Xilinx Zynq-7000 & Vitis software Introduction

Agenda

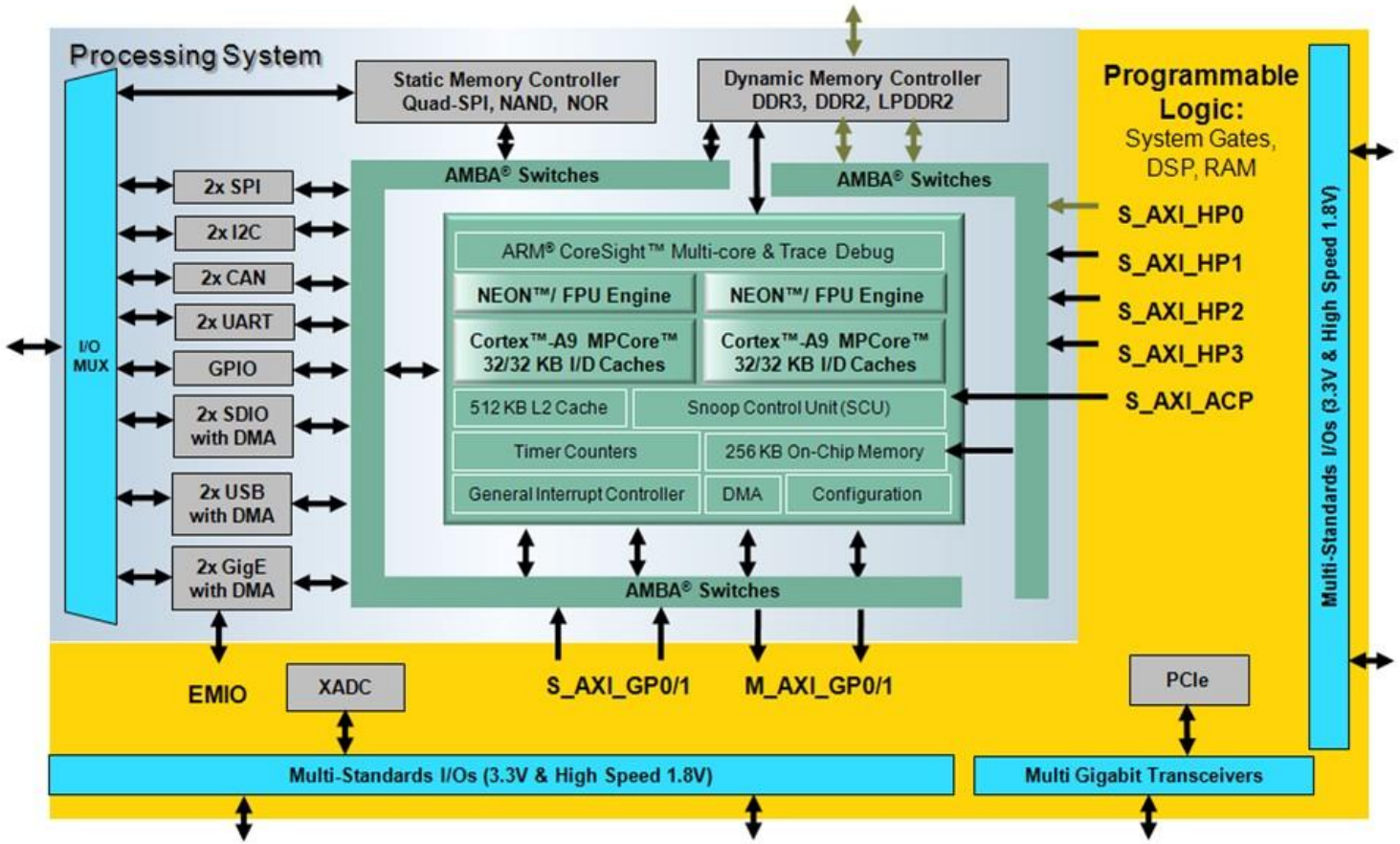
- Zynq-7000 Introduction
  - Zynq-7000 Architecture
  - Processing System (PS) Components
  - Advanced eXtensible Interface (AXI) Bus
  - PS-PL Interfaces
  - PS Clocking Sources
  - Resets
  - Zynq Boot and Configuration
- Vitis Development Workflow
  - Vitis Introduction
  - Preparatory Work from Vivado
  - Vitis Project Build

Zynq-7000 Introduction

XILINX.

Zynq-7000 Architecture Overview

- Complete ARM®-based processing system
  - Application Processor Unit (APU)
    - ARM Cortex™-A9 processor
    - Caches and support blocks
  - Fully integrated memory controllers
  - I/O peripherals
- Tightly integrated programmable logic
  - Used to extend the processing system
  - Scalable density and performance
- Flexible array of I/O
  - Wide range of external multi-standard I/O
  - High-performance integrated serial transceivers
  - Analog-to-digital converter inputs



XILINX.

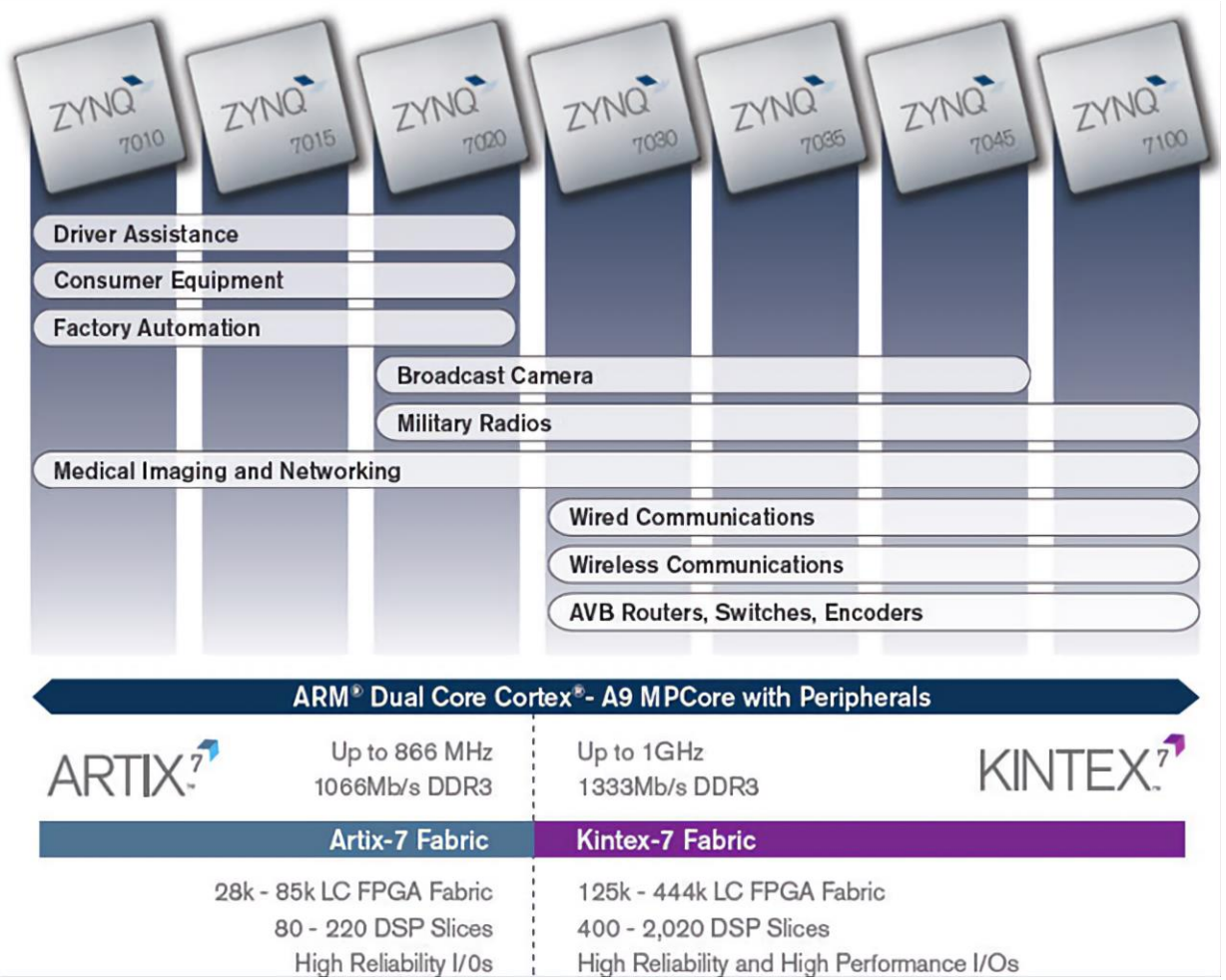
Zynq-7000 Architecture Overview

- Zynq-7000 Family

		Cost-Optimized Devices						Mid-Range Devices			
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System (PS)	Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz			Dual-Core ARM Cortex-A9 MPCore Up to 866MHz			Dual-Core ARM Cortex-A9 MPCore Up to 1GHz[1]			
	Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor									
	L1 Cache	32KB Instruction, 32KB Data per processor									
	L2 Cache	512KB									
	On-Chip Memory	256KB									
	External Memory Support[2]	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support[2]	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to PL)									
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA[2]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security[3]	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave, 4x AXI 64b/32b Memory, AXI 64b ACP, 16 Interrupts									
Programmable Logic (PL)	7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
	Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Total Block RAM	1.8Mb	2.5Mb	3.8Mb	2.1Mb	3.3Mb	4.9Mb	9.3Mb	17.6Mb	19.2Mb	26.5Mb
	(# 36Kb Blocks)	(50)	(72)	(107)	(60)	(95)	(140)	(265)	(500)	(545)	(755)
	DSP Slices	66	120	170	80	160	220	400	900	900	2,020
	PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC[2]	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[3]	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
	Speed Grades: Commercial	-1			-1			-1			-1
	Speed Grades: Extended	-2			-2,-3			-2,-3			-2
	Speed Grades: Industrial	-1, -2			-1, -2, -1L			-1, -2, -2L			-1, -2, -2L

Zynq-7000 Architecture Overview

- The Zynq-7000 AP SoC architecture consists of two major sections
  - PS: Processing system
    - ARM Cortex-A9 processor based
    - Multiple peripherals
    - Hard silicon core
  - PL: Programmable logic
    - Uses the same 7 series programmable logic
      - Artix™-based devices: Z-7010, Z-7015 and Z-7020 (high-range I/O banks only)
      - Kintex™-based devices: Z-7030, Z-7035, Z-7045, and Z-7100 (mix of high-range and high-performance I/O banks)



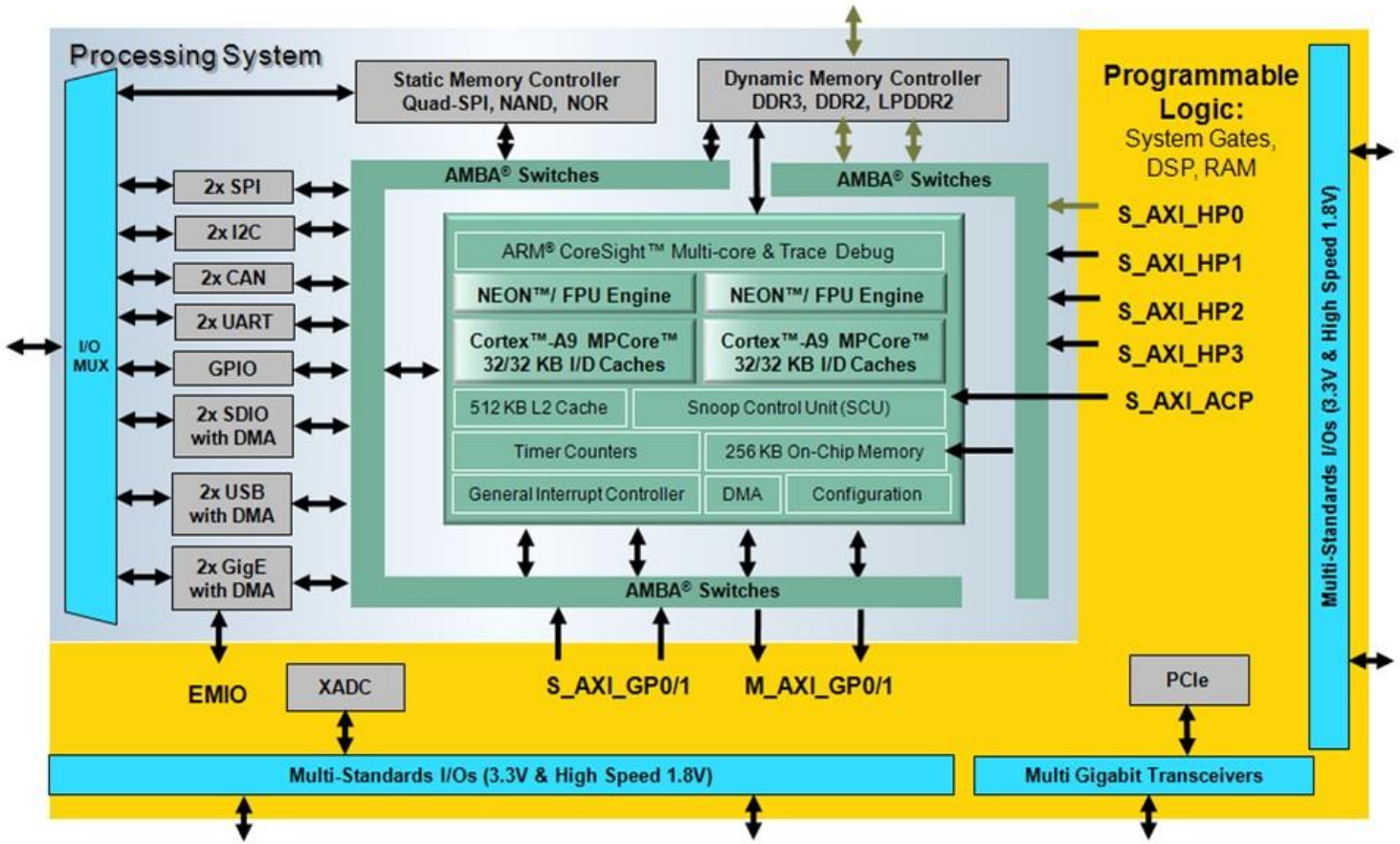
ARM Processor Product Families

- Legacy ARM processors
  - ARM7, ARM9 (not the Cortex-A9 processor), ARM11
- Cortex family of processors
  - **Cortex-A#: "A" application**
    - The products support a memory management unit (MMU)
    - Excellent for operating systems
  - **Cortex-R#: "R" real time**
    - The products support a memory protection Unit (MPU)
    - Better determinism than an MMU
  - **Cortex-M#: "M" Embedded microcontroller**

Zynq-7000 Architecture Overview

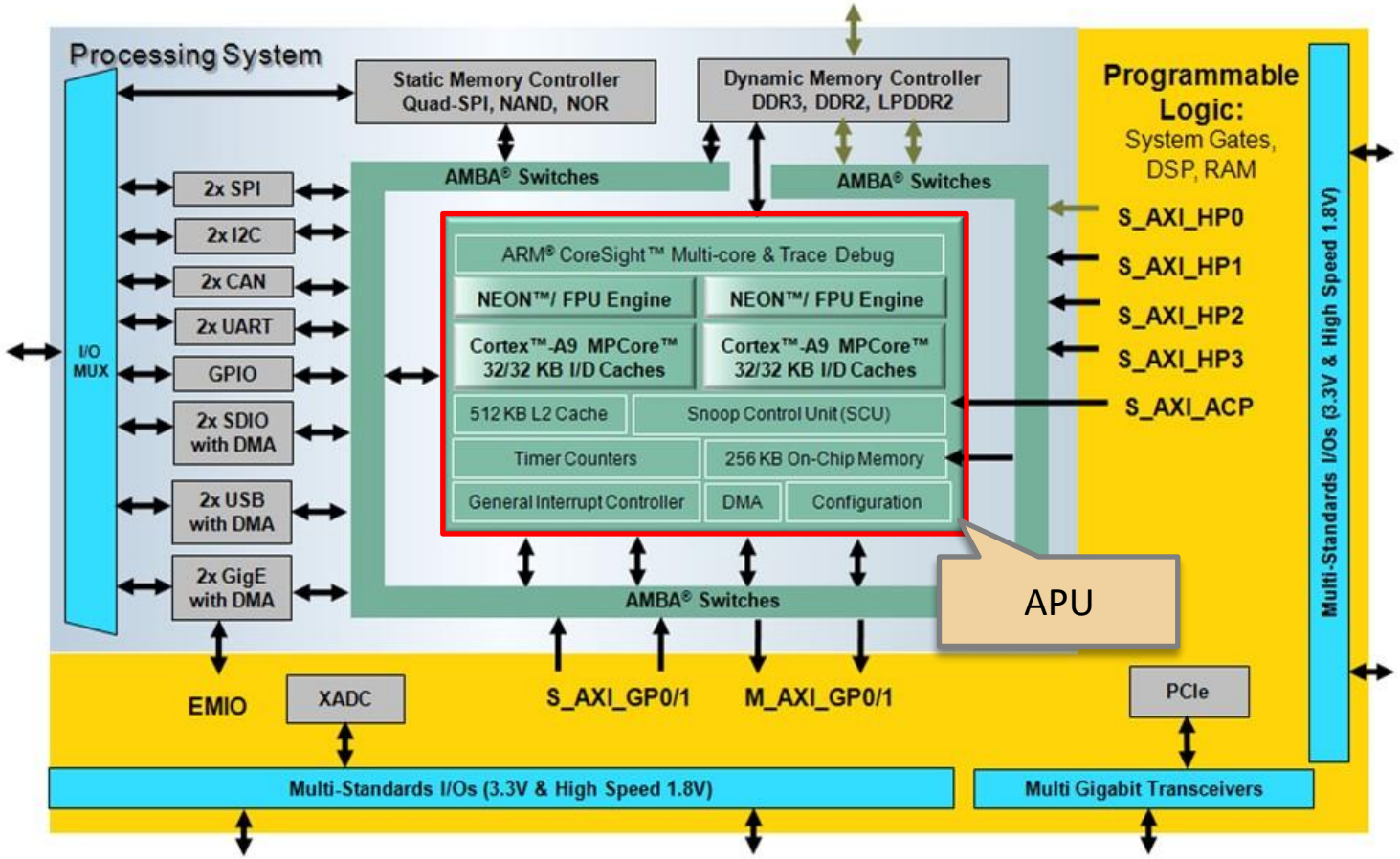
- Processing System (PS) Components

- Application processor unit (APU)
- Memory interfaces
- I/O peripherals (IOP)
- Interconnect



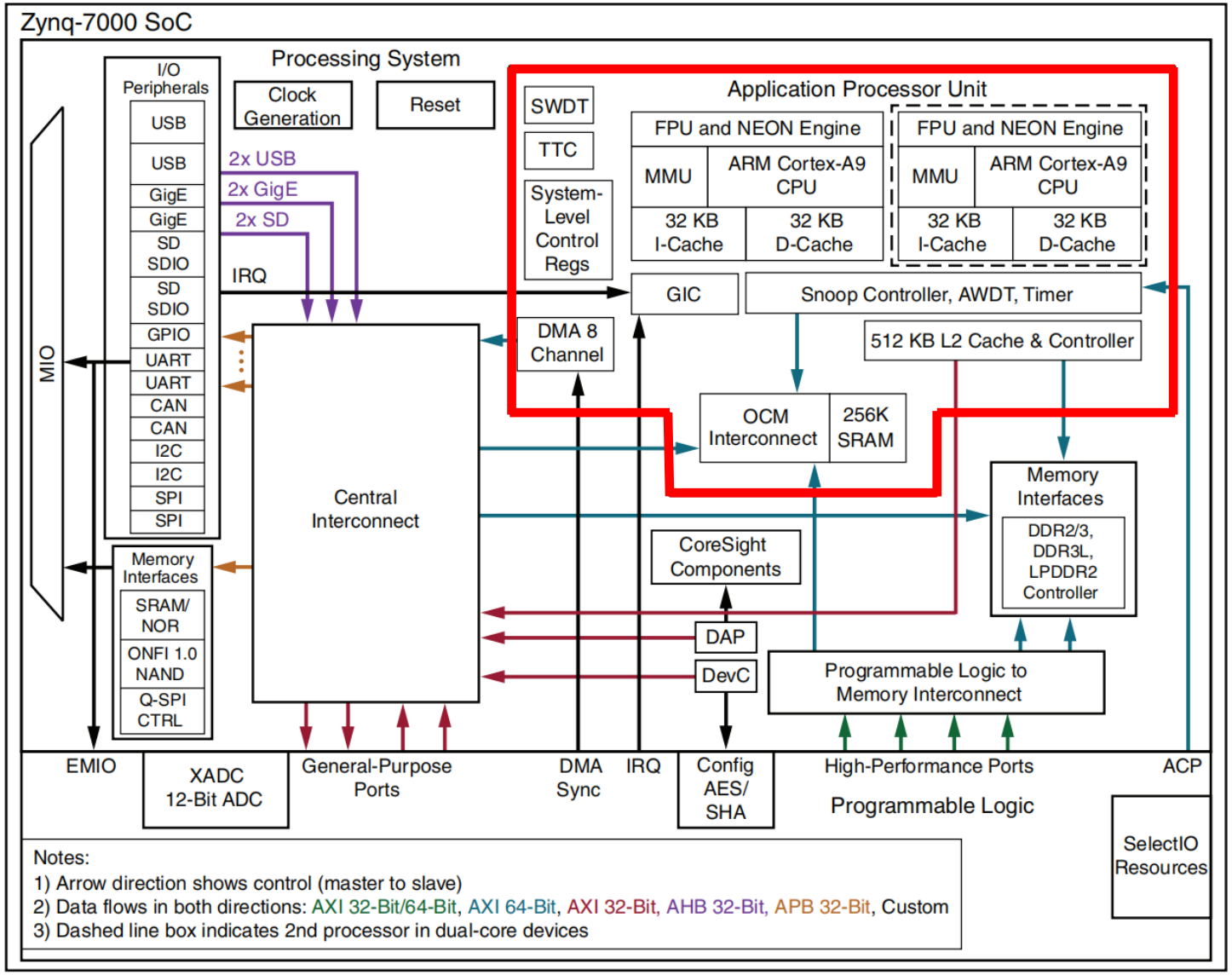
Processing System (PS) Components

- Application processor unit (APU)
- Memory interfaces
- I/O peripherals (IOP)
- Interconnect



Processing System (PS) Components

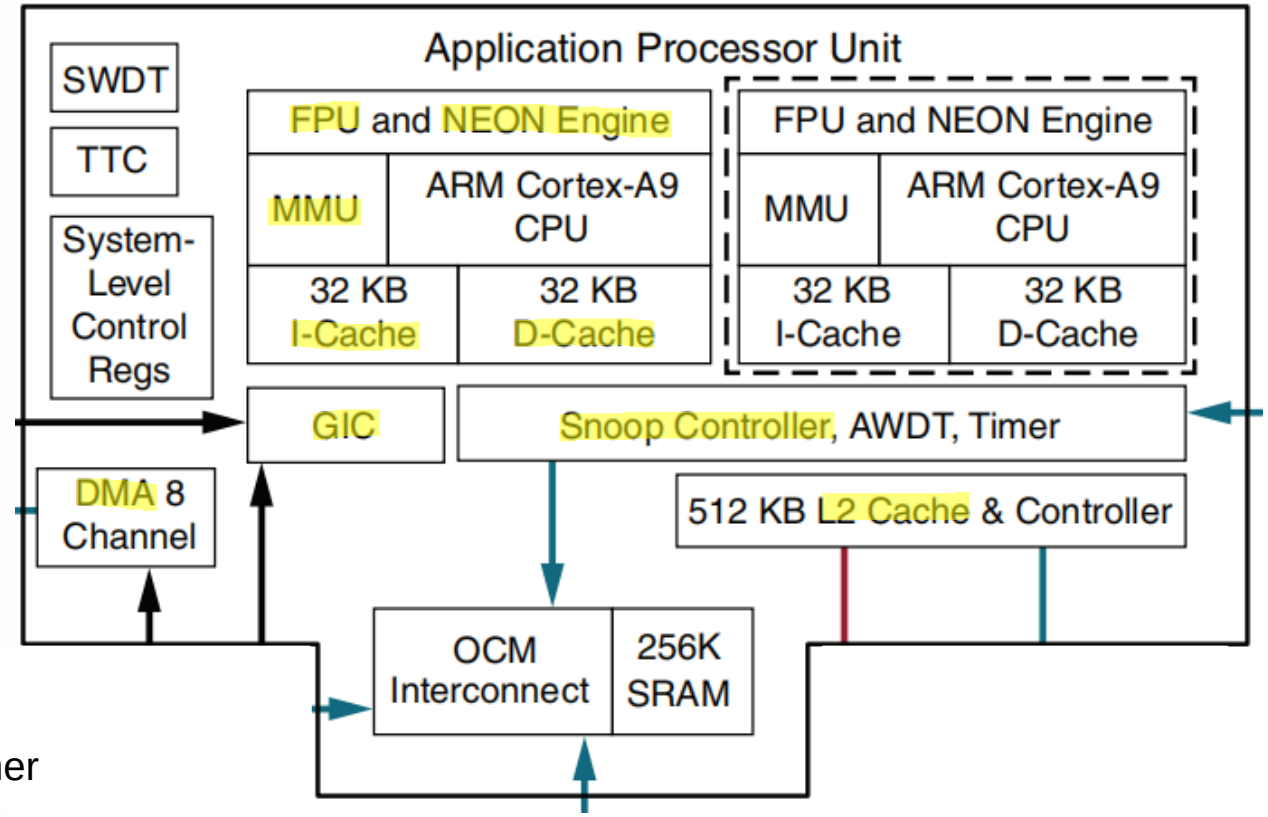
- Application Processing Unit (APU)



Processing System (PS) Components

- Application Processing Unit (APU)

1. Single/Dual A9 core
    1. FPU and NEON Engine（Float Point Unit & NEON）
    2. MMU（Memory Management Unit）
    3. A9 CPU（central processing unit）
    4. 32KB I-Cache（Instruction Cache）
    5. 32KB D-Cache（Date Cache）
2. SWDT (System Watch Dog Timer)
3. TTC (Triple Timer/Counter)
4. System-Level Control Regs
5. DMA 8 Channel（Direct Memory Access）
6. GIC（ General interrupt controller）
7. Snoop Controller, AWDT (ARM Watch Dog Timer), Timer
8. 512KB L2 Cache & Controller
9. OCM Interconnect 256K SRAM (On-Chip Memory)



Processing System (PS) Components

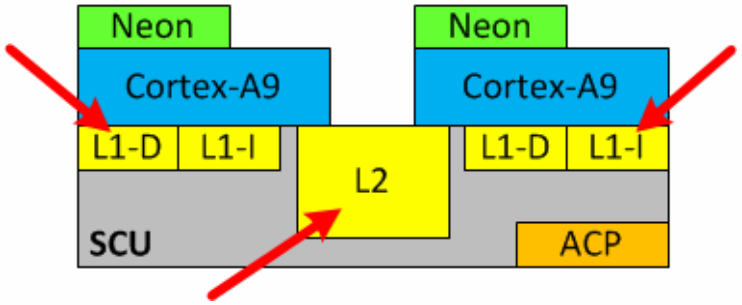
- Application Processing Unit (APU)

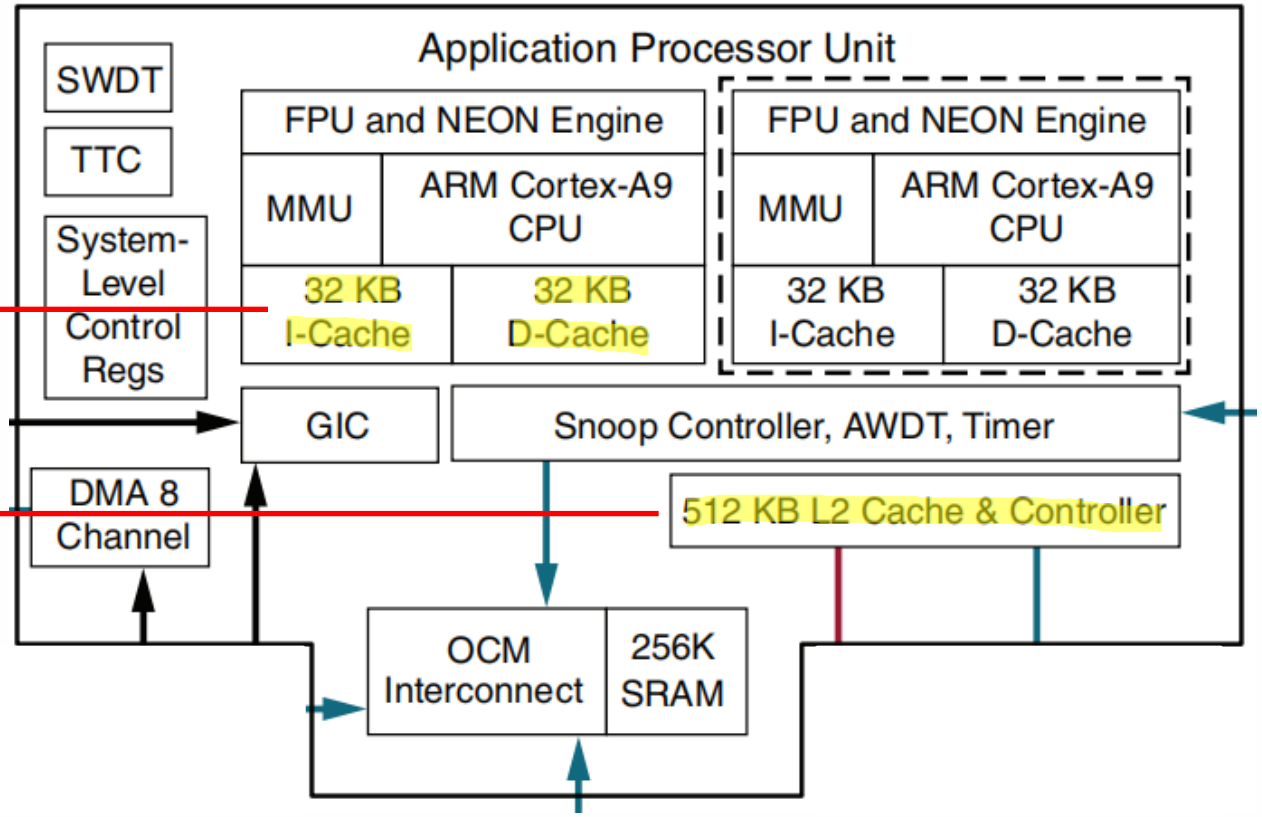
L1 cache

- Smallest size but fastest access
- 32 KB each for instruction and data
- One I-cache and D-cache per CPU

L2 cache

- Secondary, larger cache
- 512 KB
- Used for both instruction and data
- One per snoop control unit (SCU)

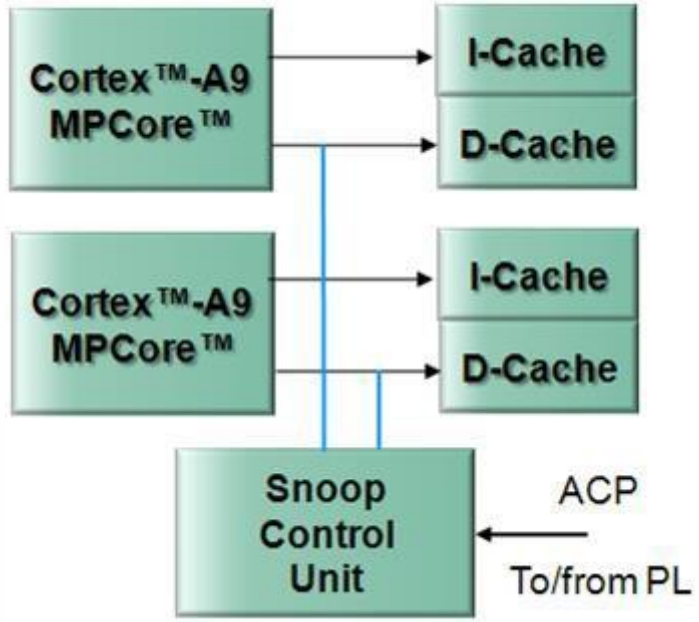


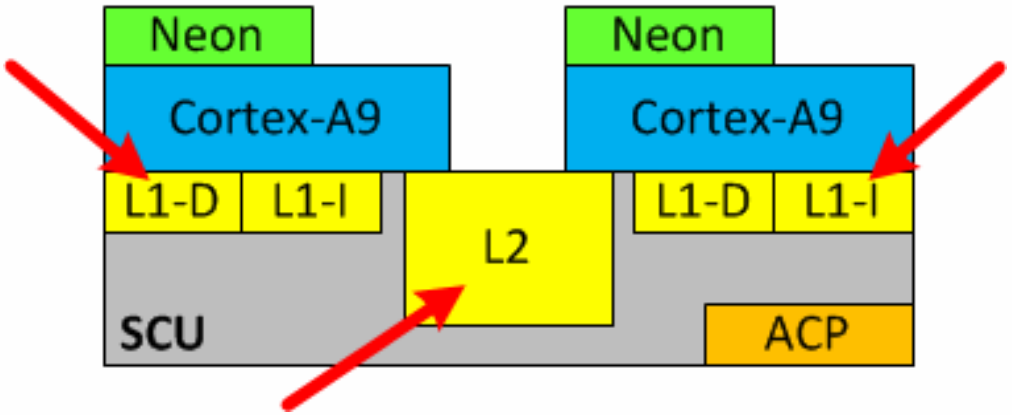


XILINX.

Snoop Control Unit (SCU)

- Shares and arbitrates functions between the two processor cores
  - Data cache coherency between the processors
  - Initiates L2 AXI memory access
  - Arbitrates between the processors requesting L2 accesses
  - Manages ACP accesses
  - A second master port with programmable address filtering between OCM and L2 memory support



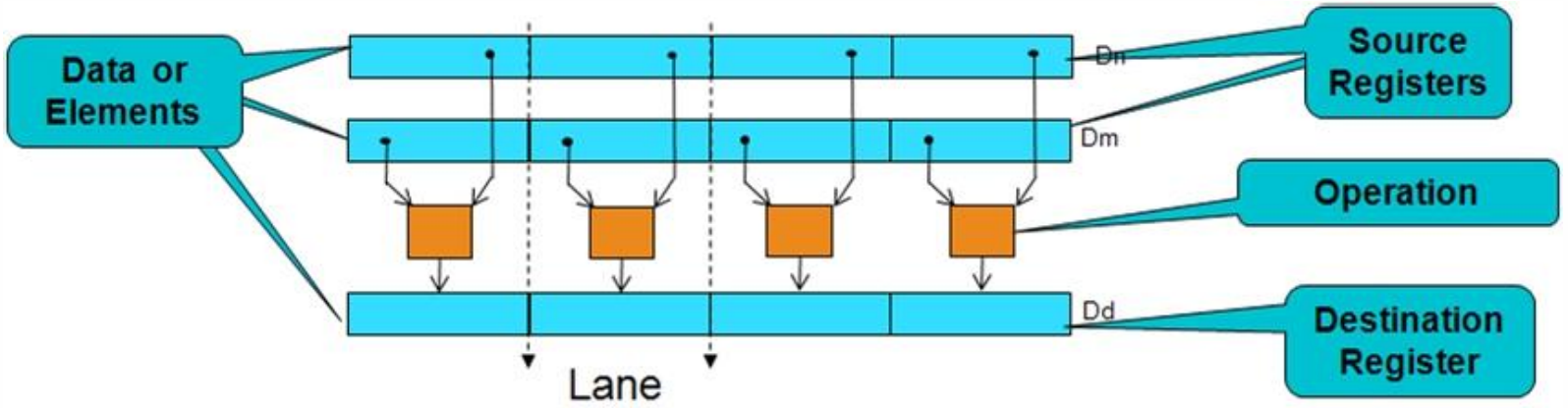


NEON Instruction

- Vector Processing using NEON

- NEON is the ARM codename for the vector processing unit
  - Provides multimedia and signal processing support
- FPU is the floating-point unit extension to NEON
  - Both NEON and FPU share a single set of registers
- NEON technology is a wide single instruction, multiple data (SIMD) parallel and co-processing architecture
  - 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide)
  - Data types can be signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, or 32-bit float

NEON™/FPU Engine

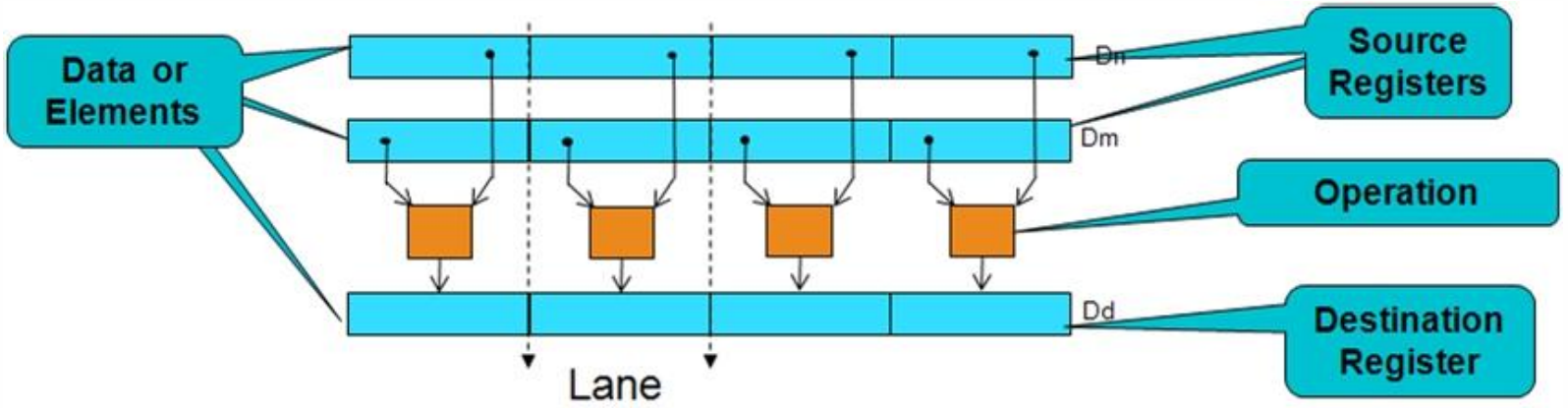


NEON Instruction

- Vector Processing using NEON

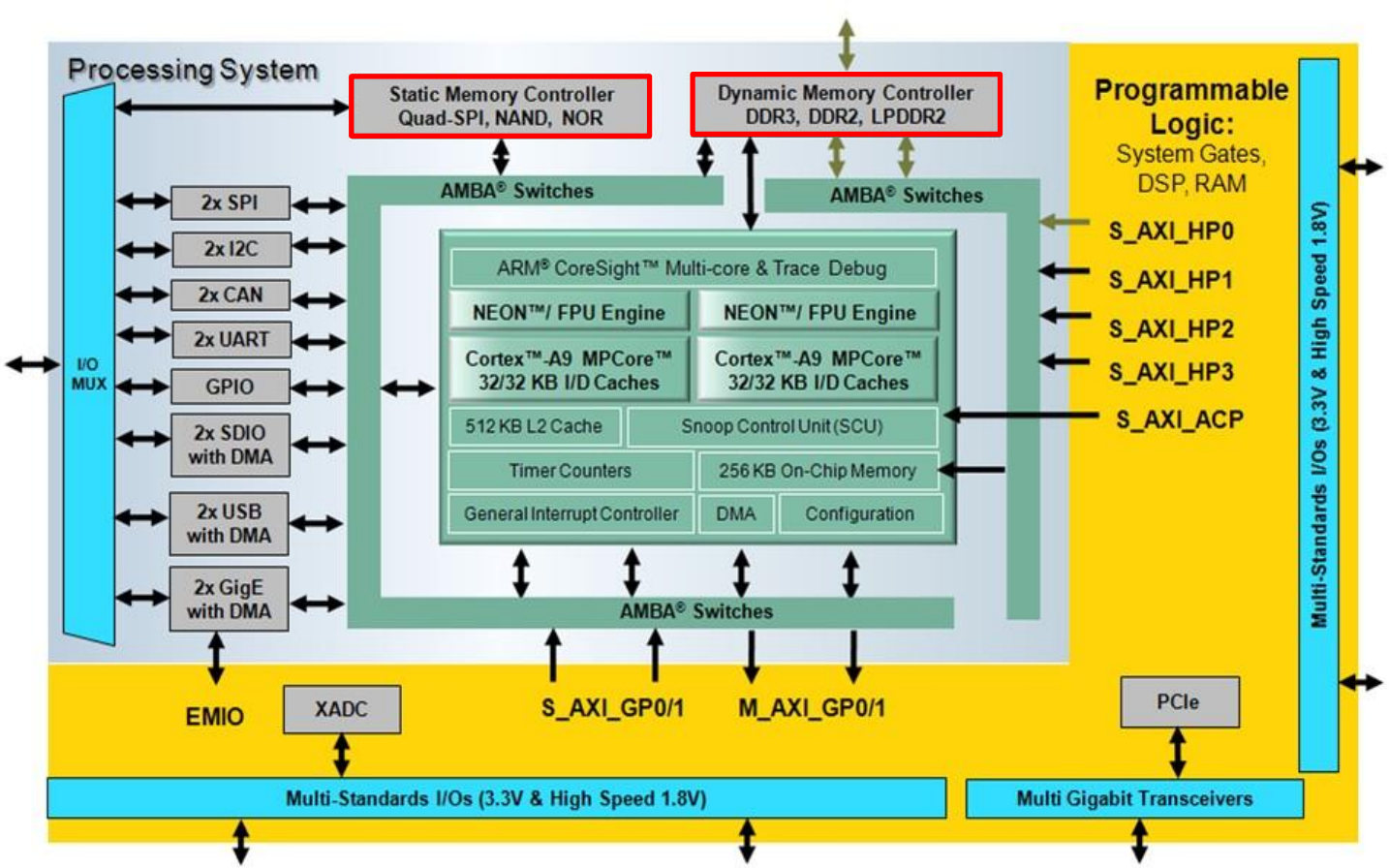
- Block-based data processing
- Audio, video, and image processing codes
- 2D graphics based on rectangular blocks of pixels
- 3D graphics
- Color-space conversion
- Physics simulations

NEON™/FPU Engine



Processing System (PS) Components

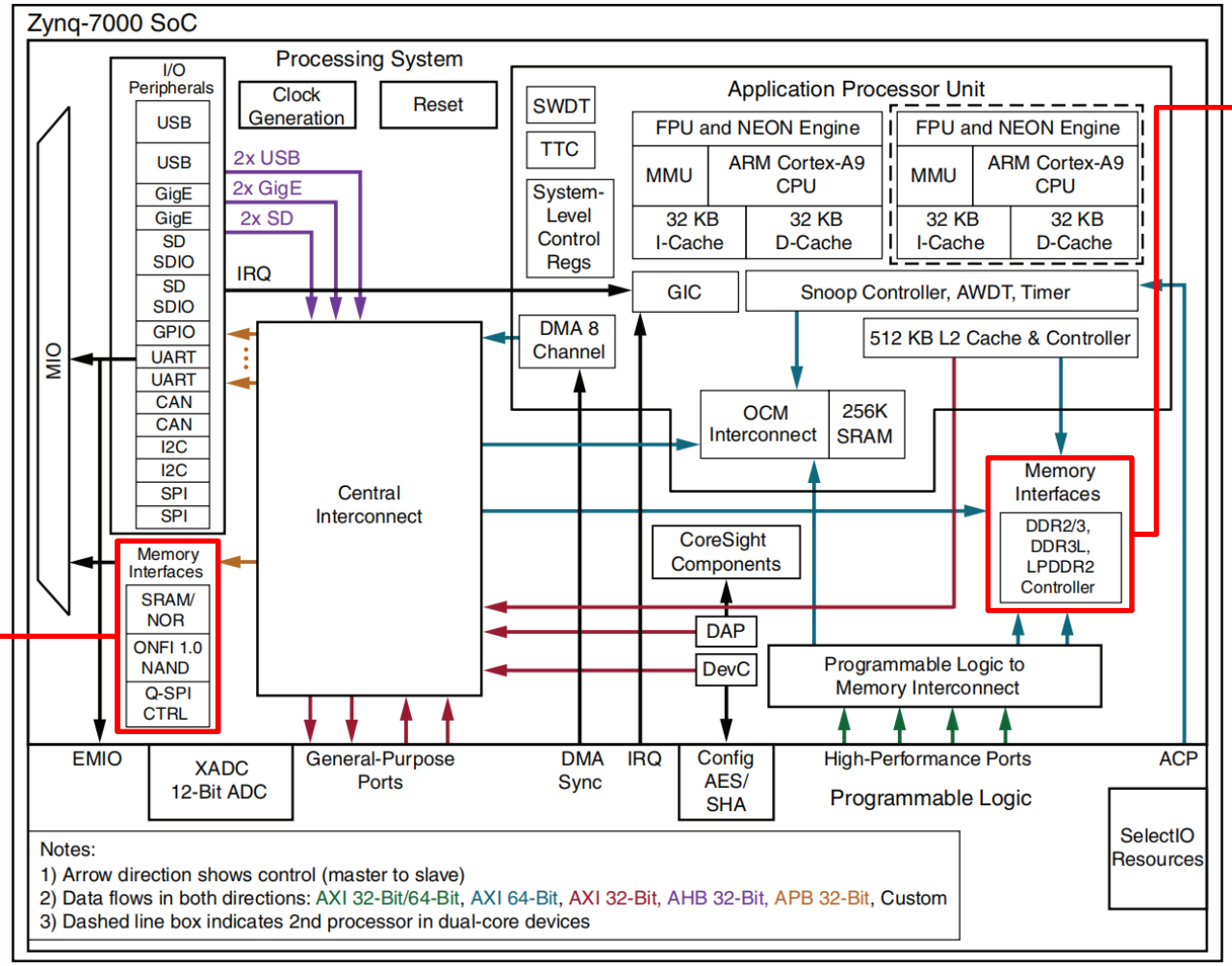
- Application processor unit (APU)
- Memory interfaces
- I/O peripherals (IOP)
- Interconnect



Processing System (PS) Components

- Memory interfaces

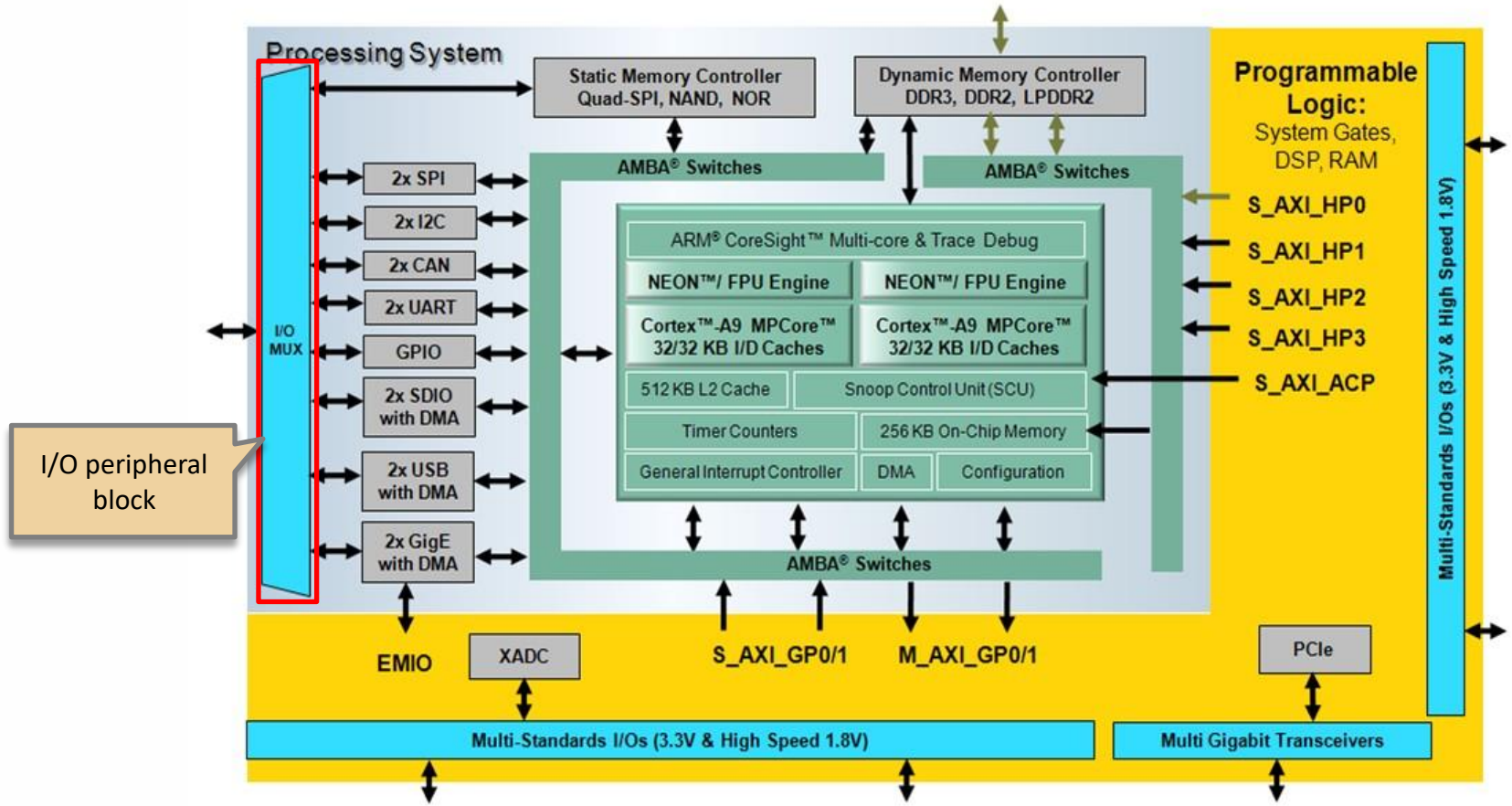
1. SRAM/NOR，SRAM interface
2. ONFI 1.0 NAND
   （Open Nand Flash Interface）
3. Q-SPI CTRL
   （Qual SPI control，quad-channel SPI FLASH interface）



1. DDR2/3
2. DDR3L
3. LPDDR2

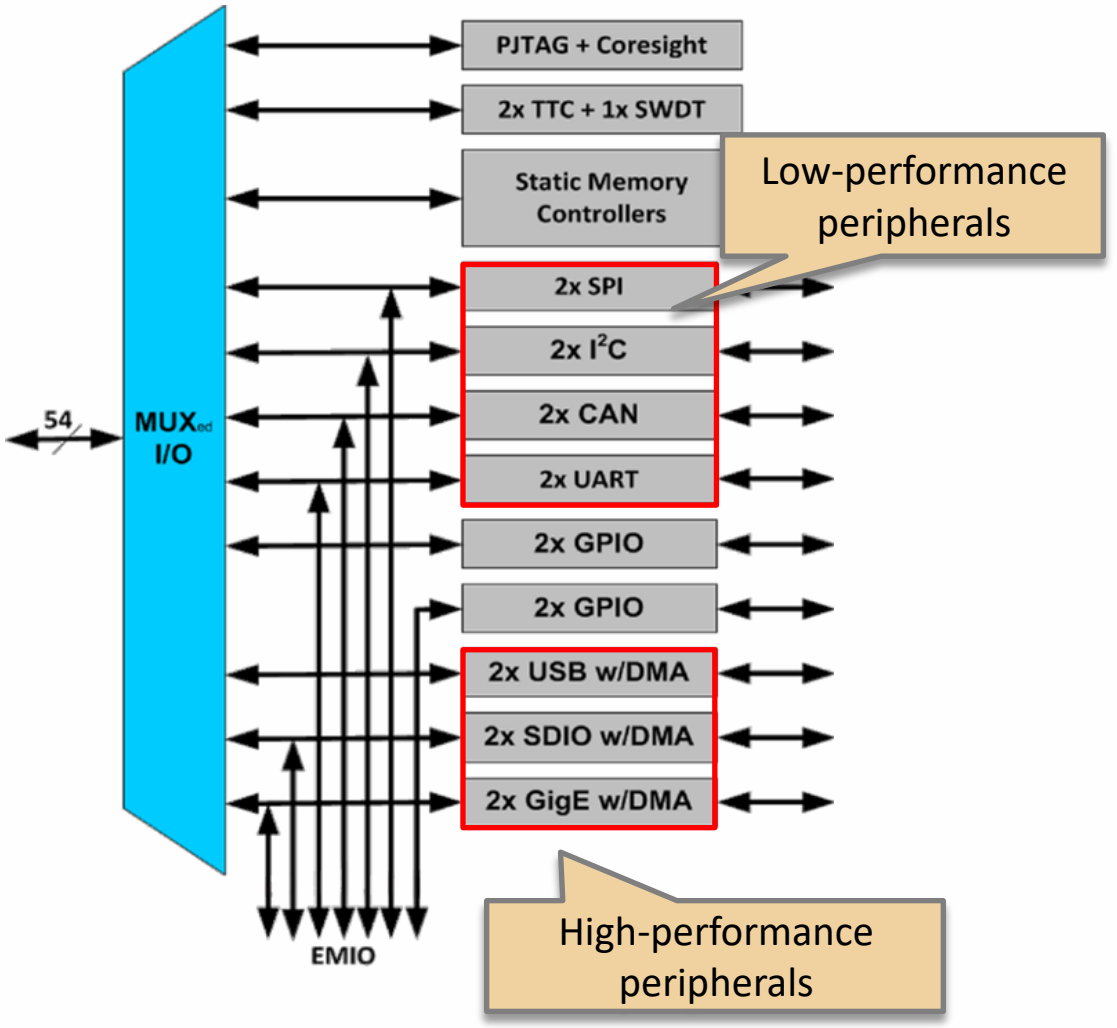
Processing System (PS) Components

- Application processor unit (APU)
- Memory interfaces
- I/O peripherals (IOP)
- Interconnect



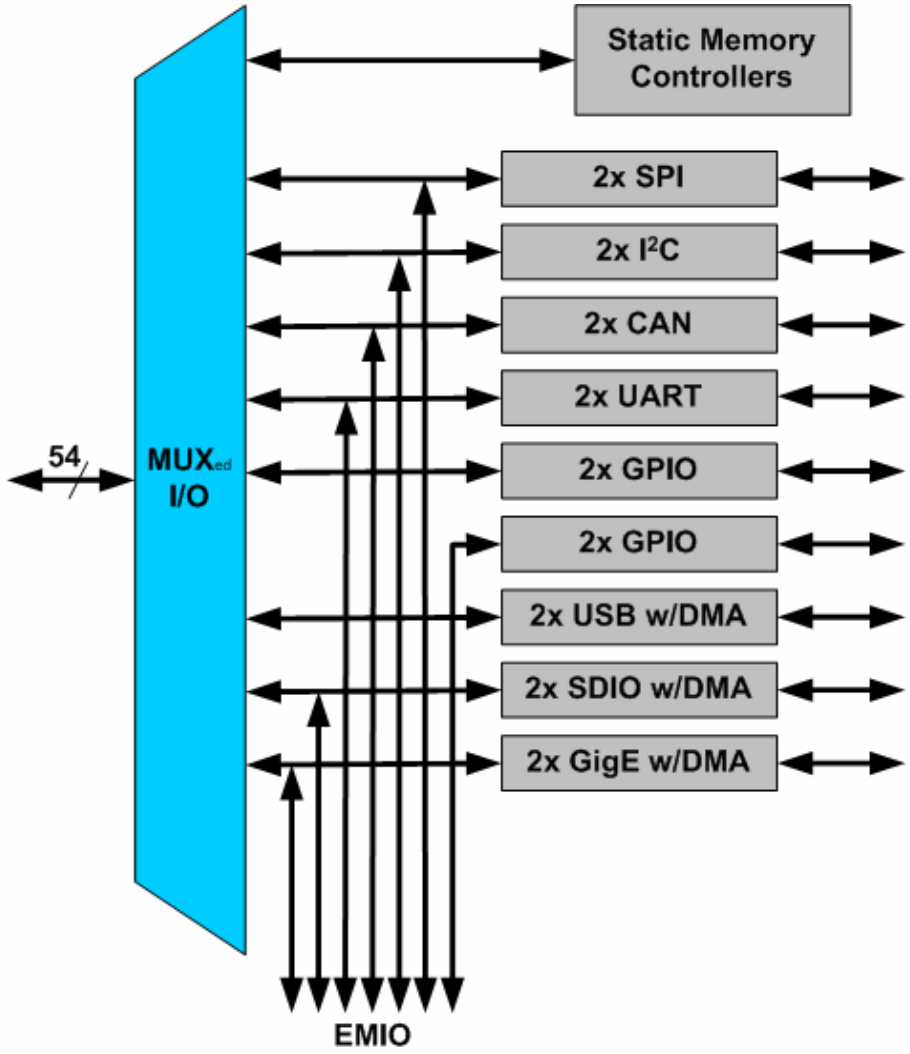
Built-in Peripherals

- Two USB 2.0 OTG/device/host
- Two tri-mode gigabit Ethernet (10/100/1000)
- Two SD/SDIO interfaces
  - Memory, I/O, and combo cards
- Two CAN 2.0Bs, SPIs, I2Cs, UARTs
- Four GPIO 32-bit blocks
  - 54 available through MIO; other 64 available through EMIO



Built-in Peripherals

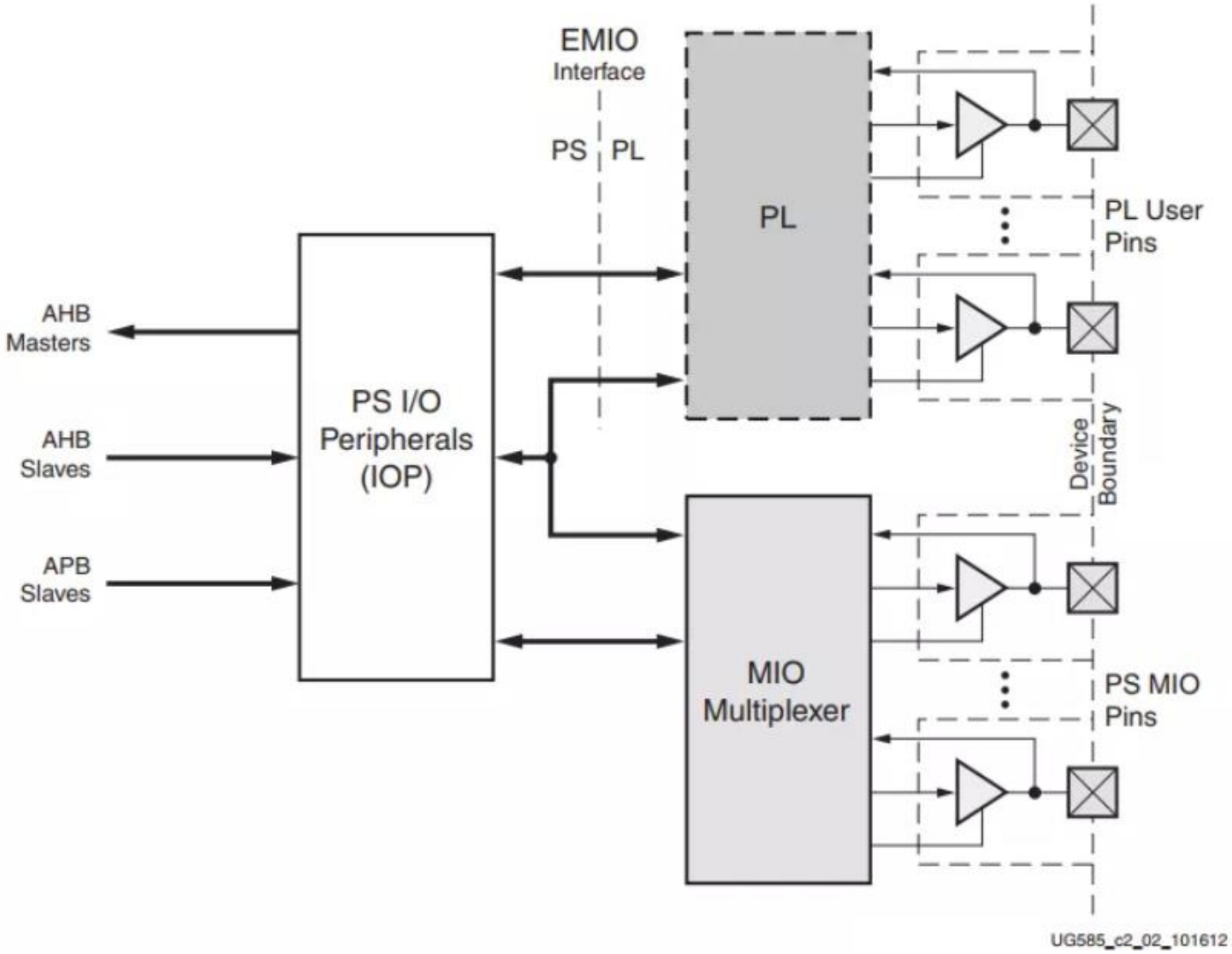
- Multiplexed input/output (MIO)
  - Multiplexed output of peripheral and static memories
  - Two I/O banks; each selectable: 1.8V, 2.5V, or 3.3V
  - Configured using configuration
  - Dedicated pins are used
- Extended MIO
  - Enables use of the SelectIO™ interface with PS peripherals
  - User constraints must be present for the signals brought out to the SelectIO pins



MIO vs. EMIO

IP	MIO	Extendable MIO in Programmable Logic
QSPI NOR/SRAM NAND	Yes	No
USB: 0,1	Yes, Phy off chip	No
SDIO: 0,1	Yes – 50MHz	Yes – 25MHz
SPI: 0,1 I2C: 0,1 CAN: 0,1 GPIO	Yes	Yes
GigE: 0,1	RGMII v2.0 (HSTL) Phy off chip	Supports GMII, RGMII v2.0 (HSTL), RGMII v1.3 (LVCMOS), RMII, MII, SGMII with wrapper in Programmable Logic
UART: 0,1	Simple UART: Only 2 pins (Tx & Rx)	Full UART (Tx, Rx, DTR, DCD, DSR, RI, RTS & CTS) either require: • 2 Processing System pins (Rx & Tx) through MIO + 6 additional Programmable Logic pins • 8 Programmable Logic pins

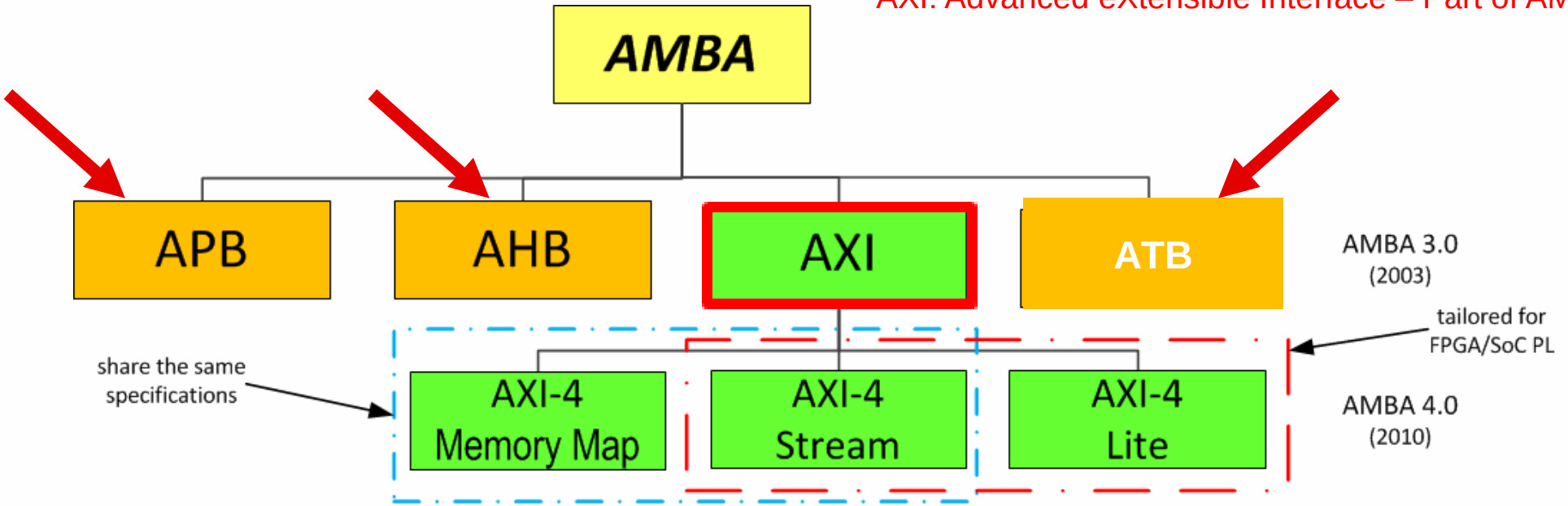
MIO vs. EMIO



Advanced eXtensible Interface (AXI) Bus

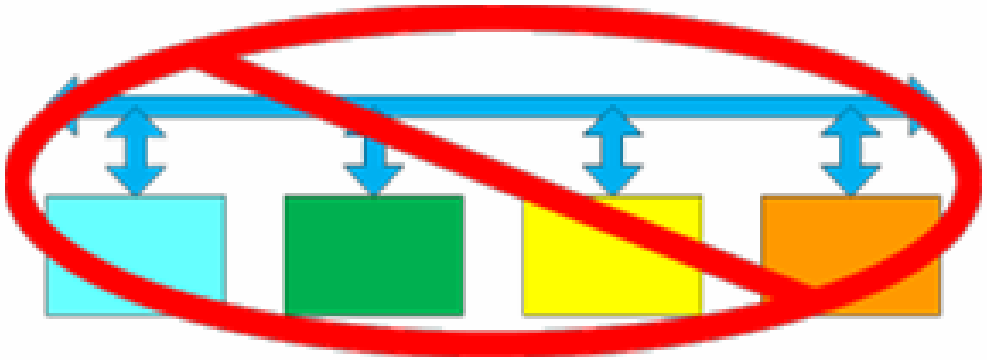
AMBA: Advanced Microcontroller Bus Architecture

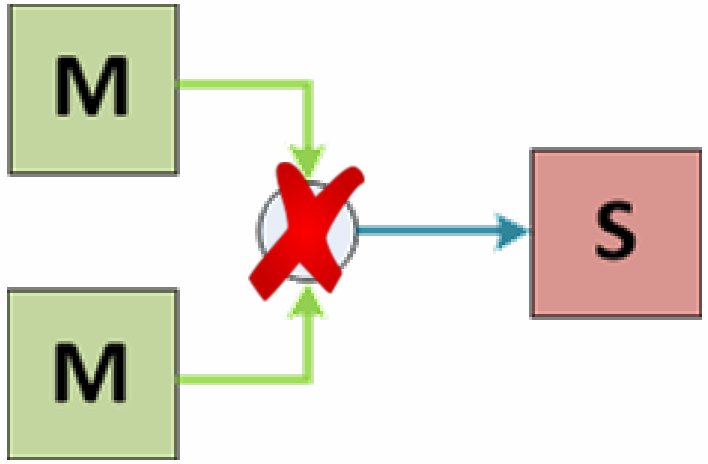
AXI: Advanced eXtensible Interface – Part of AMBA



- APB, AHB, and ATB are used for internal Arm® core topologies, AXI is targeted towards processor peripherals and other IP through the programmable logic.
- Unified interface protocol of Xilinx components.

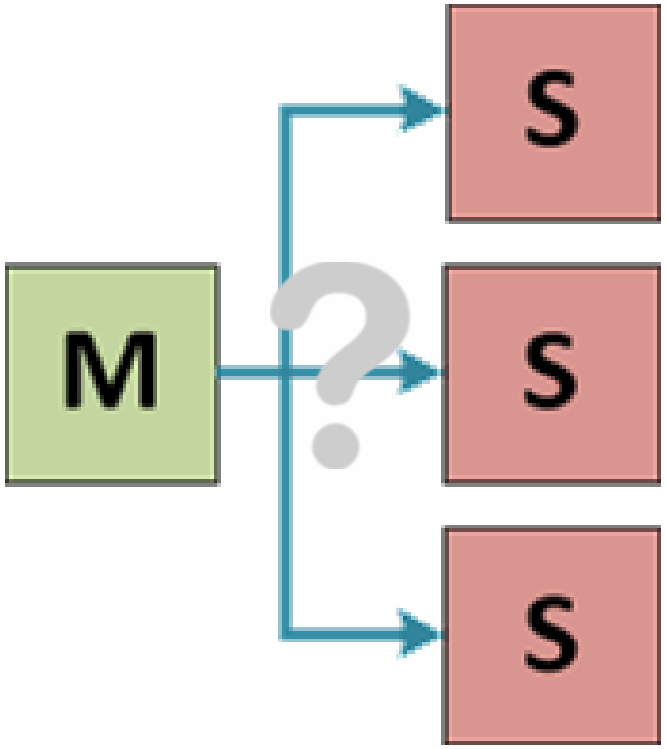
Advanced eXtensible Interface (AXI) Bus





- AXI defines a point-to-point connection, there are never any bus arbitration issues.
- One master port talks to only one slave port.

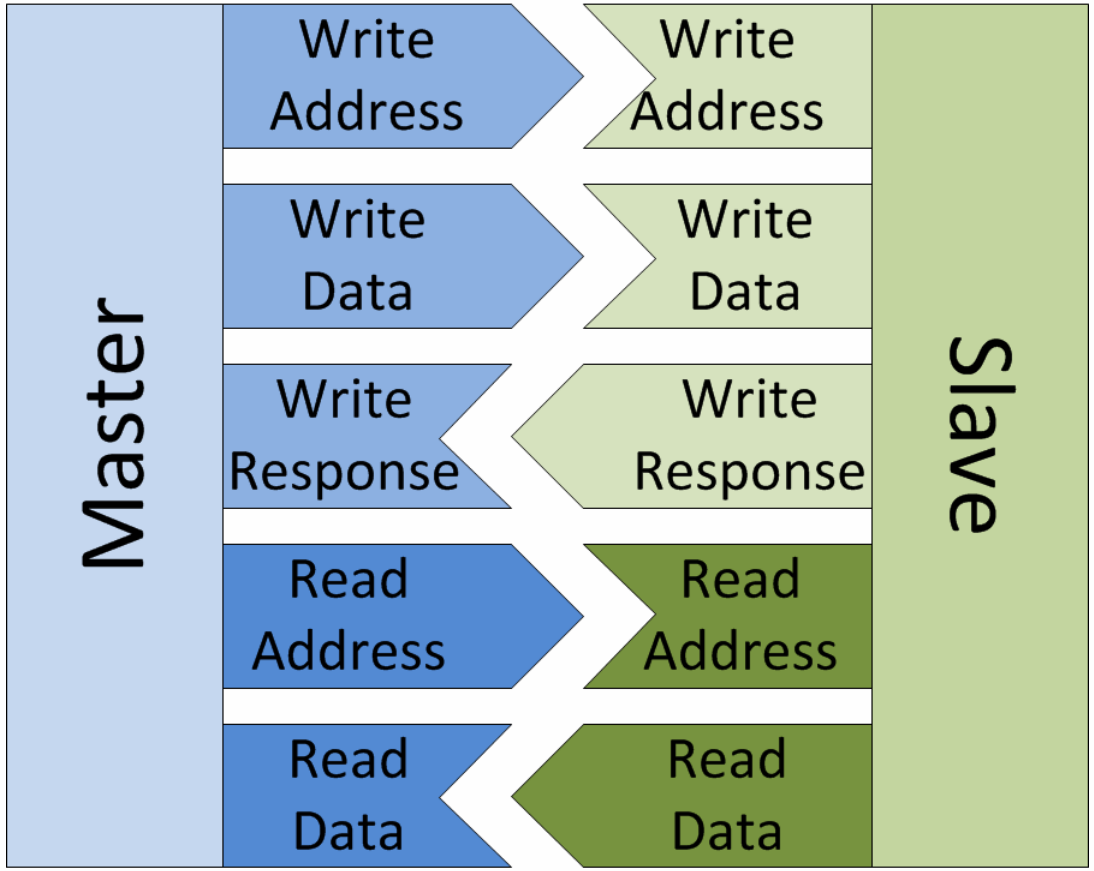
Advanced eXtensible Interface (AXI) Bus



- AXI interconnect IP refers to a piece of IP that resides in the PL and takes its input from a single AXI slave and maps it to one or more AXI masters, This then allows a single PS master port to connect to multiple slave devices.

Advanced eXtensible Interface (AXI) Bus

- Definition
  1. Separate address, control, and data phases
  2. Unaligned data transfers
  3. Burst transactions
  4. Multiple outstanding reads



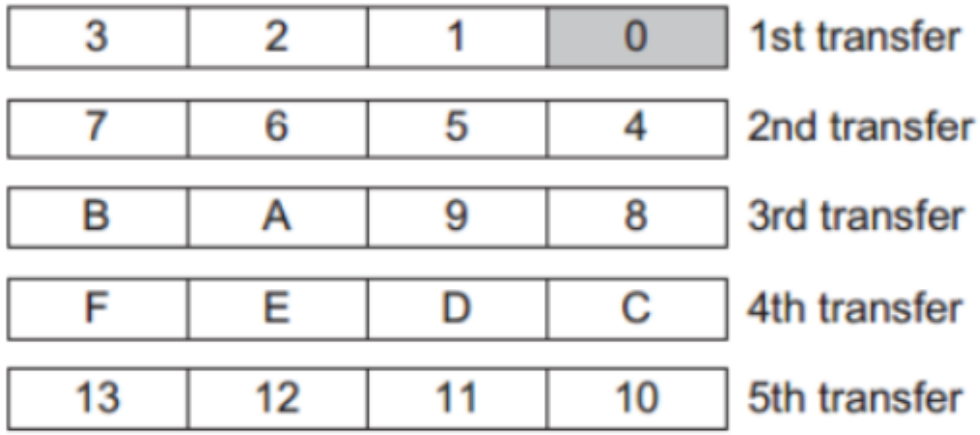
Advanced eXtensible Interface (AXI) Bus

- Definition

1. Separate address, control, and data phases
2. Unaligned data transfers

e.g.

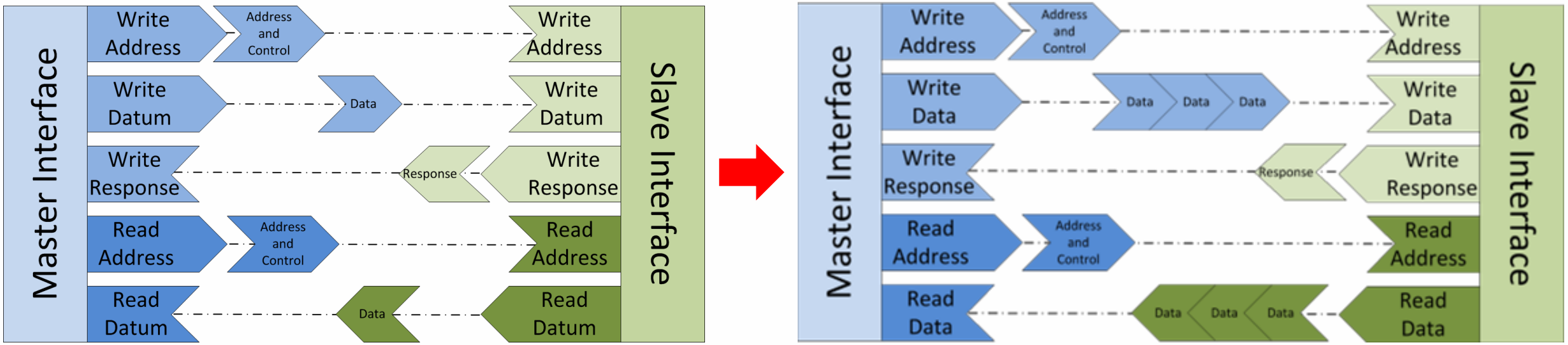
Address: 0x01
Transfer size: 32 bits
Burst type: incrementing
Burst length: 5 transfers



3. Burst transactions
4. Multiple outstanding reads

Advanced eXtensible Interface (AXI) Bus

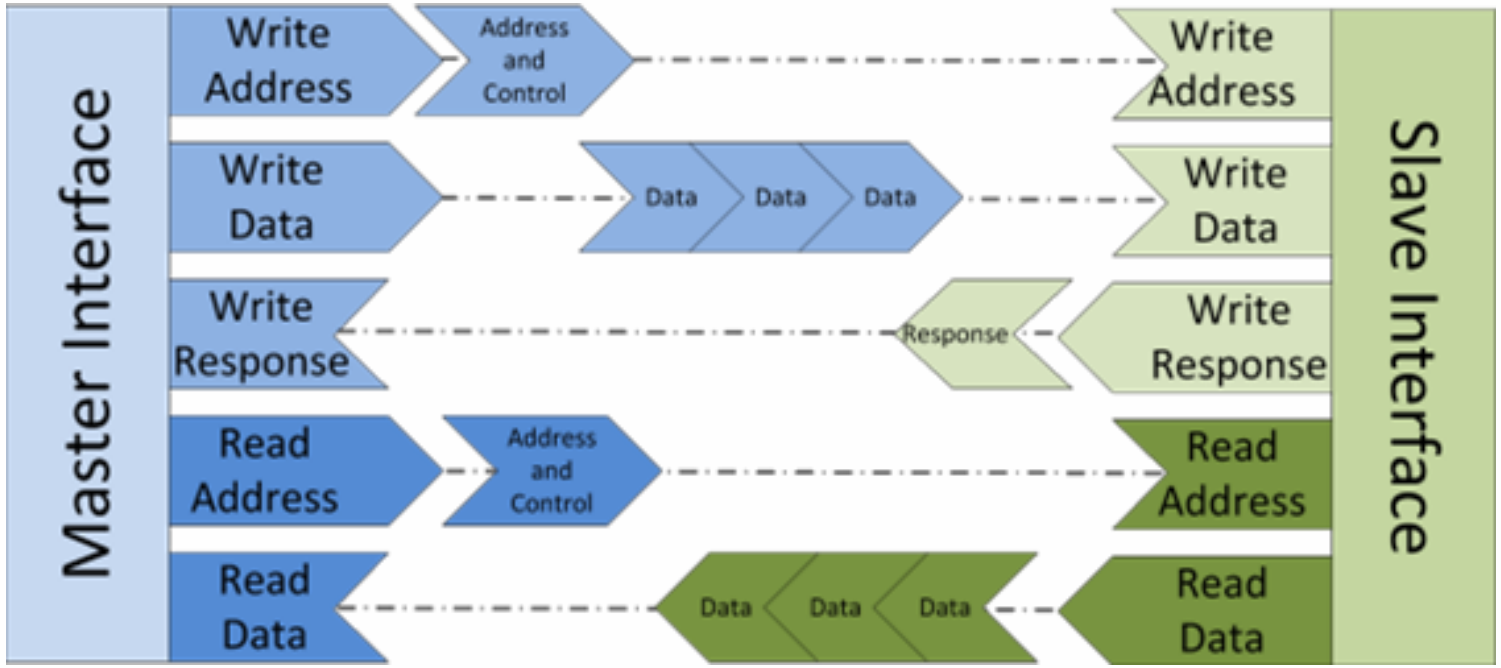
- Definition
  1. Separate address, control, and data phases
  2. Unaligned data transfers
  3. Burst transactions
  4. Multiple outstanding reads



Advanced eXtensible Interface (AXI) Bus

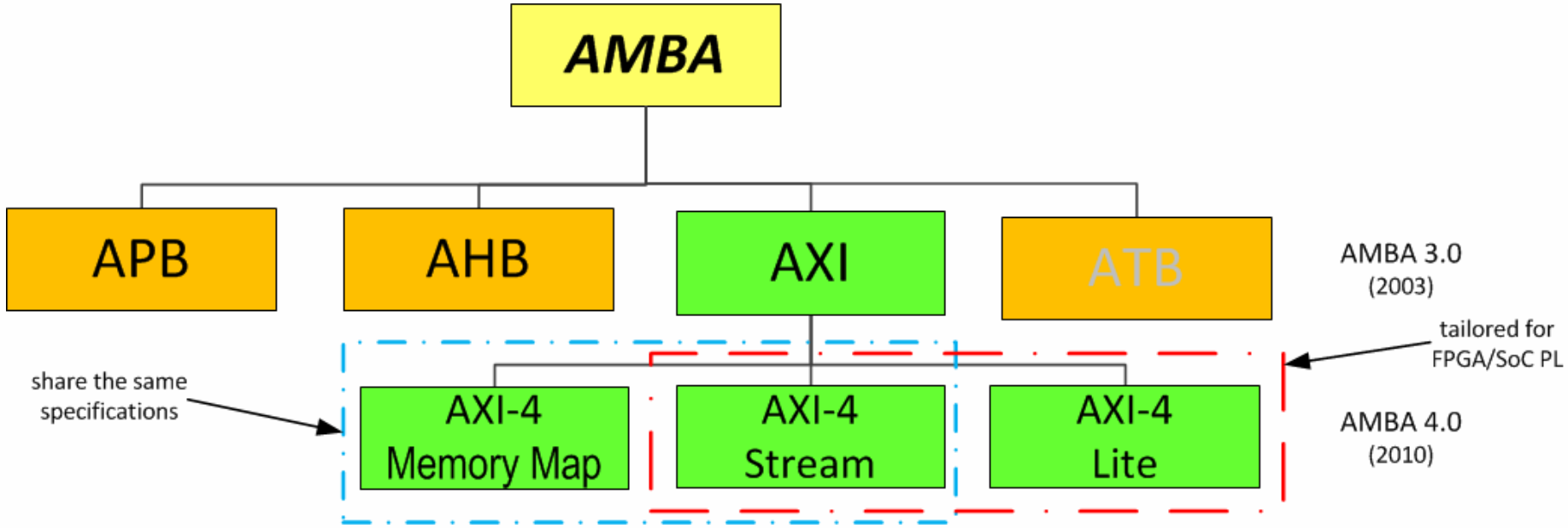
- Definition

1. Separate address, control, and data phases
2. Unaligned data transfers
3. Burst transactions
4. Multiple outstanding reads



XILINX.

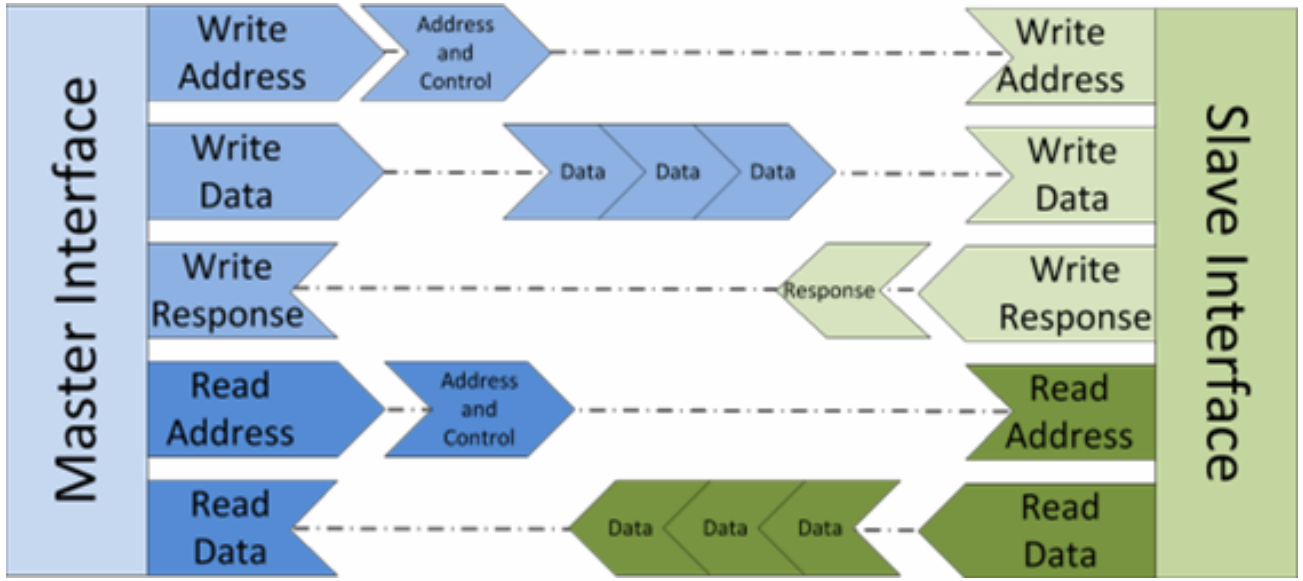
Advanced eXtensible Interface (AXI) Bus

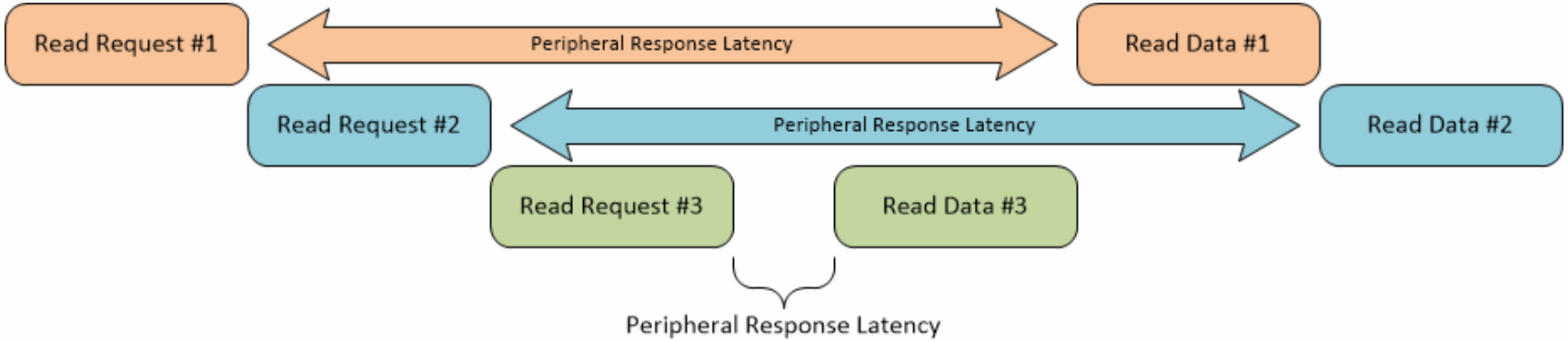


Interface	Features
Memory Map / Full	Traditional address/data burst (single address, multiple data)
Streaming	Data only, streaming, usually are used for video and audio data transfer
Lite	Traditional address/data—no burst (single address, single data only)

Advanced eXtensible Interface (AXI) Bus

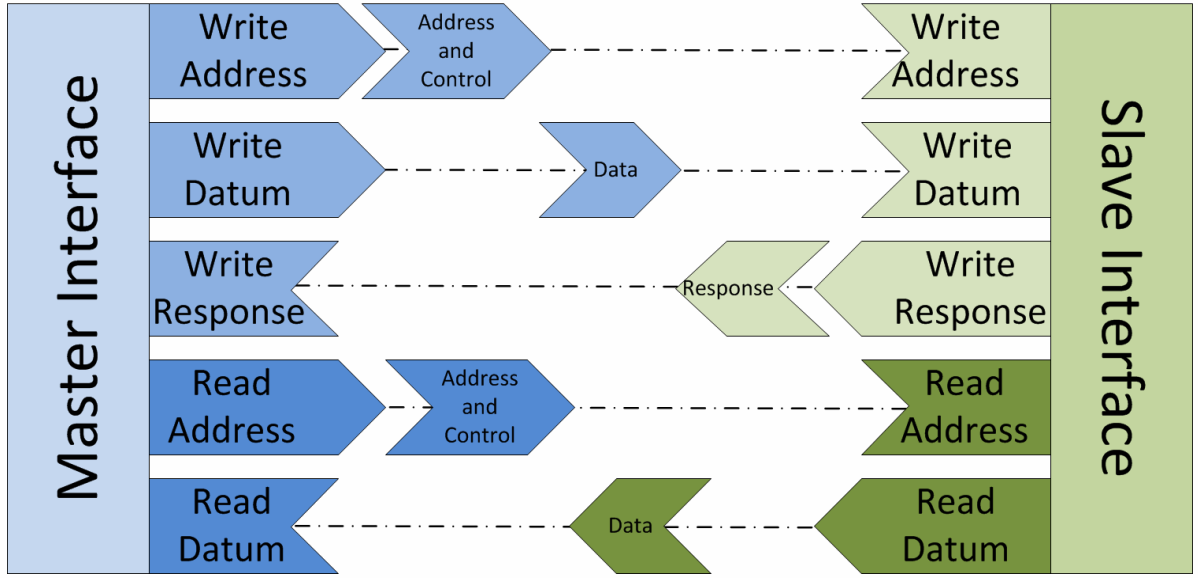
- **AXI Memory Map / Full**
  1. Single address multiple data
  2. Burst up to 256 data beats
  3. Data width parameterizable
     - ➢ 32, 64, 128, 256, 512, 1024 bits
  4. Support for overlapped transactions





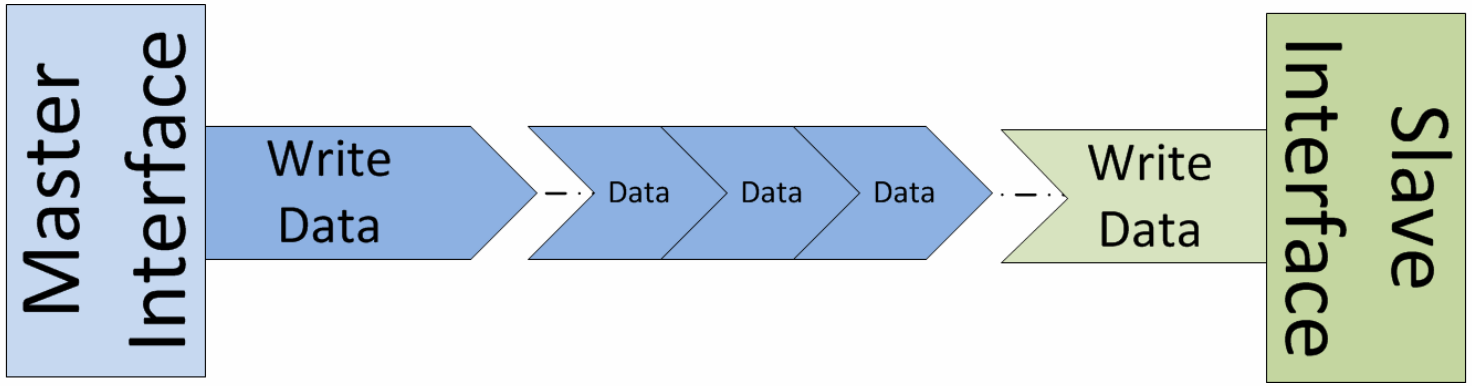
Advanced eXtensible Interface (AXI) Bus

- **AXI Lite**
  1. Subset of the AXI interface
  2. Supports simple component interfaces, save power and spaces
  3. Main purpose
     - ➢ Enable the processors to communicate control signals and messages with peripheral registers.
  4. Implementation widths are selectable as either 32-bits wide or 64-bits wide.



Advanced eXtensible Interface (AXI) Bus

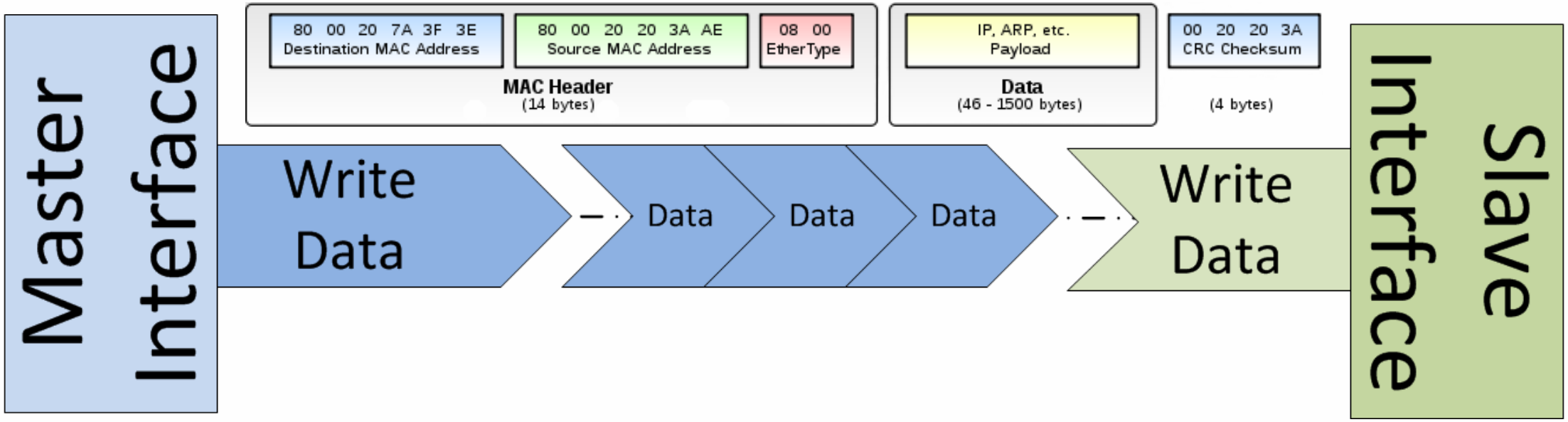
- **AXI Streaming**
  1. Transfer – Single datum moving across AXI-Stream interface. Defined by TVALID-TREADY handshake
  2. Packet – Group of bytes, is like an AXI “burst”
  3. Frame – Integer number of packets
  4. Stream – Transport of data from source to destination



1. Data Only
2. Minimum control
3. Unlimited burst length
4. Only a single direction

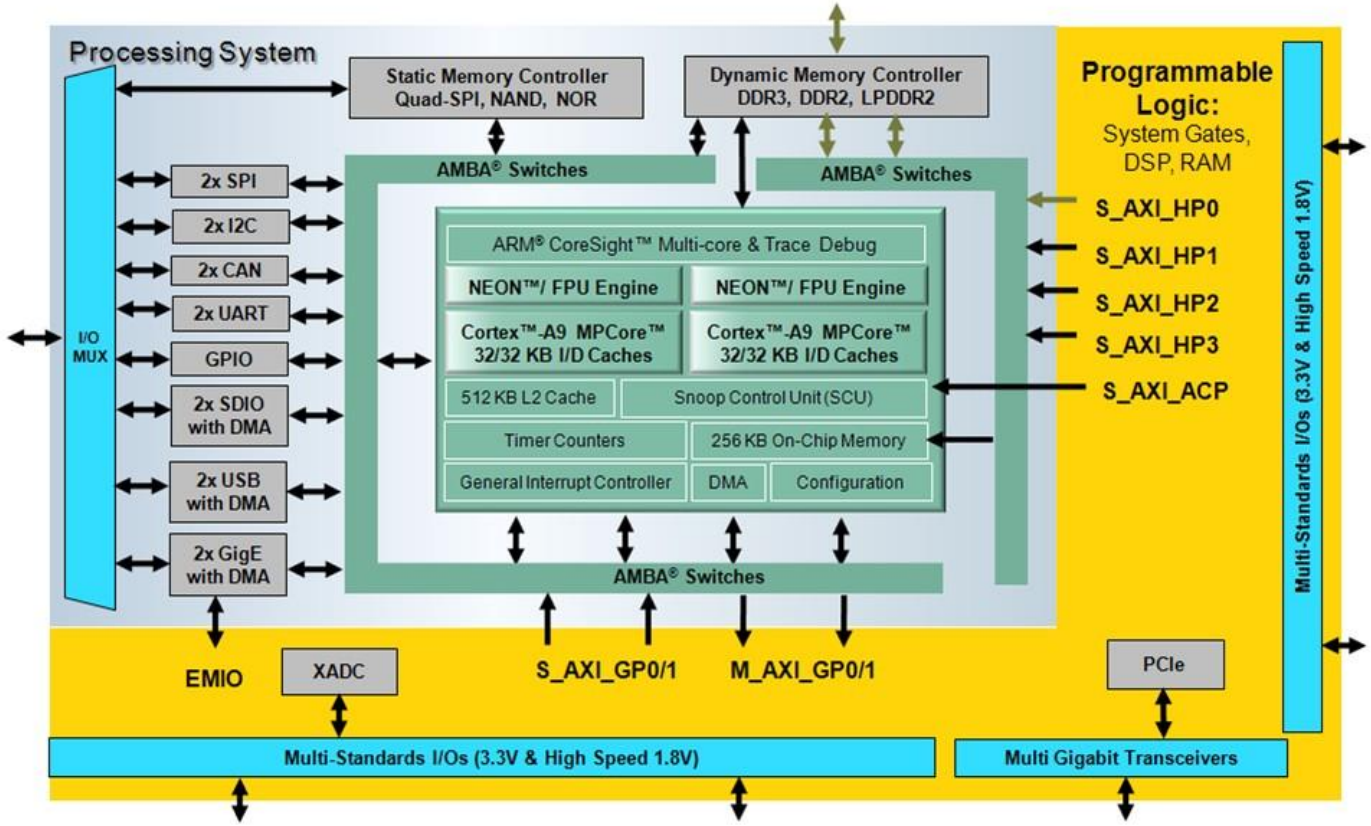
Advanced eXtensible Interface (AXI) Bus

- **AXI Streaming**
  1. Data Only
  2. Minimum control
  3. Unlimited burst length
  4. Only a single direction



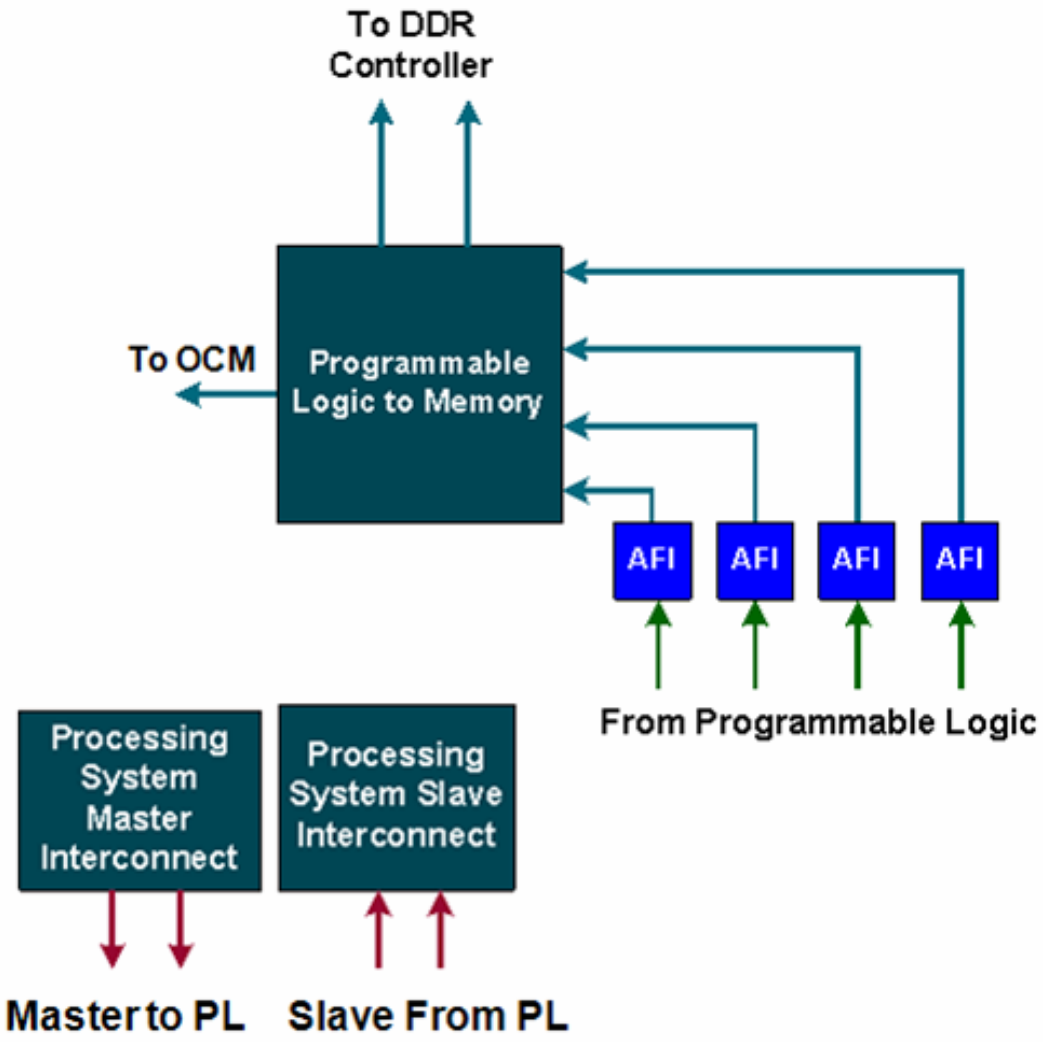
PS-PL Interfaces

- AXI high-performance slave ports (HP0-HP3)
  - Configurable 32-bit or 64-bit data width
  - Access to OCM and DDR only
  - Conversion to processing system clock domain
  - AXI FIFO Interface (AFI) are FIFOs (1KB) to smooth large data transfers
- AXI general-purpose ports (GP0-GP1)
  - Two masters from PS to PL
  - Two slaves from PL to PS
  - 32-bit data width
  - Conversation and sync to processing system clock domain



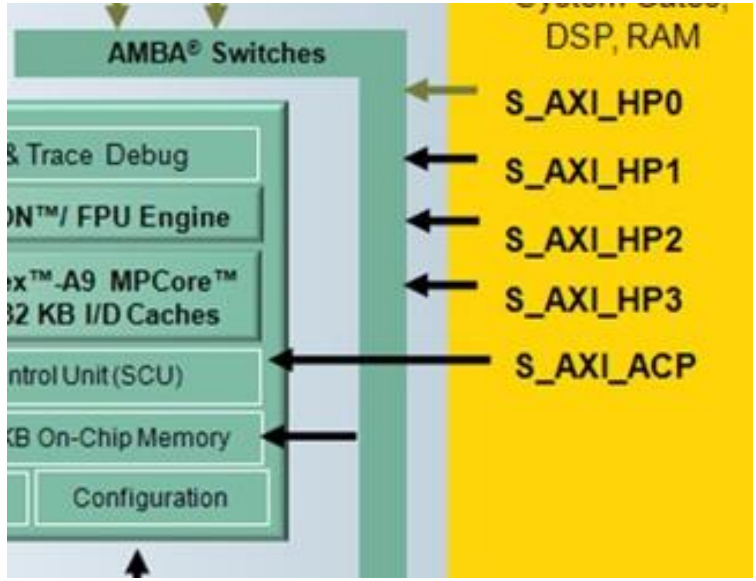
PS-PL Interfaces

- AXI high-performance slave ports (HP0-HP3)
  - Configurable 32-bit or 64-bit data width
  - Access to OCM and DDR only
  - Conversion to processing system clock domain
  - AXI FIFO Interface (AFI) are FIFOs (1KB) to smooth large data transfers
- AXI general-purpose ports (GP0-GP1)
  - Two masters from PS to PL
  - Two slaves from PL to PS
  - 32-bit data width
  - Conversation and sync to processing system clock domain



PS-PL Interfaces

- One 64-bit accelerator coherence port (ACP) AXI slave interface to CPU memory
- DMA, interrupts, events signals
  - Processor event bus for signaling event information to the CPU
  - PL peripheral IP interrupts to the PS general interrupt controller (GIC)
  - Four DMA channel RDY/ACK signals
- Extended multiplexed I/O (EMIO) allows PS peripheral ports access to PL logic and device I/O pins
- Clock and resets
  - Four PS clock outputs to the PL with enable control
  - Four PS reset outputs to the PL
- Configuration and miscellaneous



PS-PL Interfaces

- One 64-bit accelerator coherence port (ACP) AXI slave interface to CPU memory
- DMA, interrupts, events signals
  - Processor event bus for signaling event information to the CPU
  - PL peripheral IP interrupts to the PS general interrupt controller (GIC)
  - Four DMA channel RDY/ACK signals
- Extended multiplexed I/O (EMIO) allows PS peripheral ports access to PL logic and device I/O pins
- Clock and resets
  - Four PS clock outputs to the PL with enable control
  - Four PS reset outputs to the PL
- Configuration and miscellaneous

Type	PL Signal Name	I/O	Destination
PL to PS Interrupts	IRQF2P[7:0]	I	SPI: Numbers [68:61].
	IRQF2P[15:8]	I	SPI: Numbers [91:84].
	IRQF2P[19:16]	I	PPI: nFIQ, nIRQ (both CPUs).
PS to PL Interrupts	IRQP2F[27:0]	O	PI Logic. These signals are received from the I/O peripherals and are forwarded to the interrupt controller. These signals are also provided as outputs to the PL.

PS-PL Interfaces

- One 64-bit accelerator coherence port (ACP) AXI slave interface to CPU memory
- DMA, interrupts, events signals
  - Processor event bus for signaling event information to the CPU
  - PL peripheral IP interrupts to the PS general interrupt controller (GIC)
  - Four DMA channel RDY/ACK signals
- Extended multiplexed I/O (EMIO) allows PS peripheral ports access to PL logic and device I/O pins
- Clock and resets
  - Four PS clock outputs to the PL with enable control
  - Four PS reset outputs to the PL
- Configuration and miscellaneous

Type	PL Signal Name	I/O	Reference
PL Clocks	FCLKCLK[3:0]	O	Chapter 25, Clocks
PL Clock Throttle Control	FCLKCLKTRIG [3:0]	I	
PL Resets	FCLKRESETN [3:0]	O	Chapter 26, Reset System

PS Clocking Sources

- PS clocks
  - PS clock source from external package pin
  - PS has three PLLs for clock generation
  - PS has four clock ports to PL
- PL clocking resources
  - PL has a different clock source domain compared to the PS
  - The clock to PL can be sourced from external clock capable pins
  - Can use one of the four PS clocks as source
- Synchronizing the clock between PL and PS is taken care by the architecture of the PS
- PL cannot supply clock source to PS

Resets

- Internal resets
  - Power-on reset (POR)
  - Watchdog resets from the three watchdog timers
- PS resets
  - External reset: PS_SRST_B
  - Warm reset: SRSTB
- PL resets
  - Four reset outputs from PS to PL
  - FCLK_RESET[3:0]

Reset Type	POR	Non-POR
Sample Pin Straps	Yes	No
Initialize PS PLLs(1)	Yes, by the hardware	Yes, by the BootROM
PS RAM (FIFOs, buffers, etc.)	Cleared	Cleared
IOP clocks	Disabled	Disabled
BootROM executes	Yes	Yes
Retains previous boot mode	No	Yes
Reboot Status register	Resets it	Accumulates the system reset type.
Device Registers	All	Persistent registers.(2)
Resets PL	Yes	Yes

Zynq Boot and Configuration

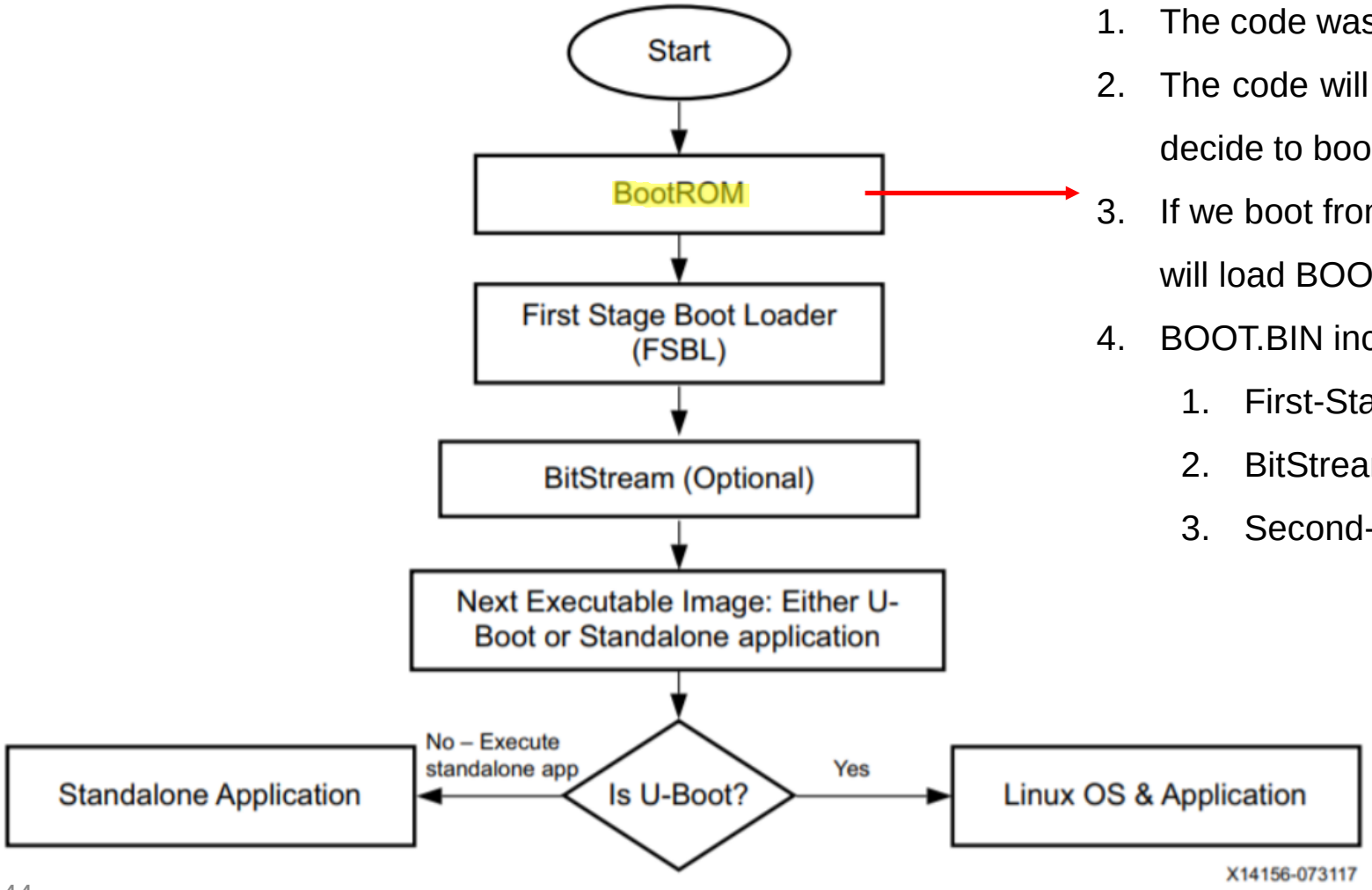
- Zynq devices can be booted and/or configured in
  - Secure mode via static memories only (JTAG excluded)
    - Ability to have secure software
    - Protects bitstream and IP
  - Non-secure mode via JTAG or static memories (debug and development environment)
    - Standard boot model
- Four master boot devices
  - QSPI: serial memory, linear addressing
  - NAND: complex parallel memory
  - NOR: parallel memory, linear addressing
  - SD: Flash memory card
- Secondary boot devices
  - USB, Ethernet, and most other peripherals

Zynq Boot and Configuration

- Multi-stage boot process
  - **Stage 0: Runs from ROM; loads from non-volatile memory to OCM**
    - Provided by Xilinx; unmodifiable
  - **Stage 1: Runs from OCM; loads from non-volatile memory to DDRx memory**
    - User developed; Xilinx offers example code through SDK project
    - Initiates PS boot and PL configuration
  - **Stage 2: Optional; runs from DDR**
    - User developed; Xilinx offers example code – U-boot
    - Sourced from flash memory or through common peripherals, programmable logic I/O, etc.
  - Programmable logic configuration can be performed in Stage 1 or 2

Zynq Boot and Configuration

- Multi-stage boot process



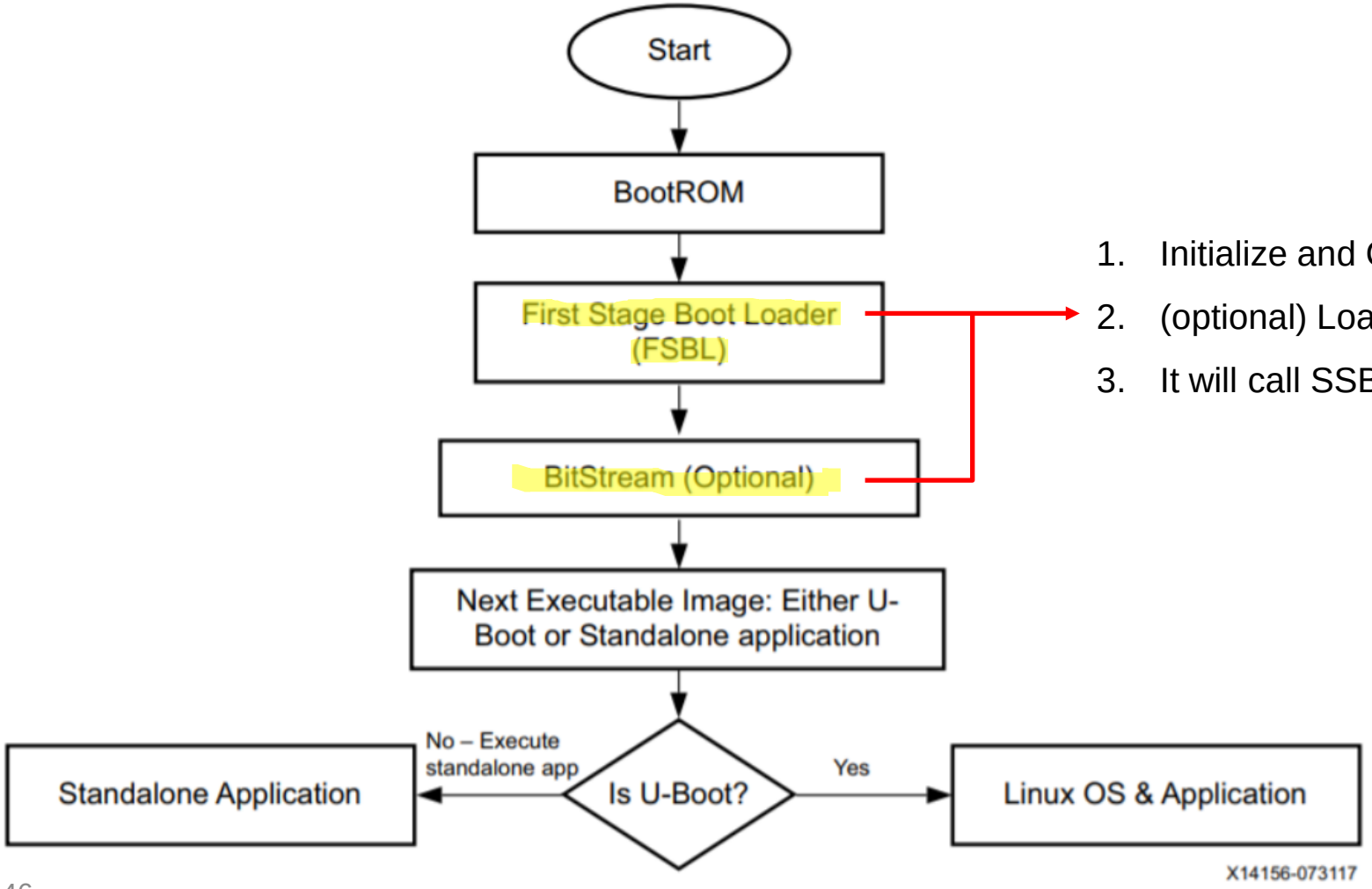
1. The code was pre-built by Xilinx will be executed in BootRom
2. The code will check the state of current BootMode Pins, then decide to boot from where
3. If we boot from SD Card, then the Boot Loader from BootRom will load BOOT.BIN which in SD Card to OCM
4. BOOT.BIN includes:
    1. First-Stage Boot Loader
    2. BitStream File (PL Logic Configuration)
    3. Second-Stage Boot Loader

Zynq Boot and Configuration

- Multi-stage boot process
  - **Stage 0: Runs from ROM; loads from non-volatile memory to OCM**
    - Provided by Xilinx; unmodifiable
  - **Stage 1: Runs from OCM; loads from non-volatile memory to DDR memory**
    - User developed; Xilinx offers example code through SDK project
    - Initiates PS boot and PL configuration
  - **Stage 2: Optional; runs from DDR**
    - User developed; Xilinx offers example code – U-boot
    - Sourced from flash memory or through common peripherals, programmable logic I/O, etc.
  - Programmable logic configuration can be performed in Stage 1 or 2

Zynq Boot and Configuration

- Multi-stage boot process



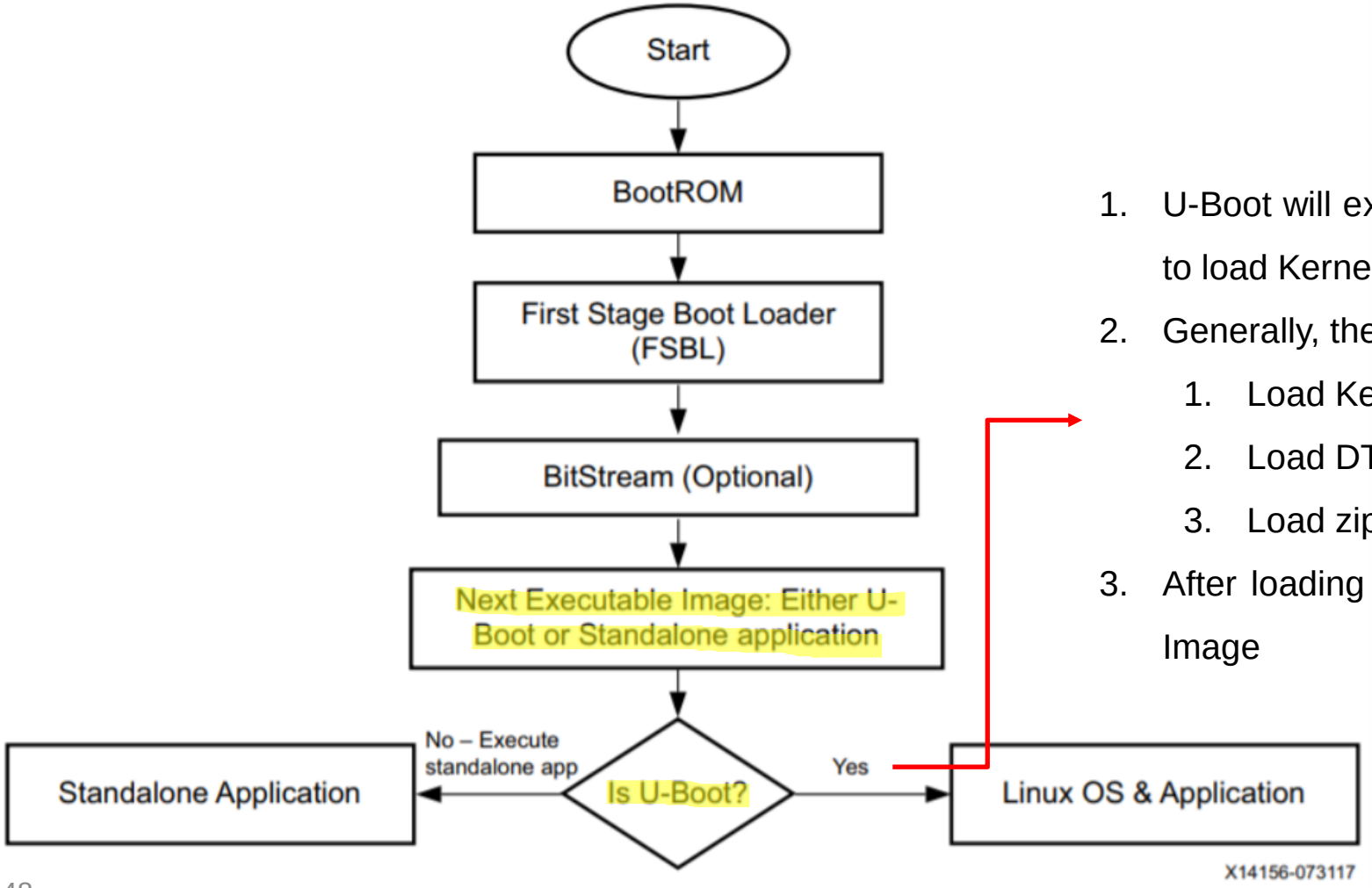
1. Initialize and Configure PS
2. (optional) Load Bitstream file and Configure PL
3. It will call SSBL finally when the above things have done

Zynq Boot and Configuration

- Multi-stage boot process
  - **Stage 0: Runs from ROM; loads from non-volatile memory to OCM**
    - Provided by Xilinx; unmodifiable
  - **Stage 1: Runs from OCM; loads from non-volatile memory to DDR memory**
    - User developed; Xilinx offers example code through SDK project
    - Initiates PS boot and PL configuration
  - **Stage 2: Optional; runs from DDR**
    - User developed; Xilinx offers example code – U-boot
    - Sourced from flash memory or through common peripherals, programmable logic I/O, etc.
  - Programmable logic configuration can be performed in Stage 1 or 2

Zynq Boot and Configuration

- Multi-stage boot process

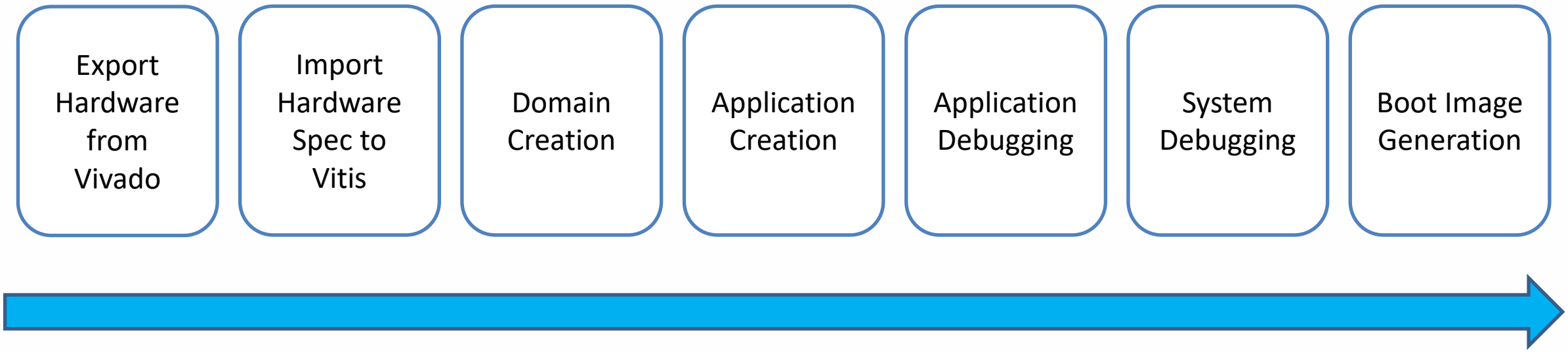


1. U-Boot will execute autoboot code and check BootMode Pins to load Kernel Image from where
2. Generally, the loading step includes:
    1. Load Kernel Image from SD Card and copy it to DDR
    2. Load DTB File (devicetree.dtb)
    3. Load zipped ramdisk file system
3. After loading completing, U-Boot will start to execute Kernel Image

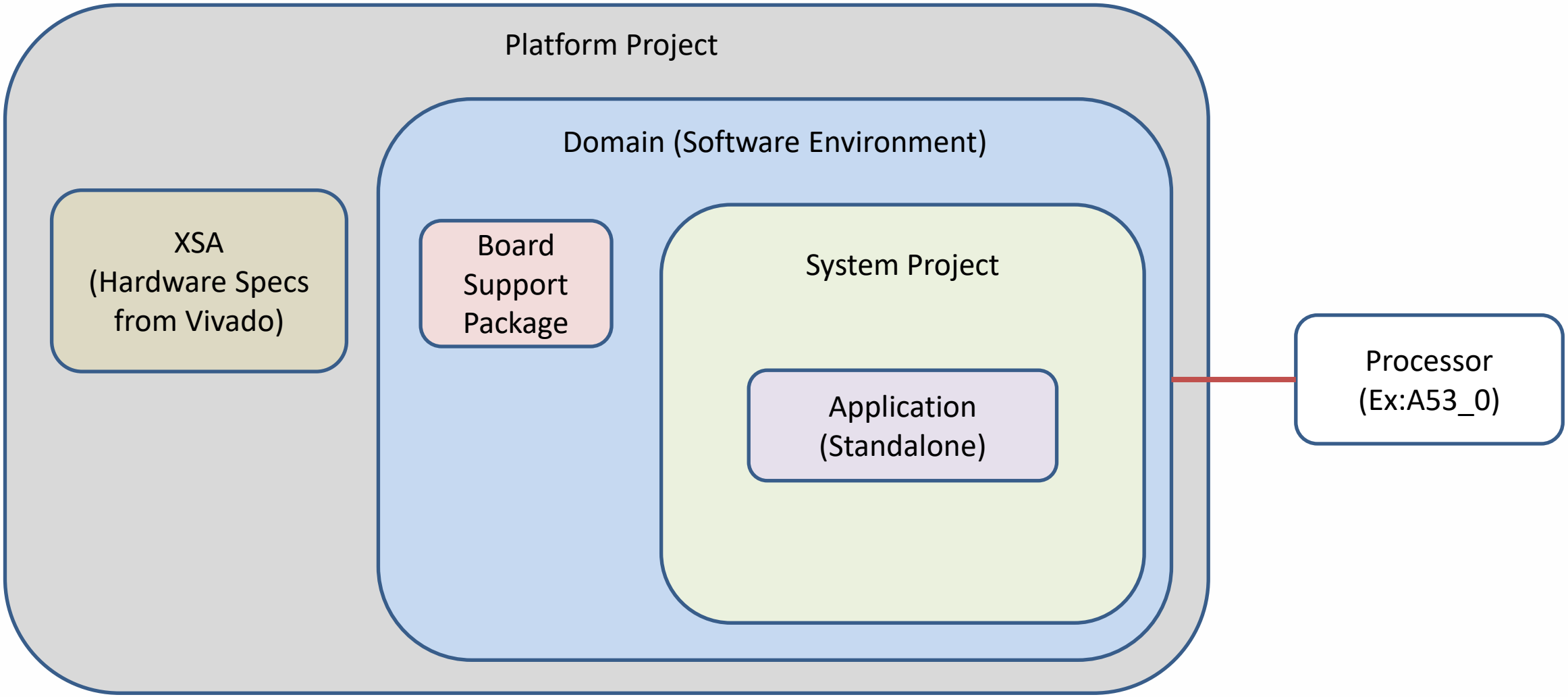
Vitis Development Workflow

XILINX.

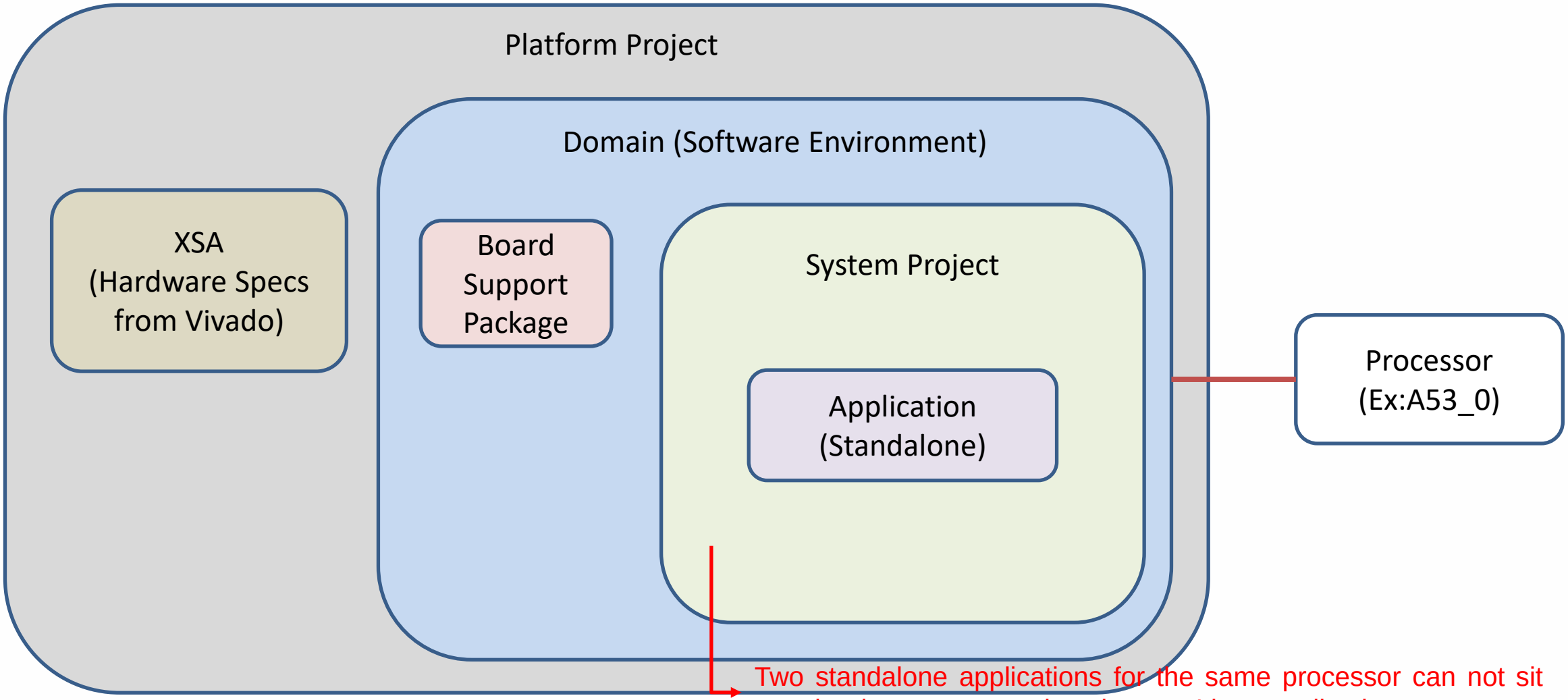
Vitis Software Development Workflow



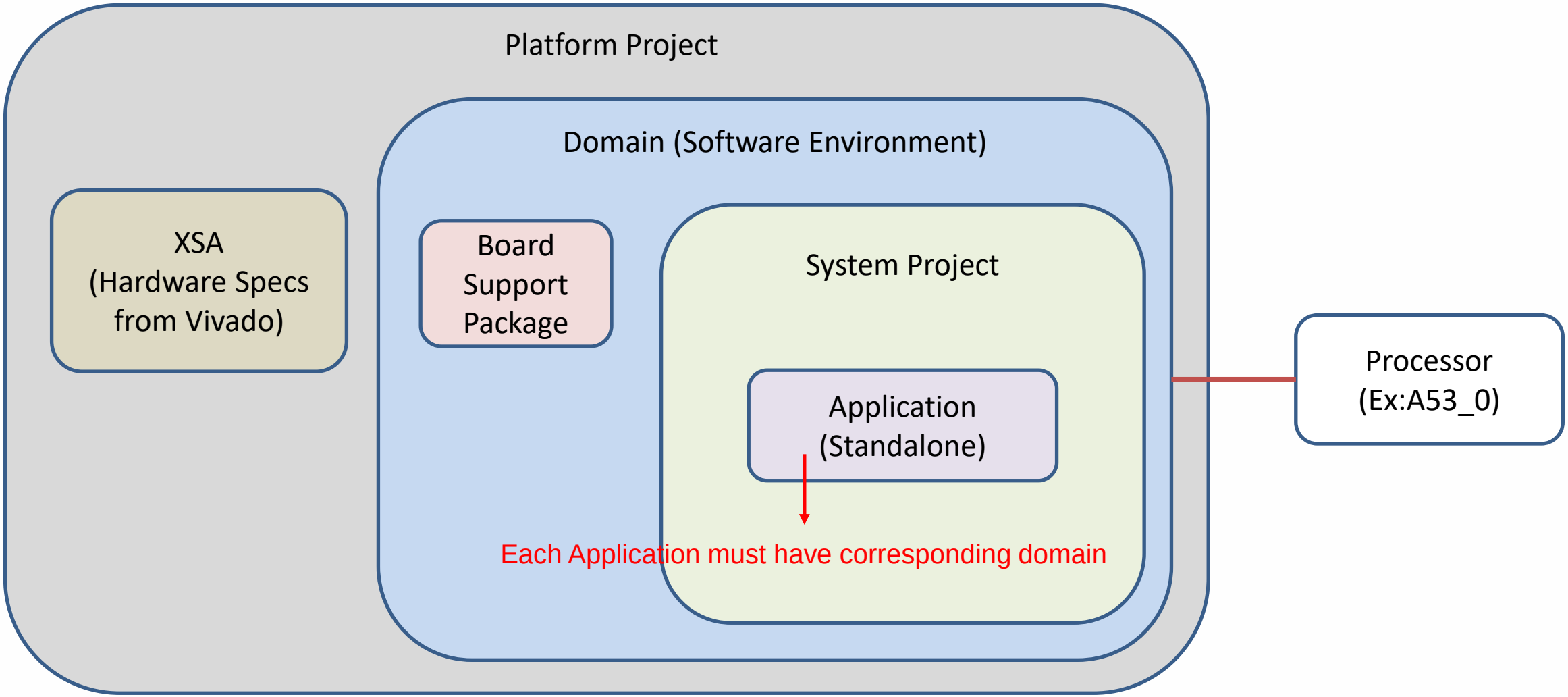
Vitis Workspace Structure



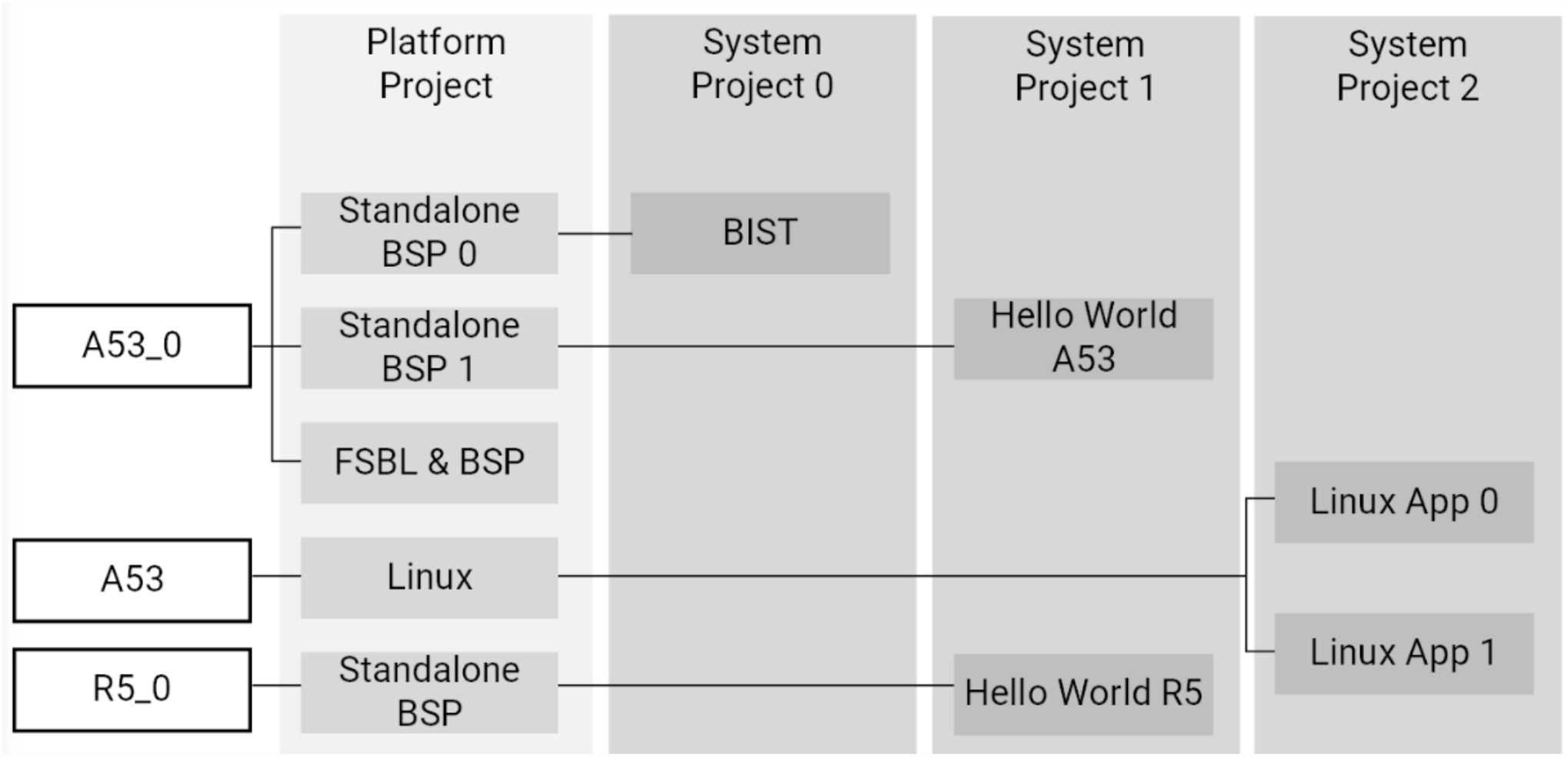
Vitis Workspace Structure



Vitis Workspace Structure



Vitis Workspace Structure



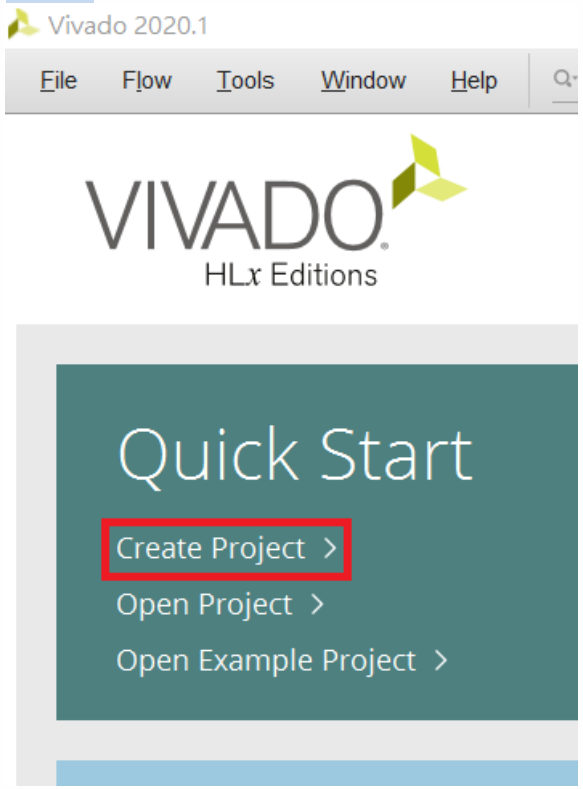
Preparatory Work from Vivado (XSA Creation)

XILINX.

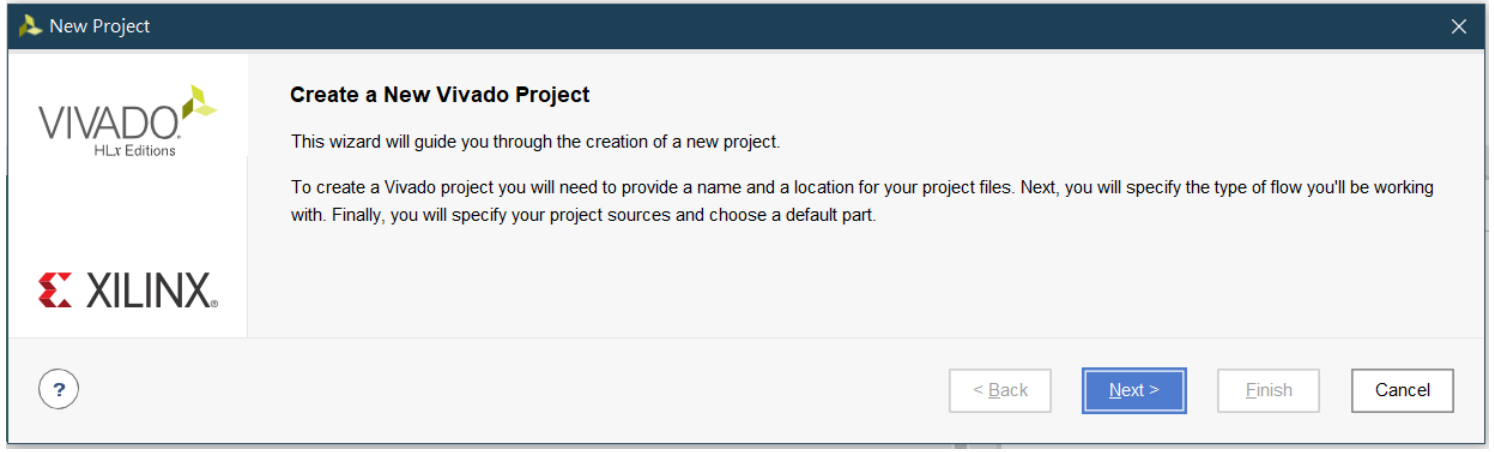
Create a Vivado Project

- This course is meant to build a baremetal application that controls LED effect on a zynq device.
- Start with creating a vivado project.

1



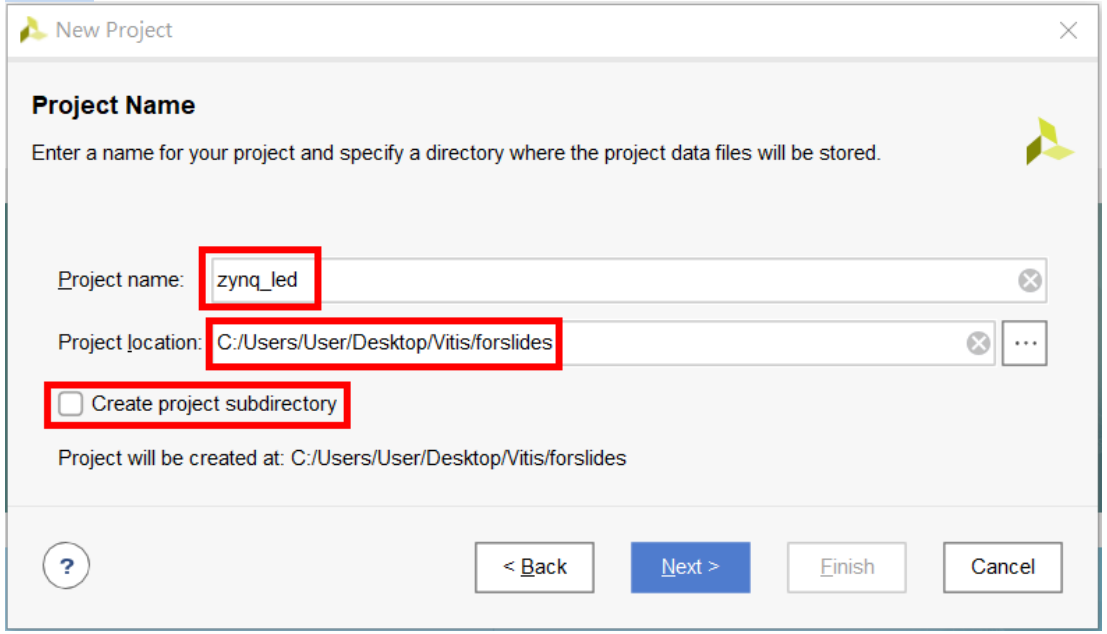
2



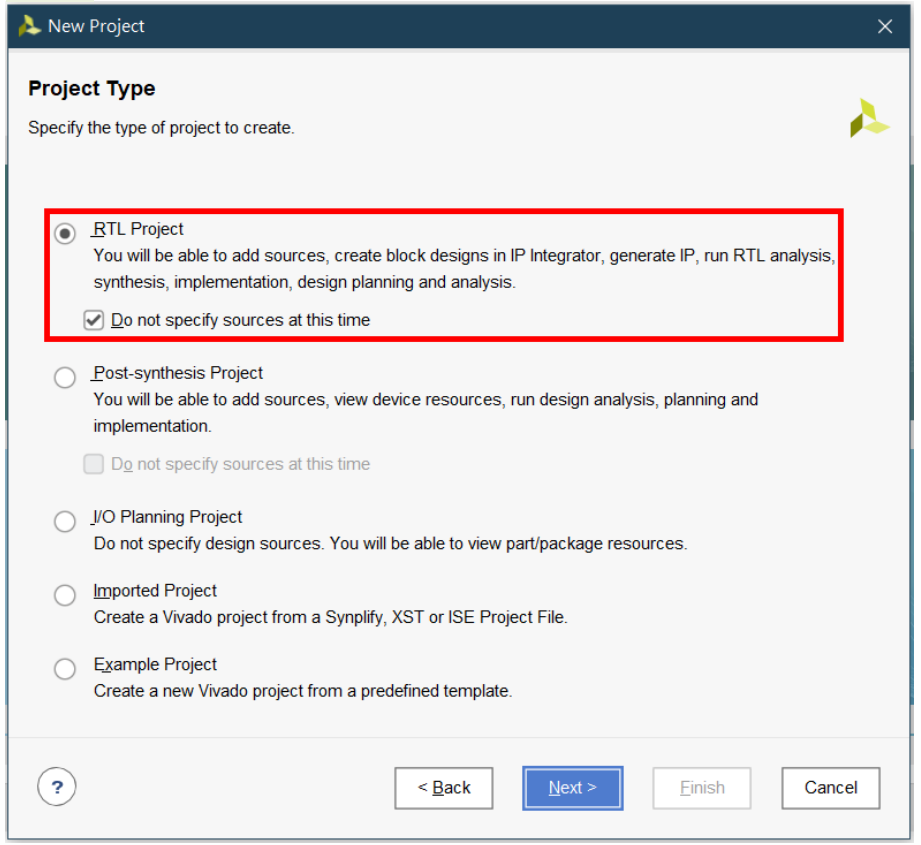
XILINX

Create a Vivado Project

1



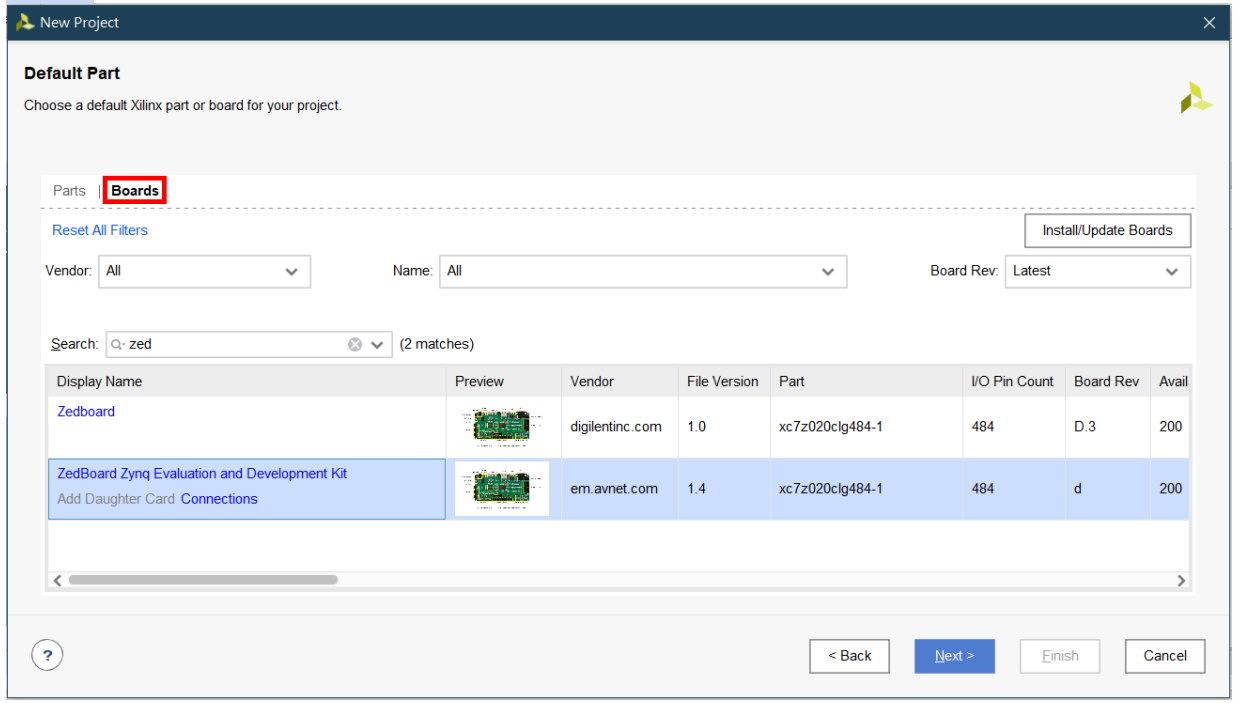
2



Create a Vivado Project

- In this step, it is vital to select the corresponding device. Vivado generates xsa file based on the device selected here.

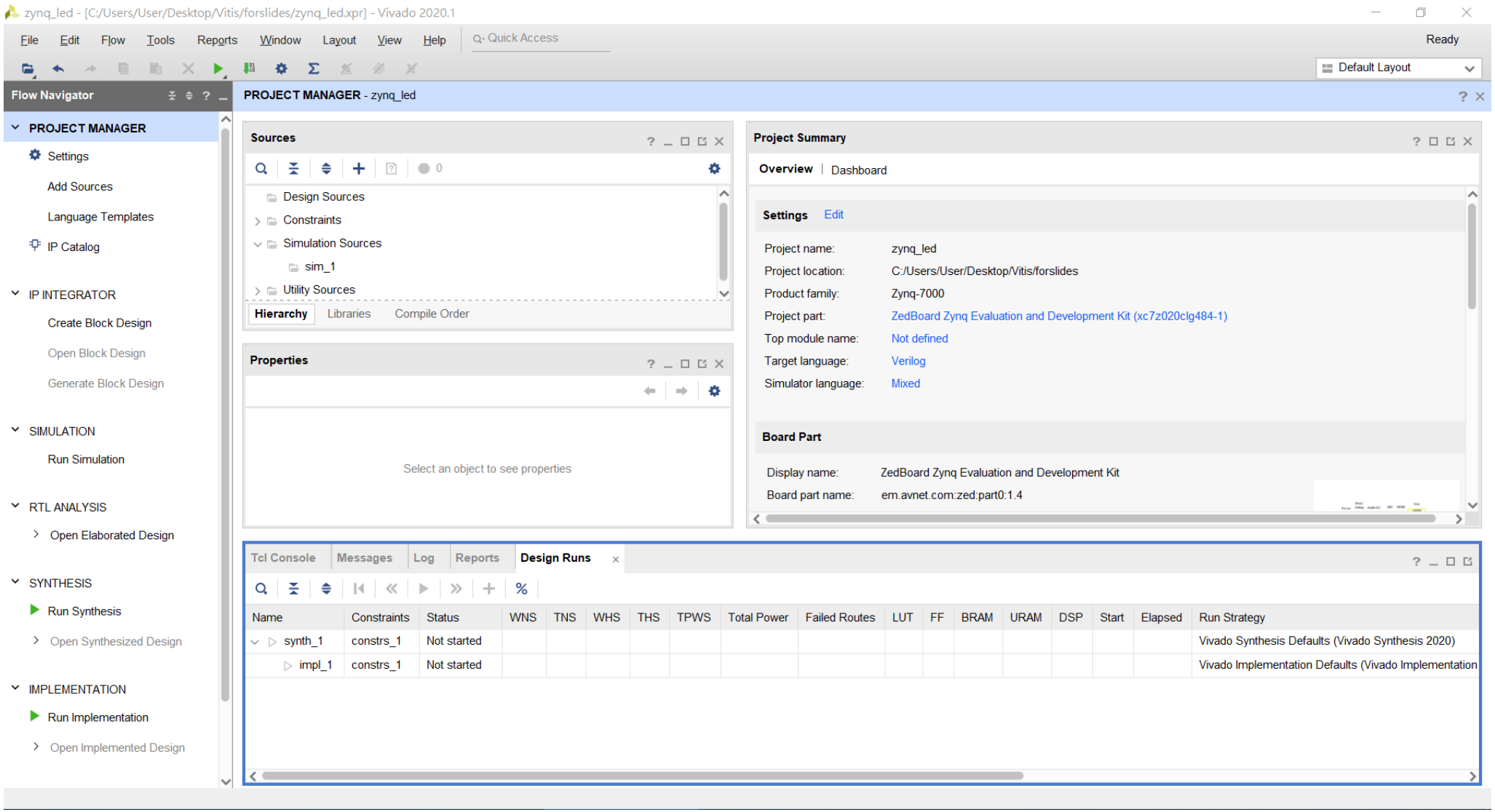
1



2

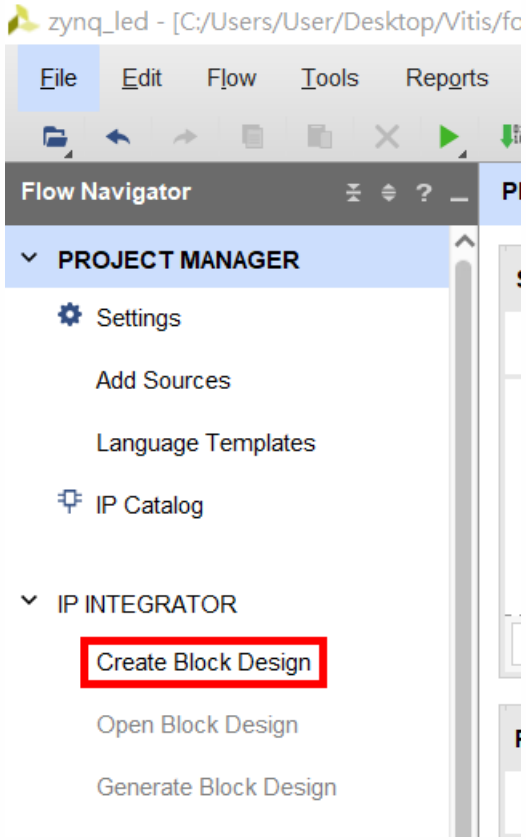
XILINX

Vivado

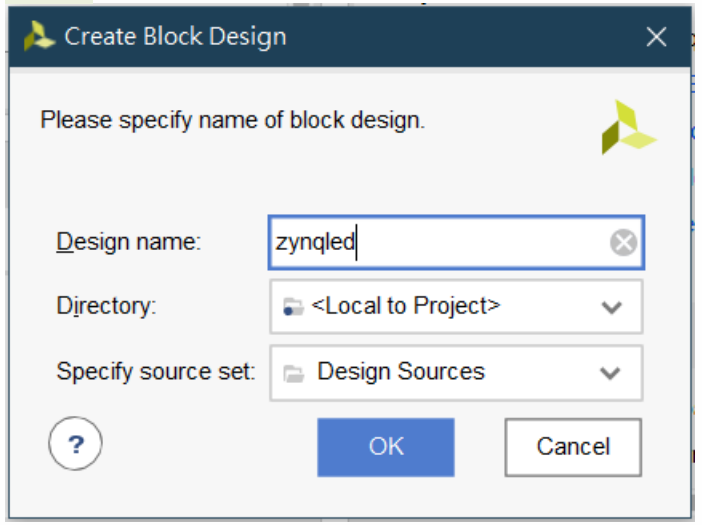


Create a block design

1

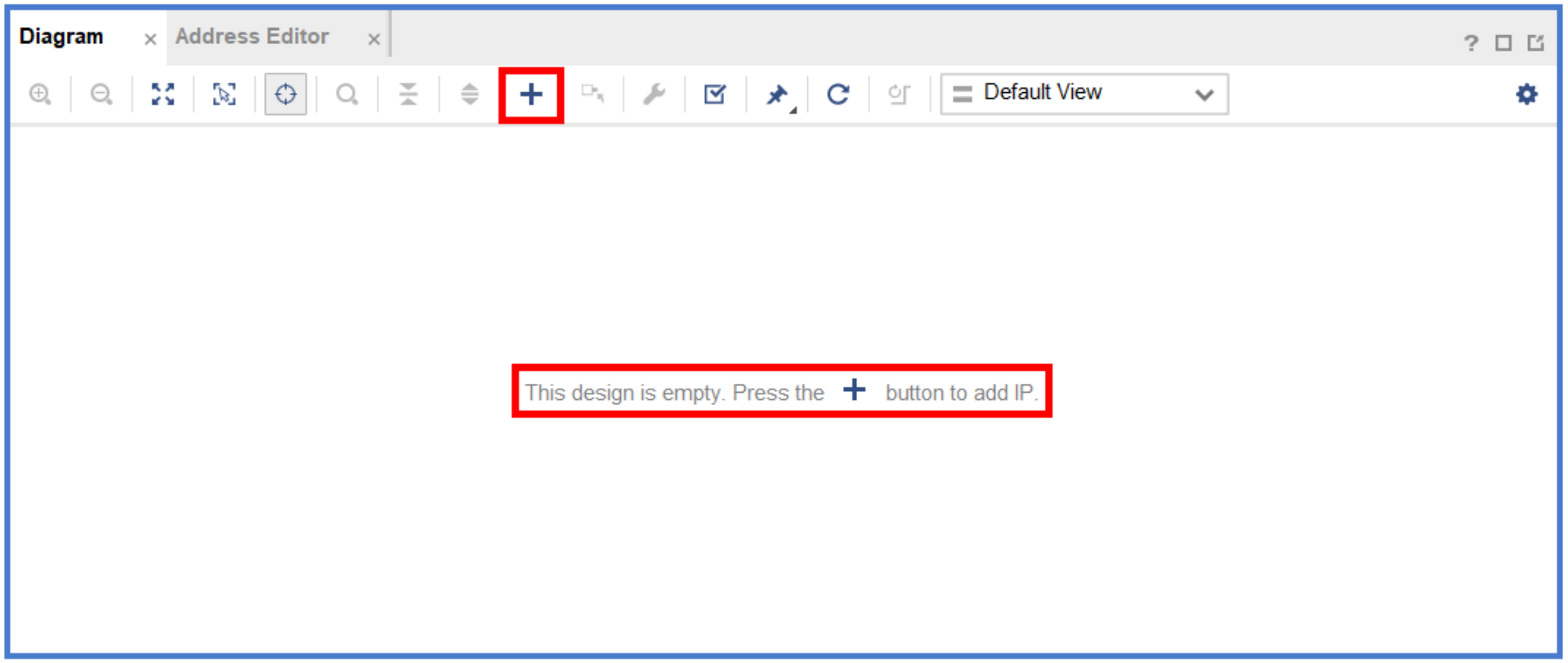


2



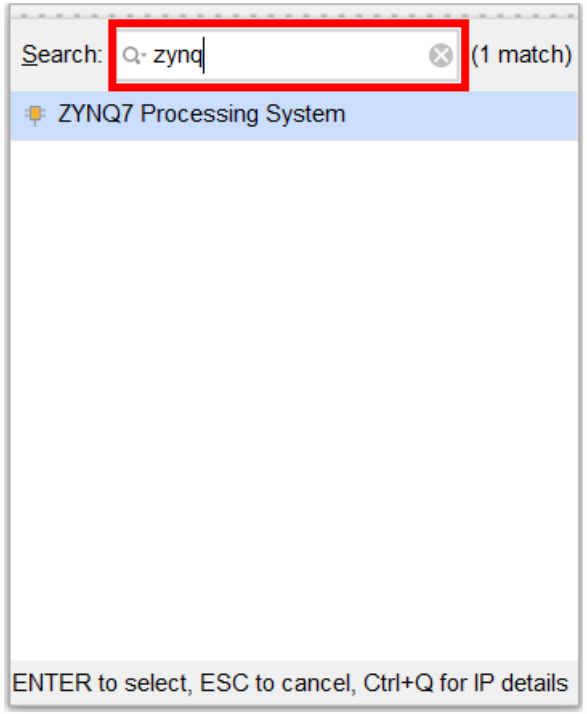
XILINX.

Add IP

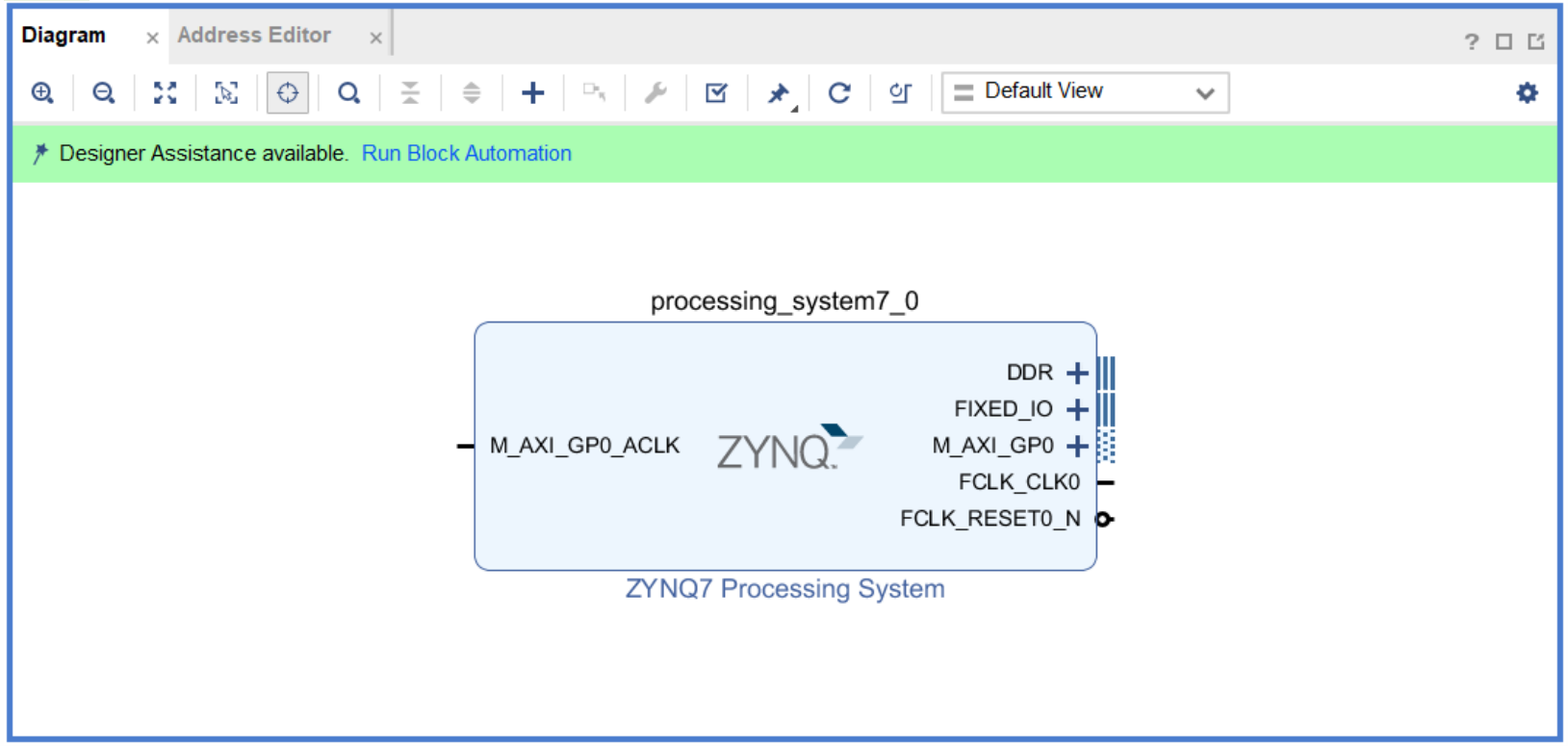


Add Zynq-7000 IP

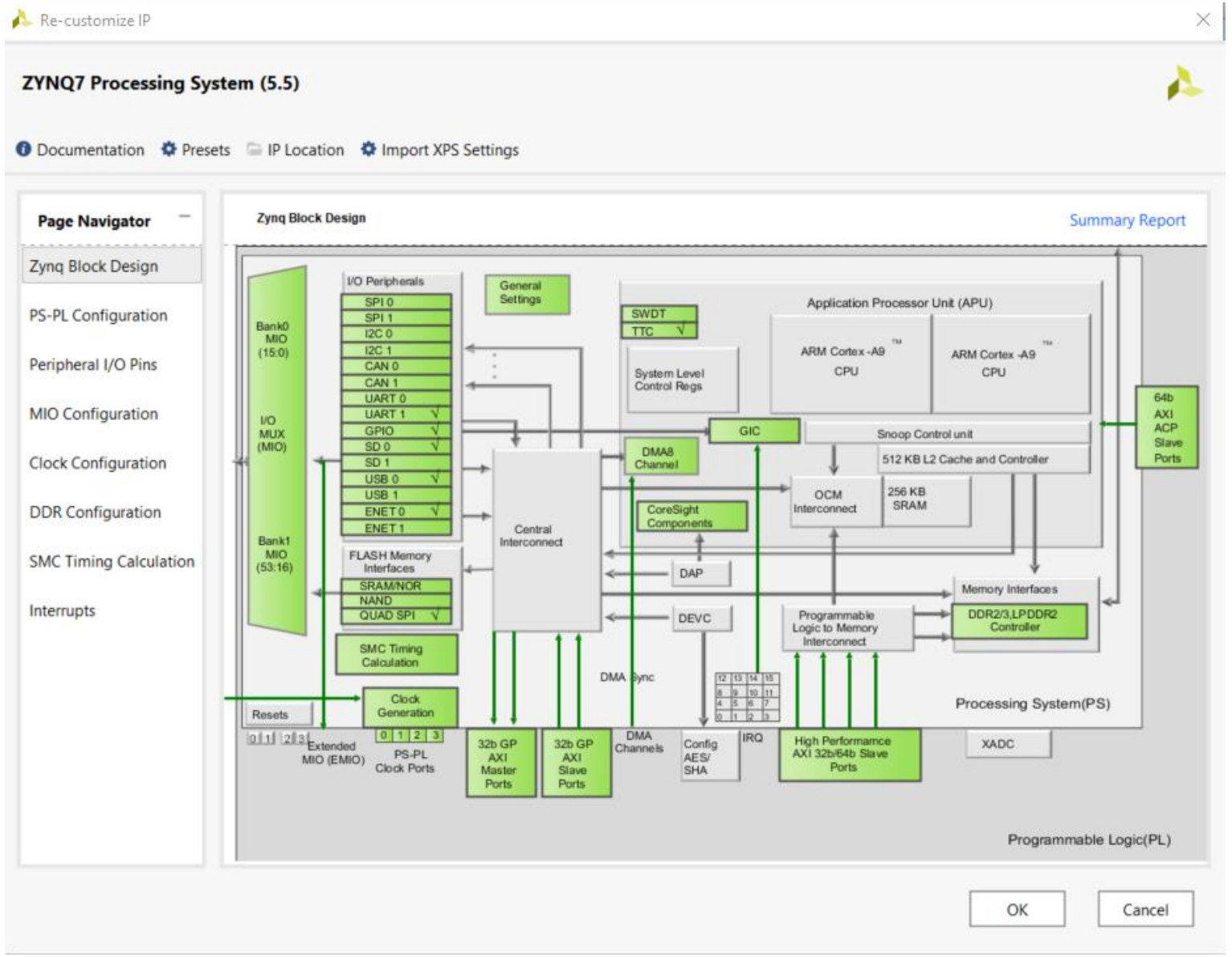
1



2

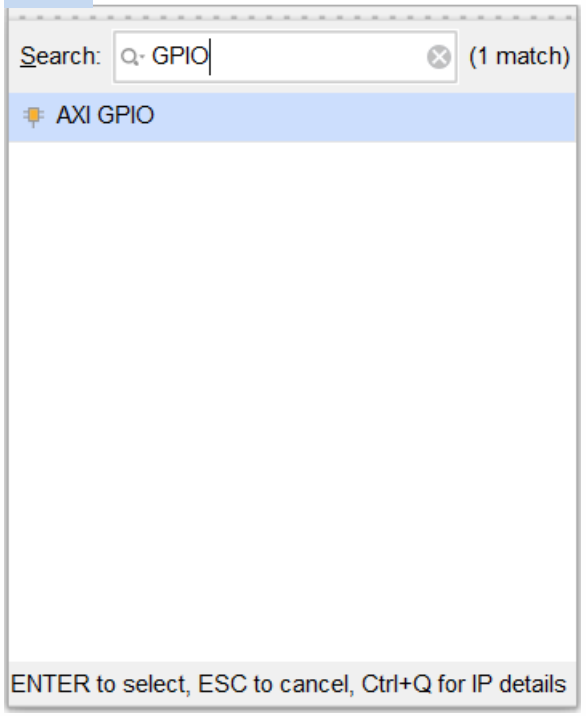


Add Zynq-7000 IP

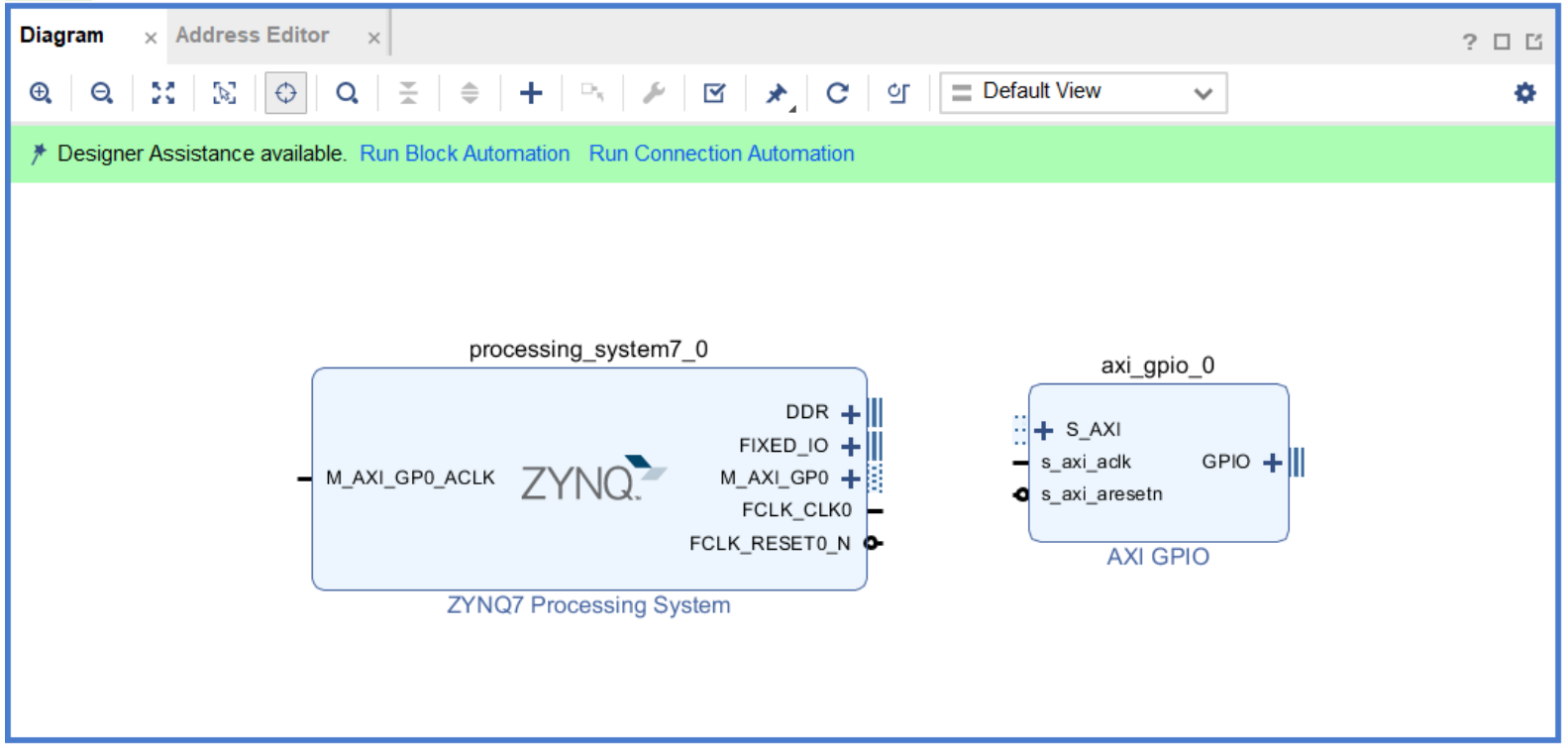


Add AXI GPIO IP

1

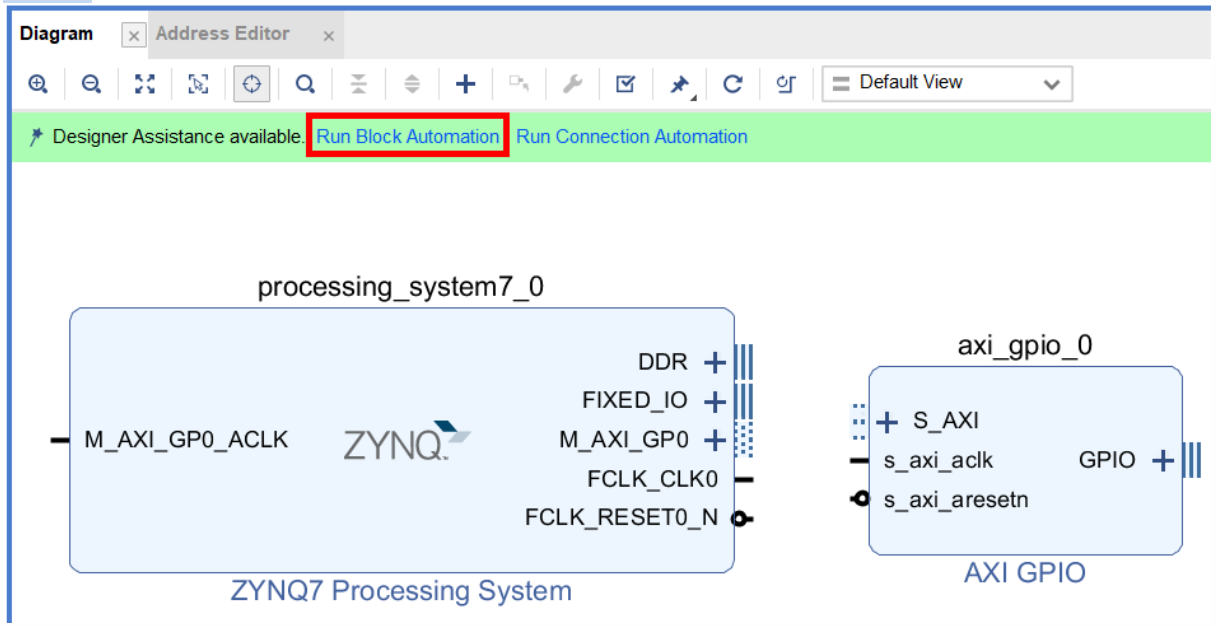


2

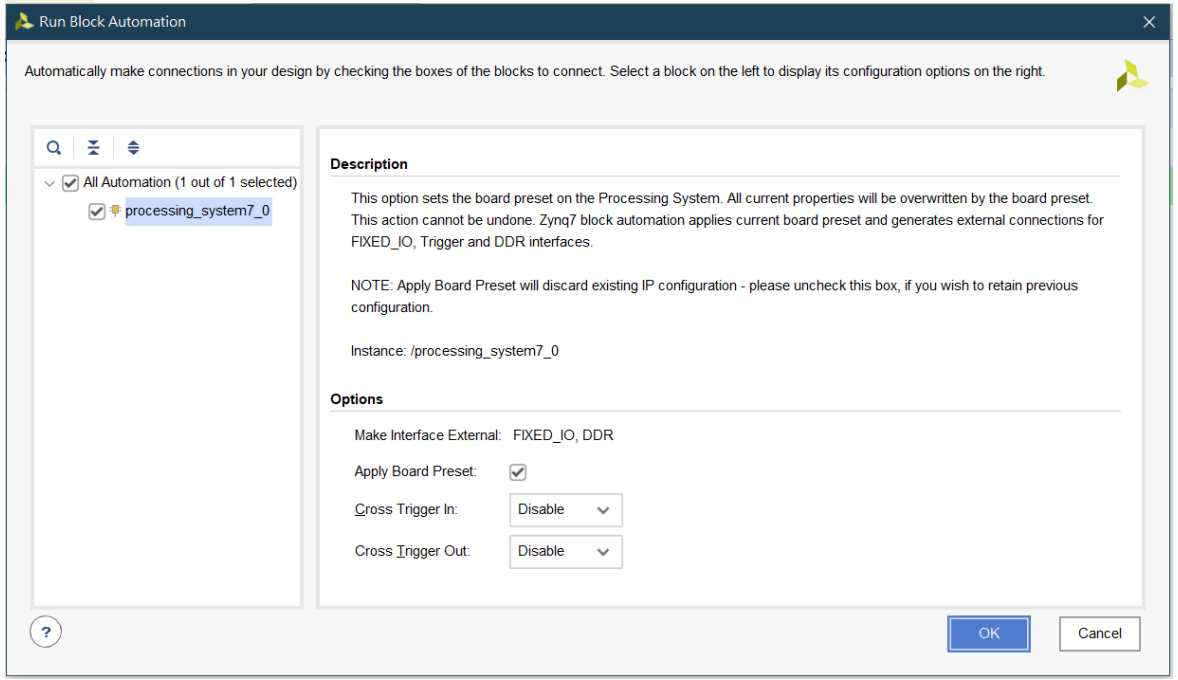


Block Automation

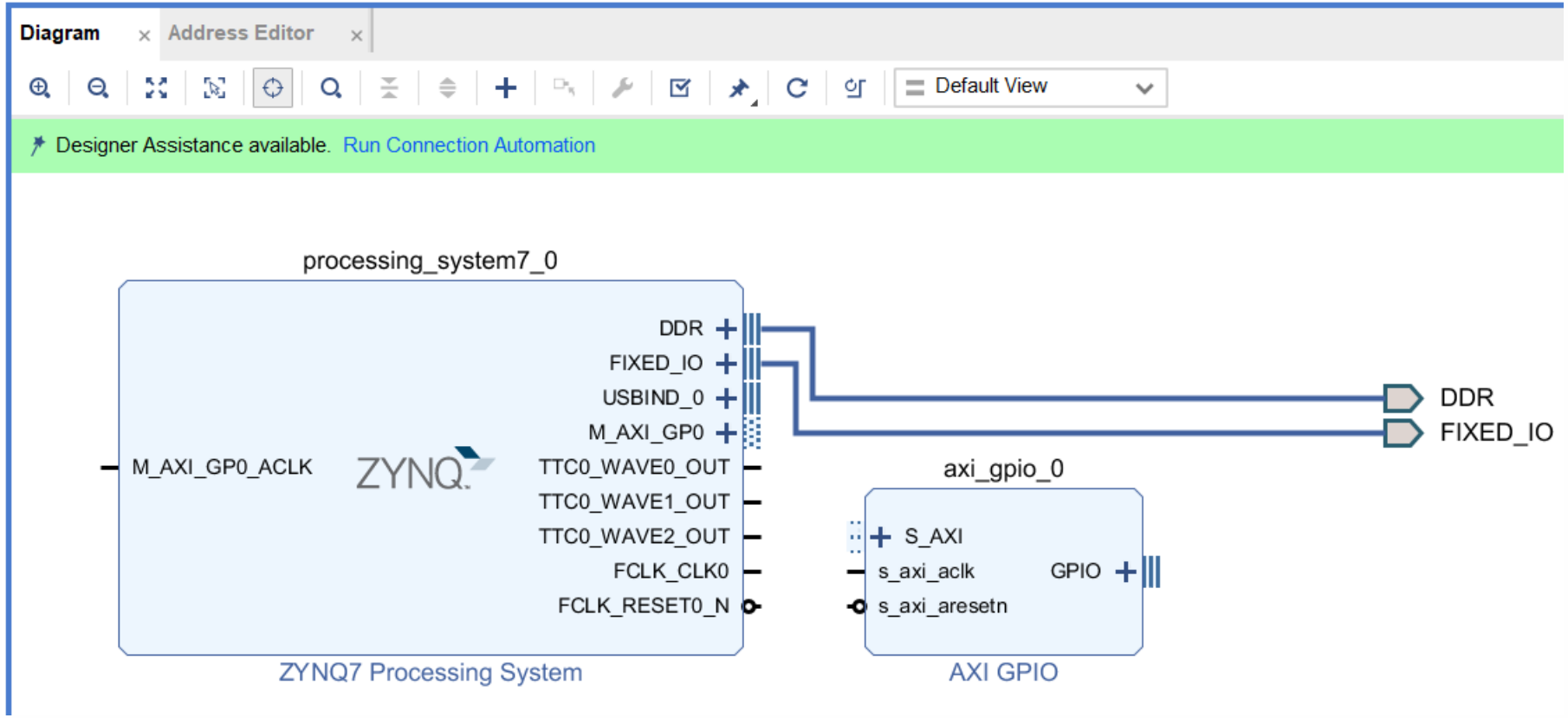
1



2

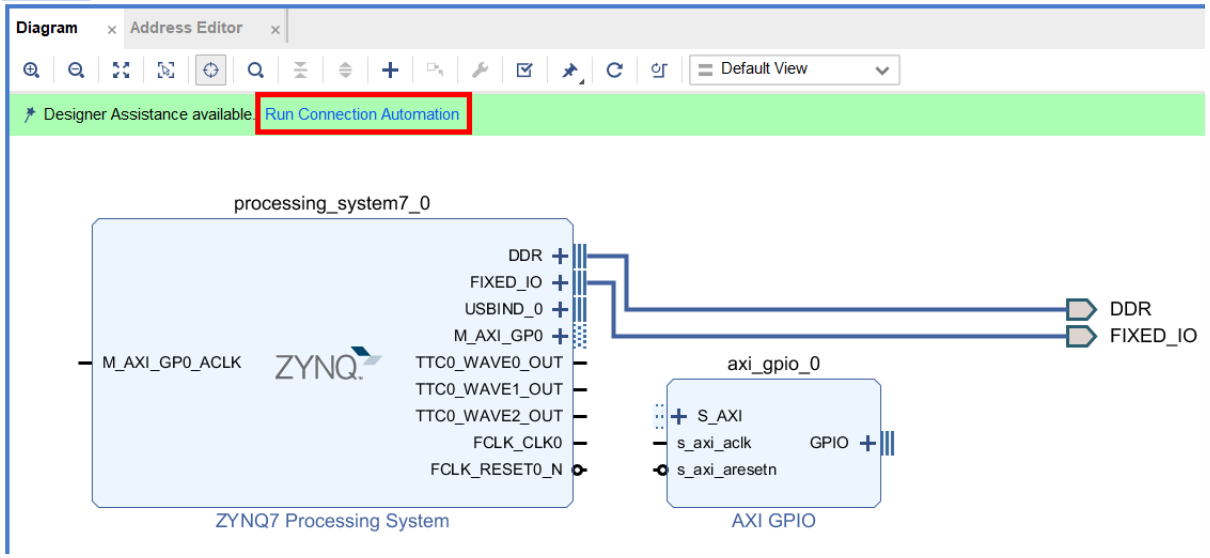


Block Automation

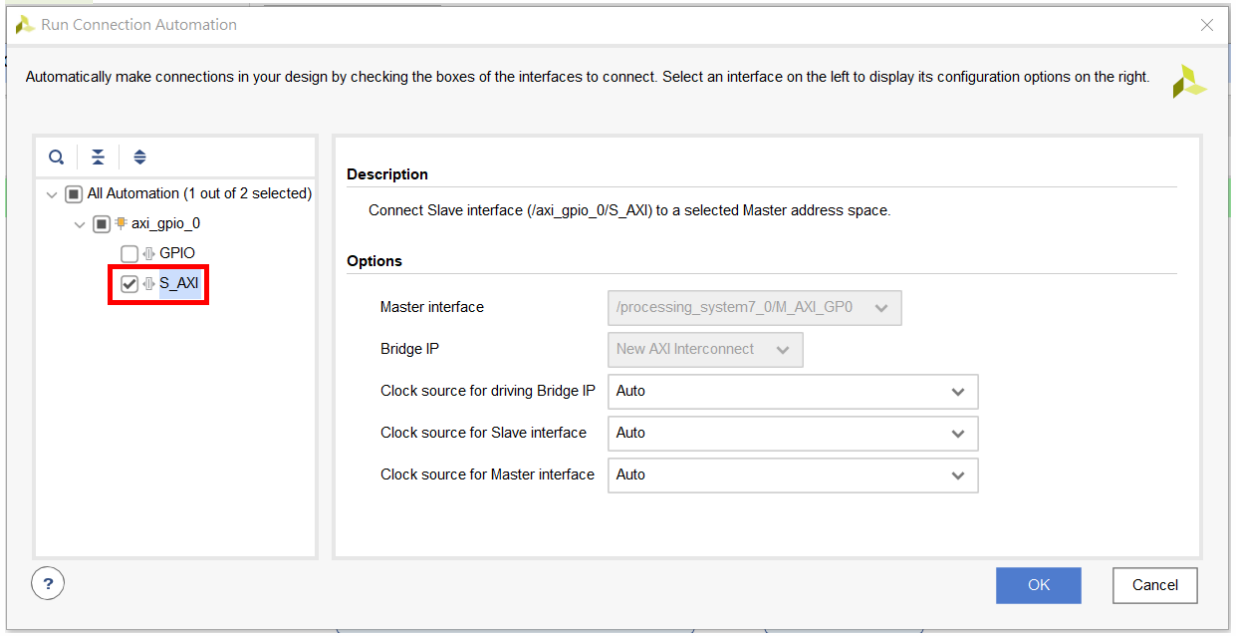


Connection automation

1



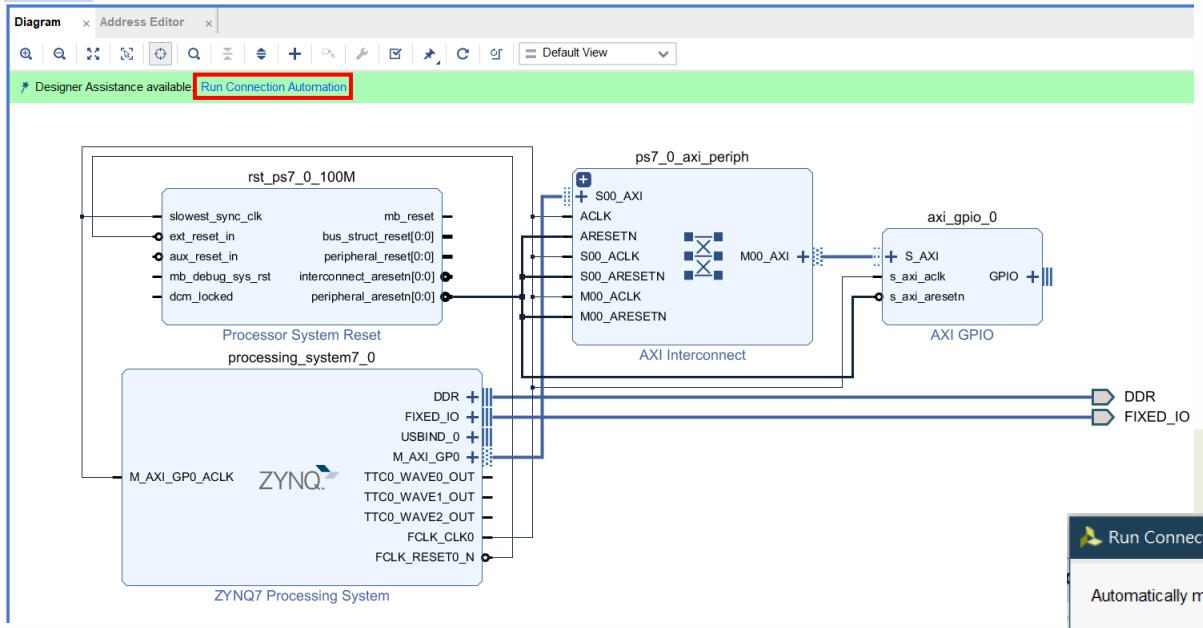
2



XILINX.

Connect GPIO to on-board LEDs

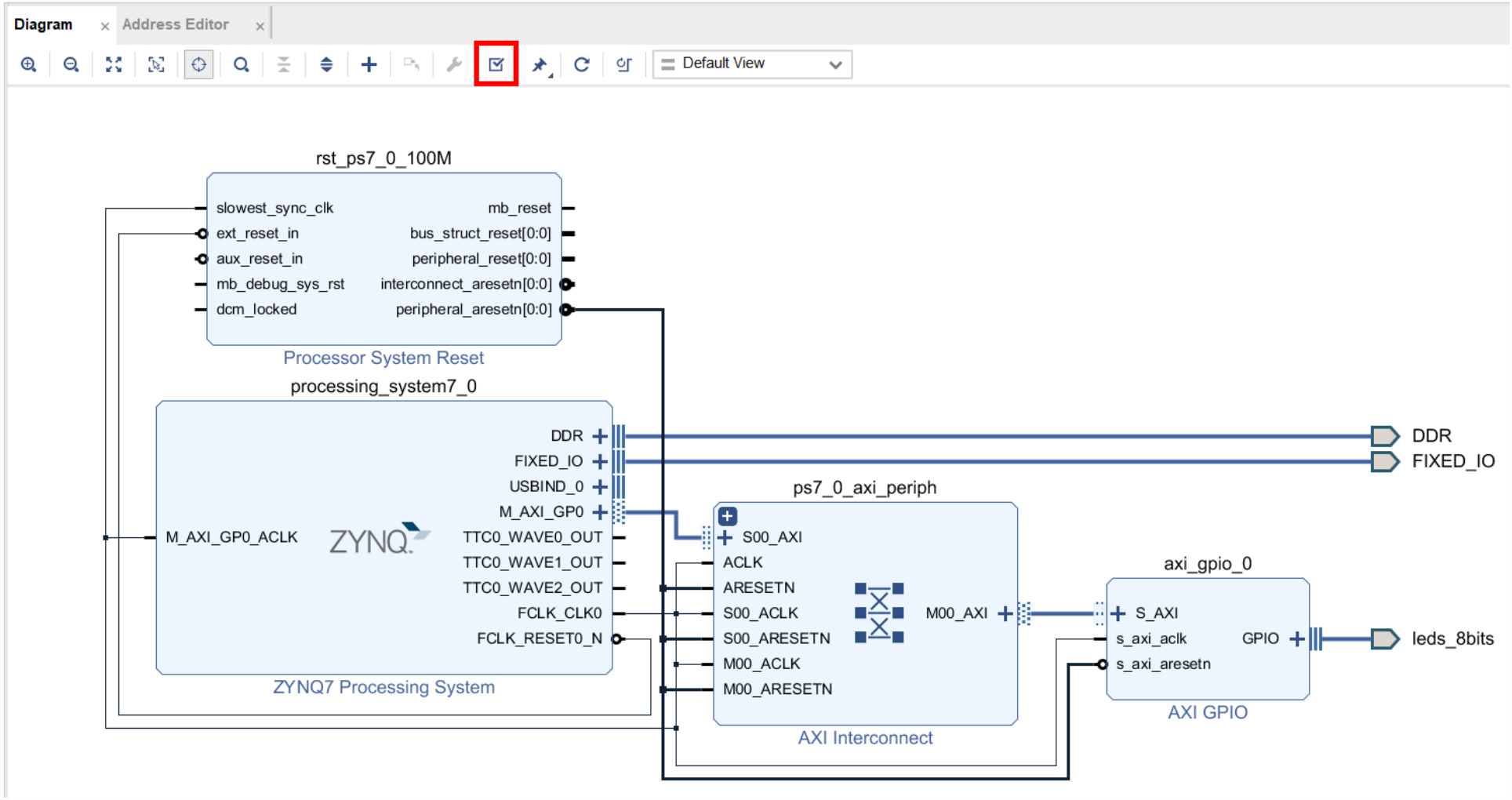
1



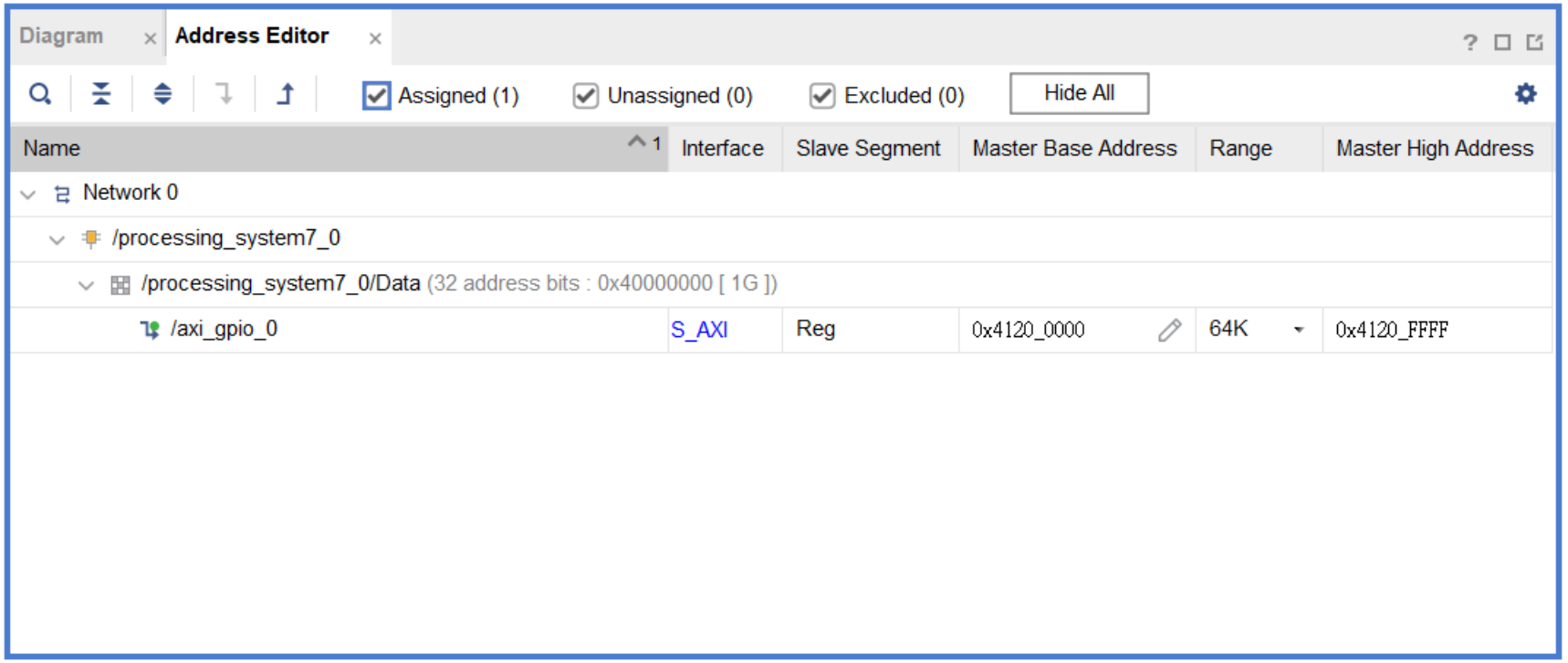
2

XILINX.

Block design validation

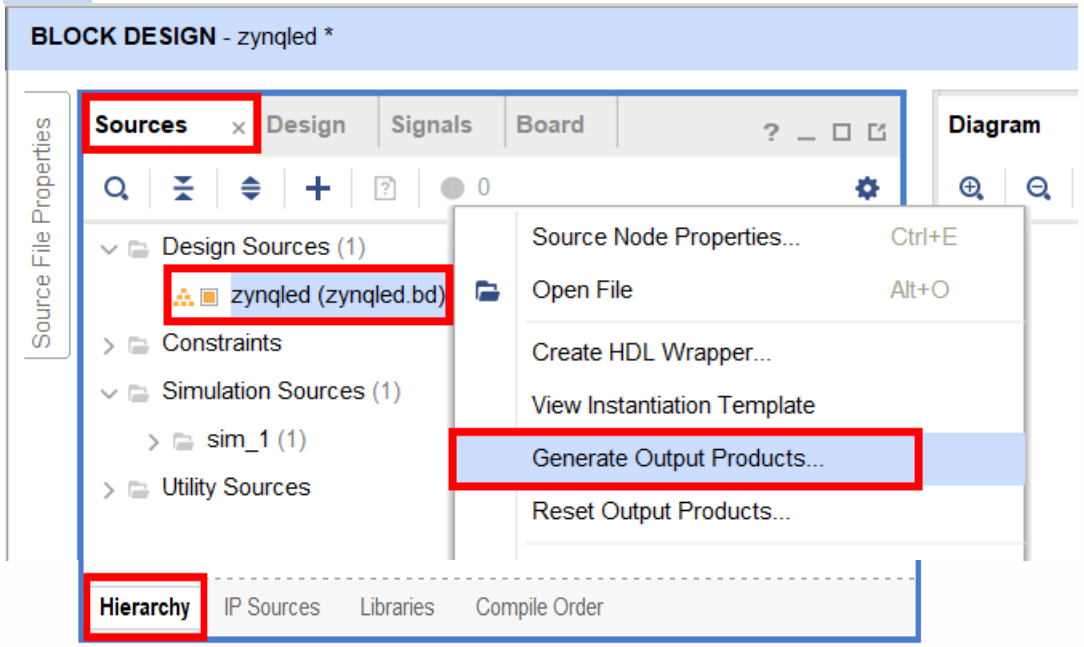


Block design address editor

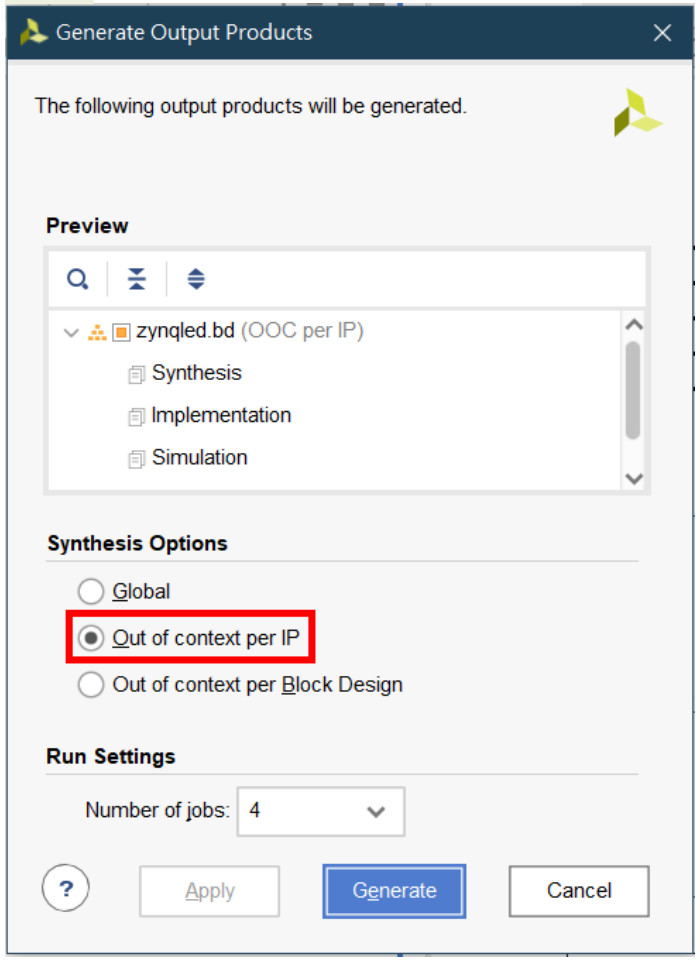


Generate output products

1

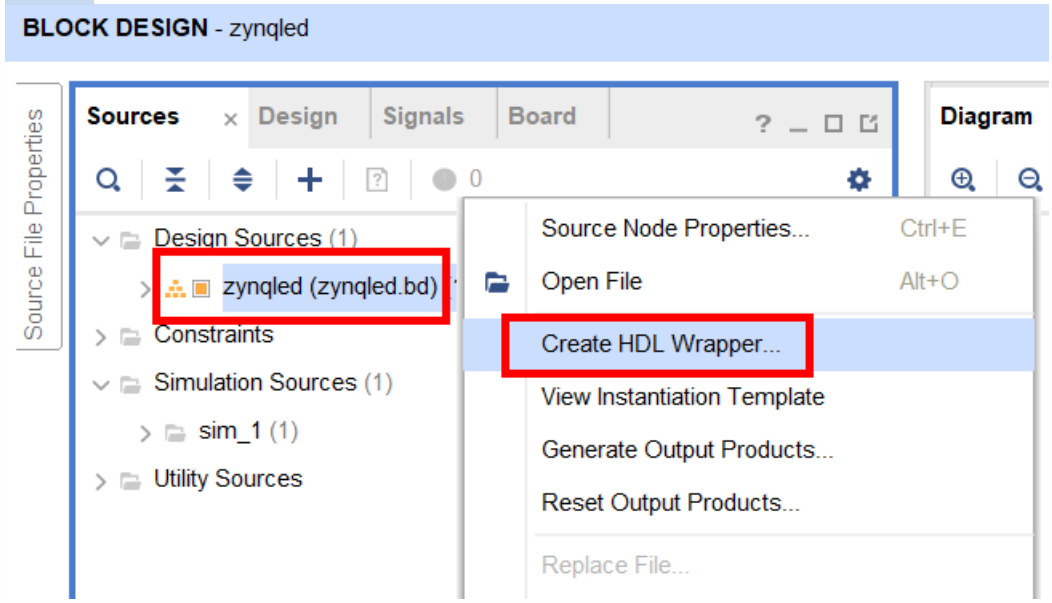


2

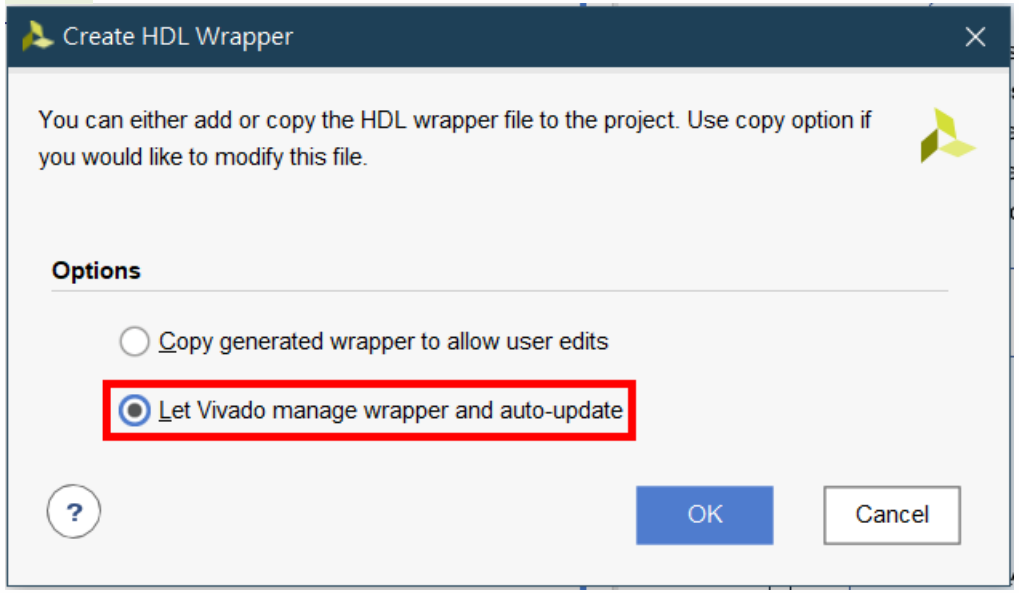


Create HDL wrapper

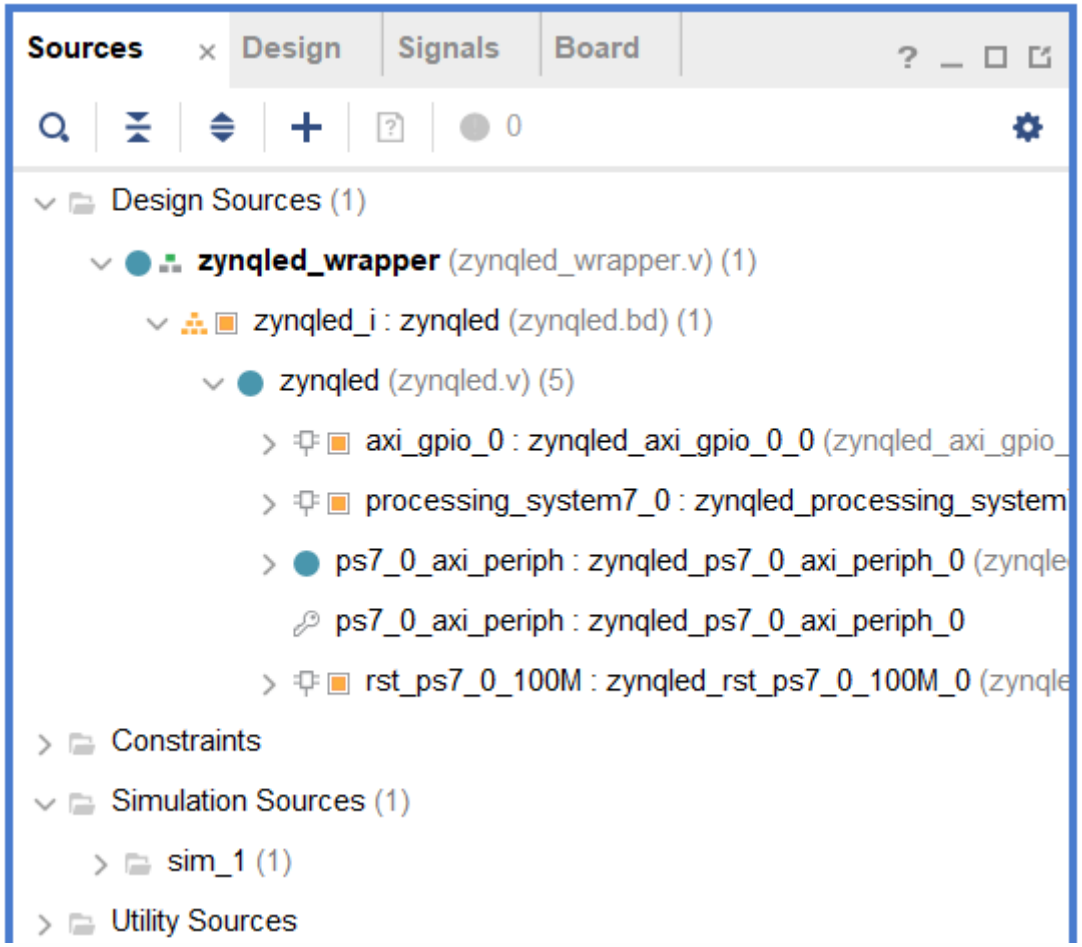
1



2

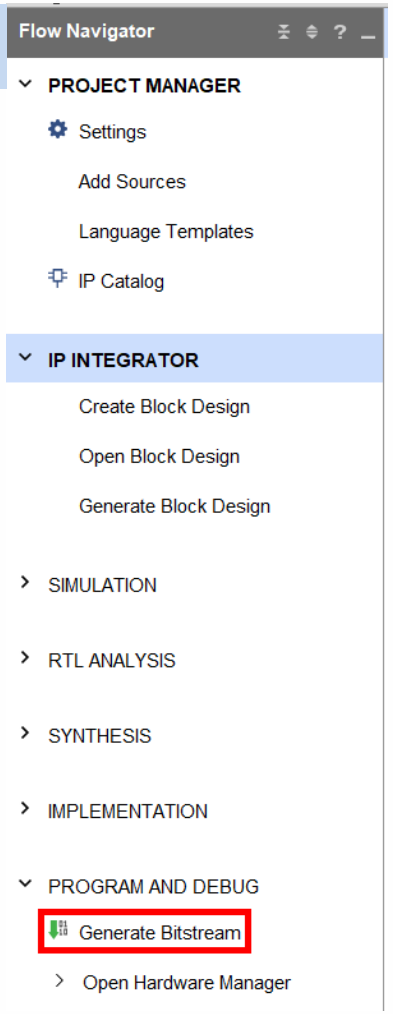


Create HDL wrapper

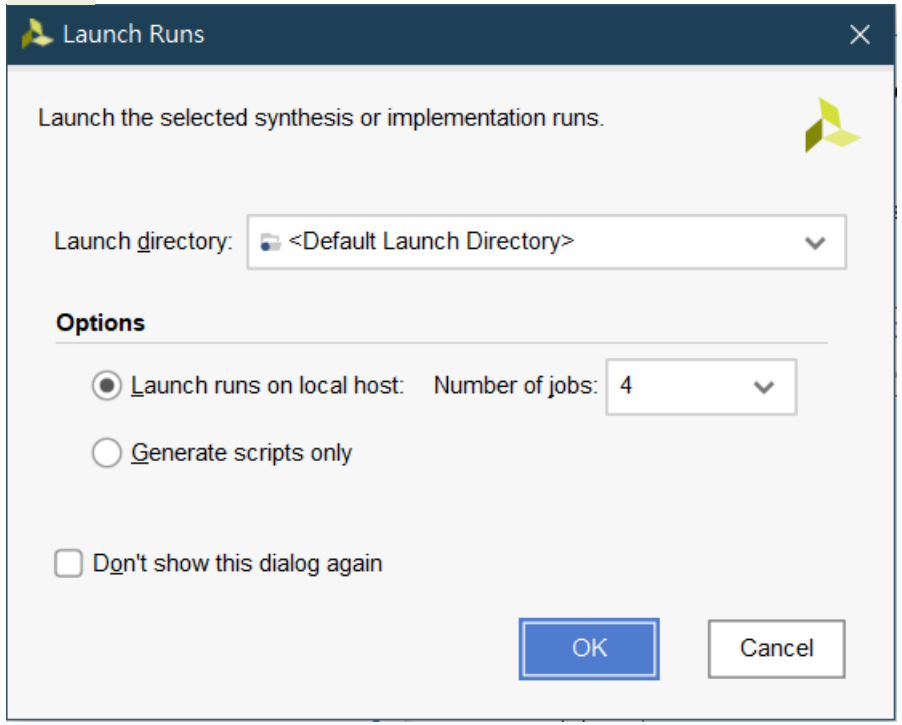


Generate Bitstream

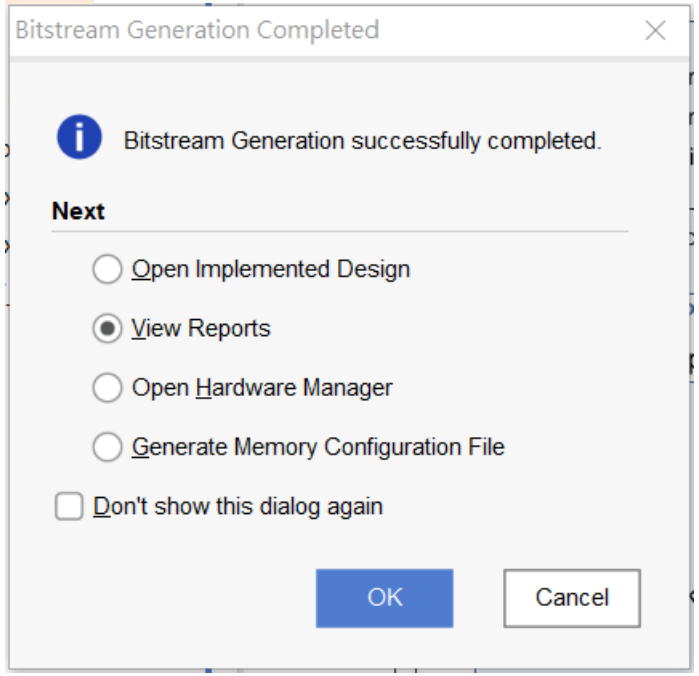
1



2

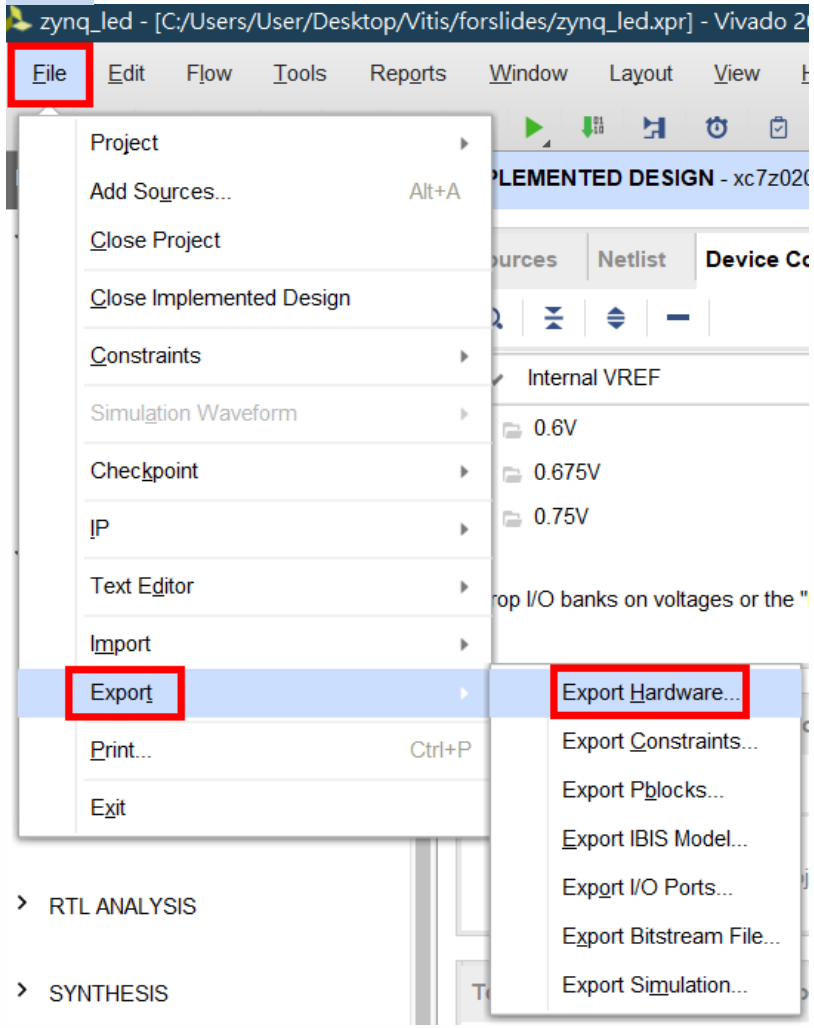


3

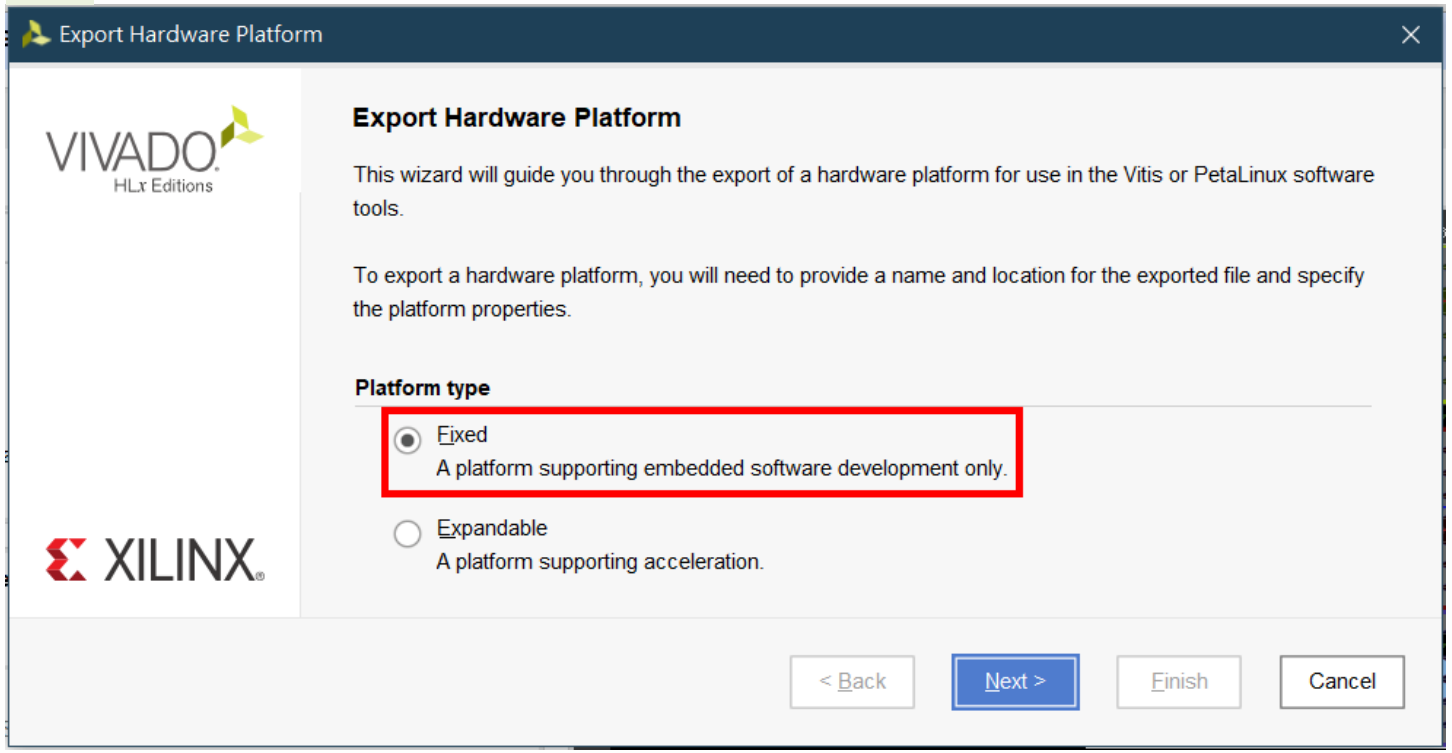


Export XSA (hardware specification)

1



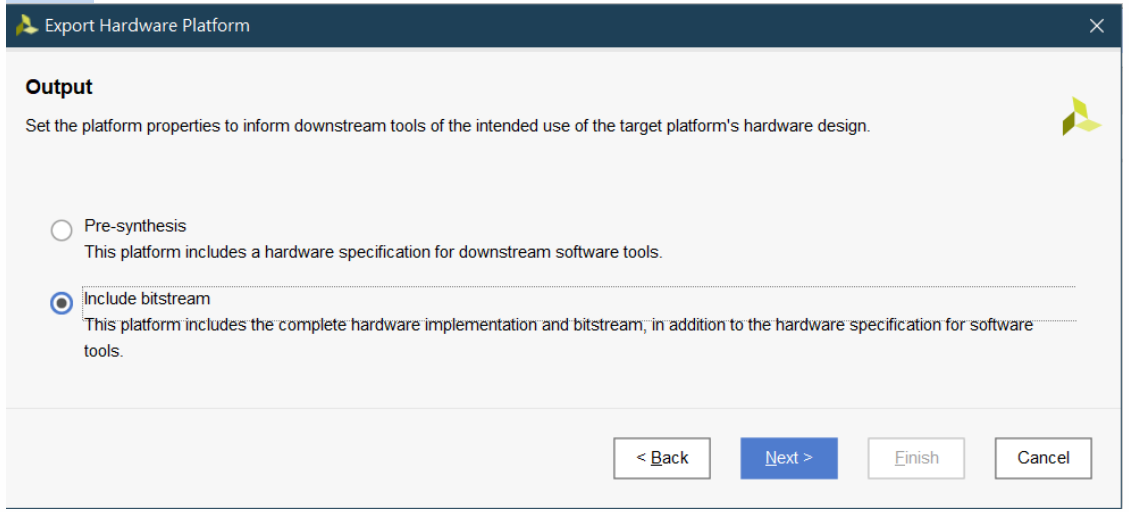
2



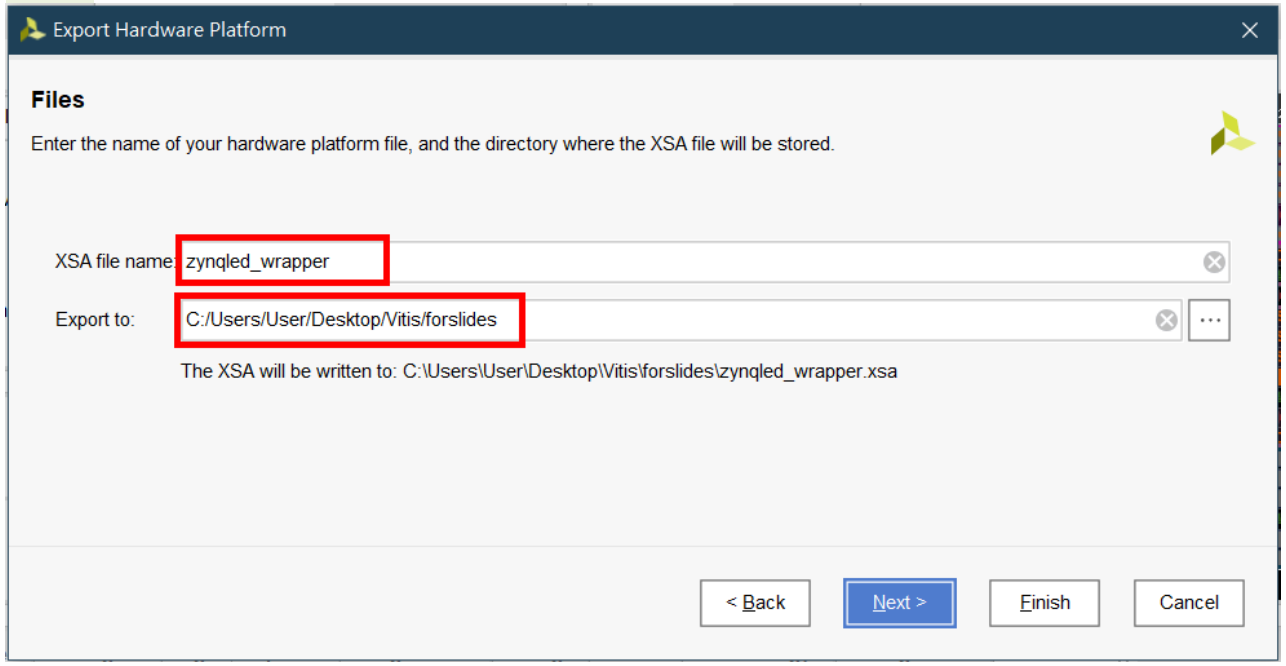
XILINX

Export XSA (hardware specification)

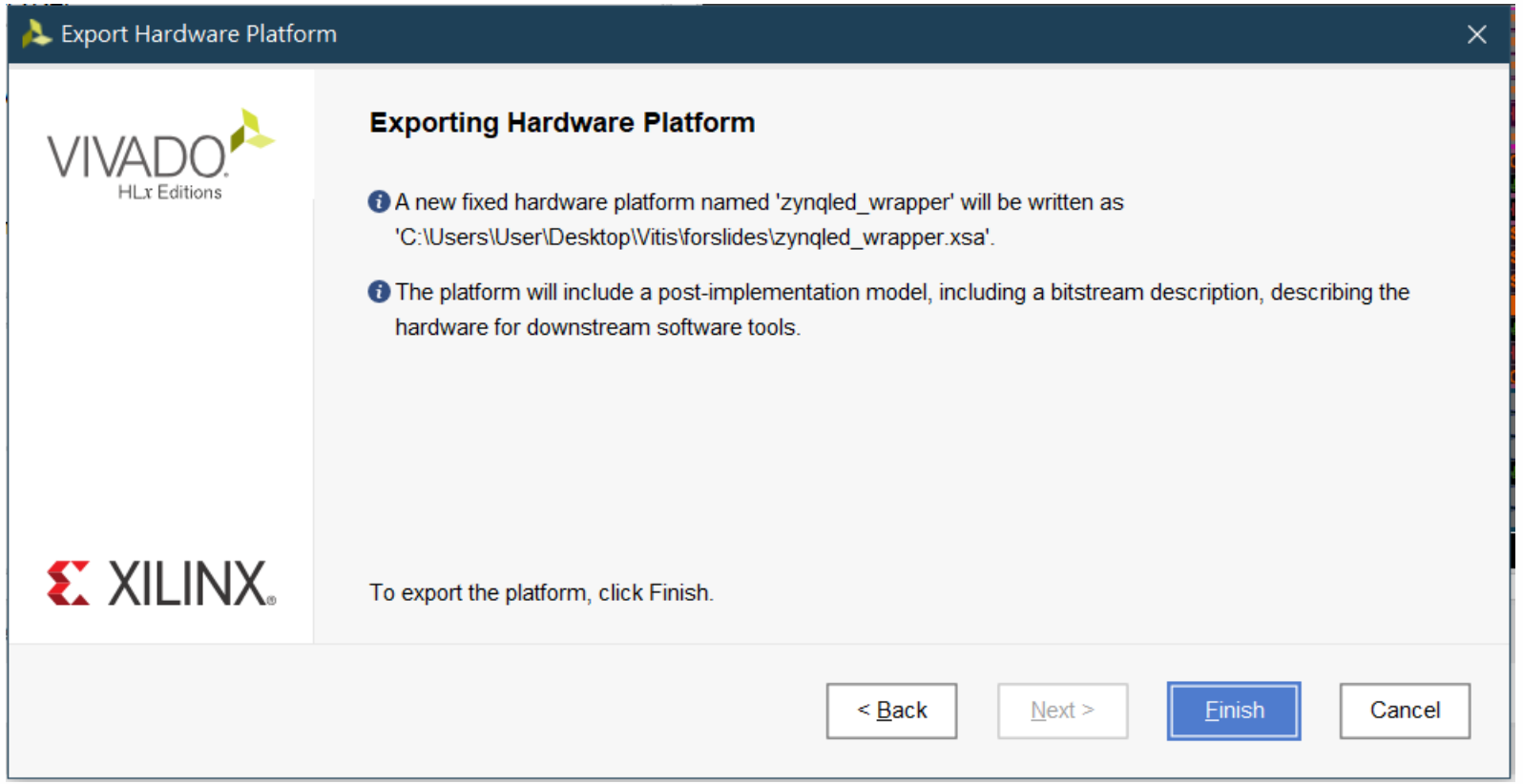
1



2



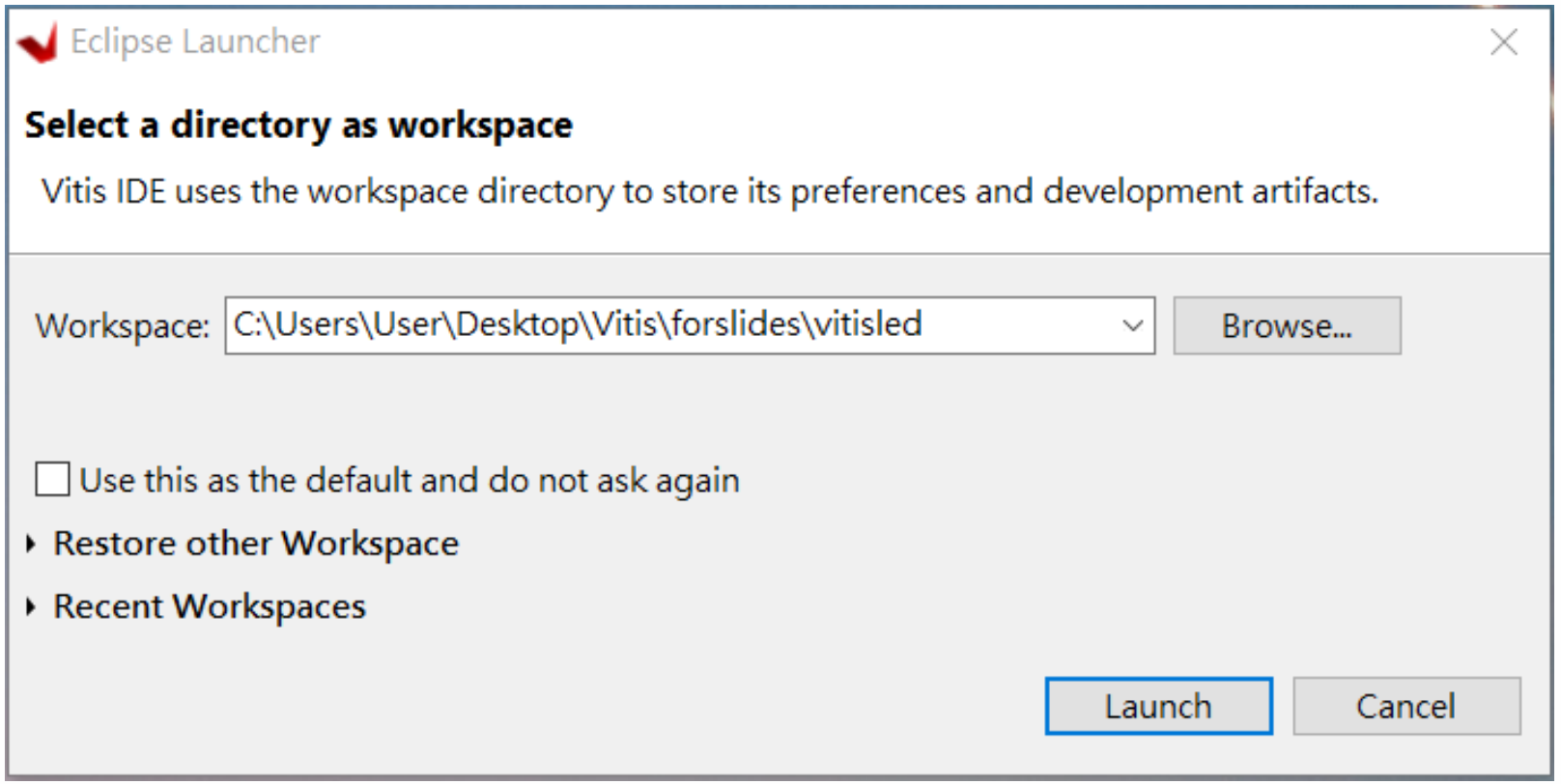
Export XSA (hardware specification)



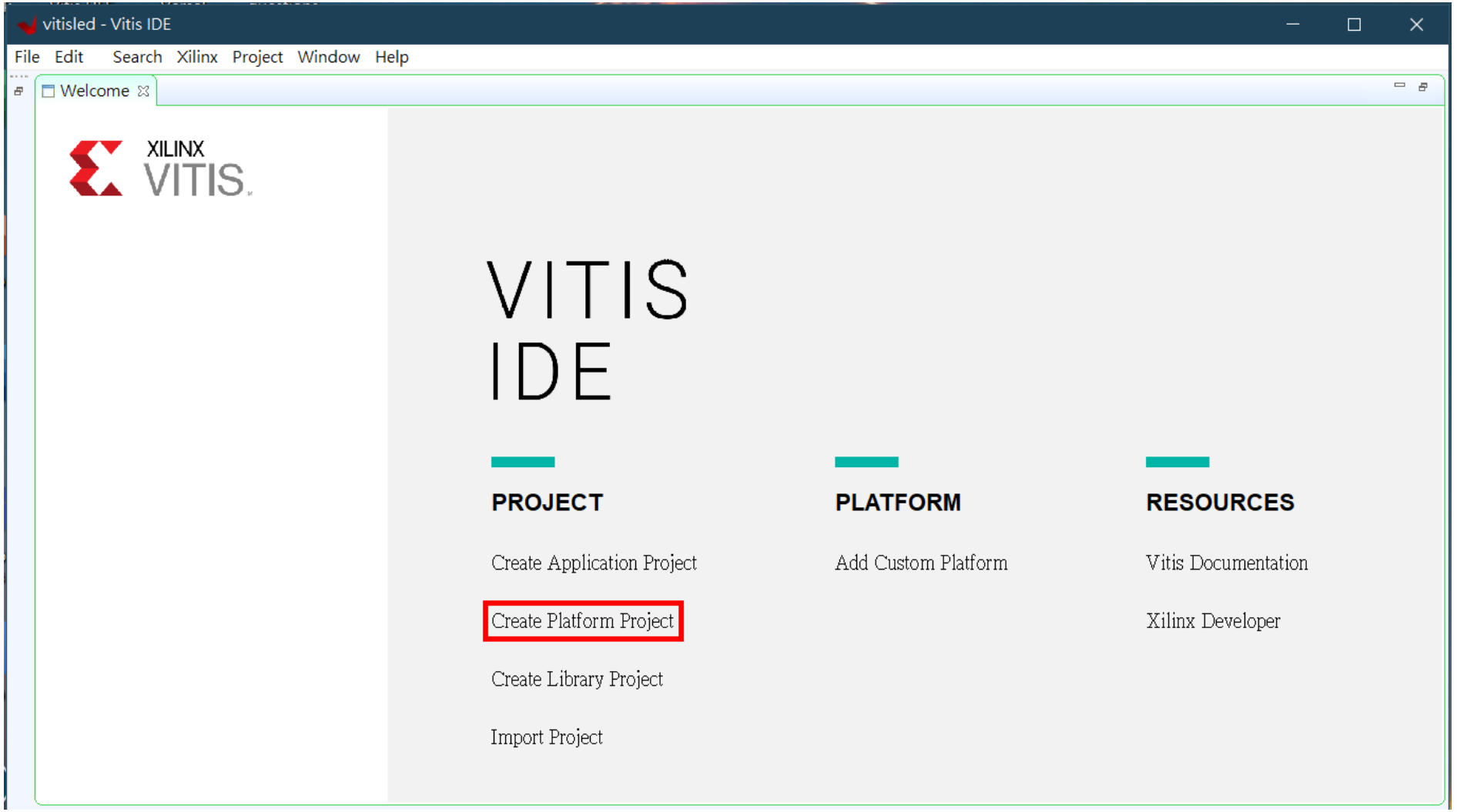
XILINX

Xilinx Vitis build

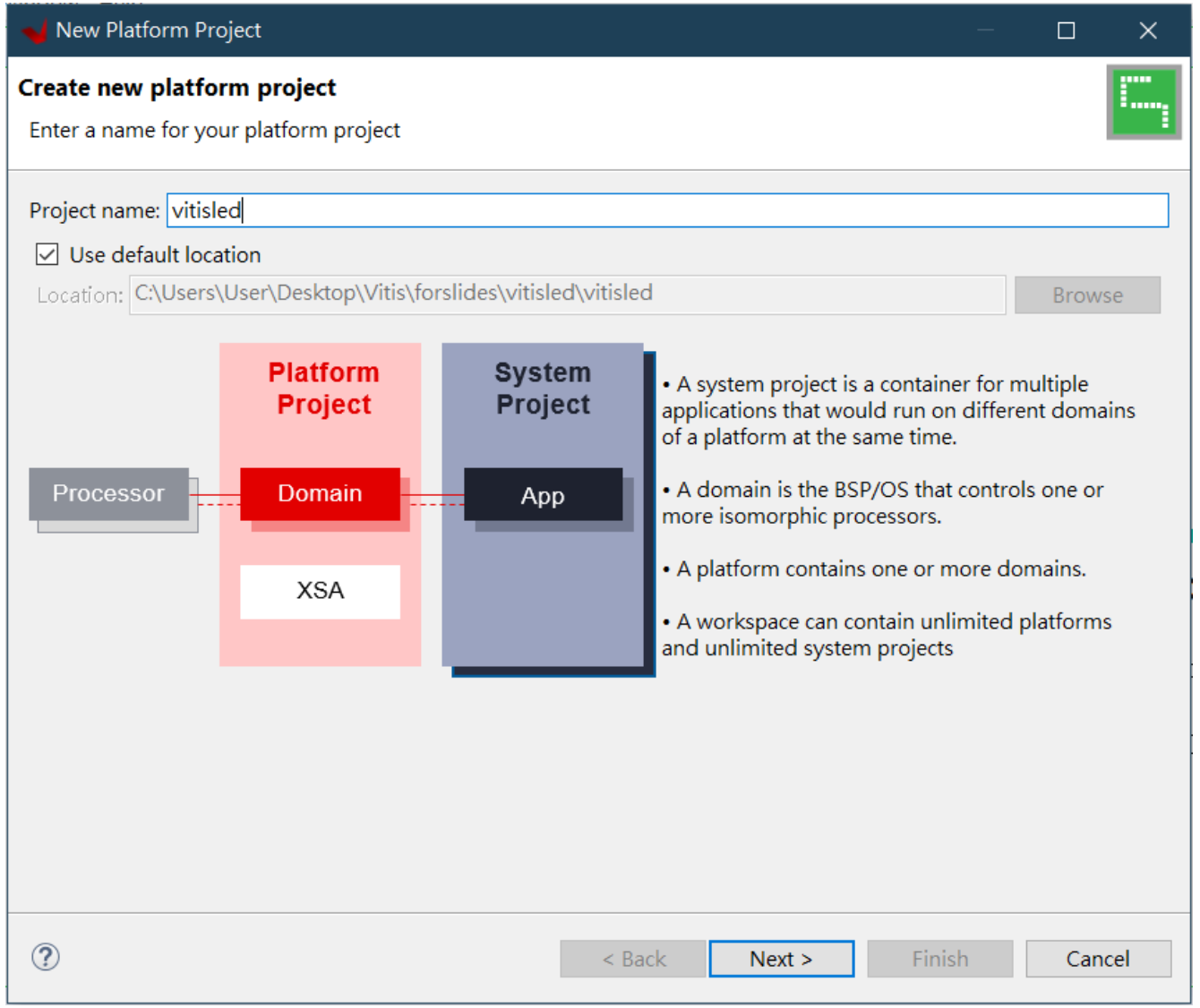
Launch Vitis



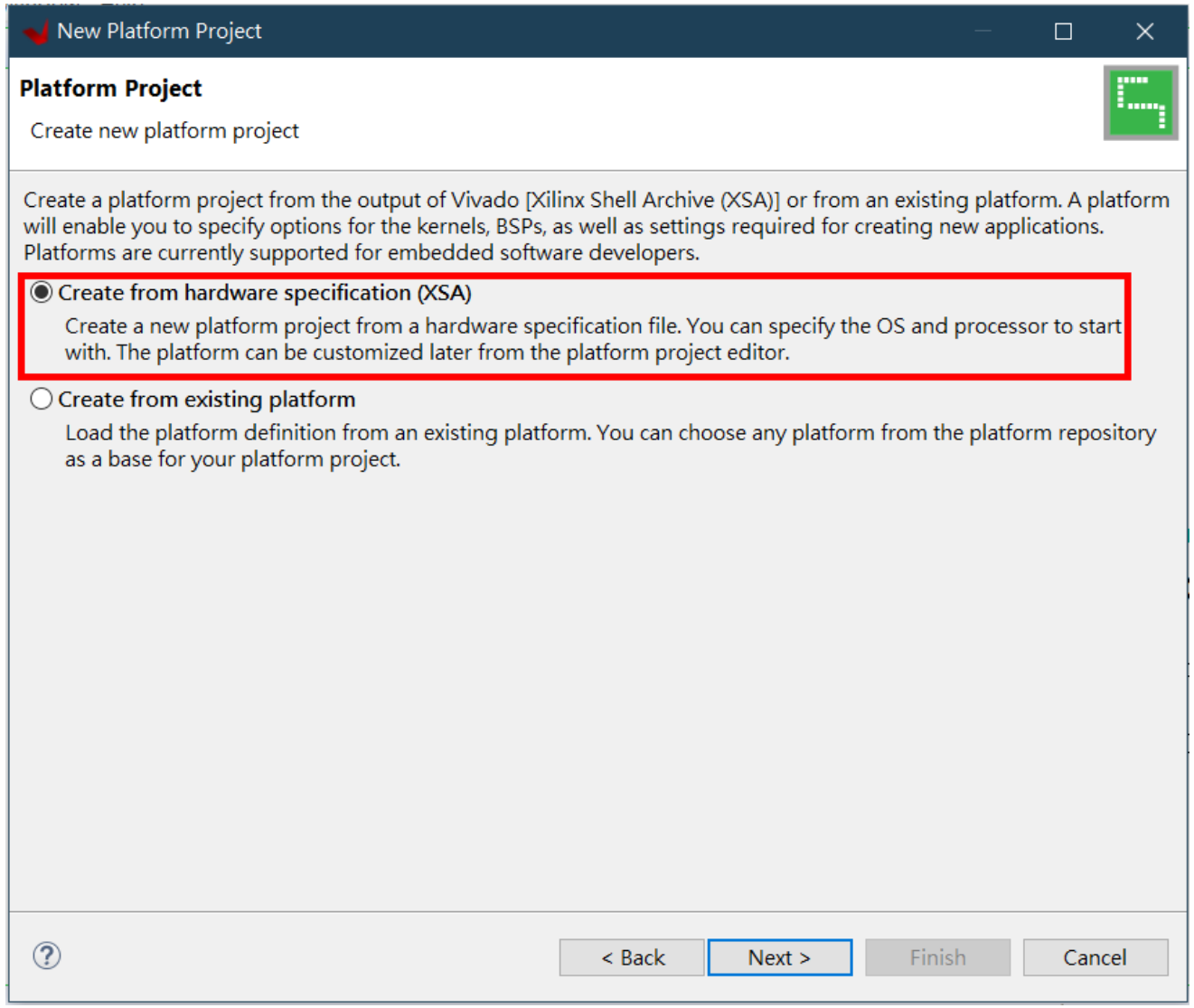
Create platform project



Create platform project

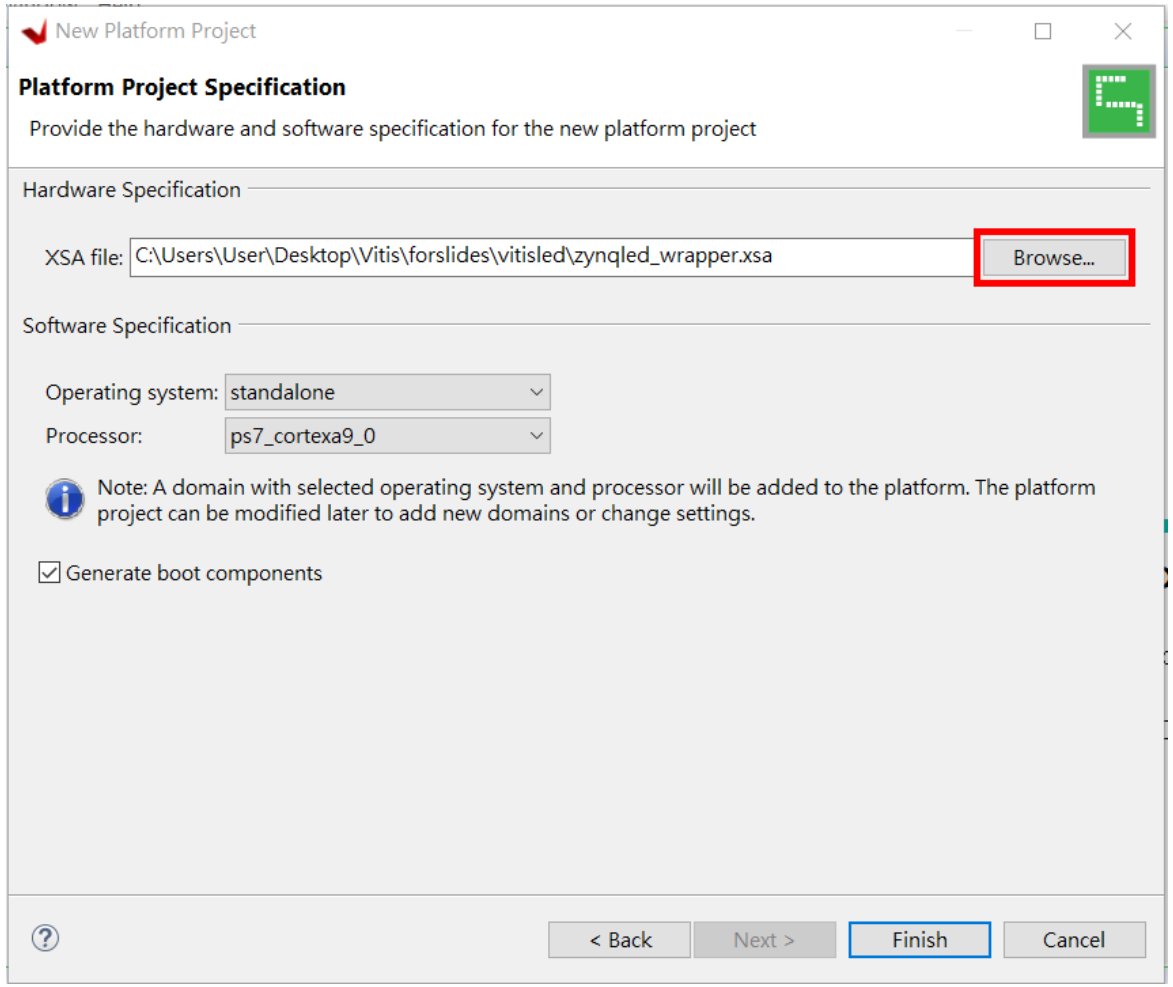


Create platform project

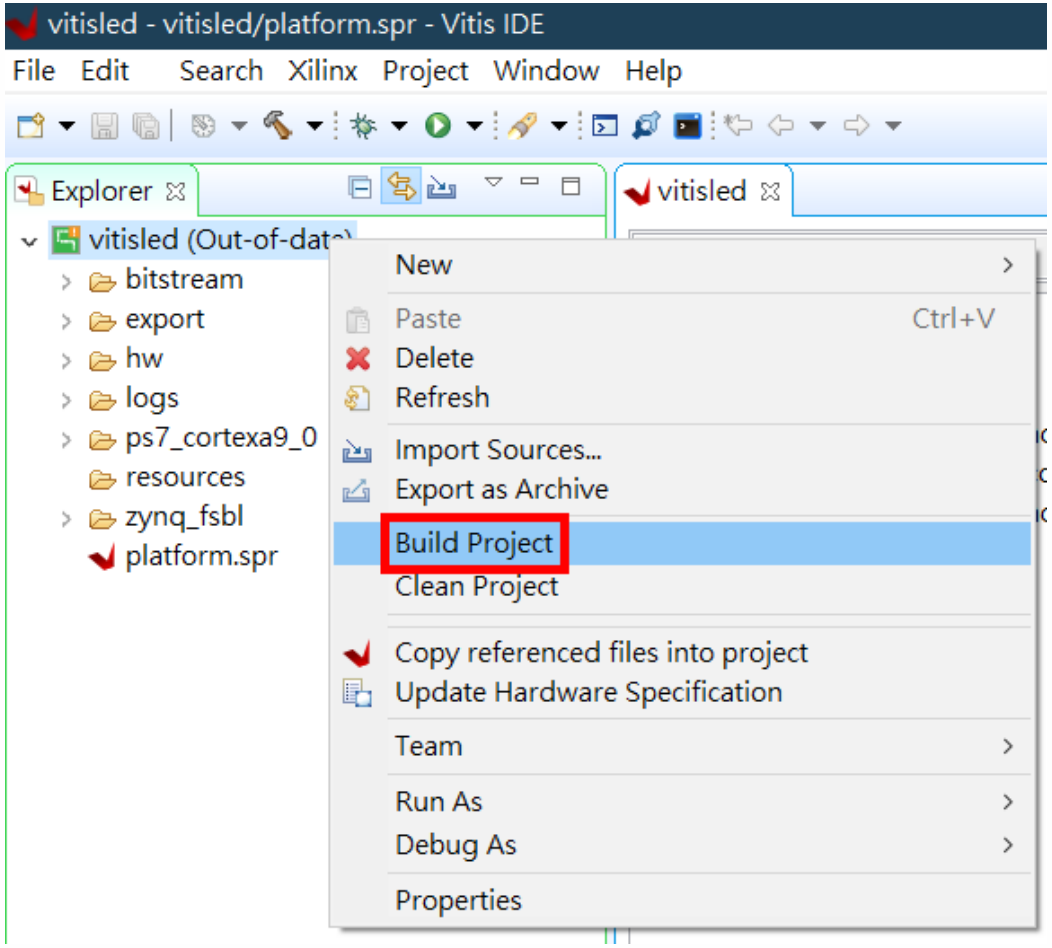


Create platform project

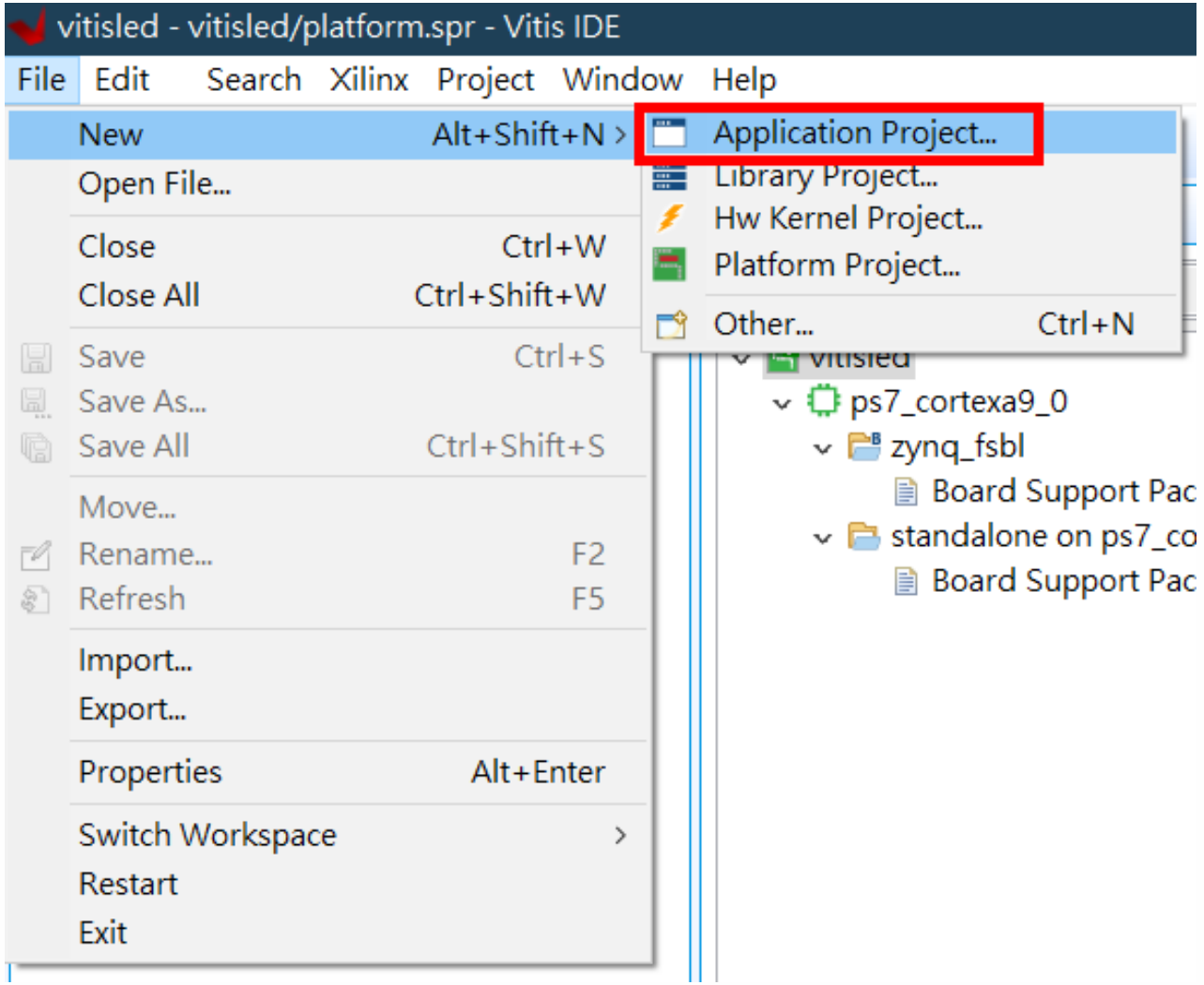
- In this step, select the .xsa file which was created previously.



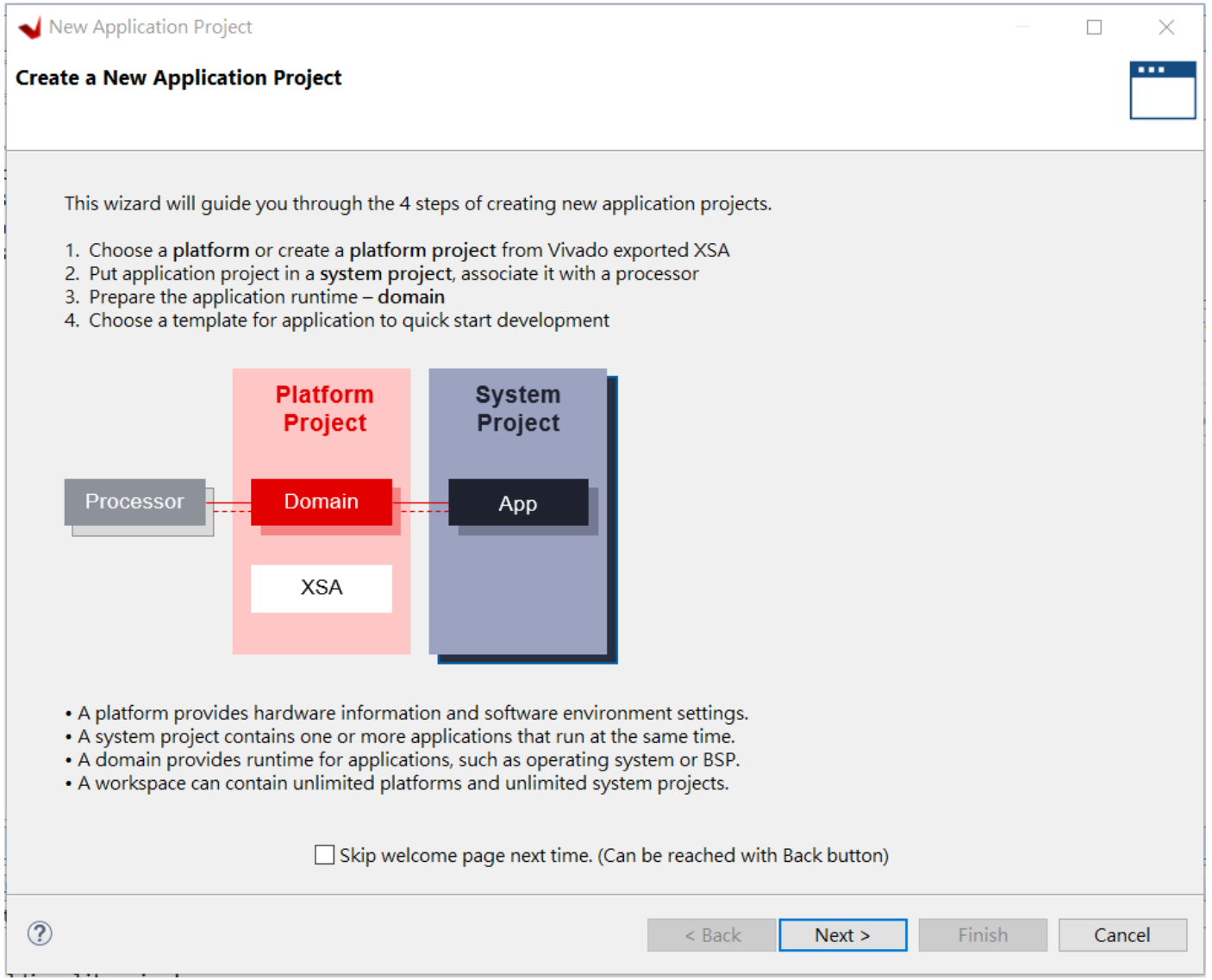
Build platform



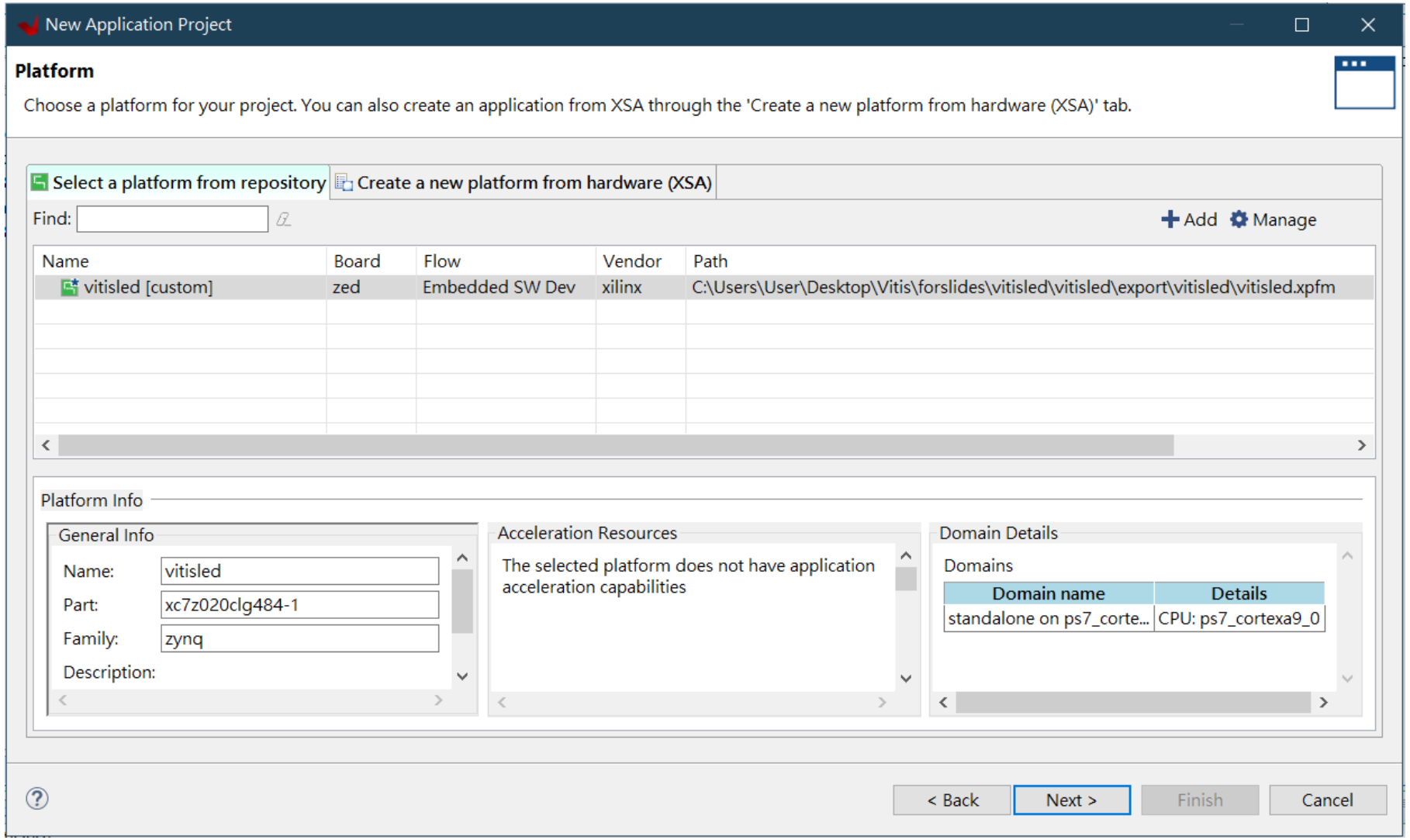
Create applications



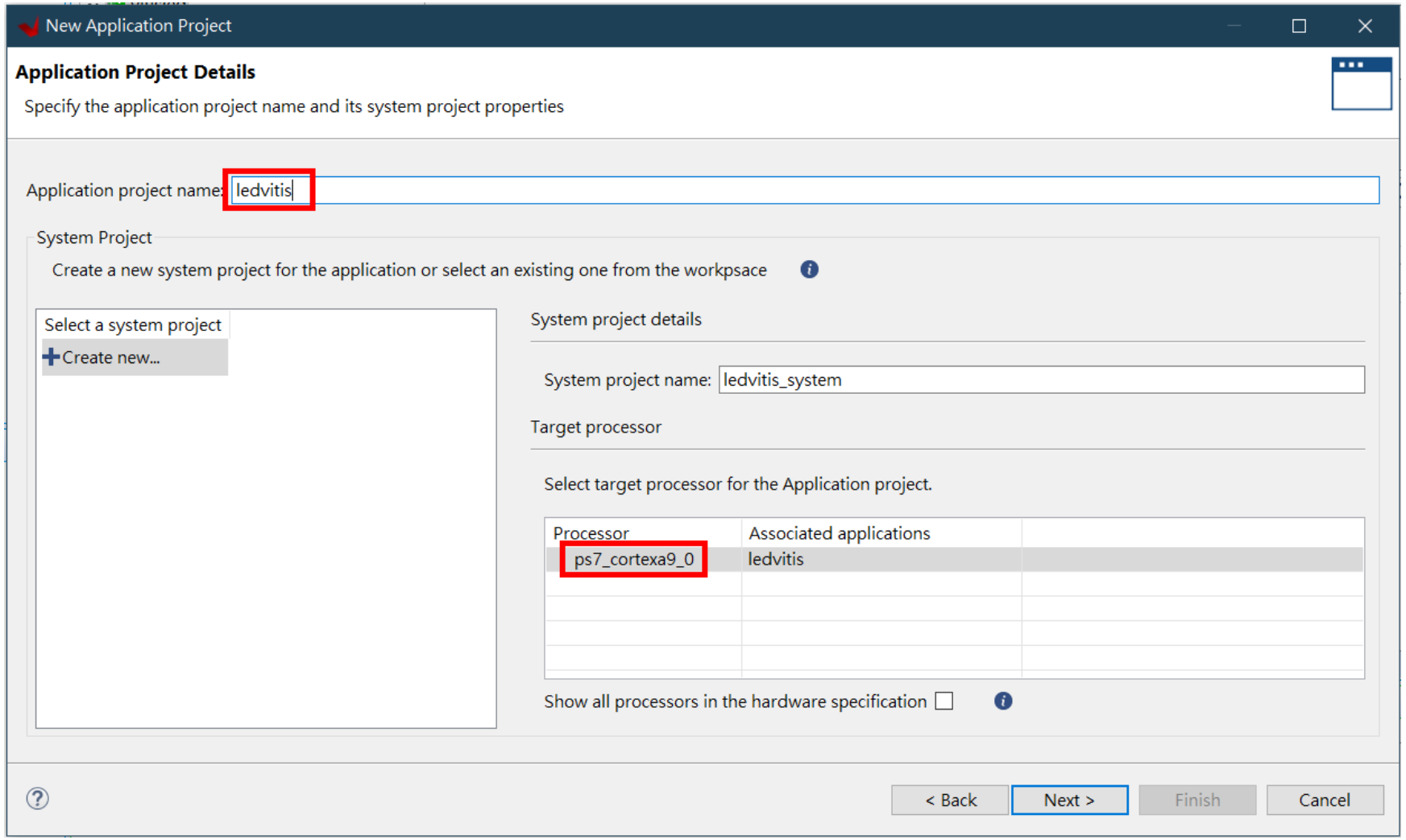
Create applications



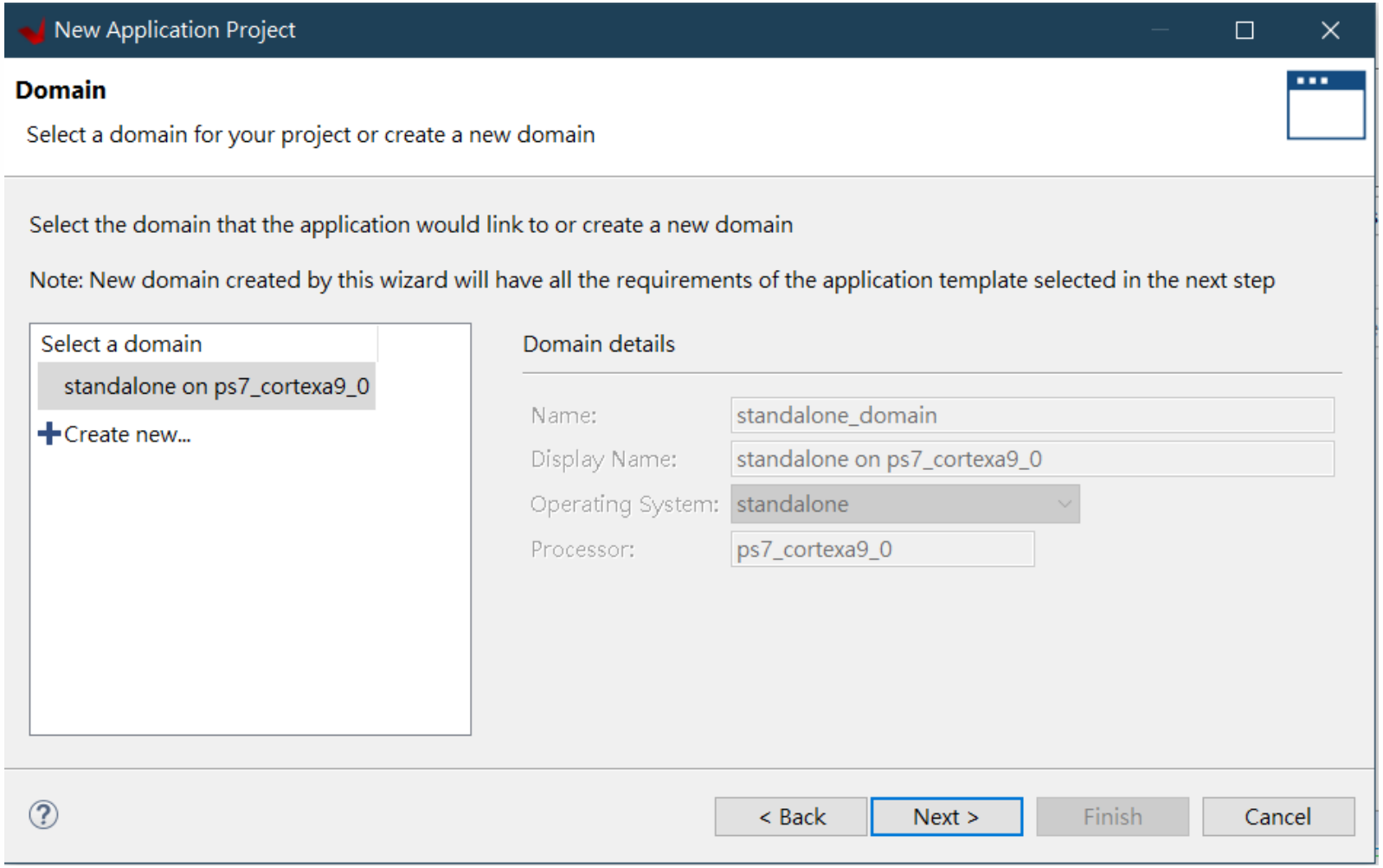
Create applications



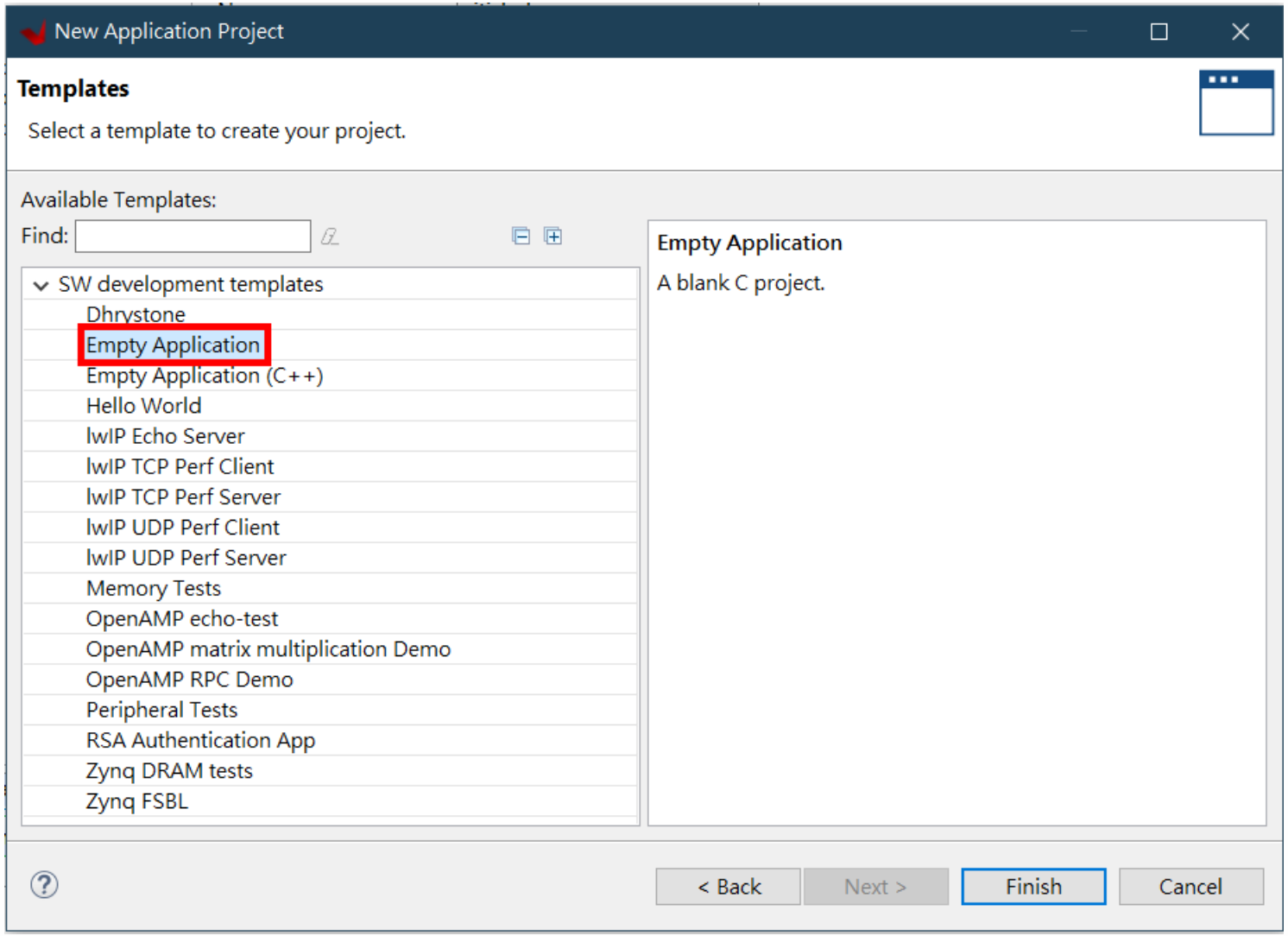
Create applications



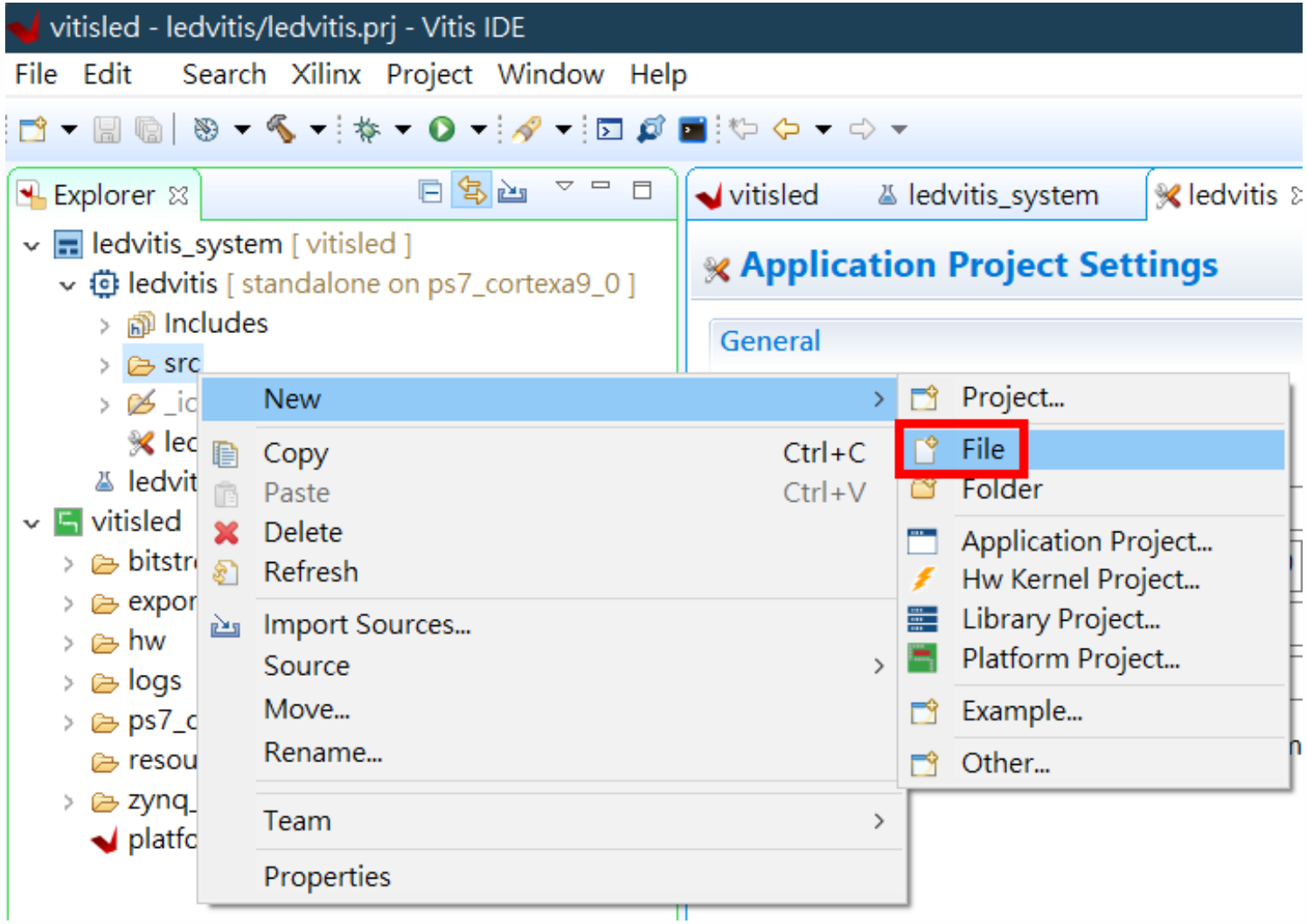
Create applications



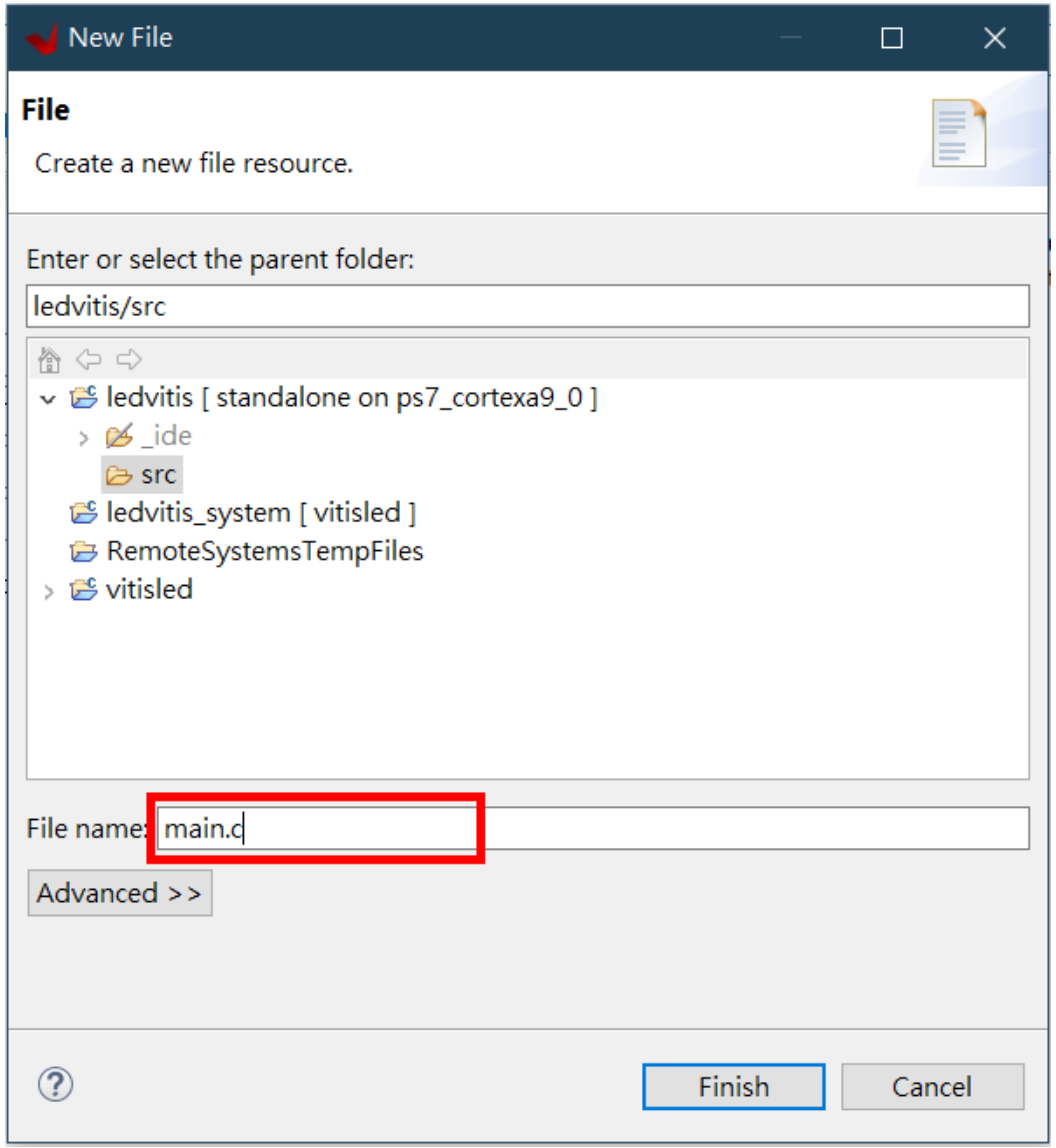
Create applications



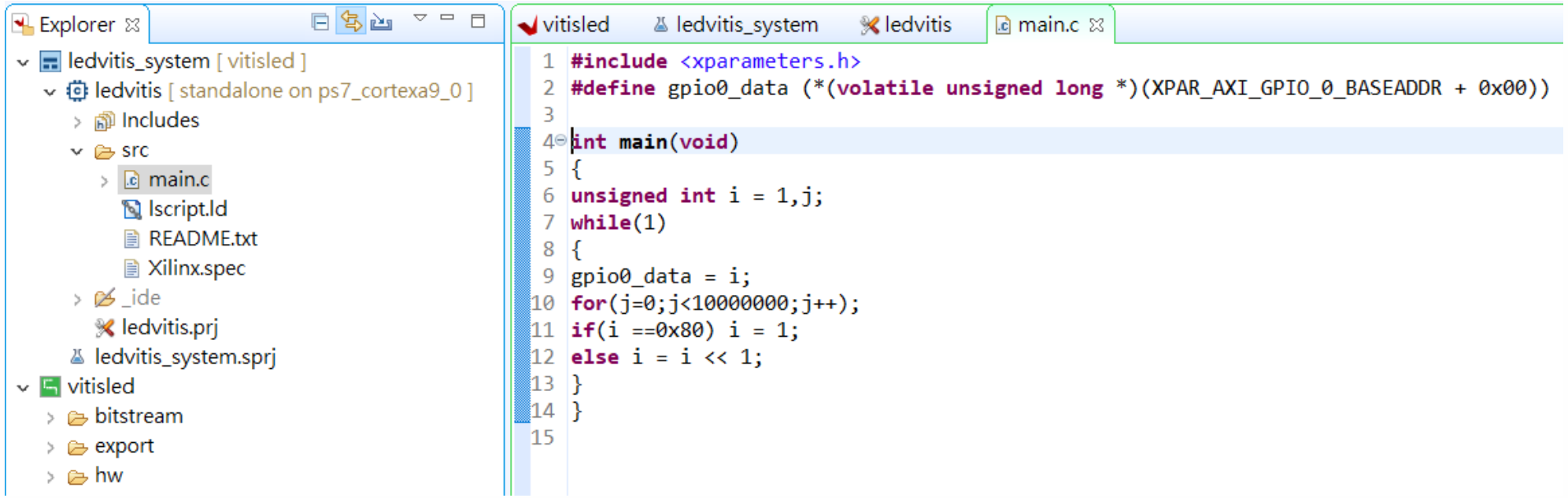
Add a new source to “src”



Add a new source to “src”



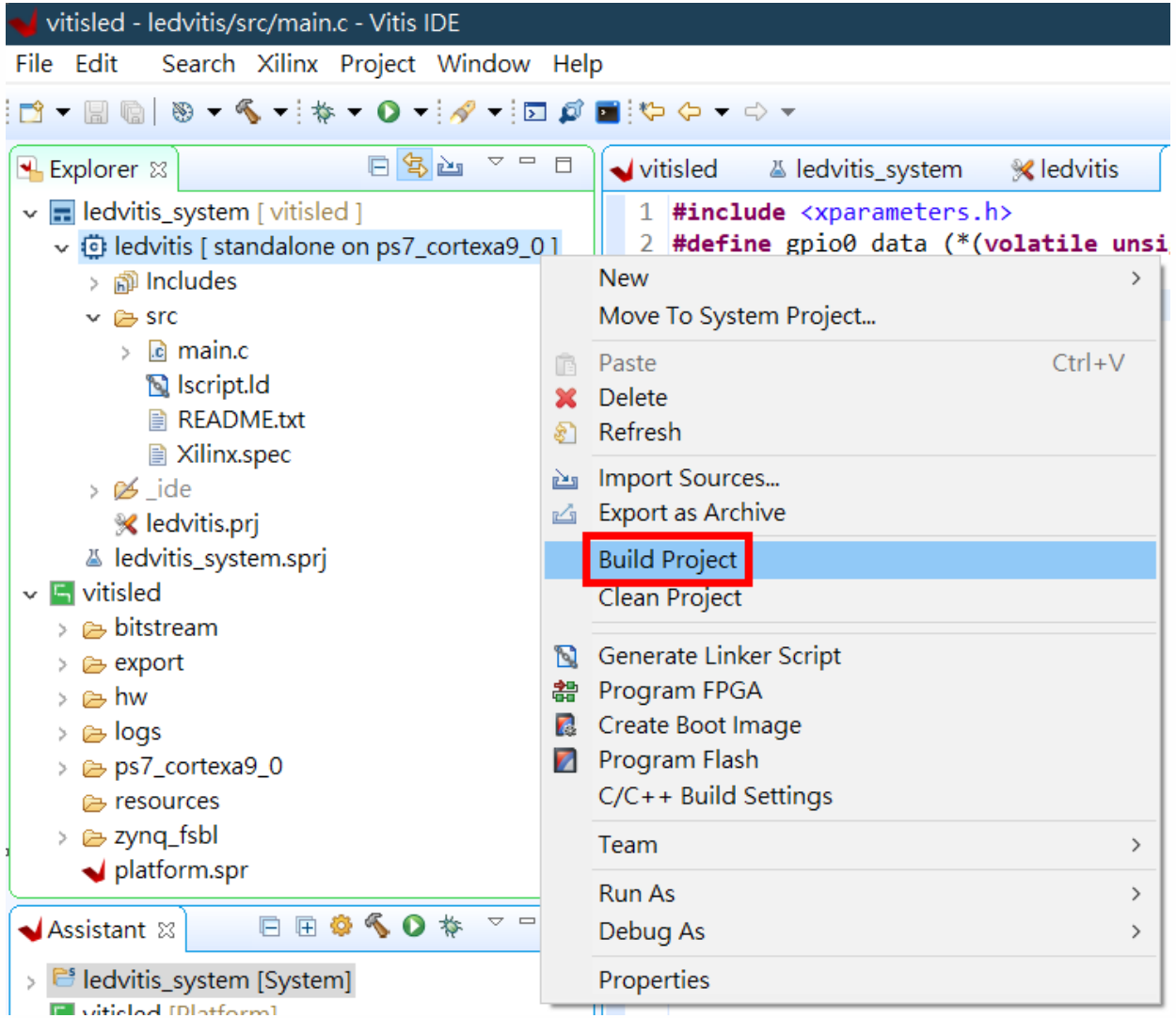
Add a new source to “src”



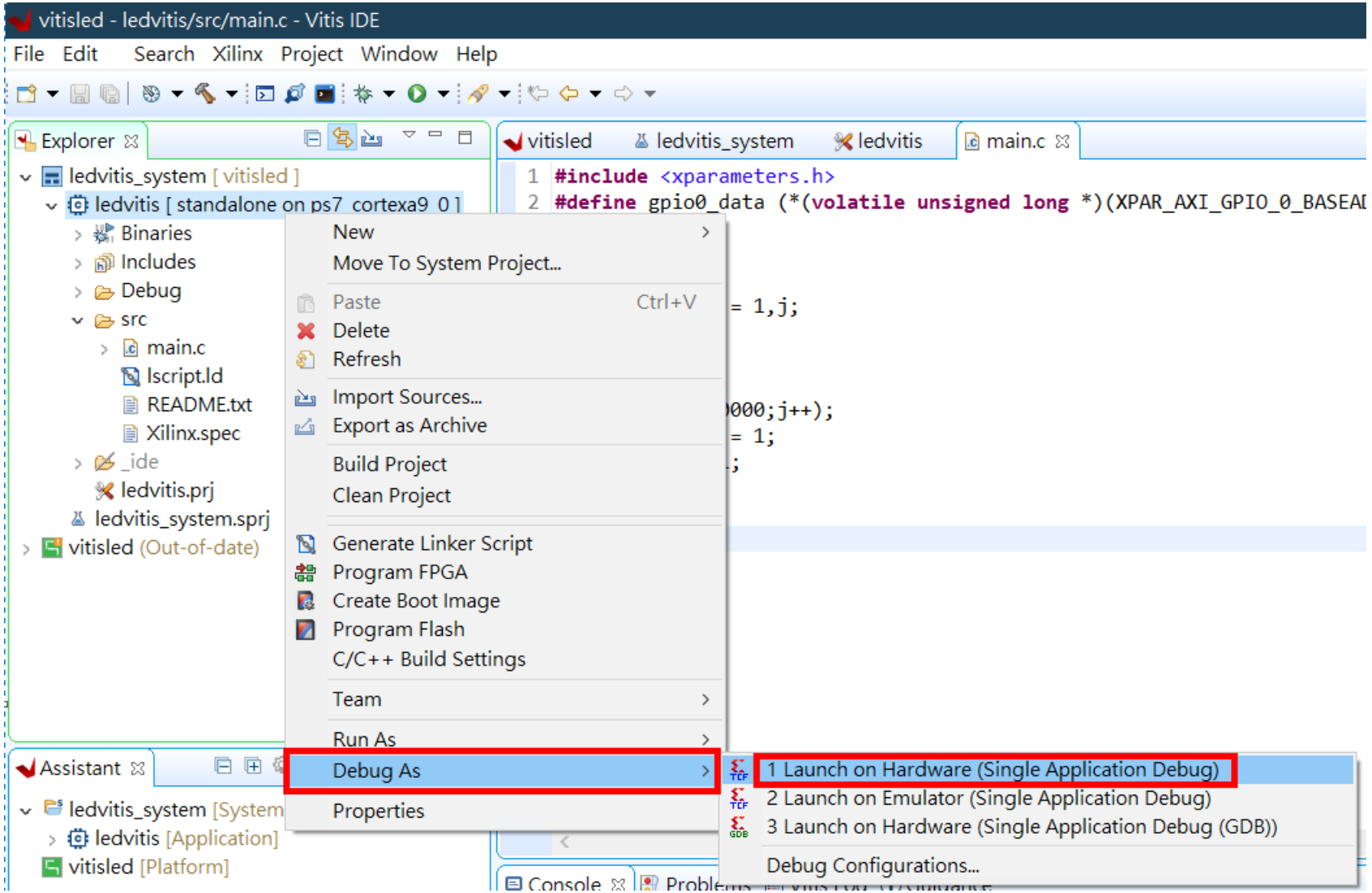

```
#include <xparameters.h>
#define gpio0_data (*(volatile unsigned long *)(XPAR_AXI_GPIO_0_BASEADDR + 0x00))

int main(void)
{
unsigned int i = 1,j;
while(1)
{
gpio0_data = i;
for(j=0;j<10000000;j++);
if(i ==0x80) i = 1;
else i = i << 1;
} }
```

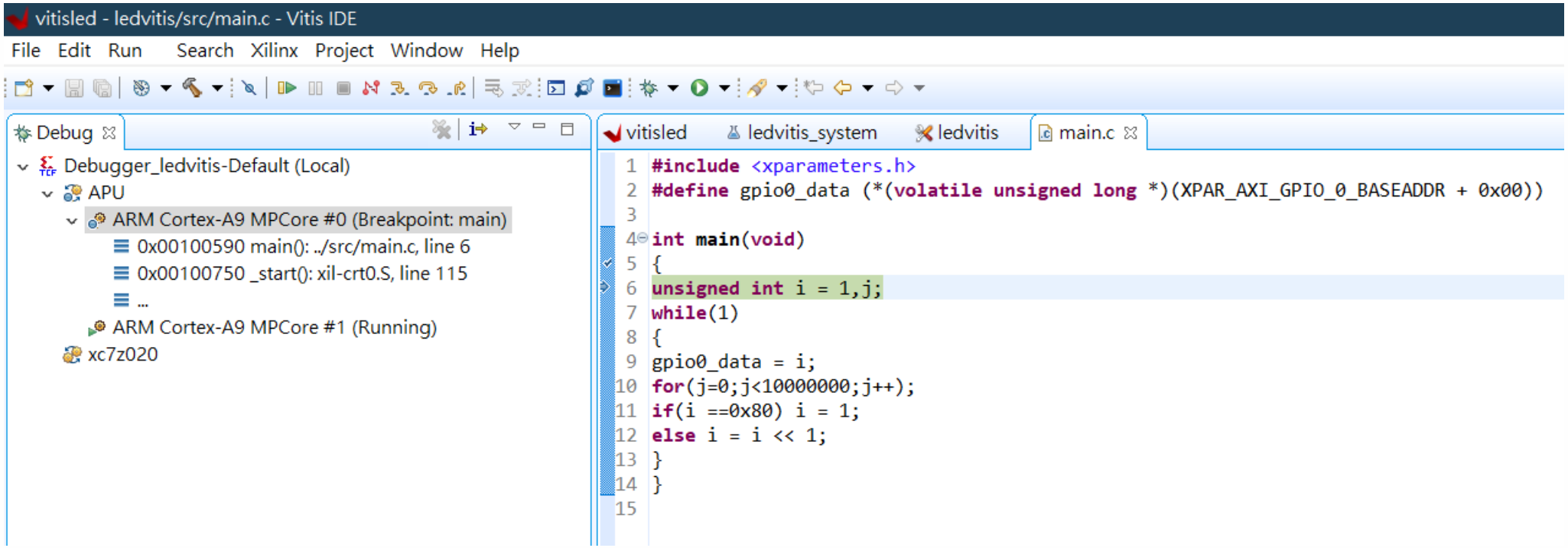
Build the application



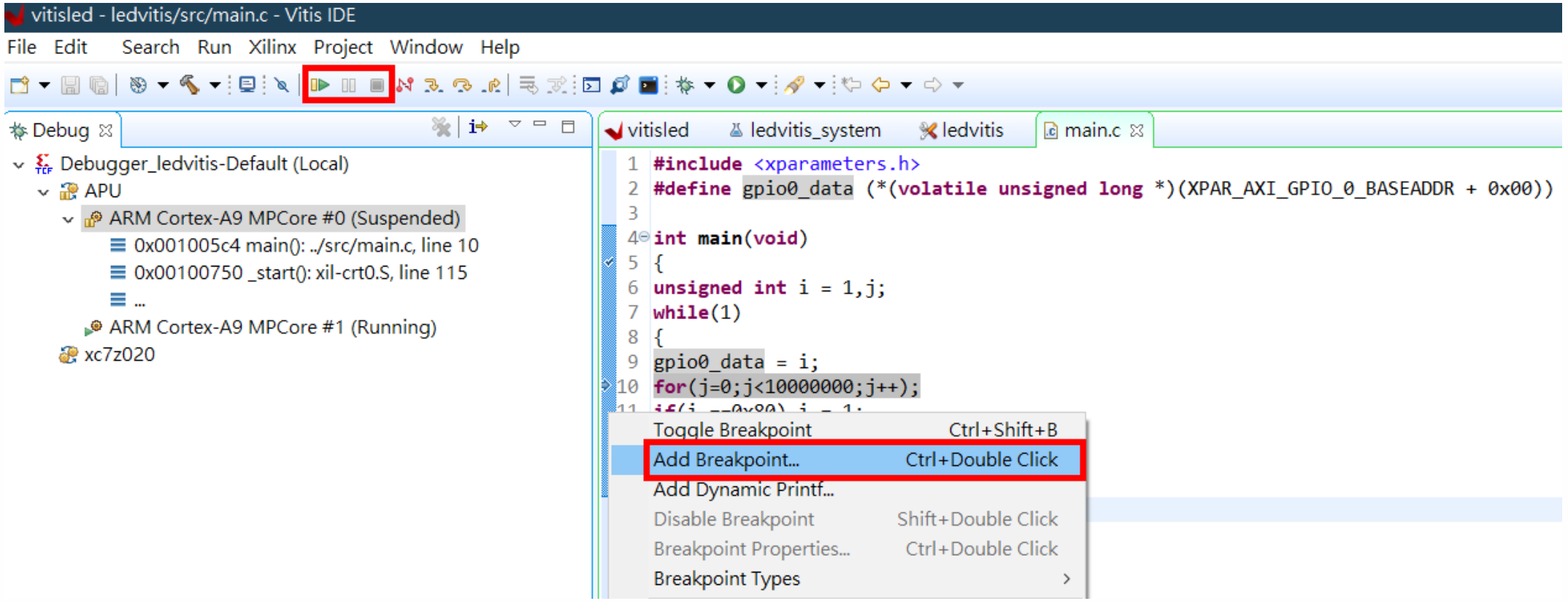
Application debug



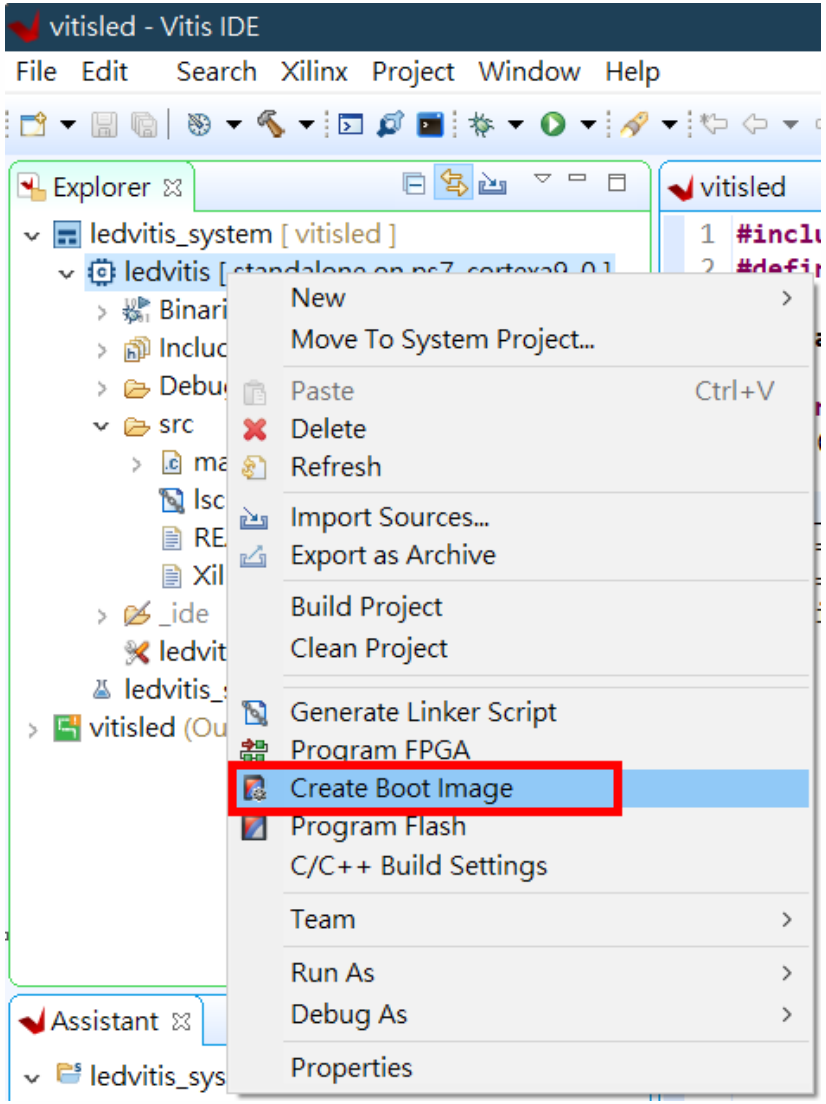
Application debug



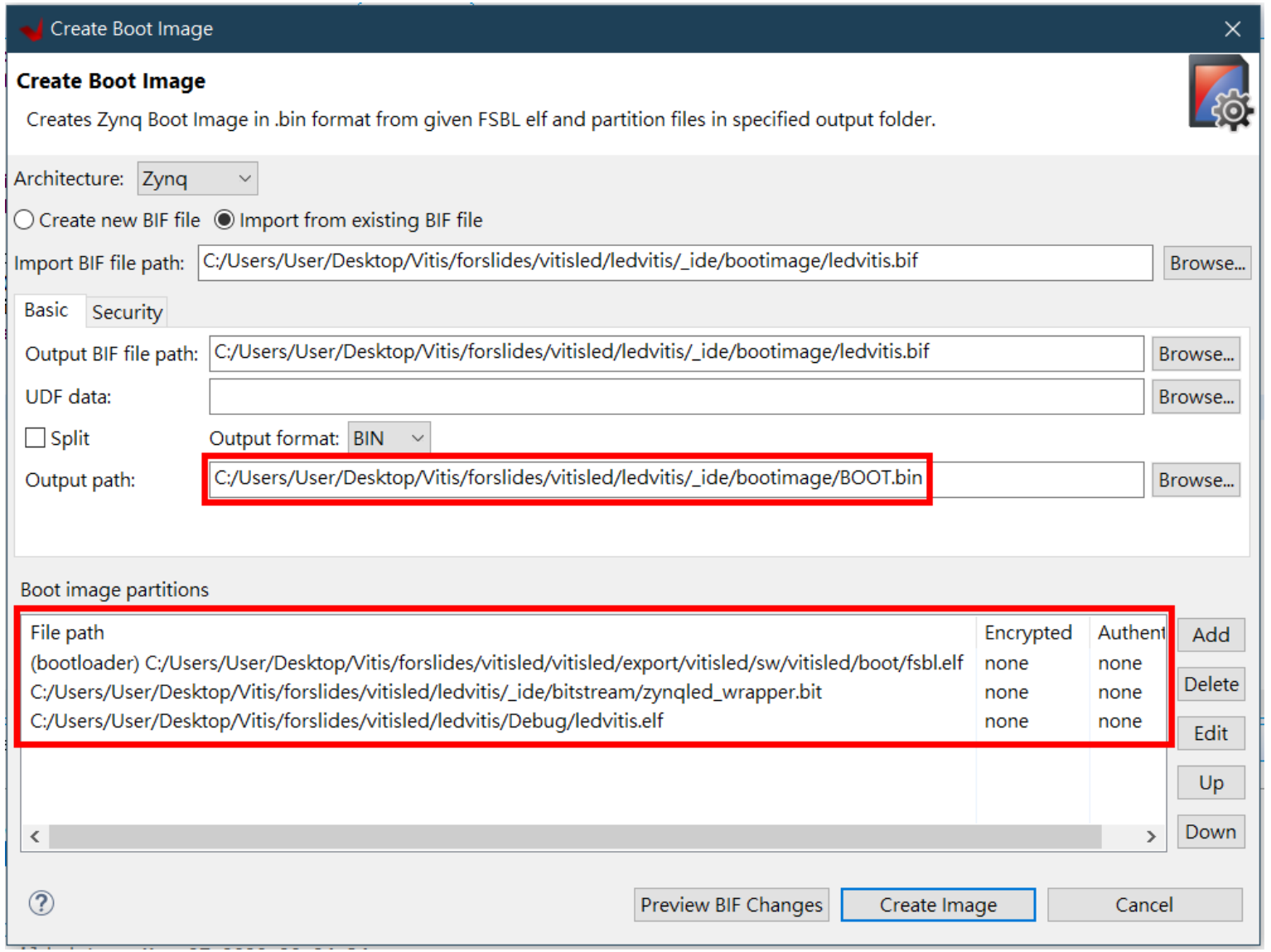
Application debug



Create boot image



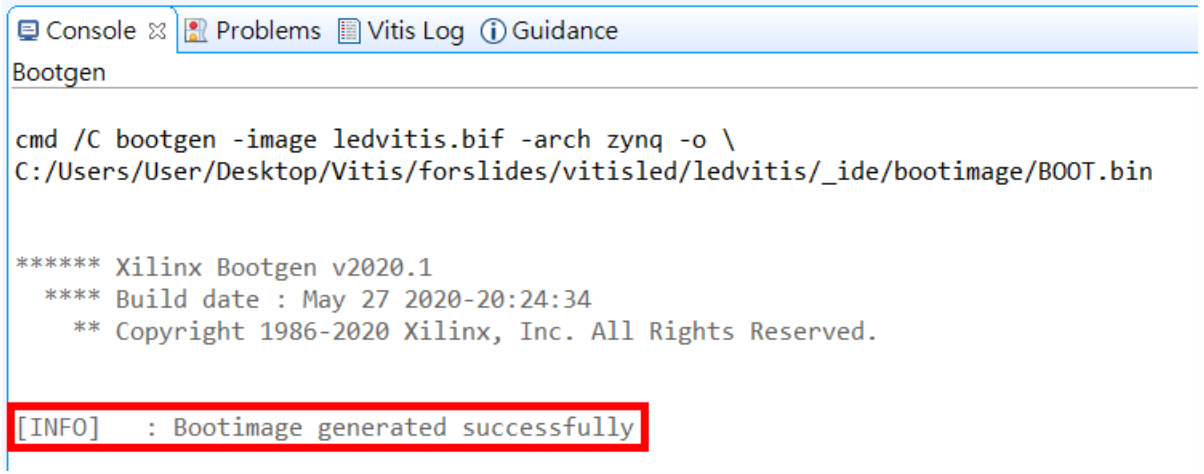
Create boot image



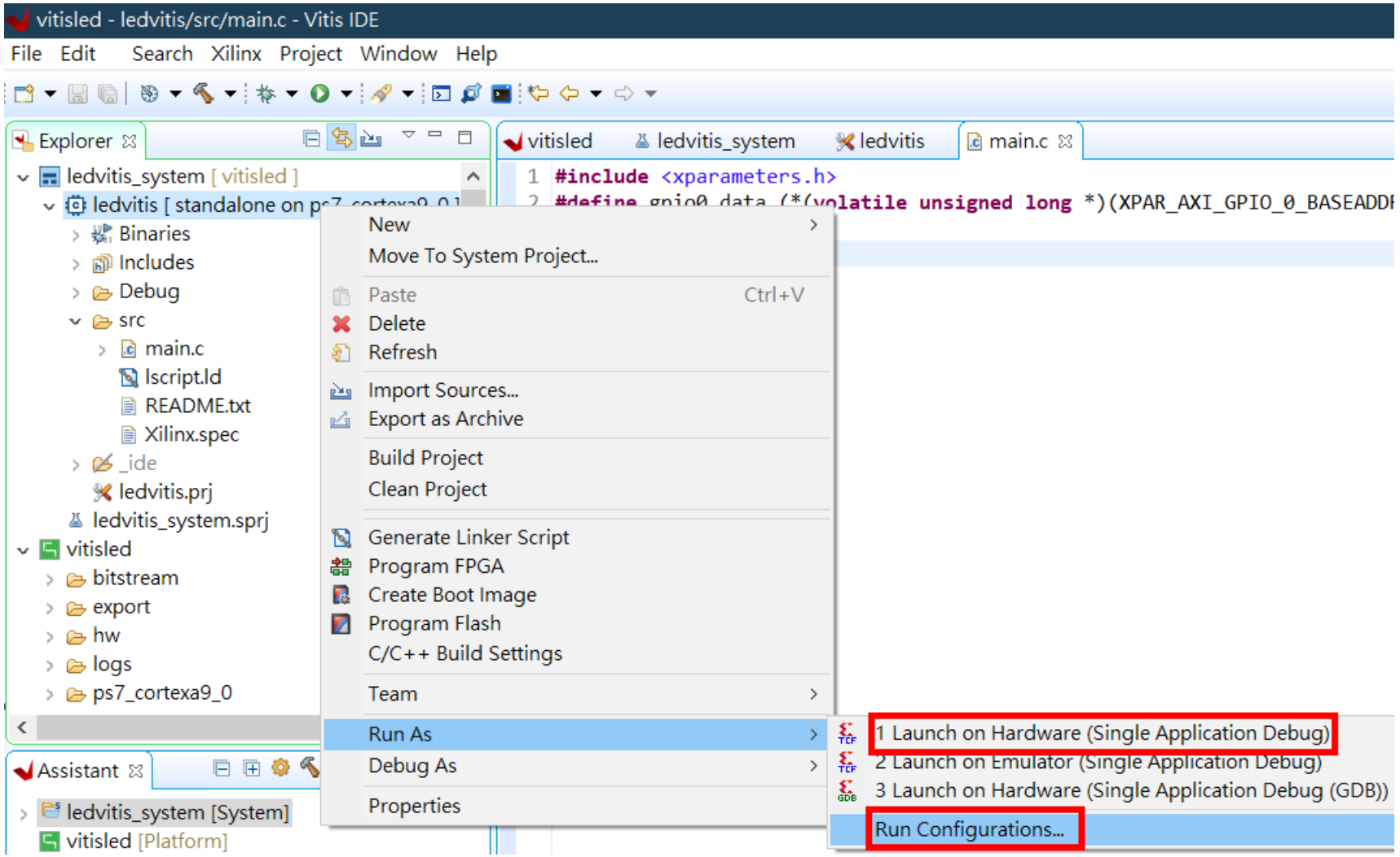
Create boot image

- After the boot image has been created, it would be stored at : <path-to-application>/_ide/bootimage/BOOT.bin

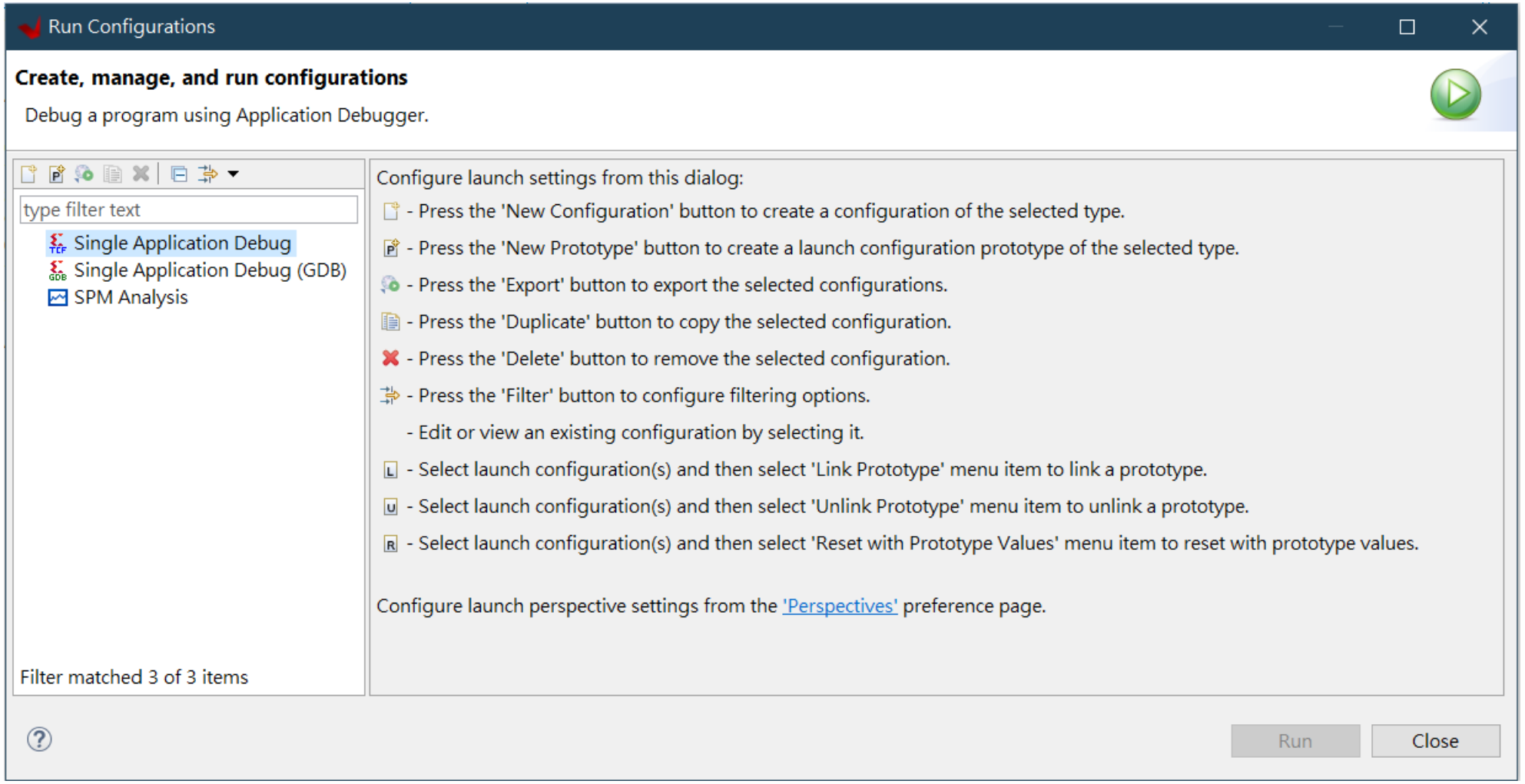
- Then put the BOOT.bin into a SD card that was formatted with “FAT32” previously.



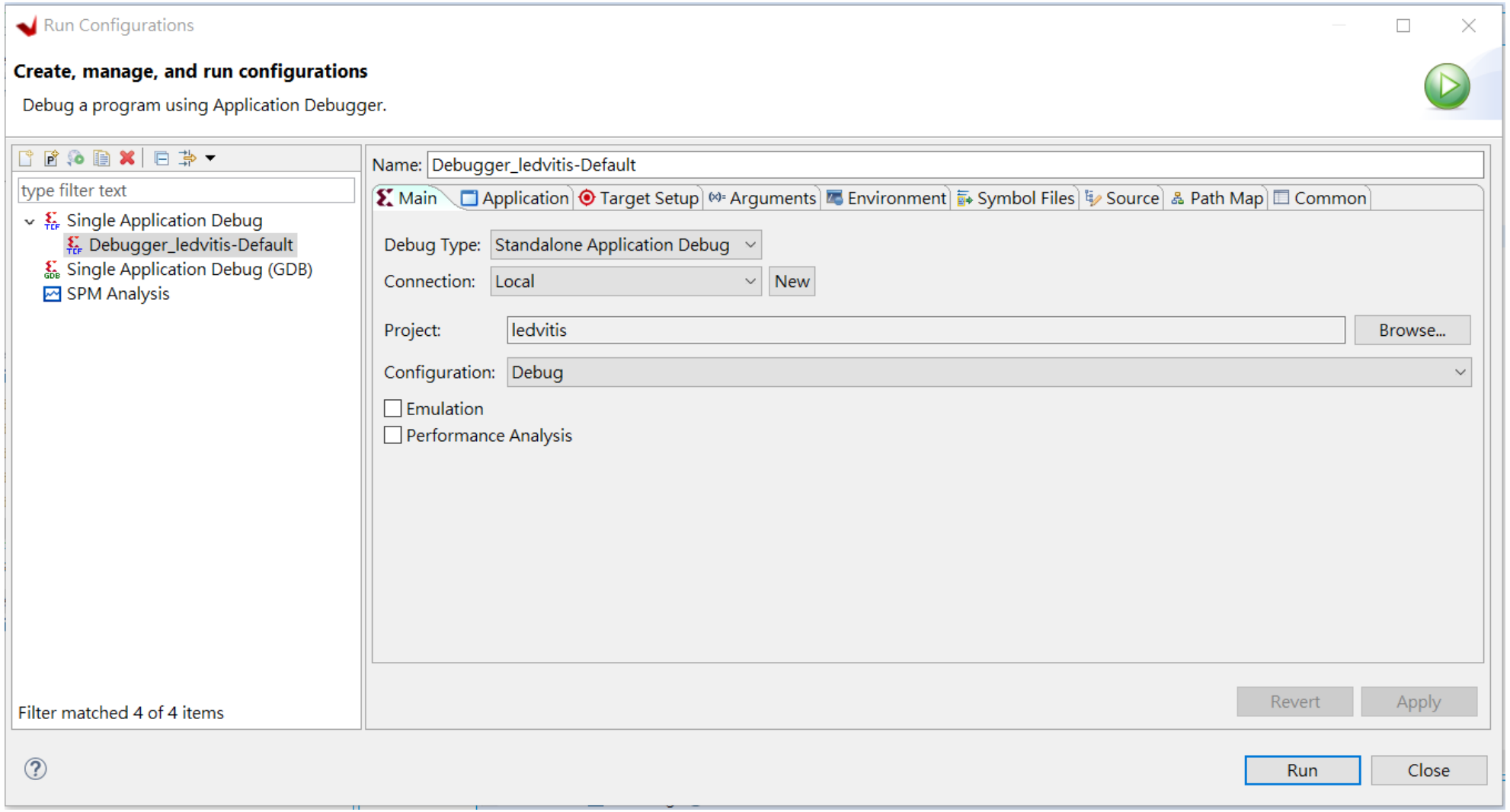
Set up run configurations



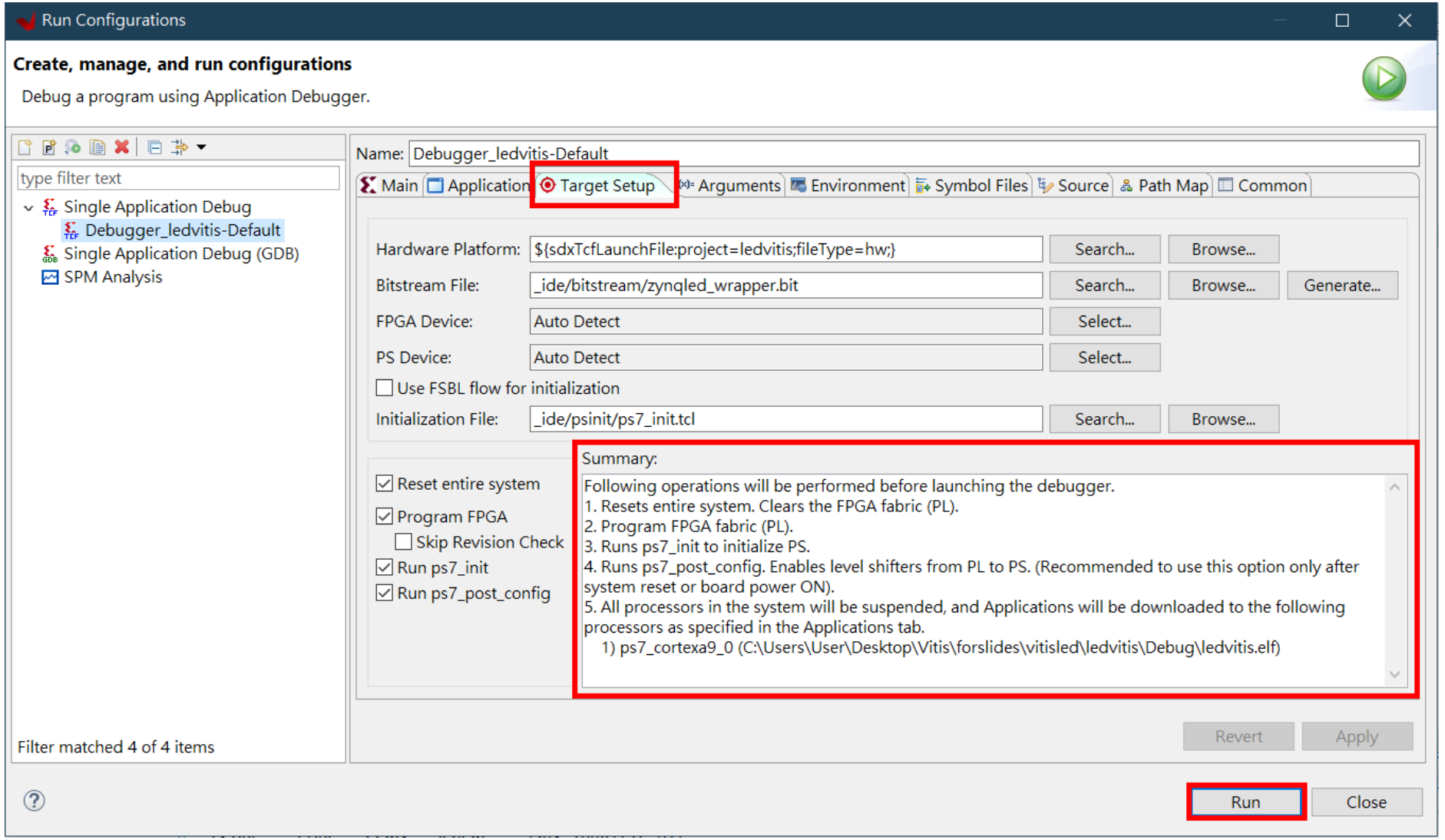
Set up run configurations



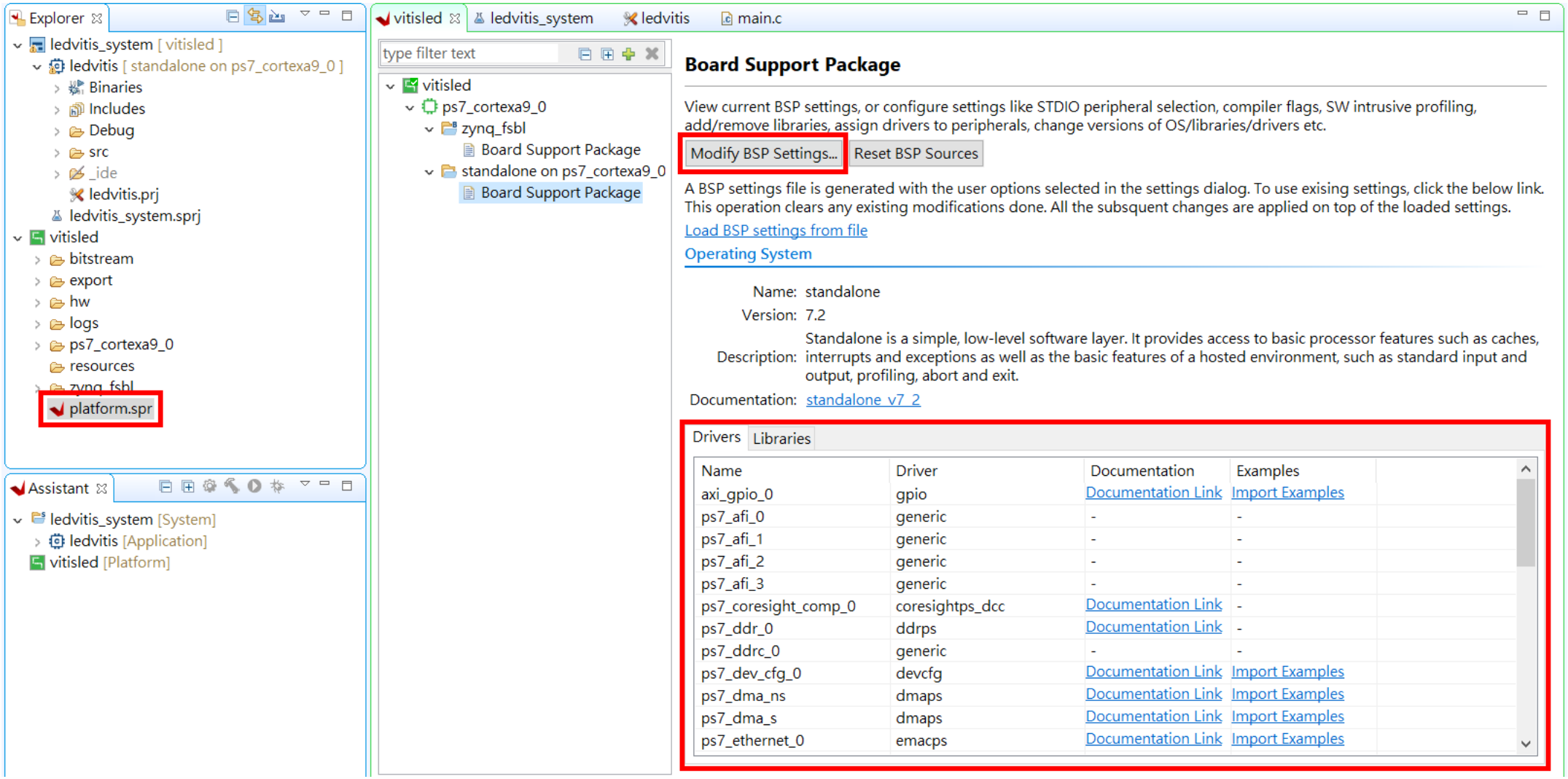
Set up run configurations



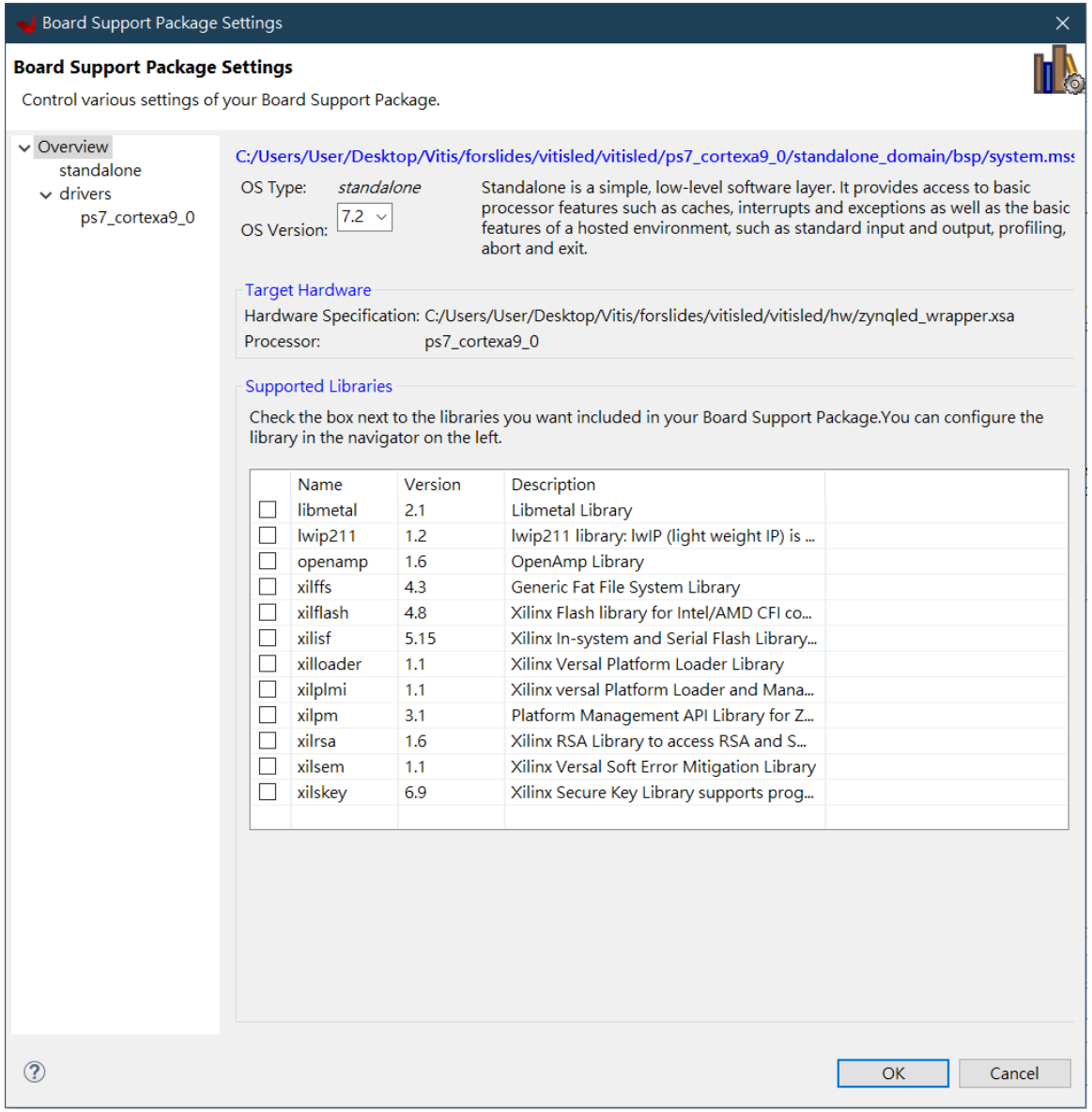
Set up run configurations



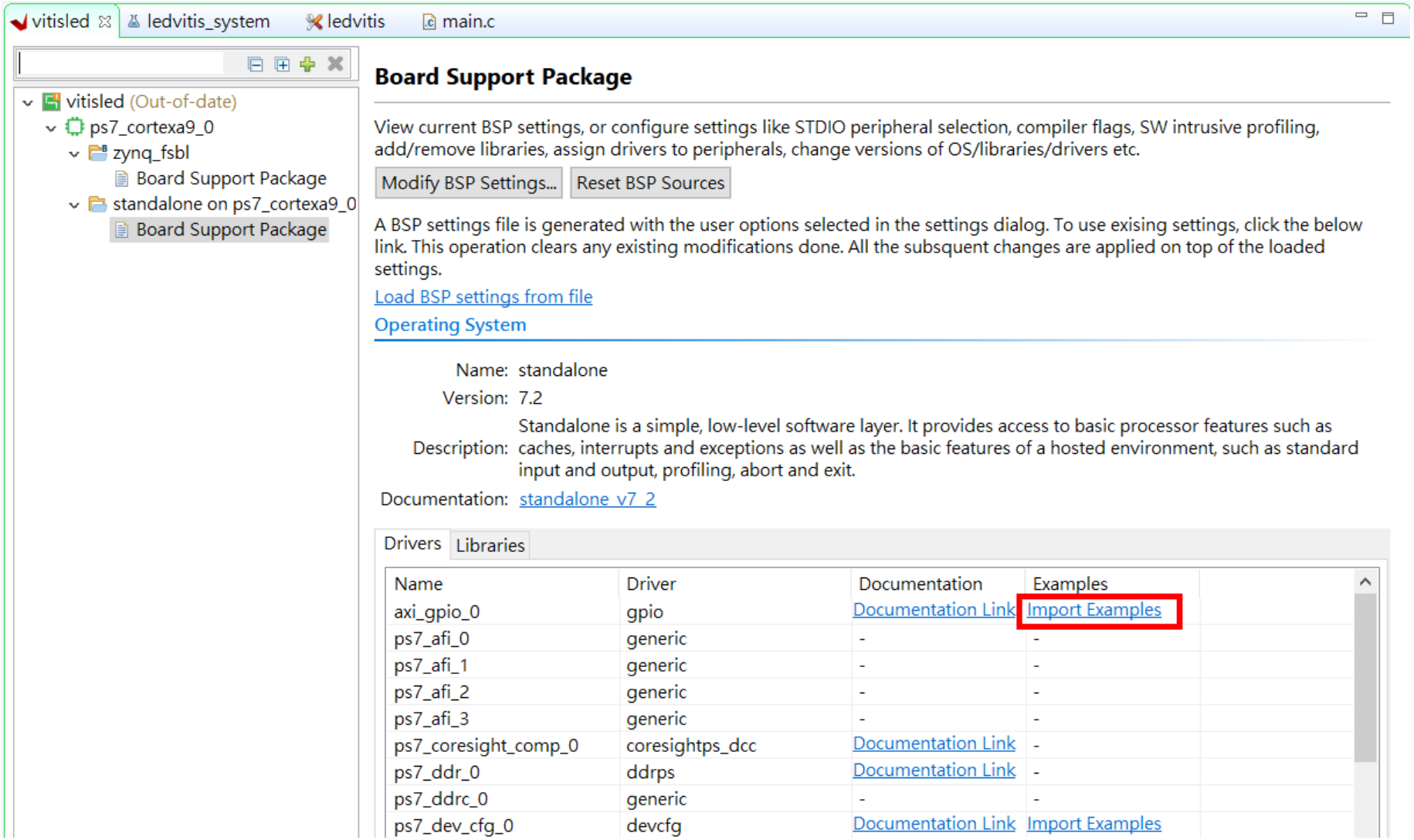
Board support package



Modify BSP settings

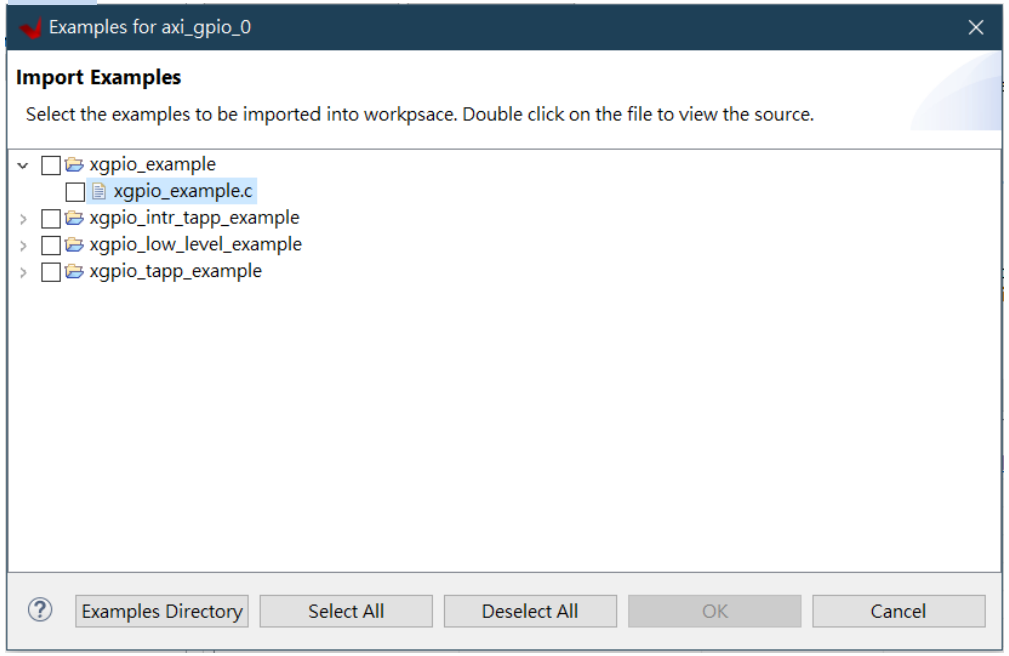


Examples

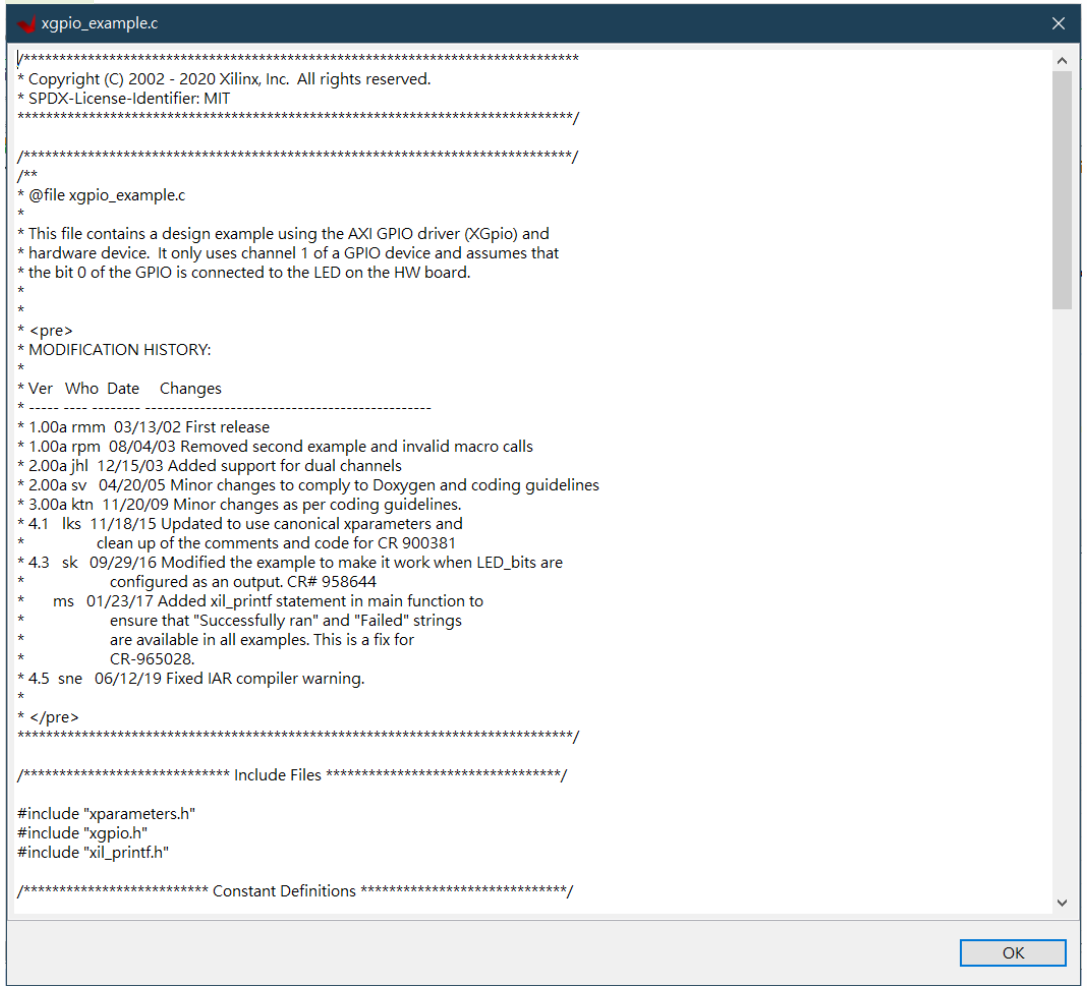


Examples

1



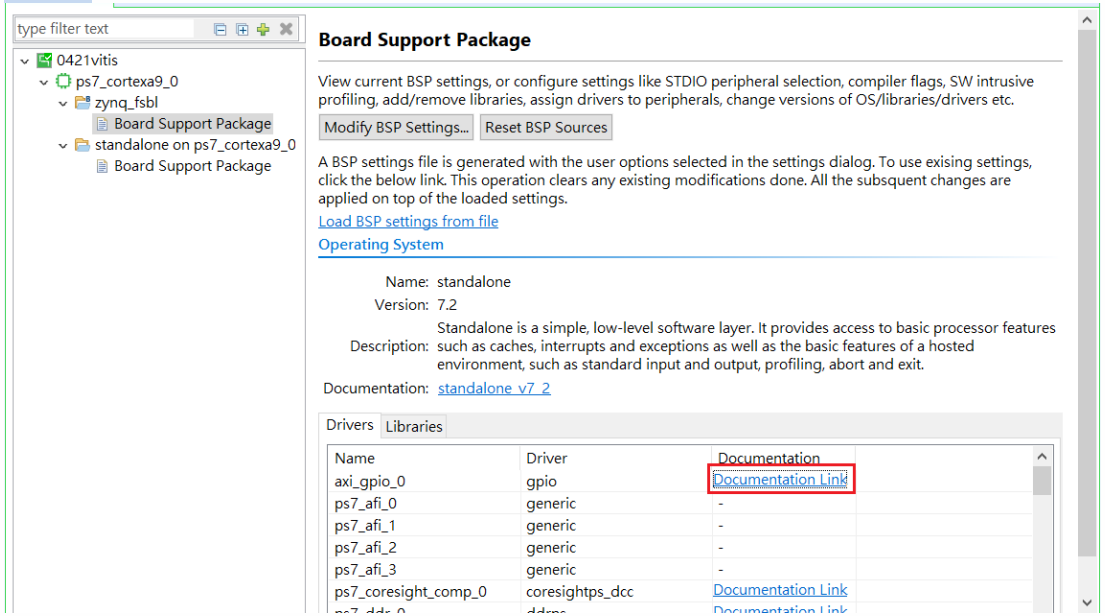
2



XILINX.

APIs

1



2

XILINX®

gpio_v4_6

Xilinx Vitis Drivers API Documentation

Overview | Data Structures | APIs | File List | Examples

- gpio_v4_6
  - Data Structures
  - APIs
    - All
    - Functions
    - Variables
    - Macros
  - File List
  - Examples

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- g -

- GpioDisableIntr() : **xgpio_intr_tapp_example.c**
- GpioHandler() : **xgpio_intr_tapp_example.c**
- GpioInputExample() : **xgpio_tapp_example.c**
- GpioIntrExample() : **xgpio_intr_tapp_example.c**
- GpioOutputExample() : **xgpio_tapp_example.c**
- GpioSetupIntrSystem() : **xgpio_intr_tapp_example.c**

- m -

- main() : **xgpio_example.c** , **xgpio_tapp_example.c** , **xgpio_low_level_example.c** , **xgpio_intr_tapp_example.c**

- x -

- XGpio_CfgInitialize() : **xgpio.c**
- XGpio_ConfigTable : **xgpio_g.c** , **xgpio_i.h**
- XGPIO_DATA2_OFFSET : **xgpio_l.h**
- XGPIO_DATA_OFFSET : **xgpio_l.h**

XILINX.

Platform hardware specification

vitisled | ledvitis_system | ledvitis | main.c

Hardware Platform Specification

Design Information

Target FPGA Device: 7z020
Part: xc7z020clg484-1
Created With: Vivado 2020.1
Created On: Fri Nov 12 14:55:01 2021

Note: To view ip parameters, double-click on the cell containing ip name in any of the below tables.

Address Map for processor ps7_cortexa9[0-1]

Filter: Search: 35 Loaded - 35 Shown - 0 Selected -

Cell	Base Address	High Address	Slave Interface	Addr Range Type
ps7_intc_dist_0	0xf8f01000	0xf8f01fff	-	register
ps7_gpio_0	0xe000a000	0xe000afff	-	register
ps7_scutimer_0	0xf8f00600	0xf8f0061f	-	register
ps7_slcr_0	0xf8000000	0xf8000fff	-	register
axi_gpio_0	0x41200000	0x4120ffff	S_AXI	register
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	-	register
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	-	register
ps7_scuc_0	0xf8f00000	0xf8f000fc	-	register

IP blocks present in the design

Filter: Search: 43 Loaded - 43 Shown - 0 Selected -

IP Instance	IP Type	IP Version	Register
ps7_0_axi_periph	axi_interconnect	2.1	-
ps7_intc_dist_0	ps7_intc_dist	1.00.a	-
rst_ps7_0_100M	proc_sys_reset	5.0	-
ps7_gpio_0	ps7_gpio	1.00.a	-
ps7_scutimer_0	ps7_scutimer	1.00.a	-

Address Vitis versus Vivado

Address Map for processor ps7_cortexa9[0-1]

Filter: Search: 35 Loaded - 35 Shown - 0 Selected -

Cell	Base Address	High Address	Slave Interface	Addr Range Type
ps7_intc_dist_0	0xf8f01000	0xf8f01fff	-	register
ps7_gpio_0	0xe000a000	0xe000afff	-	register
ps7_scutimer_0	0xf8f00600	0xf8f0061f	-	register
ps7_slcr_0	0xf8000000	0xf8000fff	-	register
axi_gpio_0	0x41200000	0x4120ffff	S_AXI	register
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	-	register
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	-	register
ps7_scuc_0	0xf8f00000	0xf8f000fc	-	register

Diagram | Address Editor

Assigned (1) Unassigned (0) Excluded (0) Hide All

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF

Results