

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
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 مدار ترتیبی 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
۲۶	پیاده‌سازی ارسال/دریافت نامهمگام در گذرگاه‌های یکطرفه و دوطرفه	ورودی/خروجی	۱۰
۲۹	پیاده‌سازی مکانیزم‌های داوری گذرگاه	ورودی/خروجی	۱۱
۳۲	پیاده‌سازی خط لوله و بررسی تسريع آن	خط لوله	۱۲
۳۶	پیاده‌سازی عملیات جمع یا تفریق واحد ممیز شناور	محاسبات ممیز شناور	۱۳
۳۷	اندازه‌گیری پارامترهای IPC, CPI, MIPS و DMIPS یک پردازنده نوعی	ارزیابی کارآیی	۱۴
۳۹	طراحی واحد کنترل سیم بندی شده (Hard wired Control Unit)	طراحی واحد کنترل	۱۵
۴۰	طراحی واحد کنترل ریزبرنامه‌ریزی شده (Microprogrammed Control Unit)	طراحی واحد کنترل	۱۶
۴۱	طراحی سلسله مراتب حافظه	حافظه‌ها	۱۷

فهرست ضمایم

صفحه	عنوان	شماره ضمیمه
۲۵	توصیف سیستم با VHDL	۱
۲۸	شبیه‌سازی با استفاده از ModelSim	۲
۳۷	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:

مقدار دهنده به 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);

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;

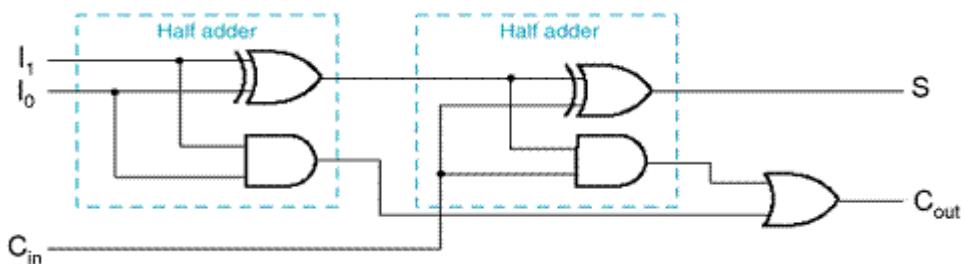
entity decoder_tb is
end entity decoder_tb;

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:

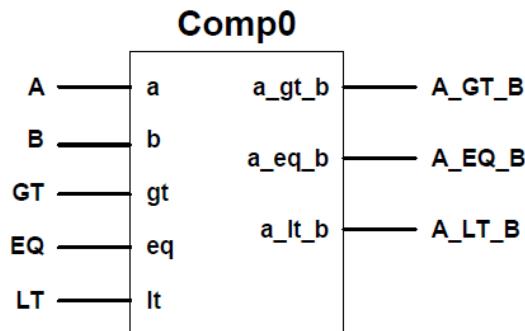
در این آزمایش هدف آشنایی با نحوه عملکرد و پیاده‌سازی هر یک از مدارهای پایه در سطح تجزیه و تحلیل است.

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 و تحلیل گیت انجام 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);

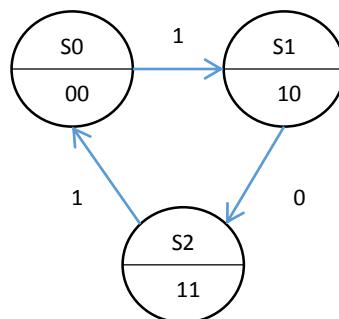
حال برای استفاده از Type تعريف شده می‌توان از آن Type یک سیگنال یا متغیر تعريف کرد. سپس از دستورالعمل Case روی Signal st : state; Process (...) Case st is When A => When B =>

```
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 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 تجزیه 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

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:

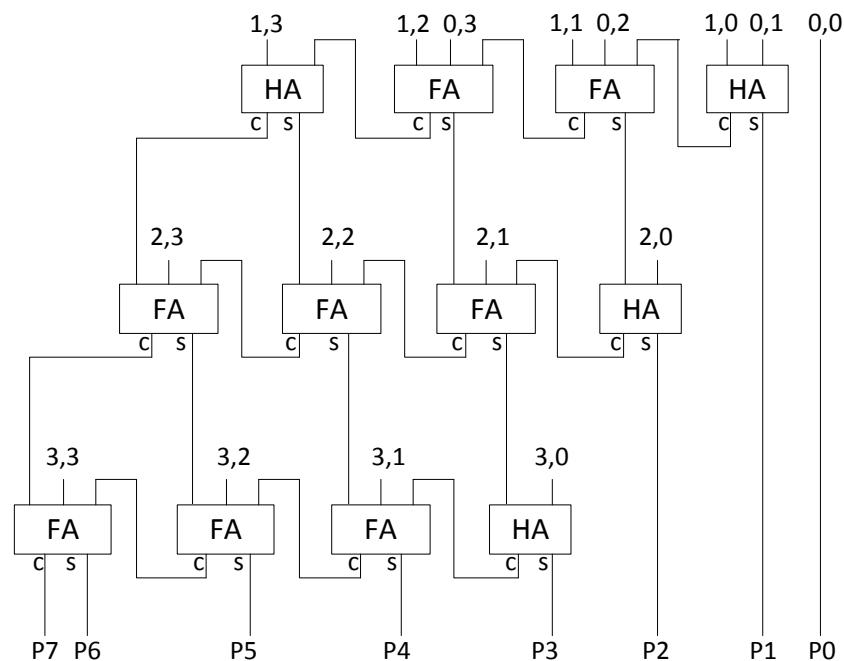
مطالعه مواردی که در بخش مراجع آزمایش به آن اشاره شده است و انجام برخی شبیه‌سازی‌های مورد نظر.

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:

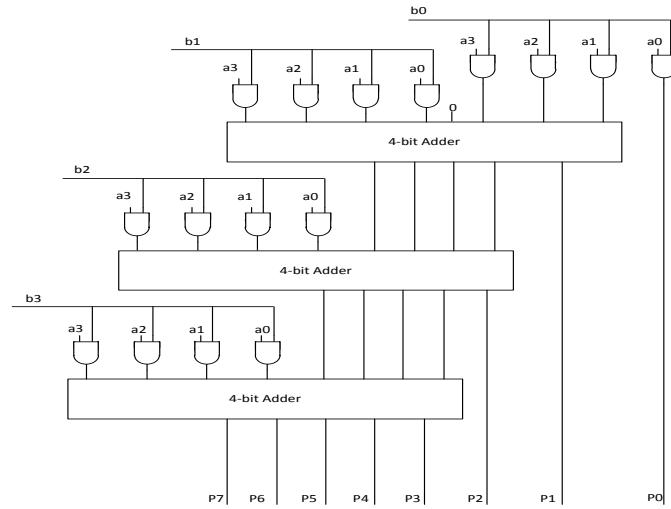
در این ضرب کننده مثل ضرب دهدۀی (decimal)، در هر مرحله یک بیت از مضروب (multiplier) در تمام بیت‌های مضروب فیه (multiplicand) ضرب شده و با یک بیت شیفت با خروجی مرحله قبل جمع می‌شود. نمودار این ضرب کننده در شکل ۱ نشان داده شده است.



شکل ۱- نمودار بلوکی ضرب کننده معمولی

ب) ضرب کننده آرایه‌ای:

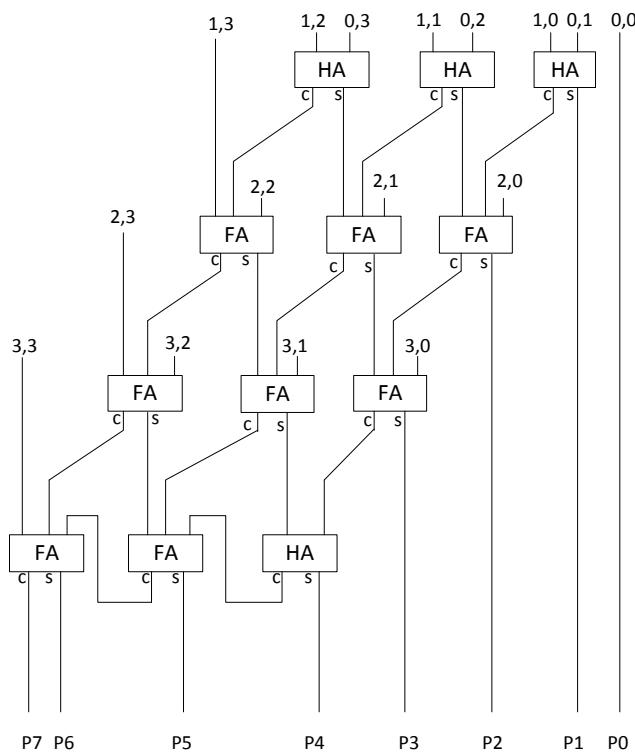
در این ضرب کننده از واحدهای جمع کننده ۴ بیتی استفاده شده است. نمودار این ضرب کننده در شکل ۲ نشان داده شده است.



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.

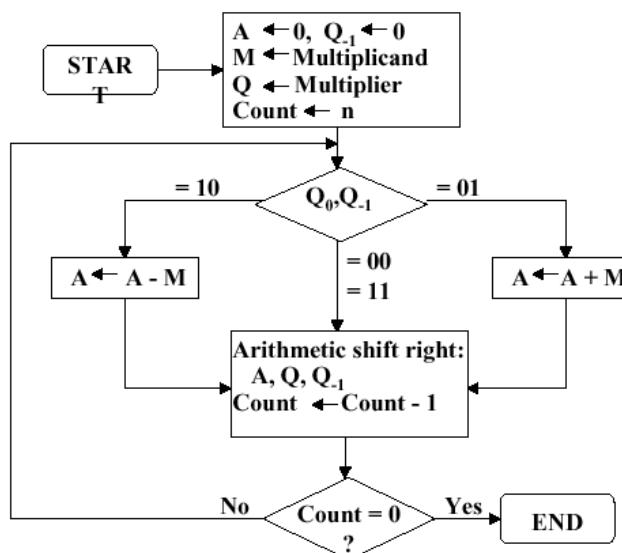


شکل ۳-نمودار بلوکی ضرب کننده

(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 استفاده از این مدارات به عنوان مدار تاخیرساز.

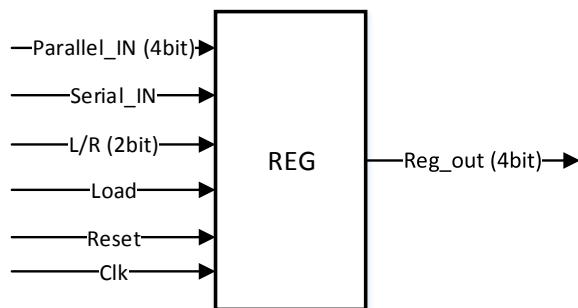
شرح آزمایش:

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. طراحی شماتیک مدار ثبات در شکل ۱ نمایش داده شده است. قابلیت‌های این بلوک به شرح زیر می‌باشد:

- 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 دو بیتی است که برای انتخاب بین حالت شیفت است.
- ورودی Load تک بیتی است که در صورت یک بودن، ورودی چهار بیتی Parallel_IN همزمان با لبه بالارونده کلک در ثبات بارگذاری می‌شود.
- در صورت صفر بودن ورودی Load در هر کلک ساعت یک بیت از Serial_IN به ثبات وارد می‌شود.

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 روزی 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 ۱۷-۴ 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.

```
#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 تهییه Testbench برای آن و نشان دادن درستی عملکرد آن‌ها.
- سنتز و پیاده‌سازی مدارها بر روی بورد FPGA و اثبات درستی عملکرد مدارها.

مراجع مطالعه/ خصایق:

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

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

هدف از این آزمایش، آشنایی با انواع حافظه‌ها، طراحی و پیاده‌سازی برخی از آن‌ها است.

آمادگی پیش از آزمایش:

مطالعه و تشریح کتبی چگونگی عملکرد حافظه‌های فوق و آشنایی با نحوه پیاده‌سازی آن‌ها.

شرح آزمایش:

یکی از بخش‌های اصلی در مدارهای دیجیتال، بخش حافظه است. مدل‌سازی نامناسب حافظه علاوه بر اینکه ممکن است زمان شیوه‌سازی را طولانی کند، در هنگام سنتز نیز در صورت توصیف نادرست باعث استفاده غیرمعمول از منابع سخت‌افزاری می‌شود. در این آزمایش هدف اصلی آشنایی با نحوه مدل کردن انواع حافظه‌ها با زبان توصیف سخت‌افزار است.

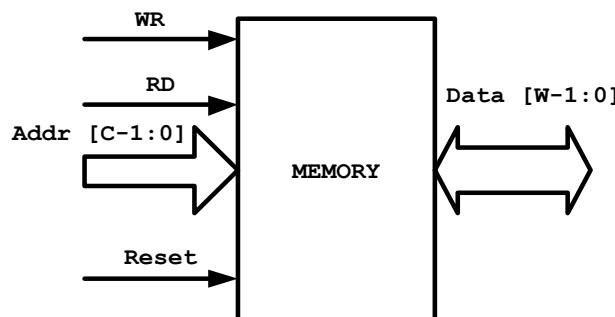
الف) طراحی حافظه RAM:

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

- عرض حافظه W

- تعداد خانه‌های حافظه D

- عرض درگاه Address برابر با $\log_2(D)$ می‌باشد که فعلاً آن را هم در کد با پارامتر C نشان دهید.



شکل ۱- بلوگ دیاگرام یک واحد حافظه

در بلوک حافظه فوق:

درگاه Data از نوع ورودی/خروجی تعریف می‌شود:

- سیگنال‌های WR و RD حساس به لبه بالارونده عمل کرده بطوریکه هنگامی WR در لبه بالارونده است مقدار Data به خانه‌ای که توسط Addr مشخص شده، نوشته می‌شود. هنگامی که RD در لبه بالارونده است عمل خواندن از حافظه انجام شود.
- درگاه Addr نشان می‌دهد 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 می‌کند.

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.

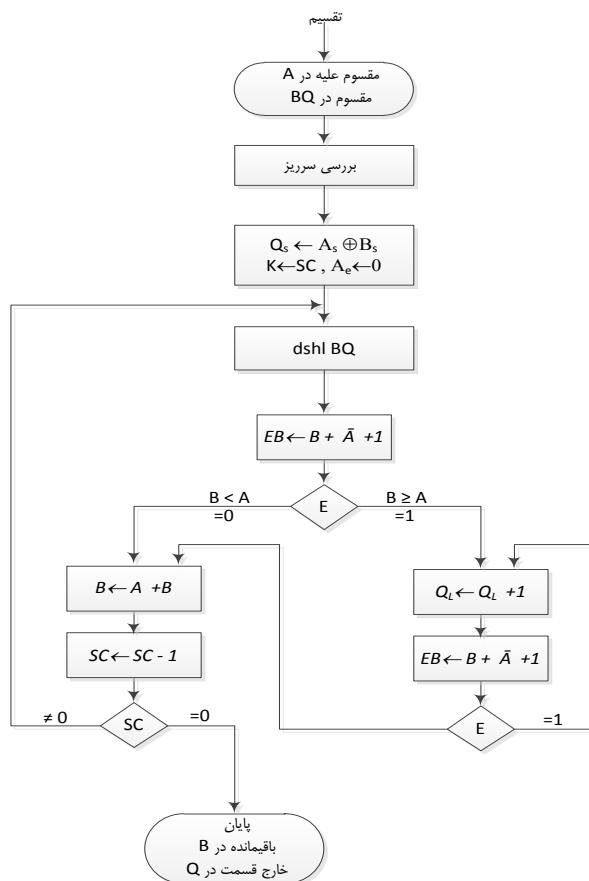
مراجع مطالعه/ خصایق:

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

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

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

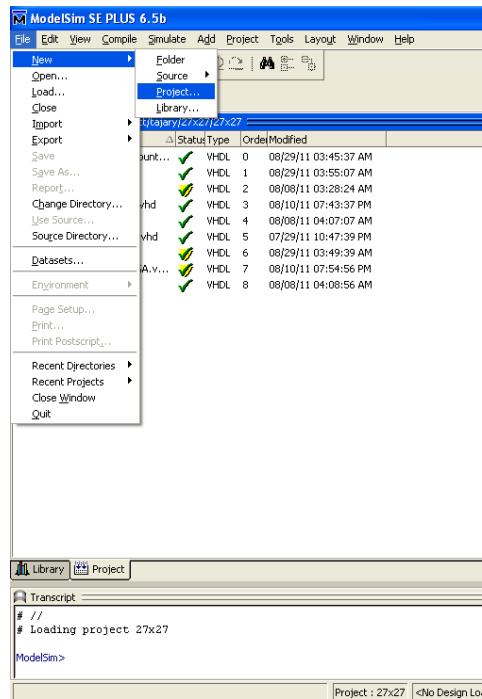
```
Port(
    A: in std_logic_vector(9 downto 0);
    C: out std_logic_vector (9 doento 0)
);
Signal b: std_logic_vector(9 downto 0);
```

مقداردهی به std_logic_vector

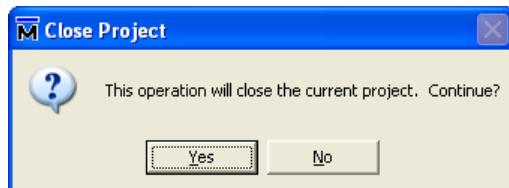
```
B(9 downto 1) <= B(8 downto 0); -- shift
B(0) <= '1';
B(2 downto 0) <= "110";
B <= A;
B(9 downto 8) <= A(2 downto 1);
B(4) <= B(5);
B <= (others => '0');
B(5 downto 0) <= A(9 downto 7) & "011";
```

ضمیمه ۲

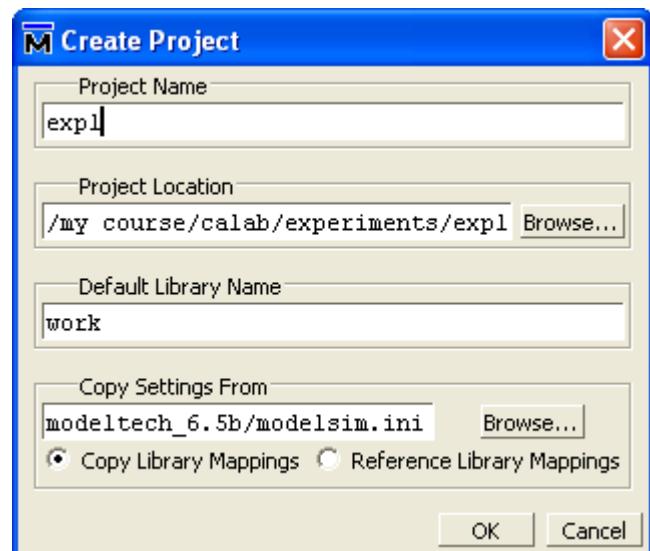
شبیه‌سازی با استفاده از
ModelSim



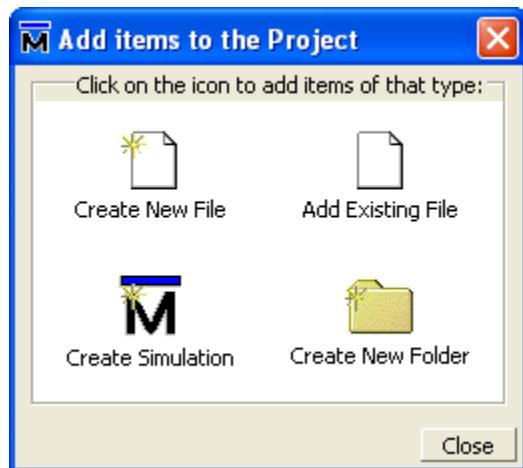
شکل ۱ ایجاد پروژه جدید



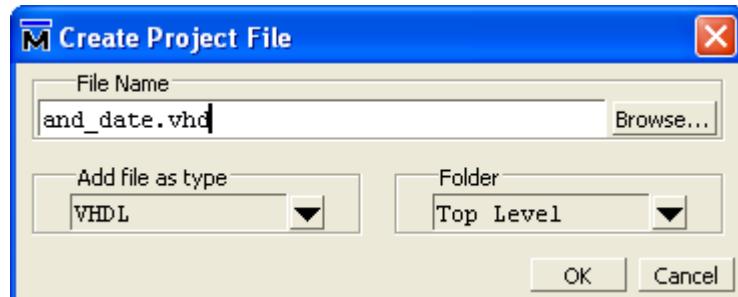
شکل ۲ پیغام هشدار بستن پروژه قبلی



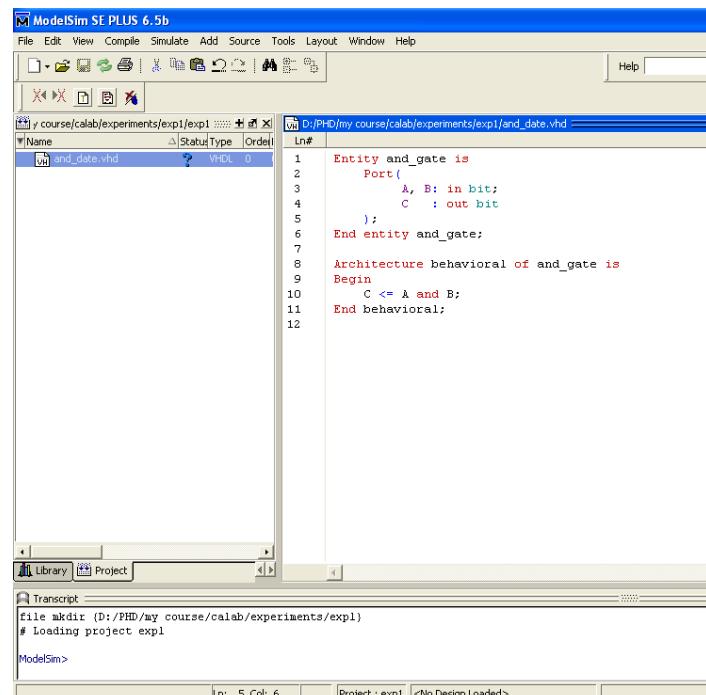
شکل ۳ تعیین نام و پوشش پروژه



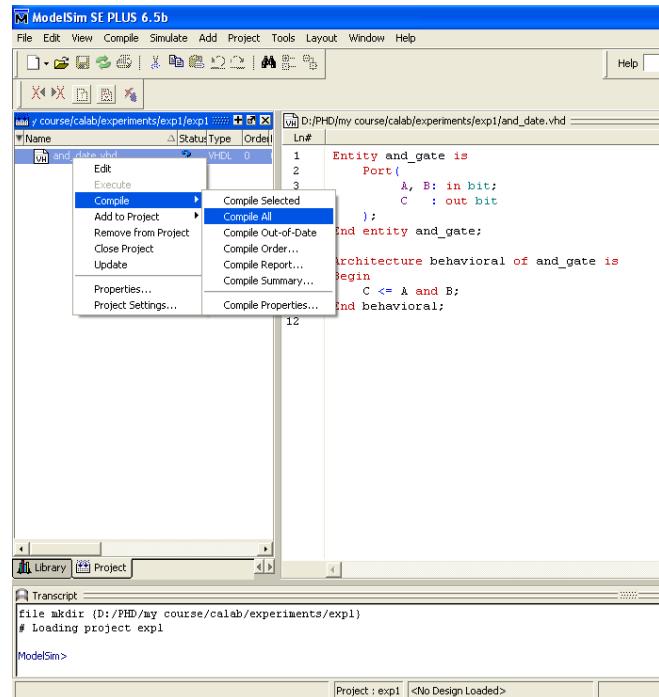
شکل ۴ افزودن فایل به پروژه



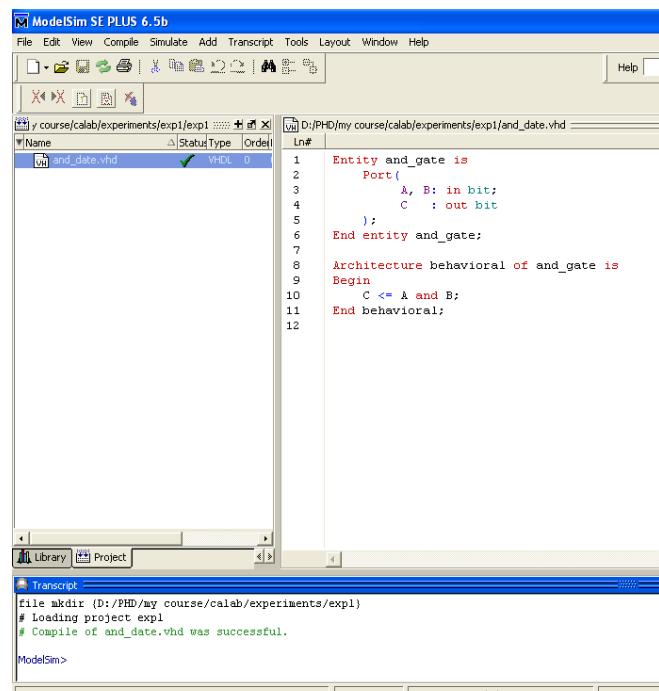
شکل ۵ تعیین نوع و نام فایل جدید



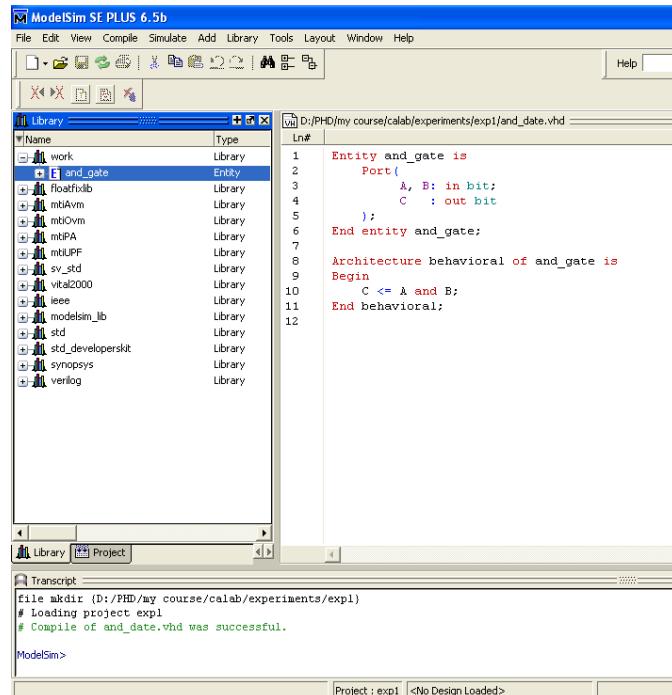
شکل ۶ نوشتن کد در ویرایشگر



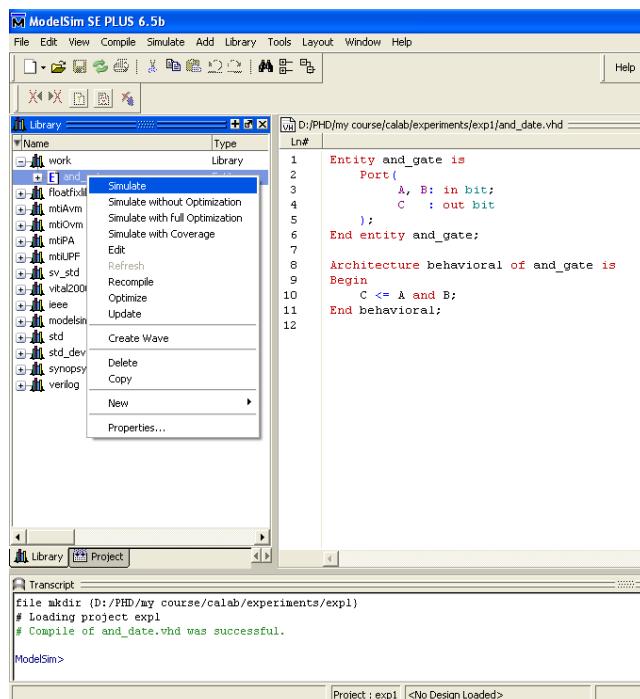
شکل ۷ منو کامپایل



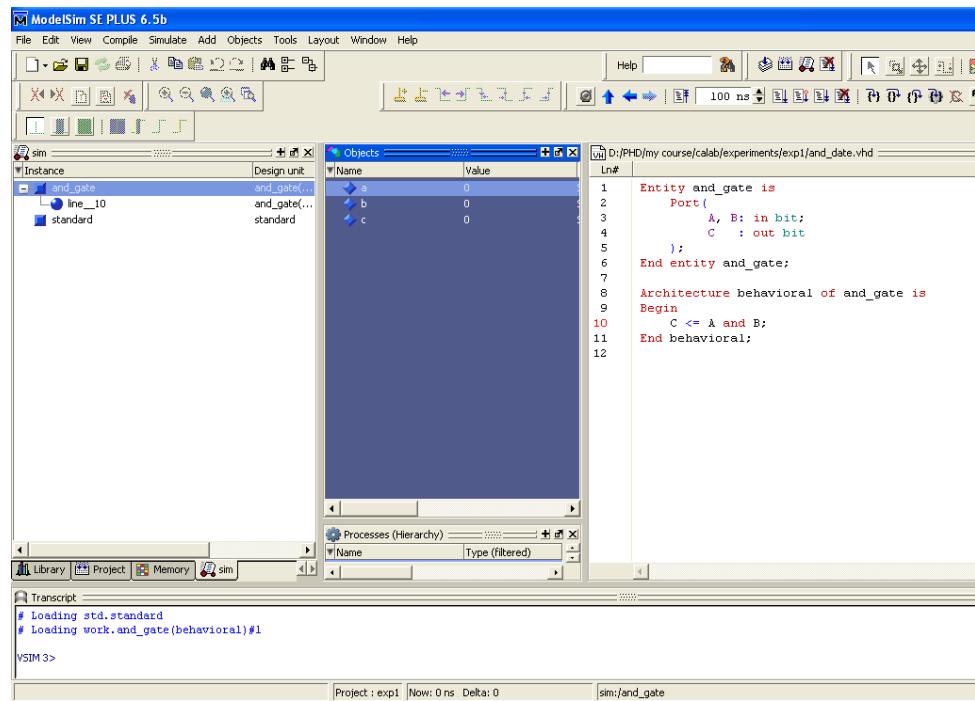
شکل ۸ نتیجه کامپایل



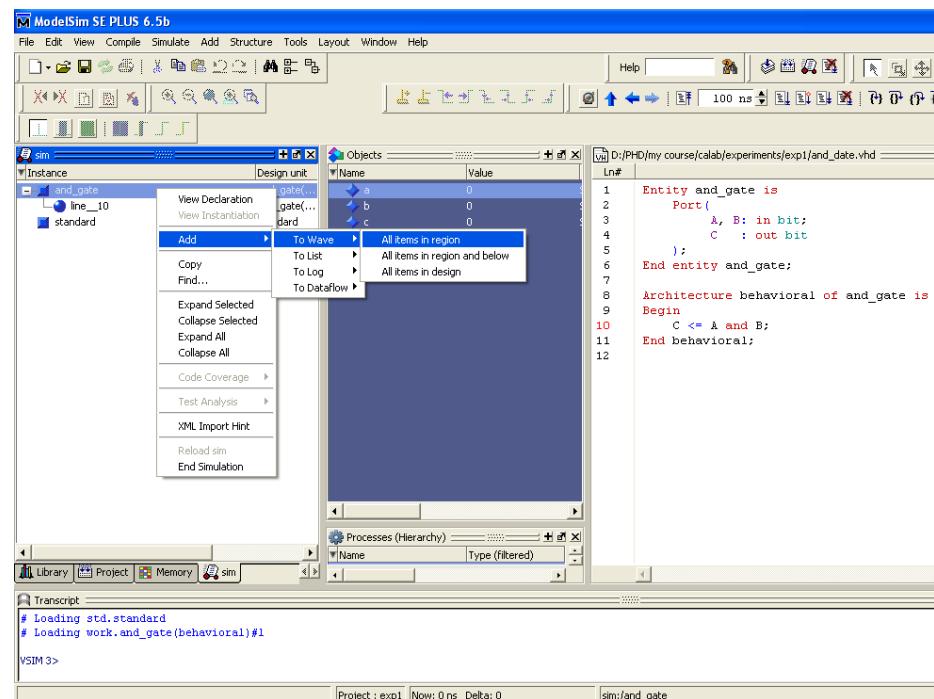
شکل ۹ بخش library برای شبیه‌سازی



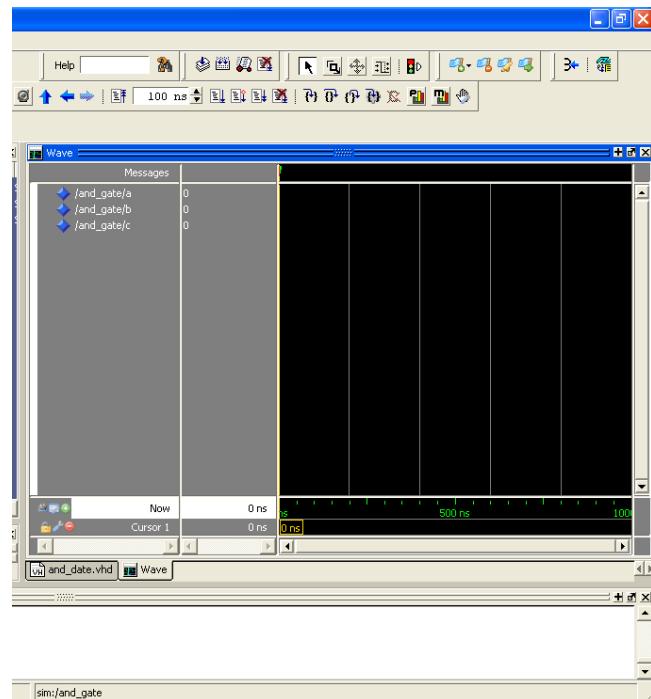
شکل ۱۰ منو شبیه‌سازی



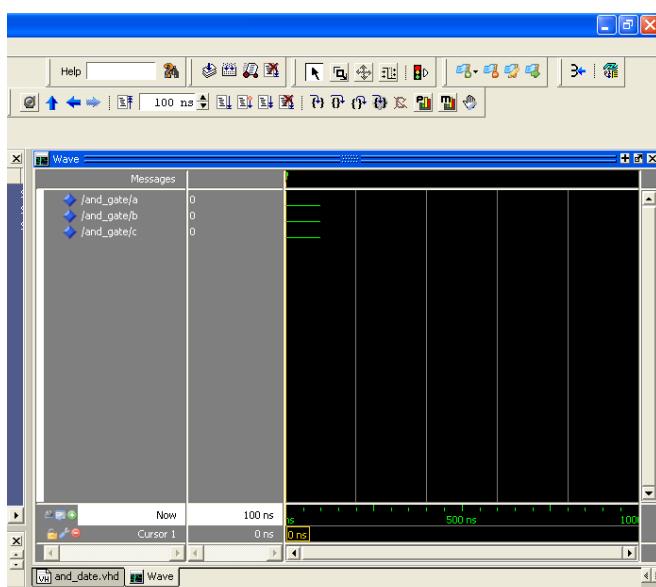
شکل ۱۱ نمای شروع شبیه‌سازی



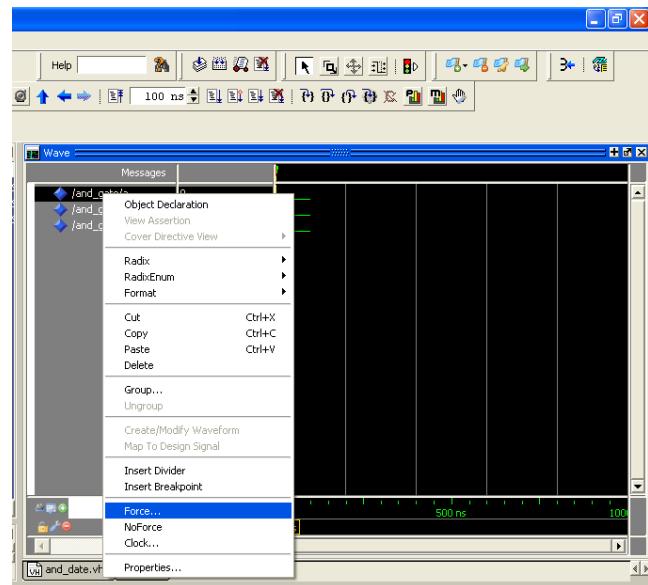
شکل ۱۲ افزودن سیگنال به پنجره wave



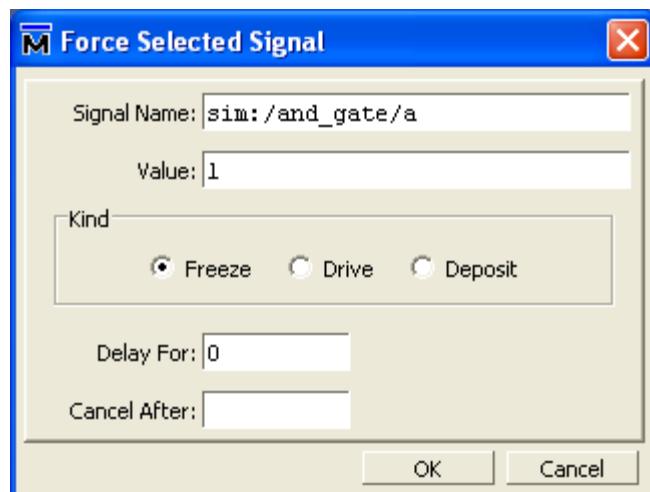
شکل ۱۳ سیگنال‌ها در پنجره wave



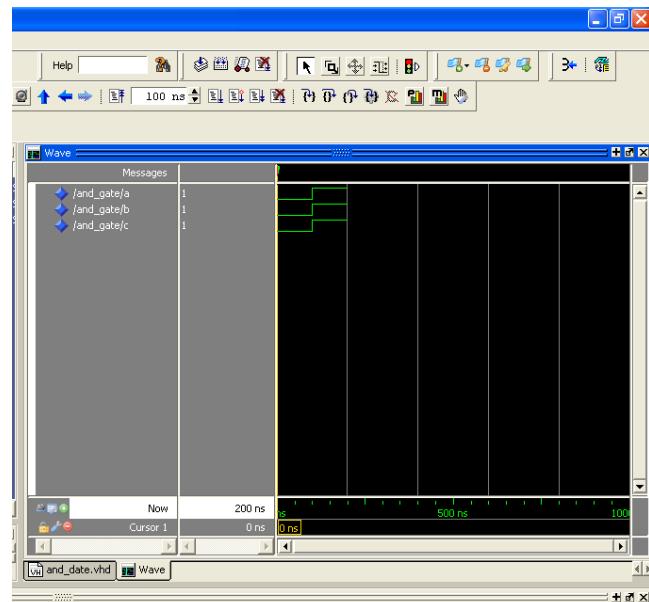
شکل ۱۴ اجرای شبیه‌سازی



شکل ۱۵ منوی force کردن سیگنال



شکل ۱۶ تعیین مقدار سیگنال در پنجره force

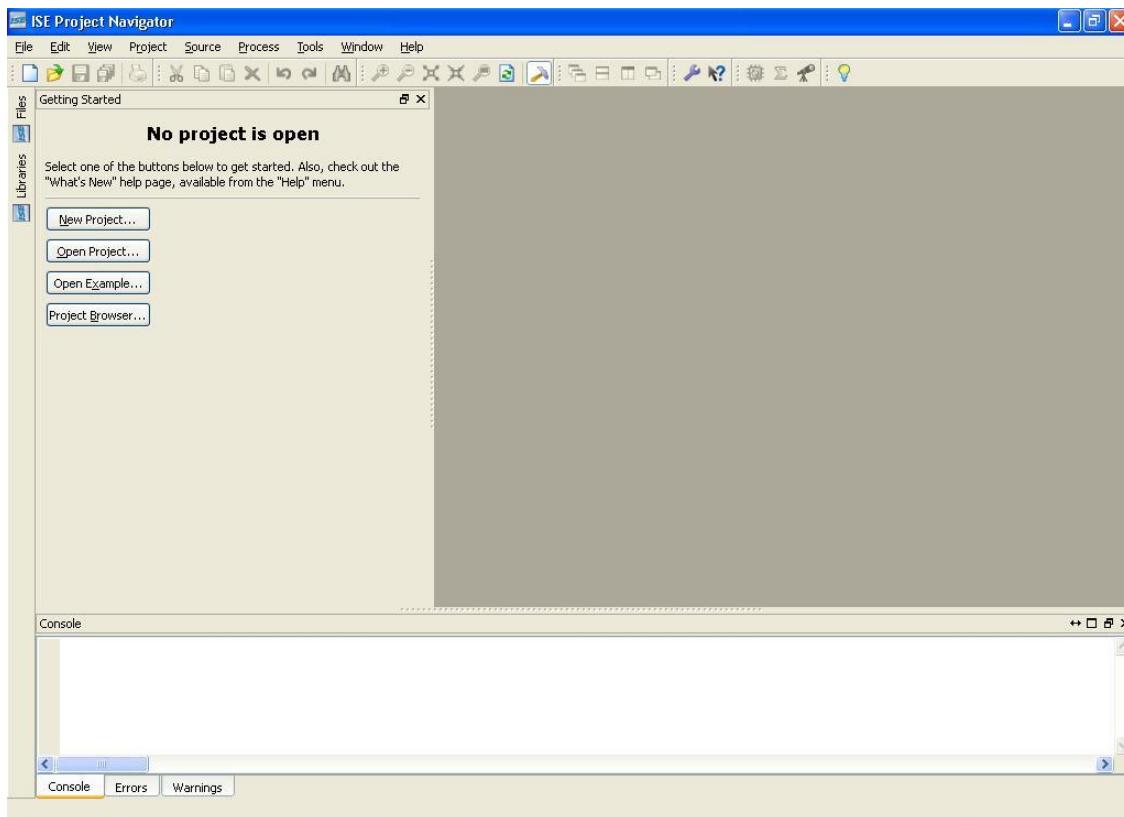


شکل ۱۷ اجرای شبیه‌سازی با مقادیر جدید

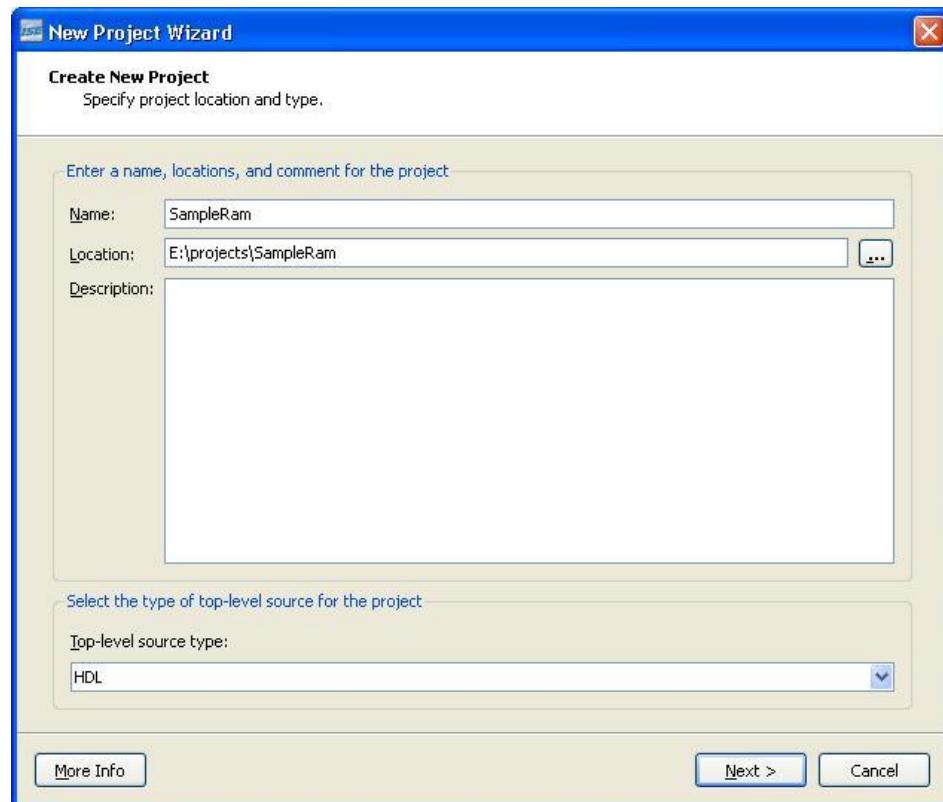
ضمیمه ۳

نحوه کارکردن با بورد FPGA

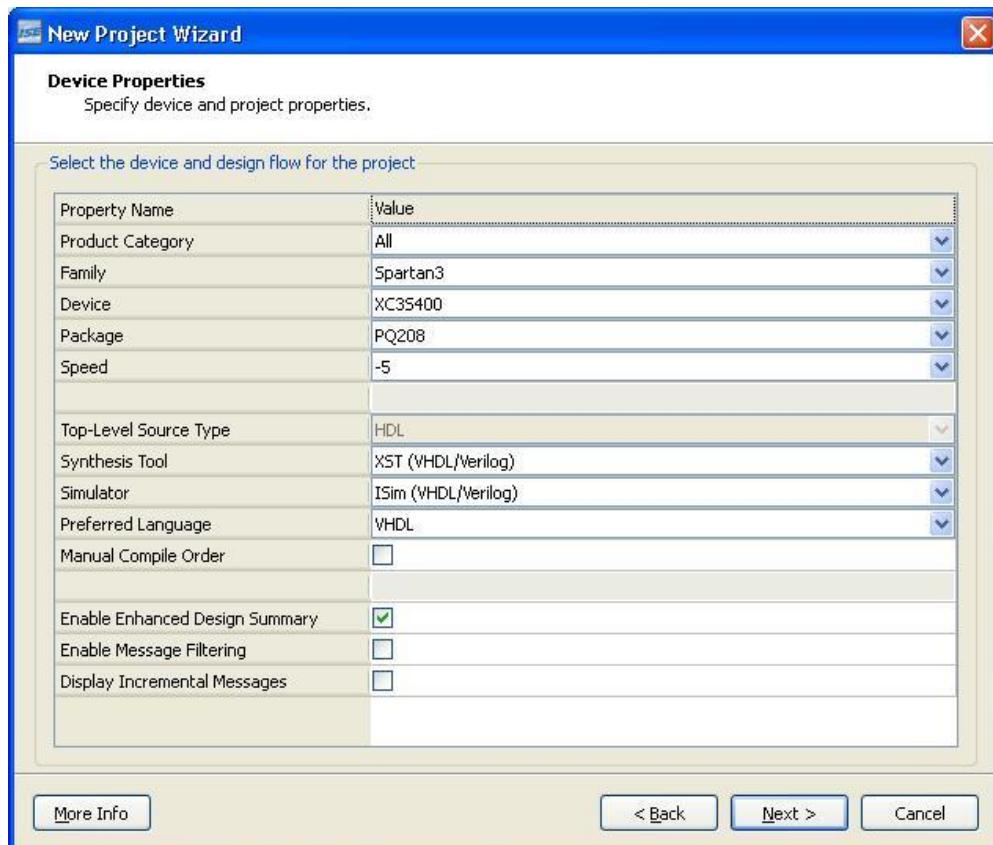
فاز اول: برنامه‌ریزی FPGA



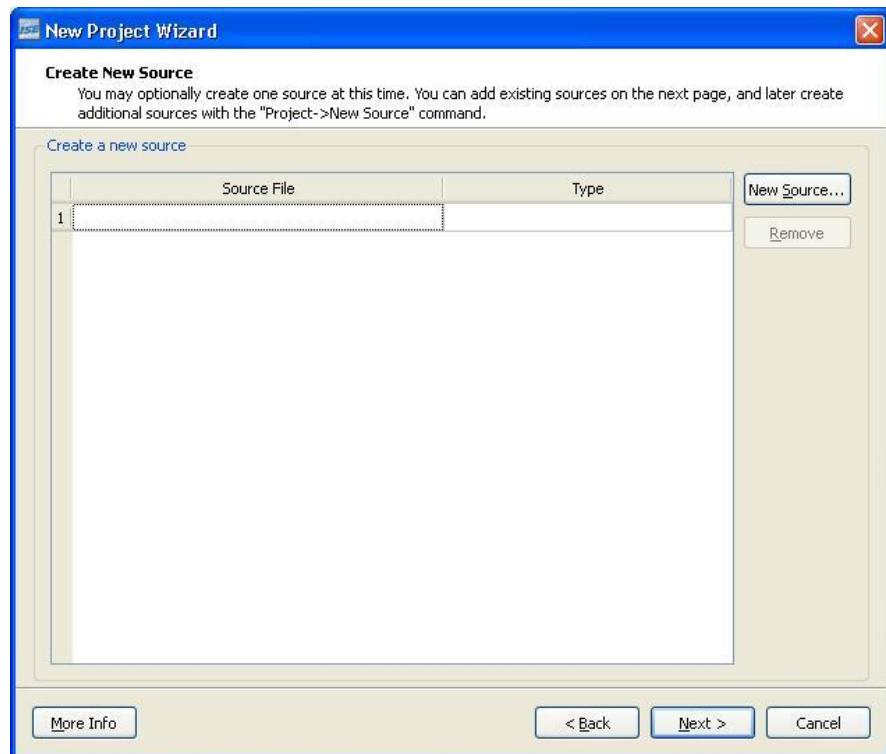
شکل ۱ صفحه شروع کار با ابزار ISE



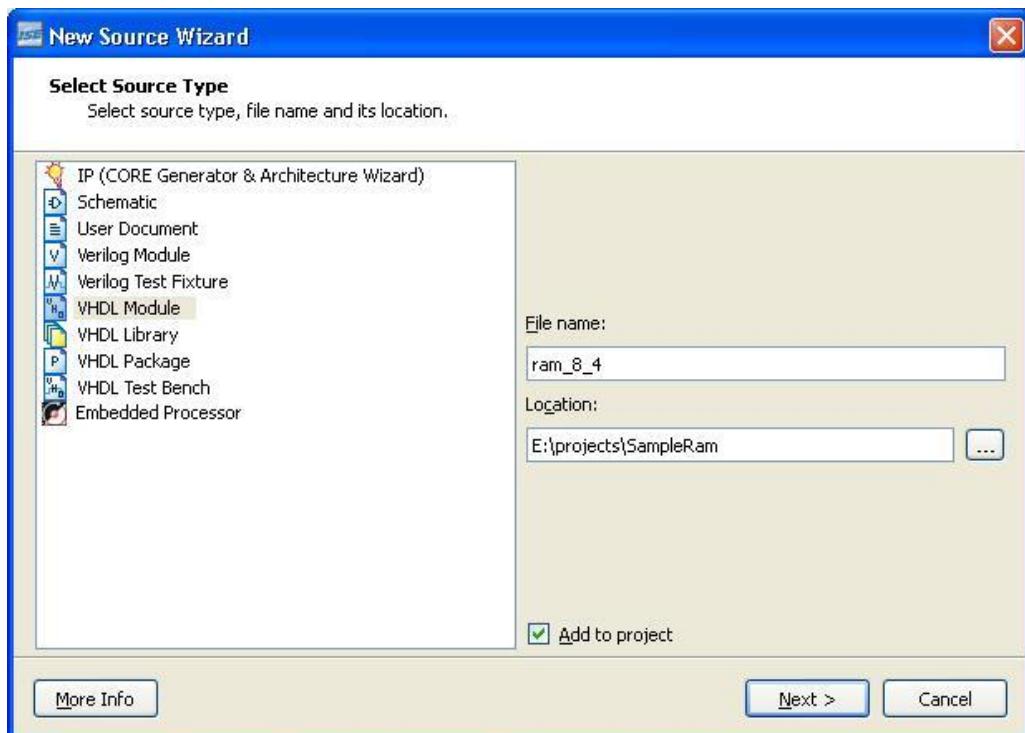
شکل ۲ ایجاد یک پروژه جدید



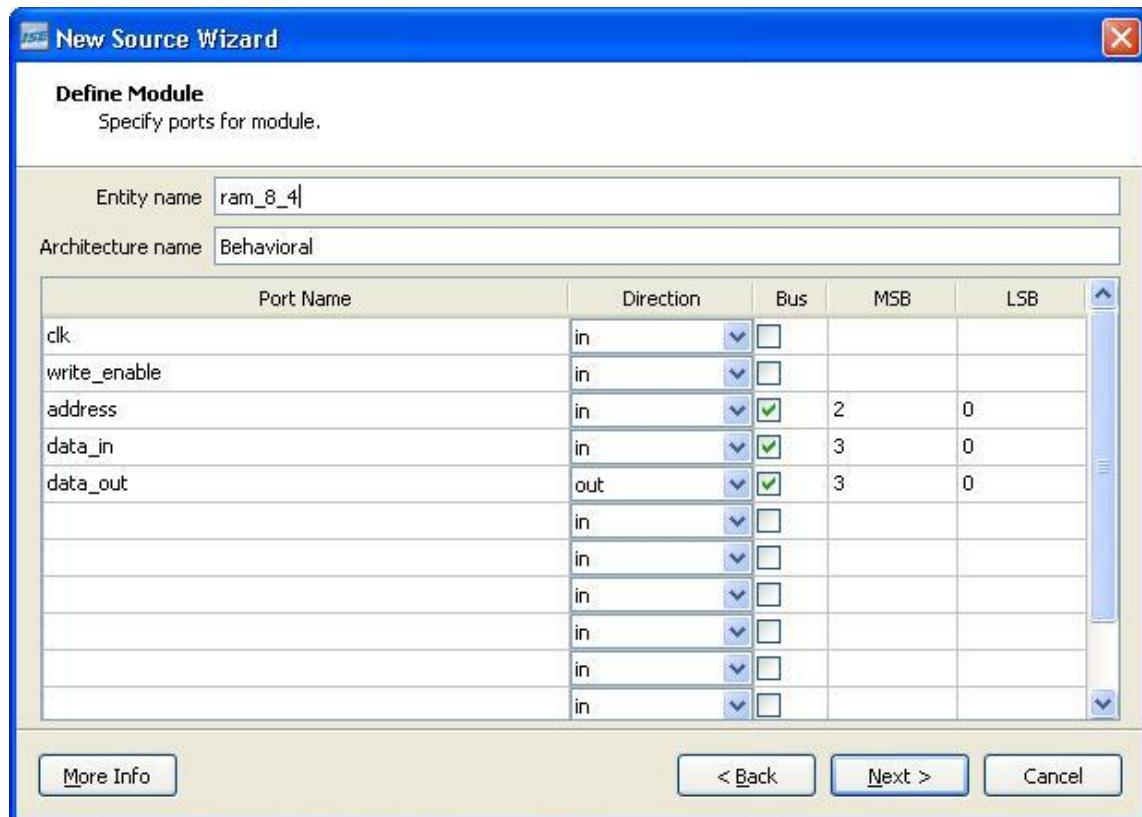
شکل ۳ تعیین ویژگی های FPGA



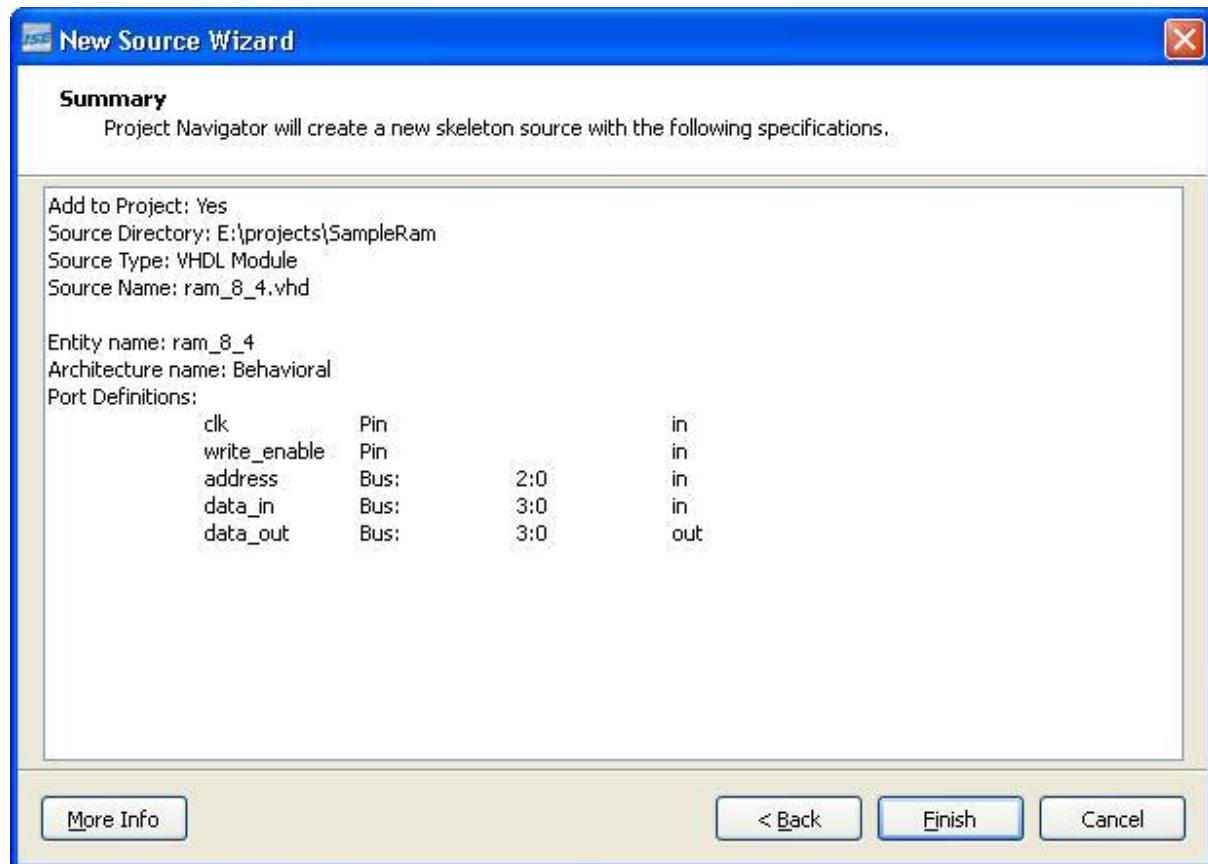
شکل ۴ ایجاد یک فایل برای entity



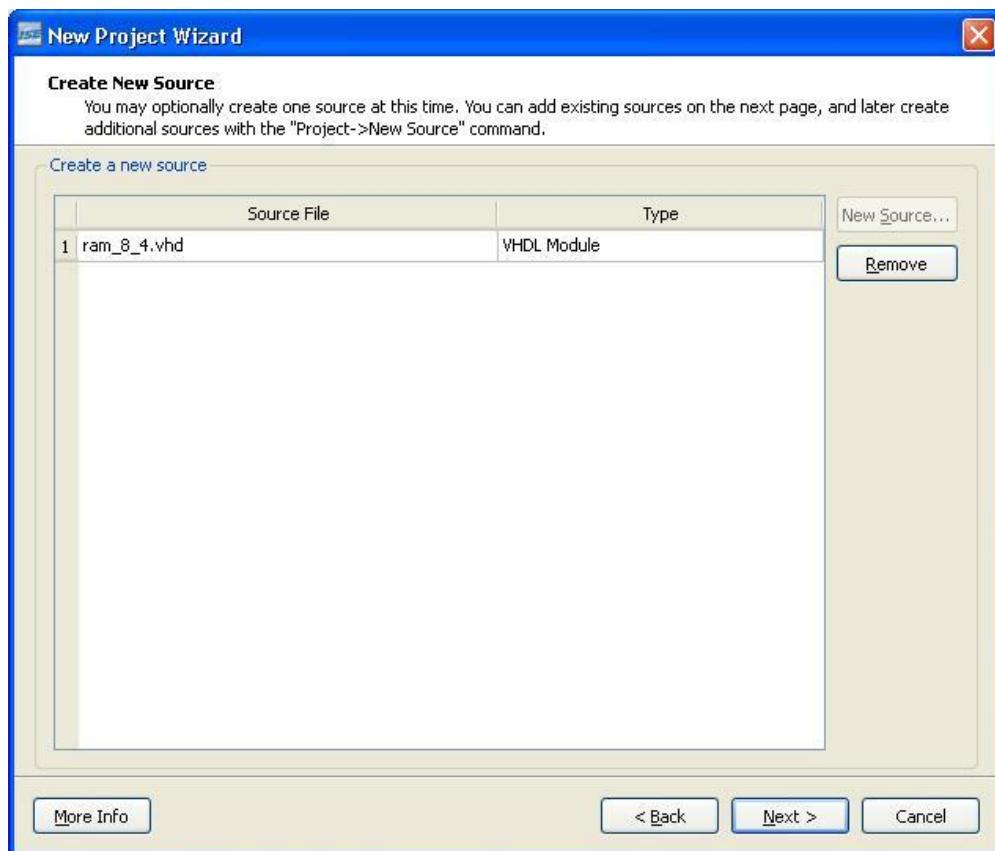
شکل ۵ تعیین نوع و نام فایل



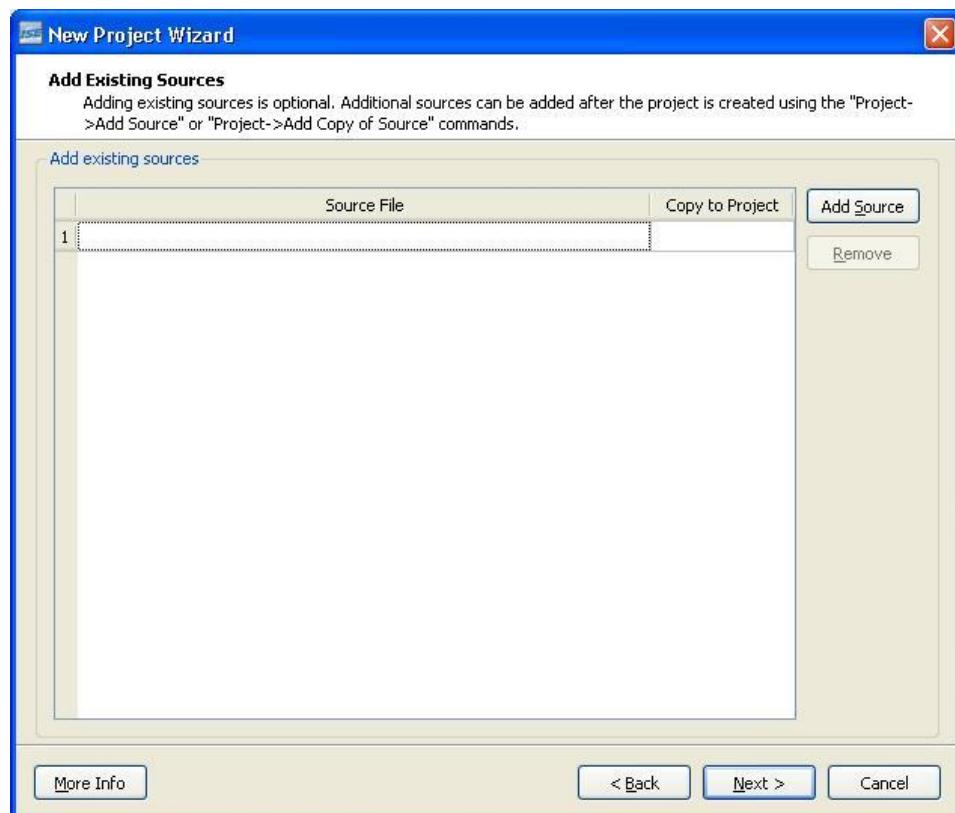
شکل ۶ تعیین پورت های ورودی و خروجی



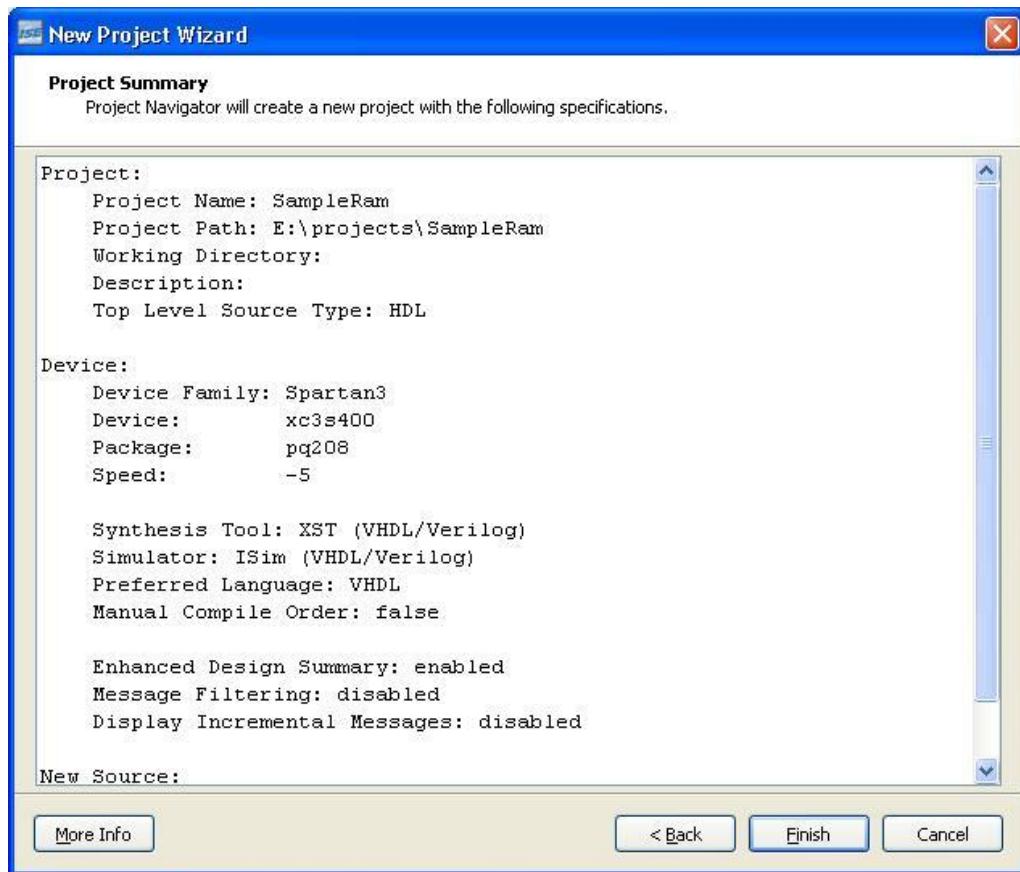
شکل ۷ خلاصه از ویژگی های فایل ایجاد شده



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

```

28 --use UNISIM.VComponents.all;
29
30 entity ram_8_4 is
31     Port ( cik : in STD_LOGIC;
32             write_enable : in STD_LOGIC;
33             address : in STD_LOGIC_VECTOR (2 downto 0);
34             data_in : in STD_LOGIC_VECTOR (3 downto 0);
35             data_out : out STD_LOGIC_VECTOR (3 downto 0));
36 end ram_8_4;
37
38 architecture Behavioral of ram_8_4 is
39     type ram is array (0 to 7) of std_logic_vector(3 downto 0) ;
40     signal mem_1 : ram;
41 begin
42
43     process (clk)
44     begin
45         if(rising_edge(cik)) then
46             if (write_enable = '1') then
47                 mem_1 (conv_integer(address)) <= data_in;
48                 data_out <= "ZZZZ";
49             else
50                 data_out<= mem_1 (conv_integer(address));
51             end if ;
52         end if ;
53     end process;
54
55 end Behavioral;
56
57

```

ISE Project Navigator - E:\projects\SampleRam\SampleRam.xise - [ram_8_4.vhd]

Design Sources for: Implementation Hierarchy Processes: ram_8_4 - Behavioral

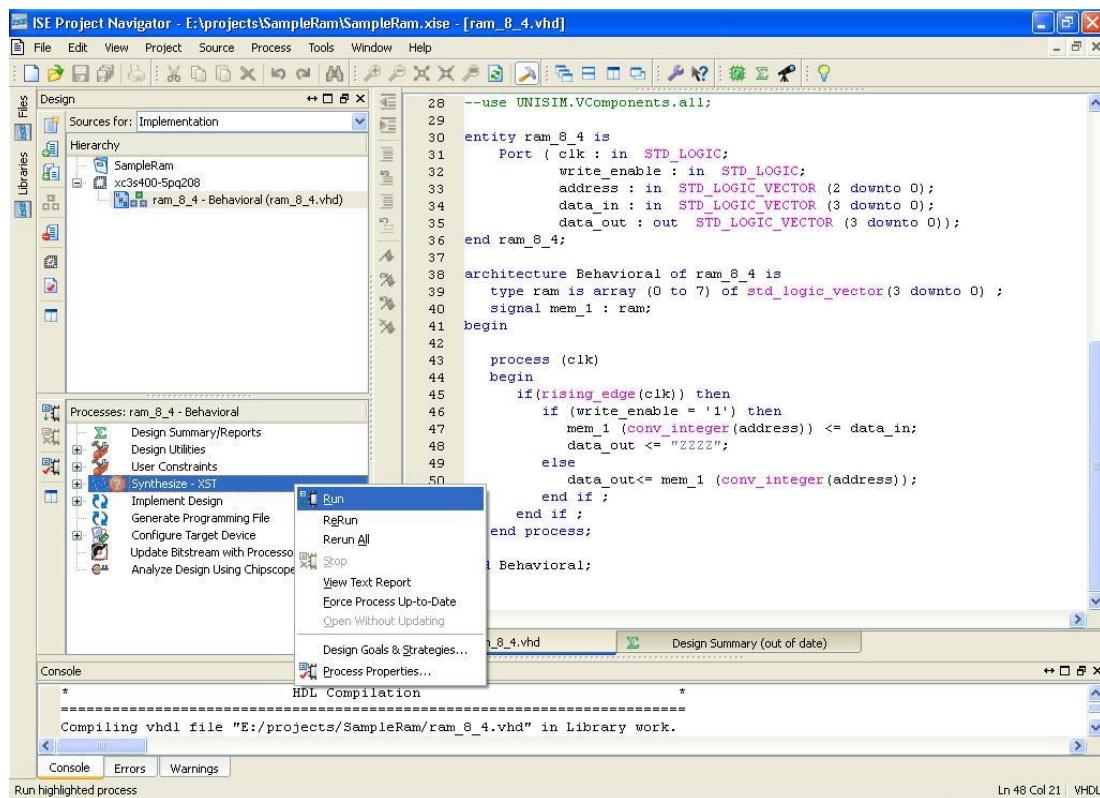
Design Summary/Reports Design Utilities User Constraints Synthesize - XST Implement Design Generate Programming File Configure Target Device Update Bitstream with Processor Data Analyze Design Using ChipScope

HDL Compilation

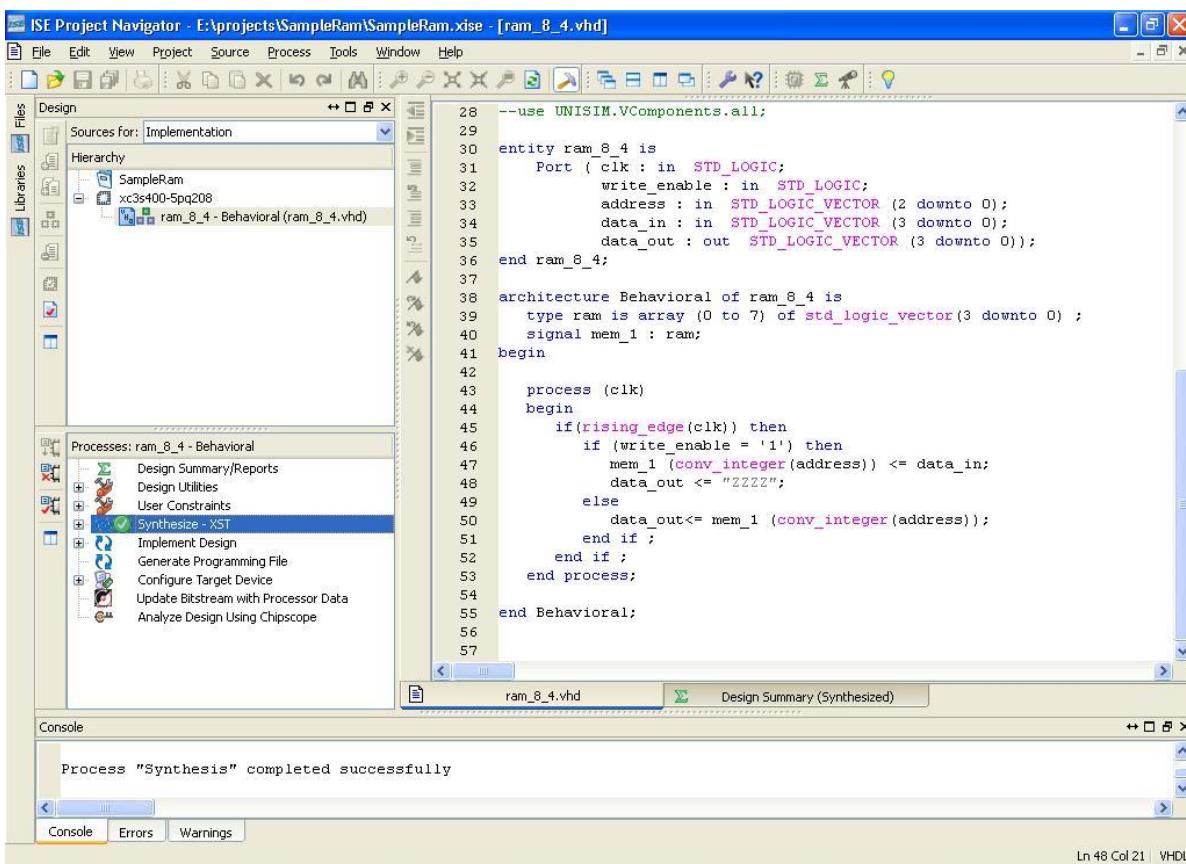
Compiling vhdl file "E:/projects/SampleRam/ram_8_4.vhd" in Library work.

Console Errors Warnings

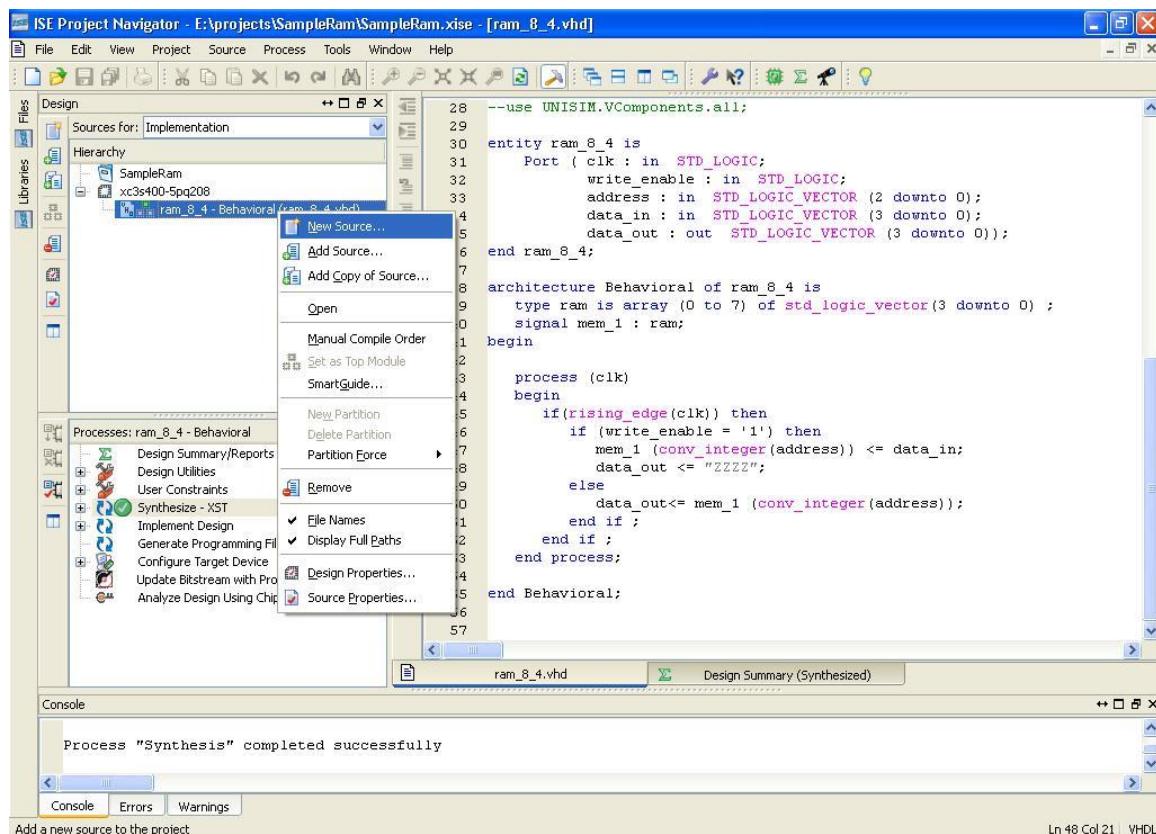
شکل ۱۱ نوشتن کد طراحی



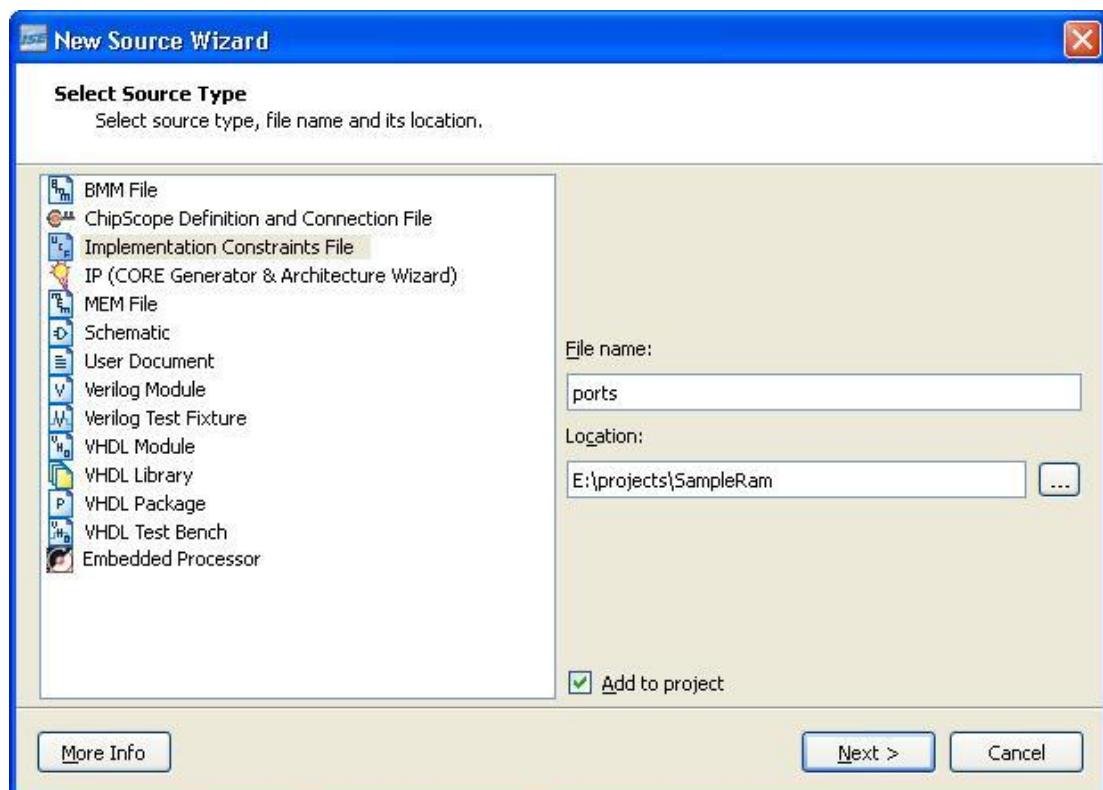
شکل ۱۲ منوی سنتز



شکل ۱۳ نتیجه سنتز



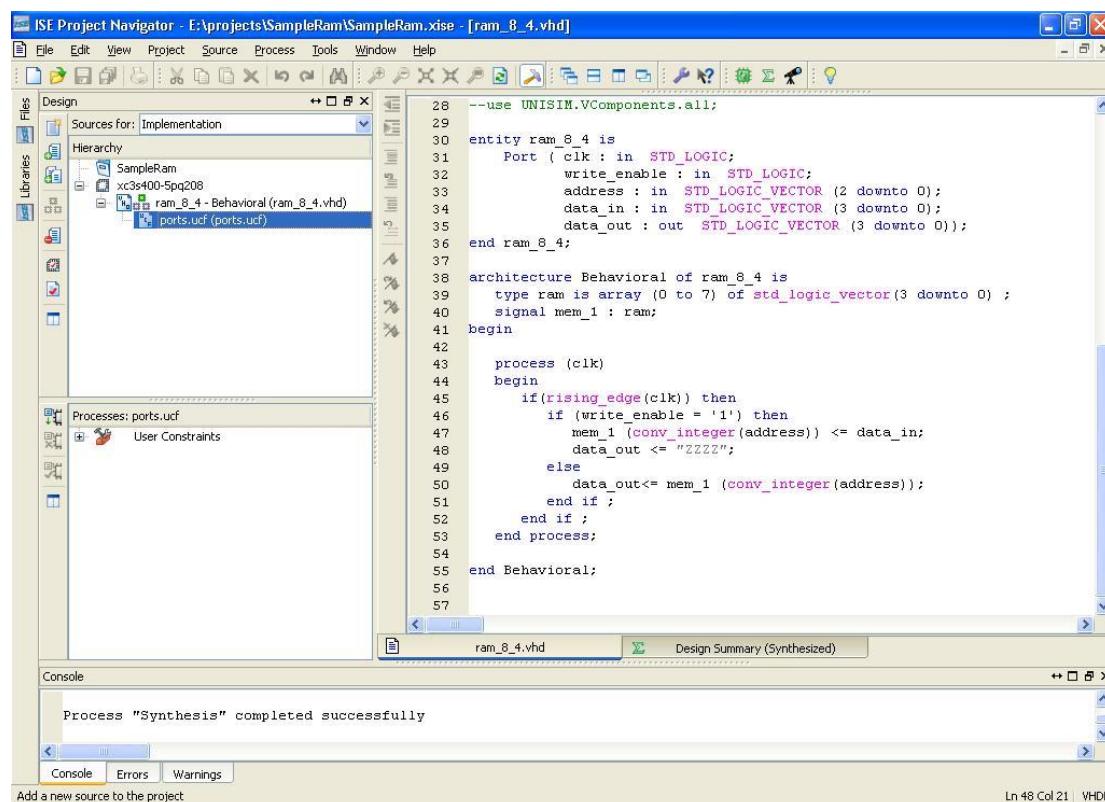
شکل ۱۴ منوی ایجاد فایل محدودیت (تعیین پورت های FPGA)



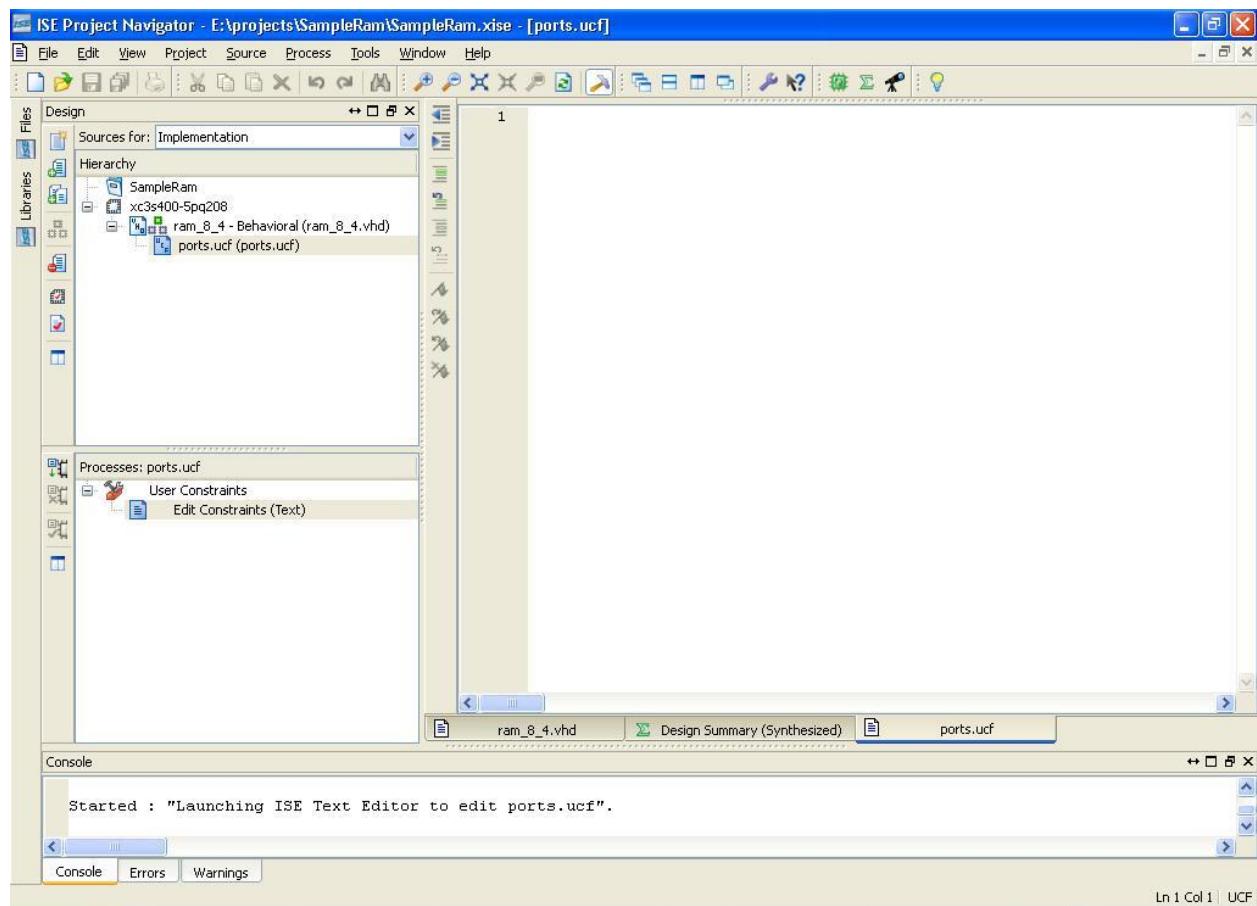
شکل ۱۵ تعیین نام فایل محدودیت



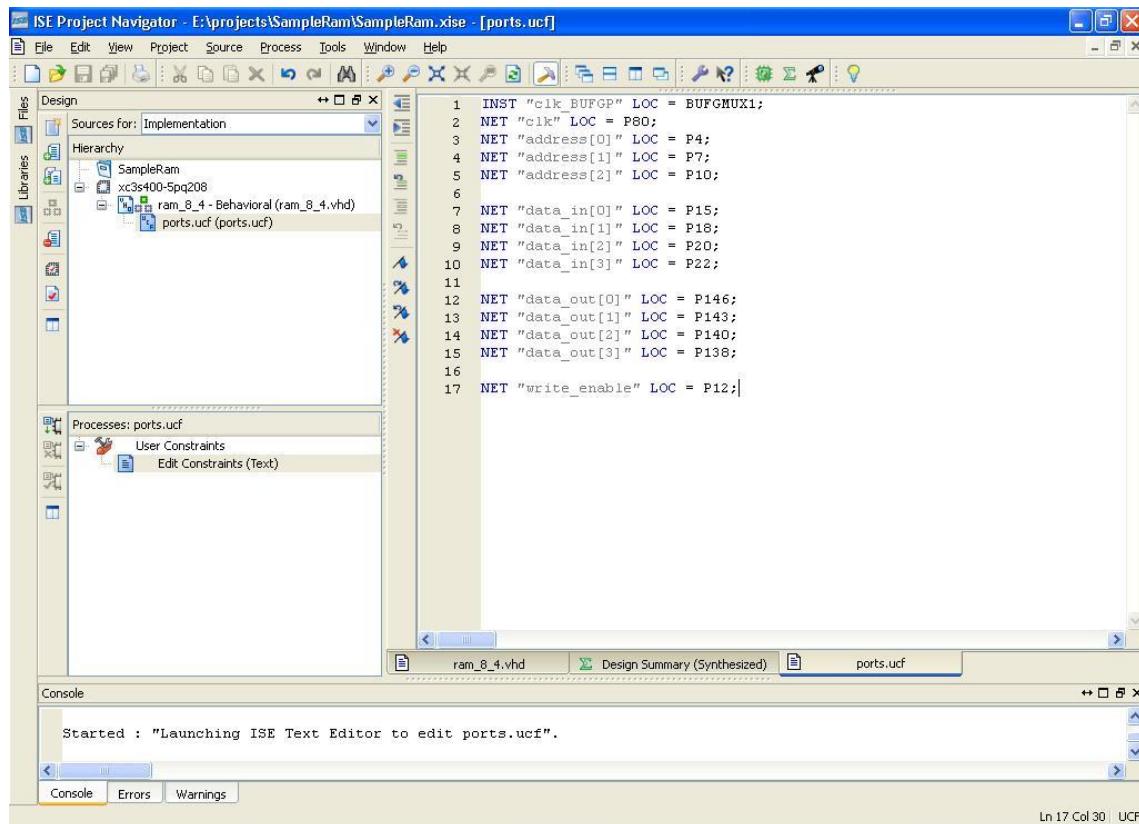
شکل ۱۶ ویزگی های فایل محدودیت ایجاد شده



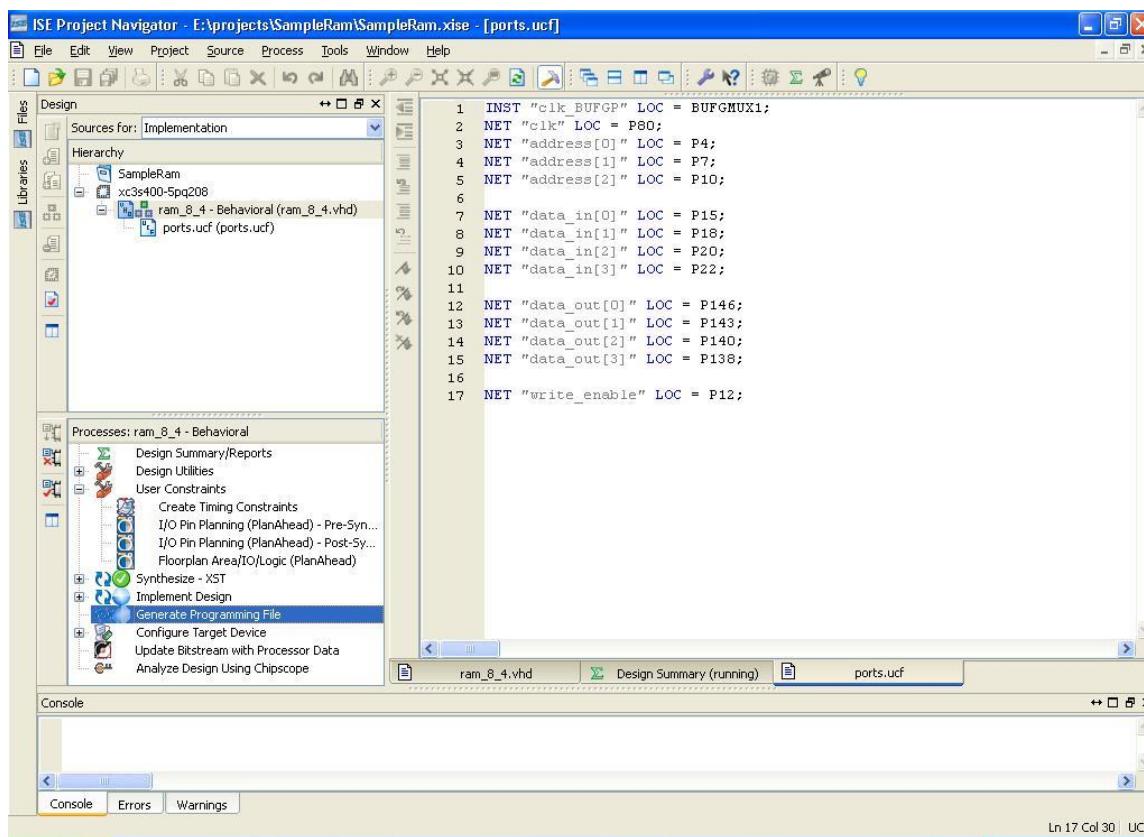
شکل ۱۷ فایل محدودیت در منوی Hierarchy



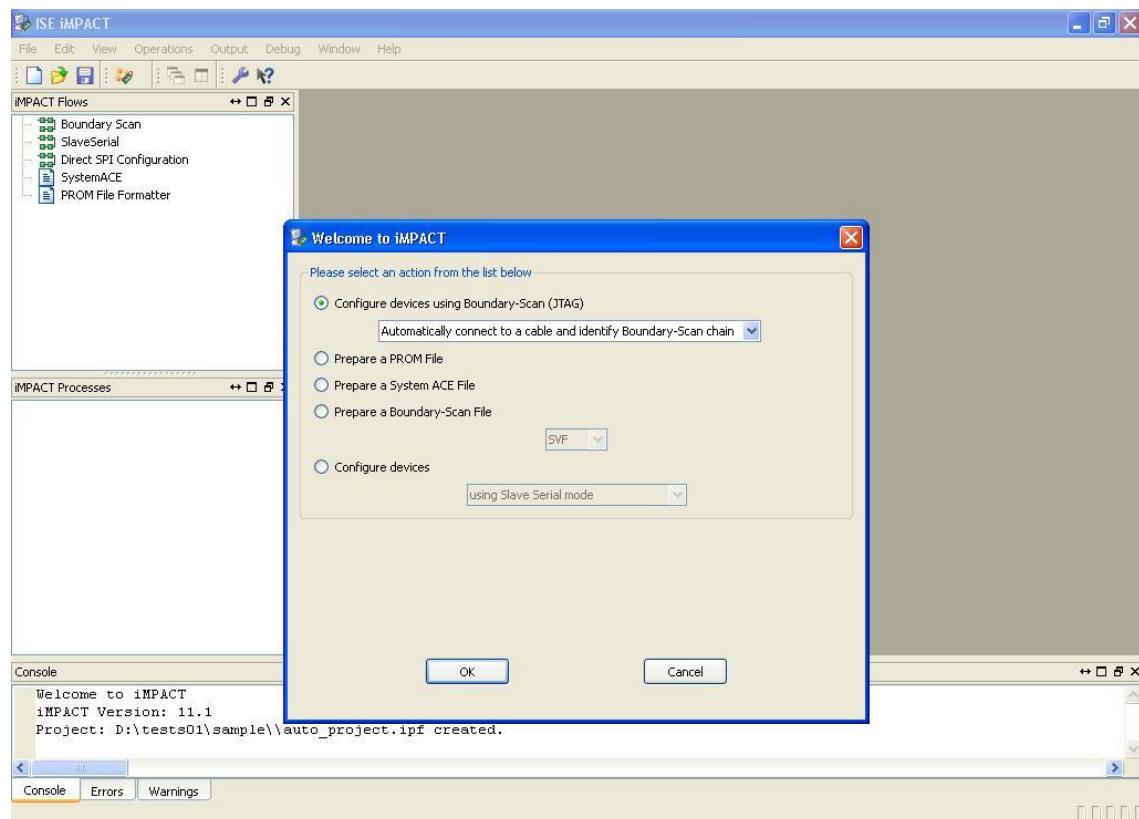
شکل ۱۸ ویرایش فایل محدودیت به صورت متنی



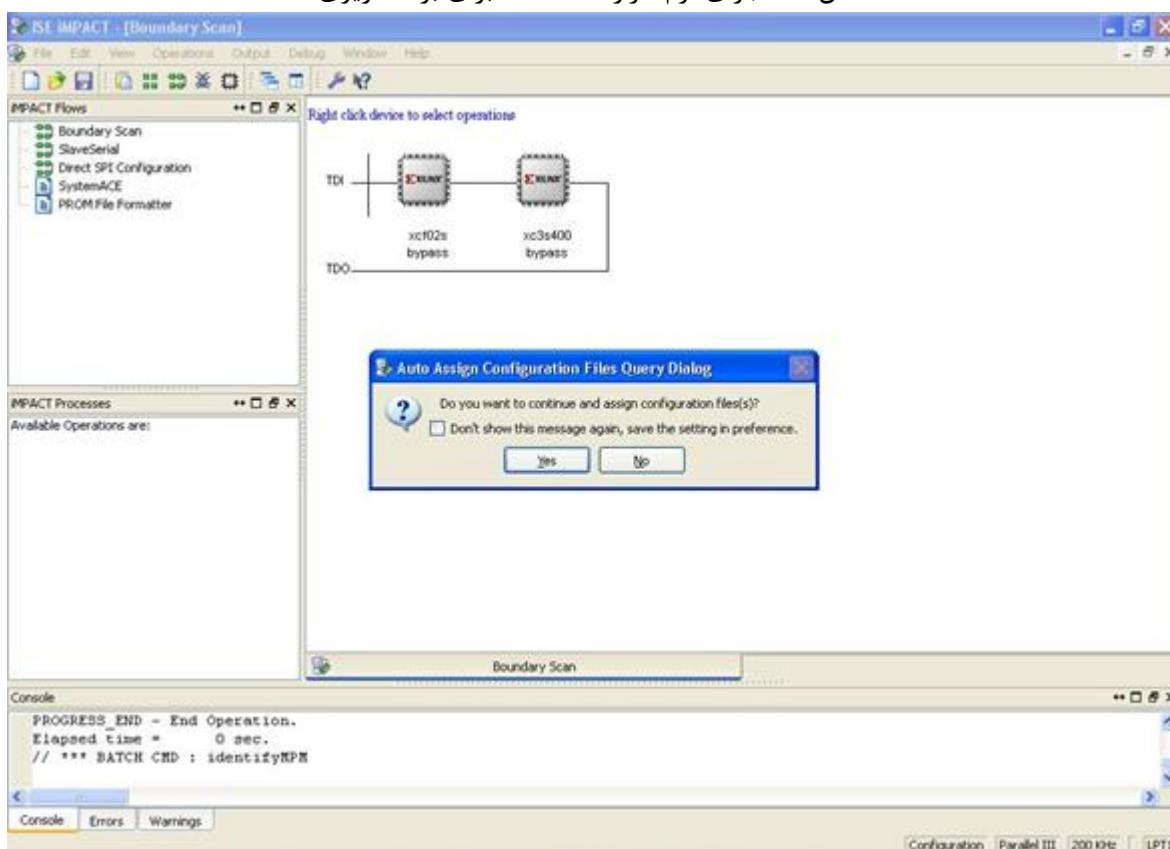
شکل ۱۹ تعیین port های طرح بر روی FPGA



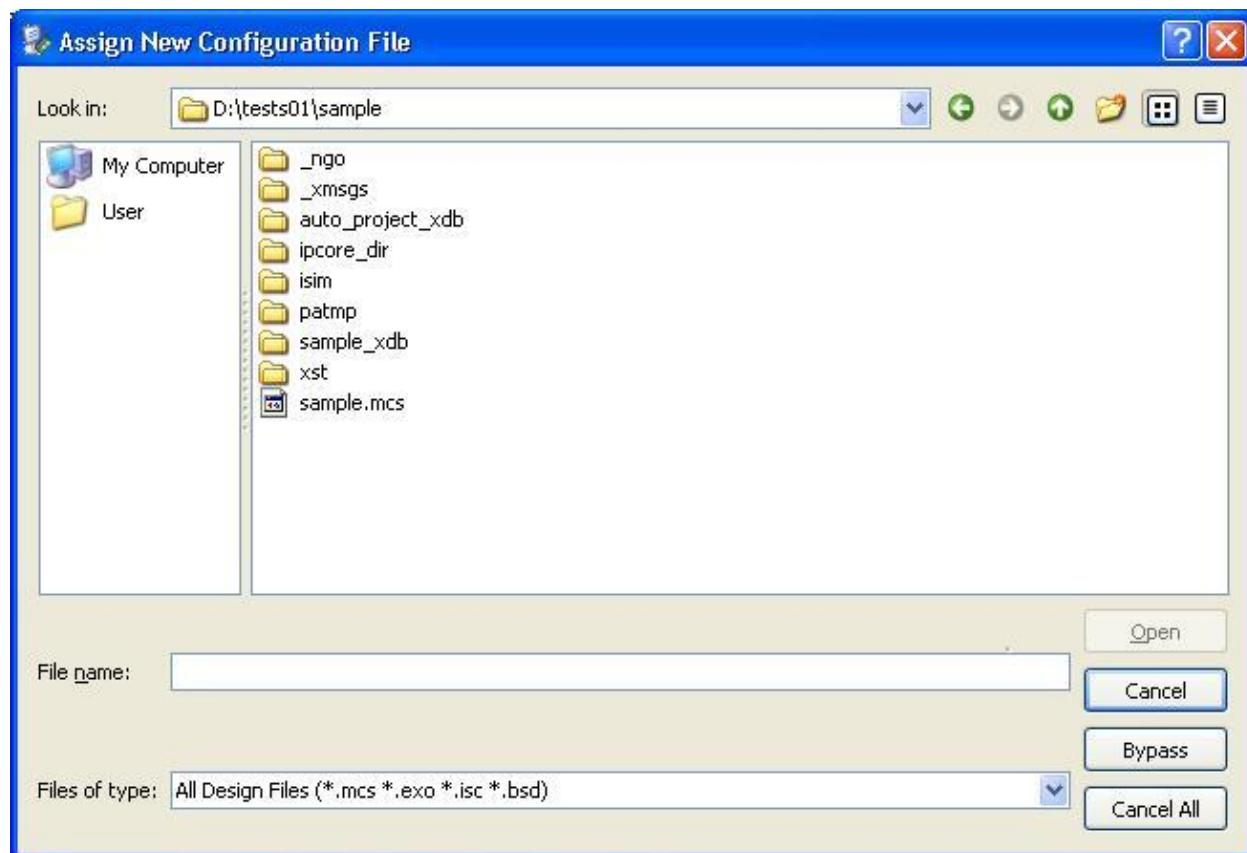
شکل ۲۰ ایجاد فایل برنامه ریزی



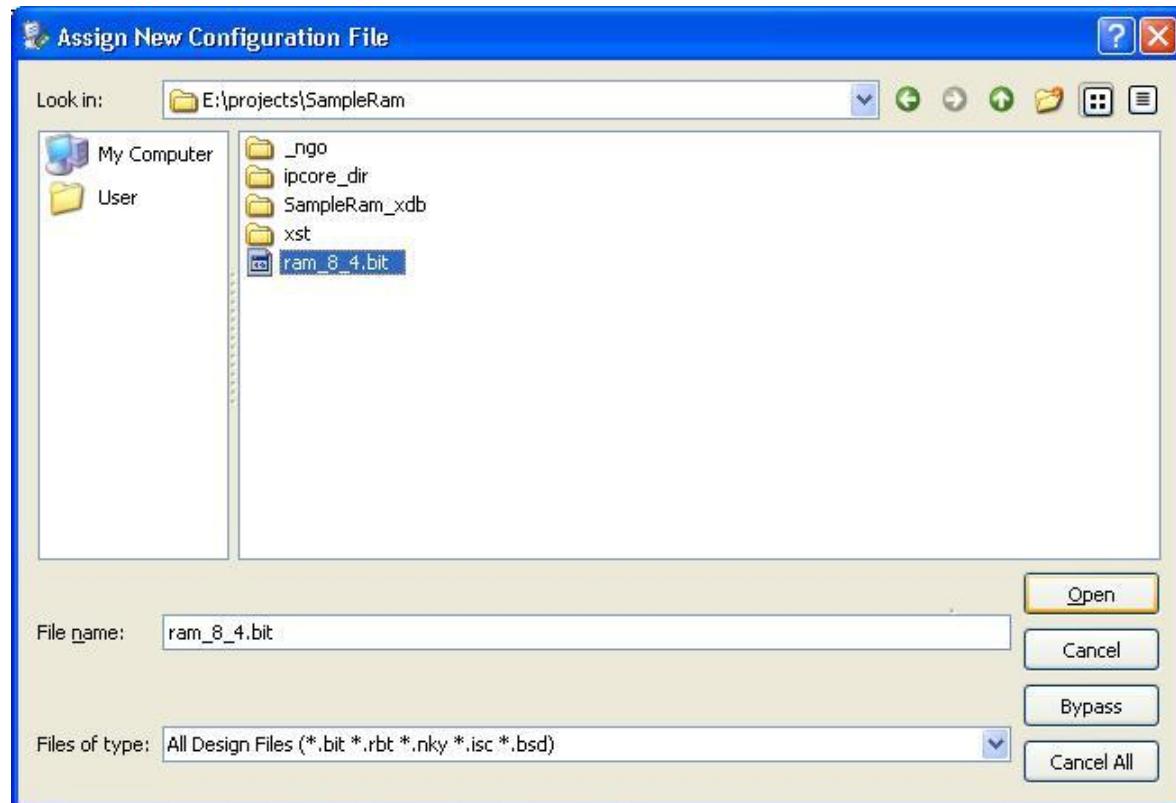
شکل ۲۱ اجرای نرم افزار iMPACT برای برنامه ریزی



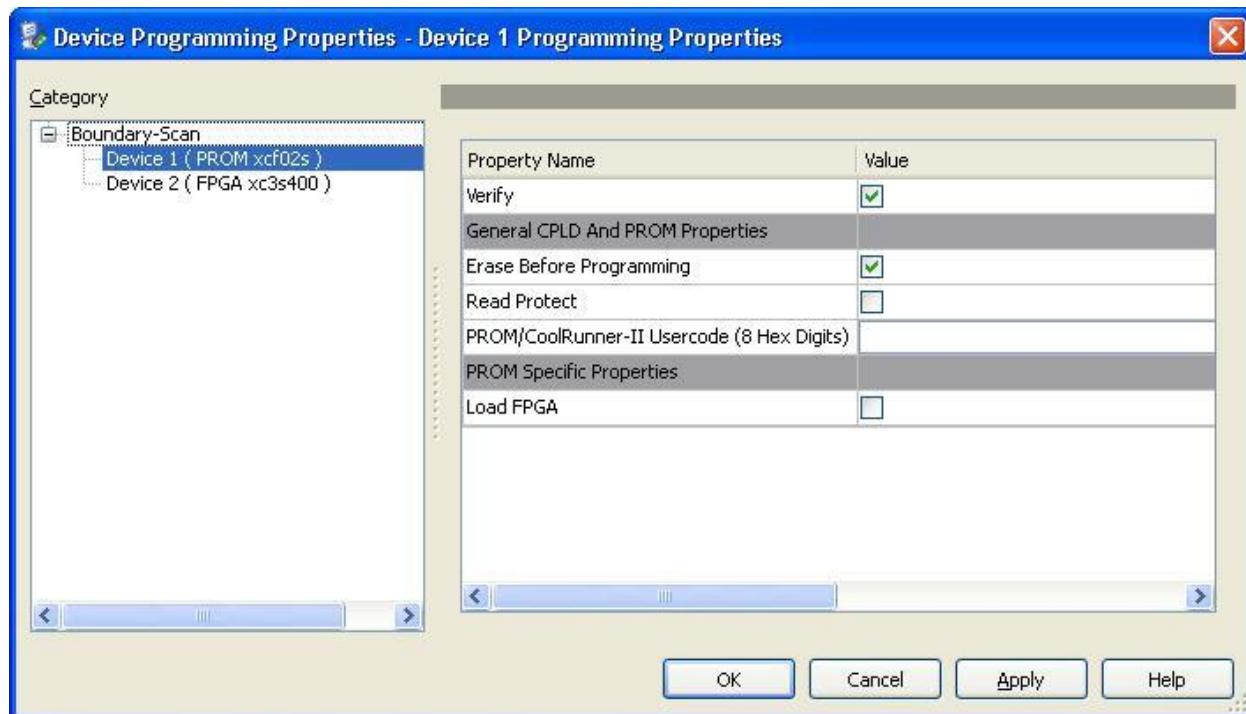
شکل ۲۲ ابزار های شناخته شده



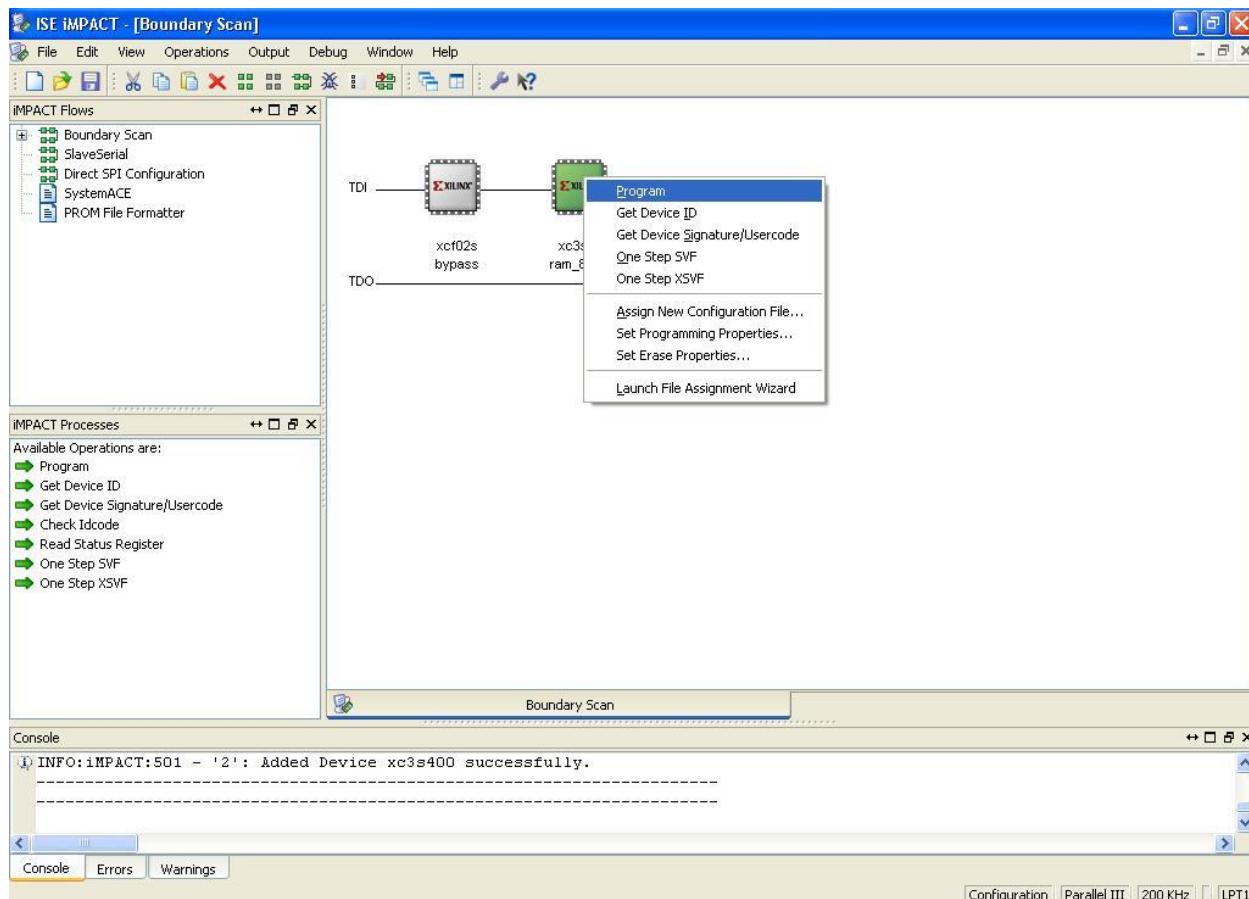
شکل ۲۳ تعیین فایل برنامه ریزی RAM موجود در روی بورد. (فعلاً این مرحله را bypass می‌کنیم).



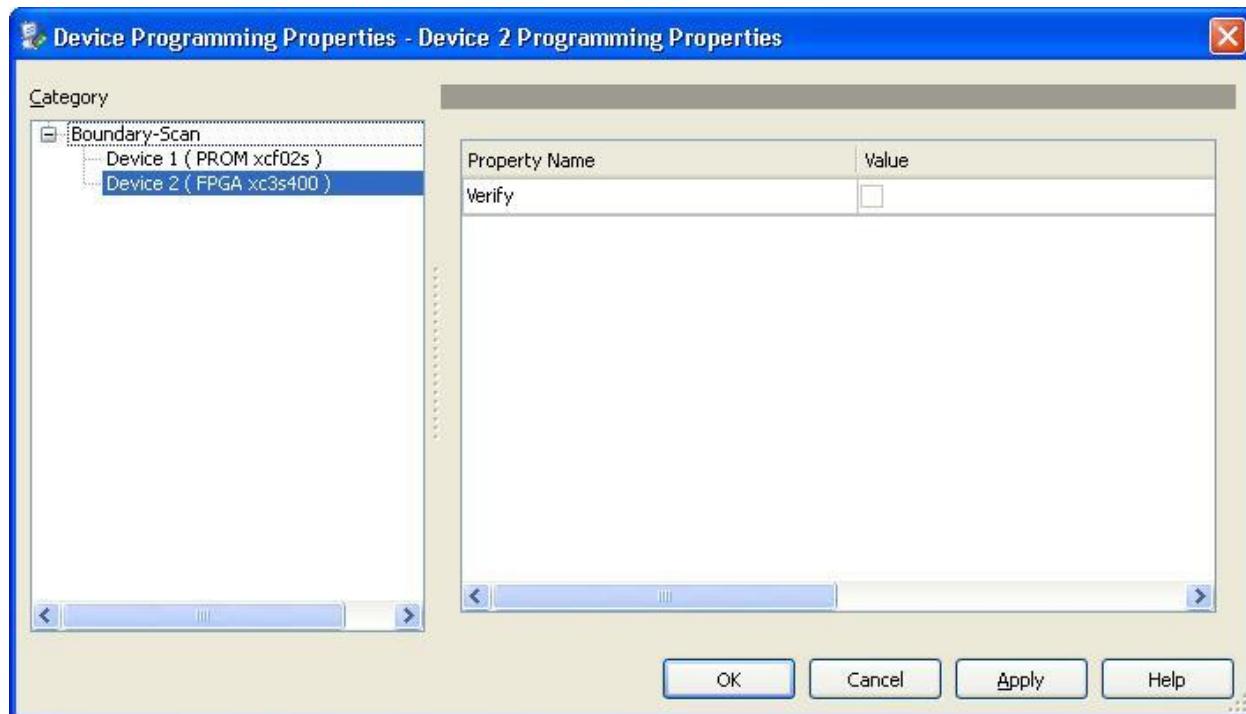
شکل ۲۴ انتخاب فایل برنامه ریزی تولید شده در مرحله قبل



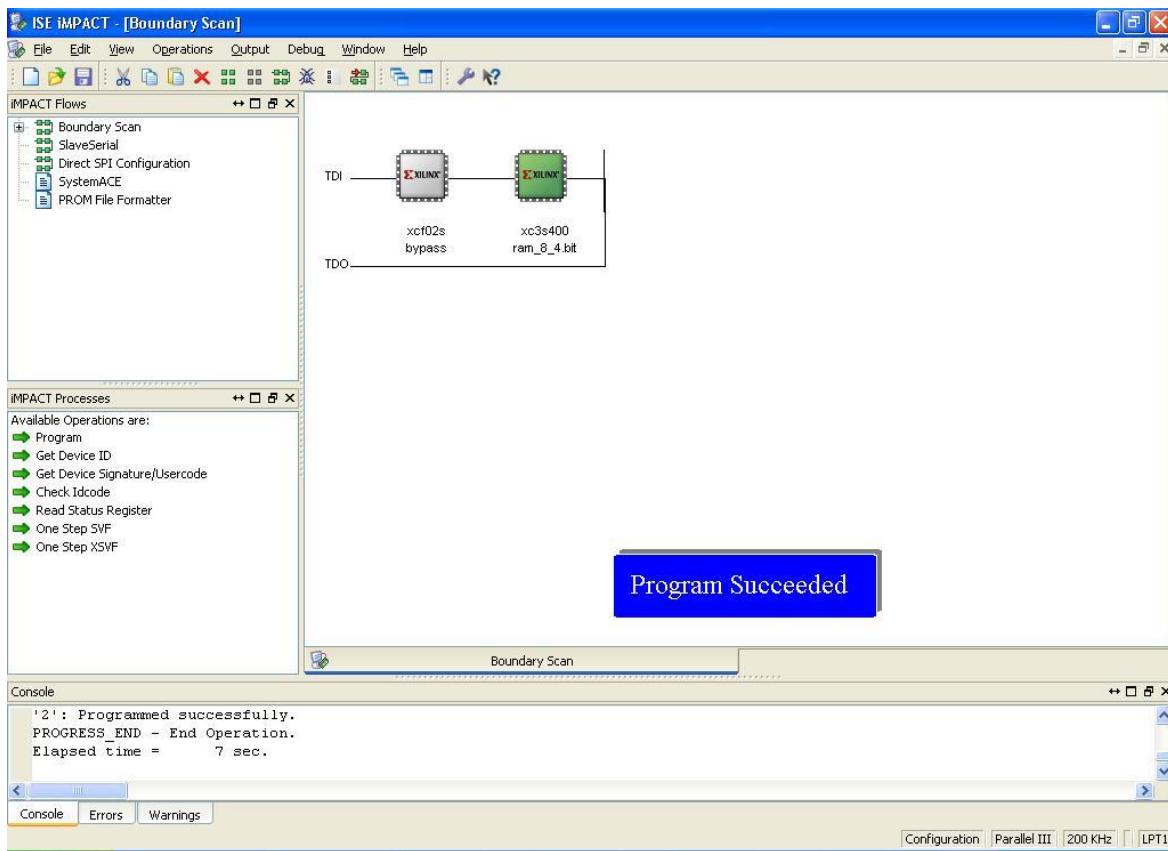
شکل ۲۵ نمایش ویژگی های برنامه ریزی ابزار



شکل ۲۶ برنامه ریزی FPGA

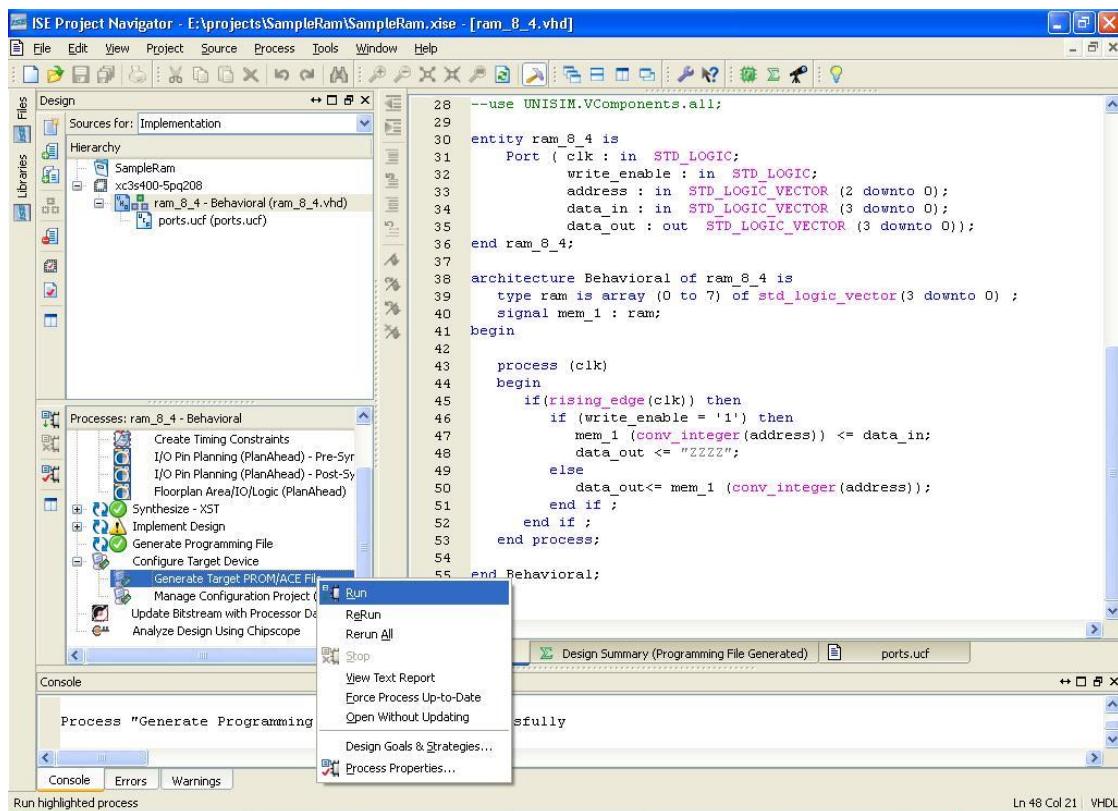


شکل ۲۷ تأیید نهایی

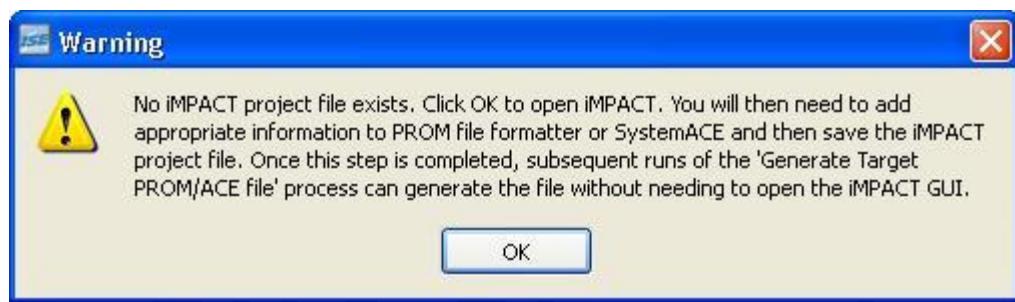


شکل ۲۸ وضعیت برنامه ریزی

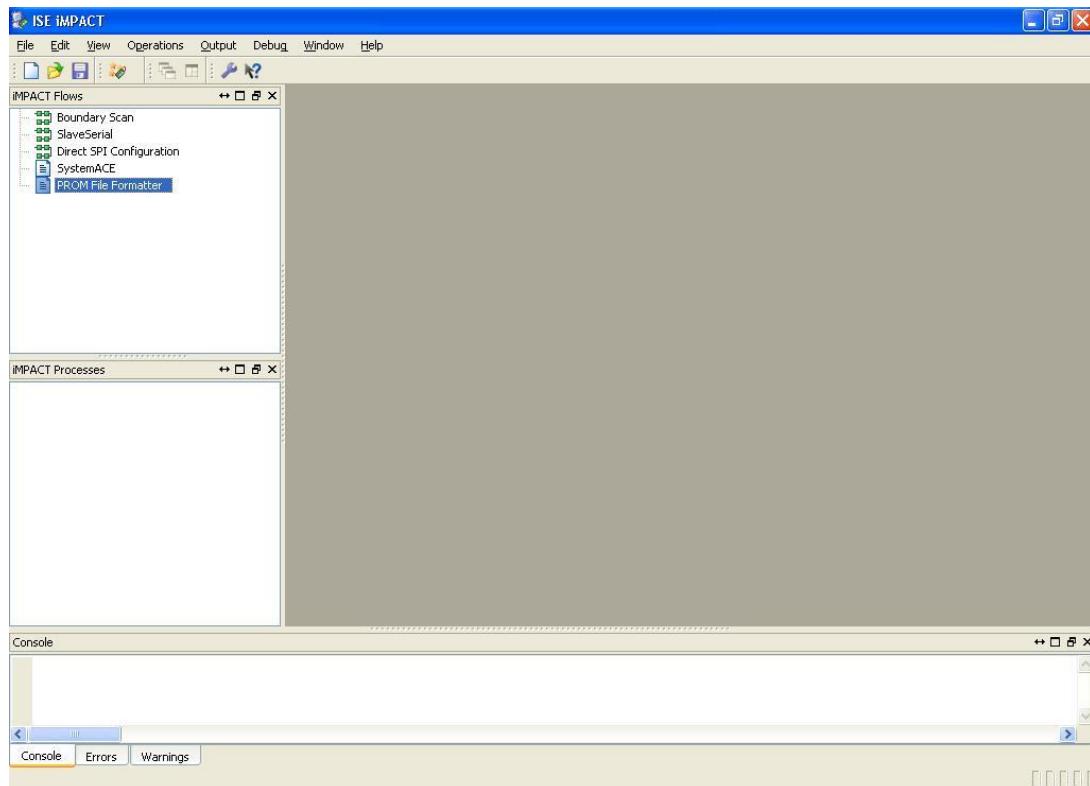
فاز دوم: برنامه ریزی RAM روی بورد



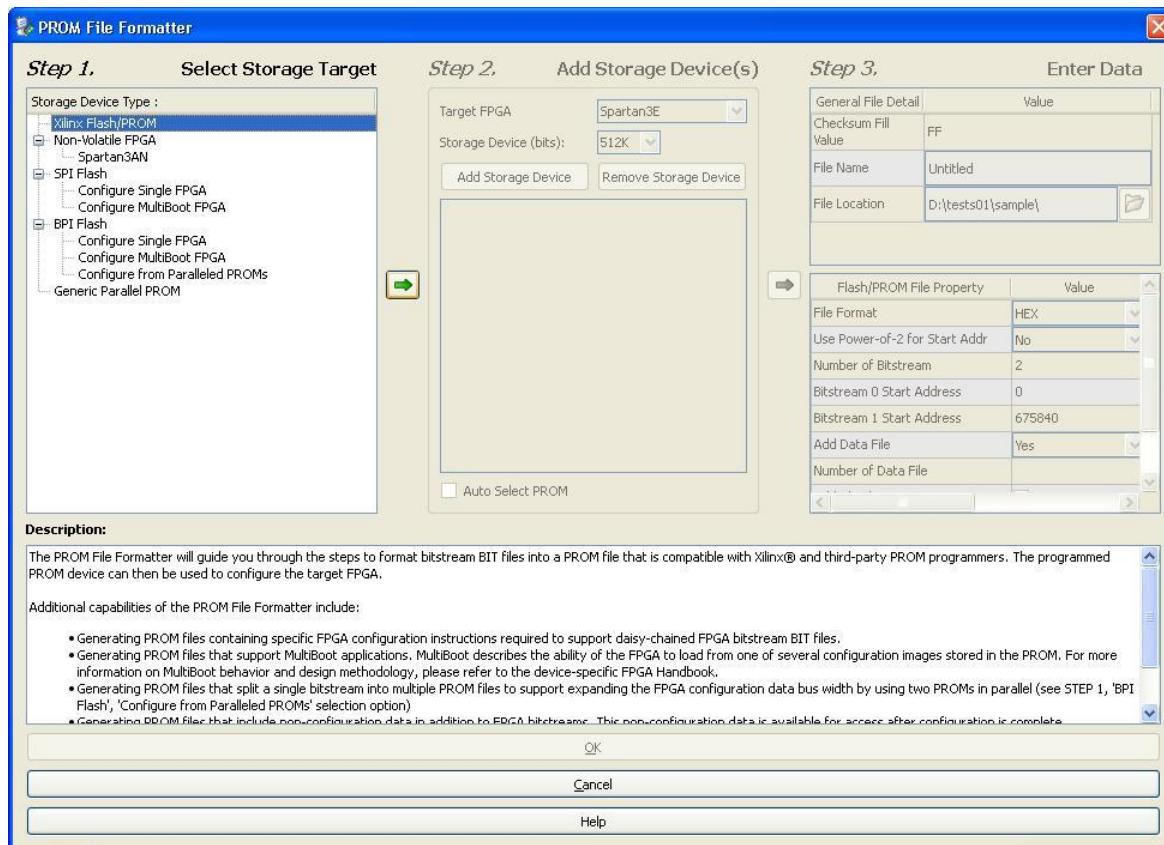
شکل ۲۹ منوی ایجاد فایل برنامه ریزی RAM



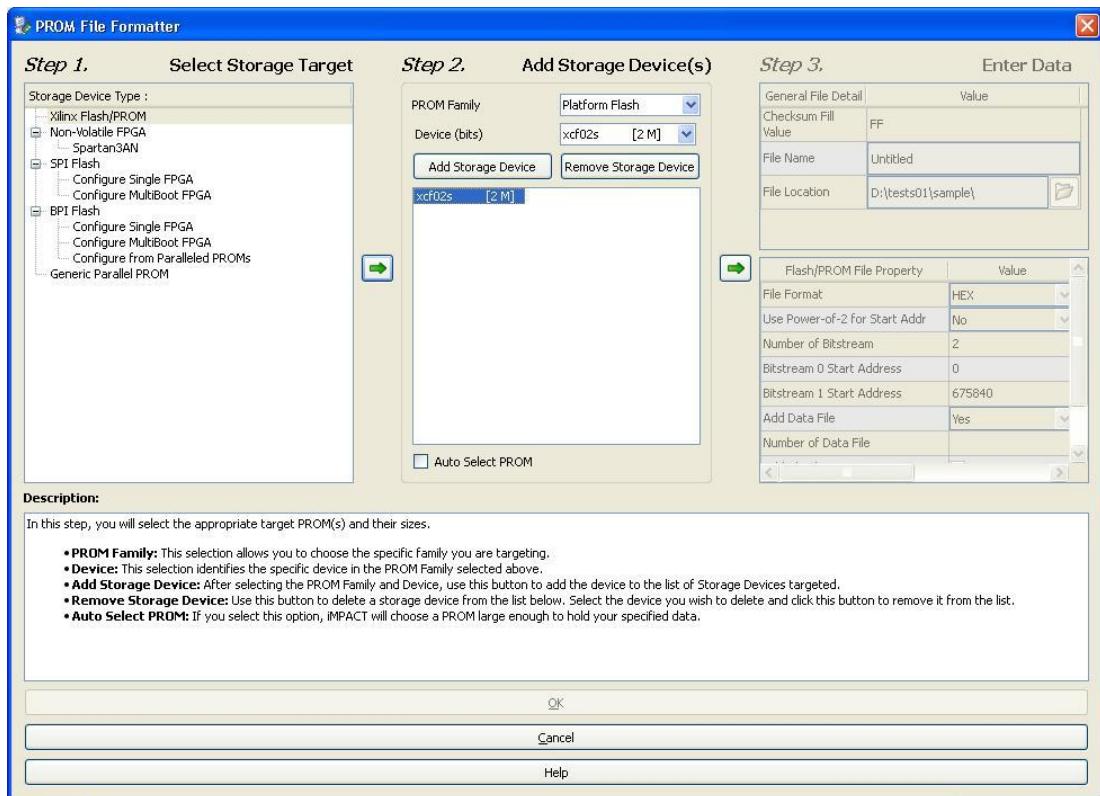
شکل ۳۰ هشدار مبنی بر ایجاد پروژه iMPACT جدید



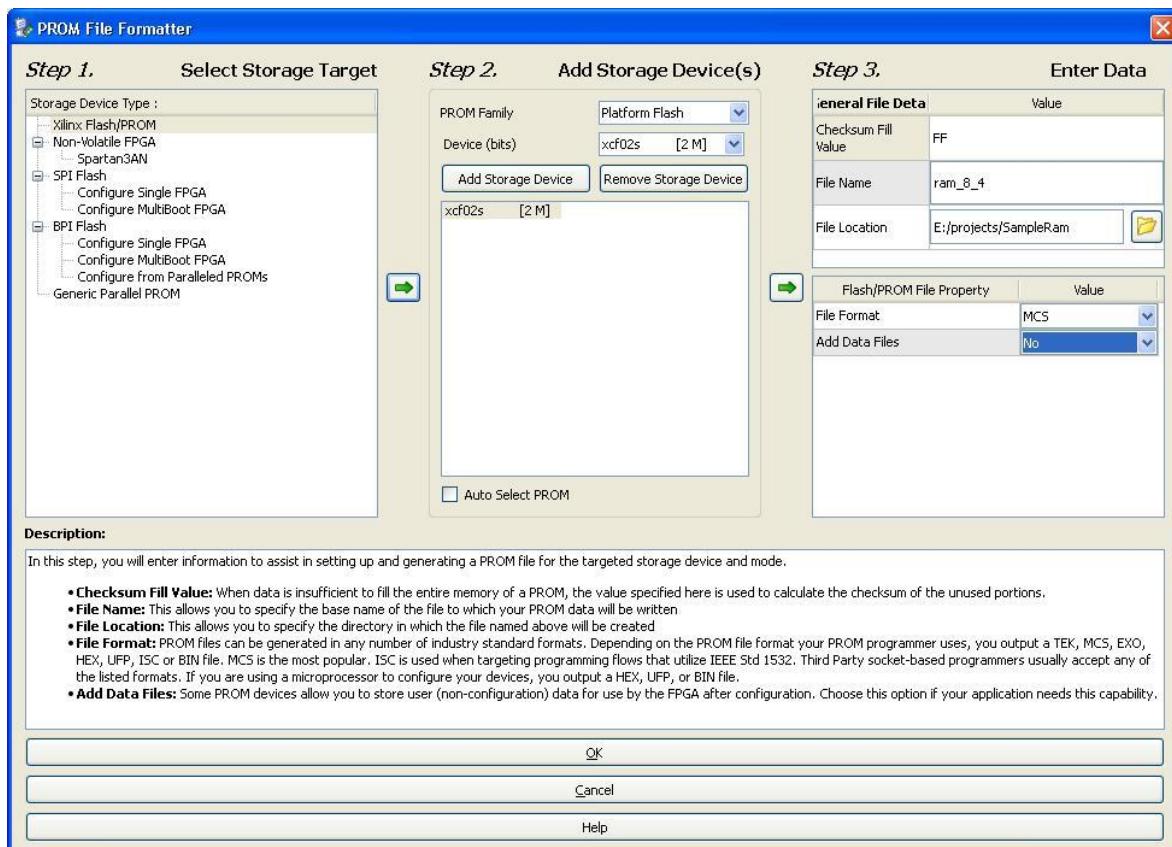
شکل ۳۱ منوی ایجاد فایل برنامه ریزی در پنجره iMPACT



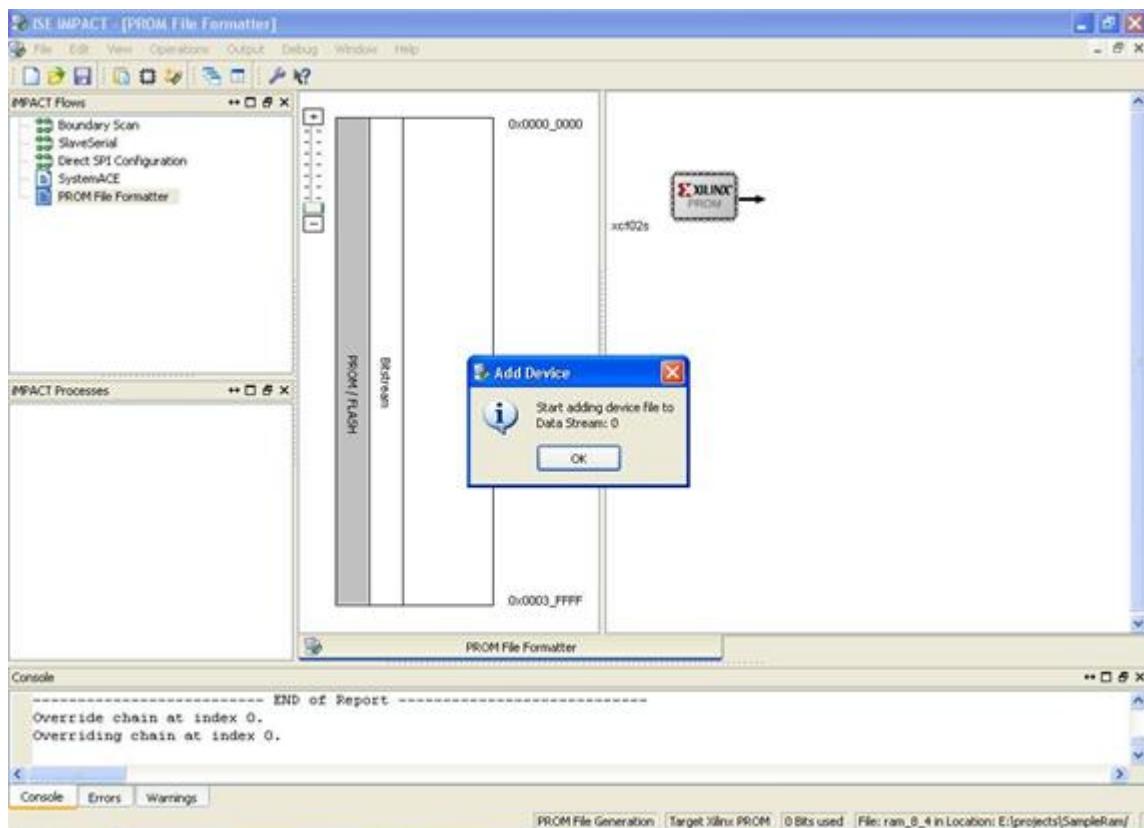
شکل ۳۲ گام اول ایجاد فایل برنامه ریزی



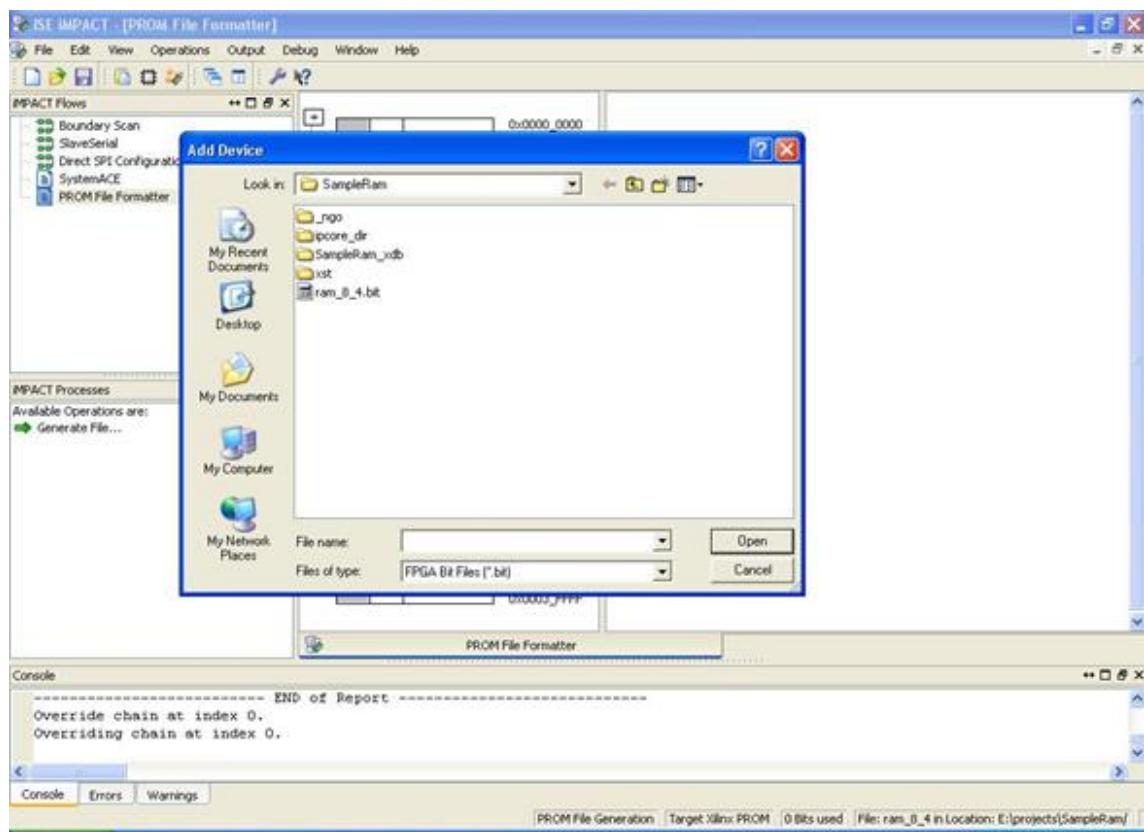
شکل ۳۳ گام دوم ایجاد فایل برنامه‌ریزی



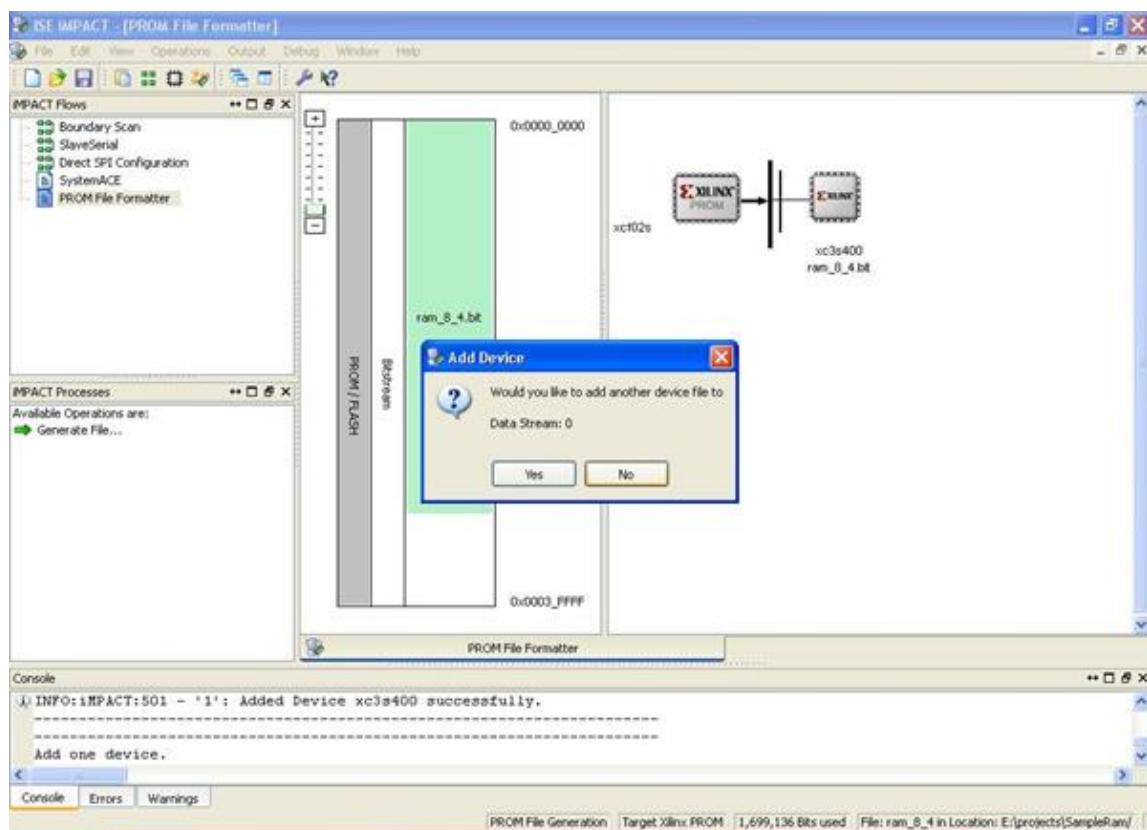
شکل ۳۴ گام سوم ایجاد فایل برنامه‌ریزی



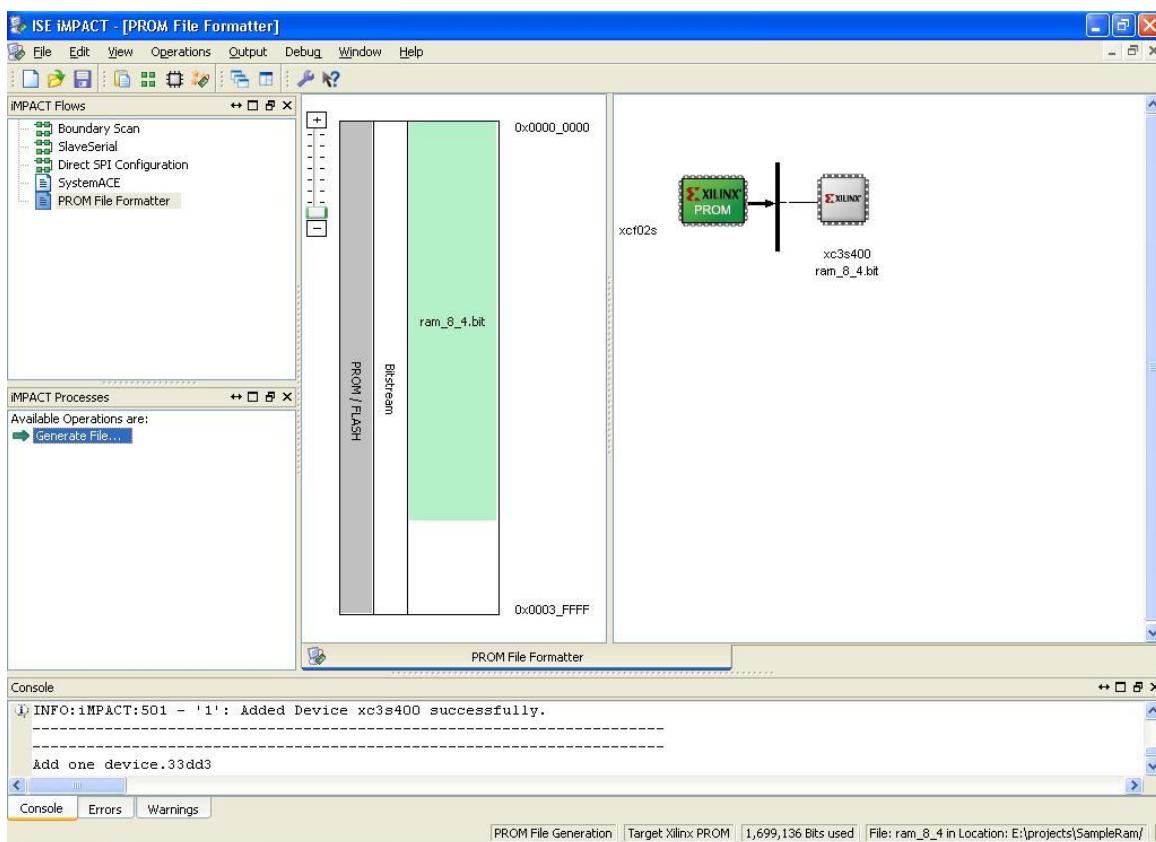
شکل ۳۵ جستجوی RAM های روی بورد توسط iMPACT



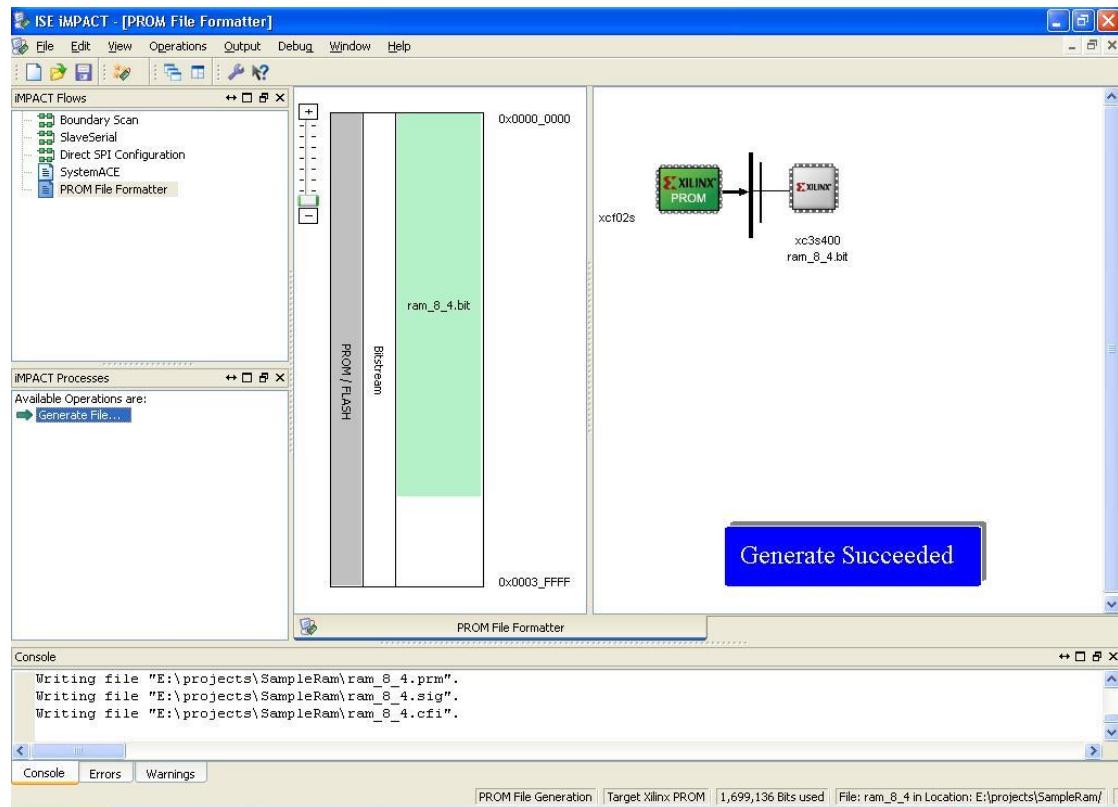
شکل ۳۶ انتخاب فایل برنامه ریزی FPGA برای برنامه ریزی RAM



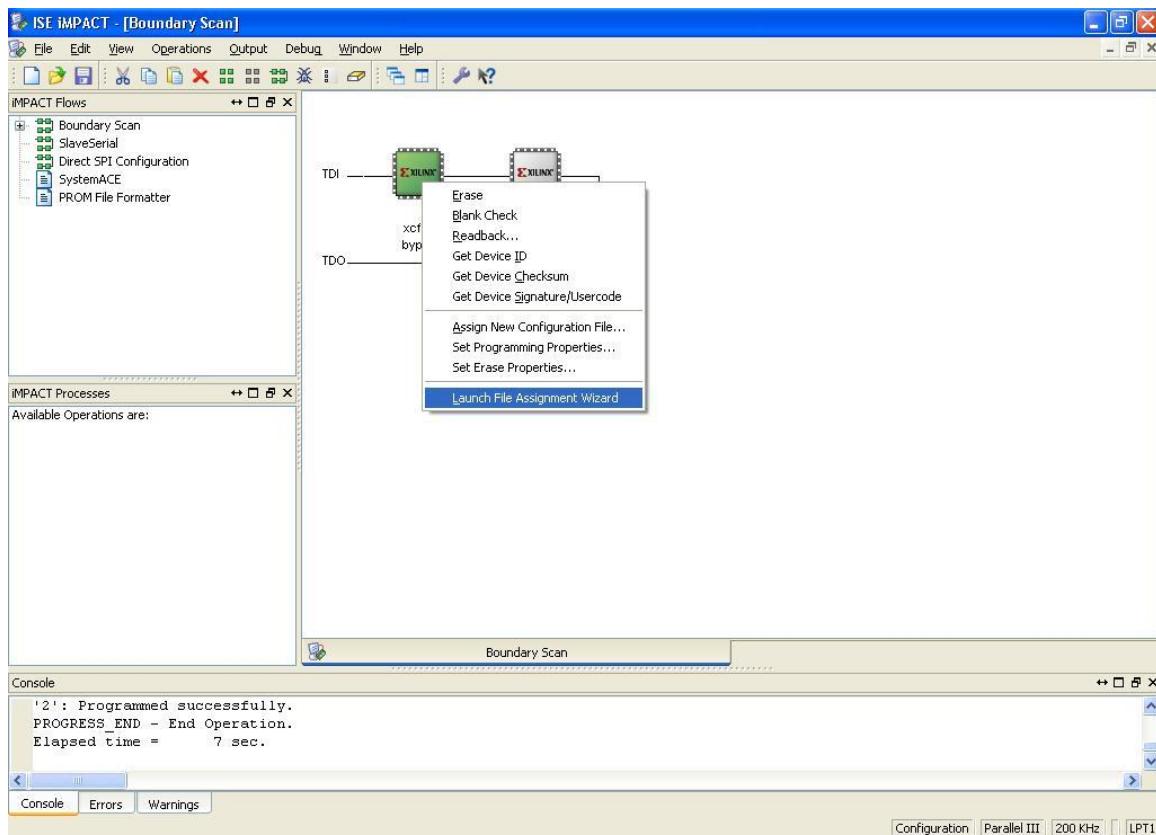
شکل ۳۷ هشدار مبنی بر افزودن فایل جدید



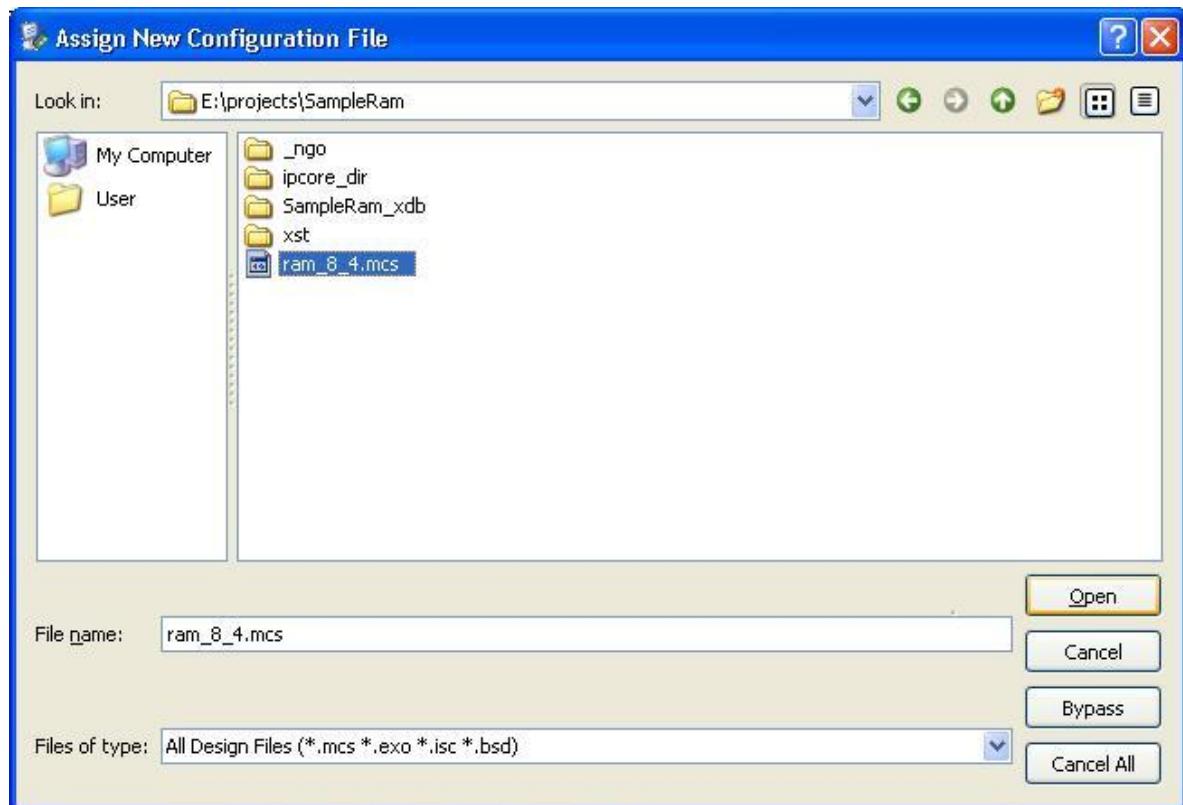
شکل ۳۸ نتیجه انتخاب فایل



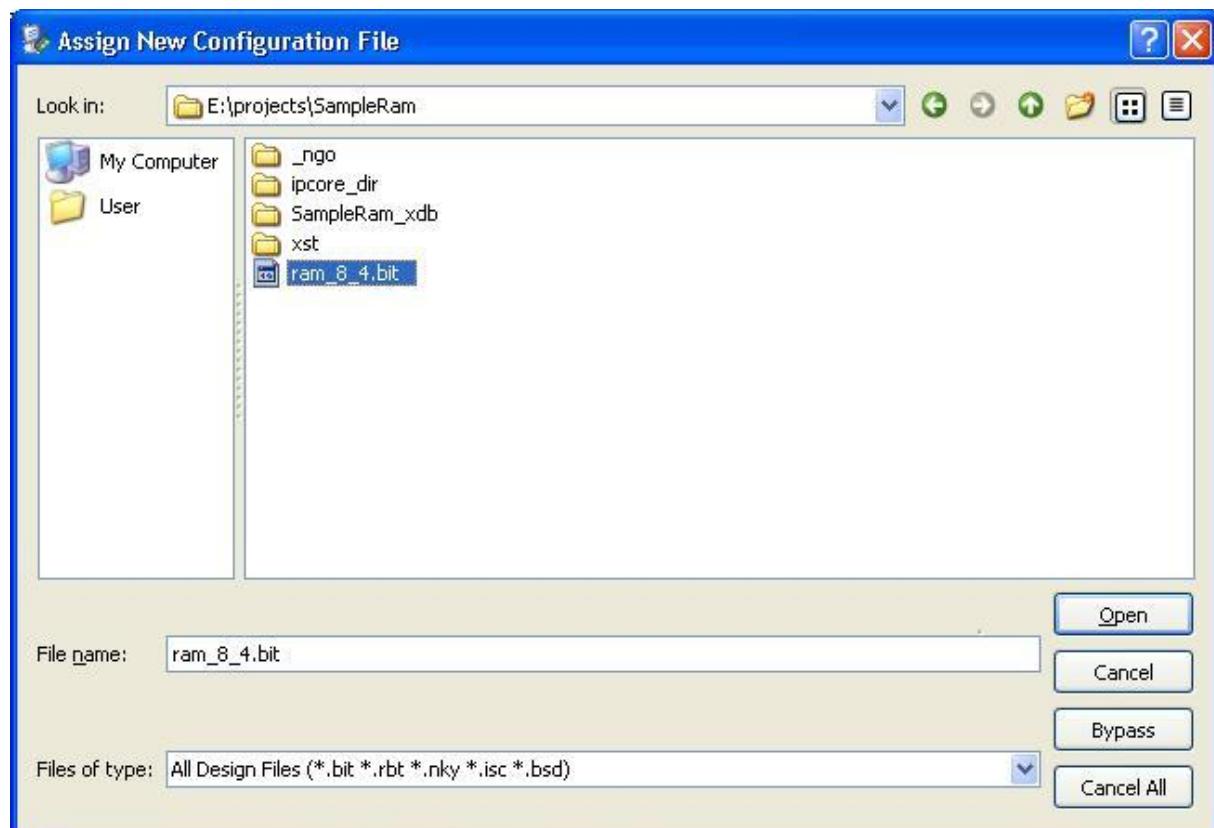
شکل ۳۹ ایجاد موفقیت آمیز فایل برنامه ریزی RAM



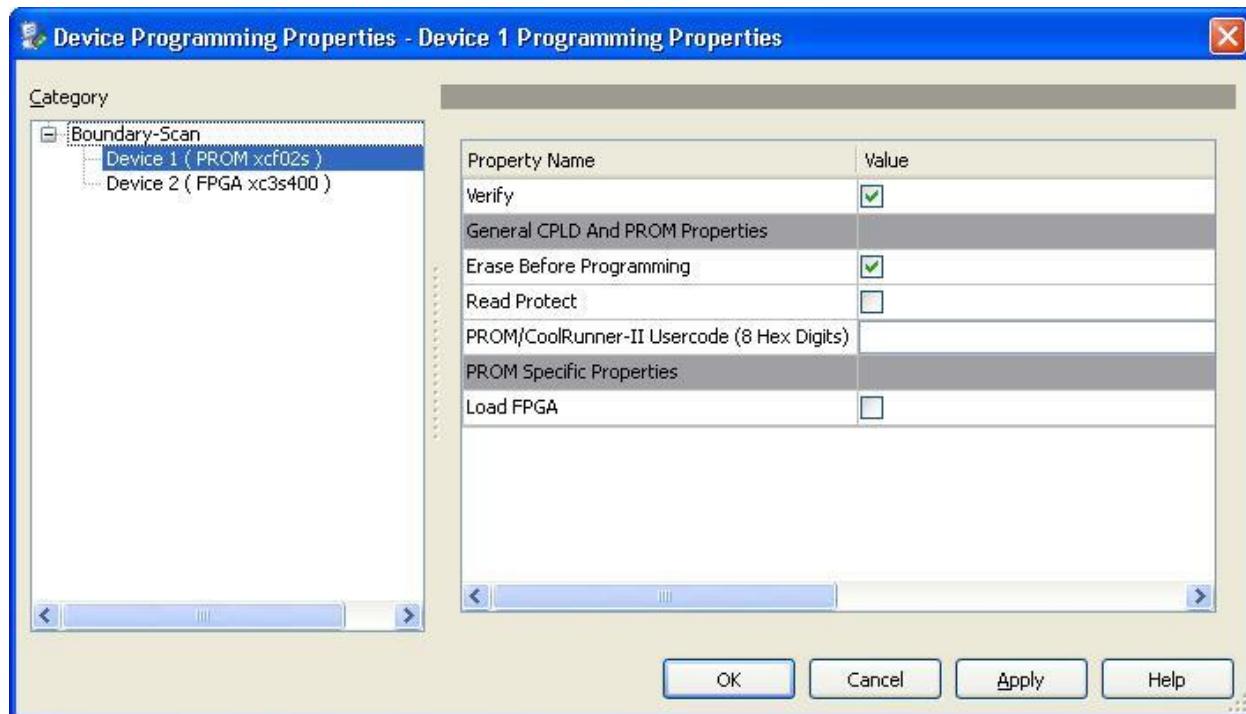
شکل ۴۰ تعیین فایل برنامه ریزی ram



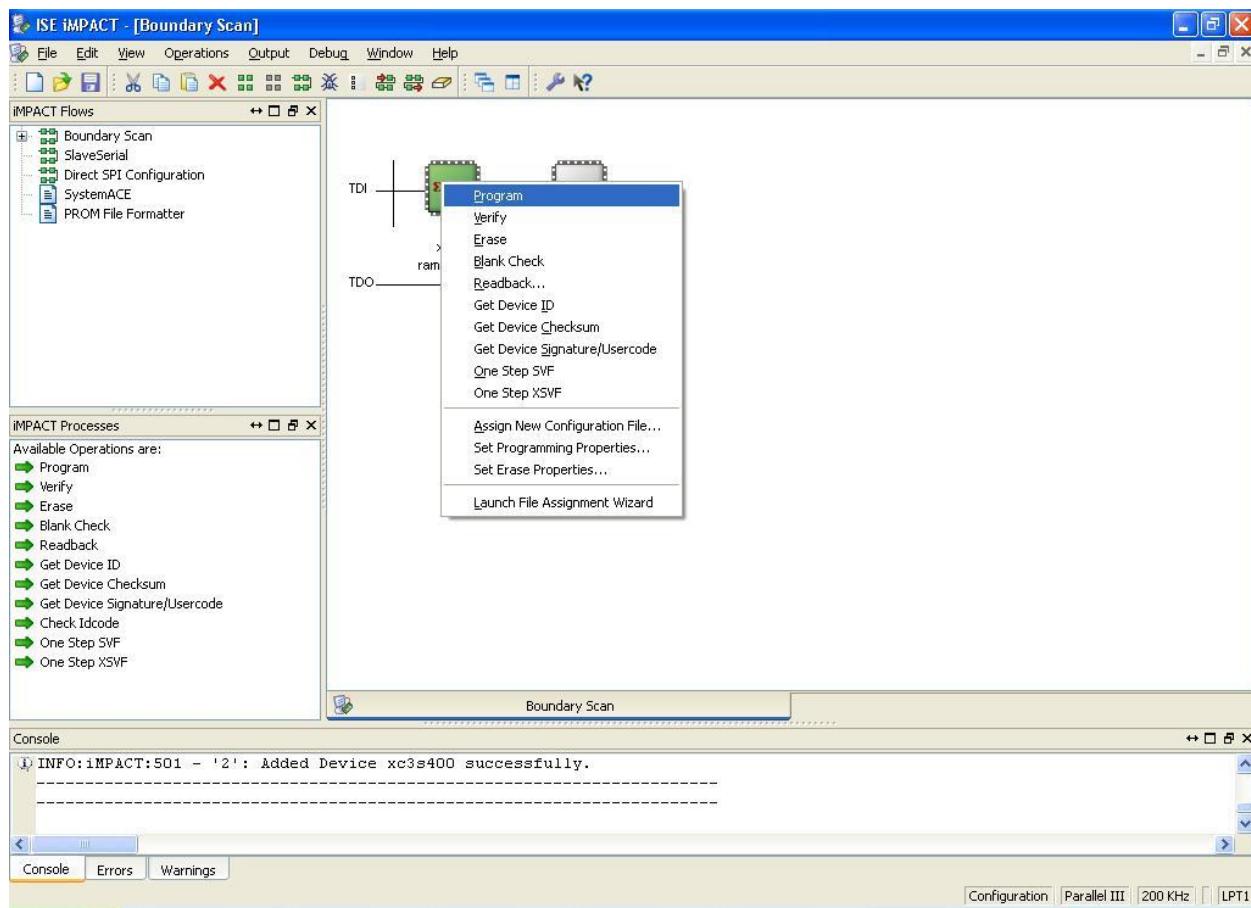
شکل ۴۱ انتخاب فایل برنامه ریزی RAM



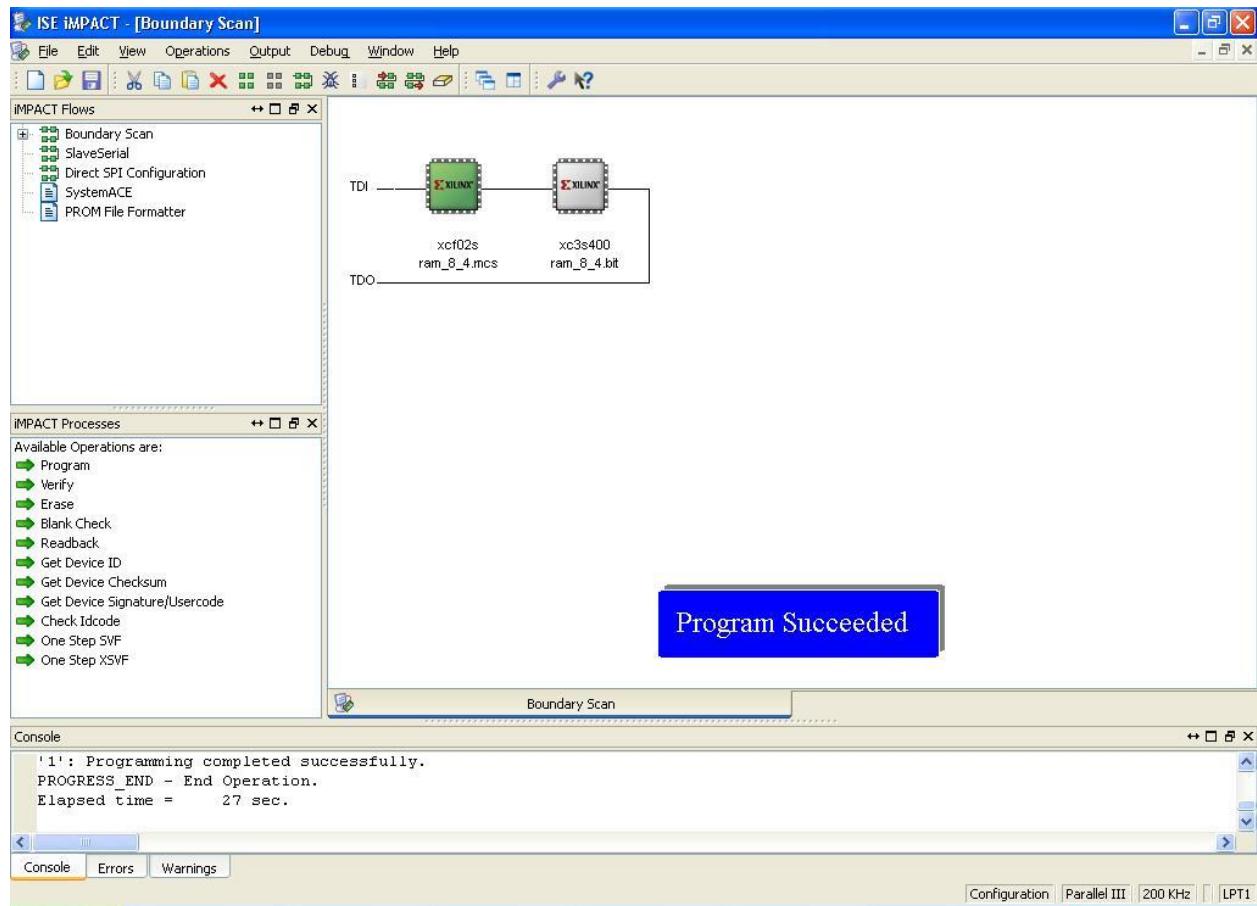
شکل ۴۲ انتخاب فایل برنامه ریزی FPGA



شکل ۴۳ ویژگی‌های برنامه‌ریزی دو ابزار



شکل ۴۴ برنامه‌ریزی RAM



شکل ۴۵ نتیجه موفقیت‌آمیز برنامه‌ریزی RAM