//

// 01/03/2018. Jgarrigos.

//13/02/2023. Updated for vivado2019.1.

// This example demonstrates an heterogeneous architecture (standalone) with a zynq arm (master) and one (or several) microblazes (slaves).

Synchronization methods:

- shared axi bram (dual port, one for each processor): OK
- Shared axi bram (with shared axi interconnect): OK (Better)
- Shared axi bram (with cascaded axi interconnect): OK (Best, for complex designs)
-
- Axi gpio??
- Axistreams???
- DMA??
-

//

==========

Part One: Define block diagram in Vivado

==========

Create a new Vivado project.

- Select Do not specify sources at this time
- Select VHDL instead of Verilog as the preferred design language
- Select Zedboard as the target board

Add a new block diagram

Add a zynq block to the diagram

Run designer assistance /connection automation to setup default configuration.

Add 2 axi_gpios and configure them for the switches and leds respectively.

Run designer assistance /connection automation to setup default configuration.

Add Microblaze:

Run block automation: config with 32k (or more) for the onchip RAM and with no cache.

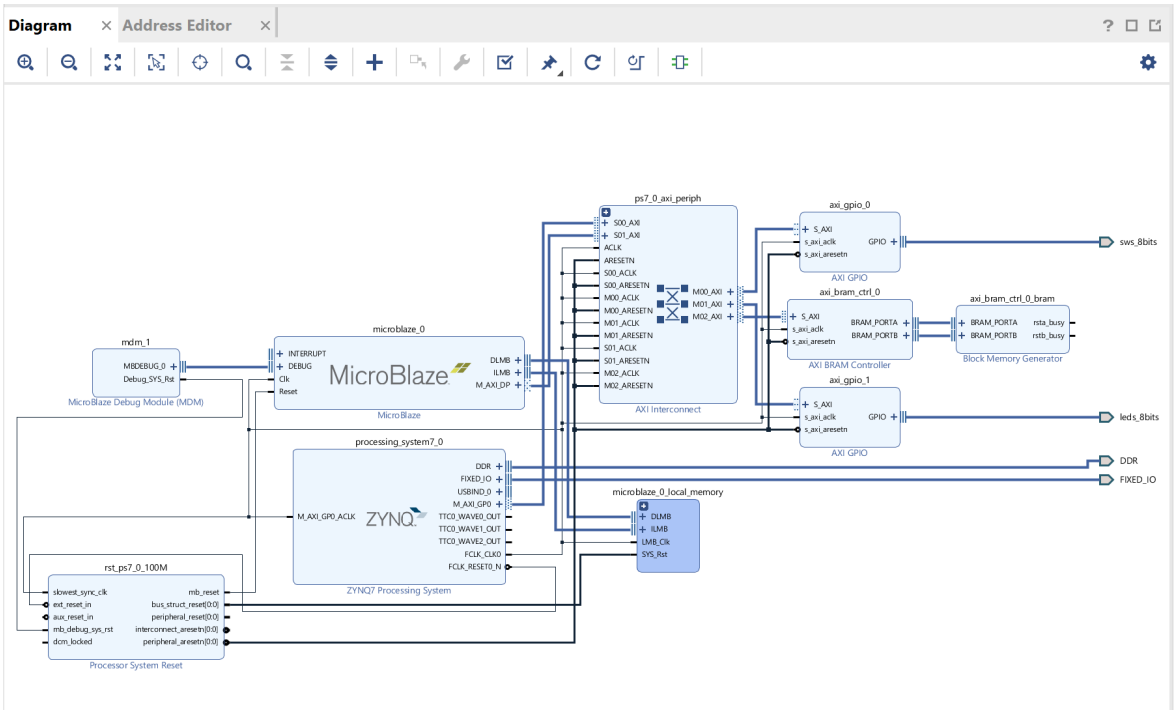
Run connection automation: default configuration.

Add an AXI BRAM Controller block. Config with dualport and no ECC. 8KBytesx32bits (this should be default settings).

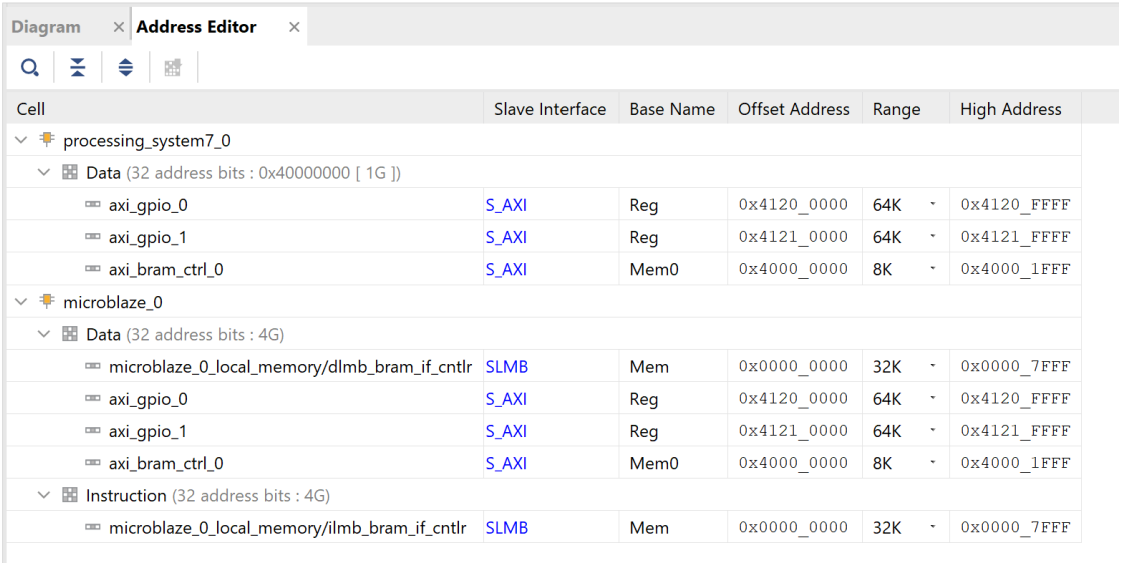
Run designer assistance /connection automation to setup default configuration.

If there is an error and it can not automatically map the axi_bram into the microblaze address map, just select the axi_bram peripheral in the Address Editor, click secondary mouse button and select assign address.

Validate design. Check that there are no errors in the block diagram. The BD should be as this one:



This symmetrical architecture (both micros seeing the same address space and connected to the same peripherals,) has the advantage that is automatically connected by Vivado. More complex architectures would need manual connections as required.



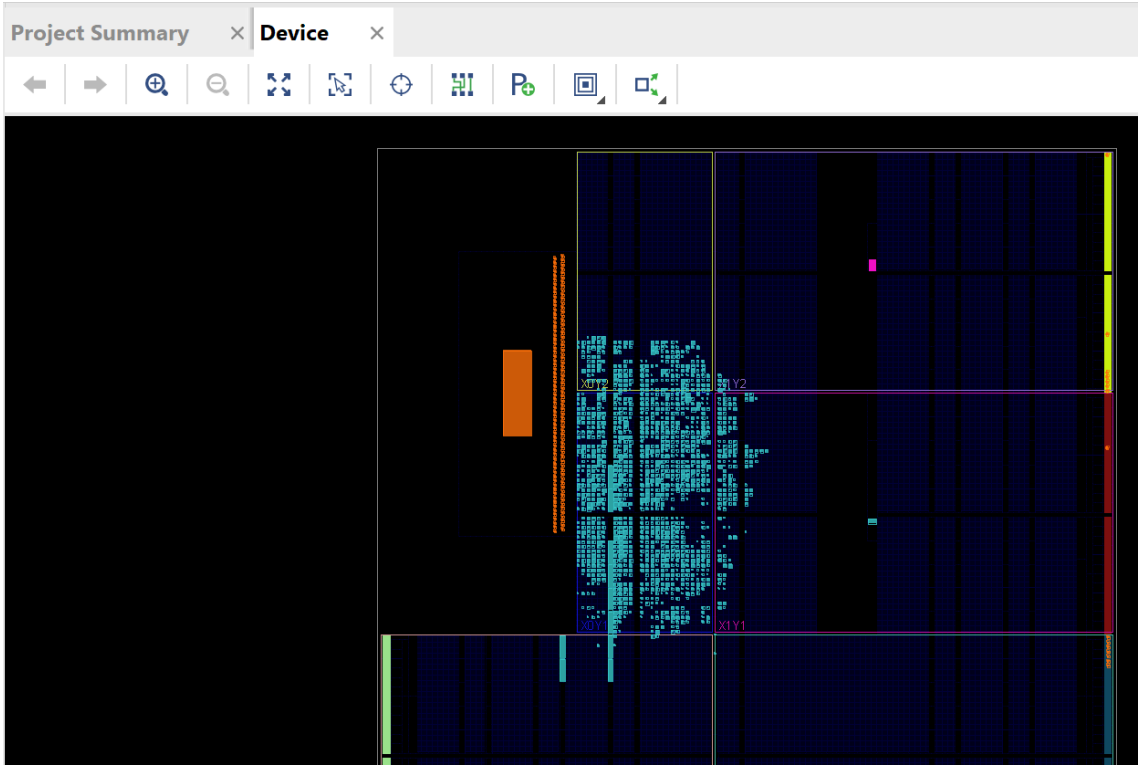
Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF

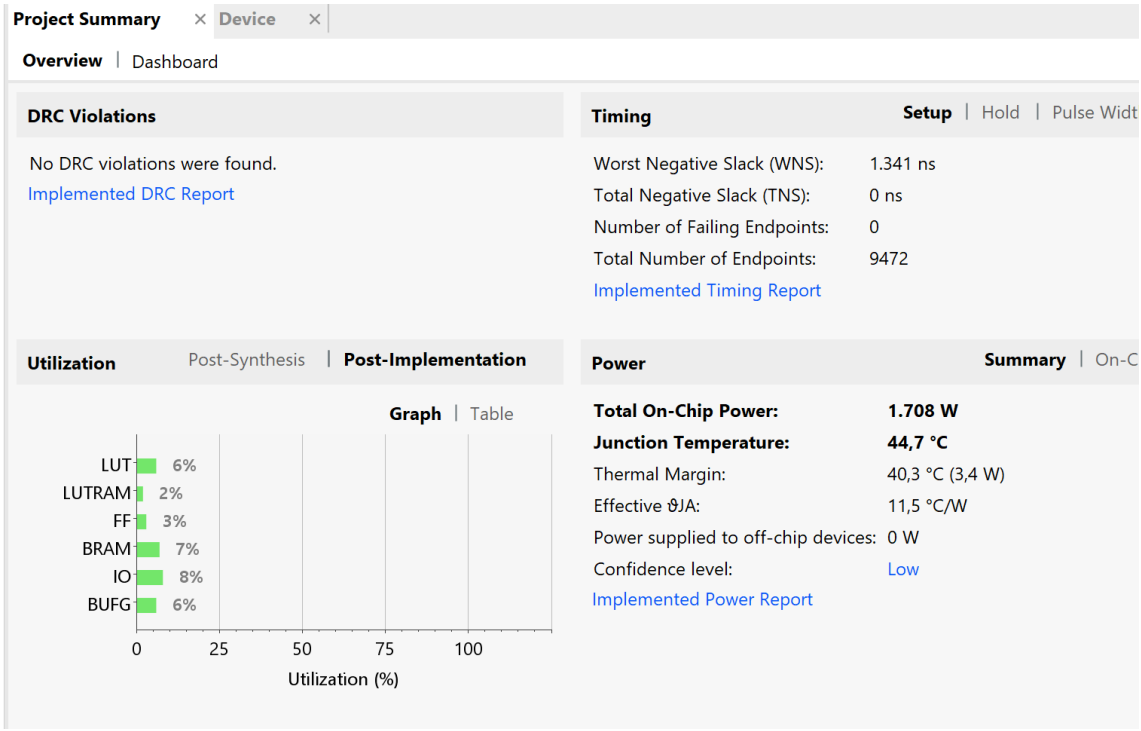
Select .bd file and click on Generate output products in the floating menu.

Select .bd file and click on Generate HDL wrapper in the floating menu.

Select Generate bitstream. Synthesis and implementation (translate, map, place&route) will be run if they don't exist or are not up-to-date.

Export hardware design to SDK (Menu > File > Export > Export hardware). Do not forget to select "Include Bitstream" option.





Launch SDK.

==========

Part Two: Design the application software to execute on the microprocessor in SDK

==========

Create app for the microblaze:

- Select Menu > File > New application project
- Enter the desired name for the app
- Check that the hw platform selected is the correct one if there is more than one
- Check that the OS is configured as standalone (no OS in fact)
- Check that a new BSP will be created for the hw platform
- Select a "peripheraltest" template for your app

Create app for the zynq arm core0:

- Select Menu > File > New application project
- Enter the desired name for the app
- Check that the hw platform selected is the correct one if there is more than one
- Check that the OS is configured as standalone (no OS in fact)
- Check that a new BSP will be created for the hw platform
- Select a "helloworld" template for your app

Modify templates with the code provided in arm_axi_sws_2mblaze.c and mblaze_axi_leds_2arm.c respectively.

Check lscript.ld to verify that allocation of every memory segment is set to correct memory (ddr for the arm, microblaze_0_local_memory_ilmb_bram_if_cntlr_microblaze_0_local_memory_dlmb_bram_if_cntlr for the microblaze). Check also that stack and heap sizes are big enough for your data (especially if you have long arrays, stack is just 0x400 by default).

Save your app file. This will also run compile. Check that no errors are shown in the log console window and that an executable (.elf) file is produced correctly.

Go to Menu > Xilinx Tools and select Program FPGA.

- Select microblaze.elf to be inserted in the bitstream file, instead a bootloop.
- Of course, you should have connected and turned on the board.
- Do not forget to connect both cables: usb-jtag to program the fpga and usb-uart for the console with the microprocessor
- Check that a blue led on the board lights on, this means programming has been successful.

Open the Teraterm utility and config a connection to the serial COM6 port with 115200bps. Let the rest of parameters with their default values.

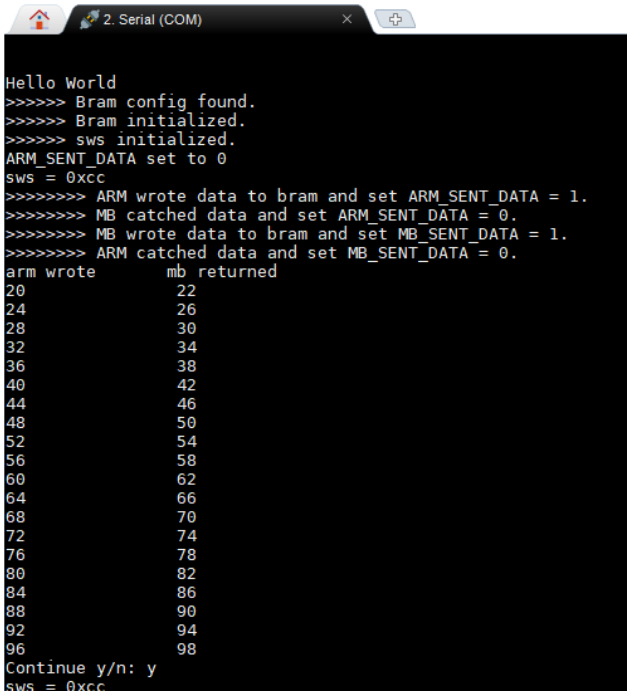
- The COM6 port will not be present if the ZedBoard is not powered.

Select your app project and in the floating menu select Run As> launch on hardware (GDB)

Check the teraterm console and see your app executing on the board.

CONGRATULATIONS!!

//////////////////////////////// Screen capture //



//////////////////////////////// Interlock communication protocol ////////////////////////////

Arm0		**MB**
Config BRAM		Config BRAM
Config sws		Config leds
Read sws		Read sws_value from BRAM
Print sws_value to Console		Write sws_value to leds
Write sws_value to BRAM		
Write dummy data to BRAM		
Set ARM_SENT_DATA	→	Wait for ARM sending data
		Read data from BRAM
Wait for MB catching data	←	Reset ARM_SENT_DATA
		Write data to BRAM
Wait for MB sending data	←	Set MB_SENT_DATA
Read data from BRAM		
Reset ARM_SENT_DATA	→	Wait for ARM catching data
Print BRAM data to console		

////////////////////////////// BRAM address space //

0x0000	ARM_SENT_DATA (0 or 1)	00	00	00
0x0004	--	--	--	--
0x0008	MB_SENT_DATA (0 or 1)	00	00	00
--	--	--	--	--
--	--	--	--	--
0x0018	sws_data	00	00	00
--	--	--	--	--
0x0020	Data0 (20 or 22)			
0x0024	Data1 (24 or 26)			
--	Data2 (28 or 30)			
--	--			
--	--			
0x0096	Data20 (96 or 98)			
--	--			
--	--			
0x1FFC	--			