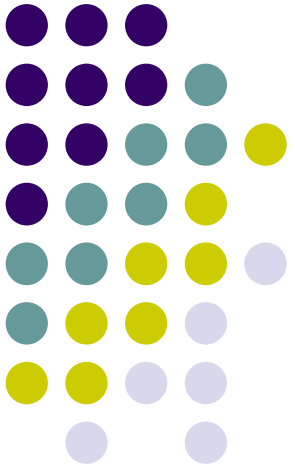
EE405 Electronic Design Lab – RoboCam

# Lab 2. Light Control

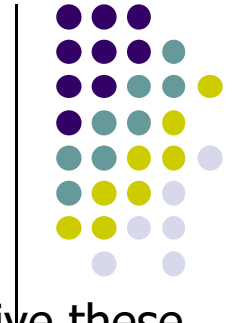


Byung Kook Kim
School of Electrical Engineering
Korea Advanced Institute of Science and Technology

# I. Purpose

- **Control LED Lights via GPIO**
  - The purpose of this lab is to control two lights (for camera illumination).
  - Hardware: hardwired LED lights using GPIO hardware.
  - Software: command lines, shell script, C program, and module .

# II. Problem Statement



- **Problem 2. Light Control.**
  - Implement an illumination light controller with two white LEDs on Beaglebone: Wire two LEDs using GPIO and transistor array, and drive these with sys file system using C.

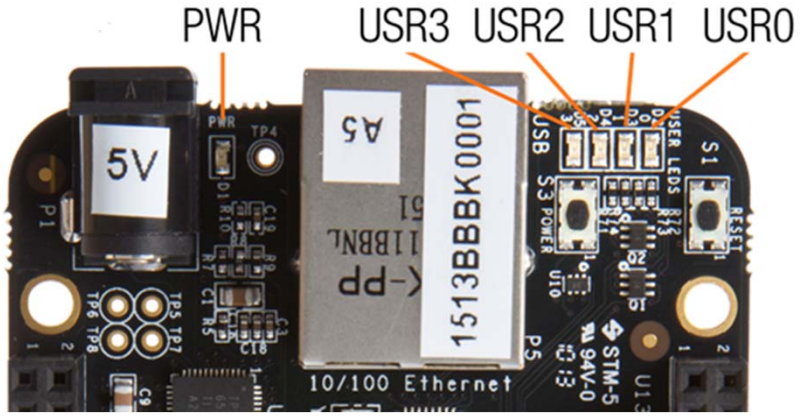
***Step-by-step improvements.***

- **Problem 2A. Light Control Command Example.**
  - Wire two LEDs using GPIO and transistor array, and control two hard-wired LED lights using sysfs and command lines.
- **Problem 2B. Light Control Shell Script.**
  - Control two hard-wired LED lights using shell script.
- **Problem 2C. Light Control C Program.**
  - Control two hard-wired LED lights using C program.
- **Problem 2D. Test Example Device Driver Program.**
  - Test an example device driver using Linux module.
- **Problem 2E. Light Control Application and Module.**
  - Control two hard-wired LED lights using application program and lights control device driver module.

# III. Technical Backgrounds

## A. User LEDs control using command line

- **1. Hardware connection summary**




### *User LED*

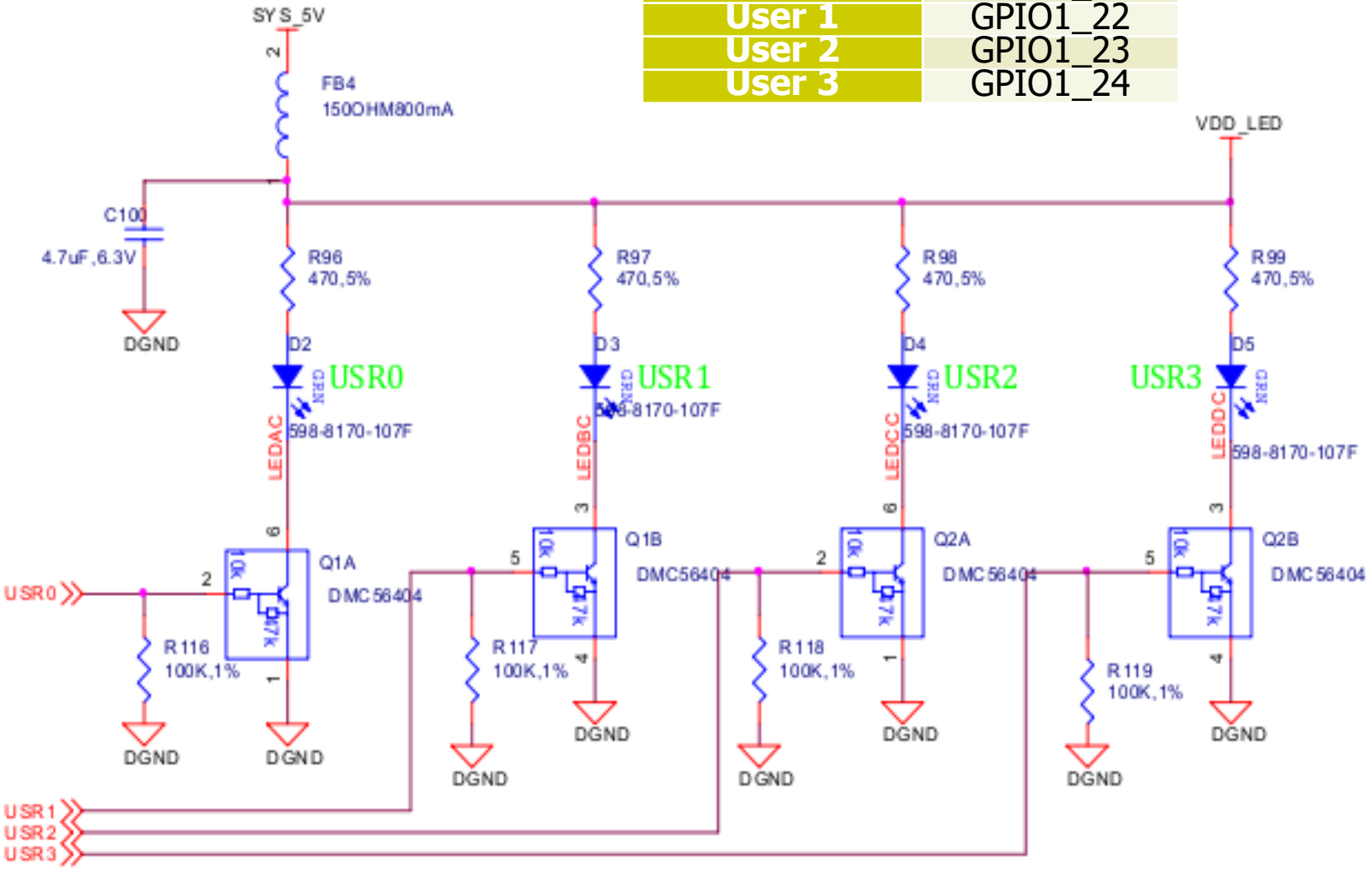
- ***AM3359 CPU → AM3359 GPIO → Bipolar Transistor → User LED with R.***

### *Custom LED*

- ***AM3359 CPU → AM3359 GPIO → P8/P9 → TR array IC → Light LED with R.***

# 2. User LEDs on Beaglebone

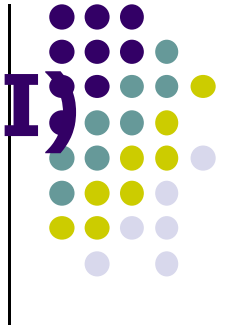
| LED | GPIO |
|---|---|
| User 0 | GPIO1_21 |
| User 1 | GPIO1_22 |
| User 2 | GPIO1_23 |
| User 3 | GPIO1_24 |



# 3. Light LED Control using Command

- gpio-sysfs is a user interface to the in kernel gpio framework.
  - It deals with pins strictly one at a time.
  - It is available in all mainline kernels from 2.6.27 onwards.
  - gpio-sysfs is the preferred method of gpio interfacing from userspace.
  - Use this unless you absolutely need to update multiple gpios simultaneously or you must use a kernel version before 2.6.27.

# Light LED Control using Command (II)

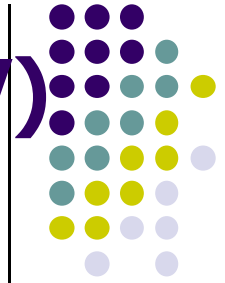


- **File structure**
  - The control files are all contained in Beaglebone file system:
    - root@beaglebone:/sys/class# ls -F /sys/class/gpio
    - export gpiochip0@ gpiochip32@ gpiochip64@ gpiochip96@ unexport
  - Initially all that is in this folder are gpiochipN folders. There is one of these for each chip which provides gpio pins. On an Beaglebone for example, you have 4 gpiochipN folders corresponding to Ports 0, 32, 64, and 96.
  - Browse the contents of gpiochip32 (GPIO32 to 63)
    - root@beaglebone:/sys/class/gpio/gpiochip32# ls -F
    - base label ngpio power/ subsystem@ uevent
    - root@beaglebone:/sys/class/gpio/gpiochip32# cat label
    - gpio
    - root@beaglebone:/sys/class/gpio/gpiochip32# cat base
    - 32
    - root@beaglebone:/sys/class/gpio/gpiochip32# cat ngpio
    - 32

# User LED Control using Command (III)

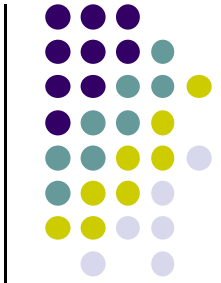
- **Getting access to a GPIO 65: "export"**
  - root@beaglebone:/sys/class/gpio/gpiochip64# echo 65 > /sys/class/gpio/export
  - root@beaglebone:/sys/class/gpio/gpiochip64# ls /sys/class/gpio
  - Export gpio65 gpiochip0 gpiochip32 gpiochip64 gpiochip96 unexport
- **Getting and Setting Direction: "direction"**
  - # echo "high" > /sys/class/gpio/gpio65/direction
  - # cat /sys/class/gpio/gpio72/direction
  - Out
- **Getting and Setting State: "echo" & "cat"**
  - To set an output high or low, write a '1' or '0' to the value attribute. To read the value, just read this file
  - # echo 1 > /sys/class/gpio/gpio65/value
  - # echo 0 > /sys/class/gpio/gpio65/value
  - # cat /sys/class/gpio/gpio65/value
  - 0

# User LED Control using Command (IV)



- Note: Different sysfs are used.
  - User LEDs
    - /sys/class/leds
  - GPIOs (and custom LEDs)
    - /sys/class/gpio

# B. Test User LED Shell Script Example



- **Shell Script**
  - The *shell* provides you with an interface to the UNIX system.
  - It reads input from you and executes the programs you specified.
  - While the programs are executing, it displays their output.
  - The real power of the shell lies in the fact that it is much more than a command interpreter.
  - It is also a powerful programming language, complete with conditional statements, loops, and functions.
- Read and test:
  - Beginners – Bash Scripting, https://help.ubuntu.com/community/Beginners/BashScripting.

```
#!/bin/bash
# nfs_server_pc.sh on PC
# Shell program on PC to start NFS Server

echo "sudo /etc/init.d/nfs-kernel-server start"
sudo /etc/init.d/nfs-kernel-server start

echo "ps -aux | grep nfs"
ps -aux | grep nfs
```

# C. Light Control Command Example

- **GPIO (General Purpose I/O)**

## AM335x Cortex™-A8 based processors

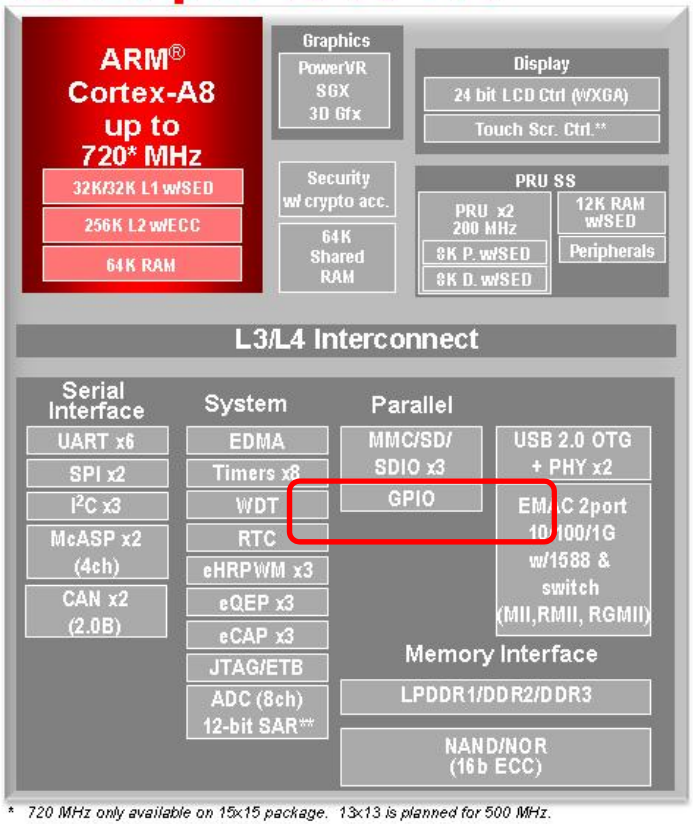
**Benefits**
- High performance Cortex-A8 at ARM9/11 prices
- PRU Subsystem for flexible, configurable communications

**Sample Applications**
- Home automation
- Home networking
- Gaming peripherals
- Consumer medical appliances
- Printers
- Building automation
- Smart toll systems
- Weighing scales
- Educational consoles
- Advanced toys
- Customer premise equipment
- Connected vending machines

**Software and development tools**
- Linux, Android, WinCE and drivers direct from TI
- StarterWare enables quick and simple programming of and migration among TI embedded processors
- RTOS (QNX, Wind River, Mentor, etc) from partners
- Full featured and low cost development board options

**Schedule and packaging**
- Samples: Today
- Dev. Tools: Order open
- Packaging: 13x13, 0.65mm via channel array
  15x15, 0.8mm

*Availability of some features, derivatives, or packages may be delayed from initial silicon availability*
*Peripheral limitations may apply among different packages*
*Some features may require third party support*
*All speeds shown are for commercial temperature range only*



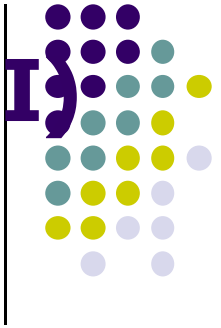

** 720 MHz only available on 15x15 package. 13x13 is planned for 500 MHz.*
*** Use of TSC will limit available ADC channels.*
*SED: single error detection/parity*

Texas Instruments

# Light Control Command Example (II)

- **Purpose of GPIO peripheral in AM335x processor**
  - The general-purpose interface combines four general-purpose input/output (GPIO) modules. Each GPIO module provides 32 dedicated general-purpose pins with input and output capabilities; thus, the general-purpose interface supports up to 128 (4 × 32) pins. These pins can be configured for the following applications:
  - Data input (capture)/output (drive)
  - Keyboard interface with a debounce cell
  - Interrupt generation in active mode upon the detection of external events. Detected events are processed by two parallel independent interrupt-generation submodules to support biprocessor operations.
  - Wake-up request generation (in Idle mode) upon the detection of signal transition(s)

# Light Control Command Example (III)



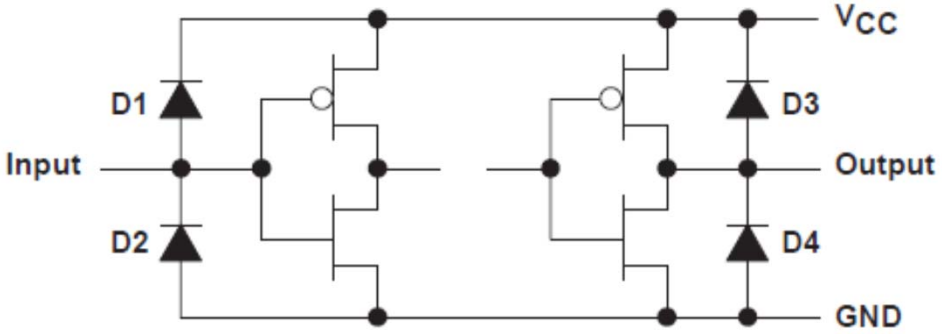
- **GPIO voltage and current**
  - The GPIO pins on the Beaglebone are ***quite fragile***
  - The Beaglebone, on the other hand, is ***decidedly intolerant of 5V.*** Hook it up to 5V, even for a split second, and it will

***die!***

# Light Control Command Example (IV)

## SAFE GPIO OUTPUT CONNECTION HOW TO

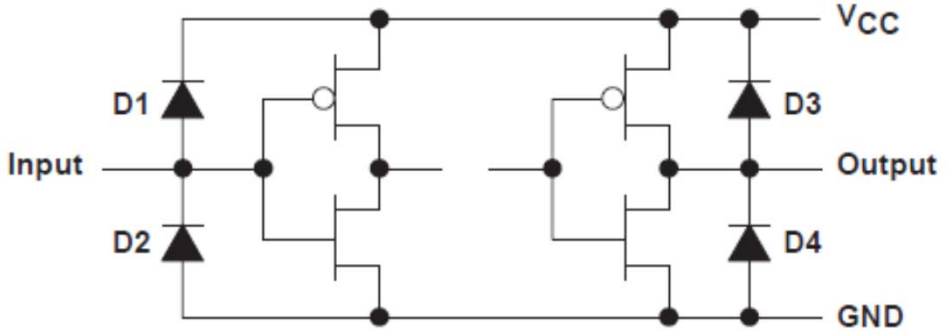
- GPIO output is 0 V or 3.3 V. We are going to limit the output current to be LESS THAN 1 mA, regardless of connected logic circuit. The connected logic circuit may be operated by +5 V power. In this case,
  - GPIO Logic 0 output of 0 V → Connected logic 0 V. Regarded as logic 0.
  - GPIO logic 1 output of 3.3 V → Connected logic 3.3 V. Regarded as logic 1 since 3.3 V > threshold (half of 5 V).
- We attach resistor of 5 Kohm between GPIO output and external logic input. In this case, the maximum current is 3.3 V / 5 Kohm = 0.66 mA.
- Hence the solution is
  - ***GPIO output → 5 Kohm resistor → External 5 V logic input.***



# Light Control Command Example (IV)

## SAFE GPIO INPUT CONNECTION HOW TO

- External logic may use +5 V power. The external logic output (0 V/5 V) is to be connected to the GPIO input, which is NOT 5V TOLERANT.
- External logic output 0 (0 V) → GPIO input 0 V. Regarded as logic 0. OK.
- External logic output 1 (5 V) → GPIO input 5 V. **BURNS BEAGLEBONE!**
  - Why external 5 V to GPIO input burns Beaglebone? In Fig 2.3, suppose the Input (in the left side) is driven by 5 V logic output. The voltages in pins of protection diode D1 are anode voltage of 5 V and cathode voltage of 3.3 V (Vcc), and hence producing voltage drop of 1.7 V. The current through D1 became very large and it will burn out!
- How to make GPIO input 5V tolerant?
  - We attach resistor of 5 Kohm between external logic output and GPIO input. In this case, the diode D1 is forward biased with 0.7 V voltage drop with current (5 – 3.3 – 0.7) V/5 Kohm = 0.2 mA. The voltage at the Input (GPIO input pin) became Vcc + 0.7 = 4 V. This will in effect make the pin 5V-tolerant for digital I/O.
- Hence the solution is
  - ***External 5 V logic output → 5 Kohm resistor → GPIO input (Left end Input in Fig. 2.3)***



# D. Light Control Shell Script

- **Light Control Shell Script**
  - Include Light control commands to form a shell script.

***Suggested Algorithm***

```
0. Print title
1. Export: Get access permission for GPIO30 & 31.
2. Set directions of GPIO 30 & 31 as output
5. User Interface Infinite loop
    A. Get user input of light_idd and onoff_str
    B. Check user input lid. Break if < 1. Check if valid.
    C. Check valid on/off string
    D. Action for correct input

8. Set directions as input
9. UnExport: Release access permission for GPIO30 & 31.
```

# E. Light Control C Program

- **Light Control C Program**
  - Design test_light_control.c

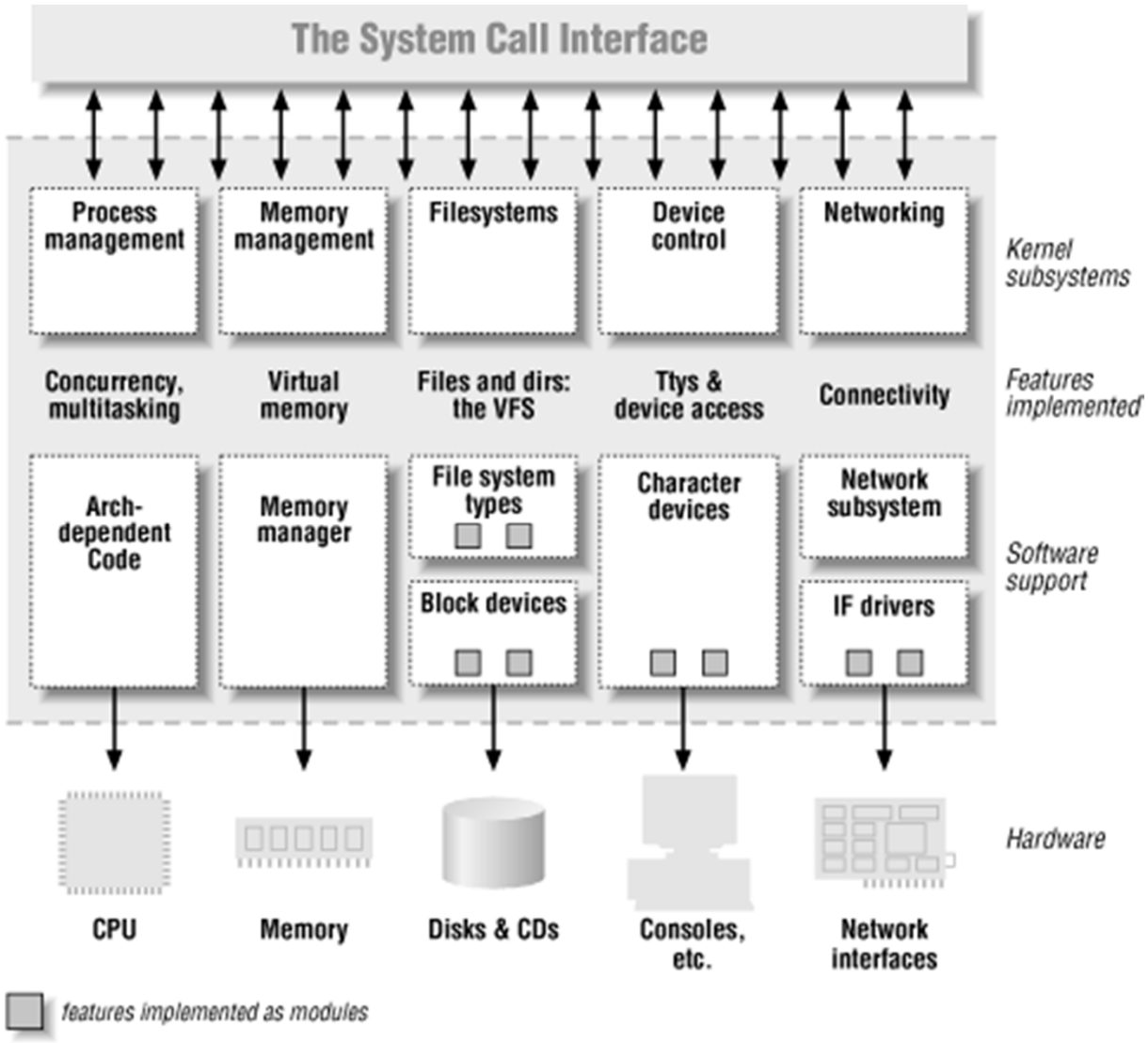
```
// File gpio_control.h
// Function definitions for gpio-control.c and app.

#define MAX_BUF 64      /* For the max length of string */

int gpio_export(unsigned int gpio);      // gpio means gpio number (0 to 127)
int gpio_unexport(unsigned int gpio);
int gpio_set_dir(unsigned int gpio, unsigned int out);  // out = 0: in. out = 1: out.

int gpio_fd_open(unsigned int gpio);     // Returns file descriptor fd
int gpio_fd_set_value(int fd, unsigned int value);     // value can be 0 or 1
int gpio_fd_get_value(int fd, unsigned int *value);    // *value will be 0 or 1
int gpio_fd_close(int fd);
```

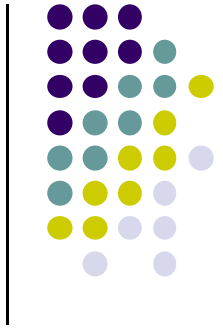
# F. Device Driver Module



# G. Light Control Device Driver Module

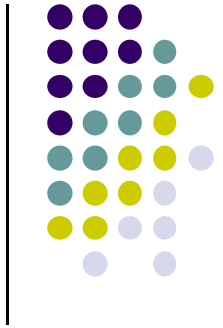
- App
  - $./test_control_LightLEDs
  - Open(), write(), close()
- Light control device driver module
  - $ insmod LightLEDs_driver_module.ko
  - LightLEDs_open()
  - LightLEDs_write()
    - Control GPCR and GPSR or
    - Write to GPDR to control LightLEDs
    - Do not affect other GPIO bits!
  - LightLEDs_close()
  - $ rmmod LightLEDs_driver_module[.ko]
- Template program is provided.

# IV. Electronic parts



| ID | Part No | Description | Qty/ group |
|---|---|---|---|
| 21 | CP41B-WES-CK0P0154 | DIP LED White, 30 mA | 2 |
| 22 | uLN2803AN | Darlington TR array, 18 pin DIP | 1 |

# V. Preparation



## Pre-report of first week

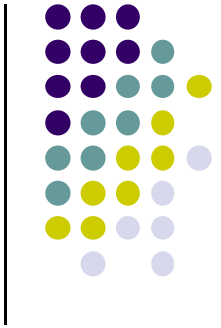
- 1. Understand and summarize the function of the GPIO in Beaglebone.
- 2. Design a hardware circuit for the Light control.

***Custom LED***

- ***AM3359 CPU → AM3359 GPIO → P8/P9***
  ***→ Resistor R1 of 5 Kohm → TR array IC***
  ***→ Resistor R2 → Cathode of Light LED .***
- ***Anode of Light LED → +5V.***

- Determine R2!

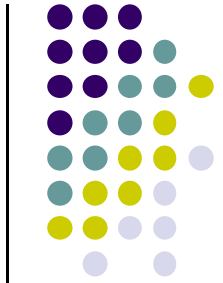
# Preparation (II)



## Pre-report of second week

- 1. Prepare Shell script for light control. You may prepare two versions.
  - Control_Lights_PC.sh for PC to test control flow on PC. GPIO outputs are changed to echo string.
  - Control_Lights_Bone.sh for Beaglebone to control actual lights on Breadboard.
- 2. Design and program to control two hard-wired LED lights using C program. Refer "Gpio-int-test.c", and modify to LightControlC.c.
- 3. Design Lights control device driver module and test application for Problem 2G.

# V. Lab Procedures



**First week**

- A. Light Control Command Example
  - Wiring on Breadboard
- B. Light Control Shell Script

**Second week**

- C. Light Control C Program
- D. Device driver module example
- E. LightLEDs driver module

# Appendx. GPIO Driver Module

## In gpio_driver.ko:

- To register the given I/O memory regions, the macro is
  - void request_mem_region (unsigned long start, unsigned long length, char *device_name);
- Physical to virtual memory mapping in Linux.
  - void* ioremap ( unsigned long *phys_addr*, unsigned long *size* )
- Set pin multiplexing
- Configure as an output (the desired bit reset in GPIO_OE),
- Clear Data Output Register (GPIO_CLEARDATAOUT):
  - A write operation in the clear data output register clears the corresponding bit in the data output register when the written bit is 1; a written bit at 0 has no effect.
- Set Data Output Register (GPIO_SETDATAOUT):
  - A write operation in the set data output register sets the corresponding bit in the data output register when the written bit is 1; a written bit at 0 has no effect.