TB3215

Getting Started with Serial Peripheral Interface (SPI)

Introduction

Author: Alin Stoicescu, Microchip Technology Inc.

This technical brief provides information about Serial Peripheral Interface (SPI) on tinyAVR® 0- and 1-series, megaAVR® 0-series, and AVR® DA devices, and intends to familiarize the user with AVR microcontrollers. The document describes the application area, the modes of operation, and the hardware and software requirements of the SPI.

Throughout the document, the configuration of the peripheral will be described in detail, starting with the location of the SPI pins, the direction of the pins, how to initialize the device as a host or a client, and how to exchange data inside the system. This document covers the following use cases:

- **Sending Data as a Host SPI Device:**
  The device will be configured as a host, will control the client, and will send data using a method called polling.
- **Receiving Data as a Client SPI Device:**
  The device will be configured as a client and will wait for the incoming data. The data reception will be triggered by interrupts.
- **Changing Data Transfer Type:**
  The device will be configured as a host and will send data with respect to the clock polarity and the clock phase.

Note: For each use case described in this document, there are two code examples: One bare metal developed on ATmega4809, and one generated with MPLAB® Code Configurator (MCC) developed on AVR128DA48.

View the ATmega4809 Code Examples on GitHub
Click to browse repository

View the AVR128DA48 Code Examples on GitHub
Click to browse repository

Table of Contents

1. Relevant Devices

This section lists the relevant devices for this document. The following figures show the different family devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features. Downward migration on tinyAVR® 1-series devices may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and, therefore, the available features
- Devices with different Flash memory sizes typically also have different SRAM and EEPROM

Figure 1-1. tinyAVR® 0-series Overview

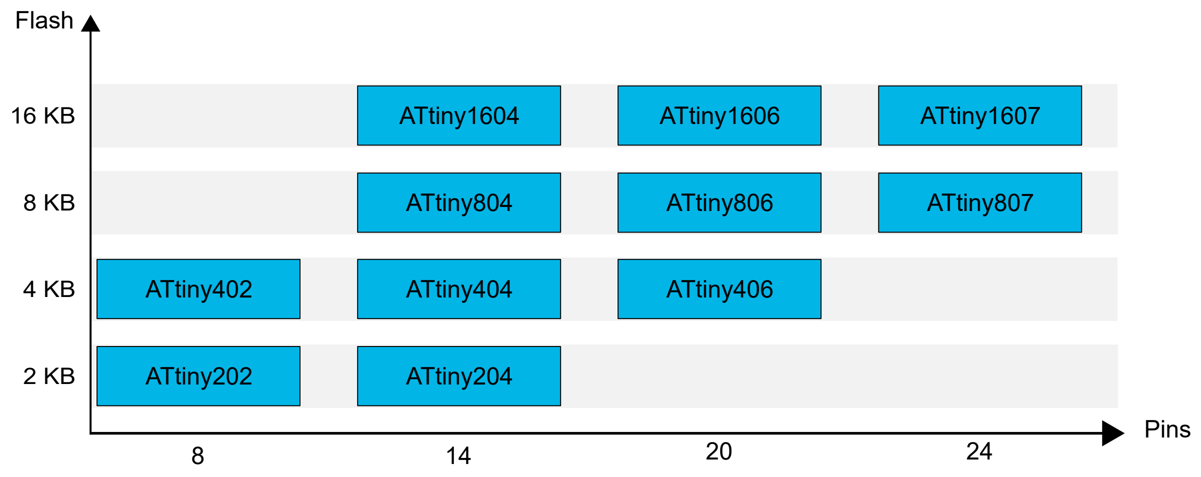


Figure 1-2. tinyAVR® 1-series Overview

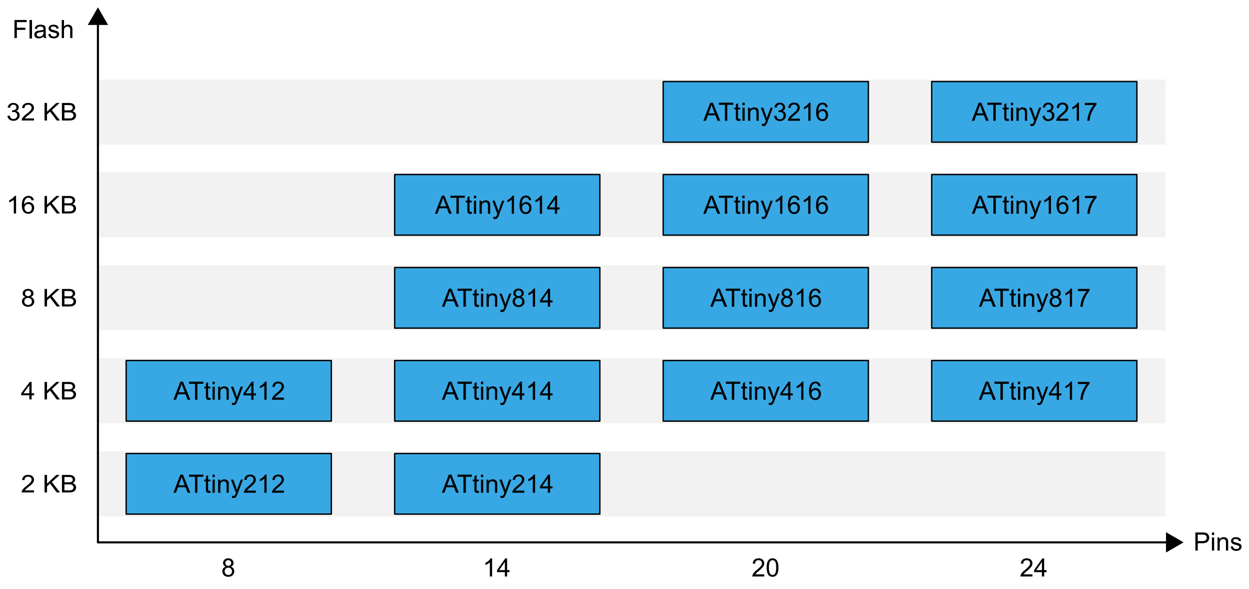


Figure 1-3. tinyAVR® 2 Family Overview

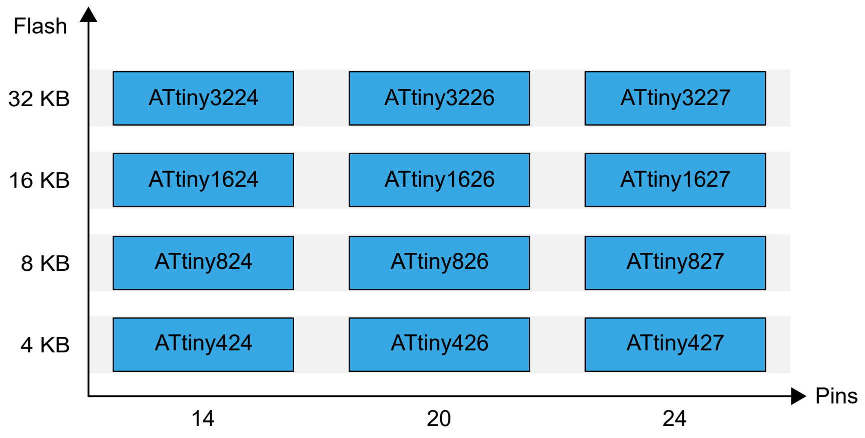


Figure 1-4. megaAVR® 0-series Overview

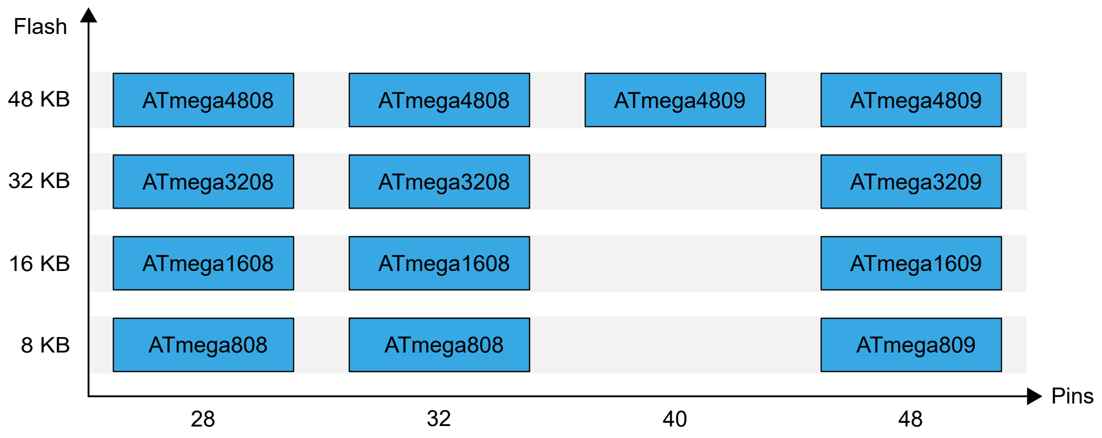


Figure 1-5. AVR® DA Family Overview

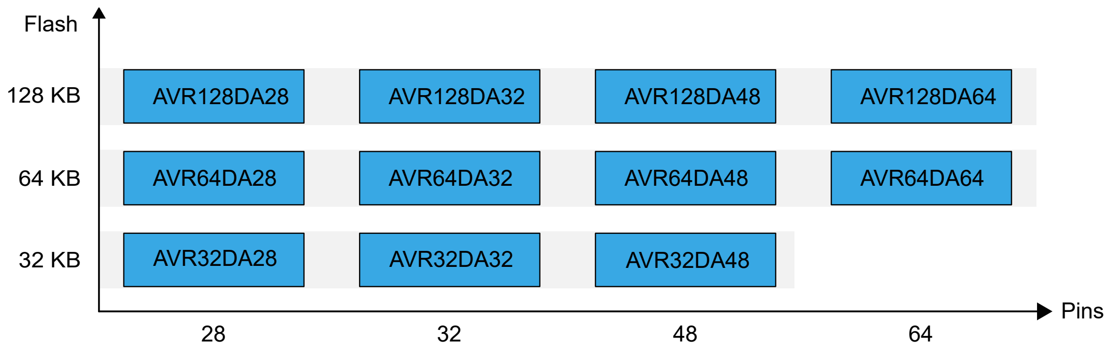
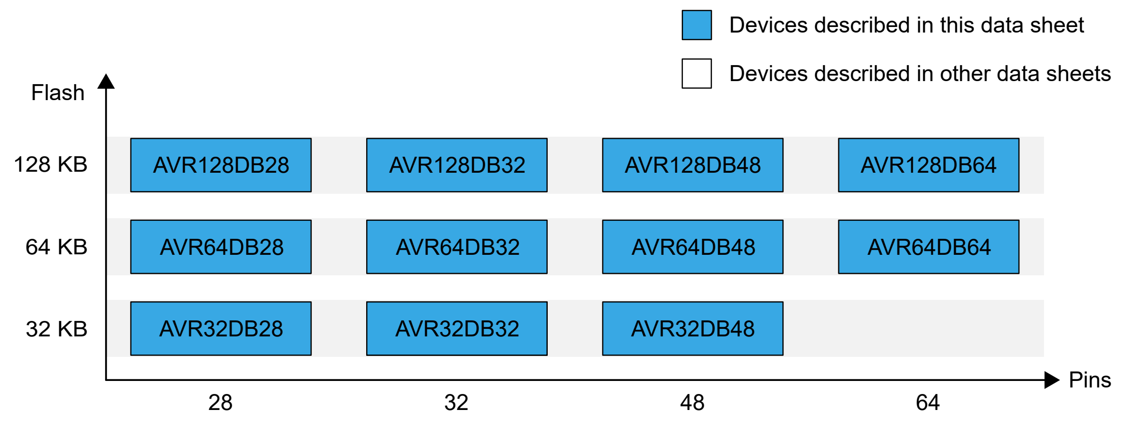


Figure 1-6. AVR® DB Family Overview



2. Overview

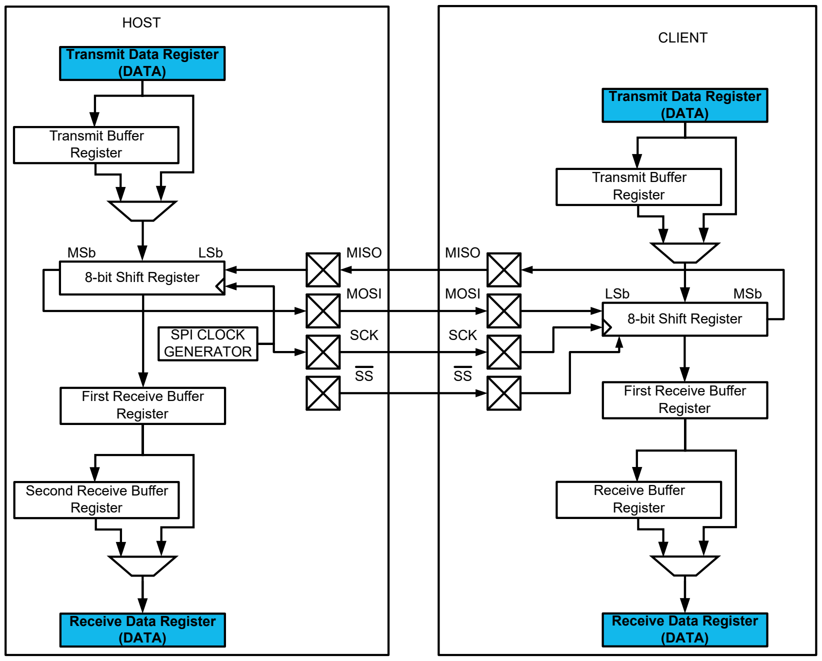
The SPI bus is a synchronous serial communication interface based on four types of logic signals:

- SCK: Serial Clock (output from the host)
- MOSI: Host Output Client Input (data output from the host)
- MISO: Host Input Client Output (data output from the client)
- $\overline{\text{SS}}$: Client Select (active-low, data output from the host)

This peripheral is used for short distance and high-speed communication, primarily in embedded systems. The SPI devices communicate in Full-Duplex mode, using a channel for transmitting and one for receiving data. The SPI is based on a host-client architecture with a single host at a time and one or more clients. The host device is the only one that can generate a clock, thus it is the initiator of the data exchange. The SPI host device uses the same SCK, MOSI and MISO channels for all the clients, but usually individual lines of $\overline{\text{SS}}$ for each of the clients. The host device selects the desired client by pulling the $\overline{\text{SS}}$ signal low.

The data to be sent will be stored in either a data register or, if a transmission is already in progress and the Buffer mode was activated, in a buffer register. The data are sent out serially on the MOSI channel, using a shift register, and every bit is synchronized using the SPI clock generator. While every bit is shifted out, new data are received on the MISO channel from the client and are shifted in a receiver buffer and further in the receive DATA register. If the receiver is busy, meaning there are already data in the receive DATA register and the Buffer mode was activated, the data will be temporarily stored in a second receiver buffer. The Buffer mode is activated by setting high the BUFEN bit of the CTRLB register.

Figure 2-1. SPI Block Diagram


HOST
Transmit Data Register (DATA)
Transmit Buffer Register
MSb
LSb
8-bit Shift Register
SPI CLOCK GENERATOR
First Receive Buffer Register
Second Receive Buffer Register
Receive Data Register (DATA)
MISO
MOSI
SCK
SS
CLIENT
Transmit Data Register (DATA)
Transmit Buffer Register
MISO
MOSI
SCK
SS
LSb
MSb
8-bit Shift Register
First Receive Buffer Register
Receive Buffer Register
Receive Data Register (DATA)

The SPI module has five registers. One register is used for data transfer and storage, two registers are used for Interrupt flags, and the last two registers (CTRLA and CTRLB) are for initializations. All the configurations required to make the peripheral work correctly are reduced to changing some bits in the CTRLA register, while the CTRLB register is focused on different modes of operation that are optional. More details regarding the registers can be found in the family data sheet of the device, on the register summary of the peripheral section.

Figure 2-2. Register Summary - SPIn

Offset	Name	Bit Pos.								
0x00	CTRLA	7:0		DORD	MASTER	CLK2X		PRESC[1:0]		ENABLE
0x01	CTRLB	7:0	BUFEN	BUFWR				SSD	MODE[1:0]	
0x02	INTCTRL	7:0	RXCIE	TXCIE	DREIE	SSIE				IE
0x03	INTFLAGS	7:0	IF	WRCOL						
0x03	INTFLAGS	7:0	RXCIF	TXCIF	DREIF	SSIF				BUFOVF
0x04	DATA	7:0	DATA[7:0]							

3. Sending Data as a Host SPI Device

The host is the device that decides when to trigger communication and which client is the recipient of the message. SPI host devices are generally used in high-speed communication and the focus is to exchange data with other devices acting as clients (e.g. sensors, memories or other MCUs).

This use case follows the steps:

- Configure the location of the SPI pins
- Initialize the peripheral
- Configure the direction of the pins
- Control client devices
- Send data as a host device

How to Configure the Location of the SPI Pins

The way to configure the location of the pins is independent of the application purpose and the SPI mode. Each microcontroller has its own default physical pin position for peripherals. The locations can be found in the PORTMUX peripheral section from the family data sheet of the megaAVR 0-series. For ATmega4809, the SPI pins are located on PA[7:4] and can be changed on PC[3:0] or PE[3:0] using the TWISPIROUTEA register of the PORTMUX module.

Figure 3-1. TWISPIROUTEA Register

Bit	7	6	5	4	3	2	1	0
			TWI0[1:0]				SPI0[1:0]	
Access			R/W	R/W			R/W	R/W
Reset			0	0			0	0

The pin order is the following: MOSI, MISO, SCK, $\overline{SS}$; MOSI representing the lowest pin number from the group. This is how a user can change the location of the SPI pins for option 1 with port C:

Value	Name	Description
0x0	DEFAULT	SPI on PA[7:4]
0x1	ALT1	SPI on PC[3:0]
0x2	ALT2	SPI on PE[3:0]
0x3	NONE	Not connected to any pins

This translates into the following code:

```
PORTMUX.TWISPIROUTEA |= PORTMUX_SPI00_bm;
```

Or option 2 with port E:

Value	Name	Description
0x0	DEFAULT	SPI on PA[7:4]
0x1	ALT1	SPI on PC[3:0]
0x2	ALT2	SPI on PE[3:0]
0x3	NONE	Not connected to any pins

This translates into the following code:

```
PORTMUX.TWISPIROUTEA |= PORTMUX_SPI01_bm;
```

How to Initialize the Peripheral

The clock frequency is derived from the main clock of the microcontroller and is reduced using a prescaler or divider circuit present in the SPI hardware. By default, the source of the main clock is a 20 MHz internal oscillator, which is divided by a prescaler whose default value is 6, resulting in a main clock frequency of approximately 3.33 MHz. More information about the clock can be found in the Clock Controller section of the megaAVR 0-series family data sheet.

The clock frequency of the SPI can also be increased using the Double Clock mode, which works only in Host mode. The Data Order bit represents the endianness (Most Significant bit or Least Significant bit) of the data, the order in which the bits are transmitted on the channel (starting with the last or the first bit from a register). All the configurations are related to the CTRLA register.

Figure 3-2. CTRLA Register

Bit	7	6	5	4	3	2	1	0
		DORD	MASTER	CLK2X		PRESC[1:0]		ENABLE
Access		R/W	R/W	R/W		R/W	R/W	R/W
Reset		0	0	0		0	0	0

Next is an example of how to configure a host SPI device with the default main clock source and with the default pin location presented in the previous topic. A 416 kHz frequency will result by configuring the device in Double-Speed mode and with a 16 times divider. The data will be shifted out starting with the Most Significant bit (MSb):

Value	Name	Description
`0x0`	DIV4	CLK_PER/4
`0x1`	DIV16	CLK_PER/16
`0x2`	DIV64	CLK_PER/64
`0x3`	DIV128	CLK_PER/128

This translates into the following code:

```
SPI0.CTRLA = SPI_CLK2X_bm
           | SPI_DORD_bm
           | SPI_ENABLE_bm
           | SPI_MASTER_bm
           | SPI_PRESC_DIV16_gc;
```

How to Configure the Direction of the Pins

Since the host devices control and initiate transmissions, the MOSI, SCK and $\overline{\text{SS}}$ pins must be configured as output, while the MISO channel will keep its default direction as input. The default values, directions and configurations explained above are still applicable here. The following example is based on the default position of the SPI pins:

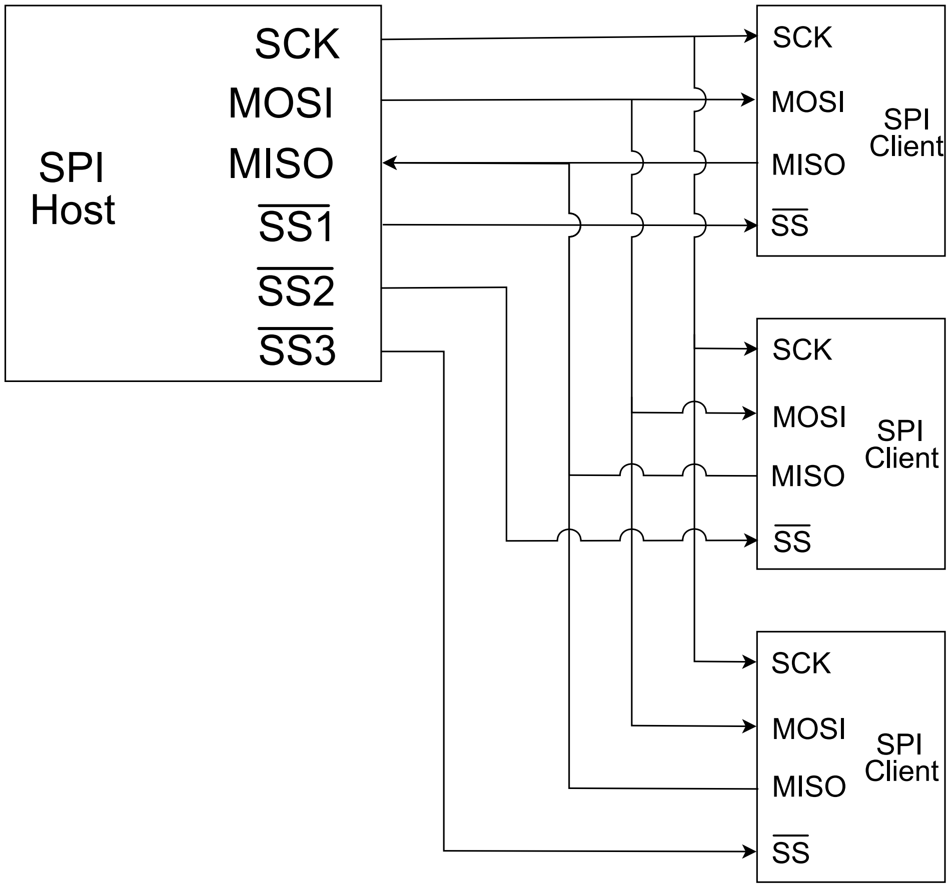
```
PORTA.DIR |= PIN4_bm;  // MOSI channel
PORTA.DIR &= ~PIN5_bm; // MISO channel
PORTA.DIR |= PIN6_bm;  // SCK channel
PORTA.DIR |= PIN7_bm;  // SS channel
```

An SPI host device can control more than one client, thus requiring more $\overline{\text{SS}}$ pins. The additional $\overline{\text{SS}}$ channels can be configured just like the one in the example above. The user must choose an unused pin and configure its direction as output.

How to Control Client Devices

A host will control a client by pulling low the $\overline{\text{SS}}$ pin. If the client has set the direction of the MISO pin to output, when the $\overline{\text{SS}}$ pin is low, the SPI driver of the client will take control of the MISO pin, shifting data out from its transmit DATA register. All client devices can receive a message, but only those with $\overline{\text{SS}}$ pin pulled low can send data back. Therefore, it is not recommended to enable more than one client in a typical connection (like the one below) because all of them will try to respond to the message and there is only one MISO channel, thus the transmission will result in a write collision. The user can check the appearance of collisions by reading the value of the WRCOL bit in the INTFLAGS register.

Figure 3-3. Typical SPI Bus



How to Send Data as a Host Device

All the settings configured before are considered in the following example and the polling method is used for flag checking. Before sending data, the user must pull low an $\overline{\text{SS}}$ signal to let the client device know it is the recipient of the message.

```
PORTA.OUT &= ~PIN7_bm;
```

Once the user writes new data into the DATA register the hardware starts a new transfer, generating the clock on the line and shifting out the bits.

Figure 3-4. DATA Register

Bit	7	6	5	4	3	2	1	0
	DATA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

```
SPI0.DATA = data;
```

When the hardware finishes shifting all the bits, it activates a receive Interrupt flag, which can be found in the INTFLAGS register.

Figure 3-5. INTFLAGS Register

Bit	7	6	5	4	3	2	1	0
	IF	WRCOL						
Access	R/W	R/W						
Reset	0	0						

The user must check the state of the flag, before writing new data in the register, by either activating the interrupts or by constantly reading the value of the flag (a method called polling), else a write collision interrupt will occur.

```
while (!(SPI0.INTFLAGS & SPI_IF_bm))
{
    ;
}
```

The user can pull the $\overline{SS}$ channel high if there is nothing left to transmit.

```
PORTA.OUT |= PIN7_bm;
```

Tip: The full code example is also available in the Appendix section.

View the ATmega4809 Code Example on GitHub
Click to browse repository

An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

4. Receiving Data as a Client SPI Device

The client devices are usually actuators. Clients do no initiate any action, they only act whenever the host initiates. A client must be always available and has to wait until the host pulls low its $\overline{SS}$ channel.

This use case follows the steps:

- Initialize the peripheral
- SPI client direction pin configuration
- Receive data as a client SPI

How to Initialize the Peripheral

The client gets its clock signal from the host device so there is no point changing the clock divider of the peripheral, a change that does not affect when in SPI Client mode. Though, the hardware peripheral has to sample the data received on the MOSI channel. For the data signal to be correctly reconstructed, the main clock frequency of the device must be at least double the clock received on the SPI SCK channel.

If the client device is a microcontroller, the user has to consider the frequency request and configure a powerful clock source. If the user does not have access to the clock generator of the client, it has to make sure the host does not exceed the limitations of the client. A host is part of the same system or application and is mainly represented by a microcontroller whose frequency can be easily changed - either SPI or main clock frequency.

To make the example easier to understand, some of the information presented in 3. Sending Data as a Host SPI Device is also applied here. The device is configured as a client, with a main clock frequency of 3.33 MHz, and the data are shifted out starting with the MSb. Configuring the device as a client resumes mainly to enabling the module and deactivating the Host (MASTER) bit from the CTRLA register:

Figure 4-1. CTRLA Register

Bit	7	6	5	4	3	2	1	0
		DORD	MASTER	CLK2X		PRESC[1:0]		ENABLE
Access		R/W	R/W	R/W		R/W	R/W	R/W
Reset		0	0	0		0	0	0

```
SPI0.CTRLA = SPI_DORD_bm
           | SPI_ENABLE_bm
           & (~SPI_MASTER_bm);
```

SPI Client Direction Pin Configuration

When the device is in SPI Client mode, the MOSI, SCK and $\overline{SS}$ pins require to be configured as input channels. By default, all Input/Output (I/O) pins are configured as input, so there is nothing that needs to be modified for these pins. Thus, the hardware circuit of the SPI will take control of these channels during transmission if the peripheral is enabled. Since it is not mandatory to send data back, the MISO channel can be configured either as output or input.

The normal mode is to configure the pin as an output, and the hardware circuit will control its behavior during data exchanges. If the pin is configured as input, it will act as an ordinary I/O pin and will not be used by the SPI.

When the pin value of the DIR register has the value 0, the pin acts as an input digital pin, respective output digital pin for value 1. The default location of the SPI pins will be considered. To be sure that the default direction value of the pins was not changed, all the required pins will be configured as follows:

```
PORTA.DIR &= ~PIN4_bm; // MOSI channel
PORTA.DIR |= PIN5_bm;  // MISO channel
PORTA.DIR &= ~PIN6_bm; // SCK channel
PORTA.DIR &= ~PIN7_bm; // SS channel
```

How to Receive Data as a Client SPI

All the client devices connected to the SPI bus will receive the message sent on the MOSI channel by the host device. A client cannot respond to a message unless the $\overline{SS}$ channel is pulled low. When the host device pulls the $\overline{SS}$

pin low, the SPI peripheral of the client device will take control of the MISO pin and will shift data out. If the user does not write into the DATA register, the client will not send data out and the peripheral will shift out a byte full of zeros.

The peripheral will signal the reception of new data by activating the IF flag of the INTFLAGS register. The user has to check the value of the bit, either by the polling method as presented in the host example or by interrupts. The following example uses interrupts to establish the value of the bit since there is no way to tell when the host will send new data and interrupts are non-blocking. Therefore, the device can do whatever it has to do during idle SPI time.

When using interrupts, three important things must be taken into consideration:

1. Activating the interrupts for the microcontroller. The macro can be used by including the `<avr/interrupt.h>` file:

```
sei();
```

2. Activating the interrupts for the peripheral can be done by activating the IE flag from the INTCTRL register:

Figure 4-2. INTCTRL Register

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	DREIE	SSIE				IE
Access	R/W	R/W	R/W	R/W				R/W
Reset	0	0	0	0				0

```
SPI0.INTCTRL = SPI_IE_bm;
```

3. Clearing the Interrupt flag, if it is not cleared automatically by the hardware. After receiving new data, the receive complete Interrupt flag will be activated. This one can be found in the INTFLAGS register.

Figure 4-3. INTFLAGS Register

Bit	7	6	5	4	3	2	1	0
	IF	WRCOL						
Access	R/W	R/W						
Reset	0	0						

Clearing the Interrupt flag is done by writing '`1`' to the bit inside the interrupt function, where the user may also insert its interrupt routine based on its application purpose.

In the example below, it is shown how to read the received data, clear the interrupt and write to the DATA register (it is the user's choice what to do with the received data and what to write back to the host).

```
ISR(SPI0_INT_vect)
{
    receiveData = SPI0.DATA;

    SPI0.DATA = writeData;

    SPI0.INTFLAGS = SPI_IF_bm;
}
```

Tip: The full code example is also available in the Appendix section.

View the ATmega4809 Code Example on GitHub
Click to browse repository

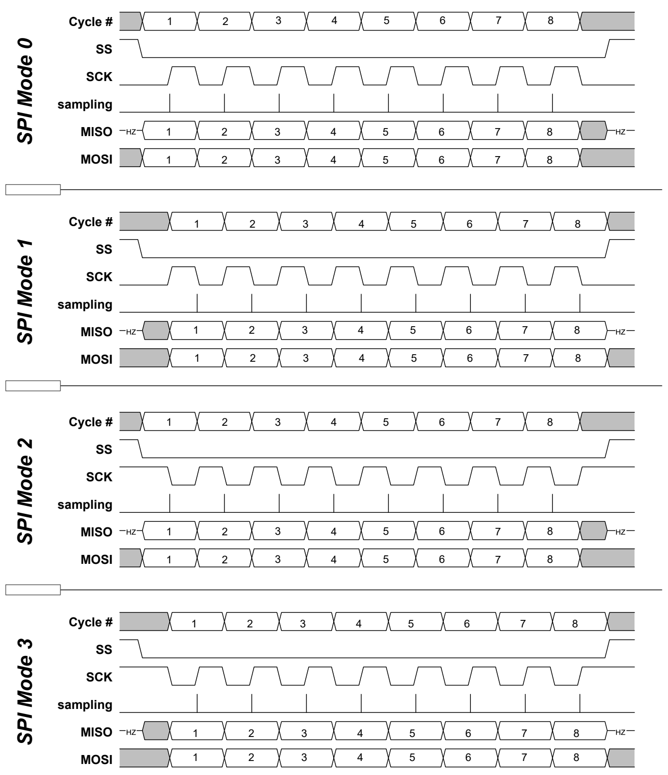
An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

5. Changing Data Transfer Type

It represents how data are transmitted concerning clock generation. The clock polarity and the clock phase are the ones important for data modes. By clock polarity, one can understand the level of the signal which can be low while in the Idle state and will start with a rising edge when transmitting data, or it can be high while in Idle state and will start with a falling when exchanging data. Depending on the phase, the data are generated or sampled concerning the clock on the channel: On a rising or a falling edge. See the figure below.

Figure 5-1. SPI Data Transfer Modes



Both the host and the client devices must be configured in the same way, so one can decode correctly what the other encoded. Data modes can be selected by changing the value of the MODE[1:0] bit field from the CTRLB register.

Figure 5-2. CTRLB Register

Bit	7	6	5	4	3	2	1	0
	BUFEN	BUFWR				SSD	MODE[1:0]	
Access	R/W	R/W				R/W	R/W	R/W
Reset	0	0				0	0	0

Until now, the examples were based on SPI Mode 0 because there was no change made to these bits and that is the default value of the bits.

Below is an example of how to configure the SPI in Data Mode 3, based on the normal/basic host SPI Initialization mode presented in 3. Sending Data as a Host SPI Device. The only difference is the change of the data transmission type:

```
SPI0.CTRLB |= SPI_MODE_3_gc;
```

Value	Name	Description
`0x0`	0	Leading edge: Rising, sample Trailing edge: Falling, setup
`0x1`	1	Leading edge: Rising, setup Trailing edge: Falling, sample
`0x2`	2	Leading edge: Falling, sample Trailing edge: Rising, setup
`0x3`	3	Leading edge: Falling, setup Trailing edge: Rising, sample

Tip: The full code example is also available in the Appendix section.

View the ATmega4809 Code Example on GitHub
Click to browse repository

An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

6. References

1. AVR128DA48 product page: www.microchip.com/wwwproducts/en/AVR128DA48
2. AVR128DA48 Curiosity Nano Evaluation Kit product page: https://www.microchip.com/Developmenttools/ProductDetails/DM164151
3. AVR128DA28/32/48/64 Data Sheet
4. Getting Started with the AVR® DA Family
5. ATmega4809 product page: www.microchip.com/wwwproducts/en/ATMEGA4809
6. megaAVR® 0-series Family Data Sheet
7. ATmega809/1609/3209/4809 – 48-Pin Data Sheet megaAVR® 0-series
8. ATmega4809 Xplained Pro product page: https://www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro

7. Appendix

Example 7-1. Sending Data as a Host SPI Device Full Code Example

```c
#include <avr/io.h>

void SPI0_init(void);
void clientSelect(void);
void clientDeselect(void);
uint8_t SPI0_exchangeData(uint8_t data);

void SPI0_init(void)
{
    PORTA.DIR |= PIN4_bm;  /* Set MOSI pin direction to output */
    PORTA.DIR &= ~PIN5_bm; /* Set MISO pin direction to input */
    PORTA.DIR |= PIN6_bm;  /* Set SCK pin direction to output */
    PORTA.DIR |= PIN7_bm;  /* Set SS pin direction to output */

    SPI0.CTRLA = SPI_CLK2X_bm           /* Enable double-speed */
               | SPI_DORD_bm            /* LSB is transmitted first */
               | SPI_ENABLE_bm          /* Enable module */
               | SPI_MASTER_bm          /* SPI module in Host mode */
               | SPI_PRESC_DIV16_gc;    /* System Clock divided by 16 */
}

uint8_t SPI0_exchangeData(uint8_t data)
{
    SPI0.DATA = data;

    while (!(SPI0.INTFLAGS & SPI_IF_bm))  /* Waits until data are exchanged*/
    {
        ;
    }

    return SPI0.DATA;
}

void clientSelect(void)
{
    PORTA.OUT &= ~PIN7_bm; // Set SS pin value to LOW
}

void clientDeselect(void)
{
    PORTA.OUT |= PIN7_bm;  // Set SS pin value to HIGH
}

int main(void)
{
    uint8_t data = 0;

    SPI0_init();

    while (1)
    {
        clientSelect();
        SPI0_exchangeData(data);
        clientDeselect();
    }
}
```

Example 7-2. Receiving Data as a Client SPI Device Full Code Example

```
#include <avr/io.h>
#include <avr/interrupt.h>

void SPI0_init(void);

volatile uint8_t receiveData = 0;
volatile uint8_t writeData = 0;

void SPI0_init(void)
{
    PORTA.DIR &= ~PIN4_bm; /* Set MOSI pin direction to input */
    PORTA.DIR |= PIN5_bm;  /* Set MISO pin direction to output */
    PORTA.DIR &= ~PIN6_bm; /* Set SCK pin direction to input */
    PORTA.DIR &= ~PIN7_bm; /* Set SS pin direction to input */

    SPI0.CTRLA = SPI_DORD_bm        /* LSB is transmitted first */
               | SPI_ENABLE_bm      /* Enable module */
               & (~SPI_MASTER_bm);  /* SPI module in Client mode */

    SPI0.INTCTRL = SPI_IE_bm;       /* SPI Interrupt enable */
}

ISR(SPI0_INT_vect)
{
    receiveData = SPI0.DATA;

    SPI0.DATA = writeData;

    SPI0.INTFLAGS = SPI_IF_bm; /* Clear the Interrupt flag by writing 1 */
}

int main(void)
{
    SPI0_init();

    sei(); /* Enable Global Interrupts */

    while (1)
    {
        ;
    }
}
```

Example 7-3. Changing Data Type Full Code Example

```
#include <avr/io.h>

void SPI0_init(void);
void clientSelect(void);
void clientDeselect(void);
uint8_t SPI0_exchangeData(uint8_t data);

void SPI0_init(void)
{
    PORTA.DIR |= PIN4_bm;  /* Set MOSI pin direction to output */
    PORTA.DIR &= ~PIN5_bm; /* Set MISO pin direction to input */
    PORTA.DIR |= PIN6_bm;  /* Set SCK pin direction to output */
    PORTA.DIR |= PIN7_bm;  /* Set SS pin direction to output */

    SPI0.CTRLA = SPI_CLK2X_bm            /* Enable double-speed */
               | SPI_DORD_bm             /* LSB is transmitted first */
               | SPI_ENABLE_bm           /* Enable module */
               | SPI_MASTER_bm           /* SPI module in Host mode */
               | SPI_PRESC_DIV16_gc;     /* System Clock divided by 16 */

    SPI0.CTRLB |= SPI_MODE_3_gc; /* Data Mode 3 */
}

uint8_t SPI0_exchangeData(uint8_t data)
{
    SPI0.DATA = data;

    while (!(SPI0.INTFLAGS & SPI_IF_bm))   /* Waits until data are exchanged*/
    {
        ;
    }

    return SPI0.DATA;
}

void clientSelect(void)
{
    PORTA.OUT &= ~PIN7_bm; // Set SS pin value to LOW
}

void clientDeselect(void)
{
    PORTA.OUT |= PIN7_bm;  // Set SS pin value to HIGH
}

int main(void)
{
    uint8_t data = 0;

    SPI0_init();

    while (1)
    {
        clientSelect();
        SPI0_exchangeData(data);
        clientDeselect();
    }
}
```

8. Revision History

Document Revision	Date	Comments
C	12/2021	Added AVR® DB and tinyAVR® 2 to relevant devices section
B	02/2021	Updated the GitHub repository links, the *References* section, and the use cases sections. Added the *AVR® DA Family Overview* and *Revision History* sections. Added MCC versions for each use case, running on AVR128DA48. Other minor corrections.
A	05/2019	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Legal Notice

Trademarks

ISBN: 978-1-5224-9287-0

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820