

Developing Zynq Software

**With Xilinx
Software Development Kit 2013.3**



Accelerating Your Success™

SDK Application Development Flow



Generate Hardware Platform (**XML**), **PS7_INIT** and **Bitstream**

HW Platform



SDK

Import HW Platform

Generate **Board Support Package**

Libraries and Drivers

Develop Application

Build

Run on HW

Debug and Profile

Works?

No

Yes

Create Boot Image

Program Flash

Archive Project

Run on HW

Today's Focus

Objectives

- **Introduce developers to Xilinx SDK**
- **Explore how SDK makes your job easier**
- **Connect SDK to hardware for execution and debug**
- **Show a basic example of using both processors**
- **Utilize a peripheral interrupt to show real-time software response**

Agenda

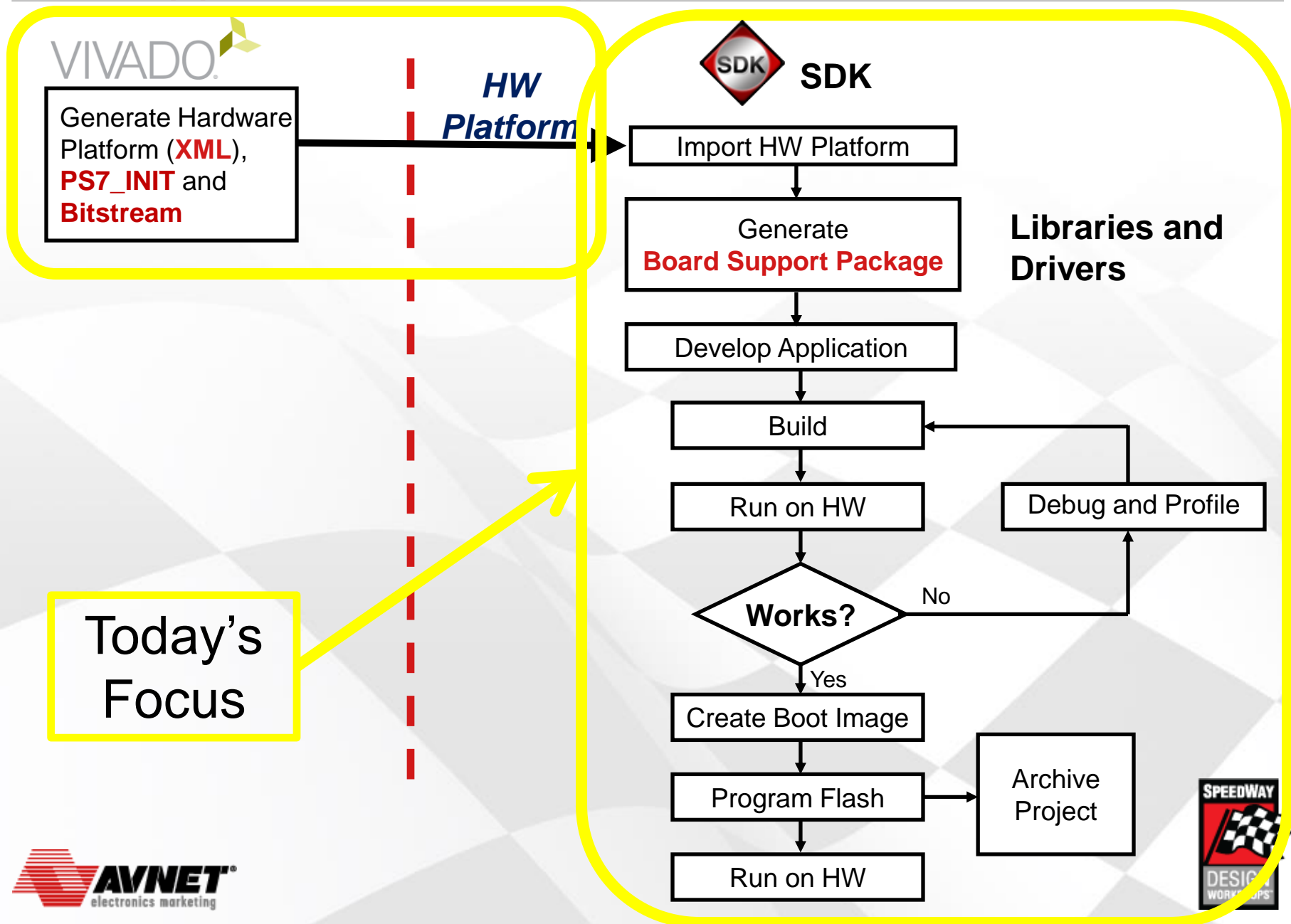
....

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

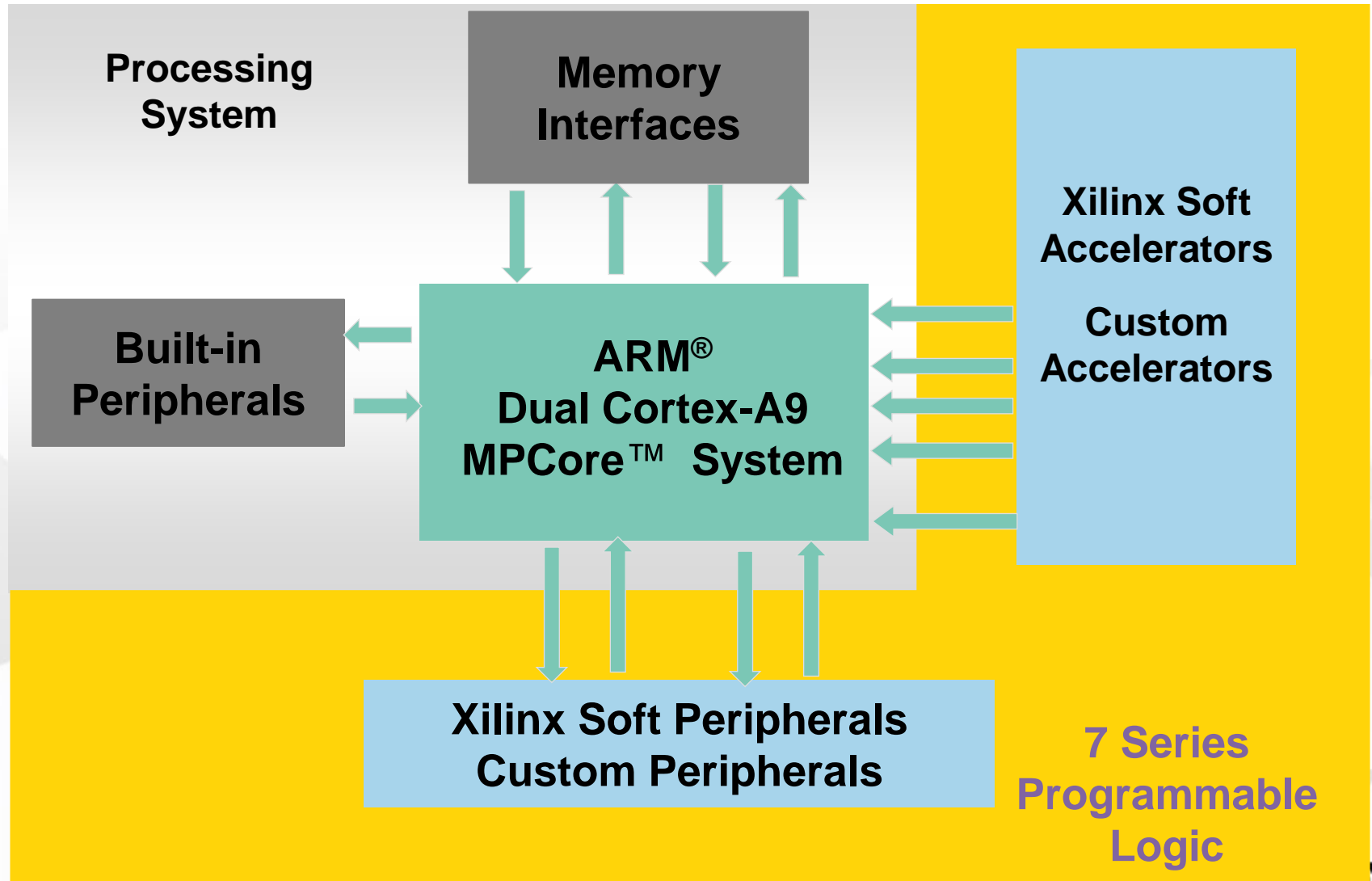
Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

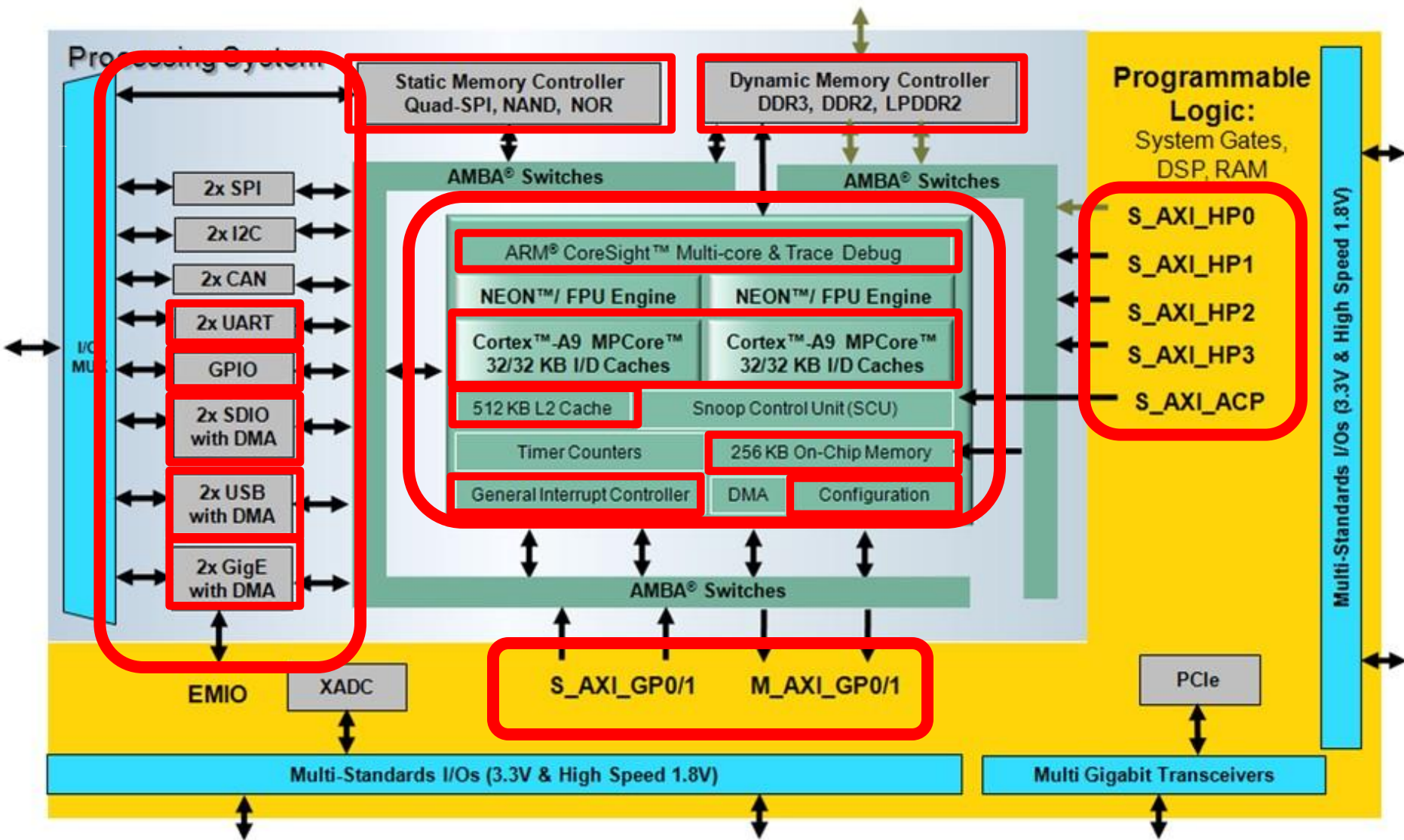
SDK Application Development Flow



Zynq-7000 AP SoC Basic Architecture



Zynq-7000 AP SoC Block Diagram



Zynq-7000 Device Portfolio Summary

Zynq-7000 AP SoC Devices		Z-7010	Z-7015	Z-7020	Z-7030	Z-7045	Z-7100
Processing System	Processor Core			Dual ARM® Cortex™-A9 MPCore™			
	Processor Extensions		NEON™ & Single / Double Precision Floating Point				
	Max Frequency		866 MHz		Up to 1 GHz		
	Memory		L1 Cache 32KB I / D, L2 Cache 512KB, on-chip Memory 256KB				
Processing System	External Memory Support		DDR3, DDR3L, DDR2, LPDDR2, 2x QSPI, NAND, NOR				
	Peripherals		2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO				
Programmable Logic	Approximate ASIC Gates	~430K (28k LC)	~1.1M (74k LC)	~1.3M (85k LC)	~1.9M (125k LC)	~5.2M (350k LC)	~6.6M (444kLC)
	Block RAM	240KB	380KB	560KB	1,060KB	2,180KB	3,020KB
	Peak DSP Performance (Symmetric FIR)	100 GMACS	200GMACS	276 GMACS	593 GMACS	1334 GMACS	2662 GMACS
	PCI Express® (Root Complex or Endpoint)	-	Gen2 x4	-	Gen2 x4	Gen2 x8	
	Agile Mixed Signal (XADC)		2x 12bit 1Msps A/D Converter				
I/O	Processor System IO				130		
	Multi Standards 3.3V IO	100	150	200	100	212	250
	Multi Standards High Performance 1.8V IO	-	-	-	150	150	150
	Multi Gigabit Transceivers	-	4	-	4	16	16

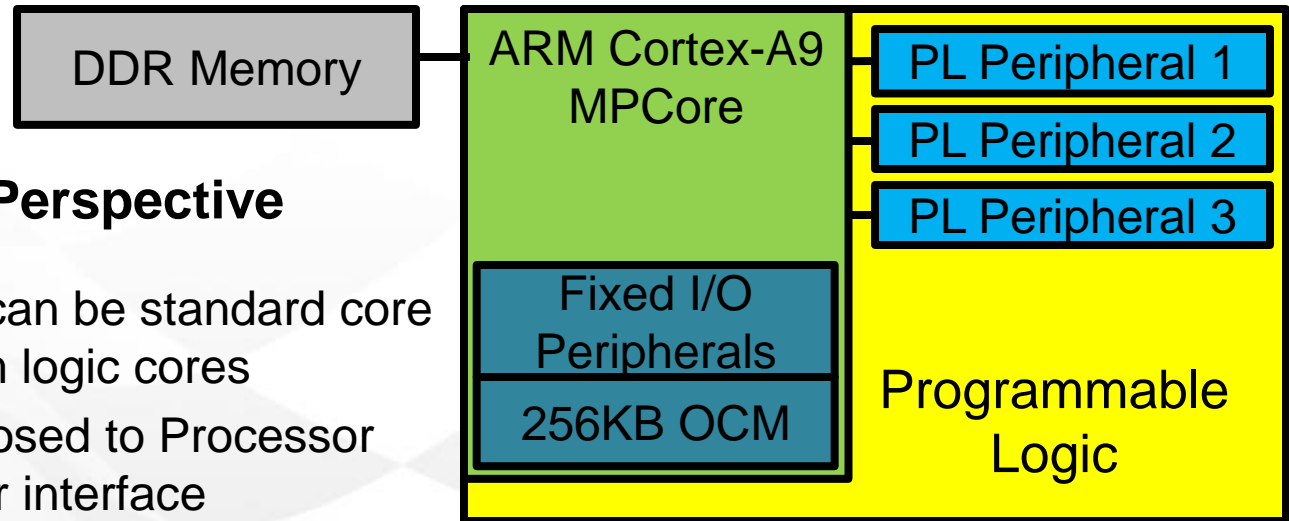
MicroZed

ZedBoard

Connecting HW and SW

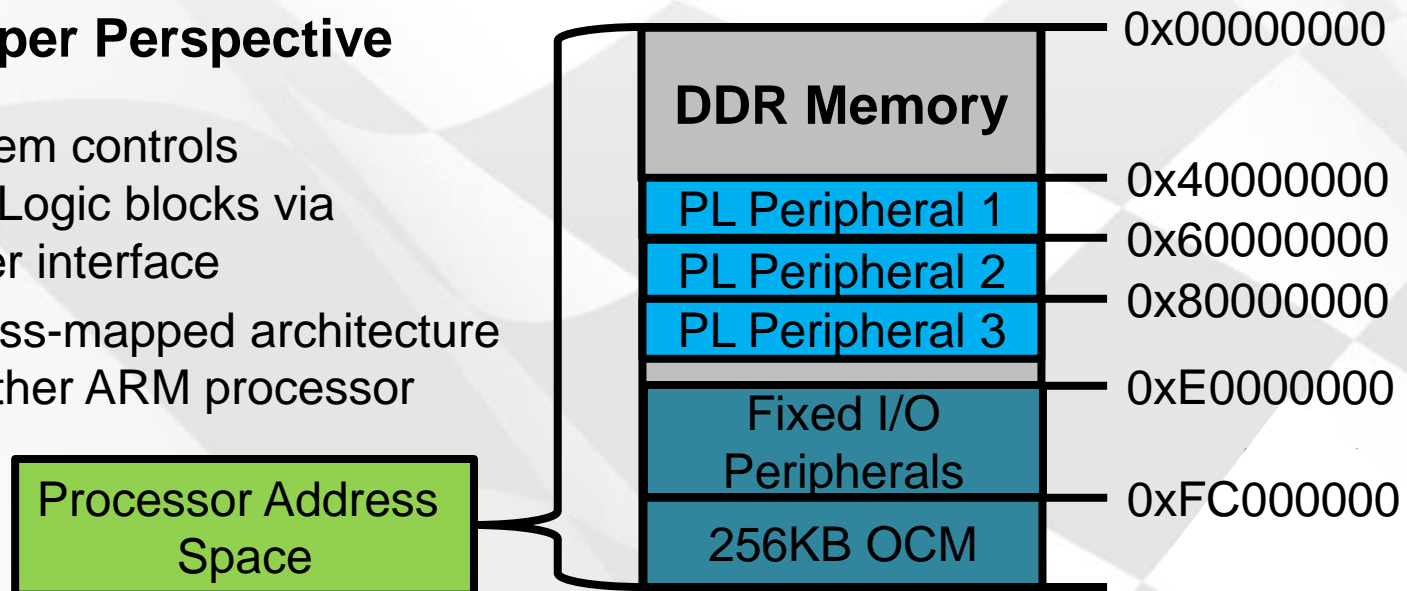
Hardware Designer Perspective

- Peripheral blocks can be standard core offerings or custom logic cores
- Logic controls exposed to Processor System via register interface



Software Developer Perspective

- Processor System controls Programmable Logic blocks via exposed register interface
- Standard address-mapped architecture similar to any other ARM processor



The Zynq Hardware Platform

- **Output from Vivado**

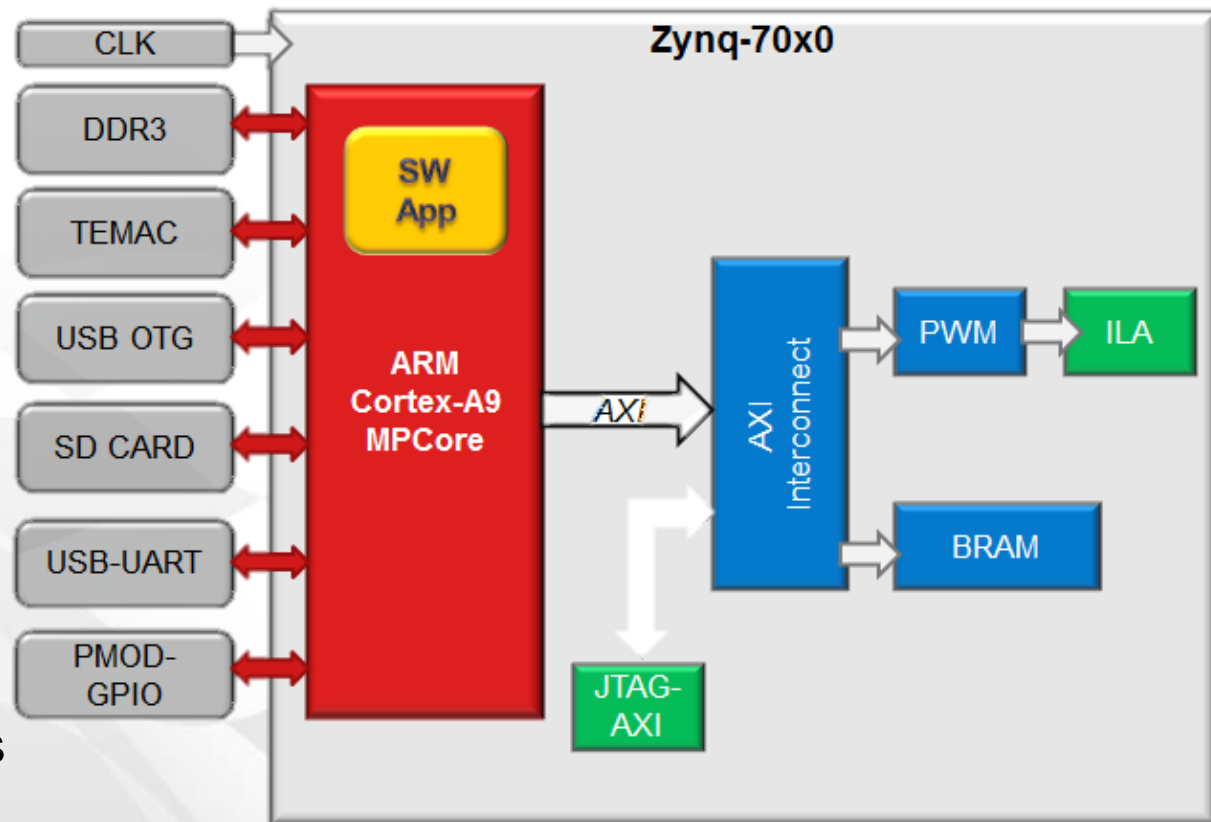
- Hardware description in XML format
- FPGA bitstream
- Block RAM Memory Map (BMM) file

- **Includes**

- Zynq processor configuration
- PS Peripheral and MIO assignments
- PL design and pinout

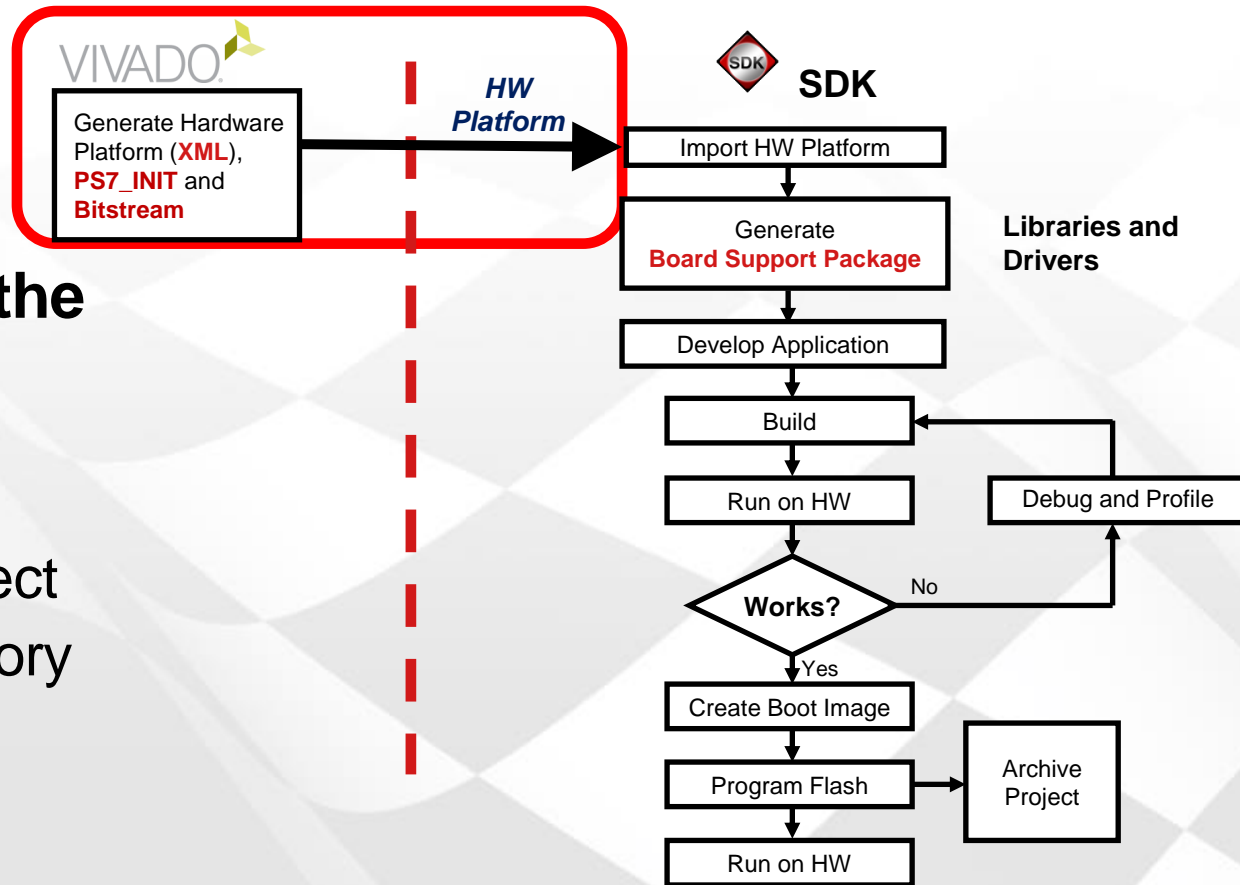
- **Pre-built for this course**

- **To learn how to create hardware platforms, look at the *Zynq Hardware Development* course**



Lab 1 – Explore an Exported Hardware Platform

- Created in Vivado
- Pre-built
- This lab includes the complete Vivado project
 - Not necessary to have the full project
 - The export directory alone is enough
- No Xilinx tools required for this lab



Questions

- **What tool suite creates the Zynq hardware platform archive?**
 - Vivado is the currently recommended tool for creating Zynq hardware platforms, although PlanAhead & XPS can also do this
- **Which file in the hardware platform archive is most comparable to a 'datasheet' for the hardware platform?**
 - ps7_init.html
- **Which file contains the information that SDK will later use to build a BSP?**
 - The XML file – Z_system.xml
- **Which file will later be used to initialize the Zynq PS during a debug session?**
 - The TCL file – ps7_init.tcl

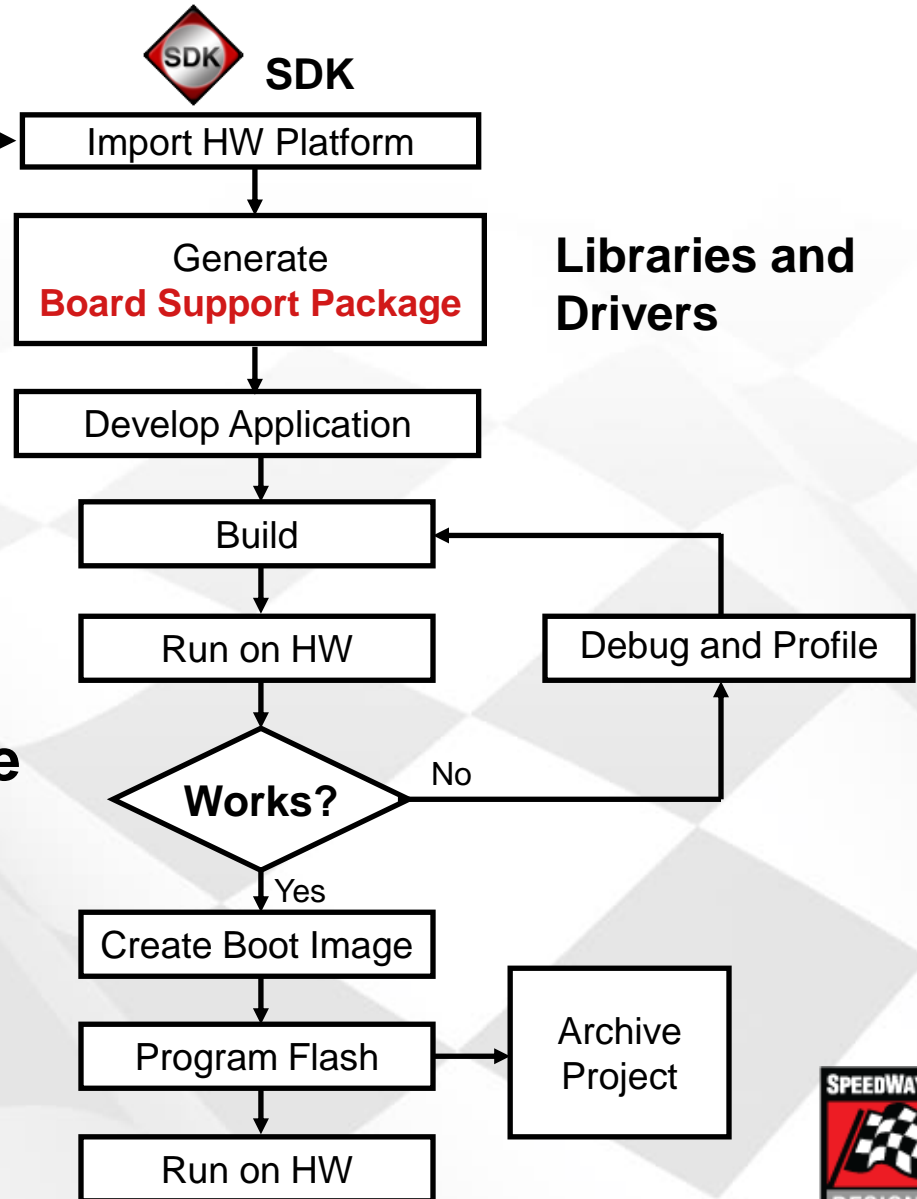
Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

SDK Application Development Flow

VIVADO

- **Generate Hardware Platform (XML), PS7_INIT and Bitstream**
- **Compiler**
 - Sourcery CodeBench™
- **BSP generation**
- **System debugger**
- **Hardware and software download**
- **Cross-probing with hardware**
- **Multi-processor debugging**
- **Linker script generation**
- **Boot image creation**
- **Flash memory programming**



Basic Eclipse Concepts

- **Workbench**

- Refers to the Eclipse IDE

- **Workspace**

- Keeps Workbench session information
- Stores Eclipse preferences
- Only one active Workspace at a time per Workbench
- Manages C/C++ projects

- **Perspectives**

- Collection of functionally related views, editors, menu items, and toolbar buttons
- Layout of views in perspective can be customized

- **Views**

- Basic User Interface element
- Provides information, control, or navigation

SDK Workbench Views – C/C++ Perspective

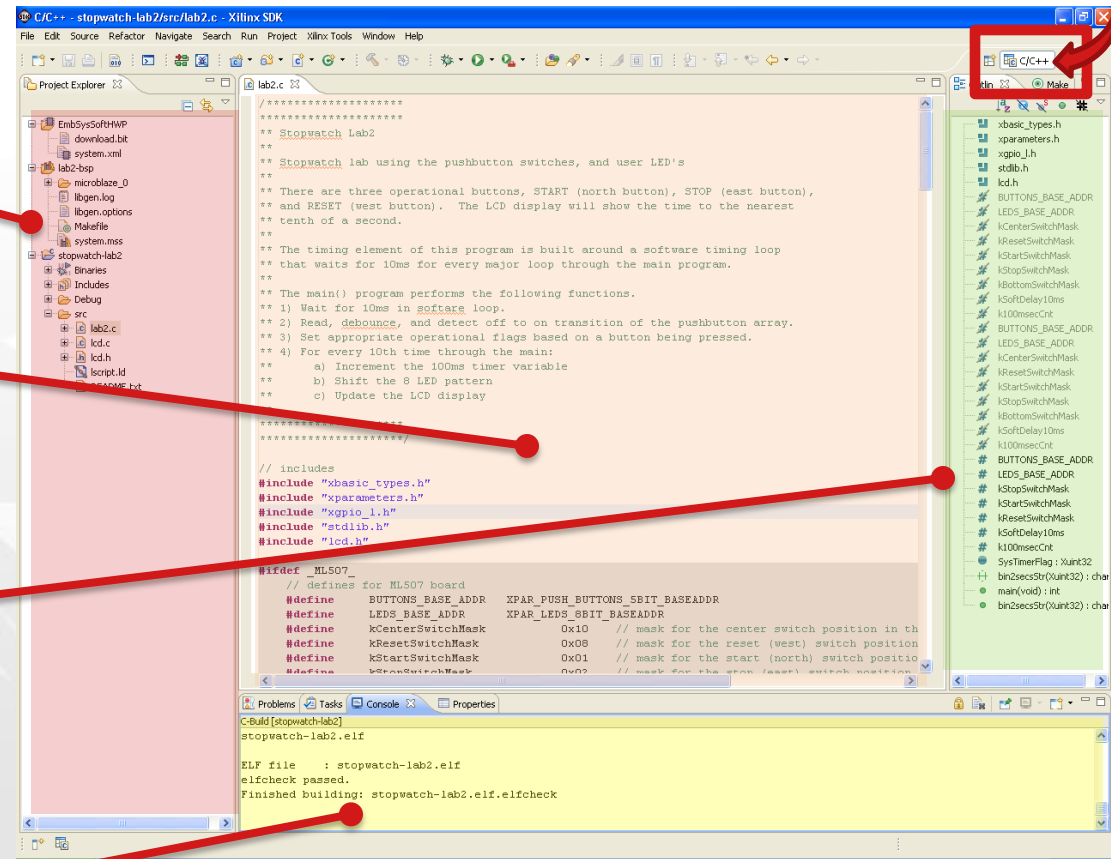
Active Perspective indicated here

Project Explorer
displays the projects with
icons for easy
identification

C/C++ editor for
integrated software
creation

Code outline displays
elements of the software
file under development
with icons for easy
identification

Problems, Console, and
Properties views list
output information

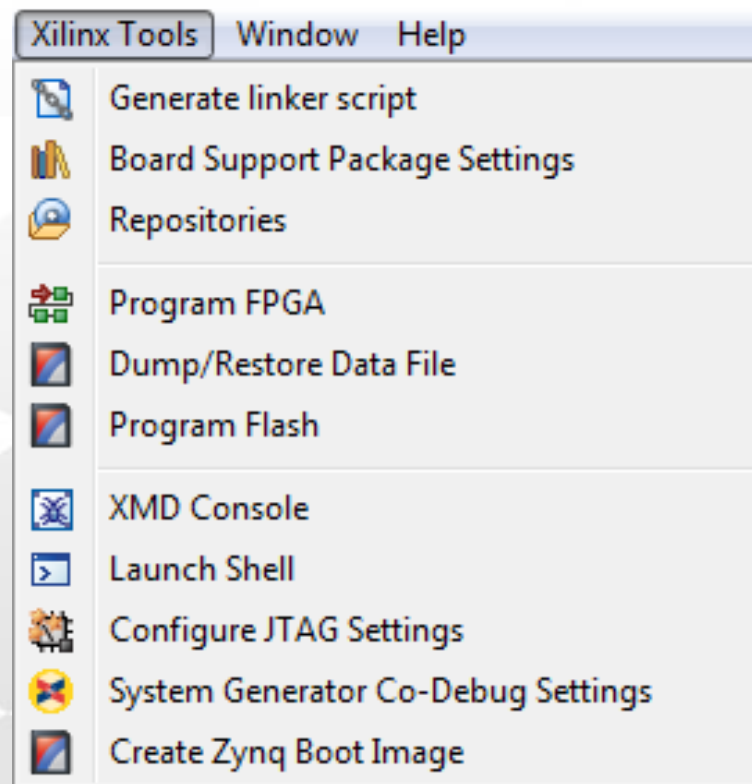


Xilinx Plug-ins

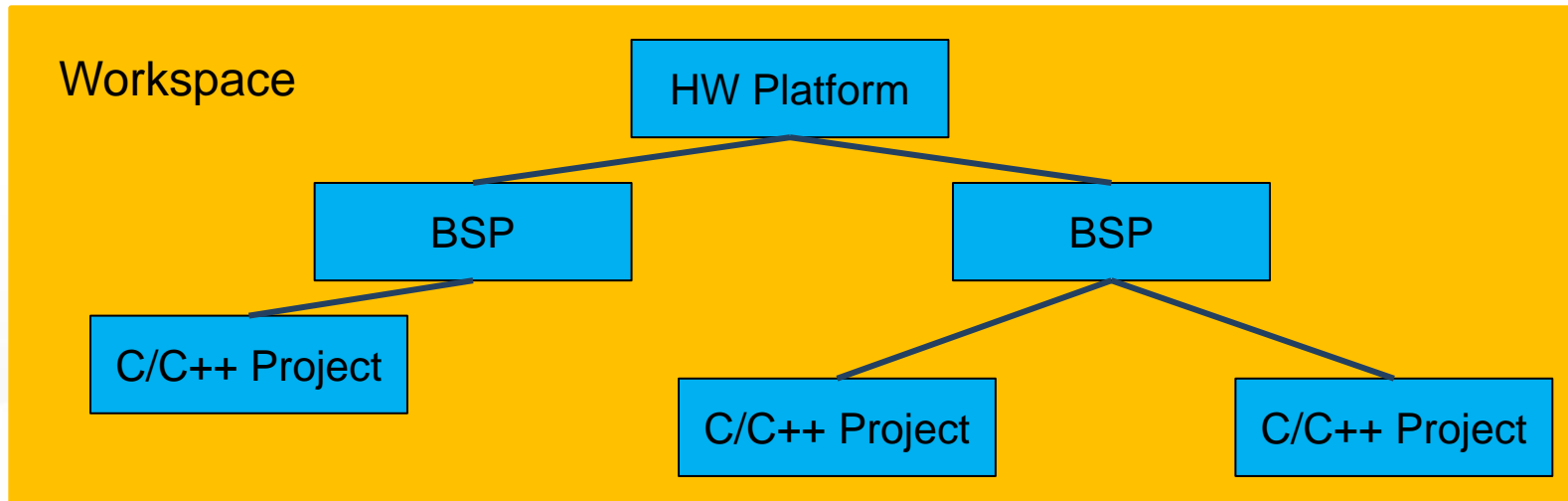
- **Plug-ins provide a way to customize Eclipse**

- **Xilinx plug-ins include**

- Project creation wizard
- Xilinx specific managed build
- Xilinx specific tool chain and utilities
- Xilinx specific menu items
- Debug integration
- Linker Script generator
- Flash Programmer
- Xilinx documentation



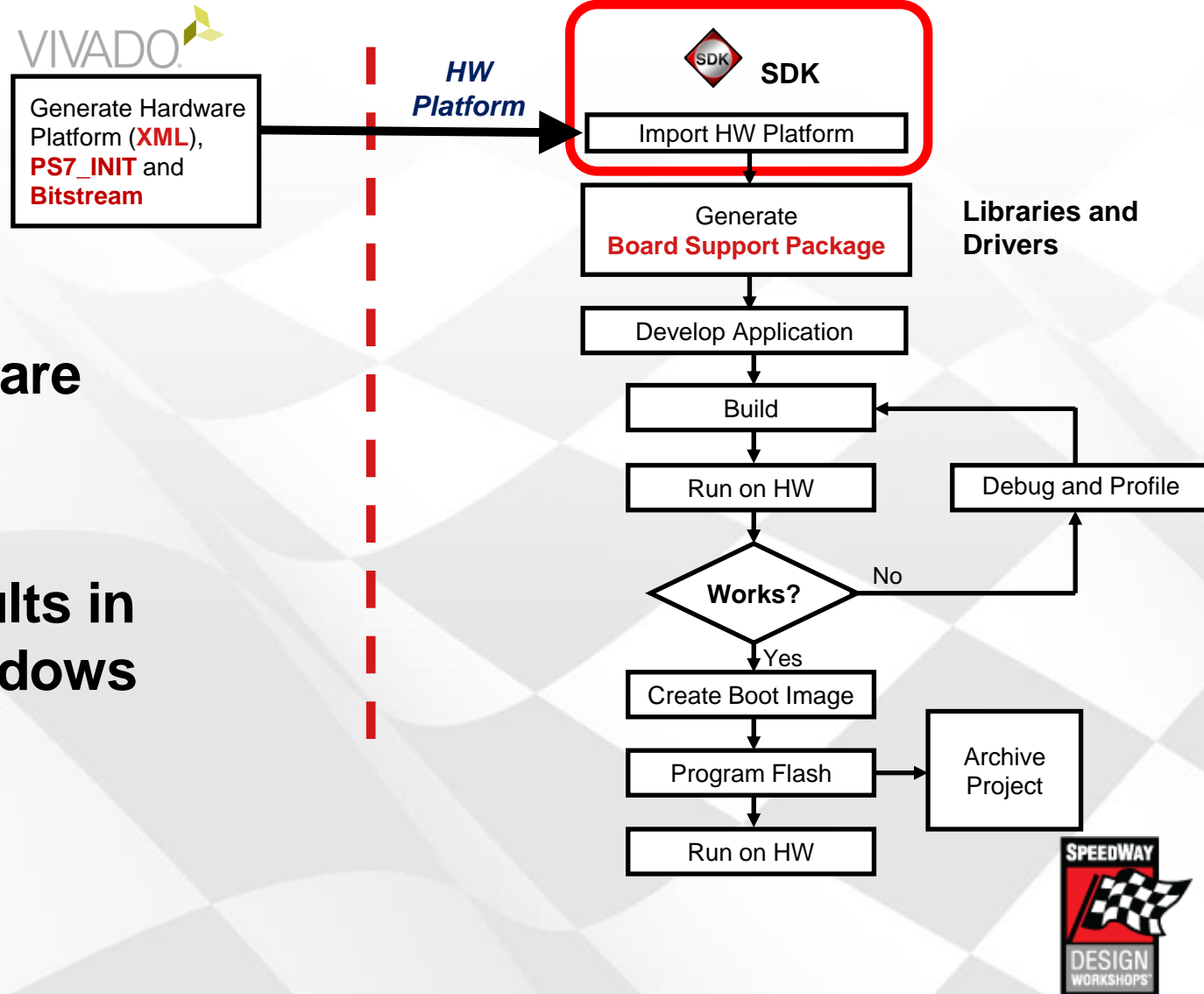
Application Dependencies



- **Every workspace has one Hardware Platform**
 - Specified through an XML file
 - XML file is generated in Vivado
- **Multiple Board Support Packages (BSPs) can reference a single Hardware Platform**
 - Xilinx provides a Standalone BSP
- **Multiple C/C++ projects can reference a single BSP**

Lab 2 – SDK Workspace and Hardware Platform

- Launch SDK
- Create a workspace
- Import hardware platform
- Examine results in SDK and Windows Explorer



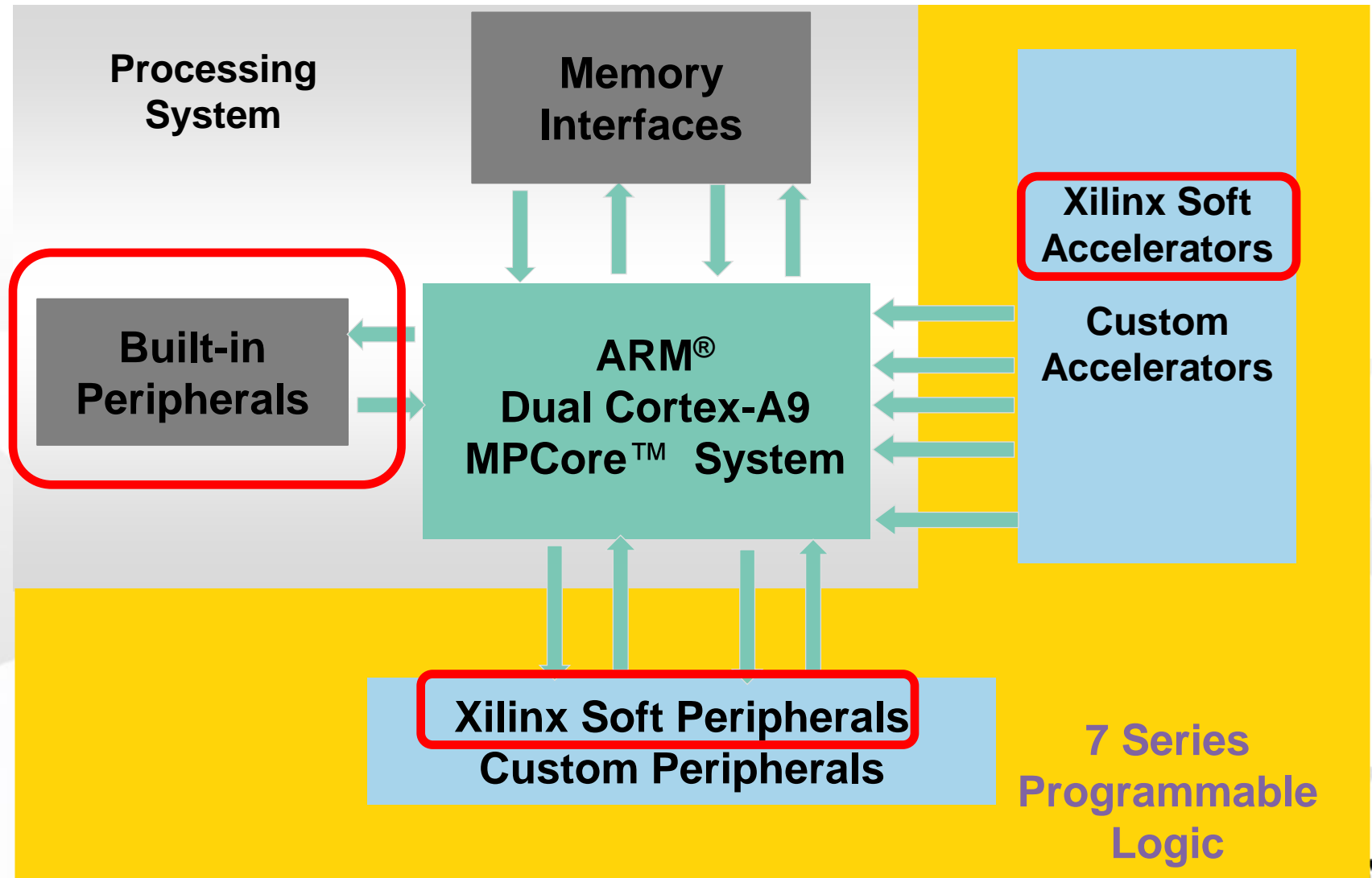
Questions

- **What is the purpose of the SDK Workspace?**
 - The Workspace holds a collection of projects related to your software application development. It will typically contain one hardware platform and one or more BSPs and software applications. The workspace contains your SDK settings, software sources, and logs.
- **When you need to share an SDK Workspace, is it a good idea to simply zip it up and send it over?**
 - NO! In Lab 8, we'll show you the proper way.
- **What is address range for the DDR3 (ps7_ddr_0)?**
 - 0x00100000 to 0x1ffffff (ZedBoard) or 0x3ffffff (MicroZed)
- **How large is the QSPI Flash in the system?**
 - $0xfcffffff - 0xfc000000 + 1 = 0x1000000 = 16 \text{ MB} = 128 \text{ Mb}$ addressable space. ZedBoard actually has a 256 Mb device, but the upper 128 Mb is accessed by flipping a control register, not in linear address space.
- **Where will you find the hardware datasheet for the ps7* peripherals?**
 - The Zynq TRM

Agenda

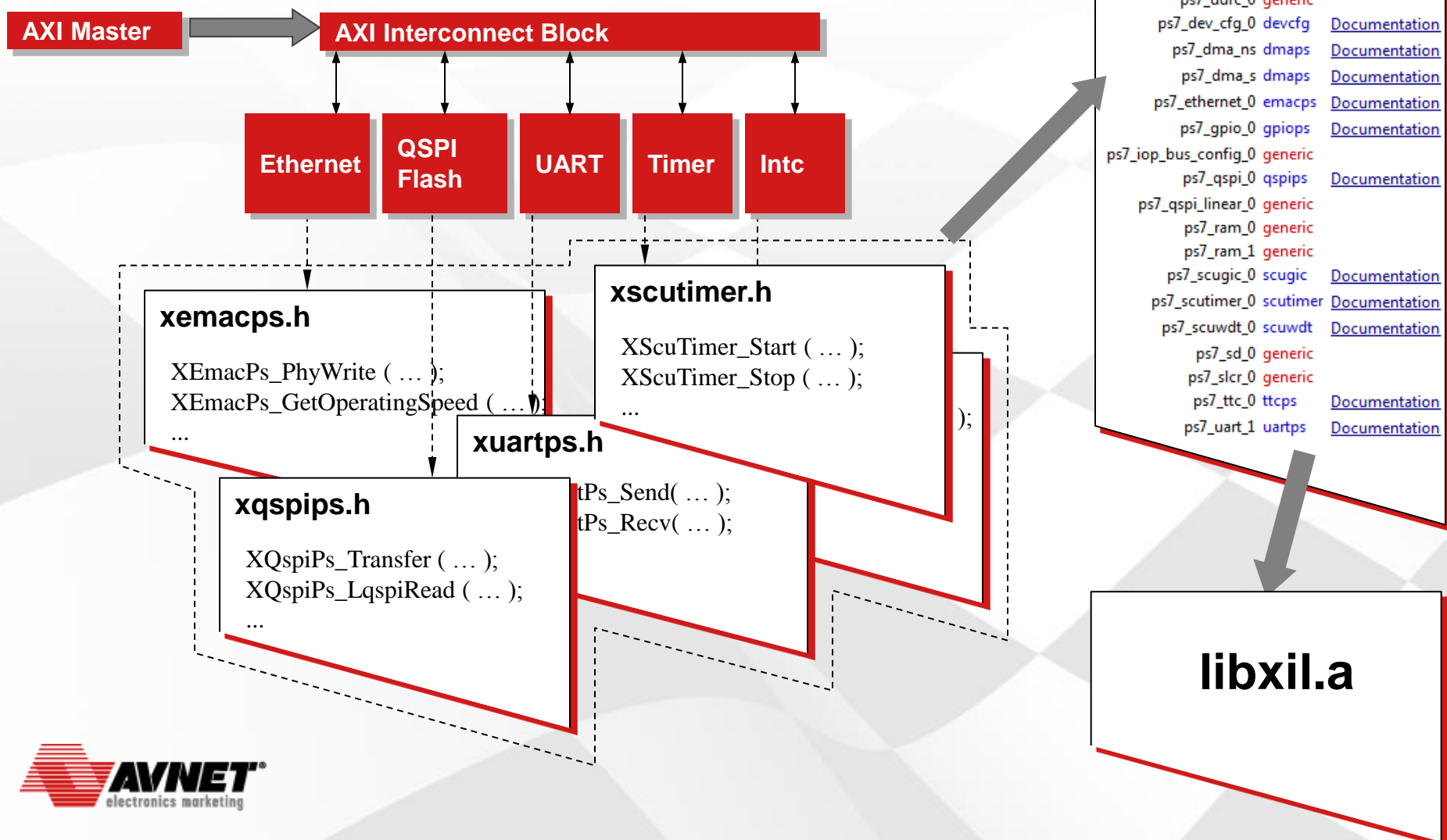
Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Xilinx Provides Drivers and Services

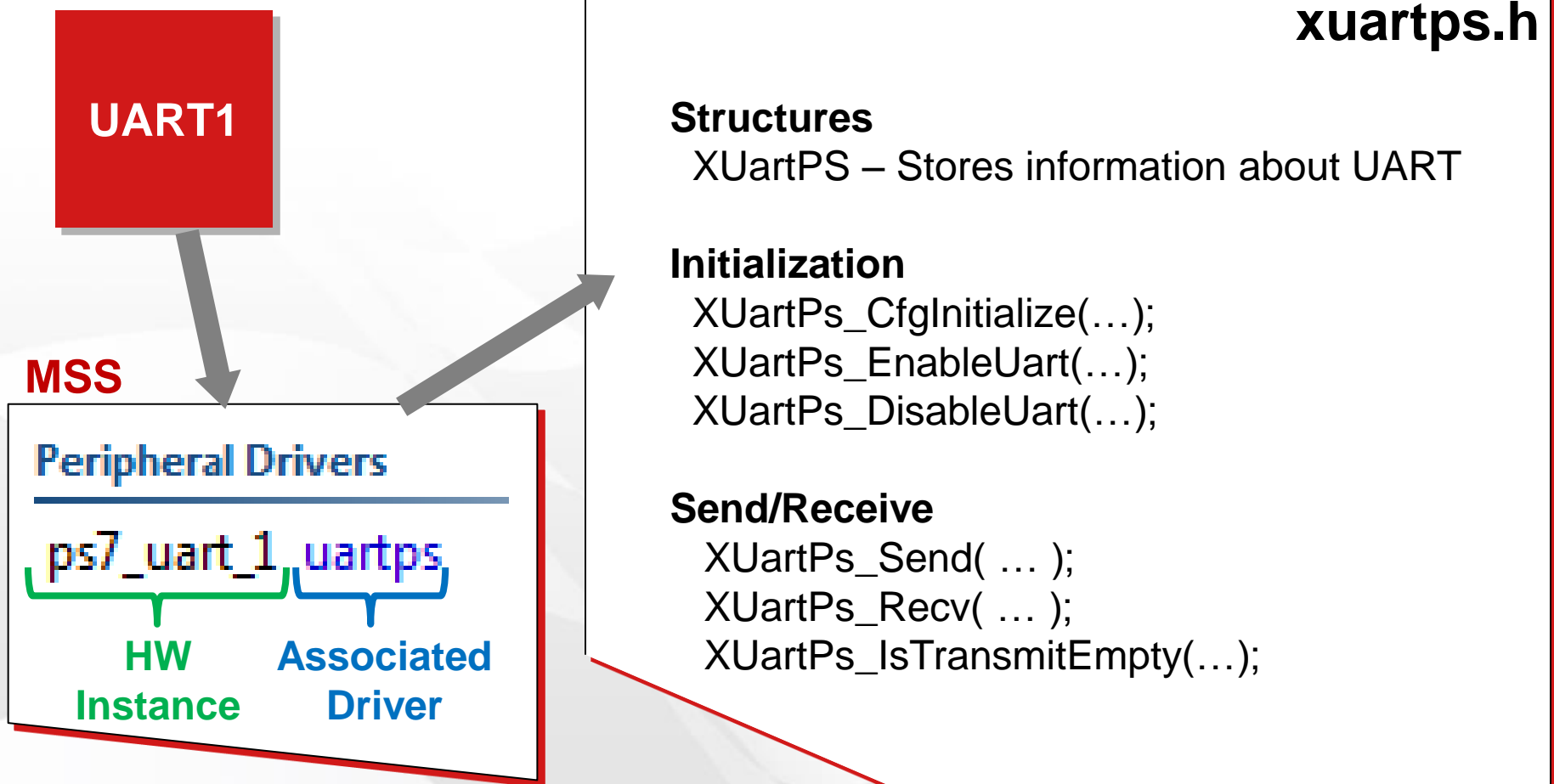


Board Support Package (BSP)

Board Support Packages are collections of parameterized drivers for a specific μ P system



UART Device Driver Example

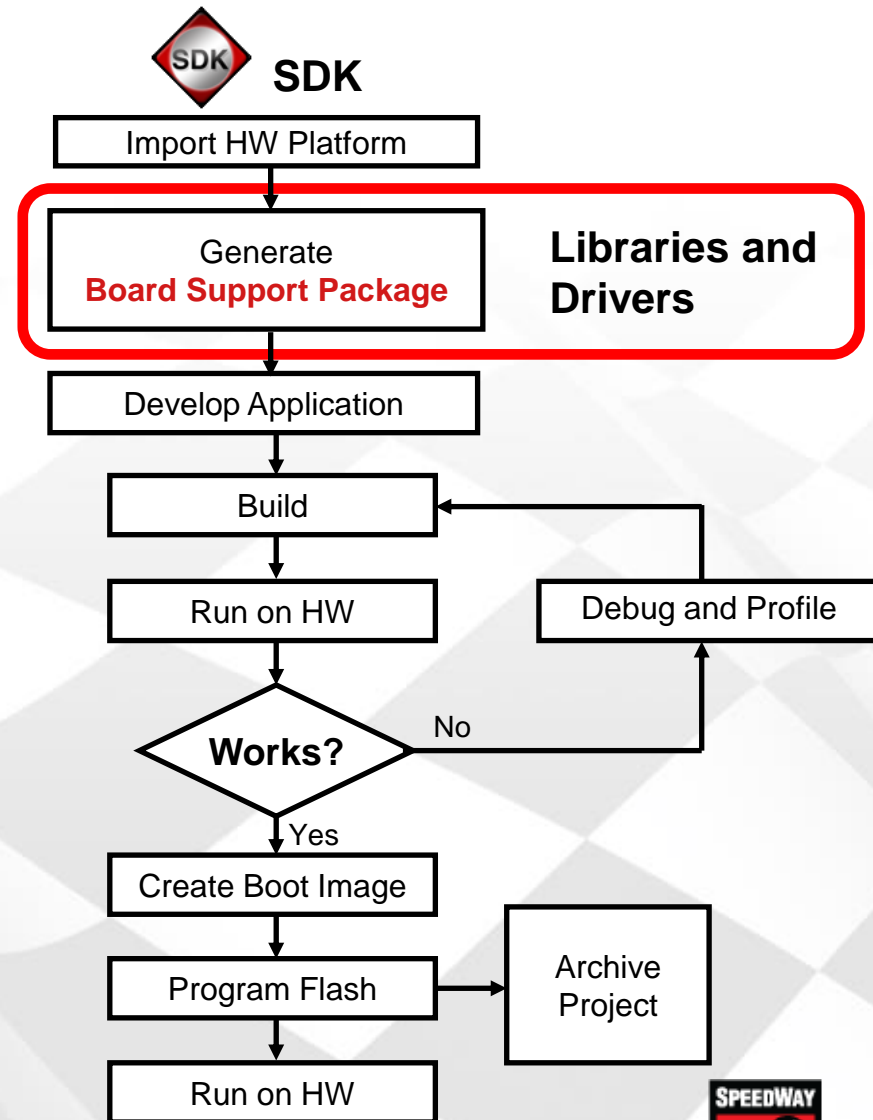


GCC Low-Level Runtime Library

- **Zynq uses standard C libraries from libgcc.a**
- **Included and statically linked from toolchain install folder**
`C:\Xilinx\SDK\2013.3\gnu\arm\nt\lib\gcc\arm-xilinx-eabi\4.7.3\`
- **Equivalent to MicroBlaze libc and libm**
- **Routines provided**
 - Integer library routines
 - Soft float library routines
 - Decimal float library routines
 - Fixed-point fractional library routines
 - Exception handling routines
 - Miscellaneous routines

Lab 3 – Board Support Package

- Create the BSP
- Examine what was created
- Explore a built-in device driver



Questions

- **What are the base addresses for the on-chip RAMs, XPAR_PS7_RAM_0_S_AXI_BASEADDR and XPAR_PS7_RAM_1_S_AXI_BASEADDR?**
 - 0x00000000 and 0xFFFF0000
- **What size is PS7_RAM_0 ?**
 - $0x0002FFFF - 0x00000000 + 1 = 0x30000 = 196608 \text{ bytes} = 192 \text{ KB}$
- **What parameter name would you access for the PWM peripheral interrupt?**
 - XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR, which happens to be set to interrupt #91

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Example Code

MSS

Peripheral Drivers

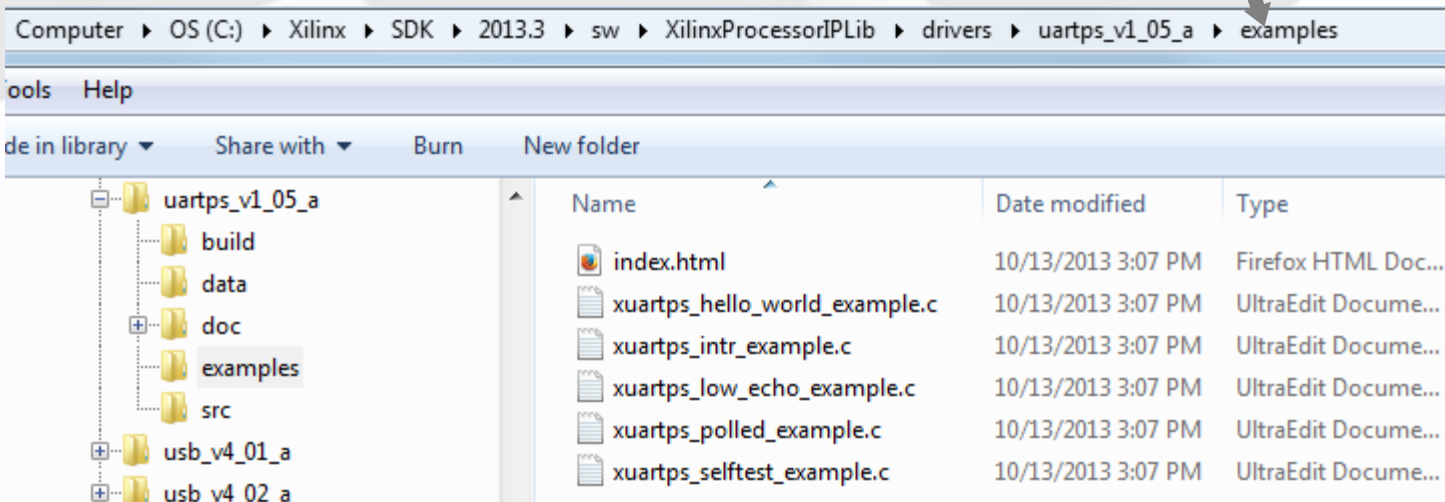
Drivers present in the Board Support Package.

axi_bram_ctrl_0	bram	Documentation	Examples
ps7_scugic_0	scugic	Documentation	Examples
ps7_scutimer_0	scutimer	Documentation	Examples
ps7_scuwdt_0	scuwdt	Documentation	Examples
ps7_uart_1	uartps	Documentation	Examples
ps7_usb_0	usbps	Documentation	Examples
ps7_xadc_0	xadcps	Documentation	Examples

Xilinx provides example code for nearly every included peripheral driver

Example Applications for the driver uartps_v1_05_a

- [uartps_hello_world_example.c](#) (source)
- [uartps_intr_example.c](#) (source)
- [uartps_low_echo_example.c](#) (source)
- [uartps_polled_example.c](#) (source)
- [uartps_selftest_example.c](#) (source)



New C Project Sample Applications

- **Dhrystone**
 - Dhrystone benchmark
- **Empty Application**
 - Will setup the project references
 - Can be used to import an existing application
- **Hello World**
 - stdout to UART
- **LwIP Echo Server**
 - Simple example using the LwIP Ethernet stack
- **Memory Tests**
 - Tests all available memories
- **Peripheral Tests**
 - Basic peripheral tests
- **Zynq FSBL**
 - First stage bootloader

Available Templates:

Dhrystone

Empty Application

Hello World

LwIP Echo Server

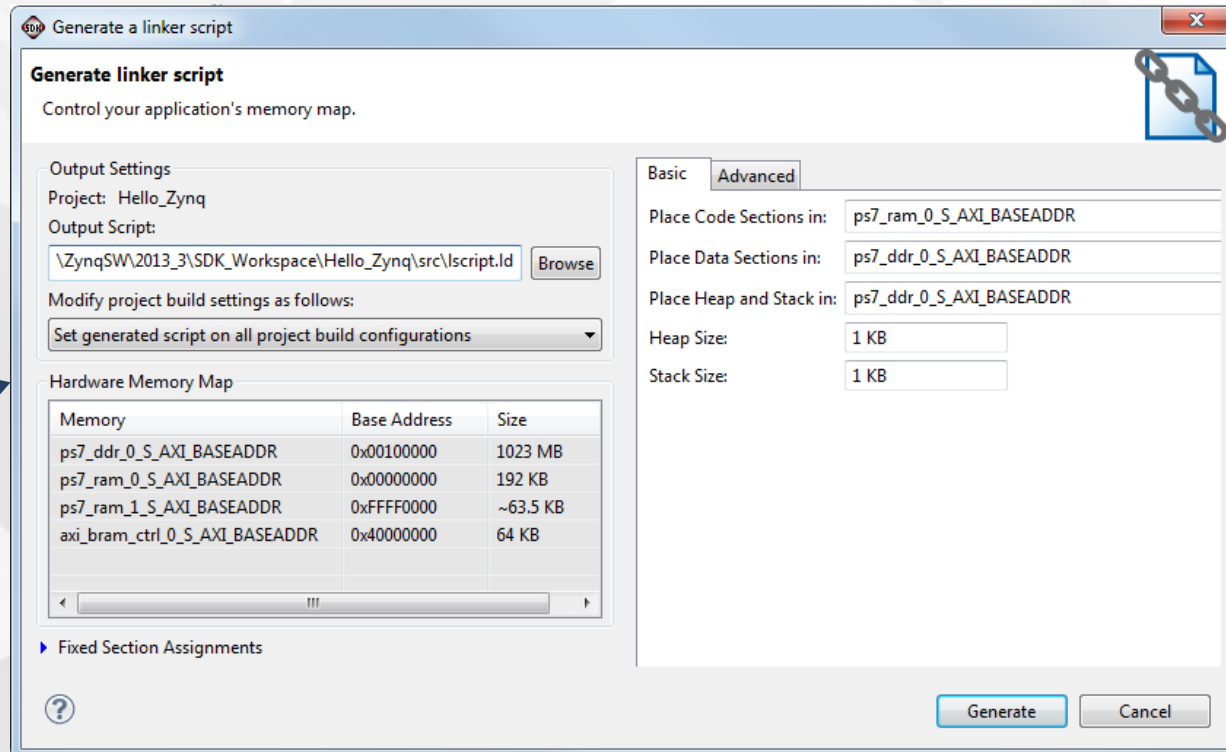
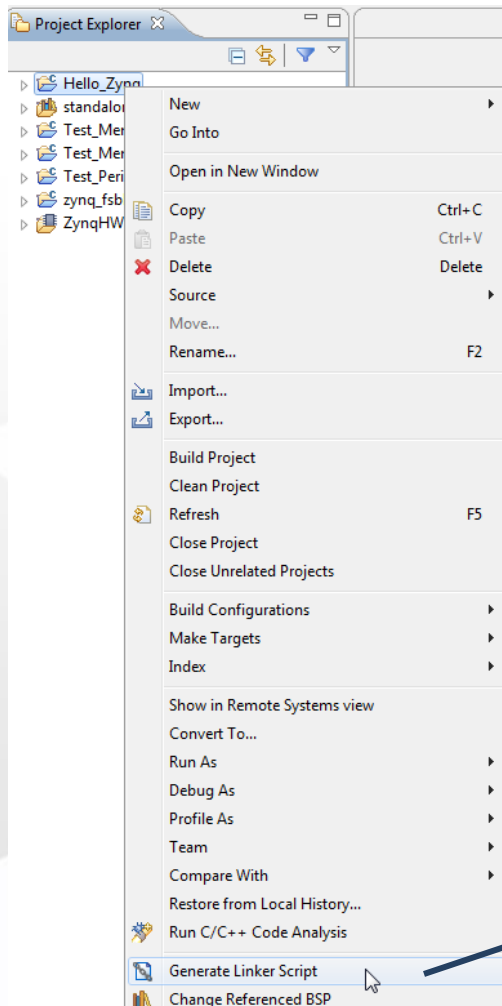
Memory Tests

Peripheral Tests

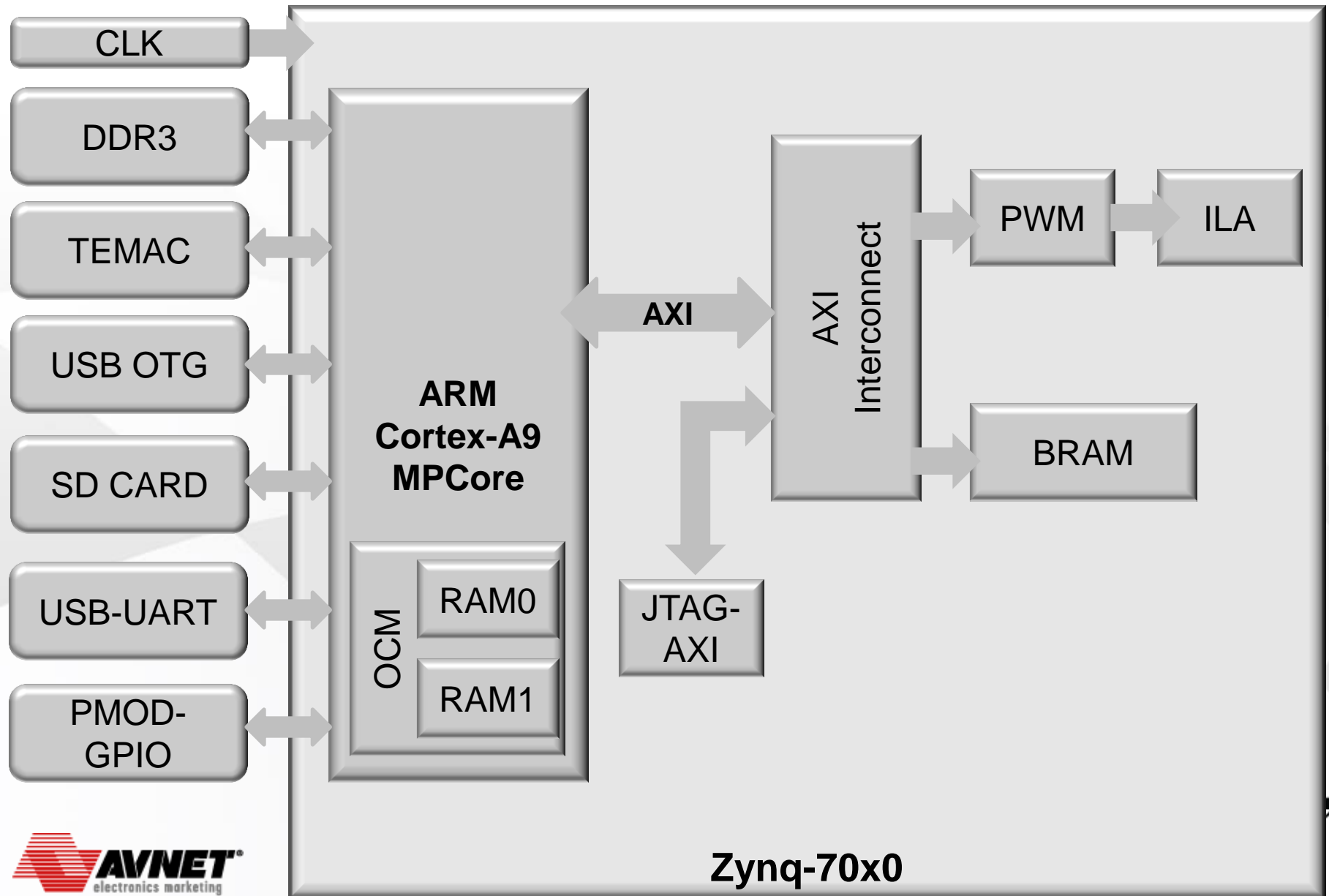
Zynq FSBL

Linker Script Generator

- Provides an easy way to modify the linker script
 - Advanced tab allows individual section assignments
 - Modify the Heap and Stack sizes

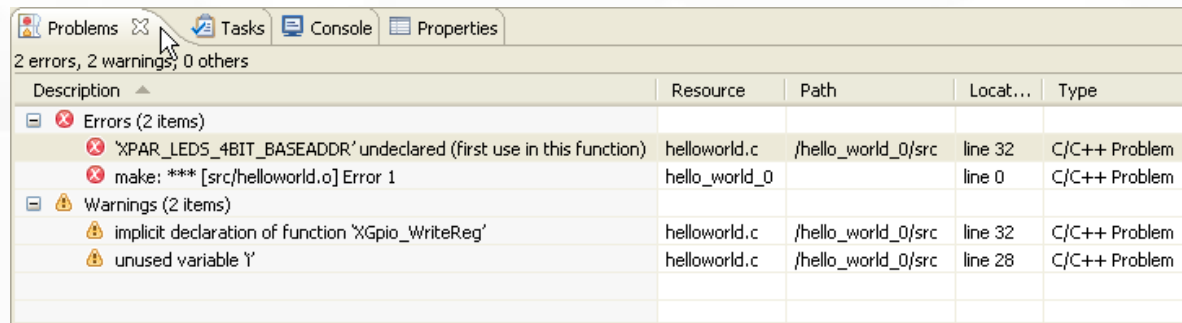


Memories in the Lab Hardware Platform



Build Errors and Warnings

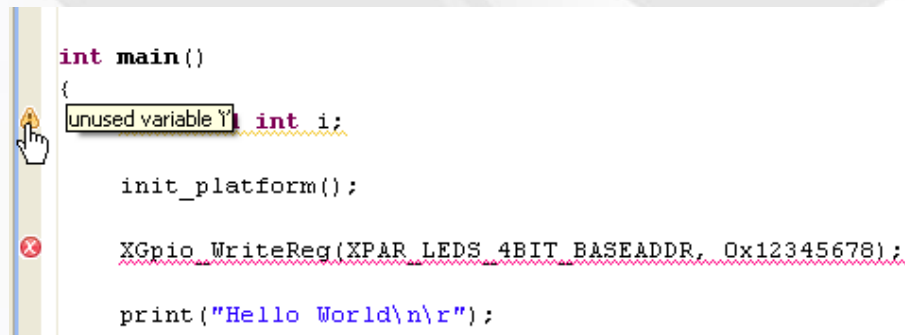
- Errors and Warning are displayed
 - In the Problems View



The screenshot shows the 'Problems' tab in an IDE. It displays a list of 2 errors and 2 warnings. The table below represents the data shown in the table:

Description	Resource	Path	Locat...	Type
Errors (2 items)				
'XPAR_LEDS_4BIT_BASEADDR' undeclared (first use in this function)	helloworld.c	/hello_world_0/src	line 32	C/C++ Problem
make: *** [src/helloworld.o] Error 1	hello_world_0		line 0	C/C++ Problem
Warnings (2 items)				
implicit declaration of function 'XGpio_WriteReg'	helloworld.c	/hello_world_0/src	line 32	C/C++ Problem
unused variable 'Y'	helloworld.c	/hello_world_0/src	line 28	C/C++ Problem

- In the Editor margin
 - Hover to see the warning or error

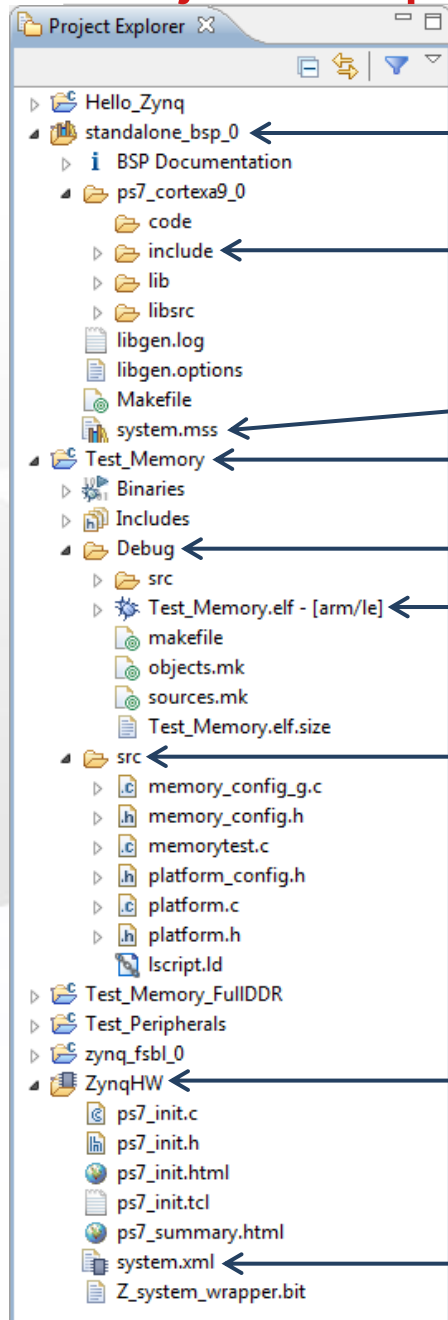


The screenshot shows a code editor with the following C code:

```
int main()  
{  
    unused variable 'Y' int i;  
  
    init_platform();  
  
    XGpio_WriteReg(XPAR_LEDS_4BIT_BASEADDR, 0x12345678);  
  
    print("Hello World\n\r");  
}
```

In the left margin, there is a yellow warning icon next to the line `unused variable 'Y' int i;` and a red error icon next to the line `XGpio_WriteReg(XPAR_LEDS_4BIT_BASEADDR, 0x12345678);`.

Project Explorer View



BSP

BSP Include files

BSP Settings file

C Project

Build results

Executable

C Project Sources

Hardware Platform

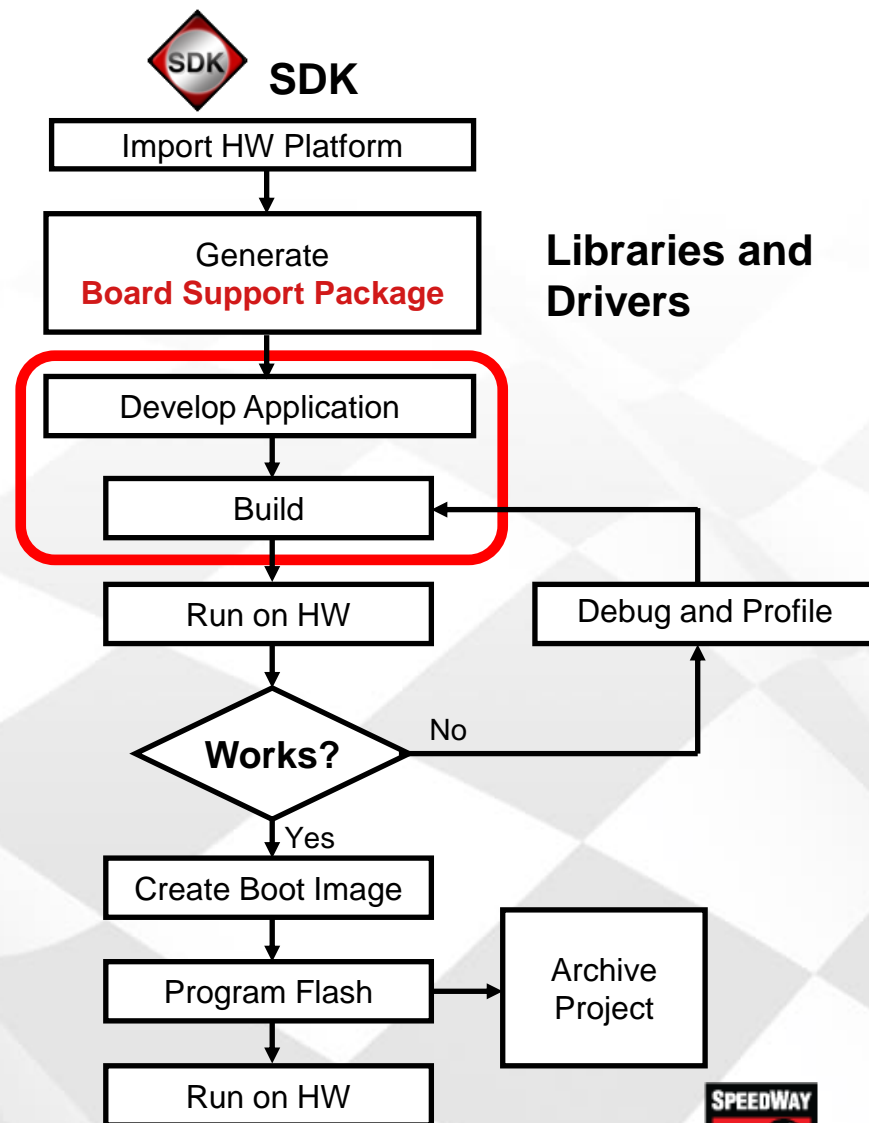
Hardware Platform Settings

- **Lists all the projects alphabetically**
- **Multiple BSPs allowed**
 - With different settings or libraries
- **Multiple C projects**
- **All the files can be opened with the editor**
 - Hardware specification file shows the memory map, core used, and contains links to the datasheets
 - BSP Settings file lists the device drivers used and contains links to the drivers documentation



Lab 4 – Developing Applications

- Create a new Xilinx C/C++ application
- Use Example Code
 - Hello World
- Use a built-in template
 - Memory Test
 - Peripheral Test
- Explore the code
- Use the Generate Linker Script tool to modify a linker script



Questions

- **You've been assigned a task to develop code to test reading and writing to the PL BRAM (peripheral axi_bram_ctrl_0 in this hardware platform). What do you do?**
 - When starting to work with a new peripheral and its associated driver, the best place to start is the example code provided by Xilinx. Go to the system.mss Overview. Find the BRAM peripheral. Click on the Examples to get to the example code.
- **To what memory region(s) is the Test_Peripherals application targeted?**
 - DDR3 for everything – Code, Data, heap, and stack
- **To what memory region(s) is the Test_Memory application targeted?**
 - ps7_ram_0_S_AXI_BASEADDR for the Code, Data, heap, and stack sections
- **How much memory is tested by default?**
 - 4096 bytes
- **How many memories are tested? Which ones?**
 - 3: DDR3, RAM_0, and RAM_1

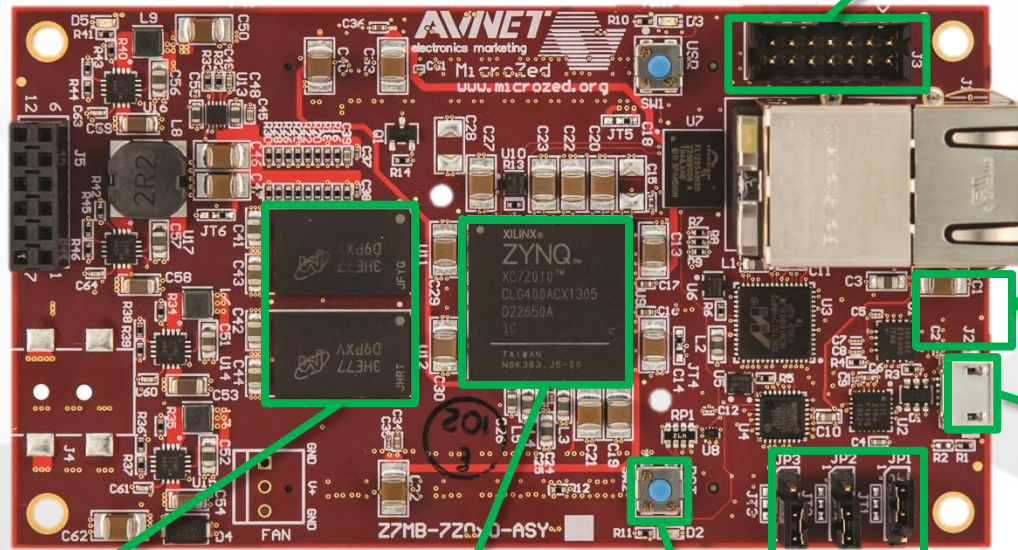
Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Lab Hardware - MicroZed



microZed™
FLEXIBILITY REDEFINED
MICROZED.ORG



J3
JTAG

J6
microSD Card
(backside)

J2
USB-UART

1 GB
DDR3

Zynq 7010

SW2
RST

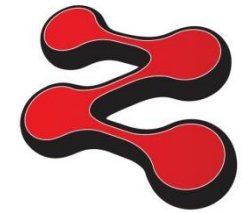
JP[3:1]
MIO/MODE
Jumpers



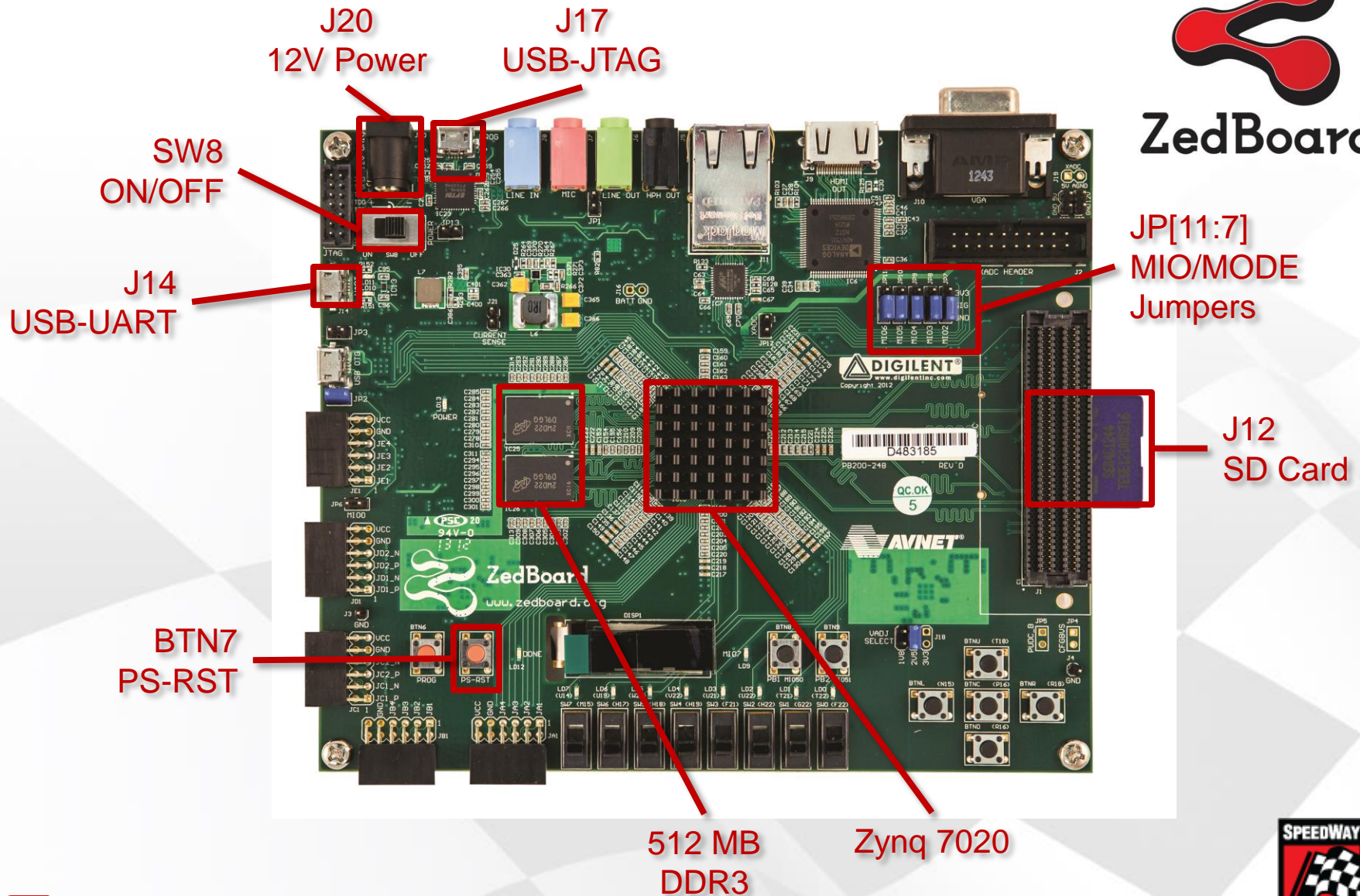
www.microzed.org



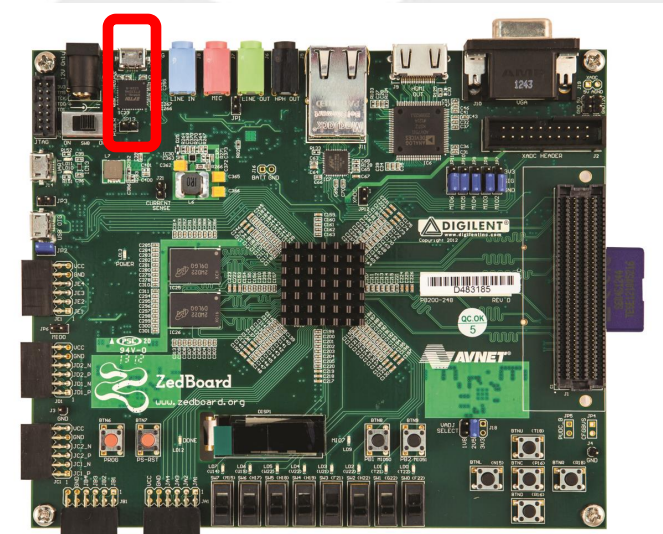
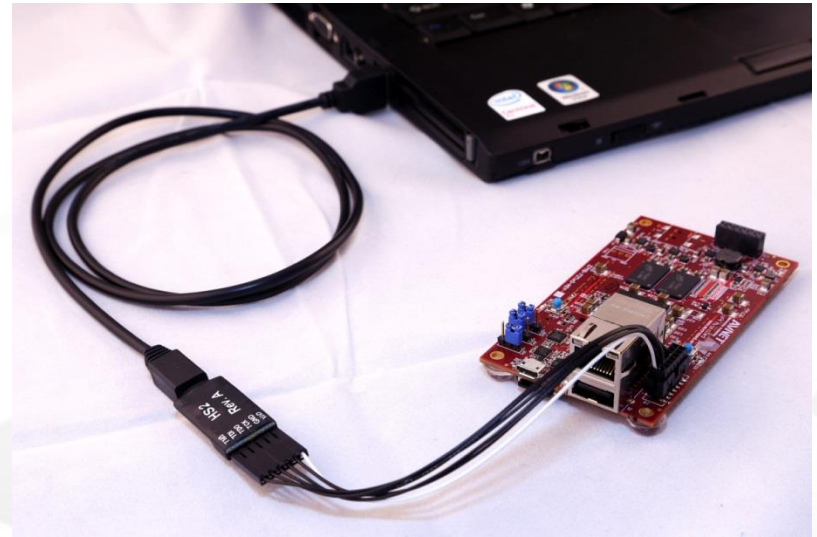
Lab Hardware - ZedBoard



ZedBoard

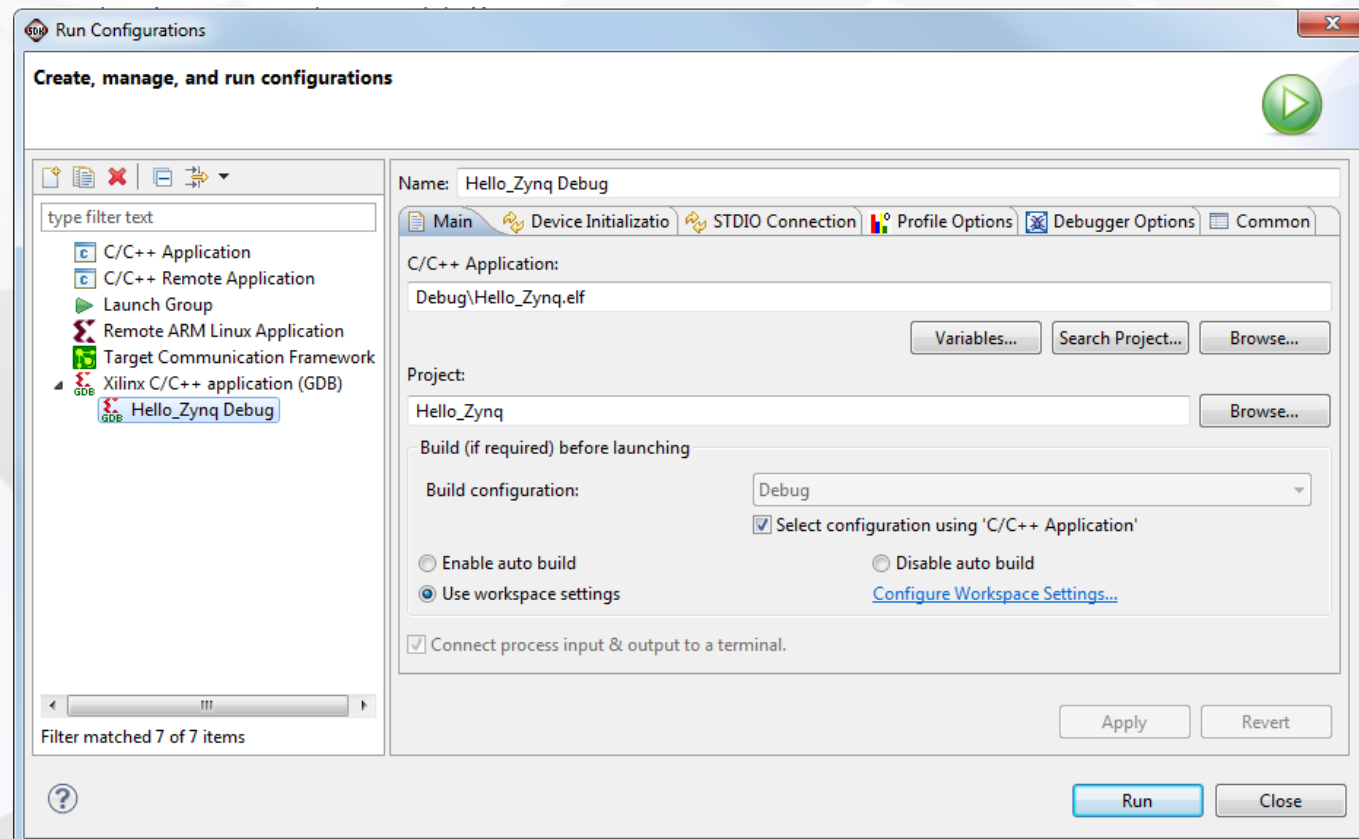


- Test access port (TAP) and boundary-scan is built onto Zynq, similar to other FPGAs
- ARM Debug Access Port (DAP) for SDK debug
- MicroZed requires external USB-JTAG pod (Digilent HS2)
<http://www.em.avnet.com/jtaghs2>
- ZedBoard has built-on USB-JTAG – no pod required



Configurations

- Configuration = settings associated with running or debugging within SDK
- 'Run' and 'Debug' configurations
- Set up for
 - Processor reset
 - Processor initialization
 - Data download
 - Terminal
 - Profiling
 - Remote debug



Application Debugging

- **System Debugger for source and assembly**
 - 3rd Party trace tools available – *Debugging ARM Processor Systems* online training

Thread and Call Stack

Active Perspective

Breakpoints

Variables

Name	Value
argc	2
argv	0xbef1e24
addr	0x402df1c0
prediv	1
mem_fd	3

Output Console

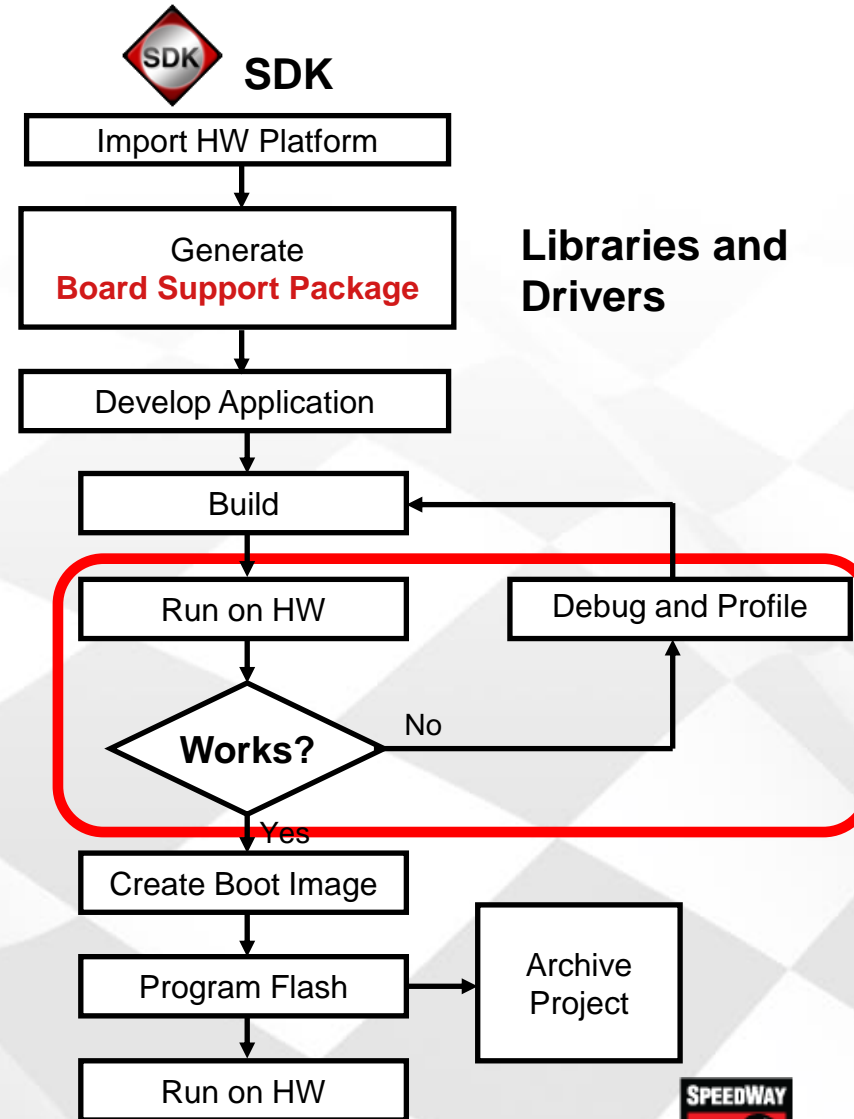
Terminal

Console

```
clkinfo_linux Debug [Remote ARM Linux Application] /tmp/clkinfo_linux.elf (1/26/12 9:31 AM)
CPU2x = 0.000000 MHz
CPU1x = 0.000000 MHz
PLL_CLK = 0.000000 MHz
fbdiv = 48
div = 2
```

Lab 5 – Connecting Hardware

- Connect the hardware
- Download an application over JTAG
- Debug an application over JTAG



Questions

- **For what can the JTAG interface be used?**
 - Read and write ARM registers
 - Configure the PL with a bitstream
 - Program attached QSPI Flash
 - Upload application code to on-chip RAM or DDR3
 - Application debug
- **Under what conditions must the hardware platform first be downloaded into the PL?**
 - Hardware Platform contains PL peripherals
- **How does the ARM get initialized when running an application from SDK?**
 - ps7_init.tcl that was provided with the hardware platform
- **How much memory is tested by default?**
 - 4 KB

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Zynq-7000 Configuration and Boot

- **Processor First! CPU configures the PS and PL**

- Standalone PL configuration (without PS configuration) is not supported
- Configuration under external host control is also possible via JTAG

- **Two boot modes**

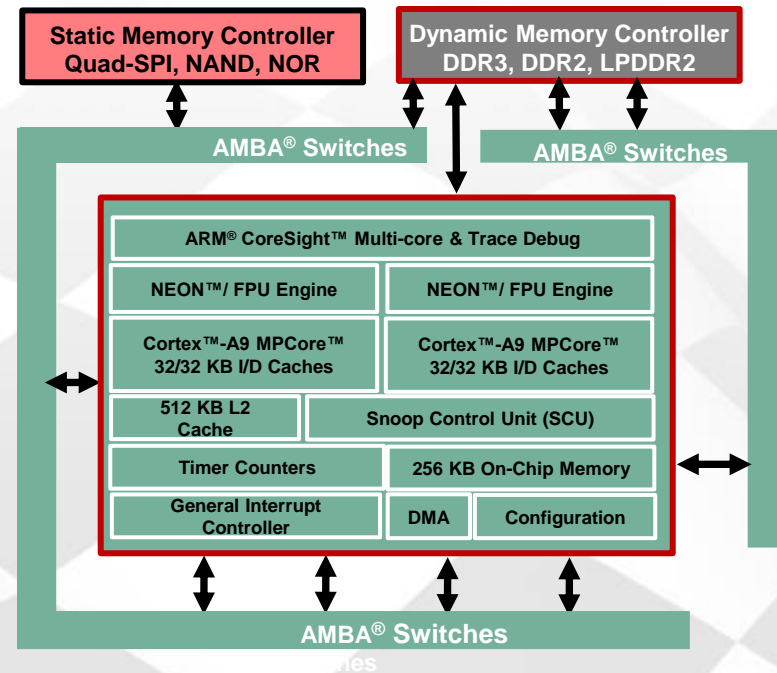
- Secure boot
- Non-secure boot

- **Four master boot methods (secure or non-secure boot)**

- QSPI (16MB, 50MB/Sec)*
- NOR (64MB, 20MB/Sec)*
- NAND (tested up to 1GB, 10MB/Sec)*
 - * cannot be used in the same design
- SD (Up to 32GB)

- **One slave boot method (non-secure)**

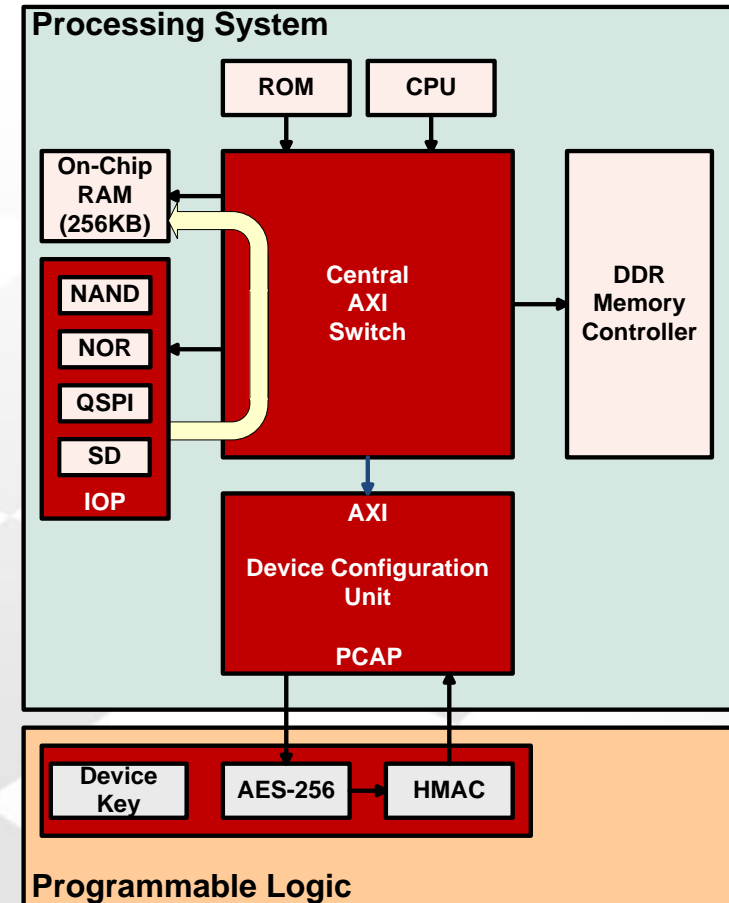
- JTAG for debug and development



Non-Secure Boot – Stage 0

- **CPU starts executing code from ROM**

1. Initializes the Cortex-A9 CPU 0
2. Checks CRC on ROM code
3. Reads the **boot mode pins** to determine the stage 1 boot mode
 - Can boot from QSPI, NAND, NOR, SD card, or JTAG
 - Stage 1 boot from SD requires SD to be connected to pre-defined MIO pins
4. Typically, a **first stage boot loader** is read from external non-volatile memory and copied into the OCM (192KB max)
 - If the Execute In Place (XIP) feature is enabled, first stage boot can be executed directly from QSPI or NOR
 - Stage 1 boot header specifies the use of XIP feature



First Stage Boot Loader (FSBL)

- **First Stage Boot Loader Functions**

- Initialize Processing System blocks
 - PLL
 - External memory controller
 - MIO
- Configure Programmable Logic with Bitstream (optional)
- Provides for secure boot option
- Execute application code

```
/* DDR Operating Mode Register */
#define DDR_MODE_STS_REG (XPS_DDR_CTRL_BASEADDR +
#define OPERATING_MODE_MASK 0x7
#define NORMAL_MODE 0x1
#define DDR_MEMORY_1 0x01000000
#define DDR_MEMORY_2 0x02000000
#define PATTERN 0xAA55AA55

***** Type Definitions *****
***** Macros (Inline Functions) Definition *****

***** Function Prototypes *****
extern void Fsb1HandoffExit(u32 Fsb1StartAddr);
extern void Fsb1HandoffJtagExit(void);

static void Fsb1Handoff(u32 Fsb1StartAddr);

static void RestartWDT(void);
static void EnableWDT(void);
static void DisableWDT(void);
static int IsWDTReset(void);

static void MarkFSBLIn(void);
static void ClearFSBLIn(void);
int Check_ddr_init(void);
void init_ddr(void);
? Get_SiliconVersion(void);
```

- **Created directly from the SDK project template**

Templates

Create one of the available templates to generate a fully-functioning application project.



Available Templates:

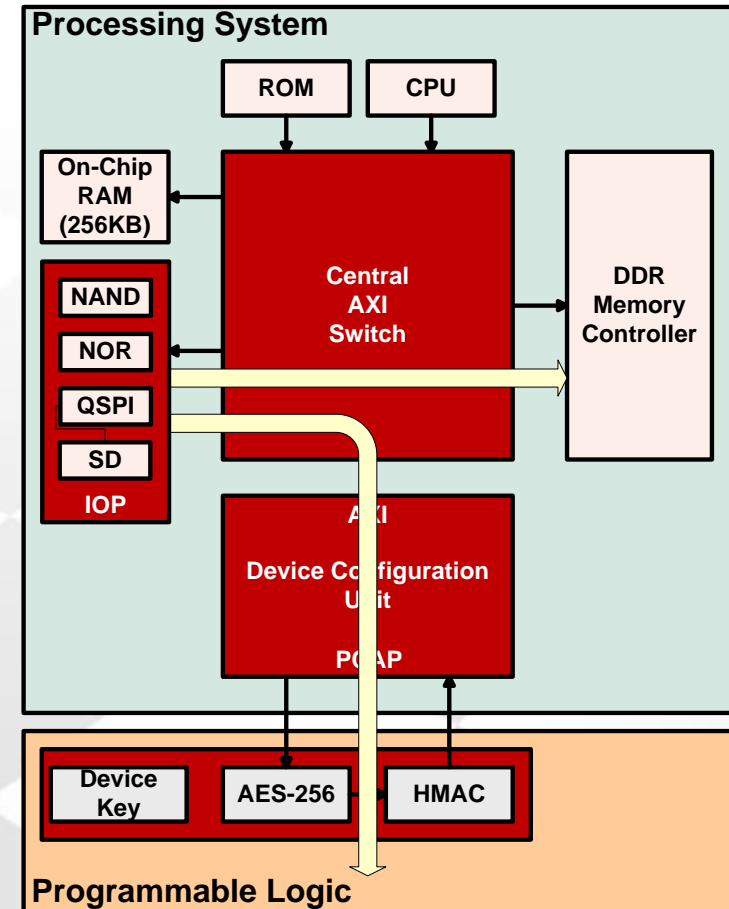
Dhrystone
Empty Application
Hello World
IwIP Echo Server
Memory Tests
Peripheral Tests
Zynq FSBL

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

Non-Secure Boot – Stage 1

- **CPU starts executing code from OCM (FSBL)**

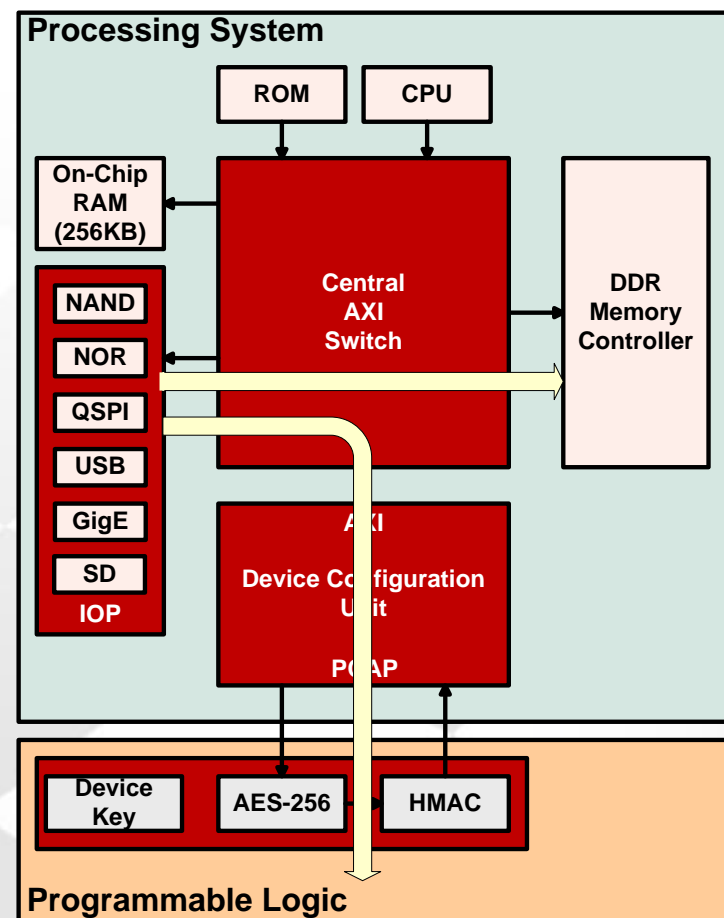
- Code execution can also be performed directly from QSPI or NOR if the Execute In Place (XIP) feature is used
 - 192KB code size limit is removed
- 1. DDR controller, clock generation module, and selected peripherals are initialized
- 2. PS application and/or second stage boot is loaded into the DDR memory
- 3. Optionally, bitstream is loaded from non-volatile memory and PL is configured
- 4. Second stage boot is optionally enabled



Stage 2 Boot

- **Stage 2 boot loader such as U-Boot runs from the DDR memory**

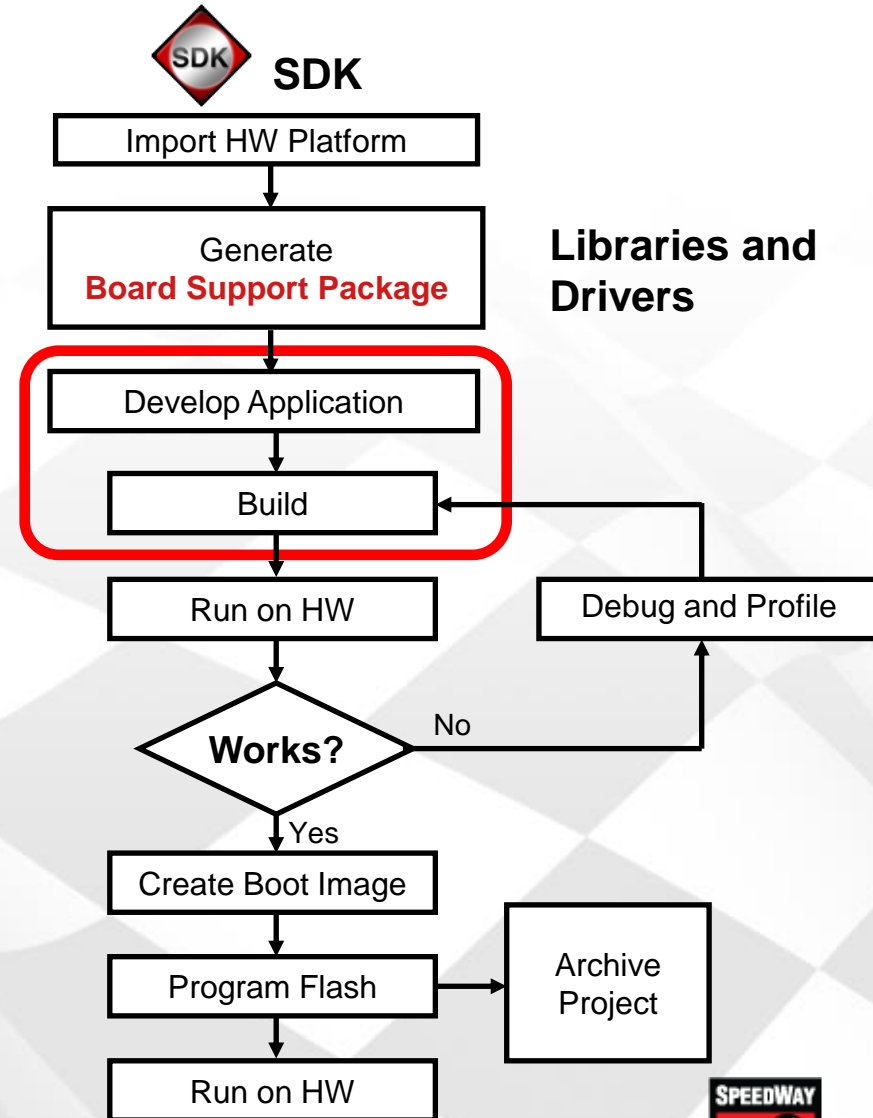
- Responsible for loading OS kernel
- OS image can be sourced from any of the following peripherals
 - NAND
 - NOR
 - QSPI
 - USB
 - GigE
 - SD
- U-boot from Xilinx does not support encryption at this point
- Optionally configure PL in stage 2



Get U-Boot source code from Xilinx's GIT Server
or <http://wiki.xilinx.com/zynq-uboot>

Lab 6 – FSBL

- Create the FSBL
- Analyze the code



Questions

- **What is the size of the FSBL application with the Debug configuration?**
 - ~ 153K bytes
- **What is the target memory for the FSBL?**
 - ps7_ram_0_S_AXI_BASEADDR, which is the 192 KB on-chip RAM
- **What is the size of the FSBL application with the Release configuration?**
 - ~140K bytes which is ~8% smaller
- **In what file was the FSBL main() function?**
 - main.c
- **Which file included all of the ARM register settings? Where did it originate (prior to FSBL generation)?**
 - ps7_init.c
 - The imported hardware platform. Remember Lab 1?

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Create Zynq Boot Image in SDK

- Graphical front end to command line bootgen

- Partitions

1. FSBL
2. Bitstream
3. Application

Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

☐ Create new BIF file ☒ Import from existing BIF file

BIF file path: C:\Speedway\ZynqSW\2013_3\SDK_Workspace\Test_Peripherals\bootimage\Test_Peripherals.bif [Browse]

☐ Use Authentication

Authentication keys:

PPK [Browse] PSK [Browse]

SPK [Browse] SSK [Browse]

SPK Signature [Browse]

☐ Use encryption

Encryption key:

Key file [Browse]

Key store: ☒ BRAM

Part name [Browse]

File Order Does Matter!

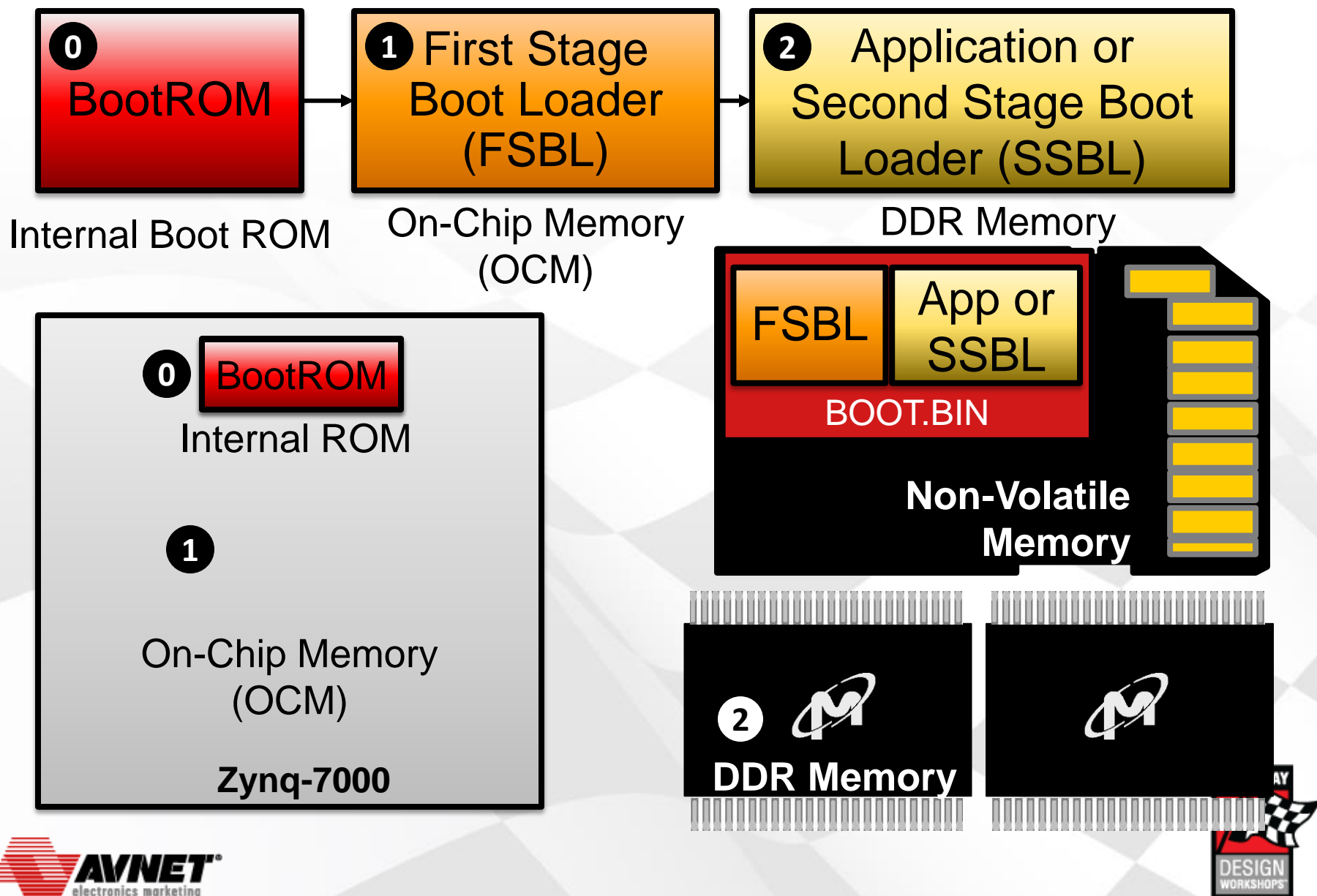
Boot image partitions

File path	Encrypted	Authenticated
(bootloader) C:\Speedway\ZynqSW\2013_3\SDK_Workspace\zynq_fsb0\Release\zynq_fsb0.elf	none	none
C:\Speedway\ZynqSW\2013_3\SDK_Workspace\ZynqHW\Z_system_wrapper.bit	none	none
C:\Speedway\ZynqSW\2013_3\SDK_Workspace\Test_Peripherals\Release\Test_Peripherals.elf	none	none

Output path: C:\Speedway\ZynqSW\2013_3\SDK_Workspace\Test_Peripherals\bootimage\output.bin [Browse]

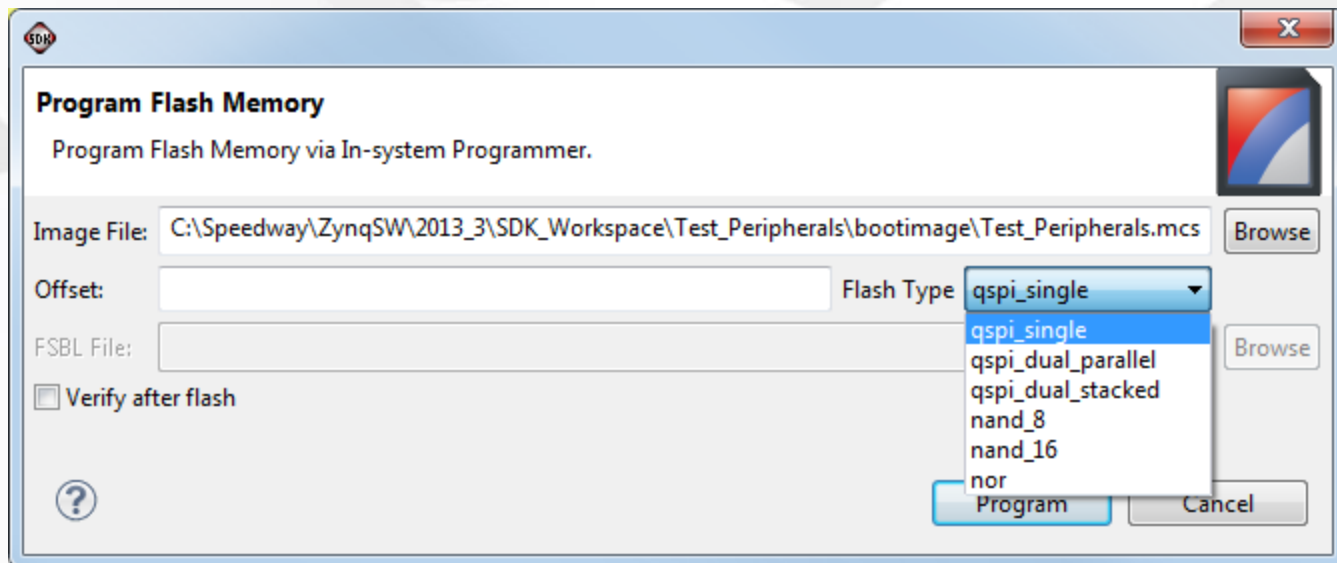
[?] [Create Image] [Cancel]

Boot Process Overview for Zynq



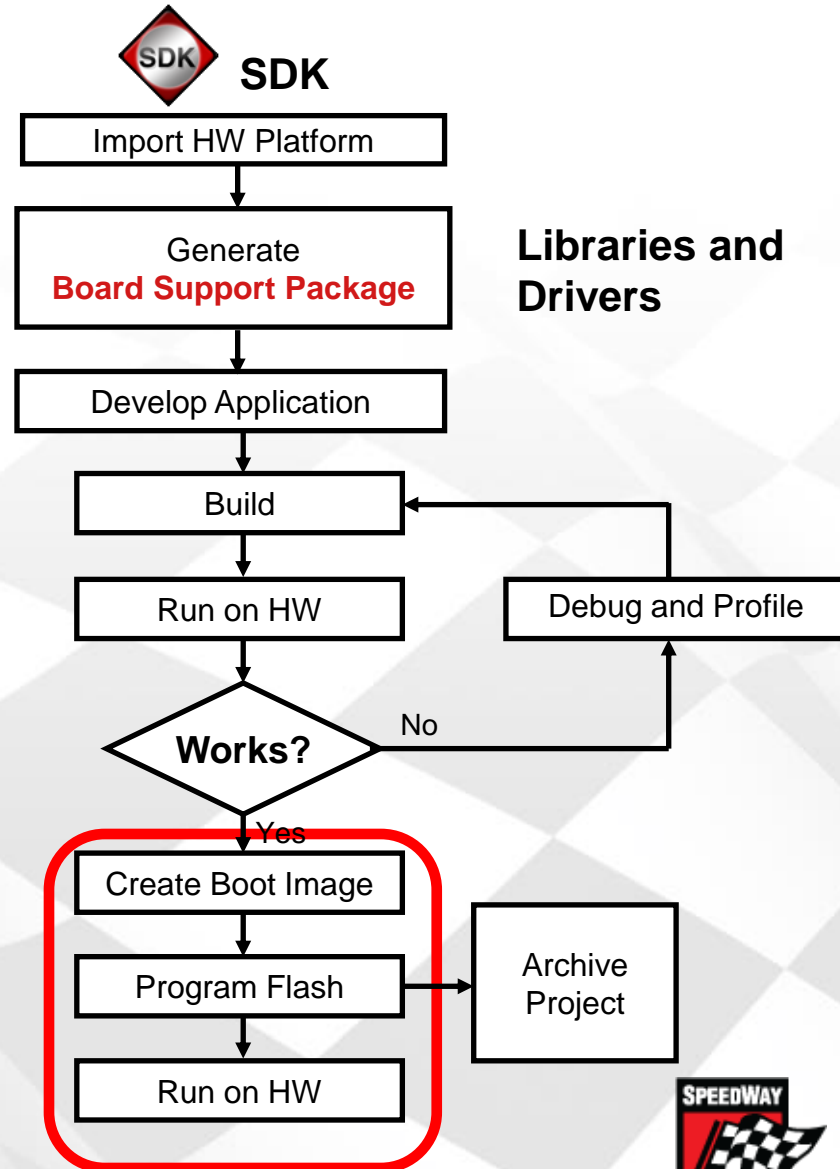
SDK Flash Programming Utility

- **Supports multiple non-volatile types**
 - Single QSPI will be used in the labs
- **Interprets MCS-formatted file**
- **Offset and Verify options available**



Lab 7 – Flash Programming and Configuration

- Create the boot images
- Copy to QSPI
- Boot from QSPI
- Copy to SD Card
- Boot from SD Card



Questions

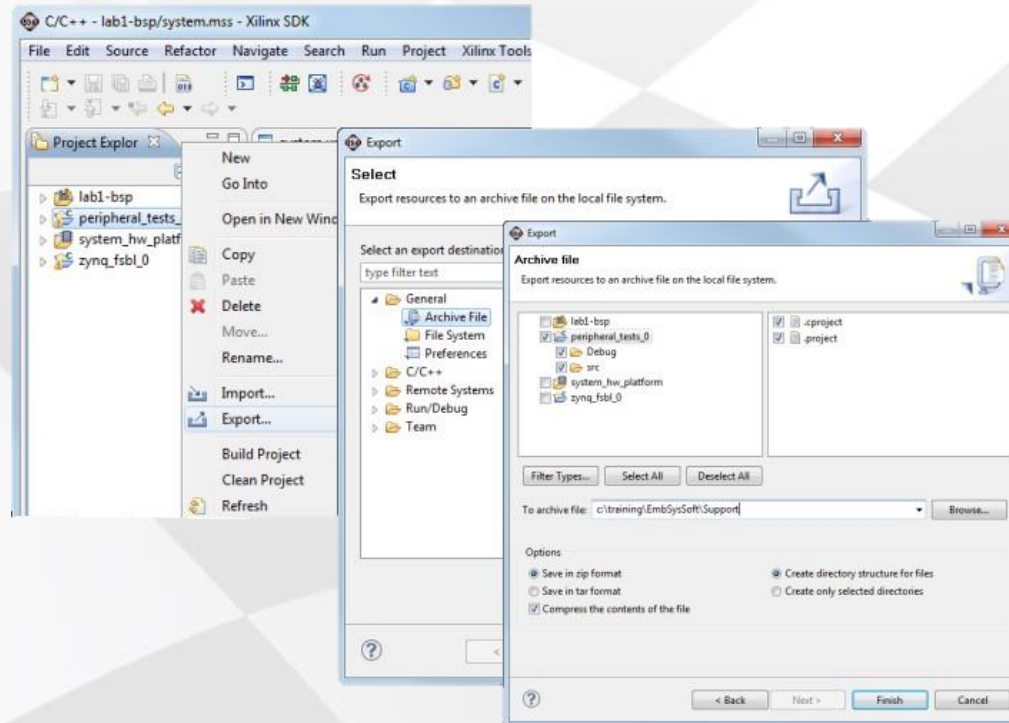
- **Is the order of the boot images critical? If so, list the order.**
 - YES! FSBL ELF, then bitstream, then application ELF
- **True or False? The Create Zynq Boot Image tool creates both the QSPI and SD card images in one operation.**
 - False. SDK used to operate that way, but in 2013.3, it requires two separate operations.
- **What is the required name for the .bin file copied to the SD card?**
 - boot.bin

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

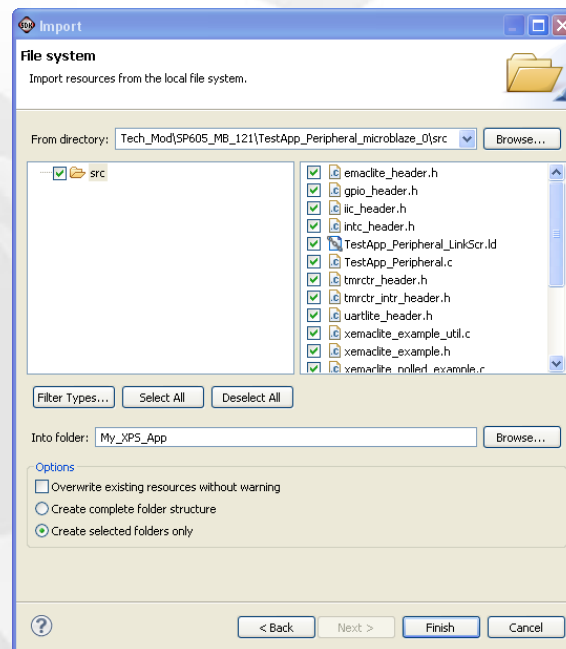
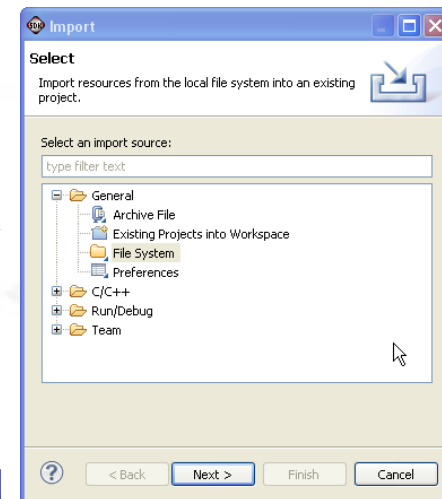
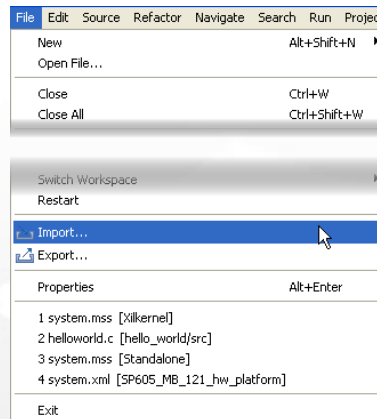
Sharing SDK Workspaces and Projects

- SDK workspaces are location and directory sensitive
- *Do not* move workspaces via Windows navigation
- Two ways to export and import with SDK
 - Archive file – zip and tar formats
 - Directory structure



Importing C/C++ Files

- **Create an Empty Application project**
 - Set HW platform and BSP as needed
- **Select Import in the File Menu**
- **Select File System option under General import source type**
- **Select the C/C++ project**
 - Directories or individual files

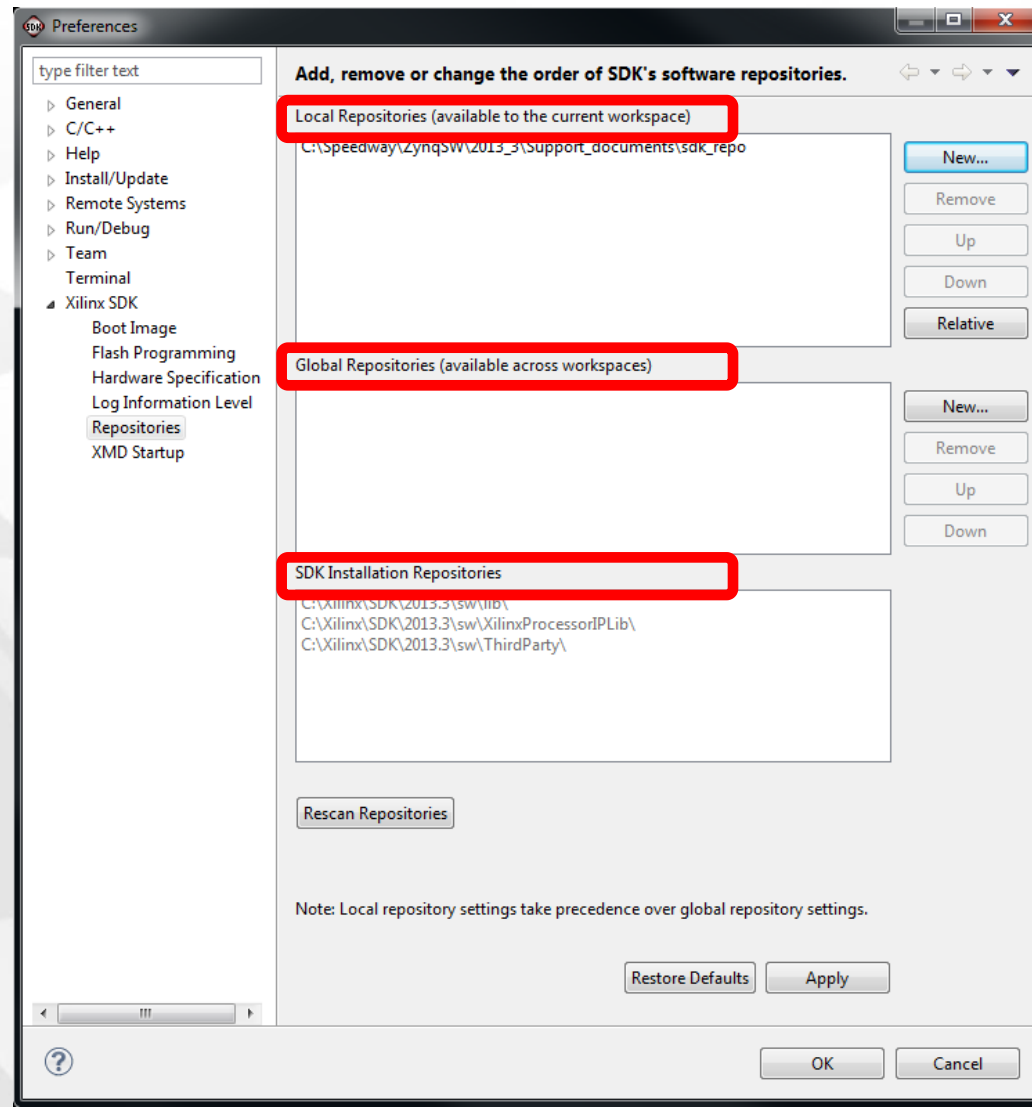
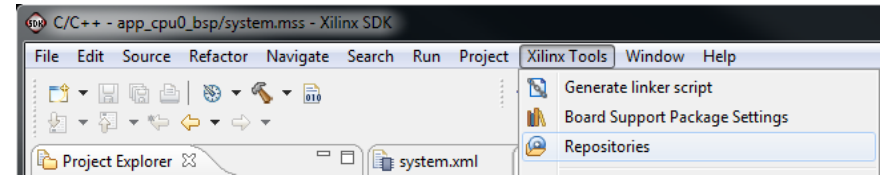


Software Repositories

- **May include**

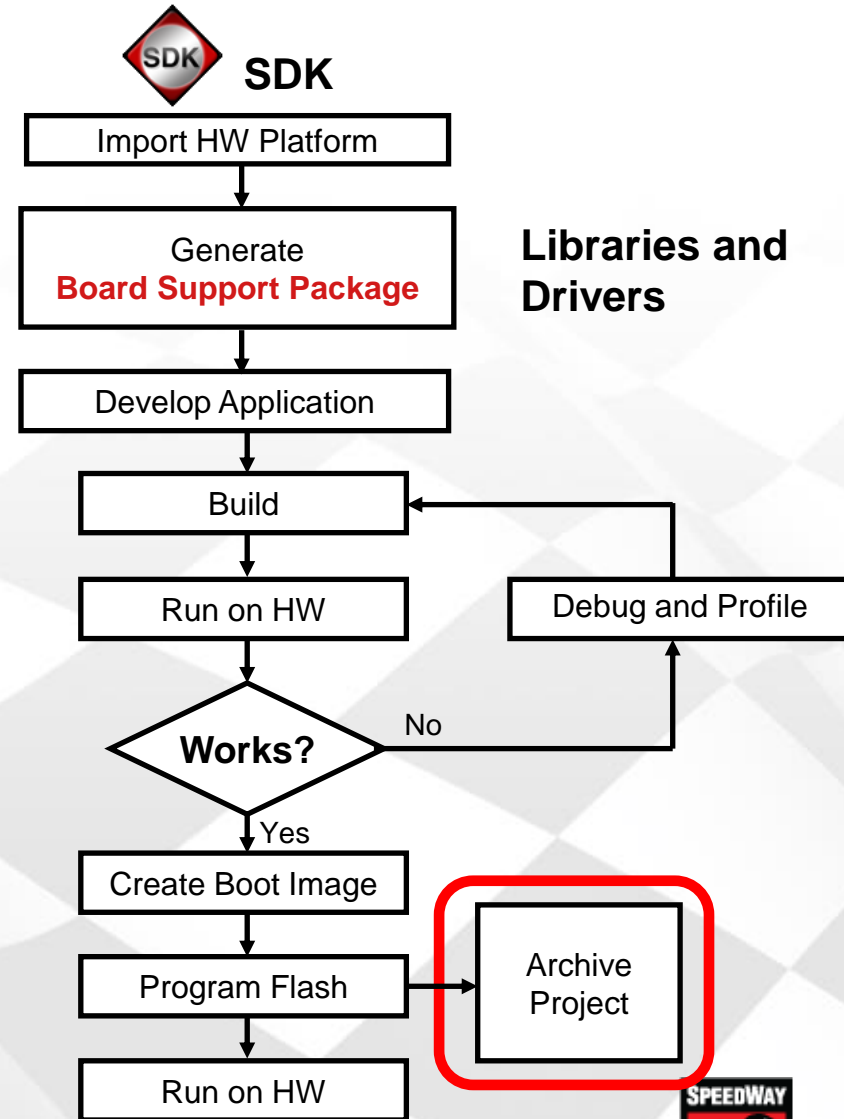
- Custom Drivers
- Custom Libraries
- Custom Board Support Packages
- Could be a modified Xilinx driver or library
- Repository sub-directories
 - bsp
 - drivers
 - sw_services

- **Prioritized**



Lab 8 – Project Management

- **Export your complete workspace**
 - Hardware Platform
 - BSP
 - All applications
- **Create a new Workspace**
- **Re-create your previous Workspace in the new Workspace**



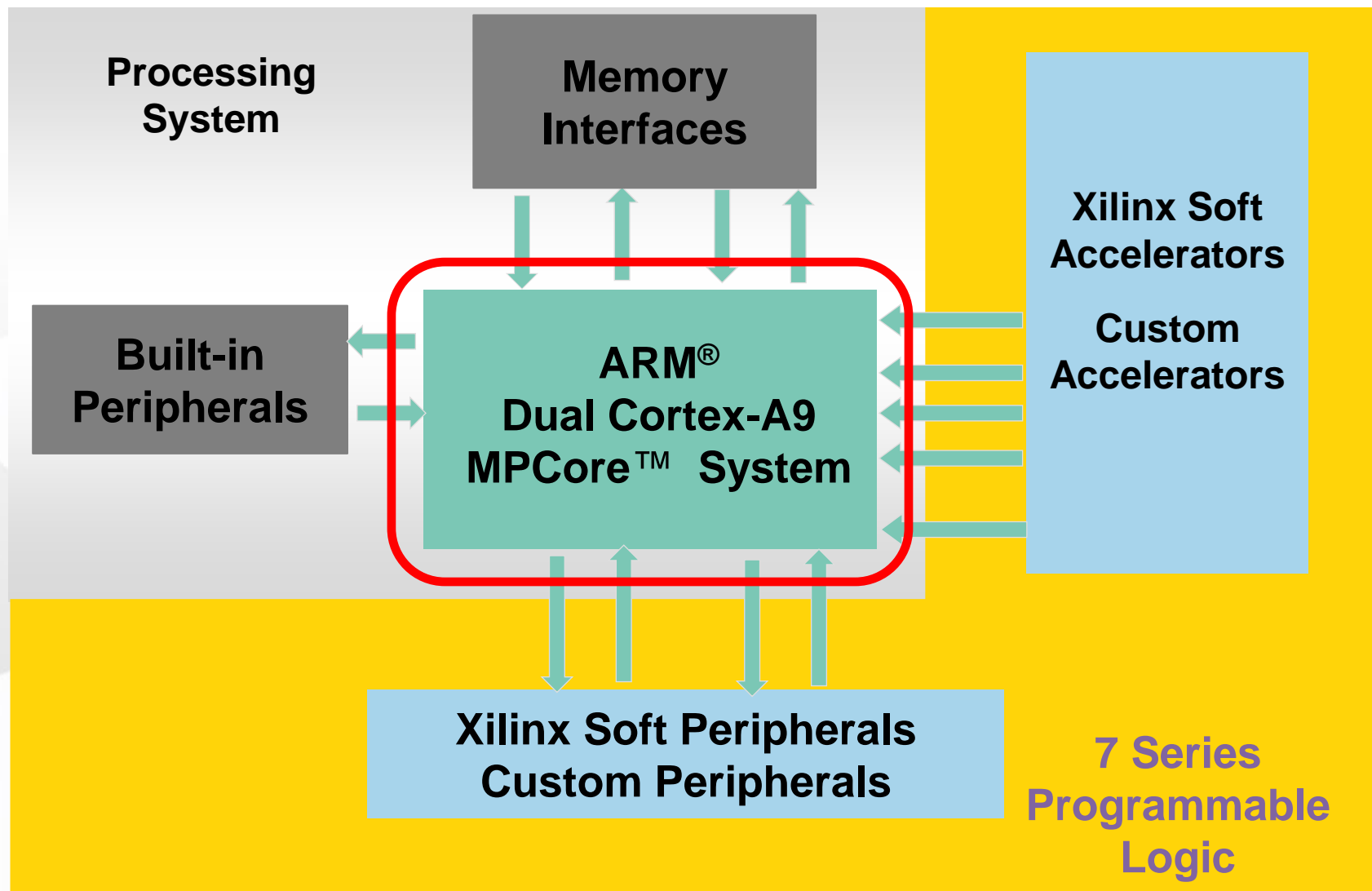
Questions

- **What is the advantage of exporting your Workspace items as opposed to simply zipping the workspace?**
 - If you zip and share your SDK workspace, there is a very good chance that it will not work when opened again. The SDK workspace is full of absolute paths, so unless the recipient unzips the SDK workspace to the exact same location, it won't fully work. It might appear to work initially, but it is likely not going to build properly.
 - If you export the workspace, it is fully transportable.

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Dual Processor Cores



Multiple Core Processing Considerations

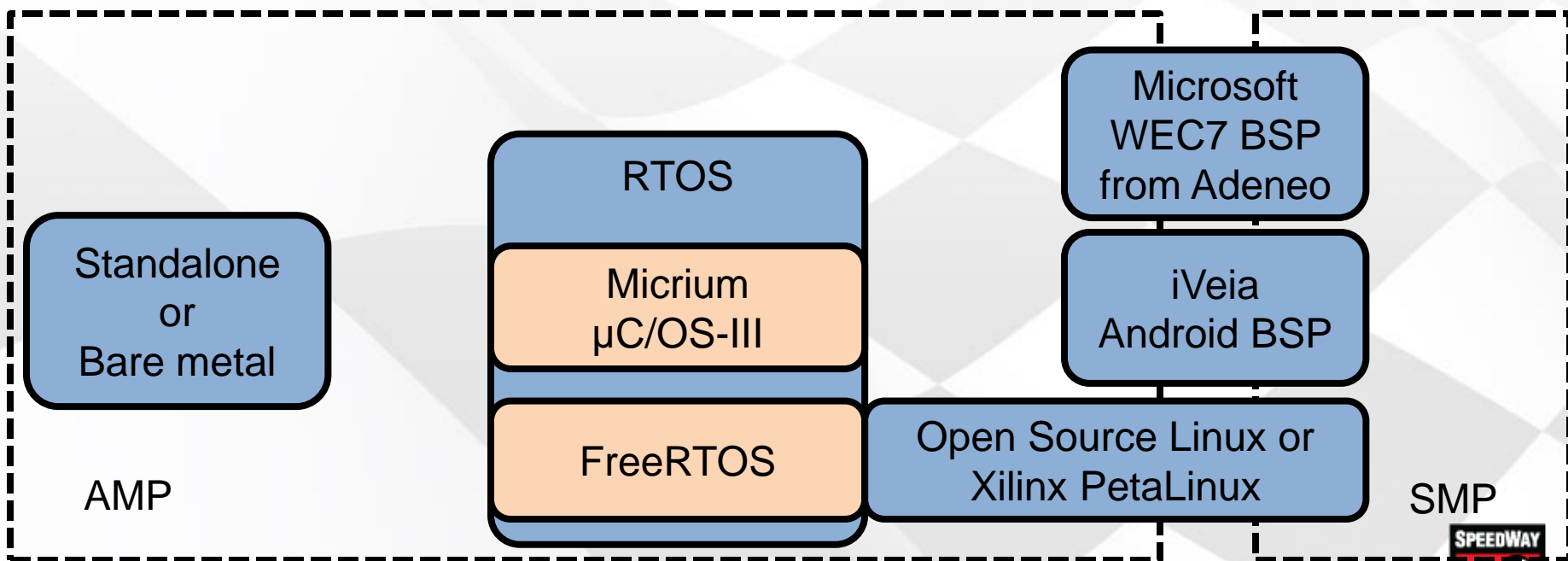
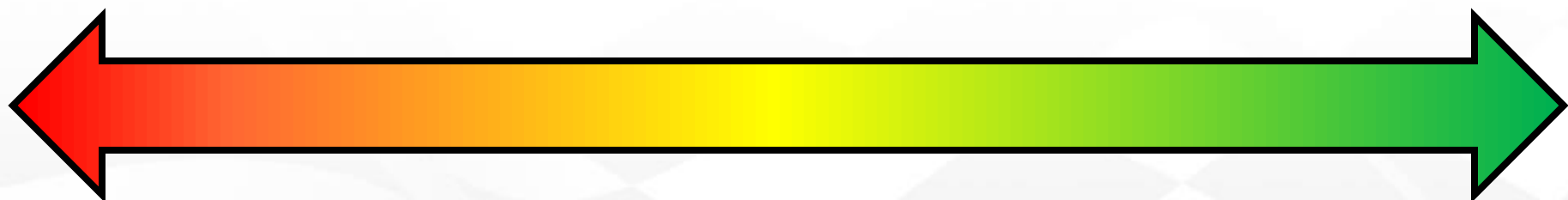
- **Cortex™-A9 processor cores in Zynq are identical and symmetrical**
 - Each has a read-only processor ID register to identify as CPU0 or CPU1
- **Both processors share the same address map**
 - Special-purpose registers exist for each processor
 - DDR memory and peripherals are shared so simultaneous access must be managed appropriately
- **Each core is independently managed resource**
 - Unused processor can be held in sleep mode to reduce power

Choosing the Optimal Dual Core Processing Strategy

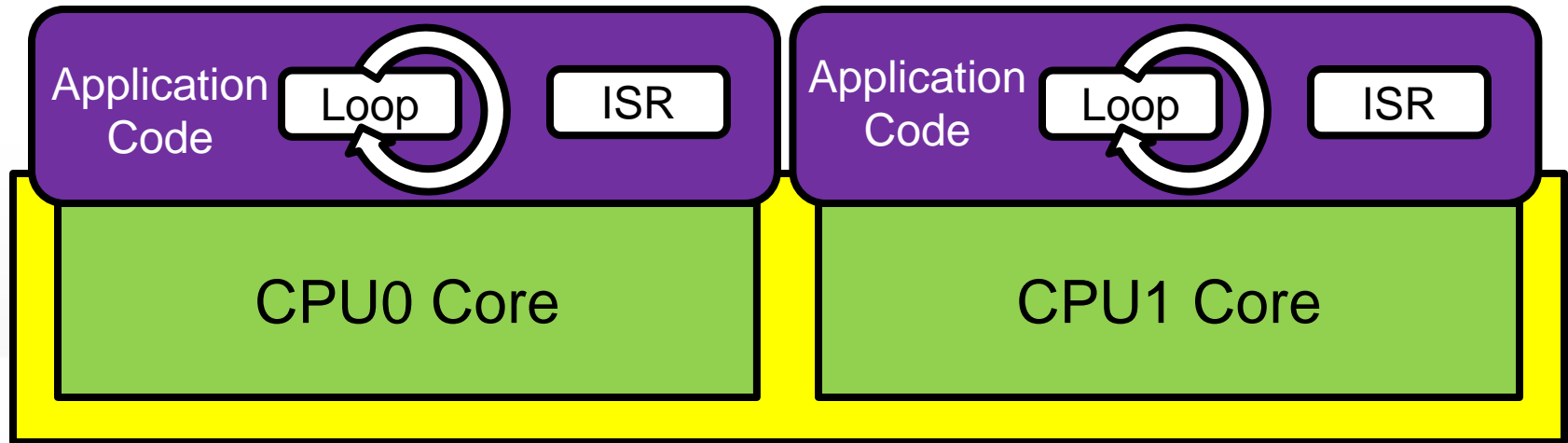
- What are your software application requirements?

Real-Time Performance

High System Performance

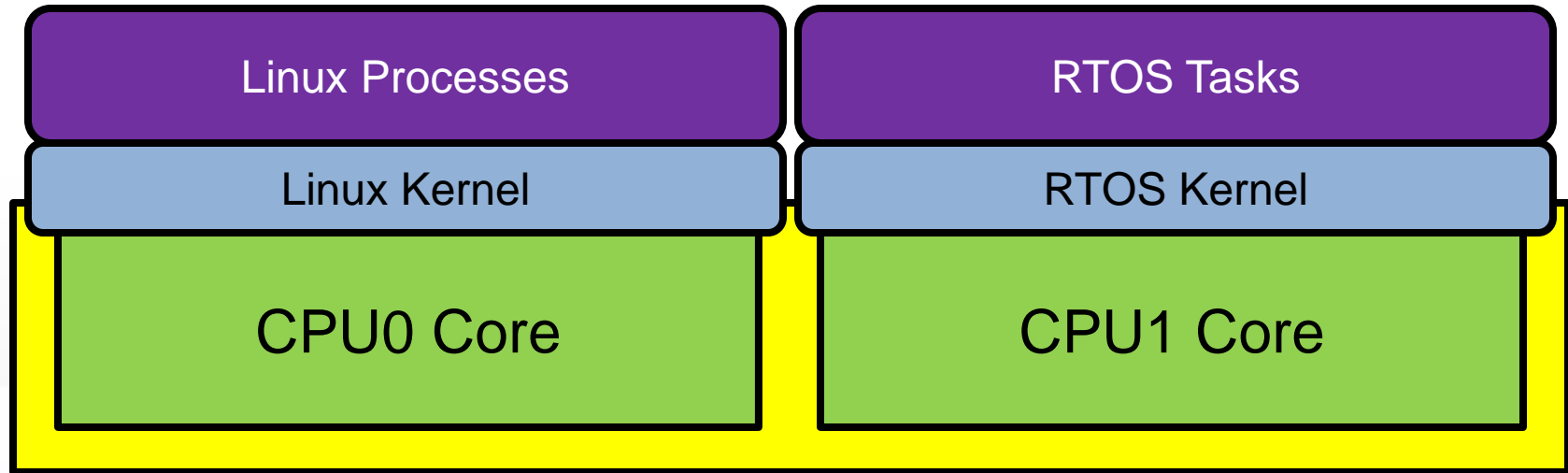


Standalone or 'Bare Metal' AMP Solution



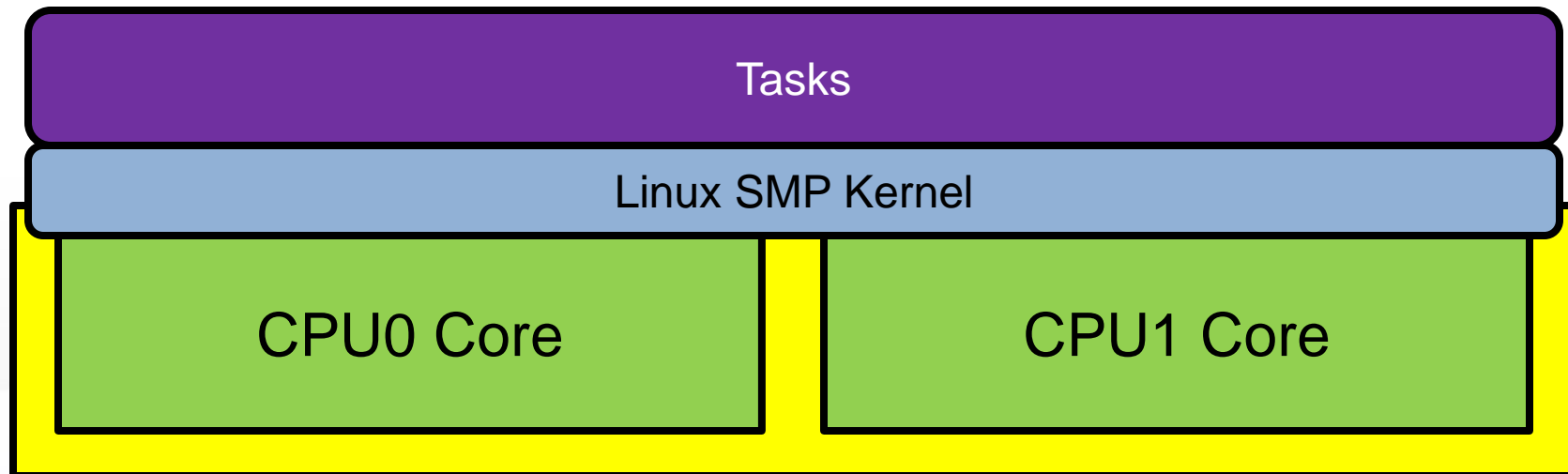
- Preferred if software tasks are simple and repetitive
- No operating system required
 - Bypass Memory Management Unit (MMU)
 - Directly access hardware memory space
- Tight control over execution but limited in functionality
- Custom inter-processor communication or synchronization mechanism needed

Linux and RTOS AMP Solution



- **The RTOS manages real time performance factors offering determinism for real time applications**
 - Interrupt latency
 - Scheduling latency
 - Kernel service timing
- **Linux Kernel and RTOS Kernel must use same inter-process communications framework**

Linux SMP Solution



- **Support for high system performance applications**
 - Symmetric multiprocessing
 - Shared libraries
 - Device drivers
 - Memory management
 - TCP/IP networking
- **Zynq Linux kernel configured for SMP mode by default**

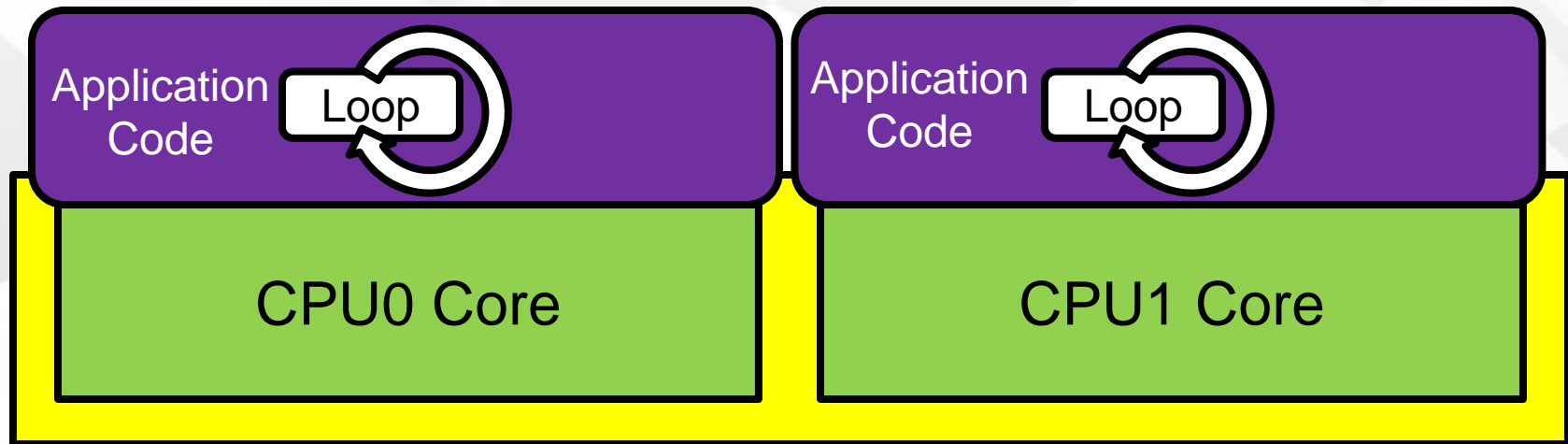
Processor Boot Sequence

- BootROM operation on CPU0 always starts execution from power on reset
- BootROM forces CPU1 into a wait for event (WFE) low-power state
- BootROM loads FSBL into OCM RAM and then begins execution on CPU0
- FSBL loads CPU1 software application into DDR memory
- CPU0 application wakes up CPU1 with one of two possible methods
 - Set event to awaken CPU1 from WFE state
 - Perform soft reset on CPU1

Refer to XAPP1079 - *Simple AMP Bare-Metal System Running on Both Cortex-A9 Processors*

Lab 9 – Dual Processors

- Boot Zynq with an image to run two processors
- Each core runs a separate standalone application executable
- Explore SDK debug view of each core



Questions

- **What is the difference between a normal standalone application running on CPU0 and an AMP application running on CPU0?**
 - There is not necessarily a difference between these two applications other than the necessity of the CPU0 application to manage access to resources shared with CPU1. This can be done through either resource arbitration (such as the DDR) or through inter process signaling (such as a semaphore) in order to avoid contention between the CPUs.
- **Why are two separate application projects required for AMP applications?**
 - Since Zynq shares the DDR memory space between the two CPU cores, each application needs to have its own memory space defined in separate linker scripts and built into the executable at link time.
- **What defines the order in which the applications must be loaded to each of the processor cores?**
 - This is a function of both the Zynq BootROM and the FSBL. The FSBL is an SDK application for which the Xilinx source code is provided. Since you have access to the source code, you could modify the FSBL to alter the AMP application load order if you should desire this behavior for your Zynq end application.
- **Did the memory view contents display what you expected when execution is paused at one of the breakpoints on one core but you are viewing the execution context of the other core which is still running?**
 - Execution context data, such as memory and register content cannot be updated on the fly with typical debugger technology such as TCF. The CPU must be suspended either with a Suspend JTAG instruction or paused at a breakpoint in order for the context information to be updated within the debug session.

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Interrupts and Exceptions

- **Hardware interrupt**

- An asynchronous signal from hardware, originating outside the processor core, indicating a peripheral's need for attention



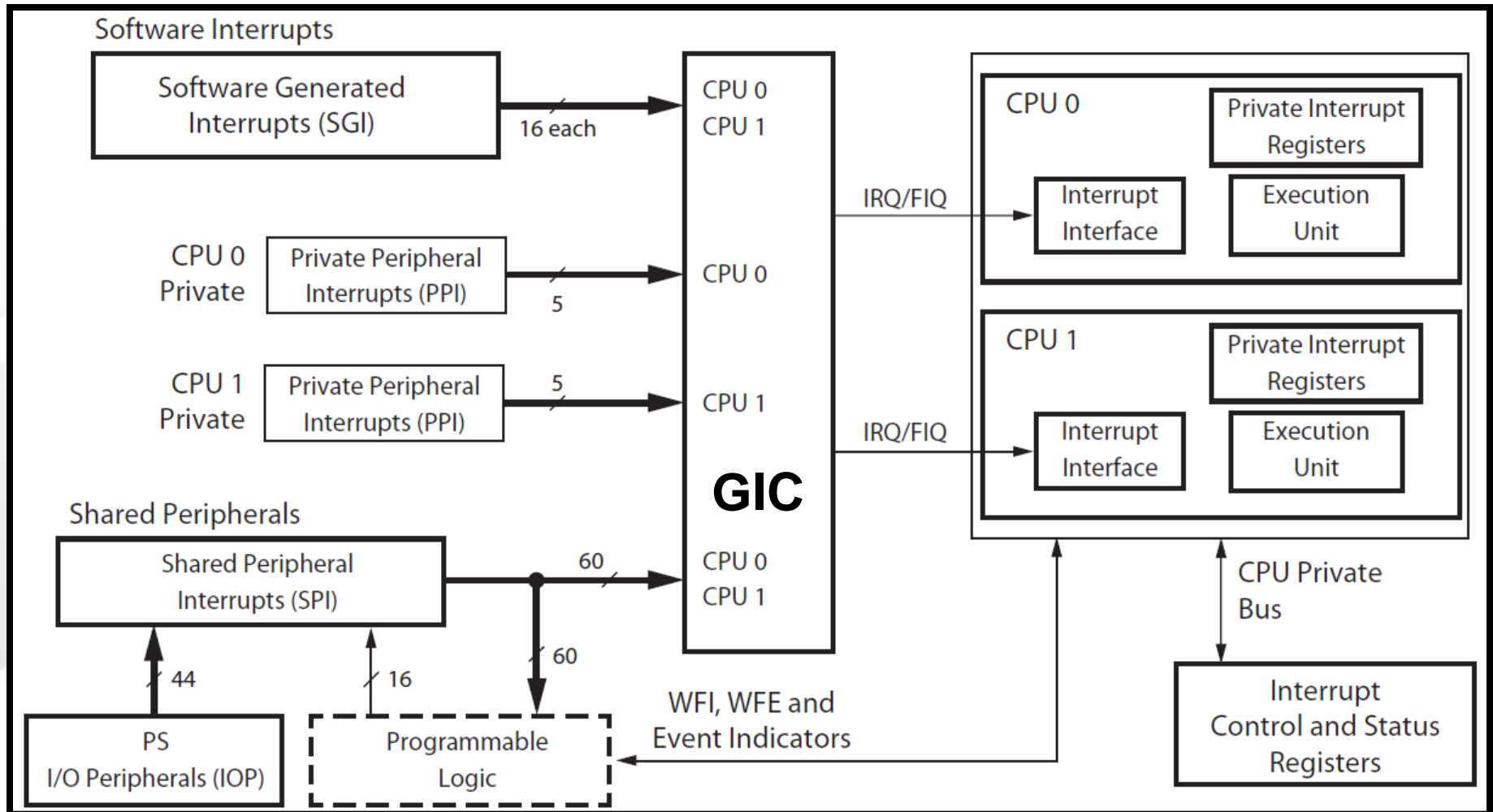
Professor I. M. Boring

- **Software Interrupt**

- A synchronous event in software indicating the need for a change in execution, also referred to as an exception

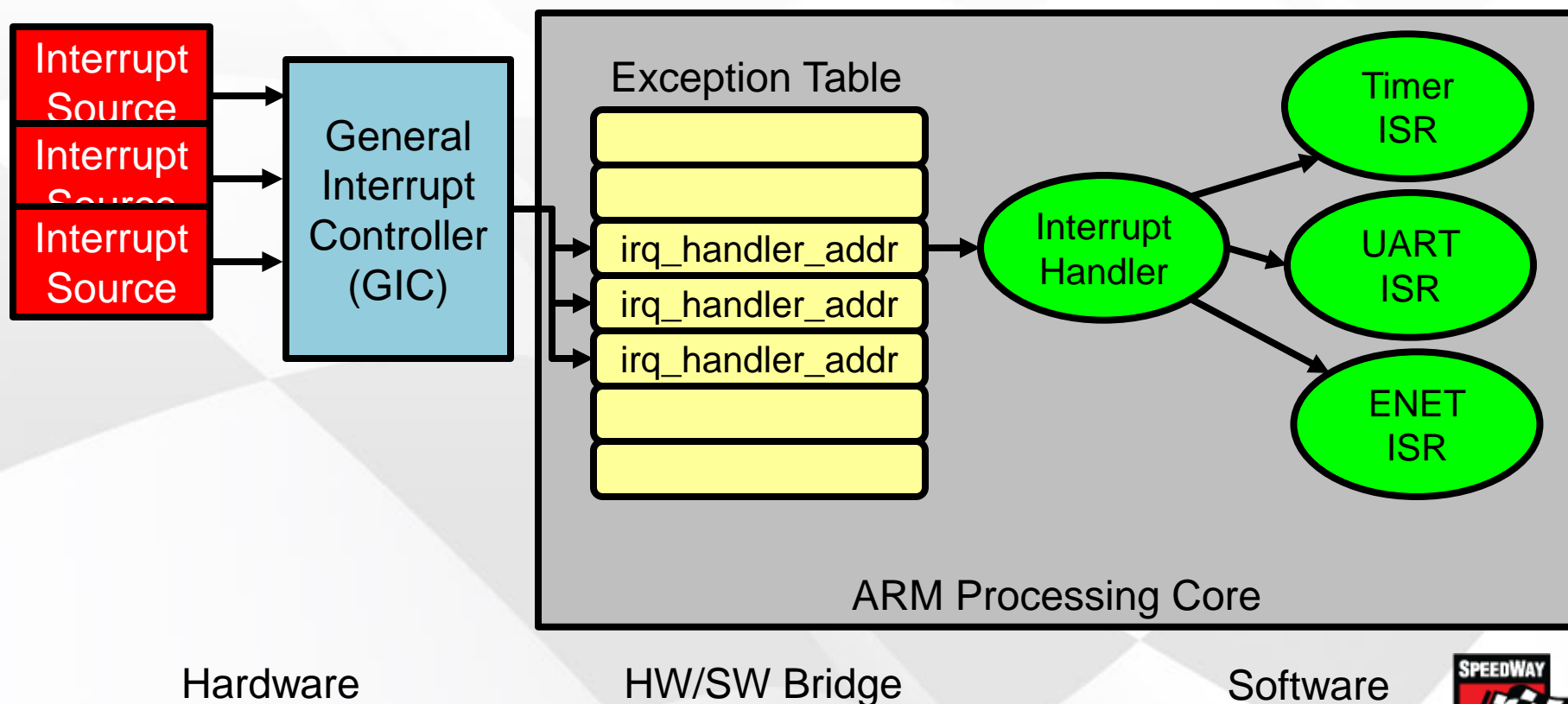
- **Occurring events which need immediate responses**

Zynq Interrupts - Generic Interrupt Controller (GIC)



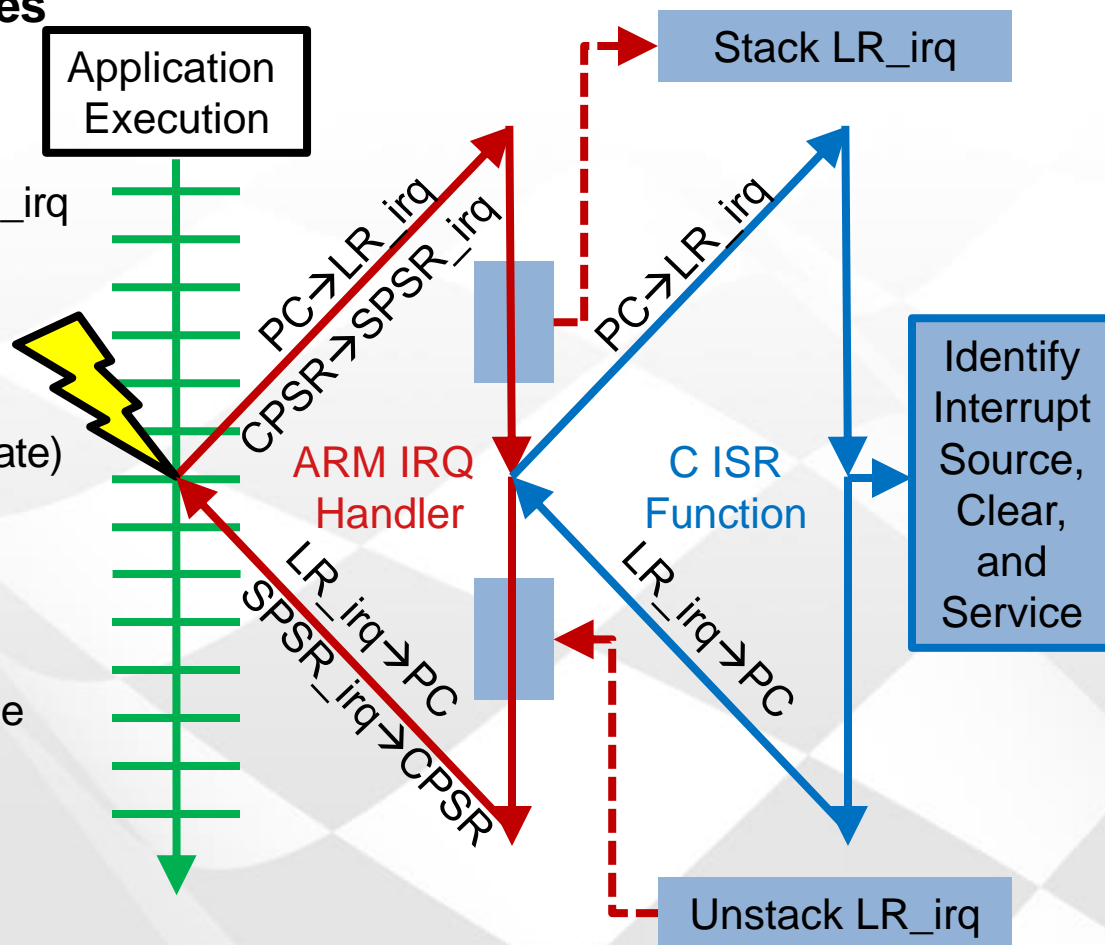
Interrupt Handling

- When an interrupt occurs, an Interrupt Handler should respond to the event by calling the appropriate ISR



Process of Handling an Interrupt

1. **Interrupt is received, and current executing instruction completes**
2. **Save processor status**
 - Copies CPSR into SPSR_irq
 - Stores the return address in LR_irq
3. **Change processor status for exception**
 - Mode field bits
 - Interrupt disable bits (if appropriate)
 - Sets PC to vector address
4. **Execute top-level exception handler**
 - Top-level handler branches to the appropriate registered ISR
5. **Return to main application**
 - Restore CPSR from SPSR_irq
 - Restore PC from LR_irq
 - When re-enabling interrupts change to system mode (CPS)

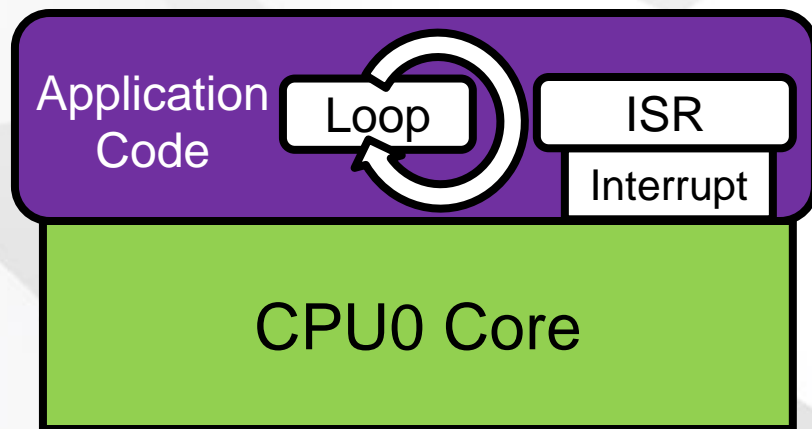


ISR Tips and Tricks

- **Keep the code short and simple; ISRs can be difficult to debug**
- **Do not allow servicing of other interrupts while in the ISR**
- **Time is of the essence!**
 - Spend as little time as possible in the ISR
 - Do not perform tasks that can be done in the background
 - Use flags to signal background functions
- **Make use of the callback argument passed in registration**
- **Make use of the provided interrupt support functions when using IP drivers**
- **Be sure to enable interrupts when leaving the registered handler or ISR**

Lab 10 – Interrupts

- Add an ISR to an existing application
- Register the ISR and enable the related interrupt source



Questions

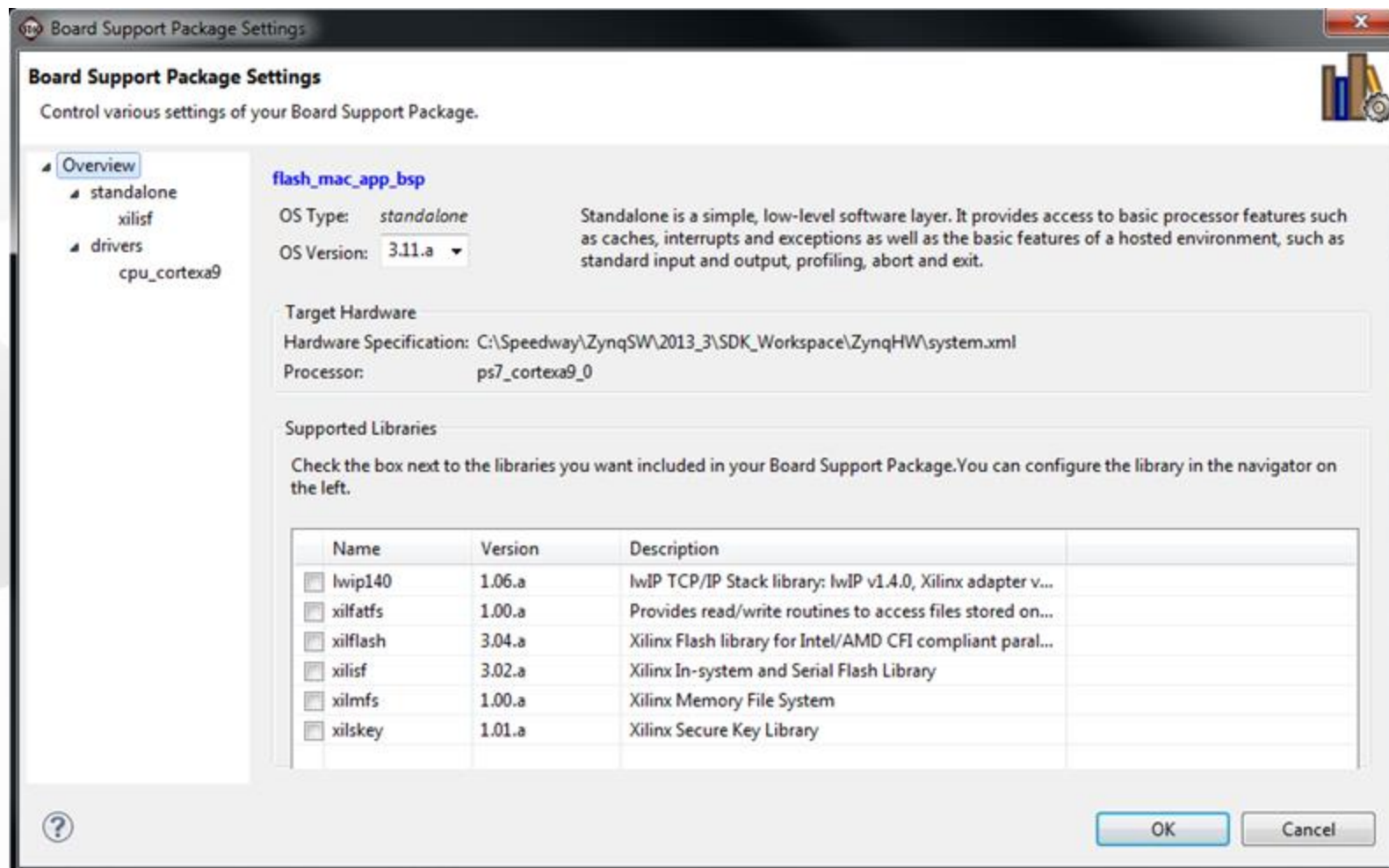
- **What happens when a value outside of the requested range of 0-9 is entered?**
 - The value is accepted, multiplied by the brightness clock multiplier and assigned to the PWM hardware block.
- **Are interrupts currently enabled in the example application provided?**
 - Hardware interrupts are currently provided from the PWM hardware block which are then routed to the GIC in the hardware platform. However, the software application at this point does not configure the GIC to allow the application execution to be interrupted so the software end of the interrupt is not currently enabled.
- **When does the function PWMIsr() get called?**
 - The PWMIsr() is the interrupt service routine callback function. It is registered with the interrupt subsystem as the callback interrupt handler through the call to XScuGic_Connect() during our interrupt setup.
- **GROUP DISCUSSION QUESTION – We handled the ISR functionality inside the ISR, what is another way of doing this?**
 - Alternatively, we could have the ISR just set a flag and then wait for the main function process the flag and write a message to the console.

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

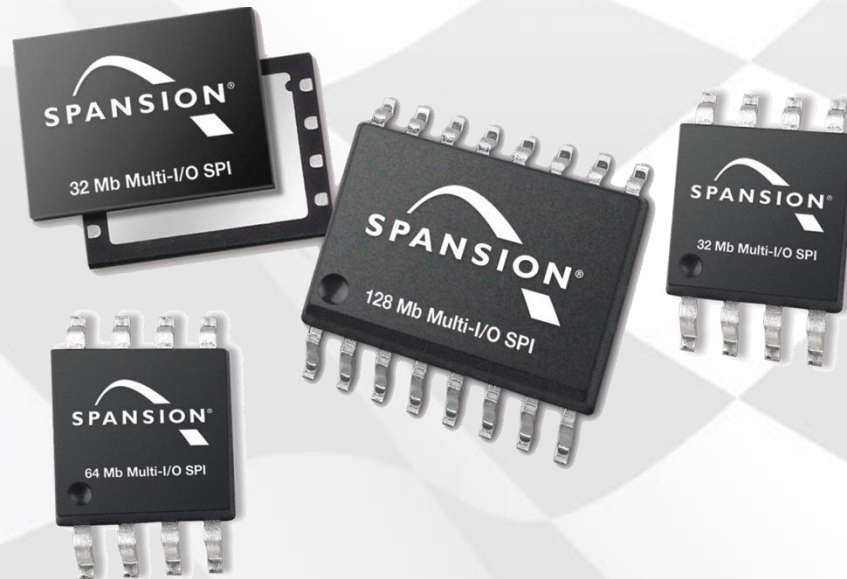
SDK BSP - Supported Libraries

- BSP settings allow for system access library options



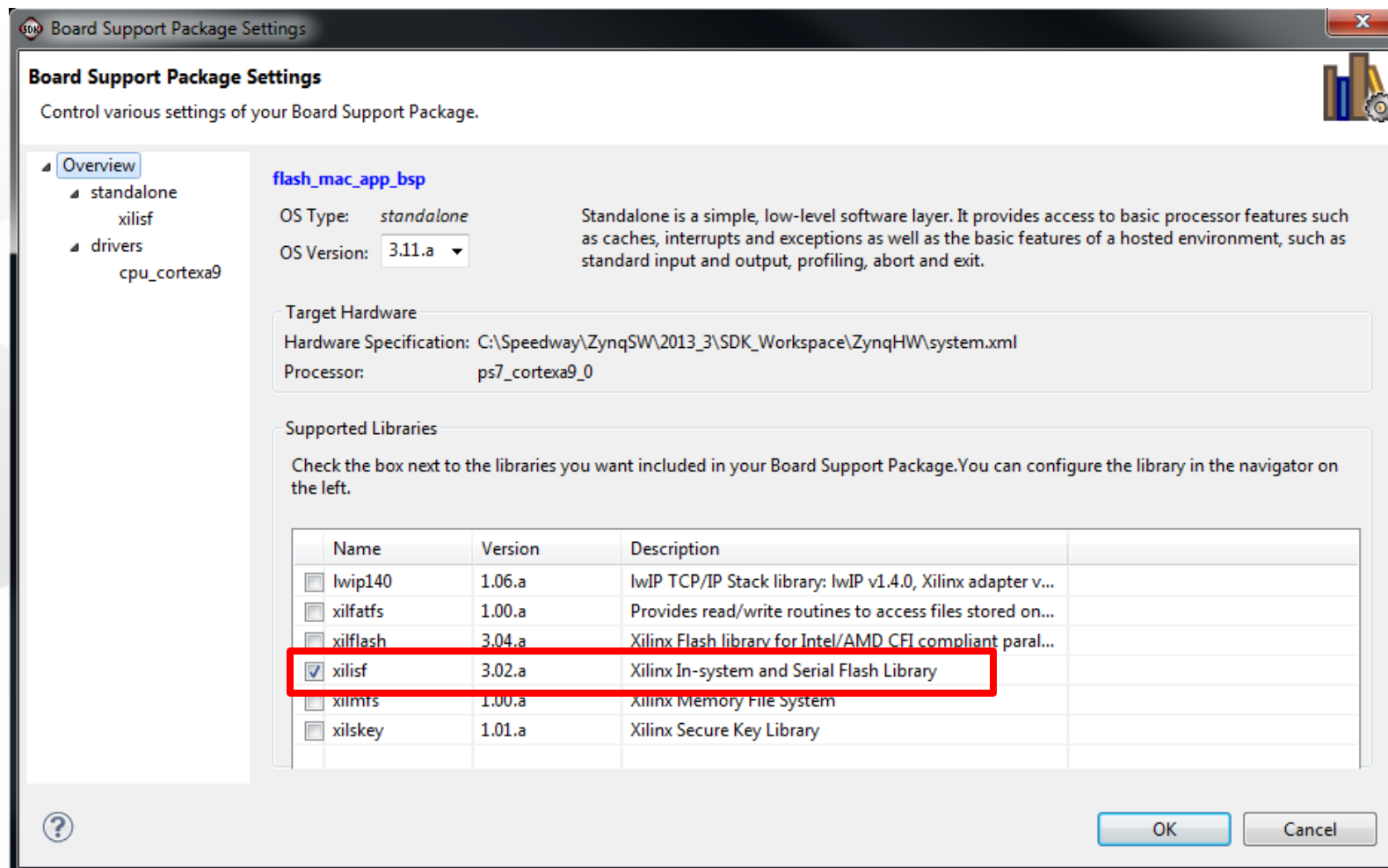
System Access Libraries

- **Xilinx provides several important system access libraries**
 - Xilinx memory file system: **xilmfs**
 - Xilinx networking support: **lwip140**
 - TCP /IP stack library
 - Parallel flash Common Flash Interface (CFI) support: **xilflash**
 - In-system serial flash support: **xilisf**
 - Support for Spansion QSPI flash devices



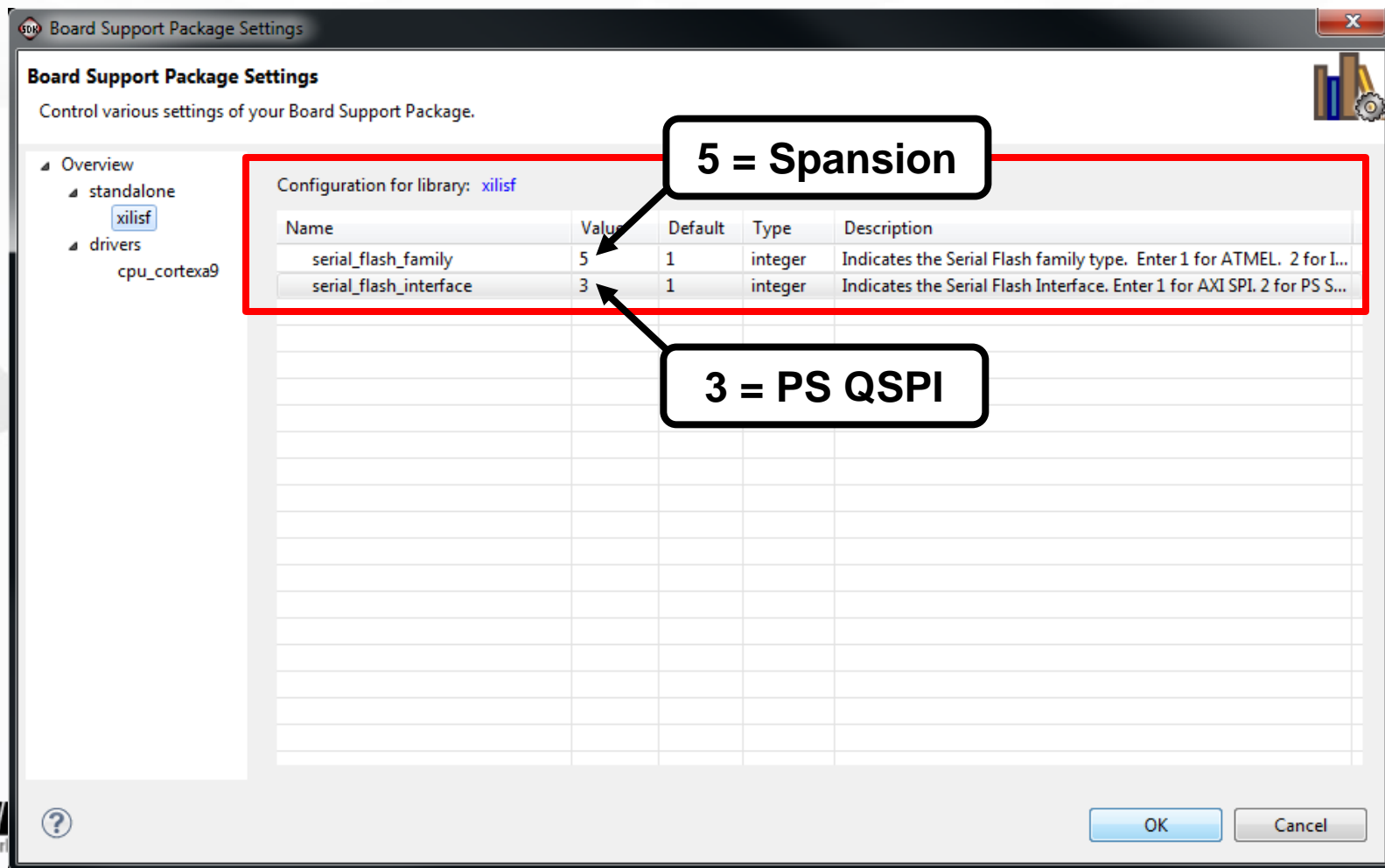
Adding xilif Library Support to a BSP

- Select xilif library within BSP settings



Configuring the xilif Library

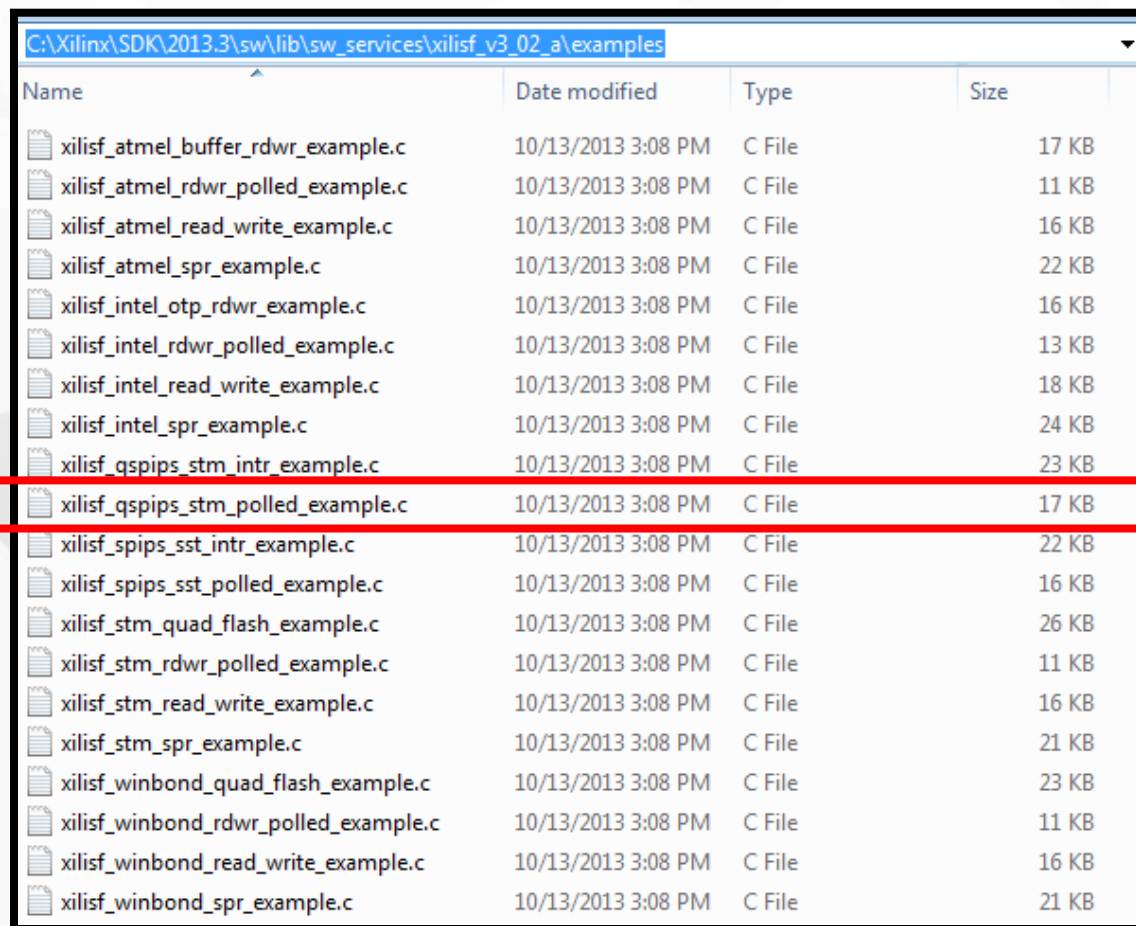
- Initialize xilif parameter types
 - Flash family type
 - Flash interface type



Leverage Example Code for Libraries

- Example code found within the Xilinx xilif library folder

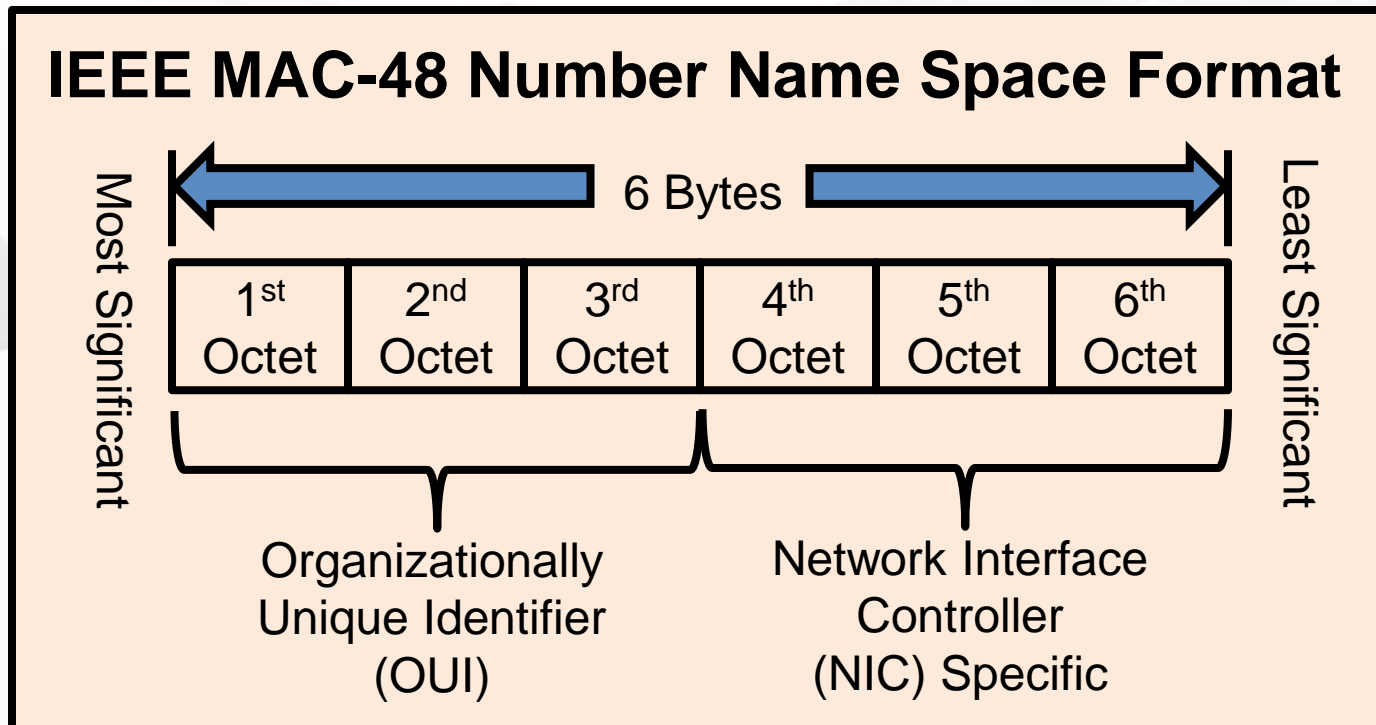
C:\Xilinx\SDK\2013.3\sw\lib\sw_services\xilif_v3_02_a\examples\



Name	Date modified	Type	Size
xilif_atmel_buffer_rdwr_example.c	10/13/2013 3:08 PM	C File	17 KB
xilif_atmel_rdwr_polled_example.c	10/13/2013 3:08 PM	C File	11 KB
xilif_atmel_read_write_example.c	10/13/2013 3:08 PM	C File	16 KB
xilif_atmel_spr_example.c	10/13/2013 3:08 PM	C File	22 KB
xilif_intel_otp_rdwr_example.c	10/13/2013 3:08 PM	C File	16 KB
xilif_intel_rdwr_polled_example.c	10/13/2013 3:08 PM	C File	13 KB
xilif_intel_read_write_example.c	10/13/2013 3:08 PM	C File	18 KB
xilif_intel_spr_example.c	10/13/2013 3:08 PM	C File	24 KB
xilif_qspips_stm_intr_example.c	10/13/2013 3:08 PM	C File	23 KB
xilif_qspips_stm_polled_example.c	10/13/2013 3:08 PM	C File	17 KB
xilif_spips_sst_intr_example.c	10/13/2013 3:08 PM	C File	22 KB
xilif_spips_sst_polled_example.c	10/13/2013 3:08 PM	C File	16 KB
xilif_stm_quad_flash_example.c	10/13/2013 3:08 PM	C File	26 KB
xilif_stm_rdwr_polled_example.c	10/13/2013 3:08 PM	C File	11 KB
xilif_stm_read_write_example.c	10/13/2013 3:08 PM	C File	16 KB
xilif_stm_spr_example.c	10/13/2013 3:08 PM	C File	21 KB
xilif_winbond_quad_flash_example.c	10/13/2013 3:08 PM	C File	23 KB
xilif_winbond_rdwr_polled_example.c	10/13/2013 3:08 PM	C File	11 KB
xilif_winbond_read_write_example.c	10/13/2013 3:08 PM	C File	16 KB
xilif_winbond_spr_example.c	10/13/2013 3:08 PM	C File	21 KB

Lab 11 – Serial Flash Library

- Use xilidf library to access flash memory to store a system configuration parameter
- Read data from a piece of memory
- Write data to a piece of memory



Questions

- **What if you wanted to reuse part of this application on your own Zynq design but you prefer to use Atmel Flash instead of Spansion Flash?**
 - Avnet also distributes Atmel devices and the only part of this software application that would need to be changed is the `serial_flash_family` field from Experiment 1, Step 10.
- **Why is a separate BSP created for this application? Why not reuse the `standalone_bsp_0` from earlier lab activities?**
 - A separate BSP was created since the Xilinx `xilisf` library was to be added specifically to support this application. We could have instead modified the `standalone_bsp_0` to add the `xilisf` library but that would cause the library code to become linked into our other standalone applications which rely upon `standalone_bsp_0` project as the BSP. This would unnecessarily increase the code size of those other standalone applications which might not be desirable.
- **How were the functions within the application code which access the flash written? Was it written entirely from scratch?**
 - Not really, the flash access functions found within the `xilisf_qspips_flash_polled.c` source file are adapted from the example code found within the Xilinx `xilisf` library folder

C:\Xilinx\SDK\2013.3\sw\lib\sw_services\xilisf_v3_02_a\examples\

Agenda

Lecture	Description	Labs
Chapter 1	Zynq System Architecture Basics	Lab 1
Chapter 2	Xilinx SDK Overview	Lab 2
Chapter 3	Standalone Board Support Package	Lab 3
Chapter 4	Developing Applications	Lab 4
Chapter 5	Connecting Hardware and Debugging	Lab 5
Chapter 6	First Stage Boot Loader	Lab 6
Chapter 7	Flash Programming and Boot-up	Lab 7
Chapter 8	SDK Project Management	Lab 8
Chapter 9	Dual Processors	Lab 9
Chapter 10	Interrupts	Lab 10
Chapter 11	Xilinx Libraries	Lab 11
Chapter 12	Next Steps	

Comprehensive Software Ecosystem Support



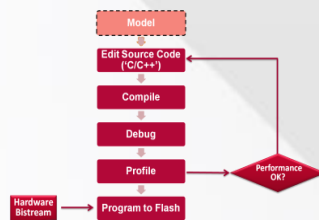
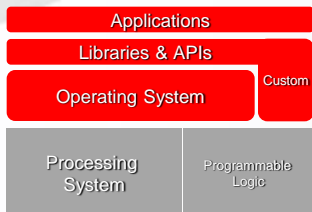
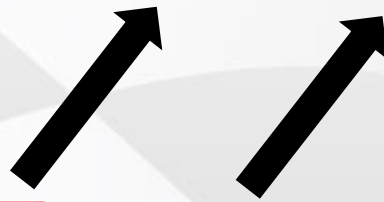
WIND RIVER

ENEAA



expresslogic

Micrium



ARM Ecosystem enables comprehensive OS & Tools support



Avnet Embedded Software Store



Xilinx® SpeedWay Design Workshops™

Featuring MicroZed™, ZedBoard™ and Vivado® Design Suite



<http://www.embeddedsoftwarestore.com/>

Vendors Supporting



USB
File Systems



Open
Source



JTAG
Tools



Embedded
SSL



DSP
Libraries



BSPs
Training



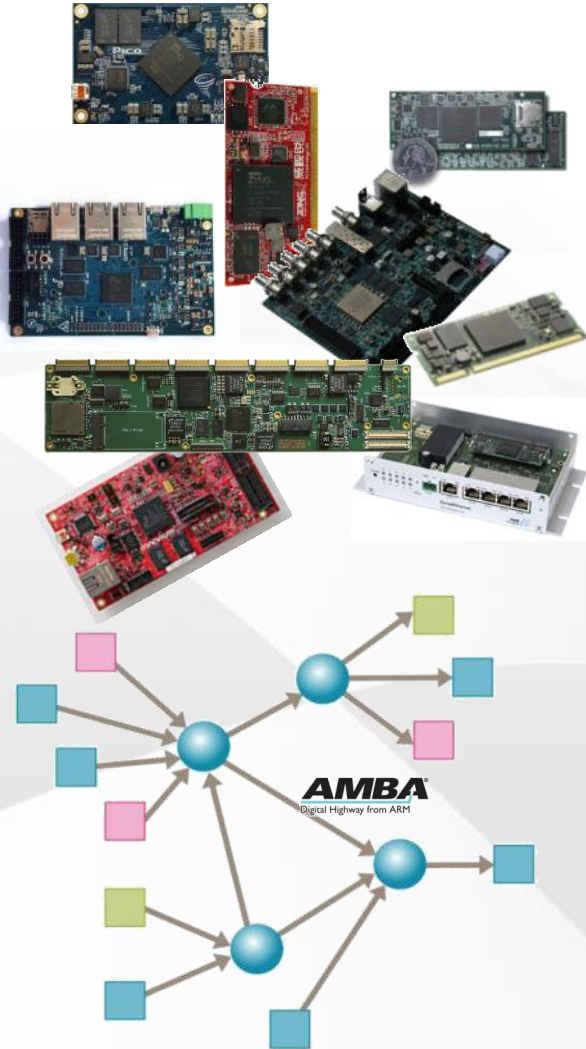
IDE
Tools



RTOS
& More



Largest Portfolio of IP, Design Kits and Ref. Designs



- **Widest Selection of Development Platforms and SOMs**

- Over 20 boards and SOM from Xilinx and partners
- Embedded, intelligent control, video signal processing
- 100+ FMC daughter cards

- **Ready to Use Segment Solutions**

- Boards, IPs, reference designs, app notes, OS and drivers
- Automotive, A&D, broadcast, consumer, industrial, medical, wired and wireless

- **Large Selection of AXI-based IPs**

- Xilinx IPs standardized on AXI
- Large availability of Xilinx and Partner IPs
- Enhanced portability between FPGAs and SoCs

Where to go for additional training?

Avnet Speedway Courses

- ◆ **Developing Zynq-7000 AP SoC Hardware**
 - Learn ARM Cortex-A9 architecture
 - Create a Zynq hardware design in Vivado
 - Debug hardware with Vivado Analyzer
- ◆ **PetaLinux for the Zynq-7000 AP SoC**
 - Hands-on experience in building the Linux environment
 - Introduces embedded Linux components, use of open-source components, environment configurations, network components, and debugging/profiling options
- ◆ **Designing Accelerators for the Zynq-7000 AP SoC**
 - Create custom hardware accelerators in the PL
 - Learn profiling techniques to identify acceleration opportunities
 - Driver and software application development



www.em.avnet.com/xilinxpeedways

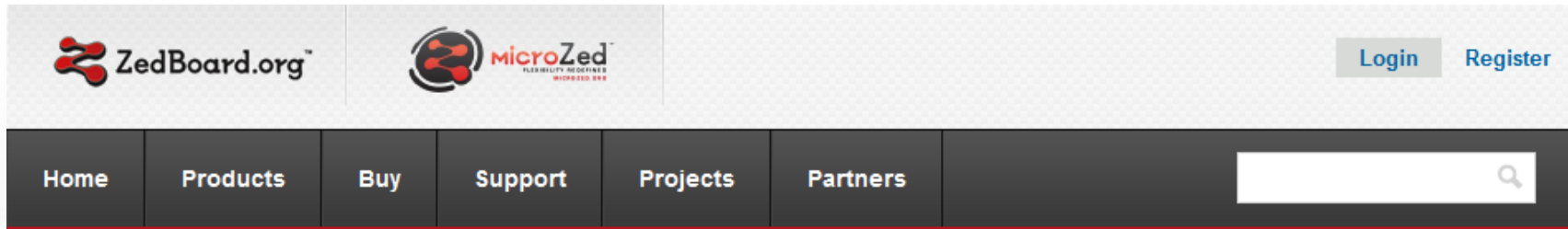


Even More Training...

Zynq-7000 All Programmable SoC Product Training		
Essentials of Microprocessors	Level: 1	Duration: 2 days
Learn what makes microprocessors tick! This class offers insights into all major aspects of microprocessors, from registers through coprocessors and everything in between. Differences between RISC and CISC architectures are explored as well as the concept of interrupts. A generic microprocessor is programmed and run in simulation to reinforce the principles learned in the lecture modules. The student will leave the class well prepared for the Xilinx Zynq-7000 All Programmable SoC training curriculum.		
C-Programming with Xilinx SDK	Level: 1	Duration: 2 days
This course is broken into a day of C language review, including variable naming, usage, and modifiers as well as an introduction to the Software Development Kit (SDK) environment, an explanation of the use of the preprocessors, program control, and proper use of functions. The second day consists of common issues and techniques employed by embedded programmers in the Xilinx SDK environment.		
How to Design Xilinx Embedded Systems in One Day	Level: 2	Duration: 1 day
The workshop introduces you to fundamental embedded design concepts and techniques for implementation in Xilinx FPGAs. The focus is on fundamental aspects of Xilinx embedded tools, IP, and the Embedded Targeted Reference Design (TRD). Design examples and labs are drawn from the Embedded TRD.		
Introduction to the Zynq-7000 All Programmable SoC Architecture	Level: 3	Duration: 1 day
This course provides hardware and firmware engineers with the knowledge to effectively utilize a Zynq-7000 All Programmable SoC. It covers the architecture of the ARM® Cortex™-A9 processor-based processing system (PS) and the integration of programmable logic (PL).		
Zynq-7000 All Programmable SoC Architecture	Level: 3	Duration: 2 days
The Xilinx Zynq-7000 All Programmable SoC provides a new level of system design capabilities. This course provides experienced system architects with the knowledge to effectively architect a Zynq-7000 All Programmable SoC. This course presents the features and benefits of the Zynq-7000 All Programmable SoC architecture for making decisions on architecting a Zynq-7000 All Programmable SoC project. It covers the architecture of the ARM® Cortex™-A9 processor-based processing system (PS) and the integration of programmable logic (PL) at a sufficiently deep level that a system designer can successfully and effectively utilize the Zynq-7000 All Programmable SoC.		
Embedded Systems Development	Level: 3	Duration: 2 days
The Xilinx Zynq-7000 All Programmable SoC provides a new level of system design capabilities. This course brings experienced FPGA designers up to speed on developing embedded systems using the Embedded Development Kit (EDK). The features and capabilities of the Zynq-7000 All Programmable SoC as well as concepts, tools, and techniques are included in the lectures and labs. The hands-on labs provide students with experience designing, expanding, and modifying an embedded system, including adding and simulating a custom AXI-based peripheral. Additionally, the features and capabilities of the Xilinx MicroBlaze™ soft processor are also included in the lectures and labs.		
Embedded Systems Software Development	Level: 3	Duration: 2 days
This two-day course introduces you to software design and development for the Xilinx Zynq-7000 All Programmable SoC using the Xilinx Software Development Kit (SDK). You will learn the concepts, tools, and techniques required for the software phase of the design cycle. Topics are comprehensive, covering the design and implementation of the board support package (BSP) for resource access and management of the Xilinx Standalone library. Major topics include device driver use and custom development and user application debugging and integration. Practical implementation tips and best practices are also provided throughout to enable you to make good design decisions and keep your design cycles to a minimum. You will have enough practical information to start developing software applications for the ARM® Cortex™-A9 and MicroBlaze™ processors.		
Advanced Features and Techniques of Embedded Systems Design	Level: 4	Duration: 2 days
Advanced Features and Techniques of Embedded Systems Design provides embedded systems developers the necessary skills to develop complex embedded systems and enables them to improve their designs by using the tools available in the Embedded Development Kit (EDK). This course also helps developers understand and utilize advanced components of embedded systems design for architecting a complex system in the Zynq-7000 All Programmable SoC or MicroBlaze™ soft processor. This course builds on the skills gained in the Embedded Systems Design course. Labs provide hands-on experience with developing, debugging, and simulating an embedded system. Utilizing memory resources and implementing high-performance DMA are also covered. Labs use demo boards in which designs are downloaded and verified.		
Advanced Embedded Systems Software	Level: 4	Duration: 1 day
This course will help software engineers fully utilize the components available in the Zynq-7000 All Programmable SoC. This course covers advanced Zynq-7000 All Programmable SoC topics for the software engineer, including advanced boot methodology, the NEON co-processor, programming PS system-level function control registers, the general interrupt controller, the DMA, Ethernet, and USB controllers, and the various low-speed peripherals included in the Zynq-7000 All Programmable SoC processing system.		
Embedded Design with PetaLinux SDK	Level: 4	Duration: 2 days
This intermediate-level, two-day course provides embedded systems developers with experience in creating an embedded PetaLinux SDK operating system on a Xilinx MicroBlaze processor development board. The course offers students hands-on experience on building the environment and booting the system using a basic, single-processor System on Chip (SoC) design with PetaLinux SDK on the MicroBlaze processor. This course also introduces embedded Linux components, use of open-source components, environment configurations, network components, and debugging/profiling options for embedded Linux platforms. The primary focus is on embedded Linux development in conjunction with the Xilinx tool flow.		

Getting Help and Support

- Visit ZedBoard.org or MicroZed.org
<http://www.zedboard.org> | <http://www.microzed.org>



- www.support.xilinx.com
 - Software, IP, and Documentation Updates
 - Access to Technical Support Web Tools
 - Searchable Answer Database with Over 4,000 Solutions
 - User Forums
 - Training - Select instructor-led classes and recorded e-learning options
- Contact Avnet Engineering Services if you are interested in customizing a kit or SOM
customize@avnet.com
- Contact your local Avnet/Silica FAE



Objectives

- Introduce developers to Xilinx SDK ✓
- Explore how SDK makes your job easier ✓
- Connect SDK to hardware for execution and debug ✓
- Show a basic example of using both processors ✓
- Utilize a peripheral interrupt to show real-time software response ✓

Thank You!

- Please Complete the Digital Survey

www.em.avnet.com/speedwaysurvey



Xilinx® SpeedWay Design Workshops™ - Course Surveys

Print Friendly



Xilinx® SpeedWay Design Workshops™
Featuring MicroZed™, ZedBoard™ and Vivado® Design Suite



Thank you for participating in our course surveys, your feedback is very valuable and we use it to improve future trainings.

Select a Course Survey to Begin:

- Course Survey - Designing Accelerators for the Zynq-7000 AP SoC
- Course Survey - Developing Zynq-7000 AP SoC Hardware
- Course Survey - Developing Zynq-7000 AP SoC Software
- Course Survey - Introduction to Vivado
- Course Survey - PetaLinux for the Zynq-7000 AP SoC