1. Problem/Objective

We will be looking at simulating the direct cache mapping algorithm in VHDL. We will be using a finite state machine (FSM) in order to implement the different states during the algorithm. This will allow us to implement the design and then test it using a test bench and produce an output waveform that should allow us to visualize the different steps in the cache mapping.

2. Methodology

https://pdfs.semanticscholar.org/9c75/07d5d4dafb01e8faa687cf10598768765e47.pdf

This paper is about the design of cache memory mapping techniques for low power processors. It starts off abstracting what are the uses of cache memory. This goes to explain that the main purpose of cache is to give faster memory access to the data that is being read. This is further explained by discussing that the idea behind cache is to provide a less expensive type of memory and which types of semiconductor memories there are and how large they should be.

Into further detail, the paper explains that cache systems are on-chip memory elements that data can be stored. The idea behind this is that by finding an acceptable hit-miss ratio you can successfully lower read times by not having to make main memory searches every time you need to access data. Then the author goes on to touch on the different types of mapping techniques, specifically direct mapping, fully associative mapping, and set associative mapping. Finally, the paper touches on the different levels of cache memory and the purpose for each. It explains how the hierarchical levels help. This is explained by stating that it is able to reduce the miss penalty by having tiered caches. But the sacrifice to this is that the penalties for tiered caches missing is dependent on how many cache tiers there are and that each one compounds the delay for a miss by the previous tier.

3. Question(s)

Please, show your work. Don't just give a single value. Explain how you obtained that number. HINT: see the Lecture 6.

a) What is the address length?

The main memory address length is comprised of the tag, line, and word. With each of the tag and line being 4 bits in length while the word is 2 bits, the address length comes out to 10 bits.

b) What is the number of addressable units?

The number of addressable units is 2^{s+w} words, which is 2^{10} = 1024 units.

c) What is the block size?

The block size is equal to the line size which is 2^{w} words, this comes out to 2^{2} = 4 words.

d) How many blocks are in main memory?

Blocks in main memory is $\frac{2^{s+w}}{2^{w}} = 2^{s}$, which comes out to 2^{8} = 256 blocks.

e) How many lines of cache are there?

Cache lines is 2^{r}, which comes out to 2^{4} = 16 lines.

f) What is the size of the cache?

The size of cache is 2^{r+w}, which comes out to $2^6 = 64$ bits.

4. Program Code

Definitions

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

package definitions is
	-- Cache
	constant cache_tag_width : positive := 4;		-- 4-bit Tag
	constant cache_line_width : positive := 4;		-- 4-bit Line
	constant cache_word_width : positive := 2;		-- 2-bit Word
	constant cache_data_width : positive := (cache_tag_width + 2**5);		-- Data width
(tag + 32-bit data)
	constant cache_len : positive := 2**cache_line_width;			-- 2^line cache

	-- Main Memory
	constant main_mem_adr_width : positive := cache_tag_width + cache_line_width;
	constant main_mem_data_width : positive := 2**5;	-- 32-bit data
	constant main_mem_len : positive := 2**(cache_tag_width + cache_line_width);
end definitions;

package body definitions is


end definitions;
```

Cache

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.definitions.all;

entity cache is
	generic (
		LINE_WIDTH : integer := cache_line_width;
		DATA_WIDTH : integer := cache_data_width);

	port (
			clk, rst : in std_logic;
			we : in std_logic;
			line : in std_logic_vector(LINE_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 cache;

architecture behavioral of cache is
        type ram_type is array (2**LINE_WIDTH-1 downto 0) of std_logic_vector(DATA_WIDTH-
1 downto 0);
        signal cache : ram_type;
        signal line_reg : std_logic_vector (LINE_WIDTH-1 downto 0);

begin
        process(clk, rst)
        begin
                if (rst = '1') then
                        -- Clear
                        for i1 in 0 to 2**LINE_WIDTH-1 loop
                                cache(i1) <= (others => '0');
                        end loop;
                elsif(rising_edge(clk)) then
                        if (we = '1') then
                                cache(conv_integer(line)) <= data_in;
                        end if;
                        line_reg <= line;
                end if;
        end process;
        data_out <= cache(conv_integer(line_reg));
end behavioral;
```

Main Memory

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.definitions.all;

entity main_mem is
        generic (
                ADDR_WIDTH : integer := main_mem_adr_width;
                DATA_WIDTH : integer := main_mem_data_width);

        port (
                        clk, rst : in std_logic;
                        we : in std_logic;
                        addr : 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 main_mem;

architecture behavioral of main_mem is
        type ram_type is array (2**ADDR_WIDTH-1 downto 0) of std_logic_vector(DATA_WIDTH-
1 downto 0);
        signal ram : ram_type;
        signal addr_reg : std_logic_vector (ADDR_WIDTH-1 downto 0);
begin
        process(clk, rst)
        begin
                if (rst = '1') then
                        -- Stores the address in the location
                        for i1 in 0 to 2**ADDR_WIDTH-1 loop
                                ram(i1) <= conv_std_logic_vector(i1,main_mem_data_width);
                        end loop;
                elsif(rising_edge(clk)) then
                        if (we = '1') then
                                ram(conv_integer(addr)) <= data_in;
                        end if;
                        addr_reg <= addr;
                end if;
        end process;
        data_out <= ram(conv_integer(addr_reg));
end behavioral;
```

Direct Cache

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.definitions.all;

entity direct_cache is
        port (
                        clk : in std_logic;
                        rst : in std_logic;
                        addr : in std_logic_vector((cache_tag_width + cache_line_width +
cache_word_width)-1 downto 0);
                        hit : out std_logic;

                        cache_we : out std_logic;
                        cache_data_in : out std_logic_vector(cache_data_width-1 downto 0);
                        cache_line : out std_logic_vector(cache_line_width-1 downto 0);
                        cache_tag : out std_logic_vector(cache_tag_width-1 downto 0);
                        cache_data_out : out std_logic_vector(cache_data_width-1 downto 0);

                        main_mem_we : out std_logic;
                        main_mem_addr    :    out    std_logic_vector(main_mem_adr_width-1
downto 0);
                        main_mem_data_in,          main_mem_data_out          :          out
std_logic_vector(main_mem_data_width-1 downto 0);

                        state : out integer);
end direct_cache;

architecture structural of direct_cache is
        component direct_cache_ctrl is
                port (
                                clk, rst : in std_logic;
                                addr : in std_logic_vector((cache_tag_width + cache_line_width
+ cache_word_width)-1 downto 0);
                                hit : out std_logic;

                                cache_data_in : in std_logic_vector(cache_data_width-1 downto
0);
                                cache_we : out std_logic;
                                cache_line : out std_logic_vector(cache_line_width-1 downto 0);
                                cache_tag : out std_logic_vector(cache_tag_width-1 downto 0);
                                cache_data_out   :   out   std_logic_vector(cache_data_width-1
downto 0);

                                main_mem_data_in                    :                    in
std_logic_vector(main_mem_data_width-1 downto 0);
                                main_mem_addr : out std_logic_vector(main_mem_adr_width-
1 downto 0);
                                main_mem_we : out std_logic;

                                state : out integer);
        end component;

        component main_mem is
                port (
                                clk, rst : in std_logic;
                                we : in std_logic;
                                addr : in std_logic_vector(main_mem_adr_width-1 downto 0);
                                data_in : in std_logic_vector(main_mem_data_width-1 downto
0);
                                data_out   :   out   std_logic_vector(main_mem_data_width-1
downto 0));
        end component;

        component cache is
                port (
                                clk, rst : in std_logic;
                                we : in std_logic;
                                line : in std_logic_vector(cache_line_width-1 downto 0);
                                data_in : in std_logic_vector(cache_data_width-1 downto 0);
                                data_out : out std_logic_vector(cache_data_width-1 downto 0));
        end component;

        --TODO: Add any signal here
        signal s_cache_we : std_logic;
        signal s_cache_line : std_logic_vector(cache_line_width-1 downto 0);
        signal s_cache_data_in : std_logic_vector(cache_data_width-1 downto 0);
        signal s_cache_data_out : std_logic_vector(cache_data_width-1 downto 0);
        signal s_main_mem_we : std_logic;
        signal s_main_mem_addr : std_logic_vector(main_mem_adr_width-1 downto 0);
        signal s_main_mem_data_out : std_logic_vector(main_mem_data_width-1 downto 0);
        signal s_main_mem_data_in : std_logic_vector(main_mem_data_width-1 downto 0);

begin
        Map_Main: main_mem port map (
                clk => clk,
                rst => rst,
                we => s_main_mem_we,
                addr => s_main_mem_addr,
                data_in => s_main_mem_data_in,
                data_out => s_main_mem_data_out);

        Map_Cache: cache port map (
                clk => clk,
                rst => rst,
                we => s_cache_we,
                line => s_cache_line,
                data_in => s_cache_data_in,
                data_out => s_cache_data_out);

        Map_Ctrl: direct_cache_ctrl port map (
                clk => clk,
                rst => rst,
                hit => hit,
                addr => addr,
                cache_data_in => s_cache_data_out,
                cache_we => s_cache_we,
                cache_line => s_cache_line,
                cache_tag => cache_tag,
                cache_data_out => s_cache_data_in,
                main_mem_data_in => s_main_mem_data_out,
                main_mem_addr => s_main_mem_addr,
                main_mem_we => s_main_mem_we,
                state => state);

        --Outputs
        cache_data_in <= s_cache_data_out;
        cache_line <= s_cache_line;
        main_mem_addr <= s_main_mem_addr;
        cache_we <= s_cache_we;
        main_mem_we <= s_main_mem_we;
        cache_data_out <= s_cache_data_out;
        main_mem_data_out <= s_main_mem_data_out;
        main_mem_data_in <= s_main_mem_data_in;

end structural;
```

Direct Cache Control

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.definitions.all;

entity direct_cache_ctrl is
        port (
                        clk, rst : in std_logic;
                        addr : in std_logic_vector((cache_tag_width + cache_line_width +
cache_word_width)-1 downto 0);
                        hit : out std_logic;

                        cache_data_in : in std_logic_vector(cache_data_width-1 downto 0);
                        cache_we : out std_logic;
                        cache_line : out std_logic_vector(cache_line_width-1 downto 0);
                        cache_tag : out std_logic_vector(cache_tag_width-1 downto 0);
                        cache_data_out : out std_logic_vector(cache_data_width-1 downto 0);

                        main_mem_data_in : in std_logic_vector(main_mem_data_width-1
downto 0);
                        main_mem_addr : out std_logic_vector(main_mem_adr_width-1
downto 0);
                        main_mem_we : out std_logic;

                        state : out integer);
end direct_cache_ctrl;

architecture behavioral of direct_cache_ctrl is
        --Signals for FSM States
        type eg_state_type is (CACHE_R, GET_TAG, GET_TAG2, COMP_TAGS, MAIN_MEM_R,
CACHE_UPDATE, CACHE_UPDATE2);
        signal state_reg, state_next: eg_state_type;

--****************************************
```

```
--*       Don't delete/change the following signals
--*****************************************
          signal tag : std_logic_vector(cache_tag_width-1 downto 0) := (others => '0');
          signal line : std_logic_vector(cache_line_width-1 downto 0) := (others => '0');
          signal word : std_logic_vector(cache_word_width-1 downto 0) := (others => '0');
          signal cache_tag_sig : std_logic_vector(cache_tag_width-1 downto 0) := (others => '0');
          signal addr_len : integer := cache_tag_width + cache_line_width + cache_word_width;

begin
--*****************************************
--*       Don't delete/change the following
--*       Seperate tag, line, and word concurrently
--*****************************************
          tag <= addr(addr_len-1 downto (addr_len-cache_tag_width));

          line    <=    addr((addr_len-cache_tag_width)-1    downto    (addr_len-cache_tag_width-
cache_line_width));

          word <= addr((addr_len-cache_tag_width-cache_line_width)-1 downto 0);
          cache_line <= line;

          cache_tag <= tag;


          --TODO: State register
          process(clk, rst)
          begin
                    --Reset states when rst is 1
                    if (rst = '1') then
                              state_reg <= CACHE_R;
                    --Update state on rising edge
                    elsif (rising_edge(clk)) then
                              state_reg <= state_next;
                    end if;
          end process;

          --TODO: Next-state
          process(state_reg)
          begin
                    --Set next state based on current state
                    case state_reg is
                              when CACHE_R =>
                                        state_next <= GET_TAG;
                              when GET_TAG =>
                                        state_next <= GET_TAG2;
                              when GET_TAG2 =>
                                        state_next <= COMP_TAGS;
                              when COMP_TAGS =>
                    if (cache_tag_sig = tag) then
                        state_next <= CACHE_R;
                    else
                        state_next <= MAIN_MEM_R;
                    end if;
                when MAIN_MEM_R =>
                    state_next <= CACHE_UPDATE;
                when CACHE_UPDATE =>
                    state_next <= CACHE_UPDATE2;
                when CACHE_UPDATE2 =>
                    state_next <= CACHE_R;
        end case;
end process;

--TODO: Moore logic
process(state_reg)
begin
        case state_reg is
                when CACHE_R =>
                    cache_we <= '0';
                    state <= 1;
                when GET_TAG =>
                    state <= 2;
                when GET_TAG2 =>
                    cache_tag_sig <= cache_data_in(35 downto 32);
                    state <= 3;
                when COMP_TAGS =>
                    state <= 4;
                when MAIN_MEM_R =>
                    main_mem_addr <= (tag & line);
                    state <= 5;
                when CACHE_UPDATE =>
                    cache_we <= '1';
                    state <= 6;
                when CACHE_UPDATE2 =>
                    cache_data_out <= (tag & main_mem_data_in);
                    state <= 7;
        end case;
end process;

--TODO: Mealy logic
process(state_reg, cache_tag_sig, tag)
begin
        case state_reg is
                when COMP_TAGS =>
                    if (cache_tag_sig = tag) then
                        hit <= '1';
                    else
```

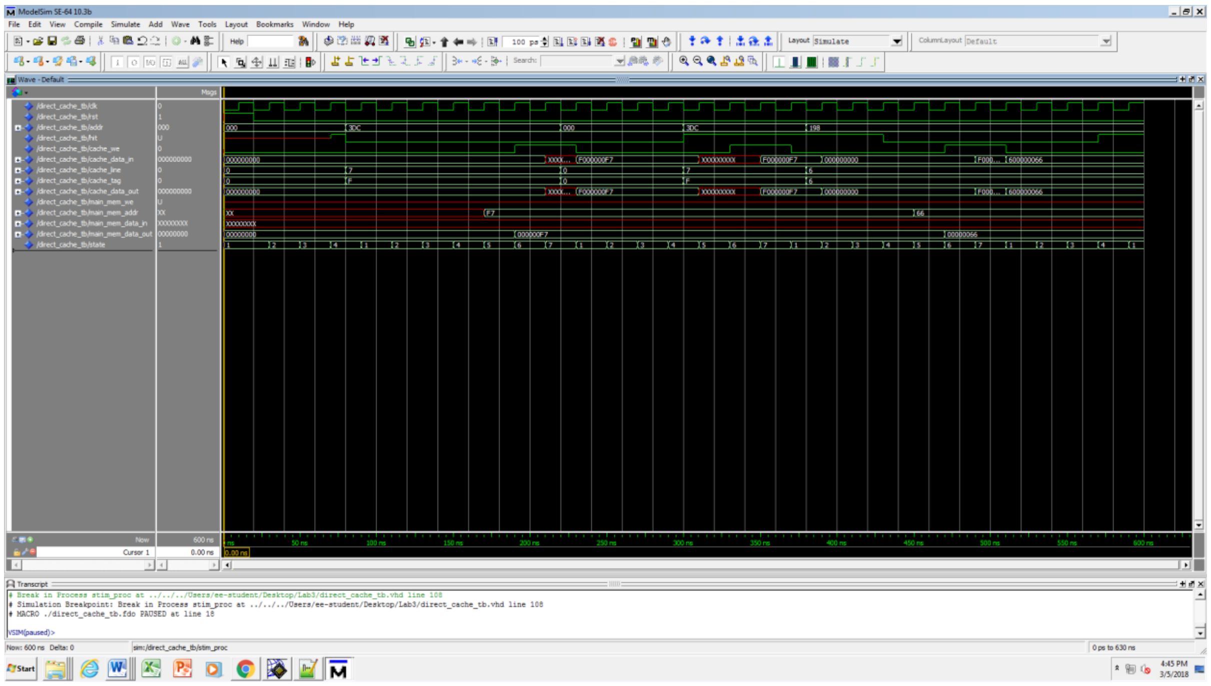
```
                                        hit <= '0';
                                end if;
                        when others =>
                                NULL;
                end case;
        end process;

end behavioral;
```

5. Test Bench

Direct Cache Test Bench

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.definitions.all;

entity direct_cache_tb is
end direct_cache_tb;

architecture behavior of direct_cache_tb is

  -- component declaration for the unit under test (uut)

  component direct_cache
  port(
     clk : in  std_logic;
     rst : in  std_logic;
     addr : in  std_logic_vector(9 downto 0);
     hit : out  std_logic;
     cache_we : out  std_logic;
     cache_data_in : out  std_logic_vector(35 downto 0);
     cache_line : out  std_logic_vector(3 downto 0);
     cache_tag : out  std_logic_vector(3 downto 0);
     cache_data_out : out  std_logic_vector(35 downto 0);
     main_mem_we : out  std_logic;
     main_mem_addr : out  std_logic_vector(7 downto 0);
     main_mem_data_in : out  std_logic_vector(31 downto 0);
     main_mem_data_out : out  std_logic_vector(31 downto 0);
                    state : out integer
    );
  end component;


  --inputs
  signal clk : std_logic := '0';
  signal rst : std_logic := '0';
  signal addr : std_logic_vector(9 downto 0) := (others => '0');

     --outputs
  signal hit : std_logic;
  signal cache_we : std_logic;
  signal cache_data_in : std_logic_vector(35 downto 0);
  signal cache_line : std_logic_vector(3 downto 0);
  signal cache_tag : std_logic_vector(3 downto 0);
  signal cache_data_out : std_logic_vector(35 downto 0);
  signal main_mem_we : std_logic;
  signal main_mem_addr : std_logic_vector(7 downto 0);
  signal main_mem_data_in : std_logic_vector(31 downto 0);
  signal main_mem_data_out : std_logic_vector(31 downto 0);
     signal state : integer;

  -- clock period definitions
  constant clk_period : time := 20 ns;

begin

     -- instantiate the unit under test (uut)
  uut: direct_cache port map (
       clk => clk,
       rst => rst,
       addr => addr,
       hit => hit,
       cache_we => cache_we,
       cache_data_in => cache_data_in,
       cache_line => cache_line,
       cache_tag => cache_tag,
       cache_data_out => cache_data_out,
       main_mem_we => main_mem_we,
       main_mem_addr => main_mem_addr,
       main_mem_data_in => main_mem_data_in,
       main_mem_data_out => main_mem_data_out,
                       state => state
     );

  -- clock process definitions
  clk_process :process
  begin
             clk <= '0';
             wait for clk_period/2;
             clk <= '1';
             wait for clk_period/2;
  end process;
```

```
    -- stimulus process
    stim_proc: process
            variable addr_len : integer := cache_tag_width + cache_line_width + cache_word_width;
    begin
                    --TODO: Complete the test bench
                    --Reset for 1 clk cycle
                    rst <= '1';
                    wait for clk_period;
                    --Clear reset and let clock run
                    rst <= '0';
                    addr <= "0000000000";
                    wait for 3*clk_period; --wait 3 cycles after reset
                    addr <= "1111011100";
                    wait for 7*clk_period; --wait 7 cycles for miss
                    addr <= "0000000000";
                    wait for 4*clk_period; --wait 4 cycles for hit
                    addr <= "1111011100";
                    wait for 4*clk_period;
                    addr <= "0110011000";
                    wait for 7*clk_period;
                    addr <= "0110011000";
                    wait for 4*clk_period;
            -- End
            assert false report "End of Simulation" severity failure;
    end process;

end;
```



```
/direct_cache_tb/clk              0
/direct_cache_tb/rst              1
/direct_cache_tb/addr             000
/direct_cache_tb/hit              U
/direct_cache_tb/cache_we         0
/direct_cache_tb/cache_data_in    000000000
/direct_cache_tb/cache_line       0
/direct_cache_tb/cache_tag        0
/direct_cache_tb/cache_data_out   000000000
/direct_cache_tb/main_mem_we      U
/direct_cache_tb/main_mem_addr    XX
/direct_cache_tb/main_mem_data_in XXXXXXXX
/direct_cache_tb/main_mem_data_out 00000000
/direct_cache_tb/state            1
```

```
# Break in Process stim_proc at ../../../Users/ee-student/Desktop/Lab3/direct_cache_tb.vhd line 108
# Simulation Breakpoint: Break in Process stim_proc at ../../../Users/ee-student/Desktop/Lab3/direct_cache_tb.vhd line 108
# MACRO ./direct_cache_tb.fdo PAUSED at line 18

VSIM(paused)>
```

Now: 600 ns Delta: 0   sim:/direct_cache_tb/stim_proc