دانشکده مهندسی کامپیوتر

استاد : خانم دکتر پریا دربانی

سید مهدی رضوی

آذر ۱۴۰۱

فهرست مطالب

فهرست تصاویر

۱ مقدمه

در این آزمایش می‌خواهیم به ساخت یک حافظه به مدل FIFO با گنجایش ۱۲۸ * ۸ (یعنی ۱۲۸ سطر که در هر سطر ۸ بیت داشته‌باشیم.)

مفهوم

بدان معناست است که اولین عنصر ورودی ۸ بیتی ما ، اولین عنصر خروجی ۸ بیتی ما نیز باشد.

۲ پیاده سازی با زبان VHDL

۱.۲ توضیحاتی درباره پیاده‌سازی

یک حافظه FIFO را در این قسمت پیاده‌سازی خواهیم‌کرد.
در ۴ قسمت کدهای این ماژول سخت‌افزاری را توضیح خواهیم‌داد.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fifo is
  generic (
    g_WIDTH : natural := 8;
    g_DEPTH : integer := 32
    );
  port (
    i_rst_sync : in std_logic;
    i_clk      : in std_logic;

    -- FIFO Write Interface
    i_write_enable : in std_logic;
    i_write_data : in std_logic_vector(g_WIDTH-1 downto 0);
    o_full   : out std_logic;

    -- FIFO Read Interface
    i_read_enable : in std_logic;
    o_read_data : out std_logic_vector(g_WIDTH-1 downto 0);
    o_empty  : out std_logic
    );
end fifo;
```

در قطعه کد بالا پارامترهای لازم برای این ماژول سخت‌افزاری نوشته شده است.
طول(تعداد سطرها) و پهنا(تعداد بیت هر سطر) ، سپس سیگنال‌های ریست و clock
سیگنال های نوشتن که عبارتند از :
درخواست نوشتن ، داده‌برای نوشتن و پرچم پربودن ثبات
همچنین سیگنال‌های خواندن که عبارتند از :
درخواست خواندن ، داده خروجی حاصل از خواندن و پرچم خالی بودن ثبات

```
architecture rtl of fifo is

  type t_FIFO_DATA is array (0 to g_DEPTH-1) of std_logic_vector(g_WIDTH-1 downto 0);
  signal r_FIFO_DATA : t_FIFO_DATA := (others => (others => '0'));

  signal r_WR_INDEX : integer range 0 to g_DEPTH-1 := 0;
  signal r_RD_INDEX : integer range 0 to g_DEPTH-1 := 0;

  -- # Words in FIFO, has extra range to allow for assert conditions
  signal r_FIFO_COUNT : integer range -1 to g_DEPTH+1 := 0;

  signal w_FULL : std_logic;
  signal w_EMPTY : std_logic;
```

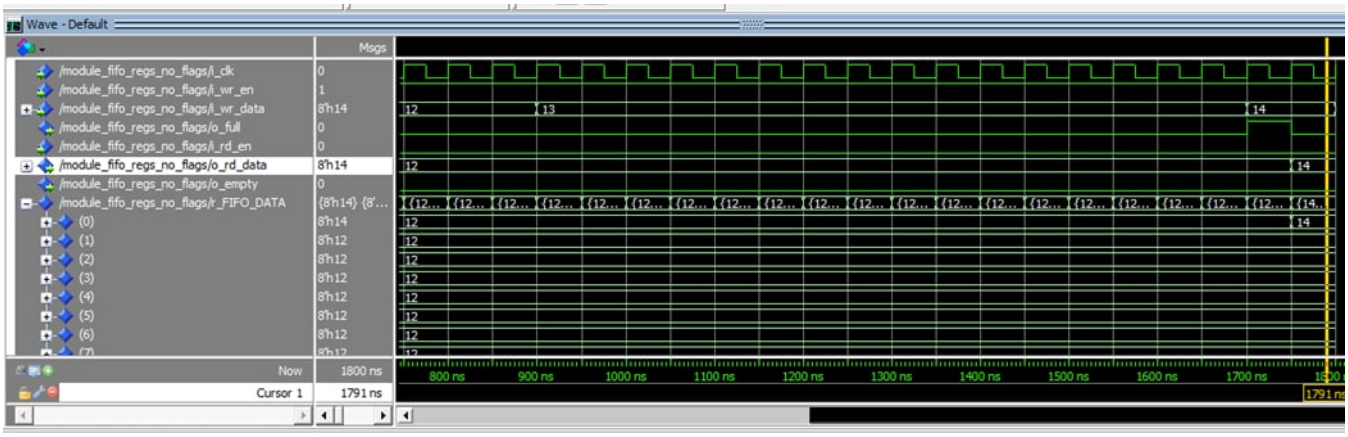
در قطعه کدبالا آرایه مدنظر و اندیس‌های خواندن و همچنین حداکثر میزان سطرهای ثبات را تعریف خواهیم کرد.

```
p_CONTROL : process (i_clk) is
  begin
    if rising_edge(i_clk) then
      if i_rst_sync = '1' then
        r_FIFO_COUNT <= 0;
        r_WR_INDEX <= 0;
        r_RD_INDEX <= 0;
      else
        if (i_write_enable = '1' and i_read_enable = '0') then
          r_FIFO_COUNT <= r_FIFO_COUNT + 1;
        elsif (i_write_enable = '0' and i_read_enable = '1') then
          r_FIFO_COUNT <= r_FIFO_COUNT - 1;
        end if;


        if (i_write_enable = '1' and w_FULL = '0') then
          if r_WR_INDEX = g_DEPTH-1 then
            r_WR_INDEX <= 0;
          else
            r_WR_INDEX <= r_WR_INDEX + 1;
          end if;
        end if;


        if (i_read_enable = '1' and w_EMPTY = '0') then
          if r_RD_INDEX = g_DEPTH-1 then
            r_RD_INDEX <= 0;
          else
            r_RD_INDEX <= r_RD_INDEX + 1;
          end if;
        end if;


        if i_write_enable = '1' then
          r_FIFO_DATA(r_WR_INDEX) <= i_write_data;
        end if;

      end if;                         -- sync reset
    end if;                           -- rising_edge(i_clk)
  end process p_CONTROL;
```

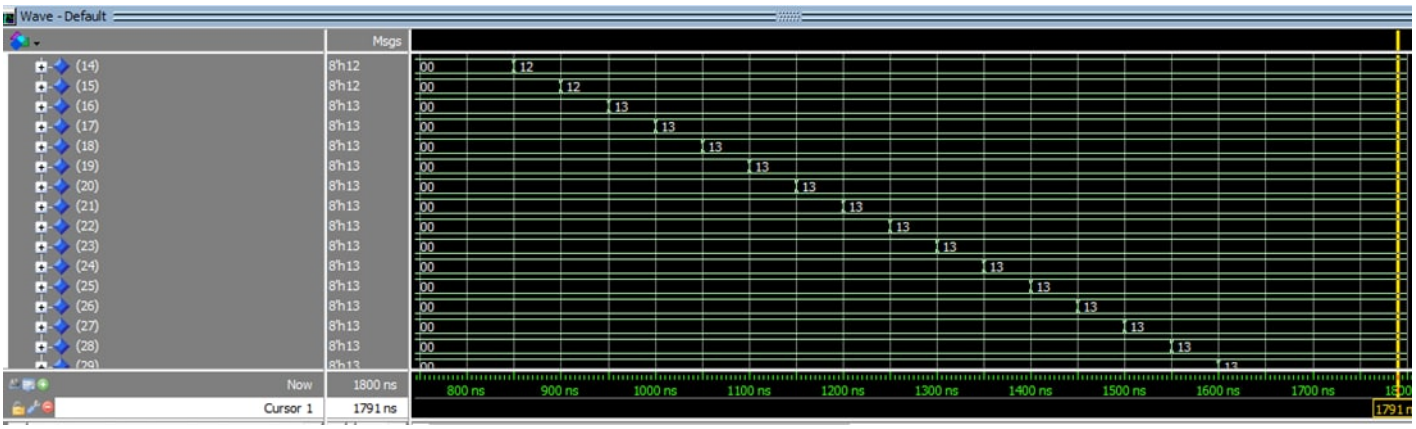
در این قسمت یک process سخت‌افزاری تعریف می‌کنیم که منطق کاری ما را اجرایی کند.
با چک کردن پرچم‌های پر و خالی بودن ثبات ، همچنین با چک کردن درخواست‌های خواندن و نوشتن ، به پرکردن خروجی می‌پردازیم.
این کار ابتدا با به‌روزرسانی اندیس‌ها صورت می‌گیرد.

```
o_read_data <= r_FIFO_DATA(r_RD_INDEX);

w_FULL <= '1' when r_FIFO_COUNT = g_DEPTH else '0';
w_EMPTY <= '1' when r_FIFO_COUNT = 0 else '0';

o_full <= w_FULL;
o_empty <= w_EMPTY;
```

در نهایت نیز پرچم‌های پر و خالی را به‌روزرسانی خواهیم کرد.



شکل ۱: شکل موج خروجی ۱ ثبات FIFO



شکل ۲: شکل موج خروجی ۲ ثبات FIFO

۳ نتیجه‌گیری

با توجه به نیاز ما برای استراتژی‌های متقاوت می‌توانیم به پیاده‌سازی متفاوت حافظه‌ها بر اساس نیازمان بپردازیم.