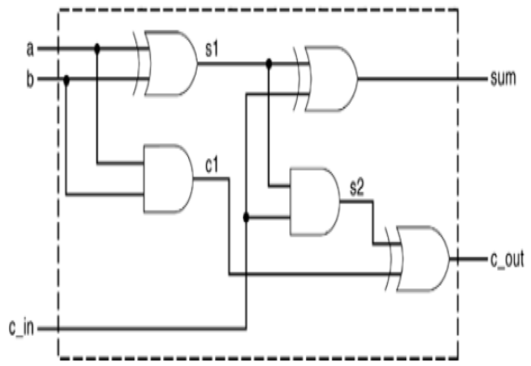
GATE LEVEL MODELING

1. FULL ADDER:

   a) DIAGRAM:



$$\text{sum} = (a \oplus b \oplus \text{cin})$$

$$\text{cout} = (a \cdot b) + \text{cin} \cdot (a \oplus b)$$

   b) CODE:

```
module fulladd(sum, c_out, a, b, c_in);
output sum, c_out;
input a, b, c_in;
// Internal nets
wire s1, s2, c1;
xor (s1, a, b);
and (c1, a, b);
xor (sum, s1, c_in);
and (s2, s1, c_in);
or (c_out, s2, c1);
endmodule
```

   c) TEST BENCH:

```
module
```

2) AND GATE:

   a) CODE:

```
module andgate (c,a,b);
input a,b;
output c;
and a1(c,a,b);
endmodule
```

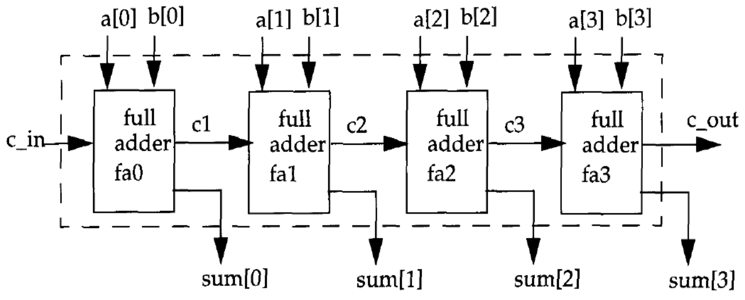
   b) TEST BENCH:

```
module stimulus;
reg at;
reg bt;
wire ct;
// instantiate the design block
andgate  t1(ct, at, bt);
initial
begin
 #5 at = 1'b0; bt=1'b0;
 #10 at = 1'b0; bt=1'b1;
 #5 at = 1'b1; bt=1'b0;
 #5 at = 1'b1; bt=1'b1;
```

```
end
endmodule
```

3) 4 BIT FULL ADDER:

a) DIAGRAM:



b) CODE:

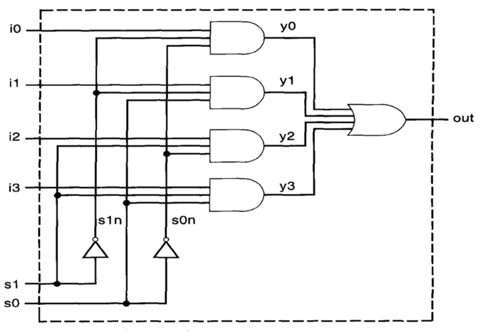
```
module fulladd4(sum, c_out, a, b, c_in);
output [3:0] sum;
output c_out;
input[3:0] a, b;
input c_in;
// Internal nets
wire c1, c2, c3;
fulladd fa0(sum[0], c1, a[0], b[0], c_in);
fulladd fa1(sum[1], c2, a[1], b[1], c1);
fulladd fa2(sum[2], c3, a[2], b[2], c2);
fulladd fa3(sum[3], c_out, a[3], b[3], c3);
endmodule
```

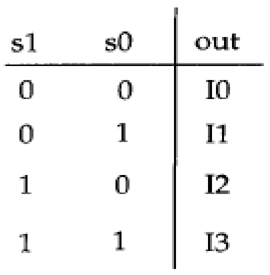
c) TEST BENCH:

```
module stimulus;
// Set up variables
reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;
fulladd4 FA1_4(SUM, C_OUT, A, B, C_IN);
Initial
begin
A = 4'd0; B = 4'd0; C_IN = 1'b0;
#5 A = 4'd3; B = 4'd4;
#5 A = 4'd2; B = 4'd5;
#5 A = 4'd9; B = 4'd9;
#5 A = 4'd10; B = 4'd15;
#5 A = 4'd10; B = 4'd5; C_IN = 1'b1;
end
initial
begin
$monitor($time," A= %b, B=%b, C_IN= %b, --- C_OUT= %b, SUM= %b\n",A, B, C_IN, C_OUT, SUM);
end
endmodule
```

4) 4 TO 1 MUX:

a) DIAGRAM:





s1	s0	out
0	0	I0
0	1	I1
1	0	I2
1	1	I3

b) CODE:

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
wire s1n, s0n;
wire y0, y1, y2, y3;
not (s1n, s1);
not (s0n, s0);
and (y0, i0, s1n, s0n);
and (y1, i1, s1n, s0);
and (y2, i2, s1, s0n);
and (y3, i3, s1, s0);
or (out, y0, y1, y2, y3);
endmodule
```

c) TEST BENCH:

```
Mod
module stimulus;
reg IN0, IN1, IN2, IN3;
reg S1, S0;
wire OUTPUT;
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
initial
begin
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
#5 S1 = 0; S0 = 0;
#5 S1 = 0; S0 = 1;
#5  S1 = 1; S0 = 0;
#5 S1 = 1; S0 = 1;
end
endmodule
```

5) D FLIP FLOP:

a) DIAGRAM:

b) CODE:

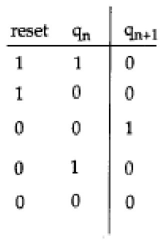
```
module D-FF(q, d, clk, reset) ;
output q;
input d, clk, reset;
reg q;
always @(posedge reset or negedge clk)
if (reset)
q = 1'b0;
```

```
else
q = d;
endmodule
```

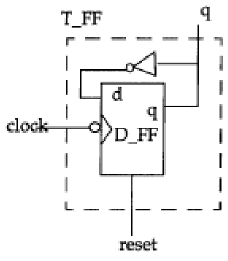
c) TEST BENCH:

6) T FLIP FLOP:

a) DIAGRAM:



reset	q_n	q_{n+1}
1	1	0
1	0	0
0	0	1
0	1	0
0	0	0



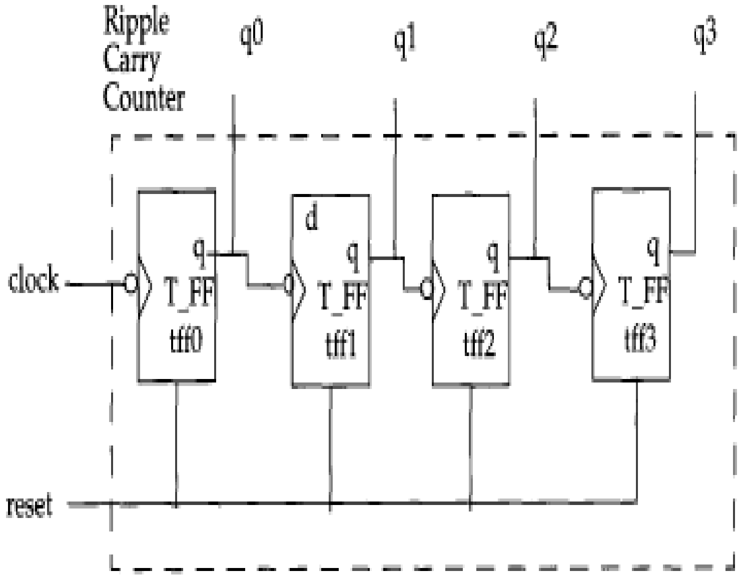
b) CODE:

```
module T-FF(q, clk, reset) ;
output q;
input clk, reset;
wire d;
D-FF dff0 (q, d, clk, reset) ;
not nl(d, q);
endmodule
```

c) TEST BENCH:

7) 4 BIT RIPPLE CARRY COUNTER:

a) DIAGRAM:



b) CODE:

```
module ripple-carry-counter(q, clk, reset);
output [3:0] q;
input clk, reset;
T-FF tffO(q[0],clk,reset);
T-FF tff1(q[1] ,q[0], reset);
T-FF tff2 (q[2] ,q[1], reset) ;
T-FF tff3(q[3] ,q[2], reset) ;
```

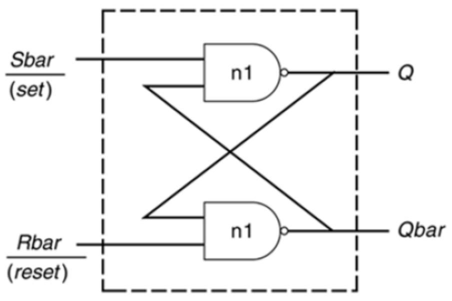
```
endmodule
```

c) TEST BENCH:

```
module stimulus;
reg clk;
reg reset;
wire[3:0] q;
// instantiate the design block
ripple-carry-counter rl(q, clk, reset);
// Control the clk signal that drives the design block.Cycle time = 10ns
initial
clk = 1'b0; //set clk to 0
always
#5 clk = -clk; //toggle clk every 5 time units
// Control the reset signal that drives the design block
initial
begin
reset = 1'b1;
#15 reset = 1'b0;
#180 reset = 1'b1;
#10 reset = 1'b0;
#20  $finish; //terminate the simulation
end
// Monitor the outputs
initial
$monitor ($time, " Output q = %d" , q) ;
endmodule
```

8) SR LATCH:

a) DIAGRAM:



b) CODE:

```
// Module name and port list
// SR_latch module
module SR_latch(Q, Qbar, Sbar, Rbar);
//Port declarations
output Q, Qbar;
input Sbar, Rbar;
// Instantiate lower-level modules
// In this case, instantiate Verilog primitive nand gates
// Note, how the wires are connected in a cross-coupled fashion.
nand n1(Q, Sbar, Qbar);
nand n2(Qbar, Rbar, Q);
// endmodule statement
endmodule
```

c) TEST BENCH:

```
// Stimulus module
// Module name and port list is not present
module Top;
// Declarations of wire, reg, and other variables
wire q, qbar;
reg set, reset;
// Instantiate lower-level modules
// In this case, instantiate SR_latch
// Feed inverted set and reset signals to the SR latch
SR_latch m1(q, qbar, ~set, ~reset);
```

BEHAVIORAL MODELING

1) 4 TO 1 MUX:

a) Verilog code:

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
// Port declarations from the I/O diagram
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
case ({s1, s0}) //Switch based on concatenation of control signals
2’b00 : out = i0;
2’b01 : out = i1;
2’b10 : out = i2;
2’d11 : out = i3;
default: $display("Invalid control signals");
endcase
endmodule
```

2) 4-bit Ripple Carry Counter:

a) Verilog code:

```
module stimulus;
// Set up variables
reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;
fulladd4 FA1_4(SUM, C_OUT, A, B, C_IN);
Initial
begin
A = 4'd0; B = 4'd0; C_IN = 1'b0;
#5 A = 4'd3; B = 4'd4;
#5 A = 4'd2; B = 4'd5;
#5 A = 4'd9; B = 4'd9;
#5 A = 4'd10; B = 4'd15;
#5 A = 4'd10; B = 4'd5; C_IN = 1'b1;
end
```

```
initial
begin
$monitor($time," A= %b, B=%b, C_IN= %b, --- C_OUT= %b, SUM= %b\n",A, B, C_IN, C_OUT, SUM);
end
endmodule
```
3)