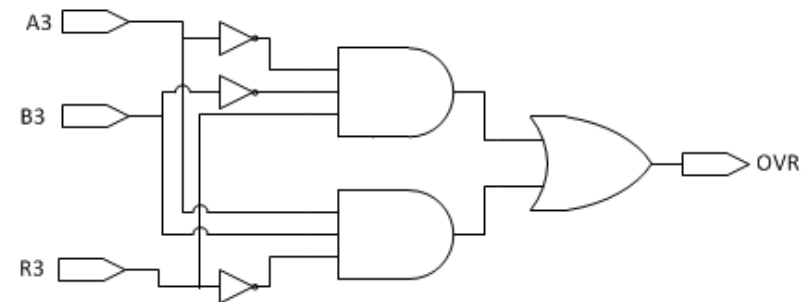
So $(A_3A_2A_1A_0) + (B_3B_2B_1B_0) = C_4R_3R_2R_1R_0$

$0111 + 0001 = 1000$ **OVERFLOW**

$1111 + 1000 = \cancel{1}0111$ **OVERFLOW**

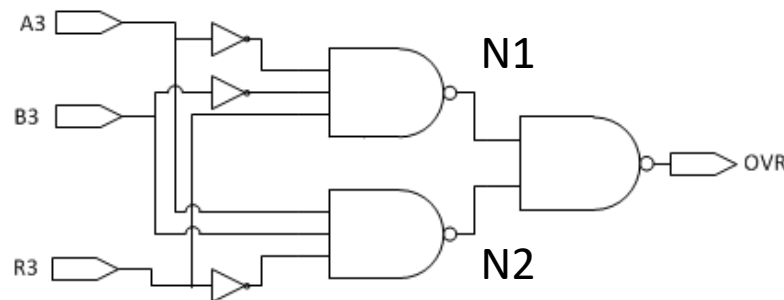
		A_3B_3			
R_3		00	01	11	10
	0			1	
	1	1			

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



NAND-NAND Realization of OVR

$$\begin{aligned} \text{OVR} &= \bar{A}_3 \bar{B}_3 R_3 + A_3 B_3 \bar{R}_3 \\ &= \overline{(\bar{A}_3 \bar{B}_3 R_3 + A_3 B_3 \bar{R}_3)} \\ &= (\bar{A}_3 \bar{B}_3 R_3) \cdot (A_3 B_3 \bar{R}_3) \\ &= N1 \cdot N2 \end{aligned}$$



OR-AND Overflow Detection Circuit

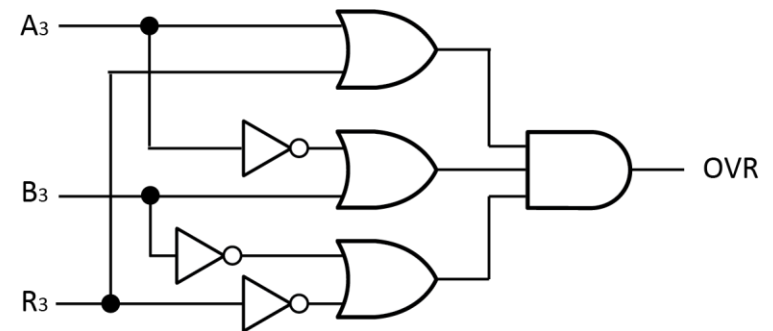
Recall $(A_3A_2A_1A_0) + (B_3B_2B_1B_0) = C_4R_3R_2R_1R_0$

0111 + 0001 = 1000 OVERFLOW

1111 + 1000 = ~~1~~0111 OVERFLOW

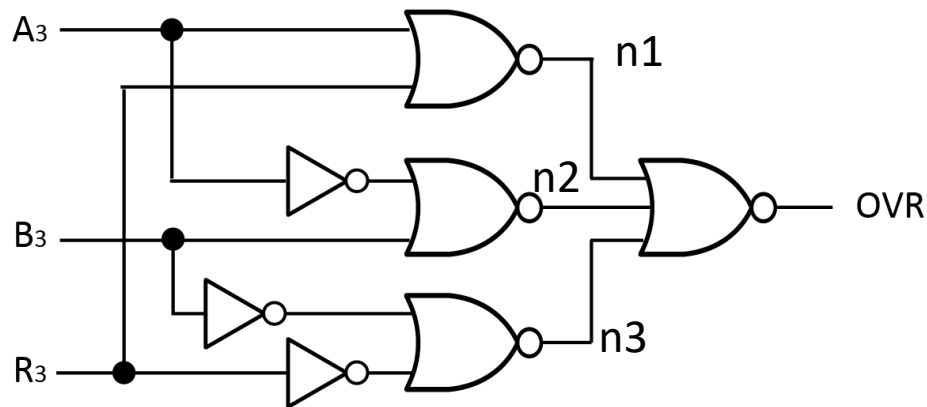
		A_3B_3			
		00	01	11	10
R_3	0	0	0	1	0
	1	1	0	0	0

$$OVR = (A_3 + R_3)(\bar{A}_3 + B_3)(\bar{B}_3 + \bar{R}_3)$$



NOR-NOR Realization of OVR

$$\begin{aligned} \text{OVR} &= (A_3 + R_3)(\bar{A}_3 + B_3)(\bar{B}_3 + \bar{R}_3) \\ &= \overline{\overline{(A_3 + R_3)(\bar{A}_3 + B_3)(\bar{B}_3 + \bar{R}_3)}} \\ &= \overline{(A_3 + R_3) + (\bar{A}_3 + B_3) + (\bar{B}_3 + \bar{R}_3)} \\ &= \overline{n1 + n2 + n3} \end{aligned}$$



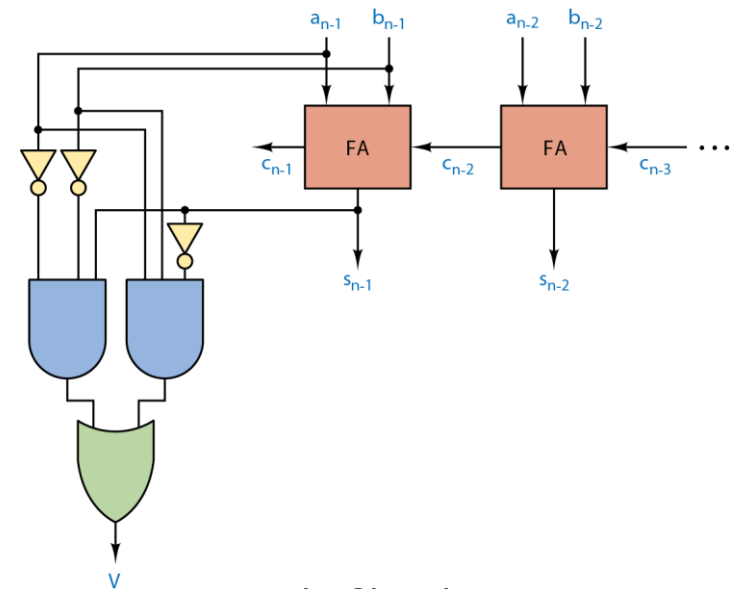
Two's Complement Overflow Detection

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



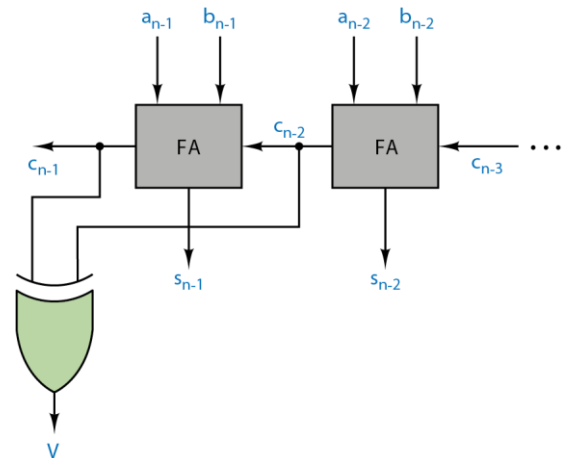
Two's Complement Overflow Detection

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



Q & A

