Exercise Sheet 4 - SPI

Background

In the last regular exercise sheet, we are going to access the flash memory IC IC12:M25P16 of board 2 by the so-called serial peripheral interface (SPI). SPI is following a synchronous protocoll: A master device (here: the MSP430) is connected to serveral slave devices by three bus lines, MOSI (master out slave in, the data output of the master), MISO (master in slave out, the data output of the slave) and SCK (serial clock applied by the master). Additionally, the master features one CS (chip select) line to every slave to address this particular device. The communication to a slave is enabled by pulling CS low. Then, sequences of 8 bits are transmitted either on the rising or on the falling edge of SCK, while data is applied to the MOSI line. At the same time, the slave can apply its data to MISO, so that a full-duplex communication can be established.

Getting Started

To start, perform the following steps:

1) Familiarize yourself with the SPI interface.

2) Establish the wiring of exercise sheet 3.

3) SPI Config: While the SI and SO signals are already connected from exercise 3 (they share the MSP430 pins with the I2C signals SCL and SDA), set up the wiring for the clock signal: CON2:P1.5 to CON6:CC_CLK.

4) Connect the SPI lines of the flash memory to the common SPI signals of the board, i.e. CON6:CC_SO to CON9:F_SO, CON6:CC_SI to CON9:F_SI and CON6:CC_CLK to CON9:F_CLK. The CS activating the SPI of the flash shall directly come from the microcontroller, so join CON9:F_/CS to CON4:P3.4.

5) The flash memory requires two additional control signals except for the SPI interface (see datasheet): CON9:F_/HOLD to CON4:P3.3, and CON9:F_/WP to CON4:P3.5.

6) Create the files `spi.c` and `flash.c`.

Task 1

a) Implement the SPI library `spi.h` using interrupts for the data handling. This includes the initialization of the module **(1 pt.)**, read and write functions **(1 pt. each, i.e. 2 pts. total)** as well as the creation of the interrupt service routines **(1 pt.)**. Keep the following things in mind:

- I2C and SPI share the same hardware module on the MSP430 (and UART, as you know). As a result, they also share some pins (remember the IC4:74HCT4066 for routing the MSP430 pins either to the I2C or to the SPI lines).
- Consider that the SPI clock can be IDLE low or IDLE high and that data can be captured on the rising or the falling edge; for further information, the diagrams of the English Wikipedia are quite useful: `https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus`

- Use your logic analyzer to debug the signals. Trigger e.g. on the falling edge of `CS` to capture the beginning of a packet.

b) Implement `flash.h` so that you can save **(1 pt.)** and read **(1 pt.)** data to/from the flash memory. Keep in mind that flash memory blocks have to be cleared before you can write to them again.

c) Now include the I2C library of exercise 3, together with the ADAC library. You will see that SPI and I2C share the same interrupt service routines (ISRs). As you can't have the ISRs twice, extract them into a separate file and implement the common ISR with callbacks such that both libraries can dynamically hook their own functions to the interrupt. As a result, you just call function pointers in your common ISR, while the individual libraries set the function pointers to their individual routines that need to be performed in case of a USART interrupt. An example of how to use function pointers is given in listing 1 (see below). **(1 pt.)**

Task 2

a) As we have now the possibility to store data permanently on the flash memory (the data is of course maintained when cycling power), we want to make use of that feature: Implement an interface (using the joystick and the display) which allows to enter a text message. Store this message in the flash memory and show it on the display as a welcome message, i.e. every time the controller is starting. **(2 pts.)**

b) **Bonus:** Implement the ability to enter the welcome message by the UART interface. Therefore, encapsulate the UART functions of the `templateEMP.h` into a separate library (`UART.h` and `UART.c`). Remember to implement the callback feature for UART, too. **(2 bonus pts.)**

Task 3

Prepare your exercise sheet for submission using the following subtasks **(1 pt.)**:

a) Fill the file `feedback.txt` with a brief feedback statement, which contains specific problems and issues you experienced while solving the exercise, additional requests, positive remarks and alike.

b) Please assure that every file contains a header comment including...

- your personal information (i.e. name, matriculation number, e-mail contact).
- the exercise sheet number.
- a brief description of the given code and its purpose; the description of the file *main.c* should also list the required pin connections.

c) Rename your project to *Exercise_[ExerciseNo]_[YourLastName]* within Code Composer Studio.

d) Export your project to an archive file (ZIP) using Code Composer Studio (File > Export... > General > Archive File).

e) Upload your file to ILIAS considering the individual deadline.

Listing 1: Function pointer example: The function pointer `operation` is either set to the add-function or to the or function.

```
#include <templateEMP.h>

int add(int a, int b)
{
    return a + b;
}

int sub(int a, int b)
{
    return a - b;
}

void delay(int ms)
{
    int i;
    for (i = 0; i < ms; i++)
    {
        _delay_cycles(1000);
    }
}

int main(void)
{
    initMSP();

    // Declare function pointer variable:
    int (*operation)(int, int);

    // variables:
    int op1 = 10;
    int op2 = 5;
    int result;

    while (1)
    {
        operation = add; // Set function pointer
        result = operation(op1, op2); // add

        // Print result:
        serialPrintInt(result);
        serialPrintln(  );

        operation = sub; // Set function pointer
        result = operation(op1, op2); // subtract

        // Print result:
        serialPrintInt(result);
        serialPrintln(  );

        delay(1000);

    }
}
```