Lab: GPIO

Objective

- Manipulate microcontroller's registers in order to access and control physical pins
- Use implemented driver to sense input signals and control LEDs
- Use FreeRTOS binary semaphore to signal between tasks

Part 0: Basic task structure to blink an LED

In this portion of the lab, you will design a basic task structure, and directly manipulate the microcontroller register to blink an on-board LED. You will not need to implement a full GPIO driver for this part. Instead, directly manipulate registers from `LPC40xx.h`

```
 void led_task(void *pvParameters) {
   // Choose one of the onboard LEDS by looking into schematics and write code for the below
   0) Set the IOCON MUX function(if required) select pins to 000
   1) Set the DIR register bit for the LED port pin
   while (true) {
     2) Set PIN register bit to 0 to turn ON LED (led may be active low)
     vTaskDelay(500);

     3) Set PIN register bit to 1 to turn OFF LED
     vTaskDelay(500);
  }
}
int main(void) {
  // Create FreeRTOS LED task
    xTaskCreate(led_task, “led”, 2048/sizeof(void*), NULL, PRIORITY_LOW, NULL);
    vTaskStartScheduler();
    return 0;}
```

Part 1: GPIO Driver

Use the following template to implement a GPIO driver composed of:

- Header file
- Source file

Implement ALL of the following methods. All methods must work as expected as described in the comments above their method name. Note that you **shall not use or reference to the existing** `gpio.h` or `gpio.c` and instead, you should build your own `gpio_lab.h` and `gpio_lab.c` as shown below.

```
// file gpio_lab.h
#pragma once
#include <stdint.h>
#include <stdbool.h>
// include this file at gpio_lab.c file
// #include "lpc40xx.h"
// NOTE: The IOCON is not part of this driver

/// Should alter the hardware registers to set the pin as input
void gpio0__set_as_input(uint8_t pin_num);
/// Should alter the hardware registers to set the pin as output
void gpio0__set_as_output(uint8_t pin_num);
/// Should alter the hardware registers to set the pin as high
void gpio0__set_high(uint8_t pin_num);
/// Should alter the hardware registers to set the pin as low
void gpio0__set_low(uint8_t pin_num);

/**
 * Should alter the hardware registers to set the pin as low
 *
 * @param {bool} high - true => set pin high, false => set pin low
 */
void gpio0__set(uint8_t pin_num, bool high);

/**
 * Should return the state of the pin (input or output, doesn't matter)
 *
```

```
 * @return {bool} level of pin high => true, low => false
 */bool gpio0__get_level(uint8_t pin_num);
```

> **Extra Credit**
> Design your driver to be able to handle multiple ports (port 1 and port 2) within a single gpio driver file
> Do this only after you have completed all of the lab

Part 2. Use GPIO driver to blink two LEDs in two tasks

This portion of the lab will help you understand the `task_parameter` that can be passed into tasks when they start to run. You will better understand that each task has its own context, and its own copies of variables even though we will use **the same function for two tasks**.

```
typedef struct {
  /* First get gpio0 driver to work only, and if you finish it
   * you can do the extra credit to also make it work for other Ports
   */
  // uint8_t port;
  uint8_t pin;
} port_pin_s;
void led_task(void *task_parameter) {
    // Type-cast the paramter that was passed from xTaskCreate()
    const port_pin_s *led = (port_pin_s*)(task_parameter);
    while(true) {
      gpio0__set_high(led->pin);
      vTaskDelay(100);

      gpio0__set_low(led->pin);
      vTaskDelay(100);
    }
}
int main(void) {
  // TODO:
  // Create two tasks using led_task() function
  // Pass each task its own parameter:
```

```
  // This is static such that these variables will be allocated in RAM and not go out of scope
  static port_pin_s led0 = {0};
  static port_pin_s led1 = {1};

  xTaskCreate(led_task, ..., &led0); /* &led0 is a task parameter going to led_task */
  xTaskCreate(led_task, ..., &led1);

  vTaskStartScheduler();
  return 0;}
```

Part 3: LED and Switch

- Design an LED task and a Switch task
  - Pass the GPIO data structure as input to the task
  - This will allow you to conveniently switch the port or pin number
  - Check this reference: https://www.freertos.org/a00125.html
- Interface the switch and LED task with a Binary Semaphore
  - Check the reference code below
  - Reference this article
  - Do not worry, we did most of the code for you
- ~~Deprecated requirements:~~
  - ~~For this final portion of the lab, you will interface an **external LED** and an **external switch**~~
  - ~~Do not use the on-board LEDs for the final demonstration of the lab~~
  - ~~Hint: You can make it work with on-board LED and a switch before you go to the external LED and an external switch~~

Requirements:

- Do not use any pre-existing code or library available in your sample project (such as `gpio.h`)
- You should use memory mapped peripherals that you can access through `LPC40xx.h`

```
#include "FreeRTOS.h"
#include "semphr.h"
static SemaphoreHandle_t switch_press_indication;
void led_task(void *task_parameter) {
  while (true) {
    // Note: There is no vTaskDelay() here, but we use sleep mechanism while waiting for the binary se
    if (xSemaphoreTake(switch_press_indication, 1000)) {
      // TODO: Blink the LED
```

```
    } else {
      puts("Timeout: No switch press indication for 1000ms");
    }
  }
}
void switch_task(void *task_parameter) {
  port_pin_s *switch = (port_pin_s*) task_parameter;

  while (true) {
    // TODO: If switch pressed, set the binary semaphore
    if (gpio0__get_level(switch->pin)) {
      xSemaphoreGive(switch_press_indication);
    }

    // Task should always sleep otherwise they will use 100% CPU
    // This task sleep also helps avoid spurious semaphore give during switch debeounce
    vTaskDelay(100);
  }
}
int main(void) {
  switch_press_indication = xSemaphoreCreateBinary();

  // Hint: Use on-board LEDs first to get this logic to work
  //       After that, you can simply switch these parameters to off-board LED and a switch
  static port_pin_s switch = {...};
  static port_pin_s led = {...};

  xTaskCreate(..., &switch);
  xTaskCreate(..., &led);
  return 0;}
```

Upload only relevant `.c` files into canvas. A good example is: `main.c`, `lab_gpio_0.c`. See Canvas for rubric and grade breakdown.

? **Extra Credit**
Add a flashy easter egg feature to your assignment, with your new found LED and switch powers! The extra credit is subject to the instructor's, ISA's and TA's discretion about what is worth the extra credit.

> Be creative. Ex: Flash board LEDs from left to right or left to right when button pressed, and preferably do this using a loop rather than hard-coded sequence

---

Revision #28
Created 5 years ago by Preet Kang
Updated 2 years ago by vidushi