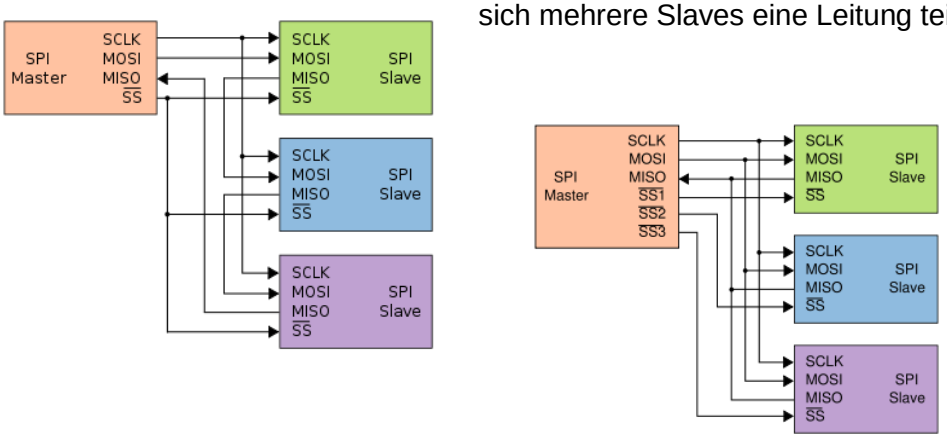
Grundlagen SPI !!!Hier bezogen auf den uController AT89C5131!!!

- Drei gemeinsame Leitungen, an denen jeder Teilnehmer angeschlossen ist:

  SDO (*Serial Data Out*) bzw. MISO (*Master in Slave out*)
  SDI (*Serial Data In*) bzw. MOSI (*Master out Slave in*)
  SCK (*Serial Clock*)

- Eine mit logisch-*0* aktive Chip-Select-Leitung für jeden Slave. Diese Leitungen werden mit SS, CS oder STE für *Slave Select*, *Chip Select* bzw. *Slave Transmit Enable* bezeichnet.
  Es gibt auch spezielle Anwendungen, bei denen sich mehrere Slaves eine Leitung teilen:



- Vollduplexfähig
- Viele Einstellmöglichkeiten, beispielsweise mit welcher Taktflanke ausgegeben bzw. eingelesen wird, Wortlänge, Übertragung MSB oder LSB zuerst.
- Unterschiedliche Taktfrequenzen bis in den MHz-Bereich zulässig.
- Vielfältige Einsatzmöglichkeiten in Audio- und Messanwendungen, zur Datenübertragung zwischen µC.

Die vielen Einstellmöglichkeiten sind unter anderem deshalb erforderlich, weil sich die Spezifikation für den SPI-Bus in vielen Dingen nicht konkret festlegt und deshalb verschiedene, zueinander inkompatible Geräte existieren. So ist häufig für jeden angeschlossenen Schaltkreis eine eigene Konfiguration des steuernden Microcontrollers (Master des SPI-Bus) erforderlich.

Viele Mikrocontroller erlauben eine Im-System-Programmierung (kurz ISP) über den SPI-Bus.

Protokollablauf und Einstellmöglichkeiten

Es können theoretisch beliebig viele Teilnehmer an den Bus angeschlossen werden, wobei es immer exakt einen Master geben muss. Er ist derjenige, der das Clock – Signal an SCK erzeugt und festlegt, mit welchem Slave er kommunizieren will. Das geschieht über die Leitung "Slave Select". Wird sie gegen Masse gezogen, wird der jeweilige Slave aktiv, "lauscht" an MOSI und legt seine Daten im Takt von SCK an MISO. Es wird somit ein Byte vom Master zum Slave und ein anderes Byte vom Slave zum Master transportiert.

Ein Protokoll für die Datenübertragung wurde von Motorola zwar nicht festgelegt, doch haben sich in der Praxis vier verschiedene Modi durchgesetzt. Diese werden durch die Parameter Clock Polarität (CPOL) und Clock Phase (CPHA) festgelegt. Bei CPOL=0 ist der Clock Idle (Ruhezustand) Low, bei CPOL=1 ist er Idle High. CPHA gibt nun an, bei der wievielten Flanke die Daten übernommen werden sollen. Bei CPHA=0 werden sie bei der ersten Flanke übernommen,

nachdem SS auf Low gezogen wurde, bei CPHA=1 bei der zweiten. Somit werden die Daten bei CPOL=0 und CPHA=0 mit der ersten Flanke übernommen, die nur eine High-Flanke sein kann. Bei CPHA=1 wäre es die zweite, also eine Low-Flanke. Bei CPOL=1 ist das ganze folglich genau anders herum, bei CPHA=0 Low-Flanke und bei CPHA=1 High-Flanke.

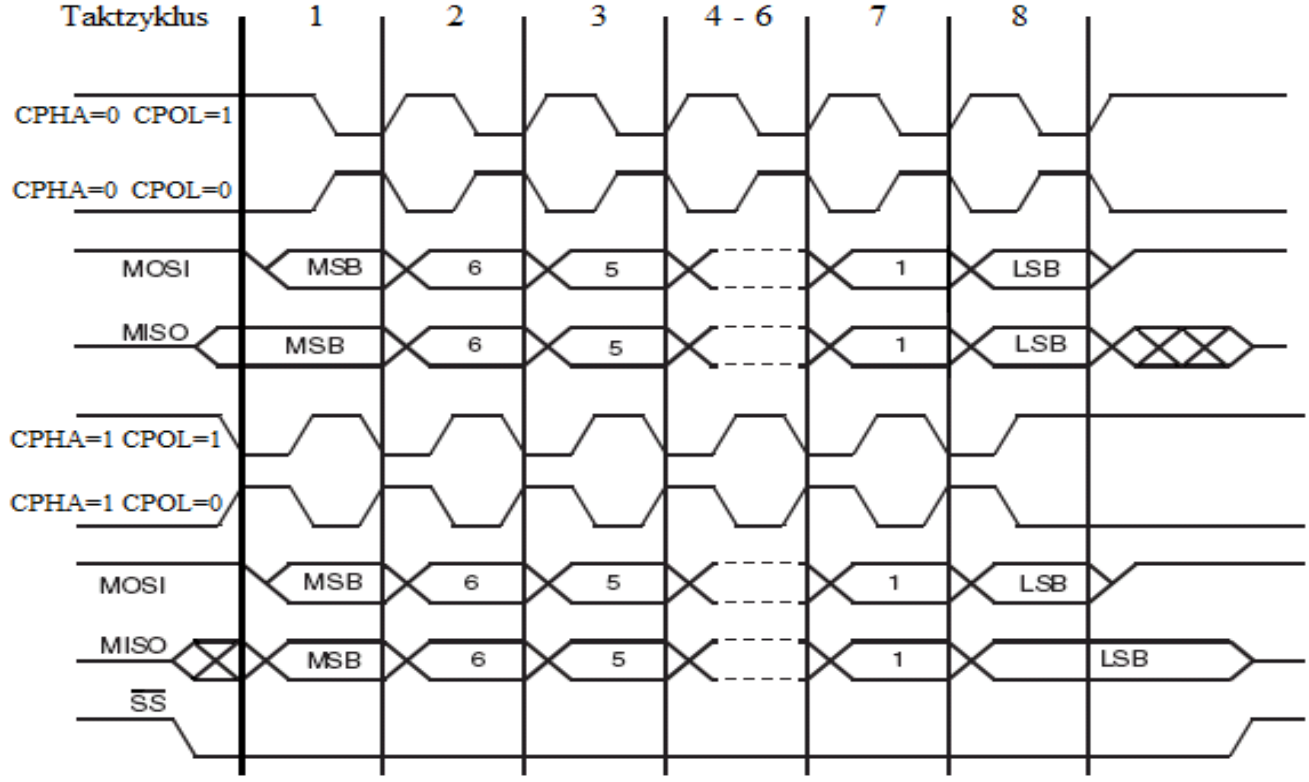
Zu beachten ist noch, dass der Slave bei CPHA=0 seine Daten schon beim Runterziehen von SS an MISO anlegt, damit der Master sie beim ersten Flankenwechsel übernehmen kann. Bei CPHA=1 werden die Daten vom Slave erst beim ersten Flankenwechsel an MISO gelegt, damit sie beim zweiten Flankenwechsel vom Master übernommen werden können. Der Master hingegen legt seine Daten immer zum gleichen Zeitpunkt an, meist kurz nach der Low-Flanke von SCK.

Mit jeder Taktperiode wird ein Bit übertragen. Beim üblichen Bytetransfer sind also 8 Taktperioden für eine vollständige Übertragung nötig. Es können auch mehrere Bytes hintereinander übertragen werden, wobei nicht festgelegt ist, ob zwischen jedem Byte das SS-Signal kurz wieder auf High gezogen werden muss. Eine Übertragung ist beendet, wenn das Slave-Select-Signal endgültig auf High gesetzt wird.

Modes

Für die verschiedenen Konstellationen existiert auch die Bezeichnung *Modes*, die die nachfolgende Tabelle auflistet.

Mode:	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



Serial Peripheral Interface (SPI)

Nach Datenblatt ATMEL uC 89C5131 ab S90
!!Bitte unbedingt beachten: Die Register lauten in verschiedenen 8051er µC verschieden!!

The Serial Peripheral Interface module (SPI) allows full-duplex, synchronous, serial communication between the MCU and peripheral devices, including other MCUs.

Features Features of the SPI module include the following:

• Full-duplex, three-wire synchronous transfers
• Master or Slave operation
• Eight programmable Master clock rates
• Serial clock with programmable polarity and phase
• Master mode fault error flag with MCU interrupt capability
• Write collision flag protection

Signal Description Figure 41 shows a typical SPI bus configuration using one Master controller and many Slave peripherals. The bus is made of three wires connecting all the devices:

Figure 41. SPI Master/Slaves Interconnection

The Master device selects the individual Slave devices by using four pins of a parallel port to control the four SS pins of the Slave devices.

Master Output Slave Input (MOSI)

This 1-bit signal is directly connected between the Master Device and a Slave Device. The MOSI line is used to transfer data in series from the Master to the Slave. Therefore, it is an output signal from the Master, and an input signal to a Slave. A byte (8-bit word) is transmitted most significant bit (MSB) first, least significant bit (LSB) last.

Master Input Slave Output (MISO)

This 1-bit signal is directly connected between the Slave Device and a Master Device. The MISO line is used to transfer data in series from the Slave to the Master. Therefore, it is an output signal from the Slave, and an input signal to the Master. A byte (8-bit word) is transmitted most significant bit (MSB) first, least significant bit (LSB) last.

SPI Serial Clock (SCK)

This signal is used to synchronize the data movement both in and out the devices through their MOSI and MISO lines. It is driven by the Master for eight clock cycles which allows to exchange one byte on the serial lines.

Slave Select (SS)

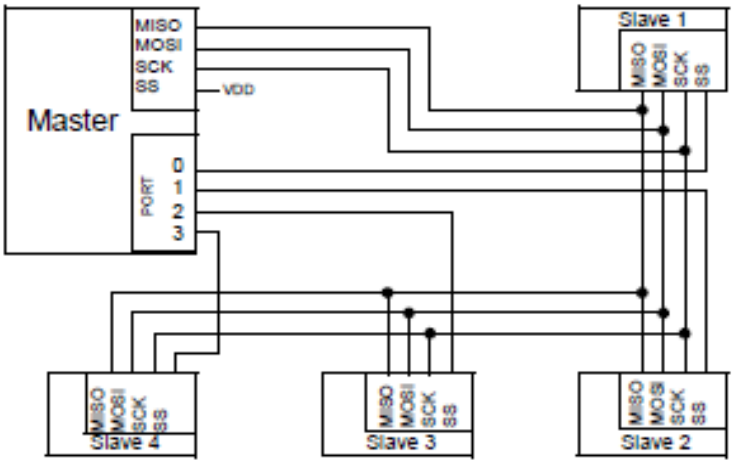
Each Slave peripheral is selected by one Slave Select pin (SS). This signal must stay low for any message for a Slave. It is obvious that only one Master (SS high level) can drive the network. The Master may select each Slave device by software through port pins (Figure 41). To prevent bus conflicts on the MISO line, only one slave should be selected at a time by the Master for a transmission. In a Master configuration, the SS line can be used in conjunction with the MODF flag in the SPI Status register (SPSTA) to prevent multiple masters from driving MOSI and SCK (see Section “Error Conditions”, page 95). A high level on the SS pin puts the MISO line of a Slave SPI in a high-impedance state. The SS pin could be used as a general-purpose if the following conditions are met:

• The device is configured as a Master and the SSDIS control bit in SPCON is set. This kind of configuration can be found when only one Master is driving the network and there is no way that the SS pin could be pulled low. Therefore, the MODF flag in the SPSTA will never be set(1).

• The Device is configured as a Slave with CPHA and SSDIS control bits set(2) This kind of configuration can happen when the system comprises one Master and one Slave only. Therefore, the device should always be selected and there is no reason that the Master uses the SS pin to select the communicating Slave device.

Notes: 1. Clearing SSDIS control bit does not clear MODF.
2. Special care should be taken not to set SSDIS control bit when CPHA =’0’ because in this mode, the SS is used to start the transmission.

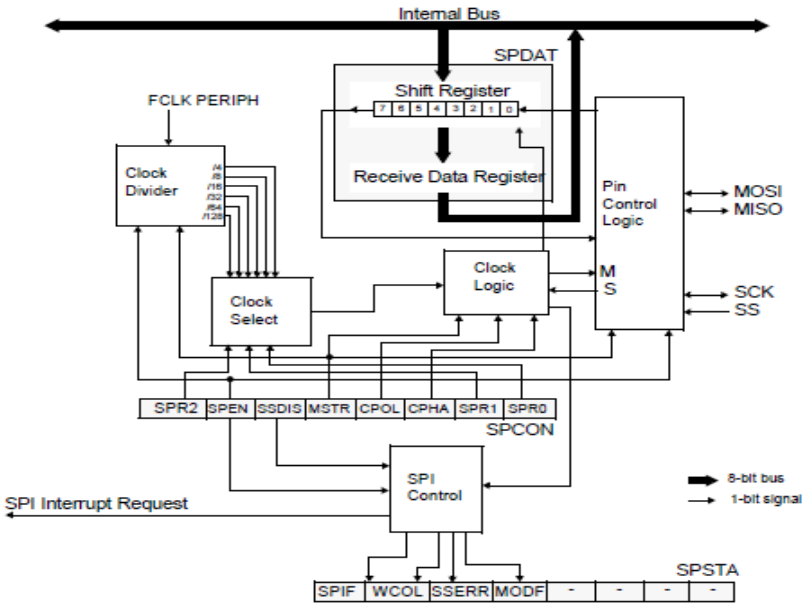
Figure 41. SPI Master/Slaves Interconnection



Functional Description

Figure 42 shows a detailed structure of the SPI module.

Figure 42. SPI Module Block Diagram



Operating Modes

The Serial Peripheral Interface can be configured as one of the two modes: Master mode or Slave mode. The configuration and initialization of the SPI module is made through one register:

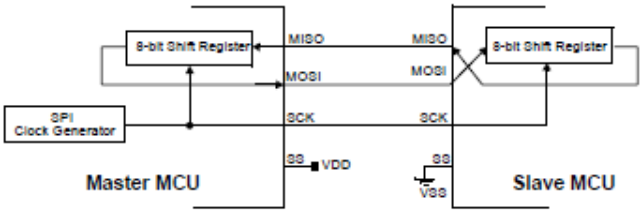
- The Serial Peripheral CONtrol register (SPCON)

Once the SPI is configured, the data exchange is made using:

- SPCON
- The Serial Peripheral STAtus register (SPSTA)
- The Serial Peripheral DATa register (SPDAT)

During an SPI transmission, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock line (SCK) synchronizes shifting and sampling on the two serial data lines (MOSI and MISO). A Slave Select line ($\overline{SS}$) allows individual selection of a Slave SPI device; Slave devices that are not selected do not interfere with SPI bus activities.

Figure 43. Full-duplex Master/Slave Interconnection



Master Mode

The SPI operates in Master mode when the Master bit, MSTR (1), in the SPCON register is set. Only one Master SPI device can initiate transmissions. Software begins the transmission from a Master SPI module by writing to the Serial Peripheral Data Register (SPDAT). If the shift register is empty, the byte is immediately transferred to the shift register. The byte begins shifting out on MOSI pin under the control of the serial clock, SCK. Simultaneously, another byte shifts in from the Slave on the Master's MISO pin. The transmission ends when the Serial Peripheral transfer data flag, SPIF, in SPSTA becomes set. At the same time that SPIF becomes set, the received byte from the Slave is transferred to the receive data register in SPDAT. Software clears SPIF by reading the Serial Peripheral Status register (SPSTA) with the SPIF bit set, and then reading the SPDAT.

Slave Mode

The SPI operates in Slave mode when the Master bit, MSTR (2), in the SPCON register is cleared. Before a data transmission occurs, the Slave Select pin, $\overline{SS}$, of the Slave device must be set to'0'. $\overline{SS}$ must remain low until the transmission is complete.

In a Slave SPI module, data enters the shift register under the control of the SCK from the Master SPI module. After a byte enters the shift register, it is immediately transferred to the receive data register in SPDAT, and the SPIF bit is set. To prevent an overflow condition, Slave software must then read the SPDAT before another byte enters the shift register (3). A Slave SPI must complete the write to the SPDAT (shift register) at least one bus cycle before the Master SPI starts a transmission. If the write to the data register is late, the SPI transmits the data already in the shift register from the previous transmission.

Transmission Formats

Software can select any of four combinations of serial clock (SCK) phase and polarity using two bits in the SPCON: the Clock POLarity (CPOL (4)) and the Clock PHAse (CPHA4). CPOL defines the default SCK line level in idle state. It has no significant effect on the transmission format. CPHA defines the edges on which the input data are sampled and the edges on which the output data are shifted (Figure 44 and Figure 45). The clock phase and polarity should be identical for the Master SPI device and the communicating Slave device.

1. The SPI module should be configured as a Master before it is enabled (SPEN set). Also the Master SPI should be configured before the Slave SPI.
2. The SPI module should be configured as a Slave before it is enabled (SPEN set).
3. The maximum frequency of the SCK for an SPI configured as a Slave is the bus clock speed.
4. Before writing to the CPOL and CPHA bits, the SPI should be disabled (SPEN ='0').

Beteiligte Register am SPI Bus:

Mnemonic	Add	Name	7	6	5	4	3	2	1	0
SPCON	C3h	Serial Peripheral Control	SPR2	SPEN	SSDIS	MSTR	CPOL	CPHA	SPR1	SPR0
SPSTA	C4h	Serial Peripheral Status-Control	SPIF	WCOL	SSERR	MODF	-	-	-	-
SPDAT	C5h	Serial Peripheral Data	R7	R6	R5	R4	R3	R2	R1	R0

Bitte Details im Datenblatt nachschauen!!!

Table 76. SPDAT Register

SPDAT - Serial Peripheral Data Register (0C5H)

Table 2.

7	6	5	4	3	2	1	0
R7	R6	R5	R4	R3	R2	R1	R0

SPSTA - Serial Peripheral Status and Control register (0C4H)

Table 1.

7	6	5	4	3	2	1	0
SPIF	WCOL	SSERR	MODF	-	-	-	-

Bit Number	Bit Mnemonic	Description
7	SPIF	**Serial Peripheral data transfer flag** Cleared by hardware to indicate data transfer is in progress or has been approved by a clearing sequence. Set by hardware to indicate that the data transfer has been completed.
6	WCOL	**Write Collision flag** Cleared by hardware to indicate that no collision has occurred or has been approved by a clearing sequence. Set by hardware to indicate that a collision has been detected.
5	SSERR	**Synchronous Serial Slave Error flag** Set by hardware when $\overline{SS}$ is de- asserted before the end of a received data. Cleared by disabling the SPI (clearing SPEN bit in SPCON).
4	MODF	**Mode Fault** Cleared by hardware to indicate that the $\overline{SS}$ pin is at appropriate logic level, or has been approved by a clearing sequence. Set by hardware to indicate that the $\overline{SS}$ pin is at inappropriate logic level.
3	-	**Reserved** The value read from this bit is indeterminate. Do not set this bit

Tabelle für Taktgeschwindigkeit, wird eingestellt in nachfolgendem SPCON Register

Table 72. SPI Master Baud Rate Selection

SPR2	SPR1	SPR0	Clock Rate	Baud Rate Divisor (BD)
0	0	0	Don't Use	No BRG
0	0	1	$F_{CLK\ PERIPH}/4$	4
0	1	0	$F_{CLK\ PERIPH}/8$	8
0	1	1	$F_{CLK\ PERIPH}/16$	16
1	0	0	$F_{CLK\ PERIPH}/32$	32
1	0	1	$F_{CLK\ PERIPH}/64$	64
1	1	0	$F_{CLK\ PERIPH}/128$	128
1	1	1	Don't Use	No BRG

Table 74. SPCON Register

7	6	5	4	3	2	1	0
SPR2	**SPEN**	**SSDIS**	**MSTR**	CPOL	CPHA	**SPR1**	**SPR0**

Bit Number	Bit Mnemonic	Description
7	SPR2	**Serial Peripheral Rate 2** Bit with SPR1 and SPR0 define the clock rate.
6	SPEN	**Serial Peripheral Enable** Cleared to disable the SPI interface. Set to enable the SPI interface.
5	SSDIS	**$\overline{SS}$ Disable** Cleared to enable $\overline{SS}$ in both Master and Slave modes. Set to disable $\overline{SS}$ in both Master and Slave modes. In Slave mode, this bit has no effect if CPHA = "0".
5	MSTR	**Serial Peripheral Master** Cleared to configure the SPI as a Slave. Set to configure the SPI as a Master.
4	CPOL	**Clock Polarity** Cleared to have the SCK set to "0" in idle state. Set to have the SCK set to "1" in idle state.
3	CPHA	**Clock Phase** Cleared to have the data sampled when the SCK leaves the idle state (see CPOL). Set to have the data sampled when the SCK returns to idle state (see CPOL).
2 1	SPR1 SPR0	**SPR2 SPR1 SPR0 Serial Peripheral Rate** **000Reserved** **00 1$F_{CLK\ PERIPH}$/4** **010 $F_{CLK\ PERIPH}$/8** **011$F_{CLK\ PERIPH}$/16** **100$F_{CLK\ PERIPH}$/32** **10 1$F_{CLK\ PERIPH}$/64** **110$F_{CLK\ PERIPH}$/128** **1 11Reserved**

Reset Value = 0001 0100b

Programm für SPI-Master in C

!!Bitte unbedingt beachten: Funktion nur bei ATMEL uC AT89C5131 nicht alle 8051er sind kompatibel!! Bitte immer prüfen!!

```
/*******************************************************************************
|       CLASS:                  8051er AT89C5131!!!!!
|       COMPILER:               Keil uV3
|       PROGRAM:        Master01.c
|       AUTHOR:                 G. Neumaier  Gewerbl. Schule Offenburg
|       DATE:                   30.11.2011
|       DESCRIPTION:    Datenübertragung über SPI:  P2 wird übertragen
|       REQUIEREMENTS:  8051er AT89C5131
|       NOTES:                  Programm komplett, keine Aenderungen notwendig
|                                       uCMaster mit uC Slave verbinden mit direkter Verbindung P1 <->P1
                                        ohne Drehung. P1.1-P1.1  P1.5-P1.5  P1.6-P1.6 P1.7-P1.7
|       Aufgabe:        Funktion:  unsigned char SPI_send_receive(unsigned char daten)
|                                       in anderen Programmen benutzen.
*******************************************************************************/
//---------------------HEADER FILES-----------------------------------------


#include <at89c5131.h>  // Header-Datei für ÁT89C5131

//---------------------PROGRAM CONSTANTS-------------------------------------

#define schalter8 P2            //Der Variablen "Schalter8" wird der PORT P2 zugewiesen
#define LEDs1 P0        //Der Variablen "LED1" wird der PORT P0 zugewiesen
#define LEDs2 P3        //Der Variablen "LED2" wird der PORT P3 zugewiesen
#define CS   P1_1               //Der Variablen "CS" wird der PORTpin 1.4 zugewiesen
//-------------------------------INIT-------------------------------------------
void init (void)/*      Steht die Funktion vor main, dann muss die Funktion nicht
                                        als Prototyp angemeldet werden*/
{
//Steht schon in der Includedateifür den AT89C5131
//Sfr ( SPCON , 0xC3 ) ; SPI control register
//Sfr ( SPSTA , 0xC4 ) ; SPI status register
//Sfr ( SPDAT , 0xC5 ) ; SPI data register

//CS   P1.1             //Slave aktivieren SS Select Slave (hier CS Chip Select)
//MOSI P1.7  SPI
//MISO P1.5  SPI  //AT89C5131
//SCK  P1.6  SPI
        SPCON = 0xF6; //1111 0110 CPHA=1CPOL=0
                //          1                   SPR2 = SPI taktrate zusammen mit SPR1 und SPR0
                //           1          SPEN = SPI enable SPI-Übertragung freigeben
                //            1                 SSDIS = Slave Select über Portpin P1.1 0 = deaktiviert
                //             1        MSTR = Master/Slave select 1 = Master !!!!
                //               0      CPOL = Clock Polarity 0 = Clock ist in Ruhe auf 0
                //                1     CPHA = Clock Phase 1 = zweite Clock-Flanke gültig für
Datenübernahme
                //                 10   SPR1 und SPR0 SPI taktrate bei 11 = f/128
}
//---------------------FUNCTION PROTOTYPES-----------------------------------

void wait (void);               //Funktionsprototyp wait, Semikolon nicht vergessen
unsigned char SPI_send_receive(unsigned char daten);   //Funktion für SPI bekanntgeben

unsigned char senden_SPI; empfangen_SPI;

//-----------------------------MAIN-------------------------------------------
void main (void)        //Beginn Hauptprogramm
{
        init();            //Funktionsaufruf init
  while (1)                     //Beginn Endlosschleife
   {
         senden_SPI = schalter8;      //Port an Variable senden_SPI
         LEDs1 = senden_SPI;       //Zur Kontrolle an LEDs1 ausgeben

         empfangen_SPI = SPI_send_receive(senden_SPI);  //Funktion aufrufen mit
                                //Übergabewert senden_SPI und Rückgabewert empfangen_SPI
```

```
        LEDs2 =empfangen_SPI;          //SPI empfangen
          wait ();                        //Funktionsaufruf "Unterprogramm" wait

  }                             //Ende WHILE-Schleife
}                               //Ende MAIN-Programm

//------------------------------FUNCTION-----------------------------------------

//Funktion zum Senden und gleichzeitigem Empfangen von SPI Daten
unsigned char SPI_send_receive(unsigned char daten)
{
        CS =0;  //Slave aktivieren, Portpin p1.1
       SPDAT = daten;   //output        SPI     Übertragung beginnt automatisch

       while((SPSTA & 0x80)==0)
       {//warten bis bit7 im Reg SPSTA 1 wird
       }
        CS = 1;//Slave deaktivieren, Portpin p1.1

       return (SPDAT); //Empfangene SPI Daten zurückgeben
}

//-----------------------------------------------
void wait (void)                // wait function: Zeitschleife ca 0,1 sec
{
  unsigned int i;           // Variable lokal, nur gueltig in dieser Funktion!
  for (i = 0; i< 60; i++)       // Zeitschleife
  { ; }
} // Ende "Unterprogramm" wait
```

Programm für SPI-Slave in C

!!Bitte unbedingt beachten: Funktion nur bei ATMEL uC AT89C5131 nicht alle 8051er sind kompatibel!! Bitte immer prüfen!!

```
/*****************************************************************************
|       CLASS:                  8051er AT89C5131!!!!!
|       COMPILER:               Keil uV3
|       PROGRAM:        Slave01.c
|       AUTHOR:                 G. Neumaier  Gewerbl. Schule Offenburg
|       DATE:                   30.11.2011
|       DESCRIPTION:    Datenübertragung über SPI  P2 wird übertragen
|       REQUIEREMENTS:  8051er AT89S8252/53
|       NOTES:                  Programm komplett, keine Aenderungen notwendig
|                                       uCMaster mit uC Slave verbinden mit direkter Verbindung P1 <->P1
                                        ohne Drehung. P1.1-P1.1  P1.5-P1.5  P1.6-P1.6 P1.7-P1.7
|       Aufgabe:        Funktion:  unsigned char SPI_send_receive(unsigned char daten)
|                                       in anderen Programmen benutzen.
*****************************************************************************/
//----------------------HEADER FILES-----------------------------------------


#include <at89c5131.h>  // Header-Datei für ÁT89C5131

//---------------------PROGRAM CONSTANTS-------------------------------------


#define LEDs1 P3        //Der Variablen "LEDs1" wird der PORT P3 zugewiesen

//------------------------------INIT---------------------------------------------
void init (void)/*      Steht die Funktion vor main, dann muss die Funktion nicht
                                       als Prototyp angemeldet werden*/
{
//Steht schon in der Includedateifür den AT89C5131
//Sfr ( SPCON , 0xC3 ) ; SPI control register
//Sfr ( SPSTA , 0xC4 ) ; SPI status register
//Sfr ( SPDAT , 0xC5 ) ; SPI data register
```

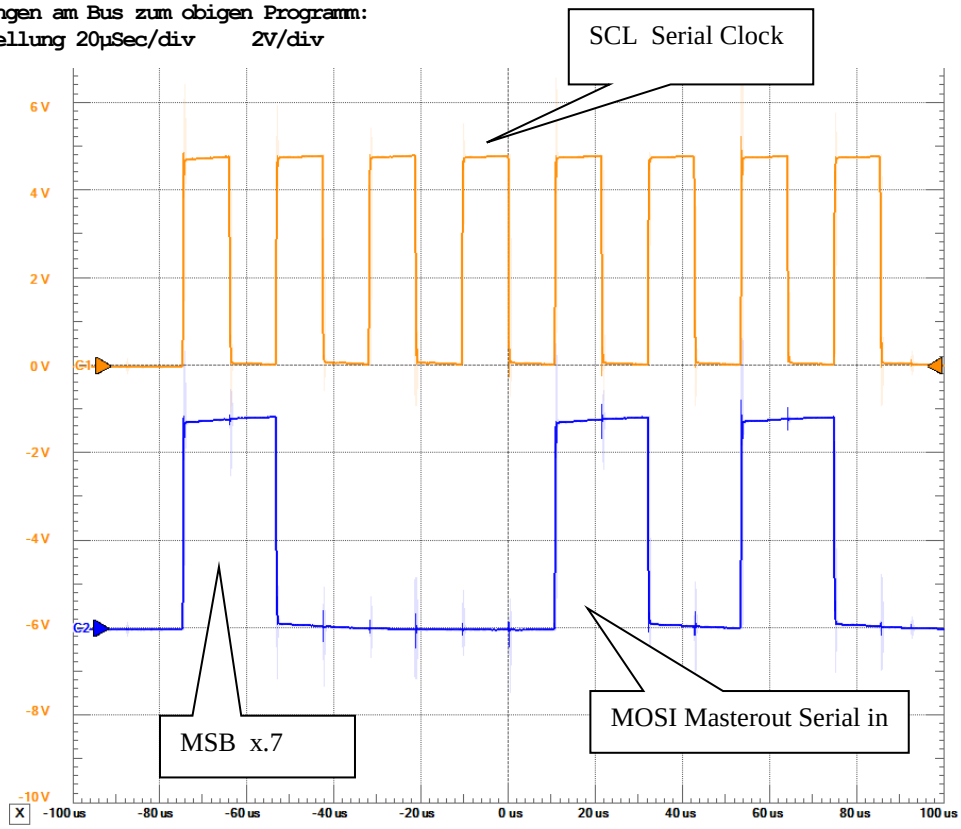
```
//CS   P1.1            //Slave aktivieren SS Select Slave (hier CS Chip Select)
//MOSI P1.7  SPI
//MISO P1.5  SPI  //AT89C5131
//SCK  P1.6  SPI
       SPCON = 0xC6; //1100 0110 CPHA=1CPOL=0
              //         1                SPR2 = SPI taktrate zusammen mit SPR1 und SPR0
              //          1          SPEN = SPI enable SPI-Übertragung freigeben
              //           0                SSDIS = Slave Select über Portpin P1.1 0 = deaktiviert
              //            0        MSTR = Master/Slave select 0 = Slave !!!!
              //              0      CPOL = Clock Polarity 0 = Clock ist in Ruhe auf 0
              //               1     CPHA = Clock Phase 1 = zweite Clock-Flanke gültig für
Datenübernahme
              //                10   SPR1 und SPR0 SPI taktrate bei 11 = f/128
}
//---------------------FUNCTION PROTOTYPES--------------------------------

void wait (void);               //Funktionsprototyp wait, Semikolon nicht vergessen
unsigned char SPI_send_receive(unsigned char daten);   //Funktion für SPI bekanntgeben

unsigned char senden_SPI; empfangen_SPI;

//------------------------------MAIN-----------------------------------------

void main (void)        //Beginn Hauptprogramm
{
       init();            //Funktionsaufruf init
  while (1)                     //Beginn Endlosschleife
   {
    senden_SPI = LEDs1;      //empfangener Wert rücksenden
       empfangen_SPI = SPI_send_receive(senden_SPI); //Funktion aufrufen mit
                               //Übergabewert senden_SPI und Rückgabewert empfangen_SPI
        LEDs1 =empfangen_SPI;  //Empfangener Wert anzeigen
        wait ();       //Funktionsaufruf "Unterprogramm" wait. Kurz warten

  }                            //Ende WHILE-Schleife
}                              //Ende MAIN-Programm

//-----------------------------FUNCTION-----------------------------------------

//Funktion zum Senden und gleichzeitigem Empfangen von SPI Daten
unsigned char SPI_send_receive(unsigned char daten)
{
       SPDAT = daten;   //output        SPI     Übertragung beginnt automatisch

       while((SPSTA & 0x80)==0)
       {//warten bis bit7 im Reg SPSTA 1 wird
       }
       return (SPDAT); //Empfangene SPI Daten zurückgeben
}

//---------------------------------------------
void wait (void)                // wait function: Zeitschleife ca 0,1 sec
{
  unsigned int i;           // Variable lokal, nur gueltig in dieser Funktion!
  for (i = 0; i< 60; i++)       // Zeitschleife
  { ; }
} // Ende "Unterprogramm" wait

--------------------------------------
```

Auf den beiden C-Programme beruhen die nachfolgend abgebildeten Messungen mit der Hardware „Analog-Discovery“ von Digilent.

Aufgaben:

1. CPOL-bit und CPHA verändern, dann die Messung wiederholen.
2. Die Übertragungsgeschwindigkeit bestimmen und dann im Programm verändern. Messungen wiederholen

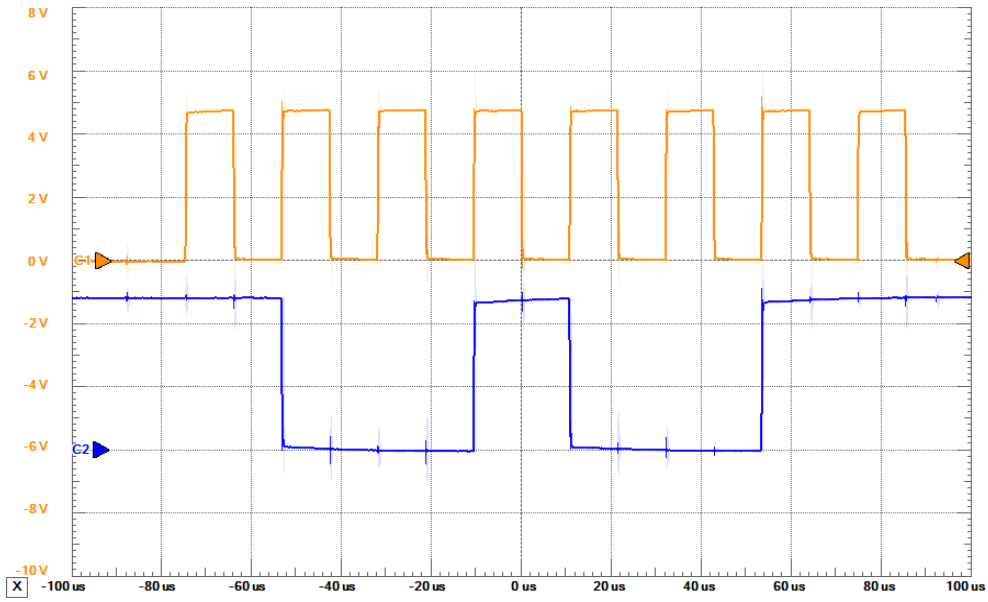
Messungen am Bus zum obigen Programm:
Einstellung 20µSec/div 2V/div



Der übertragene Datenwert: 1 0 0 1 0 0 1 0 = Hex 92

Eingestellt beim Takt: Beginn mit low-Level, Auswertung Daten an der 2ten Taktflanke

Wenn LSB High, dann bleibt Datenleitung auf High bis zur nächsten Überragung:



Der übertragene Datenwert: 1 0 0 1 0 0 1 1 = Hex 93

Darstellung mit dem Logic-Analyser von WaveForms (Analog Discovery von Digilent)