AVR32105: Master and Slave SPI Driver

32-bit AVR® Microcontrollers

Application Note

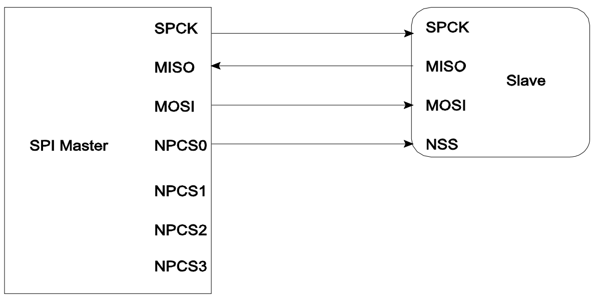
Features

- **Four chip selects with external decoder support allow communication with up to 15 peripherals**
- **Four chip select registers allowing up to four different slave setups in master mode**
- **Supports a wide range of devices**
  - **Serial memories, such as DataFlash® and 3-wire EEPROMs**
  - **Serial peripherals, such as ADCs, DACs, LCD controllers, CAN controllers and Sensors**
  - **External Co-processors**
- **Master or slave serial peripheral bus interface**
  - **8- to 16-bit programmable data length per chip select**
  - **Programmable phase and polarity per chip select**
  - **Programmable transfer delay between consecutive transfers and between clock and data per chip select**
  - **Programmable delay between consecutive transfers**
  - **Selectable Mode Fault detection**
- **Connection to PDC Channel capabilities optimizes data transfers**
  - **One channel for the receiver, one channel for the transmitter**
  - **Next buffer support**

1 Introduction

A Serial Peripheral Interface (SPI) bus is a synchronous serial data link capable of full-duplex communication with external devices in master or slave mode.

Figure 1-1. **Example SPI application**



The SPI can be used for various purposes from communicating with peripheral to other processors. SPI transfer is built up of one master and one to several slaves, each slave addressable by using a dedicated chip select line. There can only be one master at a time, but the SPI controllers can take turn being masters (multiple master protocol).

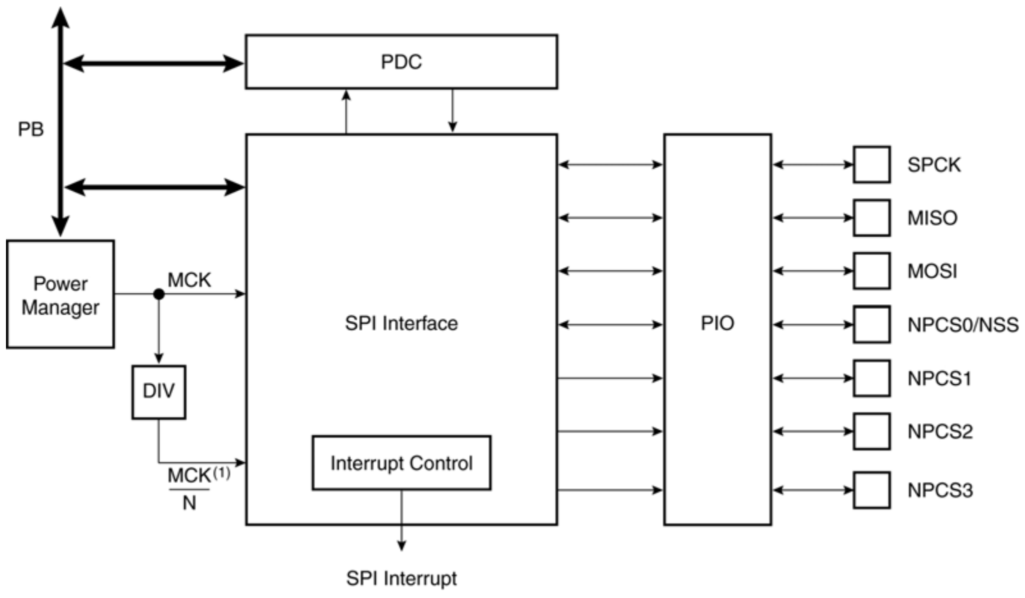
Rev. 32017A-AVR32-05/06

2 Functional description

2.1 Electrical interconnection

The SPI bus consists of two data lines, a clock line, and a number of slave select control lines. One of the data lines, MISO (master in, slave out), is used for transmission from the slave to the master, the other, MOSI (master out, slave in), is used for transmission from the master to the slave. Thus the communication is full duplex. The master also controls the clock line, SPCK, and the slave select control lines, NPCSn. When the master wishes to communicate with one of the slaves, the slave select line of that slave is driven low.

Figure 2-1. Block diagram



1. N = 32

2.2 SPI registers

Each SPI module is memory mapped to a base address, with the registers accessible by an offset from the base. Se Table 2-1 for an example from AP7000 SPI module.

Table 2-1. SPI register mapping for AP7000

Offset	Register	Register name	Access	Reset
0x00	Control register	CR	Write-only	---
0x04	Mode register	MR	Read/write	0x0
0x08	Receive data register	RDR	Read-only	0x0
0x0C	Transmit data register	TDR	Write-only	0x0
0x10	Status register	SR	Read-only	0x000000F0
0x14	Interrupt enable register	IER	Write-only	---
0x18	Interrupt disable register	IDR	Write-only	---
0x1C	Interrupt mask register	IMR	Read-only	0x0
0x20-0x2C	Reserved			
0x30	Chip select register 0	CSR0	Read/write	0x0
0x34	Chip select register 1	CSR1	Read/write	0x0
0x38	Chip select register 2	CSR2	Read/write	0x0
0x3C	Chip select register 3	CSR3	Read/write	0x0
0x40-0xF8	Reserved			
0xFC	Version register	VERSION	Read-only	
0x100-0x124	PDC registers			

2.3 Modes of operation

The SPI controller has two modes of operation, master or slave mode. This is selected with the MSTR bit in the mode register. In master mode the SPI controls the communication between the master and the slaves, while in slave mode the slave is enabled by the chip select pin is pulled low.

See Table 2-2 for a description of port directions in the different modes.

Table 2-2. Signal description

Pin name	Pin description	Type	
		Master	Slave
MISO	Master In Slave Out	Input	Output
MOSI	Master Out Slave In	Output	Input
SPCK	Serial Clock	Output	Input
NPCS1-NPCS3	Peripheral Chip Select	Output	Unused
NPCS0/NSS	Peripheral Chip Select/Slave Select	Output	Input

Figure 2-2 shows an application block diagram where a master drives multiple slaves.

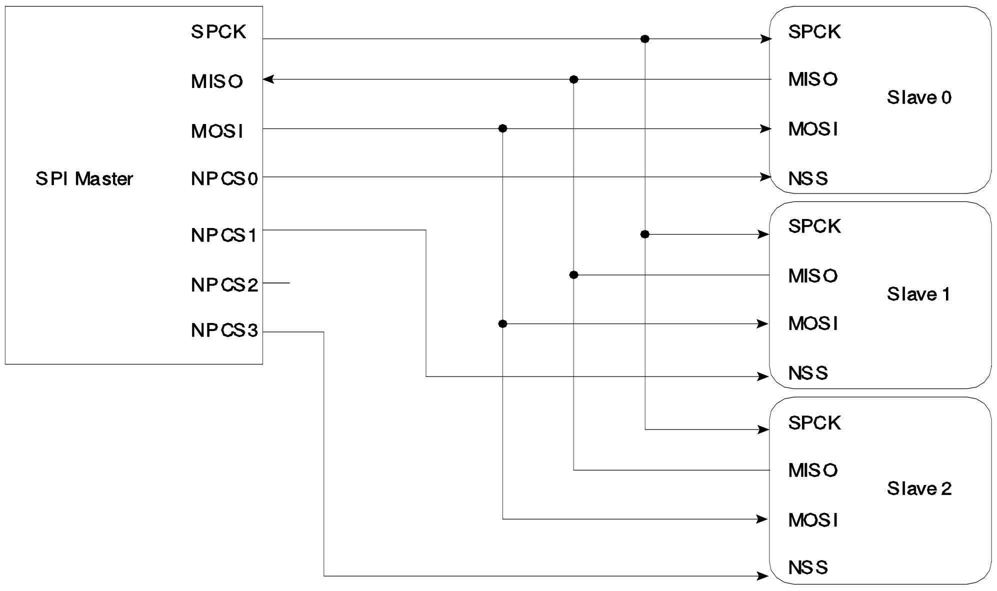


Figure 2-2. Application block diagram

For more details about master mode see chapter 2.5 page 6, and for slave mode see chapter 2.6 page 8.

2.4 Transfer format

The SPI transfers data in blocks, ranging from 8 to 16 bits. There is no acknowledgement or error control on the transfers. Transfers and receives occur simultaneously, so when a block is shifted out (transferred) a block is shifted in (received), allowing full duplex transfers.

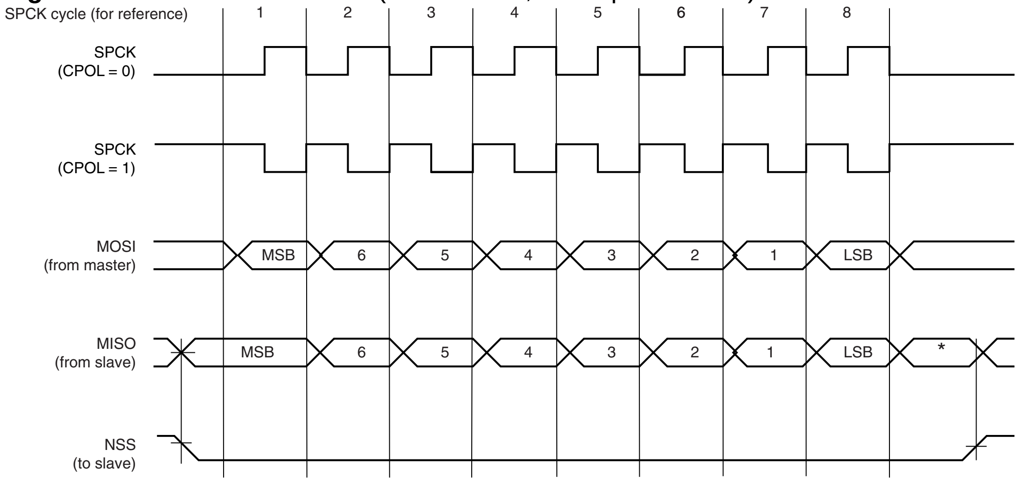
The transfer signal can be programmed in four different modes by changing clock phase and clock polarity. Changing the CPOL bit for polarity and NCPHA bit for phase in the chip select register (SCR*n*) does this. The master and the slave must have the same polarity and phase to be able to communicate.

The table below shows the four modes and corresponding parameter settings. The figures on page 5 shows examples of data transfer modes.

Table 2-3. SPI Bus Protocol Mode

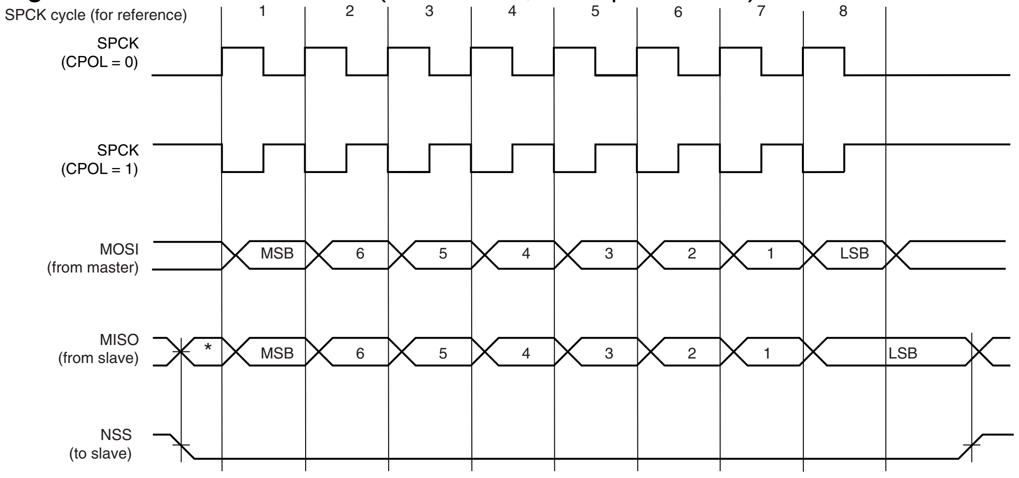
SPI mode	CPOL	NCPHA	Leading edge	Trailing edge
0	0	1	Sample (rising)	Setup (falling)
1	0	0	Setup (rising)	Sample (falling)
2	1	1	Sample (falling)	Setup (rising)
3	1	0	Setup (falling)	Sample (rising)

Figure 2-3. SPI transfer format (NCPHA = 1, 8 bits per transfer)



* Not defined, but normally MSB of previous character received.

Figure 2-4. SPI transfer format (NCPHA = 0, 8 bits per transfer)



* Not defined but normally LSB of previous character transmitted.

2.5 Master mode operations

When in master mode the SPI drives the SPCK pin, the MOSI pin is wired to the transmitter output and the MISO pin is wired to the receiver input. NPCS0 to NPCS3 is used as outputs to enable the slave devices.

The baud rate is programmable in the SPI controller by setting the SCBR field in the chip select register (CSR*n*) for the slave device the master is communicating with.

Data is transferred to the slaves by pulling a chip select pin (NPCS*n*) for a slave low and shift data out of the shift register. The master will while shifting out data sample the data from the MISO pin into the shift register.

Selecting slave devices can be done in two ways; either fixed peripheral select mode or variable peripheral mode is used. This is selected with the PS field in the mode register (MR). If variable mode is used each transfer has to specify which slave is being addressed, while fixed mode sets the chip select in the mode register (MR) field PCS.

Data transfers are initialized by writing data to the transmit data register (TDR). After the transfer is complete the received data will be available in the receive data register (RDR). Transfer only start if a valid chip select is set.

Several status bits are available in the status register (SR):

- Transmit data register empty (TDRE) is cleared when there is data in TDR; this bit is used when PDC is enabled.
- Transmission registers empty (TXEMPTY) is cleared when there is data in TDR, the shift register or when the SPI is delaying after a transfer.
- Receive data register full (RDRF) is set when there is data available in RDR.
- Overrun error status (OVRES) is set when the RDR was load twice since the last read of RDR.
- Mode fault error (MODF) is set when a mode fault is discovered while transferring data, for more details see chapter 2.5.4 on page 8.

2.5.1 Clock generation

The SPI clock to the module can use the peripheral clock as main clock (MCK) directly or it can divide this clock by 32; this is done with the mode register bit FDIV. Further the SPI can divide the main clock into a wide range of values by setting the SCBR field in the chip select register (CSR*n*). The SCBR field can be a value between 2 and 255, and is set individual for each of the four chip selects.

This allows a maximum operating baud rate at up to MCK/2 and a minimum operating baud rate of MCK/(32 * 255). The highest baud rate is MCK/2 because programming the divider to 0 is forbidden, and will lead to unpredictable results.

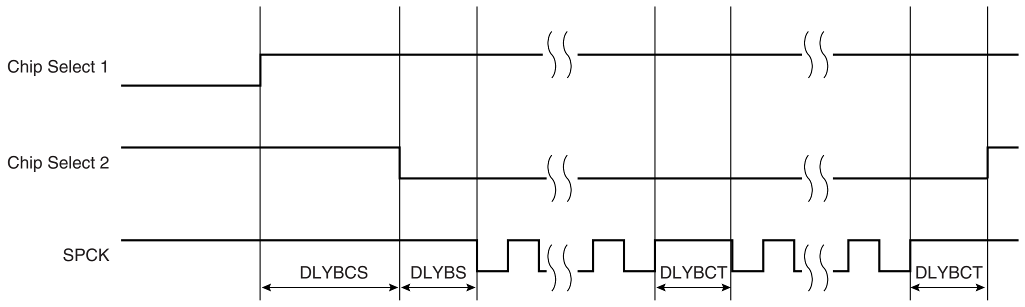
2.5.2 Transfer delays

Three delays can be programmed to modify the transfer waveforms:

- The delay between a chip disable and the next chip select, programmable only once for all the chip selects by writing the DLYBCS field in the mode register. Allows insertion of a delay between release of one chip select and before assertion of a new one.
- The delay before the SPI clock, SPCK, is sourced to a slave, independently programmable for each chip select by writing the field DLYBS. Allows the start of SPCK to be delayed after the chip select has been asserted.
- The delay between two consecutive transfers, independently programmable for each chip select by writing DLYBCT field. Allows insertion of a delay between two transfers occurring on the same chip select.

These delays allow the SPI to be adapted to the interfaced peripheral and their speed and bus release time.

Figure 2-5. Programmable delays



2.5.3 Peripheral selection

When operating in master mode the SPI can select slaves by pulling the NPCS*n* pins to ground. By default, all the chip select pins are high until a transfer begins. The SPI can be programmed to have a fixed chip select or a variable chip select. This is programmed with the PS bit in the mode register (MR). The SPI can also be programmed to use an external chip select decoder by programming the PCSDEC bit in the mode register (MR).

The SPI use the corresponding chip select register (CSR*n*) when data is transferred to a slave. For example if chip select 2 is programmed into the PCS field the setup defined in CSR2 will be used.

2.5.3.1 Fixed chip select

When fixed peripheral select is programmed in the SPI mode register (MR) bit PS, the slave is selected by setting the PCS field in MR. When entering data into the TDR the PCS field in TDR is ignored and the PCS field in the MR is used.

It is possible for the programmer to let the chip select signal be low during an entire transfer by setting the CSAAT bit in the chip select register (CSR*n*). The programmer can then set the LASTXFER bit in the transmit data register (TDR) when the last transfer to the slave is written to TDR.

2.5.3.2 Variable chip select

When variable peripheral select is programmed in the SPI mode register (MR) bit PS, the slave is selected by setting the PCS field each time data is written to the transmit data register (TDR).

2.5.3.3 Peripheral select decoding

The SPI can be set in a mode to operate up to 15 slaves by using external logic on the four chip select lines. Enabling this mode is done with the PCSDAC bit in the mode register (MR).

When operating without decoding the SPI have features allowing only one chip select to be active at any time. For example if several bits are set low in PCS field, only the lowest chip select is driven low.

When operating with decoding the SPI activates the chip select pins given by the PCS field, allowing up to 15 slaves to be accessed, because the default value 0xF is used when no slaves are activated.

The chip select registers now handle a group of chip select signals, since there is only four chip select registers. The programmer has to take care to only connect compatible slaves to each group.

The chip select registers are connected to the following chip select values:

- CSR0 handles PCS values 0 to 3
- CSR1 handles PCS values 4 to 7
- CSR2 handles PCS values 8 to 11
- CSR3 handles PCS values 12 to 14

2.5.4 Mode fault detection

The SPI has a capability to detect mode faults, which is turned on by default. Mode fault is triggered when the SPI is in master mode and an external device drives the NPCS0 signal low.

When a mode fault is detected, the MODF bit in the system register (SR) is set until the system register (SR) is read and the SPI is automatically disabled until re-enabled by writing the SPIEN bit in the control register (CR).

It is considered good design to connect a pull up resistor or use the internal pull up if available on the NPCS0 line. This will not trigger false fault error caused by noise on the NPCS0 line.

The mode fault detection is programmable by changing the MODFDIS bit in the mode register (MR).

2.6 Slave mode operations

When in slave mode the SPI uses NPCS0 as input for triggering chip select. The MISO pin is wired to the transmitter output and the MOSI pin is wired to the receiver input. SPCK is driven by the master and used to synchronize transmission of data.

A transfer is trigger by the chip select (NSS) pin driven low. When clock is received on the SPCK pin the data in the shift register is shifted out on the MISO pin. At the same time data received on the MOSI pin is sampled into the shift register. When a transfer is complete the data from the receive shift register is moved into the receive hold register (RHR) and appropriate status bits are set in the status register (SR).

If the transmit data register (TDR) is not updated since the last receive, the SPI will shift out the last received data on the MISO pin. If the SPI has not received any data since a reset, zeros will be transferred to the master.

If the transmit data register (TDR) is updated the data will be loaded immediately into the shift register, ready to be transmitted when the chip select line is driven low and SPCK starts. If new data is written to TDR it will remain in TDR until the data in the shift register has been transferred. This allows fast updates for single transfers.

3 Implementation

3.1 Driver files

The driver consists of two files “spi.c” and “spi.h”. Where “spi.h” declares all functions and “spi.c” contains the source code. The only change needed in the driver is specifying which device the driver is to be targeted.

Target is specified at the top in “spi.h”.

3.1.1 Master mode

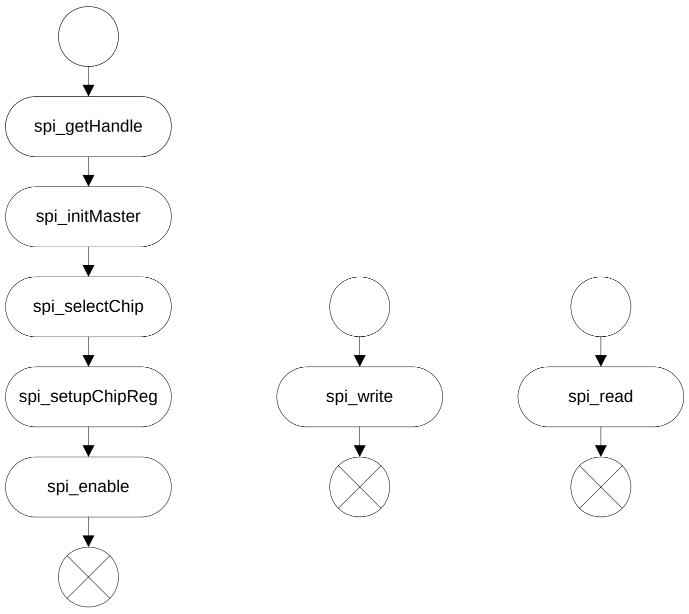
Before a transfer can start, the SPI has to be initialized. To conduct a transfer as the master, several functions have to be called. spi_initMaster() initializes a SPI in master mode. This function, like all other, need a SPI handle that tells it which SPI instance to operate on. spi_getHandle() will return the appropriate handle when called with the number as argument.

When master mode has been selected, spi_selectChip() and spi_setupChipReg() must be run. The first selects a slave chip to communicate with, while the second sets options for the communication (like baudrate, number of bits in a transferred block, SPI mode, etc.). spi_selectionMode() could also be called if the way slaves are selected should be changed. When everything is correctly set up, spi_enable() should be called.

Figure 3-1 shows the flow an example initialization and usage of the SPI in master mode. The spi_getHandle() function call is not mandatory, but recommended for more portable code.

After the SPI is initialized, the functions spi_write() and spi_read() can be used. The driver is implemented by polling the SPI, thus both spi_write() and spi_read() function has an active waiting timeout.

Figure 3-1. Master mode example flow chart



3.1.2 Slave mode

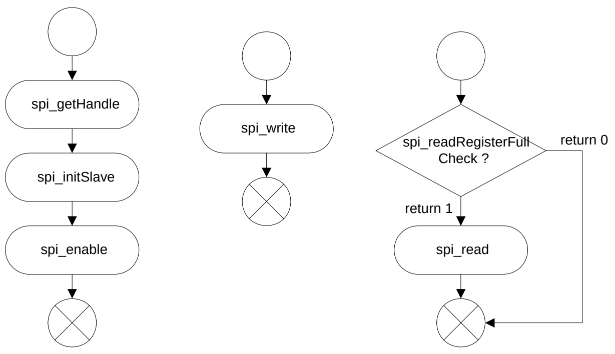
As the master will control most aspects of a transmission, a SPI in slave mode will not take many options. spi_initSlave() and spi_enable() are the only functions needed to initialize the SPI.

Figure 3-2 shows the flow an example initialization and usage of the SPI in slave mode. The spi_getHandle() function call is not mandatory, but recommended for more portable code.

After the SPI is initialized, the functions spi_write() and spi_read() can be used. The driver is implemented by polling the SPI, thus both spi_write() and spi_read() function has an active waiting timeout.

The function spi_readRegisterFullCheck() can be used for checking if there is new data in the receive register. This call can then again be used in an active wait loop.

Figure 3-2. Slave mode example flow chart



3.2 Example code

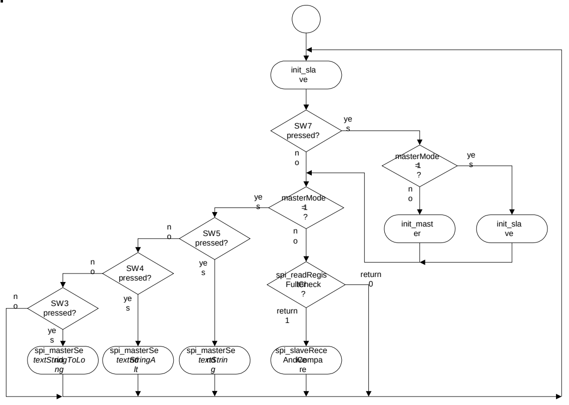
The example application does a communication between two SPI capable devices. There should be one master and one slave. The master will then send a text string to the slave device, which it receives and compares with an expected string.

The STK1000 development kit is used to be able to output debug information on LEDs and get input from the user by using the switches.

Figure 3-3 shows the flow of the example application. The application is implemented by polled function calls to make it less dependable of other modules.

For detailed explanation please see the Doxygen documentation.

Figure 3-3. Example code flow chart



3.3 Doxygen documentation

All source code is prepared for doxygen automatic documentation generation. Premade doxygen documentation is also supplied with the source to this application note, located in src/doxygen/index.html.

Doxygen is a tool for generating documentation from source code by analyzing the source code and using known keywords. For more details see http://www.stack.nl/~dimitri/doxygen/.

4 Further reading

4.1 Peripheral DMA Controller (PDC)

The SPI bus is able to transfer data at a fairly high rate. Thus the one-block buffers will be filled/emptied very often, which again translates to many interrupts to the CPU. As a result the SPI module would take a large share of the CPU time, at the cost of other tasks.

The solution to this problem is to use the peripheral DMA controller, PDC. This controller can be given a transmit buffer with data to transmit and a receive buffer to put received data. In addition, it can be given pointers to a second transmit and receive buffer, and it will automatically switch buffers when it need to. Depending on how the SPI is configured, interrupts to the CPU can be sent when first TX buffer is empty, first RX buffer is full, when both TX buffers are empty, and/or when both RX buffers are full. The per block transferred interrupts are avoided, and only a fraction of the CPU time is used on the transfer.

For more information about the peripheral DMA controller see application note *AVR®32108 - Peripheral Direct Memory Access Driver*.

4.2 Interrupt

The SPI interface has an interrupt line connected to the interrupt controller (IC). Handling the SPI interrupt requires programming the IC before configuring the SPI.

For more information and details about the interrupt controller see application note *AVR32101 – The AVR32 Interrupt Controller*.

Atmel Corporation

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

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

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

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

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature