CSE 331/503
Computer Organization
Homework 3 – ALU with Multiplication Design
REPORT

Burak Çiçek
1901042260

Testbenches:

AND32

```
and_32 a32tb (out, a, b);

]initial begin
a = 32'b01111011111111111111111111111110;
b = 32'b01111111111111111100001111111111;
#`DELAY;

a = 32'b11111111111111111111111111111111;
b = 32'b00000000000000000000000000000000;
#`DELAY;

end
```

```
# time =  0, a =01111011111111111111111111111110, b=01111111111111111100001111111111, , out=01111011111111111100001111111110
# time = 20, a =11111111111111111111111111111111, b=00000000000000000000000000000000, , out=00000000000000000000000000000000
```

Just combined of the 1 bit AND implementations, nothing complicated.

OR32

```
or_32bit or32btb (out, a, b);

]initial begin
a = 32'b01111011111111111111111111111110;
b = 32'b01111111111111111100001111111111;
#`DELAY;

a = 32'b11100000000000000000000000000000;
b = 32'b00000000000000000000000000000111;
#`DELAY;
end

-
]initial begin
$monitor("time = %2d, a =%32b, b=%32b, , out=%32b", $time, a, b,out);
end
```

```
# time =  0, a =01111011111111111111111111111110, b=01111111111111111100001111111111, , out=01111111111111111111111111111111
# time = 20, a =11100000000000000000000000000000, b=00000000000000000000000000000111, , out=11100000000000000000000000000111
```

Just combined of the 1 bit OR implementations, nothing complicated.

NOR32

```
nor_32bit or32btb (out, a, b);

initial begin
a = 32'b01111011111111111111111111111110;
b = 32'b01111111111111111100001111111111;
#`DELAY;

a = 32'b11100000000000000000000000000000;
b = 32'b00000000000000000000000000000111;
#`DELAY;
end
```

```
# time =  0, a =01111011111111111111111111111110, b=01111111111111111100001111111111, , out=10000000000000000000000000000000
# time = 20, a =11100000000000000000000000000000, b=00000000000000000000000000000111, , out=00011111111111111111111111111000
```

Opposite implementation of OR32.

FULL ADDER32

```
full_adder_32bit fa32b (out, carry_out, a, b, carry_in);

initial begin

a = 32'b00000000000000000000000000000110;
b = 32'b00000000000000000000000000000111;

#`DELAY;

a = 32'b00000000000100000100000000000001;
b = 32'b00000000000000000100000000000010;

#`DELAY;



end
```

```
# time =  0, a =00000000000000000000000000000110, b=00000000000000000000000000000111, , out=00000000000000000000000000001101
# time = 20, a =00000000000100000100000000000001, b=00000000000000000100000000000010, , out=00000000000100001000000000000011
```

Just combined of the 1 bit Full Adder implementations, nothing complicated.

NOT32

```
not_32bit n32tb (out, a);

initial begin
a = 32'b11111111111111111111111111111111;
#`DELAY;

a = 32'b00000000000000000000000000000000;
#`DELAY;

end
```

```
# time =  0, a =11111111111111111111111111111111, out=00000000000000000000000000000000
# time = 20, a =00000000000000000000000000000000, out=11111111111111111111111111111111
```

Just NOT32 implementation, not big deal.

2x1MUX32

```
mux2_gate_32 muxx(a,b,Sel,Out) ;

initial begin
Sel=0;
a = 32'b00000000000000000000000000000111;// if 0 select a
b = 32'b00000000000000000000000000000110;
#`DELAY;
Sel=1;
a = 32'b00000000000000000000000000000111;//otherwise select b.
b = 32'b00000000000000000000000000000110;
#`DELAY;

end
```

```
# time =  0 , Out=00000000000000000000000000000111
# time = 20 , Out=00000000000000000000000000000110
```

4x1MUX32

```
mux4_32 m0(a,b,c,d,Sel,Out);
initial begin
Sel=2'b00;
a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000000000110;
c = 32'b00000001110000000000000000000110;
d = 32'b11100000000000000000000000000110;
#`DELAY;
Sel=2'b01;
a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000000000110;
c = 32'b00000001110000000000000000000110;
d = 32'b11100000000000000000000000000110;
#`DELAY;

Sel=2'b10;
a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000000000110;
c = 32'b00000001110000000000000000000110;
d = 32'b11100000000000000000000000000110;
#`DELAY;

Sel=2'b11;
a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000000000110;
c = 32'b00000001110000000000000000000110;
d = 32'b11100000000000000000000000000110;
#`DELAY;
end
```

```
# time =  0 , Out=00000000000000000000000000000111
# time = 20 , Out=00000000000000000000000000000110
# time = 40 , Out=00000001110000000000000000000110
# time = 60 , Out=11100000000000000000000000000110
```

8x1MUX32

```
mux8_32 m8tb32(a,b,c,d,e,f,g,h,Sel,Out);
initial begin
Sel=3'b100;
a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000000000110;
c = 32'b00000001110000000000000000000110;
d = 32'b11100000000000000000000000000110;
e = 32'b11100000000000000000000000000110;
f = 32'b11100000000000000000000000000110;
g = 32'b11100000000000000000000000000110;
h = 32'b11100000000000000000000000000110;
#`DELAY;
Sel=3'b111;
a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000000000110;
c = 32'b00000001110000000000000000000110;
d = 32'b11100000000000000000000000000110;
e = 32'b11100000000011111110000000000110;
f = 32'b11100011100000000000000000000110;
g = 32'b11100000000000000000111000000110;
h = 32'b11111111111111111000000000000110;
#`DELAY;

end
```

```
# time =  0 , Out=11100000000000000000000000000110
# time = 20 , Out=11111111111111111000000000000110
```

2x1MUX64

```
mux2_gate_64 muxx(a,b,Sel,Out) ;

initial begin
Sel=0;
a = 64'b0000000000000000000000000000011100000000000000000000000000000111;// if 0 select a
b = 64'b0000000000000000000000000000011000000000000000000000000000000110;
#`DELAY;
Sel=1;
a = 64'b0000000000000000000000000000011100000000000000000000000000000111;//otherwise select b.
b = 64'b0000000000000000000000000000011000000000000000000000000000000110;
#`DELAY;

end
```

```
# time =  0 , Out=0000000000000000000000000000011100000000000000000000000000000111
# time = 20 , Out=0000000000000000000000000000011000000000000000000000000000000110
```

SHIFTR32

```
shiftr_32bit sr32b (out, a);

initial begin

a = 32'b00000000000000000000000000000110;

#`DELAY;

a = 32'b10000000000000000000000000010110;


end
```

```
# time =  0, a =00000000000000000000000000000110, out=00000000000000000000000000000011
# time = 20, a =10000000000000000000000000010110, out=01000000000000000000000000001011
```

SHIFTR64

```
shiftr_64bit sr64b (out, a);

initial begin

a = 64'b0000000000000000000000000000000100000000000000000000000000000000;

#`DELAY;

a = 64'b0000011100000000000000000000000100000000000000000000000000000000;

#`DELAY;
end
```

```
# time =  0, a =0000000000000000000000000000000100000000000000000000000000000000, out=0000000000000000000000000000000100000000000000000000000000000000
# time = 20, a =0000011100000000000000000000000100000000000000000000000000000000, out=0000011100000000000000000000000100000000000000000000000000000000
```

SLT32

```
slt_32b slt32btb (a, b,out);

initial begin
a = 32'b01111011111111111111111111111110;
b = 32'b01111111111111111110000111111111;
#`DELAY;

a = 32'b01100000000000000000000000000000;
b = 32'b00000000000000000000000000000111;
#`DELAY;
end
```

```
# time =  0, a =01111011111111111111111111111110, b=01111111111111111110000111111111, , out=00000000000000000000000000000001
# time = 20, a =01100000000000000000000000000000, b=00000000000000000000000000000111, , out=00000000000000000000000000000000
```

MULT32

```
control mtest(a,b,result);

initial begin

a = 32'b00000010000000100000100000000111;
b = 32'b00000000000000000000000000000100;

#`DELAY;

a = 32'b00000000000000000000100000000000;
b = 32'b00011111111100000000000000000000;

#`DELAY;
end
```

```
# time =  0, a =00000010000000100000100000000111, b=00000000000000000000000000000100, , result=00000000000000000000000000000001000000010000010000000111000
# time = 20, a =00000000000000000000100000000000, b=00011111111100000000000000000000, , result=00000000000000000000001111111100000000000000000000000000000000
```

SUBT32

```
subt_32bit subii (out, a, b);

initial begin

a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000000000110;

#`DELAY;

a = 32'b00000000000000000000000000000111;
b = 32'b00000000000000000000000001110000;

#`DELAY;


end
```

```
# time =  0, a =00000000000000000000000000000111, b=00000000000000000000000000000110, , out=00000000000000000000000000000001
# time = 20, a =00000000000000000000000000000111, b=00000000000000000000000001110000, , out=11111111111111111111111110010111
```

ALU32

```
alu_32bit alu(R, A, B, S);

initial begin

// ADD
S = 3'b000;
A = 32'b00000000000000000000000000000001;
B = 32'b00000000000000000000000000000010;
#`DELAY;

A = 32'b00000000000000000000000000110001;
B = 32'b00000000000000000000000000110010;
#`DELAY;

// XOR
S = 3'b001;
A = 32'b11111111111111111111111111111111;
B = 32'b00000000000000000000000000000000;
#`DELAY;

A = 32'b01110000000011111000000000000000;
B = 32'b01110001111100000000000000000000;

#`DELAY;

// SUB
S = 3'b010;
A = 32'b00000000000000000000000000011000;
B = 32'b00000000000000000000000000000100;
#`DELAY;
```

```
A = 32'b00000000000000000000000000011000;
B = 32'b00000000000000000000000010000100;
#`DELAY;

//MULT
S = 3'b011;
A = 32'b01010101010101010101010101010101;
B = 32'b10101010101010101010101010101010;
#`DELAY;
A = 32'b00011011100111111111000011100101O;
B = 32'b11001101100000001111110000000111;
#`DELAY;

// SLT
S = 3'b100;
A = 32'b00000000000000000000000000000000;
B = 32'b00000000000000000000000000000001;
#`DELAY;
A = 32'b00000000000000000000000000000001;
B = 32'b00000000000000000000000000000000;
#`DELAY;

// NOR
S = 3'b101;
A = 32'b00000000000000000000000000000001;
B = 32'b00000000000000000000000000000001;
#`DELAY;
A = 32'b00000000000000000000000000000001;
B = 32'b00000000000000000000000000000000;
#`DELAY;

// AND
S = 3'b110;
A = 32'b00000000000000000000000000010101;
B = 32'b00000000000000000000000000010101;
#`DELAY;
A = 32'b00000000000000000000000000000001;
B = 32'b00000000000000000000000000000000;
#`DELAY;


// OR
S = 3'b111;
A = 32'b00000000000000000000000000000011;
B = 32'b00000000000000000000000000000101;
#`DELAY;
A = 32'b00000000000000000000000000011000;
B = 32'b00000000000000000000000000000100;
#`DELAY;
```

```
# time =  0, A = 00000000000000000000000000000001, B=00000000000000000000000000000010, R=00000000000000000000000000000011, S=000
# time = 20, A = 00000000000000000000000000110001, B=00000000000000000000000000110010, R=00000000000000000000000001100011, S=000
# time = 40, A = 11111111111111111111111111111111, B=00000000000000000000000000000000, R=11111111111111111111111111111111, S=001
# time = 60, A = 01110000000011111000000000000000, B=01110001111100000000000000000000, R=00000001111111111000000000000000, S=001
# time = 80, A = 00000000000000000000000000011000, B=00000000000000000000000000000100, R=00000000000000000000000000010100, S=010
# time = 100, A = 00000000000000000000000000011000, B=00000000000000000000001000000100, R=11111111111111111111111000010100, S=010
# time = 120, A = 00000000000000000000000000000000, B=00000000000000000000000000000001, R=00000000000000000000000000000001, S=100
# time = 140, A = 00000000000000000000000000000001, B=00000000000000000000000000000000, R=00000000000000000000000000000000, S=100
# time = 160, A = 00000000000000000000000000000001, B=00000000000000000000000000000001, R=11111111111111111111111111111110, S=101
# time = 180, A = 00000000000000000000000000000001, B=00000000000000000000000000000000, R=11111111111111111111111111111110, S=101
# time = 200, A = 00000000000000000000000000010101, B=00000000000000000000000000010101, R=00000000000000000000000000010101, S=110
# time = 220, A = 00000000000000000000000000000001, B=00000000000000000000000000000000, R=00000000000000000000000000000000, S=110
# time = 240, A = 00000000000000000000000000000011, B=00000000000000000000000000000101, R=00000000000000000000000000000111, S=111
# time = 260, A = 00000000000000000000000000011000, B=00000000000000000000000000000100, R=00000000000000000000000000011100, S=111


# time =  0, A = 01010101010101010101010101010101, B=10101010101010101010101010101010, R=11100011100011100011100011100101, S=011
```

MULT32.v

```
module mult32(finalProduct, mainProduct, a,b);
input [31:0] a;  // multiplicant
input [31:0] b; // multiplier
input [63:0] mainProduct; // product
wire [63:0] result; // product
wire [63:0] result2; // product
wire [63:0] result3; // product
wire [63:0] result4; // product
output [63:0]finalProduct;
wire carry_out;
reg [31:0] zero = 32'b00000000000000000000000000000000;
wire [31:0]templ;


shiftr_64bit sr64b (result2[63:0], mainProduct[63:0]); // right decision

full_adder_32bit a1(result3[63:32], carry_out, mainProduct[63:32], a, 1'b0);
full_adder_32bit a2(result3[31:0], carry_out, mainProduct[31:0], zero, 1'b0);
shiftr_64bit sr64bl (result4[63:0], result3[63:0]); // left decision


mux2_gate_64 mymux(result2, result4, mainProduct[0],finalProduct);




endmodule
```

I calculate the both result for p0=0 and p0=1 and decide to which will be correct output for the situation. If p[0] = 0 than, I use result2 which is just right shift, that's it.If p[0]=1 than, I add multiplicant to left side and shift right only.

PS:I dont know why but my multiplier component works slow in my machine but in some of my friends machine works well, this condition effects me a little bit. So I made it Comment line part1 related codes If you want to try, you can run Mult32_test_bench or uncomment all related parts in ALU.(In ALU.v and test_bench)

Control.v

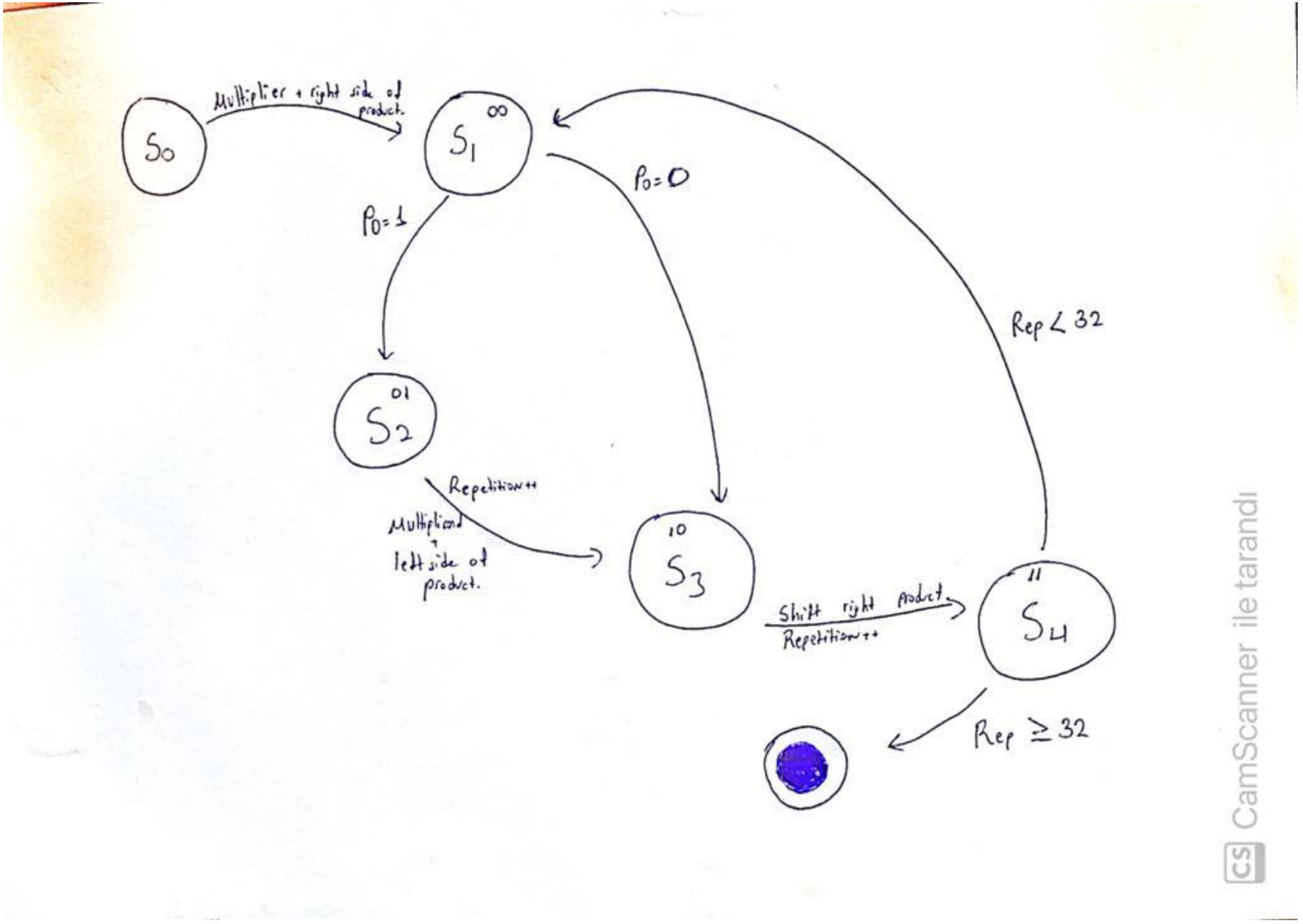
```
module control(
input [31:0] a,  // multiplicant
input [31:0] b, // multiplier
output [63:0]result
);

wire [63:0] mainProduct;
reg [63:0] product = 64'b0000000000000000000000000000000000000000000000000000000000000000;
reg [31:0] zero = 32'b00000000000000000000000000000000;
wire [63:0]temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9,temp10,temp11,temp12,temp13,temp14,temp15,temp16,temp17,temp18,temp19,temp20,temp21,temp22,temp23,temp24,temp25,temp26,temp27,
temp28,temp29,temp30,temp31;

full_adder_32bit multtest(mainProduct[31:0] , carry_out, product[31:0], b, 1'b0);
full_adder_32bit a2(mainProduct[63:32], carry_out, product[63:32],zero,1'b0); // First step for add multiplier to right side.
```

```
mult32 m1(temp1,mainProduct,a,b);
mult32 m2(temp2,temp1,a,b);
mult32 m3(temp3,temp2,a,b);
mult32 m4(temp4,temp3,a,b);
mult32 m5(temp5,temp4,a,b);
mult32 m6(temp6,temp5,a,b);
mult32 m7(temp7,temp6,a,b);
mult32 m8(temp8,temp7,a,b);
mult32 m9(temp9,temp8,a,b);
mult32 m10(temp10,temp9,a,b);
mult32 m11(temp11,temp10,a,b);
mult32 m12(temp12,temp11,a,b);
mult32 m13(temp13,temp12,a,b);
mult32 m14(temp14,temp13,a,b);
mult32 m15(temp15,temp14,a,b);
mult32 m16(temp16,temp15,a,b);
mult32 m17(temp17,temp16,a,b);
mult32 m18(temp18,temp17,a,b);
mult32 m19(temp19,temp18,a,b);
mult32 m20(temp20,temp19,a,b);
mult32 m21(temp21,temp20,a,b);
mult32 m22(temp22,temp21,a,b);
mult32 m23(temp23,temp22,a,b);
mult32 m24(temp24,temp23,a,b);
mult32 m25(temp25,temp24,a,b);
mult32 m26(temp26,temp25,a,b);
mult32 m27(temp27,temp26,a,b);
mult32 m28(temp28,temp27,a,b);
mult32 m29(temp29,temp28,a,b);
mult32 m30(temp30,temp29,a,b);
mult32 m31(temp31,temp30,a,b);
mult32 m32(result,temp30,a,b);
endmodule
```

In my Control.v Verilog code, I started with add multiplier to right side of product, I did it once in Control.v and never do that again. Actually, I didn't find a way to reusability for my outputs more than one so unfortunately, I created 31 temp variable for the calculate 32 repetition. That's the way that I thought maybe -most probably- there was a more efficient way to do this but that's my solution.



P_0	S_0	S_1	S_0+	S_1+
0	0	0	1	0
1	0	0	0	1
0	0	1	X	X
1	0	1	1	0
0	1	0	1	1
1	1	0	1	1
0	1	1	0	0
1	1	1	0	0

S_0+

$P_0'S_0'S_1' + P_0S_0'S_1 + P_0'S_0S_1' + P_0S_0S_1'$

S_1+

$P_0S_0'S_1' + P_0'S_0S_1' + P_0S_0S_1'$

For S_1+

$P_0S_1'(S_0' + S_0) + P_0'S_0S_1'$

→ $\boxed{P_0S_1' + P_0'S_0S_1'}$

For S_0+

$P_0'S_1'(S_0' + S_0) + P_0S_0'S_1 + P_0S_0S_1'$

$\boxed{P_0'S_1' + P_0S_0'S_1 + P_0S_0S_1'}$