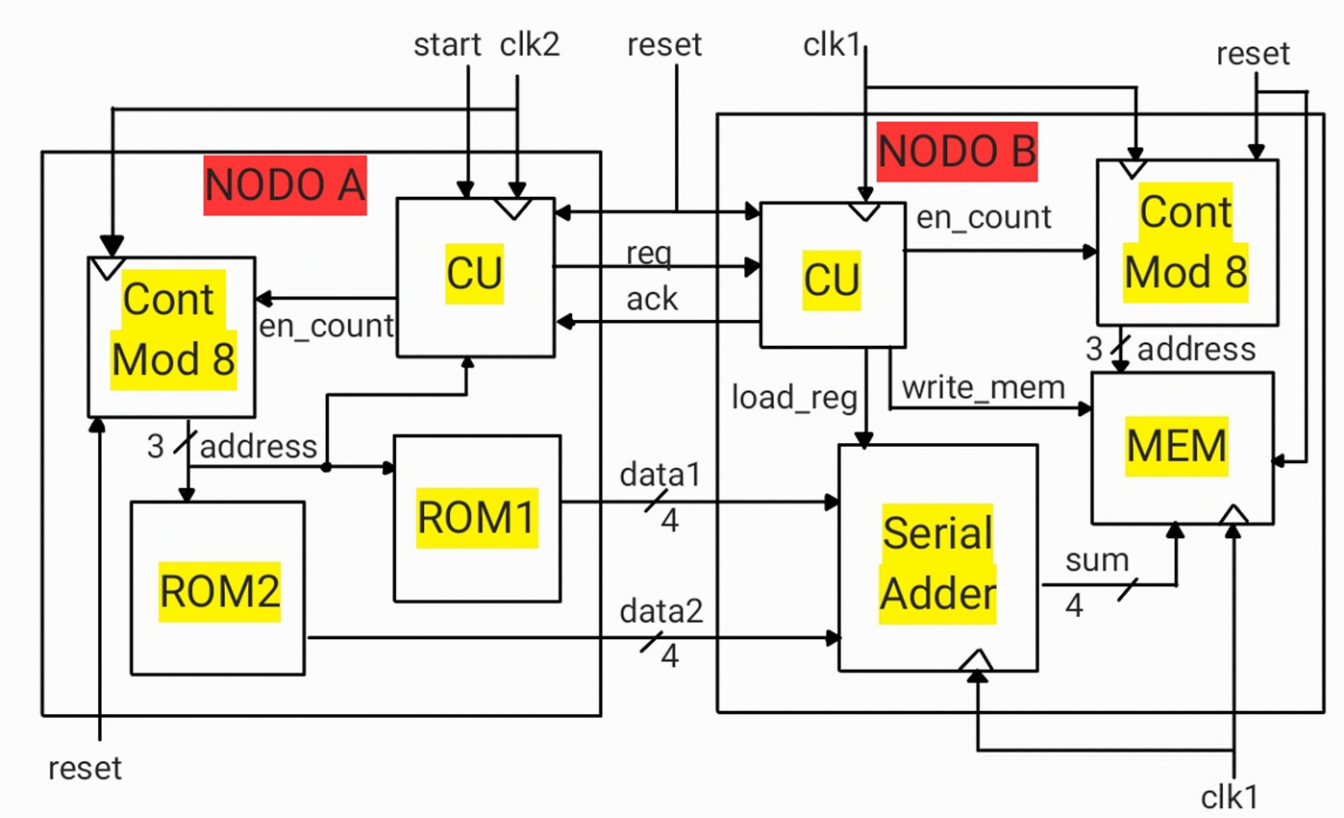
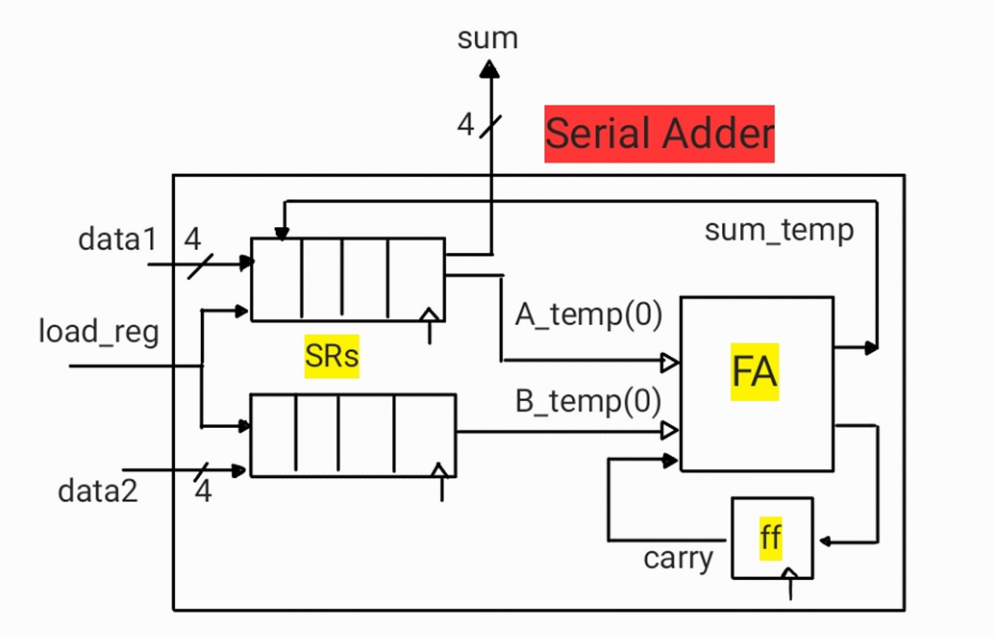
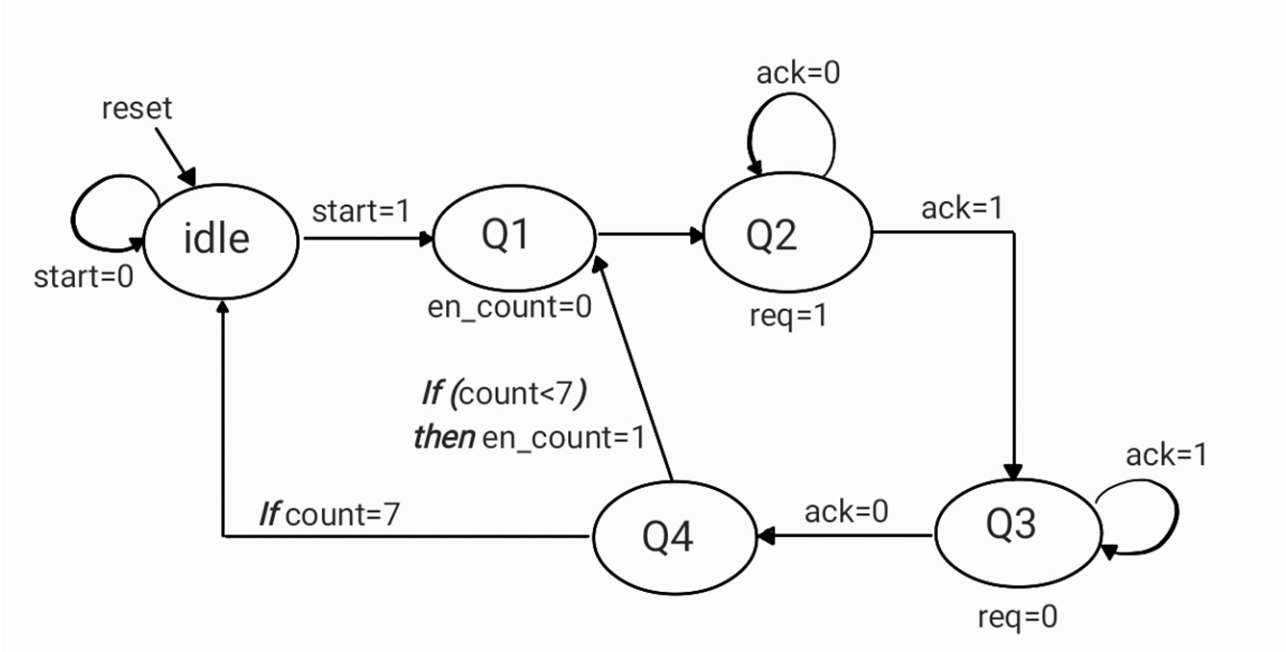
Progettare, implementare in VHDL e simulare la seguente architettura. Un sistema è composto da due nodi, A e B. A contiene 2 ROM, ROM1 e ROM2, in cui sono memorizzate 8 stringhe da 4 bit ciascuno. A legge una stringa da ROM1 e una stringa da ROM2 e le invia, una alla volta mediante handshaking semplice, a B. B somma le 2 stringhe ricevute utilizzando un sommatore seriale (cioè, un sommatore che calcola la somma un bit alla volta iterativamente) e memorizza il risultato in una memoria interna MEM.

Nota: seppure sia possibile implementare la funzione dei singoli moduli come le memorie o i registri con paradigma, ad esempio, data-flow o comportamentale, è richiesto che l'implementazione VHDL segua una progettazione strutturale per l'interconnessione dei moduli.

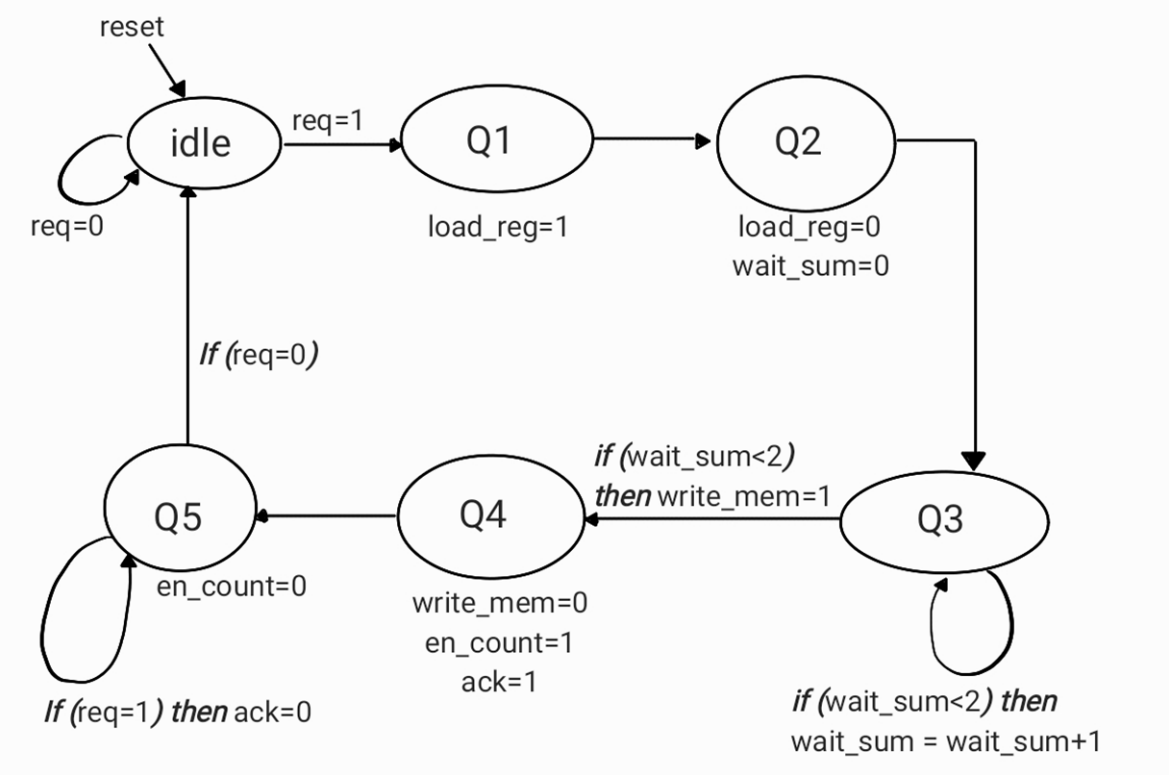




- Diagramma degli stati di A



- Diagramma degli stati di B



NODO A

ROM1.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity ROM1 is
  generic (
    DATA_WIDTH    : integer := 8;    -- Larghezza del dato in bit
    DEPTH         : integer := 16;    -- Dimensione della ROM
    ADDRESS_WIDTH : integer := 4     -- Larghezza dell'indirizzo in bit
  );
  Port (
    address : in STD_LOGIC_VECTOR(ADDRESS_WIDTH-1 downto 0);
    dataOut : out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
  );
end ROM1;

architecture Behavioral of ROM1 is

  type ROMArray is array (0 to DEPTH-1) of STD_LOGIC_VECTOR(DATA_WIDTH-1
downto 0);
  signal rom_data : ROMArray;
```

```
begin

  init_ROM: process -- 0,1,2,3,4...,DEPTH-2,DEPTH-1
  begin
    for i in 0 to DEPTH-1 loop
      rom_data(i) <= std_logic_vector(to_unsigned(i,DATA_WIDTH));
    end loop;
    wait;
  end process;

  process(address)
  begin
    dataOut <= rom_data(to_integer(unsigned(address)));
  end process;
end Behavioral;
```

ROM2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity ROM2 is
  generic (
    DATA_WIDTH    : integer := 8;    -- Larghezza del dato in bit
    DEPTH         : integer := 16;    -- Dimensione della ROM
    ADDRESS_WIDTH : integer := 4     -- Larghezza dell'indirizzo in bit
  );
  Port (
    address : in STD_LOGIC_VECTOR(ADDRESS_WIDTH-1 downto 0);
    dataOut : out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
  );
end ROM2;

architecture Behavioral of ROM2 is

  type ROMArray is array (0 to DEPTH-1) of STD_LOGIC_VECTOR(DATA_WIDTH-1
downto 0);
  signal rom_data : ROMArray;

begin

  init_ROM: process -- DEPTH-1,DEPTH-2,DEPTH-3,...,1,0
  begin
    for i in 0 to DEPTH-1 loop
      rom_data(i) <= std_logic_vector(to_unsigned(DEPTH-1-i, DATA_WIDTH));
    end loop;
    wait;
```

```
  end process;

  process(address)
  begin
    dataOut <= rom_data(to_integer(unsigned(address)));
  end process;
end Behavioral;
```

---

counter.vhd

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

entity counter is
  generic (
    WIDTH : integer := 8  -- Numero di bit usati per rappresentare il
conteggio
  );
  Port (
    clk    : in STD_LOGIC;
    rst    : in STD_LOGIC;
    enable : in STD_LOGIC;
    count  : out STD_LOGIC_VECTOR(WIDTH-1 downto 0)
  );
end counter;

architecture Behavioral of counter is

  signal c : STD_LOGIC_VECTOR(WIDTH-1 downto 0) := (others => '0');

begin
  process(clk, rst)
  begin
    if rst = '1' then
      c <= (others => '0');
    elsif rising_edge(clk) and enable = '1' then
      c <= c + 1;
    end if;
  end process;

  count <= c;
end Behavioral;
```

---

control_unit_A.vhd

```
library IEEE;
```

```vhdl
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity control_unit_A is
    Port ( clock : in STD_LOGIC;
           start : in STD_LOGIC;
           reset : in STD_LOGIC;
           ack : in std_logic;
           req : out std_logic;
           address : in std_logic_vector(2 downto 0);
           enable_cont : out std_logic
           );
end control_unit_A;

architecture Behavioral of control_unit_A is

    type stato is (idle, Q1, Q2, Q3, Q4);
    signal stato_corrente : stato := idle;

begin

process(clock)
    begin
        if rising_edge(clock) then
            if reset = '1' then
                stato_corrente <= idle;
            else
                case stato_corrente is
                   when idle =>
                      if(start='1') then
                        stato_corrente <= Q1;
                      else
                        stato_corrente <= idle;
                      end if;

                   when Q1 =>
                       enable_cont <= '0';
                       stato_corrente <= Q2;

                   when Q2 =>
                       req <= '1';
                       if(ack='0') then
                           stato_corrente <= Q2;
                       else stato_corrente <= Q3;
                       end if;

                   when Q3 =>
                       req <= '0';
                       if(ack='1') then
                           stato_corrente <= Q3;
```

```
                        else
                            stato_corrente <= Q4;
                        end if;

                    when Q4 =>
                        if(address="111") then
                            stato_corrente <= idle;
                        else
                           enable_cont <= '1';
                           stato_corrente <= Q1;
                        end if;
                end case;
            end if;
        end if;
    end process;

end Behavioral;
```

---

node_A.vhd

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

entity node_A is
   Port (  clock : in STD_LOGIC;
           start : in STD_LOGIC;
           reset : in STD_LOGIC;
           ack : in std_logic;
           req : out std_logic;
           data1 : out std_logic_vector(3 downto 0);
           data2 : out std_logic_vector(3 downto 0) );
end node_A;

architecture Behavioral of node_A is
    component ROM1 is
      generic (
        DATA_WIDTH    : integer := 8;
        DEPTH         : integer := 16;
        ADDRESS_WIDTH : integer := 4
      );
      Port (
        address : in STD_LOGIC_VECTOR(ADDRESS_WIDTH-1 downto 0);
        dataOut : out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
      );
    end component;

    component ROM2 is
      generic (
        DATA_WIDTH    : integer := 8;
```

```
        DEPTH         : integer := 16;
        ADDRESS_WIDTH : integer := 4
      );
      Port (
        address : in STD_LOGIC_VECTOR(ADDRESS_WIDTH-1 downto 0);
        dataOut : out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
      );
    end component;

    component counter is
      generic (
        WIDTH : integer := 8
      );
      Port (
        clk    : in STD_LOGIC;
        rst    : in STD_LOGIC;
        enable : in STD_LOGIC;
        count  : out STD_LOGIC_VECTOR(WIDTH-1 downto 0)
      );
    end component;

    component control_unit_A is
        Port ( clock : in STD_LOGIC;
               start : in STD_LOGIC;
               reset : in STD_LOGIC;
               ack : in std_logic;
               req : out std_logic;
               address : in std_logic_vector(2 downto 0);
               enable_cont : out std_logic
               );
    end component;

    signal address : std_logic_vector(2 downto 0);
    signal enable_count : std_logic;
begin
    cont_addr: counter
          generic map (
            WIDTH => 3
          )
          port map(
            clk => clock,
            rst => reset,
            enable => enable_count,
            count => address
          );

    mem1 : ROM1
        generic map (
            DATA_WIDTH => 4,
            DEPTH => 8,
```

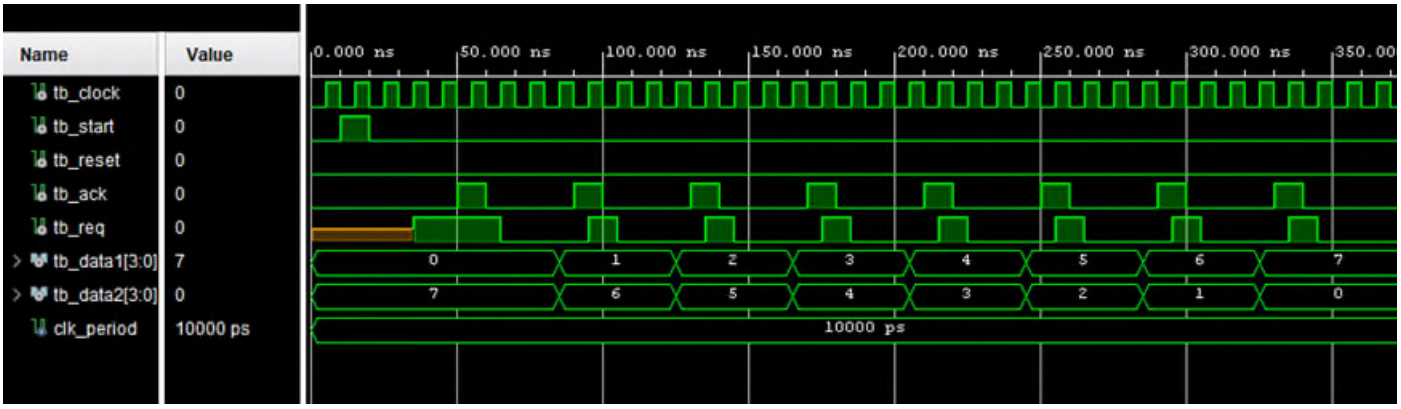
```
            ADDRESS_WIDTH => 3
        ) port map (
            address => address,
            dataOut => data1
        );

    mem2 : ROM2
        generic map (
            DATA_WIDTH => 4,
            DEPTH => 8,
            ADDRESS_WIDTH => 3
        ) port map (
            address => address,
            dataOut => data2
        );

    cu_A : control_unit_A
        port map (
            clock => clock,
            start => start,
            reset => reset,
            ack => ack,
            req => req,
            address => address,
            enable_cont => enable_count
       );

end Behavioral;
```

---

node_A_tb.vhd

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

entity nodo_A_tb is
end nodo_A_tb;

architecture behavior of nodo_A_tb is
    signal tb_clock : std_logic := '0';
    signal tb_start : std_logic := '0';
    signal tb_reset : std_logic := '0';
    signal tb_ack : std_logic := '0';
    signal tb_req : std_logic;
    signal tb_data1 : std_logic_vector(3 downto 0);
    signal tb_data2 : std_logic_vector(3 downto 0);

    constant clk_period : time := 10 ns;

    component node_A is
```

```
        Port (
            clock : in STD_LOGIC;
            start : in STD_LOGIC;
            reset : in STD_LOGIC;
            ack : in std_logic;
            req : out std_logic;
            data1 : out std_logic_vector(3 downto 0);
            data2 : out std_logic_vector(3 downto 0)
        );
    end component;

begin
   uut: node_A port map (
            clock => tb_clock,
            start => tb_start,
            reset => tb_reset,
            ack => tb_ack,
            req => tb_req,
            data1 => tb_data1,
            data2 => tb_data2
        );

    clk : process
    begin
        tb_clock <= '0';
        wait for clk_period / 2;
        tb_clock <= '1';
        wait for clk_period / 2;
    end process;

    stim: process
    begin
        wait for 10 ns;

        tb_start <= '1';
        wait for 10 ns;
        tb_start <= '0';

        for i in 1 to 8 loop
            wait for 30 ns;
            tb_ack <= '1';
            wait for 10 ns;
            tb_ack <= '0';
        end loop;

        wait;
    end process;
end behavior;
```



NODO B

MEM.vhd

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

entity memory is
  generic (
    DATA_WIDTH : integer := 8;    -- Larghezza del dato in bit
    DEPTH      : integer := 16;   -- Dimensione della memoria
    ADDR_WIDTH : integer := 4     -- Larghezza dell'indirizzo in bit
  );
  Port (
    clk     : in STD_LOGIC;
    rst     : in STD_LOGIC;
    write   : in STD_LOGIC;
    read    : in STD_LOGIC;
    address : in STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
    dataIn  : in STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
    dataOut : out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
  );
end memory;

architecture Behavioral of memory is

  type array_memoria is array (0 to DEPTH-1) of STD_LOGIC_VECTOR(DATA_WIDTH-1
downto 0);
  signal memoria_interna : array_memoria := (others => (others => '0'));

begin
  process(clk, rst)
  begin
    if rst = '1' then
```

```
      memoria_interna <= (others => (others => '0'));

    elsif rising_edge(clk) then

      if write = '1' then
        memoria_interna(to_integer(unsigned(address))) <= dataIn;
      end if;

      if read = '1' then
        dataOut <= memoria_interna(to_integer(unsigned(address)));
      end if;

    end if;
  end process;
end Behavioral;
```

---

serial_adder.vhd

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

entity serial_adder is
    generic (
        N : INTEGER := 4
        );
    port(
        clock : in STD_LOGIC;
        A : in STD_LOGIC_VECTOR (N-1 downto 0);
        B : in STD_LOGIC_VECTOR (N-1 downto 0);
        load : in STD_LOGIC;
        sum : out STD_LOGIC_VECTOR(N-1 downto 0)
        );
end serial_adder;

architecture Behavioral of serial_adder is

    signal carry: STD_LOGIC;
    signal A_temp: STD_LOGIC_VECTOR(N-1 downto 0);
    signal B_temp: STD_LOGIC_VECTOR(N-1 downto 0);

begin

    process(clock)
    variable sum_temp: STD_LOGIC;
    begin
        if(rising_edge(clock)) then
            if (load = '1') then
               A_temp <= A;
```

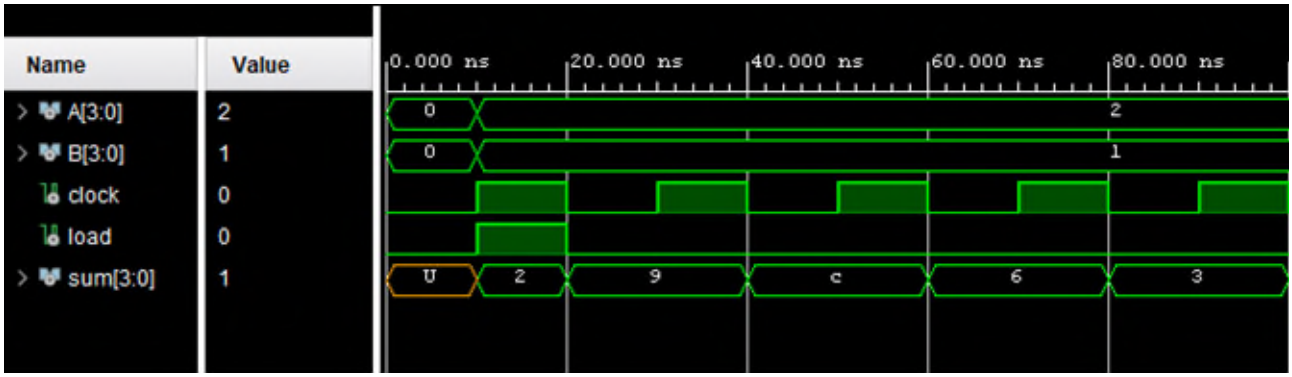
```
                B_temp <= B;
                carry <= '0';
                 end if;
            else
                 B_temp <= (B_temp(0) & B_temp(N-1 downto 1));
                 sum_temp := (A_temp(0) xor B_temp(0)) xor (carry);
                 A_temp <= (sum_temp & A_temp(N-1 downto 1));
                 carry <= (((A_temp(0) xor B_temp(0)) and carry)) or
(A_temp(0) and B_temp(0));
            end if;
    end process;

    sum <= A_temp;

end Behavioral;
```

---

serial_adder_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;

entity serial_adder_tb is
end serial_adder_tb;

architecture testbench of serial_adder_tb is

    component serial_adder
        port (
            A : in std_logic_vector (3 downto 0);
            B : in std_logic_vector (3 downto 0);
            clock : in std_logic;
            load : in std_logic;
            sum : out std_logic_vector (3 downto 0)
        );
    end component;

    signal A : std_logic_vector (3 DownTo 0) := "0000";
    signal B : std_logic_vector (3 DownTo 0) := "0000";
    signal clock : std_logic := '0';
    signal load : std_logic := '0';
    signal sum : std_logic_vector (3 downto 0) := "0000";

    begin
        dut : serial_adder
        port map (
```

```
            A => A,
            B => B,
            clock => clock,
            load => load,
            sum => sum
        );

        clk: process
        begin
            clock <= '0';
            wait for 10ns;
            clock <= '1';
            wait for 10ns;
        end process;

        stim: process
            begin
                wait for 10 ns;
                load <= '1';
                A <= "0010";
                B <= "0001";
                wait for 10 ns;
                load <= '0';

                wait;
            end process;
end testbench;
```



---

control_unit_B.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity control_unit_B is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           en_count : out STD_LOGIC;
```

```
            load_reg : out STD_LOGIC;
            write_mem : out STD_LOGIC;
            req : in std_logic;
            ack : out std_logic);
end control_unit_B;

architecture Behavioral of control_unit_B is

    type state is (idle, Q1, Q2, Q3, Q4, Q5);
    signal current_state: state;

begin

    process(clock)
    variable wait_sum : integer := 0;
    begin
        if rising_edge(clock) then
            if reset = '1' then
                current_state <= idle;
            else
                case current_state is
                    when idle =>
                        load_reg <= '0';
                        if req = '1' then
                            current_state <= Q1;
                        else
                            current_state <= idle;
                        end if;

                    when Q1 =>
                        load_reg <= '1';
                        en_count <= '0';
                        current_state <= Q2;

                    when Q2 =>
                        load_reg <= '0';
                        wait_sum := 0;
                        current_state <= Q3;

                    when Q3 =>
                        if(wait_sum=2) then
                            write_mem <='1';
                            current_state <= Q4;
                        else
                            wait_sum := wait_sum +1;
                            current_state <= Q3;
                        end if;

                    when Q4 =>
                        write_mem <= '0';
```

```
                        en_count <= '1';
                        ack <= '1';
                        current_state <= Q5;

                    when Q5 =>
                        en_count <= '0';
                        if req = '1' then
                            ack <= '0';
                            current_state <= idle;
                        else
                            current_state <= Q5;
                        end if;
                end case;
            end if;
        end if;
    end process;

end Behavioral;
```

---

node_B.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity nodo_B is
  Port (
    clock : in std_logic;
    reset : in std_logic;
    req : in std_logic;
    ack : out std_logic;
    data1 : in std_logic_vector(3 downto 0);
    data2 : in std_logic_vector(3 downto 0));
end nodo_B;

architecture Structural of nodo_B is

    component serial_adder is
    generic (N : INTEGER:=4);
    port(
        A: in STD_LOGIC_VECTOR (N-1 downto 0);
        B: in STD_LOGIC_VECTOR (N-1 downto 0);
        clock: in STD_LOGIC;
        load: in STD_LOGIC;
        sum: out STD_LOGIC_VECTOR(N-1 downto 0));
    end component;

    component MEM is
      generic (
```

```
        DATA_WIDTH : integer := 8;
        DEPTH      : integer := 16;
        ADDR_WIDTH : integer := 4
      );
      Port (
        clk     : in STD_LOGIC;
        rst     : in STD_LOGIC;
        write   : in STD_LOGIC;
        read    : in STD_LOGIC;
        address : in STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
        dataIn  : in STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0);
        dataOut : out STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
      );
    end component;

    component counter is
      generic (
        WIDTH : integer := 8
      );
      Port (
        clk    : in STD_LOGIC;
        rst    : in STD_LOGIC;
        enable : in STD_LOGIC;
        count  : out STD_LOGIC_VECTOR(WIDTH-1 downto 0)
      );
    end component;

    component control_unit_B is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           en_count : out STD_LOGIC;
           load_reg : out STD_LOGIC;
           write_mem : out STD_LOGIC;
           req : in std_logic;
           ack : out std_logic);
    end component;

    signal load_reg, write_mem, en_count : std_logic;
    signal op1, op2, ris : std_logic_vector(3 downto 0);
    signal address : std_logic_vector(2 downto 0);

begin

    count: counter
        generic map(
            WIDTH=>3
        ) port map(
            clk => clock,
            rst => reset,
            enable => en_count,
```

```
            count => address
        );

    ser_add : serial_adder
        generic map(
            N=>4
        ) port map(
            A => data1,
            B => data2,
            clock => clock,
            load => load_reg,
            sum => ris);

    m_e_m : MEM
        generic map (
            DATA_WIDTH => 4,
            DEPTH => 8,
            ADDR_WIDTH => 3
        ) port map (
              clk => clock,
              rst => reset,
              write => write_mem,
              read => '0',
              address => address,
              dataIn => ris,
              dataOut => open);

    cu_B : control_unit_B port map (
        clock => clock,
        reset => reset,
        en_count => en_count,
        load_reg => load_reg,
        write_mem => write_mem,
        req => req,
        ack => ack
    );
end Structural;
```

SISTEMA FORMATO DAI NODI A E B

---

total_system_tb.vhd

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

entity total_system_tb is
```

```
end total_system_tb;

architecture Behavioral of total_system_tb is
    component node_A is
   Port (  clock : in STD_LOGIC;
           start : in STD_LOGIC;
           reset : in STD_LOGIC;
           ack : in std_logic;
           req : out std_logic;
           data1 : out std_logic_vector(3 downto 0);
           data2 : out std_logic_vector(3 downto 0) );
    end component;

    component node_B is
      Port (
        clock : in std_logic;
        reset : in std_logic;
        req : in std_logic;
        ack : out std_logic;
        data1 : in std_logic_vector(3 downto 0);
        data2 : in std_logic_vector(3 downto 0));
    end component;

    signal clk1, clk2, start, rst, ack, req : std_logic := '0';
    signal data1, data2 : std_logic_vector(3 downto 0) := (others=>'0');
begin
    n_A : node_A port map (
        clock => clk2,
        start => start,
        reset => rst,
        ack => ack,
        req => req,
        data1 => data1,
        data2 => data2
    );

    n_B : node_B port map (
        clock => clk1,
        reset => rst,
        ack => ack,
        req => req,
        data1 => data1,
        data2 => data2
    );

    clock_gen1: process
    begin
        wait for 2 ns;
        clk1 <= not clk1;
    end process;
```

```
    clock_gen2: process
    begin
        wait for 5 ns;
        clk2 <= not clk2;
    end process;

    process
    begin
        wait for 1 ns;
        start<='1';
        wait for 6 ns;
        start<='0';
        wait for 900 ns;
        wait;
    end process;

end Behavioral;
```