Architektura systemów komputerowych

Temat: Pamięć cache (128 bloków x 4 bajty)

Wykonał:
Marcin Brach

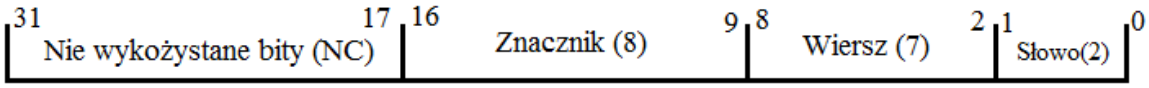
1. Cel projektu

Celem projektu jest zaprojektowanie pamięci cache o organizacji 128 bloków x 4 bajty, oraz odwzorowaniu bezpośrednim z zapisem opóźnionym WriteBack. Projekt należy wykonać w środowisku Quartus II.

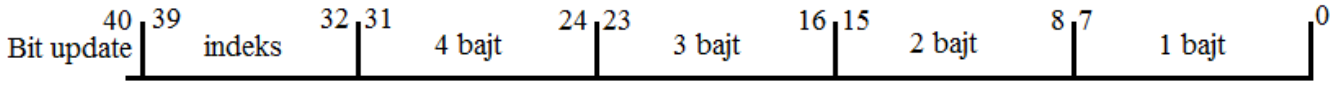
2. Sposób wykonania i organizacja pamięci cache oraz magistrali adresowej.

System pamięci podręcznej jest reprezentowany za pomocą adresów 17 bitowych. Siedmio bitowy numer wiersza jest wykorzystywany jako indeks dostępu do wiersza w pamięci podręcznej. Jeżeli 8-bitowy znacznik jest równy znacznikowi który jest przechowywany w tym wierszu to 2-bitowy numer słowa wybiera określony bajt do zapisu bądź odczytu.

adres pamięci cache - ad:

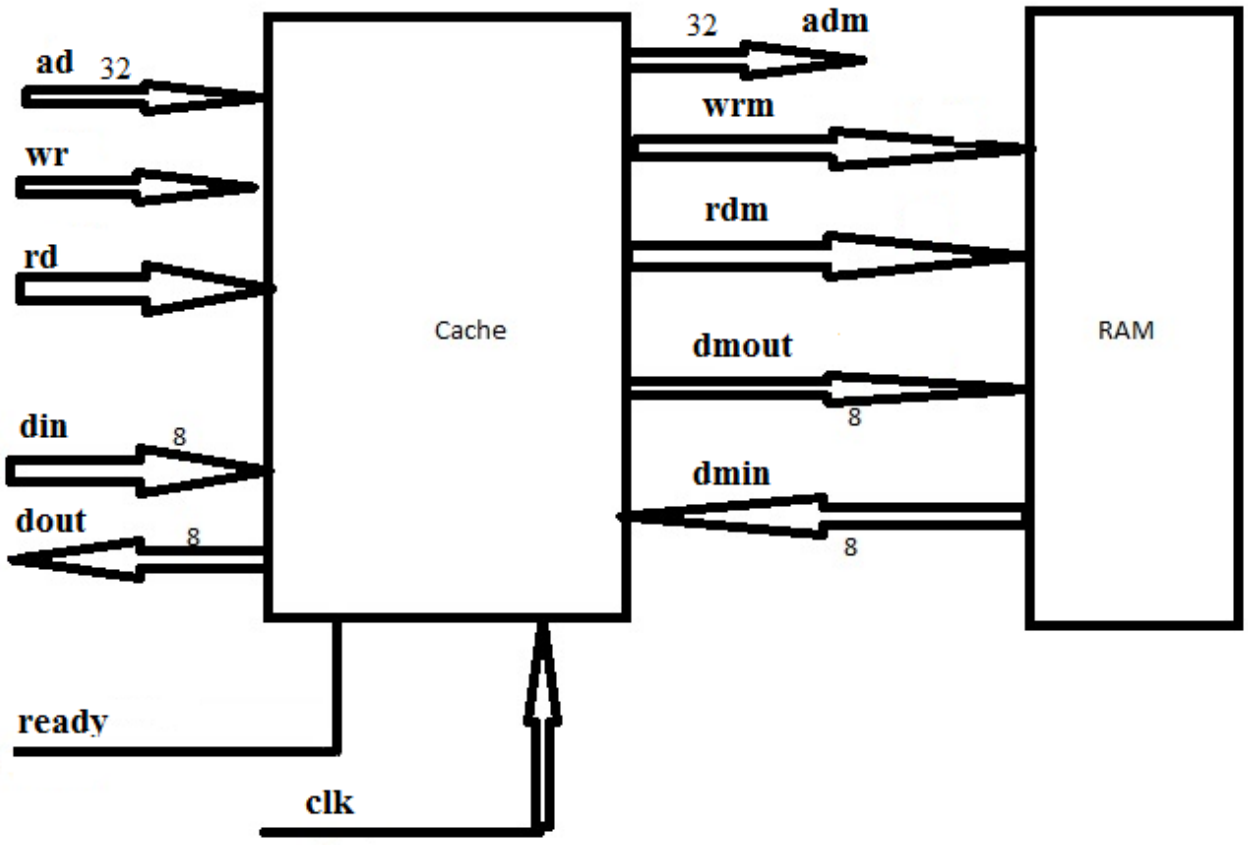


organizacja wartości przechowywanych w pamięci cache:



W projektowanej pamięci cache użyta jest pamięć wygenerowana za pomocą kreatora Mega Wizard Plugin-In Menager która znajduje się w pliku memory.vhd. Natomiast obsługa pamięci cache została zaimplementowana w pliku cache.vhd.

3. Schematu połączeń projektowanego modułu:



4. Implemantacja w języku vhdl

a) Plik memory.vhd:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;


ENTITY memory IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		data		: IN STD_LOGIC_VECTOR (40 DOWNTO 0);
		wren		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (40 DOWNTO 0)
	);
END memory;


ARCHITECTURE SYN OF memory IS

	SIGNAL sub_wire0	: STD_LOGIC_VECTOR (40 DOWNTO 0);


	COMPONENT altsyncram
	GENERIC (
		clock_enable_input_a		: STRING;
		clock_enable_output_a		: STRING;
		intended_device_family		: STRING;
		lpm_hint		: STRING;
		lpm_type		: STRING;
		numwords_a		: NATURAL;
		operation_mode		: STRING;
		outdata_aclr_a		: STRING;
		outdata_reg_a		: STRING;
		power_up_uninitialized		: STRING;
		ram_block_type		: STRING;
		widthad_a		: NATURAL;
		width_a		: NATURAL;
		width_byteena_a		: NATURAL
	);


	PORT (
			address_a	: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
			clock0	: IN STD_LOGIC ;
			data_a	: IN STD_LOGIC_VECTOR (40 DOWNTO 0);
			wren_a	: IN STD_LOGIC ;
			q_a	: OUT STD_LOGIC_VECTOR (40 DOWNTO 0)
	);
	END COMPONENT;
```

```
BEGIN
      q    <= sub_wire0(40 DOWNTO 0);

      altsyncram_component : altsyncram
      GENERIC MAP (
            clock_enable_input_a => "BYPASS",
            clock_enable_output_a => "BYPASS",
            intended_device_family => "Cyclone II",
            lpm_hint => "ENABLE_RUNTIME_MOD=NO",
            lpm_type => "altsyncram",
            numwords_a => 128,
            operation_mode => "SINGLE_PORT",
            outdata_aclr_a => "NONE",
            outdata_reg_a => "UNREGISTERED",
            power_up_uninitialized => "FALSE",
            ram_block_type => "M4K",
            widthad_a => 7,
            width_a => 41,
            width_byteena_a => 1
      )
      PORT MAP (
            address_a => address,
            clock0 => clock,
            data_a => data,
            wren_a => wren,
            q_a => sub_wire0
      );

END SYN;
```

b) Plik cache.vhd:

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

entity cache is
port(       ad                : in std_logic_vector(31 downto 0);
            din               : in std_logic_vector(7 downto 0);
            dout        : out std_logic_vector(7 downto 0);
            clk         : in std_logic;
            wr          : in std_logic;
            rd          : in std_logic;
            ready       : in std_logic;


            adm               : out std_logic_vector(31 downto 0);
            dmin        : in std_logic_vector(7 downto 0);
            dmout       : out std_logic_vector(7 downto 0);
            wrm         : out std_logic;
            rdm         : out std_logic
            );
end cache;
```

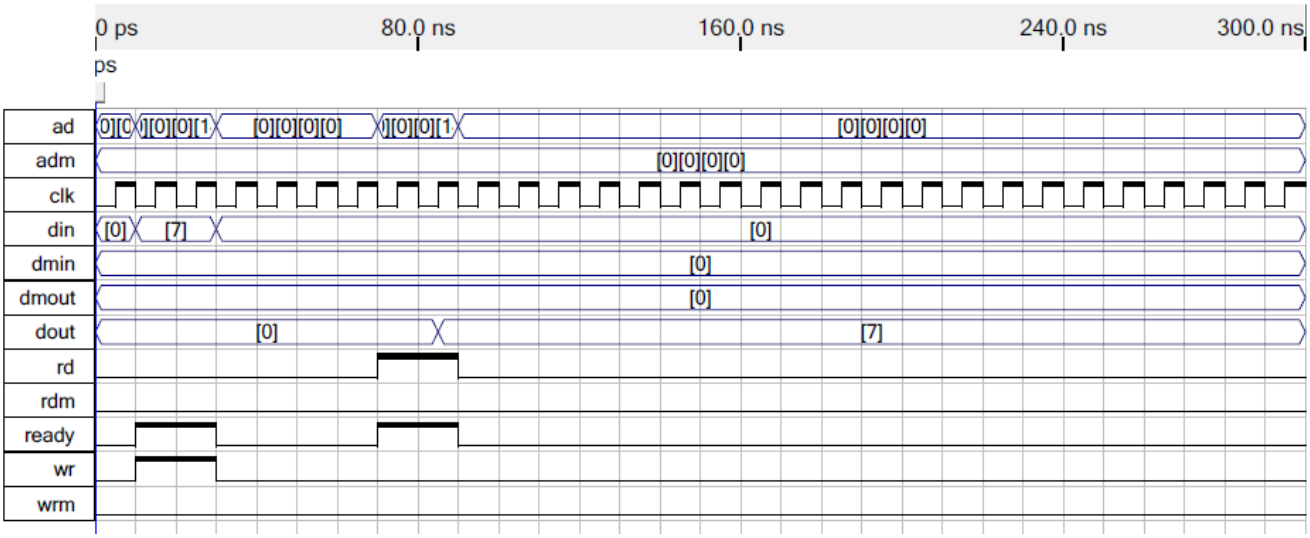
```
architecture RTL of cache is

signal adresy: std_logic_vector(6 downto 0);
signal wejscie: std_logic_vector(40 downto 0);
signal zo: std_logic;
signal wyjscie: std_logic_vector(40 downto 0);
signal t: std_logic_vector(40 downto 0);
signal s1: std_logic;

--uzycie komponentu memory (pamiec wygenerowana została w quartusie(osobny plik
memory.vhd))
component memory is
port
(
            address     : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
            clock       : IN STD_LOGIC  := '1';
            data        : IN STD_LOGIC_VECTOR (40 DOWNTO 0);
            wren        : IN STD_LOGIC ;
            q              : OUT STD_LOGIC_VECTOR (40 DOWNTO 0)
      );
end component;
begin
--mapowanie komponentu
P1: memory port map (
      address=>adresy,
      wren=>zo,
      clock=>clk,
      data=>wejscie,
      q=>wyjscie
      );

process(clk,wr,rd,zo)
variable temp : std_logic_vector(40 downto 0);
variable x: integer range 0 to 1;
begin
adresy(6 downto 0)<=ad(8 downto 2);
wejscie(39 downto 32) <= ad(16 downto 9);
if(clk'event and clk='1') then
            if (x=1) then
                  x:=0;
                  s1<='1';
            else

                  x:=x+1;
                  s1<='0';
            end if;

if(rd='1' or wr='1') then
            zo<='1';
            temp(40 downto 0):= wyjscie(40 downto 0);
            adresy(6 downto 0)<=ad(8 downto 2);
            wejscie(39 downto 32) <= ad(16 downto 9);
end if;
end if;
t<=temp;
end process;
```

```
process(clk,wr,rd)
begin
if(clk'event and clk='1') then
if(s1='1') then

      if(ready='1') then

            if(rd='1') then
            --odczyt z pamieci cache, gdy wartosc nie znajduje sie w cache-u pobranie jej
z pamieci RAM(else)
                  if(ad(16 downto 9)=t(39 downto 32)) then
                        case ad(1 downto 0) is
                        when "00" => dout(7 downto 0)<=wyjscie(7 downto 0);   --1 bajt
                        when "01" => dout(7 downto 0)<=wyjscie(15 downto 8);  --2 bajt
                        when "10" => dout(7 downto 0)<=wyjscie(23 downto 16); --3 bajt
                        when "11" => dout(7 downto 0)<=wyjscie(31 downto 24); --4 bajt
                        end case;
                  else
                        rdm<=rd;
                        adm(14 downto 0) <= ad(16 downto 2);
                        case ad(1 downto 0) is
                        when "00" => wejscie(7 downto 0) <= dmin(7 downto 0);   --1 bajt

                        when "01" => wejscie(15 downto 8) <= dmin(7 downto 0);  --2 bajt
                        when "10" => wejscie(23 downto 16) <= dmin(7 downto 0); --3 bajt
                        when "11" => wejscie(31 downto 24) <= dmin(7 downto 0); --4 bajt
                        end case;
                  end if;
            end if;

            if(wr='1') then
            --zapis do pamieci cache,
                  if(ad(16 downto 9)=t(39 downto 32)) then
                        case ad(1 downto 0) is
                        when "00" => wejscie(7 downto 0)<=din(7 downto 0);   --1 bajt
                        when "01" => wejscie(15 downto 8)<=din(7 downto 0);  --2 bajt
                        when "10" => wejscie(23 downto 16)<=din(7 downto 0); --3 bajt
                        when "11" => wejscie(31 downto 24)<=din(7 downto 0); --4 bajt
                        end case;
                        wejscie(40 downto 40) <= "1"; -- ustawienie bitu update
                  end if;

                   --zapis do zewnetrznej pamieci RAM
                  if(wyjscie(40 downto 40) = "1") then
                        wrm<=wr;
                        adm(14 downto 0) <= ad(16 downto 2);
                        case ad(1 downto 0) is
                        when "00" => dmout(7 downto 0) <= wyjscie(7 downto 0);   --1 bajt

                        when "01" => dmout(7 downto 0) <= wyjscie(15 downto 8);  --2 bajt
                        when "10" => dmout(7 downto 0) <= wyjscie(23 downto 16); --3 bajt
                        when "11" => dmout(7 downto 0) <= wyjscie(31 downto 24); --4 bajt
                        end case;
                  end if;
            end if;
```

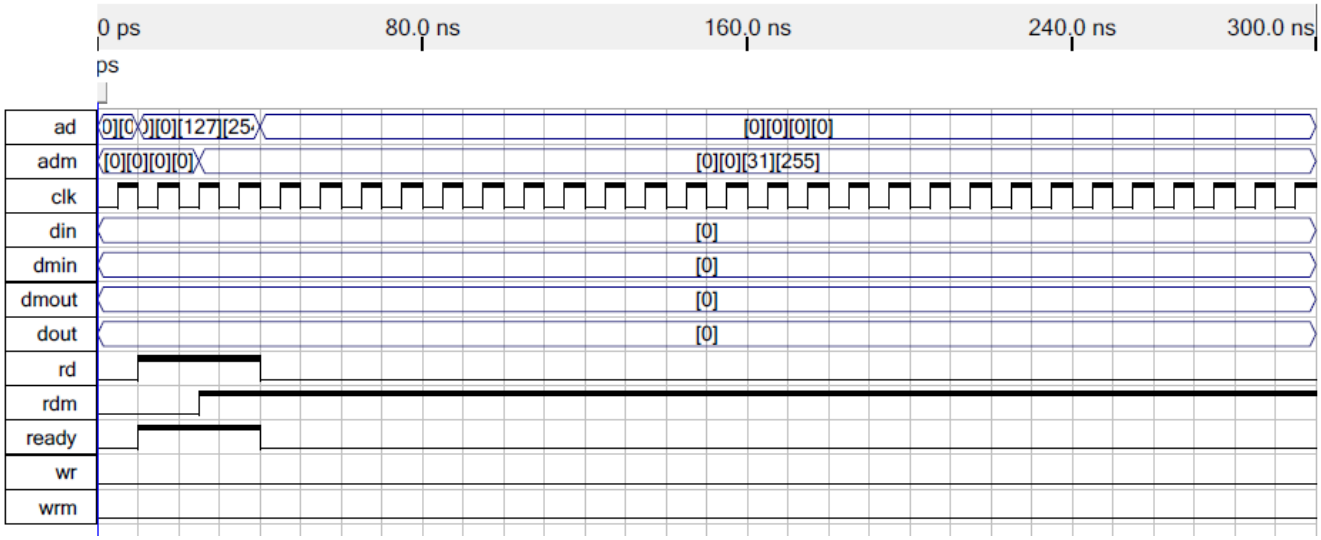
```
      end if;
end if;
end if;
end process;
end RTL;
```

5. Wyniki symulacji

a) Zapis do pamięci cache oraz odczyt tej wartości:



b) Pobranie wartości z pamięci RAM gdy nie znajduje się ona w pamięci cache:



c) Zapis do pamięci RAM (ustawiony bit update):