

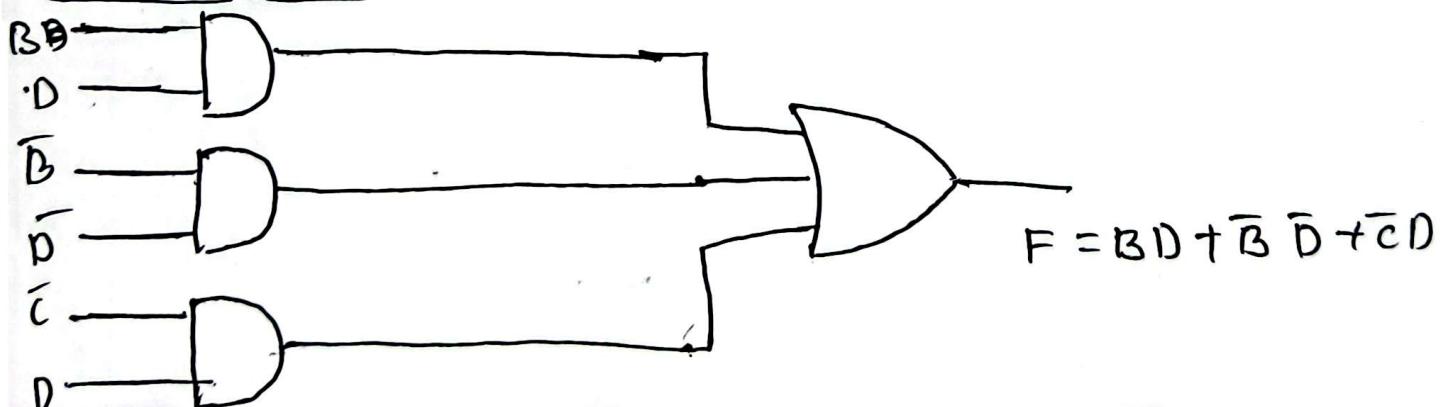
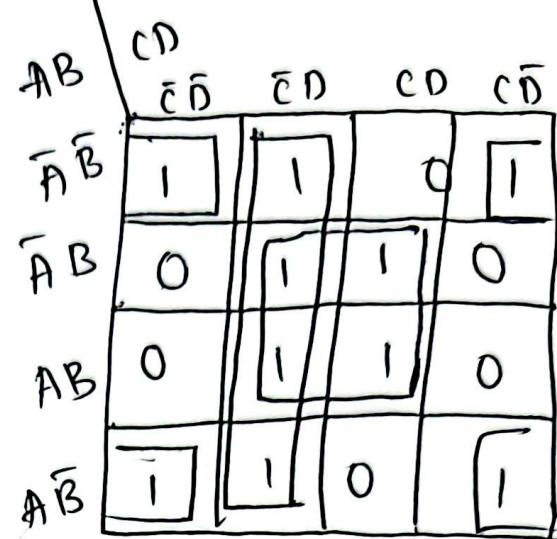


## INDEX

Sl.No.	Date	DESCRIPTION	Page No.	Marks	Signature
1	20/11/23	Program on Structural	1-2	8	<del>JF</del>
2	4/12/23	Program on half adder and full adder and Subtractor	3-6	10	<del>JF</del>
3	18/12/23	Program on multiplex	7-9	10	<del>JF</del>
4	18/1/24	Program On de-multiplex	10-12	10	<del>JF</del>
5	1/1/24	Program on flip flop	13-15	10	<del>JF</del>
6	22/1/24	program on 4 bit full adder	16-17	10	<del>JF</del>
7	12/2/24	Program on BCD adder	17-18	16	<del>JF</del>
8	12/2/24	Program On behavioural model	19-20	10	<del>JF</del>

Truth table

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Logic gateK-map



TITLE

Given a 4 variable logic expression. Simplify it using appropriate technique & stimulate the same using basic gate  $F(A, B, C, D) = \sum m(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$   
Aim:- To Simplify the given 4-variable logic expression using K-map & Stimulate using basic gates

Theory :- A K-map is a graphical representation of truth-table which is a systematic way to simplify & optimize boolean algebra expression used in digital circuit design. K-maps are helpful for minimizing the no of logic gates in a given logic func

Basic Concepts :-

Truth Table :- A truth-table is a tabular representation of all possible input combination & their corresponding output values for a logic func. It helps define the behaviour of the func.

Boolean :- A mathematical func System used to manipulate binary Variable & perform logical operation such as AND, OR & NOT. It is essential for Simplify & optimizing logic func



TITLE

Program :

module. AOI - structural (A,B,C,D,F);  
input A,B,C,D;  
output F;  
assign F =  $\neg(C \cdot (A \oplus B)) \mid (C \oplus D)$ ;  
endmodule

Program

module. AOI - structural (F,A,B,C,D);  
input A,B,C,D;  
output F;  
assign F =  $(B \oplus D) \mid (\neg C \oplus D)$ ;  
end module;

The image shows a software application window with a dark-themed interface. On the left, there is a sidebar containing five entries, each with a blue circular icon and text: 'IAOL\_structural/F', 'IAOL\_structural/A', 'IAOL\_structural/H', 'IAOL\_structural/C' (which is highlighted in white), and 'IAOL\_structural/D'. To the right of the sidebar is a vertical column of file names: 'S1', 'S20', 'S20', 'S1', and 'S20'. Below this column is a large, empty grid area consisting of 12 columns and 10 rows, with a yellow vertical line running through the 6th column.

IAOL_structural/F	S1										
IAOL_structural/A	S20										
IAOL_structural/H	S20										
IAOL_structural/C	S1										
IAOL_structural/D	S20										

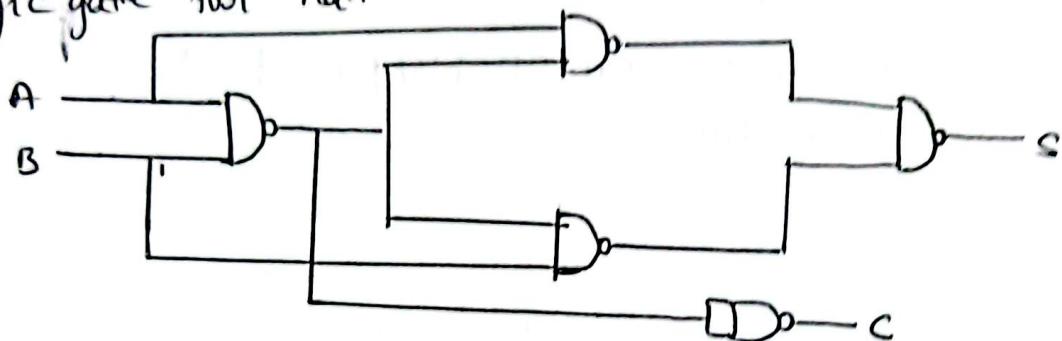
## Truth table for half adder

n	A	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \oplus B$$

$$C = AB$$

## Logic gate for half adder



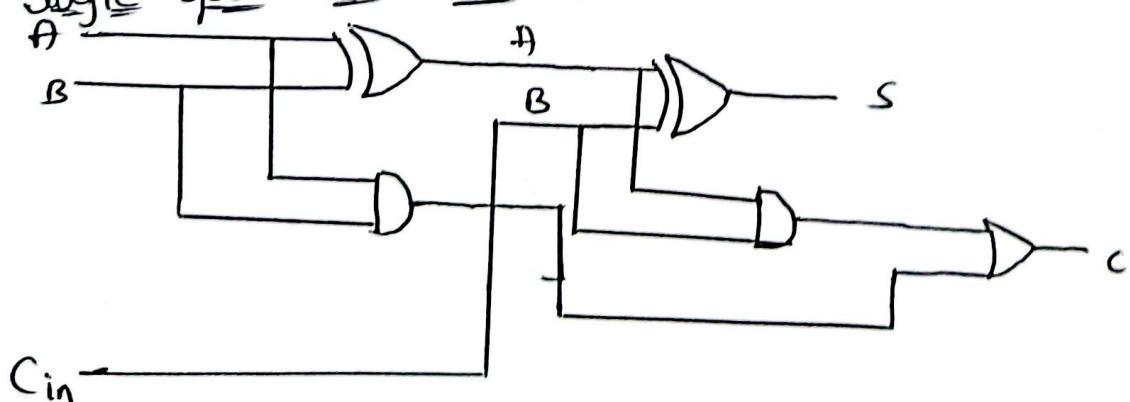
## Truth table for full adder

A	B	C <sub>in</sub>	S	C
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

$$S = A \oplus B \oplus C$$

$$C_{in} = AB + BC_{in} + AC_{in}$$

## Logic gate for full adder





TITLE

Design Verilog HDL to implement Binary Address-Subtractors Half & full adder, Half & full-subtractor

Aims- To design & verify

i) Half adder & full adder

ii) Half subtractor & full subtractor

Theory:-

Half adder: A Combinational logical circuit that perform the addition of 2 bits of data. A & B is called one of which is Sum bit & and the other is the carry bit C. The boolean func are

$$S = A \oplus B \quad C = AB$$

Full Adder: A Combinational logical circuit that perform the half adder which doesn't take carry bit from its previous stage into the carry bit in previous stage. Called Carry-in-bit that adds 2-bits, A & B. & carry a in Cin, Cin is called a full Adder. The boolean func are.

$$S = x \oplus y \oplus \text{Cin} \quad C = xy + \text{Cin}(x \oplus y)$$



TITLE

Half Subtraction :- Subtracting a single bit binary value B another A produces a difference bit P & a below out bit B-out. The operation is called half subtractor & the circuit to realize it is called half-subtractor. The boolean func are

$$S = A \oplus B C = A' B$$

Full Subtraction :- Subtracting a single bit binary value B, (in func) a single bit value. It produces a difference bit D & a boolean Out B bit this is called full subtraction. The boolean func are

$$D = (x \oplus y) \oplus \text{cin} \quad B = A' B + A' (\text{cin}) + B (\text{cin})$$

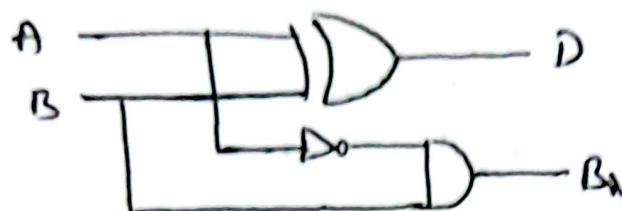
## Half Subtractor Truth Table

A	B	D	$B_1$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D = A \oplus B$$

$$B_1 = \bar{A}B$$

## Logic gate Half Subtractor



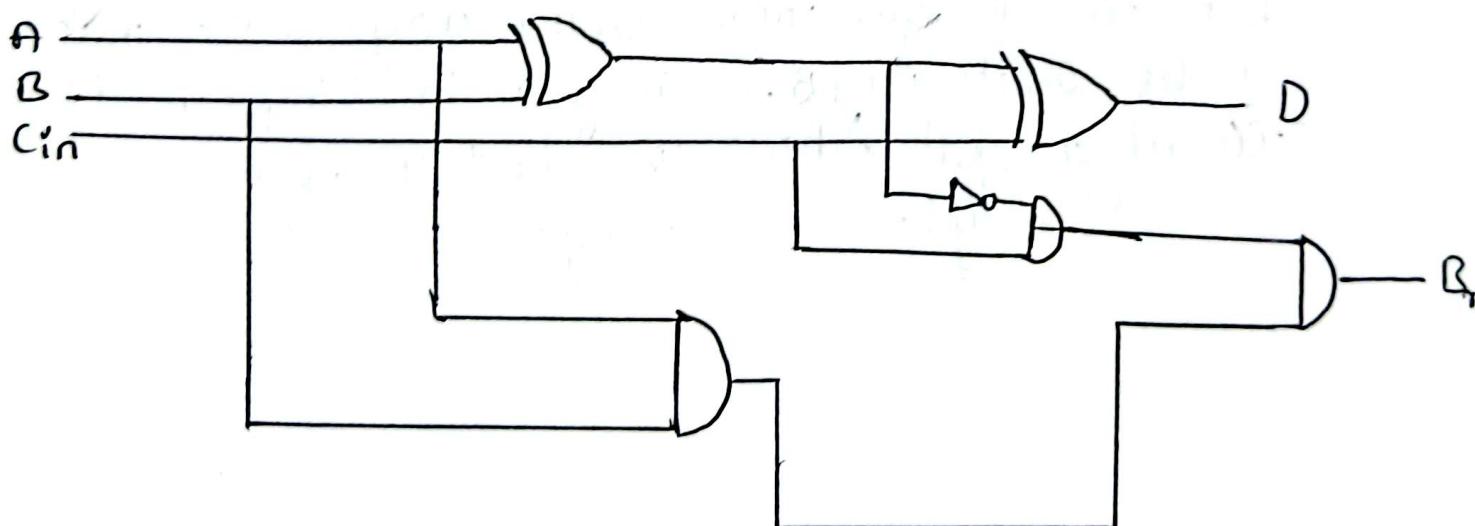
## Full Subtractor - Truth Table

A	B	$C_{in}$	D	$B_1$
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

$$D = A \oplus B \oplus C$$

$$B_1 = \bar{A}B + B C_{in} + \bar{A} C_{in}$$

## Logic gate - full subtractor



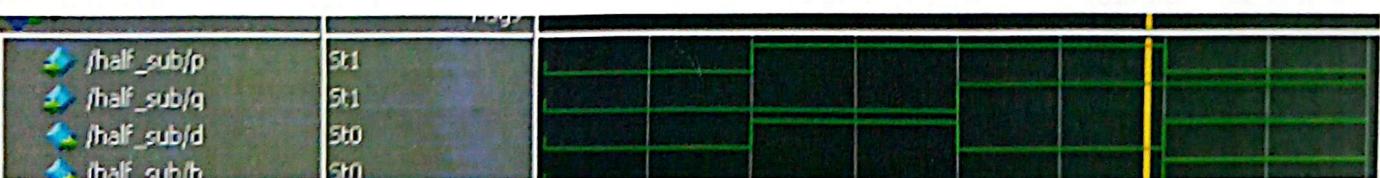
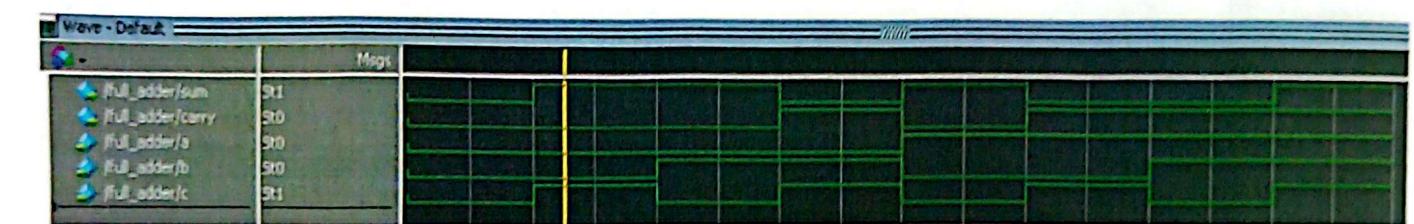


TITLE

Program on half Adder  
module half-adder(a,b,sum,carry);  
input a,b;  
output sum, carry;  
assign sum = a^b;  
assign carry = (a&b);  
endmodule;

Program on Full adder  
module full-adder(a,b,c,sum,carry);  
input a,b,c;  
Output sum = a^b^c;  
assign sum = (a&b)|(a&c)|(c&b);  
endmodule.

Program on half Subtractor  
module half-subtractor(A,B,D,B1);  
input A,B ;  
Output D, B1 ;  
assign D = A^B ;  
assign B1 = (~A)&B ;  
endmodule





TITLE

Program on full Subtractor  
module full-Subtractor ( $A, B, Cin, D, B_i$ );  
input  $A, B, Cin$ ;  
output  $D, B_i$ ;  
assign  $D = A' B' Cin$ ;  
assign  $B_i = (n(A \oplus B)) \mid (n(A' B') \& Cin)$ ;  
end module





TITLE

Design Verilog program to implement different types of multiplex 2:1, 4:1, 8:1

Aim :- Implement different types of multiplex 2:1, 4:1  
8:1

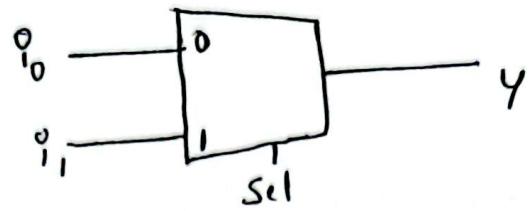
Theory :-

- 2:1 multiplex :- It has two input & one select line
- 4:1 multiplex :- It has 4 input & 2 select lines
- 8:1 multiplex :- It has 8 input & 3 select lines

A multiplex is a combination circuit that selects binary information from many input lines & direct it to a single output line. The selection of particular input lines is controlled by a set of selection lines.

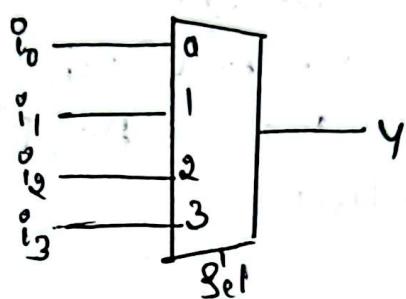
Truth table of 2:1 multiplexer

Sel.	Y
0	$i_0$
1	$i_1$



Truth table 4:1 multiplexer block diagram

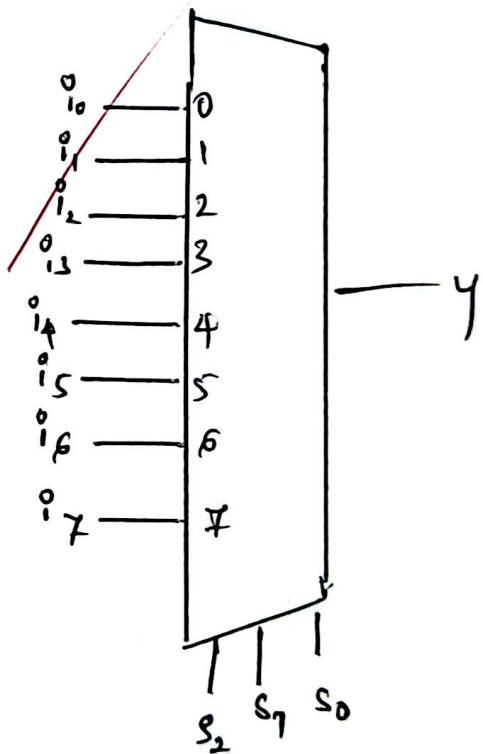
Sel[0]	Sel[1]	Y
0	0	$i_0$
0	1	$i_1$
1	0	$i_2$
1	1	$i_3$



block diagram

Truth table 8:1 Multiplexer

S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Y
0	0	0	$i_0$	
0	0	1	$i_1$	
0	1	0	$i_2$	
0	1	1	$i_3$	
1	0	0	$i_4$	
1	0	1	$i_5$	
1	1	0	$i_6$	
1	1	1	$i_7$	





TITLE

Program On 2:1 multiplex  
module m21(D0, D1, S, Y);  
input wire D0, D1, S;  
output reg Y;  
always @ (D0 or D1 or S)  
begin  
if (S)  
Y = D1;  
else  
Y = D0;  
end  
endmodule

Program On 4:1 multiplex  
module mux4(in, sel, out);  
input [3:0] in;  
input [1:0] sel;  
output reg out;  
always @ (sel)  
begin  
case (sel)  
0' b00: out = in[0];  
0' b01: out = in[1];  
0' b10: out = in[2];  
0' b11: out = in[3];  
end case  
end  
endmodule

	Msgs
/mux2x1/0	\$t0
/mux2x1/1	\$t1
/mux2x1/S	\$t0
/mux2x1/Y	0

Diagram illustrating the output of the 2x1 MUX. The output is connected to the input of a 4x1 MUX.

	Msgs
/mux41/in	0000
[3]	\$t0
[2]	\$t0
[1]	\$t0
[0]	\$t0
/mux41sel	00
[1]	\$t0
[0]	\$t0
/mux41/out	0



ANJUMAN INSTITUTE OF TECHNOLOGY  
& MANAGEMENT  
ANJUMANABAD, BHATKAL - 581 320

PAGE NO. : ..... 09 .....  
DATE : ..... .....  
EXP. NO. : ..... 03 .....

TITLE

Program On 8:1 multiplex  
module mux81(y,a,s)

input [7:0] a;

input [2:0] s;

output reg y;

always @ (a,s)

begin

case (s)

3' b000: y = a[0];

3' b001: y = a[1];

3' b010: y = a[2];

3' b011: y = a[3];

3' b100: y = a[4];

3' b101: y = a[5];

3' b110: y = a[6];

3' b111: y = a[7];

end case

end

end module



TITLE

Design Verilog program to implement types of De-multiplex

Aim & Program On De-multiplex

Theory :-

A demultiplexer is a Combinational Circuit that works exactly opposite to a multiplexer. A Demux has a single input line that connects to any one of the output lines based on its control input signal or selection lines.

There are 2 types of Demultiplexer

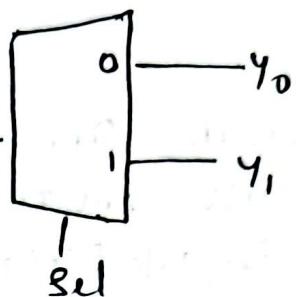
1:2 Demultiplexer : It has one select line and 2 output lines

1:4 Demultiplexer : It has one Select line & 4 output lines

Truth table 1:2 DEMUX

Sel	$y_0$	$y_1$
0	1	0
1	0	1

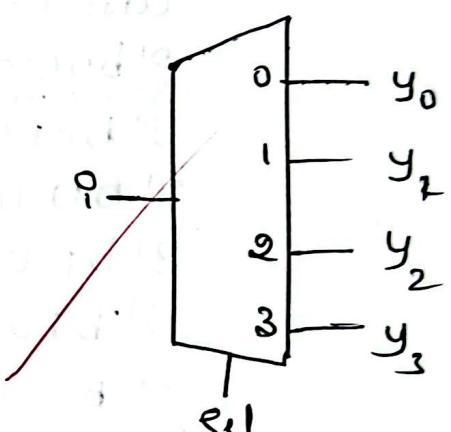
Block Diagram



Truth table 1:4 DEMUX

sel [0]	sel [1]	$y_0$	$y_1$	$y_2$	$y_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Block Diagram



Truth table 1 to 4 DEMUX

input		out put				
$s_1$	$s_2$	a-in	$y_3$	$y_2$	$y_1$	$y_0$
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0



ANJUMAN INSTITUTE OF TECHNOLOGY  
& MANAGEMENT  
ANJUMANABAD, BHATKAL - 581 320

PAGE NO. : ..... 11 .....

DATE : .....

EXP. NO. : ..... 04 .....

TITLE

Program On demux

module demux1\_4(a-in, sel, y);

input a-in;

input [1:0]sel;

output [3:0] y;

begin

case(sel)

2'b00 = begin y[0]=a-in;

y[1]= 1'b0;

y[2]= 1'b0;

y[3]= 1'b0;

end

2'b01= begin y[0]=1'b0;

y[1]= a-in;

y[2]= 1'b0;

y[3]= 1'b0;

end

2'b10: begin y[0]=1'b0;

y[1]= 1'b0;

y[2]= a-in;

y[3]= 1'b0;

end

Timing diagram showing the output of a 4-to-1 multiplexer (Demux1\_4) based on the selection input (sel).

The sel input has four states: 00, 01, 10, and 11. The corresponding outputs are 0001, 0010, 0100, and 1000 respectively.

	00	01	10	11
0001	1	0	0	0
0010	0	1	0	0
0100	0	0	1	0
1000	0	0	0	1



TITLE

$2^{'b}11 : begin \quad y[0] = 'b0;$

$y[1] = 'b0;$

$y[2] = 'b0;$

$y[3] = a - in;$

end

default :  $y = 3^{'b}000;$

endcase

end

endmodule.



TITLE

Design Verilog program for implementation of various types of flip-flops such as SR, JK and D

Aim :-

To design and implement various types of flip-flop like D using Verilog HDL

Verilog code for JK flip-flop  
module jk\_ff(jk, q, qb)

input [1:0] jk;

input clk;

output q;

output qb;

always @ (posedge clk)

begin

case (jk)

2'b00: q=q;

2'b01: q=0;

2'b10: q=1;

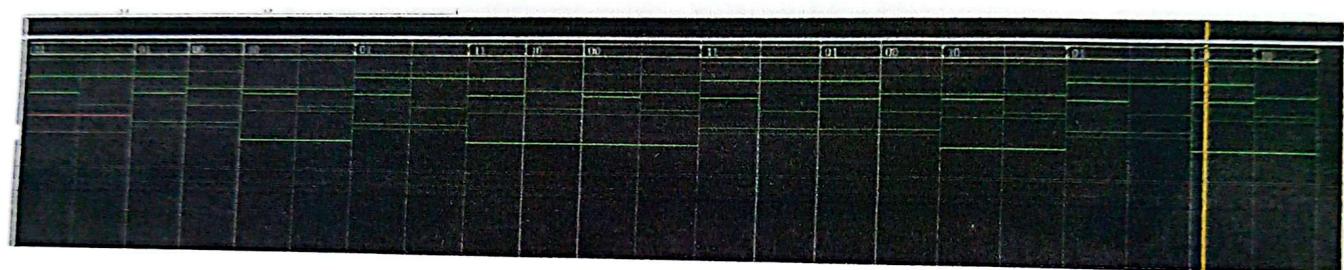
2'b11: q=~q;

endcase

end

endmodule

## JK FLIPFLOP





**ANJUMAN INSTITUTE OF TECHNOLOGY  
& MANAGEMENT**  
ANJUMANABAD, BHATKAL - 581 320

PAGE NO. : 14  
DATE : .....  
EXP. NO. : 05

TITLE

Verilog Code for SR-flipflop  
module SR-ff(sr,clk,q,qb);  
input [1:0]sr;  
input Eclk;  
output q,qb;  
reg q,qb;  
always @ (posedge clk)  
begin  
case(sr)  
2'b00: q=q;  
2'b01: q=0;  
2'b10: q=1;  
2'b11: q = 1'bz;  
endcase  
end  
endmodule



SR FLIPFLOP





ANJUMAN INSTITUTE OF TECHNOLOGY  
& MANAGEMENT  
ANJUMANABAD, BHATKAL - 581 320

PAGE NO. .... 15 .....

DATE : .....

EXP. NO. .... 05 .....

TITLE

Verilog code for D-flip flop  
module clff(d,clk,a,q,b)

input d;

input clk;

output q,qb;

reg q,qb;

always@ (posedge clk)

begin .

case (d)

1'b0: q=0;

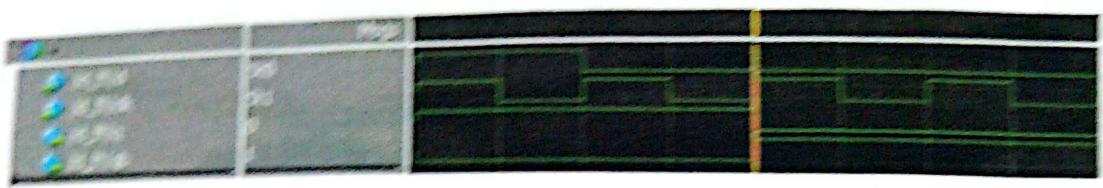
1'b1: q=1;

endcase

qb=q;

end

endmodule





TITLE

Design a ~~half~~ 1 bit full adder and Subtract  
or and stimulate the same using basic gates

Aim :- To design 4 bit full adder using logic  
gates

VHDL code for 1-bit full adder

```
module rippleverilog (sum, cout, a, b, cin);
    input a, b, cin;
    output sum, cout;
    wire s1, c1, s2;
    xor (s1, a, b);
    xor (sum, s1, cin);
    and (c1, a, b);
    and (c2, s1, cin);
    or (cout, c1, c2);
endmodule
```

// origin 4 bit ripple adder

```
module fourbitadder (sum, cout, a, b, cin);
    input [3:0] a;
    input [3:0] b;
    input cin;
    output [3:0] sum;
    output cout;
    wire c1, c2, c3;
    uippleverilog u1 (sum[0], c1, a[0], b[0], cin);
    uippleverilog u2 (sum[1], c2, a[1], b[1], c1);
    uippleverilog u3 (sum[2], c3, a[2], b[2], c2);
    uippleverilog u4 (sum[3], cout, a[3], b[3], c3);
endmodule
```

Fourbitadder

	0000	0010	0011	0100	0101	0110	0111	1000	1001	1010
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010
0010	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010
0011	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010
0100	0010	0100	0110	1000	1010	1100	1110	0000	0010	0100
0101	0010	0100	0110	1000	1010	1100	1110	0000	0010	0100
0110	0010	0100	0110	1000	1010	1100	1110	0000	0010	0100
0111	0010	0100	0110	1000	1010	1100	1110	0000	0010	0100
1000	0100	1000	1010	0000	0010	0100	0110	1000	1001	1010
1001	0100	1000	1010	0000	0010	0100	0110	1000	1001	1010
1010	0100	1000	1010	0000	0010	0100	0110	1000	1001	1010



ANJUMAN INSTITUTE OF TECHNOLOGY  
& MANAGEMENT  
ANJUMANABAD, BHATKAL - 581 320

PAGE NO. : ..... 17 .....

DATE : .....

EXP. NO. : ..... 06 .....

TITLE

~~Uipple verilog r3 [ sum[2], c3, a[1], b2, c2];  
Uipple Verilog r4 [Sum [3], cout ,a[3], b[3], c3);  
endmodule~~

Simulated output of ripple counter is obtained  
step wise of the count incrementing by 1000.

Output of ripple counter is applied to the  
display module.

Output of display module is displayed.

	Mos								
rippleverlog/a	S10								
rippleverlog/b	S11								
rippleverlog/cn	S10								
rippleverlog/sum	S11								
rippleverlog/out	S10								
rippleverlog/s1	S11								
rippleverlog/c1	S10								
rippleverlog/c2	S10								

(0), (1), (2), (3), (4)  
(5), (6), (7), (8), (9)  
and (A), (B), (C), (D)  
etc. are displayed on  
display module.

Output of display module is  
fed back to the feedback station  
which is connected to the  
input of the display module.  
This forms a feedback loop.

Simultaneously, the output of the  
display module is fed to the

display driver module.

Display driver module displays the  
output of the display module.



TITLE

Design Verilog HDL to implement Decimal Adder

Aim:

To design a decimal or BCD adder using Verilog HDL

Verilog code for BCD adder:

```
module bcd_adder(a,b,carry-in,sum,carry);
    input [3:0]a,b;
    input carry_in;
    output [3:0] sum;
    output carry;
    reg [4:0] sum_temp;
    reg [3:0] sum;
    reg carry;
    always @ (a,k,carry_in);
begin
```

Sum-temp = a+b+carry\_in;

If (sum-temp > 9) begin

sum-temp = sum-temp + 6;

carry = 1;

sum = sum-temp [1:0]; end

else begin

carry = 0;

sum = sum-temp[3:0]; end

end

	Mys	0000	0110	0011	0100	1001	1001
/bcd_adder/a	1000	0000	1001	0011	0101	1001	1001
/bcd_adder/b	0010	0000	1001	0011	0101	1001	1001
/bcd_adder/carry_in	St1				1001	1000	1001
/bcd_adder/sum	0001	0000	0110		1001	1000	11001
/bcd_adder/carry	1				01001	10001	
/bcd_adder/sum_temp	10001	00000	10110	00110	01001	10001	



TITLE

Design Verilog HDL to implement Simple Circuit using structured, Data flow an behavioural model

Aim :- To design a Simple Circuit using Standard Structural, data flow and behavioural model

Verilog Code for Structural model

```
module rippleverilog (Sum, Cout, a, b, cin);
    input a, b, cin;
    output Sum, Cout;
    wire s1, c1, c2;
    xor (s1, a, b);
    xor (Sum, s1, cin);
    and (c1, a, b);
    and ((c2, s1, cin));
    or (cout, c1, c2);
endmodule
```

Verilog Code for dataflow model :-

```
module and2( x1, x2, z1);
    output x1, x2;
    output z1;
    wire x1, x2;
    wire z1;
    assign z1 = x1 & x2;
endmodule
```

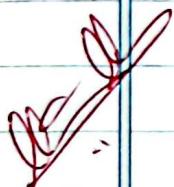
## Behavioural model

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	10010	10011	10012	10013	10014	10015	10016	10017	10018	10019	10020	10021	10022	10023	10024	10025	10026	10027	10028	10029	10030	10031	10032	10033	10034	10035	10036	10037	10038	10039	10040	10041	10042	10043	10044	10045	10046	10047	10048	10049	10050	10051	10052	10053	10054	10055	10056	10057	10058	10059	10060	10061	10062	10063	10064	10065	10066	10067	10068	10069	10070	10071	10072	10073	10074	10075	10076	10077	10078	10079	10080	10081	10082	10083	10084	10085	10086	10087	10088	10089	10090	10091	10092	10093	10094	10095	10096	10097	10098	10099	100100	100101	100102	100103	100104	100105	100106	100107	100108	100109	100110	100111	100112	100113	100114	100115	100116	100117	100118	100119	100120	100121	100122	100123	100124	100125	100126	100127	100128	100129	100130	100131	100132	100133	100134	100135	100136	100137	100138	100139	100140	100141	100142	100143	100144	100145	100146	100147	100148	100149	100150	100151	100152	100153	100154	100155	100156	100157	100158	100159	100160	100161	100162	100163	100164	100165	100166	100167	100168	100169	100170	100171	100172	100173	100174	100175	100176	100177	100178	100179	100180	100181	100182	100183	100184	100185	100186	100187	100188	100189	100190	100191	100192	100193	100194	100195	100196	100197	100198	100199	100200	100201	100202	100203	100204	100205	100206	100207	100208	100209	100210	100211	100212	100213	100214	100215	100216	100217	100218	100219	100220	100221	100222	100223	100224	100225	100226	100227	100228	100229	100230	100231	100232	100233	100234	100235	100236	100237	100238	100239	100240	100241	100242	100243	100244	100245	100246	100247	100248	100249	100250	100251	100252	100253	100254	100255	100256	100257	100258	100259	100260	100261	100262	100263	100264	100265	100266	100267	100268	100269	100270	100271	100272	100273	100274	100275	100276	100277	100278	100279	100280	100281	100282	100283	100284	100285	100286	100287	100288	100289	100290	100291	100292	100293	100294	100295	100296	100297	100298	100299	100300	100301	100302	

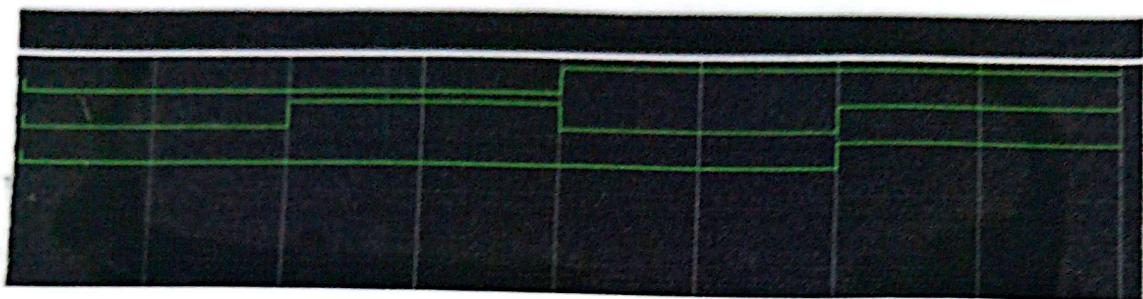


TITLE

Verilog Code for behavioural model  
module mux2x1(input I0,I1,s,output reg Y);  
always @ (c+)  
begin  
if (s)  
Y = I1;  
else  
Y = I0;  
end  
endmodule



### Dataflow model



• Dataflow model  
• Data flow graph  
• Directed acyclic graph  
• Parallel execution  
• Pipelining  
• Pipeline factors  
• Pipeline length  
• Pipeline width  
• Pipeline latency  
• Pipeline efficiency  
• Pipeline overhead  
• Pipeline cost  
• Pipeline complexity  
• Pipeline performance

• Pipeline width = number of parallel stages

• Pipeline length

• Pipeline latency = sum of pipeline stages + number of stages - 1