FPGA Application Week 9 Homework

CYEE 10828241 Chen Da-Chuan

May 1, 2023

Contents

List of Figures

List of Tables

1 Exercise 9-1

1.1 Objective

Code the most basic data register structure. It can toggle between write and not write, and read data from specified address. This code has decoder, data register, address register.

1.2 Operation

Table 1: Exercise 9-1 Operation detail

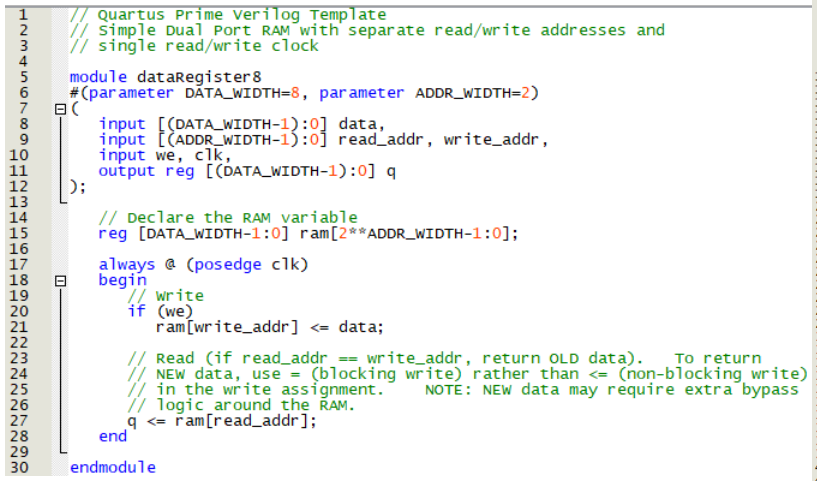
type	var	operation
input	data	data to be written into ram, 1 8bits data each time
input	read_addr	read address
input	write_addr	write address
input	we	write enable, which writes data into register
input	clk	clock
output	q	output data, return data from register

1.3 Code

1. Figure 1 Line6-6: Declare parameter for data and address width.
2. Figure 1 Line7-12: Declare input and output ports.
3. Figure 1 Line15-15: Declare register for data.
4. Figure 1 Line20-21: If clock is triggered, and if write is enabled, store input data into register.
5. Figure 1 Line27-27: If clock is triggered, send the data in specified address to output.
6. Figure 2 Line2-6: Declare parameter to use coded data register.
7. Figure 2 Line8-14: Include coded data register.
8. Figure 2 Line16-22: Set the initial value of each input port.
9. Figure 2 Line24-27: Switch clock value every 50ns.
10. Figure 2 Line29-32: Add the data by 1 every 100ns.

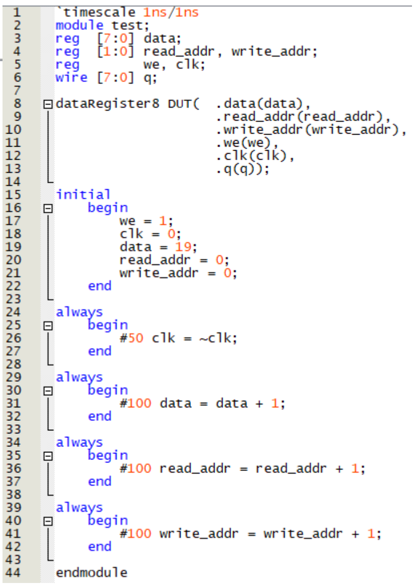
11. Figure 2 Line34-37: Add the read address by 1 every 100ns.

12. Figure 2 Line39-42: Add the write address by 1 every 100ns.



```verilog
// Quartus Prime Verilog Template
// Simple Dual Port RAM with separate read/write addresses and
// single read/write clock

module dataRegister8
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=2)
(
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] read_addr, write_addr,
    input we, clk,
    output reg [(DATA_WIDTH-1):0] q
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    always @ (posedge clk)
    begin
        // Write
        if (we)
            ram[write_addr] <= data;

        // Read (if read_addr == write_addr, return OLD data).    To return
        // NEW data, use = (blocking write) rather than <= (non-blocking write)
        // in the write assignment.    NOTE: NEW data may require extra bypass
        // logic around the RAM.
        q <= ram[read_addr];
    end

endmodule
```

Figure 1: Main file



```verilog
1  `timescale 1ns/1ns
2  module test;
3  reg  [7:0] data;
4  reg  [1:0] read_addr, write_addr;
5  reg        we, clk;
6  wire [7:0] q;
7
8  dataRegister8 DUT(  .data(data),
9                      .read_addr(read_addr),
10                     .write_addr(write_addr),
11                     .we(we),
12                     .clk(clk),
13                     .q(q));
14
15 initial
16     begin
17         we = 1;
18         clk = 0;
19         data = 19;
20         read_addr = 0;
21         write_addr = 0;
22     end
23
24 always
25     begin
26         #50 clk = ~clk;
27     end
28
29 always
30     begin
31         #100 data = data + 1;
32     end
33
34 always
35     begin
36         #100 read_addr = read_addr + 1;
37     end
38
39 always
40     begin
41         #100 write_addr = write_addr + 1;
42     end
43
44 endmodule
```

Figure 2: Test file

1.4 Execution Result

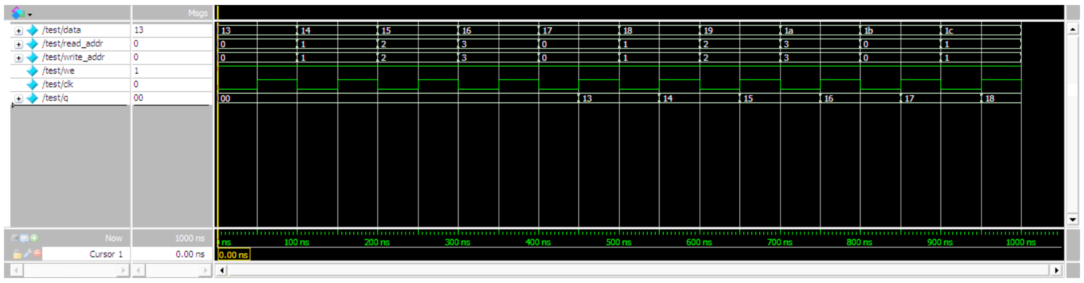


Figure 3: Exercise 9-1

Table 2: Exercise 9-1 Truth Table

we	1	1	1	1	1	1	1	1	1	1
clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
data	13	14	15	16	17	18	19	1a	1b	1c
read_addr	13	14	15	16	17	18	19	1a	1b	1c
write_addr	0	1	2	3	0	1	2	3	0	1
q	0	0	0	0	13	14	15	16	17	18
description	q= ram[0]	q= ram[1]	q= ram[2]	q= ram[3]	q= ram[0]	q= ram[1]	q= ram[2]	q= ram[3]	q= ram[0]	q= ram[1]

2 Exercise 9-2

2.1 Objective

Based on the previous code, add output which shows the data in each memory address. This code has decoder, data register, address register.

2.2 Operation

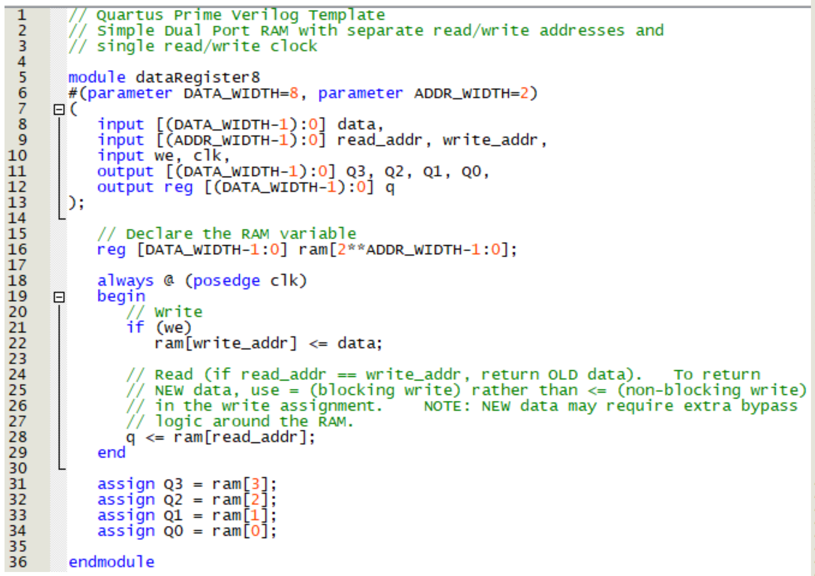
Table 3: Exercise 9-2 Operation detail

type	var	operation
input	data	data to be written into ram, 1 8bits data each time
input	read_addr	read address
input	write_addr	write address
input	we	write enable, which writes data into register
input	clk	clock
output	Q3	output address 3 data
output	Q2	output address 2 data
output	Q1	output address 1 data
output	Q0	output address 0 data
output	q	output data, return data from register

2.3 Code

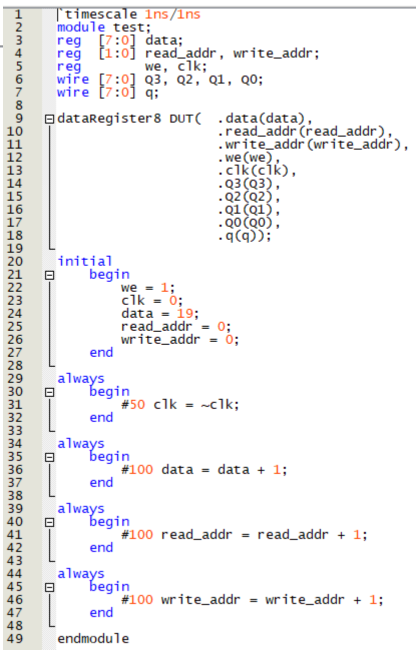
Only the portion beging modified is listed here.

1. Figure 4 Line31-34: Link the output ports to the four cooresponding memory address.
2. Figure 5 Line6-6: Add the output Q ports to the test file.
3. Figure 5 Line9-18: Include data register module.



```
1   // Quartus Prime Verilog Template
2   // Simple Dual Port RAM with separate read/write addresses and
3   // single read/write clock
4
5   module dataRegister8
6   #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=2)
7   (
8      input [(DATA_WIDTH-1):0] data,
9      input [(ADDR_WIDTH-1):0] read_addr, write_addr,
10     input we, clk,
11     output [(DATA_WIDTH-1):0] Q3, Q2, Q1, Q0,
12     output reg [(DATA_WIDTH-1):0] q
13  );
14
15     // Declare the RAM variable
16     reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
17
18     always @ (posedge clk)
19     begin
20        // Write
21        if (we)
22           ram[write_addr] <= data;
23
24        // Read (if read_addr == write_addr, return OLD data).   To return
25        // NEW data, use = (blocking write) rather than <= (non-blocking write)
26        // in the write assignment.    NOTE: NEW data may require extra bypass
27        // logic around the RAM.
28        q <= ram[read_addr];
29     end
30
31     assign Q3 = ram[3];
32     assign Q2 = ram[2];
33     assign Q1 = ram[1];
34     assign Q0 = ram[0];
35
36  endmodule
```

Figure 4: Main file



```
1   `timescale 1ns/1ns
2   module test;
3   reg  [7:0] data;
4   reg  [1:0] read_addr, write_addr;
5   reg        we, clk;
6   wire [7:0] Q3, Q2, Q1, Q0;
7   wire [7:0] q;
8
9   dataRegister8 DUT(  .data(data),
10                      .read_addr(read_addr),
11                      .write_addr(write_addr),
12                      .we(we),
13                      .clk(clk),
14                      .Q3(Q3),
15                      .Q2(Q2),
16                      .Q1(Q1),
17                      .Q0(Q0),
18                      .q(q));
19
20  initial
21      begin
22          we = 1;
23          clk = 0;
24          data = 19;
25          read_addr  = 0;
26          write_addr = 0;
27      end
28
29  always
30      begin
31          #50 clk = ~clk;
32      end
33
34  always
35      begin
36          #100 data = data + 1;
37      end
38
39  always
40      begin
41          #100 read_addr = read_addr + 1;
42      end
43
44  always
45      begin
46          #100 write_addr = write_addr + 1;
47      end
48
49  endmodule
```

Figure 5: Test file

2.4 Execution Result

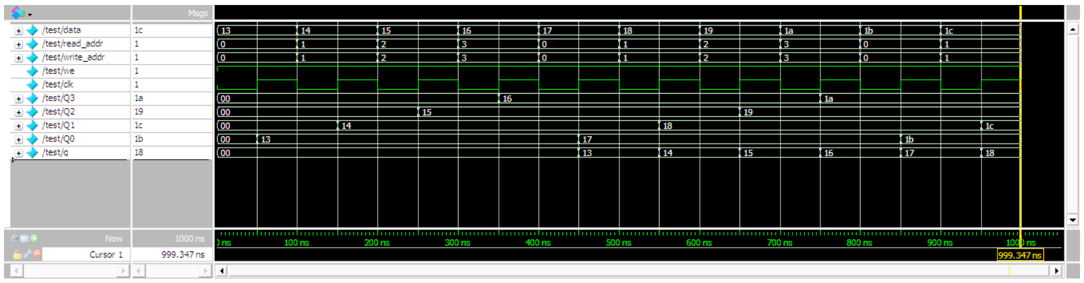


Figure 6: Exercise 9-2

Table 4: Exercise 9-2 Truth Table

we	1	1	1	1	1	1	1	1	1	1
clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
data	13	14	15	16	17	18	19	1a	1b	1c
read_addr	0	1	2	3	0	1	2	3	0	1
write_addr	0	1	2	3	0	1	2	3	0	1
Q3	00	00	00	16	16	16	16	1a	1a	1a
Q2	00	00	15	15	15	15	19	19	19	19
Q1	00	14	14	14	14	18	18	18	18	1c
Q0	13	13	13	13	17	17	17	17	1b	1b
q 0	0	0	0	0	13	14	15	16	17	18
description	q= ram[0]	q= ram[1]	q= ram[2]	q= ram[3]	q= ram[0]	q= ram[1]	q= ram[2]	q= ram[3]	q= ram[0]	q= ram[1]

3 Exercise 9-3

3.1 Objective

Add tri-state gate to the data register module. This code has decoder, data register, address register, tri-state gate, and mux. Which is the full structure diagram in page 24 of lecture powerpoint.

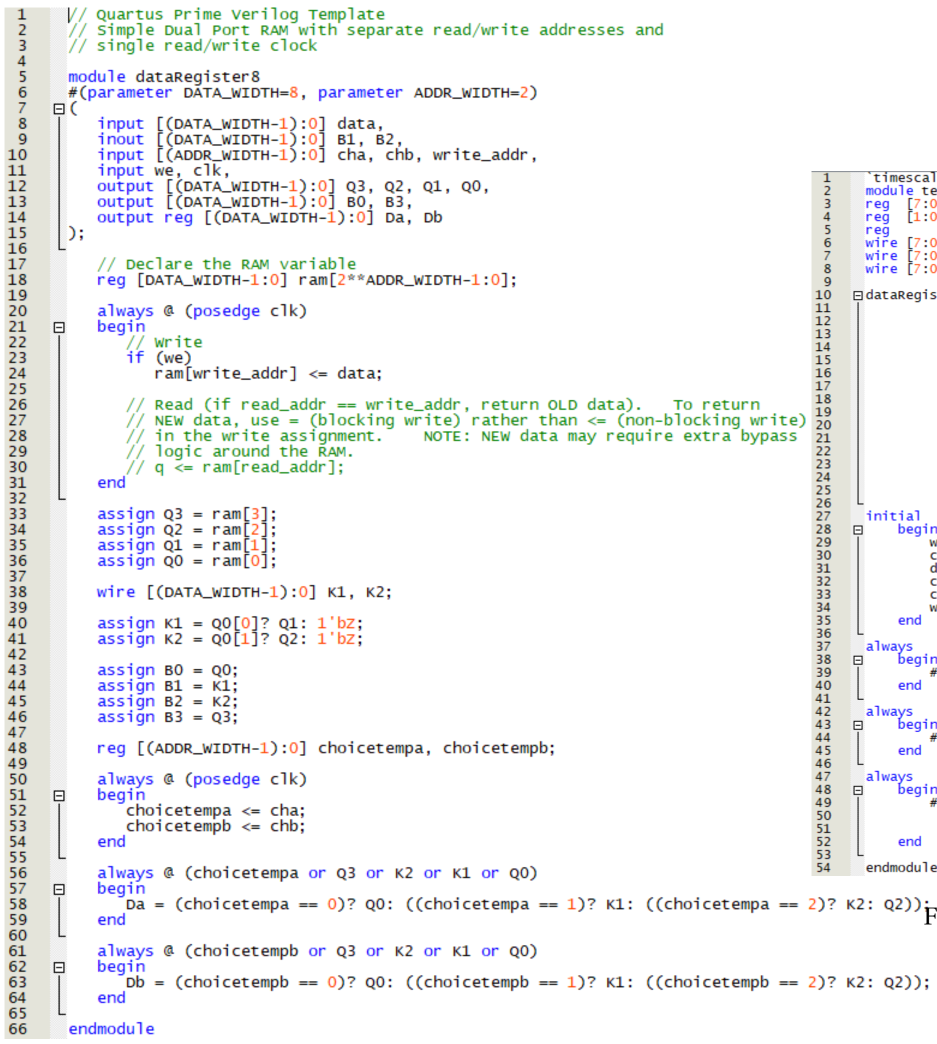
3.2 Operation

Table 5: Exercise 9-3 Operation detail

type	var	operation
input	data	data to be written into ram, 1 8bits data each time
inout	B1	tri-state gate control 1
inout	B2	tri-state gate control 2
input	cha	mux selector 1
input	chb	mux selector 2
input	write_addr	write address
input	we	write enable, which writes data into register
input	clk	clock
output	Q3	output address 3 data
output	Q2	output address 2 data
output	Q1	output address 1 data
output	Q0	output address 0 data
output	B0	provides data of memory 0
output	B3	provides data of memory 3
output	Da	output port 1, which output memory selected with B1
output	Db	output port 2, which output memory selected with B2

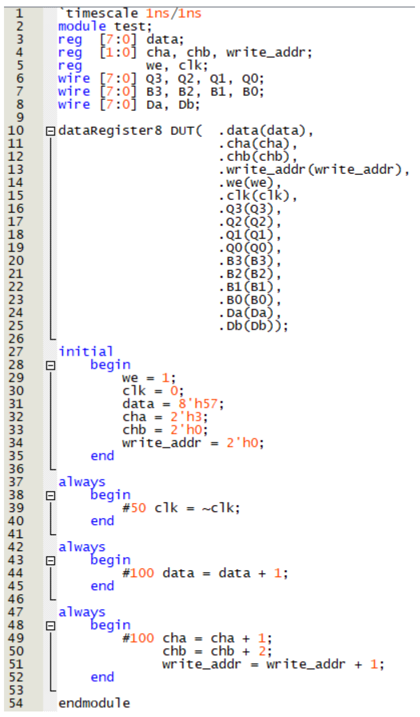
3.3 Code

1. Figure 7 Line6-15: Define module ports.
2. Figure 7 Line18-18: Declare ram variable.
3. Figure 7 Line20-31: If clock is triggered, and if write enabled, write data into ram.
4. Figure 7 Line33-36: Assign current ram values to output ports.
5. Figure 7 Line38-38: Declare wire to carry data.
6. Figure 7 Line40-41: Declare variable to send data based on Q0 or close up port, as tri-state gate control.
7. Figure 7 Line43-46: Connect output port to memory, inout port to tri-state gate control.
8. Figure 7 Line48-48: Declare register to store mux control signal.
9. Figure 7 Line50-54: If clock is triggered, pass mux selector signal to temparary register.
10. Figure 7 Line56-59: If either mux selector 1, memory 3, tri-state gate control 1, or mux selector 2, memory 0 triggered, pass data to output port 1.
11. Figure 7 Line61-64: If either mux selector 1, memory 3, tri-state gate control 1, or mux selector 2, memory 0 triggered, pass data to output port 2.
12. Figure 8 Line3-8: Declare testbench ports.
13. Figure 8 Line10-25: Include full data register module.
14. Figure 8 Line27-35: Set initial value for input ports.
15. Figure 8 Line37-40: Switch clock signal every 50ns.
16. Figure 8 Line42-45: Add data value by 1 every 100ns.
17. Figure 8 Line47-52: Change mux selector and write address every 100ns.



```verilog
// Quartus Prime Verilog Template
// Simple Dual Port RAM with separate read/write addresses and
// single read/write clock

module dataRegister8
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=2)
(
   input [(DATA_WIDTH-1):0] data,
   inout [(DATA_WIDTH-1):0] B1, B2,
   input [(ADDR_WIDTH-1):0] cha, chb, write_addr,
   input we, clk,
   output [(DATA_WIDTH-1):0] Q3, Q2, Q1, Q0,
   output [(DATA_WIDTH-1):0] B0, B3,
   output reg [(DATA_WIDTH-1):0] Da, Db
);

   // Declare the RAM variable
   reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

   always @ (posedge clk)
   begin
      // Write
      if (we)
         ram[write_addr] <= data;

      // Read (if read_addr == write_addr, return OLD data).   To return
      // NEW data, use = (blocking write) rather than <= (non-blocking write)
      // in the write assignment.    NOTE: NEW data may require extra bypass
      // logic around the RAM.
      // q <= ram[read_addr];
   end

   assign Q3 = ram[3];
   assign Q2 = ram[2];
   assign Q1 = ram[1];
   assign Q0 = ram[0];

   wire [(DATA_WIDTH-1):0] K1, K2;

   assign K1 = Q0[0]? Q1: 1'bz;
   assign K2 = Q0[1]? Q2: 1'bz;

   assign B0 = Q0;
   assign B1 = K1;
   assign B2 = K2;
   assign B3 = Q3;

   reg [(ADDR_WIDTH-1):0] choicetempa, choicetempb;

   always @ (posedge clk)
   begin
      choicetempa <= cha;
      choicetempb <= chb;
   end

   always @ (choicetempa or Q3 or K2 or K1 or Q0)
   begin
      Da = (choicetempa == 0)? Q0: ((choicetempa == 1)? K1: ((choicetempa == 2)? K2: Q2));
   end

   always @ (choicetempb or Q3 or K2 or K1 or Q0)
   begin
      Db = (choicetempb == 0)? Q0: ((choicetempb == 1)? K1: ((choicetempb == 2)? K2: Q2));
   end

endmodule
```

Figure 7: Main file



```verilog
`timescale 1ns/1ns
module test;
reg  [7:0] data;
reg  [1:0] cha, chb, write_addr;
reg        we, clk;
wire [7:0] Q3, Q2, Q1, Q0;
wire [7:0] B3, B2, B1, B0;
wire [7:0] Da, Db;

dataRegister8 DUT(  .data(data),
                    .cha(cha),
                    .chb(chb),
                    .write_addr(write_addr),
                    .we(we),
                    .clk(clk),
                    .Q3(Q3),
                    .Q2(Q2),
                    .Q1(Q1),
                    .Q0(Q0),
                    .B3(B3),
                    .B2(B2),
                    .B1(B1),
                    .B0(B0),
                    .Da(Da),
                    .Db(Db));

initial
    begin
        we = 1;
        clk = 0;
        data = 8'h57;
        cha = 2'h3;
        chb = 2'h0;
        write_addr = 2'h0;
    end

always
    begin
        #50 clk = ~clk;
    end

always
    begin
        #100 data = data + 1;
    end

always
    begin
        #100 cha = cha + 1;
             chb = chb + 2;
             write_addr = write_addr + 1;
    end

endmodule
```

Figure 8: Test file

3.4 Execution Result

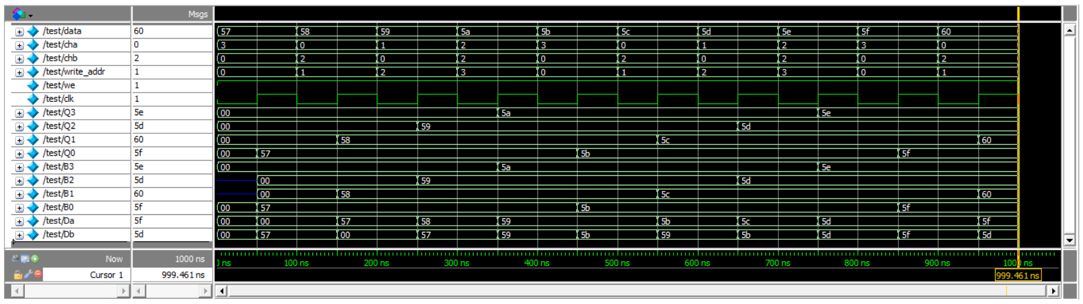


Figure 9: Exercise 9-3

Table 6: Exercise 9-4 Truth Table

we	1	1	1	1	1	1	1	1	1	1
clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
data	13	14	15	16	17	18	19	1a	1b	1c
cha	3	0	1	2	3	0	1	2	3	0
chb	0	2	0	2	0	2	0	2	0	2
write_addr	0	1	2	3	0	1	2	3	0	1
Q3	00	00	00	5a	5a	5a	5a	5e	5e	5e
Q2	00	00	59	59	59	59	5d	5d	5d	5d
Q1	00	58	58	58	58	5c	5c	5c	5c	60
Q0	57	57	57	57	5b	5b	5b	5b	5f	5f
B3	00	00	00	5a	5a	5a	5a	5e	5e	5e
B2	00	00	59	59	59	59	5d	5d	5d	5d
B1	00	58	58	58	58	5c	5c	5c	5c	60
B0	57	57	57	57	5b	5b	5b	5b	5f	5f
Da	00	57	58	59	59	5b	5c	5d	5d	5f
Db	57	00	57	59	5b	59	5b	5d	5f	5d
description	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]	Bx=Qx Da=Q[cha] Db=Q[chb]

4 Exercise 9-4

4.1 Objective

Simulate data register input and output mode with initial values.

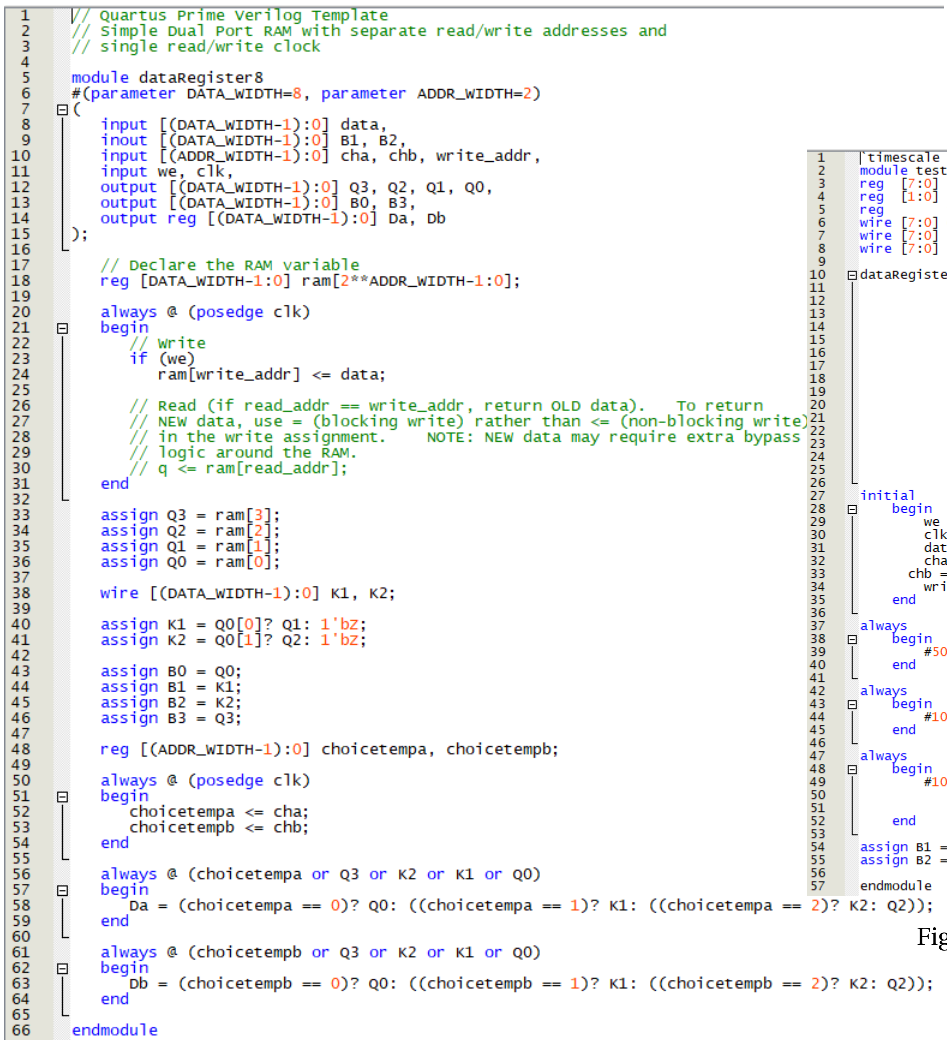
4.2 Operation

Same with previous section.

4.3 Code

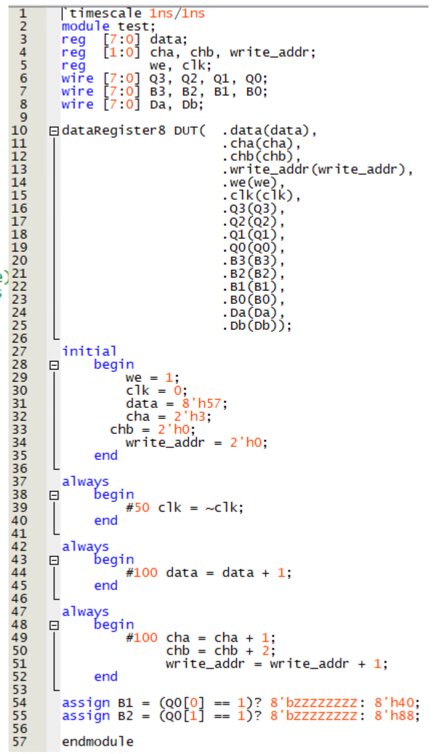
Only the modified part is shown below.

1. Figure 11 Line54-55: Add initial value for data register. If set, the port's initial value is set to be input.



```verilog
// Quartus Prime Verilog Template
// Simple Dual Port RAM with separate read/write addresses and
// single read/write clock

module dataRegister8
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=2)
(
    input [(DATA_WIDTH-1):0] data,
    inout [(DATA_WIDTH-1):0] B1, B2,
    input [(ADDR_WIDTH-1):0] cha, chb, write_addr,
    input we, clk,
    output [(DATA_WIDTH-1):0] Q3, Q2, Q1, Q0,
    output [(DATA_WIDTH-1):0] B0, B3,
    output reg [(DATA_WIDTH-1):0] Da, Db
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    always @ (posedge clk)
    begin
        // Write
        if (we)
            ram[write_addr] <= data;

        // Read (if read_addr == write_addr, return OLD data).   To return
        // NEW data, use = (blocking write) rather than <= (non-blocking write)
        // in the write assignment.    NOTE: NEW data may require extra bypass
        // logic around the RAM.
        // q <= ram[read_addr];
    end

    assign Q3 = ram[3];
    assign Q2 = ram[2];
    assign Q1 = ram[1];
    assign Q0 = ram[0];

    wire [(DATA_WIDTH-1):0] K1, K2;

    assign K1 = Q0[0]? Q1: 1'bz;
    assign K2 = Q0[1]? Q2: 1'bz;

    assign B0 = Q0;
    assign B1 = K1;
    assign B2 = K2;
    assign B3 = Q3;

    reg [(ADDR_WIDTH-1):0] choicetempa, choicetempb;

    always @ (posedge clk)
    begin
        choicetempa <= cha;
        choicetempb <= chb;
    end

    always @ (choicetempa or Q3 or K2 or K1 or Q0)
    begin
        Da = (choicetempa == 0)? Q0: ((choicetempa == 1)? K1: ((choicetempa == 2)? K2: Q2));
    end

    always @ (choicetempb or Q3 or K2 or K1 or Q0)
    begin
        Db = (choicetempb == 0)? Q0: ((choicetempb == 1)? K1: ((choicetempb == 2)? K2: Q2));
    end

endmodule
```

Figure 10: Main file



```verilog
`timescale 1ns/1ns
module test;
reg  [7:0] data;
reg  [1:0] cha, chb, write_addr;
reg        we, clk;
wire [7:0] Q3, Q2, Q1, Q0;
wire [7:0] B3, B2, B1, B0;
wire [7:0] Da, Db;

dataRegister8 DUT(  .data(data),
                    .cha(cha),
                    .chb(chb),
                    .write_addr(write_addr),
                    .we(we),
                    .clk(clk),
                    .Q3(Q3),
                    .Q2(Q2),
                    .Q1(Q1),
                    .Q0(Q0),
                    .B3(B3),
                    .B2(B2),
                    .B1(B1),
                    .B0(B0),
                    .Da(Da),
                    .Db(Db));

initial
    begin
        we = 1;
        clk = 0;
        data = 8'h57;
        cha = 2'h3;
      chb = 2'h0;
        write_addr = 2'h0;
    end

always
    begin
        #50 clk = ~clk;
    end

always
    begin
        #100 data = data + 1;
    end

always
    begin
        #100 cha = cha + 1;
             chb = chb + 2;
             write_addr = write_addr + 1;
    end

assign B1 = (Q0[0] == 1)? 8'bzzzzzzzz: 8'h40;
assign B2 = (Q0[1] == 1)? 8'bzzzzzzzz: 8'h88;

endmodule
```

Figure 11: Test file

4.4 Execution Result

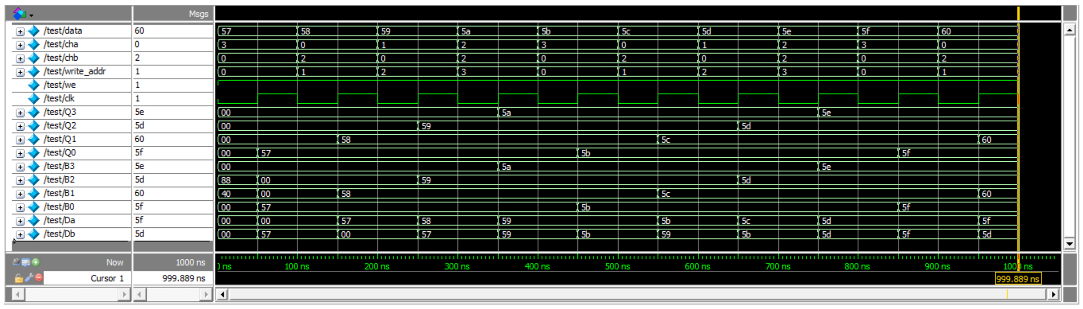


Figure 12: Exercise 9-4 B1:input B2:input

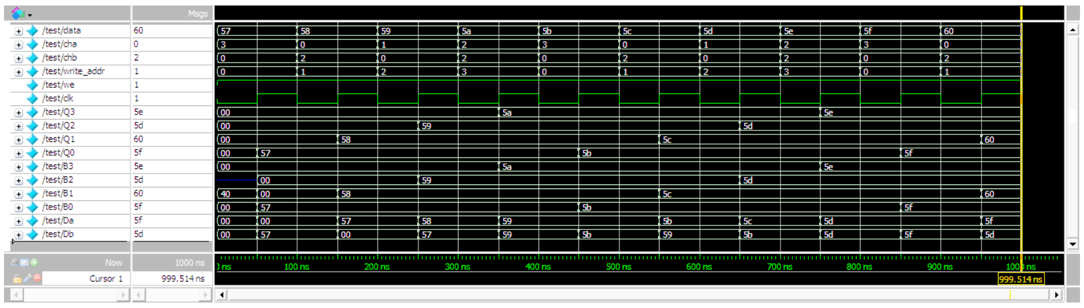


Figure 13: Exercise 9-4 B1:input B2:output

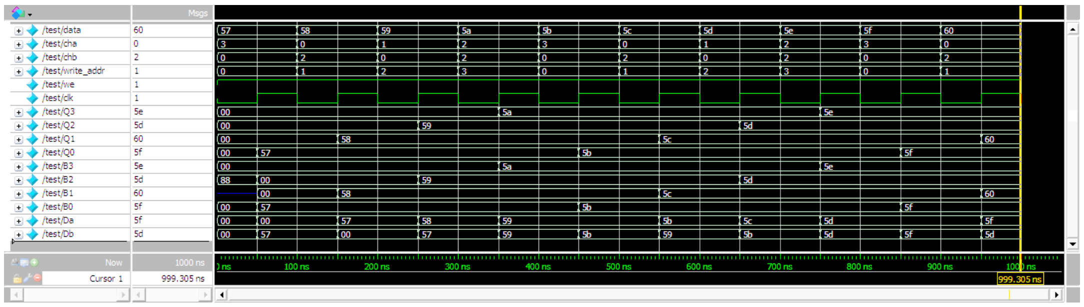


Figure 14: Exercise 9-4 B1:output B2:input

5 Exercise 9-5

5.1 Objective

Code a 8-bit ALU module.

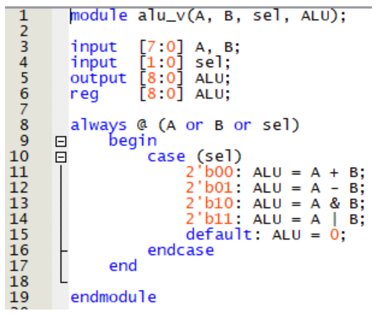
5.2 Operation

Table 7: Exercise 9-5 Operation detail

type	var	operation
input	A	input 8-bat data 1
input	B	input 8-bat data 2
input	sel	input 2-bit control signal
output	ALU	output 8-bit data with 1-bit carry or borrow

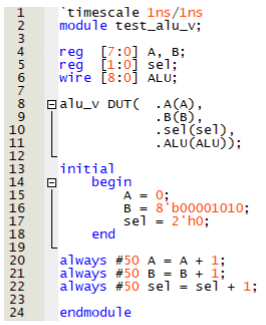
5.3 Code

1. Figure 15 Line3-6: Declare ports and internal signal.
2. Figure 15 Line8-17: Execute defined ALU operations if triggered by A, B, or sel.
3. Figure 15 Line4-6: Define variables.
4. Figure 15 Line8-11: Include ALU module.
5. Figure 15 Line13-18: Set initial values.
6. Figure 15 Line20-22: Add A, B, and sel 1 every 50ns.



```verilog
module alu_v(A, B, sel, ALU);

input   [7:0] A, B;
input   [1:0] sel;
output  [8:0] ALU;
reg     [8:0] ALU;

always @ (A or B or sel)
    begin
        case (sel)
            2'b00: ALU = A + B;
            2'b01: ALU = A - B;
            2'b10: ALU = A & B;
            2'b11: ALU = A | B;
            default: ALU = 0;
        endcase
    end

endmodule
```

Figure 15: Main file



```verilog
`timescale 1ns/1ns
module test_alu_v;

reg  [7:0] A, B;
reg  [1:0] sel;
wire [8:0] ALU;

alu_v DUT(  .A(A),
            .B(B),
            .sel(sel),
            .ALU(ALU));

initial
    begin
        A = 0;
        B = 8'b00001010;
        sel = 2'h0;
    end

always #50 A = A + 1;
always #50 B = B + 1;
always #50 sel = sel + 1;

endmodule
```

Figure 16: Test file

5.4 Execution Result

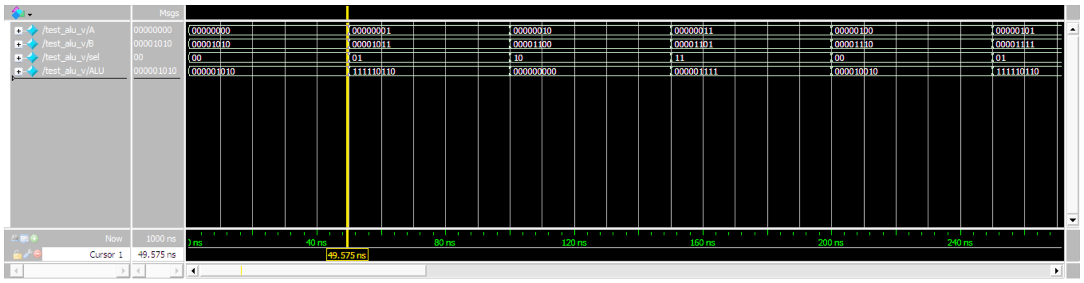


Figure 17: Exercise 9-5

Table 8: Exercise 9-5 Truth Table

Time (ns)	0~50	50~100	100~150	150~200
A	00000000	00000001	00000010	00000011
B	00001010	00001011	00001100	00001110
sel	00	01	10	11
ALU	000001010	111110110	00000000	000001111
description	A + B 00000000 00001010 ——— 00001010 pass	A - B 00000001 00001011 ——— 111110110 pass	A AND B 00000010 00001100 ——— 00000000 pass	A OR B 00000011 00001110 ——— 00001111 pass