Implementing Linux on the Zynq™-7000 SoC

Open Source Linux Development on ZedBoard™

Accelerating Your Success™

Why is this course important?

Course Objectives

By the end of the day, you will

- **Build critical software components of an embedded Linux system**
- **Explore Linux software development with Xilinx SDK**
- **Connect SDK to hardware for execution and debug**
- **Understand the advantages of running Linux on Zynq**

Morning Agenda

- **Overview of Open Source Linux on Zynq-7000 AP SoC**

- **Part 1 - Booting ZedBoard**
  - First Stage Boot Loader (FSBL)
  - U-Boot
  - Zynq Boot Image

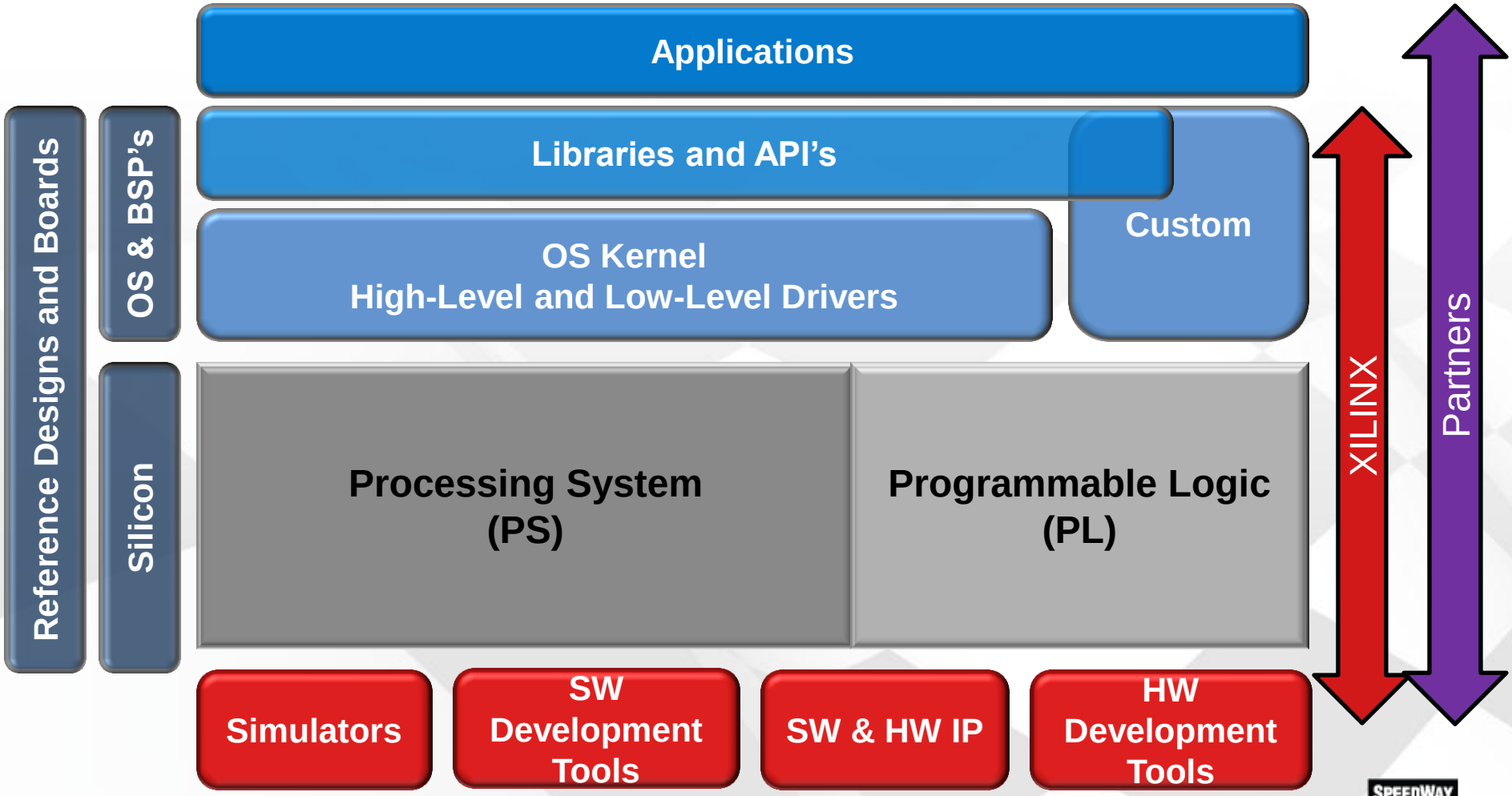
- **Part 2 - Linux Kernel Basics**
  - The Linux Kernel
  - Linux Root File System
  - LinuxLink from TimeSys for Maintenance Builds
  - Booting a Minimal Linux Platform

Afternoon Agenda

- **Part 3 - Linux Device Drivers**
  - Linux Device Driver Purpose and Relationship to the Kernel
  - Device Drivers and Programmable Logic IP Connected to AXI
  - Linux Device Tree
- **Part 4 - Linux Application Development and Debug**
  - Linux Application Development Using SDK
  - Application Debug Using SDK

Overview of Open Source Linux on Zynq-7000 AP SoC

Zynq-7000 AP SoC Programming Model

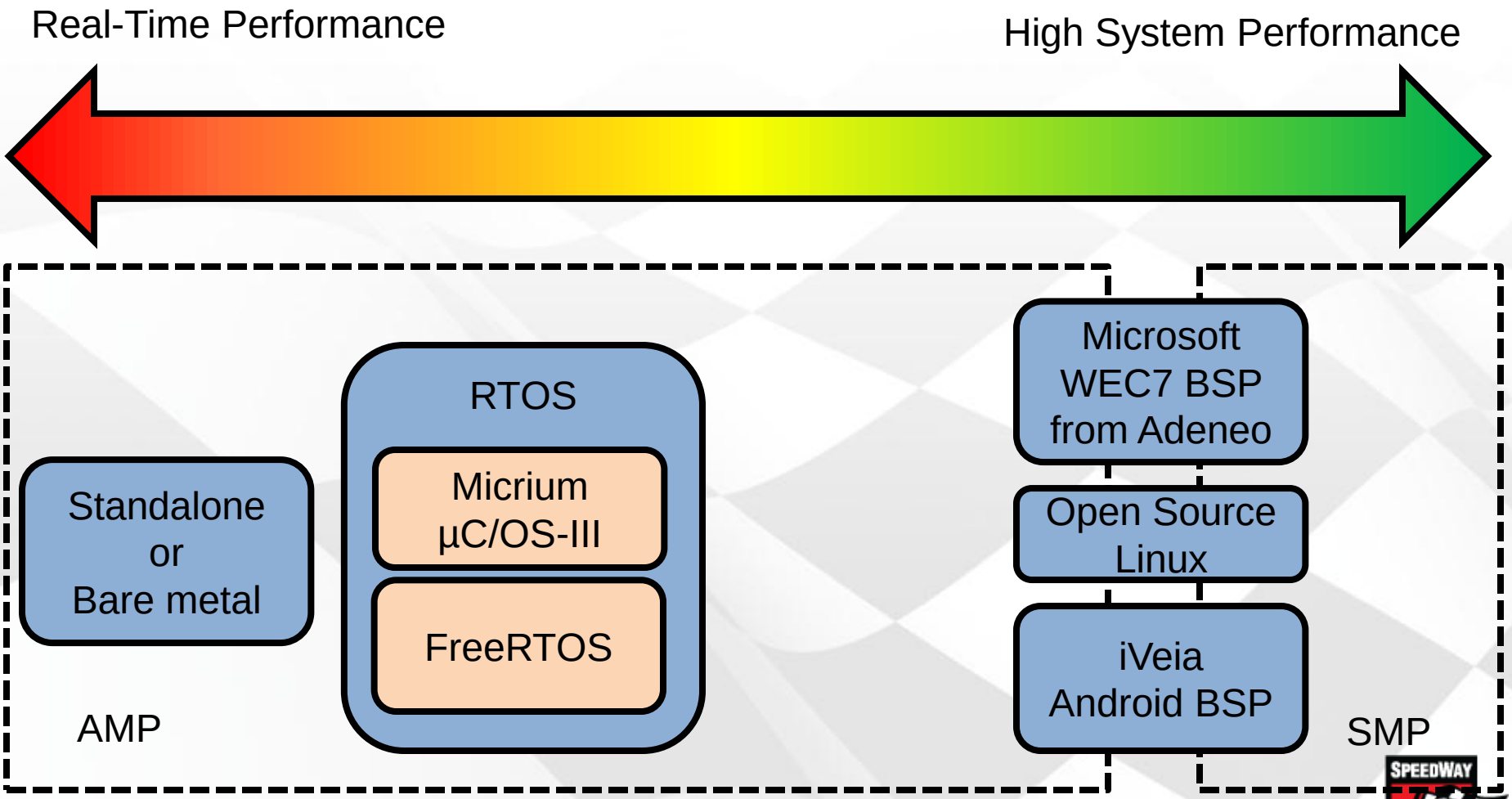


Zynq-7000 OS Support

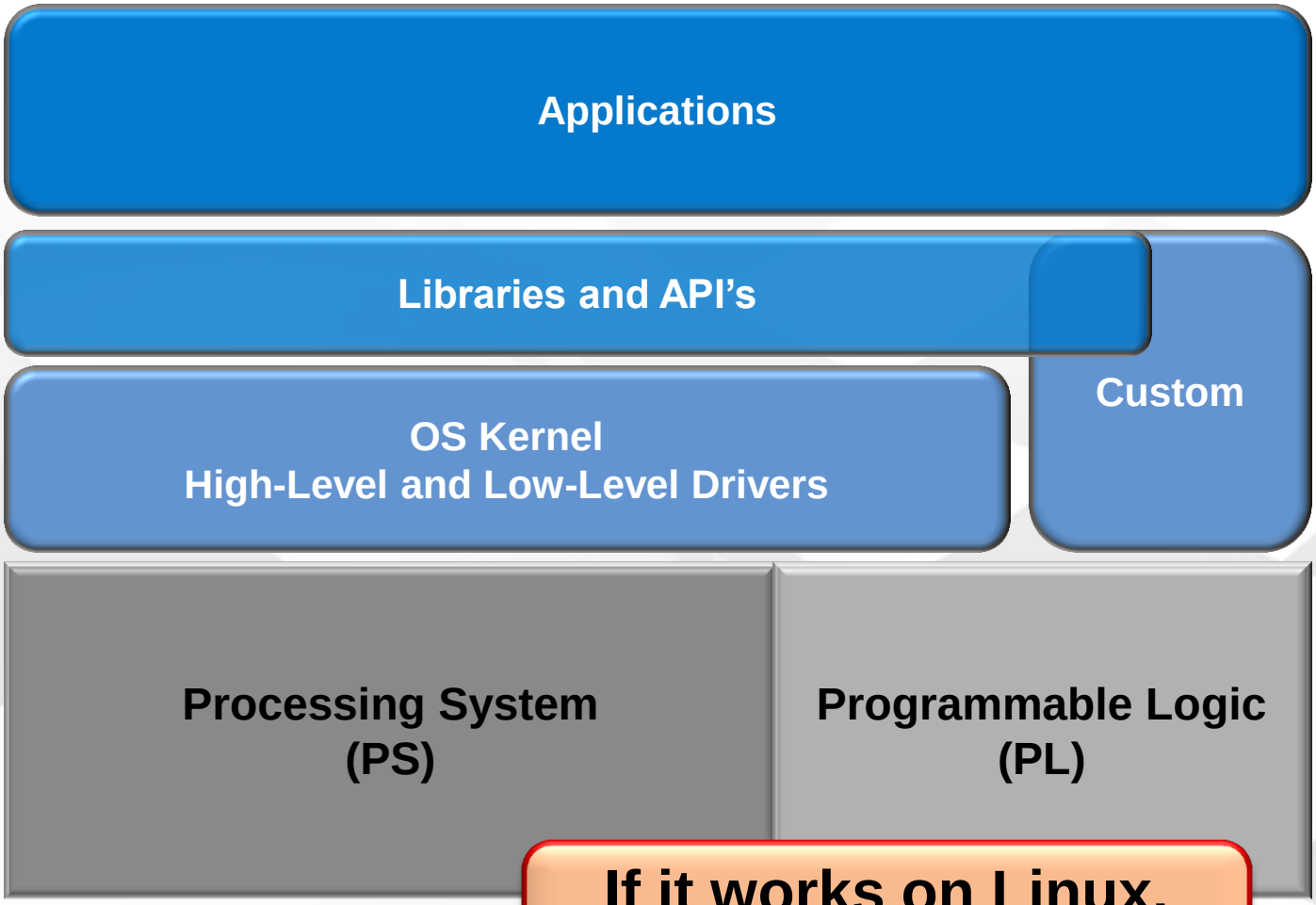
- **Zynq-7000 supports a comprehensive collection of operating systems to suit your system needs**
  - Open Source Linux
    - Xilinx provides a freely downloadable Linux solution
    - Xilinx Git Repository
    - Xilinx acquired PetaLogix
  - Open Source Android
    - Freely downloadable Android 2.3 solution
  - Open Source FreeRTOS
    - Light-weight real-time OS
    - Used in applications that demand deterministic and real-time responsiveness to events in the system

Choosing the Optimal Operating System

- **What are your software application requirements?**



Zynq-7000 AP SoC Linux Software Platform



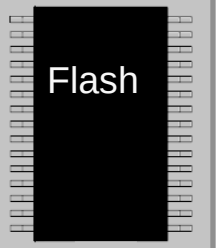
If it works on Linux, it works on Zynq!

Booting ZedBoard

Roadmap - Linux in Four Parts

❶ Boot Loading – 3 stages

- Basic hardware initialization
- Loads Linux kernel
- Passing boot parameters

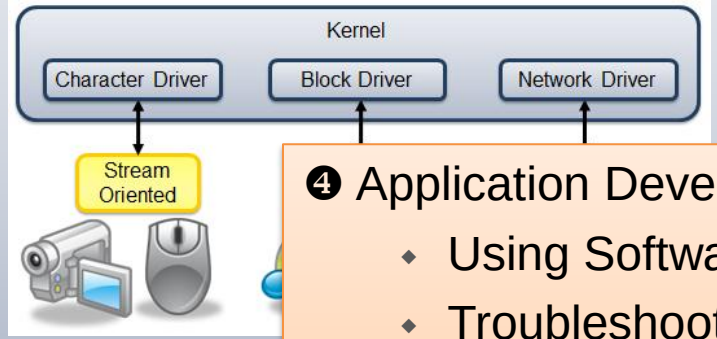


❷ Kernel

- Manages system resources
- Provides application services

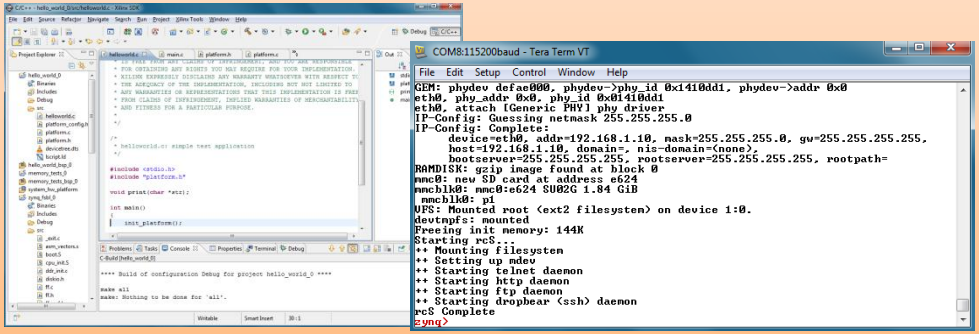
❸ Device Drivers

- Abstracts hardware details from software



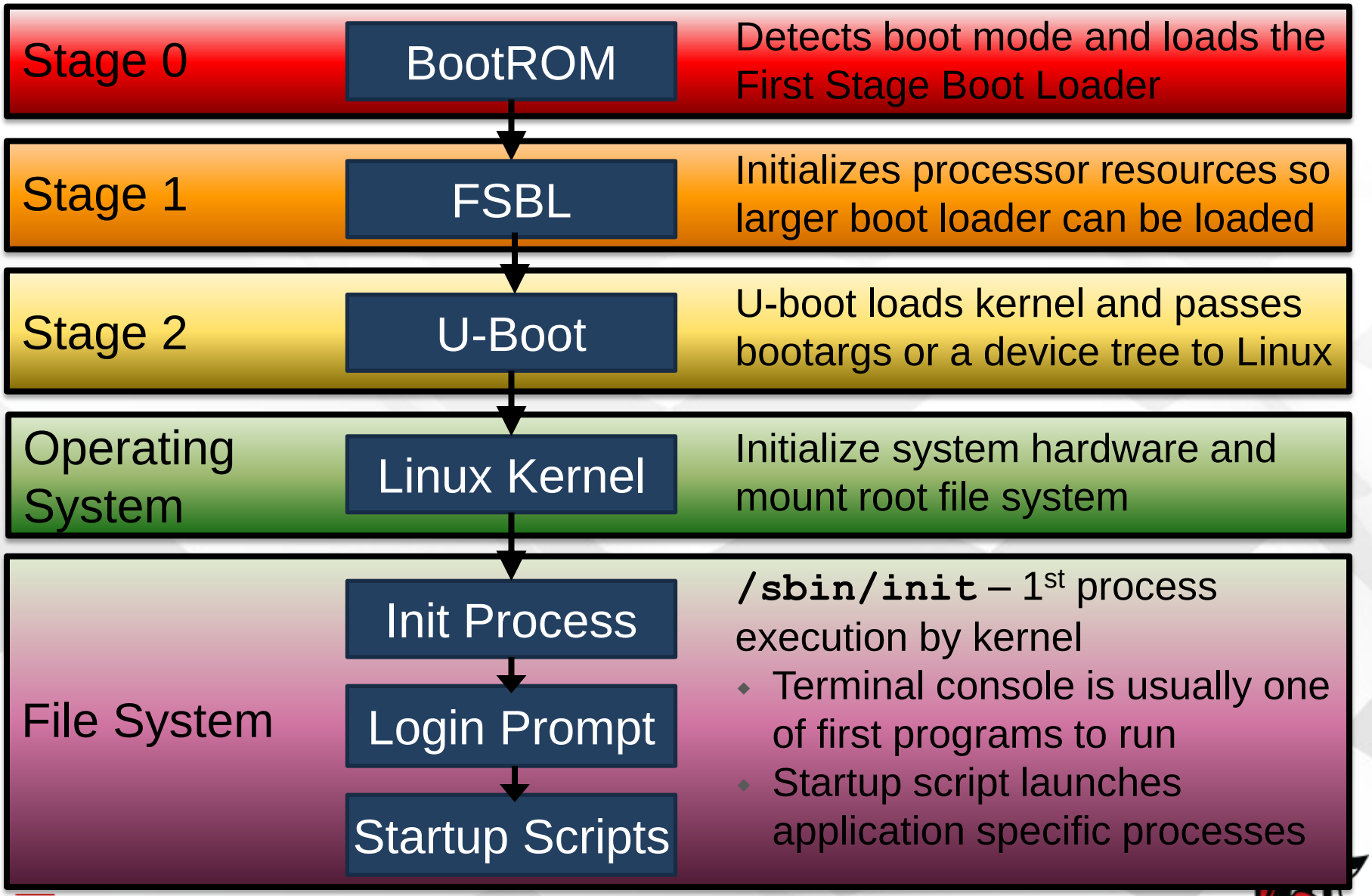
❹ Application Development and Debug

- Using Software Development Kit
- Troubleshoot user applications



AVNET electronics marketing

Typical Linux Boot Sequence Overview



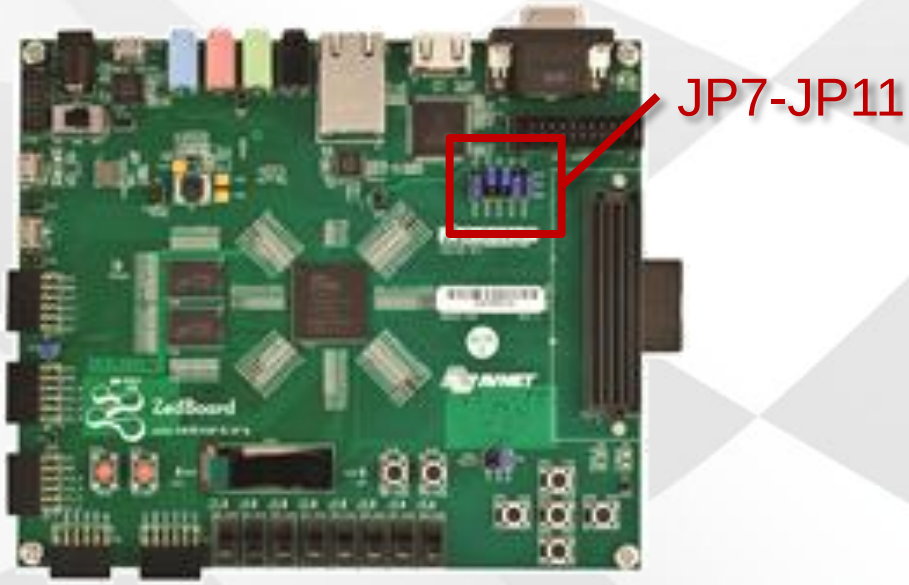
Zynq Boot Loader Components

Boot Stage	Operations	User Configurable	Zynq-7000 Terminology
Stage 0	ROM code detects desired boot mode (NAND, QSPI, JTAG) and loads executable code of First Stage Boot Loader (FSBL) from selected peripheral/interface	No	BootROM
Stage 1	Initializes external memory and system clocks so that a larger, more advanced boot loader (in our case U-boot) can be loaded	Yes	FSBL
Stage 2	Used to locate, load, and execute the Linux kernel and is also responsible for passing a device tree to the kernel	Yes	U-Boot

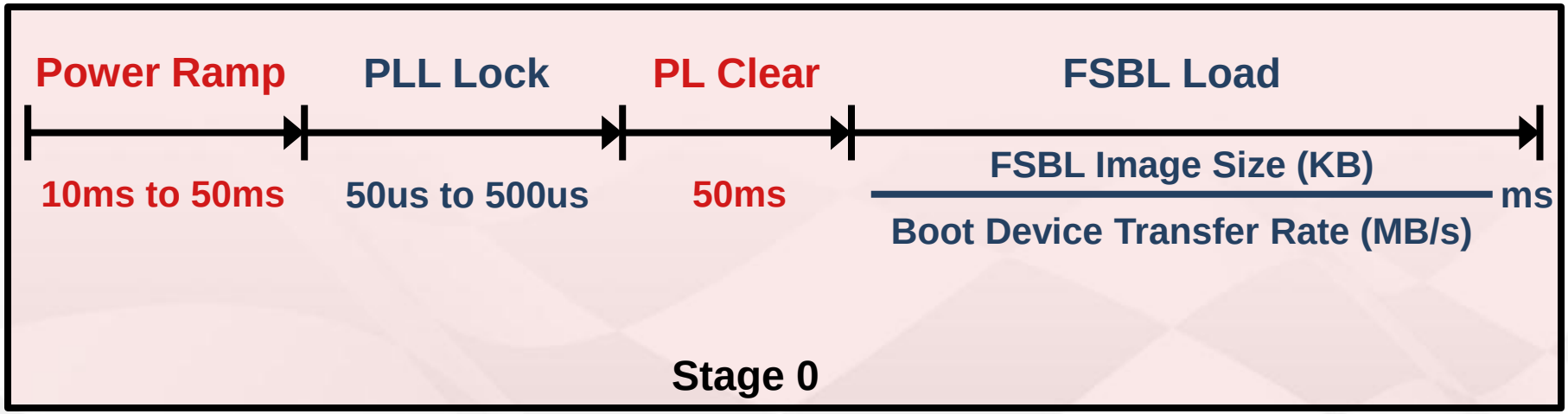
Zynq Boot Modes

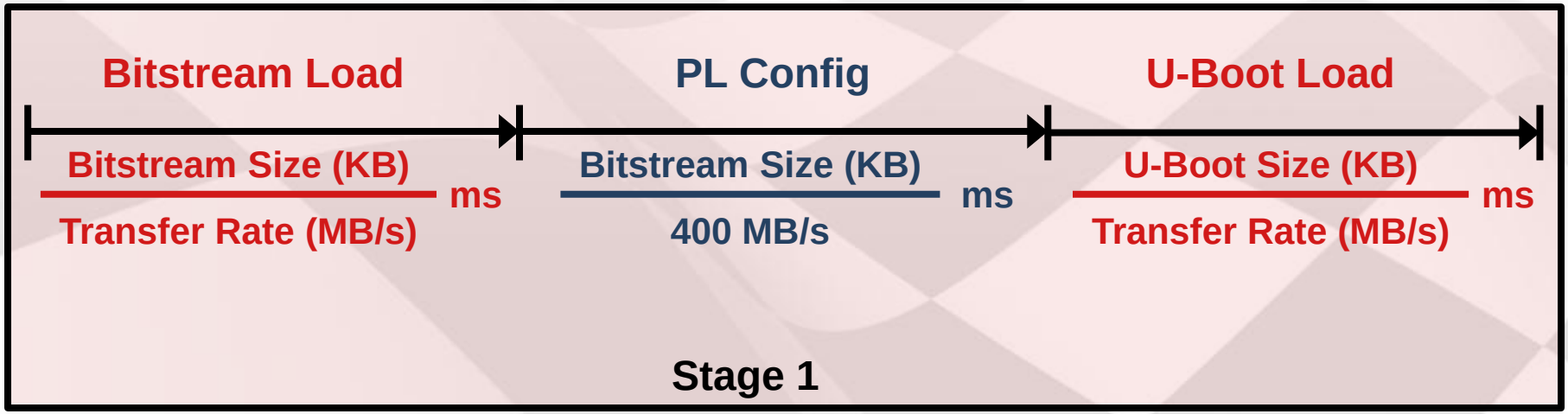
- **Boot mode latched at POR using MIO pins by BootROM**

Mode	MIO[5] (JP10)	MIO[4] (JP9)	MIO[3] (JP8)
JTAG	0	0	0
NOR	0	0	1
NAND	0	1	0
Quad-SPI	1	0	0
SD Card	1	1	0



Factors Determining Boot Time



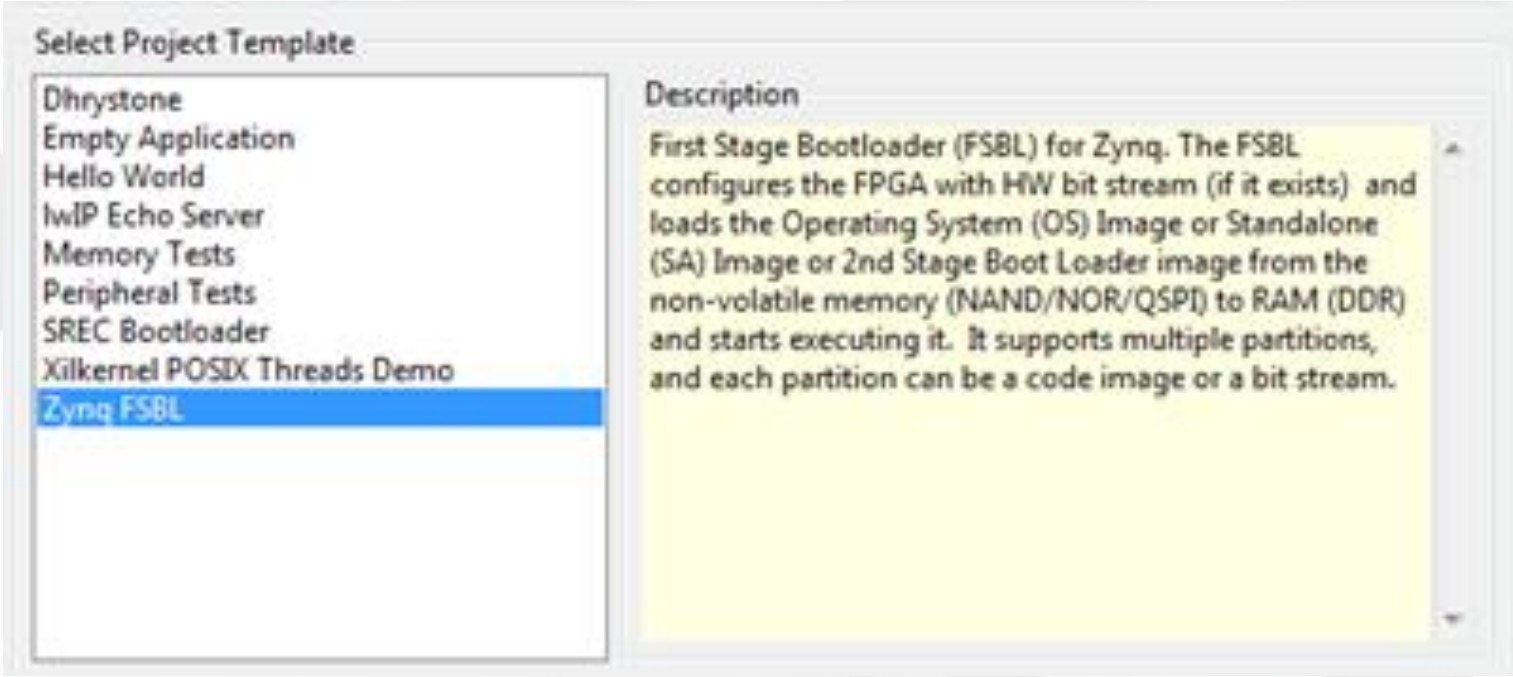


First Stage Boot Loader

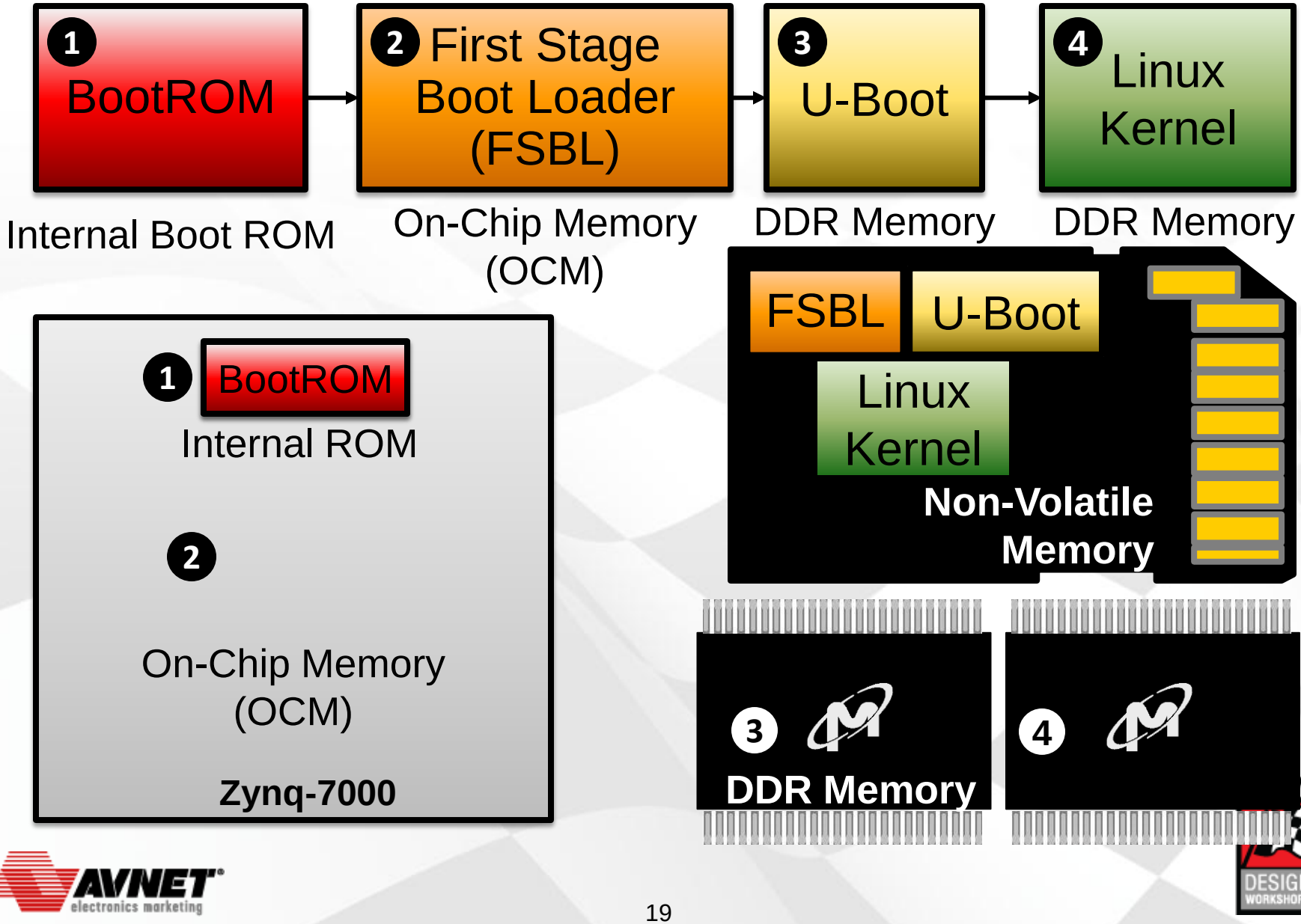
- **Initialize Zynq PS**
  - PLLs
  - DDR memory controller
  - MIO
  - UART
- **Optionally configure PL with bitstream**
- **Load application code from boot medium to memory**
- **Transfer execution to application code (U-Boot)**

Create FSBL in SDK

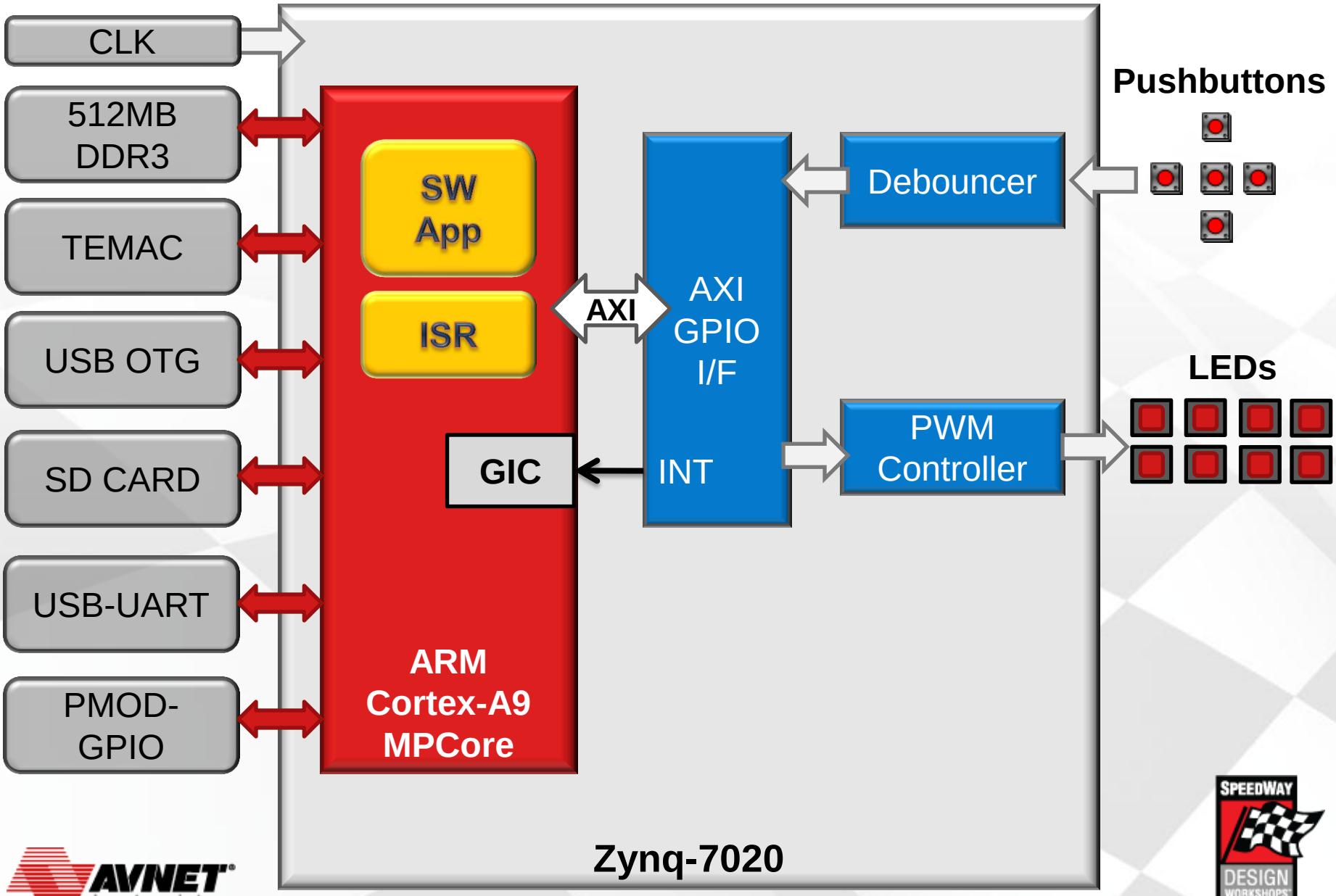
- **Generate directly from the SDK project template**
- **Initializes PS with XPS platform configuration**
- **Source code user editable**
- **Other Linux platforms refer to this code as** *user boot code*



Linux Boot Process Overview for Zynq



Zynq Intro SpeedWay Platform with PL Peripherals



Lab 1.1

- **Creating the Zynq First Stage Boot Loader (FSBL)**
  - Begin with the hardware platform built during the Introduction to Zynq SpeedWay
  - Create a FSBL based upon PlanAhead hardware platform

- **Note: All SpeedWay lab materials, including lab instructions, can be found in the following folder on the workstation:**

  C:\Speedway\Fall_12\Zynq_Linux

U-Boot Introduction

- **U-Boot is an open source software project**
- **U-Boot performs important boot functions**
  - Initialization of the platform hardware needed to load kernel
  - Loads Linux kernel from boot medium to main memory (DDR)
  - Starts the Linux kernel with specified boot parameters
- **ZedBoard out of box demo runs the open-source boot loader U-Boot**

U-Boot Features

- **U-boot also provides some convenient features that help during development**
  - Provide network access
    - Ping IP addresses
    - Download binary images via TFTP
  - Memory test, copy, and comparison
    - Reads and writes arbitrary memory locations
    - Copies binary images from one location in memory to another
  - Configure and access hardware peripheral devices directly (e.g. Quad-SPI Flash, I2C, Ethernet)
  - Detect boot mode and run related boot macros

U-Boot Commands

- **U-boot provides console interface to execute commands**

Command	Function	Example
setenv	Sets an environment variable	`setenv ipaddr 192.168.1.10`
printenv	Lists environment variables and their contents	`printenv` `printenv ipaddr`
run	Runs a macro	`run sdboot`
dhcp	Attempts to set the IP address using DHCP	`dhcp`
tftp	Downloads a file into memory using TFTP	`tftp 0x8000 zImage`
ping	Send ICMP ECHO_REQUEST to remote network host	`ping 192.168.1.50`
boot	Runs the macro in the bootcmd variable	`boot`
fatload	Loads a file from a mounted FAT filesystem	`fatload mmc 0 0x8000 zImage`

U-Boot

- **U-boot is configured using named text strings called environment variables**
- **The ‘setenv’ command is used to create, modify, and delete environment variables**

	Example command	Explanation
Create	`setenv myvar foo`	Creates a new variable named ‘myvar’ with value ‘foo’
Modify	`setenv ipaddr 192.168.0.10`	Changes the value of the ‘ipaddr’ environment variable to 192.168.0.10
Delete	`setenv myvar`	Deletes the variable ‘myvar’ when you run setenv with no contents to set into the variable

- **‘saveenv’ command stores environment to non-volatile memory – NOT Supported on ZedBoard**

U-Boot

- **U-boot macros are environment variables that contain commands**
  - Evaluated and executed using the ‘run’ command

Here is an example of the variable 'fetch_kernel' created as a macro, containing a command 'tftp'

```
> setenv serverip 192.168.1.10
> setenv bootimage zImage
> setenv fetch_kernel 'tftp ${serverip}:${bootimage}'
```

The variable 'fetch_kernel' exists now as a macro, containing command 'tftp' and a variable sequence

```
> run fetch_kernel
```

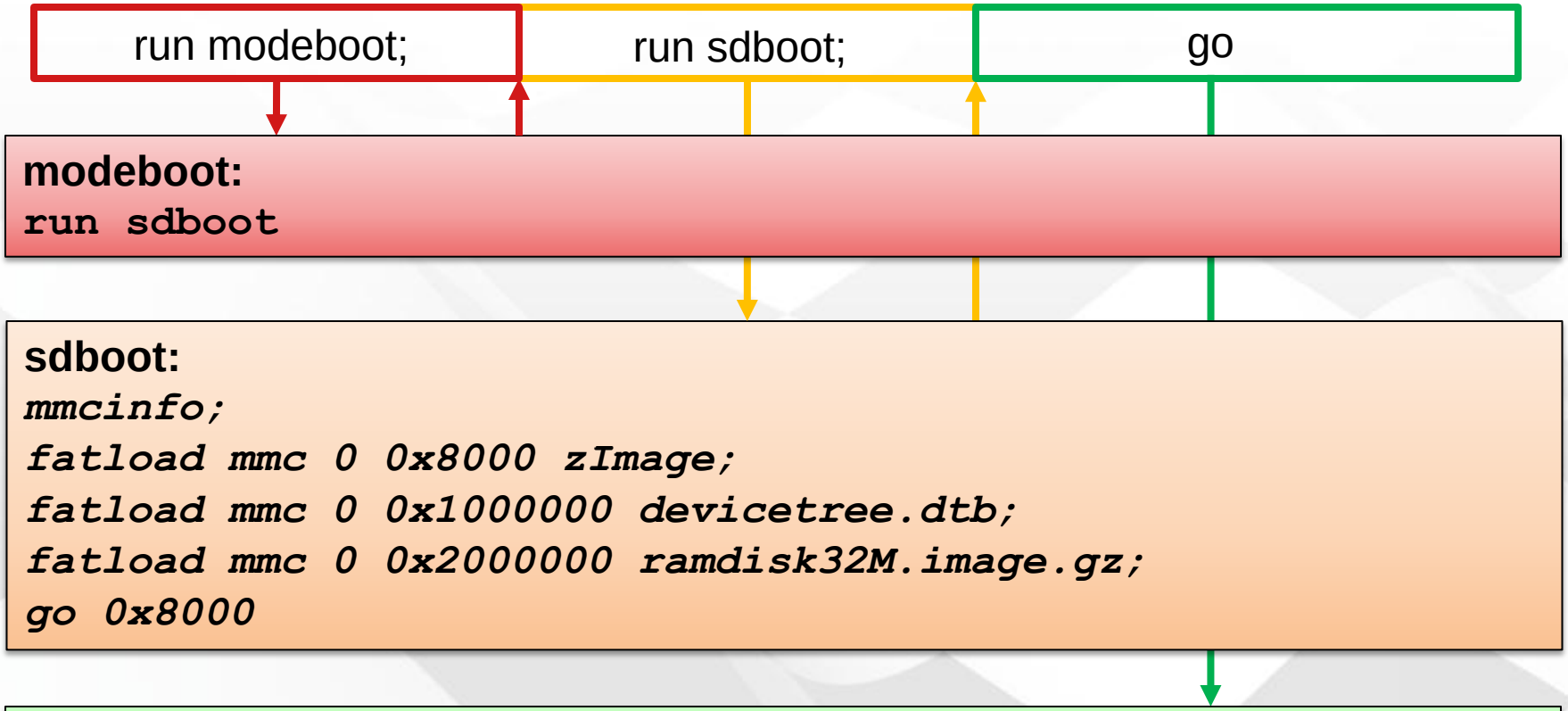
U-Boot Environment Variables

- **Certain environment variables are ‘reserved’**
  - Important standard variables used for specific U-Boot functionality

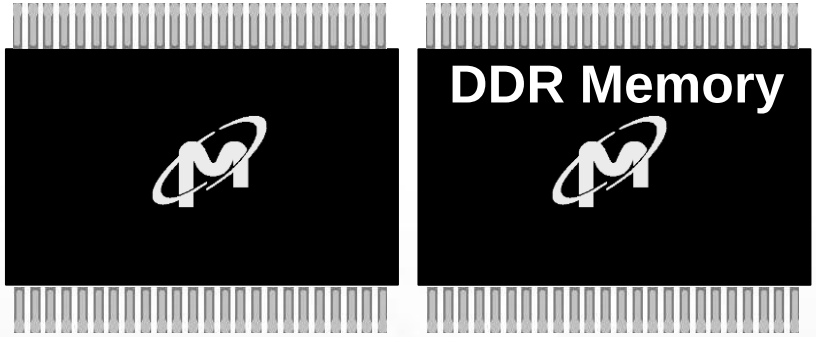
Variable	Function
`bootcmd`	Stores the macro that is run when the ‘boot’ command is executed
`bootdelay`	Delay in seconds until U-boot runs the ‘boot’ command (Set to -1 to disable autoboot)
`ethaddr`	Ethernet MAC address (read-only in our system)
`ipaddr`	The IP address of the board when running U-boot
`serverip`	The server IP address (IP address of the TFTP server used for `tftpboot`)
`modeboot`	Determines which method will be used to boot Linux (based upon **BOOT_MODE_REG**)

U-Boot Macros

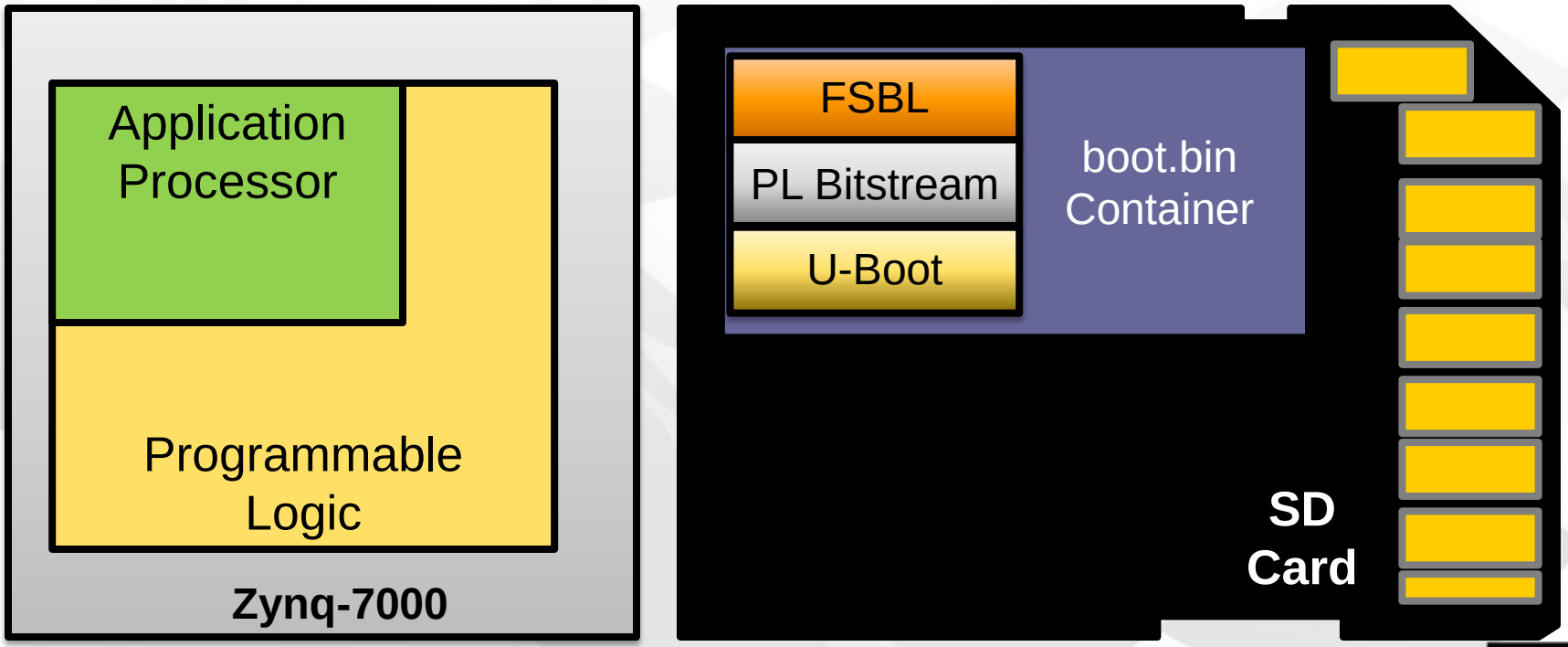
- **Example use of macros: run modeboot**



Booting U-Boot



- **Copy this file to SD Card**
  - boot.bin



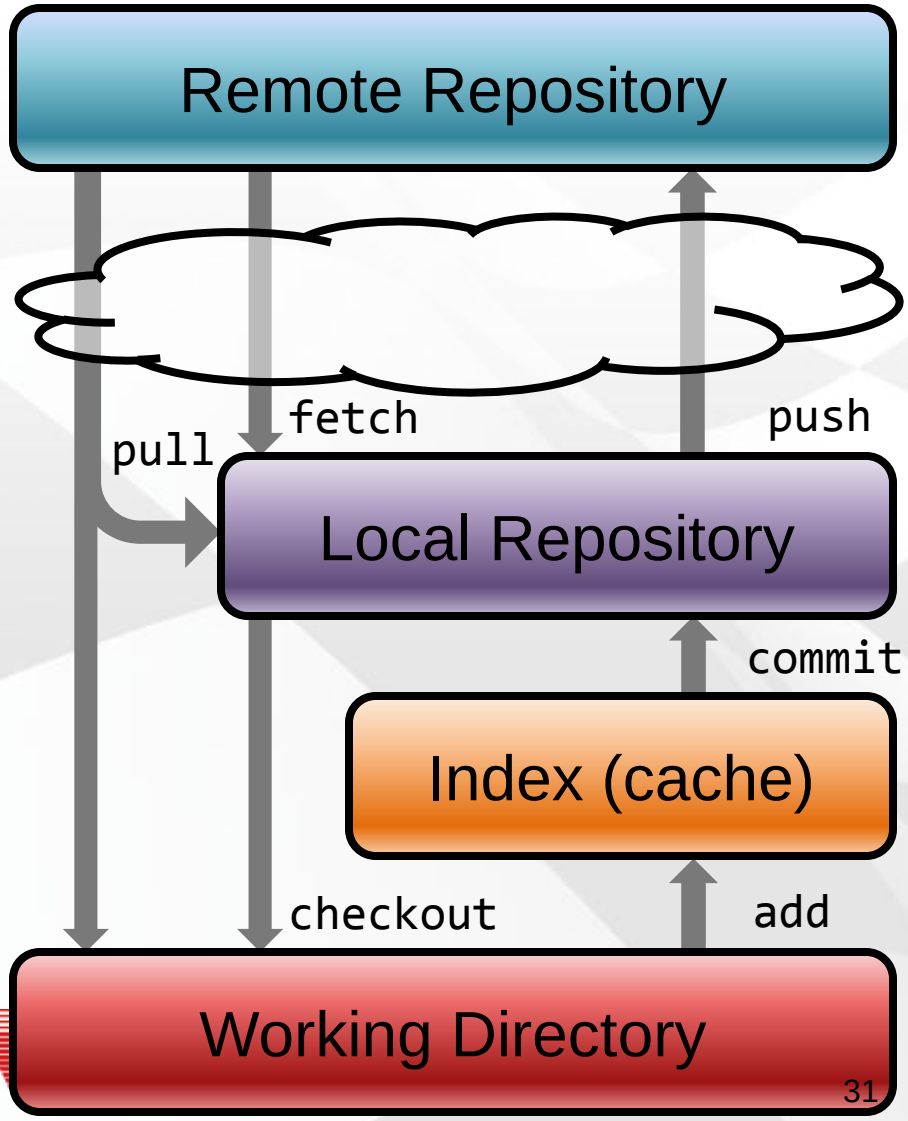
U-Boot Source

Where does the U-Boot source code come from?

- **U-boot source for Zynq hosted on the Xilinx git repository**
  - Based on version 2010.09 source code from git://git.denx.de
  - Xilinx git repository can be cloned using the **git clone** command

```
git clone git://git.xilinx.com/u-boot-xarm.git
```

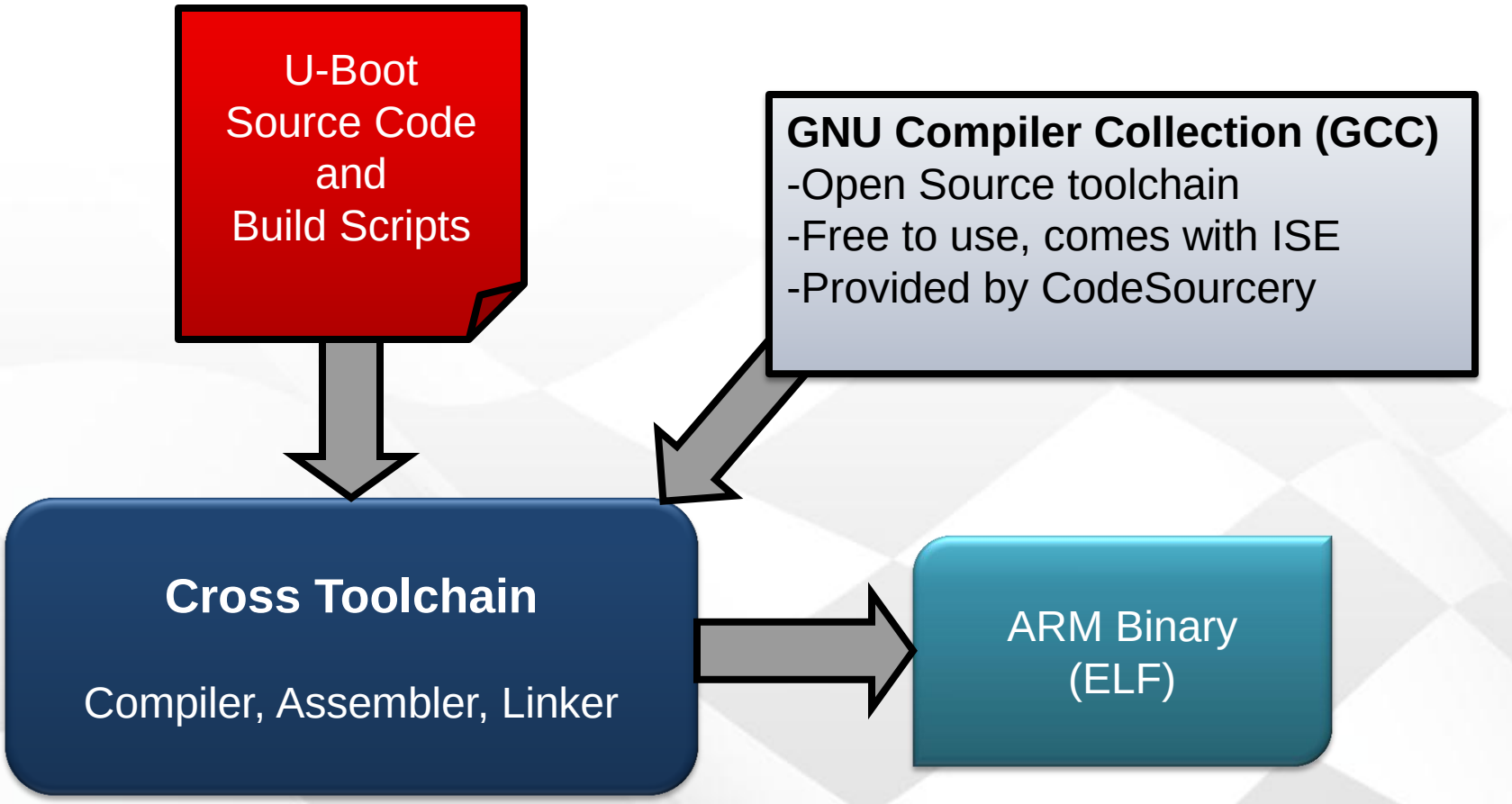
- **What is Git and why is it used?**



- **Source code management system**
- **Distributed revision control system**
- **Free software distributed under the terms of the GPL**
- **Intended to replace CVS and SVN**
- **For more info visit**

http://git-scm.com/

U-Boot Build Process



CodeSourcery cross-compiler for Zynq

- Runs on Linux host PC (x86 executable)
- Compiles executable code for Zynq (ARM executable)

Cross Build Platform

- **Virtual machines can run desktop Linux**
- **SpeedWay labs use VMware Player to run CentOS Workstation**

- **Latest VMware Player can be downloaded from the VMware website:**

  http://www.vmware.com/products/player/overview.html

- **CentOS can be downloaded from mirrors listed on CentOS website:**

  https://www.centos.org/

Lab 1.2

- **Building U-Boot from source code**
  - Configure and build U-Boot source tree for ZedBoard

- **Note: Experiment 1 is for reference only during the live SpeedWay events and should be performed only by online attendees**

Boot Image Format File

- **Zynq boot image needs an FSBL, bitstream, and second stage boot loader or application code**
- **Command line tool bootgen is used for constructing boot images for Zynq configuration**
  - Example command line usage of bootgen

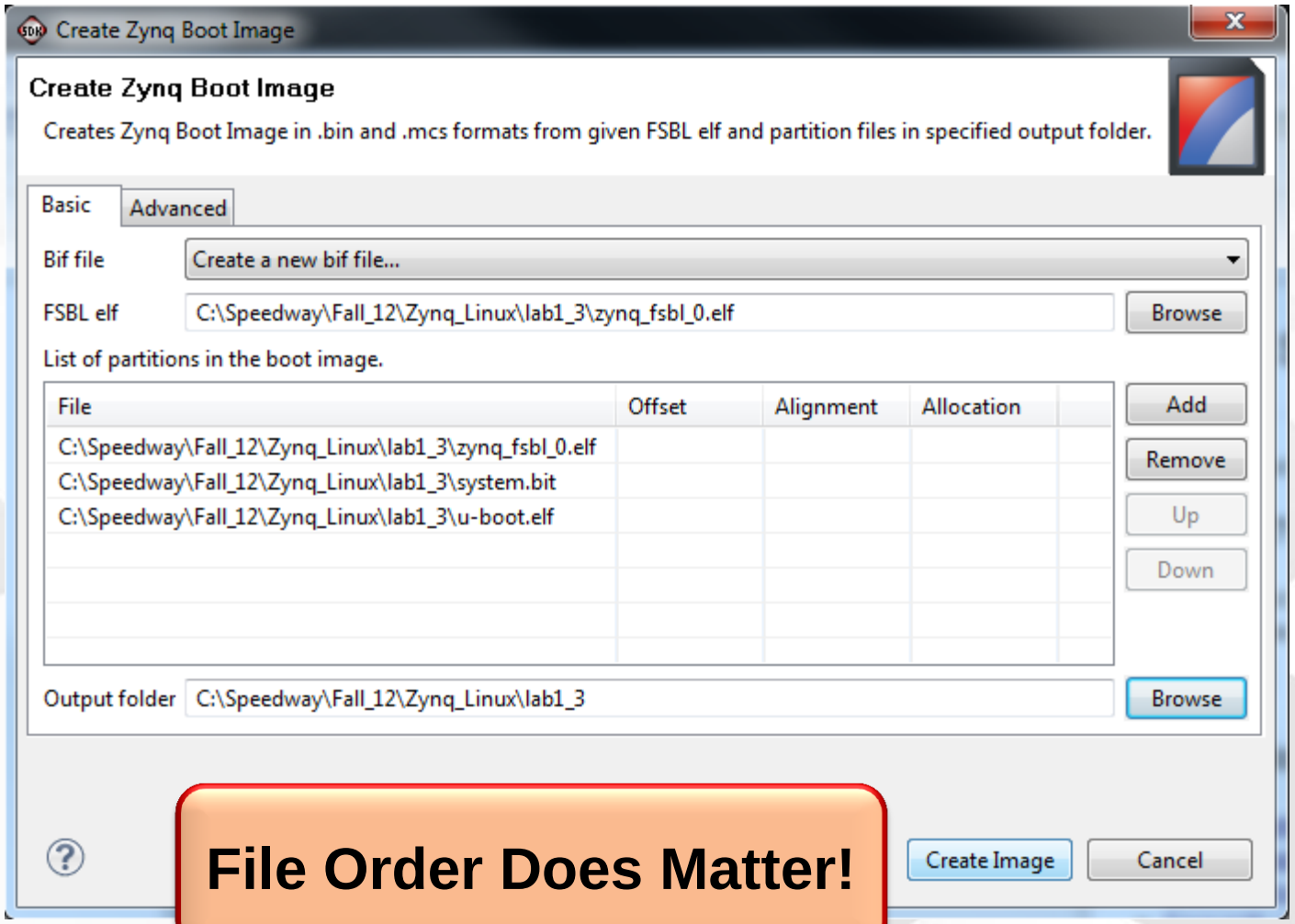
```
bootgen –image myDesign.bif –o i boot.bin
```

- **Merges .BIT and .ELF files into a single boot image, using the format of a Boot Image Format (.BIF) file**
  - Example BIF file entry

```
the_ROM_image:
        {
        [bootloader]zynq_fsbl_0.elf
        system.bit
        u-boot.elf
        }
```

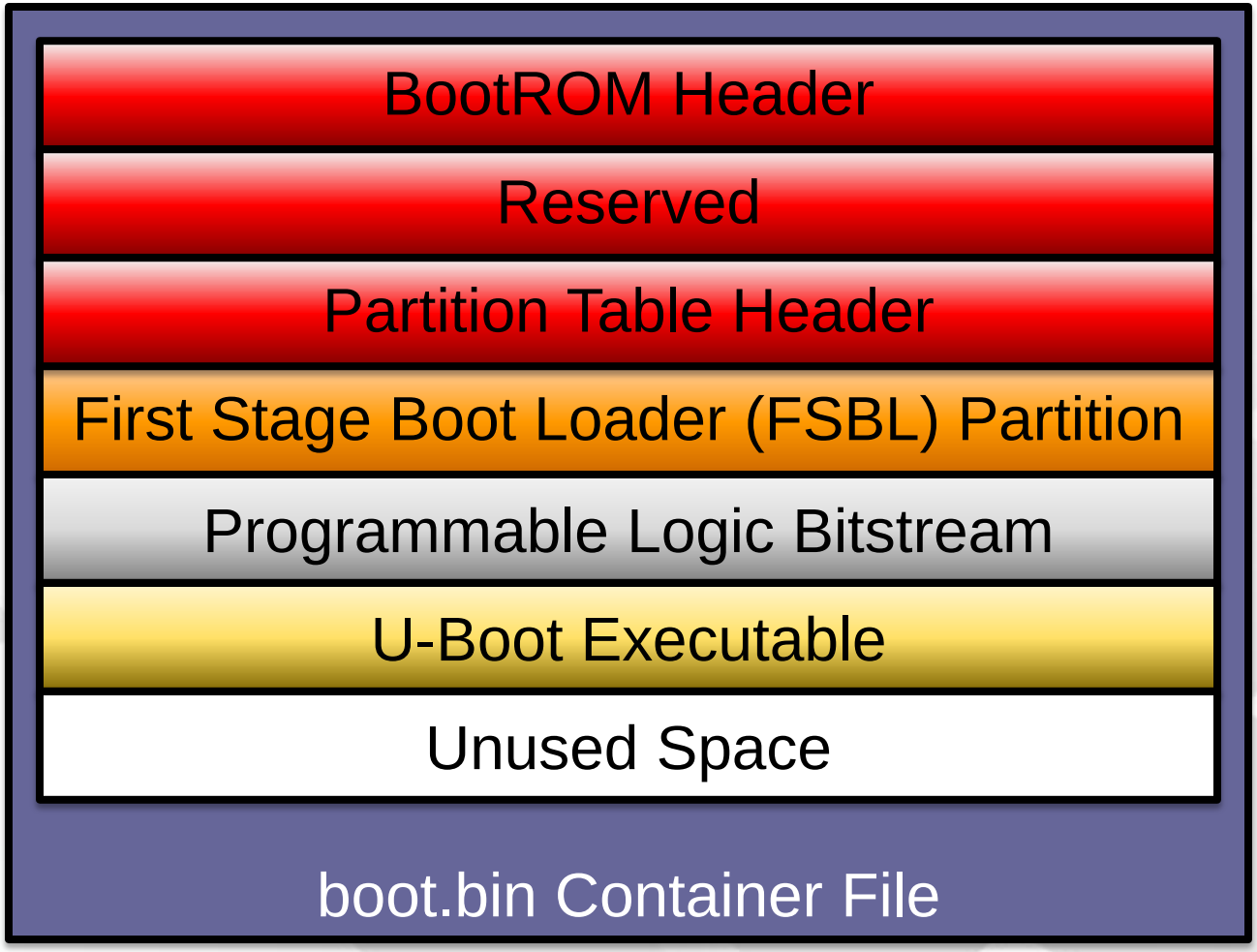
Create Zynq Boot Image in SDK

- **Graphical front end to command line bootgen**

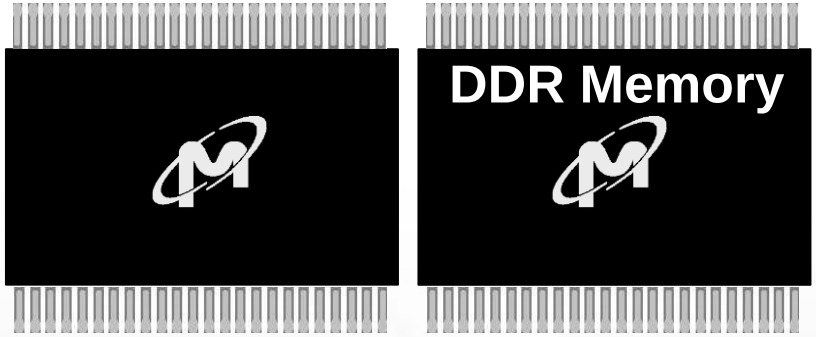


File Order Does Matter!

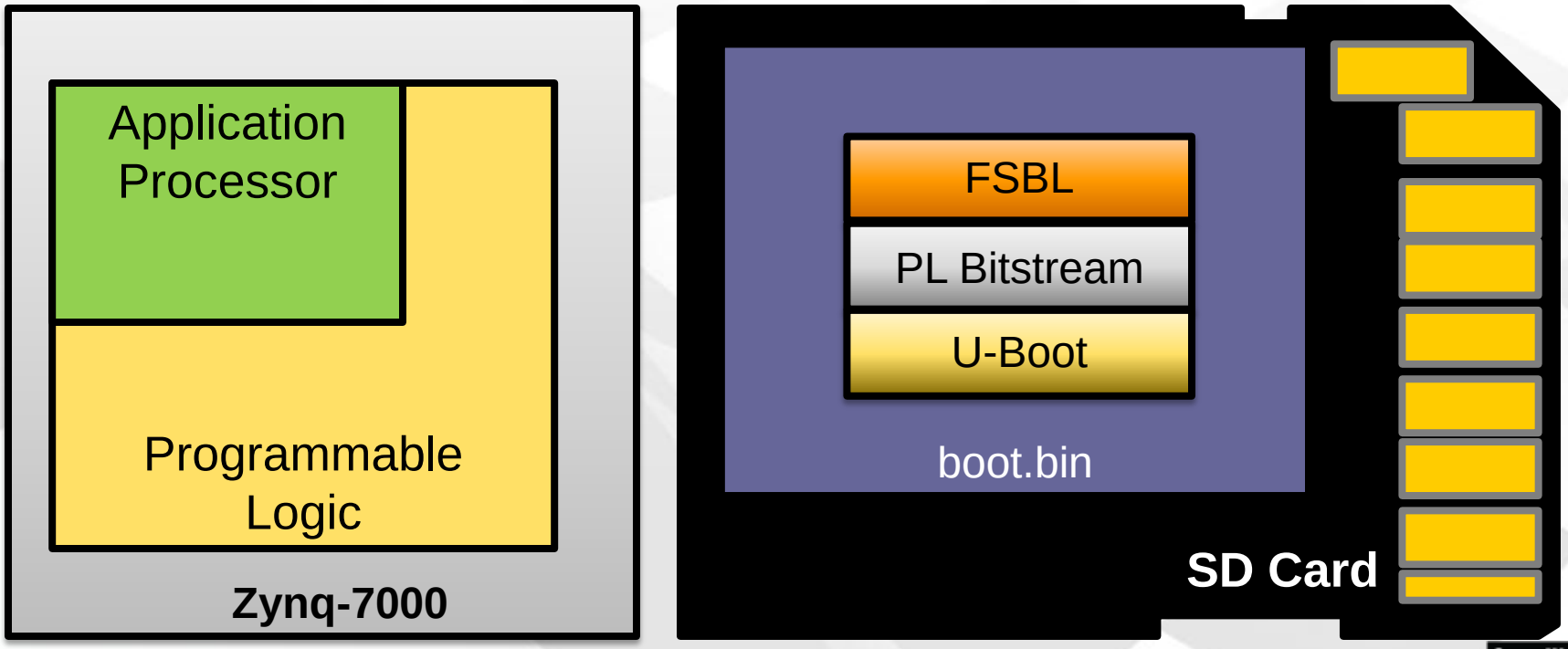
Example Of The Boot Image Format



Boot Image Mechanics



- **Copy boot.bin to SD Card**
- **Set Boot jumpers to SD mode**
- **Power on the ZedBoard**



Lab 1.3

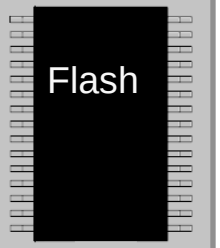
- **Create a boot image with SDK**
- **Boot to U-Boot on ZedBoard**

Linux Kernel Basics

Roadmap - Linux in Four Parts

❶ Boot Loading – 3 stages

- Basic hardware initialization
- Loads Linux kernel
- Passing boot parameters

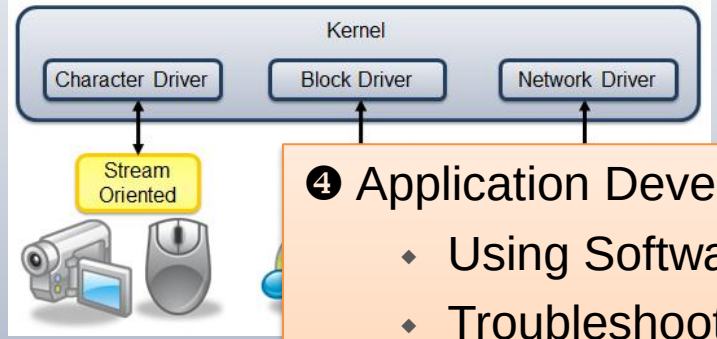


❷ Kernel

- Manages system resources
- Provides application services

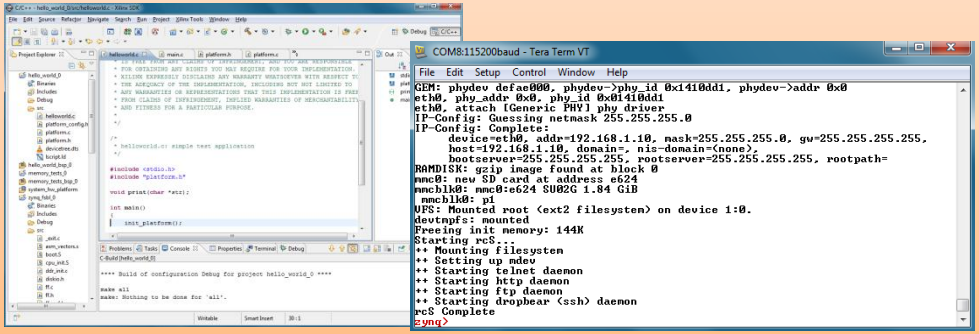
❸ Device Drivers

- Abstracts hardware details from software



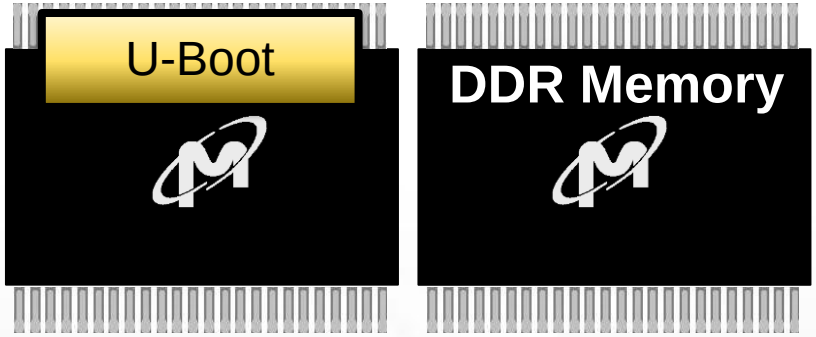
❹ Application Development and Debug

- Using Software Development Kit
- Troubleshoot user applications

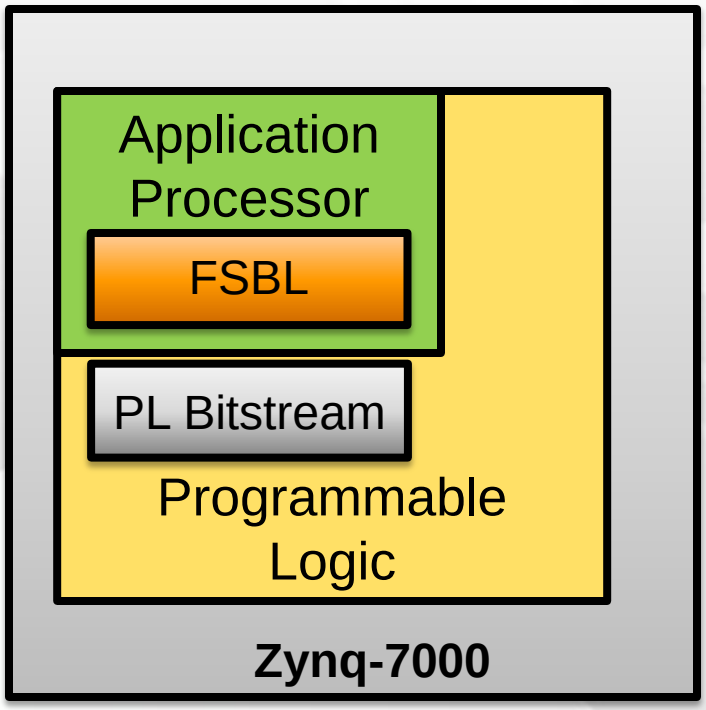


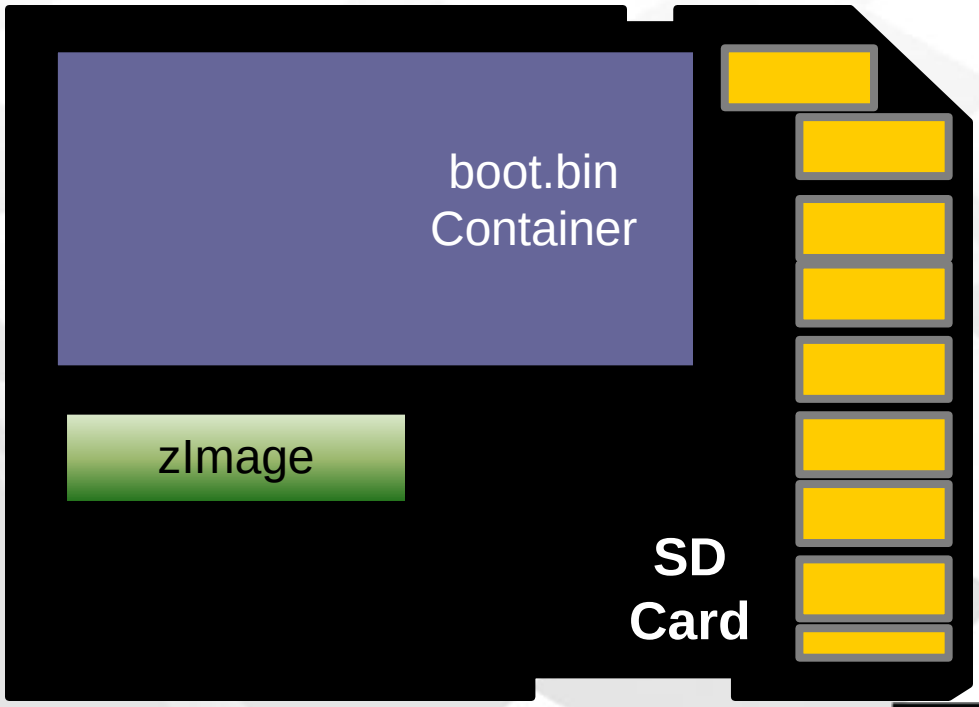
AVNET electronics marketing

Booting Linux



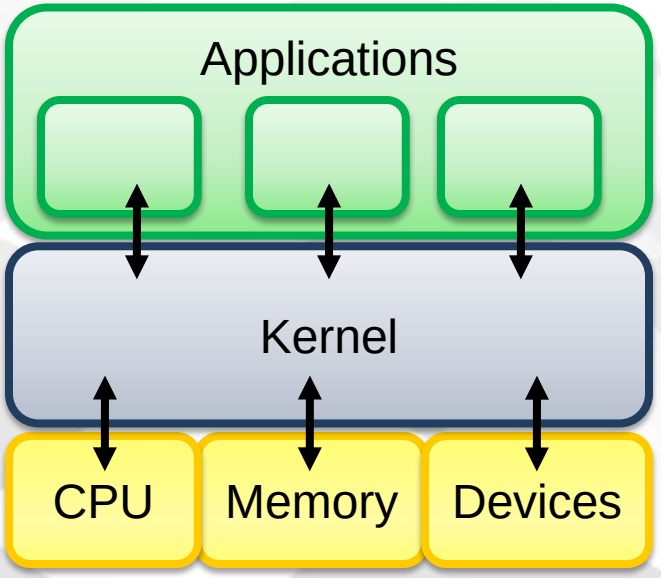
- **Copy this file to SD Card**
  - zImage





Linux Kernel Basics

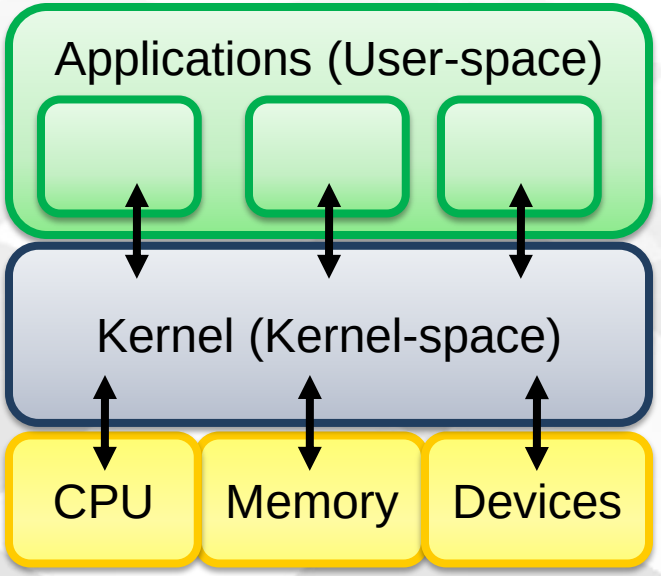
- **Advantages of Linux**
  - Broad use as an open source desktop, server, and embedded OS
  - Feature-rich
    - Symmetric multiprocessing
    - Preemptive multitasking
    - Shared libraries
    - Device drivers
    - Memory management
    - IP networking
    - Support for multiple file systems



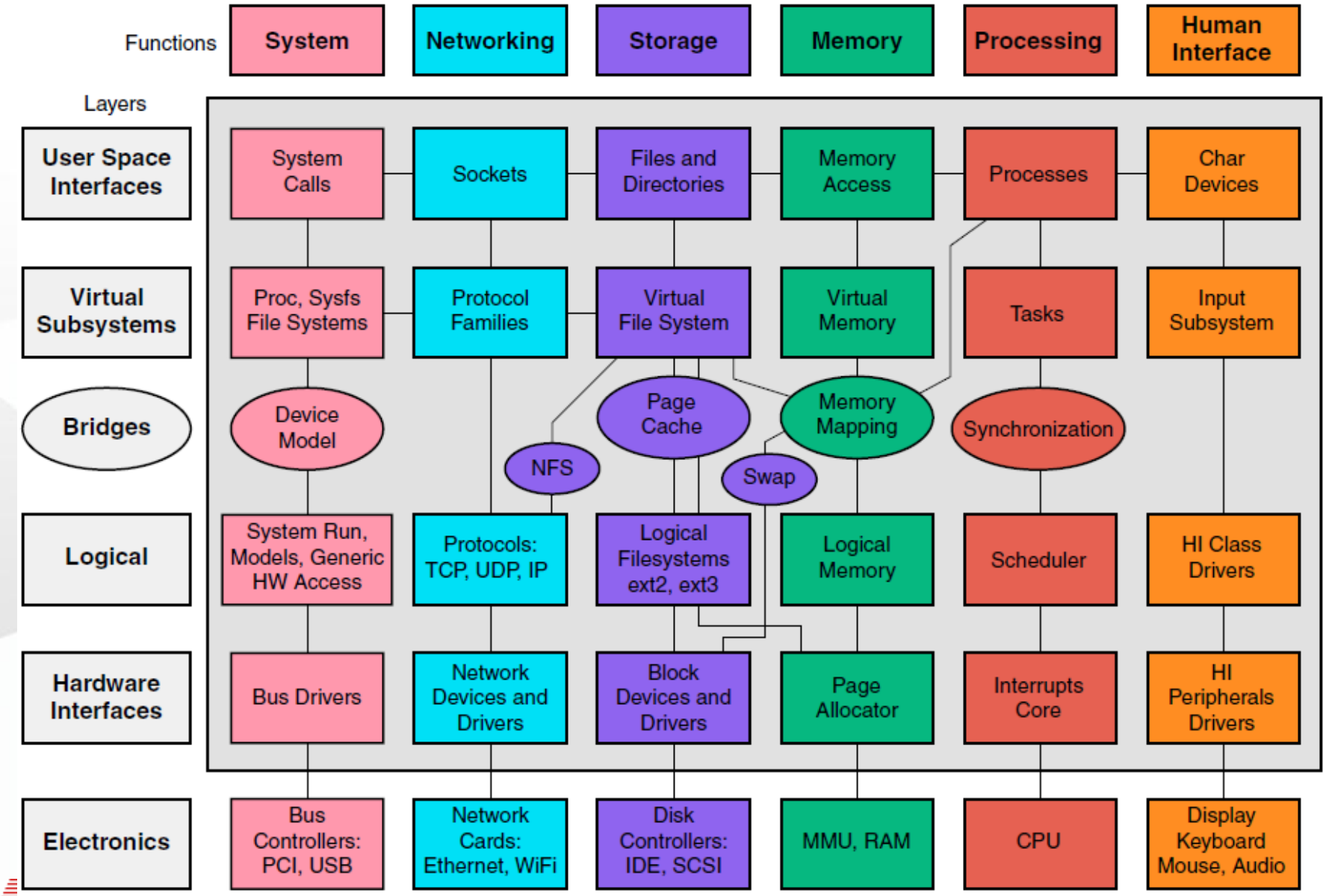
- **Linux kernel is free and open-source software**
  - Licensed under the GNU Public License, v2

Kernel Space Vs. User Space

- **Critical operations must be performed in Kernel-space**
  - Device drivers and kernel code run in kernel-space with full access to the underlying hardware
  - Context switches and interrupt handling performed by kernel
- **Applications run in User-space**
  - Allocated their own memory space using a virtual memory system
  - Access to physical memory space is prohibited except via kernel APIs
- **Replaces standalone BSP**



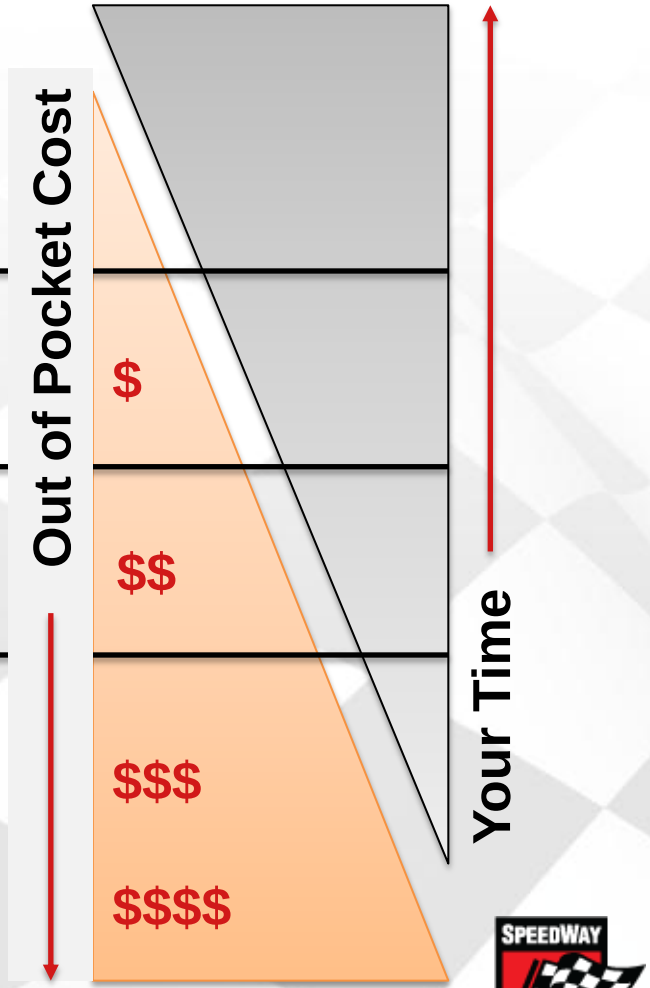
Linux Device Drivers



Linux Kernel Support Model

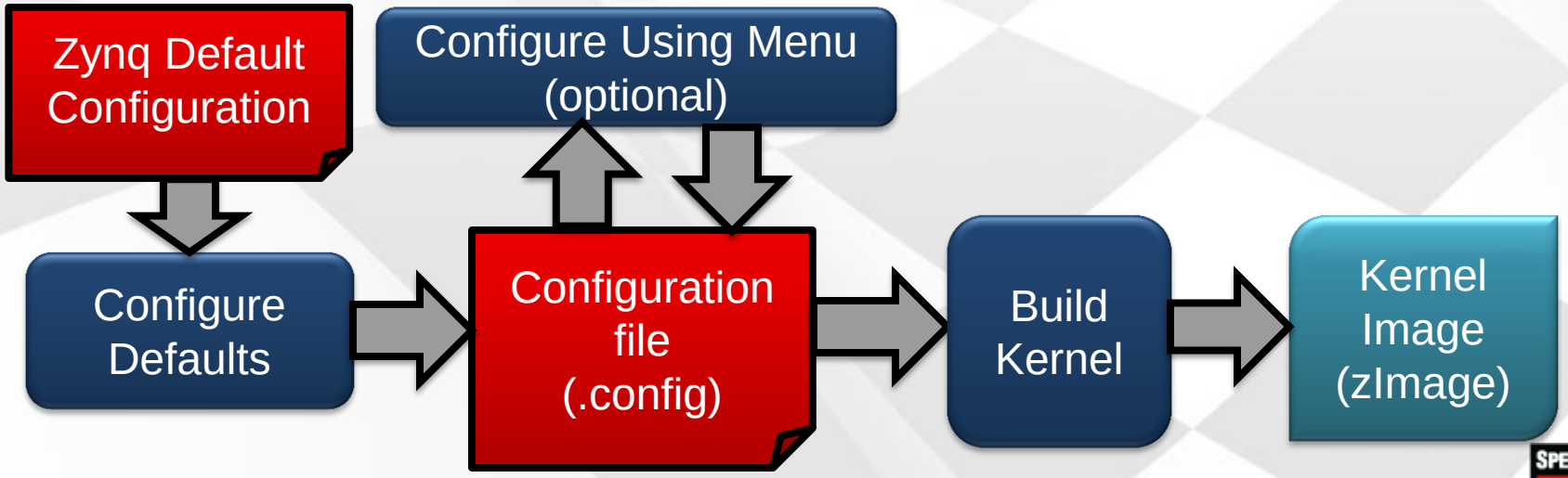
Support can come from many sources…

- **Community-based support**
  - Websites, forums, and mailing lists
- **Silicon vendors and distributors**
- **Board vendors**
- **Paid support options:**
  - Consultants
  - Commercially-available support



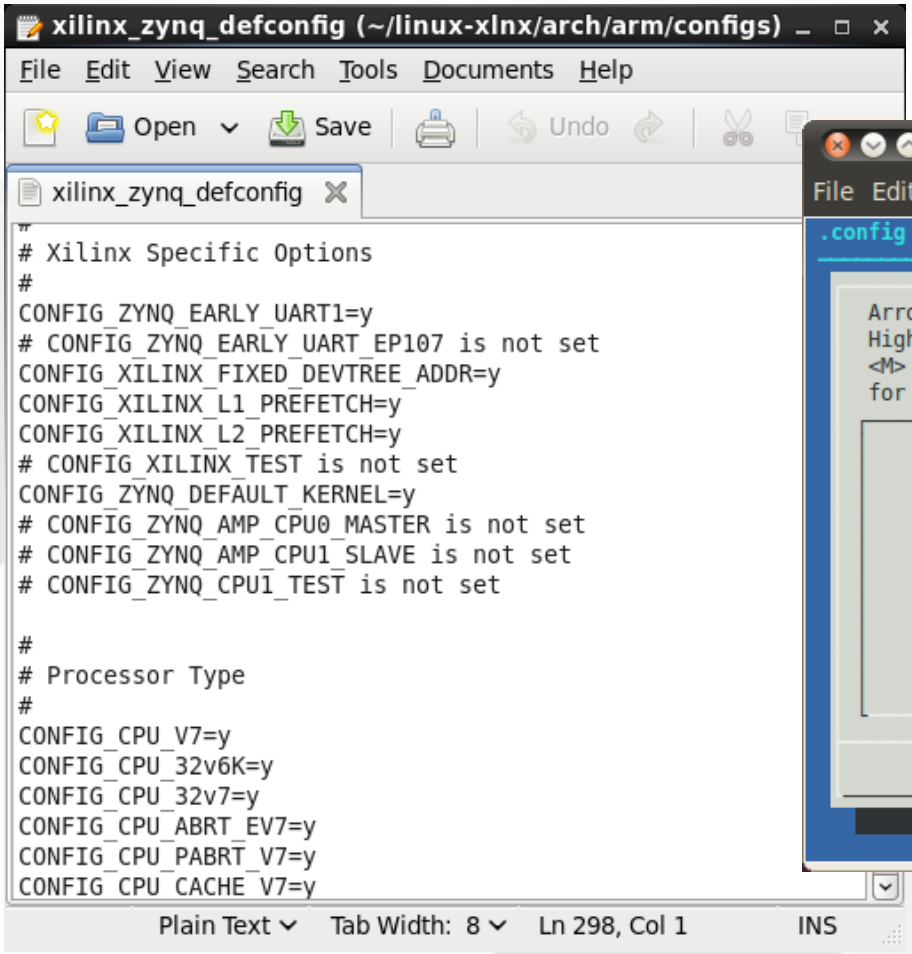
Linux Kernel Configuration and Build

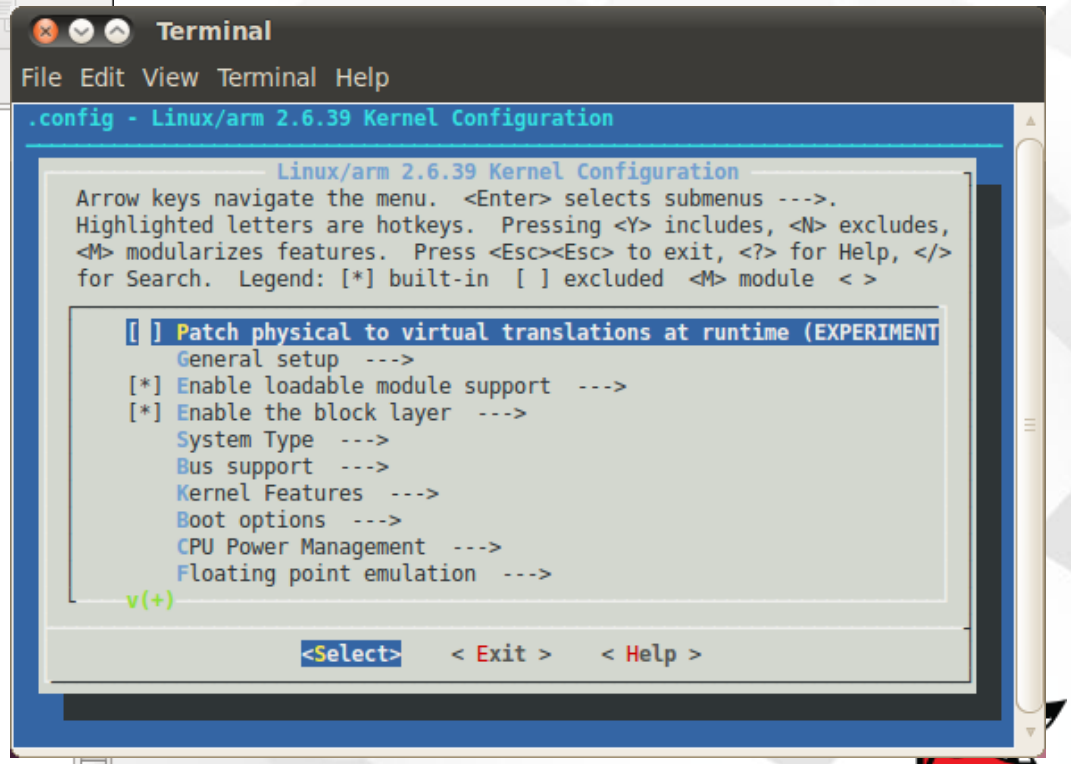
- **Kernel is highly configurable**
  - Add / remove support for
    - Debugging
    - Specific device drivers
    - Types of file systems
    - Boot options



Linux Kernel Basics

- **Default configuration available for Zynq platforms**
  - `xilinx_zynq_defconfig` – Used for ZC702 and ZedBoard
  - Located in `linux-xlnx/arch/arm/configs/` folder





Building the Linux Kernel

- **Kbuild System**
  - Utilizes GNU make tool and a highly specialized set of rules to build the kernel
- **Required environment variables**
  - ARCH=<target architecture, e.g. arm, powerpc, i386>
  - CROSS_COMPILE=<prefix to gcc for cross toolchain>

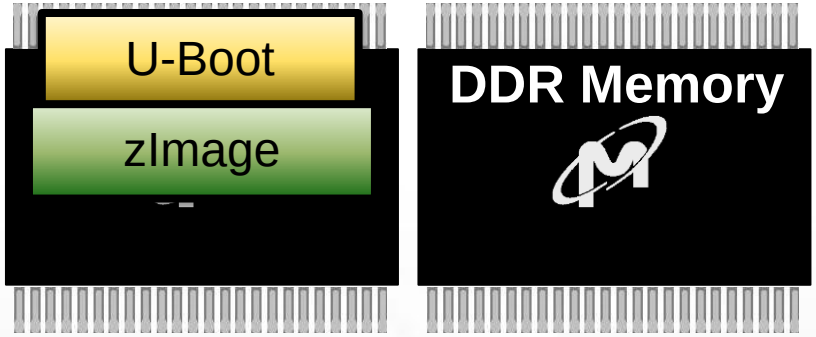
```
> make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
```

Lab 2.1

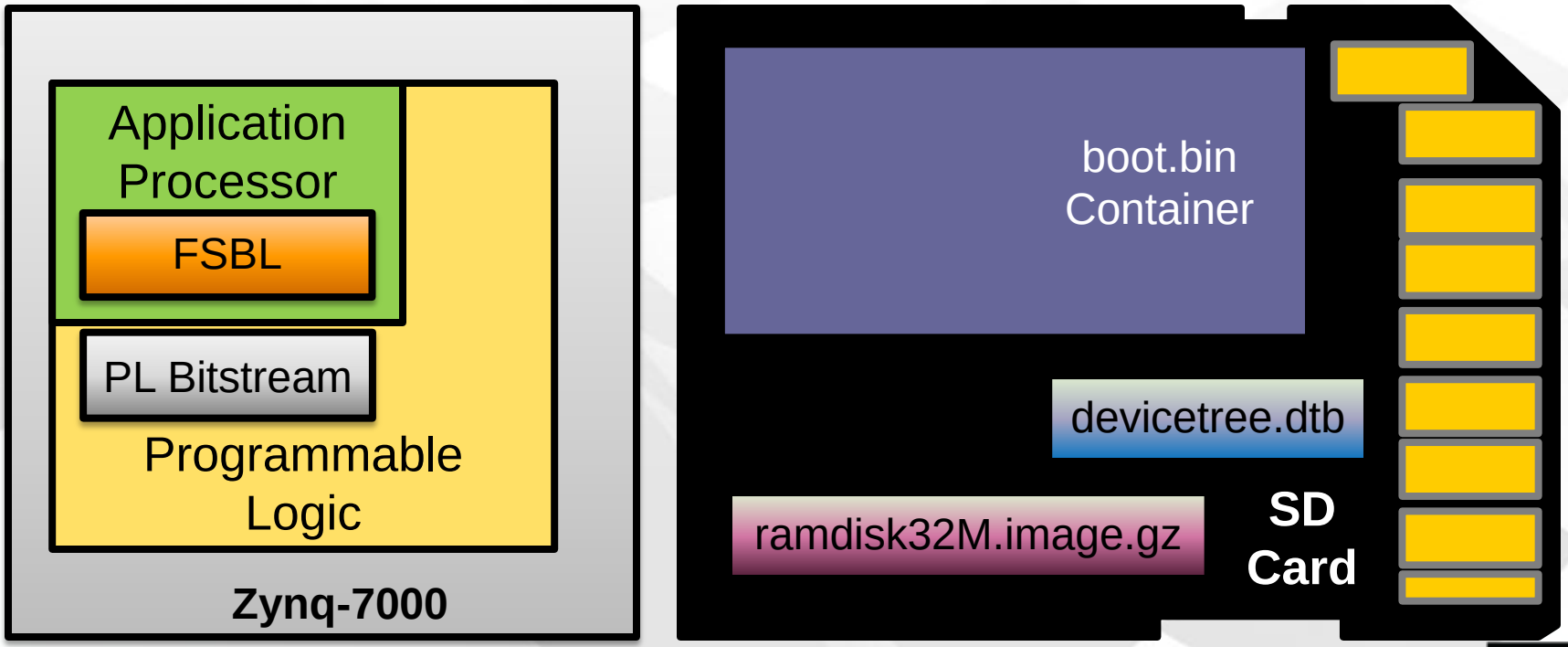
- **Kernel configuration and build from source code**
  - Configure and build Linux kernel source tree for Zynq

- **Note: Experiment 1 is for reference only during the live SpeedWay events and should be performed only by online attendees**

Booting Linux



- **Copy these files to SD Card**
  - devicetree.dtb
  - ramdisk32M.image.gz

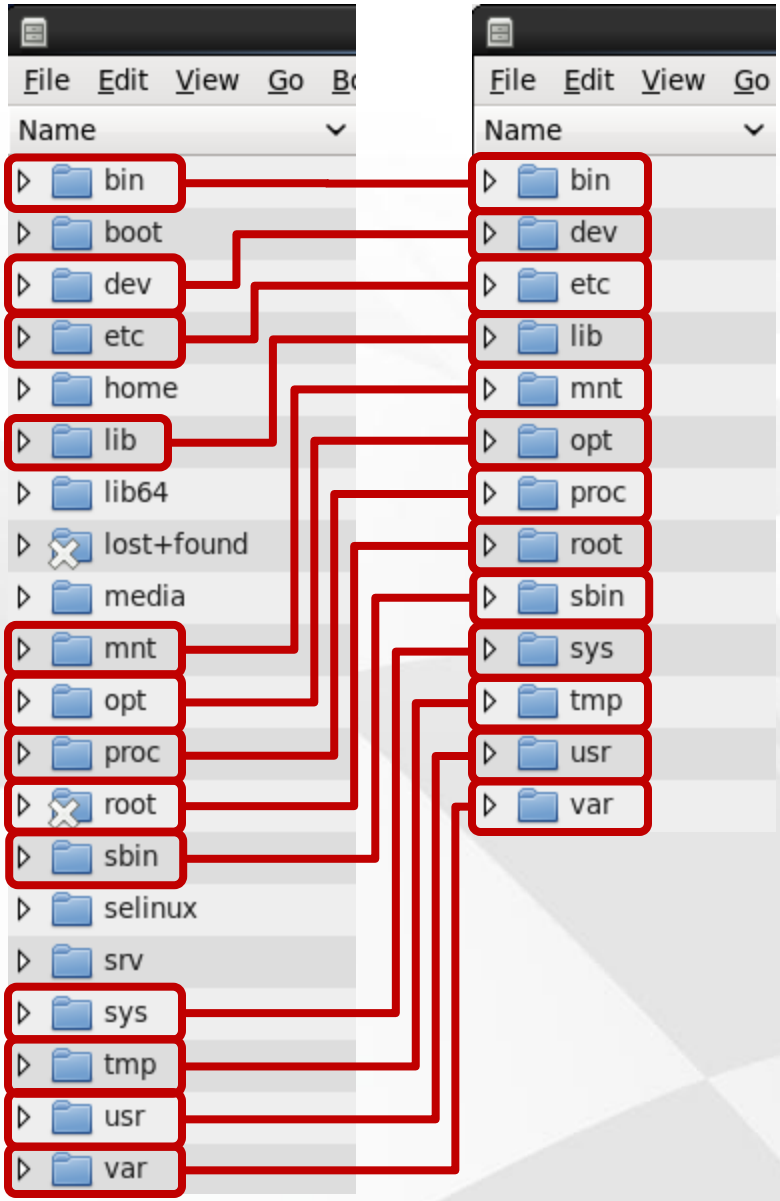


Linux Root File System

- **Root file system is an essential component of any Linux system and contains many critical system components**
  - Applications
  - Configuration files
  - Shared libraries
  - Data files
- **Different file system types used for different media**
  - RAM
  - SD Card
  - NAND/NOR Flash
  - Network
- **Mounted immediately after kernel initialization completes**
- **Contains first application run by the init process**

Desktop vs. Embedded Root File Systems

Desktop (CentOS) Target (Embedded)



Folders common to Desktop Linux:

/dev – System devices (Covered in Part 3)

/root - Storage for super user files
- Each user gets their own folder (e.g. /home/user)
- Similar to "My Documents" in Windows
- "root" user is different, that user's folder is at /root

/mnt - Mount point for other file systems
- Linux only allows one root file system but other disks can be added by mounting them to a directory in the root file system
- Similar to mapping a drive under Windows

/lib - System libraries
- Location of system shared object libraries
- Similar to Windows "C:\Windows\System"

/sys and /proc - Virtual file systems location
- Exposes kernel parameters (kobjects) as files
- Similar to Windows Registry

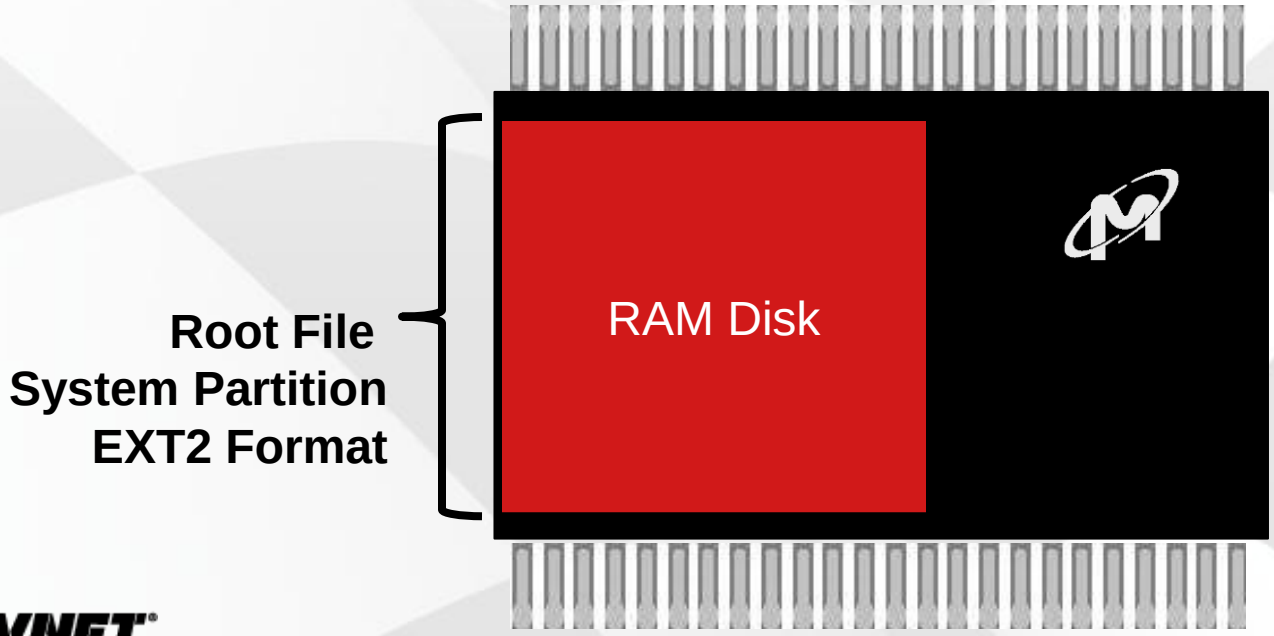
/usr - Storage for user binaries
- Similar to "Program Files" in Windows
- Linux system programs are stored in here

File System Types

Type	Location	Features
FAT32	HDD, SSD, SD Card, USB Drive	File system commonly used with Windows operating system
EXT2, EXT3, EXT4 (EXTended File System)	HDD, SSD, SD Card, RAM	Commonly used file systems rotating and solid-state drives or other devices with drive-like logical interfaces (such as USB, SD Card)
JFFS and JFFS2 (Journaling Flash File System)	Non-volatile memory (Flash)	One of the first flash file systems supported in Linux. Good for smaller file systems (under 64MB)
NFS (Network File-System)	Network Server	Small footprint, read/write, good for development purposes

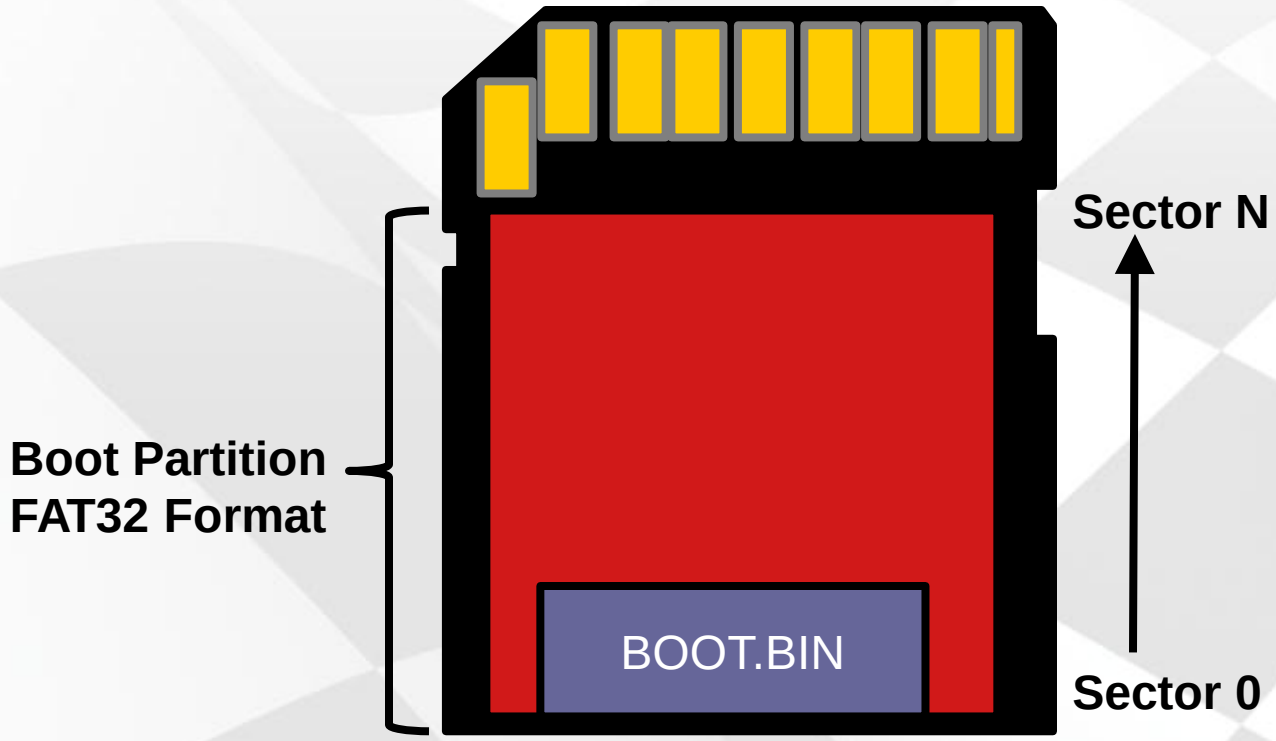
File Systems Used in Labs

- **Kernel can mount a block device image as a RAM Disk**
  - Partition must be formatted to a supported file system type
    - In this case, we're using EXT2 (EXTended file system 2)
  - Read/write, volatile file contents
  - Limited by amount of RAM available on system



File Systems Used in Labs

- **Kernel can mount a partition of the SD Card**
  - Partition must be formatted to a supported file system type
    - In this case, we're using FAT32 (32-bit File Allocation Table)
  - Read/write, non-volatile file contents
  - Can be mounted under the root file system



Lab 2.2

- **Build Busybox**
  - Used as command interpreter and toolset for Linux
- **Build Dropbear**
  - Contains SSH server application for remote control of system
- **Creating the basic root file system**
  - Create directory structure
  - Add critical libraries and user space applications
- **Note: Experiment 1 is for reference only during the live SpeedWay events and should be performed only by online attendees**

Timesys LinuxLink Automates BSP Creation

My project won't build!

Which packages do I need for this application?!!?

I never knew there was a new fix available!

Wasted time!!

Timesys LinuxLink Automates BSP Creation

Design freedom through operating system and architecture choice

CHOOSE **BUILD** **DEPLOY**

Linux Kernel and Drivers
- Latest open-source kernels
- ARM and other ARCH
- Extensive SoC/device support

Hypervisor & BareMetal OS
- Open Source
- Proprietary

Development Tools/Libraries
- Latest version of gcc, glibc, uClibc
- Tested on all supported SoCs
- Log Analyzer, Profiler, Tracer

OS Apps & Middleware
- Rich selection of packages
- Networking, industrial, consumer
- Pre-built, tested, supported

Boot loader
- For supported reference platforms
- Industry-standard U-Boot
- Latest open-source code base

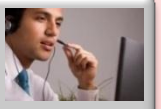
TimeStorm IDE (Eclipse)
- Application development & debug
- Fully integrated with Factory tools
- Compatible with Eclipse ecosystem

Factory Distribution Builder
- Interactive UI with intelligent advice
- Guides your selection of packages
- Web (hosted) and Desktop versions

(Work Orders)

Updates
- Automatic kernel updates
- Automatic middleware updates
- Web-based & desktop notifications

Support
- Web-based
- In-person
- Extensive documentation



Your Custom Tools (SDK)
- gcc / C library/ gdb
- Relevant application libraries

Your Custom Image (BSP)
- Kernel / drivers
- Root file system

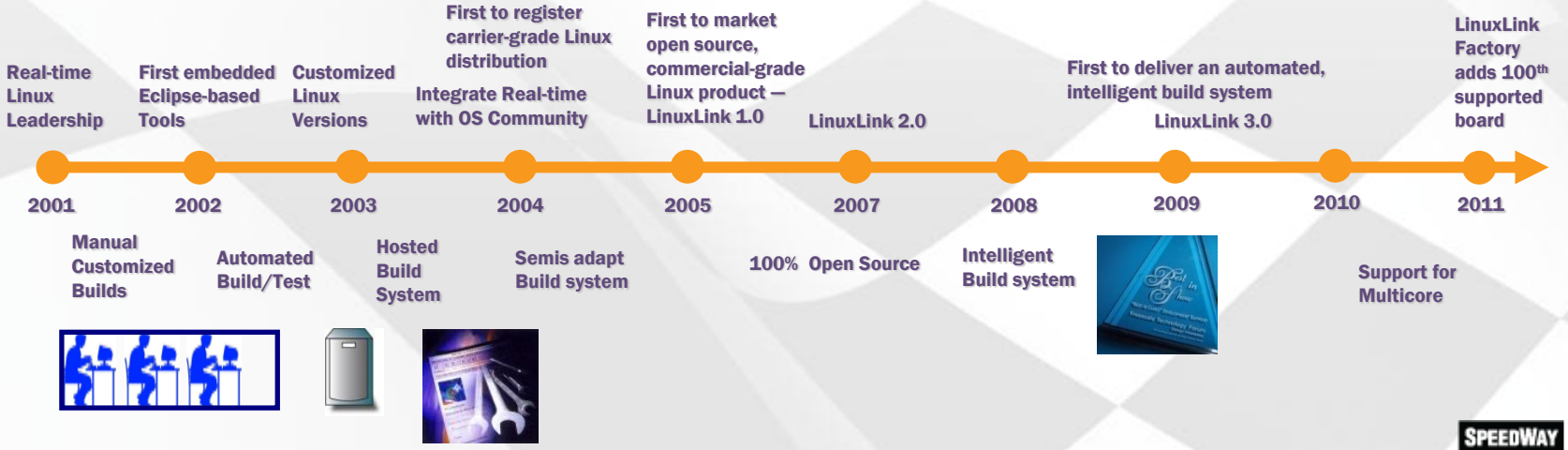
Ready to run on your hardware

LinuxLink Software Development Framework

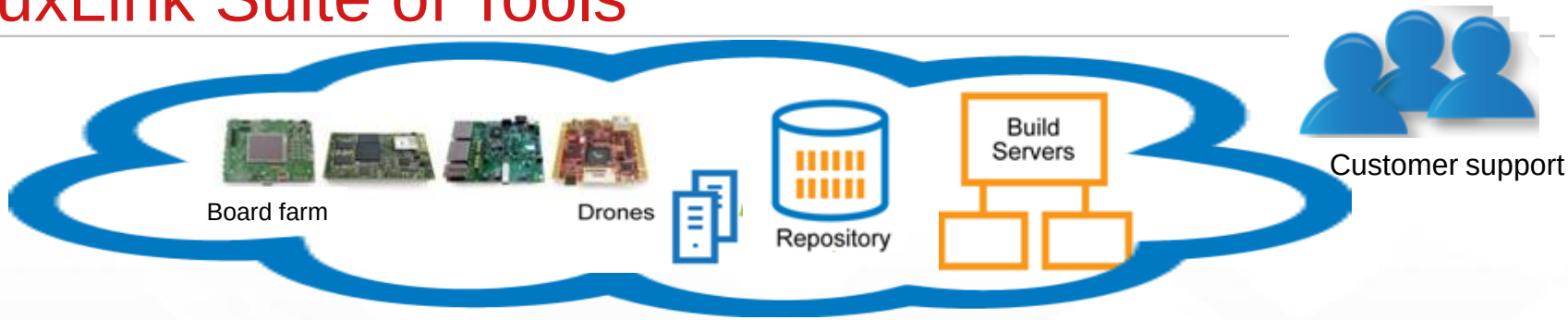
AVNET electronics marketing

Timesys: Linux Industry Pioneer

- Over 1000 projects, 200+ customers
- Applications for industrial automation, networking, mil/aero, medical, consumer and automotive segments
- Supports 8 architectures, 100+ processors/SoCs including Zynq
- Support for multi-core environments
- First to develop and deliver an award-winning, automated, intelligent, embedded Linux build system (LinuxLink 3.0)



LinuxLink Suite of Tools



Web Factory

- RFS Assembler
- Build Engine
- Advice Engine
- Update Engine
- BSP/SDK

Desktop Factory

- Toolchain builder
- Kernel Customization
- Busybox Customization
- Bootloader customization
- Multicore Hypervisor customization
- RFS customization
- Build Engine
- Advice Engine
- Update Engine

Timestorm IDE

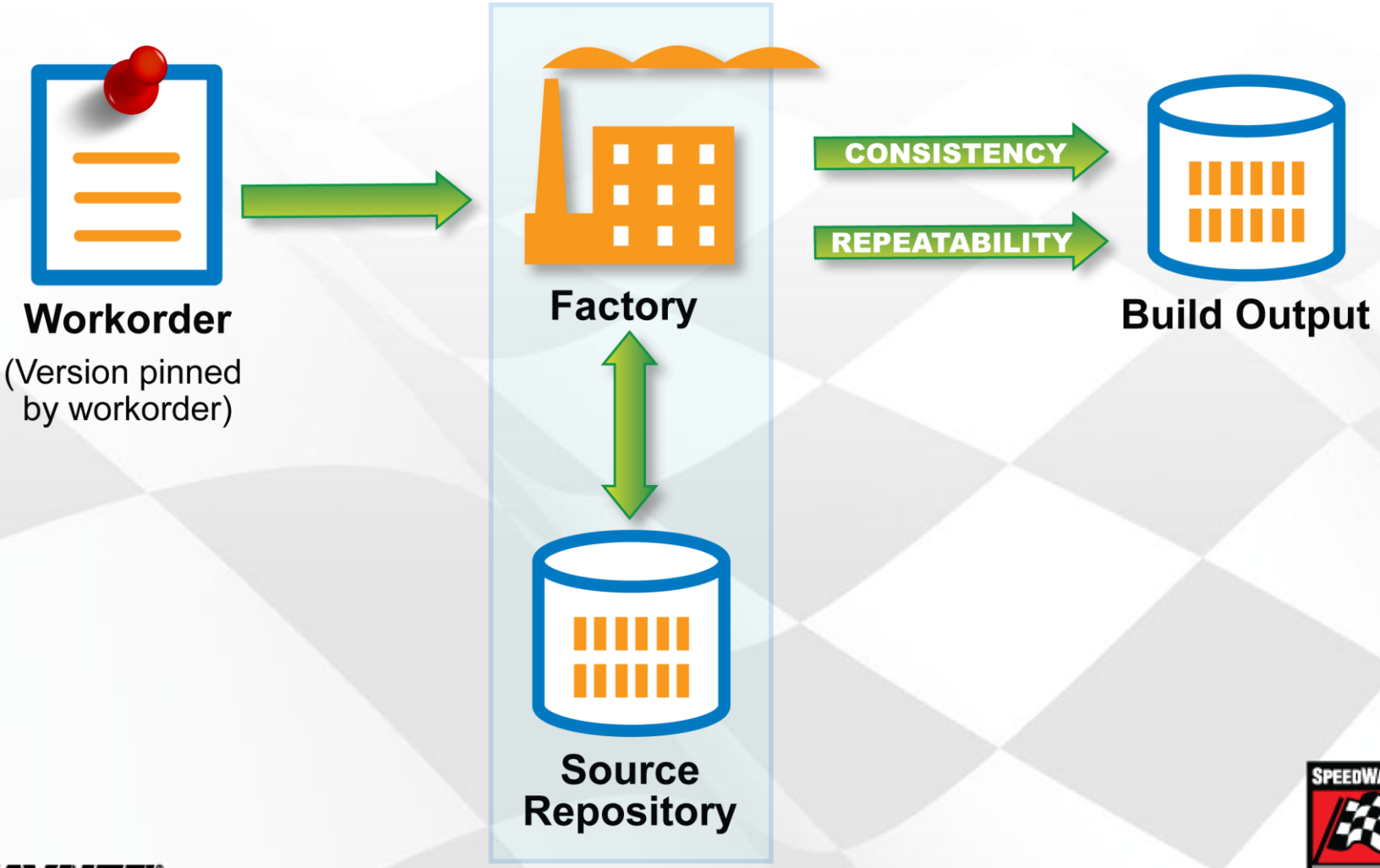
- Target Management Plugin
- Toolchain Management Plugin
- Factory plugin
  - Build Engine
  - Advice Engine
  - Update Engine
- Code Fragments
- Demos
- Qt plugin
- Community Tools
  - LTT plugin
  - Oprofile plugin
  - Memory plugin
  - RSE plugin
  - Egit plugin

Command Line Tools

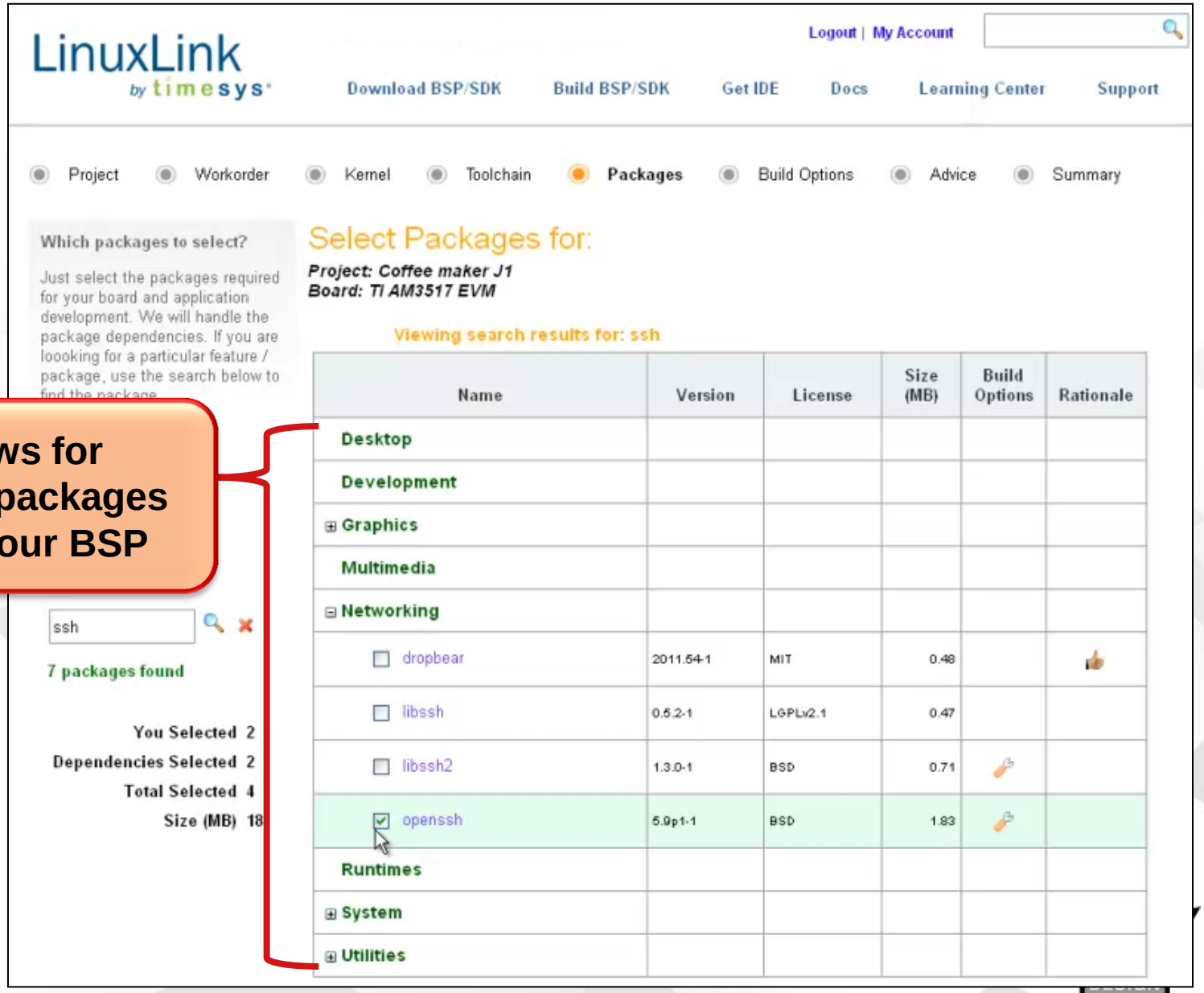
- Libraries & Packages
- Bootloaders
- Kernels
- Dev Tools
- Compliance - Publish OpenSource and build instructions

Timesys LinuxLink

- **Ensure build repeatability for certification and maintenance**



LinuxLink Web Factory



Menu allows for selecting the packages needed for your BSP

Timesys LinuxLink Options

- **Free Edition allows for building BSPs against supported reference boards**

- **Pro Edition offers capability of building for custom board**
  - Dedicated support engineers
  - 24-hour turn-around time
  - Unlimited number of tickets for build related issues
  - Every subscriber (seat) can submit their own tickets

- **Both Free and Pro Editions can be purchased from the Avnet Embedded Software Store**

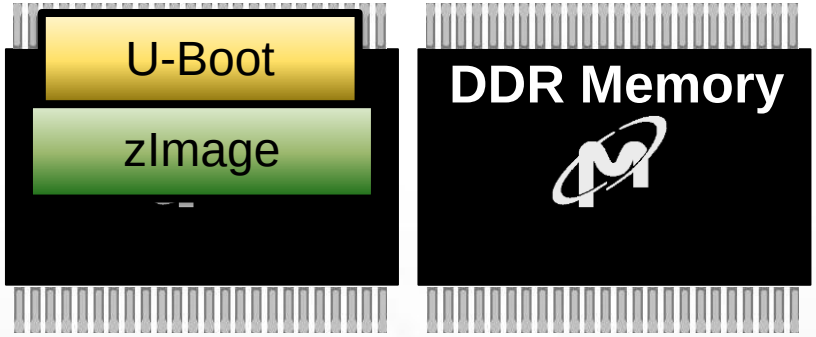
EmbeddedSoftwareStore.com

- **An online information and e-Commerce based website**
- **Dedicated to the embedded design community**
- **The Embedded Software Store aims to accelerate software innovation for embedded applications by establishing an industry marketplace.**
- **This will allows users to easily locate available software supporting the ARM architecture**
- **The site seeks to consolidate a large number of high value software options within a single site**

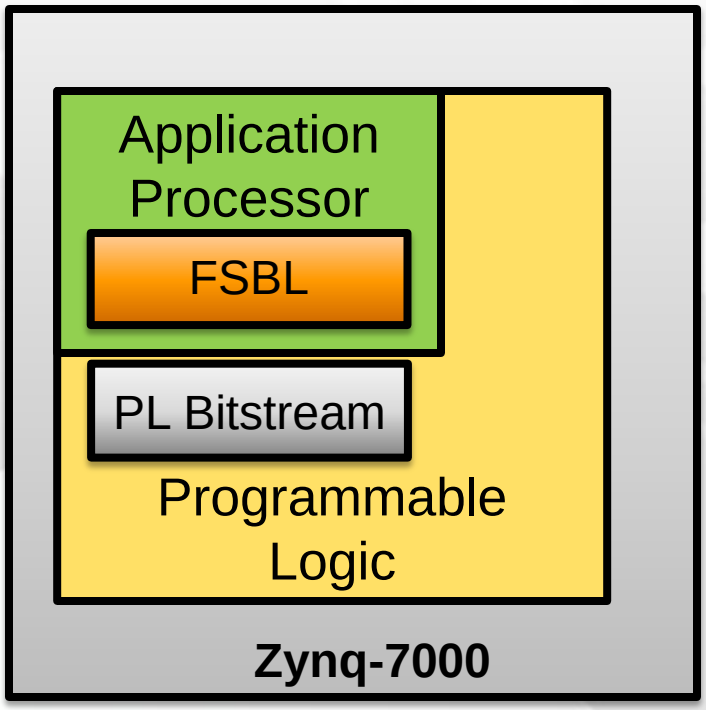
AVNET electronics marketing

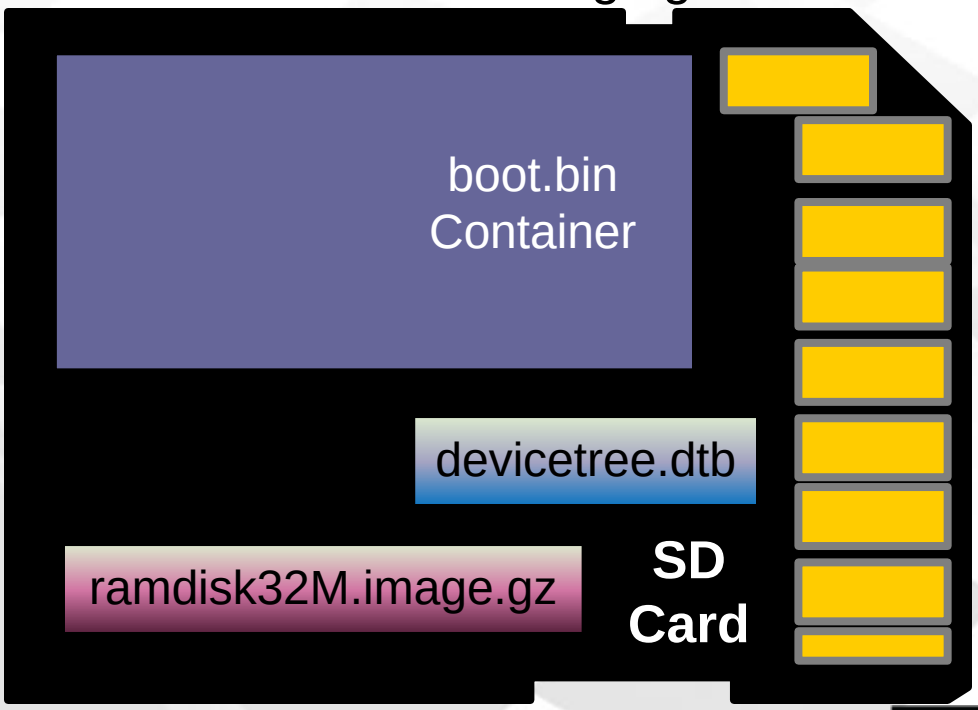
embedded software store™

Booting Linux



- **Copy these files to SD Card**
  - boot.bin
  - devicetree.dtb
  - zImage
  - ramdisk32M.image.gz





Lab 2.3

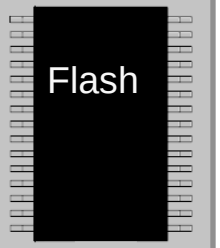
- **Booting Linux**
  - Combine the root file system with the results of previous labs and boot ZedBoard to the Linux command prompt

Linux Device Drivers

Roadmap - Linux in Four Parts

❶ Boot Loading – 3 stages

- Basic hardware initialization
- Loads Linux kernel
- Passing boot parameters

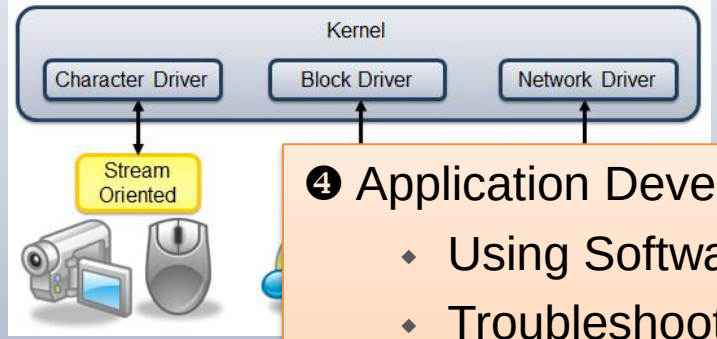


❷ Kernel

- Manages system resources
- Provides application services

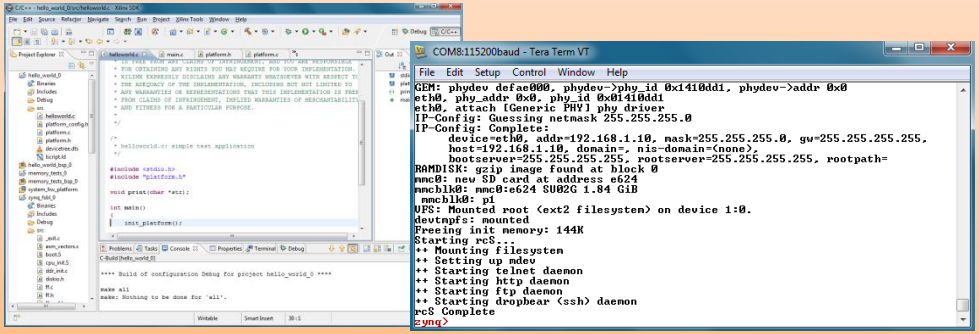
❸ Device Drivers

- Abstracts hardware details from software



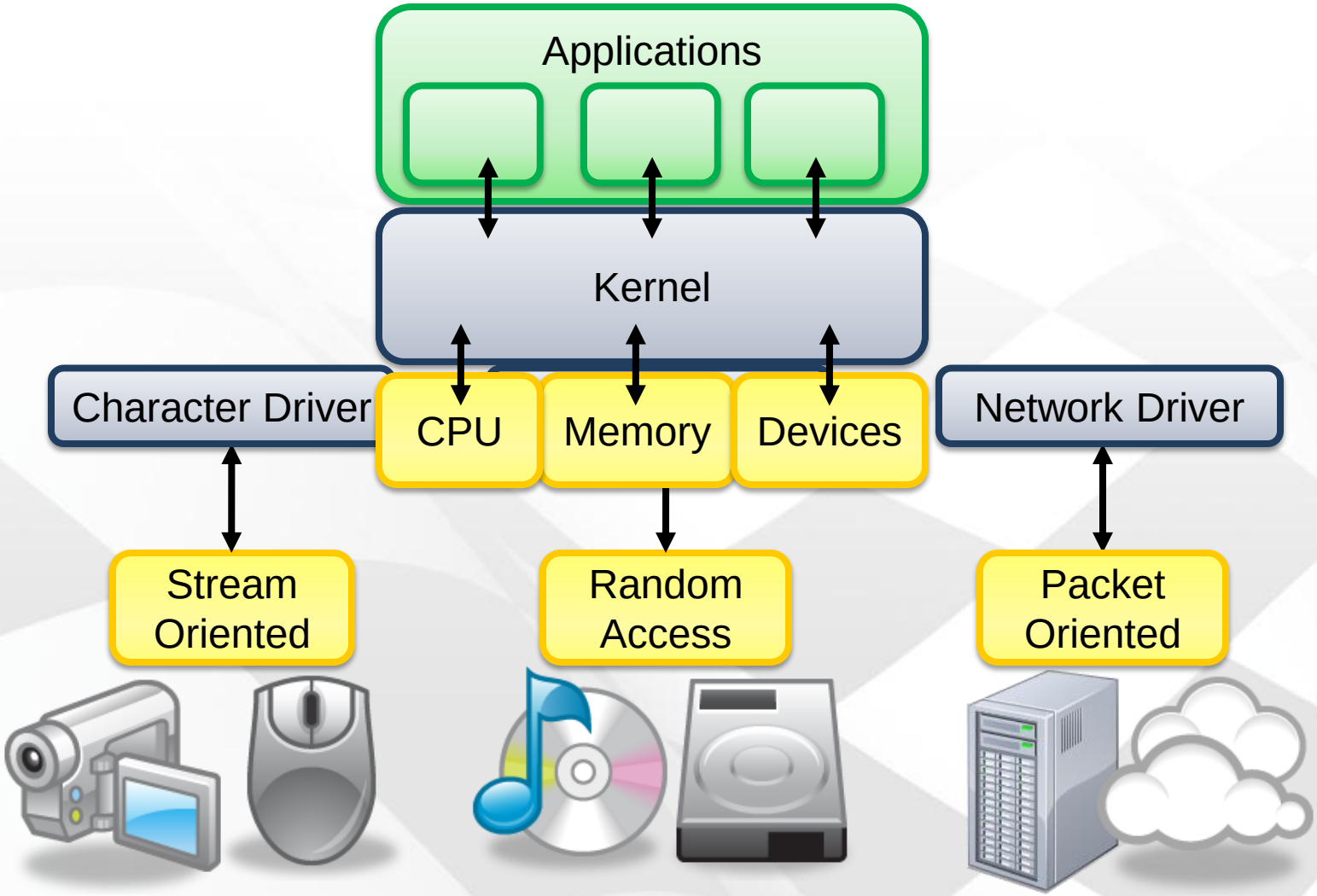
❹ Application Development and Debug

- Using Software Development Kit
- Troubleshoot user applications



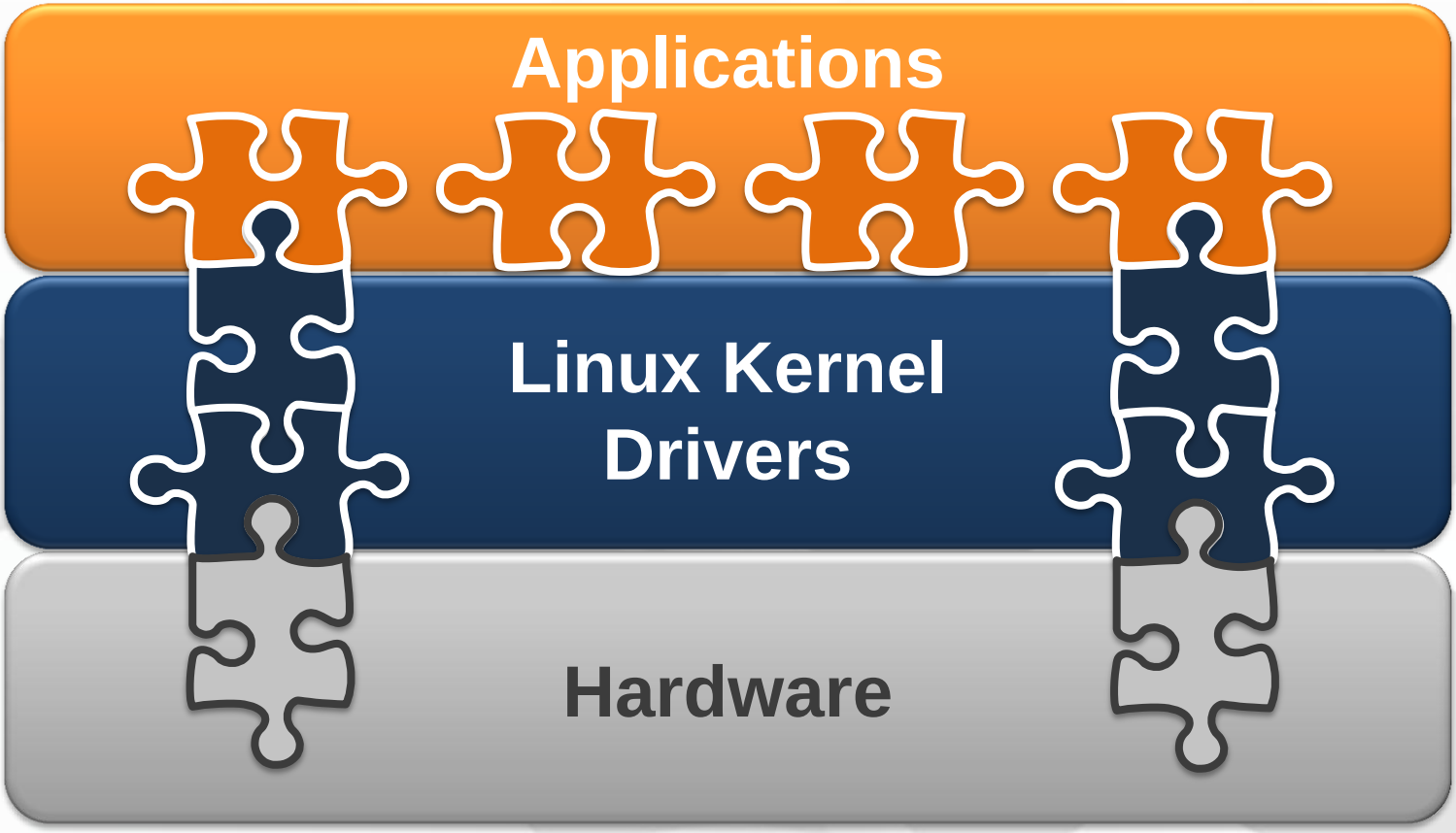
AVNET electronics marketing

Linux Device Driver Types



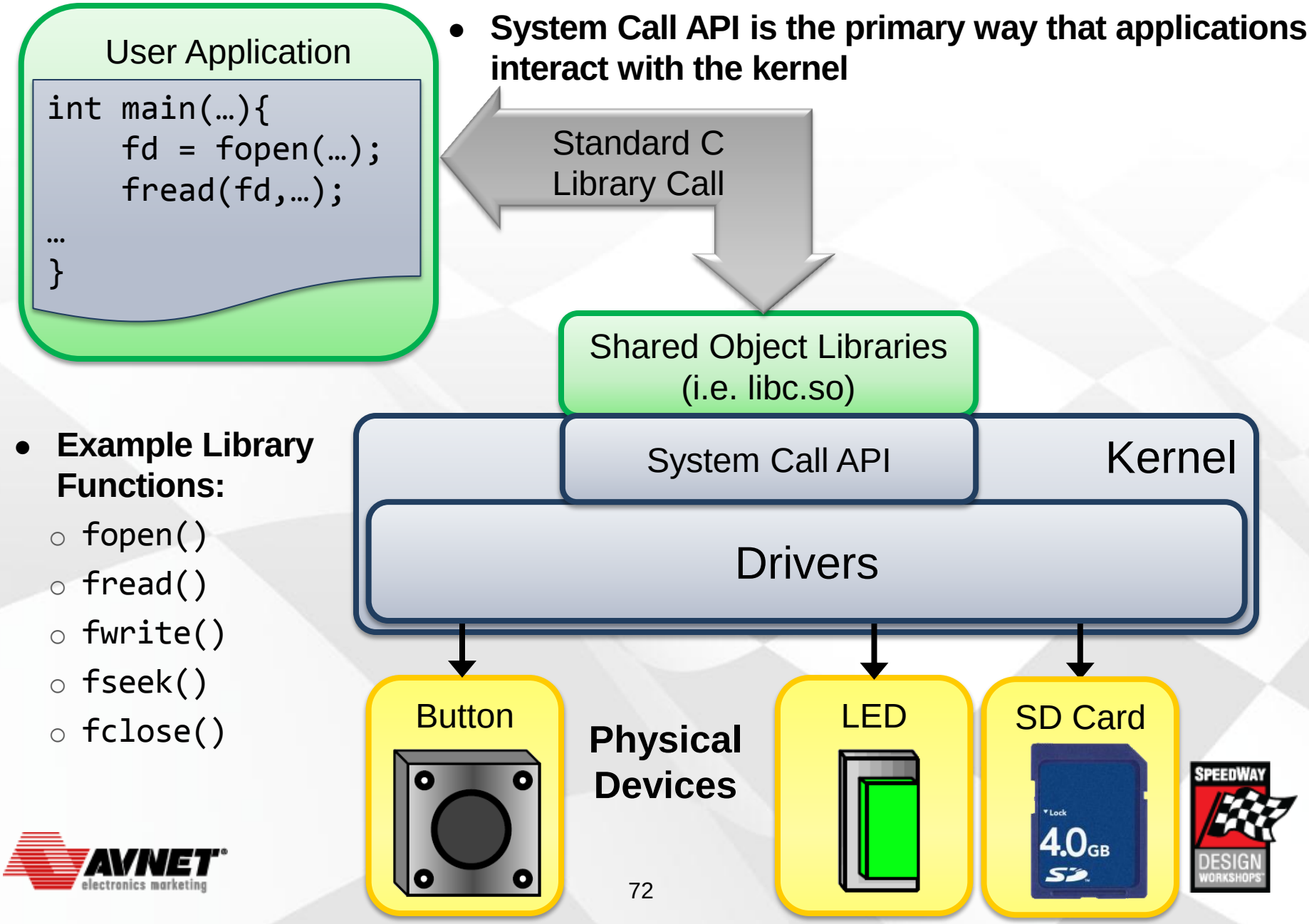
Linux Device Driver Modularity

- **Provide access to physical resources**
- **Built-in or loaded at run-time (loadable modules)**
- **Can be multi-layered subsystems (USB, I2C, Ethernet)**



System Call API

- **System Call API is the primary way that applications interact with the kernel**



- **Example Library Functions:**
  - `fopen()`
  - `fread()`
  - `fwrite()`
  - `fseek()`
  - `fclose()`

Traditional Device Drivers vs. sysfs

- **Traditional /dev devices**
  - Handles streaming data (i.e. audio/video)
  - Efficient exchange of binary data and structures rather than individual text strings
  - Protection from simultaneous access

- **Device drivers under sysfs**
  - Limited to simple single value
  - Easy access to data via shell scripts and user space programs

Weigh the tradeoffs to decide which solution is appropriate for your own application

System vs. Library Function Calls

Open /dev/led-brightness

Application calls write()

Kernel code

Call device driver write() handler function

write() handler function

Logic to determine correct action (write register)

Write IO space register

Return number of bytes written (numerical value)

Best for large data transactions

Open /sys/class/gpio/gpioXX/value

Application calls fwrite()

Kernel code uses device attribute to determine function to call

Call sysfs device driver write() handler function

sysfs write() handler function

Decode text string

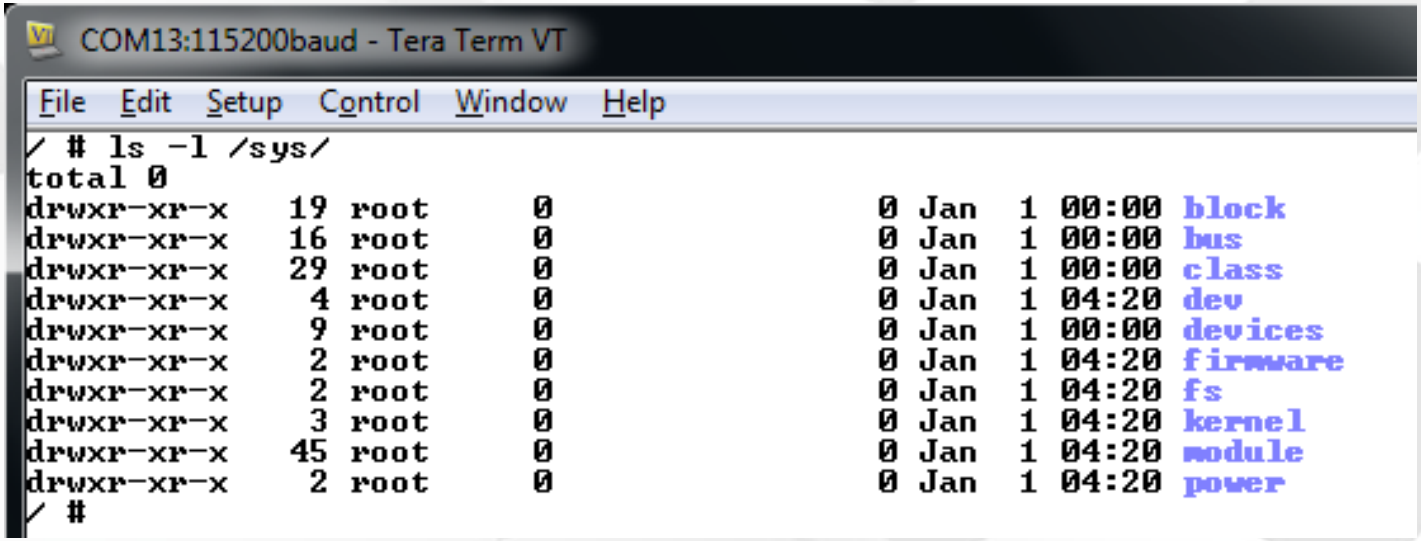
Write MIO GPIO register

Return write status

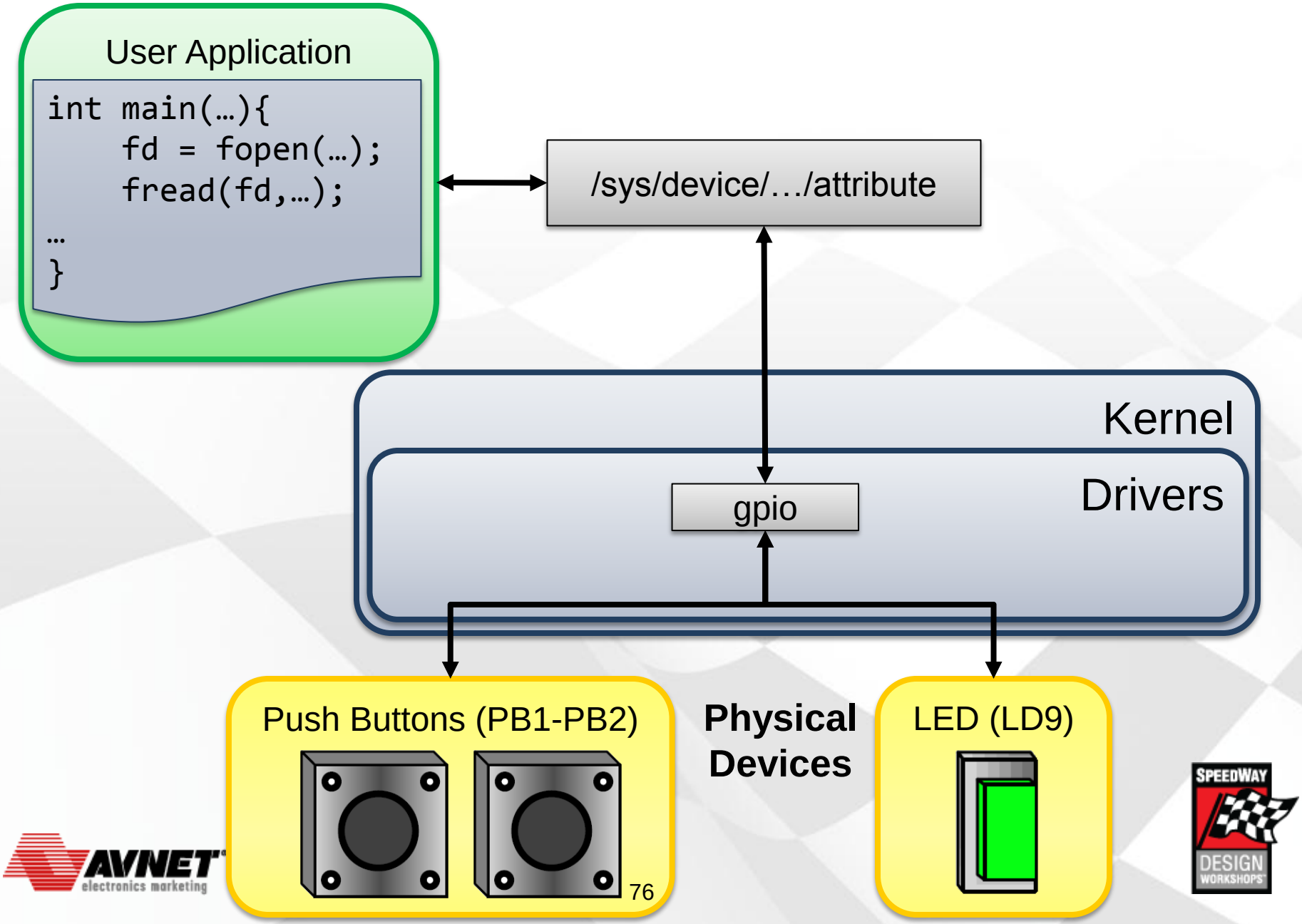
Effective for single control/status transactions

Details on sysfs

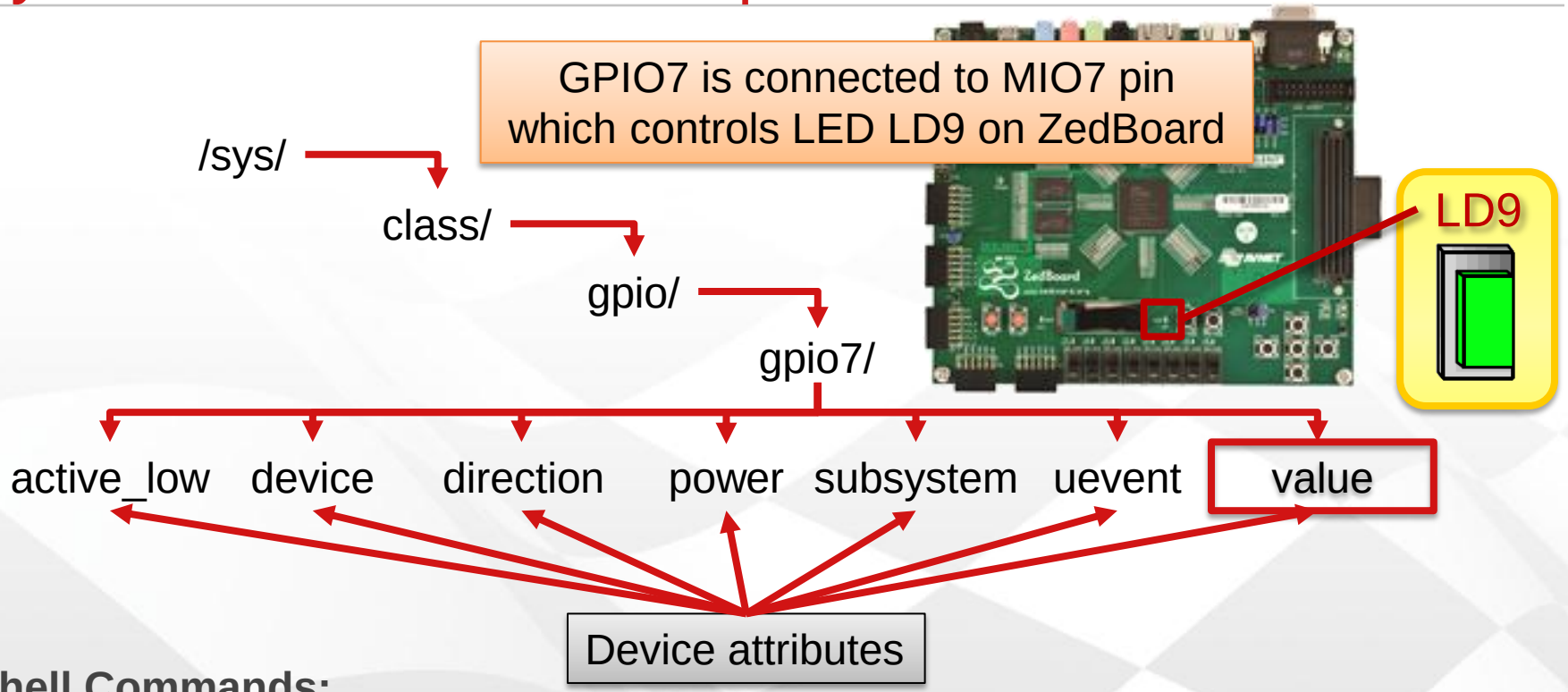
- **The sysfs subsystem is a dynamic “virtual” file system**
- **Mounted on /sys**
- **Provides a representation of the device model which is visible from user space**
  - Device attributes visible in the form of regular files
  - Allows get/set capability of specific parameters from user space



Example Device File Mapping



Sysfs Device Driver Example



Shell Commands:

```
/sys/class/gpio/gpio7 # echo 1 > value
/sys/class/gpio/gpio7 # cat value
```

Very convenient for configuring and controlling devices using shell scripts

C code:

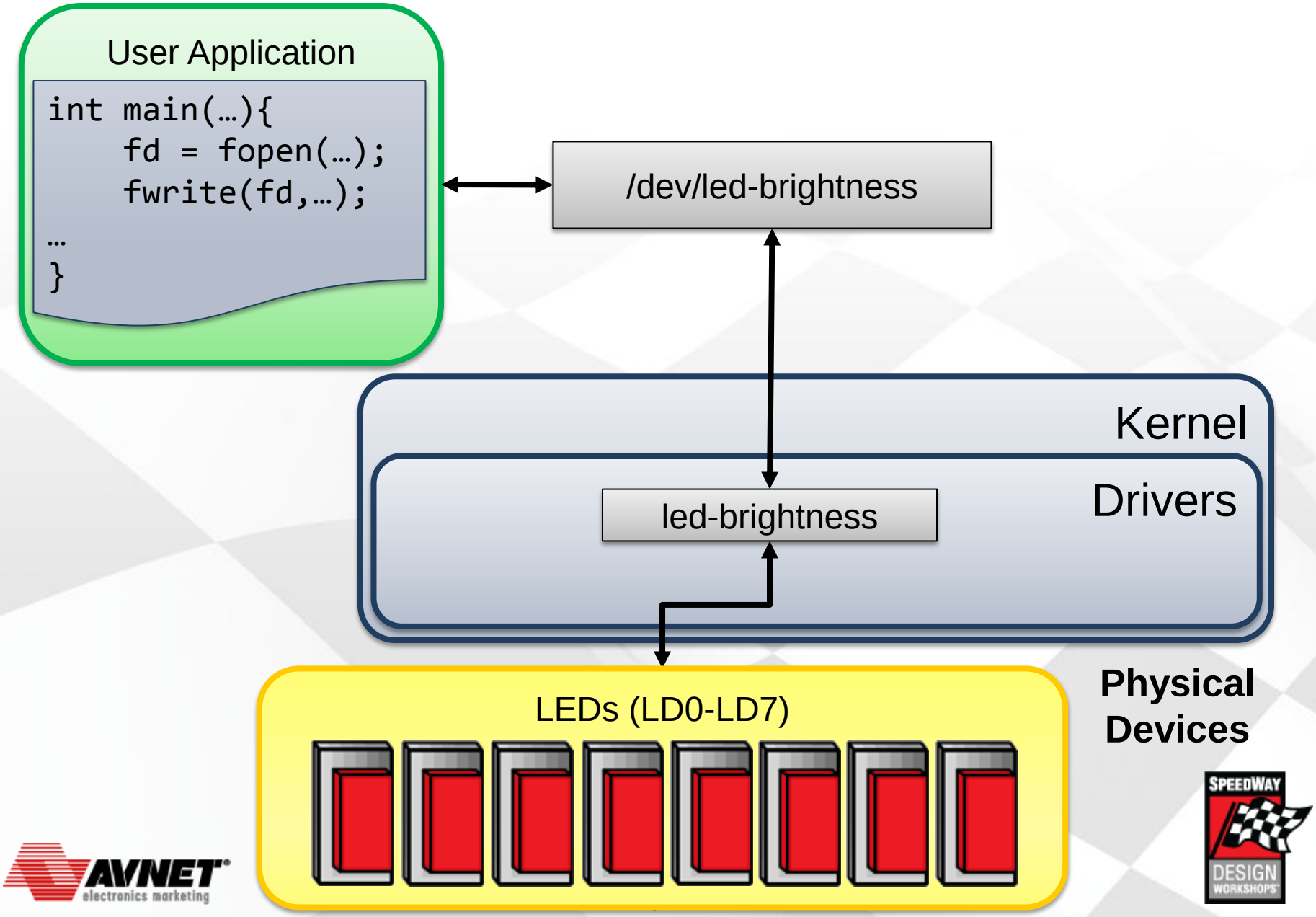
```
fprintf(file_led7, "%d", 1);          /* write */
fscanf(file_led7, "%d", &n_ch);       /* read */
```

See kernel document **Documents/gpio.txt** in kernel source tree for further details

Lab 3.1

- **Device drivers and MIO GPIO hardware**
  - Explore Sysfs and interact with ZedBoard hardware devices

Example Custom Device Driver



Linux Kernel Patching

- **Patches are applied to code bases to duplicate the desired modifications**
- **Patches are applied using:**
  - patch – a utility for applying a .patch file contents to code files or source directory trees

```
$ patch -p1 < add_lcd_panel_support.patch
```

  - Software version control systems such as git, svn, and cvs use internal apply mechanisms

```
$ git apply add_lcd_panel_support.patch
```

Patch Creation

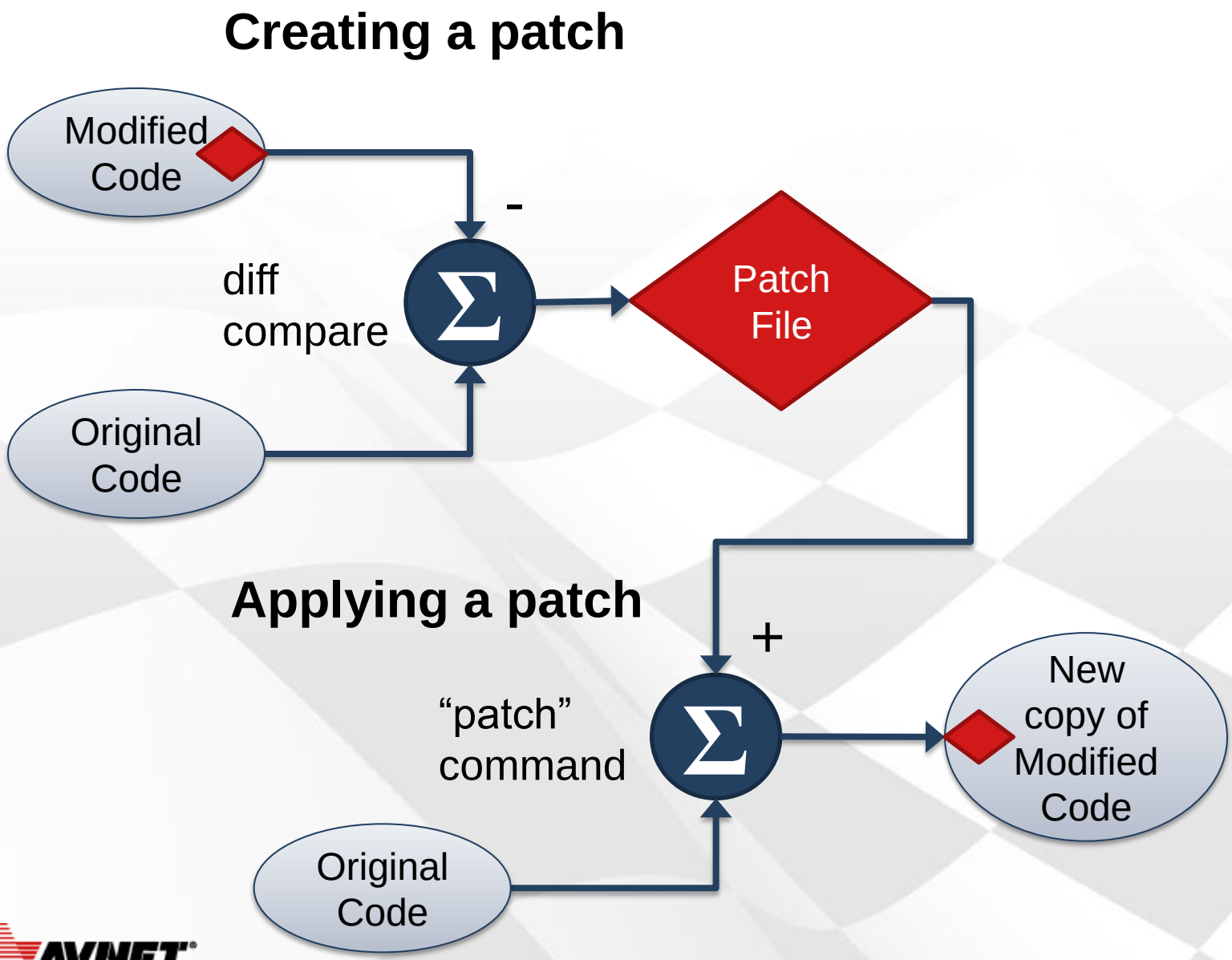
- **Vast majority of modifications to open source code bases are patch files**
- **Patch files are the difference between the original and the modified code**
- **Patches are generated using:**
  - diff – a utility for analyzing the differences between two code files or source directory trees

```
$ diff –u original.c new.c > add_lcd_panel_support.patch
```

  - Software version control systems such as git, svn, and cvs (Git is used by U-Boot and Linux kernel developers)

```
$ git format-patch -1
```

Patch Lifecycle



Kernel Device Tree

- Device Tree data structure describes the system hardware
- Data structure passed to the kernel at boot time
- In general, device nodes for IP cores take the following form:

```
(name): (generic-name)@(base-address) {
        compatible = "xlnx,(ip-core-name)-(HW_VER)"
                [, (list of compatible devices), ...];
        reg = <(baseaddr) (size)>;
        interrupts = < ... >;
        interrupt-parent = <&interrupt-controller-phandle>;
        xlnx,(parameter1) = "(string-value)";
        xlnx,(parameter2) = <(int-value)>;
};
```

- The device tree structure reflects the bus attachments as made in the system.mhs file in XPS tool

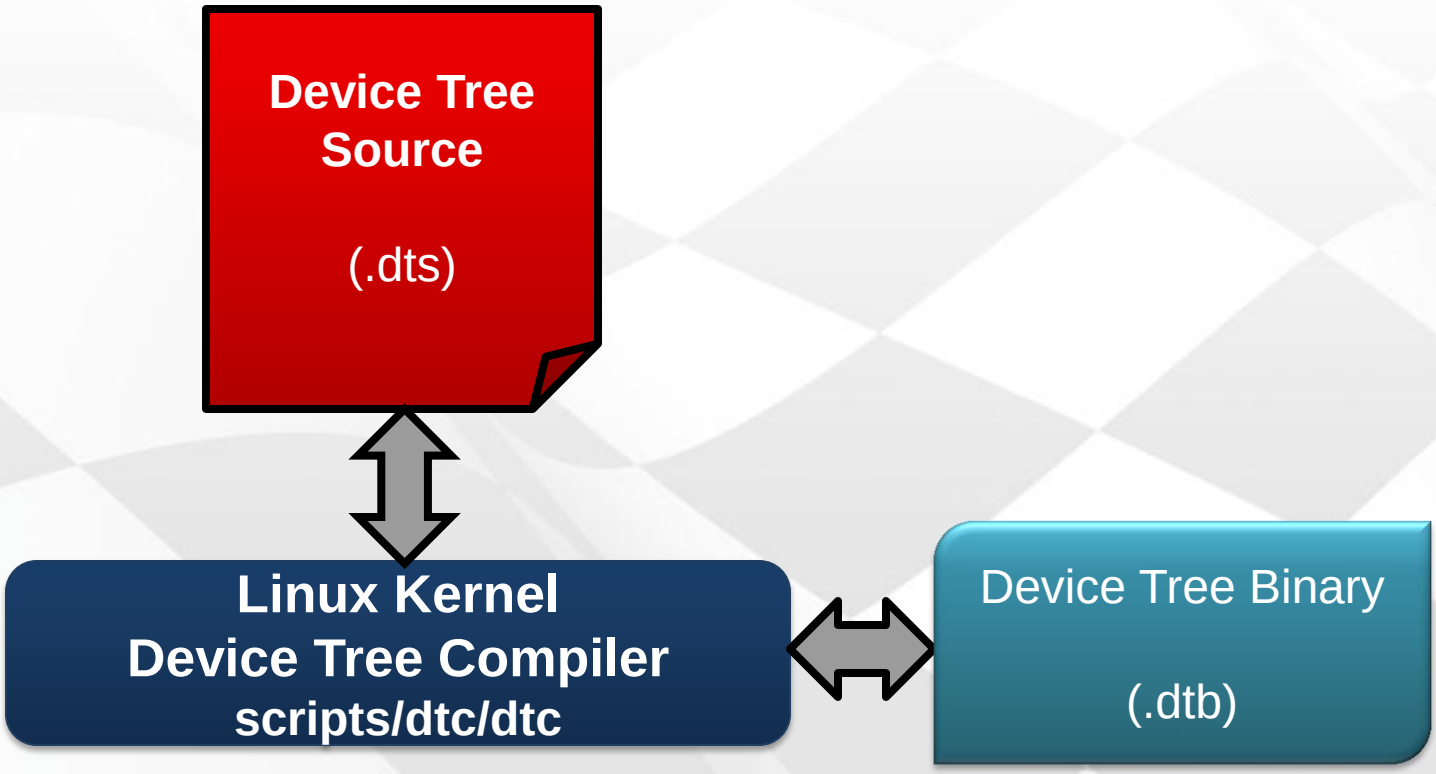
Device Tree Example

- Device trees are created in a human readable, text-based source format known as a Device Tree Source (DTS) file

```
        memory {
                device_type = "memory";
                reg = <0x0 0x20000000>;
        };

        chosen {
                bootargs = "console=ttyPS0,115200 root=/dev/ram rw initrd=0x2000000,32M
                linux,stdout-path = "/amba@0/uart@E0001000";
        };

amba@0 {
                compatible = "simple-bus";
                #address-cells = <0x1>;
                #size-cells = <0x1>;
                ranges;

                led-brightness@41200000 {
                        compatible = "avnet,led-brightness";
                        reg = <0x41200000 0x20>;
                };
        };
```

Device Tree Compiler

- DTS files are compiled with a Device Tree Compiler (DTC) to create a machine readable binary that is passed to the kernel
- The resulting binary is known as a Device Tree Binary (DTB)



Lab 3.2

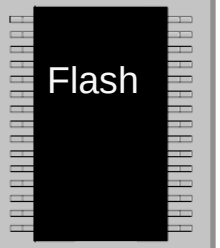
- **Device drivers for PL devices**
  - Edit device tree script and create the device tree binary file
  - Using a test bench, interact with custom logic peripherals across AXI-Lite interconnects

Linux Application Development and Debug

Roadmap - Linux in Four Parts

❶ Boot Loading – 3 stages

- Basic hardware initialization
- Loads Linux kernel
- Passing boot parameters

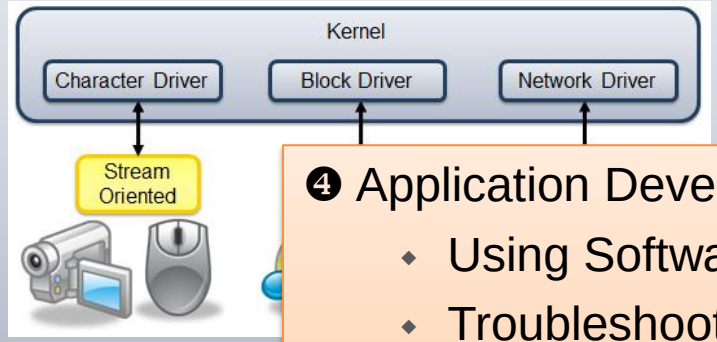


❷ Kernel

- Manages system resources
- Provides application services

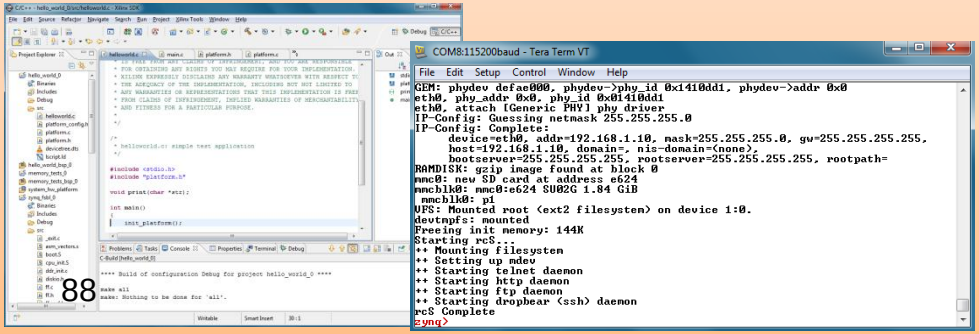
❸ Device Drivers

- Abstracts hardware details from software

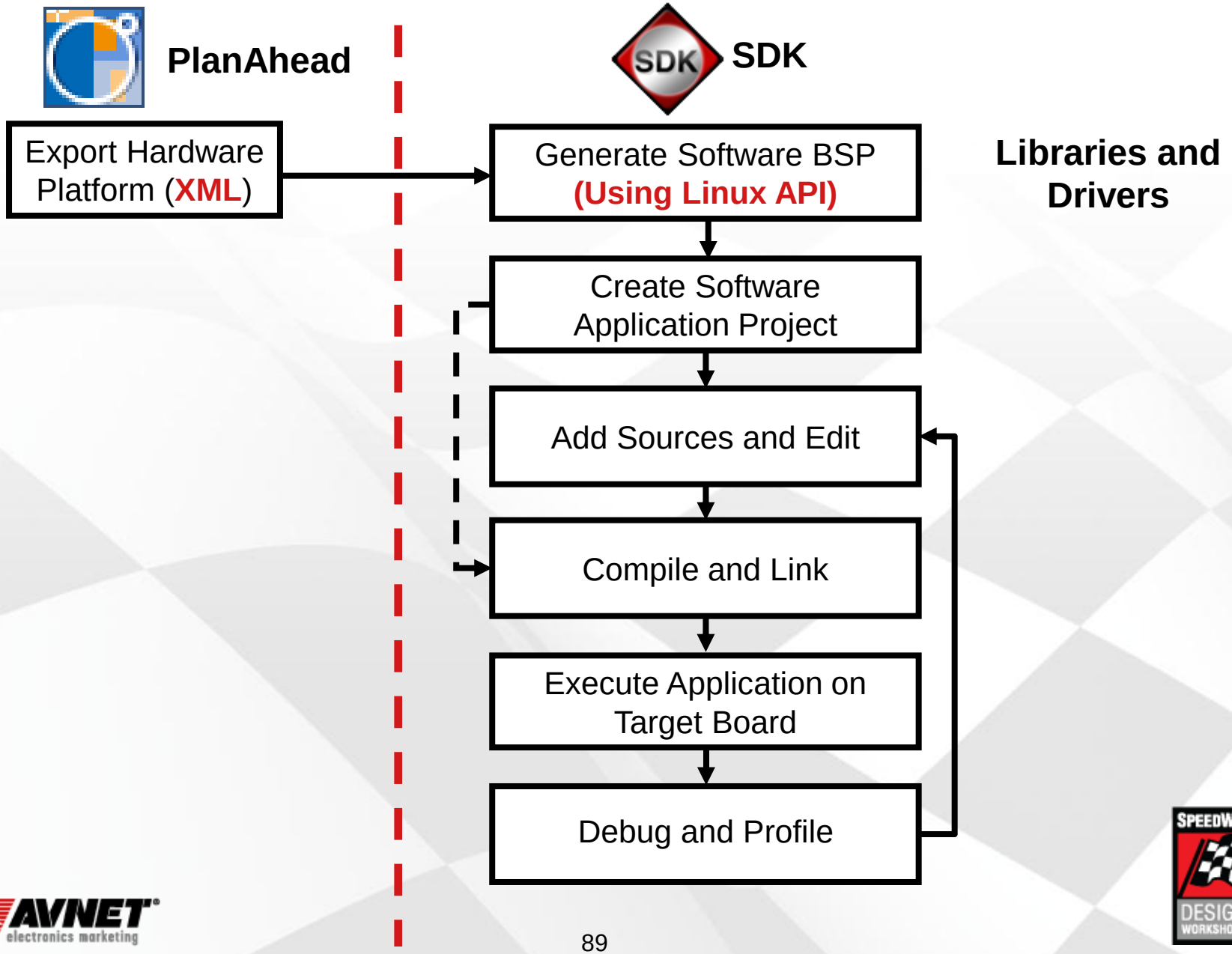


❹ Application Development and Debug

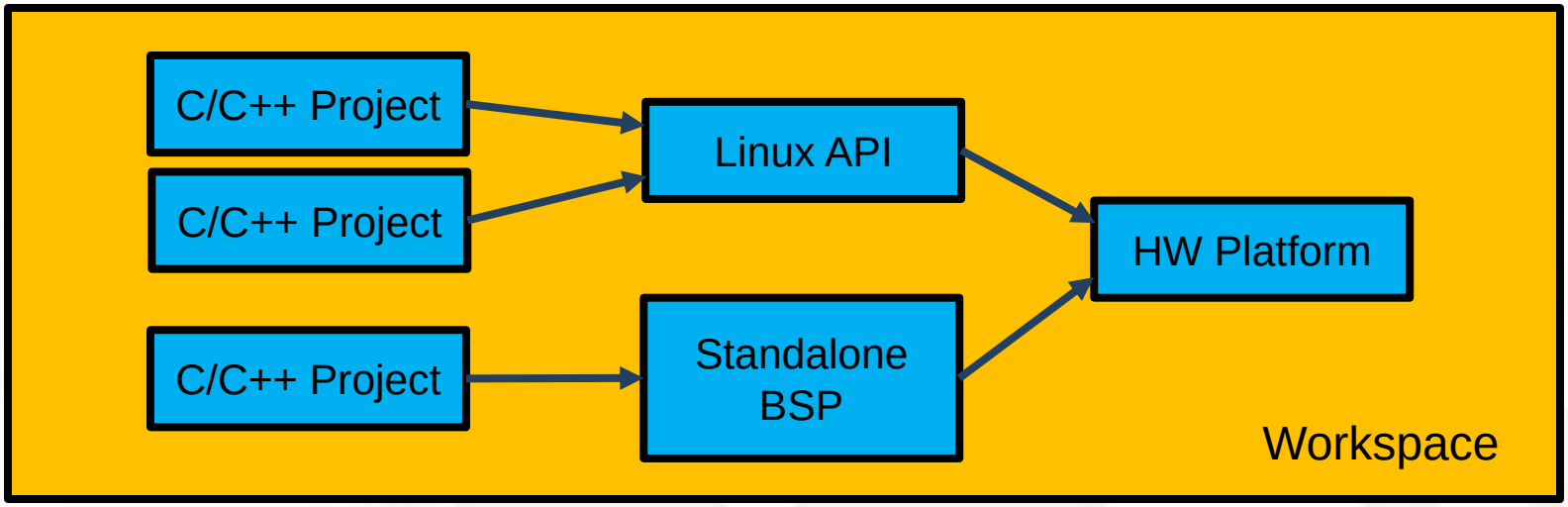
- Using Software Development Kit
- Troubleshoot user applications



Linux Application Development Flow



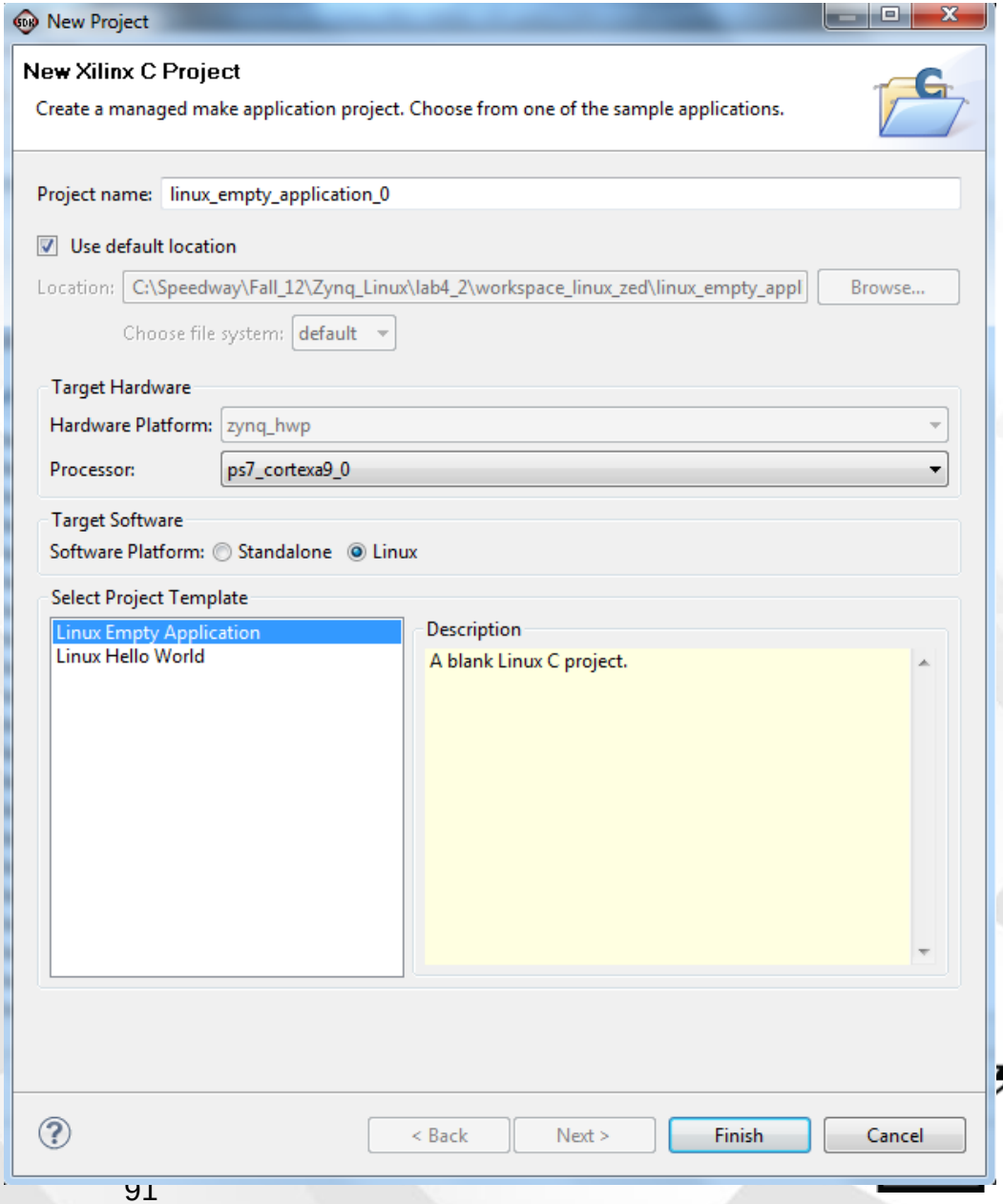
Application Dependencies



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

Xilinx C Project Wizard

- **Target applications to a Linux Software Platform**



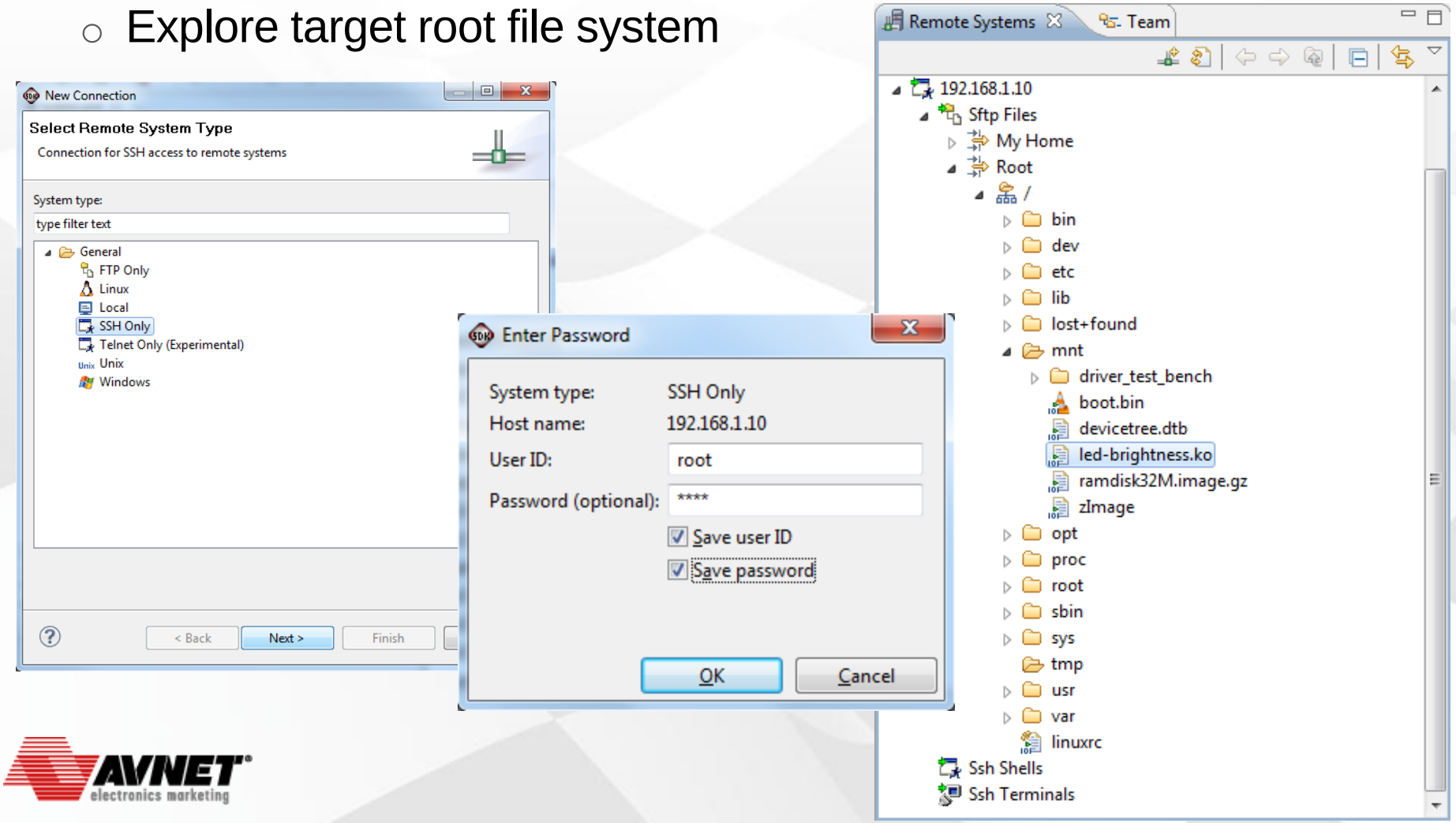
Writing Code

- **The Eclipse Editor has many useful features**
  - Remote System Explorer (Linux Target Only)
  - Code completion
  - C/C++ content assist
  - Hovering
  - Code folding
  - Refactoring
  - Shows inactive code blocks
  - Quick keys for most actions

- **The Editor can be customized in the Eclipse preferences**
  - Numerous options to help facilitate a particular coding style

Remote System Explorer

- **Secure remote network connection to target**
  - Transfer files between PC and target Zynq file system
  - Explore target root file system

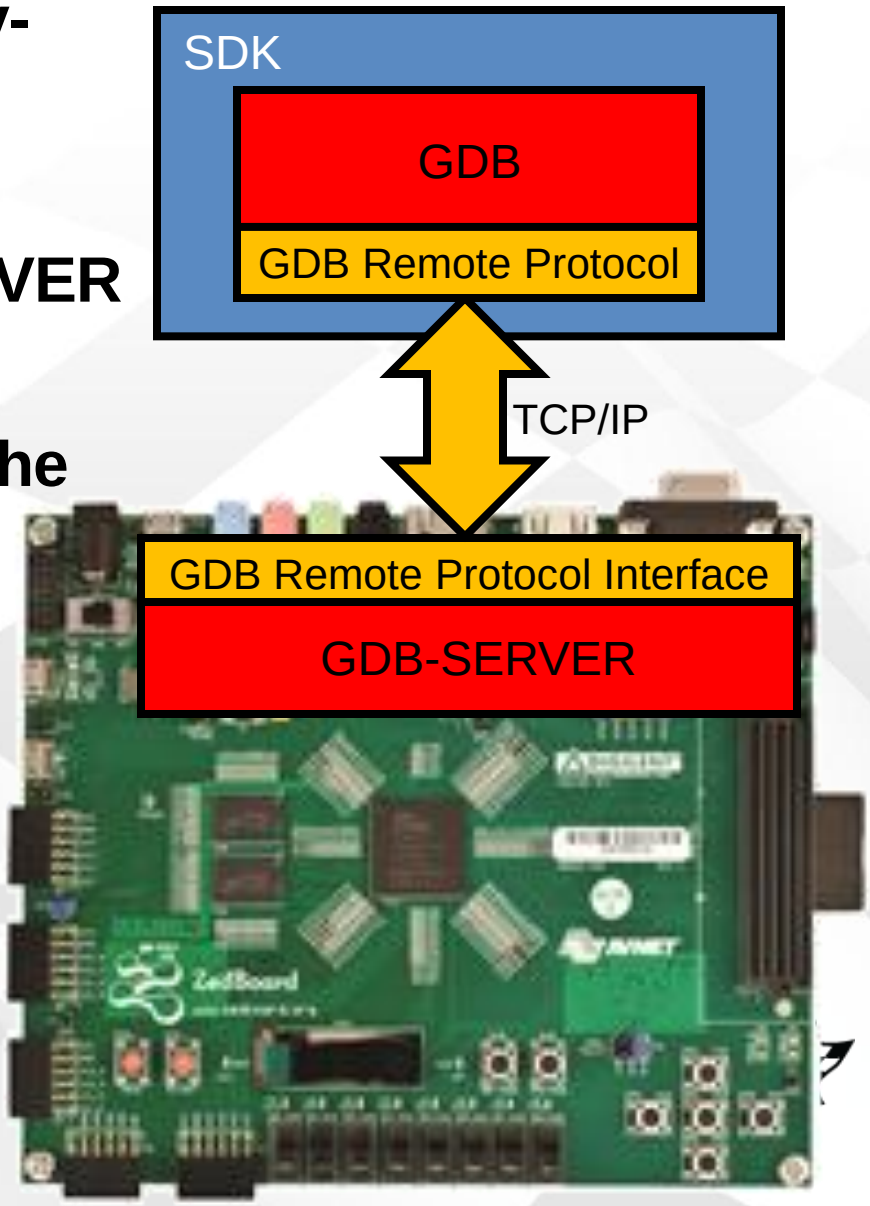


Lab 4.1

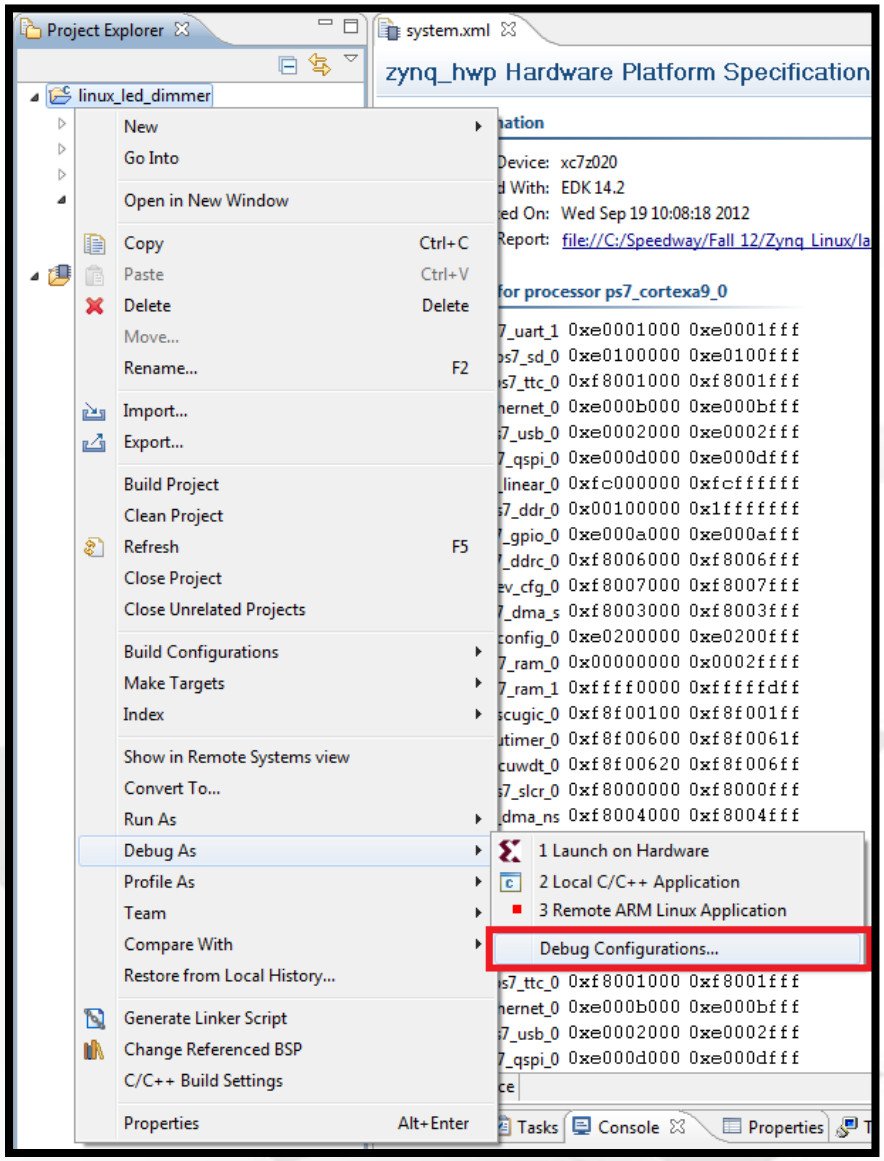
- **Linux application development with SDK**
  - Create an application which interacts with ZedBoard hardware
  - Utilize both push buttons using PS GPIO via sysfs driver
  - Utilize custom logic to control LEDs using Programmable Logic via custom device driver

Remote Application Debugging

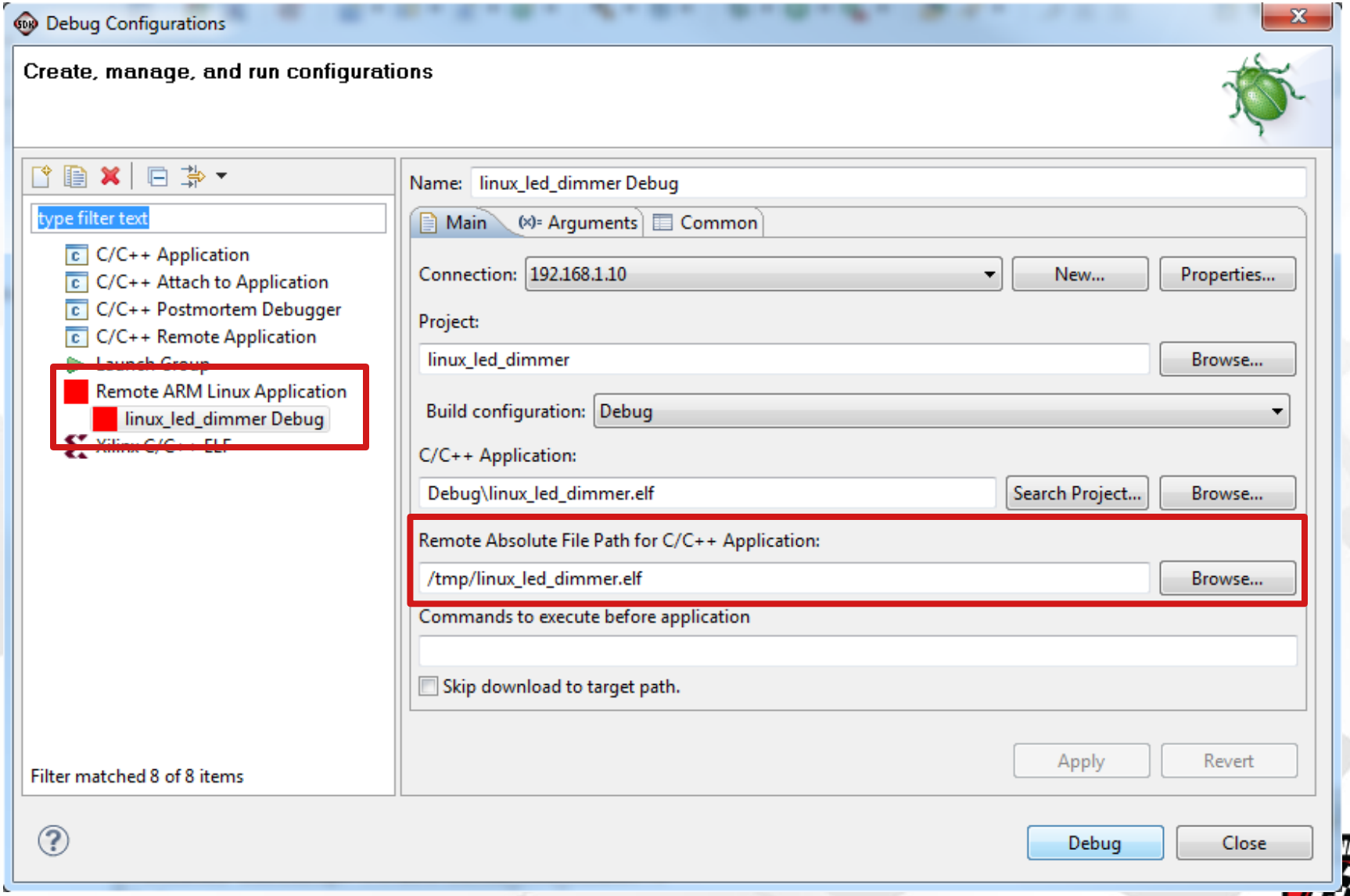
- **GDB is a source and assembly-level debugger**
- **Communicates with GDB-SERVER**
- **GDB is tightly integrated into the Eclipse Debug Perspective**



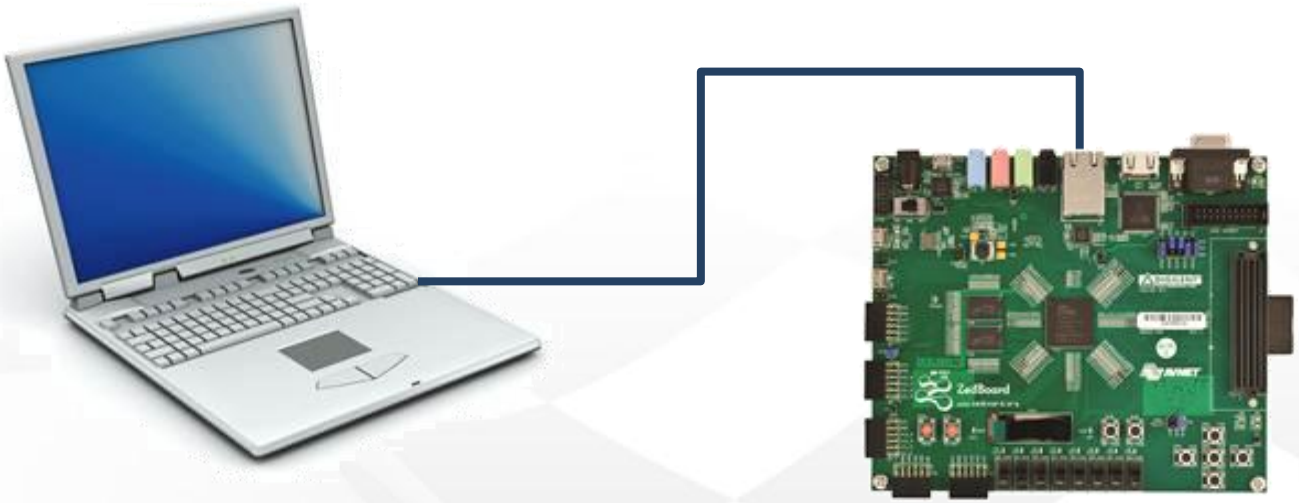
Creating A New Debug Configuration



Debug Configuration Detail



Launching A Remote Debug Session



1. Create a debug configuration to store settings for attaching to remote host
2. Remote system explorer uses SSH to control remote system
3. Remote system explorer uses SFTP to browse remote file system
4. Remote system explorer launches **`gdb-server`** automatically
5. SDK GDB attaches to remote **`gdb-server`** session for application debug

Lab 4.2

- **Linux application debug with SDK**
  - Use the SDK debugger to track down root cause for unexpected application behavior

Course Objectives Review

You now have…

- **Built critical software components of an embedded Linux system**
- **Explored Linux software development with Xilinx SDK**
- **Connected SDK to hardware for application execution and debug**
- **Learned the advantages of running Linux on Zynq**

Next Steps


SPEEDWAY
DESIGN
WORKSHOPS


AVNET
electronics marketing
Accelerating Your Success™

Where to go for additional training?

Xilinx Authorized Training Partner (ATP) Courses

- **Building Linux Platform**
  - Use of Open-Source Components
  - Environment Configurations
  - Embedded Linux Development
- **Details of Using Zynq Processor**
  - Advanced Boot Methodologies
  - Cortex-A9 Processor Services
  - Advanced DMA Controller

Embedded Open-Source Linux Design (2 Days, Level 4)
www.xilinx.com/training/atp.htm

Advanced Embedded Systems Software (1 Day, Level 4)
www.xilinx.com/training/atp.htm

Online Resources

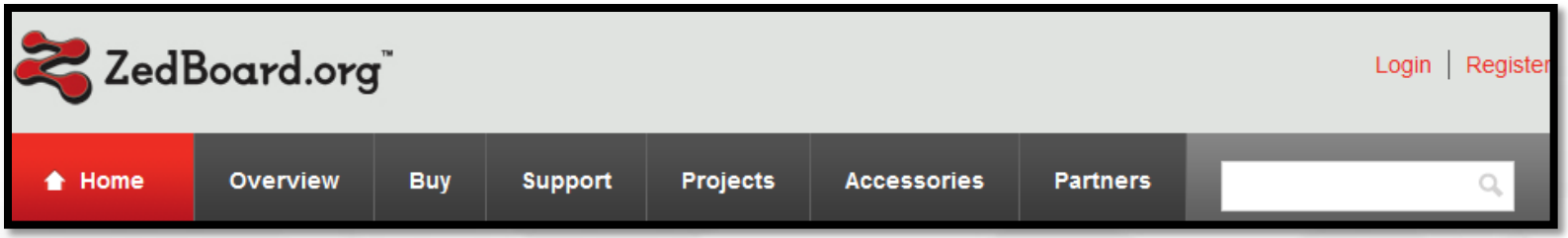
- Xilinx Zynq Linux Wiki
  wiki.xilinx.com/zynq-linux
- Xilinx Design Tools
  www.xilinx.com/products/design-tools/index.htm
- ZedBoard Community (tutorials, forums)
  www.zedboard.org
- Avnet Online OnDemand Trainings
  www.zedboard.org

AVNET electronics marketing

SPEEDWAY DESIGN WORKSHOPS

Getting Help and Support

- **Visit ZedBoard.org**
  http://www.zedboard.org



- **For Xilinx technical support, you may contact your local Avnet/Silica FAE or Xilinx Online Technical Support at www.support.xilinx.com. On this site you will also find the following resources for assistance:**
  - 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 Support for any questions regarding ZedBoard reference designs, kit hardware, or if you are interested in designing any of the kit devices into your next design.**
- http://www.em.avnet.com/techsupport

- **Contact your local Avnet/Silica FAE.**

AVNET electronics marketing

SPEEDWAY DESIGN WORKSHOPS

Thank You!

Please fill out the course survey

AVNET electronics marketing

Accelerating Your Success™

Appendix


SPEEDWAY
DESIGN
WORKSHOPS


AVNET
electronics marketing
Accelerating Your Success™

Reference

- **SDK Help: In SDK, select *Help* → *Help Contents***
- **Eclipse website**
  http://www.eclipse.org/
- **Eclipse CDT (C/C++ Development Tooling)**
  http://www.eclipse.org/cdt/
- **Xilinx Wiki**
  http://wiki.xilinx.com

Cross Build Platform

- **Virtual machines can run desktop Linux**
- **SpeedWay labs use VMware Player to run CentOS Workstation**

- **Latest VMware Player can be downloaded from the VMware website:**

  http://www.vmware.com/products/player/overview.html

- **CentOS can be downloaded from mirrors listed on CentOS website:**

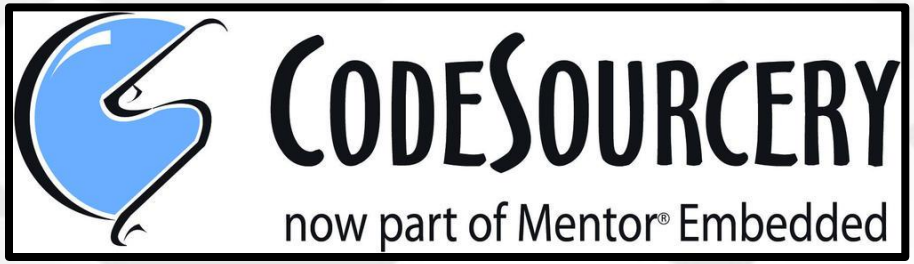
  https://www.centos.org/

Cross Build Platform

- **Virtual machines create a portable build environment where the cross toolchain can be installed**

- **Zynq cross toolchain from CodeSourcery can be downloaded from the Xilinx website:**

http://www.xilinx.com/member/mentor_codebench/xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin



- **Free and open source Git SCM can be downloaded from the Git website:**

http://git-scm.com/

Lab 0.1

- **Creating a Cross Build Platform for Zynq**

U-Boot Commands/Variables

Common U-Boot Commands:

boot	Boot using the commands in the **bootcmd** environment variable
md	Memory display, used to dump memory contents to console
mw	Memory write, used to write memory values arbitrarily
setenv	Set environment variables to indicated string
printenv	Print current setting of environment variable(s)
ping	Ping host machine using Ethernet interface
tftp	Trivial File Transfer Protocol client for moving data to/from remote host

Common U-Boot Environment Variables:

baudrate	Baudrate of the terminal connection, defaults to 115200
bootcmd	Command string executed when boot delay countdown is not interrupted
bootdelay	Delay count in seconds prior to executing **bootcmd** automatically
ethaddr	Ethernet MAC address for Ethernet interface
ipaddr	Local IP address, needed for **tftp** command
serverip	TFTP server IP address, needed for **tftp** command

Linux Command Summary

File Management

ls	Directory listing
cd	Change directory
cp	Copy file or directory
mv	Move file or directory
rm	Delete file or directory
pwd	Show current directory
tar	Create/extract .tar and tar.gz archives
chmod	Change file or directory permissions
chown	Change file or directory ownership
mkdir	Create a directory
mount	Mount a file system to a mount point
umount	Detach file system from a mount point
touch	Update access/modification time of file
df	Report file system space usage
fdisk	Modify disks and partitions
echo	Print a string or variable to console
cat	Concatenate files and print to console
sudo	Perform command as superuser
su	Change current user to superuser
whoami	Displays current username
tail -f	Monitor messages sent to log files
more	File perusal filter for terminal viewing
less	Similar to more command, but better

Networking

ifconfig	Configure network interface
ifup	Bring network interface up
ifdown	Bring network interface down
ping	Request remote host send ICMP echo

Process Control

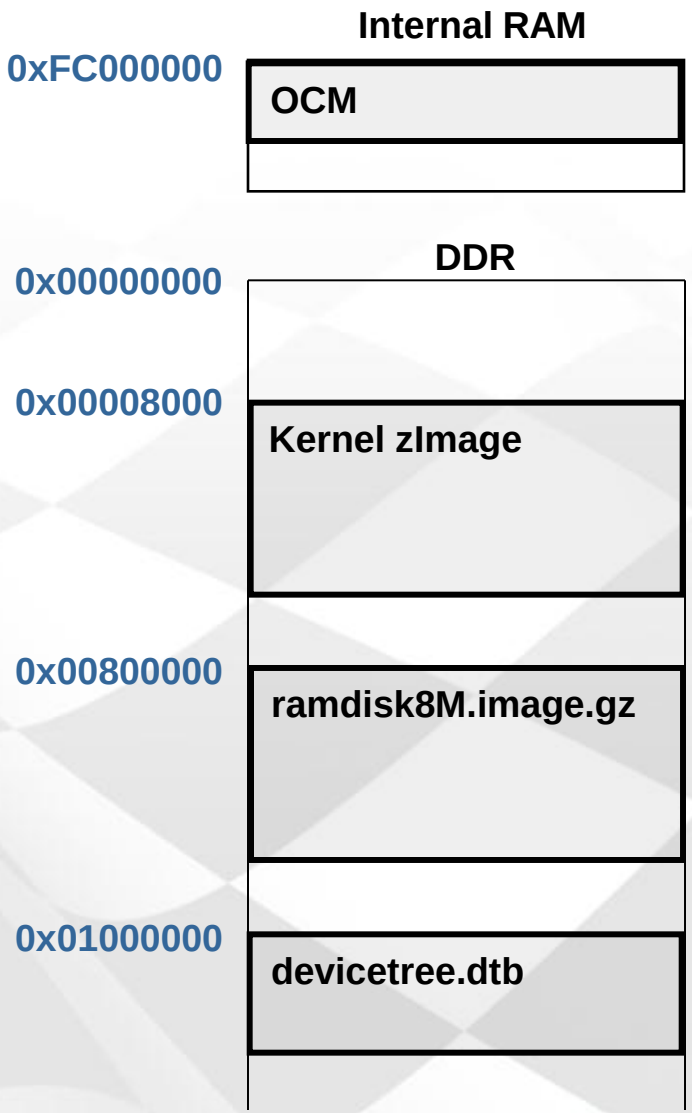
<Ctrl>-c	Terminate foreground process <SIGINT>
ps	Report snapshot of current processes
top	Display process task list
kill	Send signal to a process
renice	Alter priority of running processes

Kernel

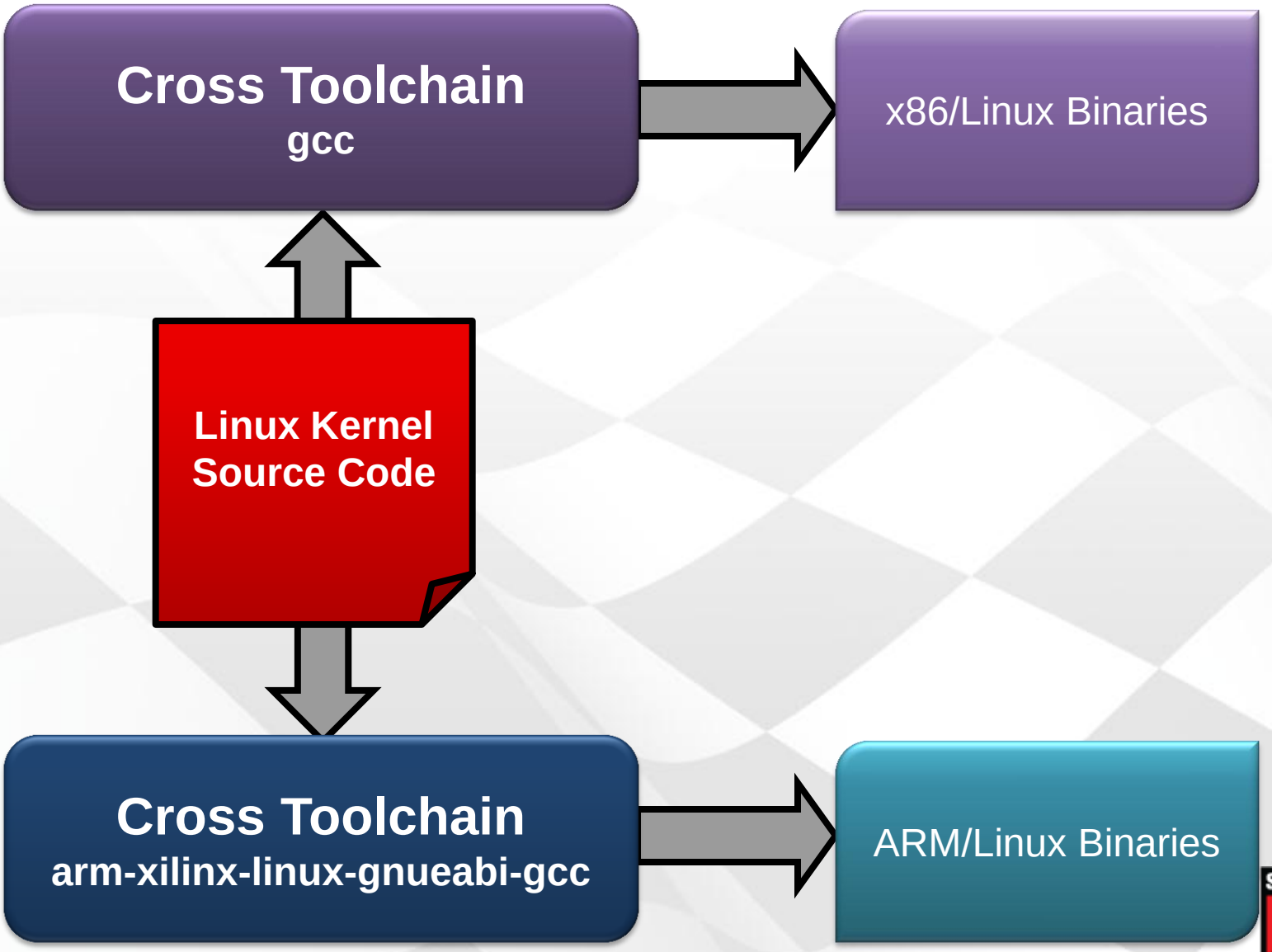
dmesg	Print the kernel ring buffer
insmod	Dynamically insert kernel module
lsmod	List status of modules in kernel
rmmod	Dynamically remove kernel module
uname -a	Show kernel version and architecture

Memory Map of Boot Process (7ynq-7000)

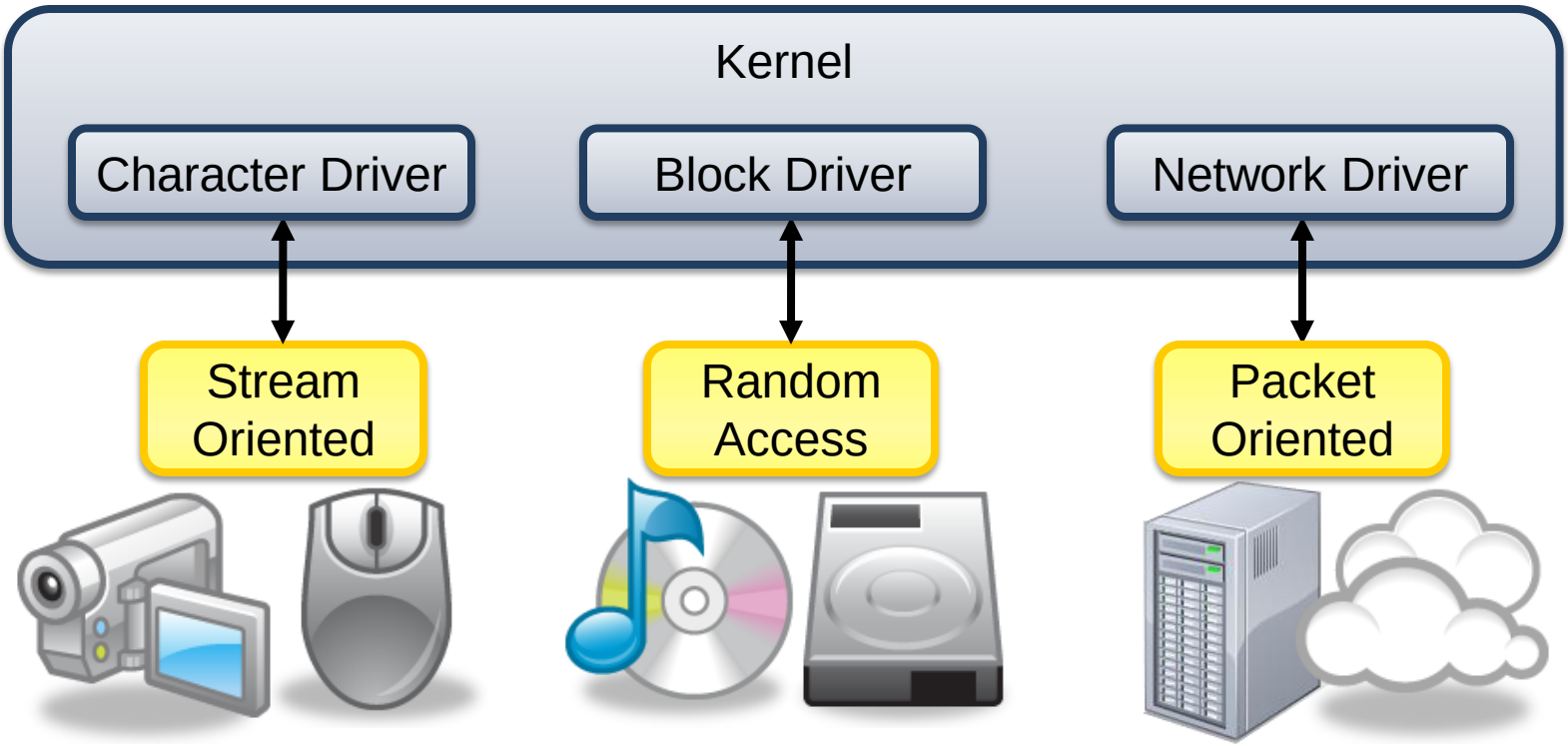
- FSBL is loaded into OCM which is remapped during handoff to U-Boot
- U-Boot is loaded first into DDR
- U-Boot then loads the zImage from boot source as directed by BOOTMODE
- U-Boot performs a checksum and then relocates (if required) the Kernel found in the zImage to the load address
- U-Boot then jumps to the entry address specified in the zImage header, linux boot starts at this point
- Note: the addresses used here are for reference only and may not apply to all devices



Native vs. Cross Compiler

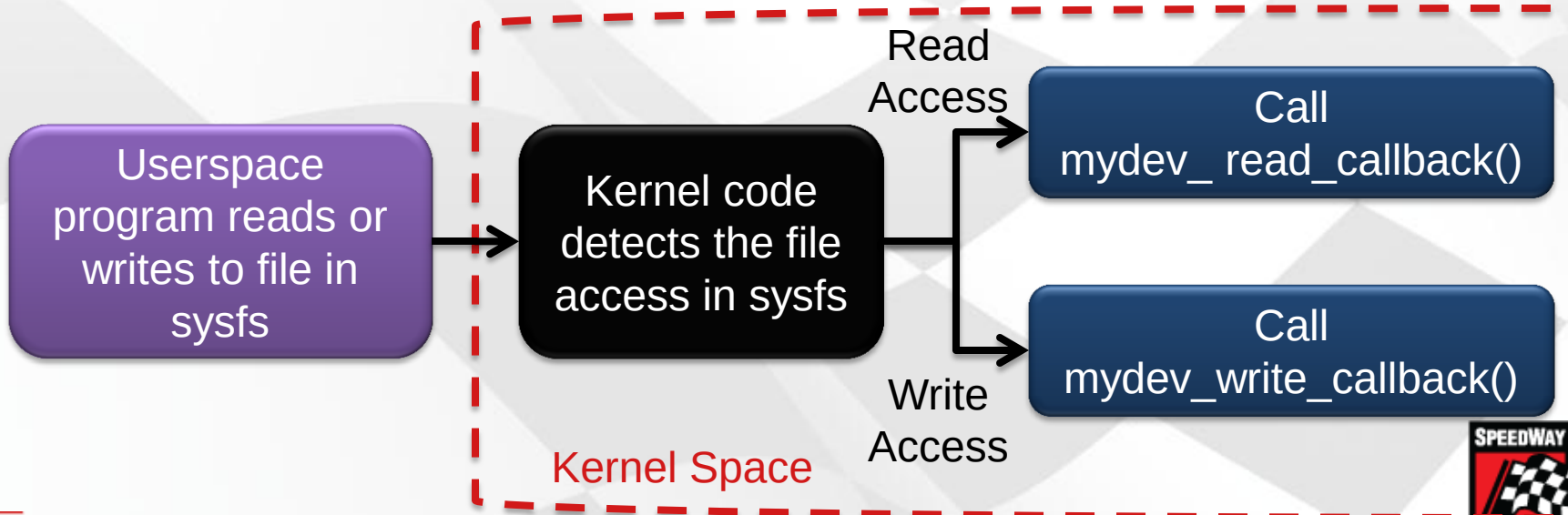


Linux Device Drivers



Linux Device Drivers

- **The sysfs device driver author exports device attributes**
- **Attributes allow user space access to driver resources using regular files in the sysfs directory structure**
- **Device driver author installs specific callback functions to handle read and write actions to the file**
  - Possibilities are endless: Trigger certain behavior, set internal driver parameters, access registers



Timesys Support

- **Expert, responsive team of Linux engineers**
  - Dedicated support engineers
  - 24-hour turn-around time
  - Unlimited number of tickets for build related issues
  - Every subscriber (seat) can submit their own tickets

- **Quick start training**
- **Self-help & education**
  - Extensive documents, videos, demos
- **Customized training – available on request**
- **Professional services available**
  - Porting/board bring-up, driver/package development
  - Fast boot, low-latency, field upgradeability

Timesys Contacts

- **contact Al Feczko at:**
  - Al.feczko@timesys.com
  - 1-866-392-4897 or (412)-325-6390
  - (412) 897-2416 (mobile)

Request a personalized, live demo and/or quote

- **contact Technical Support at:**
  - https://linuxlink.timesys.com/support