

DIGITAL LOGIC

Lecture 1 Course Introduction Number Systems

2023 Fall

Outline

- Introduction to course
- Lecture
 - Digital Number Systems
 - Data Representation
 - Binary Logic
- PreLab
 - What is an FPGA
- Reading: Textbook, Chapter 1

Course Information

- Course website: **CS207-30022126-2023FA: Digital Logic Fall 2023**
 - Blackboard: 教师: 工学院/计算机科学与工程系 白雨卉; 计算机科学与工程系 王薇;
- Instructor:
 - Dr. Yuhui BAI (baiyh@sustech.edu.cn)
 - Office: 411 College of Engineering South
 - Office hour: Wed. 14:00-16:00 (by appointment)
- Lecture
 - 10:20-12:10 Monday, Lecture Hall #1
- Lab
 - 14:00 -15:50 Mon., 511, Lecture Hall #3 (Yuhui BAI)
 - 16:20 -18:10 Tue., 504, Lecture Hall #3 (Wei WANG)
 - 14:00 -15:50 Wed., 510, Lecture Hall #3 (Wei WANG)
 - 10:20-12:10 Fri., 510, Lecture Hall #3 (Wei WANG)
 - 14:00 -15:50 Mon., 503, Lecture Hall #3 (Wei WANG)

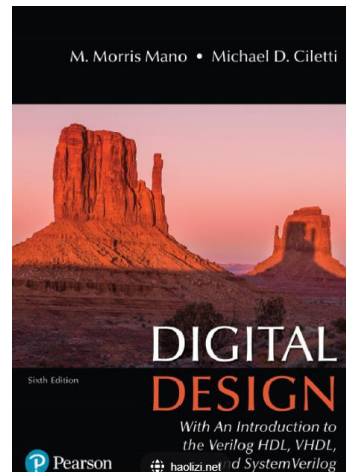


群名称: CS207-2023f

群 号: 922162289

Textbook

- Textbook:
 - Digital Design: With an Introduction to the Verilog HDL, VHDL, and System Verilog by *M. Morris Mano and Michael D. Ciletti*, 6th edition.



- Reference book:
 - Digital Principles and Logic Design by A. Saha and N. Manna.
 - Digital Logic Design by B. Holdsworth and C. Woods

Course Outline

1. Digital Systems and Binary Numbers
 - Binary Systems, Conversions, Signed Binary, Codes
2. Boolean Algebra and Logic Gates
 - Theorems, Boolean Functions, operators, gates
3. Gate-level Minimization
 - Truth table, K Map, two-level implementations, NAND, NOR
4. Combinational Logic
 - Combinational circuits, arithmetic logic, mux, de-mux, encoder, decoder
5. Synchronous Sequential Logic
 - Sequential circuit, Latches, Flip flops, State Machines
6. Registers and Counters
7. Memory and Programmable Logic
 - RAM, ROM, FPGA
8. Verilog (Lab)

Tentative Schedule

WEEK	LECTURE	TOPIC	TOPIC
1	Lec #1	Binary Numbers	Environment Setup
2	Lec #2	Boolean Algebra & Logic Gates	Structural-Based Design
3	Lec #3	Gate-Level Minimization	Dataflow Design
4	Lec #4	Two-Level Implementation	Testbench
5	Lec #5	Combinational Logic	Behavioral-Based Design
6	Lec #5	Combinational Logic (cont.)	Encoder, Decoder
7	Lec #6	Standard Components	Multiplexer, De-multiplexer
8	Lec #7	Latches and Flip-flops	Verilog Summary
9	Mid-term Exam	Mid-term Exam (contents of Lec 1-6) No lecture, lab remains unchanged	Latch, FlipFlop & Project Release
10	Lec #8	Synchronous Sequential Logic	Finite state machine
11	Lec #9	Arithmetic Circuit	Frequency divider
12	Lec #10	Registers	Full adder & Project Q&A
13	Lec #10	Registers (cont.)	Register
14	Lec #11	Counters	Counter
15	Lec #12	Memory and Programmable Logic	Project Inspection
16	Lec #12	Revision	Project Inspection

Grading criteria

- Lecture (20%)
 - 10% Attendance and in-class Quiz
 - Same mark as the actual mark if above 60;
 - 60, if 60 or below or you are absent with an accepted permission;
 - 0, if absence.
 - 10% Homework
- Exam (50%)
 - 25% Mid-term examination
 - 25% Final examination
- Lab (30%)
 - 5% Attendance and Lab practices
 - 10% Lab assignments on OJ
 - 15% Lab Project
 - In groups of 2~3. Please team up as soon as possible.
 - Please try to choose classmates from the same lab class.
 - In special circumstances where cross-class teams are needed, it is important to ensure that all team members can attend the Project Inspection at the end of the semester.

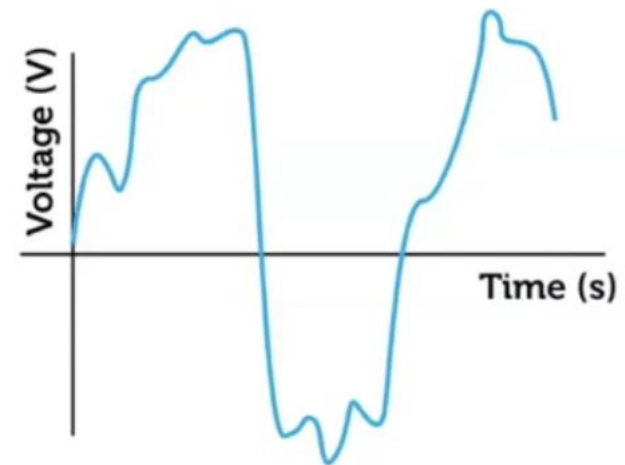
Outline

- Introduction to course
- **Lecture**
 - Digital Number Systems
 - Data Representation
 - Binary Logic
- PreLab
 - What is an FPGA

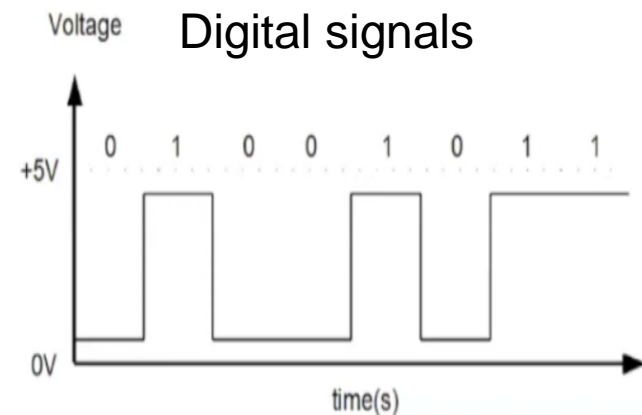
Analog vs. Digital Signals

- Signal definition
 - Quantity that can represent and convey information
 - Passed between devices to send and receive information
- Analog signals
 - Converts information into waves of varying amplitude and frequency
 - Continuously changes
 - Records exact waveform
- Digital signals
 - ON (1) or OFF (0) pulses (i.e. binary)
 - Square waves made by sampling along the wave form

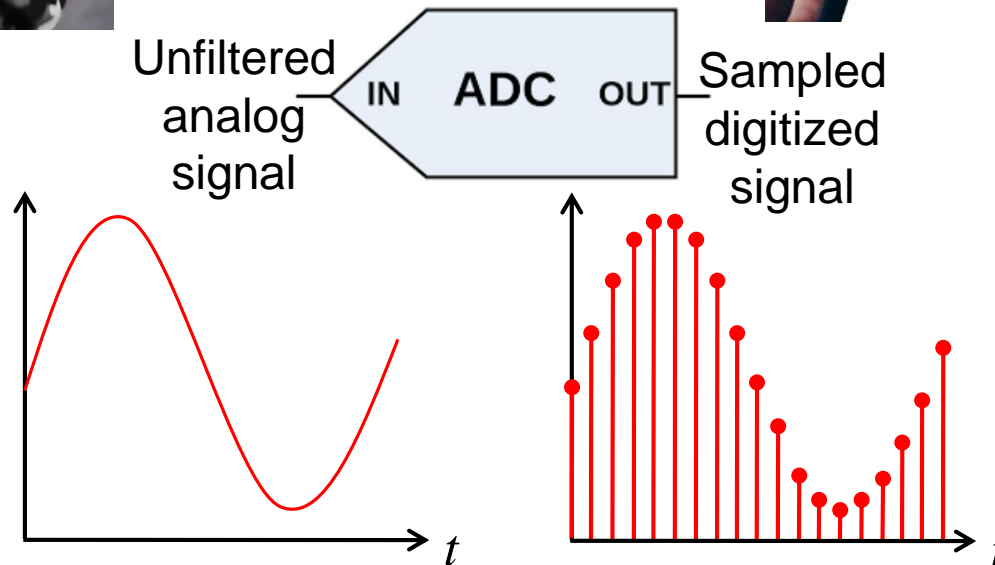
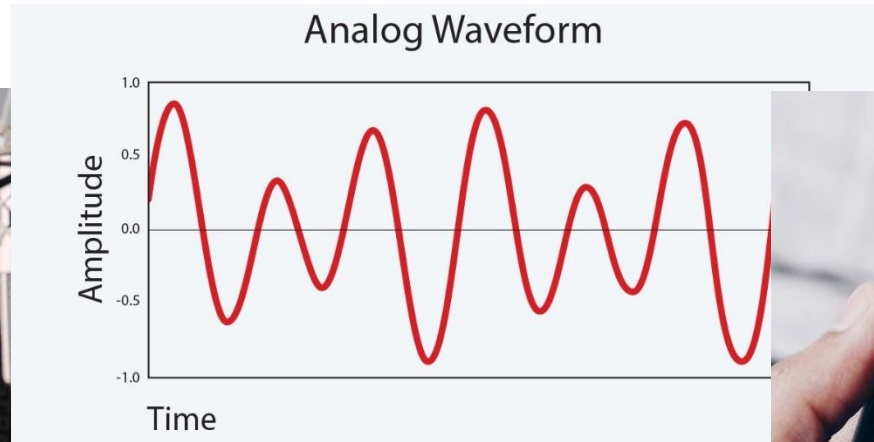
Analog signals



Digital signals



Analog vs. Digital Signals



Digital Systems

- A digital system is a system that processes digital signals or data. It operates on discrete values and performs operations such as logic, arithmetic, and data storage in a binary format.
- Digital systems are prevalent in modern electronics, including computers, smartphones, and digital communication devices, due to their reliability and ease of processing.

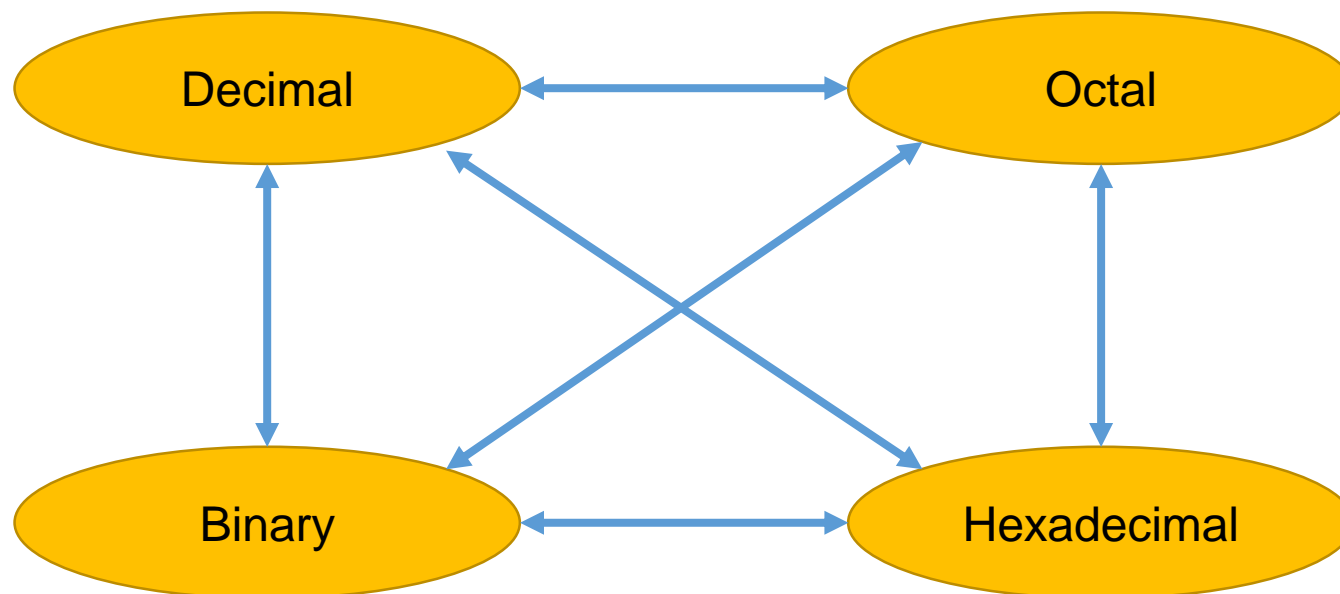
Common Number Systems

- It is natural for human to use **decimal system**(十进制)
- In a digital world, we think in **binary**(二进制)
- The **octal** (八进制) and **hexadecimal** (十六进制) numbers are shorter forms for representing binary numbers.

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
0	0000	00	0x0
1	0001	01	0x1
2	0010	02	0x2
3	0011	03	0x3
4	0100	04	0x4
5	0101	05	0x5
6	0110	06	0x6
7	0111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF

Conversion among Bases

- The possibilities:



- A quick example:

$$\bullet 25_{10} = 11001_2 = 31_8 = 19_{16}$$

Base or Radix

Radix-r to Decimal Conversion

- We use Positional Number Systems: Let r be the **radix** (or **base**), then the $(n+m)$ -digit number

$$D = d_{n-1}d_{n-2} \dots d_1d_0 \cdot d_{-1}d_{-2} \dots d_{-m} \quad 0 \leq d < r$$

has the value

radix point

$$D = d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \dots + d_1r + d_0 + d_{-1}r^{-1} + d_{-2}r^{-2} + \dots + d_{-m}r^{-m}$$

Most-significant Digit (MSD)

Least-significant Digit (LSD)

$$D = \sum_{i=-m}^{n-1} d_i r^i$$

Radix-r to Decimal Conversion

- **Decimal Number System:** Base (radix) $r = 10$

2	1	0		-1	-2
<div style="border: 2px solid orange; padding: 5px; display: inline-block;">5</div>	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">1</div>	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">2</div>	•	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">7</div>	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">4</div>

- Coefficients $D=(d_2d_1d_0.d_{-1}d_{-2}) = (512.74)_{10}$
- $(512.74)_{10} = 5 \times 10^2 + 1 \times 10^1 + 2 \times 10^0 + 7 \times 10^{-1} + 4 \times 10^{-2}$

Exercise:

$$1010.101_2 = ?_{10}$$

$$22.22_4 = ?_{10}$$

$$12.5_8 = ?_{10}$$

$$A.A_{16} = ?_{10}$$

- **Binary Number System:** Base (radix) $r = 2$

2	1	0		-1	-2
<div style="border: 2px solid orange; padding: 5px; display: inline-block;">1</div>	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">0</div>	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">1</div>	•	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">0</div>	<div style="border: 2px solid orange; padding: 5px; display: inline-block;">1</div>

- Coefficients $D=(b_2b_1b_0.b_{-1}b_{-2}) = (101.01)_2$
- $(101.01)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (5.25)_{10}$

Radix-r to Decimal Conversion

$$D = \underbrace{d_{n-1}}_{\text{MSD}} r^{n-1} + d_{n-2} r^{n-2} + \dots + d_1 r + d_0 + d_{-1} r^{-1} + d_{-2} r^{-2} + \dots + \underbrace{d_{-m}}_{\text{LSD}} r^{-m}$$

Most-significant Digit (MSD)

Least-significant Digit (LSD)

Exercise:

$$1010.101_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 10.625_{10}$$

$$22.22_4 = 2 \cdot 4^1 + 2 \cdot 4^0 + 2 \cdot 4^{-1} + 2 \cdot 4^{-2} = 10.625_{10}$$

$$12.5_8 = 1 \cdot 8^1 + 2 \cdot 8^0 + 5 \cdot 8^{-1} = 10.625_{10}$$

$$A.A_{16} = 10 \cdot 16^0 + 10 \cdot 16^{-1} = 10.625_{10}$$

Decimal to Radix-r Conversion

- Integer part: Successive divisions by r and observe the remainders
- Fraction: Successive multiplications by r and observe the integer part

Decimal to Binary Conversion (1)

- For Integer
- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
$13 / 2 =$	6	1	$a_0 = 1$
$6 / 2 =$	3	0	$a_1 = 0$
$3 / 2 =$	1	1	$a_2 = 1$
$1 / 2 =$	0	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3 a_2 a_1 a_0)_2 = (1101)_2$

↑
↑
MSB
LSB

Decimal to Binary Conversion (2)

- For Fraction, the computation is reversed again
- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

		Integer	Fraction	Coefficient
0.625	$\times 2 =$	1	. 25	$a_{-1} = 1$
0.25	$\times 2 =$	0	. 5	$a_{-2} = 0$
0.5	$\times 2 =$	1	. 0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}a_{-2}a_{-3})_2 = (0.101)_2$

↑
MSB

↑
LSB

Decimal to Binary Conversion (3)

- Easier method

Example: $(35.375)_{10}$

2	35	...	1
2	17	...	1
2	8	...	0
2	4	...	0
2	2	...	0
	1	...	1

	0.375	
x	2	
	0.750	...
x	2	
	1.50	...
x	2	
	1.0	...

• $(100011.011)_2$

Decimal to Octal Conversion

Example: $(175.3125)_{10}$

	Quotient	Remainder	Coefficient
$175 / 8 =$	21	7	$a_0 = 7$
$21 / 8 =$	2	5	$a_1 = 5$
$2 / 8 =$	0	2	$a_2 = 2$

Integer part: $(175)_{10} = (a_2 a_1 a_0)_8 = (257)_8$

	Integer	Fraction	Coefficient
$0.3125 * 8 =$	2	. 5	$a_{-1} = 2$
$0.5 * 8 =$	4	. 0	$a_{-2} = 4$

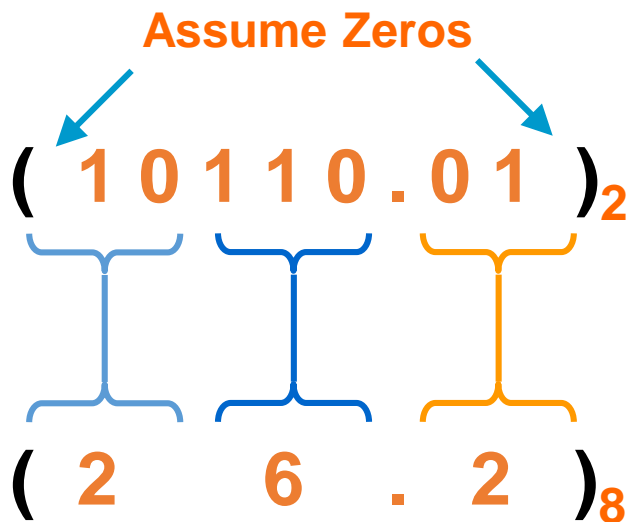
Fraction part: $(0.3125)_{10} = (0.a_{-1} a_{-2} a_{-3})_8 = (0.24)_8$

Answer: $(175.3125)_{10} = (a_2 a_1 a_0.a_{-1} a_{-2} a_{-3})_8 = (257.24)_8$

Radix-r to Radix-r Conversion

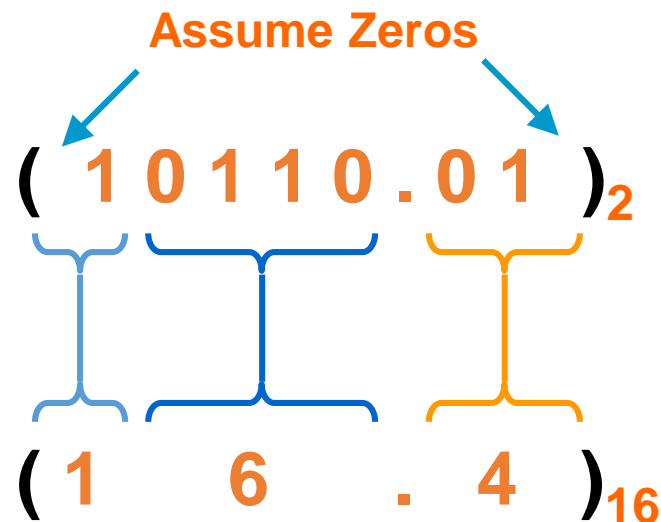
• Binary – Octal

- Each group of 3 bits represents an octal digit starting from radix point
- Works both ways (Binary to Octal & Octal to Binary)



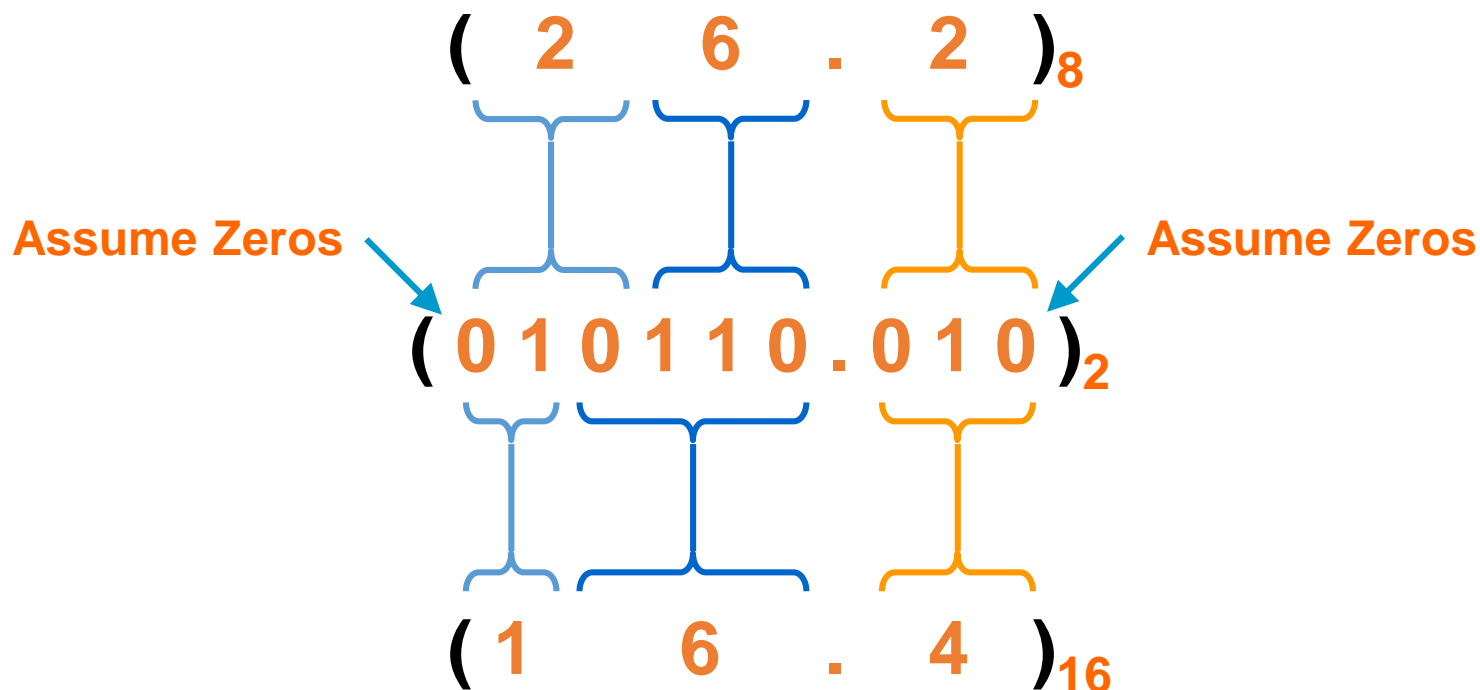
• Binary – Hexadecimal

- Each group of 4 bits represents a hexadecimal digit starting from radix point
- Works both ways (Octal to Hex & Hex to Octal)



Radix-r to Radix-r Conversion (2)

- Octal – Hexadecimal
- Convert to **Binary** as an intermediate step



Common Notions

- Bits

10010110

most significant bit least significant bit

- Bytes

byte

10010110

- Bytes

CEBF9AD7

most significant byte least significant byte

Power	Meaning	Prefix	Symbol
2^{10}	1024	Kilo	K
2^{20}	1024^2	Mega	M
2^{30}	1024^3	Giga	G
2^{40}	1024^4	Tera	T
2^{50}	1024^5	Peta	P
2^{60}	1024^6	Exa	E
2^{70}	1024^7	Zetta	Z

e.g. 1MB = 1024KB

Binary Addition

- Same rules as for decimal numbers
- Column Addition

	1	1	1	1	1	1		
		1	1	1	1	0	1	$(10)_2 = (2)_{10}$
+			1	0	1	1	1	$= 61$
								$= 23$
	1	0	1	0	1	0	0	$= 84$

Overflow

- Digital systems operate on a fixed number of bits
- Overflow(溢出): when result is too big to fit in the available number of bits
- Example: Add the following 4-bit binary numbers

$$\begin{array}{r} 1 \\ 1001 \\ + 0101 \\ \hline 1110 \end{array}$$

$$\begin{array}{r} 111 \\ 1011 \\ + 0110 \\ \hline 10001 \end{array}$$

Overflow!

Complements

- When human do subtraction, we use “borrow” concept to borrow a 1 from a higher significant position.
- It is hard for circuits to design “borrow”. So complements are used to implement subtraction.
 - Simplify the subtraction operation.
 - Simpler, less expensive circuits.
- Two types for radix-r system
 - Radix complement (补码) (r 's-complement)
 - Diminished radix complement (反码) ($(r-1)$'s-complement)
- Examples:
 - For a binary system: 2's complement and 1's complement.
 - For a decimal system: 10's complement and 9's complement.

Complements for decimal system

- Diminished radix complement
 - 9's-complement of 540 = $999 - 540 = 459$
 - 9's-complement of 12 = $999 - 012 = 987$
- Radix complement
 - 10's-complement of 540 = $1000 - 540 = 460$
 - 10's-complement of 12 = $1000 - 012 = 988$
 - **Easier method 1:** Calculate the diminished radix complement, then plus one
 - 10's-complement of 540 = $999 - 540 + 1 = 460$
 - **Easier method 2:** use r minus the least significant non-zero digit, and $r - 1$ minus digits on the left
 - The least significant non-zero digit of 540 is 4: $10 - 4 = 6$;
 - Digits on the left is 5: $9 - 5 = 4$;
 - The 10's complement of 540 is 460.

Complements for binary system

- 1's Complement (*Diminished Radix Complement*) for binary

- All '0's become '1's
- All '1's become '0's

Example $(10110000)_2$
 $\Rightarrow (01001111)_2$

- 2's Complement: 1's complement, then plus one:

- Another way: leave the first non-zero LSB unchanged, and then replacing 1's with 0's and 0's with 1's in the other MSBs:

$$\begin{array}{r}
 10110000 \\
 01001111 \\
 + 1 \\
 \hline
 01010000
 \end{array}$$

$$\begin{array}{r}
 101\textcircled{1}0000 \\
 \swarrow \\
 \text{first non-zero LSB} \\
 01010000
 \end{array}$$

Subtraction with Complements

- Replace subtraction with addition
- $M - N = M + r$'s complement of N
 - If $M \geq N$, the sum will produce an end **carry** r^n which is **discarded**, and what is left is the result $M - N$
 - If $M < N$, the sum does not produce an end carry. It is equal to the r 's complement of $(M - N)$. The correct answer is generated by
 - taking the r 's **complement** of the answer
 - then adding a **negative sign** to the front
- Pay attention to align the number of digits for two operands

Subtraction with 10's Complement

- Example with $M \geq N$
 - Using 10's complement, subtract $72532 - 3250$.

$$\begin{array}{r} M = 72532 \\ \text{10's complement of } N = +96750 \\ \hline \text{Sum} = 169282 \\ \text{Discard end carry } 10^5 = -100000 \\ \hline \text{Answer} = 69282 \end{array}$$

- Example with $M < N$
 - Using 10's complement, subtract $3250 - 72532$.

$$\begin{array}{r} M = 03250 \\ \text{10's complement of } N = +27468 \\ \hline \text{Sum} = 30718 \end{array}$$

➡ There is no end carry.

➡ Therefore, the answer is $-(10\text{'s complement of } 30718) = -69282$.

Subtraction with 2's Complement

- Example:
 - Given the two binary numbers $X = 1010100$ and $Y = 1000011$ ($X > Y$), perform the subtraction (a) $X - Y$; and (b) $Y - X$, by using 2's complement.

(a)	$X =$	1010100
	2's complement of $Y =$	<u>+0111101</u>
	Sum =	10010001
	Discard end carry $2^7 =$	<u>-10000000</u>
	Answer. $X - Y =$	0010001

(b)	$Y =$	1000011
	2's complement of $X =$	<u>+ 0101100</u>
	Sum =	1101111

There is no end carry.
Therefore, the answer is
 $Y - X = -(2\text{'s complement of } 1101111) = -0010001$.

Answer. $Y - X = -0010001$

Signed Binary Numbers

- In real life one may have to face a situation where both positive and negative numbers may arise.
 - We have + and –.
 - Digital systems represent everything with binary digits.
- Three types of representations of signed binary numbers:
 - Sign-magnitude representation
 - Signed-1's complement representation
 - Signed-2's complement representation
- In Signed binary system, the convention is to make the **sign bit (MSB)** 0 for positive and 1 for negative.

Signed Binary Numbers

- Example, assume 9-bits number representation:
- $(105)_{10}$?
- $105_{10} = 1101001_2$, represent in 9 bits
 - Signed-magnitude representation of 105: 001101001
 - Signed-1's-complement representation of 105: 001101001
 - Signed-2's-complement representation of 105: 001101001
- $(-105)_{10}$?
- Magnitude of -105 is 1101001, represent in 9 bits
 - Signed-magnitude representation of -105: 101101001
 - Signed-1's-complement representation of -105: 110010110
 - Signed-2's-complement representation of -105: 110010111

Signed Binary Numbers

- All possible four-bit signed binary numbers in the three representations.
- Which one is the best? Why?

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+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	0000	0000
-0	—	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	—	—

Signed Binary Numbers

	Addition	Representation of 0	Range
Sign-magnitude	Doesn't work $\rightarrow -6+6$ $\begin{array}{r} 1110 \\ + 0110 \\ \hline 10100 \text{ (wrong!)} \end{array}$	Two representations $\begin{array}{ll} 0000 & +0 \\ 1000 & -0 \end{array}$	$[-(2^{N-1}-1), 2^{N-1}-1]$
Signed-1's complement	Doesn't work $\rightarrow -3+6$ $\begin{array}{r} 1100 \\ + 0110 \\ \hline 10010 \text{ (wrong!)} \end{array}$	Two representations $\begin{array}{ll} 0000 & +0 \\ 1111 & -0 \end{array}$	$[-(2^{N-1}-1), 2^{N-1}-1]$
Signed-2's complement	Works $\rightarrow -3+6$ $\begin{array}{r} 1101 \\ + 0110 \\ \hline 10011 \text{ (correct!)} \end{array}$	Only one $\begin{array}{ll} 0000 & \pm 0 \\ 1000 & \text{is } -8 \end{array}$	$[-2^{N-1}, 2^{N-1}-1]$

Binary Codes

- BCD Code

- Four bits are required to code each decimal number.
 - Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
- Also known as 8-4-2-1 code, as 8, 4, 2, and 1 are the weights of the four bits of BCD.
- The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD Addition

- First add the two numbers using normal rules for binary addition.
- If the 4-bit sum is equal to or less than 9, it becomes a **valid** BCD number.
- If the 4-bit sum is greater than 9, or if a carry-out of the group is generated, it is an **invalid** result.
 - In such a case, add $(0110)_2$ or $(6)_{10}$ to the 4-bit sum in order to skip the six invalid states and return the code to BCD. If a carry results when 6 is added, add the carry to the next 4-bit group.
- Example: Consider the addition of $184 + 576 = 760$ in BCD:

BCD	1	1		
	0001	1000	0100	184
	<u>+ 0101</u>	<u>0111</u>	<u>0110</u>	+576
Binary sum	0111	10000	1010	
Add 6	<u> </u>	<u>0110</u>	<u>0110</u>	<u> </u>
BCD sum	0111	0110	0000	760

BCD Subtraction

- Same as in the binary case:
- Take the 10's complement of the subtrahend and add it to the minuend.
- Example: Consider the subtraction of $109 - 132 = -23$ in BCD:
 - Take 10's comp of 132 = 868
 - Convert difference into 10's complement

Subtraction		1		
	0001	0000	1001	109
	+1000	0110	1000	+868
Binary sum	1001	0111	10001	
Add 6	0000	0000	0110	
Difference	1001	0111	0111	977
10's complement				-23

Gray Code

- Gray Code(格雷码)
 - Minimum change code: A number changes by only one bit as it proceeds from one number to the next.
 - Error detection.
 - Representation of analog data.
 - Low power design.

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

ASCII Codes

- American Standard Code for Information Interchange (ASCII) Character Code
 - Many applications of the computer require not only handling of numbers, but also of letters.
 - To represent letters it is necessary to have a binary code for the alphabet.
 - Seven bits to code 128 characters.

$b_4b_3b_2b_1$	$b_7b_6b_5$							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	—	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

Error-Detecting Code

- Error-Detecting Code

- To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity.
- A **parity bit** (校验位) is an extra bit included with a message to make the total number of 1's either even or odd.

- Example:

	With even parity	With odd parity
ASCII A = 1000001	0 1000001	1 1000001
ASCII T = 1010100	1 1010100	0 1010100

- Suppose we use even parity

Original code	With even parity	sender	receiver	Parity check Passed?
1000001	01000001	01000001	01000001	yes
1000001	01000001	0100 0 001	0100 1 001	No
1000001	01000001	0100 0 001	0100 1 101	Yes but fails for double errors

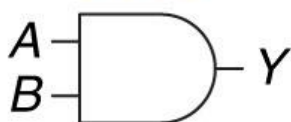
Binary Logic

- Binary logic deals with binary variables(e.g. can have two values, “0” and “1”)
- Binary variables can undergo three basic logical operators AND, OR and NOT
 - **AND** is denoted by a dot (\cdot) $z = x \cdot y$ or $z = xy$.
 - **OR** is denoted by a plus ($+$) $z = x + y$.
 - **NOT** is denoted by a single quote mark ($'$) after the variable, or an overbar ($-$) above the variable.
 - $x'y$ is pronounced as "x prime y" or "x complement y".
- Binary logic resembles binary arithmetic.
 - However, binary logic should not be confused with binary arithmetic.
 - An arithmetic variable designates a number that may consist of many digits.
 - A logic variable is always either 0 or 1.

Binary Logic

- Truth Tables, Boolean Expressions, and Logic Gates

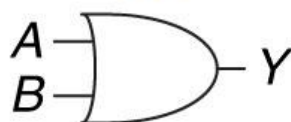
AND



$$Y = AB$$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

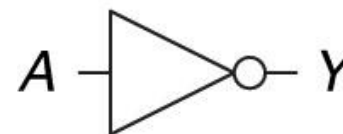
OR



$$Y = A + B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

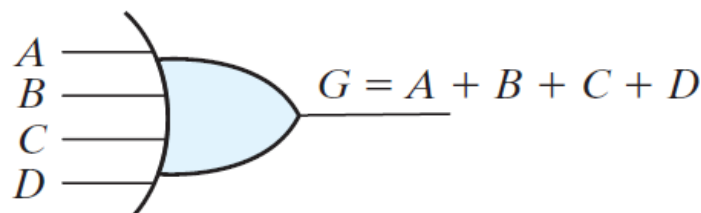
NOT



$$Y = \bar{A}$$

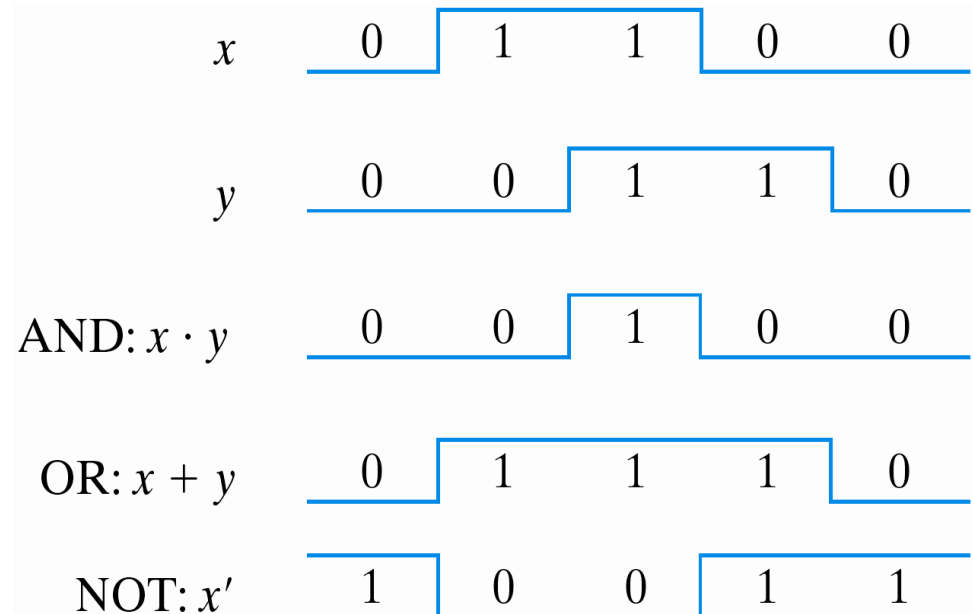
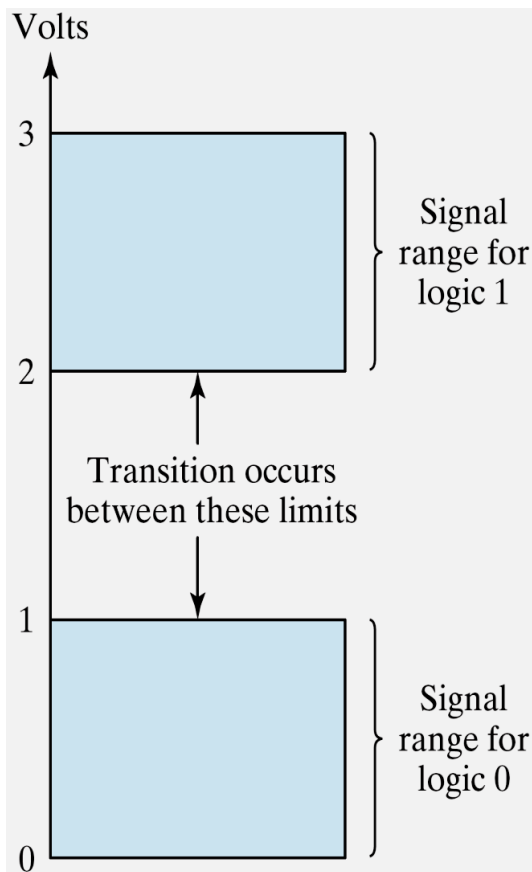
A	Y
0	1
1	0

- It is fine to have more than two inputs for AND/OR



Binary Logic

- Voltage-operated, though on a range, interpreted to be either of the two values



Input–output signals for gates

Outline

- Introduction to course
- Lecture
 - Digital Number Systems
 - Data Representation
 - Binary Logic
- **PreLab**
 - What is an FPGA

FPGA for Digital Logic

- What?
- Why?
- How?

Calculate $a + b$ using CPU

- How to calculate $a + b$?

```
int adder(int a, int b)
{
    int z = a + b;
    return z;
}
```

Compilation



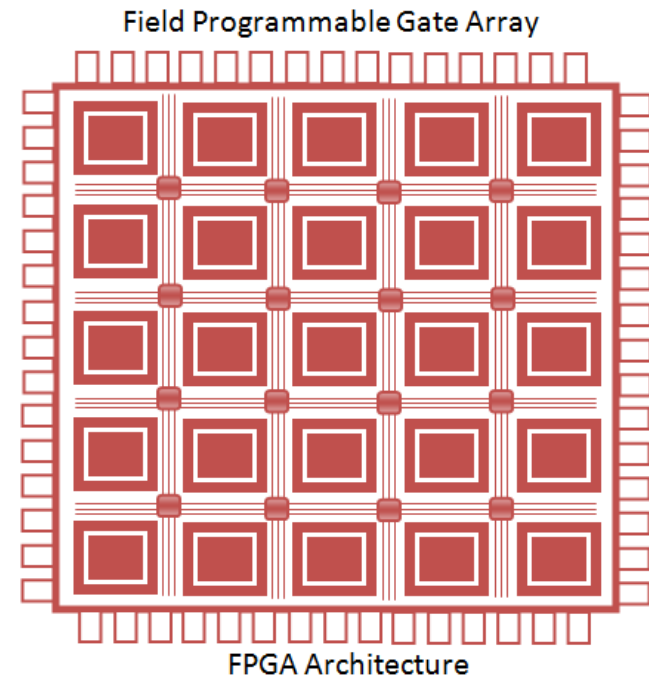
C Programming language

Calculate $a + b$ using FPGA

- How to calculate $a + b$?

```
module adder(  
    input wire [4:0] a,  
    input wire [4:0] b,  
    output wire [4:0] z  
);  
    assign z = a + b;  
endmodule
```

Synthesis



Hardware Description Language (HDL)

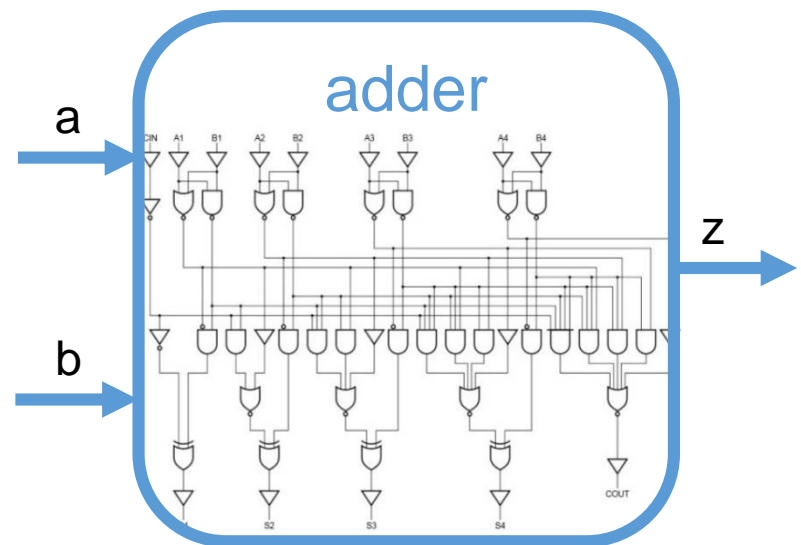
Hardware design

- These hardware blocks are comprised completely of registers and logic gates

```

module adder(
  input wire [4:0] a,
  input wire [4:0] b,
  output wire [4:0] z
);
  assign z = a + b;
endmodule
  
```

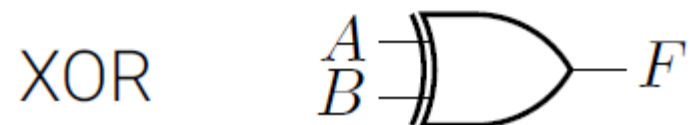
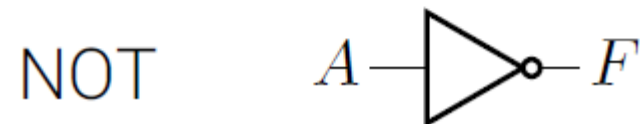
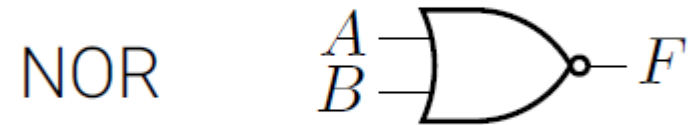
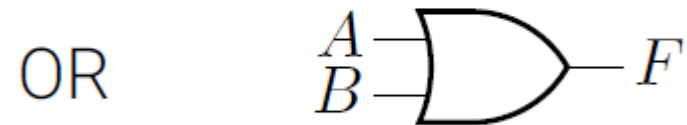
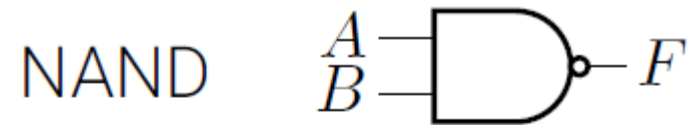
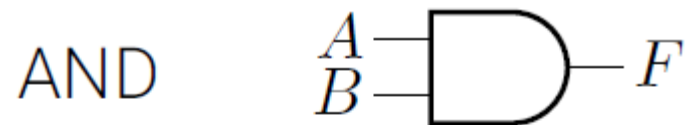
Synthesis



Hardware Description Language (HDL)

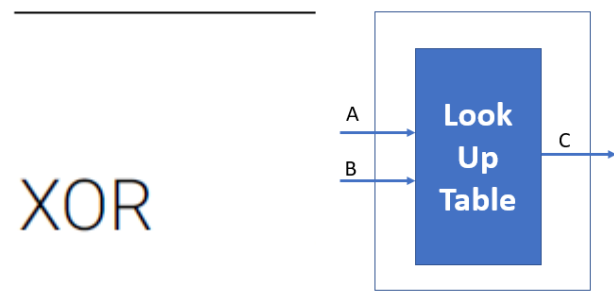
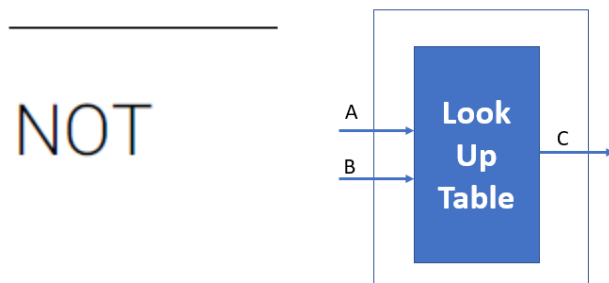
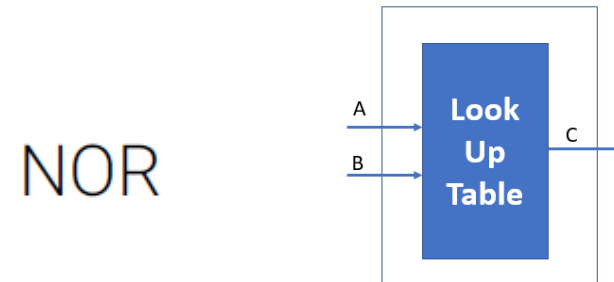
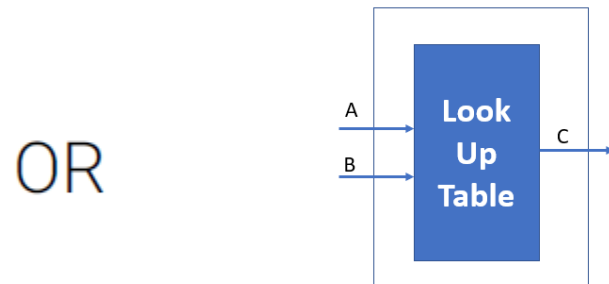
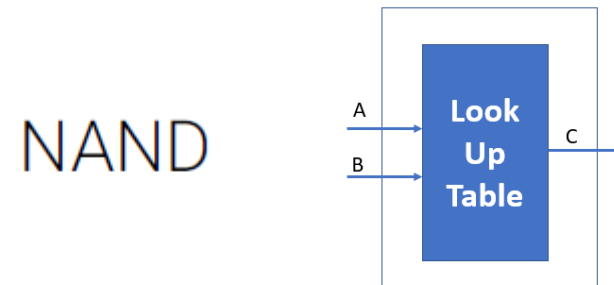
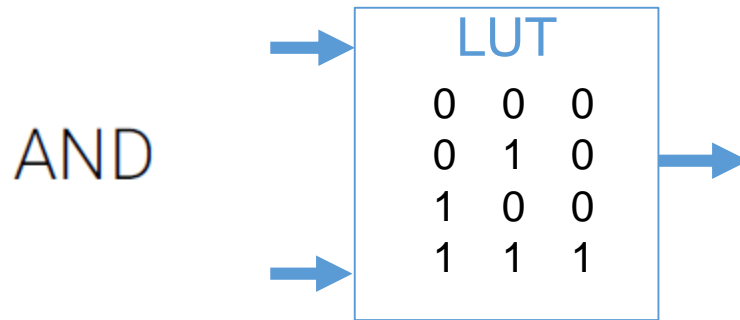
Hardware Schematic

Logic gates



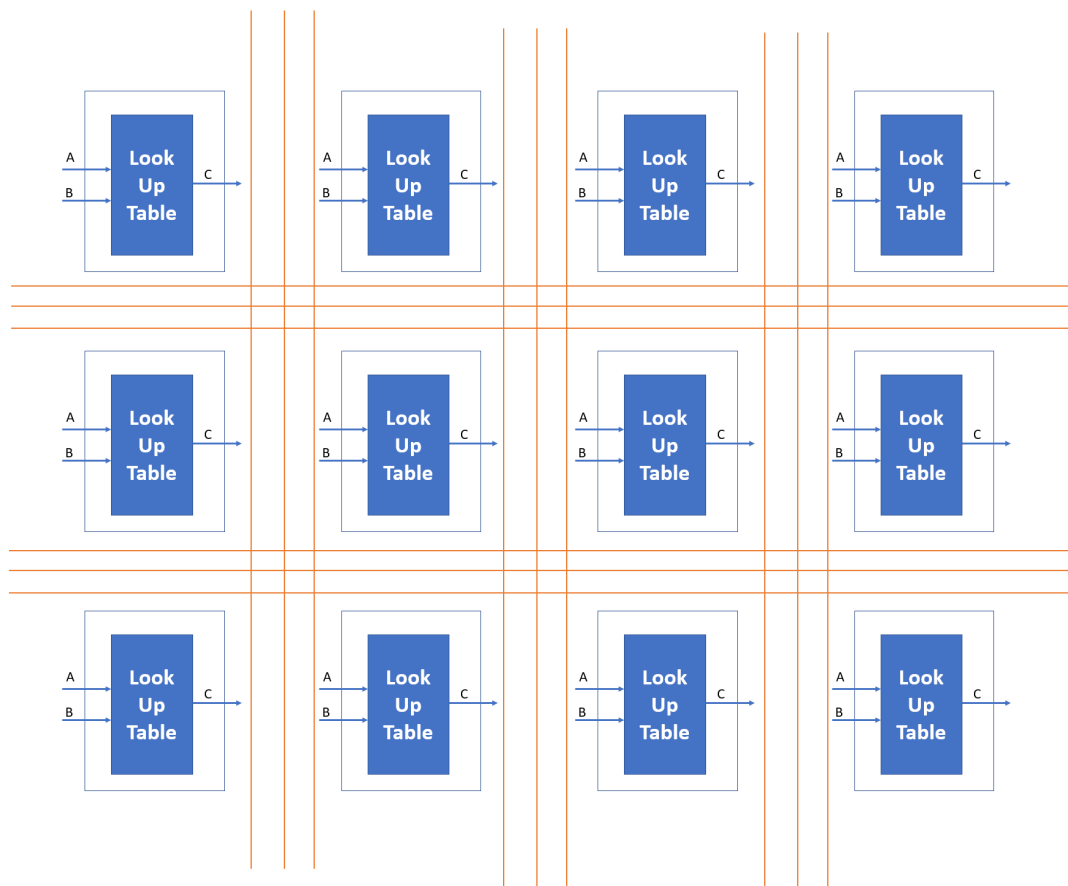
Logic gates

- The logic gates can be implemented using look-up tables.

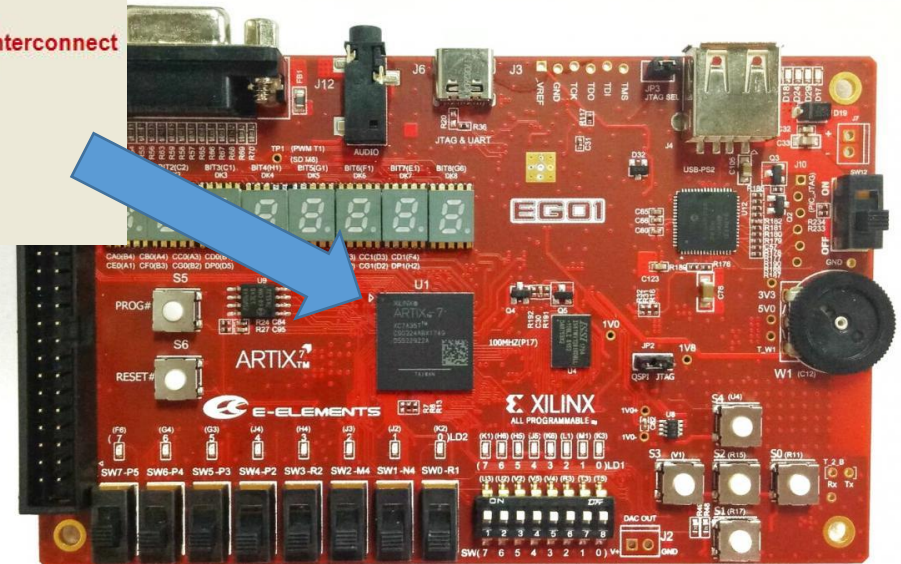
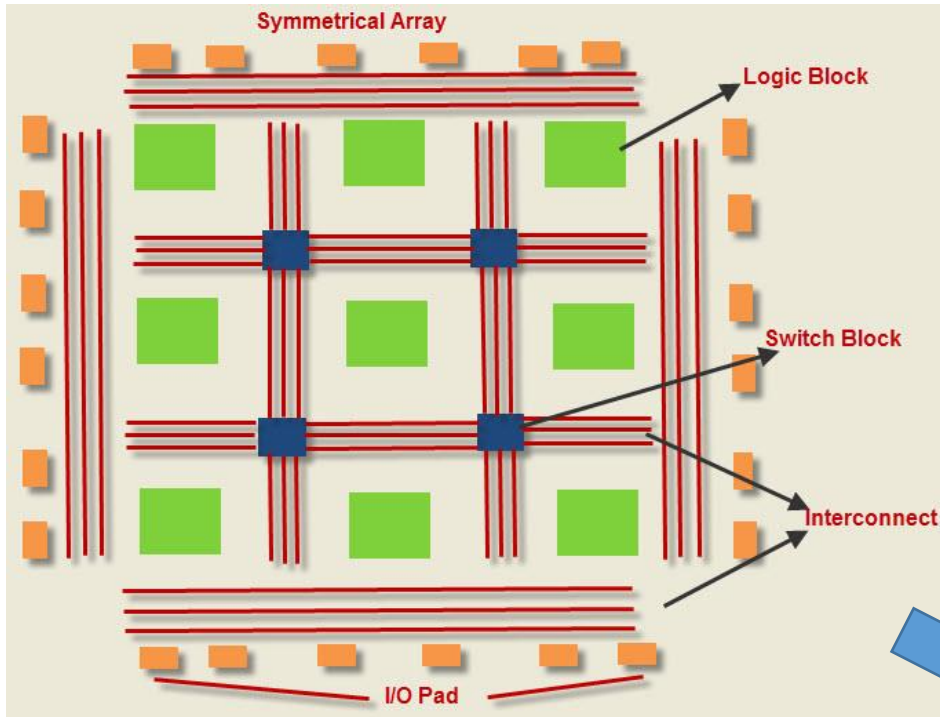


Programmable FPGA

- If you put together a bunch of look-up tables, and make them programmable, then you add a switching fabric that can connect them all together, it's just like playing with LEGO bricks !



FPGA design kit



FPGA

- What
 - A type of digital logic device that can be programmed and reprogrammed to perform a wide variety of digital functions.
- Why?
 - The programmability allows easily designing and updating designs, it provides a practical way to learn about digital system design.
- How?
 - RTL (e.g. Verilog HDL) + EDA Tools (e.g. Vivado 2017.4) + FPGA board (e.g. ego1)