



Vitis Software Development Environment

2021.2

Objectives

- ▶ **After completing this module, you will be able to:**
 - Understand the basic concepts of the Eclipse IDE in Vitis
 - List Vitis IDE features
 - Describe BSP and device drivers' architecture
 - Distinguish between Level-1 and Level-2 device drivers
 - List steps in creating a software application

Outline

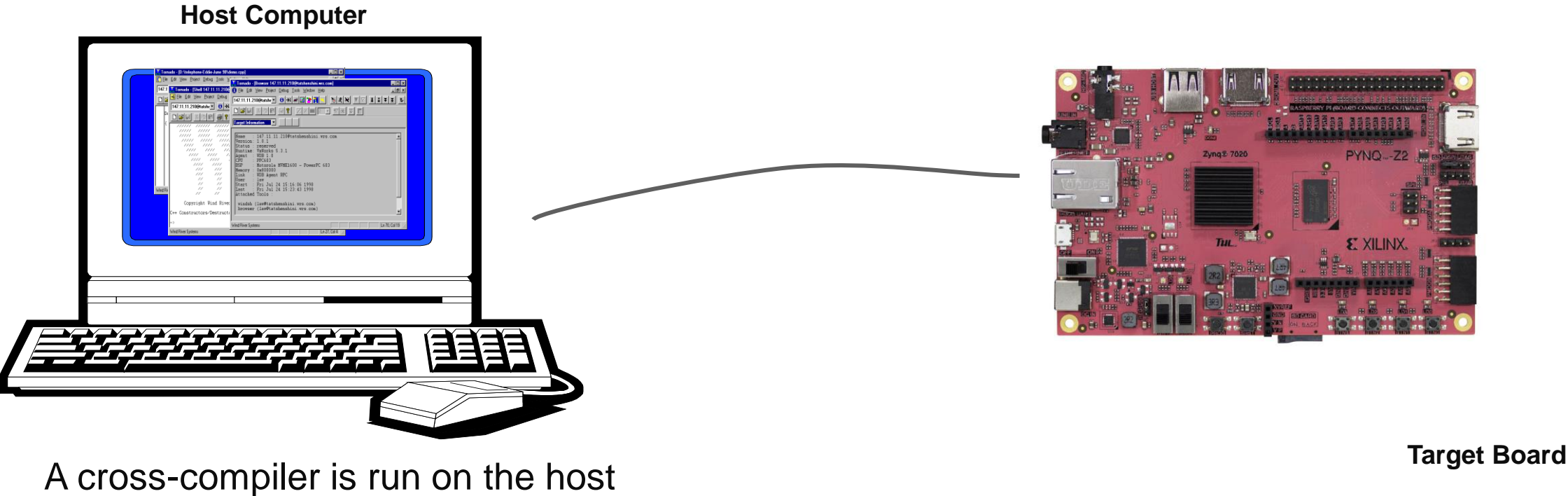
- ▶ **Introduction**
- ▶ **Vitis IDE**
- ▶ **Vitis Project Creation**
- ▶ **Domain and Board Support Package**
- ▶ **Summary**

Desktop versus Embedded

- ▶ Desktop development: written, debugged, and run on the same machine
- ▶ OS loads the program into the memory when the program has been requested to run
- ▶ Address resolution takes place at the time of loading by a program called the loader
 - The loader is included in the OS
- ▶ The programmer glues into one executable file called ELF
 - Boot code, application code, RTOS, and ISRs
 - Address resolution takes place during the *gluing* stage
- ▶ The executable file is downloaded into the target system through different methods
 - Ethernet, serial, JTAG, BDM, ROM programmer

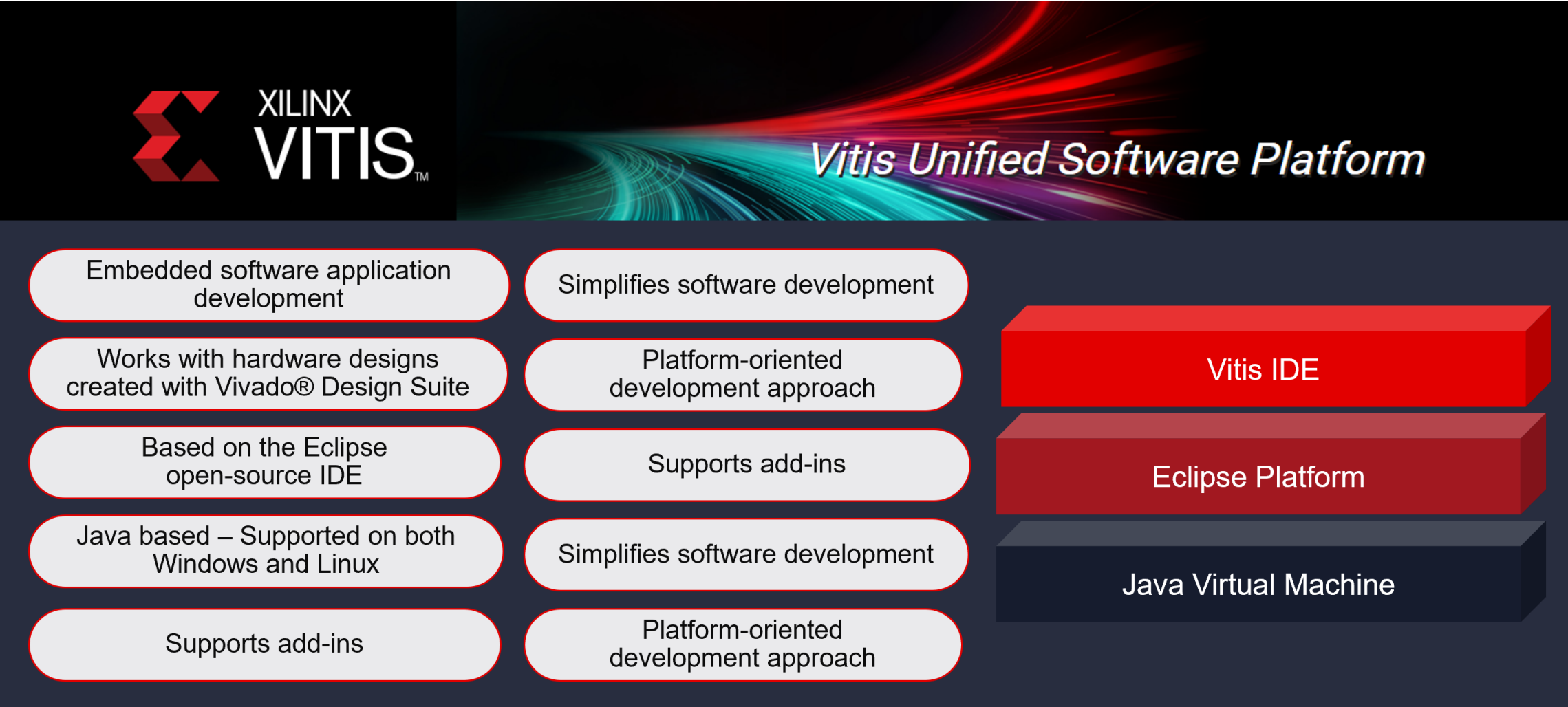
Embedded versus Desktop

- ▶ Development takes place on one machine (host) and is downloaded to the embedded system (target)



Vitis IDE

Vitis IDE



Vitis Software Development

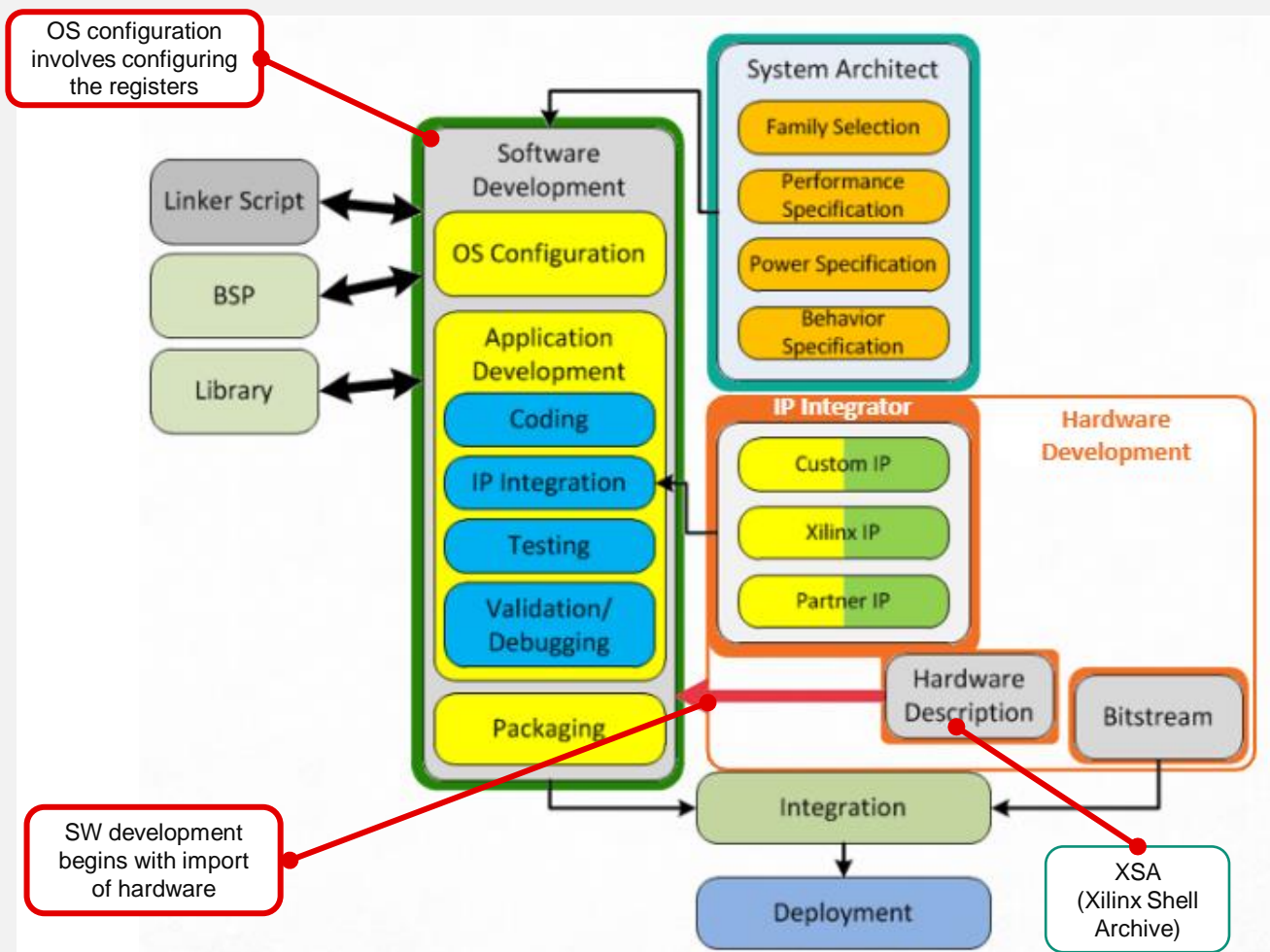
Vitis IDE

Predefined hardware platforms for Xilinx boards

Enables software development before hardware completion

Hardware description import helps in:

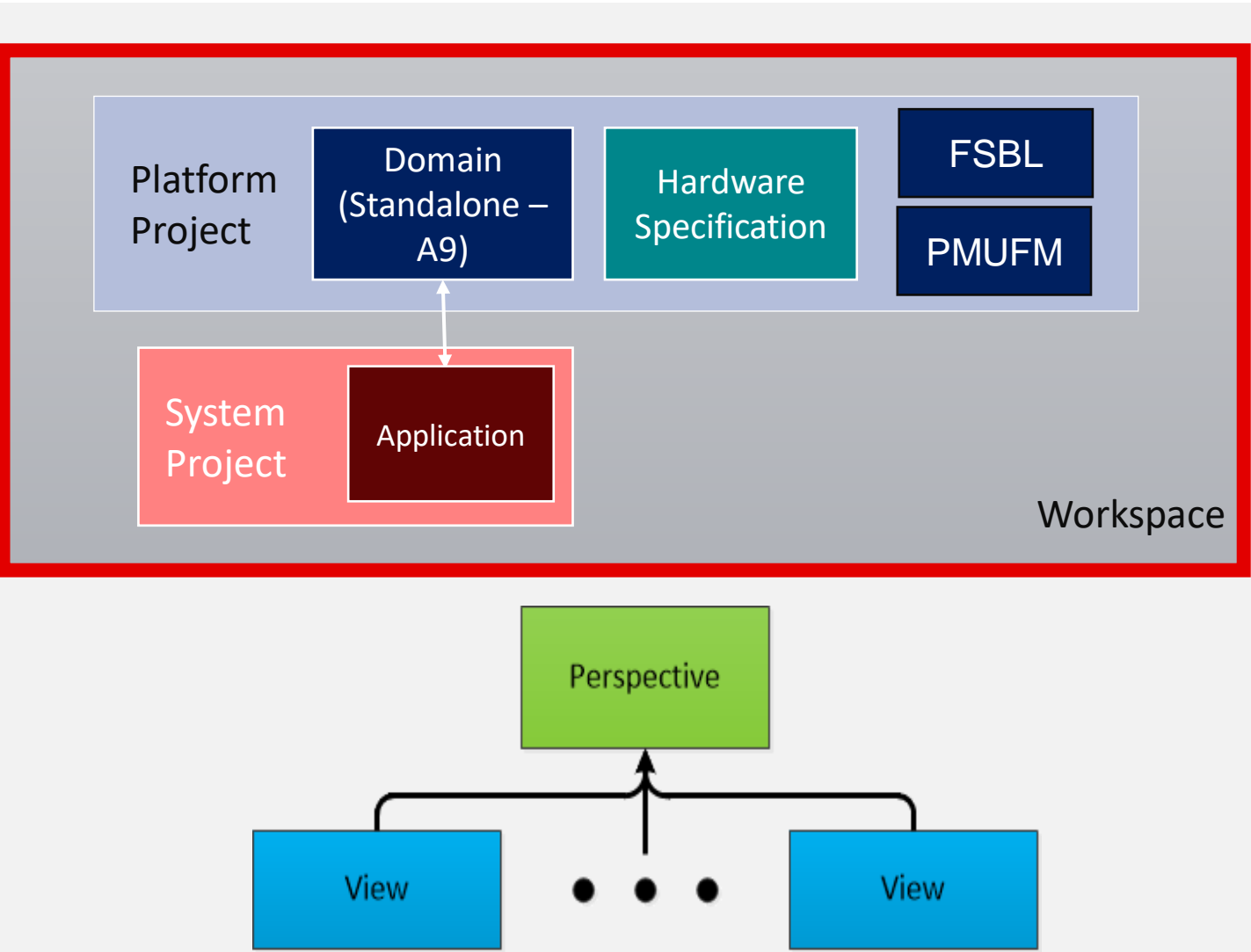
- OS configuration
- Building platform project
- Boot file generation



Workspaces and Perspectives

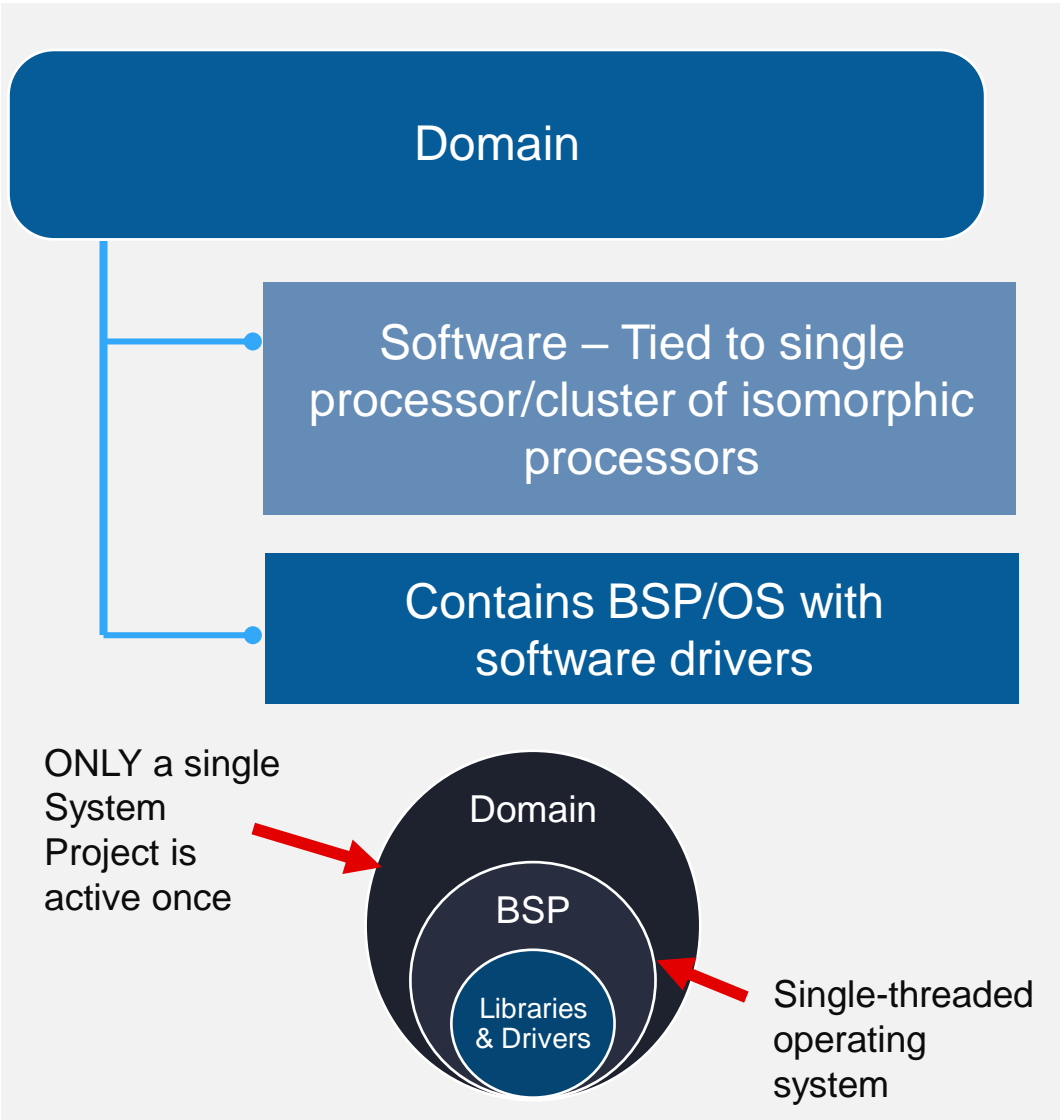
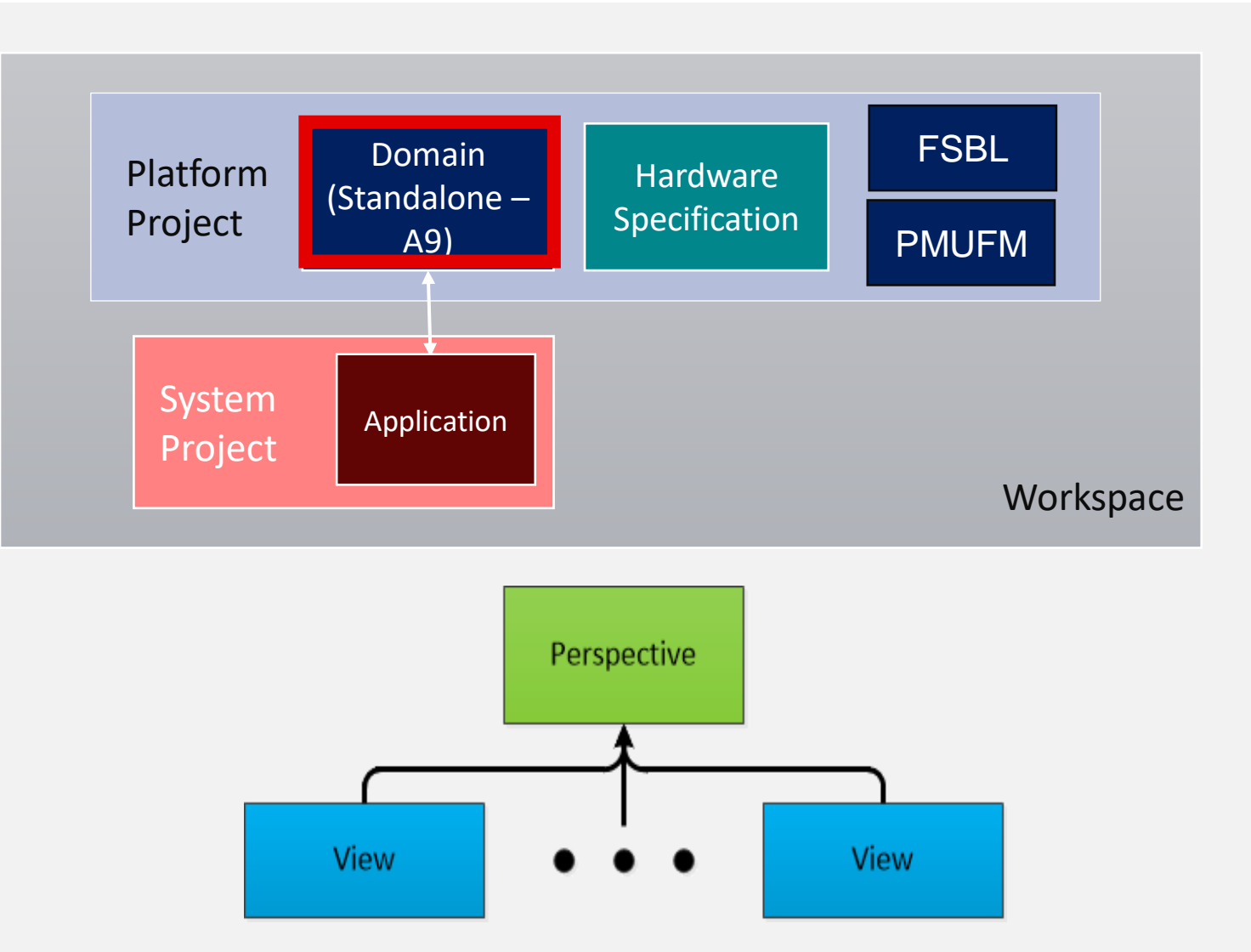
- ▶ Workspace
 - Location to store preferences & internal info about projects
 - Transparent to users
 - Source files not stored under Workspace
- ▶ Views, Editors
 - Basic user interface element
- ▶ Perspectives
 - Collection of functionally related views
 - Layout of views in a perspective can be customized according to user preference

Workspaces, Projects, and Perspectives

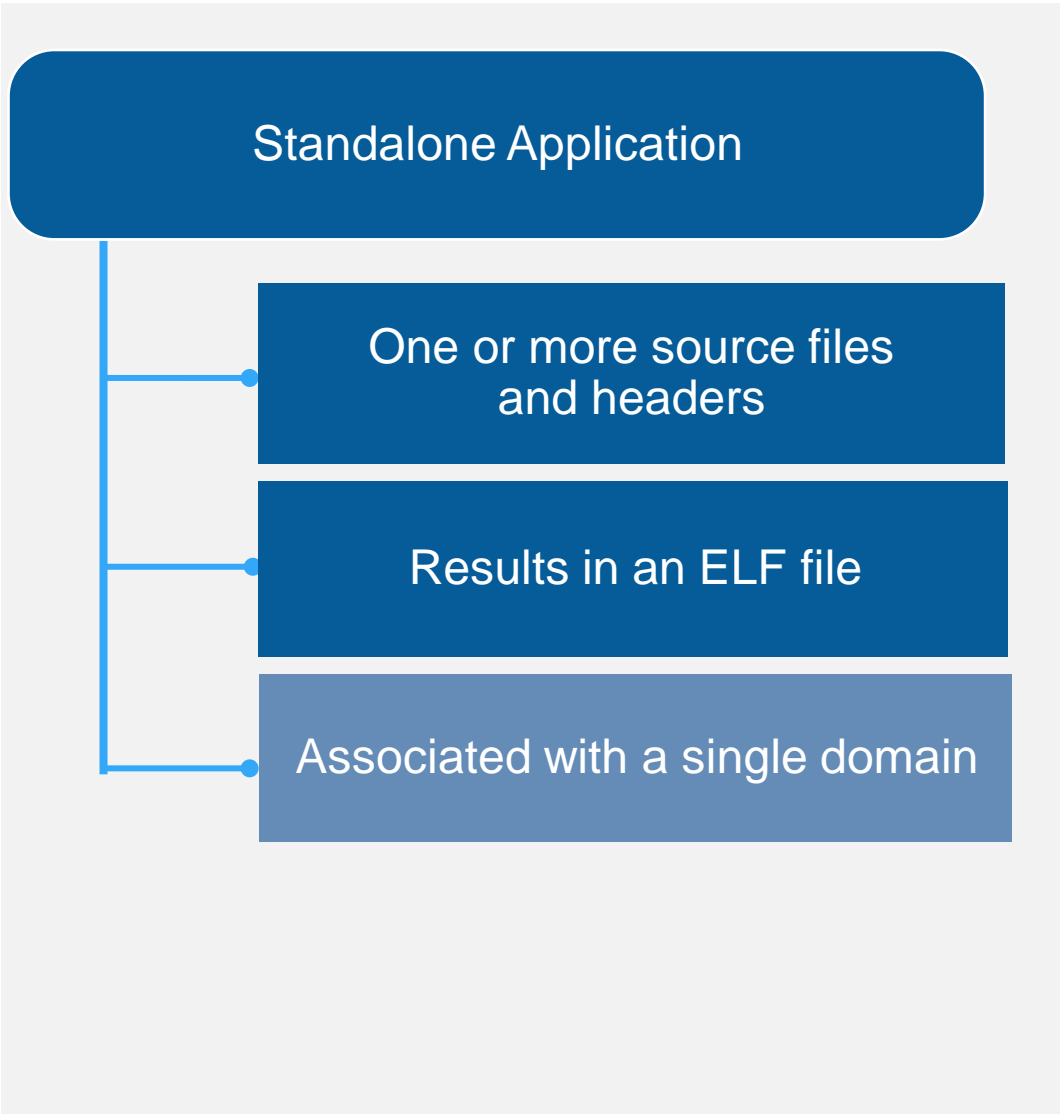
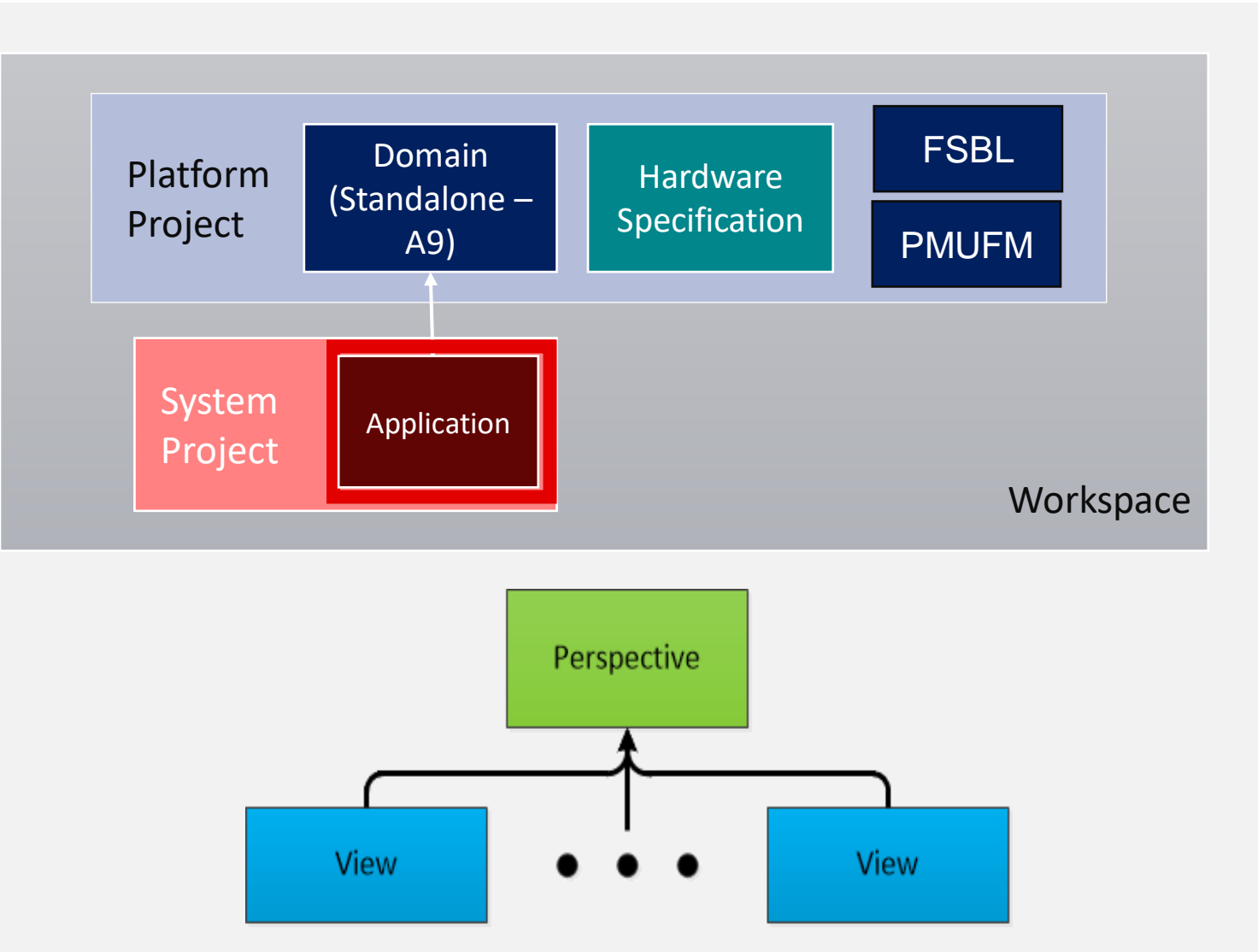


- Physical Location – Stores Vitis IDE files
- Enables navigation among projects
- Each directory contains both user and tool-generated files
- Support independent multiple workspaces

Workspaces, Projects, and Perspectives



Workspaces, Projects, and Perspectives



Understanding the Vitis Workspace

Platform Project

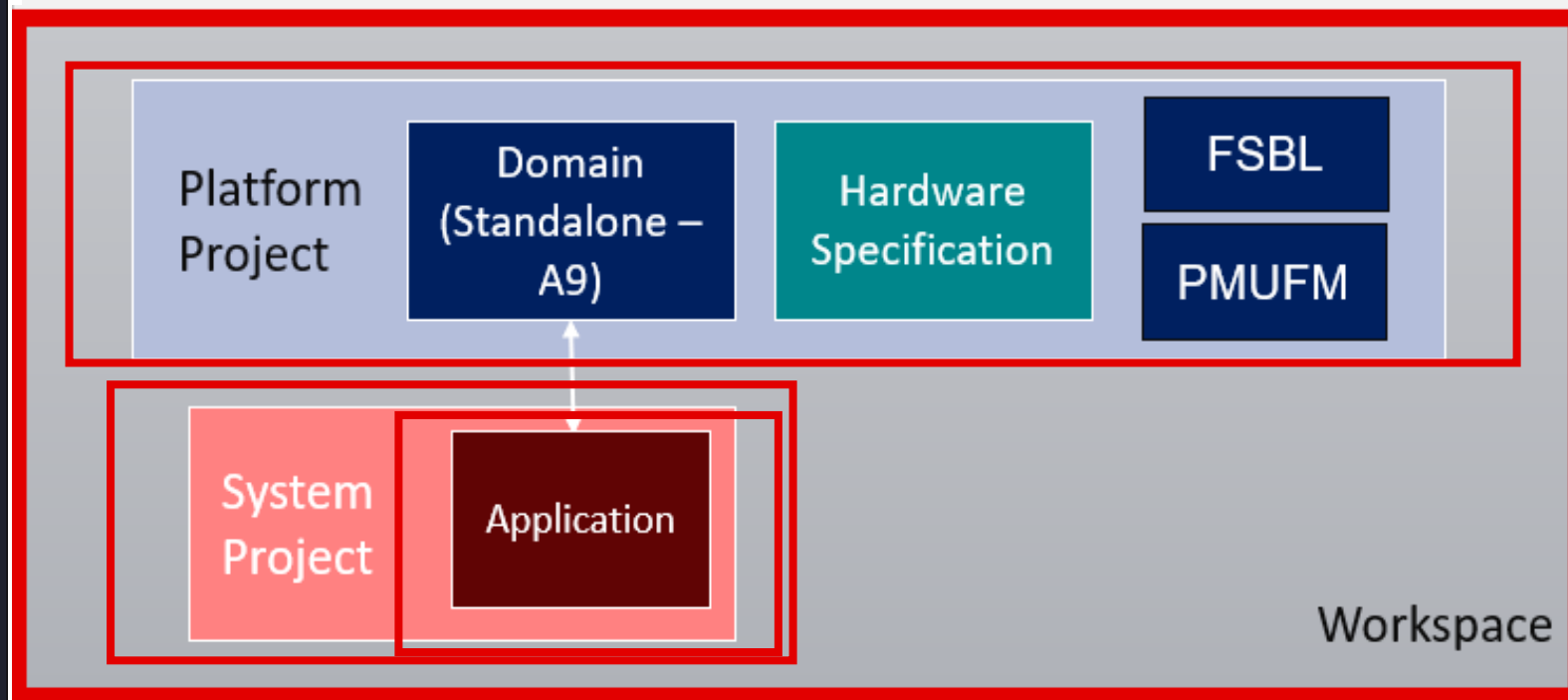
- Combination of hardware description and software (domain) components
- Customizable for adding/modifying domains
- One or more domains can be on the same hardware specification
- BSPs contain software support for the hardware underneath

Application Project

- Created based on the domain
- Contains source files and header files

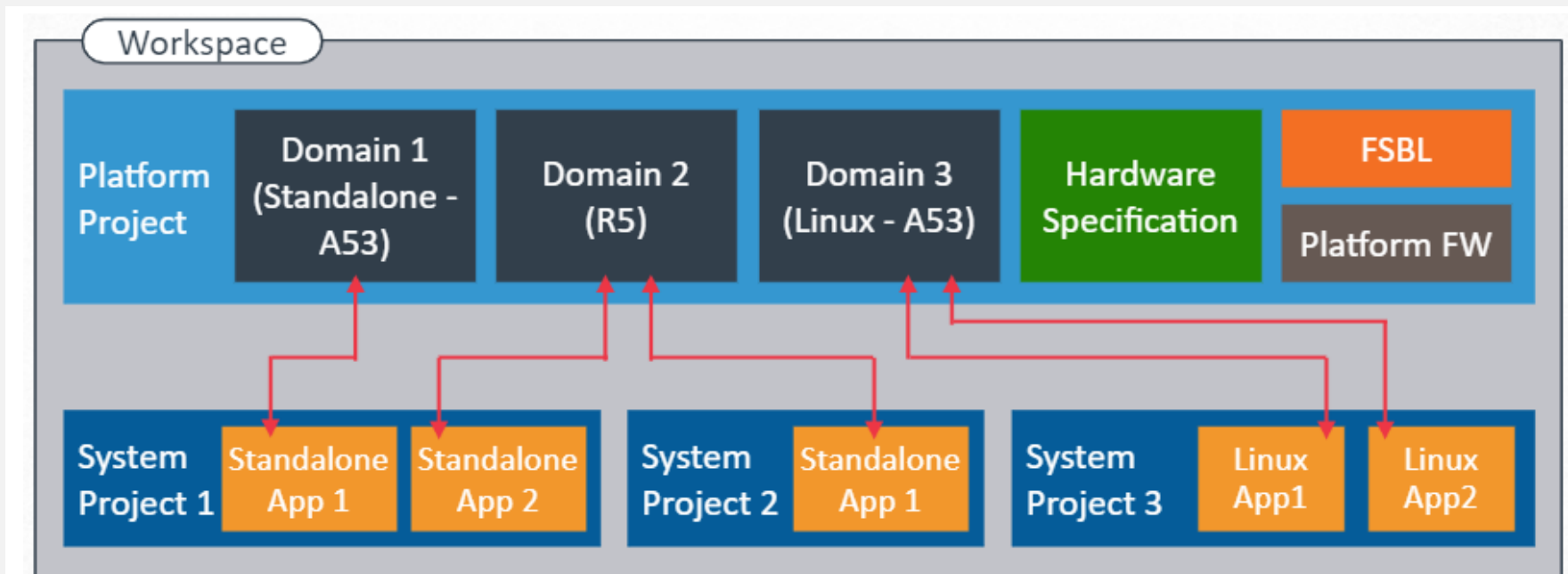
System Project

- Groups applications that run simultaneously on the device



Working with Projects

Application projects are the final application containers



Application project directory contains:



C/C++ source files



Executable output file



Associated utility files

Note:

- Two same-processor standalone apps cannot be in a system project
- Two Linux apps can be in a system project

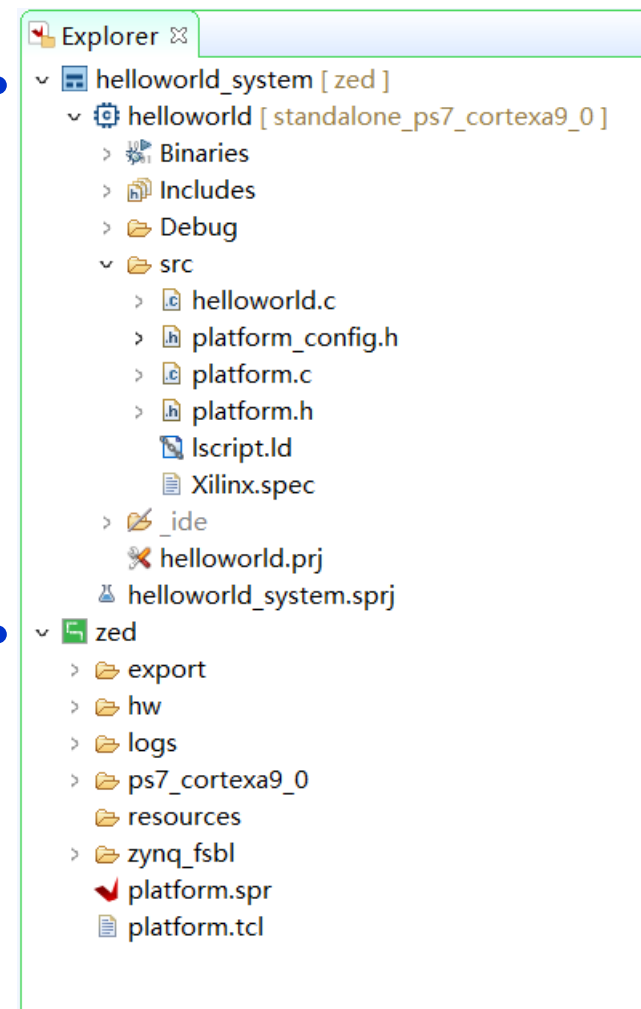
C/C++ Project View

- ▶ Hierarchical list of the workspace projects in a hierarchical format
- ▶ Double-click to open a file
- ▶ Right-click the project to access its properties

Application
Project

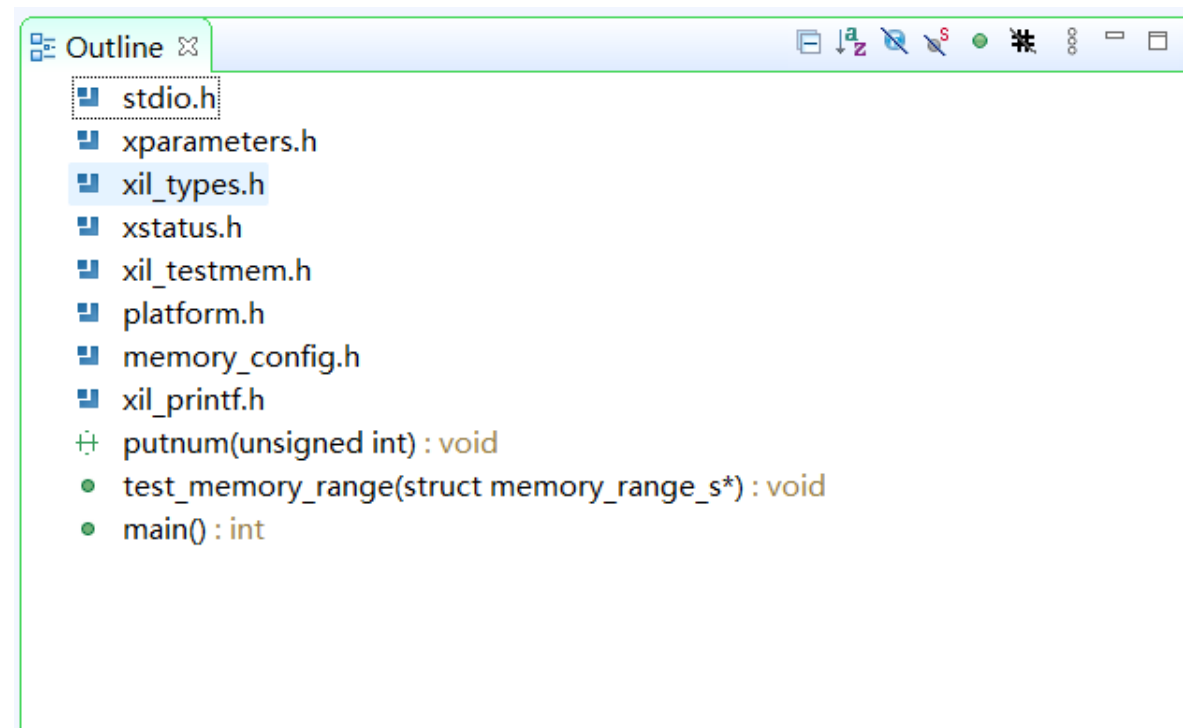
Platform
Project

Hardware
Description



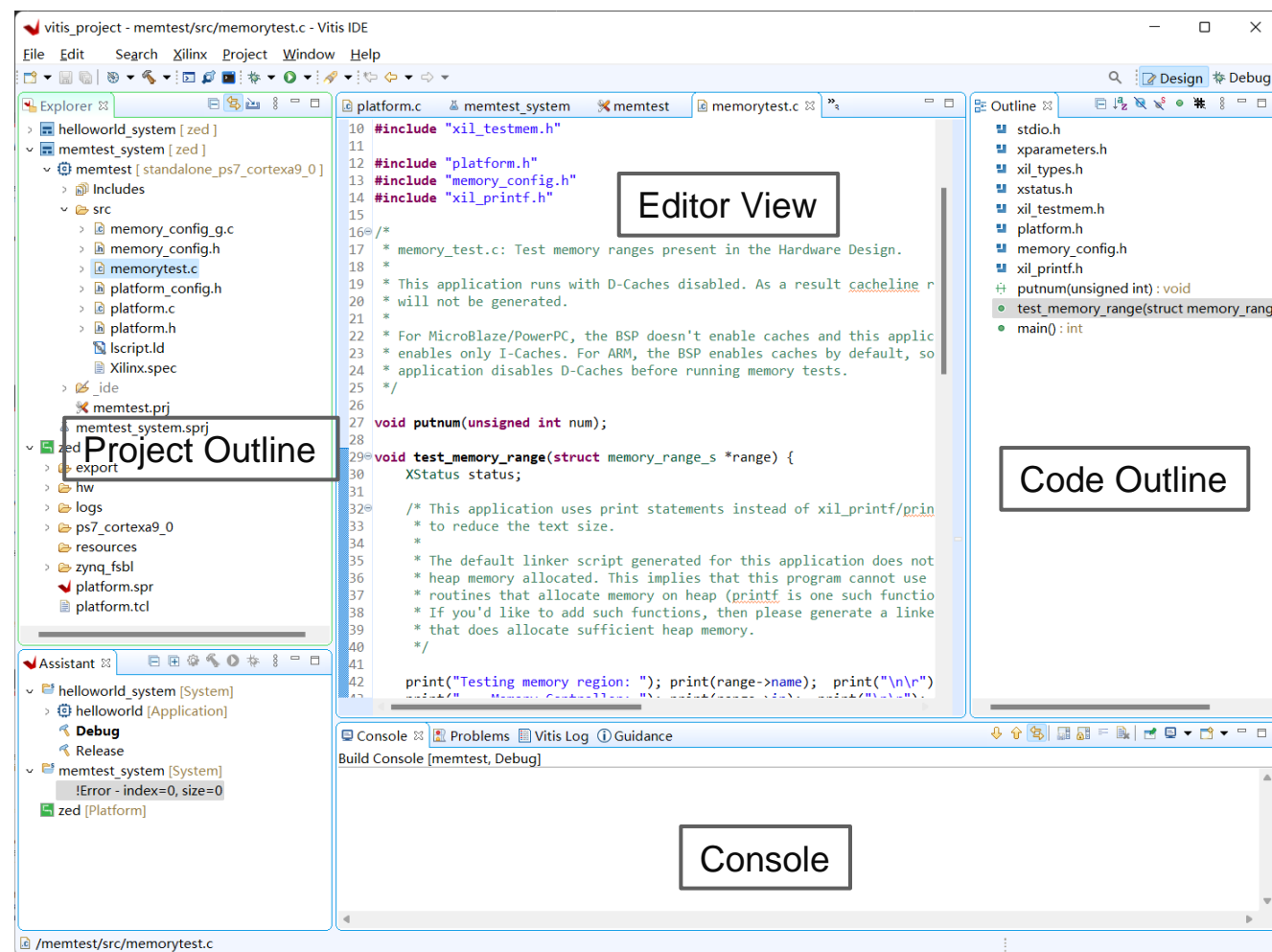
Outline View

- ▶ Displays an outline of the structured file that is currently open in the editor
- ▶ The contents of the outline view are editor specific
- ▶ Content type is indicated by the icon
- ▶ For a C source, icons represent
 - *#define* statements
 - Include files
 - Function calls
 - Declarations
- ▶ Selecting a symbol will navigate to the same in the editor window



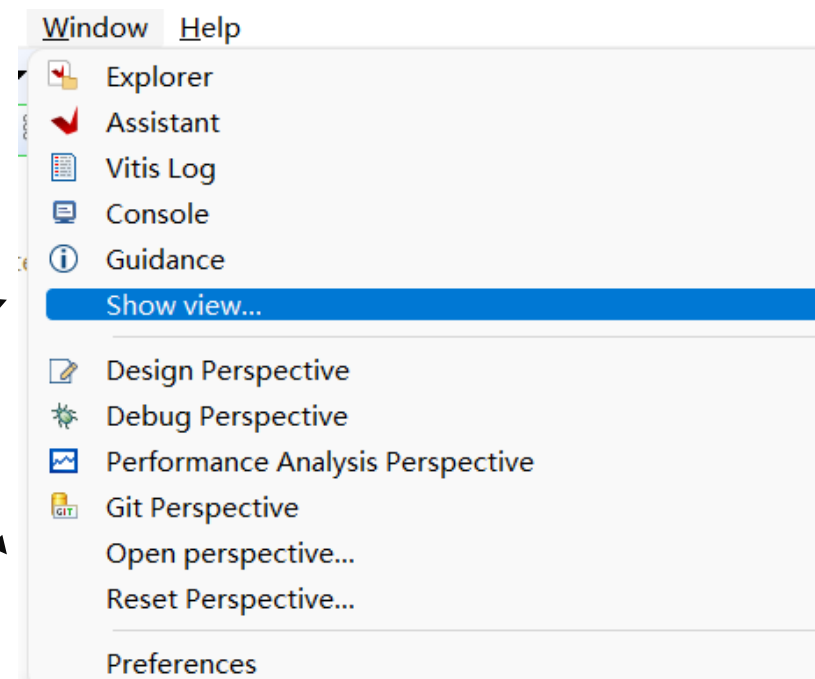
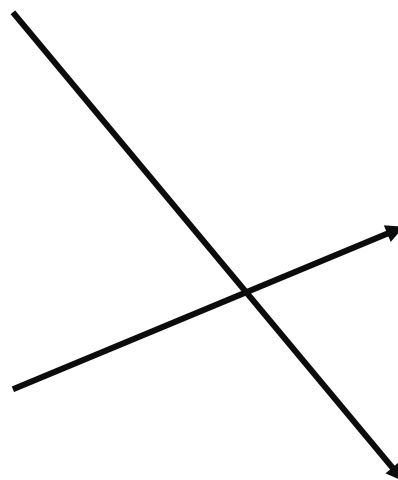
C/C++ Perspective

- ▶ C/C++ project outline displays the elements of a project with file decorators (icons) for easy identification
- ▶ C/C++ editor for integrated software creation
- ▶ Code outline displays elements of the software file under development with file decorators (icons) for easy identification
- ▶ Problems, Console, Properties view lists output information associated with the software development flow



Opening Perspectives and Views

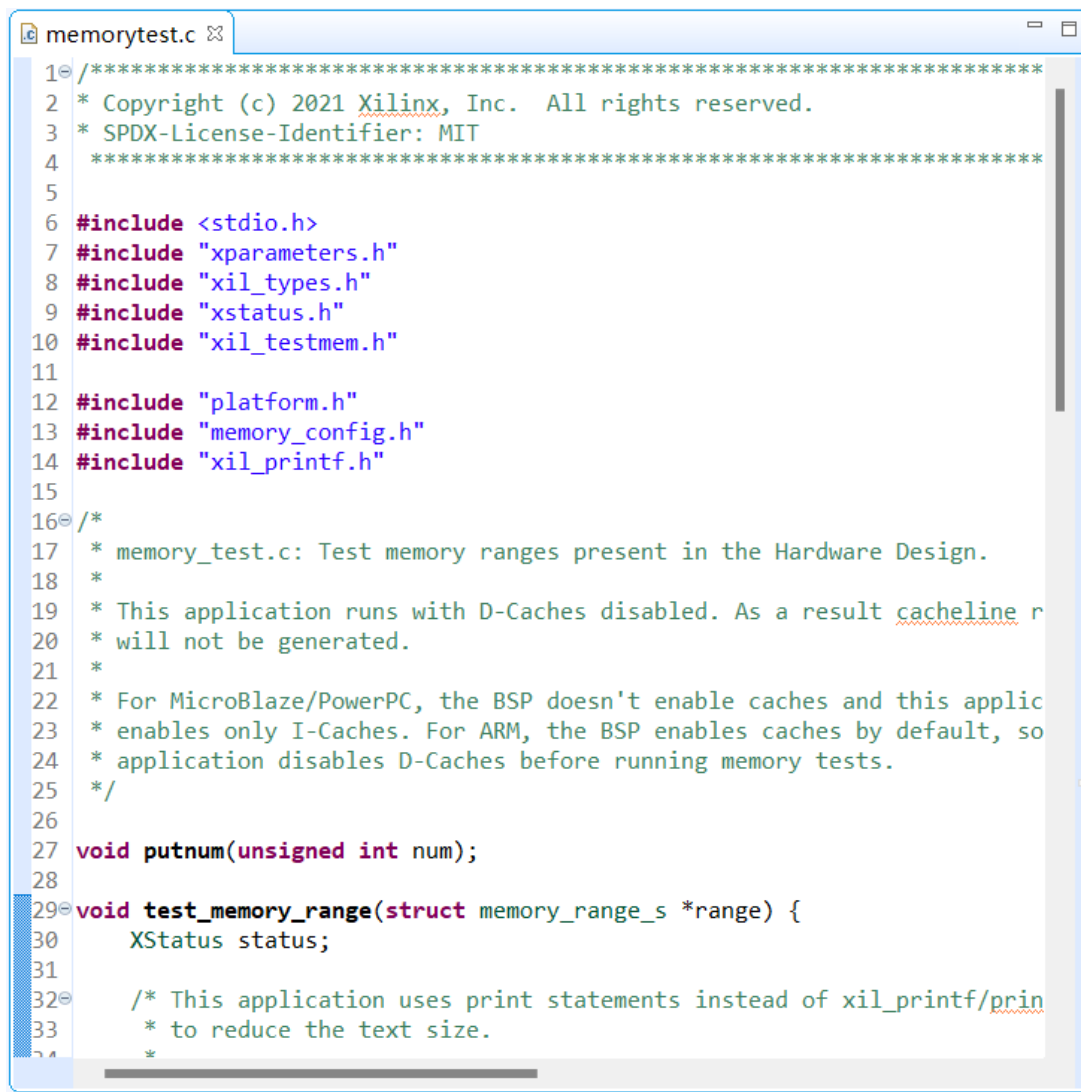
- ▶ To open a Perspective, use
 - Window → Open Perspective
- ▶ To open a view, use
 - Window → Show View
 - If the view is already present in the current perspective, the view is highlighted



Editor

► Syntax Highlighting

- bracket matching
- syntax coloring
- content assist
- refactoring
- keyboard shortcuts



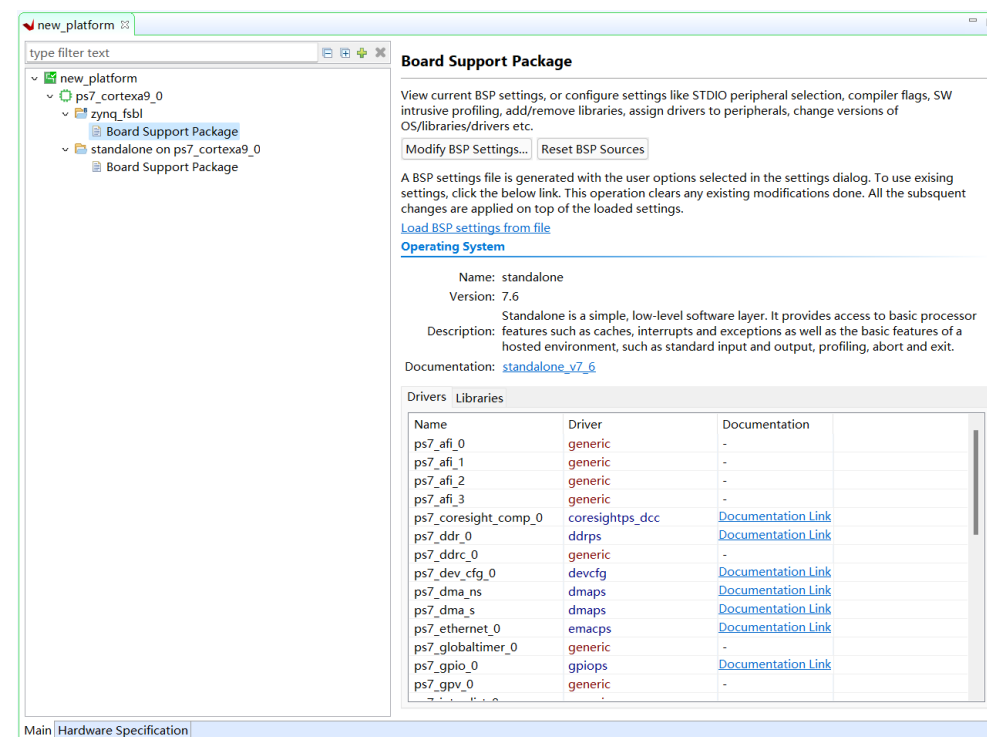
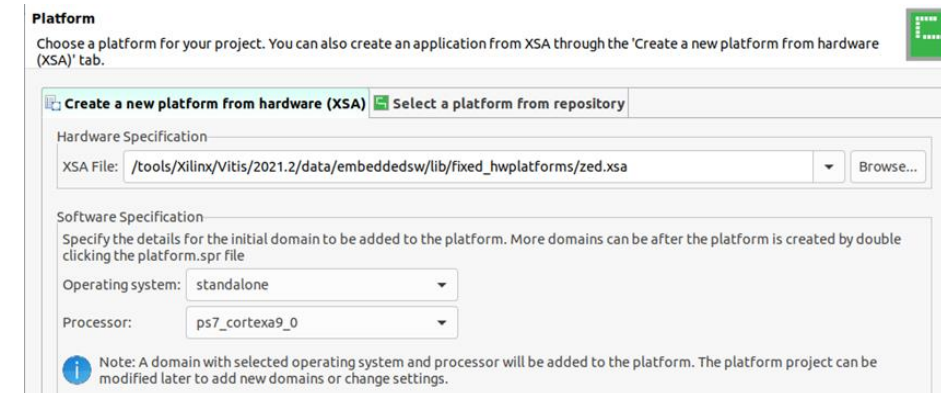
```

1  /*****
2  * Copyright (c) 2021 Xilinx, Inc. All rights reserved.
3  * SPDX-License-Identifier: MIT
4  *****/
5
6  #include <stdio.h>
7  #include "xparameters.h"
8  #include "xil_types.h"
9  #include "xstatus.h"
10 #include "xil_testmem.h"
11
12 #include "platform.h"
13 #include "memory_config.h"
14 #include "xil_printf.h"
15
16 /*
17  * memory_test.c: Test memory ranges present in the Hardware Design.
18  *
19  * This application runs with D-Caches disabled. As a result cacheline r
20  * will not be generated.
21  *
22  * For MicroBlaze/PowerPC, the BSP doesn't enable caches and this applic
23  * enables only I-Caches. For ARM, the BSP enables caches by default, so
24  * application disables D-Caches before running memory tests.
25  */
26
27 void putnum(unsigned int num);
28
29 void test_memory_range(struct memory_range_s *range) {
30     XStatus status;
31
32     /* This application uses print statements instead of xil_printf/prin
33     * to reduce the text size.
34     */
  
```

Vitis Project Creation

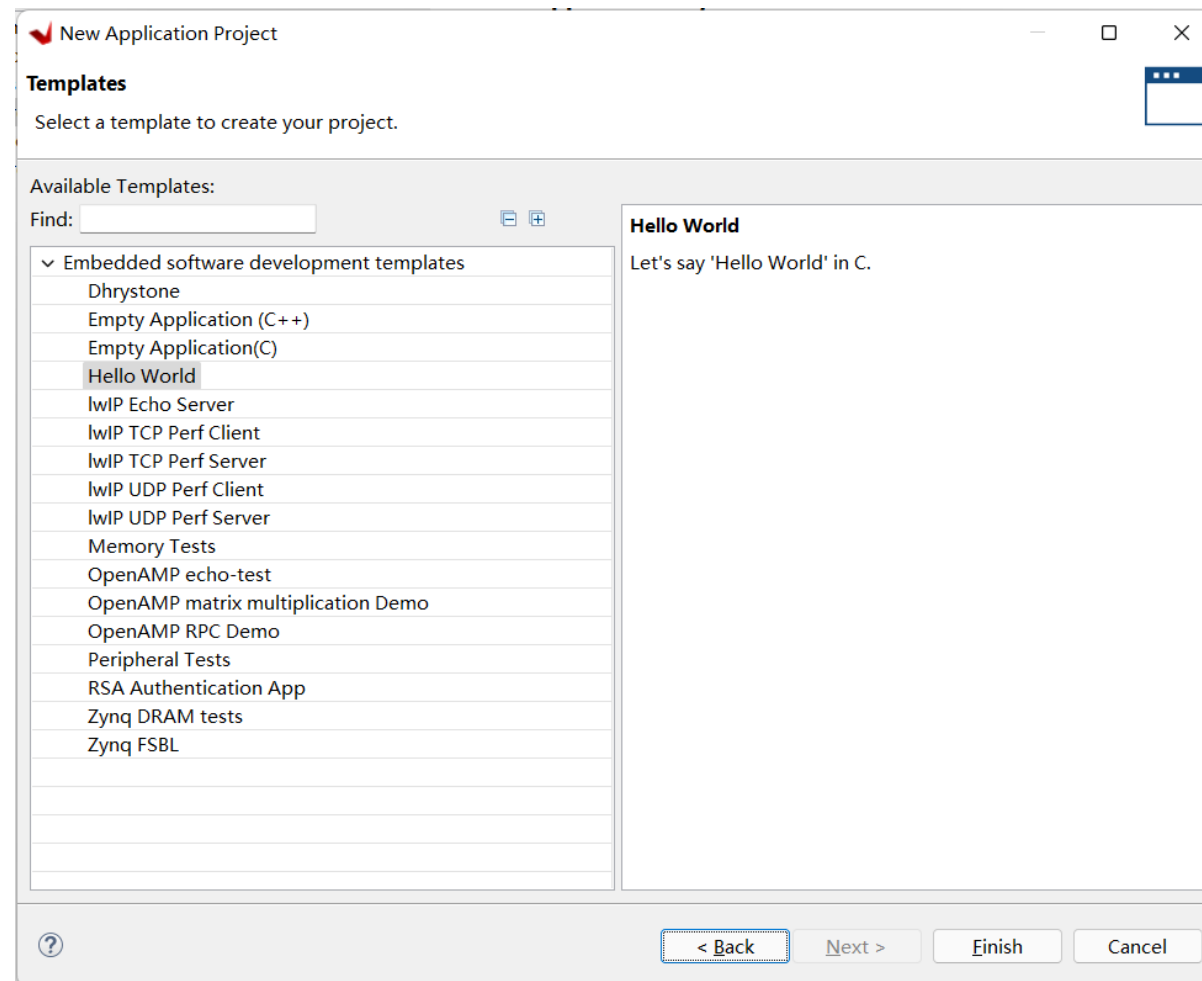
Creating a Platform Project

- ▶ A platform or platform project is comprised of a description of the hardware element and software to boot and maintain the platform
- ▶ Must be attached to a Hardware Description
 - File > New > Platform Project
 - Enter the name and select a hardware description (XSA)
 - Select domain (standalone or linux)
 - Select required libraries support



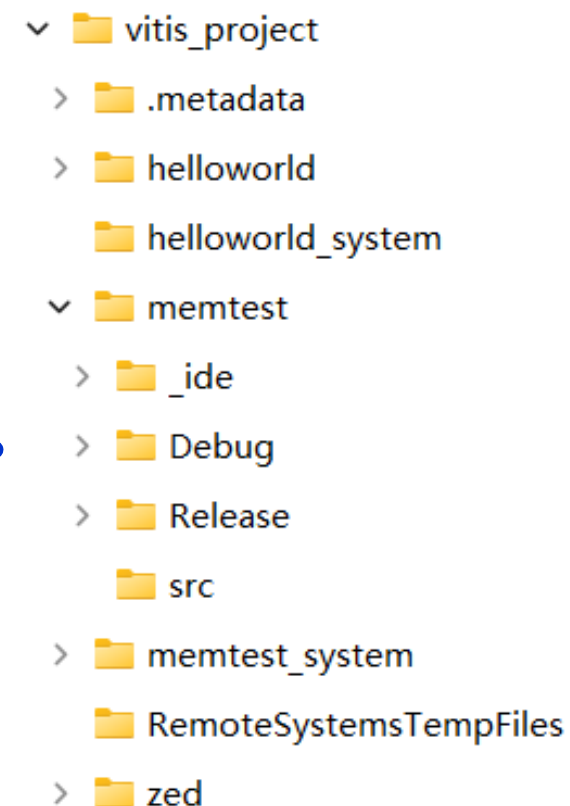
Creating a Software Application Project

- ▶ Vitis IDE supports multiple software application projects
- ▶ An application project is attached to a platform project
- ▶ Sample applications are provided
 - Great for quick test of hardware
 - Peripheral Tests
 - Starting point to base your own application on
- ▶ Typically, an Empty Application is opened to begin a non-standard project



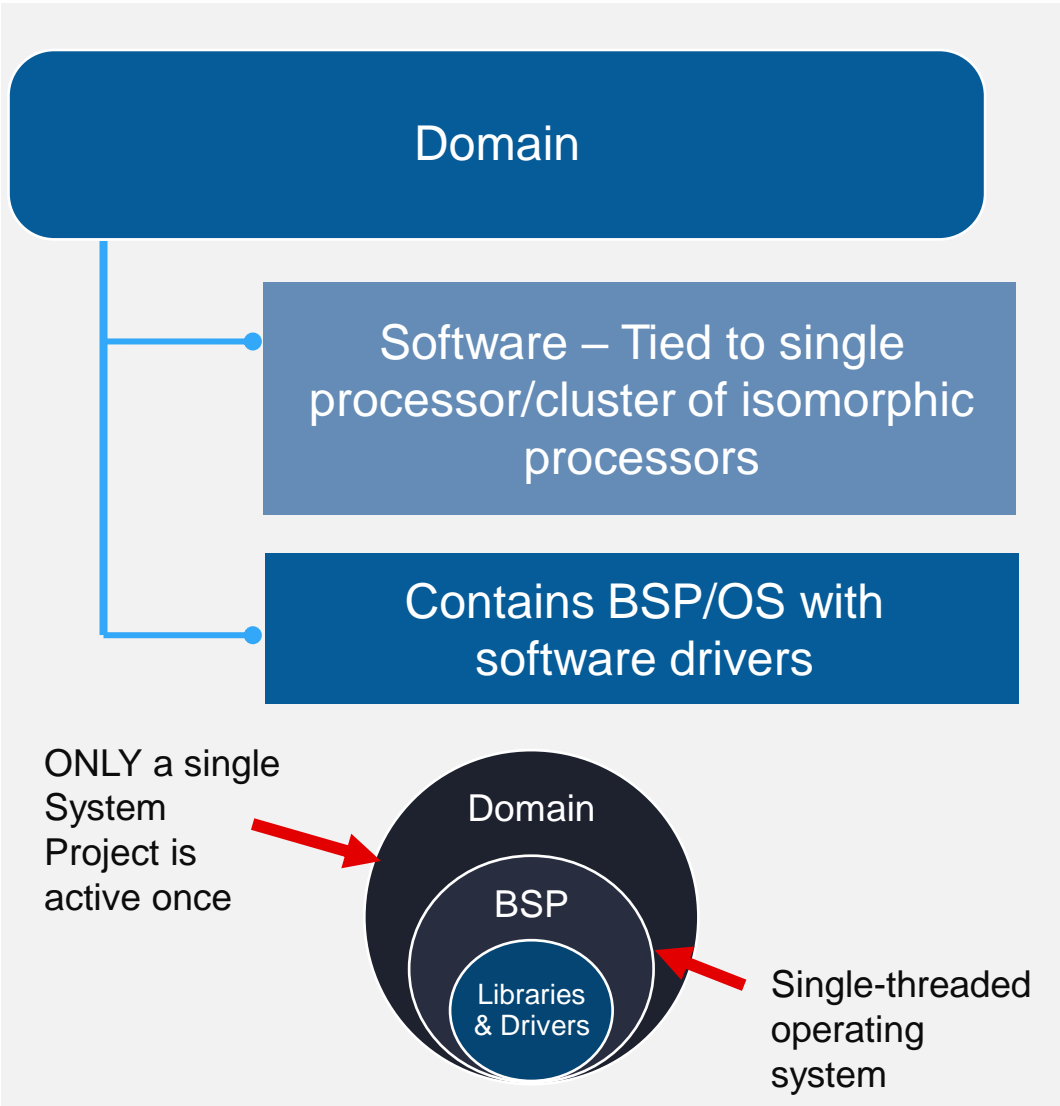
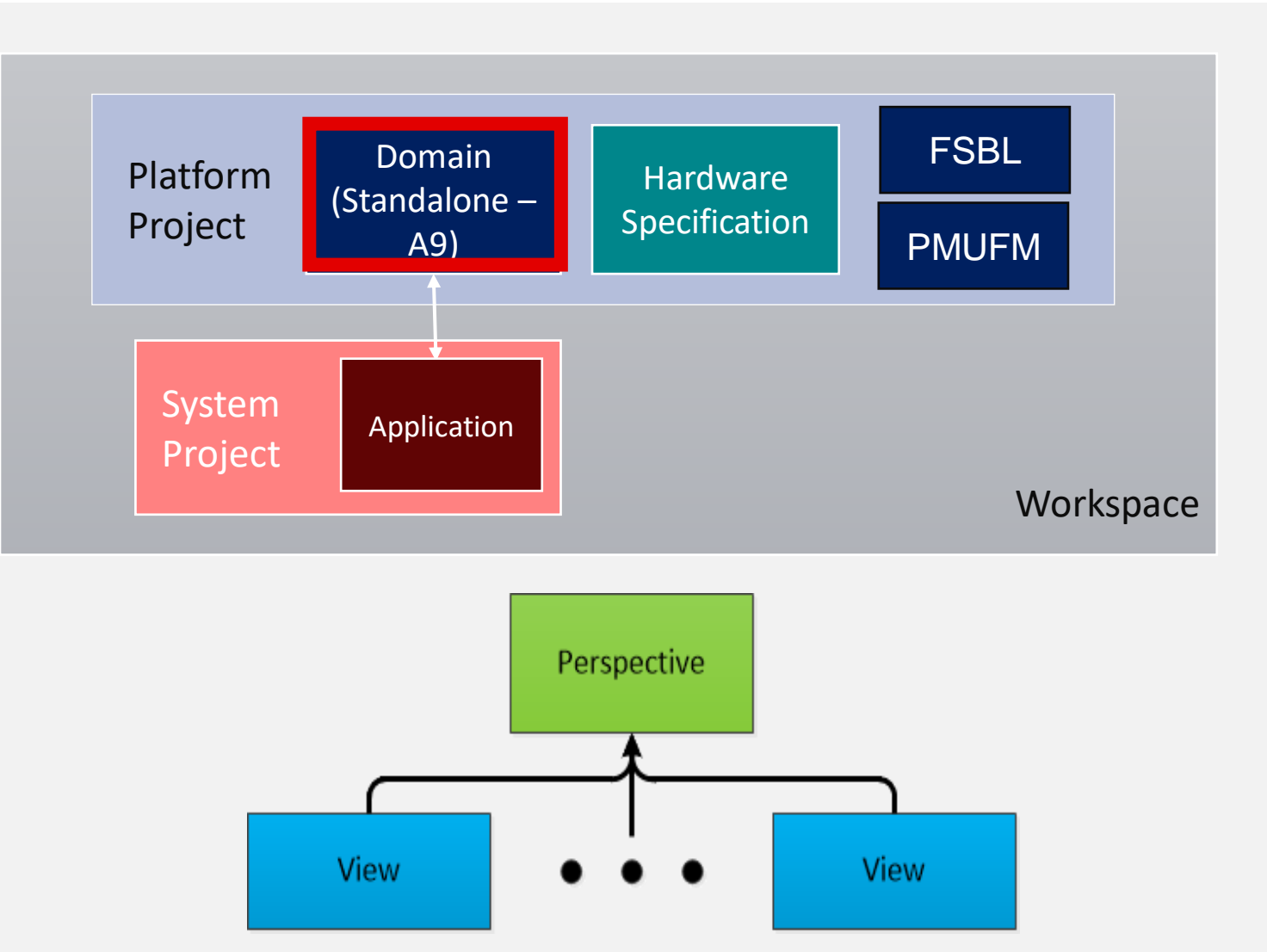
Directory Structure

- ▶ Vitis projects are placed in the workspace directory that was specified when Vitis IDE was launched
- ▶ Each project may have multiple directories for system files and configurations
- ▶ Configurations are property tool option permutations of the software application. Each configuration has project properties set depending on needs. An ELF file is generated for each
 - Release configuration
 - Debug configuration
 - Profile configuration
- ▶ A Debug configuration is created by default



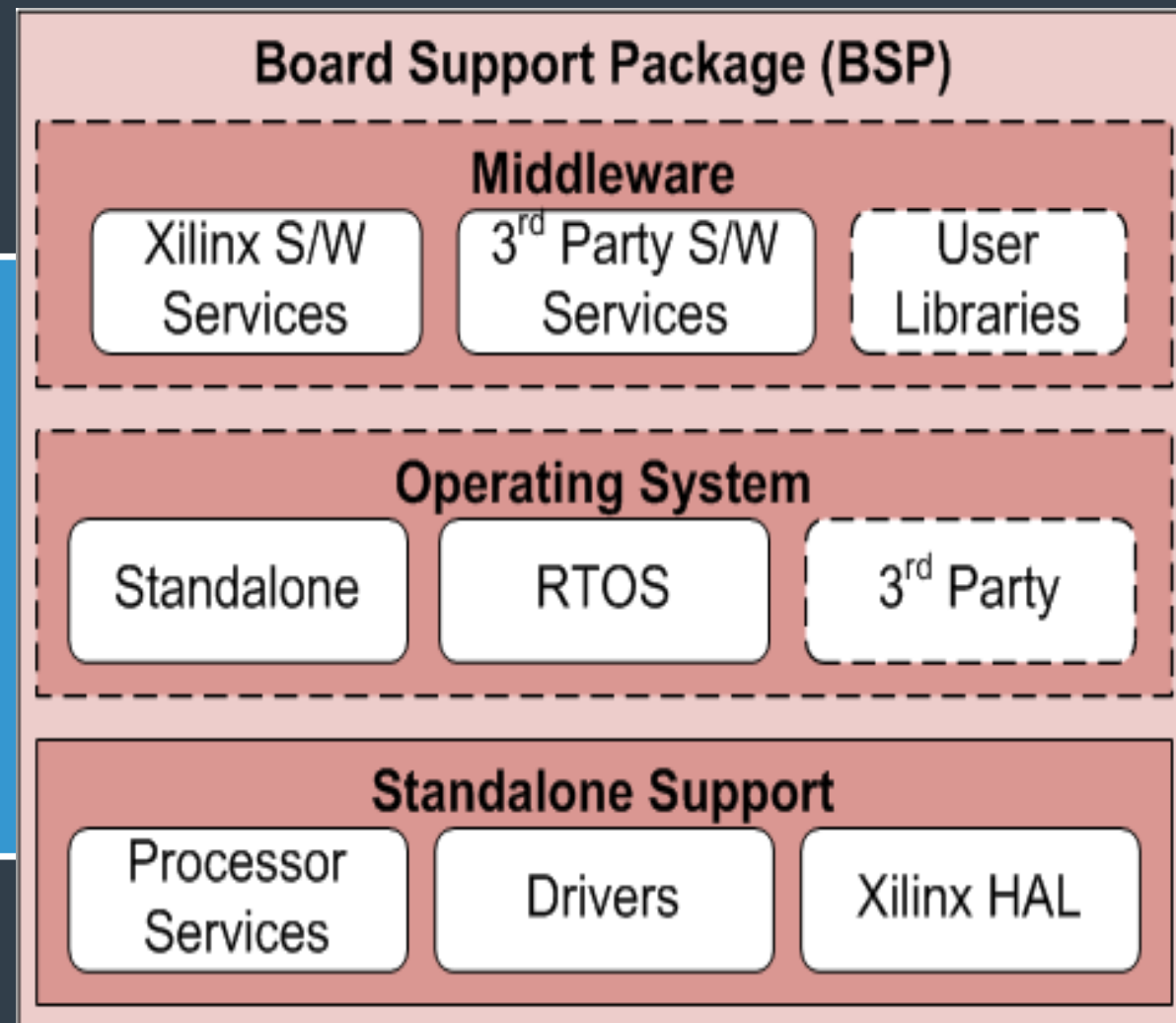
Domain and Board Support Package

Workspaces, Projects, and Perspectives

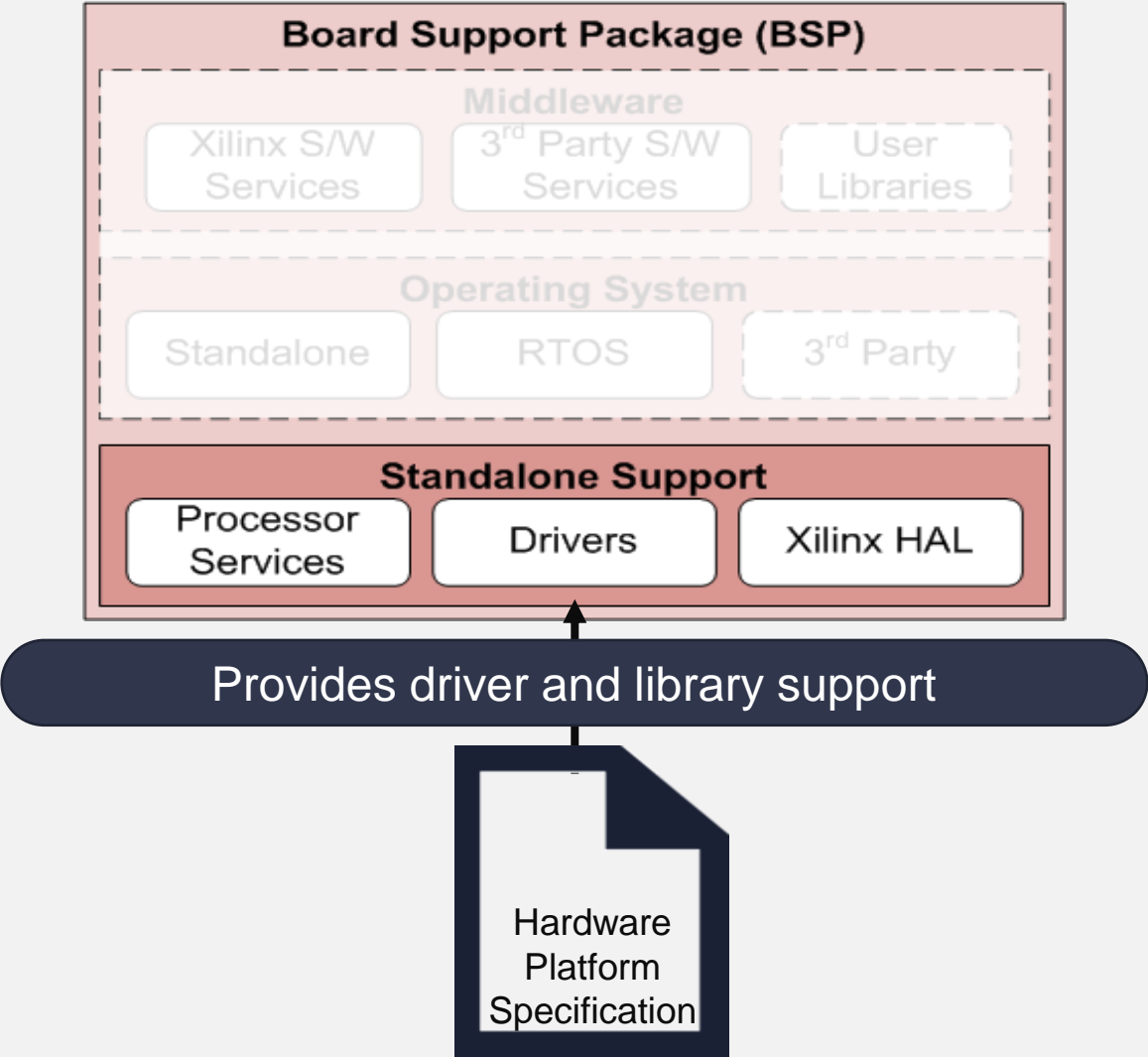


Board Support Packages (BSP)

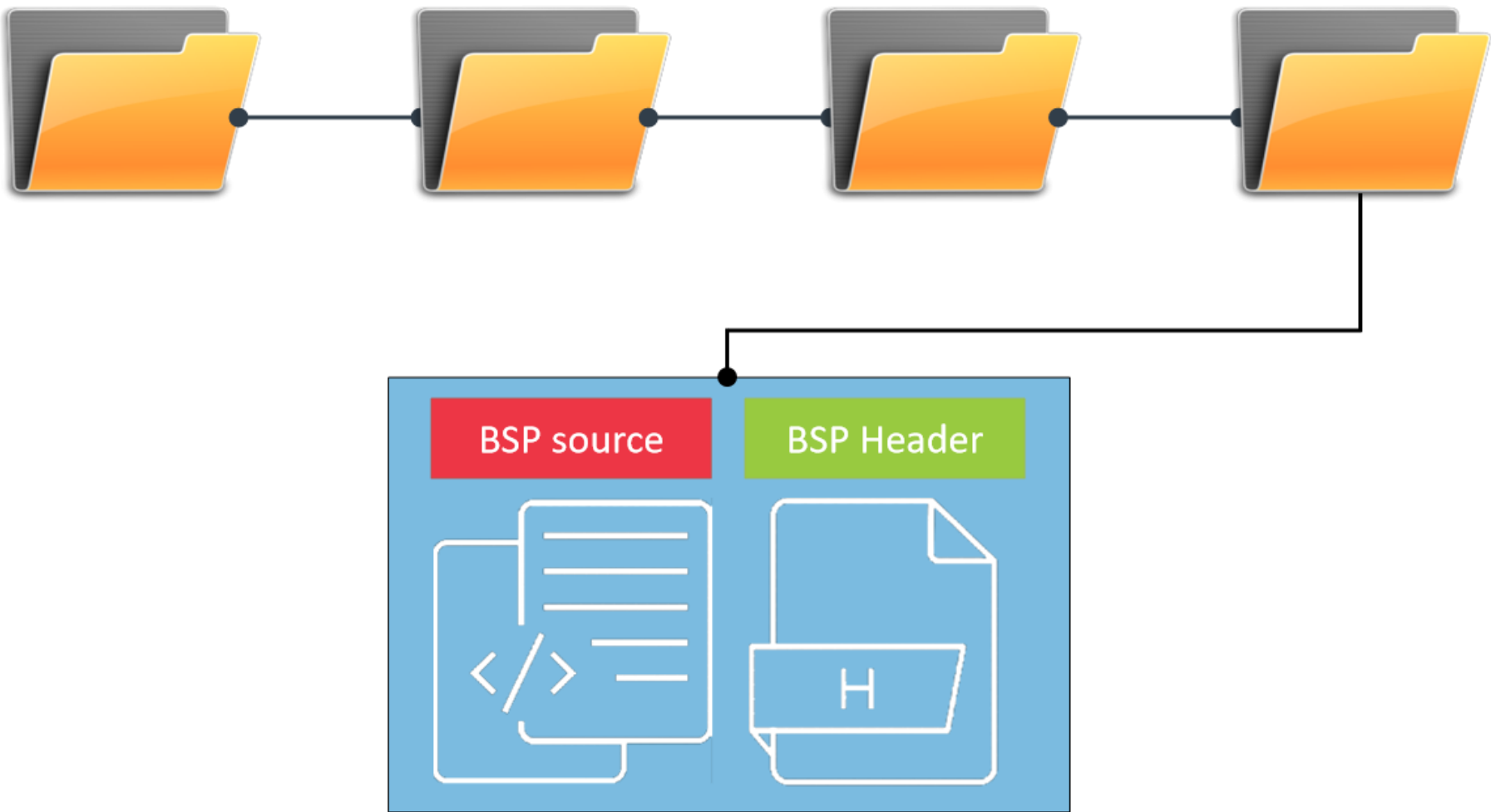
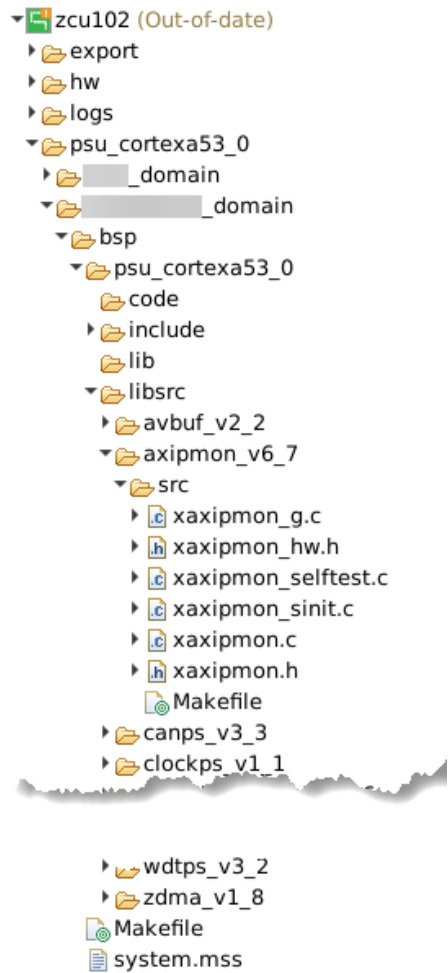
- Customized libraries, utilities, and drivers
- Created by the tool in minutes
- Xilinx BSP services:
 - Error free
 - Use standard APIs
 - No user maintenance of the BSP



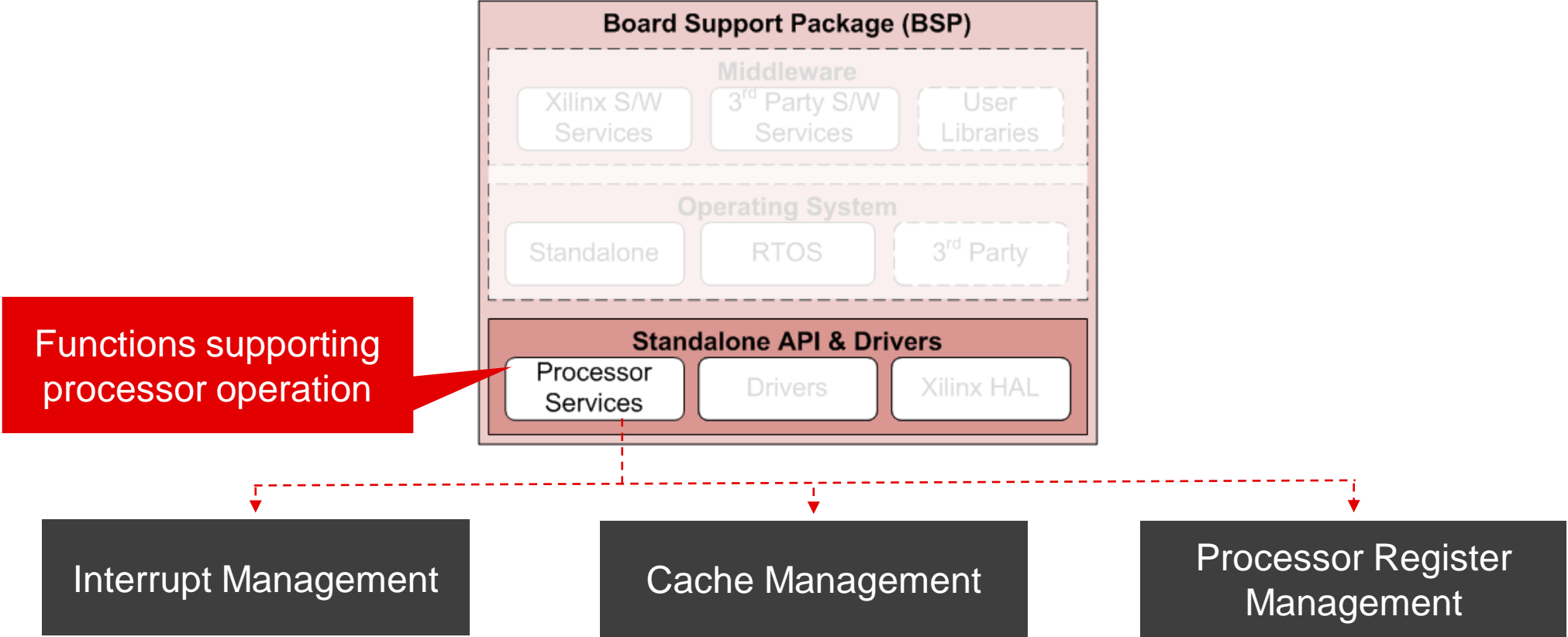
BSP: Standalone API and Drivers



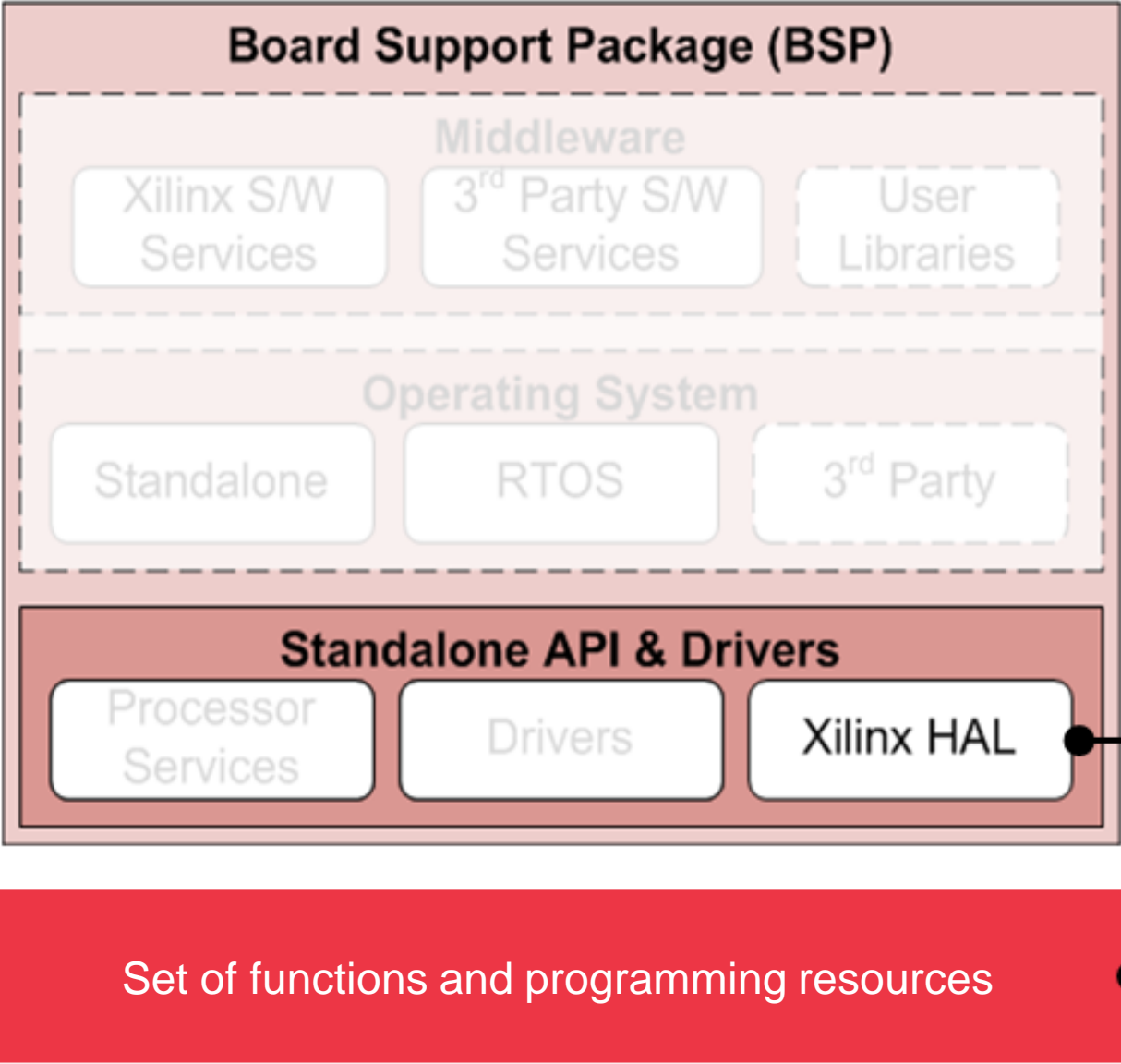
BSP: Standalone API and Drivers



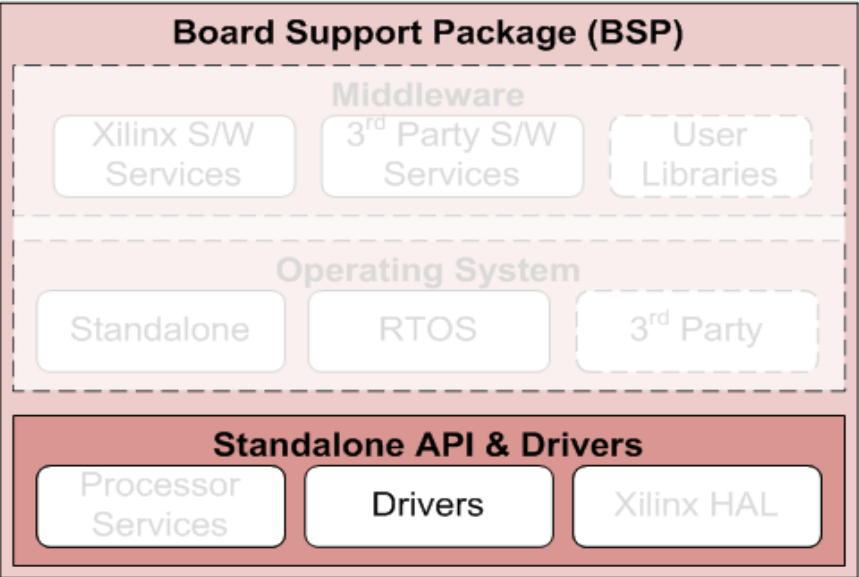
BSP: Standalone Processor Services



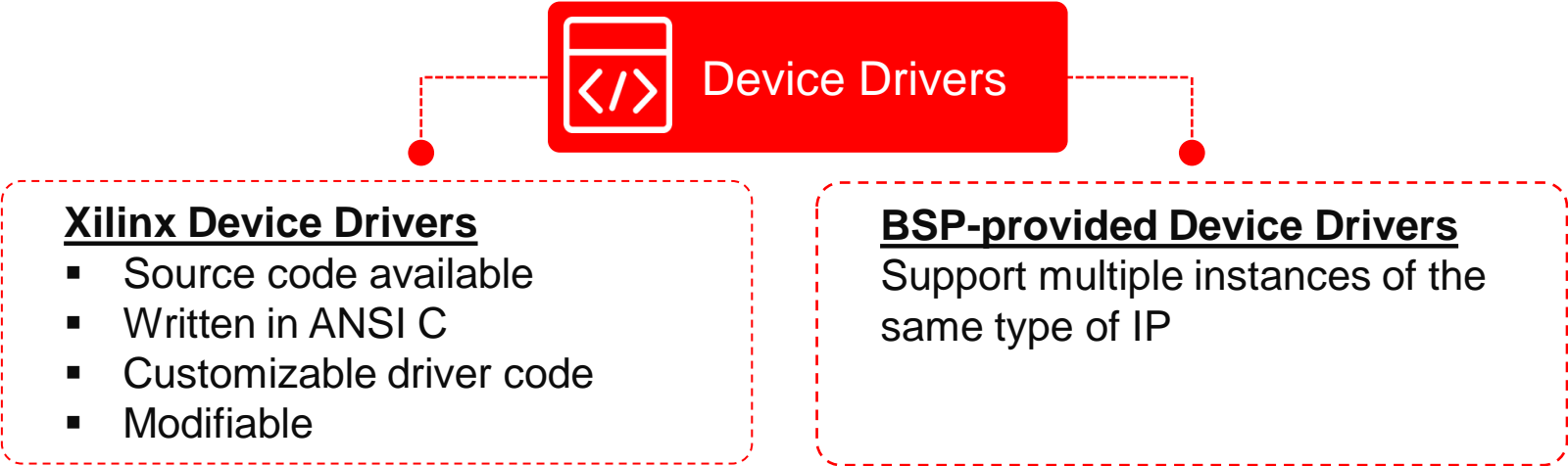
BSP: Standalone HAL



BSP: Device Drivers



- Peripheral register construction
 - Internal timing
 - Operational control
- Consistent API for user
- Supports varying functionality
 - Configurable at runtime and compile time



BSP: Device Drivers

Device Drivers: Layered Approach

Higher-level Drivers

- Enables the full capability of the peripheral
 - Supports block data and interrupts on message arrival

Lower-level Drivers

- Supports basic operations
 - UART-Lite drivers only support single-character transmission and reception

Layered approach separates device communication from OS dependencies

Drivers: Level 0/Level 1

- ▶ The layered architecture provides seamless integration with...
 - (Level 2) RTOS application layer
 - (Level 1) High-level device drivers that are full-featured and portable across operating systems and processors
 - (Level 0) Low-level drivers for simple use cases

Level 2, RTOS Adaptation
Level 1, High-level Drivers
Level 0, Low-level Drivers

Drivers: Level 0

- ▶ Consists of low-level device drivers
- ▶ Implemented as macros and functions that are designed to allow a developer to create a small system
- ▶ Characteristics:
 - Small memory footprint
 - Little to no error checking is performed
 - Supports primary device features only
 - No support of device configuration parameters
 - Supports multiple instances of a device with base address input to the API
 - Polled I/O only
 - Blocking function calls

Drivers: Level 1

- ▶ Consists of high-level device drivers
- ▶ Implemented as macros and functions and designed to allow a developer to utilize all of the features of a device
- ▶ Characteristics:
 - Abstract API that isolates the API from hardware device changes
 - Supports device configuration parameters
 - Supports multiple instances of a device
 - Polled and interrupt driven I/O
 - Non-blocking function calls to aid complex applications
 - May have a large memory footprint
 - Typically, provides buffer interfaces for data transfers as opposed to byte interfaces

Comparison Example

UARTPS Level 1

- ▶ XUartPs_CfgInitialize()
 - Initializes a specific XUartPs instance
- ▶ XUartPs_Send()
 - Sends the specified buffer using the device
 - polled or interrupt mode
- ▶ XUartPs_Recv()
 - Receive a specified number of bytes
 - store it into the specified buffer
- ▶ XUartPs_SetBaudRate()
 - Sets the baud rate

UARTPS Level 0

- ▶ XUartPs_SendByte()
 - Sends one byte
- ▶ XUartPs_RecvByte()
 - Receives one byte

Driver Settings

- ▶ Select the Drivers panel
- ▶ By default, the Driver panel displays which device driver is used for each hardware instance in the design
- ▶ Enables selection of custom drivers and versions for each device in the design

Overview

- standalone
- drivers**
- cpu_cortexa9

Drivers

The table below lists all the components found in your hardware system. You can modify the driver assigned for each component. If you do not want to assign a driver to a component or you want to assign a custom driver, select 'none'.

Component	Component Type	Driver	Dr...
ps7_cortexa9_0	ps7_cortexa9	cpu_cortexa9	1...
axi_bram_ctrl_0	axi_bram_ctrl	bram	3...
dip	axi_gpio	gpio	3...
led_ip_0	led_ip	generic	1...
ps7_ddr_0	ps7_ddr	none	1...
ps7_ddrc_0	ps7_ddrc	generic	1...
ps7_dev_cfg_0	ps7_dev_cfg	led_ip	2...
ps7_dma_ns	ps7_dma	dmaps	1...

Summary

Summary

- ▶ Software development for an embedded system in FPGA imposes unique challenges due to unique hardware platform
- ▶ Vitis IDE provides many rich perspectives which enable ease of accessing information through related views
- ▶ GNU tools are used for compiling C/C++ source files, linking, creating executable output, and debugging
- ▶ Software platform settings allow inclusion of software library support
- ▶ Compiler settings provide switches including compiling, linking, debugging, and profiling

Summary

- ▶ Embedded processor design requires you to manage
 - Peripheral address space
 - Memory address space to store data and instructions
 - Internal block memory
 - External memory
- ▶ Linker script is required when the software segments do not reside in a contiguous memory space



Thank You

Disclaimer and Attribution

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© Copyright 2022 Advanced Micro Devices, Inc. All rights reserved. Xilinx, the Xilinx logo, AMD, the AMD Arrow logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

