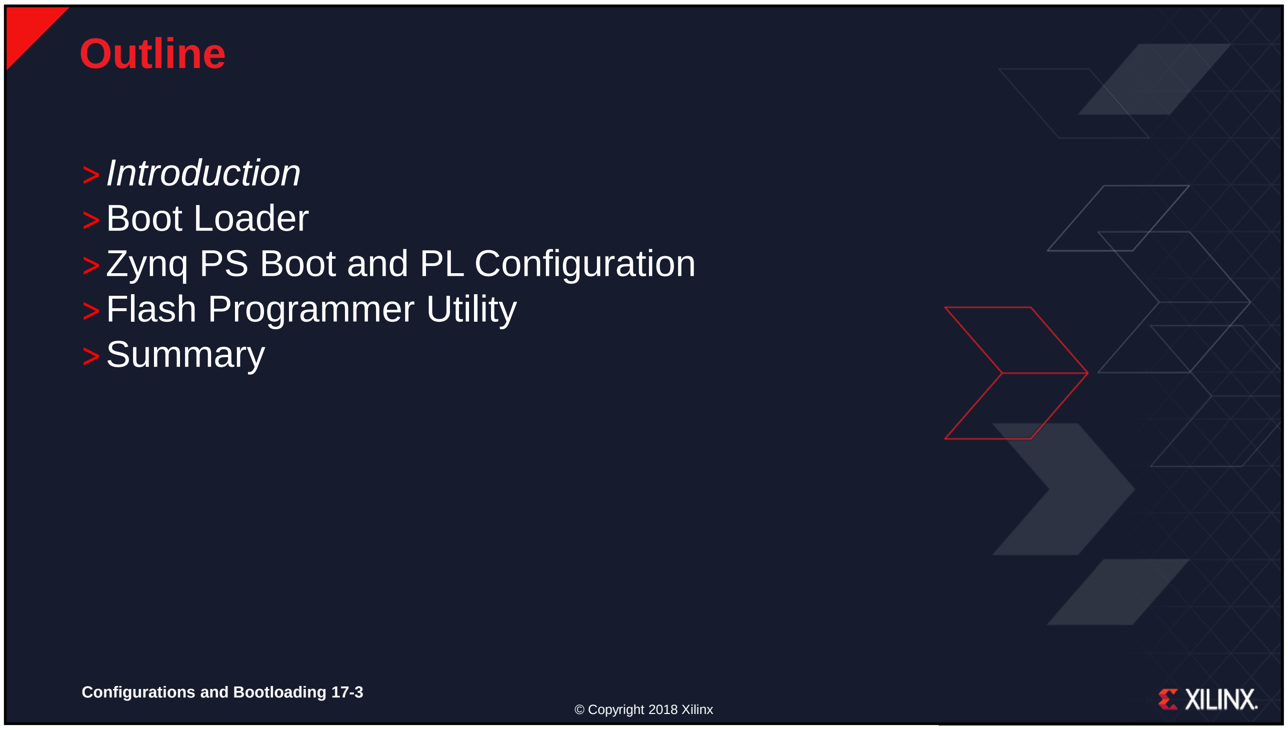
Configuration and Bootloading

XILINX.

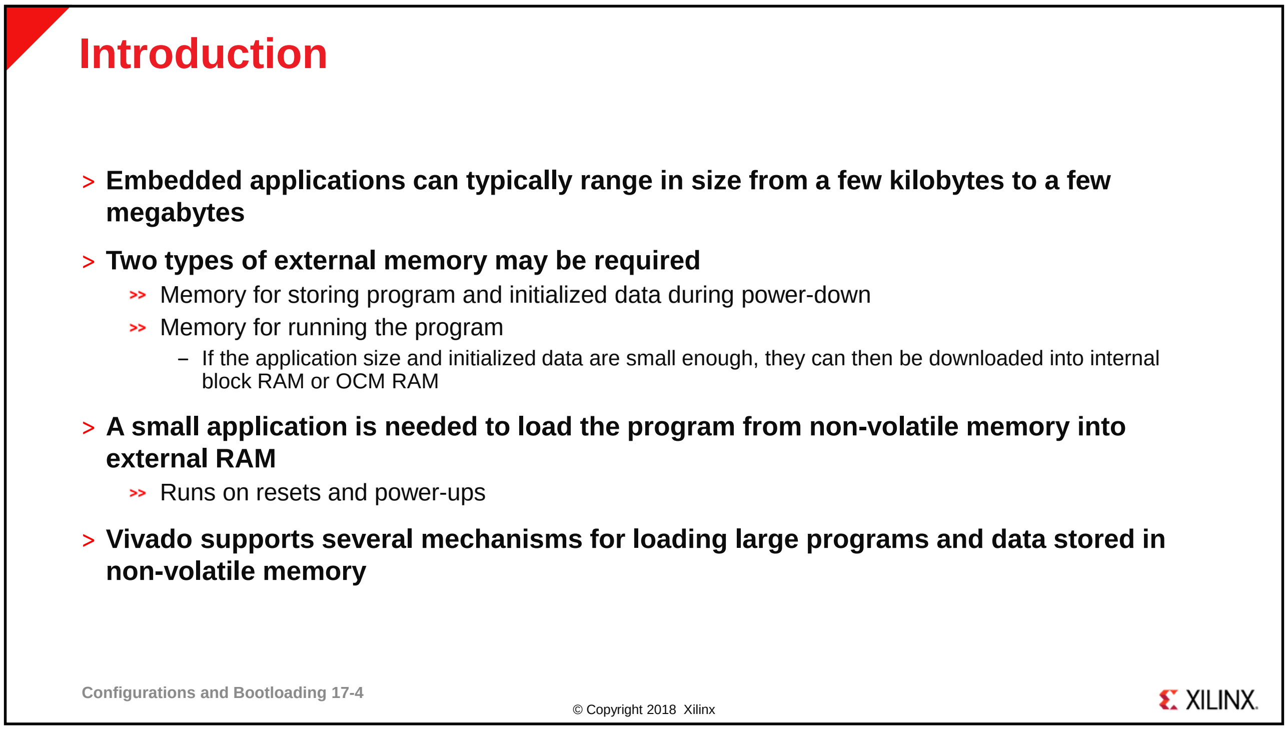
Objectives

> **After completing this module, you will be able to:**
  - State various mechanisms for system initialization and application loading
  - Describe the programmable logic configuration process from an FSBL software application
  - Describe the flash writer utility and its requirements
  - Analyze flash writing and different boot loading usage scenarios

XILINX.



Outline



Introduction

- **Embedded applications can typically range in size from a few kilobytes to a few megabytes**
- **Two types of external memory may be required**
  - Memory for storing program and initialized data during power-down
  - Memory for running the program
    - If the application size and initialized data are small enough, they can then be downloaded into internal block RAM or OCM RAM
- **A small application is needed to load the program from non-volatile memory into external RAM**
  - Runs on resets and power-ups
- **Vivado supports several mechanisms for loading large programs and data stored in non-volatile memory**

Standard Boot Model in Zynq SoC

- **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 – Uboot
    - 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 SoC Program Loading and Initialization

- **Development mode: Configure PL with bitstream then run the application from XMD**
- **Boot mode: FSBL or SSBL configure PL with bitstream**
  - Stage 1 or Stage 2, as mentioned in the previous slide
- **Size of application impacts where program can run from**
  - Very small applications can run from OCM (no DDR requirement)
  - Small applications can also run from BRAM (no DDR requirement)
  - Applications can run from non-volatile or DDR memory
- **Loading the application with a boot loader**
  - Use nonvolatile memory to store the application, initialize the processor memory from it, and execute
- **May execute application directly from flash or other non-volatile memory**
  - Slower execution

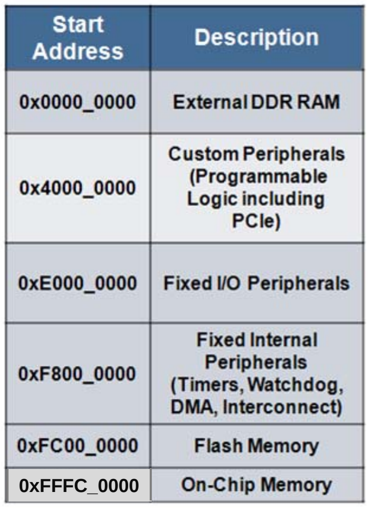
Cortex-A9 Processor Memory Space

> **Processing system and programmable logic look the same from the processor's viewpoint**

> **Zynq PS-based peripherals have a fixed address map**

> **PL-based slave peripherals must reside between 0x4000_0000 and 0xBFFF_FFFF**

  - Peripherals connected to M_AXI_GP0 will have address between 0x4000_0000 and 0x7FFF_FFFF
  - Peripherals connected to M_AXI_GP1 will have address between 0x8000_0000 and 0xBFFF_FFFF



Start Address	Description
0x0000_0000	External DDR RAM
0x4000_0000	Custom Peripherals (Programmable Logic including PCIe)
0xE000_0000	Fixed I/O Peripherals
0xF800_0000	Fixed Internal Peripherals (Timers, Watchdog, DMA, Interconnect)
0xFC00_0000	Flash Memory
0xFFFC_0000	On-Chip Memory

XILINX.

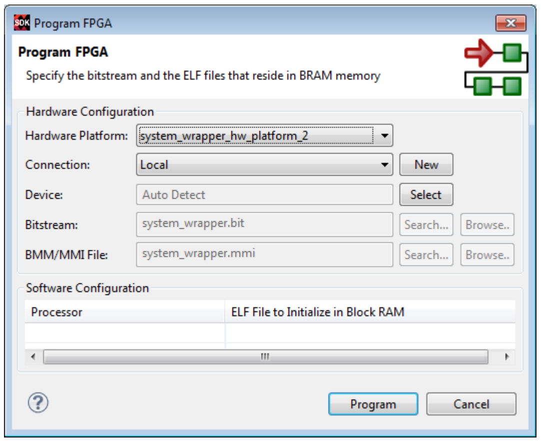
Configuring the PL through SDK

> **Download the bitstream and then the application**

  - Select **Xilinx Tools > Program FPGA**
  - Locate and select the hardware bit file
  - No BMM file as the ELF runs in PS DDR , QSPI, or OCM memory
  - Click **Program**

> **The programmable logic configures**

> **When launched (later), the XMD debugger will halt the processor and load the actual application for debugging**



Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Platform: system_wrapper_hw_platform_2

Connection: Local [New]

Device: Auto Detect [Select]

Bitstream: system_wrapper.bit [Search...] [Browse..]

BMM/MMI File: system_wrapper.mmi [Search...] [Browse..]

Software Configuration

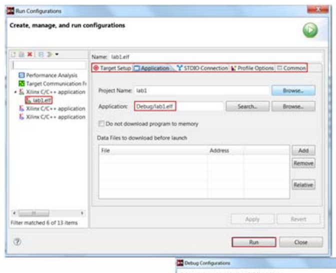
Processor	ELF File to Initialize in Block RAM

[Program] [Cancel]

XILINX.

Software Loading and Debugging

- **Run configurations through SDK**
  - Select software application in the project explorer pane
  - Select **Run As > Run Configurations**
  - Double-click **Xilinx C/C++ ELF** to create a run configuration for application
  - Click **Run** to download the executable (.elf) and run the application
- **Debug configurations through SDK**
  - Select software application in the project explorer pane
  - Select **Debug As > Debug Configurations**
  - Double-click **Xilinx C/C++ ELF** to create a debug configuration for application
  - Click **Debug** to download the executable (.elf), and suspend at the entry point



Run Configurations

Create, manage, and run configurations

Performance Analysis
Target Communication Fr
Xilinx C/C++ application
lab1.elf
Xilinx C/C++ application
Xilinx C/C++ application

Filter matched 6 of 13 items

Name: lab1.elf

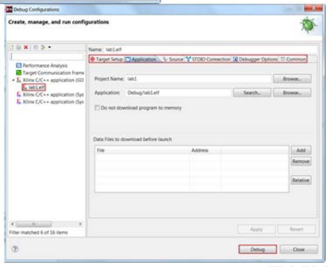
Project Name: lab1

Application: Debug/lab1.elf

Do not download program to memory

Data Files to download before launch

File	Address



Debug Configurations

Create, manage, and run configurations

Performance Analysis
Target Communication Frame
Xilinx C/C++ application (GD
lab1.elf
Xilinx C/C++ application (Sys
Xilinx C/C++ application (Sys

Filter matched 6 of 16 items

Name: lab1.elf

Project Name: lab1

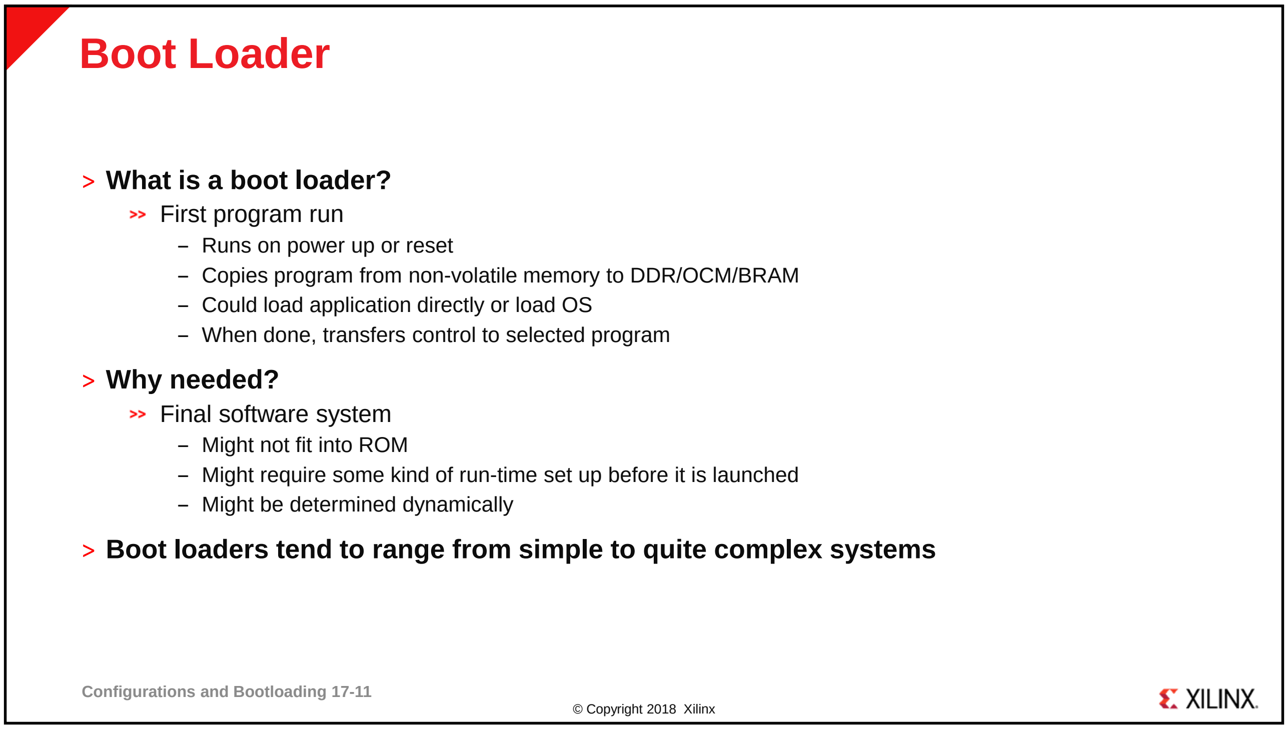
Application: Debug/lab1.elf

Do not download program to memory

Data Files to download before launch

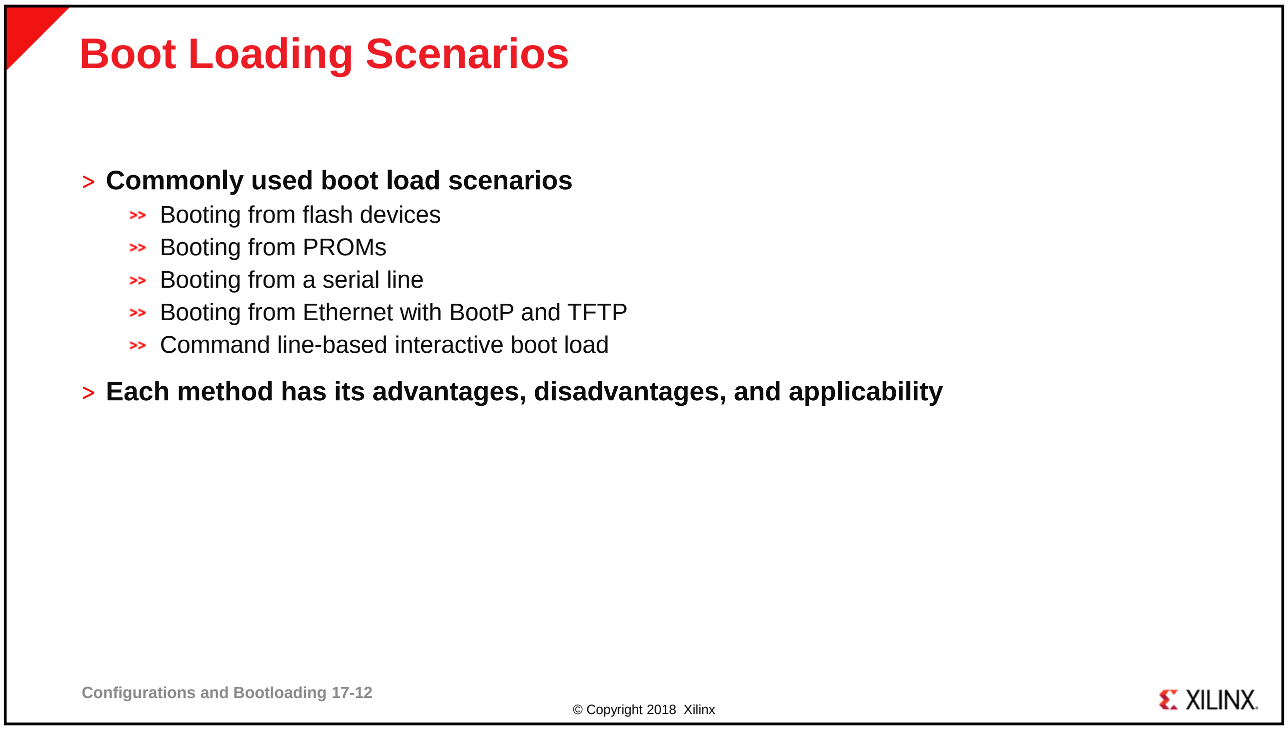
File	Address

Boot Loader



Boot Loader

- What is a boot loader?
  - First program run
    - Runs on power up or reset
    - Copies program from non-volatile memory to DDR/OCM/BRAM
    - Could load application directly or load OS
    - When done, transfers control to selected program
- Why needed?
  - Final software system
    - Might not fit into ROM
    - Might require some kind of run-time set up before it is launched
    - Might be determined dynamically
- Boot loaders tend to range from simple to quite complex systems



Boot Loading Scenarios

- Commonly used boot load scenarios
  - Booting from flash devices
  - Booting from PROMs
  - Booting from a serial line
  - Booting from Ethernet with BootP and TFTP
  - Command line-based interactive boot load
- Each method has its advantages, disadvantages, and applicability

Image Formats

- **Boot loader must understand both**
  - Image format of the file (application, bitstream, or data), and
  - Organization of the images in the Non-Volatile storage medium
- **Formats**
  - Common: ELF, Intel MCS-86 file (.mcs),binary(.BIN), Motorola SREC, Intel I-hex, gzip/bzipped images
  - Less common: Custom formats are common as well
- **Image formats have different processing complexities and sizes**
  - ELF, SREC/iHex, binary, compressed
    - Decreasing order of size requirements
  - Compressed, ELF, SREC/iHex, binary
    - Decreasing order of processing complexity

Stage 0: ROM

- **Processor boots from boot ROM (128KB)**
  - Xilinx provided
  - Not viewable
- **Copies First Stage Boot Loader (FSBL) from memory device to OCM static RAM (256KB)**
  - Maximum size is 192KB (rest can be used as stack, BSS, or non-initialized memory)
  - Xilinx provided
- **Once copied, the FSBL starts executing (from OCM RAM)**

First Stage Boot Loader (FSBL)

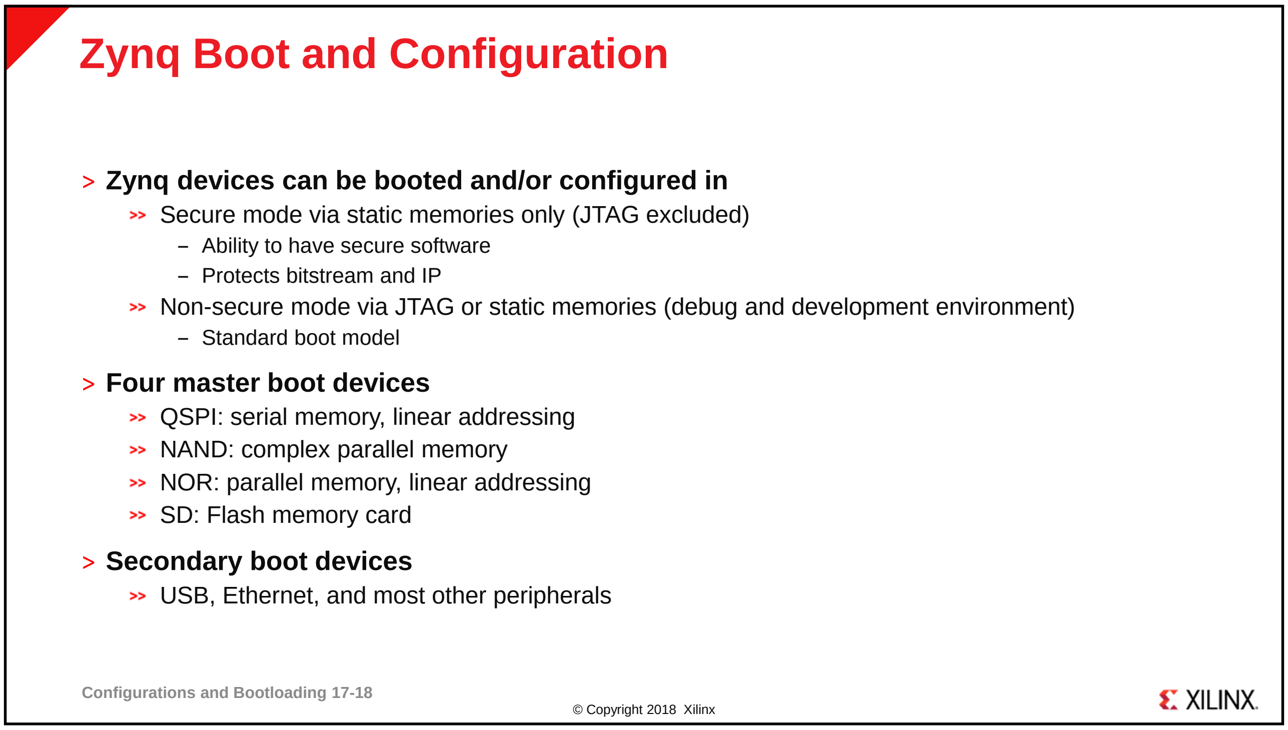
- Example FSBL provided by Xilinx as an SDK example project
  - Otherwise user developed
- Copies next stage of code into
  - DDRx or static memory (OCM)
  - And/or enables an external device for Stage 2
- Further initialization of PS components and peripherals
- Optionally configures programmable logic
- Upon completion, launches application or Second Stage Boot Load

Second Stage Boot Loader (SSBL)

- Example U-Boot provided by Xilinx
  - git://git.xilinx.com/u-boot-xlnx.git
  - Otherwise user developed
- Loaded from user-selected external device
- Flexibility in boot sources
  - Static memory
  - Dynamic memory
  - PS peripherals such as
    - USB, Ethernet, or SD
  - Programmable logic I/Os
- Initializes rest of PS
- Optionally configures PL

Zynq PS Boot and PL Configuration

XILINX



Zynq Boot and Configuration

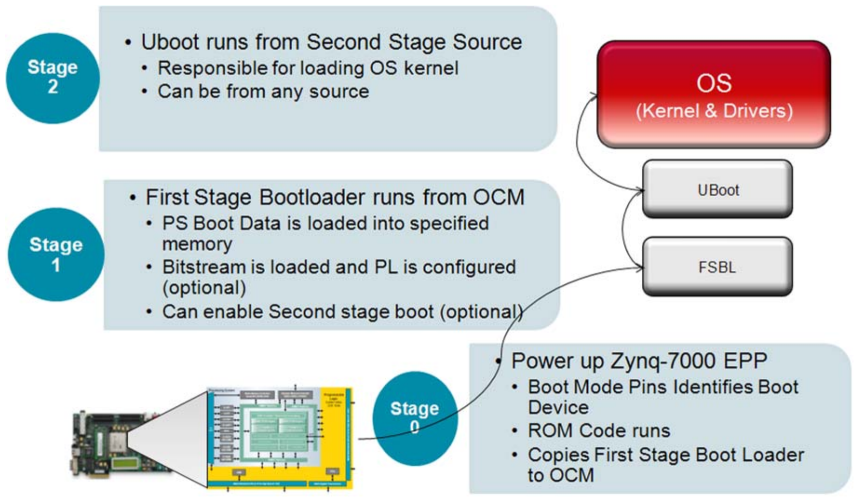
> **Zynq devices can be booted and/or configured in**
>> Secure mode via static memories only (JTAG excluded)
>>- Ability to have secure software
>>- Protects bitstream and IP
>
>> Non-secure mode via JTAG or static memories (debug and development environment)
>>- Standard boot model

> **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

XILINX

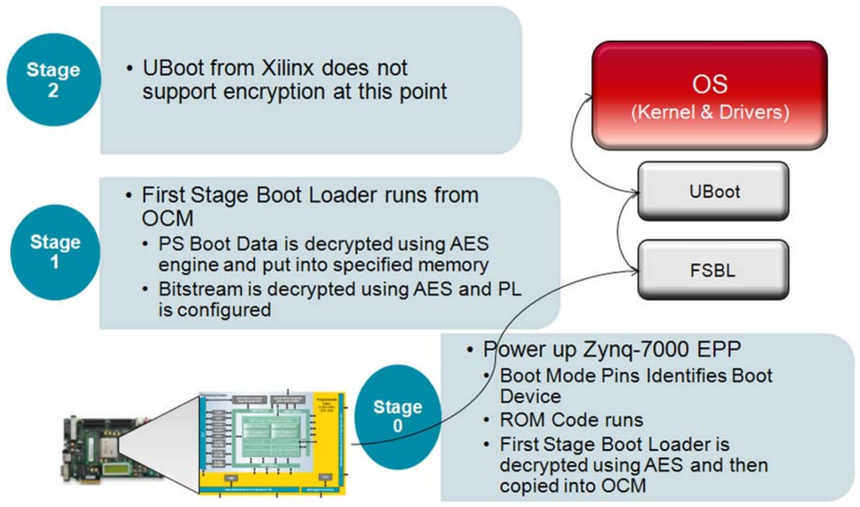
Non-Secure Boot Example



- Uboot runs from Second Stage Source
  - Responsible for loading OS kernel
  - Can be from any source
- First Stage Bootloader runs from OCM
  - PS Boot Data is loaded into specified memory
  - Bitstream is loaded and PL is configured (optional)
  - Can enable Second stage boot (optional)
- Power up Zynq-7000 EPP
  - Boot Mode Pins Identifies Boot Device
  - ROM Code runs
  - Copies First Stage Boot Loader to OCM

XILINX.

Secure Boot Example



- UBoot from Xilinx does not support encryption at this point
- First Stage Boot Loader runs from OCM
  - PS Boot Data is decrypted using AES engine and put into specified memory
  - Bitstream is decrypted using AES and PL is configured
- Power up Zynq-7000 EPP
  - Boot Mode Pins Identifies Boot Device
  - ROM Code runs
  - First Stage Boot Loader is decrypted using AES and then copied into OCM

XILINX.

Secure Linux Boot Example

Stage 2

- User developed code can be secure

uBoot For Linux

Stage 1

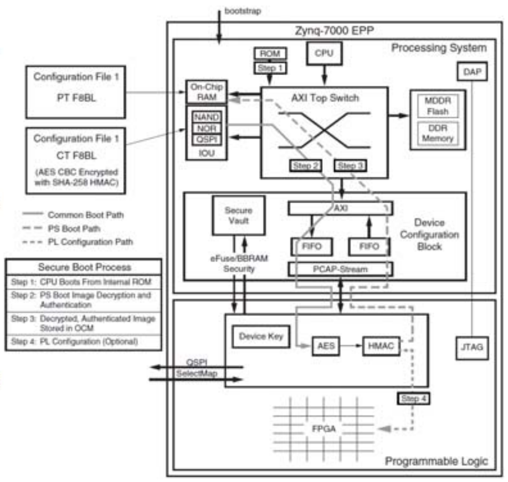
- First Stage Boot Loader runs from OCM
  - Decrypts and authenticates PS Boot Data using AES/SHA engine and puts into specified memory OR
  - Enables Second stage boot (optional)
  - Decrypts and authenticates Bitstream using AES/SHA and configures PL (optional)

First Stage Boot Loader

Stage 0

- Power up Zynq-7000 EPP
  - Boot Mode Pins Identifies Boot Device
  - ROM Code runs
  - Decrypts and authenticates FSBL using AES/SHA and then copied into OCM (PL powered on)

Boot Mode Selection



Secure Boot Process
Step 1: CPU Boots From Internal ROM
Step 2: PS Boot Image Decryption and Authentication
Step 3: Decrypted, Authenticated Image Stored in OCM
Step 4: PL Configuration (Optional)

XILINX.

PCW Boot Configuration Handoff

> **The Zynq SoC is a processor first, programmable logic second SoC**

> **Most options, features, and configurations are controlled by software setup**

- Clock generation
- MIO usage
- Processor cache and DDR memory configuration

> **The Vivado Export to SDK which generates the PS configuration code**

- Used by FSBL
- ps7_init.*c* , ps7_init.tcl

XILINX.

PL Device Configuration Services

- **Used by FSBL**
- **Set of Standalone library services**
- **The device configuration interface has three main functionalities**
  - AXI-PCAP
  - Security policy
  - System monitor
    - Currently not implemented
- **Supports the downloading of the programmable logic bitstream and readback of the decrypted image**
- **Services are detailed in the *Software Developers Guide* (UG821)**

SDK FSBL Support

- **SDK software project**
- **Complete FSBL boot application**
  - Software application load
  - PL configuration from bit file
  - Support for golden image
- **Requires **.bif* file for image generation**
- **All source code is included**
  - Can be modified for other boot sources
    - Ethernet
    - USB
    - Serial

Creating a Single Boot Image File

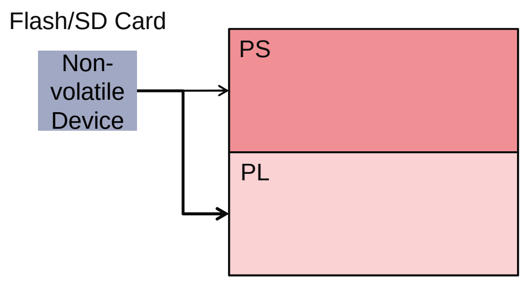
> **Select Xilinx Tools > Create Zynq Boot Image**
  - (bootgen)
  - Add the FSBL ELF file
  - Add the PL bitstream file (optional, only if the PL resources are used)
  - Add the software application ELF file

> **Select the output file directory**

> **Click Create to create the image file**
  - *.bin for booting the application from SD card
  - Rename it to BOOT.bin before placing it on the SD card

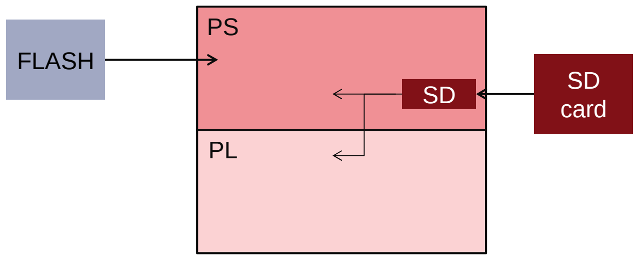
> **Creates intermediate/other boot image format file**

PS Boot and PL Configuration Example-1



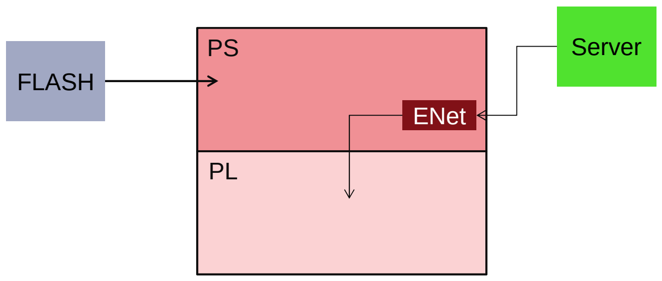
User brings everything from non-volatile memory:
1. PS Initialization code
2. Operating System
3. PL Bitstream

PS Boot and PL Configuration Example-2



User brings:
1. PS Initialization code from FLASH to configure SDIO
2. Operating System from SD card
3. PL Bitstream from SD card

PS Boot and PL Configuration Example-3

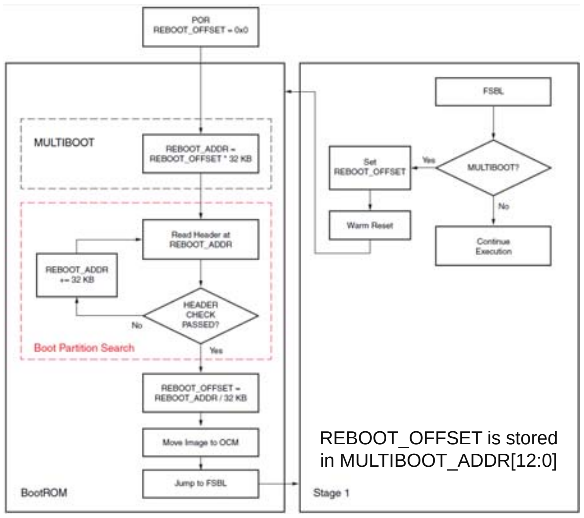


User brings:
1. PS Initialization code from FLASH to configure Ethernet
2. Operating System from server through Ethernet
3. PL Bitstream from server

Boot ROM Multi Boot

> **Multiple images must be placed in flash**
  - Each image requires a boot ROM header.
  - These images can be placed in any order, but after POR the boot ROM uses the first image in flash as the initial image

> **Procedure of using multiboot**
  - 1. Calculate the REBOOT_OFFSET address:
    - a. This offset is relative to the beginning of flash
    - b. REBOOT_OFFSET can be calculated as: Image Byte Address in Flash / 0x8000
  - 2. Write REBOOT_OFFSET to MULTIBOOT_ADDR[12:0].
  - 3. Perform a warm/soft reset.



REBOOT_OFFSET is stored in MULTIBOOT_ADDR[12:0]

Configuring and Re-Configuring

> **DevC - The PL is configured via the device configuration interface module**

> **Accessed via a software application using an AXI port in the PS**
  - Supported by Xilinx-provided APIs in SDK
  - Recommended methodology

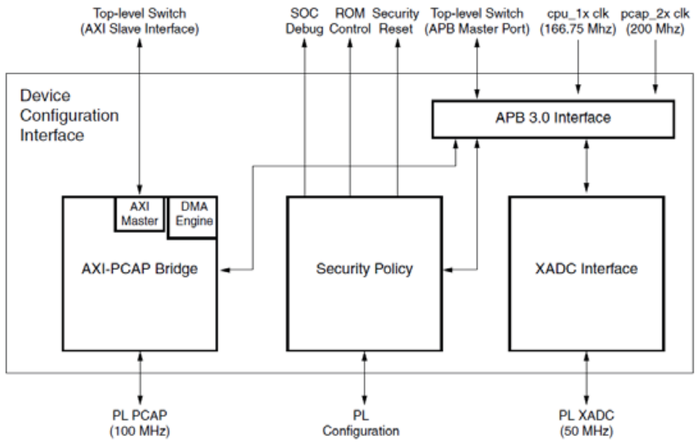
> **Separate DMA port into the Central interconnect for simultaneous PL configuration with software download**

> **Accessed from the PL via a GPx master AXI port**
  - Not recommended

Device Configuration (DevC) Interface

> **Three main blocks operate independently**
  - An AXI-PCAP bridge for interfacing to the PL configuration logic
  - Device security management
  - An XADC interface

> **Also contains an APB interface used by host to configure the three blocks, to access the overall status, and to communicate with the PL XADC**

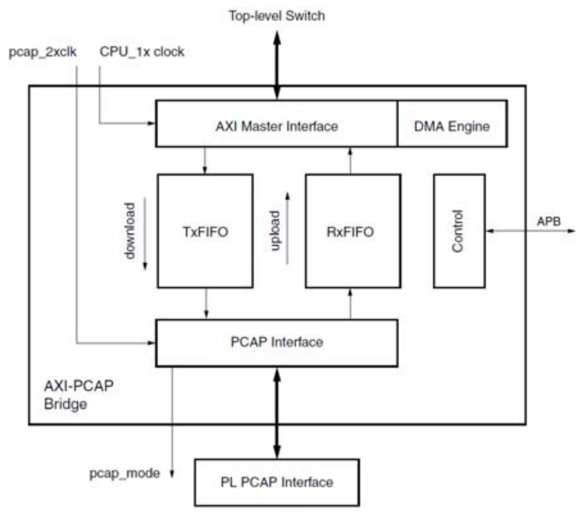


The DevC Interface

> **Manages basic device security and provides a simple DMA interface, PS setup, and PL configuration**
  - Enables PL configuration through the processor configuration access port (PCAP) in both secure and non-secure master boot, including support for compressed PL bitstreams
  - Supports PL configuration readback
  - Supports concurrent bitstream download/upload
  - Enforces Zynq-7000 device system-level security including debug security
  - Supports XADC serial interface
  - Supports XADC alarm and over-temperature interrupt
  - Secure boot ROM code protection

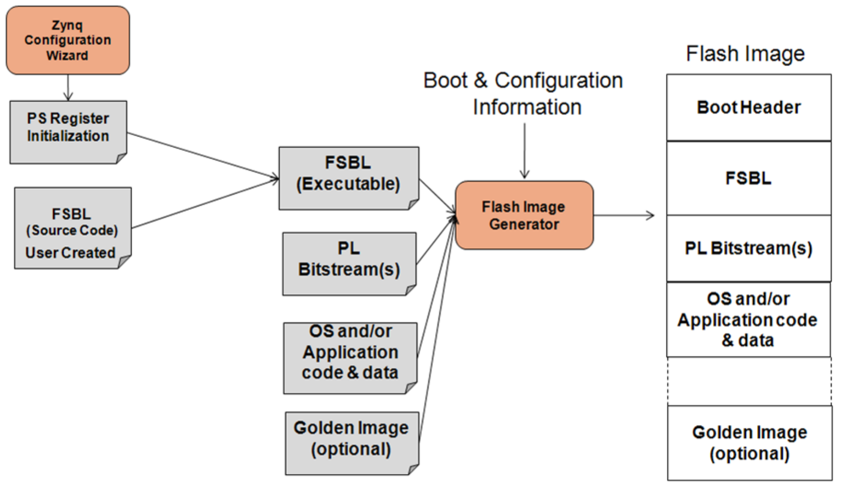
AXI-PCAP Bridge

- Converts 32-bit AXI formatted data to the 32-bit PCAP protocol and vice versa
- Supports both concurrent and non-concurrent download and upload of configuration data
- The DMA engine moves data between the FIFOs and a memory device, typically the on-chip RAM, the DDR memory, or one of the peripheral memories
- Non-secure data to the PCAP interface can be sent every clock cycle, encrypted data can be sent every four clock cycles



Flash Programmer Utility

Flash Image Generation


Zynq Configuration Wizard
PS Register Initialization
FSBL (Source Code) User Created
FSBL (Executable)
PL Bitstream(s)
OS and/or Application code & data
Golden Image (optional)
Boot & Configuration Information
Flash Image Generator
Flash Image
Boot Header
FSBL
PL Bitstream(s)
OS and/or Application code & data
Golden Image (optional)

Flash Programming Procedure

> **Use of SDK to program flash**

- Assign flash image to flash memory attached to the Zynq device
- Connect to the JTAG chain containing the ARM DAP of the Zynq device
- Download the flash programming application into the PS of the Zynq device

> **Procedure**

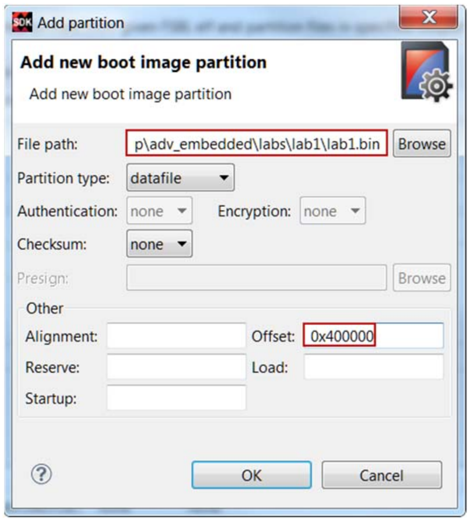
- Erase: Erase flash memory
- Program: Loads the flash image into the PS buffer of the Zynq device; Zynq device application writes the image to flash memory
- Verify: Zynq device reads the flash contents and writes into buffer; reads the buffer and compares against the original flash image

Creating FSBL

- **SDK provides an FSBL software project template**
  - Target application must be in Zynq device FSBL format in flash
  - RAM must exist at location targeted by flash image format
  - Selection of target hardware processor(s)
- **FSBL**
  - May configure the programmable logic with hardware bitstream
  - May load OS image or standalone image or SSBL image from the non-volatile memory to RAM (DDR)
  - Transfers program control to the newly loaded application/OS
- **Xilinx FSBL supports multiple partitions; each partition can be a code image and/or a bitstream**

Creating Image for Multi-Boot

- **Multi-boot application will have more than one full image**
  - Each full image may consist of
    - FSBL, hardware bitstream (if needed), an application
- **Create the full image having multi-boot application using SDK's Create Zynq Image**
  - First partition will consist of boot image of the default application at offset 0
  - It may have
    - FSBL, hardware bitstream (if needed), an application
  - Second and subsequent partitions should be stored at multiple of 0x40000 offset
- **Use Flash Programming utility (see next slide)**


Add partition

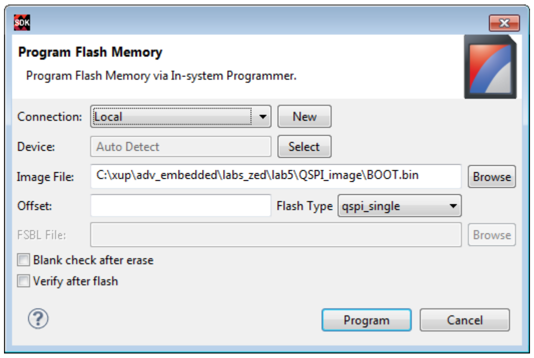
Add new boot image partition

Add new boot image partition

File path:	p\adv_embedded\labs\lab1\lab1.bin
Partition type:	datafile
Authentication:	none
Encryption:	none
Checksum:	none
Presign:	
Alignment:	
Offset:	0x400000
Reserve:	
Load:	
Startup:	

Flash Programming Utility

- Select Xilinx Tools > Program Flash
- Select the image file and offset
  - A full flash image offset will always be 0
- Click Program to "flash" the image


Program Flash Memory

Program Flash Memory via In-system Programmer.

Connection: Local

Device: Auto Detect

Image File: C:\xup\adv_embedded\labs_zed\lab5\QSPI_image\BOOT.bin

Offset:

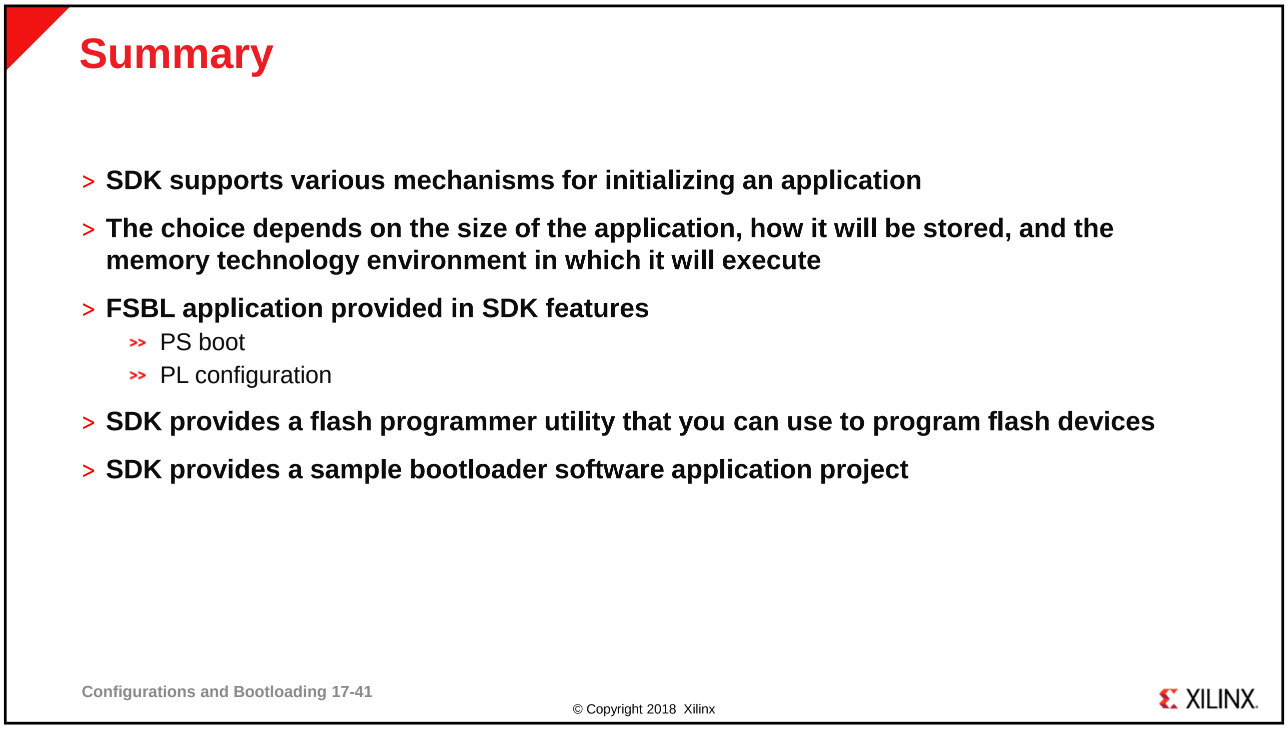
Flash Type: qspi_single

FSBL File:

☐ Blank check after erase

☐ Verify after flash

Summary



Summary

- SDK supports various mechanisms for initializing an application
- The choice depends on the size of the application, how it will be stored, and the memory technology environment in which it will execute
- FSBL application provided in SDK features
  - PS boot
  - PL configuration
- SDK provides a flash programmer utility that you can use to program flash devices
- SDK provides a sample bootloader software application project

XILINX.

Adaptable.
Intelligent.

XILINX.