同步 FIFO 设计

一、功能分析

- FIFO为空，不可从FIFO读数据，但可写；
- FIFO为满，不可向FIFO写数据，但可读；
- 非空非满时，FIFO可读、可写；
- FIFO的读写受同一时钟控制；
- FIFO的大小为N；

二、空满判定分析

- 当wr_ptr=rd_ptr时，FIFO数据为空；
- 当wr_ptr-rd_ptr=M - l 或 rd_ptr-wr_ptr=l 时， FIFO数据为满；
- 当wr_ptr>=rd_ptr时，wr_ptr-rd_ptr 为 FIFO 内 数据个数；
- 当wr_ptr<=rd_ptr时，M-(rd_ptr-wr_ptr) 为 FIFO 内数据个数；

三、端口信号分析

- 配置参数：width: 数据位宽
- 配置参数：depth: 存储器地址线长度
- 输入信号：clk, rst: 同步时钟，复位信号
- 输入信号：datain: 输入数据
- 输入信号：wr, rd：读写使能信号
- 输出信号：dataout: 读出数据
- 输出信号：full, empty: 存储器满空标志

四、VHDL 描述

4.1 顶层模块设计

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5   entity FIFO is
6       generic
7       (
```

```
 8           width : positive := 8;
 9           depth : positive := 8);
10       port ( clk: in std_logic;
11              rst: in std_logic;
12              datain : in std_logic_vector(width-1 downto 0);
13              dataout : out std_logic_vector(width -1 downto 0);
14              wr : in std_logic;
15              rd : in std_logic;
16              empty : out std_logic;
17              full : out std_logic);
18   end FIFO;
19
20   architecture Behavioral of FIFO is
21   component dualram is
22       generic
23       (
24           width : positive := 8;
25           depth : positive := 8);
26
27       port
28       (
29           ------------------------------ port a is only for writing --------------
     ----------
30           clka : in std_logic;
31           wr : in std_logic;
32           addra : in std_logic_vector(depth-1 downto 0);
33           datain : in std_logic_vector(width-1 downto 0);
34
35           ------------------------------ port b is only for reading --------------
     ----------
36           clkb : in std_logic;
37           rd : in std_logic;
38           addrb : in std_logic_vector(depth-1 downto 0);
39           dataout : out std_logic_vector(width -1 downto 0));
40   end component;
41
42   component read_pointer is
43       generic( depth: positive);
44       Port ( clk : in STD_LOGIC;
45              rst : in std_logic;
46              rq : in std_logic;
47              rd_pt : out std_logic_vector(depth-1 downto 0));
48   end component;
49
50   component write_pointer is
51       generic
52       ( depth: positive);
53       port (
```

```
54            clk : in std_logic;
55            rst: in std_logic;
56            wq : in std_logic;
57            wr_pt : out std_logic_vector(depth-1 downto 0));
58    end component;
59
60    component judge_status is
61        generic( depth: positive);
62        port ( clk : in STD_LOGIC;
63                rst : in std_logic;
64                wr_pt : in std_logic_vector(depth - 1 downto 0);
65                rd_pt : in std_logic_vector(depth - 1 downto 0);
66                empty : out std_logic;
67                full : out std_logic);
68    end component;
69
70    signal wr_pt, rd_pt: std_logic_vector(depth-1 downto 0);
71
72    begin
73        DRAM: dualram generic map(width, depth)
74                    port map(
75                         clka => clk,
76                         clkb => clk,
77                         wr => wr,
78                         rd => rd,
79                         datain => datain,
80                         dataout => dataout,
81                         addra => wr_pt,
82                         addrb => rd_pt);
83        WPointer: write_pointer generic map(depth)
84                        port map(
85                             clk => clk,
86                             wq => wr,
87                             rst => rst,
88                             wr_pt => wr_pt);
89
90       RPointer: read_pointer generic map(depth)
91                        port map(
92                             clk => clk,
93                             rq => rd,
94                             rst => rst,
95                             rd_pt => rd_pt);
96
97        Status: judge_status generic map(depth)
98                    port map(
99                         clk => clk,
100                        rst => rst,
101                        wr_pt => wr_pt,
```

```
                    rd_pt => rd_pt,
                    empty => empty,
                    full => full);
 end Behavioral;
```

4.2 双端口 RAM 设计

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dualram is
    generic
    (
        width : positive := 8;
        depth : positive := 8);

    port
    (
        ------------------------------ port a is only for writing ---------------
---------
        clka : in std_logic;
        wr : in std_logic;
        addra : in std_logic_vector(depth-1 downto 0);
        datain : in std_logic_vector(width-1 downto 0);
        ------------------------------ port b is only for reading ---------------
---------
        clkb : in std_logic;
        rd : in std_logic;
        addrb : in std_logic_vector(depth-1 downto 0);
        dataout : out std_logic_vector(width -1 downto 0));
end dualram;


architecture Behavioral of dualram is
type ram is array(2 ** depth - 1 downto 0) of std_logic_vector(width - 1 downto
0);
signal dualram: ram;
begin
    process(clka)
    begin
        if(clka'event and clka = '1') then
            if wr = '0' then
                dualram(conv_integer(addra)) <= datain;
            end if;
        end if;
```

```
    end process;

    process(clkb)
    begin
        if( clkb'event and clkb = '1') then
            if rd = '0'then
                dataout <= dualram(conv_integer(addrb));
            else
                dataout <= (others => '0');
            end if;
        end if;
    end process;
end Behavioral;
```

4.3 写地址计数器设计

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity write_pointer is
    generic
    ( depth: positive);
    port (
        clk : in std_logic;
        rst: in std_logic;
        wq : in std_logic;
        wr_pt : out std_logic_vector(depth-1 downto 0));
end write_pointer;

architecture Behavioral of write_pointer is
signal wr_pt_t : std_logic_vector(depth - 1 downto 0); -- write pointer counter
begin
    process(rst, clk)
        begin
            if rst = '0' then
                wr_pt_t <= (others => '0');
            elsif clk'event and clk = '1' then
                if wq = '0' then
                    wr_pt_t <= wr_pt_t + 1;
                end if;
            end if;
    end process;
    wr_pt <= wr_pt_t;
end Behavioral;
```

4.4 读地址计数器设计

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity read_pointer is
    generic( depth: positive);
    Port ( clk : in STD_LOGIC;
           rst : in std_logic;
           rq : in std_logic;
           rd_pt : out std_logic_vector(depth-1 downto 0));
end read_pointer;

architecture Behavioral of read_pointer is
signal rd_pt_t : std_logic_vector(depth - 1 downto 0); -- read pointer counter
begin
    process(rst, clk)
        begin
            if rst = '0' then
                rd_pt_t <= (others => '0');
            elsif clk'event and clk = '1' then
                if rq = '0' then
                    rd_pt_t <= rd_pt_t + 1;
                end if;
            end if;
    end process;
    rd_pt <= rd_pt_t;
end Behavioral;
```

4.5 空满状态产生器设计

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity judge_status is
    generic( depth: positive);
    port ( clk : in STD_LOGIC;
           rst : in std_logic;
           wr_pt : in std_logic_vector(depth - 1 downto 0);
           rd_pt : in std_logic_vector(depth - 1 downto 0);
           empty : out std_logic;
           full : out std_logic);
end judge_status;
```

```
14
15  architecture Behavioral of judge_status is
16  begin
17      process(rst, clk)
18      begin
19          if rst = '0' then
20              empty <= '1';
21          elsif clk'event and clk = '1' then
22              if wr_pt = rd_pt then
23                  empty <= '1';
24              else
25                  empty <= '0';
26              end if;
27          end if;
28       end process;
29
30       process(rst, clk)
31       begin
32          if rst = '0' then
33              full <= '0';
34          elsif clk'event and clk = '1' then
35              if wr_pt > rd_pt then
36                  if (rd_pt + 2 ** depth - 1) = wr_pt then
37                      full <= '1';
38                  else
39                      full <= '0';
40                  end if;
41              else
42                  if (wr_pt+1) = rd_pt then
43                      full <= '1';
44                  else
45                      full <= '0';
46                  end if;
47              end if;
48          end if;
49        end process;
50  end Behavioral;
```

五、仿真配置

- 仿真 Pipline: 系统复位 ⇒ 写入数据直至满 ⇒ 读出数据直至空

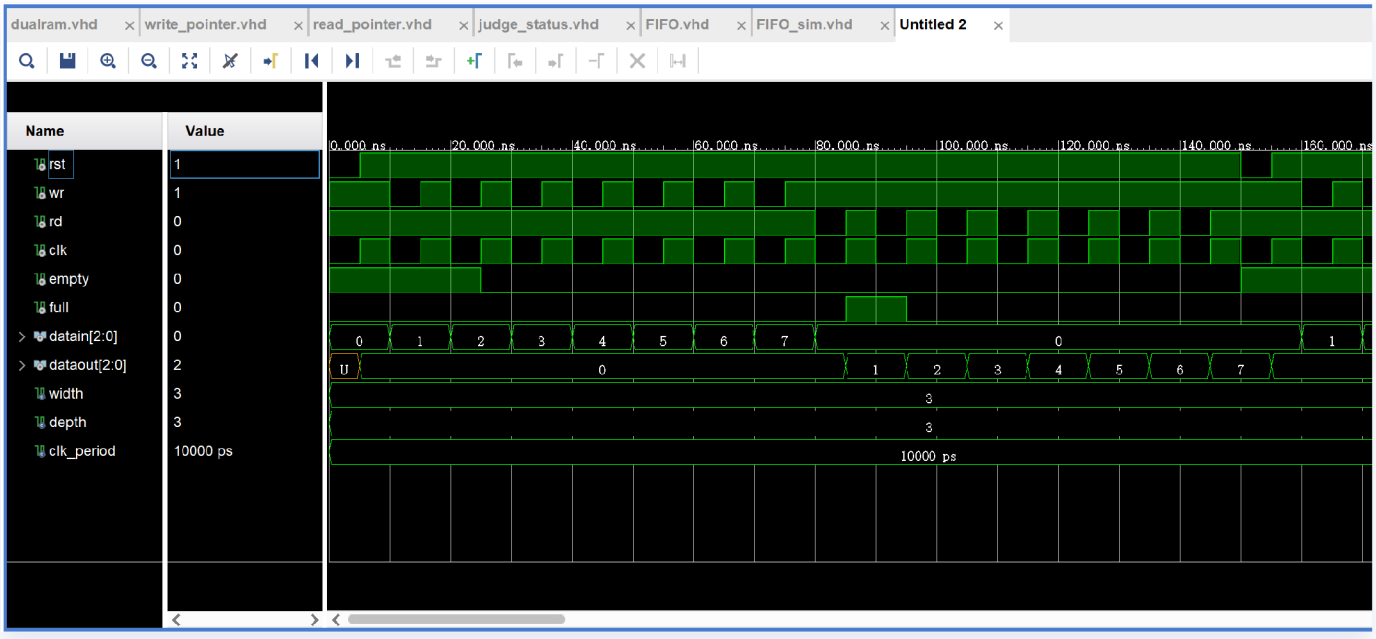
```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5   entity FIFO_sim is
```

```
6       -- Port ( );
7   end FIFO_sim;
8
9   architecture Behavioral of FIFO_sim is
10  component FIFO is
11      generic
12      (
13          width : positive := 8;
14          depth : positive := 8);
15      port ( clk: in std_logic;
16             rst: in std_logic;
17             datain : in std_logic_vector(width-1 downto 0);
18             dataout : out std_logic_vector(width -1 downto 0);
19             wr : in std_logic;
20             rd : in std_logic;
21             empty : out std_logic;
22             full : out std_logic);
23  end component;
24
25  constant width: integer := 3;      -- define 3 bits
26  constant depth: integer := 3;      -- define 000 - 111 address
27  signal rst, wr, rd: std_logic := '1';
28  signal clk, empty, full: std_logic := '0';
29  signal datain: std_logic_vector(width-1 downto 0) := "000";
30  signal dataout: std_logic_vector(width-1 downto 0) := "000";
31  constant clk_period : time := 10 ns;
32
33  begin
34      FIFO_Instance: FIFO generic map(width, depth)
35                     port map(
36                          clk => clk,
37                          rst => rst,
38                          datain => datain,
39                          dataout => dataout,
40                          empty => empty,
41                          full => full,
42                          wr => wr,
43                          rd => rd);
44
45       clk <= not clk after clk_period / 2;    -- clk production
46
47       process
48       begin
49              -- reset firstly
50              rst <= '0';
51              wait for clk_period / 2;
52              rst <= '1';
53              wait for clk_period / 2;
```

```
54
55              -- write 001-111 data to ram in turn
56              for i in 0 to 6 loop
57                  datain <= datain + 1;
58                  wr <= '0';
59                  wait for clk_period / 2;
60                  wr <= '1';
61                  wait for clk_period / 2;
62              end loop;
63
64              -- read 001-111 data from ram in turn
65              datain <= (others => '0');
66              for i in 0 to 6 loop
67                  rd <= '0';
68                  wait for clk_period / 2;
69                  rd <= '1';
70                  wait for clk_period / 2;
71              end loop;
72      end process;
73
74  end Behavioral;
```

六、功能仿真结果与分析

6.1 仿真电路时序图



- 从仿真结果可以看出，在系统初始化复位后，empty = 1, full = 0, rd = 1, wr = 1;
- depth = 3 时,表示存储器地址线宽度为 3, 即地址范围: 000~111, 但是 FIFO 存储器最后一个单元用于满标志判定, 因此实际大小 M = 7;

- 依次写入数据 1~7, 在写入完毕后下一周期存储器输出 full = 1 表示已经写满;
- 之后开始读取数据, 可以看出从存储器依次读取的数据是按照先进先出的顺序读出的, 当读出数据 7 后, empty = 1 表示存储器此时状态为空;

6.2 电路连接关系图