Embedded System Design with PICOBLAZE

Intro. to Embedded Systems - EHB326E

SERDAR DURAN

ITU Embedded System Design Laboratory

Design Steps

I. Write your algorithm as an **Assembly** Code
II. Simulate your **Assembly** Code in **Fidex IDE**
III. Generate your **instruction memory** from **Fidex IDE**
IV. Combine your **instruction memory** and **Picoblaze** design in **Vivado**
V. Write a **testbench** file to make a **simulation**.
VI. Make your simulations in **Xilinx Vivado**

Vivado and Fidex Environments

- **Install one of the recent version of Xilinx Vivado:**

https://www.xilinx.com/support/download.html

- **Install Fidex IDE – FIDEx 2016-01.0:**

https://www.fautronix.com/en/en-fidex-downloads

- **Download the ‘‘PicoBlaze for UltraScale, 7-series, 6-series FPGAs’’ design files:**

https://www.xilinx.com/products/intellectual-property/picoblaze.html#design

Picoblaze Overview

Picoblaze

- The **PicoBlaze** processor is a simple 8-bit microcontroller specifically designed and optimized for **Xilinx FPGA devices.**
- **Basic Features:**
  - 8-bit data width,
  - 8-bit ALU with carry and zero flags,
  - 16 8-bit general-purpose registers,
  - 64-byte data memory (Scratchpad RAM, 64*8 bit),
  - 8 input and 8 output pins,
  - 8 bit port identifier, meaning 256 addressable input and output ports,
  - **2 clock cycles per each instruction.**

*Link: PicoBlaze 8-bit Embedded Microcontroller User Guide

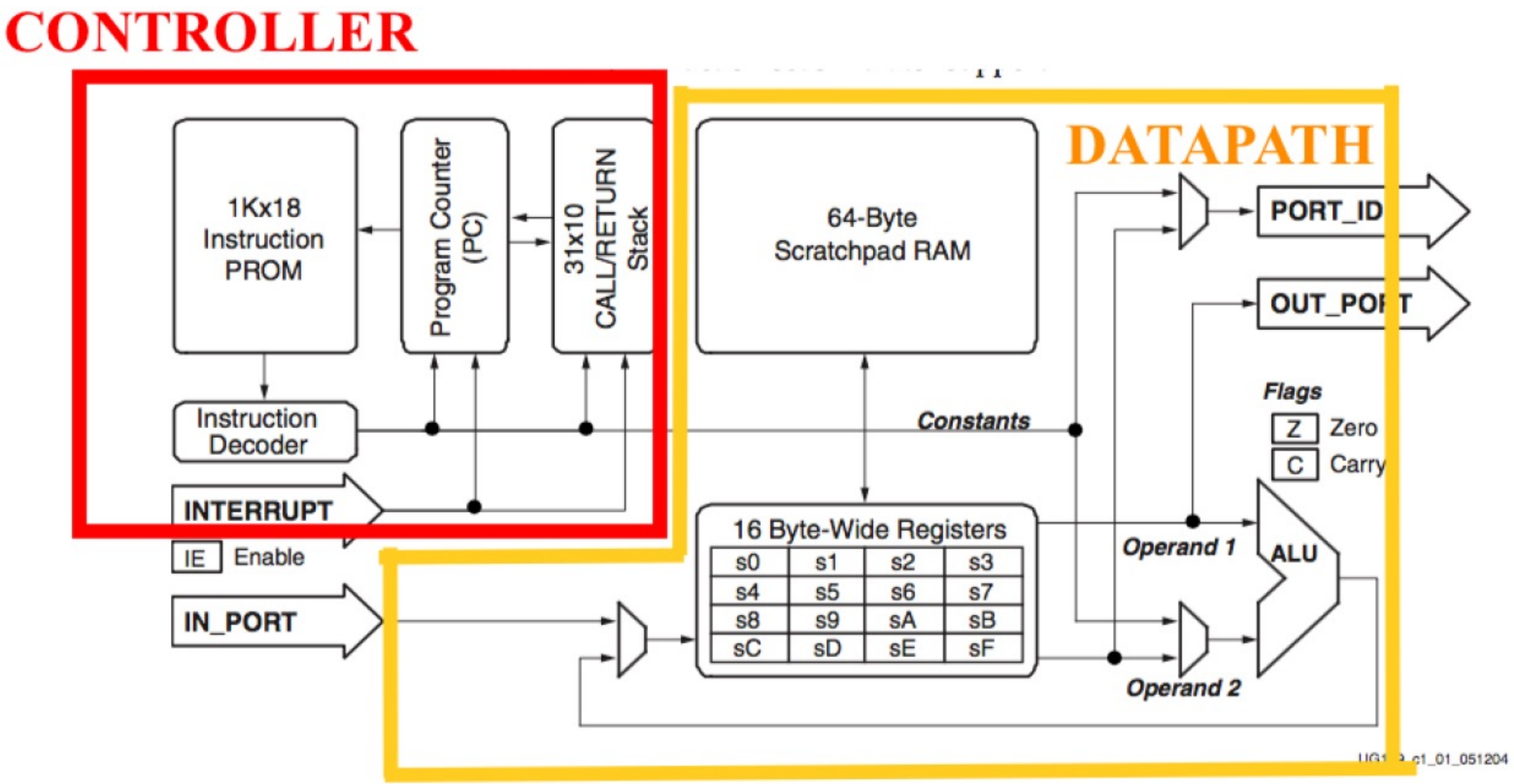
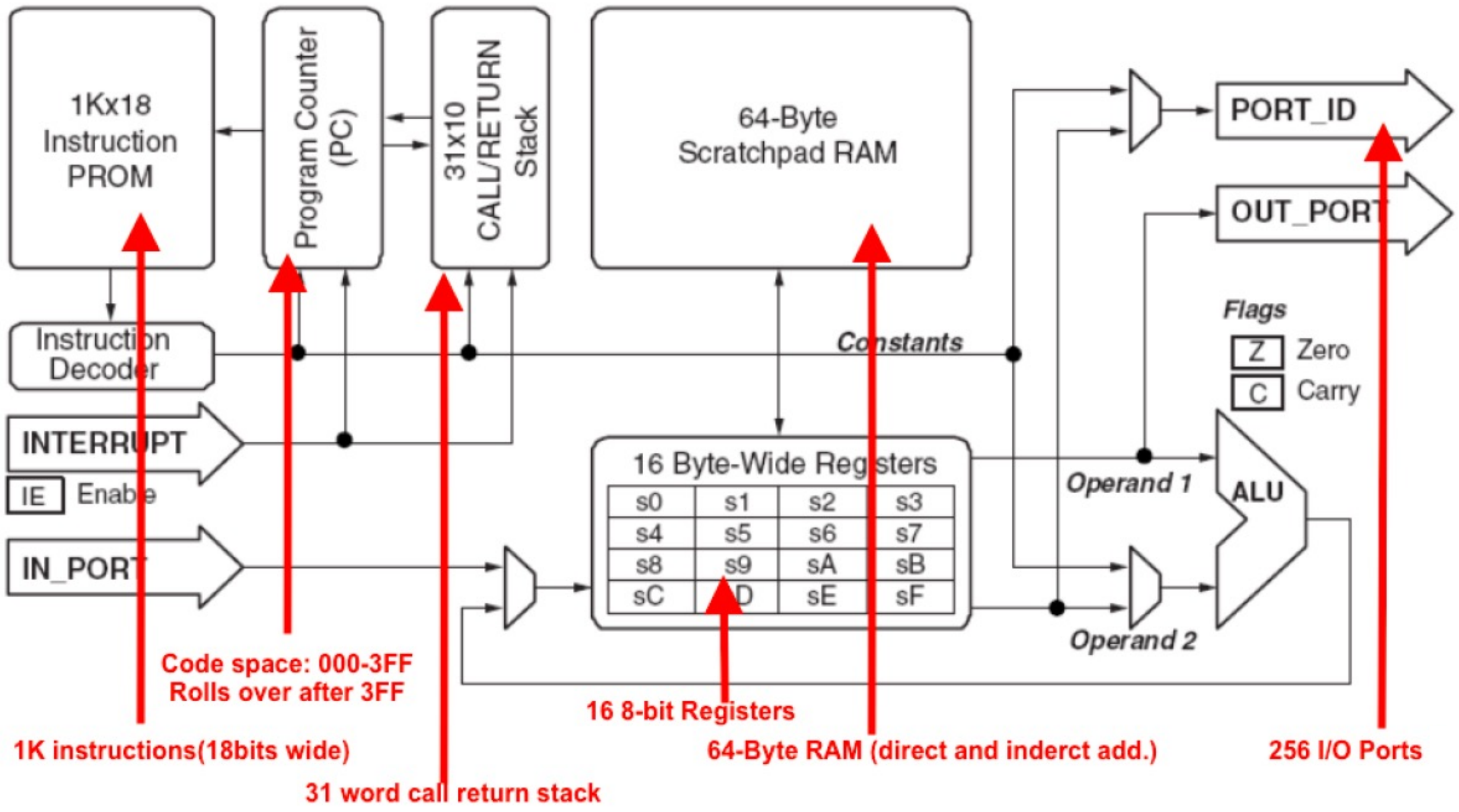


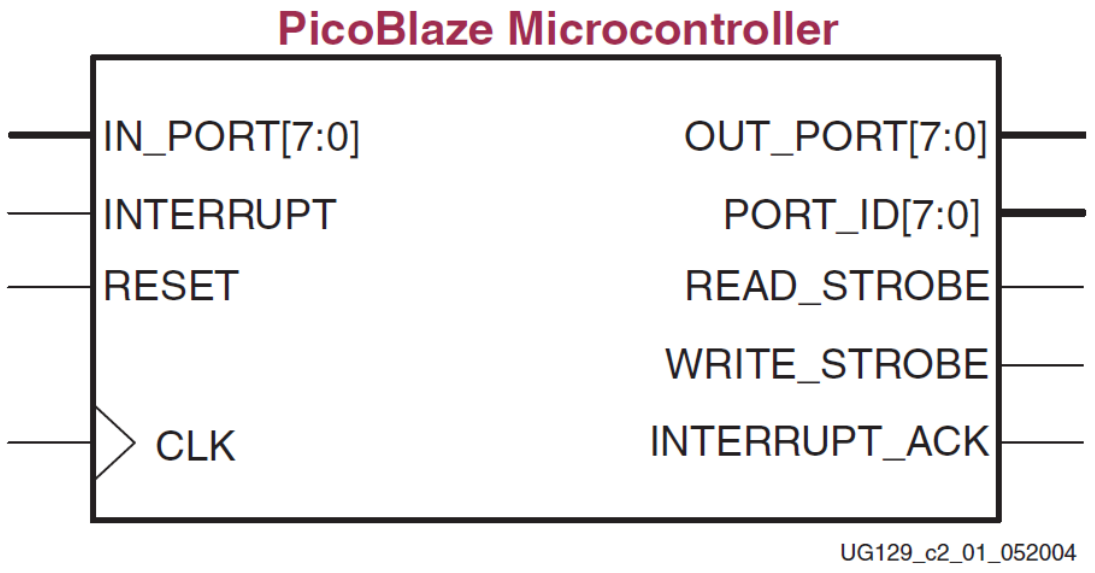
Figure 1-1: **PicoBlaze Embedded Microcontroller Block Diagram**


1Kx18 Instruction PROM
Program Counter (PC)
31x10 CALL/RETURN Stack
64-Byte Scratchpad RAM
PORT_ID
OUT_PORT
Flags
Z Zero
C Carry
Instruction Decoder
Constants
INTERRUPT
IE Enable
IN_PORT
16 Byte-Wide Registers
s0 s1 s2 s3
s4 s5 s6 s7
s8 s9 sA sB
sC sD sE sF
Operand 1
Operand 2
ALU
Code space: 000-3FF
Rolls over after 3FF
1K instructions(18bits wide)
31 word call return stack
16 8-bit Registers
64-Byte RAM (direct and inderct add.)
256 I/O Ports

Interfaces

- **IN_PORT [7:0] :** 8-bit data
- **OUT_PORT [7:0] :** 8-bit data
- **PORT_ID [7:0] :** port address for both INPUT and OUTPUT pins

- **READ_STROBE :** indicating input operation is done
- **WRITE_STROBE :** indicating output operation is done



Instruction Format

- **Instruction Format:**
  - **op sX,sY** ; register-register format. sX = sX op sY
  - **op sX,KK** ; register-constant format. sX = sX op KK
  - **op sX** ; single-register format. sX = op sX
  - **op AAA** ; single-address format (jump and call operations).

- **Instruction Types:**
  - Arithmetic
  - Logical
  - Shift and Rotate
  - Data Transfer
  - Branch Instructions

Instruction Set

- **Arithmetic Instructions:** ADD, ADDC, SUB, SUBC
  - **ADD** sX, constant ; sX = sX + constant
  - **ADD** sX, sY ; sX = sX + sY

  - **ADDC** sX, constant ; sX = sX + constant + carry bit
  - **ADDC** sX, sY ; sX = sX + sY + carry bit

- **Logical Instructions:** AND, OR, XOR
  - **AND** sX, constant ; sX = sX & constant
  - **AND** sX, sY ; sX = sX & sY

Instruction Set

- **Comparison Instructions:** COMP, COMPC, TEST, TESTC
  - **COMP** sX, sY ; if regY > regX, then set CARRY, if regY = regX then set ZERO flag.
  - **COMP** sX, const ; if regY > const, then set CARRY, if regY = regX then set ZERO flag.

- **Shift and Rotate:** SL0, SL1, SLA, SLX, SR0, SR1, SRA, SRX, RL, RR
  - **SL0** sX ; shift register sX left, zero fill
  - **SR1** sX ; shift register sX right, one fill

- **Branch Instructions:** JUMP, RETURN
  - **JUMP** aaa ; Unconditionally jump to aaa label
  - **JUMP C** aaa ; If CARRY flag set, jump to aaa label
  - **JUMP Z** aaa ; If ZERO flag set, jump to aaa label

Instruction Set

- **Data Transfer Instructions:** LOAD, FETCH, STORE, INPUT, OUTPUT
  - **LOAD** sX, constant ; Load register sX with constant
  - **LOAD** sX, sY ; Load register sX with sY

  - **WRMEM** sX, scrpdAddr ; store regX into the memory at address scrpdAddr.
  - **WRMEM** sX, (sY) ; store regX into the memory at address (sY).

  - **RDMEM** sX, scrpdAddr ; fetch the value at address scrpdAddr into the sX.
  - **RDMEM** sX, (sY) ; fetch the value at address (sY) into the sX.

  - **WRPRT** sX, busAddr ; output value on output port busAddr
  - **RDPRT** sX, (sY) ; input value on input port (sY)

Fidex Part

Default Specifications for Verilog

```
#ifDef proc::xPblze6
; PICOBLAZE 6 CONFIGs
#set proc::xPblze6:: scrpdSize,       64           ; [64, 128, 256]
#set proc::xPblze6:: clkFreq,         100000000    ; in Hz

; INST MEMORY SPECs
#set IOdev::BRAM0:: en,               TRUE
#set IOdev::BRAM0:: type,             mem
#set IOdev::BRAM0:: size,             4096
#set instmem:: pageSize,              4096
#set instmem:: pageCount,             1
#set instmem:: sharedMemLocation,     loMem         ;[ hiMem, loMem ]
#set IOdev::BRAM0:: value,            instMem

; VERILOG OUTPUT FILES
#set IOdev::BRAM0:: verilogEn,                   TRUE
#set IOdev::BRAM0:: verilogEntityName,           "BRAM0"
#set IOdev::BRAM0:: verilogTmplFile,             "ROM_form.v"
#set IOdev::BRAM0:: verilogTargetFile,           "BRAM0.v"
#endIf
```

Picoblaze 6 Configs

- Scratchpad RAM size
- Clock frequency

Instruction Memory Settings

- Page Size
- Page number

Verilog Output File Settings

- ROM_form.v is a template
- BRAM0.v is the Verilog output for instruction memory

Default Specifications for VHDL

```
#ifDef proc::xPblze6
; PICOBLAZE 6 CONFIGs
#set proc::xPblze6:: scrpdSize,     64          ; [64, 128, 256]
#set proc::xPblze6:: clkFreq,       100000000   ; in Hz

; INST MEMORY SPECs
#set IOdev::BRAM0:: en,              TRUE
#set IOdev::BRAM0:: type,            mem
#set IOdev::BRAM0:: size,            4096
#set instmem:: pageSize,             4096
#set instmem:: pageCount,            1
#set instmem:: sharedMemLocation,    loMem         ;[ hiMem, loMem ]
#set IOdev::BRAM0:: value,           instMem

; VERILOG OUTPUT FILES
#set IOdev::BRAM0:: vhdlEn,          TRUE
#set IOdev::BRAM0:: vhdlEntityName, "BRAM0"
#set IOdev::BRAM0:: vhdlTmplFile,    "ROM_form.vhd"
#set IOdev::BRAM0:: vhdlTargetFile, "BRAM0.vhd"
#endIf
```

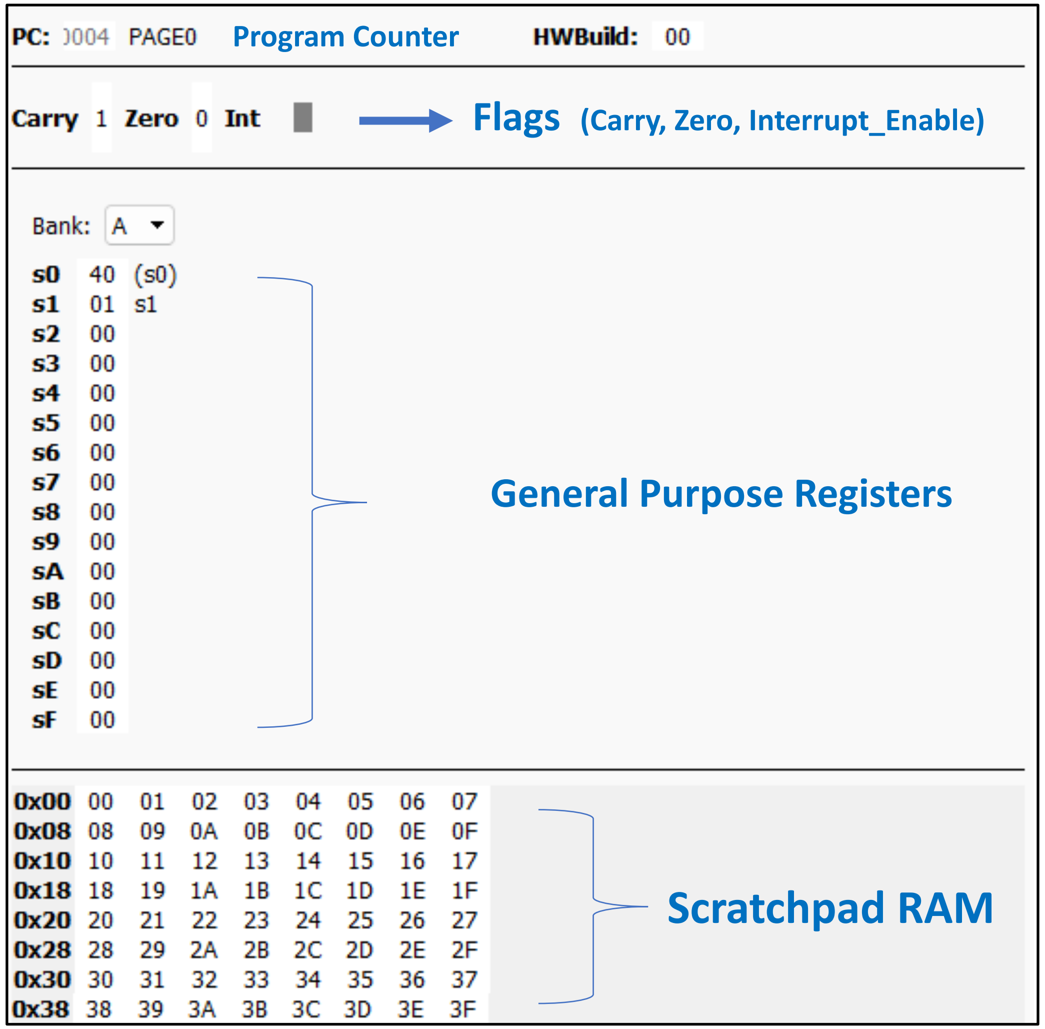
VHDL Output File Settings

- ROM_form.vhd is a template
- BRAM0.vhd is the VHDL output for instruction memory

Example - Fidex

```
              ; starting address
              #ORG ADDR, 0

              LOAD  s0, 0
              LOAD  s1, 1

loop:         ADD   s0, s1
              COMP  s0, 63
              WRMEM s0,(s0)
              JUMP C, loop

end:          JUMP end

; fill the scratchpad RAM with
values 0 to 63 (0x3F)
```

Simulation



Program Counter
Flags (Carry, Zero, Interrupt_Enable)
General Purpose Registers
Scratchpad RAM

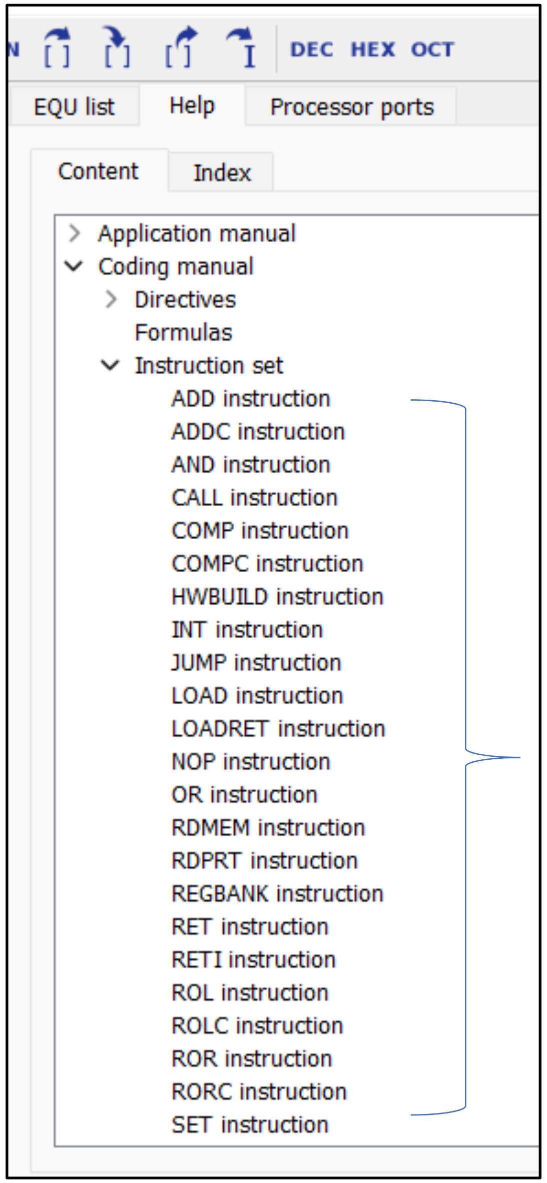
PC: 004 PAGE0 HWBuild: 00

Carry 1 Zero 0 Int

Bank: A

```
s0  40  (s0)
s1  01  s1
s2  00
s3  00
s4  00
s5  00
s6  00
s7  00
s8  00
s9  00
sA  00
sB  00
sC  00
sD  00
sE  00
sF  00
```

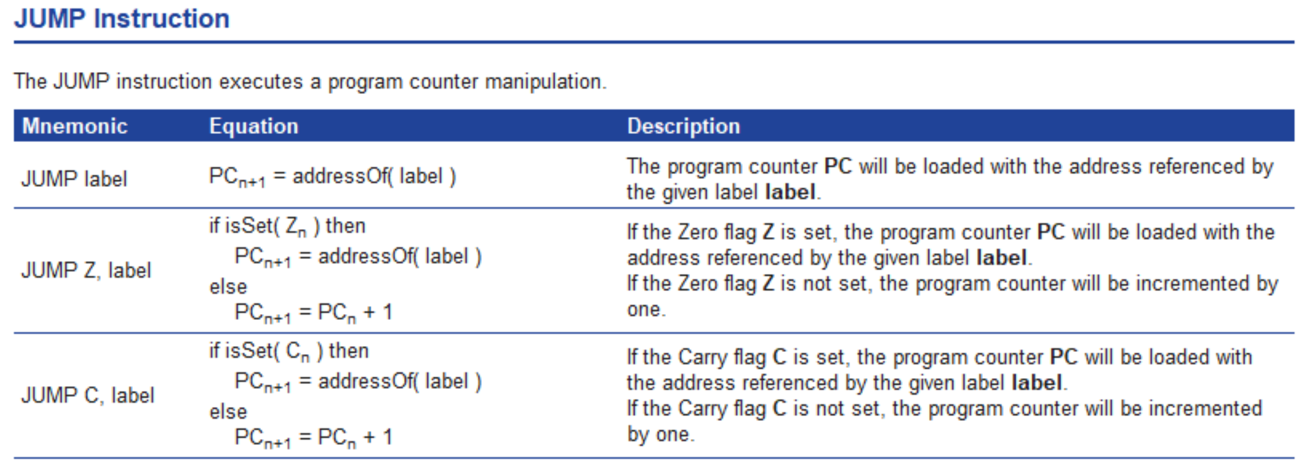
```
0x00 00 01 02 03 04 05 06 07
0x08 08 09 0A 0B 0C 0D 0E 0F
0x10 10 11 12 13 14 15 16 17
0x18 18 19 1A 1B 1C 1D 1E 1F
0x20 20 21 22 23 24 25 26 27
0x28 28 29 2A 2B 2C 2D 2E 2F
0x30 30 31 32 33 34 35 36 37
0x38 38 39 3A 3B 3C 3D 3E 3F
```



- Application manual
- Coding manual
  - Directives
  - Formulas
  - Instruction set
    - ADD instruction
    - ADDC instruction
    - AND instruction
    - CALL instruction
    - COMP instruction
    - COMPC instruction
    - HWBUILD instruction
    - INT instruction
    - JUMP instruction
    - LOAD instruction
    - LOADRET instruction
    - NOP instruction
    - OR instruction
    - RDMEM instruction
    - RDPRT instruction
    - REGBANK instruction
    - RET instruction
    - RETI instruction
    - ROL instruction
    - ROLC instruction
    - ROR instruction
    - RORC instruction
    - SET instruction

Instruction Set Manual

help->content



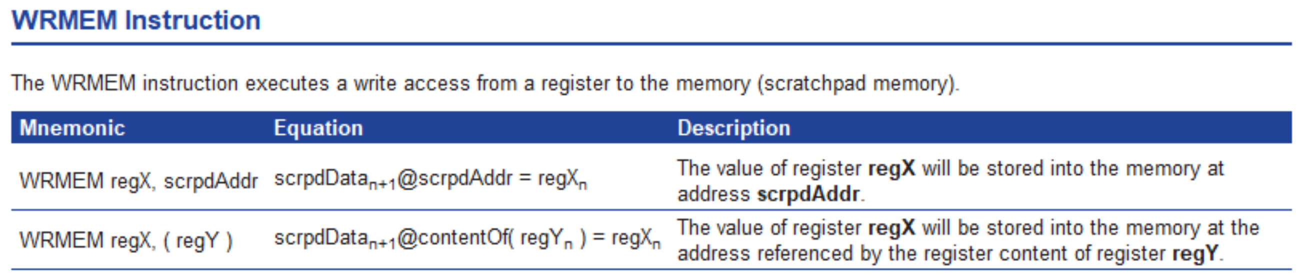
JUMP Instruction

The JUMP instruction executes a program counter manipulation.

Mnemonic	Equation	Description
JUMP label	PC_{n+1} = addressOf(label)	The program counter **PC** will be loaded with the address referenced by the given label **label**.

JUMP Z, label	if isSet(Z_n) then
	PC_{n+1} = addressOf(label)
	else
	$PC_{n+1} = PC_n + 1$
		If the Zero flag **Z** is set, the program counter **PC** will be loaded with the address referenced by the given label **label**.
		If the Zero flag **Z** is not set, the program counter will be incremented by one.

JUMP C, label	if isSet(C_n) then
	PC_{n+1} = addressOf(label)
	else
	$PC_{n+1} = PC_n + 1$
		If the Carry flag **C** is set, the program counter **PC** will be loaded with the address referenced by the given label **label**.
		If the Carry flag **C** is not set, the program counter will be incremented by one.



WRMEM Instruction

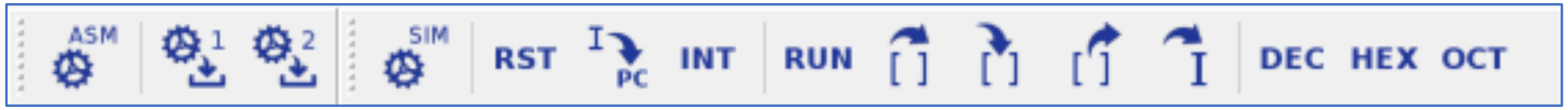
The WRMEM instruction executes a write access from a register to the memory (scratchpad memory).

Mnemonic	Equation	Description
WRMEM regX, scrpdAddr	$scrpdData_{n+1}$@scrpdAddr = $regX_n$	The value of register **regX** will be stored into the memory at address **scrpdAddr**.
WRMEM regX, (regY)	$scrpdData_{n+1}$@contentOf($regY_n$) = $regX_n$	The value of register **regX** will be stored into the memory at the address referenced by the register content of register **regY**.

Generate the HDL output of instruction memory

Reset Simulation

Run all the codes



Run Assembler

Simulate your code

Give an external interrupt

Simulate Next instruction

Change the numeric base

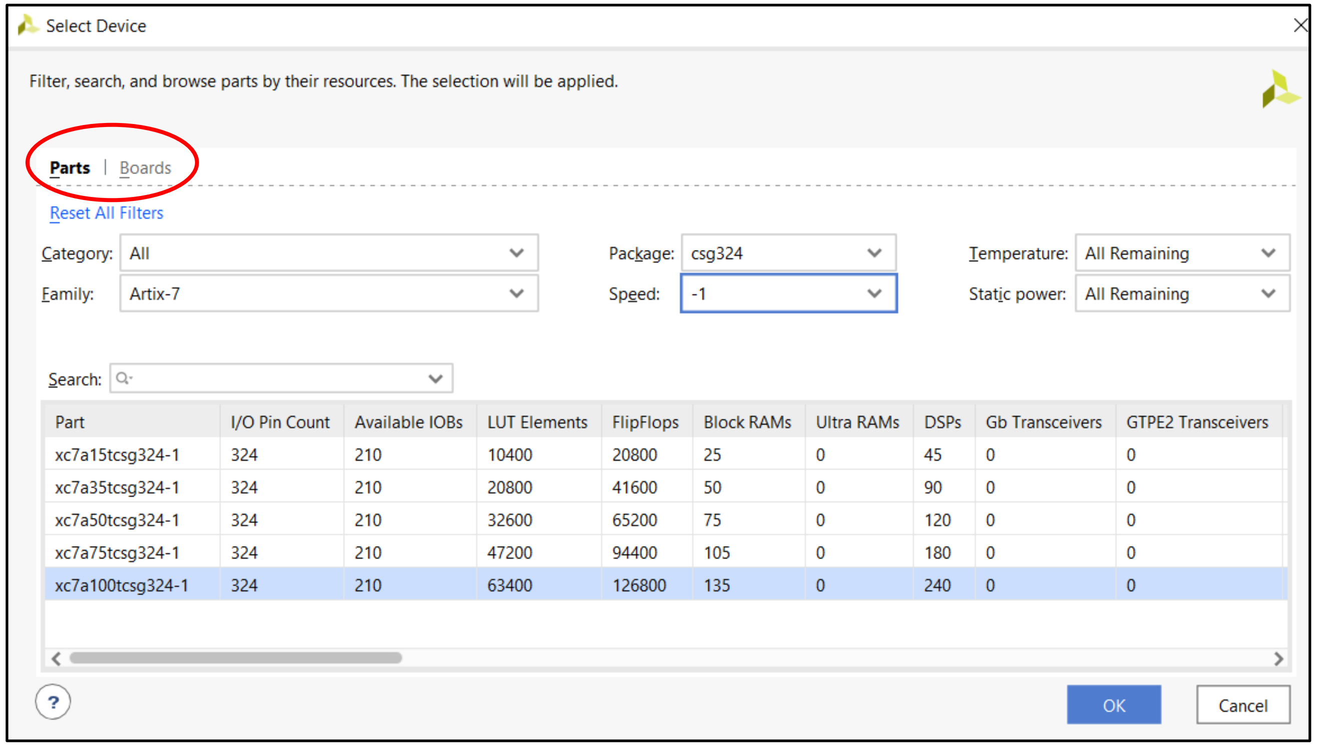
Coding Steps

1) Open or Create a new project
2) Make a **configuration** for your project
    - Xilinx Picoblaze 6
    - Scratchpad Size 64
    - Interrupt vector address 1023
    - Clock frequency 100 MHz
3) Write the default **specifications** (Verilog or VHDL)
4) Write your Assembly code
5) Run **Assembler**
6) Make a **Simulation**
7) Generate the HDL file if there is no problem with your code.
8) BRAM0.v or BRAM0.vhd will be your instruction memory file.

Vivado Part

Creating Project

- Create a new project in **Vivado** and use either **Nexys4 DDR** board or **xc7a100tcsg324-1** chip. (they are same)
- Please look at the online tutorial here!



Select Device

Filter, search, and browse parts by their resources. The selection will be applied.

Parts | Boards

Reset All Filters

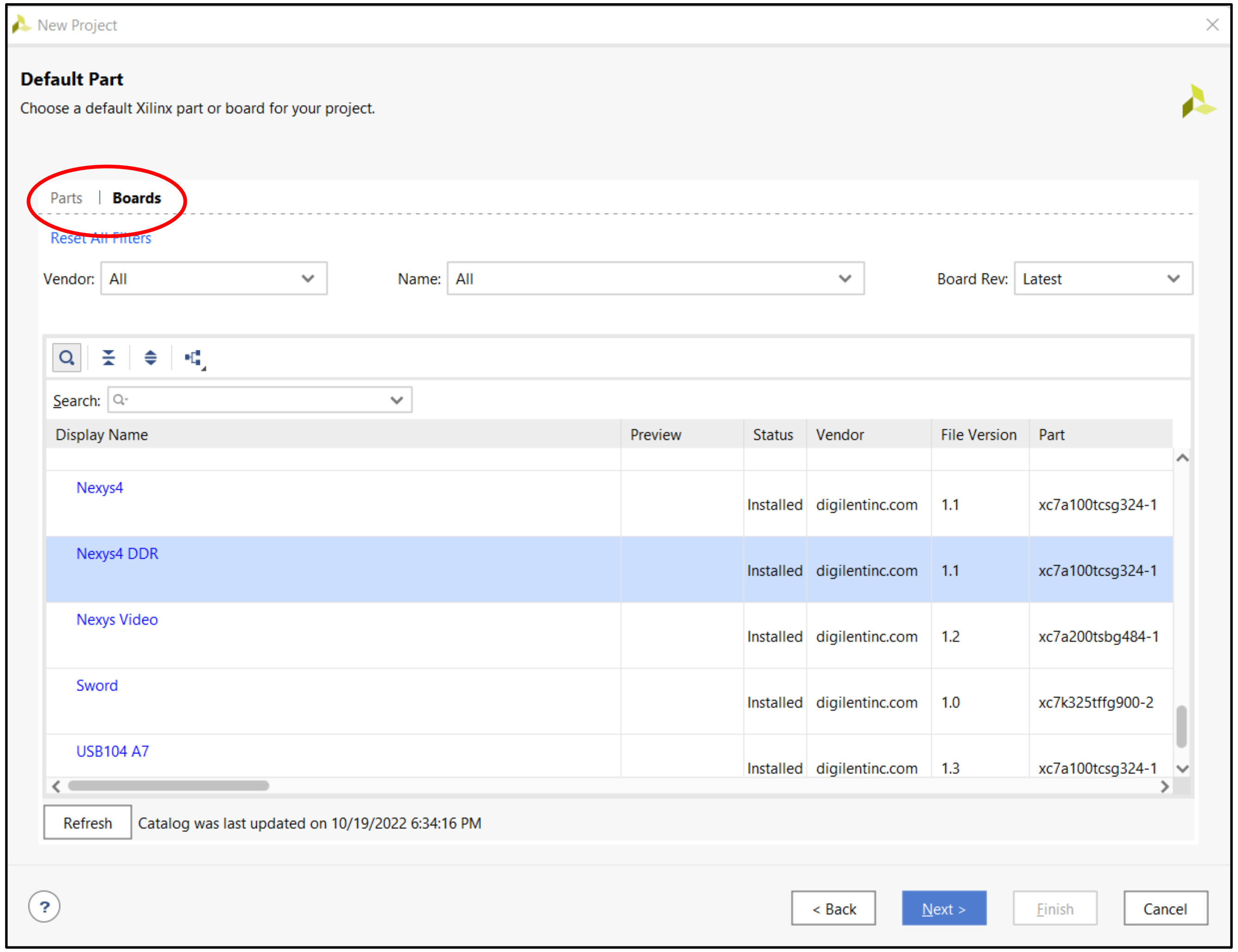
Category: All
Family: Artix-7
Package: csg324
Speed: -1
Temperature: All Remaining
Static power: All Remaining

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 Transceivers
xc7a15tcsg324-1	324	210	10400	20800	25	0	45	0	0
xc7a35tcsg324-1	324	210	20800	41600	50	0	90	0	0
xc7a50tcsg324-1	324	210	32600	65200	75	0	120	0	0
xc7a75tcsg324-1	324	210	47200	94400	105	0	180	0	0
xc7a100tcsg324-1	324	210	63400	126800	135	0	240	0	0

- To install the **NEXSYS 4 DDR** boardfile click here!

Creating Project


New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | **Boards**

Reset All Filters

Vendor: All   Name: All   Board Rev: Latest

Search:

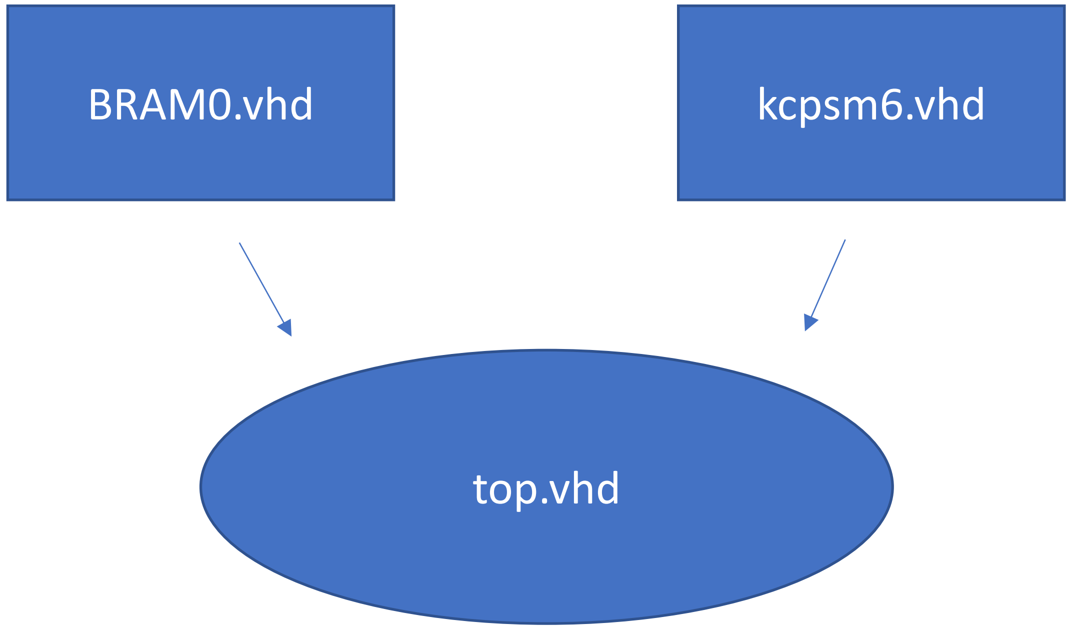
Display Name	Preview	Status	Vendor	File Version	Part
Nexys4		Installed	digilentinc.com	1.1	xc7a100tcsg324-1
Nexys4 DDR		Installed	digilentinc.com	1.1	xc7a100tcsg324-1
Nexys Video		Installed	digilentinc.com	1.2	xc7a200tsbg484-1
Sword		Installed	digilentinc.com	1.0	xc7k325tffg900-2
USB104 A7		Installed	digilentinc.com	1.3	xc7a100tcsg324-1

Refresh   Catalog was last updated on 10/19/2022 6:34:16 PM

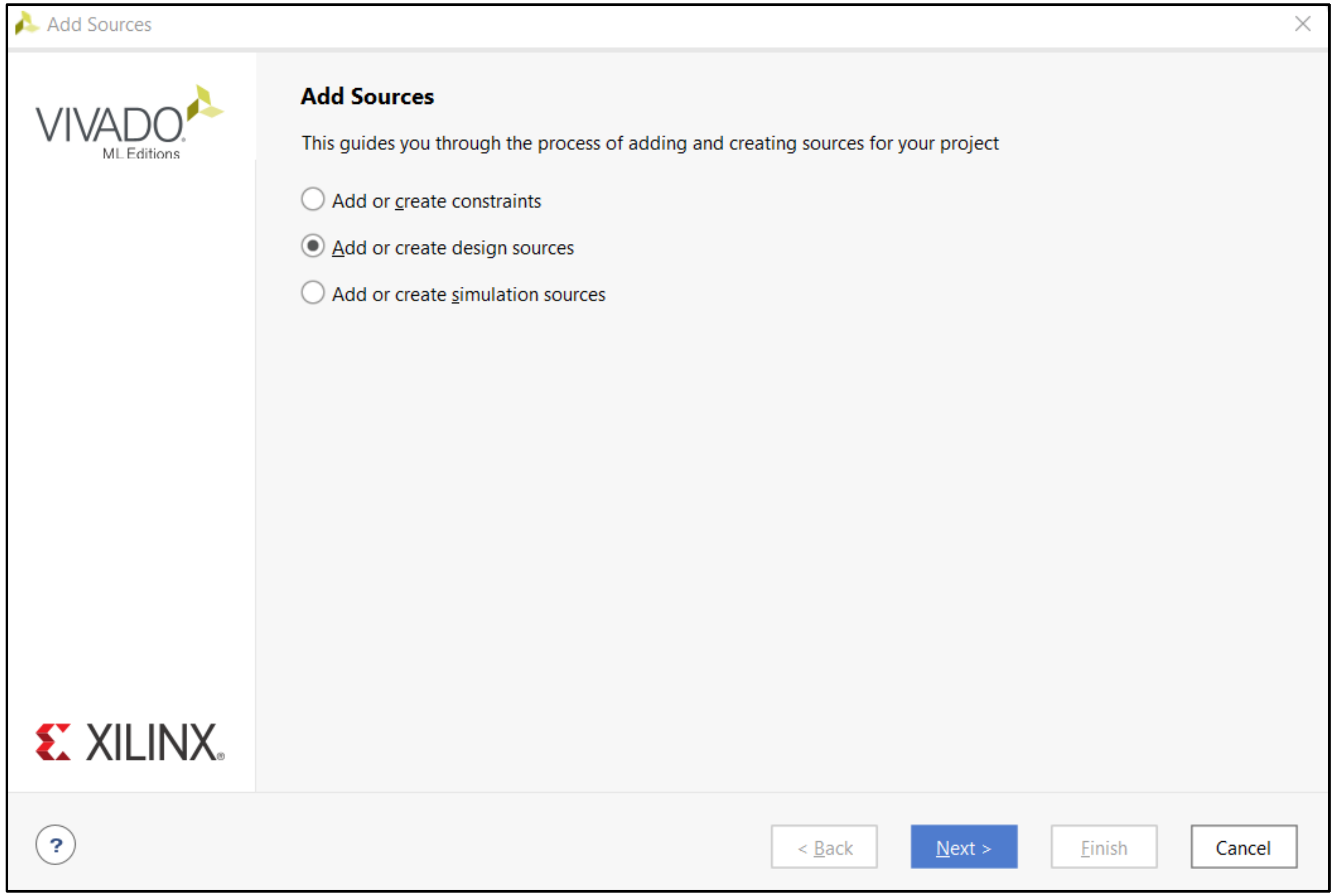
< Back   Next >   Finish   Cancel

Vivado Part

- Fidex IDE generates **instruction memory** according to your assembly code.
- Picoblaze projects are implemented by combining generated **instruction memory** and **picoblaze design** file**.**
- You can write a **top** entity that combine **BRAM0** (inst memory) and **KCPSM6** (picoblaze) files.



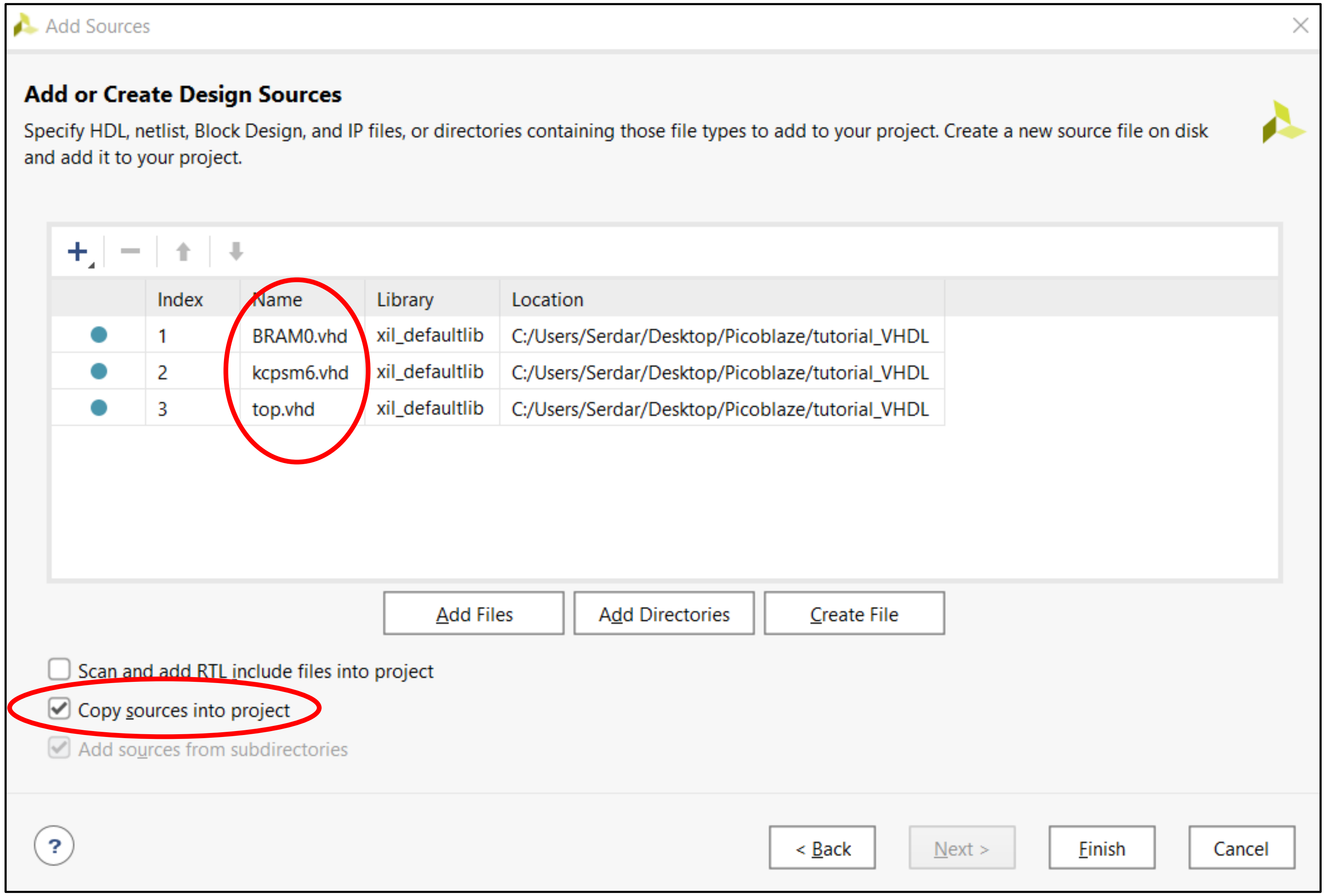
- Include the generated **BRAM0.vhd**, **kcpsm6.vhd** and **top.vhd** as Design Sources.


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 Design Sources

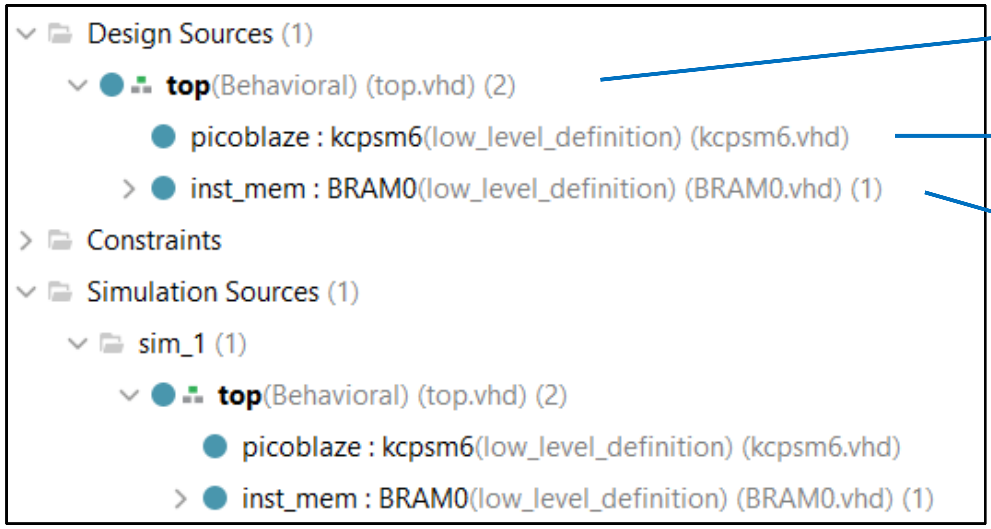
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	Index	Name	Library	Location
●	1	BRAM0.vhd	xil_defaultlib	C:/Users/Serdar/Desktop/Picoblaze/tutorial_VHDL
●	2	kcpsm6.vhd	xil_defaultlib	C:/Users/Serdar/Desktop/Picoblaze/tutorial_VHDL
●	3	top.vhd	xil_defaultlib	C:/Users/Serdar/Desktop/Picoblaze/tutorial_VHDL

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

Vivado Part

- After adding **Design Files** you can see the hierarchy between these files.
- To see your design works as expected you need to write a **testbench** file and add it as a **Simulation Source File**.



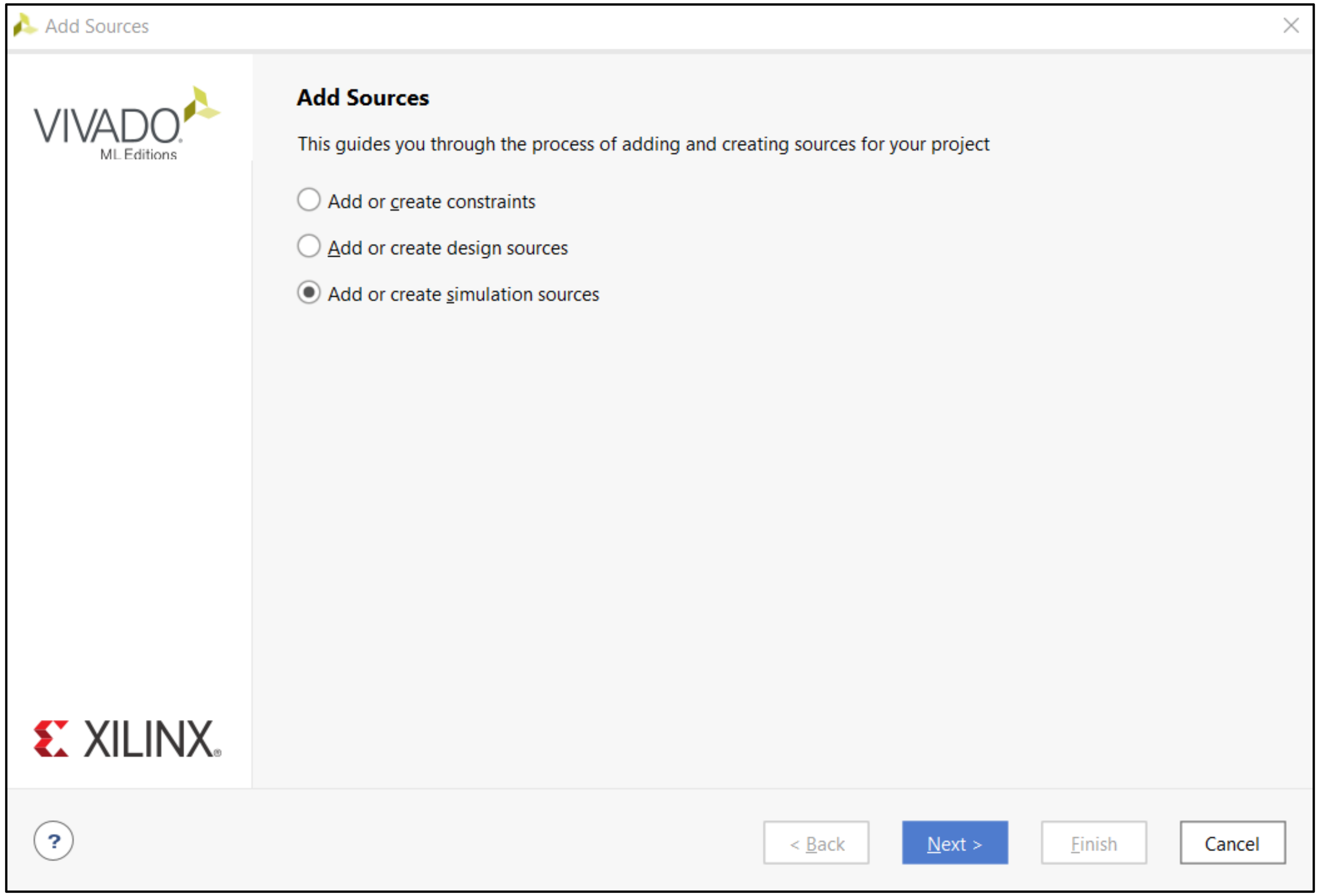
```
Design Sources (1)
  top(Behavioral) (top.vhd) (2)
    picoblaze : kcpsm6(low_level_definition) (kcpsm6.vhd)
    inst_mem : BRAM0(low_level_definition) (BRAM0.vhd) (1)
Constraints
Simulation Sources (1)
  sim_1 (1)
    top(Behavioral) (top.vhd) (2)
      picoblaze : kcpsm6(low_level_definition) (kcpsm6.vhd)
      inst_mem : BRAM0(low_level_definition) (BRAM0.vhd) (1)
```

Top entity includes inst_memory and picoblaze

Picoblaze (kcpsm6.vhd)

instruction memory (BRAM0.vhd)

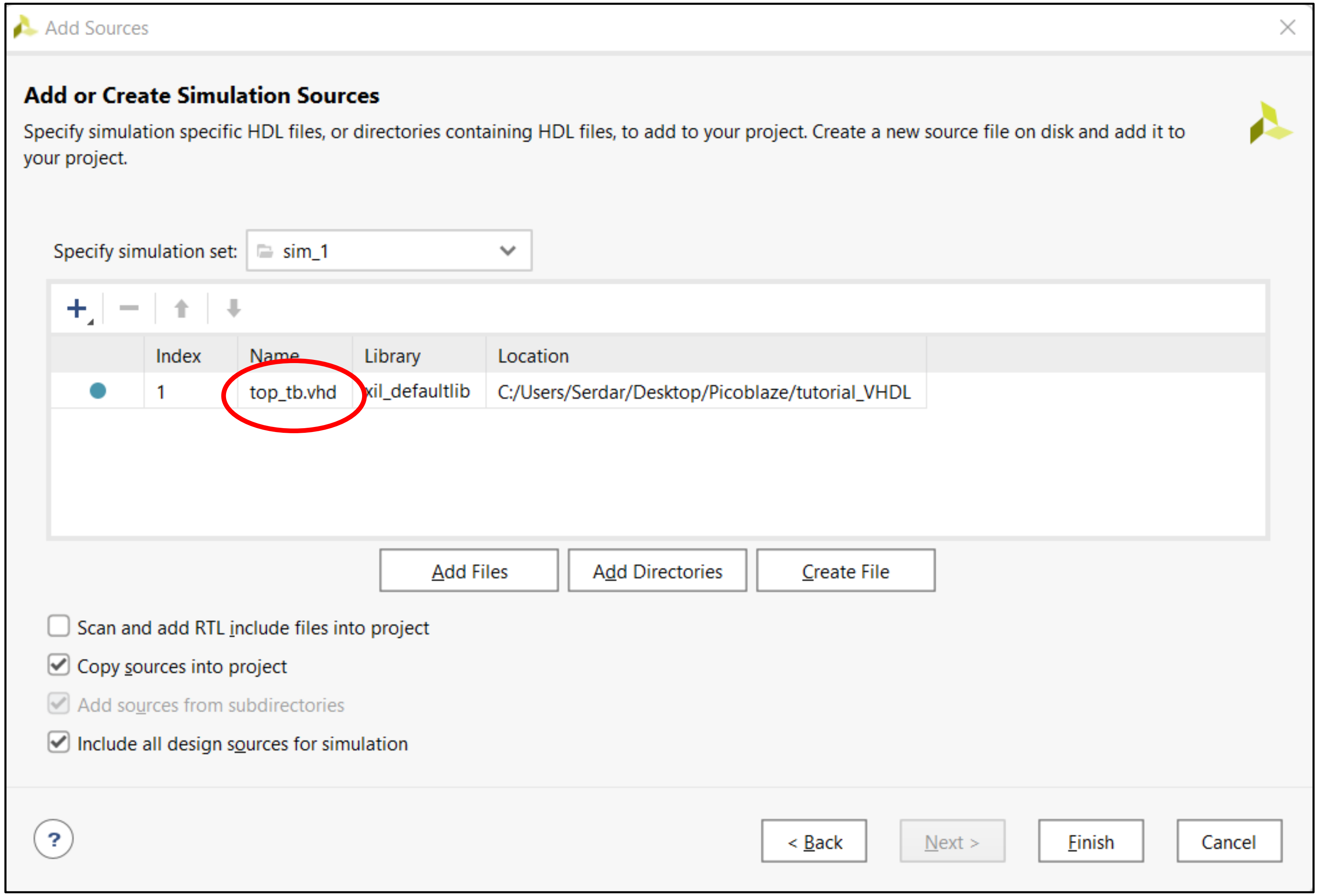
- Include the **top_tb.vhd** as a **Simulation Source**.


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 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.

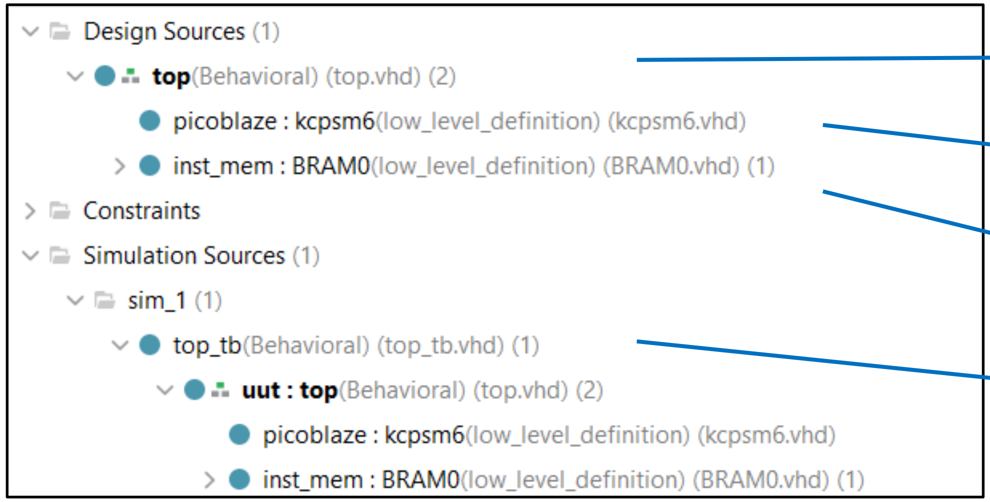
Specify simulation set: sim_1

	Index	Name	Library	Location
●	1	top_tb.vhd	xil_defaultlib	C:/Users/Serdar/Desktop/Picoblaze/tutorial_VHDL

- [ ] Scan and add RTL include files into project
- [x] Copy sources into project
- [x] Add sources from subdirectories
- [x] Include all design sources for simulation

Vivado Part

- Hierarchy between these entities (files):



```
Design Sources (1)
  top(Behavioral) (top.vhd) (2)
      picoblaze : kcpsm6(low_level_definition) (kcpsm6.vhd)
    > inst_mem : BRAM0(low_level_definition) (BRAM0.vhd) (1)
> Constraints
Simulation Sources (1)
  sim_1 (1)
    top_tb(Behavioral) (top_tb.vhd) (1)
      uut : top(Behavioral) (top.vhd) (2)
          picoblaze : kcpsm6(low_level_definition) (kcpsm6.vhd)
        > inst_mem : BRAM0(low_level_definition) (BRAM0.vhd) (1)
```

Top entity includes **inst_memory** and **picoblaze**

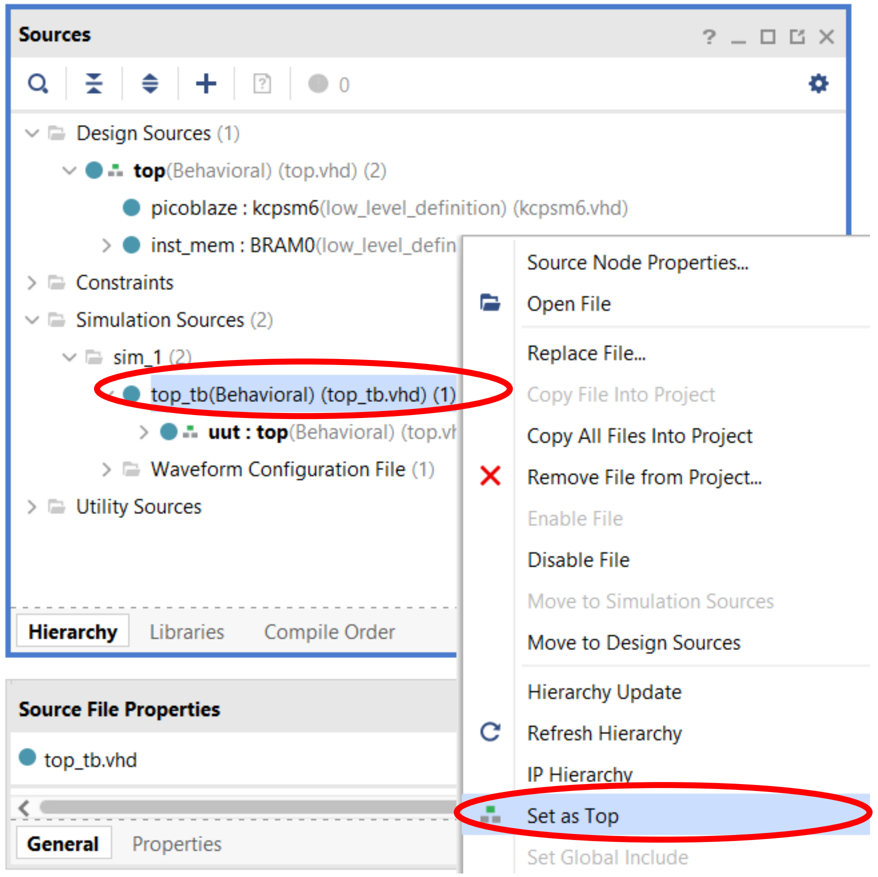
Picoblaze (**kcpsm6.vhd**)

instruction memory (**BRAM0.vhd**)

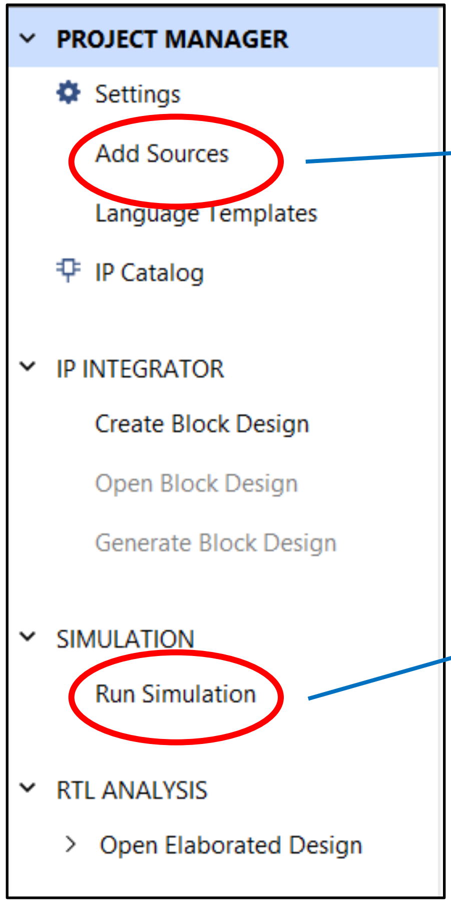
Testbench file for simulation (**top_tb.vhd**)

Vivado Part

- If the **top_tb.vhd** testbench is not seen as top entity (bald) **" right click->Set as Top"** this entity.



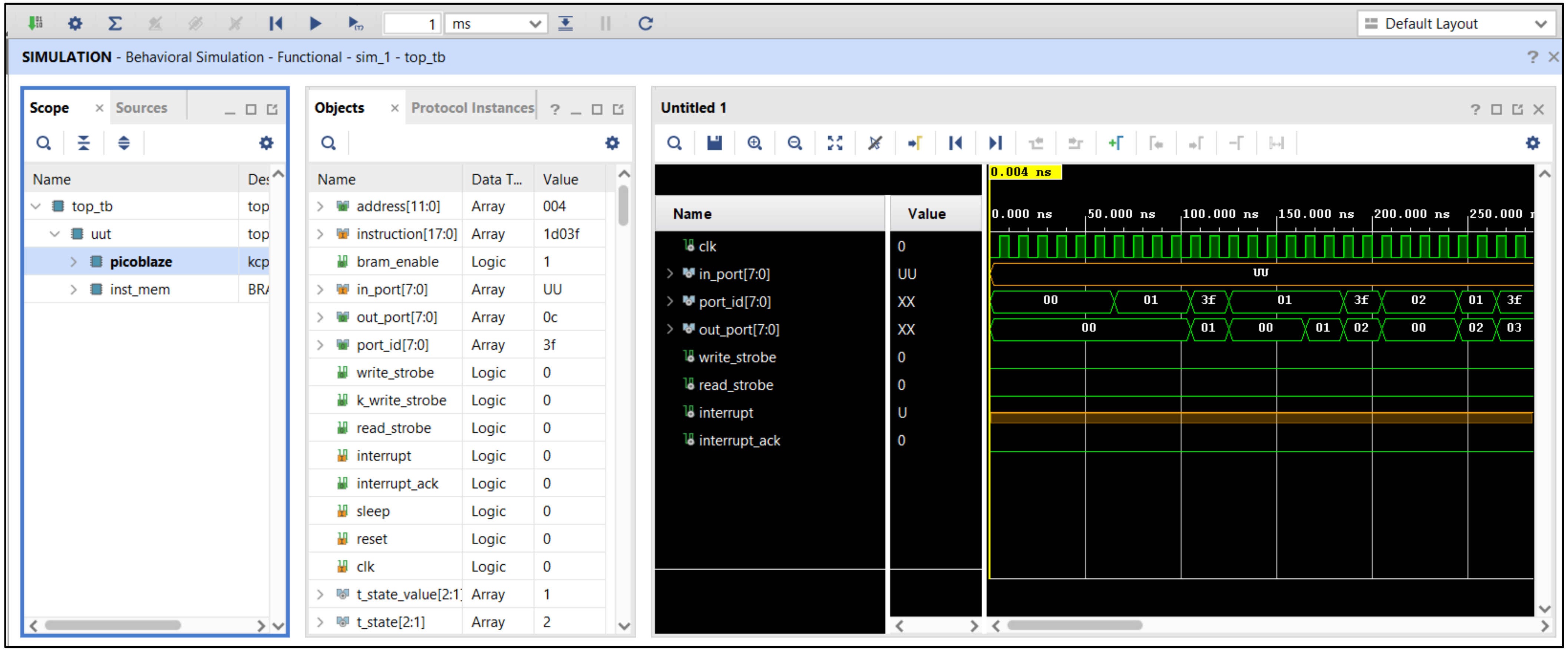
- After adding the design files and **top_tb.vhd** simulation file you can make a simulation for your design.



To add **design** and **simulation** files

Click **Run Behavioral Simulation**

- After running behavioral simulation you can see the waveform of the ports and signals of the picoblaze.



SIMULATION - Behavioral Simulation - Functional - sim_1 - top_tb

Name	Des
top_tb	top
uut	top
picoblaze	kcp
inst_mem	BRA

Name	Data T...	Value
address[11:0]	Array	004
instruction[17:0]	Array	1d03f
bram_enable	Logic	1
in_port[7:0]	Array	UU
out_port[7:0]	Array	0c
port_id[7:0]	Array	3f
write_strobe	Logic	0
k_write_strobe	Logic	0
read_strobe	Logic	0
interrupt	Logic	0
interrupt_ack	Logic	0
sleep	Logic	0
reset	Logic	0
clk	Logic	0
t_state_value[2:1]	Array	1
t_state[2:1]	Array	2

Name	Value
clk	0
in_port[7:0]	UU
port_id[7:0]	XX
out_port[7:0]	XX
write_strobe	0
read_strobe	0
interrupt	U
interrupt_ack	0

```
0.004 ns
0.000 ns   50.000 ns   100.000 ns   150.000 ns   200.000 ns   250.000 ns
in_port:  UU
port_id:  00  01  3f  01  3f  02  01  3f
out_port: 00  01  00  01  02  00  02  03
```

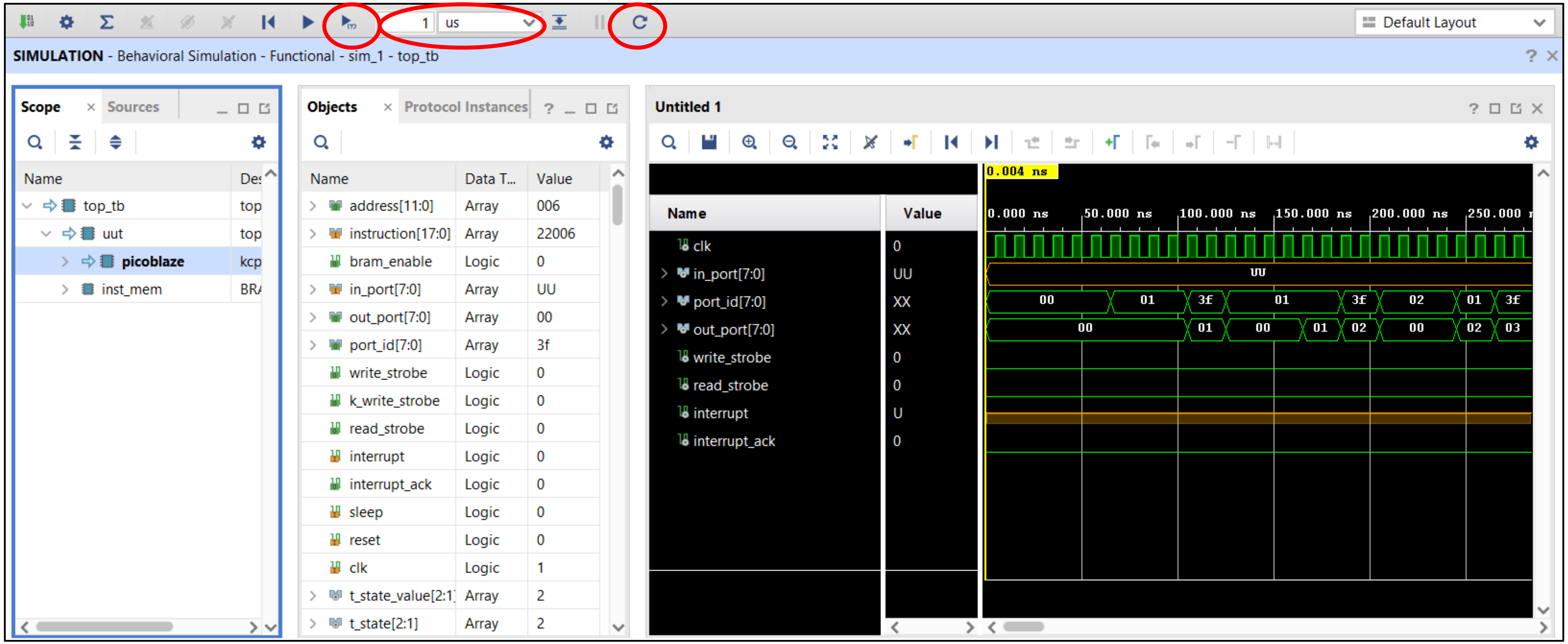
Picoblaze and Inst_memory entities inside the top entity

Ports and Signals

waveform

Advance the simulation(waveform) for 1 us more

Restart the Simulation



1 us

Default Layout

SIMULATION - Behavioral Simulation - Functional - sim_1 - top_tb

Scope | Sources

Name	Des
top_tb	top
uut	top
picoblaze	kcp
inst_mem	BRA

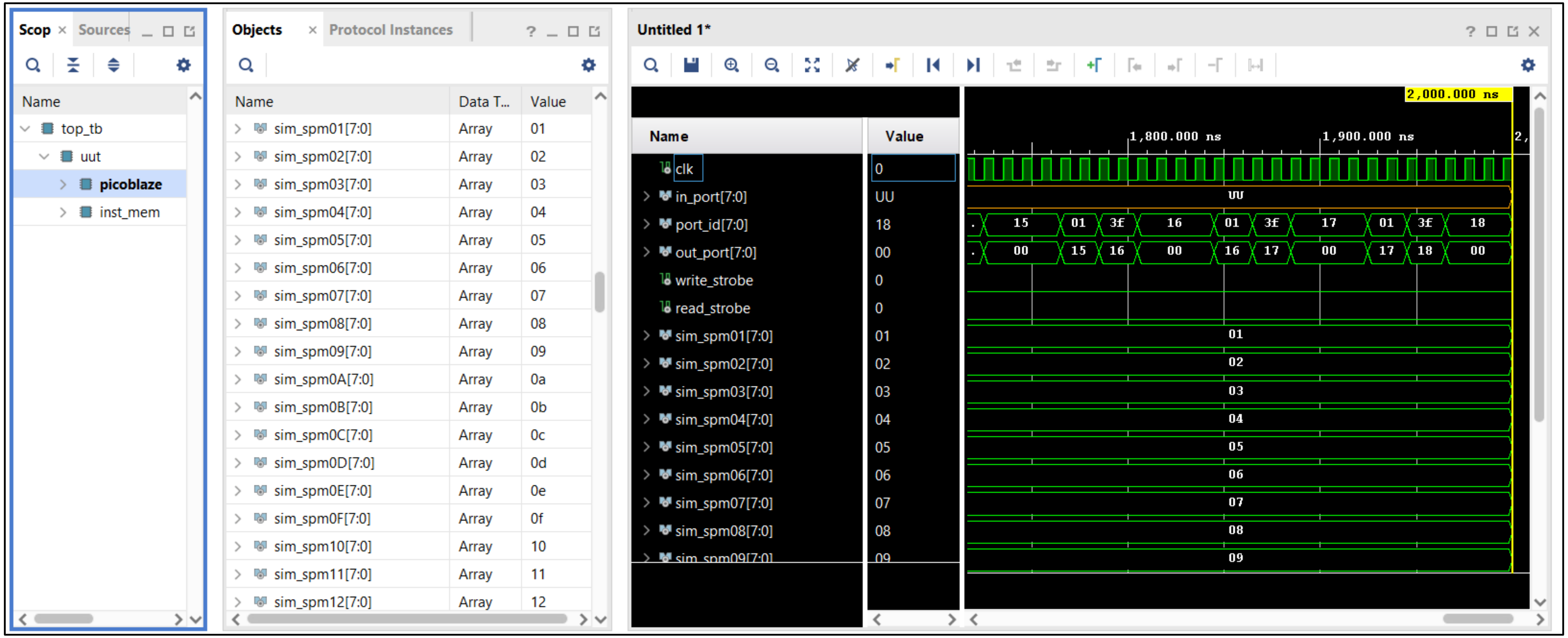
Objects | Protocol Instances

Name	Data T...	Value
address[11:0]	Array	006
instruction[17:0]	Array	22006
bram_enable	Logic	0
in_port[7:0]	Array	UU
out_port[7:0]	Array	00
port_id[7:0]	Array	3f
write_strobe	Logic	0
k_write_strobe	Logic	0
read_strobe	Logic	0
interrupt	Logic	0
interrupt_ack	Logic	0
sleep	Logic	0
reset	Logic	0
clk	Logic	1
t_state_value[2:1]	Array	2
t_state[2:1]	Array	2

Untitled 1

Name	Value
clk	0
in_port[7:0]	UU
port_id[7:0]	XX
out_port[7:0]	XX
write_strobe	0
read_strobe	0
interrupt	U
interrupt_ack	0

- Values of registers, buses, ports, ram etc. can be found from the **scope** and **objects section**
- **Drag** these objects to the **waveform** and **restart** the simulation.



Scope:

- top_tb
  - uut
    - **picoblaze**
    - inst_mem

Name	Data T...	Value
sim_spm01[7:0]	Array	01
sim_spm02[7:0]	Array	02
sim_spm03[7:0]	Array	03
sim_spm04[7:0]	Array	04
sim_spm05[7:0]	Array	05
sim_spm06[7:0]	Array	06
sim_spm07[7:0]	Array	07
sim_spm08[7:0]	Array	08
sim_spm09[7:0]	Array	09
sim_spm0A[7:0]	Array	0a
sim_spm0B[7:0]	Array	0b
sim_spm0C[7:0]	Array	0c
sim_spm0D[7:0]	Array	0d
sim_spm0E[7:0]	Array	0e
sim_spm0F[7:0]	Array	0f
sim_spm10[7:0]	Array	10
sim_spm11[7:0]	Array	11
sim_spm12[7:0]	Array	12

Name	Value
clk	0
in_port[7:0]	UU
port_id[7:0]	18
out_port[7:0]	00
write_strobe	0
read_strobe	0
sim_spm01[7:0]	01
sim_spm02[7:0]	02
sim_spm03[7:0]	03
sim_spm04[7:0]	04
sim_spm05[7:0]	05
sim_spm06[7:0]	06
sim_spm07[7:0]	07
sim_spm08[7:0]	08
sim_spm09[7:0]	09

Spad Memory Values

Spad Memory Values in the time diagram

top_tb.vhd - Testbench File

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

entity top_tb is
end top_tb;

architecture Behavioral of top_tb is
   component top is
    Port ( clk : in STD_LOGIC;
           in_port : in STD_LOGIC_VECTOR (7 downto 0);
           port_id : out STD_LOGIC_VECTOR (7 downto 0);
           out_port : out STD_LOGIC_VECTOR (7 downto 0);
           write_strobe : out STD_LOGIC;
           read_strobe : out STD_LOGIC;
           interrupt : in STD_LOGIC;
           interrupt_ack : out STD_LOGIC );
   end component top;

    signal clk : STD_LOGIC;
    signal in_port : STD_LOGIC_VECTOR(7 downto 0);
    signal port_id : STD_LOGIC_VECTOR(7 downto 0);
    signal out_port : STD_LOGIC_VECTOR(7 downto 0);
    signal write_strobe : STD_LOGIC;
    signal read_strobe : STD_LOGIC;
    signal interrupt : STD_LOGIC;
    signal interrupt_ack : STD_LOGIC;

begin
    uut: top
        port map( clk => clk, in_port => in_port, port_id => port_id, out_port => out_port, write_strobe =>
write_strobe, read_strobe => read_strobe, interrupt => interrupt, interrupt_ack => interrupt_ack );

    process
    begin
        clk <= '0'; wait for 5ns;
        clk <= '1'; wait for 5ns;
    end process;

end Behavioral;
```

Ports of the top file

Port connections

Clock signal 10ns period

top_tb.v - Testbench File

```
`timescale 1ns / 1ps


module top_tb();
    reg clk = 1'b0;
    reg  [7:0] in_port;
    wire [7:0] port_id;
    wire [7:0] out_port;
    wire write_strobe;
    wire read_strobe;
    reg  interrupt;
    wire interrupt_ack;

    always #5 clk = ~clk;

    top UUT(
        .clk(clk),
        .in_port(in_port),
        .port_id(port_id),
        .out_port(out_port),
        .write_strobe(write_strobe),
        .read_strobe(read_strobe),
        .interrupt(interrupt),
        .interrupt_ack(interrupt_ack)
    );

endmodule
```

Clock signal 10ns period

Port connections of the top file

Implementation of the Picoblaze in an FPGA

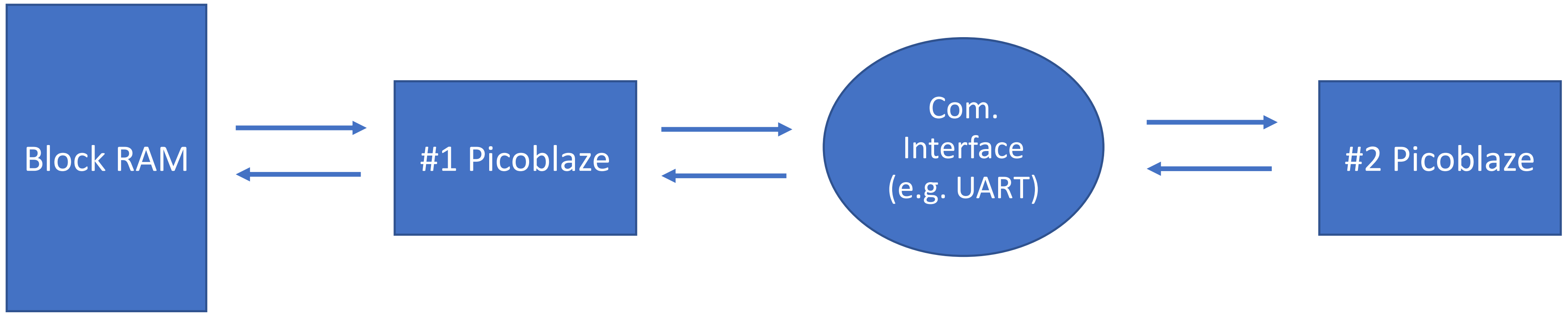
1) Create a new project in Vivado.
2) Take **kcpsm6.vhd** and **BRAM0.vhd** (generated from Fidex IDE) design files and add them as design sources.
3) Take **top.vhd** file that provided for you or write a new one and add it as a **Design Source**.
4) Take **top_tb.vhd** or **top_tb.v** file that provided for you or write a new one and add it as a **Simulation Source**.
5) Run Behavioral Simulation and check the objects and waveform to see your algorithm works properly.

Vivado Block Design

Vivado Block Design

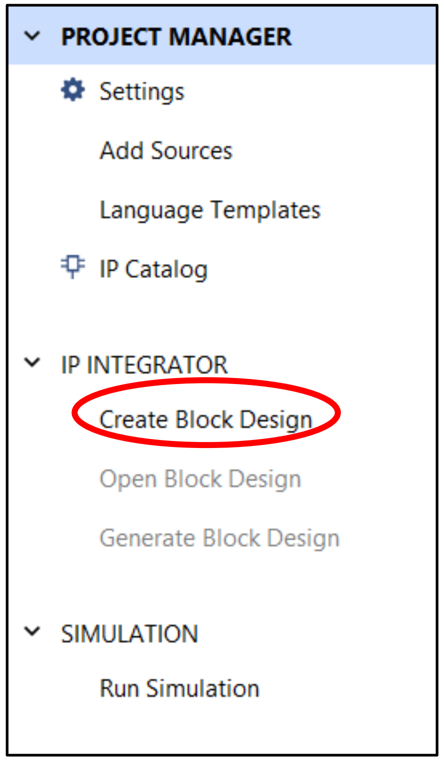
- Any blocks (entities) can be included into a project alongside with the **Picoblaze**.
- You can add **predefined IP Blocks** in Vivado (like **Block RAM**).
- You can also add your own designed entities.

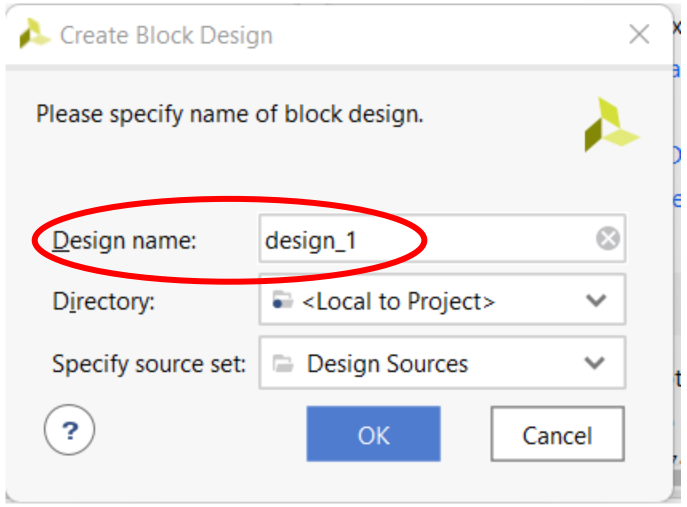
- **An Example:**



Vivado Block Design

- Create a project and add **BRAM0.vhd, kcpsm6.vhd** and **top.vhd** files into this project as Design Sources.
- Then click **"Create Block Design"** .


PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation


Create Block Design

Please specify name of block design.

Design name: design_1

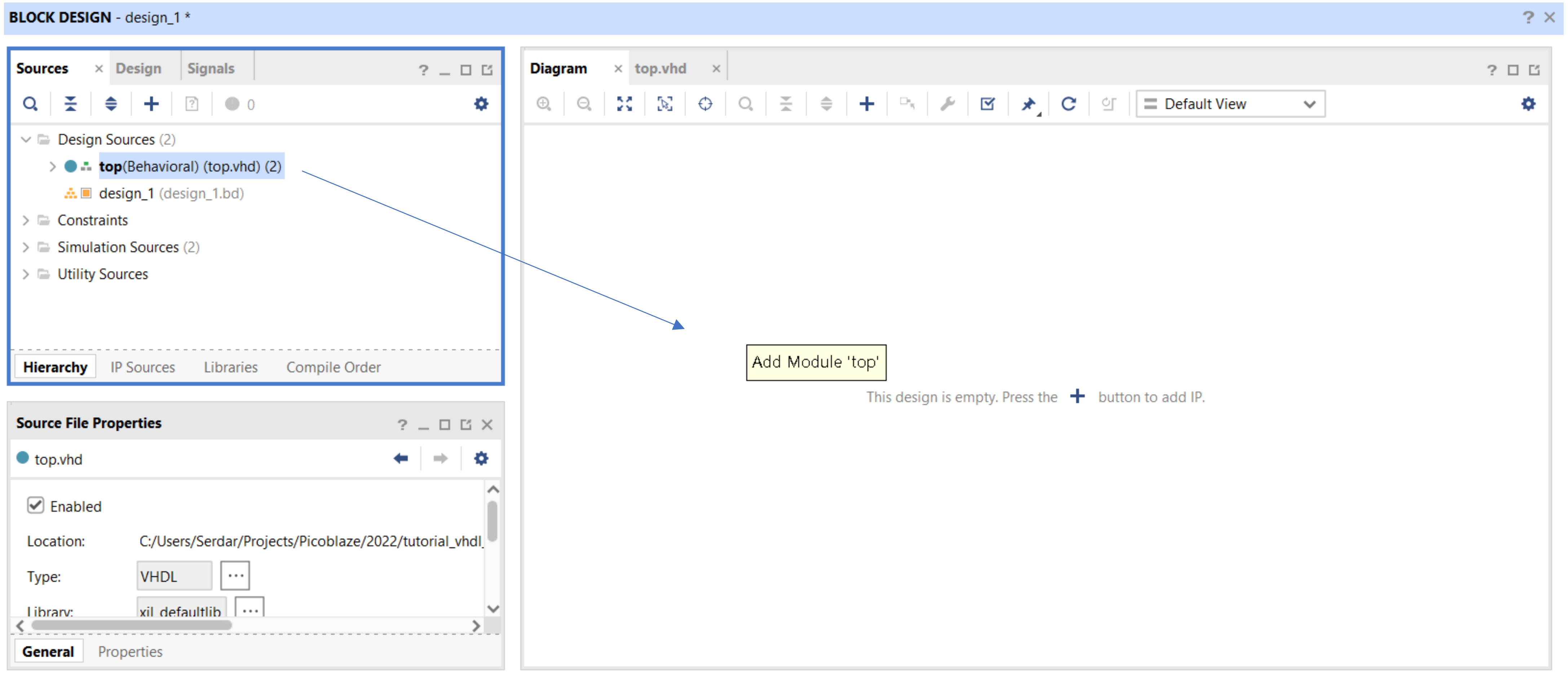
Directory: <Local to Project>

Specify source set: Design Sources

OK Cancel

- Let the design name as **top**.

- Drag **top.vhd** file into the **Diagram** window.

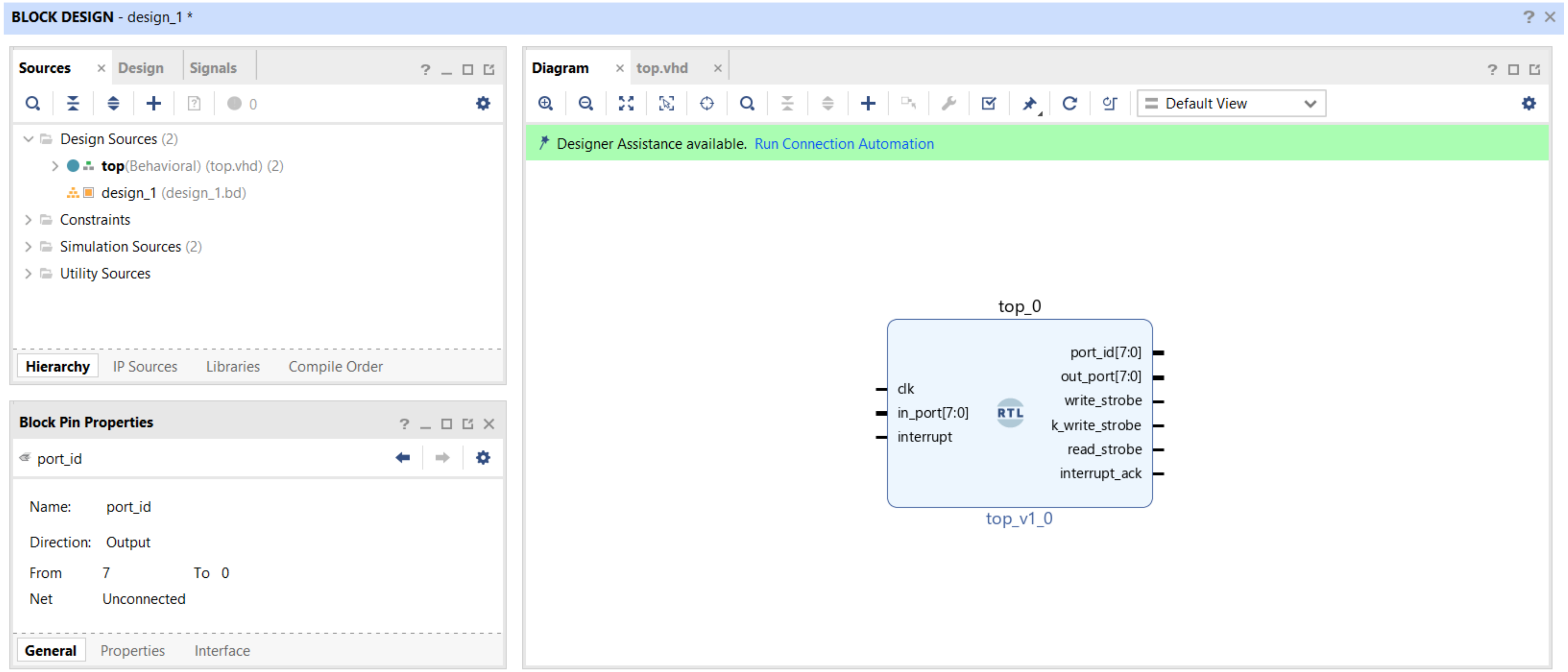


Enabled

Location: C:/Users/Serdar/Projects/Picoblaze/2022/tutorial_vhdl_

Type: VHDL

Library: xil_defaultlib


BLOCK DESIGN - design_1 *

Sources | Design | Signals

Design Sources (2)
- top(Behavioral) (top.vhd) (2)
- design_1 (design_1.bd)

Constraints
Simulation Sources (2)
Utility Sources

Hierarchy | IP Sources | Libraries | Compile Order

Block Pin Properties

port_id

Name:	port_id
Direction:	Output
From	7 To 0
Net	Unconnected

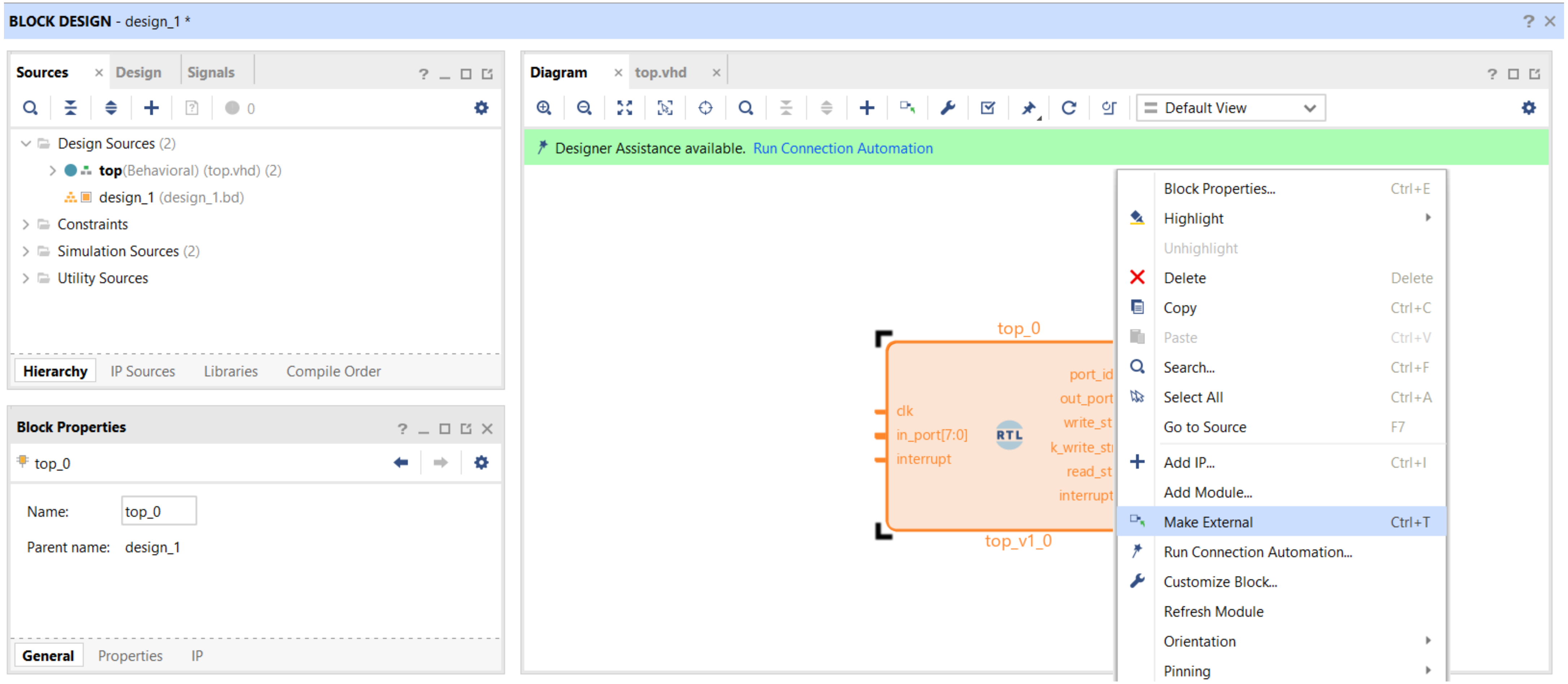
General | Properties | Interface

Diagram | top.vhd

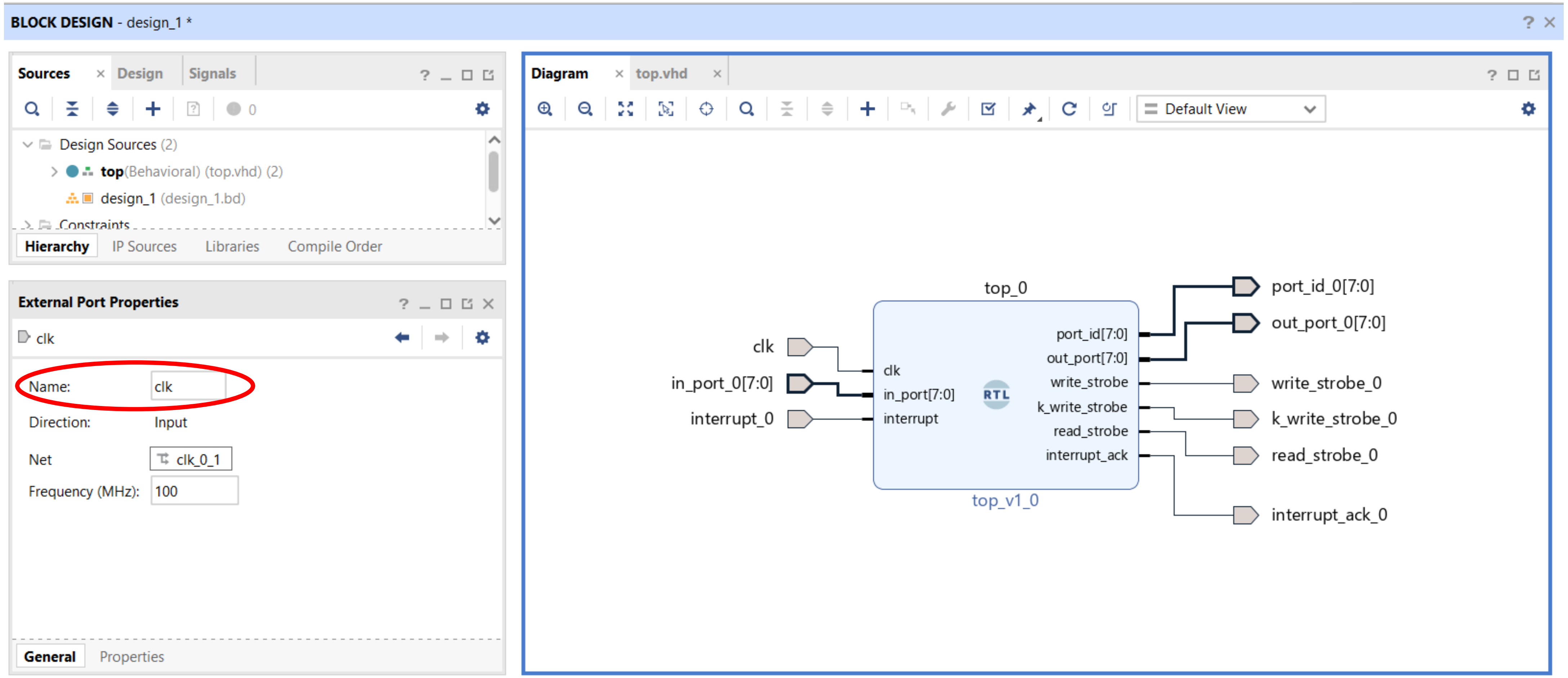
Default View

Designer Assistance available. Run Connection Automation

- To create the ports **"right click->make external"**



- You can change the names of the ports.



BLOCK DESIGN - design_1 *

Sources | Design | Signals

Design Sources (2)
top(Behavioral) (top.vhd) (2)
design_1 (design_1.bd)
Constraints

Hierarchy | IP Sources | Libraries | Compile Order

External Port Properties

clk

Name: clk
Direction: Input
Net: clk_0_1
Frequency (MHz): 100

General | Properties

Diagram | top.vhd | Default View

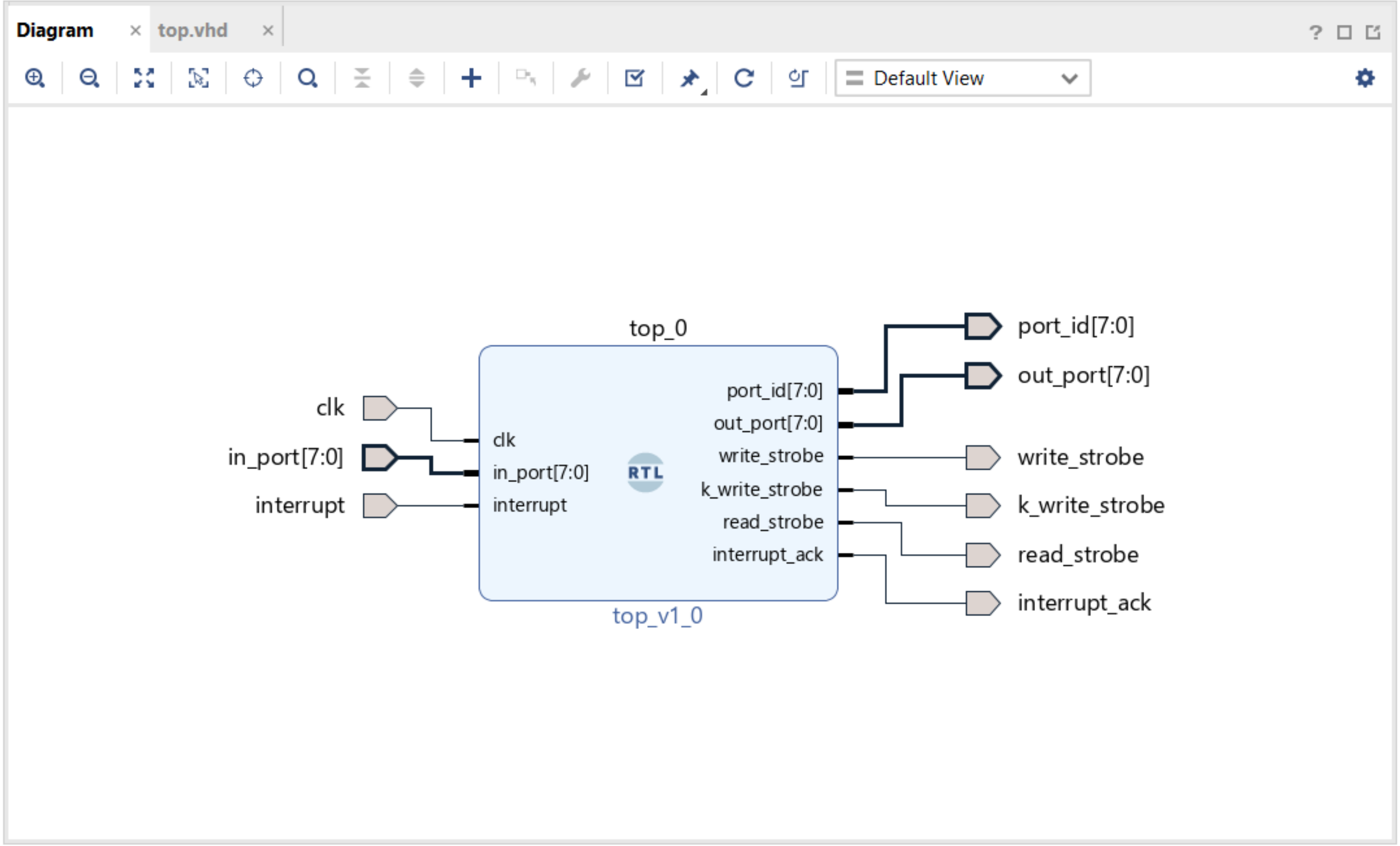
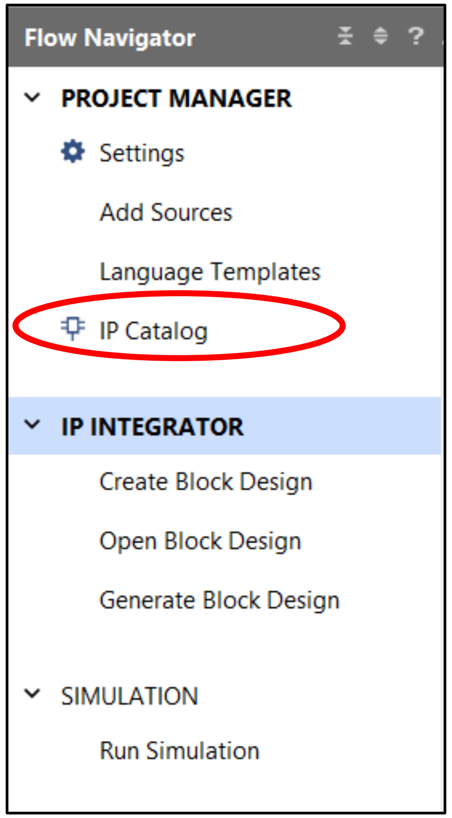
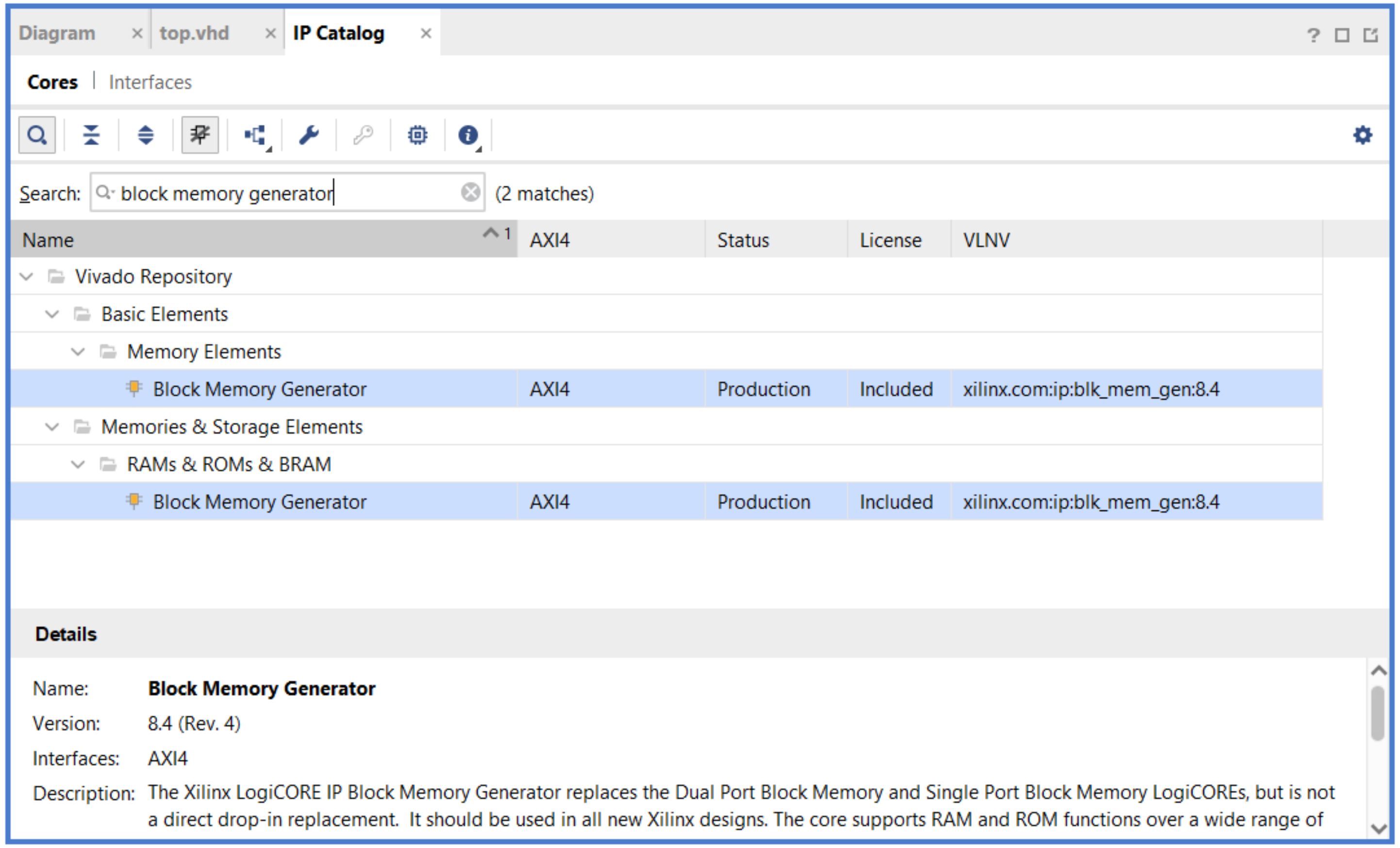


Diagram
top.vhd
Default View
top_0
clk
in_port[7:0]
interrupt
port_id[7:0]
out_port[7:0]
write_strobe
k_write_strobe
read_strobe
interrupt_ack
RTL
top_v1_0

Including a Block RAM

- Click **IP Catalog**
- Type **Block Memory Generator**





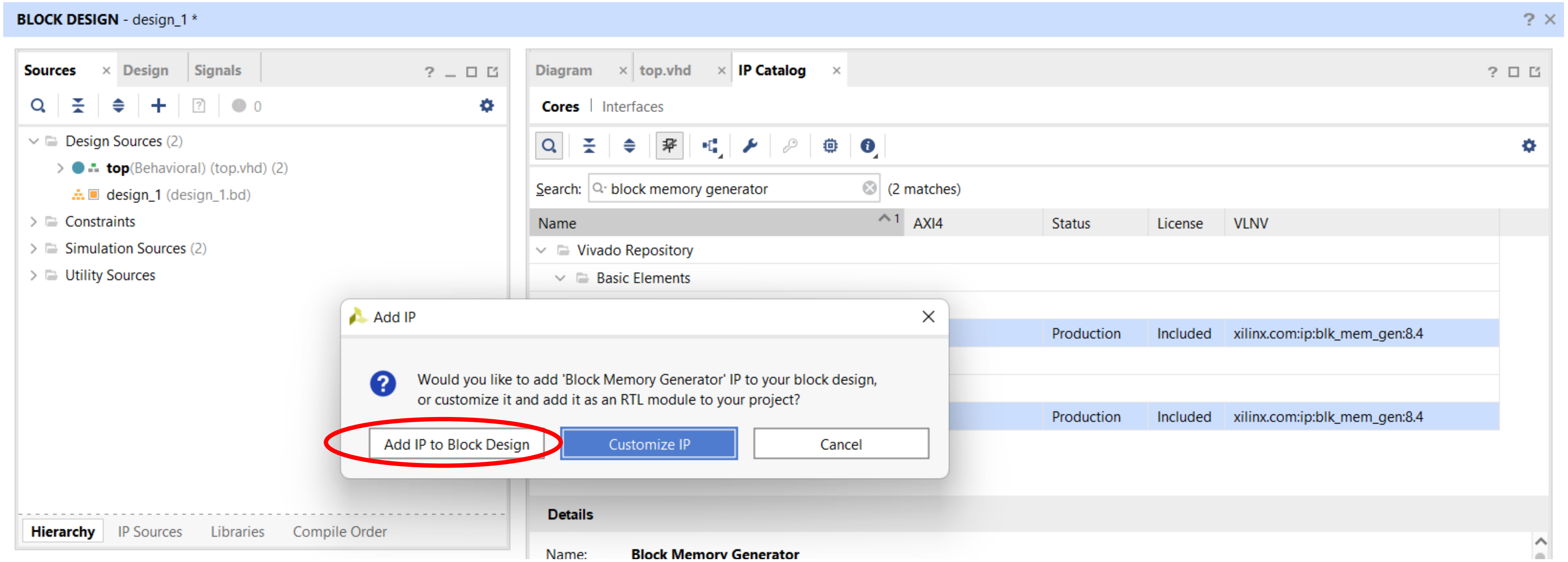
Search: block memory generator (2 matches)

Name	AXI4	Status	License	VLNV
Vivado Repository				
Basic Elements				
Memory Elements				
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip:blk_mem_gen:8.4
Memories & Storage Elements				
RAMs & ROMs & BRAM				
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip:blk_mem_gen:8.4

Details

Name: **Block Memory Generator**
Version: 8.4 (Rev. 4)
Interfaces: AXI4
Description: The Xilinx LogiCORE IP Block Memory Generator replaces the Dual Port Block Memory and Single Port Block Memory LogiCOREs, but is not a direct drop-in replacement. It should be used in all new Xilinx designs. The core supports RAM and ROM functions over a wide range of

- **Add IP to Block Design**



BLOCK DESIGN - design_1 *

Search: block memory generator (2 matches)

Name	AXI4	Status	License	VLNV
Vivado Repository				
Basic Elements				
		Production	Included	xilinx.com:ip:blk_mem_gen:8.4
		Production	Included	xilinx.com:ip:blk_mem_gen:8.4

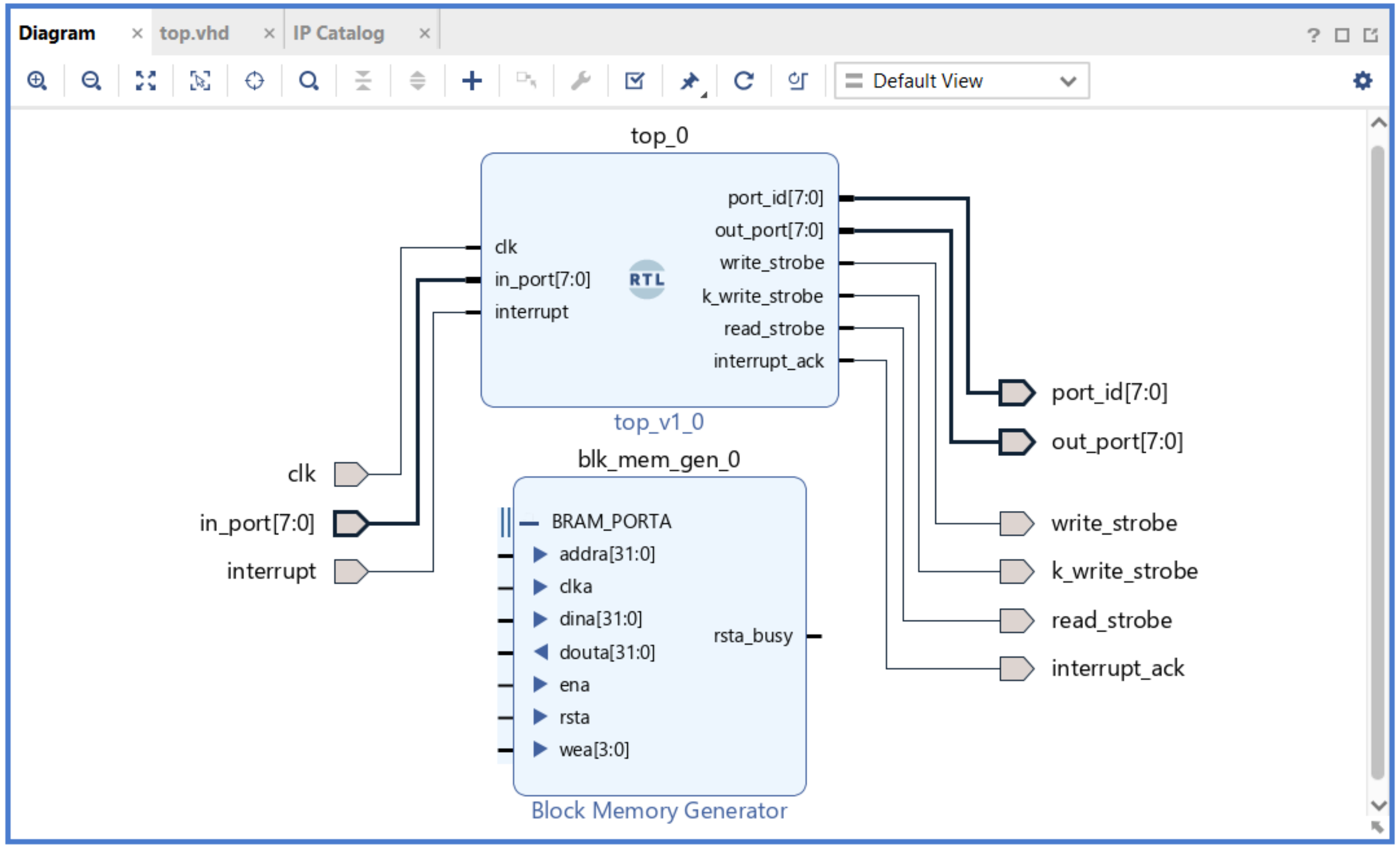
Add IP

Would you like to add 'Block Memory Generator' IP to your block design, or customize it and add it as an RTL module to your project?

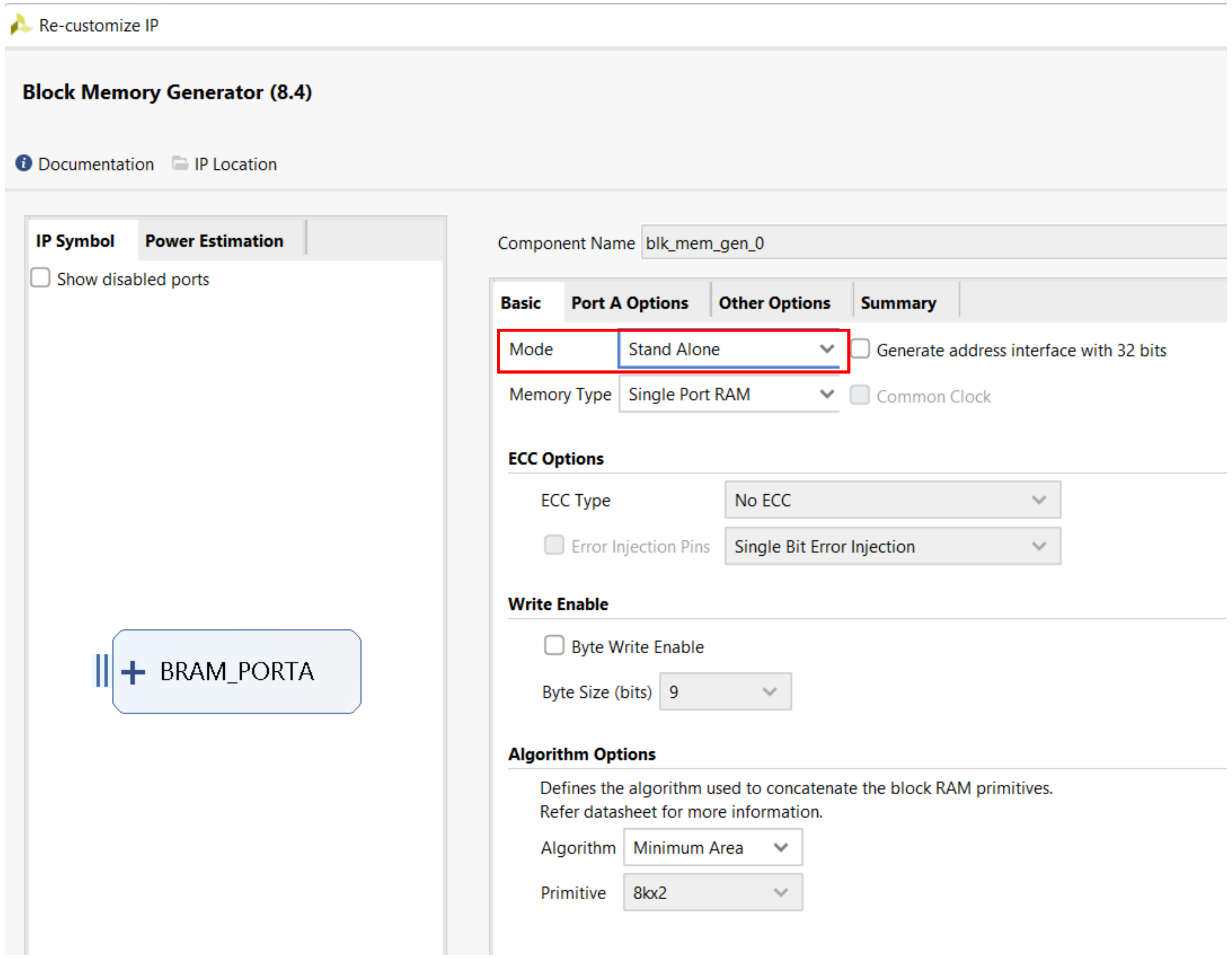
Details

Name: Block Memory Generator

- Double click to **customize** Block RAM



- Change the default specifications:


Re-customize IP

Block Memory Generator (8.4)

Documentation IP Location

Component Name: blk_mem_gen_0

Mode: Stand Alone
Generate address interface with 32 bits: ☐
Memory Type: Single Port RAM
Common Clock: ☐

ECC Options

ECC Type: No ECC
Error Injection Pins: ☐ Single Bit Error Injection

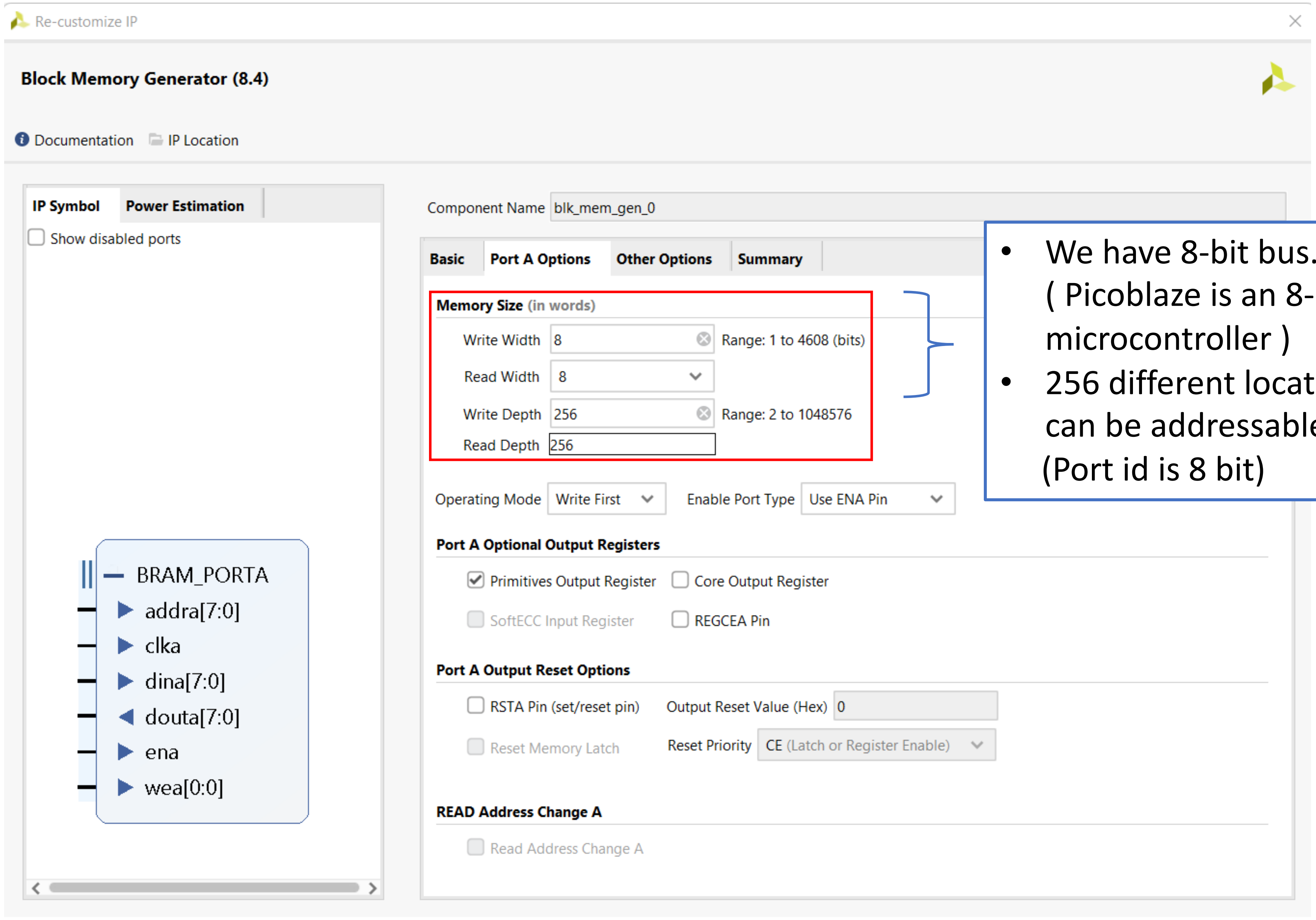
Write Enable

Byte Write Enable: ☐
Byte Size (bits): 9

Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.

Algorithm: Minimum Area
Primitive: 8kx2



Re-customize IP

Block Memory Generator (8.4)

Documentation   IP Location

IP Symbol | Power Estimation

☐ Show disabled ports

Component Name: blk_mem_gen_0

Basic | Port A Options | Other Options | Summary

Memory Size (in words)

Write Width	8	Range: 1 to 4608 (bits)
Read Width	8	
Write Depth	256	Range: 2 to 1048576
Read Depth	256	

Operating Mode: Write First   Enable Port Type: Use ENA Pin

Port A Optional Output Registers

☑ Primitives Output Register   ☐ Core Output Register

☐ SoftECC Input Register   ☐ REGCEA Pin

Port A Output Reset Options

☐ RSTA Pin (set/reset pin)   Output Reset Value (Hex): 0

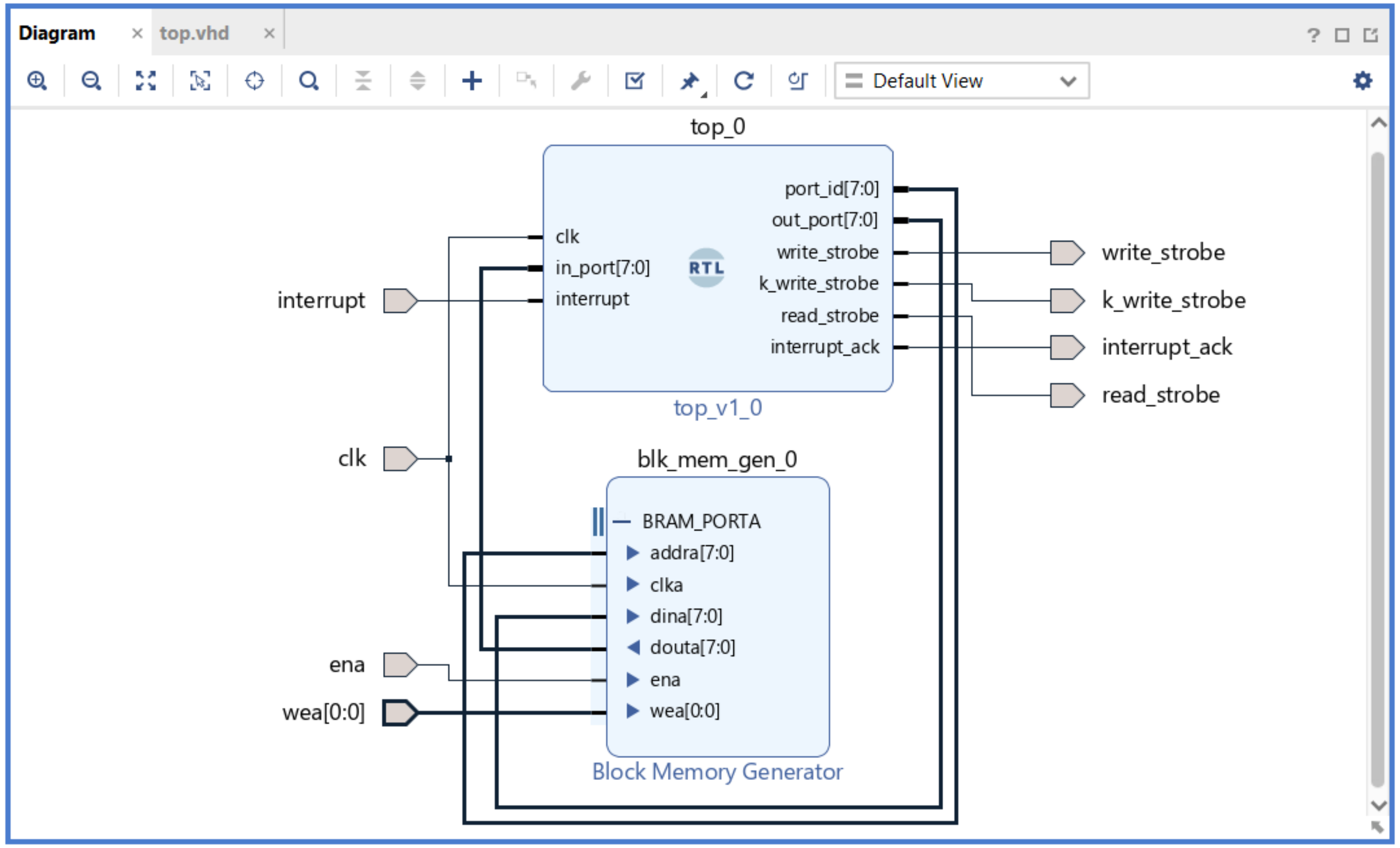
☐ Reset Memory Latch   Reset Priority: CE (Latch or Register Enable)

READ Address Change A

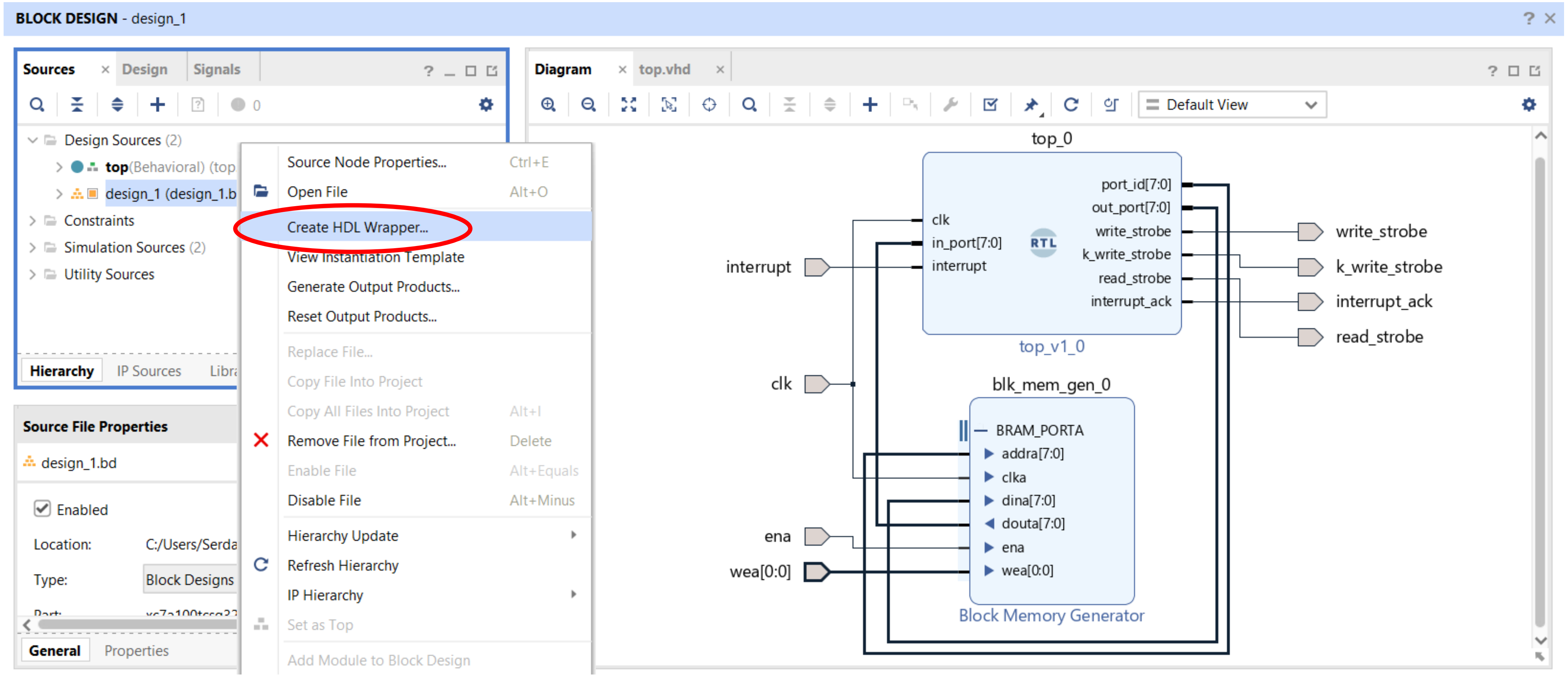
☐ Read Address Change A

- We have 8-bit bus. (Picoblaze is an 8-bit microcontroller)
- 256 different locations can be addressable. (Port id is 8 bit)

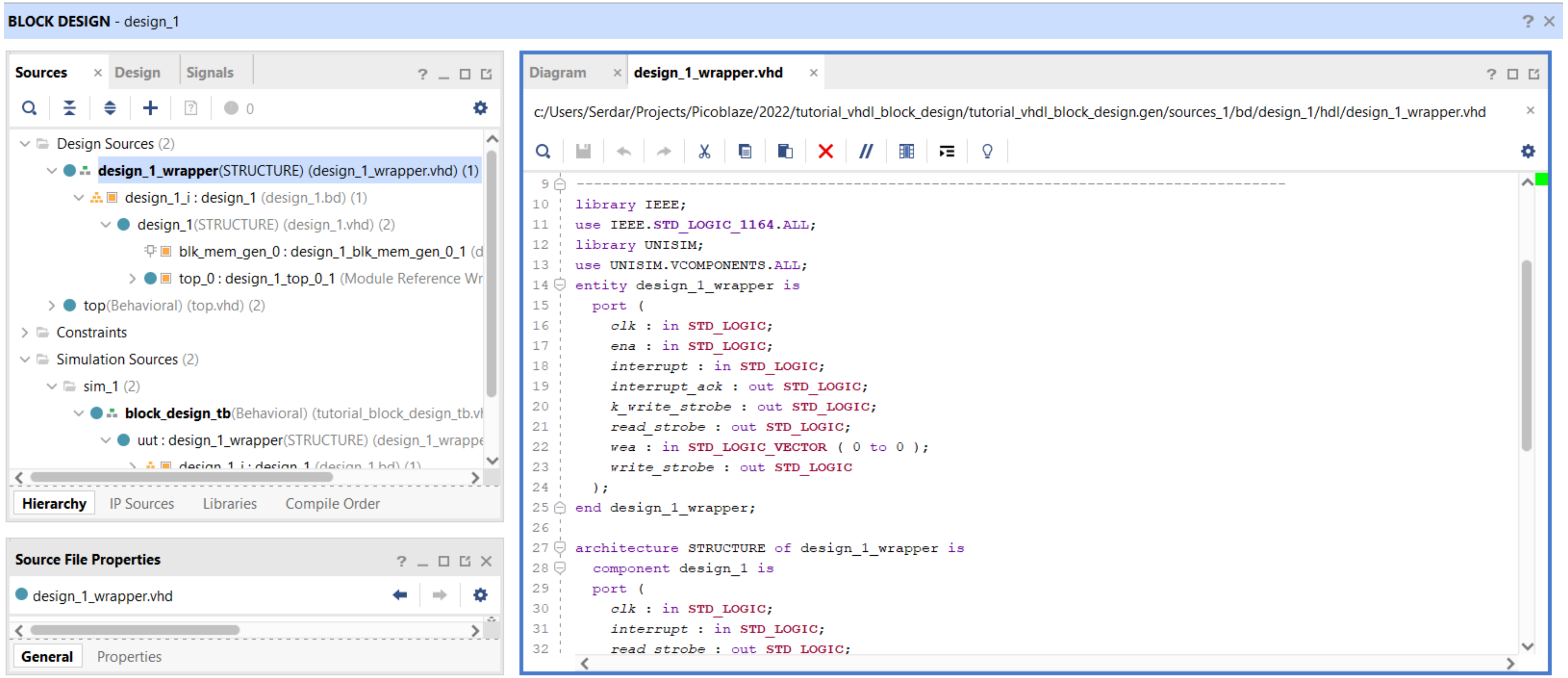
- After customizing Block RAM, make the following connections:
  - **port_id** to **addra**
  - **out_port** to **dina**
  - **in_port** to **douta**
- Create **ena** and **wea** ports.



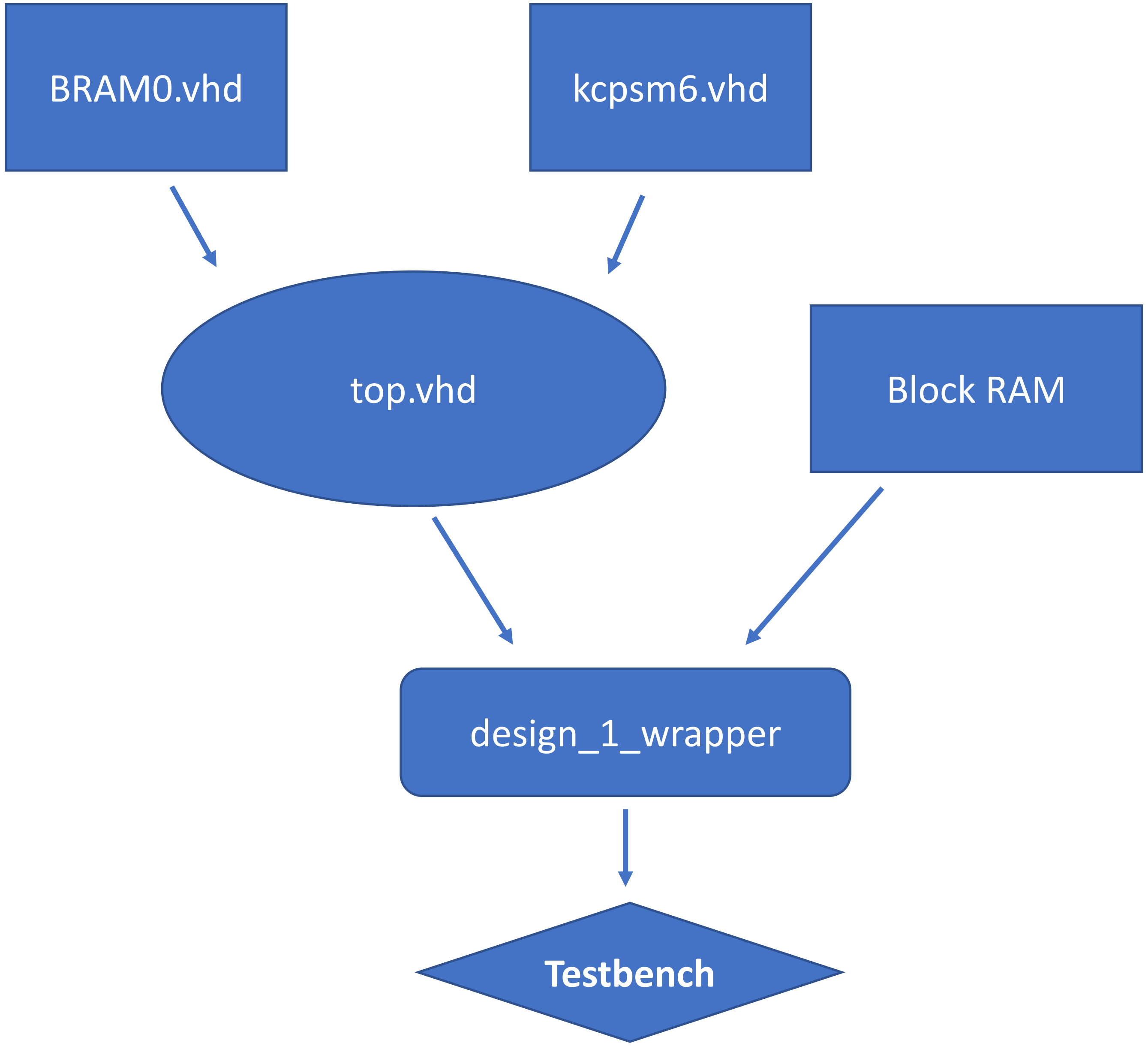
- **Right click on design_1 -> Create HDL Wrapper**



- A top entity called **design_1_wrapper** is generated by Vivado.
- **design_1_wrapper** combines the **top** (picoblaze + inst memory) and **Block RAM**.


BLOCK DESIGN - design_1
Sources
Design
Signals
Design Sources (2)
design_1_wrapper(STRUCTURE) (design_1_wrapper.vhd) (1)
design_1_i : design_1 (design_1.bd) (1)
design_1(STRUCTURE) (design_1.vhd) (2)
blk_mem_gen_0 : design_1_blk_mem_gen_0_1 (d
top_0 : design_1_top_0_1 (Module Reference Wr
top(Behavioral) (top.vhd) (2)
Constraints
Simulation Sources (2)
sim_1 (2)
block_design_tb(Behavioral) (tutorial_block_design_tb.vh
uut : design_1_wrapper(STRUCTURE) (design_1_wrappe
Hierarchy
IP Sources
Libraries
Compile Order
Source File Properties
design_1_wrapper.vhd
General
Properties
Diagram
design_1_wrapper.vhd
c:/Users/Serdar/Projects/Picoblaze/2022/tutorial_vhdl_block_design/tutorial_vhdl_block_design.gen/sources_1/bd/design_1/hdl/design_1_wrapper.vhd

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity design_1_wrapper is
  port (
    clk : in STD_LOGIC;
    ena : in STD_LOGIC;
    interrupt : in STD_LOGIC;
    interrupt_ack : out STD_LOGIC;
    k_write_strobe : out STD_LOGIC;
    read_strobe : out STD_LOGIC;
    wea : in STD_LOGIC_VECTOR ( 0 to 0 );
    write_strobe : out STD_LOGIC
  );
end design_1_wrapper;

architecture STRUCTURE of design_1_wrapper is
  component design_1 is
  port (
    clk : in STD_LOGIC;
    interrupt : in STD_LOGIC;
    read_strobe : out STD_LOGIC;
```


BRAM0.vhd
kcpsm6.vhd
top.vhd
Block RAM
design_1_wrapper
Testbench

- Write a **testbench** file and add it as a **Simulation Source File**.

block_design_verilog_tb.v - Testbench File

```
`timescale 1ns / 1ps


module block_design_verilog_tb();
    reg  clk = 1'b0;
    reg  ena;
    reg  [0:0] wea;
    reg  interrupt;
    wire interrupt_ack;
    wire read_strobe;
    wire write_strobe;
    wire k_write_strobe;

    always #5 clk = ~clk;

    design_1_wrapper UUT(
        .clk(clk),
        .ena(ena),
        .wea(wea),
        .interrupt(interrupt),
        .interrupt_ack(interrupt_ack),
        .read_strobe(read_strobe),
        .write_strobe(write_strobe),
        .k_write_strobe(k_write_strobe)
    );

endmodule
```

Clock signal 10ns period

Port connections of the **design_1_wrapper** file

block_design_tb.vhd - Testbench File

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

entity block_design_tb is
end block_design_tb;

architecture Behavioral of block_design_tb is
   component design_1_wrapper is
    port (  clk : in STD_LOGIC;
            ena : in STD_LOGIC;
            interrupt : in STD_LOGIC;
            interrupt_ack : out STD_LOGIC;
            k_write_strobe : out STD_LOGIC;
            read_strobe : out STD_LOGIC;
            wea : in STD_LOGIC_VECTOR ( 0 to 0 );
            write_strobe : out STD_LOGIC
           );
   end component design_1_wrapper;

    signal clk : STD_LOGIC;
    signal ena : STD_LOGIC;
    signal interrupt : STD_LOGIC;
    signal interrupt_ack : STD_LOGIC;
    signal k_write_strobe : STD_LOGIC;
    signal read_strobe : STD_LOGIC;
    signal wea : STD_LOGIC_VECTOR ( 0 to 0 );
    signal write_strobe : STD_LOGIC;

begin
    uut: design_1_wrapper
        port map( clk => clk, ena => ena, interrupt => interrupt, interrupt_ack => interrupt_ack,
k_write_strobe => k_write_strobe, read_strobe => read_strobe, wea => wea, write_strobe => write_strobe);

    process
    begin
        clk <= '0'; wait for 5ns;
        clk <= '1'; wait for 5ns;
    end process;

end Behavioral;
```

Ports of the **desing_1_wrapper** file

Port connections

Clock signal 10ns period

References

- Picoblaze user guide