Lab: SPI bus contention with SD Card

Overview

1. Build your own SPI2 driver
   - You will initialize your pins
   - You will initialize SPI2 peripheral

2. Using existing driver, validate the micro-SD card API to get "sector count"
   - The sector count gives you information about the total capacity of the SD card

3. Replace the SD card to use your own functions
   - Validate if you are able to read the data correctly

4. Produce a deliberate race condition while two tasks are reading the sector count
5. Use a mutex to solve the race condition

Part 0: Micro-SD Card

In this part, you will familiarize yourself with the low level functions that use the SPI bus to communicate with the (micro) SD card. You will use pre-existing SPI driver in this part just to prove that you are able to communicate with the SD card.

Inspect the following:

1. Do a code walkthrough of the `sd_card.h`
2. Study the `sd_card__ioctl()` function
3. Ensure that your micro-SD card is formatted with **FAT32**

Insert the SD card, and obtain and print the following information in an RTOS task:

1. Sector count
2. Sector size
3. Block size
4. Card Identification Number (`MMC_GET_CID`)

Part 1: SPI Driver

In this part, write your own SPI driver. Note that, there is already the `ssp2.h` driver, but the purpose of this exercise is to get practice writing a driver on your own because you will be tested (and quizzed) on this.

```
 void spi2_power_on(void) {
  // TODO: Enable PCONP bit: LPC_SC->PCONP
}
void spi_init(void) {
  // TODO:
}
char spi_exchange_byte(char byte_out) {
  // TODO: Ask SPI peripheral to start to send the data
  // TODO: Wait for transfer to complete
  // TODO: Capture the data
}
```

Part 2: Validate your SPI Driver using Part 0

In this part, you will swap out the existing SPI driver with your hand written driver.

```
// Open : sd_card_defiines.h
// Swap out existing driver implementation with your own:
#define SD_CARD__TRANSMIT_BYTE(byte)    YOUR_ssp2__exchange_byte(byte)
#define SD_CARD__RECEIVE_BYTE()         YOUR_ssp2__exchange_byte(0xff)
```

Ensure that you can still read the data from the SD card using your own SPI functions. Note that going through the exercise to develop your own driver is critical, and you don't want to merely copy the existing implementation. Hence, please make sure that you are able to write a driver by reading the datasheet alone.

Part 3: Create race condition

In this part, we will re-use the code from Part-0, but attempt to do so in multiple tasks simultaneously.

```

```

```
static void sd_card_read_task(void *p) {
  while (1) {
    // TODO: Read SD card parameters
    // Print them out
    // TODO: Add a small amount of sleep
  }
}
void main(void) {
  // Create two reader tasks that will attempt to read the SPI bus at the same time
  xTaskCreate(sd_card_read_task, "sd", 4000/sizeof(void*), NULL, PRIORITY_LOW, NULL);
  xTaskCreate(sd_card_read_task, "sd", 4000/sizeof(void*), NULL, PRIORITY_LOW, NULL);
  vTaskStartScheduler();}
```

Part 4: Mutex to the rescue

In this part, solve the race condition issue by using the SD card. Simply surround the critical section of the code that you don't want to access in multiple tasks. See the following pages for reference:

- Mutexes

For bonus points, can you figure out how the File I/O is race condition free that uses the SD card APIs? For example, try reading or writing files simultaneously in multiple tasks and you will notice that the code works robustly. Can you figure out the mutex that the File I/O APIs use to solve this problem?

Revision #10
Created 1 year ago by Preet Kang
Updated 1 year ago by Preet Kang