دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه نهایی

طراحی کمک‌پردازنده

تهیه کنندگان:

سید عرفان نوربخش

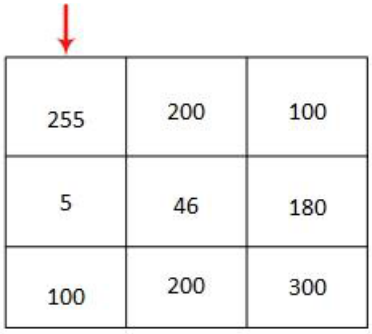
سید امین حسینی

استاد راهنما: دکتر فاطمه کاظمی

خرداد ۱۴۰۱

دستور پروژه:

مداری را طراحی کنید که ضرب دو ماتریس ۳×۳ را به صورت ستون به ستون درایه به درایه انجام دهد و نتایج را با هم جمع کند (طبق شکل زیر) نهایتا یک مقدار بدست آورده و آن را در یک رجیستر ذخیره کند. این سخت‌افزار را برای ماتریس ۳ × ۳ بسازید.



255	200	100
5	46	180
100	200	300

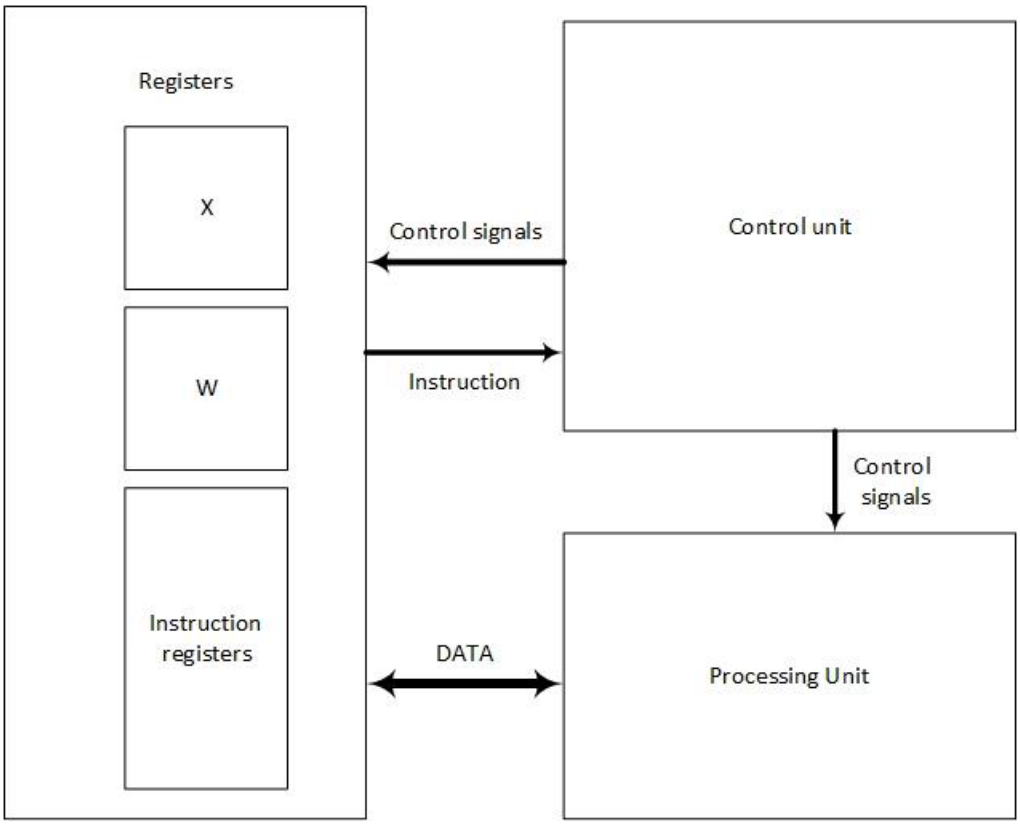
1	0	-1
1	0	-1
1	0	-1

255*1 + 5* 1 + 100 * 1+0 * 200 + 0* 46 + 0 * 200 + 100*-1 + -1* 180 + -1*300=-220

مدار مربوط به این ضرب شامل ۳ واحد اصلی می‌باشد که شامل: واحد register ها که خود شامل رجیسترهای داده (دو آرایه از نوع integer به ابعاد ۳×۳) و رجیسترهای دستورات (آرایه دوبعدی از نوع std_logic_vector(15 downto 0) به تعداد ۱۵ درایه) است. بخش رجیسترهای داده خود شامل دو آرایه ی ۳ × ۳ است که همین اعداد شکل بالا به صورت integer در آنها ذخیره شده اند و بخش رجیسترهای دستور یک آرایه از آرایه های ۱۶ بیتی از نوع std_logic_vector می باشد. اعداد درون این رجیسترها نشانگر دستورات می باشد. دستورات به ترتیب در جدول زیر تعریف شده است که می‌توانید از کد هگزادسیمال آن استفاده کنید. رجیستر دستورات و اعداد را بعد از تعریف به صورت دستی مقدار دهید. برای مقدار دهی رجیستر دستورات دستور اول load و بعد دستور mul/add به همین صورت یکی در میان دستور load و Mul/add گذاشته می شود تا دستور ۱۰اُم که دستور store است. زمانی که کنترلر این دستور را دید باید نتیجه را از واحد پردازش گرفته در یک رجیستر در واحد رجیستر ذخیره کند. همچنین کنترلر هنگامی که دستورات load را خواند باید به ترتیب گفته شده در شکل ۱ مقادیر را دانه به دانه خوانده و باهم ضرب کند و با نتیجه قبلی جمع کند. در نهایت با دستور store نتیجه آخر را ذخیره کند.

نام دستور	کد دستور
load	X(0001)
Mul/add	X(0002)
store	X(0003)

معماری مدار خواسته شده به شکل زیر می باشد. در این مدار نیاز به یک واحد پردازشگر دارید که بسته به دستورات بالا و نیاز خواسته شده شما باید داخل این مدار را تشخیص داده و ماژول های مناسب آن را بسازید. همچنین مدار کنترلی لازم دارید که با استفاده از آن داده ها را از رجیسترها واکشی کرده و براساس دستور fetch و decode شده باید اعمال لازم بر روی آن ها انجام شود. به عنوان مثال دستور اول را واکشی کرده بعد از دیکدکردن متوجه می شود دستور load است پس باید آدرس لازم برای خواندن داده ها از رجیسترهای داده را بر روی پورت آدرس رجیستر گذاشته تا آن داده ها به واحد پردازش برود و با خواندن دستور بعد و دیکد کردن آن متوجه می شود دستور ضرب و جمع است پس باید داده های قبلی را در واحد پردازشی ضرب و با نتیجه قبلی جمع نماید. پس باید سیگنال کنترلی لازم برای این اعمال را به واحد پردازشی بفرستد. برای طراحی واحد کنترلی باید از ماشین حالت استفاده شود. کلیه این مدارات با کلاک همگام می شوند و با سیگنال reset به صورت ناهمگام reset می شوند. توجه داشته باشید شکل زیر به هیچ وجه کامل نیست و شناسایی مدارات لازم داخل ماژولهای اصلی و سیگنال های کنترلی بر عهده خود شما می باشد.



توضیح پروژه:

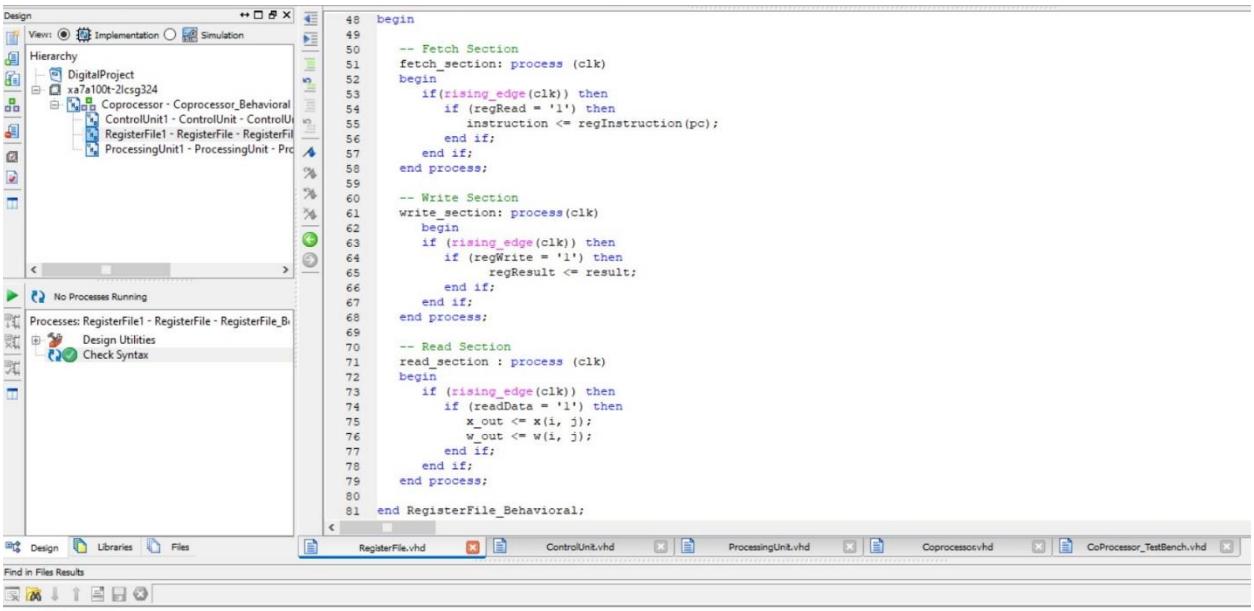
فایل RegisterFile:

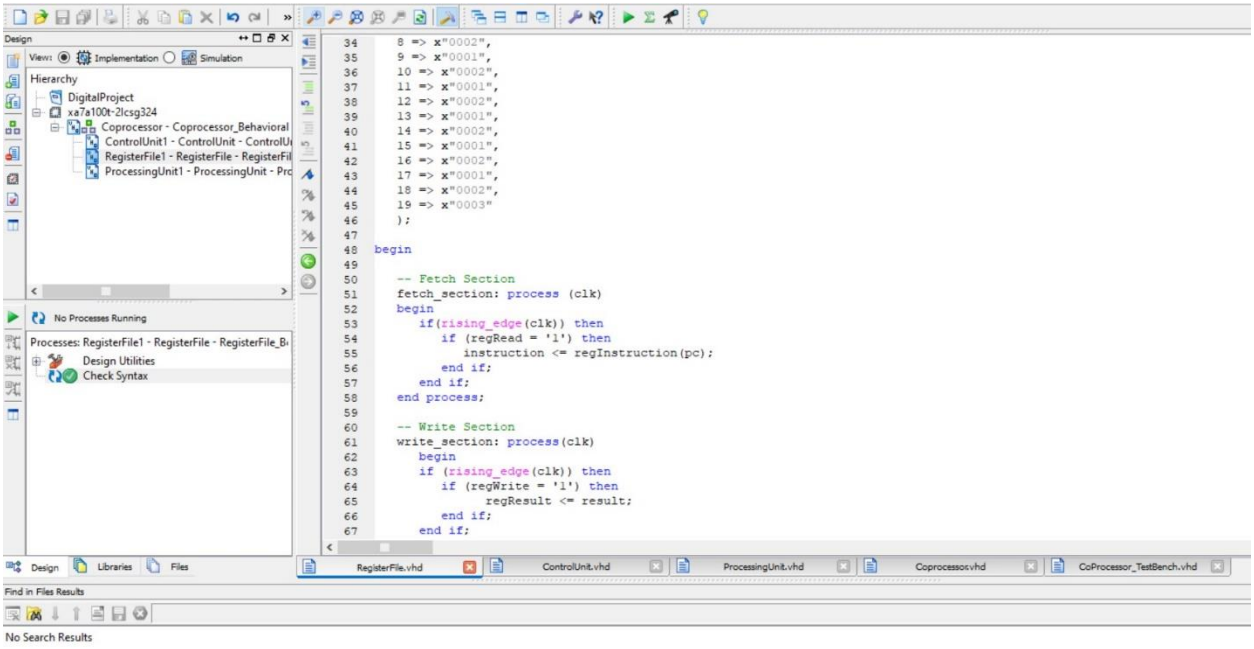
Entity RegisterFile:

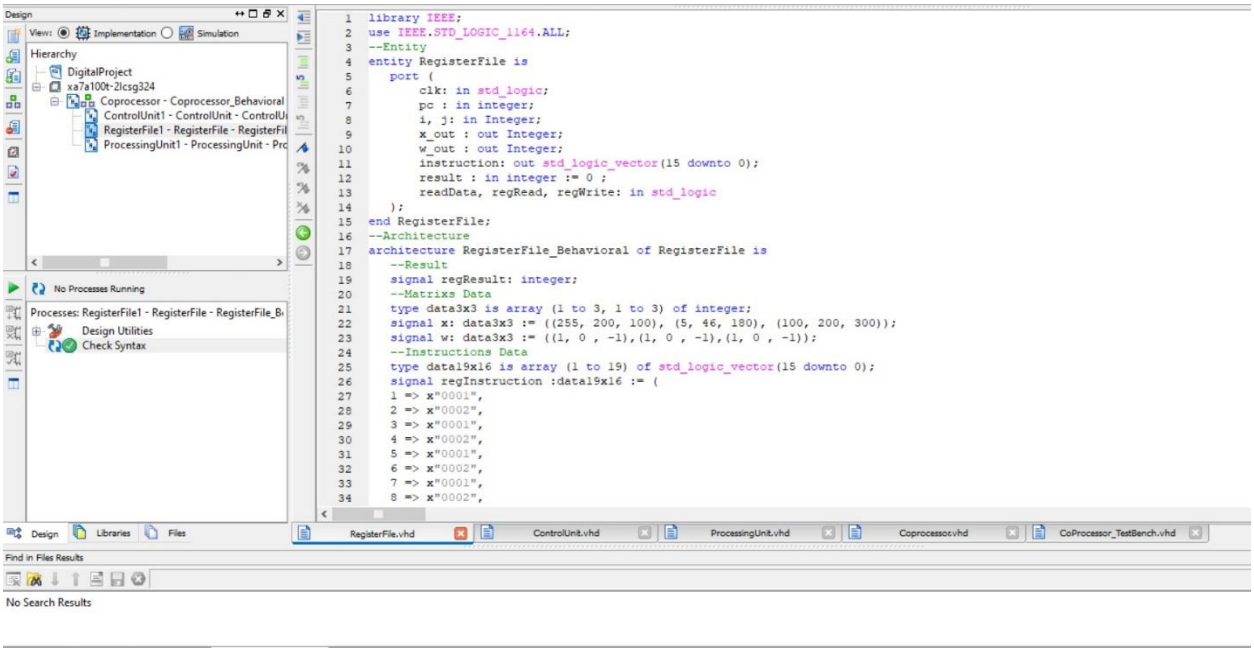
- clk: از جنس ورودی و برای پالس ساعت
- pc: از جنس ورودی و برای ذخیره‌ی **شماره‌ی** دستوری که باید انجام شود.
- i, j: از جنس ورودی و برای تعیین اندیس‌های ماتریس
- x_out: مقدارخانه‌ی مورد نظر از ماتریس x
- w_out: مقدار خانه‌ی مورد نظر از ماتریس w
- instruction: از جنس خروجی و برای ذخیره‌ی دستور فعلی. این پورت باید ۱۶ بیتی باشد.
- result: از جنس ورودی و دارای مقداری است که باید در رجیستر write (یا در فایل) ذخیره کنیم.
- readData, regRead, regWrite: از جنس ورودی و برای فعال بودن هر یک از پایه‌های واحد رجیستر

architecture RegisterFile:

- regResult: سیگنالی برای نگهداری result
- data3x3: یک type برای ایجاد ماتریس x و w
- ماتریس x و w از جنس data3x3 و حاوی مقادیر صورت پروژه
- data19x16: یک type برای ذخیره‌ی دستورات در رجیستر دستورات
- regInstruction: از جنس data19x16 برای ذخیره‌ی دستورات در رجیستر دستورات
- fetch section: این بخش یک فرآیند است که به پالس ساعت بالارونده حساس هست و در صورتی که پالس ساعت رخ بدهد و پایه‌ی regRead فعال باشد از رجیستر دستورات، دستور اندیس pc را می‌خواند و در instruction ذخیره می‌کند.
- write section: این بخش یک فرآیند است که به پالس ساعت بالارونده حساس هست و در صورتی که پالس ساعت رخ بدهد و پایه‌ی regwrite فعال باشد، result را در رجیستر regResult ذخیره می‌کند.
- read section: این بخش یک فرآیند است که به پالس ساعت بالارونده حساس هست و در صورتی که پالس ساعت رخ بدهد و پایه‌ی readData فعال باشد، عنصری از ماتریس x با اندیس i , j را در x_out ذخیره می‌کند. به همین ترتیب برای w_out نیز رخ می‌دهد.


Design
Views: Implementation Simulation
Hierarchy
DigitalProject
xa7a100t-2lcsg324
Coprocessor - Coprocessor_Behavioral
ControlUnit1 - ControlUnit - ControlU
RegisterFile1 - RegisterFile - RegisterFil
ProcessingUnit1 - ProcessingUnit - Pro
No Processes Running
Processes: RegisterFile1 - RegisterFile - RegisterFile_B
Design Utilities
Check Syntax
Design
Libraries
Files
48 begin
49
50 -- Fetch Section
51 fetch_section: process (clk)
52 begin
53 if(rising_edge(clk)) then
54 if (regRead = '1') then
55 instruction <= regInstruction(pc);
56 end if;
57 end if;
58 end process;
59
60 -- Write Section
61 write_section: process(clk)
62 begin
63 if (rising_edge(clk)) then
64 if (regWrite = '1') then
65 regResult <= result;
66 end if;
67 end if;
68 end process;
69
70 -- Read Section
71 read_section : process (clk)
72 begin
73 if (rising_edge(clk)) then
74 if (readData = '1') then
75 x_out <= x(i, j);
76 w_out <= w(i, j);
77 end if;
78 end if;
79 end process;
80
81 end RegisterFile_Behavioral;
RegisterFile.vhd
ControlUnit.vhd
ProcessingUnit.vhd
Coprocessor.vhd
CoProcessor_TestBench.vhd
Find in Files Results
No Search Results


Design
Views: Implementation Simulation
Hierarchy
DigitalProject
xa7a100t-2lcsg324
Coprocessor - Coprocessor_Behavioral
ControlUnit1 - ControlUnit - ControlU
RegisterFile1 - RegisterFile - RegisterFil
ProcessingUnit1 - ProcessingUnit - Pro
No Processes Running
Processes: RegisterFile1 - RegisterFile - RegisterFile_B
Design Utilities
Check Syntax
Design
Libraries
Files
34 8 => x"0002",
35 9 => x"0001",
36 10 => x"0002",
37 11 => x"0001",
38 12 => x"0002",
39 13 => x"0001",
40 14 => x"0002",
41 15 => x"0001",
42 16 => x"0002",
43 17 => x"0001",
44 18 => x"0002",
45 19 => x"0003"
46);
47
48 begin
49
50 -- Fetch Section
51 fetch_section: process (clk)
52 begin
53 if(rising_edge(clk)) then
54 if (regRead = '1') then
55 instruction <= regInstruction(pc);
56 end if;
57 end if;
58 end process;
59
60 -- Write Section
61 write_section: process(clk)
62 begin
63 if (rising_edge(clk)) then
64 if (regWrite = '1') then
65 regResult <= result;
66 end if;
67 end if;
RegisterFile.vhd
ControlUnit.vhd
ProcessingUnit.vhd
Coprocessor.vhd
CoProcessor_TestBench.vhd
Find in Files Results
No Search Results




```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  --Entity
4  entity RegisterFile is
5     port (
6          clk: in std_logic;
7          pc : in integer;
8          i, j: in Integer;
9          x_out : out Integer;
10         w_out : out Integer;
11         instruction: out std_logic_vector(15 downto 0);
12         result : in integer := 0 ;
13         readData, regRead, regWrite: in std_logic
14    );
15 end RegisterFile;
16 --Architecture
17 architecture RegisterFile_Behavioral of RegisterFile is
18    --Result
19    signal regResult: integer;
20    --Matrixs Data
21    type data3x3 is array (1 to 3, 1 to 3) of integer;
22    signal x: data3x3 := ((255, 200, 100), (5, 46, 180), (100, 200, 300));
23    signal w: data3x3 := ((1, 0 , -1),(1, 0 , -1),(1, 0 , -1));
24    --Instructions Data
25    type data19x16 is array (1 to 19) of std_logic_vector(15 downto 0);
26    signal regInstruction :data19x16 := (
27    1 => x"0001",
28    2 => x"0002",
29    3 => x"0001",
30    4 => x"0002",
31    5 => x"0001",
32    6 => x"0002",
33    7 => x"0001",
34    8 => x"0002",
```

فایل **ControlUnit**:

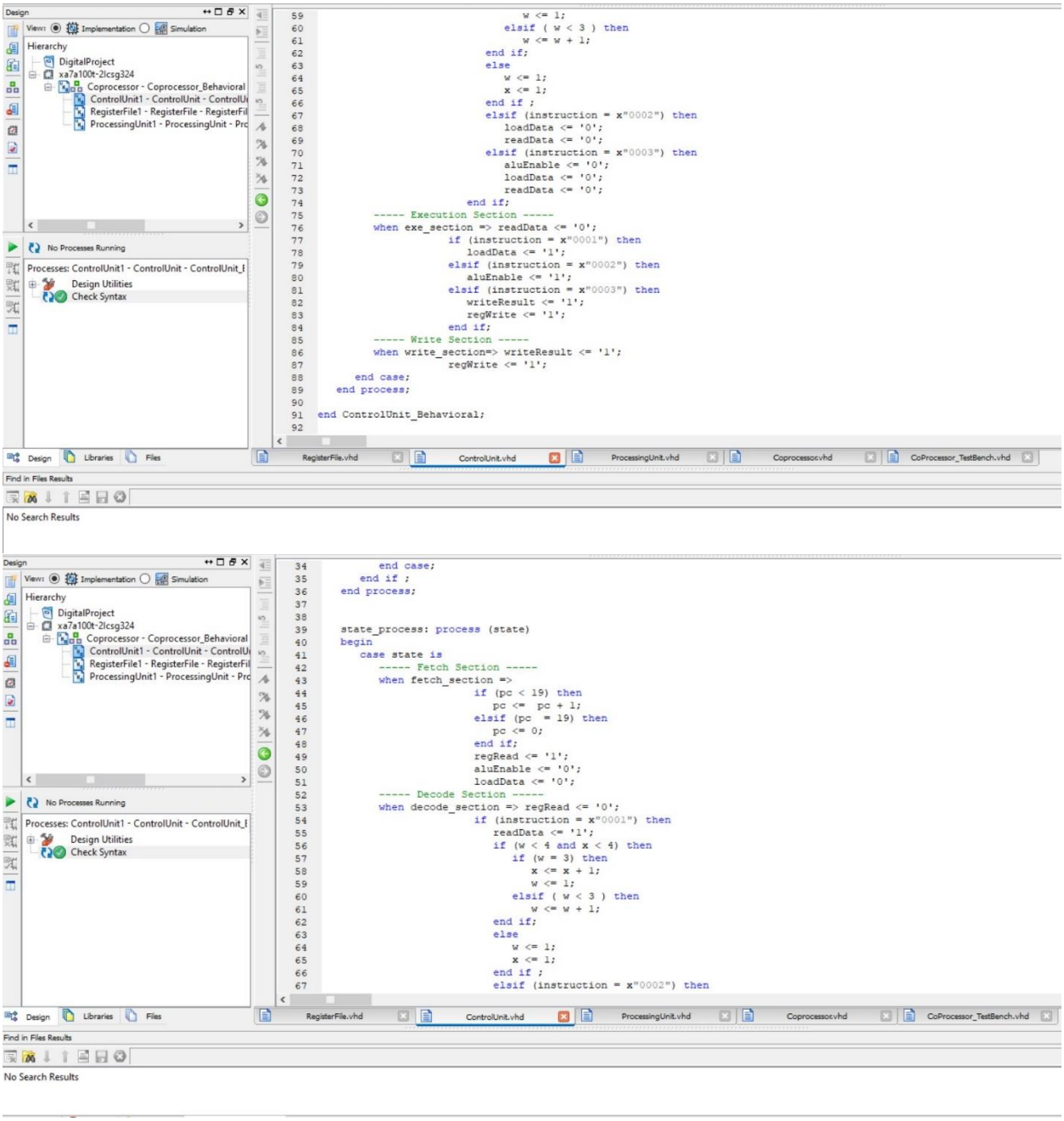
Entity ControlUnit:

- clk: از جنس ورودی و برای پالس ساعت
- pc: از جنس بافر و برای ذخیره‌ی **شماره‌ی** دستوری که باید انجام شود.
- x, w: از جنس بافر و برای اندیس سطر و ستون ماتریس استفاده می‌شوند.
- instruction: از جنس ورودی و برای گرفتن دستور فعلی. این پورت باید ۱۶ بیتی باشد.
- readData, regRead, regWrite, aluEnable, writeResult, loadData: از جنس خروجی و برای فعال کردن هر یک از پایه‌های واحد رجیستر یا واحد پراسس

architecture ControlUnit:

- stateTypes: یک type برای ایجاد سیگنال state که تعیین می‌کند ما در چه وضعیتی هستیم.
- state: یک سیگنال از جنس statetype برای تعیین وضعیت (خواندن، دیکد، اجرا یا نوشتن)
- clock_process: این بخش یک فرآیند است که به پالس ساعت بالارونده حساس هست و با هر بار پالس ساعت بالارونده باید تغییر وضعیت بدهد. در صورتی که پالس ساعت بالارونده رخ دهد و در وضعیت fetch باشیم به وضعیت decode انتقال پیدا می‌کنیم. در صورتی که در وضعیت decode باشیم به وضعیت execute انتقال پیدا می‌کنیم. در صورتی که در وضعیت execute باشیم و دستور x0003 باشد (دستور write) به وضعیت write منتقل می‌شویم.

- state_process: این بخش یک فرآیند است که به state حساس هست و در صورتی که state تغییر کرد بر اساس وضعیت، باید دستور مربوطه انجام شود:
  - fetch section: این قسمت برای خواندن دستورات است. در صورتی که به دستور آخر نرسیده باشیم به ترتیب جلو می‌رود و دستور بعدی را می‌خواند و در صورتی که به دستور آخر رسیده باشیم، به دستور اول باز می‌گردد. با توجه به دستور fetch، رجیسترهای کنترلی نیز تغییر می‌کنند.
  - decode section: این قسمت برای دیکد است. اگر دستور x0001 باشد باید خانه‌ی بعدی از ماتریس را بخوانیم. در صورتی که به آخر یک سطر رسیدیم، باید به ستون اول سطر بعدی برویم. در صورتی که به آخر ماتریس رسیدیم باید دوباره به ابتدای آن برگردیم. اگر دستور x0002 باشد باید دستور mul/add انجام شود و پایه‌های خواندن و واکشی باید غیر فعال شوند. اگر دستور x0003 باشد باید دستور store انجام شود و پایه‌های alu و واکشی و خواندن غیر فعال شوند.
  - Execution section: این قسمت برای اجرا است و باید پایه‌ی raedData را غیر فعال کند. اگر دستور x0001 باشد باید پایه‌ی واکشی فعال شود. در صورتی که دستور x0002 باشد باید پایه‌ی alu فعال شود و در صورتی که دستور x0003 باشد باید پایه‌ی writeResult و regWrite فعال شود.
  - Wrire section: این قسمت برای نوشتن استفاده می‌شود. پایه‌ی writeResult و regwrite را فعال می‌کند.



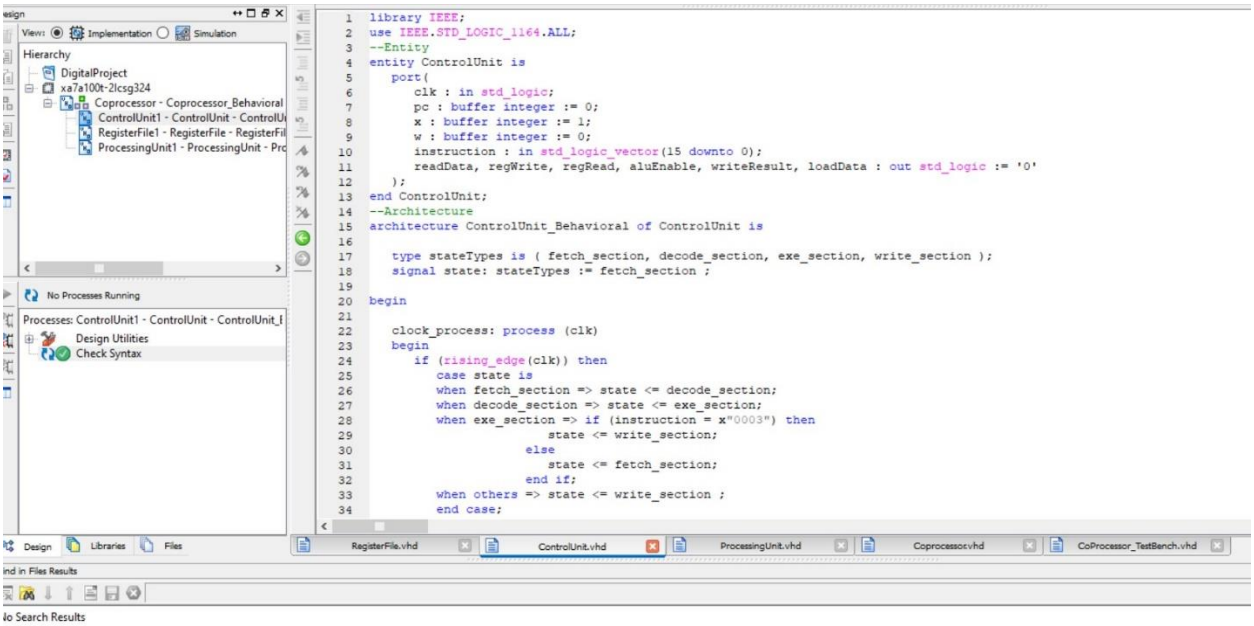

```
                              w <= 1;
                            elsif ( w < 3 ) then
                              w <= w + 1;
                         end if;
                         else
                            w <= 1;
                            x <= 1;
                         end if ;
                         elsif (instruction = x"0002") then
                            loadData <= '0';
                            readData <= '0';
                         elsif (instruction = x"0003") then
                            aluEnable <= '0';
                            loadData <= '0';
                            readData <= '0';
                      end if;
         ----- Execution Section -----
         when exe_section => readData <= '0';
                      if (instruction = x"0001") then
                         loadData <= '1';
                      elsif (instruction = x"0002") then
                         aluEnable <= '1';
                      elsif (instruction = x"0003") then
                         writeResult <= '1';
                         regWrite <= '1';
                      end if;
         ----- Write Section -----
         when write_section=> writeResult <= '1';
                      regWrite <= '1';
      end case;
   end process;

end ControlUnit_Behavioral;
```



```
         end case;
      end if ;
   end process;


   state_process: process (state)
   begin
      case state is
         ----- Fetch Section -----
         when fetch_section =>
                      if (pc < 19) then
                         pc <=  pc + 1;
                      elsif (pc  = 19) then
                         pc <= 0;
                      end if;
                      regRead <= '1';
                      aluEnable <= '0';
                      loadData <= '0';
         ----- Decode Section -----
         when decode_section => regRead <= '0';
                      if (instruction = x"0001") then
                         readData <= '1';
                         if (w < 4 and x < 4) then
                            if (w = 3) then
                               x <= x + 1;
                               w <= 1;
                            elsif ( w < 3 ) then
                               w <= w + 1;
                         end if;
                         else
                            w <= 1;
                            x <= 1;
                         end if ;
                         elsif (instruction = x"0002") then
```




```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Entity
entity ControlUnit is
   port(
      clk : in std_logic;
      pc : buffer integer := 0;
      x : buffer integer := 1;
      w : buffer integer := 0;
      instruction : in std_logic_vector(15 downto 0);
      readData, regWrite, regRead, aluEnable, writeResult, loadData : out std_logic := '0'
   );
end ControlUnit;
--Architecture
architecture ControlUnit_Behavioral of ControlUnit is

   type stateTypes is ( fetch_section, decode_section, exe_section, write_section );
   signal state: stateTypes := fetch_section ;

begin

   clock_process: process (clk)
   begin
      if (rising_edge(clk)) then
         case state is
         when fetch_section => state <= decode_section;
         when decode_section => state <= exe_section;
         when exe_section => if (instruction = x"0003") then
                     state <= write_section;
                  else
                     state <= fetch_section;
                  end if;
         when others => state <= write_section ;
         end case;
```

فایل **ProcessingUnit**:

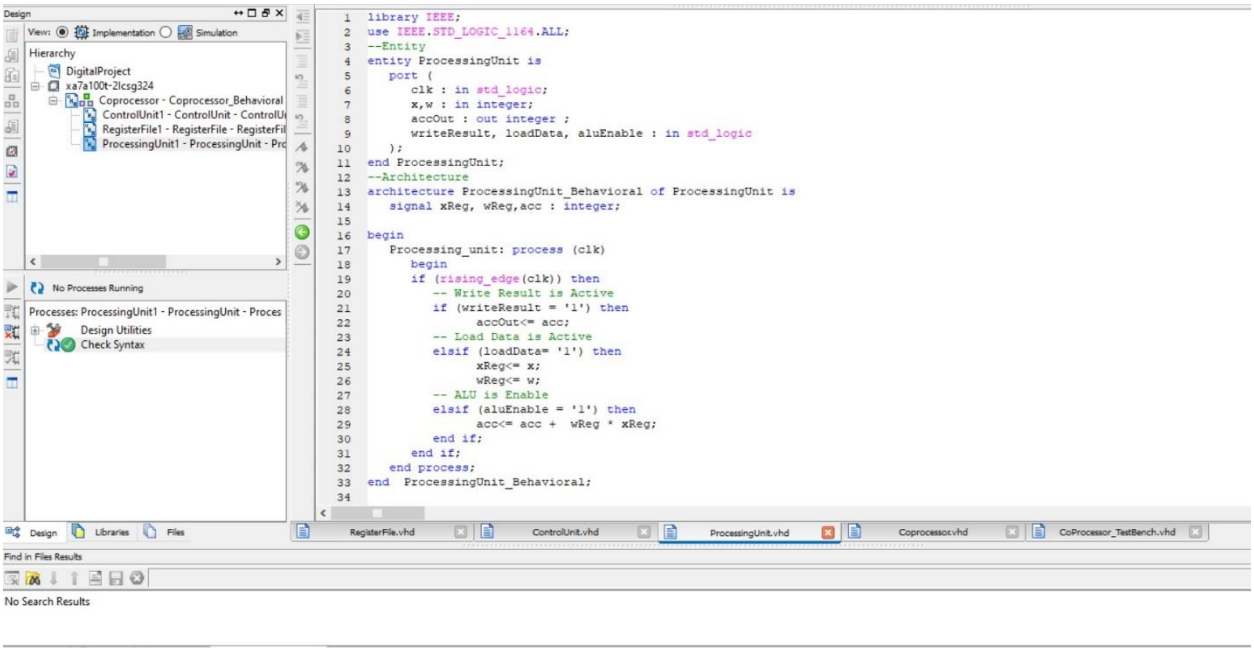
Entity ProcessingUnit:

- clk: از جنس ورودی و برای پالس ساعت
- x, w: از جنس ورودی و برای ذخیره‌ی مقادیر ماتریس x و w
- accOut: خروجی مجموع ضرب درایه‌های ماتریس x و w
- WriteResult, loadData, aluEnable: از جنس ورودی و برای تشخیص فعال بودن هر یک از پایه‌ها است.

architecture ProcessingUnit:

- xReg, wReg: سیگنالی برای ذخیره‌ی درایه‌های ماتریس که در ادامه ضرب و جمع مورد نظر را روی آن‌ها انجام دهیم.
- acc: یک سیگنال برای ذخیره‌ی مجموع ضرب درایه‌های دو ماتریس
- Processing_unit: این بخش یک فرآیند است که به پالس ساعت بالارونده حساس هست و با هر بار پالس ساعت بالارونده با توجه به پایه‌ی فعال علملیات مورد نظر را انجام می‌دهد.
  - اگر پایه‌ی writeResult فعال باشد، مجموع ضرب حساب شده را در accOut ذخیره می‌کند.
  - اگر پایه‌ی loadData فعال باشد، درایه‌های x و w را در دو سیگنال xReg و wReg ذخیره می‌کند تا ادامه عملیات ضرب و جمع را انجام دهد.

- اگر پایه‌ی aluEnable فعال باشد، ضرب دو درایه را انجام می‌دهد و به مجموع قبلی اضافه می‌کند.



فایل **Coprocessor**:

Entity Coprocessor:

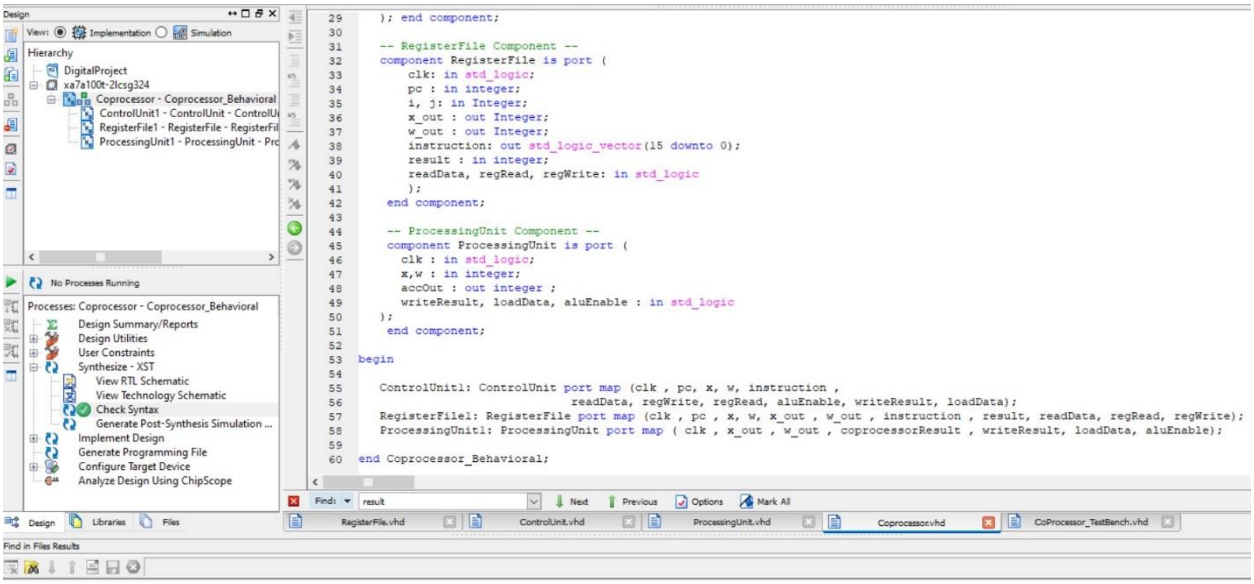
- clk: از جنس ورودی و برای پالس ساعت
- coprocessorResult: از جنس خروجی و برای نشان دادن خروجی نهایی

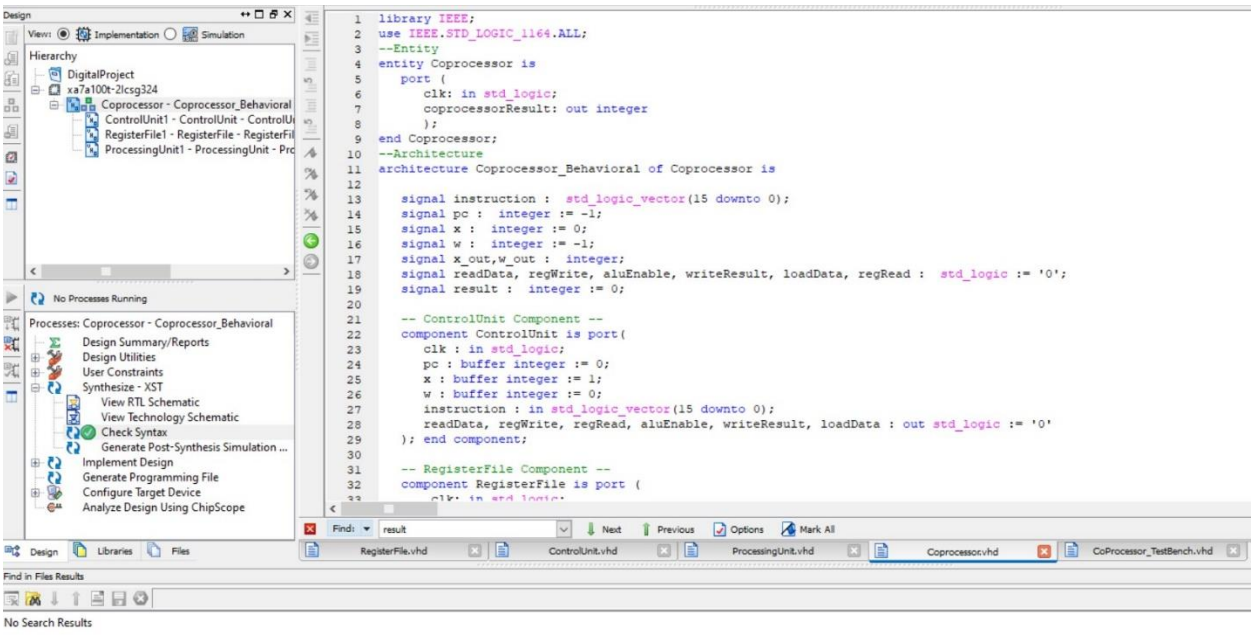
architecture Coprocessor:

- instruction: سیگنالی برای ذخیره‌ی دستور فعلی. این پورت باید ۱۶ بیتی باشد.
- pc: یک سیگنال برای برای ذخیره‌ی **شماره‌ی** دستوری که باید انجام شود.
- x, w: سیگنالی برای اندیس سطر و ستون ماتریس استفاده می‌شوند.
- x_out: سیگنالی برای ذخیره‌ی مقدار خانه‌ی مورد نظر از ماتریس x
- w_out: سیگنالی برای ذخیره‌ی مقدار خانه‌ی مورد نظر از ماتریس w
- readData, regRead, regWrite, aluEnable, writeResult, loadData: سیگنالی برای فعال کردن هر یک از پایه‌های واحد رجیستر یا واحد پراسس

- result: سیگنالی که دارای مقداری است که باید در رجیستر write (یا در فایل) ذخیره کنیم.

سه کامپوننت ساخته شده‌ی قبلی یعنی RegisterFile, ControlUnit, ProcessingUnit را به این ماژول اضافه می‌کنیم. بر اساس ورودی ماژول که در قسمت تست بنچ مشخص شده است و مقادیر اولیه‌ی سیگنال‌ها، کامپوننت‌ها اجرا می‌شوند و خروجی کل پروژه در coprocessorResult ذخیره می‌شود.

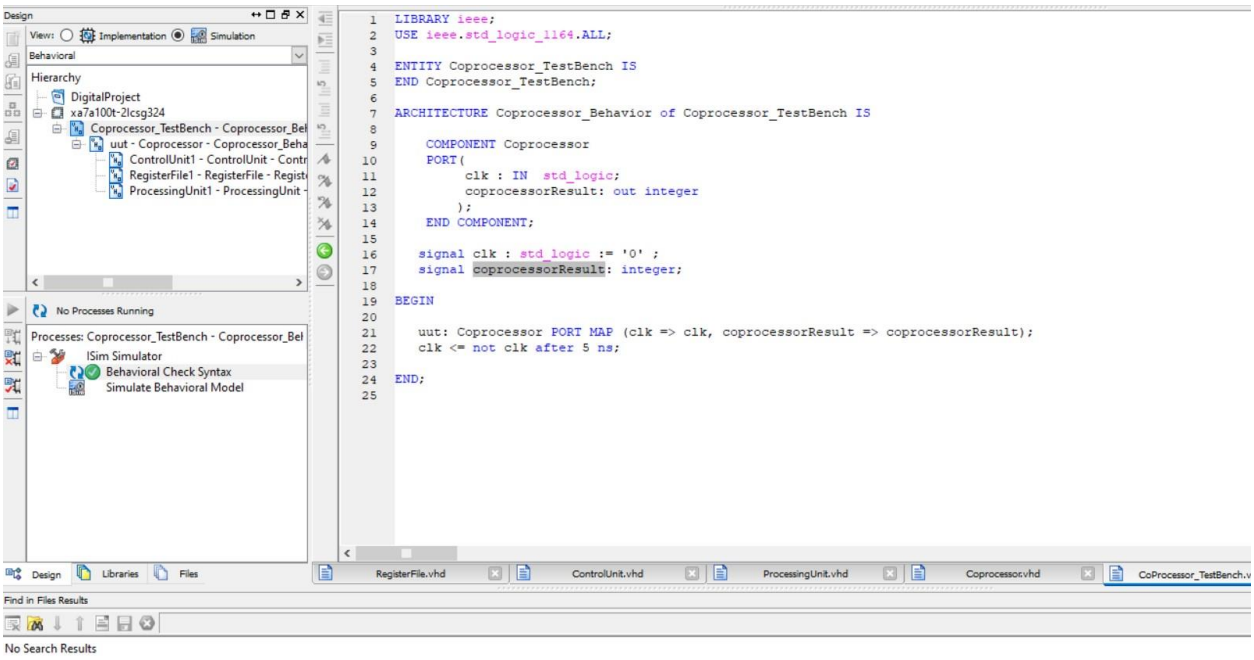




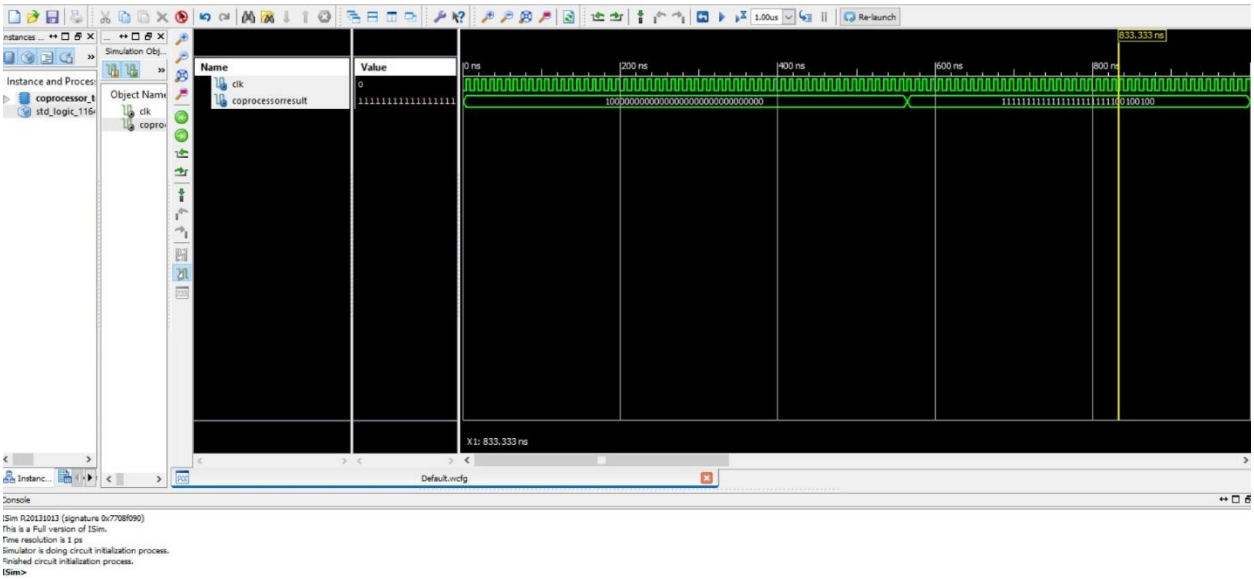
فایل **Coprocessor_TestBench**:

architecture Coprocessor:

کامپوننت Coprocessor را به تست بنچ اضافه می‌کنیم و به کمک سیگنال clk که برای پالس ساعت است، هر ۵ نانو ثانیه، کلاک را تغییر می‌دهیم. سیگنال coprocessorResult هم خروجی نهایی را نمایش می‌دهد.



خروجی نهایی شبیه‌ساز:


Name
clk
coprocessorresult
Value
0
1111111111111111
0 ns
200 ns
400 ns
600 ns
800 ns
833.333 ns
X1: 833.333 ns
Default.wcfg
Console
ISim R.20131013 (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>