Dolphin Technology

SIP RTL Training

SIP RTL

FPGA Training

Revision 1.0

12 April 2024

FPGA Training

SIP RTL Training

SIP RTL

Revision 1.0

12 April 2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

No table of figures entries found.

REVISION HISTORY

Revision	Date	Description of Changes
1.0	12 Apr 2024	Initial

1. Field Programmable Gate Arrays (FPGAs)

1.1. What is an FPGA?

Field-programmable gate arrays (FPGAs) is a type of configurable integrated circuit that can be programmed or reprogrammed after manufacturing. FPGAs contain configurable logic blocks (CLBs) and a set of programmable interconnects that allow the designer to connect blocks and configure them to perform everything from simple logic gates to complex functions. Full SoC designs containing multiple processes can be put onto a single FPGA device.

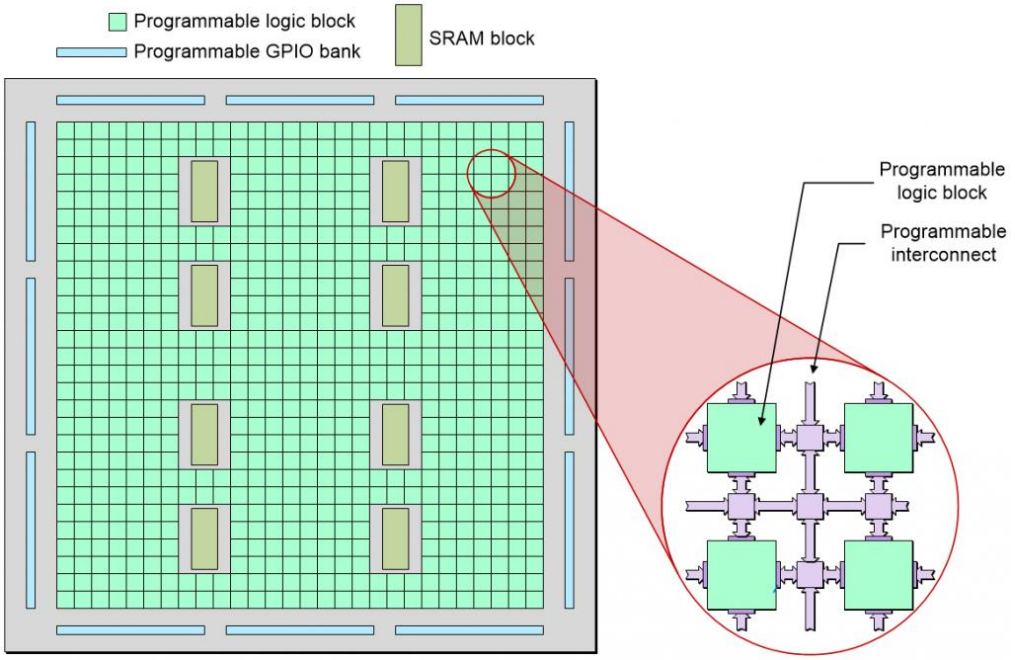


Figure 1. General structure of an FPGA

1.2. FPGA prototyping

FPGA prototyping, also referred to as FPGA-based prototyping, application-specific integrated circuit (ASIC) prototyping or system-on-chip (SoC) prototyping, is the method to prototype system-on-chip and ASIC designs on FPGAs for hardware verification and early software development.

Developers design prototypes on FPGAs to incrementally mature the design before finalizing it at tape-out. FPGAs are often used in commercial applications where there is a need for parallel computing and the requirements are dynamic, such as for telecoms and avionics.

Verification methods for hardware design as well as early software and firmware co-design have become mainstream. Prototyping SoC and ASIC designs with one or more FPGAs and electronic design automation (EDA) software has become a good method to do this.

1.3. Types of FPGAs

There are two main types of FPGAs:

- SRAM-based FPGAs: use static random-access memory (SRAM) cells to store the configuration of the programmable logic gates. These types of FPGAs are volatile, which means that the configuration is lost when power is removed
- Flash-based FPGAs: use non-volatile flash memory cells to store the configuration. These types of FPGAs retain the configuration even when power is removed.

1.4. Configurable Logic Blocks (CLBs) on an FPGA

A configurable logic block (CLB) is the basic repeating logic resource on an FPGA. When linked together by routing resources, the components in CLBs execute complex logic functions, implement memory functions, and synchronize code on the FPGA.

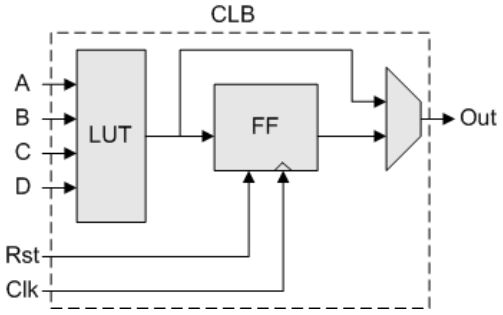


Figure 2. Structure of a CLB on an FPGA

The structure of a CLB on an FPGA is illustrated in Figure 2 with three main components. In which, Look-up Table (LUT) is a collection of gates hardwired on the FPGA. An LUT stores a predefined list of outputs for every combination of inputs. LUTs provide a fast way to retrieve the output of a logic operation because possible results are stored and then referenced rather than calculated.

1.5. Specialize Hardware Blocks

Several types of specialized hardware blocks are available in most current FPGAs, but not all of them are available in all devices and their number varies from one device to another.

1.5.1. Clock Management Blocks

The generation, control, and quality of clock signals are among the most important problems to be faced in the design of complex digital systems, particularly in the case of multirate systems or those requiring very fast data transfer rates, where synchronization among the different parts of the sys- tem is a critical issue.

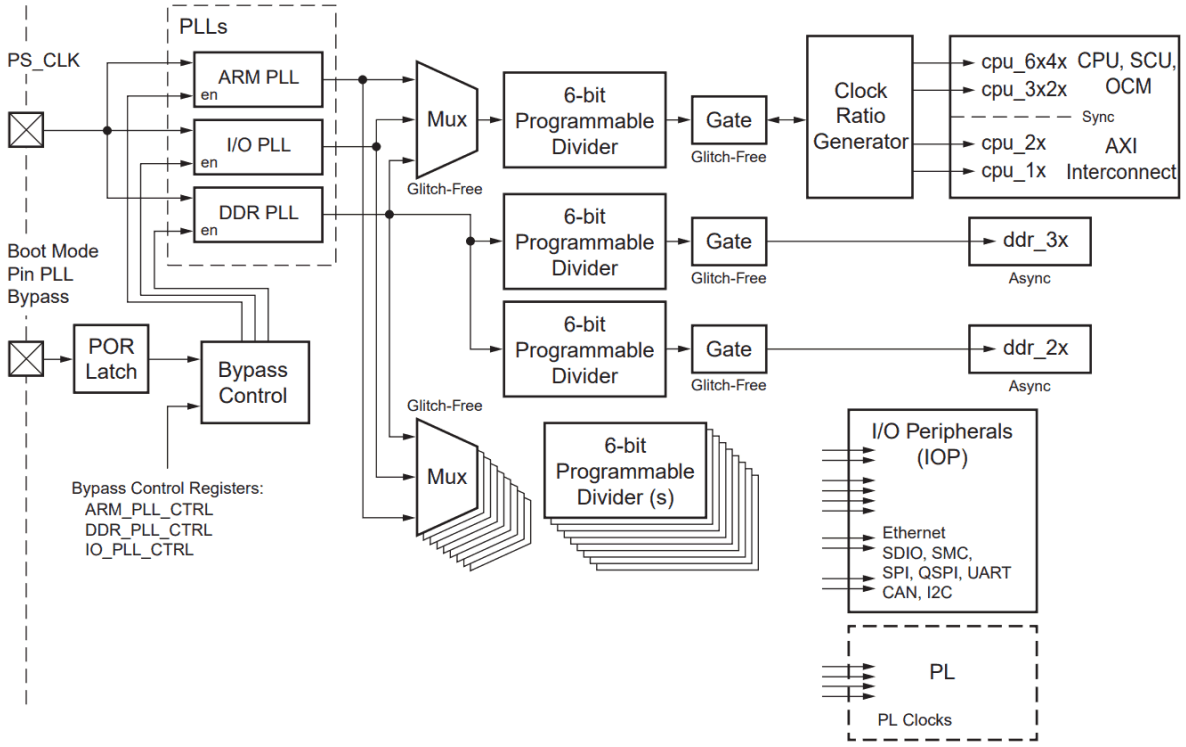


Figure 3. Zynq-7000 Series Clocks

In order to reduce the problems associated with clock signals as well as the number of external oscillators needed, FPGAs include clock management blocks (CMBs), based on either PLLs or delay-locked loops (DLLs). These CMBs are mainly used for frequency synthesis (frequency multiplication or division), skew reduction, and duty cycle/phase control of clock signals.

1.5.2. Memory Blocks

Most digital systems require resources to store significant amounts of data. Memories are the main elements in charge of this task. Since memory access times are usually much longer than propagation delays in logic circuits, memories (in particular external ones) are the bottleneck of many systems in terms of performance. Because of this, FPGA vendors have always paid special attention to optimizing logic resources so that they can support, in the most efficient possible way, the implementation of internal memories.

1.5.2.1. Distributed RAM

Distributed RAM is built with LUTs. LUTs are usually used to create the logic of your design, but can also support memory in some FPGAs. Distributed RAM is, as its name suggests, distributed throughout the FPGA. A single 6-input LUT can store 64 bits, while a 4-input LUT can store 16 bits.

Distributed RAM is read asynchronously, but written synchronously (requires a clock). Writes are limited to a single port, but you can read from up to four ports in some FPGAs. Distributed RAM is flexible in the data width it supports, for example, if dealing with 32-level data you use a width of 5 bits.

1.5.2.2. Dedicated RAM

As FPGA architectures evolved to support the implementation of more and more complex digital systems, memory needs increased. As a consequence, vendors decided to include in their devices dedicated memory blocks, which in addition use specific interconnection lines to optimize access time. They are particularly suitable for implementing “deep” memories (with a large number of positions), whereas distributed memory is more suitable for “wide” memories (with many bits per position) with few positions, shift registers, or delay lines.

1.5.3. Hard Memory Controllers

In many FPGA applications, a huge amount of data has to be handled, but there is not enough embedded memory available for that. In such cases, external memory has to be used, and the corresponding memory controller needs to be implemented in the FPGA. Since there exist a wide variety of memories, the required interfaces are also very diverse, from simple parallel or serial interfaces.

To address this issue, FPGA vendors offer different soft IP core-based solutions. However, these do not provide good-enough performance when dealing with very large memories (up to the GB range) or very fast operation requirements (hundreds of MHz or even GHz). This is the reason why FPGA vendors are including hard memory controllers in their most current devices. For instance, Arria V and 10 families from Altera include dedicated hardware for access control to external DDR/DDR2/DDR3/ DDR4 SDRAM memories. Spartan-6 and Virtex-6 families from Xilinx also include DDR3 hard memory controllers, enhanced in Series 7 families of devices and extended in the UltraScale family to support DDR4 memories.

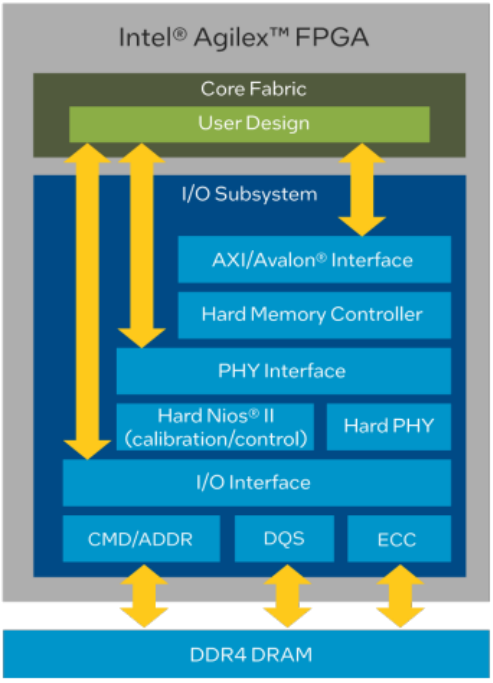


Figure 4. Intel Agilex FPGA E-Series DDR4 External Memory Interface

1.5.4. Digital Signal Processing

FPGAs incorporate dedicated digital signal processing (DSP) blocks. Considering signal processing over the years has been the most significant application of embedded multipliers, it is just natural that they evolved into more complex blocks, called DSP blocks, which includes all resources required to implement a MAC unit, eliminating the need for using distributed logic.

Different architectures exist for DSP blocks, but most of them share three main stages, namely pre-adder, multiplier, and ALU. Depending on the device, the ALU can just consist of an adder/subtractor or include additional resources aimed at giving the DSP block increased computation power.

You can also implement multiplication directly in logic (LUTs and flip-flops), but it takes significant resources. Using dedicated DSP blocks for multiplication makes sense from a performance and logic-use perspective. Hence, even small FPGAs dedicate space to DSP blocks.

1.5.5. Input Output Blocks

Input Output Blocks (IOBs) is a programmable input and output unit, which is the interface between FPGA and external circuits. Used to complete the driving and matching requirements for input/output signals under different electrical characteristics.

In order to facilitate management and adapt to multiple electrical standards, the IOB of FPGA is divided into several banks. The interface standard of each bank is determined by its interface voltage VCCO. A bank can only have one VCCO, but VCCO of different banks can different. Only ports of the same electrical standard can be connected together. The same VCCO voltage is the basic condition of the interface standard.

1.6. How is an FPGA programmed or configured?

FPGAs require configuration so the device's logic circuits and interconnects know what role they should play in the implementation of a specific application. Using specialized software provided by the FPGA supplier, developers design the logic to be implemented in the FPGA using either graphical design capture or a hardware description language (HDL). Intel FPGA AI Suite software ecosystem illustrated in Figure 5 and Xilinx Vitis™ Unified Software Platform shown in Figure 6 are two examples for specialized specific FPGA vendor softwares.

The software then compiles the design by synthesizing it and then placing and routing the logic that best fits the target FPGA, after which the software creates a bitstream that is used to configure (or program) the FPGA. Once the bitstream is downloaded to the FPGA, the device is then ready to perform its defined task.

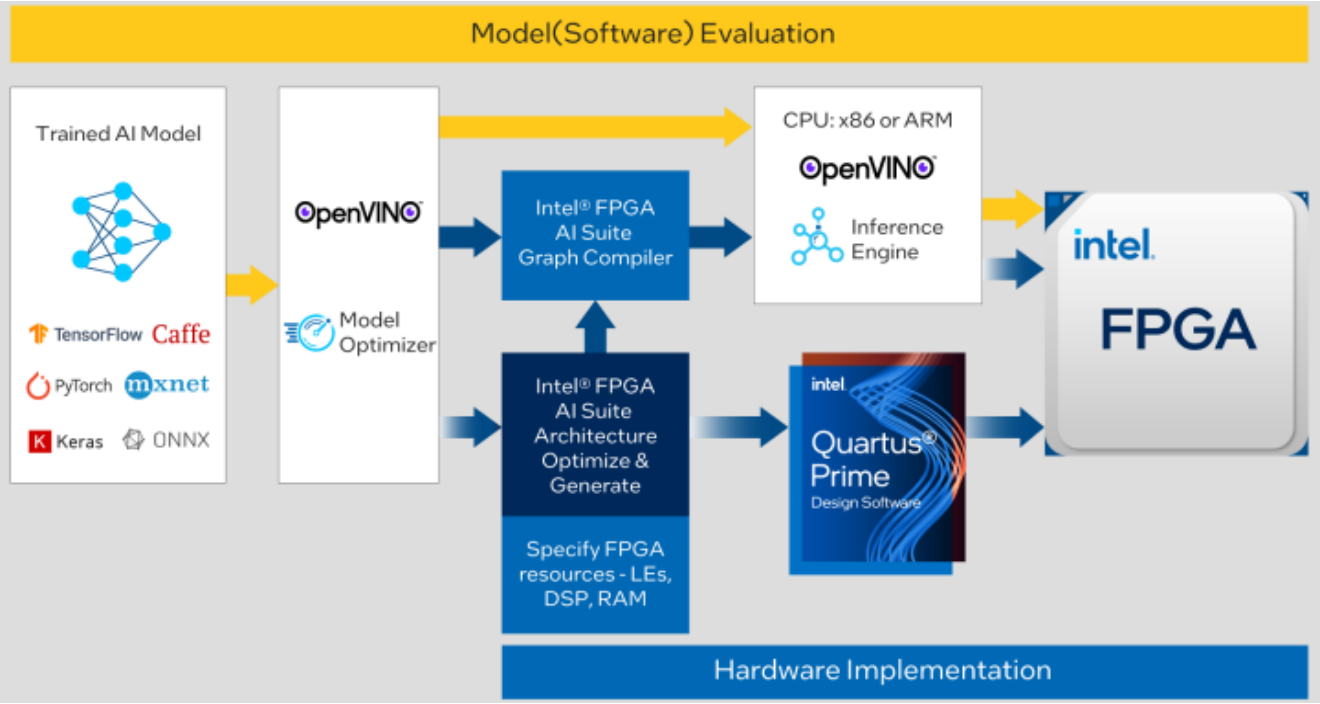


Figure 5. Intel FPGA AI Suite

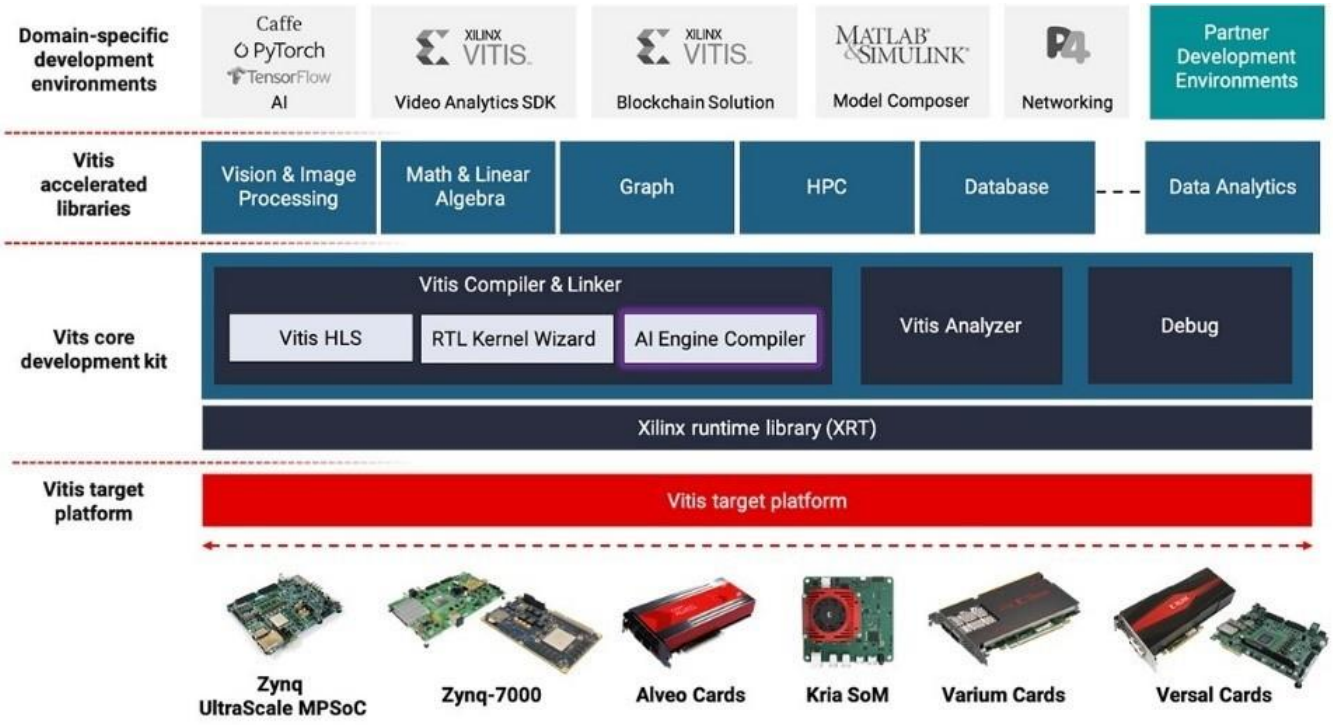


Figure 6. Xilinx Vitis™ Unified Software Platform

1.7. SoC FPGAs

SoC FPGAs are semiconductor devices that integrate programmable logic with hard processor cores, such as those from Arm. This architecture provides the ease of programming a processor along with the flexibility and performance of a programmable logic fabric.

SoC FPGAs are available from several sources:

- AMD Xilinx: Zynq®-7000 SoC and Zynq UltraScale+ MPSoC families.
- Intel Altera: Stratix® 10 SoC FPGAs, Arria® 10 SoC FPGAs, and Arria V SoC and Cyclone® V SoC families.

2. Design flow for AMD Xilinx FPGA

2.1. Nexys A7-100T

The Nexys A7 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx®. With its large, high-capacity FPGA, generous external memories, and collection of USB, Ethernet, and other ports, the Nexys A7 can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and several I/O devices allow the Nexys A7 to be used for a wide range of designs without needing any other components.

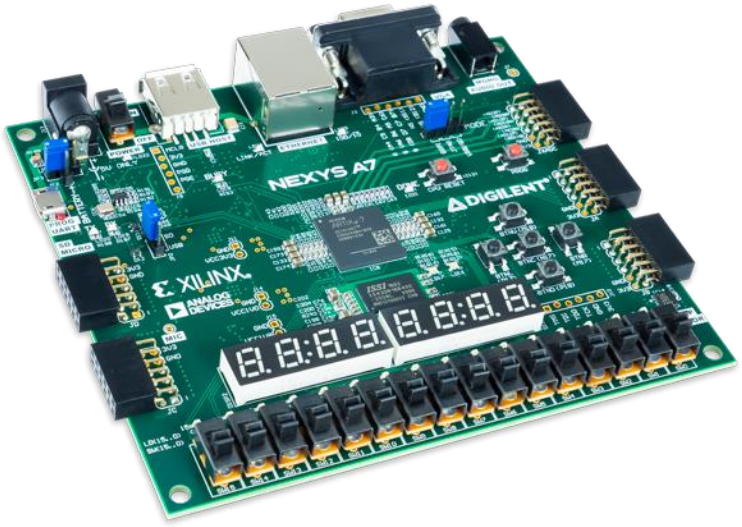


Figure 7. Nexys A7 board

2.2. Vivado Design Suite

Vivado is the design software for AMD adaptive SoCs and FPGAs. It includes: Design Entry, Synthesis, Place and Route, Verification/Simulation tools.

2.3. Design flow

Figure 8 shows the high-level design flow in the Vivado Design Suite for FPGAs and SoCs.

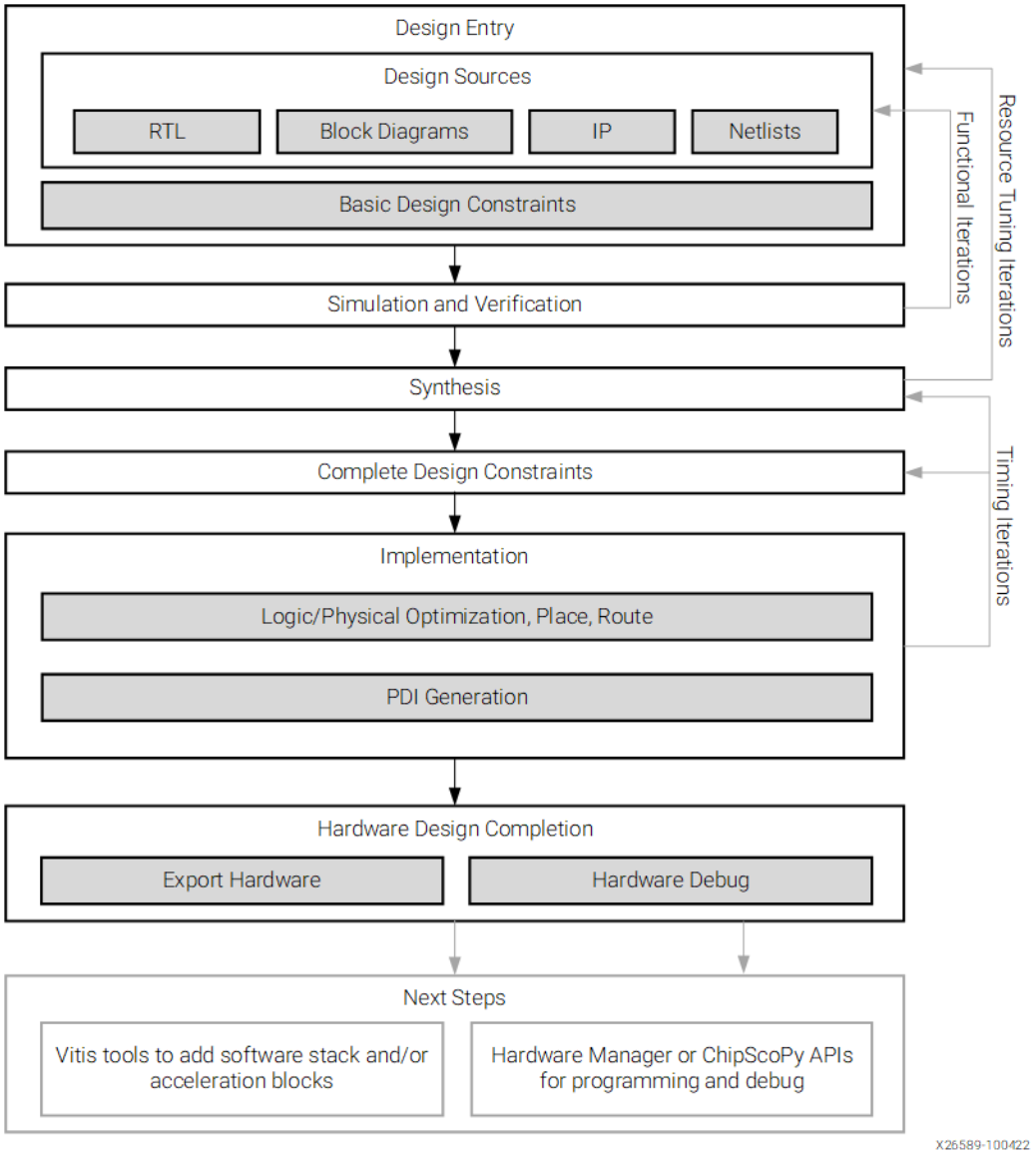


Figure 8. Platform-based design flow through the Vitis environment

2.3.1. Setup environment

Setting up environment is necessary to be able to use Vivado Design Suite. In the beginning, you need to add these following lines to ***~/.bashrc*** file

```
#Vivado
export PATH=$PATH:/tools/Xilinx/Vivado/2018.1/bin
export VIVADO_HOME=/tools/Xilinx/Vivado/2018.1
```

then type command

```
$ bash
```

2.3.2. Project directory structure

```
<name_pj>
├── doc
├── hdl
├── sim
├── lib
├── syn
├── fpga
│   ├── <name_pj>_fpga_interface      # All user interface
│   ├── <name_pj>_fpga_top            # FPGA top project
│   ├── <name_pj>_fpga_top.cache
│   ├── <name_pj>_fpga_top.hw
│   ├── <name_pj>_fpga_top.ip_user_files
│   ├── <name_pj>_fpga_top.runs
│   ├── <name_pj>_fpga_top.sdk        # Software project
│   ├── <name_pj>_fpga_top.sim
│   ├── <name_pj>_fpga_top.srcs       # Top IP and top testbench
│   └── <name_pj>_fpga_top.tmp
├──<name_pj>_fpga_<user_ip1>
│   └── The same as top project
├── <name_pj>_fpga_<user_ip1>
├── <name_pj>_fpga_<user_ip3>
├── <name_pj>_fpga_<user_ip4>
└── <name_pj>_fpga_lib                # HDL code for needed Ips used in block design
```

2.3.3. Hardware design flow

In the beginning, you can use this command to run GUI of Vivado Design Suite

```
$ vivado
```

2.3.3.1. Package IP

The first step of hardware design flow is packaging the IP. Before doing it, you must create a new project in Vivado Design Suite following the steps in Figure 9, Figure 10, Figure 11, Figure 12 and Figure 13.

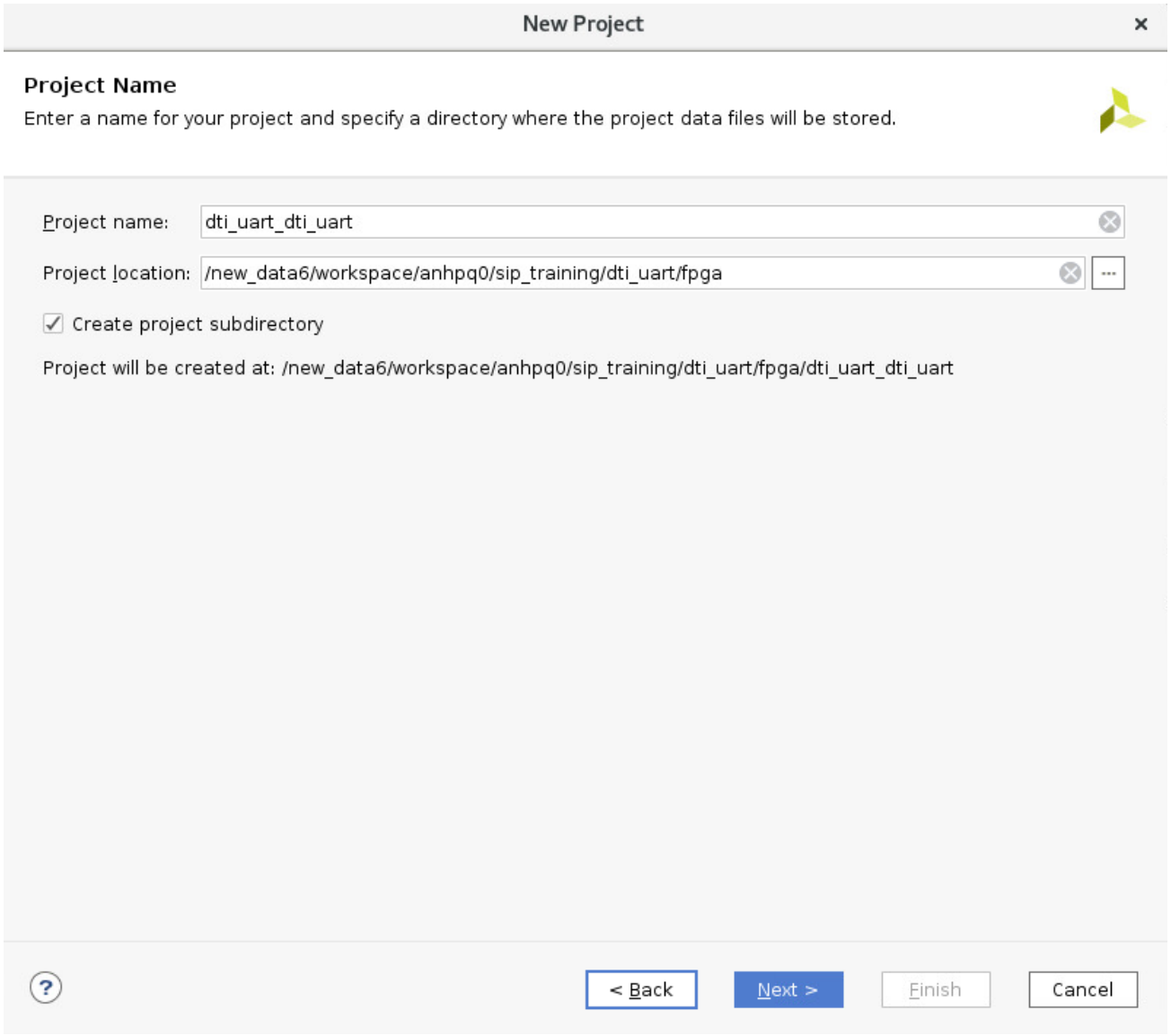


Figure 9. Create new project and set project name in Vivado Design Suite

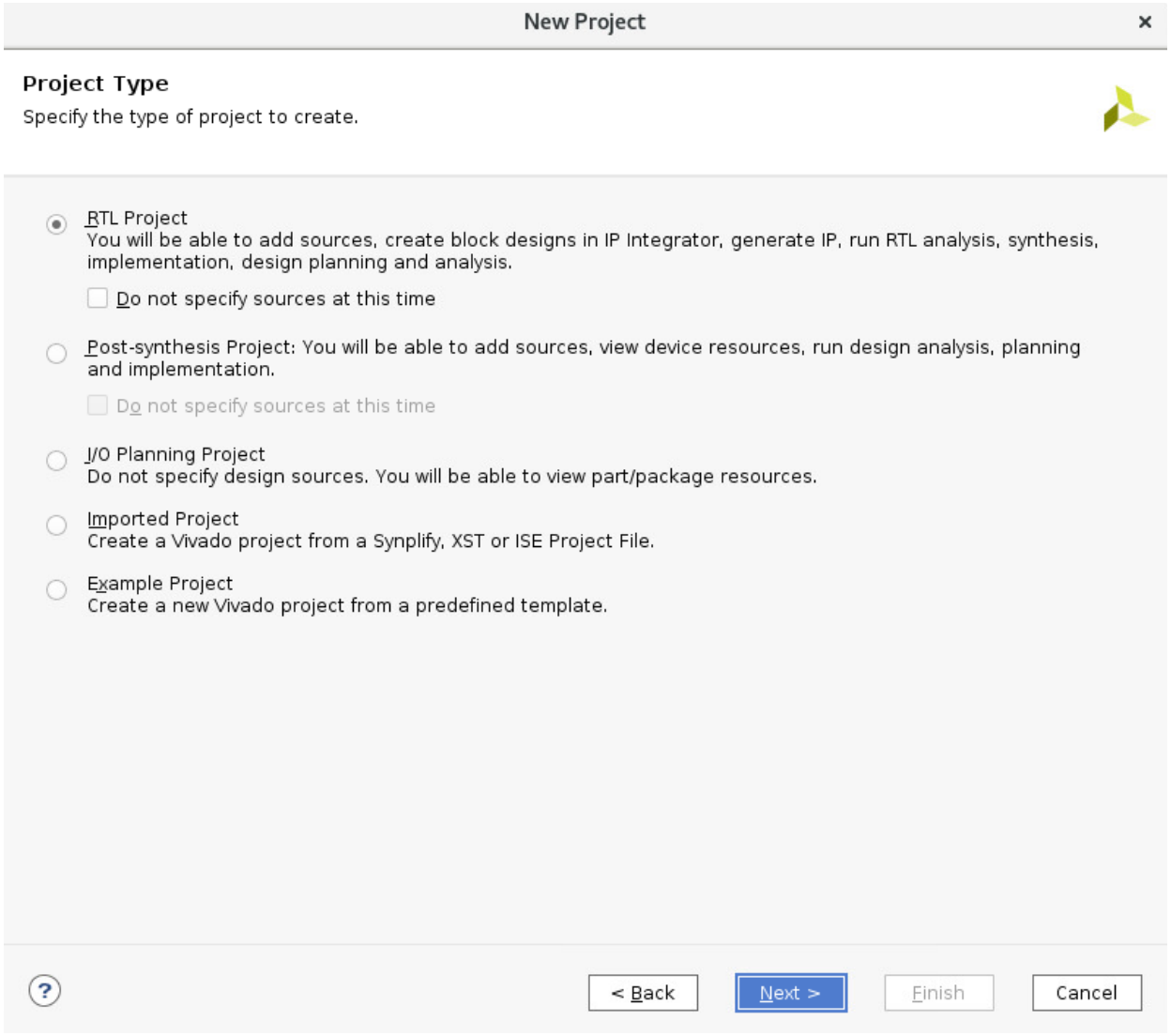


Figure 10. Create new project and set project type in Vivado Design Suite

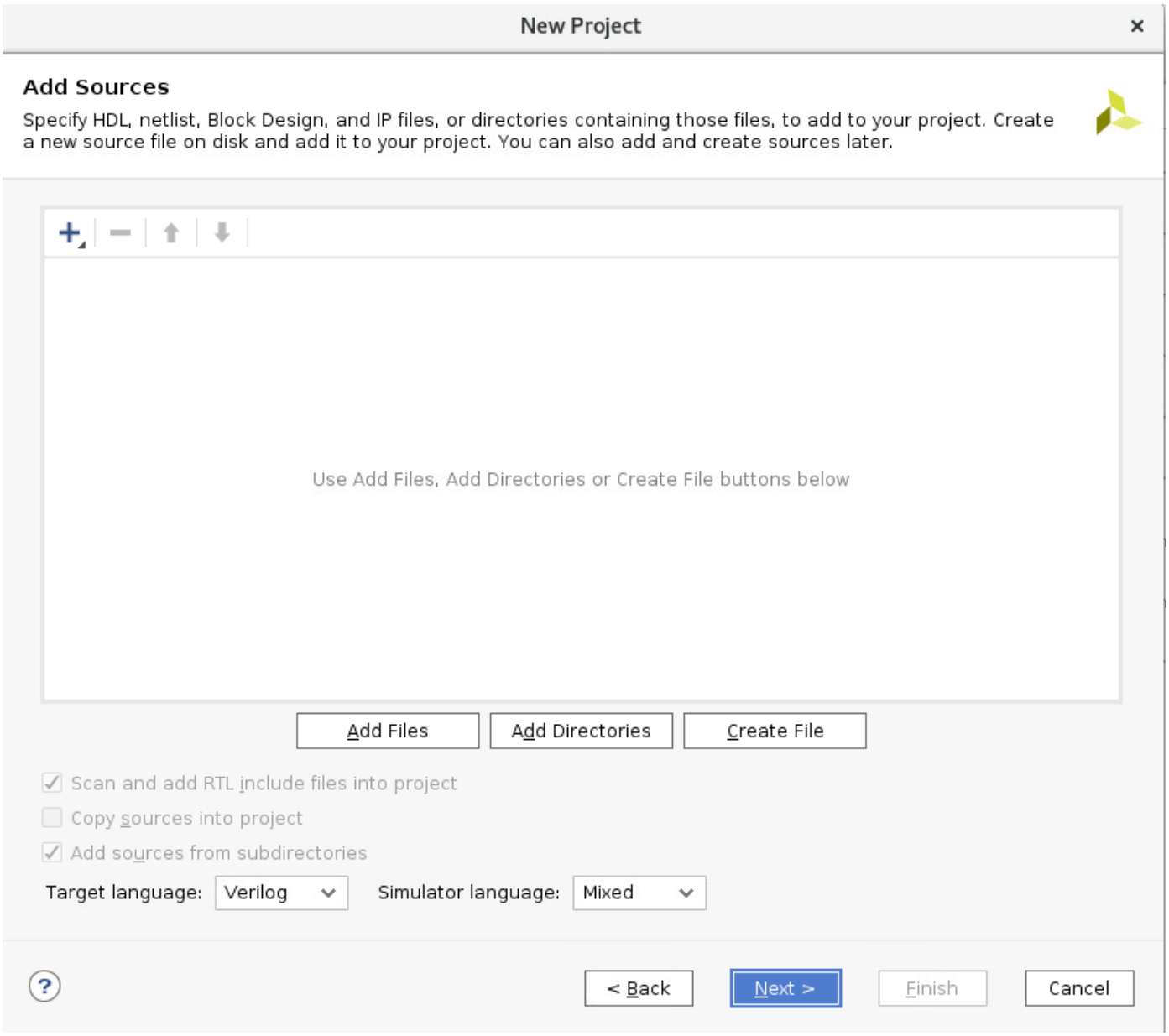


Figure 11. Create new project and set HDL source in Vivado Design Suite

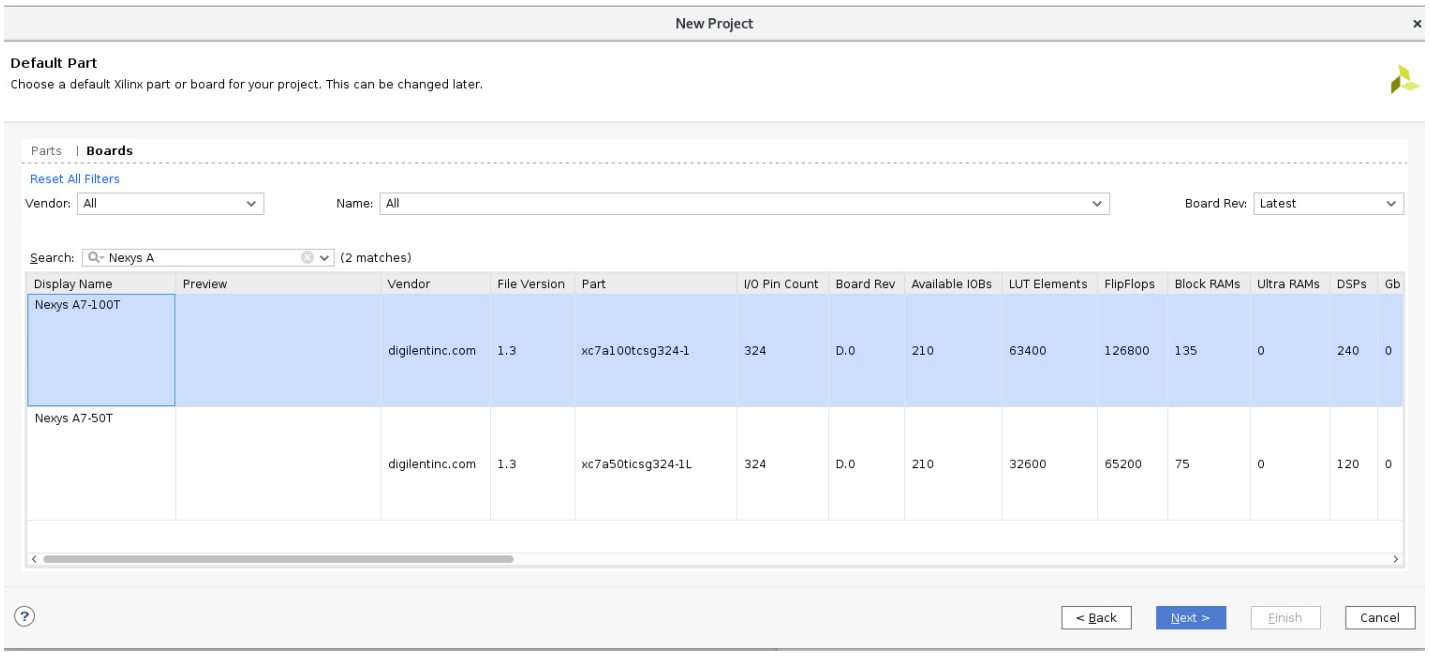


Figure 12. Create new project and set board type in Vivado Design Suite

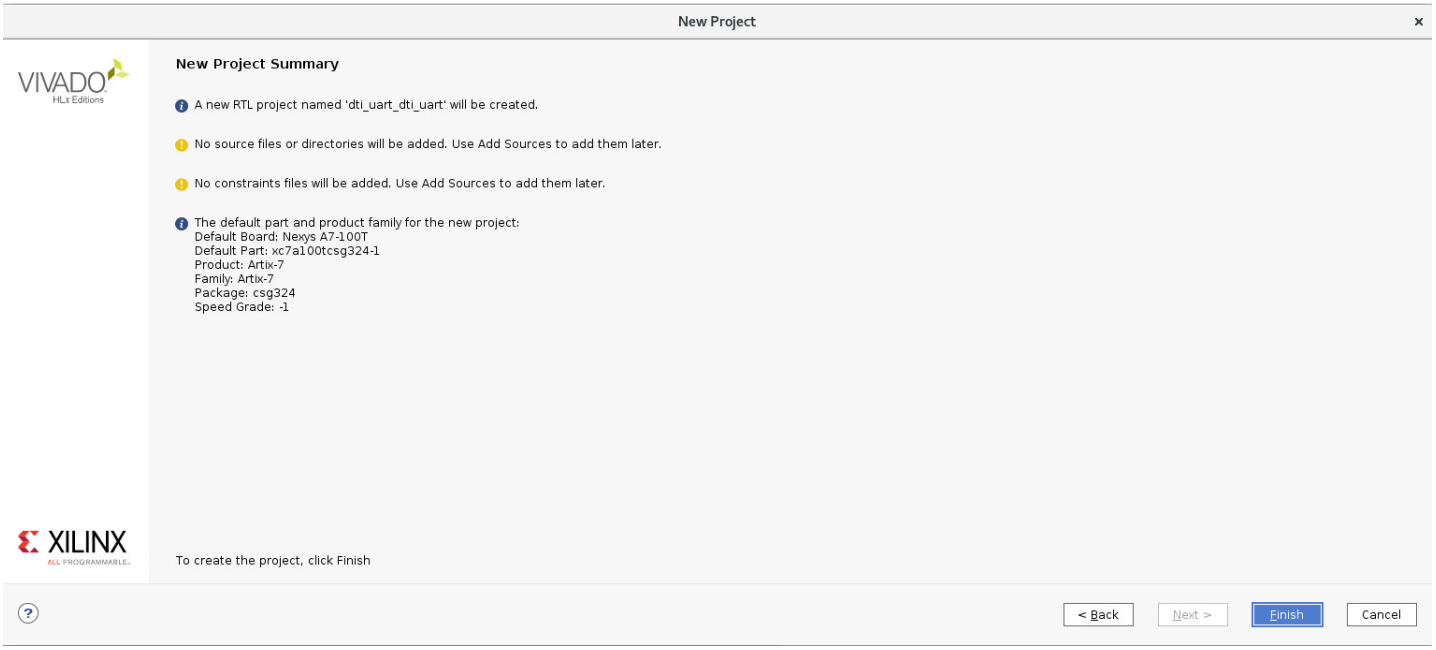


Figure 13. New project summary

After creating new project, to add RTL source or testbench source to the project, you need to follow the step in Figure 14.

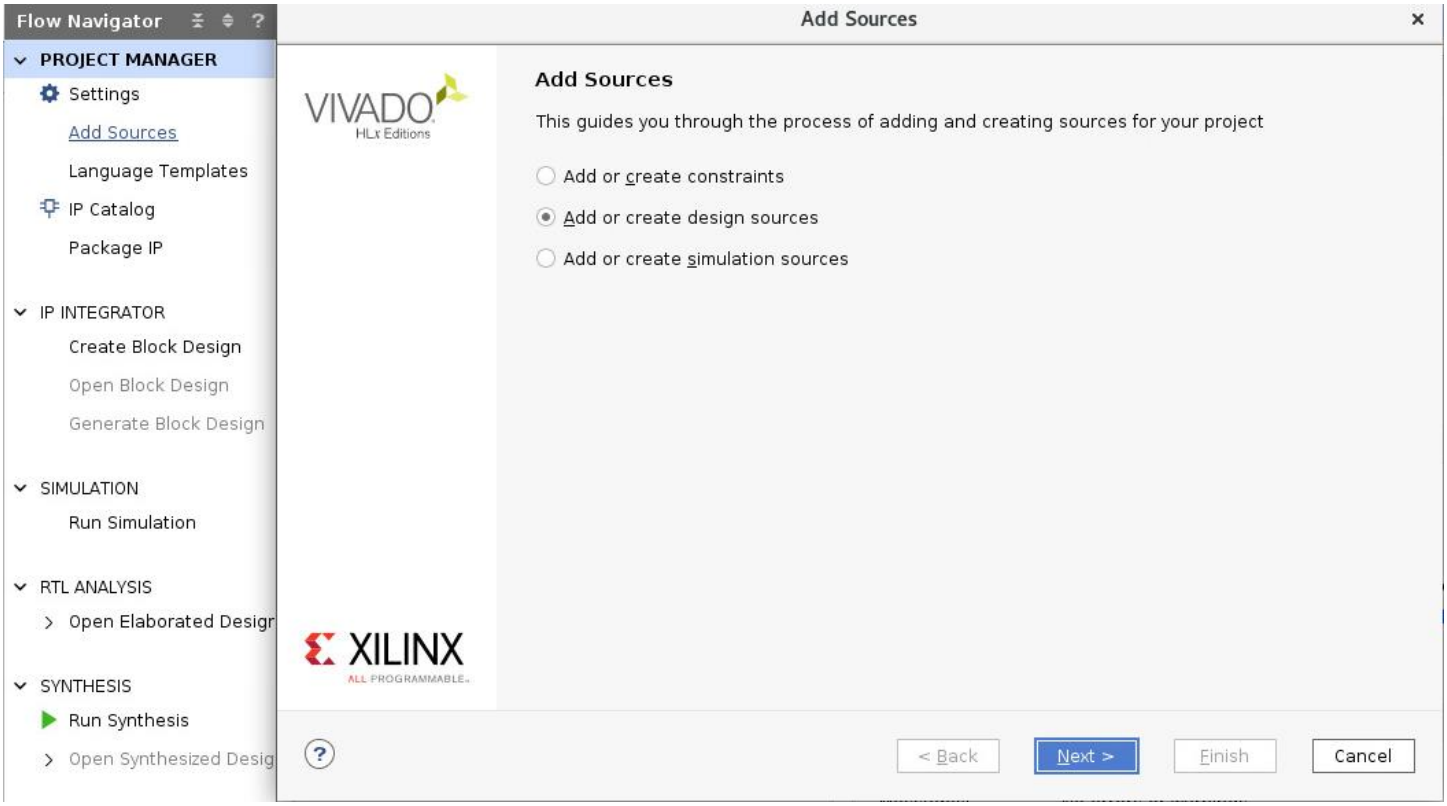


Figure 14. Add HDL source and testbench source to project

The steps in Figure 15 and Figure 16 will help you to start packaging the IP.

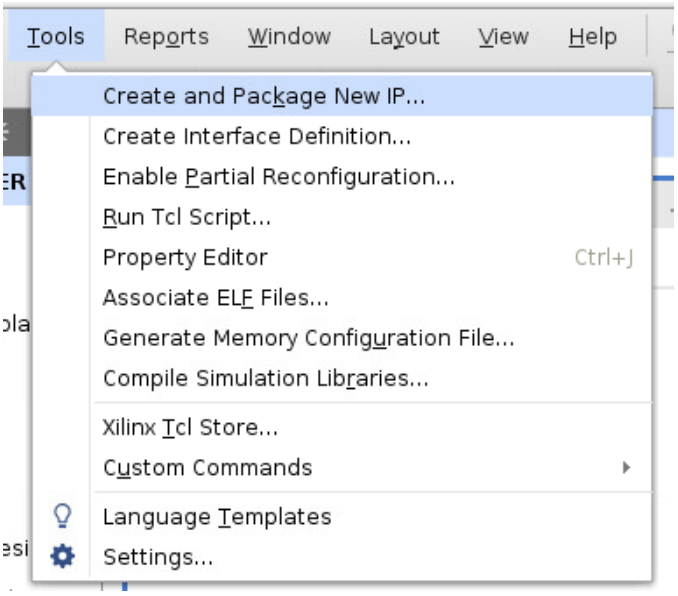


Figure 15. Package the IP step 1

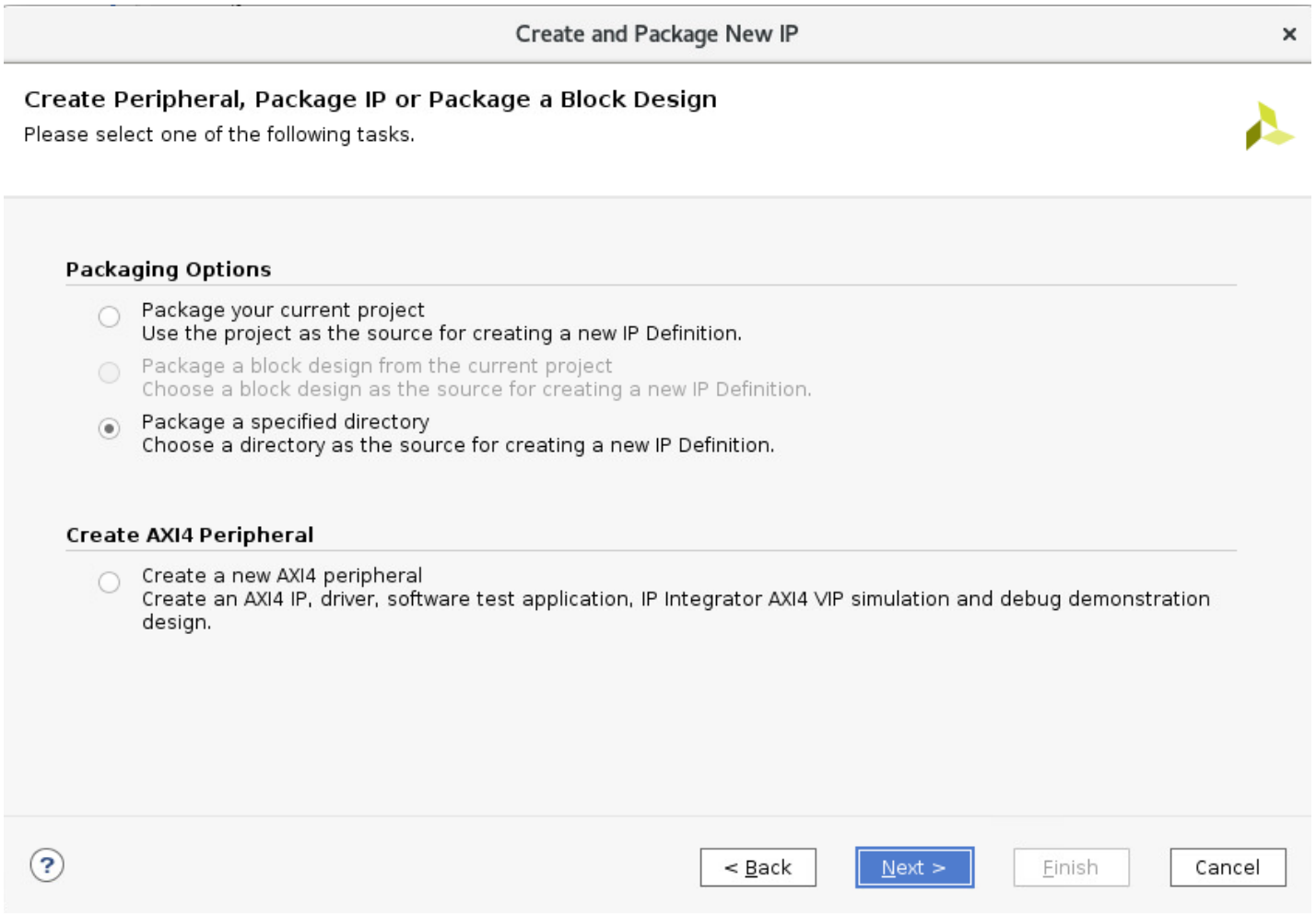


Figure 16. Package the IP step 2

After that, you can go to **Package IP** in tab **PROJECT MANAGER**. The next step is to combine all the signals of a specific protocol to a group in tab **Ports and Interfaces**.

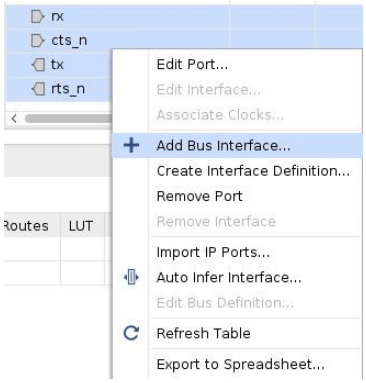


Figure 17. Grouping signals

You can set the interface definition for those signals. In the case of **dti_uart** IP, it is necessary to combine all APB signals into a group.

Figure 18. Interface definition step 1

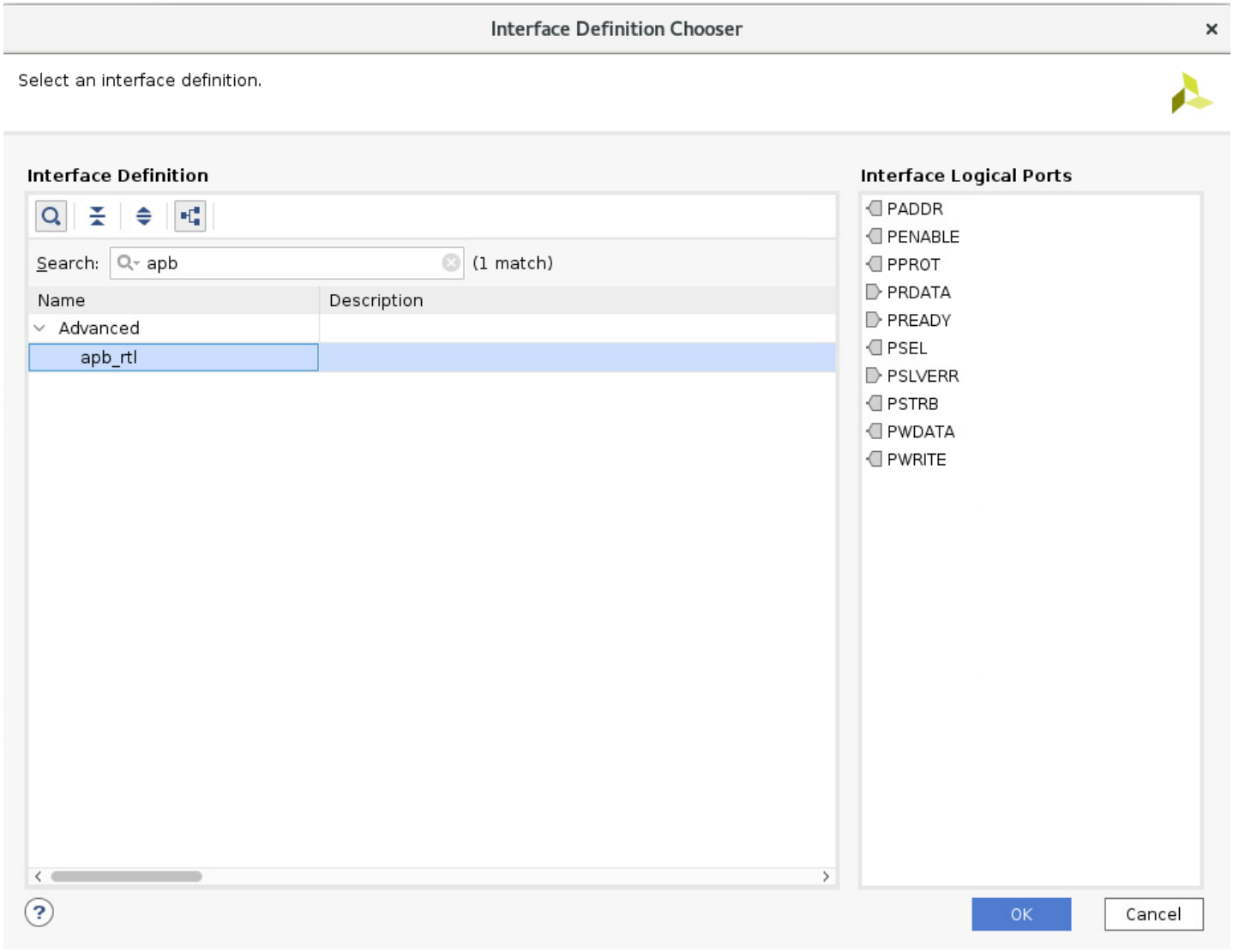


Figure 19. Interface definition step 2

Name	Interface Mode	Enablement Dependency	Is Declaration	Access Handle	Access Type	Direction	Driver Value	Size Left	Size Right	Size Left Dependency	Size Right Dependency
APB	slave										
paddr					ref	in		11	0		
psel					ref	in					
penable					ref	in					
pwrite					ref	in					
pwdata					ref	in		31	0		
pready					ref	out					
prdata					ref	out		31	0		
pslverr					ref	out					
Clock and Reset Signals											
pclk					ref	in					
presetn					ref	in					
pstrb					ref	in		3	0		
rx					ref	in					
cts_n					ref	in					
tx					ref	out					
rts_n					ref	out					

Figure 20. The result of interface definition

The final step of packaging the IP is to assign address ranges (memory map) to the created interface groups.

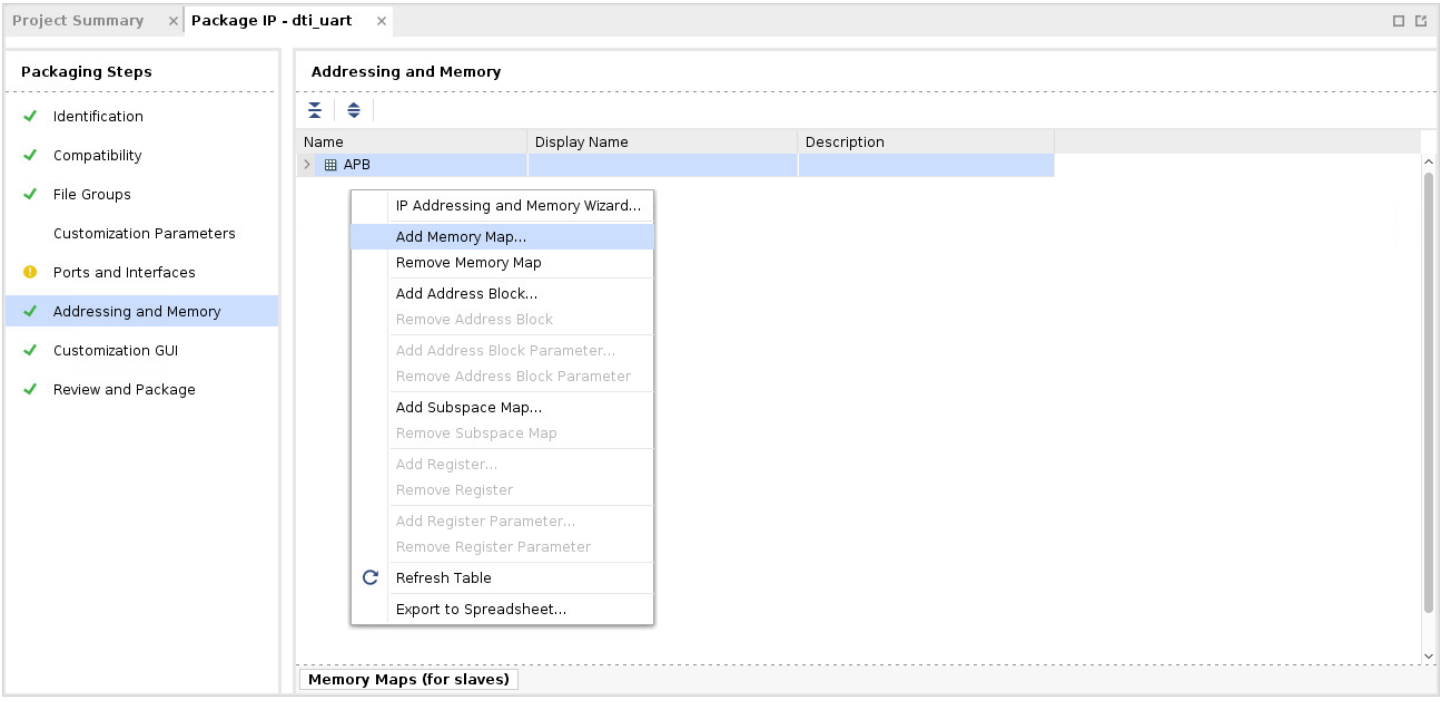


Figure 21. Address mapping step 1

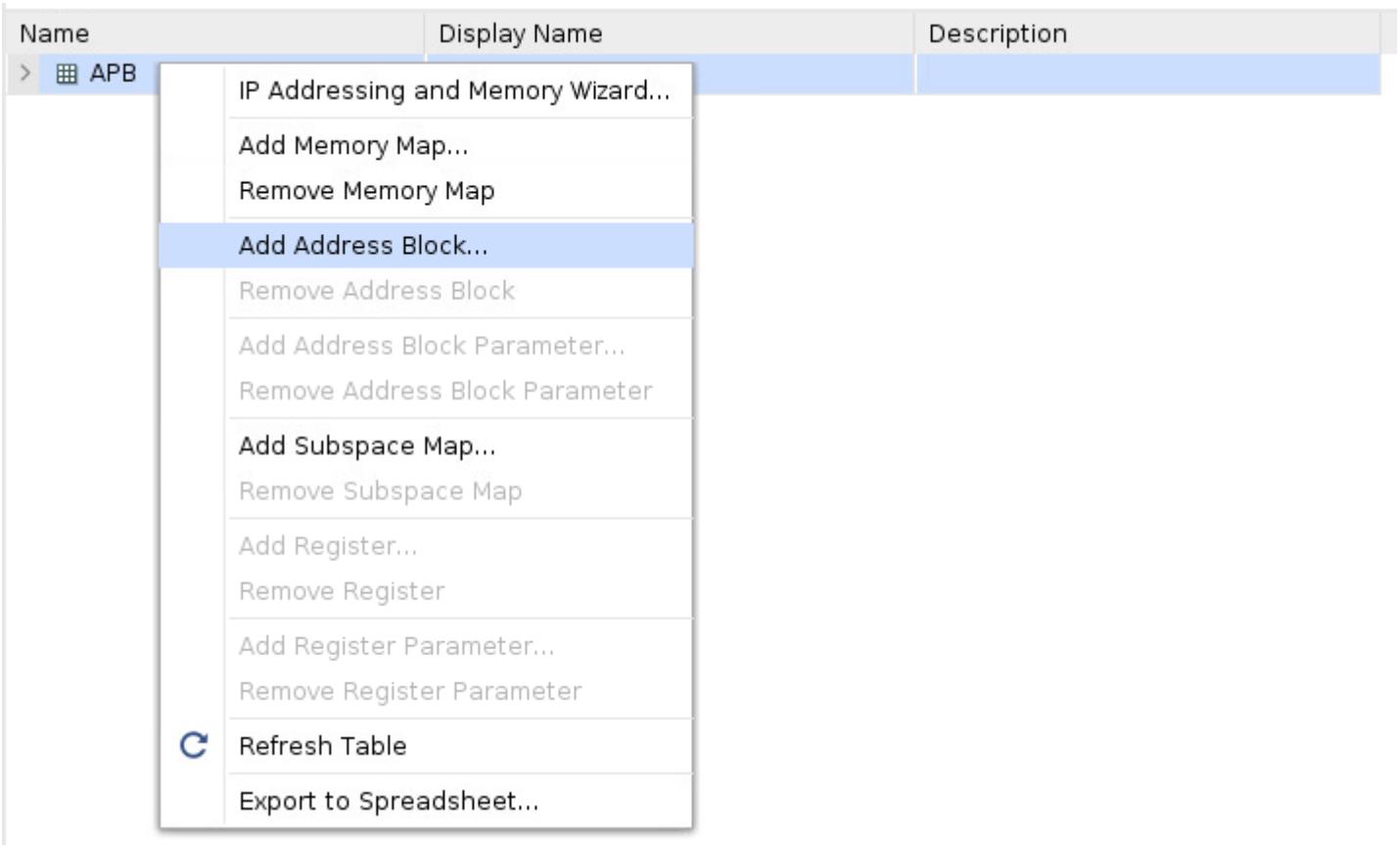


Figure 22. Address mapping step 2

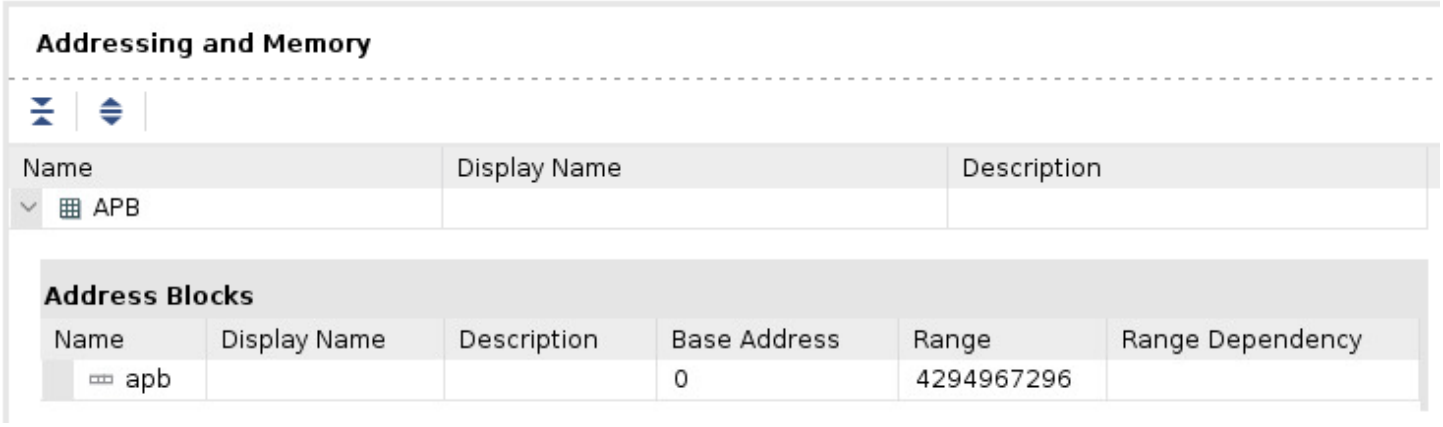


Figure 23. The result of address mapping

Packaging the IP will be finished once you click on button **Package IP** in tab **Review and Package**.

2.3.3.2. Set IP Reposity in top project

When packaging IP is finished, the hardware design flow continues with creating top project which contains the final block design.

All the steps of creating project are discussed in 2.3.3.1. This section show you how to set IP Reposity of the previous packaged IP to this project in Figure … and ….

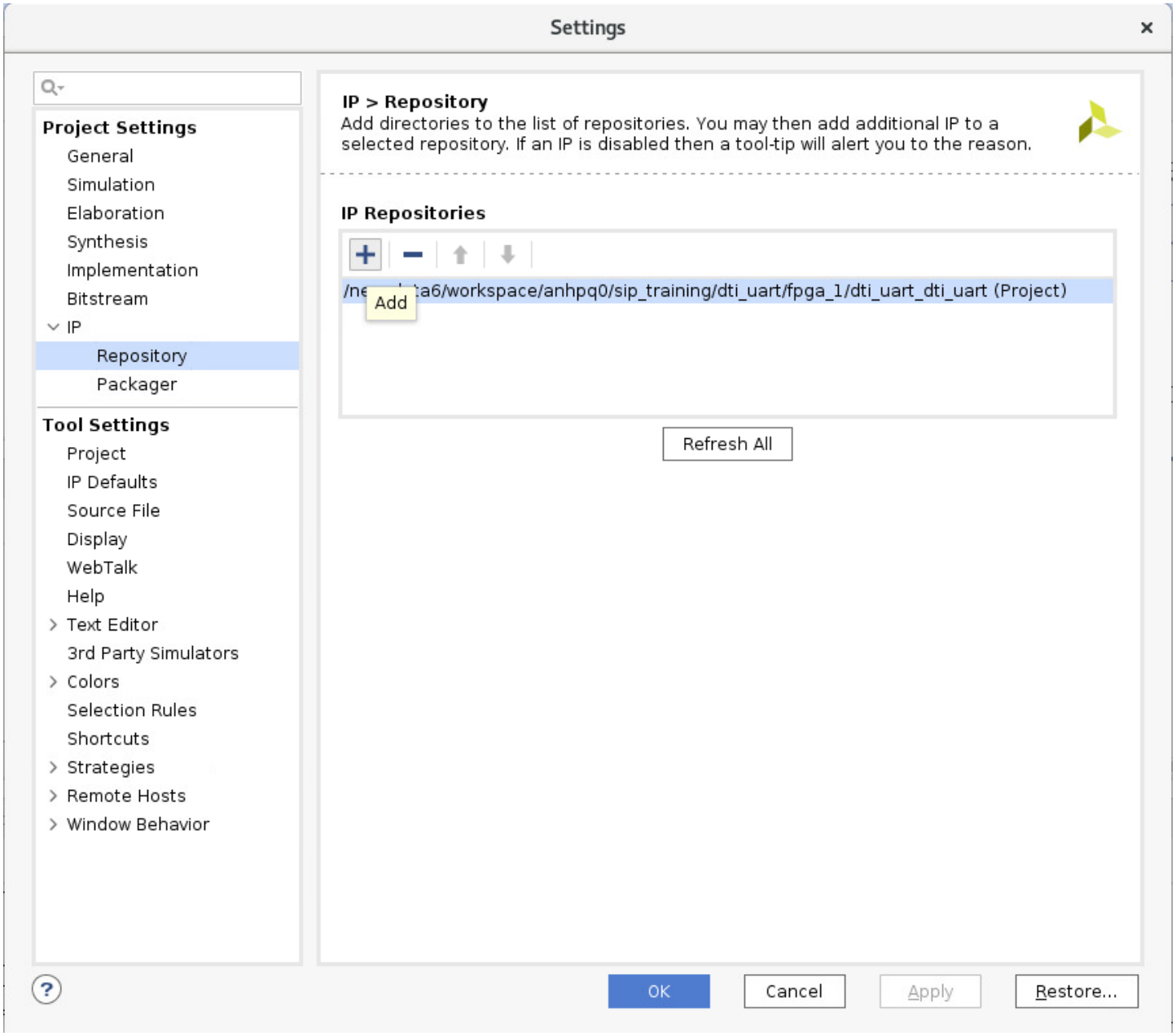


Figure 24. Set IP Reposity step 1

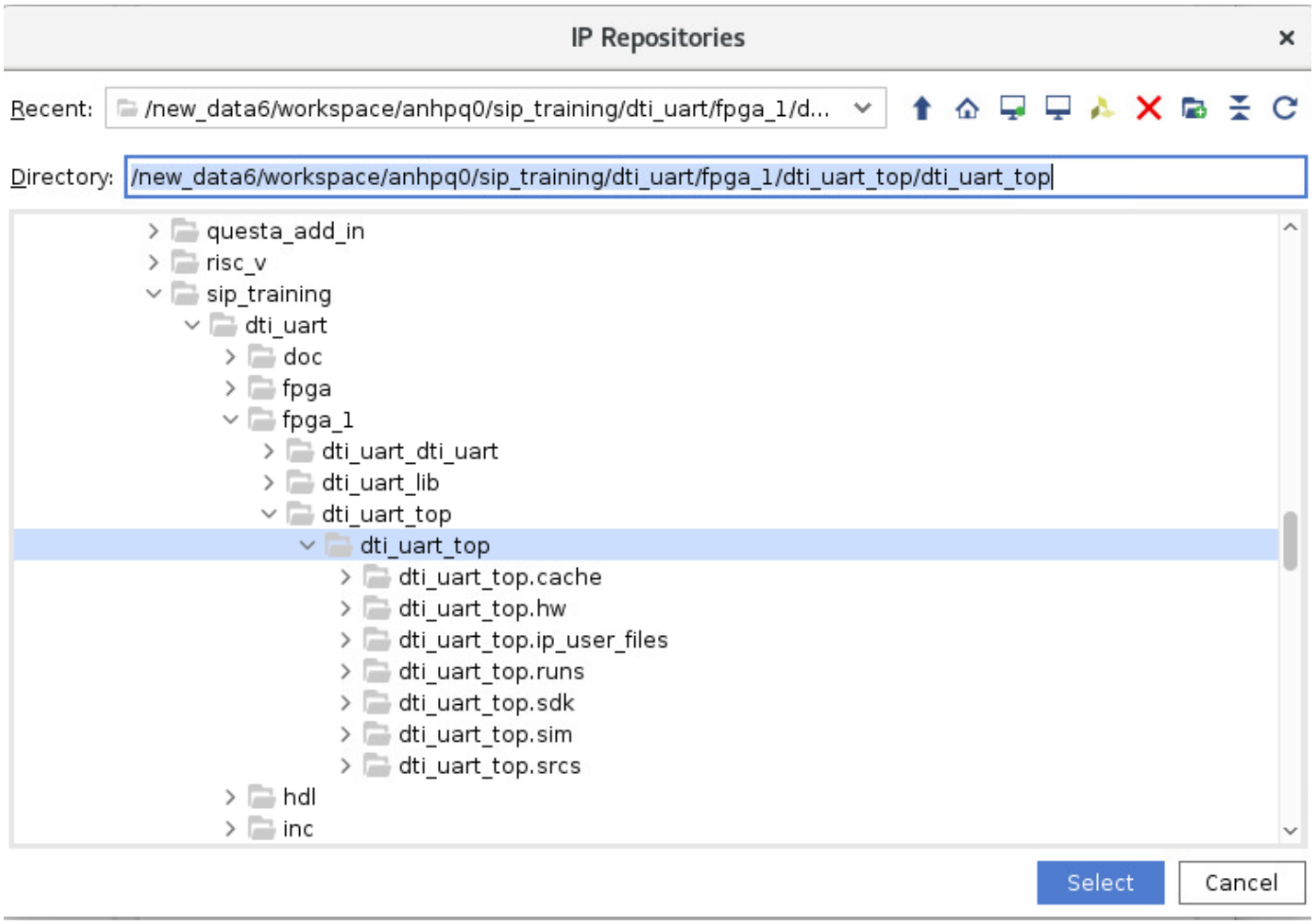


Figure 25. Set IP Reposity step 2

2.3.3.3. Create Block Design

Creating Block Design is one of the most important steps.

Figure 26. Create Block Design

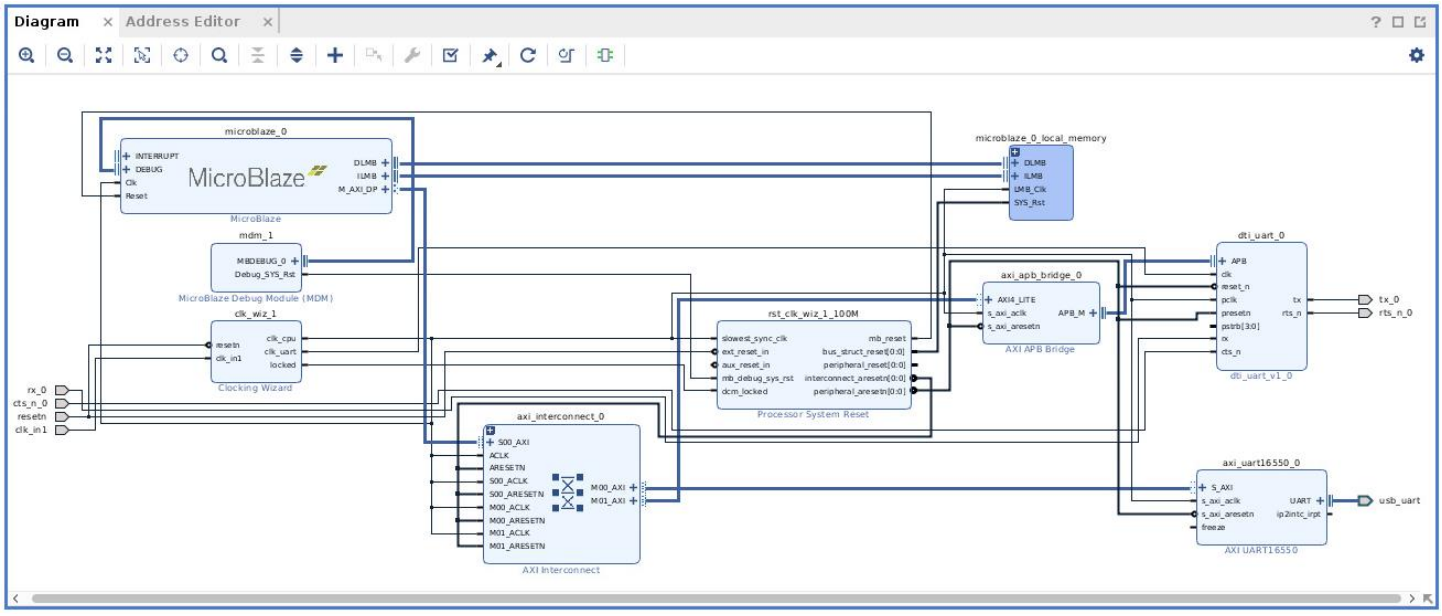
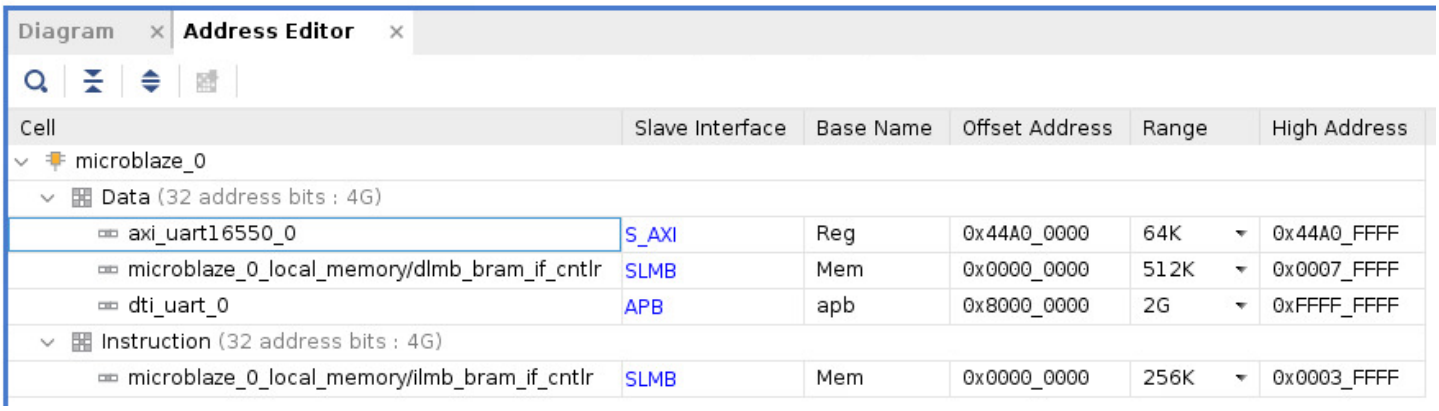


Figure 27. The final Block Design of dti_uart



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_uart16550_0	S_AXI	Reg	0x44A0_0000	64K	0x44A0_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	512K	0x0007_FFFF
dti_uart_0	APB	apb	0x8000_0000	2G	0xFFFF_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	256K	0x0003_FFFF

Figure 28. Address assignment

After creating Block Design, you need to click on **Generate Block Design**.

2.3.3.4. Behavioral Simulation

To check the correctness of Block Design, you need to run behavioral simulation. In the beginning, you need to change the simulation setting as in Figure 29 and Figure 30.

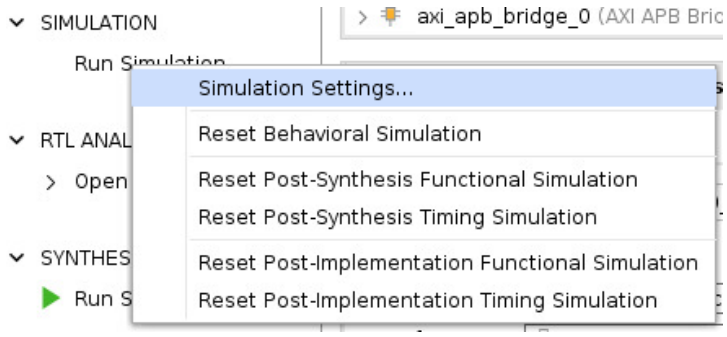


Figure 29. Simulation Settings (1)

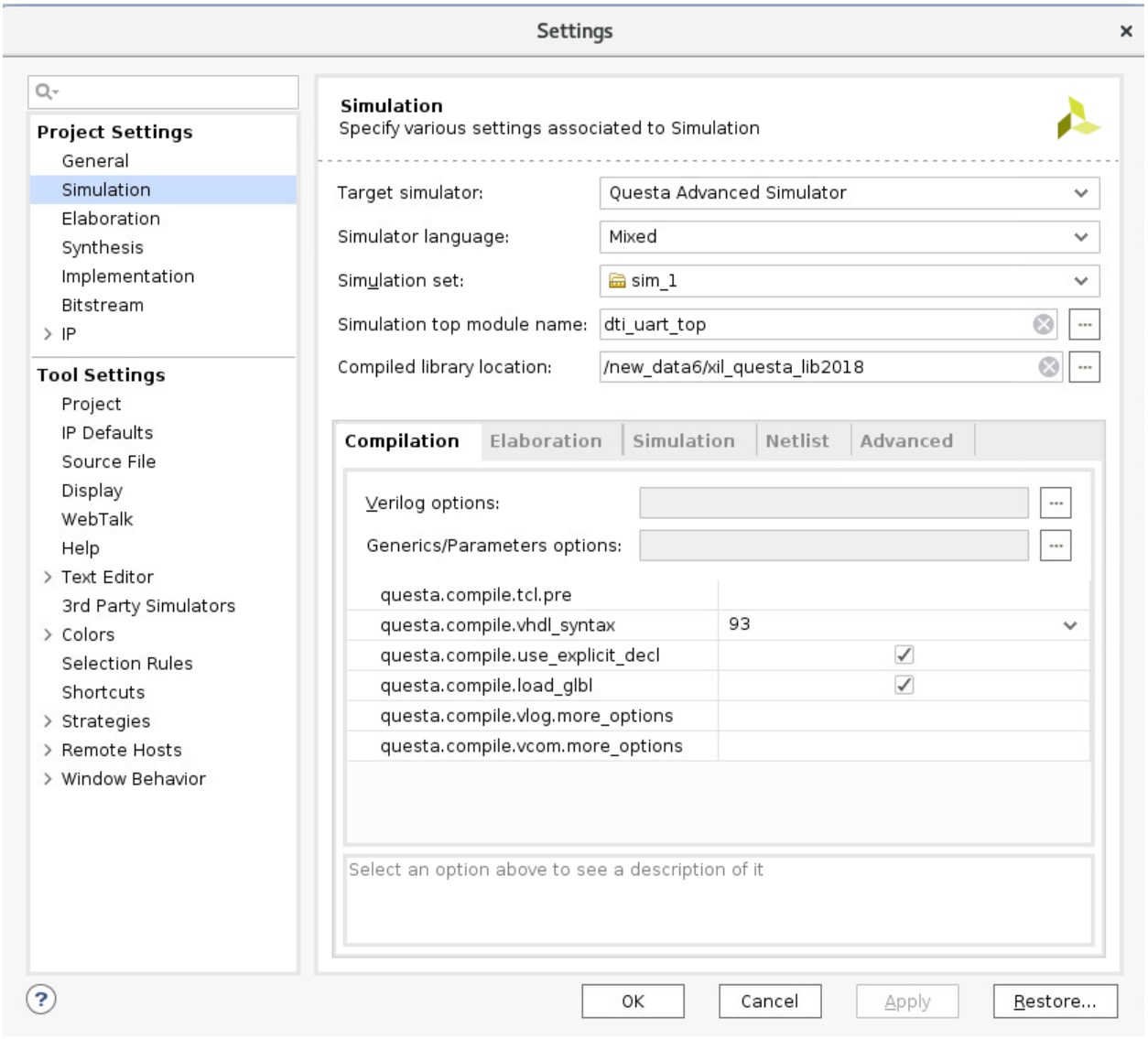


Figure 30. Simulation Settings (2)

Please note that when all C code driver in Software Design Flow, you need to set these options and run post-synthesis and post-implementation after successfully generate bitstream.

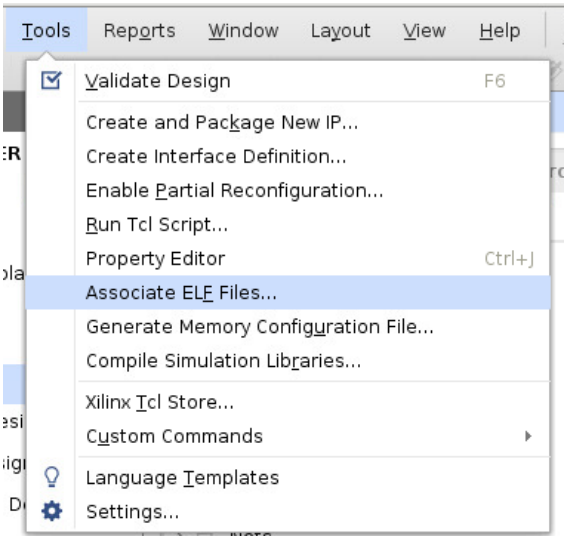


Figure 31. Set associate ELF files step 1

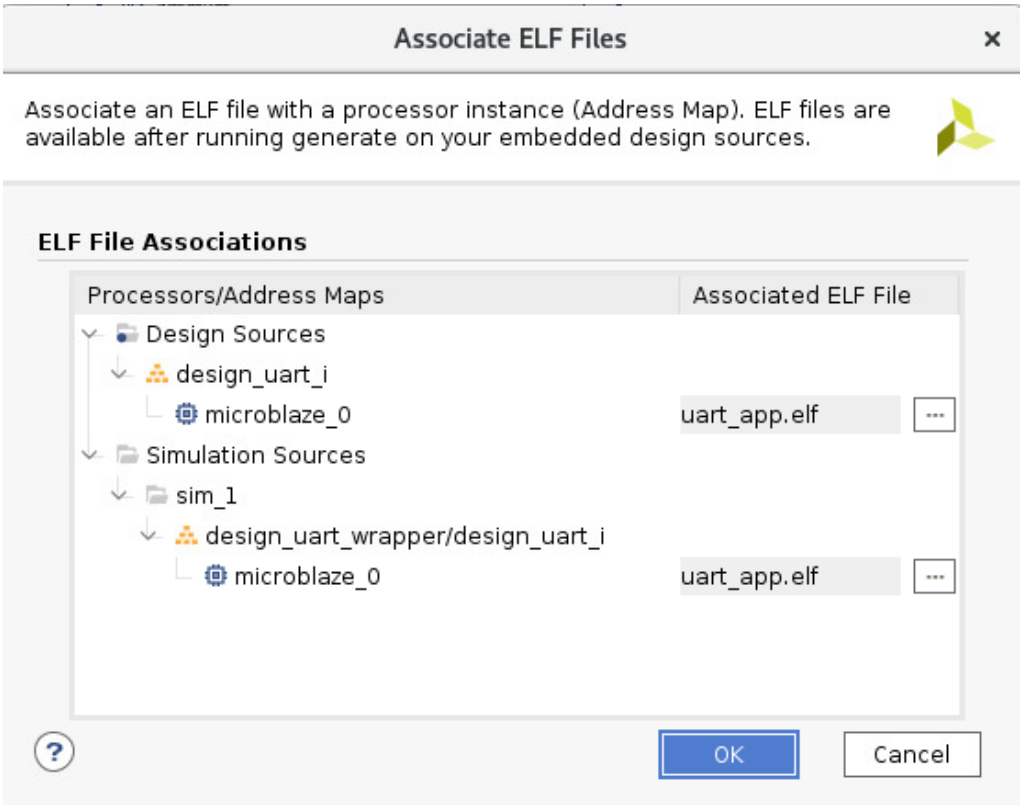


Figure 32. Set associate ELF files step 2

2.3.3.5. Generate Bitstream

To generate bistream, you need to complete the synthesis and implementation steps first. By clicking on Generate Bitstream, all of these stages will be executed.

Please note that it is essential to add the constraint file to the project before carrying out these steps. You can utilize the sample constraint file committed to this project.

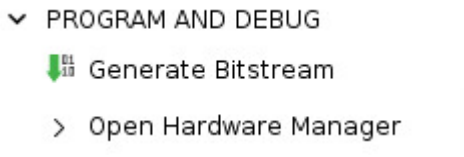


Figure 33. Generate Bitstream

2.3.3.6. Export Hardware Description

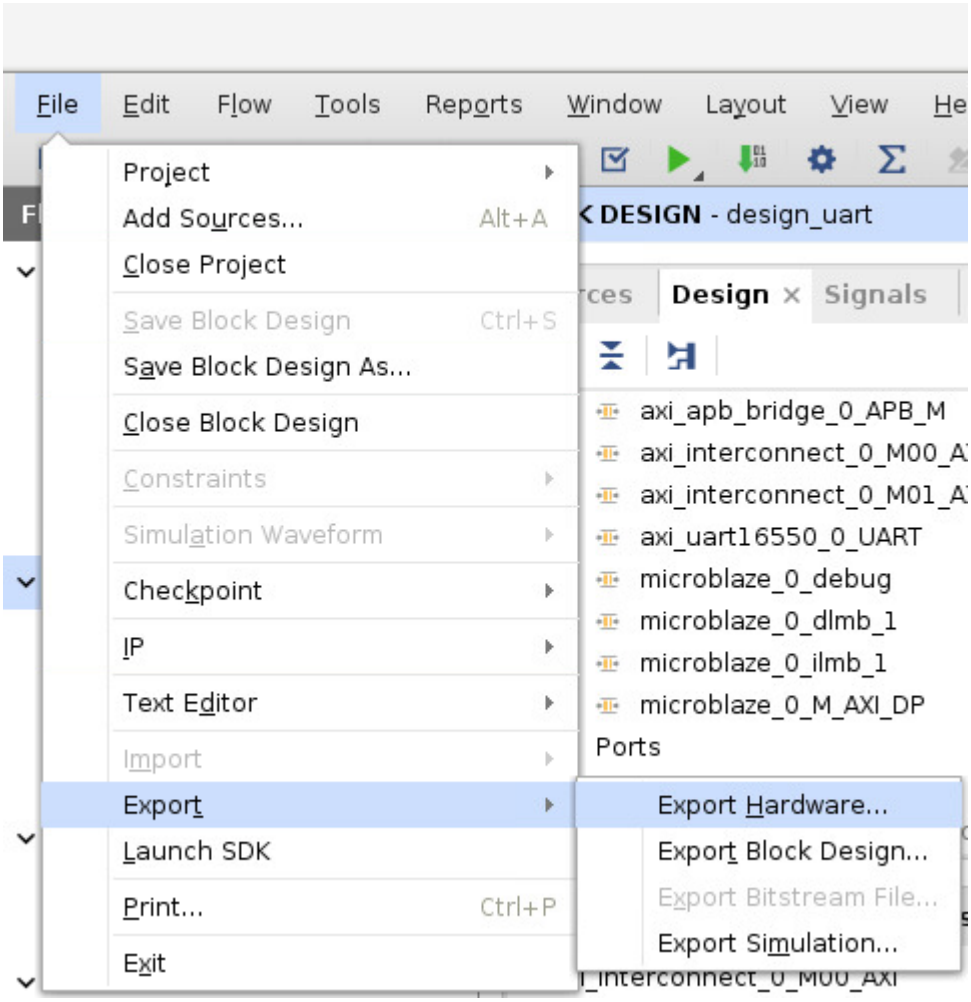


Figure 34. Export hardware

2.3.4. Software design flow

2.3.4.1. Create Project and Board Support Package

In the beginning, you must open Xilinx SDK from Vivado Design Suite.

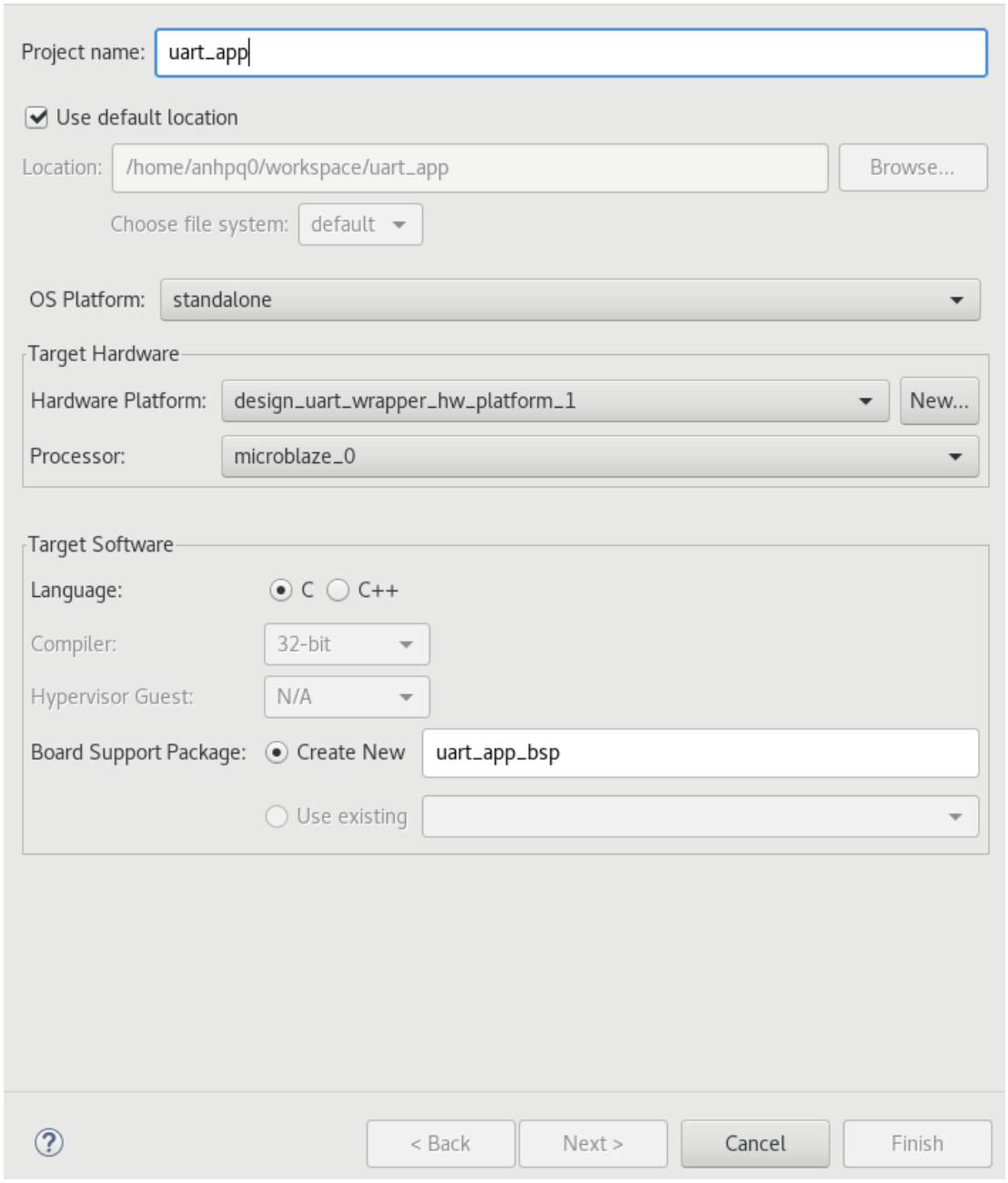


Figure 35. Create Project and Board Support Package

2.3.4.2. Set up and develop driver for the IP

The most important step of Software Design Flow is to develop C code driver for controlling the IP.

2.3.4.3. Program FPGA

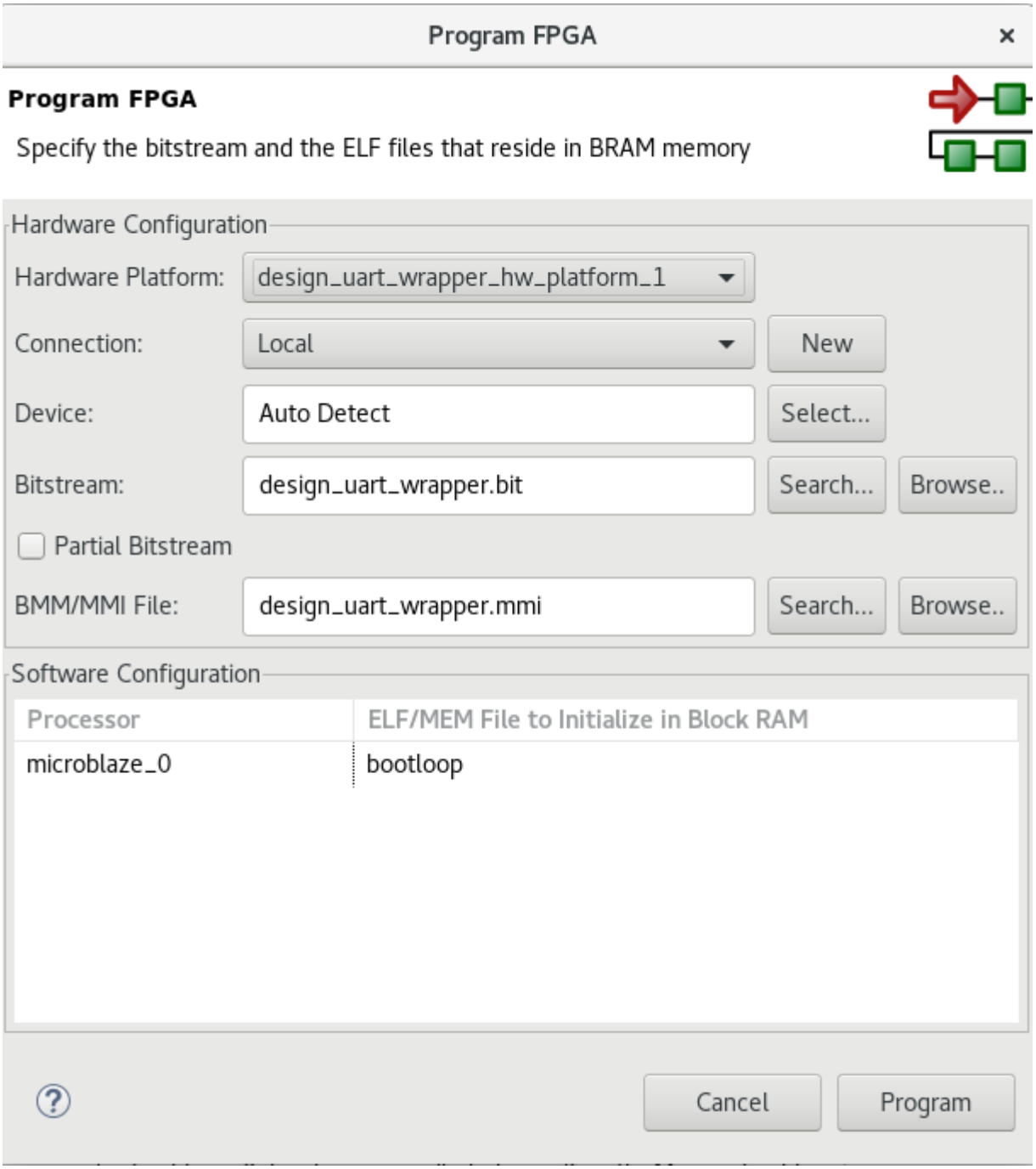


Figure 36. Program FPGA step 1

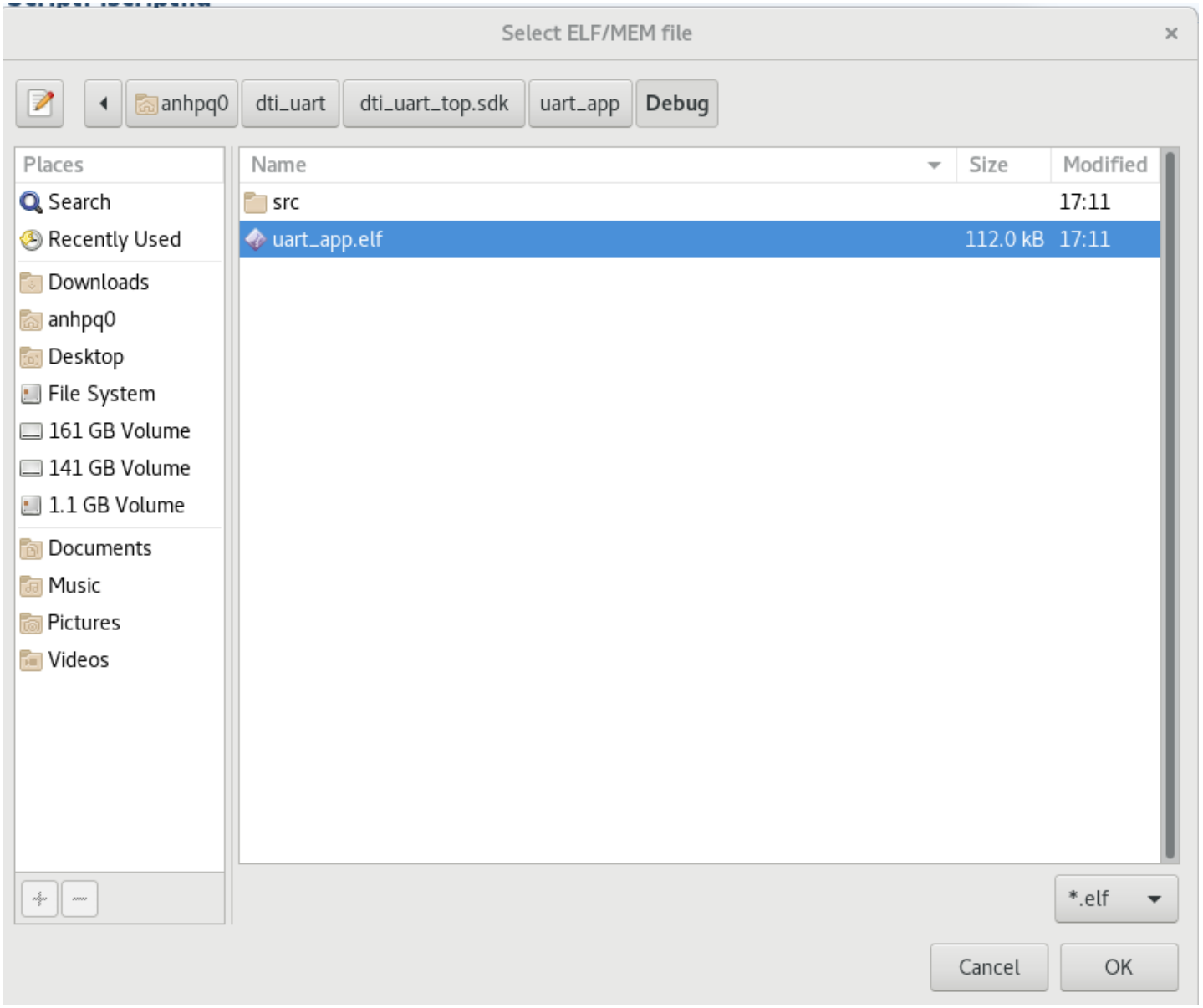


Figure 37. Program FPGA step 2