Kria™ SOM documentation

Release 1.0.0

Xilinx, Inc.

Oct 28, 2021

Table of Contents

Chapter 1

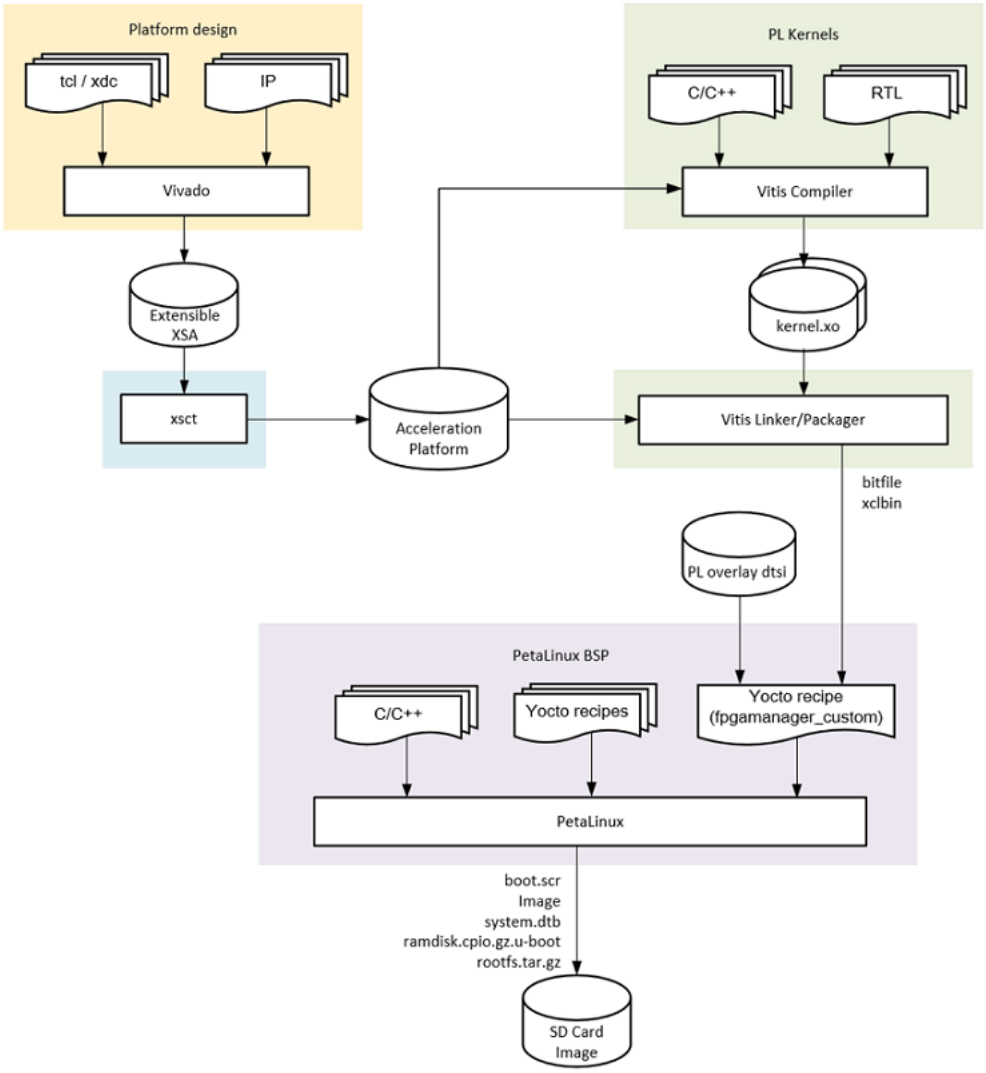
Creating Your Own Applications

1.1 Introduction

The scope of this document is to guide developers using Kria SOMs to create and test their own custom applications and programmable logic (PL) functions. Depending on the scope of development and selected workflow developers will use one or more of the Xilinx development tools (Vivado, Vitis, PetaLinux) and open source tools (e.g. Linux Device Tree Generator/Compiler) to build their applications.

The Kria SOM hardware design consist of the SOM (K26) and a carrier card. The carrier card (CC) can be a Xilinx carrier card (e.g. KV260), or a custom carrier card. The Kria K26 SOM uses the XCK26 Zynq MPSoC chip containing both the Processor Subsystem (PS) and Programmable Logic (PL). The Kria Starter Kit reference designs have a Linux operating system running in PS, which then runs applications that utilize HW accelerators implemented in PL. The PL design or bitstream is generated using Vivado and/or Vitis, and is integrated with Linux software components using PetaLinux.

The SOM board files in Vivado captures the hardware configuration of K26 SOM and maps connectivity to Xilinx provided carrier card peripherals. Developers can use Vivado to generate a custom HW design which may include a different peripheral configuration set than pre-built Xilinx reference designs. Vitis provides a design abstraction for provided "Vitis platforms" in which a subset of CC physical interface peripherals is defined and developers can focus on generating an acceleration "overlay" within the context of that platform. Developers can leverage Xilinx provided Kria Vitis platforms that align to a given CC or they can create their own Vitis platform. Developers can use the same generalized flows when creating platforms and designs for their own custom carrier card. Below is the tool flow that Xilinx uses to generate reference designs for Kria SOM.



In this document, we will outline 4 work flows: Vitis Accelerator Flow, Vitis Platform Flow, Vivado Accelerator Flow, Custom Carrier Card Flow. This section in the later part of the document gives an overview of how to choose the best flow for developers.

Below is an overview of generalized steps required to develop, build, and run applications on SOM.

1.1.1 Generate HW Configuration

The MPSoC device requires a boot-time HW configuration for the PS. This is defined in Vivado and developers must configure the MIO related I/O in this configuration step if creating a custom CC. This step is already completed if using a Xilinx provided Starter Kit BSP and/or the Vivado SOM board files which provide automation for configuring the MIO that are fixed by the SOM HW design.

1.1.2 Generate Device Boot Firmware

The boot firmware is the software that runs to support the initialization of the MPSoC platform and is captured as a BOOT.BIN design artifact. If using a Xilinx Starter Kit this step is already completed for developers. However, if developers want to create a custom boot firmware, an A/B user boot partition is provided for easier testing. Refer to Firmware Update chapter of UG1089 or wiki. The SOM Starter Kit uses a primary/secondary boot device flow where the primary boot device contains the BOOT.BIN and the secondary device contains the OS components. For Kria Starter Kits the QSPI primary boot device (BOOT.BIN) contains:

- PMU firmware
- FSBL
- ATF
- U-Boot

For Kria Starter Kits the SD card secondary device contains:

- boot.src - script read by U-boot
- Image - Linux Kernel binary
- ramdisk.cpio.gz.u-boot - Linux initramfs
- system.dtb - Linux device Tree
- rootfs.tar.gz - root file system

1.1.3 Build Application PL HW Design

The PL design defines the configuration of the PL domain of the MPSoC device on SOM, containing the PL accelerator(s). The PL configuration is defined by the user design and can be captured in Vivado or Vitis based workflows at different levels of abstraction. The design artifact captured from any of the workflows for designing a PL configuration is a bitstream (.bit, often required in .bit.bin form). This bitstream can be loaded at boot as part of the device boot firmware, or after the OS boot via a runtime library. If using a Xilinx Kria Starter Kit reference application, the bitstream is provided pre-built and always loaded after Linux boot.

1.1.4 Build Application Software

The application software refers to the SW that runs on the APU and/or RPU PS targets. This SW can be developed through Vitis, PetaLinux, or other open-source tools (e.g. Yocto). With the exception for custom Carrier Card Flow, when developing an application for SOM, developers usually do not need to rebuild entire OS. They just need to generate their application and move it onto target file system to execute.

If using a Xilinx Kria Starter Kit reference designs, applications are provided pre-built.

1.1.5 Deploy & Test On-Target

Once applications and custom HW designs are generated the user needs to move them to target.

If using the Kria Starter Kit with the Linux reference design, developers can also use on-target utilities to move their applications over and test.

If using the Kria Starter Kit a user can use various boot-modes to test monolithic boot of application software using these TCL scripts to override the Starter Kit hardware defined QSPI32 boot mode.

1.2 Prerequisites and Assumptions

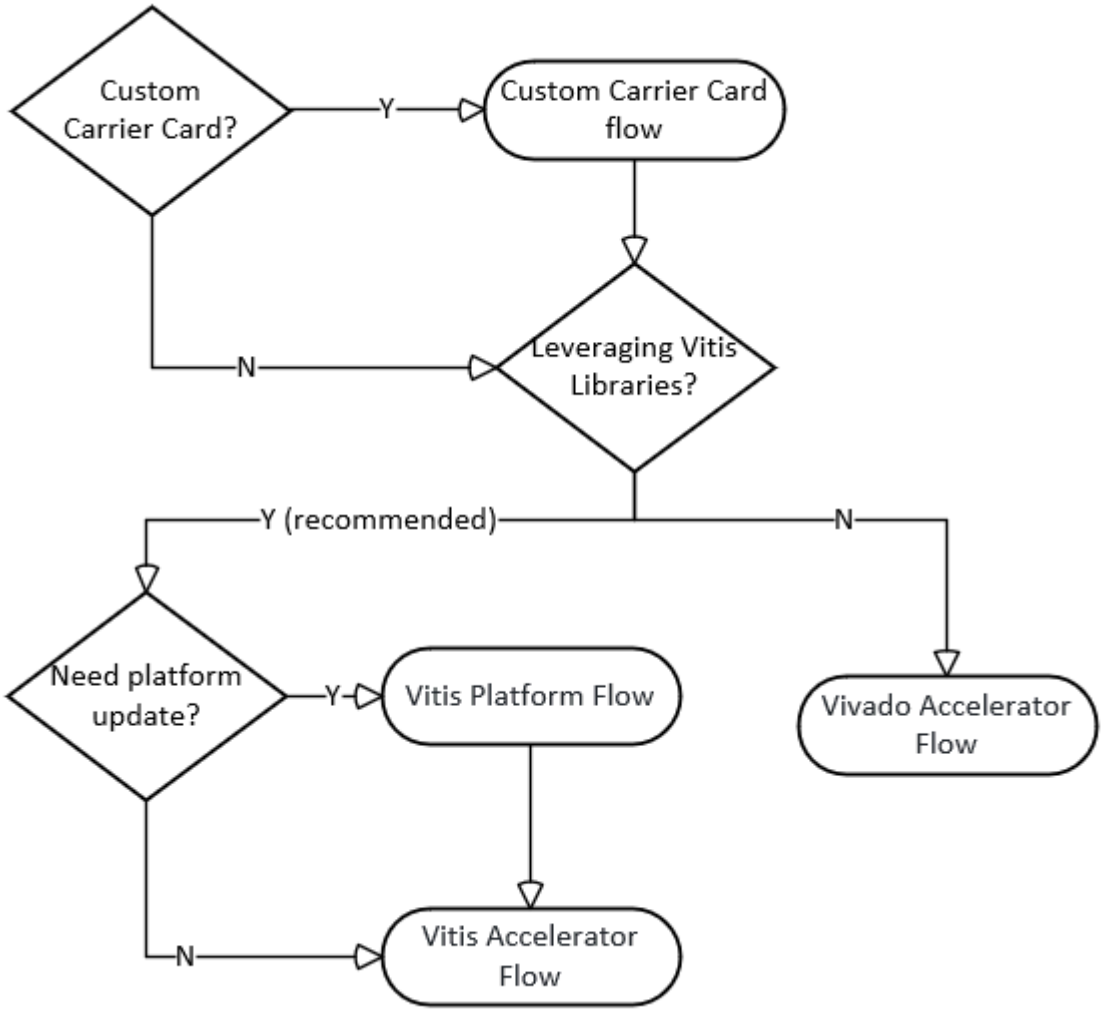
This document assumes Xilinx tools 2021.1 or later. Depending on the scope of customization and selected workflow, different tools will be needed. Please refer to various tool flow sections before determining which tools to install:

1. PetaLinux tools installation
2. Vitis tools installation (this will include Vivado)
3. Vivado tools installation (if Vitis is not required and installed)
4. Device Tree Generator (DTG) and Device Tree Compiler (DTC) installation, refer to Build Device Tree Blob

5. XSCT (will be installed as part of Vivado or Vitis)
6. PetaLinux SOM StarterKit BSP (e.g. xilinx-k26-starterkit-2021.1-final.bsp)
    - Download the latest SOM Starter Kit BSP from the SOM Wiki
7. Kv260-vitis git repository

1.3 SOM Developer Flow

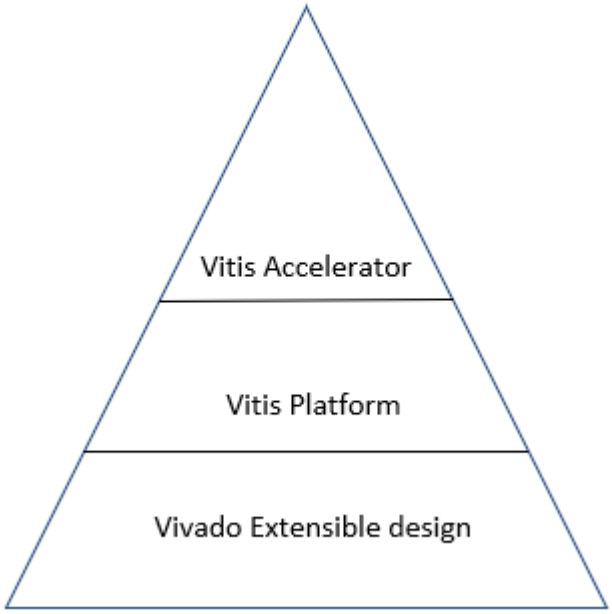
Developers may only need to touch parts of the flow to put their own applications on SOM. Based on the scope of hardware and design change there are four different flows developers can leverage when developing a custom application. The choice of flow depends on HW target definition, where the target design intersects with Xilinx's released reference designs, and tool preference. The following is a decision tree to help guide developers through appropriate workflows:



If developers plan to create their own custom carrier card, they will need to first go through the custom carrier card flow before generating their design through the Vitis or Vivado based tool flows.

If developers are leveraging a Xilinx carrier card, or have finished developing the needed base Linux designs for their custom carrier card, they will then need to generate application designs. The recommended tool for applications such as vision and video application is Vitis because Vitis supplies the Vitis_Accel_libraries and allows developers to develop quickly. Alternatively, if those advantages are

not needed or wanted, developers can also use Vivado Accelerator flow. In the Vitis tool flow, developers can leverage the Vitis Platform in the example designs and jump directly into the Vitis Accelerator flow, or they might need to update their platform using Vitis Platform flow before going into the Vitis Accelerator flow.



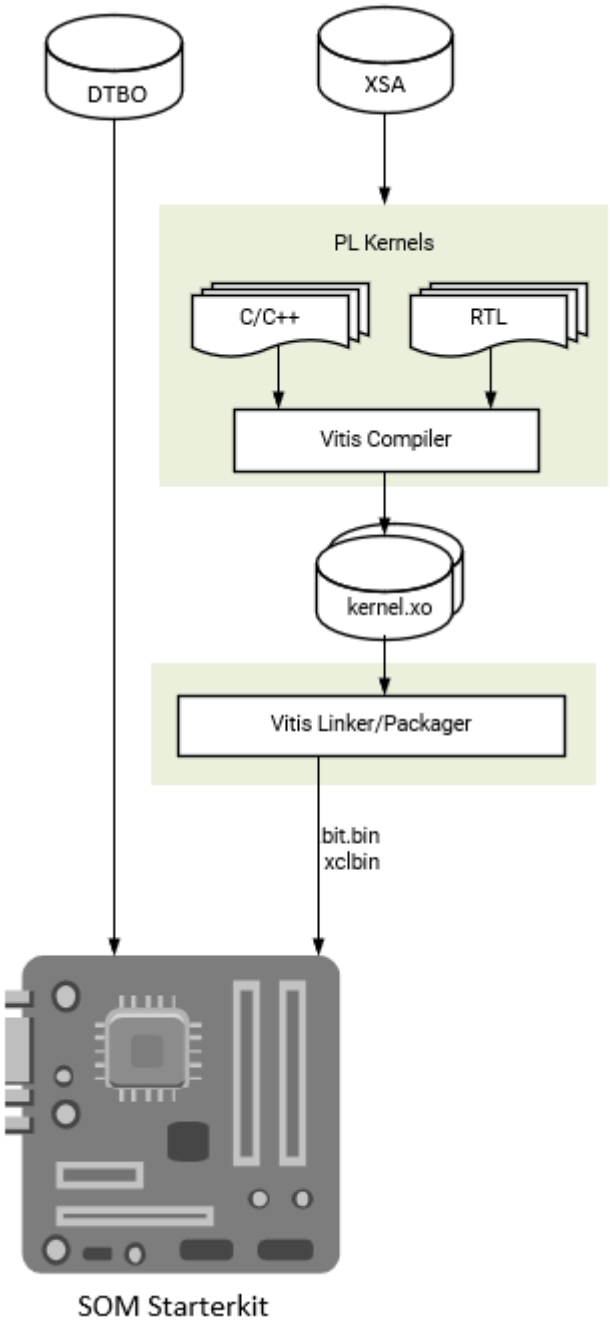
The diagram above shows the hierarchy between Vivado Extensible design, Vitis Platform and Vitis Accelerators. A Vivado Extensible design is required first to in-take board information and create proper configurations. Then a Vitis Platform can be generated on top of the Vivado Extensible design. After that, Vitis Accelerators and applications can be created in the Vitis Platform. User can intersect in different layers dependent on where their design diverge from Xilinx examples.

1.3.1 Vitis Accelerator Flow

This flow is for developers using Xilinx provided SOM Starter Kit Vitis Platforms as a basis for generating their own PL accelerators. Developers use a Vivado Extensible Platform (.xsa) file provided by Xilinx and import it into a Vitis Platform project. Developers then create their own overlay accelerator(s) within the bounds of the provided Vitis Platform, and generate a new bitstream (.bit file converted to .bit.bin) and metadata container file (.xclbin). Developers can use the existing device tree blob (.dtb) associated with the Xilinx provided Vitis platform. The resulting application accelerator files are then moved to the target SOM platform to be run.

- Constraints: developers must use the same carrier card and physical peripheral definition as Xilinx provided SOM Starter Kit Vitis Platforms
- Input: Xilinx provided Vitis Platform (.xsa), Xilinx provided Vitis platform device tree (.dtbo)
- Output: .bit.bin, .xclbin

Details of Vitis Accelerator Flow can be found here



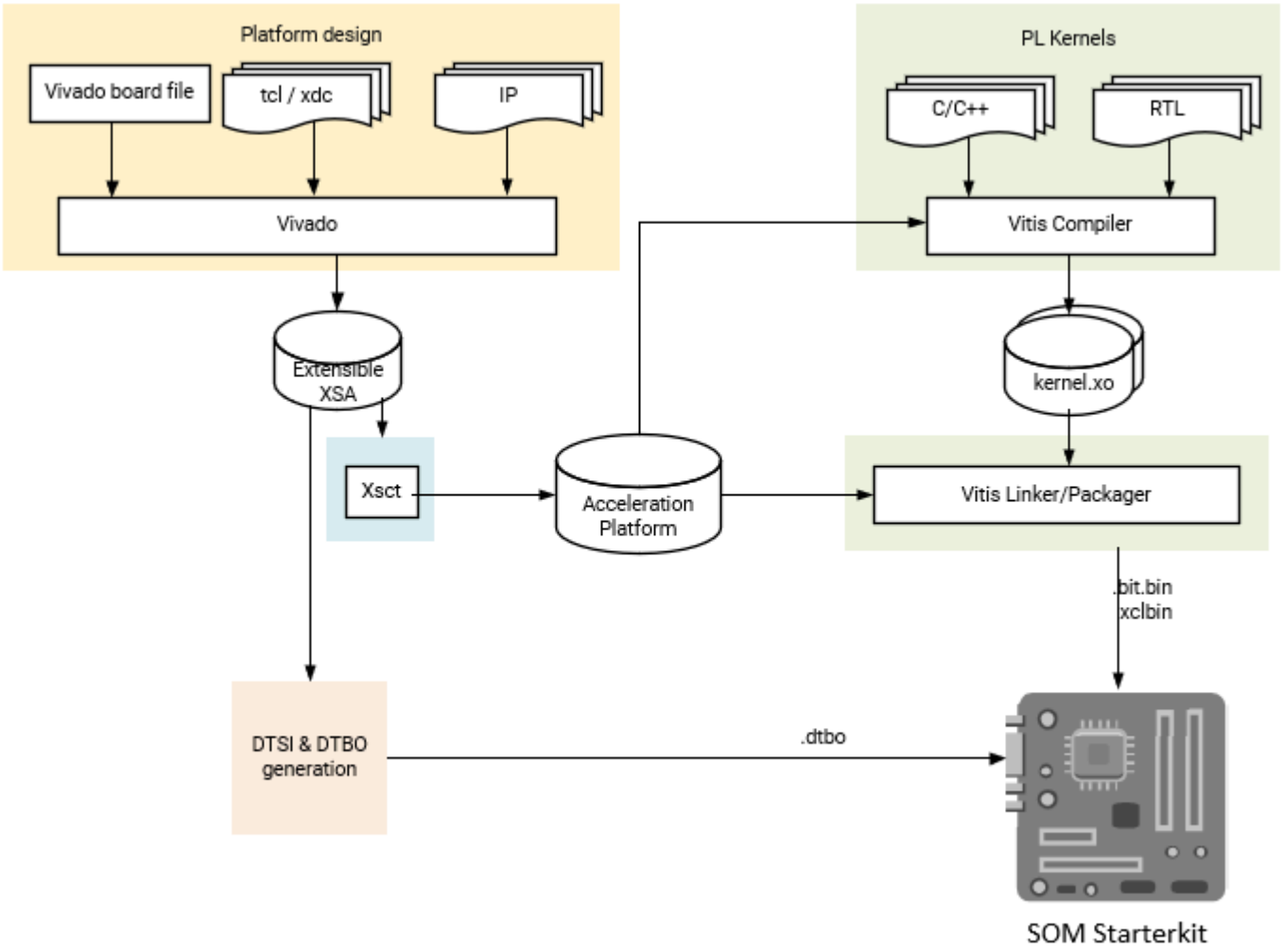
1.3.2 Vitis Platform Flow

Developers can create a custom Vitis platform if they require a different set of physical PL I/O peripherals than those provided in Xilinx generated platforms. Development starts with the Vivado tool to create an extensible hardware platform. In Vivado, the Kria SOM Starter Kit Vivado board file is provided. It automatically drives the PS subsystem HW configuration and provides pre-defined connectivity for commonly used PL IPs based on the selected carrier card (e.g. MIPI interfaces on KV260 carrier card). Developers use Vivado to generate a custom .xsa file to be ported into Vitis as a platform project. Once the platform project is created, then a corresponding device tree overlay is generated. With the extensible .xsa and .dtbo developers can now follow the same flow outlined in Vitis Accelerator flow. The resulting bitstream, .xclbin, and .dtbo files are copied into the target.

- Assumption: Xilinx provided SOM carrier card with associated Vivado board file automation
- Input: Vivado SOM Starter Kit board file

- Output: .dtbo, .bit.bin, .xlcbin

Details of the Vitis Platform flow can be found here

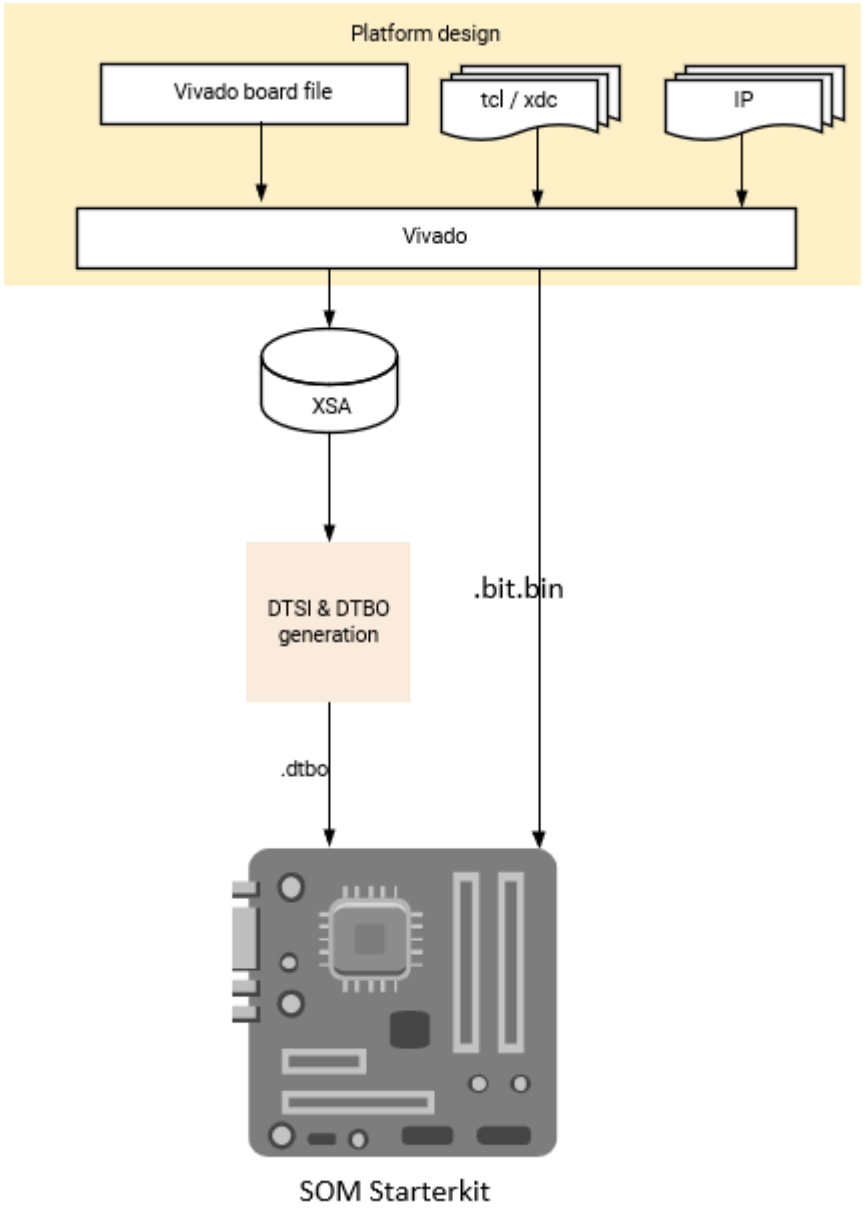


1.3.3 Vivado Accelerator Flow

Developers prefer a traditional HW design flow can generate their PL designs using Vivado. In this flow developers start from the Kria SOM starter kit board files in Vivado and implements their own PL design in Vivado to generate a .xsa file and bitstream. The resulting .xsa file is used to generate the device tree overlay. Once the PL design (.bit.bin) and HW/SW interface definition (.dtbo) files are created, they can be copied into the target and managed by dfx-mgr.

- Assumption: Xilinx built carrier cards with corresponding SOM Starter Kit board file
- Input: SOM Starter Kit board file (in Vivado) or Vivado project released with SOM BSP, developer's own accelerator designs in Vivado
- Output: .dtbo, .bit

Details of Vivado Accelerator flow can be found here



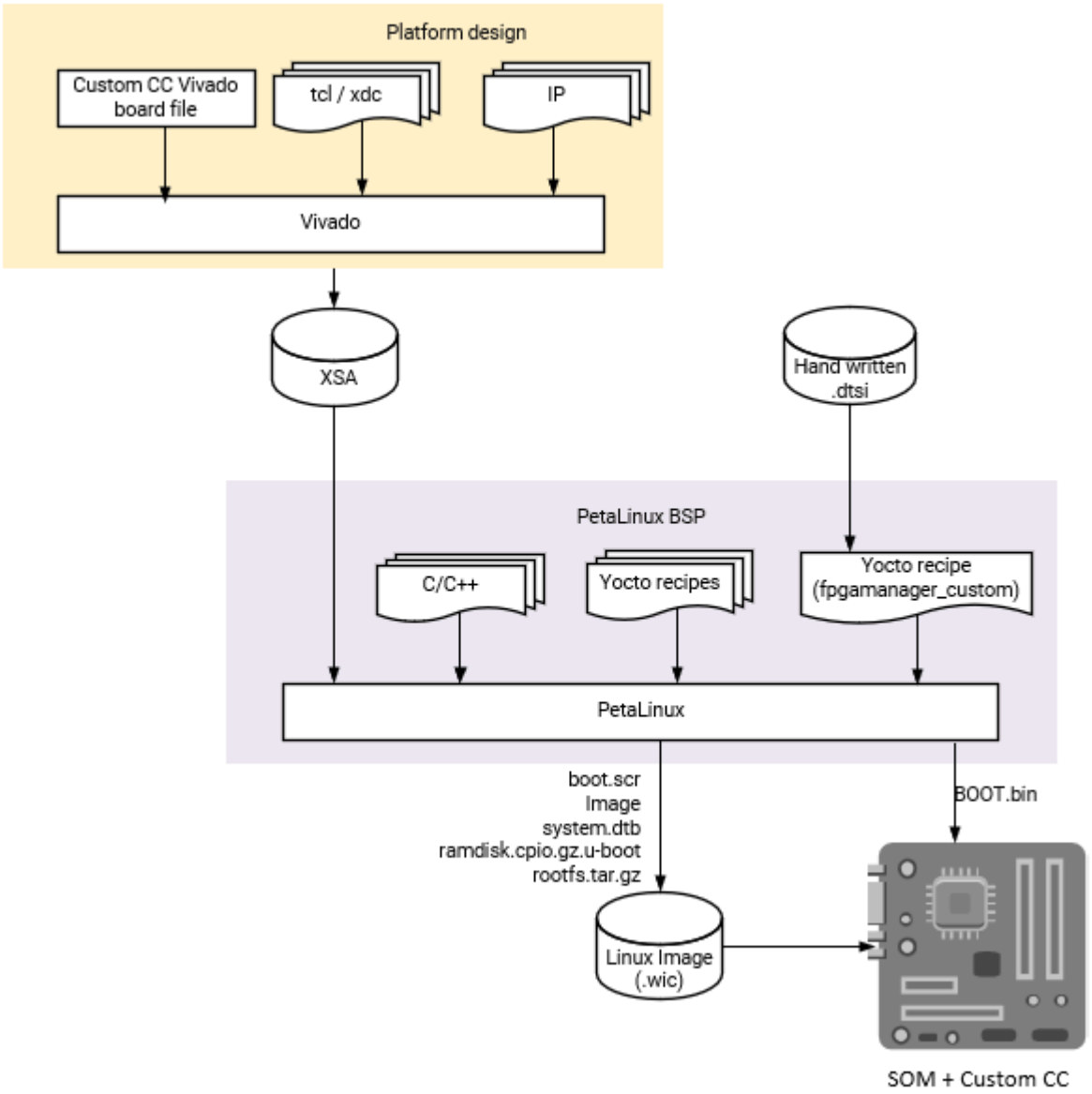
1.3.4 Custom Carrier Card Flow

Developers creating their own carrier card will create a Vivado project using the Xilinx provided K26 production SOM Vivado board file as a starting point. The K26 board file contains the MIO configuration defined by the SOM HW design, and provides a minimal HW configuration to boot to Linux. The K26 board file does not contain any information specific to a carrier card. Developers then design in their specific custom MIO and PL based physical interfaces to create their own custom HW configuration while following the Kria CC Design Guide (UG1091). After creating the integrated SOM + CC configuration, a .xsa file is exported. If using Linux, developers then create a Petalinux project to generate boot and OS images for booting Linux. Developers can then use the artifacts to create applications to run on top of the base Linux, using the previously discussed workflows: Vitis Accelerator Flow, Vitis Platform Flow, or Vivado Accelerator Flow.

- assumption: Using SOM K26 with developer defined carrier card
- input: Vivado K26 SOM board file, customer defined carrier card board configuration
- output: BOOT.bin, .wic image containing boot.src, Image, ramdisk.cpio.gz.u-boot, system.dtb,

rootfs.tar.gz

Details of custom carrier card flow can be found here



1.3.5 Bare-metal & Non-Linux Application Workflow on SOM

While the Kria Starter Kits examples are Linux centric, they can be used for bare-metal applications by using user application hooks provided in the boot FW architecture and corresonding XSDB debug hooks. The Kria Starter Kit pre-built firmware includes two user partitions labeled “A” and “B”. Only one is active at a time and is controlled by the Image Selector application at boot. Developers can load their bare-metal based application BOOT.BIN to the partition to one of the user partitions and boot their custom application with QSPI32 boot mode. Alternatively, developers can use the Xilinx System Debugger (XSDB) and JTAG to load and boot their application on the Starter Kit.

- To load custom BOOT.BIN to A/B partitions, use the Linux based xmutil image update utility or use the platform recovery tool. See the Kria Wiki or UG1089 for details on loading a BOOT.BIN to the user A/B partitions.
- See the Application Boot Modes for setting JTAG boot via XSDB.

1.4 Kria SOM References

- Kria SOM Wiki
- Kria SOM K26 Data Sheet DS987
- Kria SOM KV260 Data Sheet DS986
- Kria SOM KV260 User Guide UG1089
- Kria SOM Carrier Card Design Guide UG1091
- Zynq MPSoC TRM UG1085

1.4.1 Tool Documentation

Vitis documentation:

- Vitis Unified Software Development Platform 2021.1 User Guides
- Vitis Platform web-based documentation

Vivado documentation

- Vivado Design Suite Tutorial UG940
- Vivado Design Flow Overview UG892
- Vivado System Level Design Entry UG895
- Vivado Design Suite User Guide UG896
- Vivado Design Suite Using Constraints UG903

Petalinux Documentation

- PetaLinux User Guide UG1144

Device Tree Generator Documentation

- Build Device Tree Blob

1.5 File extension appendix:

- .bit.bin - binary file for bitstream - this is the .bin file that can be generated from Vivado/Vitis instead of .bit file
- .bsp - board support package
- .dtb - Device Tree Blob. A binary file containing binary data that describes hardware, compiled from .dtb and .dtbi files
- .dtbo - Device Tree Blob Overlay. A binary file containing hardware that can be overlaid on top of existing .dtb file
- .dts - Device Tree Source. This is typically the top level (board level) device tree description
- .dtsi - Device Tree Source Include. These files are typically used to describe hardware on a SoC and in this case, the PL designs as well
- .elf - Executable and Linkable Format - contains compiled software

- .wic - wic image helps simplify the process of deploying a platform project image to test by including the required boot, rootfs, and related partitions in the image. As a result, all developers need to do is copy the image to a storage device and use it to boot the hardware target device.
- .xdc - Xilinx Design Constraint file - indicate pin mapping, and pin constraints in Vivado
- .xml - The XML board file is a configuration file used by vivado to create board related configuration
- .xlcbin - device binary file, also known as AXLF file. it is is an extensible, future-proof container of (bitstream/platform) hardware as well as software (MPSoC/Microblaze ELF files) design data. In the flows above, the .xlcbin file has information about address space of the PL design
- .xsa - Xilinx Shell Archive. These files are generated by Vivado to contain the required hardware information to develop embedded software with Vitis. Can only be opened with Xilinx tools

1.5.1 License

Setting Bootmodes

Once applications and custom HW designs are generated, developers need to move them to target. If using the Kria Starter Kit, developers can use various boot-modes to test monolithic boot to application software using the following TCL scripts to set the preferred development boot process. Developers will first put the functions in a `<boot>.tcl` script. Then, with the host machine connected with their SOM kit, they use the following commands in XSDB or XSCT:

```
connect
source <boot>.tcl
boot_<mode>
```

To set K26 to JTAG bootmode using XSDB/XSCT, add the following TCL scripts and call the function:

```
proc boot_jtag { } {
############################
# Switch to JTAG boot mode #
############################
targets -set -filter {name =~ "PSU"}
# update multiboot to ZERO
mwr 0xffca0010 0x0
# change boot mode to JTAG
mwr 0xff5e0200 0x0100
# reset
rst -system
}
```

To set K26 to SD bootmode using XSDB/XSCT, add the following TCL scripts and call the function:

```
proc boot_sd { } {
############################
# Switch to SD boot mode #
############################
targets -set -filter {name =~ "PSU"}
# update multiboot to ZERO
mwr 0xffca0010 0x0
# change boot mode to SD
mwr 0xff5e0200 0xE100
# reset
rst -system
#A53 may be held in reset catch, start it with "con"
after 2000
con
}
```

To set K26 to QSPI bootmode using XSDB/XSCT, add the following TCL scripts and call the function:

```
proc boot_qspi { } {
############################
```

```
# Switch to QSPI boot mode #
############################
targets -set -filter {name =~ "PSU"}
# update multiboot to ZERO
mwr 0xffca0010 0x0
# change boot mode to QSPI
mwr 0xff5e0200 0x2100
# reset
rst -system
#A53 may be held in reset catch, start it with "con"
after 2000
con
}
```

To set K26 to eMMC bootmode using XSDB/XSCT, add the following TCL scripts and call the function:

```
proc boot_emmc { } {
############################
# Switch to emmc boot mode #
############################
targets -set -nocase -filter {name =~ "PSU"}
stop
# update multiboot to ZERO
mwr 0xffca0010 0x0
# change boot mode to EMMC
mwr 0xff5e0200 0x6100
# reset
rst -system
#A53 may be held in reset catch, start it with "con"
after 2000
con
}
```

To set K26 to USB bootmode using XSDB/XSCT, add the following TCL scripts and call the function:

```
proc boot_usb { } {
############################
# Switch to usb0 boot mode #
############################
targets -set -nocase -filter {name =~ "PSU"}
stop
# update multiboot to ZERO
mwr 0xffca0010 0x0
# change boot mode to EMMC
mwr 0xff5e0200 0x7100
# reset
rst -system
#A53 may be held in reset catch, start it with "con"
after 2000
con
}
```

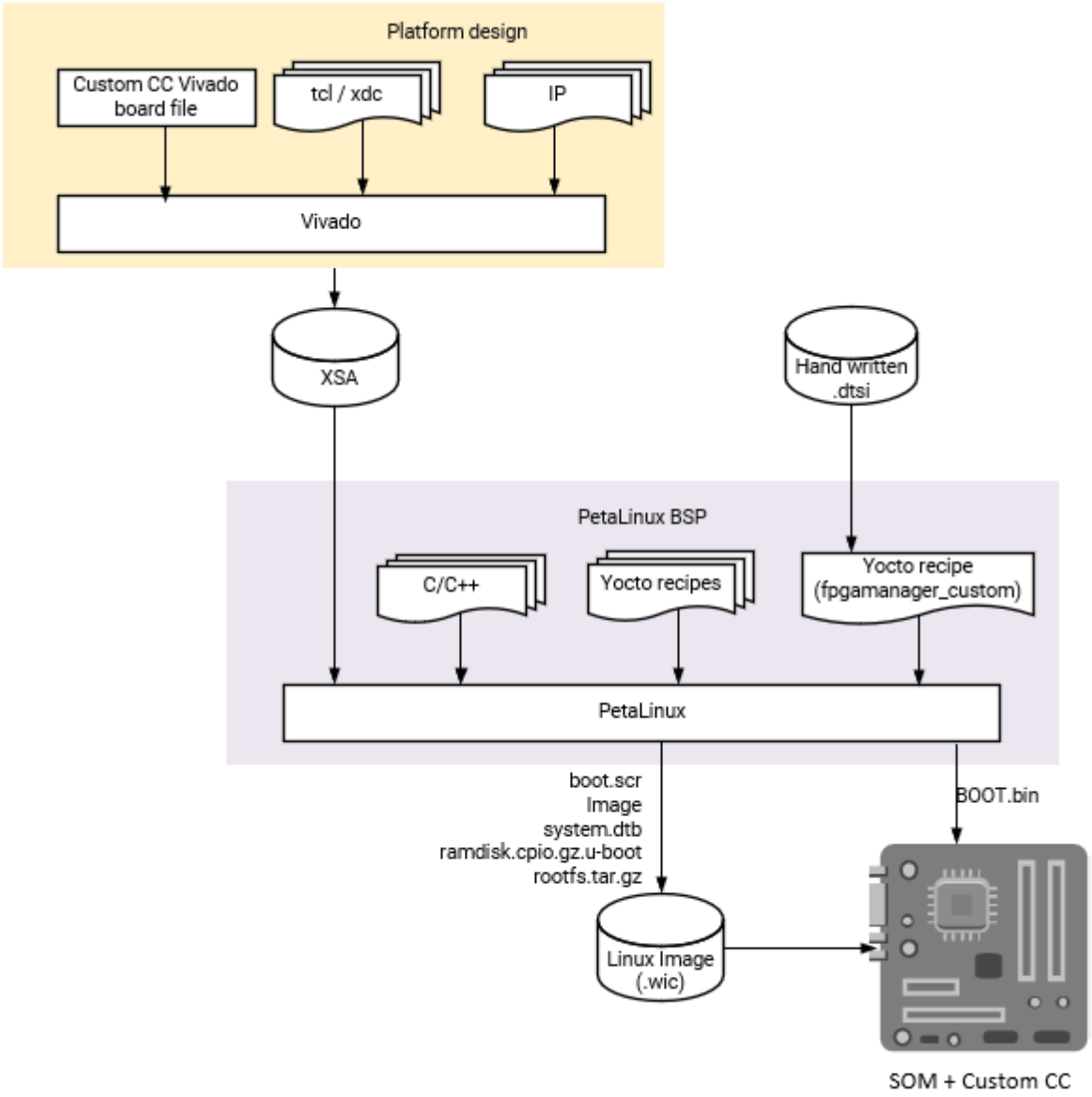
2.1 License

either express or implied. See the License for the specific language governing permissions and limitations under the License.

Custom Carrier Card Flow

Developers creating their own carrier card will create a Vivado project using the Xilinx provided K26 production SOM Vivado board file as a starting point. The K26 board file contains the MIO configuration defined by the SOM HW design, and provides a minimal HW configuration to boot to Linux. The K26 board file does not contain any information specific to a carrier card. Developers then design in their specific custom MIO and PL based physical interfaces to create their own custom HW configuration while following the Kria CC Design Guide (UG1091). After creating the integrated SOM + CC configuration, a .xsa file is exported. If using Linux, developers then create a Petalinux project to generate boot and OS images for booting Linux. Developers can then use the artifacts to create applications to run on top of the base Linux, using the previously discussed workflows: Vitis Accelerator Flow, Vitis Platform Flow, or Vivado Accelerator Flow.

- assumption: Using SOM K26 with developer defined carrier card
- input: Vivado K26 SOM board file, customer defined carrier card board configuration
- output: BOOT.bin, .wic image containing boot.src, Image, ramdisk.cpio.gz.u-boot, system.dtb, rootfs.tar.gz


Platform design
Custom CC Vivado board file
tcl / xdc
IP
Vivado
XSA
Hand written .dtsi
PetaLinux BSP
C/C++
Yocto recipes
Yocto recipe (fpgamanager_custom)
PetaLinux
boot.scr
Image
system.dtb
ramdisk.cpio.gz.u-boot
rootfs.tar.gz
BOOT.bin
Linux Image (.wic)
SOM + Custom CC

Prerequisites and Assumptions

This document assume that developer will use 2021.1 or later for tools and SOM content releases. The tool versions should match - e.g. use the same tool versions for PetaLinux, Vivado, and the released BSP.

1. Vitis tools installation
2. Vivado tools installation
3. PetaLinux tools installation

Step 1 - Aligning Kria SOM boot & SOM Linux infrastructure

Xilinx built Kria SOM Starter Kit applications on a shared, application-agnostic infrastructure in the SOM Starter Linux including kernel version, Yocto project dependent libraries, and baseline BSP. When using this tutorial, make sure to align tools, git repositories, and BSP released versions.

5.1 PetaLinux BSP Alignment

The SOM Starter Linux image is generated using the corresponding SOM variant multi-carrier card PetaLinux board support package (BSP). Developers creating applications on the Starter Kit are recommended to use this BSP as a baseline for their application development as it ensures kernel, Yocto project libraries, and baseline configuration alignment. The multi-carrier card BSP defines a minimalistic BSP that has the primary function of providing an application-agnostic operating system, and can be updated and configured dynamically at runtime.

Step 2 - Create board file and .xdc file for custom carrier card

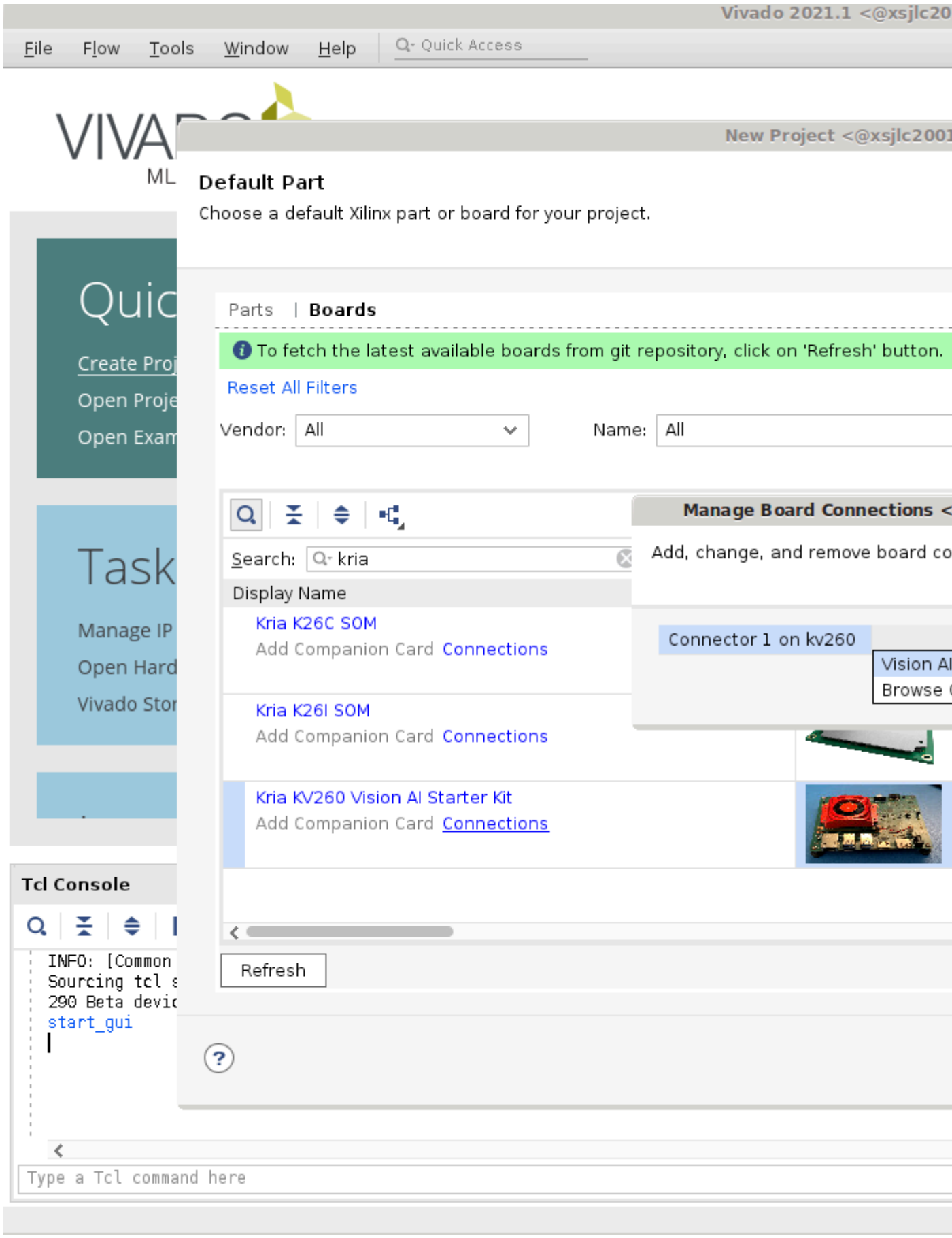
In example carrier card projects, we have created two board configuration files used by Vivado to create board related configurations - one for K26 and one for KV260. K26C/K26I, and example KV260 board files can also be found here (Please note that board files are available in 21.1 as well, though starting in 2021.2, board xmls capture pcb_min_delay/pcb_max_delay for all pins extended to carrier card). Information on generating custom Vivado board files and where to place them can be found in UG895. Developers will need to create a custom board file for their custom carrier card, and they can use KV260 as an example.

Developers also need .xdc file to create mappings between Xilinx MPSoC pins and connectors, as well as adding constraints. K26 and KV260 .xdc files can both be found on Kria documentation site. K260 .xdc file contains information on how MPSoC pins maps to the connector, and developer can use that information to create their own .xdc for their carrier card. KV260 .xdc file can be used as an example. Developers should refer to Vivado Design Suite using Constraints UG903 for more information on creating their .xdc file.

Step 3 - Generate a new custom PL design using Vivado

This flows starts with vivado board files containing information on K26 and custom CC. The K26 SOM is supported in Vivado with three board files that automate the configuration of the SOM based peripherals. These board files are available in Vivado's board list in "Create Project" wizard in 2021.1 or later.

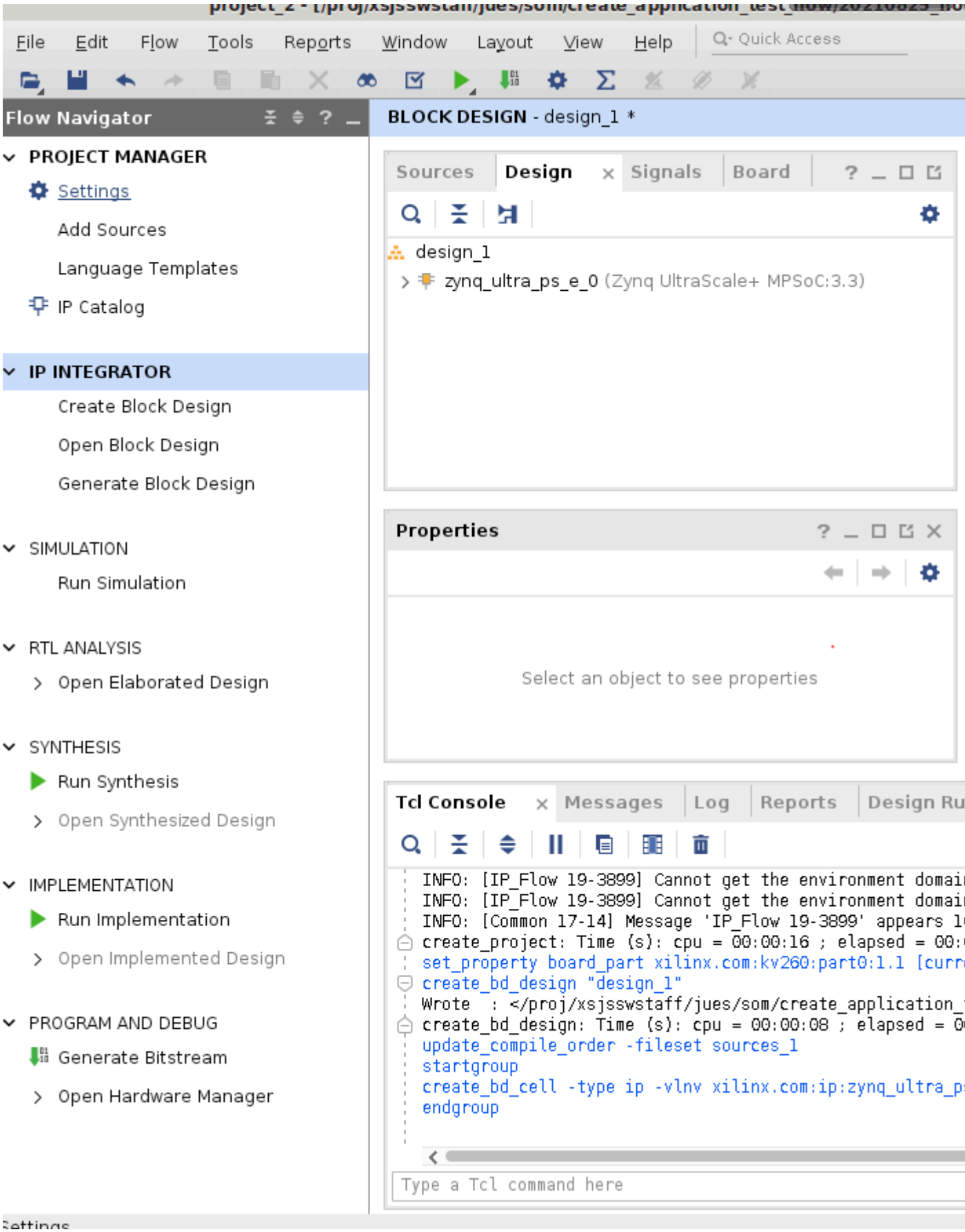
- K26C SOM - Commercial grade K26 SOM.
- K26I SOM - Industrial grade K26 SOM.
- KV260 Starter Kit - K26 based Starter Kit SOM with Vision AI Carrier Card connector interface.


Vivado 2021.1 <@xsjlc20
File
Flow
Tools
Window
Help
Quick Access
New Project <@xsjlc2001
Default Part
Choose a default Xilinx part or board for your project.
Parts
Boards
To fetch the latest available boards from git repository, click on 'Refresh' button.
Reset All Filters
Vendor:
All
Name:
All
Manage Board Connections
Add, change, and remove board co
Search:
kria
Display Name
Kria K26C SOM
Add Companion Card
Connections
Connector 1 on kv260
Vision AI
Browse
Kria K26I SOM
Add Companion Card
Connections
Kria KV260 Vision AI Starter Kit
Add Companion Card
Connections
Refresh
Tcl Console
INFO: [Common
Sourcing tcl s
290 Beta devic
start_gui
Type a Tcl command here

Be sure to select the custom board file you have created in step 2 instead of KV260 board.

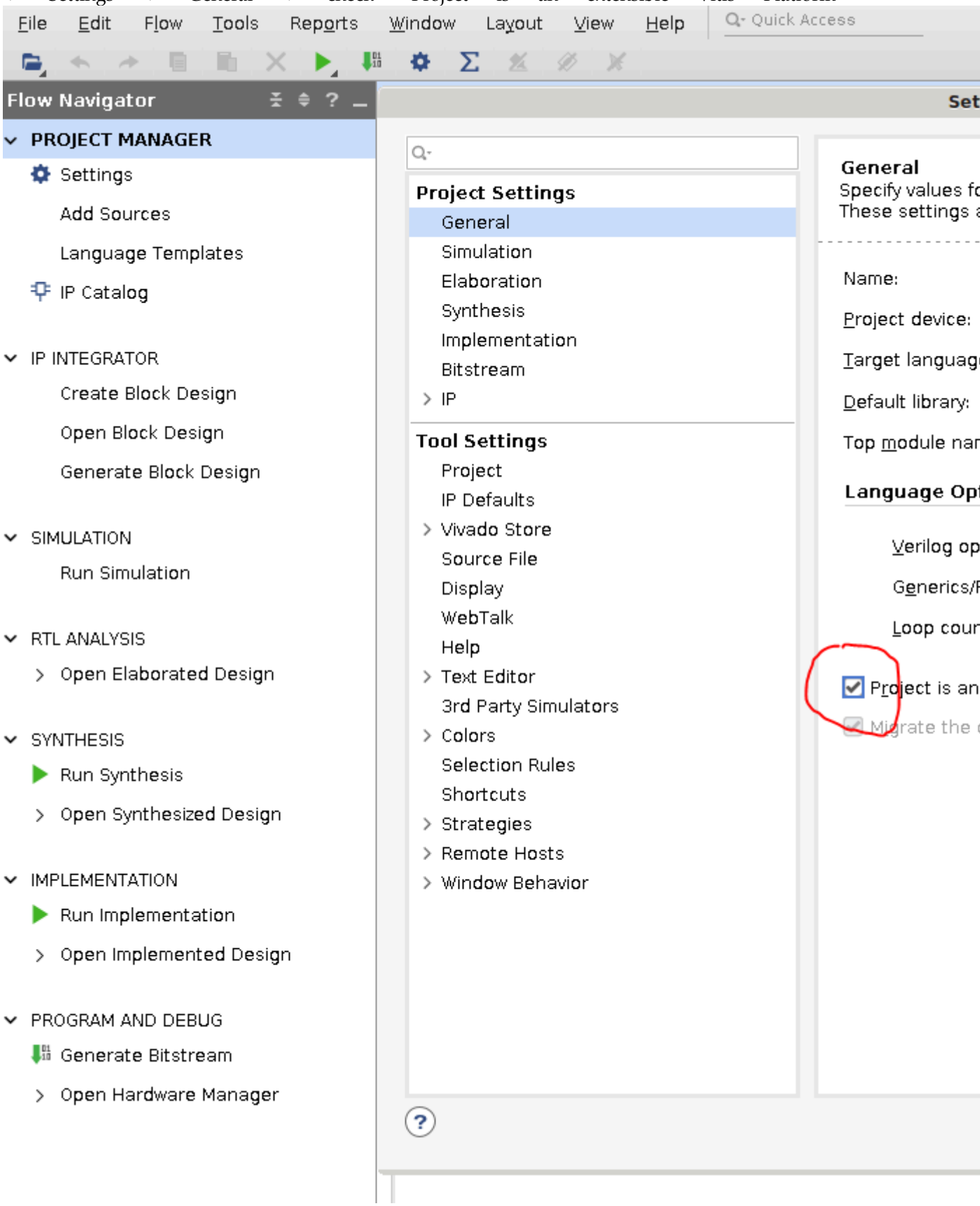
When selecting the Kria starter kit board file, make sure to click on “connections” to indicate that the K26 and KV carrier card are connected.

Once Zynq_ultra_ps_e_0 block is added to a design, make sure to click “Run Block Automation” to apply board file settings.



Developers then may need to indicate that the platform is an Extensible Vitis Platform. Whether to

make the project extensible or not is dependent on if developer intend to reuse the platform for developing in Vitis. More details on how to create Extensible Platform can be found here. Project Manager -> Settings -> General -> check "Project is an extensible Vitis Platform"



Developers then should add the .xdc files for both K26 and their custom carrier card.

7.1 Vivado Starter Project in BSP

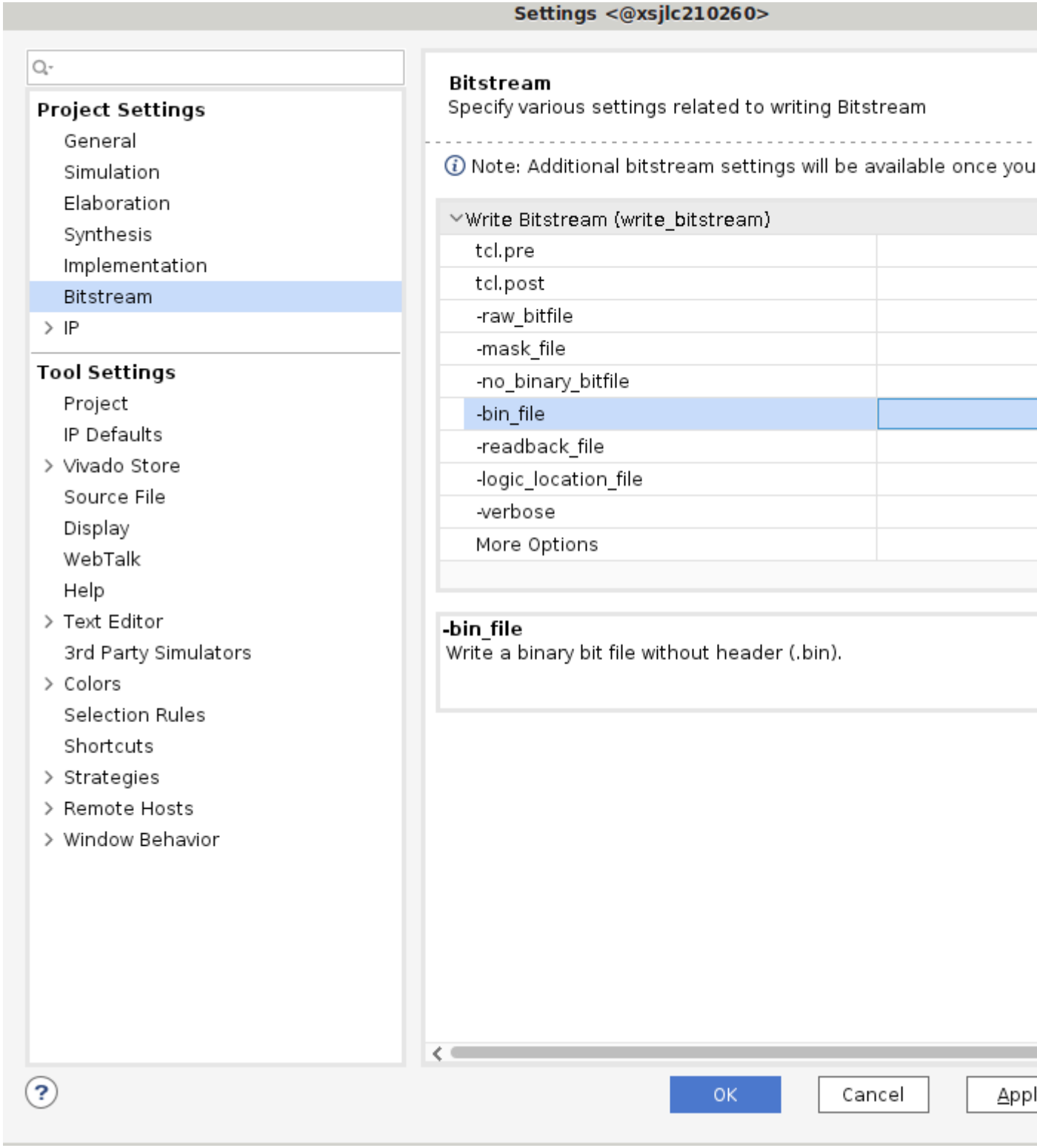
Alternatively, developers can start from the Vivado project provided in the BSP file. First, download the SOM Starter Kit BSP from the SOM Wiki Then create the project using BSP:

```
petalinux-create -t project -s xilinx-k26-starterkit-v2021.1-final.bsp
cd xilinx-k26-starterkit-2021.1
```

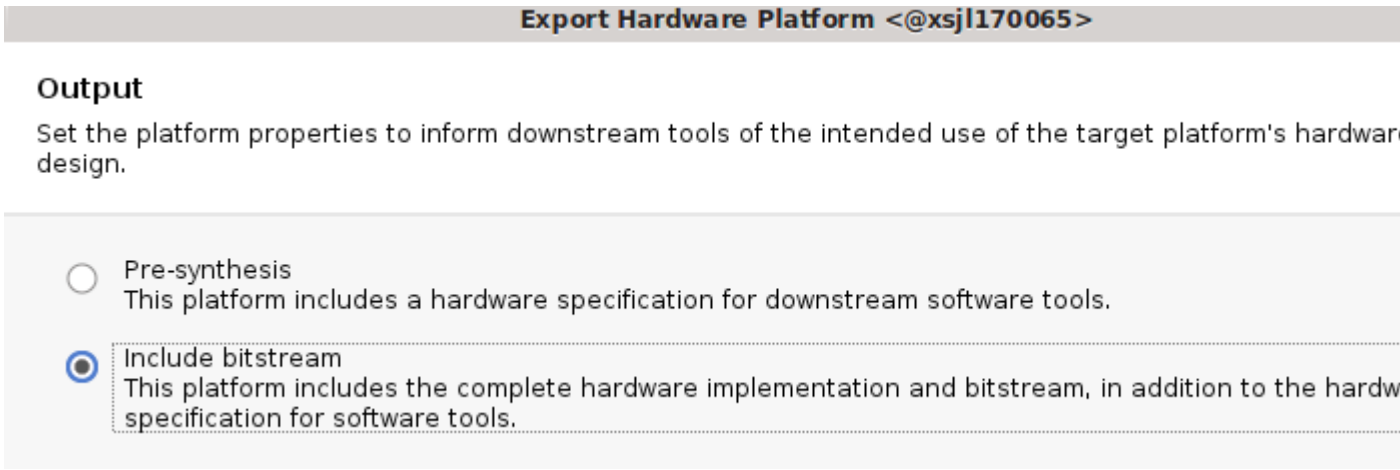
The Vivado starter project can be found in `hardware/` folder, and developers can open the project using the .xpr file. This project is a k26 project only, and will not contain any information about the carrier card being used. However, it does have enough information to boot basic Linux.

7.2 Generate .bit.bin and .xsa file

Please refer to Vivado documentation to add custom IP blocks into your design and generate a .xsa file and a binary bitstream (.bin file converted .bit.bin file). In order to generate a binary bitstream (.bin) file, go to Tools -> setting and enable -bin_file, and rename the generated .bin file to .bit.bin.



An updated .xsa file needs to be generated by using File -> Export -> Export Hardware , make sure to select “include bitstream” in the generation.



An example base design for booting Linux can be found in PetaLinux project folder generated from SOM starterkits BSP in `project-spec/hw-description/system.xsa`.

If using Linux the developer will also need to create a dtsi file for the project. An example dtsi file can be found here.

step 4 - Create Linux boot image from .xsa in Petalinux

Please refer to UG1144 for detailed guide to create an image. Below are the commands extracted from the guide:

```
petalinux-create --type project --template zynqMP --name <petalinux_project>
cd <petalinux_project>
petalinux-config --get-hw-description <path to .xsa file>
cp <path to .dtsi file>
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
petalinux-build
```

the image and boot files can be found in `images/linux/`

Developer can also use the SOM K26 production BSP to generate a Petalinux project and images. It does not have any information on any carrier cards, but it has enough SOM information to generate an image to boot to Linux and also include utilities such as xmutil and dfx-mgr. This can be used for bring up. To use released BSP, replace the above `petalinux-create` command with the following:

```
petalinux-create -t project -s xilinx-k26-starterkit-v2021.1-final.bsp
```

Step 5 - Boot new image on target platform

Depending on what developers decide to put on their custom carrier card, they may choose different boot modes. JTAG booting should be available on all carrier cards and this section will detail how to boot Linux via JTAG using PetaLinux. Details on using PetaLinux to boot Linux can be found in Petalinux User Guide to help with booting with preferred bootmode.

First package the built images into pre-built folder:

```
petalinux-package --prebuilt
```

Then generate a xsdb tcl script <test.tcl>:

```
petalinux-boot --jtag --prebuilt 3 --tcl <test.tcl>
```

<test.tcl> will contain a script to boot Linux with tiny rootfs and exit xsdb, assuming that user is still in PetaLinux project folder. Take care to copy needed files and change pointers to files if you need to move the files to a different host machine to boot. Remove "exit" at the end of the tcl script to observe printouts from the script in xsdb.

In xsdb, first connect and put the board in JTAG mode using xsdb commands here. Then `source <test.tcl>`, and Linux should boot.

Step 6 - Develop Applications

Now that a base Linux design is up a running, developers can follow Vitis Accelerator Flow, or Vitis Platform Flow, or Vivado Accelerator Flow to develop their applications.

10.1 License

Chapter 11

Generating DTSI and DTBO Overlay Files

In this step the Xilinx HW description captured in the custom PL design must be translated into a Linux understandable format. In Linux HW is described using a concept called device trees (DT). The human readable form of these are dts and dtsi files. The PL design is loaded post Linux boot, therefore this step generates a DT overlay. The overlay DT is slightly different than the Linux boot DT - it must define "fragments" that are added dynamically by Linux at runtime.

The dts/dtsi files can be generated in a number of ways, all of which require the HW description data captured in the XSA or bit file. After a .dtsi file is generated, it is then compiled into a binary .dtbo file. The .dtbo file is expected in firmware folder for each applications.

Here are the three recommended ways:

1. In Xilinx Software Command-Line Tools (XSCT), use Device Tree Generator (DTG) and .xsa file to generate .dtsi, and Device Tree Compiler (DTC) to compile a .dtbo file
2. Manually create .dtsi file, and in PetaLinux, use fpgamanger_custom bbclass to create .dtbo
3. In PetaLinux, use fpgamanger_dtg bbclass tools and petalinux-build to generate .dtsi file from .xsa file, and compiling it into .dtbo.

11.1 Using XSTC, DTG and DTC

11.1.1 Tools and Input Required:

1. XSTC (should be part of Vivado or Vitis installation)
2. DTG, make sure to check out the version that is aligned to rest of tool chain and BSP used:

```
git clone https://github.com/Xilinx/device-tree-xlnx
cd device-tree-xlnx
git checkout xlnx_rel_v2021.1
```

1. DTC:

```
git clone https://git.kernel.org/pub/scm/utils/dtc/dtc.git
cd dtc
make
export PATH=$PATH:/<path-to-dtc>/dtc
```

More information about using Xilinx's Device Tree Generator (DTG) and open source Device Tree Compiler (DTC) can be found in wiki page.

The following hardware design hand-off artifacts are required:

1. XSA file - applies to Vivado or Vitis designs

11.1.2 Generate .dtsi from .xsa using DTG

Use XSCT to call HSI & generate DTSI.

```
hsi open_hw_design <design_name.xsa>
hsi set_repo_path <path to device-tree-xlnx repository>
hsi create_sw_design device-tree -os device_tree -proc psu_cortexa53_0
hsi set_property CONFIG.dt_overlay true [hsi::get_os]
hsi generate_target -dir <desired_dts_filename>
hsi close_hw_design [current_hw_design]
```

[current_hw_design] name can be found in output of hsi open_hw_design <design_name.xsa>.

A folder <desired_dts_filename> will be created, pl.dtsi is the overlay .dtsi file to be used to compile into .dtbo file.

11.1.3 Compile the .dtsi to .dtbo using DTC

This step takes the human readable defined Linux HW description (dtsi file) generated (.pl.dtsi)and compiles it into a binary form that Linux can directly use. This is completed using the Linux Device Tree Compiler (DTC) which is an opensource tool. The actual command used to generate the desired dtbo from the dtsi file is shown below.

```
dtc -@ -O dtb -o pl.dtbo pl.dtsi
```

Rename the pl.dtbo to the appropiiate name.

11.2 Using fpgamanger_custom bbclass in PetaLinux

The `fpgamanager_custom` bitbake class is a helper class to generate a set of FPGA firmware binaries. This method requires that user hand write their own .dtsi file.

11.2.1 Tools and Input required

1. Petalinux of the appropriate version
2. SOM BSP of the appropriate release

The following hardware design hand-off artifacts are required:

1. PL bitstream - applies to Vivado or Vitis designs
2. Device tree overlay source file - the user needs to create this file based on the PL hardware design
3. json file - specifies if the overlay is slotted or flat and required by dfx-mgr. More information can be found here
4. Xclbin file - only applies to Vitis designs

11.2.2 Generate .dtbo file

Please refer to kv260-firmware for example dtsi files based on the `fpgamanager_custom` class used in the Xilinx accelerated applications.

First, run the following command to create a new Petalinux project from the provided K26 SOM bsp file:

```
petalinux-create -t project -s xilinx-k26-starterkit-2021.1-final.bsp
cd xilinx-k26-starterkit-2021.1
```

The following steps assume you have created a petalinux project and built it. If the project has not been build, run this command to configure the project:

```
petalinux-config --silentconfig
```

Run the following command to generate a new firmware recipe:

```
petalinux-create -t apps --template fpgamanager -n user-firmware --enable --srcuri
 "user.bit user.dtsi user.xclbin shell.json"
```

The generated recipe will be located at `project-spec/meta-user/recipes-apps/user-firmware/user-firmware.bb`

The recipe contains the minimum required elements but can be further customized by the user for their needs.

Then the .dtbo file will be generated when building petalinux again.

```
petalinux-build
```

The newly generated .dtbo file can be found at `$tmp_folder/sysroots-components/zynqmp_generic/user-firmware/lib/firmware/xilinx/user-f`
$tmp_folder location can be found at `project-spec/configs/config`
`CONFIG_TMP_DIR_LOCATION=$tmp_folder`

11.3 Using fpgamanger_dtg bbclass in PetaLinux

Alternatively, you can use the `fpgamanager_dtg` bitbake class which uses the Xilinx device tree generator (dtg) to generate a device tree overlay from a Vivado or Vitis-generated XSA file.

11.3.1 Tools and Input required

1. Petalinux of the appropriate version
2. SOM BSP of the appropriate release

The following hardware design hand-off artifacts are required:

1. XSA file (must include bitstream) - applies to Vivado or Vitis designs
2. json file - specifies if the overlay is slotted or flat and required by dfx-mgr. More information can be found here
3. Xclbin file - only applies to Vitis designs
4. dtsi file is not required and will be generated, but the user can optionally add a device tree source file that will be appended to the dtg-generated device tree file

11.3.2 Generate .dtbo file

First, run the following command to create a new PetaLinux project from the provided K26 SOM bsp file:

```
petalinux-create -t project -s xilinx-k26-starterkit-2021.1-final.bsp
cd xilinx-k26-starterkit-2021.1
```

The following steps assume you have created a petalinux project and built it. If the project has not been build, run this command to configure the project:

```
petalinux-config --silentconfig
```

Run the following command to generate a new firmware recipe:

```
petalinux-create -t apps --template fpgamanager_dtg -n user-firmware --enable
--srcuri "user.xsa user.xclbin shell.json"
```

The generated recipe will be located at `project-spec/meta-user/recipes-apps/user-firmware/user-firmware.bb`

The recipe contains the minimum required elements but can be further customized by the user for their needs. If you want to inspect the generated .dtsi file without petalinux/yocto cleaning things up after a successful build, add this variable into your recipe: `RM_WORK_EXCLUDE += "${PN}"` The .dtsi file can be found in `<tmpworkspace>/work/zynqmp_generic-xilinx-linux/user-firmware/1.0-r0/build/user-firmwar`

Then the .dtbo file will be generated when building PetaLinux again.

```
petalinux-build
```

The newly generated .dtbo file can be found at `$tmp_folder/sysroots-components/zynqmp_generic/user-firmware/lib/firmware/xilinx/user-f`
$tmp_folder location can be found at `project-spec/configs/config` `CONFIG_TMP_DIR_LOCATION=$tmp_folder`

11.4 License

On-target Utilities and Firmware

After generating the PL design, developers will need to move the required files to target platform. Developers can either scp or ftp the required files to SOM, or load them into SD cards and find them in /media/sd-mmcblk1p1/. Below file structures are some of the files on target developers may need to touch to test and deploy their applications.

```
|-- etc
|   |-- dfx-mgrd                    dfx-manager daemon configuration files,
including daemon.config
|   |   |-- daemon.config           Configuration file to indicate locations of
firmware for dfx-mgr
|   |-- vart.conf                   Vitis AI configuration file links
|-- lib                             Libraries
|   |-- firmware                    Misc firmware
|   |   |-- xilinx                  Location to put user application accelerator
firmware
|   |   |-- <other>                 User defined location to put user application
 accelerator firmware, need to modify daemon.config to include new location
|   |-- modules                     Contains loadable kernel modules(.ko files)
for drivers
|-- opt
|   |-- xilinx                      Folder only exist after a dnf install of a
package
|   |   |-- bin                     Application specific binaries
|   |   |-- lib                     Libraries for applications
|   |   |-- share
|   |   |   |-- ivas                IVAS.json files for applications
|   |   |   |-- notebooks           Notebooks for applications
|   |   |   |-- vitis_ai_library    VAI model files for applications
```

The SOM Starter Kit Linux includes a set of utilities to help manage dynamic deployment, loading, and configuration of accelerated applications. The xmutil application is a front-end wrapper that provides a common user experience for interacting with different platforms. The dfx-mgr utility is called by xmutil, or can be called directly for identifying, loading, and unloading multiple PL application bitstreams on target.

12.1 xmutil

xmutil is a Python-based utility wrapper that provides a front-end to other standard Linux utilities (e.g., dnf) or Xilinx platform specific sub-utilities (e.g., platform stats). In the context of dynamic deployment xmutil provides the functionality to read the package feeds defined by the on-target *.repo file and down select the package-groups based on the hardware platform name read from the SOM and carrier card (CC) EEPROM contents. For example, when calling xmutil getpkgs on the KV260 starter kit, the utility will query the package feed and then only present to you the package-groups that include the string kv260 in them. This is intended to help you quickly identify the

accelerated applications related packages for your platform. You can also use standard dnf calls to interact with the package-feed.

12.2 dfx-mgr

The dfx-mgr is a Xilinx library that implements an on-target daemon for managing a data model of on-target applications, active PL configuration, and loading/unloading the corresponding bitstreams. The dfx-mgr daemon(dfx-mgrd) maintains a data model of relevant application bitstreams, bitstream types, and active bitstream. The xmutil application is calling dfx-mgr when making the calls loadapp, unloadapp, and listapps. dfx-mgr is capable of supporting flat and hierarchical (DFX) PL designs.

The dfx-mgr requires that the application bitstreams be loaded in /lib/firmware/<company_name>/<app_name>. Up until 2021.2, xilinx is the only folder supported. In 2022.1, other folders will be supported as long as they have been added in daemon.config. For latest details, visit its github page.

The dfx-mgr uses i-notify to identify when new applications are brought into the system, and requires that the files required for an application be loading in the same <app_name>. The app_name directory must contain:

- Application bitstream converted to *.bit.bin format
- Application bitstream device tree overlay *.dtbo
- If its a Vitis based PL design using XRT - metadata file in .xclbin format
- shell.json with metadata about the PL design

Example files for dfx-mgr can be found in github.

12.3 shell.json file

The shell.json file is a metadata file for dfx-mgr. Current implementation (2021.1) only recognizes XRT_FLAT but it is being expanded to support: XRT_FLAT, XRT_DFX, PL_FLAT, PL_DFX, and AIE types in future release. For now, use XRT_FLAT. The shell.json only needs following content:

```
{
    "shell_type" : "XRT_FLAT",
    "num_slots" : "1"
}
```

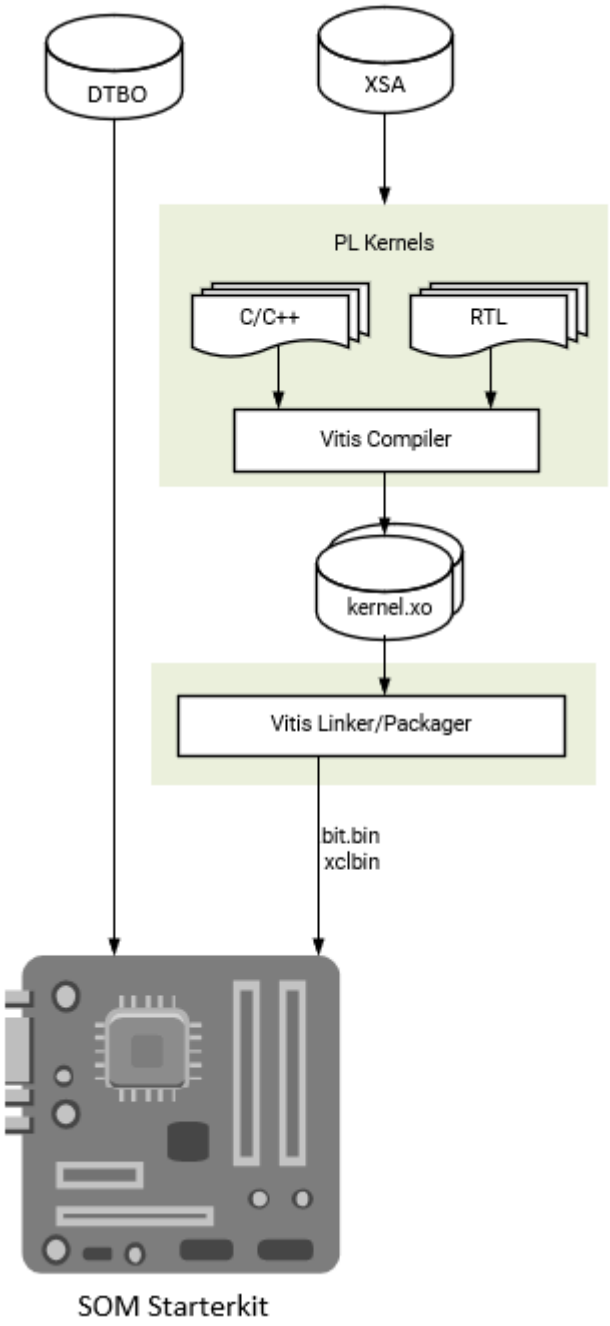
12.4 License

Vitis Accelerator Flow

This flow is for developers using Xilinx provided SOM Starter Kit Vitis Platforms as a basis for generating their own PL accelerators. Developers use a Vivado Extensible Platform (.xsa) file provided by Xilinx and import it into a Vitis Platform project. Developers then create their own overlay accelerator(s) within the bounds of the provided Vitis Platform, and generate a new bitstream (.bit file converted to .bit.bin) and metadata container file (.xclbin). Developers can use the existing device tree blob (.dtb) associated with the Xilinx provided Vitis platform. The resulting application accelerator files are then moved to the target SOM platform to be run.

To access the .xsa files for different platforms released from Xilinx, refer to Creating a Vitis Platform tutorial to generate .xsa files from released reference design.

- Constraints: developers must use the same carrier card and physical interface definition as Xilinx provided SOM starter kits and associated Vitis platforms.
- Input: Xilinx provided Vitis platform (.xsa), Xilinx provided Vitis platform device tree (.dtbo)
- Output: .bit, .xclbin



13.1 Prerequisites and Assumptions

This document assume that developers will use 2021.1 or later for their tools and SOM releases. The tool versions should match - e.g. use the same tool versions for PetaLinux, Vivado, and the released BSP.

1. Vitis tools installation
2. PetaLinux tool installation
3. PetaLinux SOM StarterKit BSP download

13.2 Step 1 - Aligning Kria SOM boot & SOM Starter Linux infrastructure

Xilinx built Kria SOM Starter Kit applications on a shared, application-agnostic infrastructure in the SOM Starter Linux including kernel version, Yocto project dependent libraries, and baseline BSP. When using this tutorial, make sure to align tools, git repositories, and BSP released versions.

13.2.1 PetaLinux BSP Alignment

The SOM Starter Linux image is generated using the corresponding SOM variant multi-carrier card PetaLinux board support package (BSP). Developers creating applications on the Starter Kit are recommended to use this BSP as a baseline for their application development as it ensures kernel, Yocto project libraries, and baseline configuration alignment. The multi-carrier card BSP defines a minimalistic BSP that has the primary function of providing an application-agnostic operating system, and can be updated and configured dynamically at runtime.

13.3 Step 2 - Obtain .xsa and .dtbo files

Developers will need to first decide on which Kria Starter Kit Vitis platform to develop on. The list of platforms can be found in Creating a Vitis Platform, which also contains a tutorial to generate .xsa files from the released reference designs.

The Vitis Platform repository has the device tree(DT) source associated with the platform captured as a .dtsi file. User can generate the .dtbo from this source using PetaLinux or it can copy a compiled binary from a corresponding pre-built platform application.

To obtain the corresponding .dtbo file, developers can copy the .dtbo files from target from applications sharing the same platform. They can be located on target, post dnf install in `/lib/firmware/xilinx/<application name>/<application name>.dtbo` An example is kv260-smartcam.

Alternatively, developers can generate them in Petalinux.

```
petalinux-create -t project -s xilinx-k26-starterkit-v2021.1-final.bsp
cd xilinx-k26-starterkit-2021.1
petalinux-build -c <application name>
```

The .dtbo file can be found in

```
$temp_folder/sysroots-components/k26/<application
name>/lib/firmware/xilinx/<application name>/<application name>.dtbo
```

location of $temp_folder can be found in `xilinx-k26-starterkit-2021.1/project-spec/configs/config`

`CONFIG_TMP_DIR_LOCATION="$tmp_folder"`

13.4 Step 3 - Create .xclbin and .bit.bin file from Vitis

Typically in this work flow, developers will use Makefiles to generate their design. Example Makefiles can be found here in kv260-vitis repo and here in Vitis Library repo.

After a successful make, there should be a .xclbin and .bit file generated. To generate a .bit.bin file, create a `<accelerator>.bif` file with the following content:

```
all:
```

```
{
    <accelerator>.bit
}
```

And then run bitgen to create a binary bitstream `<accelerator>.bit.bin` file:

```
bootgen -arch zynqmp -process_bitstream bin -image <accelerator>.bif
```

13.5 Step 4 - Move user application to the target platform

After generating the PL design, developers will need to move the required files (.bit.bin, .dtbo, cred.json, and .xclbin for Vitis flow) to target platform using standard tools (e.g. SCP, FTP). Please see On-target Utilities and Firmware for where to place the application firmware files.

13.6 Step 5 - Run the user application

Once the required files are in place, developers can run their applications using the following steps:

- Use xmutil or dfx-mgr to load the application bitstream
- Start their application software

13.6.1 License

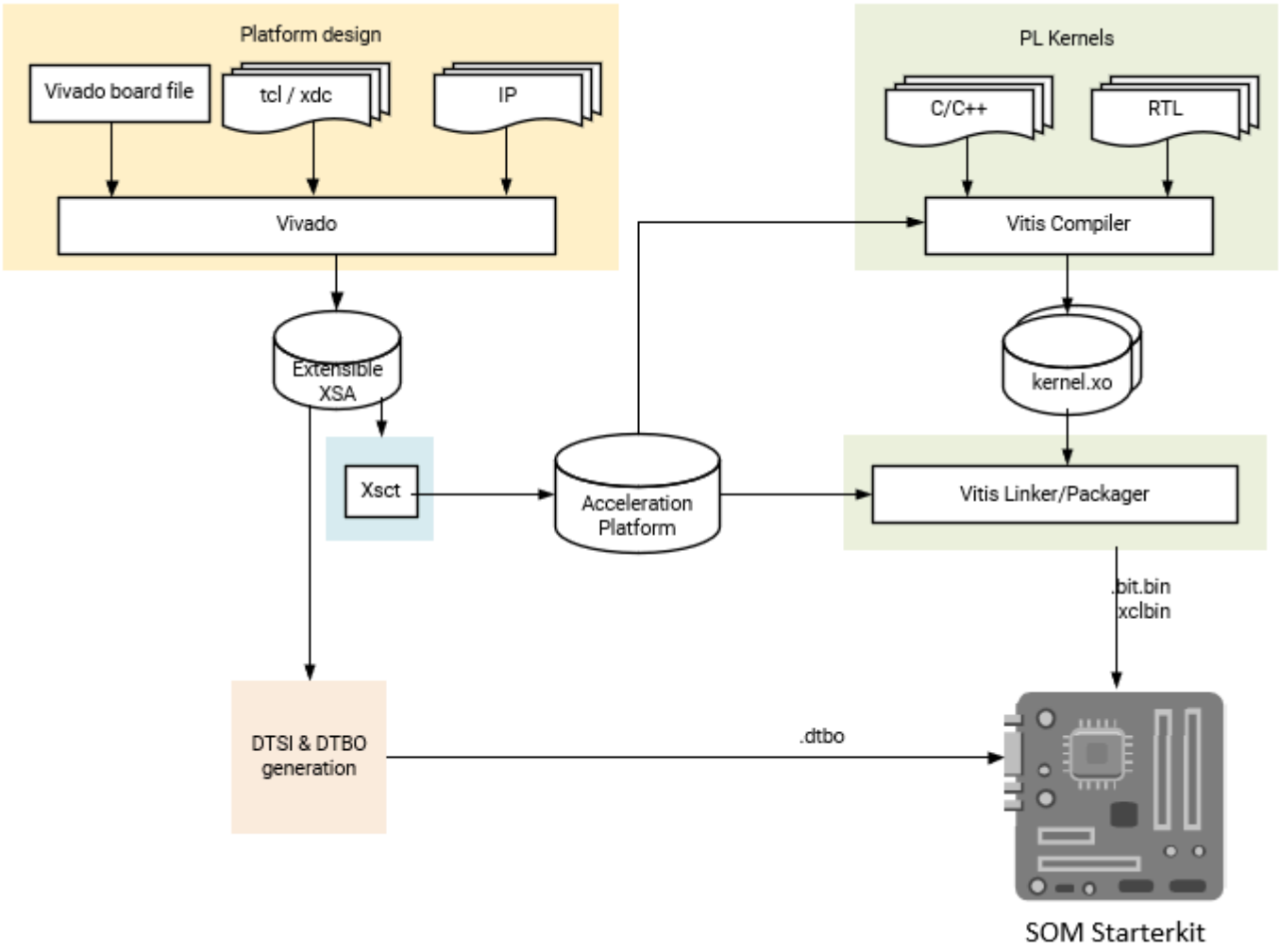
Chapter 14

Vitis Platform Flow

Developers can create a custom Vitis platform if they require a different set of physical PL I/O peripherals than those provided in Xilinx generated platforms. Development starts with the Vivado tool to create an extensible hardware platform. In Vivado, the Kria SOM Starter Kit Vivado board files are provided. It automatically drives the PS subsystem HW configuration and provides pre-defined connectivity for commonly used PL IPs based on the selected carrier card (e.g. MIPI interfaces on KV260 carrier card). Developers use Vivado to generate a custom .xsa file to be ported into Vitis as a platform project. Once the platform project is created, then a corresponding device tree overlay is generated. With the extensible .xsa and .dtbo developers can now follow the same flow outlined in Vitis Accelerator flow. The resulting bitstream, .xclbin, and .dtbo files are copied into the target.

- Assumption: Xilinx provided SOM carrier card with associated Vivado board file automation
- Input: Vivado SOM Starter Kit board files
- Output: .dtbo, .bit.bin, .xlcbin

This Vitis guide is a good detailed tutorial for Vitis Platform Flow.


Platform design
Vivado board file
tcl / xdc
IP
Vivado
PL Kernels
C/C++
RTL
Vitis Compiler
Extensible XSA
kernel.xo
Xsct
Acceleration Platform
Vitis Linker/Packager
.bit.bin
.xclbin
DTSI & DTBO generation
.dtbo
SOM Starterkit

Prerequisites and Assumptions

This document assume that developers will use 2021.1 or later for their tools and SOM releases. The tool versions should match - e.g. use the same tool versions for PetaLinux, Vivado, and the released BSP.

1. Vitis tools installation
2. PetaLinux SOM StarterKit BSP download
3. Vivado tools installation
4. tool for generating and/or compiling .dtbo file: a. PetaLinux tools installation or b. XSCT (will be installed as part of Vivado or Vitis)

Step 1 - Aligning Kria SOM boot & SOM Starter Linux infrastructure

Xilinx built Kria SOM Starter Kit applications on a shared, application-agnostic infrastructure in the SOM Starter Linux including kernel version, Yocto project dependent libraries, and baseline BSP. When using this tutorial, make sure to align tools, git repositories, and BSP released versions.

16.1 PetaLinux BSP Alignment

The SOM Starter Linux image is generated using the corresponding SOM variant multi-carrier card PetaLinux board support package (BSP). Developers creating applications on the Starter Kit are recommended to use this BSP as a baseline for their application development as it ensures kernel, Yocto project libraries, and baseline configuration alignment. The multi-carrier card BSP defines a minimalistic BSP that has the primary function of providing an application-agnostic operating system, and can be updated and configured dynamically at runtime.

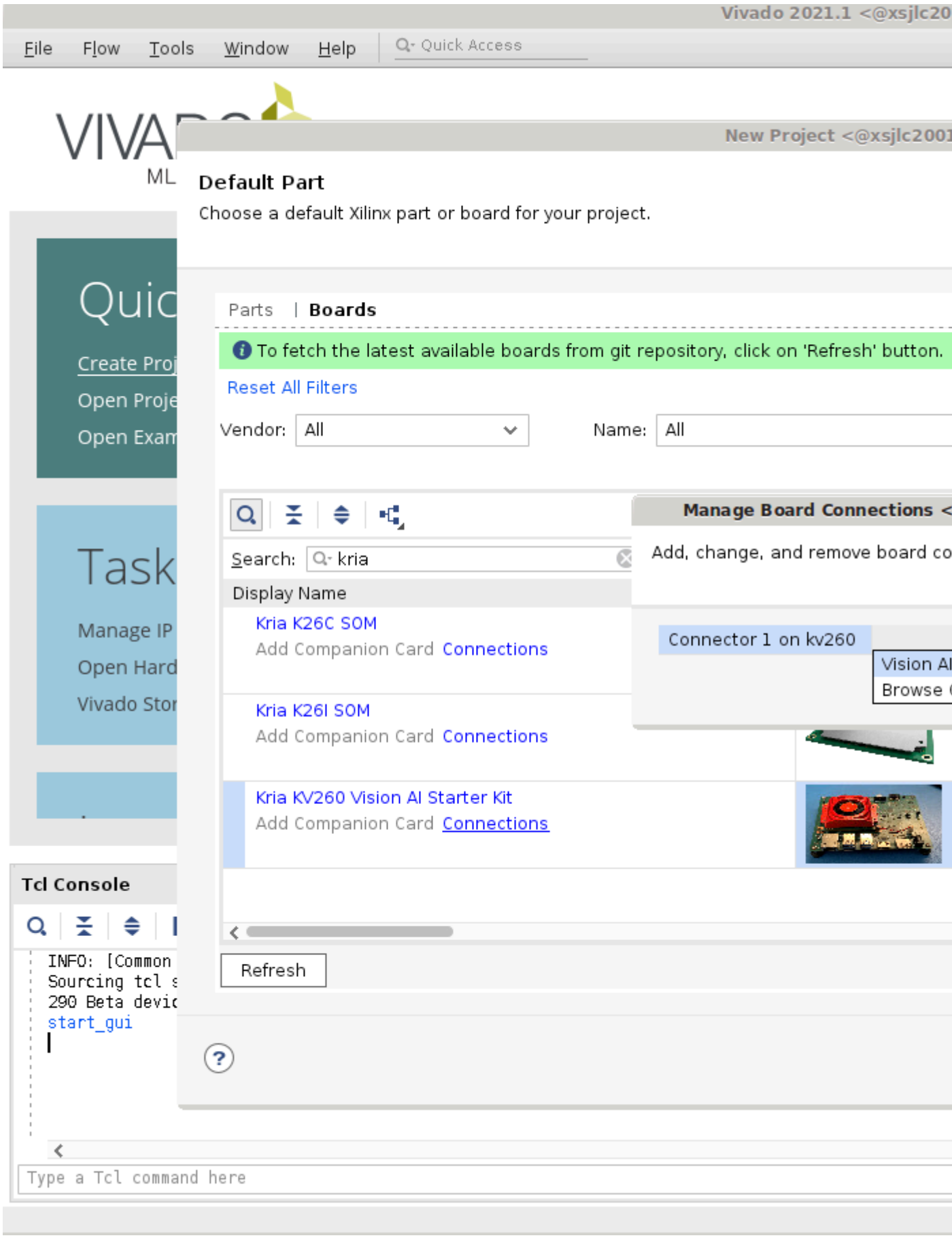
Step 2 - Generate a new custom PL design using Vivado

There are two ways to get started designing PL design for SOM Carrier cards. Developers can either start with the Vivado board file, or the released Vivado starter project in BSP.

17.1 Vivado board file

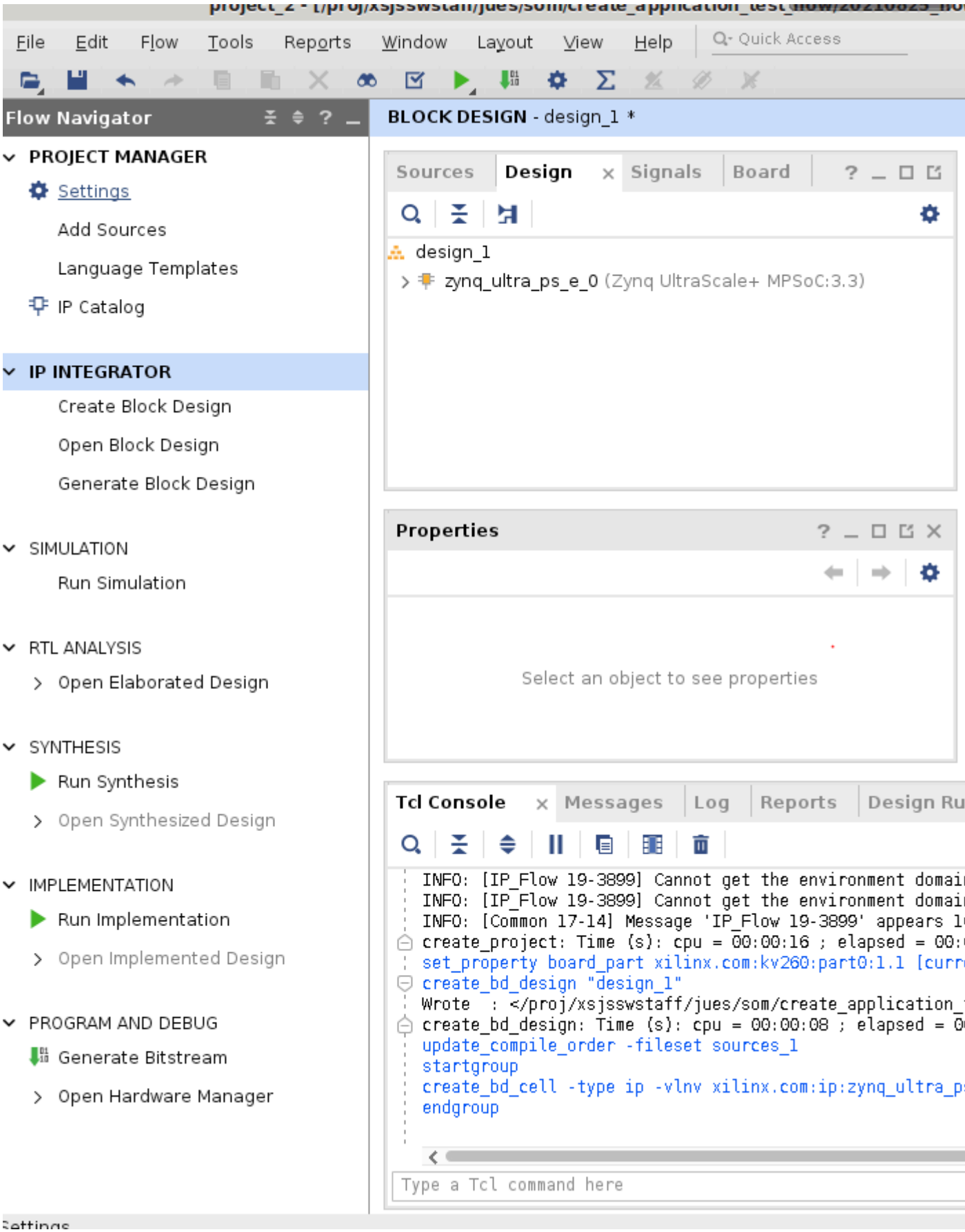
This flows starts with Vivado board files containing information on K26 and KV260 CC. The K26 SOM is supported in Vivado with three board files that automate the configuration of the SOM based peripherals. These board files are available in Vivado's board list in "Create Project" wizard in 2021.1 or later.

- K26C SOM - Commercial grade K26 SOM.
- K26I SOM - Industrial grade K26 SOM.
- KV260 Starter Kit - K26 based Starter Kit SOM with Vision AI Carrier Card connector interface.


Vivado 2021.1 <@xsjlc20
File
Flow
Tools
Window
Help
Quick Access
New Project <@xsjlc2001
Default Part
Choose a default Xilinx part or board for your project.
Parts
Boards
To fetch the latest available boards from git repository, click on 'Refresh' button.
Reset All Filters
Vendor:
All
Name:
All
Manage Board Connections
Add, change, and remove board co
Search:
kria
Display Name
Kria K26C SOM
Add Companion Card
Connections
Connector 1 on kv260
Vision AI
Browse
Kria K26I SOM
Add Companion Card
Connections
Kria KV260 Vision AI Starter Kit
Add Companion Card
Connections
Refresh
Tcl Console
INFO: [Common
Sourcing tcl s
290 Beta devic
start_gui
Type a Tcl command here

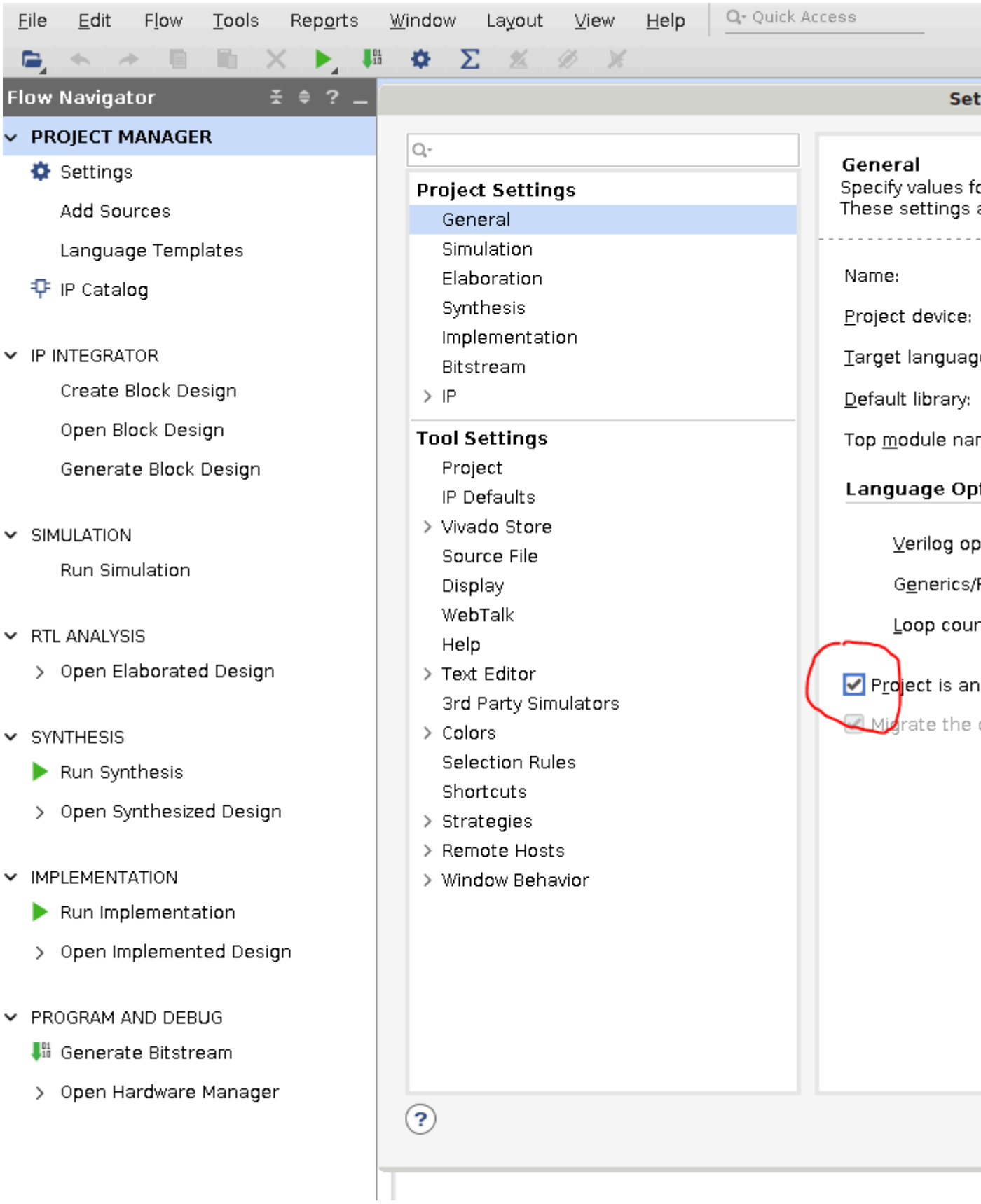
When selecting the Kria starter kit board file, make sure to click on “connections” to indicate that the K26 and KV carrier card are connected.

Once Zynq_ultra_ps_e_0 block is added to a design, make sure to click “Run Block Automation” to apply board file settings.



Developers then indicate that the platform is an Extensible Vitis Platform. More details on how to

create Extensible Platform can be found here. Project Manager -> Settings -> General -> check "Project is an extensible Vitis Platform"



17.2 Vivado Starter Project in BSP

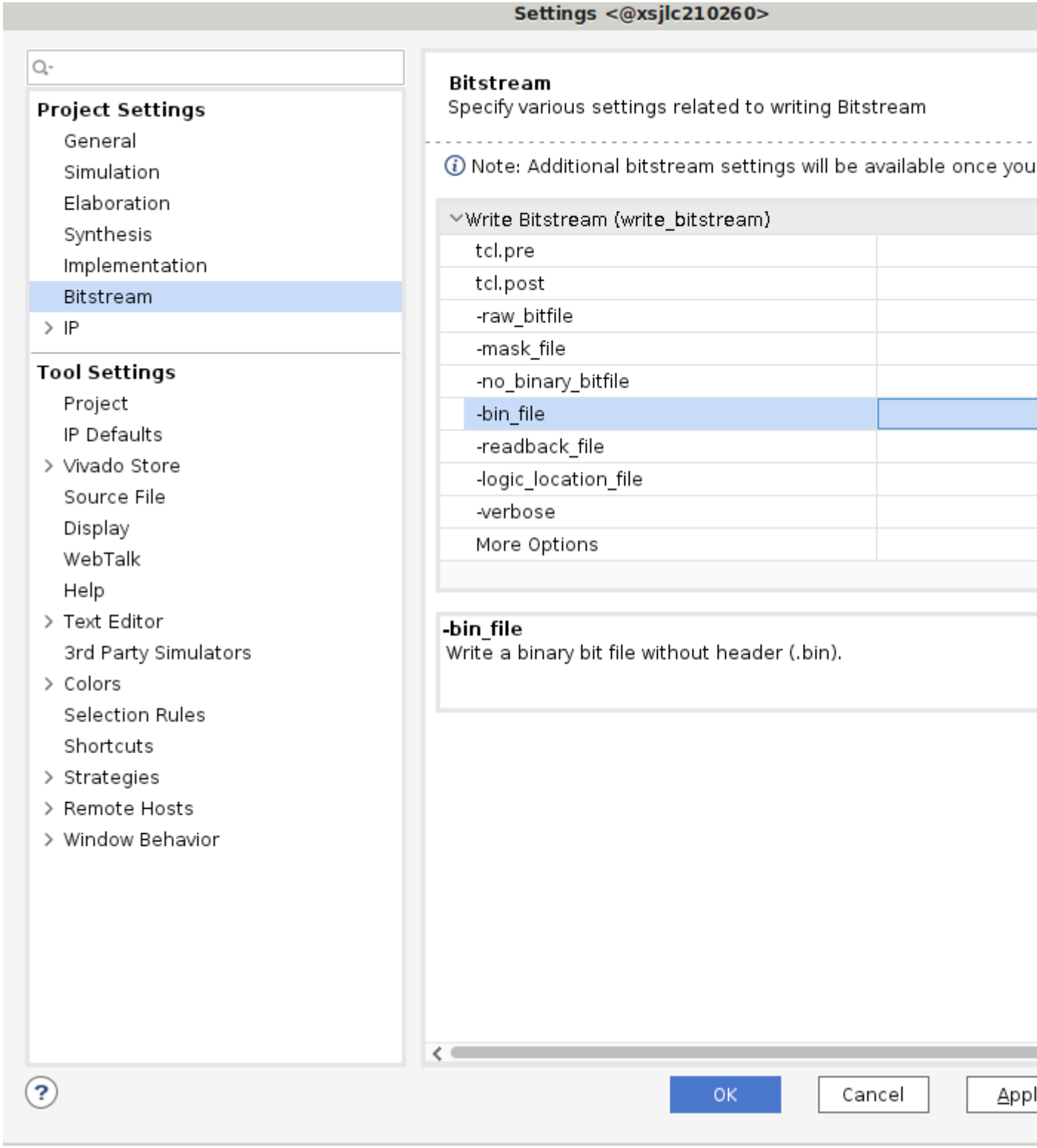
Alternatively, developers can start from the Vivado project provided in the BSP file. First, download the SOM Starter Kit BSP from the SOM Wiki Then create the project using BSP:

```
petalinux-create -t project -s xilinx-k26-starterkit-v2021.1-final.bsp
cd xilinx-k26-starterkit-2021.1
```

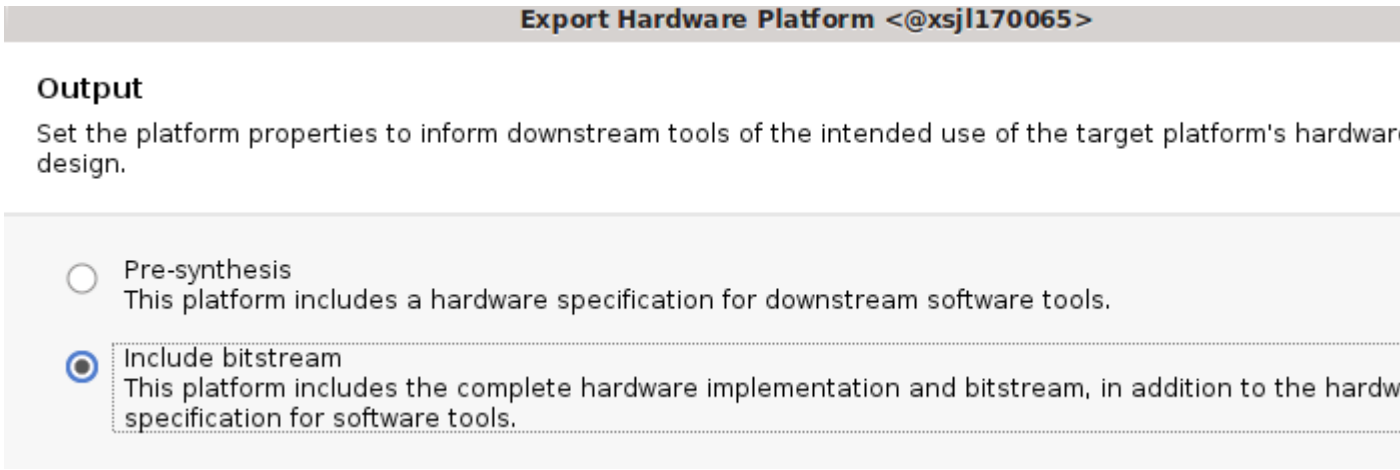
The Vivado starter project can be found in `hardware/` folder, and developers can open the project using the .xpr file. This project is a k26 project only, and will not contain any information about the carrier card being used. However, it does have enough information to boot basic Linux.

17.3 Generate .bit.bin and .xsa file

Please refer to Vivado documentation to add custom IP blocks into your design and generate a .xsa file and a binary bitstream (.bin file converted .bit.bin file). In order to generate a binary bitstream (.bin) file, go to Tools -> setting and enable -bin_file, and rename the generated .bin file to .bit.bin.



An updated .xsa file needs to be generated by using File -> Export -> Export Hardware , make sure to select “include bitstream” in the generation.



To access the example kv260 Vitis reference design, developers can follow the steps in the Using Vivado to Build the Hardware Design tutorial.

Step 3 - Compile a device tree overlay blob (.dtbo) using Petalinux

If using PL loading post Linux boot, then a DT overlay is required to add the HW/SW interfaces to the initial Linux booted device tree. For creating the DT overlay please refer to dtsi_dtbo_generation page.

step 4 - follow Vitis Accelerator Flow to generate applications

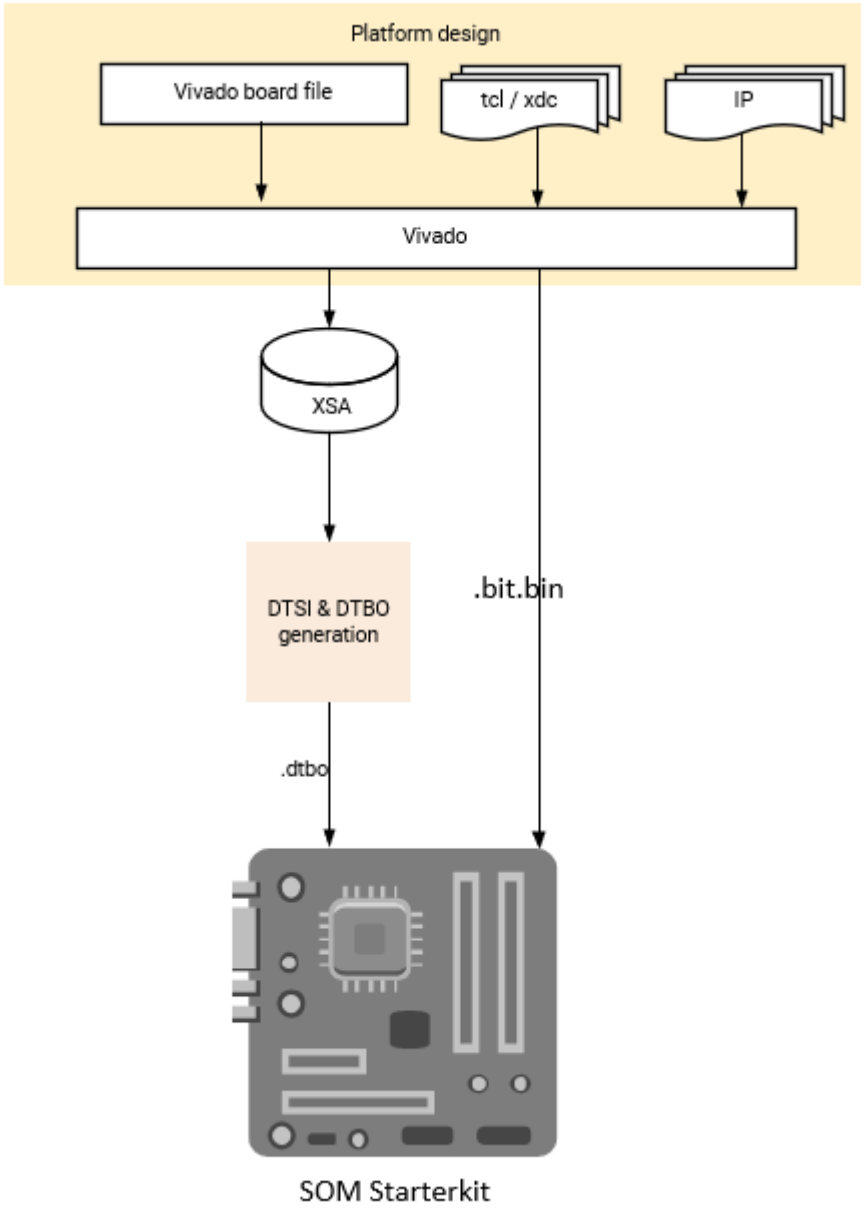
After creating .bit.bin and .xsa file, developers will be able to create their accelerators in Vitis platform. They can follow step 3 through 5 in Vitis Accelerator Flow to finish creating and running their applications.

19.1 License

Vivado Accelerator Flow

Developers prefer a traditional HW design flow can generate their PL designs using Vivado. In this flow developers start from the Kria SOM starter kit board files in Vivado and implements their own PL design in Vivado to generate a .xsa file and bitstream. The resulting .xsa file is used to generate the device tree overlay. Once the PL design (.bit.bin) and HW/SW interface definition (.dtbo) files are created, they can be copied into the target and managed by dfx-mgr.

- Assumption: Xilinx built carrier cards with corresponding SOM Starter Kit board file
- Input: SOM Starter Kit board files (in Vivado) or Vivado project released with SOM BSP, developer's own accelerator designs in Vivado
- Output: .dtbo, .bit



20.1 Prerequisites and Assumptions

This document assumes that developers will use 2021.1 or later tools and SOM content releases. The tool versions should match - e.g. use the same tool versions for PetaLinux, Vivado, and the released BSP.

1. Vivado tools installation
2. Tool for generating and/or compiling .dtbo file:
   a. PetaLinux tools installation or
   b. XSCT (will be installed as part of Vivado or Vitis)
3. PetaLinux SOM StarterKit BSP download

20.2 Step 1 - Aligning Kria SOM boot & SOM Starter Linux infra-

structure

Xilinx built Kria SOM Starter Kit applications on a shared, application-agnostic infrastructure in the SOM Starter Linux including kernel version, Yocto project dependent libraries, and baseline BSP. When using this tutorial, make sure to align tools, git repositories, and BSP released versions.

20.2.1 PetaLinux BSP Alignment

The SOM Starter Linux image is generated using the corresponding SOM variant multi-carrier card PetaLinux board support package (BSP). Developers creating applications on the Starter Kit are recommended to use this BSP as a baseline for their application development as it ensures kernel, Yocto project libraries, and baseline configuration alignment. The multi-carrier card BSP defines a minimalistic BSP that has the primary function of providing an application-agnostic operating system, and can be updated and configured dynamically at runtime.

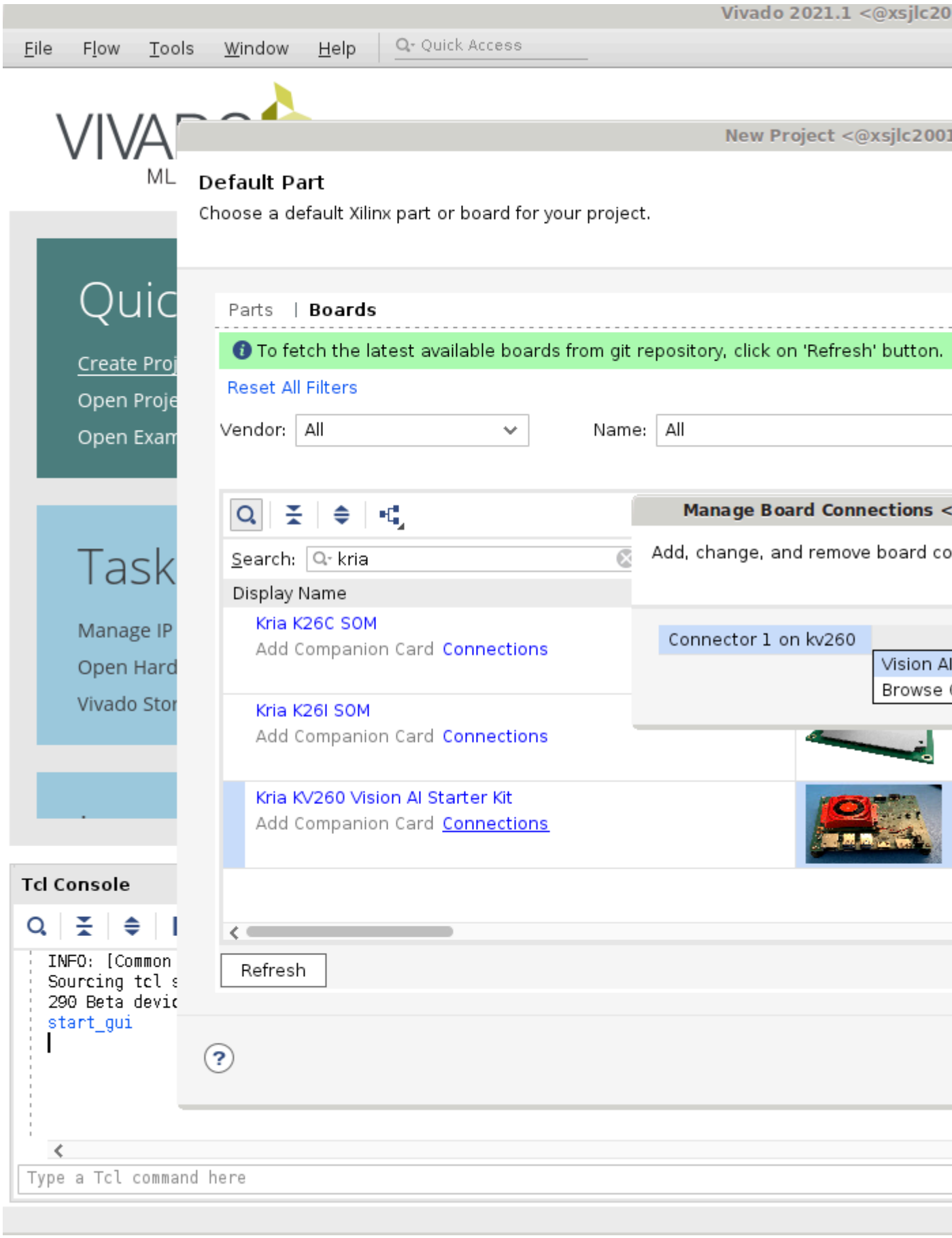
20.3 Step 2 - Generate a new custom PL design using Vivado

There are two ways to get started designing PL design for SOM Carrier cards. Developers can either start with the Vivado board file, or the released Vivado starter project in BSP.

20.3.1 Vivado board file

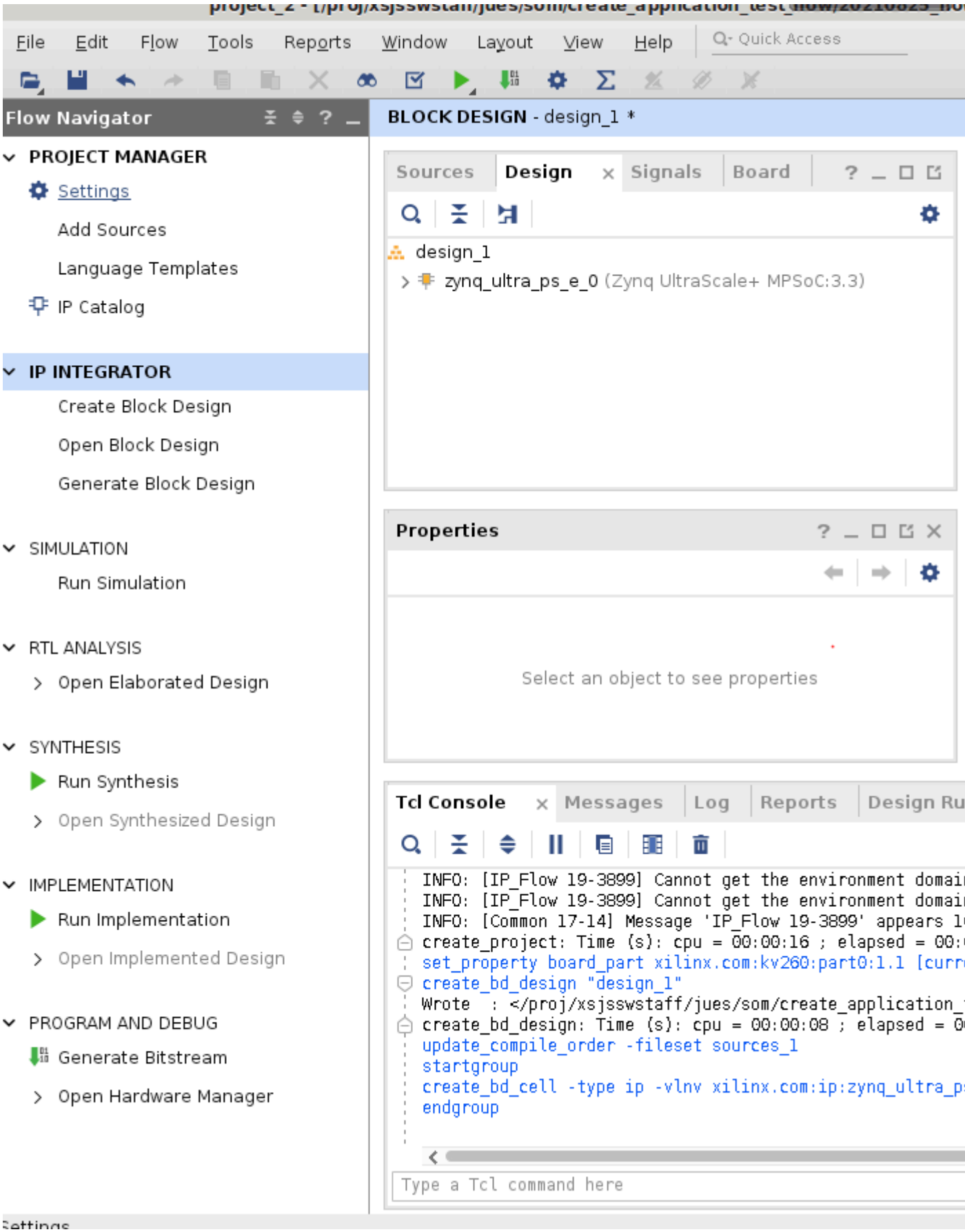
This flow starts with Vivado board files containing information on K26 and KV260 CC. The K26 SOM is supported in Vivado with three board files that automate the configuration of the SOM based peripherals. These board files are available in Vivado's board list in "Create Project" wizard in 2021.1 or later. For 2020.2.2, its available in XHUB Stores by clicking "Install/Update Boards" in "Create Project" wizard of Vivado.

- K26C SOM - Commercial grade K26 SOM.
- K26I SOM - Industrial grade K26 SOM.
- KV260 Starter Kit - K26 based Starter Kit SOM with Vision AI Carrier Card connector interface.


Vivado 2021.1 <@xsjlc20
File
Flow
Tools
Window
Help
Quick Access
New Project <@xsjlc2001
Default Part
Choose a default Xilinx part or board for your project.
Parts
Boards
To fetch the latest available boards from git repository, click on 'Refresh' button.
Reset All Filters
Vendor:
All
Name:
All
Manage Board Connections
Add, change, and remove board co
Search:
kria
Display Name
Kria K26C SOM
Add Companion Card
Connections
Connector 1 on kv260
Vision AI
Browse
Kria K26I SOM
Add Companion Card
Connections
Kria KV260 Vision AI Starter Kit
Add Companion Card
Connections
Refresh
Tcl Console
INFO: [Common
Sourcing tcl s
290 Beta devic
start_gui
Type a Tcl command here

When selecting the Kria starter kit board file, make sure to click on “connections” to indicate that the K26 and KV carrier card are connected.

Once Zynq_ultra_ps_e_0 block is added to a design, make sure to click “Run Block Automation” to apply board file settings.


File Edit Flow Tools Reports Window Layout View Help
Quick Access
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
BLOCK DESIGN - design_1 *
Sources
Design
Signals
Board
design_1
zynq_ultra_ps_e_0 (Zynq UltraScale+ MPSoC:3.3)
Properties
Select an object to see properties
Tcl Console
Messages
Log
Reports
INFO: [IP_Flow 19-3899] Cannot get the environment domai
INFO: [IP_Flow 19-3899] Cannot get the environment domai
INFO: [Common 17-14] Message 'IP_Flow 19-3899' appears l
create_project: Time (s): cpu = 00:00:16 ; elapsed = 00:
set_property board_part xilinx.com:kv260:part0:1.1 [curr
create_bd_design "design_1"
Wrote : </proj/xsjsswstaff/jues/som/create_application_
create_bd_design: Time (s): cpu = 00:00:08 ; elapsed = 0
update_compile_order -fileset sources_1
startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:zynq_ultra_p
endgroup
Type a Tcl command here

20.3.2 Vivado Starter Project in BSP

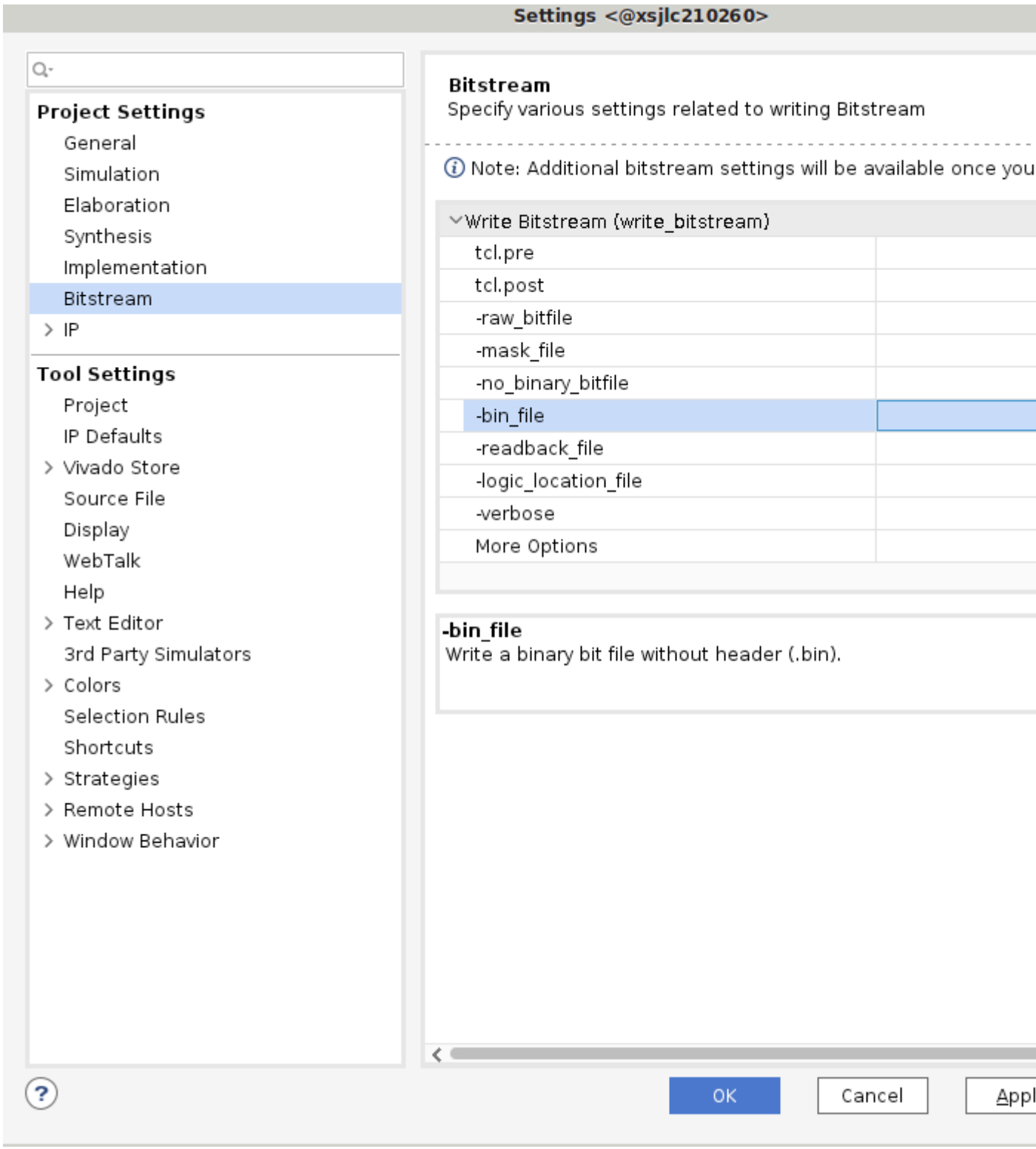
Alternatively, developers can start from the Vivado project provided in the BSP file. First, download the SOM Starter Kit BSP from the SOM Wiki Then create the project using BSP:

```
petalinux-create -t project -s xilinx-k26-starterkit-v2021.1-final.bsp
cd xilinx-k26-starterkit-2021.1
```

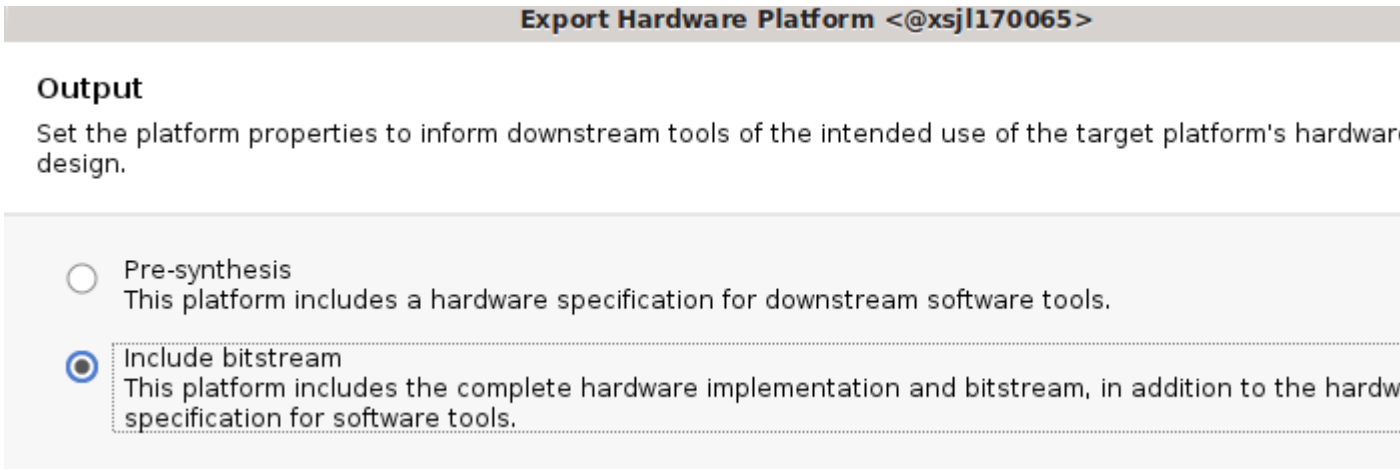
The Vivado starter project can be found in `hardware/` folder, and developers can open the project using the .xpr file. This project is a k26 project only, and will not contain any information about the carrier card being used. However, it does have enough information to boot basic Linux.

20.3.3 Generate .bit.bin and .xsa file

Please refer to Vivado documentation to add custom IP blocks into your design and generate a .xsa file and a binary bitstream (.bin file converted .bit.bin file). In order to generate a binary bitstream (.bin) file, go to Tools -> setting and enable -bin_file, and rename the generated .bin file to .bit.bin.



An updated .xsa file needs to be generated by using File -> Export -> Export Hardware , make sure to select “include bitstream” in the generation.



To access the example kv260 Vitis reference design, developers can follow the steps in the Using Vivado to Build the Hardware Design tutorial.

20.4 Step 3 - Compile a device tree overlay blob (.dtbo) using Petalinux

If using PL loading post Linux boot, then a DT overlay is required to add the HW/SW interfaces to the initial Linux booted device tree. For creating the DT overlay please refer to dtsi_dtbo_generation page.

20.5 Step 4 - Move user application to the target platform

After generating the PL design, developers will need to move the required files (.bit.bin, .dtbo, cred.json, and .xclbin for Vitis flow) to target platform into the proper area. Please see On-target Utilities and Firmware for where to place the application firmware files.

20.6 Step 5 - Run the user application

Once the required files are in place, developers can run their applications. They should

- Use xmutil or dfx-mgr to load the application bitstream
- Start their application software

20.6.1 License