به نام خدا

پروژه:

پیاده سازی رم و کش

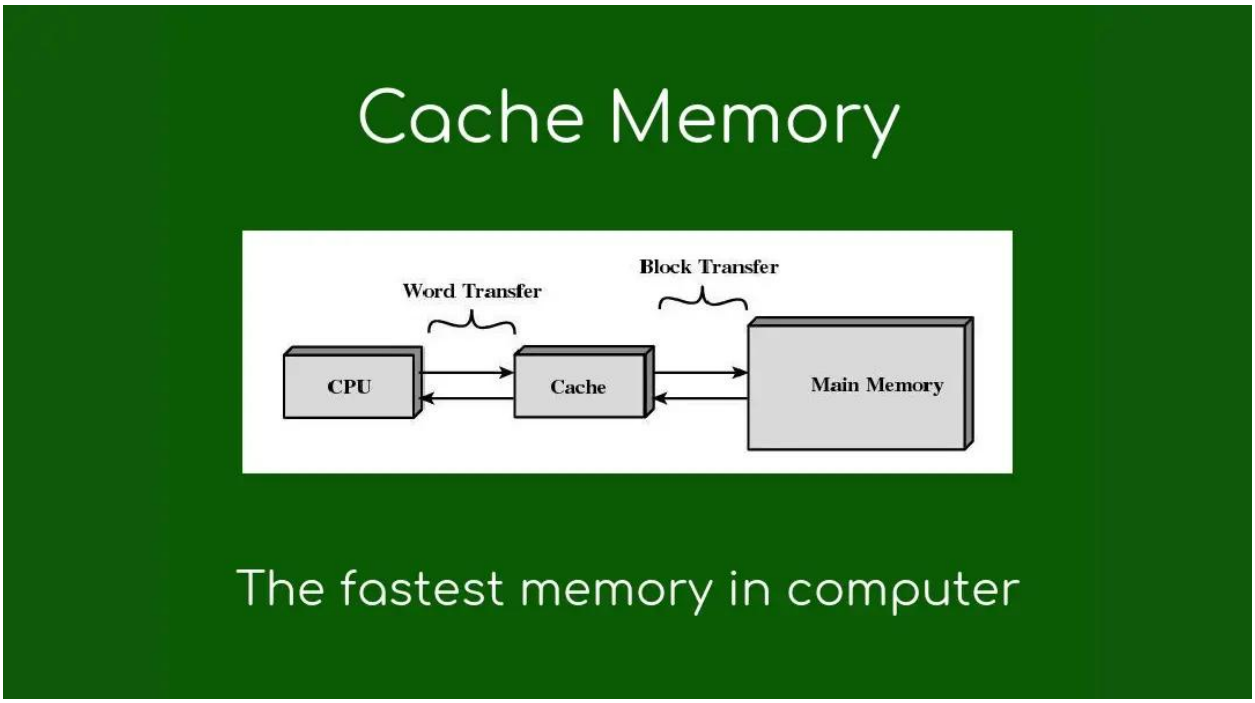
اعضای گروه:

کیانمهر رعنایی

سپیده سام خانیان

محمدفاضل عبدحقیقی

هدف از این پروژه پیاده سازی رم و کش و نمایش عملکرد بین انها است.



همان طور که در شکل بالا میبینید زمانی که cpu به داده ای در حافظه اصلی نیاز داشته باشد ابتدا در کش چک می کند اگر در ان قرار داشت با سرعت خوبی ان را دریافت می کند ، اما اگر در کش وجود نداشت در ان صورت به حافظه اصلی می رود و از ان میخواند و در کش می نویسید که اگر در اینده لازم شد، با سرعت بهتری به ان دسترسی داشته باشد و این گونه به داده مورد نظرش می رسد.

روش نگاشت مستقیم

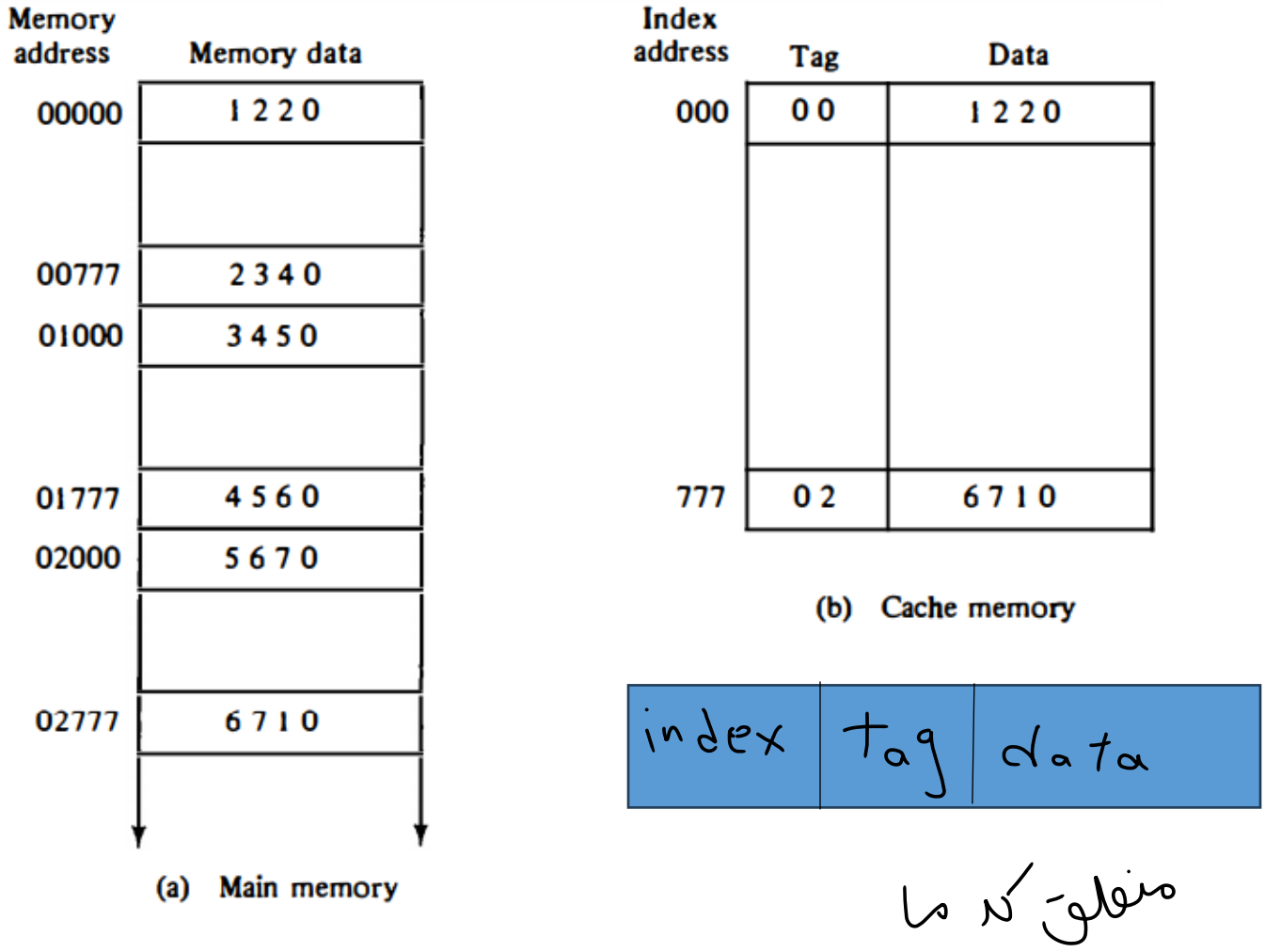


Figure 13 Direct mapping cache organization.

ما در کدمان index , tag را با هم در یک متغییر ادرس نگهداری کردیم و در هر سطر از رم و کش ما کل ادرس و داده را کنار هم قرار دادیم و بعد با جداسازی بیت های این ادرس به index ,tag می رسیدیم.

در گام نخست سه کامپوننت به نام های system_top، cache، ram تعریف می کنیم که system_top ارتباط بین دو کامپوننت دیگر است.

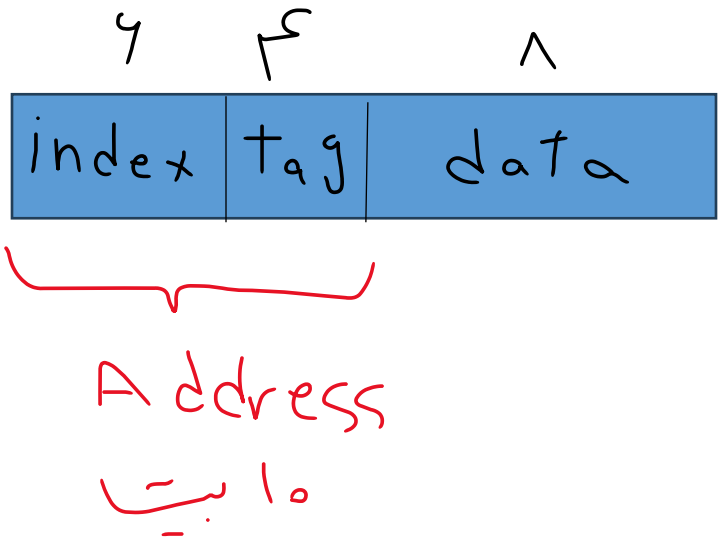
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Cache is
    generic (
        DATA_WIDTH  : positive := 8; -- Width of data in bits
        ADDR_WIDTH  : positive := 10; -- Width of address in bits
        INDEX_WIDTH : positive := 4;
        CACHE_SIZE  : positive := 16 -- Number of cache lines
    );
    port (
        clk    : in  STD_LOGIC;
        rst    : in  STD_LOGIC;
        cache_read: in  STD_LOGIC;
        cache_write  : in  STD_LOGIC;
        address : in  STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
        data_in : in  STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
        hit : out std_logic;
        data_out: out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
    );
end Cache;

architecture Behavioral of Cache is
    type CacheMemory is array (CACHE_SIZE-1 downto 0) of STD_LOGIC_VECTOR(ADDR_WIDTH+DATA_WIDTH-1 downto 0);

    signal cache_memory : CacheMemory;

begin
    process (clk, rst)
    variable is_hit : std_logic;
    variable target : std_logic_vector(ADDR_WIDTH+DATA_WIDTH-1 downto 0);
    begin
        if rst = '1' then
            -- Reset logic
            -- Initialize the cache memory
            for i in CacheMemory'range loop
                cache_memory(i) <= (others => '0');
            end loop;
        elsif rising_edge(clk) then --- 14 - 8
            if cache_memory(conv_integer(address(ADDR_WIDTH -1 downto ADDR_WIDTH-INDEX_WIDTH)))(ADDR_WIDTH+DATA_WIDTH-INDEX_WIDTH-1 downto DATA_WIDTH) = address(ADDR_WIDTH-INDEX_WIDTH-1 do
wnto 0) then
             hit <= '1';
             is_hit := '1';
             target := cache_memory(conv_integer(address(ADDR_WIDTH -1 downto ADDR_WIDTH-INDEX_WIDTH)));
            else
             hit <= '0';
            end if ;
            -- Cache operation logic
            if is_hit = '1' then
                data_out <= target;
            end if;
            if cache_read = '1' then
                -- Read operation  -- 19 -> 4 : 16 bit |
                data_out <= cache_memory(conv_integer(address(ADDR_WIDTH -1 downto ADDR_WIDTH-INDEX_WIDTH)))(DATA_WIDTH-1 downto 0);
            end if;
            if cache_write = '1' then
                -- Write operation
                cache_memory(conv_integer(address(ADDR_WIDTH -1 downto ADDR_WIDTH-INDEX_WIDTH))) <= address & data_in;
                data_out <= cache_memory(conv_integer(address(ADDR_WIDTH -1 downto ADDR_WIDTH-INDEX_WIDTH)))(DATA_WIDTH-1 downto 0);
            end if;
        end if;
    end process;
end Behavioral;

```

Cache component

در این کامپوننت ابتدا کتابخانه هایی که لازم داریم را تعریف می کنیم تا بتوانیم از انها استفاده کنیم.

سپس در entity مقادیری که میخواهیم جنریک باشند مانند طول ادرس ، طول داده و طول ایندکس را تعریف می کنیم و به انها مقدار اولیه ای می دهیم و بعد از ان ورودی ها و خروجی های خود را تعریف می کنیم.

Clk	تک بیتی
rst	یک بیتی
cache_read	قابلیت خواندن از کش، تک بیتی
cache_write	قابلیت نوشتن در کش، یک بیتی
address	ادرسی که میخواهیم در ان بخوانیم یا بنویسیم، چند بیتی (در جنریک مشخص می شود)
data_in	داده ای که به عنوان ورودی می گیریم
Hit	اگر داده مورد نظر در کش وجود داشته باشد این ، این مقدار ۱ می شود
data_out	خروجی این کامپوننت



هر سطر در کش

۱۸ بیت

که کش ما ۱۶ سطر دارد

ما یک ارایه دو بعدی به نام cache_memory با اندازه ۱۶*۱۸ تعریف می کنیم چون هر سطر ۱۸ بیت است و کش ما ۱۶ سطر دارد.

در ادامه ما یک process حساس به clk ,rst تعریف می کنیم که در ان variable های مورد نیازمان را تعریف کردیم

موفق بودن یا نبودن(۱ بیت)	Is_hit
برای نگهداری یک سطر از کش (۱۸ بیتی)	Target

و سپس شروع می کنیم.

اگر rst فعال باشد، تمام سطر های ارایه cache_memory را صفر می کنیم در غیر این صورت در لبه بالا رنده کلاک چک می کنیم اگر tag ادرس خواسته شده با tag ان ادرس در کش(ارایه دو بعدی ما) برابر بود ، یعنی ان داده در کش وجود داشته و hit را برابر یک قرار می دهیم چون در کش وجود داشت و بعد کل سطری که ادرس موجود بود را در target میریزیم و is_hit را برابر یک قرار می دهیم. گر آن شرط بر قرار نبود hit را برابر صفر می گذاریم.

در ادامه چک می کند اگر is_hit برابر یک بود target را در data_out می ریزد ، اگر خواندن کش فعال بود از کش میخواند و در data_out می ریزد و اگر نوشتن در کش فعال بود ادرس به اضافه داده را در ان ادرس می نویسد و بعد ان را در data_out میریزد.

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

entity RAM is
    generic (
        DATA_WIDTH   : positive := 8; -- Width of data in bits
        ADDR_WIDTH   : positive := 10; -- Width of address in bits
        RAM_SIZE     : positive := 16 -- Number of words in RAM
    );
    port (
        clk     : in  STD_LOGIC;
        rst     : in  STD_LOGIC;
        read    : in  STD_LOGIC;
        write   : in  STD_LOGIC;
        address : in  STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
        data_in : in  STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
        data_out: out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
    );
end RAM;

architecture Behavioral of RAM is
    type RAMType is array (0 to RAM_SIZE-1) of STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
    signal ram_memory : RAMType;

begin

    process (clk, rst)
    begin
        if rst = '1' then
            -- Reset logic
            -- Initialize the RAM memory
            for i in RAM_MEMORY'range loop
                ram_memory(i) <= (others => '0');
            end loop;
        elsif rising_edge(clk) then
            -- RAM operation logic
            if read = '1' then
                -- Read operation
             data_out <= ram_memory(conv_integer(address));
            elsif write = '1' then
                -- Write operation
                ram_memory(conv_integer(address)) <= data_in;
            end if;
        end if;
    end process;
end Behavioral;

```

Ram component

در این کامپوننت نیز تقریبا ورودی و خروجی های مشابه کامپوننت کش تعریف شده است. این کامپوننت هم چک می کند اگر rst فعال بود همه سطر های کش (ارایه دو بعدی) را صفر می کند در غیر این صورت با لبه بالا رونده کلاک چک می کند اگر خواندن رم فعال بود ان سطر مخصوص ان ادرس را از ارایه می گیرد و در خروجی میریزد و اگر نوشتن در رم فعال بود ، ان داده ورودی را در ان ادرس مشخص شده می نویسد.

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

entity SystemTop is
    generic (
        DATA_WIDTH   : positive := 8; -- Width of data in bits
        ADDR_WIDTH   : positive := 10; -- Width of address in bits
        CACHE_SIZE   : positive := 8; -- Number of cache lines
        BLOCK_SIZE   : positive := 4;  -- Number of words in a cache line
        RAM_SIZE     : positive := 32 -- Number of words in RAM
    );
    port (
        clk    : in  STD_LOGIC;
        rst    : in  STD_LOGIC;
        ram_write_direct : in std_logic;
        cpu_address: in STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
        cpu_data_in:in STD_LOGIC_VECTOR(ADDR_WIDTH+DATA_WIDTH-1 downto 0);
        cpu_data_out: out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
    );
end SystemTop;

architecture Behavioral of SystemTop is
    signal cache_read, ram_read: STD_LOGIC;
    signal cache_write,ram_write, cache_hit: STD_LOGIC;
    signal cache_data_out, ram_data_out: STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
    signal cache_data_in: STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);


    -- Instantiate Cache and RAM modules
    component Cache
        generic (
            DATA_WIDTH   : positive := 8;
            ADDR_WIDTH   : positive := 10;
            CACHE_SIZE   : positive := 8
            );
        port (
            clk    : in  STD_LOGIC;
            rst    : in  STD_LOGIC;
            cache_read    : in  STD_LOGIC;
            cache_write   : in  STD_LOGIC;
            address : in  STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
            data_in : in  STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
            hit    : out std_logic;
            data_out: out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
        );
    end component;

    component RAM
        generic (
            DATA_WIDTH   : positive := 8;
            ADDR_WIDTH   : positive := 10;
            RAM_SIZE     : positive := 32
        );
        port (
            clk    : in  STD_LOGIC;
            rst    : in  STD_LOGIC;
            read   : in  STD_LOGIC;
            write  : in  STD_LOGIC;
            address : in  STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
            data_in : in  STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
            data_out: out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
        );
    end component;
```

System_top component

```
begin
    ram_read <= not cache_hit;
        -- Instantiate Cache and RAM modules

    cache_inst: Cache
        generic map (
            DATA_WIDTH   => DATA_WIDTH,
            ADDR_WIDTH   => ADDR_WIDTH,
            CACHE_SIZE   => CACHE_SIZE
        )
        port map (
            clk     => clk,
            rst     => rst,
            cache_read    => cache_read,
            cache_write   => cache_write,
            address => cpu_address,
            data_in => cache_data_in,
            hit     => cache_hit,
            data_out=> cache_data_out
        );

    ram_inst: RAM
        generic map (
            DATA_WIDTH   => DATA_WIDTH,
            ADDR_WIDTH   => ADDR_WIDTH,
            RAM_SIZE     => RAM_SIZE
        )
        port map (
            clk     => clk ,
            rst     => rst,
            read    => ram_read,
            write   => ram_write_direct,
            address => cpu_address,
            data_in => cpu_data_in(DATA_WIDTH-1 downto 0),
            data_out=> ram_data_out
        );

    -- CPU Interaction Logic
    process (clk, rst)
    begin
        if rst = '1' then
            -- Reset logic
            -- Additional initialization logic if needed
        elsif rising_edge(clk) then
            -- CPU Interaction Logic
            if cache_hit = '1' then
                cache_write <= '0';
                cpu_data_out <= cache_data_out;
            else
                cache_read <= '0';
                cpu_data_out <= ram_data_out;
                cache_write <= '1';
                cache_data_in <= ram_data_out;
            end if ;
        end if;
    end process;
end Behavioral;

```

System_top component Cont.

در این کامپوننت ما از دو کامپوننت قبلی استفاده خواهیم کرد و انها را تعریف میکنیم و از انها یک instanst می سازیم.

ورودی های این کامپوننت عبارتند از :

Clk

Rst

Ram_write _direct برای این است که ابتدا ما در رم اطلاعاتی بنویسیم.

Cpu_address ادرسی که سی پی یو می خواهد از ان بخواند یا بنویسد

Cpu_data_in ورودی سی پی یو

Cpu_data_out خروجی سی پی یو

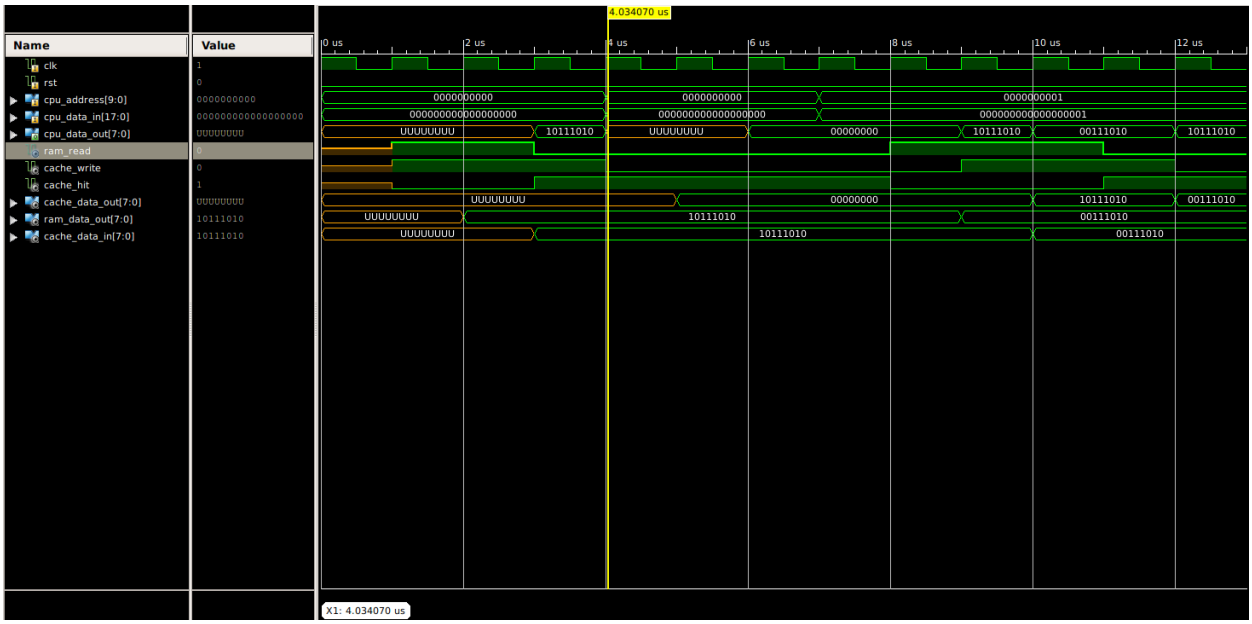
بعد از تعریف ورودی ها و تعریف کامپوننت ها و ساختن یک نمونه از انها، ارتباط بین این فرایند هارا شکل می دهیم.

ابتدا یک process حساس به clk، rst تعریف می کنیم و در ان چک می کنیم که در لبه بالارونده کلاک، اگر cache_hit (همان hit در کامپوننت cache است) برابر یک بود، cache_write را برابر صفر می کنیم یعنی نمی خواهیم در کش بنویسیم و خروجی کش را به عنوان خروجی نهایی cpu می فرستیم .

در غیر این صورت (یعنی hit برابر یک نبود) ، خواندن در کش را برابر صفر میکینم چون وقتی ان داده در کش وجود ندارد ، دیگر خواندن از کش لازم نیست و وقتی در کش وجود نداشته باشد به ram می رود و ان اطلاعات را پیدا می کند و خروجی رم را به خروجی cpu می دهد و نوشتن در کش را فعال می کند و خروجی رم را به عنوان ورودی کش میگیریم و در کش می نویسیم.

در صفحه بعد test bench را مشاهده میکنیم.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SystemTop_tb is
end SystemTop_tb;

architecture Behavioral of SystemTop_tb is
    -- Signals to connect to SystemTop
    signal clk : STD_LOGIC := '0';
    signal rst : STD_LOGIC;
    signal cpu_address: STD_LOGIC_VECTOR(9 downto 0);
    signal cpu_data_in: STD_LOGIC_VECTOR(17 downto 0);
    signal cpu_data_out: STD_LOGIC_VECTOR(7 downto 0);
    signal ram_read: STD_LOGIC;
    signal cache_write, cache_hit: STD_LOGIC;
    signal cache_data_out, ram_data_out: STD_LOGIC_VECTOR(7 downto 0);
    signal cache_data_in: STD_LOGIC_VECTOR(7 downto 0);
begin
    -- Instantiate the SystemTop component
    uut: entity work.SystemTop
        port map (
            clk => clk,
            rst => rst,
            ram_write_direct => ram_write_direct,
            cpu_address => cpu_address,
            cpu_data_in => cpu_data_in,
            cpu_data_out => cpu_data_out
        );
    -- Clock process
    clk_process : process
    begin
        clk <= '0';
        wait for 10 ns;
        clk <= '1';
        wait for 10 ns;
    end process;
    -- Stimulus process
    stimulus_process : process
    begin

        -- Scenario: Cache Hit
        cpu_address <= (others => '0');
        cpu_data_in <= (others => '0');
        wait for 20 ns; -- Wait for a response
        -- Check for RAM read and data_out

    -- Scenario: Cache Hit
        cpu_address <= (others => '0'); -- An address that is cached
        cpu_data_in <= (others => '0');
        wait for 20 ns; -- Wait for a response
        -- Check for cache_hit signal and data_out

        -- Scenario: Cache Miss
        cpu_address <= "0000000001"; -- An address that is not cached
        cpu_address <= "00000000000000001"; -- An address that is not cached
        wait for 20 ns; -- Wait for a response
        -- Check for RAM read and data_out

        wait;
    end process;
end Behavioral;


```



Simulation

در قسمت اول

0000000000 = cpu_address

000000000000000000 = cpu_data_in

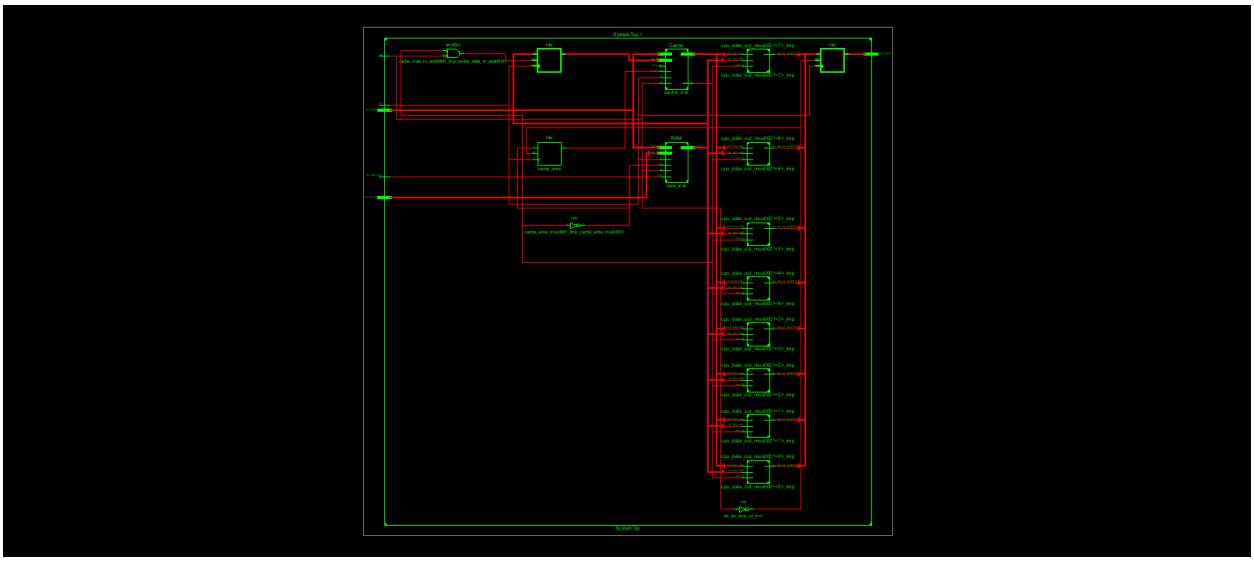
که چون در cache موجود نیست سیگنال‌های ram_read و cache_write فعال شده در کلاک بعدی دیتا به ram_data_out منتقل می‌شود، در کلاک بعدی cache_hit یک می‌شود و دیتا به cpu_data_out منتقل می‌شود.

دفعه بعدی که همان ادرس داده می‌شود، cache_hit یک باقی می‌ماند، دیتا به cache_data_out منتقل می‌شود و بعد به cpu_data_out.

اما اگر باز ادرس دیگری بخواهیم همان‌طور که در تصویر مشاهده می‌شود، دوباره ram_read و cache_write فعال شده و دیتا از ram به cache منتقل می‌شود و cache_hit صورت می‌گیرد.

و در اخر کد ما قابل سنتز است:

Synthesize



پایان