**Lab6 Pre-Lab Report**

Ziyao Chen

1002453409

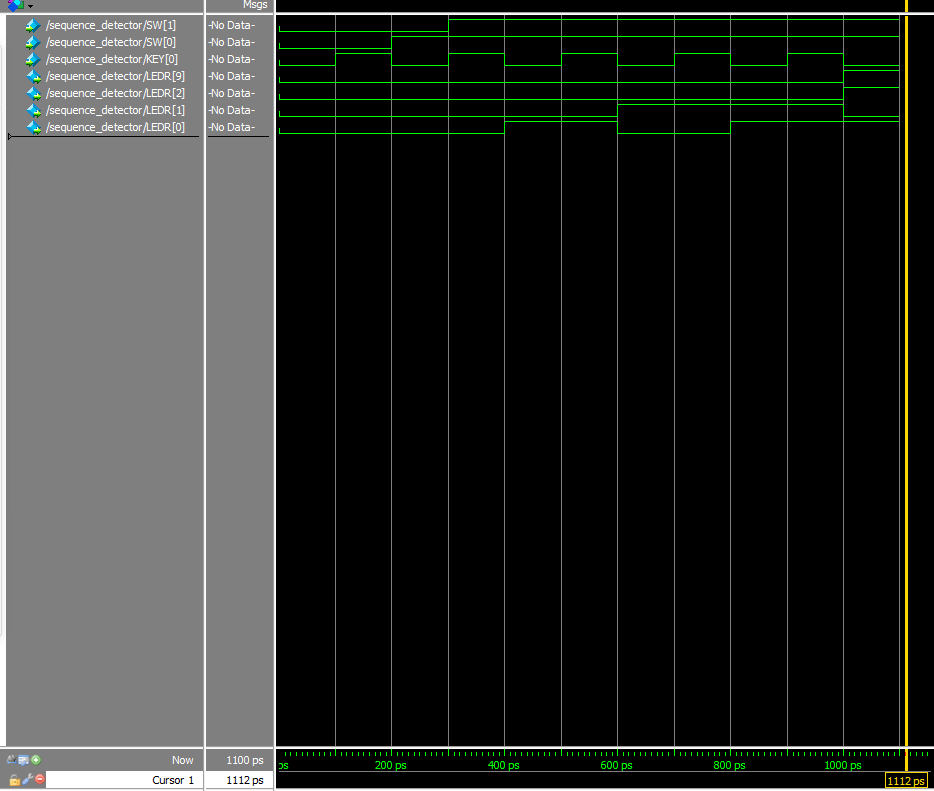
**Part1**

2. Synchronous reset, it is active low. We first want to set reset to 0, then start simulation.

3.

|  |  |  |
| --- | --- | --- |
| Current State | Input | Next State |
| A | 0 | A |
| A | 1 | B |
| B | 0 | A |
| B | 1 | C |
| C | 0 | E |
| C | 1 | D |
| D | 0 | E |
| D | 1 | F |
| E | 0 | A |
| E | 1 | G |
| F | 0 | E |
| F | 1 | F |
| G | 0 | A |
| G | 1 | C |

4.

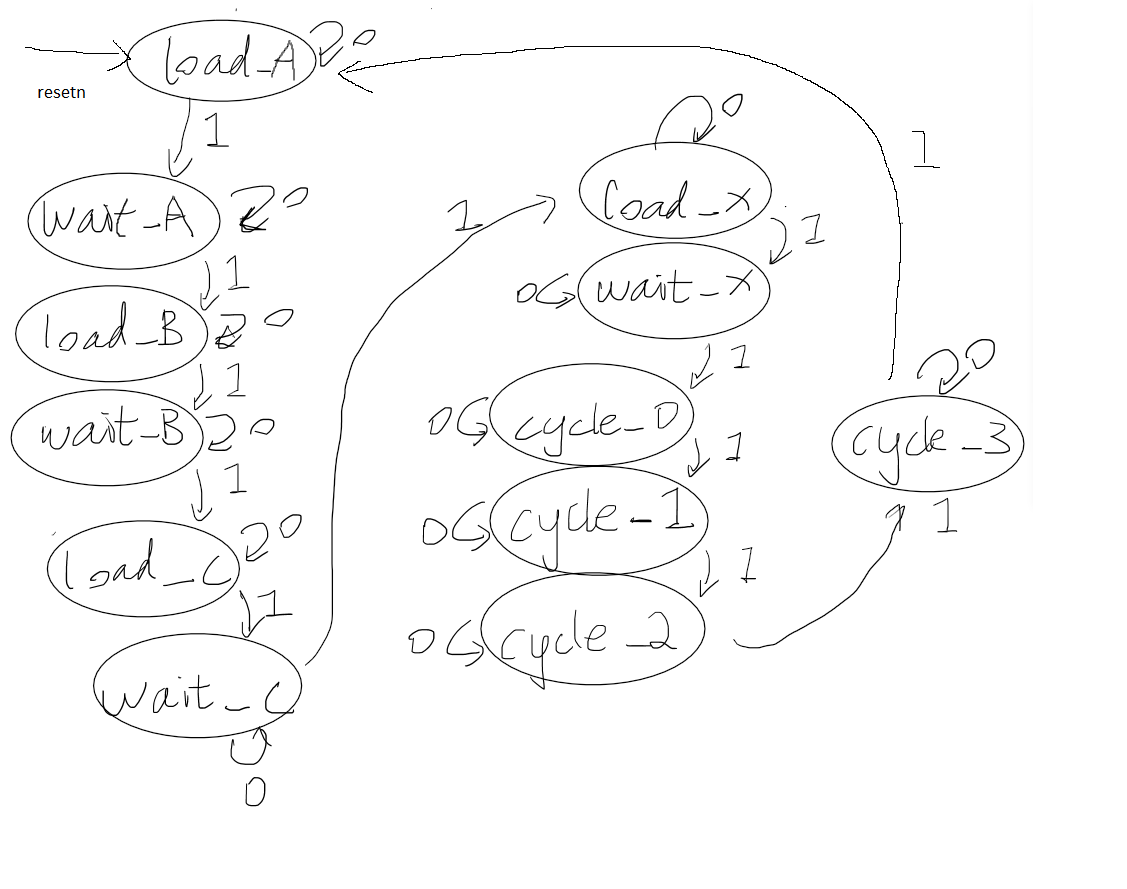


**Part2**

2.

|  |  |  |
| --- | --- | --- |
| Description | Control signals | Register states |
| Load A to register A | data\_in = A  ld\_alu\_out = 0  ld\_a = 1  ld\_b = 0  ld\_c = 0  ld\_x = 0  ld\_r = 0  alu\_select\_a = 00  alu\_select\_b = 00  alu\_op = 0 | Register A = A  Register B = 00000000  Register C = 00000000  Register x = 00000000 |
| Load B to register B | data\_in = B  ld\_alu\_out = 0  ld\_a = 0  ld\_b = 1  ld\_c = 0  ld\_x = 0  ld\_r = 0  alu\_select\_a = 00  alu\_select\_b = 00  alu\_op = 0 | Register A = A  Register B = B  Register C = 00000000  Register x = 00000000 |
| Load C to register C | data\_in = C  ld\_alu\_out = 0  ld\_a = 0  ld\_b = 0  ld\_c = 1  ld\_x = 0  ld\_r = 0  alu\_select\_a = 00  alu\_select\_b = 00  alu\_op = 0 | Register A = A  Register B = B  Register C = C  Register x = 00000000 |
| Load x to register x | data\_in = x  ld\_alu\_out = 0  ld\_a = 0  ld\_b = 0  ld\_c = 0  ld\_x = 1  ld\_r = 0  alu\_select\_a = 00  alu\_select\_b = 00  alu\_op = 0 | Register A = A  Register B = B  Register C = C  Register x = x |
| Calculate A \* x and load to register A | data\_in = N/A  ld\_alu\_out = 1  ld\_a = 0  ld\_b = 0  ld\_c = 0  ld\_x = 0  ld\_r = 0  alu\_select\_a = 00  alu\_select\_b = 11  alu\_op = 1 | Register A = Ax  Register B = B  Register C = C  Register x = x |
| Calculate Ax \* x and load to register A | data\_in = N/A  ld\_alu\_out = 1  ld\_a = 0  ld\_b = 0  ld\_c = 0  ld\_x = 0  ld\_r = 0  alu\_select\_a = 00  alu\_select\_b = 11  alu\_op = 1 | Register A = Ax2  Register B = B  Register C = C  Register x = x |
| Calculate B \* x and load to register B | data\_in = N/A  ld\_alu\_out = 1  ld\_a = 0  ld\_b = 0  ld\_c = 0  ld\_x = 0  ld\_r = 0  alu\_select\_a = 01  alu\_select\_b = 11  alu\_op = 1 | Register A = Ax2  Register B = Bx  Register C = C  Register x = x |
| Calculate Ax2 + Bx and load to register A | data\_in = N/A  ld\_alu\_out = 1  ld\_a = 0  ld\_b = 0  ld\_c = 0  ld\_x = 0  ld\_r = 1  alu\_select\_a = 00  alu\_select\_b = 01  alu\_op = 0 | Register A = Ax2 + Bx  Register B = Bx  Register C = C  Register x = x |
| Calculate Ax2 + Bx + C and load to register A | data\_in = N/A  ld\_alu\_out = 1  ld\_a = 0  ld\_b = 0  ld\_c = 0  ld\_x = 0  ld\_r = 1  alu\_select\_a = 00  alu\_select\_b = 10  alu\_op = 0 | Register A = Ax2 + Bx + C  Register B = Bx  Register C = C  Register x = x |

3.



4. module control(

input clk,

input resetn,

input go,

output reg ld\_a, ld\_b, ld\_c, ld\_x, ld\_r,

output reg ld\_alu\_out,

output reg [1:0] alu\_select\_a, alu\_select\_b,

output reg alu\_op

);

reg [3:0] current\_state, next\_state;

reg status;

localparam S\_LOAD\_A = 4'd0,

S\_LOAD\_A\_WAIT = 4'd1,

S\_LOAD\_B = 4'd2,

S\_LOAD\_B\_WAIT = 4'd3,

S\_LOAD\_C = 4'd4,

S\_LOAD\_C\_WAIT = 4'd5,

S\_LOAD\_X = 4'd6,

S\_LOAD\_X\_WAIT = 4'd7,

S\_CYCLE\_0 = 4'd8, // load Ax to register A

S\_CYCLE\_1 = 4'd9, // load Ax^2 to register A

S\_CYCLE\_2 = 4'd10, // load Bx to register B

S\_CYCLE\_3 = 4'd11; // load Ax^2 + Bx to register A

S\_CYCLE\_4 = 4'd12; // load Ax^2 + Bx + C to register R

// Next state logic aka our state table

always@(\*)

begin: state\_table

case (current\_state)

S\_LOAD\_A: next\_state = go ? S\_LOAD\_A\_WAIT : S\_LOAD\_A; // Loop in current state until value is input

S\_LOAD\_A\_WAIT: next\_state = go ? S\_LOAD\_A\_WAIT: S\_LOAD\_B;

S\_LOAD\_B: next\_state = go ? S\_LOAD\_C : S\_LOAD\_B; // Loop in current state until value is input

S\_LOAD\_B\_WAIT: next\_state = go ? S\_LOAD\_B\_WAIT: S\_LOAD\_C;

S\_LOAD\_C: next\_state = go ? S\_LOAD\_X : S\_LOAD\_C; // Loop in current state until value is input

S\_LOAD\_C\_WAIT: next\_state = go ? S\_LOAD\_C\_WAIT: S\_LOAD\_X;

S\_LOAD\_X: next\_state = go ? S\_CYCLE\_0 : S\_LOAD\_X; // Loop in current state until value is input

S\_LOAD\_X\_WAIT: next\_state = go ? S\_LOAD\_X\_WAIT: S\_CYCLE\_0;

S\_CYCLE\_0: next\_state = S\_CYCLE\_1;

S\_CYCLE\_1: next\_state = S\_CYCLE\_2;

S\_CYCLE\_2: next\_state = S\_CYCLE\_3;

S\_CYCLE\_3: next\_state = S\_CYCLE\_4;

default: next\_state = S\_LOAD\_A;

endcase

end // state\_table

// Output logic aka all of our datapath control signals

always @(\*)

begin: enable\_signals

// By default make all our signals 0

ld\_alu\_out = 1'b0;

ld\_a = 1'b0;

ld\_b = 1'b0;

ld\_c = 1'b0;

ld\_x = 1'b0;

ld\_r = 1'b0;

alu\_select\_a = 2'b00;

alu\_select\_b = 2'b00;

alu\_op = 1'b0;

case (current\_state)

S\_LOAD\_A: begin

ld\_a = 1'b1;

end

S\_LOAD\_B: begin

ld\_b = 1'b1;

end

S\_LOAD\_C: begin

ld\_c = 1'b1;

end

S\_LOAD\_X: begin

ld\_x = 1'b1;

end

S\_CYCLE\_0: begin // Do A <- A \* x

ld\_alu\_out = 1'b1; ld\_a = 1'b1; // store result back into A

alu\_select\_a = 2'b00; // Select register A

alu\_select\_b = 2'b11; // Select register x

alu\_op = 1'b1; // Do multiply operation

end

S\_CYCLE\_1: begin // Do A <- Ax \* x

ld\_alu\_out = 1'b1; ld\_a = 1'b1; // store result back into A

alu\_select\_a = 2'b00; // Select register A

alu\_select\_b = 2'b11; // Select register x

alu\_op = 1'b1; // Do multiply operation

end

S\_CYCLE\_2: begin // Do B <- B \* x

ld\_alu\_out = 1'b1; ld\_b = 1'b1; // store result back into A

alu\_select\_a = 2'b01; // Select register B

alu\_select\_b = 2'b11; // Select register x

alu\_op = 1'b1; // Do multiply operation

end

S\_CYCLE\_3: begin // Do A <- Ax^2 + Bx

ld\_alu\_out = 1'b1; ld\_a = 1'b1; // store result back into A

alu\_select\_a = 2'b00; // Select register A

alu\_select\_b = 2'b01; // Select register B

alu\_op = 1'b0; // Do multiply operation

end

S\_CYCLE\_4: begin

ld\_r = 1'b1; // store result in result register

alu\_select\_a = 2'b00; // Select register A

alu\_select\_b = 2'b10; // Select register C

alu\_op = 1'b0; // Do Add operation

end

// default: // don't need default since we already made sure all of our outputs were assigned a value at the start of the always block

endcase

end // enable\_signals

// current\_state registers

always@(posedge clk)

begin: state\_FFs

if(!resetn)

current\_state <= S\_LOAD\_A;

else

current\_state <= next\_state;

end // state\_FFS

endmodule

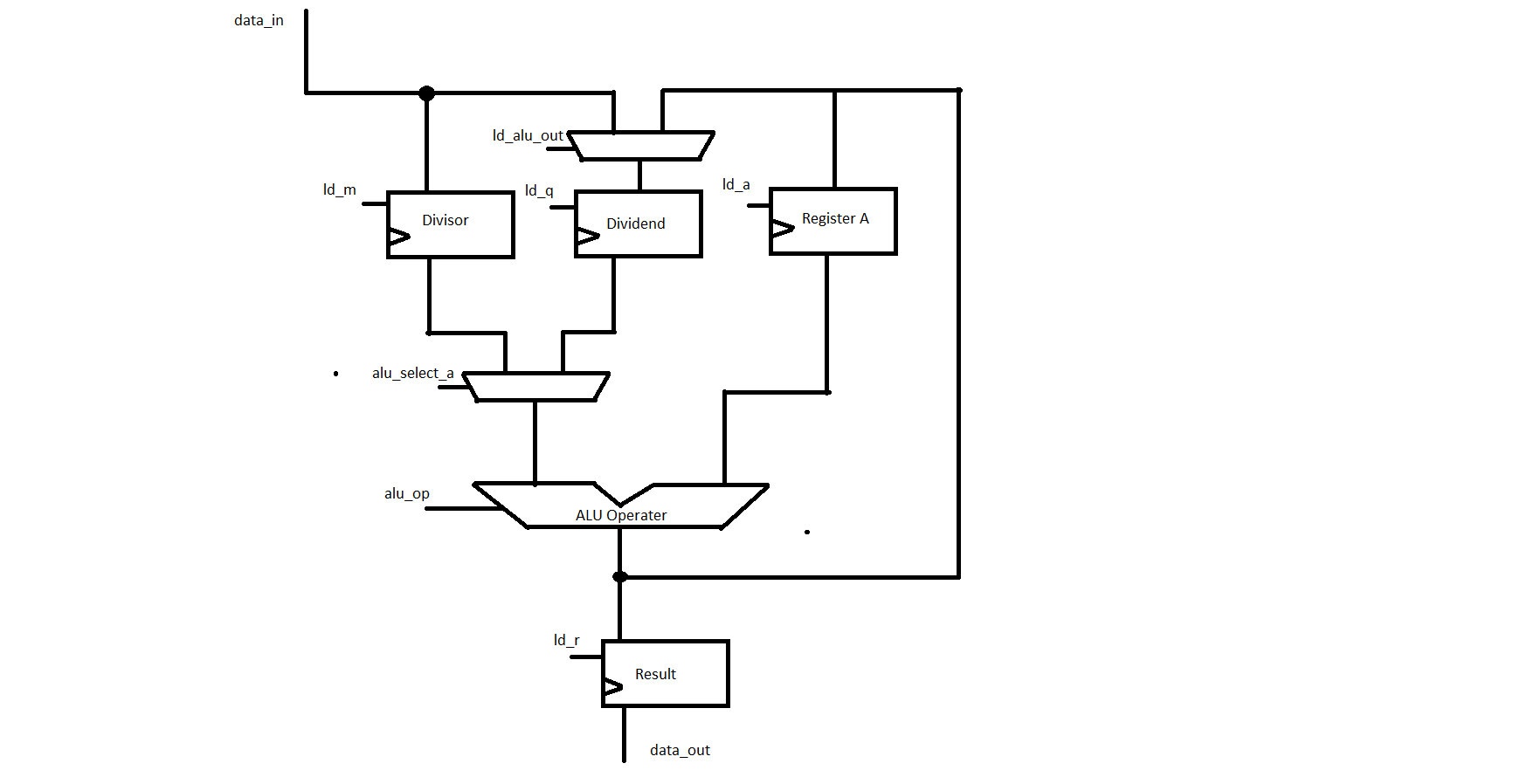
5.

6.

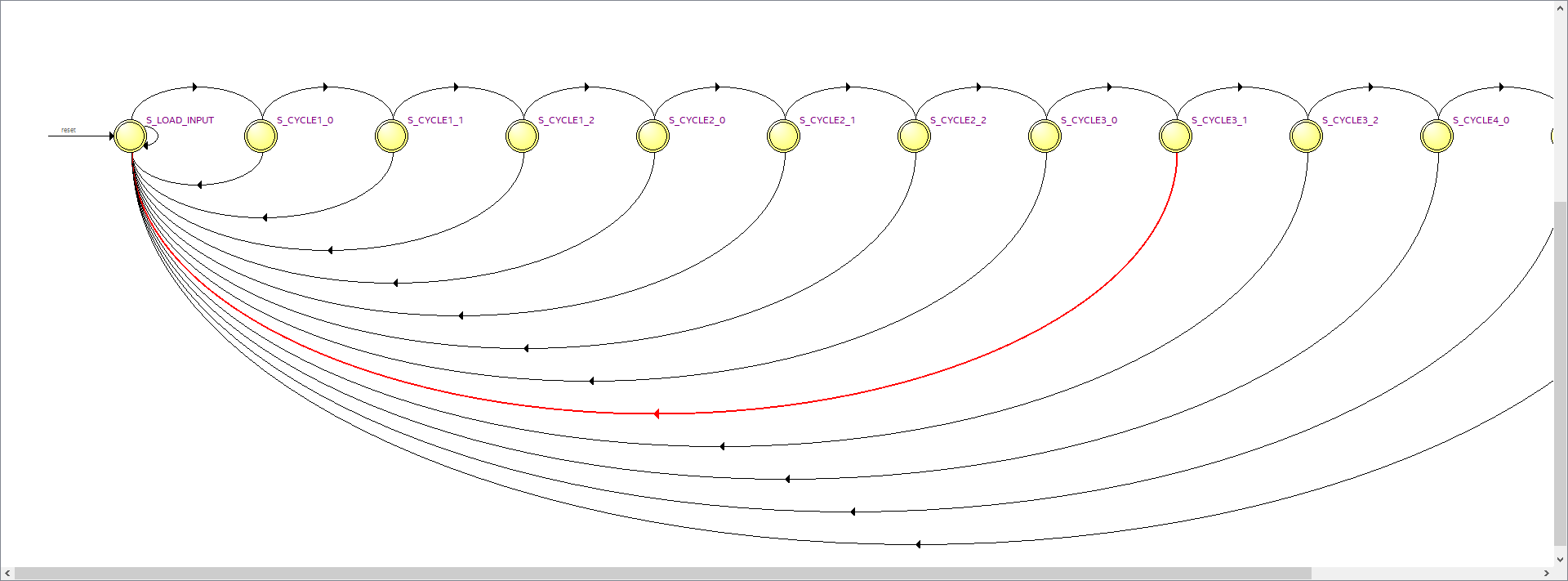


**Part3**

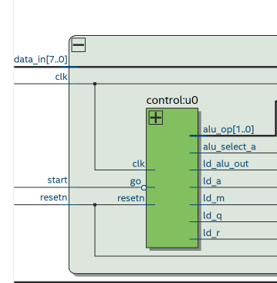




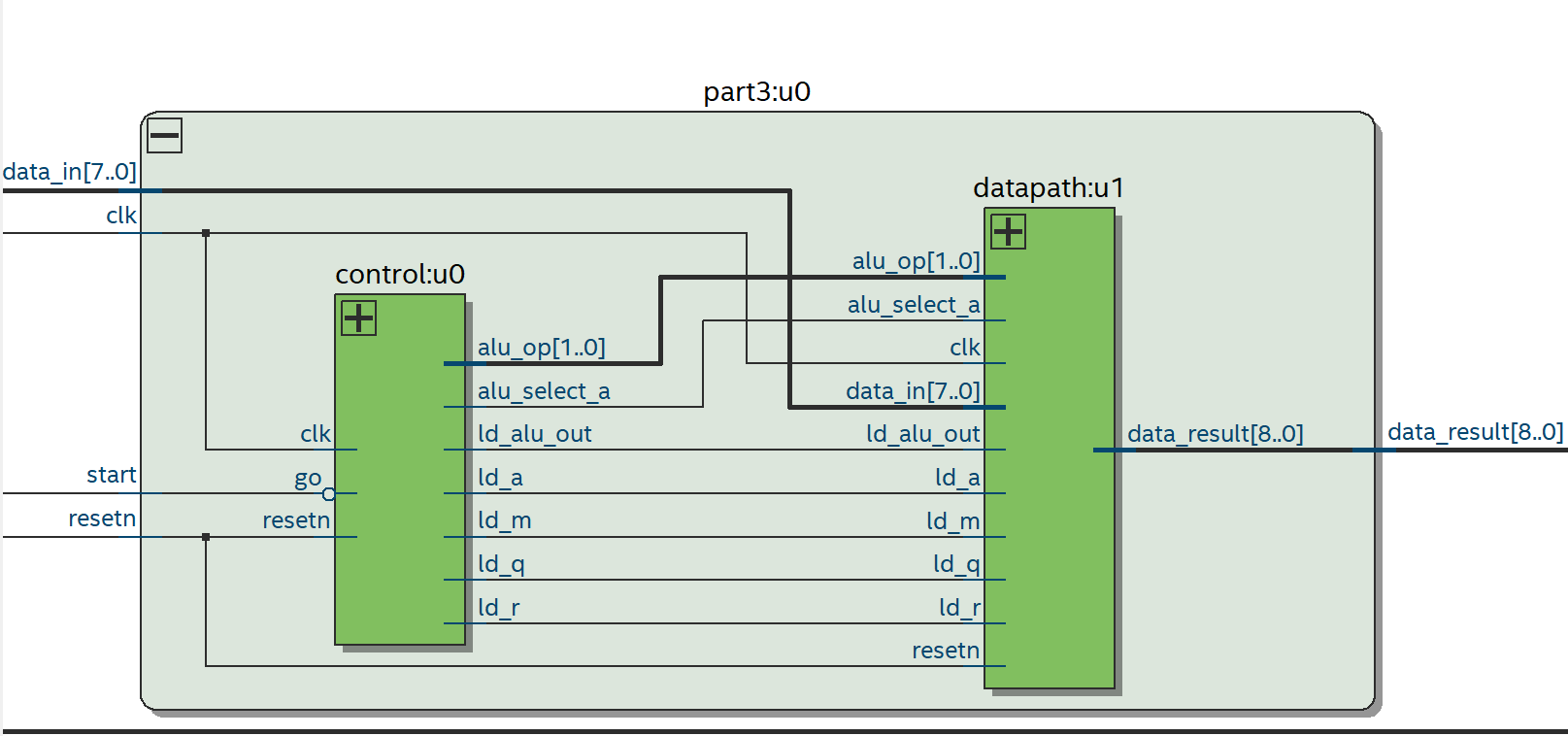














Show TA on computer



