# EXPERIMENT- 1
# Verilog Dataflow Modeling

**Objective:**
To understand the concepts related to dataflow modeling style and write Verilog Programs using the same.

**Theory:** Modules, ports and dataflow modeling

*Modules* are the basic building blocks for modeling. A module is the principal design entity in Verilog.

**Module Declaration:** The first line of a module declaration specifies the *module name* and *port list* (arguments). The next few lines specify the *i/o type* (input, output or inout) and *width* of each port.

***Syntax***

```
module module_name (port_list);
input [msb:lsb] input_port_list;
output [msb:lsb] output_port_list;
inout [msb:lsb] inout_port_list;
        ... statements...
endmodule
```

**Dataflow modeling:** The data-flow model uses signal assignment statements that are **concurrent** (The order of assign statements does not matter). Dataflow modeling uses *continuous assignment statements* with keyword *assign.*
**assign Y = Boolean Expression using variables and operators.**

A dataflow description is based on function rather than structure and hence uses a number of bit-wise operators.

| Bitwise Verilog Operator | Symbol |
|---|---|
| NOT | ~ |
| AND | & |
| OR | \| |
| XOR | ^ |
| XNOR | ^~ or ~^ |

***Exercise Problems***

1. Write a dataflow Verilog code for following digital building blocks and verify the design by simulation: [i] full adder [ii] full subtractor [iii] 32:1 mux

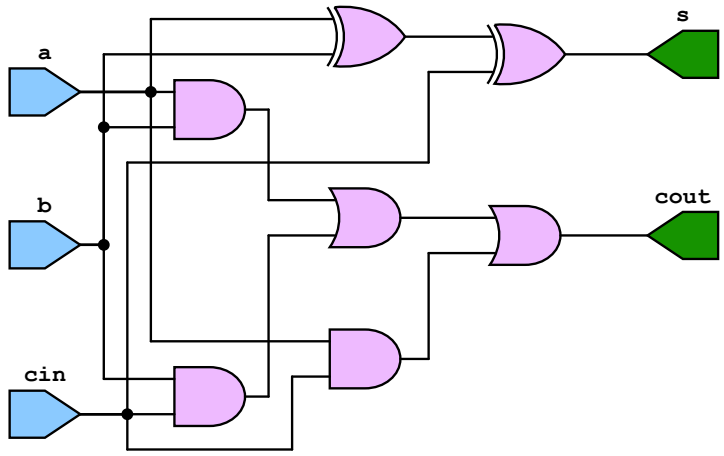
**[i] Full Adder**

A full adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit.

`s= a⊕b⊕cin`

`cout=(a.b)+(b.cin)+(a.cin)`

**Circuit**




**Truth Table**

| a | b | cin | s | cout |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

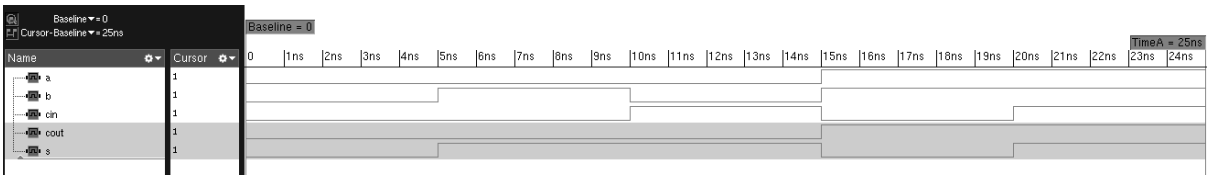
## Source Code

```
module fulladder(a,b,cin,cout,s);
input a,b,cin;
output cout,s;
assign s=a^b^cin;
assign cout=(a&b)|(b&cin)|(cin&a);
endmodule
```

## Testbench

```
module tb();
reg a,b,cin;
wire s,cout;
fulladder uut(a,b,cin,cout,s);
initial begin
   a=0;b=0;cin=0; //cout=0,s=0
#5 a=0;b=1;cin=0; //cout=0,s=1
#5 a=0;b=0;cin=1; //cout=0,s=1
#5 a=1;b=1;cin=0; //cout=1,s=0
#5 a=1;b=1;cin=1; //cout=1,s=1
end
initial begin
$monitor ($time,"a=%b,b=%b,cin=%b,cout=%b,s=%b",a,b,cin,cout,s);
#25 $finish;
end
endmodule
```

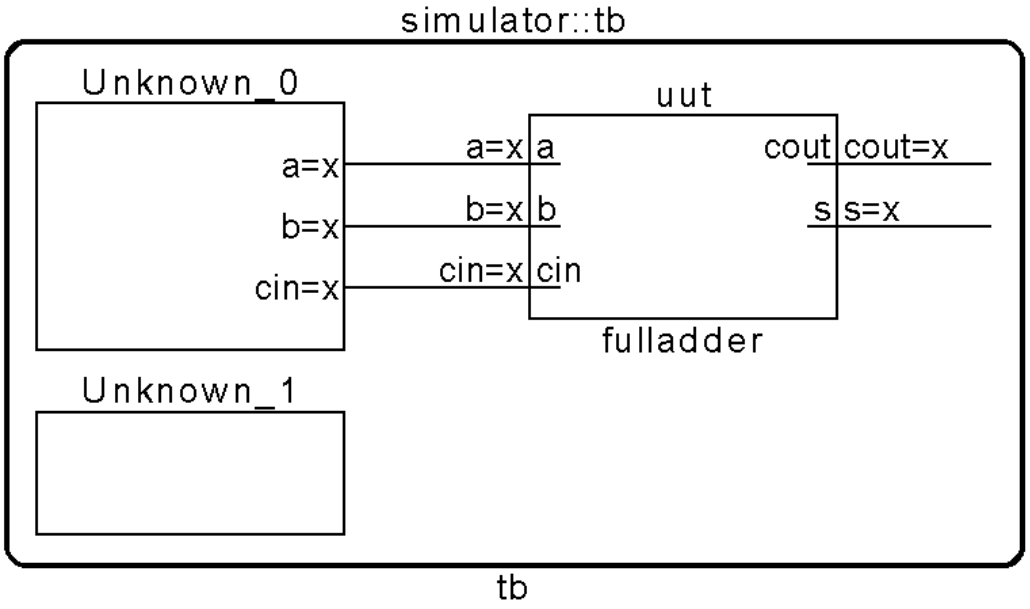
## Waveform



## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                    0a=0,b=0,cin=0,cout=0,s=0
                    5a=0,b=1,cin=0,cout=0,s=1
                   10a=0,b=0,cin=1,cout=0,s=1
                   15a=1,b=1,cin=0,cout=1,s=0
                   20a=1,b=1,cin=1,cout=1,s=1
Simulation complete via $finish(1) at time 25 NS + 0
./fulladder_tb.v:14 #25 $finish;
ncsim>
```

## Schematic



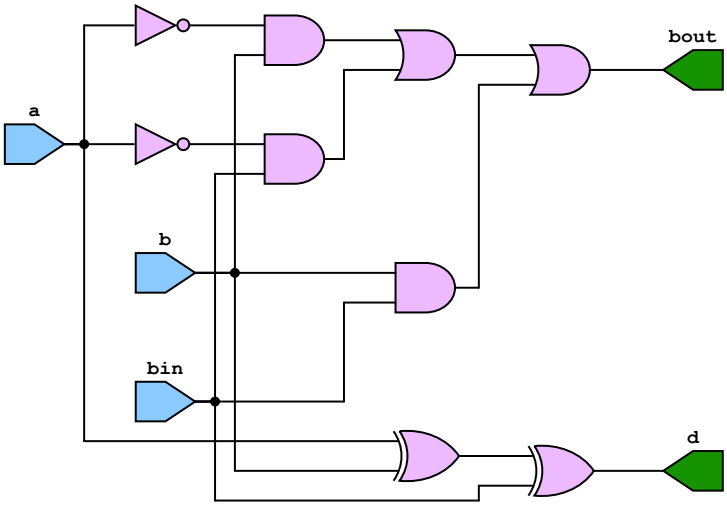
## [ii] Full Subtractor

A full subtractor is a combinational circuit that subtracts two bits and a borrow and outputs a difference bit and a borrow bit.

d= a⊕b⊕bin

cout=(a’.b)+(a’.bin)+(b.bin)

**Circuit**




**Truth Table**

| a | b | bin | d | bout |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

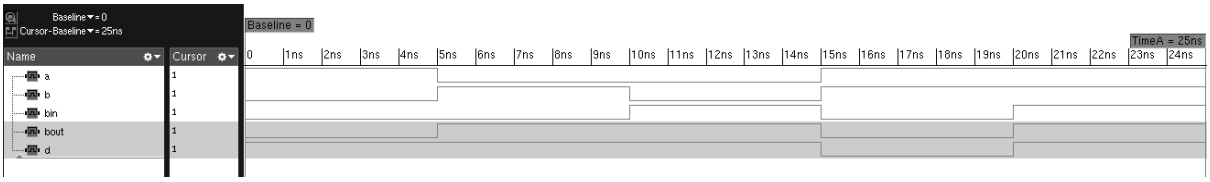
## Source Code

```
module fullsubtractor(a,b,bin,bout,d);
input a,b,bin;
output bout,d;
assign d=a^b^bin;
assign bout=(~a&b)|(~a&bin)|(b&bin);
endmodule
```

## Testbench

```
module tb();
reg a,b,bin;
wire d,bout;
fullsubtractor uut(a,b,bin,bout,d);
initial begin
   a=1;b=0;bin=0; //bout=0,d=1
#5 a=0;b=1;bin=0; //bout=1,d=1
#5 a=0;b=0;bin=1; //bout=1,d=1
#5 a=1;b=1;bin=0; //bout=0,d=0
#5 a=1;b=1;bin=1; //bout=1,d=1
end
initial begin
$monitor ($time,"a=%b,b=%b,bin=%b,bout=%b,d=%b",a,b,bin,bout,d);
#25 $finish;
end
endmodule
```

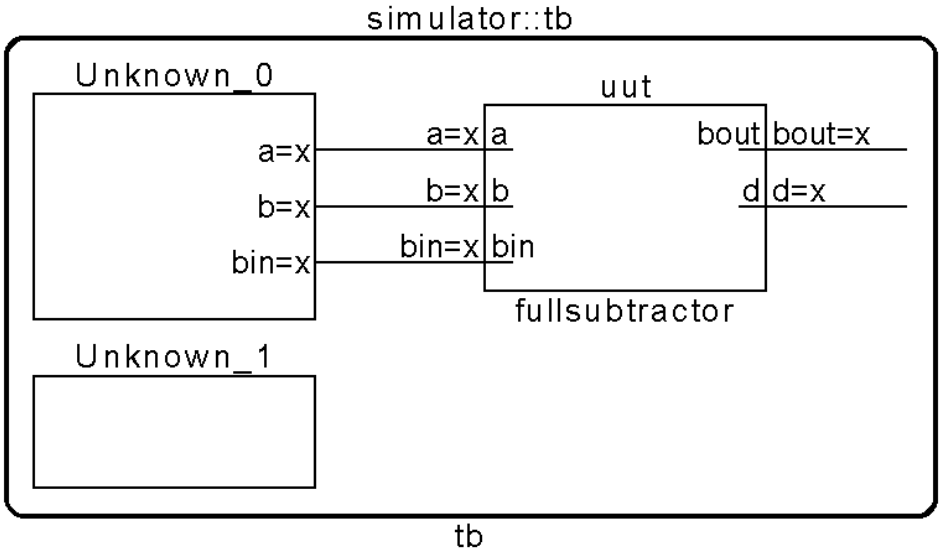
## Waveform



## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                  0a=1,b=0,bin=0,bout=0,d=1
                  5a=0,b=1,bin=0,bout=1,d=1
                 10a=0,b=0,bin=1,bout=1,d=1
                 15a=1,b=1,bin=0,bout=0,d=0
                 20a=1,b=1,bin=1,bout=1,d=1
Simulation complete via $finish(1) at time 25 NS + 0
./fullsubtractor_tb.v:14 #25 $finish;
ncsim>
```

## Schematic

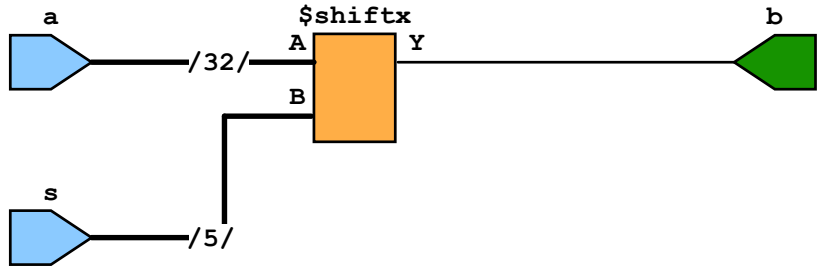



### [iii] 32:1 Mux

A 32:1 multiplexer (MUX) is a logic circuit that accepts 32 input signals and allows only one of them at a time to get through the output. The routing of desired data input to the output is controlled by the 5 select inputs.

There are $2^n$ input lines and $n$ select lines whose bit combinations determine which input is selected. Here $n=5$.

## Circuit



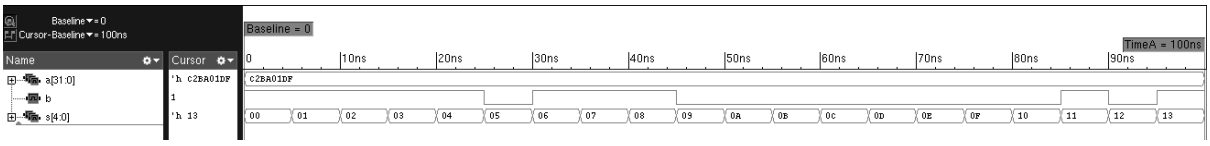
## Source Code

```
module mux(a,b,s);
input [31:0]a;
input [4:0]s;
output b;
assign b=a[s];
endmodule
```

## Testbench

```
module tb();
reg [31:0]a;
reg [4:0]s;
wire b;
mux uut (a,b,s);
initial begin
a= 32'b1100_0010_1011_1010_0000_0001_1101_1111;
for(s=0;s<32;s=s+1)
begin
#5; $display("in=%b,out=%b,sel=%b",a,b,s);
end
end
initial begin
#100 $finish;
$monitor ("in=%b,out=%b,sel=%b",$time,a,b,s);
end
endmodule
```

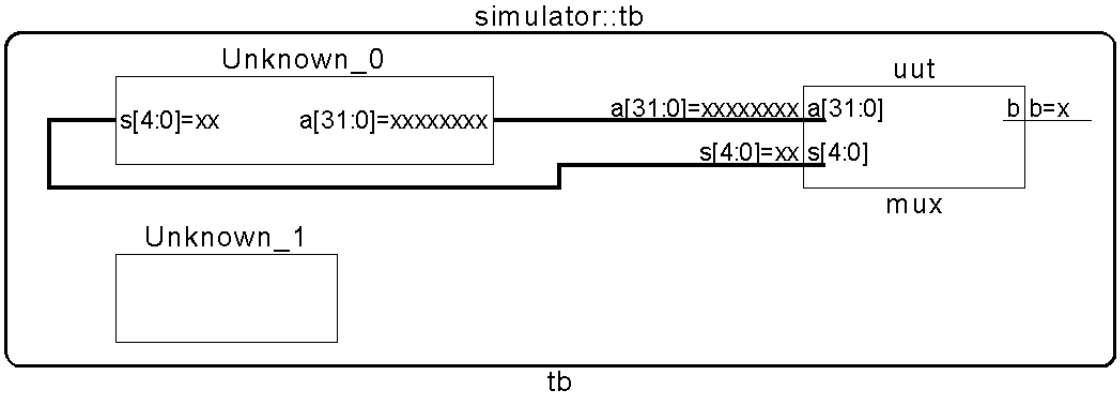
## Waveform



## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
in=11000010101110100000000111011111, out=1, sel=00000
in=11000010101110100000000111011111, out=1, sel=00001
in=11000010101110100000000111011111, out=1, sel=00010
in=11000010101110100000000111011111, out=1, sel=00011
in=11000010101110100000000111011111, out=1, sel=00100
in=11000010101110100000000111011111, out=0, sel=00101
in=11000010101110100000000111011111, out=1, sel=00110
in=11000010101110100000000111011111, out=1, sel=00111
in=11000010101110100000000111011111, out=1, sel=01000
in=11000010101110100000000111011111, out=0, sel=01001
in=11000010101110100000000111011111, out=0, sel=01010
in=11000010101110100000000111011111, out=0, sel=01011
in=11000010101110100000000111011111, out=0, sel=01100
in=11000010101110100000000111011111, out=0, sel=01101
in=11000010101110100000000111011111, out=0, sel=01110
in=11000010101110100000000111011111, out=0, sel=01111
in=11000010101110100000000111011111, out=0, sel=10000
in=11000010101110100000000111011111, out=1, sel=10001
in=11000010101110100000000111011111, out=0, sel=10010
Simulation complete via $finish(1) at time 100 NS + 0
./321mux_tb.v:14 #100 $finish;
ncsim>
```

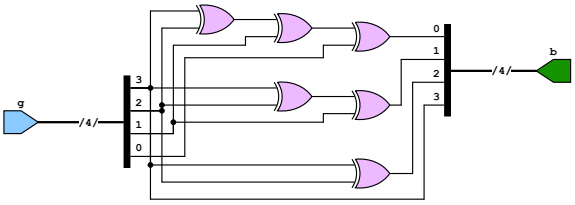
## Schematic



**2. Write a dataflow Verilog code for 4-bit Gray-to-Binary code converter and verify the design by simulation.**

The Gray code is a non-weighted code. It is a cyclic code since successive code words in this code differ in one bit position. 4-bit-Gray-to-Binary code converter is a digital circuit that converts 4-bit gray code input into its equivalent 4-bit binary code.

## Circuit




## Truth Table

| Gray Code | Binary Code |
|---|---|
| 0000 | 0000 |
| 0001 | 0001 |
| 0011 | 0010 |
| 0010 | 0011 |
| 0110 | 0100 |
| 0111 | 0101 |
| 0101 | 0110 |
| 0100 | 0111 |
| 1100 | 1000 |
| 1101 | 1001 |
| 1111 | 1010 |
| 1110 | 1011 |
| 1010 | 1100 |
| 1011 | 1101 |
| 1001 | 1110 |
| 1000 | 1111 |

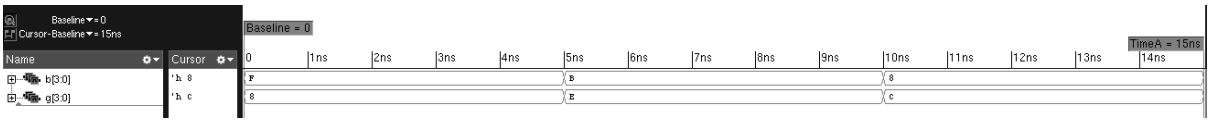
## Source Code

```
module graytobin4(b,g);
input [3:0]g;
output [3:0]b;
assign b[3]=g[3];
assign b[2]=g[3]^g[2];
assign b[1]=g[3]^g[2]^g[1];
assign b[0]=g[3]^g[2]^g[1]^g[0];
endmodule
```

## Testbench

```
module tb();
reg [3:0]g;
wire [3:0]b;
graytobin4 uut(b,g);
initial begin
g=4'b1000; // bin 1111
#5 g=4'b1110; // bin 1011
#5 g=4'b1100; // bin 1000
end
initial begin
$monitor ($time,"g=%b,b=%b",g,b);
#15 $finish;
end
endmodule
```

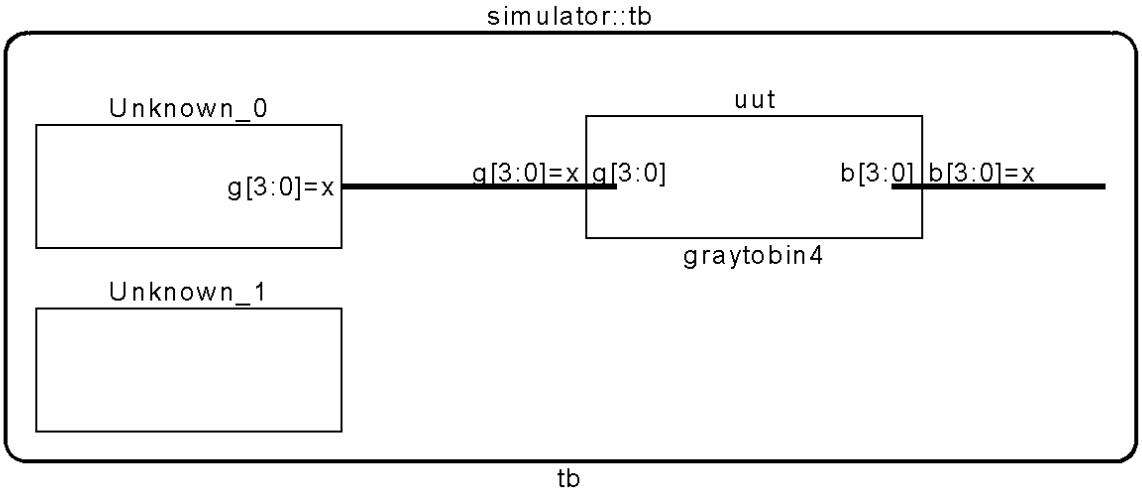
## Waveform




## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                   0g=1000,b=1111
                   5g=1110,b=1011
                  10g=1100,b=1000
Simulation complete via $finish(1) at time 15 NS + 0
./graytobin4_tb.v:12 #15 $finish;
ncsim>
```

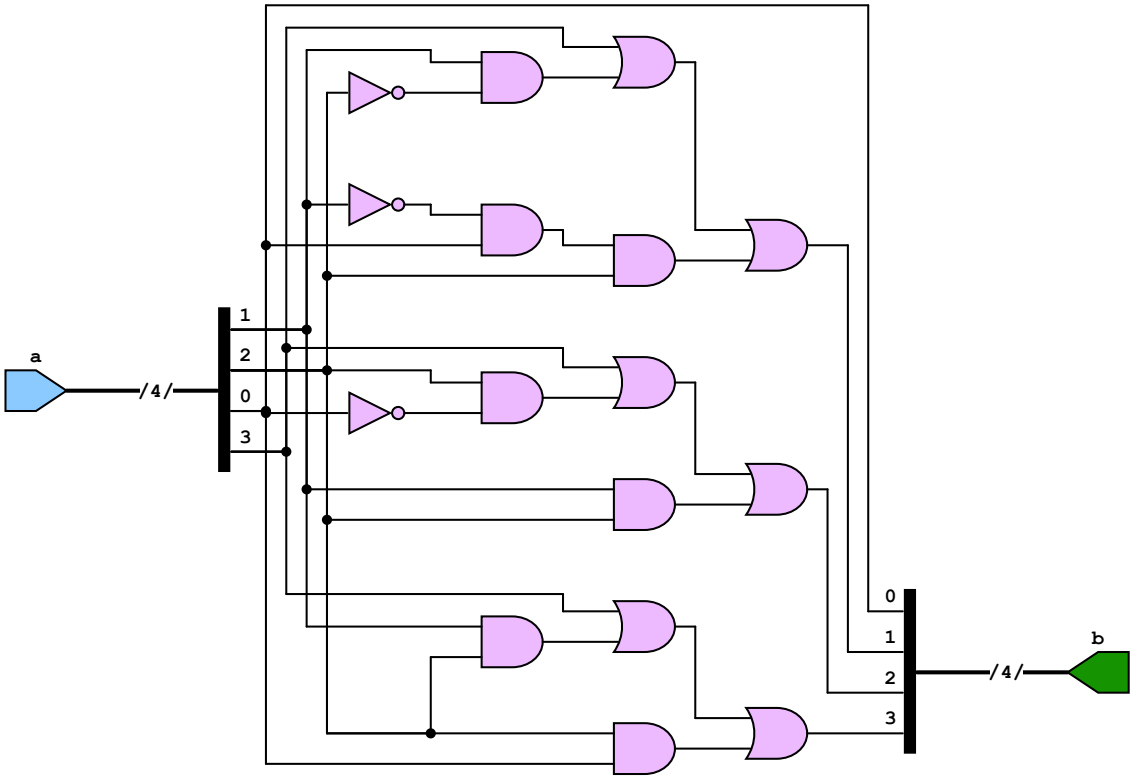
## Schematic




### 3. Write a dataflow Verilog code for 8421 to 2421 code converter and verify the design by simulation.

The 8421 BCD code is a weighted code in where each decimal digit,0 through 9 is coded by a 4-bit binary number. The 2421 code in another type of weighted code in which weights assigned to the four bits are 2,4,2,1 respectively. 8421 to 2421 code converter is a digital circuit that converts a 4-bit binary-coded decimal (BCD) number in the 8421-code format into its equivalent 4-bit 2421 code format.

## Circuit



## Truth Table

| Decimal | 8421 Code | 2421 Code |
|---|---|---|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0010 |
| 3 | 0011 | 0011 |
| 4 | 0100 | 0100 |
| 5 | 0101 | 1011 |
| 6 | 0110 | 1100 |
| 7 | 0111 | 1101 |
| 8 | 1000 | 1110 |
| 9 | 1001 | 1111 |

## Source Code

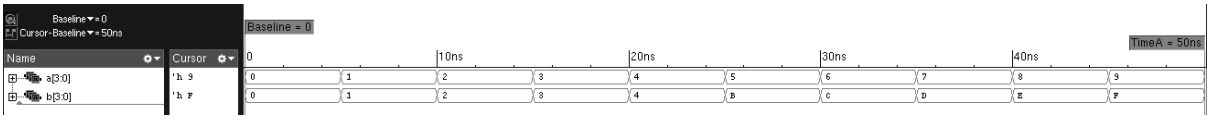
```
module codeconverter(a,b);
input [3:0]a;
output [3:0]b;
assign b[3]=a[3]|(a[1]&a[2])|(a[2]&a[0]);
assign b[2]=a[3]|(a[2]&(~a[0]))|(a[1]&a[2]);
assign b[1]=a[3]|(a[1]&(~a[2]))|((~a[1])&a[0]&a[2]);
assign b[0]=a[0];
endmodule
```

## Testbench

```
module tb();
reg [3:0]a;
wire [3:0]b;
codeconverter uut(a,b);
initial begin
a=4'b0000;
#5 a=4'b0001;
```

```
#5 a=4'b0010;
#5 a=4'b0011;
#5 a=4'b0100;
#5 a=4'b0101;
#5 a=4'b0110;
#5 a=4'b0111;
#5 a=4'b1000;
#5 a=4'b1001;
end
initial begin
$monitor ($time,"a=%b,b=%b",a,b);
#50 $finish;
end
endmodule
```

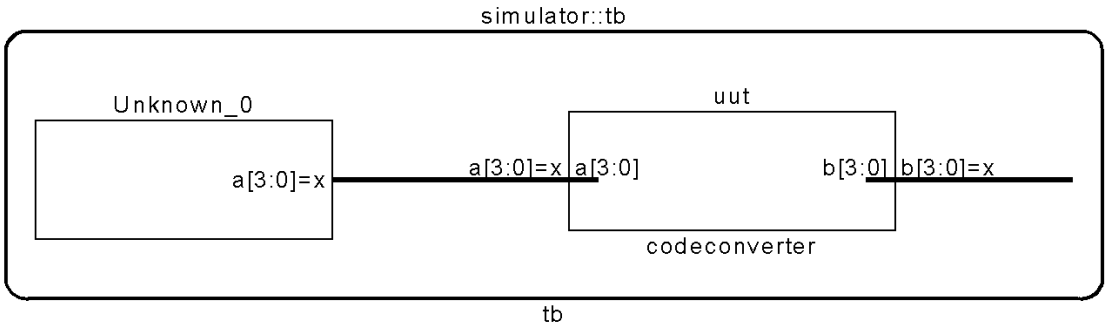
## Waveform




## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                  0a=0000,b=0000
                  5a=0001,b=0001
                 10a=0010,b=0010
                 15a=0011,b=0011
                 20a=0100,b=0100
                 25a=0101,b=1011
                 30a=0110,b=1100
                 35a=0111,b=1101
                 40a=1000,b=1110
                 45a=1001,b=1111
Simulation complete via $finish(1) at time 50 NS + 0
./codeconverter_tb.v:19 #50 $finish;
ncsim>
```

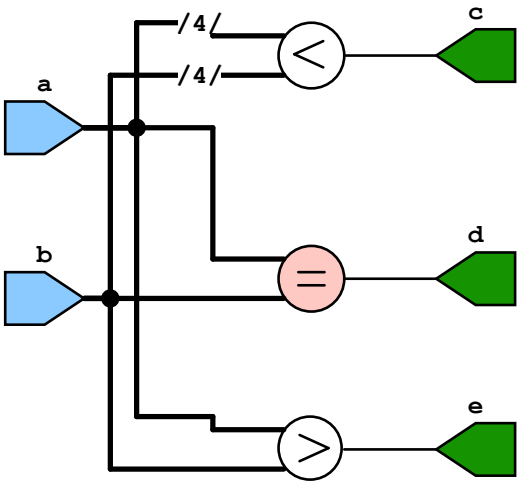
## Schematic




4. **Write a dataflow Verilog code for N bit magnitude comparator and verify the design by simulation.**

A comparator is a logic circuit used to compare the magnitudes of two n-bit binary numbers and depending on the design provides an output that is when the two numbers are equal or additionally provides output that signifies which of the number is greater when equality does not hold.

## Circuit



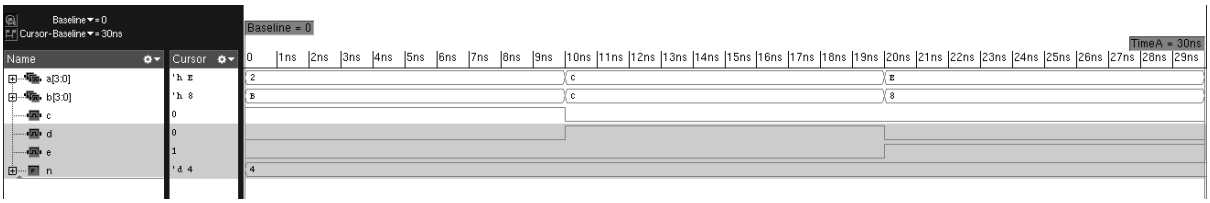

## Source Code

```
module magcomp #(parameter n=4)(a,b,c,d,e);
input [n-1:0]a;
input [n-1:0]b;
output c,d,e;
assign c=(a<b);
assign d=(a==b);
assign e=(a>b);
endmodule
```

## Testbench

```
module tb();
parameter n=4;
reg [n-1:0]a,b;
wire c,d,e;
magcomp uut (a,b,c,d,e);
initial begin
a=4'b0010; b=4'b1011;
#10 a=4'b1100; b=4'b1100;
#10 a=4'b1110; b=4'b1000;
end
initial begin
$monitor ($time,"a=%b,b=%b,a<b=%b,a=b=%b,a>b=%b",a,b,c,d,e);
#30 $finish;
end
endmodule
```

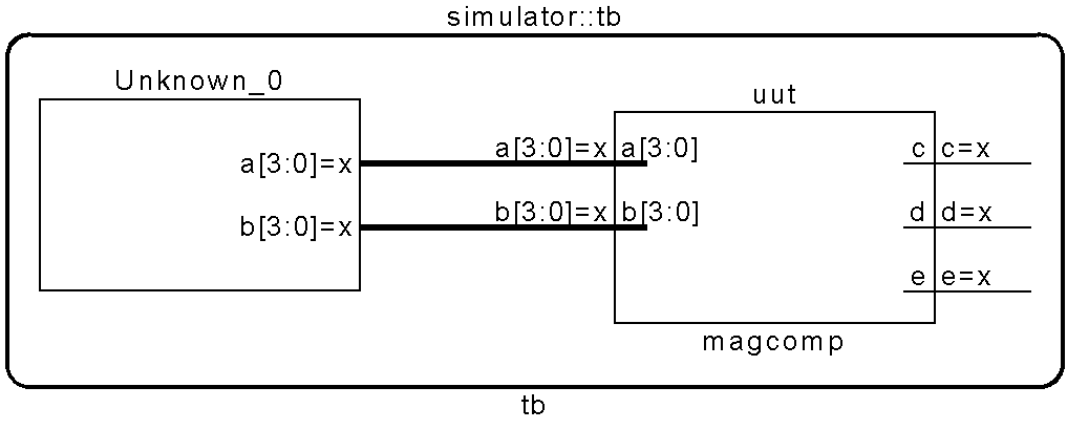
## Waveform




## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                  0a=0010,b=1011,a<b=1,a=b=0,a>b=0
                 10a=1100,b=1100,a<b=0,a=b=1,a>b=0
                 20a=1110,b=1000,a<b=0,a=b=0,a>b=1
Simulation complete via $finish(1) at time 30 NS + 0
./nbitmagnitudecomparator_tb.v:13 #30 $finish;
ncsim>
```

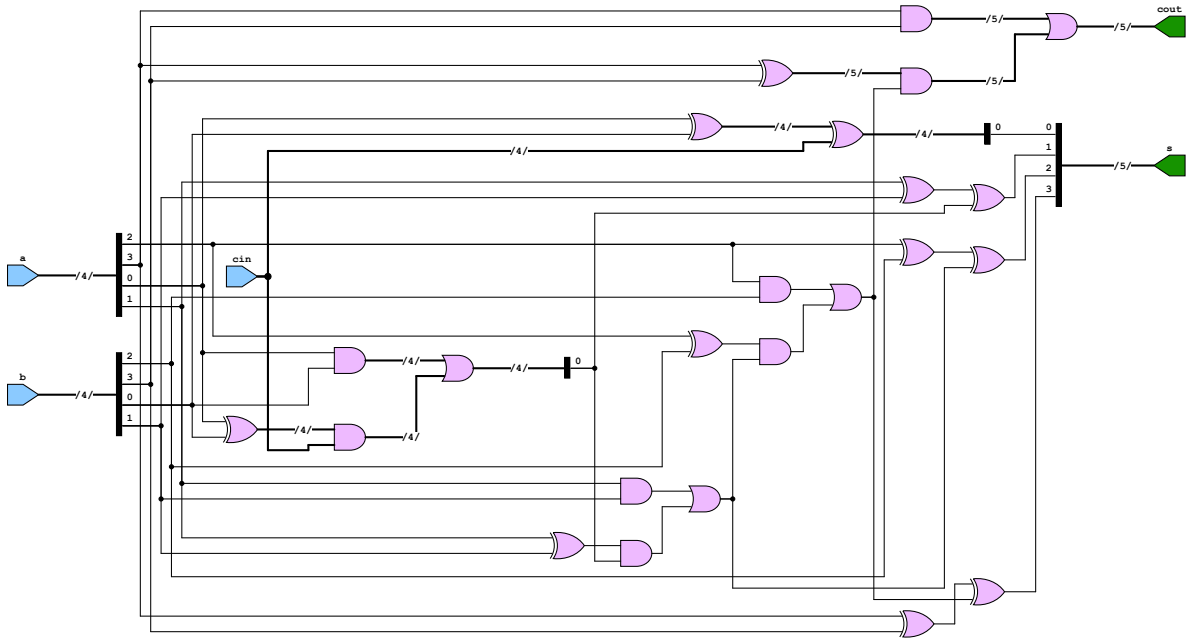
## Schematic




### 5. Write a dataflow Verilog code for 4-bit adder and verify the design by simulation.

A 4-bit adder is a logic circuit that adds two 4-bit binary numbers and outputs the sum along with a carry-out bit.

## Circuit




## Source Code

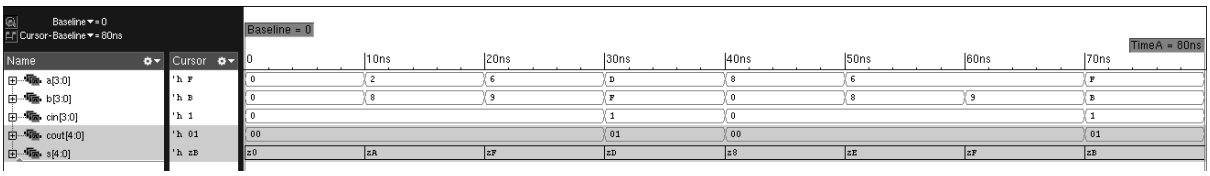
```
module bitadder(a,b,cin,s,cout);
input [3:0]a;
input [3:0]b,cin;
output [4:0]s,cout;
wire [3:1]c;
assign s[0]=a[0]^b[0]^cin;
```

```
assign c[1]=(a[0]&b[0])|((a[0]^b[0])&cin);
assign s[1]=(a[1]^b[1]^c[1]);
assign c[2]=(a[1]&b[1])|((a[1]^b[1])&c[1]);
assign s[2]=(a[2]^b[2]^c[2]);
assign c[3]=(a[2]&b[2])|((a[2]^b[2])&c[2]);
assign s[3]=(a[3]^b[3]^c[3]);
assign cout=(a[3]&b[3])|((a[3]^b[3])&c[3]);
endmodule
```

## Testbench

```
module tb();
reg [3:0]a;
reg [3:0]b,cin;
wire [4:0]s,cout;
bitadder uut(a,b,cin,s,cout);
initial begin
a=4'b0000;b=4'b0000;cin=1'b0;
#10 a=4'b0010;b=4'b1000;cin=1'b0;
#10 a=4'b0110;b=4'b1001;cin=1'b0;
#10 a=4'b1101;b=4'b1111;cin=1'b1;
#10 a=4'b1000;b=4'b0000;cin=1'b0;
#10 a=4'b0110;b=4'b1000;cin=1'b0;
#10 a=4'b0110;b=4'b1001;cin=1'b0;
#10 a=4'b1111;b=4'b1011;cin=0'b1;
end
initial begin
$monitor ($time,"a=%b,b=%b,cin=%b,s=%b,cout=%b",a,b,cin,s,cout);
#80 $finish;
end
endmodule
```

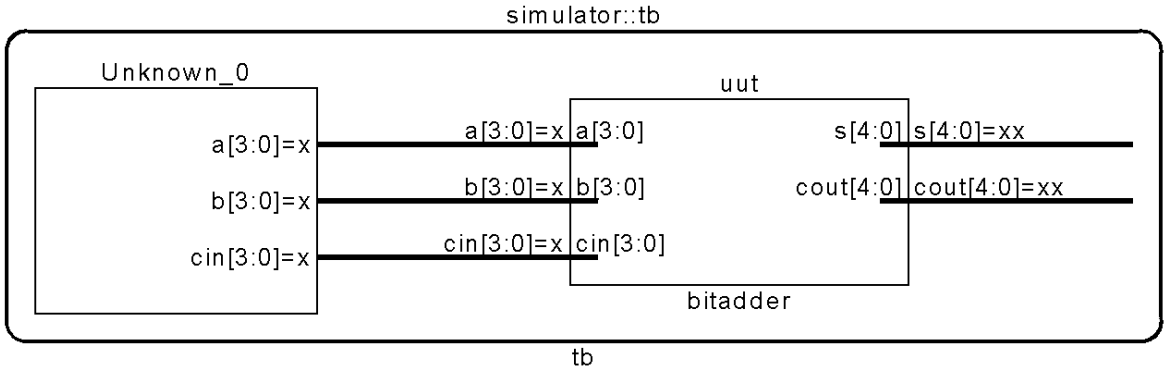
## Waveform



## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                  0a=0000,b=0000,cin=0000,s=z0000,cout=00000
                 10a=0010,b=1000,cin=0000,s=z1010,cout=00000
                 20a=0110,b=1001,cin=0000,s=z1111,cout=00000
                 30a=1101,b=1111,cin=0001,s=z1101,cout=00001
                 40a=1000,b=0000,cin=0000,s=z1000,cout=00000
                 50a=0110,b=1000,cin=0000,s=z1110,cout=00000
                 60a=0110,b=1001,cin=0000,s=z1111,cout=00000
                 70a=1111,b=1011,cin=0001,s=z1011,cout=00001
Simulation complete via $finish(1) at time 80 NS + 0
./4bitadder_tb.v:18 #80 $finish;
ncsim>
```

## Schematic

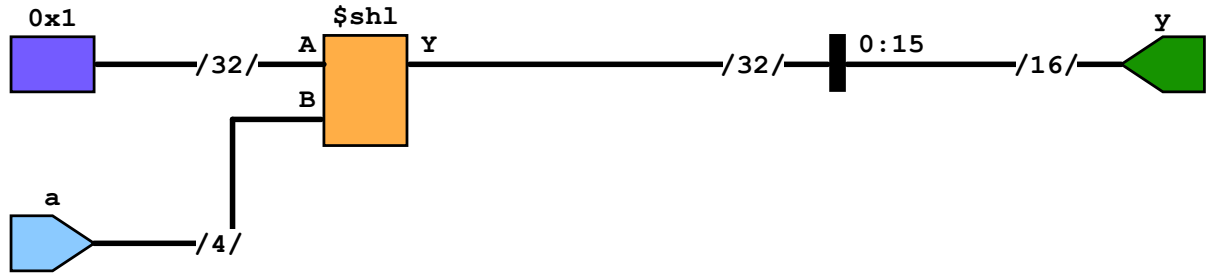


## 6. Write a dataflow Verilog code 4:16 decoder and verify the design by simulation.

A decoder is a logic circuit that converts an *N*-bit binary input code into *M* output lines such that only one output line is activated for each one of the possible combinations of inputs.

Here *N=4* and *M=16.*

### Circuit




### Source Code

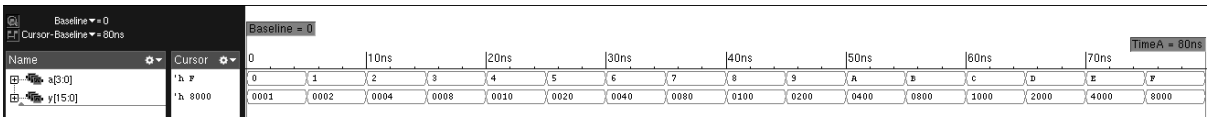
```
module decoder(a,y);
input [3:0]a;
output [15:0]y;
assign y=1<<a;
endmodule
```

### Testbench

```
module tb();
reg [3:0]a;
wire [15:0]y;
decoder uut(a,y);
initial begin
a=4'b0000;
#5 a=4'b0001;
#5 a=4'b0010;
#5 a=4'b0011;
#5 a=4'b0100;
#5 a=4'b0101;
#5 a=4'b0110;
#5 a=4'b0111;
```

```
#5 a=4'b1000;
#5 a=4'b1001;
#5 a=4'b1010;
#5 a=4'b1011;
#5 a=4'b1100;
#5 a=4'b1101;
#5 a=4'b1110;
#5 a=4'b1111;
end
initial begin
$monitor($time,"a=%b y=%b ",a,y);
#80 $finish;
end
endmodule
```

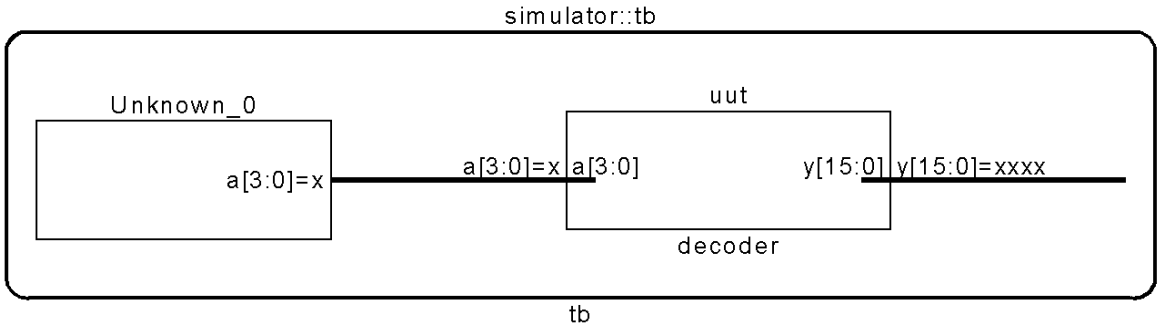
## Waveform




## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                  0a=0000 y=0000000000000001
                  5a=0001 y=0000000000000010
                 10a=0010 y=0000000000000100
                 15a=0011 y=0000000000001000
                 20a=0100 y=0000000000010000
                 25a=0101 y=0000000000100000
                 30a=0110 y=0000000001000000
                 35a=0111 y=0000000010000000
                 40a=1000 y=0000000100000000
                 45a=1001 y=0000001000000000
                 50a=1010 y=0000010000000000
                 55a=1011 y=0000100000000000
                 60a=1100 y=0001000000000000
                 65a=1101 y=0010000000000000
                 70a=1110 y=0100000000000000
                 75a=1111 y=1000000000000000
Simulation complete via $finish(1) at time 80 NS + 0
./decoder_tb.v:25 #80 $finish;
ncsim>
```

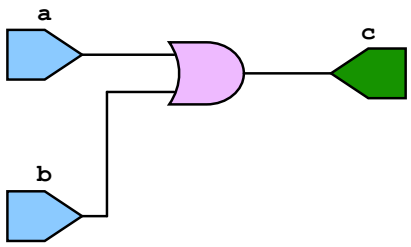
## Schematic




7. **Using inertial delay, write a dataflow Verilog code of suitable design and perform the simulation (Analyse the results).**

Inertial delay is a type of delay model that simulates the behavior of a physical circuit. Inertial delay will filter out input changes that are shorter than the specified delay time. It is useful for modeling components that have a certain minimum response time or "settling" period, where very short input pulses may not affect the output. It helps to prevent unrealistic behavior in the simulation.

## Circuit




## Source Code

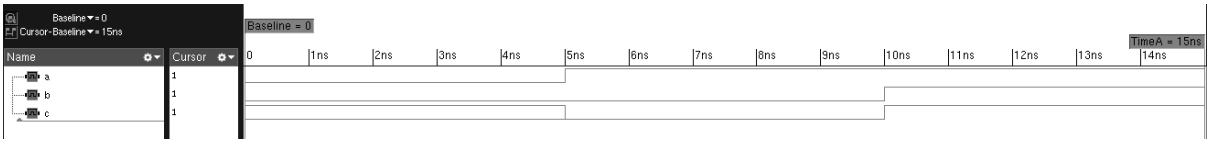
```
module inertialdelay(a,b,c);
input a,b;
output c;
assign #5 c=a|b;
endmodule
```

## Testbench

```
module tb();
reg a,b;
wire c;
inertialdelay uut(a,b,c);
```

```
initial begin
a=0;b=0;
#5 a=1;b=0;
#5 a=1;b=1;
end
initial begin
$monitor($time,"a=%b b=%b c=%b",a,b,c);
#15 $finish;
end
endmodule
```

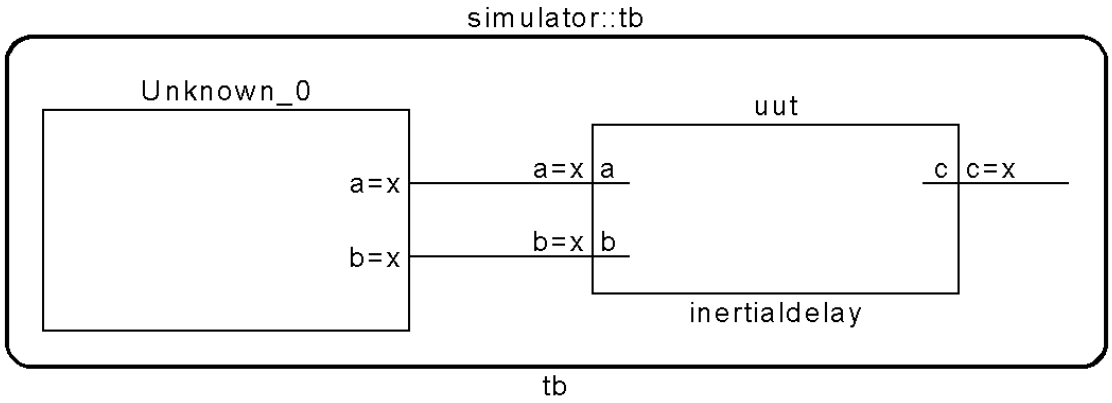
## Waveform




## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                   0a=0 b=0 c=x
                   5a=1 b=0 c=0
                  10a=1 b=1 c=1
Simulation complete via $finish(1) at time 15 NS + 0
./inertialdelay_tb.v:12 #15 $finish;
ncsim>
```

## Schematic



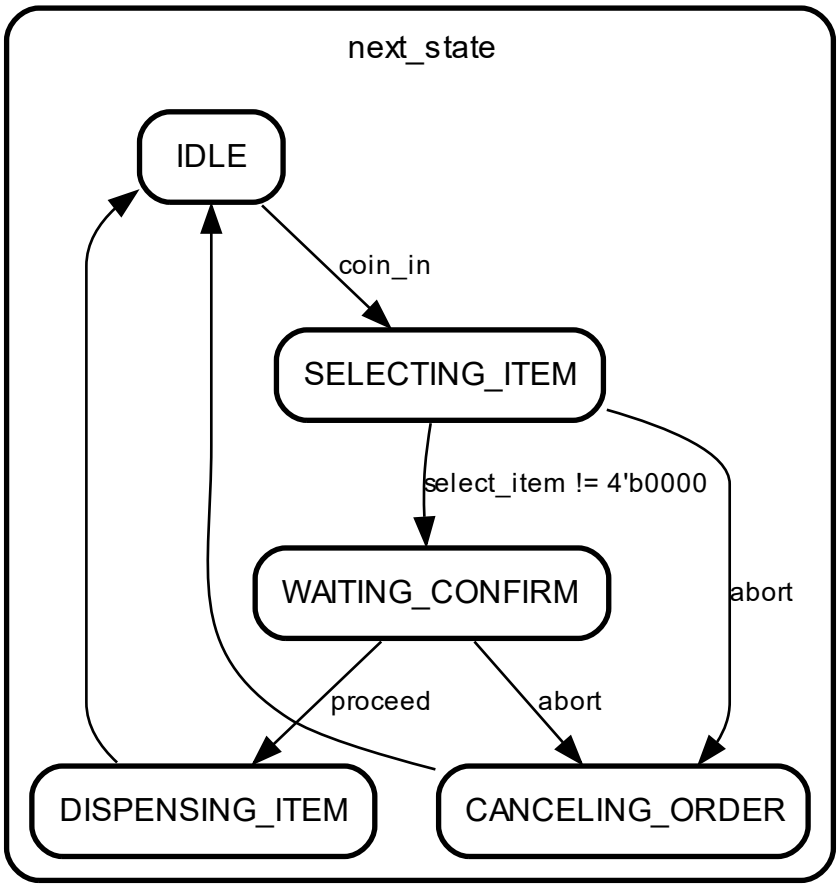

8. **Using transport delay, write a dataflow Verilog code of suitable design and perform the simulation (Analyse the results).**
   The circuit cannot be modelled in Dataflow modelling of Verilog.

## Open Ended Problems

1. **Implement and design a vending machine by using FSM. Your design should meet the following specifications:**
   a. **Input is of PKR 50 only.**
   b. **Ten items for purchasing.**
   c. **Print Order receipt.**
   d. **Cancel order option.**

   **Draw the FSM.**
   **Draw the logic circuit.**
   **Simulate using EDA tools.**

## State Diagram




## Source Code

```
module vending_machine (
input wire clk,
input wire reset,
input wire [3:0] select_item,
input wire proceed,
```

```
input wire abort,
input wire coin_in,
output reg dispense_item,
output reg print_order,
output reg print_cancel);
localparam IDLE = 4'b0000;
localparam SELECTING_ITEM = 4'b0001;
localparam WAITING_CONFIRM = 4'b0010;
localparam DISPENSING_ITEM = 4'b0011;
localparam CANCELING_ORDER = 4'b0100;
reg[2:0] current_state, next_state;
always @(posedge clk or posedge reset) begin
if (reset)
current_state <= IDLE;
else
current_state <= next_state;
end
always @(*) begin
dispense_item = 0;
print_order = 0;
print_cancel = 0;
case (current_state)
DISPENSING_ITEM: begin
dispense_item = 1;
print_order = 1;
end
CANCELING_ORDER: begin
print_cancel = 1;
end
endcase
end
always @(*) begin
next_state = current_state;
case (current_state)
IDLE: begin
if (coin_in)
next_state = SELECTING_ITEM;
end
```

```
SELECTING_ITEM: begin
if (abort)
next_state = CANCELING_ORDER;
else if (select_item != 4'b0000)
next_state = WAITING_CONFIRM;
end
WAITING_CONFIRM: begin
if (abort)
next_state = CANCELING_ORDER;
else if (proceed)
next_state = DISPENSING_ITEM;
end
DISPENSING_ITEM: begin
next_state = IDLE;
end
CANCELING_ORDER: begin
next_state = IDLE;
end
default: next_state = IDLE;
endcase
end
endmodule
```

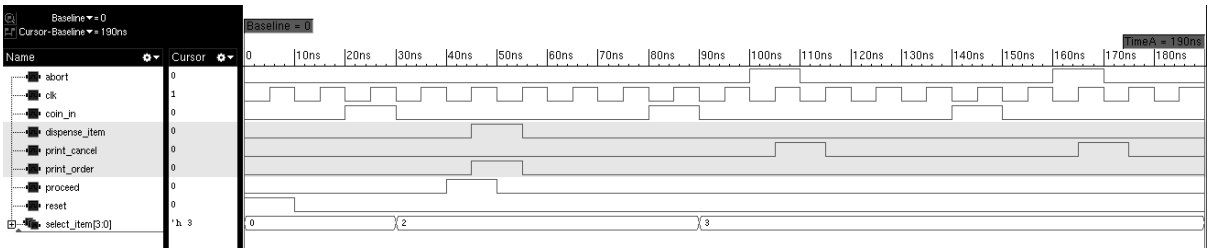
## Testbench

```
module tb();
reg clk;
reg reset;
reg [3:0] select_item;
reg proceed;
reg abort;
reg coin_in;
wire dispense_item;
wire print_order;
wire print_cancel;
vending_machine uut (
        .clk(clk),
        .reset(reset),
```

```
        .select_item(select_item),
        .proceed(proceed),
        .abort(abort),
        .coin_in(coin_in),
        .dispense_item(dispense_item),
        .print_order(print_order),
        .print_cancel(print_cancel)
    );
always begin
        #5 clk = ~clk;
end
initial begin
clk = 0;
reset = 1;
select_item = 4'b0000;
proceed = 0;
abort = 0;
coin_in = 0;
#10;
reset = 0;
#10;
coin_in = 1;
#10;
coin_in = 0;
select_item = 4'b0010;
#10;
proceed = 1;
#10;
proceed = 0;
#20;
#10;
coin_in = 1;
#10;
coin_in = 0;
select_item = 4'b0011;
#10;
abort = 1;
#10;
```

```
abort = 0;
#20;
#10;
coin_in = 1;
#10;
coin_in = 0;
#10;
abort = 1;
#10;
abort = 0;
#20;
$finish;
end
initial begin
$monitor("Time = %0t | State = %0b | Coin = %b | Item = %b | Proceed = %b
| Abort = %b | Dispense = %b | Order = %b | Cancel = %b",
$time, uut.current_state, coin_in, select_item, proceed, abort,
dispense_item, print_order, print_cancel);
end
endmodule
```

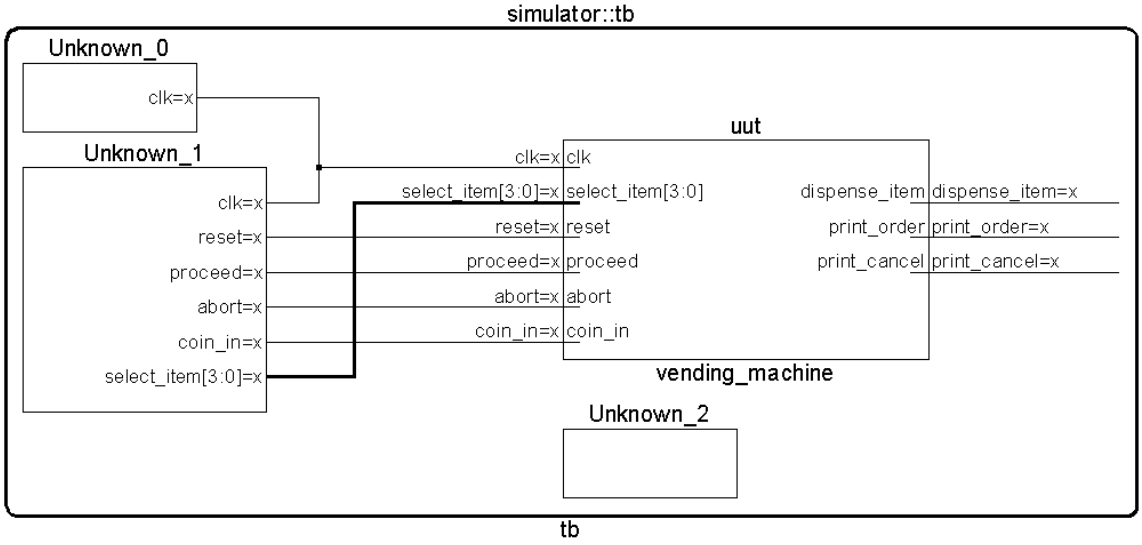
## Waveform



## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
Time = 0 | State = 0 | Coin = 0 | Item = 0000 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 20 | State = 0 | Coin = 1 | Item = 0000 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 25 | State = 1 | Coin = 1 | Item = 0000 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 30 | State = 1 | Coin = 0 | Item = 0010 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 35 | State = 10 | Coin = 0 | Item = 0010 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 40 | State = 10 | Coin = 0 | Item = 0010 | Proceed = 1 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 45 | State = 11 | Coin = 0 | Item = 0010 | Proceed = 1 | Abort = 0 | Dispense = 1 | Order = 1 | Cancel = 0
Time = 50 | State = 11 | Coin = 0 | Item = 0010 | Proceed = 0 | Abort = 0 | Dispense = 1 | Order = 1 | Cancel = 0
Time = 55 | State = 0 | Coin = 0 | Item = 0010 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 80 | State = 0 | Coin = 1 | Item = 0010 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 85 | State = 1 | Coin = 1 | Item = 0010 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 90 | State = 1 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 95 | State = 10 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 100 | State = 10 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 1 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 105 | State = 100 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 1 | Dispense = 0 | Order = 0 | Cancel = 1
Time = 110 | State = 100 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 1
Time = 115 | State = 0 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 140 | State = 0 | Coin = 1 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 145 | State = 1 | Coin = 1 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 150 | State = 1 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 155 | State = 10 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 160 | State = 10 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 1 | Dispense = 0 | Order = 0 | Cancel = 0
Time = 165 | State = 100 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 1 | Dispense = 0 | Order = 0 | Cancel = 1
Time = 170 | State = 100 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 1
Time = 175 | State = 0 | Coin = 0 | Item = 0011 | Proceed = 0 | Abort = 0 | Dispense = 0 | Order = 0 | Cancel = 0
Simulation complete via $finish(1) at time 190 NS + 0
./vend_tb.v:63 $finish;
ncsim>
```

## Schematic



2. **A small cooperation has 10 shares of stock, and each share entitles its owner to one vote at a stockholder's meeting. The 10 shares of stock are divided among 4 owners as follows:**
   **Mr. W: 1 share**
   **Mr. X: 2 shares**
   **Mr. Y: 3 shares**
   **Mr. Z: 4 shares**

   **Each of these persons has a switch to close when they vote for yes, and to open when the vote is no for their respective shares.**
   **Draw the logic circuit.**
   **Simulate using EDA tools.**

## Source Code

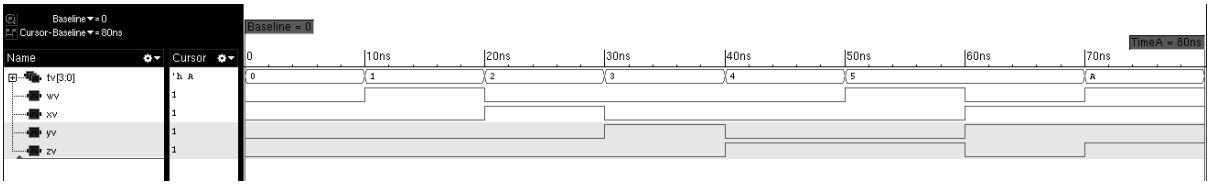
```
module vote (wv,xv,yv,zv,tv);
input wire wv;
input wire xv;
input wire yv;
input wire zv;
output wire [3:0] tv;
assign tv=(wv?1:0)+(xv?2:0)+(yv?3:0)+(zv?4:0);
endmodule
```

## Testbench

```
module tb();
reg wv,xv,yv,zv;
wire [3:0]tv;
vote uut (wv,xv,yv,zv,tv);
initial begin
wv= 0; xv= 0; yv = 0; zv = 0;#10;
wv= 1; xv= 0; yv = 0; zv = 0;#10;
wv= 0; xv= 1; yv = 0; zv = 0;#10;
wv= 0; xv= 0; yv = 1; zv = 0;#10;
wv= 0; xv= 0; yv = 0; zv = 1;#10;
wv= 1; xv= 0; yv = 0; zv = 1;#10;
wv= 0; xv= 1; yv = 1; zv = 0;#10;
wv= 1; xv= 1; yv = 1; zv = 1;#10;
$finish;
end
```

```
initial begin
$monitor("Time: %0t | wv: %b | xv: %b | yv: %b | zv: %b | tv: %d", $time,
wv,xv,yv,zv,tv);
end
endmodule
```

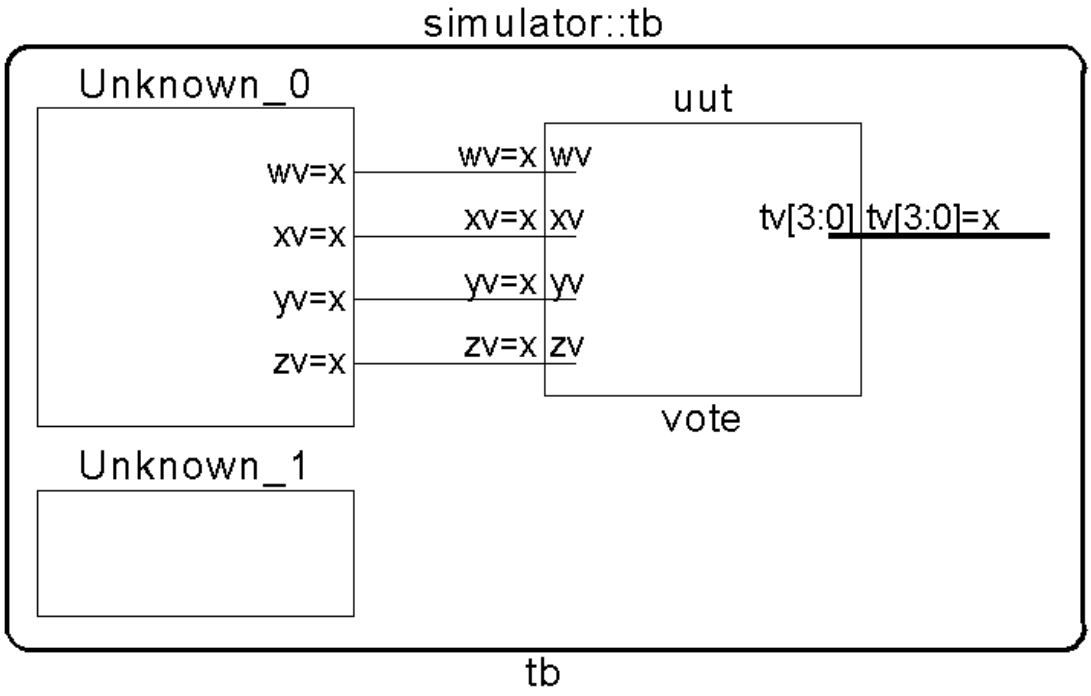
## Waveform




## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
Time: 0 | wv: 0 | xv: 0 | yv: 0 | zv: 0 | tv:  0
Time: 10 | wv: 1 | xv: 0 | yv: 0 | zv: 0 | tv:  1
Time: 20 | wv: 0 | xv: 1 | yv: 0 | zv: 0 | tv:  2
Time: 30 | wv: 0 | xv: 0 | yv: 1 | zv: 0 | tv:  3
Time: 40 | wv: 0 | xv: 0 | yv: 0 | zv: 1 | tv:  4
Time: 50 | wv: 1 | xv: 0 | yv: 0 | zv: 1 | tv:  5
Time: 60 | wv: 0 | xv: 1 | yv: 1 | zv: 0 | tv:  5
Time: 70 | wv: 1 | xv: 1 | yv: 1 | zv: 1 | tv: 10
Simulation complete via $finish(1) at time 80 NS + 0
./stocks_tb.v:14 $finish;
ncsim>
```

## Schematic



3. *Given:* **S is a sign bit where 0 indicates positive 1 and 0 indicates negative; exponent is a 7-bit excess 64 power of 2; mantissa is an 8-bit fraction.**
   *Problem:* **A 16-bit word, N=$4000_{16}$, represents what decimal numeric value?**

*Solution:*

$(4000)_{16} = (0100\ 0000\ 0000\ 0000)_2$

Sign 0 =positive

Mantissa = $(0000\ 0000)_2$

Excess 64 power of 2 exponent 1000000 – 1000000 = 0

Decimal value $0 \times 2^0 = 0$

**Source Code**

```
module decoder(input [15:0]N, output reg sign, output reg
[6:0]exponent,output reg [7:0] mantissa, output reg [15:0] result);
reg signed [7:0] true_exponent;
always @(*) begin
sign = N[15];
exponent = N[14:8];
mantissa = N[7:0];
true_exponent = exponent - 64;
if (mantissa == 0 && exponent == 64) begin
result = 1.0;
end else begin
result = (sign == 0) ? (1 + mantissa / 256.0) * (2.0 ** true_exponent) :
-(1 + mantissa / 256.0) * (2.0 ** true_exponent);
end
end
endmodule
```

**Testbench**

```
module tb();
reg [15:0] N;
wire sign;
wire [6:0] exponent;
wire [7:0] mantissa;
wire [15:0] result;
```

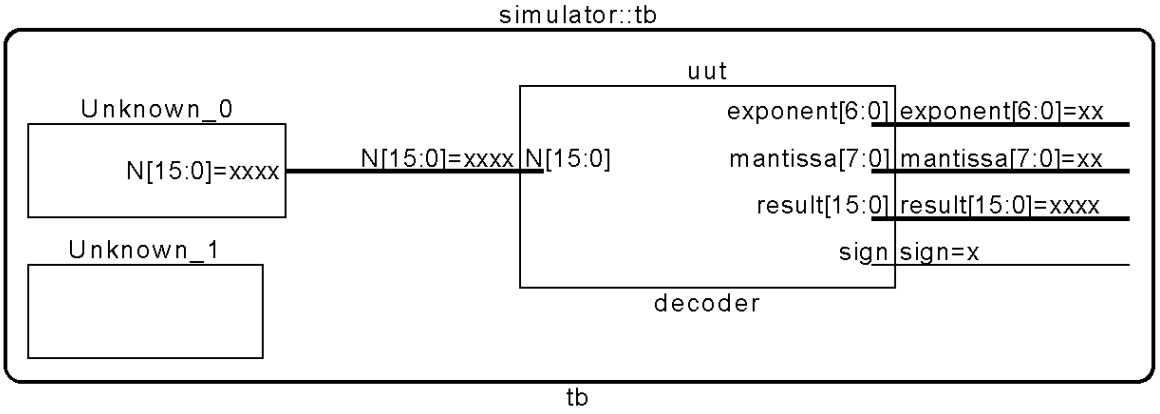
```
decoder uut (N,sign,exponent,mantissa,result);
initial begin
$monitor("Time: %0dns | N: %h | Sign: %b | Exponent: %d | Mantissa: %d |
Result: %f",$time, N, sign, exponent, mantissa, result);
N = 16'h4000;#10;
N = 16'hC100;#10;
N = 16'h3F00;#10;
N = 16'h4100;#10;
$finish;
end
initial begin
$dumpfile("decoder.vcd");
$dumpvars(0, tb);
end
endmodule
```

## Waveform


## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
Time: 0ns | N: 4000 | Sign: 0 | Exponent:  64 | Mantissa:   0 | Result: 1.000000
Time: 10ns | N: c100 | Sign: 1 | Exponent:  65 | Mantissa:   0 | Result: 65534.000000
Time: 20ns | N: 3f00 | Sign: 0 | Exponent:  63 | Mantissa:   0 | Result: 1.000000
Time: 30ns | N: 4100 | Sign: 0 | Exponent:  65 | Mantissa:   0 | Result: 2.000000
Simulation complete via $finish(1) at time 40 NS + 0
./float_tb.v:14 $finish;
ncsim>
```

## Schematic




4. *Given:* **S is a sign bit where 0 indicates positive 1 and 0 indicates negative; exponent is a 7-bit excess 64 power of 2; mantissa is an 8-bit fraction.**
   *Problem:* **Can the decimal number 111.875 be stored in this format? If not, what decimal values can be stored?**

*Solution*:

111.875 cannot be stored because the mantissa 0.875 requires more than 8 bits. However, 118.75 can be stored because this mantissa 0.75 requires only 7 bits, as shown in the following number conversions:

$0.875_{10}=0.3611_{16}=0.1101101011_2$:10 bits, $0.75_{10}=0.411_{16}=0.1001011_2$: 7 bits

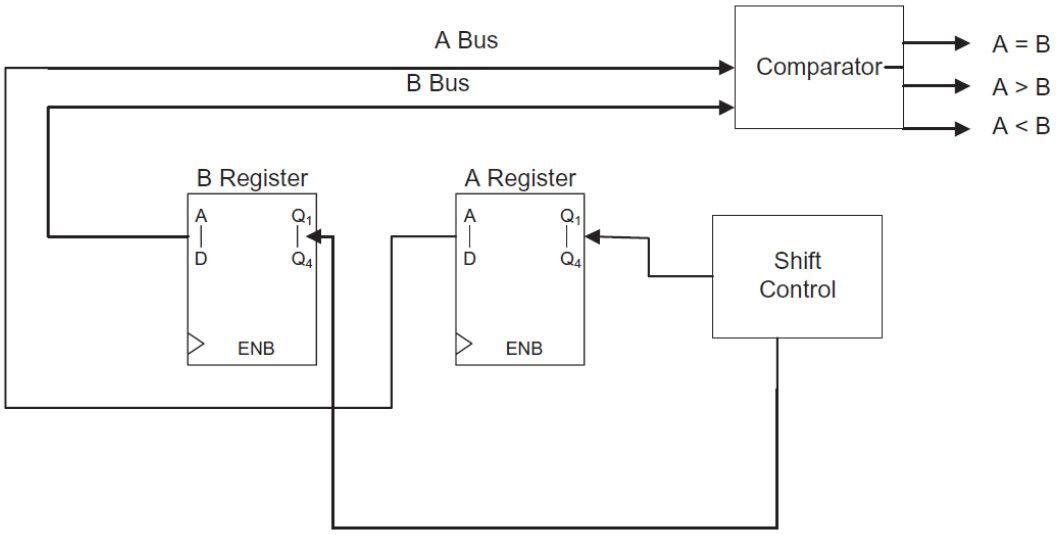
These conversions are achieved by successively dividing the decimal number by 16, recording the remainders, and assigning the remainders to the hexadecimal numbers in reverse order.

5. *Given:* **A central processing unit (CPU) is designed with one-hot encoding. This CPU cycles through 16 states and produces 32 control signals (one for flip -flop output *high* and one for output *low*, for each of 16 states).**
   *Problem:* **How many flip-flops are required?**

*Answer*: Sixteen are required, corresponding to each of the 16 states.

**6. How can a circuit be designed to analyze branch instruction logic in a program?**

*Solution:* Figure below shows how a comparator and its accompanying shift control logic can be designed to determine whether data originally in registers A and B, and then transferred to busses A and B, have the relationships A = B, A > B, or A < B. Thus, in a program, branch 1 would be taken if A = B; branch 2 would be taken if A > B; and branch 3 would be taken if A < B.




## Source Code

```
module branchanalyze(input clk,rst,en_A,en_B,shift_en,input [7:0]
A,B,output reg eq,gt,lt);

reg [7:0] reg_A_out;
reg [7:0] reg_B_out;
reg [7:0] comparator_A;
reg [7:0] comparator_B;

always @(posedge clk or negedge rst) begin
if (rst)
begin
reg_A_out <= 8'b0;
reg_B_out <= 8'b0;
end
else if (en_B)
```

```
reg_B_out <= B;
else if (en_A)
reg_A_out <= A;
end

always @(*) begin
if (shift_en) begin
comparator_A = reg_A_out;
comparator_B = reg_B_out;
end
else begin
comparator_A = reg_A_out;
comparator_B = reg_B_out;
end
end

always @(*) begin
if (comparator_A == comparator_B) begin
eq = 1;gt = 0;lt = 0;
end else if (comparator_A > comparator_B) begin
eq = 0;gt = 1;lt = 0;
end else begin
eq = 0;gt = 0;lt = 1;
end
end
endmodule
```

**Testbench**

```
module tb();
reg clk,rst,en_A,en_B,shift_en;
reg [7:0] A,B;
wire eq,gt,lt;

branchanalyze uut
(.clk(clk),.rst(rst),.en_A(en_A),.en_B(en_B),.shift_en(shift_en),.A(A),.B
(B),.eq(eq),.gt(gt),.lt(lt));
```

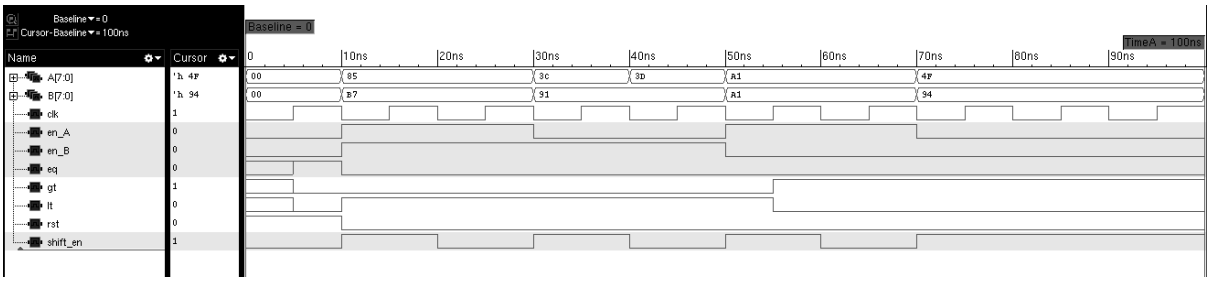
```
initial begin
clk=0; forever #5 clk=~clk;
end

initial begin
rst = 1;#10;
rst = 0;
end

initial begin
en_A=0;en_B=0;shift_en=0;A=8'b0000_0000;B=8'b0000_0000; #10;
en_A=1;en_B=1;shift_en=1;A=8'b1000_0101;B=8'b1011_0111; #10; //lt
en_A=1;en_B=1;shift_en=0;A=8'b1000_0101;B=8'b1011_0111; #10;
en_A=0;en_B=1;shift_en=1;A=8'b0011_1100;B=8'b1001_0001; #10; //gt
en_A=0;en_B=1;shift_en=0;A=8'b0011_1101;B=8'b1001_0001; #10;
en_A=1;en_B=0;shift_en=1;A=8'b1010_0001;B=8'b1010_0001; #10; //eq
en_A=1;en_B=0;shift_en=0;A=8'b1010_0001;B=8'b1010_0001; #10;
en_A=0;en_B=0;shift_en=1;A=8'b0100_1111;B=8'b1001_0100; #10;
en_A=0;en_B=0;shift_en=1;A=8'b0100_1111;B=8'b1001_0100; #10;
end


initial begin
$monitor("en_A=%b,en_B=%b,shift_en=%b,A=%b,B=%b,gt=%b,eq=%b,lt=%b",en_A,e
n_B,shift_en,A,B,gt,eq,lt);
#100 $finish;
end
endmodule
```

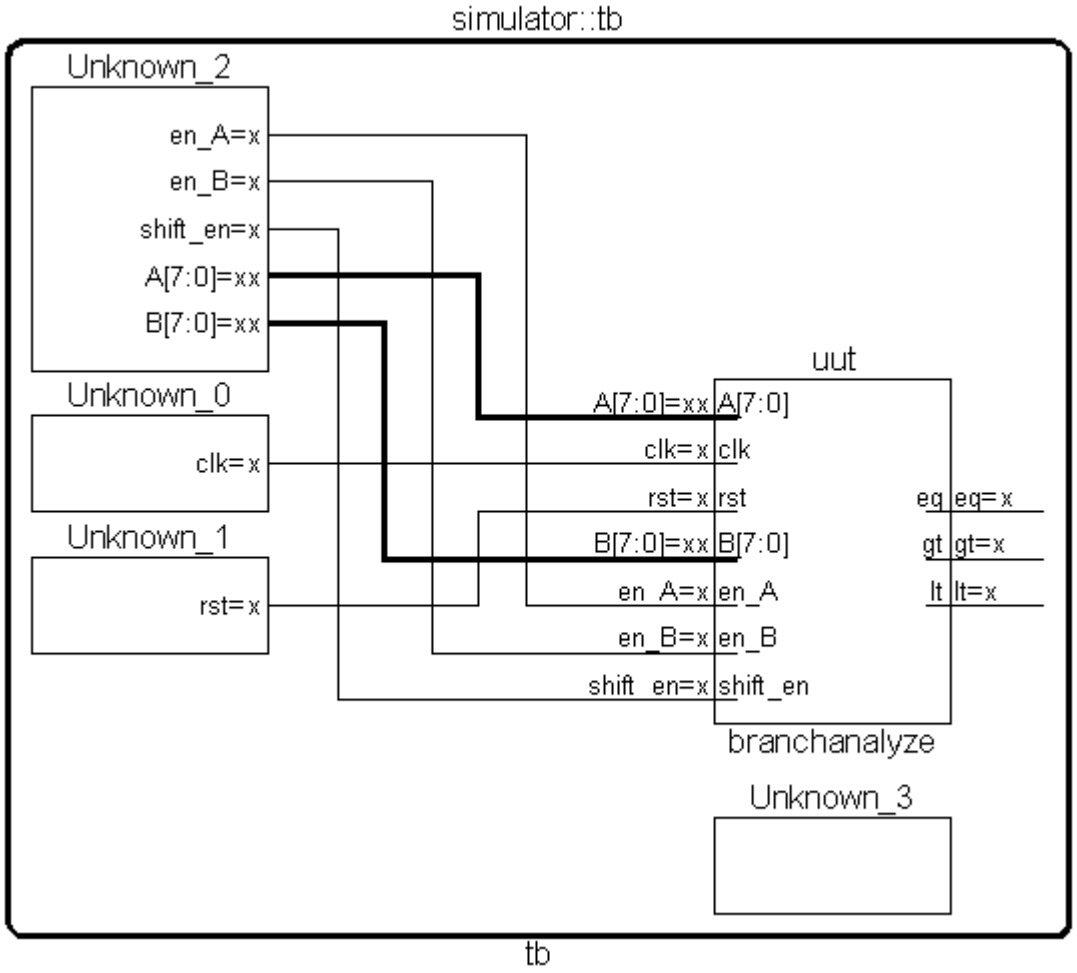
## Waveform



## Console

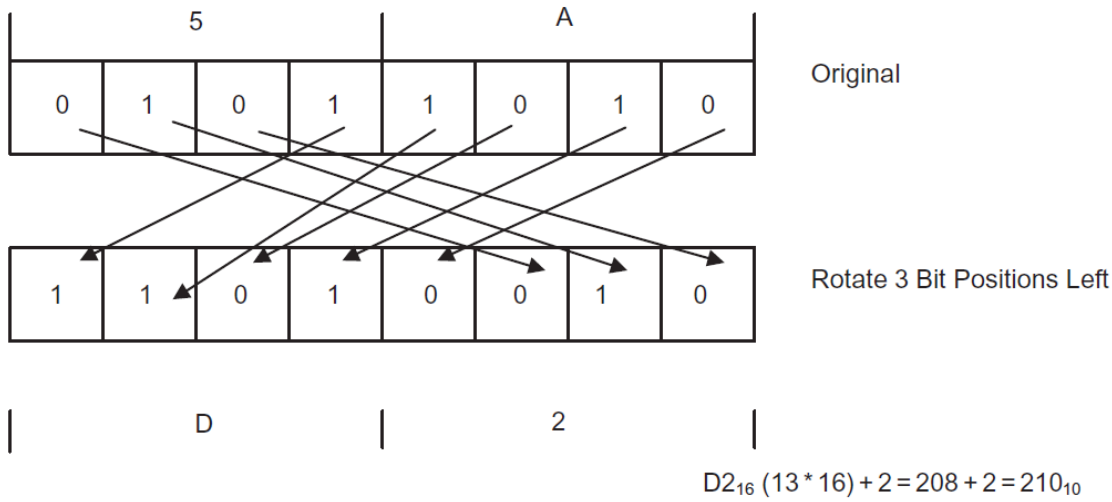
```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
en_A=0, en_B=0, shift_en=0, A=00000000, B=00000000, gt=x, eq=x, lt=x
en_A=0, en_B=0, shift_en=0, A=00000000, B=00000000, gt=0, eq=1, lt=0
en_A=1, en_B=1, shift_en=1, A=10000101, B=10110111, gt=0, eq=0, lt=1
en_A=1, en_B=1, shift_en=0, A=10000101, B=10110111, gt=0, eq=0, lt=1
en_A=0, en_B=1, shift_en=1, A=00111100, B=10010001, gt=0, eq=0, lt=1
en_A=0, en_B=1, shift_en=0, A=00111101, B=10010001, gt=0, eq=0, lt=1
en_A=1, en_B=0, shift_en=1, A=10100001, B=10100001, gt=0, eq=0, lt=1
en_A=1, en_B=0, shift_en=1, A=10100001, B=10100001, gt=1, eq=0, lt=0
en_A=1, en_B=0, shift_en=0, A=10100001, B=10100001, gt=1, eq=0, lt=0
en_A=0, en_B=0, shift_en=1, A=01001111, B=10010100, gt=1, eq=0, lt=0
Simulation complete via $finish(1) at time 100 NS + 0
./branch_tb.v:32 #100 $finish;
ncsim>
```

## Schematic



7. **The hexadecimal quantity 5A has experienced a rotated left shift of 3-bit positions. What is the resultant quantity in hexadecimal and decimal?**

*Solution:* The result is D2 in hexadecimal and 210 in decimal.



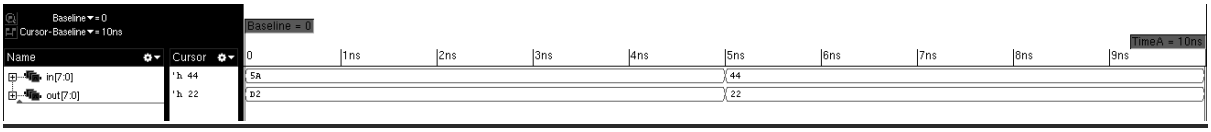

## Source Code

```
module rotateleft(out,in);
input [7:0]in;
output [7:0]out;
assign out={in[4:0],in[7:5]};
endmodule
```

## Testbench

```
module tb();
reg [7:0]in;
wire [7:0]out;
rotateleft uut(out,in);
initial begin
in=8'h5A;
#5 in=8'h44;
end
initial begin
$monitor($time,"in=%b,out=%b",in,out);
#10 $finish;
end
endmodule
```

## Waveform




## Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
                0in=01011010, out=11010010
                5in=01000100, out=00100010
Simulation complete via $finish(1) at time 10 NS + 0
./hex_tb.v:11 #10 $finish;
ncsim>
```

## Schematic