ELECTRICAL
Engineering KMITNB

Digital Circuits and Logic Design

Lecture6-3 :Number Representation and Arithmetic Circuits (VHDL)

Design **using schematic capture**

- One way to design an arithmetic circuit is via schematic capture, drawing all necessary logic gates
- To create an *n*-bit adder
  - Start with a single full adder
  - Chain together *n* instances of this to produce the *n*-bit adder
  - If a CLA adder is desired, add carry lookahead logic
- Design process becomes complex rapidly
- A better approach to to use predefined subcircuits
  - CAD tools provide a library of basic logic gates
  - Most CAD tools also provide a library of commonly used circuits, such as adders
  - Each subcircuit is provided as a module that can be imported into a schematic and used as a part of a larger circuit

Macro- **and megafunctions**

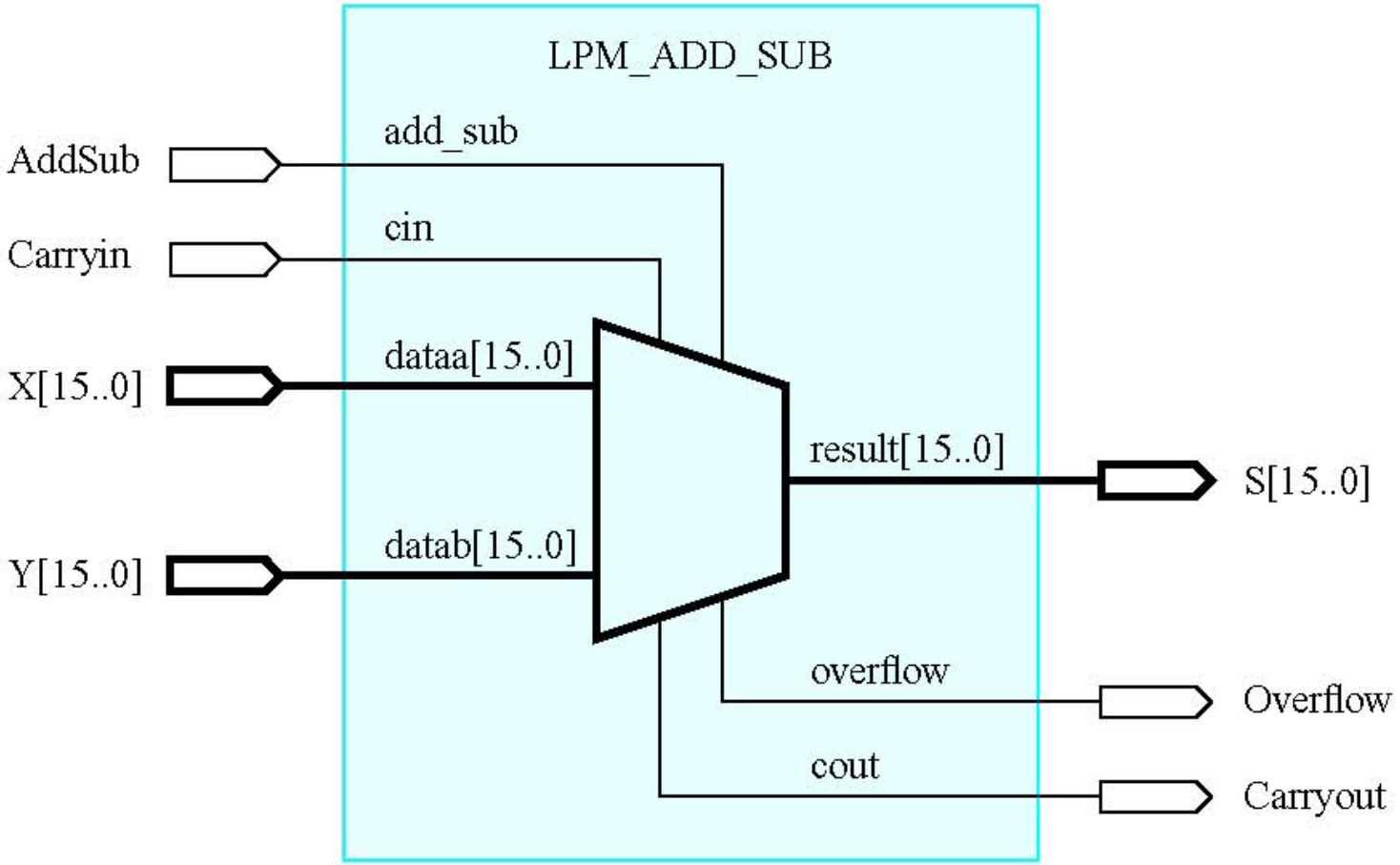
- In some CAD systems these library functions are called ***macrofunctions or megafunctions***
- Two primary types of macrofunctions:
  - Technology-dependent: designed to suit a specific type of chip (such as a particular FPGA)
  - Technology-independent: implemented in any type of chip, with different circuits for different types of chips
- A good example of a library of macrofunctions is the ***Library of Parameterized Modules (LPM) as a*** part of the Quartus II system
  - Each module is technology independent
  - Each module is parameterized: it can be used in a variety of ways

ELECTRICAL
Engineering KMITNB

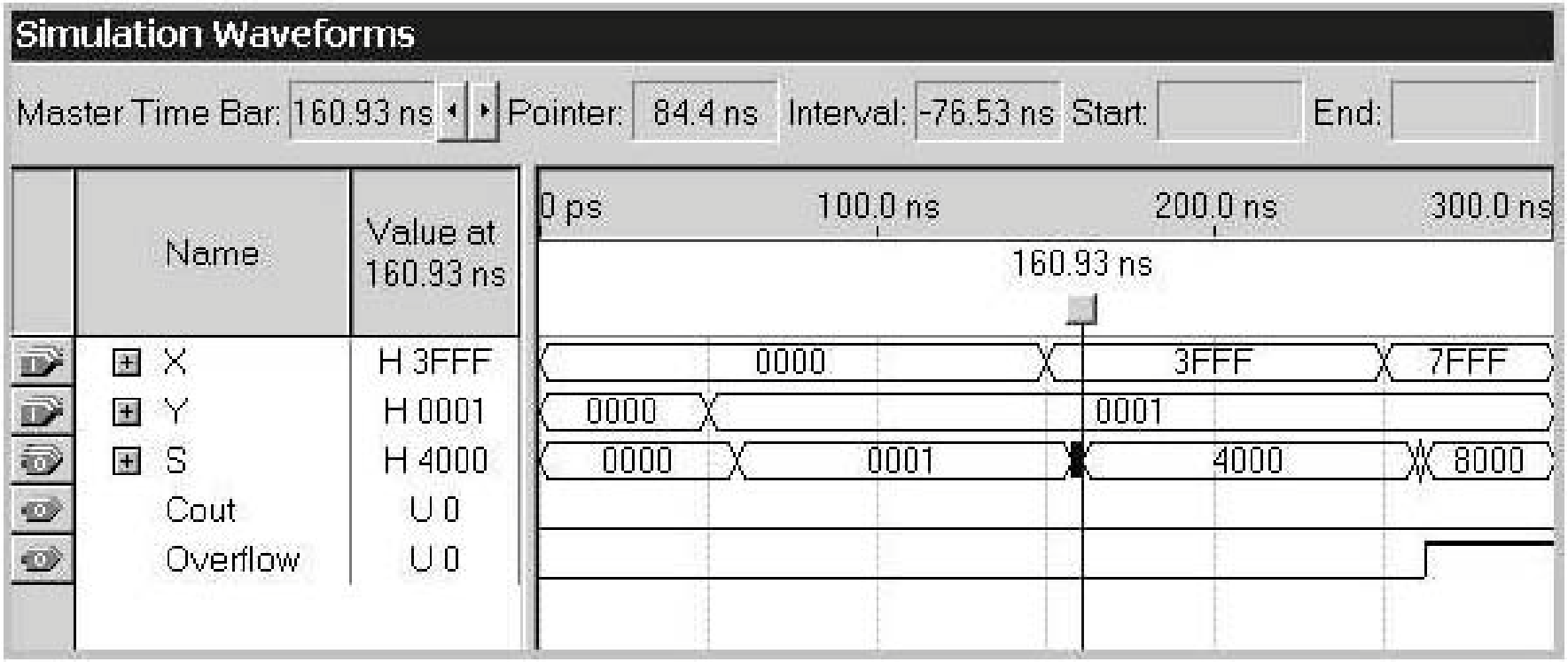
LPM_ADD_SUB

- The LPM library includes an *n*-bit adder named **LPM_ADD_SUB**
  - Implements a basic add/subtract circuit
  - The number of bits, *n*, is set by a parameter **LPM_WIDTH**
  - Another parameter **LPM_REPRESENTATION**, determines whether the numbers are treated as unsigned or signed

An adder using LPM_ADD_SUB



Simulation results for the LPM adder



ELECTRICAL Engineering KMITNB

Design using VHDL

- We can use a hierarchical approach in designing VHDL code
  - First construct a VHDL entity for a full adder
  - Use multiple ***instances*** to create a multi-bit adder
- In VHDL, a logic signal is represented as a data object
  - We used a **BIT** data type before that could only take on the values 0 and 1
  - Another data type, **STD_LOGIC**, is actually preferable because it can assume several different values [0, 1, Z (high impedance), - (don't care)]
- We must declare the library where the data type exists, and declare that we will use the data type

```
LIBRARY     ieee;
USE         ieee.std_logic_1164.all;
```

VHDL full adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladd IS
    PORT (  Cin, x, y  : IN    STD_LOGIC ;
            s, Cout    : OUT   STD_LOGIC ) ;
END fulladd ;

ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
    s <= x XOR y XOR Cin ;
    Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y) ;
END LogicFunc ;
```

VHDL 4-bit ripple carry adder

ENTITY construct

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY adder4 IS
    PORT (   Cin          : IN   STD_LOGIC ;
             x3, x2, x1, x0   : IN   STD_LOGIC ;
             y3, y2, y1, y0   : IN   STD_LOGIC ;
             s3, s2, s1, s0   : OUT STD_LOGIC ;
             Cout             : OUT STD_LOGIC ) ;
END adder4 ;
```

VHDL 4-bit ripple carry adder

ARCHITECTURE construct

```
ARCHITECTURE Structure OF adder4 IS
    SIGNAL c1, c2, c3 : STD_LOGIC ;
    COMPONENT fulladd
        PORT (    Cin, x, y  : IN   STD_LOGIC ;
                  s, Cout    : OUT STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    stage0: fulladd PORT MAP ( Cin, x0, y0, s0, c1 ) ;
    stage1: fulladd PORT MAP ( c1, x1, y1, s1, c2 ) ;
    stage2: fulladd PORT MAP ( c2, x2, y2, s2, c3 ) ;
    stage3: fulladd PORT MAP (
        Cin => c3, Cout => Cout, x => x3, y => y3, s => s3  ) ;
END Structure ;
```

ELECTRICAL Engineering KMITNB

New VHDL syntax

- There are several new constructs in the previous VHDL code
- **SIGNAL c1, c2, c3 : STD_LOGIC ;**
  - Appears in the ARCHITECTURE construct
  - Basically defines signals that will be used internal to the design (i.e. not specifically an IN or an OUT signal as appears in the PORT statement)
- **COMPONENT fulladd**
  - Appears in the ARCHITECTURE construct
  - Defines the PORT for a subcircuit (component) that is defined in another file (fulladd.vhd in this example)
  - The VHDL file (fulladd.vhd) should normally be in the same directory as the file adder4.vhd

ELECTRICAL
Engineering KMITNB

VHDL packages

- A VHDL package can be created for a component (subcircuit) such that the COMPONENT statement is not explicitly required when creating instances of the component in another file

VHDL packages

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

PACKAGE fulladd_package IS
    COMPONENT fulladd
            PORT (   Cin, x, y  : IN STD_LOGIC ;
                     s, Cout    : OUT  STD_LOGIC ) ;
    END COMPONENT ;
END fulladd_package ;
```

Usually compiled as a separate file in the same directory as fulladd.vhd

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.fulladd_package.all ;

ENTITY adder4 IS
    PORT ( Cin                  : IN    STD_LOGIC ;
           x3, x2, x1, x0       : IN    STD_LOGIC ;
           y3, y2, y1, y0       : IN    STD_LOGIC ;
           s3, s2, s1, s0       : OUT   STD_LOGIC ;
           Cout                 : OUT   STD_LOGIC ) ;
END adder4 ;

ARCHITECTURE Structure OF adder4 IS
    SIGNAL c1, c2, c3 : STD_LOGIC ;
BEGIN
    stage0: fulladd PORT MAP ( Cin, x0, y0, s0, c1 ) ;
    stage1: fulladd PORT MAP ( c1, x1, y1, s1, c2 ) ;
    stage2: fulladd PORT MAP ( c2, x2, y2, s2, c3 ) ;
    stage3: fulladd PORT MAP (
           Cin => c3, Cout => Cout, x => x3, y => y3, s => s3  ) ;
END Structure ;
```

ELECTRICAL
Engineering KMITNB

Numbers in VHDL

- A number in VHDL is a multibit SIGNAL data object
  SIGNAL C: STD_LOGIC_VECTOR(1 TO 3)
- C is a 3-bit STD_LOGIC signal
  C – a 3-bit quantity
  C <= "100";
  C(1) – a 1-bit quantity (the most significant bit)
  C(2) – a 1-bit quantity
  C(3) - a 1-bit quantity (the least significant bit)
- The ordering of the bits can be reversed
  SIGNAL X: STD_LOGIC_VECTOR(3 DOWNTO 0)
- X is a 4-bit STD_LOGIC signal
  - X(3) is the most significant bit
  - X(0) is the least significant bit

ELECTRICAL
Engineering KMITNB

Numbers in VHDL

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.fulladd_package.all ;
ENTITY adder4 IS
    PORT (   Cin : IN    STD_LOGIC ;
             X, Y    : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
             S       : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
             Cout    : OUT   STD_LOGIC ) ;
END adder4 ;

ARCHITECTURE Structure OF adder4 IS
    SIGNAL C : STD_LOGIC_VECTOR(1 TO 3) ;
BEGIN
    stage0: fulladd PORT MAP ( Cin, X(0), Y(0), S(0), C(1) ) ;
    stage1: fulladd PORT MAP ( C(1), X(1), Y(1), S(1), C(2) ) ;
    stage2: fulladd PORT MAP ( C(2), X(2), Y(2), S(2), C(3) ) ;
    stage3: fulladd PORT MAP ( C(3), X(3), Y(3), S(3), Cout ) ;
END Structure ;
```

Behavioral VHDL descriptions

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
    PORT ( X, Y   : IN    STD_LOGIC_VECTOR(15 DOWNTO 0) ;
           S      : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
BEGIN
    S <= X + Y ;
END Behavior ;
```

Allows use of STD_LOGIC signals as signed values

STD_LOGIC signals can be used with Arithmetic operator

We are really describing the behavior of the circuit

The VHDL arithmetic package

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;
ENTITY adder16 IS
    PORT (Cin              : IN  STD_LOGIC ;
          X, Y             : IN  SIGNED(15 DOWNTO 0) ;
          S                : OUT SIGNED(15 DOWNTO 0) ;
          Cout, Overflow : OUT STD_LOGIC ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
    SIGNAL Sum : SIGNED(16 DOWNTO 0) ;
BEGIN
    Sum <= ('0' & X) + Y + Cin ;
    S <= Sum(15 DOWNTO 0) ;
    Cout <= Sum(16) ;
    Overflow <= Sum(16) XOR X(15) XOR Y(15) XOR Sum(15) ;
END Behavior ;
```

Defines SIGNED and UNSIGNED Data Type

Concatenate Operator

Integer Data Object

```
ENTITY adder16 IS
    PORT ( X, Y     : IN     INTEGER RANGE -32768 TO 32767 ;
           S        : OUT    INTEGER RANGE -32768 TO 32767 ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
BEGIN
    S <= X + Y ;
END Behavior ;
```

No Library or USE clause appears in the code, because the INTEGER type is predefined in standard VHDL.

Binary-coded-decimal numbers

- It is possible to represent decimal numbers simply by encoding each decimal digit in binary form
  - Called ***binary-coded-decimal*** (BCD)
- Because there are 10 digits to represent, it is necessary to use four bits per digit
  - From 0=0000 to 9=1001
  - $(01111000)_{BCD}=(78)_{10}$
- BCD representation was used in some early computers and many handheld calculators
  - Provides a format that is convenient when numerical information is to be displayed on a simple digit-oriented display

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

Addition of BCD digits

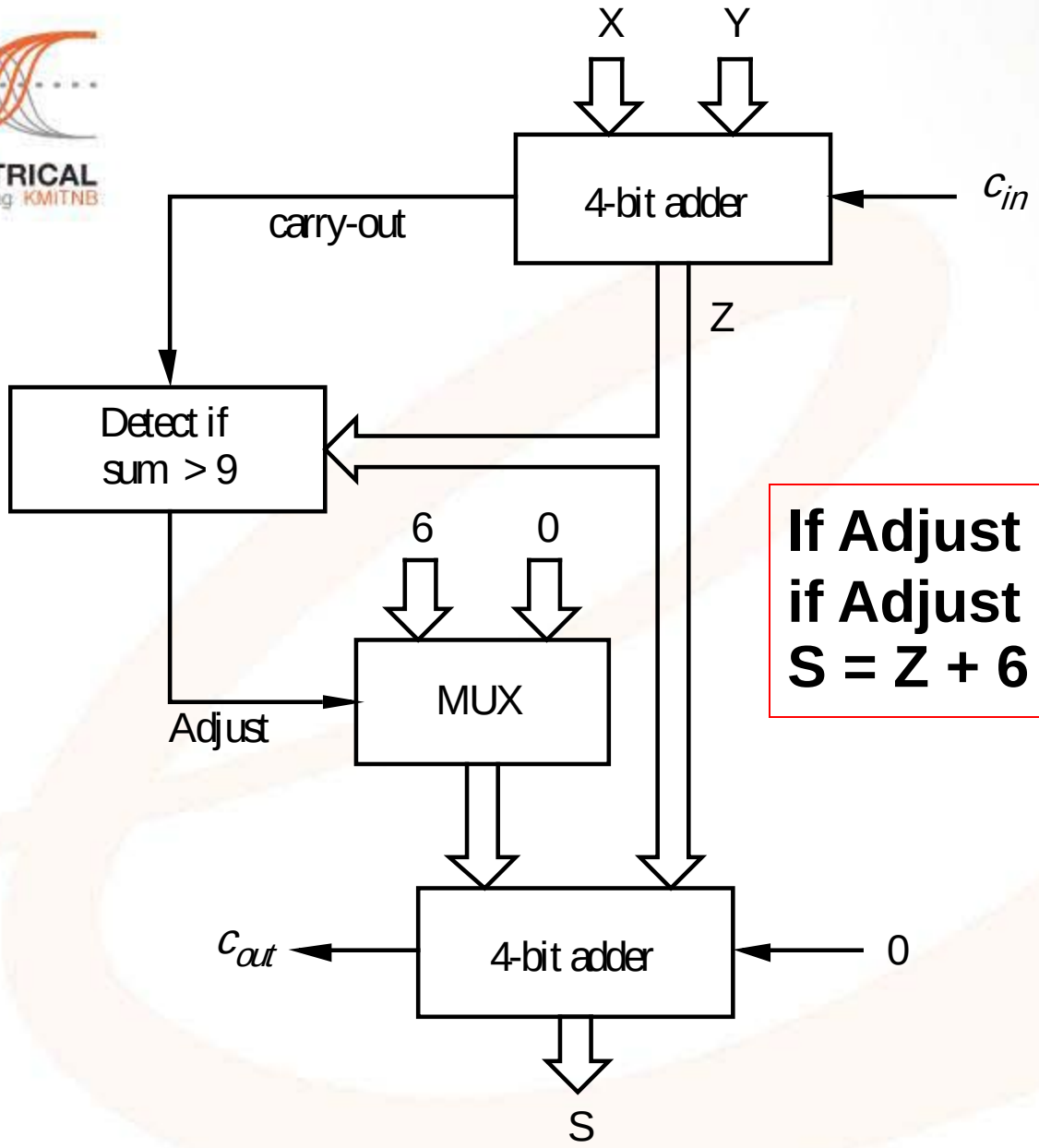
$$Z = X + Y$$

If $Z \leq 9$, then $S = Z$ and carry-out $= 0$

if $Z > 9$, then $X = Z + 6$ and carry-out $= 1$

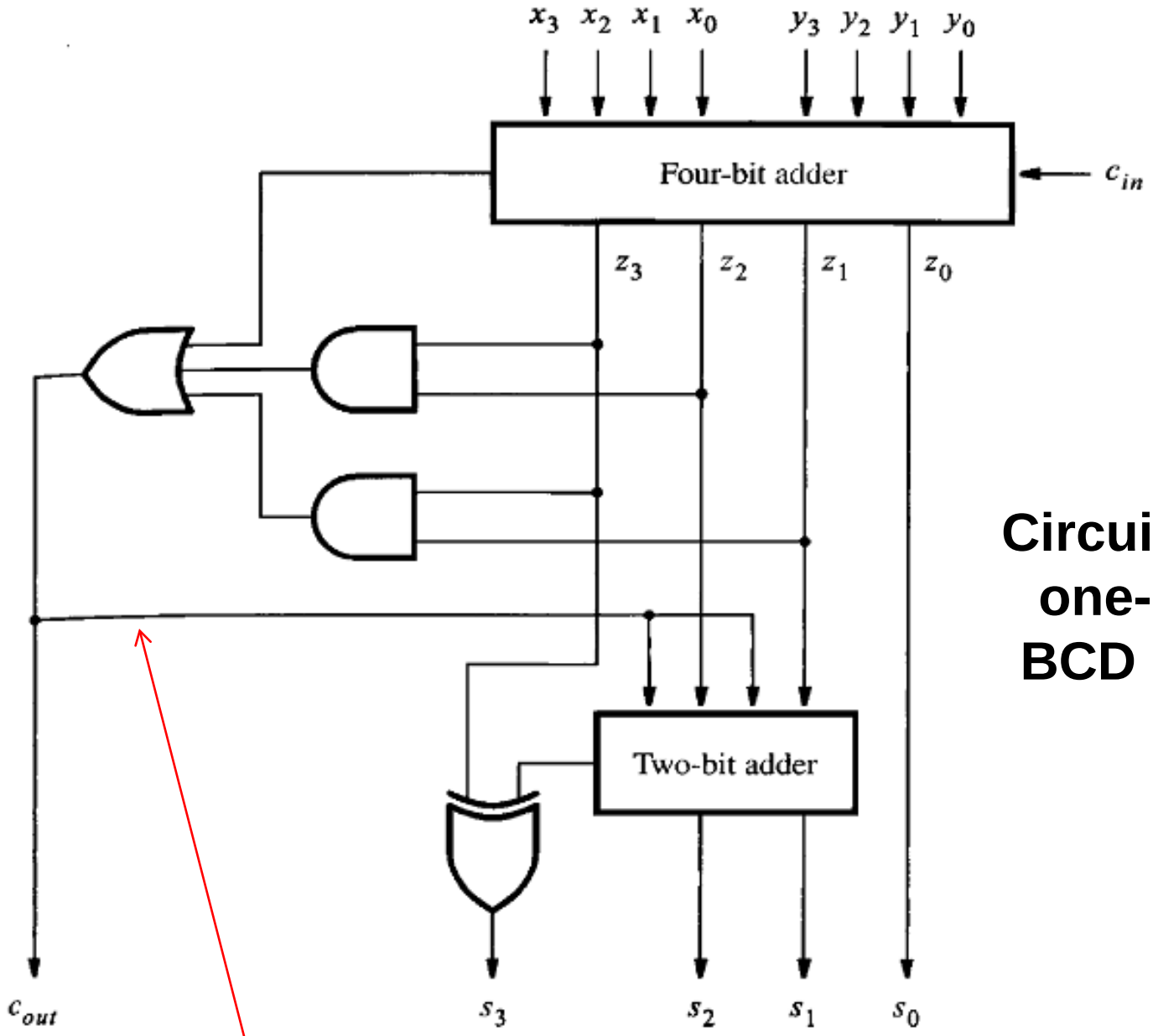
X	0 1 1 1	7
+ Y	+ 0 1 0 1	+ 5
Z	1 1 0 0	12
	+ 0 1 1 0	
carry →	1 0 0 1 0	
	S = 2	

X	1 0 0 0	8
+ Y	+ 1 0 0 1	+ 9
Z	1 0 0 0 1	17
	+ 0 1 1 0	
carry →	1 0 1 1 1	
	S = 7	



If Adjust = 0, then S = Z + 0;
if Adjust = 1, then
S = Z + 6 and carry-out = 1 ;

Block diagram for a one-digit BCD adder.



Circuit for a one-digit BCD adder

Adjust = Carry-out + $z_3(z_2 + z_1)$

VHDL code for a one-digit BCD adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY BCD IS
    PORT ( X, Y     : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           S        : OUT    STD_LOGIC_VECTOR(4 DOWNTO 0) ) ;
END BCD ;

ARCHITECTURE  Behavior  OF  BCD  IS
    SIGNAL  Z       : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
    SIGNAL  Adjust : STD_LOGIC ;
BEGIN
    Z <= ('0' & X) + Y ;
    Adjust <= '1' WHEN Z > 9 ELSE '0' ;
    S <= Z WHEN (Adjust = '0') ELSE Z + 6 ;
END Behavior ;
```

ELECTRICAL
Engineering KMITNB

ASCII character code

- The most popular code for representing information in computers is used for both numbers and letters and some control codes
- It is the ***American Standard Code for Information Interchange*** (ASCII) code
- ASCII code uses seven-bit patterns to represent 128 different symbols including
  - Digits (0-9)
  - Lowercase (a-z) and uppercase (A-Z) characters
  - Punctuation marks and other commonly used symbols
  - Control codes
- The 8-bit extended ASCII code is used to represent all of the above and another 128 graphics characters

ELECTRICAL
Engineering KMITNB

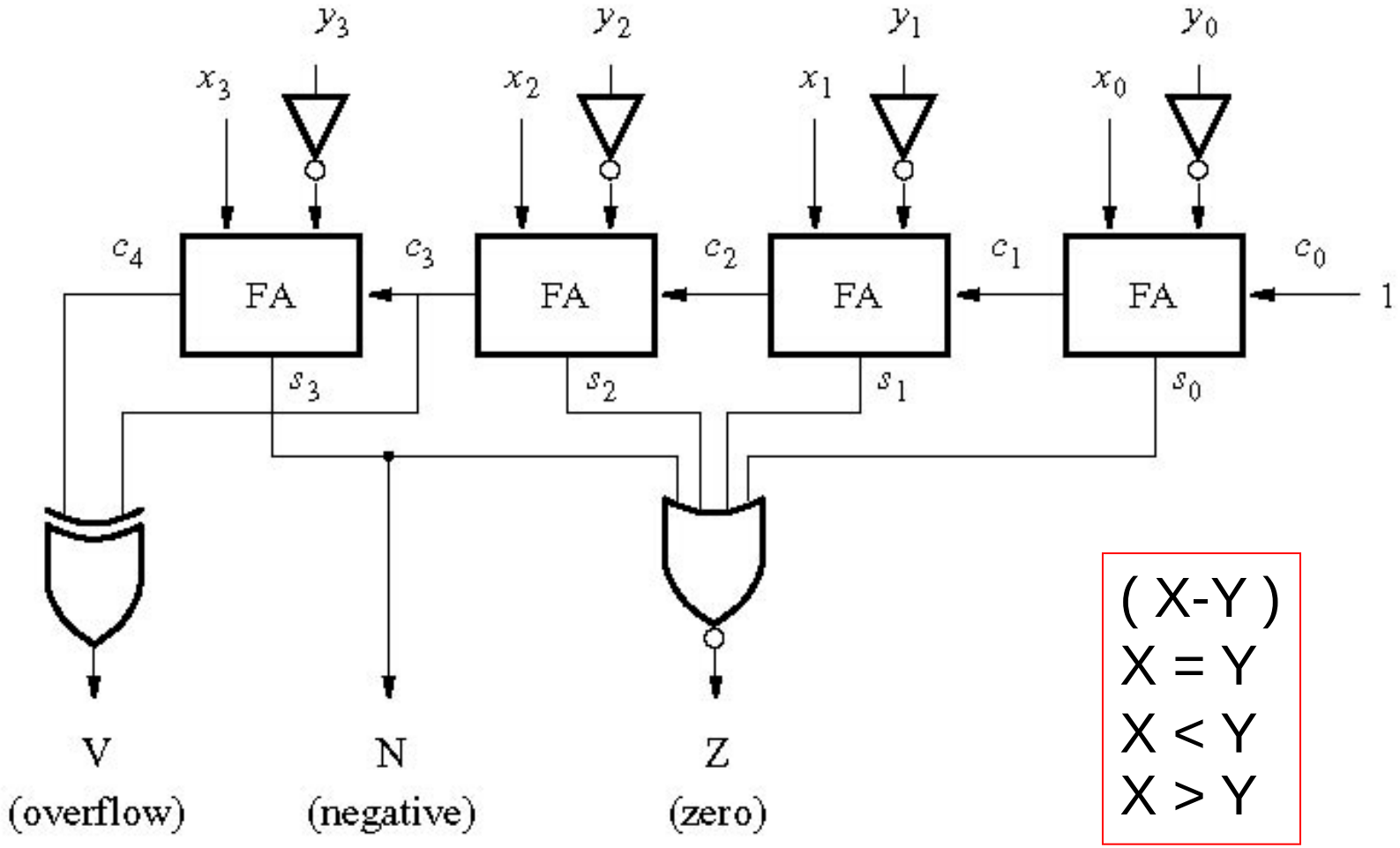
ASCII Code

Bit positions 3210	Bit positions 654							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	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	\	1	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

NUL	Null/Idle	SI	Shift in
SOH	Start of header	DLE	Data link escape
STX	Start of text	DC1-DC4	Device control
ETX	End of text	NAK	Negative acknowledgement
EOT	End of transmission	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmitted block
ACQ	Acknowledgement	CAN	Cancel (error in data)
BEL	Audible signal	EM	End of medium
BS	Back space	SUB	Special sequence
HT	Horizontal tab	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tab	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carriage return	US	Unit separator
SO	Shift out	DEL	Delete/Idle

Bit positions of code format = | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Comparator Circuit



(X-Y)
X = Y
X < Y
X > Y

ELECTRICAL
Engineering KMITNB

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.fulladd_package.all ;

ENTITY comparator IS
      PORT ( X, Y     : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
                V, N, Z  : OUT STD_LOGIC ) ;
END comparator ;

ARCHITECTURE Structure OF comparator IS
      SIGNAL S : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
      SIGNAL C : STD_LOGIC_VECTOR(1 TO 4) ;
BEGIN
      stage0: fulladd PORT MAP ( '1', X(0), NOT Y(0), S(0), C(1) )
;
      stage1: fulladd PORT MAP ( C(1), X(1), NOT Y(1), S(1),
C(2) ) ;
      stage2: fulladd PORT MAP ( C(2), X(2), NOT Y(2), S(2),
C(3) ) ;
      stage3: fulladd PORT MAP ( C(3), X(3), NOT Y(3), S(3),
C(4) ) ;
      V <= C(4) XOR C(3) ;
      N <= S(3) ;
      Z <= '1' WHEN S(3 DOWNTO 0) = "0000" ELSE '0';
END Structure ;
```

Structural VHDL code for the comparator circuit.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY comparator IS
    PORT ( X, Y     : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           V, N, Z  : OUT   STD_LOGIC ) ;
END comparator ;

ARCHITECTURE Behavior OF comparator IS
    SIGNAL S : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
BEGIN
    S <= ('0' & X) - Y ;
    V <= S(4) XOR X(3) XOR Y(3) XOR S(3) ;
    N <= S(3) ;
    Z <= '1' WHEN S(3 DOWNTO 0) = 0 ELSE '0';
END Behavior ;
```

Behavioral VHDL code for the comparator circuit.

ELECTRICAL
Engineering KMITNB

H/W

1. Simulate structural and behavioral VHDL code for the comparator circuit with Quartus II. Explain and show waveform table for x=y,x>y, and x<y.