**פרויקט VHDL בקורס "תכן לוגי ומבוא**

גאסם גריפאת:318475902

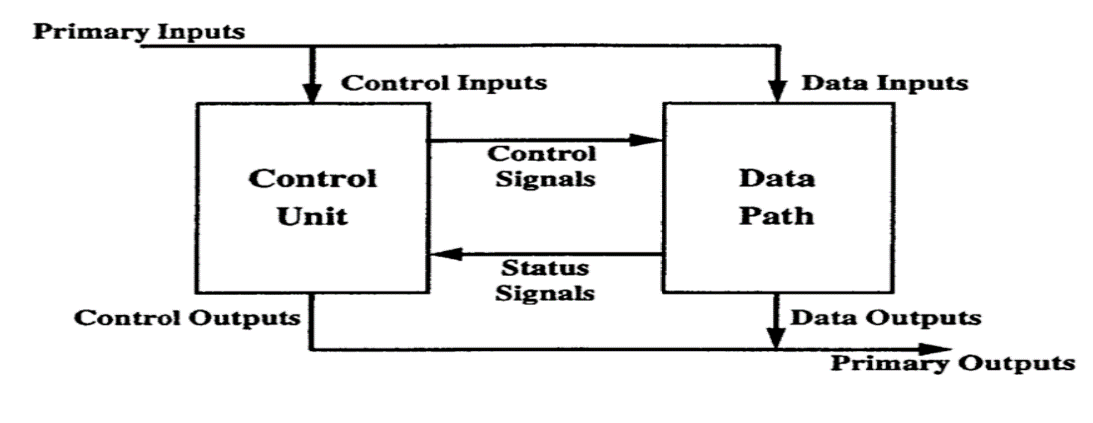
סלימאן קזל:206702128

**מבוא :**

**המערכת שיש לממש בפרויקט**

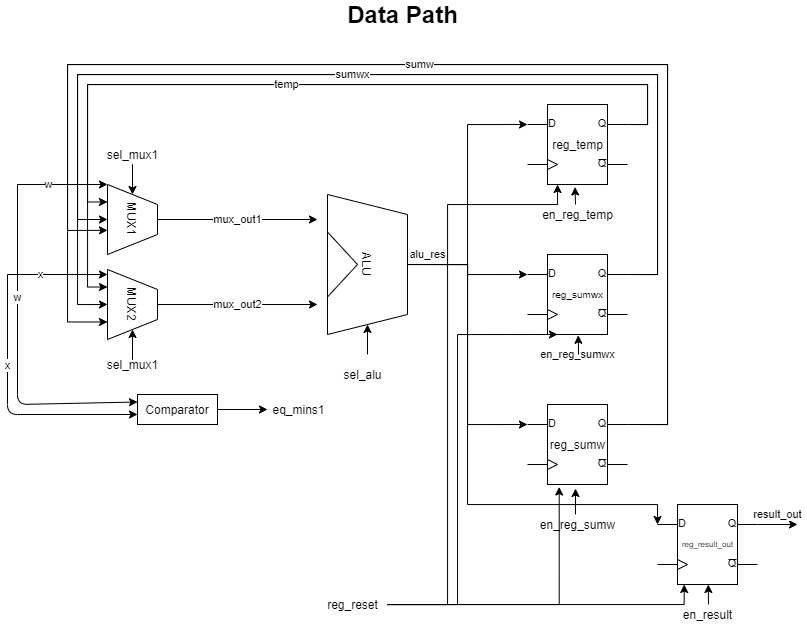
* **מטרת המערכת:** **יש לממש מערכת לחישוב ממוצע ציונים לקורסים תוך שימוש בעיקרון הפרדת מסלול נתונים ובקר.**
* **המערכת**:

המערכת בנויה על עקרון ההפרדה של מסלול נתונים ובקר.

לכן היא מורכבת ממסלול נתונים בנפרד ו בקר (המוח של המערכת) ששולט במסלול נתונים בעזרת אותות בקרה.

**מימוש מסלול הנתונים:**

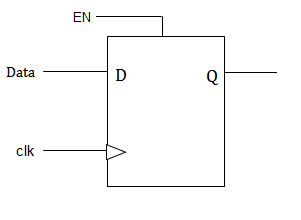
הנתונים במערכת ייוצגו בשיטת המשלים ל- 2 ב- 16 סיביות

**תרשים למסלול הנתונים:**

**תיאור היררכי של הקבצים בפרוייקט:**



**יחידות המרכיבות את מסלול הנתונים:**

**D Flip Flop**

**הסבר:**

**קוד VHDL:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity df\_f is

    port(

        clock: in std\_logic;

        reset: in std\_logic;

        enable: in std\_logic;

        d: in std\_logic\_vector(15 downto 0);

        q: out std\_logic\_vector(15 downto 0):= (others=> '0')

    );

end df\_f;

architecture arch\_df\_f of df\_f is

begin

    process(clock,enable,d,reset) begin

        if(reset = '1') then

                q <= (others=> '0');

        elsif rising\_edge(clock) then

            if(enable = '1') then

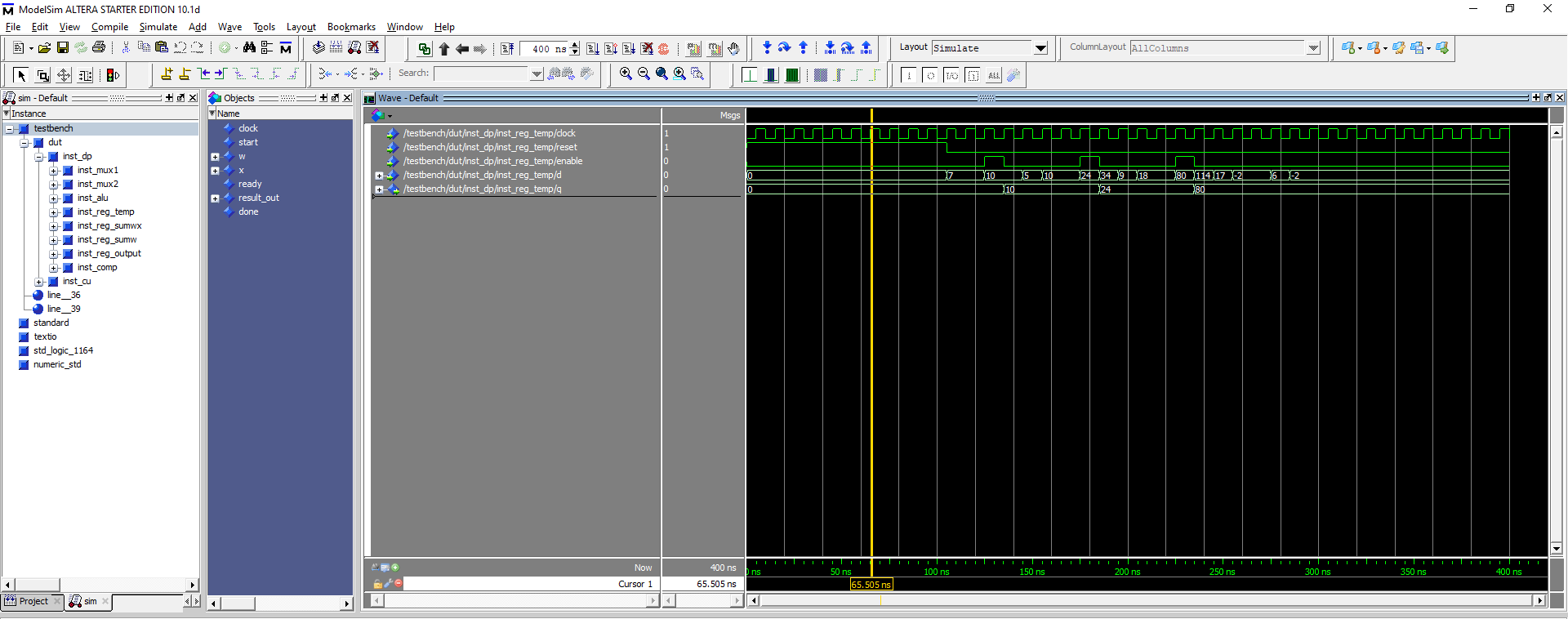
                q <= d after 2 ps;

            end if;

        end if;

    end process;

end arch\_df\_f;



**Mux**

**הסבר:**

.ALU של ה תפקיד הבורר הוא לבחור איזו תוצאה תצא החוצה מתוך הריגיסטרים אל הכניסה

**קוד VHDL:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity mux is

    port(

        mux\_in0: in std\_logic\_vector(15 downto 0);

        mux\_in1: in std\_logic\_vector(15 downto 0);

        mux\_in2: in std\_logic\_vector(15 downto 0);

        mux\_in3: in std\_logic\_vector(15 downto 0);

        mux\_sel: in std\_logic\_vector(1 downto 0);

        mux\_out: out std\_logic\_vector(15 downto 0)

    );

end mux;

architecture arch\_mux of mux is

begin

    process (mux\_sel,mux\_in0,mux\_in1,mux\_in2,mux\_in3) begin

        case mux\_sel is

            when "00" => mux\_out <= mux\_in0 after 2 ps;

            when "01" => mux\_out <= mux\_in1 after 2 ps ;

            when "10" => mux\_out <= mux\_in2 after 2 ps ;

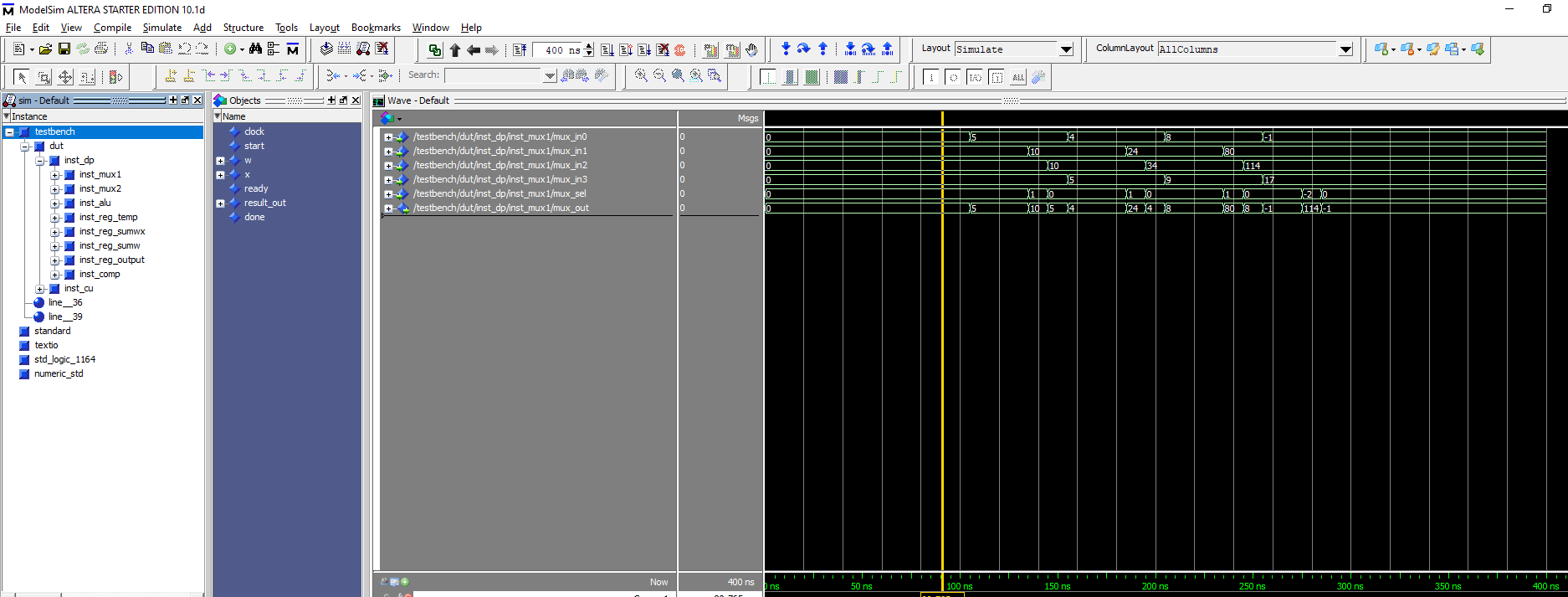
            when "11" => mux\_out <= mux\_in3 after 2 ps ;

            when others => mux\_out <= (others=> '0');

        end case;

    end process;

end arch\_mux;



# ***Comparator***

## **VHDL Code:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity comparator is

    port(

        w: in std\_logic\_vector(15 downto 0);

        x: in std\_logic\_vector(15 downto 0);

        eq\_mins1: out std\_logic

    );

end comparator;

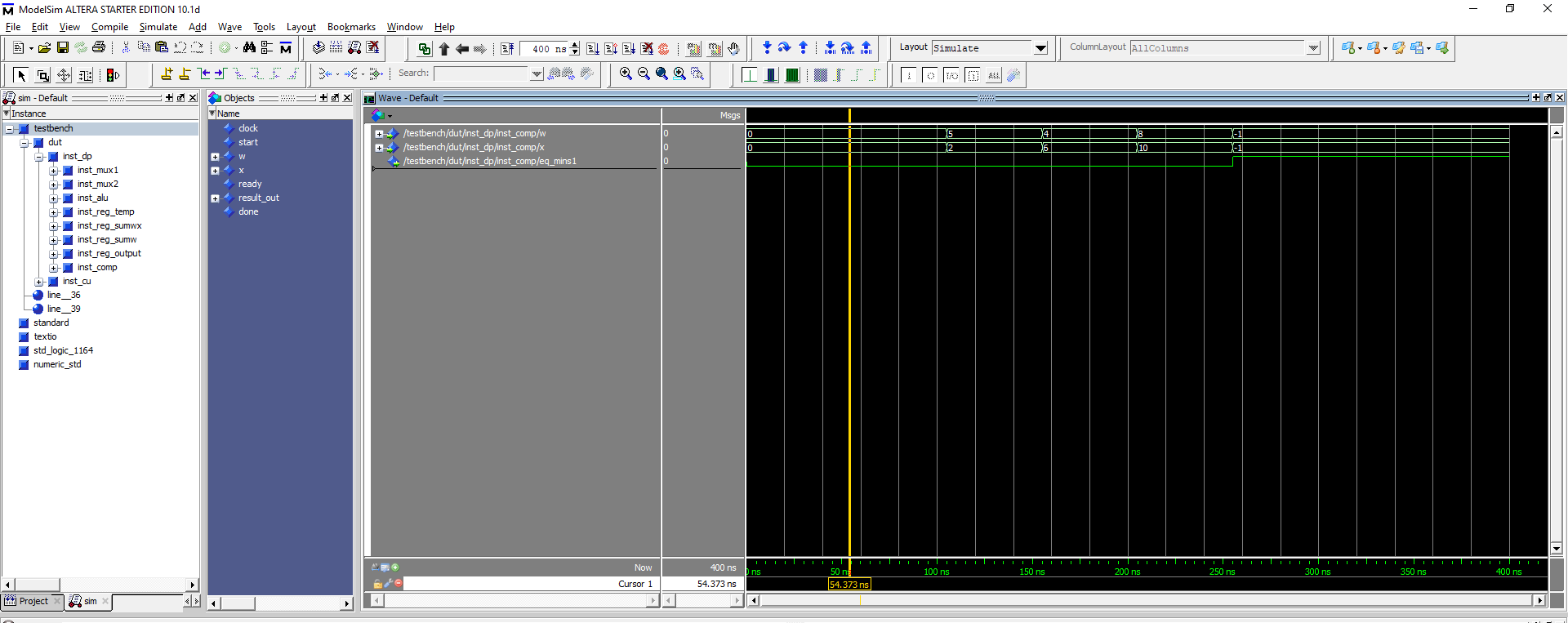
architecture arch\_comparator of comparator is

begin

    eq\_mins1 <= '1' when ((signed(x) = to\_signed(-1,x'length)) OR (signed(w) = to\_signed(-1,w'length))) else '0';

end  arch\_comparator;

## Simulation Screenshot:



# **ALU**

## **VHDL Code:**

library ieee;

use ieee.std\_logic\_1164.all;

USE ieee.numeric\_std.ALL;

entity alu is

port(

    a,b : in std\_logic\_vector(15 downto 0);

    opcode : in std\_logic\_vector(1 downto 0);

    res: out std\_logic\_vector(15 downto 0)

);

end;

architecture behave of alu is

signal tmp : integer;

begin

P1: process(opcode, a, b)

 begin

  C1: case opcode is

      when "00" => tmp <= to\_integer(signed(a))+to\_integer(signed(b)) after 5 ps;

      when "01" => tmp <= to\_integer(signed(a))\*to\_integer(signed(b)) after 7 ps;

      when "10" => tmp <= to\_integer(signed(a))/to\_integer(signed(b)) after 10 ps;

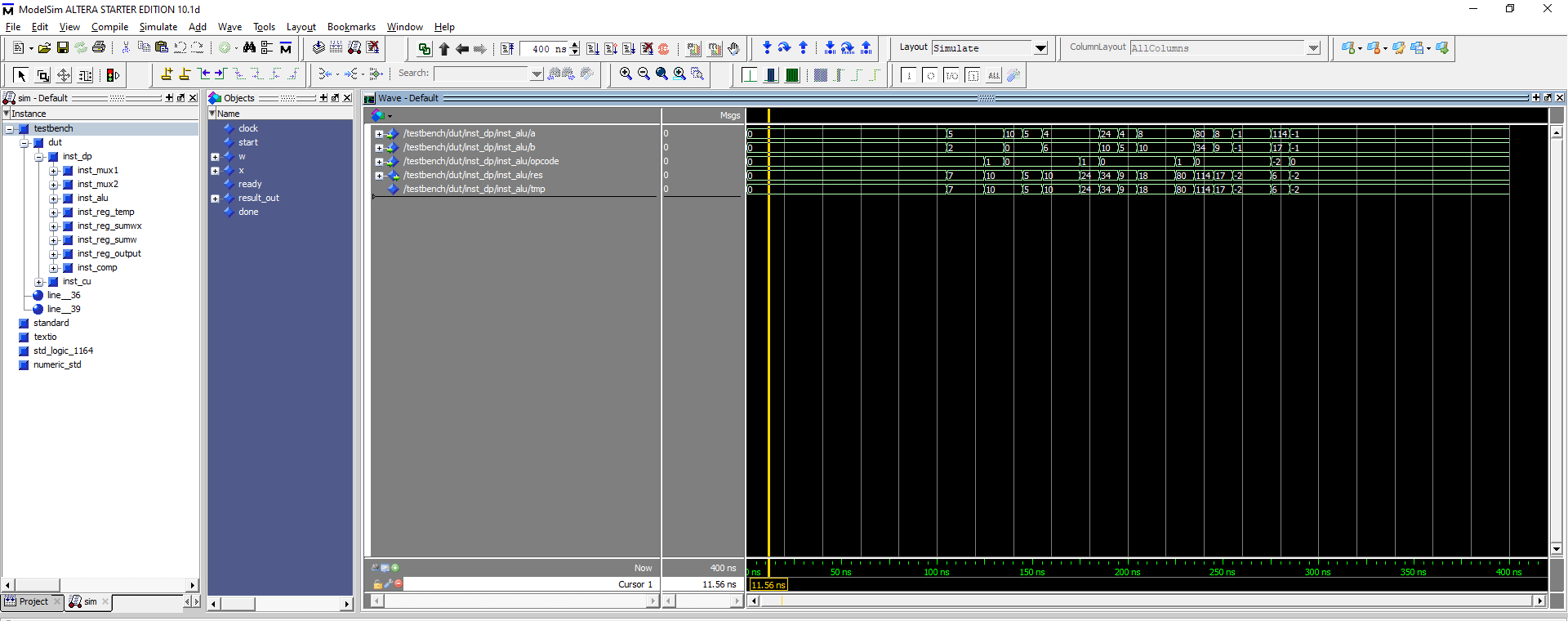
      when others => tmp <= 0;

  end case C1;

 end process;

res <= std\_logic\_vector(to\_signed(tmp, res'length));

end behave;



# **Data Path**

**הסבר:**

יחידות המרכיבות את מסלול הנתונים.

## **VHDL Code:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity datapath is

    port(

        -- inputs

        clock: in std\_logic;

        start: in std\_logic;

        w: in std\_logic\_vector(15 downto 0);

        x: in std\_logic\_vector(15 downto 0);

        en\_reg\_temp: in std\_logic;

        en\_reg\_sumwx: in std\_logic;

        en\_reg\_sumw: in std\_logic;

        en\_result: in std\_logic;

        sel\_mux1: in std\_logic\_vector(1 downto 0);

        sel\_mux2: in std\_logic\_vector(1 downto 0);

        sel\_alu: in std\_logic\_vector(1 downto 0);

        reg\_reset: in std\_logic;

        -- outputs

        eq\_mins1: out std\_logic; -- stop condition

        result\_out: out std\_logic\_vector(15 downto 0)

    );

end datapath;

architecture arch\_datapath of datapath is

    -- Signals

    signal mux\_out1: std\_logic\_vector(15  downto 0);

    signal mux\_out2: std\_logic\_vector(15  downto 0);

    signal alu\_res: std\_logic\_vector(15 downto 0);

    signal temp: std\_logic\_vector(15 downto 0);

    signal sumwx: std\_logic\_vector(15 downto 0);

    signal sumw: std\_logic\_vector(15 downto 0);

begin

    -- First mux

    inst\_mux1: entity work.mux

        port map(

            mux\_in0 => w,

            mux\_in1 => temp,

            mux\_in2 => sumwx,

            mux\_in3 => sumw,

            mux\_sel => sel\_mux1,

            mux\_out => mux\_out1

        );

    --second mux

    inst\_mux2: entity work.mux

        port map(

            mux\_in0 => x,

            mux\_in1 => temp,

            mux\_in2 => sumwx,

            mux\_in3 => sumw,

            mux\_sel => sel\_mux2,

            mux\_out => mux\_out2

        );

    -- ALU

    inst\_alu: entity work.alu

        port map(

            a => mux\_out1,

            b => mux\_out2,

            opcode => sel\_alu,

            res => alu\_res

        );

    -- Registers

    inst\_reg\_temp: entity work.df\_f

        port map(

            clock => clock,

            reset => reg\_reset,

            enable => en\_reg\_temp,

            d => alu\_res,

            q => temp

        );

    inst\_reg\_sumwx: entity work.df\_f

        port map(

            clock => clock,

            reset => reg\_reset,

            enable => en\_reg\_sumwx,

            d => alu\_res,

            q => sumwx

        );

    inst\_reg\_sumw: entity work.df\_f

        port map(

            clock => clock,

            reset => reg\_reset,

            enable => en\_reg\_sumw,

            d => alu\_res,

            q => sumw

        );

    inst\_reg\_output: entity work.df\_f

        port map(

            clock => clock,

            reset => reg\_reset,

            enable => en\_result,

            d => alu\_res,

            q => result\_out

        );

    -- comparator

    inst\_comp: entity work.comparator

        port map(

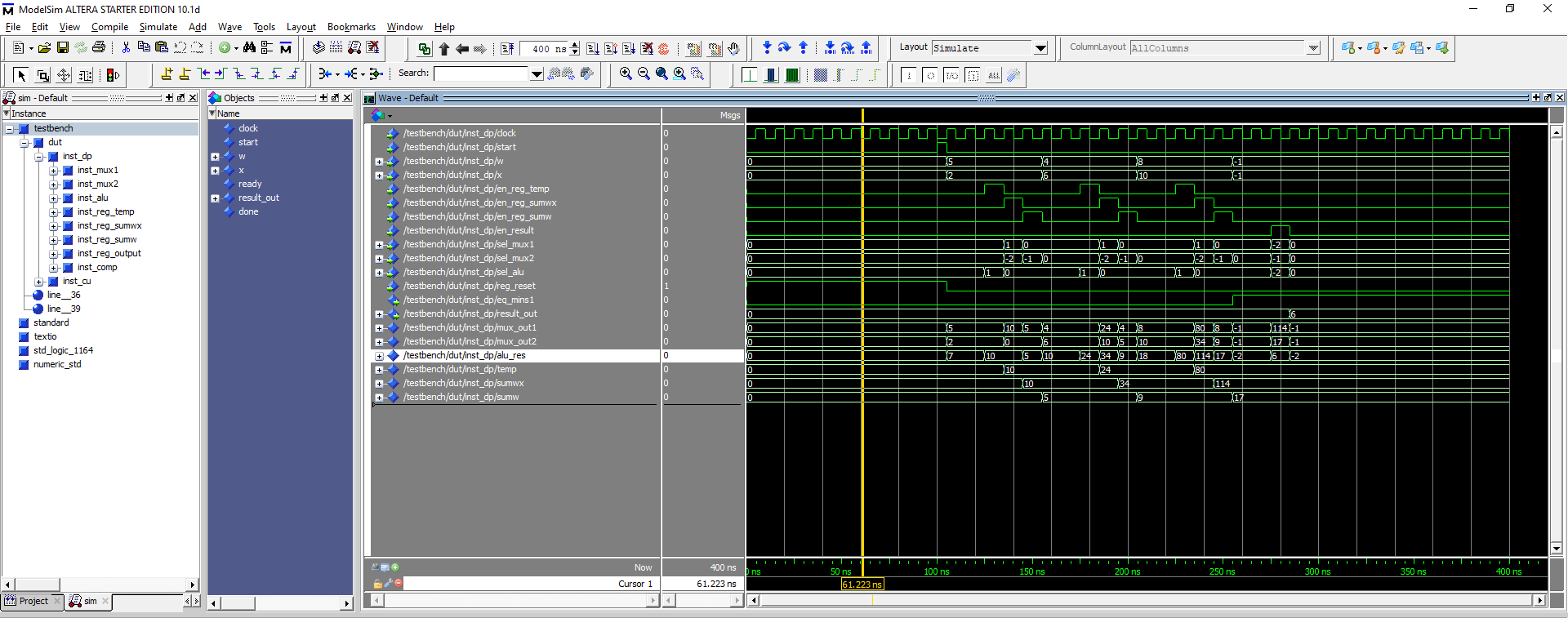
            w => w,

            x => x,

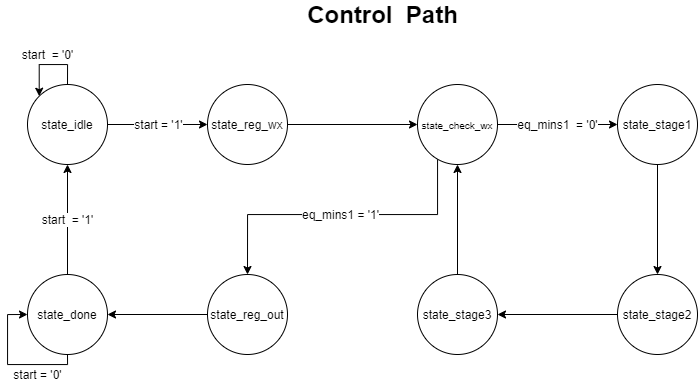
            eq\_mins1 => eq\_mins1

        );

end arch\_datapath;



# **Control Unit**:



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **state\_idle** | **state\_reg\_wx** | **state\_check\_wx** | **state\_stage1** | **state\_stage2** | **state\_stage3** | **state\_reg\_out** | **state\_done** |
| en\_reg\_temp | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| en\_reg\_sumwx | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| en\_reg\_sumw | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| en\_result | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| sel\_mux1 | 0 | 0 | 0 | 0 | 1 | 0 | 10 | 0 |
| sel\_mux2 | 0 | 0 | 0 | 0 | 10 | 11 | 11 | 0 |
| sel\_alu | 0 | 0 | 0 | 1 | 0 | 0 | 10 | 0 |
| reg\_reset | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| done | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ready | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

## **VHDL Code:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity cu is

    port(

        clock: in std\_logic;

        start: in std\_logic;

        eq\_mins1: in std\_logic; -- stop condition

        --outputs

        en\_reg\_temp: out std\_logic;

        en\_reg\_sumwx: out std\_logic;

        en\_reg\_sumw: out std\_logic;

        en\_result: out std\_logic;

        sel\_mux1: out std\_logic\_vector(1 downto 0);

        sel\_mux2: out std\_logic\_vector(1 downto 0);

        sel\_alu: out std\_logic\_vector(1 downto 0);

        reg\_reset: out std\_logic;

        ready: out std\_logic;

        done: out std\_logic

    );

end cu;

architecture arch\_cu of cu is

    -- define FSM

    type states is(

        state\_idle, -- wait for start

        state\_reg\_wx, -- input w and x

        state\_check\_wx, -- check if w or x is -1 ?

        state\_stage1, -- if not stopping condition -- then temp = w \* x

        state\_stage2, -- wx = wx + temp

        state\_stage3, -- w = w + in\_w

        state\_reg\_out, -- state to register output

        state\_done -- output is ready

    );

    signal p\_state: states:= state\_idle;

begin

    process(clock,start,eq\_mins1) begin

        if rising\_edge(clock) then

            case(p\_state) is

                when state\_idle =>

                    if(start = '1') then

                        p\_state <= state\_reg\_wx;

                    end if;

                when state\_reg\_wx =>

                    p\_state <= state\_check\_wx;

                when state\_check\_wx =>

                    if(eq\_mins1 = '1') then

                        p\_state <= state\_reg\_out;

                    else

                        p\_state <= state\_stage1;

                    end if;

                when state\_stage1 =>

                    p\_state <= state\_stage2;

                when state\_stage2 =>

                    p\_state <= state\_stage3;

                when state\_stage3 =>

                    p\_state <= state\_reg\_wx;

                when state\_reg\_out =>

                    p\_state <= state\_done;

                when state\_done =>

                    if(start = '1') then

                        p\_state <= state\_idle;

                    end if;

                when others =>

                    p\_state <= state\_idle;

            end case;

        end if;

    end process;

    --outputs

    process(p\_state) begin

        case (p\_state) is

            when state\_idle =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '0';

                en\_result  <= '0';

                sel\_mux1 <= "00";

                sel\_mux2 <= "00";

                sel\_alu <= "00";

                reg\_reset <= '1';

                done <= '0';

                ready <= '0';

            when state\_reg\_wx =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '0';

                en\_result  <= '0';

                sel\_mux1 <= "00";

                sel\_mux2 <= "00";

                sel\_alu <= "00";

                reg\_reset <= '0';

                done <= '0';

                ready <= '1';

            when state\_check\_wx =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '0';

                en\_result  <= '0';

                sel\_mux1 <= "00";

                sel\_mux2 <= "00";

                sel\_alu <= "00";

                reg\_reset <= '0';

                done <= '0';

                ready <= '0';

            when state\_stage1 =>

                en\_reg\_temp <= '1';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '0';

                en\_result  <= '0';

                sel\_mux1 <= "00";

                sel\_mux2 <= "00";

                sel\_alu <= "01";

                reg\_reset <= '0';

                done <= '0';

                ready <= '0';

            when state\_stage2 =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '1';

                en\_reg\_sumw <= '0';

                en\_result  <= '0';

                sel\_mux1 <= "01";

                sel\_mux2 <= "10";

                sel\_alu <= "00";

                reg\_reset <= '0';

                done <= '0';

                ready <= '0';

            when state\_stage3 =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '1';

                en\_result  <= '0';

                sel\_mux1 <= "00";

                sel\_mux2 <= "11";

                sel\_alu <= "00";

                reg\_reset <= '0';

                done <= '0';

                ready <= '0';

            when state\_reg\_out =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '0';

                en\_result  <= '1';

                sel\_mux1 <= "10";

                sel\_mux2 <= "11";

                sel\_alu <= "10";

                reg\_reset <= '0';

                done <= '0';

                ready <= '0';

            when state\_done =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '0';

                en\_result  <= '0';

                sel\_mux1 <= "00";

                sel\_mux2 <= "00";

                sel\_alu <= "00";

                reg\_reset <= '0';

                done <= '1';

                ready <= '0';

            when others =>

                en\_reg\_temp <= '0';

                en\_reg\_sumwx <= '0';

                en\_reg\_sumw <= '0';

                en\_result  <= '0';

                sel\_mux1 <= "00";

                sel\_mux2 <= "00";

                sel\_alu <= "00";

                reg\_reset <= '0';

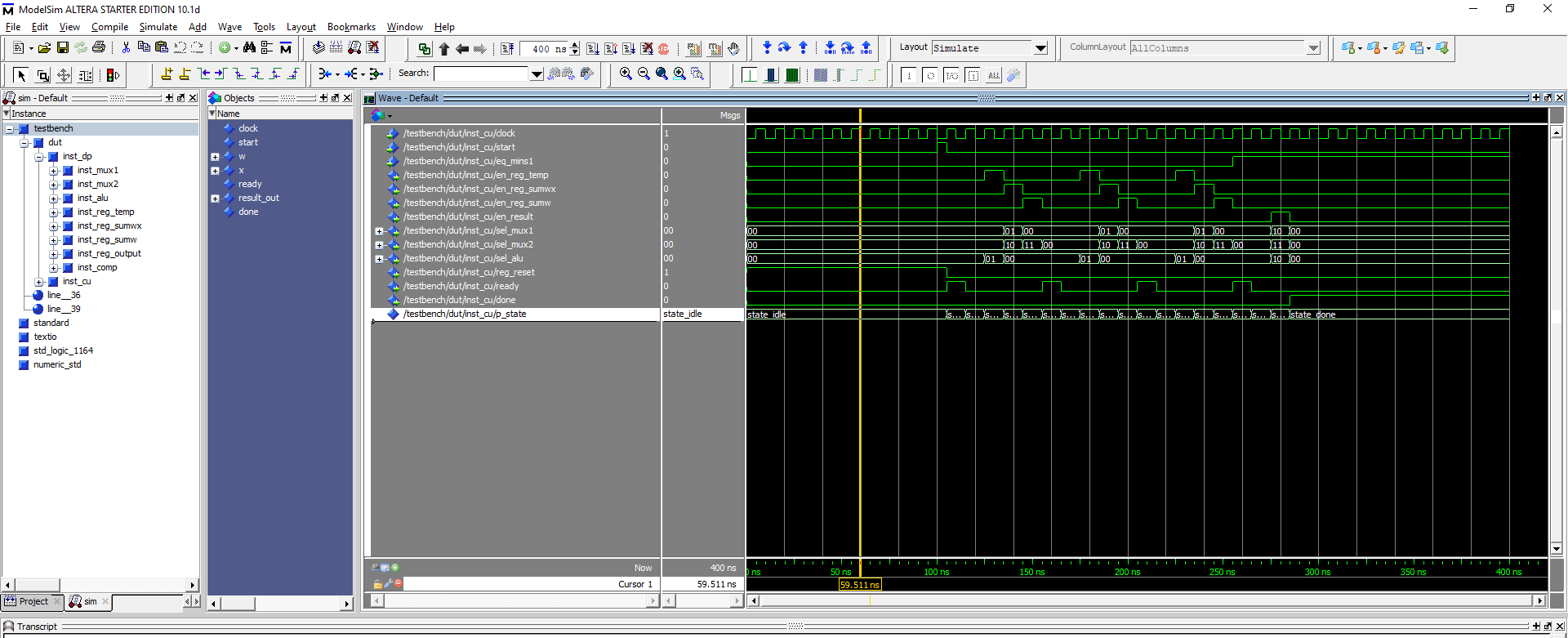
                done <= '0';

                ready <= '0';

        end case;

    end process;

end arch\_cu;



# **Final Plan**

## **VHDL Code:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity top is

    port (

        --inputs

        clock: in std\_logic;

        start: in std\_logic;

        w: in std\_logic\_vector(15 downto 0);

        x: in std\_logic\_vector(15 downto 0);

        --outputs

        ready: out std\_logic;

        result\_out: out std\_logic\_vector(15 downto 0);

        done: out std\_logic

    );

end top;

architecture arch\_top of top is

    -- Intermediate Signals

    signal eq\_mins1: std\_logic;

    signal en\_regw: std\_logic;

    signal en\_regx: std\_logic;

    signal en\_reg\_temp: std\_logic;

    signal en\_reg\_sumwx: std\_logic;

    signal en\_reg\_sumw: std\_logic;

    signal en\_result: std\_logic;

    signal sel\_mux1: std\_logic\_vector(1 downto 0);

    signal sel\_mux2: std\_logic\_vector(1 downto 0);

    signal sel\_alu: std\_logic\_vector(1 downto 0);

    signal reg\_reset: std\_logic;

begin

    -- DataPath

    inst\_dp: entity work.datapath

        port map(

            clock => clock,

            start => start,

            w => w,

            x => x,

            eq\_mins1 => eq\_mins1,

            en\_reg\_temp => en\_reg\_temp,

            en\_reg\_sumwx => en\_reg\_sumwx,

            en\_reg\_sumw => en\_reg\_sumw,

            en\_result => en\_result,

            sel\_mux1 => sel\_mux1,

            sel\_mux2 => sel\_mux2,

            sel\_alu => sel\_alu,

            reg\_reset => reg\_reset,

            result\_out => result\_out

        );

    -- Control Unit

    inst\_cu: entity work.cu

        port map(

            clock => clock,

            start => start,

            eq\_mins1 => eq\_mins1,

            en\_reg\_temp => en\_reg\_temp,

            en\_reg\_sumwx => en\_reg\_sumwx,

            en\_reg\_sumw => en\_reg\_sumw,

            en\_result => en\_result,

            sel\_mux1 => sel\_mux1,

            sel\_mux2 => sel\_mux2,

            sel\_alu => sel\_alu,

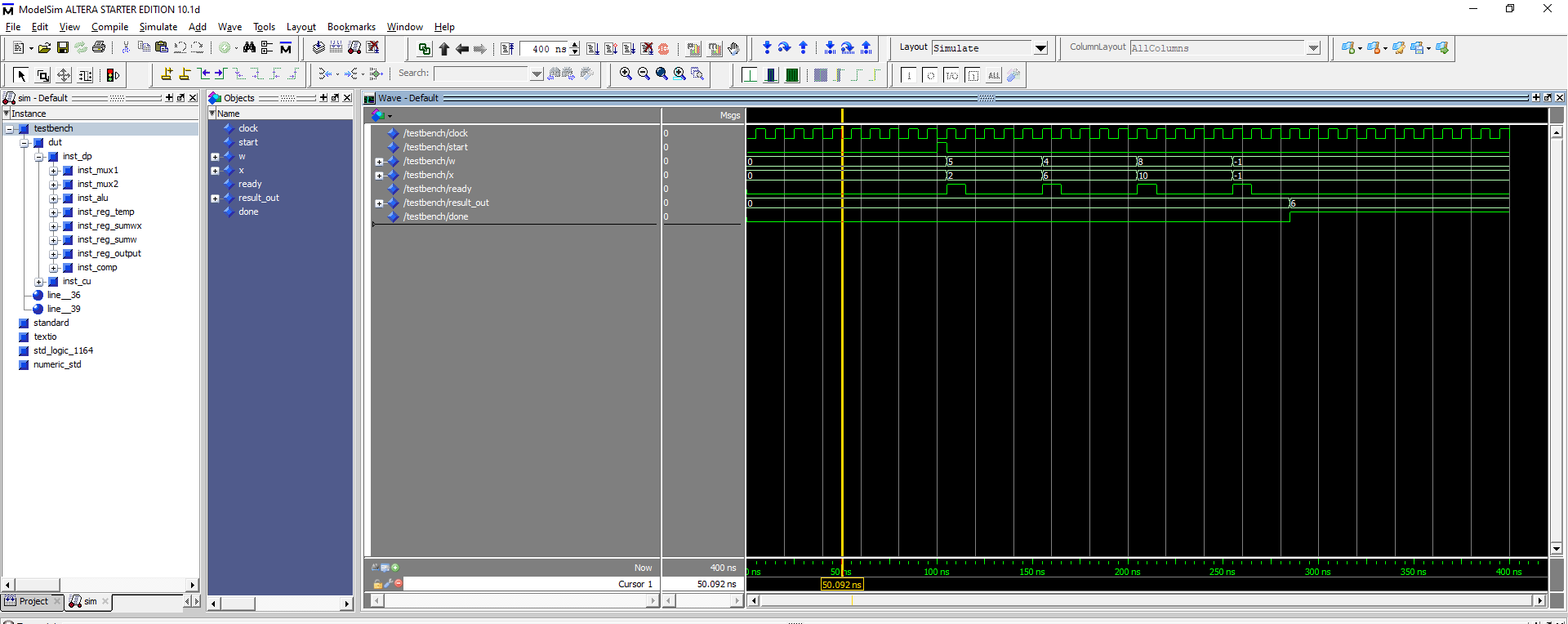
            reg\_reset => reg\_reset,

            ready => ready,

            done => done

        );

end arch\_top;



# **TESTBENCH**

# VHDL CODE:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity testbench is

end testbench;

architecture arch\_testbench of testbench is

    --intermediate signals

    --inputs

    signal clock: std\_logic:= '0';

    signal start: std\_logic:= '0';

    signal w: std\_logic\_vector(15 downto 0):= (others => '0');

    signal x: std\_logic\_vector(15 downto 0):= (others => '0');

    --outputs

    signal ready: std\_logic;

    signal result\_out: std\_logic\_vector(15 downto 0);

    signal done: std\_logic;

begin

    --design under test

    dut: entity work.top

        port map(

        --inputs

        clock => clock,

        start => start,

        w => w,

        x => x,

        --outputs

        ready => ready,

        result\_out => result\_out,

        done => done

        );

    --100 Mhz clock generator

        clock <= NOT clock after 5 ns;

    --testing process

    process begin

        wait for 100 ns;

        start <= '1';

        wait until (ready = '1');

        start <= '0';

        -- first course

        x <= x"0002";

        w <= x"0005";

        wait until (ready = '1');

        -- Second course

        x <= x"0006";

        w <= x"0004";

        wait until (ready = '1');

        -- Third course

        x <= x"000a";

        w <= x"0008";

        wait until (ready = '1');

        -- stop condition

        x <= std\_logic\_vector(to\_signed(-1,x'length));

        w <= std\_logic\_vector(to\_signed(-1,x'length));

        wait until (done = '1');

        wait for 100 ns;

        wait;

    end process;

end arch\_testbench;