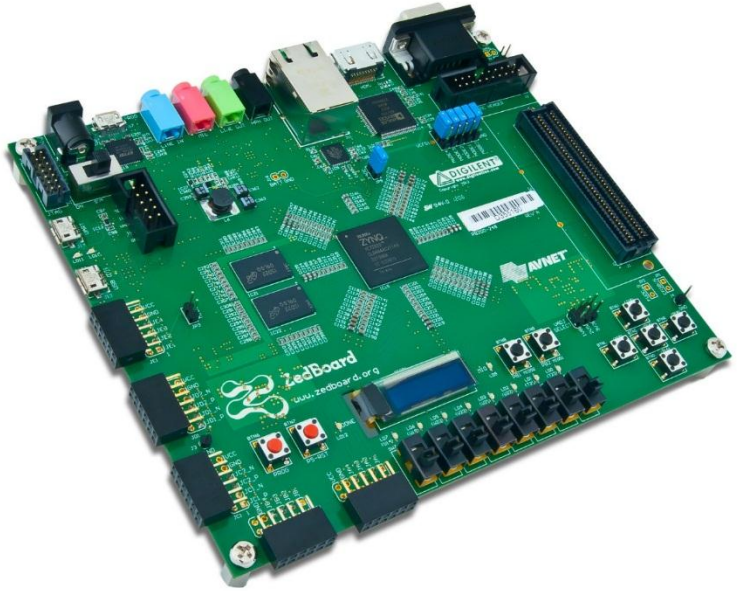
Digitronix Nepal
Innovation from Engineers...

Embedded System Design with Xilinx VIVADO & Zynq FPGA



Course Prepared by

Digitronix Nepal

XILINX
ALL PROGRAMMABLE™

VIVADO™

Image source: Xilinx.com

Digitronix Nepal
Innovation from Engineers...

Sections of the Course:

Section 1. Overview with Xilinx VIVADO Design Suit and Zynq FPGA (ZedBord)
Section 2. Basic Embedded System Architecture and Design
Section 3. Creating Custom IP in VHDL/Verilog for Zynq (Zedboard)
Section 4. Software Development with Xilinx SDK for Embedded System
Section 5. Software Writing for Timer and Debugging with SDK
Section 6. Configuration and Bootloading with Zynq

Digitronix Nepal
Innovation from Engineers...

Section and Lecturers of the Course:

Section 6. Configuration and Bootloading with Zynq

Lecture 1 : Overview of Configuration and Bootloading of Embedded Applications

Lecture 2 : Lab 6 - Create images to boot off the SD card and QSPI flash. Load previously generated hardware bitstreams and executable and execute desired application.

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

Lecture 1 : Configuration and Bootloading in Zynq

Outline

- *Introduction*
- Boot Loader
- Zynq PS Boot and PL Configuration
- Flash Programmer Utility
- Summary

Reference: Xilinx.com

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 AP 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 AP 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

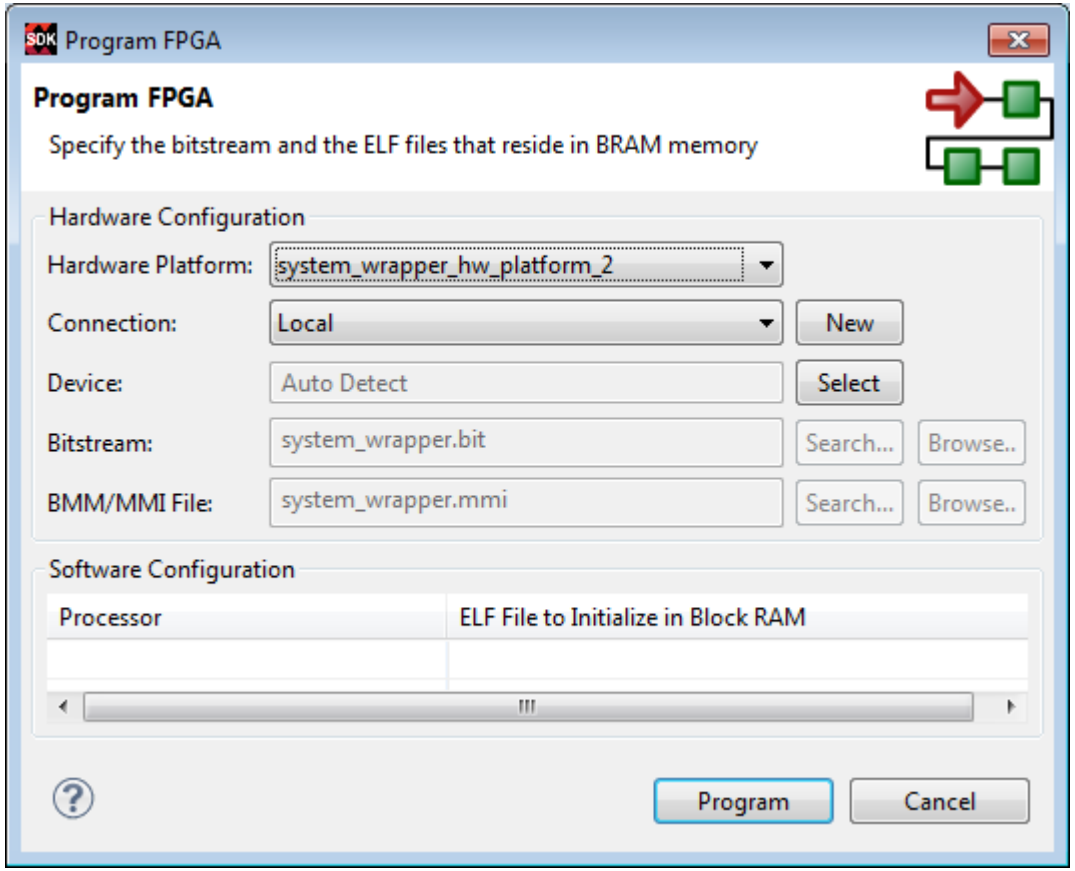
Cortex-A9 Processor Memory Space

- 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

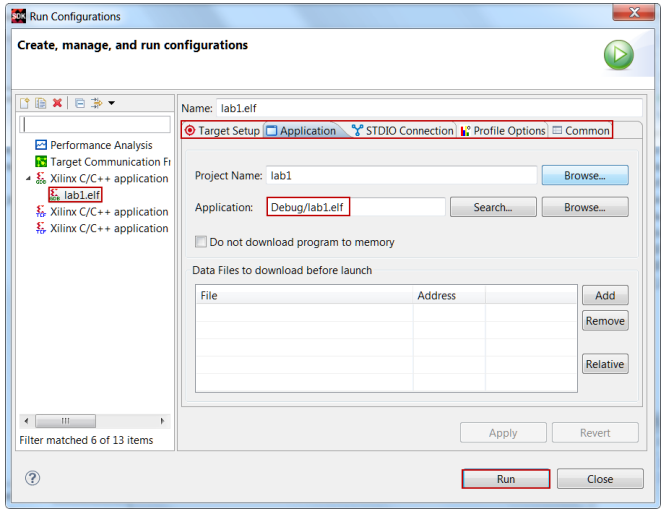
Configuring the PL through SDK

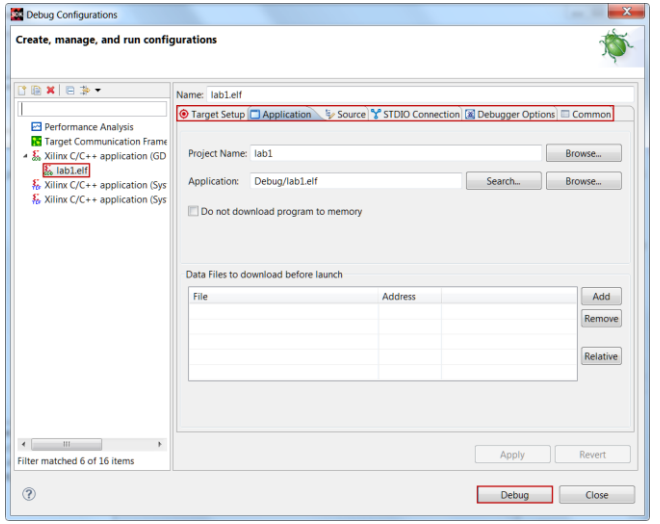
- Download the bitstream and then the application
  - Select **Xilinx Tools > Program FPGA**
  - Locate and select the hardware bit file
  - No BMM file as the ELF runs in PS DDR , QSPI, or OCM memory
  - Click **Program**
- The programmable logic configures
- When launched (later), the XMD debugger will halt the processor and load the actual application for debugging



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





Outline

- Introduction
- *Boot Loader*
- Zynq PS Boot and PL Configuration
- Flash Programmer Utility
- Summary

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

Outline

- Introduction
- Boot Loader
- *Zynq PS Boot and PL Configuration*
- Flash Programmer Utility
- Summary

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

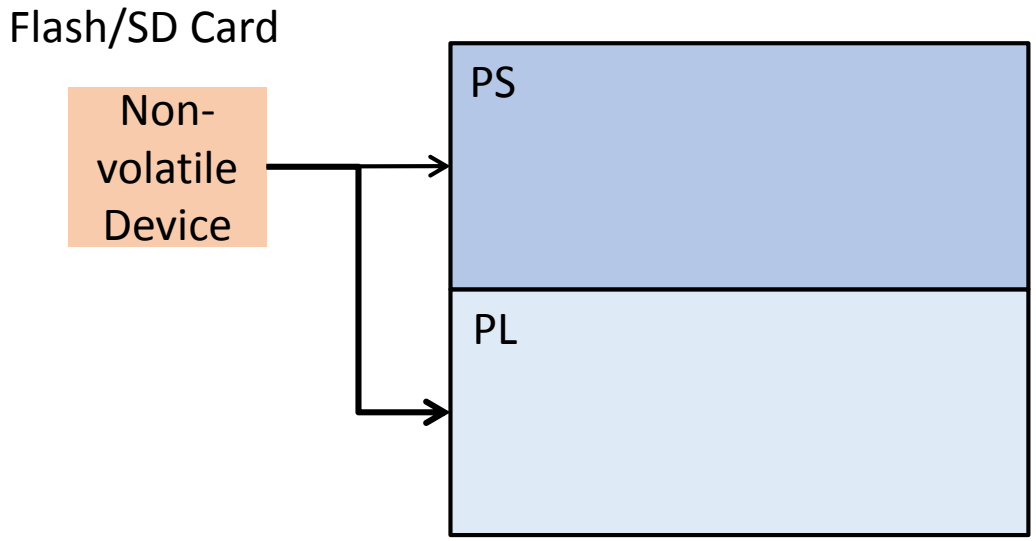
SDK FSBL Support

- 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

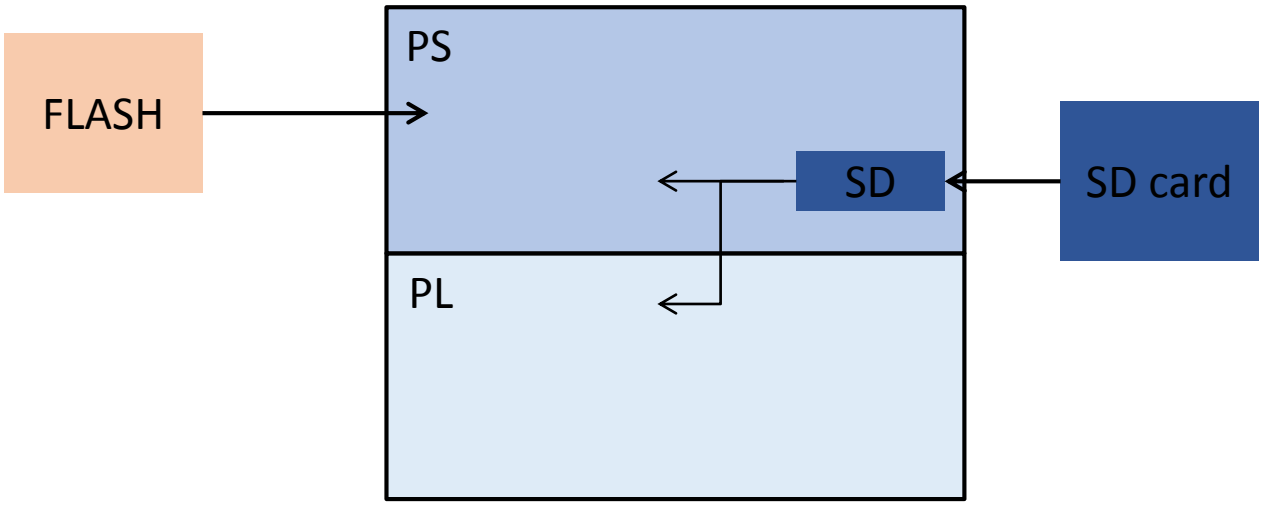
PS Boot and PL Configuration Example-1



User brings everything from non-volatile memory:

1. PS Initialization code
2. Operating System
3. PL Bitstream

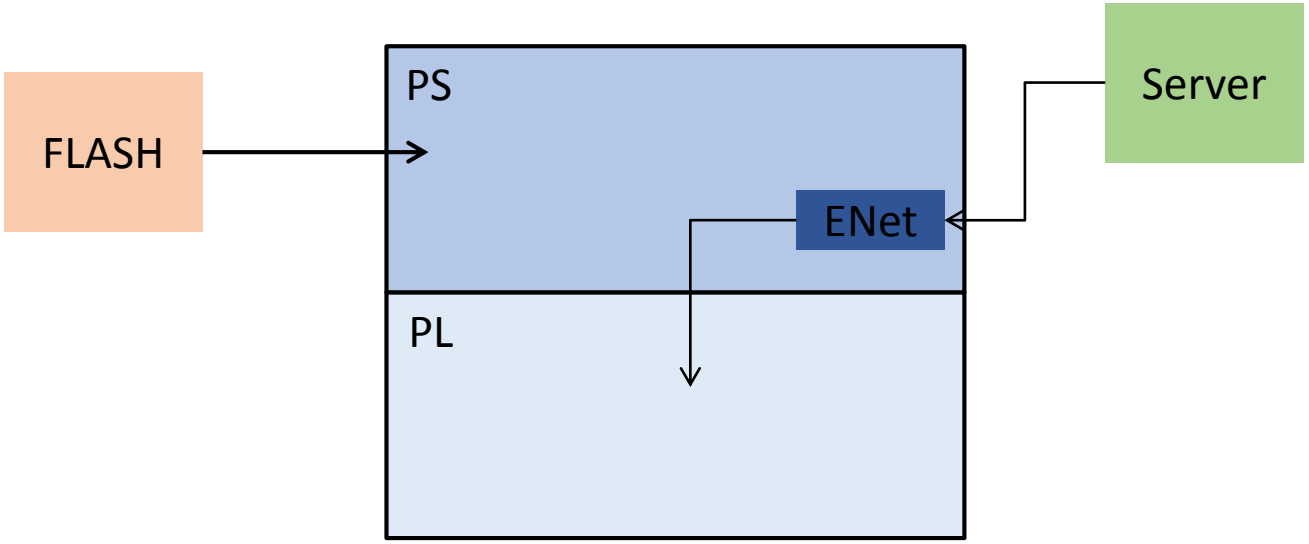
PS Boot and PL Configuration Example-2



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



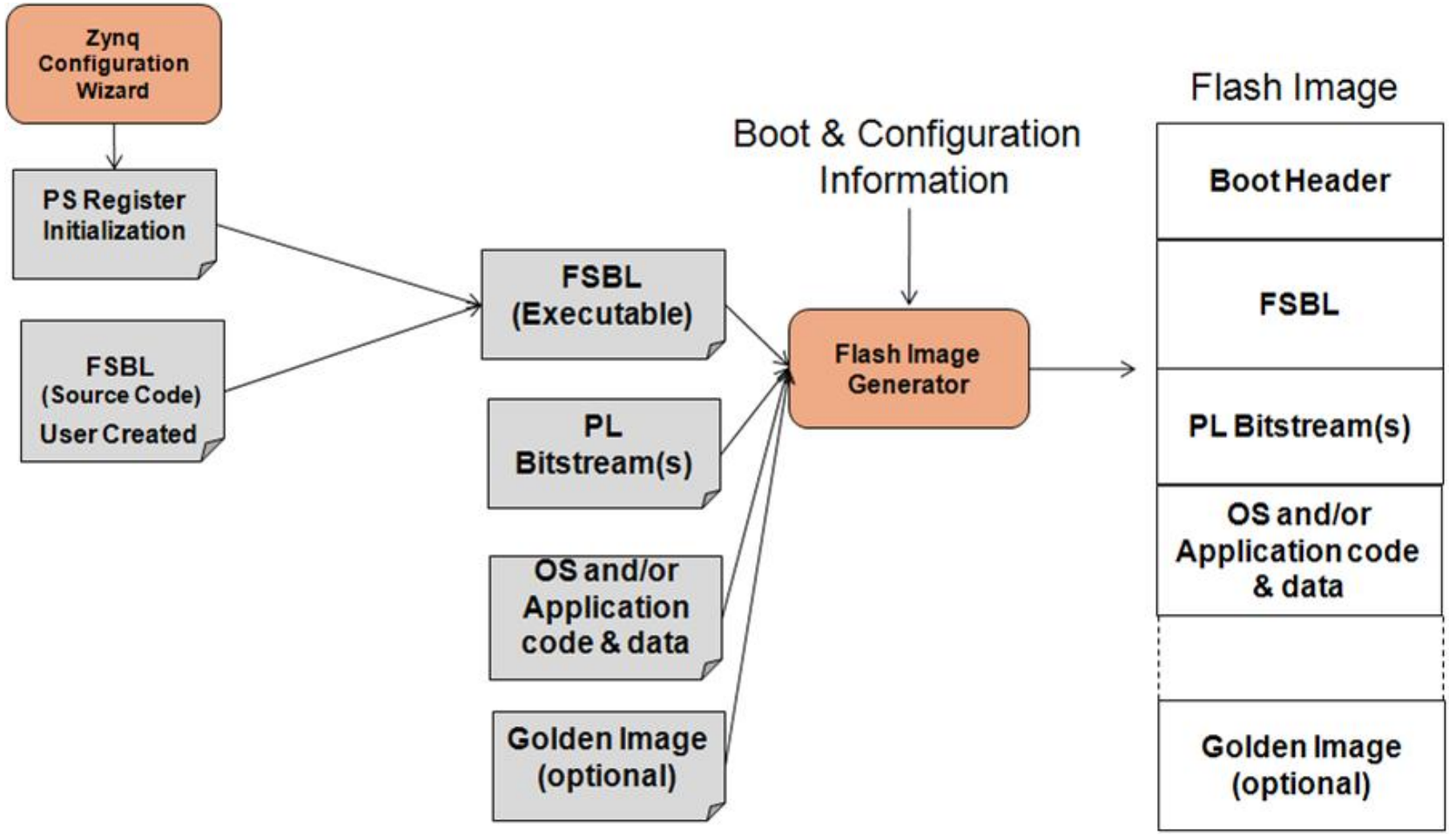
User brings:

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

Outline

- Introduction
- Boot Loader
- Zynq PS Boot and PL Configuration
- *Flash Programmer Utility*
- Summary

Flash Image Generation



Reference: Xilinx.com

Flash Programming Procedure

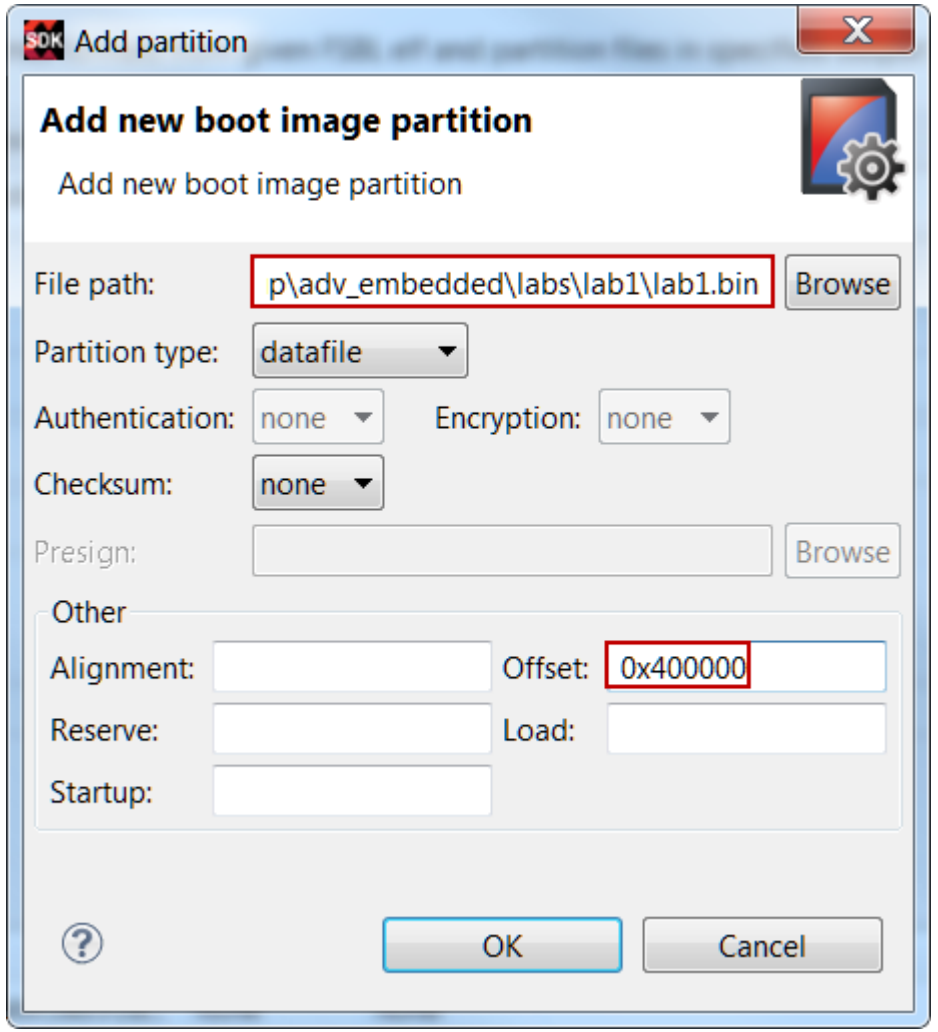
- Use of SDK to program flash
  - Assign flash image to flash memory attached to the Zynq device
  - Connect to the JTAG chain containing the ARM DAP of the Zynq device
  - Download the flash programming application into the PS of the Zynq device
- Procedure
  - Erase: Erase flash memory
  - Program: 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

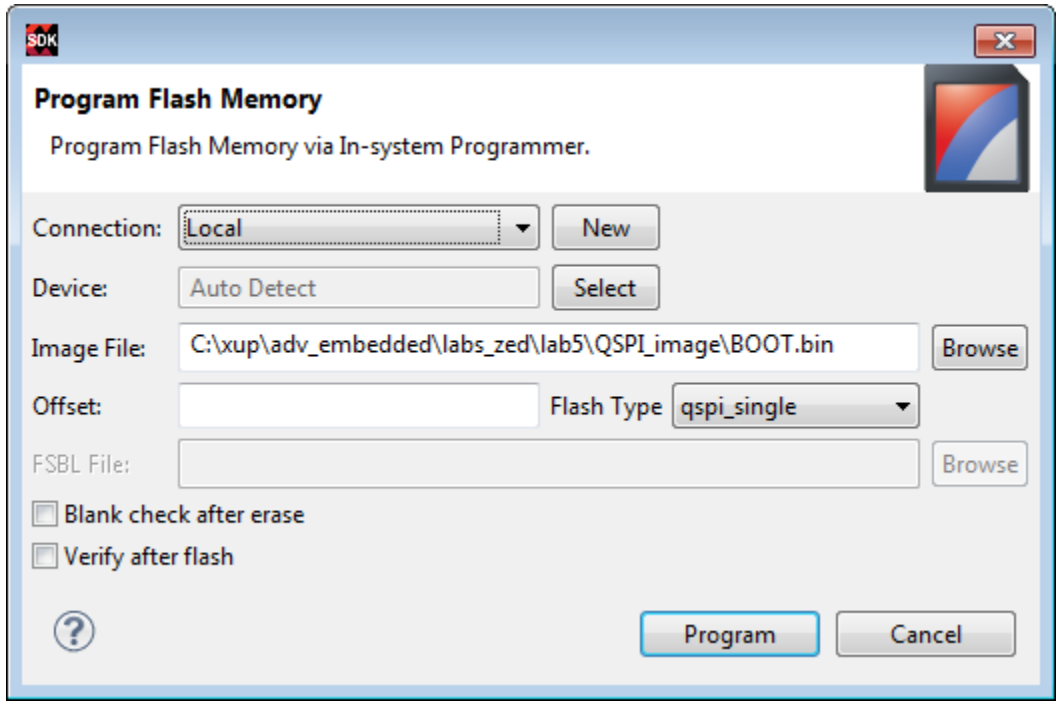
- Multi-boot application will have more than one full image
  - Each full image may consist of
    - FSBL, hardware bitstream (if needed), an application
- Create the full image having multi-boot application using SDK's Create Zynq Image
  - First partition will consist of boot image of the default application 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)



Reference: Xilinx.com

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



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

Lecture 2 : Lab 6 - Create images to boot off the SD card and QSPI flash.

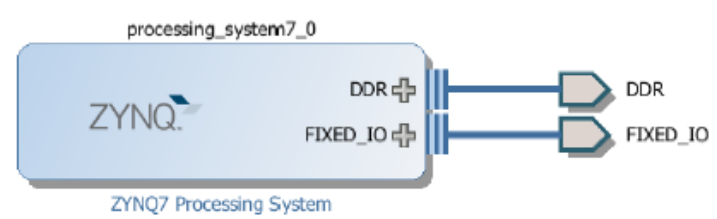
- Create a bootable system capable of booting from the SD card

- Create a bootable system capable of booting from the QSPI flash
- Load the bitstream stored on the SD card or in the QSPI flash memory
- Configure the PL section using the stored bitstream through the PCAP resource
- Execute the corresponding application

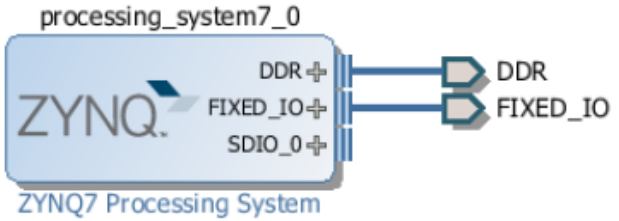
Final Block Design: ARM Cortex-A9 based Embedded System Design Configuration and Booting

Design Flow:

1. Create a Zynq PS block Design in VIVADO IPI
2. Customized the Zynq PS
3. Create HDL wrapper
4. Generate project and export hardware
5. Launch SDK
6. Create Hello World Application
7. Create another FSBL Project
8. Click on Hello world BSP
9. Go to Xilinx Tools→ create Zynq boot image
10. It creates boot.bin
11. Send that boot.bin to SD card (fat32 filesystem)
12. Insert on Zynq (Zedboard/Zybo)
13. Start the terminal of SDK :
14. Hello World will display on terminal via UART.



(a) ZedBoard



(b) Zybo

After Creating Image File of the Hello World Application with FSBL

15. Insert a blank SD/MicroSD card (FAT32 formatted) in a Card reader, and using the Windows Explorer, copy the BOOT.bin file from the **image** folder in to the SD/MicroSD card.

16. Set the board in the SD card boot mode (For Zedboard, set the mode pins JP7-JP11 as GNDSIG, GND-SIG, SIG-3V3, SIG-3V3, GND-SIG (right-to-left), and for Zybo set the JP5 jumper to SD).

17. Insert the SD/MicroSD card into the board.

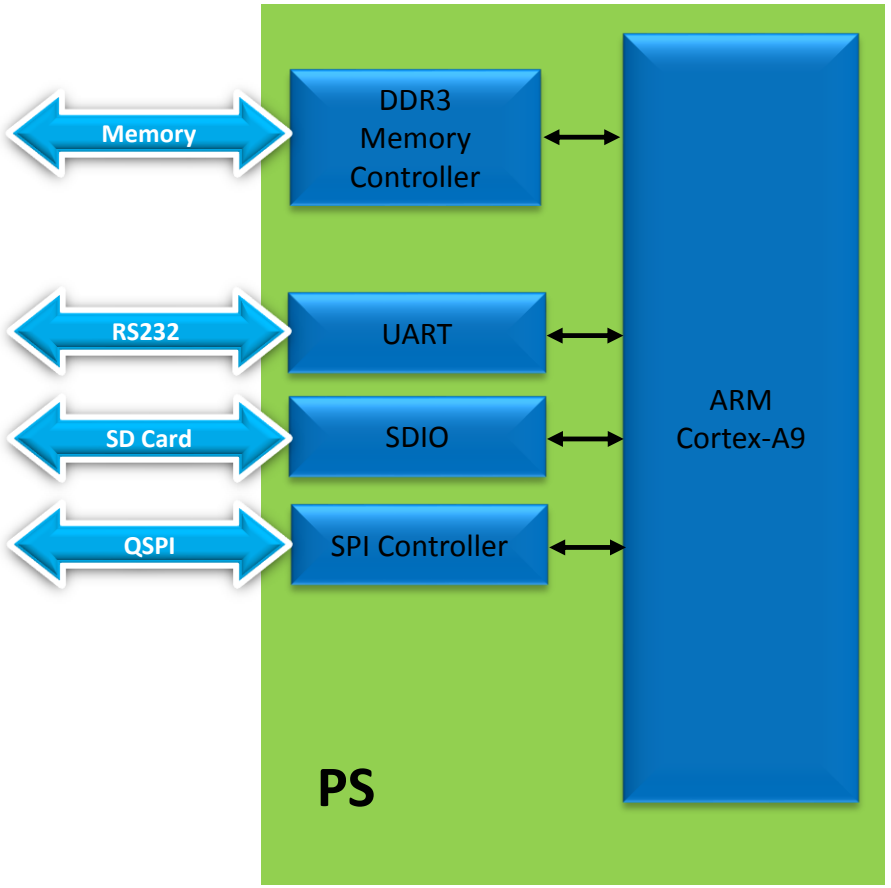
18. Power ON the board.

19. Connect your PC to the UART port with the provided micro-USB cable, and start a Terminal

20. emulator program setting it to the current COM port and 115200 baud rate.

21. You should see the **Hello World** message in the terminal emulator window.

Final Block Design: ARM Cortex-A9 based Embedded System Design Configuration and Booting



Lets start at Vivado and SDK

Thank You!