EXPERIMENT- 4
Verilog examples using task and functions, user defined primitive

Objective:

To study task and functions and user defined primitive using Verilog with examples.

Theory:

Task: A task provides the ability to execute common pieces of code from several different places. This common piece of code is written as task so it can be called from different places in the design discerption.

A task is delimited by the keywords **`task`** and **`endtask`**. The syntax for a task declaration is as follows:

```
task task_name
      input arguments
      output arguments
      inout arguments
      task declarations
      local variable declarations
      begin
      statements
      end
endtask
```

Function: Functions are behavioral statements. Functions must be called within always or initial. Functions take one or more inputs, and, in contrast to task, they return only a single output value. Functions are delimited by the keywords **`function`** and **`endfunction`** and are used to implement combinational logic; therefore, functions cannot contain event controls or timing controls.

```
function [range or type] function name
      input declaration
      other declarations
      begin
```

```
        statement
        end
    endfunction
```

User defined primitive

The syntax for a UDP is similar to that for declaring a module. The definition begins with the keyword **`primitive`** and ends with the keyword **`endprimitive`**. The UDP contains a name and a list of ports, which are declared as **`input`** or **`output`**. For a sequential UDP, the output port is declared as **`reg`**. UDPs can have one or more scalar inputs, but only one scalar output. UDPs do not support **`inout`** ports.

```
primitive udp_name (output, input_1, input_2, . . . , input_n);
    output output;
    input input_1, input_2, . . . , input_n;
    Reg sequential_output; //for sequential UDPs
    initial //for sequential UDPs
        table
            state table entries
        endtable
endprimitive
```

Exercise Problems

1. **Write a Verilog code of 4-to-1 multiplexer as UDP and verify the design description by simulation.**

Source Code

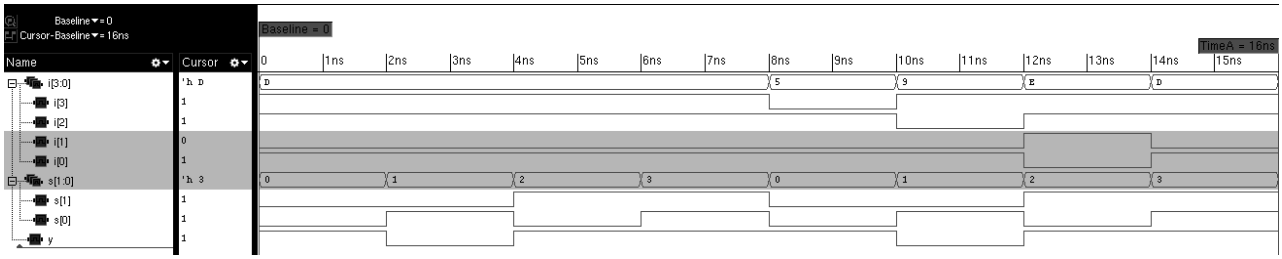
```
primitive mux41 (y,sel1,sel0,i3,i2,i1,i0);
output y;
input sel1,sel0,i3,i2,i1,i0;
table
    //sel      i          y
      0 0   ? ? ? 0  :   0;
      0 0   ? ? ? 1  :   1;
      0 1   ? ? 0 ?  :   0;
      0 1   ? ? 1 ?  :   1;
      1 0   ? 0 ? ?  :   0;
      1 0   ? 1 ? ?  :   1;
      1 1   0 ? ? ?  :   0;
      1 1   1 ? ? ?  :   1;
endtable
endprimitive

module mux4to1(output y, input [1:0]s,input [3:0]i);
mux41(y,s[1],s[0],i[3],i[2],i[1],i[0]);
endmodule
```

Testbench

```
module tb();
reg [1:0]s;
reg [3:0]i;
wire y;
mux4to1 uut(y,s,i);
initial begin
    $monitor($time,"| sel = %b %b | i[3] = %b | i[2] = %b | i[1] = %b |
i[0] = %b | y = %b ",s[1],s[0],i[3],i[2],i[1],i[0],y);
s[1]=0; s[0]=0; i[3]=1; i[2]=1; i[1]=0; i[0]=1;#2
s[1]=0; s[0]=1; i[3]=1; i[2]=1; i[1]=0; i[0]=1;#2
s[1]=1; s[0]=0; i[3]=1; i[2]=1; i[1]=0; i[0]=1;#2
```

```
s[1]=1; s[0]=1; i[3]=1; i[2]=1; i[1]=0; i[0]=1;#2
s[1]=0; s[0]=0; i[3]=0; i[2]=1; i[1]=0; i[0]=1;#2
s[1]=0; s[0]=1; i[3]=1; i[2]=0; i[1]=0; i[0]=1;#2
s[1]=1; s[0]=0; i[3]=1; i[2]=1; i[1]=1; i[0]=0;#2
s[1]=1; s[0]=1; i[3]=1; i[2]=1; i[1]=0; i[0]=1;#2
$finish();
end
endmodule
```

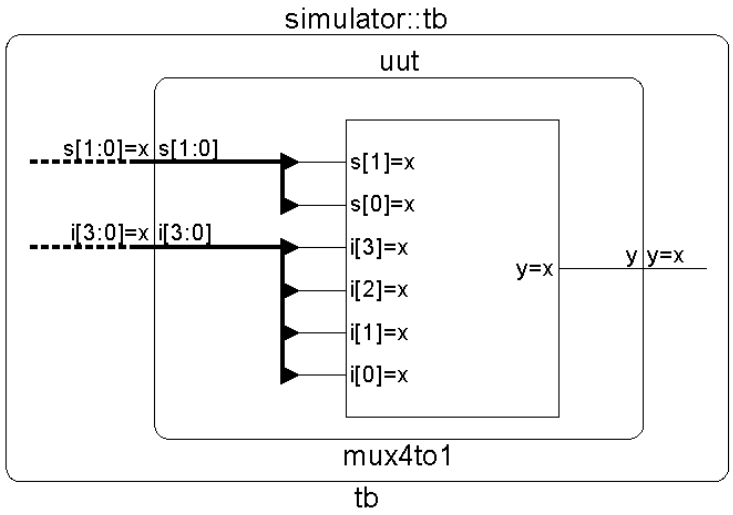
Waveform



Console

```
ncsim>
ncsim> run
                    0| sel = 0 0 | i[3] = 1 | i[2] = 1 | i[1] = 0 | i[0] = 1 | y = 1
                    2| sel = 0 1 | i[3] = 1 | i[2] = 1 | i[1] = 0 | i[0] = 1 | y = 0
                    4| sel = 1 0 | i[3] = 1 | i[2] = 1 | i[1] = 0 | i[0] = 1 | y = 1
                    6| sel = 1 1 | i[3] = 1 | i[2] = 1 | i[1] = 0 | i[0] = 1 | y = 1
                    8| sel = 0 0 | i[3] = 0 | i[2] = 1 | i[1] = 0 | i[0] = 1 | y = 1
                   10| sel = 0 1 | i[3] = 1 | i[2] = 0 | i[1] = 0 | i[0] = 1 | y = 0
                   12| sel = 1 0 | i[3] = 1 | i[2] = 1 | i[1] = 1 | i[0] = 0 | y = 1
                   14| sel = 1 1 | i[3] = 1 | i[2] = 1 | i[1] = 0 | i[0] = 1 | y = 1
Simulation complete via $finish(1) at time 16 NS + 0
./mux41udp_tb.v:16 $finish();\r
ncsim>
```

Schematic



2. Write a Verilog code for 4-bit binary-to-gray code converter using two-input xor gate UDP and verify the design by simulation.

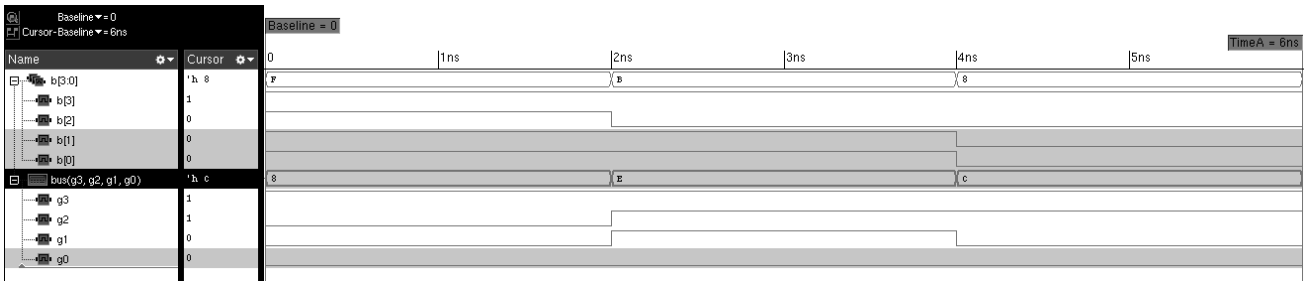
Source Code

```
primitive xorgate(y,x1,x0);
output y;
input x1,x0;
table
    // x1 x0 : y
       0  0  : 0;
       0  1  : 1;
       1  0  : 1;
       1  1  : 0;
endtable
endprimitive
module bintogray(input [3:0]b, output g3,g2,g1,g0);
wire [3:0]grey;
assign g3=b[3];
xorgate(g2,b[3],b[2]);
xorgate(g1,b[2],b[1]);
xorgate(g0,b[1],b[0]);
assign grey={g3,g2,g1,g0};
endmodule
```

Testbench

```
module tb();
reg [3:0]b;
wire g3,g2,g1,g0;
bintogray uut(b,g3,g2,g1,g0);
initial begin
    $monitor($time,"| Binary input = %b | Gray Code = %b%b%b%b
",b,g3,g2,g1,g0);
b[3]=1;b[2]=1;b[1]=1;b[0]=1;#2 //gray =1000
b[3]=1;b[2]=0;b[1]=1;b[0]=1;#2 //gray =1110
b[3]=1;b[2]=0;b[1]=0;b[0]=0;#2 //gray =1100
$finish;
end
endmodule
```

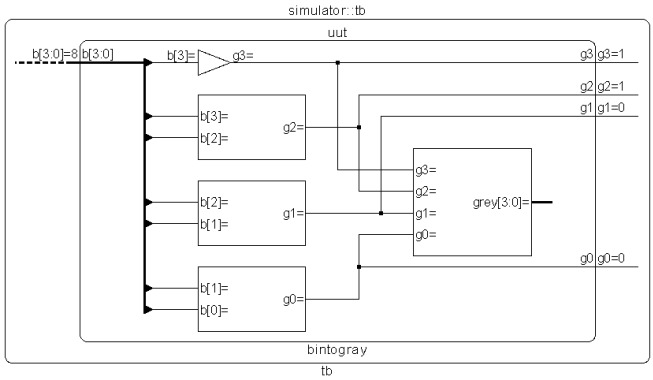
Waveform



Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                  0| Binary input = 1111 | Gray Code = 1000
                  2| Binary input = 1011 | Gray Code = 1110
                  4| Binary input = 1000 | Gray Code = 1100
Simulation complete via $finish(1) at time 6 NS + 0
./bintograyudp_tb.v:10 $finish;
ncsim>
```

Schematic



3. Write a Verilog code to define and call the function that evaluates the two-input exor expression and verify the code by simulation.

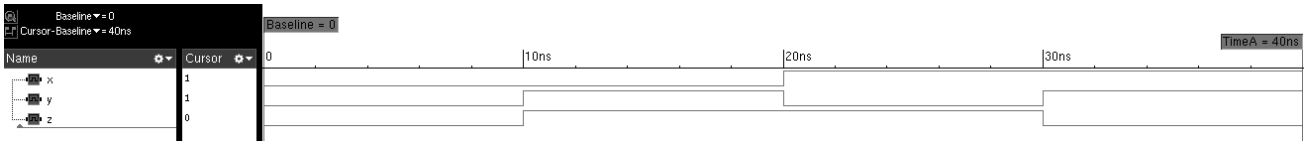
Source Code

```
module xorfunction(output z, input x,y);
  function exor;
    input a,b;
    exor = a ^ b;
  endfunction
assign  z = exor(x,y);
endmodule
```

Testbench

```
module tb();
  reg x, y;
  wire z;
  xorfunction uut (.z(z), .x(x), .y(y));
  initial
  begin
    $monitor($time ," | x=%b  | y=%b  | z=%b ", x, y, z);
    x = 0;y = 0; #10;
    x = 0;y = 1; #10;
    x = 1;y = 0; #10;
    x = 1;y = 1; #10;
    $finish;
  end
endmodule
```

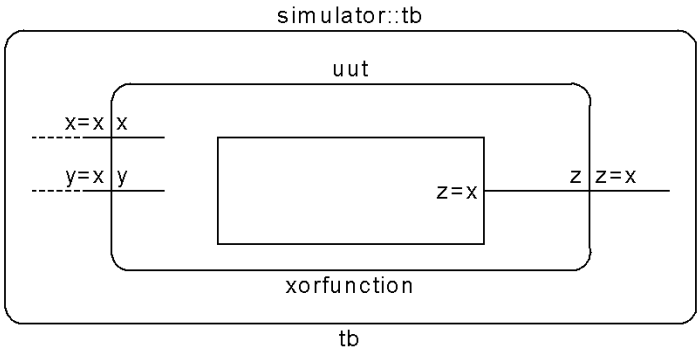
Waveform



Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.x tb.y tb.z
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                    0 | x=0  | y=0  | z=0
                 10 | x=0  | y=1  | z=1
                 20 | x=1  | y=0  | z=1
                 30 | x=1  | y=1  | z=0
Simulation complete via $finish(1) at time 40 NS + 0
./xor_function_tb.v:12     $finish;
ncsim>
```

Schematic



4. Write a Verilog code for the positive-edge-triggered D flip-flop as UDP and verify the design by simulation.

Source Code

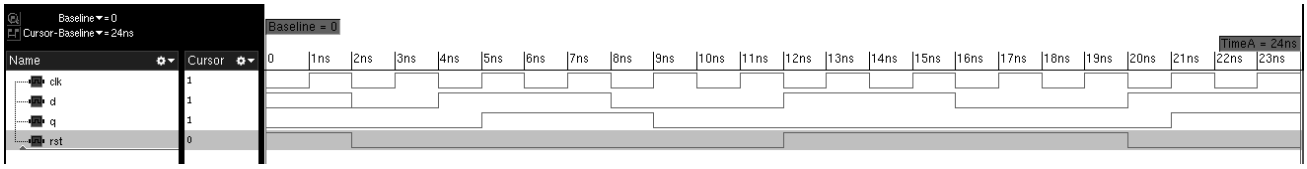
```
primitive dff_udp(q, clk, rst, d);
  output reg q;
  input clk, rst, d;
  initial q = 0;
  table
//clk rst   d :  q  :  q+1
 ?     1     ?  :  ?  :  0; // rst
 ?    (10)   ?  :  ?  :  -; // ignoring negative transiton of rst
(01)   0     1  :  ?  :  1; // q = d
(01)   0     0  :  ?  :  0; // q = d
(0x)   0     ?  :  ?  : - ; // for unknown clk,hold previous state of q
(1?)   0     ?  :  ?  :  -; // ignore negative transiton of clk
(x0)   0     ?  :  ?  :  -; // ignore negative transiton of clk
?      0   (??) :  ?  :  -; // ignore changes in d for no changes in clk
  endtable
endprimitive

module dff(input d,clk,rst,output q);
dff_udp(q,clk,rst,d);
endmodule
```

Testbench

```
module tb();
  reg clk, rst, d;
  wire q;

  dff uut(d, clk, rst, q);

  initial clk = 0;
  always #1 clk=~clk;

  initial begin
    $monitor($time, "| clk=%b | reset=%b | d=%b | q+1=%b",clk,rst,d,q);
    rst = 1;
```

```
#2; rst = 0; d=0;
#2; d = 1;
#4; d = 0;
#4; d = 1;rst = 1;
#4; d = 0;
#4; d = 1;rst = 0;
#4; $finish;
  end
endmodule
```

Waveform



Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                    0| clk=0 | reset=1 | d=x | q+1=0
                    1| clk=1 | reset=1 | d=x | q+1=0
                    2| clk=0 | reset=0 | d=0 | q+1=0
                    3| clk=1 | reset=0 | d=0 | q+1=0
                    4| clk=0 | reset=0 | d=1 | q+1=0
                    5| clk=1 | reset=0 | d=1 | q+1=1
                    6| clk=0 | reset=0 | d=1 | q+1=1
                    7| clk=1 | reset=0 | d=1 | q+1=1
                    8| clk=0 | reset=0 | d=0 | q+1=1
                    9| clk=1 | reset=0 | d=0 | q+1=0
                   10| clk=0 | reset=0 | d=0 | q+1=0
                   11| clk=1 | reset=0 | d=0 | q+1=0
                   12| clk=0 | reset=1 | d=1 | q+1=0
                   13| clk=1 | reset=1 | d=1 | q+1=0
                   14| clk=0 | reset=1 | d=1 | q+1=0
                   15| clk=1 | reset=1 | d=1 | q+1=0
                   16| clk=0 | reset=1 | d=0 | q+1=0
                   17| clk=1 | reset=1 | d=0 | q+1=0
                   18| clk=0 | reset=1 | d=0 | q+1=0
                   19| clk=1 | reset=1 | d=0 | q+1=0
                   20| clk=0 | reset=0 | d=1 | q+1=0
                   21| clk=1 | reset=0 | d=1 | q+1=1
                   22| clk=0 | reset=0 | d=1 | q+1=1
                   23| clk=1 | reset=0 | d=1 | q+1=1
Simulation complete via $finish(1) at time 24 NS + 0
./dffudp_tb.v:19 #4; $finish;
ncsim>
```

Schematic