**Module: R3: DLD + DSD**

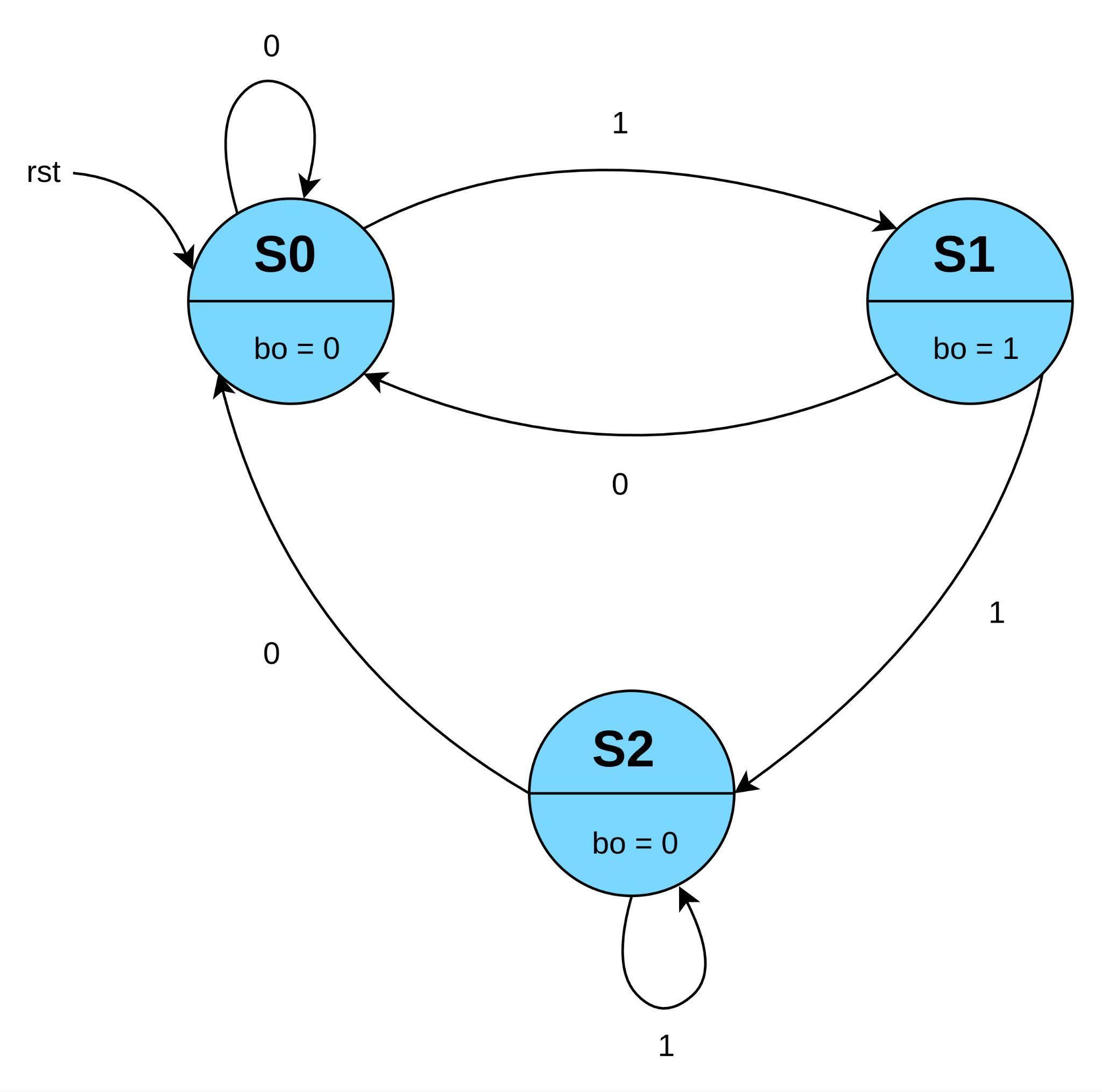
**Section:** Sequential Circuits **Task:** Assignment 2

**Assignment 2**

**Sequential Circuits**

* **Question 1: Design button press synchronizer:**

1. **FSM Diagram:**

****

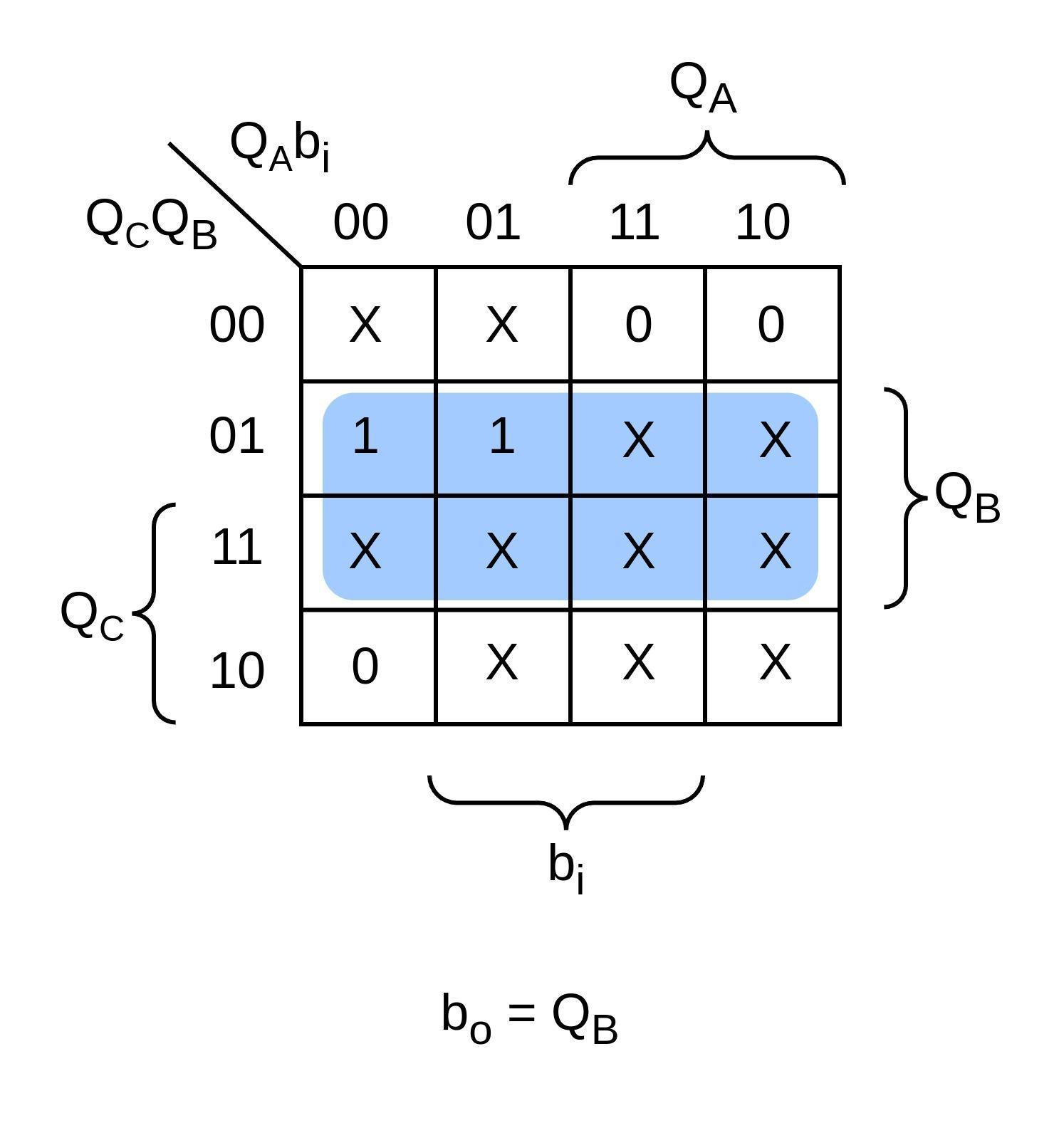
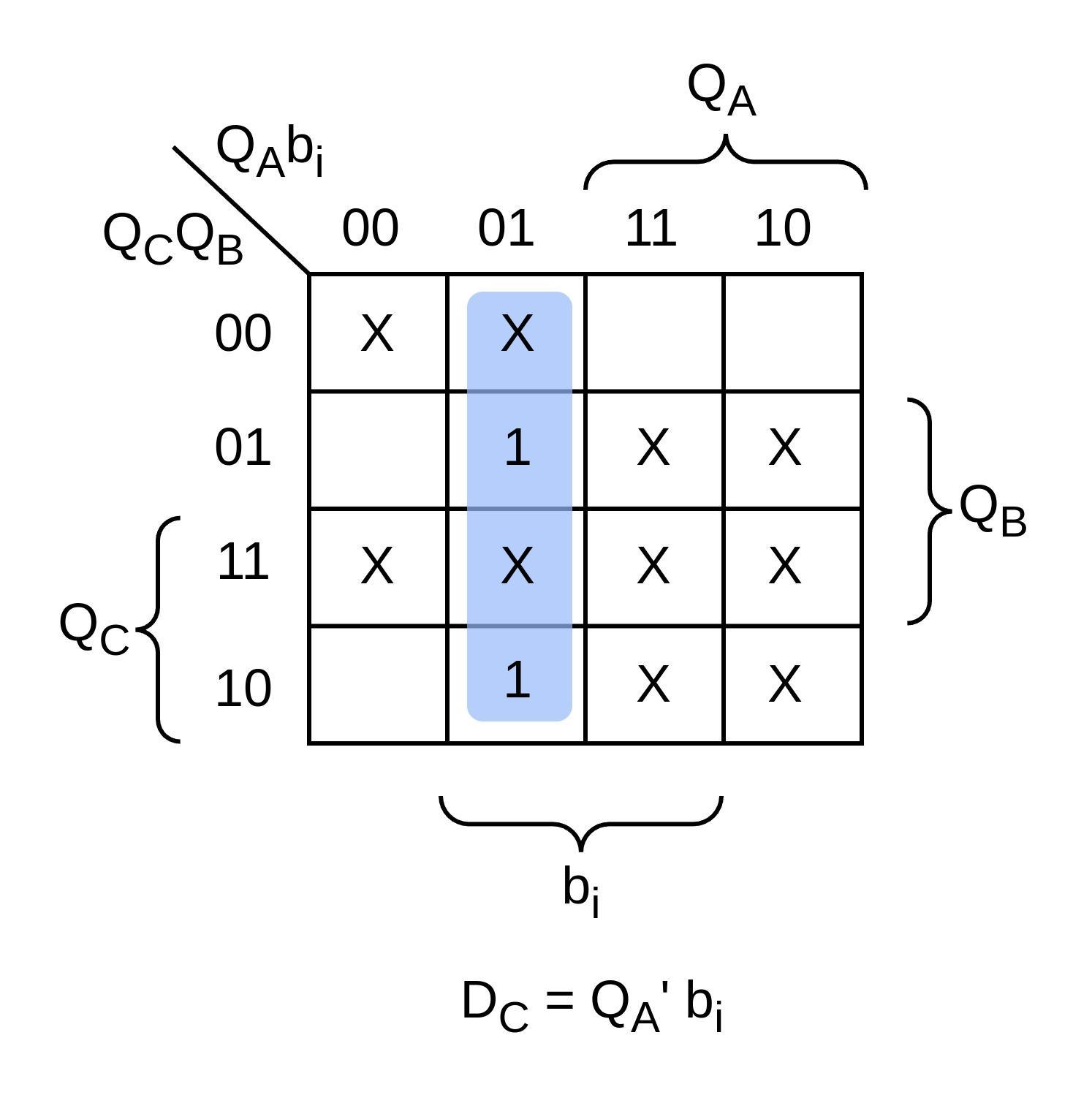
1. **Truth Table:**

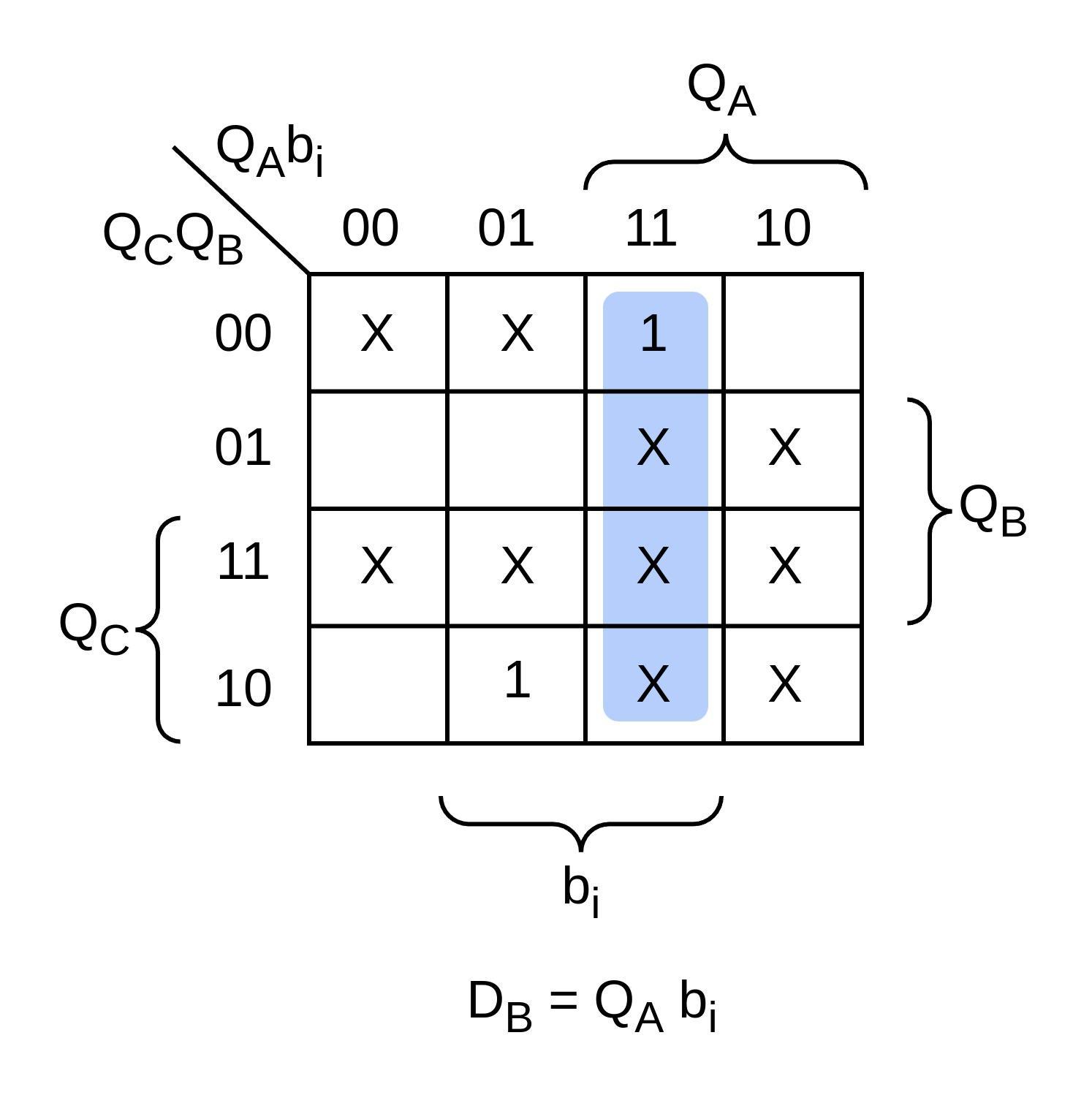
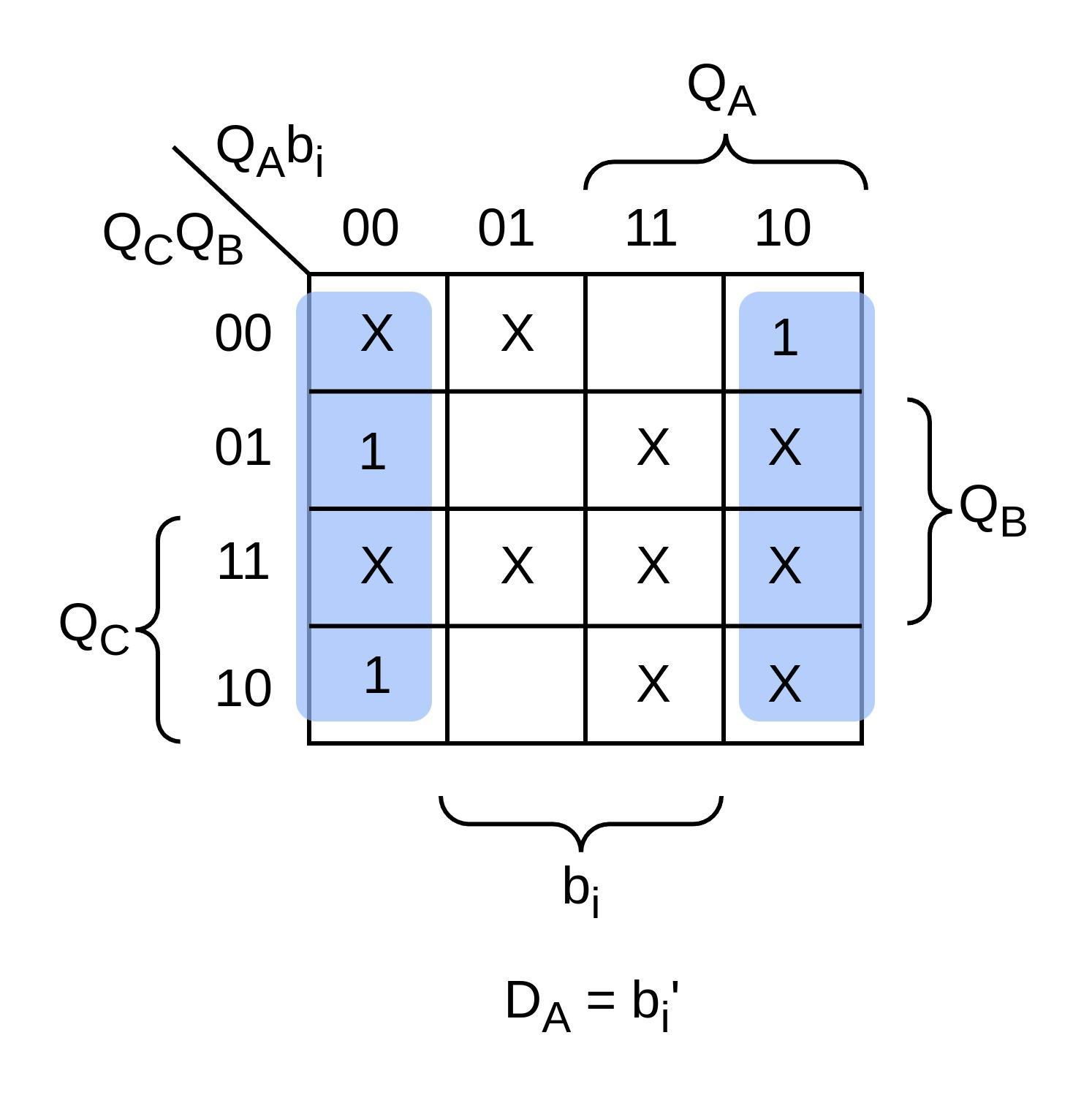
| **One-Hot Encoding** | | **S0 = 001** | **S1 = 010** | **S2 = 100** |
| --- | --- | --- | --- | --- |

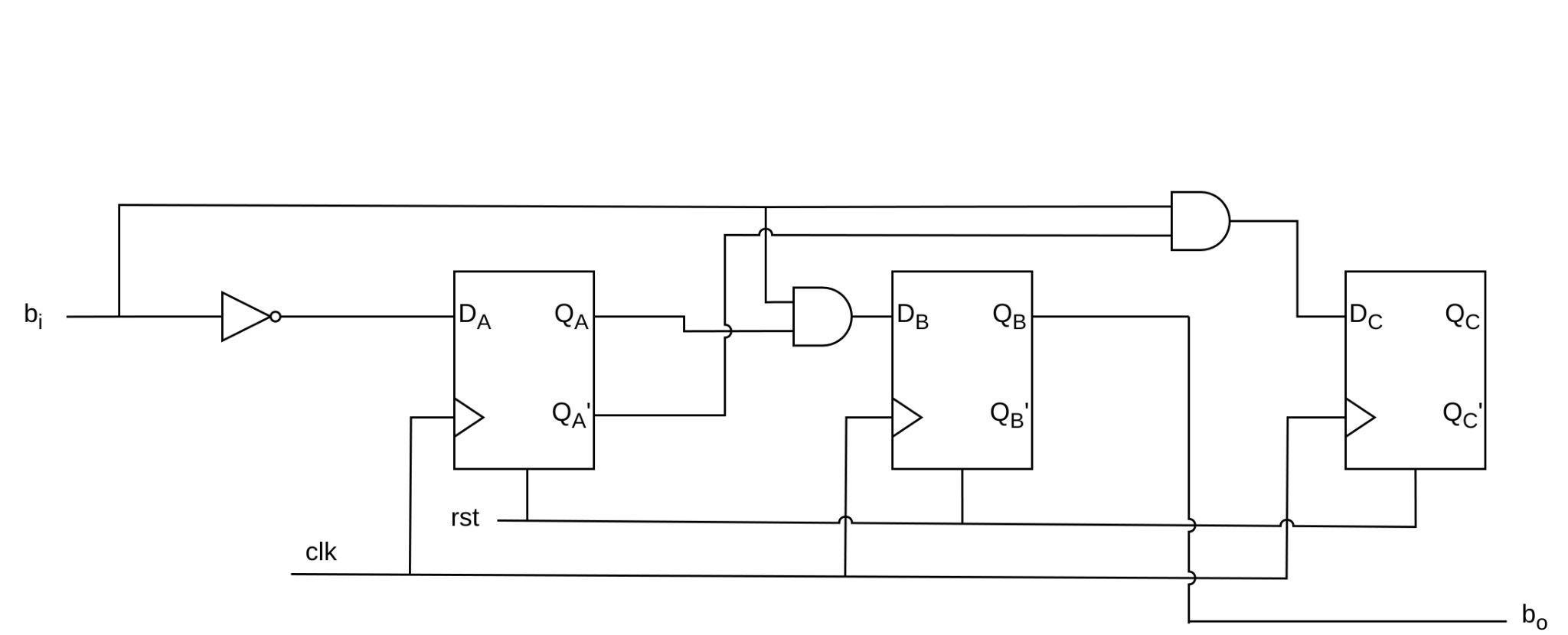
| Present State | | | Input | Next State | | | Output | Inputs to Flip-Flops | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QC | QB | QA | bi | QC+1 | QB+1 | QA+1 | bo | DC | DB | DA |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

1. **Circuit Diagram:**

Using K-Maps:

Here’s the final circuit:

1. **Verilog Code:**

module d\_ff(input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin

if (rst)

q <= 0;

else

q <= d;

end

endmodule

module d\_ff2 (input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin

if (rst)

//LSB = 1 for 1-Hot Encoding

q <= 1;

else

q <= d;

end

endmodule

module button\_press\_synchronizer (input clk, rst, bi, output reg bo);

wire Da, Db, Dc, Qa, Qb, Qc;

d\_ff2 A(.d(Da), .clk(clk), .rst(rst), .q(Qa));

d\_ff B(.d(Db), .clk(clk), .rst(rst), .q(Qb));

d\_ff C(.d(Dc), .clk(clk), .rst(rst), .q(Qc));

assign Da = ~bi;

assign Db = bi & Qa;

assign Dc = ~Qa & bi;

assign bo = Qb;

endmodule

1. **Testbench:**

module tb\_button\_press\_synchronizer;

reg clk, rst, bi;

wire bo;

button\_press\_synchronizer m1 (.clk(clk), .rst(rst), .bi(bi), .bo(bo));

initial begin

clk = 0;

rst = 0;

bi = 0;

end

always #5 clk = ~clk;

initial begin

$dumpvars;

repeat(2) @(posedge clk);

rst = 1;

repeat(3) @(posedge clk);

rst = 0;

@(posedge clk);

bi = 1;

repeat (1.5) @(posedge clk)

$display("Checking for Output Pulse");

if (bo != 1) begin

$display("Output is Incorrect Error: Line: 31");

$finish;

end

repeat (1.5) @(posedge clk);

$display("Checking if Output had one cycle");

if (bo != 0) begin

$display("Output is Incorrect Error: Line: 38");

$finish;

end

bi = 0;

repeat(5) @(posedge clk);

//Asserting Input again

bi = 1;

repeat(5) @(posedge clk);

bi = 0;

if (bo != 0) begin

@(posedge clk)

if (bo != 1) begin

$display("Output is Incorrect Line: 49");

$finish;

end

end

repeat(15) @(posedge clk);

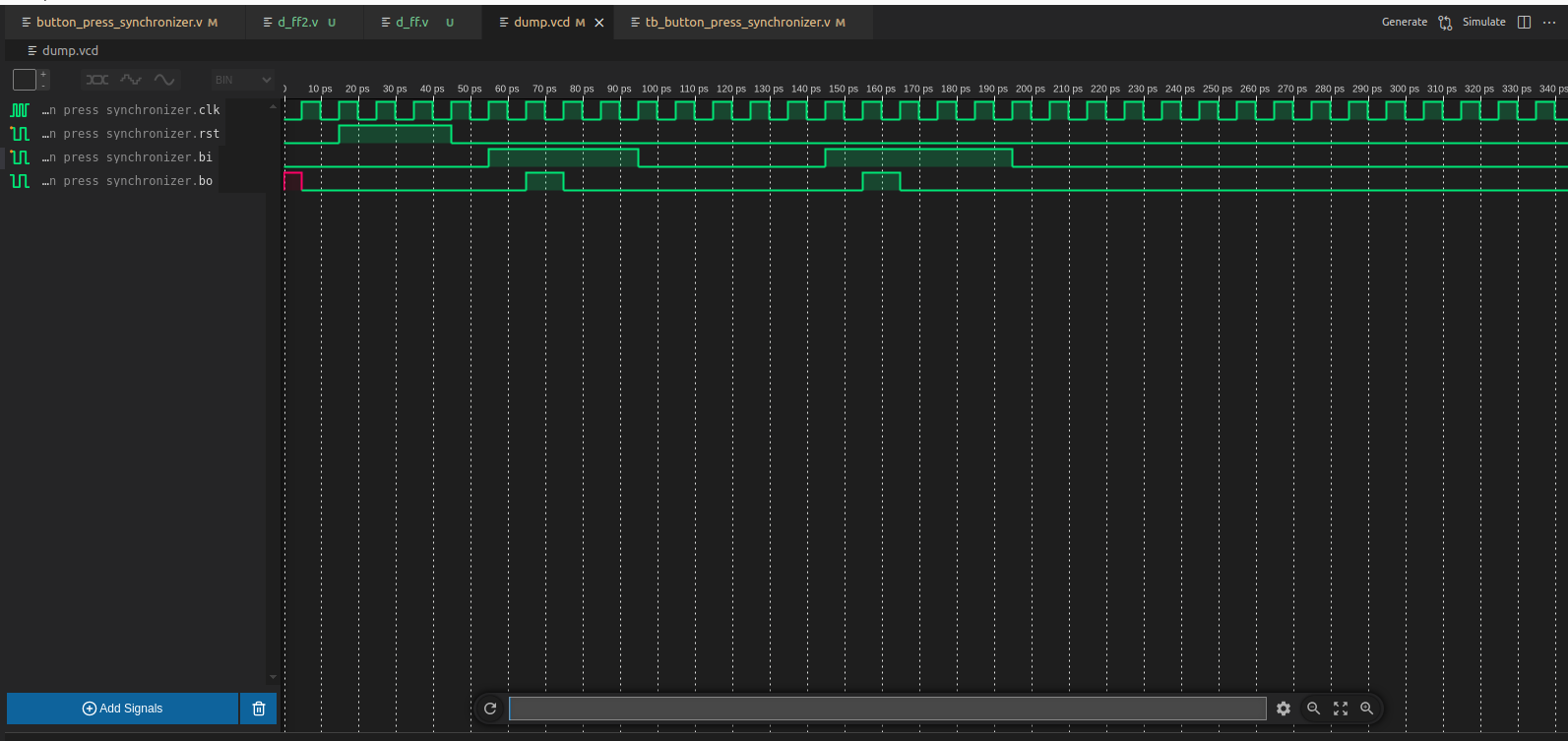
$display("All Test cases Passed!");

$finish;

end

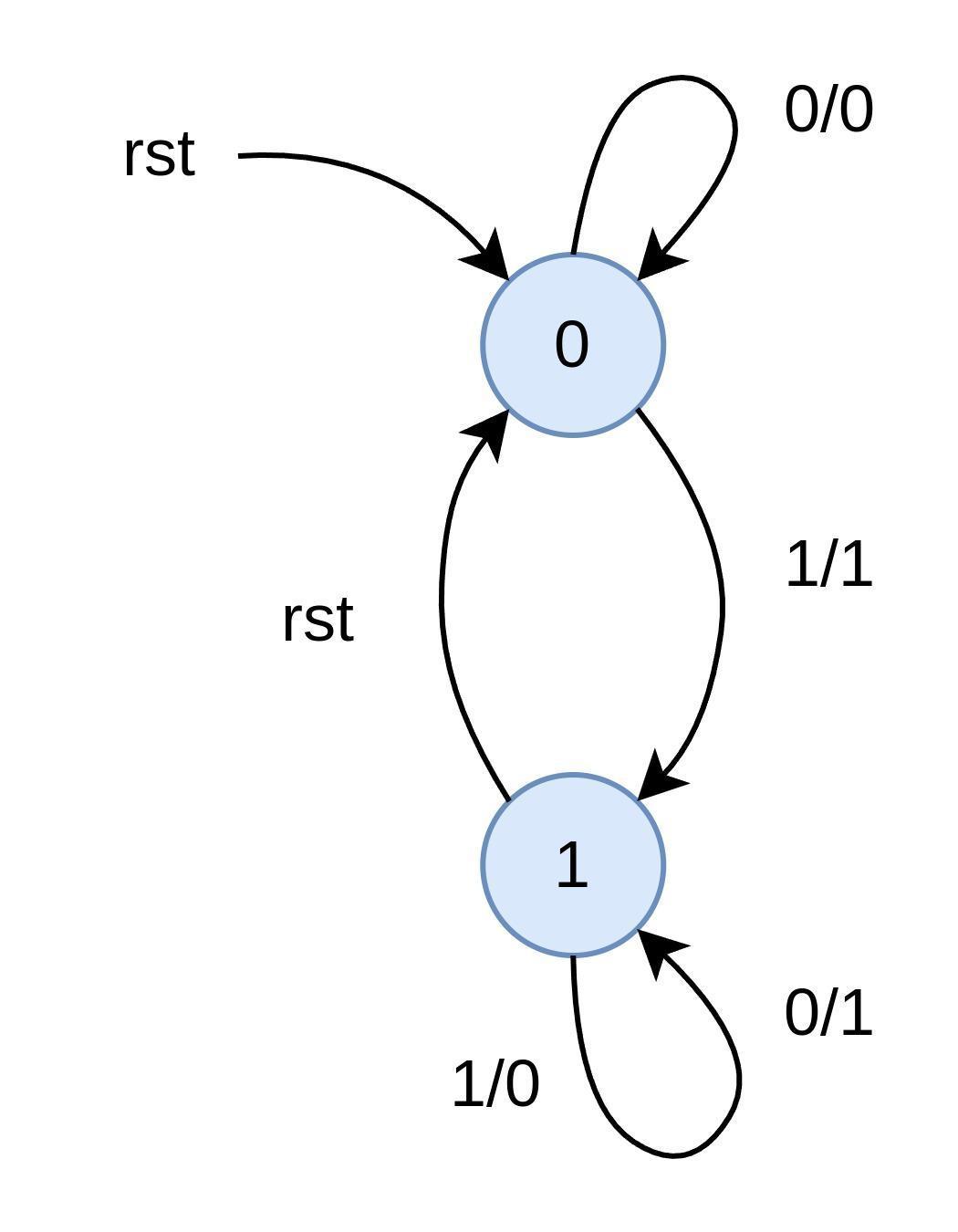
endmodule

1. **Output:**

****

* **Question 2: Serial Input 2’s Complementer:**

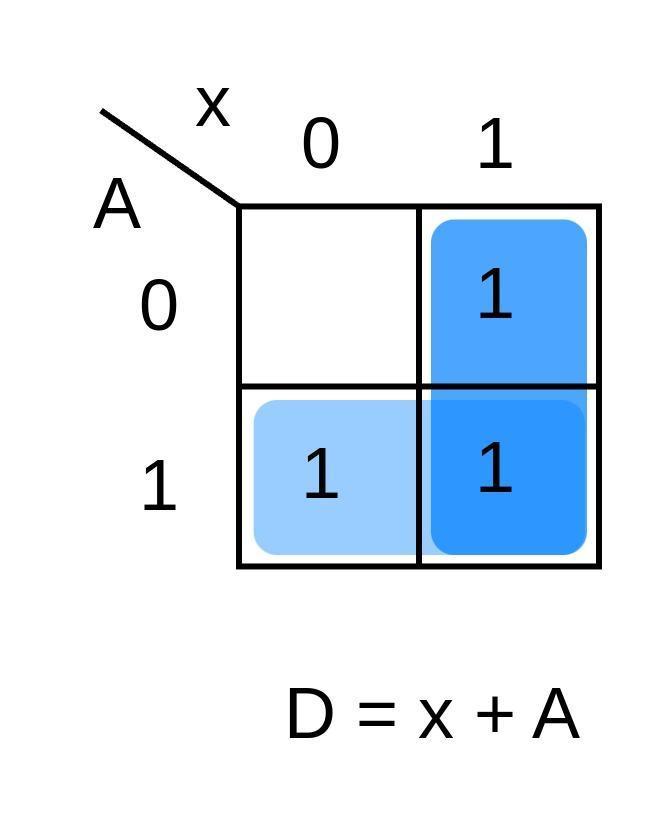
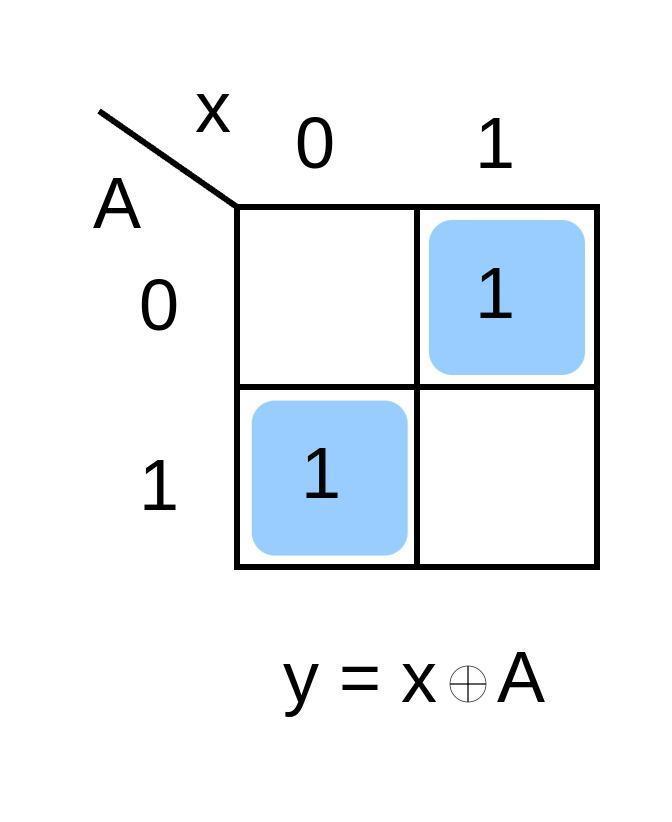
1. **State Diagram:**

****

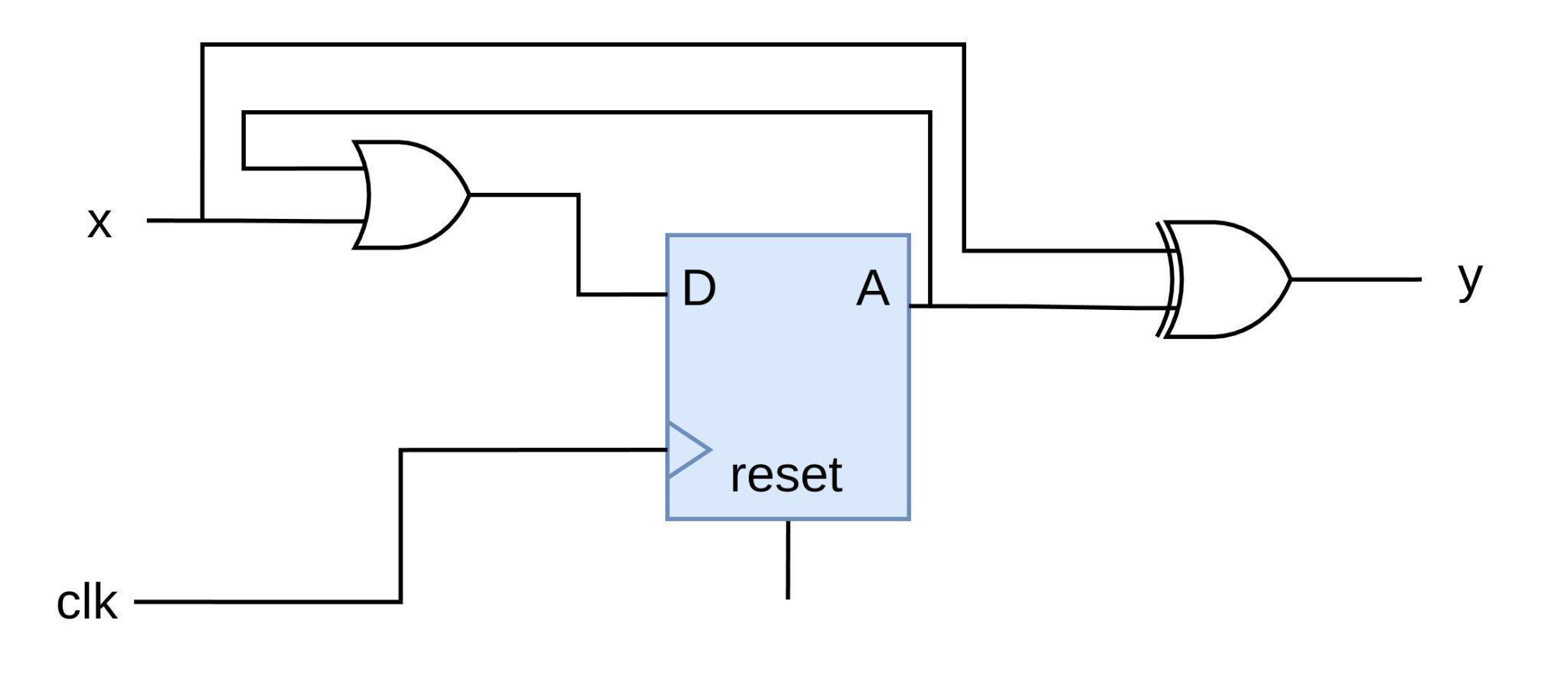
1. **State Table:**

| Present State | Input | Next State | Output | Flip-Flop Input |
| --- | --- | --- | --- | --- |
| Qt | x | Q(t+1) | y | D |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |

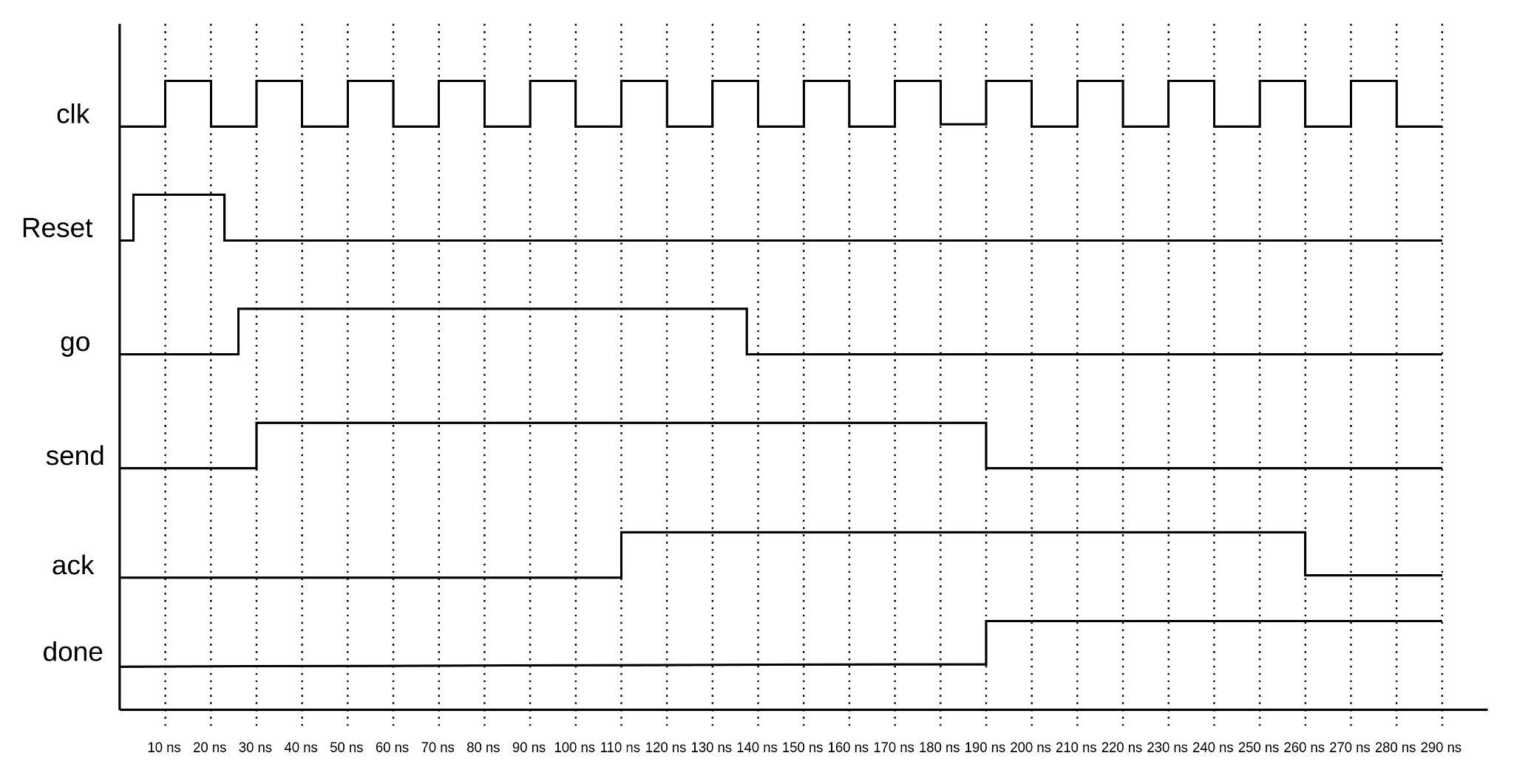
K-maps:

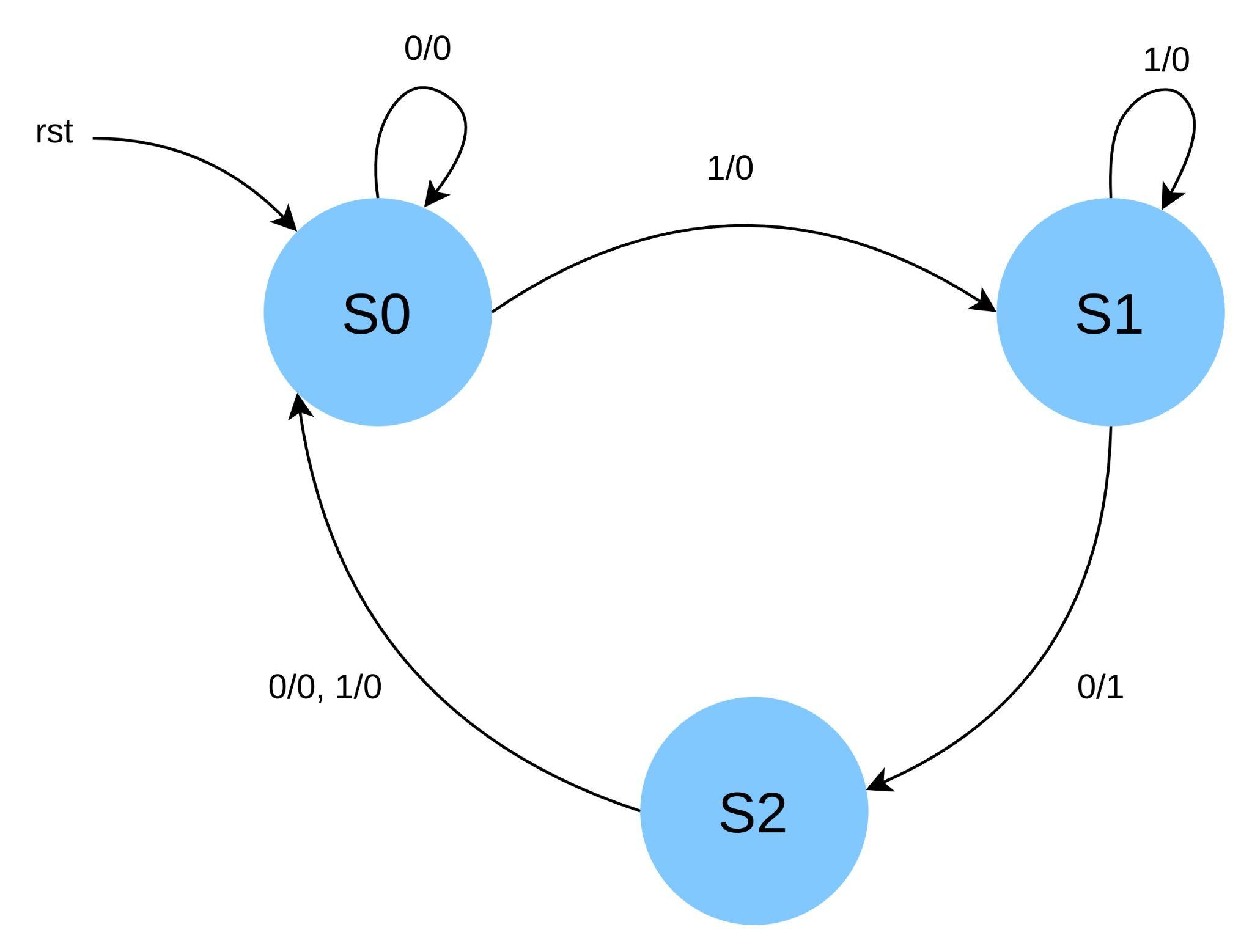
1. **Circuit Diagram:**

****

* **Question 3: Timing Diagram:**

****

* **Question 4: Pulse Detector:**
  1. **FSM Diagram:**

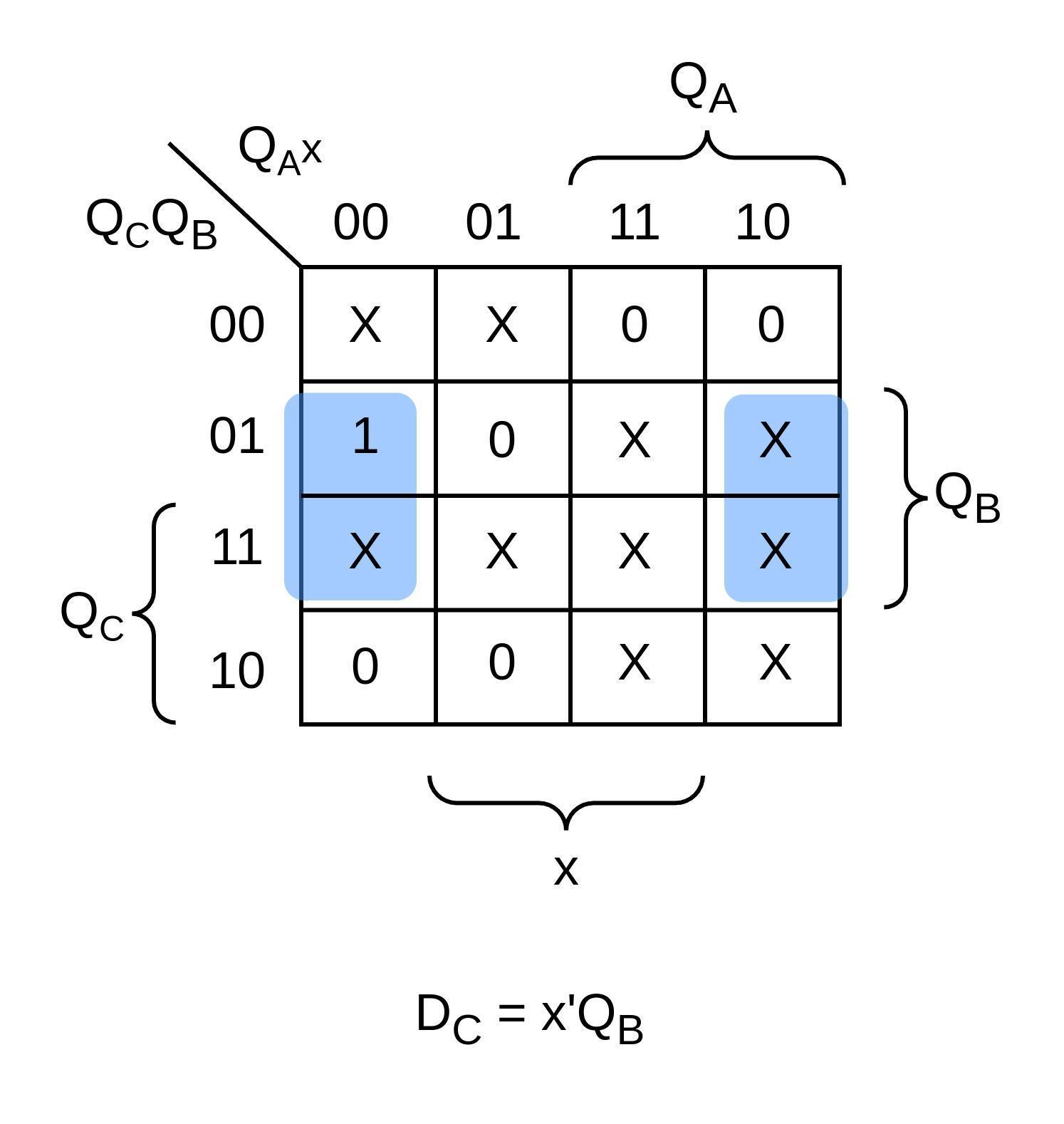
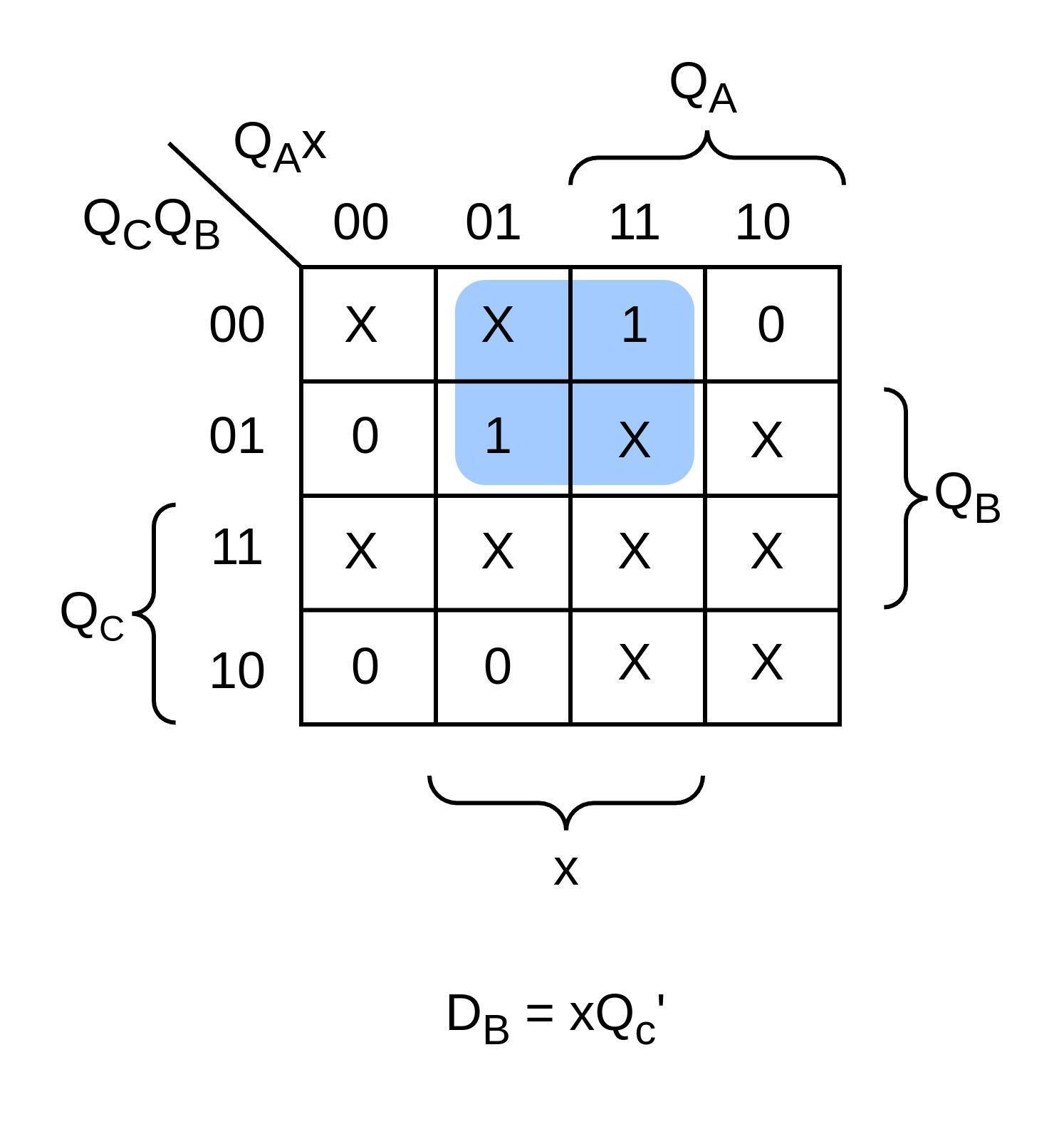
****

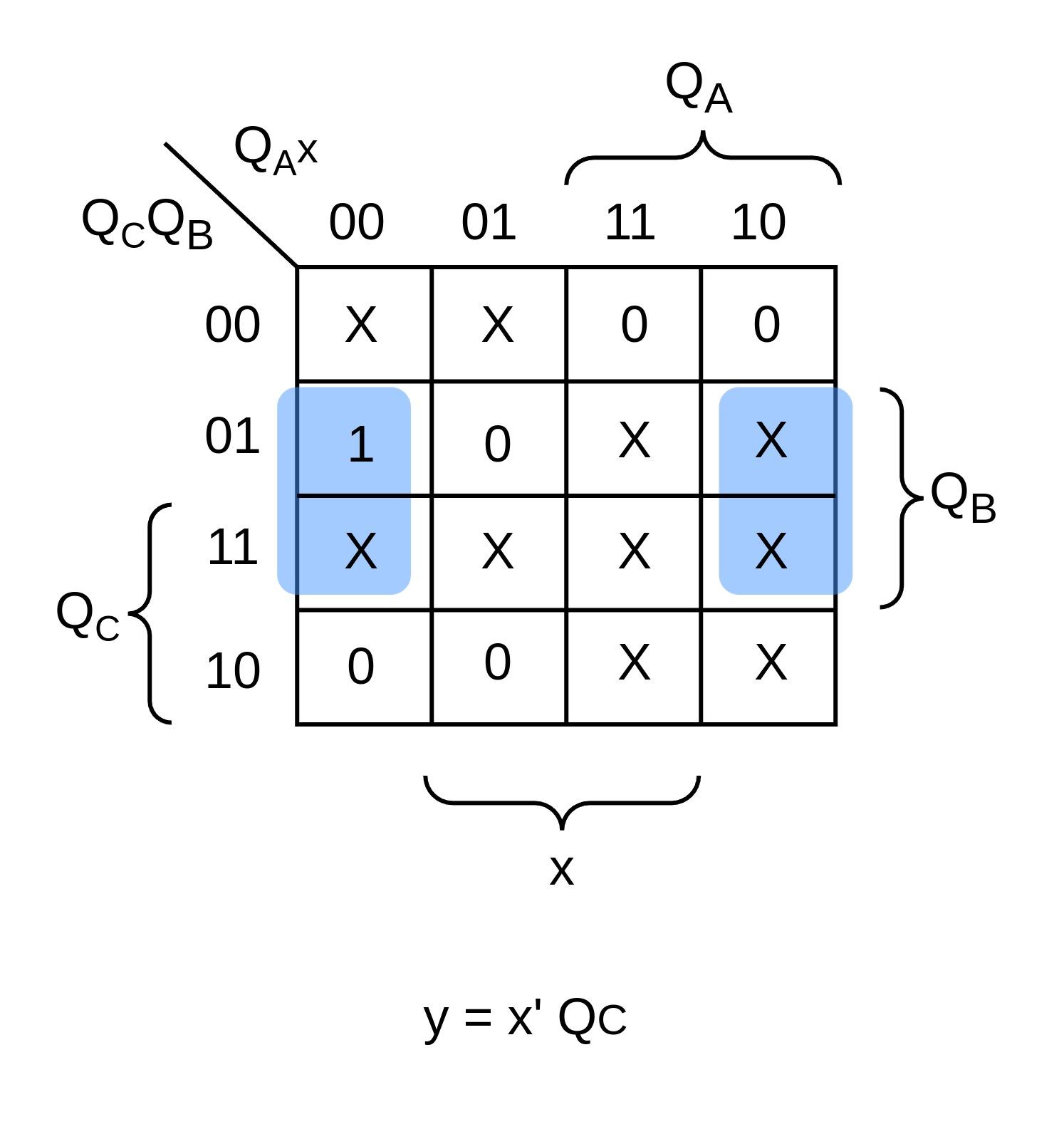
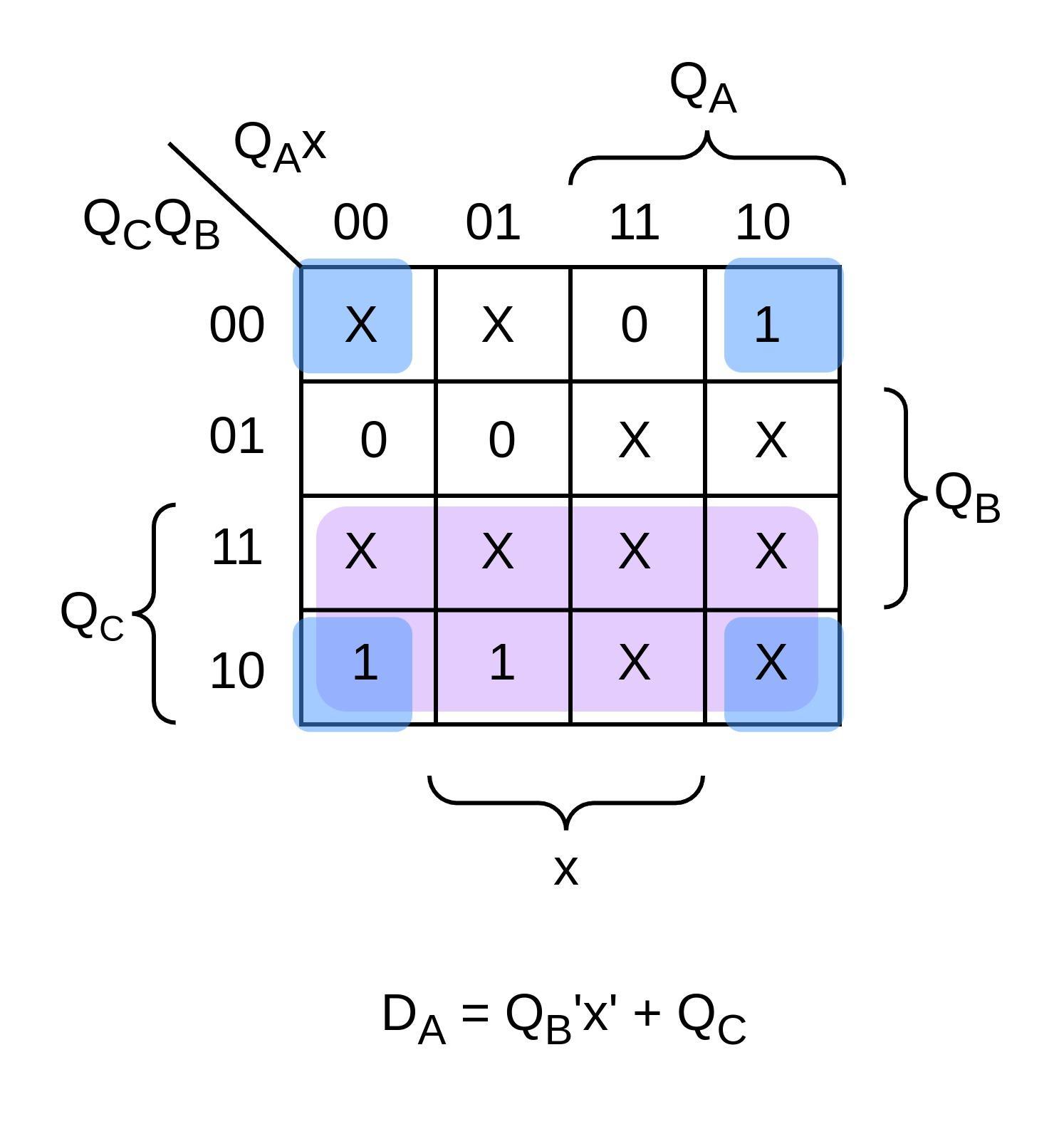
* 1. **Truth Table:**

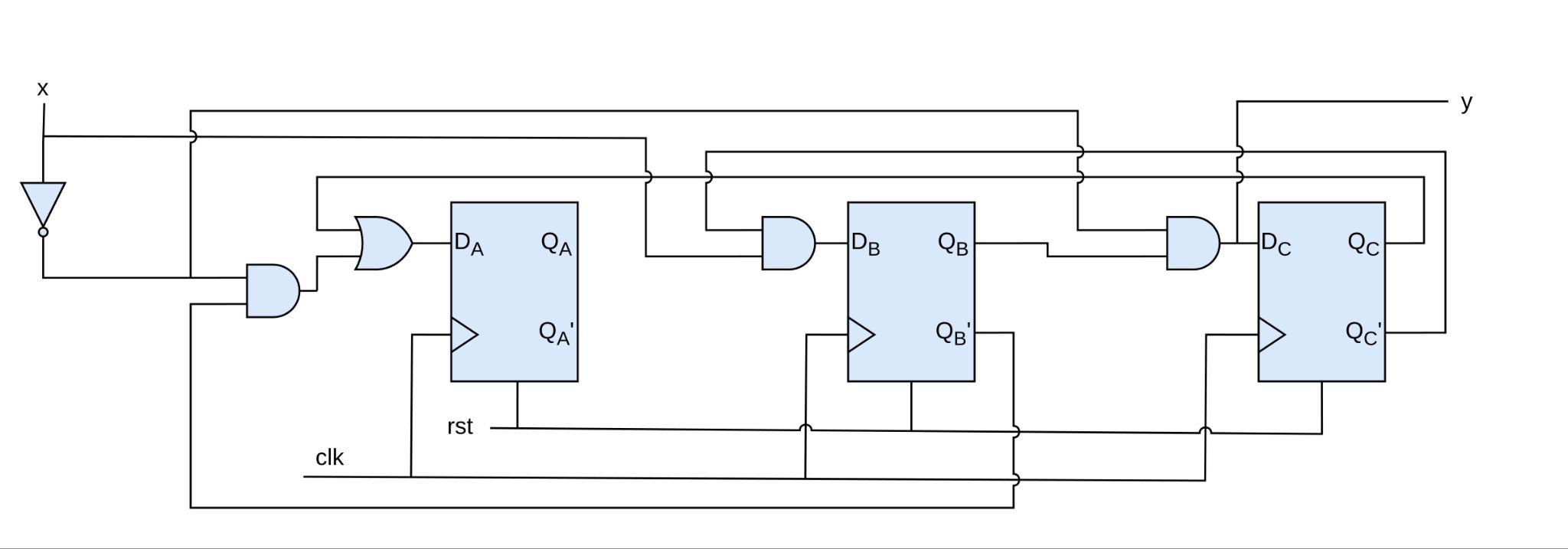
| **One-Hot Encoding** | | **S0 = 001** | **S1 = 010** | **S2 = 100** |
| --- | --- | --- | --- | --- |

| Present State | | | Input | Next State | | | Output | Inputs to Flip-Flops | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QC | QB | QA | x | QC+1 | QB+1 | QA+1 | y | DC | DB | DA |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

1. **Circuit Diagram:**

** **

****

****

1. **Verilog Code:**

module d\_ff (input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin

if (rst)

q <= 1'b0;

else

q <= d;

end

endmodule

module d\_ff2 (input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin

if (rst)

//LSB = 1 for 1-Hot Encoding

q <= 1'b1;

else

q <= d;

end

endmodule

module pulse\_detector (input clk, rst, x, output reg y);

wire Qa, Qb, Qc, Da, Db, Dc;

d\_ff2 A (.d(Da), .clk(clk), .rst(rst), .q(Qa));

d\_ff B (.d(Db), .clk(clk), .rst(rst), .q(Qb));

d\_ff C (.d(Dc), .clk(clk), .rst(rst), .q(Qc));

assign Da = ((~Qb & ~x) | Qc);

assign Db = ~Qc & x;

assign Dc = Qb & ~x;

assign y = Qb & ~x;

endmodule

1. **Testbench:**

module tb\_pulse\_detector;

reg clk, rst, x;

wire y;

pulse\_detector m0 (.clk(clk), .rst(rst), .x(x), .y(y));

always #5 clk = ~clk;

initial begin

clk = 0;

rst = 0;

end

initial begin

$dumpvars;

@(posedge clk);

rst = 1;

repeat(4) @(posedge clk);

rst = 0;

$display("Check the Output after reset");

if (y !=0 ) begin

$display("Incorrect Output after reset");

end

@(posedge clk);

x = 1;

repeat(3) @(posedge clk);

x = 0;

repeat(2) @(negedge clk);

if (y != 1) begin

@(negedge clk);

if (y != 0) begin

$display("Incorrect Output");

$finish;

end

end

@(posedge clk);

x = 1;

repeat(5) @(posedge clk);

x = 0;

repeat(2) @(negedge clk);

if (y != 1) begin

@(negedge clk);

if (y != 0) begin

$display("Incorrect Output");

$finish;

end

end

repeat(10) @(posedge clk);

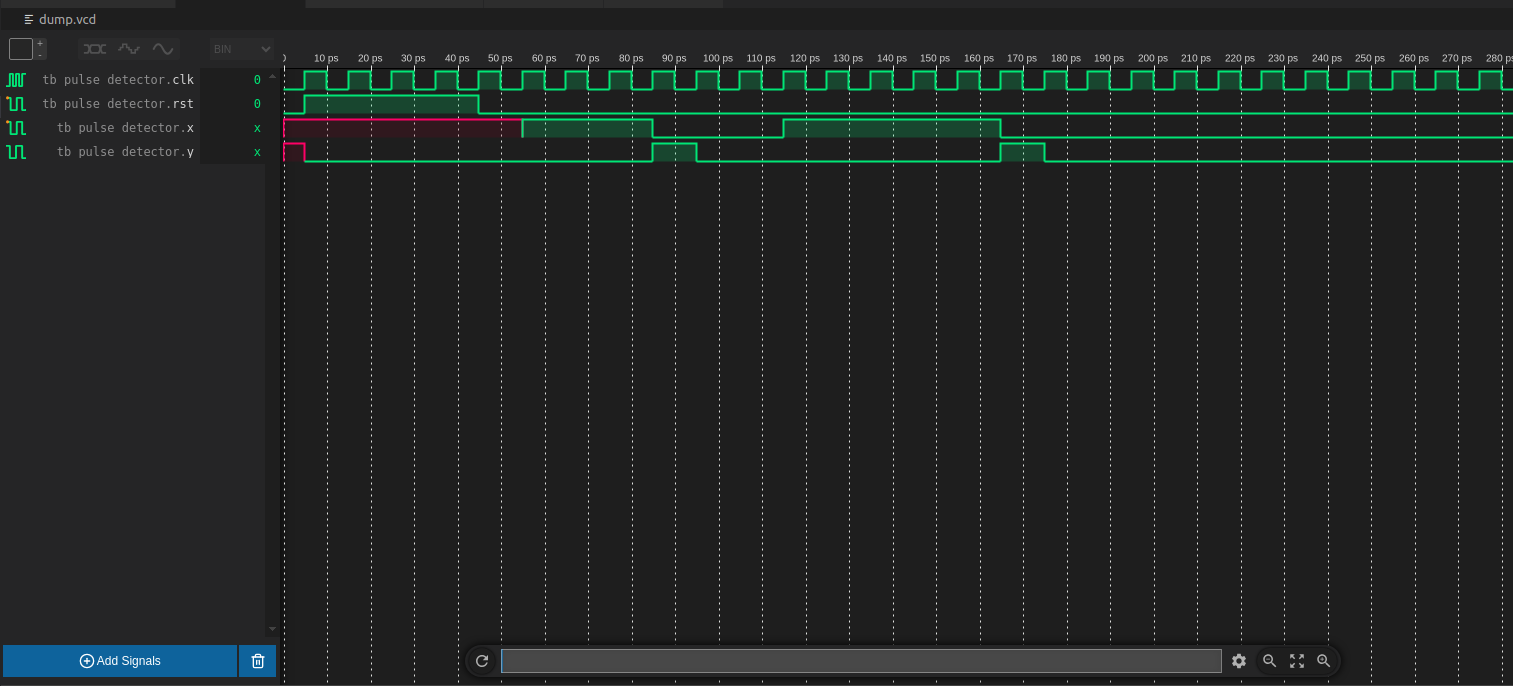
$display("All Test Cases Passed!");

$finish;

end

endmodule

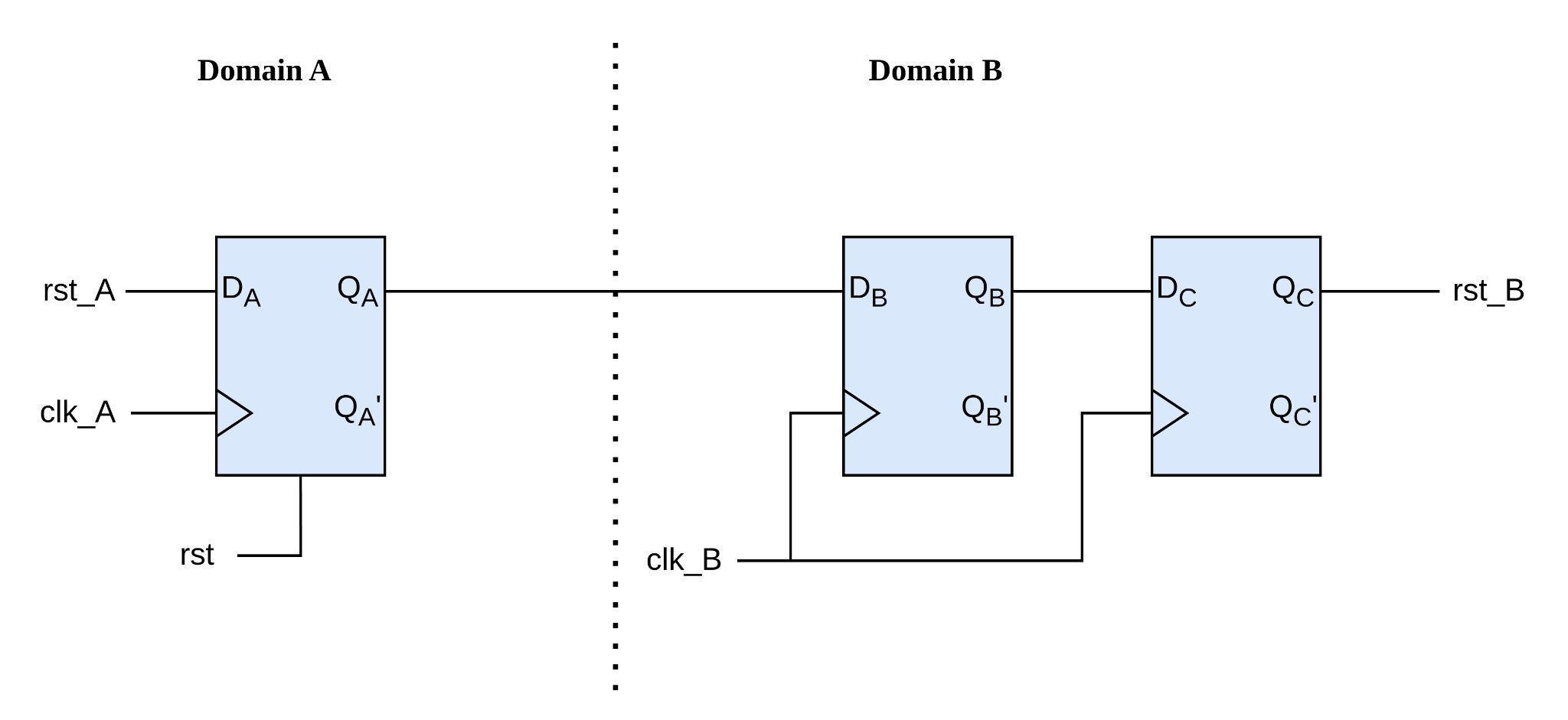
1. **Output:**

****

* **Question 5: Asynhcronus reset:**

Since we are dealing with 2 different clock domains. Hence, in order to avoid any glitches or metastability, I’ll be using a two-flop synchronizer ciruit to map the **reset\_A** signal synchronously with clock of domain B (**clk\_B**)

* 1. **Circuit Diagram:**

****

* 1. **Verilog Code:**

module d\_ff (input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin

if (rst)

q <= 1'b0;

else

q <= d;

end

endmodule

module synchronizer (input clk\_A, clk\_B, rst\_A, rst, output reg rst\_B);

wire w0, w1, w2;

d\_ff m0 (.clk(clk\_A), .rst(rst), .d(rst\_A), .q(w0));

//Two Flops Synchronizer

d\_ff m1 (.clk(clk\_B), .rst(1'b0), .d(w0), .q(w1));

d\_ff m2 (.clk(clk\_B), .rst(1'b0), .d(w1), .q(w2));

assign rst\_B = w2;

endmodule

* 1. **Testbench:**

module tb\_synchronizer;

reg clk\_A, clk\_B, rst, rst\_A;

wire rst\_B;

synchronizer dut (.clk\_A(clk\_A), .clk\_B(clk\_B), .rst(rst), .rst\_A(rst\_A), .rst\_B(rst\_B));

always #5 clk\_A = ~clk\_A;

always #20 clk\_B = ~clk\_B;

initial begin

$dumpvars;

clk\_A = 0;

clk\_B = 0;

rst = 0;

rst\_A = 0;

#20;

rst = 1;

#10;

rst\_A = 1;

#40;

rst = 0;

#40;

rst\_A = 0;

#80;

rst\_A = 1;

#20;

rst\_A = 0;

#30;

rst\_A = 1;

#30;

rst\_A = 0;

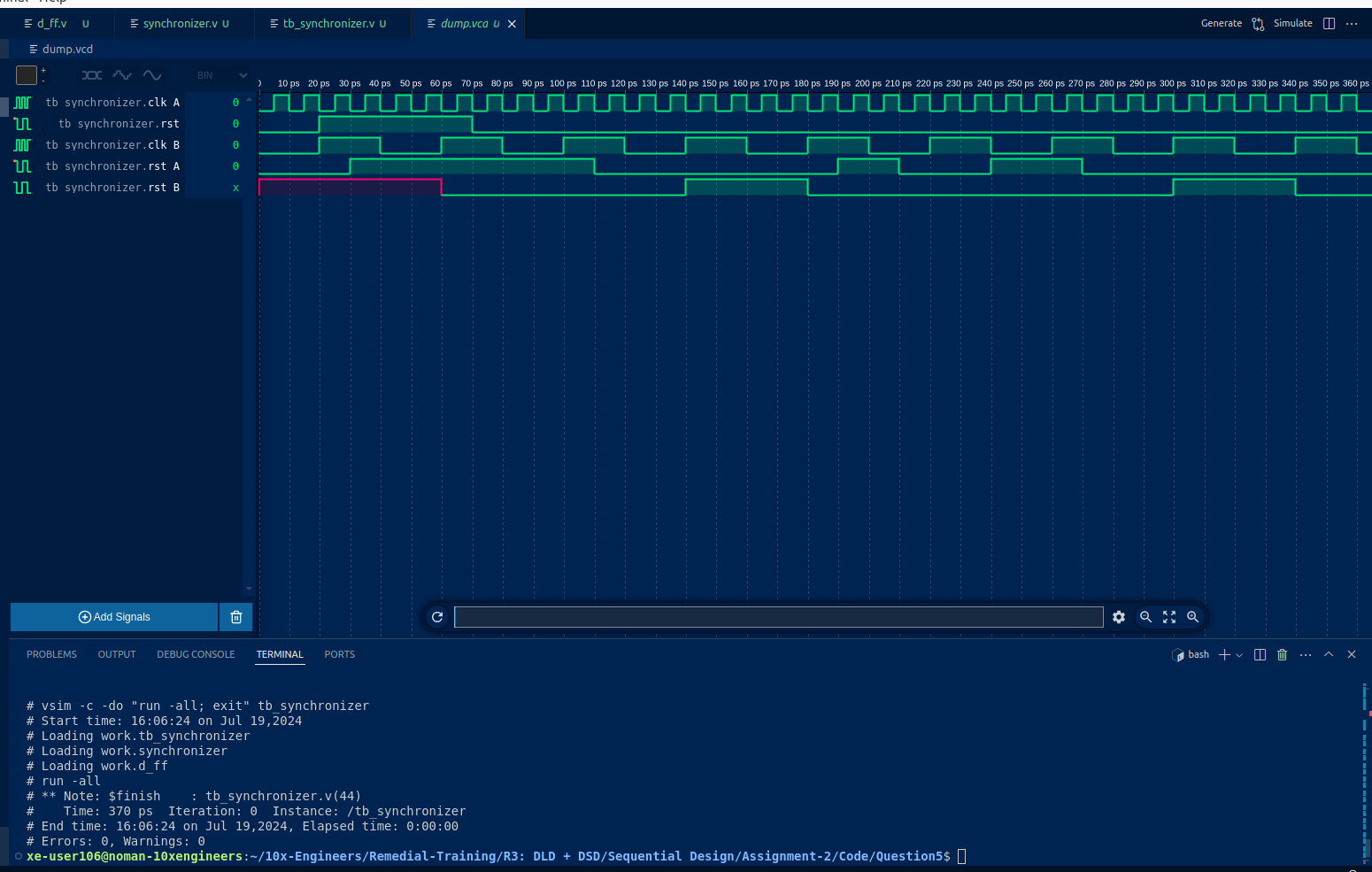
#100;

$finish;

end

endmodule

* 1. **Output:**

****