SPI – Serial Peripheral Interface

Physik- Seminar

Universität Koblenz-Landau

Christian Büch

27. Juni 2006

Einleitung

SPI bedeutet Serial Peripheral Interface – zu Deutsch „serielle Peripherie Schnittstelle“ – und beschreibt einen synchronen seriellen Bus.

Dieser wurde von Motorola entwickelt, jedoch nie in einen Vollständigen Standard oder eine Norm überführt. Außerdem wurde seitens Motorola keine Definition über ein Softwareprotokoll gemacht, sondern nur die reine Hardware-Funktionsweise beschrieben. Auch wurde SPI nie mit patenten belegt und ist somit Lizenz frei. Das hat dem Bussystem, neben der einfachen Implementierung, eine weite Verbreitung verschafft.

Wie bereits erwähnt ist SPI ein synchroner serieller Bus. Dieser ermöglicht die Anbindung von Peripherien an einen Mikrocontroller. Auch ist die Verbindung mehrerer Mikrocontroller miteinander möglich. SPI erreicht hier sehr hohe Datenübertragungsraten, da das Taktsignal bis in den MHz Bereich reichen kann. Außerdem werden die Daten in beide Richtungen gleichzeitig übertragen, also Voll-Duplex.

Der Hardware Aufwand bleibt dabei in Grenzen, da neben den Slave-Select oder Chip-Select Leitungen nur 1 Steuerleitung für den Takt sowie 2 Datenleitungen benötigt werden. In der Konfiguration (Polarität etc.) ist SPI ebenfalls sehr flexibel.

Funktionsweise

SPI ist als Master-Slave Bus ausgelegt, d.h. ein Master muss immer die Datenübertragung einleiten sowie die Slaves selektieren. Auch stellt der Master das Taktsignal bereit.
Die Daten werden auf 2 Datenleitungen übertragen. Diese sind MISO (Master-In Slave-Out, Dateneingang Master) und MOSI (Master-Out Slave-In, Datenausgang Master). Jeder Slave besitzt ein Slave Select – Signal oder Chip Select – Signal. Diese sind meistens Low-Aktiv. Solange ein Slave nicht ausgewählt – selektiert – ist, sind Taktsignal und die Datensignale im sogenannten TriState Modus und extreme hochohmig. Es werden also keine Daten / Takte zur SPI-Einheit durchgelassen.

Synchron zum Taktsignal vom Master werden die Daten an den Datenleitungen ausgegeben. Dieses wird über ein Schieberegister realisiert. Meist wird das höchstwertigste Bit (MSB) zuerst ausgegeben, die niederwertigen Bits folgend. Bei manchen Mikrocontroller (z.B. bei ATMEL 8-Bit AVR) kann man einstellen, ob zuerst MSB oder LSB gesendet werden soll. Ein Datenwort beträgt bei der SPI immer 8Bit, auch in 16 oder 32Bit Systemen. Die empfangenen Daten liegen nach dem Datentransfer im gleichen Register wie die Sendedaten. Es existiert also nur ein Register für Sende-/Empfangsdaten. Schreibt man ein Datenwort in dieses Register, so wird automatisch eine Datenübertragung eingeleitet. Um festzustellen ob die Übertragung abgeschlossen ist, gibt es Flags in Statusregistern oder man verwendet Interrupts.

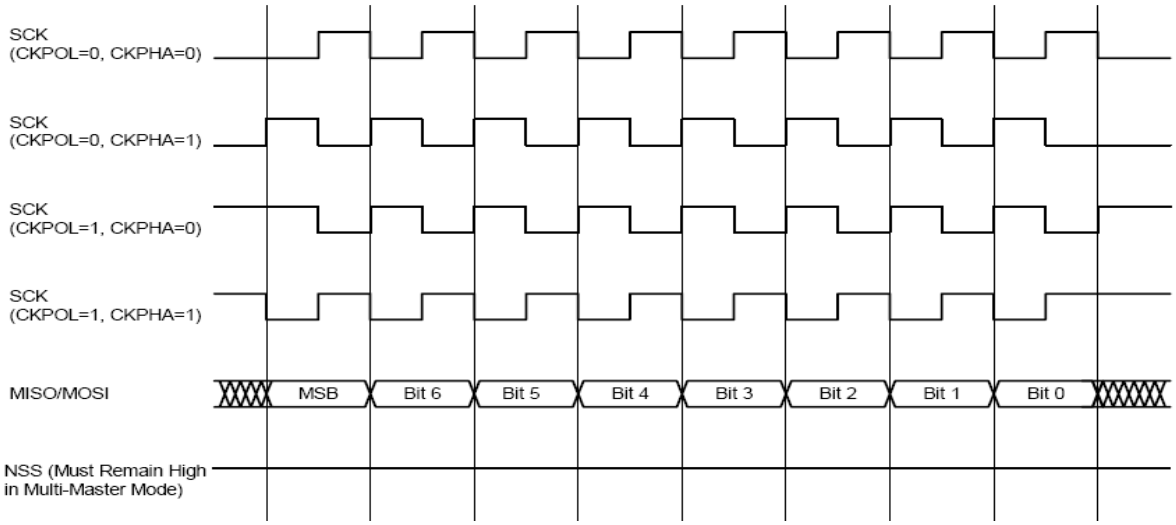

Abbildung 1 - Datenübertragung über SPI

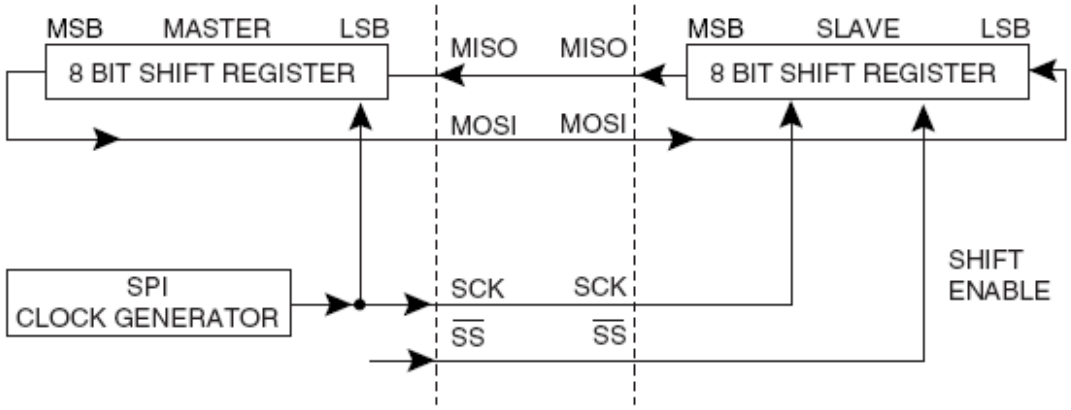

Abbildung 2 - Hardware Implementierung

SPI kennt verschiedene Modi, die über Konfigurationsregister eingestellt werden können.

Der am häufigsten genutzte Modus ist der „3-Wire Master-Slave“ Modus. Hierbei werden nur zwei Datenleitungen sowie die Taktleitung benötigt. Das CS (Chip Select) oder SS (Slave Select) Signal des Slaves liegt fest auf Masse. Damit ist der Slave immer angewählt. Hierbeikann allerdings kein weiterer Slave angesteuert werden.

Ein weiterer Modus ist dier „4-Wire Master-Slave / Multi-Master“ Modus. Hierbei existieren neben den 3 benötigten Daten-/Steuerleitungen noch ein Slave-Select Signal vom Master. Dieser wählt vor dem Datentransfer den Slave aus, welcher dann die Daten empfangen kann. Der Multi-Master Modus wird hauptsächlich verwendet wenn man zwei Mikrocontroller miteinander verbindet. Somit kann jeder Mikrocontroller einen Datentransfer einleiten. Nur während einem Datentransfer ist einer als Master und einer als Slave konfiguriert. Im „Ruhezustand“ fungieren beide als Slave. Die Umschaltung erfolgt automatisch.

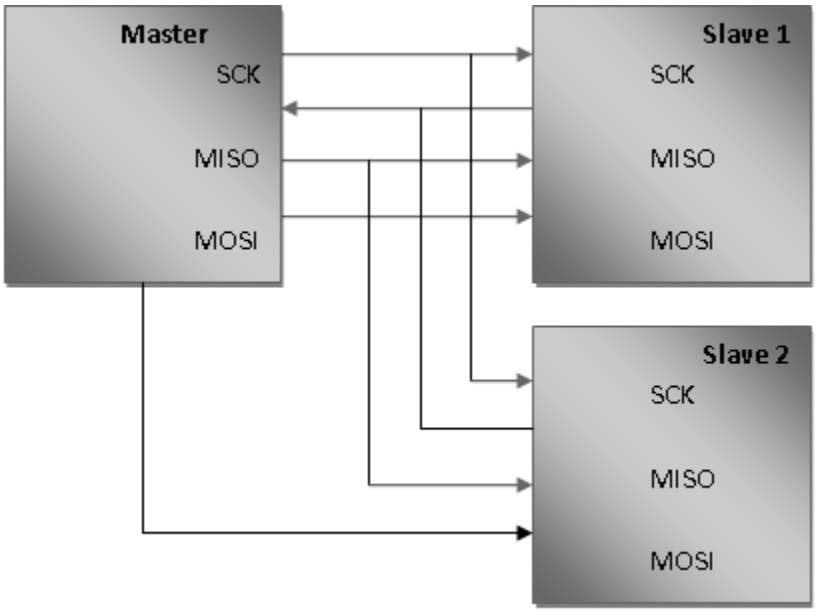

Abbildung 3 – 4-Wire Single Master / Multi Slave

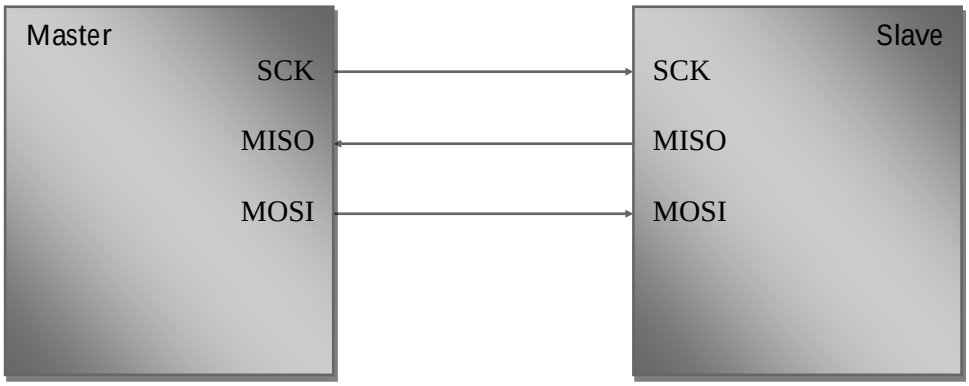

Abbildung 4 – 3 Wire Master / Slave

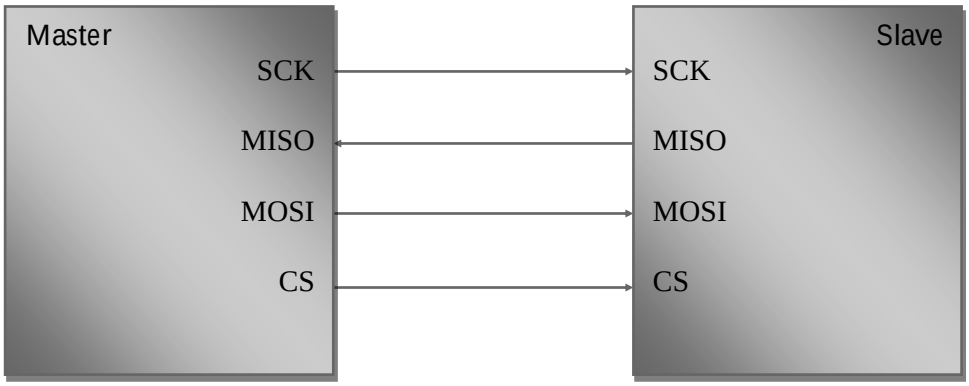

Abbildung 5 – 4-Wire Master / Slave

SPI lässt sich je nach Mikrocontroller und Peripherie recht flexibel konfigurieren. So kann man sowohl konfigurieren in welchem Logischem Zustand sich die Taktleitung befindet wenn sie im Ruhezustand ist. Ebenfalls kann man einstellen, wann ein Datenbit übernommen werden soll. Bei der ersten „Taktecke“ oder bei der letzten „Taktecke“.

Beschreibung	CKPOL	CKPHA
Taktsignal im Ruhezustand auf Low-Pegel Datenübernahme bei fallender Taktflanke	0	0
Taktsignal im Ruhezustand auf High-Pegel Datenübernahme bei steigender Taktflanke	1	0
Taktsignal im Ruhezustand auf Low-Pegel Datenübernahme bei steigender Taktflanke	0	1
Taktsignal im Ruhezustand auf High-Pegel Datenübernahme bei fallender Taktflanke	1	1

Für die Polarität sowie die Clockphase sind die beiden Bits CKPOL bzw. CKPHA zuständig.
Ist CKPOL eine logische null, so ist die Taktleitung im Ruhezustand auf einer logischen Null. Ist das Bit gesetzt, also auf logischer 1, so ist die Taktleitung im Ruhezustand auf einem logischen High-Pegel.
Ist das CKPHA auf einer logischen null, so wird ein Datenbit bei der letzten „Taktecke“ übernomen, ist das CKPHA gesetzt, also eine logische eins, so wird das Datenbit bei der ersten „Taktecke“ übernommen.

Auch lässt sich der Takt meistens recht frei einstellen. Dieser ist oft jedoch vom CPU Takt vorgegeben und kann dann über Teiler heruntergeteilt werden.

Konfigurationsbeispiel für Atmel 8-Bit AVR

In dem Atmel 8-Bit AVR Mikrocontroller gibt es 3 Register, welche für den SPI-Bus zuständig sind. Das ist zum einen das Datenregister in dem die zu übertragenden Daten geschrieben werden bzw. Empfangene Daten ausgelesen werden können. Dann gibt es noch ein Konfigurationsregister (SPCR) sowie ein Statusregister (SPSR).

Bevor man SPI verwenden kann, muss man den Bus im Mikrocontroller konfigurieren. Im Beispiel soll der Atmel AVR mit 16MHz laufen und der SPI mit der maximal möglichen Taktrate von 1MHz. Eine höhere Taktrate wäre nur möglich wenn als Slave ebenfalls ein Mikrocontroller verwendet wird. Außerdem stellt schon 1MHz Übertragung, relativ hohe Anforderungen an das Platinenlayout. Ganz zu schweigen von 8MHz oder mehr ..

Zunächst wird also das SPCR konfiguriert. Das SPCR ist ein 8 Bit Register mit 8 Konfigurationsflags.

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 4 - SPCR Register

Beschreibung der Flags

SPIE (SPI-Interrupt Enable Flag)
Beschreibt ob der SPI Interrupt aktiviert wird (logisch 1) oder ob er deaktiviert ist (logisch 0).

SPE (SPI Enable)
Beschreibt ob SPI angeschaltet ist oder abgeschaltet (nach logischem Zustand 1 oder 0)

DORD (Data Order)
Beschreibt welches Bit aus dem Datenregister zuerst gesendet werden soll. Bei einer logischen null, wird zunächst das höchstwertigste Bit gesendet, bei einer logischen eins wird zunächst das niederwertigste Bit gesendet.

MSTR (Master / Slave Select)
Beschreibt ob der Mikrocontroller als Master fungiert oder als Slave

CPOL (Idle Polarity)
Beschreibt die Polarität der Taktleitung im Ruhezustand

CPHA (Clock Phase Bit)
Beschreibt bei welcher Taktflanke die Daten übernommen werden sollen.

SPR1 / SPR0
Beschreibt die Taktrate des SCK Signals in Abhängigkeit vom CPU Takt

SPR1	SPR0	SPI2X	Taktrate
0	0	0	CPU / 4
0	0	1	CPU / 2
0	1	0	CPU / 16
0	1	1	CPU / 8
1	0	0	CPU / 64
1	0	1	CPU / 32
1	1	0	CPU / 128
1	1	1	CPU / 64

Das SPI2X Flag ist im SPI Status Register

Das SPCR wird nun mit dem Bitmuster 01010001b konfiguriert. Damit ist der Mikrocontroller als Master, SPI enable, Interrupt Disable, CPU-Takt durch 16, Polarität auf null und Datenübernahme bei der ersten Taktflanke konfiguriert.

Anschließend wird das SPI Status Register konfiguriert (SPSR).

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 5 - SPI Control Register

In dem Register kann man nur das SPI2X Flag konfigurieren. Da das Flag aber nur für die Taktrate zuständig ist und es in diesem Beispiel nicht benötigt wird (CPU-Takt / 16) bleibt das Register unberührt.

Nun kann man einen Slave selektieren (Slave Select auf logische null) und durch Schreiben von Daten in das Datenregister eine Übertragung einleiten. Wartende Daten im Slave werden bei dieser Übertragung zum Master gesendet.

Anwendungsgebiete von SPI

SPI verwendet man hauptsächlich dafür, wofür es auch entwickelt worden ist. Zur Anbindung von Peripherien an einen Mikrocontroller bzw. zur Verbindung mehrerer Mikrocontroller untereinander. Jedoch sollten sich alle Bauteile auf der gleichen Platine befinden. SPI eignet sich nur bedingt zur Verbindung von Platinen über ein Flachbandkabel. Für die Anbindung von externen Komponenten ist SPI komplett ungeeignet, da es keinerlei Störschutz gegen externe Signale gibt.

In der folgenden Tabelle sind ein paar Beispiel Peripherien aufgeführt:

Bezeichnung	Hersteller	Art	Beschreibung
AT25256	Atmel	Speicher	SPI EEPROM mit 256Kbit
AT25FS040	Atmel	Speicher	SPI Flash mit 4Mbit
AD7652	Analog Devices	ADC	16Bit Analog / Digital Konverter
AD5426	Analog Devices	DAC	8Bit Digital / Analog Konverter
MAX5499	Maxim-Dallas	Poti	Digital Potentiometer
MCP25020	Microchip	CAN	CAN Controller mit SPI
MMC	Diverse	Speicher	MultiMedia Card / RS-MMC

Eines der bekanntesten SPI – Komponenten dürfte wohl die MMC Speicherkarte sein, welche zwar nicht so schnell ist wie die SD-Karte, aber dafür Lizensfrei ist und kompatibel zur SD-Karte. Sie findet in vielen verschiedenen Arten von Geräten Anwendung (Digitalkamera, PDAs, Handys etc.).
Wie man erkennt gibt es die verschiedensten Arten von Peripherien, der Bus ist nicht auf eine Art von Peripherie festgelegt. Oft kommt es sogar vor, das Peripherien, die nur eine Ausgabe haben auf die Eingangsleitung (MOSI) verzichten.

Atmel Programmierinterface

SPI oder zumindest ein Interface, welches vom Prinzip Funktioniert wie SPI kommt auch beim Programmierinterface für Atmel AVR Prozessoren zum Einsatz. Dieses muss allerdings zunächst über eine Aktivierungsroutine angeschaltet werden. Dafür verwendet man folgende Sequenz:

1. /RESET sowie SCK auf Low-Pegel
2. 20ms warten
3. Die „Enable Serial Instruction“ an MOSI des Atmel senden.

Der Atmel AVR fungiert nun als Slave und lässt sich über eine Reihe von Befehlen lesen / schreiben sowie steuern:

Instruction[1]/Operation	Instruction Format			
	Byte 1	Byte 2	Byte 3	Byte4
Programming Enable	$AC	$53	$00	$00
Chip Erase (Program Memory/EEPROM)	$AC	$80	$00	$00
Poll RDY/$\overline{BSY}$	$F0	$00	$00	data byte out
Load Instructions				
Load Extended Address byte[1]	$4D	$00	Extended adr	$00
Load Program Memory Page, High byte	$48	adr MSB	adr LSB	high data byte in
Load Program Memory Page, Low byte	$40	adr MSB	adr LSB	low data byte in
Load EEPROM Memory Page (page access)[1]	$C1	$00	adr LSB	data byte in
Read Instructions				
Read Program Memory, High byte	$28	adr MSB	adr LSB	high data byte out
Read Program Memory, Low byte	$20	adr MSB	adr LSB	low data byte out
Read EEPROM Memory	$A0	adr MSB	adr LSB	data byte out
Read Lock bits	$58	$00	$00	data byte out
Read Signature Byte	$30	$00	0000 000aa	data byte out
Read Fuse bits	$50	$00	$00	data byte out
Read Fuse High bits	$58	$08	$00	data byte out
Read Extended Fuse Bits	$50	$08	$00	data byte out
Read Calibration Byte	$38	$00	$0b00 000bb	data byte out
Write Instructions				
Write Program Memory Page	$4C	000a aaaa	aa00 0000	$00
Write EEPROM Memory	$C0	adr MSB	adr LSB	data byte in
Write EEPROM Memory Page (page access)[1]	$C2	adr MSB	adr LSB	$00
Write Lock bits	$AC	$E0	$00	data byte in
Write Fuse bits	$AC	$A0	$00	data byte in
Write Fuse High bits	$AC	$A8	$00	data byte in
Write Extended Fuse Bits	$AC	$A4	$00	data byte in

Abbildung 6 - Befehlsübersicht Programmierinterface

Als Beispiel wird nun das EEPROM des AVR mega 16 an der Adresse 0x0010 mit 0xAA programmiert. Dafür wird zunächst das SPI Programmierinterface mit obiger Routine aktiviert. Anschließend wird zunächst der Befehl gesendet um das EEPROM zu schreiben (0xC0). Dann werden zwei Bytes gesendet, welche die zu schreibende Adresse im EEPROM beschreiben (0x00 und 0x10). Als letztes wird dann das zu schreibende Datenbyte gesendet (0xAA).

Um zu kontrollieren ob auch wirklich an stelle 0x10 der Wert 0xAA steht, wird nun das EEPROM dort wieder ausgelesen. Dafür wird wieder zunächst ein Befehl gesendet zum lesen des EEPROM (0xA0). Dann wieder die zwei Adressbytes (0x00 und 0x10). Nun wird einfach ein leeres Bytes geschickt, damit der Slave das Ergebnis ausgibt (0xAA).

Analog zu diesem Vorgehen kann man sowohl die Fuse Bits als auch das Flash des AVR lesen und schreiben.

Weiterentwicklung

Motorola hat SPI zu QSPI weiterentwickelt. QSPI enthält eine Art Warteschlange für zu sendende Daten bzw. für empfangene Daten. Es bedarf somit kein warten der Software bis ein Byte versendet wurde um ein weiteres zu senden. Die Software muss einfach nur mehrere Bytes an die Speicherstelle des Prozessors schreiben, an dem sich die Warteschlange befindet sowie die Anzahl der Bytes zum senden. Die Hardware übernimmt nun den kompletten Sendevorgang. Somit können mehrere Bytes ohne Softwarezutun gesendet und empfangen werden.

National Semiconstructor hat ebenfalls SPI adaptiert, es jedoch als Microwire bezeichnet. Die Weiterentwicklung von Microwire bezeichnet National einfach als Microwire+. Jedoch sind hier nur höhere Taktraten und damit Geschwindigkeiten spezifiziert worden.

Mit ein paar Zeilen Software lässt sich jedes SPI fähiges Gerät auch an ein QSPI Controller hängen. Somit ist QSPI abwärtskompatibel zu SPI.

Quellnachweis

ATMEL mega 16 – Handbuch
www.atmel.com

Freescale QSPI – Handbuch
www.freescale.com

Silicon Labs 8051 Handbuch
www.silabs.com

Wikipedia
www.wikipedia.de