Chapter 1 Introduction (I)

Chapter Outline (1/1)

- Logic Design
- A Brief Review of Number Systems
  - Hexadecimal
  - Binary Addition
  - Signed Numbers
  - Binary Subtraction
  - Binary Coded Decimal (BCD)
  - Other Codes
- Solved Problems

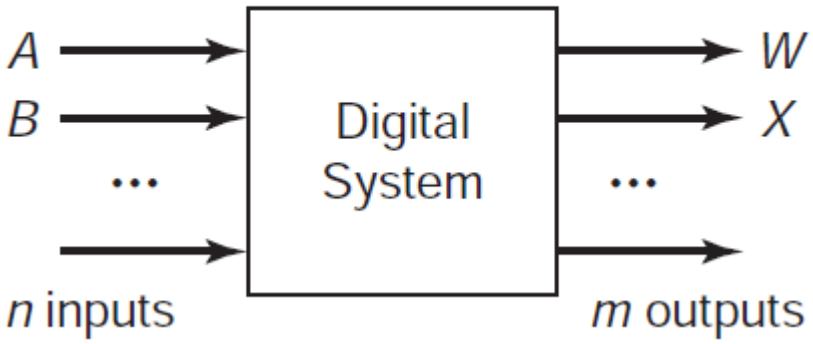
Logic Design (1/4)

- This course concerns the design of *digital systems*, that is, systems in which all the signals are represented by discrete values
- The music that we listen to on our CD players or iPods, the individual dots on a computer screen (and on the newer digital TV), and most cell phone signals are coded into strings of binary digits, referred to as bits

Logic Design (2/4)

- A simple example of digital systems is shown in Fig. 1.1
- **Example 1.1** A system with three inputs *A*, *B*, and *C*, and one output *Z*, such that *Z*=1 if and only if two of the inputs are 1
- The term *if and only if* is often abbreviated "iff". It means that the output is 1 if the condition is met and is not 1 (which means it must be 0) if the condition is not met

Figure 1.1 A digital system.



Logic Design (3/4)

- The physical manifestation of these binary quantities may be one of two voltages, for example, 0 volts (V) or ground for logic 0 and 5 V for logic 1

Table 1.1 A truth table for Example 1.1.

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

Logic Design (4/4)

- **Example 1.2** A system with eight inputs, representing two 4-bit binary numbers, and one 5-bit output, representing the sum. (Each input number can range from 0 to 15; the output can range from 0 to 31.)
- **Example 1.3** A system with one input, *A*, plus a clock, and one output *Z*, which is 1 iff the input was one at the last three consecutive clock times
- Example 1.2 is *conbinational*, that is, the output depends only on the present value of the input. Example 1.3 is *sequential*, that is, it requires memory, since we need to know something about inputs at an earlier time (previous clock times)

A Brief Review of Number Systems (1/4)

- Integers are normally written using a positional number system, in which each digit represents the coefficient in a power series

$$N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_2 r^2 + a_1 r + a_0$$

where n is the number of digits, r is the radix or base, and the a_i are the coefficients, where each is an integer in the range

$$0 \le a_i < r$$

- For decimal, r=10, and the a_i's are in the range 0 to 9. For binary, r=2, and the a_i's are all either 0 or 1. Another commonly used notation in computer documentation is hexadecimal, r=16

A Brief Review of Number Systems (2/4)

- The decimal number 7642 stands for

$$7642_{10} = 7\times10^3 + 6\times10^2 + 4\times10 + 2$$

and the binary number

$$101111_2 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2 + 1$$
$$= 32 + 8 + 4 + 2 + 1 = 47_{10}$$

Table 1.2 Powers of 2.

n	2^n	n	2^n
1	2	11	2,048
2	4	12	4,096
3	8	13	8,192
4	16	14	16,384
5	32	15	32,768
6	64	16	65,536
7	128	17	131,072
8	256	18	262,144
9	512	19	524,288
10	1,024	20	1,048,576

A Brief Review of Number Systems (3/4)

- An n-bit number can represent the positive integers from 0 to 2^n-1

Table 1.3 First 32 binary integers.

Decimal	Binary	4-bit	Decimal	Binary
0	0	0000	16	10000
1	1	0001	17	10001
2	10	0010	18	10010
3	11	0011	19	10011
4	100	0100	20	10100
5	101	0101	21	10101
6	110	0110	22	10110
7	111	0111	23	10111
8	1000	1000	24	11000
9	1001	1001	25	11001
10	1010	1010	26	11010
11	1011	1011	27	11011
12	1100	1100	28	11100
13	1101	1101	29	11101
14	1110	1110	30	11110
15	1111	1111	31	11111

A Brief Review of Number Systems (4/4)

- Most significant bit (MSB) is the leftmost bit of a binary representation; while least significant bit (LSB) is the rightmost bit of a representation
- **Example 1.8** Convert 105 to binary format. We have

105/2=52, rem 1 produces	1
52/2=26, rem 0	01
26/2=13, rem 0	001
but 13=1101	1101 001

Hexadecimal (1/2)

- Hexadecimal, often referred to as *hex* (r=16) is another base that is commonly used in computer documentation. In hexadecimal, binary digits are grouped in fours (starting at the most significant). For example, an 8-bit number,

$$N = (b_7 2^7 + b_6 2^6 + b_5 2^5 + b_4 2^4) + (b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0)$$
$$= 2^4(b_7 2^3 + b_6 2^2 + b_5 2^1 + b_4) + (b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0)$$
$$= 16h_1 + h_0$$

- The digits above 9 are represented by the first six letters of the alphabet (uppercase)

$10 \rightarrow$ A	$12 \rightarrow$ C	$14 \rightarrow$ E
$11 \rightarrow$ B	$13 \rightarrow$ D	$15 \rightarrow$ F

Hexadecimal (2/2)

- **Example 1.9** Convert binary to hexadecimal.

  $1011101010_2 = 0010\ 1110\ 1010_2$

  $= 2EA_{16}$

- **Example 1.10** Convert hexadecimal to decimal.

  $2EA_{16} = 2\times16^2 + 14\times16 + 10$

  $= 512 + 224 + 10$

  $= 746_{10}$

- Finally, to convert from decimal to hex, repeated divided by 16, producing the hex digits as the remainder (or convert to binary and then group the bits as in Example 1.9)

Binary Addition (1/4)

- Consider the addition of two bits.

Table 1.4 Binary addition.

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 (2, or a sum of 0 and a carry of 1 to the next bit)

- Consider the addition of 13 and 5. First, the least significant bits (the rightmost bits) are added. Next we must add the second digit from the right.

```
  1101        13
  0101         5
 -----       ---
 10010        18
```

Binary Addition (2/4)

- In a computer with n-bit words, when an arithmetic operation produces a result that is out of range [for example, addition of n-bit positive integers produces an $(n+1)$-bit result], it is called *overflow*
- The addition of two 1-bit operands a and b, and a 1-bit carrier carry is a three-operand problem. The addition problem then becomes

$$\begin{array}{r} c_{\text{in}} \\ a \\ b \\ \hline c_{\text{out}}\ s \end{array}$$

- A device that does this 1-bit computation is referred to a *full adder*

Binary Addition (3/4)

- Table 1.5 shows a truth table of the addition process

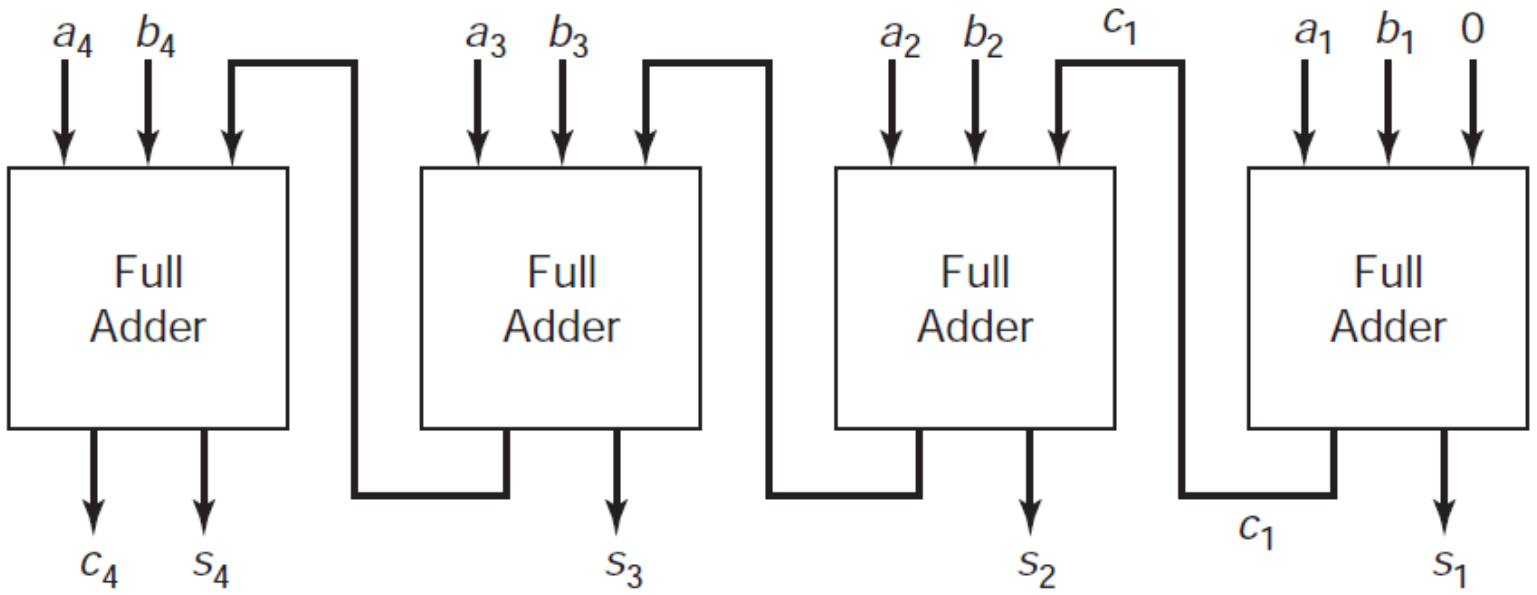
Table 1.5 One-bit adder.

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

Binary Addition (4/4)

- To add 4-bit numbers, we build four of these and connect them as shown in Fig. 1.2. The carry input of the bit 1 adder has a 0 on it, since there is no carry into that bit

Figure 1.2 A 4-bit adder.



Signed Numbers (1/6)

- Computers must deal with signed numbers. This could be incorporated into a computer, using the first bit of a number as a sign indicator (normally 0 for positive and 1 for negative) and the remaining bits for the magnitude
- Signed binary numbers are nearly always stored in *two's complement* format. The leading bit is still the sign bit (0 for positive). Positive numbers (and zeros) are just stored in normal binary. The largest number that can be stored is $2^{n-1}-1$ (7 for $n=4$). Thus, in a 4-bit system, +5 would be stored as 0101
- For positive number *a*, we just transform it into its binary form in an *n*-bit system. For example, 3 is stored as the 0011

Signed Numbers (2/6)

- The negative number, $-a$, is stored as the binary equivalent of $2^n - a$ in an n-bit system. Thus, for example, -3 is stored as the binary for 16-3=13, that is, 1101
- The most negative number that can be stored is -2^{n-1} (-8 in a 4-bit system)
- An easier way to find the storage format for negative numbers in two's complement is the following three-step approach:

1. Find the binary equivalent of the magnitude
2. Complement each bit (that is, change 0's to 1's and 1's to 0's)
3. Add 1

Signed Numbers (3/6)

- **Example 1.13** Represent -5, -1, and -0 using two's complement.

	−5	−1	−0
1.	5: 0 1 0 1	1: 0 0 0 1	0: 0 0 0 0
2.	1 0 1 0	1 1 1 0	1 1 1 1
3.	1	1	1
	−5: 1 0 1 1	−1: 1 1 1 1	0 0 0 0

- Note -0 in two's complement is the same as +0.

Signed Numbers (4/6)

- Table 1.6 lists the meaning of all 4-bit numbers both as positive (unsigned) numbers and as two's complement (signed numbers).

Table 1.6 Signed and unsigned 4-bit numbers.

Binary	Positive	Signed (two's complement)
0000	0	0
0001	1	+ 1
0010	2	+ 2
0011	3	+ 3
0100	4	+ 4
0101	5	+ 5
0110	6	+ 6
0111	7	+ 7
1000	8	− 8
1001	9	− 7
1010	10	− 6
1011	11	− 5
1100	12	− 4
1101	13	− 3
1110	14	− 2
1111	15	− 1

Signed Numbers (5/6)

- The reason that two's complement is so popular is the simplicity of addition. To add any two numbers, no matter what the sign of each is, we just do binary addition on their representations
- **Example 1.15** Three cases of addition using two's complement.

−5		1 0 1 1	−5		1 0 1 1	−5		1 0 1 1
+7		0 1 1 1	+5		0 1 0 1	+3		0 0 1 1
+2	(1)	0 0 1 0	0	(1)	0 0 0 0	−2	(0)	1 1 1 0

Signed Numbers (6/6)

- Overflow occurs when the sum is out of range. For 4-bit numbers, the range is $-8 \leqq \text{sum} \leqq +7$. Examples 1.16 and 1.17 show overflow produces a wrong answer.
- **Example 1.16**

+5		0 1 0 1	
+4		0 1 0 0	
	(0)	1 0 0 1	(looks like −7)

- **Example 1.17**

−5		1 0 1 1	
−4		1 1 0 0	
	(1)	0 1 1 1	(looks like +7)

Binary Subtraction (1/1)

- Subtraction (no matter dealing with signed or unsigned numbers) is generally accomplished by first taking the two's complement of the second operand, and then adding. Thus, a-b is computed as $a+(-b)$
- **Example 1.20** Compute 14-10.

  Note $14_{10}=1110_2$ and $10_{10}=1010_2$.

$$
\begin{array}{r}
 1 \\
 1\,1\,1\,0 \\
+0\,1\,0\,1 \\
\hline
(1)\,0\,1\,0\,0 = 4
\end{array}
$$

Binary Coded Decimal (BCD) (1/2)

- 8421 code, 5421 code, and 2421 codes are weighted code.

Table 1.7 Binary-coded decimal codes.

Decimal digit	8421 code	5421 code	2421 code	Excess 3 code	2 of 5 code
0	0000	0000	0000	0011	11000
1	0001	0001	0001	0100	10100
2	0010	0010	0010	0101	10010
3	0011	0011	0011	0110	10001
4	0100	0100	0100	0111	01100
5	0101	1000	1011	1000	01010
6	0110	1001	1100	1001	01001
7	0111	1010	1101	1010	00110
8	1000	1011	1110	1011	00101
9	1001	1100	1111	1100	00011
unused	1010	0101	0101	0000	any of
	1011	0110	0110	0001	the 22
	1100	0111	0111	0010	patterns
	1101	1101	1000	1101	with 0, 1,
	1110	1110	1001	1110	3, 4, or 5
	1111	1111	1010	1111	1's

Binary Coded Decimal (BCD) (2/2)

- The first code is referred to as the 8421 code, since those are the weights of the bits. Each decimal digital is represented by

$$8\times a_3 + 4\times a_2 + 2\times a_1 + 1\times a_0$$

- Excess 3 (XS3) where the decimal digital is represented by the binary equivalent of 3 more than the digit. For example, 0 is stored as the binary 3 (0011) and 6 as the binary of 6+3=9 (1001).
- The final column shows a 2 of 5 code, where each digit is represented by a 5-bit number, two of which are 1 (and the remaining three bits are 0). This provides some error detection capabilities.

Other Codes (1/3)

- Alphanumeric information is transmitted using the American Standard Code for Information Interchange (ASCII). Seven digits are used to represent the various characters on the standard keyboard as well as a number of control signals (such as carriage return)
- Refer to Table 1.8, "Logic" is encoded as

1001100	1101111	1100111	1101001	1100011
L	o	g	i	c

Other Codes (2/3)

Table 1.8 ASCII code.

$a_3a_2a_1a_0$	$a_6a_5a_4$ = 010	011	100	101	110	111
0000	space	0	@	P	`	p
0001	!	1	A	Q	a	q
0010	“	2	B	R	b	r
0011	#	3	C	S	c	s
0100	$	4	D	T	d	t
0101	%	5	E	U	e	u
0110	&	6	F	V	f	v
0111	‘	7	G	W	g	w
1000	(	8	H	X	h	x
1001	)	9	I	Y	i	y
1010	*	:	J	Z	j	z
1011	+	;	K	[	k	{
1100	,	<	L	\	l	|
1101		=	M	]	m	}
1110	.	>	N	^	n	~
1111	/	?	O	_	o	delete

Other Codes (3/3)

- In a Gray code, consecutive number differ in only one bit.

Table 1.9 Gray code.

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