1. 4-bit FA

```
module adder_four_bit(
output [3:0]sum,
output cout,
input [3:0]a,b);

wire c1, c2, c3, c4;

full_3 ad0( .a(a[0]), .b(b[0]),.cin(0), .s(sum[0]), .cout(c1));
full_3 ad1( .a(a[1]), .b(b[1]),.cin(c1), .s(sum[1]), .cout(c2));
full_3 ad2( .a(a[2]), .b(b[2]),.cin(c2), .s(sum[2]), .cout(c3));
full_3 ad3( .a(a[3]), .b(b[3]),.cin(c3), .s(sum[3]), .cout(c4));
assign cout= c4;
endmodule

module full_3(a,b,cin,s,cout);
input a,b,cin;
output s, cout;
assign s=a^b^cin;
assign cout = (a&b) | (b&cin) | (cin&a);
endmodule
```

2. BCD to Excess3

```
//Implementaion of BCD to Excess3 by using DataFlow Modelling
module Bcd_excess3(b,e);
input [3:0] b;
output [3:0] e;
assign e[3]=b[3]|b[2]&b[1]|b[2]&b[0];
assign e[2]=~b[2]&b[1]|~b[2]&b[0]|b[2]&~b[1]&~b[0];
assign e[1]=b[1]&b[0]|~b[1]&~b[0];
assign e[0]=~b[0];
endmodule
```

3. Binary to Gray

```
//Verilog Code 4-bit Binary to Gray by using Data Flow Modelling
`timescale 1ns / 1ps
module Binary_to_Gray(
    input [3:0] b,
    output [3:0] g
    );
assign g[0]=b[1]^b[0];
assign g[1]=b[2]^b[1];
assign g[2]=b[3]^b[2];
assign g[3]=b[3];
endmodule
```

4. Decoder – 2:4

```
module decoder24_behaviour(en,a,b,y);
  // input port
  input en,a,b;

  // use reg to store the output value
  output reg [3:0]y;
  // always is used in design block
  // only in Behavioural modeling.

  always @(en,a,b)
    begin
      // using condition if statement
      // implement the 2:4 truth table
      if(en==0)
        begin
          if(a==1'b0 & b==1'b0) y=4'b1110;
          else if(a==1'b0 & b==1'b1) y=4'b1101;
          else if(a==1'b1 & b==1'b0) y=4'b1011;
          else if(a==1 & b==1) y=4'b0111;
          else y=4'bxxxx;
        end
      else
       y=4'b1111;
    end
endmodule
```

5. ALU

```
// Arithmetic and Logical Unit verilog code
module ALU(x,y,sel,z);
input [7:0]x,y;
output reg [15:0]z;
input [2:0]sel;
parameter ADD=3'b000;
parameter SUB=3'b001;
parameter MUL=3'b010;
parameter DIV=3'b011;
parameter AND=3'b100;
parameter OR=3'b101;
parameter NOT1 =3'b110;
parameter NOT2 =3'b111;
always@(*)
case(sel)
ADD: z=x+y;
SUB: z=x-y;
MUL: z=x*y;
DIV: z=x/y;
AND: z=x&y;
OR: z=x|y;
NOT1: z=!x;
NOT2: z=!y;
endcase
endmodule
```

6. SIPO

```
module sipo_shift_register_design(input clk,b,output[3:0]q);

d_ff dut1(.clk(clk),.d(b),.q(q[3]),.rst());
d_ff dut2(.clk(clk),.d(q[3]),.q(q[2]),.rst());
d_ff dut3(.clk(clk),.d(q[2]),.q(q[1]),.rst());
d_ff dut4(.clk(clk),.d(q[1]),.q(q[0]),.rst());

endmodule
// d flip flop

module d_ff (
  input clk,
  input d,
  input rst,
  output reg q);

  always @(posedge clk)
  begin
    if (rst)
      q <= 1'b0;
    else
      q <= d;
  end

endmodule
testbench
;()module sipo_tb

;reg clk,b
;wire [3:0]q

;sipo_shift_register_design uut(.clk(clk),.b(b),.q(q))

initial
begin
;clk=1'b0
;forever #5clk=~clk
end

initial
begin
```

```
;monitor("clk=%d,b=%d,q=%d",clk,b,q)$
end

initial
begin
;b=1
;10#
;b=0
;10#
;b=1
;10#
;b=0

;50#
;finish$

end
```

PISO

```
module piso_design(input clk,sl,input[3:0]b,output q);

wire w1,w2,w3,w4,w5,w6,w7;
wire o1,o2,o3;

d_ff d0(.clk(clk),.d(b[0]),.q(w1),.rst());

ao a1(.a(w1),.b(sl),.c(sl),.d(b[1]),.z(o1));
d_ff d1(.clk(clk),.d(o1),.q(w4),.rst());
```

```
ao a2(.a(w4),.b(sl),.c(sl),.d(b[2]),.z(o2));
d_ff d2(.clk(clk),.d(o2),.q(w7),.rst());

ao a3(.a(w7),.b(sl),.c(sl),.d(b[3]),.z(o3));
d_ff d3(.clk(clk),.d(o3),.q(q),.rst());
endmodule

// d flip flop
module d_ff (
  input clk,    // clock input
  input d,      // data input
  input rst,    // asynchronous reset input
  output reg q  // output
);

  always @(posedge clk)
  begin
    if (rst) // asynchronous reset
      q <= 1'b0;
    else // normal operation
      q <= d;
  end

endmodule

//test bench
;()module piso_tb
;reg [3:0]b
;reg clk,sl
;wire q
;piso_design dut(clk,sl,b,q)
initial begin
;clk=1'b0
;forever #5 clk=~clk
end
initial begin
;sl=0;b=4'b0101
;sl=1 20#
;sl=1 20#
;sl=0 10#
;sl=0;//b=4'b0110 10#
```

```
;finish$ 100#
end
endmodule
```

7. Modulus Counter(Mod10)

```
module counter_mod6 ( clk ,reset ,dout );

output [2:0] dout ;
reg [2:0] dout ;

input clk ;
wire clk ;
input reset ;
wire reset ;

initial dout = 0;

always @ (posedge (clk)) begin
 if (reset)
  dout <= 0;
 else if (dout<5)
  dout <= dout + 1;
 else
  dout <= 0;
end

endmodule
//testbench
  ;module tb
  ;parameter N = 10
  ;parameter WIDTH = 4

  ;reg clk
  ;reg rstn
  ;wire [WIDTH-1:0] out
```

```
  ,modN_ctr u0  (    .clk(clk)
  ,rstn(rstn).
  ;(out(out).

  ;always #10 clk = ~clk

  initial begin
  ;{clk, rstn} <= 0

  ;monitor ("T=%0t rstn=%0b out=0x%0h", $time, rstn, out)$
  ;repeat(2) @ (posedge clk)
  ;rstn <= 1

  ;repeat(20) @ (posedge clk)
  ;finish$
  end
endmodule
```