University of British Columbia
Electrical and Computer Engineering
Digital Design and Microcomputers CPEN312

Lecture 7: Arithmetic Operations.

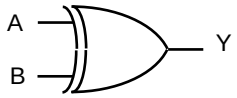
Dr. Jesús Calviño-Fraga. P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

January 24, 2017

Objectives

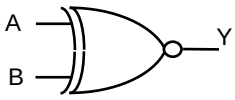
- The XOR gate
- Half adder
- Full adder
- Binary addition and subtraction
- BCD addition and subtraction

XOR and XNOR Gates



$$Y = A \oplus B$$

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



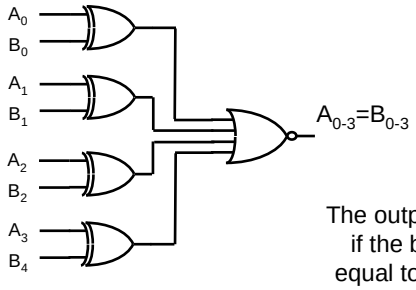
$$\overline{Y = A \oplus B}$$

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

Uses of the XOR Gate

- The XOR gate is very useful. Some uses we will see:
  - Comparators. Check if two numbers are equal. Example given in a previous lecture.
  - Selectable inverters. They allow us to select A or A' easily.
  - Parity checkers. Often used when transmitting data digitally.
  - Arithmetic: addition and subtraction.

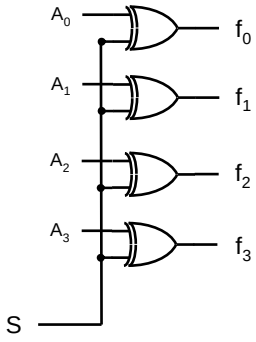
XOR Comparator



The output is one only if the bits of A are equal to the bits of B

$$f = \overline{(A_0 \oplus B_0) + (A_1 \oplus B_1) + (A_2 \oplus B_2) + (A_3 \oplus B_3)}$$

Selectable Inverter



S	A_n	f_n
0	0	0
0	1	1
1	0	1
1	1	0

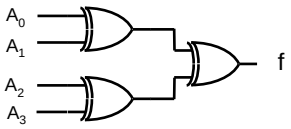
When S=1 we invert A

Parity Generator

Consider this function:

$$f = A_0 \oplus A_1 \oplus A_2 \oplus A_3$$

Implemented like this:



What is particular about f?

f is one when the number of ones in A is odd!

A_3	A_2	A_1	A_0	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

The Half Adder

- Before we start with adder circuits, let make sure we understand how to add binary numbers.

```
179      10110011   plus
112      01110000
         --------
291     100100011
```

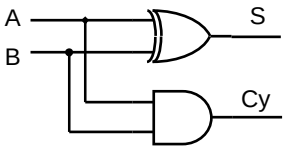
Pay attention on how the carry propagates from one bit to the next!

The Half Adder

- The half adder has two inputs: A and B; and two outputs: Sum (S), and Carry (Cy). Here it is the truth table of a half adder:

In		Out	
A	**B**	**S**	**Cy**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Circuit for half adder:



The Full Adder

- The full adder has three inputs: A, B, and Cy_i; and two outputs: Sum (S), and Carry out (Cy_o). Here it is the truth table of the full adder:

In			Out	
Cy_i	**A**	**B**	**Cy_o**	**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

Just like the parity generator we cover in a previous slide…

$S=Cy_i \oplus A \oplus B$

For the C_{yo}, we need to work a little…

Equation for Cy_o

Cy_i	A	B	Cy_o
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

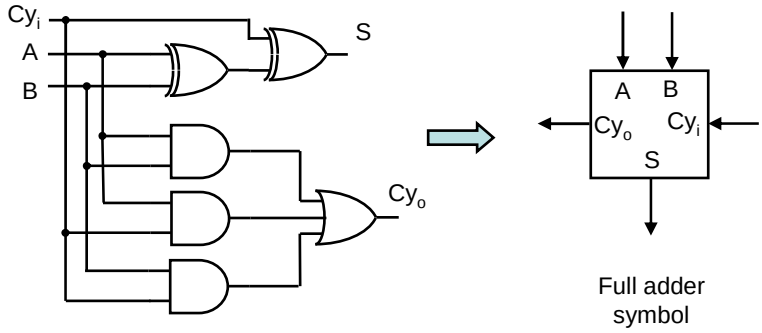
	A'B'	A'B	AB	AB'
Cy_i'	0	0	1	0
Cy_i	0	1	1	1

$$Cy_o=A.B+Cy_i.B+Cy_i.A$$

Full Adder Circuit

$$S=Cy_i \oplus A \oplus B$$

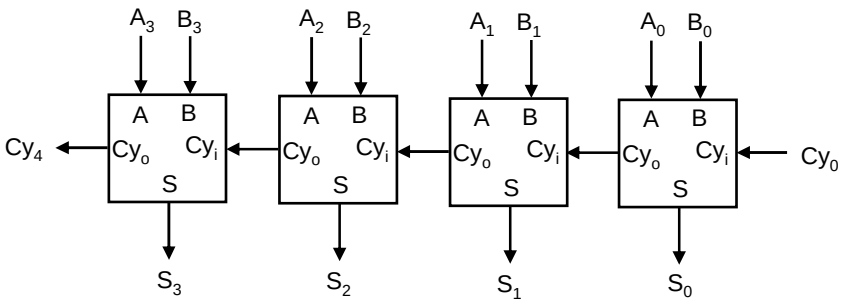
$$Cy_o=A.B+Cy_i.B+Cy_i.A$$



Full adder symbol

One possible circuit for a full adder.

4-bit Adder



Can be expanded to any number of bits.

Carry Look Ahead

- If you look at the figure from the previous slide, you'll notice that the "carry in" from one full adder is the carry out from the previous full adder.
- For the full adder I implemented, this is bad because it slows down the adder: an adder can not finish before the previous one has finished.
- To minimize this problem "Carry Look Ahead" logic is often used. I will not cover it in this lecture. If you are curious about Carry Look Ahead, it is all over the internet.

Two's Complement

- It is very easy to implement subtraction if we know how to get the two's complement of a binary number:
  - Negate each bit of the number. This is also called the 1's complement.
  - Add one to the result above.
- If you add a binary number and its 2's complement the result is zero.

2's Complement

To obtain the 2's complement of a number:

0	0	1	1	0	1	0	0	Original number: 52
1	1	0	0	1	0	1	1	1's complement
							1	Add 1
1	1	0	0	1	1	0	0	2's complement

Binary Subtraction

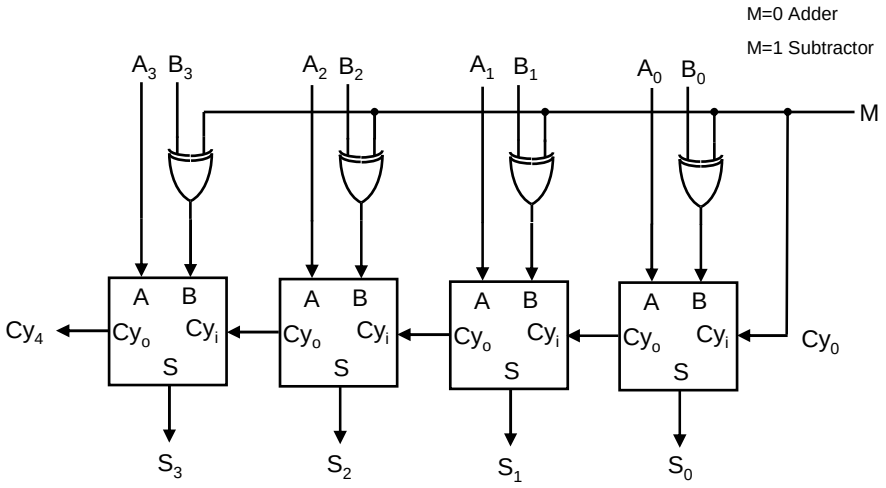
- To subtract binary number B from binary number A, just get the 2's complement of B and add it to A!

1	1	0	0	1	0	0	1	Subtract (201-52)
0	0	1	1	0	1	0	0	

1	1	0	0	1	0	0	1	Add
1	1	0	0	1	1	0	0	(2's complement of 52)
1	0	0	1	0	1	0	1	Result: 149

A very simple modification to the adder allows us to do subtraction!

4-bit Adder/Subtractor



4-bit Adder in VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity binary_adder is
    port(
        a, b     : in   std_logic_vector(3 downto 0);
        carry_in : in   std_logic;
        sum      : out  std_logic_vector(3 downto 0);
        carry    : out  std_logic
    );
end binary_adder;

architecture a of binary_adder is
    signal sum_temp : std_logic_vector(4 downto 0);
begin

    process(a, b, sum_temp, carry_in)
    begin
        sum_temp <= ('0' & a) + ('0' & b) + ("0000" & carry_in);
        carry <= sum_temp(4);
        sum <= sum_temp(3 downto 0);
    end process;
end a;
```


4-bit Adder in VHDL

- VHDL has a library with several arithmetic and logic functions: **ieee.std_logic_unsigned**
- To concatenate bits we use ‘&’.
- Almost sure carry look ahead is implemented in the library.
- Other arithmetic functions available: ‘-’, ‘*’, ‘=‘, ‘<=‘, ‘>=‘, ‘/=‘. Division is not available in this library.
- Easy to expand to any number of bits!

VHDL Operators

http://www.quicknet.se/hdc/hdl/educaton/operator/descr.htm

- Adding operators
- Assignment operators
- Logical operators
- Multiplying operators
- Relational operators
- Shift operators
- Sign
- Miscellaneous operators

Adding operators

- + Addition
- - Subtraction
- & Concatenation

Assignment operators

- <= Assignment of signals
- := Assignment of variables, also assignment of signals at declaration

Logical operators

- and
- nand
- or
- nor
- xor
- xnor
- not

Multiplying operators

- * Multiplication
- / Division
- mod Modulus
- rem Reminder

Relational operators

- = Equal
- /= Not equal
- < Less than
- <= Less than or equal
- > Greater than
- >= Greater than or equal

Shift operators

- rol Rotate left logical
- ror Rotate right logical
- sla Shift left arithmetic
- sra Shift right arithmetic
- sll Shift left logical
- srl Shift right logical

Sign

- \+ Plus
- \- Minus

Miscellaneous operators

- ** Exponentiation
- abs Absolute value

BCD Adder

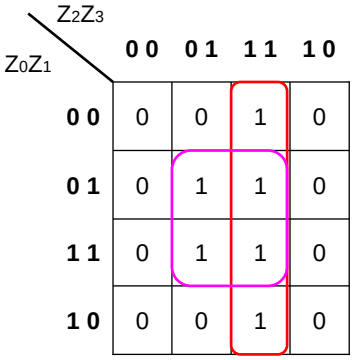
- We can use two 4-bit binary adders to implement a one digit BCD adder.
- BCD is often used when interaction with people is needed. Calculators are a good example!
- Let us start with the derivation of a BCD adder from a 4-bit binary adders

Binary Sum					BCD Sum					
K	Z_3	Z_2	Z_1	Z_0	Cy	S_3	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

In this portion of the table, BCD=Binary+6

BCD Adder

- We need to detect if the binary adder produces a number larger than 9 OR a carry out. If so, we add 6 to it. Lets focus on detecting a binary number larger than 9:

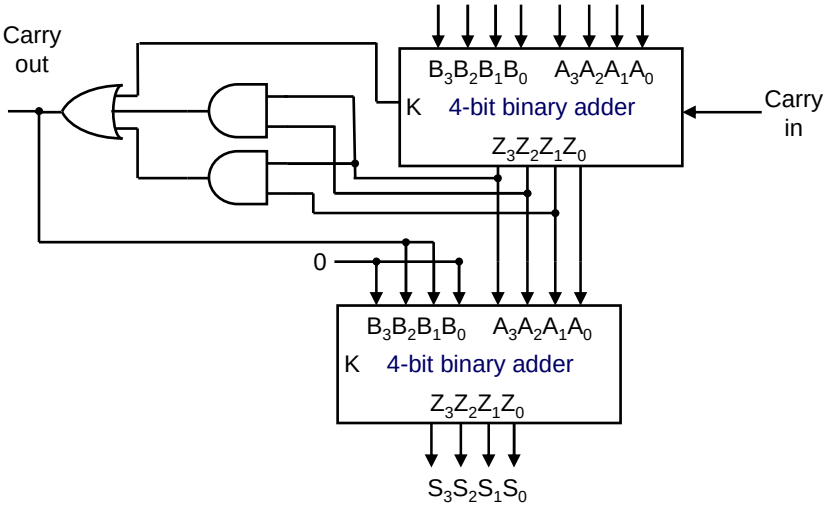


$f_{DCBA>1001} = Z_1.Z_3 + Z_2.Z_3$

So, we need to add 0110 if:

$f_{add6} = Z_1.Z_3 + Z_2.Z_3 + K$

Block Diagram of BCD Adder



BCD Adder in VHDL

- To implement the BCD adder we have two options:
  - Stick to old school methods and implement the BCD adder using an schematic diagram.
  - Embrace modern times and use a HDL, such as VHDL.
- For complicated circuits such as the BCD adder it is more convenient to useVHDL! You'll se what I mean in the next two slides…

BCD Adder in VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity bcd_adder is
    port(
        a,b      : in   std_logic_vector (3 downto 0);
        carry_in : in   std_logic;
        sum      : out  std_logic_vector (3 downto 0);
        carry    : out  std_logic
    );
end bcd_adder;
```

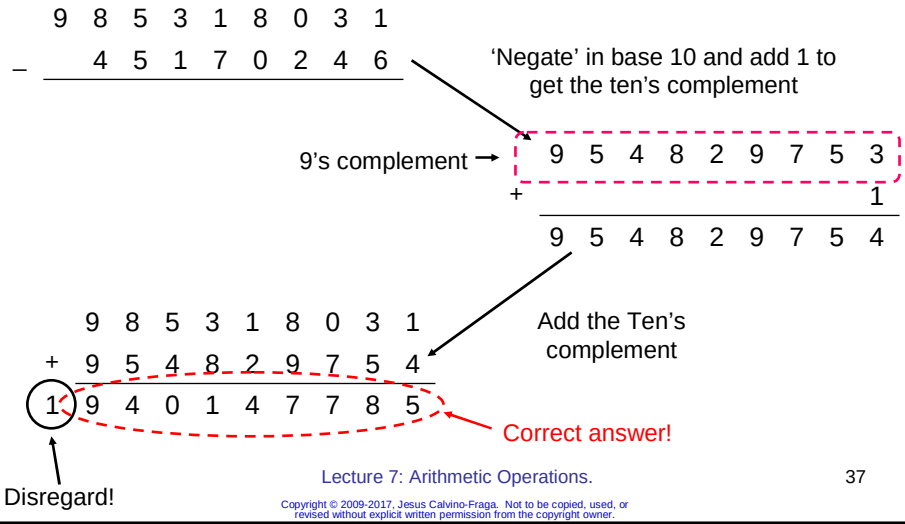
BCD Adder in VHDL

```
architecture a of bcd_adder is
signal sum_temp      : std_logic_vector(4 downto 0);
signal sum_temp_adj : std_logic_vector(4 downto 0);

begin

process (a, b, sum_temp, carry_in, sum_temp_adjs)
begin
    sum_temp <= ('0' & a) + ('0' & b) + ( "0000" & carry_in);
    sum_temp_adj <= sum_temp + "00110";
    if (sum_temp > 9) then
        carry <= '1';
        sum <= sum_temp_adj(3 downto 0);
    else
        carry <= '0';
        sum <= sum_temp(3 downto 0);
    end if;
end process;

end a;
```


BCD Subtraction

- We can do something similar to what we did for the binary subtraction:
  - Get the 10's complement of the subtrahend
  - Add it to the minuend.
- Let show with an example how this works…

BDC Subtraction. The Method of Complements.



Getting the 9's Complement…

BCD in				BCD out			
D	C	B	A	f_D	f_C	f_B	f_A
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0

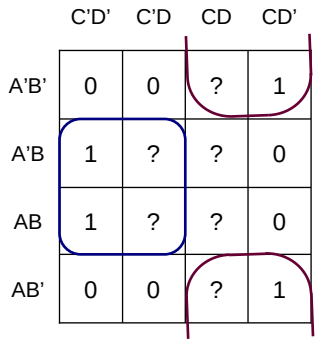
By inspection:

$f_A = A'$

$f_B = B$

Getting the 9's Complement…

D	C	B	A	f_C
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0

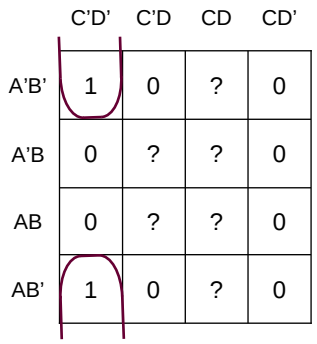


$f_C=B.C'+B'.C$

$f_C=B\oplus C$

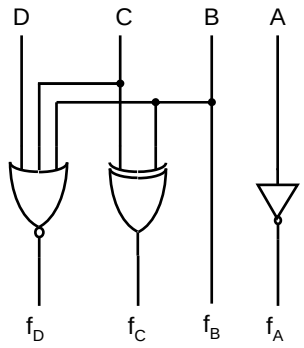
Getting the 9's Complement…

D	C	B	A	f_D
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0

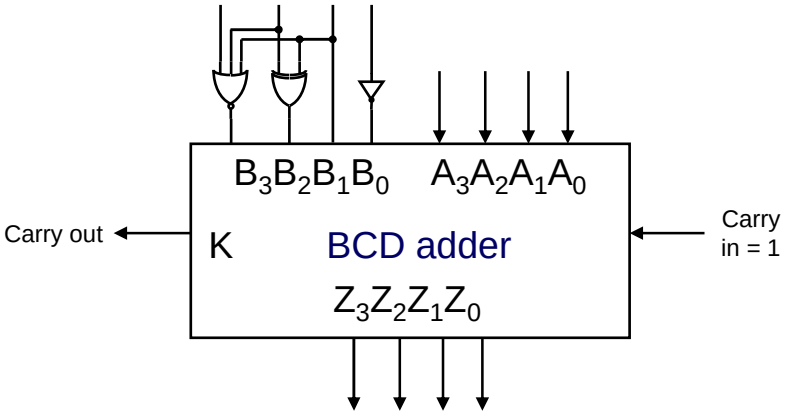


$f_D=B'.C'.D'=(B+C+D)'$

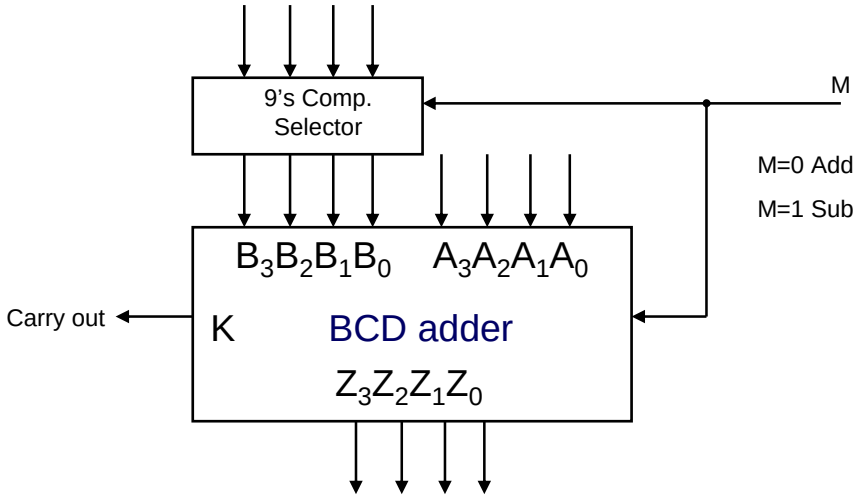
Getting the 9's Complement…



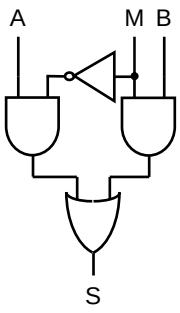
BCD Subtractor



BCD Adder/Subtractor

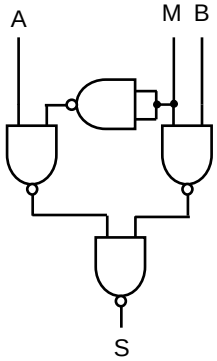


2 to 1 Data Selector

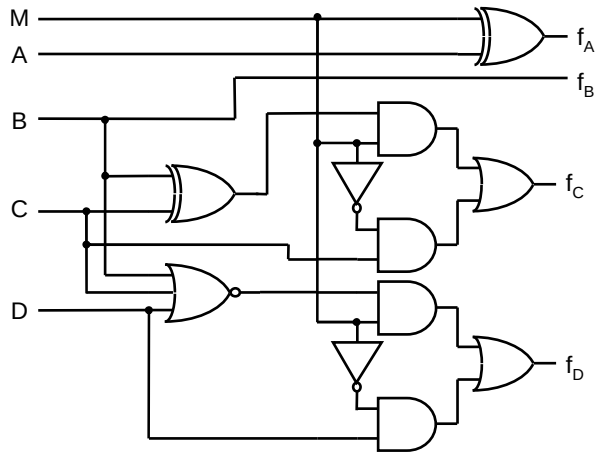


M	S
0	A
1	B

All NAND version?



9's Complement Selector



BCD Adder/Subtractor in VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity bcd_addsub is
    port (
        a,b      : in   std_logic_vector(3 downto 0);
        sub      : in   std_logic; -- 1 to subtract
        carry_in : in   std_logic;
        sum      : out  std_logic_vector(3 downto 0);
        carry    : out  std_logic
    );
end bcd_addsub;

architecture a of bcd_addsub is
    signal sum_temp : std_logic_vector(4 downto 0);
    signal sum_temp_adjs : std_logic_vector(4 downto 0);
    signal b_temp : std_logic_vector(4 downto 0);

begin
```

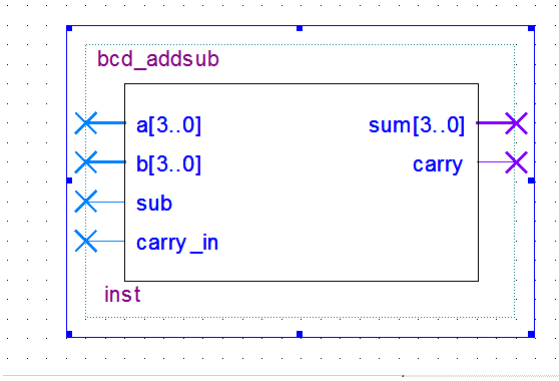
BCD Adder/Subtractor in VHDL

```
    process (b, sub) is
    begin
        if sub = '1' then
            case b is
                when "0000" => b_temp<="01001";
                when "0001" => b_temp<="01000";
                when "0010" => b_temp<="00111";
                when "0011" => b_temp<="00110";
                when "0100" => b_temp<="00101";
                when "0101" => b_temp<="00100";
                when "0110" => b_temp<="00011";
                when "0111" => b_temp<="00010";
                when "1000" => b_temp<="00001";
                when "1001" => b_temp<="00000";
                when others => b_temp<="00000";
            end case;
        else
            b_temp <= ('0' & b);
        end if;
    end process;
```


BCD Adder/Subtractor in VHDL

```
    process (a, b_temp, sum_temp, carry_in, sum_temp_adjs)
    begin
        sum_temp <= ('0' & a) + b_temp + ("0000" & carry_in);
        sum_temp_adjs <= sum_temp + "00110";
        if (sum_temp > 9) then
            carry <= '1';
            sum <= sum_temp_adjs(3 downto 0);
        else
            carry <= '0';
            sum <= sum_temp(3 downto 0);
        end if;
    end process;

end a;
```

BCD Adder/Subtractor Graphic Symbol



When cascading this block for a multi-digit BCD adder/subtractor 'sub' and 'carry_in' must be connected together for the first BCD digit.

Exercises

- Design a three input XOR gate using NAND gates only.
- Using a 4-bit binary subtractor, show how to obtain the functions a>=b and a<b.
- Use the graphic symbol for the BCD adder/subtractor from the slides above to construct a 2-digit BCD adder/subtractor.