Low Power and Embedded Systems - The Serial Peripheral Interface (SPI)

Introduction

This was originally written as a workbook, but has now been changed to a worked example for how to interface a flash memory to the microcontroller using the Serial Peripheral Interface (SPI). The original exercises are now described as steps, with an explanation of the testing involved at each step.

Supporting material

atmega168P.pdf	http://www.cl.cam.ac.uk/teaching/0910/P31/docs/atmega168P.pdf Data sheet for the Atmel ATMEGA168P used in these exercises. You will need to refer to this frequently. Within these workbooks this will be referred to as 'the datasheet'. The section numbers referred to in these workbooks refer to revision 8161C of the datasheet dated 05/09.
LP2950.pdf	http://www.cl.cam.ac.uk/teaching/0910/P31/docs/LP2950.pdf Data sheet for the 3.3 Volt voltage regulator.
connect_m25p16.pdf	http://www.cl.cam.ac.uk/teaching/0910/P31/docs/connect_m25p16.pdf Circuit diagram for attaching the Flash ROM, including creating a 3.3V supply from a 5V one.
M25P16_datasheet.pdf	http://www.cl.cam.ac.uk/teaching/0910/P31/docs/M25P16_datasheet.pdf Data sheet for the Serial Flash RAM.
spi_ram.pdf	http://www.cl.cam.ac.uk/teaching/0910/P31/docs/spi_ram.pdf Circuit Diagram for wiring up Serial Flash Ram.
serial_ram.c	http://www.cl.cam.ac.uk/teaching/0910/P31/code/serial_ram.c C code for low level functions to access devices attached via an SPI interface.
serial_ram.h	http://www.cl.cam.ac.uk/teaching/0910/P31/code/serial_ram.h C header for low level functions to access devices attached via an SPI interface.

An on-line version of this guide is available at:

workbook5.html [http://www.cl.cam.ac.uk/teaching/0910/P31/workbook5.html]

Background

This week you will learn how to use the Serial Peripheral Interface (SPI) bus, a 3-wire high speed serial connection. Many sensors use this bus, as do some memory devices.

The SPI bus consists of one device which is the Bus Master (the microcontroller in this example) and one or more slaves. The bus is a 3 wire system. The master controls the timing on the bus using a clock signal SCK. There are two data lines, Master Out Slave IN (MOSI), which is an output from the master (the microcontroller) hence an input to slaves, and Master In Slave Out (MISO), which is an input to the Master. To avoid contention if more than one slave is present, the slave devices normally also have a Chip Select (CS) pin, which when active (low) allows the slave to output to the MISO line. Slaves whose CS pin is inactive (high) put their outputs into a high impedance mode so they do not interfere.

In this worked example we will interface an SPI flash memory device, the M25P16 to the ATMEGA168 Microcontroller. Because all communication is via the SPI bus, only 4 pins are required for interfacing, compared with perhaps 24 for parallel operation. The microcontroller controls the communications by toggling the SCK signal, to send commands, addresses, and also to read or write sequences of data to the device. A single command may be followed by up to 256 bytes of data. In read mode, the microcontroller still retains overall control since it controls SCK, even when the data is being output from the memory device. One constraint is that the microcontroller must not toggle SCK faster than the memory device can accept, but in practice the memory device can run at 50MHz, mush faster than the microcontroller.

Benefits of this flash memory device:

Access to the device is via the SPI bus. This has the advantage that we use only 3 I/O pins on the microcontroller. The device uses flash memory, so data it contains is retained after power down, i.e., it is non-volatile.

The SPI bus is very fast compared to others such as the I2C bus. Because the input and output lines never change direction the delays associated with turning lines round are not present. Because the Master device clocks the bus, the delays waiting for existing traffic to clear are minismised.

Drawbacks of this particular device:

1. For efficiency data should be written in 256 byte pages.
2. Bits can only be cleared, not set. Effectively each memory location is write-once. There is a sector erase function, which sets all bits in a sector back to 1.
3. Writing to flash memory is quick, but not instant. Erasing the complete device can take several seconds. We must poll the device after issuing an erase command to see if it is ready before issuing any further commands.

In some data logging applications, a large data storage capacity and data retention after power down are vital. For more capacity, it is possible to use SPI interfacing to connect MMC memory cards to the microcontroller. The following gives comparative memory capacities for 3 devices.

`ATMEGA168`	1K byte static RAM
`M25P16`	2M byte flash RAM
`MMC`	512M byte flash RAM (the size is limited by the FAT16 filesystem, other filesystems may be too big for implementation on a microcontroller)

For this example we will add a library containing some low level functions (eg, write byte to SPI), and then build upon this for higher level functions (eg, copy 256 bytes from RAM to flash). When building up in this way, it is vital that each low level function is well defined. As an example, consider what would happen if most functions exited leaving SCK low, but one left SCK high. Depending on the order in which functions are called, the memory device might encounter an extra transition of SCK, and would abort the command.

Libraries and Portability

Libraries can save a great deal of time if used correctly. The main advantage of using a library is that the function implemented in the library is thoroughly tested, and provided any requirements and constraints for the library are met, the functions should behave correctly.

In the microcontroller world, there are some potential problems:

1. Memory space is often in short supply. Library functions should be written as a set of small-footprint functions each of which does a simple task.

2. For IO functions especially, the naming of pins can cause problems. Either the library constrains the choice of pins, or it must use generic names, and these must be mapped to actual IO pin names, in the same way we used generic names for the LCD interfacing in workbook 4.

3. Some library functions may build upon lower level functions and the implementation of these may have timing constraints. In the same way that the level of a line must be defined on exit from a function, any delays required must also be considered when exiting the function. For example, the M25P16 device requires that at the end of a command, CS must remain low for a short time after SCK has been taken low. The delay required may be very small, but the timing constraint must be met if the device is to function correctly.

4. Library functions will require that on entry to a function, certain conditions are met, for example clock signals may need to be in a specific state.

Libraries and Naming

To overcome the second problem listed above, a library may use generic names, which are mapped to the actual pins used. This is done as follows:.

A definition such as the following maps the library name (RAM_CS in this case) to the pin in use (PB1).

```
#define RAM_CS  PB1
```

However, this definition must be made available to both the library and the main program. A convenient way of doing this is to include the definition in a header file, and including the header file in both the main code and the library. In these worksheets the file used is called `"config.h"`

Step 1

Change PSU voltage, crystal frequency and test serial communications

Important

The M25P16 flash memory is a 3.3V device, so the 5V supply needs to be regulated to a 3.3V one, the serial lead exchanged for a 3.3V one, and the clock frequency reduced since the microcontroller can only run at a reduced frequency at lower supply voltages.

Hardware

1. Refer to the schematic at `connect_m25p16.pdf` [`http://www.cl.cam.ac.uk/teaching/0910/P31/docs/connect_m25p16.pdf`] and the datasheet for the voltage regulator at `LP2950.pdf` [`http://www.cl.cam.ac.uk/teaching/0910/P31/docs/LP2950.pdf`]
2. The serial lead is changed for a 3.3V one.
3. The crystal is exchanged for a 3.6864 MHz one. The rather unusual choice of frequency allows us to select a suitable divider to create an accurate baud rate clock. F_CPU is redefined in the config.h file.
4. Refer to the circuit diagram at connect_m25p16.pdf [http://www.cl.cam.ac.uk/teaching/0910/P31/docs/connect_m25p16.pdf] for the connections from the M25P16 device to the microcontroller.

Software

1. A file config.h is created in the same directory as the main program to map the names used by library functions, such as RAM_PORT to the ports actually in use in this project, e.g. PORTD. This allows the library functions to be used on any port by changing config.h, rather than editing the library file itself. For the SPI flash ram we also make use of the library spi.c for the low level writes to the device, so we also include definitions required by spi.c

   config.h contains the following definitions:

   ```
   #define RAM_CS   PD7   // Output Chip Select to memory device

   #define SPI_CLK  PD6   // Output Clock to memory device
   #define SPI_SEND PD5   // Output on microcontroller, D input on memory
   #define SPI_RECV PD4   // Input on microcontroller, Q output on memory

   #define RAM_PORT PORTD    // Library operations will use RAM_PORT not PORTD
   #define RAM_PIN  PIND
   #define RAM_DDR  DDRD
   ```

2. The following line is added to the main program and any library files called:

   ```
   #include "config.h"
   ```

3. A new baud rate setting is rquired for the revised crystal frequency.

Exercise 2 - low level SPI interface functions

At this stage we won't use the SPI functions available in the microcontroller. They do more than we need.

Instead we will use and build upon some low level functions available in spi.c. They are available in a library `serial_ram.c` `[http://www.cl.cam.ac.uk/teaching/0910/P31/code/lib/serial_ram.c]`, and header file `serial_ram.h` `[http://www.cl.cam.ac.uk/teaching/0910/P31/code/lib/serial_ram.h]`

1. Download serial_ram.h and serial_ram.c and save both to your current directory, ~/workbook5/exercise2/
2. Include a line in your file exercise2.c, to gain access to the functions listed in spi.h and serial_ram.h:

   ```
   #include <spi.h>
   #include <serial_ram.h>
   ```

3. The Makefile must be edited to compile and link `spi.o` and `serial_ram.o` in to the `.elf` file by editing the line:

   ```
   exercise2.elf: exercise2.o
   ```

   to

   ```
   exercise2.elf: exercise2.o spi.o serial_ram.o
   ```

4. Both exercise2.c and serial_ram.c will need to see the definitions such as `RAM_CS` and `SPI_CLK` created earlier

   Edit exercise2.c to include the line:

   ```
   #include "config.h"
   ```

5. The F_CPU setting must be made available to exercise2.c and to the library functions. This is done by adding it to config.h.
6. Modify the port_direction_init() function to set up the four memory interface pins SPI_SEND, SPI_RECV, SPI_CLK, RAM_CS. SPI_RECV is the connection from the Q output on the M25P16 to an *input* on the microcontroller. The other 3 lines are *outputs* from the microcontroller, and need to be defined as outputs and set to suitable initial values. Initialise `SPI_CLK` low, `SPI_SEND` low, and `RAM_CS` high.

   We use DDRD |= ... instead use RAM_DDR |= to make the code easier to maintain.

   As multiple lines in the initialisation code may refer to the same Data Direction register, it is vital to `AND` and `OR` bits, not set the whole register.

You now have access to the low level functions in spi.h but more usefully the functions in serial_ram.h which allow you to read or write to the memory device.