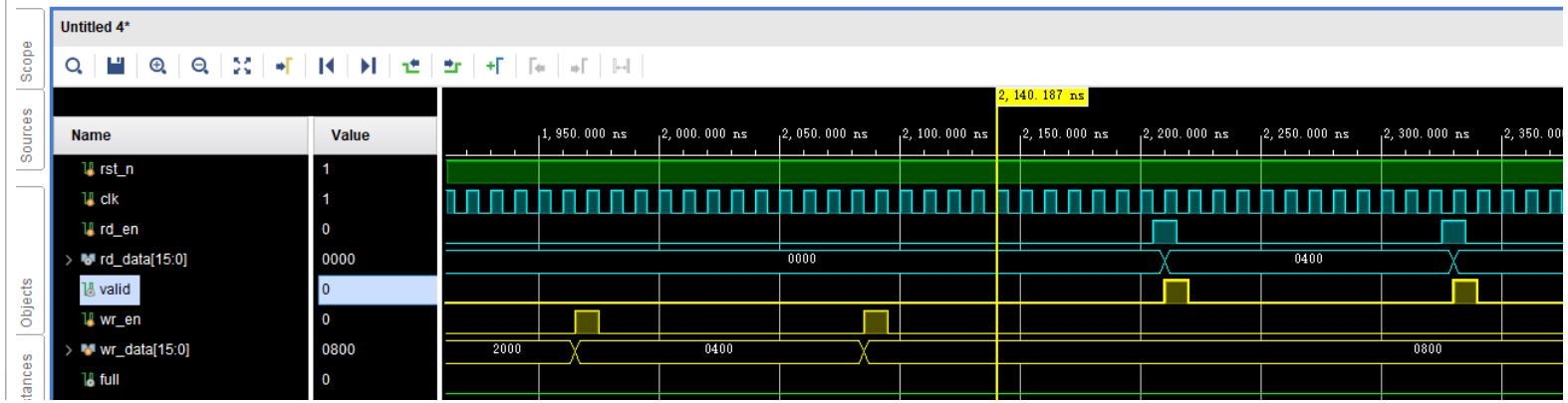
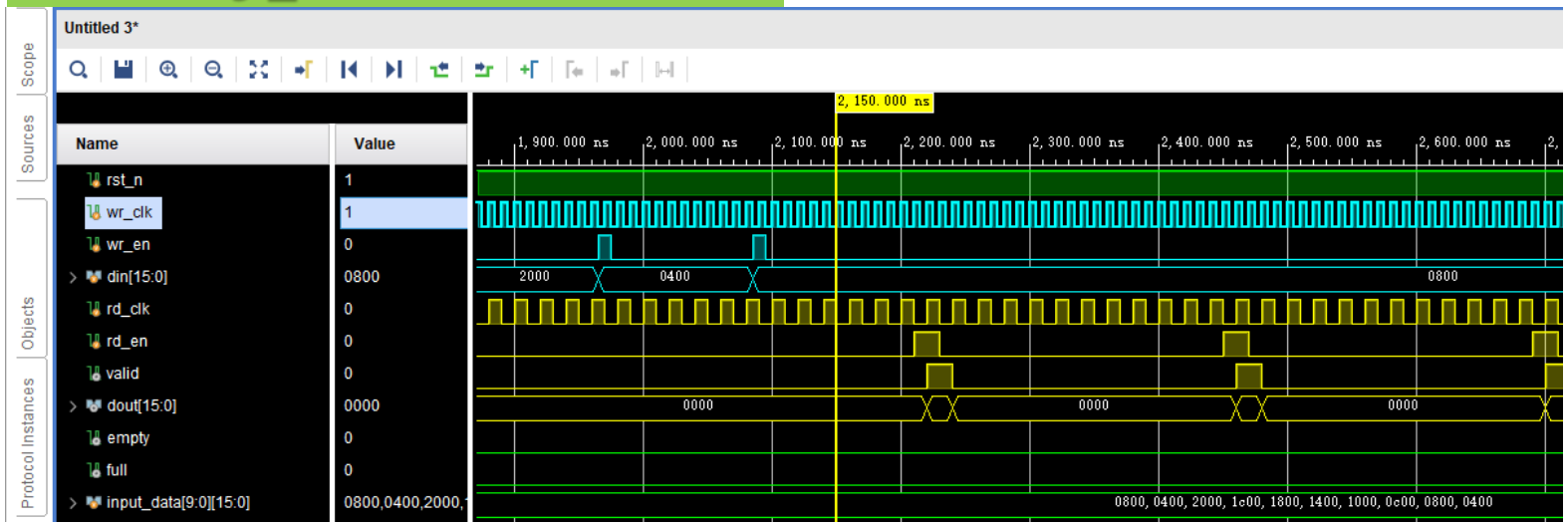
3. project_fifo

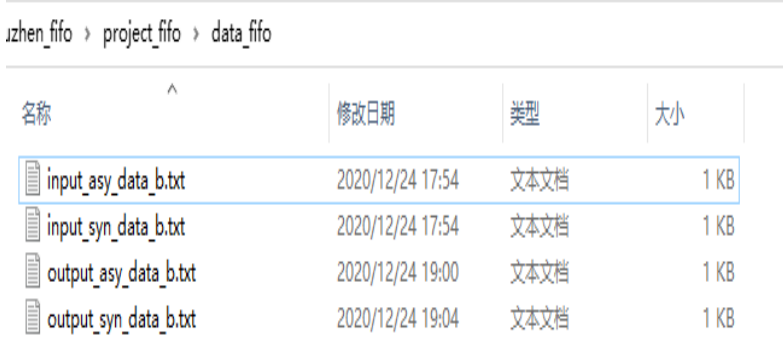
3.1. syn_fifo 同步fifo



3.2. asy_fifo 异步fifo



读写.txt文件



课堂练习

1. 仿真理解同步和异步的差别
2. 学会使用 readmemb 和 fwrite函数读写文件
3. 完成同步和异步FIFO设计与仿真

注意：ADDR_WIDTH = 5

3.1. syn_fifo 同步fifo

①

```
module syn_fifo #(
    parameter DATA_WIDTH = 16,
    parameter ADDR_WIDTH = 14,   //    depth

    parameter remain_num = 'd6
    ) (
    input                           clk ,
    input                           rst_n ,

    input                           rd_en ,
    output reg      [DATA_WIDTH-1:0]    rd_data ,
    output reg                      valid,

    input                           wr_en ,
    input           [DATA_WIDTH-1:0]    wr_data ,

    output                          full,
    output                          near_full,
    output                          empty
);
```

②

```
    wire [ADDR_WIDTH-1:0] rd_addr ;
    wire [ADDR_WIDTH-1:0] wr_addr ;
    reg [ADDR_WIDTH:0] rd_addr_ptr ;
    reg [ADDR_WIDTH:0] wr_addr_ptr ;


    assign rd_addr = rd_addr_ptr[ADDR_WIDTH-1:0];
    assign wr_addr = wr_addr_ptr[ADDR_WIDTH-1:0];
```

③

```
    reg [DATA_WIDTH-1:0] fifo_mem [(ADDR_WIDTH{1'b1}}:0];
    integer i;
    //*************************************
    // write fifo data
    //*************************************
    always @(posedge clk or negedge rst_n)
    begin
        if (!rst_n)
        begin
            for (i = 0;i <= {ADDR_WIDTH{1'b1}};i = i + 1)
                fifo_mem[i] <= {DATA_WIDTH{1'b0}};
        end
        else
        begin
            if (wr_en && (~full))
            begin
                fifo_mem[wr_addr] <= wr_data;
            end
            else
            begin
                fifo_mem[wr_addr] <= fifo_mem[wr_addr];
            end
        end
    end
    //*************************************
    // read fifo data
    //*************************************
    always @(posedge clk or negedge rst_n)
```

④

⑤

```
    begin
        if (!rst_n)
        begin
            rd_data <= {DATA_WIDTH{1'b0}};
            valid <= 1'b0;
        end
        else
        begin
            if (rd_en && (~empty))
            begin
                rd_data <= fifo_mem[rd_addr];
                valid <= 1'b1;
            end
            else
            begin
                rd_data <= rd_data;
                valid <= 1'b0;
            end
        end
    end
    //*************************************
    // write addr control
    //*************************************
    always @(posedge clk or negedge rst_n)
    begin
        if (!rst_n)
            wr_addr_ptr <= {(ADDR_WIDTH+1){1'b0}};
        else
        begin
```

3.1. syn_fifo 同步fifo

```
110         if (wr_en && (~full))
111             wr_addr_ptr <= wr_addr_ptr + 1'b1;
112         else
113             wr_addr_ptr <= wr_addr_ptr;
114     end
115 end
116 //************************************
117 // read addr control
118 //************************************
119 always @(posedge clk or negedge rst_n)
120 begin
121     if (!rst_n)
122         rd_addr_ptr <= {(ADDR_WIDTH+1){1'b0}};
123     else
124     begin
125         if (rd_en && (~empty))
126             rd_addr_ptr <= rd_addr_ptr + 1'b1;
127         else
128             rd_addr_ptr <= rd_addr_ptr;
129     end
130 end
131 //************************************
132 // full and empty judgement
133 //************************************
134 assign full = ((rd_addr == wr_addr) && (rd_addr_ptr[ADDR_WIDTH] != wr_addr_ptr[ADDR_WIDTH]));
135 assign near_full = ((rd_addr + remain_num) >= wr_addr) && (rd_addr_ptr[ADDR_WIDTH] != wr_addr_ptr[ADDR_WIDTH]);
136
137 assign empty = (rd_addr_ptr == wr_addr_ptr);
138 endmodule
```

6

7

8

注意：ADDR_WIDTH = 5

3.1. tb_syn_fifo

```
23  module tb_syn_fifo #(
24      parameter DATA_WIDTH = 16,
25      parameter ADDR_WIDTH = 14,   //    depth
26
27      parameter remain_num = 'd6
28       ) (
29
30      );
31
32  reg clk,rst_n;
33
34  reg rd_en;
35  wire [DATA_WIDTH-1:0] rd_data;
36  wire valid;
37
38  reg wr_en;
39  reg [DATA_WIDTH-1:0] wr_data;
40
41  wire full,near_full,empty;
42  syn_fifo syn_fifo(
43      .clk(clk),
44      .rst_n(rst_n),
45
46      .rd_en(rd_en),
47      .rd_data(rd_data),
48      .valid(valid),
49
50      .wr_en(wr_en),
51      .wr_data(wr_data),
```

```
53      .full(full),
54      .near_full(near_full),
55      .empty(empty)
56  );
57
58  initial begin
59      clk = 1;
60      forever #5 clk = ~clk;
61  end
62
63  initial begin
64      rst_n = 'd1;
65      rd_en = 'd0;
66      wr_en = 'd0;
67      wr_data = 'd0;
68      #50 rst_n = 'd0;
69      #50 rst_n = 'd1;
70  end
71
72  reg [DATA_WIDTH-1:0] input_data [9:0];
73  initial begin
74      #500;
75      $readmemb("C:/Users/yuzhi/Desktop/zhuzhen_fifo/project_fifo/data_fifo/input_syn_data_b.txt",input_data);
76  end
77
78  integer cnt_rd_data;
79  integer fp_rd_data;
80  initial begin
81      cnt_rd_data = 0;
```

3.1. tb_syn_fifo

```
82      fp_rd_data = $fopen("C:/Users/yuzhi/Desktop/zhuzhen_fifo/project_fifo/data_fifo/output_syn_data_b.txt","w");
83  end
84
85  integer ii;
86  initial begin
87      #1000;
88      for(ii = 0;ii < 10; ii++)begin
89          @(negedge clk)begin
90              wr_en <= 'd1;
91              wr_data <= input_data[ii];
92          end
93          @(negedge clk)begin
94              wr_en <= 'd0;
95          end
96          repeat (10) @(negedge clk);
97      end
98
99      for(ii = 0;ii < 10; ii++)begin
100         @(negedge clk)begin
101             rd_en <= 'd1;
102         end
103         @(negedge clk)begin
104             rd_en <= 'd0;
105         end
106         repeat (10) @(negedge clk);
107     end
108 end
109
110 always @(negedge clk)begin
111     if(syn_fifo.valid)begin
112         cnt_rd_data <= cnt_rd_data + 'd1;
113         if(cnt_rd_data == 'd9)begin
114             $fclose(fp_rd_data);
115         end
116         $fwrite(fp_rd_data,"%b\n",$signed(syn_fifo.rd_data));
117     end
118 end
119
120 endmodule
121
```

10

11

10

注意：ADDR_WIDTH = 5

3.2. asy_fifo 异步fifo

①

```
module asy_fifo #(
    parameter   data_width = 16,
    parameter   addr_width = 14
    ) (
     input                          rst_n,
     input                          wr_clk,
     input                          wr_en,
     input      [data_width-1:0]    din,
     input                          rd_clk,
     input                          rd_en,
     output reg                     valid,
     output reg [data_width-1:0]    dout,
     output                         empty,
     output                         full
    );
    localparam  data_depth = 1 << addr_width;

    reg    [addr_width:0]    wr_addr_ptr;
    reg    [addr_width:0]    rd_addr_ptr;
    wire   [addr_width-1:0]  wr_addr;
    wire   [addr_width-1:0]  rd_addr;
```

②

```
    wire   [addr_width:0]    wr_addr_gray;
    reg    [addr_width:0]    wr_addr_gray_d1;
    reg    [addr_width:0]    wr_addr_gray_d2;
    wire   [addr_width:0]    rd_addr_gray;
    reg    [addr_width:0]    rd_addr_gray_d1;
    reg    [addr_width:0]    rd_addr_gray_d2;
```

```
    reg [data_width-1:0] fifo_ram [data_depth-1:0];

    //=============================================================write fifo
    integer i;
    always@(posedge wr_clk or negedge rst_n) begin
        if(rst_n == 1'b0) begin
            for(i = 0; i < data_depth; i = i + 1 ) begin
                fifo_ram[i] <= 'h0;
            end
        end else begin
            if(wr_en && (~full)) begin
                fifo_ram[wr_addr] <= din;
            end else begin
                fifo_ram[wr_addr] <= fifo_ram[wr_addr];
            end
        end
    end
    //=============================================================read_fifo
    always@(posedge rd_clk or negedge rst_n) begin
        if(rst_n == 1'b0) begin
            dout <= 'h0;
            valid <= 1'b0;
        end else begin
            if(rd_en && (~empty)) begin
                dout <= fifo_ram[rd_addr];
                valid <= 1'b1;
            end else begin
                dout <= 'd0;            // 复位
                valid <= 1'b0;
```

3.2. asy_fifo 异步fifo

```
63          end
64      end
65  end
66  assign wr_addr = wr_addr_ptr[addr_width-1-:addr_width];
67  assign rd_addr = rd_addr_ptr[addr_width-1-:addr_width];
68  //=============================================================格雷码同步化
69  always @(posedge wr_clk, negedge rst_n)
70  begin
71      if(!rst_n)
72      begin
73          rd_addr_gray_d1 <= 'b0;
74          rd_addr_gray_d2 <= 'b0;
75      end
76      else
77      begin
78          rd_addr_gray_d1 <= rd_addr_gray;
79          rd_addr_gray_d2 <= rd_addr_gray_d1;
80      end
81  end
82  always @(posedge wr_clk or negedge rst_n) begin
83      if(rst_n == 1'b0)
84          wr_addr_ptr <= 'h0;
85      else if(wr_en && (~full))
86          wr_addr_ptr <= wr_addr_ptr + 1;
87      else
88          wr_addr_ptr <= wr_addr_ptr;
89  end
90  //=============================================================rd_clk
91  always @(posedge rd_clk or negedge rst_n)
```

3

4

```
92      begin
93          if(rst_n == 1'b0)
94          begin
95              wr_addr_gray_d1<=0;
96              wr_addr_gray_d2<=0;
97          end
98          else
99          begin
100             wr_addr_gray_d1 <= wr_addr_gray;
101             wr_addr_gray_d2 <= wr_addr_gray_d1;
102         end
103     end
104     always@(posedge rd_clk or negedge rst_n) begin
105         if(rst_n == 1'b0)
106             rd_addr_ptr <= 'h0;
107         else if(rd_en && (~empty))
108             rd_addr_ptr <= rd_addr_ptr + 1;
109         else
110             rd_addr_ptr <= rd_addr_ptr;
111     end
112
113     //============================================================ translation gary code
114     assign wr_addr_gray = (wr_addr_ptr >> 1) ^ wr_addr_ptr;
115     assign rd_addr_gray = (rd_addr_ptr >> 1) ^ rd_addr_ptr;
116
117     assign full = (wr_addr_gray == {~(rd_addr_gray_d2[addr_width-:2]),rd_addr_gray_d2[addr_width-2:0]})
118     assign empty = ( rd_addr_gray == wr_addr_gray_d2 );
119
120 endmodule
```

5

6

注意：ADDR_WIDTH = 5

3.2. tb_asy_fifo

```
23  module tb_asy_fifo #(
24      parameter   data_width = 16,
25      parameter   addr_width = 8
26       ) (
27
28      );
29
30  reg rst_n;
31
32  reg wr_clk,wr_en;
33  reg [data_width-1:0] din;
34
35  reg rd_clk,rd_en;
36
37  wire valid;
38  wire [data_width-1:0] dout;
39  wire empty,full;
40
41  asy_fifo asy_fifo(
42      .rst_n(rst_n),
43
44      .wr_clk(wr_clk),
45      .wr_en(wr_en),
46      .din(din),
47
48      .rd_clk(rd_clk),
49      .rd_en(rd_en),
50
51      .valid(valid),
```

```
52      .dout(dout),
53
54      .full(full),
55      .empty(empty)
56  );
57
58  initial begin
59      wr_clk = 1;
60      forever #5 wr_clk = ~wr_clk;
61  end
62
63  initial begin
64      rd_clk = 1;
65      forever #10 rd_clk = ~rd_clk;
66  end
67
68  initial begin
69      rst_n = 'd1;
70      rd_en = 'd0;
71      wr_en = 'd0;
72      din = 'd0;
73      #50 rst_n = 'd0;
74      #50 rst_n = 'd1;
75  end
76
77  reg [data_width-1:0] input_data [9:0];
78  initial begin
79      #500;
80      $readmemb("C:/Users/yuzhi/Desktop/zhuzhen_fifo/project_fifo/data_fifo/input_asy_data_b.txt",input_data);
```

⑦

3.2. tb_asy_fifo

```
81  end
82
83  integer cnt_rd_data;
84  integer fp_rd_data;
85  initial begin
86      cnt_rd_data = 0;
87      fp_rd_data = $fopen("C:/Users/yuzhi/Desktop/zhuzhen_fifo/project_fifo/data_fifo/output_asy_data_b.txt","w");
88  end
89
90  integer ii;
91  initial begin
92      #1000;
93      for(ii = 0;ii < 10; ii++)begin
94          @(negedge wr_clk)begin
95              wr_en <= 'd1;
96              din <= input_data[ii];
97          end
98          @(negedge wr_clk)begin
99              wr_en <= 'd0;
100         end
101         repeat (10) @(negedge wr_clk);
102     end
103
104     for(ii = 0;ii < 10; ii++)begin
105         @(negedge rd_clk)begin
106             rd_en <= 'd1;
107         end
108         @(negedge rd_clk)begin
109             rd_en <= 'd0;
110         end
111         repeat (10) @(negedge rd_clk);
112     end
113 end
114
115 always @(negedge rd_clk)begin
116     if(asy_fifo.valid)begin
117         cnt_rd_data <= cnt_rd_data + 'd1;
118         if(cnt_rd_data == 'd9)begin
119             $fclose(fp_rd_data);
120         end
121         $fwrite(fp_rd_data,"%b\n",$signed(asy_fifo.dout));
122     end
123 end
124
125 endmodule
126
```