How To Store Your SDK Project in SPI Flash

Overview

This guide will show you how to easily store your Microblaze/SDK project in the SPI Flash on your 7 series device. Storing a program in Flash allows you to preserve your project through power cycles.

Prerequisites

This guide requires the existence of a Vivado project containing a Microblaze system built complete with **Quad SPI**, **External Memory**, and **Uart** cores, and that you have the appropriate QSPI mode jumper setting on the board. You can follow the steps presented in the Microblaze tutorial (https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze/start#prerequisites), and see below the missing steps for adding QSPI Flash and perform all the connections. Since there are slight differences in the Vivado 2019.1 and the version in which the Microblaze tutorial was created, the user needs to check all the connections in the block design to make sure that are correctly made or present.

Hardware

- **Digilent Nexys 4 DDR FPGA Board and Micro USB Cable for UART communication and JTAG programming**

Software

- **Xilinx Vivado 2019.1 with the SDK package.**

Board Support Files

- **Board Support Files**
  - *These files will describe GPIO () interfaces on your board and make it easier to select your FPGA board and add GPIO () IP blocks*
  - *Follow this Wiki guide (Vivado Board Files for Digilent 7-Series FPGA Boards (https://digilent.com/reference/vivado/boardfiles)) on how to install Board Support Files for Vivado*

> The xilisf library was officially depreciated by Xilinx starting in 2019.2 with the release of Vitis. This guide is only directly compatible with 2019.1 and older versions of the Xilinx software.

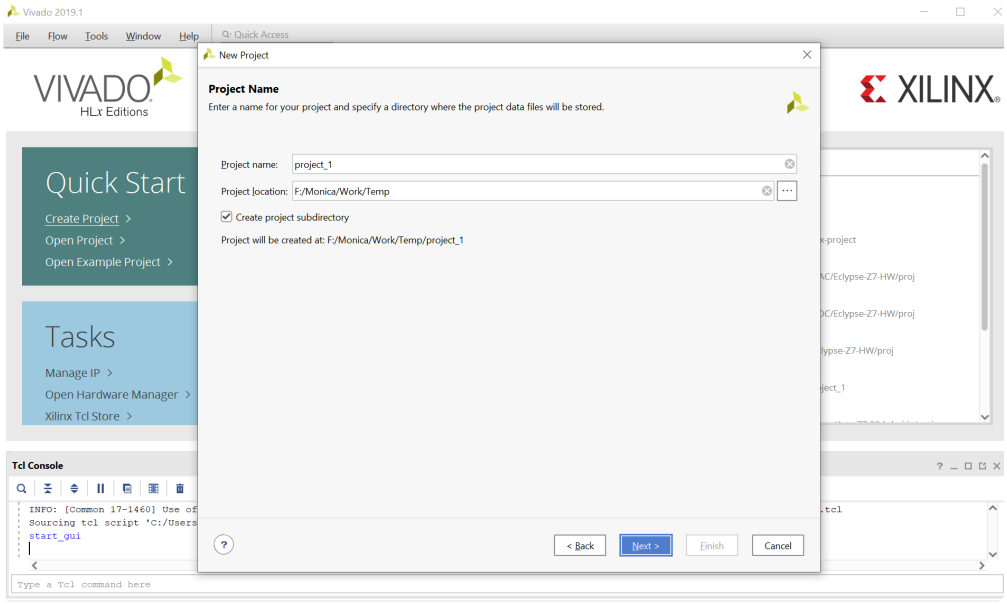
Tutorial

Vivado Steps for creating the block design and generate bitstream

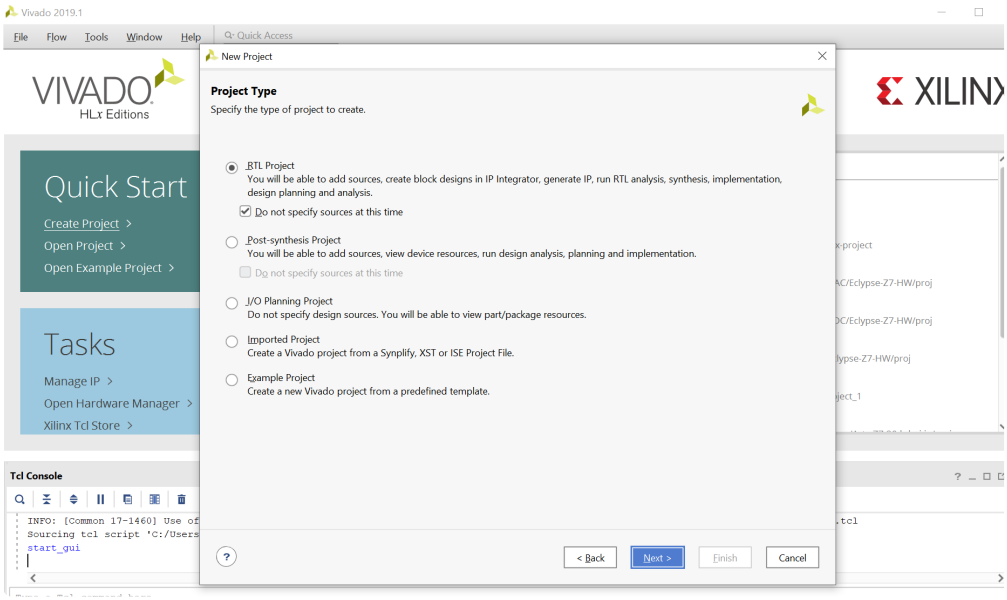
1. Create new project

When you first run Vivado this will be the main start window where you can create a new project or open a recent one.

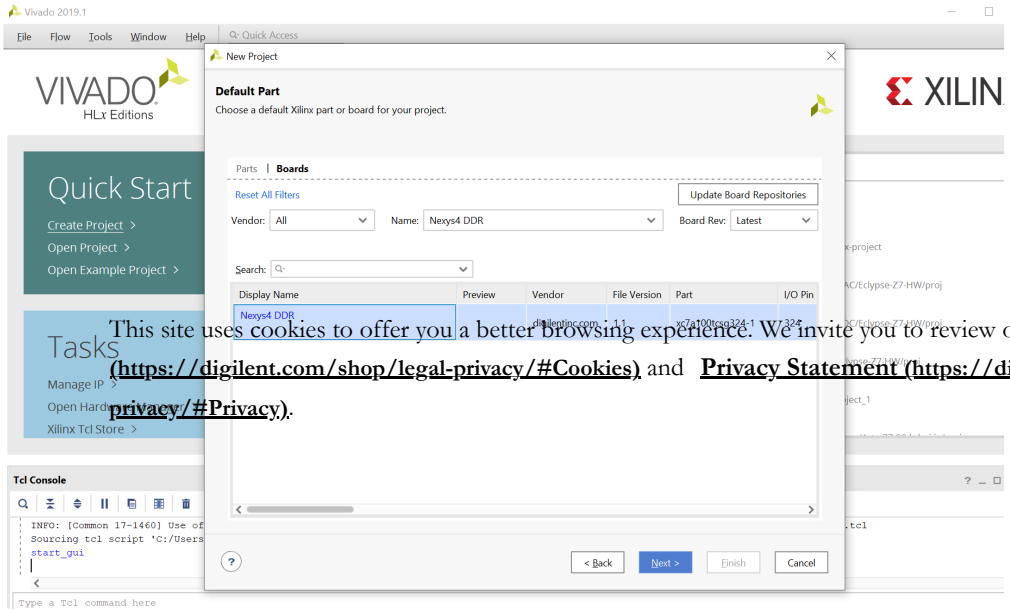
- Click on **Create New Project**. Choose the Project Name and Location such that there are **no blank spaces**. This is an important naming convention to follow for project names, file names and location paths. Underscore is a good substitute for empty spaces. It is good practice to have a dedicated folder for Vivado Projects, preferably with the smallest possible path length. Example: C:/Vivado_Projects. Name your Project and select the Project location and click **Next**.



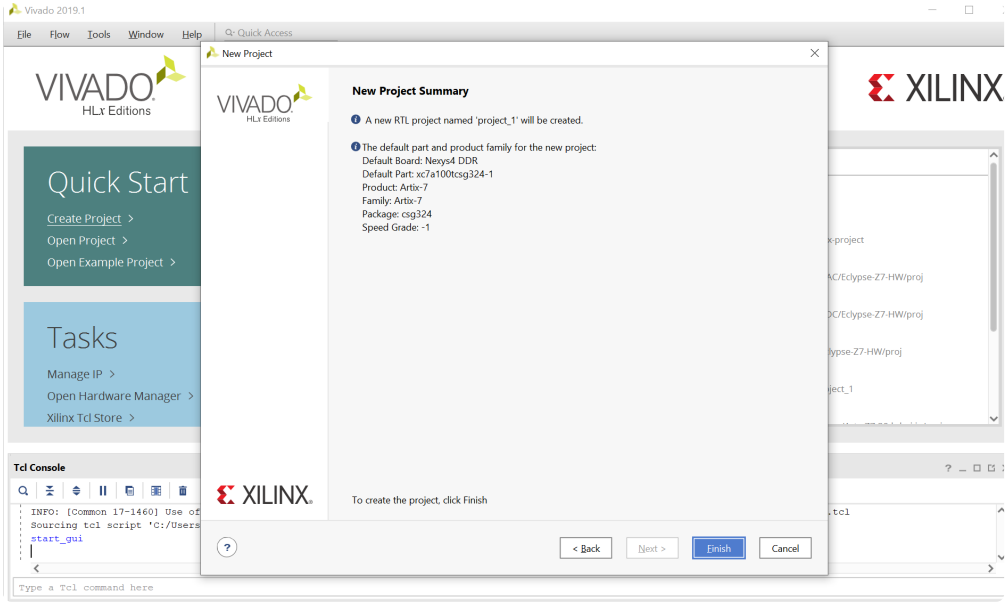
- Choose Project Type as **RTL Project**. Leave the **Do not specify sources** box unchecked and click **Next**.



- If you have followed the Board Support File Wiki guide then click next and select **Boards**. From the filter options make required selections for Vendor, Display Name and Board Revision. **Nexys 4 DDR** should be displayed in the selection list. A mismatch in selecting the correct board name will cause errors.

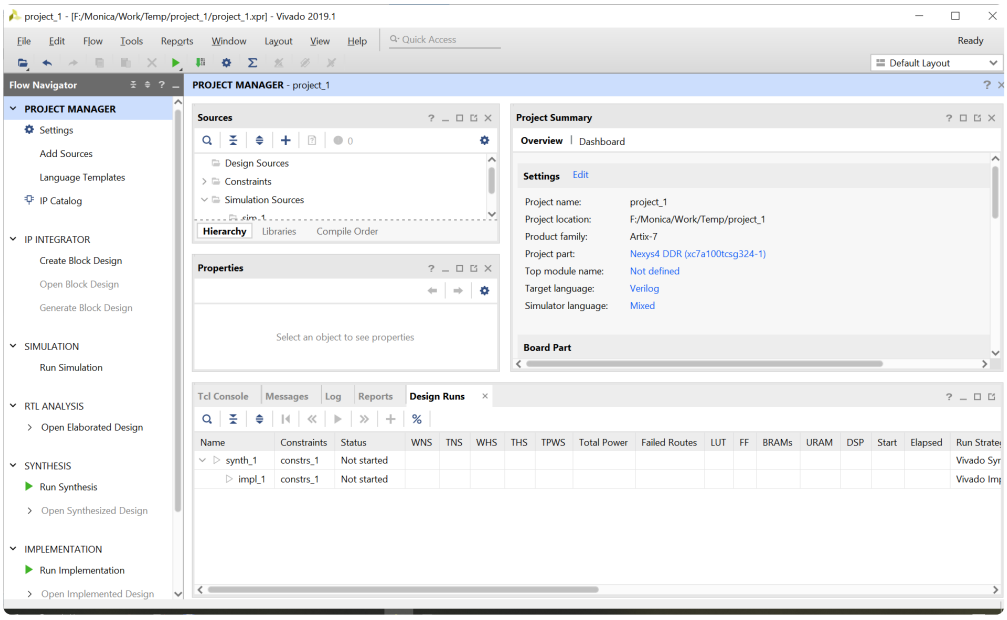


- A summary of the new project design sources and target device is displayed. Click **Finish**.

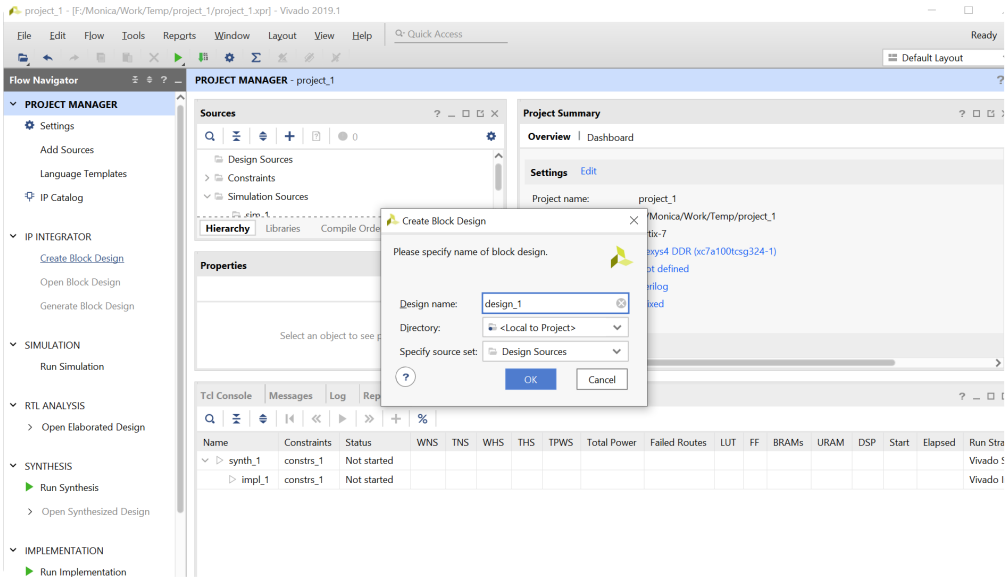


2. Creating New Block Design

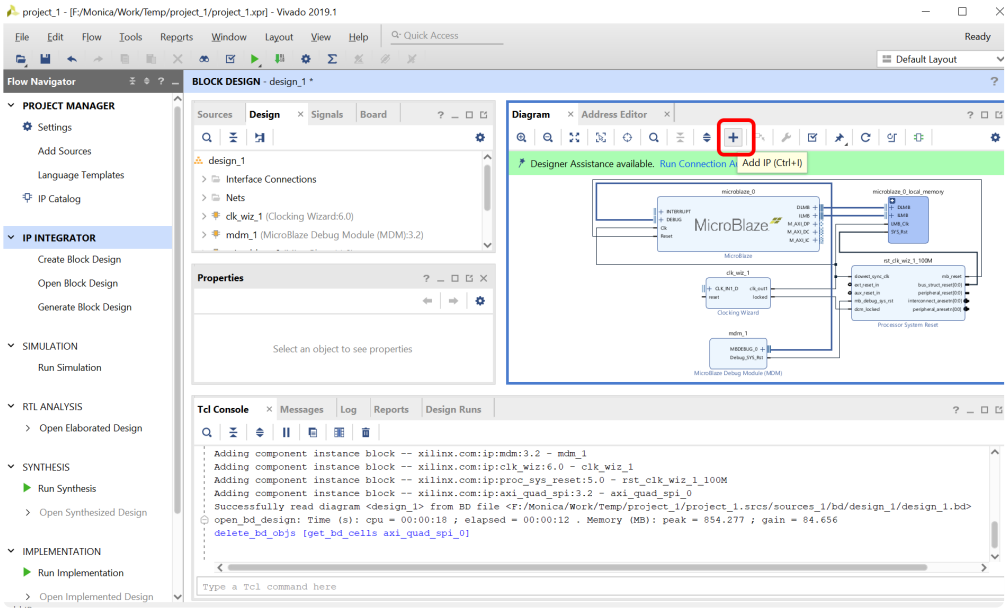
This is the main project window where you can create a IP based block design or add RTL based design sources. The flow navigator panel on the left provides multiple options on how to create a hardware design, perform simulation, run synthesis and implementation and generate a bit file. You can also program the board directly from Vivado with the generated bit file for an RTL project using the Hardware Manager. For our design, we will use the IP Integrator to create a new block design.



- On the left you should see the Flow Navigator. Select **Create Block Design** under the IP Integrator. Give a name to your design without any empty spaces.

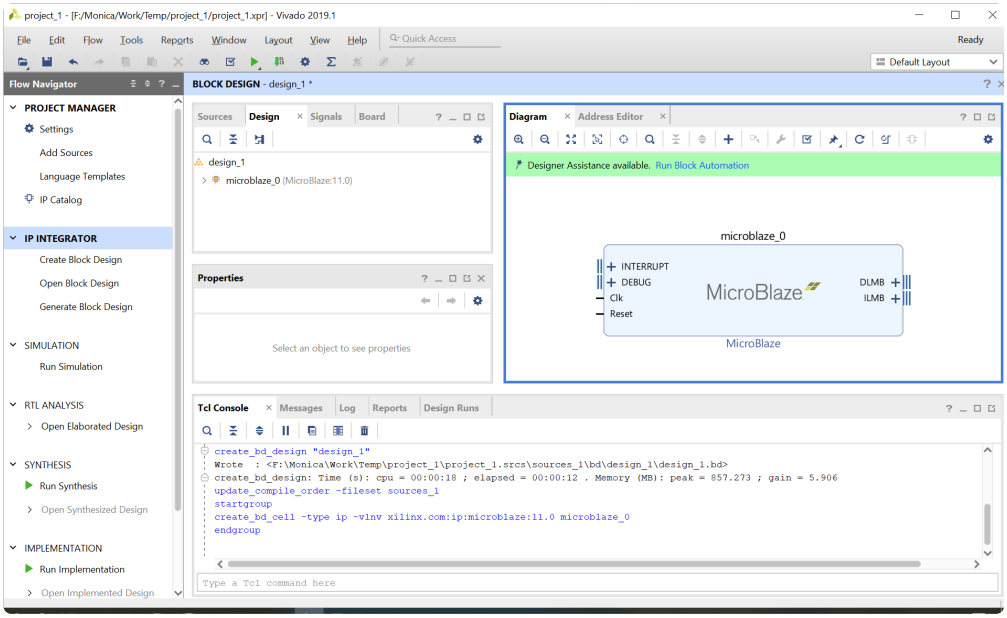


- An empty design workspace is created where you can add IP blocks. Add an IP core by clicking on the **Add IP** icon. This should open a catalog of pre-built IP blocks from Xilinx IP repository. Search for "Microblaze" and double click on it to add the IP block to your empty design.

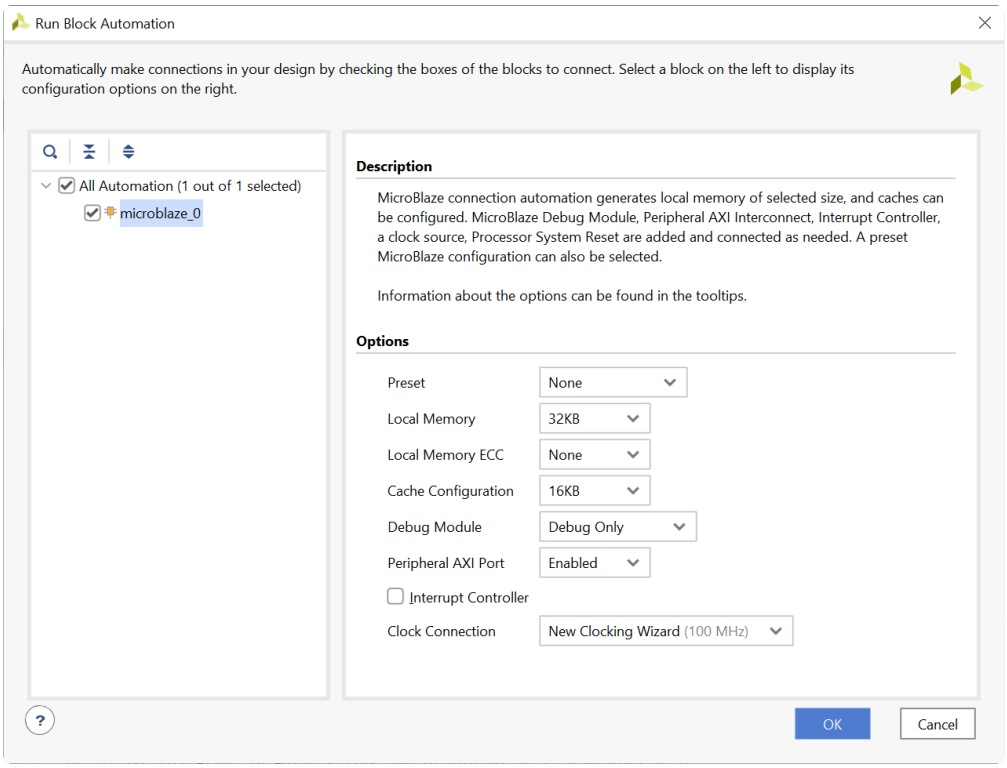


3. Adding Microblaze IP and Customization

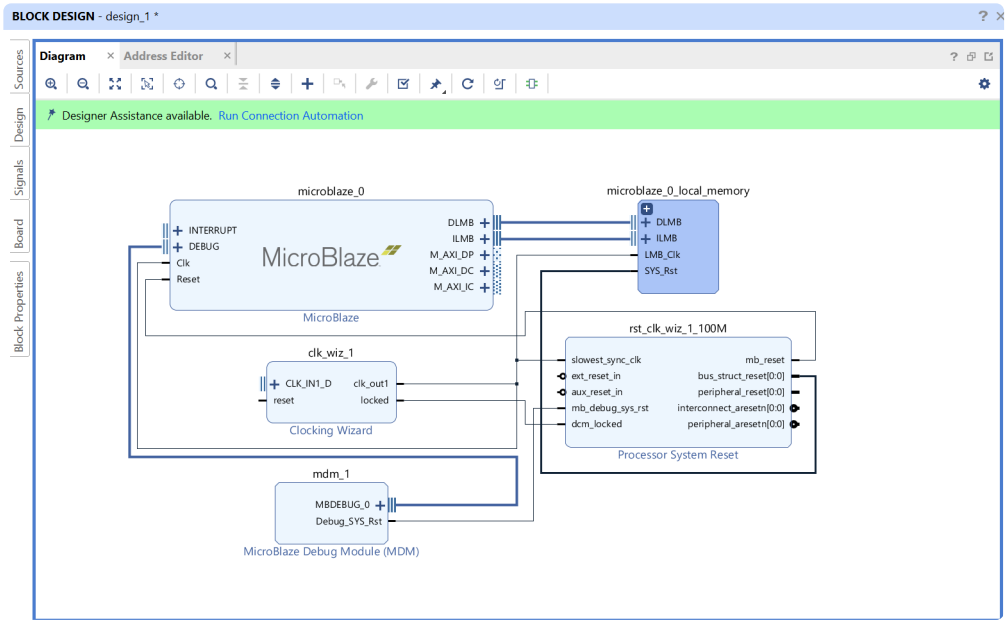
- This is the Xilinx Microblaze IP block. When a new IP block is added the user can customize the block properties by either clicking on the **Run Block Automation** message prompt or by double clicking on the block itself.



- Select **Run Block Automation** and a customization assistant window will open with default settings. Change default settings in the block options as shown below and click **OK**. This will customize the block with our new user settings.

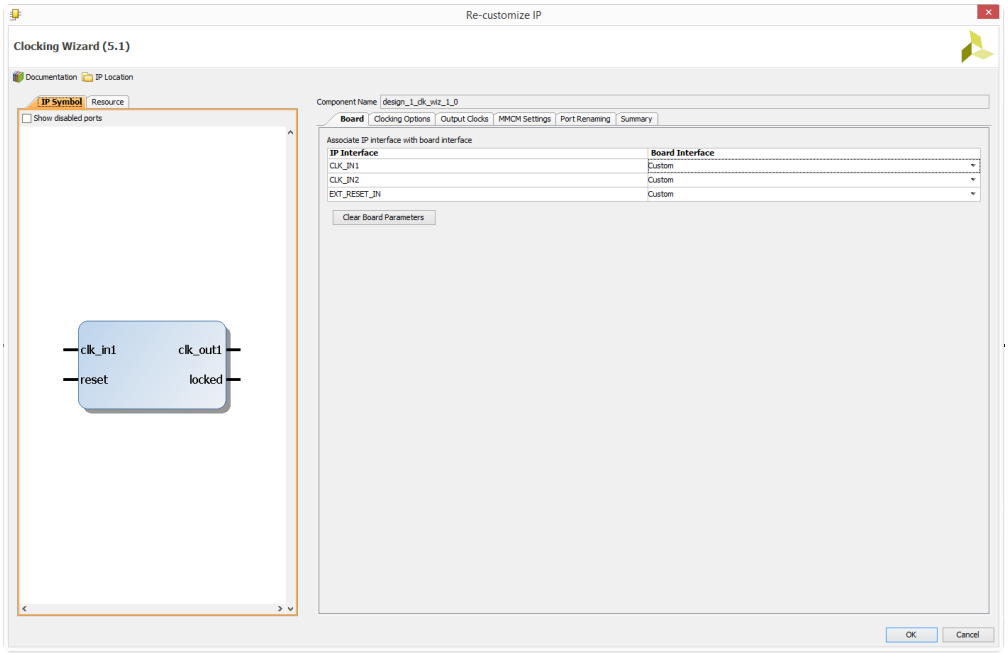


- Running the block automation will auto-generate a set of additional IP blocks which will be added to our hardware design automatically based on the options selected in the previous step. **Do not click on Run Connection Automation yet.**



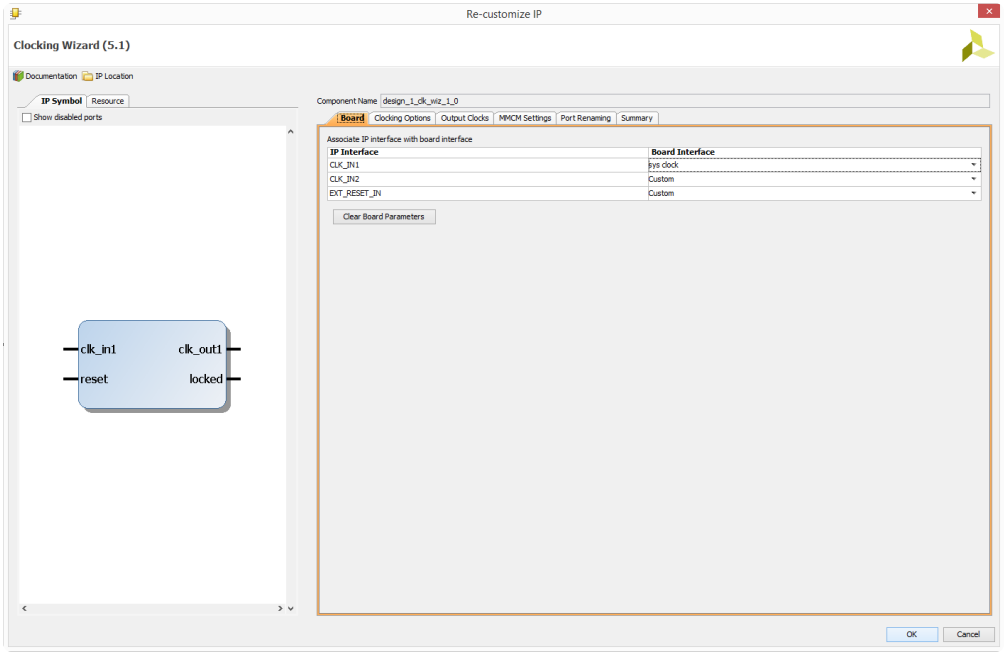
4. Customize Clock Wizard IP Block

- Double click on the **Clock Wizard** (clk_wiz_1) IP block.



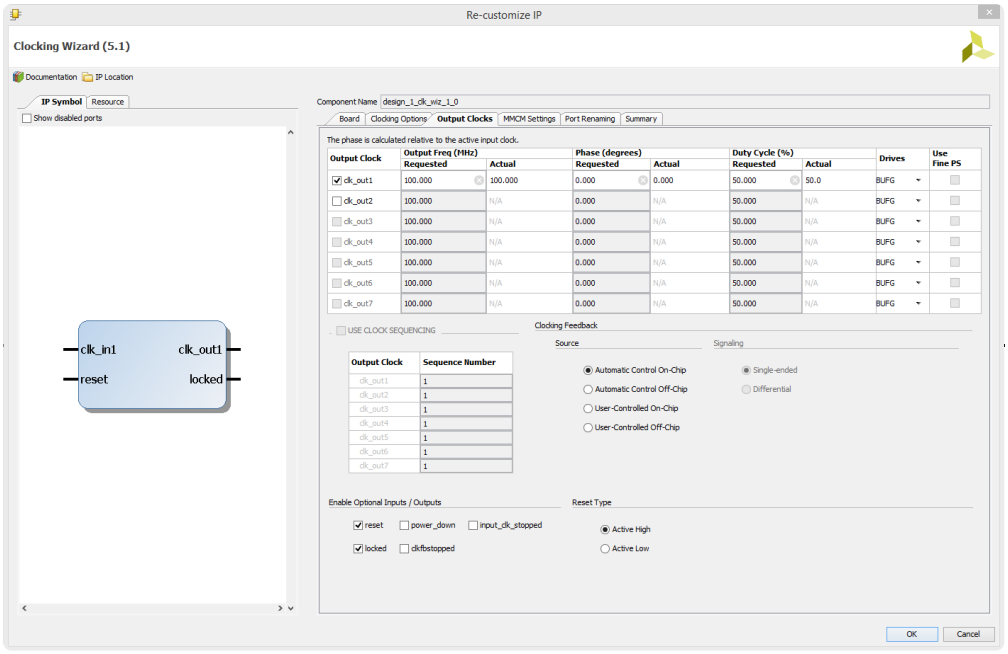
(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-clkwizard1.png)

- Choose **sys clock** for CLK_IN1.



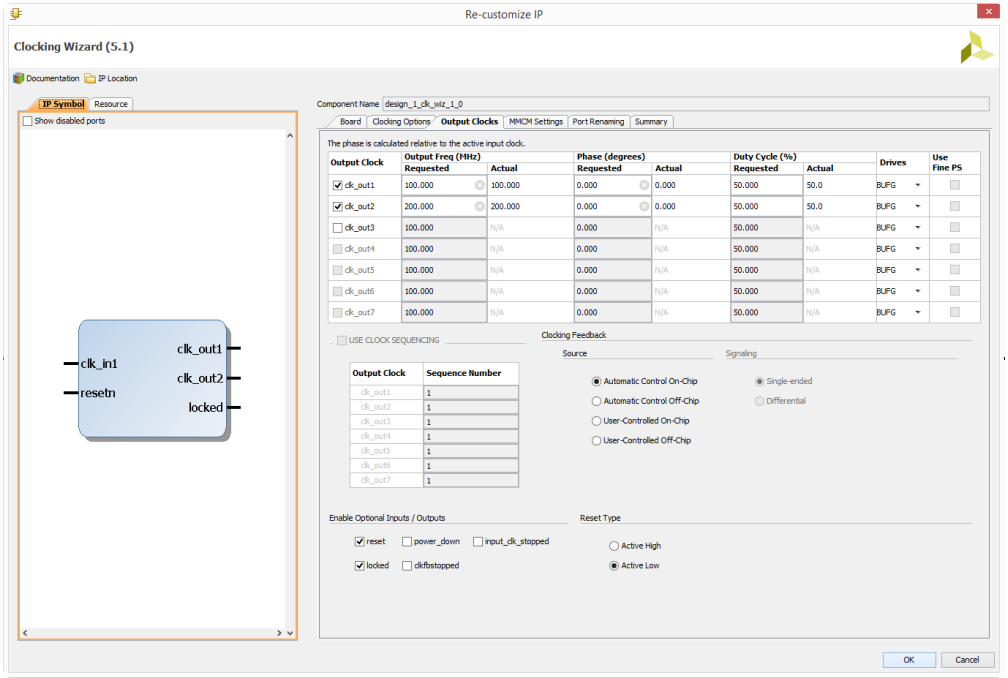
(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-clkwizard2.png)

- Select the **Output Clocks** tab.



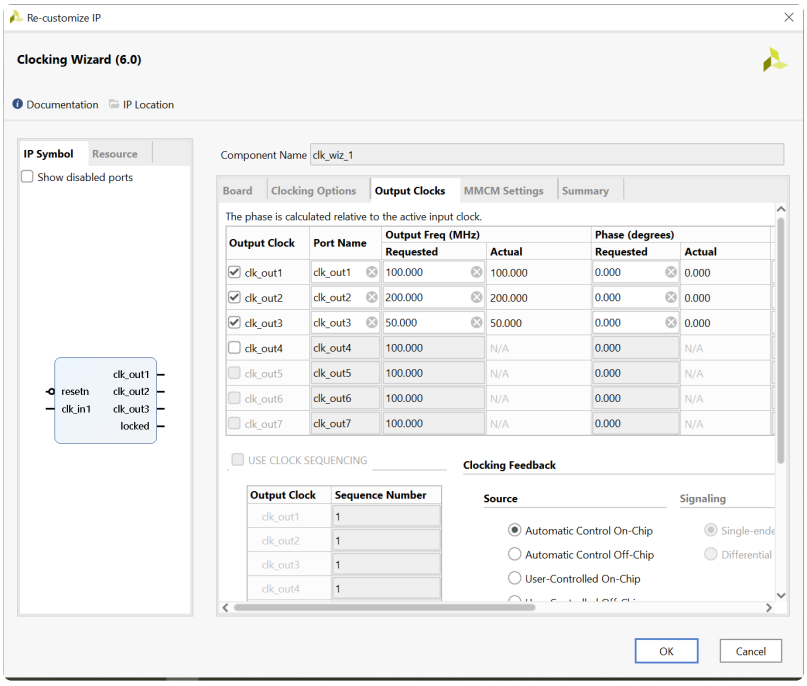
(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-clkwizard3.png)

- Select **clk_out2** output frequency as "200.000" (Mhz) and set **Reset Type** as *Active Low*. The left panel shows a GUI () representation of the block and its internal settings. Observe that the reset pin will now read as *resetn*. This graphically represents the internal setting for active low.



(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-clkwizard4.png)

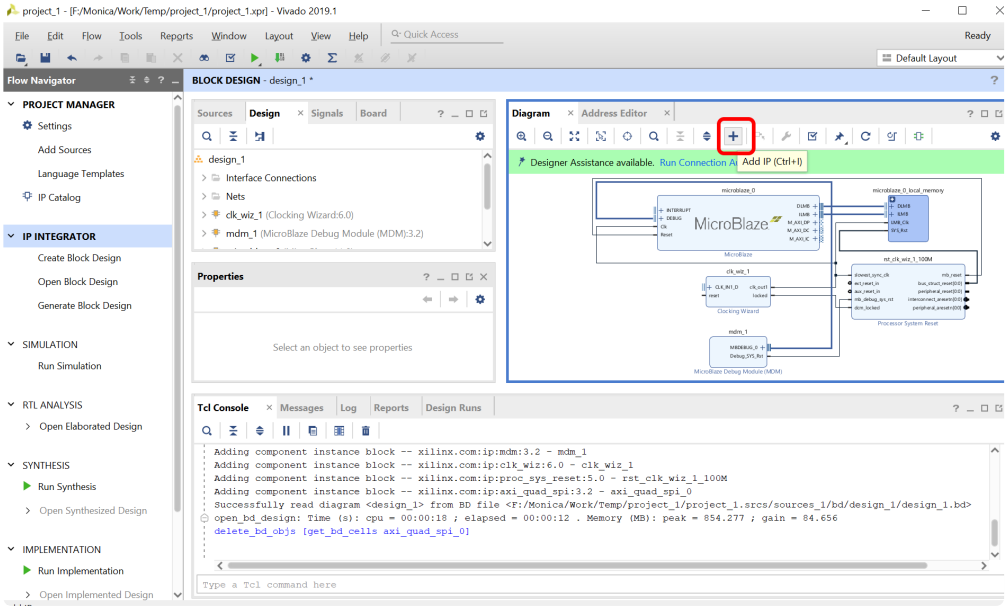
- Select the third clock, having 50MHz frequency:



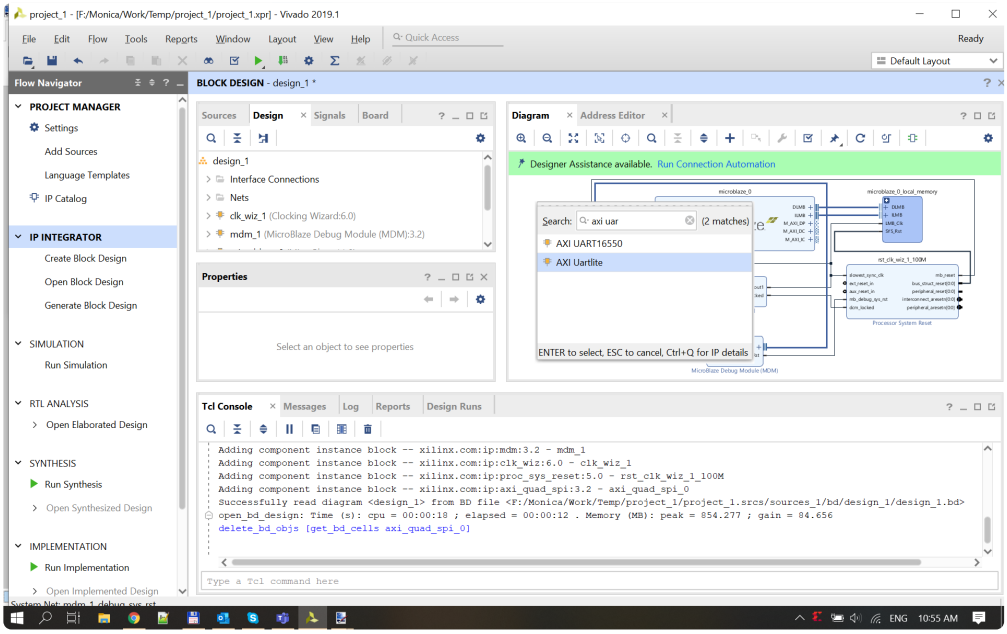
- The **Port Renaming** tab will give you a summary of the inputs and outputs to the Clock Wizard IP block. Click **OK** to finish block automation of Clock Wizard. **Do not select Run Connection Automation yet**.

5. Adding UART IP Block

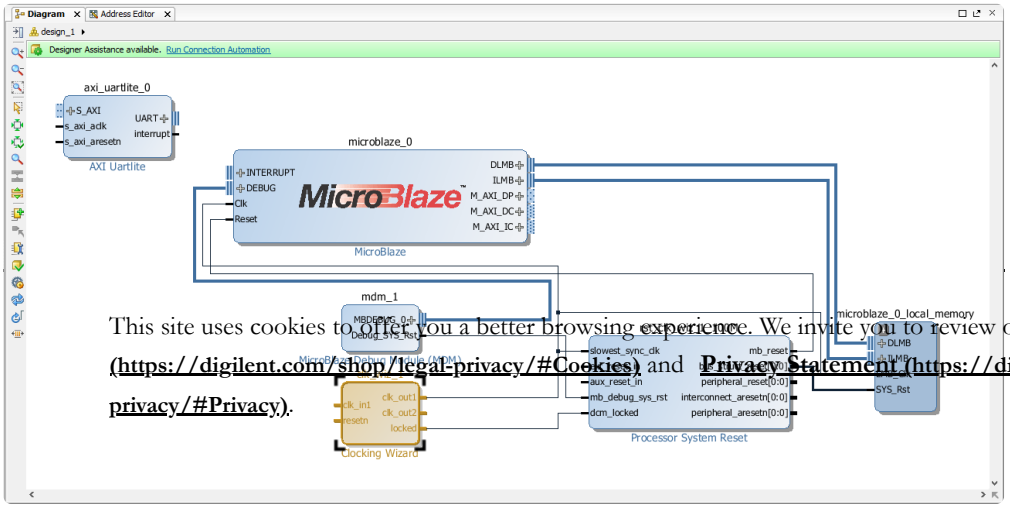
- Go to **Add IP** and search for "UART".



- Select the **AXI Uartlite** IP block.



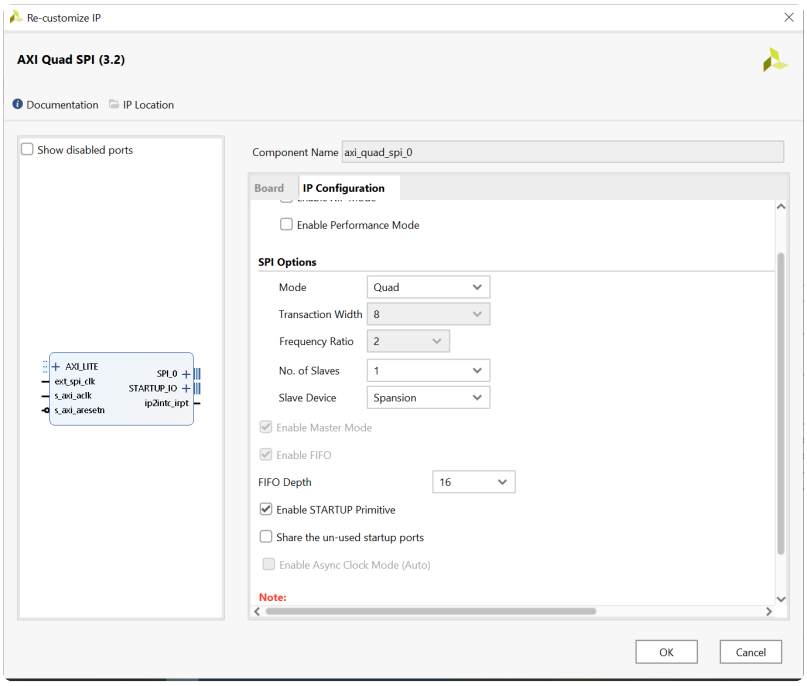
- This will add a UART block to the existing design. We need a UART controller to communicate between the terminal window on the Host-PC and the Nexys 4 DDR hardware.



(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-adduart.png)

6. Add AXI Quad SPI IP

From the Add IP option, select AXI Quad SPI IP with the following customization: Mode - Quad, slave device - Spansion (or Macronix for boards with Macronix flash)

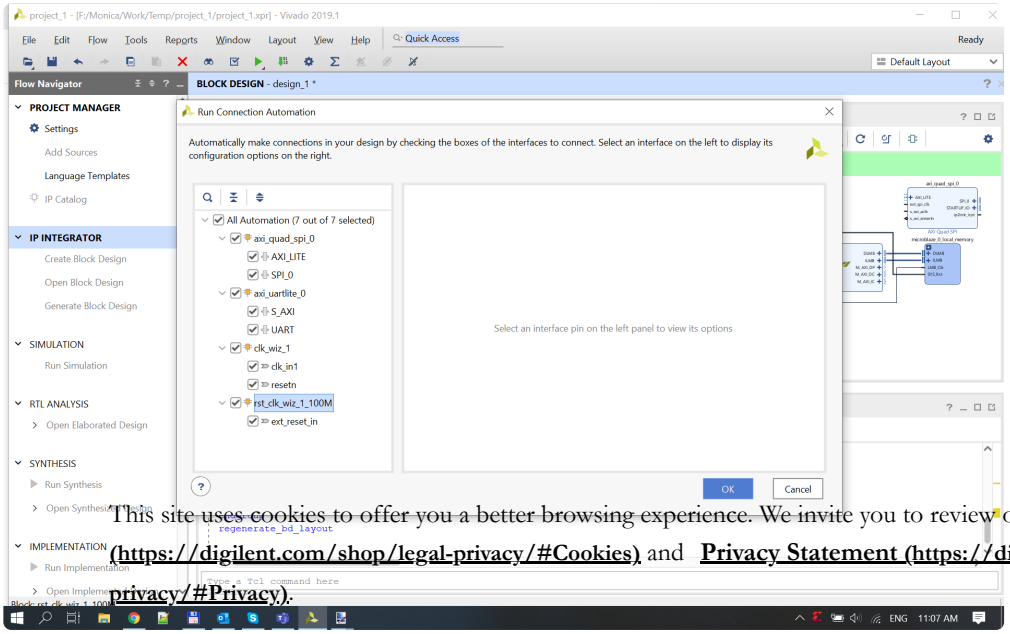


7. Running Connection Automation for the First Time

- Select the **Run Connection Automation** from the *Designer Assistance* bar message prompt. This will open up the Run Connection Automation window. Select all available connections and click **OK**. Completing this step will connect all the IP blocks that have been added and customized up to this point. In addition to performing auto-connection of available IP blocks, a new IP block called **microblaze_0_axi_periph** will be added to our design. Two signal pins **reset** and **sys_clock** will be added as well. The pin signals point to the right indicating that they are inputs to the clock wizard block (clk_wiz_1) and reset clock wizard block (rst_clk_wiz_1_100M).

The **clk_out2** pin will be manually connected later.

Do not select Run Connection Automation at this point.

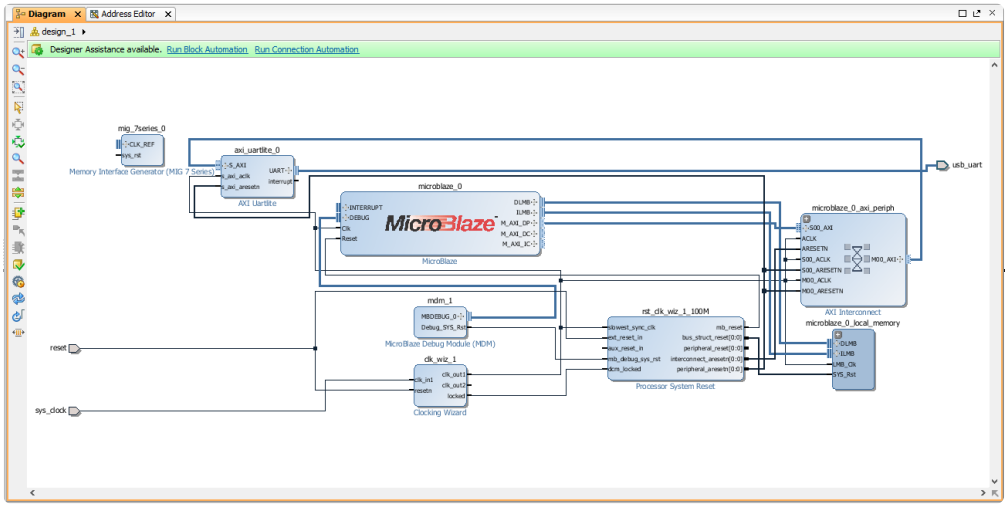


8. Perform some manual connections/check the connections

- Connect the Clocking Wizard “clk_out3” (50Mhz clock) to the “ext_spi_clk” pin.
- Check if the “clk_out1” (100Mhz clock) is connected to the “s_axi_aclk” pin
- Check if the Processor System Reset output port “peripheral_aresetn” is connected to the “s_axi_aresetn” inputs of UARTLite and QSPI blocks

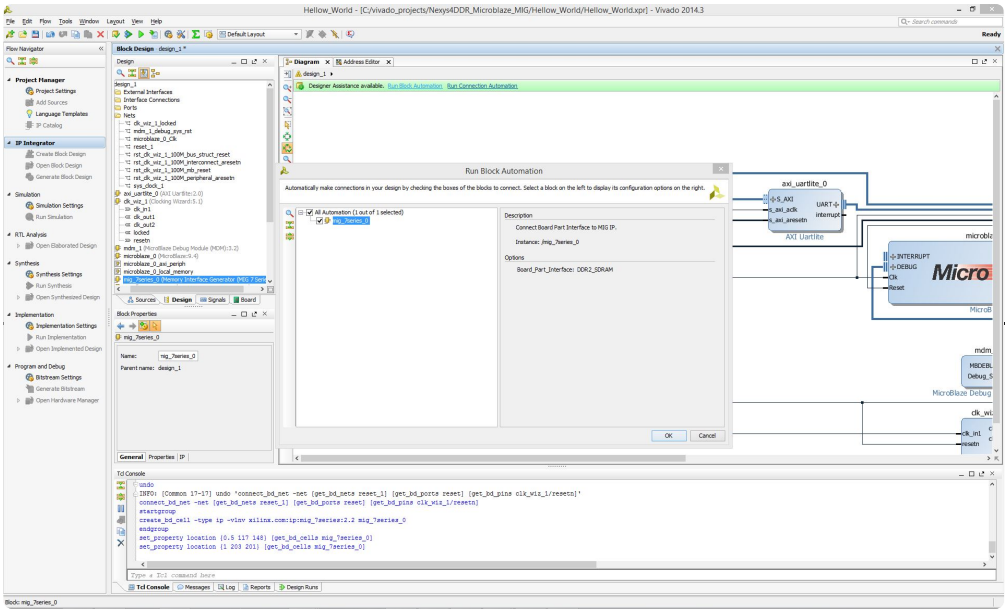
9. Adding and Customizing Memory Interface Generator IP Block

- Memory Interface Generator will be the final IP block we will add in our design.



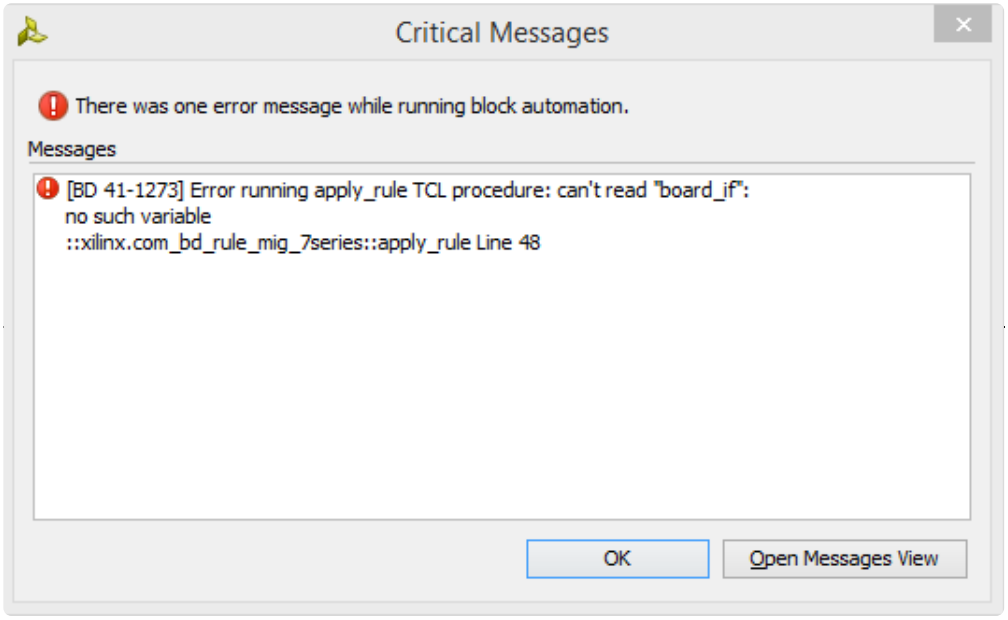
(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-addmig.png)

- After adding the MIG IP block, click on **Run Block Automation**.
- Board part interface will be displayed as DDR2_SDRAM. Click **OK** to run the block automation.



(https://digilent.com/reference/_detail/vivado/mig_26.jpg?id=learn%3Aprogrammable-logic%3Atutorials%3Ahtsspisf%3Astart)

- When the MIG block automation is run, you will see this specific error message [BD 41-1273]. You can ignore this for now. It will not affect your design in any way. The MIG block will be configured as per the board support files that have been downloaded for Nexys 4 DDR. Click OK to dismiss this message. You will find the MIG IP block now has additional input and output pins which have to be connected to valid signals.

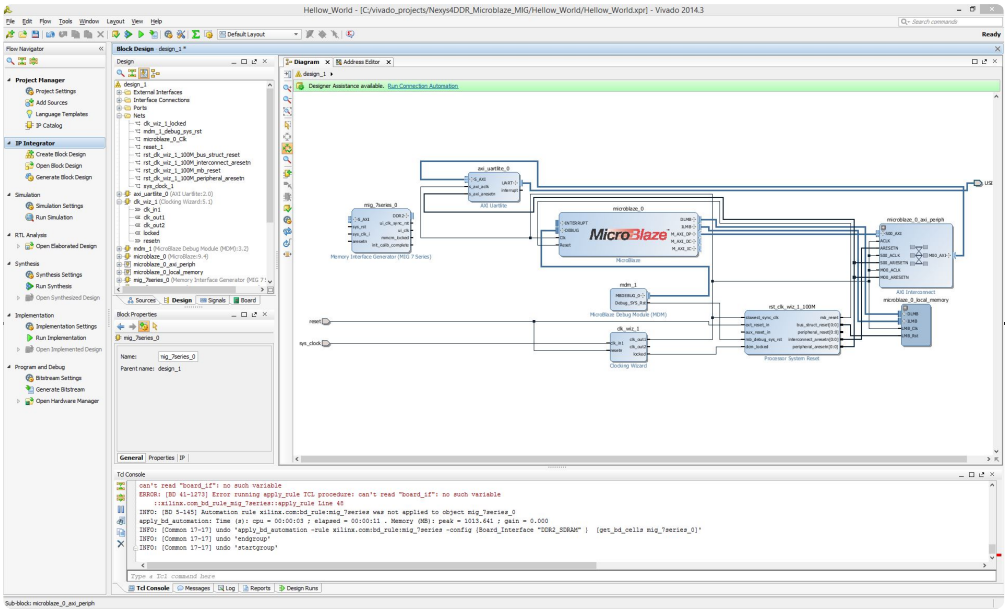


(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-error41-1273.png)

Connect the clocking wizard "clk_out2" (200 MHz ()) clock to "sys_clk_i" of the Ram controller. Run connection automation. Right click the DDR controller "DDR2" pin and make it external.

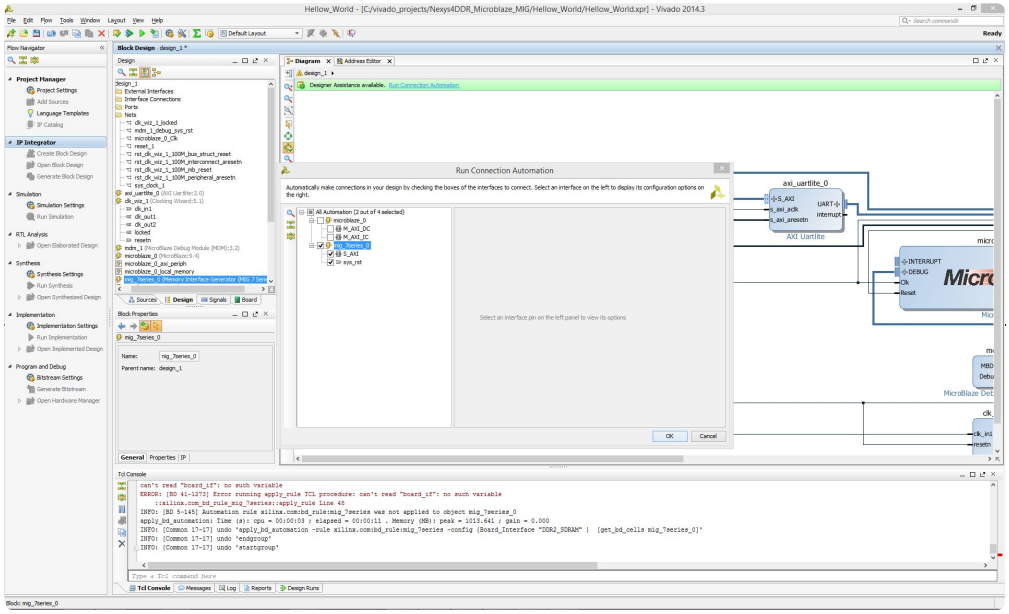
10. Running Connection Automation for the Second Time

- **Now click on Run Connection Automation** message prompt on the *Designer Assistance* bar.



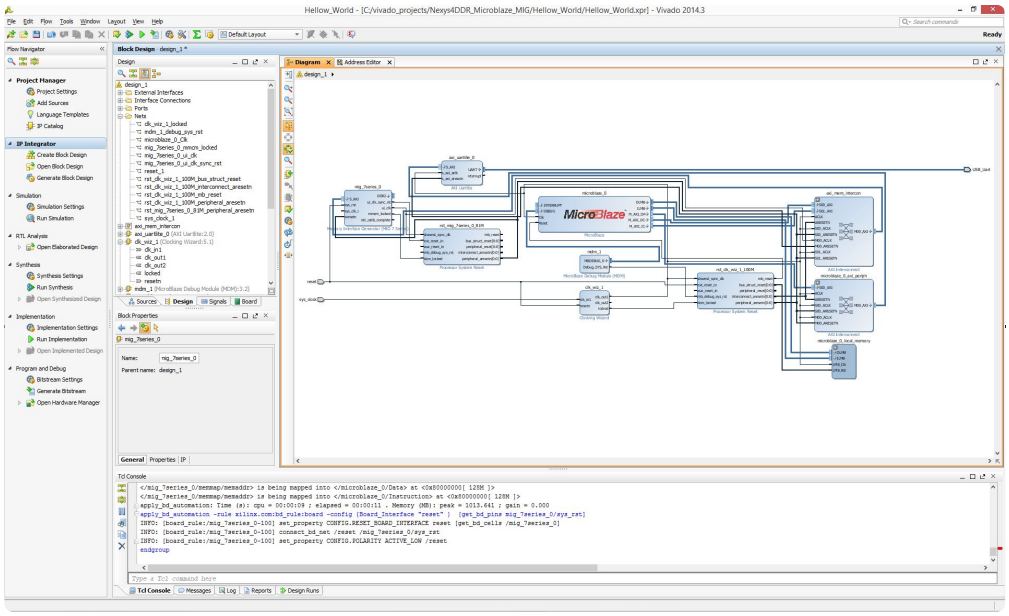
(https://digilent.com/reference/_detail/vivado/mig_28.jpg?id=learn%3Aprogrammable-logic%3Atutorials%3Ahtsspisf%3Astart)

- **Select only the mig_7series_0 in the connection automation list. Do not select Microblaze section in this step**. Click OK.



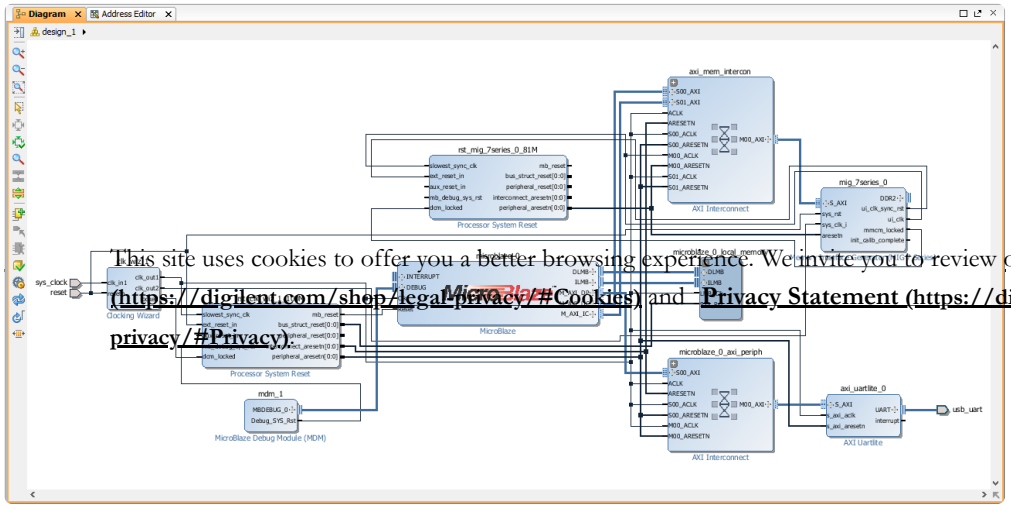
(https://digilent.com/reference/_detail/vivado/mig_29.jpg?id=learn%3Aprogrammable-logic%3Atutorials%3Ahtsspisf%3Astart)

- New signal connections will be made and displayed.



(https://digilent.com/reference/_detail/vivado/mig_30.jpg?id=learn%3Aprogrammable-logic%3Atutorials%3Ahtsspisf%3Astart)

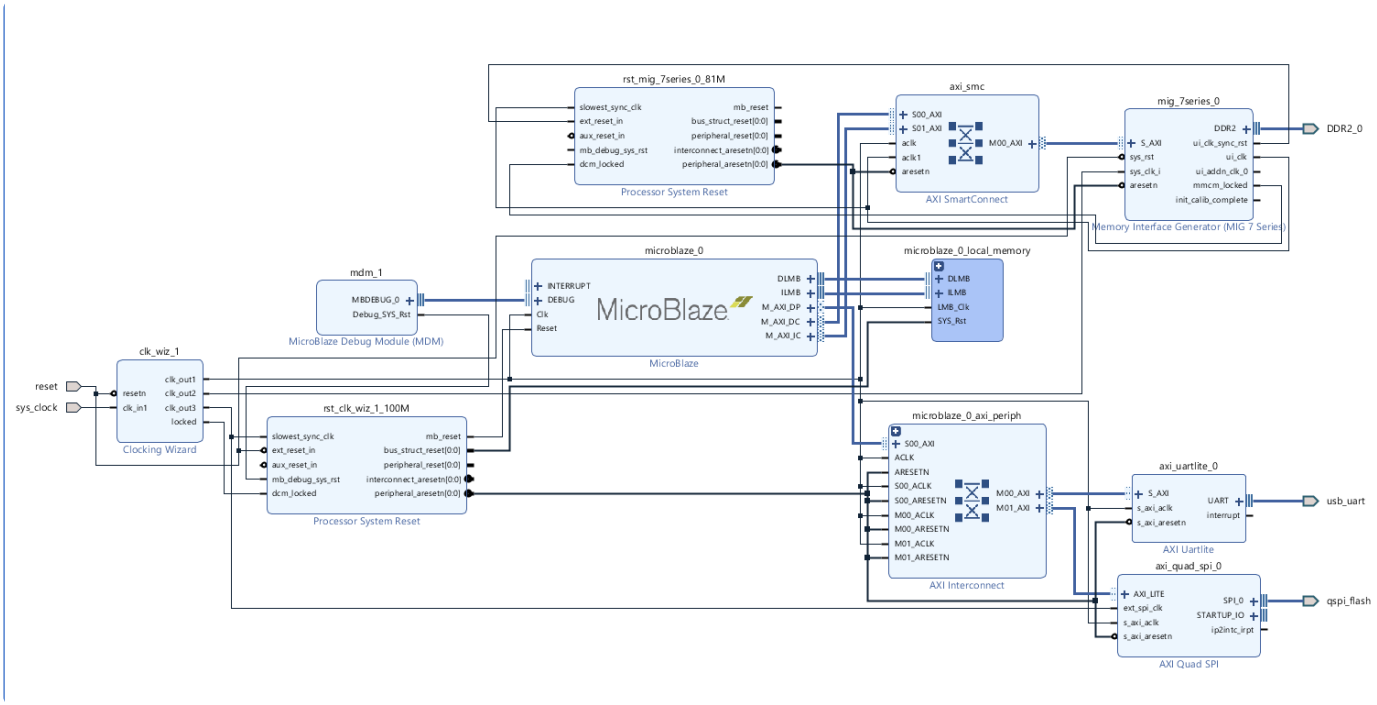
- Select the button circled in blue. This is the **Regenerate Layout** option that will re-arrange the IP blocks in the design.



(https://digilent.com/reference/_media/nexys4-ddr/nexys4ddr-secmancon2.png)

11. Regenerate layout and validate design

The result should look like in the below image representing the Block Diagram:



12. Create HDL wrapper

Right click on the Block Design, and select the Create HDL Wrapper option.

13. Synthesize

Open the synthesized design, and under I/O Ports tab below the page, check the name of the signals to be written in the XDC file.

14. Create an XDC file and add the constraints in it

You can copy paste the syntax from the Master XDC file for Nexys4DDR (https://github.com/Digilent/digilent-xdc/blob/master/Nexys-4-DDR-Master.xdc)

XDC File signals

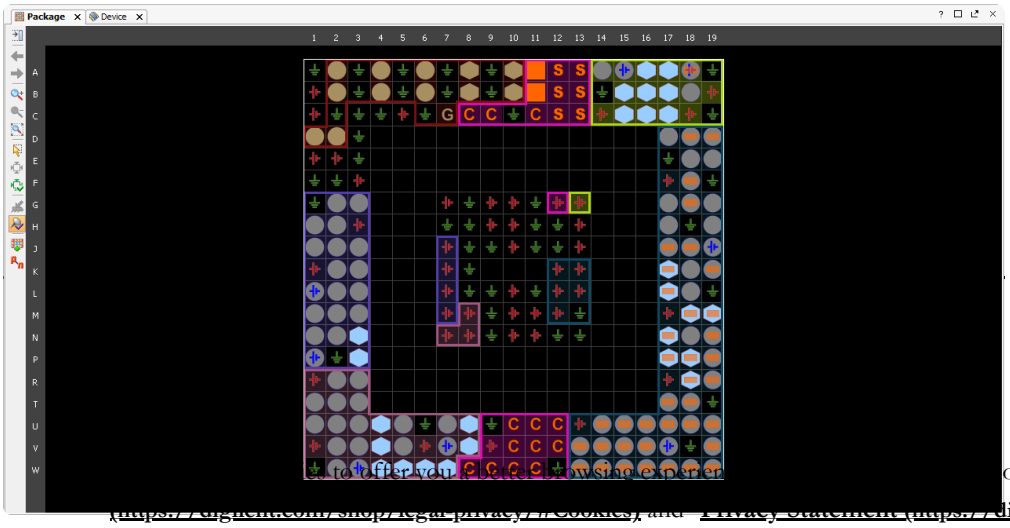
```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level s

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { sys_clock }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {sys_clock
set_property -dict {PACKAGE_PIN C12 IOSTANDARD LVCMOS33} [get_ports {reset}]

#compress bit file
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]

#QSPI signals
set_property -dict {PACKAGE_PIN K17 IOSTANDARD LVCMOS33} [get_ports {qspi_flash_io0_io}
set_property -dict {PACKAGE_PIN K18 IOSTANDARD LVCMOS33} [get_ports {qspi_flash_io1_io}
set_property -dict {PACKAGE_PIN L14 IOSTANDARD LVCMOS33} [get_ports {qspi_flash_io2_io}
set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports {qspi_flash_io3_io}
set_property -dict {PACKAGE_PIN L13 IOSTANDARD LVCMOS33} [get_ports {qspi_flash_ss_io}]

#USB-UART signals
set_property -dict {PACKAGE_PIN D4 IOSTANDARD LVCMOS33} [get_ports {usb_uart_txd}]
set_property -dict {PACKAGE_PIN C4 IOSTANDARD LVCMOS33} [get_ports {usb_uart_rxd}]

#DDR2 Memory signals\\
set_property -dict {PACKAGE_PIN R7 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[0]}]
set_property -dict {PACKAGE_PIN V6 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[1]}]
set_property -dict {PACKAGE_PIN R8 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[2]}]
set_property -dict {PACKAGE_PIN U7 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[3]}]
set_property -dict {PACKAGE_PIN V7 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[4]}]
set_property -dict {PACKAGE_PIN R6 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[5]}]
set_property -dict {PACKAGE_PIN U6 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[6]}]
set_property -dict {PACKAGE_PIN R5 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[7]}]
set_property -dict {PACKAGE_PIN T5 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[8]}]
set_property -dict {PACKAGE_PIN U3 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[9]}]
set_property -dict {PACKAGE_PIN V5 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[10]}]
set_property -dict {PACKAGE_PIN U4 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[11]}]
set_property -dict {PACKAGE_PIN V4 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[12]}]
set_property -dict {PACKAGE_PIN T4 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[13]}]
set_property -dict {PACKAGE_PIN V1 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[14]}]
set_property -dict {PACKAGE_PIN T3 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dq[15]}]
set_property -dict {PACKAGE_PIN T6 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dm[0]}]
set_property -dict {PACKAGE_PIN U1 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_dm[1]}]

set_property -dict {PACKAGE_PIN U9 IOSTANDARD DIFF_SSTL18_II} [get_ports {DDR2_0_dqs_p[
set_property -dict {PACKAGE_PIN V9 IOSTANDARD DIFF_SSTL18_II} [get_ports {DDR2_0_dqs_n[
set_property -dict {PACKAGE_PIN U2 IOSTANDARD DIFF_SSTL18_II} [get_ports {DDR2_0_dqs_p[
set_property -dict {PACKAGE_PIN V2 IOSTANDARD DIFF_SSTL18_II} [get_ports {DDR2_0_dqs_n[
set_property -dict {PACKAGE_PIN N6 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[12]}]
set_property -dict {PACKAGE_PIN K5 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[11]}]
set_property -dict {PACKAGE_PIN R2 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[10]}]
set_property -dict {PACKAGE_PIN N5 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[9]}]
set_property -dict {PACKAGE_PIN L4 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[8]}]
set_property -dict {PACKAGE_PIN N1 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[7]}]
```

```
set_property -dict {PACKAGE_PIN M2 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[6]}]
set_property -dict {PACKAGE_PIN P5 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[5]}]
set_property -dict {PACKAGE_PIN L3 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[4]}]
set_property -dict {PACKAGE_PIN T1 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[3]}]
set_property -dict {PACKAGE_PIN M6 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[2]}]
set_property -dict {PACKAGE_PIN P4 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[1]}]
set_property -dict {PACKAGE_PIN M4 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_addr[0]}]

set_property -dict {PACKAGE_PIN R1 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_ba[2]}]
set_property -dict {PACKAGE_PIN P3 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_ba[1]}]
set_property -dict {PACKAGE_PIN P2 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_ba[0]}]

set_property -dict {PACKAGE_PIN L6 IOSTANDARD DIFF_SSTL18_II} [get_ports {DDR2_0_ck_p[0
set_property -dict {PACKAGE_PIN L5 IOSTANDARD DIFF_SSTL18_II} [get_ports {DDR2_0_ck_n[0

set_property -dict {PACKAGE_PIN N4 IOSTANDARD SSTL18_II} [get_ports DDR2_0_ras_n]
set_property -dict {PACKAGE_PIN L1 IOSTANDARD SSTL18_II} [get_ports DDR2_0_cas_n]
set_property -dict {PACKAGE_PIN N2 IOSTANDARD SSTL18_II} [get_ports DDR2_0_we_n]
set_property -dict {PACKAGE_PIN M1 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_cke[0]}]
set_property -dict {PACKAGE_PIN M3 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_odt[0]}]
set_property -dict {PACKAGE_PIN K6 IOSTANDARD SSTL18_II} [get_ports {DDR2_0_cs_n[0]}]
```

15. Compress Bitstream (Optional)

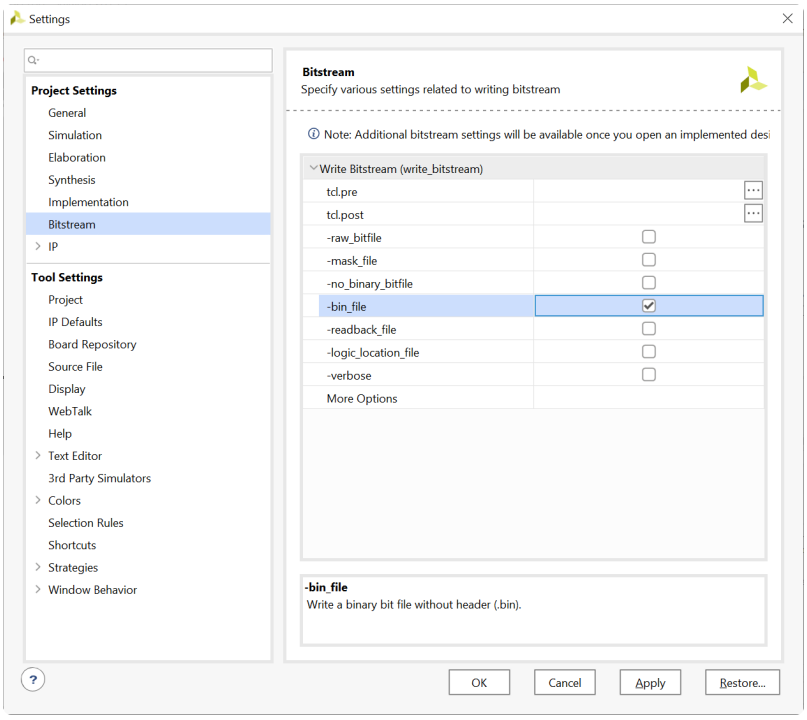
In certain applications (often when using the Cmod A7 with Microblaze) you might find that you do not have the necessary space in flash in order to store both your program and bitstream configuration. If this is the case it would be worth a try to compress your bitstream in order to get it to fit. Note that using a compressed bitstream in flash will not only take up less space but will also improve initial FPGA programming speeds. These steps take place in Vivado before the bitstream gets generated.

- With your design ready to be generated into a bitstream select Run Implementation from the Flow Navigator.
- Once the implementation has completed either open the implemented design from the pop up window with radio buttons or select Open Implemented Design from the Flow Navigator. The resulting window should look similar to the following:



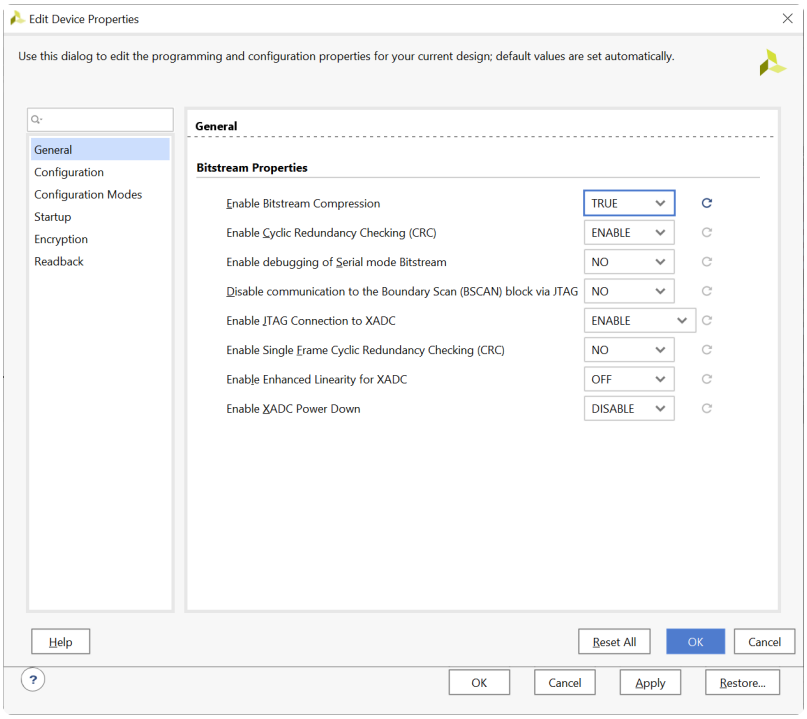
(https://digilent.com/reference/_media/impdesign.png)

- With the Implemented Design still on screen select Bitstream Settings and the bitstream settings should pop up.
- Inside of the Bitstream settings check the **-bin_file** box.



(https://digilent.com/reference/_detail/learn/programmable-logic/tutorials/htsspisf/bitcompress.png?id=learn%3Aprogrammable-logic%3Atutorials%3Ahtsspisf%3Astart)

- Next click on **Configure additional bitstream settings** highlighted in blue at the top of the window. On the following window change the property **Enable Bitstream Compresssion** to **TRUE** then select OK until both windows are closed.

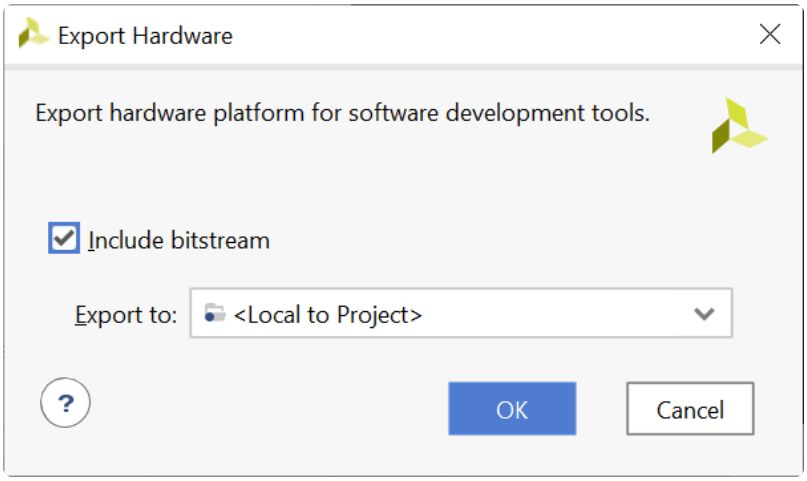


(https://digilent.com/reference/_detail/learn/programmable-logic/tutorials/htsspisf/bitcompresst.png?id=learn%3Aprogrammable-logic%3Atutorials%3Ahtsspisf%3Astart)

- Now Vivado has been configured to output a compressed bitstream which will transfer to your SDK project. Select Generate Bitstream and a window might pop up asking you to save your XDC. If so, give the XDC a name and save it. Your bitstream should start generating afterwards.

16. Export Hardware Handoff

From File→Export→Export Hardware, make sure to check "Include bitstream" setting:

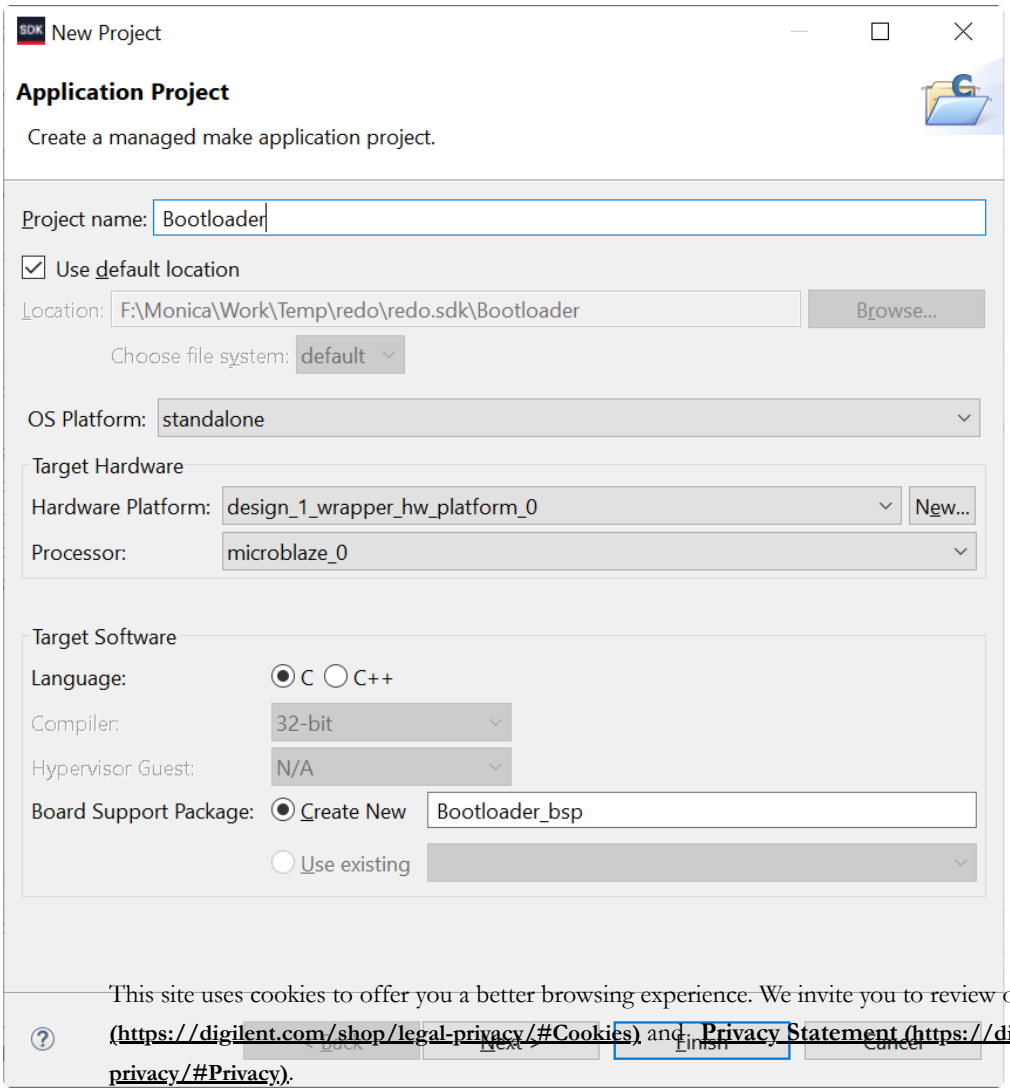


SDK Steps to create a bootloader and program the board

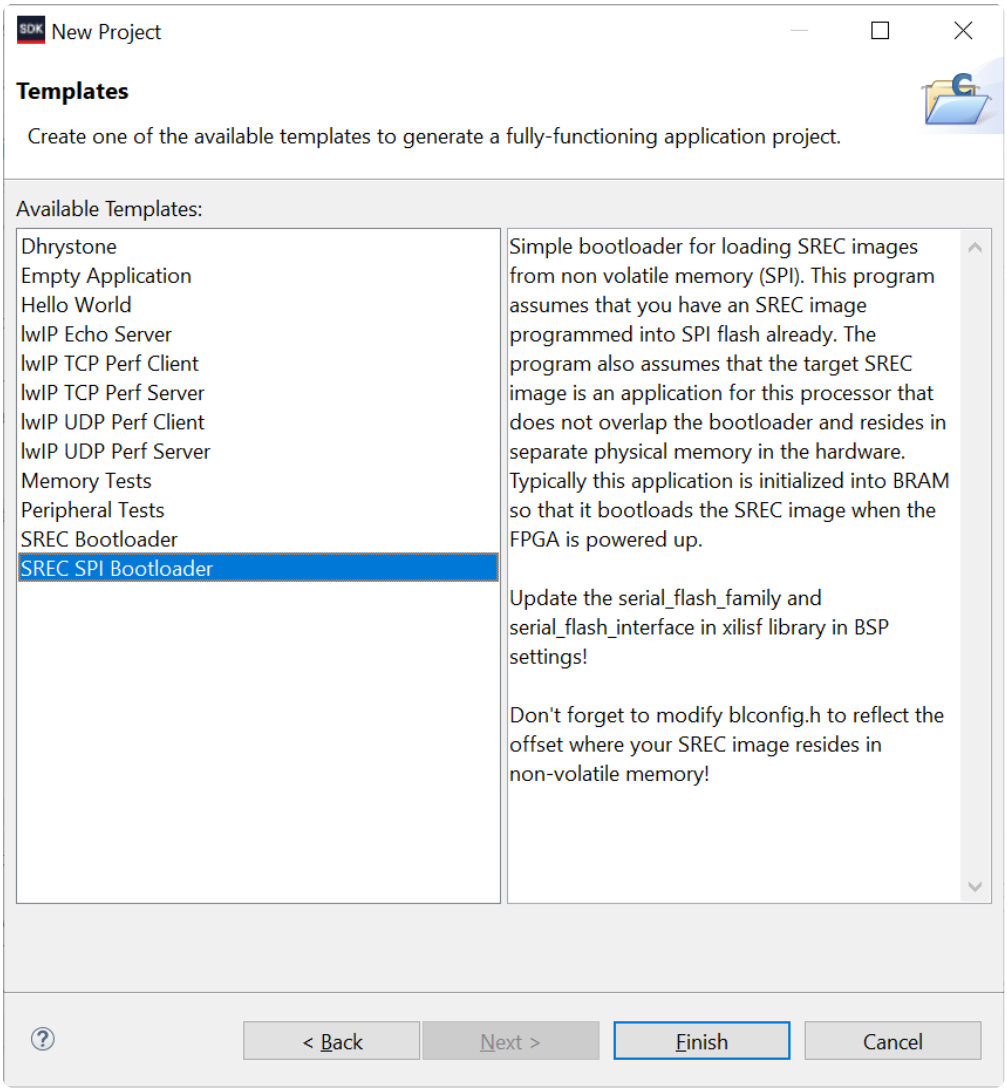
1. Create SPI Bootloader and BSP

- To create the bootloader, go to *File* > *New* > *Application Project*.

Name the bootloader then hit Next.



- Select **SREC SPI Bootloader** and select **Finish**.

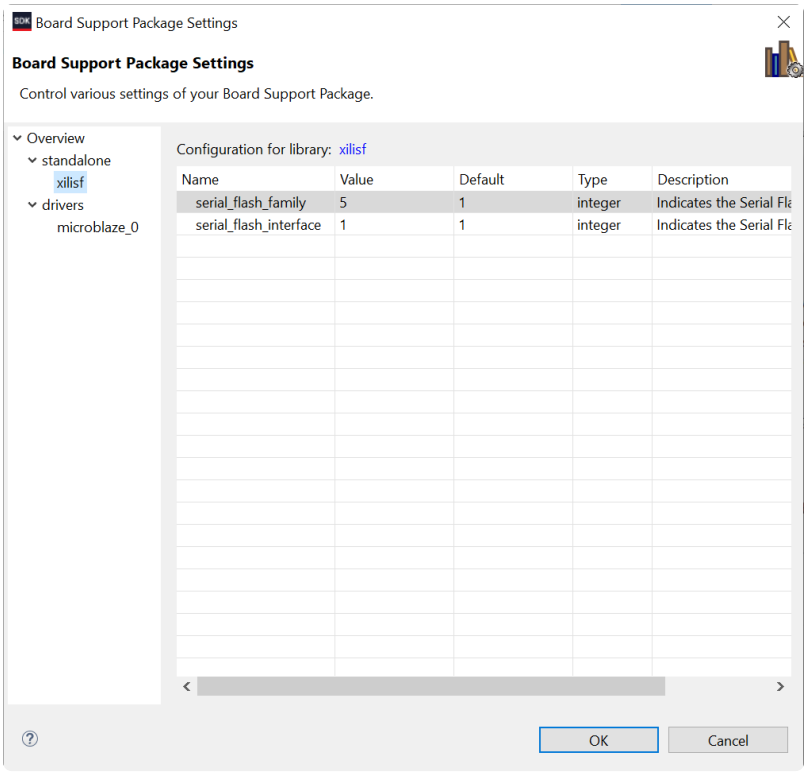


2. Configure Bootloader BSP

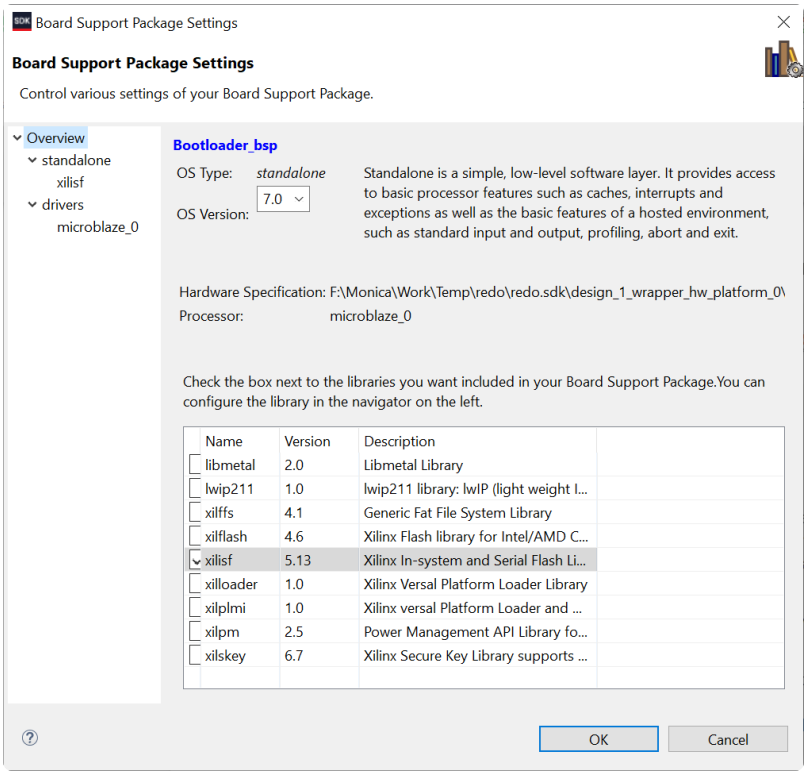
* Open the BSP project and open system.mss. Click “Modify this BSP's Settings”. The memory device info needs to be passed to the xilifs library (*Overview* > *standalone* > *xilifs*).

- *serial_flash_family = 5(Spansion/Micron/Macronix)*
- serial_flash_interface = 1(AXI SPI)

Click “Regenerate BSP sources”.



Next we should check that the BSP is updated in order to use the updated version of **xilisf**. To do this, right click on the BSP and select **Board Support Package settings** and make sure that **xilisf** is checked.



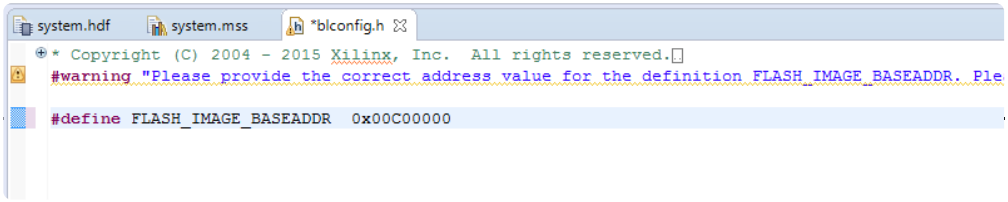
- In your newly created bootloader application, open **blconfig.h** located in bootloader/src/blconfig.h and change **FLASH_IMAGE_BASEADDR** to suit your needs. Save the file in order to rebuild the bootloader project to contain the latest updates.

It is important to note that this value is the offset that the image will have in Flash.

NOTE: If you are using the Cmod-A7 try the offset: 0x00300000.

NOTE: If you are using the Nexys 4 DDR or Arty-A7-100T try the offset: 0x003D0900.

NOTE: If you are using the Arty-A7-35T, Nexys Video or the Genesys 2 try the offset: 0x00C00000.



(https://digilent.com/reference/_media/learn/programmable-logic/tutorials/htsspisf/flash_image_baseaddr.png)

- In order to increase the speed even further we can shorten and silence our bootloader. To do this navigate to *srec_spi_bootloader* > *src* > *bootloader.c*, scroll down to where the preprocessor directive **VERBOSE** is declared and comment it out.

```
/* Comment the following line, if you want a smaller and faster bootloader which will be silent */
#define VERBOSE
```

NOTE: If you are using a board with Macronix flash follow the additional steps:

- Open **bootloader.c** from the sources and scroll down until you find the call to **XIsf_Initialize**. Double click the function name to highlight it, right click and select "Open Declaration". This will open **xilisf.c**.
- Add the following line somewhere near the top of **xilisf.c**

```
#define XISF_MACRONIX_DEV_MX25L3233F 0x2016  /**< Device ID for MX25L3233F */
```

- In **xilisf.c** find the definition for IntelStmDevices[] and add the following:

```
{XISF_MANUFACTURER_ID_MACRONIX, XISF_MACRONIX_DEV_MX25L3233F,
XISF_BYTES256_PER_PAGE, XISF_PAGES256_PER_SECTOR,
XISF_NUM_OF_SECTORS64}
```

- Save **xilisf.c**

Important if you are using a board with Macronix flash.

The source files that need editing are part of a library (https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841939/xilisf (https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841939/xilisf)) compiled into a bsp/domain. By navigating to the source files in SDK/Vitis, the local copy of the library's source files are opened. Any modification is done on this local copy and build into the static library (and application) upon build. However, when the source files of the bsp/domain are re-generated (context menu, Regenerate BSP sources), the changes made as explained above are overwritten from the originals in the Xilinx install directory. Take care this does not happen.

A more permanent solution is forking the library repository (https://github.com/Xilinx/embeddedsw (https://github.com/Xilinx/embeddedsw)) editing the sources there and including a path to the modified repository in SDK/Vitis Xilinx→Repositories→Global.

- Make sure to re-build the application after making these changes.

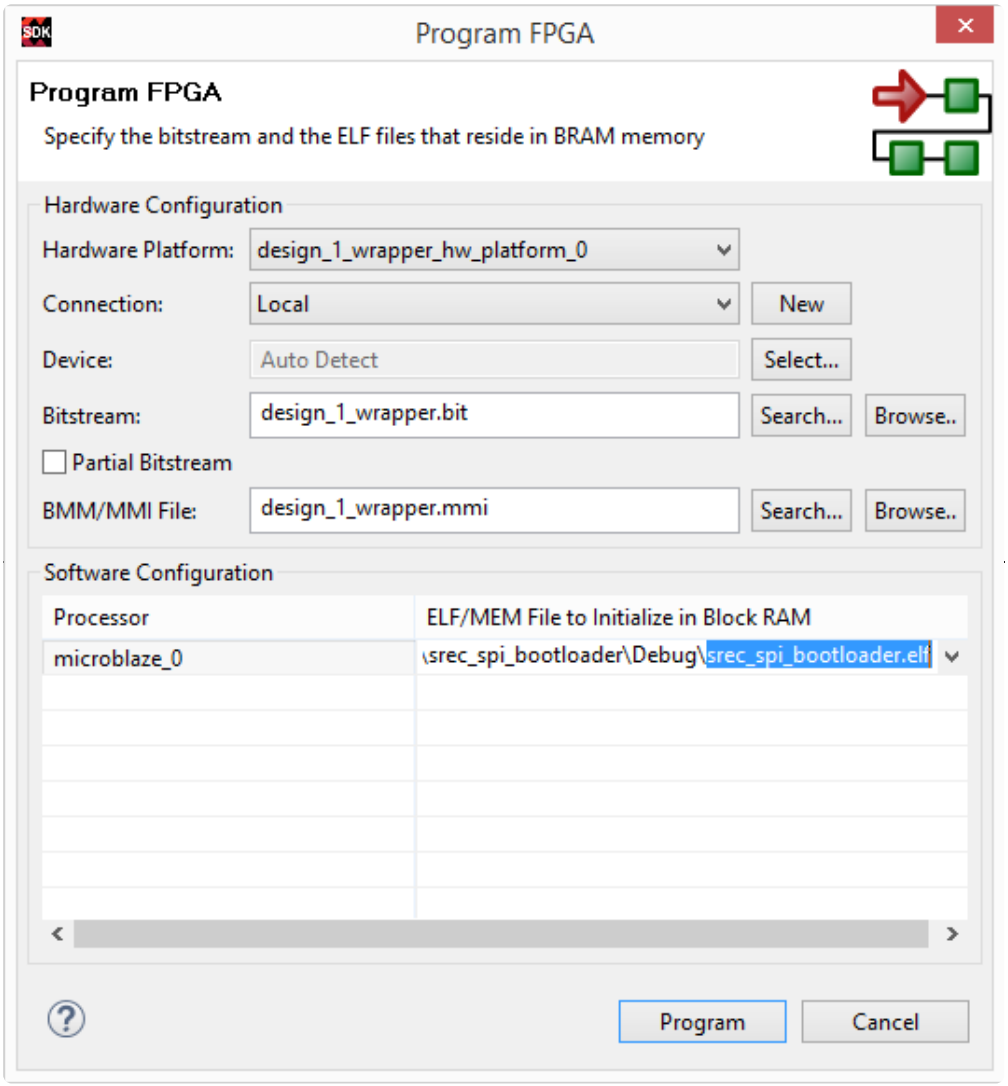
3. Program FPGA with bootloader

The hardware design including bootloader will be programmed into flash such that the FPGA loads it every time it boots. Configure the SPI jumper on the Nexys4 DDR board on the QSPI position. This allows the FPGA to load its bit file from the SPI flash. The FPGA bit file contains the configuration of the FPGA including initial contents of the Microblaze local BRAM.

Next connect the Nexys4 DDR board via the JTAG usb connector. Start a serial terminal (eg. Teraterm) and select the port and set the baud rate to 9600 (as configured in the UARTlite block design).

Select the bootloader.elf file (in the Bootloader/Debug project) instead of "bootloop". Selecting "program" will generate a download.bit file in the "design_wrapper_hw_platform_0" project.

- Open **Program FPGA** (*Xilinx Tools* > *Program FPGA*) and select the bootloader.ELF under **ELF/MEM File to Initialize in block RAM ()**, and select **Program** to continue.



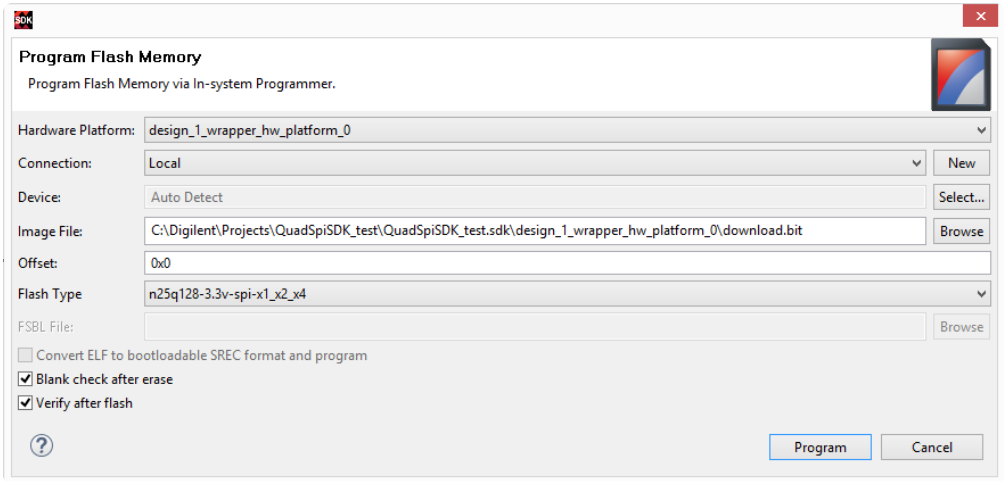
(https://digilent.com/reference/_media/learn/programmable-logic/tutorials/htsspisf/programfpga.png)

This will run update_mem, and will output a download.bit file.

4. Flashing FPGA

The **Program Flash Memory** utility will be used again to store the bitstream initialized with the bootloader in BRAM. Move the jumper on the board in the JTAG position. The **Image File** is the **download.bit** that was generated previously, and should be located inside of your hardware platform folder.The **Offset** is 0x0 since the bitstream will be loading our project into memory. Select **Program** to continue. The flash type is "s25fl128sxxxxx0-spi-x1_x2_x4". Check "verify after flash". And finish by clicking "Program".

After flashing you can restart the Nexys4 DDR by pressing the "prog" button. Note that while the FPGA can load its bit file from flash it can also still be programmed as usual using JTAG.

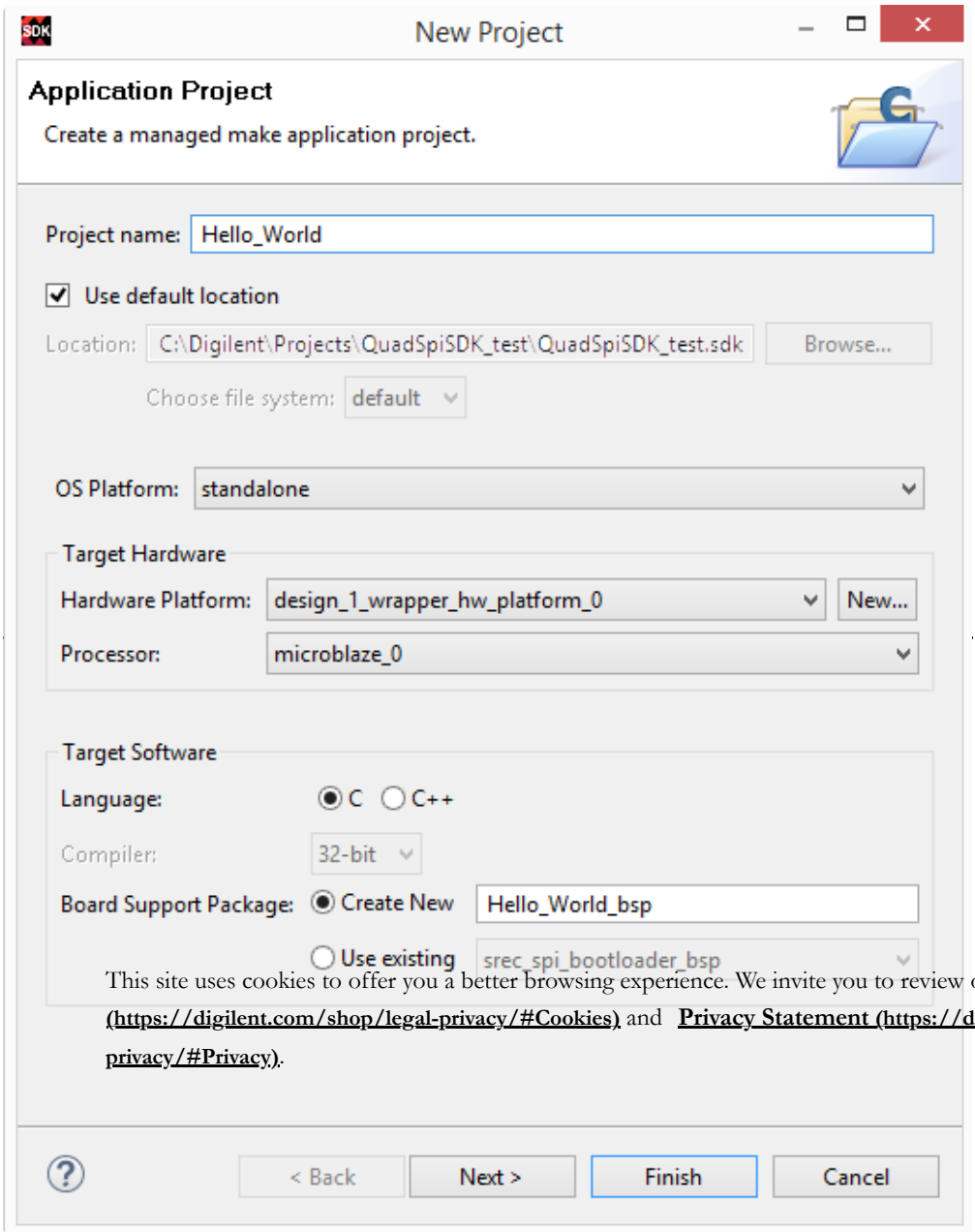


(https://digilent.com/reference/_media/learn/programmable-logic/tutorials/htsspisf/progflash_bit.png)

5. Create a User Application

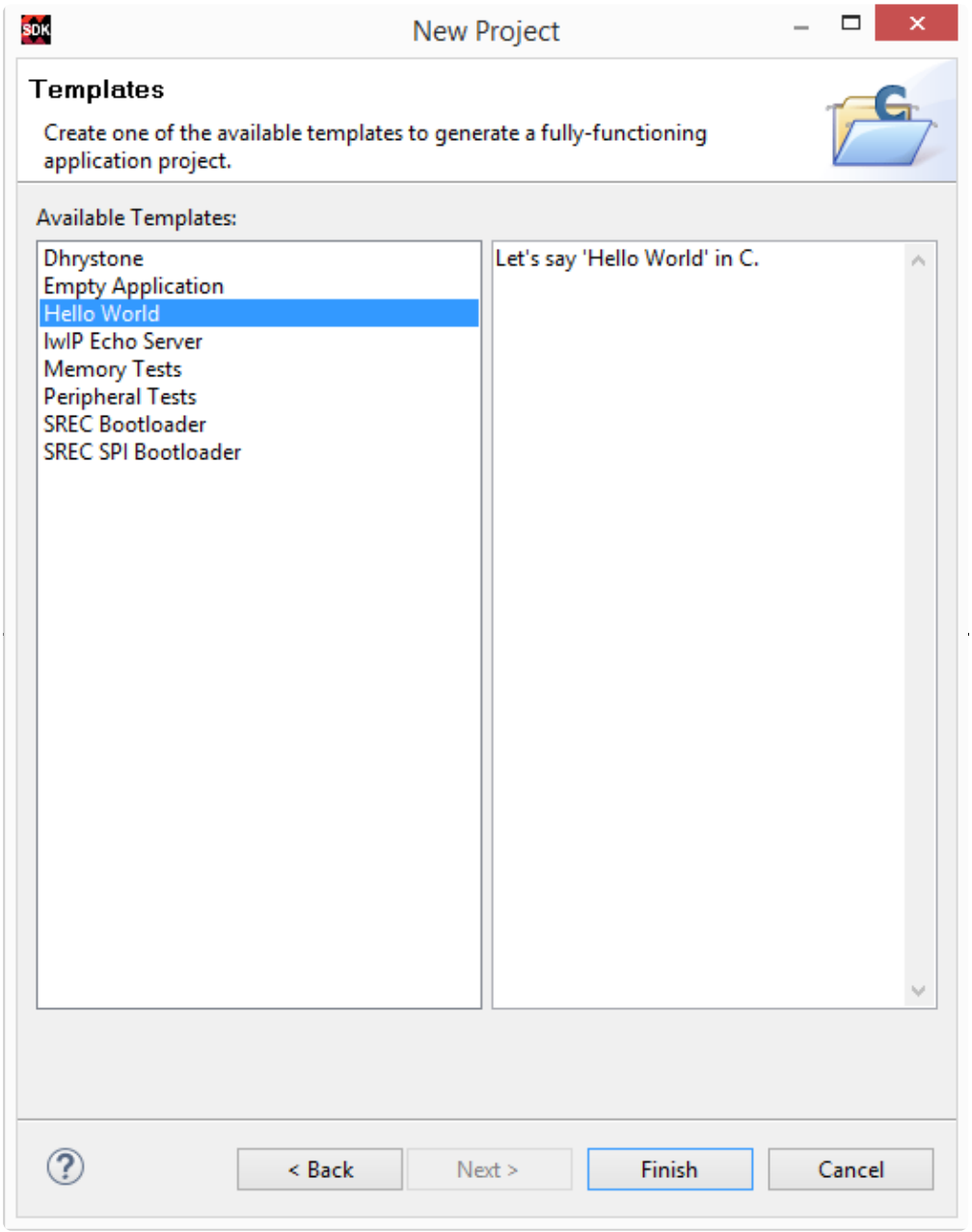
This application will be the one that gets stored into Flash. For this tutorial we will be creating a simple **Hello World** project but the process applies to any project that you would want stored in flash.

- Create a **Hello World** application. *File* > *New* > *Application Project* and name your app. Then select **Next**.



(https://digilent.com/reference/_media/learn/programmable-logic/tutorials/htsspisf/hello_world_1.png)

- Select the **Hello World** application template.

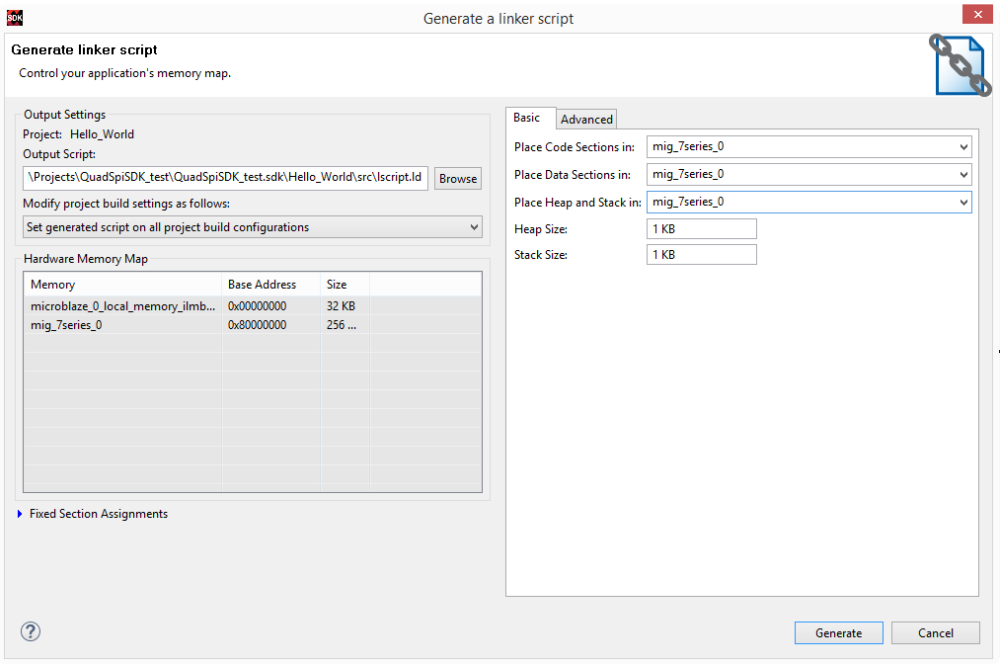


(https://digilent.com/reference/_media/learn/programmable-logic/tutorials/htsspisf/hello_world_2.png)

6. Configure User App BSP

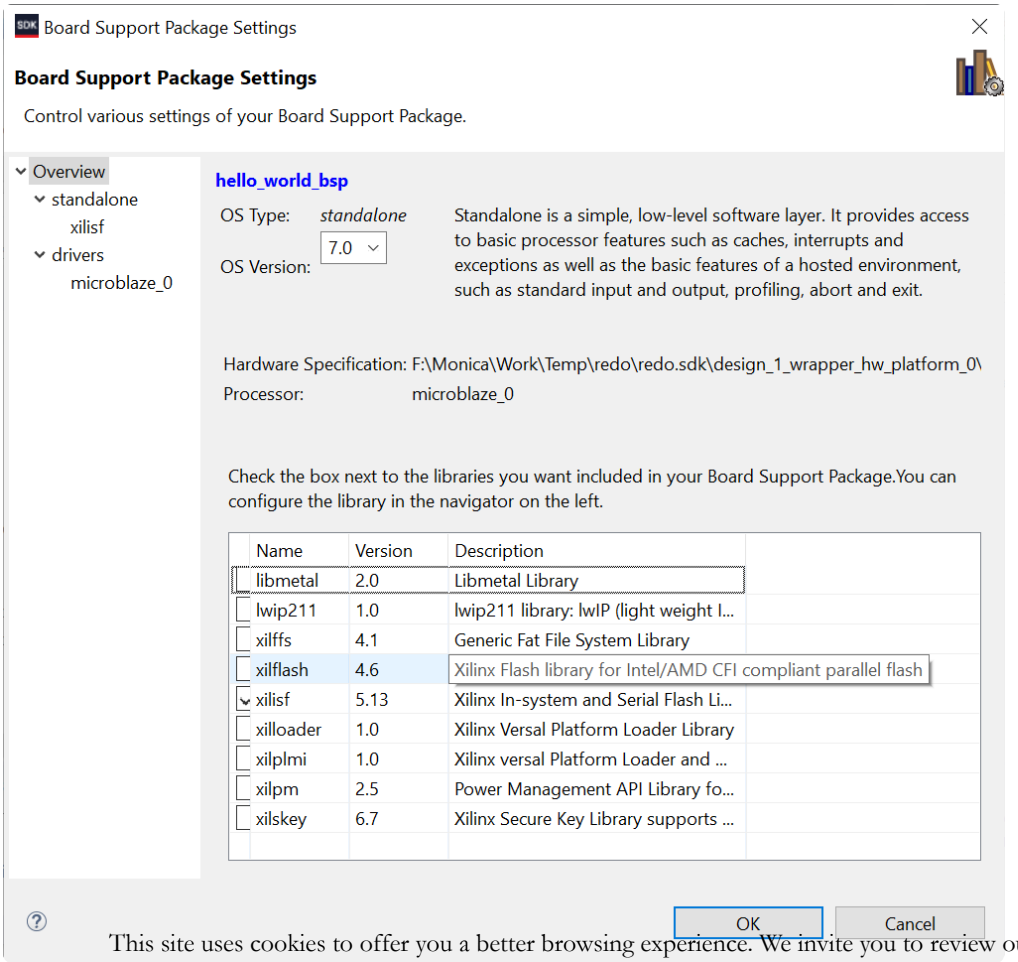
- Next we need to make sure to store our program in DDR. In order to do this, right click on the **Hello World** application, and select **Generate Linker Script**.

Place all sections into DDR:

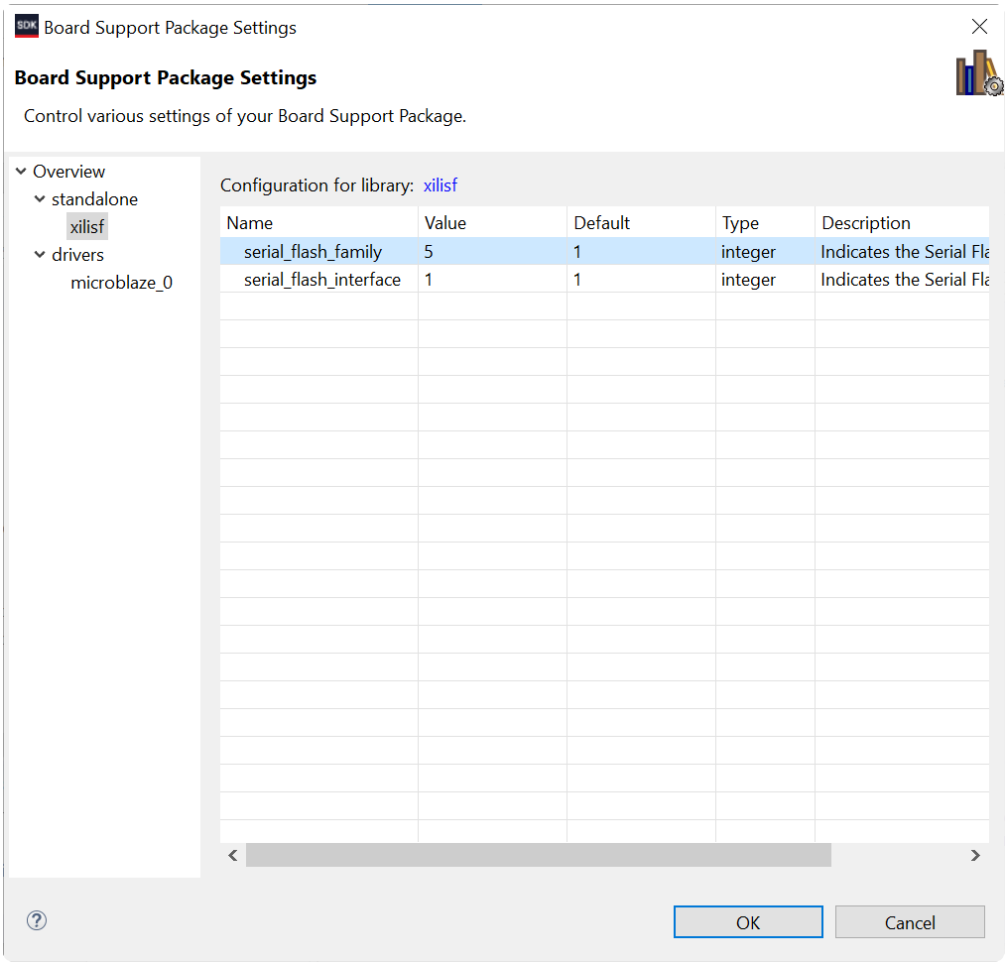


(https://digilent.com/reference/_media/learn/programmable-logic/tutorials/htsspisf/generatelinker.png)

- Open the Hello_World_bsp project and open "system.mss". Click "Modify this BSP's Settings". Enable "xilisf" in the supported libraries pane.



Select "xilisf" in the navigation tree view. Set the "serial_flash_family" type to 5 (Spansion).

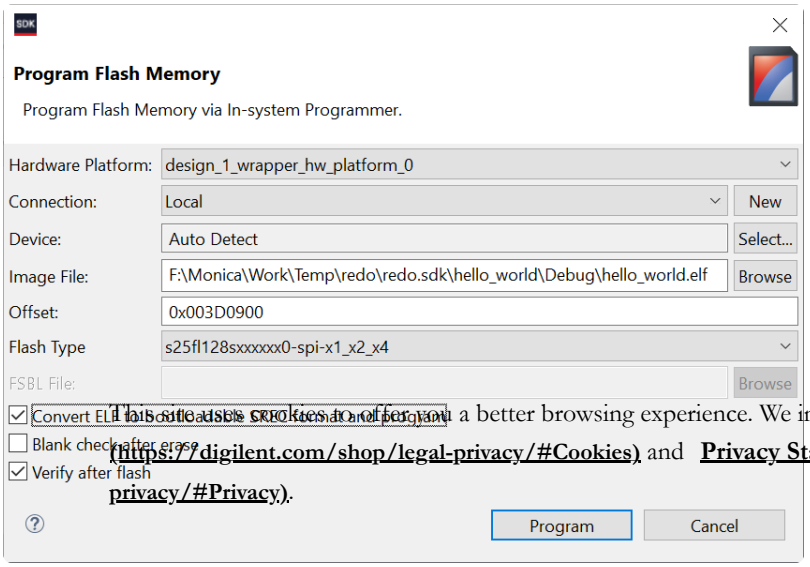


7. Program Flash

In this step we will use the SDK **Program Flash Memory** utility to program our **Hello World** application to Flash.

- Go to *Xilinx Tools* > *Program Flash*. Change the **Offset** to the value used in blconfig.h before: 0x003D0900.
- The flash type is "s25fl128sxxxxx0-spi-x1_x2_x4". Tick "convert elf file to bootable SREC…". Check "verify after flash". And finish by clicking "Program".

Reset the Nexys4 DDR by pressing the "prog" button. Move the jumper back to QSPI position, start the board again, and run the User application from SDK, check the message displayed in the serial terminal. The bootloader should start loading and eventually run the user application.



Test

The flash has now been programmed with our Hello World, SREC, bitstream, and bootloader. With your board connected to a serial port reset the board. This will configure the FPGA and run our Hello World application!

learn (https://digilent.com/reference/tag/learn?do=showtag&tag=learn), programmable-logic (https://digilent.com/reference/tag/programmable-logic?do=showtag&tag=programmable-logic), tutorial (https://digilent.com/reference/tag/tutorial?do=showtag&tag=tutorial), nexys-4-ddr (https://digilent.com/reference/tag/nexys-4-ddr?do=showtag&tag=nexys-4-ddr), nexys-video (https://digilent.com/reference/tag/nexys-video?do=showtag&tag=nexys-video), arty (https://digilent.com/reference/tag/arty?do=showtag&tag=arty), cmod-s6 (https://digilent.com/reference/tag/cmod-s6?do=showtag&tag=cmod-s6), cmod-a7 (https://digilent.com/reference/tag/cmod-a7?do=showtag&tag=cmod-a7), microblaze (https://digilent.com/reference/tag/microblaze?do=showtag&tag=microblaze)