Platform Creation

SDx 2018.2

XILINX

Objectives

> **After completing this module, you will be able to:**

- Describe what platform is
- List platform’s components
- List supported operating systems
- Understand the platform project creation functionality

Outline

> Introduction
> SDSoC Platform Components
> Creating an SDSoC Platform
> Summary
> Lab7 Intro

What is an SDSoC Platform?

- **An SDSoC platform defines**
  - A base hardware and software architecture and application context
    - The hardware includes processing system, external memory interfaces, and custom input/output
    - The software run time includes operating system (possibly "bare metal"), boot loaders, drivers for platform peripherals and root file system
- **Every project in the SDSoC environment targets a specific platform**
- **Use SDSoC IDE to customize the platform with application-specific hardware accelerators and data motion networks that connect accelerators to the platform**

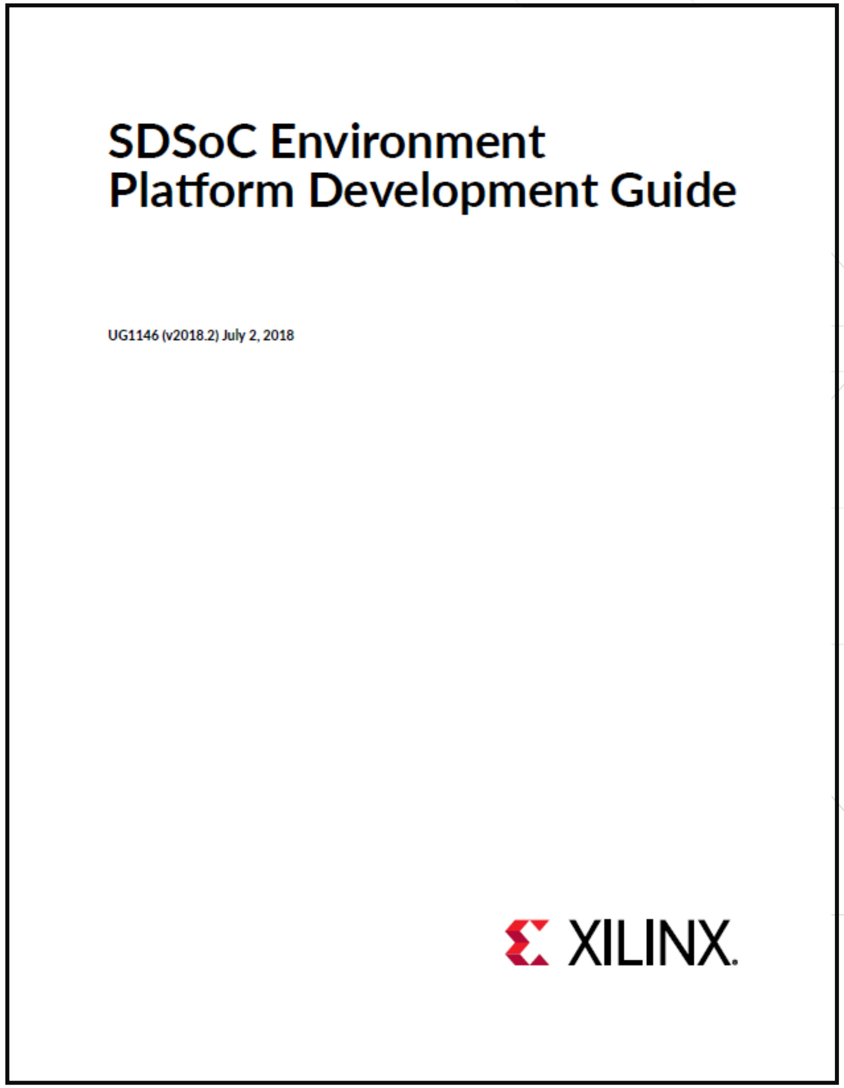
SDSoC: Software-defined Systems-on-Chip

- **A *software-defined SoC* extends a platform with application-specific hardware *and* software to realize a software application**
  - Multiple applications can target a given platform
  - An application can target multiple platforms

- **Application software defines the SoC**
  - User specifies hardware functions to implement in programmable logic
  - System-optimizing compiler analyzes program dataflow and compiles program into an application-specific SoC
    - Analysis engine employs mappings from function prototypes onto IP blocks
    - Function argument properties are constraints to the system optimizing compiler
  - **`#pragma`** assist the compiler and override inference engine

SDSoC: Platform Hardware is a Vivado Design

- **…with a well-defined “platform interface”**
  - AXI, AXI-S, clocks, resets, interrupts
- **Zynq processing system**
- **Memory interfaces and custom I/O**
  - Often domain-specific


SDSoC Environment
Platform Development Guide

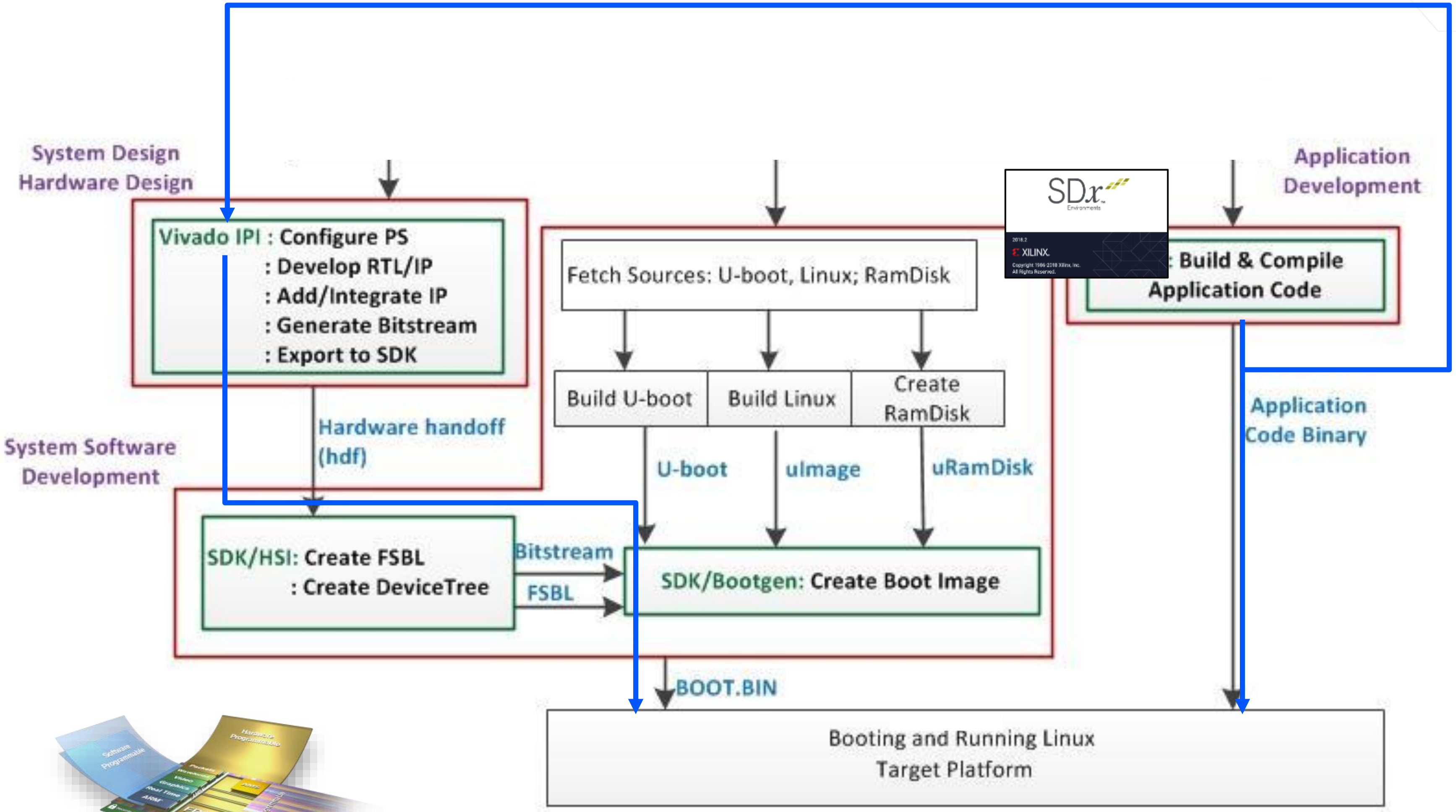
UG1146 (v2018.2) July 2, 2018

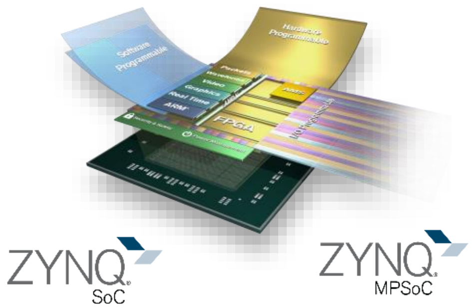
XILINX.

SDSoC: Software is also Part of the Platform

> Operating systems: Linux, FreeRTOS, or bare metal
> Device drivers for platform peripherals
> Boot loaders, e.g., FSBL, u-boot
> Root file system
> Libraries

System-on-Zynq SoC / MPSoC Platform Software





source: wiki.xilinx.com

SDSoC Platforms

- The SDSoC development environment includes standard “memory-based I/O” platforms
  - zc702, zc706, zed
  - zcu102, zcu104, zcu106
- Additional downloads from Xilinx and partners
  - Zynq Base Targeted Reference Design
  - http://www.xilinx.com/products/design-tools/software-zone/sdsoc.html#boardskits
- And several “teaching” platform examples
  - How to support direct I/O
  - How to use platform-specific libraries
  - How to share PS7 AXI interfaces between platform and SDSoC

SDSoC Platform Components

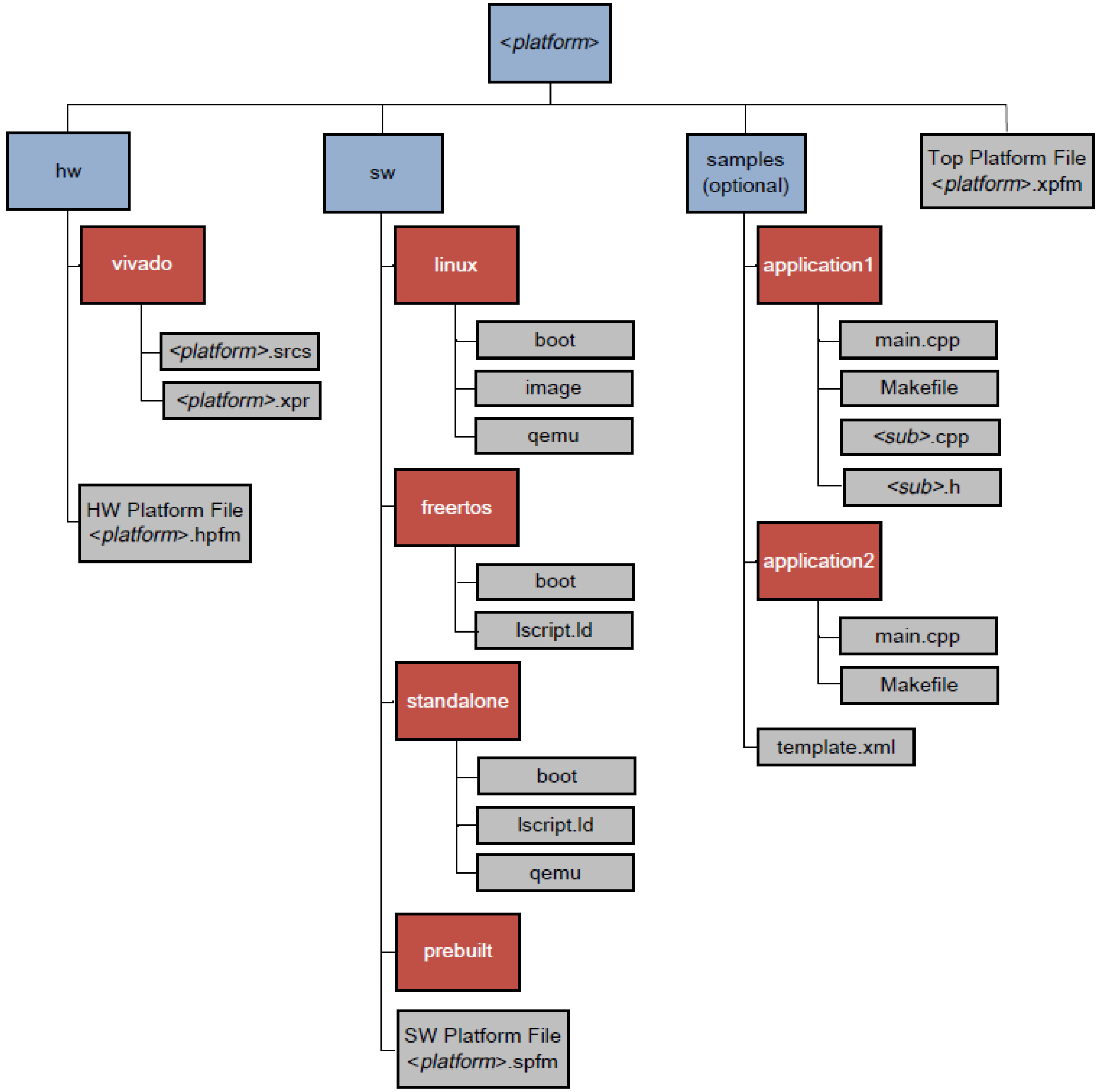
XILINX

SDSoC Platform Components

- An SDSoC platform consists of the following elements
  - Hardware Folder
    - Device Support Archive (DSA) file
  - Software Folder
    - System Configurations and Processor Domains defining the boot process and OS assignment per processor
    - Common boot objects (first stage boot loader, for Zynq® UltraScale+™ MPSoC Arm® trusted firmware and power management unit firmware)
    - Linux related objects (u-boot and Linux device tree, kernel and ramdisk as discrete objects or an image.ub FIT boot image)
    - Pre-built hardware files (optional)
      - Bitstream
      - Exported hardware files for SDK
      - Pre-generated port information software file
      - Pre-generated software interface files
    - Library header files (optional)
    - Static libraries (optional)
  - Metadata files generated as part of the platform project
  - Platform sample applications (optional)

- **Metadata files**
  - Top-level description file <platform>.xpfm
  - Hardware description file <platform>.dsa
  - Software description file <platform>.spfm

Directory Structure



Vivado Generated Hardware

- Build and verify the hardware system using the Vivado Design Suite and IP integrator feature
  - Generate Output Products of the IP in the block design
  - Use the Create HDL Wrapper command to create the top-level RTL design
- Configure platform and interface properties either from the Platform Interfaces tab or using TCL commands (see next slide)
- Write and validate DSA file using `write_sda` and `validate_sda` commands

- Hardware design requirements
  - The Vivado project name must match the hardware platform name
  - Every IP used in the platform design that is not part of the standard IP catalog must be local to the project
  - Every hardware platform design must contain a PS IP block from the Xilinx IP catalog
  - Every hardware port interface to the SDSoC platform must be an AXI,AXI4-Stream, clock, reset, or interrupt type interface only
    - Custom bus types or hardware interfaces must remain internal to the hardware platform
  - Every platform must declare at least one general purpose AXI master port from PS or an interconnect IP connected to such an AXI master port
  - Every platform must declare at least one AXI slave port that will be used by the compilers to access DDR from datamover and accelerator IP

Generating DSA using Tcl Commands

- Add platform properties (PFM) to define the platform name and configure platform interfaces
  - Platform identification property
    - `set_property` **PFM_NAME** `string [get_files design.bd]`
  - Four interface properties
    - `set_property` **PFM.AXI_PORT** `{ <port_name> {parameters} \ <port2> {parameters} ...} [get_bd_cells <cell_name>]`
    - `set_property` **PFM.AXIS_PORT** `{ <port_name> {parameters} \ <port2> {parameters} ...} [get_bd_cells <cell_name>]`
    - `set_property` **PFM.CLOCK** `{ <port_name> {parameters} \ <port2> {parameters} ...} [get_bd_cells <cell_name>]`
    - `set_property` **PFM.IRQ** `{ <port_name> {} <port2> {} ...} \ [get_bd_cells <cell_name>]`

Generating DSA (2)

- Define an AXI_port on interconnect (assuming two ports are already implemented):
  - ```
    set parVal []
    for {set i 2} {$i < 64} {incr i} {
    lappend parVal M[format %02d $i]_AXI \
    {memport "M_AXI_GP"}
    }
    ```
  - ```
    set_property PFM.AXI_PORT $parVal [get_bd_cells /axi_interconnect_0]
    ```
- Define Interrupt ports
  - ```
    set_property PFM.IRQ { <port_name> {} <port2> {} ...} \
    [get_bd_cells <cell_name>]
    ```

- Generate a DSA file using the `write_dsa` command from the Tcl console in Vivado
    ```

DSA Tcl File Example

```
set design_name "pynq_z2"

# open project and block design
open_project -quiet ./${design_name}/${design_name}.xpr
open_bd_design ./${design_name}/${design_name}.srcs/sources_1/bd/${design_name}/${design_name}.bd

# set sdx platform properties
set_property PFM_NAME "xilinx.com:${design_name}:${design_name}:1.0" \
        [get_files ./${design_name}/${design_name}.srcs/sources_1/bd/${design_name}/${design_name}.bd]
set_property PFM.CLOCK { \
    clk_out1 {id "0" is_default "true" proc_sys_reset "proc_sys_reset_0" } \
    clk_out2 {id "1" is_default "false" proc_sys_reset "proc_sys_reset_1" } \
    clk_out3 {id "2" is_default "false" proc_sys_reset "proc_sys_reset_2" } \
    clk_out4 {id "3" is_default "false" proc_sys_reset "proc_sys_reset_3" } \
    } [get_bd_cells /clk_wiz_0]
set_property PFM.AXI_PORT { \
    M_AXI_GP0 {memport "M_AXI_GP"} \
    M_AXI_GP1 {memport "M_AXI_GP"} \
    S_AXI_ACP {memport "S_AXI_ACP" sptag "ACP" memory "ps7_0 ACP_DDR_LOWOCM"} \
    S_AXI_HP0 {memport "S_AXI_HP" sptag "HP0" memory "ps7_0 HP0_DDR_LOWOCM"} \
    S_AXI_HP1 {memport "S_AXI_HP" sptag "HP1" memory "ps7_0 HP1_DDR_LOWOCM"} \
    S_AXI_HP2 {memport "S_AXI_HP" sptag "HP2" memory "ps7_0 HP2_DDR_LOWOCM"} \
    S_AXI_HP3 {memport "S_AXI_HP" sptag "HP3" memory "ps7_0 HP3_DDR_LOWOCM"} \
    } [get_bd_cells /ps7_0]

set intVar []
for {set i 0} {$i < 16} {incr i} {
    lappend intVar In$i {}
}
set_property PFM.IRQ $intVar [get_bd_cells /xlconcat_0]

generate_target all \
[get_files ./${design_name}/${design_name}.srcs/sources_1/bd/${design_name}/${design_name}.bd]

# generate dsa
write_dsa -force ./${design_name}.dsa
validate_dsa ./${design_name}.dsa
```

Hardware Metadata File (.hpfm) Generation from Tcl

- Use the Tcl file as shown on
  - Make sure that the instance names matches instances in your design

```
set pfm [sdsoc::create_pfm zc702.hpfm]
sdsoc::pfm_name        $pfm "xilinx.com" "xd" "zc702" "1.0"
sdsoc::pfm_description $pfm "Zynq ZC702 Board"
sdsoc::pfm_clock       $pfm FCLK_CLK0 ps7 0 false proc_sys_reset_0
sdsoc::pfm_clock       $pfm FCLK_CLK1 ps7 1 false proc_sys_reset_1
sdsoc::pfm_clock       $pfm FCLK_CLK2 ps7 2 true  proc_sys_reset_2
sdsoc::pfm_clock       $pfm FCLK_CLK3 ps7 3 false proc_sys_reset_3
sdsoc::pfm_axi_port    $pfm S_AXI_ACP ps7 S_AXI_ACP
for {set i 1} {$i < 64} {incr i} {
  sdsoc::pfm_axi_port  $pfm M[format %02d $i]_AXI axi_ic_gp0 M_AXI_GP
}
sdsoc::pfm_axi_port    $pfm S_AXI_HP0 ps7 S_AXI_HP
sdsoc::pfm_axi_port    $pfm S_AXI_HP1 ps7 S_AXI_HP
sdsoc::pfm_axi_port    $pfm M_AXI_GP1 ps7 M_AXI_GP
for {set i 0} {$i < 16} {incr i} {
  sdsoc::pfm_irq       $pfm In$i xlconcat
}
sdsoc::generate_hw_pfm $pfm
```

Platform Clocks

AXI Ports

63 ports made available on AXI Interconnect instance axi_ic_gp0 to the SDS linker connected to M_AXI_GP0

Platform Interrupts

IPI port name

IPI block name

Memory type ∈ { S_AXI_ACP, S_AXI_HP, M_AXI_GP, MIG }

XILINX.

Software Requirements

- **Software components**
  - Operating system
  - Boot loaders, and
  - Libraries
- **Supported OS**
  - Standalone (use Xilinx SDK)
  - Linux (use Xilinx PetaLinux tools)
  - FreeRTOS (use Xilinx SDK)
- **Platform may have one or more OS support**
- **By default, the SDSoC environment creates an SD card image to boot a board into a Linux prompt or execute a standalone program**

Software Requirements (2)

- Boot files - first stage bootloader or FSBL; U-boot; Linux unified boot image `image.ub` or separate `devicetree.dtb`, kernel and ramdisk files; boot image file or BIF used to create `BOOT.BIN` boot files
- Optional prebuilt data used by SDSoC when building applications without hardware accelerators, such as a pre-generated hardware bitstream to save time and SDSoC data files
  - You cannot add accelerators when prebuilt is used
- Optional header and library files if the platform provides software libraries
- Optional emulation data files, if the platform supports emulation flows using the Vivado Simulator for programmable logic and QEMU for the processing subsystem

Standalone OS Boot Files

- **A standalone boot image is created using:**
  - First Stage Boot Loader (FSBL)
    - Export hardware from Vivado and invoke XSDK
    - Create a new application project using ZynqFSBL template in XSDK
    - The platform creation wizard will use the generated `fsbl.elf` file
    - The sds++ compiler will use the `fsbl.elf` to generate the boot image, `boot.bin`
  - Boot Information File (BIF)
    - A BIF file is required to use an executable in the boot image
    - The file should have a pointer to where the `fsbl.elf` is located
  - Linker script file
    - The file is required while linking the baremetal (standalone) executable
    - Make sure that the stack and heap memory is larger compared to the default values; E.g.
      - Stack size `0x40000`
      - Heap size `0x4000000`

Linux Boot Files

- Use PetaLinux tools flow
  - Set up shell environment with PetaLinux tools in your PATH environment variable
  - Create and `cd` into a working directory
  - Create a new PetaLinux project targeting a BSP targeting the desired board
    `petalinux-create -t project -n <project_name > -s <path_to_base_BSP>`
  - Use the hardware handoff file (.hdf) from the Vivado project for the custom hardware platform
    `petalinux-config -p <petalinux_project> --get-hw-description=<HDF path>`
  - Configure PetaLinux kernel
    `petalinux-config -p <petalinux_project> -c kernel`
  - Configure Petalinux rootfs
    `petalinux-config -p <petalinux_project> -c rootfs`
  - Add device tree fragment for APF driver
    - At the bottom of <>/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi add
      `/{ xlnk { compatible = "xlnx,xlnk-1.0"; }; };`
  - Build the PetaLinux image
    `petalinux-build -p software`

Creating an SDSoC Platform

Creating an SDSoC Platform from SDx

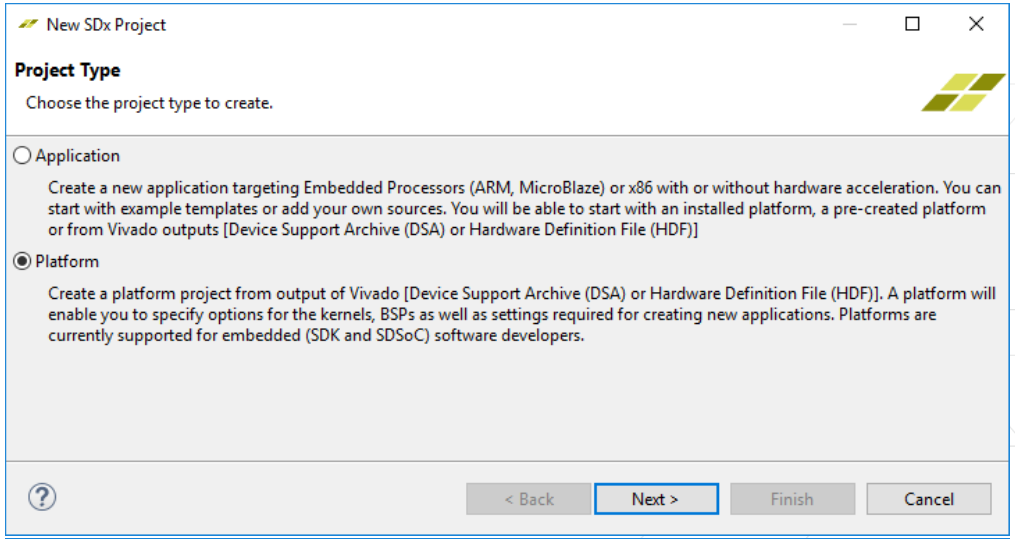
> **Start SDx**

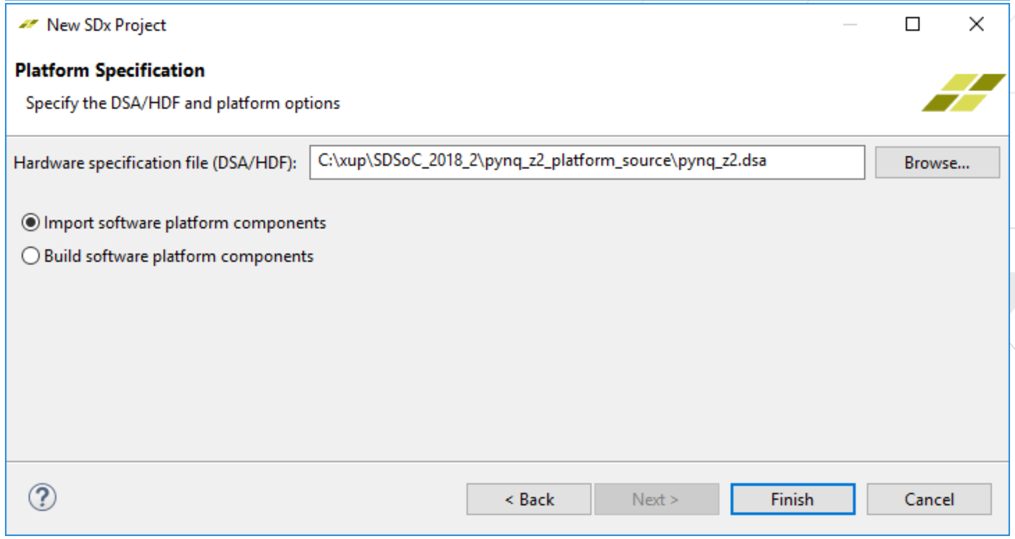
> **Identify workspace**

> **Select *Platform* as the Project Type**

> **Specify Hardware Platform DSA**
  >> This is generated using Vivado flow
  >> Select `Import software platform components` option

> **Click *Finish* to create the project**

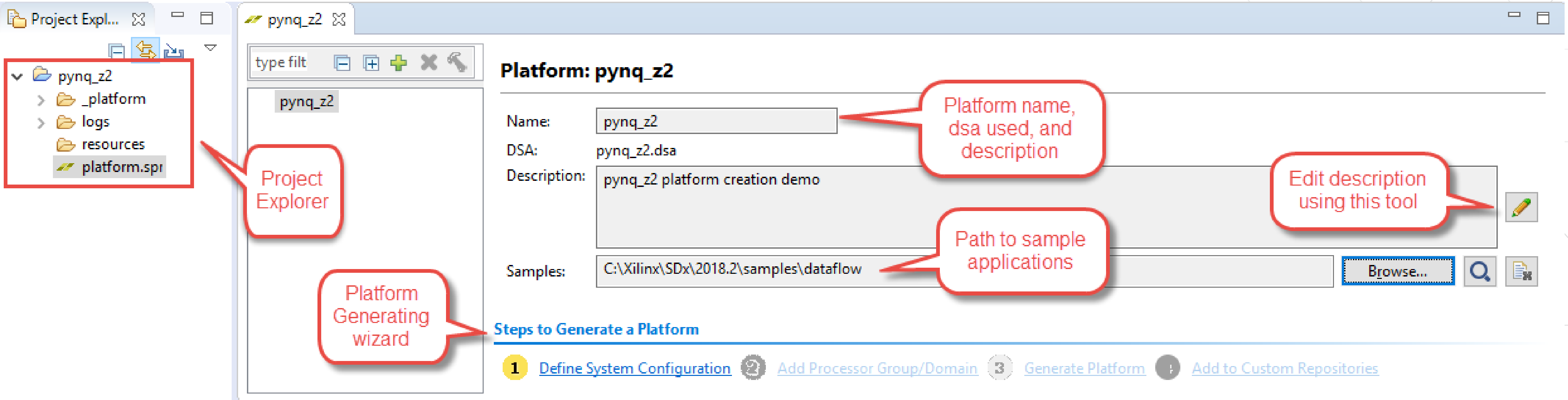




XILINX

Created Platform Project

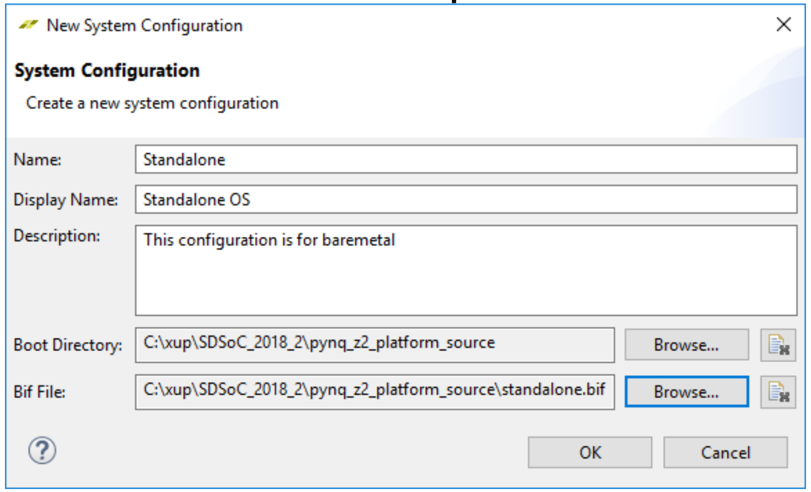
- **Platform project is created**
  - Cannot change platform name
  - Edit *Description* field with the Pencil tool
  - Set path, if applicable, to where sample applications are provided in the *Samples* field
  - Click on **Define System Configuration** link as the first step

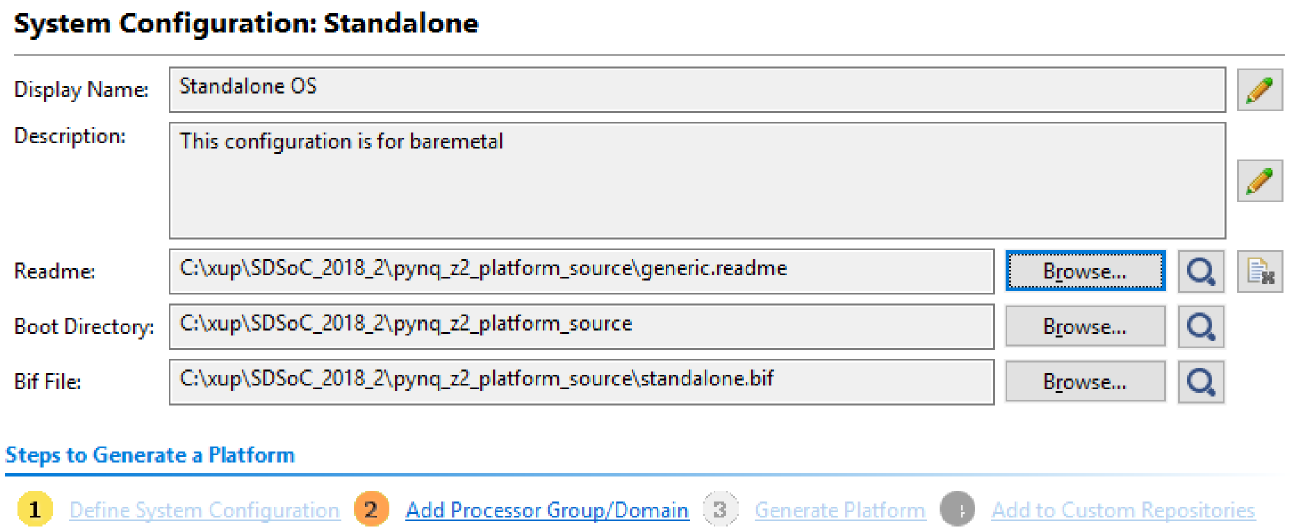


XILINX.

Wizard - Defining System Configuration

- Defines the software environment that is booted and runs on the hardware platform
  - Specify OS and run-time settings
  - Include software-configurable hardware parameters
- After entering System Configuration name, provide path in *Boot Directory* and *BIF File* fields, and click **OK**
  - Boot directory will contain files, such as the FSBL, U-Boot, ARM Trusted Firmware, etc. referenced in the BIF file
- Provide *Readme* file path

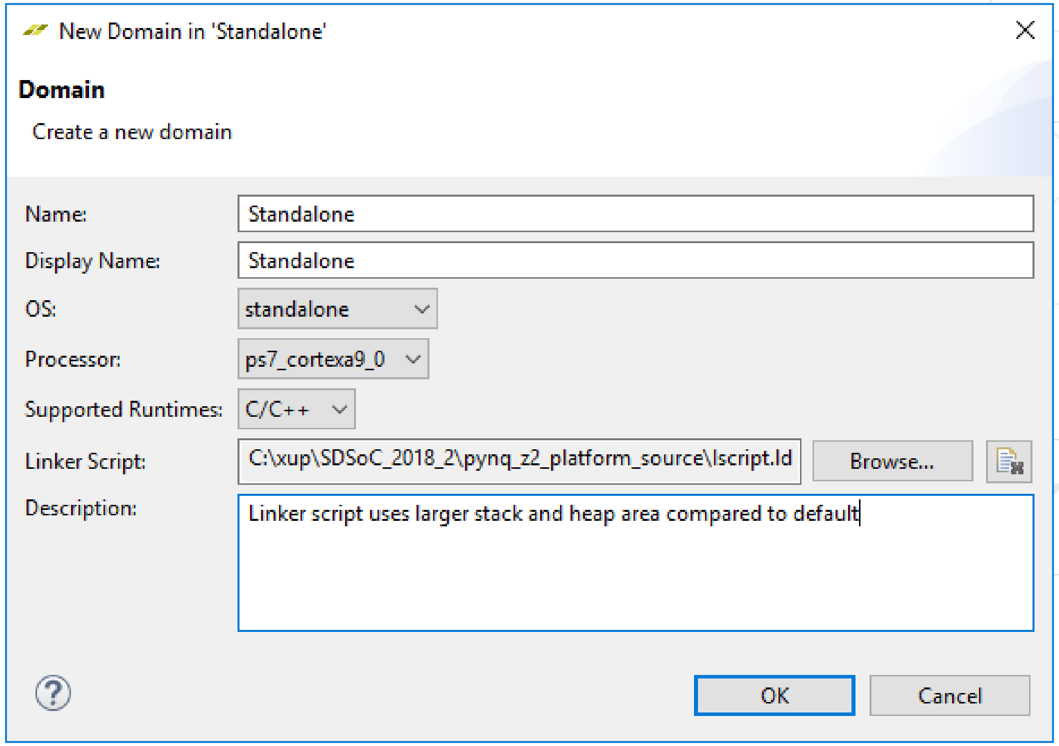




XILINX

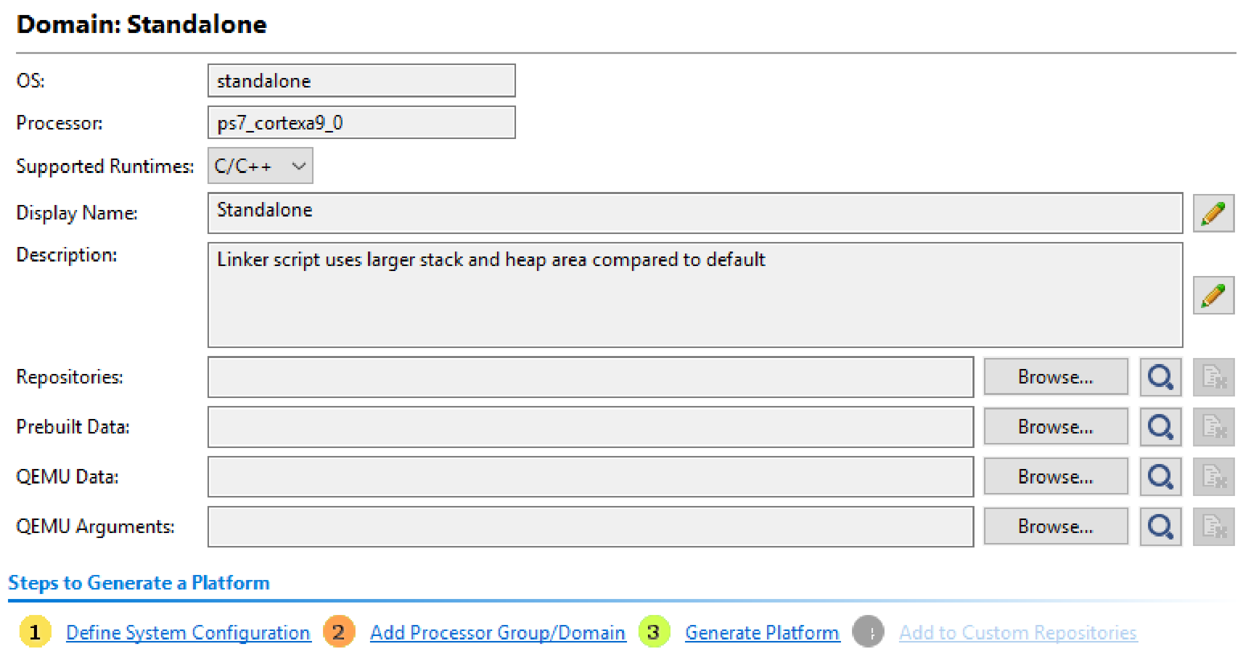
Wizard – Adding Processor Domain

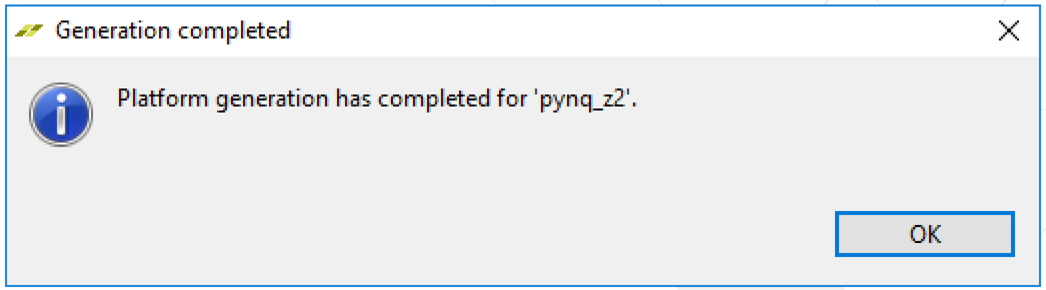
- **The Processor Domain defines the OS operating on one or more processors on the device, and the run-time**
- **Click on the `Add Processor Group/Domain` link**
  - Fill Name and Display Name fields
  - Select OS and processor
    - standalone
    - linux
    - freertos
    - freertos10_Xilinx
  - For standalone, freertos, freertos10_Xilinx provide linker script file
  - For linux provide prebuilt linux image file
  - Click **OK**



Generate Platform

- **Enter/Edit various paths and description as needed**
- **Click on the `Generate Platform` link**
  - The platform directory structure will be created under the **export** directory in the current project
  - Click **OK** to close the *Generation completed* dialog box

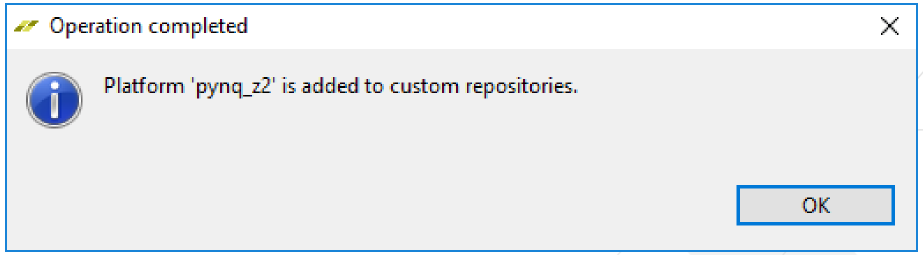




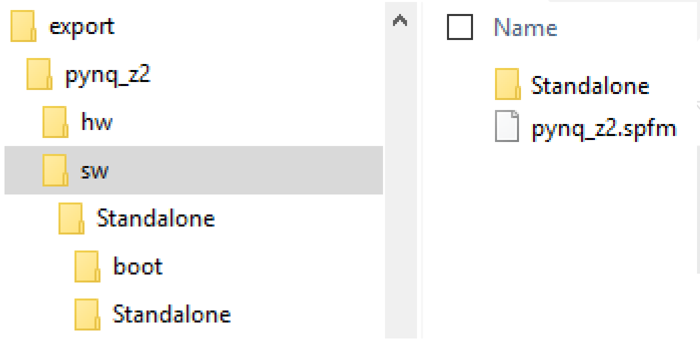
Add the Generated Platform to the Repository

> **Click on the `Add to Custom Repositories` link**

>> Click **OK** to close the dialog box indicating the platform is added to the Custom Repositories



>> The custom platform directory will be generated under the export directory under the current workspace



Top-Level Created Metadata File (.xpfm)

> Contains references to the hardware (*.dsa) and software XML file (*.spfm) and the folders that contain them

> Includes **vendor, library, platform name, and version number**

> Also includes description

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sdx:platform sdx:vendor="xilinx.com"
              sdx:library="sdx"
              sdx:name="pynq_z2"
              sdx:version="1.0"
              xmlns:sdx="http://www.xilinx.com/sdx">
    <sdx:description>
pynq_z2 platform creation demo
    </sdx:description>
    <sdx:hardwarePlatforms>
        <sdx:hardwarePlatform sdx:path="hw" sdx:name="pynq_z2.dsa"/>
    </sdx:hardwarePlatforms>
    <sdx:softwarePlatforms>
        <sdx:softwarePlatform sdx:path="sw" sdx:name="pynq_z2.spfm"/>
    </sdx:softwarePlatforms>
</sdx:platform>
```

Software Metadata File (.spfm)

> Describes the software environments, or system configurations available for use by the platform

> Each configuration has an operating system (OS) associated with it, and the user selects the system configuration when creating a design on the hardware platform

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sdx:platform sdx:vendor="xilinx.com"
              sdx:library="sdx"
              sdx:name="pynq_z2"
              sdx:version="1.0"
              sdx:schemaVersion="1.0"
              xmlns:sdx="http://www.xilinx.com/sdx">
    <sdx:description>
pynq_z2 platform creation demo
    </sdx:description>
    <sdx:systemConfigurations sdx:defaultConfiguration="Standalone">
        <sdx:configuration sdx:name="Standalone"
                           sdx:displayName="Standalone OS"
                           sdx:defaultProcessorGroup="Standalone"
                           sdx:runtimes="cpp">
            <sdx:description>This configuration is for baremetal</sdx:description>
            <sdx:bootImages sdx:default="standard">
                <sdx:image sdx:name="standard"
                           sdx:bif="Standalone/boot/standalone.bif"
                           sdx:readme="Standalone/boot/generic.readme"
                           />
            </sdx:bootImages>
            <sdx:processorGroup sdx:name="Standalone"
                                sdx:displayName="Standalone"
                                sdx:cpuType="cortex-a9"
                                sdx:cpuInstance="ps7_cortexa9_0">
                <sdx:os sdx:name="standalone"
                        sdx:displayName="standalone"
                        sdx:ldscript="Standalone/Standalone/lscript.ld"
                />
            </sdx:processorGroup>
        </sdx:configuration>
    </sdx:systemConfigurations>
</sdx:platform>
```

Each OS is defined using systemconfiguration

Software Metadata File (.spfm) (2)

- Indicates where to find boot files (first stage boot loader or FSBL) and Linux images used to generate an SD card image
- If the platform includes optional header and library files, SDSoC automatically adds paths to the files when compiling and linking the user's application
- Optional prebuilt hardware bitstreams reduce run times when building applications without hardware accelerators

Software Metadata File (.spfm) (3)

```
<sdx:systemConfigurations sdx:defaultConfiguration="linux">
    <sdx:configuration sdx:name="linux"
                       sdx:displayName="linux OS"
                       sdx:defaultProcessorGroup="linux"
                       sdx:runtimes="cpp">
        <sdx:description>This configuration is for Linux</sdx:description>
        <sdx:bootImages sdx:default="standard">
            <sdx:image sdx:name="standard"
                        sdx:bif="linux/boot/linux.bif"
                        sdx:imageData="linux/linux/image"
                        sdx:mountPath="/mnt"
                        sdx:readme="linux/boot/generic.readme"
                        sdx:qemuArguments="linux/qemu/qemu_args.txt"
                        />
        </sdx:bootImages>
        <sdx:processorGroup sdx:name="linux"
                            sdx:displayName="linux"
                            sdx:cpuType="cortex-a9">
            <sdx:os sdx:name="linux"
                    sdx:displayName="linux"
            />
        </sdx:processorGroup>
    </sdx:configuration>
</sdx:systemConfigurations>
```

Linux OS Support

```
/* linux */
the_ROM_image:
 {
     [bootloader]<boot/fsbl.elf>
     <bitstream>
     <boot/u-boot.elf>
 }
```

Refer to Chapter 4: Software Platform Data Creation of UG1146 for how to configure Linux

Software Metadata File (.spfm) (4)

```
<sdx:configuration sdx:name="Standalone"
                   sdx:displayName="Standalone OS"
                   sdx:defaultProcessorGroup="Standalone"
                   sdx:runtimes="cpp">
    <sdx:description>This configuration is for baremetal</sdx:descripti
    <sdx:bootImages sdx:default="standard">
        <sdx:image sdx:name="standard"
                    sdx:bif="Standalone/boot/standalone.bif"
                    sdx:readme="Standalone/boot/generic.readme"
                    sdx:qemuArguments="standalone/qemu/qemu_args.txt"
                    />
    </sdx:bootImages>
    <sdx:processorGroup sdx:name="Standalone"
                        sdx:displayName="Standalone"
                        sdx:cpuType="cortex-a9"
                        sdx:cpuInstance="ps7_cortexa9_0">
        <sdx:os sdx:name="standalone"
                sdx:displayName="standalone"
                sdx:ldscript="Standalone/Standalone/lscript.ld"
        />
    </sdx:processorGroup>
</sdx:configuration>
```

Standalone OS Support

```
/* standalone */
the_ROM_image:
 {
     [bootloader]<boot/fsbl.elf>
     <bitstream>
     <elf>
 }
```

Summary

XILINX

Summary

> **An SDSoC platform defines**
  >> A base hardware and software architecture and application context
  >> The hardware includes processing system, external memory interfaces, and custom input/output
  >> The software run time includes operating system (possibly "bare metal"), boot loaders, drivers for platform peripherals and root file system

> **Creating an SDSoC platform involves generating hardware and software meta data files**
  >> Use the Platform project type to generate the hardware and software metadata files
  >> Provide hardware dsa file

> **The platform name and the base directory name should be the same**

Lab7 Intro

XILINX

Lab7 Intro

- **Introduction**
  - This lab guides you through the steps involved in creating a custom platform
- **Objectives**
  - Create an SDSoC platform for a custom application
  - Use the SDSoC environment to test the custom platform

Adaptable.

Intelligent.