SRI SUKHMANI INSTITUTE OF ENGINEERING AND TECHNOLOGY,
DERA BASSI (MOHALI)

VLSI LAB MANUAL

ECE DEPARTMENT
Introduction to VHDL

It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. The system may be a single gate to a complete digital electronic system.

VHDL is a hardware description language used in electronic design automation to describe digital and mixed-signal system such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming language.

It can be considered a combination of following languages as:

a) Sequential language
b) Concurrent language
c) Net list language
d) Timing language
e) Waveform Generation language

Need of VHDL

The requirement for it was generated in 1981 under VHSIC program. In this program a number of US companies were involved in designing VHSIC chips for DoD (defense department).

Most of the companies were using different hardware description to describe and develop their IC, as a result different vendors could not efficiently exchange designing with one another. Also they were provided DoD, descriptions of their chips in different hardware description language. Reuse was also an issue, thus a need for a standard language for design and documentation of the digital system was generated.

Capabilities of VHDL

1. It is used as an exchange medium between different chip vendors and CAD tool users.
2. It can be used for communication medium between different CAD and CAE tools.
3. Digital system can be modeled as a set of interconnected components. Each component in turn can be modeled as a set of interconnected components further.
4. It supports flexible design methodologies: Top-down Bottom-up mixed
5. It is not technology specific but it is capable of supported technology specific features.
6. It supports both synchronous and asynchronous timing modules.
7. It is an IEEE and ANSI standard.
8. It supports three basic different description styles.
9. It supports a wide range of abstraction levels ranging from abstract behavioral descriptors to vary precise gate level descriptions.
10. It has elements that make large scale design modeling easier such as components, functions and procedures and package.

11. It is publically available, human readable and above all, it is not proprietary.

**Package:**

It provides convenient mechanism to store and share declarations that are common across many design units. Standard package used is IEEE std_logic_1164. It is decided by IEEE and ANSI.

**Hardware abstraction:**

VHDL is used to describe a model for digital hardware device. This model specifies the external view of device and one or more internal views. The internal views describe the functionality or the structure while the external view specifies the interface through which it communicates with other modes in the environment. Figure 1.1 shows hardware device and corresponding software of the device.

**Figure 1.1 Device versus device model.**

In VHDL each device model is treated as a distinct representation of unique device, called an entity. Figure 1.2 shows VHDL view of hardware device that has multiple device models, with each device model representing an entity. Each entity is described using one model, which contains one external view and one or more internal views.

**Figure 1.2 A VHDL view of a device.**
VHDL provides 5 different primary constructs called the design units. They are-

1. **Entity Declaration**
   It describes the external view of entity.

   Ex.- Input-output signal names

2. **Architecture Body**:
   It contains internal description of entity.

   Ex.- A set of inter connected components that represents the structure of entity or set of concurrent or sequential statements that represent the behavior of entity.

3. **Configuration Declaration**:
   It is used to create an entity; it specifies the binding of one architecture body from many architecture bodies that may be associated with the entity. It may also specify the binding of components used in selected architecture body to other entities. An entity may have number of different configuration.

4. **Package Declaration**:
   A package declaration is used to store a set of common declarations like components, types, procedures, and functions. These declarations can then be imported into other design units using a context clause.

5. **Package Body**:
   A package body is primarily used to store the definitions of functions and procedures that were declared in the corresponding package declaration, and also the complete constant declarations for any deferred constants that appear in the package declaration. Therefore, a package body is always associated with a package declaration.
Steps to implement the design

Step 1: Start the Xilinx project navigator by Stat->programs->Xilinx ISE->Project Navigator

Step 2: In the project navigator window click on new project->give file name->next.

Step 3: In the projector window right click on project name-> new source->VHDL module->give file name->define ports->finish.

Step 4: Write the VHDL code for any gate or circuit.

Step 5: Check Syntax and remove error if present.

Step 6: Simulate design using Modelsim.

Step 7: In the project navigator window click on simulation->click on simulate behavioral model.

Step 8: Give inputs by right click on any input->force constant

Step 9: Run simulation

Step 10: Analyze the waveform.
LAB VLSI

BTEC-605
Internal Marks: 30
External Marks: 20
Total Marks: 50

List of Experiments

Combinational Design Exercises
1. Design of basic Gates: AND, OR, NOT.
2. Design of universal gates.
3. Design of 2:1 Mux using other basic gates.
4. Design of 2 to 4 Decoder.
7. Design of 8:3 Priority Encoder.
8. Design of 4 Bit Binary to Grey code Converter.
10. Design an 8 Bit parity generator (with for loop and Generic statements).

Sequential Design Exercises
12. Design of all type of Flip-Flops using (if-then-else) Sequential Constructs
13. Design of 8-Bit Shift Register with shift Right, shift Left, Load and Synchronous reset.
15. Design of Synchronous 8-Bit universal shift register (parallel-in, parallel-out) with 3-state output (IC 74299).
17. Design a decimal up/down counter that counts up from 00 to 99 or down from 99 to 00.
18. Design 3-line to 8-line decoder with address latch.

Learning beyond Syllabus
1. Design memory using VHDL.
2. Design Arithmetic Logical Unit (ALU) using VHDL.
Experiment No.-1

Aim: Write VHDL code for basic gates: AND, OR, NOT.

Apparatus: Xilinx ISE 8.1 software

AND Gate:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity AND1 is
  port (a : in STD_LOGIC;  b : in STD_LOGIC;  c : out STD_LOGIC);
end AND1;
architecture behavioral of AND1 is
begin
  process (a, b)
  begin
    if (a='1' and b='1')
      then
        c<='1';
      else
        c<='0';
      end if;
  end process;
end behavioral;
```
Snapshots of VHDL code of AND gate:

```vhdl
entity andgate is
    Port ( a : in STD_LOGIC;
            b : in STD_LOGIC;
            c : out STD_LOGIC);
end andgate;

architecture Behavioral of andgate is
begin
    process (a,b)
    begin
        if (a='1' and b='1')
            c := '1';
        else
            c := '0';
        end if;
    end process;
end Behavioral;
```
OR Gate:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity OR1 is
port (a : in STD_LOGIC; b : in STD_LOGIC; c : out STD_LOGIC);
end OR1;
architecture behavioral of OR1 is
begin
process (a, b)
begin
if (a='0' and b='0')
then
    c<='0';
else
    c<='1';
end if;
end process;
end behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity NOT1 is
port (a : in STD_LOGIC; c : out STD_LOGIC) ;
end NOT1;
architecture behavioral of NOT1 is
begin
process (a)
begin
if (a='0')
then
  c<=’1’;
else
  c<=’0’;
end if;
end process;
end behavioral;
Experiment No.-2

Aim: Write VHDL code for universal logic gates: NAND, NOR and XOR, XNOR gates using basic gates.

Apparatus: Xilinx ISE 8.1 software

NAND Gate:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity NAND1 is
port (a : in STD_LOGIC; b : in STD_LOGIC; c : out STD_LOGIC); end NAND1;
ar ar chitecture behavioral of NAND1 is
begin
process (a, b)
begin
if (a='1' and b='1') then
c<= '0';
else
c<= '1';
end if;
end process;
end behavioral;
Snapshot of VHDL code of NAND gate:

```vhdl
entity nand is
  Port ( a : in STD_LOGIC;
       b : in STD_LOGIC;
       c : out STD_LOGIC);
end nand;

architecture Behavioral of nand is
begin
  process (a, b)
  begin
    if (a='1' and b='1') then
      c <= '0';
    else
      c <= '1';
    end if;
  end process;
end Behavioral;
```
NOR Gate:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity NOR1 is
  port (a : in STD_LOGIC;  b : in STD_LOGIC;   c : out STD_LOGIC) ;
end NOR1;

architecture behavioral of NOR1 is
begin
  process (a, b)
  begin
    if (a='0' and b='0')
    then
      c<='1';
    else
      c<='0';
    end if;
  end process;
end behavioral;
XOR Gate using basic gates:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity XOR1 is
port (a : in STD_LOGIC; b : in STD_LOGIC; c : out STD_LOGIC);
end XOR1;

architecture behavioral of XOR1 is
begin
process (a, b)
variable (s1, s2, s3, s4:STD_LOGIC)
begin
s1:= NOT a;
s2:= NOT b;
s3:= s1 AND b;
s4:= s2 AND a;
c<= s3 OR s4;
end process;
end behavioral;
Experiment No.-3

Aim: Write VHDL code for 2:1 mux using other basic gates.

Apparatus: Xilinx ISE 8.1 software

2:1 mux:

A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.

\[
Z = (A \cdot \overline{S}) + (B \cdot S)
\]

<table>
<thead>
<tr>
<th>S</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux_2 to 1 is
port (a : in STD_LOGIC; b : in STD_LOGIC; s : in STD_LOGIC z : out STD_LOGIC) ;
end mux_2 to 1 is;
architecture behavioral of mux_2 to 1 is
begin
process (a, b, s)
begin
```

```
if (s='0') then
  z<=a;
else
  z<=b;
end if;
end process;
end behavioral;

Snapshot of MUX 2:1 VHDL code:
2:1 mux using gates:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux_2 to 1 is
port (a : in STD_LOGIC; b : in STD_LOGIC; s : in STD_LOGIC  z : out STD_LOGIC) ;
end mux_2 to 1 ;

architecture behavioral of mux_2 to 1 is
begin
process (a, b, s)
variable (s1, s2, s3:STD_LOGIC)
begin
s1:=NOT s;
s2:=s1 AND a;
s3:=s AND b;
z<=s2 OR s3;
end process;
end behavioral;
Experiment No.-4

Aim: Write VHDL code for 2:4 decoder.

Apparatus: Xilinx ISE 8.1 software

Introduction:

A decoder is a combinational circuit that converts binary information from n inputs line to a maximum of 2^n unique output lines.

![2-4 Decoder Diagram]

<table>
<thead>
<tr>
<th>E</th>
<th>A</th>
<th>B</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder_2_to_4 is
port ( a : in STD_LOGIC_VECTOR; E : in STD_LOGIC;
    d : out STD_LOGIC_VECTOR (3 downto 0) );
end decoder_2_to_4 ;
architecture behavioral of decoder_2_to_4 is
begin
process (a)
begin
    case a is
    when "00" => d<="0001";
    when "01" => d<="0010";
    when "10" => d<="0100";
    when others => d<="1000";
    end case;
end process;
end behavioral;

Waveform of 2:4 decoder:
2:4 mux using data flow modeling:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder_2_to_4 is
port (a : in STD_LOGIC; b : in STD_LOGIC; E : in STD_LOGIC; d : out
STD_LOGIC_VECTOR (3 downto 0)) ;
end decoder_2_to_4;

architecture dataflow of decoder_2_to_4 is
signal (abar, bbar: STD_LOGIC)
begin
abar <= NOT a;
bbar <= NOT b;
d(0) <= abar AND bbar AND E;
d(1) <= abar AND b AND E;
d(2) <= a AND bbar AND E;
d(3) <= a AND b AND E;
end dataflow;
Experiment No.-5

Aim: Write VHDL code for Half-adder, full-adder, half-subtractor and full-subtractor.

Apparatus: Xilinx ISE 8.1 software

Half-adder:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity half_adder is
port (a : in STD_LOGIC; b : in STD_LOGIC; s : out STD_LOGIC; c : out STD_LOGIC);
end half_adder;
architecture behavioral of half_adder is
begin
process (a,b)
begin
if (a='0' and b='0')
then
s<='0';
c<='0';
elsif (a='1' and b='1')
s<='0';
c<='1';
else
s<='1';
c<='0';
end if;
end process;
end behavioral;

Truth table for half-adder:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
**Full-Adder:**

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity full_adder is
port (a : in STD_LOGIC_VECTOR (0 to 2); s : out STD_LOGIC_VECTOR (0 to 1));
end full_adder;
architecture behavioral of full_adder is
begin
    process (a)
    begin
        case a is
            when “000” => s<="00";
            when “001” => s<="10";
            when “010” => s<="10";
            when “011” => s<="01";
            when “100” => s<="10";
            when “101” => s<="01";
            when “110” => s<="01";
            when others => s<="11";
        end case;
    end process;
end behavioral;

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Snapshots:

---

```vhdl
-- any Xilinx primitives in this code.
--library UNISON;
--use UNISON.VComponents.all;

entity fa is
  Port ( a : in STD_LOGIC_vector (0 to 2);
         s : out STD_LOGIC_vector (0 to 1));
end fa;

architecture Behavioral of fa is
  begin
    process (a)
    begin
      case a is
        when "000" => s <= "00";
        when "001" => s <= "01";
        when "010" => s <= "10";
        when "011" => s <= "11";
        when "100" => s <= "01";
        when "101" => s <= "10";
        when "110" => s <= "00";
        when others => s <= "11";
      end case;
    end process;
    end Behavioral;
```
Half-subtractor:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity half_sub is
port (a : in STD_LOGIC; b : in STD_LOGIC;
d : out STD_LOGIC; b1 : out STD_LOGIC);
end half_sub;
architecture behavioral of half_sub is
begin
  process (a,b)
  begin
    if (a=b) then
d<='0';
else
d<='1';
end if;
if (a='0' and b='1') then
b1<='1';
else
b1<='0';
end if;
  end process;
end behavioral;

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Diff</th>
<th>Borrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Full-Subtractor:**

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

useIEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_subs is
port (a : in STD_LOGIC_VECTOR (0 to 2); s : out STD_LOGIC_VECTOR (0 to 1));
end full_subs;

architecture behavioral of full_subs is
begin
process (a)
begin
case a is
when "000" => s<="00";
when "001" => s<="11";
when "010" => s<="11";
when "011" => s<="01";
when "100" => s<="10";
when "101" => s<="00";
when "110" => s<="00";
when others => s<="11";
end case;
end process;
end behavioral;

**Truth-table for full-subtractor:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Diff</th>
<th>Borrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Experiment No.-6

Aim: Write VHDL code for 3:8 decoder.

Apparatus: Xilinx ISE 8.1 software

Truth-table for 3:8 decoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Outputs</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder_3_to_8 is
  port (a : in STD_LOGIC_VECTOR (2 downto 0);
        d : out STD_LOGIC_VECTOR (7 downto 0));
end decoder_3_to_8;
architecture behavioral of decoder_3_to_8 is
begin
process (a)
begin
case a is
when “000”=> d<>“00000001”;
when “001”=> d<>“00000010”;
when “010”=> d<>“00000100”;
when “011”=> d<>“00001000”;
when “100”=> d<>“00010000”;
when “101”=> d<>“00100000”;
when “110”=> d<>“01000000”;
when others=> d<>“10000000”;
end case;
end process;
end behavioral;
Experiment No.-7

Aim: Write VHDL code for 8:3 priority encoder.

Apparatus: Xilinx ISE 8.1 software

Introduction:

An encoder is a digital circuit that performs inverse operation of decoder. An encoder has $2^n$ input lines and n output lines. The output lines generate the binary code corresponding to the input value.

Truth-table for 8:3 priority encoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7 A6 A5 A4 A3 A2 A1 A0</td>
<td>D2 D1 D0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>X X X</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 X</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 1 X X</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 0 0 0 1 X X X</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 0 0 1 X X X X</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 1 X X X X X</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0 1 X X X X X X</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 X X X X X X X</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity p_encoder_8_to_3 is
port (a : in STD_LOGIC_VECTOR (7 downto 0);
      d : out STD_LOGIC_VECTOR (2 downto 0));
end p_encoder_8_to_3;

architecture behavioral of p_encoder_8_to_3 is
begin
  process (a)
  begin
    case a is
      when "00000001" => d<="000";
      when "0000001X" => d<="001";
      when "000001XX" => d<="010";
      when "00001XXX" => d<="011";
      when "0001XXXX" => d<="100";
      when "001XXXXX" => d<="101";
      when "01XXXXXX" => d<="110";
      when "1XXXXXXX" => d<="111";
      when others => d<="XXX";
    end case;
  end process;
end behavioral;
Experiment No.-8

Aim: Design of 4 Bit Binary to Grey code Converter.

Apparatus: Xilinx ISE 8.1 software

Introduction:

The binary to grey converter is a combinational circuit that takes binary number as input and converts it into grey code. Grey code differs from the preceding and succeeding number by a single bit.

VHDL Code:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity b2g is
port (b : in std_logic_vector (3 downto 0); g : out std_logic_vector (3 downto 0));
end b2g;

architecture behavioral of b2g is
begin
process (b)
begin
begin
    case b is
    when "0000" => g<= "0000";
    when "0001" => g<= "0001";
    when "0010" => g<= "0011";
    when "0011" => g<= "0010";
    when "0100" => g<= "0110";
    when "0101" => g<= "0111";
    when "0110" => g<= "0101";
    when "0111" => g<= "0100";
    when "1000" => g<= "1100";
    when others => g<= "0000";
    end case;
end;
end process;
end;
```

when "1001" => g <= "1101";
when "1010" => g <= "1111";
when "1011" => g <= "1110";
when "1100" => g <= "1010";
when "1101" => g <= "1011";
when "1110" => g <= "1001";
when others => g <= "1000";
end case;
end process;
end behavioral;
Data flow model for binary to grey code converter:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

dataflow entity bin2grey_conv is
  port (b : in std_logic_vector (3 downto 0); g : out std_logic_vector (3 downto 0));
end bin2grey_conv;

architecture dataflow of bin2grey_conv is
begin
  g(3)<=b(3);
g(2)<= (b(3)) xor (b(2));
g(1)<= b(2) xor b(1);
g(0)<= b(1) xor b(0);
end dataflow;

Schematic of Binary to grey code converter:
Experiment No.-10

Aim: Write VHDL code for parity generator using for loop and generic statement.
Apparatus: Xilinx ISE 8.1 software

Introduction:
The parity generator is a combinational circuit that adds additional bit (parity) to the pattern so that number of total bits is odd or even. If a bit is added so that the total number of 1’s becomes odd then it is called odd parity and if the number of 1’s is even then it is called even parity.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

dentity paritygen is
generic (n:integer:=7);
port (a : inbit_vector (n downto 0); z : outbit_vector (n downto 0));
end paritygen;
architecture behavioral of paritygen is
begin
process (a)
variable temp1:bit;
variable temp2:bit_vector(z’range);
begin
  temp1:= ‘0’;
  for i in a’range loop
    temp1:=temp1 xor a(i);
    temp2(i):=a(i);
  end loop;
  temp2(z’high):=temp1;
  z<=temp2;
end process;
end behavioral;
Experiment No.-11

Aim: Design of 2’s Complementary for 8-bit Binary number using Generate statements.

Apparatus: Xilinx ISE 8.1 software

Introduction:
The 8-bit binary completer is a combinational circuit that takes 8-bit binary number and complements all the bits.

VHDL code:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity complement8bit is
  port (a : in std_logic_vector (7 downto 0); b : out std_logic_vector (7 downto 0));
end complement8bit;

architecture behavioral of complement8bit is
begin
  for i in 0 to 7 generate
    b(i) <= not (a);
  end generate;
end behavioral;
Experiment No.-12

Aim: Design of all type of Flip-Flops using (if-then-else) Sequential Constructs

Apparatus: Xilinx ISE 8.1 software

(1) S-R flip-flop:

<table>
<thead>
<tr>
<th>( S )</th>
<th>( R )</th>
<th>( Q_{n+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( Q_n )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity flipflop_SR is
    port (s, r, clk, rst : in std_logic; q : out std_logic);
end flipflop_SR;
architecture behavioral of flipflop_SR is
begin
process (s, r, clk, rst)
begin
if (clk= ‘1’ and clk’event) then
if (rst= ‘1’) then
q<= ‘0’;
end
end
end process;
end behavioral;
elsif (rst= ‘0’) then
q<= ‘1’;
elif (s= ‘0’ and r= ‘0’ and rst= ‘0’) then
q<=q;
elif (s= ‘0’ and r= ‘1’ and rst= ‘0’) then
q<= ‘0’;
elif (s= ‘1’ and r= ‘0’ and rst= ‘0’) then
q<= ‘1’;
elif (s= ‘1’ and r= ‘1’ and rst= ‘0’) then
q<= ‘U’;
end if;
end if;
end process;
end behavioral;

(2) J-K flip-flop:

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q_n</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Not Q_n</td>
</tr>
</tbody>
</table>

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity flipflop_JK is
    port (j, k, clk, rst : in std_logic; q : inout std_logic);
end flipflop_JK;
architecture behavioral of flipflop_JK is
begin
process (j, k, clk, rst)
begin
if (clk= ‘1’ and clk’event) then
if (rst= ‘1’) then
q<= ‘0’;
elsif (rst= ‘0’) then
q<= ‘1’;
elsif (j= ‘0’ and k= ‘0’ and rst= ‘0’) then
q<=q;
elsif (j= ‘0’ and k= ‘1’ and rst= ‘0’) then
q<= ‘0’;
elsif (j= ‘1’ and k= ‘0’ and rst= ‘0’) then
q<= ‘1’;
elsif (j= ‘1’ and k= ‘1’ and rst= ‘0’) then
q<= NOT q;
end if;
end if;
end process;
end behavioral;
(3) D flip-flop:

![D Flip-flop Diagram]

<table>
<thead>
<tr>
<th>D</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity flipflop_D is
  port (d, clk, rst : in std_logic; q : inout std_logic);
end flipflop_D;

architecture behavioral of flipflop_D begin
  process (d, clk, rst)
  begin
    if (clk= '1' and clk'event) then
      if (rst= '1') then
        q<= '0';
      else
        q<=q;
      end if;
    end if;
  end process;
end behavioral;
```
Experiment No.-13

Aim: Design of 8-bit shift register.

Apparatus: Xilinx ISE 8.1 software

Introduction:

The term register can be used in variety of specific applications, but in all cases it refers to a group of flip-flops operating as a coherent unit to hold data. This is different from counter, which is a group of flip-flops operating to generate new data by tabulating it.

The demonstration circuit below is known as shift register because data is shifted through it, from flip-flop to flip-flop. If we apply 8-bits of data to the initial data input and apply one clock pulse to the circuit after setting each bit of data, we will find the entire byte present at the flip-flop outputs in parallel format. Therefore, this circuit is known as a serial-in, parallel-out shift register.

VHDL Code:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity leftshift is
  port (a : inout bit vector (0 to 7); r, l, rst, load, clk : in bit; q : out bit vector (0 to 7));
end leftshift;

architecture behavioral of leftshift is
begin
  process (load, rst, a, clk)
  begin
    if (clk= ‘1’ and clk’event) then
      if (load= ‘1’) then
        q<=a;
      elsif(load= ‘0’) then
        if (rst= ‘1’) then
          q<= “00000000”;  
        else
          if (l= ‘1’) then
            q<=a sll l;
          end if;
          if (r= ‘1’) then
            q<= a srl l;
          end if;
        end if;
      end if;
    end if;
  end process;
end behavioral;
Experiment No.-14

Aim: Design of Synchronous 8-bit Johnson Counter.

Apparatus: Xilinx ISE 8.1 software

Introduction:

A Johnson counter is a modified ring counter, where the inverted output from the last flip flop is connected to the input to the first. The register cycles through a sequence of bit-patterns. The MOD of the Johnson counter is 2^n if n flip-flops are used. The main advantage of the Johnson counter is that it only needs half the number of flip-flops compared to the standard ring counter for the same MOD.

VHDL Code:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Johnson is
  port (a : inout bit_vector (3 downto 0):= "1000" ; clk : in std_logic);
end johnson;

architecture behavioral of Johnson is
signal clock : std_logic:= ‘0’;
begin
process (clk)
variable d : integer:= 0;
begin
if (clk= ‘1’ and clk’event) then
  d := d+1;
  if (d= 10000000) then
    clock<=not clock;
end if;
end if;
end process;
end johnson;
d := 0;
end if;
end if;
end process;

process (clock)
variable v : bit;
begin
if (clock= '1' and clock’event) then
v := a(0);
for i in a’high-1 downto a’low loop
a(i)<=a(i+1);
end loop;
a(a’length-1)<=not v;
end if;
end process;
end behavioral;
Experiment No.-15

**Aim**: Design of Synchronous 8-Bit universal shift register (parallel-in, parallel-out) with 3-state output.

**Apparatus**: Xilinx ISE 8.1

VHDL Code:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PIPO is
  port (a : inout bit_vector (0 to 7); r, l, rst, load, clk : in bit; q : out bit_vector (0 to 7));
end leftshift;

architecture behavioral of PIPO is
begin
process (load, rst, a, clk)
begin
if (clk= ‘1’ and clk’event) then
  if (load= ‘1’) then
    q <= a;
```
elsif(load='0') then
    if (rst='1') then
        q <= "00000000";
    else
        if (l='1') then
            q <= a sll;
        end if;
        if (r='1') then
            q<= a srl;
        end if;
    end if;
end if;
end elseif;
end if;
end process;
end behavioral;
Experiment No.-16

**Aim:** Design of Counters (MOD 3, MOD 5 etc.)

**Apparatus:** Xilinx ISE 8.1

**Introduction:** A sequential circuit that goes through a prescribed sequence of states upon application of input pulse is called a counter. A counter that follows binary sequence is called binary counter.

**Synchronous counter:** In synchronous counters all flip-flops are given common clock pulse.

**Asynchronous counter:** In this type of counter the output of one flip-flop acts as a clock for next flip-flop.

**Design Description:**

Step 1: Receive the design specification.
Step 2: Study specification in detail to get an insight into the real operational behavior of the circuit.
Step 3: Make a block diagram model of the design. Identify all inputs and outputs.
Step 4: Design a primitive diagram based on information obtained from above steps.
Step 5: Develop a primitive state table from primitive state diagram.
Step 6: Develop a simplified state diagram from the simplified state table.
Step 7: Make a state assignment.
Step 8: Develop a present/next state table using assignments from the simplified state diagram with state assignment.
Step 9: Develop the next state maps using the present/next state table with state assignments.
Step 10: Make a selection for memory elements i.e.flip-flops.
Step 11: Develop output decoder logic.
Step 12: Draw the schematic diagram.
MOD 3 Counter:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mod3 is
  port (clk, rst : in std_logic; c : out std_logic_vector (0 to 1));
end mod3;
architecture behavioral of mod3 is
signal a : std_logic_vector (0 to 1):= "00";
signal clock : std_logic:= '0';
begin
process (clk, rst)
  variable d : integer:= 0;
  --variable clock : std_logic;
begin
if (clk= '1' and clk'event) then
  d := d+1;
  if (d=10000000) then
    clock<= not clock;
    d := 0;
  end if;
end if;
end if;
end process;
process (clock, rst)
variable a : std_logic_vector (0 to 1) := "00";
begin
if (rst = '1') then
  a := "00";
end if;
a := a+ “01”;
if (a = “11”) then
a := “00”;
end if;
end if;
c <= a;
end process;
end behavioral;

MOD 5 Counter:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mod5
is
port (clk, rst : in std_logic; c : out std_logic_vector (0 to 2));
end mod5;

architecture behavioral of mod5

signal clock : std_logic:= ‘0’;

begin
process (clk, rst)
variable d : integer:= 0;
--variable clock : std_logic;

begin
if (clk= ‘1’ and clk’event) then
d := d+1;
if (d=10000000) then
clock<= not clock;
d := 0;
end if;
end if;
end process;
end behavioral;
end if;
end process;

process (clock, rst)
variable a : std_logic_vector (0 to 2) := "000";
begin
if (rst = '1') then
a := "000";
elsif(rising_edge (clock)) then
a := a+ "001";
if (a = "100") then
a := "000";
end if;
end if;
c <= a;
end process;
end behavioral;

MOD 7 Counter:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mod7 is
port (clk, rst : in std_logic; c : out std_logic_vector (0 to 2));
end mod7;
architecture behavioral of mod7 is
signal a : std_logic_vector (0 to 2) := "000";
signal clock : std_logic:= '0';
begin
process (clk, rst)
variable d : integer := 0;

begin
if (clk = '1' and clk'event) then
    d := d + 1;
    if (d = 10000000) then
        clock <= not clock;
        d := 0;
    end if;
end if;
end if;
end process;

process (clock, rst)
variable a: std_logic_vector (0 to 2) := "000";
begin
if (rst = '1') then
    a := "000";
elsif (rising_edge (clock)) then
    a := a + "001";
    if (a = "111") then
        a := "000";
    end if;
end if;
c <= a;
end process;
end behavioral;
**Experiment No.-17**

**Aim:** Design a decimal up/down counter that counts up from 00 to 99 or down from 99 to 00.

**Apparatus:** Xilinx ISE 8.1

**VHDL Code:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity count_00to99 is
  port (e : out std_logic_vector (1 downto 0);
        clk : in std_logic; q : out std_logic_vector (6 downto 0));
end count_00to99;

architecture behavioral of count_00to99 is
  signal cli, clock : std_logic := "0";
  signal er : std_logic_vector (0 to 6) := "0000000";
begin
  process (clk)
  variable d : integer range 0 to 5000000 := 0;
  begin
    if (clk’event and clk = ‘1’) then
      d := d +1;
      if (d = 5000000) then
        clock <= not clock;
        cli <= not cli;
        d := 0;
      end if;
    end if;
  end process;
```
process (clock)

variable t : integer range 0 to 10 := 0;

begin

e <= “11”;

if rising_edge (clock) then

t := t+1;

case t is

when 0 => q <= “0111111”;
when 1 => q <= “0111111”;
when 2 => q <= “0000110”;
when 3 => q <= “1011011”;
when 4 => q <= “1001111”;
when 5 => q <= “1100110”;
when 6 => q <= “1101101”;
when 7 => q <= “1111100”;
when 8 => q <= “0000111”;
when 9 => q <= “1111111”;
when 10 => q <= “1100111”;
if t=10 then

t := 0;

end if;

end case;

end if;

end process;

end behavioral;
Experiment No.-18

Aim: Write VHDL code for 3:8 decoder with address latch.

Apparatus: Xilinx ISE 8.1 software

Truth-table for 3:8 decoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder_3_to_8 is
port (a : in STD_LOGIC_VECTOR (2 downto 0);
    d : out STD_LOGIC_VECTOR (7 downto 0));
end decoder_3_to_8;
architecture behavioral of decoder_3_to_8is
begin
process (a)
begin
case a is
when "000" => d <= "00000001";
when "001" => d <= "00000010";
when "010" => d <= "00000100";
when "011" => d <= "00001000";
when "100" => d <= "00100000";
when "101" => d <= "01000000";
when "110" => d <= "10000000";
when others => d <= "10000000";
end case;
end process;
end behavioral;
Learning beyond Syllabus

Aim: Write a VHDL program for memory.

Tool: Xilinx ISE 8.1 software

Program:

entity mem is

    port (clk, rst : in std_logic;
            mren, mwen : in std_logic; mem_add : in std_logic_vector (0 to 3);
            mem_data : in std_logic_vector (0 to 7);
            mem_out : out std_logic_vector (0 to 7));

end mem;

architecture behavioral of mem is;

type RAM type is array (0 to 15) of std_logic_vector (0 to 7)

signal RAM1 : RAM type;

begin

    write: process (clk, rst, mwen, mem_add, mem_data)

    begin

        if rst = ‘1’ then

            RAM1 <= (RAM1’range => “00000000”);

        elsif clk’event and clk = ‘1’ then

            if mwen = ‘1’ then

                RAM1 (conv_integer (mem_add)) <= mem_data;

            end if;

        end if;

    end process;

    read: process (clk, rst, mren, mem_add)
begin

if (rst = '1') then

mem_out <= "00000000";

elsif clk'event and clk = '1' then

if mren = '1' then

mem_out <= RAM (conv_integer (mem_add));

end if;

end if;

end if;

end process;

end behavioral.
Aim: Write VHDL program to perform Arithmetic Logic Unit (ALU) operation.

Tool: Xilinx ISE 8.1 software

Introduction:

An ALU performs arithmetic and logical operation. It receives data from register file and perform operations on it given by control signals generated through control unit.

<table>
<thead>
<tr>
<th>Sel</th>
<th>Unit</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Arithmetic Unit</td>
<td>$z \leftarrow x$</td>
</tr>
<tr>
<td>001</td>
<td>Arithmetic Unit</td>
<td>$z \leftarrow x+1$</td>
</tr>
<tr>
<td>010</td>
<td>Arithmetic Unit</td>
<td>$z \leftarrow y$</td>
</tr>
<tr>
<td>011</td>
<td>Arithmetic Unit</td>
<td>$z \leftarrow x+y$</td>
</tr>
<tr>
<td>100</td>
<td>Logic Unit</td>
<td>$z \leftarrow \text{not } x$</td>
</tr>
<tr>
<td>101</td>
<td>Logic Unit</td>
<td>$z \leftarrow x \text{ and } y$</td>
</tr>
<tr>
<td>110</td>
<td>Logic Unit</td>
<td>$z \leftarrow x \text{ or } y$</td>
</tr>
<tr>
<td>111</td>
<td>Logic Unit</td>
<td>$z \leftarrow x \text{ xor } y$</td>
</tr>
</tbody>
</table>

```vhdl
entity ALU is
  port (x, y: in std_logic_vector (0 to 7);
   sel: in std_logic_vector (0 to 2);
   z: out std_logic_vector (0 to 7));
end ALU;
```
end ALU;

architecture dataflow of ALU is

signal arith, logic : std_logic_vector (0 to 7);

begin

with sel (0 to 1) select

    arith <= x when "00";

        x+1 when "01";

        y when "10";

        x+y when others;

with sel (0 to 1) select

    logic <= not x when "00";

        x and y when "01";

        x or y when "10";

        x xor y when others;

with sel (2) select

    z <= arith when '0';

        logic when others;

end dataflow;