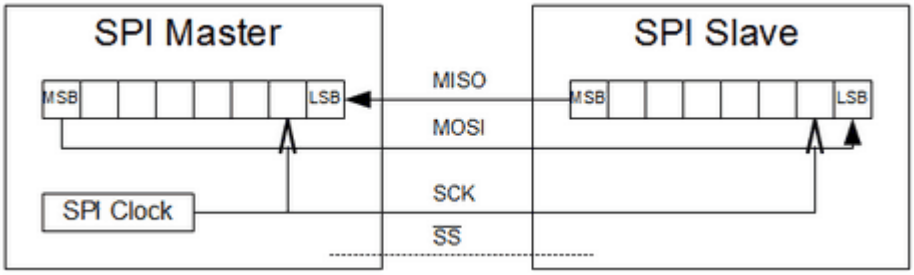
EMBEDDS

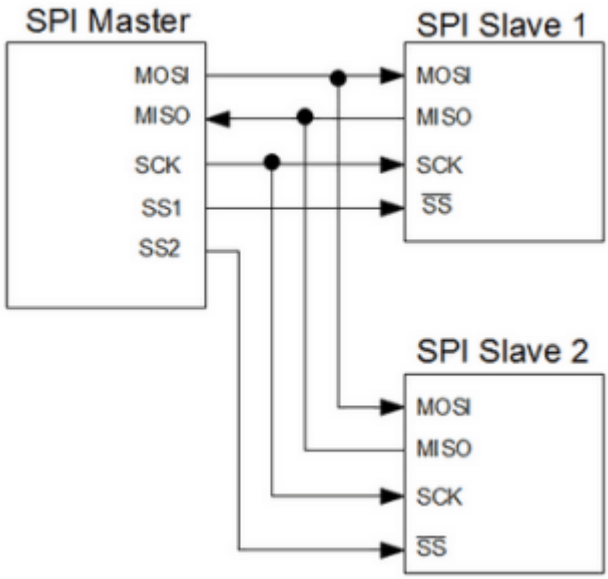
Serial peripheral interface in AVR microcontrollers

BY ADMIN AVR TUTORIAL

Serial Peripheral Interface (SPI) is the fastest synchronous communication interface allowing data transfer speeds up to half of the core clock. If the AVR microcontroller is clocked at 16MHz, then the SPI clock may reach 8MHz in master mode. SPI communication interface is a standard way to talk to other peripherals around MCU like a flash, EEPROM, sensors, and even other microcontrollers.



Generally speaking, devices communicate over the SPI interface using four wires MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (synchronization clock), and SS (Slave Select). Usually, if only one slave device is used, the SS line is omitted while the slave chip select pin is connected to the GND. However, this is a particular case. In all other cases SS pin has to be controlled manually in software – this isn't handled automatically. If more slaves are connected to the SPI interface, there are options in selecting the right slave device: one is to use dedicated SS pins for each slave, or if the slave supports this use the address byte in data packets to pick one (for instance in MCP23S17 I/O expanders).



SPI peripheral in AVR can operate in Master or Slave modes. If master mode is selected, then it takes care of generating the clock signal and starting transfer. Slave only waits for the SS line to pull down to get ready for transfer.

In any mode, master, and slave, shift registers are connected in a ring, which means full-duplex communication. Simply speaking, if the master shifts out one byte to the slave, it receives a byte from the slave as long as both – MISO and MOSI lines are connected. So if the master wants to read a single byte from a slave, it must shift the dummy byte to receive one.

AVR in master and slave SPI mode

To demonstrate AVR working in master and slave SPI modes, we will connect two Atmega328P microcontrollers. One will be the master and the second slave SPI device. First of all, let's take a look at the master SPI. In this mode, we will program AVR continuously send incrementing 8-bit value via SPI device every 10ms. The program doing this is straightforward:

```
//SPI master
#include <avr/io.h>
#include <util/delay.h>
//SPI init
void SPIMasterInit(void)
{
//set MOSI, SCK and SS as output
```

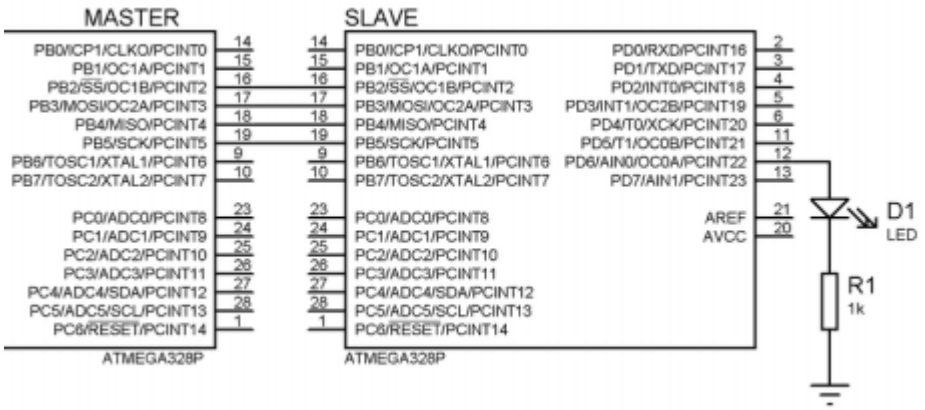


```
SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
```

```
}
//master send function
void SPIMasterSend(uint8_t data)
{
//select slave
PORTB &= ~(1<<PB2);
//send data
SPDR=data;
//wait for transmition complete
while (!(SPSR &(1<<SPIF)));
//SS to high
PORTB |= (1<<PB2);
}
int main(void)
{
//initialize master SPI
SPIMasterInit();
//initial PWM value
uint8_t pwmval = 0;
while (1)
{
SPIMasterSend(pwmval++);
_delay_ms(10);
}
}
```

First of all, we have to configure the SPI device where MOSI, SCK, and SS pins have to be configured as an output. SS pin has to be kept high when data isn't shifted out through SPI. We also need to enable the SPI device, set it to master, and select the clock rate (by setting SPE, MSTR, and SPR0 bits in the SPCR register). In our case, we simply chose it to be Fck/16. As our Atmega328p is clocked at 16MHz, then the SPI transfer rate becomes 1mb/s, or simply speaking, data is shifted out at the 1MHz clock rate. Once SPI is initialized, we have to implement the data transfer function. Before sending any data first, we have to enable SPI slave by pulling the SS pin low. Then we write a byte to the SPDR register and wait until the transfer is complete. In the main loop, we send an incrementing byte value every 10ms.

-



Once the value overflow, it simply starts over from zero. This is all we need from the master.

Then let us get to the slave device. For this, we are using another Atmega328P board clocked at 16MHz. To indicate that SPI transfer is successful, we generate a PWM signal on the OC0A pin where LED is attached. PWM is generated with Timer0 in fast PWM mode. There PWM duty cycle is controlled with the OCR0A register value. In slave, this value will be updated from SPI. As we made master send incrementing byte value, our PWM signal will vary from 0% to 100% duty cycle leading to fading LED effect. Here is a Slave SPI code:

```
//SPI slave
#include <avr/io.h>
#include <avr/interrupt.h>
//SPI init
void SPISlaveInit(void)
{
//set MISO as output
DDRB |= (1<<PB4);
//enable SPI and enable SPI interrupt
SPCR = (1<<SPE)|(1<<SPIE);
}
void InitPort(void)
{
//set PD6 (OC0A) as output
DDRD|=(1<<PD6);
}
//Initialize Timer0
```

- `void InitTimer0(void)`

```
//Set fast PWM mode
//and make clear OC0A on compare match
TCCR0A|=(1<<COM0A1)|(1<<WGM01)|(1<<WGM00);
}
void StartTimer0(void)
{
//Set prescaller 64 and start timer
TCCR0B|=(1<<CS01)|(1<<CS00);
}
ISR(SPI_STC_vect)
{
OCR0A=SPDR;
}
int main(void)
{
//initialize slave SPI
SPISlaveInit();
InitPort();
InitTimer0();
StartTimer0();
sei();
while (1)
{
//loop
}
}
```

In slave, we have set up an SPI transfer complete interrupt service routine, which will be called each time an SPI transfer is complete. ISR updates the OCR0A register with the received value.

Actually, this is it. SPI is really simple to control. And with this simplicity, you get high-speed synchronous data transfer.

It is worth to mention that the AVR microcontroller USART module can be set to work in SPI master mode. It is somehow a little bit limited since it is built using USART resources, but still, you get fully functioning master SPI.

TAGGED AVR SPI tutorial, AVR Tutorial, master slave spi, spi transfer.

3 Comments

Jo
APRIL 20, 2013 AT 6:57 PM

The picture with master and two slaves is wrong. MISO arrow at the 'slave 2' block is wrong.

admin
APRIL 21, 2013 AT 12:45 PM

Thank you for notice. It will be fixed soon.

GENOTRONEX
JANUARY 25, 2015 AT 9:41 PM

Excellent Tutorial, thank You

SEARCH

CATEGORIES

AVR Projects Arduino PIC Projects Misc Business Technology Other MCU Projects Linux board projects FPGA Projects ARM Cortex Internet MSP430 Projects Games Education Software PIC32 ARM7 Projects AVR Tutorial MSC-51 Projects PCB Health Handy Circuits Guide Lifestyle 68HC Projects ARM Cortex Tutorial BASIC Stamp Finances Casino ChipKIT Projects MSP430 Tutorial Raspberry Pi ZiLOG Marketing

RECENT COMMENTS

Ed Woodrick on Is Investing in High-End Ethernet Cable Worth it for Programmers?

admin on Order PCBs from PCBWay and get a free mask to fight COVID-19

Ed on Order PCBs from PCBWay and get a free mask to fight COVID-19

John Jennings on Programming STM32-Discovery using GNU tools. Linker script

Ashok on Programming STM32-Discovery using GNU tools. Startup code

RECENT TUTORIALS

Programming AVR I2C interface

Using Direct Memory Access (DMA) in STM32 projects

RECENT POSTS

6 Benefits of Using Solar Inverter for Residential Energy

How to make pancakes healthier with Kratom?

The Top 5 Photo Editing Software Choices

CRM vs. ERP — Do You Know the Difference?

Five Tips To Be Successful In Data Science

TOP POSTS & PAGES

TFTLCD Menu interface for Arduino

Programming AVR I2C interface

Programming STM32-Discovery using GNU tools. Linker script

Using Direct Memory Access (DMA) in STM32 projects

Accessing AVR EEPROM memory in AVRGCC