دانشکده مهندسی کامپیوتر

استاد : خانم دکتر پریا دربانی

سید مهدی رضوی

آذر ۱۴۰۱

فهرست مطالب

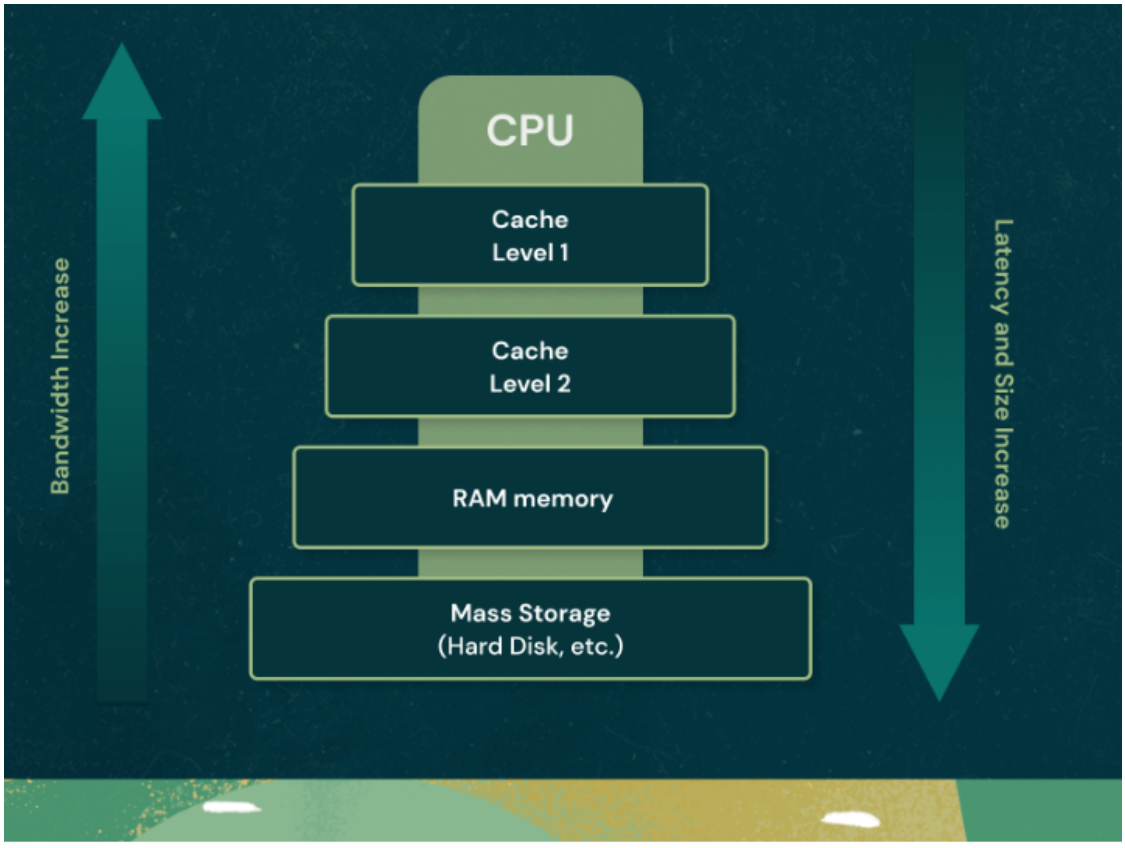
فهرست تصاویر

۱ مقدمه

در این آزمایش به بررسی انواع حافظه‌ها و پیاده‌سازی حافظه تصادفی RAM با زبان VHDL می‌پردازیم.
ابتدا یک معرفی مختصر از انواع حافظه‌های نهان ، حافظه تصادفی و همچنین حافظه دیسک خواهیم‌داشت سپس به پیاده‌سازی و نمایش شکل موج یک حافظه RAM ۱۲۸ * ۸ می‌پردازیم.

۲ مروری بر انواع حافظه‌ها

همانطور که در شکل ۱ مشاهده می‌کنید ، اولین حافظه بعد از پردازنده مرکزی یا همان CPU حافظه نهان cache است.
هدف از این لایه ذخیره سازی ذخیره زیر مجموعه ای از داده های اولیه (داده های مربوطه) است تا هرگونه درخواست آینده برای این داده ها سریعتر ارائه شود و در نتیجه زمان بارگذاری کلی برنامه کاهش یابد. این حافظه نهان در طی یک محاسبه قبلی داده ها ایجاد می شود یا ممکن است کپی از داده های قبلاً بازیابی شده باشد.
عنصر حافظه‌ای بعدی ما RAM خواهد بود که در واقع مخفف Random Access Memory است.
این حافظه یکی از اساسی ترین عناصر محاسبات است.
در واقع رابطه این حافظه با حافظه‌نهان ، مانند رابطه حافظه نهان با پردازنده مرکزی خواهد بود و نقش معادل با آن را برای ما ایفا خواهد کرد.
RAM یک بانک حافظه موقت است که کامپیوتر شما اطلاعاتی را که برای بازیابی سریع نیاز دارد، در آن ذخیره می کند. RAM داده‌ها را به راحتی در دسترس نگه می‌دارد تا پردازنده شما بتواند به سرعت آن‌ها را بدون نیاز به ذخیره‌سازی طولانی‌مدت برای تکمیل کارهای پردازش فوری پیدا کند.
Hard Disk هم نهایی‌ترین حافظه ما خواهدبود و کمترین سرعت را در بین این حافظه‌هایی که بیان کردیم خواهد داشت اما میزان ظرفیت آن از دو حافظه بیان‌شده بالا بیشتر خواهد بود.



شکل ۱: سلسله مراتب حافظه‌ها

۳ پیاده سازی با زبان VHDL

۱.۳ توضیحاتی درباره پیاده‌سازی

همانطور که در کد مشخص است ، با توجه به این که ما داریم یک حافظه با ۱۲۸ خانه را پیاده سازی می‌کنیم ، می‌بایستی که آدرس ما ۷ بیتی باشد. $(128 = 2^7)$ در نتیجه ثبات آدرس رم ما ۷ بیتی خواهد بود.
از طرفی دیگر ، حافظه RAM ما می‌بایستی در هر خانه خود یک متغیر ۸ بیتی را ذخیره کند. در نتیجه ثبات داده ما ۸ بیتی خواهد بود.
در آرایه RAM ARRAY نیز ۱۲۸ تا مقدار متغیر خواهیم داشت که هر کدام از این متغیرها ۸ بیتی هستند.
اگر سیگنال خواندن فعال بود ، داده ورودی RAMDATAIN را در همان خانه‌ای از آرایه RAM ARRAY که ثبات آدرس اشاره می‌کند ، قرار می‌دهیم.
برای پرکردن خانه‌ای که ثبات آدرس اشاره می‌کند ، باید ابتدا مقدار ثبات آدرس را به مقدار unsigned تبدیل کرد و سپس عملیات خواندن را انجام دهیم.
برای عملیات نوشتن نیز عکس این عمل را انجام خواهیم داد و مقدار ثبات خروجی حافظه RAMDATAOUT را پر می‌کنیم.

```
entity RAM is
port(
 RAM_ADDRESS: in std_logic_vector(6 downto 0); -- Address to write/read RAM
 RAM_DATA_IN: in std_logic_vector(7 downto 0); -- Data to write into RAM
 RAM_WRITE: in std_logic; -- Write enable
 RAM_CLOCK: in std_logic; -- clock input for RAM
 RAM_DATA_OUT: out std_logic_vector(7 downto 0) -- Data output of RAM
);
end RAM;

architecture Behavioral of RAM is
-- define the new type for the 128x8 RAM
type RAM_ARRAY is array (0 to 127 ) of std_logic_vector (7 downto 0);
-- initial values in the RAM
signal RAM: RAM_ARRAY :=(
   x"55",x"66",x"77",x"67",x"99",x"00",x"00",x"11",-- 0x04:
   x"00",x"00",x"00",x"00",

   x"00",x"00",x"00",x"00",
   x"00",x"00",x"00",x"00",
   x"00",x"00",x"00",x"00",
   x"00",x"00",x"00",x"00",
   x"00",x"00",x"00",x"00",
   x"00",x"00",x"00",x"00",
   x"00",x"00",x"00",x"00"
   );
begin
process(RAM_CLOCK)
begin
 if(rising_edge(RAM_CLOCK)) then
 if(RAM_WRITE='1') then
 RAM(to_integer(unsigned(RAM_ADDRESS))) <= RAM_DATA_IN;
 end if;

 end if;
end process;

 RAM_DATA_OUT <= RAM(to_integer(unsigned(RAM_ADDRESS)));
end Behavioral;
```

۲.۳ تست کردن حافظه رم

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
-- VHDL testbench code for the single-port RAM
ENTITY tb_RAM IS
END tb_RAM;

ARCHITECTURE behavior OF tb_RAM IS

    -- Component Declaration for the single-port RAM in VHDL

    COMPONENT RAM
    PORT(
         RAM_ADDRESS : IN std_logic_vector(6 downto 0);
         RAM_DATA_IN : IN std_logic_vector(7 downto 0);
         RAM_WRITE : IN std_logic;
         RAM_CLOCK : IN std_logic;
         RAM_DATA_OUT : OUT std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal RAM_ADDRESS : std_logic_vector(6 downto 0) := (others => '0');
   signal RAM_DATA_IN : std_logic_vector(7 downto 0) := (others => '0');
   signal RAM_WRITE : std_logic := '0';
   signal RAM_CLOCK : std_logic := '0';

  --Outputs
   signal RAM_DATA_OUT : std_logic_vector(7 downto 0);

   -- Clock period definitions
   constant RAM_CLOCK_period : time := 10 ns;
```

```
BEGIN
   uut: RAM PORT MAP (
         RAM_ADDRESS => RAM_ADDRESS,
         RAM_DATA_IN => RAM_DATA_IN,
         RAM_WRITE => RAM_WRITE,
         RAM_CLOCK => RAM_CLOCK,
         RAM_DATA_OUT => RAM_DATA_OUT
       );

  RAM_CLOCK_process :process
  begin
  RAM_CLOCK <= '0';
  wait for RAM_CLOCK_period/2;
  RAM_CLOCK <= '1';
  wait for RAM_CLOCK_period/2;
   end process;

  stim_proc: process
   begin
  RAM_WRITE <= '0';
  RAM_ADDRESS <= "0000000";
  RAM_DATA_IN <= x"FF";
     wait for 100 ns;
  -- start reading data from RAM
  for i in 0 to 5 loop
  RAM_ADDRESS <= RAM_ADDRESS + "0000001";
     wait for RAM_CLOCK_period*5;
  end loop;
  RAM_ADDRESS <= "0000000";
  RAM_WRITE <= '1';
  -- start writing to RAM
  wait for 100 ns;
  for i in 0 to 5 loop
  RAM_ADDRESS <= RAM_ADDRESS + "0000001";
  RAM_DATA_IN <= RAM_DATA_IN-x"01";
     wait for RAM_CLOCK_period * 5;
  end loop;
  RAM_WRITE <= '0';
     wait;
   end process;

END;
```

شکل ۲: شکل موج خروجی حافظه RAM

شکل ۳: شکل موج خروجی(۲) حافظه RAM

۴ نتیجه‌گیری

در این آزمایش به نحوه طراحی یک سیستم حافظه‌ای معروف به RAM پرداختیم.
میزان بیت اختصاصی به هر ثبات و وابستگی آن به تعداد خطوط حافظه با تابع لگاریتم قابل بررسی خواهد بود.