Chapter-6

How to use IP Cores

Intellectual property (IP) is a general name that covers unique creations of human intellect, and mostly encloses copyrights, patents, and trademarks. The term IP is also valid in FPGA implementations ,i.e. called as IP cores. IP cores are configurable black boxes that can be added to main VHDL or Verilog based designs. Most of these cores were created by Xilinx. Some of them are allowed to be used free of charge while some of them requires charges. Moreover, third party IP cores also can be developed and presented to the other users with reasonable charges.

IP cores are classified with respect to their usage areas. Some of the classes are like below

- Basic Elements
 - Accumulators
 - Counters
 - Memory Elements
 - Registers, Shifters & Pipelining
- Communication & Networking
 - Error Correction
 - Ethernet
 - Modulation
 - Networking
 - Serial Interfaces
 - o Telecommunications
 - o Wireless
- Debug & Verification
- Digital Signal Processing
 - Building Blocks
 - o Filters
 - o Transforms
 - o Trig Functions
 - Waveform Synthesis
- Math Functions
 - Adders & Subctractors
 - \circ Conversions
 - o CORDIC
 - o Dividers
 - $\circ \quad Square \ Root$
 - Trig Functions.

There are many other IP cores that can be added to list. Aim of this chapter is to show how to use an IP core. A math function example, a clock manager example and a Read Only Memory (ROM) example is going to be demonstrated in deatil throughout the chapter. It's important to

state that Some of the IP cores aren't valid for some FPGA chips. Artix XC7A100T should be selected at beginning of each example below.

Example: Design a 4-bit unsigned adder circuit by using Adder/Subtracter IP Core. Simulate the IP core based design on ISim with respect to below two sample calculations.

$$\begin{array}{r} A_{3}A_{2}A_{1}A_{0} \\
 + B_{3}B_{2}B_{1}B_{0} \\
 \hline
 C_{out} S_{3} S_{2} S_{1} S_{0}
 \end{array}$$

Solution: Step-1) As the first step to achive example's task, create a new project and name it as IP_CORE_SAMPLE_1. Additionally, name of the main design should be "Sum_func". After that, **Adder/Subtracter** IP Core should be added to the main implementation files as a *component*. Right click to the FPGA chip and click **New Source**.



Figure 6-1

Choose IP (CORE Generator & Architecture Wizard) and name it as "Sum_function". Click Next.

>	New Source Wizard
Select Source Type Select source type, file name and its location.	
BMM File ChipScope Definition and Connection File Implementation Constraints File IP (CORE Generator & Architecture Wizard) Schematic User Document Verilog Module Verilog Module VHDL Hodule VHDL Library VHDL Dekcage VHDL Test Bench Embedded Processor	Ejle name: Sum_function Logation: C:\VP_CORE_SAMPLE_1\VP_CORE_SAMPLE_1\pcore_dir
More Info	Next > Cancel

Figure 6-2

Step-2) As seen in Figure 6-3, there are many IP Core classes to be used.

elect IP Create Coregen or	Architecture Wizard	d IP (N Core.	lew S	ource Wizard	Ŀ				
View by Function	View by <u>N</u> ame	1								
Name		-	Version	AXI4	AXI4-Stream	AXI4-Lite	Status	License	Vendor	Library
🗄 📂 Automotiv	e & Industrial									
🕀 💆 AXI Infrast	ructure									
🕀 💋 BaselP										
🕀 💋 Basic Elem	ents									
E Communi	cation & Network	ang								
Debug & V	erification									
	Processing									
Encoded	ures and Design									
Math Func	tions									
Hemories	& Storage Eleme	nts								
🗄 📂 Standard B	us Interfaces									
🗄 📂 📂 Video & In	nage Processing									
Search IP Catalog:										Clear
All IP versions							Only	y IP compa	tible with	chosen part
Please select IP										

Figure 6-3

Adder/Subtracter core is under the class of *Math Functions*. Choose it and click Next.

View by Function	View by <u>N</u> ame								
Name		*	Version	AXI4	AXI4-Stream	AXI4-Lite	Status	License	Vendc ^
🗄 📂 Digital Sigr	al Processing								
Embedded	Processing								
PGA Featu Math Func	tions								- 1
in Contract Punct	& Subtracters								
Ad	der Subtracter		11.0				Pre-Production		xilinx.
🕀 📂 Conver	sions								
🕀 📂 CORDI	с								
🕀 📂 Divider	s								
🕀 📂 Floatin	g Point								
🕀 📂 Linear /	Algebra Toolkit								
🕀 📂 Multipl	iers								
🕀 💋 Square	Root								
🗄 💋 Trig Fu	nctions								~
<									>
earch IP Catalog:									Clear

Figure 6-4

Click **Finish** as seen in the Figure 6-5.

	New Sour	ce Wizard		×
Froject Navigator will cre	ate a new skeleton source with the follow	ving specifications.		
Add to Project: Yes Source Directory: C:\IP_CORE Source Type: IP (CORE Gener Source Name: Sum_function.)	SAMPLE_1UP_CORE_SAMPLE_1Upcore_ stor & Architecture Wizard)	ġr		
Cure type: Adder Subtracter;	403016 II-0			
More Info		< <u>B</u> ac	k <u>F</u> inish	Cancel

Figure 6-5

Step-3) Figure 6-6 is emerged and core configuration should be done to be able use the IP core as planned. This Figure shows the default arrengement of the core from this graphical user interface (GUI). On the left hand side of the figure, block diagram of the core can be seen. Inputs of the core appear on the left hand side of the diagram while outputs appear on the right hand side of the diagram.

4	Adder Su	btracter	_ 🗆 🗙
Documents View			
IP Symbol & X	logi ^{CKRE} Add	er Subtracter	xilinx.com:ip:c_addsub:11.0
	Component Name Sum_fur	nction	
	Implement using	Fabric 🔹	
	A Input Type	Signed 💌	
	B Input Type	Signed 💌	
	A Input Width	15 Range: 2256	
	B Input Width	15 Range: 2256	
B[14:0]→ → S[15:0]	Add Mode	Add	
	Output Width	16 Range: 1516	
	Latency Configuration	Manual - La	tency 1 Range: 0258
	Constant Input	Constant Value	00000000000000 (Bin)
SCLR	Control		
SSET	Clock Enable (CE)		
	Carry In (C_IN)	Carry Out (C_OUT)	Borrow In/Out Sense Active Low 💌
	Synchronous Clear (SCLR)		
	Synchronous Set (SSET)		_
	Synchronous Init (SINIT)	Init Value	0 (Hex)
	Bypass	Bypass Sense	Active High 👻
	Synchronous Set and Clear(Res	et) Priority	Reset Overric 💌
V IP Symbol V Information	Datasheet		Generate Cancel Help

Figure 6-6

Aim of the example is to build 4-bit unsigned adder. Change Input type to unsigned and input width to 4. Check for the *Carry Out* option to be able to observe carry of the summation result. Moreover, set zero latency for simplicity during observation. As the configuration settings are changed inputs and outputs change. It's important to be aware of differences in between block diagrams of Figure 6-6 and Figure 6-7.

4	Adder Subt	racter	_ 🗆 🗙
Documents View			
IP Symbol 8 ×	logi CRE Adde	er Subtracter	xilinx.com:ip:c_addsub:11.0
	Component Name Sum_funct	ion	<u>*</u>
	Implement using	Fabric 💌	
	A Input Type	Unsigned 💌	
	B Input Type	Unsigned 💌	
	A Input Width	4 Range: 1256	
A[3:0] → C_OUT	B Input Width	4 Range: 1256	
B[3:0] → S[3:0]	Add Mode	Add	
ADD	Output Width	4 Range: 45	
	Latency Configuration	Manual 💌 Late	ency 0 Range: 0258
BYPASS	Constant Input	Constant Value	0000 (Bin)
SCLR	Control		
SINIT	Clock Enable (CE)		
	Carry In (C_IN)	Carry Out (C_OUT)	Borrow In/Out Sense Active Low *
	Synchronous Clear (SCLR)		
	Synchronous Set (SSET)		_
	Synchronous Init (SINIT)	Init Value	0 (Hex)
	□ Bypass	Bypass Sense	Active High 👻
	Synchronous Set and Clear(Reset)) Priority	Reset Overric 💌 💌
🌾 IP Symbol 💐 Information	Datasheet		Generate Cancel Help

Figure 6-7

Each IP core has its own datasheet. When **Datasheet** button is clicked a pdf file will be opened as seen in Figure 6-8. These datasheets exists for all IP cores and should be read carefully in order to use them effectively.



Figure 6-8

Click **Generate** button and IP core GUI will be closed. Configuration is done. If reconfiguration is needed double clicking to the core opens the GUI again. Adder/Subtracter core is added to design as seen in Figure 6-9.



Figure 6-9

Step-4) IP core is added to design but it isn't tied to the main implementation of "Sum_func".

Click **View HDL Instantiation Template** to be able to see port and component declarations of the generated IP core. This feature is presented by ISE in order to rapid component construction in main design.



Figure 6-10

Figure 6-11 shows the port and component declarations of the Adder/Subtracter IP core. As noticed, there are two 4-bit inputs and one 4 bit summation output and one bit carry output.

Eile Edit View Project Source Process Tools Wind	ow Layout	Help
- 『 「 「 「 」 『 『 』 『 『 』 『 『 『 『 『 『 『 『 『 『 『 『 『	887	🗟 🍌 🕾 🗉 🖻 🥬 🥙 🕨 🗵 📌 💡
Design ↔ □ ♂ ×	4 54	^
👔 View: 💿 🏧 Implementation 🔿 🎆 Simulation	F 55	Begin Cut here for COMPONENT Declar
J Hierarchy	56	COMPONENT Sum_function
C IP_CORE_SAMPLE_1	= 57	PORT (
🛄 🖻 🛄 xc7a100t-1csg324	≦ 58	<pre>a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);</pre>
B Sum_func - Behavioral (Sum_func.vhd)	3 59	<pre>b : IN STD_LOGIC_VECTOR(3 DOWNTO 0);</pre>
Sum_function (Sum_function.xco)	×2	c_out : OUT STD_LOGIC;
	61	s : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
	/ 62);
	% 63	END COMPONENT;
-	% 64	COMP_TAG_END End COMPONENT Declaration
	34 65	- The fellowing code must encour in the WHDT are
	- 66	the following code must appear in the vhos arc
No Processes Running	6	body. Subscitute your own instance name and ne
Processes: Sum function	0 69	Begin Cut here for INSTANTIATION TE
THE COPE Consister	70	your instance name : Sum function
Manage Cores	71	PORT MAP (
Regenerate Core	72	a => a,
Update Core to Latest Version	73	b => b,
U View HDL Functional Model	74	c out => c out,
View HDL Instantiation Template	75	s => s
	76);
	77	INST_TAG_END End INSTANTIATION Template
	< 70	>
> Start Int Design In Files In Libraries	E Design St	mmary 🛛 🖹 Sum func.yhd 🖾 🖹 Sum function yho 🔽
	Ocaign oc	
Console		+□♂×
		^ ^ ^
Started : "Launching ISE Text Editor to vie	w sum_func	tion.vno".

Figure 6-11

Arrange main implementation file as in PS 6-1.

```
library ieee;
use ieee.std_logic_1164.all;
entity Sum func is
  Port (a : in std_logic_vector (3 downto 0);
         b : in std_logic_vector (3 downto 0);
         c: out std_logic_vector (4 downto 0));
end Sum func;
architecture Behavioral of Sum func is
component Sum_function
 port (
  a : in std_logic_vector(3 downto 0);
  b : in std_logic_vector (3 downto 0);
  c out : out std logic;
  s : out std_logic_vector (3 downto 0));
end component;
begin
u1: Sum function
 port map (
  a \Rightarrow a,
  b \Rightarrow b,
  c_{out} => c(4),
  s \Rightarrow c(3 \text{ downto } 0));
end Behavioral;
```

When PS 6-1 is completed IP Core – which is named as "u1-Sum_function" becomes a sub unit of the main implementation design of "Sum_func".



Figure 6-12

Step-5) Implementation is completed. Now, simulation scenario should be set. Open a VHDL Test Bench file and named it as "Summation_func_TB" and click **Next**.

>	New Source Wizard
Select Source Type Select source type, file name and its location.	
BMM File ChipScope Definition and Connection File Implementation Constraints File IP (CORE Generator & Architecture Wizard) MEM File Schematic User Document Verilog Test Fixture VHDL Module VHDL Library VHDL Test Bench Embedded Processor	Elle name: Summation_func_TB Logation: C:\UP_CORE_SAMPLE_1\UP_CORE_SAMPLE_1
More Info	Next > Cancel

Figure 6-13

In the next window of Figure 6-14, associate test bench with main implementation file of "Sum_func" and click **Next**.

File Edit View Project Source Process Tools	Window Layout Help)		0	
2 □				×
Wext: 	Sum_function	Inew Source Wize	ra	
Start wit Design Thes Libraries				_
Concele Concernation Concernation	More Info		< gad. Mext > Cancel	

Figure 6-14

Arrange test bench as in PS 6-2.

```
library ieee;
use ieee.std_logic_1164.all;
entity Summation func TB is
end Summation func TB;
architecture behavior of Summation func TB is
  component Sum func
  port(
     a : in std logic vector(3 downto 0);
     b : in std logic vector(3 downto 0);
     c : out std_logic_vector(4 downto 0)
    );
  end component;
 --Inputs
 signal a : std logic vector(3 downto 0) := (others => '0');
 signal b : std logic vector(3 downto 0) := (others => '0');
--Outputs
 signal c : std_logic_vector(4 downto 0);
begin
 uut: Sum func port map (
      a \Rightarrow a.
      b \Rightarrow b,
      c \Rightarrow c
    );
 stim_proc: process
 begin
   a <= "0001"; b <= "0010";
   wait for 100 ns;
   a <= "1001"; b <= "0111";
   wait for 100 ns;
  end process;
end;
```

PS 6-2

Figures 6-15 and 6-16 shows two different summation operations. As a result, Adder Subtracter block is used successfully.



Figure 6-15



Figure 6-16

Example: Increase the clock rate of the FPGA from 100 MHz to 300 MHz. Observe the result in ISim.

Solution: Step-1) Open a new project and create a main VHDL design, name it as "Clock_Manager". This design should have an input port for 100 MHz clock and a output port for 300 MHz clock signal. Clocking Wizard IP core is suitable for such an aim. Add a new source as represented in Figure 6-2 and name it as "clock_manager_core". After typing core name click Next. Click View By Name option in the window as seen in Figure 6-17 and choose Clocking Wizard IP core. Click Next.

Clocking Wizard inherits choices of Phase Locked Loop (PLL) and Mixed Mode Clock Manager (MMCM). The PLL is an analog clock management cell that can generate different phases of clock, does clock division and de-skew a clock. Moreover, it can generate different frequencies at the same time and has better jitter performance with respect to digital clock manager (DCM). MMCM cell is a simply PLL cell that is modified with DCM features. Since DCM has more precisie phase shifting ability, analog and digital managers are used together.

View by Function View by Name					_
Name	Version	AXI4	AXI4-Stream	AXI4-Lite	Sti 1
Chroma Resampler	3.00.a		AXI4-Stream	AXI4-Lite	Pn
CIC Compiler	2.0		AVIA Character		Pri
Clock Forwarding / Board Deskew (DCM)	3.0 13.1		AAI4-Stream		PR
Clock Forwarding / Board Deskew (DCM) ADV)	13.1				
Clock Forwarding / Board Deskew (DCM_SP)	13.1				
R Clock Switching with Two DCM_SPs	13.1				
R Clock Switching with Two DCMs	13.1				
🍯 Clocking Wizard	 3.6				Pre
Color Correction Matrix	2.0				
Color Correction Matrix	5.00.a		AXI4-Stream	AXI4-Lite	Pri
Color Filter Array Interpolation	5.0		AVIA-Stream	A VIA-Lite	Dr
Complex Multiplier	3.1		AAI4 Stream	AVIA FILE	Prix
<					>
Search IP Catalog:				d	ear
				the second second	

Figure 6-17 12

Clocking Wizard core includes six pages for configuration. GUI from Figure 6-18 to 6-21 shows required settings for the example's aim. Since Nexys 4 DDR board has 100 MHz oscillator, check the primary input clock value as seen in below Figure 6-18. Click Back and Next buttons to pass from page to page.

9		Clo	cking Wizar	d				>
Documents View								
P Symbol	= ×	gi CXRE	C	locking W	izard		xilinx.co	m:ip:clk_wiz:3
CLK_IN1 -> CLK_OUT1								
CLK_OUT1	CE						Clocki	ng Featu
	CLR	Component name:		clock	_manager_core		/ 1	nput Clo
							/ -	iput cio
E CLK_OUTZ	CE	- Clocking Feature	5					
	CLR		thesis					
		 Frequency syn 	unesis					
E-CLK_OUT3	CE	Spread Spect	rum					
E	CLR	Phase alignm	ant (known r	hase relationship to inc	ut clock)			
CLKFB_IN_N> CLK_OUT4		I♥ Filase alignini	enic (known p	nase relationship to hip	ut clock)			
I CLK_OUT4	CE	Minimize pow	er			Primitive		
RESET CLK_OUT4	CLR							-Input Jitter
POWER_DOWN		Dynamic phas	e shift			MMCME2_ADV C PLL	E2_ADV	• UI C
CLK_OUTS	CE	Dynamic reco	nfiguration (i	n system output freq m	odification)			
CLK_OUT5	CLR	- litter Optimizat	ion					
DADDR[6:0]		Jitter Optimizat	ION					
DCLK	CE	Balanced Balanced Alignment Align Alignment Align Alignment Alignment Alig						
DEN	CLR	C Minimize ou	tout iitter (lo	w clock jitter filtering)				
DIN[15:0]		C Maximize in	nut jitter filte	ring (allow larger input	litter)			
DOUT[15:0] CLK_OUT7	CE	- meaning of the	par juter me	ang (alow larger input	Jucci			
DRDY - CLK_OUT7	CLR	- Input Clock Inform	mation					
DWE		input crock inter						
> INPUT_CL	STOPPED	Trans Chart	Inp	ut Freq (MHz)	Transfer The	C		
> CLKF8_ST	PPED	Input Clock	Value	Valid Range	Input Jitter	Source		
PSCLK		primary	100.000	10.000 - 800.000	0.010	Finale anded clock canable nin	-	
PSEN		printidry	100.000	10.000 - 800.000	0.010 3	single ended clock capable pin	•	
PSINCDEC	•	· · ·)
PSDONE (· .	atachoot			and the second	et a la anno la anno la	Canad	1 Unio

Figure 6-18

Arrange requested output frequency to 300 MHz liken in Figure 6-19.

	C	locking Wiza	rd						
5×	PE								
-	Logiczin		JOCK	ng wiz	zard			xilinx.o	om:ip:clk_wiz:
CLK_OUT1									Outou
CLK_OUT1_CE									Outpu
CLK_OUT1_CLR									Cloc
CLK_OUT2	The phase is calc	ulated relative	to the active	input clock.					Setting
CLK_OUT2_CE		Output Fr	ea (MHz)	Phase (d	earees)	Duty Cy	de (%)		lleo
CLK_OUT2_CLR	Output Clock	Boguested	Actual	Poguostod	Actual	Boguested	Actual	Drives	Fine Ps
CLK_OUT3		Requested	Actual	Requested	Actual	Requested	Actual		
CLK_OUT3_CE	CLK_OUT1	300.000	300.000	0.000	0.000	50.000	50.0	BUFG -	
CLK_OUT3_CLR	CLK OUT2	100.000	N/A	0.000	N/A	8 The requi	octod duby	avela for the 1ct o	utput clock
CLK_OUT4	-			-		Inerequ	esteu uuty	cycle for the 1st c	
CLK_OUT4_CE	I CLK_OUT3	100.000	N/A	0.000	N/A	50.000	N/A	BUFG -	'
CLK_OUT4_CLR	CLK_OUT4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG -	Г
CLK_OUT5		100.000		0.000		50.000		0.050	
CLK_OUT5_CE	CLK_0015	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	· ·
CLK_OUTS_CLR	CLK_OUT6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG 👻	
		100.000	NI/A	0.000	NI/A	50.000	N/A	DUEC -	
CLK_OUTB_CLB		100.000	N/ A	0.000	N/A	50.000	N/A	BUFG	
INPUT CLK STOPPED									
CLKFB STOPPED									
LOCKED									
	•								
•							1	1	
			Clocking Wiza	Øx Øx Cuckourn Clocking Wizard cuckourn Cuckourn cuckourn Cuckourn Cuckourn Cuckourn	Clocking Wizard	Org Org Cuckouri cuckouri cuckouri, cui cuckouri, cui cuckouri cucko	Clocking Wizard	Clocking Wizard	Oxform Output Cock Output Freq (MHz) Phase (degrees) Duty Cycle (%) Drives cuc,ourn cuc,our

Figure 6-19

Reset and *Locked* features can be added to the design. Click on the check boxes to set them. *Reset* port is the input of the core while *Locked* is shown as output of the core in Figure 6-20. *Locked* output will be logic-1 when input and output clock signals are phase aligned.



Figure 6-20

Don't change anything in page 4 and 5. Figure 6-21 shows the last page. This page shows the multiplier and divider settings to reach 300 MHz clock from 100 MHz. Multiplier coefficient is set as 10.125 while divider is set to 3.375 where



$100MHz * \frac{10.125}{3.375} = 300 MHz.$

Figure 6-21 14

Configuration is done. Click Generate button.

Step-2) Arrange the main code as given in PS 6-3.

library ieee; use ieee.std_logic_1164.all; entity Clock Manager is **Port** (clk 100MHz,reset : in std logic; clk 300MHz,clock locked :out std_logic); end Clock_Manager; architecture Behavioral of Clock Manager is component clock manager core port (CLK IN1 : in std_logic; CLK OUT1 : out std_logic; RESET : in std_logic; LOCKED : out std logic); end component; begin u1: clock manager core port map (CLK IN1 => clk 100MHz, CLK OUT1 => clk 300MHz, RESET = reset, LOCKED => clock locked); end Behavioral;

PS 6-3

Step-3) VHDL implementation is completed. Now simulation can be done. Use PS 6-4 for simulation. Open a VHDL Test Bench file and named it as "Clock_Manager_TB".

```
library ieee;
use ieee.std logic 1164.all;
entity Clock Manager TB is
end Clock_Manager_TB;
architecture behavior of Clock Manager TB is
  component Clock Manager
  port(
     clk 100MHz : in std logic;
     RESET : in std logic;
     clock locked : out std logic;
     clk 300MHz: out std logic
    );
  end component;
 --Inputs
 signal clk 100MHz : std logic := '0';
 signal RESET : std logic := '0';
 --Outputs
 signal clock locked : std_logic;
 signal clk 300MHz : std logic;
 -- Clock period definitions
 constant clk 100MHz period : time := 10 ns;
begin
 uut: Clock Manager port map (
     clk 100MHz \Rightarrow clk 100MHz,
     RESET => RESET,
     clock locked => clock locked,
     clk 300MHz \Rightarrow clk 300MHz);
 clk 100MHz process :process
 begin
       clk 100MHz \leq 0';
       wait for clk 100MHz period/2;
       clk 100MHz <= '1';
       wait for clk 100MHz period/2;
 end process;
 stim proc: process
 begin
    reset <= '1';
   wait for 100 ns;
   reset <= '0';
   wait;
 end process;
end;
```

```
PS 6-4
```

Figure 6-22 shows the result of the simulation. Phases of 100 MHz and 300 MHz clocks become same when *clock locked* becomes '1' at approximately at $1,3 \mu s$.





Figure 6-23 demonstrates a zoomed version of the resulting waves. In this figure, period of the output is measured by using cursor feature of the ISim. Period of 300 MHz clock output is approximately 3.3 ns. This result shows that **Clocking Wizard** IP core is used correctly.



Figure 6-23

Example: Design a ROM that hold the values from 0 to 15, in 4-bit format. Read and show the values of ROM via simulation on ISim.

Solution: Step-1) Open a new project and create a main VHDL design, name it as "ROM_Usage". Add a new source as represented in Figure 6-2 and name it as "ROM_core". After typing core name click Next. Choose Distributed Memory Generator IP core as seen in Figure 6-24. Click Next.

View by Function View by M	lame									
Name	<u>م</u>	Version	AXI4	AXI4-Stream	AXI4-Lite	Status	License	Vendor	Library	^
Combedded Processing FPGA Features and De Math Functions Memories & Storage E FIFOs Memory Interface Manory Interface RAMS & ROMs Stork Memory	sign Iements Generators	72	ΔΧΙΔ		۵XI4-I ite	Pre-Production		viliny com	in	1
Distributed Me	mory Generator	7.2	AVIT		AAH LILE	Pre-Production		xilinx.com	ip	
Standard Bus Interface DisplayPort PCI PCI PCI Express RapidIO Spdif	5	1.1		AXI4-Stream	AXI4-Lite	Beta	8	xilinx.com	ip	~
Search IP Catalog							-			Clear
All IP versions								Only	IP compatible wit	h chosen part

Figure 6-24

Distributed Memory Generator IP core settings consists of three pages. In the first page, *Depth* and *Data Width* should be set as 16 and 4, respectively. As stated in the question, memory type should be set as ROM.

4		Dis	tributed Memory Generator	_ 🗆 🗙
⊻iew Documents IP Symbol	ē×	Logi CXRE	Distributed Memory Generator	xilinx.com:ip:dist_mem_gen:7.2
a[3:0] → d[3:0] → dpra[3:0] →	→ spo[3:0] → qspo[3:0] → dpo[3:0]	Component Name re Options Depth 16 Data Width 4	ead_from_rom Range: 1665536 Range: 11024	^
spirit2-07 i_ce		ROM ROM Dual Port RAM Simple Dual Port	C Single Port RAM C SRL16-based Memory	
qapo_srst> qapo_srst> qdpo_srst>				
			< Back Page 1 of 3 Next > Generation	rate <u>C</u> ancel <u>H</u> elp

Figure 6-25

In the second page change nothing and click Next.

ew Documents Symbol &	×		
	logic RE Distri	buted Memory Generator	xilinx.com:ip:dist_mem_gen:
	Input Options		
	 Non Registered 	C Registered	
a[3:0]	☐ Input Clock Enable	C Qualify WE with I_CE	
d[3:0] → qspo[3:0]	- Dual Port Address		
spra[3:0]	Non Registered	C Registered	
	- Output Ontions		
we	G Nee Desistered	C Paristand C Pat	
cik —>	Common Output CLK	Registered Single Part Output C	n
qdpo_olk		Dual Port Output CE	•
qspo_rst →		, buan or ouput ee	
dabo_rat →	Pipelining Options		
qdpo_srst	Pipeline Stages: 0		
		< Back Page 2 of 3 Next > Gen	erate <u>C</u> ancel <u>H</u> elp

Figure 6-26

Figure 6-27 shows the last page of the GUI. In this page, values of the ROM should be stated. ROM values can be inserted by a Coefficients File.

4	Distributed Memory Generator 🛛 🗕 🗆 🗙
View Documents IP Symbol 5	Distributed Memory Generator xdimx.com:ip:dist_mem_gen:7.2
a(3.0) → spo(3.0) d(3.0) → spo(3.0) d(a,0) → spo(3.0)	Load COE File If desired the initial memory content can be set by using a COE file. This will be passed to the core as a Memory Initialisation File (MIF). Coefficients File : no_coe_file_loade
spra[2] → → dpo[2] 0] spra[2] → → dpo[2] 0] (sp → + + + + + + + + + + + + + + + + + +	COE Options Default Data : 0 Radix : 16 2 Reset Options 10 16 16 16 If Reset QSPO If Reset QDPO 16 16 16
qdpo_sk →→ qspo_srst →→ qspo_srst →→ qdpo_srst →→	Synchronous Reset QSPO Synchronous Reset QDPO CE Overrides Sync Controls Sync Controls Overrides CE
	< <u>B</u> ack Page 3 of 3 <u>Next > Generate Gancel H</u> elp

Figure 6-27

Coefficients file construction can be done in MATLAB. Below MATLAB script create values from 0 to 15 in base 10.

clc;clear all;close all; fid = fopen('rom_values.coe', 'wt'); fprintf(fid, sprintf('memory_initialization_radix=10;\n\n')); fprintf(fid, sprintf('memory_initialization_vector=\n\n')); for i = 1:16 fprintf(fid, sprintf('%d,\n',i-1)); end fclose(fid);

PS 6-5

Click **Browse** button to upload generated *.coe file into ROM.

iew Documents			
Symbol	e ×	Logic PE Distributed Memory Generator	xilinx.com:ip:dist_mem_gen:7
a[3:0]→ d[3:0]→	→ spo[3:0] → qspo[3:0]	Load COE File If desired the initial memory content can be set by using a COE fil Coefficients File : e/rom_values.coe growse Show	e. This will be passed to the core as a Memory Initialisation File (MIF).
spra(3.0) → i_ce → qspo_ce → we →		COE Options Default Data : 0 Reset Options	Radix : 10 •
ck			
qspo_stat		CE Overrides Sync Controls	C Sync Controls Overrides CE
			-

Figure 6-28

"rom_values.coe" file is loaded into ROM. Uploaded values can be observed by clicking **Show** button. Figure 6-29 shows the values of each index.

JOE TEELOTT	memory_mudizadi	on_vector	
Index	Value		
0	0		
1	1		
2	2		
3	3		
4	4		
5	5		
6	6		
7	7		
8	8		
9	9		
10	10		
11	11		
12	12		
13	13		
14	14		
15	15		

Figure 6-29

Step-2) Arrange the main code as given in PS 6-6.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std logic arith.all;
use ieee.std logic unsigned.all;
entity ROM Usage is
  Port ( clk : in std_logic;
         data : out std_logic_vector (3 downto 0));
end ROM Usage;
architecture Behavioral of ROM Usage is
component ROM core
 PORT (
  a : in std_logic_vector(3 downto 0);
  spo : out std_logic_vector(3 downto 0)
 );
end component;
signal address : std logic vector(3 downto 0):="0000";
begin
 u1 : ROM core
 port map (
  a \Rightarrow address,
  spo \Rightarrow data);
 process(clk)
 begin
      if (rising_edge(clk)) then
        address <= address +1;
      end if:
 end process;
end Behavioral;
```

PS 6-6

Step-3) VHDL implementation is completed. Now, simulation can be done. Use PS 6-7 for simulation. Open a VHDL Test Bench file and named it as "ROM_Usage_TB".

```
library ieee;
use ieee.std_logic_1164.all;
entity ROM Usage TB is
end ROM_Usage_TB;
architecture behavior of ROM Usage TB is
  component ROM Usage
  port (
     clk: in std logic;
     data : out std logic vector(3 downto 0)
    );
  end component;
 signal clk : std_logic := '0';
 signal data : std_logic_vector(3 downto 0);
 constant clk period : time := 10 ns;
begin
 uut: ROM_Usage port map (
     clk => clk,
     data => data
    );
 clk process :process
 begin
       clk <= '0';
       wait for clk period/2;
       clk <= '1';
       wait for clk period/2;
 end process;
end;
```

PS 6-7

Simulation scenario is constructed such that addresses of the ROM are read at each clock cycle. Figure 6-30 shows the result of reading operation on the output port of *data*.



Figure 6-30