Schwerpunktseminar Systemtechnik

Die Zweidrahtbussysteme I²C-Bus und SPI-Bus:

Eigenschaften, Protokolle, Anwendungen im Vergleich der beiden Systeme

Fachbereich Mathematik, Naturwissenschaften und Informatik

September 2008

Inhaltsverzeichnis:

Abbildungsverzeichnis:

1. Einleitung

1.1 Themenvorstellung

Die Online-Ressource Wikipedia definiert den Bus in Bezug auf die Informatik als „*ein Leitungssystem mit zugehörigen Steuerungskomponenten, das zum Austausch von Daten und/oder Energie zwischen Hardware-Komponenten dient*“.

In dieser Ausarbeitung beschäftige ich mich mit den seriellen, rechnerinternen Zweidrahtbussystemen SPI und I²C. Ich werde deren Eigenschaften, Protokolle und Anwendungen aufführen, wobei ich beim I²C deutlich näher ins Detail gehen möchte. Der einfache Grund hierfür ist die schiere Menge an Information, die man über diesen Bus finden kann und nicht etwa, weil er gegenüber dem SPI-Bus mehr Vorteile brächte. Beide Systeme haben ihre Vor- und Nachteile, die ich ebenfalls aufführen werde.

Beide Themenbereiche werden nacheinander abgehandelt und anschließend in einer kurzen Zusammenfassung miteinander verglichen. So werden die Merkmale, Stärken und Schwächen beider Zweidrahtbusse noch einmal übersichtlich aufgeführt.

1.2 Motivation

Zunächst möchte ich jedoch noch eine Erklärung zur Motivation abgeben, mich für das Thema Zweidrahtbussysteme entschieden zu haben.

Busse sind ein sehr wichtiger Bestandteil der Hardware. Ebenso wichtig wie wesentlich und gleichermaßen faszinierend ist allein der Gedanke, welche Datenmengen inzwischen in hochmodernen Rechnern über lediglich einige Leitungen übermittelt werden können.
Speziell die Zweidrahtbussysteme, die – wie der Name schon sagt – mit zwei Leitungen auskommen (bei SPI ist es nicht wirklich so, doch dazu später mehr) haben mein Interesse daher besonders geweckt. Ich wollte mehr darüber erfahren, mit welchen Methoden denn nun ganze Bytestränge zwischen Rechnerkomponenten übertragen werden, wie schnell dies geschieht und was dabei verhindert, dass sich zu sendende Daten nicht in die Quere kommen und kollidieren.

Die prinzipielle Nähe zur Übertragung von Daten in Rechnernetzen gab dabei den nötigen Anstoß. Das Internet könnte vom Konzept her mit einem gigantischen Bussystem verglichen werden und um einen alltäglicheren Vergleich zu finden, genügt es, sich auf die einfachen Busliniensysteme unserer Gesellschaft zu stützen. Menschen werden als Daten angesehen, Gebäude und Einrichtungen wie Ortschaften als Komponenten, von oder zu denen sie gesendet (respektive empfangen) werden.
Behält man diesen Gedanken im Hinterkopf und stelle sich nun eine solche Reise- und Übertragungslinie auf einer Platine innerhalb eines Rechners vor oder – noch kleiner – auf einer Krankenversicherungskarte beispielsweise, so ist es doch fast unglaublich, welche enorme Leistung diese simple Verdrahtung und ihre Konzepte erbringen. Und da die Zweidrahtbussysteme SPI und I²C einen so simplen Aufbau haben und einfachen Konzepten folgen, konnte es nur motivierend sein, sich mit diesem Thema zu beschäftigen.

1.3 Zweidrahtbussysteme – was ist das überhaupt?

Wie bereits kurz erwähnt, dient ein Bussystem der Verbindung zwischen rechnerinternen Komponenten, aber dieses Konzept wird auch für andere elektronische Systeme angewandt. Nun sollen (und müssen) die durch den Bus verbundenen Bausteine natürlich auch miteinander kommunizieren können. Dies würde speziell für moderne Systeme bedeuten, einen ganzen Strang an Leitungen zu ziehen, nur um der Kommunikation Willen.
Um das in diesem Maß scheinbar unrealisierbare Problem zu lösen, griff man die Idee auf, das System mit möglichst wenigen Leitungen zu versorgen und dennoch die gewünschte Kommunikation zu erreichen. Das Zweidrahtbussystem wurde entwickelt. Prinzipiell gilt, dass solche Systeme mit nur zwei Leitungen auskommen, um Daten zu senden und zu empfangen (bidirektional). Sie sind nur zweifach verdrahtet, woraus sich auch ihr Name ableitet.
I²C und SPI sind solche Zweidrahtbussysteme, wenn Letzterer auch im Grunde vier Leitungen nutzt (und daher manchmal auch als Vierdrahtbus bezeichnet wird), mal abgesehen von Masse und Versorgungsspannung, diese werden sowohl bei SPI als auch I²C nicht mitgerechnet. Trotzdem sind sie natürlich vorhanden und notwendig!
Zudem sind beide Bussysteme seriell aufgebaut, was sich aufgrund ihrer einfachen Verkabelung großer Beliebtheit erfreute. Und auch die mittlerweile stark gestiegene Leistungsfähigkeit sorgt dafür, dass sie den parallelen Bussen in nichts nachstehen. Man nehme beispielsweise einen weiteren „seriellen Vertreter“ wie USB, um nur einen Namen zu nennen.

2. SPI

2.1 Allgemeine Informationen über den SPI-Bus

SPI steht für *Serial Periphal Interface* (zu Deutsch: serielle Peripherie Schnittstelle) und wurde von Motorola entwickelt. Neben dieser Bezeichnung findet man häufig auch den Begriff *Microwire*, der Markenzeichen der Firma National Semiconductor ist. Funktionsprinzip für SPI und Microwire ist aber dasselbe.
Das System gilt als – wenn auch recht lockerer – Standard für einen synchronen seriellen Datenbus, auch wenn es von Seiten Motorolas niemals in einen vollständigen Standard überführt worden ist. Tatsächlich existieren nicht einmal Spezifikationen über Protokolle von SPI, lediglich die Hardware-Funktionsweise wurde beschrieben. SPI ist lizenzfrei, da es niemals mit Patenten belegt wurde.
Mittels SPI und ganz nach dem Master-Slave-Prinzip, welches im Folgenden anhand der Funktionen des SPI-Busses genauer erläutert wird, können digitale Schaltungen miteinander verbunden werden und kommunizieren. Hierbei ist der Mikrocontroller als Master anzusehen und weitere Peripherie-Chips (ICs, integrierte Schaltkreise/Schaltungen) werden als Slaves behandelt.
Seine Einsatzmöglichkeiten sind weiträumig, hauptsächlich verwendet man den SPI-Bus bei synchronen, seriellen Übertragungen zwischen Mikrocontrollern und ICs in der Audio- und Messtechnik.

2.2 Das Prinzip

SPI wurde speziell für den mit hoher Geschwindigkeit verwendeten Austausch von Daten zwischen verschiedenen ICs entwickelt. Aufgrund dieses Hochgeschwindigkeitsprinzips dürfen die Leitungen nicht zu lang werden, weshalb man ihn im Grunde nur auf PCBs antreffen wird (PCB = *printed circuit board,* Leiterplatte/Platine). Zu lange Leiterbahnen würden zu hohe Widerstände mit sich führen und den Bus unbrauchbar machen. Doch auf der Platine ist der SPI-Bus sehr gut untergebracht, denn man kann durch ihn praktisch so viele ICs miteinander verbinden wie man will. Dies macht ihn neben dem bereits erwähnten Merkmal des Datenaustausches zusätzlich sehr nützlich und interessant.
Zunächst soll aber erst einmal das grundlegende Prinzip des Busses erläutert werden. Es ist nicht schwer, aber man sollte es trotzdem verstehen, ehe man näher ins Detail geht.

Der SPI-Bus besitzt als Zweidrahtbussystem zwei Leitungen für Daten und eine dritte Leitung für den Takt, an denen jeder Teilnehmer angeschlossen ist:

- SDO (Serial Data Out, Datenausgang), auch MOSI genannt (Master Out Slave In)
- SDI (Serial Data In, Dateneingang), auch MISO genannt (Master In Slave Out)
- SCLK bzw. SCK (Serial Clock, Takt)

Die Taktleitung dient der Synchronisierung der Datenkommunikation und wird unabhängig davon, ob ein Slave-Gerät selektiert worden ist oder nicht, an alle Geräte gesendet.
Doch damit nicht genug. Hinzu kommt noch die SS- bzw. CS-Leitung (seltener auch STE). Diese Abkürzungen stehen für:

- SS = Slave Select
- CS = Chip Select
- STE = Slave Transmit Enable

Drei Bezeichnungen, hinter denen sich allerdings ein und dieselbe Leitung verbirgt. Im weiteren Verlauf der Ausarbeitung werde ich mich auf die Abkürzung SS beschränken.
Eine SS-Leitung ist low-aktiv (mit logisch 0 aktiv) und befindet sich zwischen jedem Slave und dem Master. Sie wird für das Selektieren des jeweiligen Slaves gebraucht. Es gibt aber auch Anwendungen, bei denen sich sämtliche Slaves eine Leitung teilen, wie es unten in **Bild [1]** zu sehen ist. **Bild [2]** zeigt das System mit jeweils einer SS-Leitung pro Slave. Auf beide Strukturen komme ich noch genauer zu sprechen.

In seiner Ausgangskonfiguration benötigt SPI sowohl zwei Daten- (SS und SCK) als auch zwei Steuerleitungen (SDI, SDO). In den meisten Fällen der im auf dem Markt befindlichen Bausteine sind die vier Leitungen so vorhanden. Es gibt aber auch Ausnahmen. So existieren Bausteine, bei denen SDI und SDO aus nur einer Leitung bestehen oder dass sogar eine von beiden gänzlich fehlt. Letzteres kommt vor allem bei Komponenten vor, die nicht konfiguriert werden müssen. Sie benötigen dann keine Eingangsleitung, ein Ausgang reicht. Wird ein solches Gerät selektiert, beginnt es sofort mit dem Aussenden der Daten. Hier würde also die SDI-Leitung nicht vorhanden sein, doch es kann auch vorkommen, dass Bausteine keinen

Datenausgang haben. Um ein Beispiel zu nennen, sei hier allgemein der LCD-Controller genannt. Ein Gerät wie dieses kann zwar konfiguriert werden, bietet dem Nutzer allerdings keine Möglichkeit des Ausgebens von Statusmeldungen oder des Rücklesen von Daten. Im Grunde zählt ein solcher Baustein nicht zu den SPI-fähigen Komponenten, doch weil sein Verhalten ansonsten dem der üblichen Bausteine vollkommen gleicht, wird er trotzdem als SPI-Komponente bezeichnet.

Der SPI-Bus arbeitet im Vollduplex-Betrieb, das heißt, Daten können gleichzeitig in beide Richtungen übertragen werden. Zudem existieren eine Menge Einstellmöglichkeiten für den Bus. Man kann beispielsweise entscheiden, mit welcher Taktflanke eingegeben bzw. ausgelesen werden soll oder auch, ob das MSB (most significant bit, höherwertigstes Bit) oder LSB (least significant bit, niederwertigstes Bit) zuerst übertragen werden soll. Des Weiteren sind Taktfrequenzen bis in den MHz-Bereich zulässig.
All diese Einstellungen sind unter anderem aufgrund einer sehr locker gehaltenen Spezifikation notwendig. Kurzum, in den Spezifikationen fehlen konkrete Festlegungen, weshalb viele verschiedene Geräte auf dem Markt erhältlich sind, welche zueinander nicht unbedingt kompatibel sein könnten. Daher ist es meist erforderlich, für jede Schaltung eine eigene Konfiguration des steuernden Mikrocontrollers (des Masters auf dem Bus) vorzunehmen.
Viele Mikrocontroller ermöglichen über SPI auch die so genannte In-System-Programmierung. Das bedeutet, dass eine logische Schaltung direkt innerhalb des laufenden Systems programmiert werden kann, ohne dass man sie vorher daraus entfernen muss. Das macht den gesamten Programmiervorgang schneller.

In einem SPI-Bussystem fungiert genau ein Teilnehmer als Master. Er erzeugt das Taktsignal SCK und legt durch Selektierung mittels der SS-Leitung fest, mit welchem anderen Baustein – man nennt diese dann Slaves – er kommunizieren will. Wird die Slave-Select Leitung eines Slave-Gerätes gegen Masse gelegt (beschreibt also den logischen Wert 0), wird dieser Slave aktiviert, „lauscht“ an SDO (bzw. MOSI) und legt seine Daten entsprechend dem Taktsignal an SDI (bzw. MISO) an. Nach diesem Prinzip wird somit ein Byte vom Master zum Slave und umgekehrt übertragen.

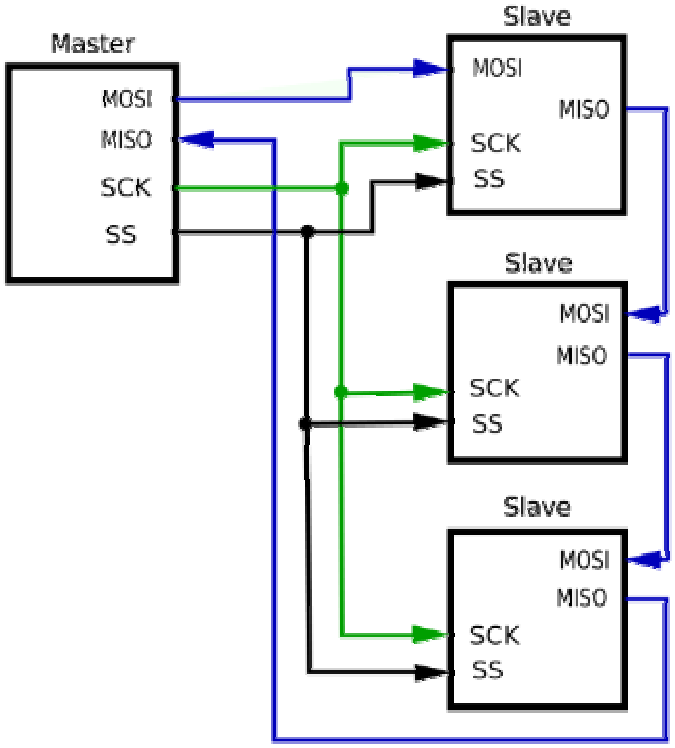

Bild [1]: SPI-Verbindung mit gemeinsam genutzter SS-Leitung, Struktur: Kaskadierung / Daisy-Chain
Quelle: Wikipedia

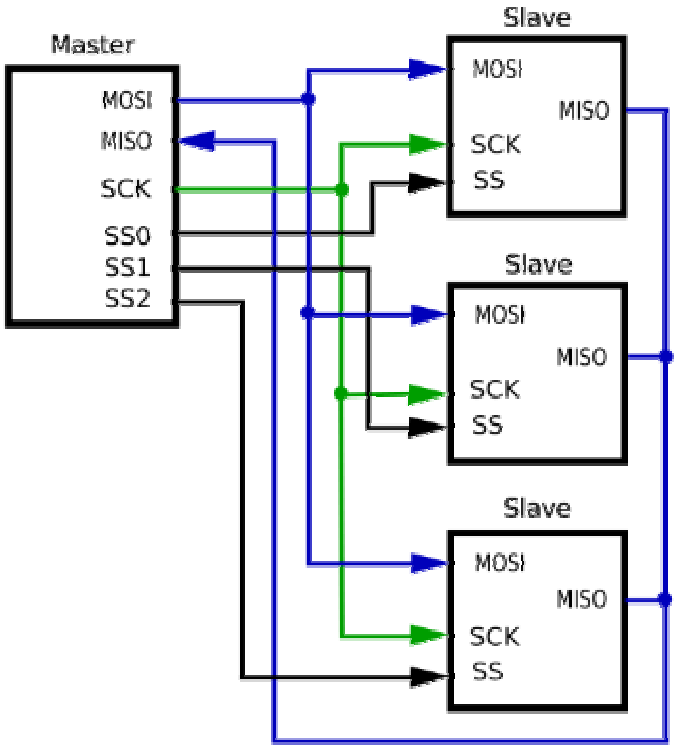

Bild [2]: SPI-Verbindung mit jeweils eigener SS-Leitung für die Slaves, Struktur: sternförmig
Quelle: Wikipedia

2.3 Protokolle des SPI

Wie auf dem obigen **Bild [1]** zu sehen, dient der Datenausgang SDO (auf den Abbildungen als MISO bezeichnet) nicht nur zum Rücklesen der Daten, sondern gleichzeitig auch als Dateneingang SDI (MOSI) des nachfolgenden Slave-Gerätes. Eine solche Hintereinanderschaltung einzelner Bausteine bezeichnet man als Kaskadierung oder auch Daisy-Chain-Konfiguration (daisy chain = Gänseblümchenkette). **Bild [2]** hingegen zeigt deutlich die gemeinsame Verbindung aller Ausgänge der Slaves mit Eingang dem des Masters.

Slave und Master sind die beiden Modi, mit denen Geräte am SPI-Bus betrieben werden können. Hinzu kommt noch die Möglichkeit, den SPI-Bus mit unterschiedlicher Polarität zu betreiben. Dazu im nächsten Abschnitt mehr.
Das Gerät, welches sich im Master-Modus befindet, gibt nicht nur das Taktsignal vor, sondern aktiviert auch über die SS-Leitung den Baustein, mit dem er kommunizieren möchte. Dieser befindet sich wie alle anderen übrigen am Bus angeschlossenen Komponenten im Slave-Modus. Auf einem SPI-Bus kann immer nur ein Gerät Master sein. Die Anzahl der Slaves begrenzt sich auf die Anzahl der SS-Leitungen. Somit ist es sowohl theoretisch als auch praktisch möglich, nahezu beliebig viele Slaves an den Bus anzubinden. Vor- und Nachteile dieses Gedankens werden später noch einmal aufgegriffen und erläutert.

Die Leitungen „Slave Select“ und „Clock“ des Masters werden somit als Ausgänge angesehen, die der Slaves sind Eingänge. Über sie erhält ein jeder Slave das Taktsignal und kann vom Master selektiert werden.
Die Komplexität eines Bausteins hängt von seiner Betriebsart ab. Somit kann in einem Baustein nicht nur das immer anzutreffende Grundprinzip des Schieberegisters Teil des Ganzen sein, sondern sogar ein eigenes Subsystem.
Befehle und Daten werden über die Datenleitung gesendet und gelangen von dort in eines der Schieberegister, wo sie nun vom Baustein verarbeitet werden können. Was hierbei auf jeden Fall zu berücksichtigen ist, ist die Länge eines solchen Schieberegisters. Diese ist nämlich nicht durch Spezifikationen vordefiniert und kann bei jedem Baustein unterschiedlich sein. Üblicherweise sind diese Register 8 Bit oder ein Vielfaches davon breit (ganzzahlige Variante), aber es gibt auch Schieberegister mit ungerader Bitanzahl. Verbindet man somit beispielsweise zwei 9-Bit-EEPROMs (elektrisch lösch- und wieder programmierbares ROM), so kann man Datenwörter mit einer Länge von 18 Bit speichern.
Um zu gewährleisten, dass nicht mehrere SPI-Komponenten Daten auf die Leitung ausgeben, geht der Ausgang eines nicht selektierten Bausteins automatisch in einen hochohmigen Zustand HIGH über. Werden mehrere SPI-Komponenten kaskadiert, so gilt: Sie müssen an derselben Slave-Select Leitung angeschlossen sein. Somit werden sie als ein einziger Slave angesehen.

Auf diese Weise können zwei durchaus sinnvolle Architekturen für Master- und Slave-Verbindungen des SPI-Busses entstehen, die bereits oben aufgezeigt worden sind. Die erste Struktur erreicht man durch Kaskadierung mehrerer Bausteine wie in Bild [1] zu sehen und die zweite Struktur wie in Abbildung [2] nennt man sternförmig.
Nun erschließt sich auch der Sinn einer gemeinsam genutzten SS-Leitung im Falle von Kaskadierung. Die Slaves werden als ein Ganzes, als ein großer Baustein angesehen, der durch die Verbindung vom Ausgang des einen Gerätes mit dem

Eingang des nachfolgenden Gerätes ein breiteres Schieberegister zur Verfügung stellen. Sie brauchen nicht einzeln selektiert zu werden und somit genügt eine SS-Leitung. Ganz im Gegensatz zur sternförmigen Struktur, wo jeder Baustein ein eigener Slave ist und demnach auch seine eigene Verbindung mit der Slave-Select Leitung, sowie eine eigene Zuführung von Taktleitung SCK und Dateneingang SDI benötigt.
Beide Strukturen können auch miteinander kombiniert werden.

Außerdem besteht auch die Möglichkeit, zwei Mikrocontroller über SPI miteinander zu verbinden. In diesem Fall stehen dem Anwender zwei Protokollvarianten zur Verfügung, um den Bus zu betreiben.
Die erste Variante nennt sich *Single-Master Protokoll* und sieht nur einen Master vor, wie es bei SPI prinzipiell üblich ist. Alle anderen Komponenten werden als Slaves betrachtet, auch weitere Mikrocontroller. Bei der zweiten Variante, dem so genannten *Multi-Master Protokoll*, hat jeder Mikrocontroller die Möglichkeit in die Rolle des Masters zu „schlüpfen“. Hierbei kann der als aktueller Master betrachtete Mikrocontroller den anderen ansprechen. Es ist jedoch zu gewährleisten, dass ein Controller ständig ein Taktsignal liefern muss. Mikrocontroller von Motorola stellen bei einem Multi-Master-System als zusätzliche Funktion hardwaremäßig eine Fehlererkennung bereit. Diese entdeckt beispielsweise, wenn mehrere SPI-Komponenten gleichzeitig als Master agieren wollen oder auch, wenn eine Kollision beim Schreiben entstand, sollten die Bausteine mit unterschiedlichen Polaritäten arbeiten. Die Funktion der Polaritäten wird im folgenden Abschnitt erläutert.
Die Slaves können wiederum auf zwei Arten ausgewählt werden. Die erste Möglichkeit bietet Slave-Selektierung durch Hardwareunterstützung an. Hier wird mittels der SS-Leitung gearbeitet und entsprechend selektiert. Bei Möglichkeit Zwei geschieht die Slave-Auswahl über in Frames gepackte IDs, deren Vergabe softwaremäßig realisiert wird.
Für beide Möglichkeiten gilt: nur der selektierte Slave öffnet seine Ausgabeleitung, alle anderen Slaves sind hochohmig geschaltet (HIGH) und somit deaktiviert. Der Ausgang des gewählten Slaves bleibt so lange geöffnet, wie dieser durch seine Adresse angesprochen wird.

2.4 SPI-Konfigurationen mittels Polaritäten

Motorola, Entwicklungsfirma des SPI, hat keine offizielle Spezifikation zu Protokollen der Datenübertragung herausgebracht und auch nicht fest vorgeschrieben, bei welcher Taktflanke (steigend oder fallend) denn nun Daten übernommen werden sollen. Im Laufe der Zeit haben sich aber in der Praxis vier Betriebsarten standardisiert, auch wenn sie von Motorola selbst nicht spezifiziert worden sind.
Diese Betriebsarten oder auch Modi ergeben sich aus den möglichen Kombinationen der Parameter *Clock Polarität (CPOL)* und *Clock Phase (CPHA).*
Somit ergeben sich vier mögliche Betriebsarten:

SPI-Modus	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Ist die Phase des Taktsignals gleich Null (CPHA = 0), so findet bei Taktpolarität 0 (CPOL = 0) die Datenübernahme mit steigender Taktflanke statt. Bei CPOL = 1 schlussfolgernd dann mit fallender Taktflanke.
Stellt man allerdings CPHA auf Eins ein, so drehen sich die Polaritäten um. In diesem Fall findet die Datenübernahme bei CPOL = 0 mit fallender und bei CPOL = 1 mit steigender Taktflanke statt.
Um es sich einfacher zu machen, kann man sich auch merken, dass die Daten immer dann bei steigender Taktflanke übernommen werden, wenn beide Parameter gleiche Einstellungen aufweisen (beide also 0 oder beide 1). Andernfalls erfolgt die Datenübernahme bei fallender Taktflanke.

Noch einmal zur kurzen Übersicht:

CPOL	CPHA	Datenübernahme bei …
0	0	steigender Taktflanke
0	1	fallender Taktflanke
1	0	fallender Taktflanke
1	1	steigender Taktflanke

Die eigentliche Datenübernahme geschieht jedoch erst, wenn zuvor die SS-Leitung auf LOW gezogen wurde (also eine logische 0 beschreibt). Zu beachten sei hier noch, dass der Slave bei Taktpolaritätsmodus Null (CPHA = 0) die Daten schon während des Runterziehens der Slave-Select Leitung den Datenausgang legt. Auf diese Weise wird gewährleistet, dass der Master sie beim ersten Flankenwechsel bereits übernehmen kann.
Befindet sich Polarität allerdings auf Eins (CPHA = 1), so legt der Slave die Daten erst beim ersten Flankenwechsel an, damit der Master sie dann beim zweiten Flankenwechsel übernehmen kann. Der Master selbst legt seine Daten immer zum gleichen Zeitpunkt an, nämlich immer kurz nach der LOW-Flanke des Taktsignals SCK (wenn der Takt die logische 0 beschreibt).
Pro Taktzyklus wird ein Bit übertragen, es sind also 8 Zyklen nötig, um ein Byte und somit eine vollständige Übertragung zu transferieren. Natürlich ist es auch möglich, mehrere Bytes hintereinander zu übertragen. Auch hier gibt es keine Festlegung, ob zwischen den einzelnen Bytes die SS-Leitung kurzfristig auf HIGH (logisch 1) gezogen werden muss.
Eine Datenübertragung endet, sobald die SS-Leitung endgültig zurück auf HIGH gesetzt wird.

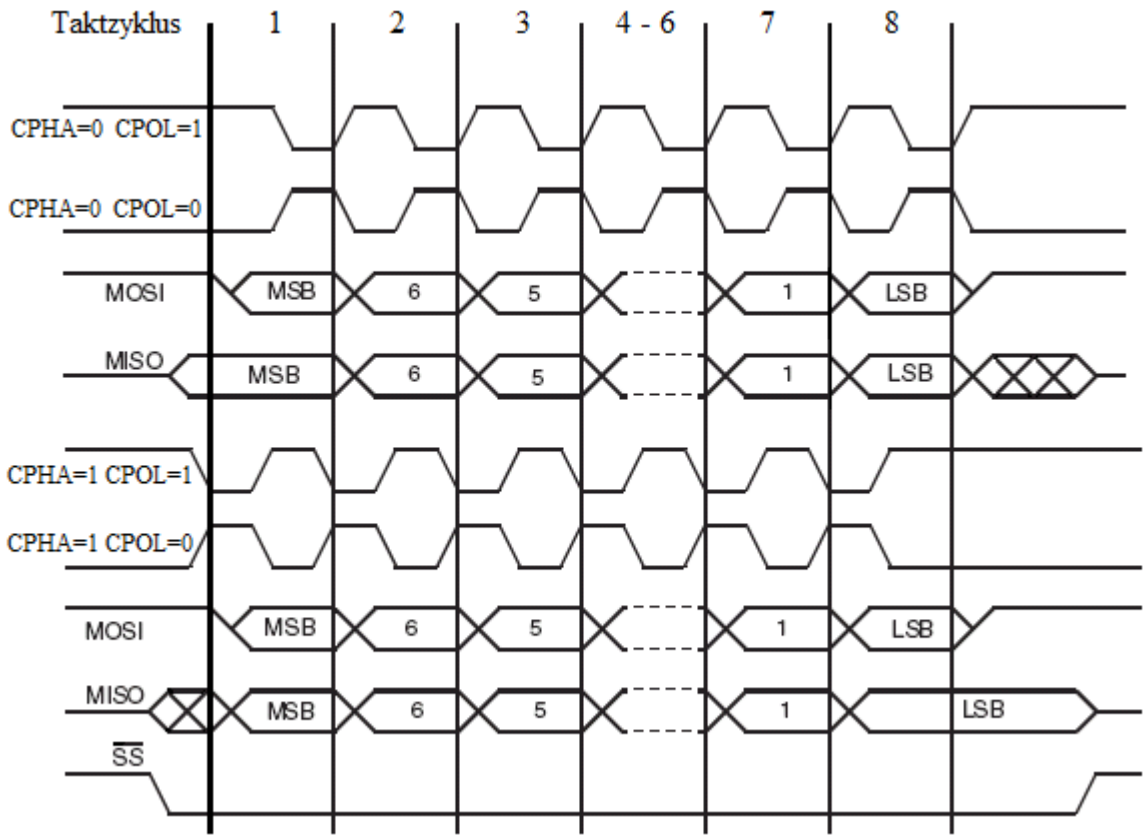

Bild [3]: Datenübertragung in allen Modi von CPOL und CPHA
Quelle: Wikipedia

2.5 Datenübertragung beim SPI

Da nun die prinzipielle Funktionsweise und die möglichen Einstellungen am SPI-Bus bekannt sind, komme ich zur eigentlichen Kommunikation auf dem Bus: der Datenübertragung.
Um überhaupt erst einmal eine solche Kommunikation zu beginnen, muss der Master zuerst den Takt konfigurieren, indem er eine Frequenz einsetzt, die kleiner oder gleich der maximalen Frequenz ist, die die Slave-Geräte noch unterstützen (meist im Bereich von 1-70 MHz).
Die SS-Leitung des zu selektierenden Slaves wird vom Master nun auf LOW gezogen. Es kann vorkommen, dass der Master hierbei einen vorgegebenen Zyklus lang warten muss, denn in manchen Fällen (beispielsweise bei einer Analog-zu-Digital-Wandlung) ist eine solche Warteperiode Voraussetzung. Anschließend startet er den Ausgabezustand des Taktzyklus.

Synchron zum Taktsignal des Masters werden die Daten an den Datenleitungen ausgegeben, was über ein Schieberegister realisiert wird. Das höchstwertigste Bit MSB wird zuerst ausgegeben (sofern die Einstellungen nicht verändert wurden, was bei einigen Geräten möglich ist), die niederwertigen Bits folgen. Ein Datenwort beträgt bei SPI 8 Bit bzw. 1 Byte.
Die empfangenen Daten liegen nach der Übertragung im gleichen Register vor, wie die Sendedaten. Es existiert somit nur ein Register für Sende- und Empfangsdaten. Mithilfe von Flags und Statusregistern, ferner auch über Interrupts, kann festgestellt

werden, ob die Übertragung abgeschlossen ist. Anschließend wird SS wieder auf HIGH gesetzt.

Während eines SPI-Taktzyklus' treten also folgende Ereignisse bei der Datenübertragung ein:

- Der Master sendet ein Bit an den Dateneingang des Slave (SDI-/MOSI-Leitung). Der Slave liest das Bit von dieser Leitung ein.
- Der Slave sendet ein Bit von seinem Datenausgang (SDO-/MISO-Leitung) und der Master liest es über diese Leitung ein.

Da SPI im Vollduplex-Betrieb arbeitet, geschieht das Senden und Empfangen von Daten gleichzeitig über beide Leitungen SDI und SDO. **Bild [4]** verdeutlicht noch einmal grafisch eine solche Datenübertragung:

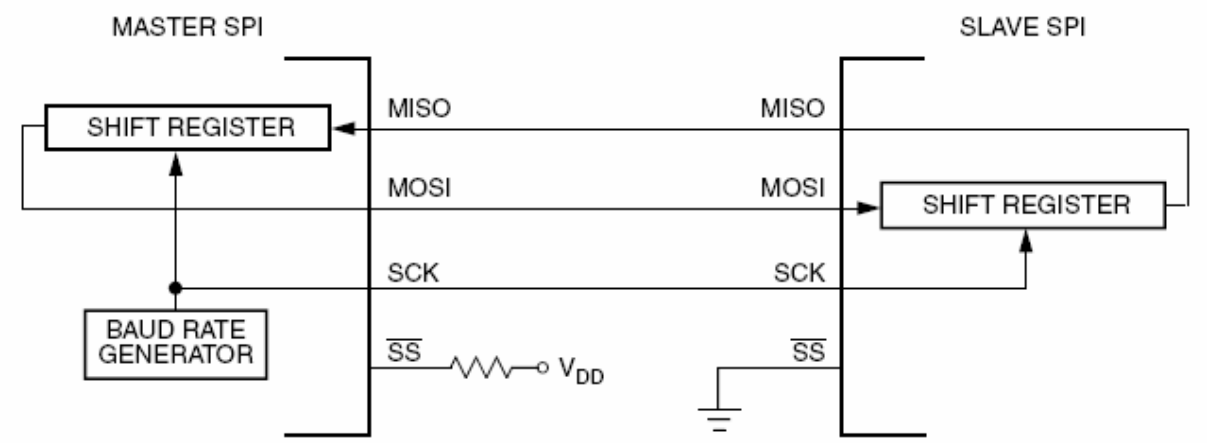

Bild [4]: Datenübertragung zwischen Master und Slave
Quelle: Freescale SPI Handbuch

2.6 Anwendungsgebiete des SPI-Busses

SPI wird hauptsächlich zur Anbindung von Komponenten an einen Mikrocontroller bzw. zur Verbindung mehrer Mikrocontroller untereinander verwenden, um dies noch einmal in Erinnerung zu rufen. Man sollte dabei beachten, dass sich alle Bausteine auf der gleichen Platine befinden sollten, da sich SPI für platinenübergreifende Verbindungen nicht wirklich eignet. Auch für die Verbindung zu externen Komponenten ist SPI komplett ungeeignet, da es keinerlei Störschutz gegen externe Signale gibt.

SPI wird für folgende Peripherietypen eingesetzt:

- Wandler (Analog-Digital und Digital-Analog)
- Speicher (EEPROM, Flash)
- Real Time Clocks (RTC)
- Sensoren (Temperatur, Druck)
- Sonstiges wie Signalmixer, Potentiometer, LCD-Controller, USB-Controller …

Eine der bekanntesten SPI-Komponenten dürfte die MMC Speicherkarte (MultiMediaCard) sein. Sie findet in vielen verschiedenen Arten von Geräten Anwendung (Digitalkamera, PDAs, Handys …). SPI ist nicht auf eine spezielle Art von Peripherie festgelegt und kann somit weiträumig eingesetzt werden.

3. I²C

3.1 Allgemeine Informationen über den I²C-Bus

Der I²C-Bus ist ein serieller, synchroner Zweidraht-Bus, der 1979 von Philips Semiconductors (seit 2006 NXP) entwickelt wurde. I²C steht für Inter-Integrated Circuit (zu Deutsch: interne integrierte Schaltung) und wird I-Quadrat-C, I-square-C oder fälschlicherweise I-Two-C ausgesprochen. Rein aus Lizenzgründen führen andere Hersteller diesen Bus unter dem Namen TWI (Two Wire Interface).
Inzwischen wird I²C als Industriestandard für Steuerungs-, Diagnose- und Überwachungslösungen in vielen eingebetteten Systemen (embedded systems) gezählt. Gründe hierfür sind die einfache Implementierung, niedrige Kosten und eine Bus-Geschwindigkeit bis zu 3,4 MBit/s.
Sein ursprünglicher Zweck war es, auf einfache Weise Kommunikation zwischen verschiedenen integrierten Schaltungen (ICs) zu ermöglichen und bereitzustellen. In vielen modernen elektronischen Systemen ist eine solche Kommunikation unter den einzelnen Bausteinen nämlich längst eine Notwendigkeit geworden. Die Entwicklung des I²C-Busses verhinderte, dass man heutzutage für eine solche Kommunikation zig Leitungen durch das gesamte System legen muss.
Der I²C-Bus wird Zweidraht-Bus genannt, weil er – abgesehen von Leitungen für Masse und Versorgungsspannung – mit zwei bidirektionalen Leitungen auskommt:

- SDA (serial data), über die die eigentlichen Daten seriell übertragen werden
- SCL (serial clock) sendet die Taktimpulse

Der I²C-Bus besitzt eine im Laufe der Jahre erweiterte und ergänzte Spezifikation, die in ihrer ursprünglichsten Form (veröffentlicht 1982) eine maximale Bus-Geschwindigkeit von 100 kHz vorsah. Für heutige Anwendungen ist dies nicht immer ausreichend, aber die Spezifikationen sind auch überarbeitet worden. Neben neuen Angaben zur Maximalgeschwindigkeit finden sich nun auch Definitionen zur Adressierung und weiteren Punkten, die in den folgenden Abschnitten Erläuterung finden.

Die maximale Länge des I²C-Busses hängt von der Last des Busses und der Geschwindigkeit ab, mit der man ihn laufen lässt. In typischen Anwendungen ist die Länge ein paar Meter (2,74 – 3,65 m).
Hierbei sollte man aber vor allem der Menge an Störgeräuschen Beachtung beimessen, die durch zu lange Leitungen aufgenommen wird. Dieses Rauschen kann das über den Bus übertragene Signal dermaßen stören, dass es unlesbar wird.

Der I²C-Bus arbeitet nach dem Master-Slave-Prinzip. Das heißt, dass die Geräte auf dem Bus entweder Master oder Slave sind.
Der Master ist immer das Gerät, das den Takt auf der SCL-Leitung erzeugt und betreibt. Die Slaves reagieren auf den Master, sie selbst können keine Datenübertragung einleiten. Dies obliegt nur dem Master-Gerät. Natürlich können aber beide Geräte-Kategorien, also sowohl Master als auch Slave, Daten über den

I²C-Bus übertragen, eine solche Übertragung wird allerdings immer vom Master kontrolliert.

Bild [5] zeigt ein vereinfachtes Schaltbild eines I²C-Verbunds zwischen zwei Geräten. Es zeigt zudem alle Faktoren, die für I²C von Bedeutung sind.

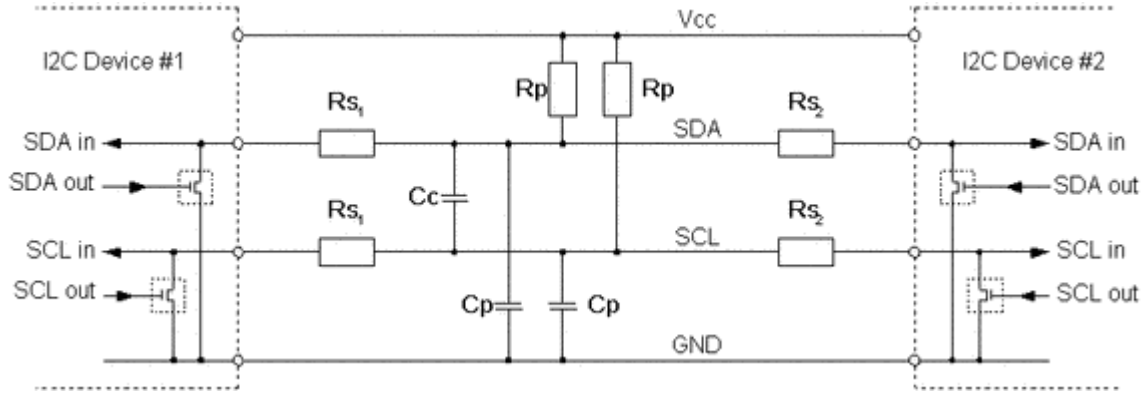

Bild [5]: Schaltbild eines I²C-Verbunds zweier Geräte
Quelle: I²C-bus.org

Zugehörige Abkürzungen:

VCC	Versorgungsspannung, typischerweise von 1,2 – 1,5 V
GND	Masse (Ground)
SDA	I²C Datenleitung (serial data)
SCL	I²C Taktleitung (serial clock)
Rp	Pull-up Widerstand, auch I²C Terminator/Endwiderstand
Rs	serieller Widerstand
Cp	Leitungskapazität (eines Kondensators)
Cc	Kondensator

Man beachte! Werden die Pull-up Widerstände vergessen, befindet sich der gesamte I²C-Bus ständig im Zustand LOW. Die angeschlossenen Bausteine gehen dann davon aus, dass der Bus aktiv genutzt wird und kein Gerät wird einschreiten, um den scheinbar aktiven Bus zu übernehmen. Das System hängt dann.

Prinzipiell funktioniert die Hardware des I²C-Busses folgendermaßen:
Der Bus überträgt Daten und Takt über die Leitungen SDA und SCL. Hierbei sei noch angemerkt, dass beide Leitungen *open-drain* sind. Das heißt, dass I²C-Geräte diese Leitungen nur low-aktiv betreiben können (wenn sie die Leitungen also soweit „runterziehen“, dass sie eine logische 0 beschreiben) oder offen lassen. Dies bedeutet dann nichts Anderes, als dass I²C-Geräte mit den Leitungen nichts tun, denn sie sind nicht in der Lage, solche Leitungen zurück in den HIGH-Zustand zu versetzen. Darum kümmern sich die Pull-up Widerstände.
Der Endwiderstand Rp zieht die Leitung auf VCC hoch, wenn kein Gerät am I²C-Bus sie runterzieht. Zusammen mit der Leitungskapazität Cp beeinflusst der Endwiderstand das zeitliche Verhalten der Signale auf SDA und SCL. Während die Geräte am Bus die Leitungen bei begehrter Nutzung runterziehen, ist es Aufgabe des Pull-up Widerstandes Rp die Signale zurück auf ihren HIGH-Pegel zu bekommen. Im Allgemeinen weist ein solcher Pull-up Widerstand Werte zwischen 1 und 10 kΩ (Ohm) auf. Dies ist der Grund für das charakteristisch sägezahnartige

Aussehen der I²C-Signale (siehe **Bild [6]**). Jeder „Zahn“ zeigt hierbei die Entlade-/Lade-Charakteristik einer Leitung bei steigender und fallender Flanke.

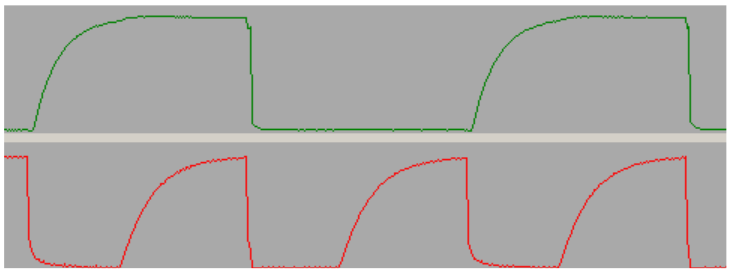

Bild [6]: SDA (oben) und SCL (unten), Rp = 10 kΩ, Cp = 300 pF, SCL-Takt läuft mit 100 kHz (normal)
Quelle: I²C-Bus.org

Leistungskapazität und Endwiderstand beschränken allgemein die maximale Datenrate, welche über SDA und SCL übertragen werden kann. Hohe Leistungskapazität Cp kann durch einen niedrigen Endwiderstand Rp abgeglichen werden oder umgekehrt.
Standardmäßig weist ein I²C-Leistungskapazitätskondensator einen Maximalwert von 400 pF auf. Mit einem entsprechenden Endwiderstand ist es allerdings möglich, mit höheren Kapazitäten zu arbeiten, dies wird jedoch nicht empfohlen, da es Geräte am Bus oder den Bus selbst beschädigen könnte.

3.2 Voraussetzungen für I²C-Bus-Geräte

Um einwandfreies Arbeiten in verschiedenen Umgebungen zu gewährleisten, wurden mithilfe der I²C-Spezifikationen einige Voraussetzungen an I²C-Geräte gestellt. Worauf auf jeden Fall geachtet werden sollte, wären folgende Punkte:

- Beide Leitungen, SDA und SCL, müssen open-drain sein und dürfen von keinem am I²C-Bus angeschlossenen Gerät HIGH betrieben werden.
- In den meisten I²C-Bussen müssen LOW- und HIGH-Eingangsspannungspegel von SDA und SCL von der Versorgungsspannung VCC abhängen. Beispielsweise wird ein SDA-Spannungspegel von 1,1V in einem I²C-Bussystem mit VCC = 5V als LOW und in einem I²C-Bussystem mit VCC = 1,2V als HIGH interpretiert.
- Die SCL- und SDA-Signale müssen von so genannten *Schmitt-Trigger-Eingängen* abgefragt werden und zwar mit einer besonderen Hysterese. Dieser Begriff bezeichnet ein System, dessen Ausgangsgröße nicht allein von Art der Eingangsgröße abhängig ist, sondern auch vom Entwicklungsgang, welchen die Eingangsgröße hatte. Dadurch beweist das System seine Pfadabhängigkeit.

Kurze Nebeninformation: Schmitt-Trigger
Der Schmitt-Trigger (nach seinem Erfinder Otto Schmitt) ist eine elektronische Komparator-Schaltung, die auch als IC vorhanden sein kann. Komparator-Schaltung heißt, dass sie zwei analoge Spannungen vergleicht und als Schwellwertschalter fungiert. Er wird hierbei ausgelöst, wenn die von einem Sensor gemessene physikalische Größe einen festgelegten Grenzwert über- oder unterschreitet.
Der Schmitt-Trigger reagiert auf eine langsame Änderung der Eingangsspannung mit einer schlagartigen Änderung des logischen Zustands am Ausgang.

- Spitzen in SCL- und SDA-Signalen müssen auf einen besonderen Betrag gefiltert sein. Dies gilt jedoch nur für High-Speed-I²C (dazu später mehr).

- Einrichtung von Haltezeiten; diese beziehen eine festgelegte maximale Taktrate ein. Beispielsweise 100 kHz für normale Geschwindigkeit, 400 kHz für Fast-Mode (dazu später mehr) usw.

Im Gegensatz zu I²C-Software-Implementierungen in Mikrocontrollern erfüllen die meisten serienmäßigen I²C-ICs diese Voraussetzungen. Doch bei den Mikrocontrollern muss das nicht unbedingt ein Problem sein, solange das System eine solche Unterstützung der I²C-Geräte am Bus nicht verlangt. Trotzdem sollte man sich im Falle auftretender Probleme dies als mögliche Ursache in Erinnerung rufen.

3.3 Taktrate, Clock Stretching und Taktsynchronisation

3.3.1 Taktrate

Der Master am I²C-Bus gibt den Takt auf dem Bus an. Er bestimmt die Taktrate und somit auch den Geschwindigkeitsmodus, mit dem der Bus betrieben wird. Diese sind nur durch ihren maximal erlaubten Bustakt begrenzt, aber wohl voneinander unterscheidbar. Sofern vom Master-Gerät unterstützt, können auch langsamere Taktraten verwendet werden.
Die nachfolgende Tabelle zeigt die maximalen Taktraten pro Modus auf. Auf die Modi komme ich in späteren Abschnitten noch einmal einzeln drauf zu sprechen.

Modus	**maximale Taktrate**
Standard Mode	100 kHz
Fast Mode	400 kHz
Fast Mode Plus	1 MHz
High Speed Mode	3,4 MHz

3.3.2 Clock Stretching

Es kann schon einmal vorkommen, dass ein Slave-Gerät die Daten nicht in der Zeit verarbeiten (senden oder empfangen) kann, die der Master durch seinen Takt vorgibt. Um zu verhindern, dass es durch solch ein Problem zu möglichem Datenverlust kommt, gibt es für den Slave noch die Möglichkeit des so genannten *Clock Stretchings* (zu Deutsch: Taktstreckung).
Ein I²C-Slave darf mittels dieses Mechanismus' den Takt unten halten (auf LOW ziehen, so dass SCL eine logische 0 beschreibt), wenn er die Busgeschwindigkeit reduzieren muss. Auf diese Weise schützt ein Slave sich auch vor einer zu schnellen Adressierung durch den Master, denn es gibt Slave-Geräte, die sich manchmal tatsächlich „ausruhen" müssen, um einwandfreies Arbeiten zu gewährleisten.
Es gibt allerdings auch Geräte, die das Clock Stretching ignorieren oder eine von sich aus festgelegte Zeit akzeptieren, die der Slave für Clock Stretching verwenden darf (meist im Bereich von einer Millisekunde). Im Falle solcher Geräte sollte man immer berücksichtigen, dass es unter diesen Umständen dann zu fehlerhaften Datenübertragungen kommen kann!
Die I²C-Spezifikationen haben übrigens keine maximale Zeitbeschränkung für Clock Stretching festgelegt. Jedes Gerät kann die SCL-Leitung so lange unten halten, wie es dies für nötig hält.

3.3.3 Taktsynchronisation

Damit der Datentransfer auf dem I²C-Bus problemlos abläuft, muss sichergestellt sein, dass auch der langsamste Baustein den Takt korrekt verarbeiten kann. Dafür wird die Taktsynchronisation verwendet.
Hierbei kommt auch die so genannte „Wired-AND“-Verknüpfung des Busses zum Tragen.

Kurze Informationen zu: Wired-AND-Verknüpfung
Eine Wired-AND-Verknüpfung (zu Deutsch: verdrahtetes UND) entsteht, wenn zwei oder mehr Ausgänge so miteinander verbunden werden, dass die digitale Schaltung wie eine logische UND-Schaltung wirkt. Hierfür benötigt man meist keine zusätzliche Hardware, weil spezielle Konfigurationen der Ausgänge (z.B. open-drain) verwendet werden.

Die Taktsynchronisation kommt folgendermaßen zum Einsatz: Eine fallende Flanke auf der Taktleitung SCL veranlasst alle am Bus angeschlossenen Geräte einen internen Zähler zu starten und ihren SCL-Ausgang auf LOW zu setzen. Wenn dieser interne Zähler abgelaufen ist – wenn der jeweilige Baustein also seine Verarbeitung abgeschlossen hat –, setzt der Baustein seinen SCL-Ausgang wieder auf HIGH. Nun greift die Wired-And-Funktionalität mit ein, denn die Taktleitung nimmt erst wieder den Zustand HIGH an, wenn der letzte Zähler abgelaufen ist und somit der letzte Baustein am Bus aufhört, die SCL-Leitung auf LOW zu ziehen. Somit müssen alle schnelleren Geräte warten, bis auch das langsamste Gerät die Verarbeitung beendet hat. Bei der darauf folgenden steigenden Taktflanke besteht somit kein Unterschied mehr zwischen dem Takt der einzelnen Bausteine und dem Takt auf dem I²C-Bus und die Geräte starten einen internen HIGH-Zähler. Das erste Gerät, dessen Zähler abgelaufen ist, legt seinen SCL-Ausgang auf LOW und zieht damit die gesamte Taktleitung auf LOW.
Auf diese Weise wird ein synchroner Takt generiert, dessen LOW-Phase durch das Gerät der längsten LOW-Phase und dessen HIGH-Phase durch das Gerät mit der kürzesten HIGH-Phase bestimmt wird. Natürlich muss hierbei die maximale Taktrate eingehalten werden.

3.4 Multi-Master-Bussystem und Arbitration

3.4.1 Multi-Master-Bussystem

Bei I²C besteht die Möglichkeit, mehrere Master an einem Bus zu betreiben, denn der I²C-Bus ist ein Multi-Master-Bussystem. Das bedeutet, dass mehr als ein IC am Bus eine Datenübertragung einleiten kann. Die Spezifikationen des I²C-Bussystems legen fest, dass das Gerät, welches die Übertragung einleitet, als Master auf diesem Bus bestimmt wird. Folglich werden zur selben Zeit alle anderen Geräte auf dem Bus als Slaves angesehen.
Wenn man den I²C-Bus als Multi-Master-System betreibt und somit Multi-Master- und Synchronisationsprotokoll nutzt, sollte man darauf achten, dass die Taktleitung bidirektional ist. Bei nur einem Master ist eine bidirektionale Leitung nicht erforderlich, denn der Takt wird immer nur von diesem einen Gerät erzeugt. Bei Multi-Master-Systemen ändert sich dies allerdings. In einem solchen Fall muss jedes Gerät es handhaben und entsprechend kooperieren, dass ein anderes Gerät gerade über den Bus kommuniziert und dieser daher beschäftigt ist.

Ein solcher Master muss also den Regeln der Arbitrationslogik folgen können und erkennen können, wenn der Bus beschäftigt ist. Dann darf es die Kommunikation auf dem Bus nämlich nicht unterbrechen, sondern muss auf eine Stopp-Bedingung warten, bevor es selbst einen Kommunikationsversuch beginnen kann.
Wenn man plant, ein Multimastergerät auf einem Bus zu benutzen, müssen alle anderen an diesem Bus angeschlossenen Master ebenfalls multimasterfähig sein. Ein Master, der diese Fähigkeit nicht besitzt, wird als Singlemaster bezeichnet. Wenn dieser zusammen mit einem Multimaster an einem Bus hängt, könnte der Singlemaster den anderen unterbrechen und unvorhergesehene Probleme verursachen.

3.4.2 Arbitration

Verschiedene I²C-Multimastergeräte, die am selben Bus verbunden sind, arbeiten auf diesem natürlich gleichzeitig. Durch andauerndes Überwachen der beiden Leitungen SDA und SCL auf Start- und Stopp-Bedingungen können sie entscheiden, ob der sich der Bus gerade im Ruhezustand befindet oder nicht. Ist der I²C-Bus beschäftigt, verzögert der Master eine ausstehende Übertragung bis eine Stopp-Bedingung anzeigt, dass der Bus wieder frei ist.
Es kann allerdings auch passieren, dass zwei Master eine Übertragung zur selben Zeit starten. Während der Übertragung überwachen die Master konstant SDA und SCL. Wenn einer von ihnen entdeckt, dass die Datenleitung SDA LOW ist, obwohl sie eigentlich im HIGH-Zustand sein sollte, nimmt er an, dass ein anderes Mastergerät aktiv ist und stoppt sofort seine Übertragung. Diesen Prozess nennt man Arbitration.
Sollten zwei Mastergeräte wirklich absolut gleichzeitig ihre Kommunikation beginnen, so gewinnt dasjenige, welches mehr Nullen auf den Bus schreibt (oder das langsamere Gerät).

Im Allgemeinen versteht man unter Arbitration die möglichst gerechte Zuteilung von Ressourcen auf verschiedene Benutzer/Geräte.

3.5 Das I²C-Bus-Protokoll (Datenübertragung)

Bevor ein Baustein mit einer Datenübertragung beginnen kann, muss sichergestellt sein, dass der I²C-Bus frei ist. Dieser gilt als frei, wenn nach dem Ende einer vorangegangenen Datenübertragung beide Leitungen SDA und SCL für mindestens 4,7 ns einen HIGH-Pegel aufweisen.
Hat zuvor keine Datenübertragung stattgefunden, so genügt die einfache Feststellung: Der Bus ist frei, wenn Daten- und Taktleitung HIGH-Pegel besitzen.

3.5.1 Bitgültigkeit

Während einer Übertragung müssen die Daten natürlich gültig sein. Dies ist nur dann der Fall, wenn sich ihr logischer Pegel beim Transfer nicht ändert, während die Taktleitung HIGH ist. Ausnahmen sind hier die Signale *Start, Stopp, wiederholter Start*, auf die ich noch eingehen werde.
Im Bereich eines als gültig bezeichneten Zeitabschnittes darf sich also der Pegel der Datenleitung SDA nicht plötzlich beispielsweise von LOW auf HIGH ändern oder umgekehrt. Ein solcher Zeitabschnitt wird auch als Zeitfenster bezeichnet und ist mindestens 4 µs lang. **Bild [7]** verdeutlicht diese Regelung noch einmal:

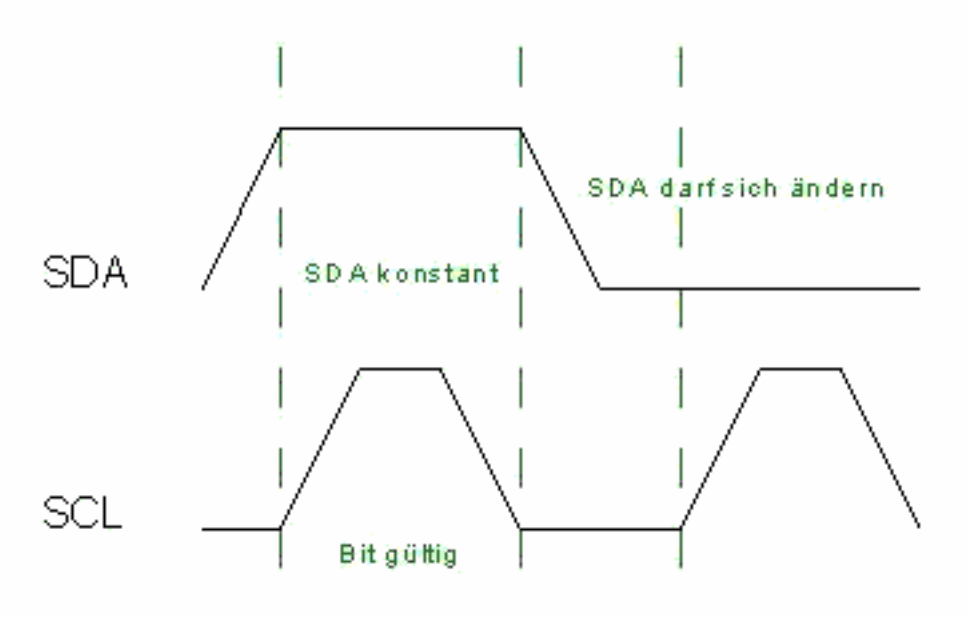

Bild [7]: Wann ist ein Bit gültig?
Quelle: Roboternetz.de

3.5.2 Start- und Stopp-Bedingung

Die Ausnahme der eben erläuterten Gültigkeitsregel wird bewusst genutzt, um die Start- und Stopp-Bedingung einer Datenübertragung zu kennzeichnen. Jeglicher Datentransfer wird vom Master durch eine *Start*-Bedingung eingeleitet und mittels einer *Stopp*-Bedingung beendet.
Die Start-Bedingung, kurz: START, erzeugt der Master durch Änderung des Pegels auf der Datenleitung SDA von HIGH auf LOW, während die Taktleitung SCL im Zustand HIGH verbleibt (siehe **Bild [8]**, unten). Diese Kombination ist während einer Datenübertragung einmalig und somit unverwechselbar für alle Bausteine. Sämtliche Komponenten erkennen diese spezielle Situation dann als Beginn einer Datenübertragung.

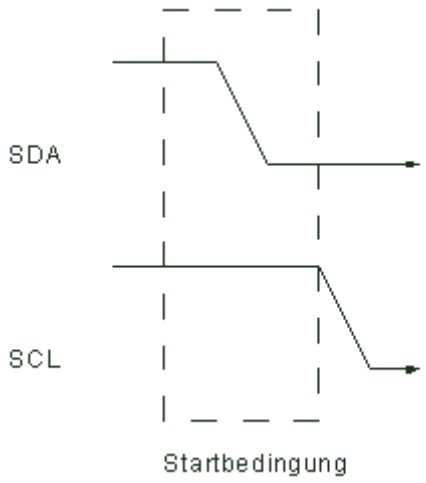

Bild [8]: Startbedingung
Quelle: Roboternetz.de

Nach dieser Start-Bedingung erfolgt eine bitweise serielle Übertragung der Daten, die nach jedem Byte (nach jedem 8. Bit demnach) vom ausgewählten Slave bestätigt wird.
Um eine solche Übertragung schließlich ordnungsgemäß zu beenden, generiert der Master eine *Stopp*-Bedingung. Er sendet also ein STOPP, das sich dadurch kennzeichnet, dass der Pegel der SDA-Leitung von seinem LOW- in den HIGH-Zustand wechselt, während die Taktleitung SCL noch immer auf HIGH ist und auch bleibt (siehe **Bild [9]**, unten). Diese Kombination gilt ebenfalls als einmalig, während einer Datenübertragung und wird von allen Bausteinen als Ende eines solchen Transfers erkannt.

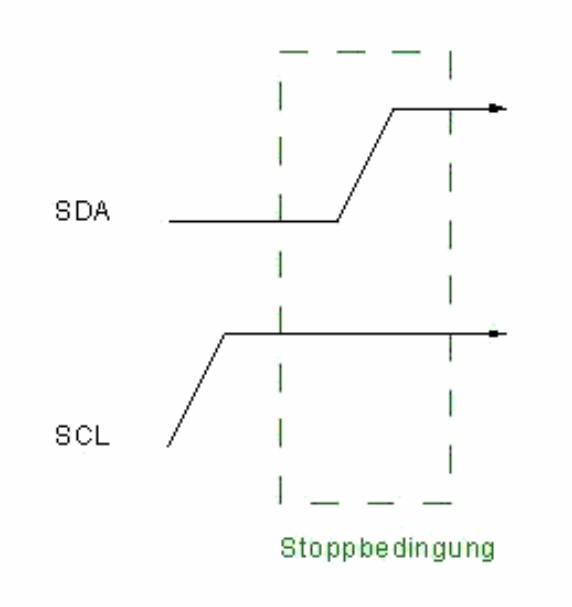

Bild [9]: Stopp-Bedingung
Quelle: Roboternetz.de

Die Pegel beider Leitungen SDA und SCL befinden sich anschließend im Zustand HIGH, was nichts Anderes heißt, als dass der Bus für eine erneute Kommunikation freigegeben ist.

Da die Stopp-Bedingung gleichzeitig auch eine Freigabe des Busses bedeutet, könnte zu diesem Zeitpunkt auch ein anderer Master den Bus übernehmen (denn I²C ist wie bereits erwähnt multi-master-fähig). Aus diesem Grund gibt es auch den Start ohne vorherigen Stopp, die so genannte *wiederholte Startbedingung (repeated start)*.
Ihr Zweck ist es, kombinierte Lese-/Schreiboperationen zu einem oder mehr Geräten zu senden, ohne den Bus freizugeben. Dadurch wird garantiert, dass die Bearbeitung nicht unterbrochen wird.
Ohne Rücksicht auf die Anzahl der gesendeten Start-Bedingungen während einer Übertragung, muss diese nur mit genau einer Stopp-Bedingung beendet werden.

Die Abfolge einer Übertragung mit wiederholter Start-Bedingung sieht dann folgendermaßen aus:

```
I2C START
I2C Send Write-Adress
I2C Send Argument
I2C Start oder Repeated Start
I2C Send Read-Adress
I2C Read Data
...
I2C Stopp oder Release Bus
```

3.5.3 Übertragung eines Bytes

Wurde eine Start-Bedingung vom Master erzeugt, kann die eigentliche Übertragung der Daten beginnen, die byteweise erfolgt. Das erste Byte beinhaltet die Adresse des Slaves, mit dem der Master kommunizieren will. Bisher haben nämlich alle angeschlossenen Slaves am Bus „gelauscht“, alarmiert durch die ausgesandte Start-Bedingung. Die Slaves hören nun also den Bus ab und erwarten das erste Byte mit der Adresse eines Slaves. Der angesprochene Slave wird auf den Kommunikationswunsch des Masters reagieren, während alle anderen Geräte die weitere Übertragung auf dem Bus ignorieren und auf die nächste Transaktion warten, die dann ja für einen von ihnen bestimmt sein könnte.

Hier noch einmal die Beschreibung der Abfolge:

1. Im Ruhezustand befinden sich die Leitungen SDA und SCL auf HIGH. Der Bus ist frei.
2. Die Übertragung beginnt mit der Start-Bedingung, indem der Master die Datenleitung SDA von HIGH auf LOW setzt. Die Taktleitung SCL verbleibt im HIGH-Pegelzustand. Nun ist der Bus besetzt.
3. Der Master beginnt mit der Datenübertragung, indem er auch SCL auf LOW zieht. In dieser Zeit, in der SCL LOW ist, legt der Master das erste zu übertragende Bit auf die Datenleitung. Noch ist dieses Bit ungültig! Der Master erzeugt nun den ersten Taktimpuls durch Hochziehen der SCL-Leitung auf HIGH, das gleichzeitig auf SDA liegende Bit ist jetzt gültig.
4. Die Taktleitung SCL wird wieder auf LOW gezogen (Daten sind nun ungültig) und das 2. Bit wird vom Master auf die Datenleitung SDA gelegt.
5. Mit der zweiten HIGH-Periode von SDA (Daten gültig) erfolgt die Übertragung des 2. Bits auf der Datenleitung nach dem gleichen Prinzip.
6. Dieser taktgesteuerte Vorgang wiederholt sich, bis das 8. Bit (also insgesamt ein Byte) abgeschickt ist.
7. Der Master zieht die Datenleitung nun wieder auf HIGH und signalisiert mit der HIGH-Periode des 9. Taktes dem Empfänger, dass er für eine Bestätigung des übertragenen Bytes bereit ist.

Das Empfängergerät hat bereits nach Empfang des 8. Bits reagiert und die Leitung SDA auf LOW gezogen. Auf diese Weise überschreibt es durch die Wired-AND-Verknüpfung den vom Master erzeugten HIGH-Pegel.

Dieses Bit, das nun auf der Datenleitung SDA liegt, wird als *Acknowledge*-Bit des Empfängergerätes bezeichnet. Ein solches Bestätigungssignal muss während der gesamten HIGH-Periode des vom Master erzeugten 9. Taktes auf der Datenleitung liegen, sonst wird es vom Master nicht erkannt.

3.5.4 Das Acknowledge-Bit (ACK)

Der Taktimpuls für eine Bestätigung wird immer vom Master-Gerät erzeugt. Der sendende Baustein setzt nach der Übertragung eines Bytes den Pegel der Datenleitung auf HIGH, damit er nun vom Empfänger das Acknowledge-Bit in Form eines LOW-Pegels auf derselben Leitung lesen kann. Das ACK-Bit muss zeitlich so erfolgen, dass der LOW-Pegel auf der Datenleitung während des Taktimpulses für die Bestätigung (in obiger Erläuterung wäre dies der 9. Takt) stabil ist. Ansonsten wird der Master das ACK unter Umständen nicht als solches erkennen.
Hardwaremäßig geschieht dies automatisch. Bausteine, die das Busprotokoll hardwaremäßig nicht integriert haben, sondern per Software nachvollziehen müssen, sind vom Entwickler so zu konfigurieren, dass die Zeitbedingungen eingehalten werden.
Sollte der empfangende Slave die Datenübertragung nicht bestätigen können – etwa, weil er im Augenblick andere Funktionen auszuführen hat – darf der Slave SDA während des Bestätigungstaktes des Masters nicht auf LOW ziehen. Er erzeugt also kein Acknowledge-Bit. Der Master nimmt das zur Kenntnis und generiert dann eine Stopp-Bedingung, um die Datenübertragung abzubrechen.
Gleiches gilt für einen Slave, der das erste gesendete Byte (seine Adressierung) zwar schon bestätigt hat, die nachfolgend gesendeten Bytes allerdings nicht empfangen oder verarbeiten kann. In diesem Fall wird der Slave die Datenleitung SDA während des Bestätigungstaktes ebenfalls im HIGH-Zustand belassen.

Bild [10] zeigt noch einmal die Bestätigung nach Senden des 8. Bits vom Master:

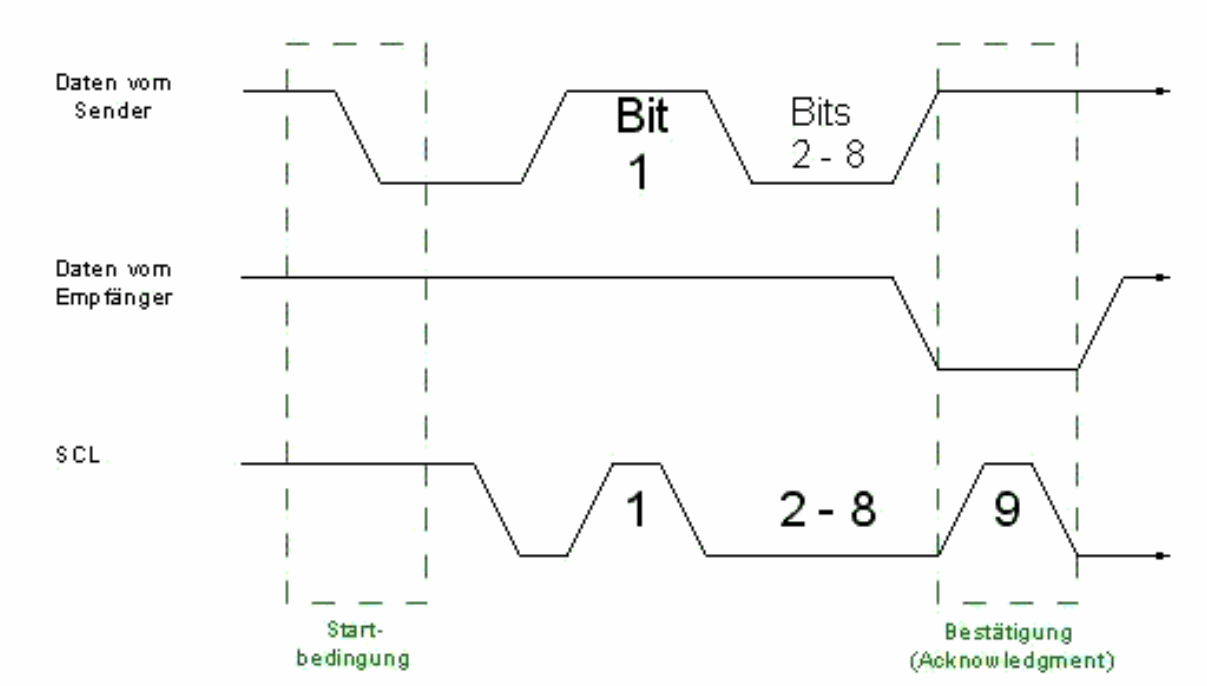

Bild [10]: Bestätigung bei der Datenübertragung
Quelle: Roboternetz.de

Aber auch ein Master kann in die Rolle des Empfängers treten und muss dann ein Acknowledge-Bit zum Slave senden. Dies geschieht zum Beispiel auf den Empfang eines Bytes von einem Slave. Der Master muss dies dem Slave-Gerät natürlich ebenfalls bestätigen wie zuvor der Slave dem Master bekannt geben musste, dass er adressiert – man nennt es auch selektiert – wurde.
Man beachte: Ein ACK eines empfangenen Bytes von einem Slave ist *immer* notwendig, *außer* beim letzten empfangenen Byte.

Neben dem Acknowledge-Bit gibt es auch noch einen Zustand, den man nicht ganz als Bedingung bezeichnen kann. Es handelt sich wirklich eher um einen Zustand im Datenfluss zwischen Master und Slave. Es ist der so genannte NACK (not-acknowledge).
Auch wenn es für not acknowledge – also nicht bestätigt – steht, könnte man es als eine Bestätigung vom Master an den Slave ansehen, um das Ende einer Übertragung anzuzeigen. NACK wird vom Master nach dem letzten Byte eines Lesezugriffs erzeugt.
Not acknowledge sollte *keinesfalls* mit **no** acknowledge verwechselt werden!

Wenn der Slave, nach Übertragung des 8. Bits vom Master zum Slave, die Datenleitung SDA nicht in den Zustand LOW zieht, kommt es zu einer **No** ACK-Bedingung. Das bedeutet, dass entweder:

- der Slave nicht erreichbar/nicht da ist (im Falle einer Adresse)
- der Slave einen Impuls verpasst hat und aus der Synchronisation mit der Taktleitung SCL des Masters kam.
- der Bus „steckengeblieben“ ist (eine der Leitungen könnte permanent LOW gehalten sein)

In jedem Fall sollte der Master dann den Versuch abbrechen, eine Stopp-Bedingung auf den Bus zu legen.
Ein Test für einen „steckengebliebenen“ Bus kann im Zyklus der Stopp-Bedingung ausgeführt werden.

Der Unterschied zwischen beiden „Bedingungen“ ergibt sich aus der Situation, in der sie auftreten können:

Bedingung	**kann nur auftreten …**
Not Acknowledge (NACK)	nachdem ein Master ein Byte **von** einem Slave **gelesen** hat
no Acknowledge	nachdem ein Master ein Byte **auf** einen Slave **geschrieben** hat

3.6 Adressierung

Das erste Byte, das nach der Startbedingung vom Master gesendet wird, stellt die Adresse des Slaves dar, mit dem der Master kommunizieren möchte. Die ersten 7 Bit zählen hierbei zur eigentlichen Selektierungs-Adresse und das 8. Bit legt die Lese- bzw. Schreibrichtung fest, d.h. ob der Master Daten in das Slave-Gerät schreiben oder daraus lesen möchte. Dieses 8. Bit wird auch mit R/W-Bit abgekürzt.
Nach diesem Konzept ist es beim I²C-Bus möglich, einen 7-Bit-Adressraum zu nutzen, was bedeutet, dass 112 Knoten auf dem Bus erlaubt sind. Es stimmt,

eigentlich sind 128 Adressen möglich, doch 16 von ihnen wurden für besondere und/oder zukünftige Zwecke reserviert und dürfen sollten daher nicht genutzt werden.
Inzwischen hat es bei I²C eine Adresserweiterung gegeben. Das System erlaubt auch die Verwendung von 10 Bit, indem es 4 der 16 reservierten Adressen verwendet. Beide Adressierungsarten können in einem System problemlos kombiniert werden.
Nun möchte ich noch einmal auf beide Adressierungsarten sowie die reservierten Adressen detaillierter eingehen.

3.6.1 Die 7-Bit-Adressierung

Die 7-Bit-Adressierung ist die erste Adressierungsform des I²C-Bussystems und ermöglicht – wie bereits oben erwähnt – bis zu 128 (= 2^7) Geräte am Bus anzuschließen, so dass jedes Gerät eine einmalige Adresse hat.
Das erste Byte einer Datenübertragung beinhaltet also die Adresse des Slaves, die 7 Bit lang ist. Diese 7 Bit werden noch einmal aufgespaltet in einen Teil, der in jedem I²C-Baustein fest verdrahtet ist einen restlichen Teil, der vom Nutzer des Systems noch verändert werden kann. Diese Aufsplittung ermöglicht es, mehrere Bausteine desselben Typs am I²C-Bus zu betreiben. Man spricht von einem festen und einem variablen Teil der Slave-Adresse, wobei man letzteren Teil oftmals auch als Subadresse bezeichnet. Üblicherweise beträgt ihre Länge 3 Bit (Bit 0-3). Mit ihnen lassen sich 8 (= 2^3) Adresskombinationen bilden, folglich können also maximal 8 Geräte dieses Typs gleichzeitig am Bus betrieben werden.

Zur Veranschaulichung sei das Prinzip der Subadresse noch einmal am 8-Bit-Ein-/Ausgabe-Baustein PCF 8574 vorgestellt:

	feste Adresse				**Subadresse**		
Adresse des PCF8574:	0	1	0	0	x	x	x
Bitwertigkeit:	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Man erkennt nun, dass die ersten vier Bit der Adresse (Bit 6 bis Bit 3) den Teil der festen Adresse bilden und die übrigen drei mit „x“ markierten Bit die veränderbare Subadresse darstellen.
Was ist nun der Zweck einer solchen Subadressierung?
Bei komplexeren Bausteinen mit vielen Steuerfunktionen reicht es oftmals nicht aus, lediglich auf ein einziges 8-Bit-Register des Slaves zugreifen zu können. Es sollten auch mehrere Register erreichbar und somit adressierbar sein. Mittels Subadressen innerhalb des Slaves ist dies zu bewerkstelligen.
Der Master sendet zur Adressierung dieser zusätzlichen Register nach der Slave-Adresse als zweites Byte die gewünschte Subadresse und erhält somit anschließend direkten Zugriff auf den Inhalt dieses Registers.

3.6.2 Datenrichtungs-Bit

Bisher sind allerdings nur 7 Bit der Slave-Adresse erwähnt worden. Das 8. Bit, das LSB (least significant bit, niederwertigstes Bit) ist das so genannte und bereits kurz erwähnte R-/W-Bit (siehe **Bild [11]**, unten). Es gibt die Datenrichtung an; es bestimmt also ob der Master Daten empfangen möchte oder ob er dem Slave seine Daten schicken möchte, die es dem Slave mitteilt.

Da es die Datenrichtung angibt, bezeichnet man es seltener auch als Datenrichtungs-Bit und je nach Wertigkeit LOW (= 0) oder HIGH (= 1) gibt es an, ob gelesen oder geschrieben werden soll:

- R-/W-Bit = 0: Schreiben
- R-/W-Bit = 1: Lesen

Bild [11]: Slave-Adresse und R-/W-Bit
Quelle: Roboternetz.de

Und so sähe nun ein Lese-Zugriff auf den Baustein PCF 8574 aus:

	Adresse				**Sub-Adresse**			**R/W**		**Daten**								
START	0	1	0	0	x	x	x	1	ACK	x	x	x	x	x	x	x	x	STOPP

Man sieht hier nochmals nach der Start-Bedingung die ersten 4 Bit, die die feste Adresse des Bausteins bilden, gefolgt von der jeweils vom Baustein abhängenden Subadresse. Um diese übrigens zu ändern, bietet der Baustein speziell Pins an, die am Gerät herausgeführt sind und somit angesteuert werden können.
Als 8. Adress-Bit ist das R-/W-Bit zu erkennen, hier mit „1“ als Lesezugriff gesetzt. Danach muss das Acknowledge-Bit vom Slave erfolgen, ehe der Master die eigentlichen Daten auslesen kann. Sobald er keine weiteren Daten mehr benötigt, wird die Stopp-Bedingung gesendet.

3.6.3 Reservierte Adressen

Von den eigentlich 128 Adressen, die Geräten am I²C-Bus zur Verfügung stehen sollten, wurden 16 Adressen reserviert, um den Bus ausbaufähig zu halten oder um Missverständnissen vorzubeugen:

Adresse	**Beschreibung**
0000 0000	General Call, Allgemeine Ruf-Adresse
0000 0001	Start-Byte / Start-Adresse
0000 001x	CBUS-Adressen
0000 010x	für verschiedene Busformate reserviert
0000 011x	Für zukünftige Zwecke reserviert
0000 1xxx	High-Speed Master Code
1111 0xxx	10-Bit-Adressierung
1111 1xxx	Für zukünftige Zwecke reserviert

General Call: Allgemeine Ruf-Adresse
Dies ist die allgemeine Ruf-Adresse aller Geräte am I²C-Bus. Sie wird genutzt, um Zugriff auf alle Geräte am Bus zu erhalten – zumindest auf jene, die in der Lage sind, diesen Ruf zu handhaben und die diese Daten brauchen. Geräte, die mit der allgemeinen Ruf-Adresse zwar umgehen können, doch diese nicht brauchen, werden darauf auch nicht reagieren.

Start-Byte
Nicht jeder Mikrocontroller, der mit dem I²C-Bus verbunden ist, hat einen integrierten I²C-Controller. Solche Mikrocontroller müssen die Busleitungen permanent beobachten, um eine I²C-Übertragung entdecken zu können. Dieses ständige Beobachten bzw. diese permanente Abfrage, ob etwas passiert, nennt man Polling und sie kostet viel Rechenzeit. Um diese Verschwendung der CPU-Leistung zu reduzieren, überträgt der Master die Start-Bedingung, gefolgt vom Start-Byte, einem unechten ACK-Impuls und einer wiederholten Start-Bedingung. Der überwachende Mikrocontroller muss nur eine der 7 Nullen des Start-Bytes auf der Datenleitung SDA entdecken, um eine I²C-Übertragung zu erkennen. Dies kann er mit einer relativ langsamen Polling-Rate (eine niedrige Abtastfrequenz) tun. Sobald der Controller dann entdeckt, dass die Datenleitung (über 7 Takte hinweg) den Zustand LOW besitzt, kann er auf eine höhere Polling-Rate schalten, um die wiederholte Start-Bedingung und die darauf folgende Übertragung zu erwarten.
Ist die Übertragung anschließend wieder beendet, kann er auf eine für die CPU deutlich entlastende Polling-Rate zurückschalten, um die nächste anstehende Übertragung zu erkennen.

CBUS-Adressen
Der CBUS ist ein Dreidraht-Bus, der andere Übertragungsformate als I²C verwendet. Damit man auch CBUS-Empfänger mit einem I²C-Bus verbinden kann, sind diese speziellen Adressen reserviert worden. I²C-Geräte müssen und werden Nachrichten an diese Adresse ignorieren.

Unterschiedliche Busformate
Mittels dieser Adresse wird es ermöglicht, I²C-Geräte mit Geräten zu verbinden, die unterschiedliche Protokolle auf demselben Bus benutzen. Nur I²C-Geräte, die mit diesen Protokollen arbeiten können, dürfen auf derartige Nachrichten antworten.

High-Speed Master Code
Das Adress-Schema für Hochgeschwindigkeitsübertragungen unterscheidet sich von der normalen Adressierungsprozedur, doch hierauf komme ich im Abschnitt zum High-Speed-Modus noch zu sprechen.

3.6.4 Die 10-Bit-Adressierung

Um Adress-Überschneidungen zu verhindern aufgrund der begrenzten Auswahl an 7-Bit-Adressen und der zugleich steigenden Beliebtheit des I²C-Bussystems wurde ein neues Adressierungsschema eingeführt: die 10-Bit-Adressierung. Sie kann mit der 7-Bit-Adressierung kombiniert werden und erhöht den verfügbaren Adressraum um das Zehnfache.
Nach der Start-Bedingung eröffnet eine führende Bitfolge „1111 0“ das 10-Bit-Adressierungsschema. Die letzten 2 Adress-Bits des ersten Bytes verknüpft mit dem

zweiten Byte (also mit allen 8 Bits des zweiten Bytes) formen die gesamte 10-Bit-Adresse.
Geräte, die nur das 7-Bit-Adressierungsschema nutzen, ignorieren Kommunikationsnachrichten mit der Bitfolge „1111 0“ einfach.

In der folgenden Abbildung **Bild [12]** werden die ersten 2 Byte einer Übertragung mit einer 10-Bit-Adresse gezeigt.
Wie man sehen kann, sind die ersten 5 Bit die reservierte Adresse „1111 0“, die die 10-Bit-Adressierung einleitet. Danach folgen die letzten 2 Bit der 10-Bit-Adresse: das MSB (most significant bit, höherwertigstes Bit) und das R-/W-Bit.
Nun senden mehrere Slaves ihr ACK und der Master überträgt die letzten 8 Bit der Adresse. Jetzt wird sich nur noch ein Slave mit einem ACK melden und die restliche Übertragung kann wie bei der 7-Bit-Adressierung vorgenommen werden.

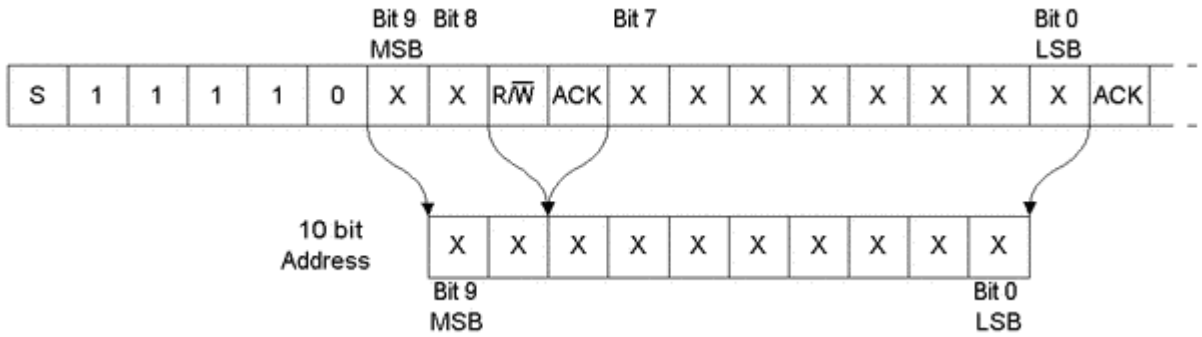

Bild [12]: Übertragung einer 10-Bit-Adresse
Quelle: I²C-Bus.org

3.7 Busgeschwindigkeiten

3.7.1 Standard-Modus (Standard Mode)

Wie schon einmal erwähnt, wurde die Busgeschwindigkeit für I²C in den ersten veröffentlichten Spezifikationen auf 100 kHz, also 100 kbit/s, festgelegt. Über die Jahre hinweg wurde diese Geschwindigkeit und Taktrate erhöht, so dass heute weitere Modi verfügbar sind.

3.7.2 Erweiterter I²C, Schneller Modus (Fast Mode)

Ursprünglich war der I²C-Bus eigentlich nur zum Austausch von Statusinformationen gedacht, heutzutage werden allerdings auch Text und andere Daten übertragen. Dazu reichen die 100 kbit/s nicht mehr aus. Daher erweiterte Philips 1992 seine Spezifikationen und es konnten nun 400 kbit/s übertragen werden. Inzwischen dürften alle modernen Bausteine den sogenannten Fast-Mode unterstützen.
Zusammen mit diesem neuen Geschwindigkeitsmodus kamen übrigens auch die Spezifikationen für das 10-Bit-Adressierungsschema.
Im Fast-Mode sind die physikalischen Kenngrößen des Busses nicht verändert worden. Ebenso wurden Protokoll, Buspegel, Ladekapazitäten und so weiter nicht verändert. Lediglich die Datenfrequenz ist gestiegen und es wurden Randbedingungen bezüglich Störgeräuschen wie Rauschen gesetzt.
Aufgrund der Neuerungen auch im Bereich der Zeitvorgaben (Berechnung und Steuerung) unterstützten alte Geräte des Dreidrahtbussystems CBUS den Fast-

Mode nicht mehr. Die Entwicklung von ICs mit CBUS-Schnittstelle wurden eingestellt, ebenso wie deren Produktion.
Die Eingänge der von da an entwickelten Fast-Mode-Geräte beinhalten alle Schmitt-Triggers, um den Störfaktor des Rauschens zu unterdrücken. In den Ausgangspuffern befinden sich nun Abfall-Controller. Sie haben nichts mit Müll zu tun, sondern sind für die fallenden Flanken von Signalen der Daten- und Taktleitung zuständig.
Zudem ist dafür gesorgt, dass die Bus-Pins potenzialfrei sind und den Bus nicht blockieren, wenn die Stromversorgung eines Fast-Mode-Geräts ausgeschaltet ist. Auch die Endwiderstände müssen für die neue Busgeschwindigkeit angepasst sein. Hierbei wurde spezifiziert, dass ein Widerstand für Ladungen bis zu 200 pF ausreichend ist.

3.7.3 Fast Mode Plus – macht den Fast-Mode schneller

Der I²C Fast-Mode wurde ursprünglich bestimmt, mit 400 kHz zu laufen. Aber was hindert den Menschen daran, diese Grenze einfach zu ignorieren und den Bus stattdessen mit noch höherer Geschwindigkeit laufen zu lassen?
Es gibt eine Anzahl von Faktoren, die die maximale Geschwindigkeit auf dem Bus begrenzen. Zum Beispiel wäre da die Kapazität, die eine Steigzeit für Signale einleitet. Als „Gegenmaßnahme“ kann hier der Strom auf dem Bus erhöht werden, indem man die Endwiderstände verkleinert.
Die Fast-Mode Plus Spezifikationen FM+, im April 2006 von Philips Semiconductors vorgestellt, definiert einen solchen Bus mit maximaler Geschwindigket von 1 MHz. Anders als unter dem nächsten Punkt vorgestellten High-Speed Mode gibt es keine zusätzliche Logik zu implementieren.
Fast-Mode Plus Geräte sind zudem noch abwärtskompatibel mit Standard- und Fast-Mode Geräten.

3.7.4 Hochgeschwindigkeits-Modus, High-Speed Mode, Hs-Mode

Es existieren Anwendungen (z.B. RAM-Bausteine), bei denen die vorherig erwähnten I²C-Übertragungsgeschwindigkeiten einen Begrenzungsfaktor darstellen. Um höhere Übertragungsraten zu erlauben und gleichzeitig eine gewisse Kompatibilität beizubehalten, hat Philips den HS I²C Standard eingeführt.
Geräte des Standard- oder Fast-Modes kann man **nicht** an dieselben Leitungen anschließen, wenn man den High-Speed- bzw. den Hs-Mode verwenden will. Für diesen Fall sind an einem Master mit Hs-Mode-Unterstützung zwei zusätzliche Pins für die normalen Geschwindigkeiten herausgeführt. Der Master übernimmt dann die Konvertierung.

Hier sind einige Details und Besonderheiten des High-Speed Modes:

Elektrische Charakteristika:
- Die High-Speed-Variante des I²C-Busses erlaubt eine Bitübertragung von bis zu 3,4 Mbit/s
- Beide Geräte, Master und Slave, müssen high-speed-aktiviert sein, um diese Erweiterung nutzen zu können.
- Hs-ICs sind abwärtskompatibel, das heißt, sie erlauben gemischte Bussysteme
- Hs-Mode Master-Geräte haben haben einen open-drain-Ausgangspuffer für das SDAH-Signal (SDA-Signal für den Hs-Mode) und eine Kombination eines open-

drain pull-down- und Stromquellen-pull-up-Schaltkreis auf dem SCLH Ausgang. Dieser Stromquellen-Schaltkreis verkürzt die Signal-Steigzeit des SCLH-Signals (SCL-Signal für den Hs-Mode).
- Hs-IC-Master können sogar den Strom auf dem Bus verstärken und somit als „verstärkte Stromquelle“ dienen. Diese funktioniert aber nur während der High-Speed-Verarbeitung und nur für einen Master!
- Hs-Mode-Mastergeräte können eine eingebaute Bridge (Brücke) besitzen, um Geräte mit langsamer Geschwindigkeit während einer Hochgeschwindigkeitsübertragung vom Bus zu trennen. Der Hauptzweck einer solchen Bridge ist es, die Ladekapazität auf dem Bus zu reduzieren und Konflikte zu verhindern, die durch Geräte mit langsamerer Geschwindigkeit begründet sind.
- Der einzige Unterschied zwischen Hs-Mode Slaves und Standard oder Fast-Mode Slaves ist die Geschwindigkeit, mit welcher sie in Betrieb sind. Hs-Mode Slaves haben open-drain-Ausgangspuffer an den SCLH und SDAH Ausgängen.

Eine High-Speed-Übertragung fährt im Fast-Mode an, z. B. mit maximal 400 kBit. Die Abfolge einer solchen Übertragung lautet wie folgt:

- Start-Bedingung wurde gesendet
- Für den Hs-Mode wird zuerst im Standard- oder Fast-Mode ein *Master Code* geschickt bevor auf die erhöhte Frequenz umgeschaltet wird. Wenn dieser nicht bestätigt wird (not acknowledge), schaltet der aktive Master in den Hs-Mode.
- Der Stromquellenschaltkreis ist nach Übertragung des Master Codes aktiviert
- Der aktive Master sendet eine wiederholte Start-Bedingung, gefolgt von der Adresse des Slaves, mit dem er kommunizieren möchte
- Diese Adresse wird bestätigt oder nicht (ACK oder NACK)
- Die Übertragung wird auf inzwischen bekannte Art und Weise fortgesetzt
- Der Stromquellenschaltkreis wird nach jeder wiederholten Start-Bedingung und jedem ACK oder NACK deaktiviert, um dem Slave eine Möglichkeit zu geben, den Takt zu strecken (Clock Stretching). Er wird reaktiviert, sobald die Taktleitung SCL von allen Geräten freigegeben wurde
- Alle Geräte kehren in den Fast-Mode zurück, nachdem eine Stopp-Bedingung gesendet wurde.

Bild [13] zeigt den Beginn einer High-Speed-Übertragng:

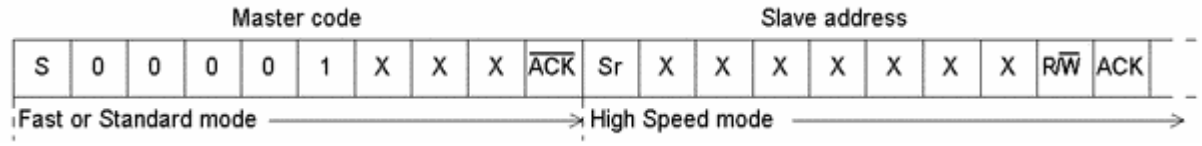

Bild [13]: Hs-Mode-Übertragung
Quelle: I²C-Bus.org

3.8 Verwendung von I²C

Der I²C-Bus besitzt ein weiträumiges Anwendungsgebiet und ist in vielen Komponenten enthalten, wie beispielsweise in Speichermodulen wie dem EEPROM, in AD-Wandlern, speziellen Schnittstellen-Modulen wie LCD, Temperatur- und Drucksensoren, Zeitgebern und so fort.

Der „Abkömmling“ des I²C-Busses ist der System-Management-Bus (SM-Bus). Auch er ist ein Zweidrahtbus, der mit Daten- und Taktleitung auskommt und wurde vor allem für Halbleiter-ICs entwickelt.
Man kann ihn aber nicht nur deshalb als Abkömmling bezeichnen, sondern in erster Linie, weil er auf dem I²C-Busprotokoll basiert. Er hilft dabei, den Zustand von Komponenten zu erkennen und Hardwareeinstellungen vorzunehmen. Zudem wird er auch in jedem handelsüblichen PC eingesetzt, um Daten wie Lüfterdrehzahlen, Spannungs- und Temperaturwerte zu sammeln.
Busse wie SMP und I²C wurden entwickelt, als man merkte, dass ein Großteil der Kosten eines ICs und der verwendeten Leiterplatte von der Größe des Gehäuses und der Anzahl der Pins abhingen.
Ihre besonderen Vorteile sind die niedrigen Kosten und ihre Einfachheit. Da I²C allerdings recht simpel ist, besitzt es eine sehr hohe Störanfälligkeit. Dieses Tatsache schränkt seine Anwendung auf störungsarme Bereich ein.
Er wird häufig für Lautstärkeregler, Analog-Digital- oder Digital-Analog-Wandler, Echtzeituhren, kleine nichtflüchtige Speicher oder bidirektionale Schalter und Multiplexer eingesetzt.

Große Bedeutung hat das I²C-Protokoll aber in der Vergangenheit im Chipkartenbereich. Beispielsweise ist die in Deutschland verwendete Krankenverischertenkarte eine I²C-Karte. Unter den goldenen Kontaktflächen der Chipkarte verbirgt sich ein simpler I²C-EEPROM, der vom Kartenleser über das I²CProtokoll ausgelesen und beschrieben werden kann.

4. Zusammenfassung

Zum Ende dieser Ausarbeitung möchte ich noch einmal eine kurze Zusammenfassung beider Busse aufzeigen, um Vor- und Nachteile von SPI und I²C, sowie deren Unterschiede und Gemeinsamkeiten zu verdeutlichen.

Vergleichskriterium	**SPI**	**I²C**
Anwendungen des Busses	beide Bussysteme finden gleichermaßen Anwendung in der Steuerung von Speichermodulen (z.B. EEPROM), bei Druck- und Temperatursensoren, Wandlern (A/D, D/A), Echtzeituhren …	
Ist der Bus ein offizieller Standard?	nein	ja, besitzt Philips I²C-Bus Spezifikationen
Entwickler	Motorola	Philips Seminconductors
Art?	„unechtes“ Zweidrahtbussystem (besitzt im Grunde 4 Leitungen)	Zweidrahtbussystem
Synchron oder Asynchron?	Synchron	Synchron
Seriell oder Parallel?	Seriell	Seriell
Arbitration, Multi-Master-Fähigkeit?	nein	ja
Zeit für Datentransfer?	1 SPI-Takt für jedes zu übertragende Bit + Zeit für Start- und Stopp-Bedingung	1 I²C-Takt für jedes zu übertragende Bit + 1 ACK-Bit für jedes Byte + Zeit für Start- und Stopp-Bedingung (Singlemaster). Im Multimasterbetrieb ist die Arbitrierung ausschlaggebend

Vergleichskriterium	SPI	I²C
Datenblocklängen	beliebig	beliebig
Bitrate (von … bis)	1 bit/s .. ca. 5 Mbit/s	1 bit/s ... 3,4 Mbit/s
Lizenzgebühr?	nein	Ja, für den Chiphersteller
Kosten für Master/Slave?	3 I/O-Pins des Mikrocontrollers für den Bus und für jeden Slave eine CS-Leitung (oder eine gemeinsame)	2 I/O-Pins des Mikrocontrollers
Zukunftsaussichten?	So einfach, das ein Ersatz nicht zu sehen ist	Massiver Einsatz in der Unterhaltungselektronik

Sonstige Vorteile des SPI-Busses:
- Einfach per Software im Master zu implementieren
- gut geeignet für Kommunikation in kleinen Systemen

Sonstige Nachteile des SPI-Busses:
- Keine Kontrolle der Flankensteilheit der Signale
- Keine Fehlerabsicherung bei der Übertragung

Sonstige Vorteile des I²C-Busses:
- der Master generiert den Bustakt
- Jedes Gerät am Bus hat seine einzigartige Adresse

Sonstige Nachteile des I²C-Busses:
- Sehr anfällig für Störgeräusche

5. Resumée

Zum Abschluss und nachdem ich mich lange und intensiv mit beiden Zweidrahtbussystemen beschäftigt habe, kann ich sagen, dass sowohl SPI als auch I²C ihre Vor- und Nachteile haben.
Je nach Anwendungsgebiet sollte man den einen Bus dem anderen vorziehen.
Bei beiden Bus-Systemen gefiel mir vor allem die Einfachheit, die dahinter steckt. Nach kurzer Einarbeitung ist das Konzept beider Busse sehr gut zu verstehen und zeigt, dass sich auch schon weiter zurückliegende Ideen und Entwicklungen immer noch auf unvorhersagbare Dauer in Gegenwart und Zukunft problemlos nutzen lassen.
Besonders interessant wäre der Vorbereitungen für diese Seminararbeit waren die vielen kleinen Nebeninformationen zu anderen Themen wie beispielsweise Schmitt-Trigger. Zudem hat es mir meiner Meinung nach geholfen, mich nicht nur mehr in die Informatik zu begeben, sondern mich auch auf physikalisch und elektrotechnisch nahes Gebiet zu wagen – und das Dank der Einfachheit beider Busse, ohne mich dabei zu sehr in der Digitaltechnik oder physikalischen Formeln und Größen zu verlieren.
Besonders erfreut hatte mich, nachdem ich mich tiefer in das Thema eingearbeitet hatte, dass ich mein kürzlich erworbenes Wissen im Bereich der Mikroprozessortechnik hier wiederfinden konnte. Daisy-Chaining bei SPI oder der Einsatz beider Bussysteme bei der A/D- und D/A-Wandlung erleichterten mir nicht nur das Einfinden in die Arbeitsweisen, sondern vermittelten mir auch einen praxisnahen Nutzen der Bussysteme, der deutlich weniger abstrakt ist als einfache Kennlinien.
Abschließend kann ich sagen, dass mich diese Ausarbeitung viel Arbeit und Zeit gekostet hat – vor allem, was die Recherche betraf –, ich aber der Meinung bin, dass allein das neu erworbene Wissen die Mühe wert war.

Literaturverzeichnis:

Hinweis: Für diese Ausarbeitung wurden ausschließlich Informationen aus dem Internet verwendet.

Online-Literatur:

Wikipedia:
http://de.wikipedia.org

Grundlagenartikel von Martin Schwerdtfeger:
http://www.mct.de/faq/spi.html

Freescale SPI-Handbuch:
http://www.freescale.com/files/microcontrollers/doc/ref_manual/S12SPIV3.pdf

Roboternetz:
http://www.roboternetz.de/wissen/index.php/SPI

Offizielle I²C-Bus-Webseite:
http://www.i2c-bus.org/

Elektronik-Magazin:
http://www.elektronik-magazin.de/page/der-i2c-bus-was-ist-das-21

Mikrocontroller.net:
http://www.mikrocontroller.net/articles/I2C

Robot-Electronics:
http://www.robot-electronics.co.uk/htm/using_the_i2c_bus.htm

Esacademy.com:
http://www.esacademy.com/faq/i2c/

I²C-Busspezifikationen von NXP:
www.nxp.com/acrobat_download/usermanuals/UM10204_3.pdf

Dr. Königs Märklin-Digital-Page:
http://home.arcor.de/dr.koenig/digital/i2c.htm#i2c

Erkenntishorizont.de:
http://www.erkenntnishorizont.de/robotik/bussys/i2c.c.php?screen=800

TransistorNET.de
http://www.transistornet.de/viewtopic.php?f=23&t=2705