ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Кваліфікаційна робота

Моделювання цифрових пристроїв з використанням мови VHDL

Виконав:
студент гр. КІУКІ-21-1
Шаповалов Д.А.

Керівник:
ст. викл. Дяченко В.О.

2

Мета роботи та завдання

Метою роботи є комплексний аналіз методології моделювання складних цифрових пристроїв із використанням мови опису апаратури VHDL, а також практична реалізація та симуляція вибраних цифрових схем.

Завдання:

- порівняльний аналіз існуючих моделей опису елементів пам'яті з використанням мови VHDL;
- аналіз існуючих алгоритмів моделювання елементів з використанням VHDL;
- побудова VHDL-моделей пам'яті у середовищі Active-HDL.

Аналіз основних мов опису апаратури

3

Критерій оцінки	VHDL	Verilog	SystemVerilog	Альтернативи
Складність освоєння	Висока	Помірна	Дуже висока	Варіюється
Рівень типізації	Строга	Слабка	Покращена	Залежить від базової мови
Продуктивність розробки	Помірна	Висока	Висока для складних систем	Потенційно висока
Можливості верифікації	Базові	Базові	Розширені	Обмежені
Промислове поширення	Широке	Домінуюче	Зростаюче	Нішеве
Підтримка інструментів	Універсальна	Найширша	Розвивається	Обмежена
Ресурсні вимоги	Помірні	Низькі	Високі	Варіюються

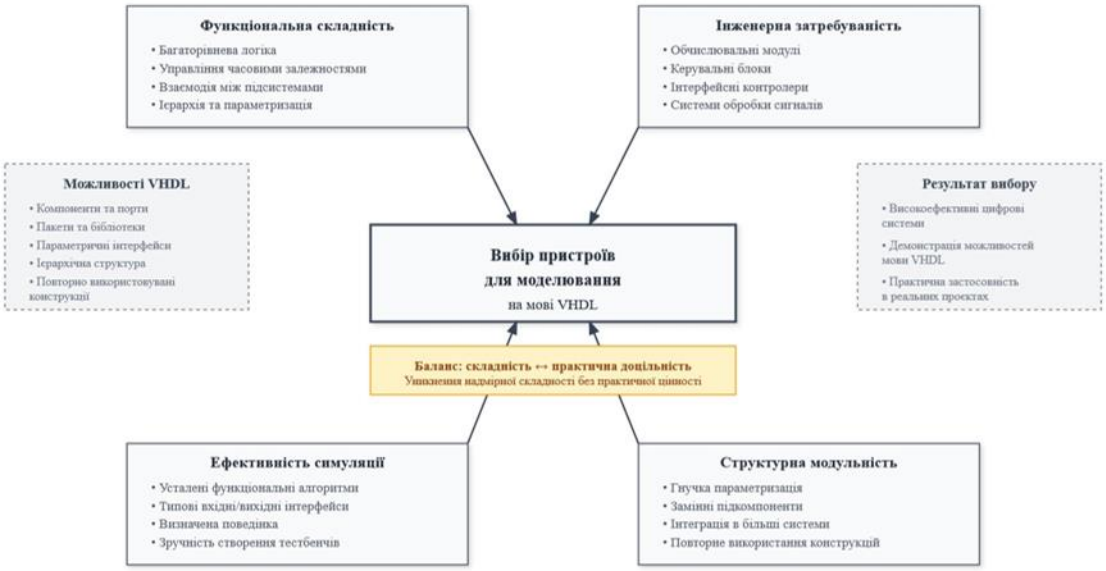
Порівняльний аналіз функціональних можливостей середовищ розробки та симуляції

4

Критерій	Active-HDL	ModelSim	Vivado	Quartus Prime	GHDL
Симуляція					
Швидкість симуляції	Висока	Дуже висока	Висока	Висока	Помірна
Мультимовність	VHDL/Verilog	VHDL/Verilog/SV	VHDL/Verilog	VHDL/Verilog	Тільки VHDL
Візуалізація	Відмінна	Відмінна	Добра	Добра	Відсутня
Верифікація					
Налагодження	Розширені засоби	Професійні	Інтегровані	Інтегровані	Базові
Тестбенчі	Автогенерація	Ручне створення	Змішаний	Змішаний	Ручне
Покриття коду	Підтримується	Підтримується	Обмежена	Обмежена	Відсутня
Синтез і реалізація					
Синтез	Відсутній	Відсутній	Інтегрований	Інтегрований	Відсутній
FPGA підтримка	Універсальна	Через треті сторони	Xilinx	Intel/Altera	Відсутня
IP інтеграція	Обмежена	Обмежена	Розширена	Розширена	Відсутня
Використання					
Складність освоєння	Помірна	Помірна	Висока	Висока	Низька
Вартість	Комерційна	Комерційна	Комерційна	Комерційна	Безкоштовна
Цільова аудиторія	Академія/Промисловість	Промисловість	FPGA розробка	FPGA розробка	Дослідження

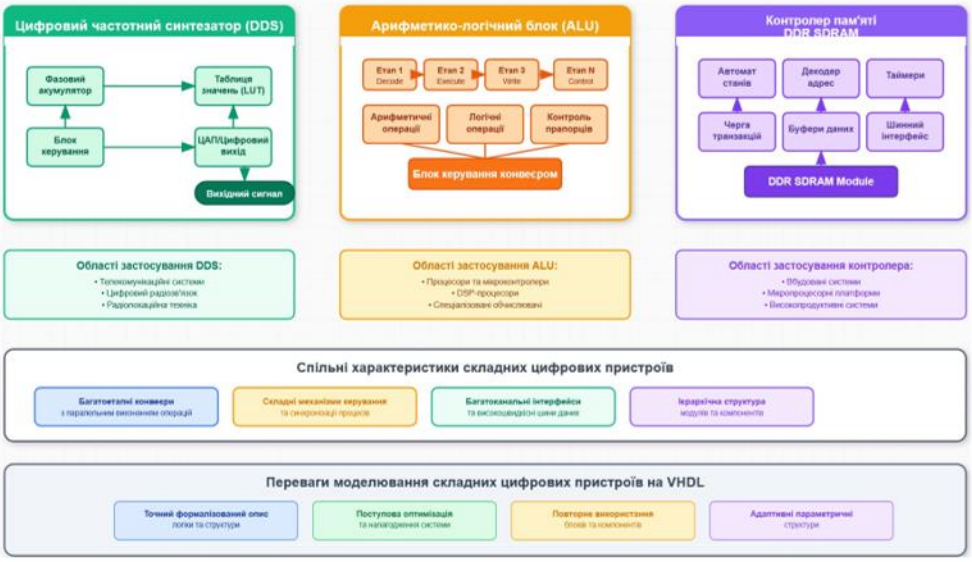
Критерії вибору цифрових пристроїв для моделювання на VHDL

5



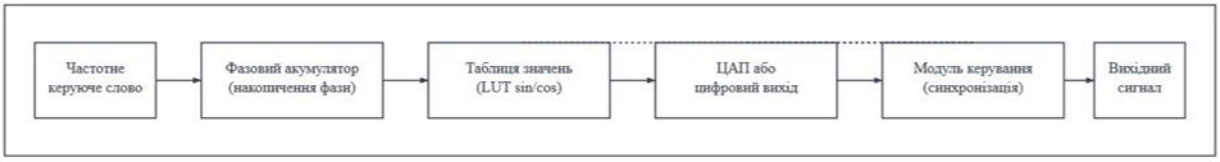
6

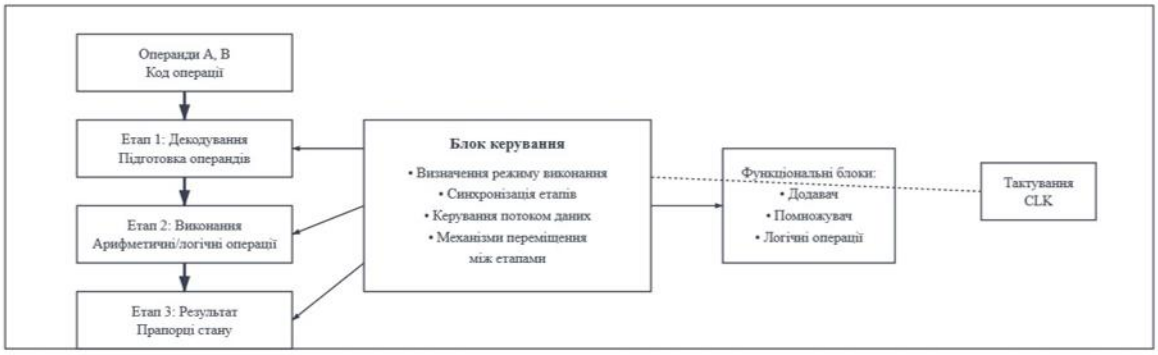
Архітектура обраних пристроїв



Цифровий структурний синтезатор. Конвейєрний АЛП

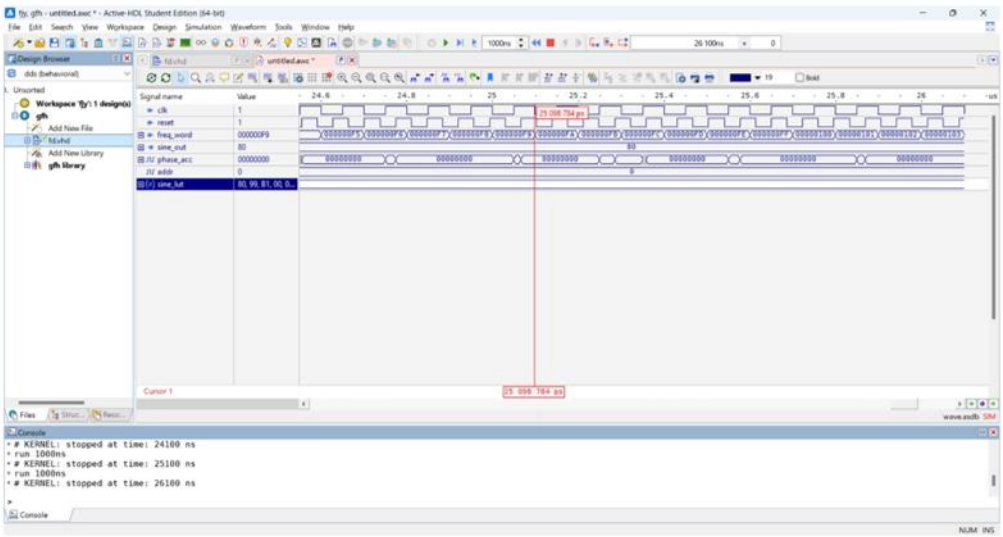
7



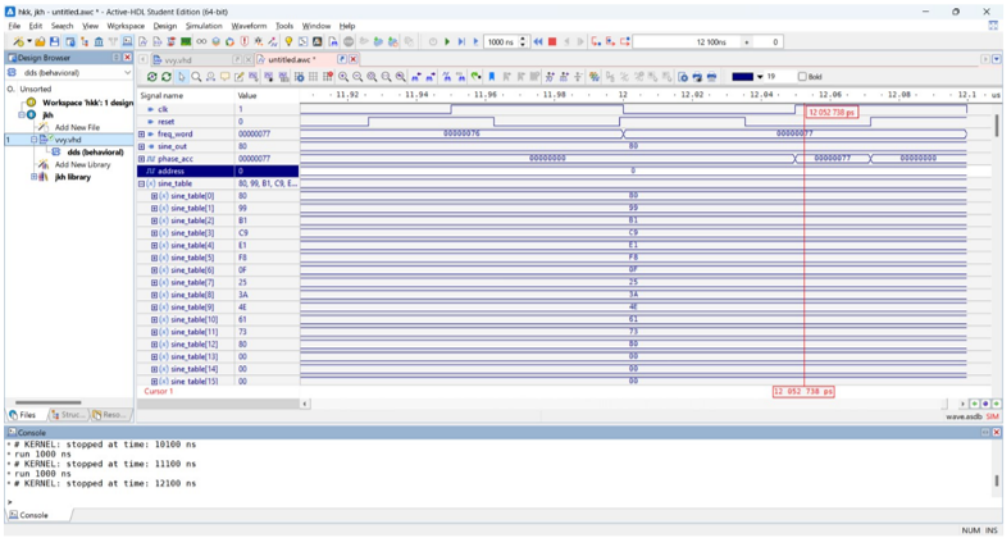


Моделювання цифрового частотного синтезатора

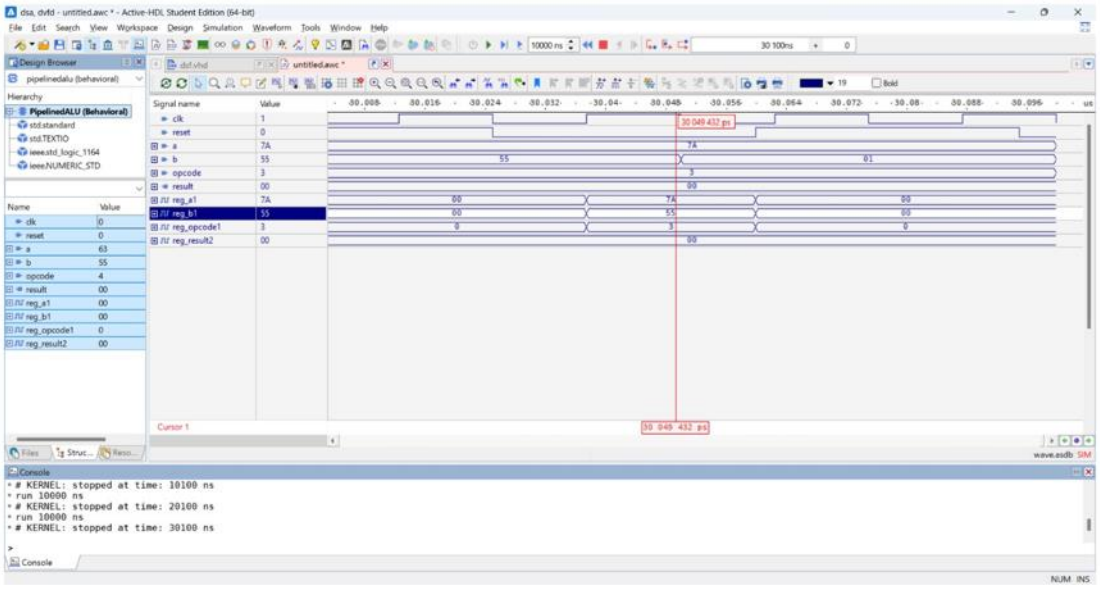
8



Моделювання генератора синусоїдального сигналу 9

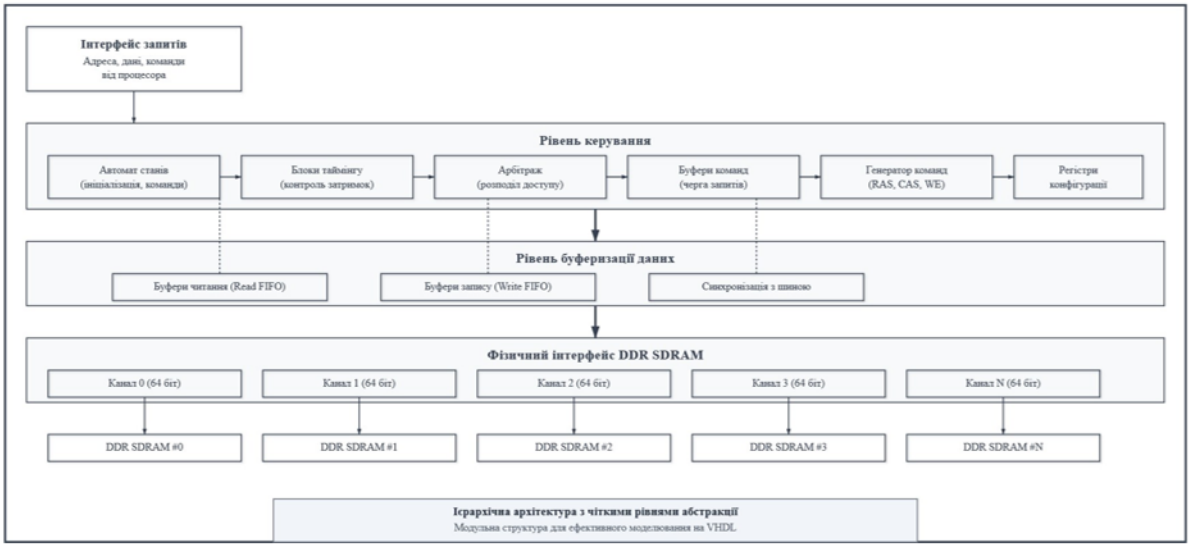


Конвейєрний АЛП 10



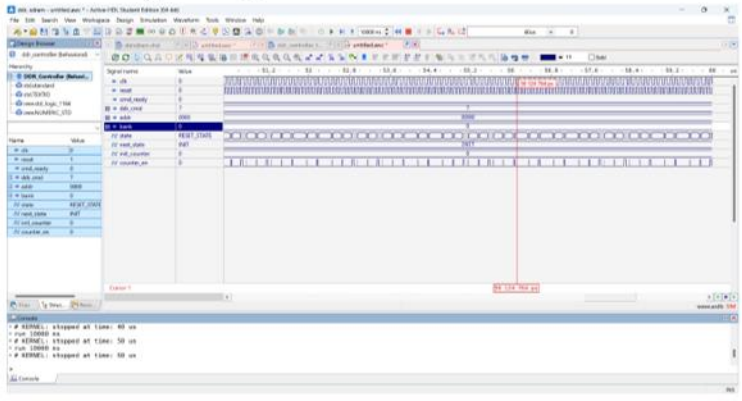
Багатоконтрольний контролер пам’яті DDR SDRAM

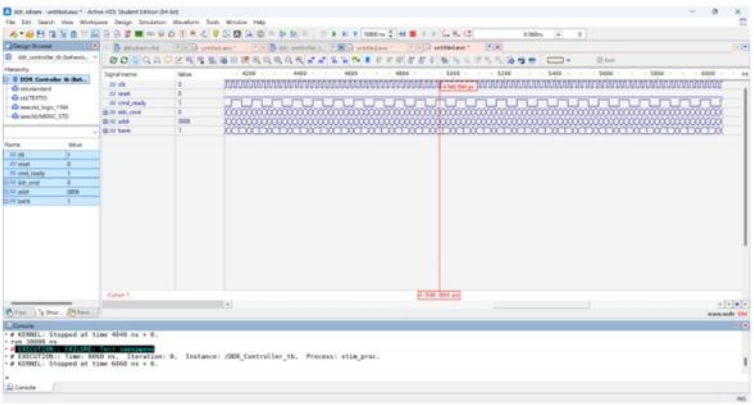
11



Моделювання:автомат станів DDR SDRAM контролера на VHDL та тестбенч

12





Висновки 13

У процесі реалізації було розроблено моделі трьох цифрових пристроїв: цифрового частотного синтезатора, конвеєрного арифметико-логічного пристрою та багатоканального контролера DDR SDRAM. Для кожного з них побудовано повноцінну структурну архітектуру, визначено сигнальні інтерфейси, внутрішню логіку, реалізовано автомат керування та забезпечено повну підтримку симуляції. Застосування мови VHDL дозволило забезпечити як поведінковий, так і структурний опис моделей, що сприяло високій точності відображення функціонування пристроїв та можливості подальшого синтезу для FPGA.

Усі реалізовані моделі були протестовані в середовищі Active-HDL та продемонстрували очікувану поведінку. Отримані часові діаграми підтвердили правильність логіки, відповідність сигналів інтерфейсним вимогам, а також стабільність роботи моделей у межах проєктних обмежень. Зокрема, моделювання DDS продемонструвало генерацію періодичного сигналу із частотою, що динамічно змінюється; конвеєрний ALU успішно реалізував обробку логічних і арифметичних операцій з мінімальною затримкою; а контролер DDR SDRAM довів здатність до управління складною послідовністю команд, арбітражу доступу та підтримки паралельної роботи кількох каналів.

ДОДАТОК Б

Програмний код

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

entity DDS is
    Port (
        clk      : in  STD_LOGIC;
        reset    : in  STD_LOGIC;
        freq_word: in  STD_LOGIC_VECTOR(31 downto 0);
        sine_out : out STD_LOGIC_VECTOR(7 downto 0)
    );
end DDS;

architecture Behavioral of DDS is
    signal phase_acc    : STD_LOGIC_VECTOR(31 downto 0) :=
(others => '0');
    type sine_lut_type is array (0 to 255) of STD_LOGIC_VECTOR(7
downto 0);

    constant sine_lut : sine_lut_type := (
        -- Приклад: заповнено лише перші значення, решта – нулі
        0 => "10000000", 1 => "10011001", 2 => "10110001", --
...
        others => "00000000"
    );

    signal addr : integer range 0 to 255;
begin
    process(clk, reset)
    begin
        if reset = '1' then
            phase_acc <= (others => '0');
        elsif rising_edge(clk) then
            phase_acc <= std_logic_vector(unsigned(phase_acc) +
unsigned(freq_word));
        end if;
    end process;

    addr <= to_integer(unsigned(phase_acc(31 downto 24)));
    sine_out <= sine_lut(addr);
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DDS is
```

```
    Port (
        clk       : in  STD_LOGIC;
        reset     : in  STD_LOGIC;
        freq_word : in  STD_LOGIC_VECTOR(31 downto 0);
        sine_out  : out STD_LOGIC_VECTOR(7 downto 0)
    );
end DDS;

architecture Behavioral of DDS is
    signal phase_acc : STD_LOGIC_VECTOR(31 downto 0) := (others
=> '0');
    signal address   : integer range 0 to 255;

    type sine_table_type is array (0 to 255) of
STD_LOGIC_VECTOR(7 downto 0);
    constant sine_table : sine_table_type := (
        0 => "10000000", 1 => "10011001", 2 => "10110001", 3 =>
"11001001",
        4 => "11100001", 5 => "11111000", 6 => "00001111", 7 =>
"00100101",
        8 => "00111010", 9 => "01001110", 10 => "01100001", 11
=> "01110011",
        12 => "10000000", -- решта значень — імітаційно
        others => "00000000"
    );

begin
    process(clk, reset)
    begin
        if reset = '1' then
            phase_acc <= (others => '0');
        elsif rising_edge(clk) then
            phase_acc <= std_logic_vector(unsigned(phase_acc) +
unsigned(freq_word));
        end if;
    end process;

    address <= to_integer(unsigned(phase_acc(31 downto 24)));
    sine_out <= sine_table(address);
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity PipelinedALU is
    Port (
        clk     : in  STD_LOGIC;
        reset   : in  STD_LOGIC;
        a       : in  STD_LOGIC_VECTOR(7 downto 0);
        b       : in  STD_LOGIC_VECTOR(7 downto 0);
        opcode  : in  STD_LOGIC_VECTOR(2 downto 0); -- 3-бітовий
код операції
        result  : out STD_LOGIC_VECTOR(7 downto 0)
```

```
    );
end PipelinedALU;

architecture Behavioral of PipelinedALU is
    -- Проміжні регістри між стадіями
    signal reg_a1, reg_b1     : STD_LOGIC_VECTOR(7 downto 0);
    signal reg_opcode1        : STD_LOGIC_VECTOR(2 downto 0);

    signal reg_result2        : STD_LOGIC_VECTOR(7 downto 0);
begin
    -- Стадія 1: Збереження вхідних даних
    process(clk, reset)
    begin
        if reset = '1' then
            reg_a1      <= (others => '0');
            reg_b1      <= (others => '0');
            reg_opcode1 <= (others => '0');
        elsif rising_edge(clk) then
            reg_a1      <= a;
            reg_b1      <= b;
            reg_opcode1 <= opcode;
        end if;
    end process;

    -- Стадія 2: Виконання операції
    process(clk, reset)
    begin
        if reset = '1' then
            reg_result2 <= (others => '0');
        elsif rising_edge(clk) then
            case reg_opcode1 is
                when "000" => reg_result2 <=
std_logic_vector(unsigned(reg_a1) + unsigned(reg_b1)); --
додавання
                when "001" => reg_result2 <=
std_logic_vector(unsigned(reg_a1) - unsigned(reg_b1)); --
віднімання
                when "010" => reg_result2 <= reg_a1 and reg_b1;
-- AND
                when "011" => reg_result2 <= reg_a1 or reg_b1;
-- OR
                when "100" => reg_result2 <= reg_a1 xor reg_b1;
-- XOR
                when "101" => reg_result2 <=
std_logic_vector(shift_left(unsigned(reg_a1), 1)); -- зсув вліво
                when "110" => reg_result2 <=
std_logic_vector(shift_right(unsigned(reg_a1), 1)); -- зсув
вправо
                when others => reg_result2 <= (others => '0');
            end case;
        end if;
    end process;
```

```
    -- Стадія 3: Вивід результату
    process(clk, reset)
    begin
        if reset = '1' then
            result <= (others => '0');
        elsif rising_edge(clk) then
            result <= reg_result2;
        end if;
    end process;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DDR_Controller is
    Port (
        clk       : in  STD_LOGIC;
        reset     : in  STD_LOGIC;
        cmd_ready : out STD_LOGIC;
        ddr_cmd   : out STD_LOGIC_VECTOR(2 downto 0); --
RAS/CAS/WE
        addr      : out STD_LOGIC_VECTOR(15 downto 0);
        bank      : out STD_LOGIC_VECTOR(2 downto 0)
    );
end DDR_Controller;

architecture Behavioral of DDR_Controller is
    type state_type is (RESET, INIT, IDLE, ACTIVE, READ,
PRECHARGE, REFRESH);
    signal state, next_state : state_type;

    signal init_counter : integer range 0 to 255 := 0;
begin

    -- Машина станів
    process(clk, reset)
    begin
        if reset = '1' then
            state <= RESET;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    process(state, init_counter)
    begin
        case state is
            when RESET =>
                cmd_ready <= '0';
                ddr_cmd   <= "111"; -- NOP
                addr      <= (others => '0');
                bank      <= "000";
```

```
                next_state <= INIT;

            when INIT =>
                if init_counter < 100 then
                    init_counter <= init_counter + 1;
                    ddr_cmd <= "111"; -- NOP
                    next_state <= INIT;
                else
                    next_state <= IDLE;
                end if;

            when IDLE =>
                cmd_ready <= '1';
                ddr_cmd   <= "000"; -- ACTIVE
                addr      <= "0000000000001000";
                bank      <= "001";
                next_state <= READ;

            when READ =>
                cmd_ready <= '1';
                ddr_cmd   <= "101"; -- CAS = read
                addr      <= "0000000000000100";
                bank      <= "001";
                next_state <= PRECHARGE;

            when PRECHARGE =>
                ddr_cmd   <= "010"; -- PRE
                addr      <= "0000000000010000";
                bank      <= "001";
                next_state <= REFRESH;

            when REFRESH =>
                ddr_cmd   <= "001"; -- REFRESH
                addr      <= (others => '0');
                bank      <= "000";
                next_state <= IDLE;

            when others =>
                next_state <= RESET;
        end case;
    end process;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Arbiter2ch is
    Port (
        clk         : in  STD_LOGIC;
        reset       : in  STD_LOGIC;
        req_0       : in  STD_LOGIC;
        req_1       : in  STD_LOGIC;
        grant_0     : out STD_LOGIC;
```

```
        grant_1     : out STD_LOGIC
    );
end Arbiter2ch;

architecture Behavioral of Arbiter2ch is
    signal turn : STD_LOGIC := '0';
begin
    process(clk, reset)
    begin
        if reset = '1' then
            turn <= '0';
        elsif rising_edge(clk) then
            if turn = '0' and req_0 = '1' then
                grant_0 <= '1';
                grant_1 <= '0';
                turn <= '1';
            elsif turn = '1' and req_1 = '1' then
                grant_0 <= '0';
                grant_1 <= '1';
                turn <= '0';
            else
                grant_0 <= '0';
                grant_1 <= '0';
            end if;
        end if;
    end process;
end Behavioral;
```