AVR151: Setup And Use of The SPI

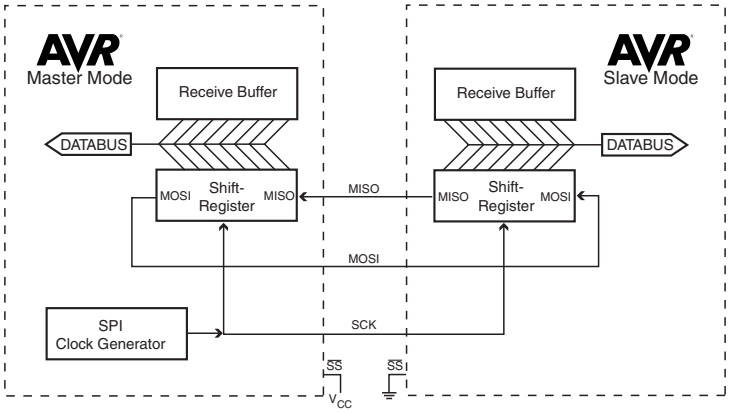
Features

- **SPI Pin Functionality**
- **Multi Slave Systems**
- **SPI Timing**
- **SPI Transmission Conflicts**
- **Emulating the SPI**
- **Code examples for Polled operation**
- **Code examples for Interrupt Controlled operation**

1. Introduction

This application note describes how to setup and use the on-chip Serial Peripheral Interface (SPI) of the AVR micro-controller. Most AVR devices come with an on board SPI and can be configured according to this document. After a theoretical background it will be shown how to configure the SPI to run in both master mode and slave mode

Figure 1. Master and Slave Interface.



AVR®

8-bit RISC Microcontroller

Application Note

Rev. 2585C–AVR–07/08

2. General description of the SPI

The SPI allows high-speed synchronous data transfer between the AVR and peripheral devices or between several AVR devices. On most parts the SPI has a second purpose where it is used for In System Programming (ISP). See application note AVR910 for details.

The interconnection between two SPI devices always happens between a master device and a slave device. Compared to some peripheral devices like sensors which can only run in slave mode, the SPI of the AVR can be configured for both master and slave mode. The mode the AVR is running in is specified by the settings of the master bit (MSTR) in the SPI control register (SPCR). Special considerations about the $\overline{SS}$ pin have to be taken into account. This will be described later in the section “Multi Slave Systems - SS pin Functionality” on page 3.

The master is the active part in this system and has to provide the clock signal a serial data transmission is based on. The slave is not capable of generating the clock signal and thus can not get active on its own. The slave just sends and receives data if the master generates the necessary clock signal. The master however generates the clock signal only while sending data. That means that the master has to send data to the slave to read data from the slave.

Note: This can be confusing especially if “passive” peripherals like sensors are used. The need to send random data to a sensor just to read its data is not always clear.

2.1 Data transmission between Master and Slave

The interaction between a master and a slave AVR is shown in Figure 1 on page 1. Two identical SPI units are displayed. The left unit is configured as master while the right unit is configured as slave. The MISO, MOSI and SCK lines are connected with the corresponding lines of the other part. The mode in which a part is running determines if they are input or output signal lines. Because a bit is shifted from the master to the slave and from the slave to the master simultaneously in one clock cycle both 8-bit shift registers can be considered as one 16-bit circular shift register. This means that after eight SCK clock pulses the data between master and slave will be exchanged.

The system is single buffered in the transmit direction and double buffered in the receive direction. This influences the data handling in the following ways:

1. New bytes to be sent can not be written to the data register (SPDR) / shift register before the entire shift cycle is completed.
2. Received bytes are written to the Receive Buffer immediately after the transmission is completed.
3. The Receive Buffer has to be read before the next transmission is completed or data will be lost.
4. Reading the SPDR will return the data of the Receive Buffer.

After a transfer is completed the SPI Interrupt Flag (SPIF) will be set in the SPI Status Register (SPSR). This will cause the corresponding interrupt to be executed if this interrupt and the global interrupts are enabled. Setting the SPI Interrupt Enable (SPIE) bit in the SPCR enables the interrupt of the SPI while setting the I bit in the SREG enables the global interrupts.

2.2 Pins of the SPI

The SPI consists of four different signal lines. These lines are the shift clock (SCK), the Master Out Slave In line (MOSI), the Master In Slave Out line (MISO) and the active low Slave Select

line ($\overline{SS}$). When the SPI is enabled, the data direction of the MOSI, MISO, SCK and $\overline{SS}$ pins are overridden according to the following table.

Table 2-1. SPI Pin overrides

Pin	Direction Master Mode	Direction Slave Mode
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

This table shows that just the input pins are automatically configured. The output pins have to be initialized manually by software. The reason for this is to avoid damages e.g. through driver contention.

2.3 Multi Slave Systems - $\overline{SS}$ pin Functionality

The Slave Select ($\overline{SS}$) pin plays a central role in the SPI configuration. Depending on the mode the part is running in and the configuration of this pin, it can be used to activate or deactivate the devices. The $\overline{SS}$ pin can be compared with a chip select pin which has some extra features.

In master mode, the $\overline{SS}$ pin must be held high to ensure master SPI operation if this pin is configured as an input pin. A low level will switch the SPI into slave mode and the hardware of the SPI will perform the following actions:

1. The master bit (MSTR) in the SPI Control Register (SPCR) is cleared and the SPI system becomes a slave. The direction of the pins will be switched according to Table 2-1.
2. The SPI Interrupt Flag (SPIF) in the SPI Status Register (SPSR) will be set. If the SPI interrupt and the global interrupts are enabled the interrupt routine will be executed.

This can be useful in systems with more than one master to avoid that two masters are accessing the SPI bus at the same time. If the $\overline{SS}$ pin is configured as output pin it can be used as a general purpose output pin which does not affect the SPI system.

Note: In cases where the AVR is configured for master mode and it can not be ensured that the $\overline{SS}$ pin will stay high between two transmissions, the status of the MSTR bit has to be checked before a new byte is written. Once the MSTR bit has been cleared by a low level on the $\overline{SS}$ line, it must be set by the application to re-enable SPI master mode.

In slave mode the $\overline{SS}$ pin is always an input. When $\overline{SS}$ is held low, the SPI is activated and MISO becomes output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Table 2-2 shows an overview of the $\overline{SS}$ Pin Functionality.

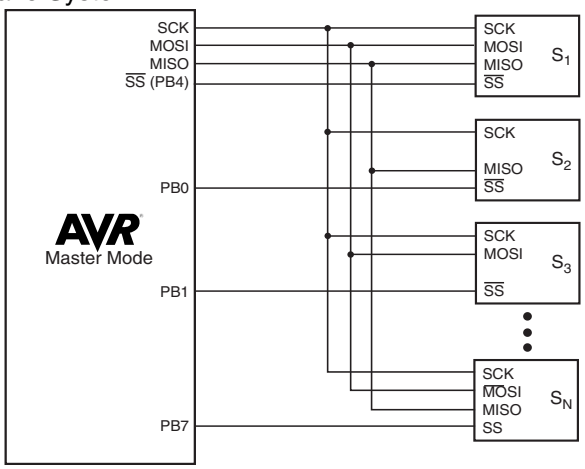
Note: In slave mode, the SPI logic will be reset once the $\overline{SS}$ pin is brought high. If the $\overline{SS}$ pin is brought high during a transmission, the SPI will stop sending and receiving immediately and both data received and data sent must be considered as lost.

Table 2-2. Overview of the $\overline{SS}$ pin functionality

Mode	$\overline{SS}$ Configuration	$\overline{SS}$ Pin-level	Description
Slave	Always Input	High	Slave deactivated (deselected)
		Low	Slave activated (selected)
Master	Input	High	Master activated (selected)
		Low	Master deactivated, switched to slave mode
	Output	High	Master activated (selected)
		Low	

As shown in Table 2-2, the $\overline{SS}$ pin in slave mode is always an input pin. A low level activates the SPI of the device while a high level causes its deactivation. A Single Master Multiple Slave System with an AVR configured in master mode and $\overline{SS}$ configured as output pin is shown in Figure 2-1. The amount of slaves which can be connected to this AVR is only limited by the number of I/O pins to generate the slave select signals.

Figure 2-1. Multi Slave System



The ability to connect several devices to the same SPI-bus is based on the fact that only one master and only one slave is active at the same time. The MISO, MOSI and SCK lines of all the other slaves are tristated (configured as input pins of a high impedance with no pullup resistors enabled). A false implementation (e.g. if two slaves are activated at the same time) can cause a driver contention which can lead to a CMOS latchup state and must be avoided. Resistances of 1 to 10 k ohms in series with the pins of the SPI can be used to prevent the system from latching up. However this affects the maximum usable data rate, depending on the loading capacitance on the SPI pins.

Unidirectional SPI devices require just the clock line and one of the data lines. If the device is using the MISO line or the MOSI line depends on its purpose. Simple sensors for instance are just sending data (see S2 in Figure 2-1), while an external DAC usually just receives data (see S3 in Figure 2-1).

2.4 SPI Timing

The SPI has four modes of operation, 0 through 3. These modes essentially control the way data is clocked in or out of an SPI device. The configuration is done by two bits in the SPI control register (SPCR). The clock polarity is specified by the CPOL control bit, which selects an active high or active low clock. The clock phase (CPHA) control bit selects one of the two fundamentally different transfer formats. To ensure a proper communication between master and slave both devices have to run in the same mode. This can require a reconfiguration of the master to match the requirements of different peripheral slaves.

The settings of CPOL and CPHA specify the different SPI modes, shown in Table 2-3. Because this is no standard and specified different in other literature, the configuration of the SPI has to be done carefully.

Table 2-3. SPI Mode Configuration

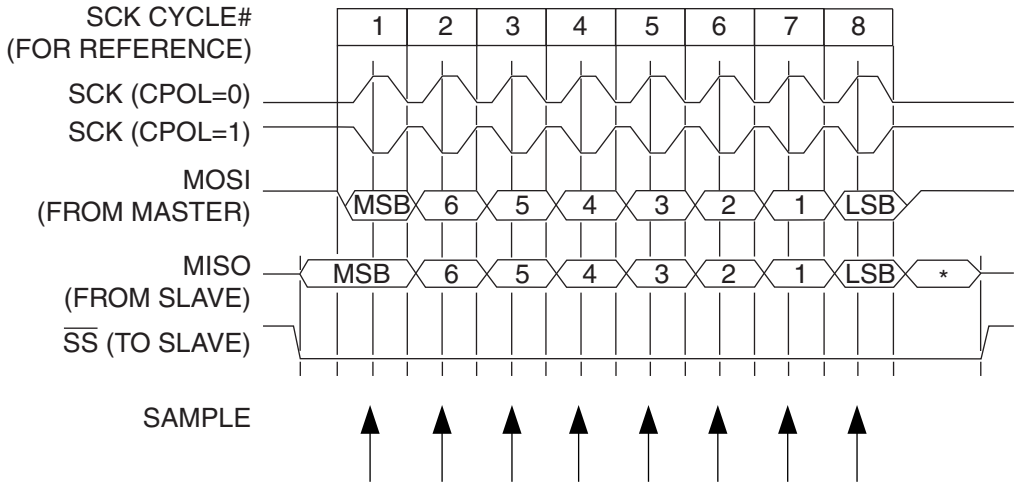
SPI Mode	CPOL	CPHA	Shift SCK-edge	Capture SCK-edge
0	0	0	Falling	Rising
1	0	1	Rising	Falling
2	1	0	Rising	Falling
3	1	1	Falling	Rising

The clock polarity has no significant effect on the transfer format. Switching this bit causes the clock signal to be inverted (active high becomes active low and idle low becomes idle high). The settings of the clock phase, however, selects one of the two different transfer timings, which are described closer in the next two chapters. Since the MOSI and MISO lines of the master and the slave are directly connected to each other, the diagrams show the timing of both devices, master and slave. The $\overline{SS}$ line is the slave select input of the slave. The $\overline{SS}$ pin of the master is not shown in the diagrams. It has to be inactive by a high level on this pin (if configured as input pin) or by configuring it as an output pin.

2.5 A.) CPHA = 0 and CPOL = 0 (Mode 0) and CPHA = 0 and CPOL = 1 (Mode 1)

The timing of a SPI transfer where CPHA is zero is shown in Figure 2-2. Two wave forms are shown for the SCK signal - one for CPOL equals zero and another for CPOL equals one.

Figure 2-2. SPI Transfer Format with CPHA = 0



*Not defined but normally MSB of character just received.

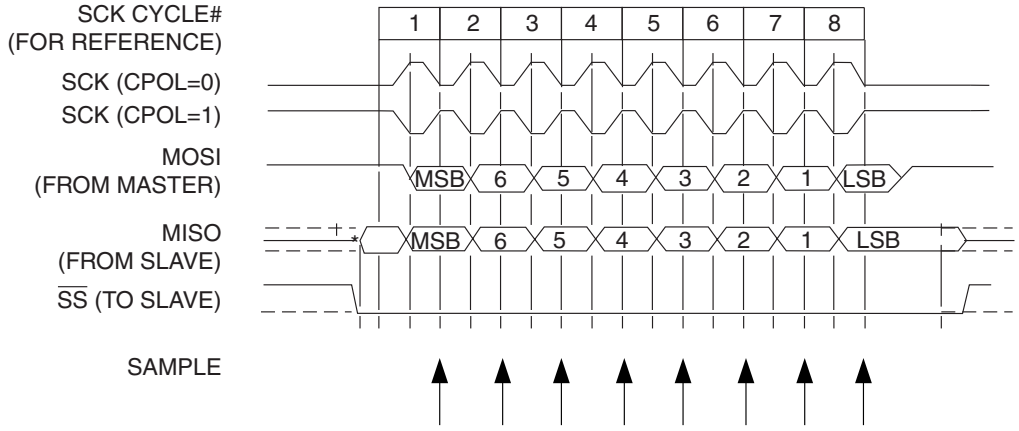
When the SPI is configured as a slave, the transmission starts with the falling edge of the $\overline{SS}$ line. This activates the SPI of the slave and the MSB of the byte stored in its data register (SPDR) is output on the MISO line. The actual transfer is started by a software write to the SPDR of the master. This causes the clock signal to be generated. In cases where the CPHA equals zero, the SCK signal remains zero for the first half of the first SCK cycle. This ensures that the data is stable on the input lines of both the master and the slave. The data on the input lines is read with the edge of the SCK line from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one). The edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one) causes the data to be shifted one bit further so that the next bit is output on the MOSI and MISO lines.

After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to the receive buffer.

2.6 B.) CPHA = 1 and CPOL = 0 (Mode 2) and CPHA = 1 and CPOL = 1 (Mode 3)

The timing of a SPI transfer where CPHA is one is shown in Figure 2-3. Two wave forms are shown for the SCK signal - one for CPOL equals zero and another for CPOL equals one.

Figure 2-3. SPI Transfer Format with CPHA = 1



*Not defined but normally LSB of previously transmitted character.

Like in the previous cases the falling edge of the $\overline{SS}$ lines selects and activates the slave. Compared to the previous cases, where CPHA equals zero, the transmission is not started and the MSB is not output by the slave at this stage.

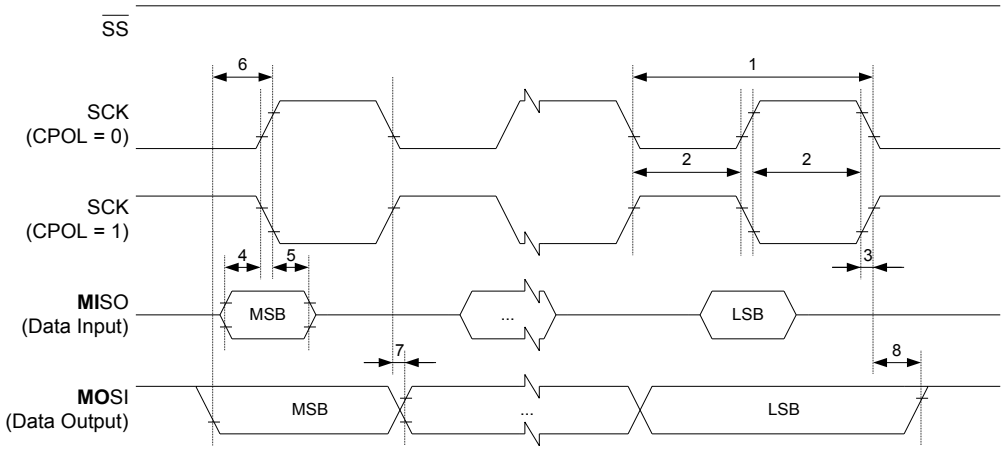
The actual transfer is started by a software write to the SPDR of the master what causes the clock signal to be generated. The first edge of the SCK signal from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one) causes both the master and the slave to output the MSB of the byte in the SPDR. As shown in Figure 2-3, there is no delay of half a SCK-cycle like in Mode 0 and 1. The SCK line changes its level immediately at the beginning of the first SCK-cycle. The data on the input lines is read with the edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one).

After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to the receive buffer.

2.6.1 Considerations for high speed transmissions

Parts which run at higher system clock frequencies and SPI modules capable of running at speed grades up to half the system clock require a more specific timing to match the needs of both the sender and receiver. The following two diagrams show the timing of the AVR in master and in slave mode for the SPI Modes 0 and 1. The exact values of the displayed times vary between the different parts and are not an issue in this application note. However the functionality of all parts is in principle the same so that the following considerations apply to all parts.

Figure 2-4. Timing Master Mode



The minimum timing of the clock signal is given by the times “1” and “2”. The value “1” specifies the SCK period while the value “2” specifies the high / low times of the clock signal. The maximum rise and fall time of the SCK signal is specified by the time “3”. These are the first timings of the AVR to check if they match the requirements of the slave.

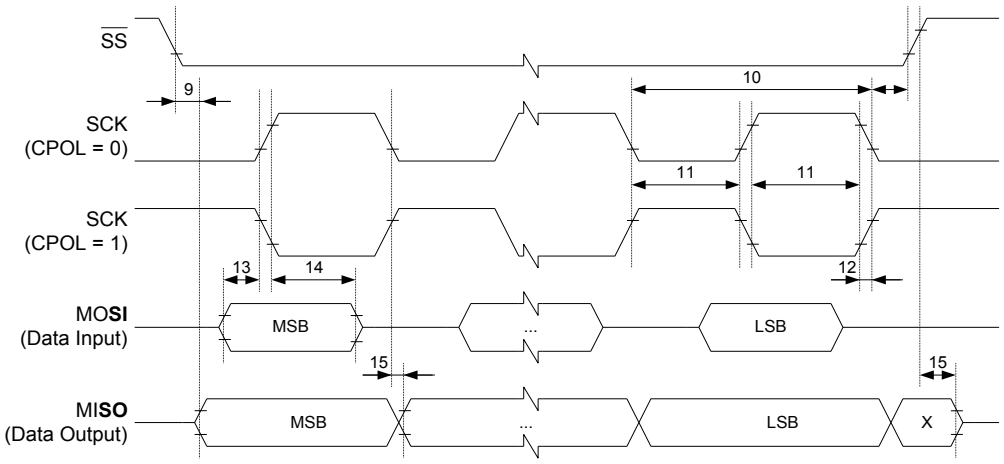
The Setup time “4” and Hold time “5” are important times because they specify the requirements the AVR has on the interface of the slave. These times determine how long before the clock edge the slave has to have valid output data ready and how long after the clock edge this data has to be valid.

If the Setup and Hold time are long enough the slave suits to the requirements of the AVR but does the AVR suit to the requirements of the slave?

The time “6” (Out to SCK) specifies the minimum time the AVR has valid output data ready before the clock edge occurs. This time can be compared to the Setup time “4” of the slave.

The time “7” (SCK to Out) specifies the maximum time after which the AVR outputs the next data bit while the time “8” (SCK to Out high) the minimum time specifies during which the last data bit is valid on the MOSI line after the SCK was set back to its idle state.

Figure 2-5. Timing Slave Mode



In principle the timings are the same in slave mode like previously described in master mode. Because of the switching of the roles between master and slave the requirements on the timing are inverted as well. The minimum times of the master mode are now maximum times and vice versa.

2.7 SPI Transmission Conflicts

A write collision occurs if the SPDR is written while a transfer is in progress. Since this register is just single buffered in the transmit direction, writing to SPDR causes data to be written directly into the SPI shift register. Because this write operation would corrupt the data of the current transfer, a write-collision error in generated by setting the WCOL bit in the SPSR. The write operation will not be executed in this case and the transfer continues undisturbed.

A write collision is generally a slave error because a slave has no control over when a master will initiate a transfer. A master, however, knows when a transfer is in progress. Thus a master should not generate write collision errors, although the SPI logic can detect these errors in a master as well as in a slave mode.

2.8 Emulating the SPI

When emulating the SPI using the ICE200 hardware emulator, be aware of the fact that the peripherals on this emulator are not stopped on a break point but continue to run with the speed they are configured for.

When emulating the SPI using the ICEPRO the timing can be less accurate than it is the case on the part itself. This is caused by longer internal signal lines of the ICEPRO which is the price we had to pay for its ability to upgrade and its flexibility.

2.9 Setup the SPI

The configuration of the SPI in master mode will be shown in two different ways. The first example will show how to implement an SPI communication which is controlled by polling the interrupt flags. The second example will show how to implement an interrupt controlled communication.

A communication between two AVR devices will be shown by sending a “Text String” from the part configured as master to the other part configured as slave. The received characters will be compared to the expected ones and the result of this communication test will be output on the Port D. These examples are well suited to be implement by using two development boards like the STK500.

In all the examples shown here the SPI is configured to run in mode 0 with MSB transmitted first. This is done by setting the bits CPOL, CPHA and DORD in the register SPCR to zero. In the same register the SPI is enabled by setting the SPE bit, while the SCK frequency is specified to CK/4 in the first example and the assembler code of the second example. It is set to CK/16 in the C code of the second example.

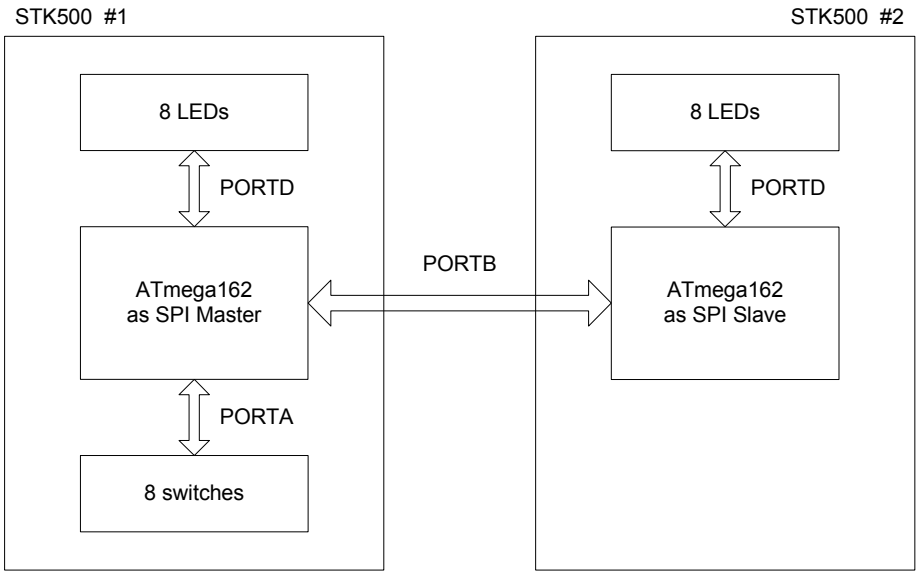
To compare the configuration of the SPI in the different examples the attention has to be directed on the settings of the Master / Slave Select (MSTR) bit and the SPI Interrupt Enable (SPIE) bit.

Notes:
1. Because both examples show the transmission between a single master and a single slave it is not necessary to check if the MSTR bit is still set before the master initiates a new transmission. This code has to be added in a multi master application.
2. Although the settings of the Clock Rate Select bits have no effect in slave mode it has to be ensured that the system clock (CK) of the slave is at least four times higher than the SPI clock (SCK).
3. Pending SPI interrupts are cleared by a dummy access to the SPSR and the SPDR.

Two files come along with this application note which contain the C code shown in this examples.

To run the code, setup two STK500 development boards as shown in Figure 2-6. The code is written for ATmega162, but can be compiled for any part with hardware SPI and PORTA, PORTB and PORTD.

Figure 2-6. Hardware setup

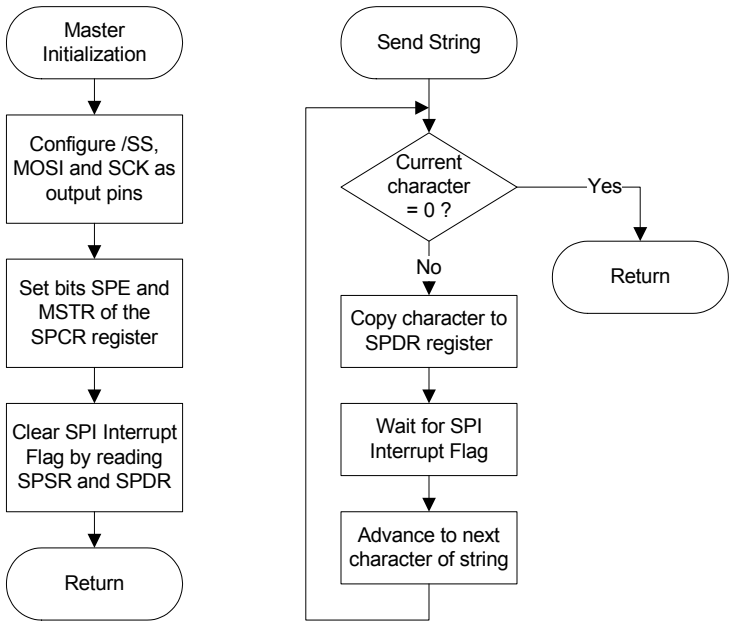


2.9.1 Example 1 - SPI communication controlled by polling:

2.9.1.1 Master Side:

If no interrupts are used there is just the SPI module and its pins to configure. Important in this example is the setting of the $\overline{SS}$ pin as output pin. This has to be done before the SPI is enabled in master mode. Enabling the SPI while the $\overline{SS}$ pin is still configured as an input pin would cause the SPI to switch to slave mode immediately if a low level is applied to this pin. This pin is always configured as an input pin in slave mode (see Figure 2-7 on page 11). Using polling gives the fastest communication. This is why polling is most commonly used in master mode.

Figure 2-7. Polled master - initialization and transmission

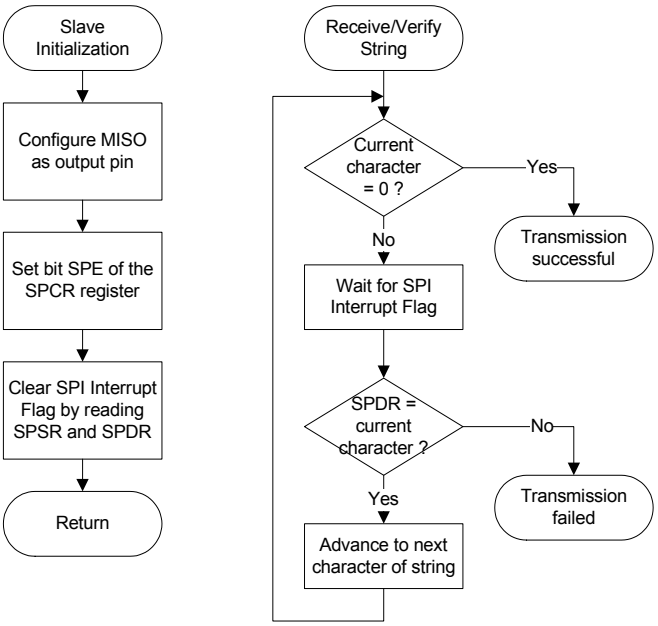


2.9.1.2 Slave Side:

To configure the AVR to run in slave mode there is no order required in which the registers have to be initialized. The MISO pin has to be defined as an output pin, while all other pins are configured automatically as input pins if the SPI is enabled (see Table 2-8). To configure the AVR to run in slave mode the MSTR bit has to be set to zero. In this case the Clock Rate Select bits SPR0 and SPR1 do not care because of the synchronous transmission.

All other settings of the SPI configuration register (SPCR) have to be the same as in master mode. This is essential for a successful communication between the two devices.

Figure 2-8. Polled slave - initialization and reception



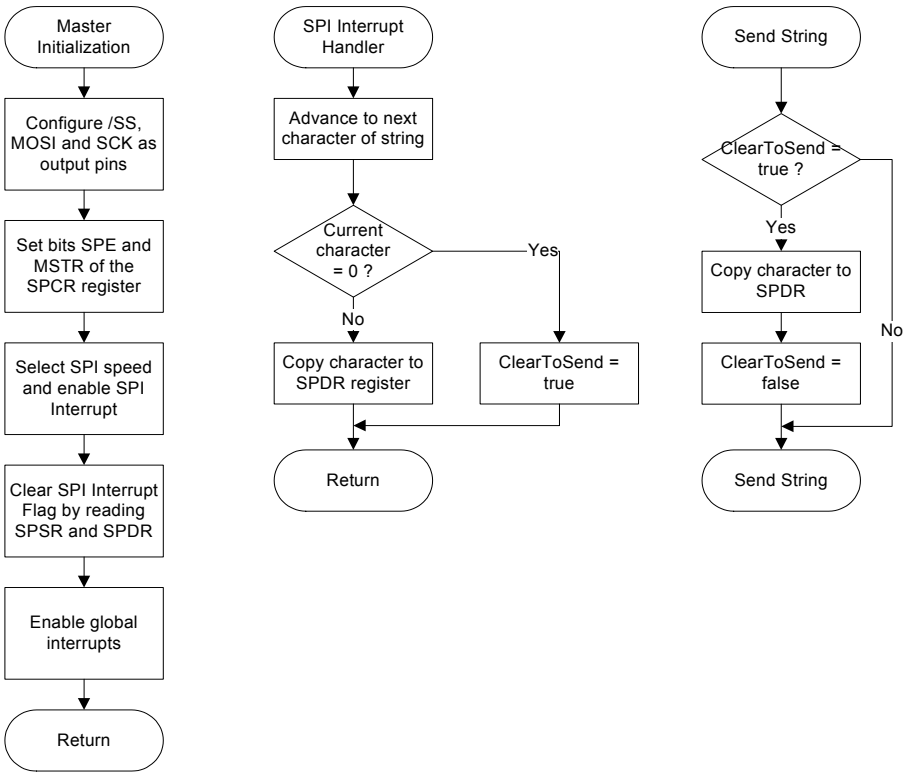
2.9.2 Example 2 - SPI communication controlled by interrupts:

In master mode interrupt controlled communication makes mainly sense if the SCK clock is generated by dividing the system clock by a large division factor (like 64 or 128). In this case the processor can do other processing instead of just waiting to send/receive the next byte. In slave mode where the part does not know when a communication starts an interrupt controlled implementation can ensure that the part will react in time so that write collision errors will be avoided.

2.9.2.1 Master Side:

The initialization of the SPI happens in a similar way to the one in the previous example. Like before in master mode the $\overline{SS}$ pin has to be set as output first and then the SPI can be enabled. The SPI interrupt is enabled by setting the SPIE bit in the SPCR.

Figure 2-9. Interrupt controlled master - initialization and transmission

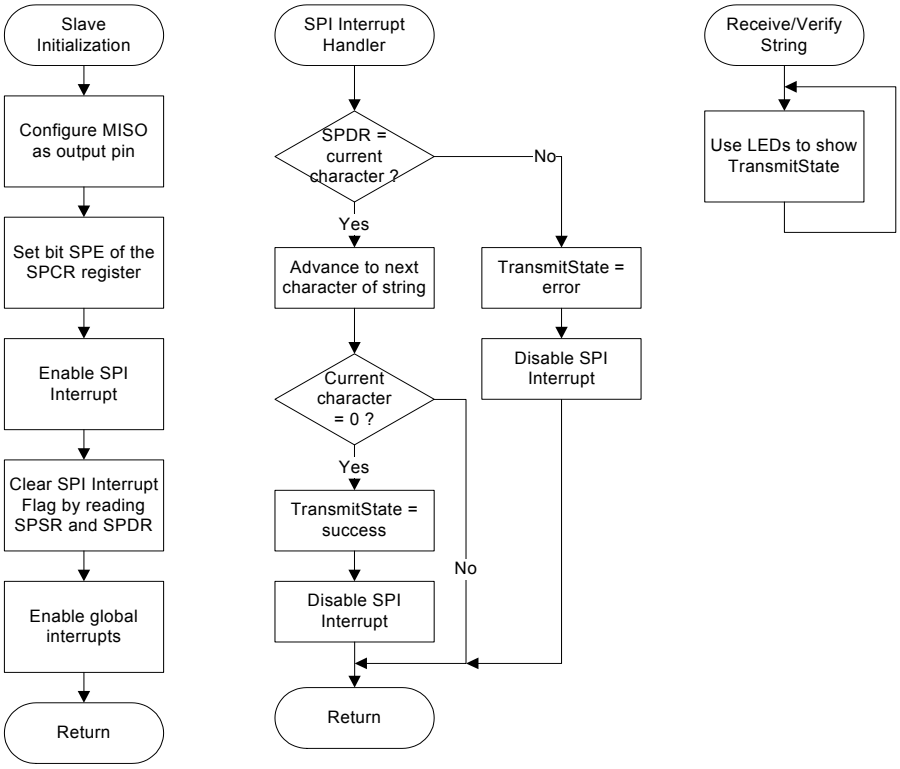


2.9.2.2 Slave Side:

A slave never knows when the master is going to start a new communication. Interrupts are a perfect feature to react on such undetermined events so this is a common way to implement the SPI in slave mode.

In this example the main program has to be notified about transmission errors and the completion of the transmission.

Figure 2-10. Interrupt controlled slave - initialization and transmission



Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature