1.read_libs/home/install/FOUNDRY/digital/90nm/dig/lib/slow.lib
2. read_hdl counter.v
3. elaborate
4. report_timing > counter_timing_bs.rep
5. report_area > counter_area_bs.rep
6. report_power > counter_power_bs.rep
7. read_sdc counter.sdc
8. set_db syn_generic_effort medium
9. set_db syn_map_effort medium
10. set_db syn_opt_effort medium
11. syn_generic
12. syn_map
13. syn_opt
14. report_timing > counter_timing_as.rep
15. report_area > counter_area_as.rep
16. report_power > counter_power_as.rep
17. write_hdl > counter_netlist.v
18. write_sdc > counter_sdc.sdc

3. 32 Bit ALU

Source Code (using Case Statement)

```
module alu_32bit_case(y,a,b,f);
input [31:0]a,b;
input [2:0]f;
output reg [32:0]y;
always@(*)
begin
case(f)
3'b000: y=a&b; //AND Opera□on
3'b001: y=a|b; //OR Opera□on
3'b010: y=~(a&b); //NAND Opera□on
3'b011: y=~(a|b); //NOR Opera□on
3'b010: y=a+b; //Addi□on
3'b011: y=a-b; //Subtrac□on
3'b100: y=a*b; //Mul□ply
default: y=32'bx;
endcase
end
endmodule
```

Source Code (using if Statement)

```
module alu_32bit_if(y,a,b,f);
input [31:0]a,b;
input [2:0]f;
output reg [32:0]y;
always@(*)
begin
if(f==3'b000)
y=a&b; //AND Opera□on
else if (f==3'b001)
y=a|b; //OR Opera□on
else if (f==3'b010)
y=a+b; //Addi□on
else if (f==3'b011)
y=a-b; //Subtrac□on
else if (f==3'b100)
y=a*b; //Mul□ply
else
y=32'bx;
end
endmodule
```

Test-bench Code (same for both)

```
module alu_32bit_tb_if;
reg [31:0]a,b;
reg [2:0]f;
wire [32:0]y;
alu_32bit_if test(.y(y),.a(a),.b(b),.f(f));
/* or alu_32bit_case
test(.y(y),.a(a),.b(b),.f(f));
if source code used for case statement*/
initial
begin
a=32'h00000000;
b=32'hFFFFFFFF;
#10; f=3'b000;
#10; f=3'b001;
#10; f=3'b010;
#10; f=3'b100;
end
endmodule
```

sdc-

```
create_clock -name clk -period 2 -waveform {0 1} [get_port "clk"]
set_clock_transition -rise 0.01 [get_clock "clk"]
set_clock_transition -fall 0.01 [get_clock "clk"]
set_clock_uncertainty 0.01 [get_ports "clk"]
set_input_delay -max 0 -clock clk [all_inputs]
set_output_delay -max 0 -clock clk [all_outputs]
set_load 0.15 [all_outputs]
```

1. 4-Bit Up/Down Asynchronous Counter

Source Code

```
`timescale 1ns/1ps
module counter(clk,rst,m,count);
input clk,rst,m;
output reg [3:0]count;
always@(posedge clk or negedge rst)
begin
if(!rst)
count=0;
else if(m)
count=count+1;
else
count=count-1;
end
endmodule
```

Test-bench Code

```
`timescale 1ns/1ps
module counter_test;
reg clk, rst,m;
wire [3:0] count;
initial
begin
clk=0;
rst=0; #25;
rst=1;
end
initial
begin
m=1;
#600; m=0;
rst=0; #25;
rst=1;
#500; m=0;
end
counter
counter1(clk,rst,m,count);
always #5 clk=~clk;
initial
#1400 $finish;
endmodule
```

sdc file-

```
create_clock -name clk -period 2 -waveform {0 1} [get_port "clk"]
set_clock_transition -rise 0.01 [get_clock "clk"]
set_clock_transition -fall 0.01 [get_clock "clk"]
set_clock_uncertainity 0.01 [get_ports "clk"]
set_input_delay -max 0.8 [get_ports "rst"] -clock [get_clocks "clk"]
set_output_delay -max 0.8 [get_ports "count"] -clock [get_clocks
"clk"]
set_input_transition 0.12 [all_inputs]
set_load 0.15 [all_outputs]
```

4-Bit Adder

Source Code (fa.v)

```
module full_adder( A,B,CIN,S,COUT);
input A,B,CIN;
output S,COUT;
assign S = A^B^CIN;
assign COUT = (A&B) | (CIN&(A^B));
endmodule
```

Source Code (fa_4bit.v)

```
module four_bit_adder(A,B,C0,S,C4);
input [3:0] A,B;
input C0;
output [3:0] S;
output C4;
wire C1,C2,C3;
full_adder fa0 (A[0],B[0],C0,S[0],C1);
full_adder fa1 (A[1],B[1],C1,S[1],C2);
full_adder fa2 (A[2],B[2],C2,S[2],C3);
full_adder fa3 (A[3],B[3],C3,S[3],C4);
endmodule
```

Test-bench Code

```
module test_4_bit;
reg [3:0] A,B;
reg C0;
wire [3:0] S;
wire C4;
four_bit_adder dut(A,B,C0,S,C4);
initial begin
A=4’b0011; B=4’b0011; C0=1’b0; #10;
A=4’b1011; B=4’b0111; C0=1’b1; #10;
A=4’b1111; B=4’b1111; C0=1’b1; #10;
end
endmodule
```

sdc file-

```
create_clock -name clk -period 2 -waveform {0 1} [get_port "clk"]
set_clock_transition -rise 0.01 [get_clock "clk"]
set_clock_transition -fall 0.01 [get_clock "clk"]
set_clock_uncertainty 0.01 [get_ports "clk"]
set_input_delay -max 0 -clock clk [all_inputs]
set_output_delay -max 0 -clock clk [all_outputs]
set_load 0.15 [all_outputs]
```

Latches and Flip Flops

Source code for D-Flip Flop :

```
module DFF( Q,Qbar,D,Clk,Reset);
output reg Q;
output Qbar;
input D,Clk,Reset;
always @(posedge Clk)
begin
if (Reset == 1'b1) //If at reset
Q <= 1'b0;
else
Q <= D;
end
assign Qbar = ~Q;
endmodule
```

Source code for D-Latch :

```
module DFF( Q,Qbar,D,en,Reset);
output reg Q;
output Qbar;
input D,en,Reset;
assign Reset = ~D;
always @(en)
begin
if (Reset == 1'b1) //If at reset
Q <= 1'b0;
else
Q <= D;
end
assign Qbar = ~Q;
endmodule
```

Source code for SR Flip Flop :

```
module Main(S,R,clk,Q,Qbar);
input S,R,clk;
output Q,Qbar;
reg M,N;
always @(posedge clk)
begin
M = !(S & clk);
N = !(R & clk);
end
assign Q = !(M & Qbar);
assign Qbar = !(N & Q);
endmodule
```

Source Code for SR Latch :

```
module Main(S,R,en,Q,Qbar);
input S,R,en;
output Q,Qbar;
reg M,N;
always @(en)
begin
M <= !(S & clk);
N <= !(R & clk);
end
assign Q <= !(M & Qbar);
assign Qbar <= !(N & Q);
endmodule
```

Source Code for JK Flip Flop :

```
module jkff(J, K, clk, Q);
input J, K, clk;
output reg Q,Qm;
always @(posedge clk)
begin
if(J == 1 && K == 0)
Qm = 1;
else if(J == 0 && K == 1)
Qm = 0;
else if(J == 1 && K == 1)
Qm = ~Qm;
end
endmodule
```

Source of JK Latch :

```
module jkff(J, K, en, Q);
input J, K, en;
output reg Q,Qm;
always @(en)
begin
if(J == 1 && K == 0)
Qm <= 1;
else if(J == 0 && K == 1)
Qm <= 0;
else if(J == 1 && K == 1)
Qm <= ~Qm;
end
endmodule
```

---

sdc for latch-

```
create_clock -name en -period 2 -waveform {0 1} [get_port "en"]
set_clock_transition -rise 0.01 [get_clock "en"]
set_clock_transition -fall 0.01 [get_clock "en"]
set_clock_uncertainity 0.01 [get_ports "en"]
set_input_delay -max 1.0 -clock en [get_ports "D"]
set_input_delay -max 1.0 -clock en [get_ports "Reset"]
set_output_delay -max 1.0 -clock en [get_ports "Q"]
set_output_delay -max 1.0 -clock en [get_ports "Qbar"]
set_load 0.15 [all_outputs]
```

sdc ff-

```
create_clock -name clk -period 2 -waveform {0 1} [get_port "clk"]
set_clock_transition -rise 0.01 [get_clock "clk"]
set_clock_transition -fall 0.01 [get_clock "clk"]
set_clock_uncertainity 0.01 [get_ports "clk"]
set_input_delay -max 1.0 -clock clk [get_ports "Reset"]
set_input_delay -max 1.0 -clock clk [get_ports "D"]
set_output_delay -max 1.0 -clock clk [get_ports "Q"]
set_output_delay -max 1.0 -clock clk [get_ports "Qbar"]
set_load 0.15 [all_outputs]
```