香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lecture 08: Rapid Prototyping (II) – Embedded Operating System

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Prototyping Styles with Zynq ZedBoard

ZYNQ

Xilinx SDK (C/C++)

Xilinx Vivado (HDL)

SDK (Shell, C, Java, …)

Process System (PS)

software

hardware

Program Logic (PL)

Style 1)	Style 2)	Style 3)
	Bare-metal Applications	**Applications**
		Operating System
	Board Support Package	Board Support Package
Programmable Logic Design	**Hardware Base System**	Hardware Base System
Style 1) FPGA (PL)	**Style 2) ARM + FPGA**	**Style 3) Embedded OS**
VHDL or Verilog Programming	ARM Programming & IP Block Design	Shell Script Programming

Outline

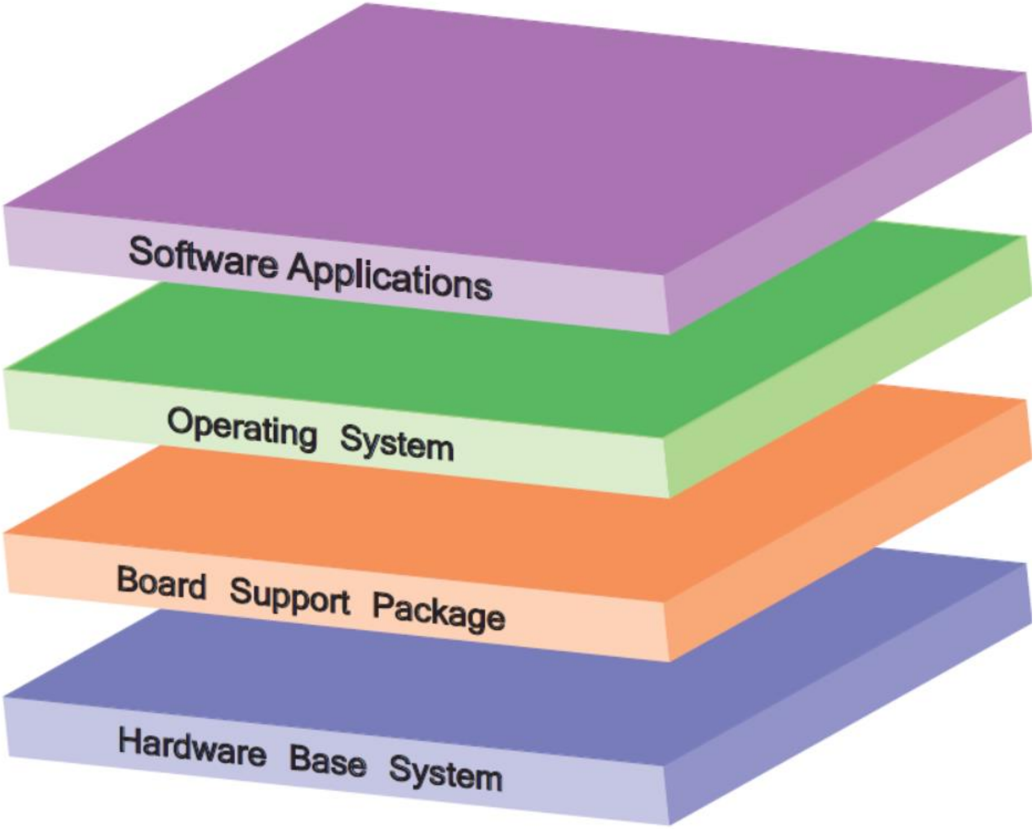
- Embedded Operating System
- Case Study: Embedded Linux
  - Linux System Overview
  - System Call Interface
  - Linux Kernel Services
  - Linux Device Driver
- Lab 08: Software Stopwatch with Zynq-Linux
  - Shell Script
  - GPIO on Zynq
  - Example Scripts

Why Embedded Operating Systems

- An embedded OS is ***not necessary*** for all digital systems, but it has the following advantages:
  - **Reducing Time to Market**
    - OS vendors provide support for various architectures and platforms.
  - **Make Use of Existing Validated Features**
    - **Graphical interface-level support** deals with the high-level graphical content that is to be displayed.
    - **Driver-level support** provides the low-level drivers that makes the connection between the processor and the device.
  - **Reduce Maintenance and Development Costs**
    - By making use of an embedded OS, the amount of custom code that needs to be developed and tested can be reduced.

Board Support Package (BSP)

- Zynq is a flexible platform.
  - The hardware base system can be customized.
  - BSP is tuned to the **hardware base system**, allowing an **OS** to operate efficiently on the given hardware.
    - It typically includes
      - bootloader (to boot the board);
      - hardware parameters;
      - device drivers;
      - low-level OS functions;
      - etc.
    - It must be refreshed if changes are made to the hardware.



Zynq Operating Systems

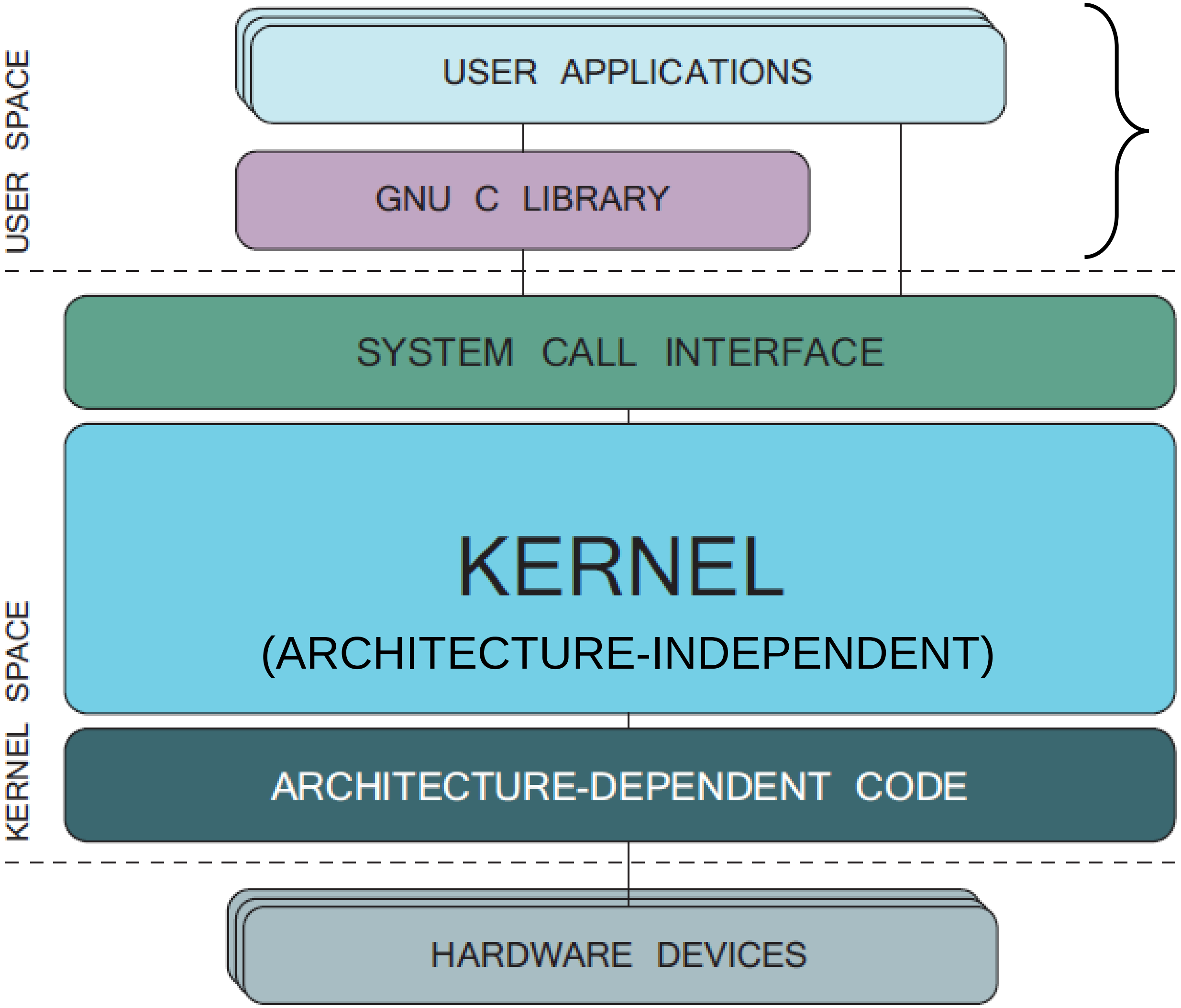
- There're many Zynq-compatible embedded OSs:
  - 🎯**Xilinx Zynq-Linux**: An open-source OS based on the Linux kernel 3.0 with additions such as board support package.
  - **Petalogix® - Petalinux**: It provides a complete package to build, test, develop and deploy embedded Linux systems.
  - **Xillybus – Xillinux**: A desktop distribution of Linux that can run a full graphical desktop environment on the Zedboard.
    - A keyboard and mouse can be attached via the USB On-The-Go port, while a monitor can be connected to the provided VGA port.
  - **FreeRTOS**: a lightweight real-time OS that is available for a wide range of devices and processor architectures.
  - **Further Operating Systems**: There are a large number of OSs for Zynq which are provided by Xilinx partners:
    - E.g., Adeneo Embedded Windows CE 7.0, Linux, Android and QNX.

Outline

- Embedded Operating System
- Case Study: Embedded Linux
  - Linux System Overview
  - System Call Interface
  - Linux Kernel Services
  - Linux Device Driver
- Lab 08: Software Stopwatch with Zynq-Linux
  - Shell Script
  - GPIO on Zynq
  - Example Scripts

Linux System Overview

- Below shows a generalized GNU/Linux system:



Applications run on top of the kernel in user-space.

SCI facilitates function calls from the user space to the kernel of system.

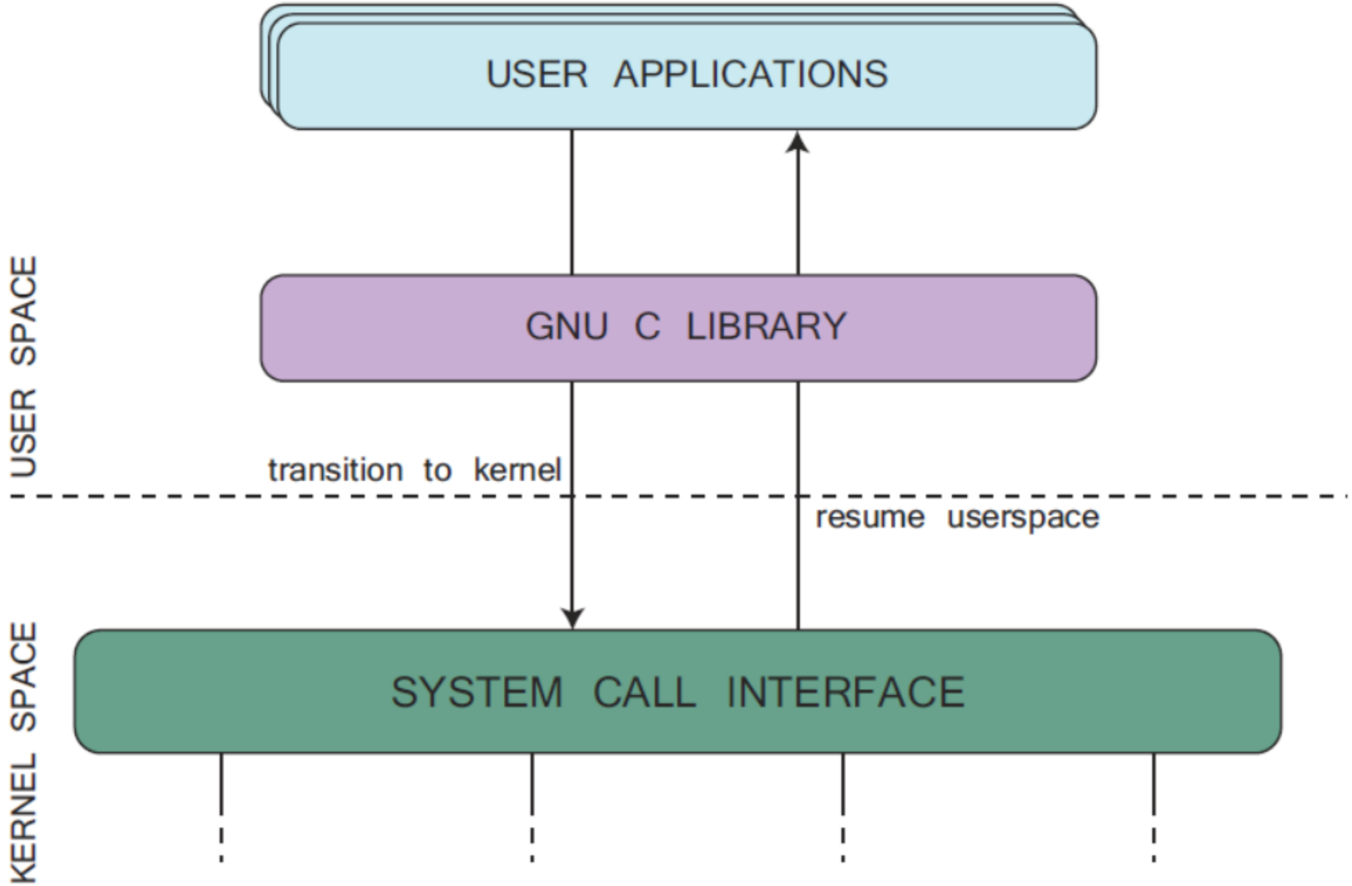
Kernel provides a set of functions with which user can interact with hardware.

It is platform-specific and is often referred as a BSP.

Hardware devices are abstracted from the user-level by the kernel space.

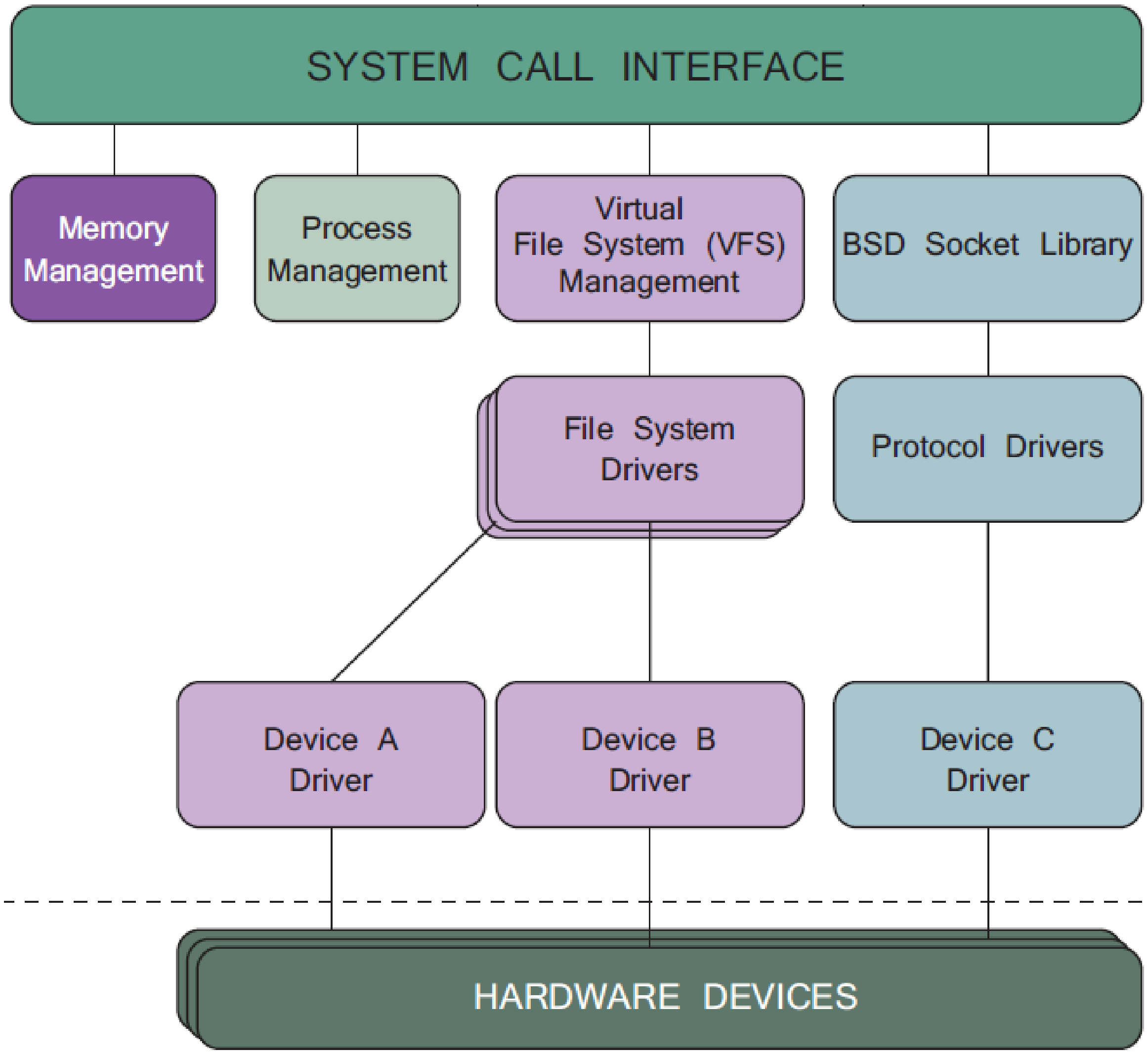
System Call Interface

- A **system call** is an interaction between user-space applications and kernel-space services.
  - Direct calls are ***NOT*** allowed for hardware isolation.



Linux Kernel Services

- **Linux kernel** is of subsystems providing key services:



Linux Device Driver

- **Linux device driver** provides an abstraction between hardware devices and running applications.
  - A standardized set of calls can be implemented across all programs which are independent of the specific device.

Table: Included Device Drivers for Zynq-Linux

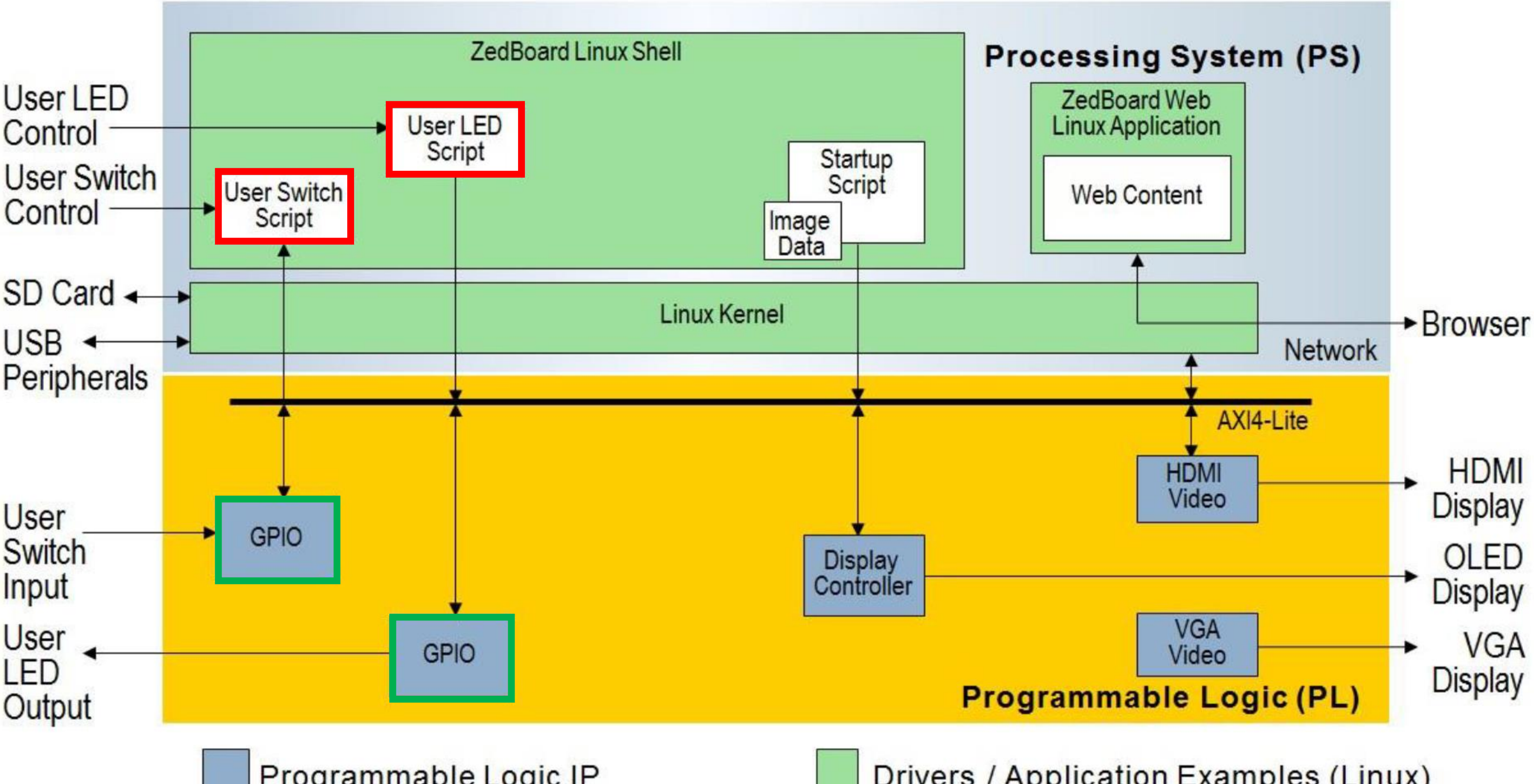
Analog-to-Digital Converter	drivers/hwmon/xilinx-xadcps.c	L2 Cache Controller (PL310)	arch/arm/mm/cache-l2x0.c
ARM global timer	drivers/clocksource/arm_global_timer.c	QSPI Flash Controller	drivers/spi/spi-xilinx-qps.c
ARM local timers	arch/arm/kernel/smp_twd.c	SD Controller	drivers/mmc/host/sdhci-of-arasan.c
CAN Controller	drivers/net/can/xilinx_can.c	SDIO WiFi	drivers/net/wireless/ath/ath6kl/sdio.c
DMA Controller (PL330)	drivers/dma/pl330.c	SPI Controller	drivers/spi/spi-xilinx-ps.c
Ethernet MAC	drivers/net/ethernet/xilinx/xilinx_emacps.c	Triple Timing Counter	drivers/clocksource/cadence_ttc.c
	drivers/net/ethernet/cadence/macb.c	UART	drivers/tty/serial/xilinx_uartps.c
GPIO	drivers/gpio/gpio-xilinxps.c	USB Host	drivers/usb/host/xusbps-dr-of.c
I2C Controller	drivers/i2c/busses/i2c-cadence.c	USB Device	drivers/usb/gadget/xilinx_usbps_udc.c
Interrupt Controller	arch/arm/common/gic.c	USB OTG	drivers/usb/otg/xilinx_usbps_otg.c

Outline

- Embedded Operating System
- Case Study: Embedded Linux
  - Linux System Overview
  - System Call Interface
  - Linux Kernel Services
  - Linux Device Driver
- Lab 08: Software Stopwatch with Zynq-Linux
  - Shell Script
  - GPIO on Zynq
  - Example Scripts

Software Stopwatch with Zynq-Linux

- In Lab 08, we will implement a software stopwatch in Zynq-Linux by using the shell script.
  - It interacts with PL peripherals via GPIO (device driver).

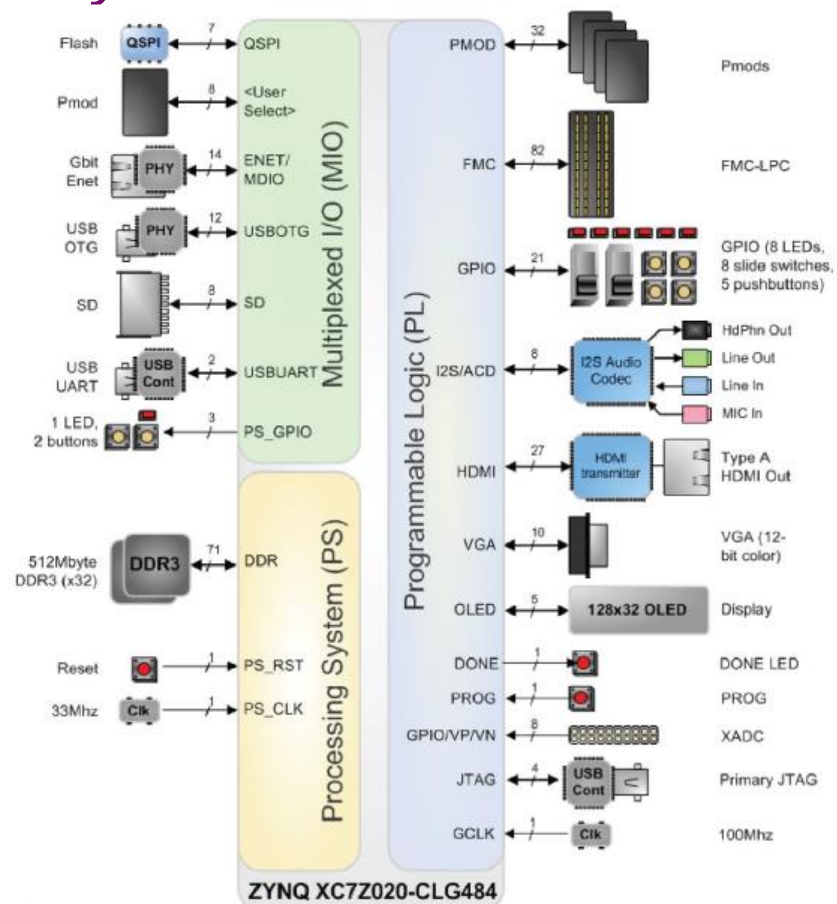


Dash Shell Script (#/bin/sh)

- A **shell script** is a list of commands that can run by the Unix shell directly in a sequential manner.
  - Unix shell is a command line (or terminal) interpreter.
- Common commands of a shell script:
  - Comment: `#` *comment*
  - Arguments: `$0, $1, $2, ...`
  - Variable: `$`*var*
  - Command Execution: `$(`*command*`)` or `` ` ``*command*`` ` ``
  - Expression: `$((`*expression*`))`
  - Loop: `for` *i* `in $(seq` *1 n*`) do ... done;`
  - Function Call: *function_name parameters;*
  - Read from File: `cat` *file_path*`;`
  - Write to File: `echo $value >` *file_path*`;`

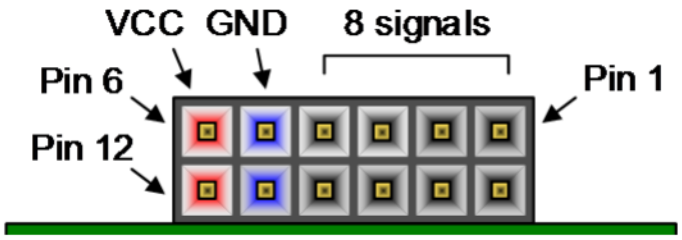
General-Purpose Input/Output (GPIO)

- **General-purpose input/output (GPIO)**:
  - *Uncommitted digital signal pins on an integrated circuit or board whose behavior—including whether it acts as input or output—is directly controllable by the user at run time.*
- There are total **118 GPIO pins** on Zynq.
  - **54 Multiplexed I/O (MIO)**: Connections to PS peripherals
    - GPIO IDs: from 0 to 53
  - **64 Extended MIO (EMIO)**: Connections to PL peripherals
    - GPIO IDs: from 54 to 117



GPIO-EMIO Pins of Zynq-Linux

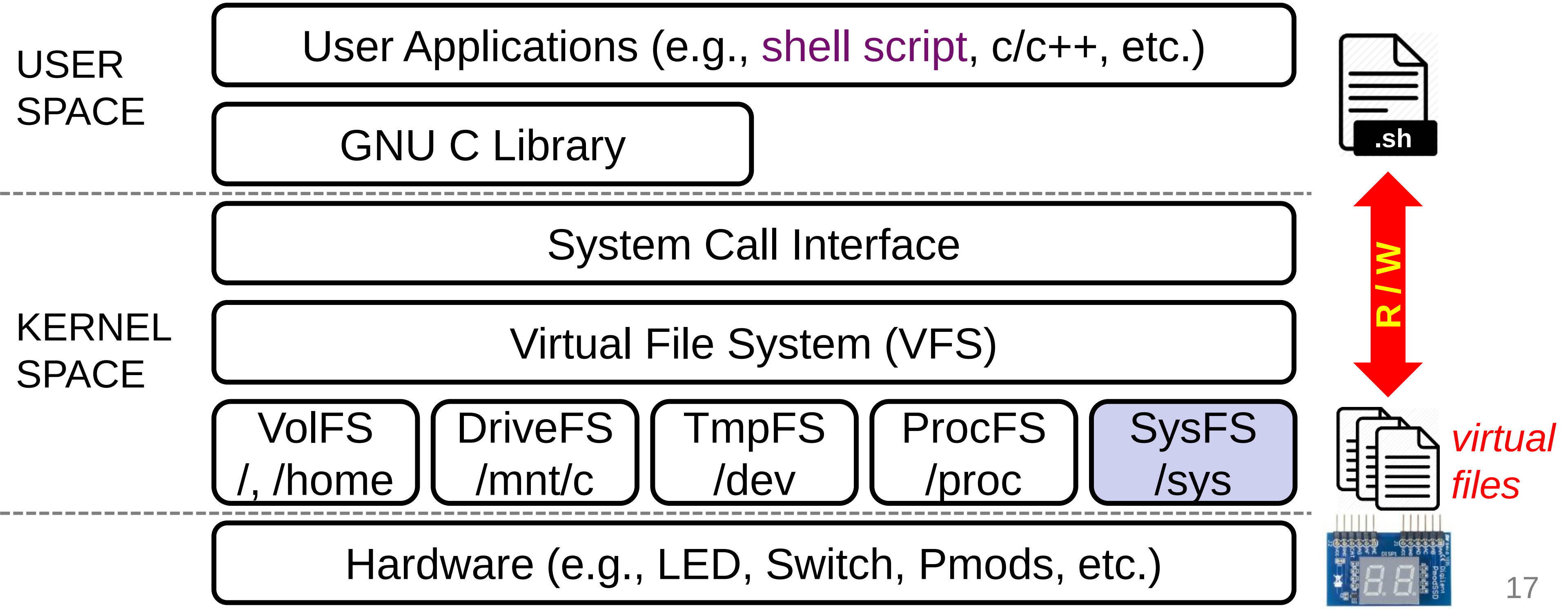
- Zynq-Linux defines **60** GPIO-EMIO signals to control the PL peripherals in `system.ucf`:
  - **USB OTG Reset**: `processing_system7_0_GPIO<0>`
  - **OLED**: `processing_system7_0_GPIO<1>~<6>`
  - **LED**: `processing_system7_0_GPIO<7>~<14>`
  - **Switches**: `processing_system7_0_GPIO<15>~<22>`
  - **Buttons**: `processing_system7_0_GPIO<23>~<27>`
  - **Pmod (JA~JD)**: `processing_system7_0_GPIO<28>~<59>`
    - **JA:** `<28>~<35>`
    - **JB:** `<36>~<43>`
    - **JC:** `<44>~<51>`
    - **JD:** `<52>~<59>`



  - *Note: The actual GPIO IDs for EMIO pins should be shifted by 54, since GPIO IDs #0 to #53 are used by MIO pins .*

Accessing GPIOs as Files (1/2)

- The standard way to control GPIO in Linux is through the **`sysfs`** interface (`/sys/class/gpio`):
  - ***`sysfs`*** *is a pseudo file system provided by the Linux kernel that exports information about various kernel subsystems, hardware devices, and associated device drivers from the kernel's device model to user space through virtual files.*



Accessing GPIOs as Files (2/2)

- GPIO (`/sys/class/gpio`) can be operated by regular file operations under Linux.
  - **Export** an GPIO (from the kernel space to the user space):
    `$ echo $id > /sys/class/gpio/export`
  - **Set the direction** of an GPIO:
    `$ echo “in” > /sys/class/gpio/gpio$id/direction`
    `$ echo “out” > /sys/class/gpio/gpio$id/direction`
  - **Read the value** of an GPIO:
    `$ cat /sys/class/gpio/gpio$id/value`
  - **Change the value** of an GPIO:
    `$ echo $var > /sys/class/gpio/gpio$id/value;`
  - **Un-export** an GPIO:
    `$ echo $id > /sys/class/gpio/unexport`

Example 1) read_sw.sh

```
#!/bin/sh # “shebang” is used to mark the start of a script

value=0; # initialize a “non-type” variable named “value” with 0
for i in 0 1 2 3 4 5 6 7; # total 8 switches, GPIO IDs from 69~76
do
  sw=$((76-$i)); # i-th GPIO pin corresponding to (7-i)-th LED
  sw_tmp=`cat /sys/class/gpio/gpio$sw/value`; # read the “value” of sw
                                 via the GPIO pin by executing the “cat command”
  value=$(($value*2)); # multiply the current value by 2
                                 (i.e., left shift the value for 1 bit)
  value=$(($value+$sw_tmp)); # add the “value” of sw to the current value
done;

printf "0x%x %d\n" $value $value; # print out the final value in both
                                                   hexadecimal & decimal format
```

Example 2) write_led.sh

```
#!/bin/sh # "shebang" is used to mark the start of a script

value=$(($1)); # the "second" argument of script (e.g., write_led 0xFF)
if [ $value -ge 0 ];
then
  for i in 0 1 2 3 4 5 6 7; # total 8 LEDs, GPIO IDs from 61~68
  do
    led=$(($i+61)); # i-th GPIO pin corresponding to i-th LED

    echo $(($value & 0x01)) > /sys/class/gpio/gpio$led/value;
    # use bitwise AND operation ('&') to get the right-most bit
      and write it to the "value" of the corresponding LED via GPIO

    value=$(($value/2)); # divide the value by 2
                         (i.e., right shift the value for 1 bit)
  done;
fi;
```

Class Exercise 8.1

- Complete the shell script that lights up the 8 LEDs based on the 8 switches:

```
#!/bin/sh # "shebang" is used to mark the start of a script
for i in 0 1 2 3 4 5 6 7;
do




done;
```

Example 3) single_digit_counter.sh

```
#!/bin/sh
display() { # function display
  value=$1 # the first argument is the
             number to be shown on SSD
  echo $2 > /sys/class/gpio/gpio93/value;
  # the second argument determines which
    digit is used (GPIO ID 93 is ssdcat)
  for i in 0 1 2 3 4 5 6;
  do
    pin=$((92-$i)); # JB: 90~92
    if [ $i -gt 2 ];
    then
      pin=$(($pin-4)); # JA:82~85
    fi;
   echo $(($value&0x01)) >
   /sys/class/gpio/gpio$pin/value;
   # write to the corresponding segment
  value=$(($value/2));
  done;
}
```

```
# define seven-segment display patterns,
representing in decimal values
ssd_0=126;
ssd_1=48;
ssd_2=109;
...
ssd_15=71;
```

Digit	Segments	Value (ssd)
0	A B C D E F	"1111110"
1	B C	"0110000"
2	A B D E G	"1101101"
3	A B C D G	"1111001"

```
# count down from 15 to 0 at 1 Hz
for i in $(seq 0 15);
do
  num=$((15-$i)); # number to be shown

  display $((ssd_$num)) 0;
  # invoke the display function:
    1st argument is the pattern of num,
    2nd argument is the ssdcat for
    selecting the left/right digit

  sleep 1; # delay one sec (1 Hz)
done;
```

Class Exercise 8.2

- Modify the shell script to make it count from 0 to 15 on the left digit of the Pmod SSD at 2 Hz:

```
#!/bin/sh # "shebang" is used to mark the start of a script
# function display
display() { ... }
# define seven-segment display patterns, representing in decimal values
ssd_0=126;
ssd_1=48;
ssd_2=109;
...
# count from 0 to 15 at 2 Hz
for i in $(seq 0 15);
do
  num=$i;
  display $((ssd_$num)) 0;
  sleep 1;
done;
```

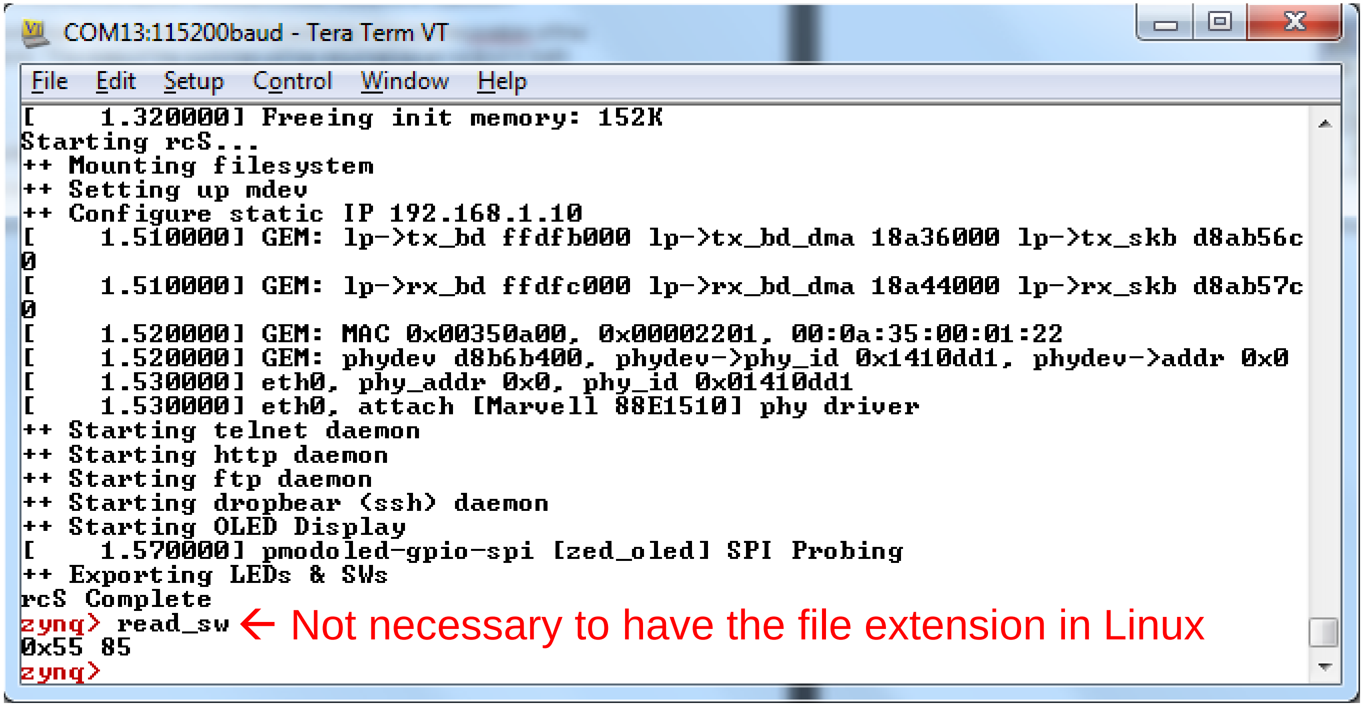
How to Run .sh Files?

- Give execute permission to your script:

```
zynq> chmod +x /path/to/yourscript.sh
```

- Run your script (“.” refers to current directory):

```
zynq> /path/to/yourscript.sh
zynq> ./yourscript.sh
```



Summary

- Embedded Operating System
- Case Study: Embedded Linux
  - Linux System Overview
  - System Call Interface
  - Linux Kernel Services
  - Linux Device Driver
- Lab 08: Software Stopwatch with Zynq-Linux
  - Shell Script
  - GPIO on Zynq
  - Example Scripts

What Else Can We Do? (1/2)

- We've learnt how to control GPIO-based Pmod.
  - How about SPI-based or I2C-based Pmod?

Analog-to-Digital Converter	drivers/hwmon/xilinx-xadcps.c	L2 Cache Controller (PL310)	arch/arm/mm/cache-l2x0.c
ARM global timer	drivers/clocksource/arm_global_timer.c	QSPI Flash Controller	drivers/spi/spi-xilinx-qps.c
ARM local timers	arch/arm/kernel/smp_twd.c	SD Controller	drivers/mmc/host/sdhci-of-arasan.c
CAN Controller	drivers/net/can/xilinx_can.c	SDIO WiFi	drivers/net/wireless/ath/ath6kl/sdio.c
DMA Controller (PL330)	drivers/dma/pl330.c	SPI Controller	drivers/spi/spi-xilinx-ps.c
Ethernet MAC	drivers/net/ethernet/xilinx/xilinx_emacps.c	Triple Timing Counter	drivers/clocksource/cadence_ttc.c
	drivers/net/ethernet/cadence/macb.c	UART	drivers/tty/serial/xilinx_uartps.c
GPIO	drivers/gpio/gpio-xilinxps.c	USB Host	drivers/usb/host/xusbps-dr-of.c
I2C Controller	drivers/i2c/busses/i2c-cadence.c	USB Device	drivers/usb/gadget/xilinx_usbps_udc.c
Interrupt Controller	arch/arm/common/gic.c	USB OTG	drivers/usb/otg/xilinx_usbps_otg.c

- We've learnt how to use the shell script to develop the application software.
  - How about other high-level languages (such as C/Python)?

What Else Can We Do? (2/2)

- How about other Zynq-compatible embedded OSs?
  - **Xilinx Zynq-Linux**: An open-source OS based on the Linux kernel 3.0 with additions such as board support package.
  - **Petalogix® - Petalinux**: It provides a complete package to build, test, develop and deploy embedded Linux systems.
  - **Xillybus – Xillinux**: A desktop distribution of Linux that can run a full graphical desktop environment on the Zedboard.
    - A keyboard and mouse can be attached via the USB On-The-Go port, while a monitor can be connected to the provided VGA port.
  - **FreeRTOS**: a lightweight real-time OS that is available for a wide range of devices and processor architectures.
  - **Further Operating Systems**: There are a large number of OSs for Zynq which are provided by Xilinx partners:
    - E.g., Adeneo Embedded Windows CE 7.0, Linux, Android and QNX.