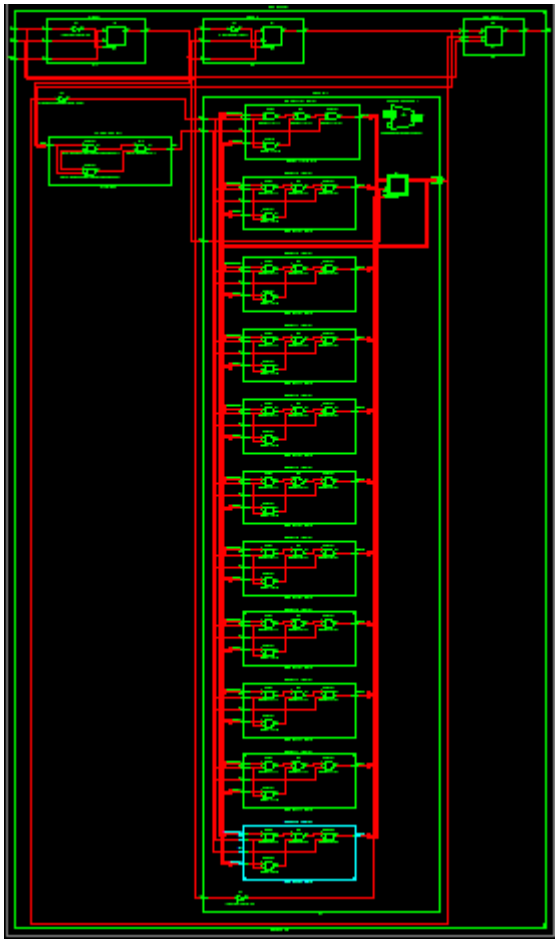


Solutions to the given Assignment Questions :

1 . Ans :

8-bit Register

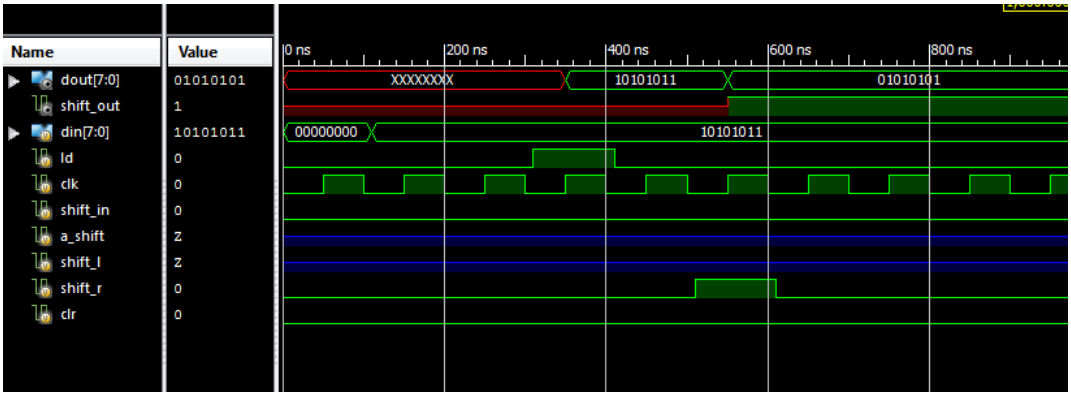
Verilog Code

```
1
2  //----------------------------------Register--------------------------------------
3
4  `timescale 1ns / 1ps
5  module Multi_register(dout,shift_out,din,ld,clk,shift_in,a_shift,shift_l,shift_r,clr);
6
7  parameter N = 8;
8
9  input [N-1:0] din;
10 input ld,clk,shift_in,shift_l,shift_r,a_shift,clr;
11 output reg shift_out;
12 output reg [N-1:0] dout;
13
14 always @(posedge clk)
15 begin
16
17 if(clr == 1'b1) dout <= 8'b0;
```

```
18
19  else if(ld) dout <= din;
20
21  else if(a_shift == 1'b1)
22  begin
23  dout <= {dout[N-1],dout[N-1:1]};
24  shift_out <= dout[0];
25  end
26
27  else if(shift_l == 1'b1)
28  begin
29  dout <= {dout[N-2:0],shift_in};
30  shift_out <= dout[N-1];
31  end
32
33  else if(shift_r == 1'b1)
34  begin
35  dout <= {shift_in,dout[N-1:1]};
36  shift_out <= dout[0];
37  end
38
39  else dout <= dout;
40
41  end
42
43  endmodule
44
45  //---------------------------TestBench---------------------------------------
46
47  `timescale 1ns / 1ps
48  module tst;
49
50  // Inputs
51  reg [7:0] din;
52  reg ld;
53  reg clk;
54  reg shift_in;
55  reg a_shift;
56  reg shift_l;
57  reg shift_r;
58  reg clr;
59
60  // Outputs
61  wire [7:0] dout;
62  wire shift_out;
63
64  // Instantiate the Unit Under Test (UUT)
65  Multi_register uut (
66  .dout(dout),
67  .shift_out(shift_out),
68  .din(din),
69  .ld(ld),
70  .clk(clk),
71  .shift_in(shift_in),
72  .a_shift(a_shift),
73  .shift_l(shift_l),
74  .shift_r(shift_r),
75  .clr(clr)
76  );
```

```
77
78  initial begin
79
80  clk= 0;
81  forever #50 clk = ~clk;
82
83  end
84
85  initial begin
86  // Initialize Inputs
87  din = 0;
88  ld = 0;
89  shift_in = 0;
90  a_shift = 1'bz;
91  shift_l = 1'bz;
92  shift_r = 0;
93  clr = 0;
94
95  // Wait 100 ns for global reset to finish
96  //Q,s_out_Q,m_in_2,ldQ,clk,s_out_A,NC,NC,shiftQ,clrQ
97  //dout,shift_out,din,ld,clk,shift_in,a_shift,shift_l,shift_r,clr
98  #100;
99  #10
100 din = 8'b10101011;
101 #100
102 clr = 1'b0;
103 #100
104 ld = 1'b1;
105 #100
106 ld = 1'b0;
107 #100
108 shift_r = 1'b1;
109 #100
110 shift_r = 1'b0;
111
112 // Add stimulus here
113
114 end
115
116 endmodule
```

Simulation Results



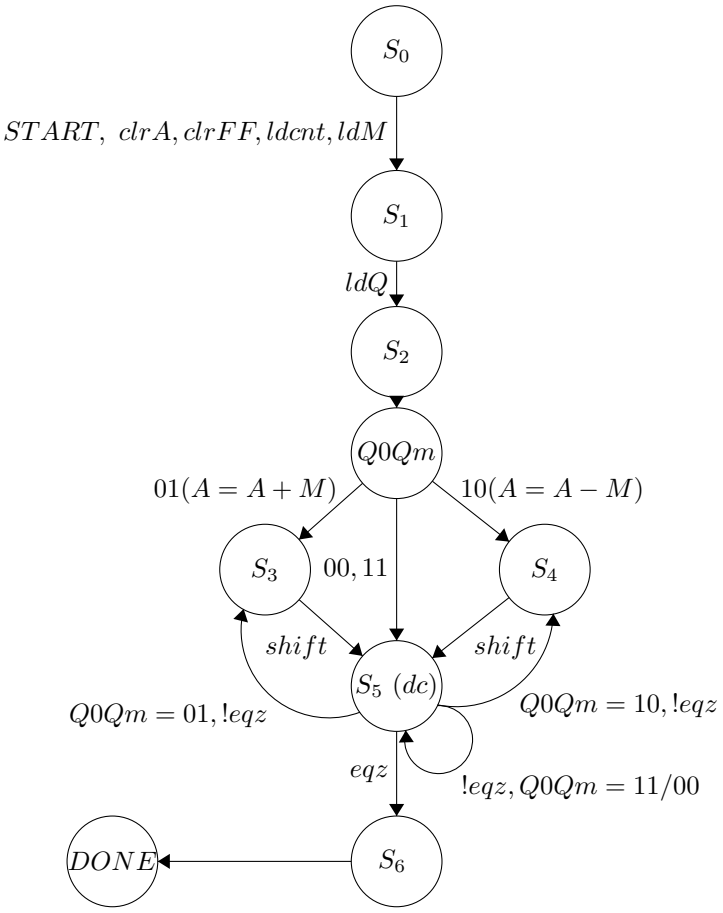
Example of Usage

Please note that this register can be used in multiplication and division using various algorithms. Here we demonstrate multiplication *Booth's Algorithm*.

8 bit Multiplication using *Booth's Algorithm*

The design of 8 bit Booth Multiplier using Datapath and Control Design is presented as follows:

State Diagram



Verilog Code

```
//----------------------------------COUNTER-----------------------

`timescale 1ns / 1ps
module Counter(out,ldcnt,clk,decr);

input ldcnt,clk,decr;
output reg [3:0] out;

always @(posedge clk)
begin

if(ldcnt == 1'b1) out <= 4'b1001;
```

```
13 else if(decr == 1'b1) out <= out-1;
14
15 end
16
17 endmodule
18
19
20
21 //-----------------------------------ALU-------------------------
22
23 `timescale 1ns / 1ps
24 module ALU(out,in1,in2,addsub);
25
26 parameter N =8;
27 input [N-1:0] in1,in2;
28 input addsub;
29
30 output reg [N-1:0] out;
31
32 always @(*)
33 begin
34
35 if(addsub == 1'b0) out <= in1 - in2;
36 else out <= in1 + in2;
37
38 end
39
40
41 endmodule
42 //-----------------------------------REGISTER-------------------------
43
44 `timescale 1ns / 1ps
45 module Multi_register(dout,shift_out,din,ld,clk,shift_in,a_shift,shift_l,shift_r,clr);
46
47 parameter N = 8;
48
49 input [N-1:0] din;
50 input ld,clk,shift_in,shift_l,shift_r,a_shift,clr;
51 output reg shift_out;
52 output reg [N-1:0] dout;
53
54 always @(posedge clk)
55 begin
56
57 if(clr == 1'b1) dout <= 8'b0;
58
59 else if(ld) dout <= din;
60
61 else if(a_shift == 1'b1)
62 begin
63 dout <= {dout[N-1],dout[N-1:1]};
64 shift_out <= dout[0];
65 end
66
67 else if(shift_l == 1'b1)
68 begin
69 dout <= {dout[N-2:0],shift_in};
70 shift_out <= dout[N-1];
71 end
```

```
72
73  else if(shift_r == 1'b1)
74  begin
75  dout <= {shift_in,dout[N-1:1]};
76  shift_out <= dout[0];
77  end
78
79  else dout <= dout;
80
81  end
82
83  endmodule
84  //----------------------------------D FLIPFLOP-------------------------
85
86  `timescale 1ns / 1ps
87  module D_FF(Q,D,clk,clr);
88
89  input D,clk,clr;
90  output reg Q;
91
92  always @(posedge clk)
93  begin
94
95  if(clr == 1'b1) Q <= 1'b0;
96
97  else Q <= D;
98
99  end
100
101 endmodule
102
103 //----------------------------------DATAPATH-------------------------
104
105 `timescale 1ns / 1ps
106 module mul_datapath(out_A,out_Q,Qm,eqz,Q0,ldA,ldQ,clrA,shiftA,
107 clrQ,shiftQ,ldM,clrFF,addsub,clk,m_in_1,m_in_2,ldcnt,decr);
108
109 output Qm,Q0,eqz;
110 parameter N = 8;
111 parameter NC = 1'bz;
112 output [N-1:0] out_A,out_Q;
113 input [N-1:0] m_in_1,m_in_2;
114 input ldA,ldQ,ldM,clrA,clrQ,clrFF,shiftA,shiftQ,clk,addsub,ldcnt,decr;
115 wire [N-1:0] A,M,Q,Z;
116 wire [3:0] count;
117 wire s_out_A,s_out_Q,dum;
118
119 assign Q0 = Q[0];
120 assign eqz = ~|count;
121 assign out_A = A;
122 assign out_Q = Q;
123
124 Multi_register A1(A,s_out_A,Z,ldA,clk,NC,shiftA,NC,NC,clrA);
125 Multi_register Q1(Q,s_out_Q,m_in_2,ldQ,clk,s_out_A,NC,NC,shiftQ,clrQ);
126
127 D_FF RQ(Qm,s_out_Q,clk,clrFF);
128
129 Multi_register M1(M,dum,m_in_1,ldM,clk,NC,NC,NC,NC,NC);
130
```

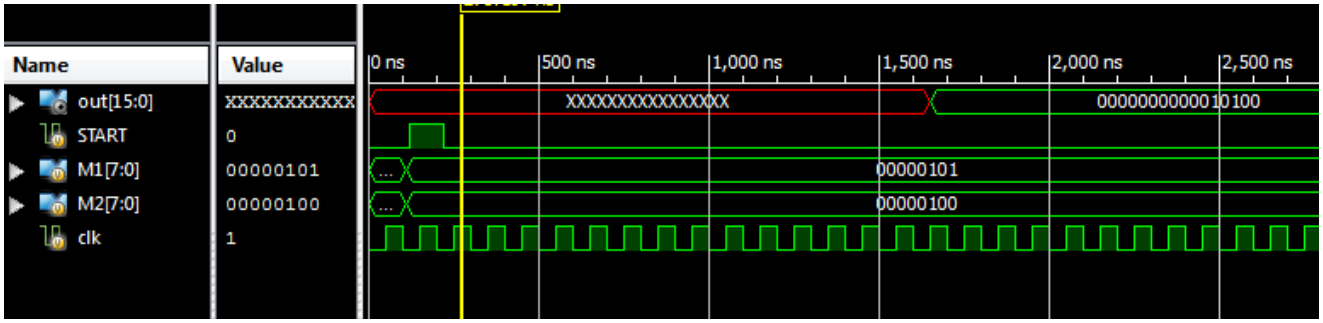
```
131 ALU Alu1(Z,A,M,addsub);
132
133 Counter C1(count,ldcnt,clk,decr);
134
135 endmodule
136
137
138
139
140
141 //-----------------------------------CONTROLPATH-----------------
142
143 `timescale 1ns / 1ps
144 module mul_controlpath(ldA,ldQ,DONE,
145 ldcnt,decr,ldM,clrA,clrQ,clrFF,addsub,shiftA,shiftQ,Q0,Qm,eqz,clk,START);
146
147 input Q0,Qm,clk,eqz,START;
148 output reg ldA,ldQ,ldcnt,ldM,decr,addsub,DONE,clrA,clrQ,clrFF,shiftA,shiftQ;
149
150 reg [2:0] state;
151
152 parameter S0 = 3'b000,S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4 = 3'b100, S5 = 3'b101,S6 = 3'b110;
153
154 always @(posedge clk)
155
156 begin
157
158 case(state)
159
160
161 S0 : if(START) state <= S1;
162 S1 : state <= S2;
163 S2 :  begin
164 case({Q0,Qm})
165
166 2'b10 : state <= S4;
167 2'b01 : state <= S3;
168 default : state <= S5;
169 endcase
170
171 end
172 S3 :  state <= S5;
173 S4 :  state <= S5;
174 S5 : begin
175 if(({Q0,Qm} == 2'b10) && !eqz) state <= S4;
176 else if(({Q0,Qm} == 2'b01) && !eqz) state <= S3;
177 else if((({Q0,Qm} == 2'b11)||({Q0,Qm} == 2'b00)) && !eqz) state <= S5;
178 else if(eqz) state <= S6;
179 end
180
181 S6 : state <= S6;
182
183 default : state <= S0;
184
185 endcase
186
187 end
188
189 always @(state)
```

```
190
191 begin
192
193 case(state)
194
195 S0 : begin
196 clrA = 1'b0;
197 clrQ = 1'b0;
198 clrFF = 1'b0;
199 ldA = 1'b0;
200 ldQ = 1'b0;
201 ldM = 1'b0;
202 decr = 1'b0;
203 shiftA = 1'b0;
204 shiftQ = 1'b0;
205 end
206
207 S1 : begin
208 clrA = 1'b1;
209 clrFF = 1'b1;
210 ldM = 1'b1;
211 ldcnt = 1'b1;
212 end
213
214 S2 : begin
215 clrA = 1'b0;
216 clrFF = 1'b0;
217 ldM = 1'b0;
218 ldcnt = 1'b0;
219 ldQ = 1'b1;
220 end
221
222 S3 : begin
223 addsub = 1'b1;
224 ldA = 1'b1;
225 ldQ = 1'b0;
226 shiftA = 1'b0;
227 shiftQ = 1'b0;
228 decr = 1'b0;
229 end
230
231 S4 : begin
232 addsub = 1'b0;
233 ldA = 1'b1;
234 ldQ = 1'b0;
235 shiftA = 1'b0;
236 shiftQ = 1'b0;
237 decr = 1'b0;
238 end
239
240 S5 : begin
241 ldA = 1'b0;
242 ldQ = 1'b0;
243 shiftA = 1'b1;
244 shiftQ = 1'b1;
245 decr = 1'b1;
246 end
247
248 S6 :    begin
```

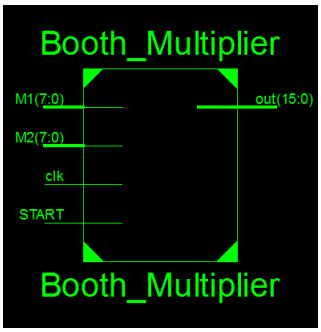
```
DONE  = 1'b1;
decr= 1'b0;
shiftA= 1'b0;
shiftQ= 1'b0;
ldA= 1'b0;
end


default : begin
clrA = 1'b0;
shiftA = 1'b0;
ldQ = 1'b0;
shiftQ = 1'b0;
end
endcase

end


endmodule
//--------------------------------MAIN MODULE-------------------------

`timescale 1ns / 1ps
module Booth_Multiplier(out,START,M1,M2,clk);

parameter N = 8;
output reg [2*N-1:0] out;
input [N-1:0] M1,M2;
input clk,START;


wire ldA,ldQ,ldM,shiftA,shiftQ,ldcnt,decr,clrA,clrQ,clrFF,addsub,DONE,Q0,QM,eqz;
wire [N-1:0] out1,out2;

mul_controlpath CON(ldA,ldQ,DONE,ldcnt,decr,ldM,clrA,clrQ,
clrFF,addsub,shiftA,shiftQ,Q0,Qm,eqz,clk,START);
mul_datapath DAT(out1,out2,Qm,eqz,Q0,ldA,ldQ,clrA,
shiftA,clrQ,shiftQ,ldM,clrFF,addsub,clk,M1,M2,ldcnt,decr);

always @(posedge clk)
begin

if(DONE == 1'b1) out <= {out1,out2};

end

endmodule
//---------------------------TEST BENCH-------------------------

`timescale 1ns / 1ps
module TEST;

// Inputs
reg START;
reg [7:0] M1;
reg [7:0] M2;
reg clk;

// Outputs
```

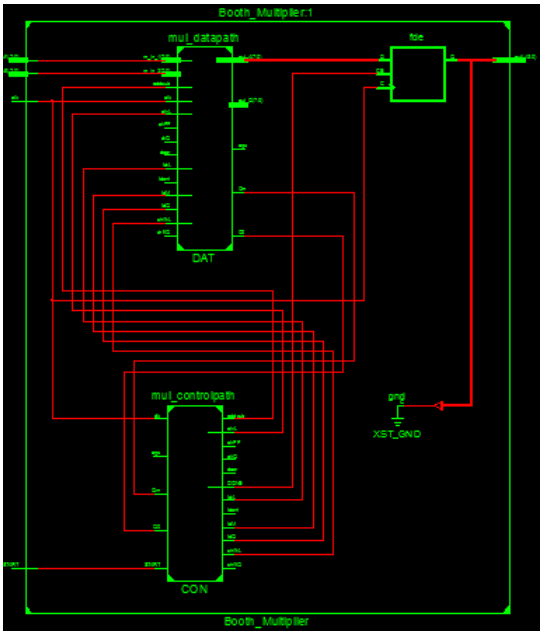
```
wire [15:0] out;

// Instantiate the Unit Under Test (UUT)
Booth_Multiplier uut (
.out(out),
.START(START),
.M1(M1),
.M2(M2),
.clk(clk)
);

initial begin
clk = 0;
forever #50 clk = ~clk;

end

initial begin
// Initialize Inputs
START = 0;
M1 = 0;
M2 = 0;


// Wait 100 ns for global reset to finish
#100;
#10

M1 = 8'b00000101;
M2 = 8'b00000100;

#10

START = 1'b1;

#100
START = 1'b0;


// Add stimulus here

end

endmodule
```

Simulation Results :



Schematic :





Please note the register can be similarly used for Multiplication and Division using various other algorithms.