Lab Assignment: Device Interfacing w/ SPI + Data Structures

To learn how to create a single dynamic thread-safe driver for **Synchronous Serial Port** and to communicate with an external **SPI Flash** device.

This lab will utilize:

- Mutexes
- Enumerations
- Bit field and structure mapping

Assignment

Part 0: Simple SPI driver

This part is just for you to get started. After you get something functional, you need to move on to Part1 to elaborate your code. **Thus this code is not something you should turn in**. Also, before you go further, you should first read through the SPI flash datasheet to understand what you will try to communicate with.

This is your first step to completing the assignment. Get the code below to work and validate that you are able to read the SPI flash memory's "signature" bytes and compare with the SPI flash datasheet to ensure that this is correct.

```
void spi_init(void)
{
  /* Init the exact SPI config to talk to the SPI flash */
}
uint8_t spi_transfer(uint8_t out)
{
}
// WARNING: This is just a sample; you will have to fill in plenty of your own code per requirements
```

```
void read_sig(void)
{
  uint8_t d[2];

  // The simplest test is to try to read the signature of the Adesto flash and print it out
  adesto_cs();
  {
    d[0] = spi_transfer(0xAA); // TODO: Find what to send to read Adesto flash signature
    d[1] = spi_transfer(0xBB);
  }
  adesto_ds();

  printf("Returned data: %x %x\n", d[0], d[1]);
}
void main(void)
{
  spi_init();
  read_sig();}
```

Code Block 1. Simple SPI test

Part 1: Elaborate **SPI** driver

Using the following class template

1. Implement ALL class methods.
2. All methods must function work as expected by their comment description.

```
class LabSpi
{
 public:
    enum FrameModes
    {
        /* Fill this out based on the datasheet. */
    };
    /**
     * 1) Powers on SPPn peripheral
     * 2) Set peripheral clock
```

```
     * 3) Sets pins for specified peripheral to MOSI, MISO, and SCK
     *
     * @param data_size_select transfer size data width; To optimize the code, look for a pattern in th
     * @param format is the code format for which synchronous serial protocol you want to use.
     * @param divide is the how much to divide the clock for SSP; take care of error cases such as the
     *
     * @return true if initialization was successful
     */
    bool initialize(uint8_t data_size_select, FrameModes format, uint8_t divide);
    /**
     * Transfers a byte via SSP to an external device using the SSP data register.
     * This region must be protected by a mutex static to this class.
     *
     * @return received byte from external device via SSP data register.
     */
    uint8_t transfer(uint8_t send);
    LabSpi();
    ~LabSpi();

  private:
   };
```

Code Block 2. SSP Driver Template Class

Part 2: SPI Flash Reader Application

Application is to retrieve the information from the SJOne board's SPI flash's and print the information about each bit (or field of bits) to STDOUT in a nicely formatted human understandable way. If the 7th bit of a status register has been set you cannot simply print that ***"bit 7 is set"***. If bit 7 is the **busy bit**, then print a sentence like, "SPI Flash is currently busy" or "SPI Flash is NOT busy."

Read the signature bits and output them every second -- vTaskDelay(1000) in the task loop.

> **WHEN** setting the clock in the application, set the clock rate to something reasonable according to the specifications of the device.

?

Part 3: Mutex

Since multiple tasks may access the SPI device in parallel, we need to create a Mutex to guard from simultaneous access from multiple tasks. In your sample project, the SPI bus is interfaced to the SD card and the SPI flash, and the guards are implemented at diskio.c file so when you perform file I/O operations, there will never be contention between multiple tasks trying to use the SPI bus. Study the diskio.c file and make sure you understand its intent.

Assume that your chip signature reader task can be instantiated twice, as in, you will create two tasks that will read the signature and output the data repeatedly. Guard your tasks to ensure that they will not step on each other in the middle of the CS() to DS() transaction. In fact, if you want to do it right, you should use the API from spi_sem.h such that even if I am trying to read a file using the terminal command, it will not interrupt your read-signature transaction.

Requirements

- Write your own SSP1 driver: Initialize the pins, clock, and the SSP driver.
- Remember that the CS signal outside of the SSP driver, and it should just be a GPIO driver
- Communicate to the Adesto flash and print out the following registers from the **SPI FLASH** (not the SSP peripheral)
  - Manufacture ID (print 8-bit hex value)
  - 16-bit Device ID (print 16-bit hex value)
  - 16-bit status register
    - Detailed description of each bit's status
    - **MUST** create and use a bit field structure map for this register and use that to print the bit information
- Ensure that you have the mutex in place during the SPI transactions
- Capture Saleae logic analyzer screenshot during the chip signature read

What to turn in:

- Place everything inside of main file or include all relevant files.
- Turn in the **screenshots** of terminal output.
- Logic Analyzer Screenshots
  - Decoded Waveform of SPI retrieving manufacture ID.

> For the logic analyzer, you must not forget to include the CS gpio pin otherwise, the waveform will not decode properly.

> SCE (Engr294) has many logic analyzers that you can loan (or borrow). Please align your work schedule to make sure you can go to SCE while it is open.

?

?

Extra Credit :

- Read page zero (first 512 bytes) using the API at spi_flash.h, and print the following:
  - **MUST** create and use a structure mapping for the whole 512 byte page.
  - All meaningful contents of the 4 partition table entries.
    - **MUST** create and use a packed bit field structure for the partition table entries.
  - Boot signature

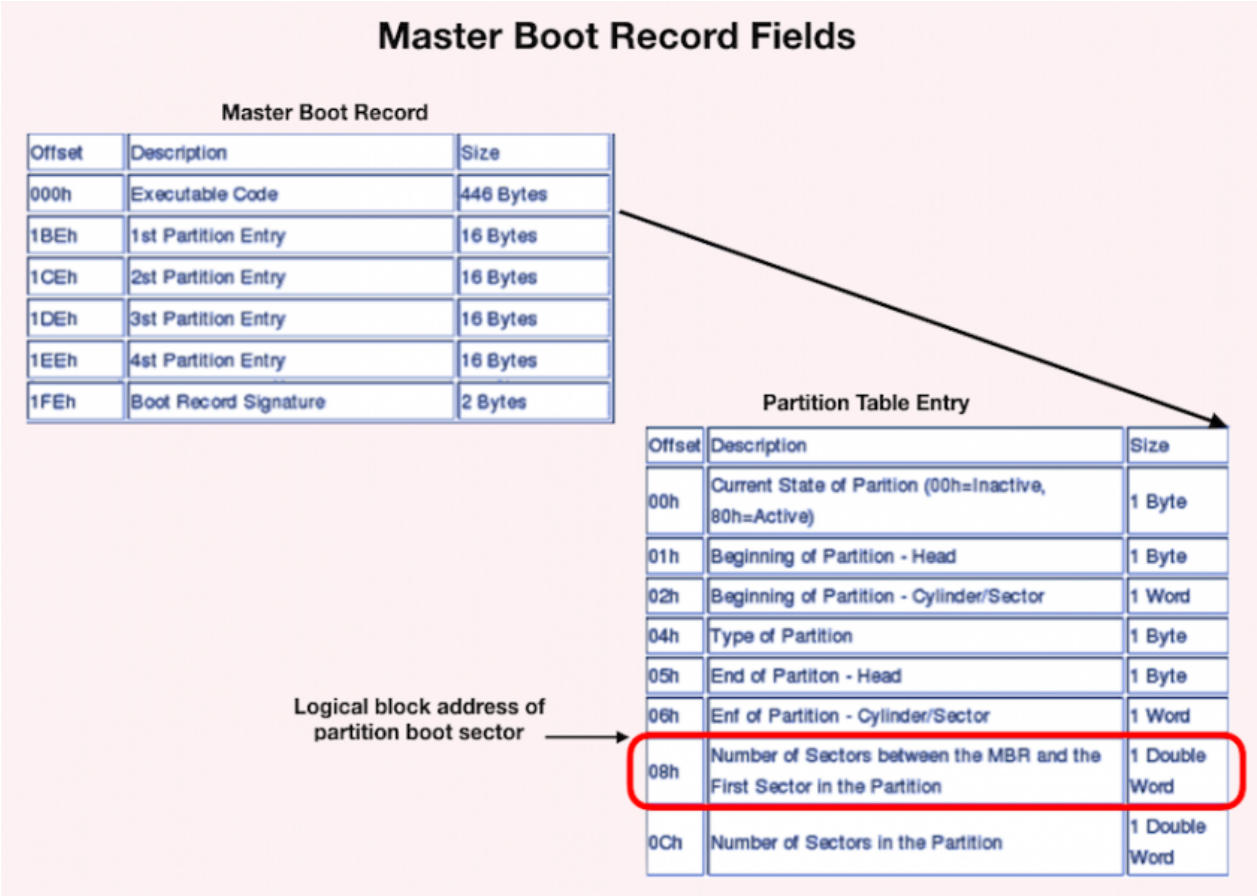
- Use a terminal command to execute this application.

FAT Information

File System essentially consists of data structures used to manage all the files in a storage device. FAT File system uses a data structure called File Allocation Table(FAT) to organize and store information about the location of files in a partition. A partition with FATFS stores all metadata about the files and file systems in its first sector called Partition Boot Sector. Each partition in a storage disk has its own Boot Sector that also contains boot code if that partition is bootable. The information about all such partitions are located in table called Partition Table which is in first sector of disk named Master Boot Record.

Master Boot Record

The first 512 bytes (Page 0 of memory) of the flash contains a Master Boot Record (MBR) which has pointers to where the partitions are placed in the storage device.



The first 446 bytes is the bootstrap code area. After that there is a partition table with four entries and then a boot signature (0x55AA). Each entry is 16 bytes long. To see the structure of the Master Boot Record: Sector layout and the structure of the entry can be found here Master Boot Record: Partition table entries.

> ? In Summary, getting FAT File System information is a 2 step process where,
>
> 1. Read the Master Boot Record to obtain the location of the starting sector of FATFS partition which is

its boot sector.
2. Read the Boot sector of the partition to obtain the FATFS information.

Additional Information:

One of the fields in the partition entry is "partition type". This should tell us what type of filesystem is resident on the partition. In our case it should be FAT12 (0x01). The last 8 bytes in the partition entry will tell us the Logical Block Address (LBA) of the first absolute sector in the partition and the total number of sectors in the partition. Essentially, we want to read the sector pointed to by this LBA for the FAT boot sector. That sector will give us the required information (no of clusters, total sectors, etc..).

Revision #29
Created 6 years ago by Admin
Updated 3 years ago by Preet Kang

?