

## اهداف

این مقاله به بررسی IP Core های آماده موجود در محیط Xilinx Vivado می‌پردازد. ابتدا روند طراحی با استفاده از ابزار IP Catalog در محیط Xilinx Vivado شرح داده شده است. سپس توضیحاتی در رابطه با عملیات CORDIC ارائه شده و در نهایت با استفاده از امکاناتی که ابزار Vivado جهت پیاده‌سازی توابع مثلثاتی و ریشه‌ی دوم در اختیار می‌گذارد یک مدار ساده پیاده‌سازی جذرگیر پیاده‌سازی شده است. یک محیط آزمون (testbench) جهت درستی‌سنجی و شبیه‌سازی مدار طراحی شده طراحی شده است.

## تعریف

هسته مالکیت معنوی (Intellectual Property Core) یک سیستم یا مدار طراحی شده است که به منظور طراحی سیستم‌های دیجیتال FPGA یا ASIC و یا ... مورد استفاده قرار می‌گیرد. نیازی به طراحی و پیاده‌سازی دوباره آن نیست و می‌توان از آن‌ها برای توسعه سیستم‌های دیجیتال استفاده کرد. این هسته‌ها معمولاً با در نظر گرفتن ساختار داخلی FPGA و به صورت بهینه طراحی می‌شود.


## انواع IP Core ها

IP Core های موجود در محیط Vivado به دو دسته‌ی اصلی نرم و سخت هستند. منظور از هسته‌های مالکیت معنوی سخت (Hard IP Core) این است که مدار مورد نظر به صورت سخت در داخل تراشه قرار داده شده است. به عنوان مثال یک مدار دیکد کننده‌ی رمز AES یا یک پردازنده ARM بخشی از تراشه را اشغال کرده است. در مقابل هسته‌های مالکیت معنوی نرم (Soft IP Core) در حالت عادی در تراشه وجود ندارند و در صورتی که کاربر به آن نیاز داشته باشد، یک مدار بهینه توصیف شده و با استفاده از بلوک‌های منطقی پیاده‌سازی می‌شود. به عنوان مثال پردازنده‌ی MicroBlaze یک هسته‌ی پردازشی نرم است.

باتوجه به این که تعداد IP های سخت در یک تراشه‌ی FPGA محدود است، لذا در محیط Vivado این امکان وجود دارد که بتوان IP های سخت را با استفاده از LUT ها طراحی کرد. به عنوان مثال ۸۰ عدد بلوک DSP سخت (DSP Slices) در تراشه‌ی XC7Z010 وجود دارد. البته کاربر می‌تواند آن‌را به صورت نرم با استفاده از بلوک‌های منطقی پیاده‌سازی کرده و استفاده نماید.

برای هر IP Core در محیط Vivado یک فایل راهنما وجود دارد که ساختار داخلی، امکانات موجود و نحوه‌ی پیکربندی در آن شرح داده شده است. با مراجعه به آدرس زیر و جستجوی IP Core مورد نظر فایل راهنمای آن را یافت.

<https://www.xilinx.com/products/intellectual-property.html>



Browse by AI Inference

Browse Audio Video & Imaging

Browse Communications

Browse DSP and Math

Browse Embedded

Browse IP Utility

Browse Interface & Interconnect

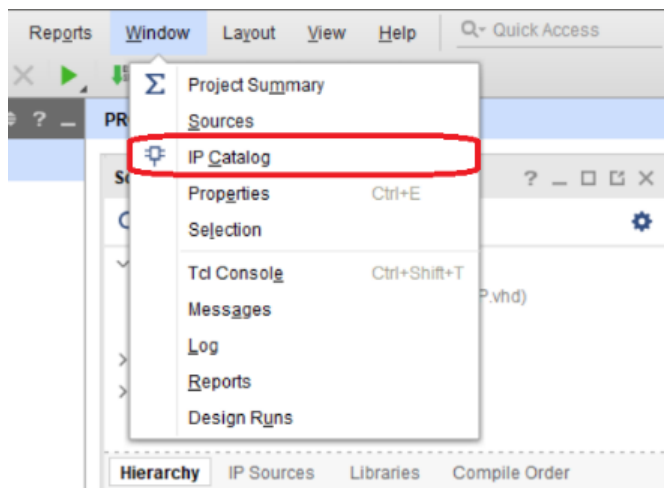
Browse Memory & Controllers

Browse by Market

شکل ۱ طبقه‌بندی IP Core های شرکت Xilinx

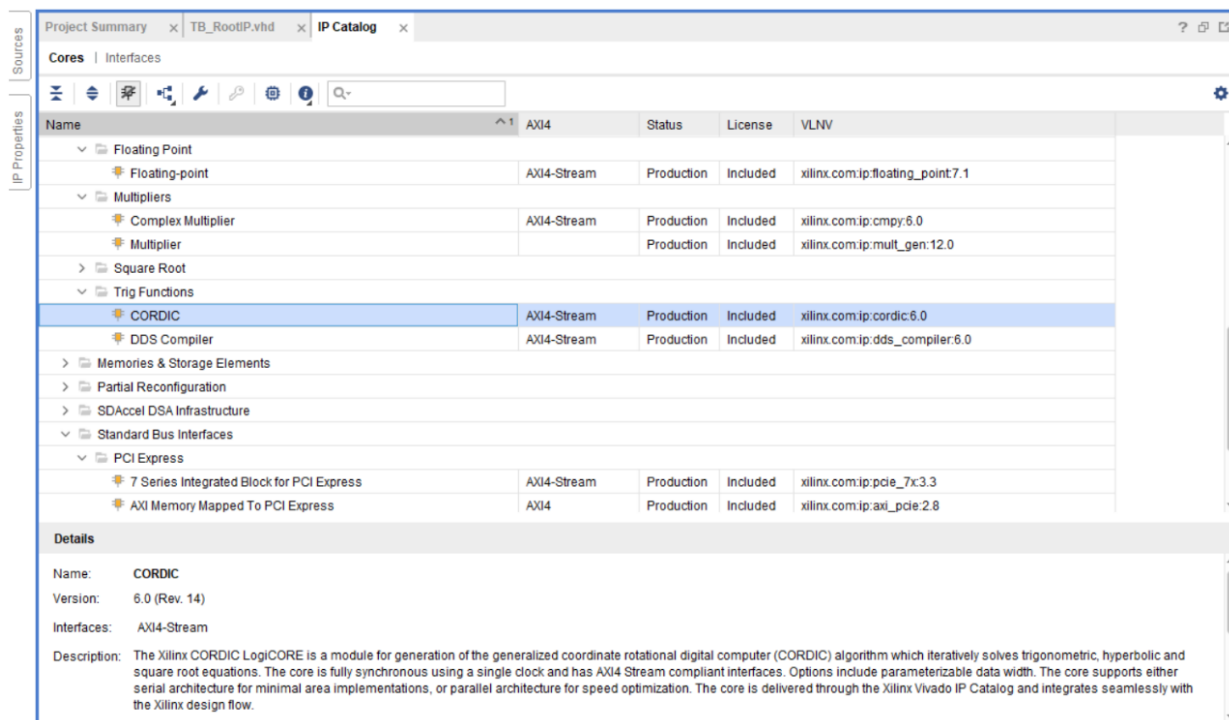
## Xilinx Vivado محیط IP Core های آماده در

با مراجعه به بخش IP Catalog در محیط نرم‌افزار Vivado می‌توان IP مورد نظر را انتخاب و پیکربندی کرد. برای این منظور IP Catalog را از منوی Window انتخاب کنید.



شکل ۲ مسیر دسترسی به IP Catalog

شکل زیر تعدادی از IP Core های موجود در محیط IP Catalog را نشان می‌دهد.



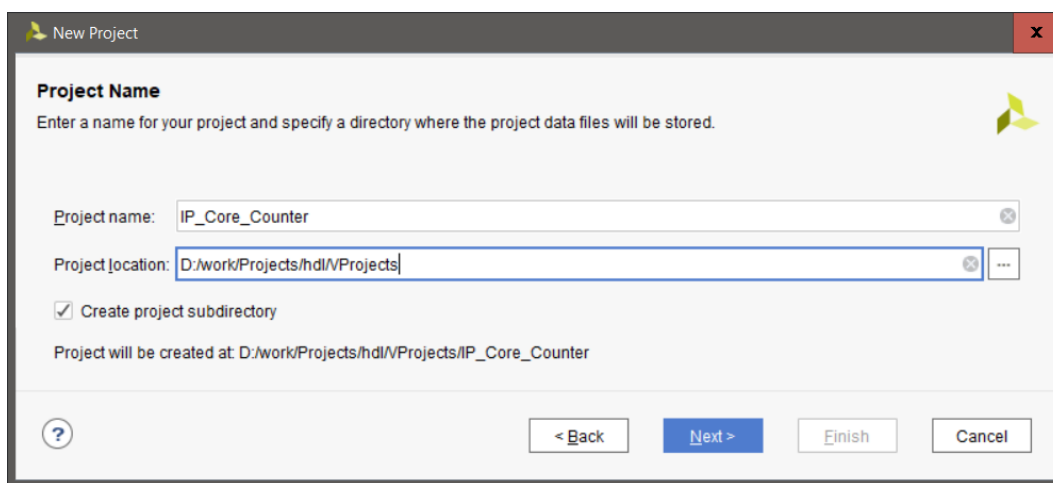
شکل ۳ تعدادی از IP Core های موجود در IP Catalog

## روند طراحی با استفاده از IP Core در محیط Vivado

### طراحی شمارنده‌ی چهار بیتی با استفاده از Binary Counter IP Core

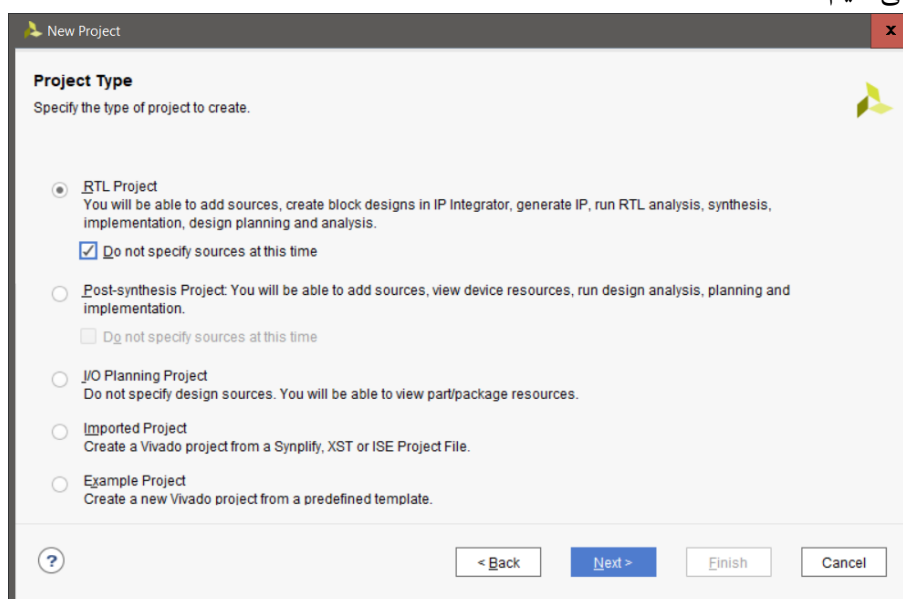
ایجاد پروژه‌ی جدید

- (۱) ابتدا یک پروژه جدید ایجاد می‌کنیم. برای این منظور از منوی File از بخش Project گزینه‌ی New... را انتخاب می‌کنیم.
- (۲) در پنجره‌ی باز شده (New Project) گزینه‌ی Next را انتخاب می‌کنیم.
- (۳) در پنجره‌ی جدید نام پروژه و مسیری که قرار است تا پروژه در آن ذخیره شود را مشخص می‌کنیم. سپس بر روی گزینه‌ی Next کلیک می‌کنیم.



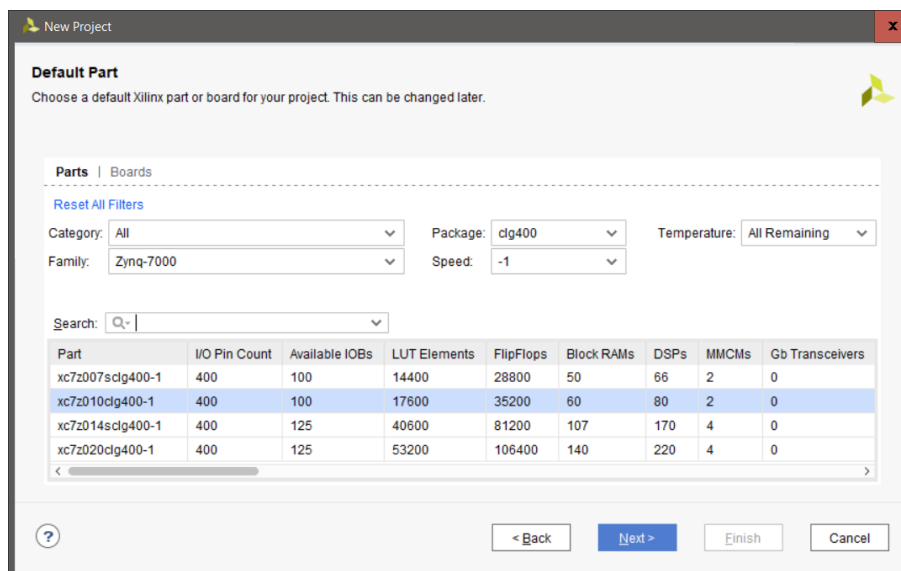
شکل ۴ انتخاب نام و مسیر ذخیره‌ی پروژه

- (۴) در بخش Project Type نوع پروژه را مشخص می‌کنیم. نوع پروژه را RTL Project انتخاب می‌کنیم. چون در حال حاضر نیازی به اضافه کردن فایل نداریم، گزینه‌ی Do not specify sources at this time را انتخاب می‌کنیم. سپس گزینه‌ی Next را انتخاب می‌کنیم.



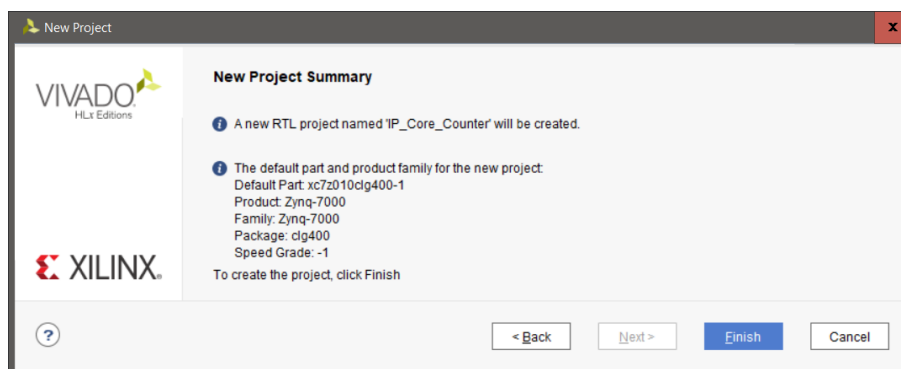
شکل ۵ انتخاب نوع پروژه

۵) در بخش Default Part تراشه‌ی مورد نظر را انتخاب می‌کنیم. البته می‌توان بعداً نیز آن را مشخص کرد. به‌عنوان مثال تراشه‌ی xc7z010clg400-1 را انتخاب کرده و بر روی گزینه‌ی Next کلیک می‌کنیم.



شکل ۶ انتخاب تراشه‌ی پیش‌فرض

۶) بخش New Project Summary خلاصه‌ای از اطلاعات پروژه را نشان می‌دهد. اگر اطلاعات نشان داده‌شده درست باشد بر روی گزینه‌ی Finish کلیک می‌کنیم تا پروژه مورد نظر ایجاد گردد. در غیر این صورت با بازگشت به مراحل قبل می‌توان آن را تغییر داد.

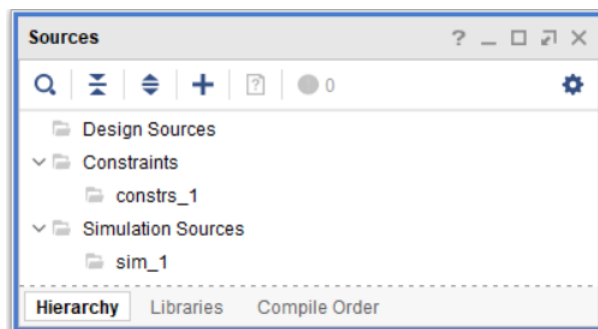


شکل ۷ خلاصه‌ای از اطلاعات پروژه جدید

۷) حال یک پروژه‌ی جدید ایجاد شده است.

#### محیط Sources

قبل از طراحی شمارنده ابتدا توضیح مختصری در رابطه با بخش Sources می‌دهیم. پس از ایجاد پروژه مطابق صفحه‌ی بعد بخشی با نام Sources وجود خواهد داشت.



شکل ۸ پنجره‌ی Sources

در این پنجره می‌توان از طریق **+** فایل ایجاد نمود و از بخش **⚙️** نوع تراشه را تغییر داد. منظور از Design Sources کدهای طراحی مانند کد VHDL و یا IP Core است. منظور از Constraints فایل‌هایی است که برای ملاحظات چینش، مسیریابی، تعیین پین و ... مورد استفاده قرار می‌گیرد. منظور از Simulation Sources نیز کدها و فایل‌هایی است که جهت درستی‌سنجی و طراحی محیط آزمون مورد استفاده قرار می‌گیرند. در ادامه هر کدام با جزئیات بیشتری بررسی خواهند شد.

#### ایجاد IP شماره‌ده

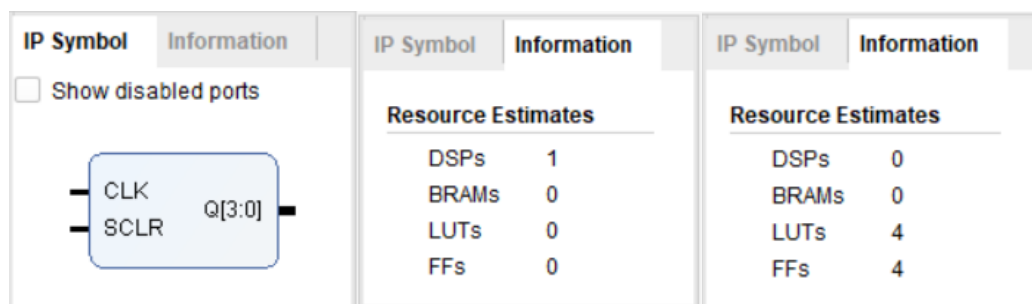
(۱) در پنجره‌ی IP Catalog در قسمت جستوجو کلمه‌ی Counter را جستوجو می‌کنیم. همان‌طوریکه در شکل زیر مشخص است یک IP با نام Binary Counter در نتایج جستوجو آمده است. در قسمت Details خلاصه‌ای از جزئیات آن آمده است.

(۲) با دوبار کلیک بر روی IP مورد نظر و یا با کلیک راست بر روی آن و انتخاب گزینه‌ی Customize IP و یا با کلیک بر روی **🔧** آن را سفارشی می‌کنیم.

(۳) در پنجره‌ی Customize IP نام IP و نوع آن را مشخص می‌کنیم.

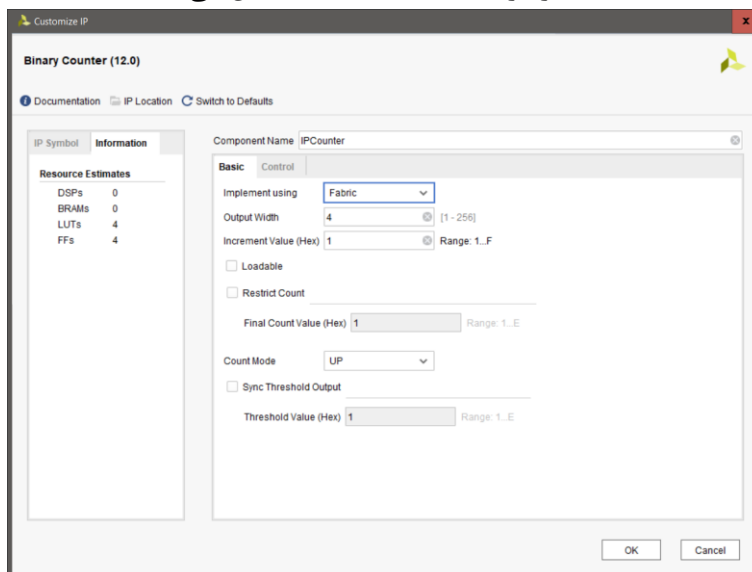
- در بخش Implementation Using می‌توان انتخاب کرد که برای پیاده‌سازی از واحدهای DSP یا LUT استفاده کند.
- در بخش Output Width تعداد بیت شماره‌ده را می‌توان انتخاب کرد و ...

برای طراحی شماره‌دهی چهار بیتی بالا شمار با Reset هم‌گام (Synchronous) تعداد بیت خروجی را برابر ۴ قرار می‌دهیم و در بخش Control گزینه‌ی Synchronous Clear را انتخاب می‌کنیم. در بخش سمت چپ شماتیک مدار و منابع مصرفی آمده است. جهت پیاده‌سازی از LUT ها استفاده می‌کنیم. (حالت Fabric)

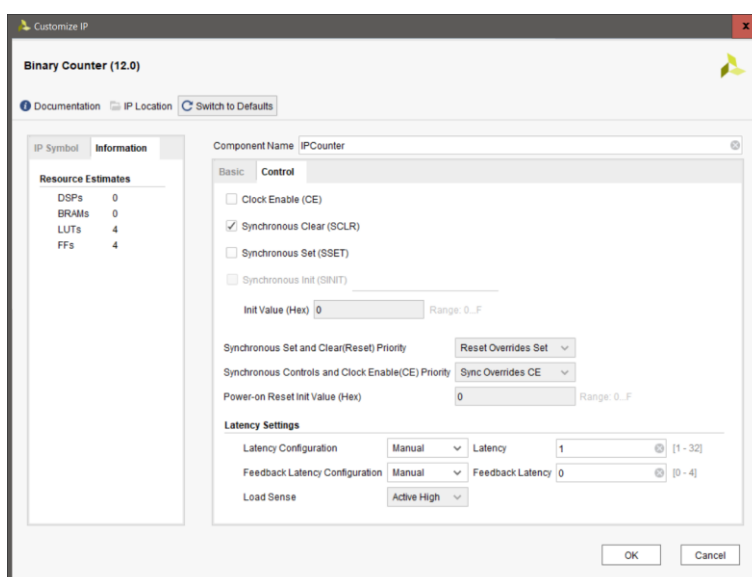


شکل ۹ شماتیک و منابع مصرفی شماره‌دهی چهاربیتی در دو حالت پیاده‌سازی با واحد DSP یا بلوک‌های منطقی

شکل‌های زیر نمایی از محیط Customize IP را برای Binary Counter نشان می‌دهد.

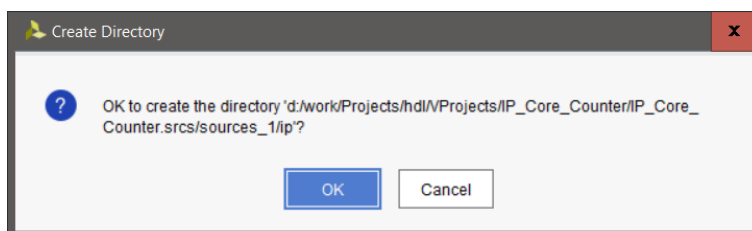


شکل ۱۰ محیط Customize IP برای Binary Counter



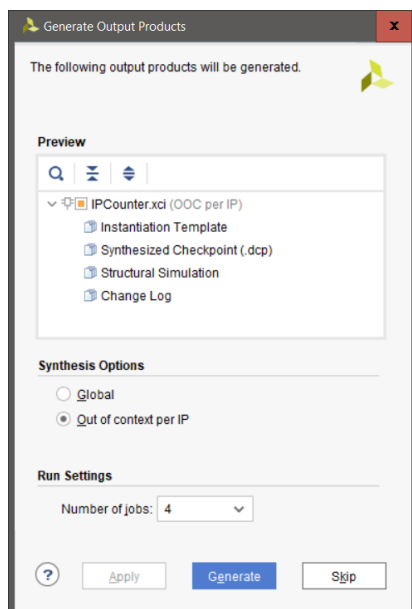
شکل ۱۱ محیط Customize IP برای Binary Counter

۴) پس از انتخاب پارامترهای پیاده‌سازی برروی گزینه‌ی OK کلیک می‌کنیم تا شمارنده تولید شود.



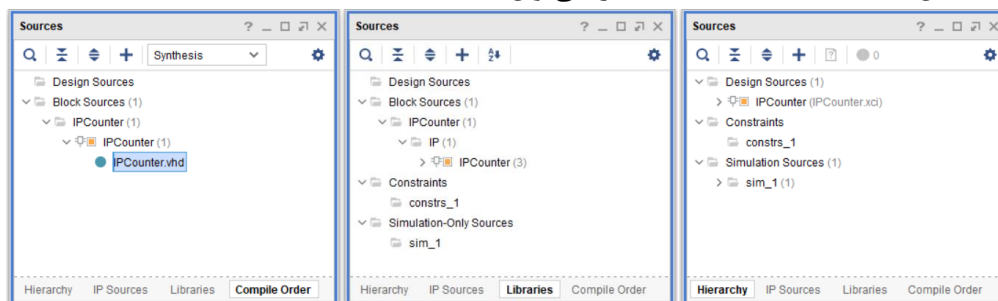
شکل ۱۲ پنجره‌ی ایجاد مسیر جهت تولید کدهای IP Core

۵) در پنجره‌ی Generate Output Product تعداد هسته‌ی پردازشی سیستمی که نرم‌افزار Vivado بر روی آن نصب است خواسته شده است. آن را نصف تعداد پردازنده‌ها قرار دهید. بر روی گزینه‌ی Generate کلیک کنید تا قالب نمونه‌سازی و شبیه‌سازی ساختاری نیز ایجاد گردد. در صورتی که پنجره‌ی دیگری نشان داده شود گزینه‌ی OK را انتخاب کنید.



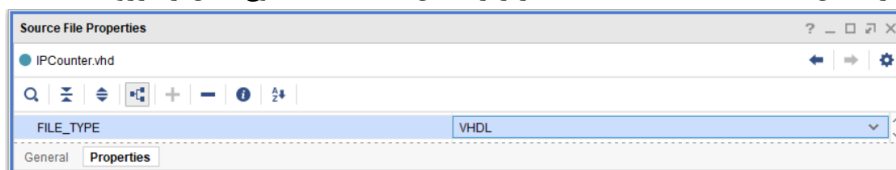
شکل ۱۳ محیط Generate Output Product

۶) با مراجعه به بخش Sources هسته‌ی ایجادشده را می‌توان یافت.

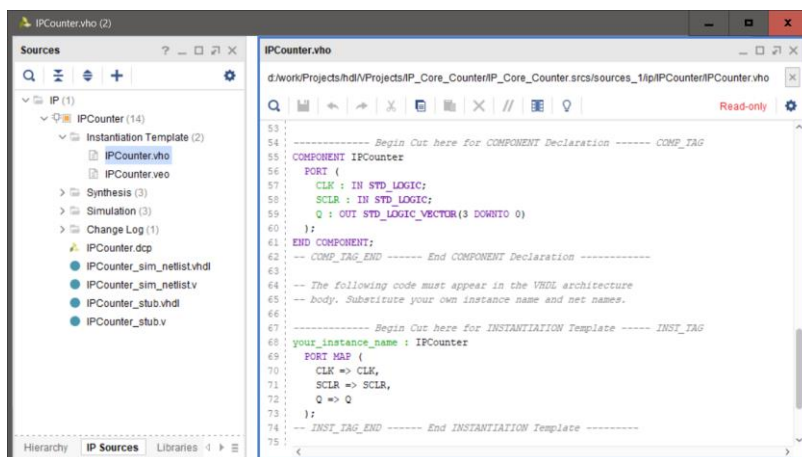


شکل ۱۴ هسته ایجادشده در پنجره Sources

۷) اگر در شکل فوق بر روی IPCounter.vhd کلیک کنید، کد تولیدشده برای شمارنده‌ی چهاربیتی را مشاهده خواهید کرد.  
۸) جهت استفاده از هسته‌ی تولیدشده، در زیربخش IP Sources از قسمت Instantiation Template فایل IPCounter.vho را انتخاب کنید. این فایل توصیف شمارنده را به صورت Component در اختیار قرار می‌دهد. اگر کد تولیدشده به زبان VHDL نیست از بخش Source File Properties از زیر بخش Properties نوع فایل را بر روی VHDL تنظیم نمایید.



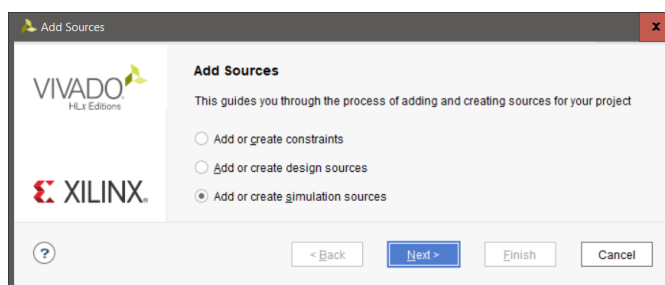
شکل ۱۵ پنجره Source File Properties



شکل ۱۶ کد Instantiation Template در فایل IPCounter.vho

ایجاد نمونه از شمارنده در محیط آزمون و شبیه‌سازی آن جهت نشان‌دادن درستی کارکرد مدار طراحی‌شده، یک فایل محیط آزمون ایجاد می‌کنیم و یک نمونه از شمارنده را در آن ایجاد کرده و شبیه‌سازی می‌کنیم.

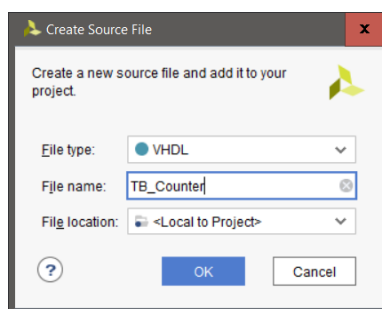
۱) جهت ایجاد فایل گزینه‌ی Add Sources را انتخاب می‌کنیم و برای ایجاد کد شبیه‌سازی مطابق شکل زیر انتخاب کرده و بر روی گزینه‌ی Next کلیک می‌کنیم.



شکل ۱۷ پنجره‌ی اضافه کردن فایل

۲) در پنجره‌ی جدید بر روی گزینه‌ی Create File کلیک می‌کنیم. نوع فایل، نام فایل و محل ذخیره‌ی فایل را مشخص می‌کنیم. سپس بر روی گزینه‌ی OK کلیک می‌کنیم. سپس گزینه‌ی Finish را انتخاب می‌کنیم.

در پنجره‌ی Define Module گزینه‌ی OK را کلیک می‌کنیم.



شکل ۱۸ ایجاد فایل جدید



(۳) جهت شبیه‌سازی قطعه کد زیر را به فایل TB\_Counter.vhd اضافه می‌کنیم.

```

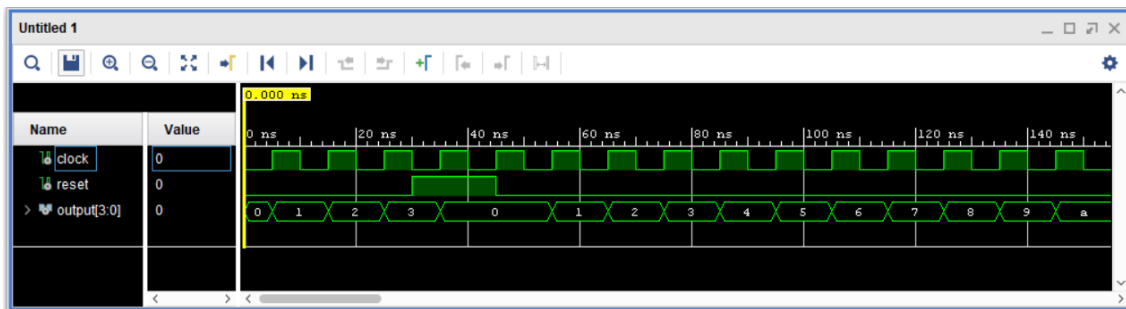
TB_Counter.vhd
D:\work\Projects\hdl\IP_Counter\IP_Counter.srcs\sim_1

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity TB_Counter is
5 end TB_Counter;
6
7 architecture Behavioral of TB_Counter is
8     COMPONENT IPCounter
9     PORT (
10         CLK : IN STD_LOGIC;
11         SCLR : IN STD_LOGIC;
12         Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
13     );
14     END COMPONENT;
15
16     signal clock : std_logic:= '0';
17     signal reset : std_logic:= '0';
18
19     signal output : std_logic_vector(3 downto 0);
20
21 begin
22     unit_under_test : IPCounter
23     PORT MAP (
24         CLK => clock,
25         SCLR => reset,
26         Q => output
27     );
28     clock <= not clock after 5 ns;
29     reset <= '1' after 30 ns, '0' after 45 ns;
30
31 end Behavioral;
32

```

شکل ۱۹ قطعه کد محیط آزمون

(۴) حال از بخش Simulation گزینه‌ی Run Simulation را انتخاب می‌کنیم و با کلیک بر روی Run Behavioral Simulation شکل موج خروجی را بررسی می‌کنیم.



شکل ۲۰ شکل موج حاصل از شبیه‌سازی شمارنده‌ی چهار بیتی

## پیاده‌سازی توابع سینوسی به روش الگوریتم CORDIC

### الگوریتم CORDIC

CORDIC چیست و چگونه کار می‌کند؟

الگوریتم CORDIC یا COordinate Rotation DIgital Computer روش کارآمدی برای محاسبات نمایی و مثلثاتی است. این الگوریتم ابتدا توسط Volder در سال ۱۹۵۹م. معرفی شد. الگوریتم کوردیک در قادر به محاسبه سینوس، کسینوس، تانژانت معکوس و ضرب و تقسیم با استفاده از عملیات شیفت و جمع  $a + b \times 2^i$  بود. در سال ۱۹۷۱م. Walther این الگوریتم را برای محاسبه‌ی توابع نمایی، لگاریتمی و محاسبه ریشه‌ی دوم اعداد تعمیم داد. الگوریتم CORDIC روشی سریع برای محاسبه‌ی حاصل ضرب یا توابع نمایی و مثلثاتی نیست، با استفاده از آن می‌توان محاسبات را فقط با استفاده از عملیات شیفت و جمع انجام داد و پیاده‌سازی آن به‌وسیله‌ی سخت‌افزار مناسب است.

### Xilinx CORDIC IP CORE

CORDIC نسخه‌ی ۶/۰

هسته‌ی CORDIC در نرم‌افزار Vivado قادر است عملیات زیر را انجام دهد.

- عملیات چرخش بردار (Vector Rotation)
- عملیات انتقال بردار (Vector Translation)
- محاسبات سینوسی و کسینوسی (Sin and Cos)
- محاسبات هایپربولیک سینوسی و کسینوسی (Sinh and Cosh)
- محاسبات معکوس تانژانت (ArcTan)
- محاسبات معکوس تانژانت هایپربولیک (ArcTanh)
- محاسبه‌ی ریشه‌ی دوم عدد (Square Root)

نحوه‌ی کار و مشخصات

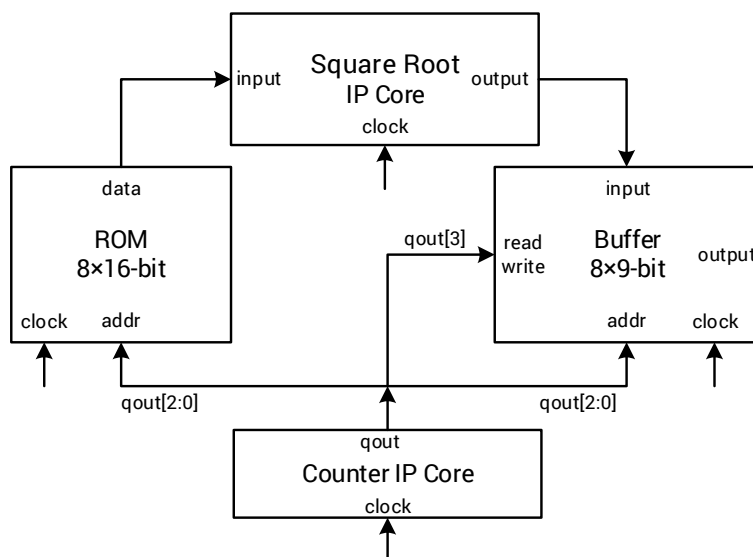
جزئیات نحوه‌ی استفاده از CORDIC IP Core و مشخصات آن در فایل راهنمای آن با نام pg105-cordic.pdf آمده است.

طراحی یک مدار محاسبه‌ی عملیات جذر

فرض کنیم می‌خواهیم مداری طراحی کنیم که عدد ۱۶ بیتی را به عنوان ورودی گرفته و جذر آن را محاسبه کند. با توجه به اینکه  $\varepsilon - 2^9 = \sqrt{2^{17} - 1} - 1$  در صورتی که بخواهیم به‌صورت تقریبی خروجی مشخص گردد به ۹ بیت نیاز داریم. بنابراین تعداد بیت خروجی را ۹ در نظر می‌گیریم

همچنین یک حافظه‌ی ۸ کلمه‌ای (۱۶ بیتی) با قابلیت خواندن همگام را در نظر می‌گیریم که اعداد در آن ذخیره شده‌اند. یک حافظه‌ی ۸ کلمه‌ای (۹ بیتی) جهت نوشتن نتایج در آن طراحی می‌کنیم. یک شمارنده نیز جهت تولید آدرس حافظه‌ها در هر کلاک ایجاد می‌کنیم. (از شمارنده‌ی چهاربیتی طراحی شده در قسمت قبل استفاده می‌کنیم). شکل مدار نهایی به صورت زیر خواهد بود.

شکل مدار نهایی به صورت زیر خواهد بود.



شکل ۲۱ شماتیک طرح

#### طراحی حافظه ROM

ابتدا یک فایل جدید با نام ROM8\_16.vhd ساخته و قطعه کد زیر را در آن می‌نویسیم.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.all;
4
5  entity ROM8_16 is
6      Port (
7          clk : in std_logic;
8          addr : in std_logic_vector(2 downto 0);
9          data : out std_logic_vector(15 downto 0));
10 end ROM8_16;
11
12 architecture Behavioral of ROM8_16 is
13
14     type rom_t is array (0 to 7) of std_logic_vector(data'range);
15     constant rom : rom_t :=(
16         X"04", -- 4
17         X"08", -- 8
18         X"09", -- 9
19         X"10", -- 16
20         X"20", -- 32
21         X"30", -- 48
22         X"F4", -- 244
23         X"FF"); -- 255
24
25 begin
26     process(clk)
27     begin
28         if (rising_edge(clk)) then
29             data <= rom(TO_INTEGER(unsigned(addr)));
30         end if;
31     end process;
32 end Behavioral;

```

شکل ۲۲ قطعه کد حافظه ROM

### طراحی حافظه‌ی Buffer8\_9

یک فایل جدید با نام Buffer8\_9.vhd ساخته و قطعه کد زیر را در آن می‌نویسیم.

```
buffer8_9.vhd
D:\work\Projects\hdl\IP_Cores\IP_Core_Counter\IP_Core_Counter.srcs\srcs_1\new\buffer8_9.vhd

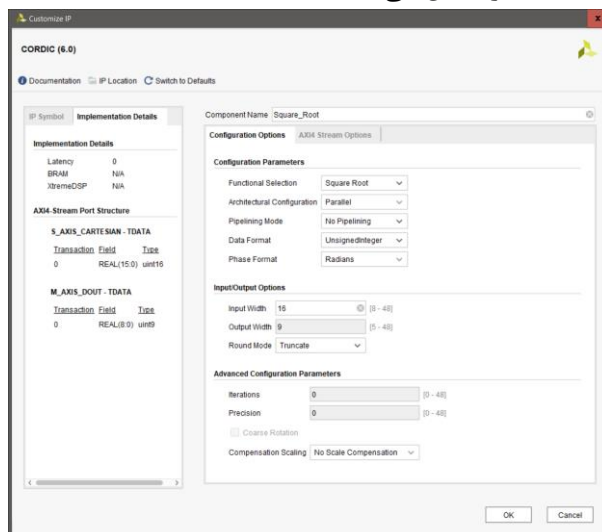
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.numeric_std.all;
4
5 entity Buffer8_9 is
6   Port (
7     clk : in std_logic;
8     read_write : in std_logic;
9     addr : in std_logic_vector(2 downto 0);
10    input : in std_logic_vector(8 downto 0);
11    output : out std_logic_vector(8 downto 0));
12 end Buffer8_9;
13
14 architecture Behavioral of Buffer8_9 is
15
16   type buffer_t is array (0 to 7) of std_logic_vector(output'range);
17   signal buffer8 : buffer_t;
18 begin
19
20   process(clk)
21   begin
22     if (rising_edge(clk)) then
23       if (read_write = '0') then
24         output <= buffer8(TO_INTEGER(unsigned(addr)));
25       else
26         buffer8(TO_INTEGER(unsigned(addr))) <= input;
27       end if;
28     end if;
29   end process;
30
31 end Behavioral;
```

شکل ۲۳ قطعه کد حافظه

### طراحی مدار جذرگیر با استفاده از CORDIC IP CORE

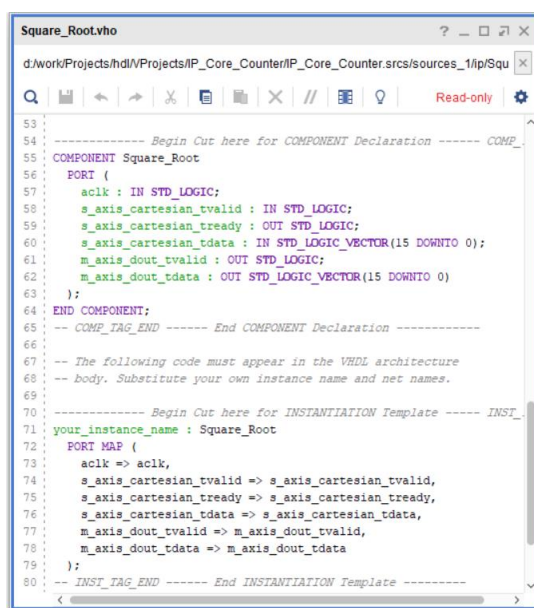
- ۱) در پنجره‌ی IP Catalog کلمه‌ی CORDIC را جست‌وجو می‌کنیم و بر روی آن دوبار کلیک می‌کنیم.
- ۲) از قسمت Functional Selection گزینه‌ی Square Root را انتخاب می‌کنیم.
- ۳) بخش Architectural Configuration مشخص می‌کند که داده‌های ورودی به چه صورتی وارد شوند. برای مدار جذر فقط حالت موازی امکان‌پذیر است.
- ۴) بخش Pipelining Mode این امکان را می‌دهد تا محاسبات به صورت خط‌لوله انجام شود. گزینه‌ی No Pipelining را انتخاب می‌کنیم. بنابراین در هر کلاک یک محاسبه انجام خواهد شد.
- ۵) در بخش Data Format می‌توان نوع داده‌ی ورودی را مشخص کرد. چون داده‌های ما همگی صحیح هستند، گزینه‌ی UnsignedInteger را انتخاب می‌کنیم.
- ۶) طول داده‌ی ورودی را ۱۶ قرار می‌دهیم. طول داده‌ی خروجی به صورت خودکار ۹ خواهد بود.
- ۷) بخش Round Mode نوع گرد کردن عدد را مشخص می‌کند. گزینه‌ی Truncate را جهت حذف تقریب استفاده می‌کنیم.
- ۸) در بخش AXI Stream Options حالت Flow Control را به صورت Blocking قرار می‌دهیم تا در هر کلاک یک عملیات کامل انجام شود. با انتخاب این حالت Latency مدار برابر یک خواهد شد.
- ۹) نام هسته را Square\_Root قرار می‌دهیم و بر روی گزینه‌ی OK کلیک کرده و همانند بخش قبل IP Core را تولید می‌کنیم.

شکل زیر نمایی از پنجره Customize IP را نشان می‌دهد.



شکل ۲۴ پنجره Customize IP CORDIC

در قسمت IP Sources فایل Square\_Root.vho را انتخاب می‌کنیم. نحوه‌ی استفاده از IP Core در فایل آمده است.



شکل ۲۵ کد Instantiation Template در فایل Square\_Root.vho

- aclk درواقع کلاک ورودی است.
- منظور از ورودی s\_axis\_cartesian\_tvalid این است که داده‌ی ورودی معتبر است.
- ورودی s\_axis\_cartesian\_tdata همان داده‌ی ورودی است.
- خروجی m\_axis\_dout\_tdata همان داده‌ی خروجی است.
- خروجی m\_axis\_dout\_tvalid نشان می‌دهد که داده‌ی خروجی معتبر است.

### طراحی مدار Top

جهت اتصال ماژول‌ها به یک دیگر، فایلی با نام TOP.vhd ایجاد می‌کنیم. سپس کد RTL مورد نظر را توصیف می‌کنیم.

```

11 architecture Behavioral of TOP is
12
13     COMPONENT Square_Root
14     PORT (
15         clk : IN STD_LOGIC;
16         s_axis_cartesian_tvalid : IN STD_LOGIC;
17         s_axis_cartesian_tready : OUT STD_LOGIC;
18         s_axis_cartesian_tdata : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
19         m_axis_dout_tvalid : OUT STD_LOGIC;
20         m_axis_dout_tdata : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
21     END COMPONENT;
22
23     COMPONENT IPCounter
24     PORT (
25         CLK : IN STD_LOGIC;
26         SCLR : IN STD_LOGIC;
27         Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
28     );
29     END COMPONENT;
30
31     COMPONENT ROM8_16
32     PORT (
33         clk : in std_logic;
34         addr : in std_logic_vector(2 downto 0);
35         data : out std_logic_vector(15 downto 0));
36     END COMPONENT;
37
38     COMPONENT Buffer8_9
39     PORT (
40         clk : in std_logic;
41         read_write : in std_logic;
42         addr : in std_logic_vector(2 downto 0);
43         input : in std_logic_vector(8 downto 0);
44         output : out std_logic_vector(8 downto 0));
45     END COMPONENT;
46
47     signal qout_address : STD_LOGIC_VECTOR(3 DOWNTO 0);
48     signal square_root_input : STD_LOGIC_VECTOR(15 DOWNTO 0);
49     signal square_root_output : STD_LOGIC_VECTOR(15 DOWNTO 0);
50
51
52 begin
53
54     counter_unit : IPCounter
55     PORT MAP (
56         CLK => clock,
57         SCLR => '0',
58         Q => qout_address
59     );
60
61     square_root_unit: Square_Root
62     PORT MAP (
63         clk => clock,
64         s_axis_cartesian_tvalid => '1',
65         s_axis_cartesian_tready => open,
66         s_axis_cartesian_tdata => square_root_input,
67         m_axis_dout_tvalid => open,
68         m_axis_dout_tdata => square_root_output);
69
70     rom8_16_unit: ROM8_16
71     PORT MAP (
72         clk => clock,
73         addr => qout_address(2 downto 0),
74         data => square_root_input);
75
76     buffer8_9_unit: Buffer8_9
77     PORT MAP (
78         clk => clock,
79         read_write => qout_address(3),
80         addr=> qout_address(2 downto 0),
81         input => square_root_output(8 downto 0),
82         output => buffer_output);
83
84 end Behavioral;
85

```

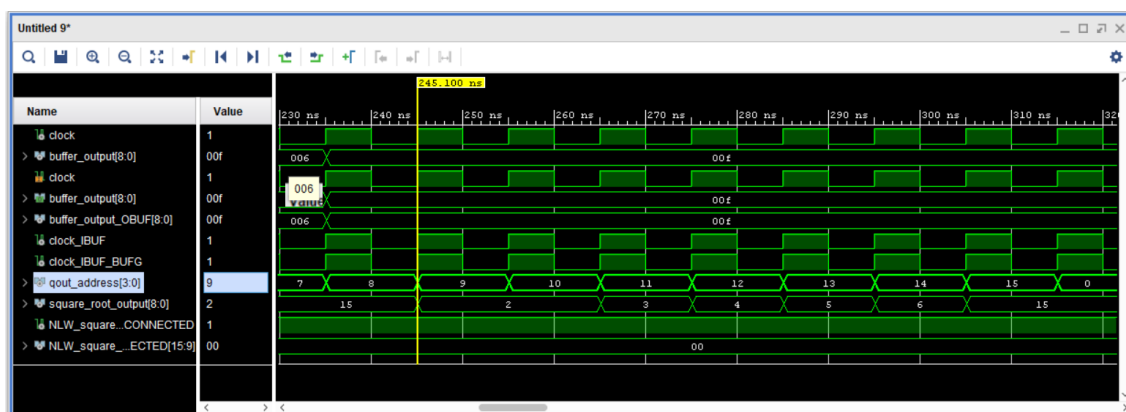
شکل ۲۶ کد توصیف ماژول TOP

### طراحی محیط آزمون و شبیه‌سازی طرح

فایل جدیدی با نام TB\_TOP.vhd ایجاد می‌کنیم و یک نمونه از ماژول TOP در آن ایجاد می‌کنیم.

جهت شبیه‌سازی طرح پس از سنتز، در بخش Simulation گزینه‌ی Run Simulation را انتخاب می‌کنیم و در ادامه، گزینه‌ی Run Post-Synthesis Functional Simulation را انتخاب می‌کنیم.

شکل زیر شکل موج طرح را نشان می‌دهد با توجه به محتوای ROM طراحی‌شده، عملیات جذرگیری به‌درستی کار می‌کند. جهت اضافه‌کردن سیگنال‌های داخلی ماژول TOP به شکل موج، در بخش Scope سیگنال را به‌صورت سلسله مراتبی پیدا کرده و بر روی آن کلیک راست کنید. سپس گزینه‌ی Add to Wave Window را انتخاب کنید.



شکل ۲۷ شکل موج نهایی طرح