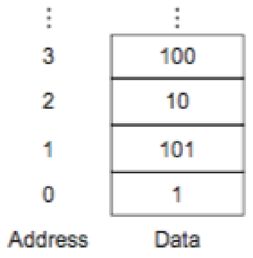
Tutorial 1 : Block RAM Simulation

Chanon Khongprasongsiri, Pinit Kumhom
Department of Electronic and Telecommnication Engineering
King Mongkut's University of Technology Thonburi
Bangkok, 10140
`{chanon.khong, pinit}@mail.kmutt.ac.th`

1 A Brief Review of Memory



The figure 4 shows the memory address which is a value that delineate the location of a specific data element and contents of memory. First of all, we need to write some data to the memory array. In the figure, we write 1 to address 0, 101 to address 1 and so on. The most often operation after this should be read the memory. For example, the address of the third data element is 2, and the value of the memory[2] is 10.

2 What is Block RAM

In Xilinx FPGA, a Block RAM is a two port memory which containing several kilobits of RAM. It can be implement in form of Random Access Memory (RAM), Read Only Memory (ROM), and First In First Out (FIFO) buffer while support Error Correcting Coding (ECC).

Each Block RAM can store up to 36Kb of information, and may be configured either as one 36Kb RAM, or two independent 18Kb RAMs. The default word size is 18 bits, and in this configuration each RAM comprises 2048 memory elements. The RAM can also be 'reshaped' such that it contains more, smaller elements (for example 4096 elements×9 bits, or 8192×4 bits), or alternatively, fewer, longer elements (e.g. 1024 elements×36 bits, 512× 72 bits). Larger capacity memories can be formed by combining two or more Block RAMs together.

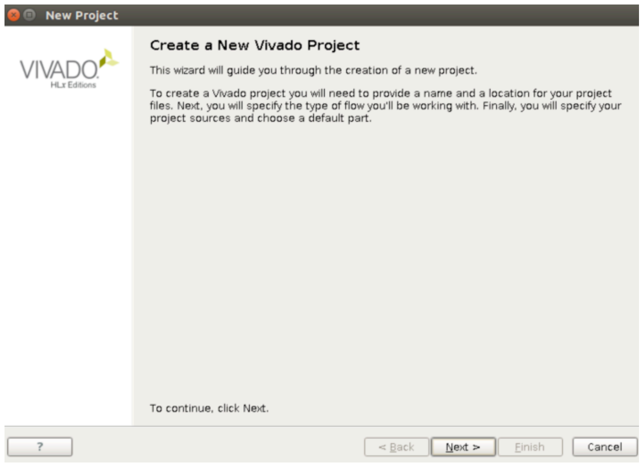
3 A First Design of Block RAM

The objective here is simulation the characteristic of Block RAM using Vivado. For the Block RAM, we write some data and then read the data from Block RAM. The benefit of using Block RAM instead of array variable type is it provide a number of access different configurations (single, dual port), which have different memory configurations.

1. First of all, we need to open the Vivado program
   - For Window users, just double click on the Vivado icon.
   - For Ubuntu users, open the terminal and type vivado and press enter.

```
vdesi@ubuntu: $> vivado
```

2. Press create new project, it will show the new project guide wizard. Then press next.



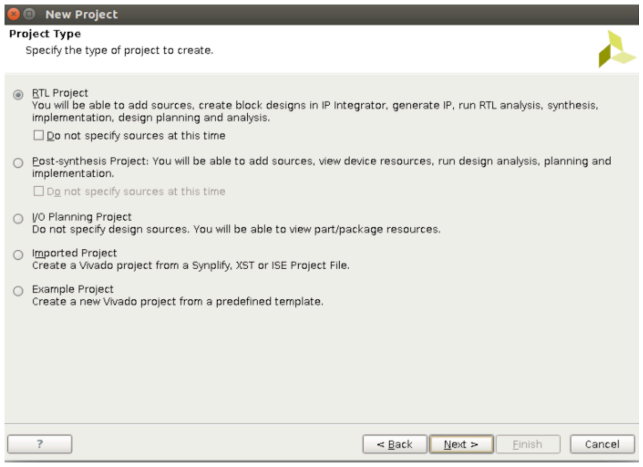
Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, click Next.

   - You need to specify the name and the project location in this section.
   - For the project type, if you do not know what to do, chose RTL project and press next.



Project Type
Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time

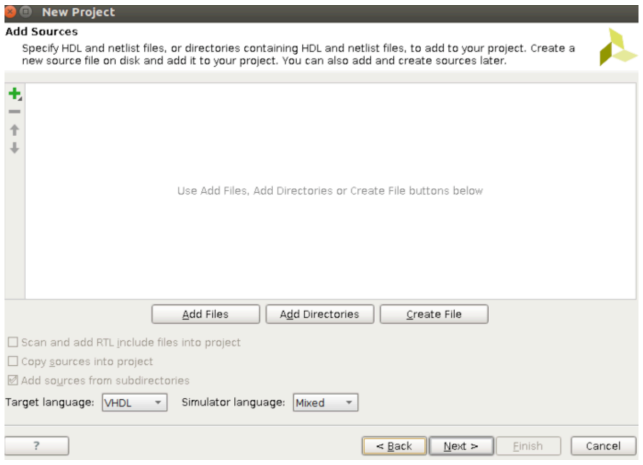
( ) Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

( ) Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

( ) Example Project
Create a new Vivado project from a predefined template.

   - For the Add sources, if you have the previous module click add files. You need to change the target language to VHDL if you compatible with VHDL. Finsh this then press next.


New Project

Add Sources

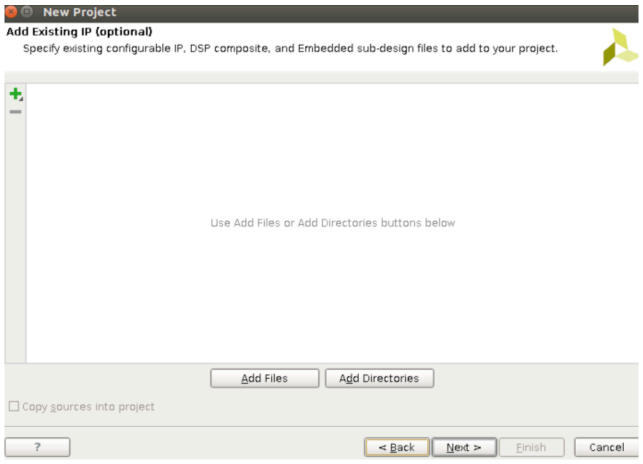
Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

Target language: VHDL Simulator language: Mixed

- For the add existing IP and add constrained file, unless you have the file press add files, press next.

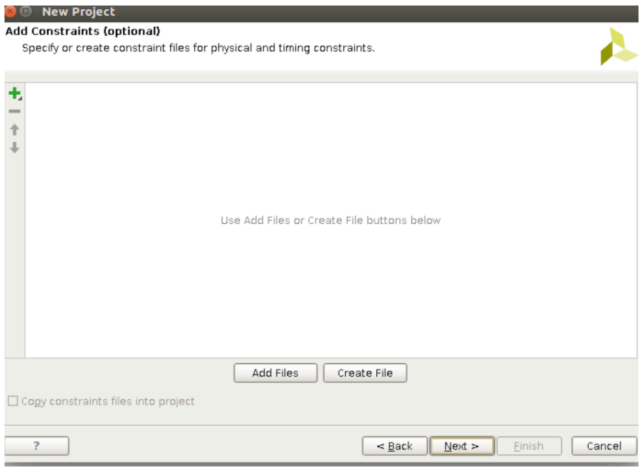

New Project

Add Existing IP (optional)

Specify existing configurable IP, DSP composite, and Embedded sub-design files to add to your project.

Use Add Files or Add Directories buttons below

☐ Copy sources into project


New Project

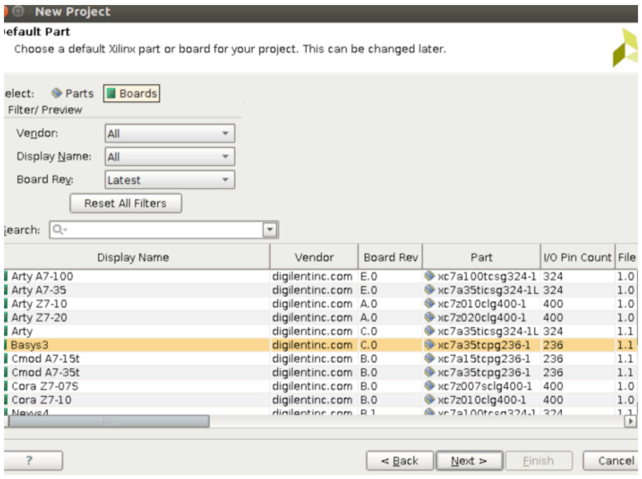
Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Use Add Files or Create File buttons below

☐ Copy constraints files into project

- For choosing the part, just pick up you part or boards that compatible with your boards. For example, press Boards and choose Basys3.



New Project

efault Part

Choose a default Xilinx part or board for your project. This can be changed later.

elect: Parts [✓ Boards]

Filter/ Preview

Vendor: All

Display Name: All

Board Rev: Latest

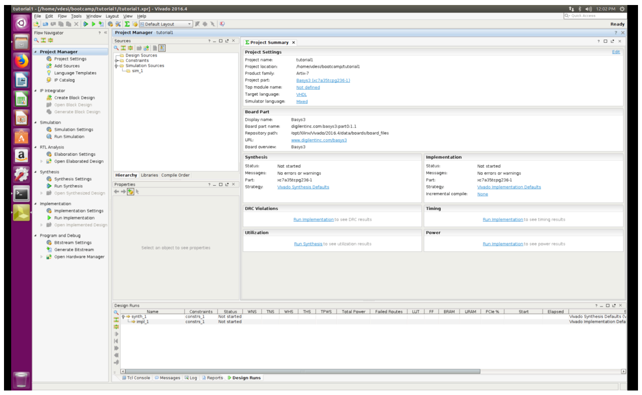
Reset All Filters

earch:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File
Arty A7-100	digilentinc.com	E.0	xc7a100tcsg324-1	324	1.0
Arty A7-35	digilentinc.com	E.0	xc7a35ticsg324-1L	324	1.0
Arty Z7-10	digilentinc.com	A.0	xc7z010clg400-1	400	1.0
Arty Z7-20	digilentinc.com	A.0	xc7z020clg400-1	400	1.0
Arty	digilentinc.com	C.0	xc7a35ticsg324-1L	324	1.1
Basys3	digilentinc.com	C.0	xc7a35tcpg236-1	236	1.1
Cmod A7-15t	digilentinc.com	B.0	xc7a15tcpg236-1	236	1.1
Cmod A7-35t	digilentinc.com	B.0	xc7a35tcpg236-1	236	1.1
Cora Z7-07S	digilentinc.com	B.0	xc7z007sclg400-1	400	1.0
Cora Z7-10	digilentinc.com	B.0	xc7z010clg400-1	400	1.0
Nexys4	digilentinc.com	B.1	xc7a100tcsg324-1	324	1.1

? < Back Next > Finish Cancel

- After finish choosing part, this is the summary of the project. Press Finish. The Vivado tool will show up.



Project Summary

Project Settings

Project name: tutorial1
Project location: /home/vdesi/bootcamp/tutorial1
Product family: Artix-7
Project part: Basys3 (xc7a35tcpg236-1)
Top module name: Not defined
Target language: VHDL
Simulator language: Mixed

Board Part

Display name: Basys3
Board part name: digilentinc.com:basys3:part0:1.1
Repository path: /opt/Xilinx/Vivado/2016.4/data/boards/board_files
URL: www.digilentinc.com/basys3
Board overview: Basys3

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

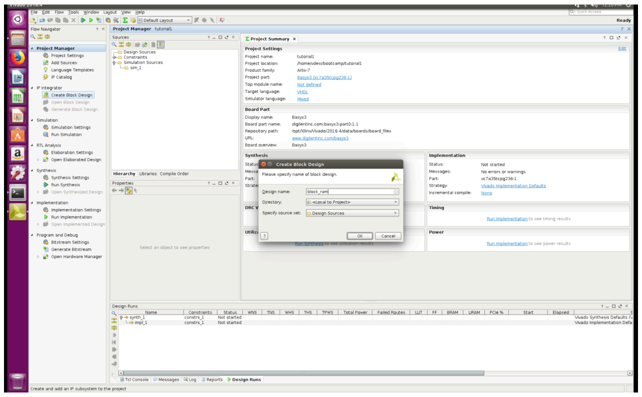
DRC Violations: Run Implementation to see DRC results

Timing: Run Implementation to see timing results

Utilization: Run Synthesis to see utilization results

Power: Run Implementation to see power results

3. To create the design, press create block design under IP integrator in the Flow navigator and named it.



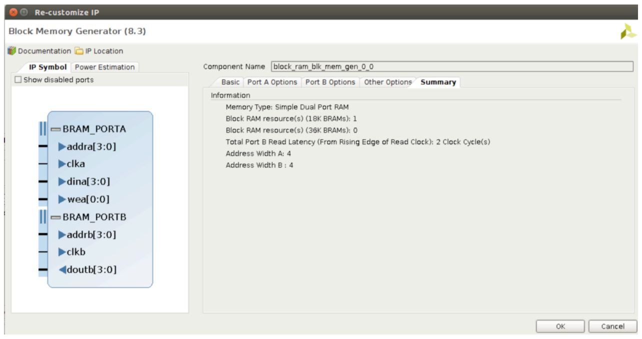
Create Block Design

Please specify name of block design.

Design name: block_ram
Directory: <Local to Project>
Specify source set: Design Sources

? OK Cancel

- First we create the Block RAM using Block Memory Generator by press add ip which is in the middle of the block design. Then double click the ip for configuration. In our design we will use stand alone mode with simple dual port memory type with no ECC. For port A is for write to Block RAM with 4 bit width and depth is 16. The enable port type is always enabled for simple reason. The design summary of Block RAM is shown. Then press OK.



Component Name block_ram_blk_mem_gen_0_0

Information

Memory Type: Simple Dual Port RAM

Block RAM resource(s) (18K BRAMs): 1

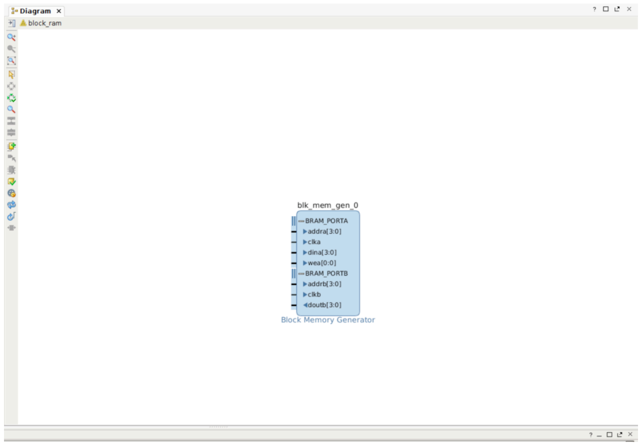
Block RAM resource(s) (36K BRAMs): 0

Total Port B Read Latency (From Rising Edge of Read Clock): 2 Clock Cycle(s)

Address Width A: 4

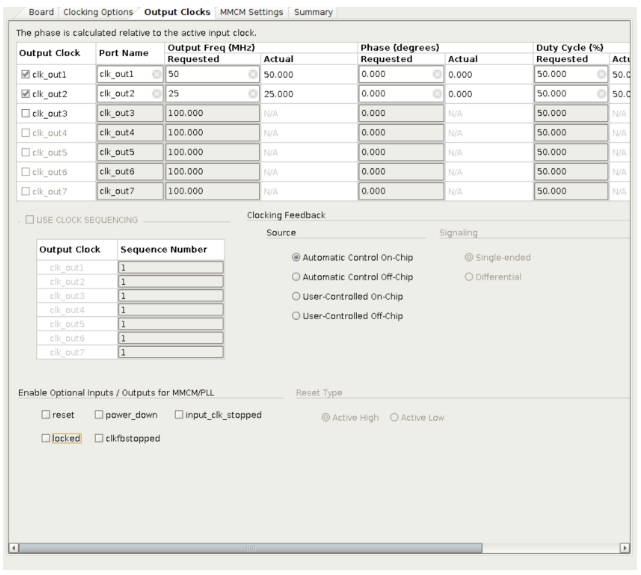
Address Width B : 4

- For the block design, now we will get this.



4. For the clock in Block RAM port A and B is difference value of clock due to did not check the common clock. Therefore we will generate the clock using Xilinx Clocking Wizard. For port A we will use faster clock (50 Mhz) due to writing the data and port B is slower (25 Mhz).

   - Press add ip which is the left side of the block design and choose clock wizard. Double click the clock wizard to configure it. First we look at the clocking option, the primary clock is set to 100Mhz and the rest should not be change. For the output clock, we will have 2 output clock: clk_out1 and clk_out2 which is 50Mhz and 25Mhz respectively. After this, unchecked the reset and locked.



Board | Clocking Options | **Output Clocks** | MMCM Settings | Summary

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)	
		Requested	Actual	Requested	Actual	Requested	Actu
☑ clk_out1	clk_out1	50	50.000	0.000	0.000	50.000	50.0
☑ clk_out2	clk_out2	25	25.000	0.000	0.000	50.000	50.0
☐ clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A
☐ clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A
☐ clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A
☐ clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A
☐ clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source: [✓ Automatic Control On-Chip] Automatic Control Off-Chip, User-Controlled On-Chip, User-Controlled Off-Chip

Signaling: [✓ Single-ended] Differential

Enable Optional Inputs / Outputs for MMCM/PLL

☐ reset ☐ power_down ☐ input_clk_stopped

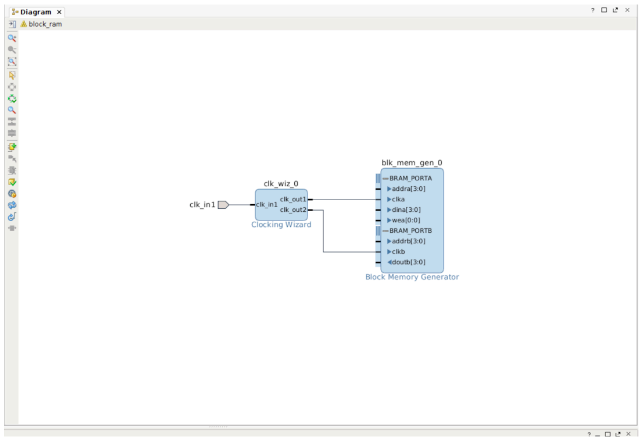
☐ locked ☐ clkfbstopped

Reset Type: [✓ Active High] Active Low

- Connect the faster clock to port clka and the slower to clkb.

- Connect the system clock to clk_in1 by right click that port and press make external.

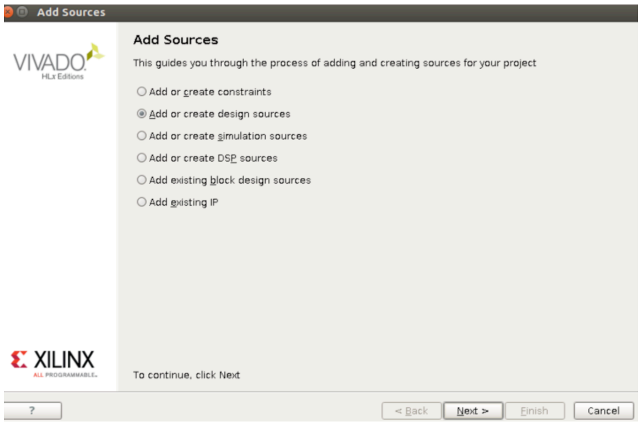
- The design now will be like this.



5. The next step is to create the module that write the data to the Block RAM. For the easy reason, we will write this table to the Block RAM.

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
data	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f

Therefore, we will write the VHDL counter module to write the data to the Block RAM by add design source in the flow navigator and create VHDL module. The another things that has to mentioned is the write enable port must be active when we write. Because of that we will have the input enable port that enable the wen port.

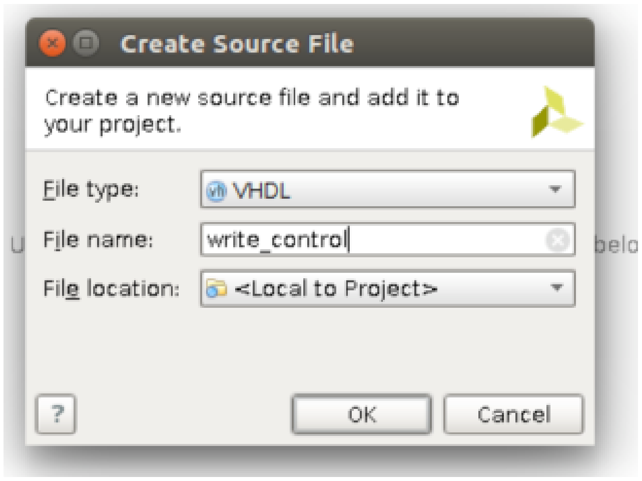

Add Sources

Add Sources

This guides you through the process of adding and creating sources for your project

- ○ Add or create constraints
- ◉ Add or create design sources
- ○ Add or create simulation sources
- ○ Add or create DSP sources
- ○ Add existing block design sources
- ○ Add existing IP

To continue, click Next


Create Source File

Create a new source file and add it to your project.

File type: VHDL

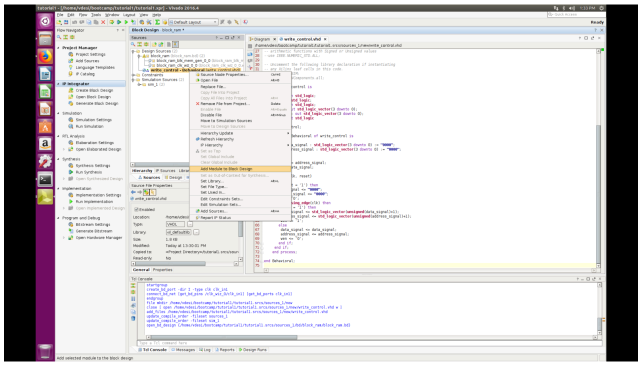
File name: write_control

File location: <Local to Project>

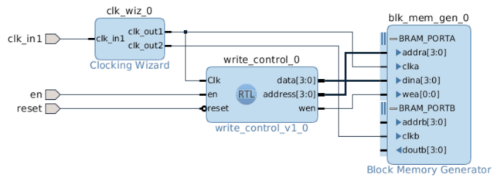
The VHDL counter is write below

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity write_control is
Port (
     Clk :  in std_logic;
     en : in std_logic;
     reset :in std_logic;
     data : out std_logic_vector(3 downto 0);
     address : out std_logic_vector(3 downto 0);
     wen : out std_logic
      );
end write_control;

architecture Behavioral of write_control is

   signal data_signal : std_logic_vector(3 downto 0) := "0000";
   signal address_signal : std_logic_vector(3 downto 0) := "0000";

begin
    address <= address_signal;
    data <= data_signal;
    process(clk, reset)
    begin
     if (reset = '1') then
      data_signal <= "0000";
      address_signal <= "0000";
      wen <= '0';
     elsif rising_edge(clk) then
       if (en = '1') then
        data_signal <= std_logic_vector(unsigned(data_signal)+1);
        address_signal <= std_logic_vector(unsigned(address_signal)+1);
        wen <= '1';
       else
        data_signal <= data_signal;
        address_signal <= address_signal;
        wen <= '0';
       end if;
     end if;
    end process;

end Behavioral;
```

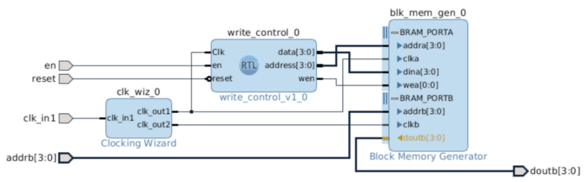
After finish write the code, we right click the file which is in sources under hierarchy section and press add module to block design.



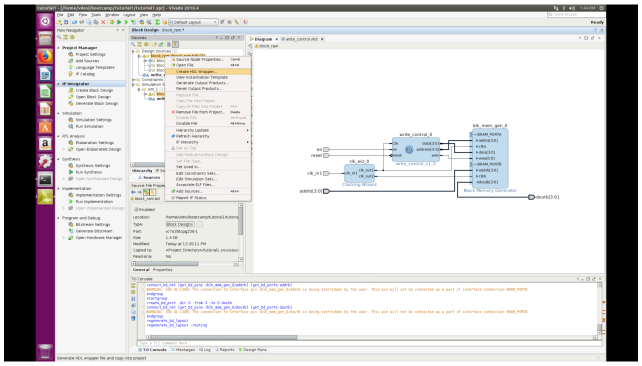
- For the write control module, we will use the 50 Mhz clock for meet our design requirement and the address and data connect to Block RAM port A. The en and reset pin is from outside our design. The design will be like this now.



6. Last but not least, we need create the pin for Block RAM port B address and data for simulation by right click make external. The final design will be like this.



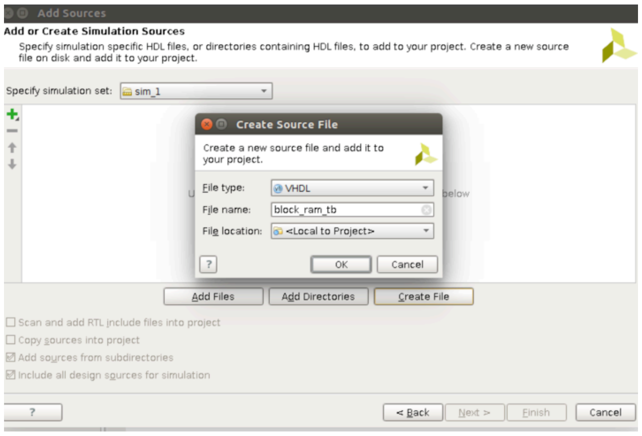
7. Now we finish the design, we need to create top module by right click the block_ram.bd and press create HDL wrapper.



Now our design is finish, the next is to simulation.

4 Block RAM Simulation

We will write the VHDL test bench module to read the data to the Block RAM by add simulation source in the flow navigator and create VHDL module.



Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Create Source File

Create a new source file and add it to your project.

File type: VHDL

File name: block_ram_tb

File location: <Local to Project>

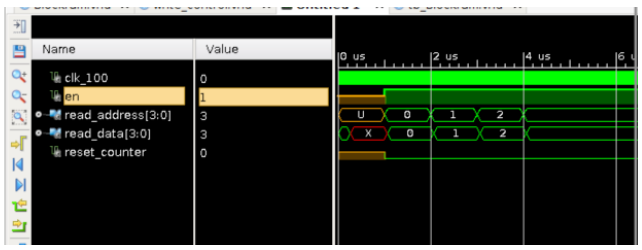
OK
Cancel
Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
Include all design sources for simulation
< Back
Next >
Finish
Cancel

The VHDL test bench is shown below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity tb_Blockram is
end tb_Blockram;

architecture Behavioral of tb_Blockram is

  component Blockram_wrapper
  port (
    clk_100 : in STD_LOGIC;
    en : in STD_LOGIC;
    read_address : in STD_LOGIC_VECTOR ( 3 downto 0 );
    read_data : out STD_LOGIC_VECTOR ( 3 downto 0 );
    reset_counter : in STD_LOGIC
  );
  end component;

  signal clk_100 : STD_LOGIC;
  signal en : STD_LOGIC;
  signal read_address : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal read_data : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal reset_counter : STD_LOGIC;
  constant clk_100_period : time := 10 ns;
begin
     -- Clock process definitions
     clk_100_process :process
     begin
        clk_100 <= '0';
        wait for clk_100_period/2;
        clk_100 <= '1';
        wait for clk_100_period/2;
     end process;

  uut : Blockram_wrapper PORT MAP (
    clk_100 => clk_100,
    en => en,
    read_address => read_address,
    read_data => read_data,
    reset_counter => reset_counter
  );

 -- Stimulus process
    stim_proc: process
    begin
       -- hold reset state for 100 ns.
       wait for 1000 ns;
       reset_counter <= '0';
       en <= '1';
       read_address <= "0000";

       wait for 1 us;
       read_address <= "0001";
       wait for 1 us;
       read_address <= "0010";
       wait for 1 us;
       read_address <= "0011";
       wait for 1 us;
       -- insert stimulus here

       wait;
    end process;
end Behavioral;
```

You can clearly see that we will read only address 0 to 3. We will run simulation. The simulation result is show below which the result from read address 0 to 3 is 0 to 3 respectively.



References

[1] David A. Patterson and John L. Hennessy. 2013. Computer Organization and Design, Fifth Edition: The Hardware/Software Interface (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[2] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, and Robert W. Stewart. 2014. The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 all Programmable Soc. Strathclyde Academic Media, , UK.