

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 بردّهای 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 تجزیه 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 يك 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 يك 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
۶	توصیف مدارهای پایه آزمایش‌ها و انجام شبیه‌سازی	VHDL مقدماتی	۱
۱۲	طراحی مدارهای اولیه Decoder, Encoder, Multiplexer DeMultiplexer (و مقایسه‌گر) ۴ بیتی	مدارهای پایه	۲
۱۳	Process‌ها و چگونگی پیاده‌سازی شمارندها	پیاده‌سازی مدار ترتیبی	۳
۱۶	پیاده‌سازی انواع جمع کننده‌ها و تحلیل سرعت آنها	جمع کننده‌ها	۴
۱۷	پیاده‌سازی انواع ضرب کننده ۴ بیتی	ضرب کننده‌ها	۵
۲۰	طراحی حافظه RAM, CAM و حافظه‌های با دو درگاه	حافظه‌ها	۶
۲۳	پیاده‌سازی یک کامپیوتر پایه	کامپیوتر پایه	۷
۲۶	پیاده‌سازی ارسال/دریافت ناهمگام در گذرگاه‌های یکطرفه و دوطرفه	ورودی/خروجی	۸
۲۹	پیاده‌سازی مکانیزم‌های داوری گذرگاه	ورودی/خروجی	۹
۳۲	پیاده‌سازی خط لوله و بررسی تسريع آن	خط لوله	۱۰
۳۴	تقسیم‌کننده دودویی در منطق مکمل ۲	تقسیم‌کننده‌ها	۱۱
۳۶	پیاده‌سازی عملیات جمع یا تفریق واحد ممیز شناور	محاسبات ممیز شناور	۱۲
۳۷	اندازه‌گیری پارامترهای IPC, CPI, MIPS و DMIPS یک پردازنده نوعی	ارزیابی کارآیی	۱۳
۳۹	طراحی واحد کنترل سیم بندی شده (Hard wired Control Unit)	طراحی واحد کنترل	۱۴
۴۰	طراحی واحد کنترل ریزبرنامه‌ریزی شده (Microprogrammed Control Unit)	طراحی واحد کنترل	۱۵
۴۱	طراحی سلسله مراتب حافظه	حافظه‌ها	۱۶

## فهرست ضمایم

صفحه	عنوان	شماره ضمیمه
۱۹	توصیف سیستم با VHDL	۱
۲۲	شبیه‌سازی با استفاده از ModelSim	۲
۳۱	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 test bench

```

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;

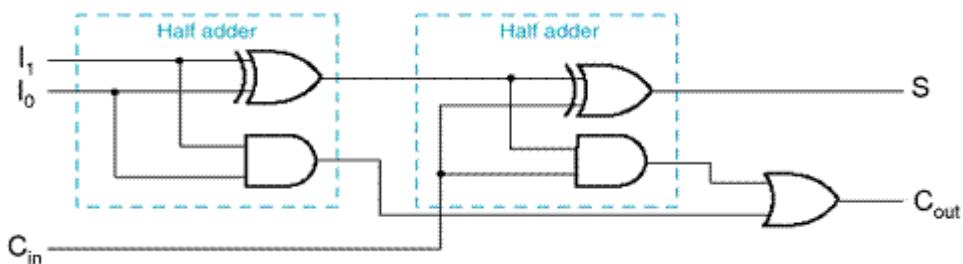
entity decoder_tb is
end entity decoder_tb;

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 (Decoder, Encoder, Multiplexer DeMultiplexer) ۴ بیتی	عنوان:
	هدف:

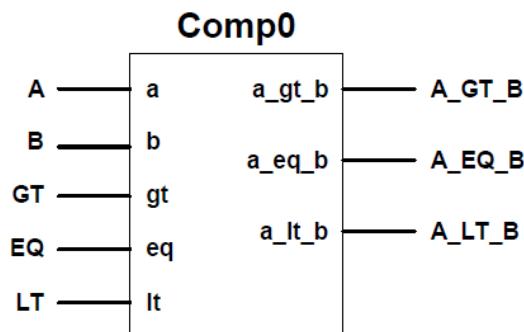
در این آزمایش هدف آشنایی با نحوه عملکرد و پیاده‌سازی هر یک از مدارهای پایه در سطح تجزیه شده است.

#### 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 مازول‌های تکبیتی آن‌ها طراحی و پیاده‌سازی 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 مدار مورد نظر با توجه به ورودی‌های آن با شبیه‌سازی
- پیاده‌سازی بر روی مدار FPGA و نیز اطمینان از درستی عملکرد آن با استفاده از ورودی‌ها و خروجی‌های قابل استفاده روی FPGA برد

#### مراجع مطالعه/ضمایم:

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

آشنایی با پیاده‌سازی مدارهای ترتیبی در VHDL

۳

موضوع:

شماره آزمایش:

عنوان:

هدف:

Process ها و چگونگی پیاده‌سازی شمارندها و ماشین‌های حالت

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

### آمادگی پیش از آزمایش:

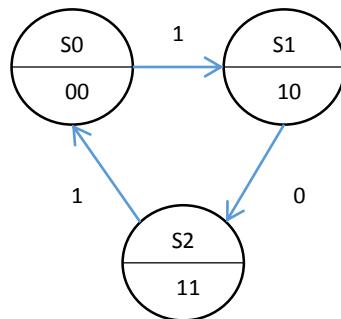
ها ماهیت ترتیبی دارند و باید در بدنه‌ی Architecture نوشته شوند. این ساختارها به صورت ترتیبی اجرا می‌شوند لذا درون آن‌ها می‌توان از دستورالعمل‌های If ... then ... else ... استفاده کرد. هر Process برای اجرا به Sensitivity list نیاز دارد. منظور از Sensitivity list سیگنال‌هایی هستند که وقتی Event روی آن‌ها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

برای طراحی ماشین‌های حالت از Process ها 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 بلوك 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 تعریف شده می‌توان از آن Type یک سیگنال یا متغیر تعریف کرد. سپس از دستورالعمل Case روی سیگنال یا متغیر تعریف شده برای انشعاب به حالت‌های سیستم استفاده می‌شود. واضح است که این دستورالعمل Case بسته به اینکه ماشین حالت به صورت Moore یا Mealy باید طراحی شود، در داخل Process تعریف می‌شود. به عنوان مثال:

```
Signal st : state;
Process (...) is
Case st is
When A => ....
When B => ....
```

برای طراحی مدارات Mealy 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.



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 فلیپ فلاب از نوع D(DFF) با سیگنال Reset ناهمگام (asynchronous) در منطق منفی (active low) طراحی کنید.
- ۲) یک فلیپ فلاب از نوع T(TFF) با سیگنال Reset ناهمگام طراحی کنید.
- ۳) یک Ripple Counter 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 Mealy و سپس Moore طراحی کرده و آن را با استفاده از زبان VHDL طراحی کنید.
- ۵) حال طراحی مدار Mealy را با استفاده از دو Process که قسمت ترکیبی (Combinational) آن از ترتیبی (Sequential) جدا شده است، بنویسید.
- ۶) مداری طراحی کنید که رخداد هر یک از دو رشته "۰۱۱۰" و "۰۱۰۱" 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.

**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. این نوع از جمع کننده‌ها به منظور تسريع عمل جمع بکار می‌رود. در این آزمایش هدف آشنایی با نحوه عملکرد و پیاده‌سازی سخت‌افزاری آن‌ها در سطح تجزید گیت است.

#### 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 تجزید گیت انجام شود. در این پیاده‌سازی باید 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] [http://en.wikipedia.org/wiki/Carry-save\\_adder](http://en.wikipedia.org/wiki/Carry-save_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 ساخت‌افزاری تمامی واحدها در سطح تجزیه و گزینه صورت گیرد.

#### آمادگی پیش از آزمایش:

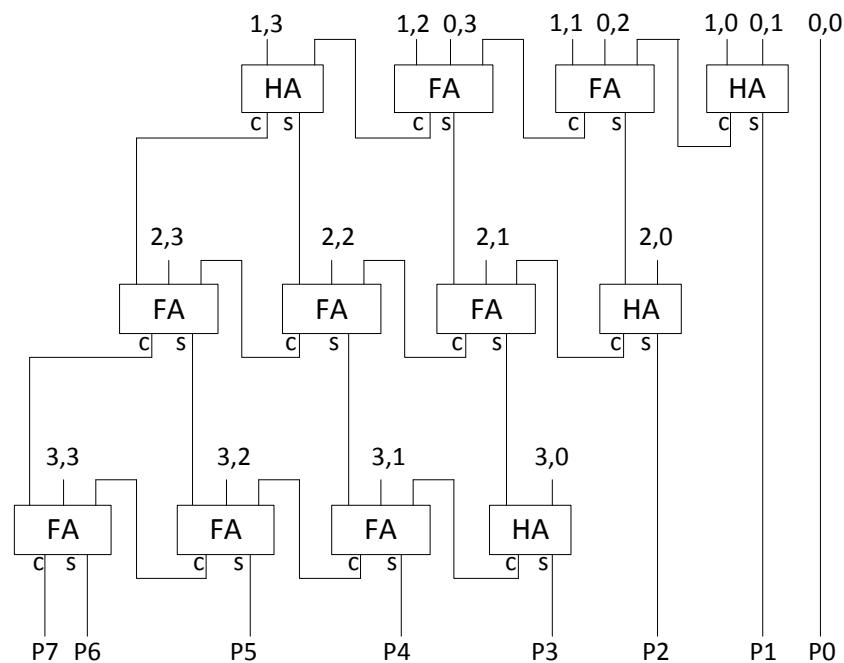
مطالعه مواردی که در بخش مراجع آزمایش به آن اشاره شده است و انجام برخی شبیه‌سازی‌های مورد نظر.

#### 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:

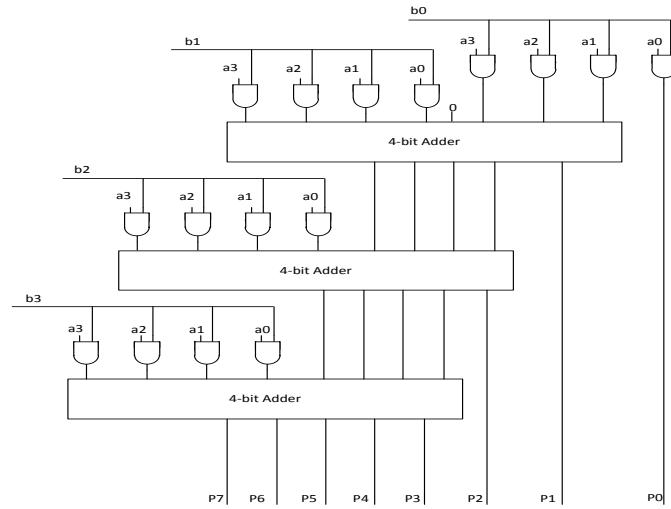
در این ضرب کننده مثل ضرب دهدۀی (decimal)، در هر مرحله یک بیت از مضروب (multiplier) در تمام بیت‌های مضروب فیه (multiplicand) ضرب شده و با یک بیت شیفت با خروجی مرحله قبل جمع می‌شود. نمودار این ضرب کننده در شکل ۱ نشان داده شده است.



شکل ۱- نمودار بلوکی ضرب کننده معمولی

**ب) ضرب کننده آرایه‌ای:**

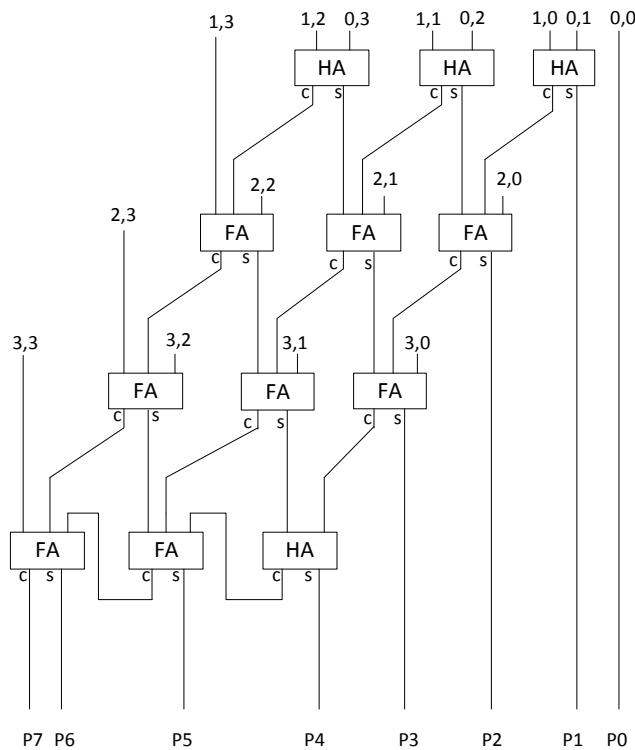
در این ضرب کننده از واحدهای جمع کننده ۴ بیتی استفاده شده است. نمودار این ضرب کننده در شکل ۲ نشان داده شده است.



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.

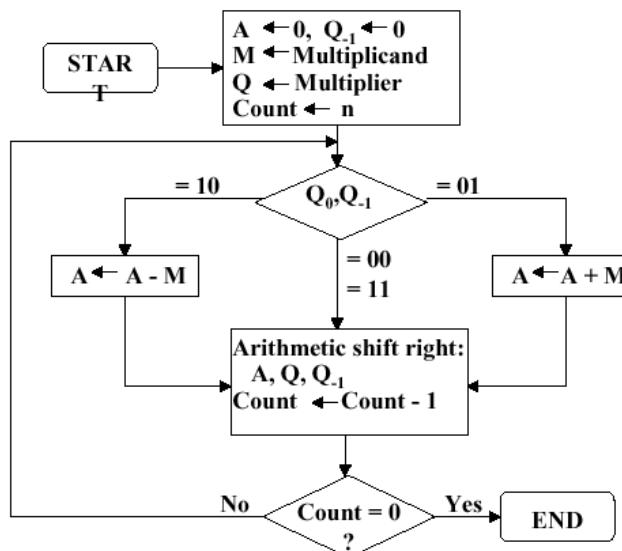


شکل ۳- نمودار بلوکی ضرب کننده carry save adder

## (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.

## 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، کامپیوتر، 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.
<=	عملکر مقداردهی است.

## معرفی کتابخانه IEEE

توصیف یک سیستم با VHDL : بدن 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

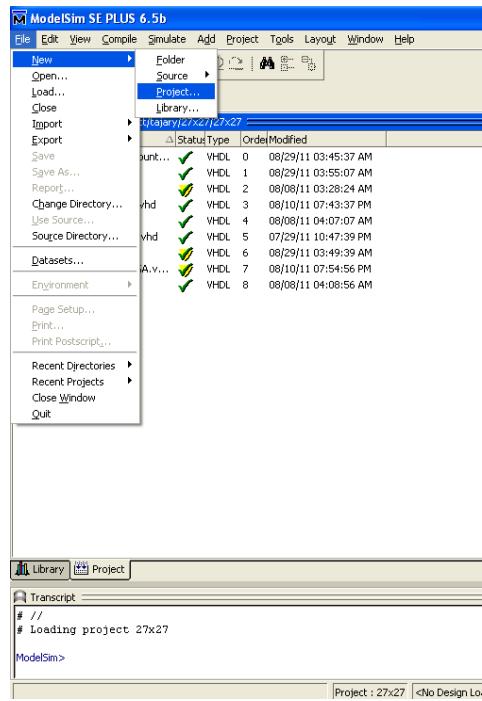
```
Port(
  A: in std_logic_vector(9 downto 0);
  C: out std_logic_vector (9 doento 0)
);
Signal b: std_logic_vector(9 downto 0);
```

## مقداردهی به std\_logic\_vector

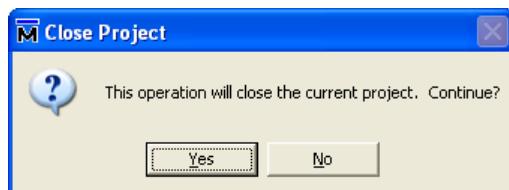
```
B(9 downto 1) <= B(8 downto 0); -- shift
B(0) <= '1';
B(2 downto 0) <= "110";
B <= A;
B(9 downto 8) <= A(2 downto 1);
B(4) <= B(5);
B <= (others => '0');
B(5 downto 0) <= A(9 downto 7) & "011";
```

## ضمیمه ۲

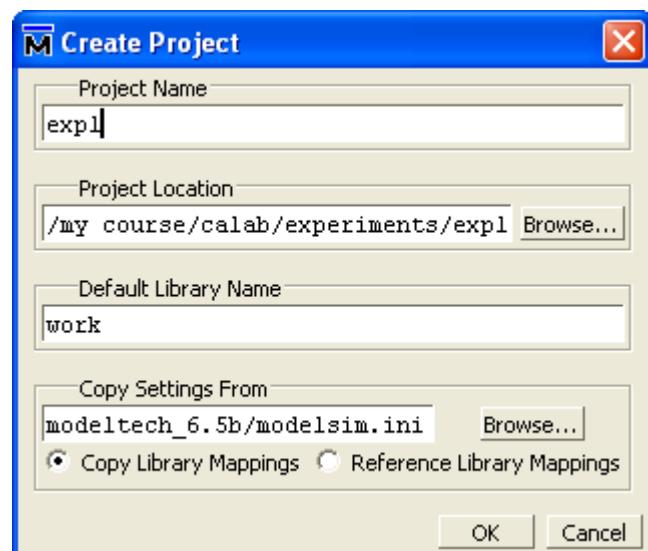
شبیه‌سازی با استفاده از  
**ModelSim**



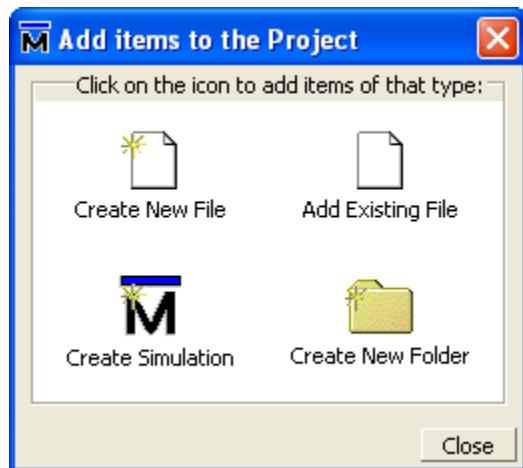
شکل ۱ ایجاد پروژه جدید



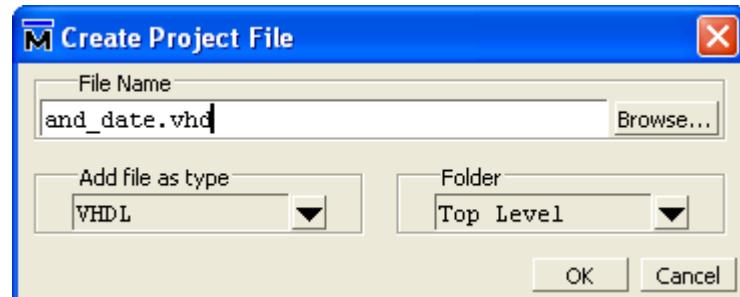
شکل ۲ پیغام هشدار بستن پروژه قبلی



شکل ۳ تعیین نام و پوشش پروژه



شکل ۴ افزودن فایل به پروژه



شکل ۵ تعیین نوع و نام فایل جدید

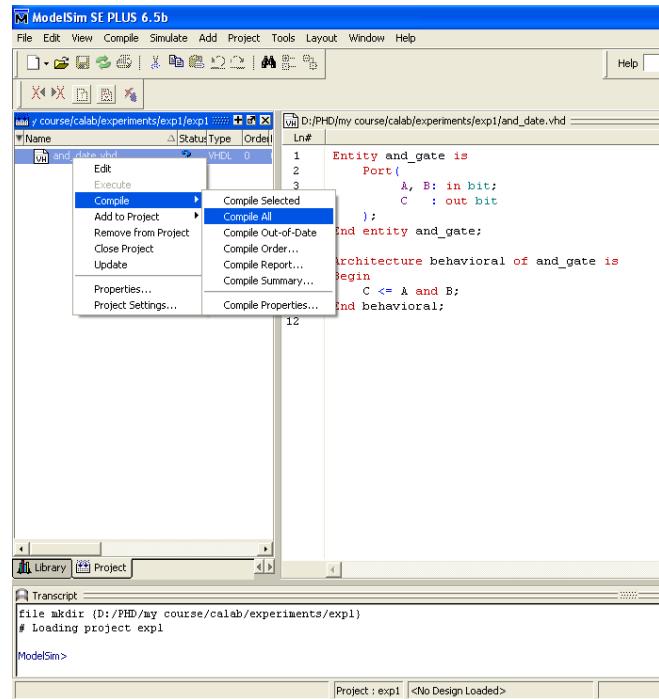
```

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

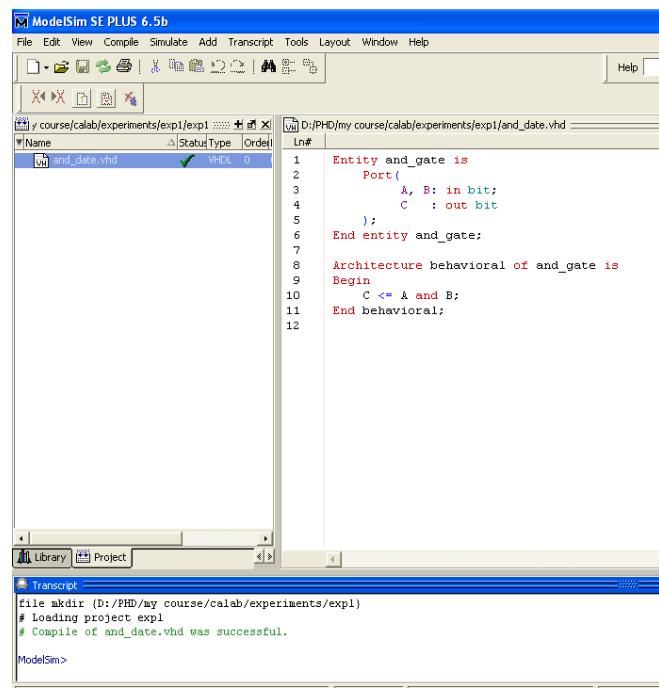
```

The screenshot shows the ModelSim SE PLUS 6.5b interface. The main window displays a VHDL code for an AND gate. The code defines an entity named 'and\_gate' with two input ports (A, B) and one output port (C). The architecture 'behavioral' describes the logic as C = A and B. The interface includes a menu bar, toolbars, a project browser, and a transcript window at the bottom.

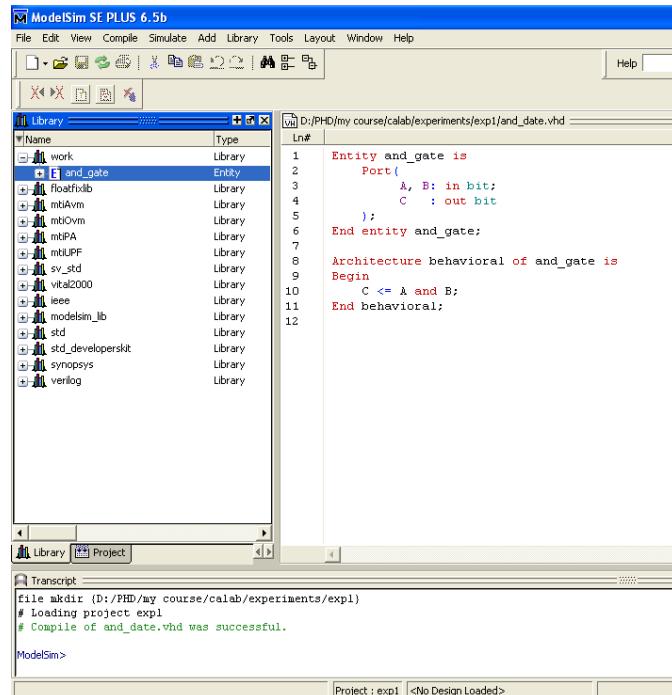
شکل ۶ نوشتن کد در ویرایشگر



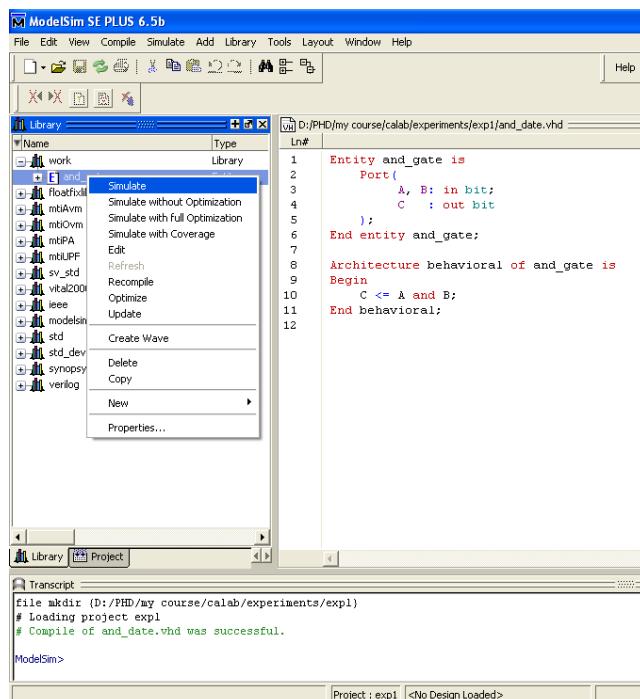
شکل ۷ منو کامپایل



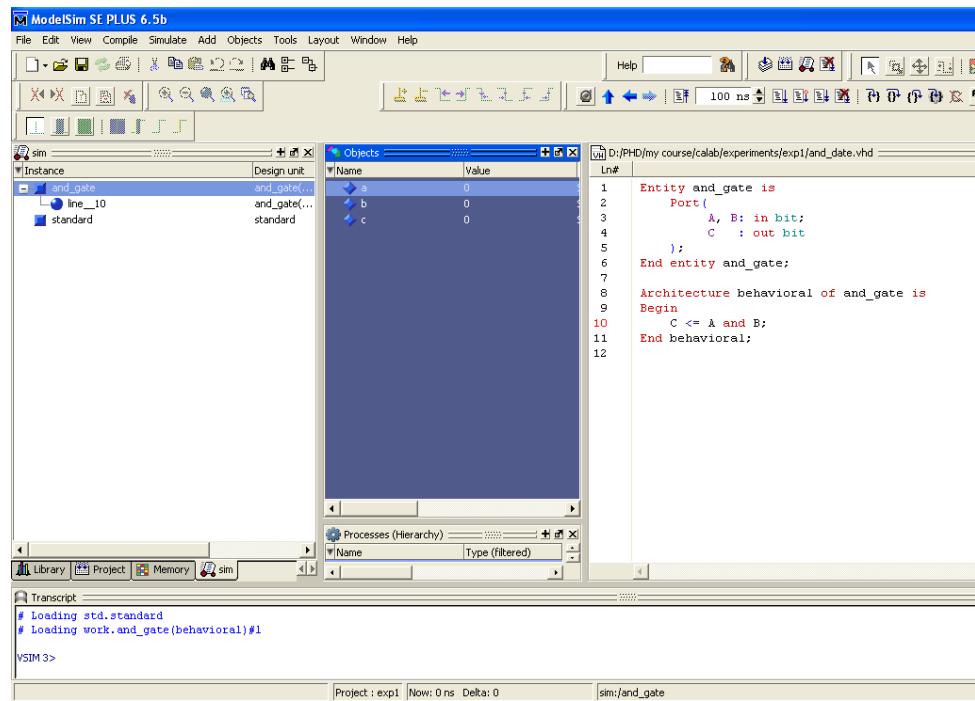
شکل ۸ نتیجه کامپایل



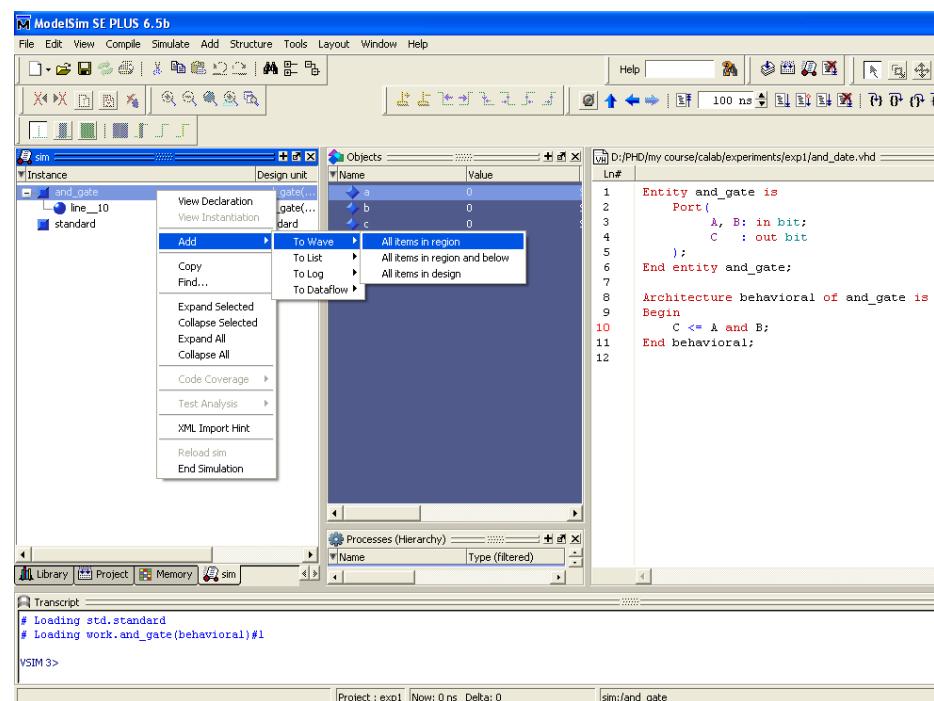
شکل ۹ بخش library برای شبیه‌سازی



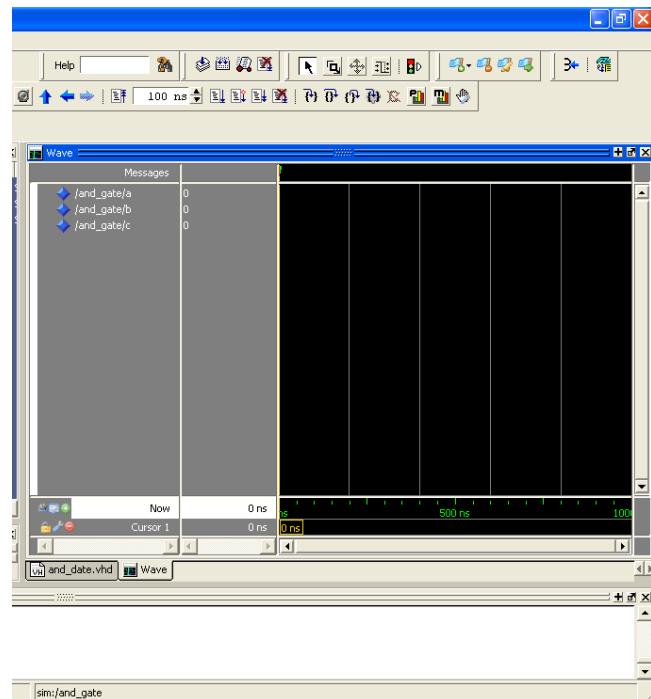
شکل ۱۰ منو شبیه‌سازی



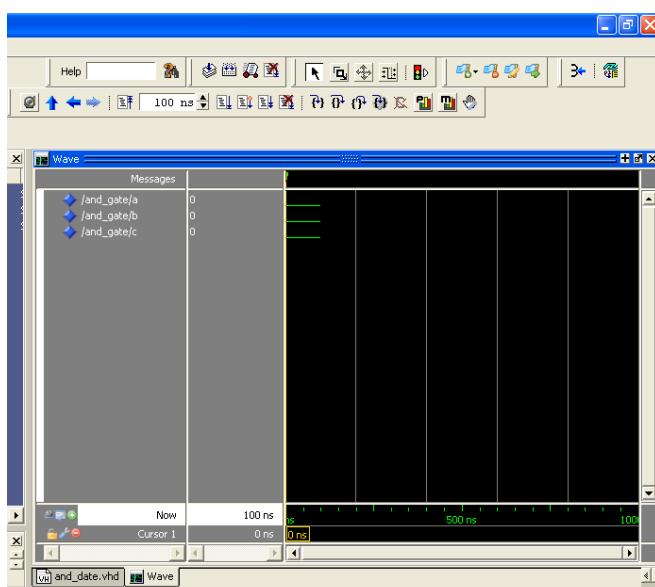
شکل ۱۱ نمای شروع شبیه‌سازی



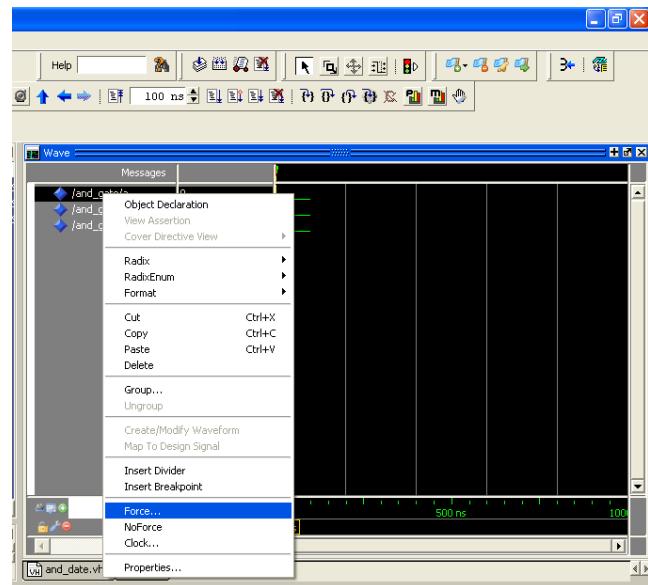
شکل ۱۲ افزودن سیگنال به پنجره wave



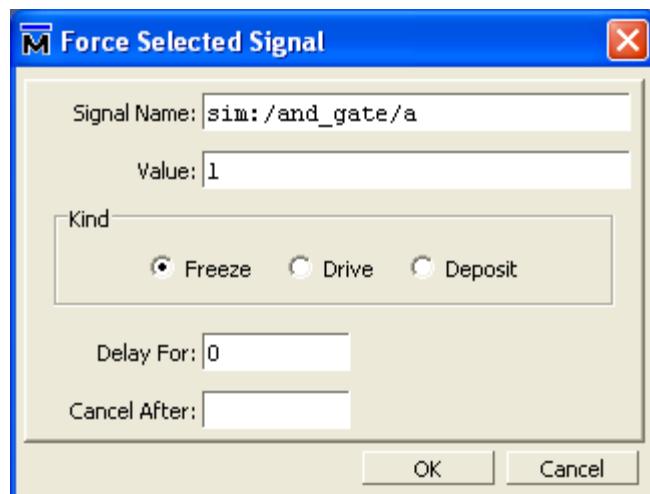
شکل ۱۳ سیگنال‌ها در پنجره wave



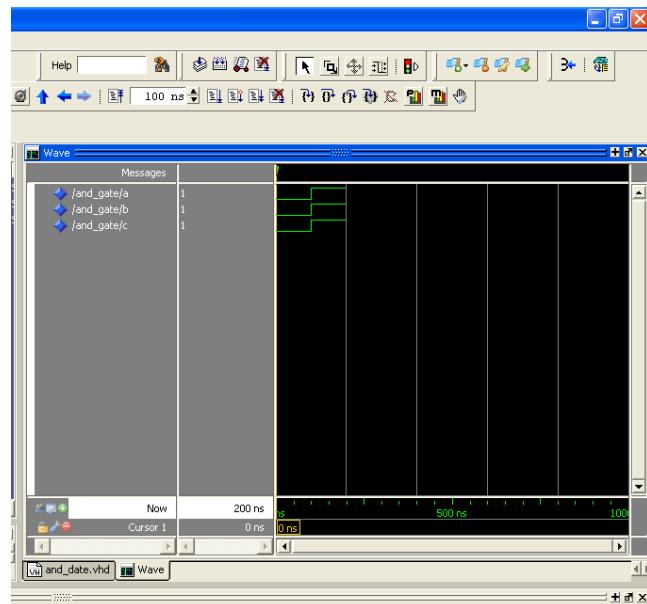
شکل ۱۴ اجرای شبیه‌سازی



شكل ۱۵ منوی force کردن سیگنال



شكل ۱۶ تعیین مقدار سیگنال در پنجره force

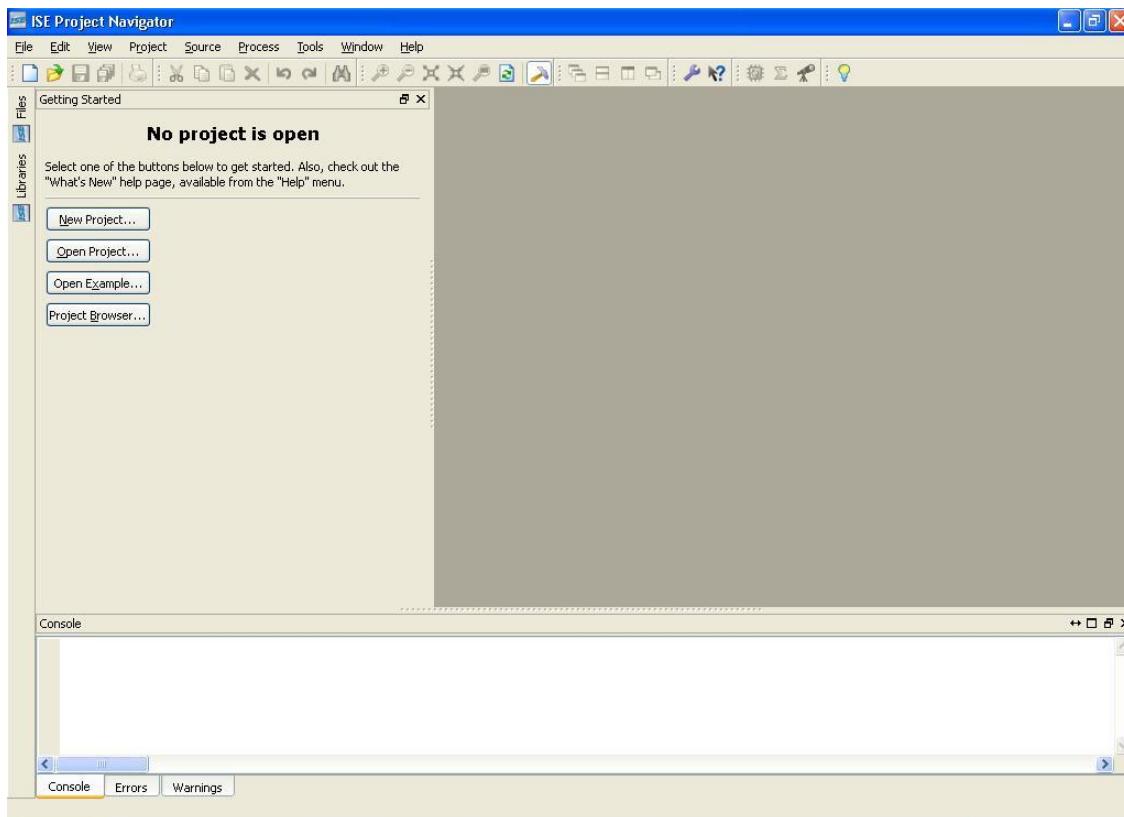


شکل ۱۷ اجرای شبیه‌سازی با مقادیر جدید

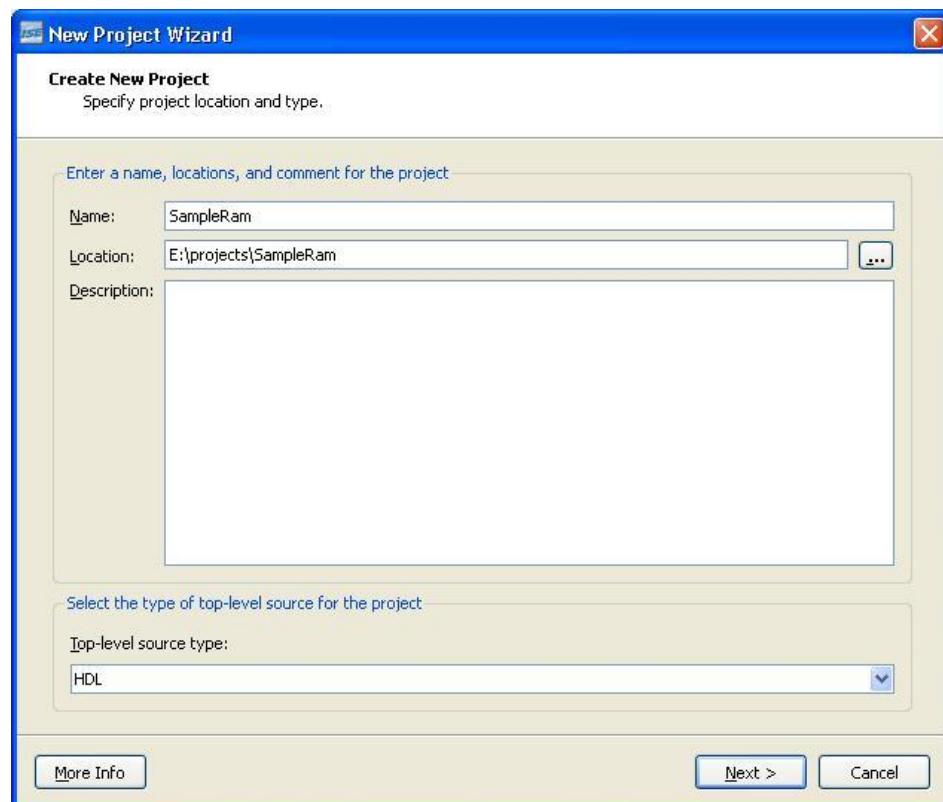
## ضمیمه ۳

نحوه کار کردن با بورد **FPGA**

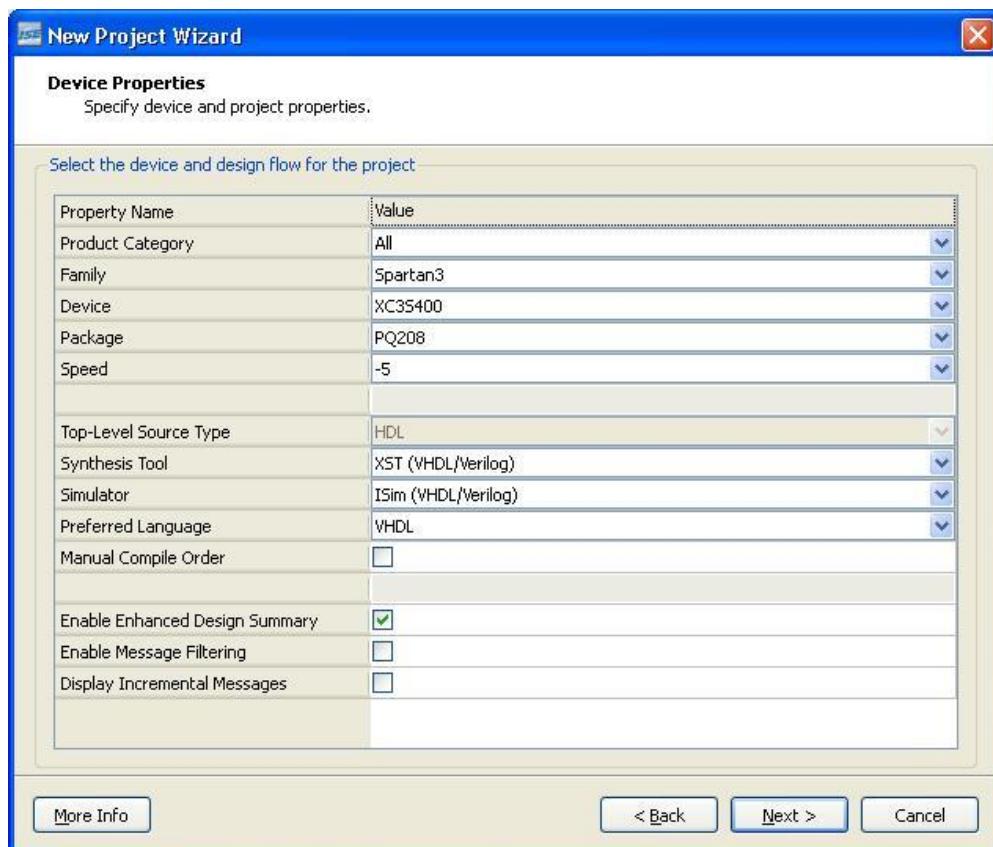
## فاز اول: برنامه‌ریزی FPGA



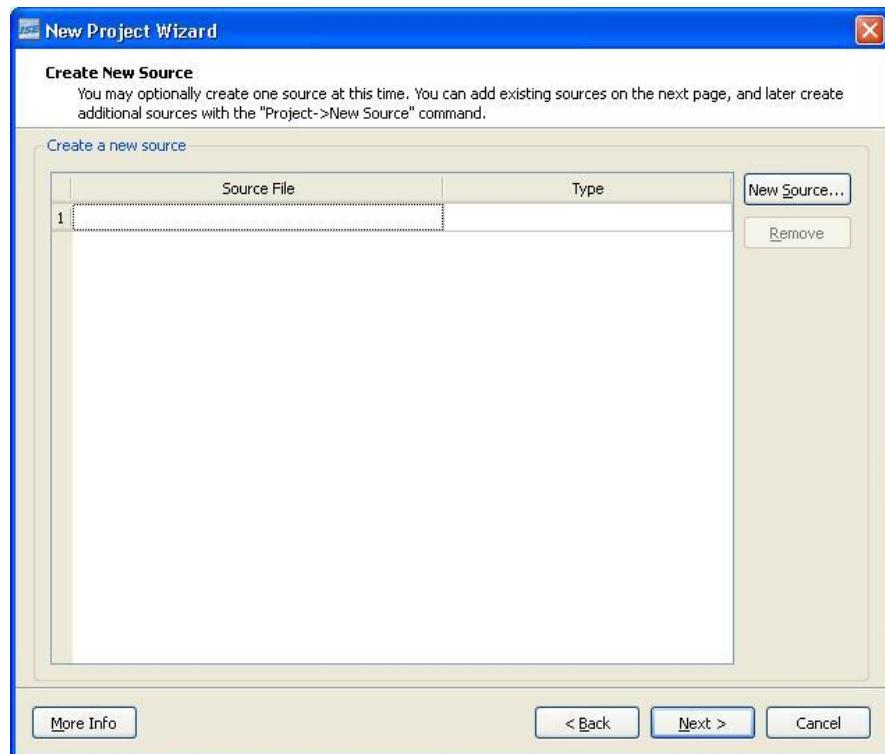
شکل ۱ صفحه شروع کار با ابزار ISE



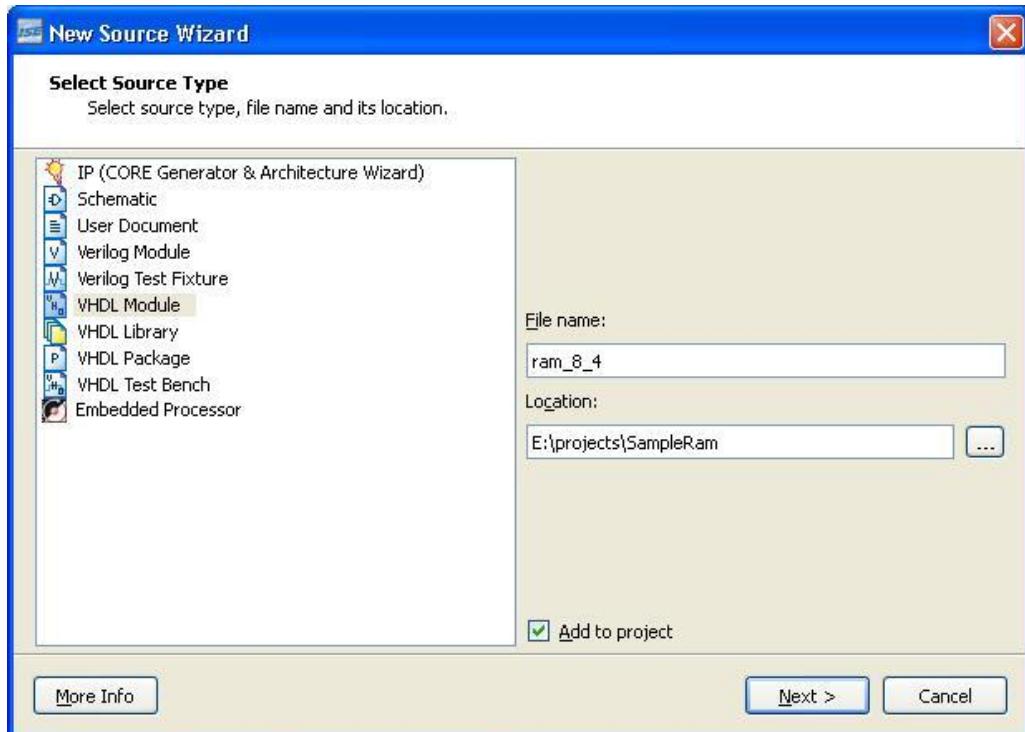
شکل ۲ ایجاد یک پروژه جدید



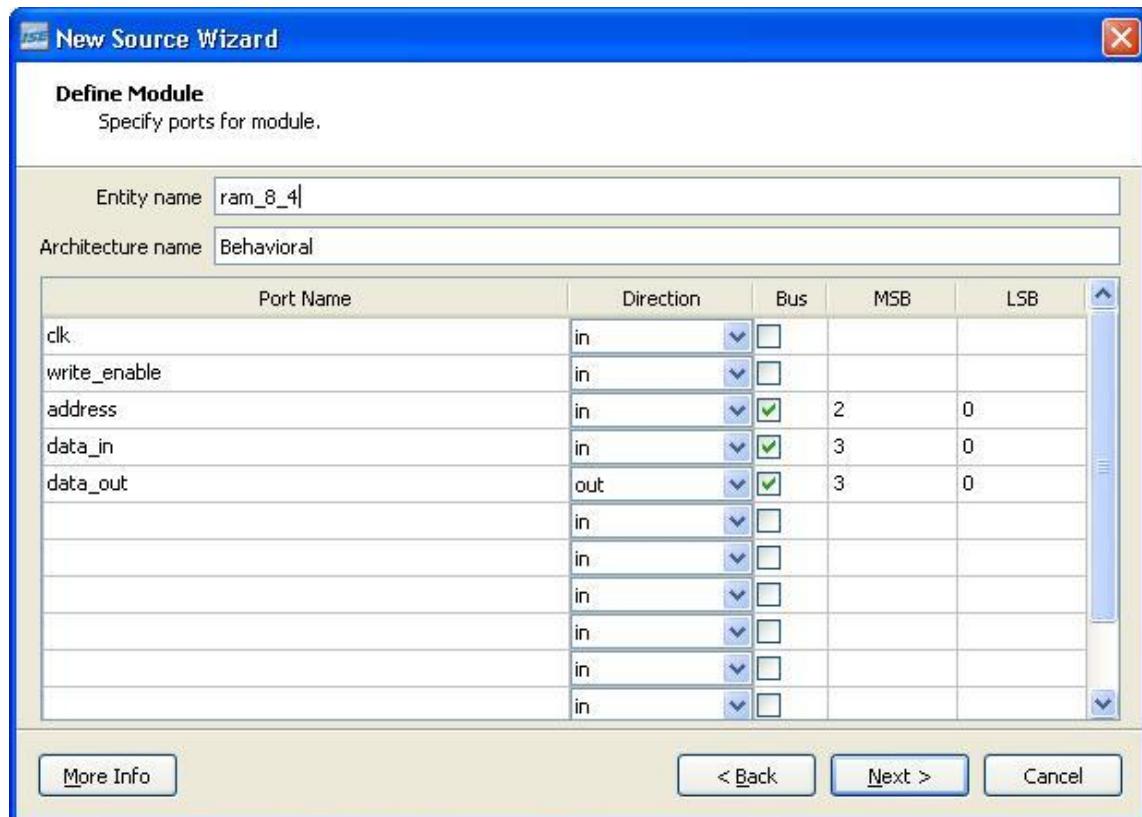
شکل ۳ تعیین ویژگی های FPGA



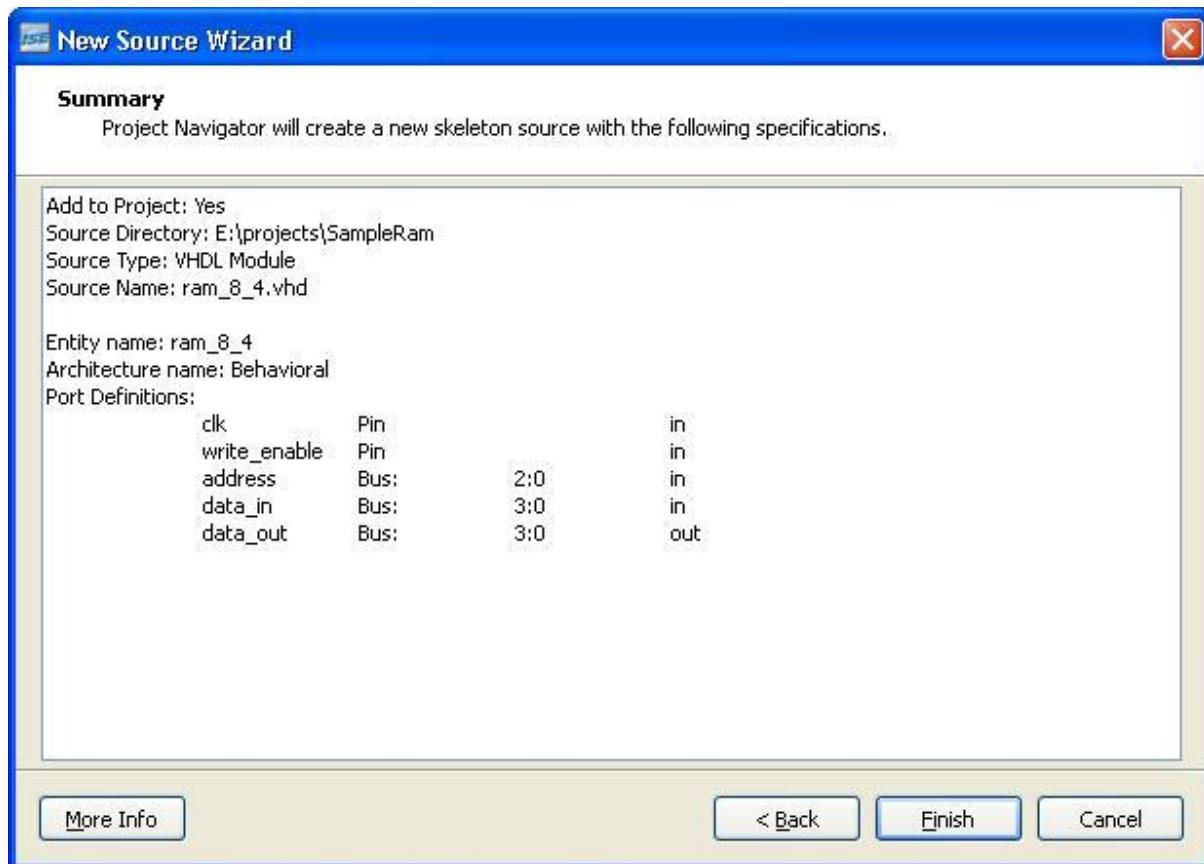
شکل ۴ ایجاد یک فایل برای entity



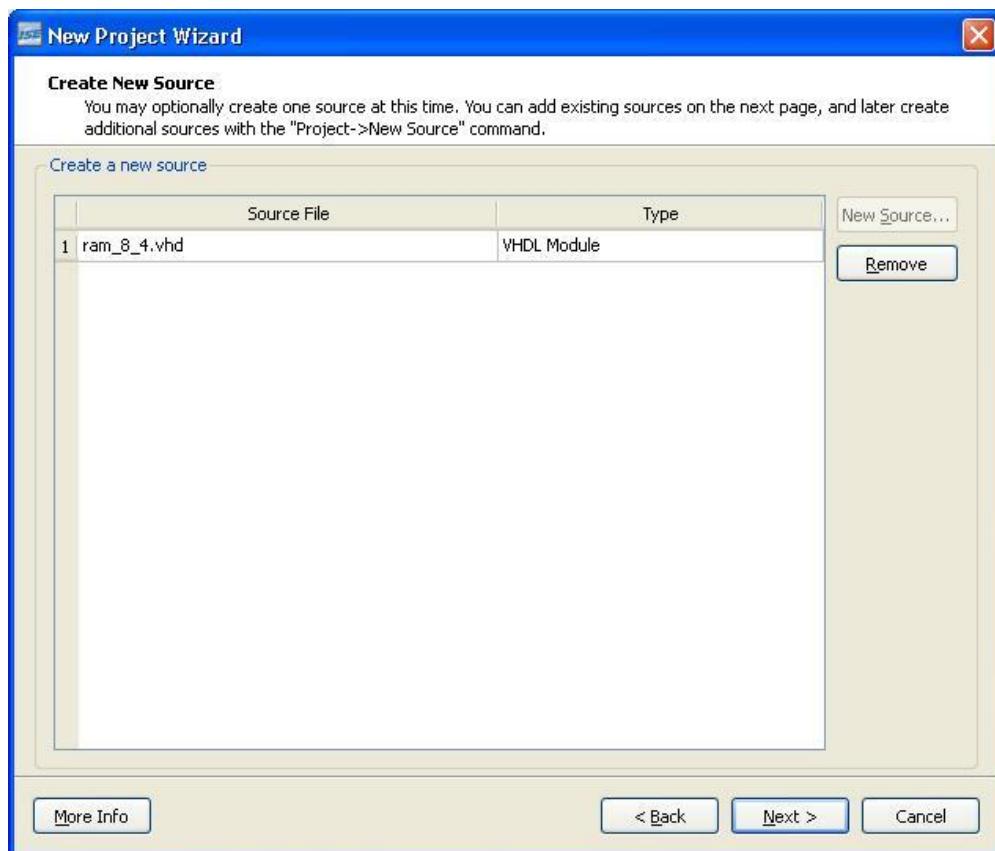
شکل ۵ تعیین نوع و نام فایل



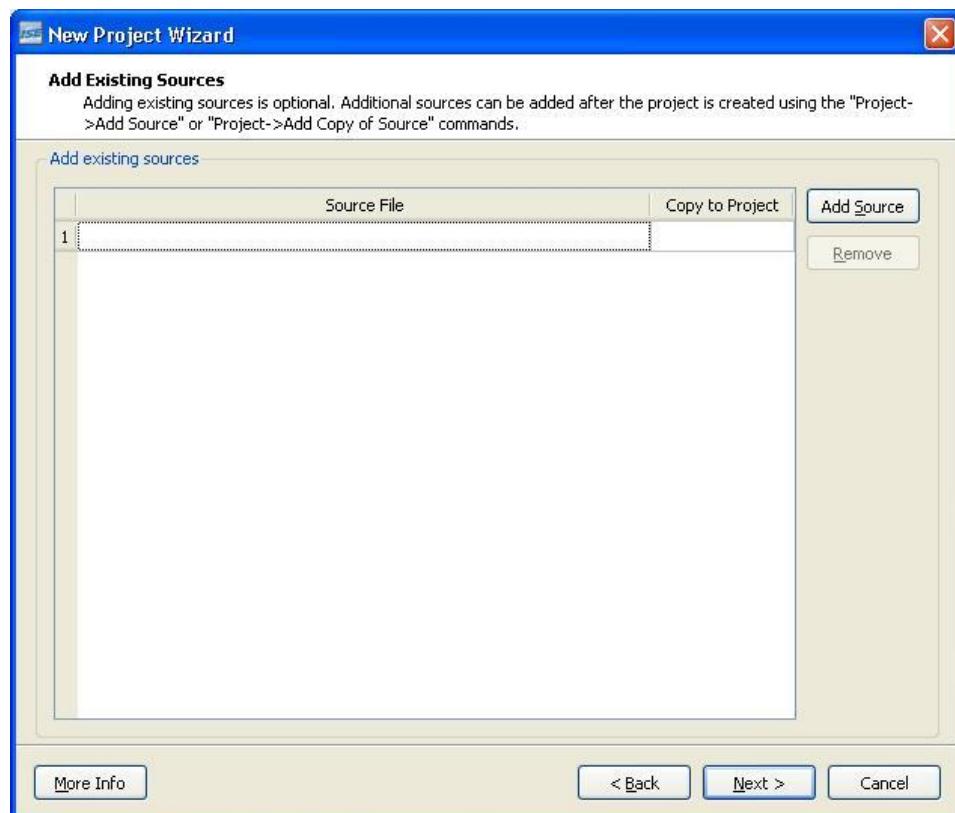
شکل ۶ تعیین پورت های ورودی و خروجی



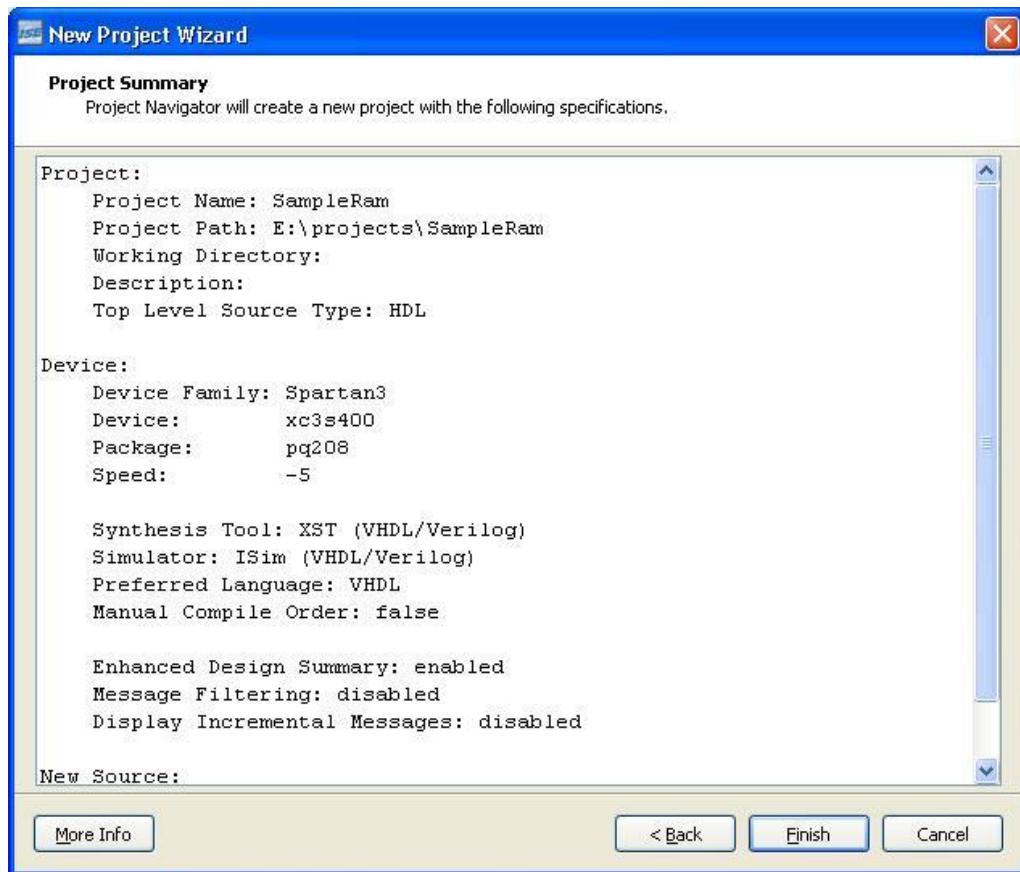
شکل ۷ خلاصه از ویژگی های فایل ایجاد شده



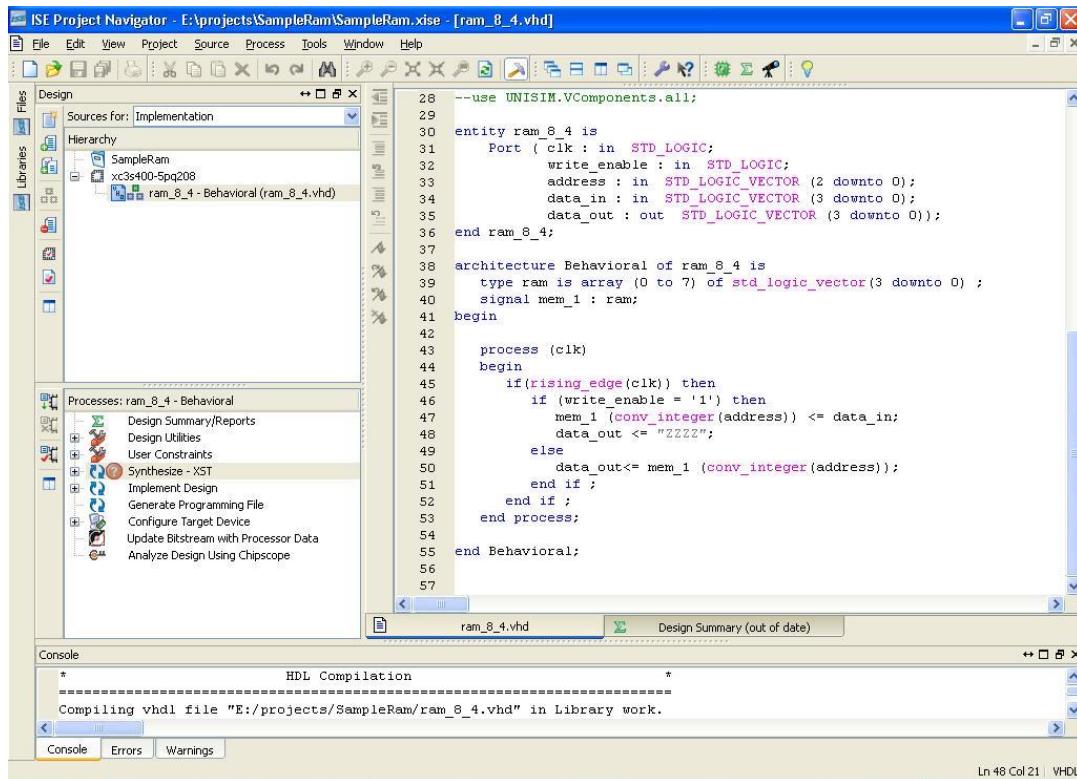
شکل ۸ لیست فایل های ایجاد شده



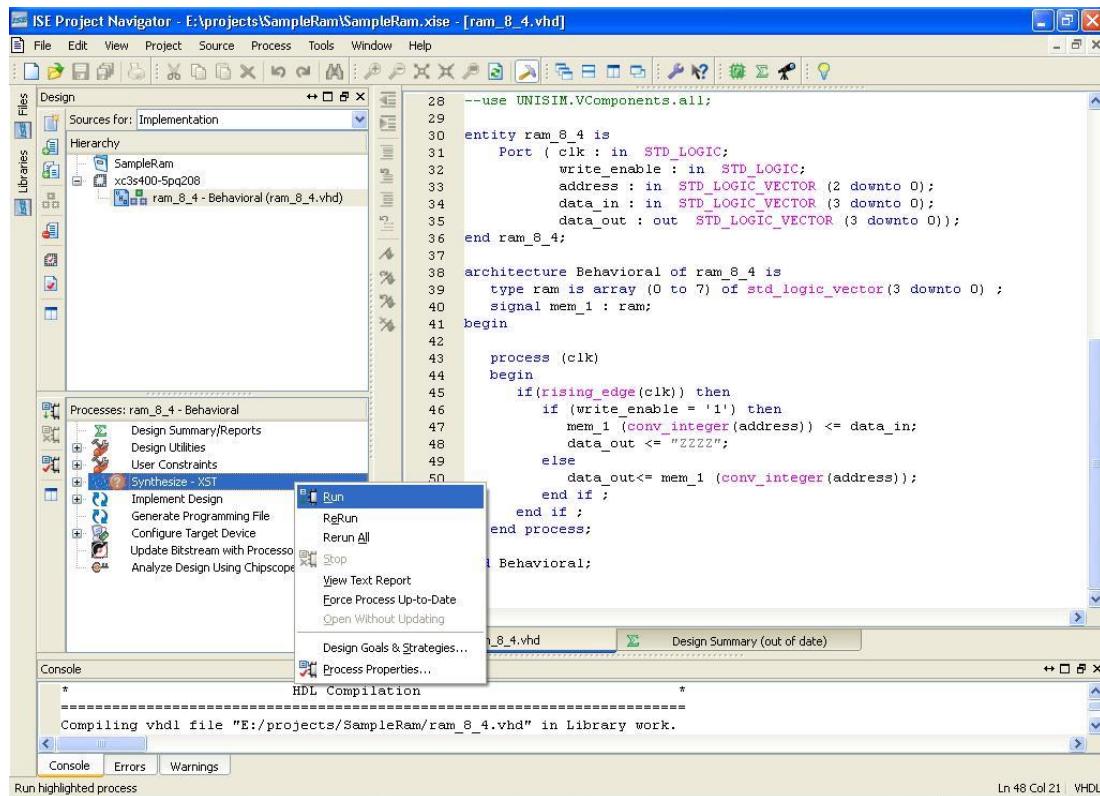
شکل ۹ افزودن فایل هایی که از قبل وجود دارند



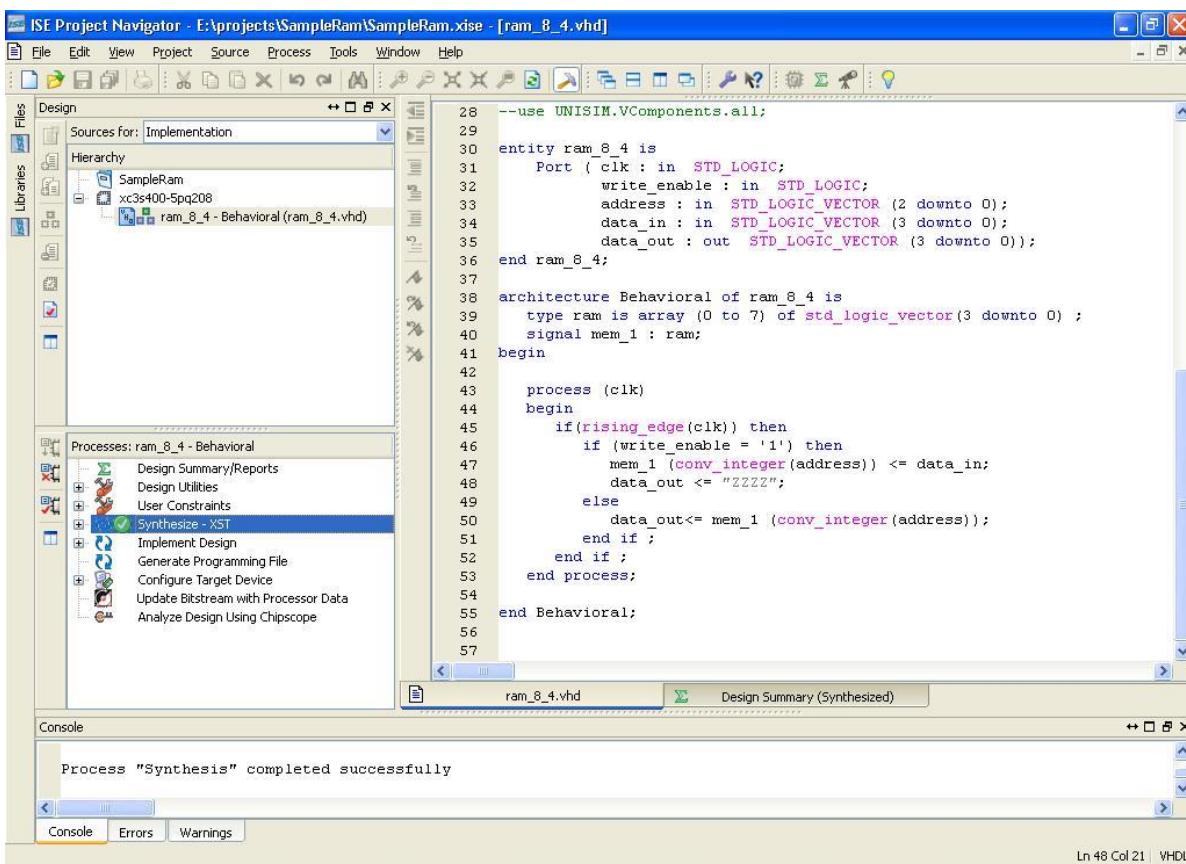
شکل ۱۰ خلاصه ای از ویژگی های پروژه ایجاد شده



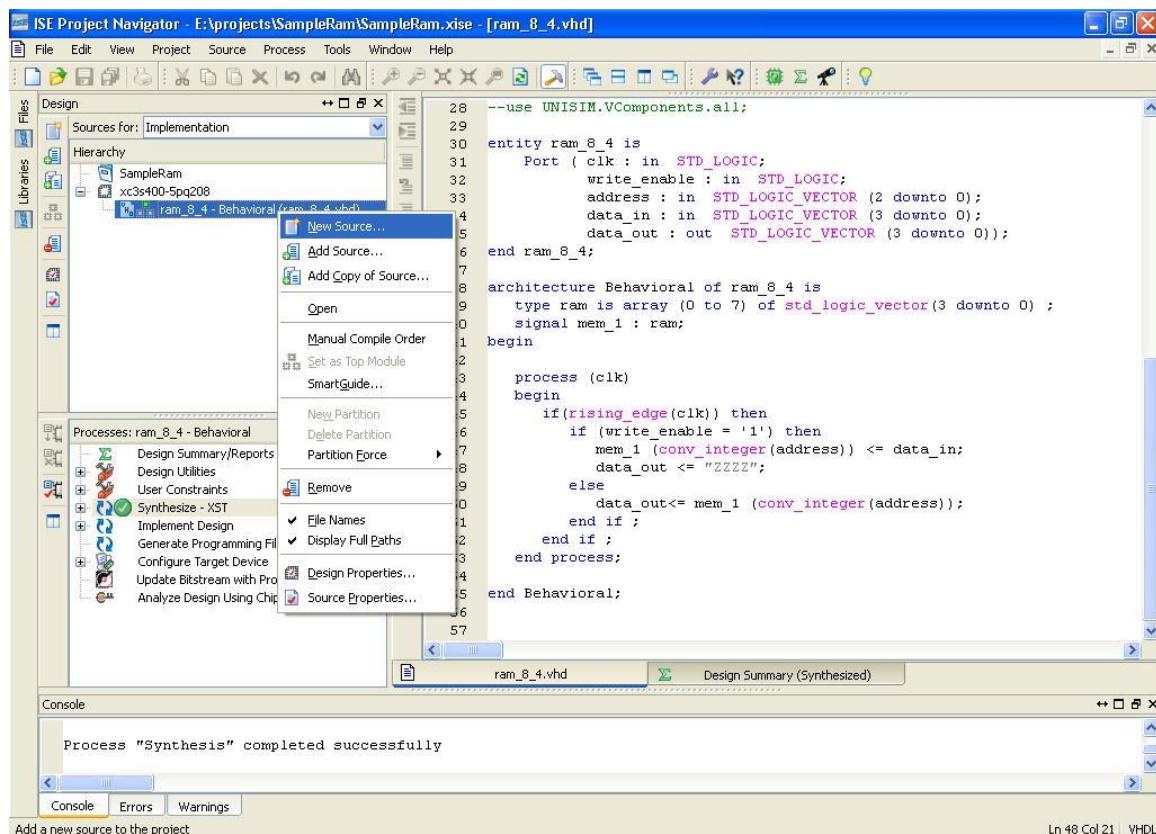
شکل ۱۱ نوشتن کد طراحی



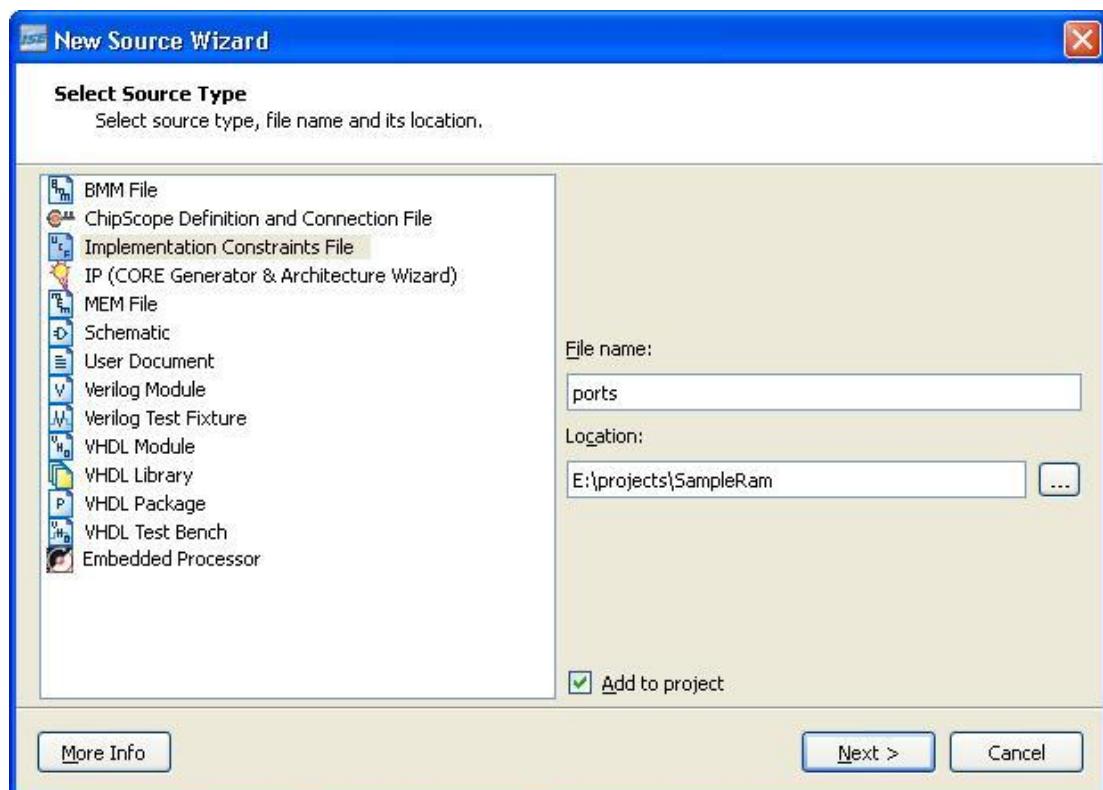
شکل ۱۲ منوی سنتز



شکل ۱۳ نتیجه سنتز



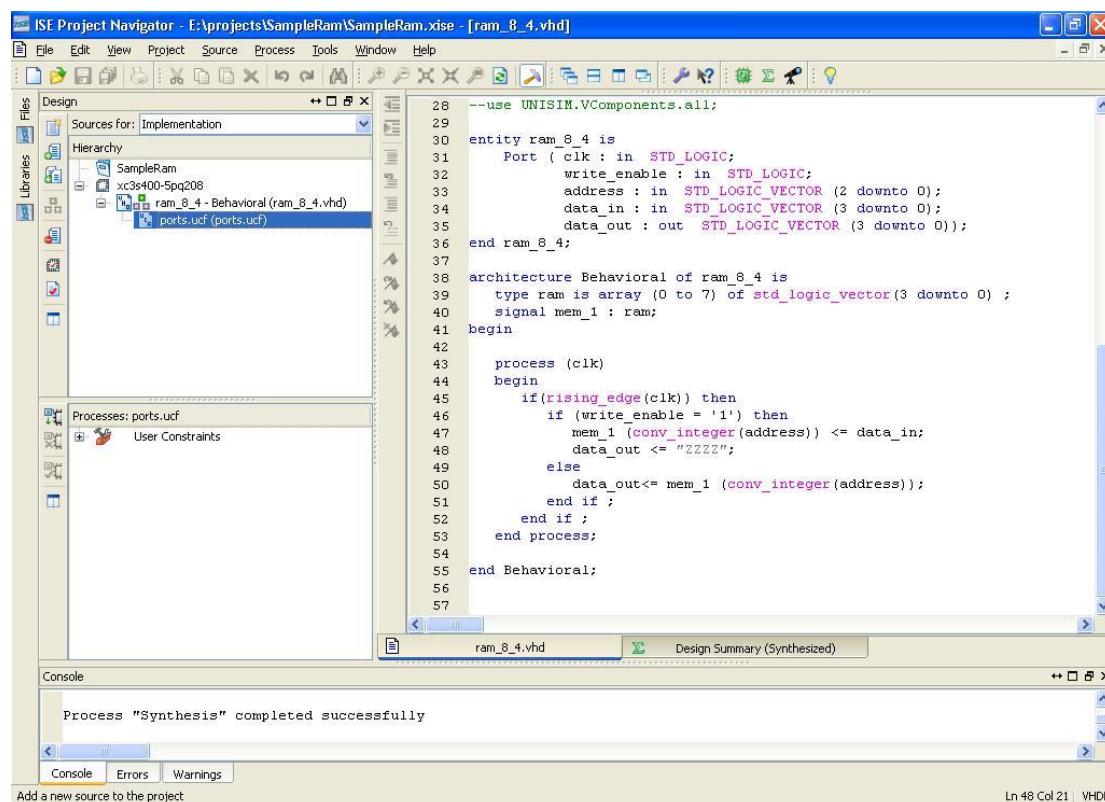
شکل ۱۴ منوی ایجاد فایل محدودیت (تعیین پورت های FPGA)



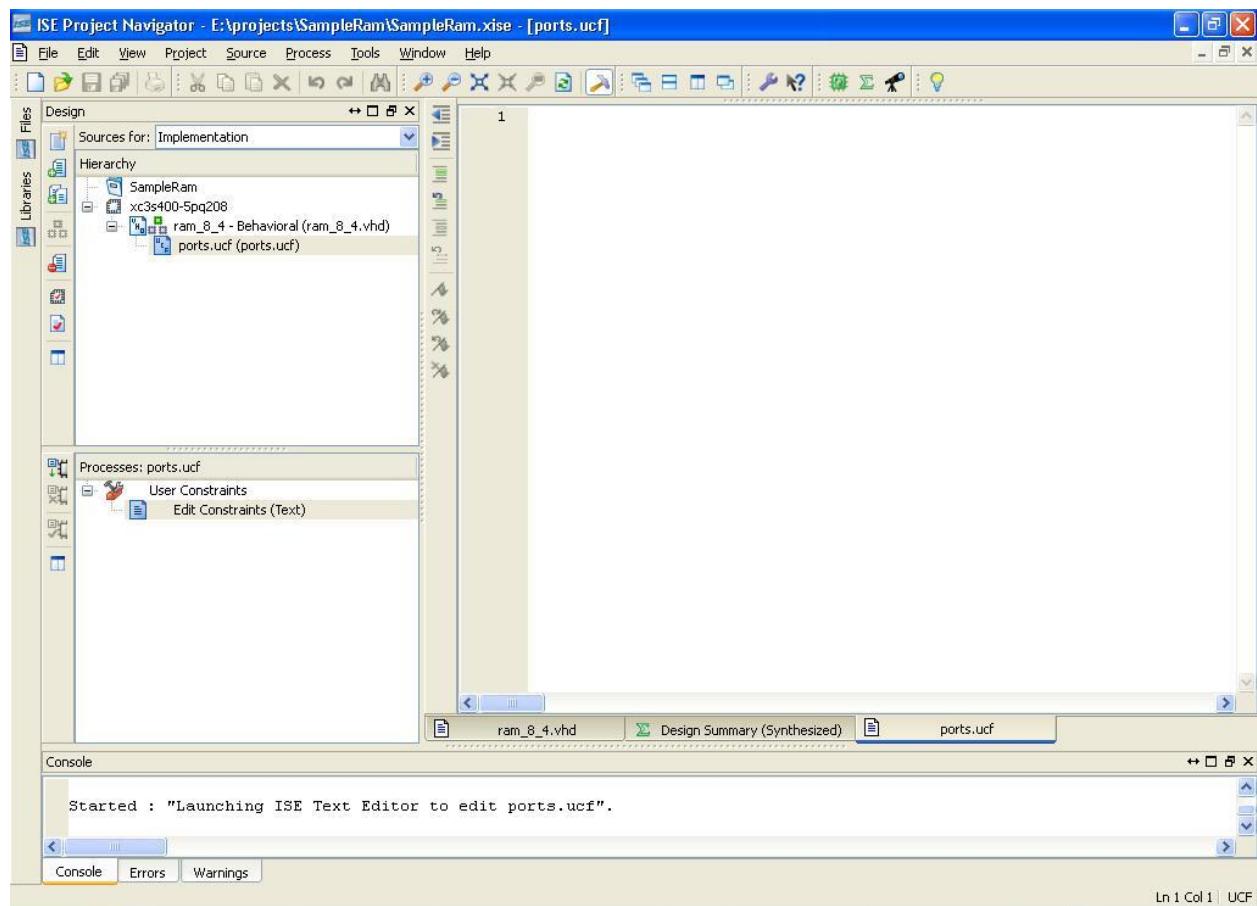
شکل ۱۵ تعیین نام فایل محدودیت



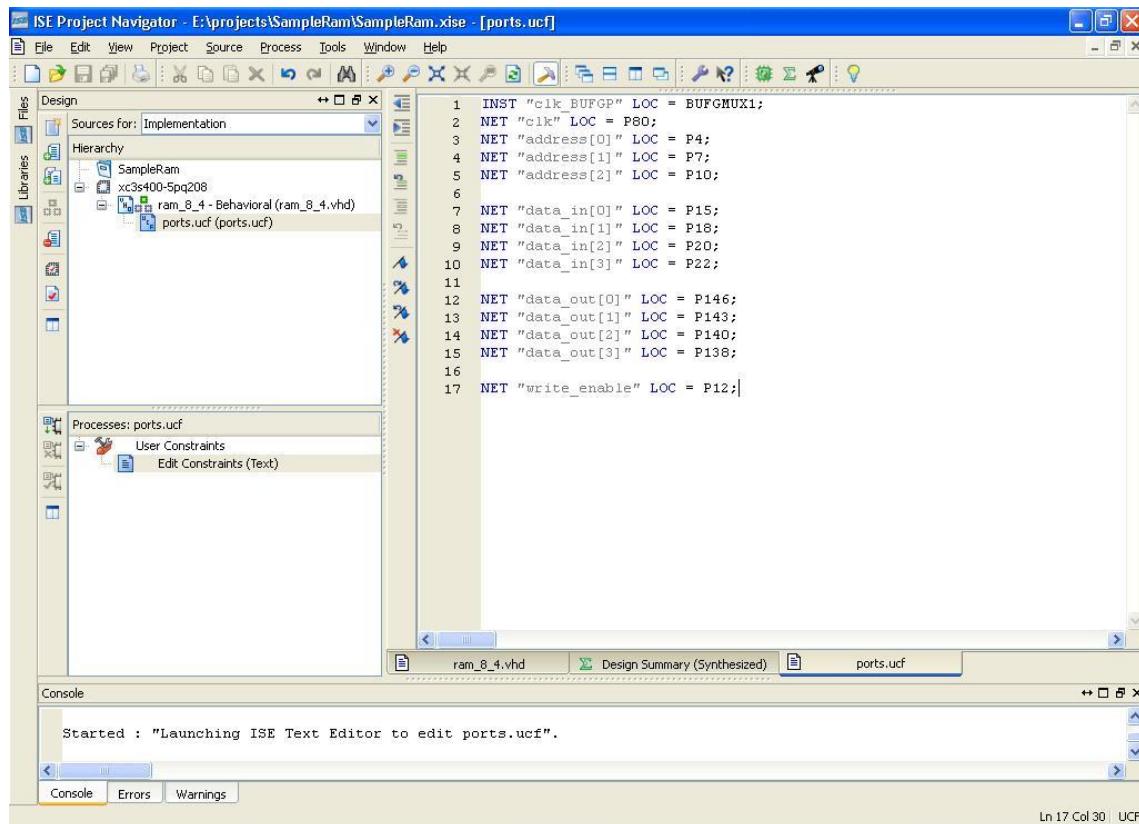
شکل ۱۶ ویزگی های فایل محدودیت ایجاد شده



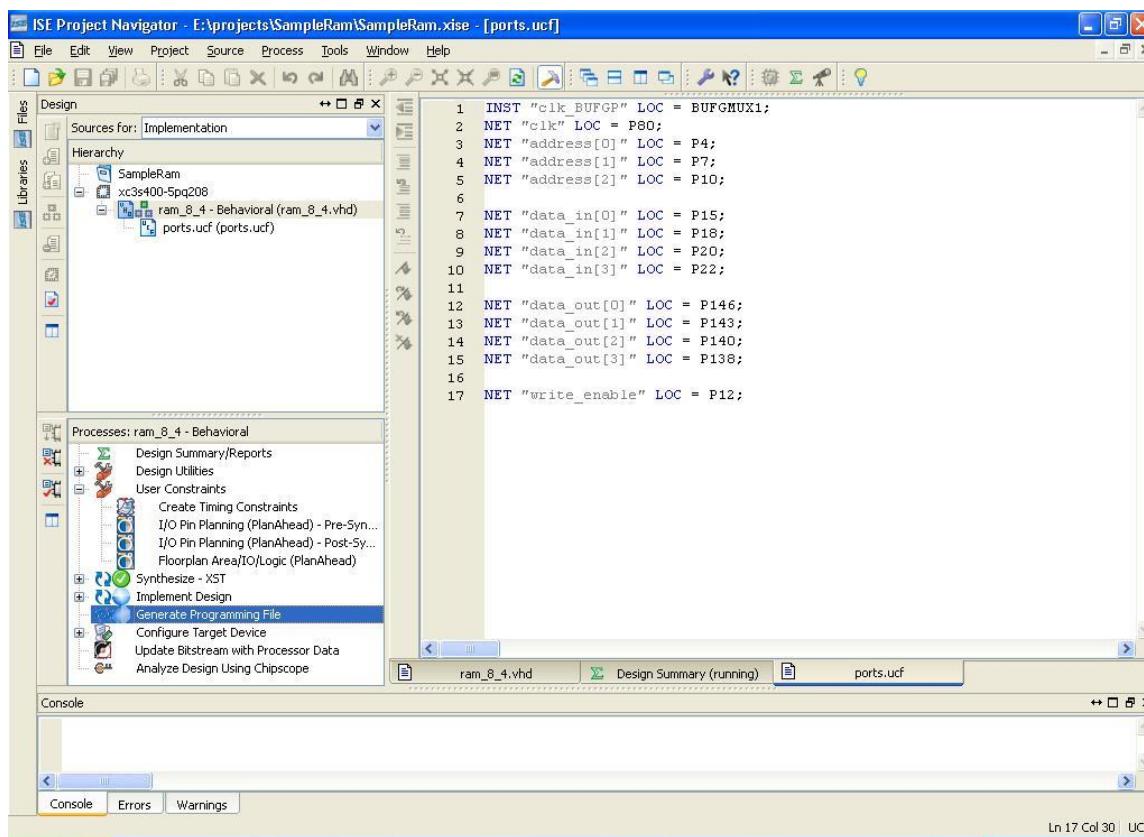
شکل ۱۷ فایل محدودیت در منوی Hierarchy



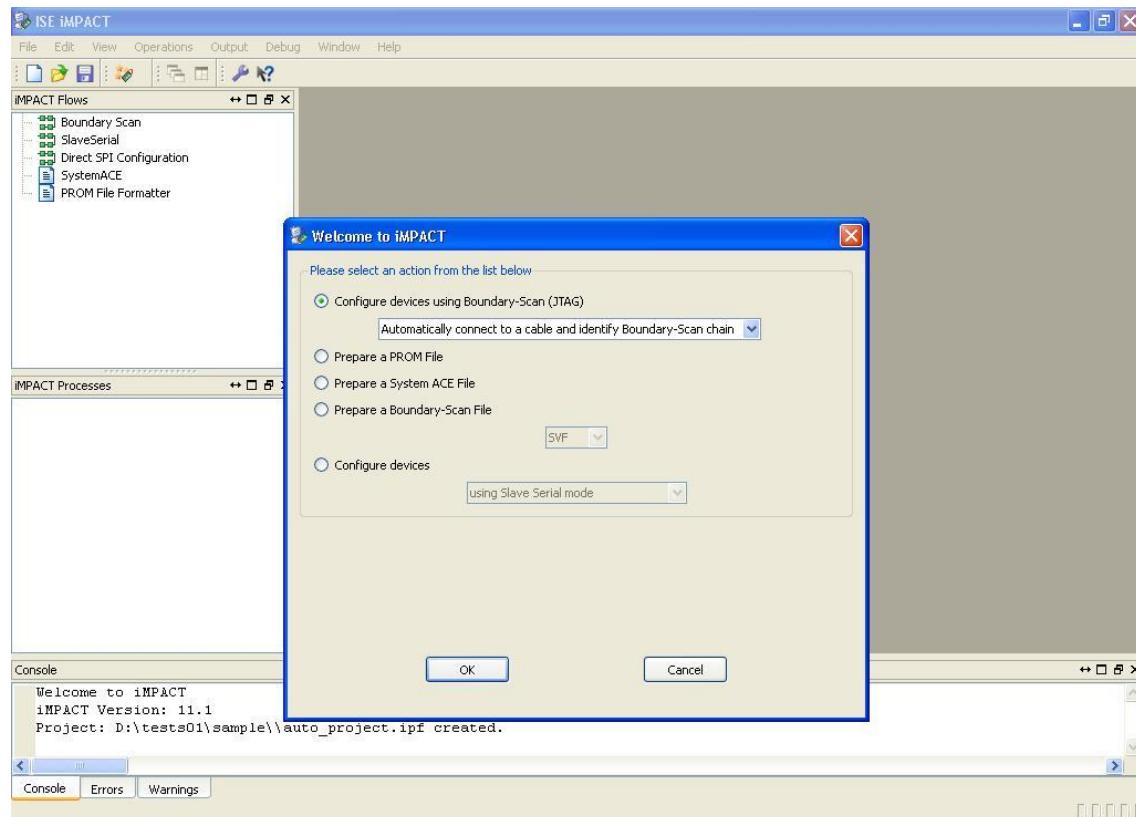
شکل ۱۸ ویرایش فایل محدودیت به صورت متنی



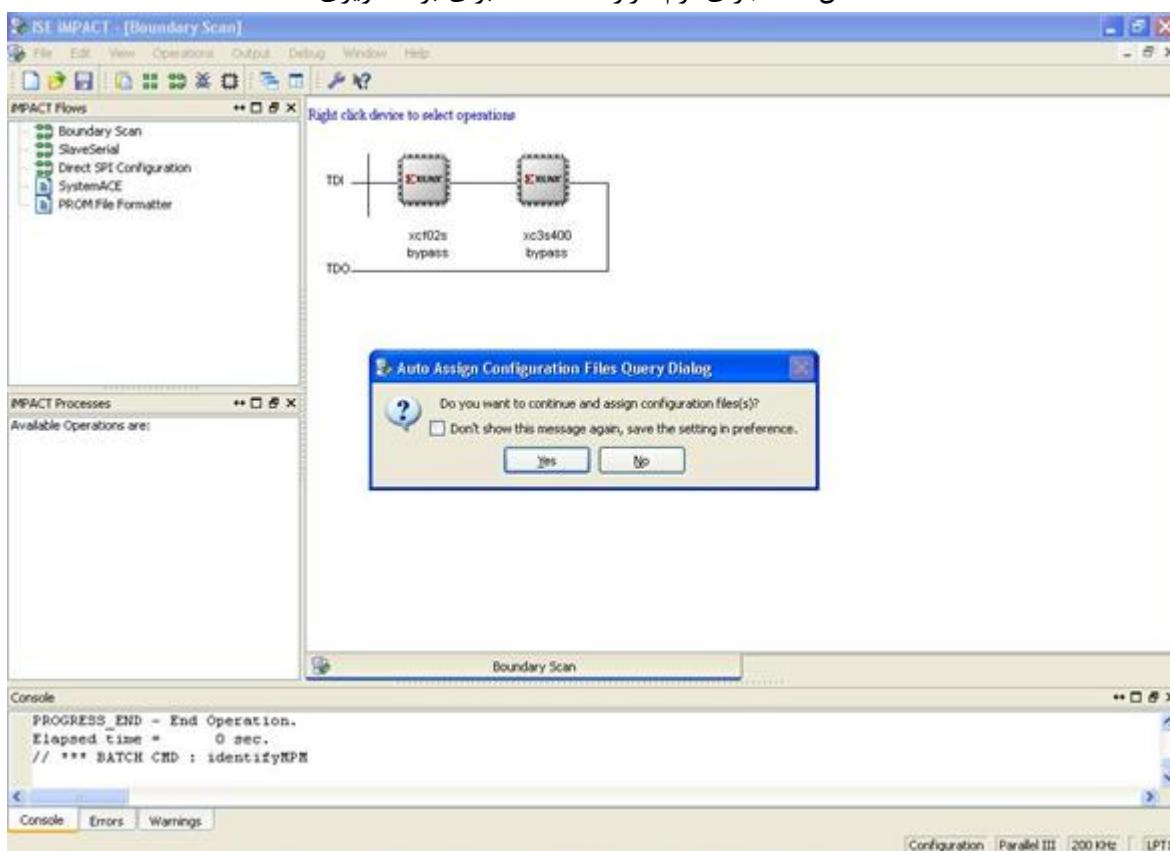
شکل ۱۹ تعیین port های طرح بر روی FPGA



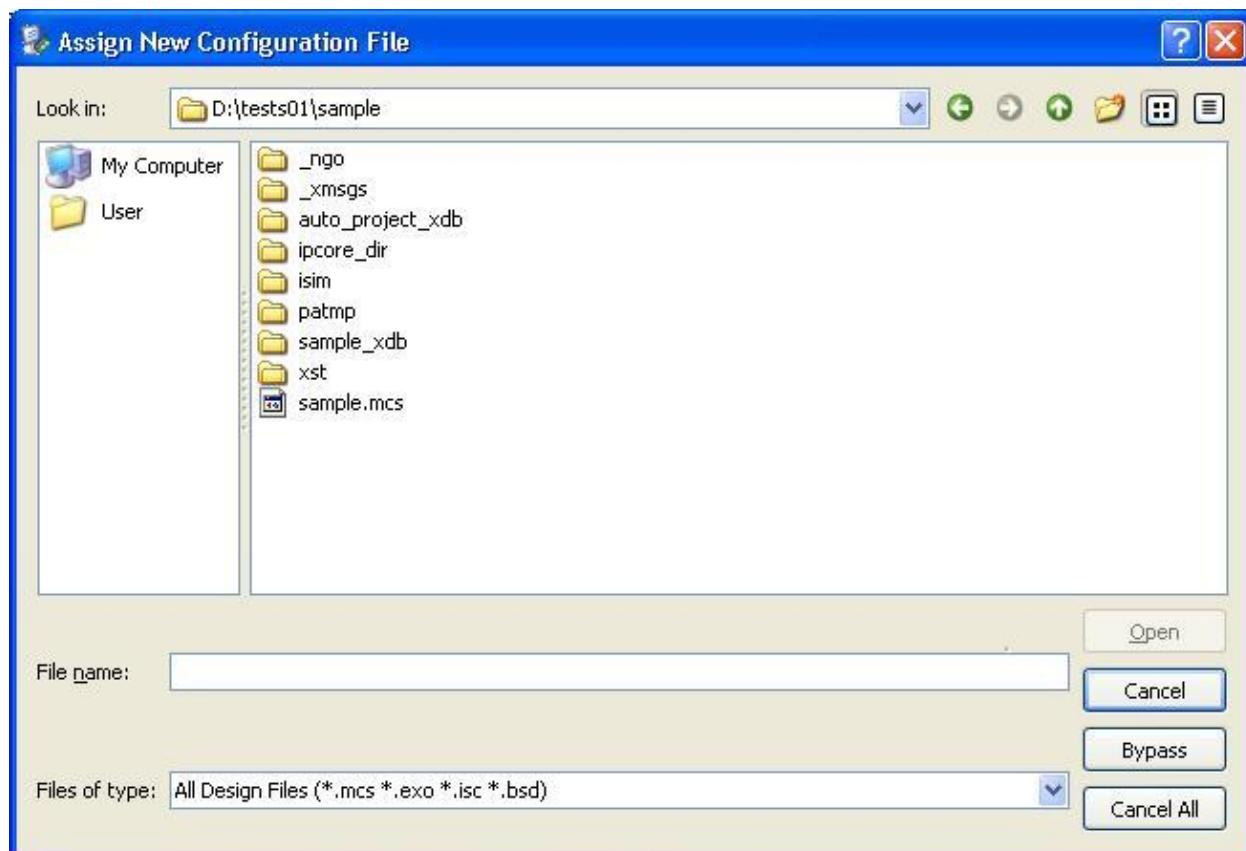
شکل ۲۰ ایجاد فایل برنامه ریزی



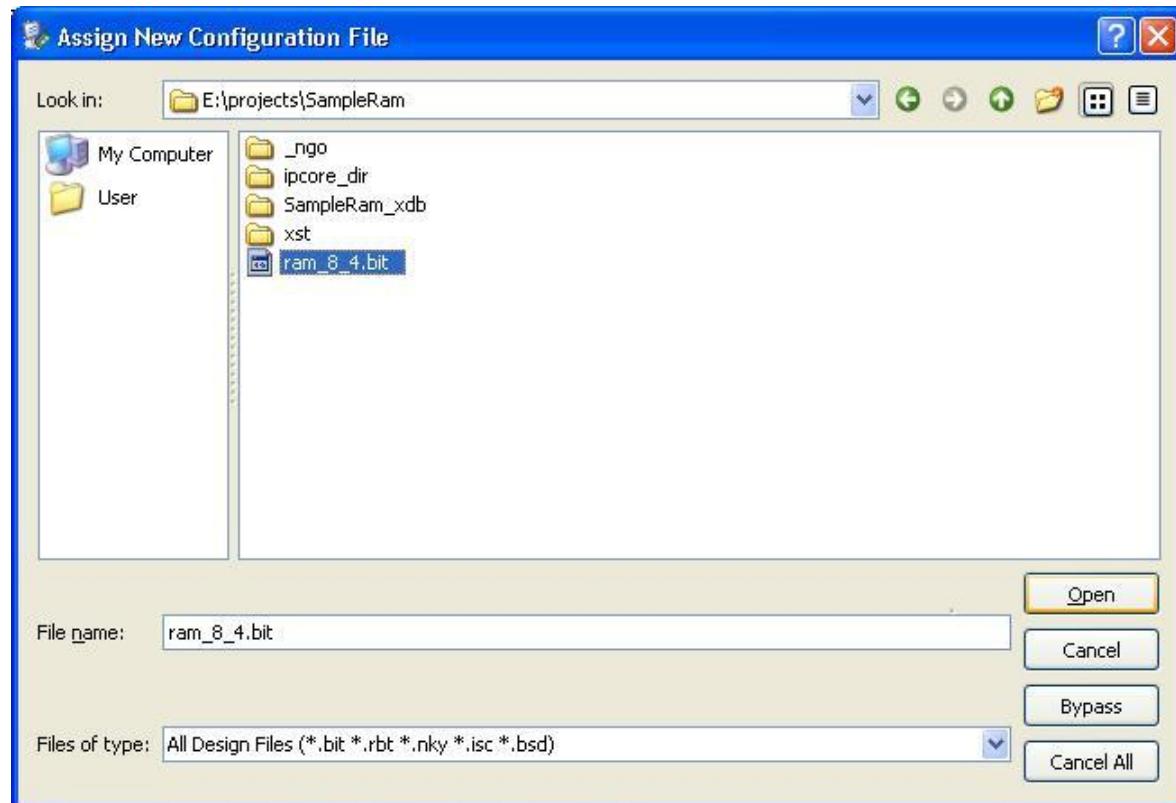
شکل ۲۱ اجرای نرم افزار iMPACT برای برنامه ریزی



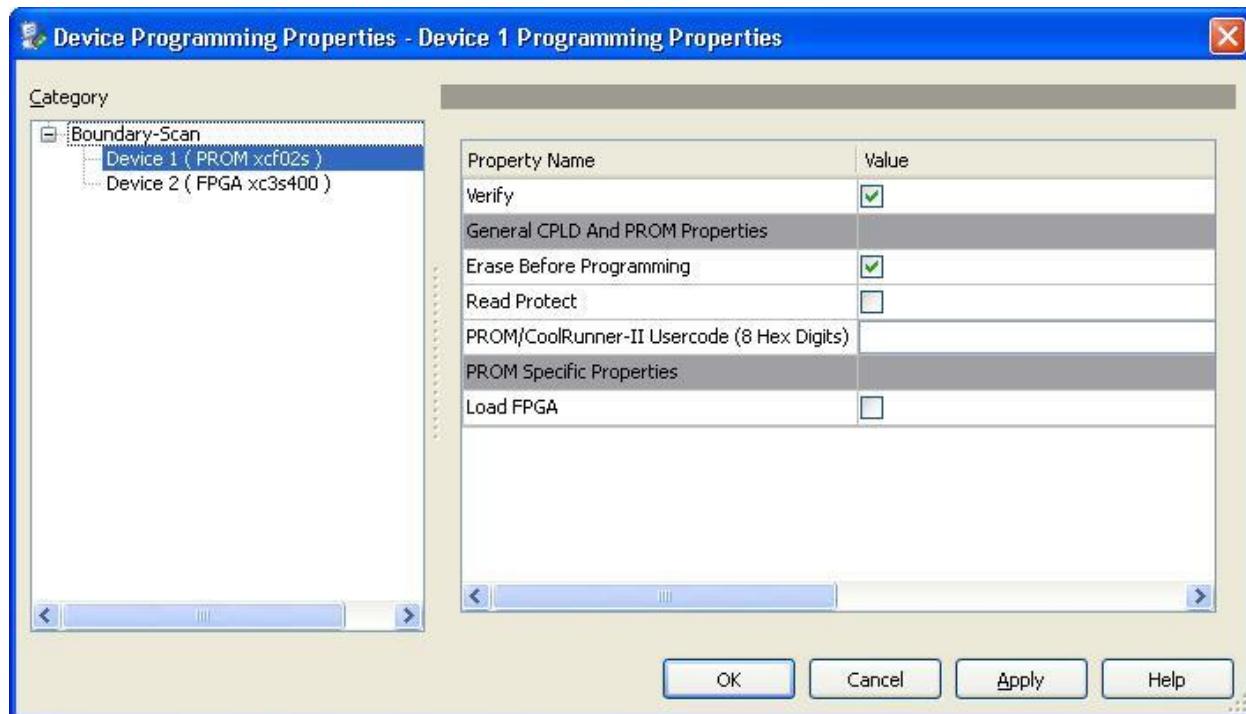
شکل ۲۲ ابزار های شناخته شده



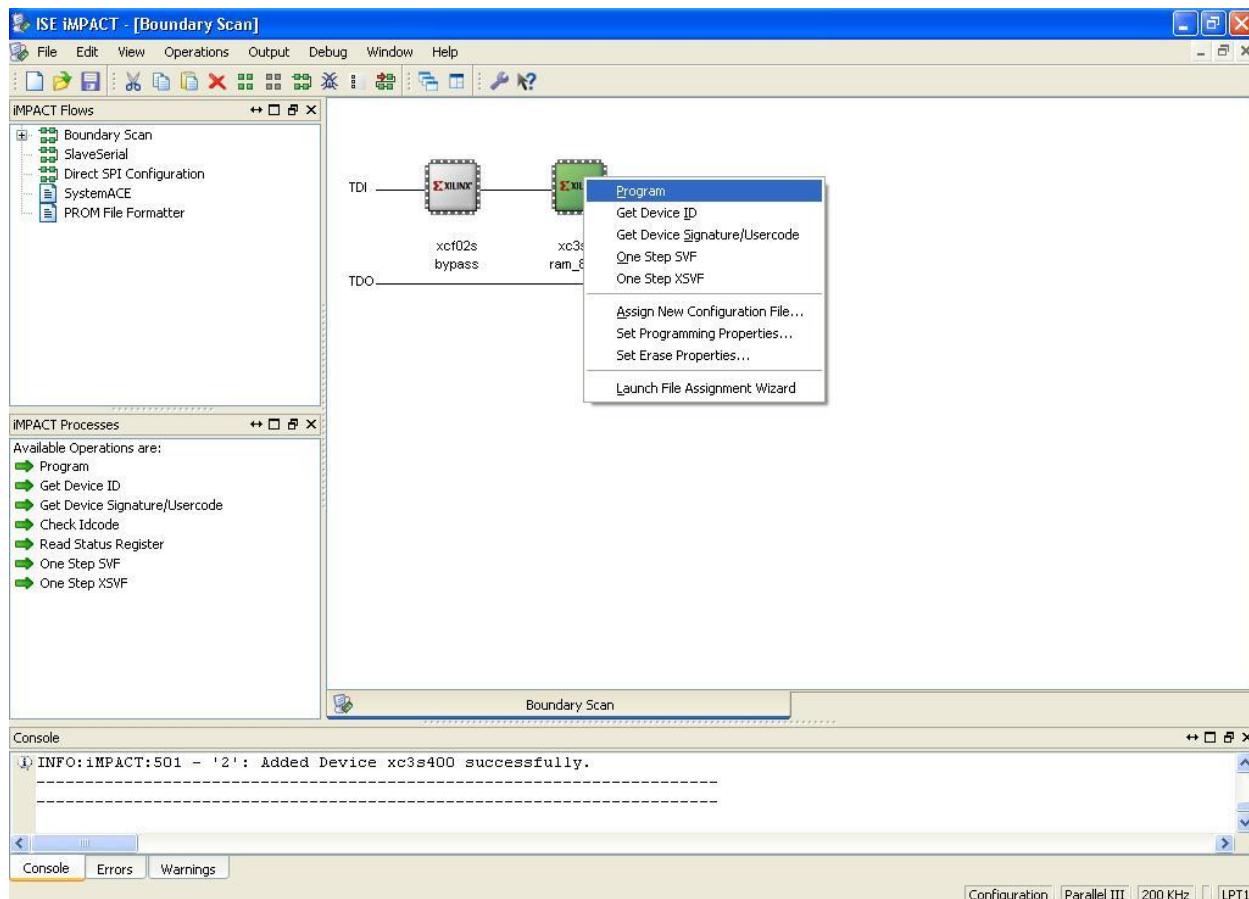
شکل ۲۳ تعیین فایل برنامه ریزی RAM موجود در روی بورد. (فعلاً این مرحله را bypass می‌کنیم.)



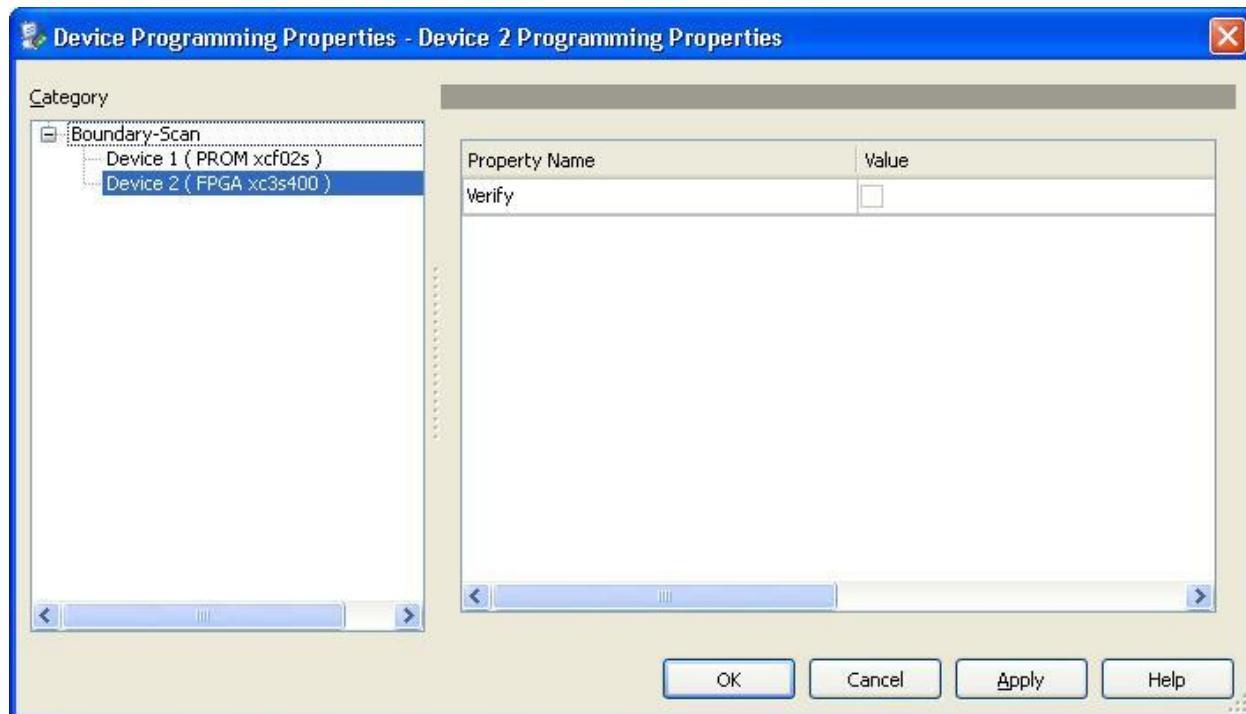
شکل ۲۴ انتخاب فایل برنامه ریزی تولید شده در مرحله قبل



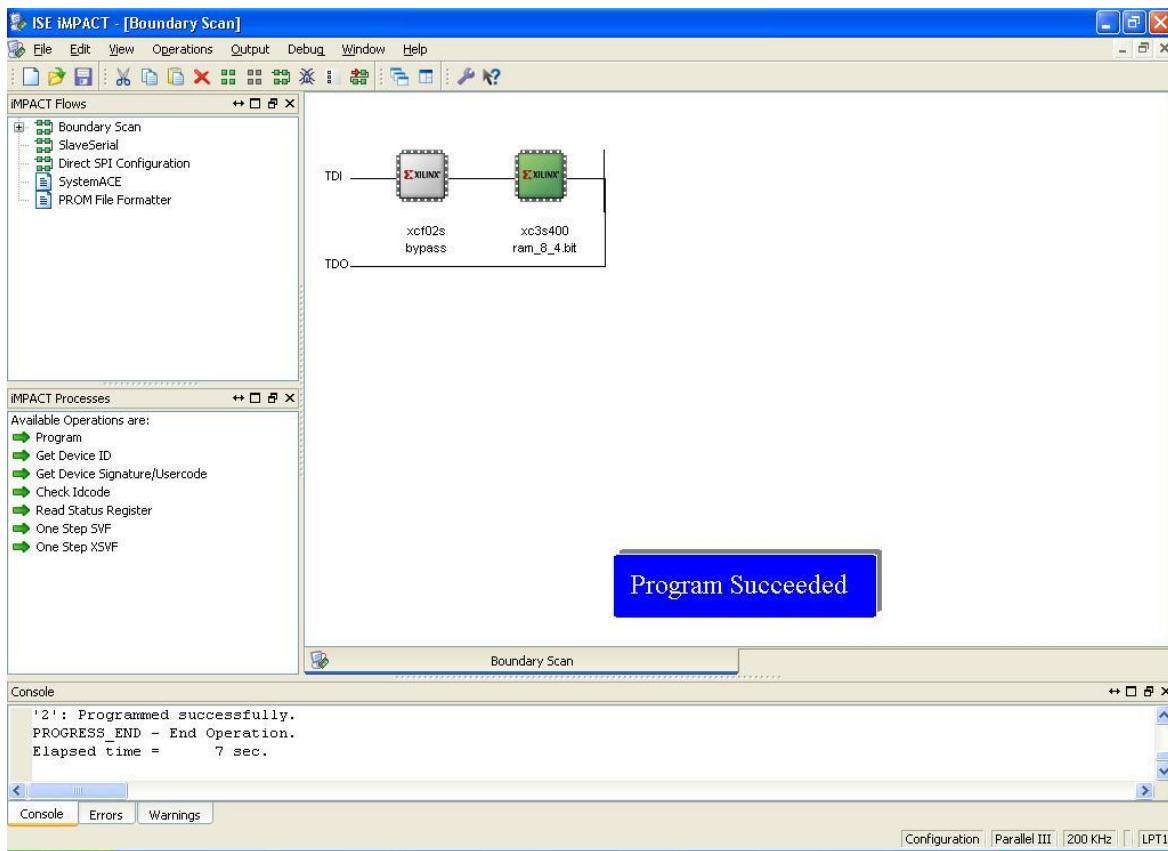
شکل ۲۵ نمایش ویژگی های برنامه ریزی ابزار



شکل ۲۶ برنامه ریزی FPGA

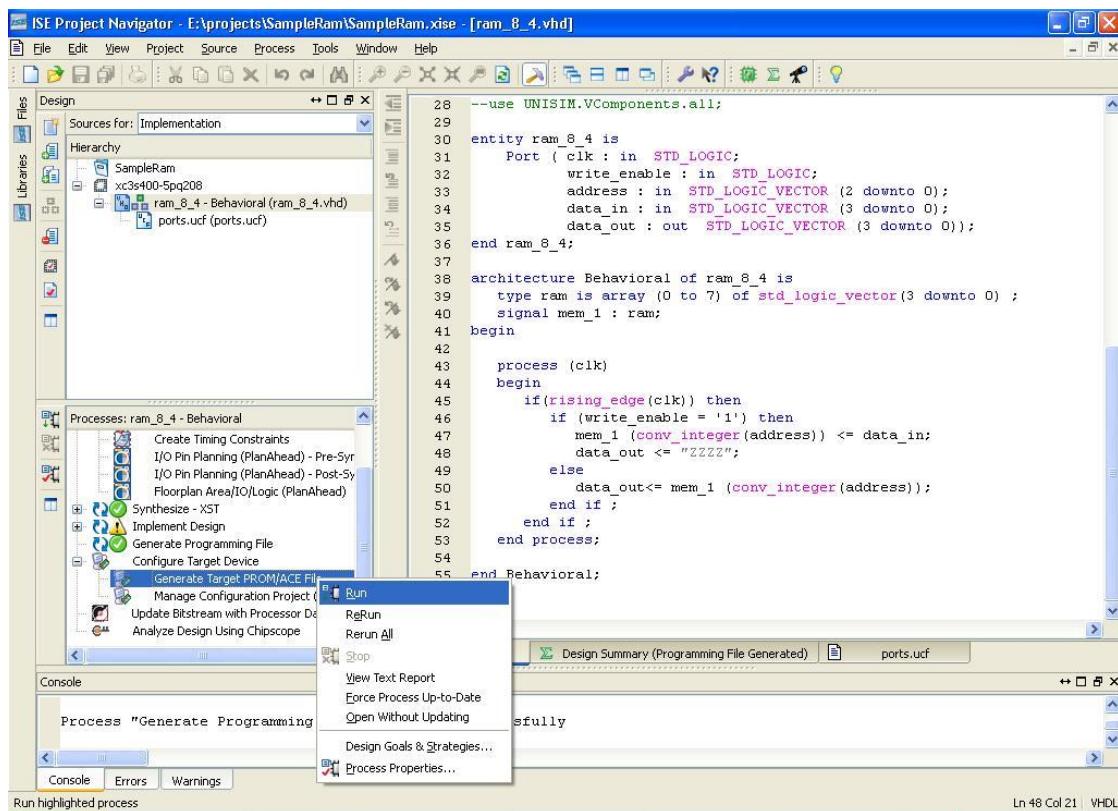


شکل ۲۷ تأیید نهایی



شکل ۲۸ وضعیت برنامه ریزی

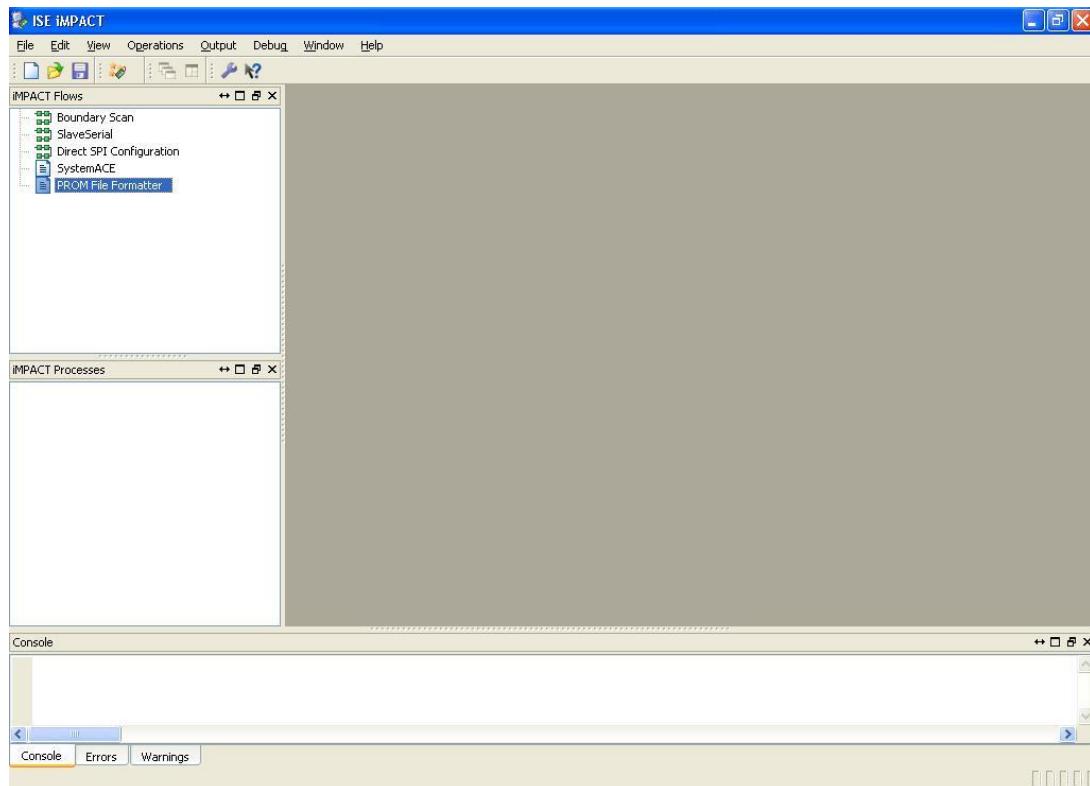
## فاز دوم: برنامه ریزی RAM روی بورد



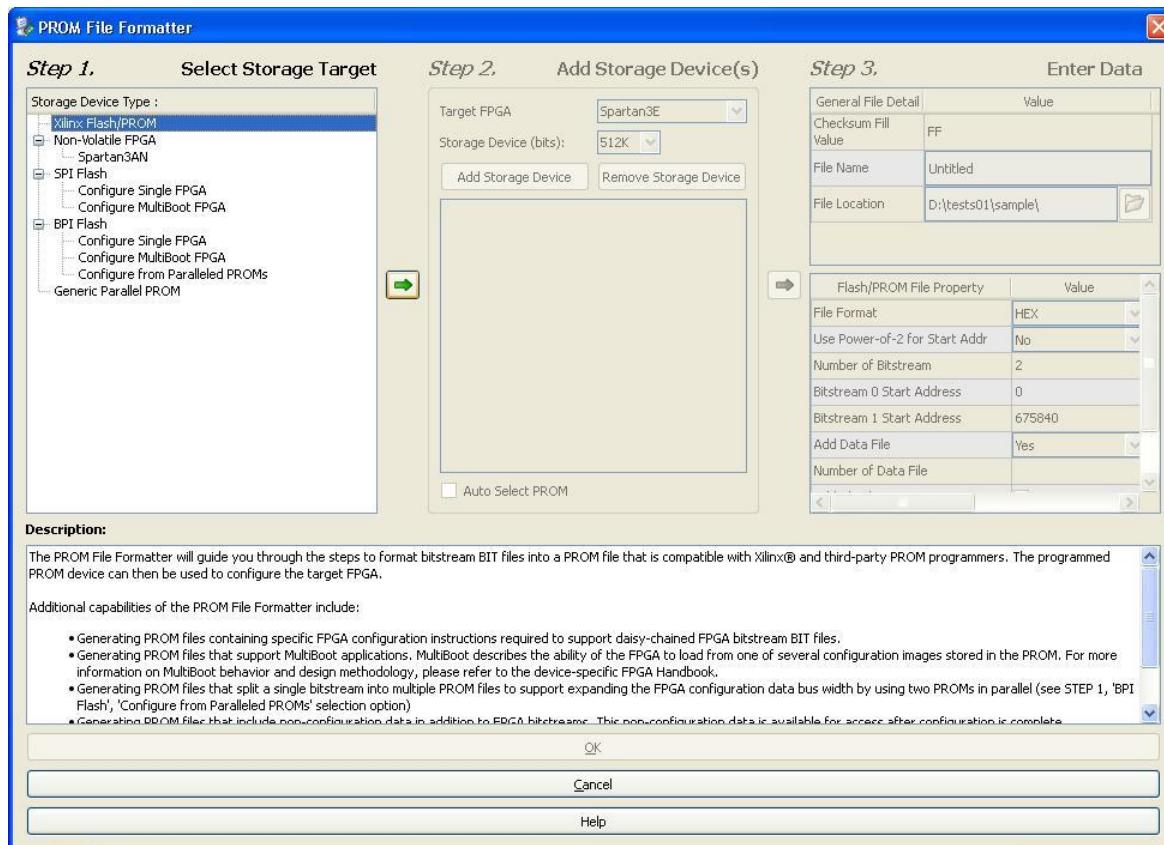
شکل ۲۹ منوی ایجاد فایل برنامه ریزی RAM



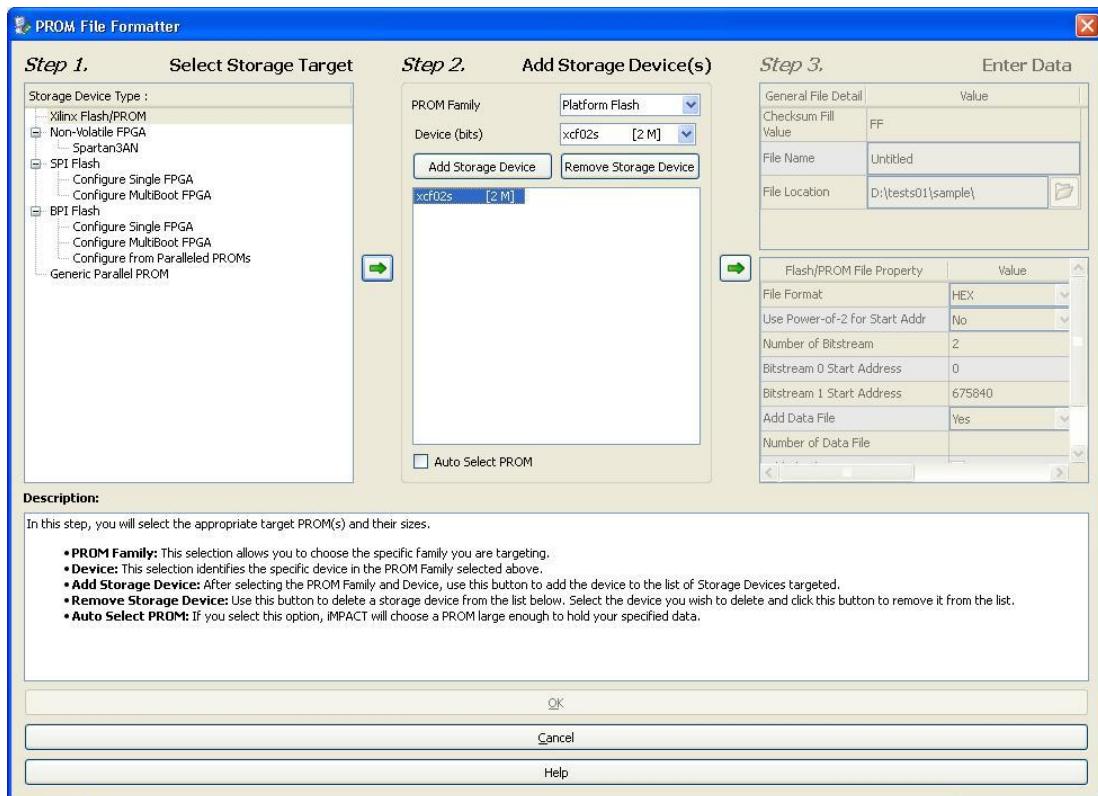
شکل ۳۰ هشدار مبنی بر ایجاد پروژه iMPACT جدید



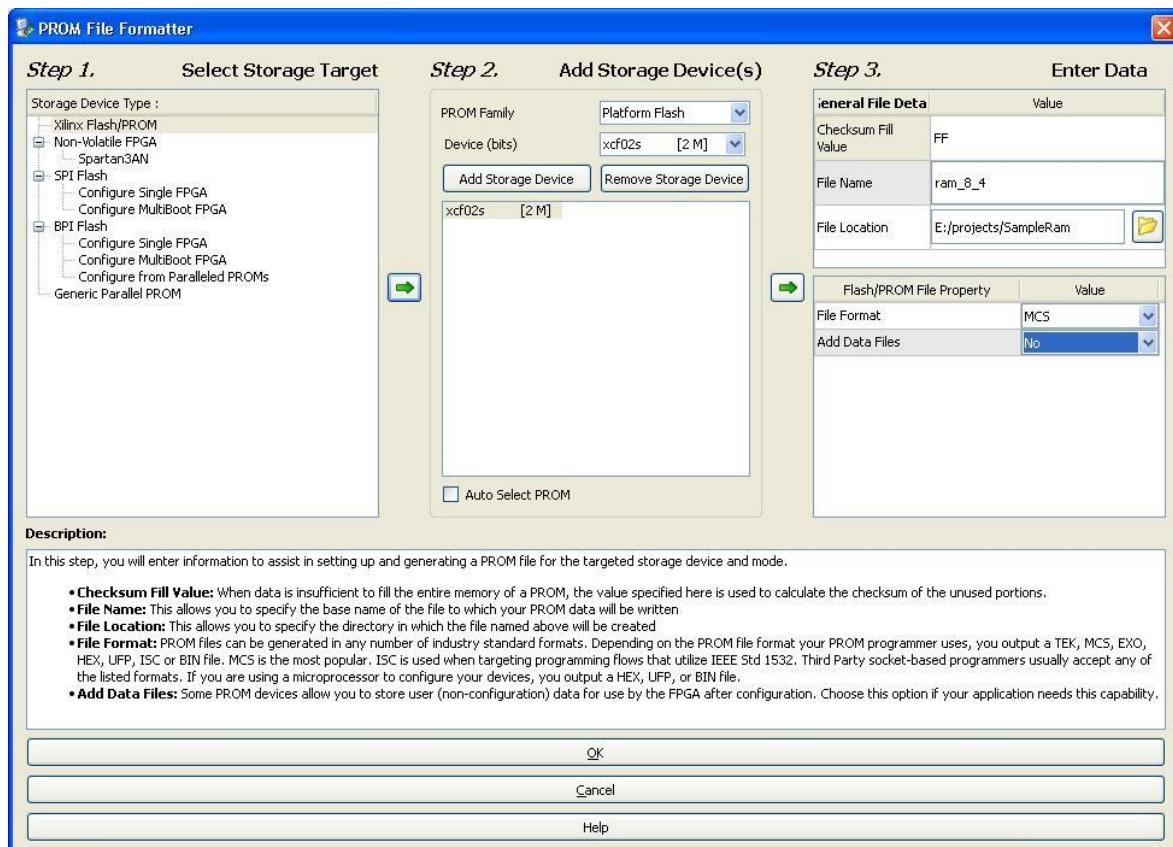
شکل ۳۱ منوی ایجاد فایل برنامه ریزی در پنجره iMPACT



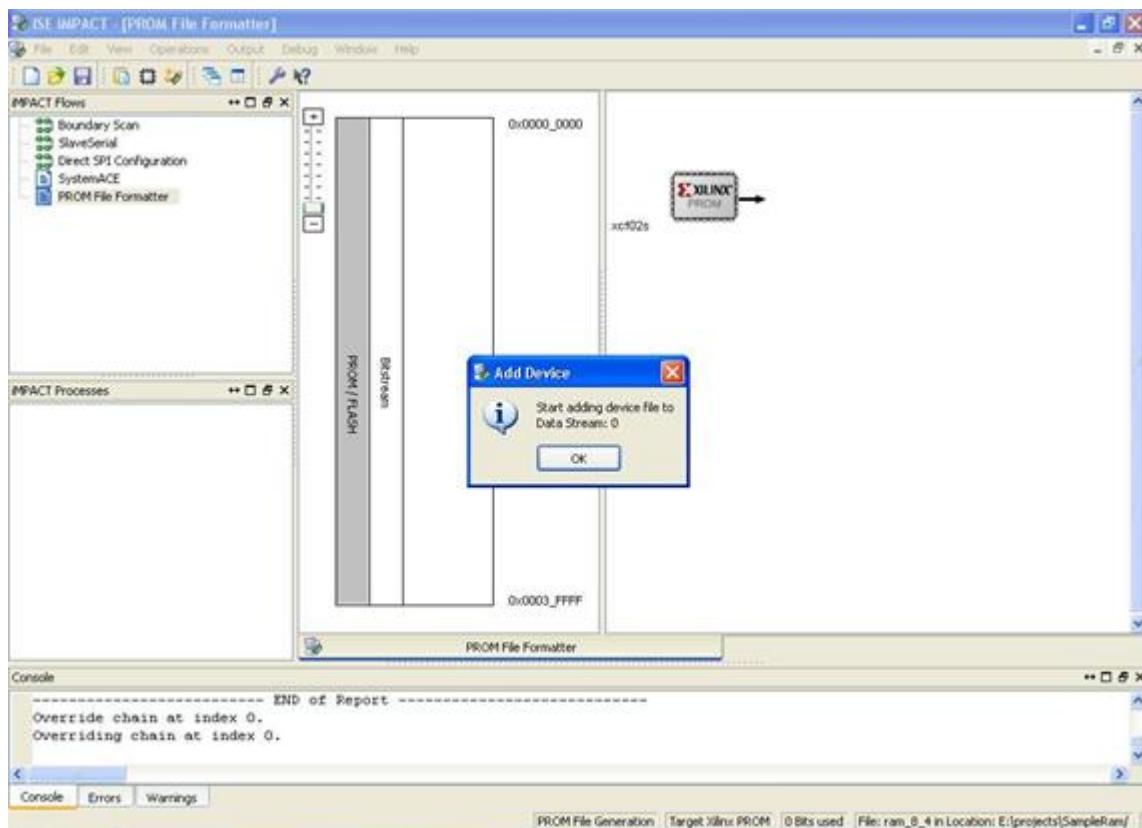
شکل ۳۲ گام اول ایجاد فایل برنامه ریزی



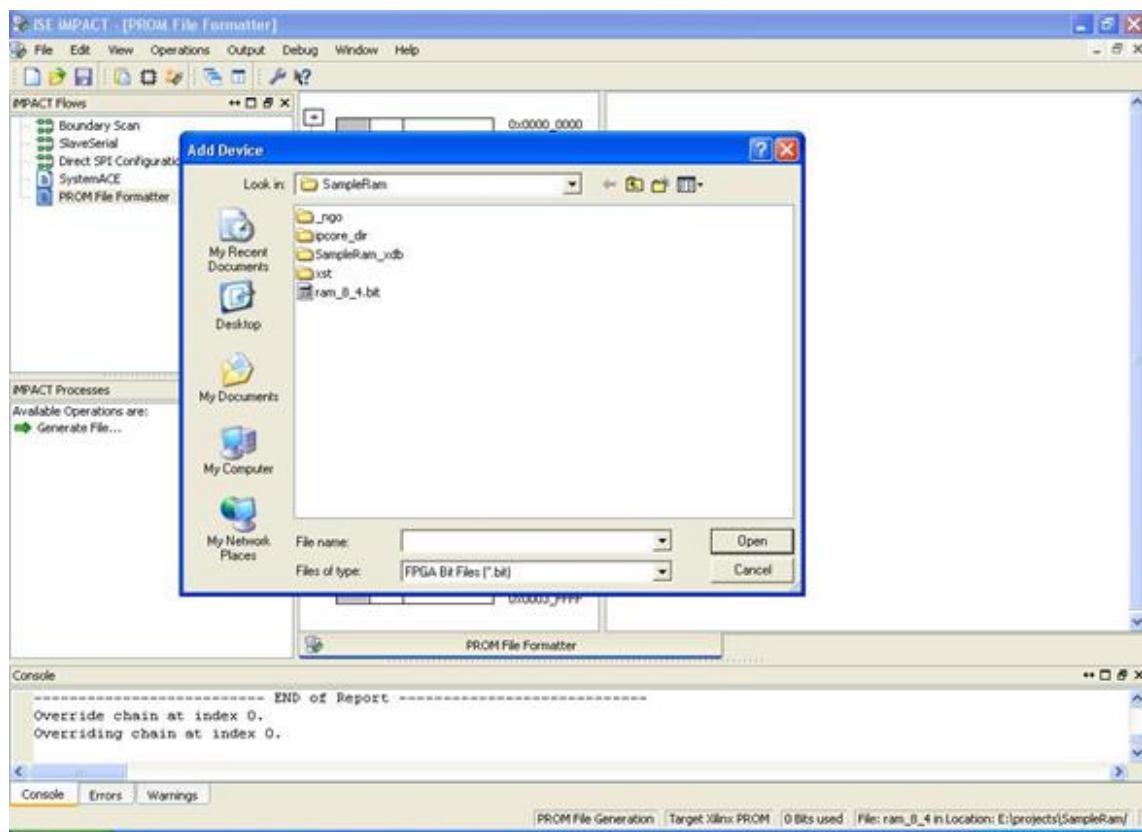
شکل ۳۳ گام دوم ایجاد فایل برنامه‌ریزی



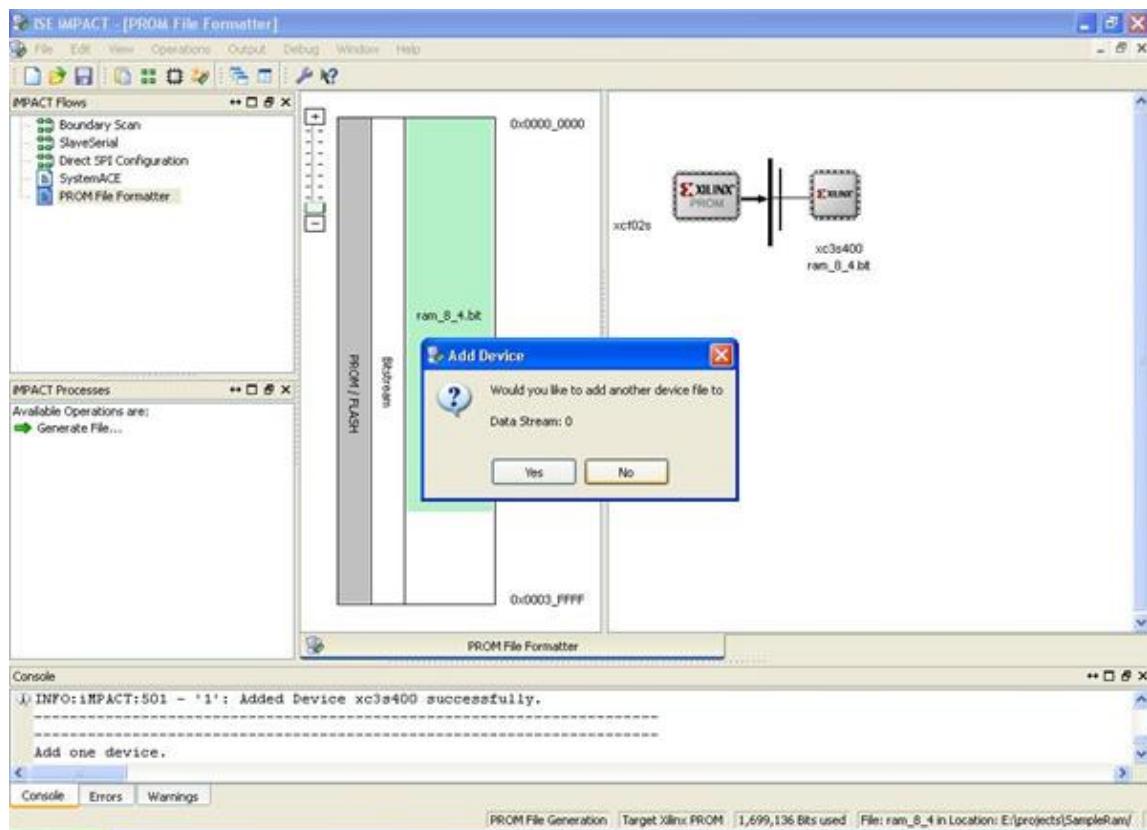
شکل ۳۴ گام سوم ایجاد فایل برنامه‌ریزی



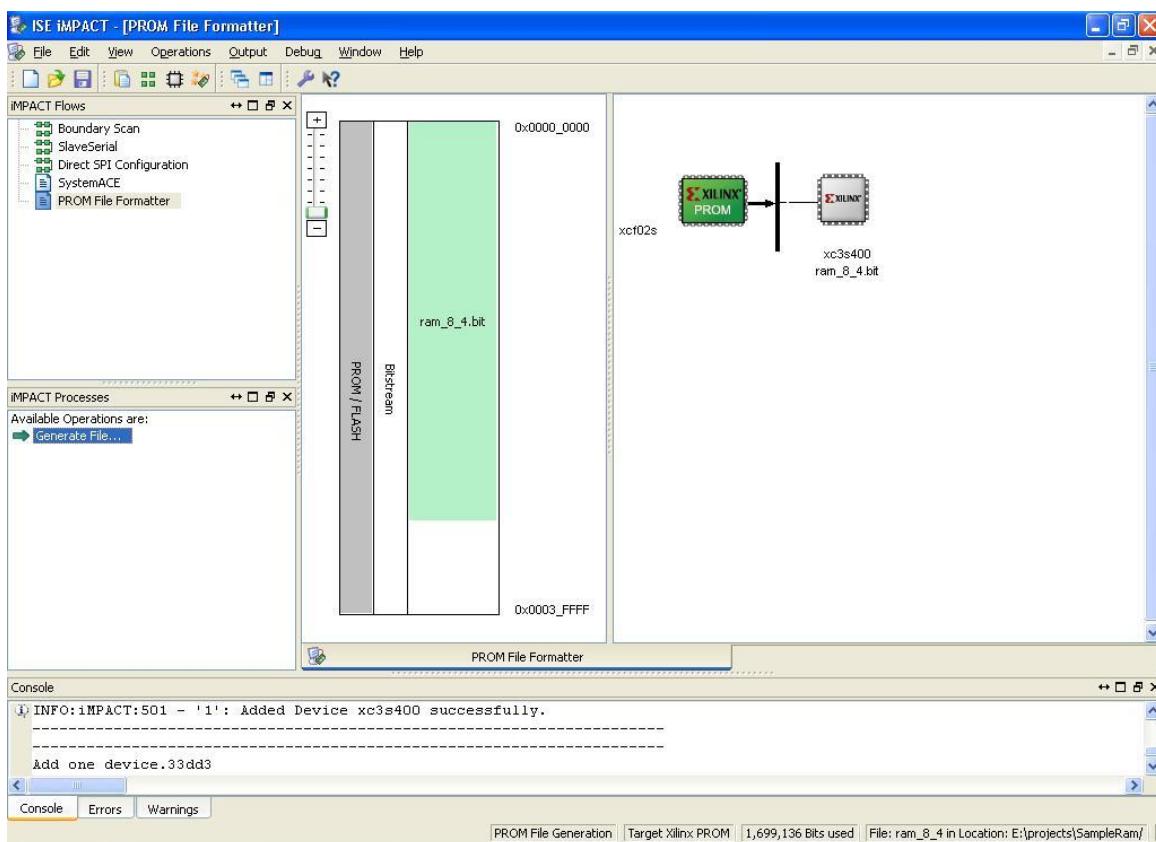
شکل ۳۵ جستجوی RAM های روی بورد توسط iMPACT



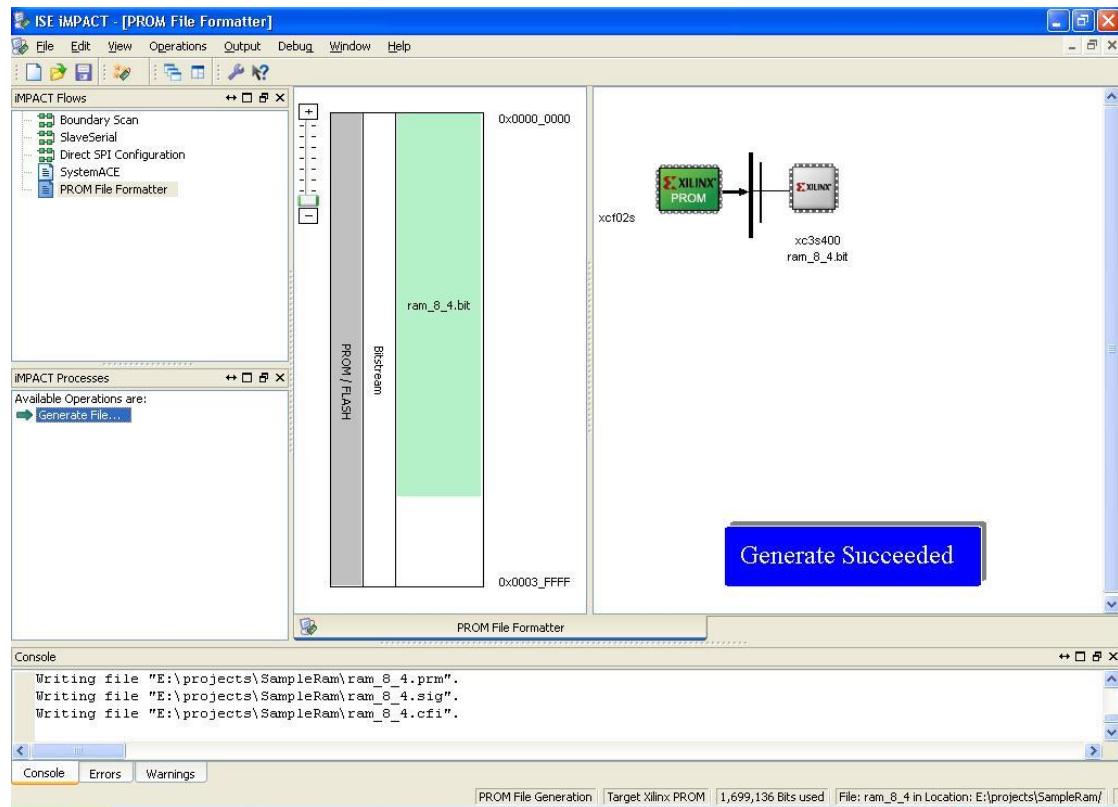
شکل ۳۶ انتخاب فایل برنامه ریزی FPGA برای برنامه ریزی RAM



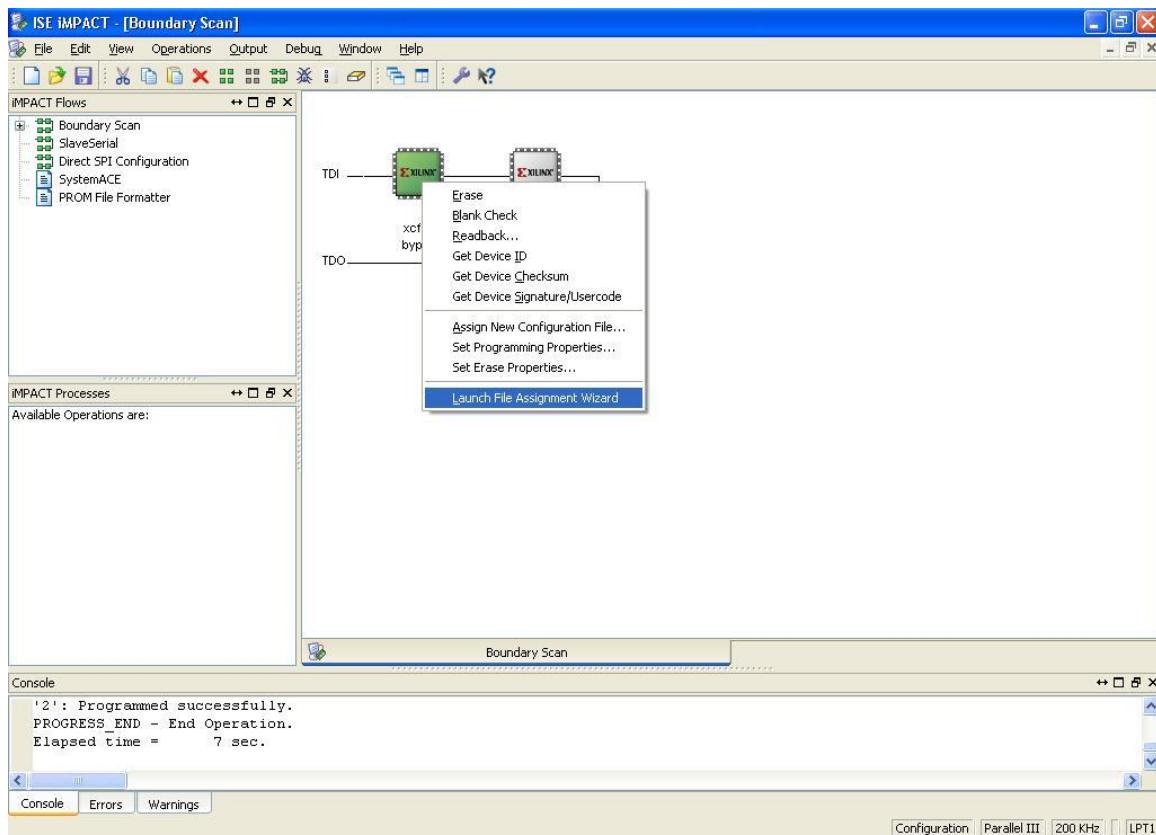
شکل ۳۷ هشدار مبنی بر افزودن فایل جدید



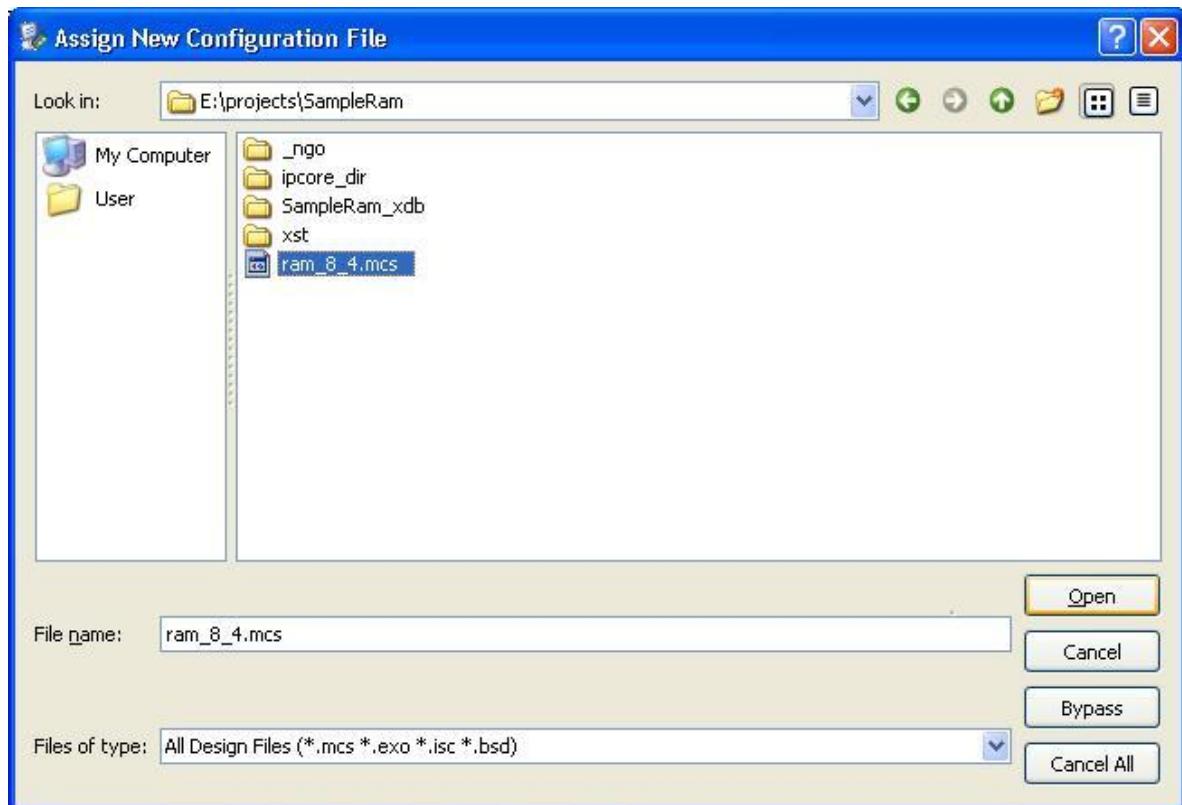
شکل ۳۸ نتیجه انتخاب فایل



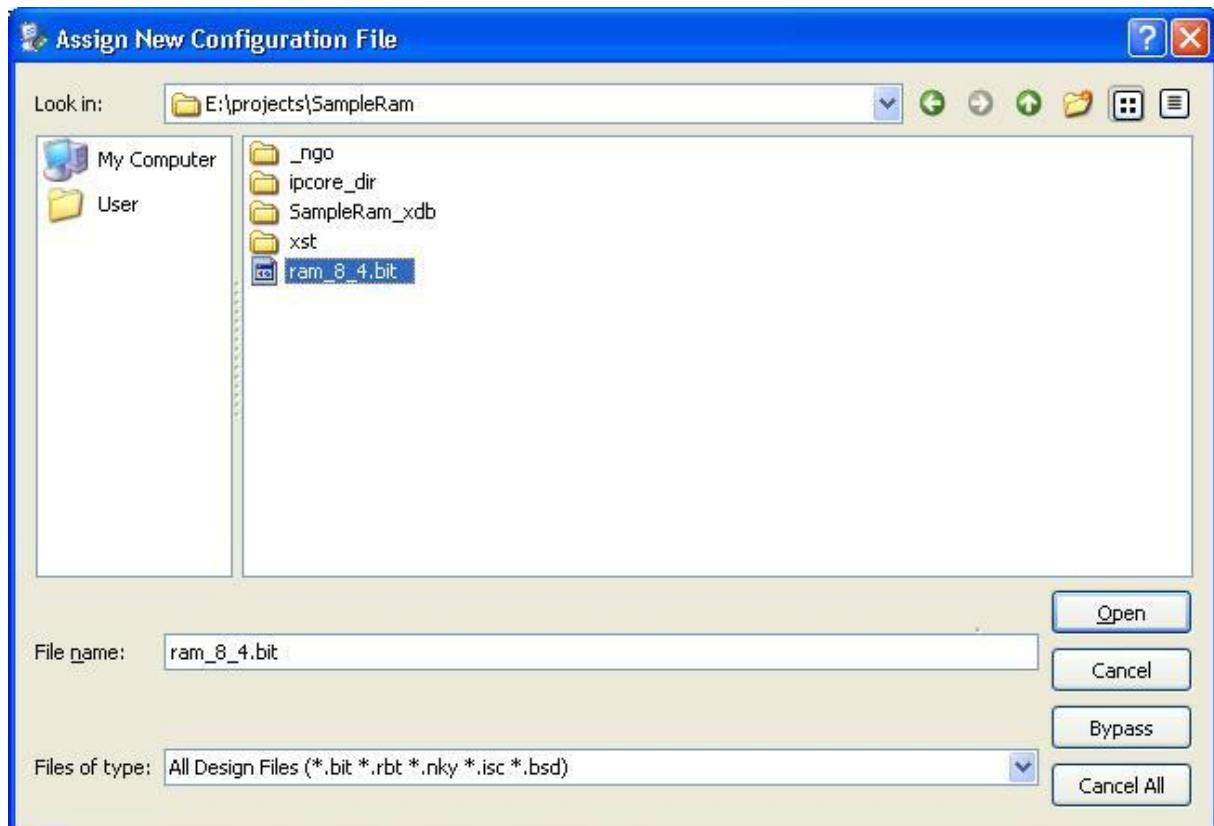
شکل ۳۹ ایجاد موفقیت آمیز فایل برنامه ریزی RAM



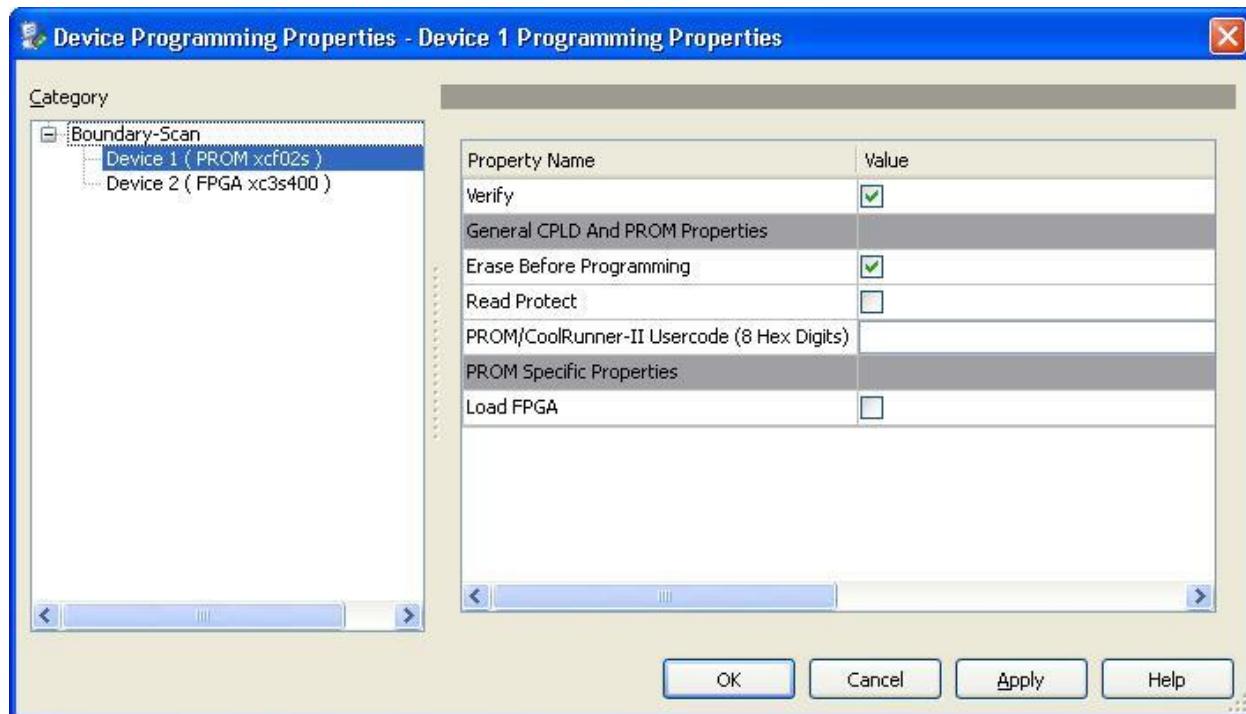
شکل ۴۰ تعیین فایل برنامه ریزی ram



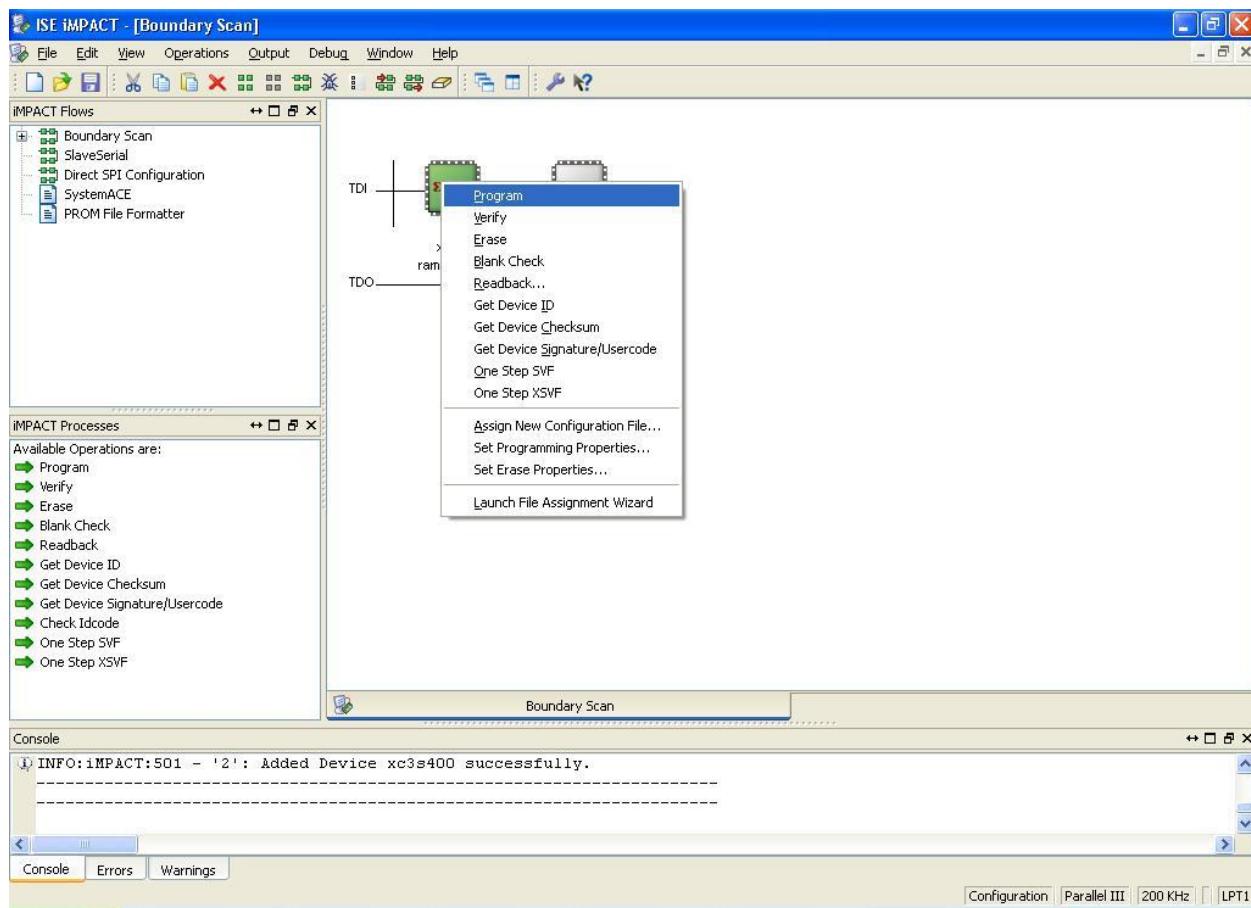
شکل ۴۱ انتخاب فایل برنامه ریزی RAM



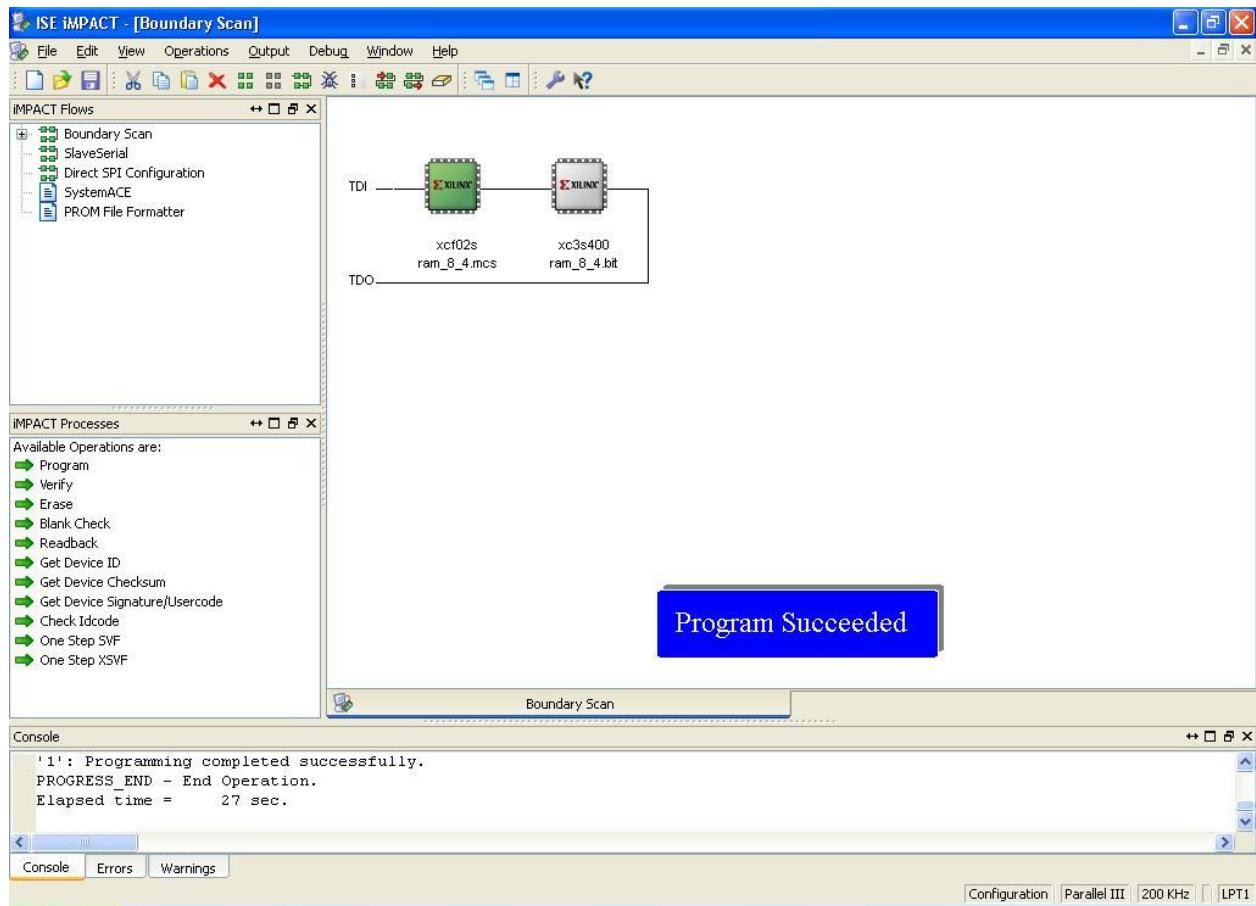
شکل ۴۲ انتخاب فایل برنامه ریزی FPGA



شکل ۴۳ ویژگی‌های برنامه‌ریزی دو ابزار



شکل ۴۴ برنامه‌ریزی RAM



شکل ۴۵ نتیجه موفقیت‌آمیز برنامه‌ریزی RAM