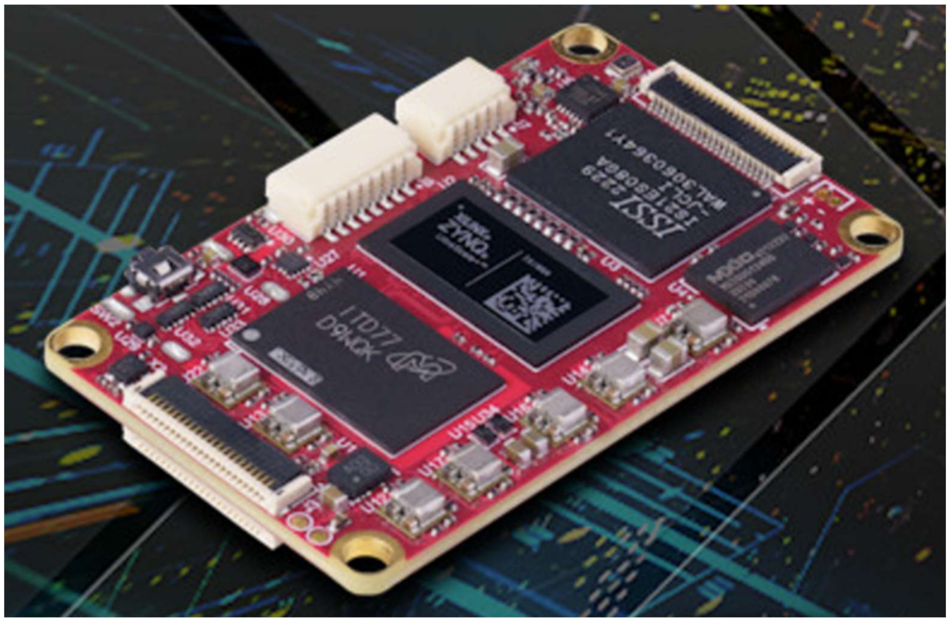
Unit / Module Description:	Zynq UltraScale MPSoC + 9-axis IMU + 4x FPC
Unit / Module Number:	VCS3
Document Issue Number:	1.1
Issue Date:	15/07/2025
Original Author:	Paul Dorrell

VCS3 Application Starter's Guide



Revision History

Issue	Changes Made	Date	Initials
1.0	First issue.	18/03/2025	PD
1.1	Added appendix (linked on p6, p39), added note on p38, corrected Fig. 48 to dual parallel flash type, added Fig. 50 and 51 on p42.	15/07/2025	NS

Table of Contents

Table of Figures

1 Introduction

This document is a guide for those who own a VCS-3 and would like to develop applications to run on the Zynq MPSoC, either or both the PS (processor side) and the PL (programmable logic). Presented here are basic steps to create simple projects with the various AMD/Xilinx tools.

The hardware development kit is comprehensively described in companion document “VCS3 Development Kit Getting Started Guide”. The user is strongly advised to read this as it contains details of JTAG debug setup needed for application development.

2 Developing for VCS3 using Vivado and Vitis

The VCS3 has been used in two versions of Vivado, 2022.2 and 2023.2. Although the Zynq MPSoC XCZU3EG device has been supported since 2017.x, it is recommended always to use a newer version.

Sundance Multiprocessor Technology LTD provides documentation and resources in GitHub:

https://github.com/SundanceMultiprocessorTechnology/

Board files from Sundance are accessible at repository:

https://github.com/SundanceMultiprocessorTechnology/VCS-3/tree/main/board_files

To include the board files in your system, download the corresponding files. They can be added at the installation path of Vivado, normally:

<installationpath>/Xilinx/Vivado/2023.2/data/xhub/boards

Old version of the tool used the following path:

<installationpath>/Xilinx/Vivado/20XX.X/data/boards/board_files/

Depending on how these files are copied, you may need to change the access privilege to allow Vivado to access them, for example:

```
root@Ubuntu-22-04:/tools/Xilinx/Vivado/2023.2/data/xhub/boards/Sundance#
chmod -R a+rX *
```

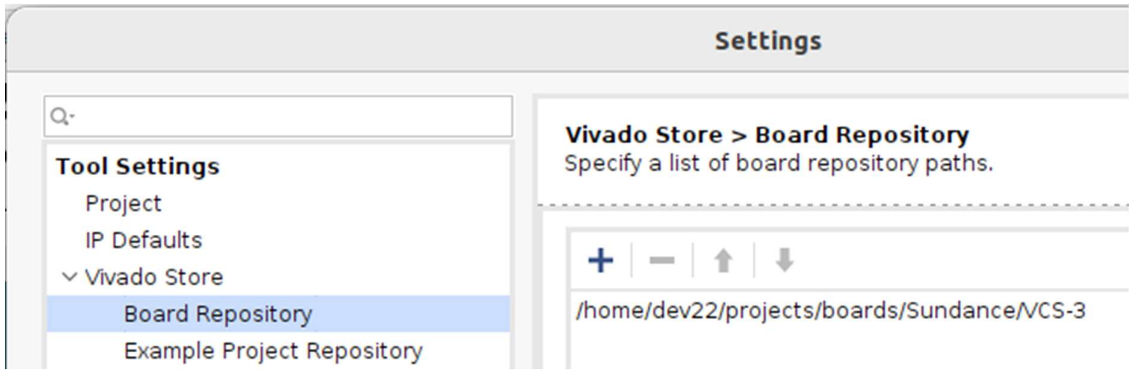
Furthermore, the above command may not work unless you have already elevated yourself to root using the command:

```
dev22@Ubuntu-22-04:/$ su
Password:
root@Ubuntu-22-04:/# whoami
root
```

NOTE: all the examples shown in this guide use Vivado 2023.2 and Vitis

[If you encounter issues when cloning the repository, see section 5.1]

An easier way to install the board files in your system is to tell Vivado where you have copied them using the Tools Settings as shown in this example.

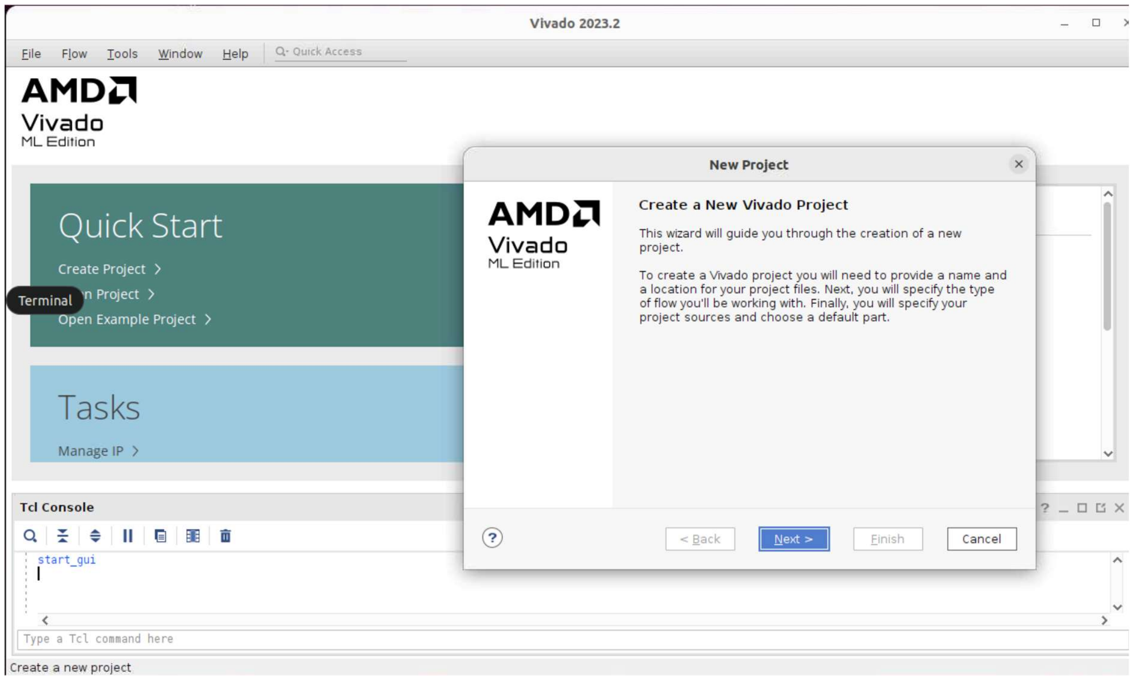


Vivado Store > Board Repository
Specify a list of board repository paths.

/home/dev22/projects/boards/Sundance/VCS-3

Figure 1 – Setting path to board files

Having Vivado open, create a new project, either on "Create Project", or File→Project→New:



Create a New Vivado Project

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

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

```
start_gui
```

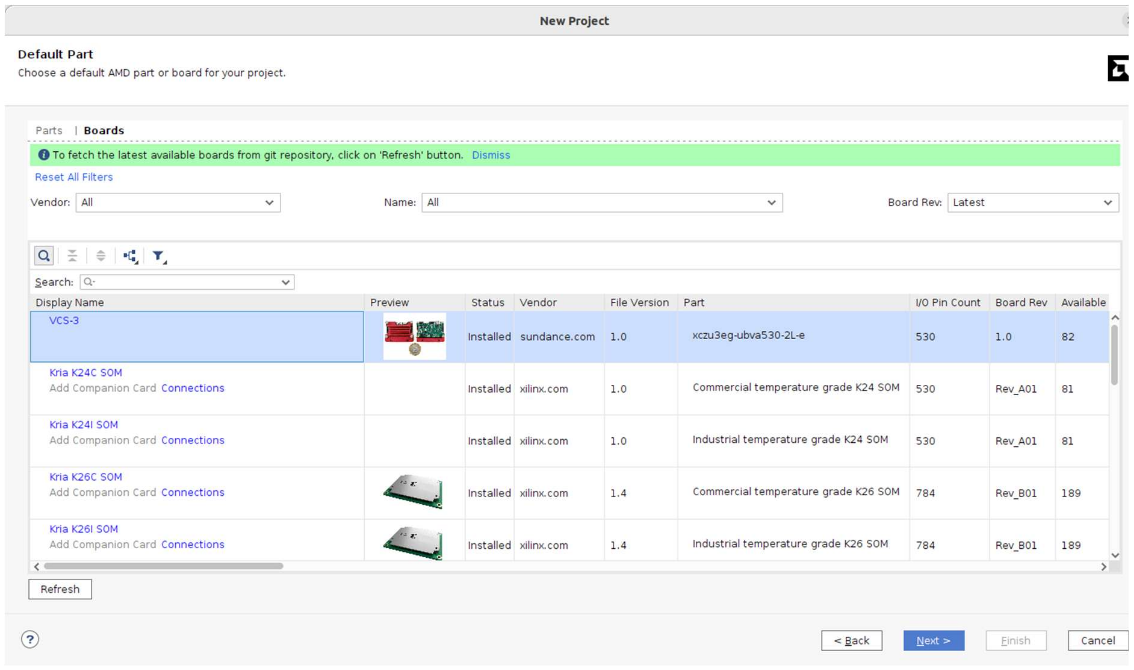
Type a Tcl command here

Figure 2 - Creating a new project

Click "Next" and select the path of the project.

Choose RTL project and mark the square "Do not specify sources at this time" in case the user doesn't require importing any source and it's a blank project.

Then, when Vivado asks for a device part, select "Boards", and choose the corresponding board files.


New Project

Default Part

Choose a default AMD part or board for your project.

Parts | **Boards**

To fetch the latest available boards from git repository, click on 'Refresh' button. Dismiss

Reset All Filters

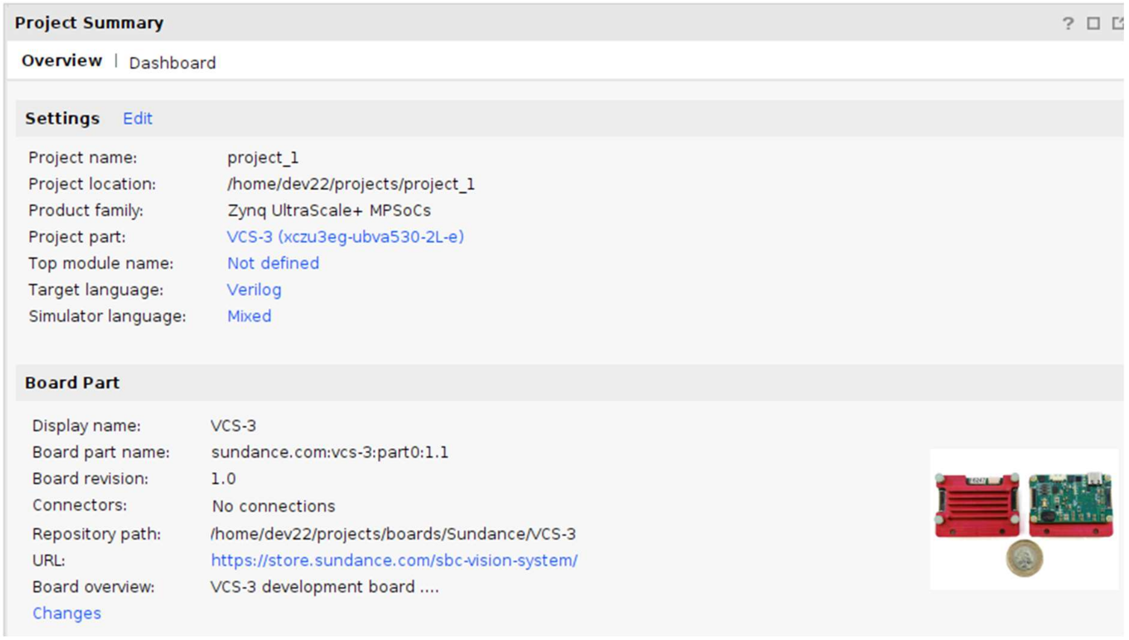
Vendor: All   Name: All   Board Rev: Latest

Search:

Display Name	Preview	Status	Vendor	File Version	Part	I/O Pin Count	Board Rev	Available
VCS-3		Installed	sundance.com	1.0	xczu3eg-ubva530-2L-e	530	1.0	82
Kria K24C SOM Add Companion Card Connections		Installed	xilinx.com	1.0	Commercial temperature grade K24 SOM	530	Rev_A01	81
Kria K24I SOM Add Companion Card Connections		Installed	xilinx.com	1.0	Industrial temperature grade K24 SOM	530	Rev_A01	81
Kria K26C SOM Add Companion Card Connections		Installed	xilinx.com	1.4	Commercial temperature grade K26 SOM	784	Rev_B01	189
Kria K26I SOM Add Companion Card Connections		Installed	xilinx.com	1.4	Industrial temperature grade K26 SOM	784	Rev_B01	189

Figure 3 – Selecting board file

Once the project is opened, the user can see the information about the board and the part used:


Project Summary

Overview | Dashboard

Settings Edit

Project name:	project_1
Project location:	/home/dev22/projects/project_1
Product family:	Zynq UltraScale+ MPSoCs
Project part:	VCS-3 (xczu3eg-ubva530-2L-e)
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

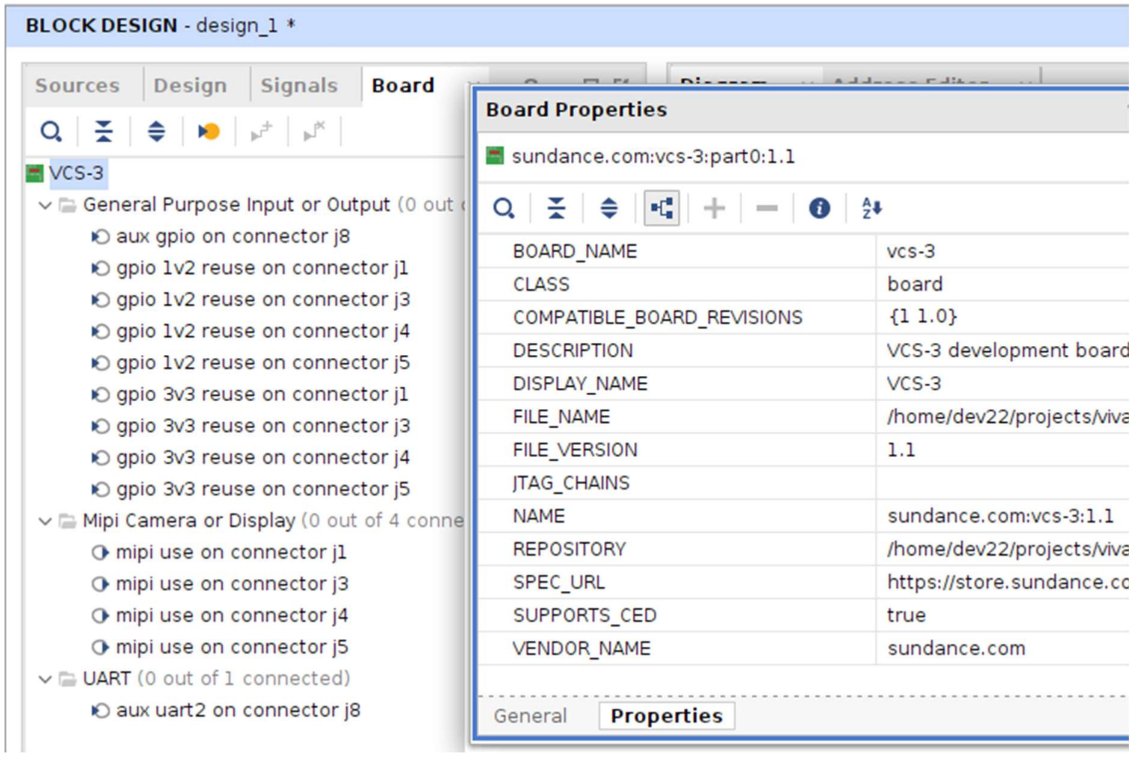
Board Part

Display name:	VCS-3
Board part name:	sundance.com:vcs-3:part0:1.1
Board revision:	1.0
Connectors:	No connections
Repository path:	/home/dev22/projects/boards/Sundance/VCS-3
URL:	https://store.sundance.com/sbc-vision-system/
Board overview:	VCS-3 development board ....

Changes

Figure 4 - Project information in Vivado

As soon as the user creates a new block design in IP Integrator, there will be some interfaces available, already set up and ready to use:



BLOCK DESIGN - design_1 *

Sources | Design | Signals | Board

VCS-3
- General Purpose Input or Output (0 out
  - aux gpio on connector j8
  - gpio 1v2 reuse on connector j1
  - gpio 1v2 reuse on connector j3
  - gpio 1v2 reuse on connector j4
  - gpio 1v2 reuse on connector j5
  - gpio 3v3 reuse on connector j1
  - gpio 3v3 reuse on connector j3
  - gpio 3v3 reuse on connector j4
  - gpio 3v3 reuse on connector j5
- Mipi Camera or Display (0 out of 4 conne
  - mipi use on connector j1
  - mipi use on connector j3
  - mipi use on connector j4
  - mipi use on connector j5
- UART (0 out of 1 connected)
  - aux uart2 on connector j8

Board Properties

sundance.com:vcs-3:part0:1.1

BOARD_NAME	vcs-3
CLASS	board
COMPATIBLE_BOARD_REVISIONS	{1 1.0}
DESCRIPTION	VCS-3 development board
DISPLAY_NAME	VCS-3
FILE_NAME	/home/dev22/projects/viva
FILE_VERSION	1.1
JTAG_CHAINS	
NAME	sundance.com:vcs-3:1.1
REPOSITORY	/home/dev22/projects/viva
SPEC_URL	https://store.sundance.cc
SUPPORTS_CED	true
VENDOR_NAME	sundance.com

Figure 5 - Board interfaces in IP Integrator

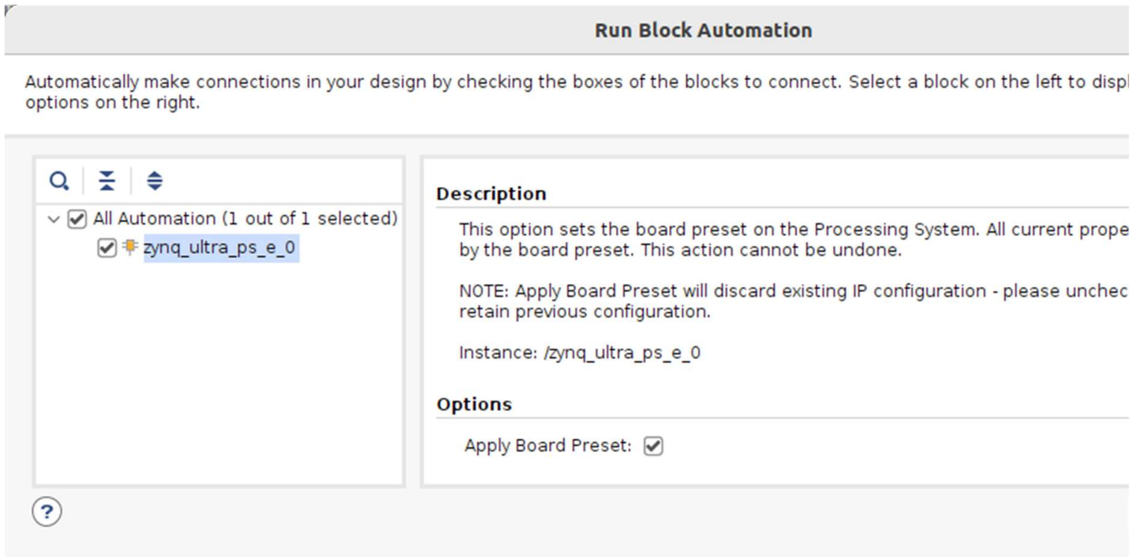
The board interfaces available are those connected to the PL side of the Zynq. Other board interfaces are accessible through the Zynq PS MIO pins as described in the Hardware section.

3 Hardware

In addition to “VCS3 Development Kit Getting Started Guide” the following sections show the raw pinout from the FPGA to peripherals or board connectors. However, board files are provided to make it easier for the application developer to interface to certain off-board peripheral such as cameras or displays. The board files also offer an easy way to repurpose board connector as general purpose io.

3.1 How can I setup the Zynq PS interfaces?

The processor side interfaces are easily setup by ensuring that “Apply Board Preset” is selected when adding “Zynq UltraScale+ MPSoc” IP core to your design and running “Block Automation”:



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to displ
options on the right.

All Automation (1 out of 1 selected)
zynq_ultra_ps_e_0

Description

This option sets the board preset on the Processing System. All current prope
by the board preset. This action cannot be undone.

NOTE: Apply Board Preset will discard existing IP configuration - please unchec
retain previous configuration.

Instance: /zynq_ultra_ps_e_0

Options

Apply Board Preset: ✓

Figure 6 – Running Block Automation in IP Integrator

The resulting configuration can be viewed by recustomizing this block and cancelling when finished.

The preset configures PS interfaces for

- Clocks into programmable logic (PL) @ 100MHz, 200MHz, and 25MHz
- Quad SPI Flash
- SD (eMMC)
- CAN
- I2C
- SPI x2
- UART x2
- GPIO (x21 to PS pins, x1 into PL)
- Timer/counter x3
- USB (including USB 3.0)
- LPDDR4 memory controller

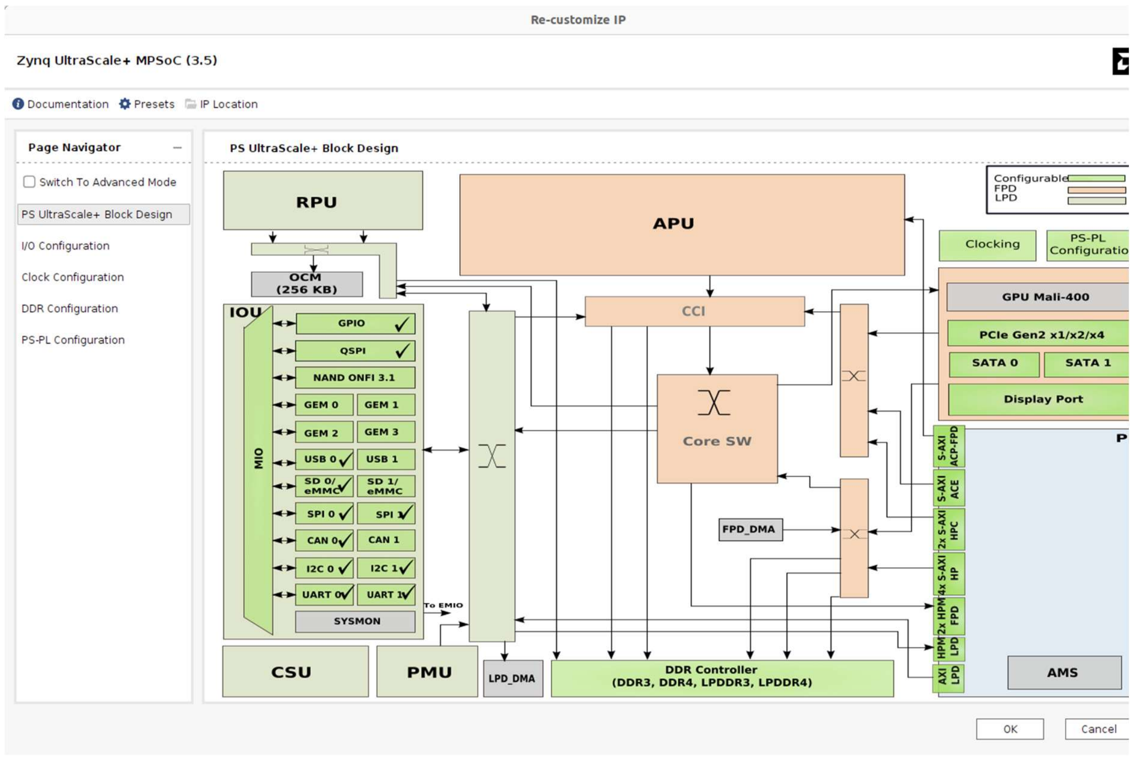


Figure 7 – Zynq configuration after Block Automation

3.2 How can I interface to the IMU?

The onboard 9-axis Inertial Measurement Unit (DS-000189-ICM-20948-v1.3.pdf) is connected using the following FPGA pins:

IMU device pin name	IMU device pin	board net name	connection type	FPGA pin	note
/CS	U5:22	IMU_SS0	direct connection	A12	PS_MIO67_502
SCL/SCLK	U5:23	IMU_SCLK	direct connection	B15	PS_MIO64_502
SDA/SDI	U5:24	IMU_MOSI	direct connection	A14	PS_MIO69_502
SDO/AD0	U5:9	IMU_MISO	direct connection	A13	PS_MIO68_502
FSYNC	U5:11	IMU_FSYNC	direct connection	B19	PS_MIO70_502
INT1	U5:12	IMU_INT	direct connection	A15	PS_MIO71_502

This peripheral is selected automatically when "Zynq UltraScale+ MPSoC" IP is added to a block design and the board preset is applied. It can then be accessed by PS software using SPI 0.

3.3 How can I interface with external cameras or displays?

Four identical connectors (J1, J3, J4, and J5) can be used for any combination of external cameras and displays. These use the MIPI (Mobile Industry Processor Interface) standard, CSI (camera serial interface) or DSI (display serial interface). The following four connection tables are for CSI with the data lanes as inputs from the camera. For DSI, the connections are similar except that the data lanes are outputs to the display.

For each connector, data lanes are matched in length to within 0.5mm.

SPI control to each connector is done via an I2C to SMBus switch (pca9546a.pdf)

3.3.1 j1: camera_a interface

MIPI net name	connector pin	board net name	connection type	FPGA pin	note
CSI_CLK0_P	J1:9	CAM_A_CK_P	direct connection	N6	
CSI_CLK0_N	J1:8	CAM_A_CK_N	direct connection	P6	
CSI_DATA0_P	J1:3	CAM_A_D0_P	direct connection	N4	
CSI_DATA0_N	J1:2	CAM_A_D0_N	direct connection	P4	
CSI_DATA1_P	J1:6	CAM_A_D1_P	direct connection	M7	
CSI_DATA1_N	J1:5	CAM_A_D1_N	direct connection	N7	
CSI_DATA2_P	J1:12	CAM_A_D2_P	direct connection	T6	
CSI_DATA2_N	J1:11	CAM_A_D2_N	direct connection	U5	
CSI_DATA3_P	J1:15	CAM_A_D3_P	direct connection	T4	
CSI_DATA3_N	J1:14	CAM_A_D3_N	direct connection	U4	
	J1:17	CAM_A_EN	direct connection	K15	Under control of PS side of Zynq
	J1:18	CAM_A_CLK_28	via resistive divider	D10	CAM_A_CLK
	J1:20	CAM_A_SCL	indirect connection via I2C mux U6	J11	CAM_SCL
	J1:21	CAM_A_SDA	indirect connection via I2C mux U6	M10	CAM_SCD

3.3.2 j3: camera_b interface

MIPI net name	connector pin	board net name	connection type	FPGA pin	note
CSI_CLK0_P	J3:9	CAM_B_CK_P	direct connection	T2	
CSI_CLK0_N	J3:8	CAM_B_CK_N	direct connection	U2	
CSI_DATA0_P	J3:3	CAM_B_D0_P	direct connection	N2	
CSI_DATA0_N	J3:2	CAM_B_D0_N	direct connection	N1	
CSI_DATA1_P	J3:6	CAM_B_D1_P	direct connection	R1	
CSI_DATA1_N	J3:5	CAM_B_D1_N	direct connection	T1	
CSI_DATA2_P	J3:12	CAM_B_D2_P	direct connection	V4	
CSI_DATA2_N	J3:11	CAM_B_D2_N	direct connection	V3	
CSI_DATA3_P	J3:15	CAM_B_D3_P	direct connection	K7	
CSI_DATA3_N	J3:14	CAM_B_D3_N	direct connection	K6	
	J3:17	CAM_B_EN	direct connection	M12	Under control of PS side of Zynq
	J3:18	CAM_B_CLK_28	via resistive divider	D9	CAM_B_CLK
	J3:20	CAM_B_SCL	indirect connection via I2C mux U6	J11	CAM_SCL
	J3:21	CAM_B_SDA	indirect connection via I2C mux U6	M10	CAM_SCD

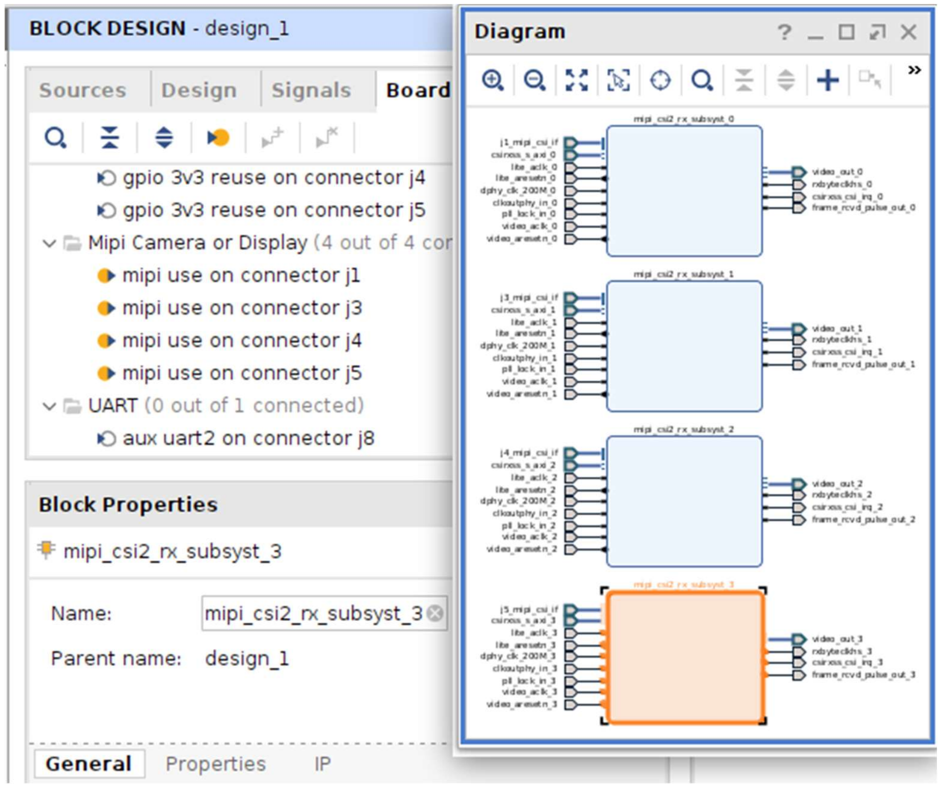
3.3.3 j4: camera_c interface

MIPI net name	connector pin	board net name	connection type	FPGA pin	note
CSI_CLK0_P	J4:9	CAM_C_CK_P	direct connection	K2	
CSI_CLK0_N	J4:8	CAM_C_CK_N	direct connection	L1	
CSI_DATA0_P	J4:3	CAM_C_D0_P	direct connection	L2	
CSI_DATA0_N	J4:2	CAM_C_D0_N	direct connection	M1	
CSI_DATA1_P	J4:6	CAM_C_D1_P	direct connection	G7	
CSI_DATA1_N	J4:5	CAM_C_D1_N	direct connection	H7	
CSI_DATA2_P	J4:12	CAM_C_D2_P	direct connection	J1	
CSI_DATA2_N	J4:11	CAM_C_D2_N	direct connection	K1	
CSI_DATA3_P	J4:15	CAM_C_D3_P	direct connection	H8	
CSI_DATA3_N	J4:14	CAM_C_D3_N	direct connection	J7	
	J4:17	CAM_C_EN	direct connection	N9	Under control of PS side of Zynq
	J4:18	CAM_C_CLK_28	via resistive divider	D13	CAM_C_CLK
	J4:20	CAM_C_SCL	indirect connection via I2C mux U6	J11	CAM_SCL
	J4:21	CAM_C_SDA	indirect connection via I2C mux U6	M10	CAM_SCD

3.3.4 j5: camera_d interface

MIPI net name	connector pin	board net name	connection type	FPGA pin	note
CSI_CLK0_P	J5:9	CAM_D_CK_P	direct connection	H2	
CSI_CLK0_N	J5:8	CAM_D_CK_N	direct connection	H1	
CSI_DATA0_P	J5:3	CAM_D_D0_P	direct connection	G2	
CSI_DATA0_N	J5:2	CAM_D_D0_N	direct connection	G1	
CSI_DATA1_P	J5:6	CAM_D_D1_P	direct connection	F7	
CSI_DATA1_N	J5:5	CAM_D_D1_N	direct connection	G6	
CSI_DATA2_P	J5:12	CAM_D_D2_P	direct connection	E1	
CSI_DATA2_N	J5:11	CAM_D_D2_N	direct connection	F1	
CSI_DATA3_P	J5:15	CAM_D_D3_P	direct connection	C1	
CSI_DATA3_N	J5:14	CAM_D_D3_N	direct connection	D1	
	J5:17	CAM_D_EN	direct connection	K12	Under control of PS side of Zynq
	J5:18	CAM_D_CLK_28	via resistive divider	C13	CAM_D_CLK
	J5:20	CAM_D_SCL	indirect connection via I2C mux U6	J11	CAM_SCL
	J5:21	CAM_D_SDA	indirect connection via I2C mux U6	M10	CAM_SCD

3.3.5 Using board file for mipi camera serial interface on j1, j3, j4, and j5

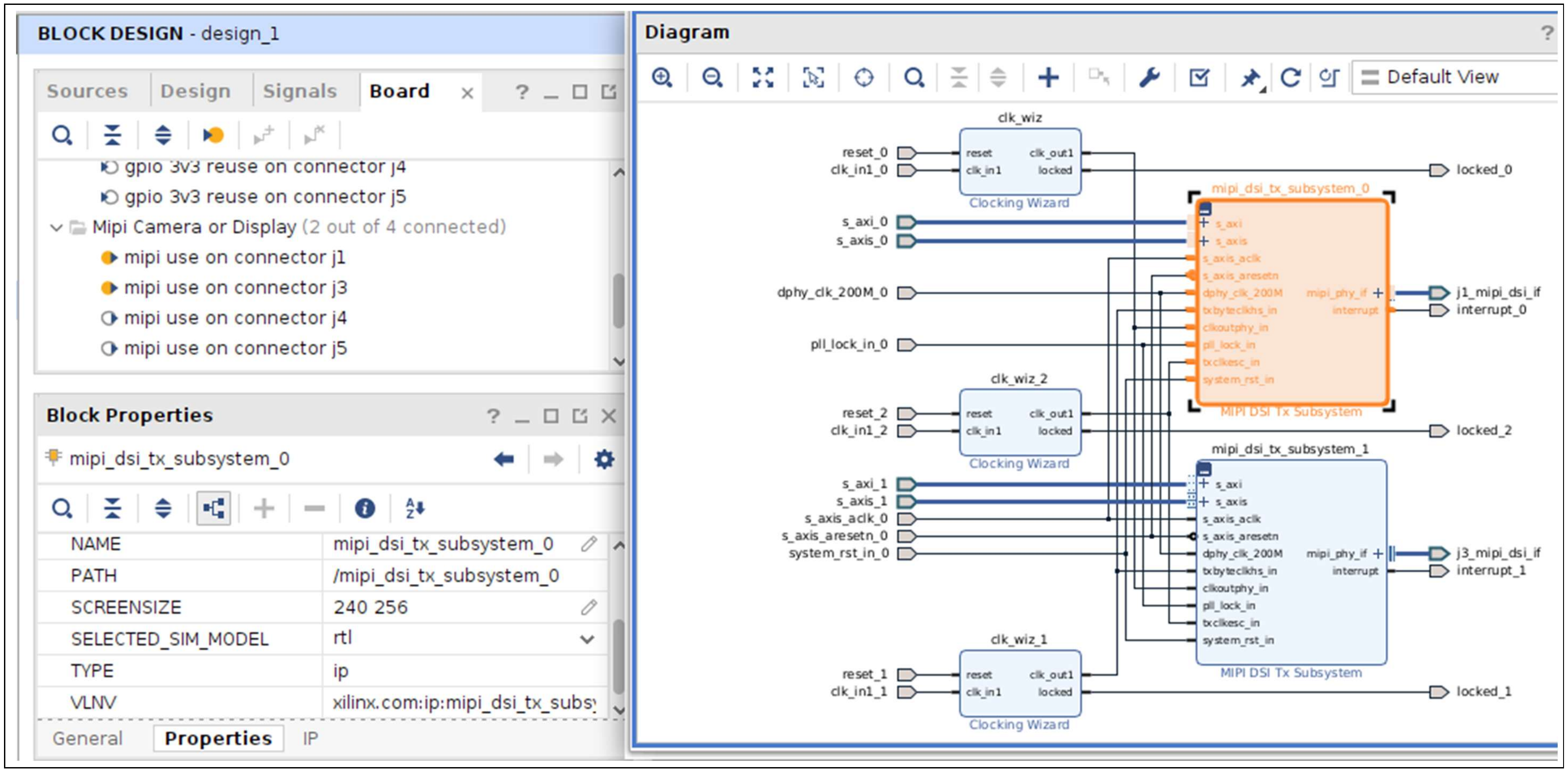


Name: mipi_csi2_rx_subsyst_3

Parent name: design_1

Figure 8 – board file mipi camera interfaces

3.3.6 Using board file for mipi display serial interface on j1 and j3



BLOCK DESIGN - design_1

Sources | Design | Signals | Board

- gpio 3v3 reuse on connector j4
- gpio 3v3 reuse on connector j5
- Mipi Camera or Display (2 out of 4 connected)
  - mipi use on connector j1
  - mipi use on connector j3
  - mipi use on connector j4
  - mipi use on connector j5

Block Properties

mipi_dsi_tx_subsystem_0

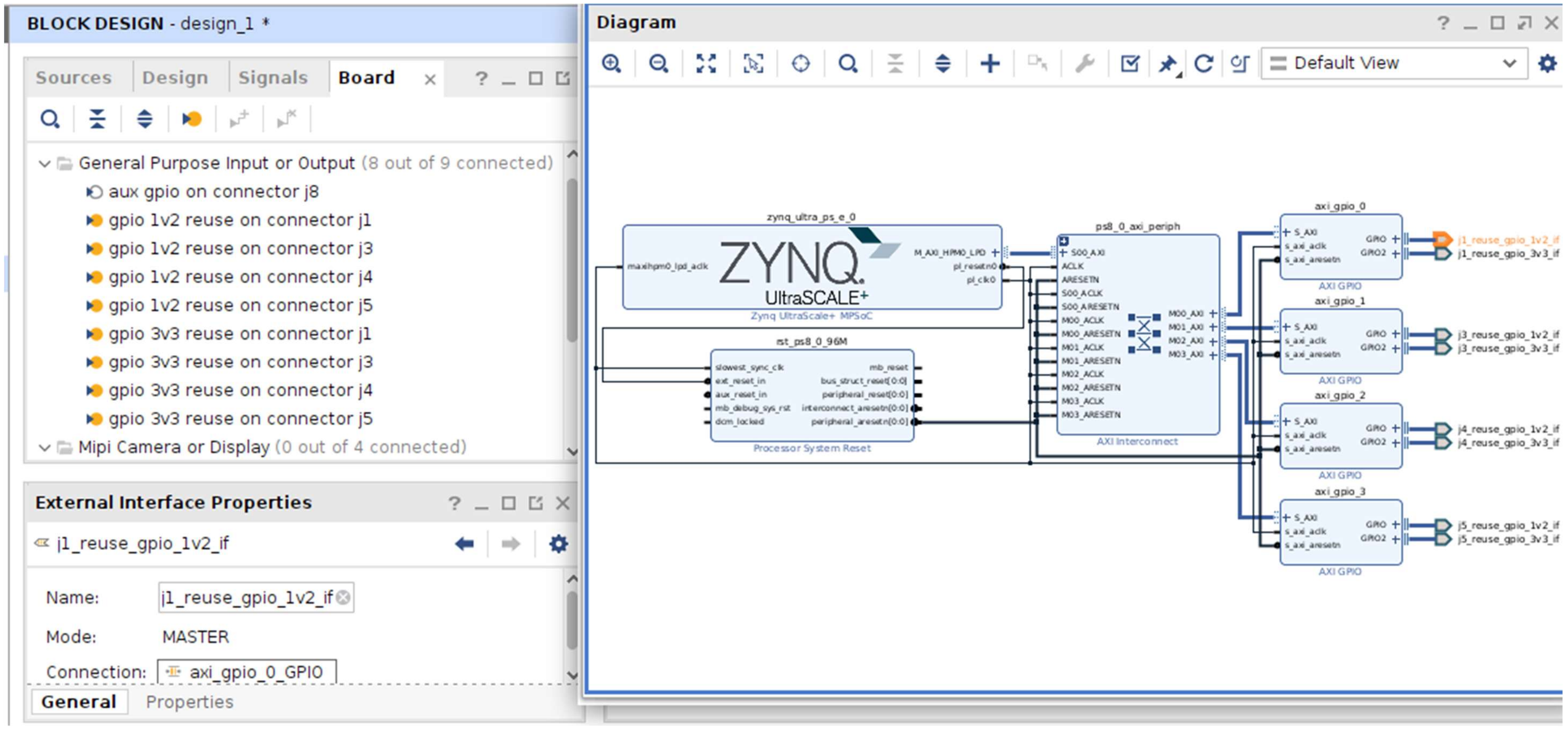
NAME	mipi_dsi_tx_subsystem_0
PATH	/mipi_dsi_tx_subsystem_0
SCREENSIZE	240 256
SELECTED_SIM_MODEL	rtl
TYPE	ip
VLNV	xilinx.com:ip:mipi_dsi_tx_subsy

General | Properties | IP

Figure 9 – board file mipi display interfaces

3.4 How can I repurpose the board connectors for GPIO?

Any of the four identical connectors (J1, J3, J4, and J5) can be repurposed for “General Purpose IO” using the Board tab. The image below shows a complete repurpose of all four. Note that one pin in each connector is at 3v3 logical standard and handled by the GPIO2 interface of each respective “AXI GPIO”.


BLOCK DESIGN - design_1 *

Sources | Design | Signals | Board

General Purpose Input or Output (8 out of 9 connected)
- aux gpio on connector j8
- gpio 1v2 reuse on connector j1
- gpio 1v2 reuse on connector j3
- gpio 1v2 reuse on connector j4
- gpio 1v2 reuse on connector j5
- gpio 3v3 reuse on connector j1
- gpio 3v3 reuse on connector j3
- gpio 3v3 reuse on connector j4
- gpio 3v3 reuse on connector j5

Mipi Camera or Display (0 out of 4 connected)

External Interface Properties

j1_reuse_gpio_1v2_if

Name: j1_reuse_gpio_1v2_if
Mode: MASTER
Connection: axi_gpio_0_GPIO

General | Properties

Figure 10 – board file repurposed gpio interfaces

3.5 What other interfaces are provided?

3.5.1 I2C for monitoring voltage supplies (U7, U13) etc

Device pin	board net name	connection type	FPGA pin	PS pin
U7:4 U13:4 U27:8 U21: G7 U11:9	SCL	direct connection	A18	PS_MIO74_502 Pulled up to 1V8 using 10k ohms
U7:1 U13:1 U27:7 U21: H8 U11:8	SDA	direct connection	A16	PS_MIO75_502 Pulled up to 1V8 using 10k ohms

This peripheral is selected automatically when “Zynq UltraScale+ MPSoC” IP is added to a block design and the board preset is applied. It may be accessed from PS software as I2C 0.

3.5.2 I2C to SPI camera/display control (U6)

Device pin	board net name	connection type	FPGA pin	PS pin
U6:14	CAM_SCL	direct connection	J11	PS_MIO28_501 Pulled up to 3V3 using 4k7 ohms
U6:15	CAM_SDA	direct connection	M10	PS_MIO29_501 Pulled up to 3V3 using 4k7 ohms

This peripheral is selected automatically when “Zynq UltraScale+ MPSoC” IP is added to a block design and the board preset is applied. It may be accessed from PS software as I2C 1.

3.5.3 J7: can bus

J7

connector pin	connection type	board net name	FPGA pin	PS pin
J7:4	indirect connection	CAN_RXD	K10	PS_MIO51_501
J7:5	indirect connection	CAN_TXD	J10	PS_MIO50_501
		CAN_SCLK	M13	PS_MIO44_501
		CAN_MOSI	M14	PS_MIO49_501
		CAN_MISO	N12	PS_MIO48_501
		CAN_SS0	N13	PS_MIO47_501

This peripheral is selected automatically when “Zynq UltraScale+ MPSoC” IP is added to a block design and the board preset is applied. It may be accessed from PS software as SPI 1.

3.5.4 J9: uart0

J9

connector pin	board net name	connection type	FPGA pin	PS pin
J9:2	UART0_RX	direct connection	K14	PS_MIO30_501
J9:3	UART0_TX	direct connection	J12	PS_MIO31_501

This peripheral is selected automatically when “Zynq UltraScale+ MPSoC” IP is added to a block design and the board preset is applied. It may be accessed from PS software as UART 0.

3.5.5 J7: uart1

J7

connector pin	board net name	connection type	FPGA pin	PS pin
J7:1	UART1_RX	direct connection	M9	PS_MIO32_501
J7:2	UART1_TX	direct connection	H14	PS_MIO33_501

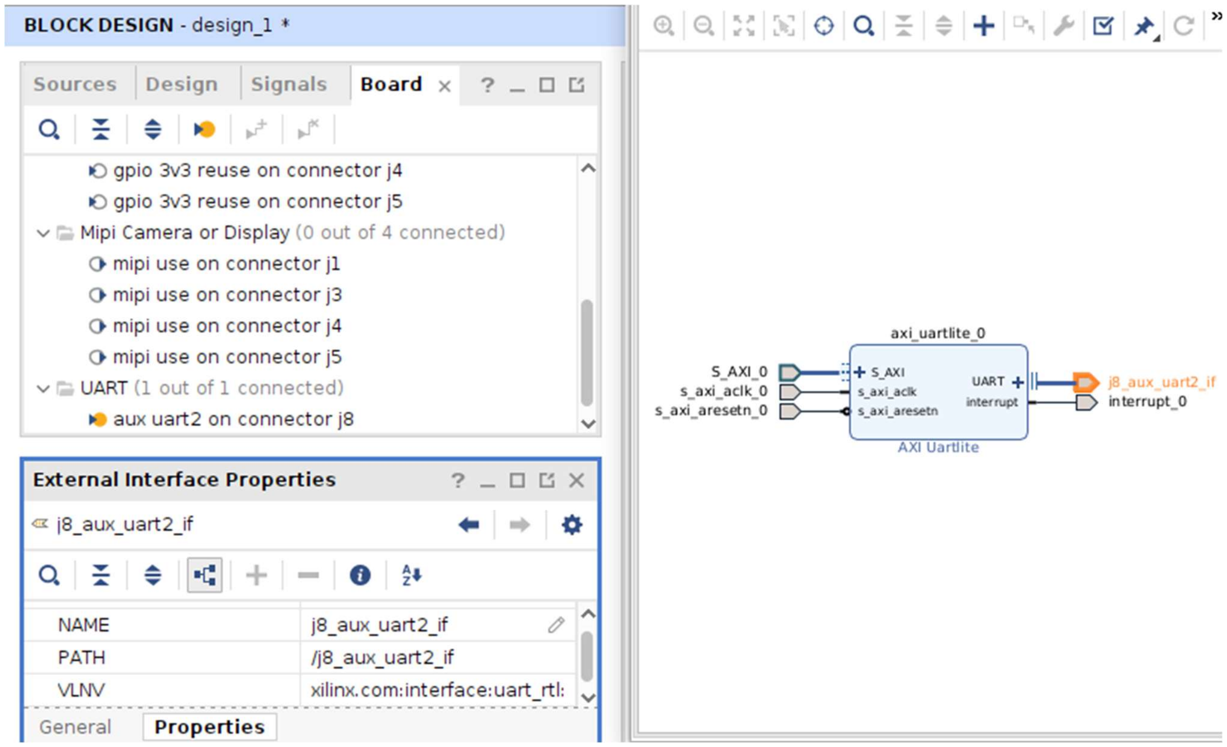
This peripheral is selected automatically when “Zynq UltraScale+ MPSoC” IP is added to a block design and the board preset is applied. It may be accessed from PS software as UART 1.

3.5.6 J8: uart2

J8

connector pin	board net name	connection type	FPGA pin	note
J8:2	AUX_UART2_RX	direct connection	B10	
J8:3	AUX_UART2_TX	direct connection	C10	

This peripheral is selected when the “aux uart2 on connector j8” is selected from the Board tab or when “AXI Uartlite” IP is added to a block design.



BLOCK DESIGN - design_1 *

Sources | Design | Signals | Board

- gpio 3v3 reuse on connector j4
- gpio 3v3 reuse on connector j5
- Mipi Camera or Display (0 out of 4 connected)
  - mipi use on connector j1
  - mipi use on connector j3
  - mipi use on connector j4
  - mipi use on connector j5
- UART (1 out of 1 connected)
  - aux uart2 on connector j8

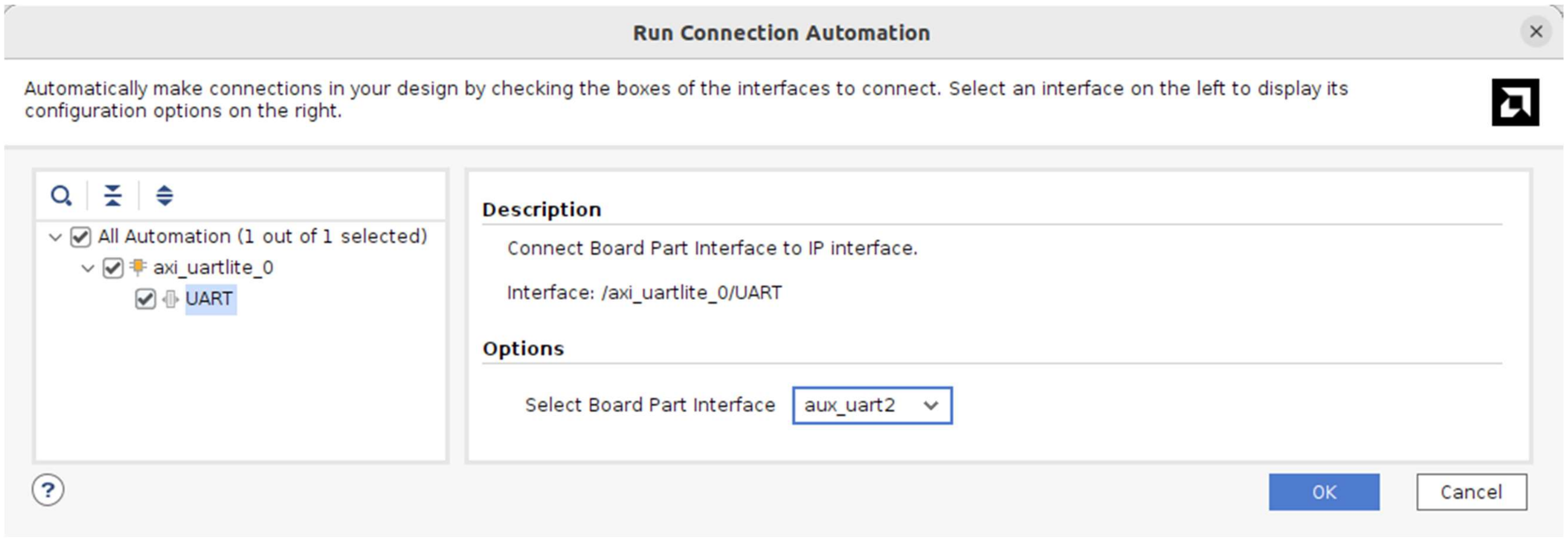
External Interface Properties

j8_aux_uart2_if

NAME	j8_aux_uart2_if
PATH	/j8_aux_uart2_if
VLNV	xilinx.com:interface:uart_rtl:

General | Properties

Figure 11 – board file aux uart2 interface


Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- [✓] All Automation (1 out of 1 selected)
  - [✓] axi_uartlite_0
    - [✓] UART

Description

Connect Board Part Interface to IP interface.

Interface: /axi_uartlite_0/UART

Options

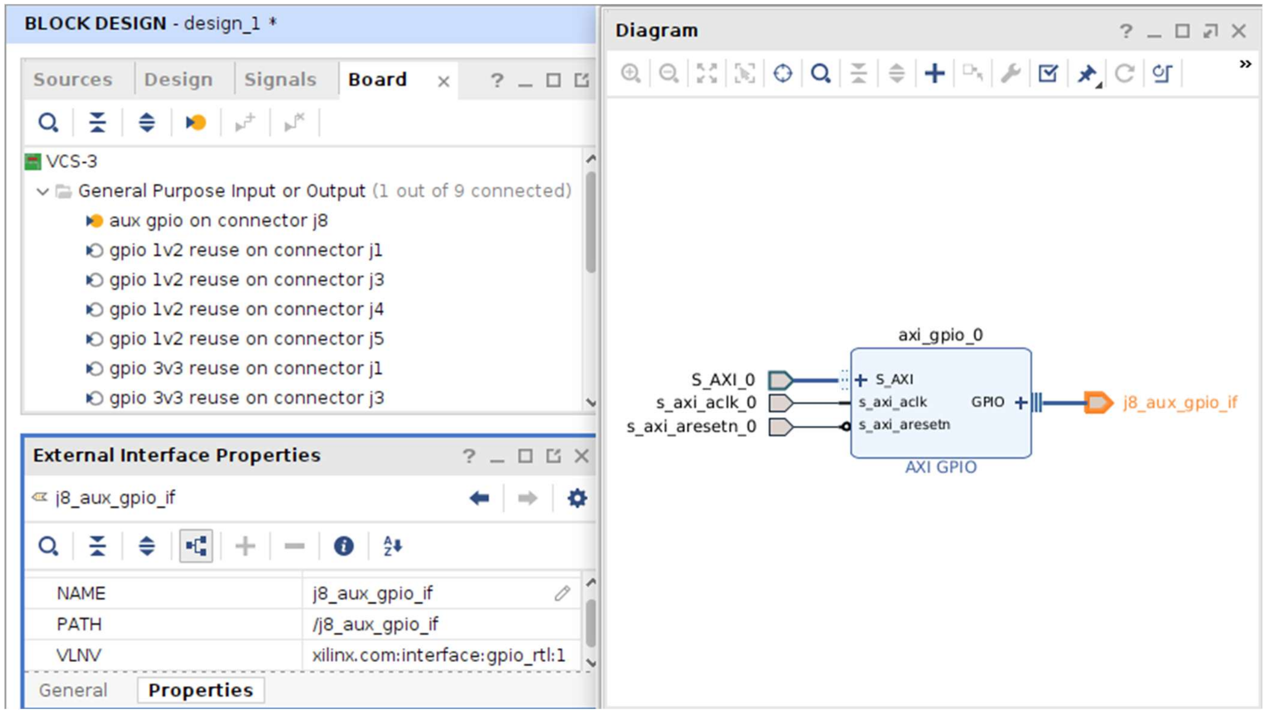
Select Board Part Interface: aux_uart2

Figure 12 – Connection Automation for aux uart2 interface

3.5.7 j8: gpio

connector pin		board net name	connection type	FPGA pin	note
J8:4		GPIO0	direct connection	B9	
J8:5		GPIO1	direct connection	A11	
J8:6		GPIO2	direct connection	A10	
J8:7		GPIO3	direct connection	A9	

This peripheral is selected when either the "aux gpio on connector j8" is selected from the Board tab or when "AXI GPIO" IP is added to a block design:


BLOCK DESIGN - design_1 *

Sources | Design | Signals | Board

VCS-3
General Purpose Input or Output (1 out of 9 connected)
- aux gpio on connector j8
- gpio 1v2 reuse on connector j1
- gpio 1v2 reuse on connector j3
- gpio 1v2 reuse on connector j4
- gpio 1v2 reuse on connector j5
- gpio 3v3 reuse on connector j1
- gpio 3v3 reuse on connector j3

External Interface Properties

j8_aux_gpio_if

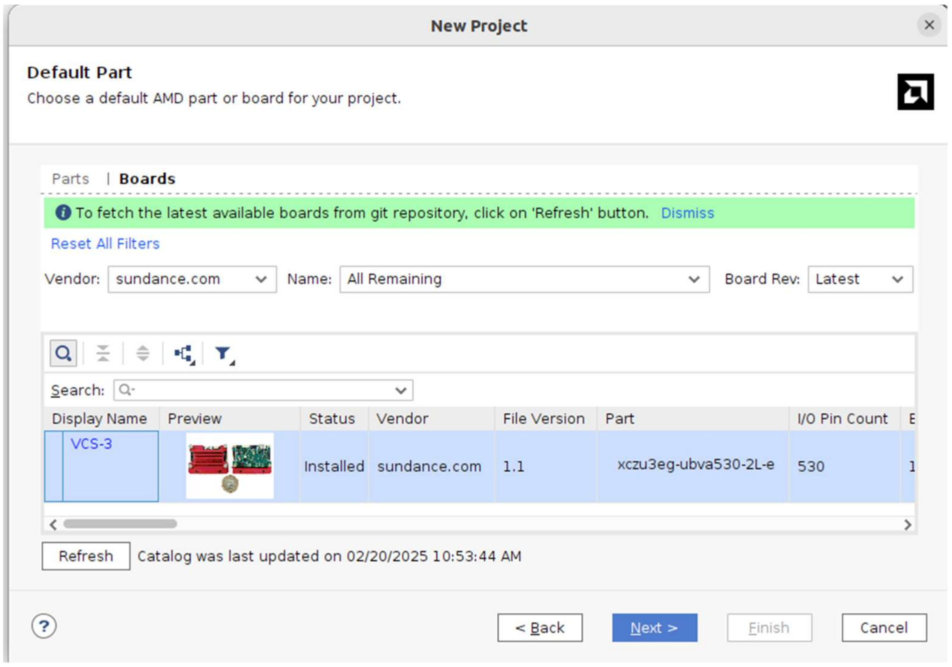
NAME	j8_aux_gpio_if
PATH	/j8_aux_gpio_if
VLNV	xilinx.com:interface:gpio_rtl:1

General | Properties

Figure 13 – board file aux gpio interface

3.6 How can I create a simple reference hardware design?

A simple reference design is created here using the board files described above. Open Vivado to create a new project and, when invited to choose a part or board, select the VCS-3.



Choose a default AMD part or board for your project.

To fetch the latest available boards from git repository, click on 'Refresh' button. Dismiss

Vendor: sundance.com Name: All Remaining Board Rev: Latest

Search:

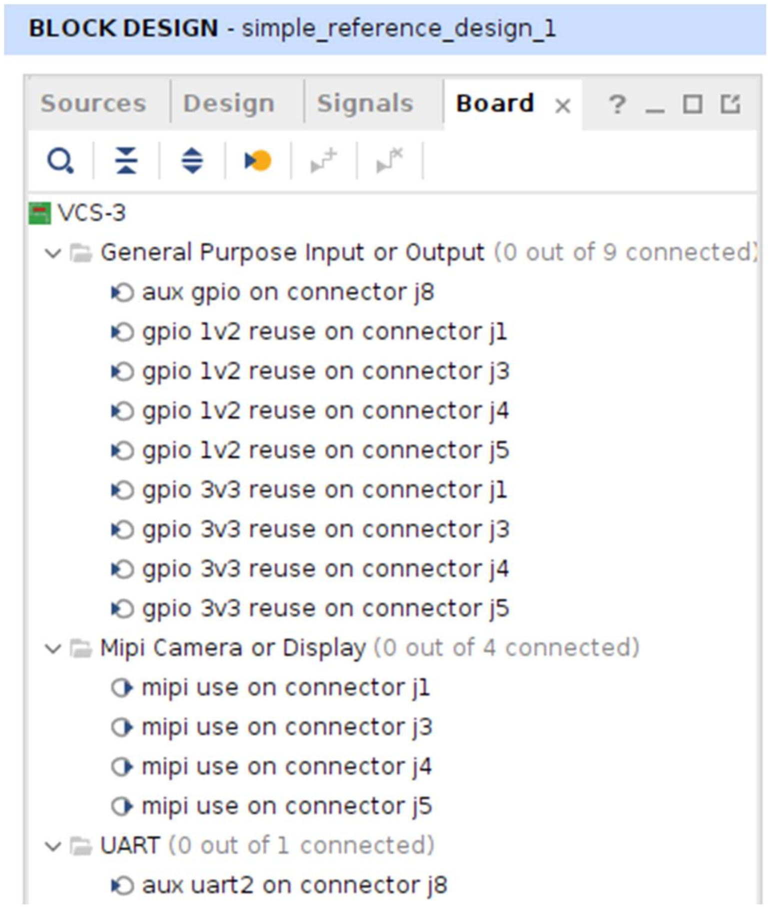
Display Name	Preview	Status	Vendor	File Version	Part	I/O Pin Count	E
VCS-3		Installed	sundance.com	1.1	xczu3eg-ubva530-2L-e	530	1

Catalog was last updated on 02/20/2025 10:53:44 AM

Figure 14 – Selecting VCS3 board file

Once the project is created click "Create Block Design" and give it a meaningful name.

Notice the Board tab in Block Design window is populated with a range of possible board interfaces.

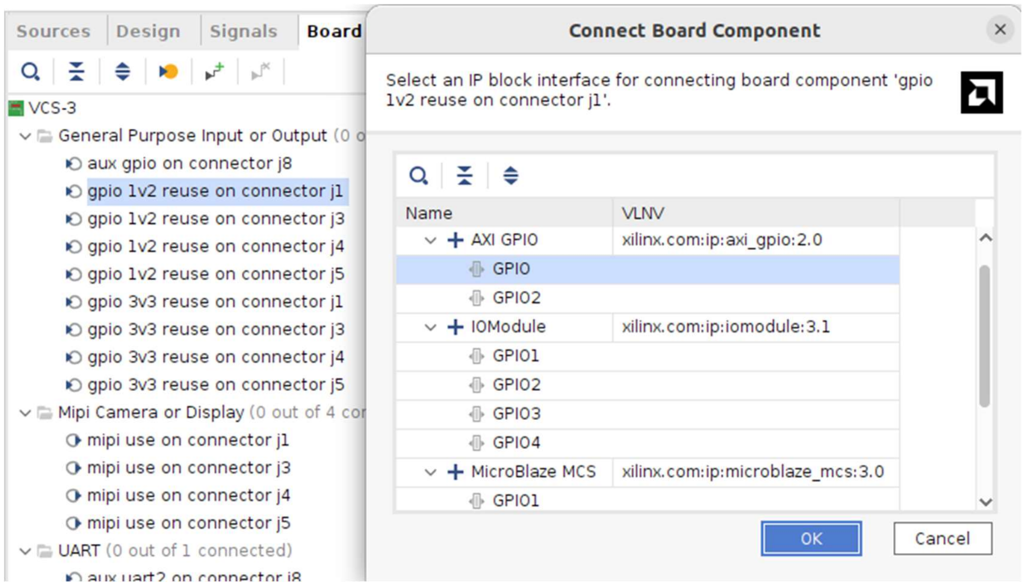

BLOCK DESIGN - simple_reference_design_1

VCS-3
- General Purpose Input or Output (0 out of 9 connected)
  - aux gpio on connector j8
  - gpio 1v2 reuse on connector j1
  - gpio 1v2 reuse on connector j3
  - gpio 1v2 reuse on connector j4
  - gpio 1v2 reuse on connector j5
  - gpio 3v3 reuse on connector j1
  - gpio 3v3 reuse on connector j3
  - gpio 3v3 reuse on connector j4
  - gpio 3v3 reuse on connector j5
- Mipi Camera or Display (0 out of 4 connected)
  - mipi use on connector j1
  - mipi use on connector j3
  - mipi use on connector j4
  - mipi use on connector j5
- UART (0 out of 1 connected)
  - aux uart2 on connector j8

Figure 15 – VCS3 board interfaces in IP Integrator

For this simple design we'll use all the board connectors for gpio and uarts.

Here j1 is selected and connected to an axi_gpio IP core, the 1v2 logic to the first IP port and 3v3 logic to the second IP port.



Sources | Design | Signals | **Board**

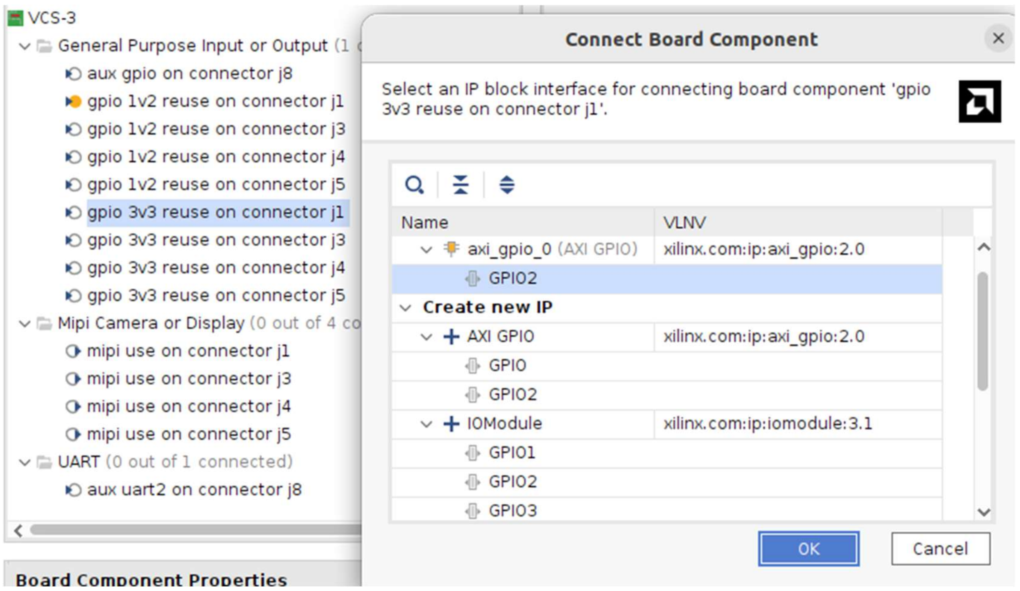
```
VCS-3
  General Purpose Input or Output (0 o
      aux gpio on connector j8
      gpio 1v2 reuse on connector j1
      gpio 1v2 reuse on connector j3
      gpio 1v2 reuse on connector j4
      gpio 1v2 reuse on connector j5
      gpio 3v3 reuse on connector j1
      gpio 3v3 reuse on connector j3
      gpio 3v3 reuse on connector j4
      gpio 3v3 reuse on connector j5
  Mipi Camera or Display (0 out of 4 cor
      mipi use on connector j1
      mipi use on connector j3
      mipi use on connector j4
      mipi use on connector j5
  UART (0 out of 1 connected)
      aux uart2 on connector j8
```

Connect Board Component

Select an IP block interface for connecting board component 'gpio 1v2 reuse on connector j1'.

Name	VLNV
AXI GPIO	xilinx.com:ip:axi_gpio:2.0
GPIO	
GPIO2	
IOModule	xilinx.com:ip:iomodule:3.1
GPIO1	
GPIO2	
GPIO3	
GPIO4	
MicroBlaze MCS	xilinx.com:ip:microblaze_mcs:3.0
GPIO1	

OK | Cancel



```
VCS-3
  General Purpose Input or Output (1 c
      aux gpio on connector j8
      gpio 1v2 reuse on connector j1
      gpio 1v2 reuse on connector j3
      gpio 1v2 reuse on connector j4
      gpio 1v2 reuse on connector j5
      gpio 3v3 reuse on connector j1
      gpio 3v3 reuse on connector j3
      gpio 3v3 reuse on connector j4
      gpio 3v3 reuse on connector j5
  Mipi Camera or Display (0 out of 4 co
      mipi use on connector j1
      mipi use on connector j3
      mipi use on connector j4
      mipi use on connector j5
  UART (0 out of 1 connected)
      aux uart2 on connector j8
```

Board Component Properties

Connect Board Component

Select an IP block interface for connecting board component 'gpio 3v3 reuse on connector j1'.

Name	VLNV
axi_gpio_0 (AXI GPIO)	xilinx.com:ip:axi_gpio:2.0
GPIO2	
Create new IP	
AXI GPIO	xilinx.com:ip:axi_gpio:2.0
GPIO	
GPIO2	
IOModule	xilinx.com:ip:iomodule:3.1
GPIO1	
GPIO2	
GPIO3	

OK | Cancel

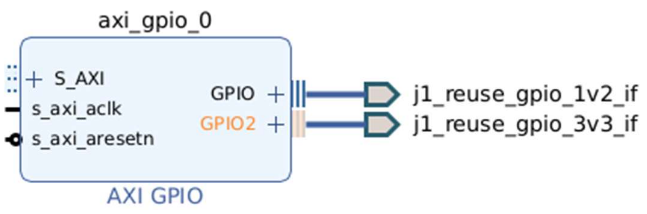


Figure 16 – Selecting axi_gpio for J1 board interface

Doing the same for all the other connectors …

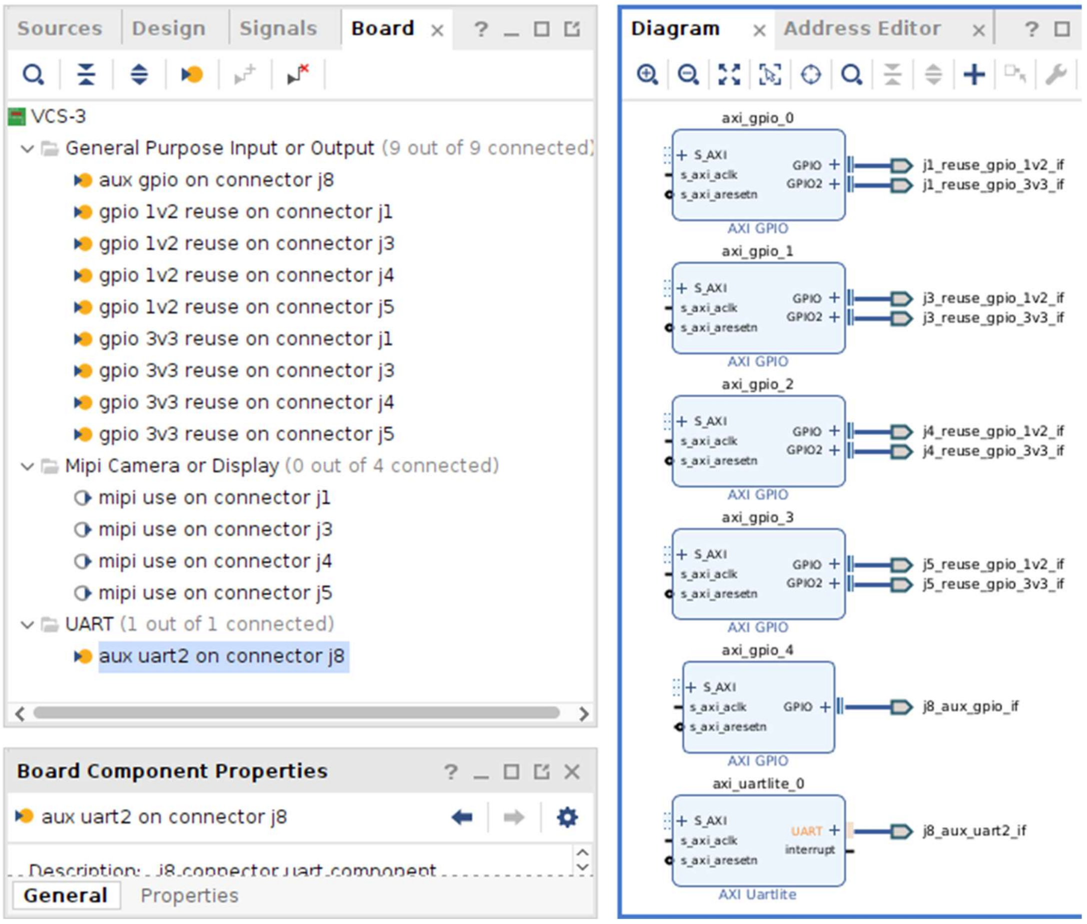
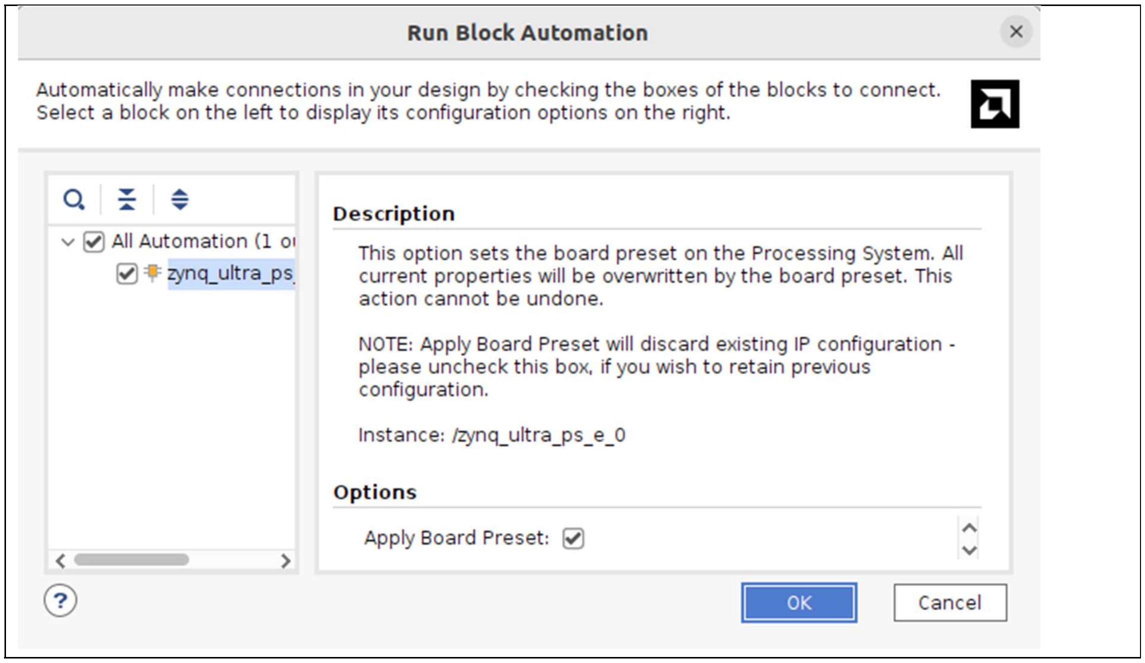


Figure 17 – Complete reused of all VCS3 board interfaces

After adding a Zynq UltraScale+ MPSoC core to the block design you are invited to use “Designer Assistance”: Run Block Automation and Run Connection Automation.

Block Automation configures the Zynq with board presets and ensures connectivity from the PS side pins to respective peripherals.



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

All Automation (1 o

zynq_ultra_ps

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /zynq_ultra_ps_e_0

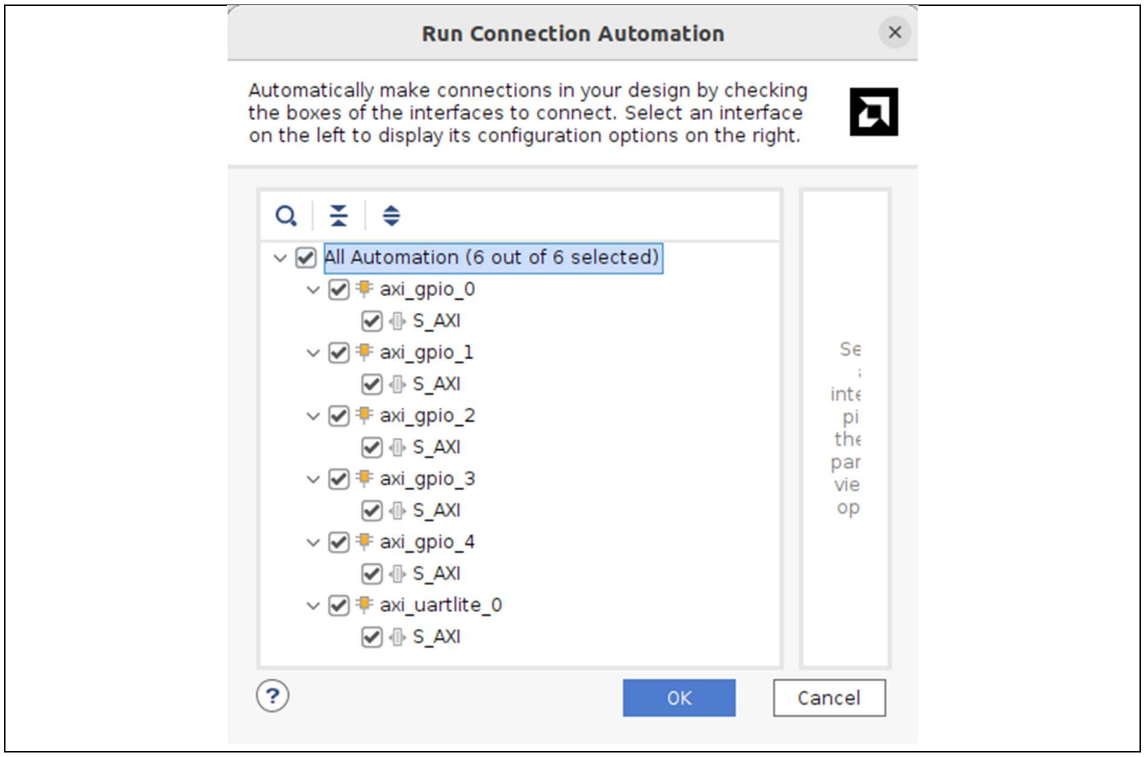
Options

Apply Board Preset: ✓

OK Cancel

Figure 18 – Running Block Automation for Zynq

Connection Automation wires up all the selected IP cores.



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

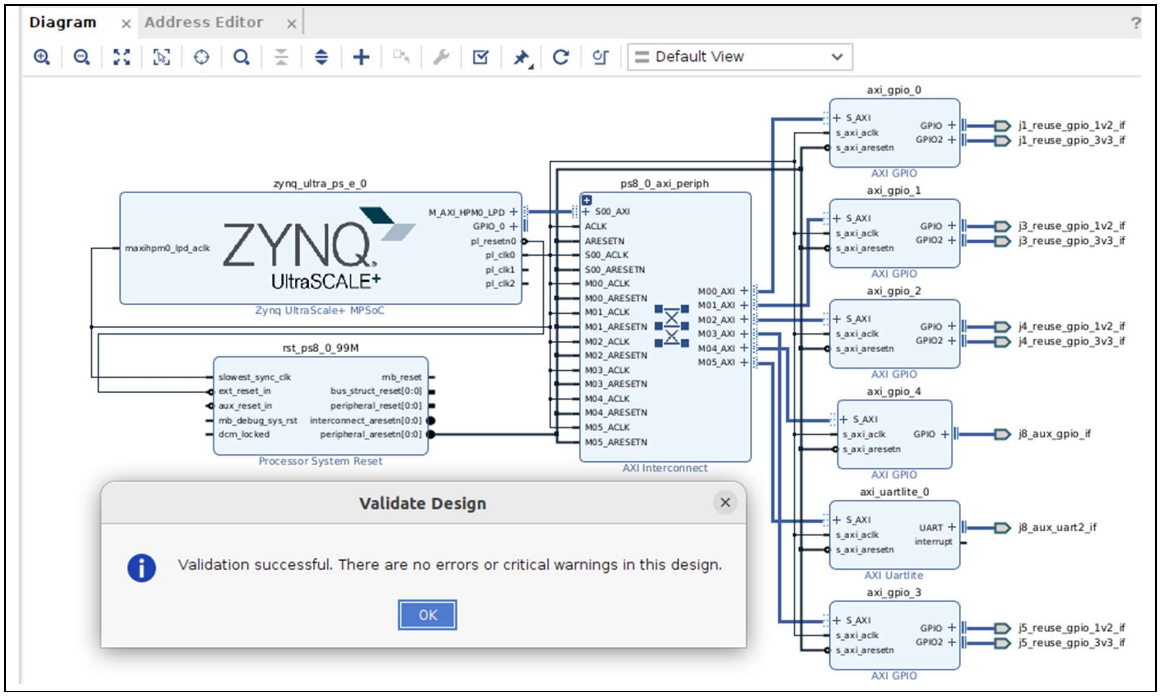
- ✓ All Automation (6 out of 6 selected)
  - ✓ axi_gpio_0
    - ✓ S_AXI
  - ✓ axi_gpio_1
    - ✓ S_AXI
  - ✓ axi_gpio_2
    - ✓ S_AXI
  - ✓ axi_gpio_3
    - ✓ S_AXI
  - ✓ axi_gpio_4
    - ✓ S_AXI
  - ✓ axi_uartlite_0
    - ✓ S_AXI

Se inte pi the par vie op

OK Cancel

Figure 19 – Running Connection Automation

The resulting block design can be validated using menu Tools→Validate Design



Validation successful. There are no errors or critical warnings in this design.

Figure 20 – Validating block design

The block design can now be closed and saved. When it appears in the Project Manager Sources window Hierarchy, you can right click it to select “Create HDL Wrapper”.

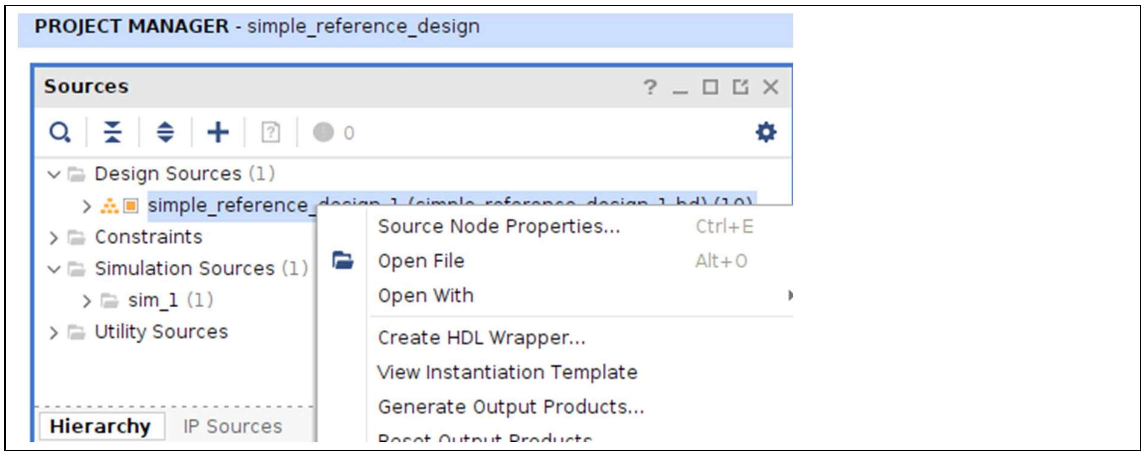
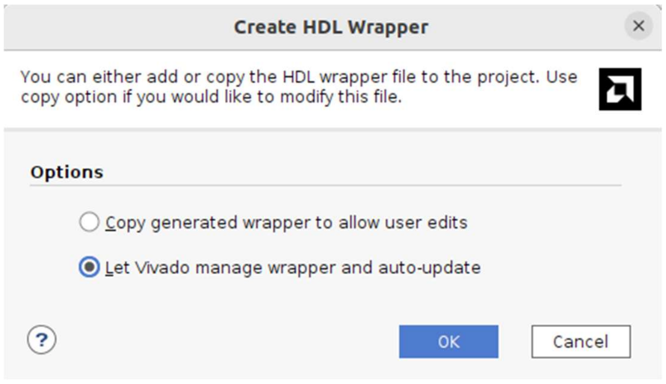

PROJECT MANAGER - simple_reference_design
Sources
Design Sources (1)
simple_reference_design_1 (simple_reference_design_1.bd) (10)
Constraints
Simulation Sources (1)
sim_1 (1)
Utility Sources
Hierarchy
IP Sources
Source Node Properties... Ctrl+E
Open File Alt+O
Open With
Create HDL Wrapper...
View Instantiation Template
Generate Output Products...
Reset Output Products

Figure 21 – Creating block design top level HDL wrapper

Choosing to let Vivado manage the wrapper is usually the better option unless you want to add other HDL entities (VHDL) or modules (Verilog) to the top level.



Create HDL Wrapper

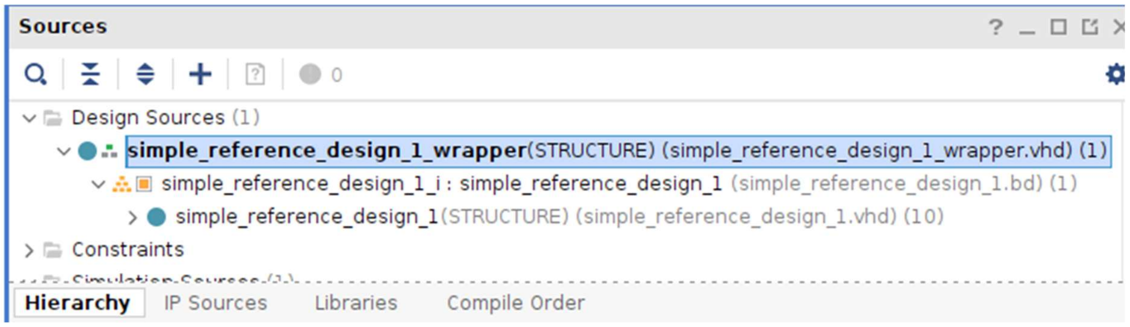
You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

Options

( ) Copy generated wrapper to allow user edits

(●) Let Vivado manage wrapper and auto-update

Figure 22 – Letting Vivado manage top level HDL wrapper



Sources

Design Sources (1)
simple_reference_design_1_wrapper(STRUCTURE) (simple_reference_design_1_wrapper.vhd) (1)
simple_reference_design_1_i : simple_reference_design_1 (simple_reference_design_1.bd) (1)
simple_reference_design_1(STRUCTURE) (simple_reference_design_1.vhd) (10)
Constraints

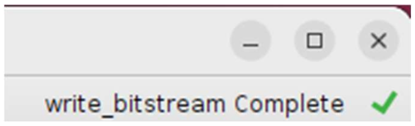
Figure 23 – Top level wrapper appearing in sources hierarchy

The generated wrapper here is in VHDL but it could be in Verilog if the project default HDL had been set to that.

To generate the complete hardware project just select “Generate Bitstream” in the Project Manager window and choose Yes or OK to any popup dialog boxes. This process will:

- Generate output products for each IP core in the block diagram
- Run synthesis for each of them and also the top level HDL wrapper
- Run Implementation (place and route)
- Produce a bitstream that can be downloaded to the VCS3

The whole process may take some time … relax! You’ll know when it’s finished when you see this in the top right corner of Vivado:

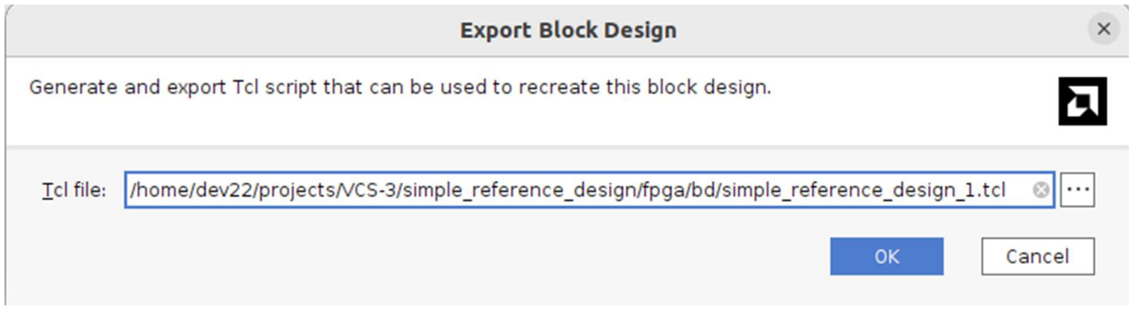


write_bitstream Complete ✓

Figure 24 – Bitstream complete

3.7 How can I archive and reproduce my simple reference design?

Vivado makes it easy for you to archive your block design as a tcl scripts (tcl stands for Tool Command Language). You just need to create the script using menu File→Export→Export block Design.

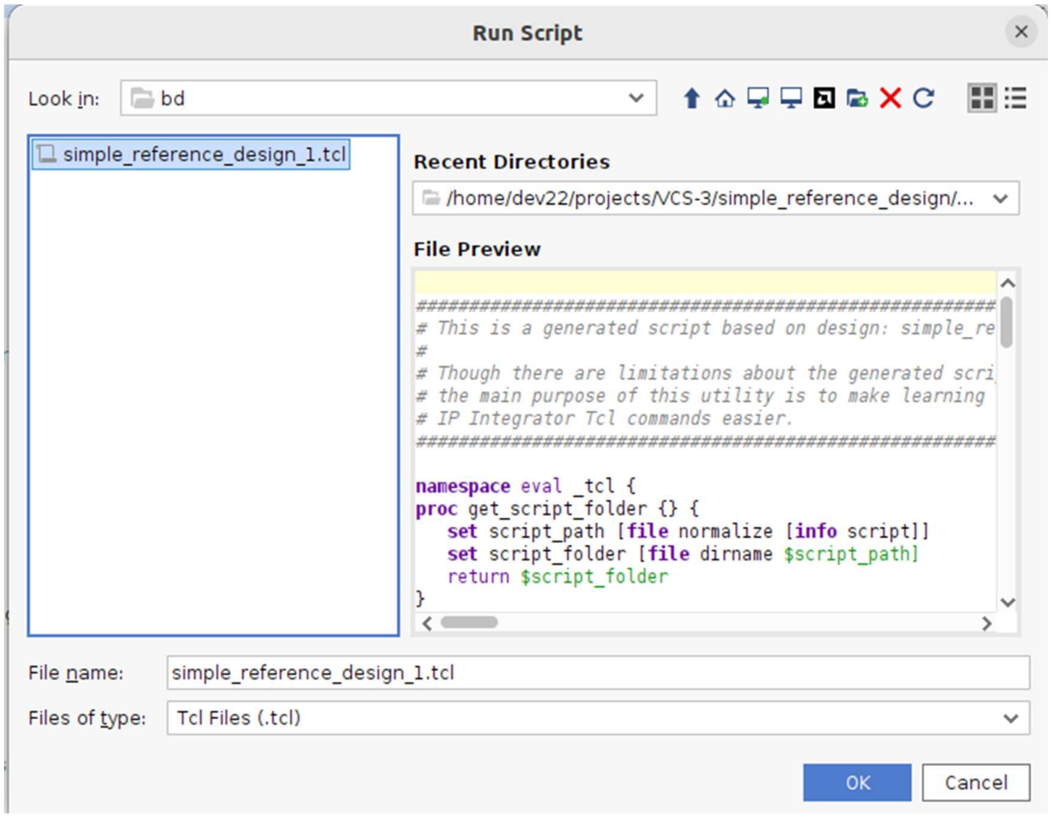


Generate and export Tcl script that can be used to recreate this block design.

Tcl file: /home/dev22/projects/VCS-3/simple_reference_design/fpga/bd/simple_reference_design_1.tcl

Figure 25 – Exporting block design as a tcl script

To recreate the block design, just open a new project for VCCS-3 and run the script.



Look in: bd

simple_reference_design_1.tcl

/home/dev22/projects/VCS-3/simple_reference_design/...

```
##########################################################
# This is a generated script based on design: simple_re
#
# Though there are limitations about the generated scri
# the main purpose of this utility is to make learning
# IP Integrator Tcl commands easier.
##########################################################

namespace eval _tcl {
proc get_script_folder {} {
   set script_path [file normalize [info script]]
   set script_folder [file dirname $script_path]
   return $script_folder
}
```

File name: simple_reference_design_1.tcl

Files of type: Tcl Files (.tcl)

Figure 26 – Recreating block design using tcl script

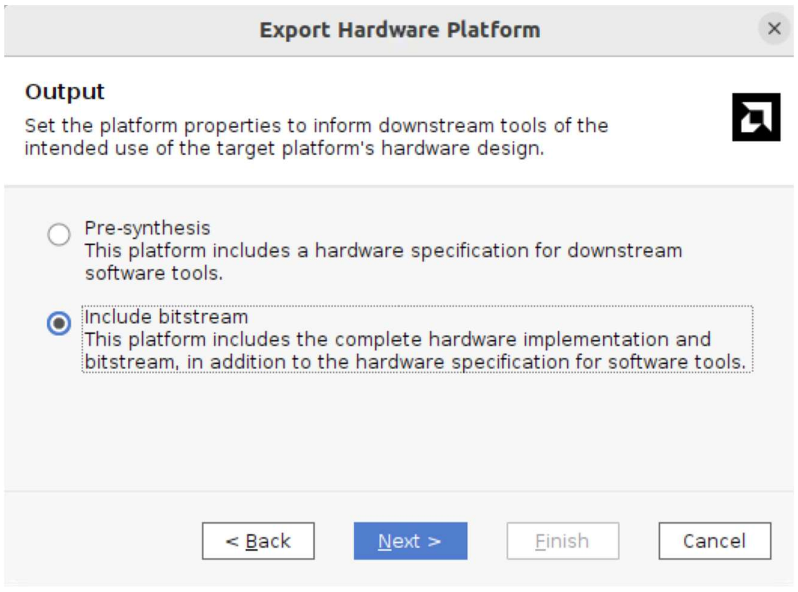
You can then recreate the top level wrapper and generate the bitstream as before. The above tcl script is saved in the VCS-3 archive for you to try.

4 Software

Before any software can be written for the VCS3 processor side (PS) the hardware developed previously must be exported from Vivado and imported into Vitis.

4.1 Creating Vitis platform component from Vivado hardware

To export the hardware use menu File→Export→Export Hardware



Export Hardware Platform

Output

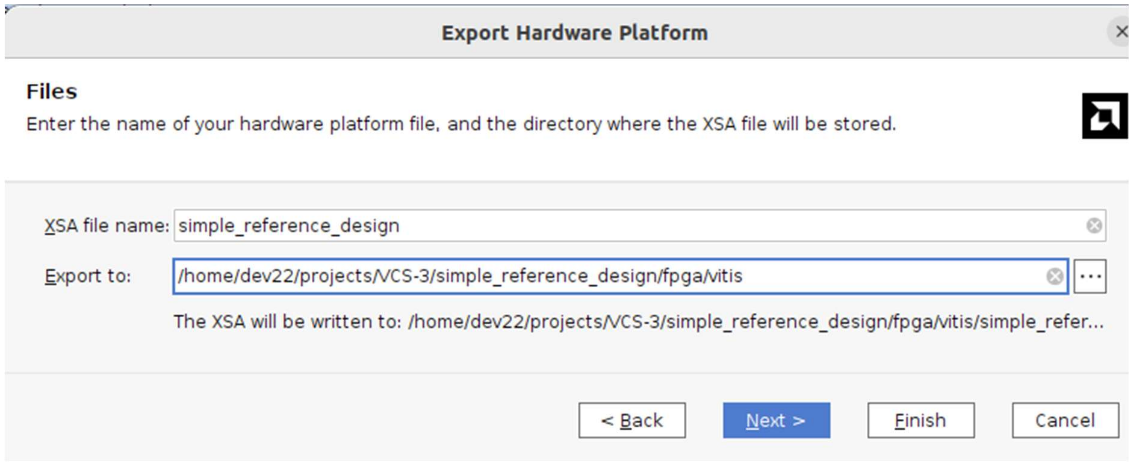
Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

( ) Pre-synthesis
This platform includes a hardware specification for downstream software tools.

(●) Include bitstream
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.

Figure 27 - Exporting hardware including the bitstream

Continuing with the above simple reference design …



Export Hardware Platform

Files

Enter the name of your hardware platform file, and the directory where the XSA file will be stored.

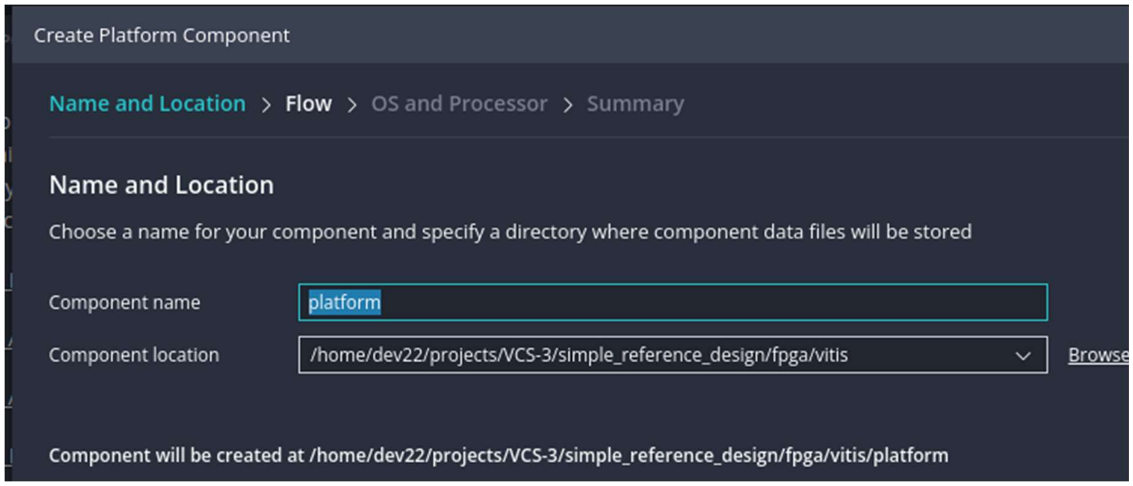
XSA file name: simple_reference_design

Export to: /home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis

The XSA will be written to: /home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/simple_refer...

Figure 28 – Exporting hardware to a directory where Vitis will use it

When starting Vitis you must choose a working directory. Then to import the above hardware .xsa file use menu File→New Component→Platform and follow the sequence of dialogs.


Create Platform Component

Name and Location > Flow > OS and Processor > Summary

Name and Location

Choose a name for your component and specify a directory where component data files will be stored

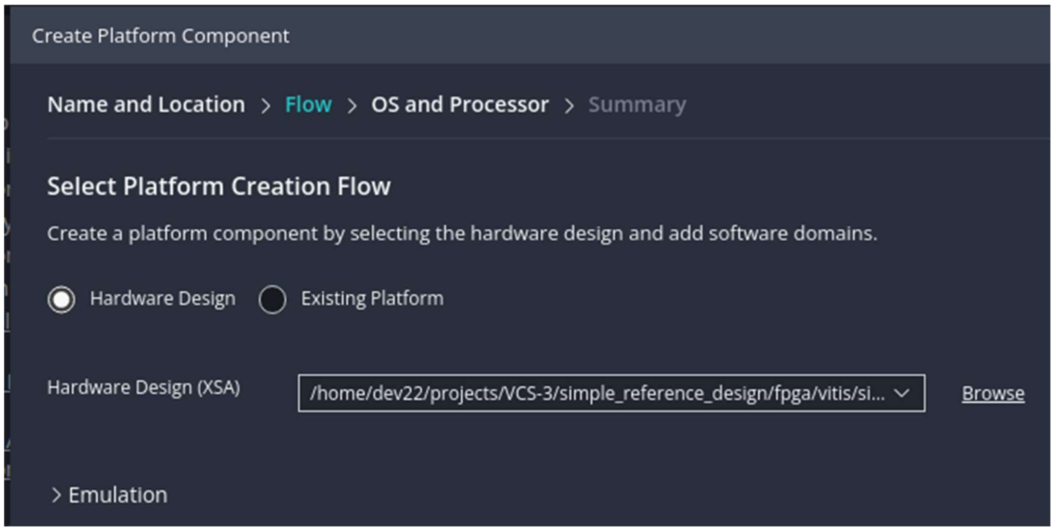
Component name: platform

Component location: /home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis

Browse

Component will be created at /home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/platform

Figure 29 – Creating platform in Vitis – naming platform component


Create Platform Component

Name and Location > Flow > OS and Processor > Summary

Select Platform Creation Flow

Create a platform component by selecting the hardware design and add software domains.

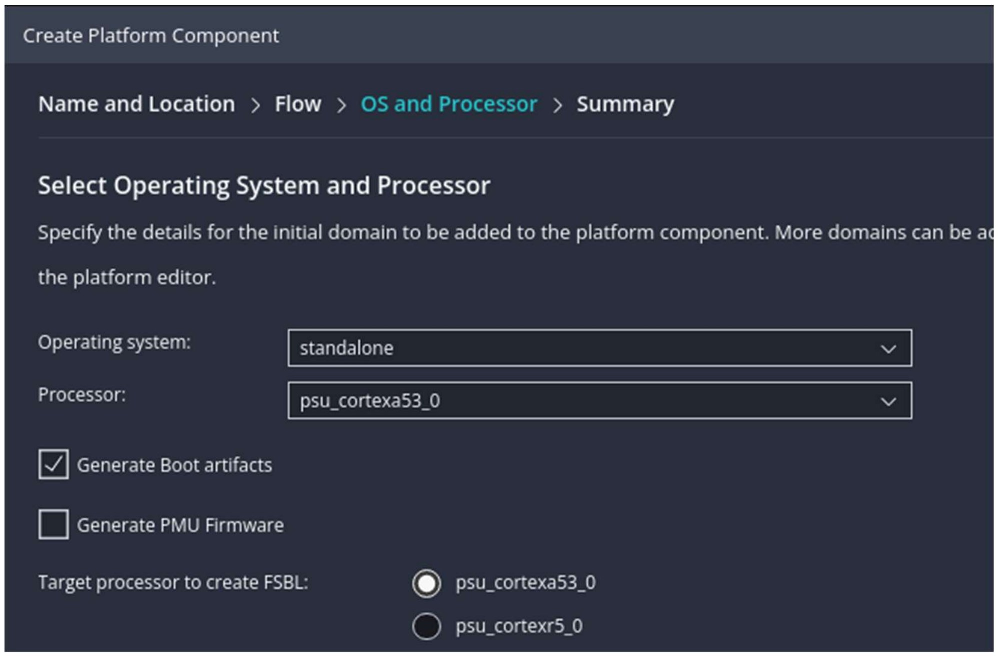
[✓ Hardware Design] Existing Platform

Hardware Design (XSA): /home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/si...

Browse

> Emulation

Figure 30 – Creating platform in Vitis – finding .xsa file


Create Platform Component

Name and Location > Flow > OS and Processor > Summary

Select Operating System and Processor

Specify the details for the initial domain to be added to the platform component. More domains can be ac
the platform editor.

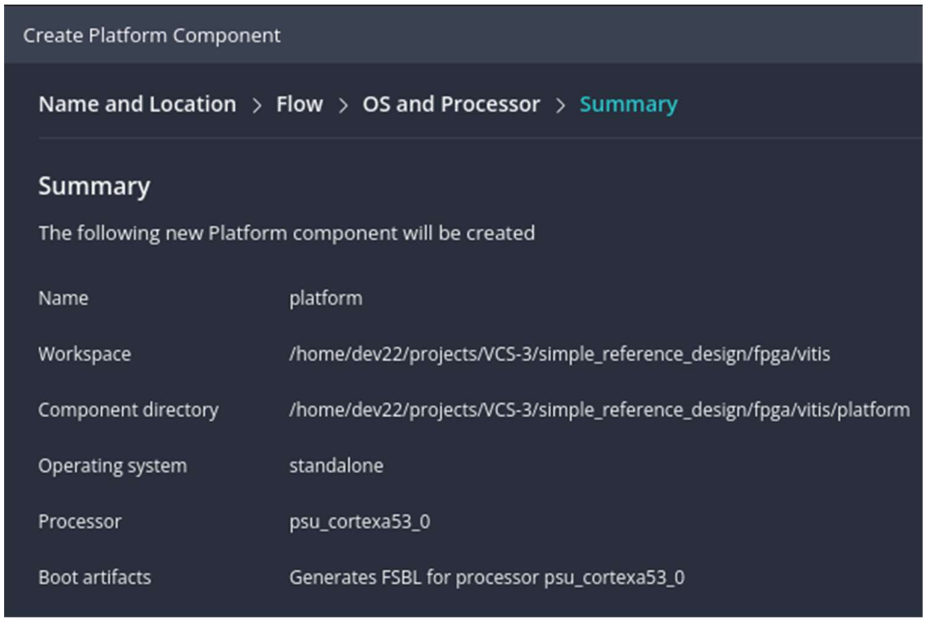
Operating system:	standalone
Processor:	psu_cortexa53_0

☑ Generate Boot artifacts

☐ Generate PMU Firmware

Target processor to create FSBL: (● psu_cortexa53_0) ○ psu_cortexr5_0

Figure 31 – Creating platform in Vitis – selecting processor and operating system


Create Platform Component

Name and Location > Flow > OS and Processor > Summary

Summary

The following new Platform component will be created

Name	platform
Workspace	/home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis
Component directory	/home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/platform
Operating system	standalone
Processor	psu_cortexa53_0
Boot artifacts	Generates FSBL for processor psu_cortexa53_0

Figure 32 – Creating platform in Vitis – summary

This platform component can now be built by clicking the Build icon

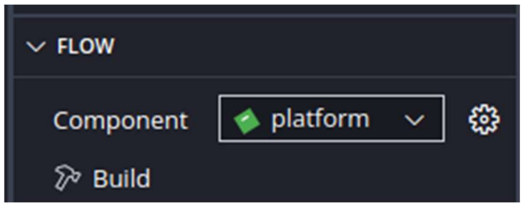


Figure 33 – Building platform in Vitis

4.2 How can I create a “hello world” app?

There are a few options to creating a new software application. One of the easiest is to copy an example application and elaborate it to your own specification. To do this click on the examples icon and select one of the many preprepared applications e.g. “Hello World” and following the sequence of dialogs.

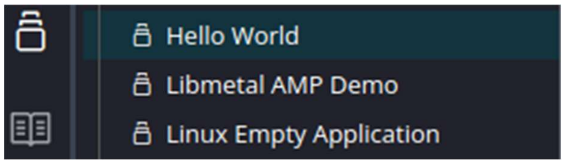
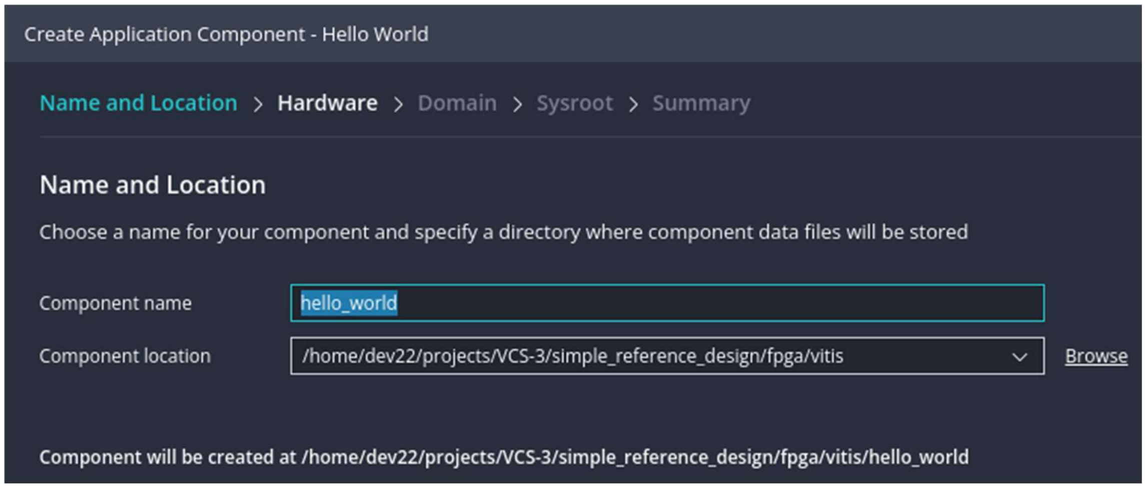


Figure 34 – Selecting example application



Create Application Component - Hello World

Name and Location > Hardware > Domain > Sysroot > Summary

Name and Location

Choose a name for your component and specify a directory where component data files will be stored

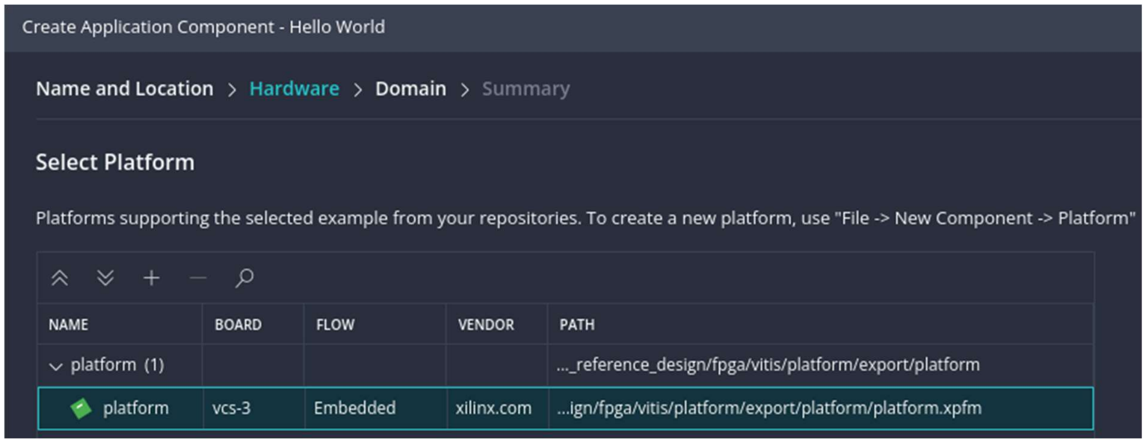
Component name: hello_world

Component location: /home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis

Browse

Component will be created at /home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/hello_world

Figure 35 – Creating an application in Vitis – choosing application component name



Create Application Component - Hello World

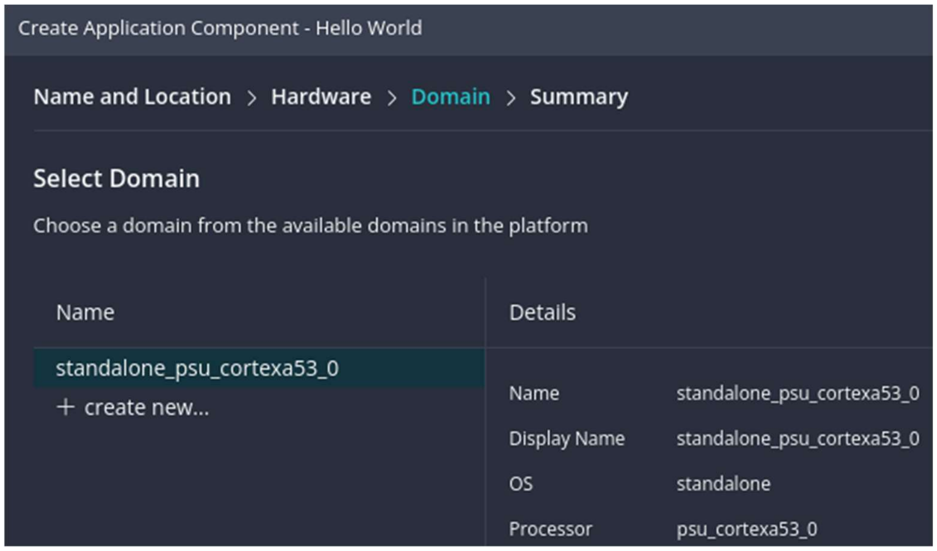
Name and Location > Hardware > Domain > Summary

Select Platform

Platforms supporting the selected example from your repositories. To create a new platform, use "File -> New Component -> Platform"

NAME	BOARD	FLOW	VENDOR	PATH
platform (1)				..._reference_design/fpga/vitis/platform/export/platform
platform	vcs-3	Embedded	xilinx.com	...ign/fpga/vitis/platform/export/platform/platform.xpfm

Figure 36 – Creating an application in Vitis – associating hardware platform



Create Application Component - Hello World

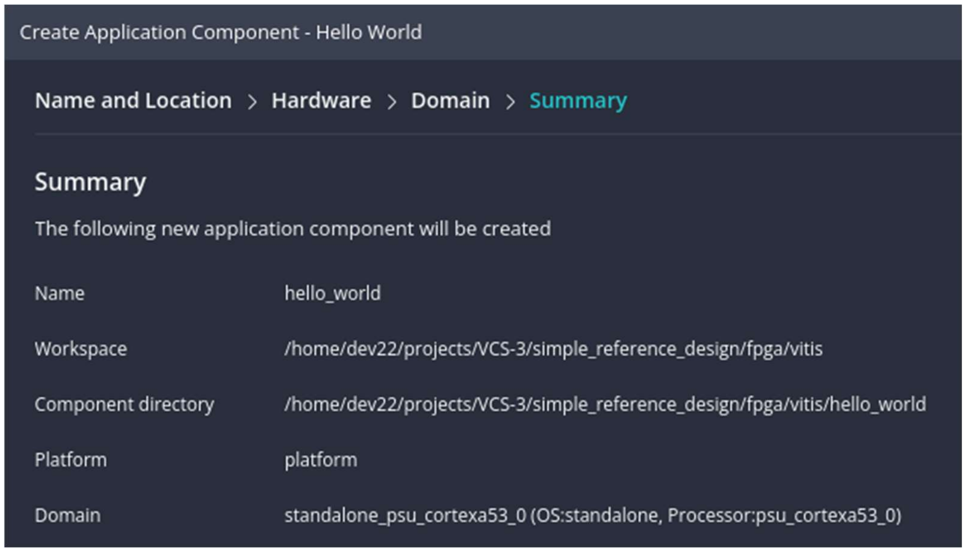
Name and Location > Hardware > Domain > Summary

Select Domain

Choose a domain from the available domains in the platform

Name	Details	
standalone_psu_cortexa53_0	Name	standalone_psu_cortexa53_0
+ create new...	Display Name	standalone_psu_cortexa53_0
	OS	standalone
	Processor	psu_cortexa53_0

Figure 37 – Creating an application in Vitis – choosing processor domain



Create Application Component - Hello World

Name and Location > Hardware > Domain > Summary

Summary

The following new application component will be created

Name	hello_world
Workspace	/home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis
Component directory	/home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/hello_world
Platform	platform
Domain	standalone_psu_cortexa53_0 (OS:standalone, Processor:psu_cortexa53_0)

Figure 38 – Creating an application in Vitis – summary

This application component can now be built by clicking the Build icon

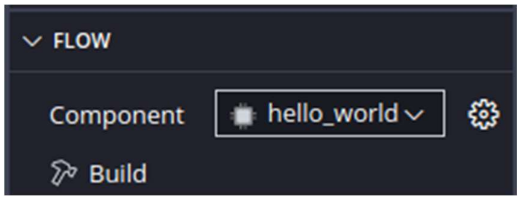
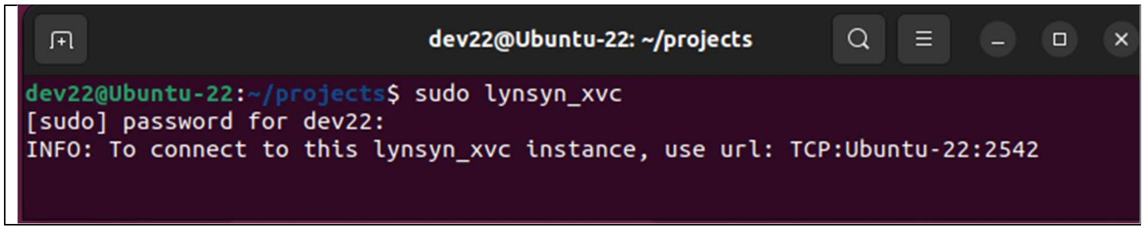


Figure 39 – Building application in Vitis

4.3 Running "hello world" app on hardware?

The built application can now be downloaded and run on the VCS3 hardware depending on your JTAG connection. If you have the VCS3 development kit (as described in "VCS3 Development Kit Getting Started Guide") then the easiest way is to invoke the Lynsyn Xilinx Virtual Cable driver as shown here.

NOTE: If you are not using the VCS3 Development Kit (i.e. not using the Lynsyn), skip to debug session.



```
dev22@Ubuntu-22:~/projects$ sudo lynsyn_xvc
[sudo] password for dev22:
INFO: To connect to this lynsyn_xvc instance, use url: TCP:Ubuntu-22:2542
```

Figure 40 – Running Lynsyn Xilinx Virtual Cable

If running Vitis from a VM don't forget to claim the required USB devices from the host machine..

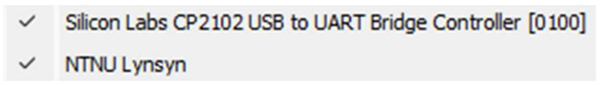
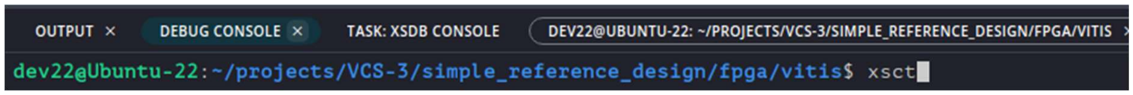
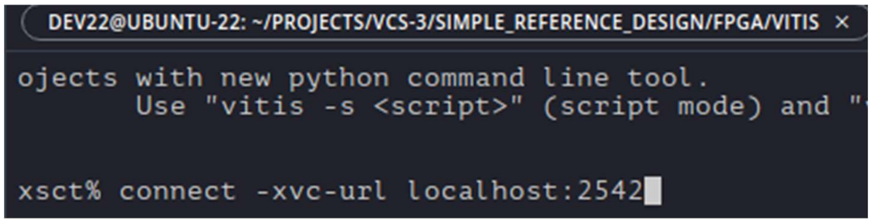


Figure 41 – VM taking control of required host USB device

So that Vitis can use the Lynsyn, open a terminal using menu Terminal→New Terminal and invoke xsct and run command line: connect -xvc-url localhost:2542


OUTPUT
DEBUG CONSOLE
TASK: XSDB CONSOLE
DEV22@UBUNTU-22: ~/PROJECTS/VCS-3/SIMPLE_REFERENCE_DESIGN/FPGA/VITIS

```
dev22@Ubuntu-22:~/projects/VCS-3/simple_reference_design/fpga/vitis$ xsct
```


DEV22@UBUNTU-22: ~/PROJECTS/VCS-3/SIMPLE_REFERENCE_DESIGN/FPGA/VITIS

```
ojects with new python command line tool.
        Use "vitis -s <script>" (script mode) and "

xsct% connect -xvc-url localhost:2542
```

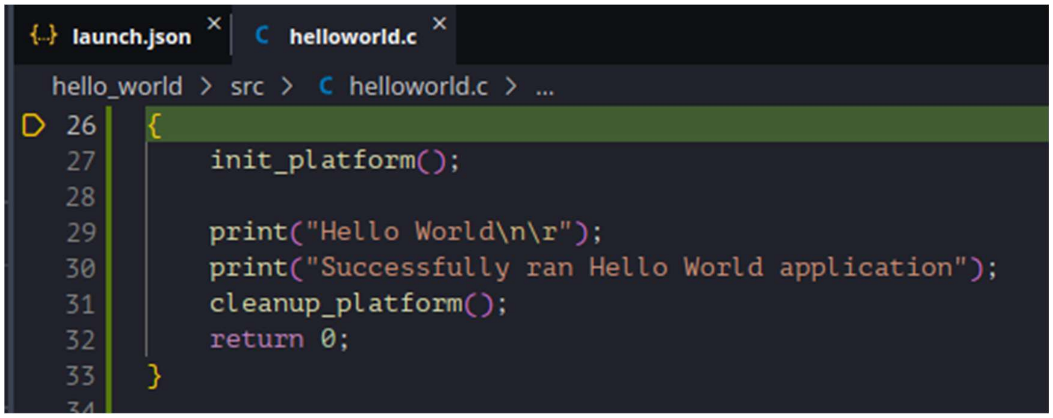
Figure 42 – Connecting Vitis to Lynsyn Xilinx Virtual Cable

The debug session can now be started …

```
15:36:04 INFO  : XSDB server has started successfully from frontend.
15:36:05 INFO  : Connection to XSDB Server established.
15:36:05 INFO  : connect -url tcp:127.0.0.1:3121
15:36:05 INFO  : Done
15:36:13 INFO  : bpremove -all
15:36:13 INFO  : Context for 'APU' is selected.
15:36:14 INFO  : System reset is completed.
15:36:17 INFO  : 'after 3000' command is executed.
15:36:17  INFO    :  'targets  -set  -filter  {jtag_cable_name  =~  "Xilinx  Virtual  Cable
localhost:2542" && level==0 && jtag_device_ctx=="jsn-XVC-localhost:2542-14710093-0"}' command
is executed.
15:36:53   INFO     :   Device   configured   successfully   with   "/home/dev22/projects/VCS-
3/simple_reference_design/fpga/vitis/hello_world/_ide/bitstream/simple_reference_design.bit"
15:36:53         INFO              :          loadhw        -hw        /home/dev22/projects/VCS-
3/simple_reference_design/fpga/vitis/platform/export/platform/hw/simple_reference_design.xsa
-mem-ranges    [list   {0x80000000   0xbfffffff}   {0x400000000   0x5ffffffff}   {0x1000000000
0x7ffffffff}]
15:36:53 INFO  : targets -set -nocase -filter {name =~"APU*"}
15:36:53          INFO                     :            source          /home/dev22/projects/VCS-
3/simple_reference_design/fpga/vitis/hello_world/_ide/psinit/psu_init.tcl
15:37:06 INFO  : psu_init
15:37:07 INFO  : after 1000
15:37:08 INFO  : psu_ps_pl_isolation_removal
15:37:09 INFO  : after 1000
15:37:09 INFO  : psu_ps_pl_reset_config
15:37:09 INFO  : targets -set -nocase -filter {name =~ "*A53*#0"}
```

```
15:37:11 INFO  : rst -processor
15:37:15          INFO                    :          dow          /home/dev22/projects/VCS-
3/simple_reference_design/fpga/vitis/hello_world/build/hello_world.elf
15:37:15 INFO  : bpadd main
15:37:16 INFO  : con
15:37:17 INFO  : Testing the connection for 127.0.0.1
```

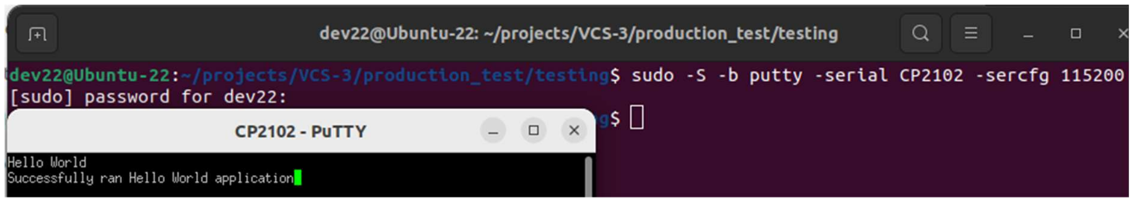
… and halts at the source code entry point:



```
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

Figure 43 – Vitis debug session for helloworld

If a terminal emulator is attached to the Lynsyn USB serial device, the message can be seen from the application when run



```
dev22@Ubuntu-22:~/projects/VCS-3/production_test/testing$ sudo -S -b putty -serial CP2102 -sercfg 115200
[sudo] password for dev22:
```

```
Hello World
Successfully ran Hello World application
```

Figure 44 – Terminal emulator showing output of helloworld

[If you encounter issues when debugging, see section 5.2]

4.4 How can I create a boot image for "hello world"?

Simply by selecting "Create Boot Image" in the Flow window …

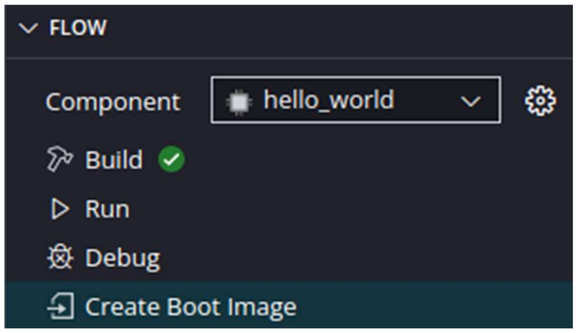
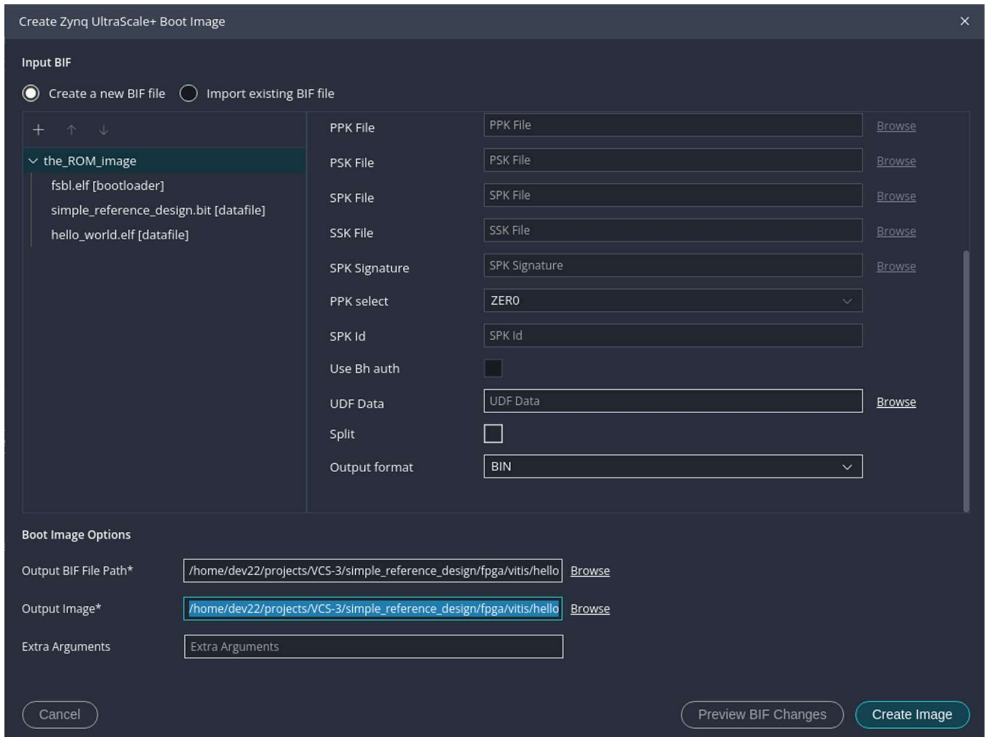


Figure 45 – Running Create Boot Image in Vitis

… and following dialog. Here we don't bother about encryption so most of the fields are empty.


Create Zynq UltraScale+ Boot Image

Input BIF

Create a new BIF file [✓] Import existing BIF file

the_ROM_image
- fsbl.elf [bootloader]
- simple_reference_design.bit [datafile]
- hello_world.elf [datafile]

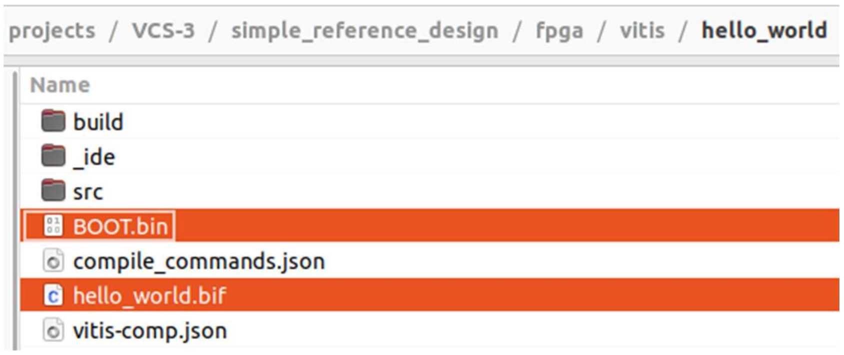
PPK File	PPK File
PSK File	PSK File
SPK File	SPK File
SSK File	SSK File
SPK Signature	SPK Signature
PPK select	ZERO
SPK Id	SPK Id
Use Bh auth	
UDF Data	UDF Data
Split	
Output format	BIN

Boot Image Options

Output BIF File Path*	/home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/hello
Output Image*	/home/dev22/projects/VCS-3/simple_reference_design/fpga/vitis/hello
Extra Arguments	Extra Arguments

Figure 46 – Create Boot Image in Vitis

The result of selecting “Create Image” is a .bin file called “BOOT.bin” and a .bif file, in this case, called “hello_world.bif”


projects / VCS-3 / simple_reference_design / fpga / vitis / hello_world

Name
- build
- _ide
- src
- BOOT.bin
- compile_commands.json
- hello_world.bif
- vitis-comp.json

Figure 47 – Output of Create Boot Image

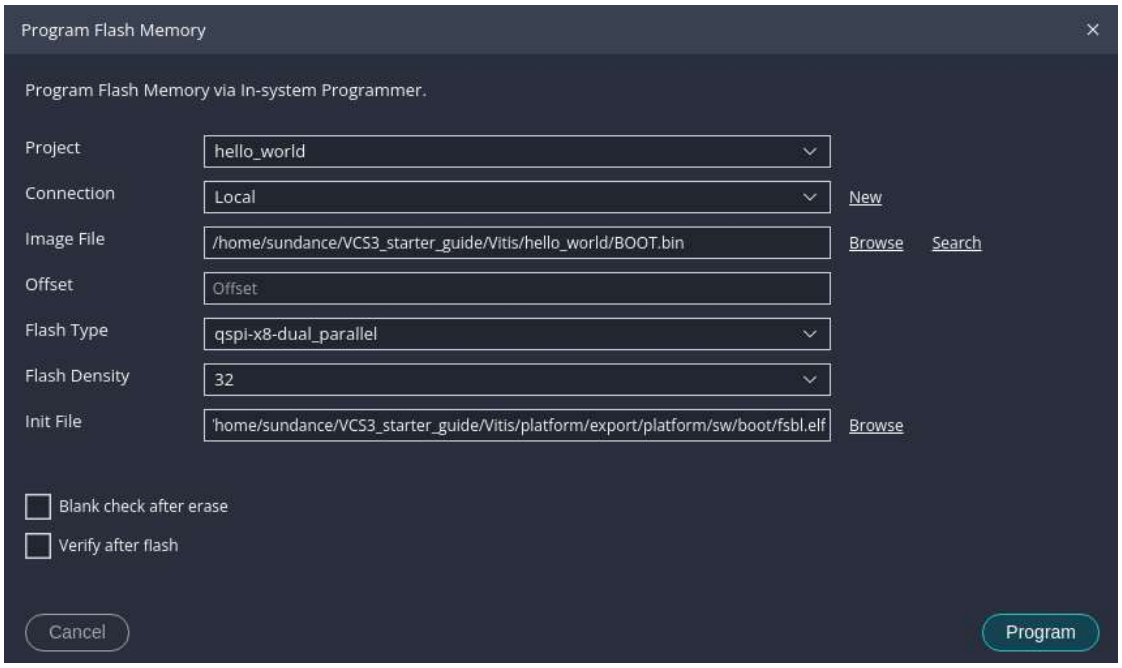
The bif file is simply a recipe for creating the bin file from three images:

- first stage boot loader (FSBL) .elf file
- programmable logic (PL) firmware application .bit file
- processor side (PS) software application .elf file

```
//arch = zynqmp; split = false; format = BIN
the_ROM_image:
{
 [bootloader, destination_cpu = a53-0]/home/dev22/projects/VCS-
3/simple_reference_design/fpga/vitis/platform/export/platform/sw/boot/fsbl.elf
 [destination_device = pl]/home/dev22/projects/VCS-
3/simple_reference_design/fpga/vitis/hello_world/_ide/bitstream/simple_reference_design.bit
 [destination_cpu = a53-0, exception_level = el-3]/home/dev22/projects/VCS-
3/simple_reference_design/fpga/vitis/hello_world/build/hello_world.elf
}
```

The .bif file can be reused to recreate the .bin file when any image is changed.

The .bin file can be programmed into eMMC or SPI ROM memory on the VCS3 using menu Vitis→Program Flash.



Program Flash Memory via In-system Programmer.

Project: hello_world
Connection: Local
Image File: /home/sundance/VCS3_starter_guide/Vitis/hello_world/BOOT.bin
Offset: Offset
Flash Type: qspi-x8-dual_parallel
Flash Density: 32
Init File: 'home/sundance/VCS3_starter_guide/Vitis/platform/export/platform/sw/boot/fsbl.elf

☐ Blank check after erase
☐ Verify after flash

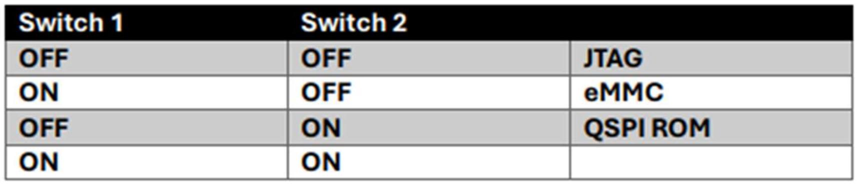
Figure 48 – Programming flash in Vitis

The process of erasing flash, programming verifying may take some time … but you can monitor progress in the Output window:

```
…
…
device 0 offset 0x578000, size 0xfd4
SF: 4052 bytes @ 0x578000 Read: OK
ZynqMP> cmp.b FFFC0000 FFFC8000 FD4

Total of 4052 byte(s) were the same
ZynqMP> INFO: [Xicom 50-44] Elapsed time = 836 sec.
Verify Operation successful.

Flash Operation Successful

Program flash finished.
--------------------------------------------------------------------------------
[04/03/2025, 16:21:29]: Program flash with id 'f3edcb0b-6395-42d3-9101-d47d5cd22f3c' ended.
```

4.5 Selecting boot memory

By setting dual switch SW1 on the VCS3 as described below, the Zynq device will be configured from the selected memory when powering up the board.



Switch 1	Switch 2	
OFF	OFF	JTAG
ON	OFF	eMMC
OFF	ON	QSPI ROM
ON	ON	

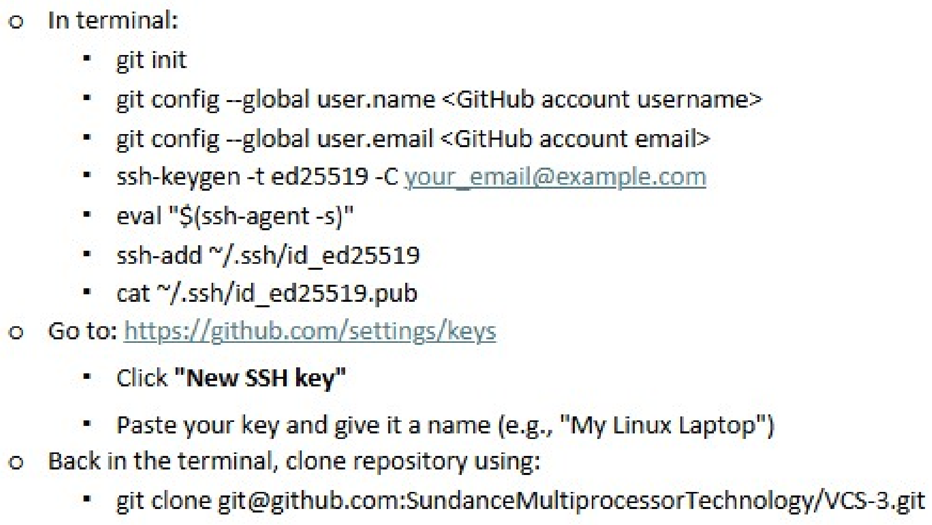
Figure 49 – Setting boot mode

5 Appendix

Additional information to catch and help solve potential issues.

5.1 Cloning the GitHub repository

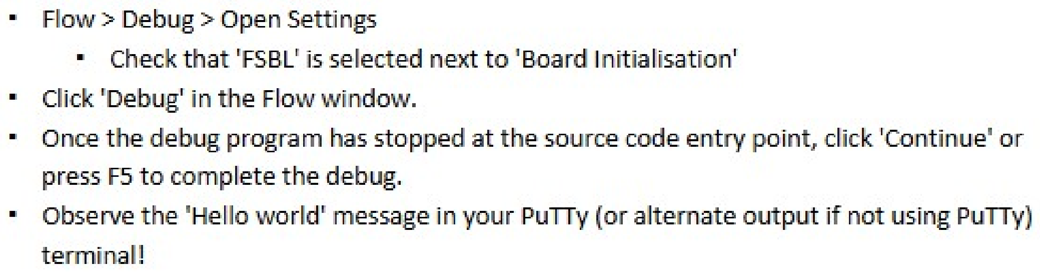
Cloning the ***https://github.com/SundanceMultiprocessorTechnology/VCS-3.git*** repository may be difficult because of access permissions.



- In terminal:
  - git init
  - git config --global user.name <GitHub account username>
  - git config --global user.email <GitHub account email>
  - ssh-keygen -t ed25519 -C your_email@example.com
  - eval "$(ssh-agent -s)"
  - ssh-add ~/.ssh/id_ed25519
  - cat ~/.ssh/id_ed25519.pub
- Go to: https://github.com/settings/keys
  - Click **"New SSH key"**
  - Paste your key and give it a name (e.g., "My Linux Laptop")
- Back in the terminal, clone repository using:
  - git clone git@github.com:SundanceMultiprocessorTechnology/VCS-3.git

Figure 50 – Creating an SSH key

5.2 Application debug help



- Flow > Debug > Open Settings
  - Check that 'FSBL' is selected next to 'Board Initialisation'
- Click 'Debug' in the Flow window.
- Once the debug program has stopped at the source code entry point, click 'Continue' or press F5 to complete the debug.
- Observe the 'Hello world' message in your PuTTy (or alternate output if not using PuTTy) terminal!

Figure 51 – Application debug help

If FSBL Board Initialisation mode fails, fully power cycle the board and retry.