AMS Lab Exercise 4
Serial Peripheral Interface (SPI)

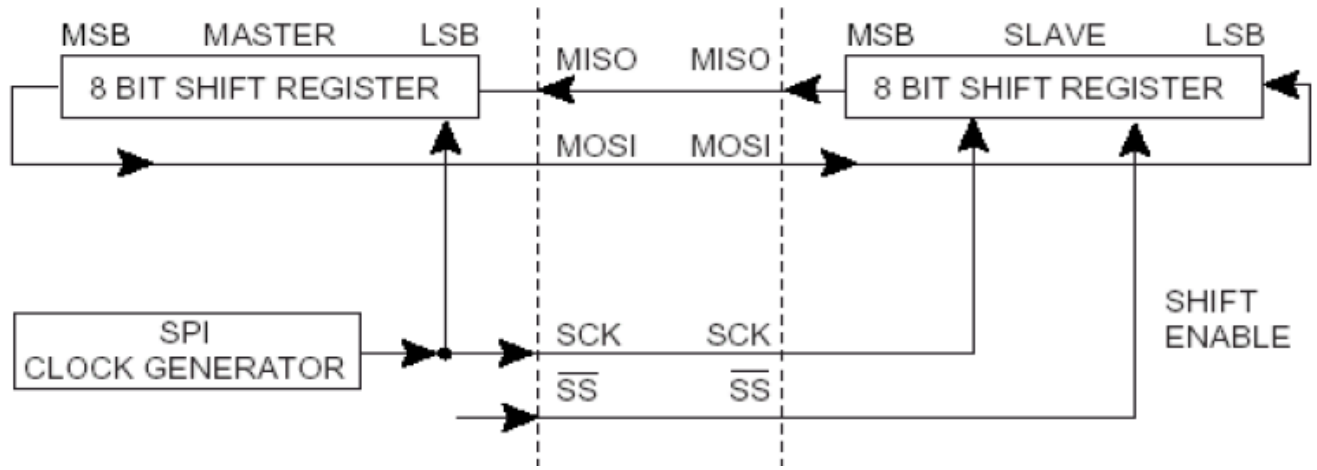
Purpose
To get experienced with the Mega32 SPI interface.

Literature
- Data sheet for: DS1305 Real Time Clock.

General information
SPI is based on a synchronous communication bus, meaning the receiver and the transmitter must share the same clock signal. Normally, the SPI bus is used for communications for very short distances among SPI compatible devices or microcontrollers placed at the same printed circuit board.
This is in contrast to the common use of the UART, normally being used for communications over larger distances – typically between two or more physically separated systems.

SPI communication involves a master and one or more slaves.
The master and the slave(s) transmit and receive data simultaneously, but only the master supplies the clock signal. Thus, the master decides when (and at what speed), data is to be exchanged at the bus.

This figure shows the very simple connection between an SPI master and an SPI slave:



The master generates the clock signal and supplies the 8 data bits, to be shifted out at the MOSI pin (**M**aster-**O**ut-**S**lave-**I**n). These 8 bits will be shifted in at the slave MOSI pin (one bit per clock cycle). At the same time, 8 bits from the slave will be shifted out at the slave MISO pin (**M**aster-**I**n-**S**lave-**O**ut) and into the master MISO pin.
Thus, the master and the slave will exchange 8 bit data by a single SPI communication cycle initiated by the master unit.

The slave SS pin has to be held active during the data transfer.
To synchronize the slave to the master, the master normally will pulse the SS pin between the transfer cycles.

The Exercise

In this exercise we are going to use the Mega32 SPI interface.

Important notice: When downloading programs to the STK500, the SPI interface is used (for ISP serial programming)!
The connection from the MOSI pin to the MISO pin has to be removed while downloading!
Otherwise we will have bus conflicts that can cause problems while programming.

You might want to start by reading the (quite well written) application note AVR151 (available at Campusnet, folder LAB4).

Part 1: "Simple SPI test"

For Mega32: Connect the MOSI pin to the MISO pin.
By doing this, the SPI transmitted data will be received by the microcontroller itself (loop back).

Connect the STK500 switches to PORT D, and the STK500 LED's to PORT C.

Write and test a C program, initializing the SPI interface as master and select a proper clock frequency.
Then the program should read the STK500 switches (at PORT D), and transmit these data using SPI.
The received SPI data should be sent to the STK500 LED's (PORT C).
Use the SPI interrupt for the solution (the main program only has to initializing – then enter an endless loop).
After initialization, there shall be a continuous transfer of data from the switches to the LED's.

Part 2: "Interfacing the DS1305 RTC"

At the "AVR Demo Board" there is a Real Time Clock device (DS1305) having SPI interface.
This demo board can be borrowed for the exercise.

Write a program that interfaces this RTC.
At least, you should be able to set the RTC to fixed values and to read and display the date and time at the LCD display (also mounted at the "AVR Demo Board") once every second.

The date and time format shown at the display could be like:

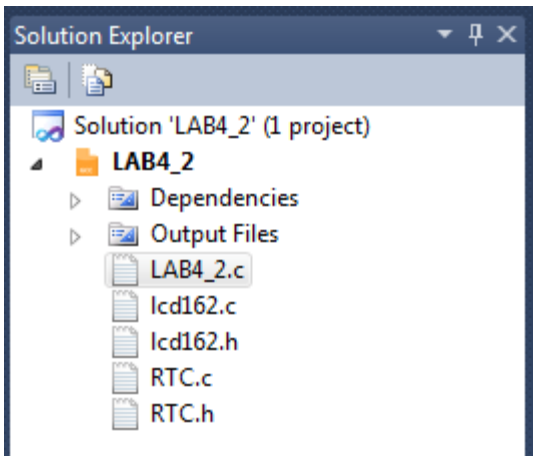
"January 12 2012"
"Thursday 12:03:45"

For a good program structure (and possible later reuse) it might be a good idea to write a driver for DS1305 RTC. A proposal for the driver's header file (made available at Campusnet) could be as shown at the next page.

```
void RTCInit();
void SetClock(unsigned char year, unsigned char month, unsigned char date, unsigned char day,
              unsigned char hour, unsigned char minutes, unsigned char seconds);
char Year10();
char Year1();
char Date10();
char Date1();
char Hours10();
char Hours1();
char Minutes10();
char Minutes1();
char Seconds10();
char Seconds1();
unsigned char Month();
unsigned char Day();
```

RTCInit() is for general initialization of the RTC.
SetClock() is used for initiating the date- and time registers.
The rest of the functions is for reading the actual registers.

Then the structure of the project would be like:



”lcd162.c” and ”lcd162.h” is the LCD driver from LAB3.

The DS1305 data sheet is available at Campusnet (folder LAB4).
Obviously, this data sheet has to be studied carefully before writing the program.