

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

دستور کار آزمایشگاه معماری کامپیوتر

دکتر حمیدرضا زرندي
با همکاري مدرسین آزمایشگاه
زمستان ۱۳۹۶

بسمه تعالیٰ

درس معماري کامپیوتر، همنياز اين درس است، چنانچه دانشجویی چندين ترم پيش، اين درس را گذرانده است، لازم است مجدداً اين درس را بازنگري كرده و مطالب آن را به ياد آورد. همچنین، در اين آزمایشگاه، از زبان توصیف سختافزار VHDL برای توصیف، شبیه‌سازی، سنتز و در نهايیت پیاده‌سازی بر روی بردھای FPGA استفاده خواهد شد، در نتيجه لازم است دانشجویان نسبت به اين زبان توصیف يا زبانی مشابه همچون Verilog يا SystemC آشنا باشند. هر آزمایش باید در سطح تجزیید خواسته شده (گیت، RTL) طراحی و پیاده‌سازی شود.

منظور از توصیف در سطح گیت، توصیفی است که تنها از گیت‌های منطقی استفاده شده باشد.

Decoder, Encoder, MUX, DeMUX, Register, Bus, Buffer، توصیفی است که از RTL استفاده شده باشد.

تعداد آزمایش‌های که در طول ترم انجام می‌شود، در جلسه اول، توسط مدرس آزمایشگاه تعیین خواهد شد، لازم است که دانشجویان در قالب اعضای دو نفری (و در موارد خاص با تایید مدرس، سه نفری) تشکیل گروه داده و به مدرس آزمایشگاه اطلاع دهند.

توصیه می‌شود مدرس آزمایشگاه در ابتدای ترم یک ساختار استاندارد برای کامپیوتر پایه مورد نظر خود (عرض بیتی گذرگاه‌ها، تعداد و عرض بیتی ثبات‌ها، سایز حافظه‌ها، عملیات ALU) قرارداد نماید و بر مبنای آن آزمایشات مورد نظر خود را طراحی نماید. به عنوان مثال، در آزمایش جمع‌کننده‌ها و یا ضرب کننده‌ها، ورودی‌های ماثول بر مبنای عرض بیتی ثبات‌ها تنظیم شود.

زمان اتمام هر آزمایش، توسط مدرس آزمایشگاه با توجه به محتوای آزمایش مشخص خواهد شد و دانشجویان قبل از شروع هر آزمایش نسبت به مهلت انجام آن اطلاع کسب کنند.

هر گروه، قبل از شروع آزمایش لازم است، پیش‌گزارشی تهیه کرده و قبل از شروع کلاس تحويل مدرس آزمایشگاه دهد. مدرس آزمایشگاه قبل از شروع هر کلاس ممکن است پرسش‌های شفاهی یا کتبی نسبت به آزمایش مورد نظر مطرح نماید و دانشجویان موظف هستند که به آن‌ها پاسخ کامل و صحیح دهند.

از آنجایی که شروع کلاس‌های آزمایشگاه پس از زمان حذف و اضافه است، تعداد جلسات برگزار شده کمتر بوده و لذا حضور در کلیه جلسات الزامی است و تنها یک جلسه غیبت مجاز خواهد بود. همچنین از ورود افراد بیش از ۱۰ دقیقه تاخیر ممانعت به عمل خواهد آمد.

 نمره دهی نهایی بر اساس موارد زیر انجام خواهد شد؛ (مدرسین آزمایشگاه در صورت لزوم می‌توانند تغییراتی ایجاد نمایند):

مجموع آزمایش‌ها حدود ۱۰ درصد	پیش‌گزارش‌های تحويل داده شده
مجموع آزمایش‌ها حدود ۱۵ درصد	نمره پرسش‌های شفاهی/کتبی قبل از شروع هر آزمایش
مجموع آزمایش‌ها حدود ۳۰ درصد	انجام کامل هر آزمایش
مجموع آزمایش‌ها حدود ۳۰ درصد	کیفیت انجام هر آزمایش و پیاده‌سازی آن
حدود ۱۵ درصد	حضور فعال، موثر در گروه همکاری با مدرس (کار کلاسی)
به انتخاب مدرس آزمایشگاه	موارد دیگر (با صلاحیت مدرس آزمایشگاه)

 انجام کلیه آزمایش‌ها برای هر نوع داده‌ای زیر امکان‌پذیر است، قبل از شروع این مجموعه آزمایش‌ها، مدرس انتخاب نوع نمایش اعداد را برای شما مشخص خواهد ساخت و کلیه دانشجویان موظف به انجام آزمایش‌ها براساس همان نوع نمایش اعداد هستند.

توضیحات	نوع نمایش اعداد
مجموعه اعداد طبیعی	Unsigned بی علامت
اعداد صحیح (مثبت و منفی) در منطق مکمل ۲	Two's Complement مکمل دو
علامت و اندازه عدد بصورت جداگانه ذخیره می‌گردد	Sign Magnitude اندازه علامت
اعداد صحیح (مثبت و منفی) در منطق مکمل ۱	One's Complement مکمل ۱
اعداد حقیقی (مثبت و منفی) متناهی با تعداد بیت اعشار ثابت	Fixed Point ممیز ثابت
اعداد حقیقی (مثبت و منفی) متناهی با تعداد بیت اعشار شناور	Floating Pint ممیز شناور

 شبیه‌سازی‌ها توسط یکی از دو شبیه‌ساز ISim و یا ModelSim انجام خواهد گرفت.

فهرست آزمایش‌ها

صفحه	عنوان آزمایش	موضوع	شماره آزمایش
۵	توصیف مدارهای پایه آزمایش‌ها و انجام شبیه‌سازی	VHDL مقدماتی	۱
۱۱	طراحی مدارهای اولیه (Comprator و Decoder, Encoder, Multiplexer) ۴ بیتی	مدارهای پایه	۲
۱۲	پیاده‌سازی Process‌ها و چگونگی پیاده‌سازی شمارندها	پیاده‌سازی مدار ترتیبی	۳
۱۵	پیاده‌سازی انواع جمع کننده‌ها و تحلیل سرعت آنها	جمع کننده‌ها	۴
۱۶	پیاده‌سازی انواع ضرب کننده ۴ بیتی	ضرب کننده‌ها	۵
۱۹	ایجاد تاخیر مشخص در مدارهای دیجیتال	تاخیر در مدارهای دیجیتال	۶
۲۱	طراحی حافظه RAM, ROM و CAM	حافظه‌ها	۷
۲۳	تقسیم کننده دودویی در منطق مکمل ۲	تقسیم کننده‌ها	۸
۲۵	پیاده‌سازی یک کامپیووتر پایه	کامپیووتر پایه	۹
۲۸	طراحی واحد کنترل سیم بندی شده (Hard wired Control Unit)	طراحی واحد کنترل	۱۰
۲۹	پیاده‌سازی خط لوله و بررسی تسریع آن	خط لوله	۱۱
۳۱	پیاده‌سازی ارسال/دریافت ناممگام در گذرگاه‌های یکطرفه و دوطرفه	ورودی/خروجی	۱۲
۳۴	پیاده‌سازی مکانیزم‌های داوری گذرگاه	ورودی/خروجی	۱۳
۳۷	پیاده‌سازی عملیات جمع یا تفریق واحد ممیز شناور	محاسبات ممیز شناور	۱۴
۳۸	اندازه‌گیری پارامترهای IPC, CPI, MIPS و DMIPS یک پردازنده نوعی	ارزیابی کارآیی	۱۵
۴۰	طراحی سلسله مراتب حافظه	حافظه‌ها	۱۶

فهرست ضمایم

صفحه	عنوان	شماره ضمیمه
۴۲	توصیف سیستم با VHDL	۱
۴۵	شبیه‌سازی با استفاده از ModelSim	۲
۵۴	نحوه کارکردن با بورد FPGA	۳

موضوع: VHDL مقدماتی

شماره آزمایش: ۱

عنوان:

هدف:

توصیف مدارهای پایه آزمایش‌ها و انجام شبیه‌سازی

آشنایی با نحوه مدل کردن مدارهای پایه آزمایش‌ها در زبان VHDL و معرفی ساختارهای همروند

آمادگی پیش از آزمایش:

شرح آزمایش:

توصیف سخت‌افزار در VHDL دارای دو بخش است: توصیف پورت‌های ورودی و خروجی و توصیف معماری سخت‌افزار. به عنوان مثال، توصیف گیت And در مثال زیر ذکر شده است:

توصیف گیت And

```
library IEEE;
use IEEE.std_logic_1164.all;

Entity and_gate is
Port(
    A, B: in std_logic;
    C : out std_logic
);
End entity and_gate;

Architecture behavioral of and_gate is
Begin
    C <= A and B;
End behavioral;
```

در توصیف سخت‌افزار می‌توان از عملگرهای and, or, xor, not استفاده کرد. این عملگرها مثل گیت‌های ساده منطقی عمل می‌کنند. عملگر مقداردهی به سیگنال $=>$ است. در مثال بالا std_logic می‌تواند یک مقدار منطقی داشته باشد. در صورتی که نیاز به تعریف پورتی بیش از یک بیت باشد، می‌توان از std_logic_vector استفاده کرد. به مثال زیر توجه نمایید:

توصیف گیت And به صورت برداری

```
library IEEE;
use IEEE.std_logic_1164.all;

Entity and_gate_vector is
Port(
    A, B: in std_logic_vector(3 downto 0);
    C : out std_logic_vector(3 downto 0)
);
End entity and_gate_vector;
```

```

Architecture behavioral of and_gate_vector is
  Signal internal_signal : std_logic_vector(3 downto 0);
Begin
  internal_signal <= A and B;
  C <= internal_signal;
End behavioral;

```

توجه شود که در مثال بالا خروجی ۴ بیتی است و هر بیت آن با AND دو مقدار متناظر در A و B بدست آمده است. علاوه بر آن به تعریف سیگنال در داخل architecture توجه شود. به چند روش می‌توان به سیگنال‌های از نوع std_logic_vector مقدار داد. در جدول زیر چند نمونه ذکر شده است:

std_logic_vector مقدار دهی به
Port(
A: in std_logic_vector(9 downto 0);
C: out std_logic_vector (9 doento 0)
);

Signal b: std_logic_vector(9 downto 0);

B(9 downto 1) <= B(8 downto 0); -- shift
B(0) <= '1';
B(2 downto 0) <= "110";
B <= A;
B(9 downto 8) <= A(2 downto 1);
B(4) <= B(5);
B <= (others => '0');
B(5 downto 0) <= A(9 downto 7) & "011";

استفاده از with-select و when-else

```

Signal q: std_logic_vector(3 downto 0);
Signal t: std_logic_vector(3 downto 0);

```

```

Q<= "0000" when t="1010" else
  "1011" when t="0010" else
  "1111";

```

```

with t select
q<= "0000" when "1010",
  "1011" when "0010",
  "1111" when others;

```

معرفی Component و نحوه نمونه‌سازی از testbench

```

library ieee;
use ieee.std_logic_1164.all;
entity Decoder_vec is
port(

```

```

input: in std_logic_vector(2 downto 0);
output: out std_logic_vector(7 downto 0)
);
end entity Decoder_vec;
architecture behav of Decoder_vec is
begin
    output(0) <= '1' when input="000" else '0';
    output(1) <= '1' when input="001" else '0';
    output(2) <= '1' when input="010" else '0';
    output(3) <= '1' when input="011" else '0';
    output(4) <= '1' when input="100" else '0';
    output(5) <= '1' when input="101" else '0';
    output(6) <= '1' when input="110" else '0';
    output(7) <= '1' when input="111" else '0';
end behav;

```

```

library ieee;
use ieee.std_logic_1164.all;

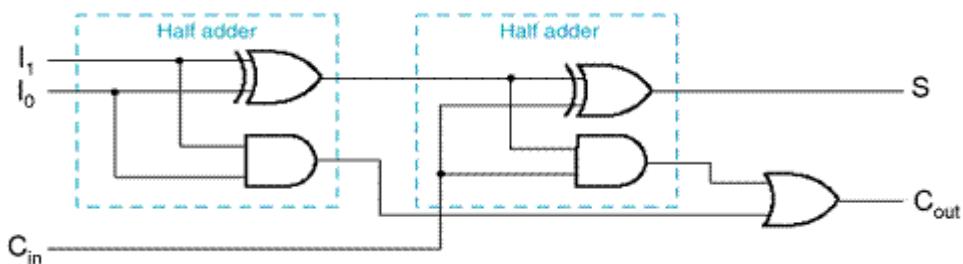
entity decoder_tb is
end entity decoder_tb;

architecture test of decoder_tb is
component Decoder_vec is
port(
    input: in std_logic_vector(2 downto 0);
    output: out std_logic_vector(7 downto 0)
);
end component;
signal i :std_logic_vector(2 downto 0);
signal o : std_logic_vector(7 downto 0);
begin
dec:Decoder_vec port map ( input => i, ouput => o);
i <= "000", "001" after 100 ns, "101" after 200 ns;
end test;

```

طراحی ساختاری

در این روش طراحی، هر موجودیت از چند کامپوننت تشکیل شده است. مثلا یک جمع کننده تک بیتی از دو نیم جمع کننده و یک گیت OR تشکیل شده است (مانند شکل ۱).



شکل ۱ جمع کننده تک بیتی

علاوه بر آن هر نیم جمع کننده از دو گیت AND و XOR تشکیل شده است. طراحی ساختاری این جمع کننده به صورت زیر است:

```
library IEEE;
use IEEE.std_logic_1164.all;
Entity and_gate is
Port(
  A, B: in std_logic;
  C : out std_logic
);
End entity and_gate;
Architecture gatelevel of and_gate is
Begin
  C <= A and B;
End gatelevel;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
Entity or_gate is
Port(
  A, B: in std_logic;
  C : out std_logic
);
End entity or_gate;
Architecture gatelevel of or_gate is
Begin
  C <= A or B;
End gatelevel;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
Entity xor_gate is
Port(
  A, B: in std_logic;
  C : out std_logic
);
End entity xor_gate;
Architecture gatelevel of xor_gate is
Begin
  C <= A xor B;
End gatelevel;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
Port(
  in1, in2: in std_logic;
  out1, out2 : out std_logic
);
end entity half_adder;
architecture structure of half_adder is
  component xor_gate is
    Port(
```

```

A, B: in std_logic;
C : out std_logic
);
End component xor_gate;
component and_gate is
    Port(
        A, B: in std_logic;
        C : out std_logic
    );
End component and_gate;
begin
    xor_gate_instance0: xor_gate port map ( A => in1, B => in2, C=>out1 );
    and_gate_instance0: and_gate port map ( A => in1, B => in2, C=>out2 );
end structure;

library ieee;
use ieee.std_logic_1164.all;
entity full_adder is
    Port(
        i0, i1, cin: in std_logic;
        s, cout : out std_logic
    );
end entity full_adder;
architecture structure of full_adder is
    component or_gate is
        Port(
            A, B: in std_logic;
            C : out std_logic
        );
    End component or_gate;
    component half_adder is
        Port(
            in1, in2: in std_logic;
            out1, out2 : out std_logic
        );
    End component half_adder;
    signal internal_signal0, internal_signal1, internal_signal2: std_logic;
begin
    half_adder_instance0: half_adder port map ( in1 => i0, in2 => i1, out1 => internal_signal0, out2 => internal_signal1);
    half_adder_instance1: half_adder port map ( in1 => internal_signal0, in2 => cin, out1 => s, out2 => internal_signal2);
    or_gate_instance0: or_gate port map ( A => internal_signal1, B => internal_signal2, C => cout );
end structure;

```

خروجی‌های مورد انتظار آزمایش:

انتظار می‌رود دانشجویان بتوانند سخت افزار شکل ۱ را توصیف کرده و واحد تست مربوط به آن را نیز بنویسند. و توصیف‌های انجام شده در شبیه‌ساز بدون خطاب کامپایل و شبیه‌سازی شوند.

خروجی‌های این آزمایش عبارت است از:

- تحويل درستی عملکرد مدار موردنظر
- پیاده‌سازی طراحی ذکر شده، کامپایل و شبیه‌سازی آن
- پیاده‌سازی بر روی FPGA و نیز اطمینان از درستی عملکرد آن با استفاده از ورودی‌ها و خروجی‌های قابل استفاده روی برد FPGA

مراجع مطالعه/ضمایم:

مدارهای پایه	موضوع:
۲	شماره آزمایش:
طراحی مدارهای اولیه (Comprator و Decoder2to4, Encoder4to2, Multiplexer4to1) ۴ بیتی	عنوان:
	هدف:

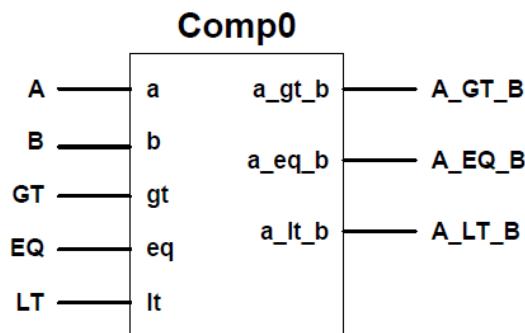
در این آزمایش هدف آشنایی با نحوه عملکرد و پیاده‌سازی هر یک از مدارهای پایه در سطح تجزیه شده است.

آمادگی پیش از آزمایش:

طراحی مدارهای پایه فوق و تحويل آن به مدرس آزمایشگاه قبل از شروع آزمایش.

شرح آزمایش:

در این آزمایش، یک مالتی‌پلکسر، کدکننده، کدگشا و یک مقایسه‌گر ۴ بیتی براساس مارژول‌های تک‌بیتی آن‌ها طراحی و شبیه‌سازی گردد. طراحی انجام شده باید توسط زبان توصیف سخت‌افزار VHDL و در سطح تجزیه شده تجربه شود. تهیه TestBench برای این جمع‌کننده نیز جزء الزام‌های آزمایش است. مدار تک بیتی مورد استفاده برای مقایسه‌گر مطابق با ساختار شکل ۱ پیاده‌سازی شود.



شکل ۱ ساختار ورودی/خروجی‌های مقایسه‌گر تک‌بیتی

خروجی‌های مورد انتظار آزمایش:

هر یک از موارد زیر باید تحويل داده شود:

- طراحی شماتیک طرح (محتوای توصیف) به صورت فیزیکی (بر روی کاغذ)
- بررسی درستی سیگنال‌های خروجی مدارهای مورد نظر با توجه به ورودی‌ها در شبیه‌سازی
- پیاده‌سازی مقایسه‌گر ۴ بیتی بر روی FPGA و نیز اطمینان از درستی عملکرد آن با استفاده از ورودی و خروجی‌های قابل استفاده روی برد FPGA

مراجع مطالعه/ضمایم:

[1] B. Parhami, *Computer Arithmetic-Algorithms and Hardware Designs*, Oxford Univ. Press, 2000.

آشنایی با پیاده‌سازی مدارهای ترتیبی در VHDL

موضوع:

۳

شماره آزمایش:

عنوان:

هدف:

Processها و چگونگی پیاده‌سازی شمارندها و ماشین‌های حالت

آشنایی با Processها به عنوان بخش‌هایی که به صورت ترتیبی اجرا می‌شوند و استفاده از آن‌ها برای طراحی انواع فلیپ فلاپ‌ها، و پیاده‌سازی نمونه‌ای از شمارنده‌ها.

آمادگی پیش از آزمایش:

Processها ماهیت ترتیبی دارند و باید در بدنه‌ی Architecture نوشته شوند. این ساختارها به صورت ترتیبی اجرا می‌شوند لذا درون آن‌ها می‌توان از دستورالعمل‌های If ... then ... else ... استفاده کرد. هر Process برای اجرا به Sensitivity list نیاز دارد. منظور از Sensitivity list سیگنال‌هایی هستند که وقتی Event روی آن‌ها رخ می‌دهد Process را تحریک می‌کند. مطالعه شیوه توصیف و استفاده از این قابلیت‌ها قبل از آزمایش الزامی است.

برای طراحی ماشین‌های حالت از Processها استفاده می‌شود. همان‌طور که گفته شد اگر مقدار سیگنالی که در Sensitivity list وجود دارد تغییر کند Process اجرا می‌شود. اگر Process مدل کننده‌ی یک بلوک ترکیبی است، تمام ورودی‌های آن باید در Sensitivity list لحاظ شود. ماشین‌های حالت در لبه‌های کلاک تغییر حالت می‌دهند، لذا کلاک باید در ورودی Sensitivity list وارد شود. علاوه بر این حالت درونی سیستم هم باید در یک متغیر داخلی نگهداری شود. برای نگهداری حالت‌ها در خود VHDL می‌توان Type جدید تعریف کرد که در اینجا مثلاً دو مقدار A و B نشان‌دهنده دو حالت از سیستم هستند: Type state is (A,B);

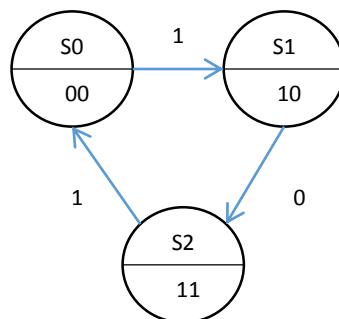
حال برای استفاده از Type تعريف شده می‌توان از آن Type یک سیگنال یا متغیر تعريف کرد. سپس از دستورالعمل Case روی Signal st : state; Process (...) Case st is When A => When B =>

```
Signal st : state;
Process (...)

Case st is
When A => ....
When B => ....
```

برای طراحی مدارات با حافظه مدلی به نام مدل هافمن وجود دارد که قسمت ترکیبی مدار را از قسمتی ترتیبی آن جدا می‌کند. قسمت ترتیبی آن معمولاً با Processی که نسبت به سیگنال کلاک و reset حساس است نوشته می‌شود و قسمت ترکیبی آن در Process که نسبت به حالت‌های مدار حساس است نوشته می‌شود.

در ادامه مثالی از ماشین حالت Moore و نحوه پیاده‌سازی آن آورده شده است.



شکل ۱- نمونه‌ای از ماشین حالت Moore

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity moore_machine is
    port(
        input : in std_logic ;
        output : out std_logic_vector(1 downto 0);
        clk : in std_logic
    );
end moore_machine;
architecture Behavioral of moore_machine is
    type state_t is (s0 , s1 , s2);
    signal state : state_t := s0;
    signal next_state : state_t := s0;
begin
    CMB : process(state , input)
    begin
        case state is
            when s0=>
                if(input = '1') then
                    next_state <= s1;
                else
                    next_state <= state ;
                end if;
            when s1=>
                ....
            when s2=>
                ....
            when others=>
                next_state <= s0;
        end case;
    end process;
    REG : process(clk)
    begin
        if(clk'event and clk = '1') then
            state <= next_state;
        end if;
    end process;
    output <= "00" when state = s0  else
        "10" when state = s1  else
        "11" when state = s2;
end Behavioral;
  
```

شرح آزمایش:

- ۱) یک فلیپفلاب از نوع D(DFF) با سیگنال Reset ناهمگام (asynchronous) در منطق منفی (active low) طراحی کنید.
- ۲) یک فلیپفلاب از نوع T(TFF) با سیگنال Reset ناهمگام طراحی کنید.
- ۳) یک Ripple Counter، ۴ بیتی که نمونه‌ای از شمارنده‌های ناهمگام می‌باشد را با استفاده از TFF ساخته شده در (۲) طراحی کنید.
- ۴) مدار یک Sequence detector برای رشته‌ی "۱۱۰۱" را ابتدا به صورت Moore و سپس Mealy طراحی کرده و آن را با استفاده از زبان VHDL طراحی کنید.
- ۵) مداری طراحی کنید که رخداد هر یک از دو رشته "۱۱۰۰" و "۱۰۱۰" را در ورودی تشخیص دهد.

خروجی‌های مورد انتظار آزمایش:

هریک از بخش‌های این آزمایش در قالب‌های زیر به ترتیب تحويل مدرس آزمایشگاه گردد:

- طراحی شماتیک طرح (محتوای توصیف) بر صورت فیزیکی
- پیاده‌سازی مدارها با استفاده از زبان‌های توصیف
- شبیه‌سازی مدارهای توصیف شده و تهیه Testbench برای آن و نشان دادن درستی عملکرد آن‌ها.
- سنتز و پیاده‌سازی مدارها بر روی برد FPGA و اثبات درستی عملکرد مدارها. (برای استفاده از کلک برد و مشاهده نتایج نیاز به کاهش فرکانس برد می‌باشد، بدین جهت متناسب با شرایط کلاس این کاهش فرکانس توسط مدرس آزمایشگاه و یا دانشجویان انجام گیرد).

مراجع مطالعه/ضمایم:

- [1] M. Zwolinski, "Digital System Design with VHDL," Pearson Educated, 2000, pp. 104-128.

موضع:	جمع کننده‌ها
شماره آزمایش:	۴
عنوان:	پیاده‌سازی انواع جمع کننده‌ها و تحلیل سرعت آنها
هدف:	

هدف از این آزمایش، آشنایی با چگونگی عملکرد هر یک از جمع کننده‌ها است. تفاوت این جمع کننده‌ها در سرعت محاسبه عملیات جمع می‌باشد. در این آزمایش هدف آشنایی با نحوه عملکرد و پیاده‌سازی سخت‌افزاری آن‌ها در سطح تجزیه و تحلیل می‌باشد.

آمادگی پیش از آزمایش:

مطالعه انواع جمع کننده‌های زیر، طراحی شماتیک سخت‌افزاری آن با استفاده از واحدهای Full Adder

(الف) Ripple Adder (Cascaded Adder)

(ب) Carry-Lookahead Adder

(ج) Carry Select Adder

شرح آزمایش:

در این آزمایش، یک جمع کننده ۴ بیتی از نوع‌های زیر، طراحی و پیاده‌سازی گردد. طراحی انجام شده با زبان توصیف سخت‌افزاری VHDL و در سطح تجزیه و تحلیل می‌باشد. در این پیاده‌سازی از واحدهای HalfAdder و Full Adder استفاده گردد. تهیه TestBench برای این جمع کننده نیز جزء الزام‌های آزمایش است.

(الف) Ripple Adder (Cascaded Adder)

(ب) Carry-Lookahead Adder

(ج) Carry Select Adder

خروجی‌های مورد انتظار آزمایش:

هر یک از موارد زیر باید تحویل داده شود:

- طراحی شماتیک طرح (محتوای توصیف) بر روی کاغذ
- بررسی درستی سیگنال‌های خروجی مدار مورد نظر با انجام شبیه‌سازی
- پیاده‌سازی بر روی FPGA و نیز اطمینان از درستی عملکرد آن با استفاده از ورودی‌ها و خروجی‌های قابل استفاده روی برد FPGA

مراجع مطالعه / ضمایم:

[1] B. Parhami, *Computer Arithmetic-Algorithms and Hardware Designs*, Oxford Univ. Press, 2000.

[2] https://en.wikipedia.org/wiki/Carry-select_adder

موضوع:	ضرب کننده‌ها
شماره آزمایش:	۵
عنوان:	پیاده‌سازی انواع ضرب کننده ۴ بیتی
هدف:	آشنایی با پیاده‌سازی انواع ضرب کننده آرایه‌ای و ضرب کننده‌های سریع با زبان VHDL.

آشنایی با پیاده‌سازی انواع ضرب کننده آرایه‌ای و ضرب کننده‌های سریع با زبان VHDL.
پیاده‌سازی ساخت افزاری تمامی واحدها در سطح تجزیه و گزینه صورت گیرد.

آمادگی پیش از آزمایش:

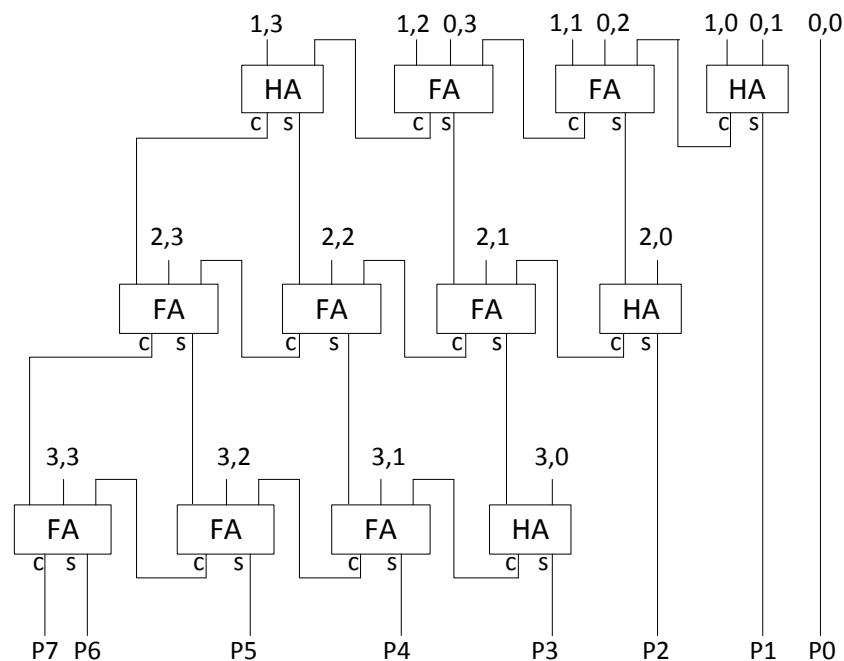
مطالعه مواردی که در بخش مراجع آزمایش به آن اشاره شده است و انجام برخی شبیه‌سازی‌های مورد نظر.

شرح آزمایش:

ضرب کننده‌ها یکی از مهمترین واحدهای محاسباتی در طراحی پردازنده‌ها محسوب می‌شوند و روش‌های گوناگونی برای پیاده‌سازی این واحد ارائه شده است. در این آزمایش چهار روش متفاوت برای انجام عملیات ضرب ۴ بیتی صحیح بی‌علامت ارائه شده است. در هریک از این روش‌ها دو عدد ۴ بیتی به عنوان ورودی و یک عدد ۸ بیتی در خروجی محاسبه می‌شود.

الف) ضرب کننده معمولی:

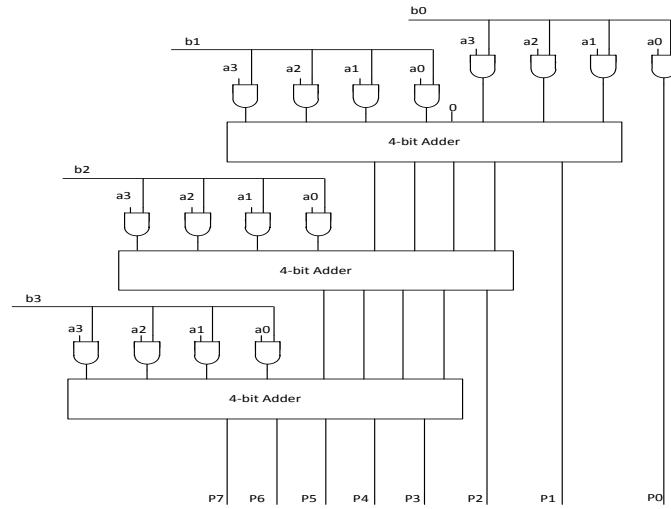
در این ضرب کننده مثل ضرب دهدۀی (decimal)، در هر مرحله یک بیت از مضروب (multiplier) در تمام بیت‌های مضروب فیه (multiplicand) ضرب شده و با یک بیت شیفت با خروجی مرحله قبل جمع می‌شود. نمودار این ضرب کننده در شکل ۱ نشان داده شده است.



شکل ۱- نمودار بلوکی ضرب کننده معمولی

ب) ضرب کننده آرایه‌ای:

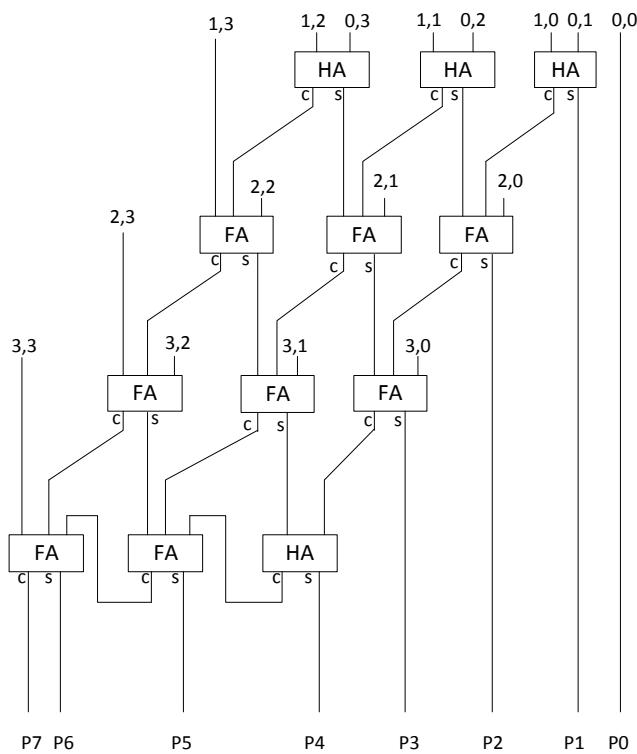
در این ضرب کننده از واحدهای جمع کننده ۴ بیتی استفاده شده است. نمودار این ضرب کننده در شکل ۲ نشان داده شده است.



شکل ۲- نمودار بلوکی ضرب کننده آرایه ای

ج) ضرب کننده Carry-Save Adder

در این نوع ضرب کننده با انتقال بیت نقلی به مرحله بعد سرعت انجام ضرب افزایش می‌یابد. نمودار این ضرب کننده در شکل ۳ نشان داده شده است.

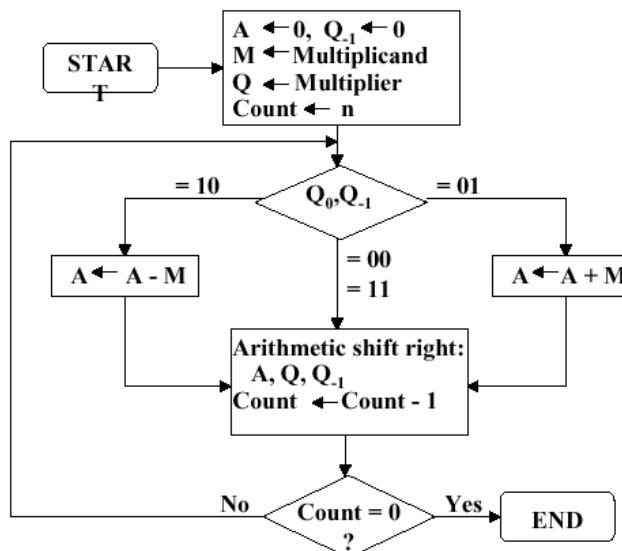


شکل ۳- نمودار بلوکی ضرب کننده Carry save adder

(d) ضرب کننده Booth (اختیاری)

در این نوع ضرب کننده که از روش‌های ضرب سریع محسوب می‌شود ضرب‌های جزئی کمتری خواهیم داشت که می‌توان این ضرب‌های جزئی را به روش‌های مختلف با هم جمع کرد. این روش هر چند روند ضرب را سریع‌تر می‌کند ولی آن را پیچیده‌تر خواهد کرد.

فلوچارت این روش در شکل ۴ نشان داده شده است. تفاوت اصلی این روش با روش‌های قبل در ترتیبی بودن آن است. به این صورت که مضروب فیل (multiplicand) در یک shift register (multipliсand) در یک ۸ بیتی قرار می‌گیرد و در هر پالس ساعت یک بیت به سمت راست داده می‌شود. خروجی این ضرب کننده پس از ۸ پالس ساعت قابل استفاده است.



شکل ۴-فلوچارت الگوریتم ضرب Booth

خروجی‌های مورد انتظار آزمایش:

- پیاده‌سازی بر روی مدار FPGA و نیز اطمینان از درستی عملکرد آن با استفاده از ورودی‌ها و خروجی‌های قابل استفاده روی برد FPGA

مراجع مطالعه/ضمایم:

- [1] M. Morris Mano, *Computer system architecture*, Prentice-Hall, 3rd edition.

موضوع: تاخیر در مدارات دیجیتال

شماره آزمایش: ۶

عنوان: ایجاد تاخیر مشخص در مدارهای دیجیتال

هدف:

هدف از این آزمایش، آشنایی با چگونگی ایجاد تاخیر در مدارهای دیجیتال و بررسی آن بر روی برد می‌باشد.

آمادگی پیش از آزمایش:

مطالعه و تشریح مدارهای شیفت دهنده (منطقی و ریاضی) و چگونگی ایجاد تاخیر در مدارهای دیجیتال.

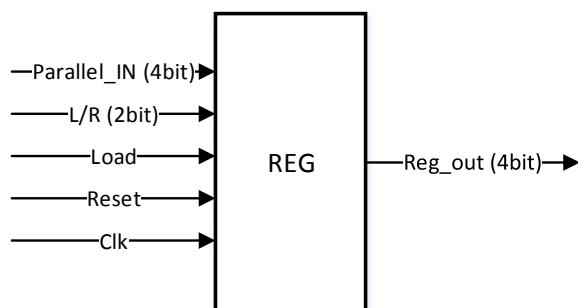
شرح آزمایش:

یکی از بحث‌های اصلی در مدارهای دیجیتال، بخش زمان‌بندی است. زمان‌بندی نامناسب مدارات می‌تواند مشکلات زیادی را در هنگام شبیه‌سازی و پیاده‌سازی به وجود آورد. در این آزمایش به چگونگی ایجاد تاخیر در مدارات دیجیتال پرداخته می‌شود. برای نیل به این منظور از ثبات‌های دارای قابلیت انتقال (منطقی و ریاضی) استفاده می‌کنیم.

الف) پیاده‌سازی ثبات با قابلیت انتقال (Shift):

یک ثبات چهار بیتی با قابلیت‌های زیر طراحی نمایید. ساختار بلوک ثبات در شکل ۱ نشان داده شده است. قابلیت‌های این بلوک به شرح زیر می‌باشد: (ثبت حساس به لبه بالارونده کلک است).

- انتقال منطقی (چپ / راست)
- انتقال ریاضی (چپ / راست)



شکل ۱- ساختار ورودی/خروجی‌های بلوک ثبات

در بلوک ثبات فوق در گاههای ورودی خروجی به شرح زیر است:

- ورودی LR دویتی است که برای انتخاب بین حالت شیفت است. ("00" LR = "00" ثبات مقدار قبلی خود را حفظ می‌کند).
- انتقال به چپ ریاضی و منطقی یک حالت فرض شوند).
- ورودی Load تک بیتی است که در صورت یک بودن، ورودی چهاربیتی Parallel_IN همزمان با لبه بالارونده کلک در ثبات بارگذاری می‌شود. و در صورت صفر بودن ثبات مقدار قبلی خود را نگه می‌دارد.

ب) چگونگی ایجاد مقدار تاخیر مشخص برای نمایش خروجی روی برد:

برای ایجاد تاخیر و قابل رویت شدن تغییرات خروجی‌ها بر روی برد، از شمارنده برای تنظیم و کاهش فرکانس کلک ر روی برد استفاده می‌شود. فرکانس کاری برد موجود در آزمایشگاه ۴۰ مگاهرتز می‌باشد، بنابراین برای دستیابی به تاخیر ۱ ثانیه، نیاز به شمارنده‌ای است که به میزان 1×10^7 بار بشمارد تا تاخیری به میزان ۱ ثانیه ایجاد گردد.

در این بخش از آزمایش شمارنده‌ای طراحی و به کد بخش الف اضافه کنید تا بتوان خروجی‌های شیفت رجیستر را با تاخیرهای مناسب بر روی برد مشاهده نمود.

ج) نمایش خروجی بر روی Seg-7:

چگونگی اتصال پایه‌ها به منظور نمایش بر روی seg-7 به صورت زیر می‌باشد. (کلک برد به پایه 184 متصل شده است).

```
#PlanAhead Generated physical constraints
```

```
NET "SEG_DATA[0]" LOC = P10;
NET "SEG_DATA[1]" LOC = P7;
NET "SEG_DATA[2]" LOC = P11;
NET "SEG_DATA[3]" LOC = P5;
NET "SEG_DATA[4]" LOC = P4;
NET "SEG_DATA[5]" LOC = P12;
NET "SEG_DATA[6]" LOC = P9;
NET "SEG_DATA[7]" LOC = P3;

NET "SEG_SEL[0]" LOC = P15;
NET "SEG_SEL[1]" LOC = P20;
NET "SEG_SEL[2]" LOC = P19;
NET "SEG_SEL[3]" LOC = P18;
NET "SEG_SEL[4]" LOC = P16;
```

خروجی‌های مورد انتظار آزمایش:

هریک از بخش‌های این آزمایش در قالب‌های زیر به ترتیب تحويل مدرس آزمایشگاه گردد:

- پیاده‌سازی مدارها با استفاده از زبان‌های توصیف سخت افزار VHDL
- شبیه‌سازی مدارهای توصیف شده و تهیه Testbench برای آن و نشان دادن درستی عملکرد آن‌ها.
- سنتز و پیاده‌سازی مدارها بر روی بورد FPGA و اثبات درستی عملکرد مدارها.

مراجع مطالعه/ضمایم:

[1] Mano, Kime, *Logic and Computer Design Fundamentals*, Prentice-Hall, 2004.

هدف:	حافظه ها
شماره آزمایش:	۷
عنوان:	طراحی حافظه CAM و RAM ,ROM

هدف از این آزمایش، آشنایی با انواع حافظه ها، طراحی و پیاده سازی برخی از آن ها است.

آمادگی پیش از آزمایش:

مطالعه و تشریح کتبی چگونگی عملکرد حافظه های فوق و آشنایی با نحوه پیاده سازی آن ها.

شرح آزمایش:

یکی از بخش های اصلی در مدارهای دیجیتال، بخش حافظه است. مدل سازی نامناسب حافظه علاوه بر اینکه ممکن است زمان شیوه سازی را طولانی کند، در هنگام سنتز نیز در صورت توصیف نادرست باعث استفاده غیر معمول از منابع سخت افزاری می شود. در این آزمایش هدف اصلی آشنایی با نحوه مدل کردن انواع حافظه ها با زبان توصیف سخت افزار است.

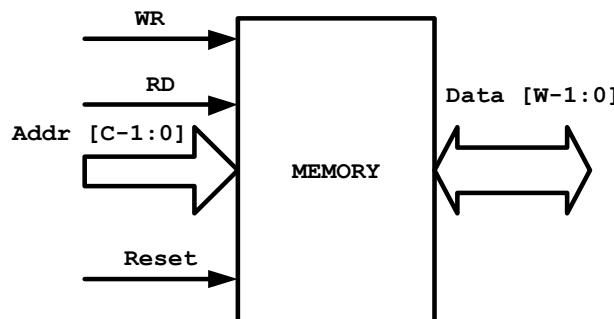
الف) طراحی حافظه RAM

یک حافظه RAM مطابق با شکل زیر طراحی کنید. پارامترهای این بلوک به شرح زیر می باشد:

- عرض حافظه W

- تعداد خانه های حافظه D

- عرض در گاه Address برابر با $\log_2(D)$ می باشد که فعلا آن را هم در کد با پارامتر C نشان دهید.



شکل ۱- بلوگ دیاگرام یک واحد حافظه

در بلوک حافظه فوق:

در گاه Data از نوع ورودی/خروجی تعریف می شود:

- سیگنال های WR و RD حساس به لبه بالارونده عمل کرده بطوریکه هنگامی WR در لبه بالارونده است مقدار Data به خانه ای که توسط Addr مشخص شده، نوشته می شود. هنگامی که RD در لبه بالارونده است عمل خواندن از حافظه انجام شود.
- در گاه Addr نشان می دهد که داده به کدام خانه SRAM باید نوشته و از کدام خانه باید خوانده شود.
- سیگنال Clk پالس ساعت است.
- سیگنال Reset در منطق منفی و ناهمگام (Asynchronous active low) عمل می کند.

در این بلوک، خانه‌های SRAM به هنگام Reset به بدین ترتیب مقداردهی شوند: مقدار خانه شماره "۰" برابر ".۰". مقدار خانه شماره "۱" برابر "۱" و به همین ترتیب مقدار اولیه هر خانه برابر با آدرس آن خانه باشد. برنامه VHDL فوق را نوشه و شبیه‌سازی کنید. (در حالت‌های مختلف)

ب) طراحی حافظه ROM

با تغییری در طراحی انجام شده در بخش (الف)، این حافظه را به حافظه ROM تبدیل کنید.

ج) طراحی حافظه RAM دو درگاهه (Dual port RAM)

با تغییری در طراحی انجام شده در بخش (الف)، این حافظه را به حافظه دو درگاه مستقل برای خواندن/نوشتن بطور کاملاً مستقل وجود داشته باشد.

د) طراحی حافظه CAM

با مشخصات داده شده در (الف) حافظه آدرس پذیر محتوی (Content Addressable Memory) طراحی و پیاده‌سازی کنید. این حافظه بدون درگاه آدرس بوده، و خواندن یا نوشتن در آن بر اساس محتوا انجام خواهد شد. در هنگام نوشتن داده، اگر داده داخل حافظه نباشد، آن داده در اولین مکان خالی نوشته می‌شود و در هنگام خواندن، در صورتی که داده وارد شده در حافظه وجود داشته باشد، سیگنال match (تک بیتی) یک خواهد شد به معنای آنکه داده در حافظه یافت شده است و در غیر اینصورت صفر خواهد بود.

خروجی‌های مورد انتظار آزمایش:

هریک از بخش‌های این آزمایش در قالب‌های زیر به ترتیب تحويل مدرس آزمایشگاه گردد:

- طراحی شماتیک طرح و محتوای توصیف بر روی کاغذ
- پیاده‌سازی مدارها با استفاده از زبان‌های توصیف
- شیوه‌سازی مدارهای توصیف شده و تهیه Testbench برای آن و نشان دادن درستی عملکرد آن‌ها.
- سنتز و پیاده‌سازی مدارها بر روی بورد FPGA و اثبات درستی عملکرد مدارها.

مراجع مطالعه/ضمایم:

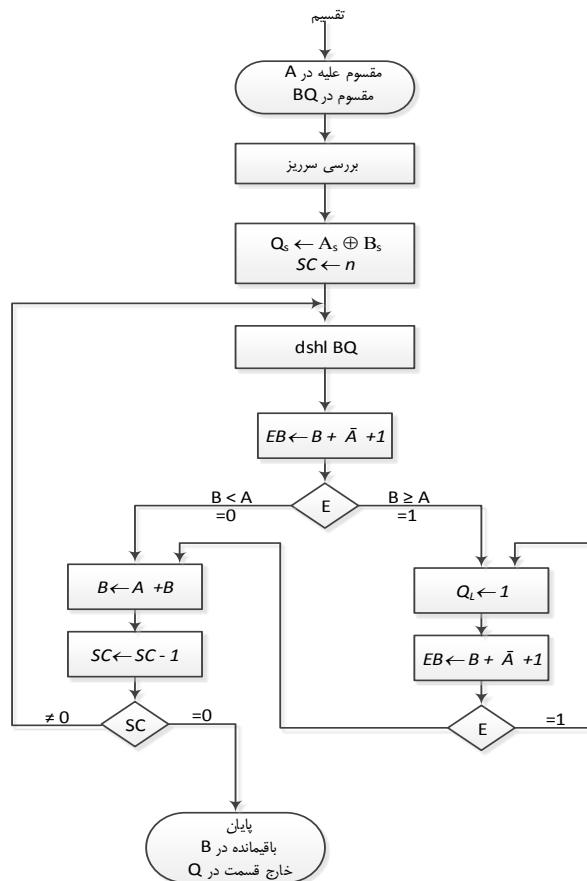
[1] Mano, Kime, *Logic and Computer Design Fundamentals*, Prentice-Hall, 2004, chapter 12.

توضیع کننده‌ها	موضع:
۸	شماره آزمایش:
تقسیم کننده دودویی در منطق مکمل ۲	عنوان:
	هدف:

طراحی تقسیم کننده دودویی با کمک زبان توصیف سخت‌افزار

آمادگی پیش از آزمایش:

در مورد تقسیم کننده‌ها اطلاعات لازم مطالعه شود. می‌توان از الگوریتم تقسیم کننده زیر استفاده کرد. تشریح نحوه کار این فلوچارت و اثبات درستی عملکرد مدار الزامی است.



شکل ۱- فلوچارت الگوریتم تقسیم

شرح آزمایش:

تقسیم کننده دودویی در منطق نمایش مکمل ۲ طراحی کنید که یک مقسوم ۸ بیتی را بر مقسوم علیه ۴ بیتی تقسیم کند.

خروجی‌های مورد انتظار آزمایش:

موارد زیر باید تحويل مدرس آزمایشگاه شود:

- طراحی شماتیک سخت‌افزاری مورد نظر و اثبات درستی آن بطور تحلیلی
- توصیف و پیاده‌سازی تقسیم‌کننده مورد نظر به زبان توصیف
- انجام شبیه‌سازی و ارایه آن به مدرس آزمایشگاه
- تمام کدهای توصیفی به همراه Testbench برای مشاهده نتایج ارائه شود.

مراجع مطالعه/ضمایم:

[1] M. Mano, *Computer System Architecture*, Prentice-Hall, 3rd Edition, pp. 345-351, 1993.

موضوع:	کامپیوتر پایه
شماره آزمایش:	۹
عنوان:	پیاده‌سازی یک کامپیوتر پایه
هدف:	

پیاده‌سازی یک کامپیوتر پایه و مشاهده عملی چگونگی عملکرد آن.

آمادگی پیش از آزمایش:

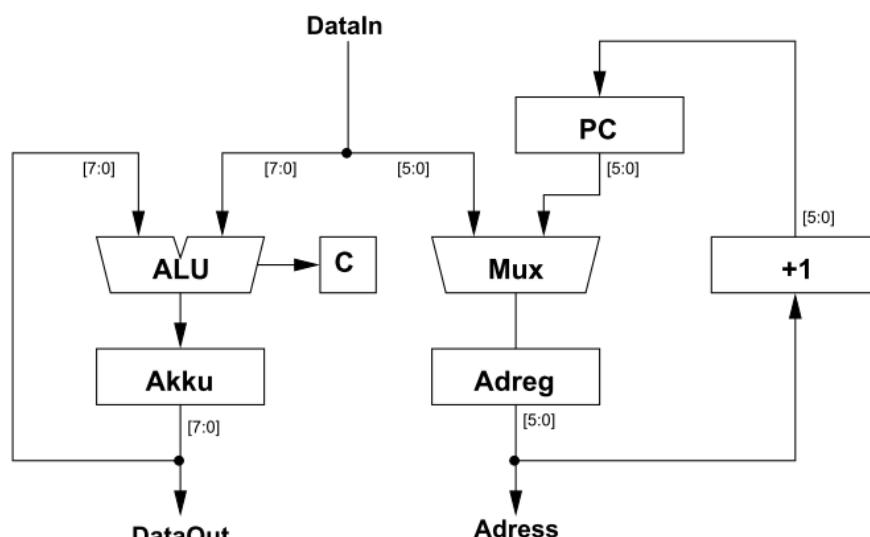
بررسی کلیات کامپیوتر پایه درس معماری کامپیوتر؛ مطالعه کد موجود در دستورکار و کسب آمادگی لازم جهت پیاده‌سازی آن.

شرح آزمایش:

در این آزمایش یک کامپیوتر پایه طراحی و سپس روی FPGA پیاده‌سازی می‌شود. کامپیوتر پایه از سه جز تشکیل می‌شود: ۱) پردازنده، ۲) حافظه و ۳) واحد ورودی/خروجی. با توجه به بوردهای موجود، امکان ایجاد هر سه این قسمت‌ها بر روی بورد وجود دارد. کامپیوتری که پیاده‌سازی می‌شود می‌تواند دستورات موجود در شکل ۱ را داشته باشد: (دستور show محتوای یک رجیستر را روی LED‌ها نشان می‌دهد).

یک نمونه کد کامپیوتر پایه و مسیر داده مربوط به آن در زیر آمده است.

Load R []
Store R []
Brz []
Brnz []
Add R []
Sub R []
Ror R
Rol R
Shr R
Shl R
Show R
And R []
Or R []
Xor R []
Not R
Halt



شکل ۲- دیاگرام بلوکی کامپیوتر پایه

شکل ۱- مجموعه دستورالعمل‌های پیشنهادی برای کامپیوتر پایه

```
-- Minimal 8 Bit CPU
--
-- rev 15102001
-- 01-02/2001 Tim Böscke
-- t.boescke@tuhh.de

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity CPU8BIT2 is
    port ( data: inout std_logic_vector(7 downto 0);
            adress: out    std_logic_vector(5 downto 0);
            oe:      out    std_logic;
            we:      out    std_logic;           -- Asynchronous memory interface
            rst:     in     std_logic;
            clk:     in     std_logic);
end;

architecture CPU_ARCH of CPU8BIT2 is
    signal akku: std_logic_vector(8 downto 0); -- akku(8) is carry !
    signal adreg: std_logic_vector(5 downto 0);
    signal pc:    std_logic_vector(5 downto 0);
    signal states: std_logic_vector(2 downto 0);
begin
    Process(clk,rst)
    begin
        If (rst = '0') then
            adreg <= (others => '0');      -- start execution at memory location 0
            states <= "000";
            akku <= (others => '0');
            pc  <= (others => '0');

        elsif rising_edge(clk) then
            -- PC / Adress path
            If (states = "000") then
                pc    <= adreg + 1;
                adreg <= data(5 downto 0);
            else
                adreg <= pc;
            end If;

            -- ALU / Data Path
            Case states is
                when "010" => akku <= ("0" & akku(7 downto 0)) + ("0" & data);    -- add
                when "011" => akku <= ("0" & akku(7 downto 0)) - ("0" & data);    -- sub
                when "100" => akku <= ("0" & akku(7 downto 0)) * ("0" & data);    -- mult
                when "101" => akku <= ("0" & akku(7 downto 0)) / ("0" & data);    -- div
                when "110" => akku <= "00000000";                                     -- clear
                when others => akku <= akku;                                         -- no operation
            end Case;
        end If;
    end process;
end;
```

```

when "011" => akku(7 downto 0) <= akku(7 downto 0) nor data;      -- nor
when "101" => akku(8) <= '0';           -- branch not taken, clear carry
when others => null;                  -- instr. fetch, jcc taken (000), sta (001)
end Case;

-- State machine
If (states /= "000") then states <= "000";                      -- fetch next opcode
elsif (data(7 downto 6) = "11" and akku(8)='1') then states <= "101"; -- branch not taken
else states <= "0" & not data(7 downto 6);                     -- execute instruction
end If;
end If;
end Process;

-- output
adress <= adreg;
data    <= "ZZZZZZZZ" when states /= "001" else akku(7 downto 0);
oe      <= '1' when (clk='1' or states = "001" or rst='0' or states = "101") else '0';
-- no memory access during reset and state "101" (branch not taken)
we <= '1' when (clk='1' or states /= "001" or rst='0') else '0';

end CPU_ARCH;

```

خروجی‌های مورد انتظار آزمایش:

یک برنامه در حافظه کامپیوتر load شود و بتوان اجرای آن را روی LED های روی بورد FPGA دید.

مراجع مطالعه / ضمایم:

موضوع: طراحی واحد کنترل

شماره آزمایش: ۱۰

عنوان: طراحی واحد کنترل سیم‌بندی شده (Hard wired Control Unit)

هدف:

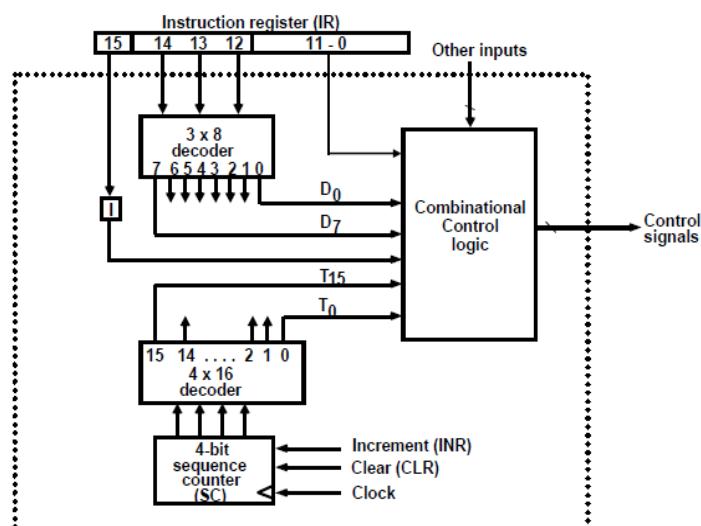
آشنایی با طراحی واحد کنترل سیم‌بندی شده

آمادگی پیش از آزمایش:

مطالعه مواردی که در بخش مراجع آزمایش به آن اشاره شده است و انجام برخی شبیه سازی‌های مورد نظر.

شرح آزمایش:

وظیفه واحد کنترل در یک کامپیوتر، دنبال کردن دنبالهای از ریز دستورالعمل‌هاست. که دستورالعمل‌های زبان ماشین را به سیگنال‌های کنترلی برای کامپیوتر تبدیل می‌کند. شکل زیر شما کلی واحد کنترل کامپیوتر پایه را نشان می‌دهد.



شکل ۱: واحد کنترل کامپیوتر

این واحد به دو صورت می‌تواند طراحی شود، واحد کنترل سیم کشی شده که برای تولید سیگنال‌های کنترلی از مدارهای ترتیبی و ترکیبی استفاده می‌کند و واحد کنترل ریزبرنامه نویسی شده که متشکل از یک حافظه‌ی کنترل است که شامل ریزبرنامه‌هایی است که سیگنال‌های کنترلی لازم را فعال می‌کند.

در این آزمایش واحد کنترلی بر اساس مجموعه دستورالعمل‌های داده شده از سوی مدرس آزمایشگاه، طراحی و پیاده‌سازی گردد.

خروجی‌های مورد انتظار آزمایش:

یک دستورالعمل ۱۶ بیتی را انتخاب نموده و سیگنال‌های کنترلی که واحد کنترل تولید می‌کند را نشان دهید.

مراجع مطالعه / ضمایم:

موضع:	خط لوله
شماره آزمایش:	۱۱
عنوان:	پیاده‌سازی خط لوله و بررسی تسریع آن
هدف:	

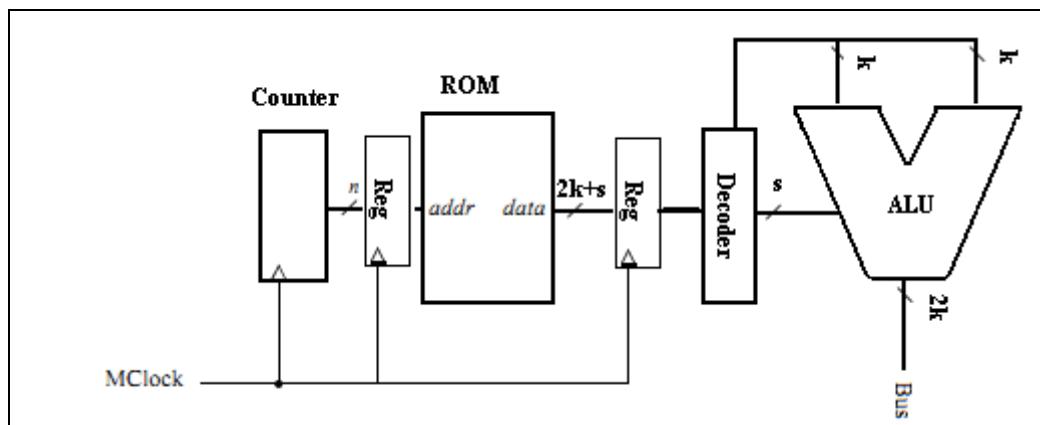
هدف از این آزمایش، آشنایی با روش خط‌لوله در تسریع سازی مدارات دیجیتالی است.

آمادگی پیش از آزمایش:

مطالعه مفاهیم خط‌لوله، حداکثر میزان تسریع آن، ملاحظات تقسیم‌بندی مدارت برای استفاده در مراحل خط‌لوله الزامی است.

شرح آزمایش:

استفاده از خط لوله برای موازی‌سازی پردازنده‌ها در سطح دستورالعمل به منظور افزایش کارایی پردازنده‌ها راهکاری مؤثر است. شکل زیر یک نمودار بلوکی برای خط لوله ساده را نشان می‌دهد.



شکل ۱- نمایی از یک خط لوله ۳ مرحله‌ای

- ۱) مداری که حاصل ضرب $a*b*c*data$ را بدون ساختار خط‌لوله انجام می‌دهد، در زیر نمایش داده شده است. آن را با خط‌لوله سه مرحله‌ای پیاده‌سازی کنید.

```

Process (clock,a,b,c, data)
Begin
If rising_edge (clock)then
--multiplication is done in a single stage.
    result <= a * b * c * data;
end If;
End Process;
  
```

- ۲) خط‌لوله‌ای که دارای چهار مرحله برای ضرب دو ماتریس 4×4 است را با زبان VHDL طراحی و توصیف کنید.

خروجی‌های مورد انتظار آزمایش:

کدهای VHDL مربوطه به همراه فایل های TestBench ارایه شود. در مورد تمرین دوم ابتدا چهار مرحله خط‌لوله طراحی شود و سپس کدهای توصیفی آن و درستی عملکرد آن را توسط شبیه‌ساز نشان دهید.

مراجع مطالعه/ضمایم:

- [1] D. Patterson, J. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann Publishers, Inc., Third edition, 2007.

موضوع: ورودی/خروجی

شماره آزمایش: ۱۲

عنوان: پیاده‌سازی ارسال/دریافت ناهمگام در گذرگاه‌های یکطرفه و دوطرفه

هدف:

آشنایی با پیاده‌سازی انواع روش‌های ارسال/دریافت ناهمگام داده‌ها روی گذرگاه‌های یکطرفه و دوطرفه

آمادگی پیش از آزمایش:

مطالعه مواردی که در بخش مراجع آزمایش آورده شده است و انجام برخی شبیه‌سازی‌های مورد نظر.

شرح آزمایش:

در سیستم‌های پردازشی، انتقال داده‌ها میان پردازنده و واحدهای حافظه‌ای و ورودی/خروجی‌ها اهمیت بالایی دارد. گذرگاه (باس) یکی از مهمترین ارتباطاتی است که چندین گیرنده و فرستنده داده می‌توانند به آن دسترسی داشته باشند. بنابر این لازم است در هر لحظه از زمان تنها یک گذرگاه داده قرار دهد و سایر واحدها تنها می‌توانند آن را دریافت کنند.

در این آزمایش ابتدا یک گذرگاه دو طرفه ساده پیاده‌سازی می‌شود و در ادامه آن برخی از پروتکل‌های همگام‌سازی انتقال اطلاعات و داوری (Arbitration) بررسی می‌شود.

الف) پیاده‌سازی گذرگاه داده دو طرفه

گذرگاه داده دو طرفه که قابلیت نوشتن و خواندن روی آن وجود دارد با سیگنال‌های کنترلی clock و output enable قابل پیاده‌سازی است.

```
ENTITY bidir IS
  PORT(
    bidir : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    oe, clk : IN STD_LOGIC;
    inp   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    outp  : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
  END bidir;
```

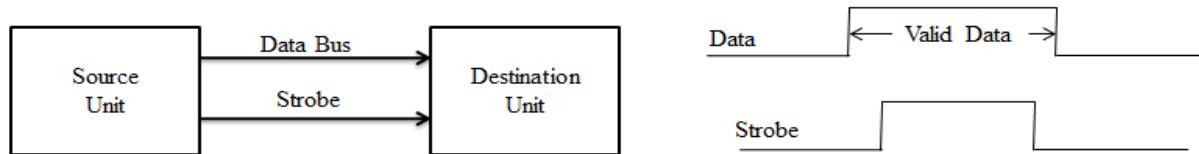
نمونه‌ای از این گذرگاه ۸ بیتی را با زبان VHDL پیاده‌سازی نمایید.

ب) انتقال داده ناهمگام

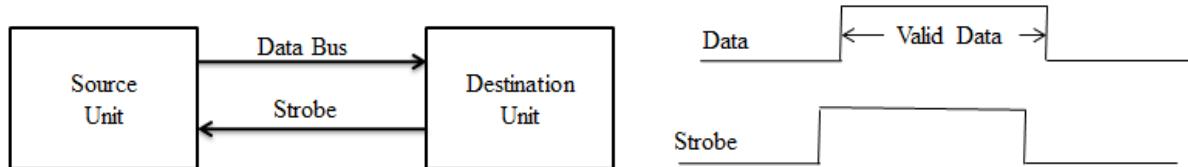
انتقال داده میان یک گیرنده و فرستنده بدون استفاده از clock را انتقال ناهمگام می‌نامند. برای هماهنگی میان گیرنده و فرستنده دو روش متداول استفاده از سیگنال strobe و روش دستدهی وجود دارد. که در ادامه بررسی می‌شوند.

(۱) استفاده از سیگنال کنترلی strobe

سیگنال strobe یکی از روش‌های هماهنگ‌سازی میان گیرنده و فرستنده داده است. در این روش تنها یک بیت سیگنال کنترلی به سیستم افزوده می‌شود. پیاده‌سازی این روش به دو شیوه "به ابتکار گیرنده" و "به ابتکار فرستنده" صورت می‌گیرد. شکل ۱ و ۲ نمودار این دو روش به همراه زمان‌بندی سیگنال‌ها را نشان می‌دهد.



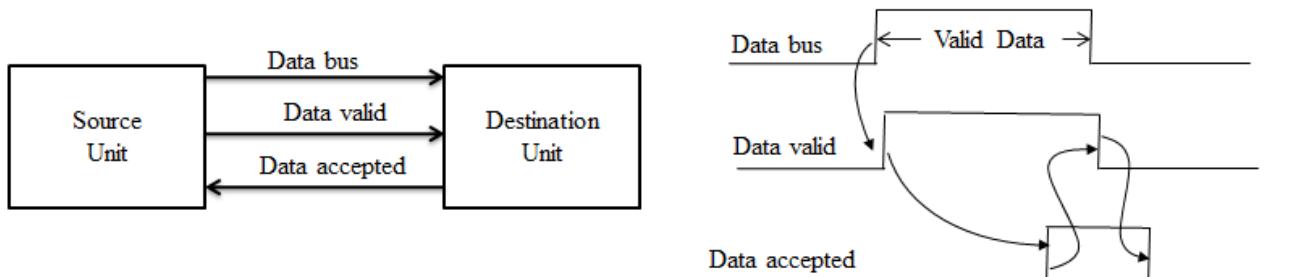
شکل ۱- انتقال داده با سیگنال strobe به ابتکار فرستنده



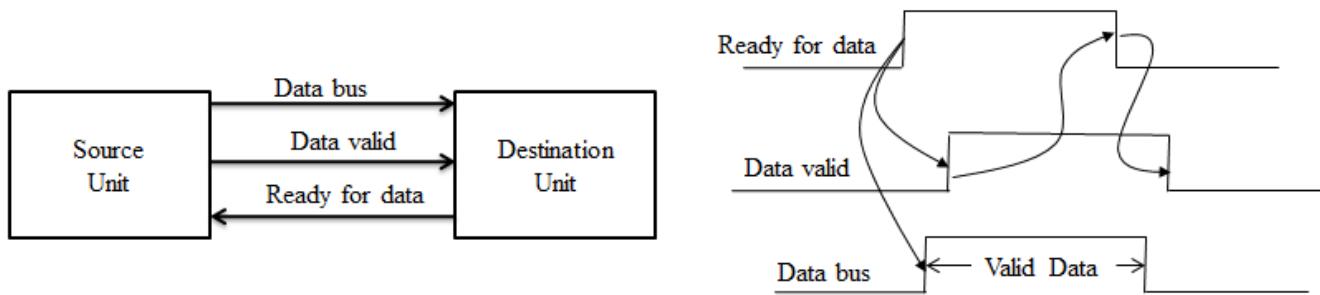
شکل ۲- انتقال داده با سیگنال strobe به ابتکار گیرنده

(۲) روش دستدهی

روش دستدهی یکی دیگر از روش‌های متداول انتقال غیرهمگام داده است که با استفاده از دو سیگنال تک بیتی Data valid و Data accepted می‌کند میان گیرنده و فرستنده هماهنگی ایجاد کند. پیاده‌سازی این روش نیز به دو شیوه "به ابتکار گیرنده" و "به ابتکار فرستنده" صورت می‌گیرد. شکل ۳ و ۴ نمودار این دو روش به همراه زمان‌بندی سیگنال‌ها را نشان می‌دهد.



شکل ۳- انتقال داده با روش دستدهی به ابتکار فرستنده



شکل ۴- انتقال داده با روش دستدهی به ابتکار گیرنده

خروجی های مورد انتظار آزمایش:

لازم است هر گروه از دانشجویان از هریک از بخش های (الف) و (ب) را با استفاده از زبان توصیف سخت افزار پیاده سازی و شبیه سازی نمایند و در رابطه با زمان بندی سیگنال ها تحلیل ارائه دهند.

هر یک از گروه ها می توانند دو entity جداگانه با عملکرد درونی انتخابی به عنوان گیرنده و فرستنده پیاده سازی کنند. واحد فرستنده در زمان های دلخواه، اطلاعات خود را برای گیرنده ارسال نمایند. واحد گیرنده نیز پردازش درونی خود را روی داده های دریافتی ارسال می کند.

مراجع مطالعه / ضمایم:

- [1] M. Morris Mano, *Computer system architecture*, Prentice-Hall, 3rd edition, Chapter 11.

موضع:	وروودی/خروجی
شماره آزمایش:	۱۳
عنوان:	پیاده‌سازی مکانیزم‌های داوری گذرگاه
هدف:	

آشنایی با تداخل در ارتباطات مبتنی بر گذرگاه و استفاده از مکانیزم‌های داوری برای رفع تداخل همزمان

آمادگی پیش از آزمایش:

مطالعه مواردی که در بخش مراجع آزمایش به آن اشاره شده است و انجام برخی شبیه‌سازی‌های مورد نظر.

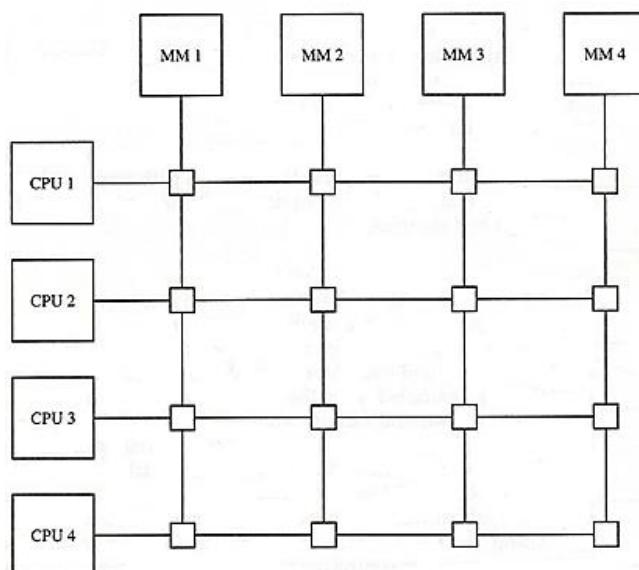
شرح آزمایش:

مکانیزم‌های داوری در گذرگاه مشترک:

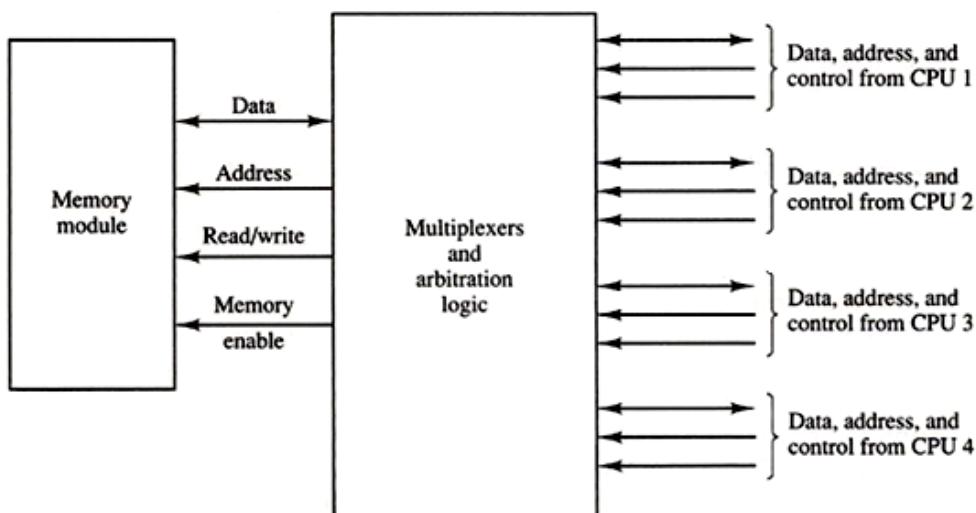
در بسیاری از سیستم‌های محاسباتی، چندین فرستنده و گیرنده داده به طور مشترک از یک گذرگاه داده استفاده می‌کنند؛ بنابراین لازم است برای جلوگیری از ایجاد اختلال میان داده‌هایی که که روی گذرگاه قرار می‌گیرد، در هر لحظه از زمان حداکثر یکی از سیستم‌ها روی گذرگاه داده قرار دهد. برای این منظور مکانیزم‌هایی جهت ایجاد نظم برای استفاده از باس مشترک طراحی شده است.

۱) استفاده از سوییچ تقاطعی (Crossbar switch)

روش سوییچ تقاطعی یکی از روش‌هایی است که برای ایجاد نظم ارتباط $M \times N$ واحد حافظه مورد استفاده قرار می‌گیرد. در این روش با استفاده از چند واحد تسهیم‌کننده (Multiplexer) و مدارات منطقی می‌توان چند ارتباط همزمان میان یک زوج پردازنده و واحد حافظه را ایجاد کرد. شکل ۱ بلوک سوییچ تقاطعی 4×4 را نشان می‌دهد و شکل ۲ جزئیات بیشتری از سیگنال‌های کنترلی مورد نیاز برای ارتباط 4×4 پردازنده با یک واحد حافظه مشترک به وسیله یک سوییچ تقاطعی را نشان می‌دهد.



شکل ۱ - یک سوییچ تقاطعی نوعی 4×4



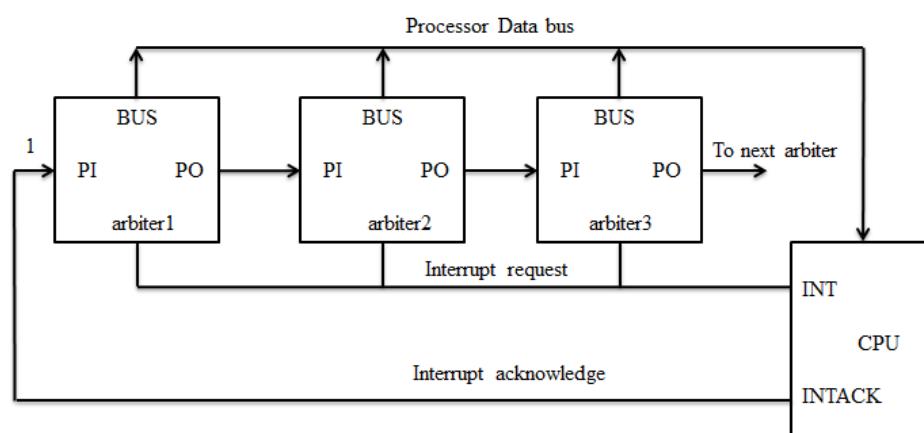
شکل ۲- بلوک نمودار یک سوییچ تقاطعی با جزئیات بیشتر

۲) روش‌های اولویت‌دهی

یکی دیگر از روش‌هایی که می‌توان انتقال داده‌های همزمان را نظم‌دهی کرد استفاده از اولویت‌دهی به دستگاه‌ها است. به این ترتیب که هر یک از واحدهایی که به باس داده متصل هستند دارای یک اولویت هستند و اگر دو بخش همزمان داده‌ای برای قرار دادن روی باس داشته باشند، باس در اختیار واحد با اولویت بیشتر خواهد بود.

۱-۲) روش زنجیره‌ای (سری)

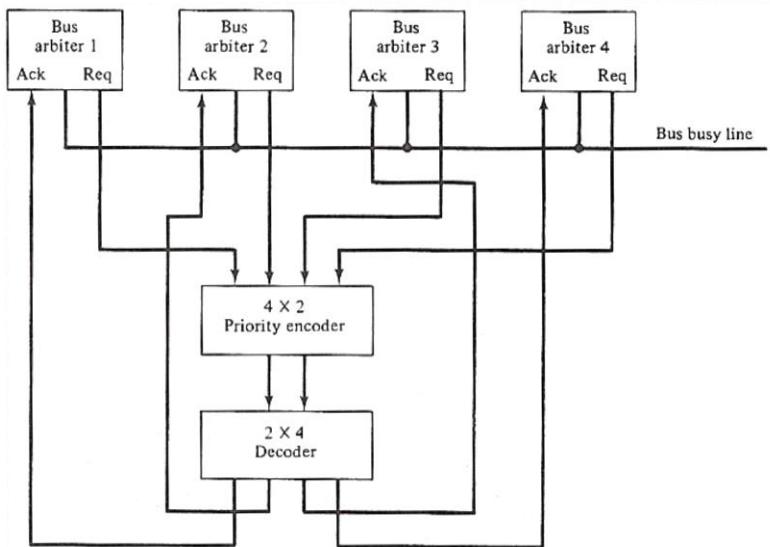
در این روش، چند واحد که به یک پردازنده متصل هستند در یک زنجیره در کنارهم قرار می‌گیرند. واحد با اولویت بیشتر در اولین محل قرار دارد و به ترتیب اولویت واحدها کم می‌شود. وقتی یکی از واحدها داده‌ای برای ارسال دارد، سیگنال وقفه پردازنده را فعال می‌کند. پردازنده پاسخ این وقفه را به اولین واحد ارسال می‌کند. در صورتی که این واحد دارای داده باشد آن را روی باس قرار خواهد داد و در غیر اینصورت به واحد بعد انتقال خواهد داد. شکل ۳ بلوک نمودار این روش را نشان می‌دهد.



شکل ۳- روش اولویت دهی زنجیره‌ای (daisy-chain)

۲-۲) روش موازی

در این روش واحدهایی دارای داده می‌توانند همزمان با هم درخواست ارسال دهند. این درخواست در یک کدگذار اولویت (Priority Encoder) وارد می‌شود و پس از تصمیم‌گیری راجع به واحدی که اولویت بیشتری دارد، پاسخ درخواست به واحد مربوطه ارسال می‌شود و پس از آن می‌تواند داده خود را روی باس قرار دهد. شکل ۴ بلوک نمودار این روش را نشان می‌دهد.



شکل ۴- روش اولویتدهی موازی

خروجی‌های مورد انتظار آزمایش:

لازم است هر گروه از دانشجویان از هریک از بخش‌های (۱) و (۲) را با استفاده از زبان توصیف سخت‌افزار پیاده‌سازی و شبیه‌سازی نمایند و در رابطه با زمان‌بندی سیگنال‌ها تحلیل ارائه دهند.

هر یک از گروه‌ها می‌توانند سه یا چهار entity جداگانه با عملکرد درونی انتخابی به عنوان واحدهای متصل به باس پیاده‌سازی کنند.

هر یک از واحدها در زمان‌های اختیاری درخواست باس به arbiter ارسال می‌کنند. در صورت پذیرش درخواست، واحد مورد نظر داده‌های تولیدی خود را با یک زمان‌بندی مشخص روی باس که خروجی اصلی entity است قرار خواهد داد.

مراجع مطالعه / ضمایم:

[1] M. Morris Mano, *Computer system architecture*, Prentice-Hall, 3rd edition, Chapter 11.

موضوع:	محاسبات ممیز شناور
شماره آزمایش:	۱۴
عنوان:	پیاده سازی عملیات جمع یا تفریق واحد ممیز شناور
هدف:	

در این آزمایش هدف طراحی و پیاده‌سازی یک واحد محاسبات ساده برای انجام عملیات جمع و تفریق اعداد ممیز شناور به صورت سخت‌افزاری است.

آمادگی پیش از آزمایش:

مطالعه مواردی که در یخش مراجع آزمایش به آن اشاره شده است و انجام پرخی شبیه‌سازی‌های مورد نظر الزامی است.

شرح آزمایش:

برای این منظور از استاندارد IEEE 754 که در آن نحوه نمایش اعداد ممیز شناور، نحوه انجام محاسبات و نحوه انجام اصلاحات بر روی نتایج شرح داده شده است، استفاده می‌شود. یک ALU ساده برای پیاده‌سازی عملیات زیر بر روی اعداد ممیز شناور بر اساس استاندارد IEEE 754 طراحی و پیاده‌سازی کنید.

الف) جمع **ب) تفريق**

مدارات طراحی شده را شبیه‌سازی کرده و در انتهای بر روی FPGA پیاده‌سازی کنید، برای سادگی نمایش اعداد ۸ بیت در نظر بگیرید. لازم است خروجی این واحد نیز به صورت استاندارد باشد و نیز کل طرح قابل سنتز باشد. برای تست سیستم، یک برنامه تست که مقادیر عددی، آنرا به صورت دستی، تعیین کرده‌اید، بنویسید.

خر و جی های مورد انتظار آزمایش:

با دادن اعداد ممین شناور، دلخواه، عملیات جمع و تفریق، روی آن‌ها انجام شود.

مراجع مطالعه / ضمایم:

[1] Mano, Kime, *Logic and Computer Design Fundamentals*, Prentice-Hall, 2004, chapter 10.

هدف:	آنها را آشنایی با پارامترهای کارآیی و نحوه اندازه‌گیری آنها
عنوان:	اندازه‌گیری پارامترهای CPI، IPC، MIPS و DMIPS یک پردازنده نوعی
شماره آزمایش:	۱۵
موضوع:	ارزیابی کارآیی

مطالعه منابع معرفی شده در انتهای آزمایش. در ابتدای جلسه سوالات کتبی یا شفاهی کوتاه صورت خواهد گرفت.

شرح آزمایش:

یکی از روش‌های مقایسه میان دو سیستم پردازنده‌ای، ارزیابی کارآیی آن‌هاست. برای این منظور پارامترهایی مانند تعداد دستورالعمل اجرا شده در واحد زمان (IPC)، زمان متوسط اجرای یک دستور العمل (CPI) تعریف شده است. برای قابل استناد بودن مقایسه‌های صورت گرفته، لازم است واحدهایی که با هم مقایسه می‌شوند برنامه‌های یکسان و استانداردی را اجرا نمایند. مجموعه برنامه‌های SPEC و MiBench از متداول‌ترین برنامه‌های استاندارد هستند که دسته اول برای پردازنده‌های Desktop و دسته دوم برای پردازنده‌های Embedded مورد استفاده قرار می‌گیرند.

الف) آشنایی با برنامه‌های استاندارد

در این آزمایش لازم است هر گروه از دانشجویان ۴ نمونه از برنامه‌های مجموعه SPEC2000 یا SPEC2006 را روی سیستم خود اجرا نمایند و زمان اجرای آن را محاسبه کنند. پس از آن هر دو گروه نتایج به دست آمده خود را مقایسه کرده و مورد بررسی قرار دهند.

ب) محاسبه پارامترهای ارزیابی کارآیی

در این آزمایش هر گروه از دانشجویان با هماهنگی مربی آزمایشگاه روشی برای محاسبه IPC، CPI، MIPS و DMIPS سیستم خود ارائه می‌دهند و پس از آن این پارامترها را برای سیستم خود محاسبه کرده و نتایج را با سایر گروه‌ها مقایسه می‌کنند.

ج) ارزیابی تأثیر برنامه‌نویسی چند نخی برای افزایش سرعت برنامه *

در این آزمایش هر گروه از دانشجویان ابتدا یک برنامه را به صورت تک نخی روی سیستم خود اجرا می‌کنند. در مرحله بعد دانشجویان تلاش می‌کنند همان برنامه را با تکنیک‌های برنامه‌نویسی چند نخی پیاده‌سازی و اجرا کنند و تأثیر موازی‌سازی در اجرای برنامه را گزارش نمایند.

د) استفاده از کتابخانه MPI برای برنامه‌نویسی موازی *

ه) استفاده از ابزار Simple Scalar برای ارزیابی کارآیی یک سیستم نوعی *

خروجی‌های مورد انتظار آزمایش:

مقدارهای عددی بدست آمده برای هر یک از پارامترهای فوق در هر مرحله به همراه توضیح دقیق و فنی روشی که از آن استفاده کرده‌اید.

مراجع مطالعه/ضمایم:

[1] John. L. Hennessy and David A. Patterson, *Computer Architecture, A Quantitative Approach*, Prentice-Hall, 4th Edition, pp. 28 -37.

هدف:	طراحی سلسله مراتب حافظه
شماره آزمایش:	۱۶
عنوان:	حافظه ها

هدف از این آزمایش، آشنایی با سلسله مراتب حافظه و نحوه نگاشت آدرس و دریافت داده از RAM و Cache است.

آمادگی پیش از آزمایش:

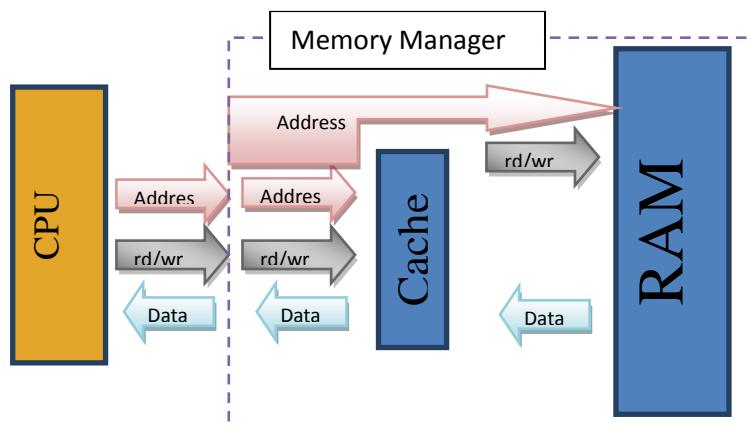
مطالعه مواردی که در بخش مراجع آزمایش به آن اشاره شده است و انجام برخی شبیه‌سازی‌های مورد نظر.

شرح آزمایش:

حافظه نهان (Cache) یکی از مهمترین راهکارهایی است که برای کاهش اختلاف سرعت پردازنده و حافظه RAM ارائه شده و طراحی می‌شود. این حافظه به صورت یک واحد میانی در کنار پردازنده قرار می‌گیرد و سعی می‌کند بخشی از داده‌هایی که پردازنده به صورت متناوب به آن نیاز دارد را در خود ذخیره کند تا در زمان نیاز پردازنده بتواند در زمان کوتاهی آن را فراهم نماید.

الف) چگونگی عملکرد حافظه سلسله مراتبی:

شکل زیر نمودار بلوکی یک حافظه سلسله مراتبی را نشان می‌دهد. مطابق این شکل، پردازنده آدرس داده مورد نظر خود را به همراه سیگنال خواندن/نوشتن به واحد مدیریت حافظه ارسال می‌کند. این واحد کنترل Cache و RAM را بر عهده دارد.



شکل ۱- دیاگرام بلوکی سلسله مراتب حافظه در ارتباط با پردازنده

در ابتدا آدرس‌ها به Cache داده می‌شوند و هر یک از حالات زیر امکان رخداد خواهد داشت:

۱. خواندن آدرسی که در Cache وجود دارد: در این حالت، داده مورد نظر بلافارصله برگردانده می‌شود.
۲. خواندن آدرسی که در Cache وجود ندارد: در این حالت، لازم است آدرس مورد نظر به RAM ارجاع شود و داده از RAM خواند شود. همچنین این داده باید درون Cache نیز به روزرسانی شود.

۳. نوشتن در آدرسی که در Cache وجود دارد: در این صورت داده جدید در Cache و RAM به طور همزمان جایگزین می‌شود. (مکانیزم Write Through)

۴. نوشتن در آدرسی که در Cache وجود ندارد: در این حالت داده مورد نظر تنها در RAM نوشته خواهد شد.

ب) طراحی سلسله مراتب حافظه:

با مشخصات داده شده در (الف) یک نمونه ساده از سلسله مراتب حافظه (Memory Hierarchy) طراحی و پیاده‌سازی کنید. طول آدرس پردازنده، سایز RAM و Cache و بلوک‌بندی را به گونه‌ای منطقی طراحی کند و تعداد بیت‌های بخش‌های Index، Tag و Offset را محاسبه نمایید. مکانیزم شرکت‌پذیری Cache را به صورت Direct map در نظر بگیرید.

خروجی‌های مورد انتظار آزمایش:

هریک از بخش‌های این آزمایش در قالب‌های زیر به ترتیب تحويل مدرس آزمایشگاه گردد:

- طراحی شماتیک طرح (محتوای توصیف) بر روی کاغذ
- پیاده‌سازی مدارها با استفاده از زبانهای توصیف
- شبیه‌سازی مدارهای توصیف شده و تهییه Testbench برای آن و نشان دادن درستی عملکرد آنها.

مراجع مطالعه/ضمایم:

[1] J. L. Hennessy and D. A. Patterson, "Computer Architecture, A Quantitative Approach" 4th edition, appendix C.

ضمیمه ۱

توصیف سیستم با VHDL

توصیف یک سیستم با VHDL : موجودیت سخت افزاری

```
Entity mobile is
```

```
...
```

```
End entity mobile;
```

هر سیستم شامل دو جزء اصلی: ورودی/خروجی و بدن سیستم

مثل: CPU, RAM, مودم، ماوس، کامپیووتر، موبایل و مانند آن.

توصیف یک سیستم با VHDL : پورت‌های ورودی و خروجی

```
Entity and_gate is
```

```
Port(
```

```
    A, B: in bit;
```

```
    C : out bit
```

```
);
```

```
End entity and_gate;
```

توصیف یک سیستم با VHDL : بدن سیستم

```
Entity and_gate is
```

```
Port(
```

```
    A, B: in bit;
```

```
    C : out bit
```

```
);
```

```
End entity and_gate;
```

```
Architecture behavioral of and_gate is
```

```
Begin
```

```
    C <= A and B;
```

```
End behavioral;
```

در مثال بالا:

And_gate	نام ماجولی که آن را توصیف می‌کنیم. این ماجول یک گیت AND است.
A, B	دو ورودی گیت AND
C	خروجی گیت AND
behavioral	سطحی که این کد را در آن توصیف می‌کنیم.
bit	یک نوع (type) است. مثل int در زبان C. سیگنال‌های از این نوع مقادیر '0' یا '1' می‌گیرند.
<=	عملگر مقداردهی است.

معرفی کتابخانه IEEE

توصیف یک سیستم با VHDL : بدنه سیستم

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
Entity and_gate is
Port(
    A, B: in std_logic;
    C : out std_logic
);
End entity and_gate;
```

```
Architecture behavioral of and_gate is
Begin
    C <= A and B;
End behavioral;
```

استفاده از std_logic_vector

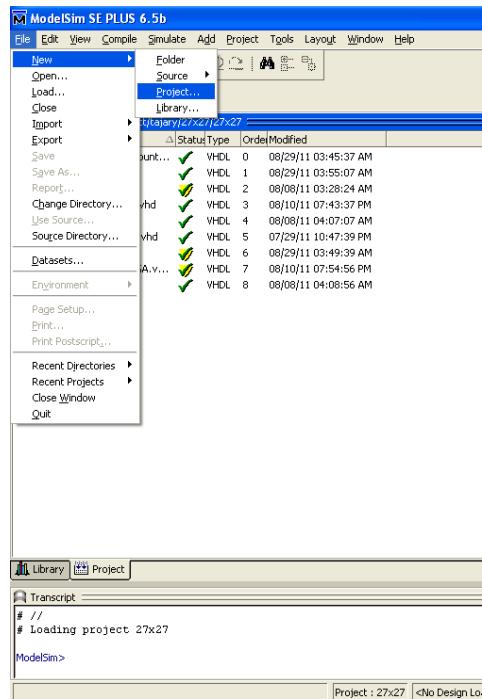
```
Port(
    A: in std_logic_vector(9 downto 0);
    C: out std_logic_vector (9 doento 0)
);
Signal b: std_logic_vector(9 downto 0);
```

مقداردهی به std_logic_vector

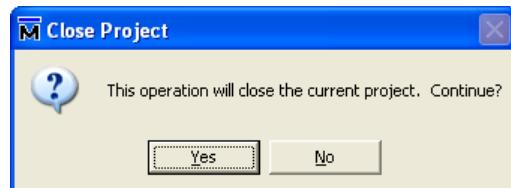
```
B(9 downto 1) <= B(8 downto 0); -- shift
B(0) <= '1';
B(2 downto 0) <= "110";
B <= A;
B(9 downto 8) <= A(2 downto 1);
B(4) <= B(5);
B <= (others => '0');
B(5 downto 0) <= A(9 downto 7) & "011";
```

ضمیمه ۲

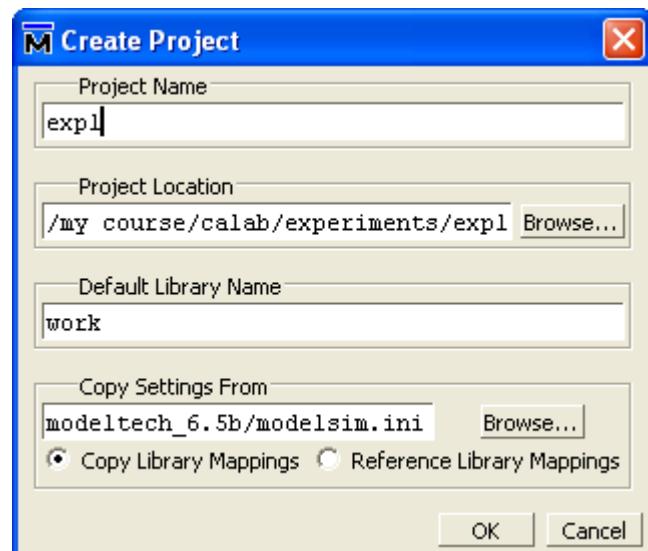
شبیه‌سازی با استفاده از
ModelSim



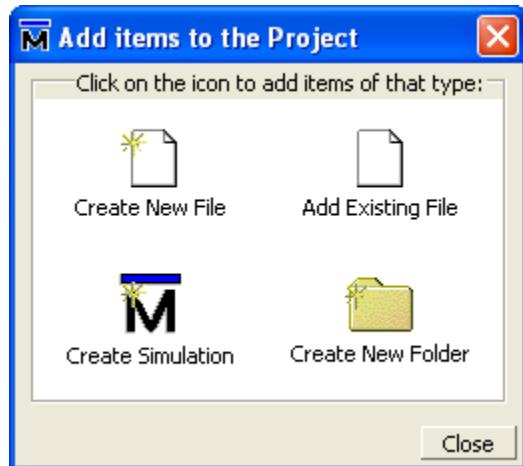
شکل ۱ ایجاد پروژه جدید



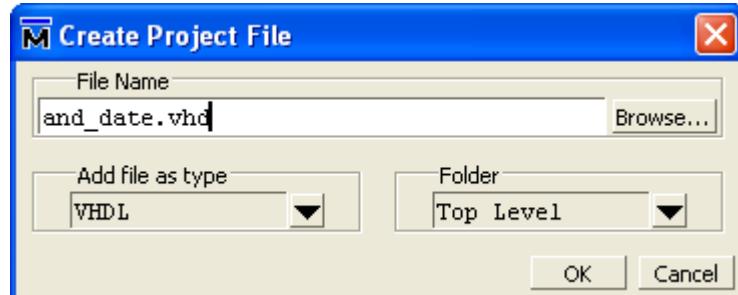
شکل ۲ پیغام هشدار بستن پروژه قبلی



شکل ۳ تعیین نام و پوشش پروژه



شکل ۴ افزودن فایل به پروژه



شکل ۵ تعیین نوع و نام فایل جدید

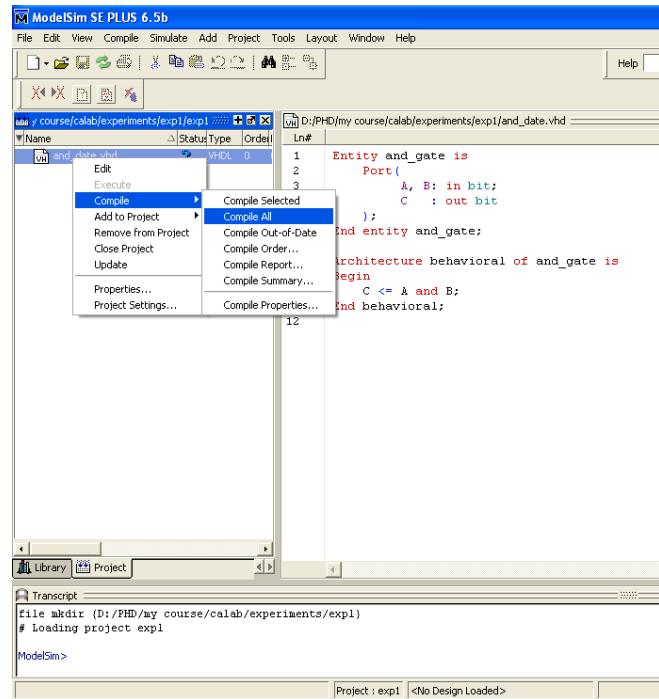
```

ModelSim SE PLUS 6.5b
File Edit View Compile Simulate Add Source Tools Layout Window Help
D:\PHD\my course\calab\experiments\exp1\exp1\and_date.vhd
Ln# 1 Entity and_gate is
2   Port(
3     A, B: in bit;
4     C : out bit
5   );
6 End entity and_gate;
7
8 Architecture behavioral of and_gate is
9 Begin
10   C <= A and B;
11 End behavioral;
12

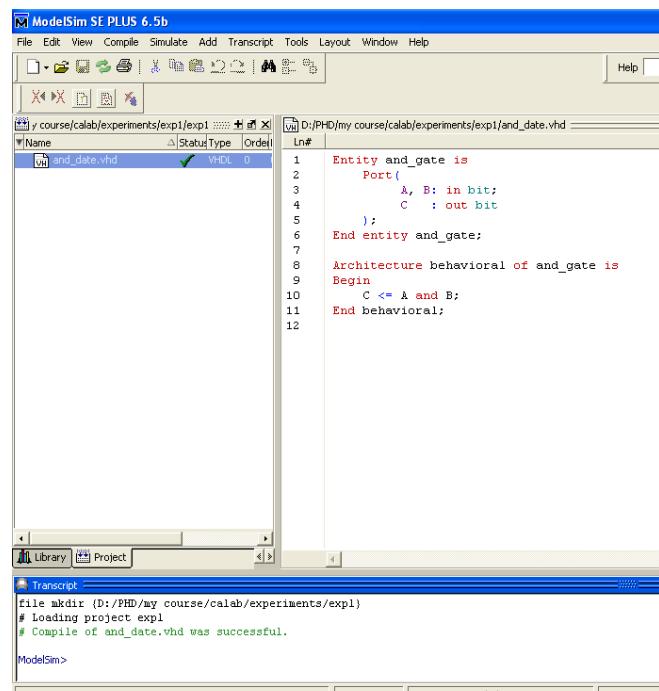
```

The screenshot shows the ModelSim SE PLUS 6.5b interface. The menu bar includes File, Edit, View, Compile, Simulate, Add, Source, Tools, Layout, Window, and Help. The toolbar has various icons for file operations. The main window displays a VHDL code for an AND gate. The left pane shows a project tree with 'and_date.vhd' selected. The bottom pane shows a transcript of terminal commands: 'file mkdir (D:/PHD/my course/calab/experiments/exp1)' and '# Loading project exp1'. The status bar at the bottom indicates 'In: 5 Col: 6' and 'Project : exp1 <No Design Loaded>'.

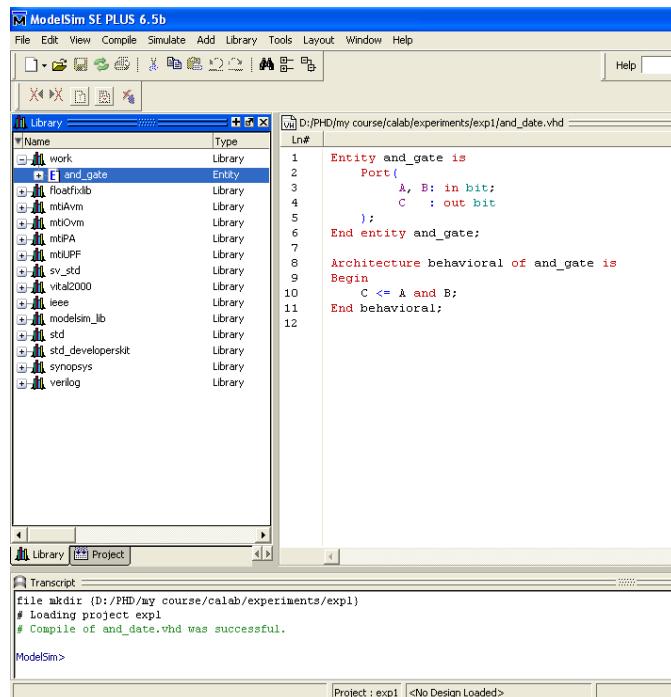
شکل ۶ نوشتن کد در ویرایشگر



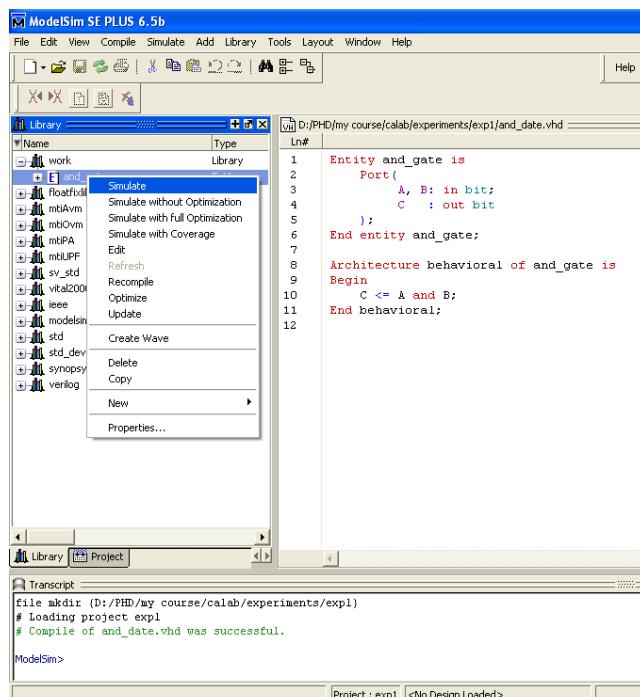
شکل ۷ منو کامپایل



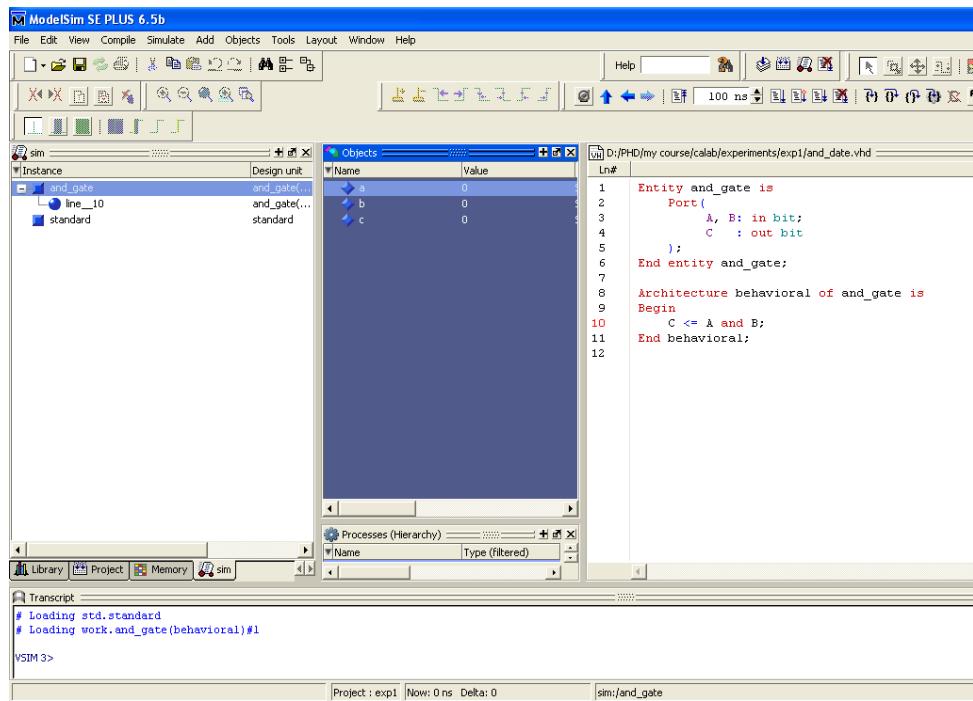
شکل ۸ نتیجه کامپایل



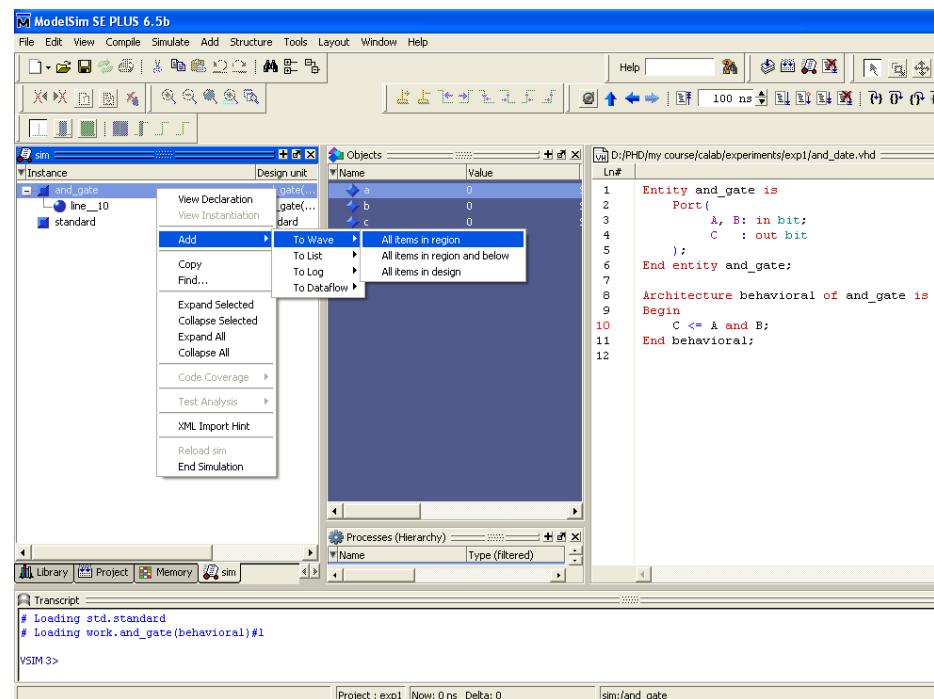
شکل ۹ بخش library برای شبیه‌سازی



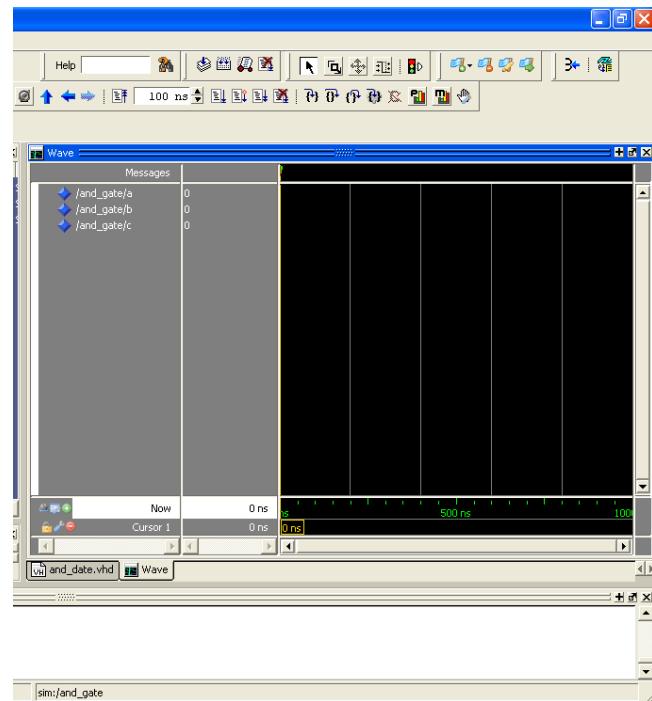
شکل ۱۰ منو شبیه‌سازی



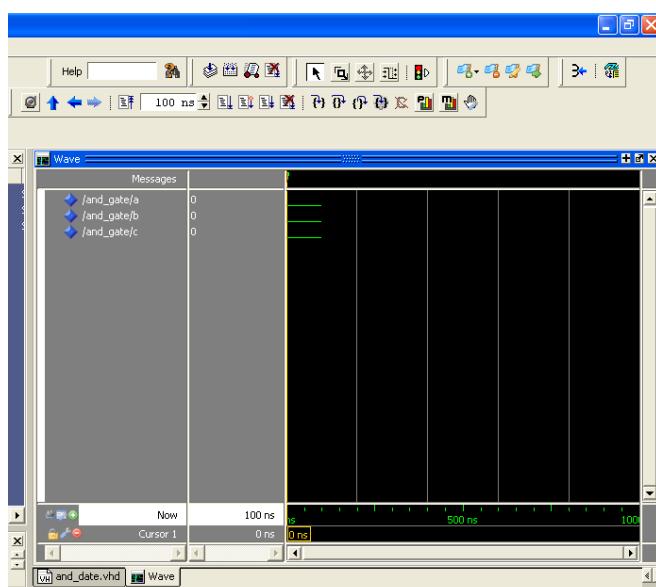
شکل ۱۱ نمای شروع شبیه‌سازی



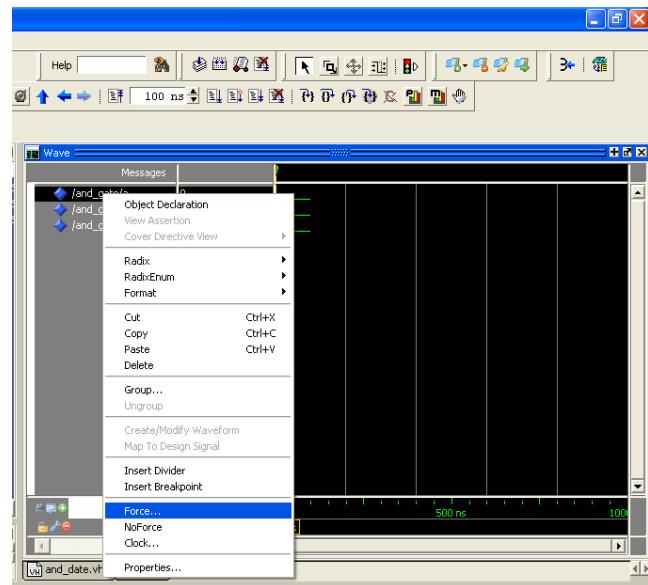
شکل ۱۲ افزودن سیگنال به پنجره wave



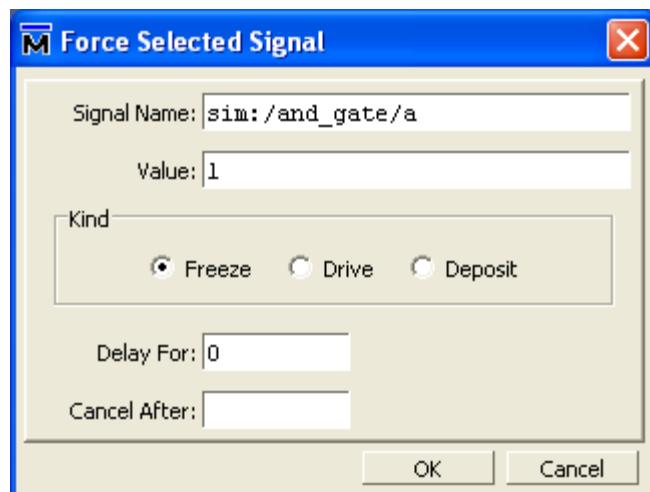
شکل ۱۳ سیگنال‌ها در پنجره wave



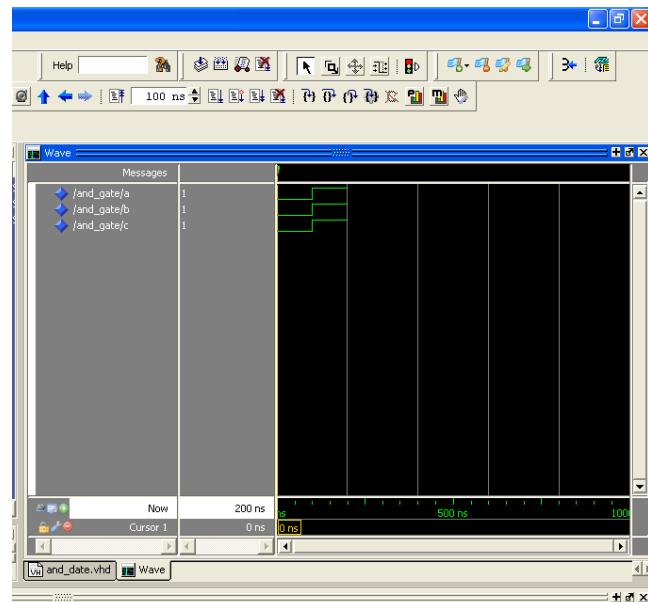
شکل ۱۴ اجرای شبیه‌سازی



شكل ۱۵ منوی force کردن سیگنال



شكل ۱۶ تعیین مقدار سیگنال در پنجره force

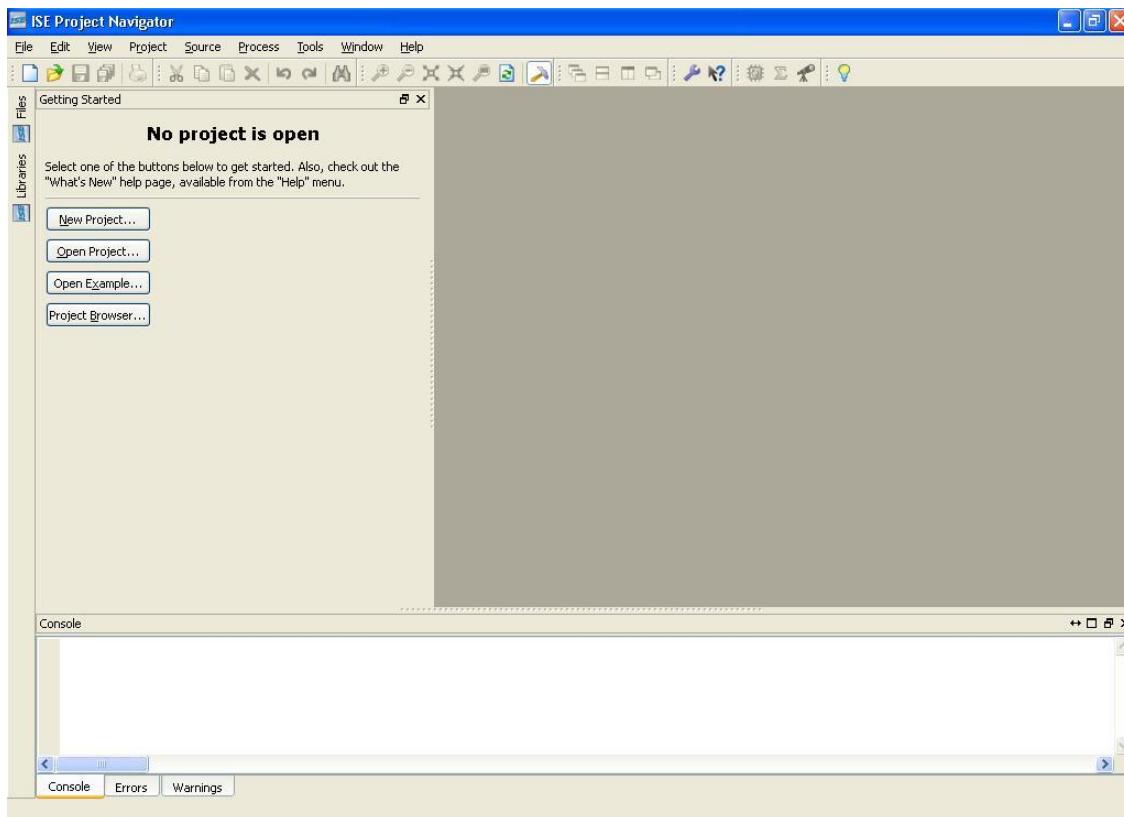


شکل ۱۷ اجرای شبیه‌سازی با مقادیر جدید

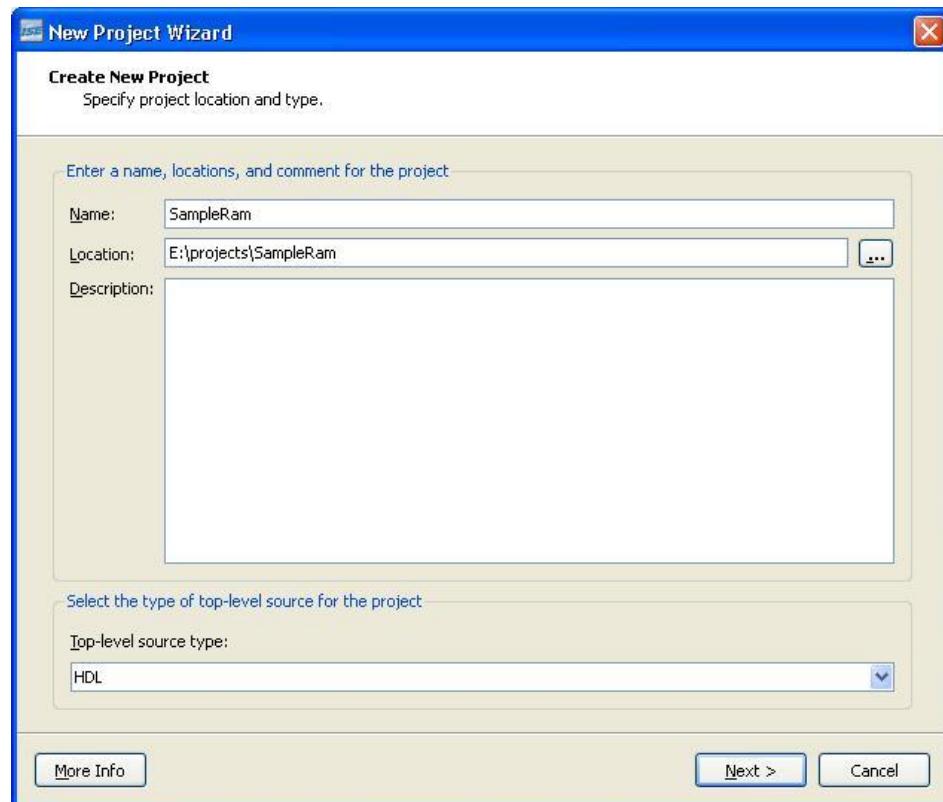
ضمیمه ۳

نحوه کارکردن با بورد FPGA

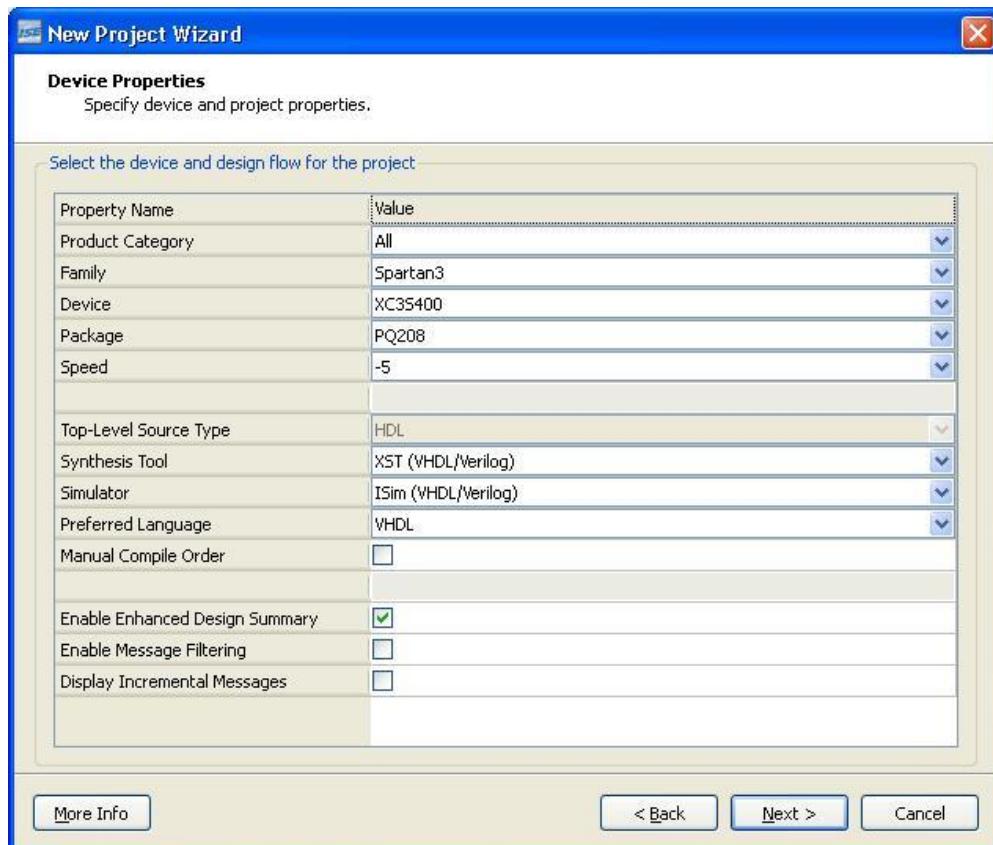
فاز اول: برنامه‌ریزی FPGA



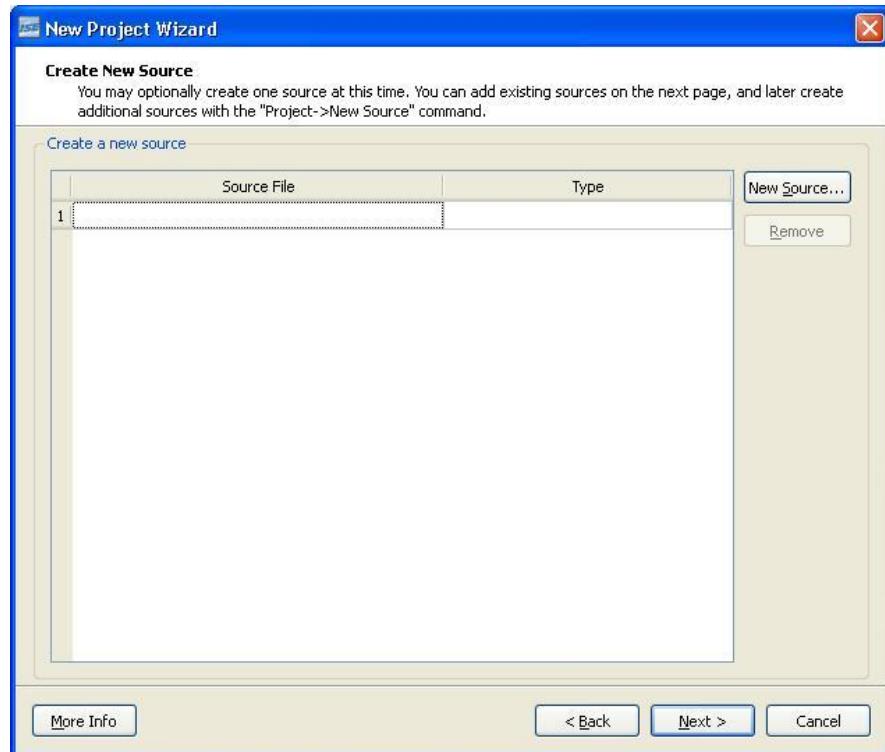
شکل ۱ صفحه شروع کار با ابزار ISE



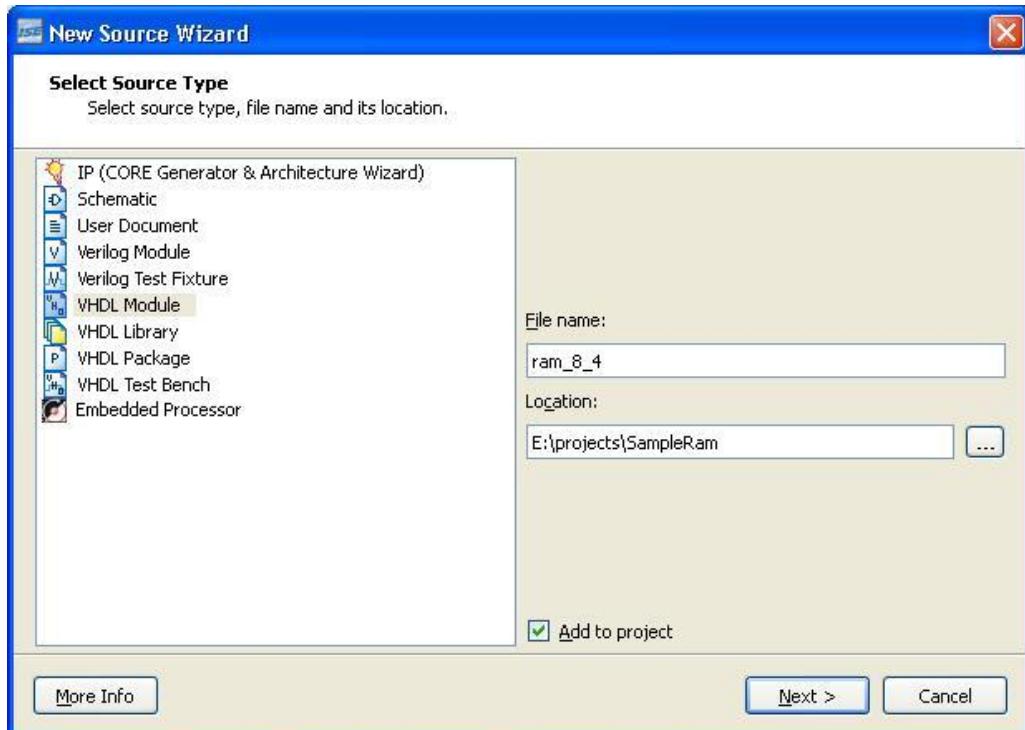
شکل ۲ ایجاد یک پروژه جدید



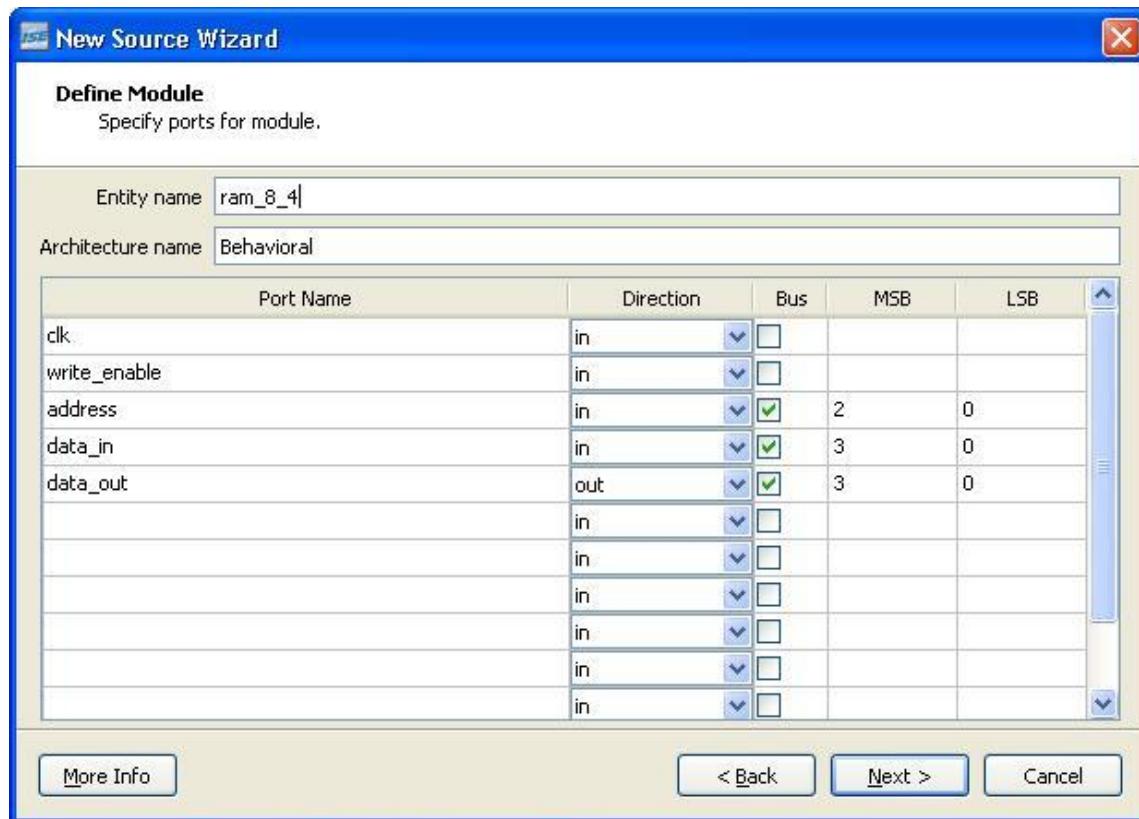
شکل ۳ تعیین ویژگی های FPGA



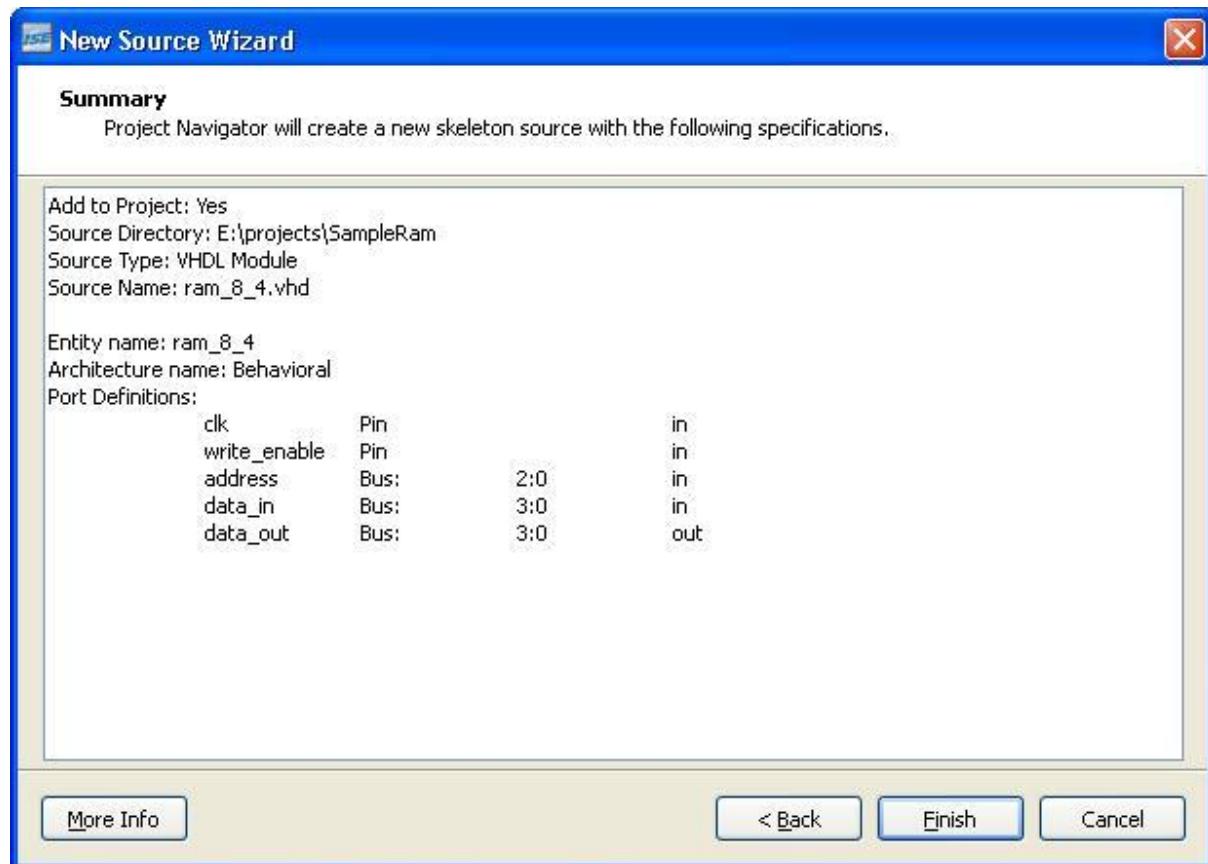
شکل ۴ ایجاد یک فایل برای entity



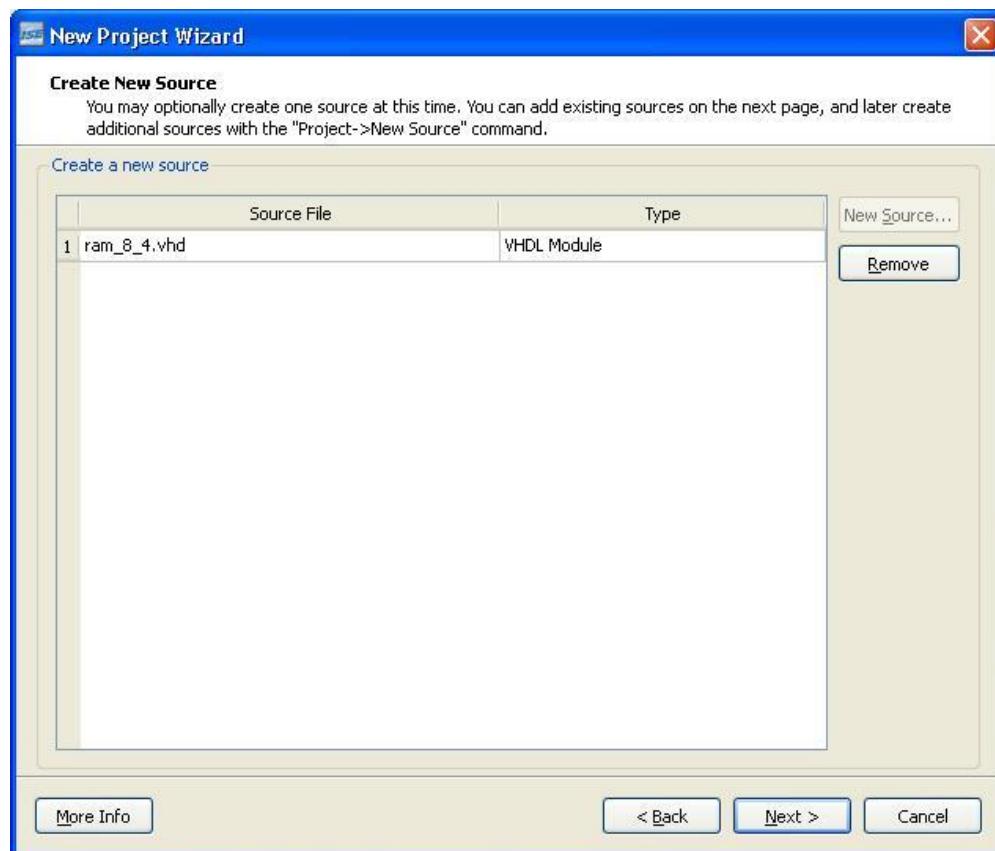
شکل ۵ تعیین نوع و نام فایل



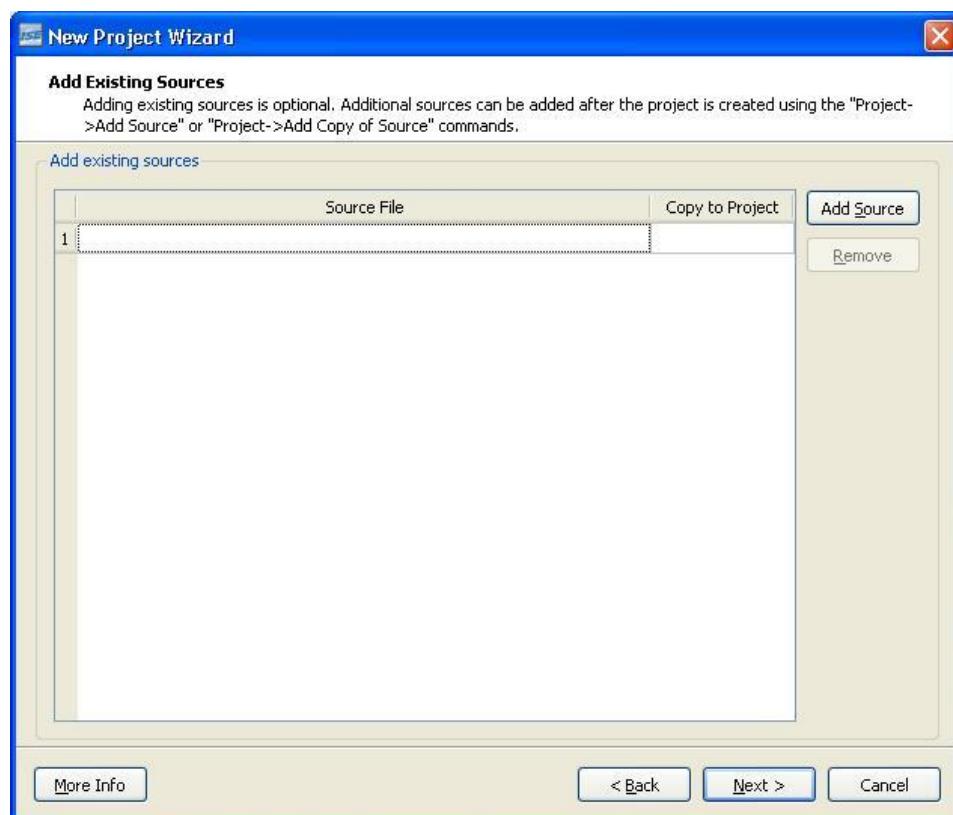
شکل ۶ تعیین پورت های ورودی و خروجی



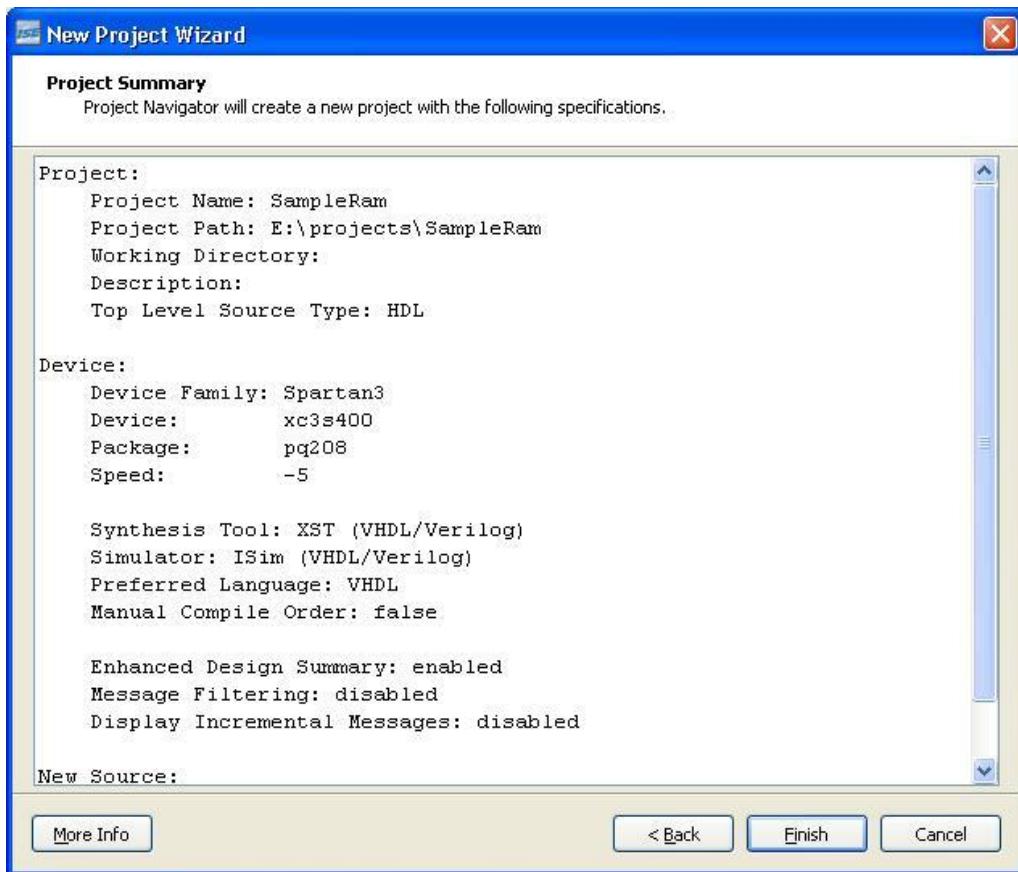
شکل ۷ خلاصه از ویژگی های فایل ایجاد شده



شکل ۸ لیست فایل های ایجاد شده



شکل ۹ افزودن فایل هایی که از قبل وجود دارند



شکل ۱۰ خلاصه ای از ویژگی های پروژه ایجاد شده

```

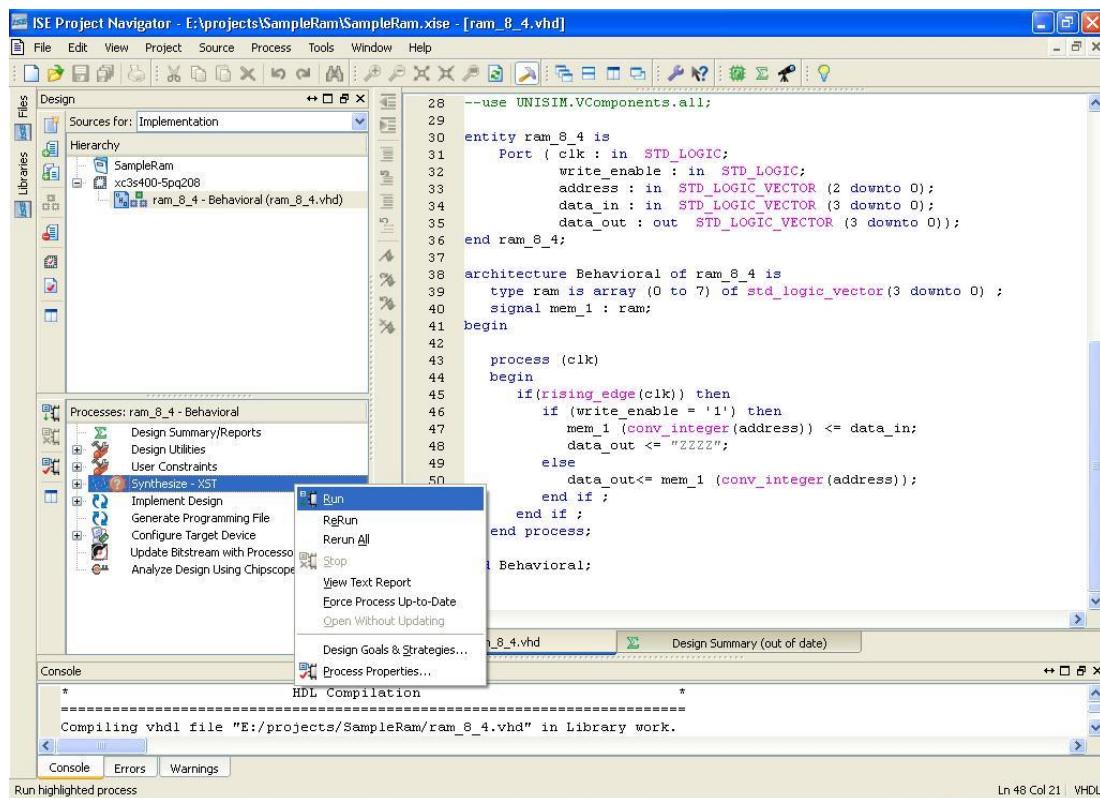
--use UNISIM.VComponents.all;
entity ram_8_4 is
    Port ( cik : in STD_LOGIC;
           write_enable : in STD_LOGIC;
           address : in STD_LOGIC_VECTOR (2 downto 0);
           data_in : in STD_LOGIC_VECTOR (3 downto 0);
           data_out : out STD_LOGIC_VECTOR (3 downto 0));
end ram_8_4;

architecture Behavioral of ram_8_4 is
begin
    process (cik)
    begin
        if(rising_edge(cik)) then
            if (write_enable = '1') then
                mem_1 (conv_integer(address)) <= data_in;
                data_out <= "ZZZZ";
            else
                data_out <= mem_1 (conv_integer(address));
            end if ;
        end if ;
    end process;
end Behavioral;

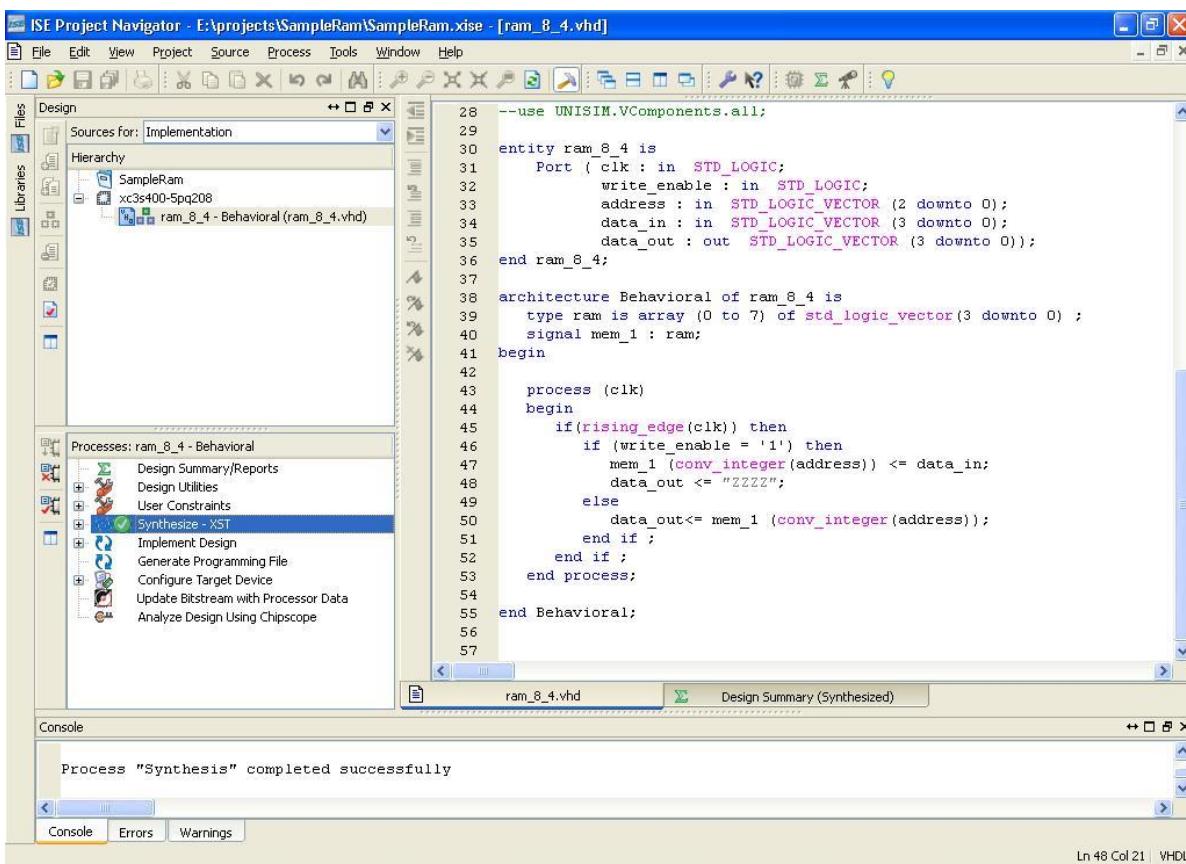
```

The screenshot shows the ISE Project Navigator interface with the following details:
- **Title Bar:** ISE Project Navigator - E:\projects\SampleRam\SampleRam.xise - [ram_8_4.vhd]
- **Menu Bar:** File, Edit, View, Project, Source, Process, Tools, Window, Help
- **Toolbars:** Standard and Advanced
- **Left Panel:** Files, Libraries, Hierarchy (Shows SampleRam, xc3s400-5pq208, ram_8_4 - Behavioral (ram_8_4.vhd)), Processes (Shows Design Summary/Reports, Design Utilities, User Constraints, Synthesize-XST, Implement Design, Generate Programming File, Configure Target Device, Update Bitstream with Processor Data, Analyze Design Using ChipScope), and Console.
- **Main Area:** Code Editor showing the VHDL code for the ram_8_4 entity.
- **Bottom Panel:** Console showing the compilation log: "HDL Compilation", "Compiling vhdl file "E:/projects/SampleRam/ram_8_4.vhd" in Library work.", and tabs for Console, Errors, and Warnings.

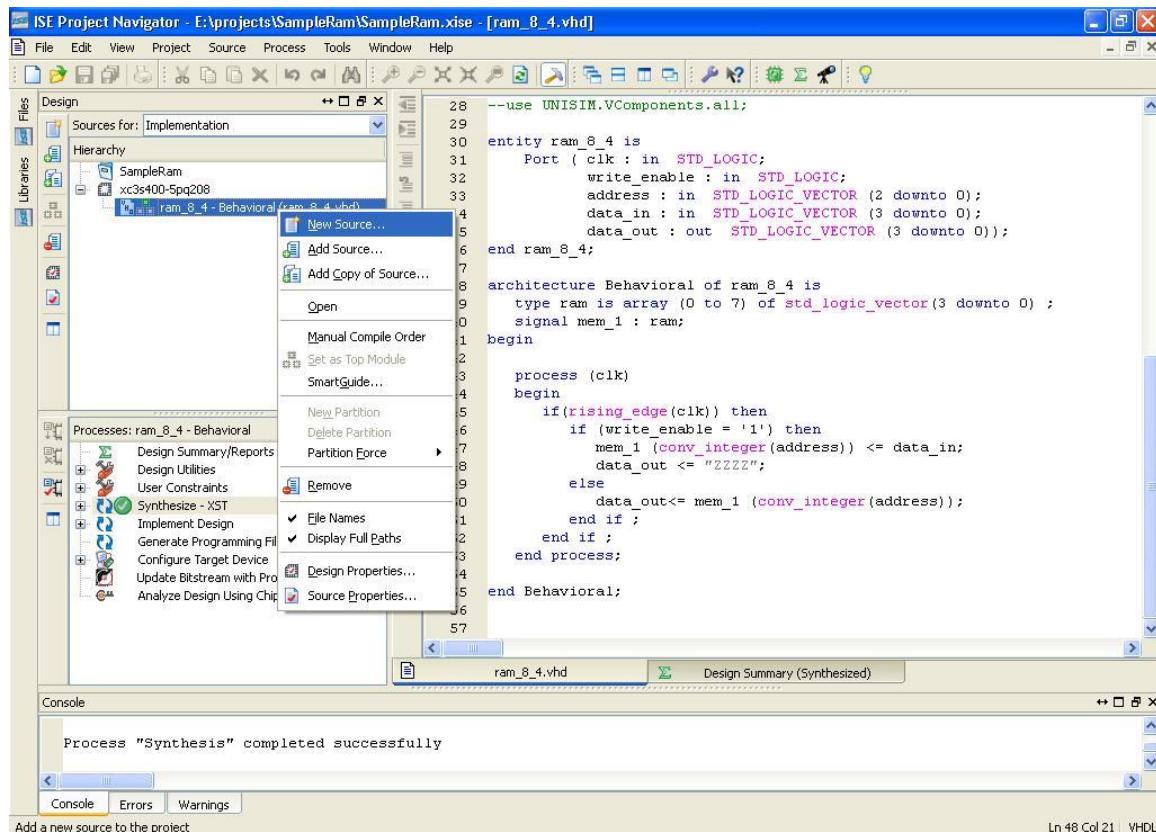
شکل ۱۱ نوشتن کد طراحی



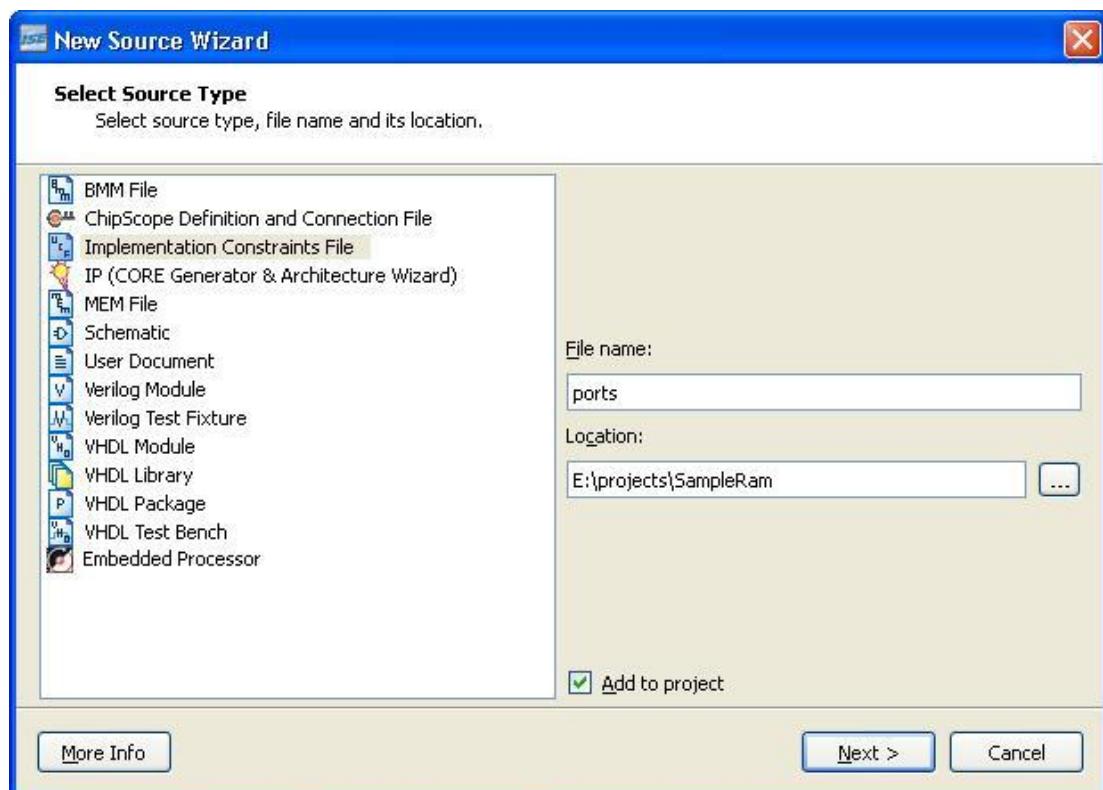
شکل ۱۲ منوی سنتز



شکل ۱۳ نتیجه سنتز



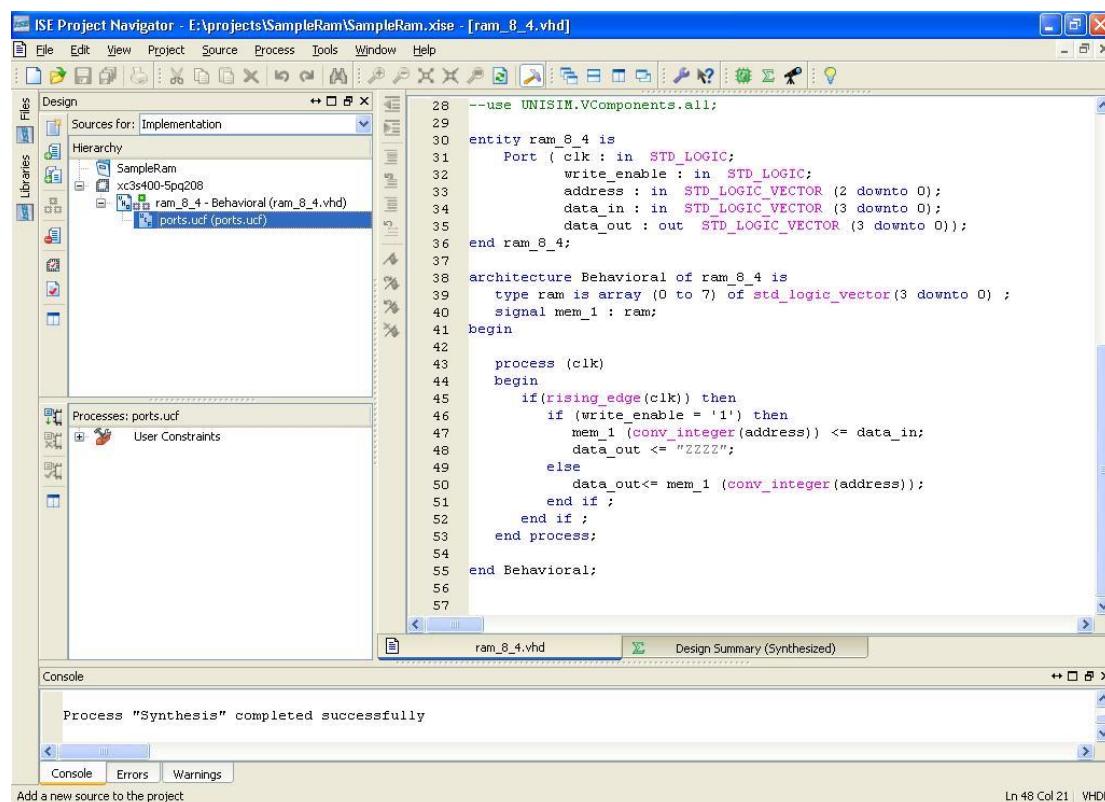
شکل ۱۴ منوی ایجاد فایل محدودیت (تعیین پورت های FPGA)



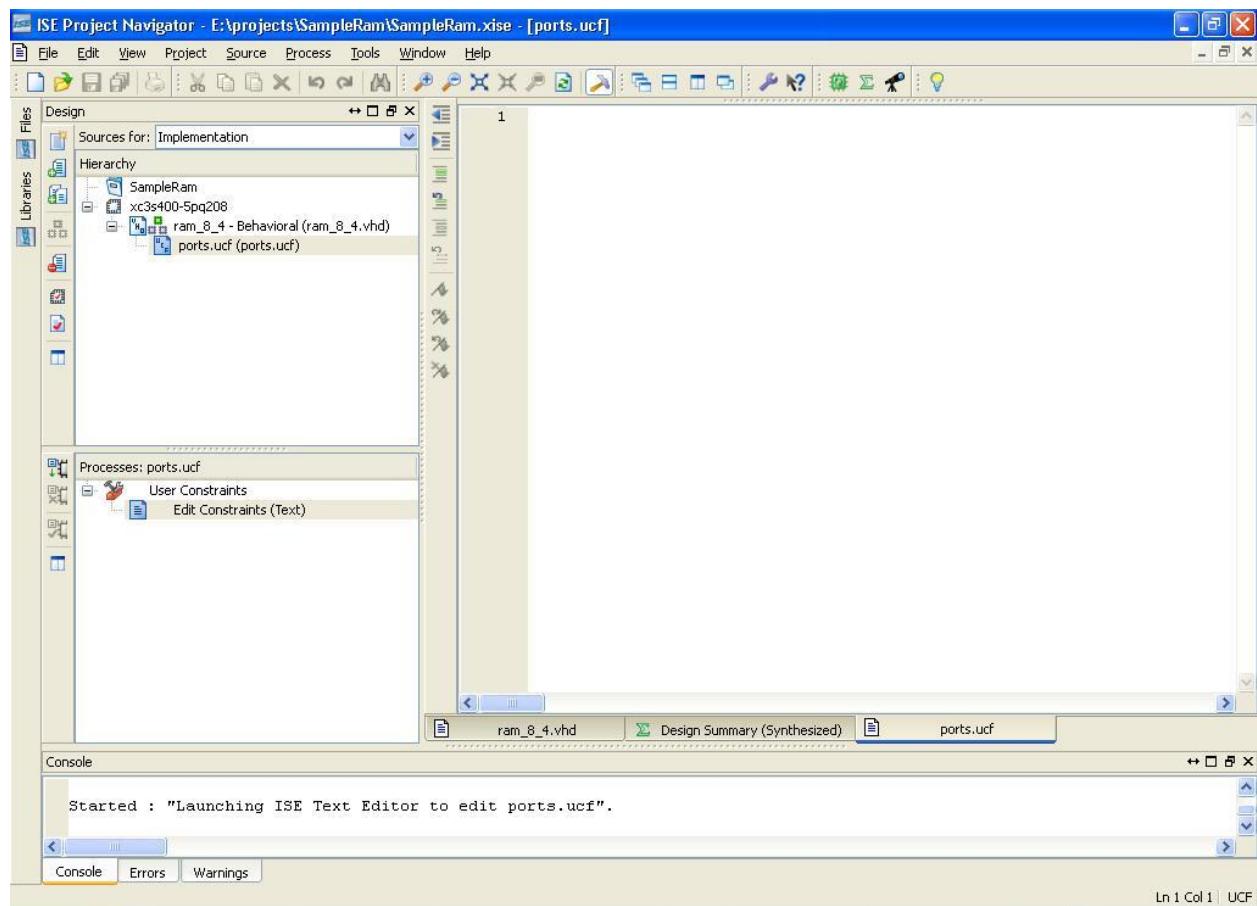
شکل ۱۵ تعیین نام فایل محدودیت



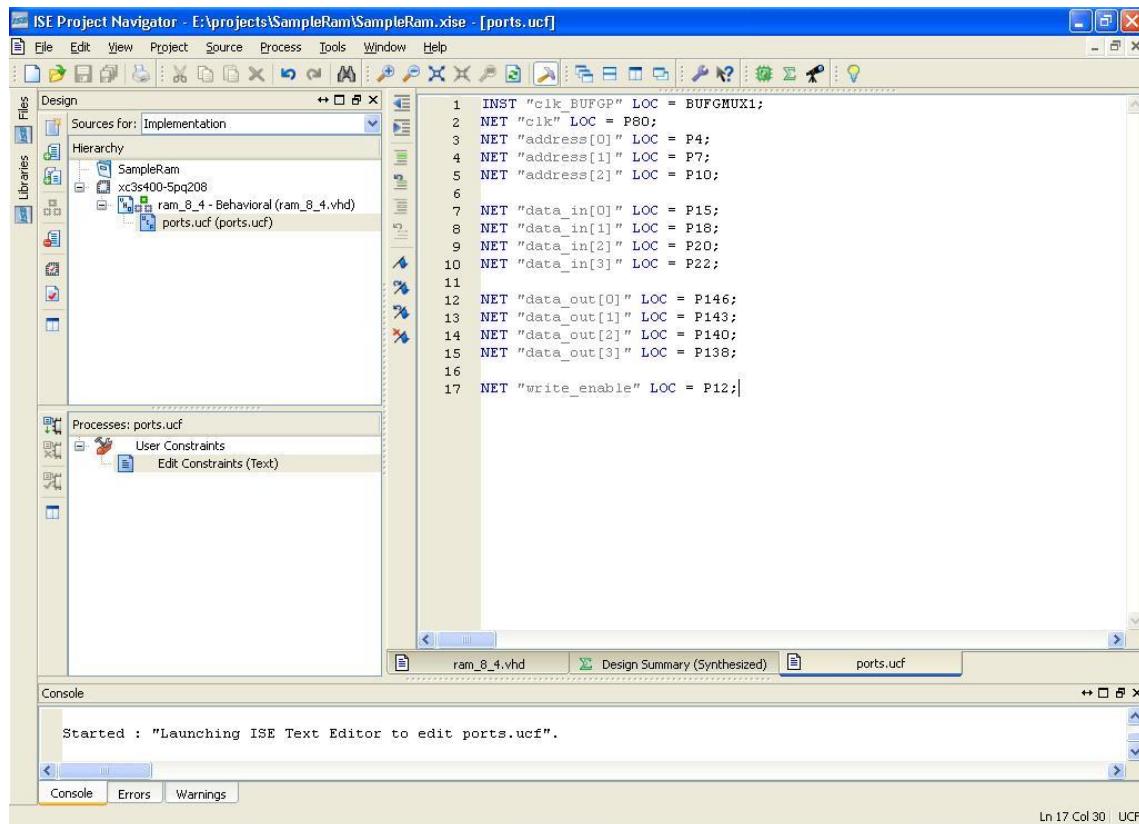
شکل ۱۶ ویزگی های فایل محدودیت ایجاد شده



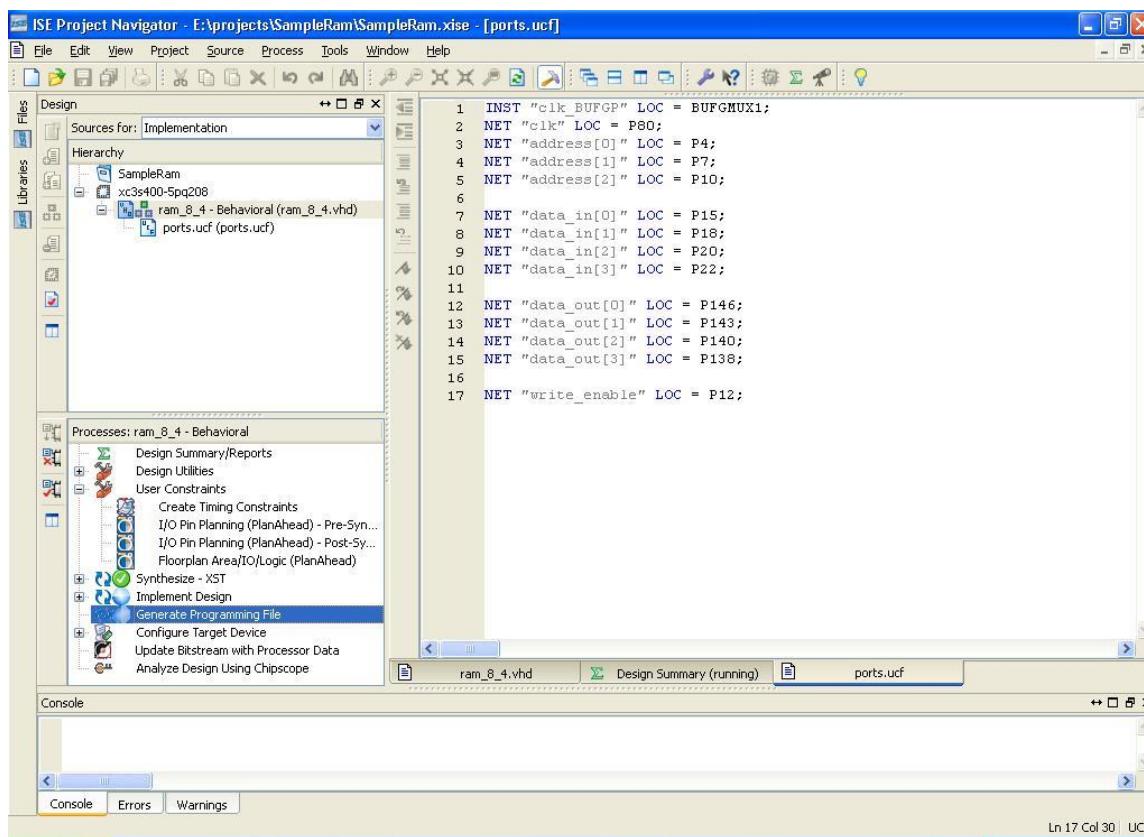
شکل ۱۷ فایل محدودیت در منوی Hierarchy



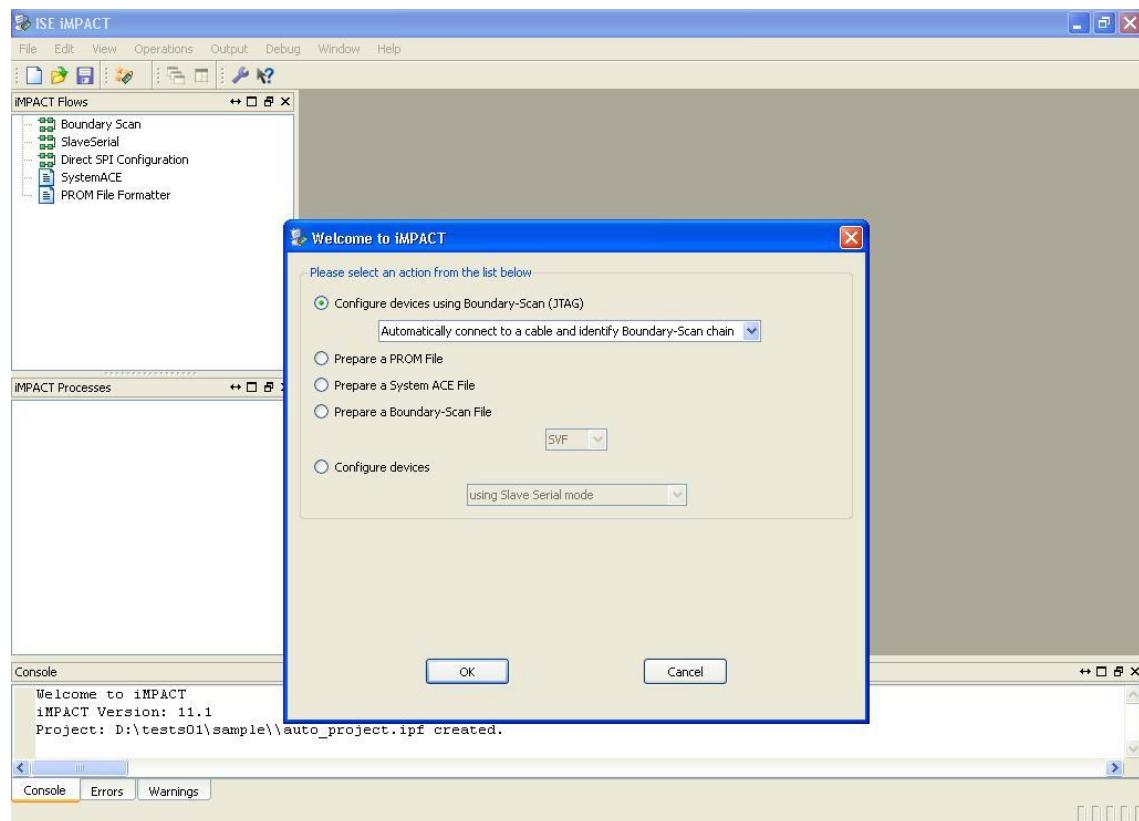
شکل ۱۸ ویرایش فایل محدودیت به صورت متنی



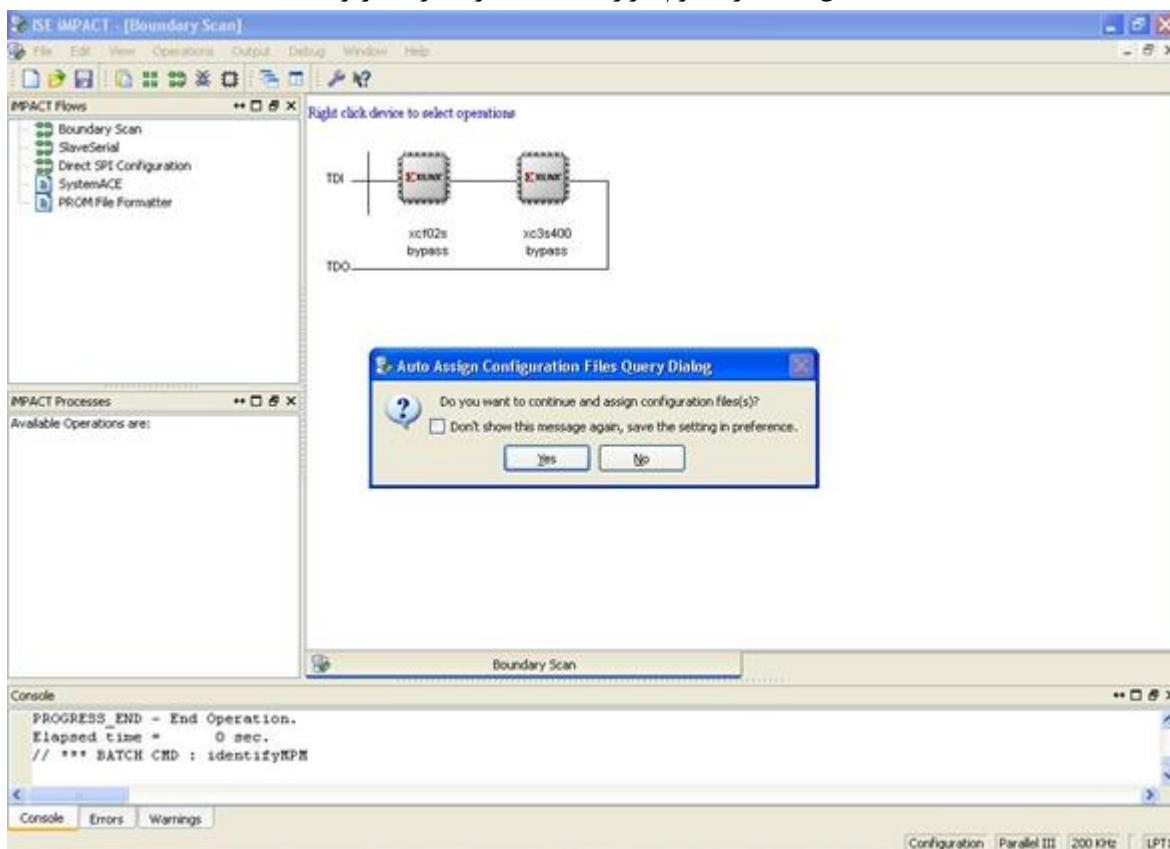
شکل ۱۹ تعیین port های طرح بر روی FPGA



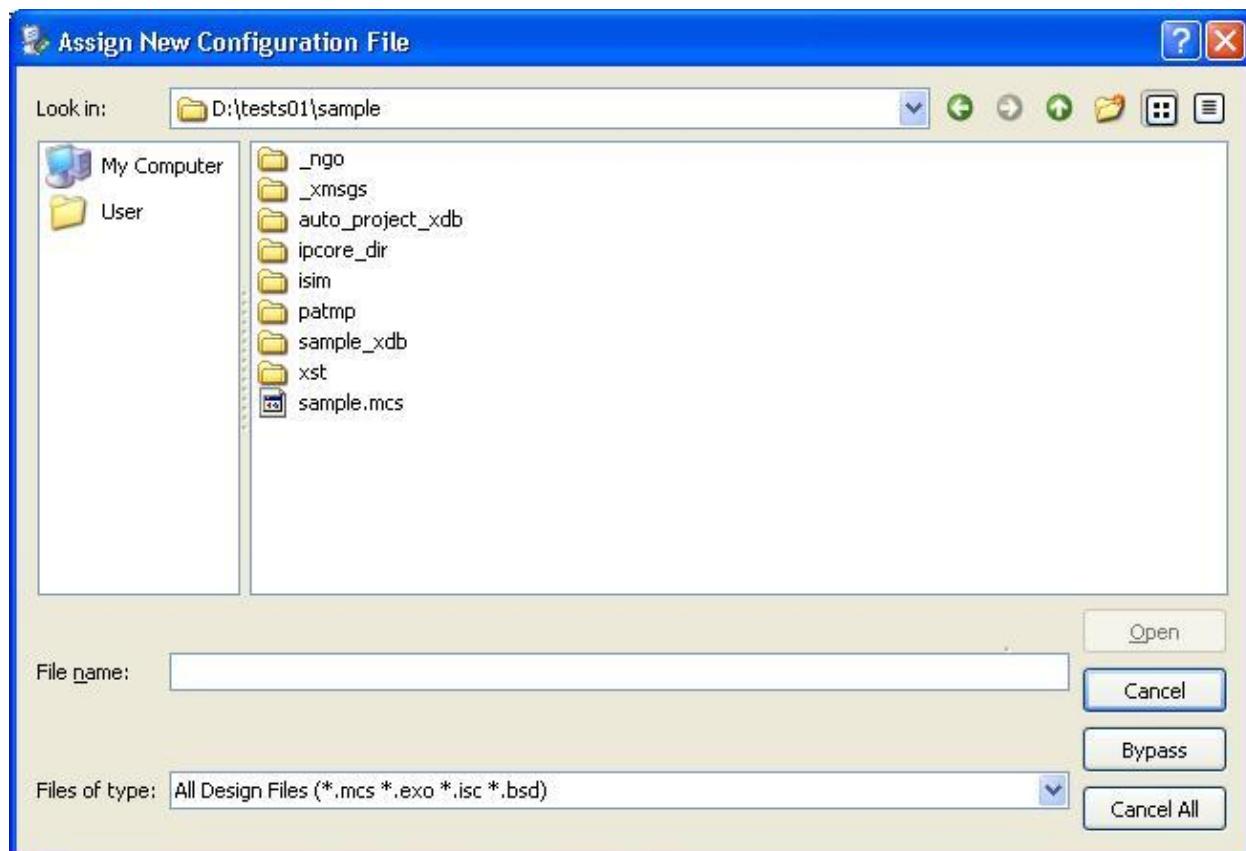
شکل ۲۰ ایجاد فایل برنامه ریزی



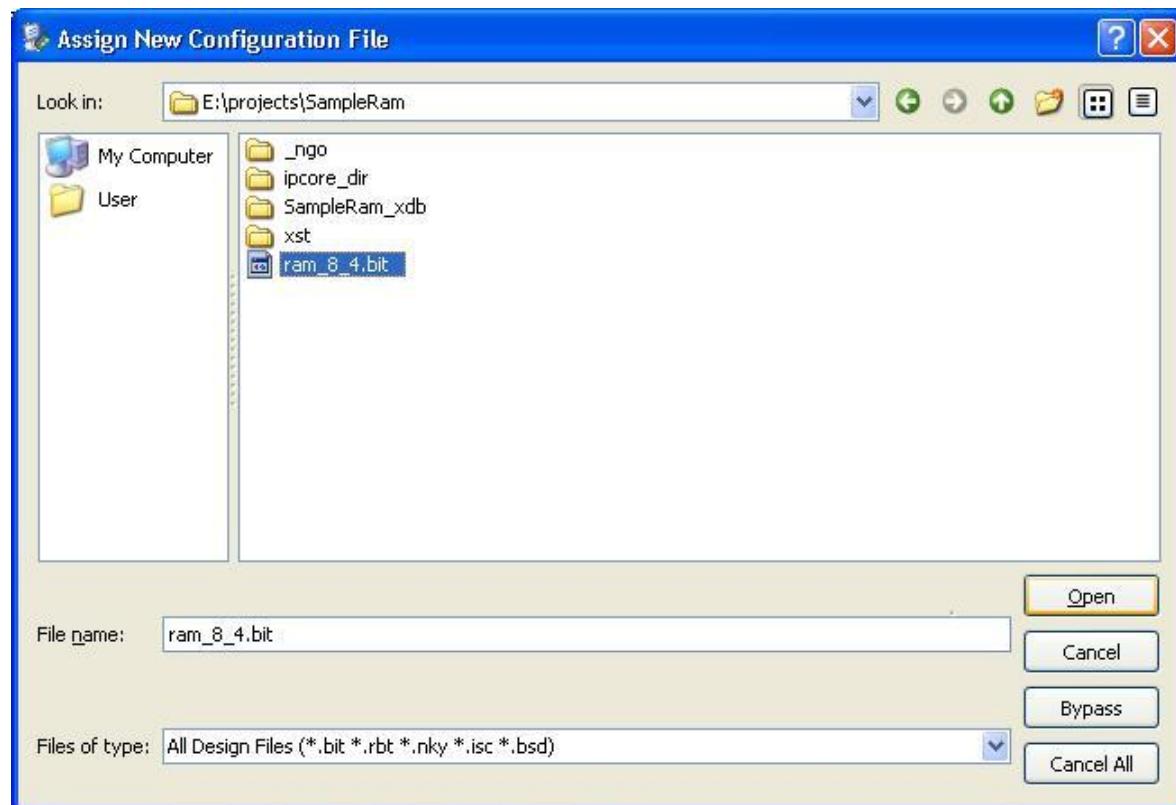
شکل ۲۱ اجرای نرم افزار iMPACT برای برنامه ریزی



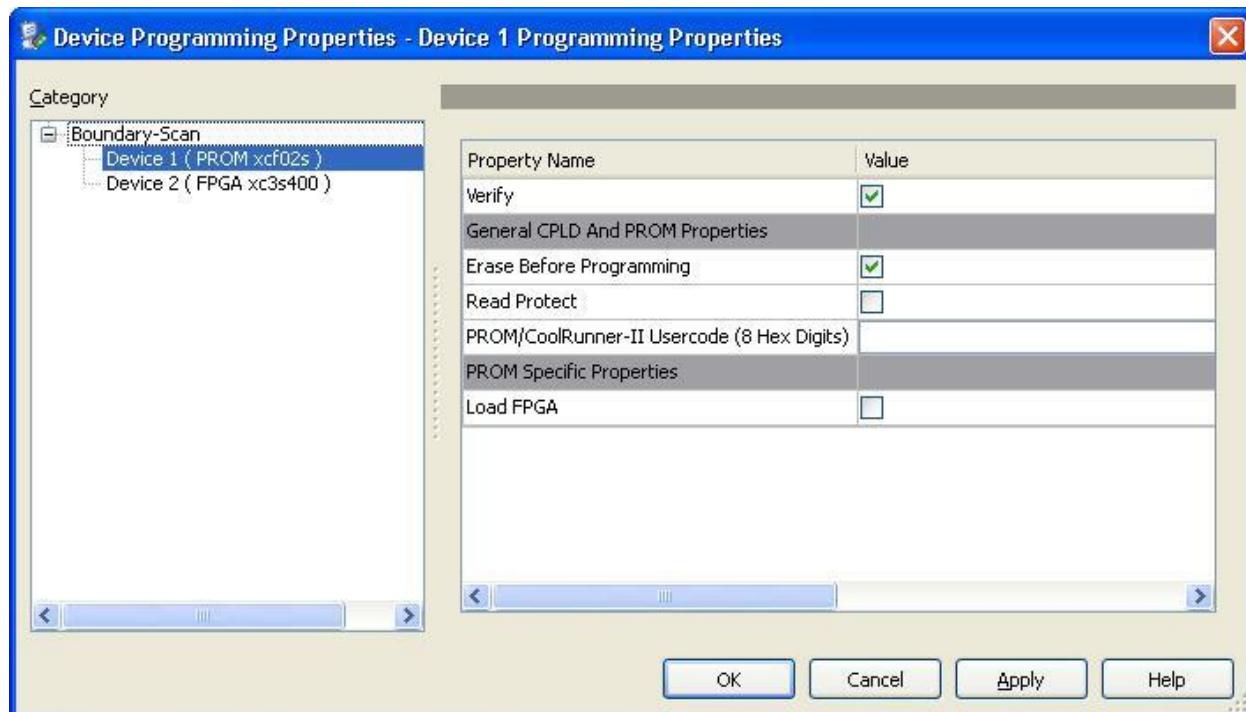
شکل ۲۲ ابزار های شناخته شده



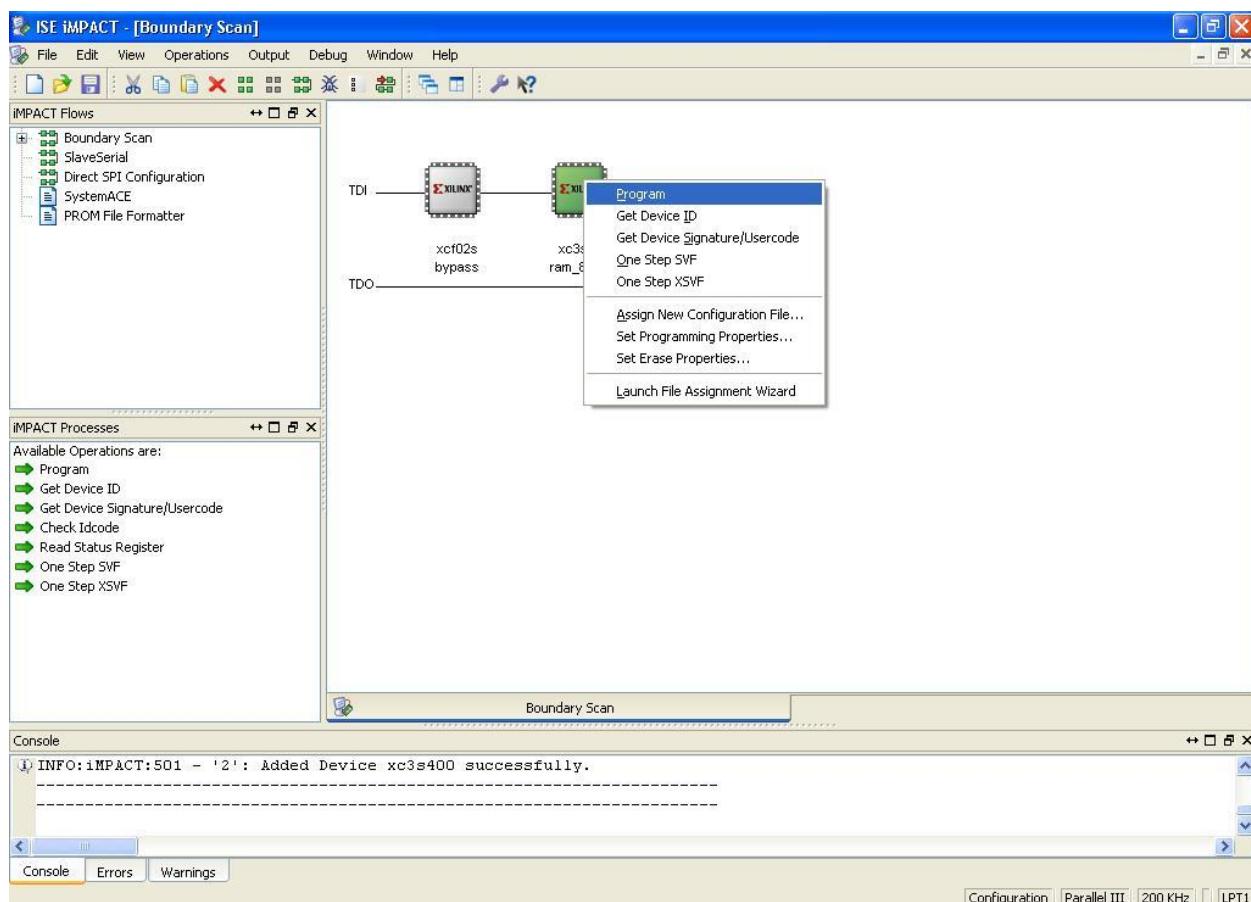
شکل ۲۳ تعیین فایل برنامه ریزی RAM موجود در روی بورد. (فعلاً این مرحله را bypass می‌کنیم).



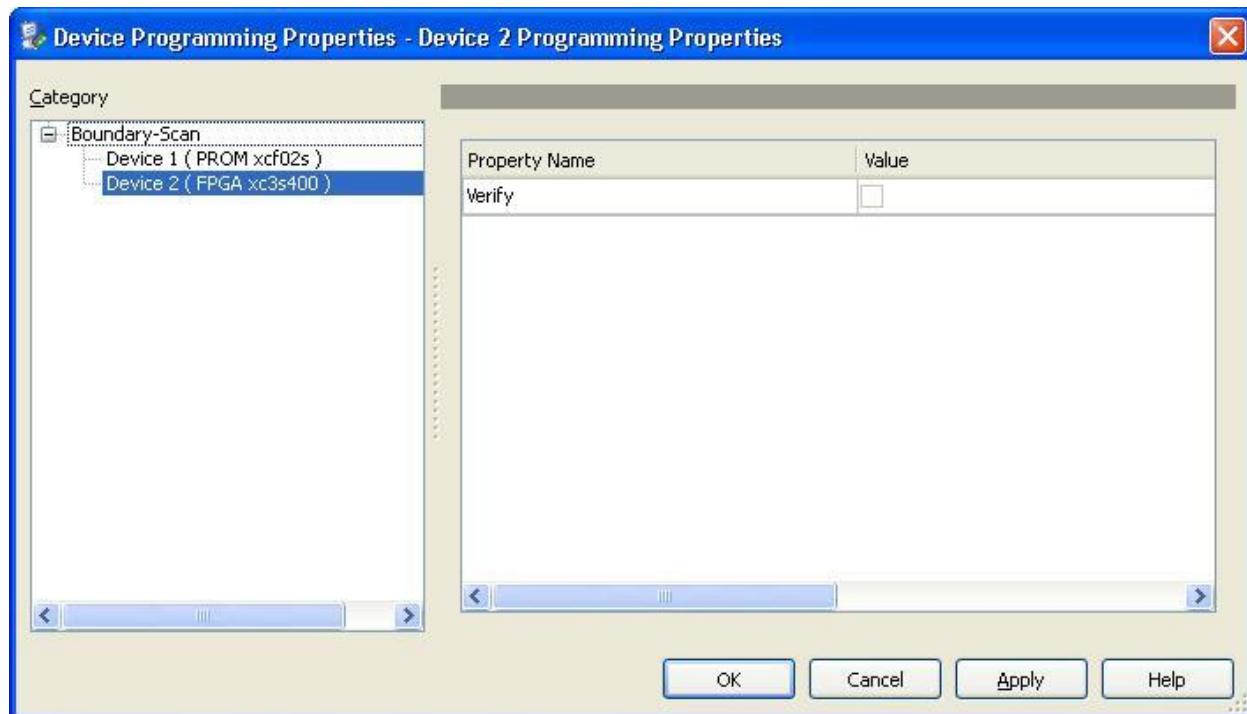
شکل ۲۴ انتخاب فایل برنامه ریزی تولید شده در مرحله قبل



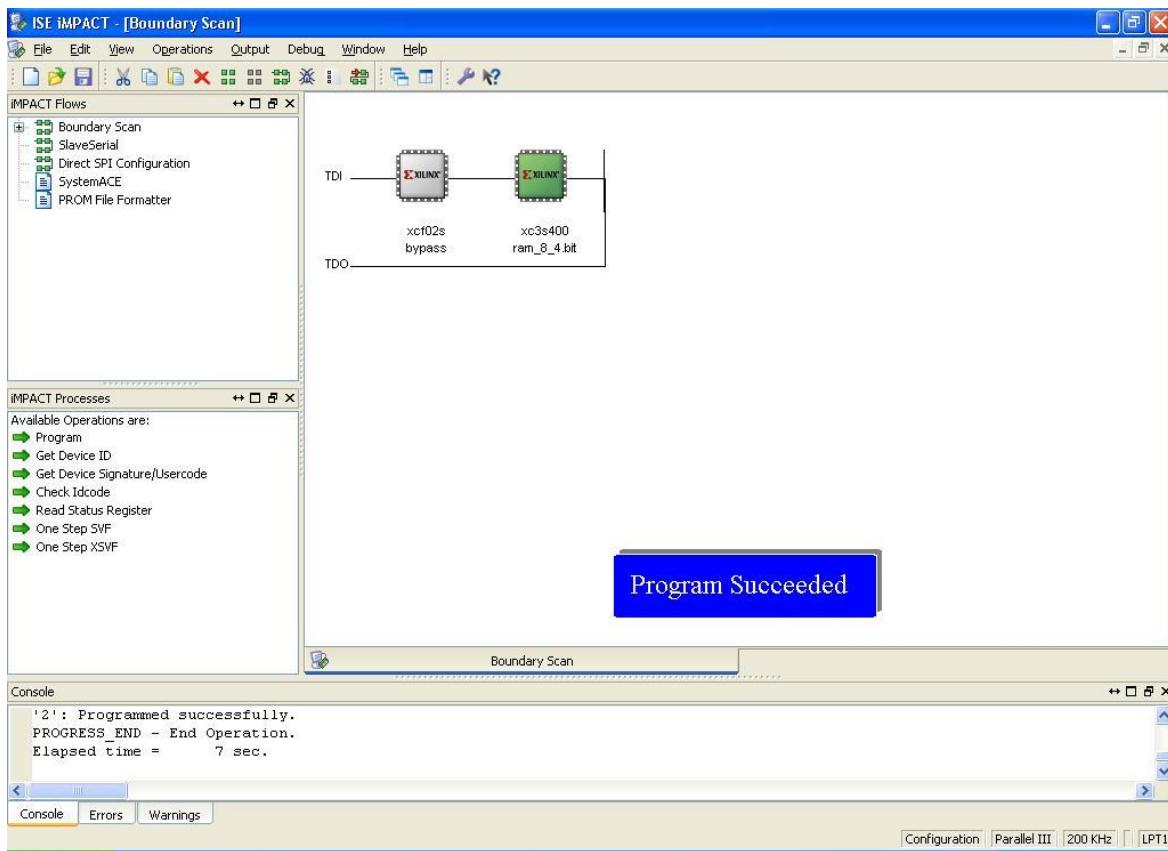
شکل ۲۵ نمایش ویژگی های برنامه ریزی ابزار



شکل ۲۶ برنامه ریزی FPGA

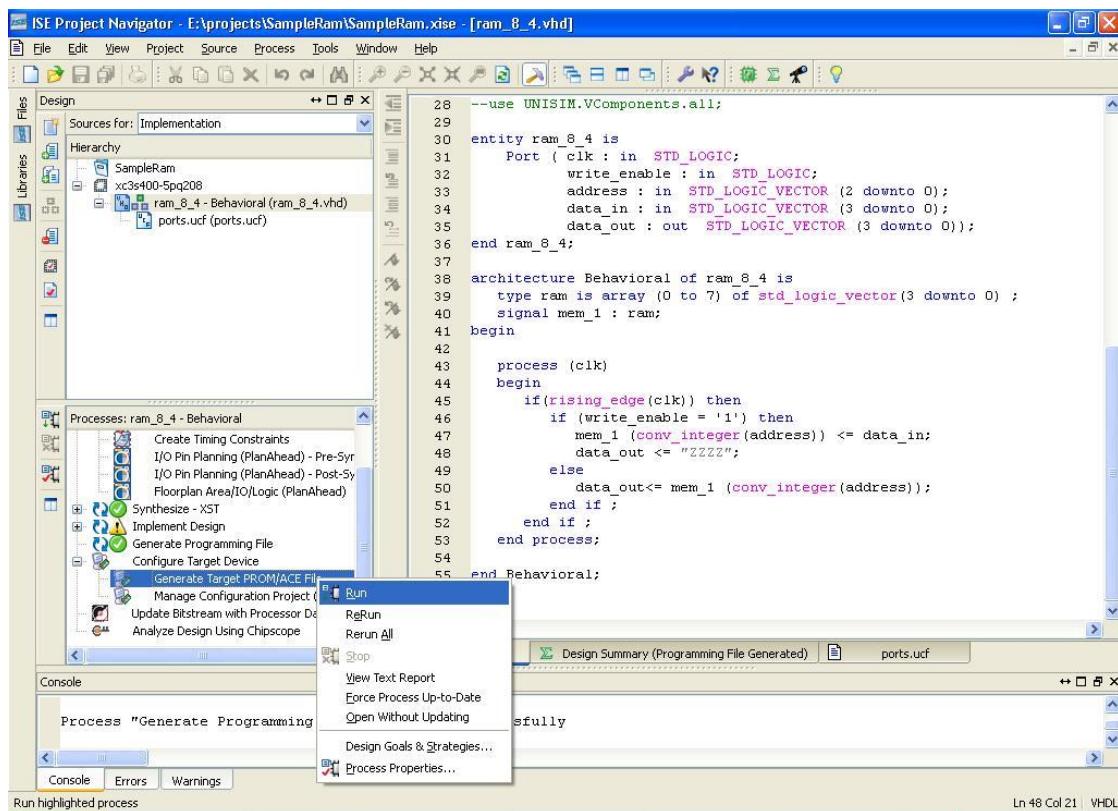


شکل ۲۷ تأیید نهایی

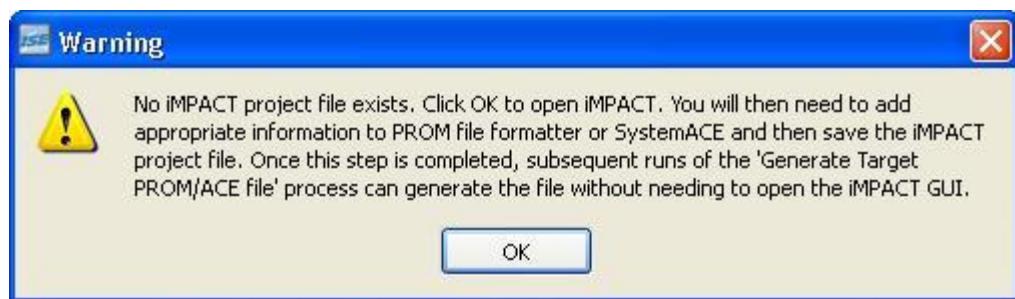


شکل ۲۸ وضعیت برنامه ریزی

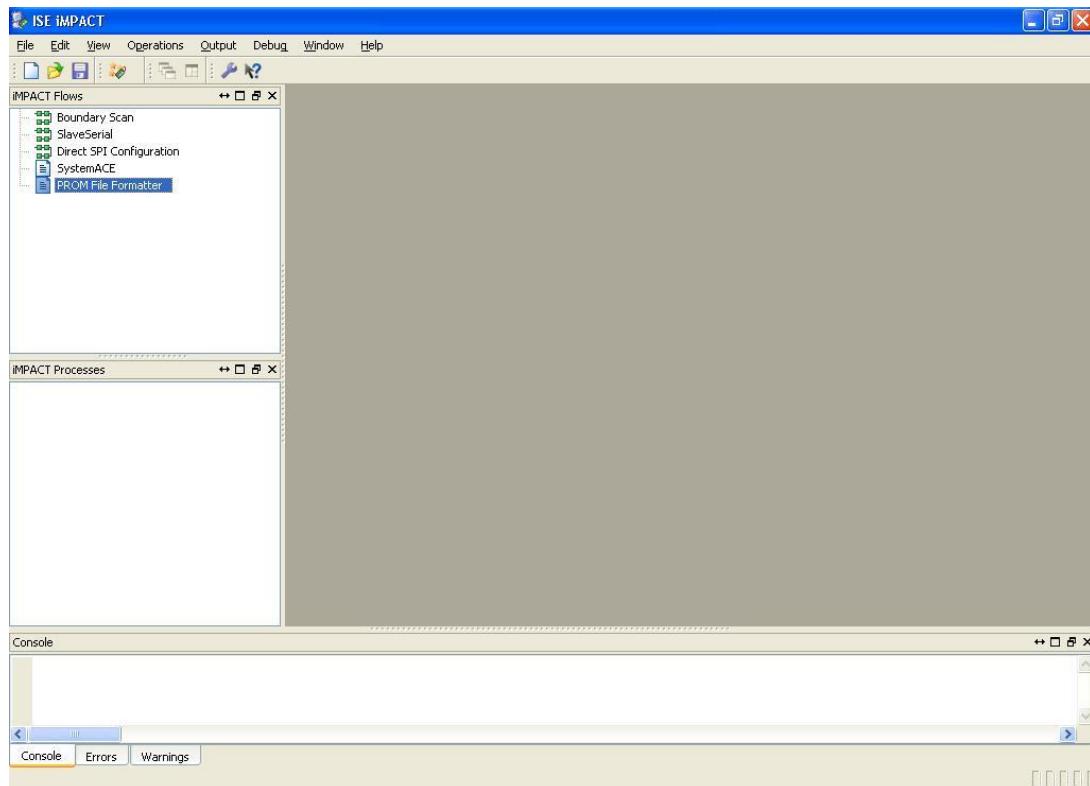
فاز دوم: برنامه ریزی RAM روی بورد



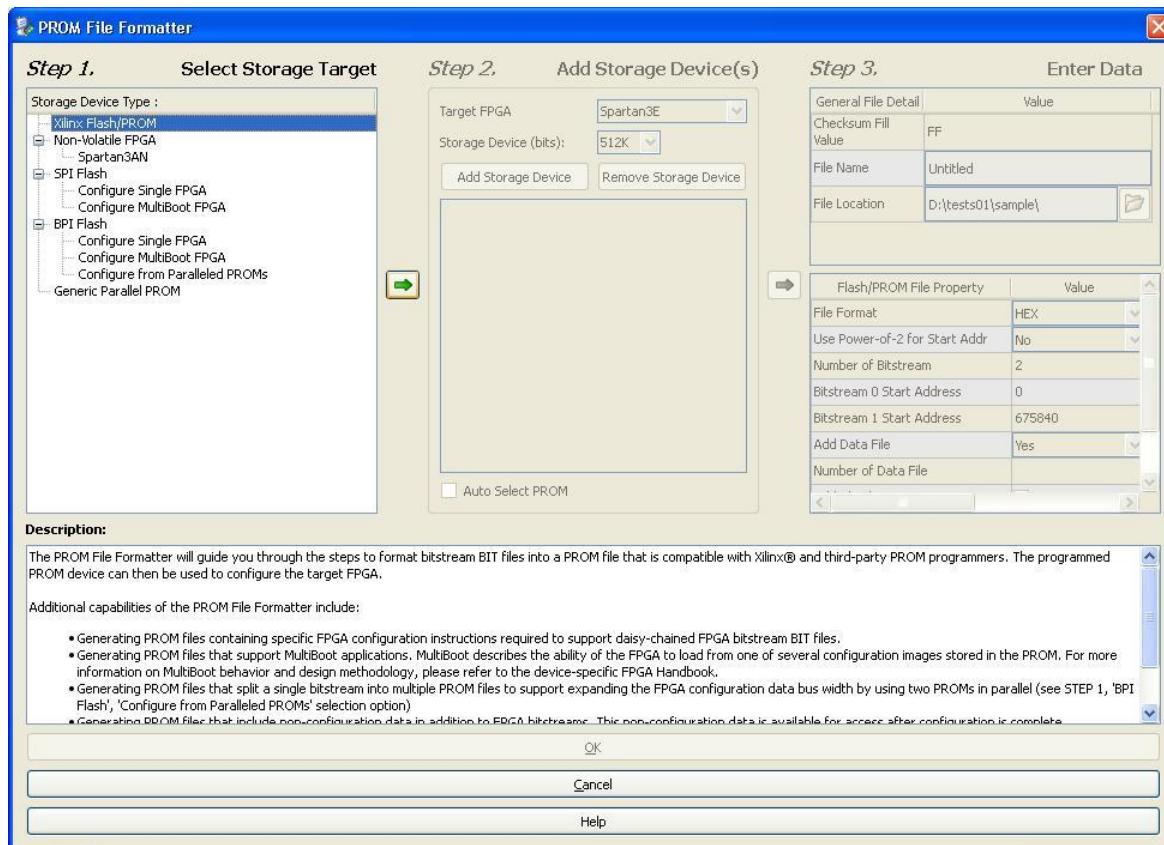
شکل ۲۹ منوی ایجاد فایل برنامه ریزی RAM



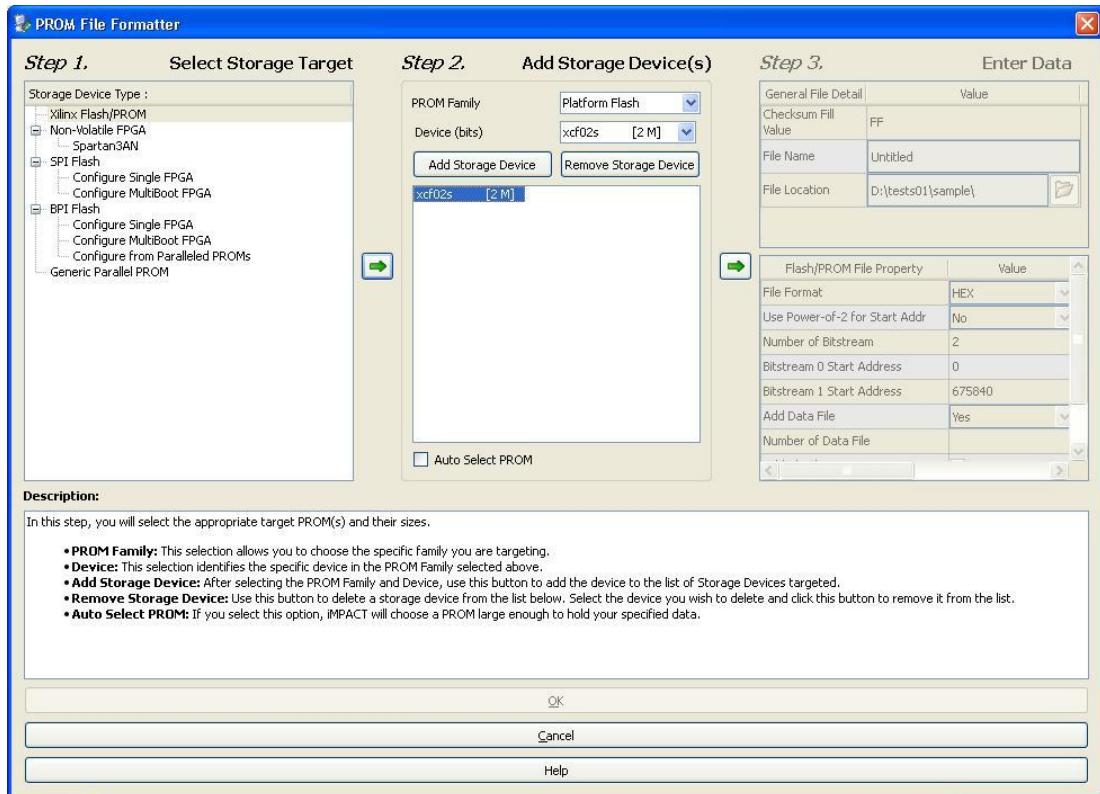
شکل ۳۰ هشدار مبنی بر ایجاد پروژه iMPACT جدید



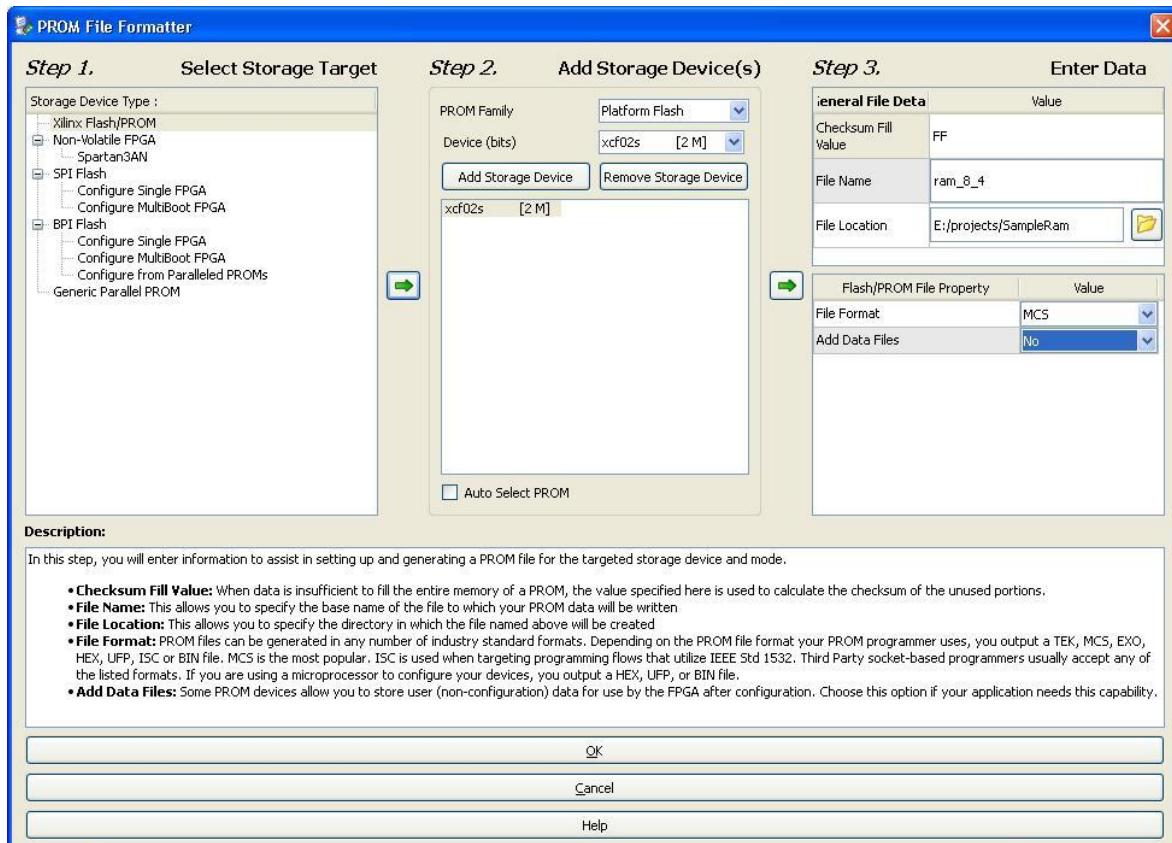
شکل ۳۱ منوی ایجاد فایل برنامه ریزی در پنجره iMPACT



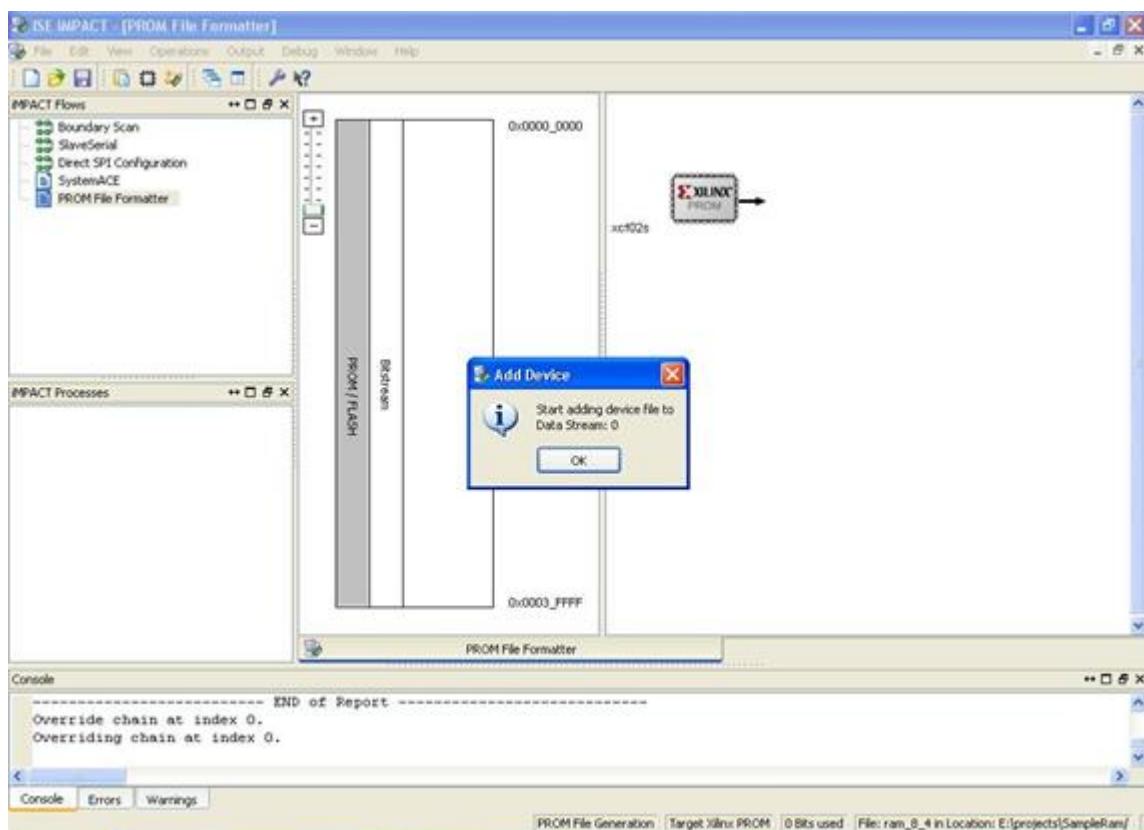
شکل ۳۲ گام اول ایجاد فایل برنامه ریزی



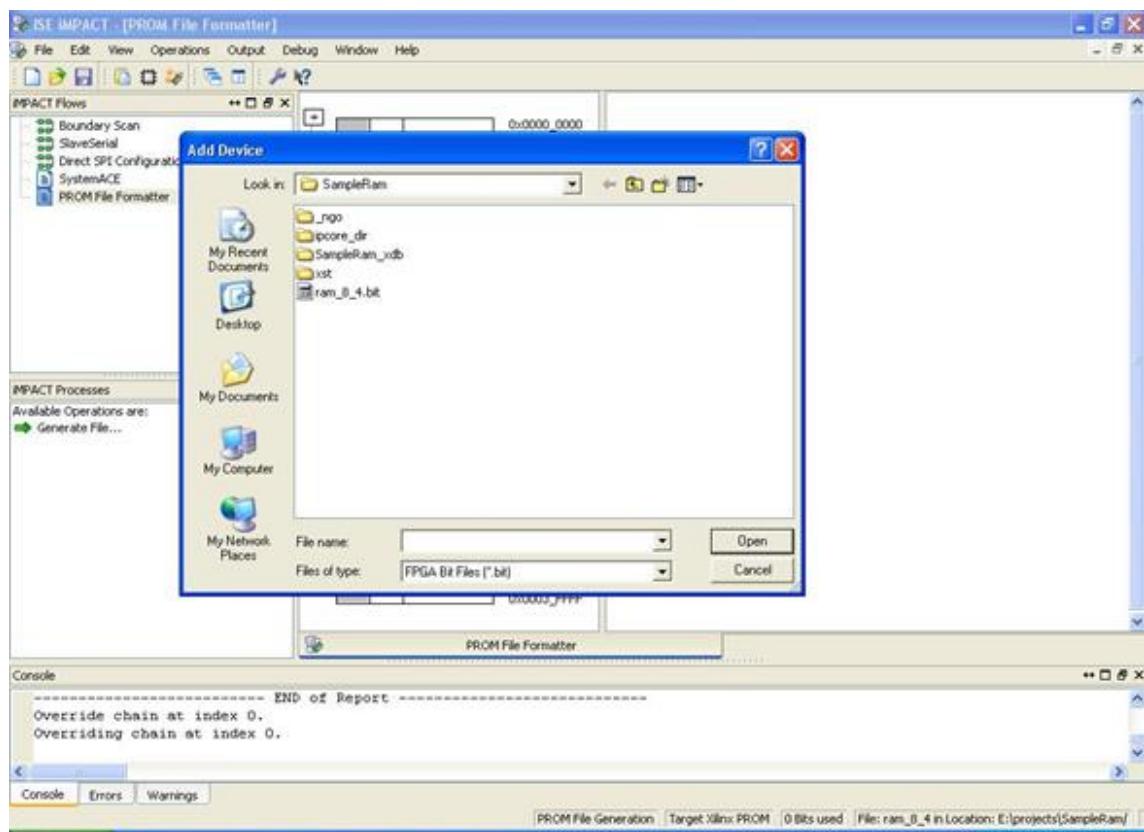
شکل ۳۳ گام دوم ایجاد فایل برنامه‌ریزی



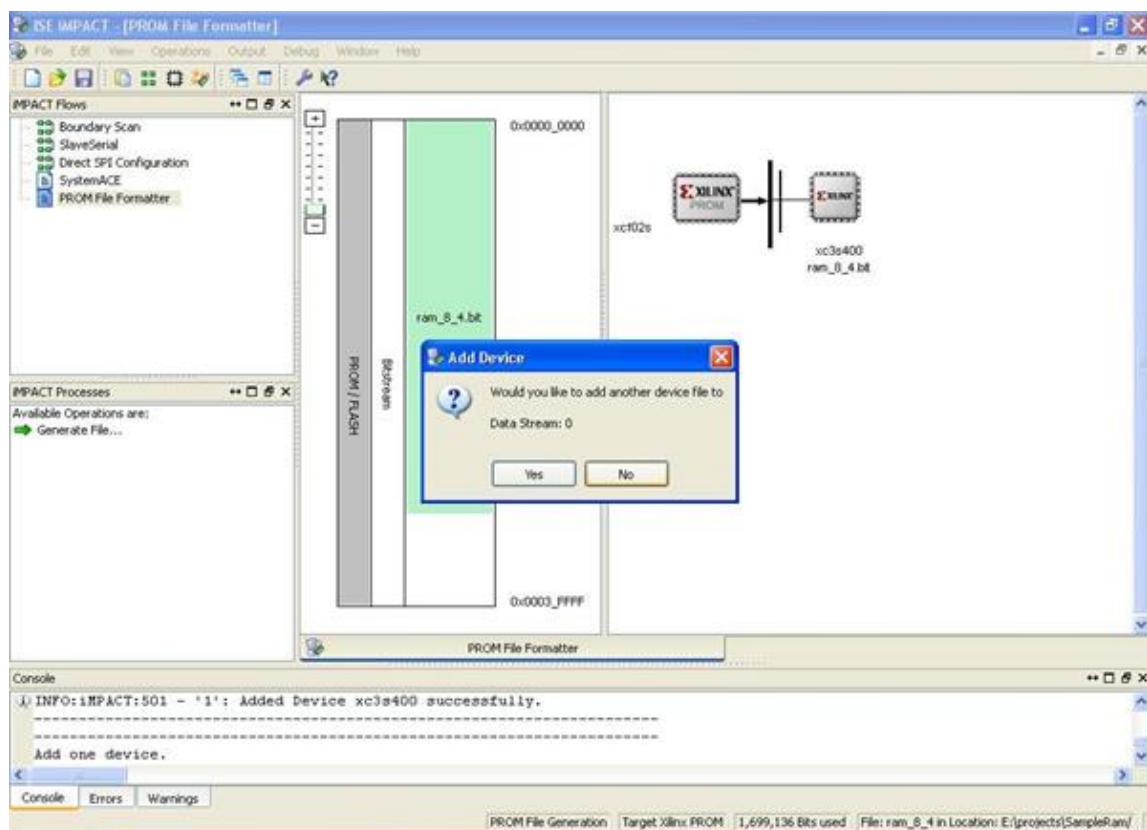
شکل ۳۴ گام سوم ایجاد فایل برنامه‌ریزی



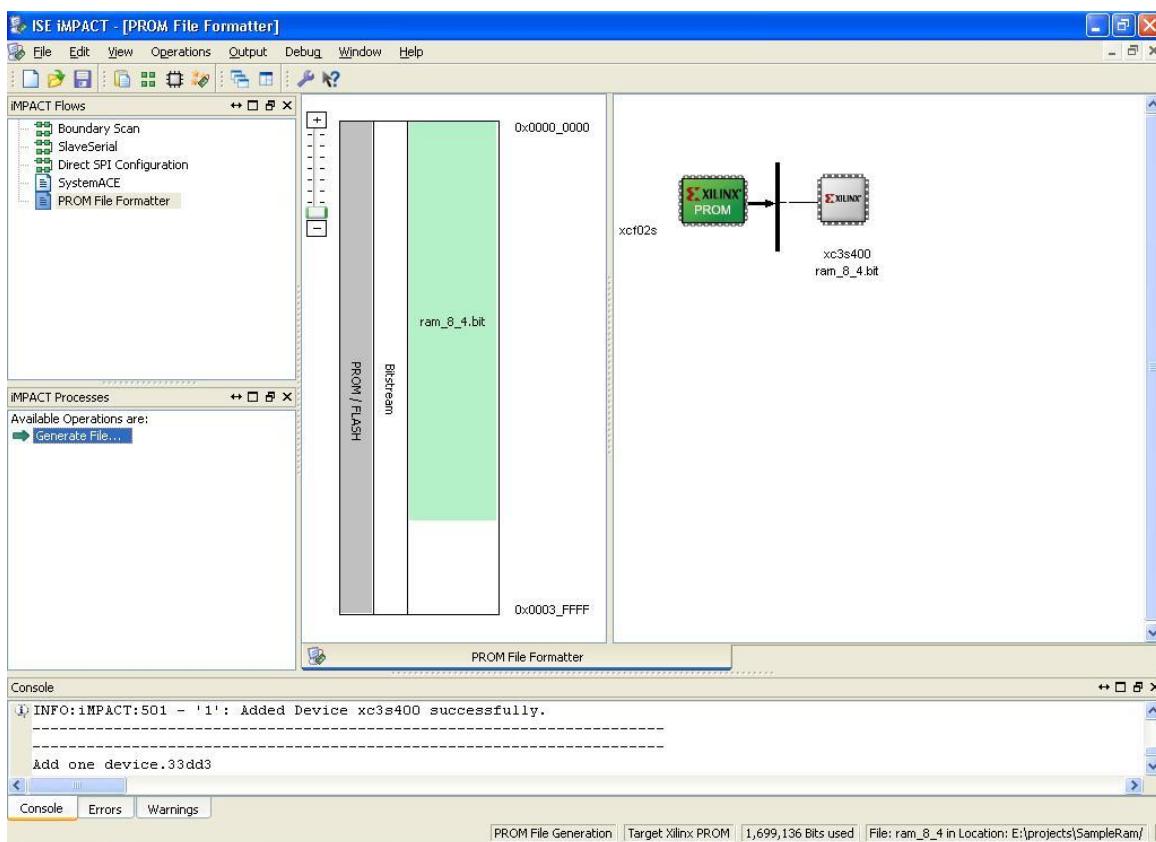
شکل ۳۵ جستجوی RAM های روی بورد توسط iMPACT



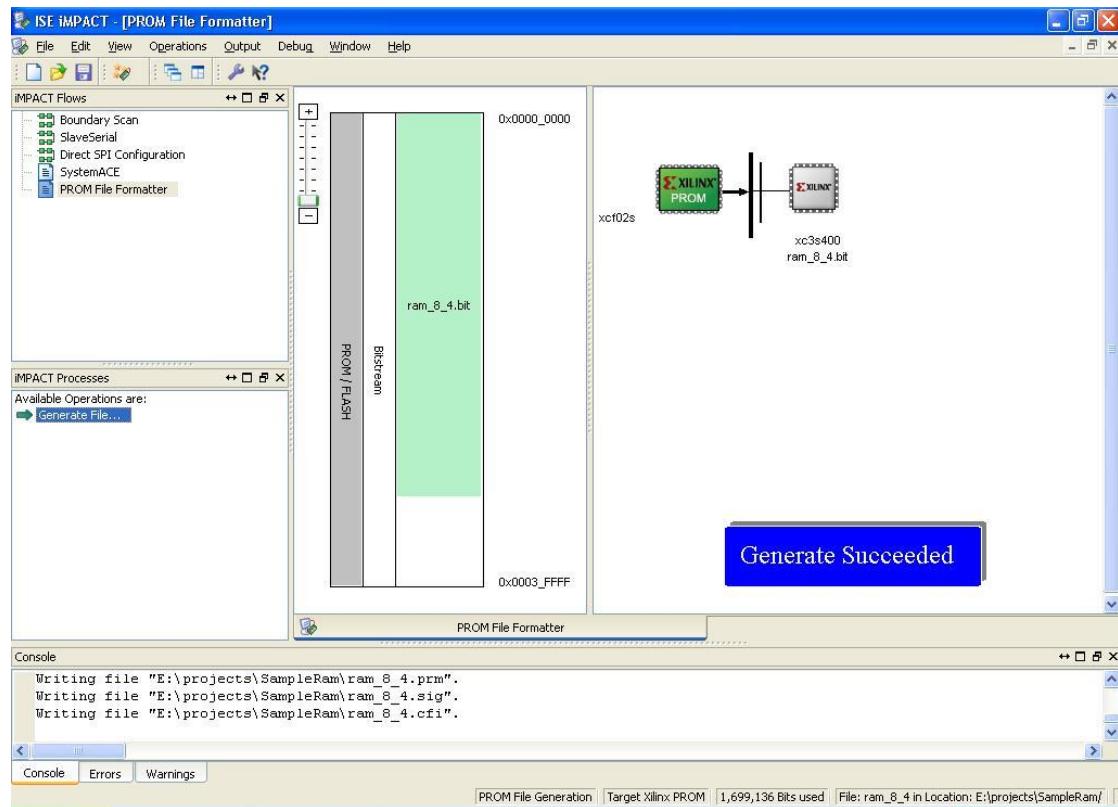
شکل ۳۶ انتخاب فایل برنامه ریزی FPGA برای برنامه ریزی RAM



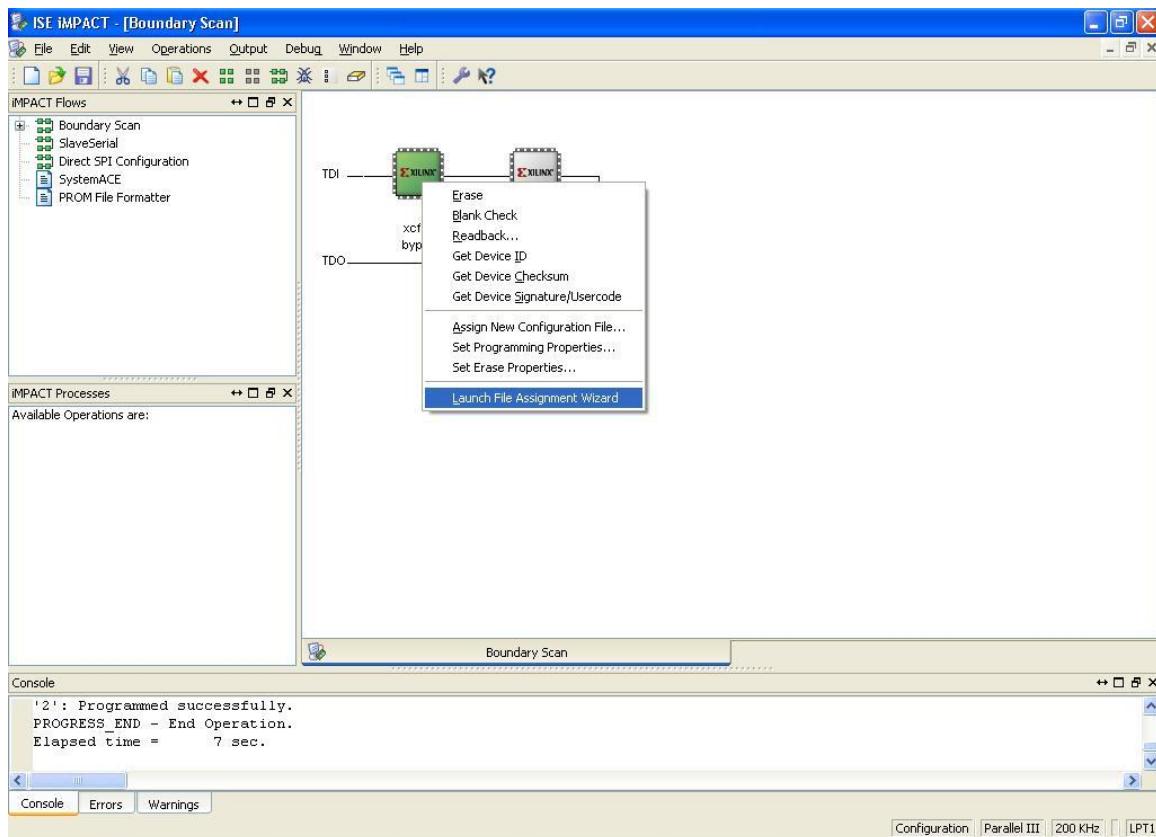
شکل ۳۷ هشدار مبنی بر افزودن فایل جدید



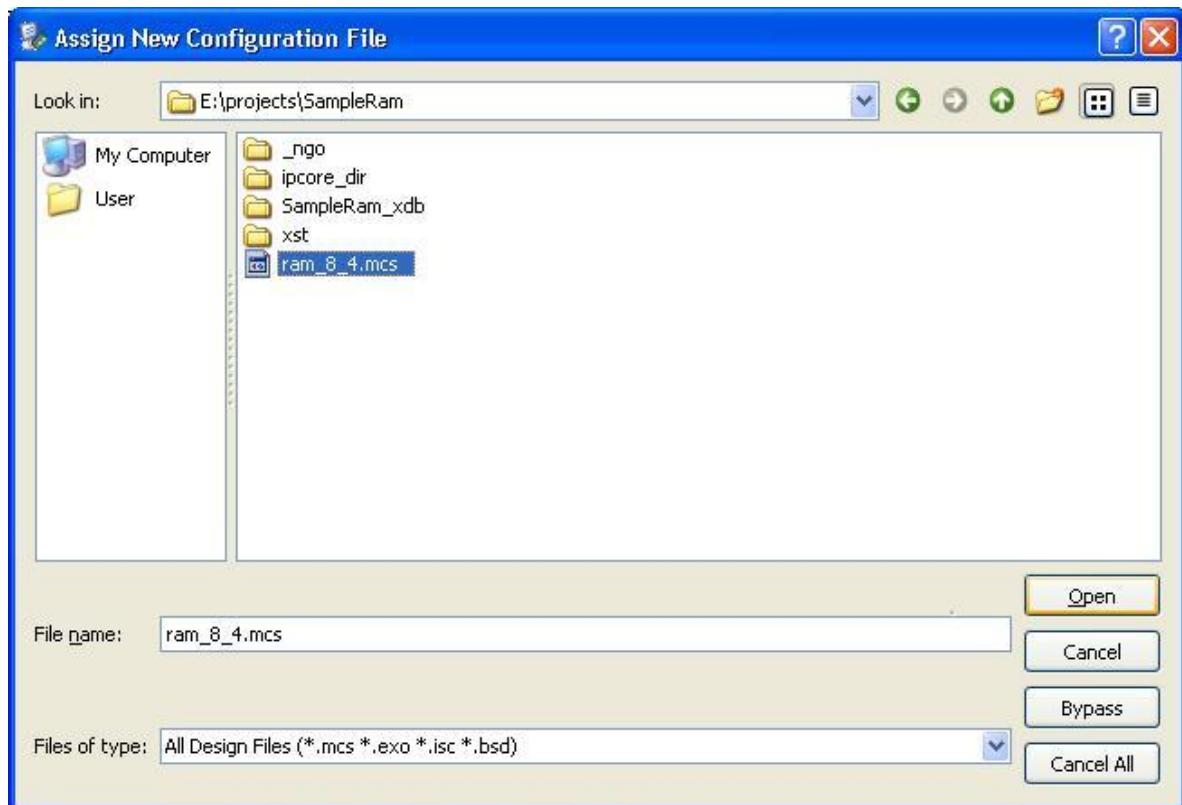
شکل ۳۸ نتیجه انتخاب فایل



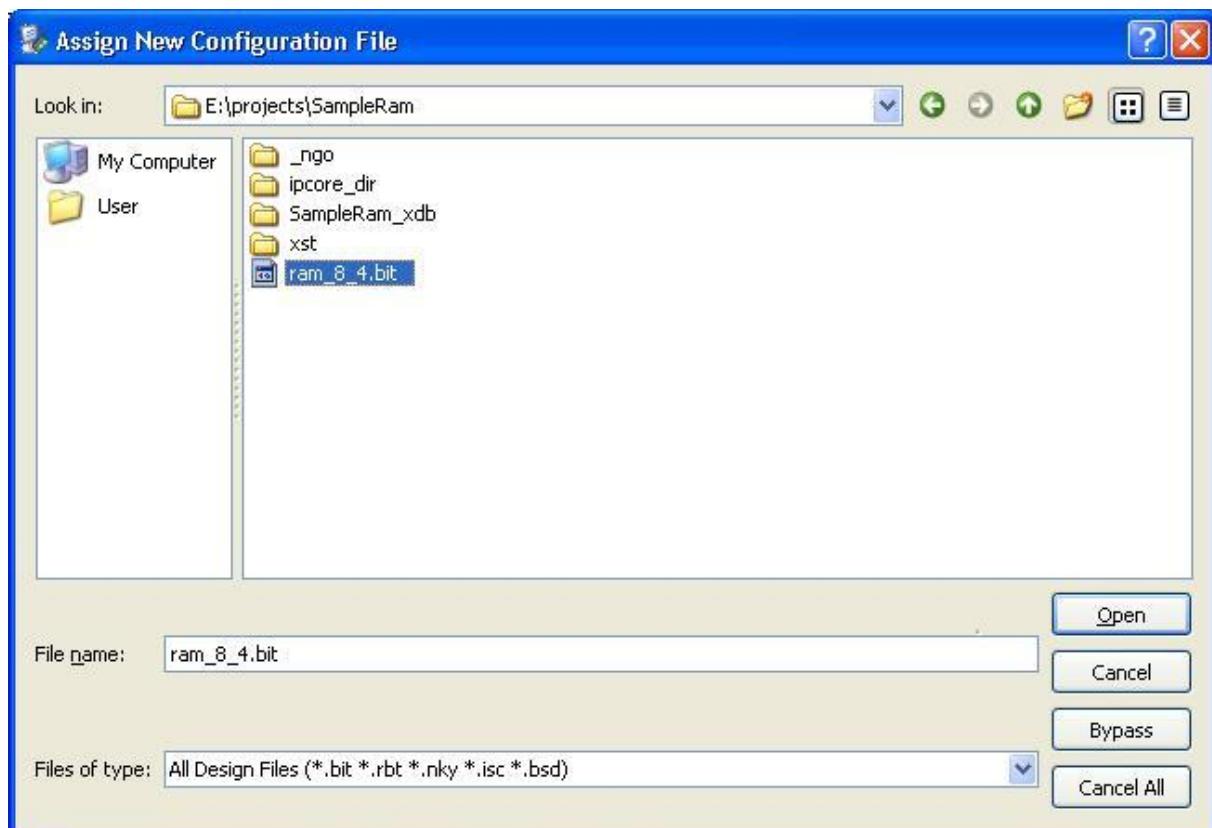
شکل ۳۹ ایجاد موفقیت آمیز فایل برنامه ریزی RAM



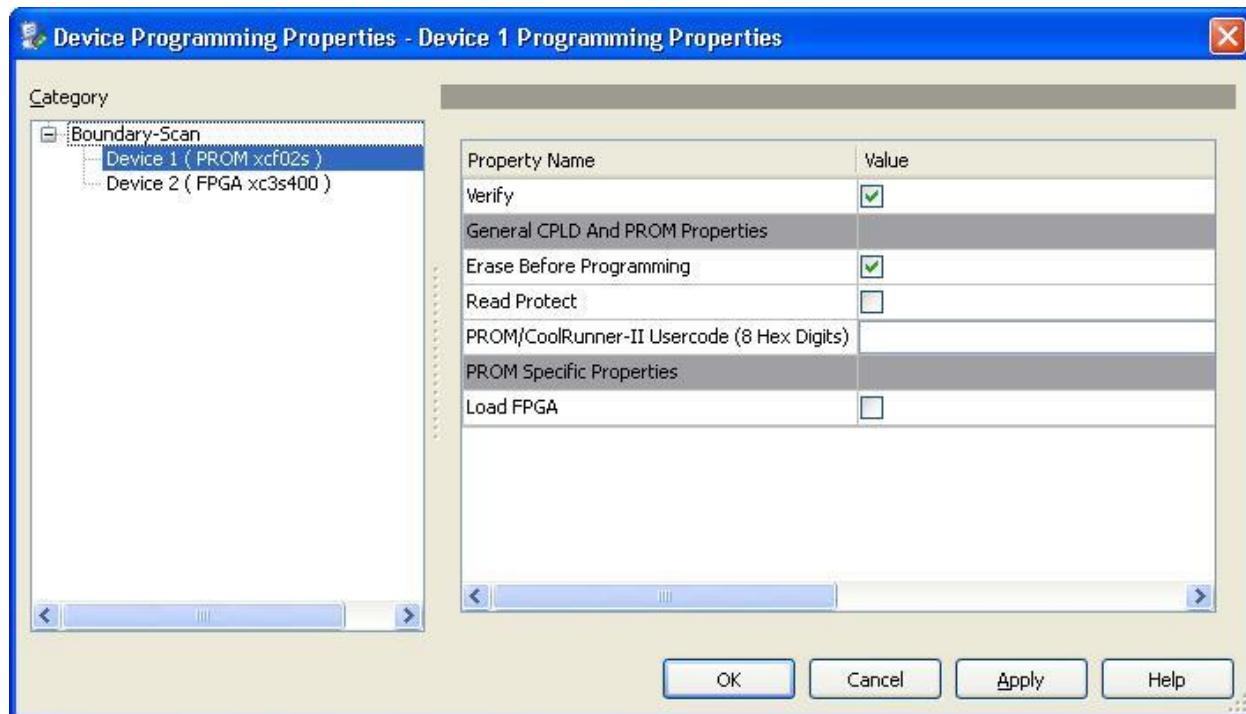
شکل ۴۰ تعیین فایل برنامه ریزی ram



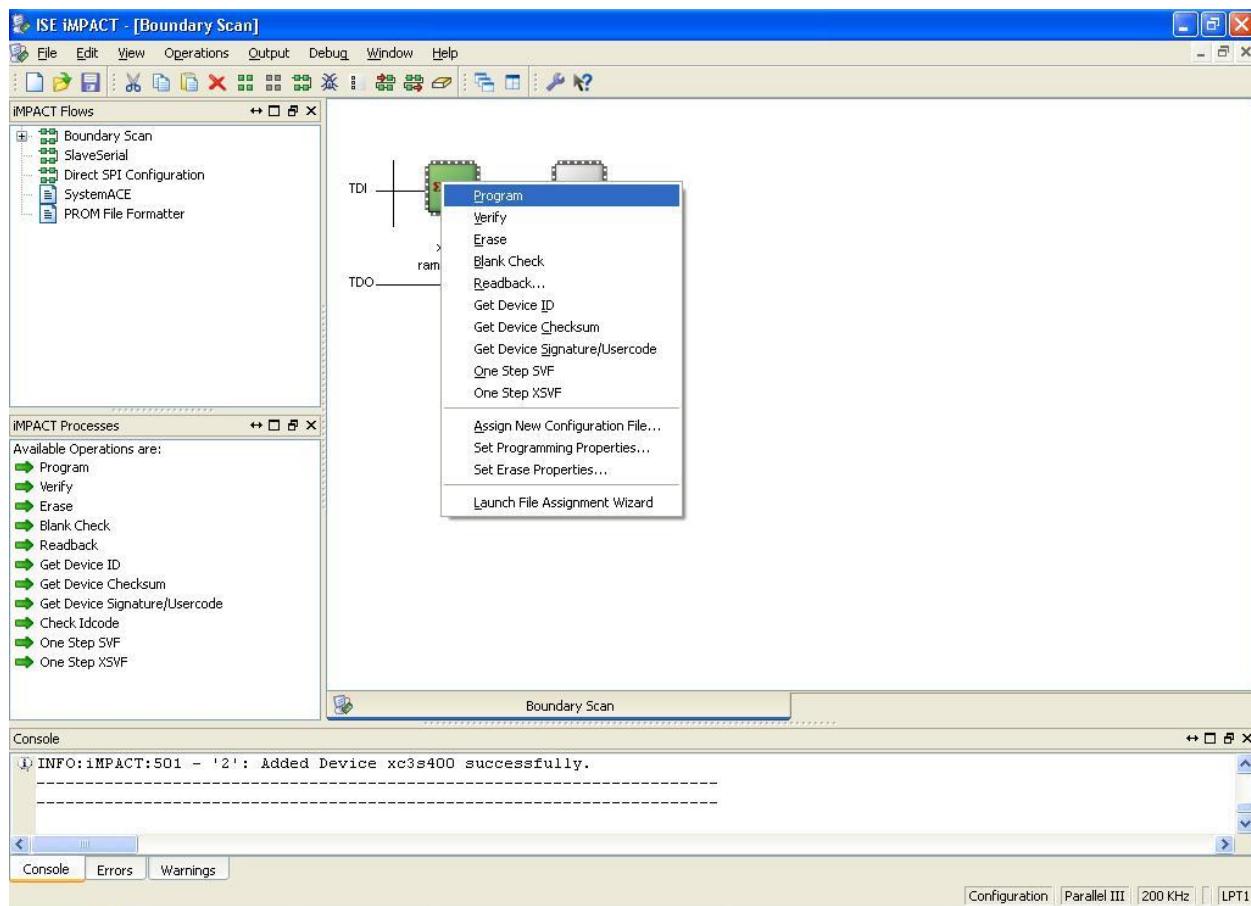
شکل ۴۱ انتخاب فایل برنامه ریزی RAM



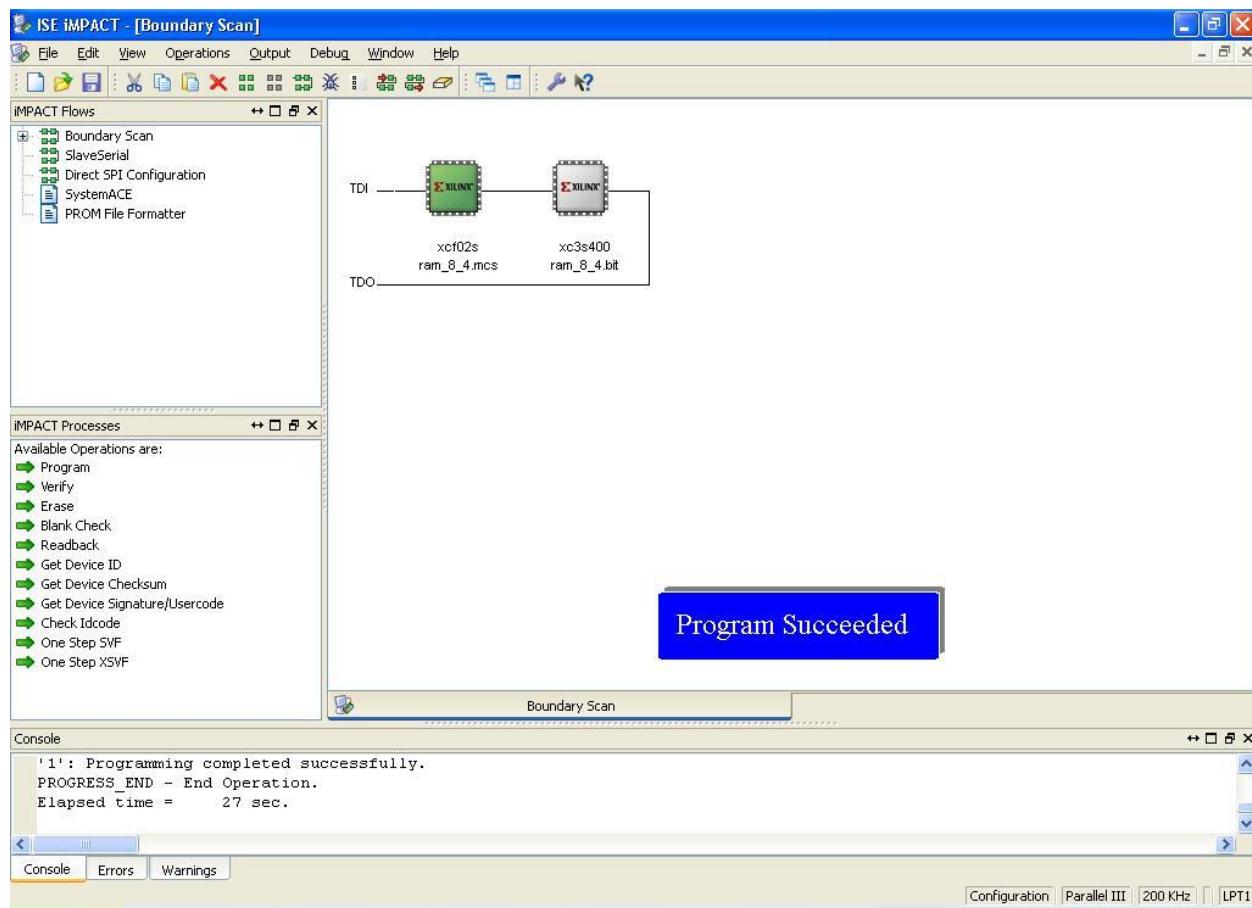
شکل ۴۲ انتخاب فایل برنامه ریزی FPGA



شکل ۴۳ ویژگی‌های برنامه‌ریزی دو ابزار



شکل ۴۴ برنامه‌ریزی RAM



شکل ۴۵ نتیجه موفقیت‌آمیز برنامه‌ریزی RAM