Nexys 4 DDR External Memory

Acknowledgement

This tutorial is derived from a tutorial from Digilent Inc.

Goal

- Use IP integrator to connect and configure the external memory controller
- Be able to use a C program in SDK to interact with the external memory
- Use some software debugging tools in an embedded processor environment.

Requirements

- Xilinx Vivado software
- Xilinx SDK software
- Xilinx Nexys 4 DDR board and a programming cable
- Nexys 4 Board Files
- Enough disk space for the project files

Background

The external memory on the The Nexys4 DDR board is available in two forms: volatile DRAM and non-volatile flash. This tutorial will focus on using the faster DRAM module as an AXI peripheral.

Since the DRAM is external to the FPGA, logic must be created to communicate with the memory in the form of a **memory controller**. This memory controller generates a clock for the RAM as well as manages the transfer of data on both edges of the clock for double data-rate operation. In addition, it presents a simple interface for the rest of the system to access the memory.

The Micron MT47H64M16HR-25 DDR2 module on the Nexys 4 DDR is a 1 Gb (giga*bit*) or 128 MB (mega*byte*) memory with 16-bit wide data bus. This provides much more memory capacity than is available on the FPGA itself, but the bandwidth is somewhat limited since all data must pass through the 16 data pins connecting the two devices.

1. Import the Board Package

The Xilinx memory interface generator is highly configurable and requires detailed information about the memory to function correctly. To simplify this process, the settings have been packaged in a board file by Digilent.

1. Unzip the board_repository.zip file.

2. Invoke the Vivado IDE.

3. Bring up the **Tcl Console** at the bottom of the window and enter:

set_param board.repoPaths <path-to-board-files>/board_files/

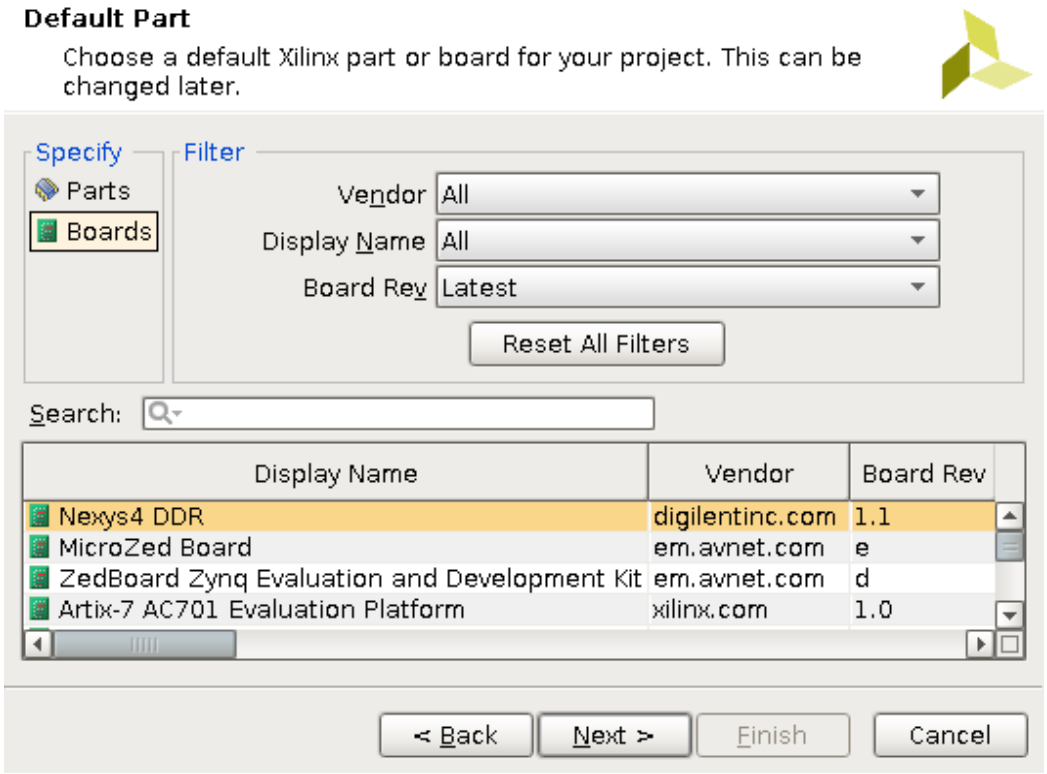
replacing <path-to-board-repository> with the location of the unzipped directory. The board file will have all the pin connections defined for the switches, LEDs, memories, etc. defined so that you do not have to manually build the constraints file to define the pin locations.

2. Create a Project

1. Create a **New Project.**

2. At the **Default Part** dialog, specify **Boards** and select the **Nexys 4 DDR** board.

3. **Finish** creating the project.



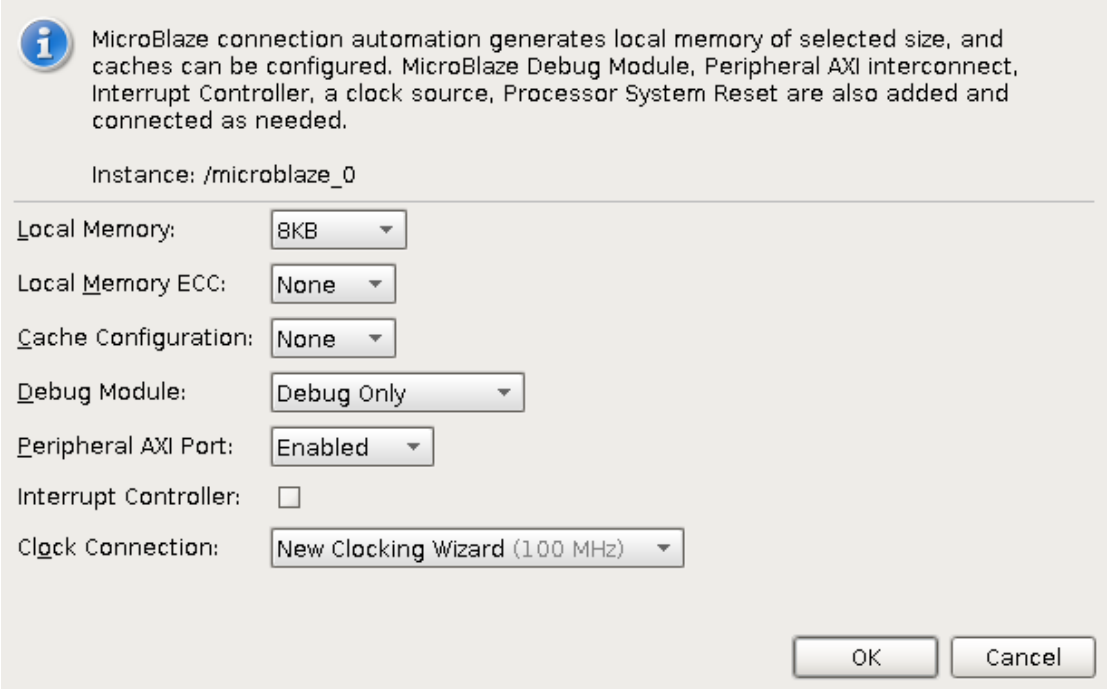
3. Create an IP Integrator Design

1. From Naviator > IP Integrator, select **Create Block Design.**

Basic MicroBlaze System

1. Right click anywhere in the Diagram and select **Add IP** and add a MicroBlaze block to the design

2. **Run Block Automation** for the MicroBlaze and use the default settings and press OK

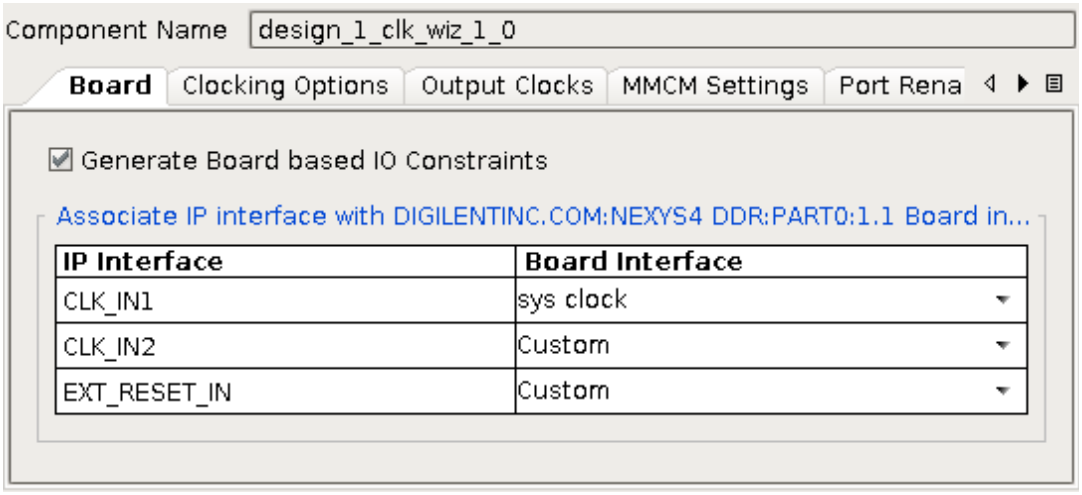


Clock Customization

The processing system will operate at 100 MHz, but the memory controller requires a 200 MHz input clock to generate the appropriate clock for the external DRAM. We will create this next.

1. Double click the Clock Wizard (clk_wiz_1) block to re-customize it.

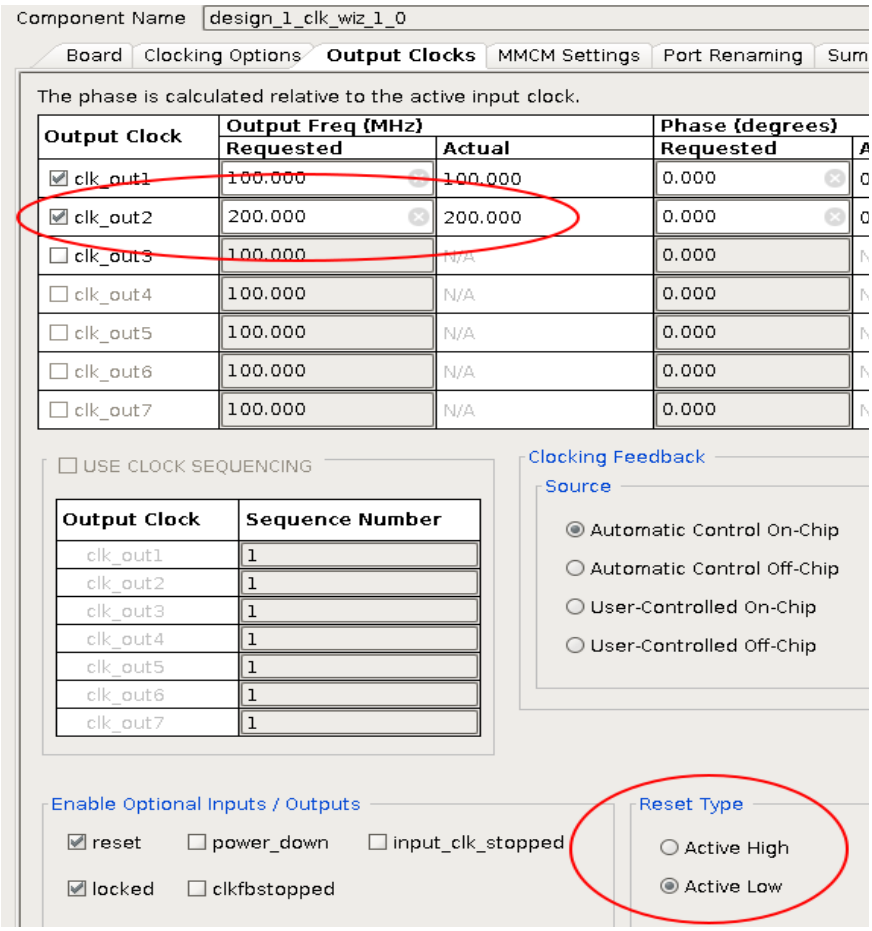
2. Under the **Board** tab, select **Generate Board based IO Constraints**. Use the **Board Interface** pull-down menu for IP Interface **CLK_IN1** and select **sys clock.**



4. Select the **Output Clocks** tab.

5. Check the radio box for Output Clock **clock_out2** and enter a requested clock frequency of **200.**

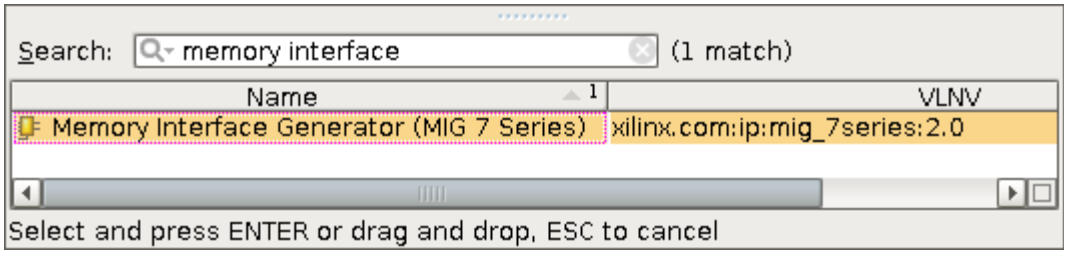
6. While in the Output Clocks tab, change the **Reset Type** to **Active Low.**



UART and Memory Controller

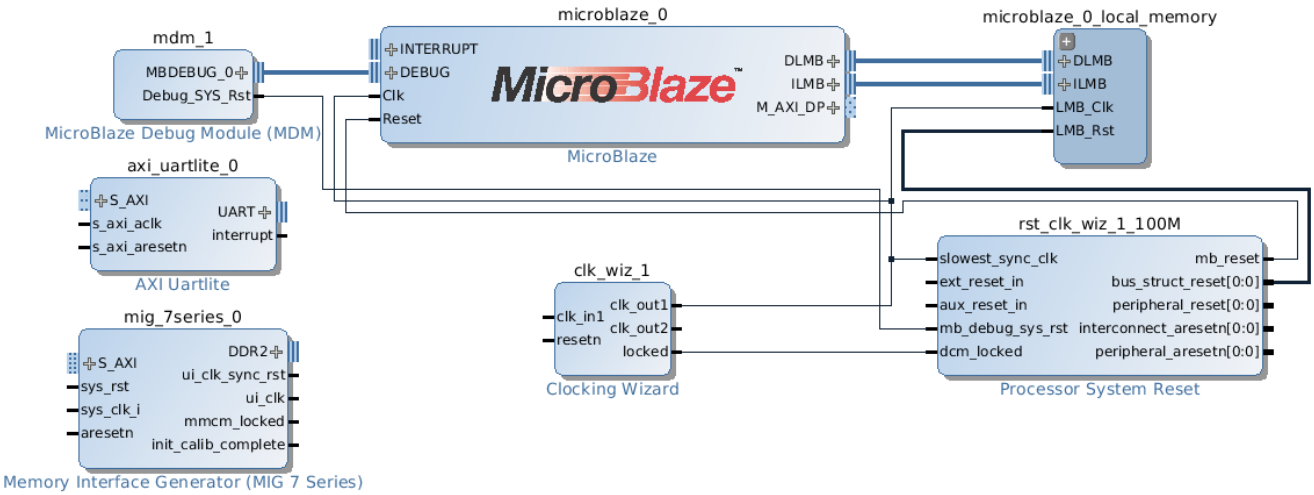
1. Right click anywhere in the Diagram and select **Add IP** and add an **AXI Uartlite** block to the design.

2. Repeat the process to search for and add a **Memory Interface Generator (MIG)** peripheral.



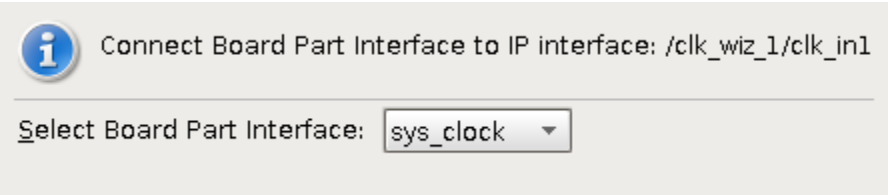
3. **Run Block Automation** for the memory controller (/mig_7series_0) component. This configures the block for the DRAM on the Nexys 4 DDR board. Error [BD 41-1273] may appear during automation, but can be safely ignored.

At this point you should have a design similar to:



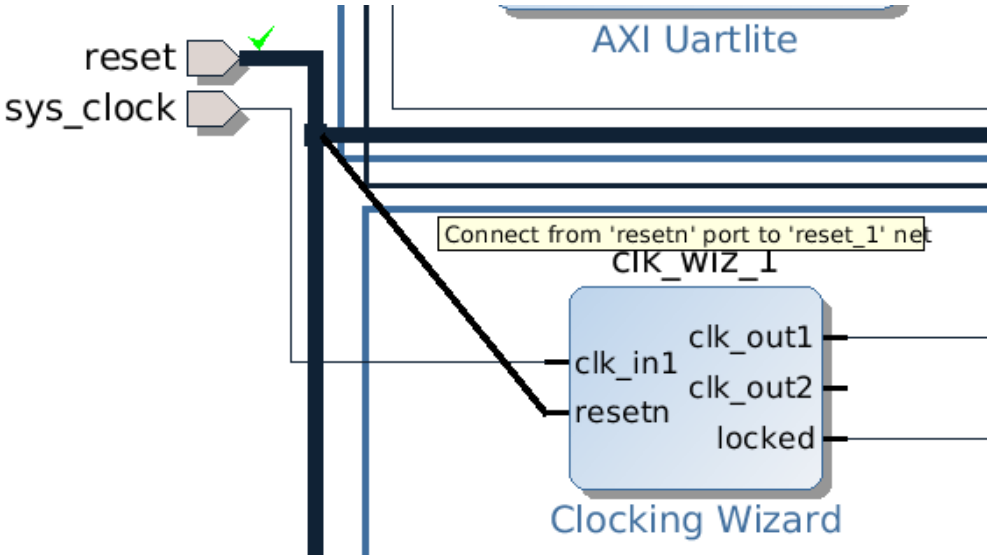
Making Connections

1. **Run Connection Automation** for clk_wiz_1/clk_in1 and choose **sys_clock**.

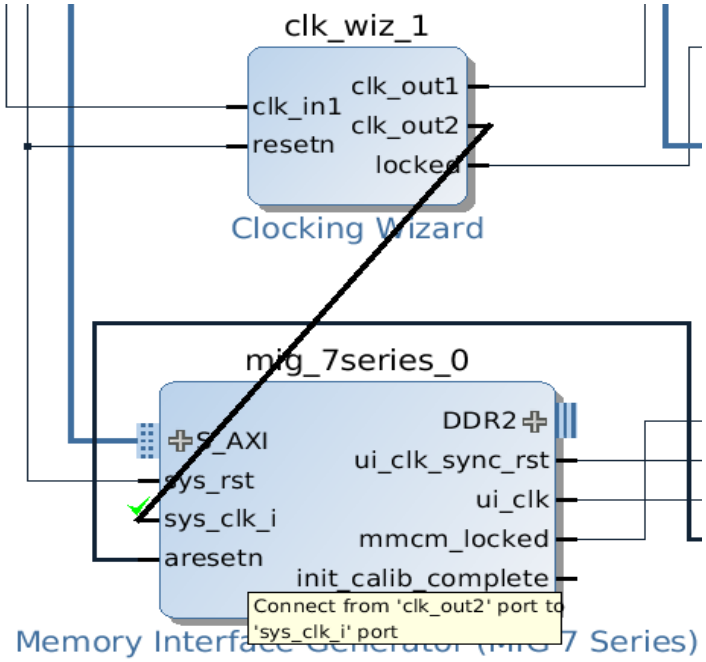


2. **Run Connection Automation** for /rst_clk_wiz_1_100M/ext_reset_in and choose **reset**.
3. **Run Connection Automation** for /axi_uartlite_0/S_AXI and leave the Clock Connection to **auto**.
4. **Run Connection Automation** for /axi_uartlite_0/UART and select **USB_Uart**.
5. **Run Connection Automation** for /mig_7series_0/S_AXI and leave the Clock Connection to **auto**.
6. **Run Connection Automation** for /mig_7series_0/sys_rst and select **reset**.
7. **Run Connection Automation** for /rst_clk_wiz_100M/ext_reset_in and select **reset**.

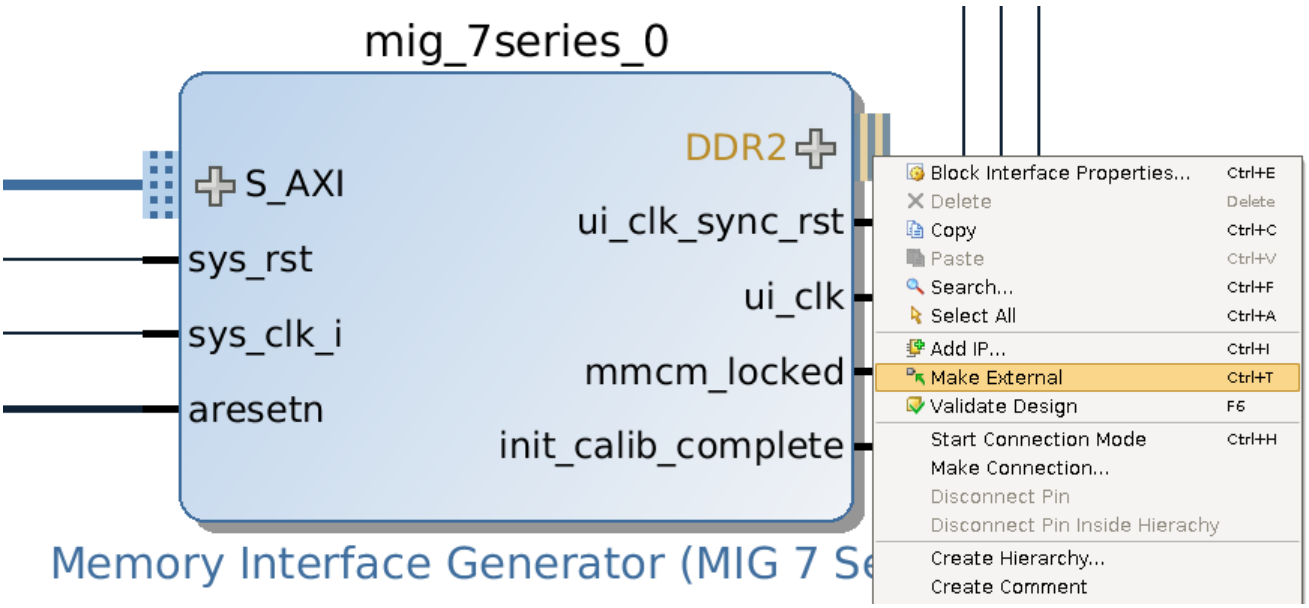
8. If not connected, connect the **resetn** of the clock wizard to the **reset** on the diagram. You can drag a wire from the pin to the **reset** signal.



9. Connect the **clk_out2** of the clock wizard to the **sys_clk_i** of the memory interface generator.



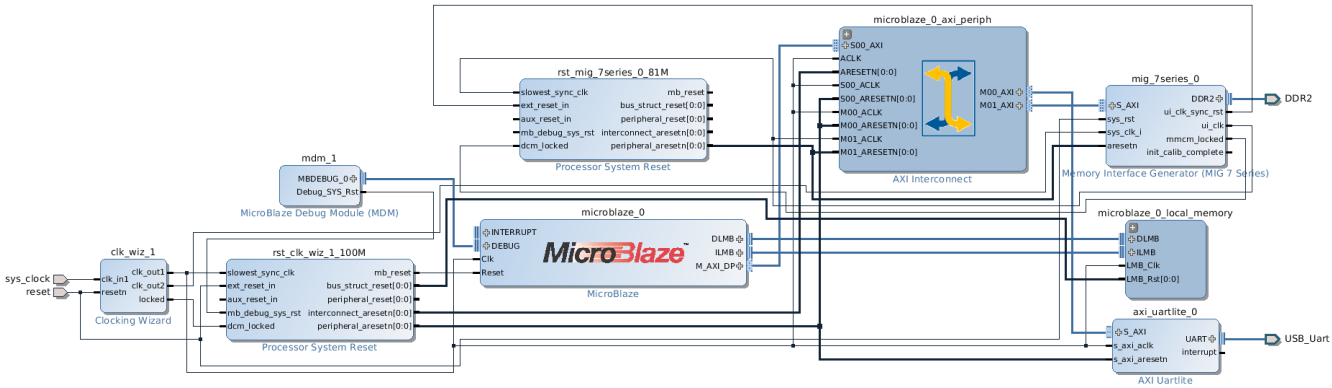
10. Select the **DDR2** bus on the memory controller, right click and select **Make External**. To propagate these signals to the pins of the FPGA.



Build the bitstream

1. Click on to validate your design

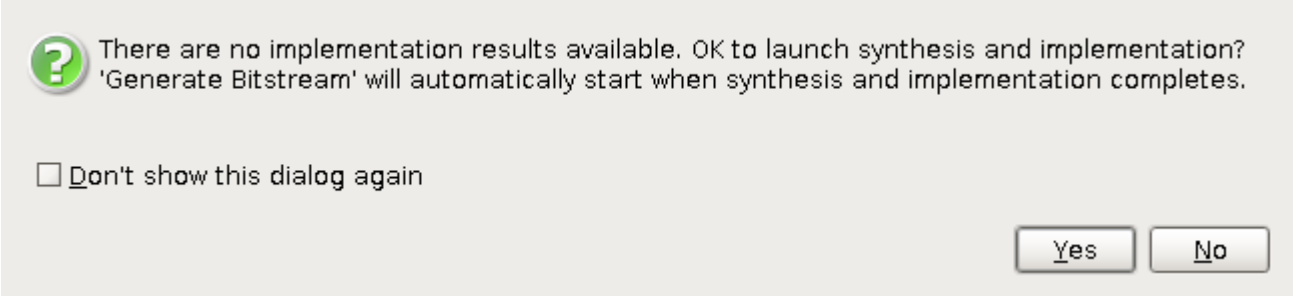
Your design show resemble the following:



2. In the **Sources** box, click **IP Sources.** Right click your design and choose **Create HDL Wrapper**.

Let Vivado manage the wrapper and auto-update.

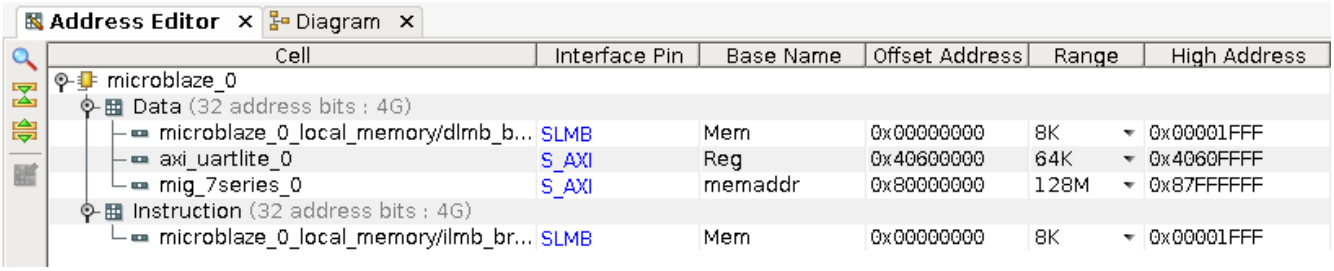
3. Click on **Generate Bitstream**. If you haven't saved the project yet, a dialog will appear prompting you to save. A second dialog will appear stating that no implementation results are available, click Yes to run through synthesis, implementation and bitstream generation.



Note: Warnings during implementation about the pin constraints may be safely ignored.

4. Test the Memory

The connection automation steps mapped the AXI peripherals on to the address space of the MicroBlaze processor. This can be viewed in the **Address Editor** tab of IP Integrator



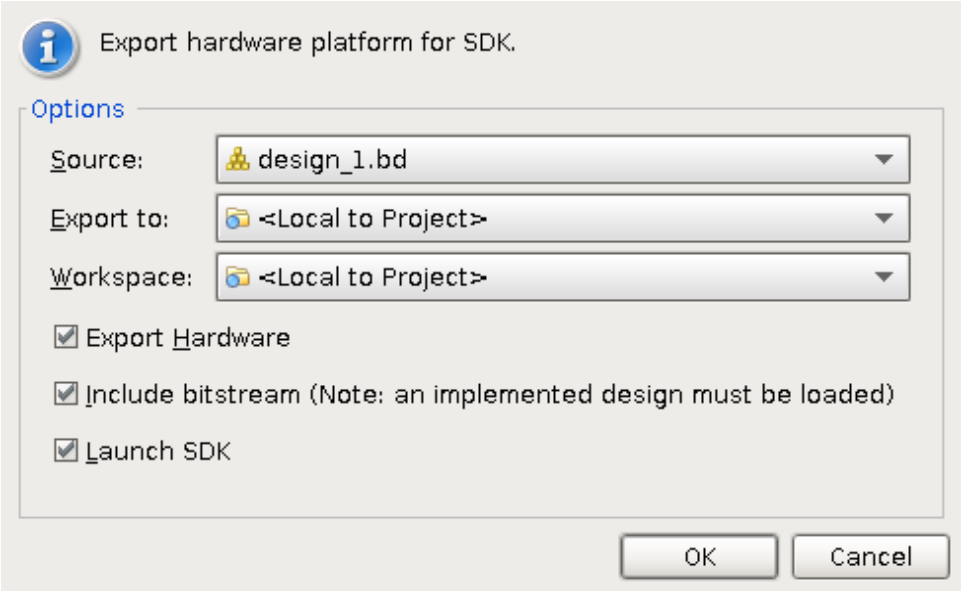
Note that the external memory is mapped with a range of 128 MB at starting address 0x80000000 in this example. Also notice that the external memory is accessible from the **Data** port but not Instruction. In this design we chose to have the AXI data port (M_AXI_DP) connected as the master to the memory. Thus we cannot use the external RAM to store instructions. To use the DDR for instructions, **Enable Peripheral AXI Instruction Interface** should be selected and the M_AXI_IP port should be connected to the memory slave. This example will use the external RAM only for data storage.

Export the Design to SDK

1. Once the bitstream is generated, open the implemented design

2. Select File > Export > Export Hardware for SDK

3. Export the hardware and Launch SDK



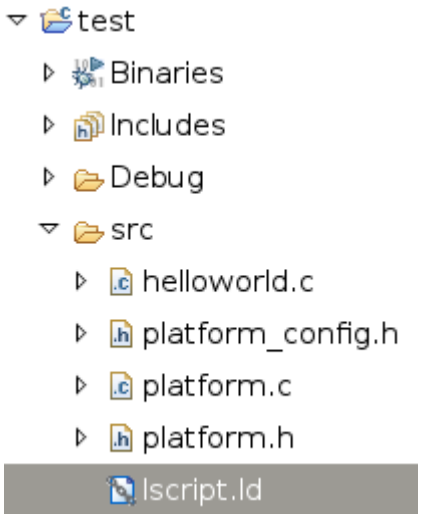
4. In SDK, Create a **New Application Project** and give it a name

5. Select **Hello World** as the Template and Finish the wizard

Creating a Test Program

In this test we will be using the Block RAM (BRAM) on the FPGA to run the program on the MicroBlaze. This is the **microblaze_0_local_memory** in the block design.

1. Open the **ldscript.ld** for your hello world program, in this example it is called “test”



The linker script describes where certain parts of the program should be mapped. At the top you should see two memory regions visible to the MicroBlaze, the local BRAM and the external DDR:

Available Memory Regions

Name	Base Address	Size
microblaze_0_local_memory_ilmb_bram_if_cntl	0x00000050	0x00001FB0
mig_7series_0	0x80000000	0x08000000

Since we want the program to run just on the local use the drop down boxes under **Section to Memory Region Mapping** to assign all Sections to the local memory:

Section to Memory Region Mapping

Section Name	Memory Region
.text	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.init	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.fini	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.ctors	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.dtors	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.rodata	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.sdata2	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.sbss2	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.data	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.got	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.got1	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.got2	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.eh_frame	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.jcr	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.gcc_except_table	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.sdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.sbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.tdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.tbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.bss	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.heap	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze
.stack	microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze

2. Change the helloworld.c program to the following:

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
// Read/Write 16384 words or 64kB
#define TEST_SIZE 16384

// Pointer to the external memory
volatile unsigned int * memptr = (unsigned int*) XPAR_MIG_7SERIES_0_BASEADDR;
// Thomas Wang's 32-bit mix hash
unsigned int hash(unsigned int key)
{
  key += ~(key << 15);
  key ^= (key >> 10);
  key += (key << 3);
  key ^= (key >> 6);
  key += ~(key << 11);
  key ^= (key >> 16);
  return key;
}
int main()
{
    init_platform();

    int i, errors;

    // Write TEST_SIZE words to memory
    print("BEGIN WRITE\n\r");
    for (i = 0; i < TEST_SIZE; i++)
    {
            memptr[i] = hash(i);
    }

    // Read TEST_SIZE words to memory and compare with golden values
    print("BEGIN READ\n\r");
    errors = 0;
    for (i = 0; i < TEST_SIZE; i++)
    {
            if (memptr[i] != hash(i))
                    errors++;
    }

    // Print Results
    if (errors != 0)
            print("ERROR FOUND\n\r");
```

```
    else
        print("ALL GOOD!\n\r");

    return 0;
}
```

This program writes 64kB of values to the beginning of external memory and reads them back, note this is eight times more memory than was allocated to the MicroBlaze BRAM. Is this test comprehensive? How could you modify the program to test all the available memory?

Run the example

1. Use Xilinx Tools > Program FPGA to program the hardware design onto the FPGA

2. Create a **Run Configuration** with your test program as in previous labs

You should see:

```
BEGIN WRITE
BEGIN READ
ALL GOOD!
```

Appear on your Terminal or Console tab