Buffer verilog program

```
module buffer_code(x,z);
output z;
input x;
assign z = x ;
endmodule
```

Buffer test bench program

```
module buffer_test(x,z);
input z;
output x;
reg x;

buffer_code a(x,z);
initial
 begin
     x=1'b0;
     #05;
     x=1'b1;
     #05;
 end
endmodule
```

Inverter verilog program

```
module inv_code(x,z);
input x;
output z;
assign z= ~(x);
endmodule
```

Inverter test bench program

```
module inv_test(x,z);
input z;
output x;
reg x;

inv_code y(x,z);
initial
 begin
      x=1'b0;
      #05;
      x=1'b1;
      #05;
 end
endmodule
```

Transmission Gates verilog program

```
module trangate (out,in,cntrl1,cntrl2);
output out;
input in;
input cntrl1,cntrl2;
pmos (out,in cntrl1);
nmos (out,in,cntrl2);
endmodule
```

Transmission Gates Test bench program

```
module trangate_test;
wire out;
reg in;
reg cntrl1,cntrl2;

trangate t1 (out,in,cntrl1,cntrl2);
task display;
begin
$display (
       "time=%0d" $time, "ns"
        "Input=", in, "Output=", out
        , "Control1=",cntrl1,control2=",cntrl2
       ),
end
endtask

initial
begin
in = 1'b0' contrl1 =1'b0; cntrl2 = 1'b1  ;#10; display ;
in = 1'b0' contrl1 =1'b1; cntrl2 = 1'b0  ;#10; display ;
in = 1'b1' contrl1 =1'b0; cntrl2 = 1'b1  ;#10; display ;
in = 1'b1' contrl1 =1'b1; cntrl2 = 1'b0  ;#10; display ;

end
endmodule
```

AND gate verilog program

```
module andecode(a,b,y);
input a,b;
output y;
assign y=a & b;
endmodule
```

And gate test bench program

```
module andetb(a,b,y);
input y;
output a,b;
reg a,b;

andecode x(a,b,y);
initial
 begin
      a=1'b0;
      b=1’b0;
      #05;
      b=1'b1;
      #05;
      a=1'b1;
      b=1'b0;
      #05;
      b=1'b1;
      #05;

 end
endmodule
```

OR gate verilog program

```
module orecode(a,b,y);
input a,b;
output y;
assign y= a | b;
endmodule
```

Or gate test bench program

```
module oretb(a,b,y);
input y;
output a,b;
reg a,b;

orecode x(a,b,y);
initial
 begin
     a=1'b0;
     b=1’b0;
     #05;
     b=1'b1;
     #05;
     a=1'b1;
     b=1'b0;
     #05;
     b=1'b1;
     #05;
 end
endmodule
```

NOR gate verilog program

```
module norecode(a,b,y);
input a,b;
output y;
assign y= ~(a | b);
endmodule
```

NOR gate test bench program

```
module noretb(a,b,y);
input y;
output a,b;
reg a,b;

norecode n(a,b,y);
initial
 begin
      a=1'b0;
      b=1’b0;
      #05;
      b=1'b1;
      #05;
      a=1'b1;
      b=1'b0;
      #05;
      b=1'b1;
      #05;
 end
endmodule
```

NAND gate verilog program

```
module nandecode(a,b,y);
input a,b;
output y;
assign y= ~(a & b);
endmodule
```

NAND gate test bench program

```
module nandetb(a,b,y);
input y;
output a,b;
reg a,b;

nandecode n(a,b,y);
initial
 begin
     a=1'b0;
     b=1'b0;
     #05;
     b=1'b1;
     #05;
     a=1'b1;
     b=1'b0;
     #05;
     b=1'b1;
     #05;
 end
endmodule
```

XOR gate verilog program

```
module xorecode(a,b,y);
input a,b;
output y;
assign y= (a ^ b);
endmodule
```

XOR gate test bench program

```
module xoretb(a,b,y);
input y;
output a,b;
reg a,b;

xorecode n(a,b,y);
initial
 begin
      a=1'b0;
      b=1'b0;
      #05;
      b=1'b1;
      #05;
      a=1'b1;
      b=1'b0;
      #05;
      b=1'b1;
      #05;
 end
endmodule
```

XNOR gate verilog program

```
module xorecode(a,b,y);
input a,b;
output y;
assign y= ~(a ^ b);
endmodule
```

XNOR gate test bench program

```
module xoretb(a,b,y);
input y;
output a,b;
reg a,b;

xorecode n(a,b,y);
initial
 begin
      a=1'b0;
      b=1’b0;
      #05;
      b=1'b1;
      #05;
      a=1'b1;
      b=1'b0;
      #05;
      b=1'b1;
      #05;
 end
endmodule
```

NOT verilog program

```
module note_code(x,z);
input x;
output z;
assign z= ~(x);
endmodule
```

NOT test bench program

```
module note_test(x,z);
input z;
output x;
reg x;

inv_code y(x,z);
initial
 begin
      x=1'b0;
      #05;
      x=1'b1;
      #05;
 end
endmodule
```

Flip Flops

SR Flip flop verilog program

```
module srff(q,qbar,s,r,clk);
output q,qbar;
input clk,s,r;
reg tq;
always @(posedge clk)
begin
 if(s==1'b0 && r==1'b0)
      tq<=tq;
 else if(s==1'b0 && r==1'b1)
      tq<=1'b0;

 else if(s==1'b1 && r==1'b0)
      tq<=1'b1;
 else if(s==1'b1 && r==1'b1)
      tq<=1'bx;
end
 assign q=tq;
 assign qbar=~tq;
endmodule
```

SR Flip flop test bench program

```
module srff_test;
reg clk,s,r;
wire q,qbar;
wire s1,r1,clk1;
srff sr1(q,qbar,s,r,clk);
 assign s1=s;
 assign r1=r;
 assign clk1=clk;
initial
 clk=1'b0;
 always
 #10clk=~clk;
initial
begin
```

```
 s=1'b0;r=1'b0;
 #30s=1'b1;
 #29s=1'b0;
 #1r=1'b1;
 #30s=1'b1;
 #30r=1'b0;
 #20s=1'b0;
 #19s=1'b1;
 #200s=1'b1;r=1'b1;
 #50s=1'b0;r=1'b0;
 #50s=1'b1;r=1'b0;
 #10;
end
always
 #5 $display($time,"clk=%b s=%b r=%b",clk,s,r);
initial
 #500 $finish;
specify
$setup(s1,posedge clk1,2);
$setup(r1,posedge clk1,2);
$hold(posedge clk1,s1,2);
$hold(posedge clk1,r1,2);
endspecify
endmodule
```

T Flip flop verilog program

```
module t_ff(q,qbar,clk,tin,rst);
output q,qbar;
input clk,tin,rst;
reg tq;
always@(posedge clk or negedge rst)
begin
 if(!rst)
      tq<=1'b0;
 else
 begin
 if(tin)
      tq<=~tq;
 end
end
 assign q=tq;
 assign qbar=~q;
endmodule
```

T Flip flop test bench program

```
module tff_test;
reg clk,tin,rst;
wire q,qbar;
t_ff t1(q,qbar,clk,tin,rst);
initial
 clk=1'b0;
always
 #10 clk=~clk;
initial
begin
 rst=1'b0; tin=1'b0;
 #30 rst=1'b1;
 #10 tin=1'b1;
 #205 tin=1'b0;
 #300 tin=1'b1;
 #175 tin=1'b0;
 #280 rst=1'b0;
 #20 rst=1'b1;
 #280 tin=1'b1;
 #10;
end
initial
 #2000 $finish;
endmodule
```

D Flip flop verilog program

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

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

            else
            q <= din;
end
endmodule
```

D Flip flop test bench program

```
module d_ff_test;
reg clk, din, rst;
wire q, d1, clk1;
d_ff df1 (q, clk, rst, din);
assign d1=din;
assign clk1=clk;
initial
clk = 1'b0;

always
#10 clk = ~clk;

initial
begin
            din = 1'b0;
            rst = 1'b1;
            #20 rst = 1'b0;
            #10 din = 1'b1;
            #20 rst = 1'b1;
            #18 din = 1'b0;
```

```
                    #1 din = 1'b1;
                    #20 din = 1'b0;
                    #10 ;
end


always
#5 $display ($time," clk=%b  din=%b   q=%b", clk1,
din, q);

initial
                    #100 $finish;

specify
                    $setup(d1, posedge clk1, 2);
                    $hold(posedge clk1, d1, 2);
                    $width(negedge d1, 2);
endspecify

endmodule
```

JK Flip flop verilog program

```
module jk_ff(q,qbar,clk,rst,j,k);

input clk,rst,j,k;
output q,qbar;
reg q,tq;

always @(posedge clk or negedge rst)
begin
            if (!rst)
            begin
            q <= 1'b0;
            tq <= 1'b0;
            end

            else
            begin
            if (j == 1'b1 && k == 1'b0)
            q <= j;

            else if (j == 1'b0 && k == 1'b1)
            q <= 1'b0;

            else if (j == 1'b1 && k == 1'b1)
            begin
            tq <= ~tq;
            q <= tq;
            end
            end
            end
            assign qbar = ~q;
endmodule
```

JK Flip flop test bench program

```
module jak_ff_test;
reg clk,rst,j,k;
wire q,qbar;
wire clk1,j1,k1;
jk_ff inst(q,qbar,clk,rst,j,k);

            assign clk1=clk;
            assign j1=j;
            assign k1=k;
initial
clk = 1'b0;
always
#10 clk = ~clk;
initial
begin
j = 1'b0; k = 1'b0; rst = 1'b0;
#30 rst = 1'b1;
#60 j = 1'b0; k = 1'b1;
#29 j = 1'b1; k = 1'b0;
#1 j = 1'b0; k = 1'b1;
#20 j = 1'b1; k = 1'b1;
#40 j = 1'b1; k = 1'b0;
#5 j = 1'b0; #20 j = 1'b1;
#50 rst = 1'b0;
#10 ;
end

always
#5 $display($time,"  clk=%b  j=%b  k=%b  ",clk,j,k);

initial
#300 $finish;

specify
            $setup(j1, posedge clk1, 2);
```

```
                  $setup(k1, posedge clk1, 2);
                  $hold(posedge clk1, j1, 2);
                  $hold(posedge clk1, k1, 2);
Endspecify

endmodule
```

MS Flip flop verilog program

```
module ms_jkff(q,q_bar,clk,j,k);

output q,q_bar;
input clk,j,k;
reg tq,q,q_bar;

always @(clk)
begin
     if (!clk)
begin
     if (j==1'b0 && k==1'b1)
              tq <= 1'b0;
     else if (j==1'b1 && k==1'b0)
              tq <= 1'b1;
     else if (j==1'b1 && k==1'b1)
              tq <= ~tq;
     end
     if (clk)
     begin
          q <= tq;
          q_bar <= ~tq;
     end
 end
endmodule
```

MS Flip flop test bench program

```
module tb_ms_jkff;

reg clk,j,k;
wire q,q_bar;
wire clk2,j2,k2;

ms_jkff inst(q,q_bar,clk,j,k);

assign clk2=clk;
assign j2=j;
assign k2=k;

initial
       clk = 1'b0;

always #10
       clk = ~clk;

initial
       begin
       j = 1'b0; k = 1'b0;
       #60 j = 1'b0; k = 1'b1;
       #40 j = 1'b1; k = 1'b0;
       #20 j = 1'b1; k = 1'b1;
       #40 j = 1'b1; k = 1'b0;
       #5 j = 1'b0; #20 j = 1'b1;
       #10 ;
       end

 always
 #5 $display($time,"  clk=%b  j=%b  k=%b  ",clk,j,k);

 initial
 #200 $finish;


 Specify
         $setup(j2, posedge clk2, 2);
         $setup(k2, posedge clk2, 2);
         $hold(posedge clk2, j2, 2);
         $hold(posedge clk2, k2, 2);
 endspecify

endmodule
```

Parallel Adder verilog program

```
module fulladd (cin,x,y,s,cout);

  input cin,x,y;
  output s,cout;

  assign s = x^y^cin;
  assign cout =( x & y) | (x & cin) |( y & cin);

endmodule


module adder4 ( carryin,x,y,sum,carryout);

  input carryin;
  input [3:0] x,y;
  output [3:0] sum;
  output carryout;

  fulladd stage0 (carryin,x[0],y[0],sum[0],c1);
  fulladd stage1 (c1,x[1],y[1],sum[1],c2);
  fulladd stage2 (c2,x[2],y[2],sum[2],c3);
  fulladd stage3 (c3,x[3],y[3],sum[3],carryout);

endmodule
```

Parallel Adder test bench program

```
module adder4_t ;

reg [3:0] x,y;
reg carryin;
wire [3:0] sum;
wire carryout;

adder4 a1 ( carryin,x,y,sum,carryout);

initial
 begin
  $monitor($time,"SUM=%d",sum);
  x = 4'b0000; y= 4'b0000;carryin = 1'b0;
  #20 x =4'b1111; y = 4'b1010;
  #40 x =4'b1011; y =4'b0110;
  #40 x =4'b1111; y=4'b1111;

 #50 $finish;
end

endmodule
```

Counter verilog program

```
module counter_behav ( count,reset,clk);
 input wire reset, clk;
 output reg [3:0] count;

 always @(posedge clk)
  if (reset)
   count <= 4'b0000;
  else
   count <= count + 4'b0001;

 endmodule
```

Counter Test bench Program

```
module mycounter_t ;
wire [3:0] count;
reg reset,clk;

initial
clk = 1'b0;

always
 #5 clk = ~clk;

counter_behav m1 ( count,reset,clk);

initial
begin
  reset = 1'b0 ;

 #15 reset =1'b1;
 #30 reset =1'b0;

 #300 $finish;

end

initial
$monitor ($time,  "Output count = %d ",count );

endmodule
```

Ripple counter verilog program

```
module ripple_counter (clock, toggle, reset, count);
   input clock, toggle, reset;
   output [3:0] count;
   reg [3:0] count;
   wire c0, c1, c2;
   assign c0 = count[0], c1 = count[1], c2 = count[2];

    always @ (posedge reset or posedge clock)
      if (reset == 1'b1) count[0] <= 1'b0;
     else if (toggle == 1'b1) count[0] <= ~count[0];

   always @ (posedge reset or negedge c0)
      if (reset == 1'b1) count[1] <= 1'b0;
      else if (toggle == 1'b1) count[1] <= ~count[1];

   always @ (posedge reset or negedge c1)
      if (reset == 1'b1) count[2] <= 1'b0;
      else if (toggle == 1'b1) count[2] <= ~count[2];

   always @ (posedge reset or negedge c2)
      if (reset == 1'b1) count[3] <= 1'b0;
      else if (toggle == 1'b1) count[3] <= ~count[3];
endmodule
```

Ripple counter testbench program

```
module ripple_counter_t ;
reg clock,toggle,reset;
wire [3:0] count ;

ripple_counter r1 (clock,toggle,reset,count);

initial
    clock = 1'b0;

always
    #5 clock = ~clock;

initial
begin
  reset = 1'b0;toggle = 1'b0;
  #10 reset = 1'b1; toggle = 1'b1;
  #10 reset = 1'b0;
  #190 reset = 1'b1;
  #20 reset = 1'b0;
  #100 reset = 1'b1;
  #40 reset = 1'b0;
  #250 $finish;

end

initial
    $monitor ($time, " output q = %d", count);

endmodule
```

Successive approximation Register verilog program

```
Module shiftrne (R,L,E,w,clock,q);
Parameter n=8;
input [n-1:0] R;
input L,W,clock;
output [n -1:0] q;
reg [n-1:0]q;
integer k;

always@(oosedge clock)
if(L)
  q=<=R;
 else if (E)
  begin
       for(k=n-1;k>0;k=l-1)
           q[K-1] <=q[k];
           q[n-1] <=w;
       end
endmodule
```

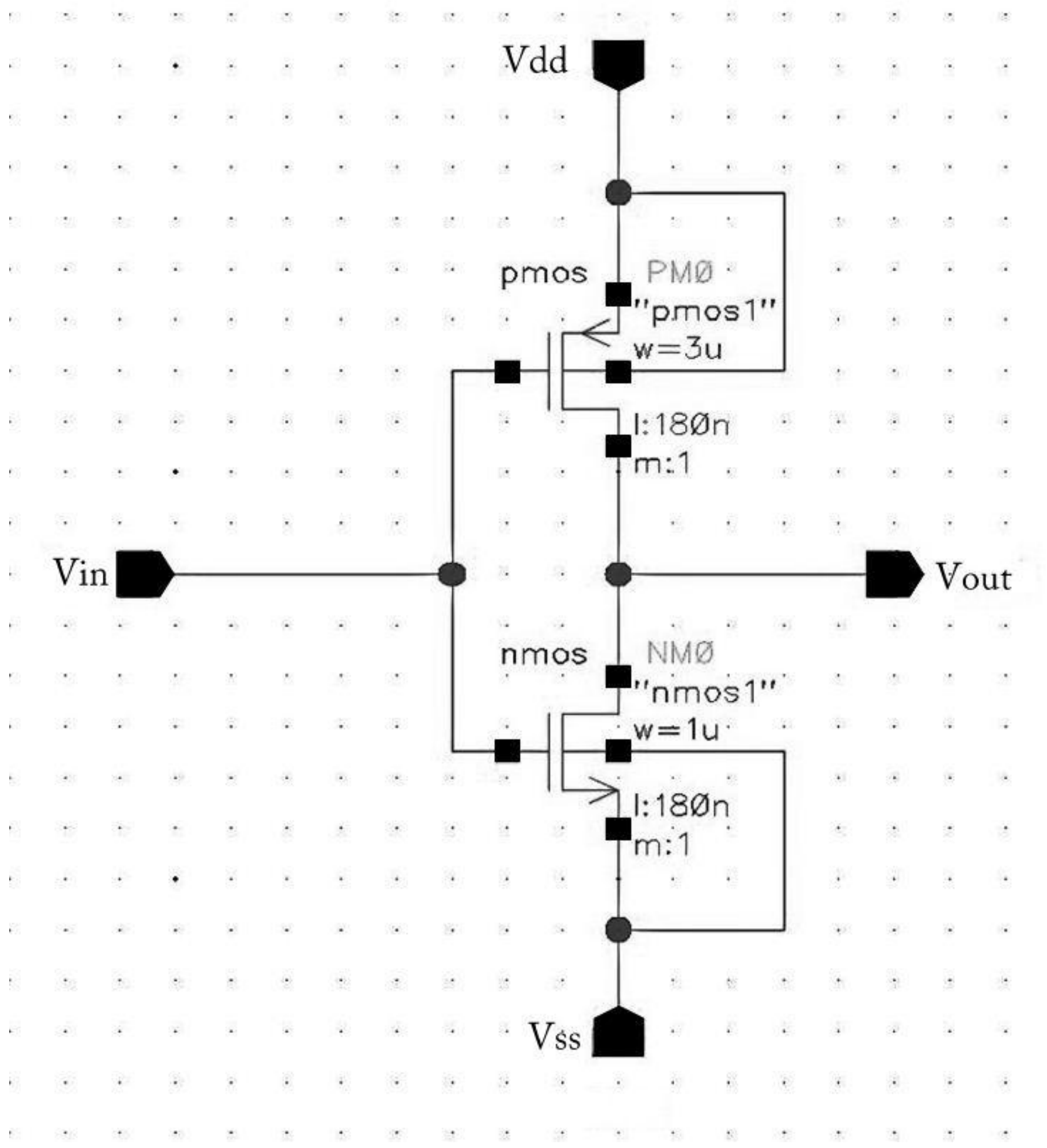
Successive approximation Register Test bench program

```
module sar_t;

reg [7:0] r;
reg 1;
reg e;
reg w;
regclk;

wire [7:0) q;

shiftrne sf(  .R(r),.L(1),.E(e),.W(w),.clock(clk),.q(q));
initial
  begin
   clk = 1'b0;
     l = 1'b1;
     w = 1'b0;
     e = 1'b0;
#S   r = 8'b1111_0000;
#10  l = 1'b0;
     e = 1'b1;
     w = 1'b0'
#10  w = 1'b0;
#10  w = 1'b0;
#10  w = 1'b0;
#10  w = 1'b1;
#10  w = 1'b1;
#10  w = 1'b1;
#10  w = 1'b1;
#10  w = 1'b1;
#10  $finish;
```

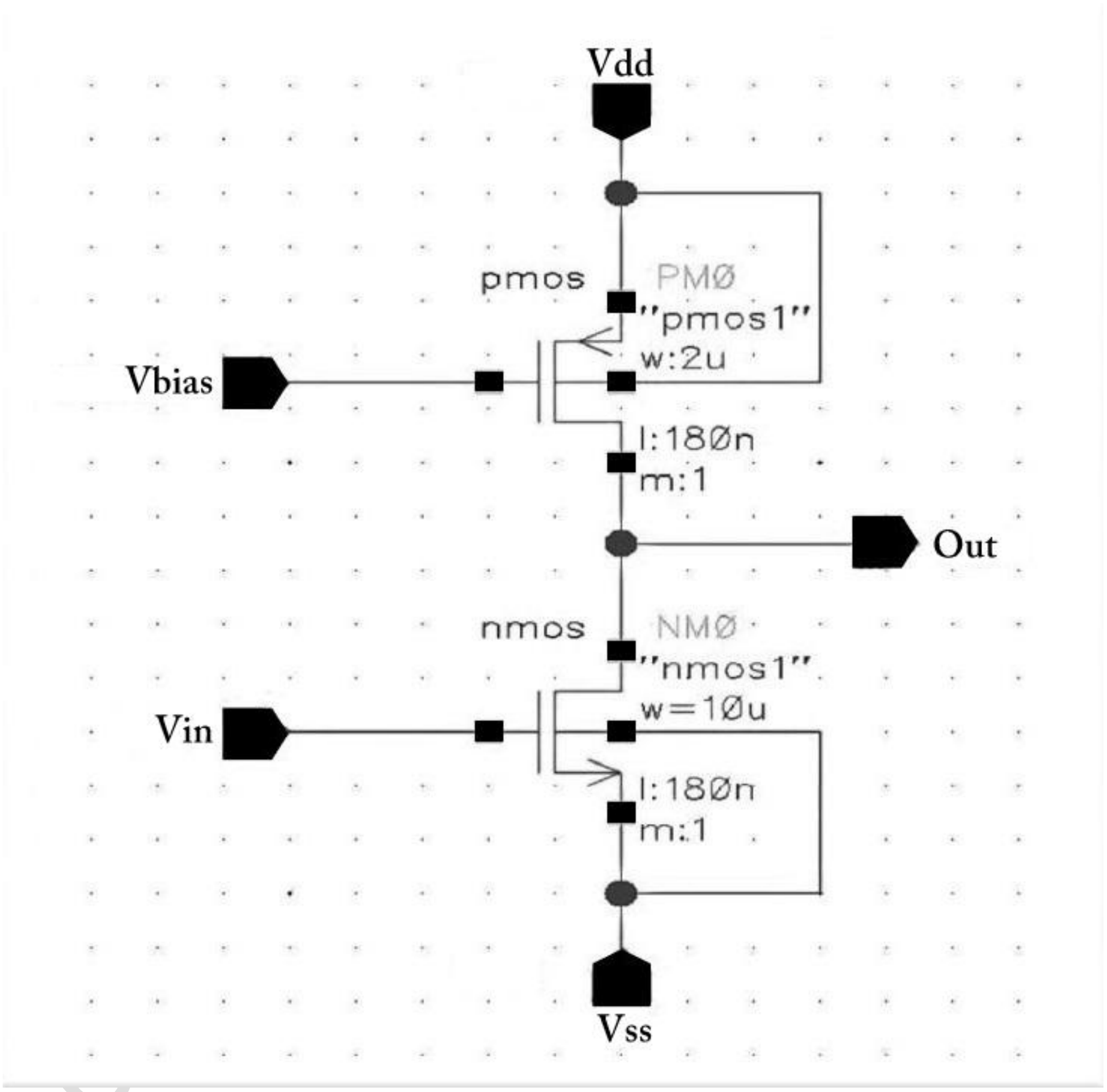
```
end
 always#5 clk = _clk;

endmodule
```

PART-B

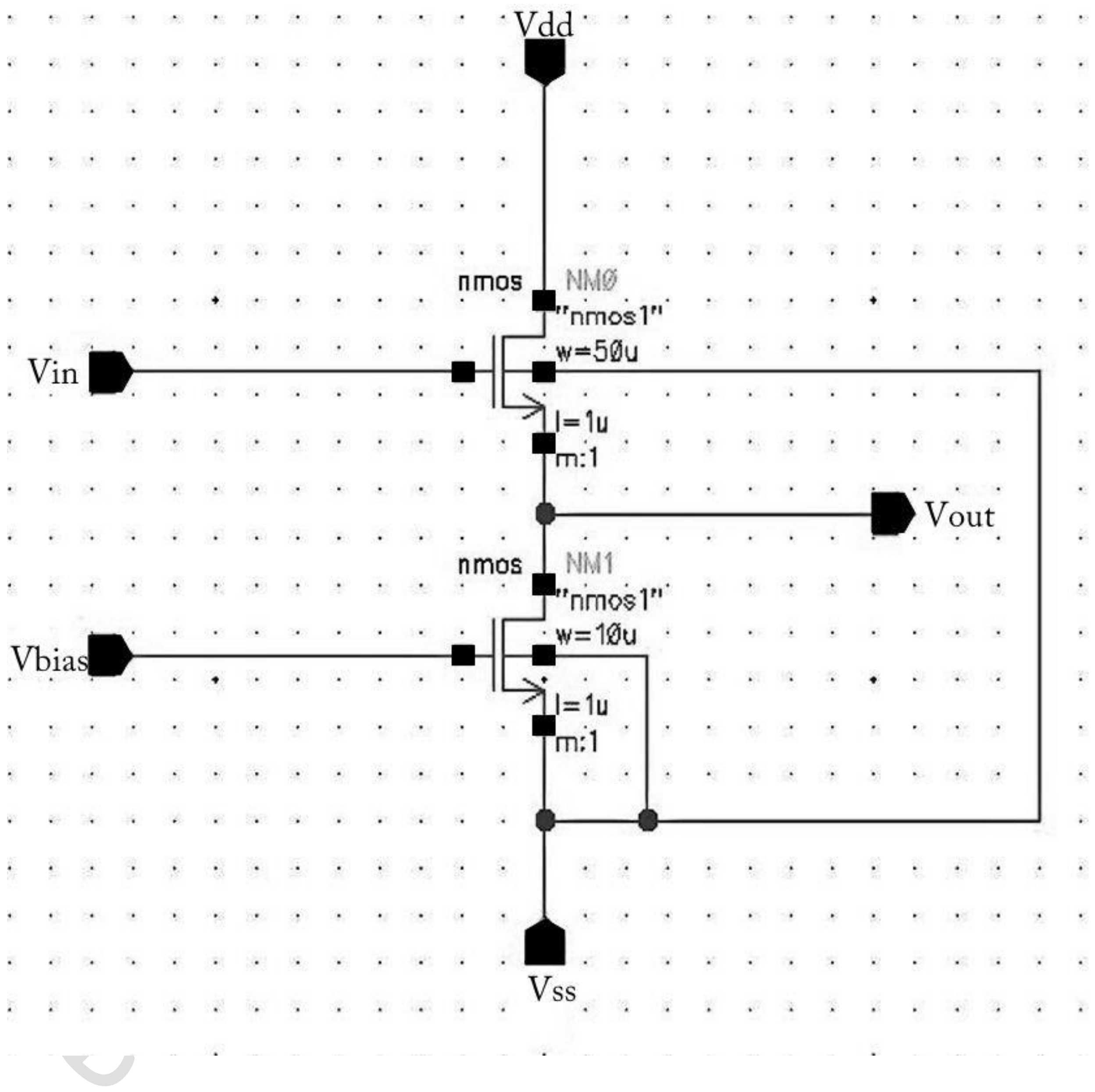
Inverter schematic



Common Source Amplifier Schematic



Common Drain Amplifier Schematic



Part B Important Points to remember

1. Inverter

Library name	Cell Name	Properties
gpdk180	pmos	Total Width= **wp**, Length=180n
gpdk180	nmos	Total Width = 1u, Length=180n

Pin Names	Direction
in, Vdd, Vss	input
out	output

Library name	Cellview name	Properties
MyDesignLib	Inverter	Symbol
analogLib	Vpulse (as input signal)	voltage1=0, voltage2=1.8,period =20n, pulse width= 10n.
analogLib	vdc	DC voltage =1.8
analogLib	gnd	

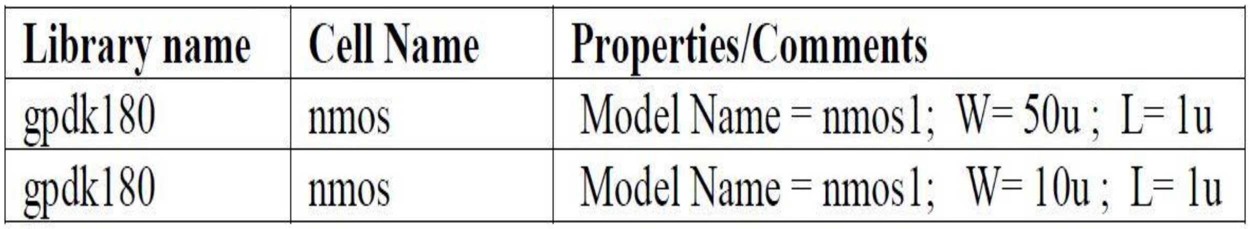
2. CS Amplifier

Library name	Cell Name	Properties
gpdk180	pmos	Total Width= 2u, Length=180n
gpdk180	nmos	Total Width = 10u, Length=180n

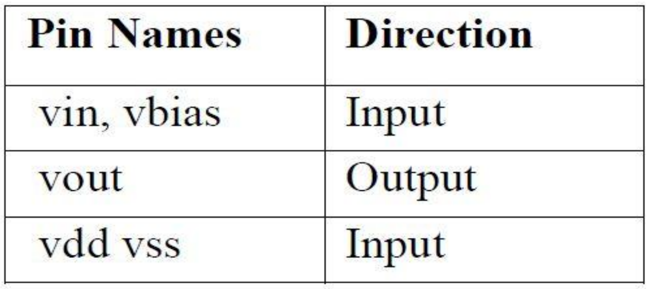
Pin Names	Direction
Vin, Vbais, Vdd, Vss	input
Out	output

Library name	Cellview name	Properties
MyDesignLib	cs_amp	Symbol
analogLib	Vsin (for Vin)	AC Magnitude= 1; DC Voltage= -1.8; Offset Voltage= 0; Amplitude= 5m; Frequency= 1K
analogLib	Vdc	DC Voltage = 2.5 (for Vdd) DC Voltage = -2.5 (for Vss) DC Voltage = 1 (for Vbias)
analogLib	gnd	

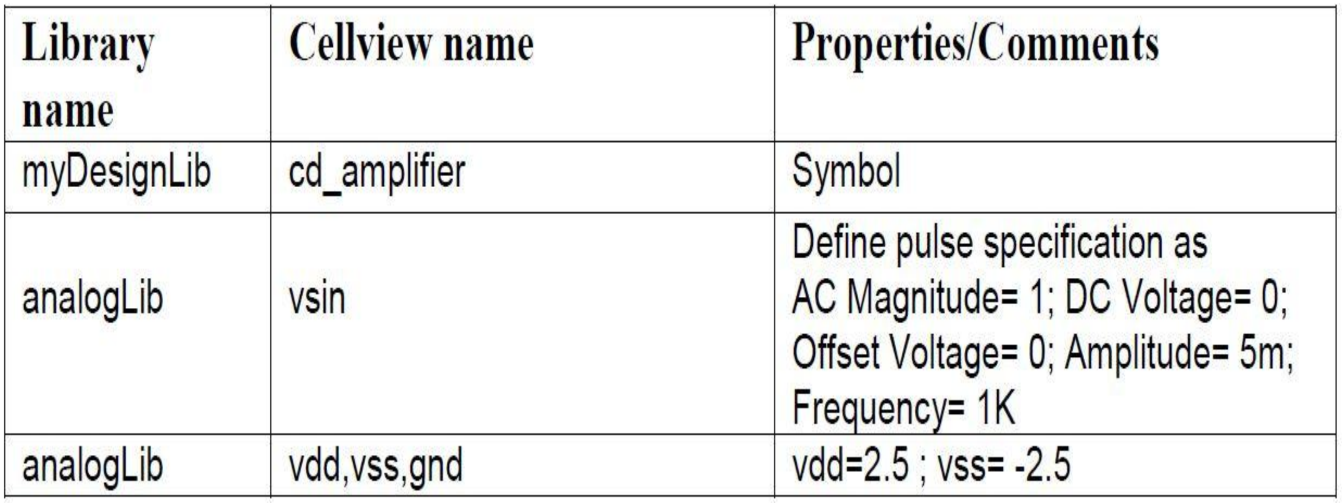
3. CD Amplifier



Library name	Cell Name	Properties/Comments
gpdk180	nmos	Model Name = nmos1; W= 50u ; L= 1u
gpdk180	nmos	Model Name = nmos1; W= 10u ; L= 1u



Pin Names	Direction
vin, vbias	Input
vout	Output
vdd vss	Input



Library name	Cellview name	Properties/Comments
myDesignLib	cd_amplifier	Symbol
analogLib	vsin	Define pulse specification as AC Magnitude= 1; DC Voltage= 0; Offset Voltage= 0; Amplitude= 5m; Frequency= 1K
analogLib	vdd,vss,gnd	vdd=2.5 ; vss= -2.5

Commands for Flipflops (with clk devices)

```
set_attribute  lib_search_path  /home/username/library

set_attribute  library slow_normal.lib

set_attribute  hdl_search_path  /home/ username/rtl

read_hdl filename.v

elaborate  module name

dc::set_clock_transition -rise 0.1 [dc::get_clocks "clk"]

dc::set_clock_transition -fall 0.1 [dc::get_clocks "clk"]

dc::set_clock_uncertainty 1.0 [dc::get_ports "clk"]

synthesize  -to_mapped  -effort  medium
```

Write_hdl > filename_netlist.v

Commands for basic gate(without clk devices)

```
set_attribute  lib_search_path  /home/username/library

set_attribute  library slow_normal.lib

set_attribute  hdl_search_path  /home/ username/rtl

read_hdl filename.v

elaborate  module name

synthesize  -to_mapped  -effort  medium
```

Report_timing

Report_area

Report_power

Write_hdl filename_netlist.v

Write_sdc filename.sdc