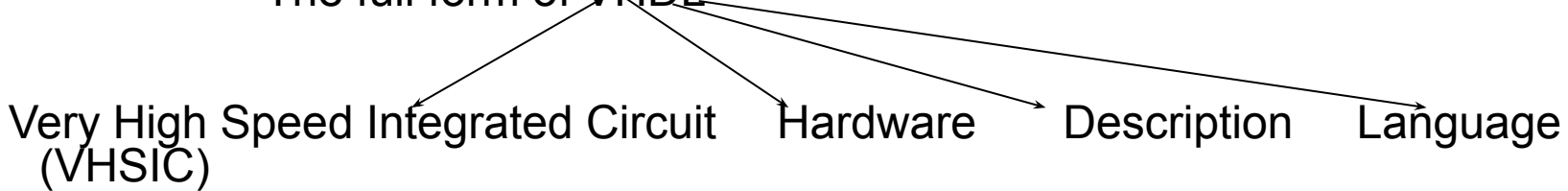


Computer Architecture (VHDL coding) Lab

Basic structure of VHDL programming

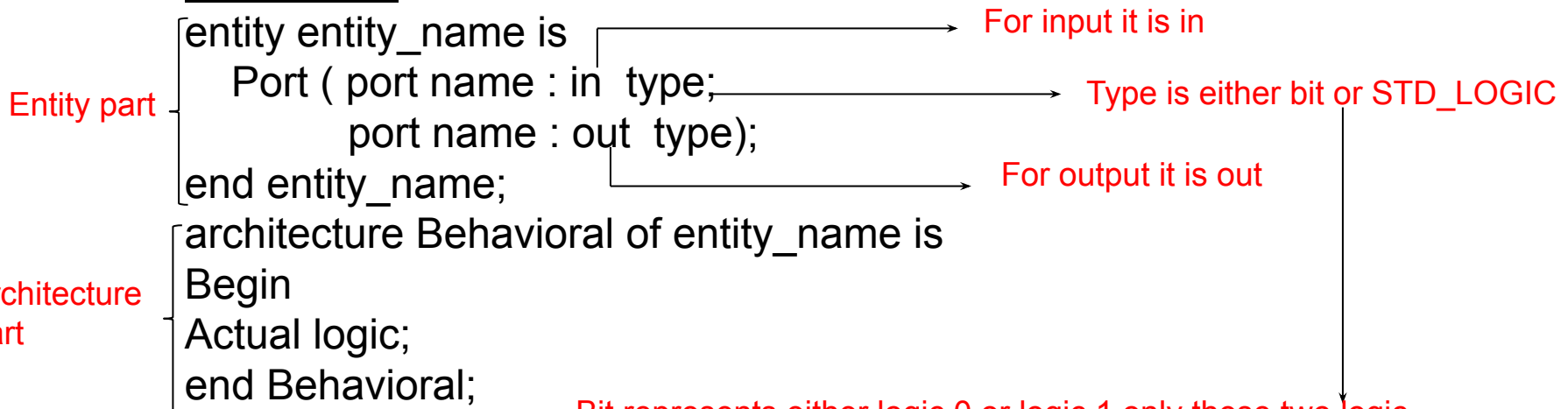
The full form of VHDL



There are two portion in VHDL coding.

1. **Entity part:** In entity part all inputs, outputs of a hardware circuit and their types are declared.
2. **Architecture part:** In architecture part the actual logic is defined.

VHDL code

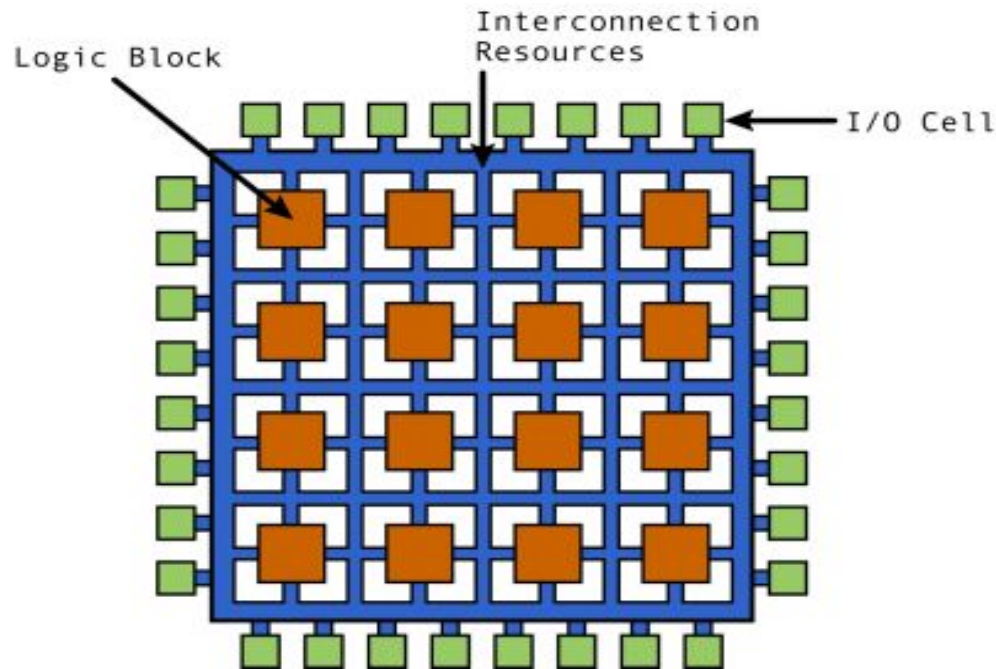


Bit represents either logic 0 or logic 1 only these two logic.
STD_LOGIC represent standard logic (logic 0, logic1, Uninitialized(U), Don't care (X), High Impedance (Z) like different types of logic)

FPGA (Field-Programmable Gate Array)

A **field-programmable gate array (FPGA)** is an integrated circuit designed to be configured by a customer or a designer after manufacturing.

Uses for FPGAs cover a wide range of areas—from equipment for video and imaging, to circuitry for computer, auto, aerospace, and military applications.



FPGA architecture

FPGA architecture contd.

The three basic types of programmable elements for an FPGA

Configurable logic block (CLB):

An individual CLB consists of a number of discrete logic components itself, such as look-up tables (LUTs) and flip-flops.

I/O block: Input/output blocks are the components through which data transfers in to and out of the FPGA.

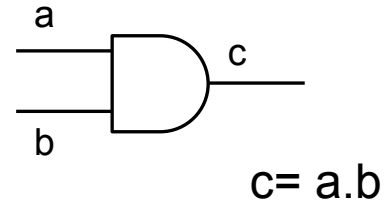
Programmable interconnect:

Programmable Interconnect Points (PIP) provide the routing paths used to connect the CLBs and inputs/outputs.

It is a CMOS transistor switch which can be turned on or off by using programming.

Logically these are nothing but MUX and DMUX.

AND Gate



VHDL code for AND Gate (Dataflow design):

entity andgate is

Port (a, b : in bit;
c : out bit);

end entity;

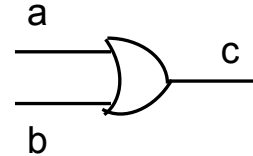
architecture Behavioral of andgate is

begin

c<=a AND b;

end Behavioral;

OR Gate



$$c = a + b$$

VHDL code for OR Gate (Dataflow design):

entity orgate is

```
    Port ( a : in  STD_LOGIC;  
          b : in  STD_LOGIC;  
          c : out STD_LOGIC);
```

end orgate;

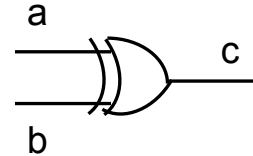
architecture Behavioral of orgate is

begin

```
c<=a OR b;
```

end Behavioral;

XOR Gate



$$c = a \oplus b$$

VHDL code for XOR Gate (Dataflow design):

entity xorgate is

```
    Port ( a : in  STD_LOGIC;  
          b : in  STD_LOGIC;  
          c : out STD_LOGIC);
```

end xorgate;

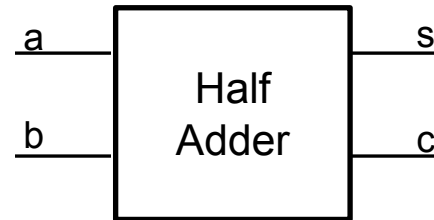
architecture Behavioral of xorgate is

begin

```
c<=a XOR b;
```

end Behavioral;

Half Adder



$$s = a \oplus b$$

$$c = a.b$$

VHDL code for Half adder (Dataflow design):

entity HA is

Port (a : in STD_LOGIC;

b : in STD_LOGIC;

s : out STD_LOGIC;

c : out STD_LOGIC);

end HA;

architecture Behavioral of HA is

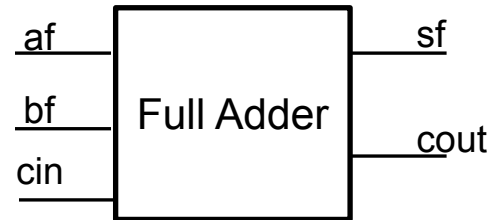
begin

s<=a XOR b;

c<=a AND b;

end Behavioral;

Full Adder



$$sf = af \oplus bf \oplus cin$$

$$cout = af.bf + bf.cin + cin.af$$

VHDL code for Full Adder (Dataflow design):

entity FA is

```
Port ( af : in  STD_LOGIC;
      bf : in  STD_LOGIC;
      cin : in  STD_LOGIC;
      sf : out STD_LOGIC;
      cout : out STD_LOGIC);
```

end FA;

architecture Behavioral of FA is

Begin

```
sf<=af XOR bf XOR cin ;
```

```
cout<=(af AND bf) OR (bf AND cin) OR (cin AND af) ;
```

```
end Behavioral;
```

There are three types of design in VHDL code

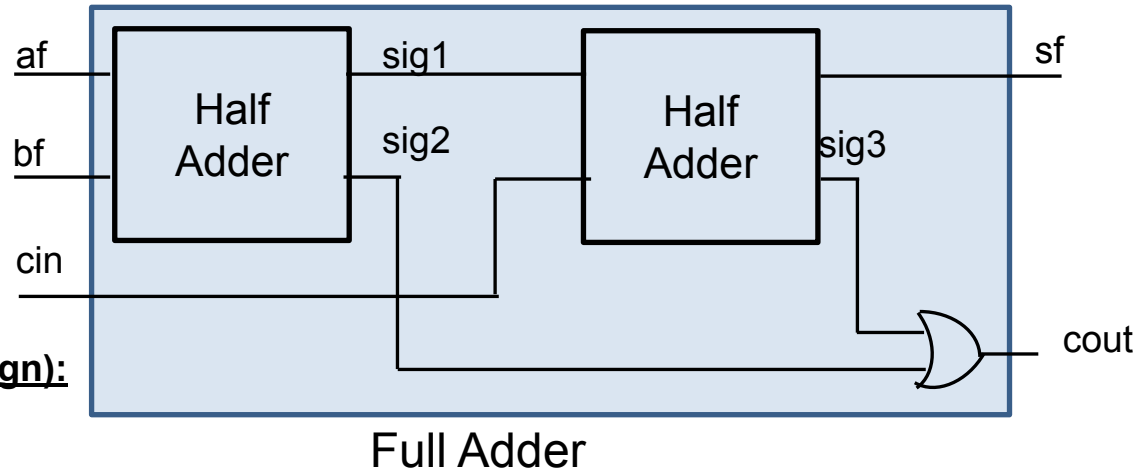
- 1) Dataflow design
- 2) Structural design
- 3) Behavioral design

1) Dataflow design: This design is completely logic expression based.

2) Structural design: Here basic building blocks are used to design a new hardware circuit. Like function

3) Behavioral design: This design is completely truth table based.

Full Adder using two Half Adder



VHDL code for Full Adder (Structural design):

entity FA is

```
Port ( af : in  STD_LOGIC;  
      bf : in  STD_LOGIC;  
      cin : in  STD_LOGIC;  
      sf : out STD_LOGIC;  
      cout : out STD_LOGIC);
```

end FA;

architecture Behavioral of FA is

component HA is

```
Port ( a : in  STD_LOGIC;  
      b : in  STD_LOGIC;  
      s : out STD_LOGIC;  
      c : out STD_LOGIC);
```

Component
It is like a function
prototype

end component;

```
signal sig1,sig2,sig3: STD_LOGIC;
```

Here sig1, sig2 and sig3 are intermediate input output which are called signal.

begin

```
HA1:HA port map(af,bf,sig1,sig2);
```

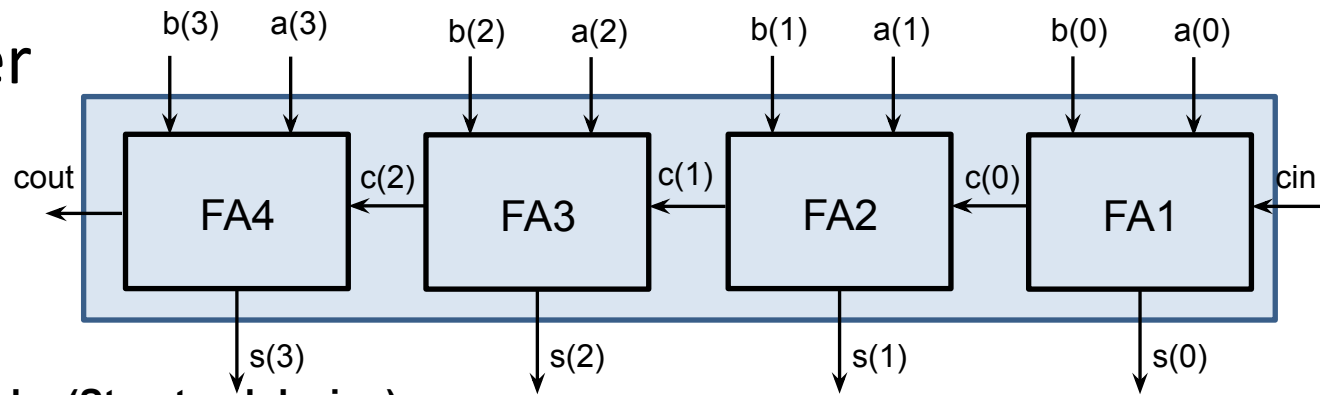
```
HA2:HA port map(sig1,cin,sf,sig3);
```

Components are calling here two times.

```
cout<=sig2 OR sig3;
```

end Behavioral;

Ripple Carry Adder



VHDL code for Ripple Carry Adder (Structural design):

```
entity ripplecarryadder is
  Port ( a :in STD_LOGIC_VECTOR (3 downto 0);
        b : in STD_LOGIC_VECTOR (3 downto 0);
        cin : in  STD_LOGIC;
        s :out STD_LOGIC_VECTOR (3 downto 0);
        cout : out  STD_LOGIC);
```

```
end ripplecarryadder;
```

```
architecture Behavioral of ripplecarryadder is
```

```
component FA is
```

```
  Port ( af : in  STD_LOGIC;
        bf : in  STD_LOGIC;
        cin : in  STD_LOGIC;
        sf : out STD_LOGIC;
        cout : out STD_LOGIC);
```

```
end component;
```

```
signal c : STD_LOGIC_VECTOR (2 downto 0);
```

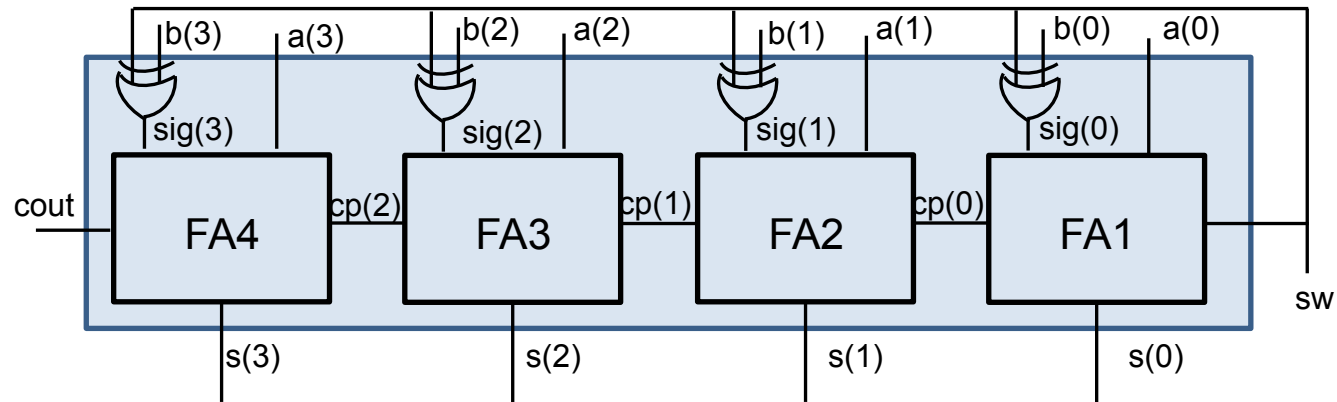
```
begin
```

```
  fa1:FA port map(a(0),b(0),cin,s(0),c(0));
  fa2:FA port map(a(1),b(1),c(0),s(1), c(1));
  fa3:FA port map(a(2),b(2),c(1),s(2), c(2));
  fa4:FA port map(a(3),b(3),c(2),s(3),cout);
```

```
end Behavioral;
```

Here VECTOR is used like an array.
3 downto 0 means a is 4 bit binary number
(a(3),a(2),a(1),a(0))

Adder-Subtractor composite unit



VHDL code for Adder-Subtractor (Structural design):

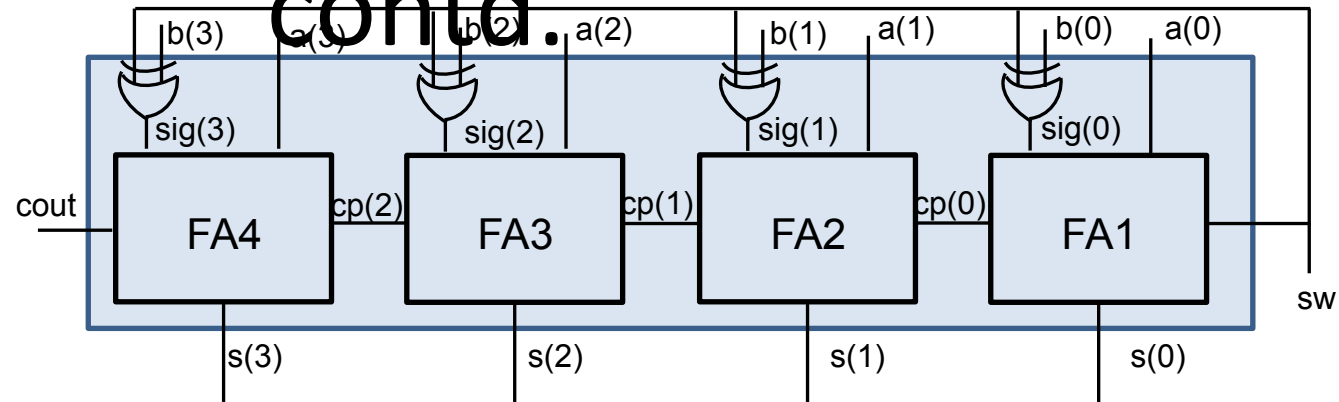
```
entity addsub is
  Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
        b : in  STD_LOGIC_VECTOR (3 downto 0);
        sw : in  STD_LOGIC;
        s : out STD_LOGIC_VECTOR (3 downto 0);
        cout : out STD_LOGIC);
end addsub;

architecture Behavioral of addsub is
  component FA is
    Port ( af : in  STD_LOGIC;
          bf : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          sf : out STD_LOGIC;
          cout : out STD_LOGIC);
  end component;

  -- Structural design would go here, showing the instantiation of FA components
  -- and their connections to form the addsub entity.
end architecture;
```

Adder-Subtractor composite unit

contd.



```
signal sig : STD_LOGIC_VECTOR (3 downto 0);
signal cp : STD_LOGIC_VECTOR (2 downto 0);
begin
  sig(0) <= sw XOR b(0);
  sig(1) <= sw XOR b(1);
  sig(2) <= sw XOR b(2);
  sig(3) <= sw XOR b(3);
  fa1: FA port map(a(0), sig(0), sw, s(0), cp(0));
  fa2: FA port map(a(1), sig(1), cp(0), s(1), cp(1));
  fa3: FA port map(a(2), sig(2), cp(1), s(2), cp(2));
  fa4: FA port map(a(3), sig(3), cp(2), s(3), cout);
end Behavioral;
```

Multiplexer

VHDL code for 4:1 Multiplexer (Behavioral design):

entity multiplexer is

Port (I : in STD_LOGIC_VECTOR (3 downto 0);

S : in STD_LOGIC_VECTOR (1 downto 0);

O : out STD_LOGIC);

end multiplexer;

architecture Behavioral of multiplexer is

begin

Process(I,S)

begin

case S is

when "00" => O<=I(0);

when "01" => O<=I(1);

when "10" => O<=I(2);

when "11" => O<=I(3);

when Others => O<='U';

end case;

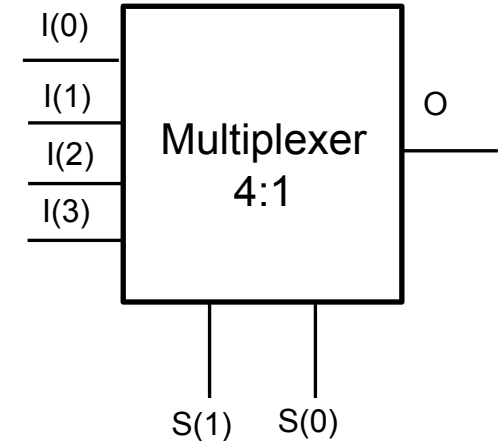
end Process;

end Behavioral;

These parameters
are called sensitivity list

Process block
within process block execution
is sequential

switch case



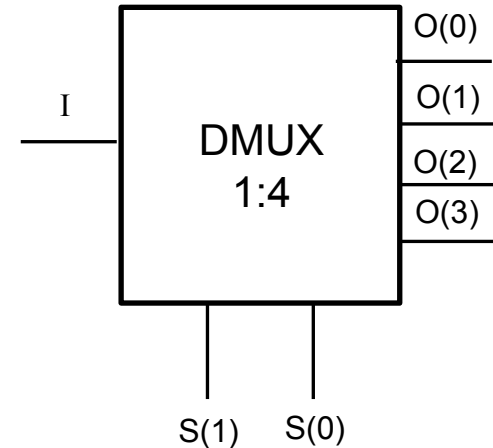
Truth Table

S(1)	S(0)	O
0	0	I(0)
0	1	I(1)
1	0	I(2)
1	1	I(3)

Demultiplexer

VHDL code for 1:4 Demultiplexer (Behavioral design):

```
entity demux is
  Port ( I : in  STD_LOGIC;
        S : in  STD_LOGIC_VECTOR (1 downto 0);
        O : out STD_LOGIC_VECTOR (3 downto 0));
end demux;
architecture Behavioral of demux is
begin
  process(I,S)
  begin
    O<="UUUU";——— Here U means Uninitialized
    case S is
      when "00"=> O(0)<=I;
      when "01"=> O(1)<=I;
      when "10"=> O(2)<=I;
      when others => O(3)<=I;
    end case;
  end process;
end Behavioral;
```



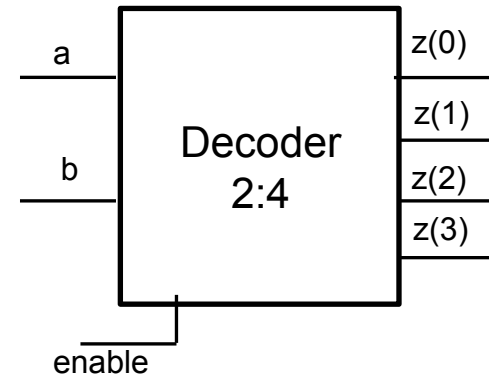
Truth Table

S(1)	S(0)	O(3)	O(2)	O(1)	O(0)
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Decoder

VHDL code for 2:4 Decoder:

```
entity decoder is
  Port ( a : in  STD_LOGIC;
        b : in  STD_LOGIC;
        enable : in  STD_LOGIC;
        z : out STD_LOGIC_VECTOR (3 downto 0));
end decoder;
architecture Behavioral of decoder is
begin
  process(a,b,enable)
    variable abar,bbar:STD_LOGIC;—— Here abar, bbar are two variables
  begin
    abar:=NOT a;
    bbar:= NOT b;
    if enable='1'then
      z(0)<=abar AND bbar;
      z(1)<=abar AND b;
      z(2)<=a AND bbar;
      z(3)<=a AND b;
    else
      z<="UUUU";
    end if;
  end process;
end Behavioral;
```



Truth Table

a	b	z(3)	z(2)	z(1)	z(0)
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$$z(0) = a' \cdot b'$$

$$z(1) = a' \cdot b$$

$$z(2) = a \cdot b'$$

$$z(3) = a \cdot b$$

Difference between signal and variable

1. Signal is used to represent intermediate input output which has a real existence. Variable is used to store the value temporary and not really exist in the hardware circuit.
2. Assigning value to signal and variable is different.

```
signal s :STD_LOGIC;  
s<='1';  
variable v : STD_LOGIC;  
v:='1';
```

VHDL code for Full Adder (Behavioral design):

```
entity fulbehaviour is
  Port ( a : in  STD_LOGIC;
        b : in  STD_LOGIC;
        cin : in  STD_LOGIC;
        s : out STD_LOGIC;
        cout : out STD_LOGIC);
end fulbehaviour;
architecture Behavioral of fulbehaviour is
begin
  process(a,b,cin)
    variable sw: std_logic_vector(2 downto 0);
  begin
    sw(0):=cin;
    sw(1):=b;
    sw(2):=a;
    case sw is
      when "000" => s<='0';cout<='0';
      when "001" => s<='1'; cout<='0';
      when "010" => s<='1'; cout<='0';
      when "011" => s<='0'; cout<='1';
      when "100" => s<='1'; cout<='0';
      when "101" => s<='0'; cout<='1';
      when "110" => s<='0'; cout<='1';
      when others => s<='1'; cout<='1';
    end case;
  end process;
end Behavioral;
```

Truth Table of Full Adder

a	b	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Multiplication Using ADD and shift method

M = 8

M = 1000

Q = 0110

Q = 6

A	Q	Size	Comment
0000	0110	4	initialize
0000 1000	0011	3	Right Shift A and Q
1000	0011	3	A+M
0100 1000	0001	2	Right Shift A and Q
1100	0001	2	A+M
0110	0000	1	Right Shift A and Q
0011	0000	0	Right Shift A and Q

Product = 0011 0000

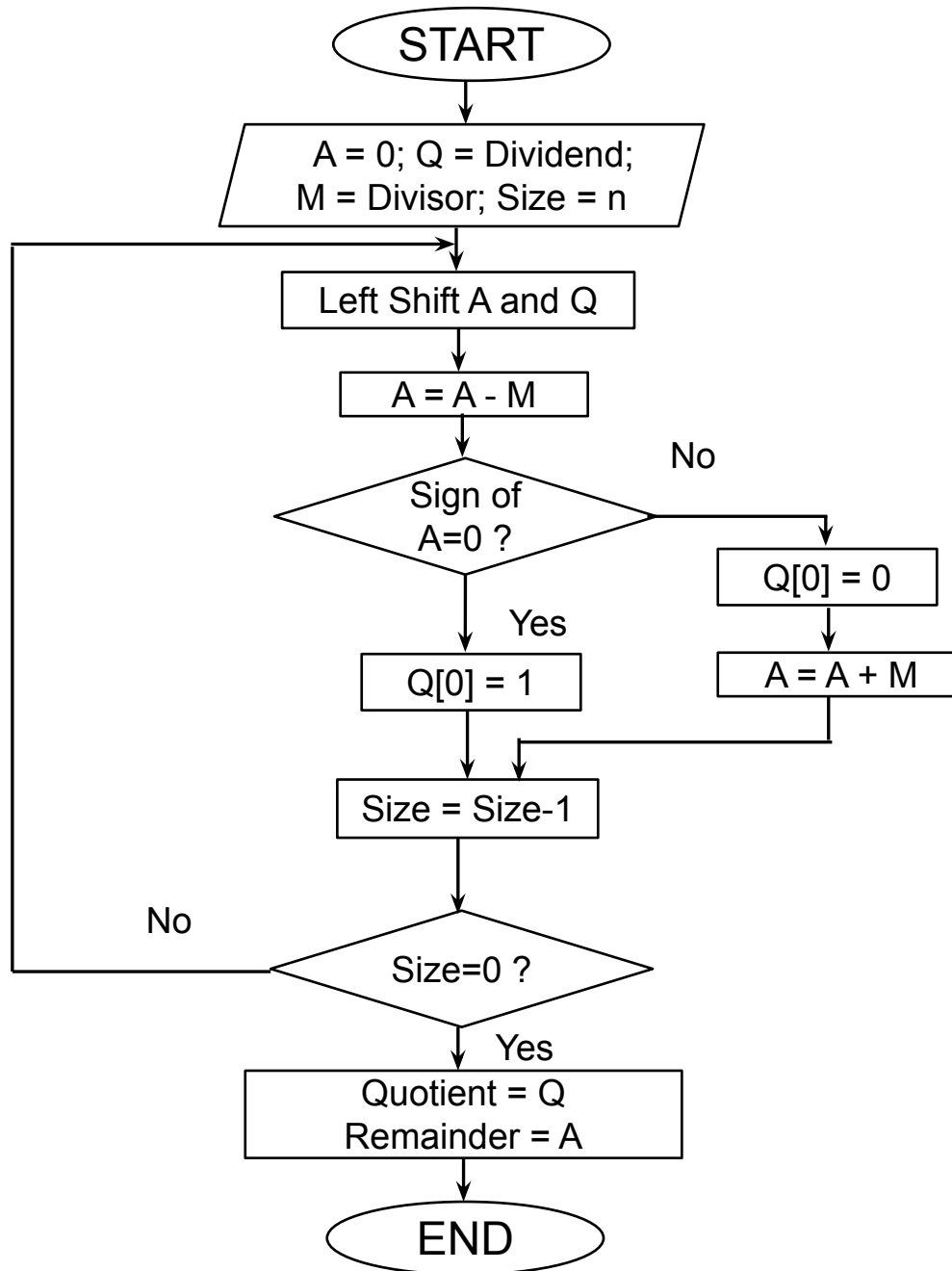
VHDL code for 4 bit binary multiplier

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity binary_multiplier is
    Port ( m : in  STD_LOGIC_VECTOR (3 downto 0);
          q : in  STD_LOGIC_VECTOR (3 downto 0);
          r : out STD_LOGIC_VECTOR (7 downto 0));
end entity;

architecture multiplier of binary_multiplier is
begin
    process(m,q)
        variable acc: std_logic_vector(8 downto 0);
        variable multiplicand:std_logic_vector(4 downto 0);
    begin
        acc(8 downto 4):="00000";
        acc(3 downto 0):=q;
        multiplicand:='0' & m;
        for i in 1 to 4 loop
            if acc(0)='1' then
                acc(8 downto 4):=acc(8 downto 4)+ multiplicand;
            end if;
            acc:='0' & acc(8 downto 1);
        end loop;
        r<= acc(7 downto 0);
    end process;
end multiplier;
```

Restoring Division Flow chart



Restoring Division Example

M = 3, Q = 7 M = 0011
Q = -M = 1101

A	Q	Size	Comment
0000	0111	4	initialize
0000	111□	4	Left Shift A and Q
1101			
1101	111□	4	A=A-M
0011			Set Q[0]=0 and
0000	1110	3	A=A+M
0001	110□	3	Left Shift A and Q
1101			
1110	110□	3	A=A-M
0011			Set Q[0]=0 and
0001	1100	2	A=A+M
0011	100□	2	Left Shift A and Q
1101			
0000	100□	2	A=A-M
0000	1001	1	Set Q[0]=1
0001	001□	1	Left Shift A and Q
1101			
1110	001□	1	A=A-M
0011			Set Q[0]=0 and
0001	0010	0	A=A+M
Remainder	Quotient		

VHDL code for 4 bit restoring division

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_arith.ALL;  
use IEEE.STD_LOGIC_unsigned.ALL;
```

entity division is

```
Port ( d : in  STD_LOGIC_VECTOR (3 downto 0);  
      m : in  STD_LOGIC_VECTOR (3 downto 0);  
      q : out STD_LOGIC_VECTOR (3 downto 0);  
      r : out STD_LOGIC_VECTOR (3 downto 0));
```

end division;

architecture Behavioral of division is

begin

process(d,m)

variable ac:std_logic_vector(7 downto 0);

variable Mbar:std_logic_vector(3 downto 0);

begin

Mbar:=not m;

ac:="0000" & d;

for i in 1 to 4 loop

ac(7 downto 0):=ac(6 downto 0) & 'U';

ac(7 downto 4):=ac(7 downto 4)+Mbar+"0001";

if ac(7)='1' then

ac(0):='0';

ac(7 downto 4):=ac(7 downto 4)+m;

else

ac(0):='1';

end if;

end loop;

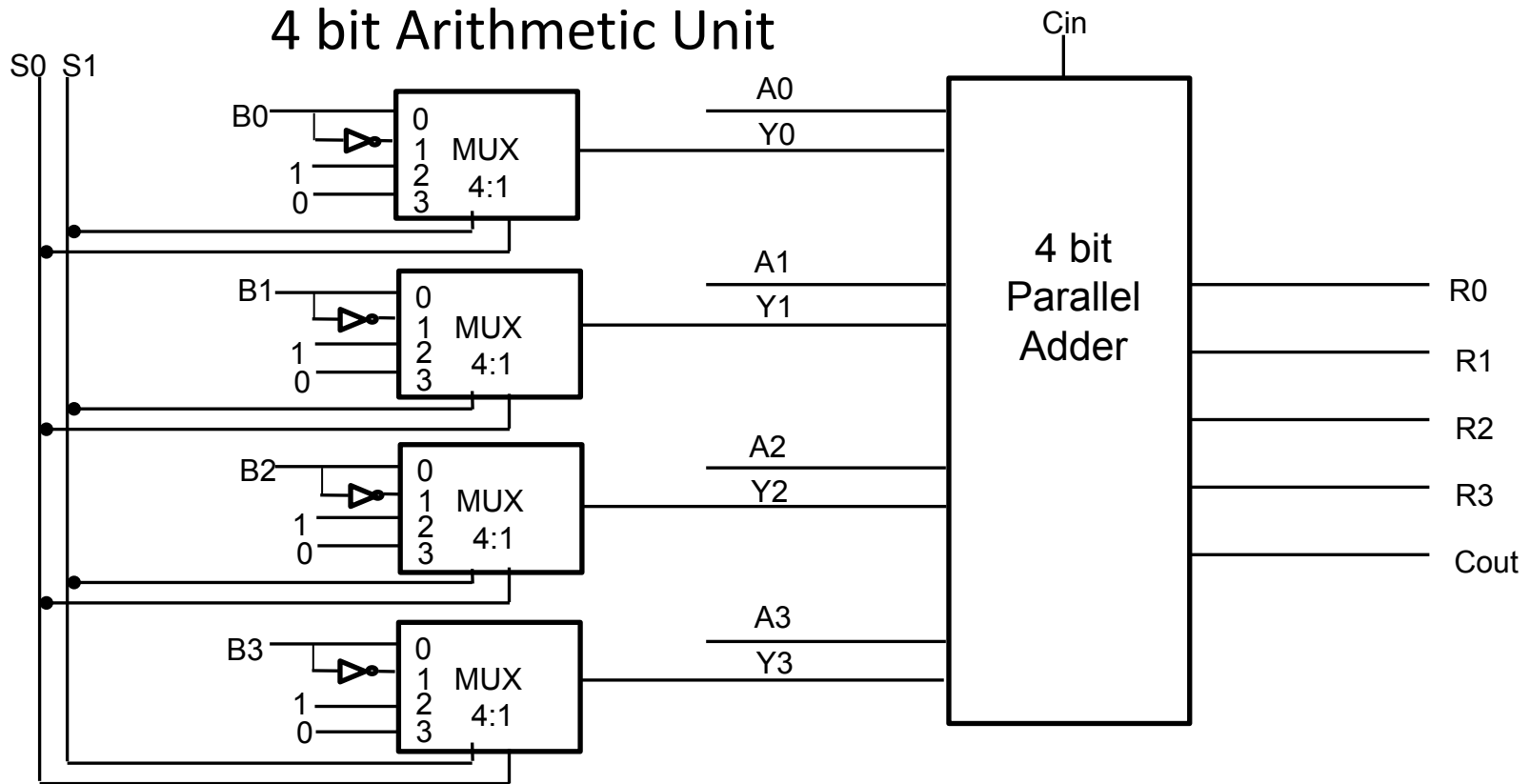
q<=ac(3 downto 0);

r<=ac(7 downto 4);

end process;

end Behavioral;

4 bit Arithmetic Unit



S1	S0	Cin	Y	$F=A+Y+Cin$	Operation
0	0	0	B	$F=A+B$	Addition
0	0	1	B	$F=A+B+1$	Addition with carry
0	1	0	\bar{B}	$F=A+\bar{B}$	Subtraction with borrow
0	1	1	\bar{B}	$F=A+\bar{B}+1$	Subtraction
1	0	0	-1	$F=A-1$	Decrement
1	0	1	-1	$F=A$	Transfer
1	1	0	0	$F=A$	Transfer
1	1	1	0	$F=A+1$	Increment

VHDL code for 4 bit Arithmetic Unit

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity alu4bit is
  port(a,b : in std_logic_vector(3 downto 0);
        s: in std_logic_vector(1 downto 0);
        cin: in std_logic;
        r: out std_logic_vector(3 downto 0);
        cout : out std_logic );
end alu4bit;
architecture arch1 of alu4bit is
begin
  process(a,b,s,cin)
    variable sw: std_logic_vector(2 downto 0);
    variable a1,b1,r1: std_logic_vector(4 downto 0);
    begin
```

```
      sw(0):=cin;
      sw(1):=s(0);
      sw(2):=s(1);
      a1:='0' & a;
      b1:='0' & b;case sw is
        when "000" => r1:=a1+b1;
        when "001" => r1:=a1+b1+'1';
        when "010" => r1:=a1+not(b1);
        when "011" => r1:=a1+not(b1)+'1';
        when "100" => r1:=a1-'1';
        when "101" => r1:=a1;
        when "110" => r1:=a1;
        when "111" => r1:=a1+'1';
        when others=> r1:="UUUUUU";
      end case;
      r<=r1(3 downto 0); cout<=r1(4);
    end process;
  end arch1;
```

VHDL code for designing 128 x 8 bit RAM chip

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity memory1 is
port(clk: in std_logic ;
cs : in std_logic ;
r : in std_logic ;
w : in std_logic ;
clr: in std_logic;
abus:in std_logic_vector(6 downto 0);
dbus:inout std_logic_vector(7 downto 0));
end memory1;
```

```
architecture arch1 of memory1 is
type ma is array(0 to 127) of
std_logic_vector(7 downto 0);
signal mem : ma;
begin
process(clk)
begin
if(rising_edge(clk) and cs='1') then
    if w='1' then
        mem(conv_integer(unsigned(abus)))<=dbus;
    end if;
    if r='1' then
        dbus<=mem(conv_integer(unsigned(abus)));
    end if;
    if clr='1' then
        for i in 0 to 127 loop
            mem(i)<="UUUUUUUUU";
        end loop;
    end if;
end if;
end process;
end arch1;
```

Thank You