AVR1309: Using the XMEGA SPI

Features

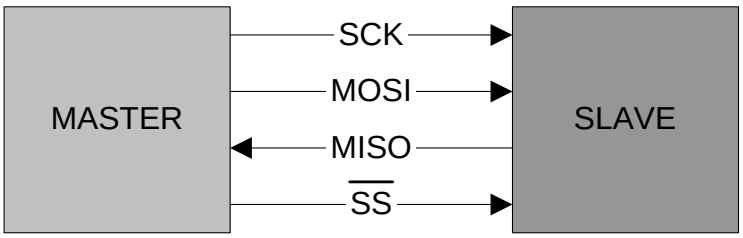
- **Introduction to SPI and the XMEGA™ SPI module**
- **Setup and use of the XMEGA SPI module**
- **Implementation of module drivers**
  - **Polled master**
  - **Interrupt controlled master**
  - **Polled slave**
  - **Interrupt controlled slave**
- **Code examples for interrupt controlled and polled drivers**

1 Introduction

This application note describes how to set up and use the SPI module in the AVR® XMEGA. Both interrupt controlled and polled C code drivers and examples are included for master and slave applications.

Serial buses are more and more preferred over parallel. The wiring is simpler, and as the efficiency of serial interfaces increases, the speed advantage of a parallel transmission gets less important. Typical peripherals that use a serial interface are converters (A/D and D/A), memories (RAM and EEPROM), real time clocks, sensors and other controllers for LCD, CAN USB etc.

Figure 1-1. Basic SPI implementation



8-bit AVR® Microcontrollers

Application Note

Rev. 8057A-AVR-02/08

2 The SPI bus

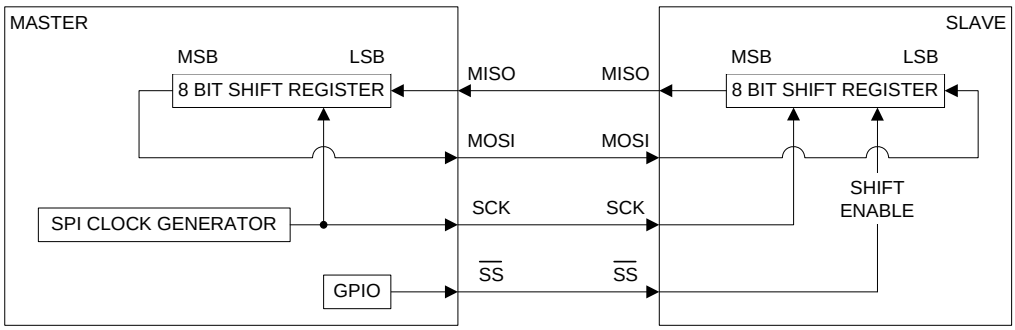
The Serial Peripheral Interface (SPI) is mainly used in synchronous serial transmissions in a master/slave relationship. The master initiates and controls the transfer, while the slave responds.

SPI is a full duplex interface, and at a low cost enabling high-speed communication between master and slave. SPI does not have a specific higher-level protocol, which means there is almost no overhead. The drawback is that there is no acknowledgement and flow control, and the master doesn't even have to be aware of the slave's presence.

2.1 Data and control lines

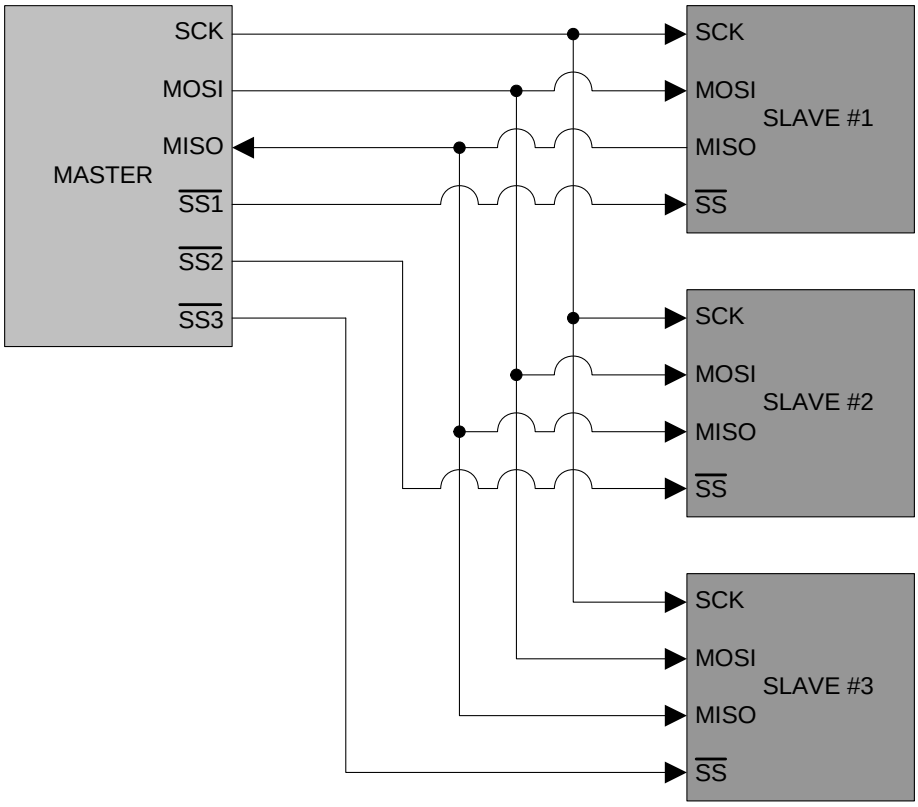
The standard SPI configuration makes use of two control and two data lines. The data lines are MOSI (Master Out, Slave In) and MISO (Master In, Slave Out), transferring data in each direction. The control lines are SCK (SPI Clock) and SS (Slave Select). If SS is used, the master selects a slave device by pulling this line low, and supplies the clock signal. Data is now transferred in both directions simultaneously, and it is up to a higher-level protocol to define the meaning of each byte.

Figure 2-1. Master/slave interconnection



If multiple slaves exists and should be independently addressed, the master must generate a SS signal for each slave. This is illustrated in Figure 2-2.

Figure 2-2. Multi slave implementation



2.2 Modes and configuration

There is no official specification for SPI communication, making flexibility of the devices important. Clock polarity (CPOL) and clock phase (CPHA) determines the data setup and sampling point, and must be configured the same for devices to communicate.

Table 2-1. SPI modes

SPI mode	Configuration		Leading edge		Trailing edge	
	CPOL	CPHA				
0	0	0	Rising	Sample	Falling	Setup
1	0	1	Rising	Setup	Falling	Sample
2	1	0	Falling	Sample	Rising	Setup
3	1	1	Falling	Setup	Rising	Sample

For more information about SPI modes and configuration, please see the XMEGA SPI data sheet.

3 The XMEGA SPI module

The XMEGA SPI module is designed for high-speed data transfers between the XMEGA and other SPI devices. The control bits allow flexible configuration to enable a flawless connection.

3.1 Registers

The SPI module consists of the baud rate generator, status and control logic with supporting registers listed in Table 3-1.

Table 3-1. SPI module registers.

Register name	C struct and element
SPI Control Register	SPIx.CTRL
SPI Interrupt Control Register	SPIx.INTCTRL
SPI Status Register	SPIx.STATUS
SPI Data Register	SPIx.DATA

All control bits with exception of the interrupt level bits, are located in CTRL. The two interrupt level bits are found in INTCTRL, and the SPI interrupt and write collision flags in STATUS.

The DATA register is a read/write register for data transfers. Reading the registers returns the data currently in the SPI shift register, while writing will initiate a data transmission. The system is single buffered in the transmit direction and double buffered in the receive direction. As a result, new data must not be written to the DATA register before the entire shift cycle is complete. To avoid losing data, a received character must be read from DATA before the next character has been completely shifted in.

3.2 The SS pin

In master mode the SS pin is fully configurable from software, and typically used as one of these three options:

- Input (interrupt) from other master(s) accessing the bus
- Output SS signal to slave
- General output

If the SPI module's SS pin is configured as input, the function is like the first option above. This SS input function is controlled from the SPI module hardware, and the SS pin must be held logic high to ensure master SPI operation. If pulled low by other master(s) using the SPI bus, the SPI module will avoid bus contention by entering slave mode and consequently not driving the SCK and MOSI lines. Entering slave mode is signaled by setting the SPI interrupt flag, generating an interrupt if enabled.

Configuring the SS pin as output enables the two last typical options, both controlled from software and not affecting the SPI module operation. The SS pin is no different than any other GPIO pins when it is configured as an output.

Often, several slaves are connected to the same bus, while the application would address one slave at a time. As illustrated in Figure 2-2, this can be done using

several GPIO pins, one for each slave or through some external logic reducing the number of pins needed.

4 SPI driver

The driver included in this application note supports both polled and interrupt-controlled SPI operation for both master and slave. The driver is intended for rapid prototyping and to get started with the XMEGA SPI module.

4.1 Files

The SPI driver source code consists of the following files:

- *spi_driver.c* – driver source file
- *spi_driver.h* – driver header file

For a complete overview of the available driver interface functions and their use, please refer to the source code documentation.

5 Code examples

Two code examples are included that show how to use the SPI drivers for interrupt-driven and polled operation. In both examples, SPIC is used as master, while SPID serves as slave.

5.1 Files

The SPI code examples are contained in the following files:

- *spi_interrupt_example.c* – Example of interrupt-driven SPI operation
- *spi_polled_example.c* – Example of polled SPI operation

5.2 Interrupt-driven

The interrupt-driven code example uses the SPI driver to instantiate a master and a slave. A data packet is then initialized and sent to the slave. When data is received at the slave, an ISR increments the data before it is put back in the data register, ready to be shifted back to the master. The master interrupt handler will transfer the number of bytes contained in the data packet. When the complete flag for the data packet is set, the master verifies that it has received the correct values.

5.3 Polled

The polled code example uses the SPI driver to instantiate a master and slave. The example is split into two sections:

First, the master sends one byte at the time to the slave, the slave increments the byte and puts it in the shift register. The master sends a dummy byte, in order to fetch the byte from the slave. The master then verifies that it has received the correct value.

In the second section, a data packet is initialized and sent to the slave. The slave does not manipulate the data, which is shifted back to the slave as new bytes are

transferred. When the whole packet is sent, the master verifies that the received bytes match the sent bytes (except for the last byte which is not shifted back to the master).

6 Doxygen Documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit http://www.doxygen.org. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the *readme.html* file in the source code folder.

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 Request
www.atmel.com/literature