SPI Lab

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:

- Enumerations
- Bit field 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 Part 1 to design your driver class. 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 SpiInitialize(void)
{
  // Initialize SSP peripheral
}
uint8_t SpiTransfer(uint8_t data_out)
{
  // Send data_out and retrieve returned byte
}
// WARNING: This is just a sample; you will have to fill in plenty of your own code as per requirements
void ReadSignature()
{
  uint8_t data[2];
```

```
  // The simplest test is to try to read the signature of the Adesto flash and print it out
  AdestoSelect();
  {
    // TODO: Find the correct commands you need to send to the Adesto flash device
    //       to retrieve its device signature.
    data[0] = SpiTransfer(0xAA);
    data[1] = SpiTransfer(0xBB);
  }
  AdestoDeselect();

  printf("Returned data: %x %x\n", d[0], d[1]);
}
int main()
{
  SpiInitialize();
  ReadSignature();}
```

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);


 private:
// Fill in as needed };
```

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. Use the `#include "utility/time.hpp"` file's `Delay(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. 1 Mhz and below should be fine.

Requirements

- Write your own SSP2 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 description.
- Capture the following information from Saleae logic analyzer:
    - Complete waveform of command sent via MOSI and data retrieved via MISO from slave device along with chip select, and serial clock signals.
    - A screenshot of the **Decoded Protocol section** (on the lower right) including
        - manufacture id
        - 16-bit device id

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

- Write your group name along with the names of your group members to the SPI flash memory at address zero
- Be able to read back the names in that section of flash memory.

Revision #3
Created 5 years ago by Khalil Estell
Updated 5 years ago by Khalil Estell