8x8 bit multiplier

The algorithms :

One of the most Effective algorithms for Multiplication is the Russian Peasant Multiplication algorithm .

The problem: Compute the product of two 8 bit positive integers

Can be solved by this algorithm based on the following formulas:

For even values of n:
n * m = n/2 * 2m

For odd values of n:
if n > 1 → n * m = (n – 1)/2* 2m + m
if n = 1 → n * m = m

Compute :

20 * 26

n	**m**	
20	26	
10	52	
5	104	104
2	208	+
1	416	= 416
		= **520**

Note: Method reduces to adding m's values corresponding to odd n's.

A single left shift multiplies a binary number by 2:
0010 << 1 → 0100
0010 is 2
0100 is 4

a single right shift divides a number by 2, throwing out any remainders :
0101 >>> 1 → 0010
0101 is 5
0010 is 2

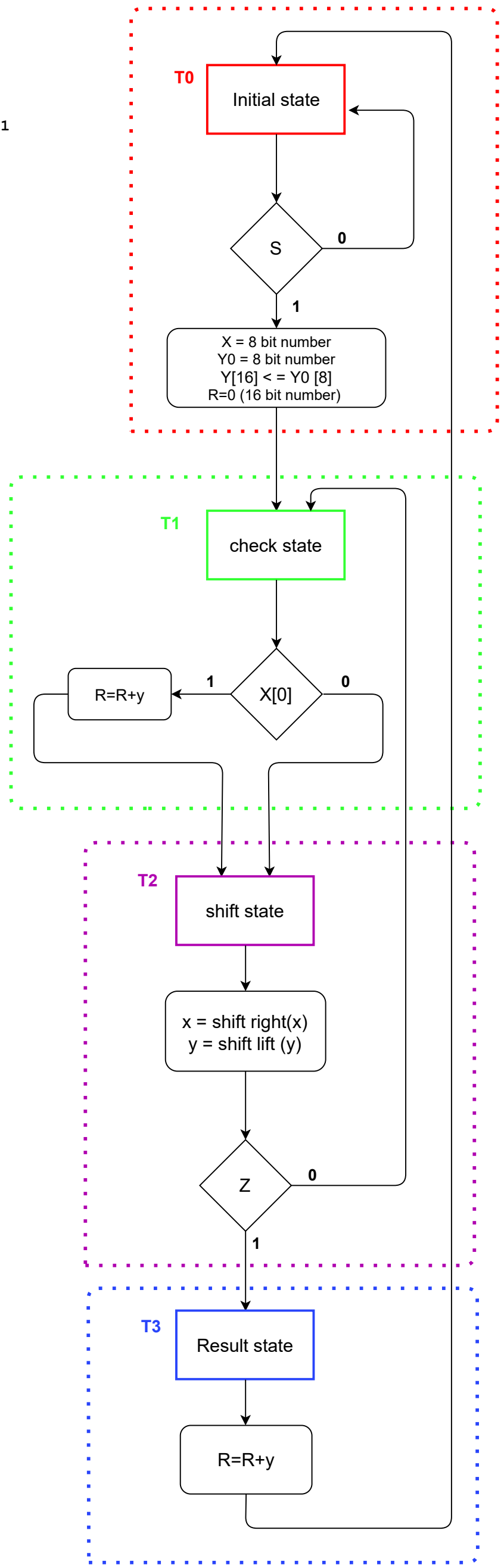
ASM Chart

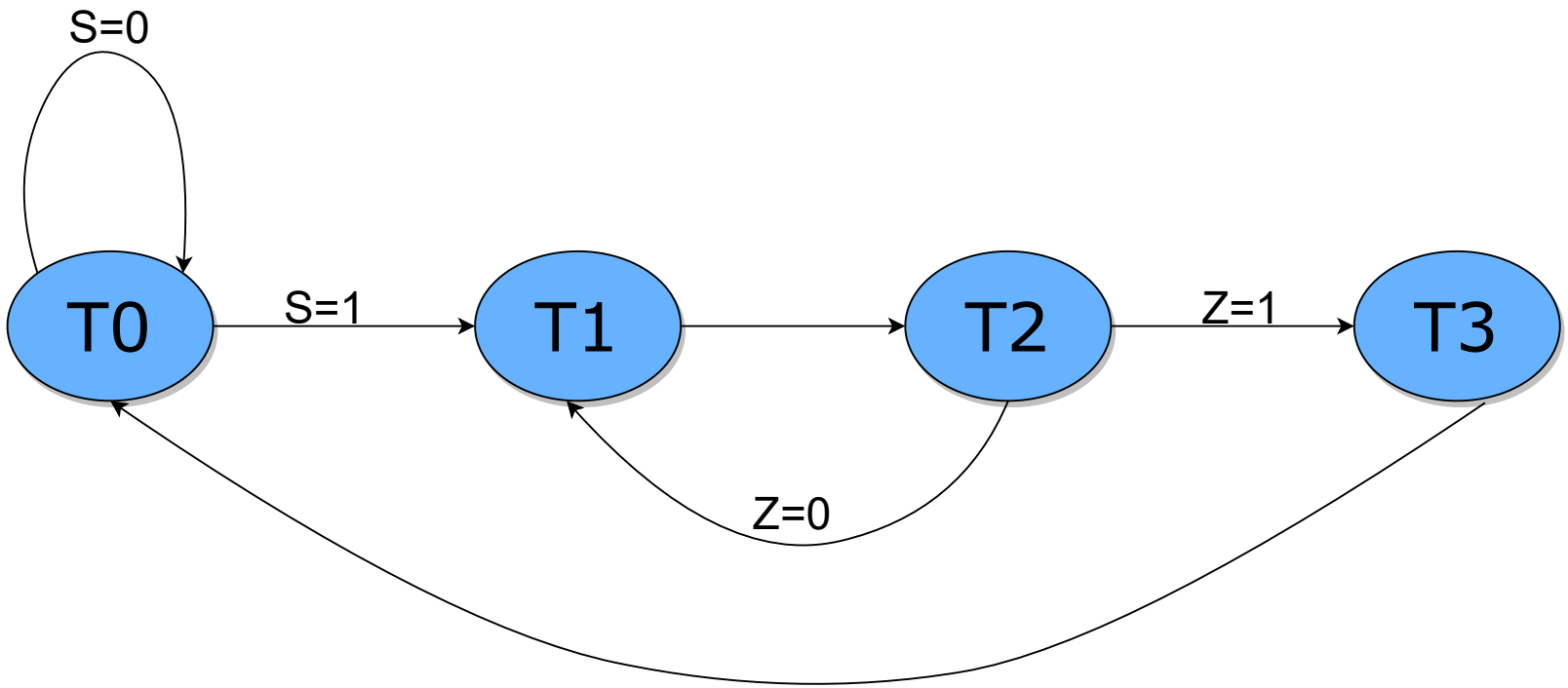
```
X[8] = 8 bit number
Y0 [8] = 8 bit number

Y[16] < = Y0 [8]

R[16] (result 16 bit )

Z flag = 1 if x=00000001
         else z=0
```



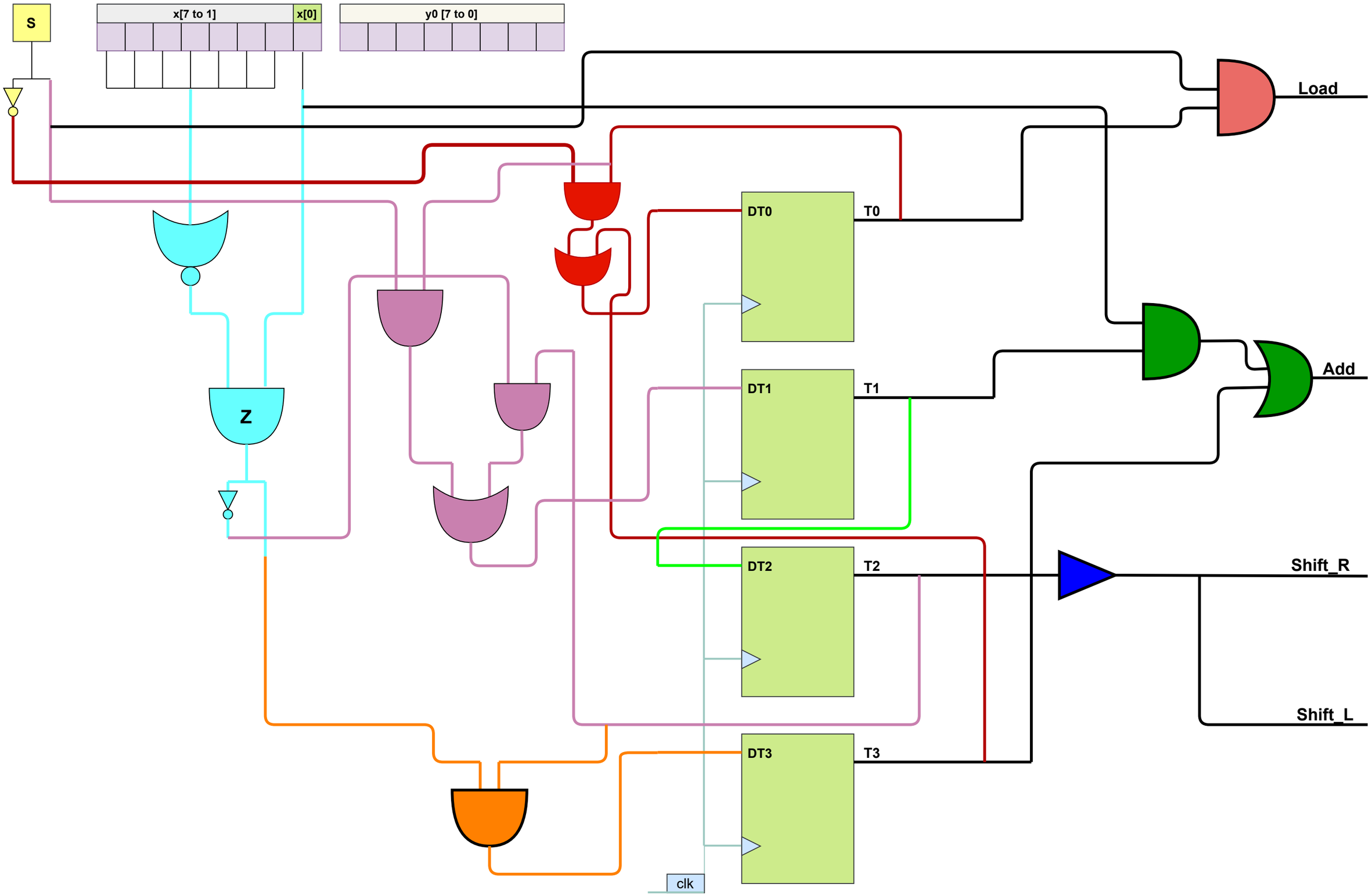


DT0=T0.S'+T3
DT1=T0.S+T2.Z'
DT2=T1
DT3=T2.Z

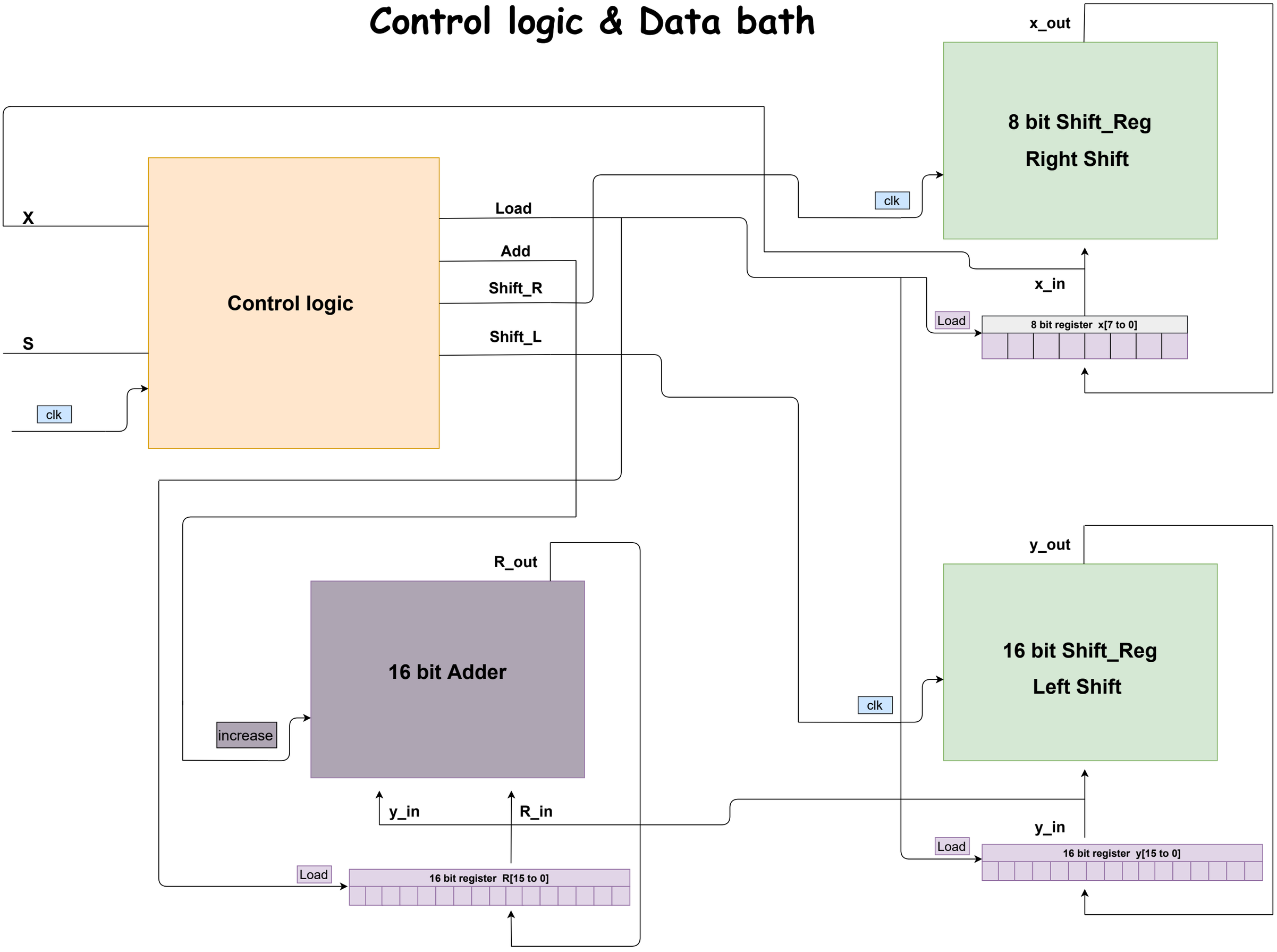
Control Signals

Load=T0.S
Add=T1.x[0]+T3
Shift_R=T2
Shift_L=T2

Control logic



Control logic & Data bath



```
// Code your design here
// Code your design here


module Binary_8_bit_Multiplier (output reg [15:0] R0 ,input   reg s,  reg [7:0] x,  reg [7:0] y0 , reg clk);

  reg [7:0] x1;
  reg [15:0] y1=y0;
  reg [15:0] R1 ;


  reg [15:0] y = 16'b0000000000000000;
  reg [15:0] R = 16'b0000000000000000;

  reg load;
  reg add;
  reg shift_r;
  reg shift_l;

  always @ ( posedge load )
    begin
     y=y0;
     R0=R;
    end

   always @ ( posedge clk )
     begin
  control_logic c1 (load,add,shift_r,shift_l, s, x,clk);
     end


  always @ ( posedge shift_l )
    begin
    Left_16_bit_Shift_Register c2 ( y1, y, shift_l);
    y=y1;
    end


  always @ ( posedge shift_r )
    begin
    Right_8_bit_Shift_Register c3 ( x1, x, shift_r);
    x=x1;
    end
```

```
46
47
48
49    always @ ( posedge add )
50      begin
51      binary_16_bit_adder c4 (R1,R,y);
52      R=R1;
53      end
54
55  endmodule
56
57
58
59
60
61
62
63  module Right_8_bit_Shift_Register ( output reg [7: 0] x_out, input   reg [7: 0] x_in,  reg CLK);
64
65    reg m = 1'b0;
66
67    always @ ( posedge CLK )
68      x_out <= { m , x_in[7 : 1]}; //right Shift
69
70  endmodule
71
72
73
74  module Left_16_bit_Shift_Register ( output reg [15: 0] y_out, input reg [15: 0] y_in,  reg CLK);
75    reg l= 1'b0;
76    always @ ( posedge CLK )
77      y_out <= {y_in[14: 0], l }; // left Shift
78
79  endmodule
80
81
82
83
84
85  module binary_16_bit_adder (R_out,R_in,y_in);
86     output [15: 0] R_out;
87     input [15: 0] R_in, y_in;
88     assign  R_out= R_in + y_in ;
89  endmodule
90
91
```

```
96
97
98
99  module DFF (Q, D, Clk);
100 output  reg Q;
101 input   reg D;
102 input   reg Clk;
103 always @ ( posedge Clk)
104 Q <= D;
105 endmodule
106
107
108 module control_logic (output  reg load,output reg add,output reg shift_r,output reg shift_l,input reg s, reg [7:0] x, reg clk);
109
110   reg z,t0input,t1input,t2input,t3input;
111
112   reg t0output;
113   reg t1output;
114   reg t2output;
115   reg t3output;
116
117
118
119   assign z=(!(x[7]||x[6]||x[5]||x[4]||x[3]||x[2]||x[1])) && x[0];
120
121   assign t0input=(t0output &&(!s)) ||t3output;
122   assign t1input=(t0output && s) ||(t2output && (!z));
123   assign t2input=t1output;
124   assign t3input=t2output && z;
125
126
127   DFF dt0(t0output,t0input,clk);
128   DFF dt1(t1output,t1input,clk);
129   DFF dt2(t2output,t2input,clk);
130   DFF dt3(t3output,t3input,clk);
131
132
133   assign load=t0output&&s;
134   assign add=(t1output&&x[0])||t3output;
135   assign shift_r=t2output;
136   assign shift_l=t2output;
137
138 endmodule
139
140 // jiha&Mohammad&ahmad
141
```

Languages & Libraries

Testbench + Design

SystemVerilog/Verilog

UVM / OVM

None

Other Libraries

None
OVL 2.8.1
SVUnit 2.11

- [ ] Enable TL-Verilog
- [ ] Enable Easier UVM
- [ ] Enable VUnit

Tools & Simulators

Synopsys VCS 2020.03

Compile Options

-timescale=1ns/1ns +vcs+flush+

Run Options

Run Options

- [ ] Use run.do Tcl file
- [ ] Use run.bash shell script
- [ ] Open EPWave after run if not
- [ ] downloading files after run

Examples

using EDA Playground
VHDL
Verilog/SystemVerilog
UVM
EasierUVM

testbench.sv

SV/Verilog Testbench

```
// Code your testbench here
// or browse Examples

  module t_8_bit_Multiplier;
  reg [7:0] x;
  reg [7:0] y;
  reg [15:0] R;
  reg clk = 1'b0;
  reg s = 1'b0;

 Binary_8_bit_Multiplier m1 ( R ,s,x,y,clk);


initial
forever #1 clk = ~clk;


initial
#5000 $finish ;


initial
  begin
    x=8'b10110100; y=8'b00111101; s=1'b0;R=16'b0000000000000000;
    #500 $display("x=%b,y=%b,R=%b,s=%b \n",x,y,R,s);

    #50 s=1'b1;
    #500 $display("x=%b,y=%b,R=%b,s=%b \n",x,y,R,s);

    #50 x=8'b11110111; y=8'b11101011;
    #500  $display("x=%b,y=%b,R=%b,s=%b \n",x,y,R,s);

    #50 x=8'b11111111; y=8'b11111111;
    #500 $display("x=%b,y=%b,R=%b,s=%b \n",x,y,R,s);

  end


endmodule
```