



香港中文大學

The Chinese University of Hong Kong

*CENG3430 Rapid Prototyping of Digital Systems*

**Lecture 08:**

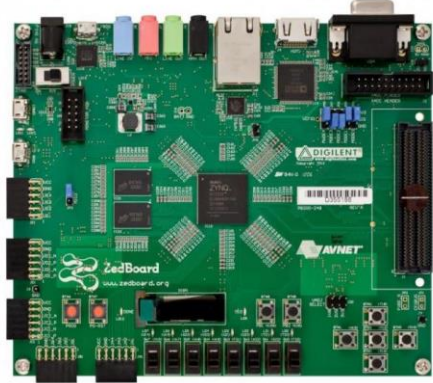
**Rapid Prototyping (II) –  
Embedded Operating System**

**Ming-Chang YANG**

[mcyang@cse.cuhk.edu.hk](mailto:mcyang@cse.cuhk.edu.hk)



# Prototyping Styles with Zynq ZedBoard



Xilinx  
SDK  
(C/C++)

**Bare-metal  
Applications**

**Applications**

**SDK**  
(Shell, C,  
Java, ...)

Operating  
System

**Process  
System  
(PS)**

Board Support  
Package

Board Support  
Package

*software*  
*hardware*

Xilinx  
Vivado  
(HDL)

**Programmable  
Logic Design**

**Hardware Base  
System**

Hardware Base  
System

**Program  
Logic  
(PL)**

**Style 1)  
FPGA (PL)**

VHDL or Verilog  
Programming

**Style 2)  
ARM + FPGA**

ARM Programming  
& IP Block Design

**Style 3)  
Embedded OS**

Shell Script &  
sysfs EMIO GPIO



- Embedded Operating System
  - Why Embedded Operating Systems
  - Types of Operating Systems
  - Zynq Operating Systems
- Case Study: Embedded Linux
  - Linux System Overview
  - Linux Kernel
  - Linux GPIO Driver: GPIO sysfs Interface
  - Shell Script
- Lab 08: Software Stopwatch with Embedded Linux

# 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**.
      - If a software is mainly developed for an OS rather than a device, it is easy to be moved to another new architecture or device.
  - **Make Use of Existing Features**
    - Embedded OSs offer support for **many validated features** which would otherwise have to be developed by the system designer.
      - **Driver-level support** provides the **low-level drivers** that makes the connection between the embedded processor and the device.
      - **Graphical interface-level** support deals with the **high-level graphical content** that is to be displayed.
  - **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**.



- 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 **BSP** and device **drivers**.
  - **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.

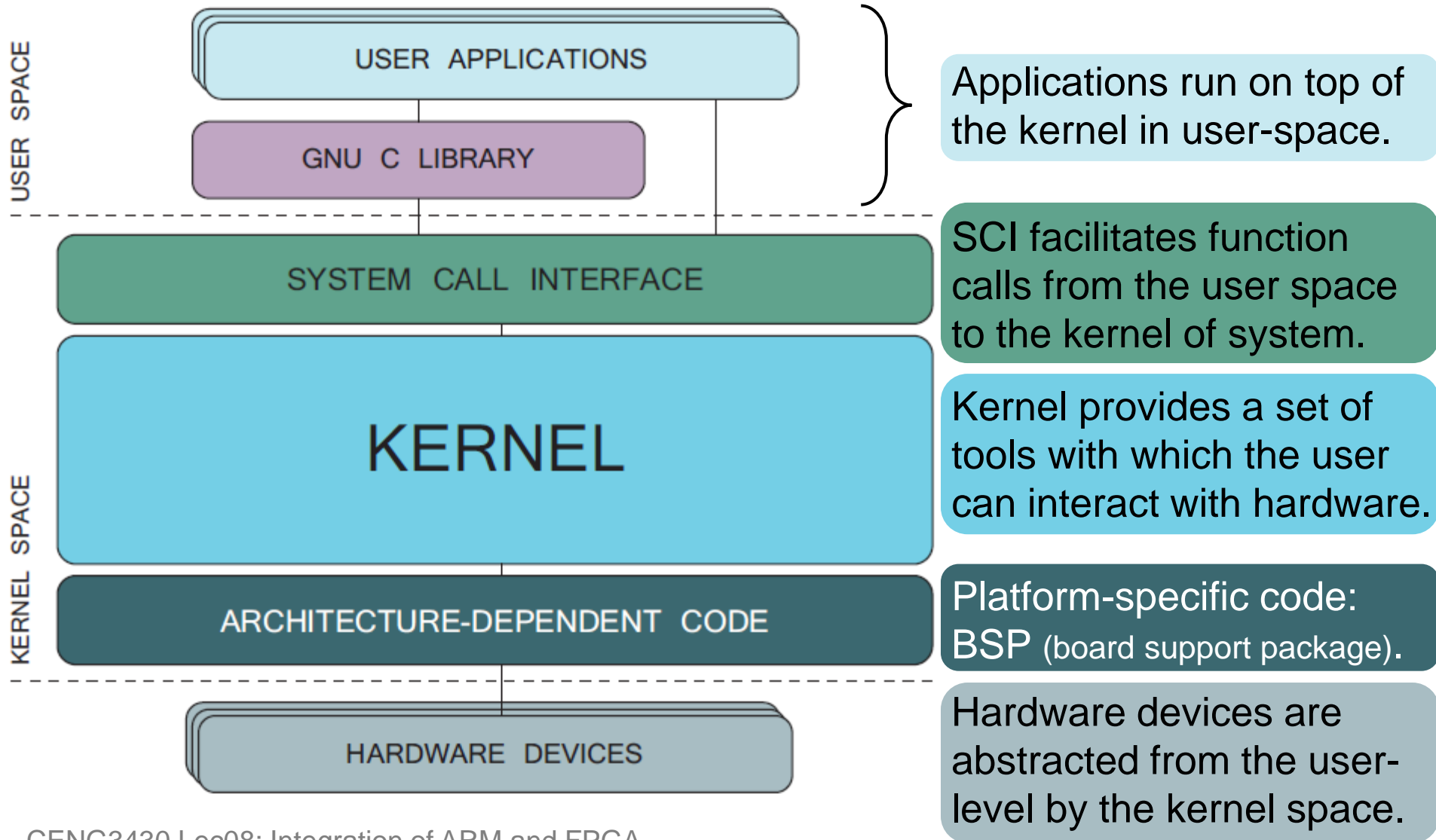


- Embedded Operating System
  - Why Embedded Operating Systems
  - Types of Operating Systems
  - Zynq Operating Systems
- Case Study: Embedded Linux
  - Linux System Overview
  - Linux Kernel
  - Linux GPIO Driver: GPIO sysfs Interface
  - Shell Script
- Lab 08: Software Stopwatch with Embedded Linux

# Linux System Overview



- Below shows a generalized GNU/Linux System:

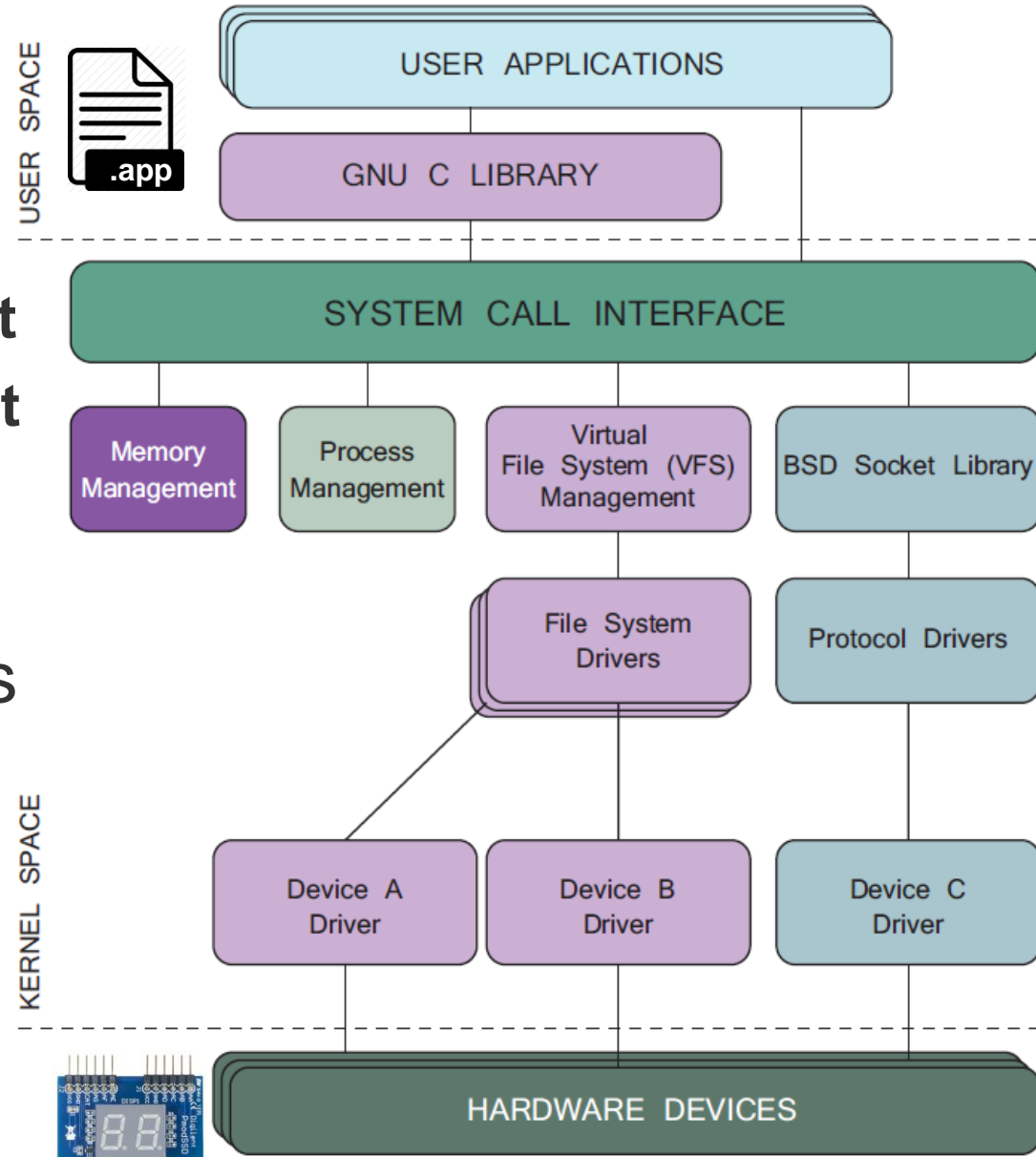




# Linux Kernel



- **Linux kernel** is of subsystems providing **required services**:
  - **Memory Management**
  - **Process Management**
  - **Virtual File System**
  - **Device Drivers**
- A **system call** provides interaction between user application and kernel services.
  - Where direct calls are **NOT** possible.

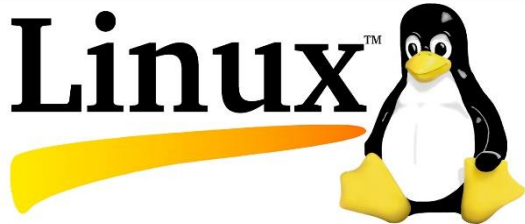




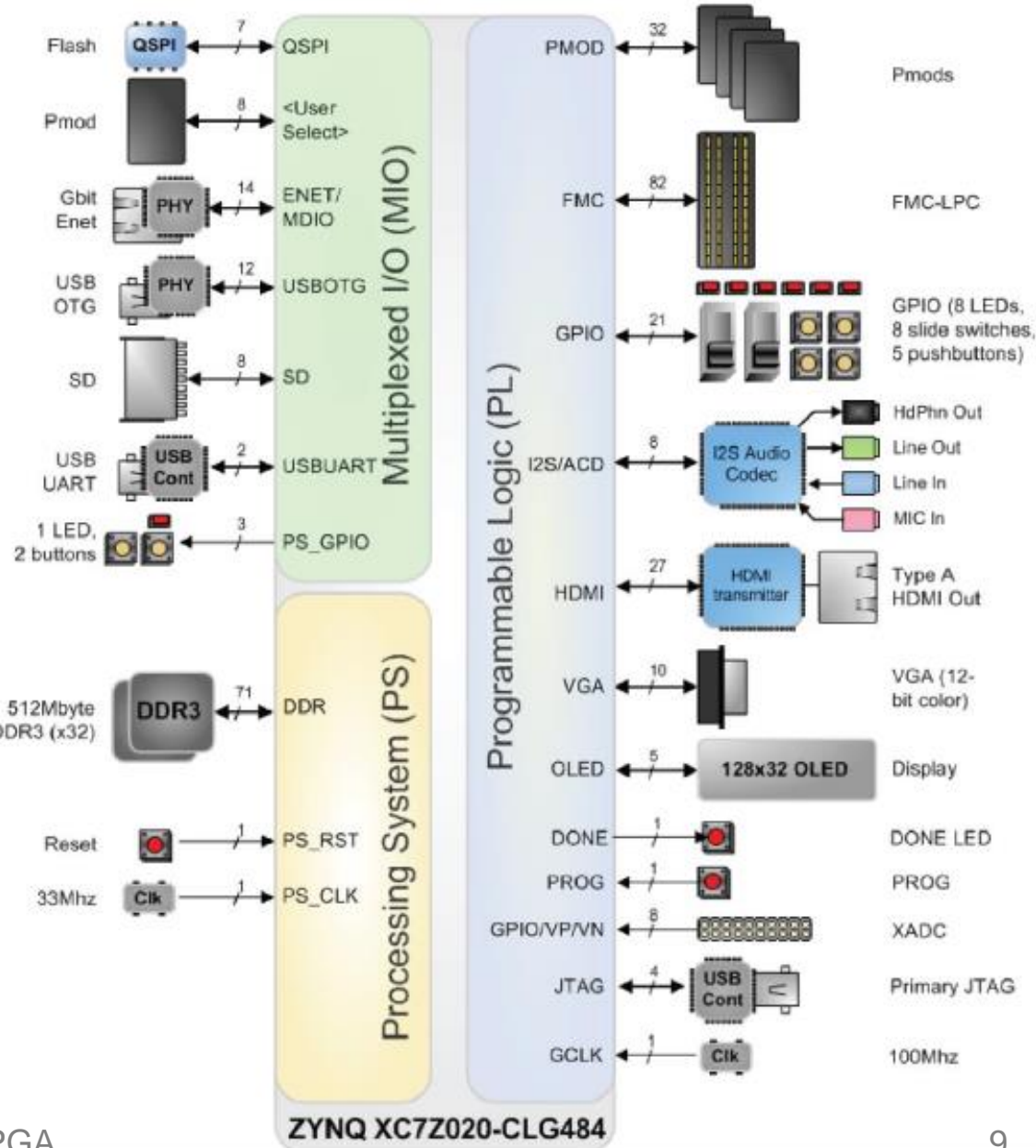
# Interactions between PS and PL



- *Question: Can we design an application to access PS or PL peripherals on top of Zynq-Linux?*
- *Answer: Yes, through the **GPIO** interface.*



The Zynq-Linux can be performed on the ARM CPU (PS) of ZedBoard.



# 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 **controllable by the user** at run time.*
- There are total **118 GPIO pins** (54 MIO pins and 64 EMIO pins) on Zynq.
  - **Multiplexed I/O (MIO):** Connections to PS peripheral ports
    - GPIO IDs: from 0 to 53
  - **Extended MIO (EMIO):** Connections to PL peripheral ports
    - GPIO IDs: from 54 to 117

# Connections to PS/PL Peripherals

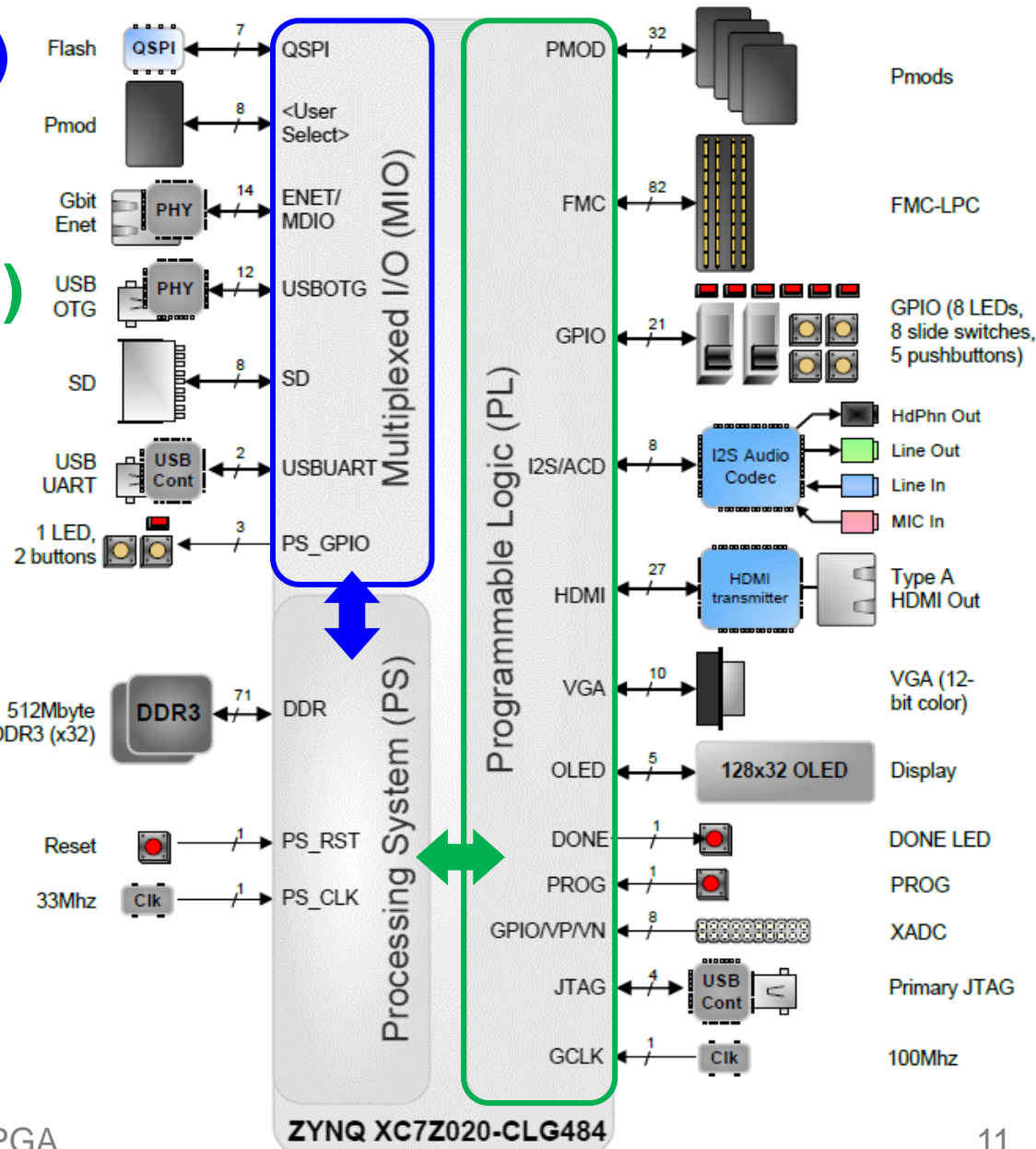
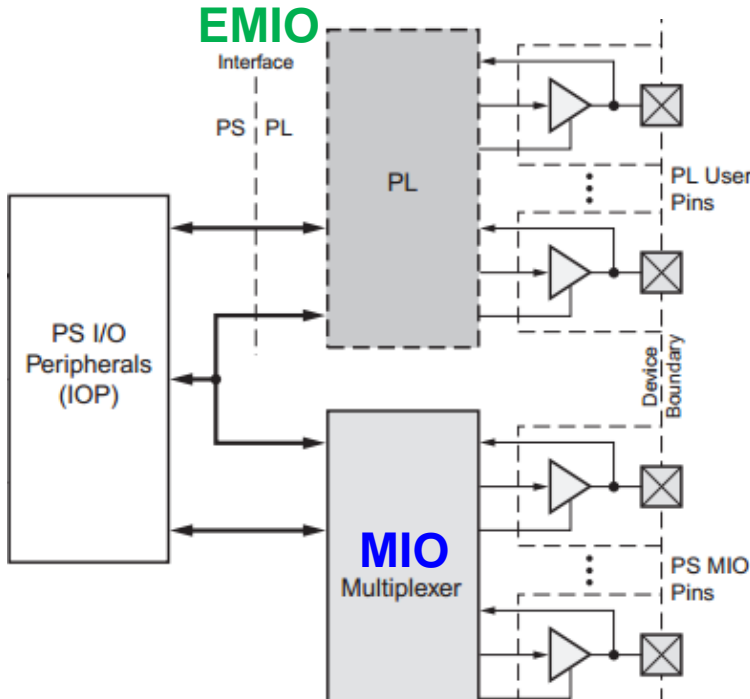


- Multiplexed I/O (MIO)**

- Connections to PS peripheral ports

- Extended MIO (EMIO)**

- Connections to PL peripheral ports



# EMIO Pins of Zynq-Linux

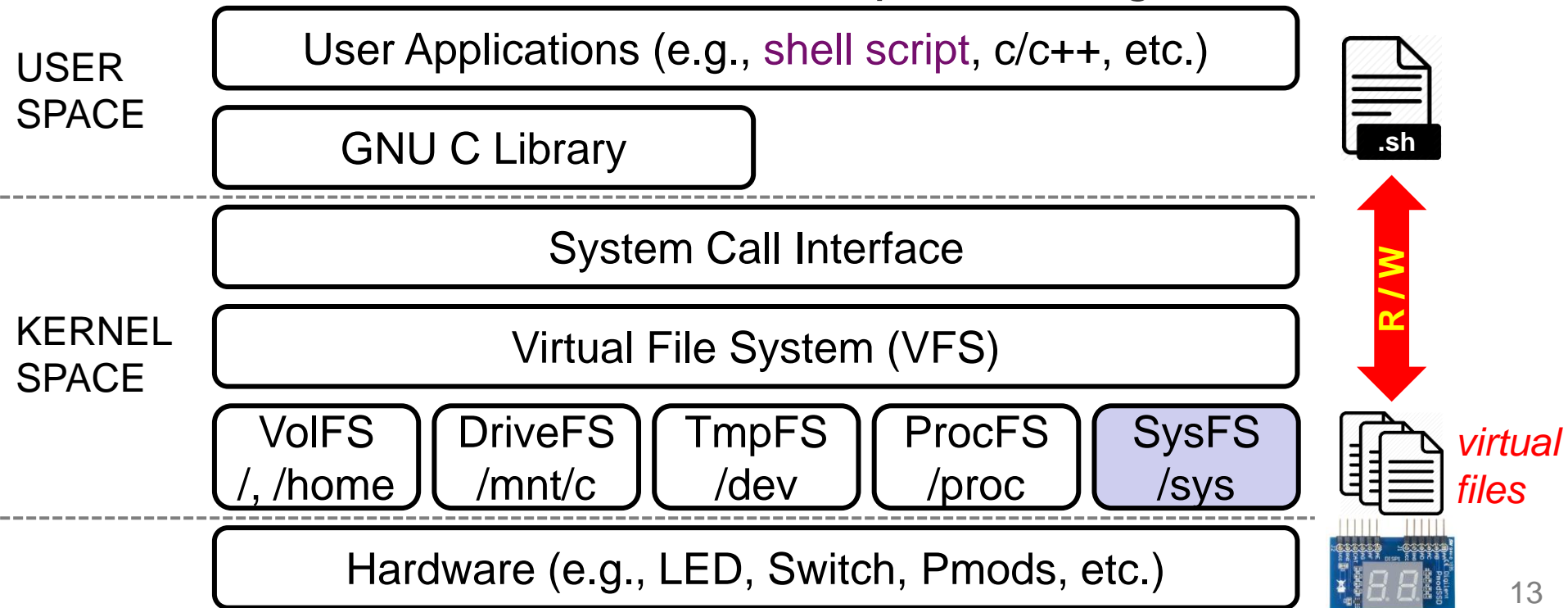


- Zynq-Linux defines **60 EMIO signals** to control the PL peripherals in `system.ucf`:
  - **USB OTG Reset:** `processing_system7_0_GPIO<0>`
  - **OLED:** `processing_system7_0_GPIO<1>~<6>`
  - **LED:** `processing_system7_0_GPIO<7>~<14>`
  - **Switches:** `processing_system7_0_GPIO<15>~<22>`
  - **Buttons:** `processing_system7_0_GPIO<23>~<27>`
  - **Pmod (JA~JD):** `processing_system7_0_GPIO<28>~<59>`
  - *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 GPIOs 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 $v > /sys/class/gpio/gpio$id/value;`
  - **Un-export** an GPIO:  
`$ echo $id > /sys/class/gpio/unexport`

# 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: `$(command)` or ``command``
  - Expression: `$(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;`



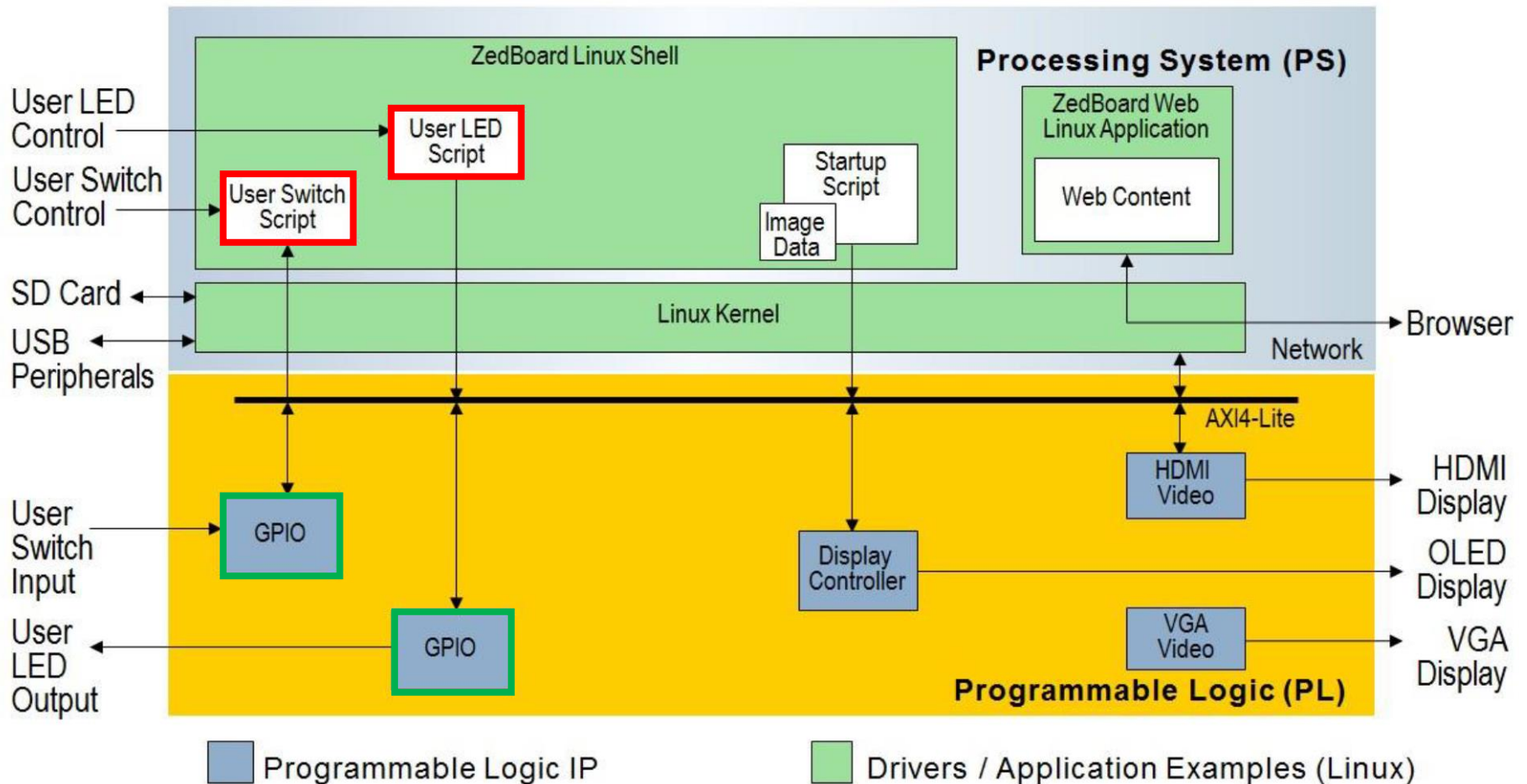


- Embedded Operating System
  - Why Embedded Operating Systems
  - Types of Operating Systems
  - Zynq Operating Systems
- Case Study: Embedded Linux
  - Linux System Overview
  - Linux Kernel
  - Linux GPIO Driver: GPIO sysfs Interface
  - Shell Script
- **Lab 08: Software Stopwatch with Embedded Linux**

# Lab 08: Linux GPIO Stopwatch



- In Lab 08, we will implement a software stopwatch, which interacts with PL peripherals via GPIOs, in Zynq-Linux by using the shell scripts.



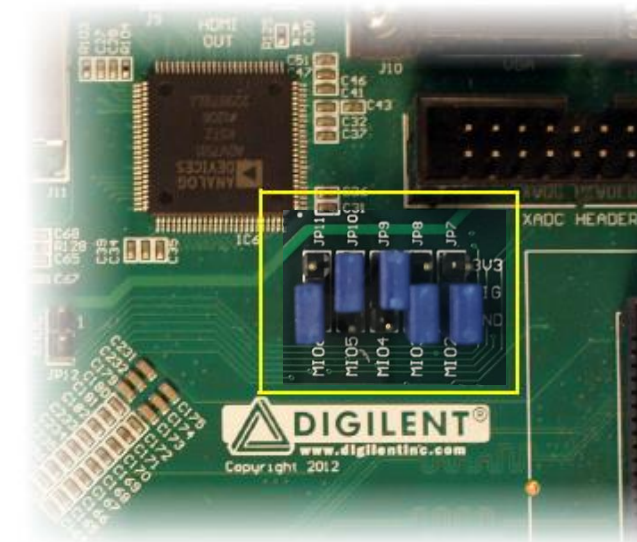
# Booting the ZedBoard from SD Card



- The ZedBoard user specifies the method of booting / programming via a set of jumper pins.
  - The middle three are for specifying programming source.

	MIO[6]	MIO[5]	MIO[4]	MIO[3]	MIO[2]
In Xilinx Technical Reference Manual...	Boot_Mode[4]	Boot_Mode[0]	Boot_Mode[2]	Boot_Mode[1]	Boot_Mode[3]
<i>JTAG Mode</i>					
Cascaded JTAG <sup>a</sup>	-	-	-	-	0
Independent JTAG	-	-	-	-	1
<i>Boot Device</i>					
JTAG	-	0	0	0	-
Quad-SPI (flash)	-	1	0	0	-
SD Card <sup>a</sup>	-	1	1	0	-
<i>PLL Mode</i>					
PLL Used <sup>a</sup>	0	-	-	-	-
PLL Bypassed	1	-	-	-	-

Cascaded: A single JTAG connection is used to interface to the debug access ports in both the PS and PL.



The PLL mode determines whether the process of configuring the device includes a phase of waiting for the PLL to lock

# Zynq Development Setup

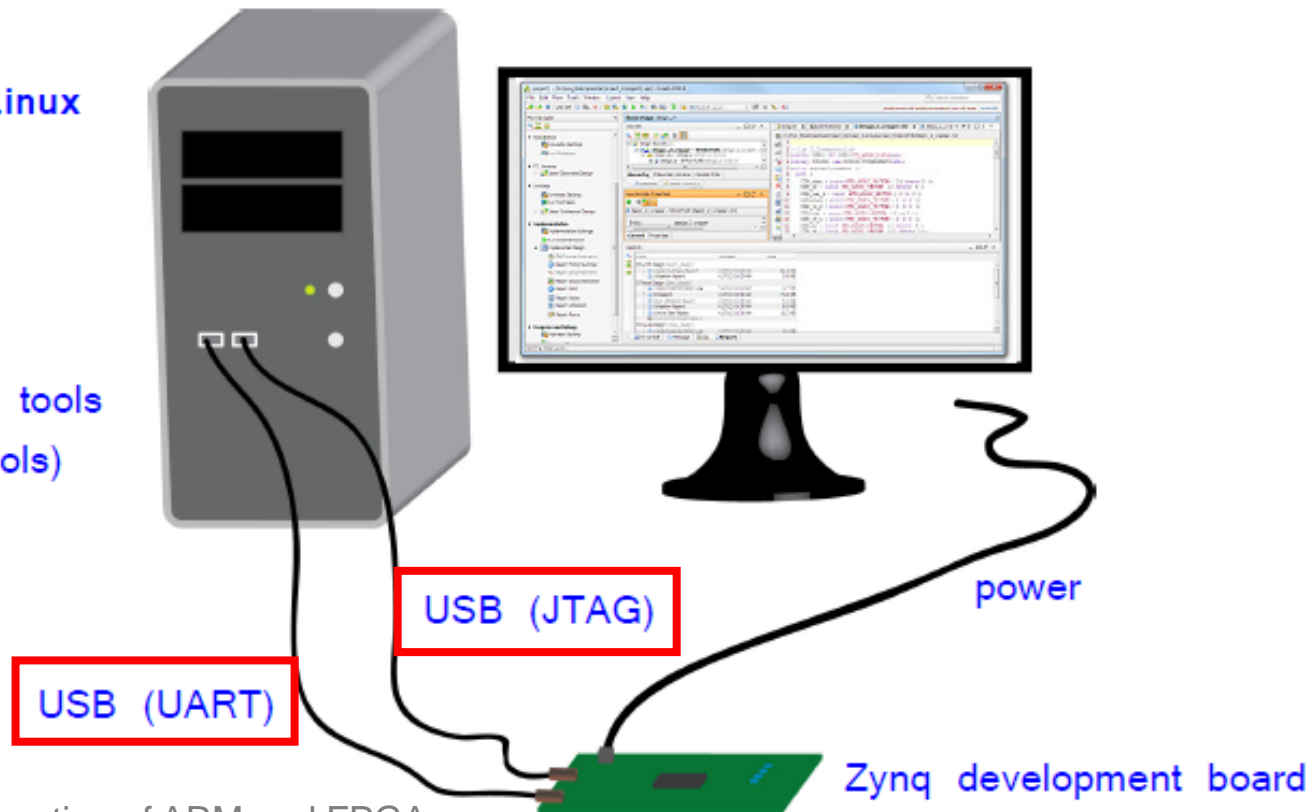


- Joint Test Action Group (JTAG): Downloading designs onto the development board over JTAG
- Universal Asynchronous Receiver/Transmitter (UART) and Terminal Applications: Interfacing and debugging

Windows / Linux  
computer

4GB+ RAM

Xilinx design tools  
(3rd party tools)



# Sample Script 1) read\_sw.sh



```
#!/bin/sh
value=0;

for i in 0 1 2 3 4 5 6 7; # total 8 switches, GPIO ID from 69~76.
do
    sw=$((76-$i));
    sw_tmp=`cat /sys/class/gpio/gpio$sw/value`; # read the value
    from the sw using corresponding gpioID

    value=$(( $value*2 )); # add the value in order, since we read the
    binary value so using 2 instead of 10 here
    value=$(( $value+$sw_tmp ));
done;

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

# Sample Script 2) write\_led.sh



```
#!/bin/sh
value=$((($1)); # arguments of the 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 bit-wise and (i.e., '&') operation to get the right-most
        bit and write to i-th GPIO pin

        value=$((($value/2)); # use divide operation to remove the
        previous right-most bit
    done;
fi;
```

# Sample Script 3) single\_count\_down.sh

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

```
# seven-segment display patterns,
representing in decimal values
p0=126;
p1=48;
p2=109;
...
p15=71;
:
for i in $(seq 0 15); # display 0~15
do
    idx=$((15-$i)); # count down the
number to be shown on the SSD

    display $((p$idx)) 0; # invoke
the display function, argument #1 is
the pattern of the i-th number,
argument #2 is the ssdcat for
selecting the left/right seven-
segment

    sleep 1; # delay one sec
done;
```

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"



# Recall: 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: `$(command)` or ``command``
  - Expression: `$( (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;`

# 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`

```
COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help
[ 1.320000] Freeing init memory: 152K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Configure static IP 192.168.1.10
[ 1.510000] GEM: lp->tx_bd ffdfb000 lp->tx_bd_dma 18a36000 lp->tx_skb d8ab56c
0
[ 1.510000] GEM: lp->rx_bd ffdfc000 lp->rx_bd_dma 18a44000 lp->rx_skb d8ab57c
0
[ 1.520000] GEM: MAC 0x00350a00, 0x00002201, 00:0a:35:00:01:22
[ 1.520000] GEM: phydev d8b6b400, phydev->phy_id 0x1410dd1, phydev->addr 0x0
[ 1.530000] eth0, phy_addr 0x0, phy_id 0x01410dd1
[ 1.530000] eth0, attach [Marvell 88E1510] phy driver
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
++ Starting OLED Display
[ 1.570000] pmodoled-gpio-spi [zed_oled] SPI Probing
++ Exporting LEDs & SWs
rcS Complete
zynq> read_sw ← Not necessary to have the file extension in Linux
0x55 85
zynq>
```



- Embedded Operating System
  - Why Embedded Operating Systems
  - Types of Operating Systems
  - Zynq Operating Systems
- Case Study: Embedded Linux
  - Linux System Overview
  - Linux Kernel
  - Linux GPIO Driver: GPIO sysfs Interface
  - Shell Script
- Lab 08: Software Stopwatch with Embedded Linux