Department of Electronics and Communication Engineering

HDL Lab

10ECL48

B.E - IV Semester

Lab Manual 2015-16

Name : ______________________________

USN : ______________________________

Batch : ______________ Section : ____________
Department of Electronics and Communication Engineering

HDL Lab

Version 1.0

Prepared by:
Mr. Sanjay C P
Mrs. Smitha U V
Mrs. Sangeetha G
Mrs. Kouser Taj
Assistant Professor

Reviewed by:
Mr. Nagaraja P
Associate Professor

Approved by:
Prof. Rajendra
Professor & Head,
Dept. of ECE
Subject Code : 10ECL48  IA Marks : 25
No. of Practical Hrs/Week : 03  Exam Hours : 03
Total no. of Practical Hrs.: 42  Exam Marks : 50

Note: Programming can be done using any compiler. Download the programs on a FPGA/CPLD boards such as Apex/Acex/Max/Spartan/Sinfi/TK Base or equivalent and performance testing may be done using 32 channel pattern generator and logic analyzer apart from verification by simulation with tools such as Altera/Modelsim or equivalent.

PROGRAMMING (using VHDL / Verilog)

1. Write HDL code to realize all the logic gates
2. Write a HDL program for the following combinational designs
   a. 2 to 4 decoder
   b. 8 to 3 (encoder without priority & with priority)
   c. 8 to 1 multiplexer
   d. 4 bit binary to gray converter
   e. Multiplexer, de-multiplexer, comparator.
3. Write a HDL code to describe the functions of a Full Adder Using three modeling styles.
4. Write a model for 32 bit ALU using the schematic diagram shown below

   - ALU should use combinational logic to calculate an output based on the four bit op-code input.
   - ALU should pass the result to the out bus when enable line in high, and tri-state the out bus when the enable line is low.
• ALU should decode the 4 bit op-code according to the given in example below.

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>ALU OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>A + B</td>
</tr>
<tr>
<td>2.</td>
<td>A – B</td>
</tr>
<tr>
<td>3.</td>
<td>A Complement</td>
</tr>
<tr>
<td>4.</td>
<td>A * B</td>
</tr>
<tr>
<td>5.</td>
<td>A AND B</td>
</tr>
<tr>
<td>6.</td>
<td>A OR B</td>
</tr>
<tr>
<td>7.</td>
<td>A NAND B</td>
</tr>
<tr>
<td>8.</td>
<td>A XOR B</td>
</tr>
</tbody>
</table>

5. Develop the HDL Code for the following flip-flops: SR, D, JK, and T

6. Design 4-bit binary, BCD counters (Synchronous reset and asynchronous reset) and “any sequence” counters

**INTERFACING** (at least four of the following must be covered using VHDL / Verilog)

1. Write HDL code to display messages on the given seven segment display and LCD and accepting Hex key pad input data.

2. Write HDL code to control speed, direction of DC and Stepper motor.

3. Write HDL code to accept 8 channel Analog signal, Temperature sensors and display the data on LCD panel or Seven segment display.

4. Write HDL code to generate different waveforms (Sine, Square, Triangle, Ramp etc.,) using DAC change the frequency and amplitude.

5. Write HDL code to simulate Elevator operations

6. Write HDL code to control external lights using relays.
OBJECTIVES

- To introduce Xilinx compiler and in-built simulator
- To describe the simulation and synthesis of the systems using Hardware Description Languages and explain its various abstraction levels.
- To code, generate and implement on FPGA Kit.
- To interface the FPGA kit with different external devices.

OUTCOMES

The student will be able to:

- Write efficient hardware designs both in VHDL and Verilog and perform high-level HDL simulation, synthesis and verify the expected output.
- Explain different levels of abstraction with the programming examples.
- To generate and implement the program/s on FPGA Kit.
- Interface the FPGA with different external devices such as motors, relays, DAC, seven segment and LCD displays.
GENERAL INSTRUCTIONS TO STUDENTS

1. Students should come with thorough preparation for the experiment to be conducted.

2. Students should take prior permission from the concerned faculty before availing the leave.

3. Students should come with formals and to be present on time in the laboratory.

4. Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.

5. Students will be permitted to attend laboratory unless they bring the observation book fully completed in all respects pertaining to the experiment conducted in the present class.

6. They should obtain the signature of the staff-in-charge in the observation book after completing each experiment.

7. Practical record and observation book should be maintained neatly.
<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Name of the Experiment</th>
<th>Date</th>
<th>Manual Marks (Max. 25)</th>
<th>Record Marks (Max. 10)</th>
<th>Signature (Student)</th>
<th>Signature (Faculty)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>PART A</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>HDL code to realize all the logic gates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>HDL program for the following combinational designs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a. 2 to 4 decoder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b. 8 to 3 (encoder without priority &amp; with priority)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c. 8 to 1 multiplexer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d. 4 bit binary to gray converter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>e. Di-multiplexer, comparator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>HDL code to describe the functions of a Full Adder Using three modeling styles.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>HDL code to realize 32 bit ALU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>HDL Code for the following flip-flops: SR, D, JK, and T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>4-bit binary ,BCD counters (Synchronous reset and asynchronous reset) and “any sequence” counters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>PART B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>HDL code to display numbers on the given seven segment display using Hex keypad input data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>HDL code to control speed, direction of DC and Stepper motor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>HDL code to generate different waveforms (Sine, Square, Triangle, Ramp etc.,) using DAC change the frequency.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>HDL code to control external lights using relays</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# INDEX

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>EXPERIMENT NAMES</th>
<th>Page. No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>HDL code to realize all the gates</td>
<td>1 - 4</td>
</tr>
</tbody>
</table>
| 2.     | HDL program for the following combinational designs  
 a) 2 to 4 decoder  
 b) 8 to 3 encoder (without priority & with priority)  
 c) 8 to 1 multiplexer  
 d) 4 bit binary to gray converter  
 e) Demux and Comparator | 5 - 32   |
| 3.     | HDL code to describe the functions of a Full adder using all modeling styles. | 33 - 40  |
| 4.     | HDL model for 8-bit ALU. | 41 - 44  |
| 5.     | HDL codes for the following flip-flops, SR, D, JK, T. | 45 - 60  |
| 6.     | HDL code for 4-bit binary, BCD counters (synchronous reset and asynchronous reset) and “any sequence” counters. | 61 – 80  |

## INTERFACING PROGRAMS

<table>
<thead>
<tr>
<th></th>
<th>EXPERIMENT NAMES</th>
<th>Page. No</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>HDL code to display messages on the Seven Segment Display.</td>
<td>81 – 84</td>
</tr>
<tr>
<td>8.</td>
<td>HDL code to control speed, direction of dc and stepper motor.</td>
<td>85 – 90</td>
</tr>
<tr>
<td>9.</td>
<td>VHDL code to generate different waveforms (Square, Triangle, Ramp etc) using DAC change the frequency and amplitude.</td>
<td>91 – 98</td>
</tr>
<tr>
<td>10.</td>
<td>VHDL code to control lights using relays.</td>
<td>99-100</td>
</tr>
<tr>
<td>11.</td>
<td>HDL code to accept 8 channel analog signals, temperature sensors and display the data on LCD panel or Seven Segment Display.</td>
<td>101 -112</td>
</tr>
<tr>
<td>12.</td>
<td>VHDL code to simulate elevator operations.</td>
<td>113-122</td>
</tr>
</tbody>
</table>

## EXTRA PROGRAMS

<table>
<thead>
<tr>
<th></th>
<th>EXPERIMENT NAMES</th>
<th>Page. No</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.</td>
<td>HDL code to 4-Bit Braun multiplier</td>
<td>123-124</td>
</tr>
</tbody>
</table>

Sample Viva Questions

Question Bank
**Introduction to Xilinx ISE**

Xilinx ISE means Xilinx® Integrated Software Environment (ISE), i.e. programmable logic design tool in electronics industry. This Xilinx® design software suite allows taking design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes design through several steps in the ISE design flow. These steps are Design Entry, Synthesis, Implementation, Simulation/Verification, and Device Configuration. Xilinx is one of most popular software tool used to synthesize VHDL code.

**Tool Procedure:**

1. Double click on Project Navigator Icon.
2. Select new project in file menu.
3. Enter the project name and location as shown below and hit **Next**
4. Select the Family, Device, Package and speed as per the requirements and hit Next

5. Create a new source by using new source icon or right click on the device/project folder to create new source.
6. Select the appropriate source type and enter the file name in New Source Wizard window and hit Next
7. Enter the architecture name – dataflow/behavioral/structural, port name and select the direction. This will create .vhd source file. Hit Next and finish the initial project creation.

8. Write complete VHDL/Verilog code implementation and save.

9. Click on implementation and check for syntax using “Check syntax” option under synthesize tab. If any error, edit and correct VHDL/Verilog code and repeat check syntax until zero errors.

10. Double click on ISIM simulator by selecting simulation mode to complete the functional simulation of your design.
**Procedure for Interfacing:**
1. Repeat the steps 1 to 10 from the tool procedure.
2. Make the connection between appropriate FRC’s of the FPGA board and the DIP switch connector of the GPIOcard-2/
3. Make the connection between appropriate FRC’s of the FPGA board and the LED connector of the GPIOcard-2.
4. Right click on the device and select “**New Source**”, Select the option “Implementation constraint File” and provide the file name and click on next and then hit Finish. This creates an `.ucf` file.

![Image of New Source Wizard]

5. 

6. Double click on the added `.ucf` file and assign the pin numbers to inputs and outputs referring to FRC sheet using the syntax as shown. Save the constraint file.

![Image of Constraint File Editor]
7. Connect USB programmer for FPGA between FPGA kit and USB port of your computer.
8. Go to process window, select the Vhdl or Verilog file and click on “configure target device”.
9. Click OK for the warning below.
10. Select boundary scan to impact the target device.
11. Right click on the impact window to establish a connection between system and FPGA by selecting “INITIALIZE CHAIN” option.

12. Both prom device and FPGA device gets identified after step 10 and bypass the procedure to select only FPGA which of main interest.

13. Now choose device 2 (FPGA Xc6Slx9) and hit ok to complete the impact.
14. Now right click on the device to assign a new .bit file by selecting an option “ASSIGN NEW CONFIGURATION FILE”.

15. Select the corresponding .bit file from the project folder and hit Open.

16. Hit No for the following dialog box.
17. Finally right click on the device (xc6slx9) and implement the program by choosing an option “PROGRAM”

Note: The implementation is verified using GPIO cards.

18. Once again select the FPGA device (xc6slx9) by clicking ok and now the program will be identified and succeeded.

Note: The implementation is verified using GPIO cards.
Block Diagram:

![Block Diagram Image]

Logic Diagram:

![Logic Diagram Image]

Truth Table:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>not_op (a_in)</th>
<th>and_op</th>
<th>nand_op</th>
<th>or_op</th>
<th>nor_op</th>
<th>xor_op</th>
<th>xnor_op</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_in</td>
<td>b_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<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>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Experiment 1: HDL Code to Realize all logic gates

VHDL CODE:

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

entity gates is
port (a_in,b_in: in std_logic;
not_op,and_op,nand_op,or_op,nor_op,xor_op,xnor_op:
out std_logic);
end gates;

architecture dataflow of gates is
begin
not_op<= not a_in;
and_op<= a_in and b_in;
nand_op<= a_in nand b_in;
or_op<= a_in or b_in;
nor_op<= a_in nor b_in;
xor_op<= a_in xor b_in;
xnor_op<= a_in xnor b_in;
end ;
Result:
VERILOG CODE:

module gates(a_in, b_in,
    not_op, and_op, nand_op, or_op, nor_op, xor_op, xnor_op);
input a_in, b_in;
output
    not_op, and_op, nand_op, or_op, nor_op, xor_op, xnor_op;
assign not_op = ~a_in;
assign and_op = a_in & b_in;
assign nand_op = ~(a_in & b_in);
assign or_op = a_in | b_in;
assign nor_op = ~(a_in | b_in);
assign xor_op = a_in ^ b_in;
assign xnor_op = ~(a_in ^ b_in);
endmodule
Block Diagram:

```
  d_in
  |    2 |
  |      |
  |      |
  |      |
  |      |
  |      |
inputs

Decoder 2 to 4

  |      |
  |      |
  |      |
  |      |
  |      |
  |      |
  |      |
  |      |
  |      |
  |      |
outputs
d_op
```

Truth Table:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>d_in(1)</td>
</tr>
<tr>
<td>1</td>
<td>X</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>
</tbody>
</table>
Experiment 2a: VHDL Code to Realize 2:4 Decoder

VHDL CODE:

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

entity decoder2_4 is
  port (en: in std_logic;
        d_in: in std_logic_vector (1 downto 0);
        d_op: out std_logic_vector (3 downto 0));
end decoder2_4;
architecture behavioral of decoder2_4 is
begin
  process (en, din)
  begin
    if (en = '1') then
      d_op <= "ZZZZ";
    else
      case (d_in) is
      when "00" => d_op <= "0001";
      when "01" => d_op <= "0010";
      when "10" => d_op <= "0100";
      when "11" => d_op <= "1000";
      when others => null;
    end case;
  end if;
end process;
end;
```
Result:
VERILOG CODE:

module decoder (d_in,en,d_op);
input [1:0] d_in;
input en;
output [3:0] d_op;
reg [3:0] d_op;
always @(d_in,en)
begin
  if (en==1)
    d_op=4'bzzzz;
  else
    case (d_in)
      2'b00: d_op = 4'b0001;
      2'b01: d_op = 4'b0010;
      2'b10: d_op = 4'b0100;
      2'b11: d_op = 4'b1000;
      default: d_op = 4'bxxxx;
    endcase
end
endmodule
**Block Diagram:**

![Block Diagram Image]

**Truth Table:**

<table>
<thead>
<tr>
<th>en</th>
<th>a_in(7)</th>
<th>a_in(6)</th>
<th>a_in(5)</th>
<th>a_in(4)</th>
<th>a_in(3)</th>
<th>a_in(2)</th>
<th>a_in(1)</th>
<th>a_in(0)</th>
<th>y_op(2)</th>
<th>y_op(1)</th>
<th>y_op(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<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>
<td>1</td>
<td>1</td>
</tr>
<tr>
<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>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Experiment 2b(i) : HDL Code to Realize 8:3 Encoder without priority**

**VHDL CODE :**

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

entity encoder8_3 is
port (en: in std_logic;
     a_in: in std_logic_vector (7 downto 0);
     y_op: out std_logic_vector (2 downto 0));
end encoder8_3;
architecture behavioral of encoder8_3 is
begin
    process (en, a_in)
    begin
        if (en = '1') then
            y_op <= "ZZZ";
        else
            case (a_in) is
                when "00000001" => y_op <= "000";
                when "00000010" => y_op <= "001";
                when "00000100" => y_op <= "010";
                when "00001000" => y_op <= "011";
                when "00010000" => y_op <= "100";
                when "00100000" => y_op <= "101";
                when "01000000" => y_op <= "110";
                when "10000000" => y_op <= "111";
                when others => null;
            end case;
        end if;
    end process;
end;
Result:
VERILOG CODE:

module encoder8_3(en, a_in, y_op);
input en;
input [7:0] a_in;
output [2:0] y_op;
reg [2:0] y_op;
always @ (a_in, en)
beg
if(en==1 )
y_op =3’bzzz;
else
    case (a_in)
        8’b00000001: y_op = 3’b000;
        8’b00000010: y_op = 3’b001;
        8’b00000100: y_op = 3’b010;
        8’b00001000: y_op = 3’b011;
        8’b00010000: y_op = 3’b100;
        8’b00100000: y_op = 3’b101;
        8’b01000000: y_op = 3’b110;
        8’b10000000: y_op = 3’b111;
        default: y_op =3’bxxx;
    endcase
end
endmodule
**Block Diagram:**

![Block Diagram Image]

**Truth Table:**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y_op(2)</td>
</tr>
<tr>
<td>a_in(7)</td>
<td>X</td>
</tr>
<tr>
<td>a_in(6)</td>
<td>X</td>
</tr>
<tr>
<td>a_in(5)</td>
<td>X</td>
</tr>
<tr>
<td>a_in(4)</td>
<td>X</td>
</tr>
<tr>
<td>a_in(3)</td>
<td>X</td>
</tr>
<tr>
<td>a_in(2)</td>
<td>X</td>
</tr>
<tr>
<td>a_in(1)</td>
<td>X</td>
</tr>
<tr>
<td>a_in(0)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
**Experiment 2b(ii) : HDL Code to Realize 8:3 Encoder with priority**

**VHDL CODE :**

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

entity prio_enco is
    port (en: in std_logic;
          a_in: in std_logic_vector (7 downto 0);
          y_op: out std_logic_vector (2 downto 0));
end ;

architecture dataflow of prio_enco is
begin

    y_op<="111" when a_in(7)=’1’ else
             "110" when a_in(6)=’1’ else
             "101" when a_in(5)=’1’ else
             "100" when a_in(4)=’1’ else
             "011" when a_in(3)=’1’ else
             "010" when a_in(2)=’1’ else
             "001" when a_in(1)=’1’ else
             "000" ;
end;
Result:
VERILOG CODE:

module prio_enco(en, a_in, y_op);
input en;
input [7:0] a_in;
output [2:0] y_op;
reg [2:0] y_op;
always @ (a_in, en)
begin
    case (a_in)
        8'b00000001: y_op = 3'b000;
        8'b0000001x: y_op = 3'b001;
        8'b000001xx: y_op = 3'b010;
        8'b00001xxx: y_op = 3'b011;
        8'b0001xxxx: y_op = 3'b100;
        8'b001xxxxx: y_op = 3'b101;
        8'b01xxxxxx: y_op = 3'b110;
        8'b1xxxxxxx: y_op = 3'b111;
        default: y_op = 3'bxxx;
    endcase
end
endmodule
**Block Diagram:**

![Block Diagram Image]

**Truth Table:**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sel (2)</td>
<td>sel (1)</td>
</tr>
<tr>
<td>--------</td>
<td>--------</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>
Experiment 2c : HDL Code to Realize 8:1 Multiplexer

VHDL CODE :

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

entity mux8_1 is
port (sel: in std_logic_vector (2 downto 0);
i_in: in std_logic_vector (7 downto 0);
y_out: out std_logic);
end mux8_1;

architecture behavioral of mux8_1 is
begin
process (sel, i_in)
begin
  case (sel) is
    when "000" => y_out<= i_in(0);
    when "001" => y_out<= i_in(1);
    when "010" => y_out<= i_in(2);
    when "011" => y_out<= i_in(3);
    when "100" => y_out<= i_in(4);
    when "101" => y_out<= i_in(5);
    when "110" => y_out<= i_in(6);
    when "111" => y_out<= i_in(7);
    when others => null;
  end case;
end process;
end;
Result:
**VERILOG CODE:**

module mux8_1(i_in, sel, y_out);
input [7:0] a_in;
input [2:0] sel;
output y_out;
reg y_out;
always@ (i_in, sel)
begin
  case (sel)
    3'b000: y_out = i_in[0];
    3'b001: y_out = i_in[1];
    3'b010: y_out = i_in[2];
    3'b011: y_out = i_in[3];
    3'b100: y_out = i_in[4];
    3'b101: y_out = i_in[5];
    3'b110: y_out = i_in[6];
    3'b111: y_out = i_in[7];
    default: y_out = 3'b000;
  endcase
end
endmodule
Block Diagram:

Logical Diagram and expressions:

Truth Table:
**Experiment 2d: HDL Code to realize 4 bit binary to gray convertor**

**VHDL CODE:**

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

entity binary_gray is
  port(b_in: in std_logic_vector(3 downto 0);
  g_op: out std_logic_vector (3 downto 0));
end binary_gray;

architecture dataflow of binary_gray is
begin
  g_op(3)<= b_in(3);
  g_op(2)<= b_in(3) xor b_in(2);
  g_op(1)<= b_in(2) xor b_in(1);
  g_op(0)<= b_in(1) xor b_in(0);
end;

**VERILOG CODE:**

module binary_gray(b_in, g_op);
input [3:0] b_in;
output [3:0] g_op;
assign g_op[3] = b_in[3];
assign g_op[1] = b_in[2] ^ b_in[1];
assign g_op[0] = b_in[1] ^ b_in[0];
endmodule
Result:
Worksheet:
**Block Diagram:**

![Block Diagram Image]

**Truth Table:**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>sel (1)</td>
<td>y_out (3)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Experiment 2e(i) : HDL Code to realize 1:4 DEMUX

VHDL CODE :

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

entity demux1_4 is
port (a_in: in std_logic;
sel: in std_logic_vector (1 downto 0);
y_out: out std_logic_vector (3 downto 0));
end demux1_4;

architecture behavioral of demux1_4 is
begin
process (a_in, sel)
begin
case (sel) is
when "00" => y_out (0) <= a_in; y_out (1) <= '0'; y_out (2) <= '0'; y_out (3) <= '0';
when "01" => y_out (0) <= '0'; y_out (1) <= a_in; y_out (2) <= '0'; y_out (3) <= '0';
when "10" => y_out (0) <= '0'; y_out (1) <= '0'; y_out (2) <= a_in; y_out (3) <= '0';
when "11" => y_out (0) <= '0'; y_out (1) <= '0'; y_out (2) <= '0'; y_out (3) <= a_in;
when others => null;
end case;
end process;
end;
Result:
VERILOG CODE:

module demux1_4(a_in, sel, y_out);
input a_in;
input [1:0] sel;
output [3:0] y_out;
reg [3:0] y_out;
always @(a_in, sel)
begin
  case (sel)
    2'b00:begin y_out[0]=a_in; y_out[1]= 1'b0;
y_out[2]= 1'b0;y_out[3]=1'b0; end
    2'b01: begin y_out[0]= 1'b0;y_out[1]=a_in;
y_out[2]= 1'b0;y_out[3]=1'b0; end
    2'b10: begin y_out[0]= 1'b0;y_out[1]=1'b0;
y_out[2]=a_in; y_out[3]=1'b0; end
    2'b11: begin y_out[0]= 1'b0; y_out[1]= 1'b0;
y_out[2]=1'b0;y_out[3]=a_in; end
    default: y_out=3'b000;
  endcase
end
endmodule
**Block Diagram:**

![Comparator 4bit block diagram]

**Truth Table:**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_in &gt; b_in</td>
<td>a_in = b_in</td>
<td>a_in &lt; b_in</td>
<td></td>
</tr>
<tr>
<td>a_in</td>
<td>b_in</td>
<td>g_op</td>
<td>e_op</td>
<td>L_op</td>
</tr>
<tr>
<td>1100</td>
<td>0011</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Experiment 2e(ii) : HDL Code to realize 4-bit comparator

VHDL CODE :

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

entity comparator is
port (a_in, b_in: in std_logic_vector (3 downto 0) ;
g_op, e_op, L_op: out std_logic);
end comparator;

architecture behavioral of comparator is
begin
  process (a_in, b_in)
  begin
    if (a_in<b_in) then
      L_op<= '1';
    else
      L_op<= '0';
    end if;
    if (a_in>b_in) then
      g_op<= '1';
    else
      g_op<= '0';
    end if;
    if (a_in = b_in) then
      e_op<= '1';
    else
      e_op<= '0';
    end if;
  end process;
end;
Result:
VERILOG CODE:

module comparater(a_in, b_in, L_op, g_op, e_op);
input [3:0] a_in;
input [3:0] b_in;
output L_op;
output g_op;
output e_op;
reg L_op, g_op, e_op;
always @ (a_in, b_in)
begin
    if (a_in < b_in)
        L_op = 1'b1;
    else
        L_op = 1'b0;
    if (a_in > b_in)
        g_op = 1'b1;
    else
        g_op = 1'b0;
    if (a_in == b_in)
        e_op = 1'b1;
    else
        e_op = 1'b0;
end
endmodule
Block Diagram:

Expression:

\[ \text{sum} = a_{\text{in}} \oplus b_{\text{in}} \oplus c_{\text{in}}; \]
\[ \text{carry} = (a_{\text{in}} \cdot b_{\text{in}}) + (b_{\text{in}} \cdot c_{\text{in}}) + (a_{\text{in}} \cdot b_{\text{in}}); \]

Logic Diagram:

Truth Table:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_{\text{in}} )</td>
<td>( b_{\text{in}} )</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>
Experiment 3a : HDL Code to Describe Full adder using 
DATAFLOW Style

VHDL CODE :

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

entity fulladder is
  port (a_in, b_in, c_in: in std_logic;
  sum, carry: out std_logic);
end fulladder;
architecture dataflow of fulladder is
begin
  sum <= a_in xor b_in xor c_in;
  carry<= (a_in and b_in) or (b_in and c_in) or (a_in and b_in);
end dataflow;

VERILOG CODE :

module fulladder(a_in, b_in, c_in, sum, carry);
  input a_in, b_in,c_in;
  output sum, carry;
    assign sum = a_in^b_in^c_in;
    assign carry = (a_in & b_in) | (b_in & c_in) | (a_in & c_in);
endmodule
Experiment 3b: HDL Code to Describe Full adder using Behavioral Style

VHDL CODE:

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

entity fulladder is
  port (abc: in std_logic_vector(2 downto 0);
       sum, carry: out std_logic);
end fulladder;
architecture behavioral of fulladder is
begin
  process(abc)
  begin
    case (abc) is
      when"000"=>sum<='0'; carry<='0';
      when"001"=>sum<='1'; carry<='0';
      when"010"=>sum<='1'; carry<='0';
      when"011"=>sum<='0'; carry<='1';
      when"100"=>sum<='1'; carry<='0';
      when"101"=>sum<='0'; carry<='1';
      when"110"=>sum<='0'; carry<='1';
      when"111"=>sum<='1'; carry<='1';
      when others=>null;
    end case;
  end process;
end ;
VERILOG CODE:

module fulladder(abc, sum, carry);
input [2:0] abc;
output sum, carry;
reg sum, carry;
always@(abc)
begina
case (abc)
  3’b000:begin  sum=1’b0; carry=1’b0;end
  3’b001:begin  sum=1’b1; carry=1’b0;end
  3’b010:begin  sum=1’b1; carry=1’b0;end
  3’b011:begin  sum=1’b0; carry=1’b1;end
  3’b100:begin  sum=1’b1; carry=1’b0;end
  3’b101:begin  sum=1’b0; carry=1’b1;end
  3’b110:begin  sum=1’b0; carry=1’b1;end
  3’b111:begin  sum=1’b1; carry=1’b1;end
endcase
end
endmodule
**Block Diagram:**

![Block Diagram](image)

**Logic Diagram:**

![Logic Diagram](image)

**Truth Table:**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>a_in</td>
<td>b_in</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>
Experiment 3c : HDL Code to Describe Full adder using Structural Style

VHDL CODE :

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

entity fulladder is
port (a_in, b_in, c_in: in std_logic;
     sum, carry: out std_logic);
end fulladder;

architecture structural of fulladder is
  component halfadder is
    port (p, q: in std_logic;
         r, s: out std_logic);
  end component;
  signal temp1, temp2, temp3: std_logic;
begin
  ha1: halfadder port map (a_in, b_in, temp1,temp2);
  ha2: halfadder port map (temp1, c_in, sum, temp3);
  carry <=temp2 or temp3;
end structural;

component program:
entity halfadder is
port (p, q: in std_logic;
     r, s: out std_logic);
end halfadder;
architecture dataflow of halfadder is
begin
  r<= p xor q;
  s<= p and q;
end;
Result:
VERILOG CODE:

module fulladder(a_in, b_in, c_in, sum, carry);
input a_in, b_in, c_in;
output sum, carry;
wire temp1, temp2, temp3;
halfadder ha1 (a_in, b_in, temp1, temp2);
halfadder ha2 (c_in, temp1, sum, temp3);
assign carry = temp3 | temp2;
endmodule

module halfadder(a, b, s, c);
input a, b;
output s, c;
assign s = a ^ b;
assign c = a & b;
endmodule
Block Diagram:

 Opcode Table:

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>ALU OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A + B</td>
</tr>
<tr>
<td>2</td>
<td>A - B</td>
</tr>
<tr>
<td>3</td>
<td>A Complement</td>
</tr>
<tr>
<td>4</td>
<td>A*B</td>
</tr>
<tr>
<td>5</td>
<td>A AND B</td>
</tr>
<tr>
<td>6</td>
<td>A OR B</td>
</tr>
<tr>
<td>7</td>
<td>A NAND B</td>
</tr>
<tr>
<td>8</td>
<td>A XOR B</td>
</tr>
</tbody>
</table>
**Experiment 4 : HDL Code to realize 32-bit ALU**

**VHDL CODE :**

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

entity alu is
  port (a, b: in std_logic_vector (31 downto 0);
        opcode: in std_logic_vector (2 downto 0);
        en: in std_logic;
        y: out std_logic_vector (31 downto 0)
        y_mul: out std_logic_vector (63 downto 0));
end alu;

architecture behavioral of alu is
begin
  process (a, b, opcode, enable)
  begin
    if (en= '1') then
      case (opcode) is
        when "000" => y<=a+b;
        when "001" => y<=a-b;
        when "010" => y<=not a;
        when "011" => y_mul<=a*b;
        when "100" => y<=a and b;
        when "101" => y<=a or b;
        when "110" => y<=a nand b;
        when "111" => y<=a xor b;
        when others =>null;
      end case;
    else
      y<=(others =>'Z');
      y_mul<=(others=>'Z');
    end if;
  end process;
end behavioral;
Result:
VERILOG CODE:

module alu(a, b, sel, en, y, y_mul);
input [31:0] a;
input [31:0] b;
input en;
input [2:0] sel;
output [31:0] y;
output[63:0]y_mul;
reg [31:0] y;
reg [63:0] y_mul;
always @(a, b, sel)
begin
  if (en==1)
    case (sel)
      3'b000:y=a+b;
      3'b001:y=a-b;
      3'b010:y=~a;
      3'b011:y_mul=a*b;
      3'b100:y= a&b;
      3'b101:y=a|b;
      3'b110:y=~(a&b);
      3'b111:y=a^b;
    default:
      begin
        y=32’bz;
        y_mul=64’bz
      end
    endcase
  else
    begin
      y=32’bz;
      y_mul=64’bz
    end
  end
endmodule
Block Diagram:

![SR Flip Flop Diagram]

Truth Table:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst</td>
<td>clk</td>
<td>s  r</td>
</tr>
<tr>
<td>1</td>
<td>↑</td>
<td>X  X</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>0  0</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>0  1</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>1  0</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>1  1</td>
</tr>
</tbody>
</table>
Experiment 5a : HDL Code to Describe SR Flipflop

VHDL CODE:

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

department srff is
port (rst, clk: in std_logic;
    sr: in std_logic_vector (1 downto 0);
    q, qb: inout std_logic);
end srff;
architecture behavioral of srff is
begin
    process (clk)
    begin
        if (rst='1') then
            q<='0';
            qb<='1';
        elsif (rising_edge (clk)) then
            case sr is
            when "00"=>q<='0'; qb<='1';
            when "01"=>q<='1'; qb<='0';
            when "11"=>q<='Z'; qb<='Z';
            when others=>null;
        end case;
    end if;
end process;
end behavioral;
Result:
VERILOG CODE:

module sr_ff(sr, clk, rst, q, qb);
input [1:0]sr;
input rst, clk;
output q, qb;
reg q, qb;
always @ (posedge clk)
begin
    if (rst==1)
        begin
            q=0;
            qb=1;
        end
    else
        case (sr)
            2'b00: begin q=q; qb=qb; end
            2'b01: begin q=0; qb=1; end
            2'b10: begin q=1; qb=0; end
            2'b11: begin q=1'bx; qb=1'bx; end
            default:begin end
        endcase
        end
endmodule
**Block Diagram:**

![D Flip Flop Diagram](image)

**Truth Table:**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst</td>
<td>clk</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>↑</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>1</td>
</tr>
</tbody>
</table>
**Experiment 5b : VHDL Code to Describe D Flipflop**

**VHDL CODE :**

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

entity dff is
  port (clk, rst: in std_logic;
       d: in std_logic;
       q, qb: out std_logic);
end dff;

architecture behavioral of dff is
begin
  process (clk)
  begin
    if rst='1' then
      q<='0';
      qb <='1';
    elsif (rising_edge (clk)) then
      q<=d;
      qb<=not (d);
    end if;
  end process;
end;
Result:
VERILOG CODE:

module d_ff(d, rst, clk, q, qb);
input d;
input rst;
input clk;
output q;
output qb;
reg q, qb;
always@(posedge clk)
begin
    if (rst==1)
        begin
            q=0;
            qb=1;
        end
    else
        begin
            q=d;
            qb=~d;
        end
end
endmodule
Block Diagram:

![Block Diagram of JK Flip Flop](image)

Truth Table:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst</td>
<td>clk</td>
<td>j k</td>
</tr>
<tr>
<td>1</td>
<td>↑</td>
<td>X X</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>0 0</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>0 1</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>1 0</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>1 1</td>
</tr>
</tbody>
</table>
Experiment 5c : HDL Code to Describe JK Flipflop

VHDL CODE:

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

entity jkff is
  port (clk, rst: in bit;
         jk: in bit_vector (1 downto 0);
         q, qb: inout bit);
end jkff;

architecture behavioral of jkff is
begin
  signal div: std_logic_vector(22 downto 0);
  signal clkdiv:std_logic;
  begin
    process(clk) is
    begin
      if rising_edge (clk) then
        div<=div+'1';
      end if;
    end process;
    clkdiv<=div(22);
    process (clkdiv, rst)
    begin
      if (rst='1') then
        q<='0';
        qb<='1';
      elsif (rising_edge (clkdiv)) then
        case (jk) is
        when"00"=>q<=q; qb<=qb;
        when"01"=>q<='0'; qb<='1';
        when"10"=>q<='1'; qb<='0';
        when"11"=>q<=not(q); qb<=not(qb);
        when others=>null;
        end case;
      end if;
    end process;
  end ;
Result:
**VERILOG CODE:**

module jk_ff(j, k, clk, reset, q, qb);
input [1:0] jk;
input clk, rst;
output q, qb;
reg q, qb;
reg [22:0] div;
reg clkdiv;
always @ (posedge clk)
begin
    div = div+1'b1;
    clkdiv = div[22];
end
always @ (posedge clkdiv)
begin
    if(rst==1)
        begin
            q=0;
            qb=1;
        end
else
    case (jk)
        2'b00: begin q=q; qb=qb; end
        2'b01: begin q=0; qb=1; end
        2'b10: begin q=1; qb=0; end
        2'b11: begin q=~(q); qb=~(qb); end
        default: begin end
    endcase
end
endmodule
**Block Diagram:**

![Block Diagram of T Flip Flop](image)

**Truth Table:**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst</td>
<td>clk</td>
<td>q</td>
</tr>
<tr>
<td>1</td>
<td>↑</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>↑</td>
<td>1</td>
</tr>
</tbody>
</table>
**Experiment 5d : HDL Code to Describe T Flipflop**

**VHDL CODE :**

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

entity tff is
port (t: in std_logic;
     clk, rst: in std_logic;
     q, qb: inout std_logic);
end tff;
architecture behavioral of tff is
signal div:std_logic_vector(22 downto 0);
signal clkdiv:std_logic;
begin
    process (clk)
    begin
        if (rising_edge(clk)) then
            div<=div+'1';
        end if;
    end process;
    clkdiv<=div(22);
    process (clkdiv)
    begin
        if (rst='1') then
            q<='0';
            qb<='1';
        elsif (rising_edge(clkdiv)) then
            case (t) is
                when '0'=>q<=q; qb<=qb;
                when '1'=>q<=not(q); qb<=not(qb);
                when others=>null;
            end case;
        end if;
    end process;
end ;
Result:
VERILOG CODE:

module t_ff(t, clk, rst, q, qb);
input t, clk, rst;
output q, qb;
reg q, qb;
always @ (posedge clk)
  begin
    div = div+1'b1;
    clkdiv = div[22];
  end
always @ (posedge clkdiv)
  begin
    if (rst==1)
      begin
        q=0;
        qb=1;
      end
    else
      case ( t)
        1'b0:begin q=q; qb=qb; end
        1'b1:begin q=~(q); qb=~(qb); end
        default: begin end
      endcase
  end
endmodule
**Block diagram:**

![Block diagram of a 4-bit synchronous counter](image)

**Truth table:**

<table>
<thead>
<tr>
<th>Clock</th>
<th>rst</th>
<th>bin_out(3)</th>
<th>bin_out(2)</th>
<th>bin_out(1)</th>
<th>bin_out(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>0</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>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</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>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Experiment 6a: HDL Code to Design 4-bit Binary Synchronous Reset counter

VHDL CODE:

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

entity bin_syn is
  port (clk, rst: in std_logic;
        bin_out: out std_logic_vector(3 downto 0));
end bin_syn;
architecture behavioral of bin_syn is
begin
  signal div:std_logic_vector(22 downto 0);
  signal clkdiv:std_logic;
  process (clk)
  begin
    if (rising_edge(clk)) then
      div<=div+'1';
    end if;
  end process;
  clkdiv<=div(22);
  process (clkdiv)
  variable temp:std_logic_vector(3 downto 0):="0000";
  begin
    if (rising_edge(clkdiv))then
      if (rst='1') then
        temp:=temp+'1';
      end if;
    end if;
  end process;
  bin_out<=temp;
end;
Result:
VERILOG CODE:

module bin_sync( clk, rst, bin_out);
input  clk, rst;
output [3:0] bin_out;
reg [3:0] bin_out;
initial
bin_out=4'b0000;
always @ (posedge clk)
begin
  div = div+1'b1;
  clkdiv = div[22];
end
always @ (posedge clkdiv)
begin
  if (rst==1)
    bin_out=bin_out+4'b0001;
end
endmodule
**Block diagram:**

![Block diagram](image)

**Truth table:**

<table>
<thead>
<tr>
<th>Clock</th>
<th>rst</th>
<th>bin_out(3)</th>
<th>bin_out(2)</th>
<th>bin_out(1)</th>
<th>bin_out(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</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>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Experiment 6b: HDL Code to Design 4-bit Binary Asynchronous Reset Counter

VHDL CODE:

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

entity bin_asyn is
port (clk, rst: in std_logic;
       bin_out: out std_logic_vector(3 downto 0));
end bin_syn;
architecture behavioral of bin_syn is
signal div:std_logic_vector(22 downto 0);
signal clkdiv:std_logic;
signal temp:std_logic_vector(3 downto 0);
begin
    process (clk)
    begin
        if rising_edge(clk) then
            div<=div+'1';
        end if;
    end process;
    clkdiv<=div(22);
    process (clkdiv,rst)
    begin
        if (rst='0') then
            temp<="0000";
        elsif (rising_edge(clkdiv))then
            temp<=temp+'1';
        end if;
    end process;
    bin_out<=temp;
end ;
Result:
VERILOG CODE:

module bin_sync( clk, rst, bin_out);
input  clk, rst;
output [3:0] bin_out;
reg [3:0] bin_out;
always @ (posedge clk)
  begin
    div = div+1'b1;
    clkdiv = div[22];
  end
always @ (posedge( clkdiv))
  begin
    if (rst=0)
      bin_out=4’b0000;
    else
      bin_out=bin_out+4’b0001;
  end
endmodule
**Block diagram:**

![Block diagram of BCD Synchronous Reset 4bit Counter](image)

**Truth table:**

<table>
<thead>
<tr>
<th>Clock</th>
<th>rst</th>
<th>bcd_out(3)</th>
<th>bcd_out(2)</th>
<th>bcd_out(1)</th>
<th>bcd_out(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</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>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Experiment 6c : HDL Code to Design 4-bit BCD Synchronous Reset counter

VHDL CODE:

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

entity bcd_syn is
  port (clk, rst: in std_logic;
        bcd_out: out std_logic_vector(3 downto 0));
end bcd_syn;
architecture behavioral of bcd_syn is
  signal div:std_logic_vector(22 downto 0);
  signal clkdiv:std_logic;
begin
  process (clk)
  begin
    if rising_edge(clk) then
      div<=div+1;
    endif;
  end process;
  clkdiv<=div(22);
  process (clkdiv)
  variable temp:std_logic_vector(3 downto 0);
  begin
    if (rising_edge(clkdiv))then
      if (rst='0' or temp="1001") then
        temp:="0000";
      else
        temp:=temp+'1';
      end if;
    end if;
  end process;
  bcd_out<=temp;
end ;
Result:
VERILOG CODE:

module bin_sync( clk, rst, bcd_out);
input   clk, rst;
output  [3:0] bcd_out;
reg     [3:0] bcd_out;
initial
  begin
    bcd_out=4’d0;
  end
always @ (posedge clk)
  begin
    div = div+1'b1;
    clkdiv = div[22];
  end
always @ (posedge clkdiv)
  begin
    if (rst)
      bcd_out=4’d0;
    else if(bcd_out<4’d9)
      bin_out=bin_out+4’d1;
    else
      bin_out=4’d0;
  end
endmodule
Result:
Experiment 6d: HDL Code to Design 4-bit BCD Asynchronous Reset counter

VHDL CODE:

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library unisim;
use unisim.vcomponents.all;

entity bcd_asyn is
port (clk, rst: in std_logic;
bcd_out: out std_logic_vector(3 downto 0));
end bcd_asyn;
architecture behavioral of bin_syn is
signal div:std_logic_vector(22 downto 0);
signal clkdiv:std_logic;
signal temp:std_logic_vector(3 downto 0);
begin
    process (clk)
    begin
        if rising_edge(clk) then
            div<=div+1;
        endif;
    end process;
    clkdiv<=div(22);
    process (clkdiv,rst)
    begin
        if (rst='0' or temp="1010") then
            temp<="0000";
        elsif (rising_edge(clkdiv))then
            temp<=temp+'1';
        end if;
    end process;
    bcd_out<=temp;
end ;
module bin_sync (clk, rst, bin_out);
input clk, rst;
output [3:0] bin_out;
reg [3:0] bin_out;
reg [22:0] div;
reg clkdiv;
always @ (posedge clk)
    begin
        div = div + 1'b1;
        clkdiv = div[22];
    end
always @ (posedge clkdiv)
begin
    if (rst == 0)
        bcd_out = 4'd0;
    else if (count < 4'd9)
        bcd_out = bcd_out + 4'd1;
end
endmodule
WORKSHEET:
Sequence: 0 5 6 4

State Diagram:

Truth Table:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>q(1) q(0)</td>
<td>qp(1) qp(0)</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
</tbody>
</table>

K-map Simplification:
**Experiment 6e : HDL Code to Design Any Sequence Counter**

**VHDL CODE :**

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

entity seq is
  port ( clk : in  std_logic;
         z : out  std_logic_vector (2 downto 0));
end seq;

architecture behavioral of seq is
begin
  signal clkdiv: std_logic;
  signal div:std_logic_vector(22 downto 0);
  process(clk) is
    begin
      if(rising_edge(clk)) then
        div<= div + '1';
      end if;
  end process;
  clkdiv<=div(22);

  process(clkdiv)
  variable  q,qp: std_logic_vector(1 downto 0);
  begin
    if (rising_edge(clkdiv)) then
      qp(1):= q(1) xor q(0);
      qp(0):= not q(0);
      z(2)<= q(1) or q(0);
      z(1)<= q(1) and (not q(0));
      z(0)<= not q(1) and q(0);
    end if;
    q<=qp ;
  end process;
end;
```


Result:
Work Space:

Write a VERILOG Code to design any sequence generator:
**Procedure:**

1. Make the connection between FRC5 of the FPGA board and the seven-segment connector of the GPIOcard-1.

2. Make the connection between FRC4 of the FPGA board and the keyboard connector of the GPIOcard-1.

3. Assign appropriate pins to input and output.

4. Connect USB PROGRAMMER cable and power supply to the FPGA board.

4. Press the hex keys and analyze the data.
Experiment 1: VHDL Code to display a character on the given seven segment display accepting HEX keypad input

**VHDL CODE:**

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

entity key is
Port (col: inout std_logic_vector(3 downto 0):="0001";
row : in std_logic_vector(3 downto 0);
clk : in std_logic;
disp_sel: out std_logic_vector(3 downto 0);
ss : out std_logic_vector(6 downto 0));
end key;

architecture Behavioral of key is
begin
process(clk) is
begin
if (rising_edge(clk)) then
    col<=col(2 downto 0)& col(3);
end if;
end process;
disp_sel<="1110";
process(col,row)
begin
    case (col) is
    when "0001" => case row is
    when "0001" => ss<= "1111110";
    when "0010" => ss <= "0110011";
    when "0100" => ss <= "1111111";
    when "1000" => ss <= "1001110";
    when others=> ss <= "0000000";
end case;
end case;
end process;
Result:
when "0010" => case (row) is
  when "0001" => ss <= "0110000";
  when "0010" => ss <= "1011011";
  when "0100" => ss <= "1110111";
  when "1000" => ss <= "0111101";
  when others => ss <= "0000000";
end case;

when "0100" => case (row) is
  when "0001" => ss <= "1101101";
  when "0010" => ss <= "1011111";
  when "0100" => ss <= "1110111";
  when "1000" => ss <= "1001111";
  when others => ss <= "0000000";
end case;

when "1000" => case (row) is
  when "0001" => ss <= "1111001";
  when "0010" => ss <= "1110000";
  when "0100" => ss <= "0011111";
  when "1000" => ss <= "1000111";
  when others => ss <= "0000000";
end case;
when others => null;
end case;
end process;
end Behavioral;
Hardware Details:

1. When rly=0, pwm(o)=1, pwm(1)=1

2. When rly=1, pwm(o)=1, pwm(1)=1
**Experiment 2a : VHDL Code to Control speed and direction of DC Motor**

**VHDL CODE :**

```vhdl
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity dc_motor is
  port(clk,rst:in std_logic;
       rly:out std_logic;
       pwm:out std_logic_vector(1 downto 0);
       keys:in std_logic_vector(3 downto 0));
end dc_motor;

architecture behavioral of dc_motor is
  signal div:std_logic_vector(16 downto 0);
  signal clkdiv:std_logic;
  signal counter:std_logic_vector(7 downto 0);
  signal dc:integer range 0 to 255;
  signal tick:std_logic;
begin
  process(clk) is
  begin
    if (rising_edge (clk)) then
      div<=div+1;
    end if;
  end process;
  clkdiv<=div(12);
  tick<=keys(0) and keys(1)and keys(2) and keys(3);
  process(tick) is
  begin
    if (falling_edge (tick)) then
      case (keys) is
      when "1110"=>dc<=255;
      when "1101"=>dc<=128;
      when "1011"=>dc<=100;
      when "0111"=>dc<=64;
      when others=>dc<=64;
      end case;
    end if;
  end process;
end behavioral;
```


3. When rly=0,pwm(o)=1,pwm(1)=0

4. When rly=1,pwm(o)=1,pwm(1)=0

**Procedure:**

1. Make the connection between FRC9 of the fpga board and the DC motor connector of the GPIOcard-2.
2. Make the connection between FRC7 of the fpga board and the keys connector of the GPIOcard-2.
3. Make the connection between FRC1 of the fpga board and the DIP switch connector of the GPIOcard-2.
4. Assign appropriate pins to input and output.
5. Connect USB PROGRAMMER cable and power supply to the fpga board.
6. Press the keys and vary the speed of the DC motor.
end if;
end process;
process (clkdiv,rst) is
begin
if( rst='1') then
counter<="00000000";
pwm<="01";
elsif (rising_edge(clkdiv)) then
counter<=counter+1;
if (counter >=dc)then
pwm(1)<='0';
else
pwm(1)<='1';
end if;
end if;
end process;
end Behavioral;
end Behavioral;
**Hardware Details:**

**Procedure:**
1. Make the connection between FRC9 of the fpga board and stepper motor connector of interfacing card.

2. Make the connection between FRC1 of the fpga board and DIP switch connector of the GPIOcard-2.

3. Assign appropriate pins to input and output.

4. Connect USB PROGRAMMER cable and power supply to the fpga board.
**Experiment 2b : VHDL Code to Control direction of Stepper Motor**

**VHDL CODE :**

```vhdl
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity stepper is
  port ( dout : out std_logic_vector(3 downto 0);
         clk,reset,dir : in std_logic);
end stepper;

architecture behavioral of stepper is
  signal div : std_logic_vector(20 downto 0);
  signal clkdiv: std_logic;
  signal shift_reg : std_logic_vector(3 downto 0);
begin
  process(clk)
  begin
    if (rising_edge (clk)) then
      div <= div + '1';
    end if;
  end process;
  clkdiv <=div(16);
  process(reset,clkdiv)
  begin
    if (reset='0') then
      shift_reg <= "0001";
    elsif (rising_edge(clkdiv)) then
      if (dir='1')then
        shift_reg <= shift_reg(0) & shift_reg(3 downto 1);
      else
        shift_reg<=shift_reg(2 downto 0)& shift_reg(3);
      end if;
    end if;
  end process;
  dout <= shift_reg;
end Behavioral;
```

**Design:**

Given frequency $[f_m] = 2\text{KHz}$

System frequency $[f_s] = 4\text{MHz}$

Time $[T] = 1/f_m = 0.5\text{ms}$

Duty cycle $[DC] = 50\%$

- Count $\times 1/fc = T$
  
  \[256 \times 1/fc = 0.5\text{ms}\]

- $fc = 512\text{KHz}$

- $2^N = f_s/fc$
  
  \[4\text{MHz}/512\text{KHz} = 8\]

- $N = 3$

- Counter value $= DC \times \text{Count}$
  
  \[0.50 \times 256 = 128\]

- Divide by $N$ counter

- 8-bit Counter

- 4MHz $\rightarrow$ Divide by $N$ counter $\rightarrow$ fc $\rightarrow$ 8-bit Counter $\rightarrow$ 0 to 255

- fc $\rightarrow$ 0 to 255

- f $\rightarrow$ 0 to 127
Experiment 3a: VHDL Code to generate Square wave using DAC

Procedure:
1. Make the connection between FRC5 of the fpga board and DAC connector of GPIOcard-2
2. Make the connection between FRC1 of the fpga board and DIP switch connector of the GPIOcard-2.
3. Assign appropriate pins to input and output.
4. Connect USB PROGRAMMER cable and power supply to the fpga board.

VHDL CODE:

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

entity sqwave is
port(clk,rst:in std_logic;
dac_out:out std_logic_vector(7 downto 0));
end sqwave;

architecture Behavioral of sqwave is
signal counter : std_logic_vector(7 downto 0);
signal div: std_logic_vector(3 downto 0);
signal clkdiv:std_logic;
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      div <= div + '1';
    end if;
  end process;
  clkdiv<=div(3);
  process(clkdiv)
  begin
    if (rst='1') then
      counter <= "00000000";
      elsif (rising_edge(clkdiv)) then
```

Result:
counter <= counter + 1 ;
end if;
end process;

process(counter) is
begin
if(counter<=128)then
dac_out<="11111111";
else
dac_out<="00000000";
end if;
end process;
end Behavioral;
Result:

**Procedure:**
1. Make the connection between FRC5 of the fpga board and DAC connector of GPIOcard-2
2. Make the connection between FRC1 of the fpga board and DIP switch connector of the GPIOcard-2.
3. Assign appropriate pins to input and output.
4. Connect USB PROGRAMMER cable and power supply to the fpga board.
Experiment 3b: VHDL Code to generate Triangular wave using DAC

VHDL CODE:

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

entity triwave is
  port(clk,rst:in std_logic;
       dac_out:out std_logic_vector(7 downto 0));
end triwave;

architecture Behavioral of triwave is
  signal counter : std_logic_vector(8 downto 0);
  signal div: std_logic_vector(3 downto 0);
  signal clkdiv:std_logic;
  begin
    process(clk)
      begin
        if rising_edge(clk) then
          div <= div + '1';
        end if;
      end process;
    clkdiv<=div(2);
    process(clkdiv)
      begin
        if (rst='1') then
          counter <= "00000000";
        elsif rising_edge(clkdiv) then
          counter <= counter + ‘1’;
        end if;
      end process;
    process(counter) is
      begin
        if(counter(8)=’1’)then
          dac_out<= not (counter (7 downto 0));
        else
          dac_out<=counter (7 downto 0);
        end if;
      end process;
  end Behavioral;
```
Result:
**Experiment 3e: VHDL Code to generate Ramp wave using DAC**

**Procedure:**

1. Make the connection between FRC5 of the fpga board and DAC connector of GPIOcard-2
2. Make the connection between FRC1 of the fpga board and DIP switch connector of the GPIOcard-2.
3. Assign appropriate pins to input and output.
4. Connect USB PROGRAMMER cable and power supply to the fpga board.

**VHDL CODE:**

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

entity sawwave is
  port(clk,rst:in std_logic;
       dac_out:out std_logic_vector(7 downto 0));
end sawwave;

architecture Behavioral of sawwave is
  signal counter : std_logic_vector(7 downto 0);
  signal div: std_logic_vector(3 downto 0);
  signal clkdiv:std_logic;
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      div <= div + '1' ;
    end if;
  end process;
  clkdiv<=div(3);
  process(clkdiv)
  begin
    if rst='1' then
      counter <= "00000000";
    elsif (rising_edge(clkdiv)) then
      counter <= counter + 1 ;
    end if;
  end process;
  dac_out<=counter;
end Behavioral;
```

```
Result:
Experiment 4: VHDL Code to control external lights using Relays

Procedure:

1. Make the connection between FRC9 of the fpga board and the external light connector of GPIO card-2.

2. Make the connection between FRC1 of the fpga board and DIP switch connector of the GPIO card-2.

3. Assign appropriate pins to input and output.

4. Connect USB PROGRAMMER cable and power supply to the fpga board.

VHDL CODE:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library unisim;
use unisim.vcomponents.all;

entity light is
    Port ( cntrl1, cntrl2 : in STD_LOGIC;
           led : out STD_LOGIC); 
end light;

architecture Behavioral of light is
begin 
    led<=cntrl1 or ctrl2; 
end Behavioral;
```
**Experiment 5a: VHDL code to accept 8 channel analog signal, temperature sensors and display the data on LCD panel or seven segment display.**

**Procedure:**

1. Make the connection between frc5 of the fpga board to the LCD display connector of the vtu card1.
2. Make the connection between frc10 of the fpga board to the adc connector of the vtu card1.
3. Make the connection between frc6 of the fpga board to the dip switch connector of the vtu card1.
4. Short the jumper j1 to the vin to get the analog signal.
5. Press the hex keys and analyze the data.

**VHDL CODE:**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use work.lcd_grap.all;
entity adc_lcd is
  port ( clk: in std_logic; -- 4 mhz clock
           reset: in std_logic; -- master reset pin
           intr: in std_logic;
           adc_out: in std_logic_vector(7 downto 0);
           cs, rd, wr: out std_logic;
           lcd_rw : out std_logic;
           lcd_select : out std_logic;
           lcd_enable : out std_logic;
           lcd_data: out std_logic_vector (7 downto 0)); -- gives registered data output
end adc_lcd;
architecture adc_beh of adc_lcd is
  type state_type is (initial, display, clear, location, putchar);
  signal state, next_state: state_type;
  -- clear screen.
  constant clr: std_logic_vector(7 downto 0) := "00000001";
  -- display on, without cursor.
  constant don: std_logic_vector(7 downto 0) := "00001100";
  -- function set for 8-bit data transfer and 2-line display.
  constant set: std_logic_vector(7 downto 0) := "00111000";
```

```vhdl
define the rest of the VHDL code here...```
--frequency divider
signal counter : std_logic_vector(18 downto 0);
signal clk_div : std_logic;
constant big_delay: integer :=16;
constant small_delay: integer :=2;
constant reg_setup: integer :=1;
signal digital_data1,data1,data2: std_logic_vector(7 downto 0);
signal digital_data : integer range 0 to 255;
signal ntr : std_logic;
begin
ibuf_inst : ibuf
-- edit the following generic to specify the i/o standard for this port.
generic map ( 
iostandard => "lvcmos25")
port map ( 
o => ntr, -- buffer output
i => intr -- buffer input (connect directly to top-level port));
process(clk)
begin
if clk='1' and clk'event then
  counter<=counter+'1';
end if;
end process;
clk_div<=counter(7);
cs <='0';
wr <=ntr;
digital_data1 <= adc_out ;
rd <='0';
digital_data<=conv_integer(digital_data1) ;
process(digital_data)
begin
case (digital_data) is
when 0 to 100 => data1 <= one ; data2 <= nine ;
when 101 to 110 => data1 <= two ; data2 <= zero ;
when 111 to 120 => data1 <= two ; data2 <= one ;
when 121 to 130 => data1 <= two ; data2 <= two ;
when 131 to 140 => data1 <= two ; data2 <= three ;
when 141 to 150 => data1 <= two ; data2 <= four ;
when 151 to 160 => data1 <= two ; data2 <= five ;
when 161 to 170 => data1 <= two ; data2 <= six ;
when 171 to 180 => data1 <= two ; data2 <= seven ;
when 181 to 190 => data1 <= two ; data2 <= eight ;
when 191 to 200 => data1 <= two ; data2 <= nine ;
when 201 to 205 => data1 <= three; data2 <= zero;
when 206 to 210 => data1 <= three; data2 <= one;
when 211 to 215 => data1 <= three; data2 <= two;
when 216 to 220 => data1 <= three; data2 <= three;
when 221 to 225 => data1 <= three; data2 <= four;
when 226 to 230 => data1 <= three; data2 <= five;
when 231 to 235 => data1 <= three; data2 <= six;
when 236 to 240 => data1 <= three; data2 <= seven;
when 241 to 245 => data1 <= three; data2 <= eight;
when 246 to 250 => data1 <= three; data2 <= nine;
when others => data1 <= four; data2 <= zero;
end case;
end process;
lcd_rw<='0';
process (clk_div,reset)
variable count: integer range 0 to big_delay;
variable c1 : std_logic_vector(7 downto 0);
begin
if reset = '1' then
  state<=initial;
  count:=0;
  lcd_enable<='0';
  lcd_select<='0';
c1 := "01111111";
elsif clk_div'event and clk_div = '1' then
  case state is
    when initial => -- to set the function
      if count=reg_setup then
        lcd_enable<='1';
      else
        lcd_enable<='0';
      end if;
lcd_data<=set;
lcd_select<='0';
      if count=small_delay then
        state<=display;
        count:=0;
      else
        count:=count+1;
      end if;
    when display => -- to set display on
      if count=reg_setup then
        lcd_enable<='1';
else
lcd_enable<='0';
end if;
lcd_data<=don;
lcd_select<='0';
if count=small_delay then
state<=clear;
count:=0;
else
count:=count+1;
end if;
when clear => -- clear the screen
if count=reg_setup then
lcd_enable<='1';
else
lcd_enable<='0';
end if;
lcd_data<=clr;
lcd_select<='0';
if count=big_delay then
state<=location;
count:=0;
else
count:=count+1;
end if;
when location => -- clear the screen
if count=reg_setup then
lcd_enable<='1';
else
lcd_enable<='0';
end if;
if count=0 then
if c1="10001111" then
c1:="10000000";
else
c1:=c1+'1';
end if;
end if;
lcd_data <= c1 ;
lcd_select<='0';
if count=big_delay then
state<=putchar;
count:=0; else
count:=count+1;
end if;
when putchar=> -- display the character on the lcd
if count=reg_setup then
    lcd_enable<='1';
else
    lcd_enable<='0';
end if;
case c1 is
when "10000000" => lcd_data<= a ;
when "10000001" => lcd_data<= d ;
when "10000010" => lcd_data<= c ;
when "10000011" => lcd_data<= space ;
when "10000100" => lcd_data<= v ;
when "10000101" => lcd_data<= o ;
when "10000110" => lcd_data<= l ;
when "10000111" => lcd_data<= t ;
when "10001000" => lcd_data<= a ;
when "10001001" => lcd_data<= g ;
when "10001010" => lcd_data<= e ;
when "10001011" => lcd_data<= space ;
when "10001100" => lcd_data<= equal ;
when "10001101" => lcd_data<= data1 ;
when "10001110" => lcd_data<= dot ;
when "10001111" => lcd_data<= data2 ;
when "11000000" => lcd_data<= space ;
when "11000001" => lcd_data<= space ;
when "11000010" => lcd_data<= space ;
when "11000011" => lcd_data<= space ;
when "11000100" => lcd_data<= space ;
when "11000101" => lcd_data<= space ;
when "11000110" => lcd_data<= space ;
when "11000111" => lcd_data<= space ;
when "11001000" => lcd_data<= space ;
when "11001001" => lcd_data<= space ;
when "11001010" => lcd_data<= space ;
when "11001011" => lcd_data<= space ;
when "11001100" => lcd_data<= space ;
when "11001101" => lcd_data<= space ;
when "11001110" => lcd_data<= space ;
when "11001111" => lcd_data<= space ;
when others => null;
end case ;
lcd_select<='1';
if count=small_delay then
  state<=location;
  count:=0;
else
  count:=count+1;
end if;
end case;
end if;
end process;
end adc_beh;
Experiment 5b: VHDL code to accept 8 channel analog signal, temperature sensors and display the data on lcd panel or seven segment display.

Procedure:
1. Make the connection between frc5 of the fpga board to the lcd display connector of the vtu card1.
2. Make the connection between frc10 of the fpga board to the adc connector of the vtu card1.
3. Make the connection between frc6 of the fpga board to the dip switch connector of the vtu card1.
4. Short the jumper j1 to the tc to sense the temperature.
5. press the hex keys and analyze the data.

VHDL CODE:

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.lcd_grap.all;
library unisim;
use unisim.vcomponents.all;
entity temp_lcd is
port ( clk: in std_logic; -- 4 mhz clock
      reset: in std_logic; -- master reset pin
      intr: in std_logic;
      adc_out: in std_logic_vector(7 downto 0);
      cs,rd,wr:out std_logic;
      lcd_rw : out std_logic;
      lcd_select : out std_logic;
      lcd_enable : out std_logic;
      lcd_data: out std_logic_vector (7 downto 0)); -- gives registered data output
end temp_lcd;
architecture temp_beh of temp_lcd is
type state_type is (initial,display,clear,location,putchar);
signal state,next_state: state_type;
-- clear screen.
constant clr: std_logic_vector(7 downto 0) := "00000001";
-- display on, without cursor.
constant don: std_logic_vector(7 downto 0) := "00001100";
-- function set for 8-bit data transfer and 2-line display
constant set: std_logic_vector(7 downto 0) := "00111000";
--frequency divider
signal counter : std_logic_vector(18 downto 0);
signal clk_div : std_logic;
--signal c: std_logic_vector(7 downto 0);
constant big_delay: integer := 16;
constant small_delay: integer := 2;
constant reg_setup: integer := 1;
signal digital_data1, data1, data2: std_logic_vector(7 downto 0);
signal digital_data: integer range 0 to 255;
signal ntr: std_logic;
begi
ibuf_inst : ibuf
-- edit the following generic to specify the i/o standard for this port.
generic map(
iostandard => "lvcmos25")
port map(
o => ntr, -- buffer output
i => intr -- buffer input (connect directly to top-level port));
process(clk)
begi
if clk='1' and clk'event then
counter<=counter+'1';
end if;
end process;
clk_div<=counter(7);
cs <='0';
wr <= ntr;
digital_data1 <= adc_out ;
rd <= '0';
digital_data<=conv_integer(digital_data1);
process(digital_data)
begi
case (digital_data) is
when 0 to 60 => data1 <= zero ; data2 <= nine ;
when 61 to 65 => data1 <= one ; data2 <= zero ;
when 66 to 70 => data1 <= one ; data2 <= one ;
when 71 to 75 => data1 <= one ; data2 <= two ;
when 76 to 80 => data1 <= one ; data2 <= three ;
when 81 to 85 => data1 <= one ; data2 <= four ;
when 86 to 90 => data1 <= one ; data2 <= five ;
when 91 to 95 => data1 <= one ; data2 <= six ;
when 96 to 100 => data1 <= one ; data2 <= seven ;
when 101 to 105 => data1 <= one; data2 <= eight;
when 106 to 110 => data1 <= one; data2 <= nine;
when 111 to 115 => data1 <= two; data2 <= zero;
when 116 to 120 => data1 <= two; data2 <= one;
when 121 to 125 => data1 <= two; data2 <= two;
when 126 to 130 => data1 <= two; data2 <= three;
when 131 to 135 => data1 <= two; data2 <= four;
when 136 to 140 => data1 <= two; data2 <= five;
when 141 to 145 => data1 <= two; data2 <= six;
when 146 to 150 => data1 <= two; data2 <= seven;
when 151 to 155 => data1 <= two; data2 <= eight;
when 156 to 160 => data1 <= two; data2 <= nine;
when 161 to 165 => data1 <= three; data2 <= zero;
when 166 to 170 => data1 <= three; data2 <= two;
when 171 to 175 => data1 <= three; data2 <= three;
when 176 to 180 => data1 <= three; data2 <= four;
when 181 to 185 => data1 <= three; data2 <= five;
when 186 to 190 => data1 <= three; data2 <= six;
when 191 to 195 => data1 <= three; data2 <= seven;
when 196 to 200 => data1 <= three; data2 <= eight;
when 201 to 205 => data1 <= three; data2 <= nine;
when 206 to 210 => data1 <= four; data2 <= zero;
when 211 to 214 => data1 <= four; data2 <= one;
when 215 to 219 => data1 <= four; data2 <= two;
when 220 to 224 => data1 <= four; data2 <= three;
when 225 to 229 => data1 <= four; data2 <= four;
when 230 to 234 => data1 <= four; data2 <= five;
when 235 to 239 => data1 <= four; data2 <= six;
when 240 to 244 => data1 <= four; data2 <= seven;
when 245 to 249 => data1 <= four; data2 <= eight;
when 250 to 255 => data1 <= four; data2 <= nine;
when others => data1 <= five; data2 <= zero;
end case;
end process;
lcd_rw<='0';
process (clk_div,reset)
variable count: integer range 0 to big_delay;
variable c1 : std_logic_vector(7 downto 0);
begbegin
if reset = '1' then
state<=initial;
count:=0;
lcd_enable<='0';
lcd_select<='0';
c1 := "01111111";
elif clk_div'event and clk_div = '1' then
  case state is
  when initial => -- to set the function
    if count=reg_setup then
      lcd_enable<='1';
    else
      lcd_enable<='0';
    end if;
    lcd_data<=set;
    lcd_select<='0';
    if count=small_delay then
      state<=display;
      count:=0;
    else
      count:=count+1;
    end if;
  when display => -- to set display on
    if count=reg_setup then
      lcd_enable<='1';
    else
      lcd_enable<='0';
    end if;
    lcd_data<=don;
    lcd_select<='0';
    if count=small_delay then
      state<=clear;
      count:=0;
    else
      count:=count+1;
    end if;
  when clear => -- clear the screen
    if count=reg_setup then
      lcd_enable<='1';
    else
      lcd_enable<='0';
    end if;
    lcd_data<=clr;
    lcd_select<='0';
    if count=big_delay then
      state<=location;
      count:=0;
else
count:=count+1;
end if;
when location => -- clear the screen
if count=reg_setup then
lcd_enable<='1';
else
lcd_enable<='0';
end if;
if count=0 then
if c1="10001111" then
  c1:="10000000";
else
  c1:=c1+'1';
end if;
end if;
lcd_data <= c1;
lcd_select<='0';
if count=big_delay then
state<=putchar;
count:=0;
else
count:=count+1;
end if;
when putchar=> -- display the character on the lcd
if count=reg_setup then
lcd_enable<='1';
else
lcd_enable<='0';
end if;
case c1 is
when "10000000" => lcd_data<= t ;
when "10000001" => lcd_data<= e ;
when "10000010" => lcd_data<= m ;
when "10000011" => lcd_data<= p ;
when "10000100" => lcd_data<= e ;
when "10000101" => lcd_data<= r ;
when "10000110" => lcd_data<= a ;
when "10000111" => lcd_data<= t ;
when "10001000" => lcd_data<= u ;
when "10001001" => lcd_data<= r ;
when "10001010" => lcd_data<= e ;
when "10001011" => lcd_data<= space ;
when "10001100" => lcd_data<= equal ;
when "10001101" => lcd_data<= data1 ;
when "10001110" => lcd_data<= data2 ;
when "10001111" => lcd_data<= c ;
when "11000000" => lcd_data<= space ;
when "11000001" => lcd_data<= space ;
when "11000010" => lcd_data<= space ;
when "11000011" => lcd_data<= space ;
when "11000100" => lcd_data<= space ;
when "11000101" => lcd_data<= space ;
when "11000110" => lcd_data<= space ;
when "11000111" => lcd_data<= space ;
when "11001000" => lcd_data<= space ;
when "11001001" => lcd_data<= space ;
when "11001010" => lcd_data<= space ;
when "11001011" => lcd_data<= space ;
when "11001100" => lcd_data<= space ;
when "11001101" => lcd_data<= space ;
when "11001110" => lcd_data<= space ;
when "11001111" => lcd_data<= space ;
when others => null;
end case;
lcd_select<='1';
if count=small_delay then
  state<=location;
  count:=0;
else
  count:=count+1;
end if;
end case;
end if;
end process;
end temp_beh;
**Experiment 6: VHDL code to simulate elevator operations.**

**Procedure:**

1. Make the connection between frc5 of the fpga board to the lcd display connector of the vtu card1.
2. Make the connection between frc1 of the fpga board to the keyboard connector of the vtu card1.
3. Make the connection between frc6 of the fpga board to the dip switch connector of the vtu card1.
4. Connect the downloading cable and power supply to the fpga board.
5. Press the hex keys and analyze the data.

**VHDL CODE:**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.lcd_grap.all;
entity elevator is
  generic(bits : integer := 8 ); -- number of bits used for duty cycle.
  -- also determines pwm period.
  port ( clk: in std_logic; -- 4 mhz clock
         reset,en: in std_logic; -- master reset pin
         lcd_rw : out std_logic;
         pwm : out std_logic_vector(1 downto 0);
         lcd_select : out std_logic;
         lcd_enable : out std_logic;
         row: in std_logic_vector(0 to 3); -- this are the row lines
         lcd_data: out std_logic_vector (7 downto 0); -- gives registered data output
         col: inout std_logic_vector(0 to 3));
end elevator;
architecture rtl of elevator is
  signal counter : std_logic_vector(bits - 1 downto 0):="00000000";
  type keypad_state_type is (wait_r_0, c3, c2, c1, c0, found, sample, wait_r_1);
  -- state names
  type state_type is (initial,display,clear,location,putchar);
  signal state,next_state: state_type;
  -- clear screen.
  constant clr: std_logic_vector(7 downto 0) := "00000000";
  -- display on, without cursor.
  constant don: std_logic_vector(7 downto 0) := "00001100";
  -- function set for 8-bit data transfer and 2-line display
```
constant set: std_logic_vector(7 downto 0) := "00111000";
--frequency divider
constant big_delay: integer :=16;
constant small_delay: integer :=2;
constant reg_setup: integer :=1;
signal cs, ns: keypad_state_type; -- signals for current and next states
signal duty_cycle,duty_cycle1 : std_logic_vector(bits - 1 downto 0);
signal div_reg: std_logic_vector (22 downto 0); -- clock divide register
signal dclk,ddclk: std_logic; -- this has the divided clock.
signal col_reg_value: std_logic_vector (0 to 3);
signal r1,clk_d,start,stop: std_logic; -- row detection signal
signal key_value1,floor,key_value: integer range 0 to 15;
signal data,data1,floor_num: std_logic_vector (7 downto 0);
signal temp1,temp2,temp3,temp4: std_logic_vector (7 downto 0);
signal temp5,temp6,temp7,temp8: std_logic_vector (7 downto 0);
begin
  --clk_out <= dclk;
  r1 <= row(3) or row(2) or row(1) or row(0);
 ----------------------------- begining of fsm1 (keypad scanner) ----------------------
sync_proc: process (dclk, reset, key_value) -- this is the synchronous part
begin
  if (reset = '0') then -- you must have a reset for fsm to synthesize properly
    cs <= wait_r_0;
  elsif (dclk'event and dclk = '1') then
    cs <= ns;
  end if; end process;
  comb_proc: process (cs, r1, col_reg_value) -- this is the combinational part
begin
  case cs is
    when wait_r_0 => -- waits till a button is pressed
      col <= "1111"; -- keep all columns activated
      if r1 = '1' then -- a button was pressed. but which one?
        ns <= c3; -- let's find out
      else
        ns <= wait_r_0;
      end if;
    when c3 => --
      col <= "0001"; -- activate column 3
      if r1 = '0' then -- this means button was not in column 3
        ns <= c2; -- so check if it was in column 2
      else
        ns <= found; -- button was in column 3
      end if;
  end case;
end;
when c2 => --
col <= "0010"; -- activate column 2
if r1 = '0' then -- this means button was not in column 2
ns <= c1; -- so check if it was in column 1
else
ns <= found; -- button was in column 2
end if;
----------------------------------------------------------------------------------------------
when c1 => --
col <= "0100"; -- activate column 1
if r1 = '0' then -- this means button was not in column 1
ns <= c0; -- so check if it was in column 0
else
ns <= found; -- button was in column 1
end if;
----------------------------------------------------------------------------------------------
when c0 => --
col <= "1000"; -- activate column 0
if r1 = '0' then -- this means button was not in column 0 ??
ns <= wait_r_0; -- so the button must have been depressed fast
else
ns <= found; -- button was in column 3
end if;
----------------------------------------------------------------------------------------------
when found => --
col <= col_reg_value;
if r1 = '0' then -- this means button is depressed
ns <= wait_r_0; -- so go back to initial state
else
ns <= sample; -- otherwise write the key value to data register
end if;
----------------------------------------------------------------------------------------------
when sample => -- this state will generate a signal with one clock period for sampling
col <= col_reg_value;
ns <= wait_r_1; -- otherwise wait till button is pressed
----------------------------------------------------------------------------------------------
when wait_r_1 => --
col <= col_reg_value;
if r1 = '0' then -- this means button was depressed
ns <= wait_r_0; -- so go back to initial state
else
ns <= wait_r_1; -- otherwise wait till button is pressed
end if;
----------------------------------------------------------------------------------------------
end case;
end process;
----------------------------------------------------------------------------------------------
write_data: process (dclk, cs, key_value) -- write valid data to register
begin
if dclk'event and dclk = '0' then -- on the falling edge
if cs = found then
key_value <= key_value1;
end if;
end if;
end process; -- write_data
----------------------------------------------------------------------------------------------
col_reg: process (dclk, cs, col) -- this is the column value register
begin
if (dclk'event and dclk = '0') then -- register the col value on the falling edge
if (cs = c3 or cs = c2 or cs = c1 or cs = c0) then -- provided we're in states c3 thru c0 only
col_reg_value <= col; -- otherwise the column value is not valid
end if;
end if;
end process; -- col_reg
----------------------------------------------------------------------------------------------
decoder: process(row, col_reg_value) -- decodes binary value of pressed key from row and
column
variable code: std_logic_vector (0 to 7);
begin
code := (row & col_reg_value);
case code is
-- col
-- row 0 0123
when "00010001" => key_value1 <= 0;
when "00010010" => key_value1 <= 1;
when "00010100" => key_value1 <= 2;
when "00011000" => key_value1 <= 3;
-- row 1
when "00100001" => key_value1 <= 4;
when "00100010" => key_value1 <= 5;
when "00100100" => key_value1 <= 6;
when "00101000" => key_value1 <= 7;
when "01000001" => key_value1 <= 8;
when "01000010" => key_value1 <= 9;
when "01000100" => key_value1 <= 10;
when "01001000" => key_value1 <= 11;
when "10000001" => key_value1 <= 12;
when "10000010" => key_value1 <= 13;
when "10000100" => key_value1 <= 14;
when "10001000" => key_value1 <= 15;
when others => key_value1 <= 0;
end case;
end process; -- decoder

-- row 3

-- select the appropriate lines for setting frequency

clock_div: process (clk, div_reg) -- clock divider
begin
if (clk'event and clk='1') then
  div_reg <= div_reg + 1;
end if;
end process;
dclk <= div_reg(8);
ddclk<=div_reg(10);
clk_d<=div_reg(22);

-- end of clock divider

-- end of fsm1 (keypad scanner)

-- select the appropriate lines for setting frequency

process (clk, div_reg) -- clock divider
begin
if (clk'event and clk='1') then
  div_reg <= div_reg + 1;
end if;
end process;
dclk <= div_reg(8);
ddclk<=div_reg(10);
clk_d<=div_reg(22);

-- end of clock divider
lcd_data<=set;
lcd_select<='0';
if count=small_delay then
  state<=display;
  count:=0;
else count:=count+1;
end if;
when display => -- to set display on
if count=reg_setup then
  lcd_enable<='1';
else lcd_enable<='0';
end if;
lcd_data<=don;
lcd_select<='0';
if count=small_delay then
  state<=clear;
  count:=0;
else count:=count+1;
end if;
when clear => -- clear the screen
if count=reg_setup then
  lcd_enable<='1';
else lcd_enable<='0';
end if;
lcd_data<=clr;
lcd_select<='0';
if count=big_delay then
  state<=location;
  count:=0;
else count:=count+1;
end if;
when location => -- clear the screen
if count=reg_setup then
  lcd_enable<='1';
else lcd_enable<='0';
end if;
if count=0 then
  if c1="10001111" then
    c1:="11000000";
  elsif c1="11001111" then
    c1:="10000000";
  else c1:=c1+'1';
  end if;
end if;
end if;
lcd_data <= c1;
lcd_select<='0';
if count=big_delay then
state<='putchar';
count:=0;
else count:=count+1;
end if;
when putchar=> -- display the character on the lcd
if count=reg_setup then
lcd_enable<='1';
else lcd_enable<='0';
end if;
case c1 is
when "10000000" => lcd_data<= f ;--single line
when "10000001" => lcd_data<= l ;--single line
when "10000010" => lcd_data<= o ;--single line
when "10000011" => lcd_data<= o ;--single line
when "10000100" => lcd_data<= r ;--single line
when "10000101" => lcd_data<= space ;--single line
when "10000110" => lcd_data<= n ;--single line
when "10000111" => lcd_data<= u ;--single line
when "10001000" => lcd_data<= m ;
when "10001001" => lcd_data<= b ;
when "10001010" => lcd_data<= e ;
when "10001011" => lcd_data<= r ;
when "10001100" => lcd_data<= space ;
when "10001101" => lcd_data<= equal ;
when "10001110" => lcd_data<= floor_num ;
when "10001111" => lcd_data<= space ;
when "11000000" => lcd_data<= s ;--single line
when "11000001" => lcd_data<= t ;--single line
when "11000010" => lcd_data<= a ;--single line
when "11000011" => lcd_data<= t ;--single line
when "11000100" => lcd_data<= u ;--single line
when "11000101" => lcd_data<= s ;--single line
when "11000110" => lcd_data<= space ;--single line
when "11000111" => lcd_data<= temp1 ;--single line
when "11001000" => lcd_data<= temp2 ;
when "11001001" => lcd_data<= temp3 ;
when "11001010" => lcd_data<= temp4 ;
when "11001011" => lcd_data<= space ;
when "11001100" => lcd_data<= temp5 ;
when "11001101" => lcd_data<= temp6 ;
when "11001110" => lcd_data<= temp7 ;
when "11001111" => lcd_data<= temp8 ;
when others => null;
end case ;
lcd_select<='1';
if count=small_delay then
  state<=location;
  count:=0;
else count:=count+1;
end if; end case; end if; end process;
process(clk_d,reset)
variable cou : std_logic_vector(1 downto 0);
variable start,stop: std_logic; -- row detection signal
begin
if reset='0' then
  cou:="00";
  temp1 <= l ;
  temp2 <= i ;
  temp3 <= f ;
  temp4 <= t ;
  temp5 <= i ;
  temp6 <= d ;
  temp7 <= l ;
  temp8 <= e ;
  floor_num <= zero ;
elsif rising_edge(clk_d) then
  case key_value is
  when 0 => floor_num <= zero ;
  floor <=0;
  when 1 => floor_num <= one ;
  floor <=1;
  when 2 => floor_num <= two ;
  floor <=2;
  when 3 => floor_num <= three ;
  floor <=3;
  when 4 =>
  temp1 <= d ;
  temp2 <= o ;
  temp3 <= o ;
  temp4 <= r ;
  temp5 <= o ;
  temp6 <= p ;
temp7 <= e;
temp8 <= n;
when 5 =>
temp1 <= d;
temp2 <= o;
temp3 <= o;
temp4 <= r;
temp5 <= c;
temp6 <= l;
temp7 <= o;
temp8 <= s;
when 6 =>
start:='1';
stop:='0';
when 7 =>
stop:='1';
start:='0';
when others =>
temp1 <= i;
temp2 <= d;
temp3 <= l;
temp4 <= e;
temp5 <= k;
temp6 <= e;
temp7 <= y;
temp8 <= space;
end case;
if start='1' then
if cou=floor then
start := '0';
cou:=cou;
temp7 <= "001100" & cou;
elsif cou<=floor then
cou:=cou + '1';
temp1 <= u;
temp2 <= p;
temp3 <= space;
temp4 <= space;
temp5 <= space;
temp6 <= "01111110";
temp7 <= "001100" & cou;
temp8 <= space;
elsif cou>=floor then
cou:=cou-'1';
temp1 <= d  ;
temp2 <= o  ;
temp3 <= w  ;
temp4 <= n  ;
temp5 <= space  ;
temp6 <= "01111111"  ;
temp7 <= "001100" & cou  ;
temp8 <= space  ;
end if; end if; end if;
end process;
end rtl;
EXTRA EXPERIMENT

Experiment: VHDL code to realize 4-bit Braun multiplier

entity braun_multiplier4 is
  port ( a : in  std_logic_vector (3 downto 0);
        b : in  std_logic_vector (3 downto 0);
        p : out  std_logic_vector (7 downto 0));
end braun_multiplier4;

architecture behavioral of braun_multiplier4 is
signal s1,s2,s3,s4 :std_logic_vector (3 downto 0);
signal f1,f2,f3,f4 :std_logic_vector (4 downto 0);
component and21
  port( a : in std_logic;
        b : in std_logic;
        c : out std_logic);
  end
end component;
component f_adder
  port ( a : in std_logic;
        b : in std_logic;
        c : in std_logic;
        sum : out std_logic;
        carry : out std_logic);
  end component;
begins
  g1: for i in 0 to 3 generate
    a1: and21 port map (b(0),a(i),s1(i));
    p(0) <=s1(0);
  end generate;
  g2: for j in 0 to 3 generate
    a2: and21 port map (b(1),a(j),s2(j));
  end generate;
  fa1: f_adder port map (s1(1),s2(0),'0',p(1),f1(0));
  fa2: f_adder port map (s1(2),s2(1),'0',f1(1),f1(2));
  fa3: f_adder port map (s1(3),s2(2),'0',f1(3),f1(4));
  g3: for k in 0 to 3 generate
    a3: and21 port map (b(2),a(k),s3(k));
  end generate;
  fa4: f_adder port map (s3(0),f1(0),f1(1),p(2),f2(0));
  fa5: f_adder port map (s3(1),f1(2),f1(3),f2(1),f2(2));
  fa6: f_adder port map (s3(2),f1(4),s2(3),f2(3),f2(4));
end;
g4: for l in 0 to 3 generate
   a3: and21 port map (b(3),a(l),s4(l));
end generate;

fa7: f_adder port map (s4(0),f2(0),f2(1),p(3),f3(0));
fa8: f_adder port map (s4(1),f2(2),f2(3),f3(1),f3(2));
fa9: f_adder port map (s4(2),f2(4),s3(3),f3(3),f3(4));
fa10: f_adder port map ('0',f3(0),f3(1),p(4),f4(0));
fa11: f_adder port map (f4(0),f3(2),f3(3),p(5),f4(1));
fa12: f_adder port map (f4(1),f3(4),s4(3),p(6),p(7));

end behavioral;
VIVA QUESTIONS

1. What type of language is VHDL?
2. What do all VHDL designs begin with?
3. Which block describes a design’s interface and behavior?
4. What is the difference between simulation and synthesis?
5. Which data type defines a single logic signal and bus?
6. What two ways can a vector’s range be described?
7. What are the IEEE STD_LOGIC_1164 data types for single logic signals and buses?
8. What are the only two values for a Boolean type?
9. What are the numerical data types?
10. What is SUBTYPING used for?
11. What type is use to create a user data type?
12. Create the use data type DAYS and assign it the values: MON, TUE, WED, THU, FRI, SAT and SUN.
13. Which data type is used for a string of ASCII characters?
14. Which data type includes time units as values?
15. Create the entity block for a three input XOR gate.
16. Which symbol is used to end all VHDL statements?
17. What part of a port declaration defines signals in or out direction?
18. Which VHDL construct is used to define a literal constant in an entity block?
19. Create the integer constant included in an entity block called BUS_SIZE and assign it a value of 32.
20. Which symbols are used as an assignment operator to assign a literal to an identifier name?
21. What are the two primary ways to describe a logic circuits function within an architecture block?
22. Create the architecture block for the 3-input XOR gate of question 21.
23. Which symbols are used to assign an expression’s result to an output Interface signal?
24. What are the rules used to define an identifier name?
25. Write the statements that will allow a design to access all the contents of the IEEE ARITH.
26. How does a transport delay differ from an inertial delay?
27. What is the purpose of a SIGNAL declaration?
28. Where are SIGNAL declarations placed in the design?
29. Write an assignment statement that assigns the contents of s(5) to t(2).
30. What is the purpose of a process’ sensitivity list?
31. Under what conditions is a process run?
32. What is an EVENT? What is the difference between event and non-event driven process execution?
33. Which symbols are used to differentiate between logic 1 & an integer1?
34. What are the results of using CLK' event as a condition in the if statement of the DEF example?
35. In an if..then..else construct, which statements are executed if the condition is TRUE and which if it is FALSE?
36. Which reserved word is used to nest if..then..else statements?
37. Write the process block that separately tallies positive and negative transitions of the signal TIME_OUT.
38. What are the requirements of a for loop?
39. Write a process block that uses a for loop to set a zero flag high if all the bits in a sixteen (16) bit word are low (zero).
40. What is meant by instantiating a component?
41. How do signal declarations differ from port interface declarations?
42. What is the prime use of signals?
43. How many parameters can be passed into a function?
44. How many results can be returned from a function?
45. Write a function that returns the sum of two 8-bit words.
46. How do procedures differ from functions?
47. Which standard library does not require a library or use statement?
48. What is a PACKAGE?
49. What is the name of the library used by the current design to store compiled results?
1. a) Write HDL code to realize all the logic gates.
b) Write VHDL code to display a character on the given seven segment display by accepting hex key pad input data.

2. a) Write HDL code to realize 2 to 4 decoder.
b) Write VHDL code to display a character on the given seven segment display by accepting hex key pad input data.

3. a) Write HDL code to realize 8 to 3 encoder with priority/without priority.
b) Write VHDL code to control the direction of stepper motor.

4. a) Write HDL code to realize 8 to 1 multiplexer.
b) Write VHDL code to control speed and direction of DC motor.

5. a) Write HDL code to realize 4 to 1 De -multiplexer.
b) Write VHDL code to control speed and direction of DC motor.

6. a) Write HDL code to realize 4- bit binary to gray converter.
b) Write VHDL code to generate triangular wave form using DAC.

7. a) Write HDL code to realize 4-bit comparator.
b) Write VHDL code to generate square wave form using DAC for given duty cycle.

8. a) Write HDL code to model 8-bit ALU.
b) Write VHDL code to generate ramp wave form using DAC.

9. a) Write HDL code to realize SR flip flop.
b) Write VHDL code to generate ramp wave form using DAC.

10. a) Write HDL code to realize JK flip flop.
b) Write VHDL code to generate Square wave form using DAC for given duty cycle.

11. a) Write HDL code to realize D and T flip flop.
b) Write VHDL code to control external lights using relays.

12. a) Write HDL code to realize 4 bit asynchronous reset binary counter.
b) Write VHDL code to control external lights using relays.

13. a) Write HDL code to realize 3 bit synchronous reset binary counter.
b) Write VHDL code to control external lights using relays.