Inter-Integrated Circuit (I2C)

Proseminar Microcontroller und eingebettete Systeme WS2013/2014

Clemens Jonischkeit
Lehrstuhl für Echtzeitsysteme und Robotik
Fakultät für Informatik
Technische Universität München
Email: clemens.jonischkeit@in.tum.de

Kurzfassung

Der von Philips enwickelte I2C Bus ist ein Bidirektionaler, Serieller Bus zur effizienten inter-IC kontrolle. Der Bus benötigt nur 2 Leitungen, eine Serielle Takt leitung (SCL) und eine Serielle Daten Leitung (SDA). Er arbeitet Seriell, ist byte orientiert und überträgt mit geschwindigkeiten von bis zu 100 kbit/s im Standard Mode und 3.4 Mbit/s im High-speed Mode. Ausserdem gibt es noch den unidirektionalen Ultra Fast-Mode, bei dem ein Übertragunsgeschwindigkeit von bis zu 5 Mbit/s erzielt werden kann.

Schlüsselworte

I2C, Bus, Standart-mode, Fast-mode, Fast-mode Plus, SDA, SCL

I. Einführung und Motivation

Bus Systeme sind aus Computersystemen heutzutage nicht mehr Wegzudenken, sei es USB im Computer oder die Verschiedenen Busse in der Steuerung von Autos. Sie vereinfachen die Kommunikation mit verschiedenen Komponenten maßgeblich und gehen dabei je nach Bus sparsam mit der Hardware um. Ein anderer prominenter Bus ist beispielsweise PCI. Mit dessen Hilfe können Grafikkarten im Computer angeschlossen werden. Weiterhin ist der Inter-Integrated Circuit (I2C) Bus durch seine universelle Einsetzbarkeit ein sehr vielseitiger Bus. Er kann zum Beispiel auf Grund seiner besonders niedrigen Hardwareanforderungen dazu verwendet werden kleine Mikroprozessoren mit Peripherie zu verbinden. Dieser Artikel führt in die Grundlagen des I2C Bussystems ein und hilft beim Verständnis wie dieser funktioniert. Er gibt einen Überblick über Verbindungsaufbau, Datenübertragung und Kollisionserkennung in den Modi Standard Mode (SM), Fast Mode (FM) und Fast Mode+ (FM+).

II. Allgemeinens

A. Bus Systeme

Ein Bussystem ist ein Netzwerk zur Datenübertragung, bei dem alle Teilnehmer miteinander verbunden sind. Somit kommt die Nachricht eines Teilnehmers bei jedem andern an. Diese Busse können noch einmal in Parallel und Seriell unterschieden werden. Ein Paralleler Bus überträgt pro Takt ein ganzes Maschinenwort, wohingegen ein serieller Bus das Wort stück für stück überträgt. Beide Arten haben Vor- und Nachteile. Ein Paralleler Bus kann mehr Daten pro Takt übertragen, benötigt dafür aber auch eine größere Menge an Verbindungen als ein serieller Bus. In Computersystemen bekannte Busse sind Parallel ATA, was zum Anschließen von Festplatten verwendet wurde. Das Nachfolgesystem ist Seriell ATA (SATA). Bekannt ist auch der PCI Bus, der verwendet werden kann um Grafikkarten und andere Erweiterungen anzuschließen

B. Der I2C Bus

Der I2C Bus wird auch oft mit I^2C oder IIC bezeichnet. Es handelt sich dabei um einen seriellen Bus der seine Daten Bit für Bit überträgt. Durch seine Kollisionserkennung ist er ein echter Multimaster Bus der auch mehrere Slaves unterstützt. Er ist in vielen Mikrokontrollern, wie der DS1337 I2C Serial Clock, direkt integriert. Durch seine weite Verbreitung und seine Geringe Hardwareanforderung lässt er sich somit sehr leicht in eigene Projekte Integrieren. Allgemein ist er dort besonders gut, wo man eine intelligente Kontrolle, also einen Mikrokontroller mit allgemeinen ICs wie LCD Treibern, Speicher und Echtzeituhren mit Anwendungsspezifischen ICs für beispielsweise Sensoren oder Signalverarbeitung verbinden will. Eine Wetterstation ist hierfür ein geeignetes Beispiel. Man Benötigt Sensoren zur Messung von Niederschlag, Temperatur, Windstärke und Windrichtung. Außerdem sollen die gesammelten Daten zur späteren Auswertung abgespeichert werden und eventuell zeigen wir sie auch über ein kleines LCD Display an. Als letztes benötigen wir nur noch einen Mikrokontroller, der einfach nur die Daten ausliest und zu bestimmten Zeiten abspeichert.

III. Hardware

Der I2C Bus benutzt für Kommunikation 2 Leitungen, die Serial Clock Line (SCL) und die Serial Data Line (SDA). Diese werden jeweils über einen pullup Wiederstand mit der Betriebsspannung V_{CC} verbunden. Dadurch liegt auf beiden Leitungen Standardmäßig ein ‘HIGH‘ an. Um nun ein ‘LOW‘ auf eine Leitung zu schreiben muss diese mit Ground verbunden werden. Daraus folgt eine wichtige Eigenschaft der Leitungen. Sie führen jeweils ein Logisches und durch, denn ein Busteilnehmer der ein ‘HIGH‘ lässt die Leitung in Ruhe, somit reicht ein Teilnehmer der ein ‘LOW‘ schreibt um die ganze Leitung auf ‘LOW‘ zu ziehen. Wobei ‘LOW‘ 30% und ‘HIGH‘ 70% der Referenzspannung betragen. Während der Bus frei ist haben beide Leitungen Konstant den Wert ‘HIGH‘ und während einer Übertragung muss die Datenleitung konstant sein während SCL ‘HIGH‘ ist. Bus Teilnehmer können allgemein in zwei Kategorien unterschieden werden, nämlich den Master und den Slave. Während ein Master jede Übertragung beginnt warten die Slaves darauf angesprochen zu werden. Die ganze Kontrolle über den Bus gehört dem Master, er kontrolliert die Übertragung und erzeugt das Takt Signal unabhängig davon ob er Sender oder Empfänger ist. Slaves können sich nicht über den Bus bemerkbar machen, das heißt wenn man eine Lichtschranke in einem Echtzeitsystem hat gibt es zwei Möglichkeiten einem Mikrokontroller Bescheid zu geben. Der Kontroller der Lichtschranke kann selbst ein Master sein und den Mikrokontroller wie einen Slave ansprechen oder er benötigt eine direkte Verbindung abgesehen vom Bus um eine Unterbrechung der Lichtschranke anzuzeigen. Ansonsten ist der Mikrokontroller darauf angewiesen den Mikrokontroller oft genug zu fragen, ob die Lichtschranke unterbrochen ist.

IV. Datenübertragung

Um eine Datenübertragung starten zu können muss sich der Bus im Zustand ‘Frei‘ befinden. Dann kann, durch senden von START, der Bus in den Zustand ‘Beschäftigt‘ überführen werden. START wird gesendet, indem während SCL den Wert ‘HIGH‘ hat, man SDA auf ‘LOW‘ zieht. Kurze Zeit nach diesem START beginnt dann der Master mit der Erzeugung des Taktes auf SCL. Nun beginnt die eigentliche Übertragung der Daten, die Byteweise erfolgt. Dabei wird das Höchstwertigste Bit immer zuerst übertragen und das Niederwertigste zuletzt. Das erste Byte einer Übertragung legt die Übertragungsrichtung fest und Adressiert den Slave. Nach der Übertragung eines vollen Bytes muss der Erhalt der Daten vom Empfänger bestätigt werden. Dazu lässt der Sender die Taktleitung für einen Takt in Ruhe und der Empfänger schreibt ein ‘LOW‘ um den korrekten Empfang zu bestätigen. Der Grund dafür, dass die Bestätigung aus einem ‘LOW‘ besteht, ist, ein Slave der nicht vorhanden oder Defekt ist, wird mit dem Bus nicht interagiert und der SDA hat weiterhin den Wert ‘HIGH‘. Abgesehen von defekten und nicht existierenden Teilnehmern gibt es weitere Gründe für das Ausbleiben der Bestätigung. Der Empfänger kann momentan nicht in der Lage sein, Daten zu empfangen. Dies kann passieren, wenn der Empfänger beispielsweise in einem

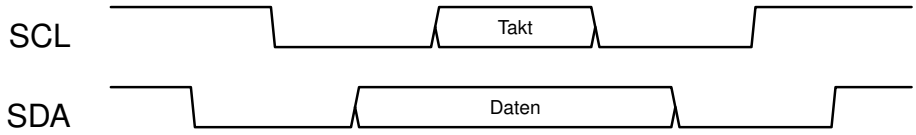

Abbildung 1. START und STOP auf SCL und SDA

Echtzeitsystem einen Zeitkritische Aufgabe, wie der Abarbeitung eines Interrupts, erfüllen muss. Ein voller Netzwerkpuffer kann hierfür auch der Grund sein. Grund für keine Bestätigung kann auch sein, dass Übertragene Daten nicht verstanden werden. Das kommt vor, wenn der Empfänger Daten erhält, die nicht seiner Spezifikation entsprechen. Ein Master, der Daten von einem Slave empfängt, kann durch nicht Bestätigung der Daten dem Slave anzeigen, dass er die Übertragung beenden möchte. Ist eine Übertragung Abgeschlossen muss der Bus wieder mithilfe eines STOPs in seinen Zustand ‘Frei‘ gebracht werden. Dieser STOP ähnelt dem START, so wird hier bei einem ‘HIGH‘ Signal auf der Takt Leitung ein Übergang von ‘LOW‘ nach ‘HIGH‘ auf der Datenleitung gesendet. Will der Master jedoch direkt im Anschluss an die erste Übertragung eine Zweite machen, so kann er auch wiederholt ein START senden. Dieses START beendet dann die letzte Übertragung und startet sofort eine neue, ohne den Bus dazwischen frei zu geben. START und STOP können nur von einem Master erzeugt werden, der Takt wird auch immer vom Master erzeugt, unabhängig von der Übertragungsrichtung und selbst während des Bestätigungsbits. STOP darf nie direkt auf START folgen. Das Bedeutet zwischen einem START und einem STOP muss immer mindestens ein Byte übertragen werden.

A. Adressierung

Damit ein Master gezielt Slaves ansprechen kann, wird Standardmäßig eine sieben Bit Adressierung verwendet, eine Adressierung mit zehn Bit ist jedoch auch möglich. Die Adresse jedes Slaves muss dabei eindeutig sein. Diese Adresse wird mit dem Ersten Byte jeder Übertragung gesendet und nimmt dabei die höherwertigen Bit ein. Das niederwertigste Bit entspricht der Übertragungsrichtung wobei 0 für Schreiben und 1 für Lesen steht. Der angesprochene Slave bestätigt die Adresse und die Übertragung beginnt entsprechend ihrer Richtung. Adressierung eines Slaves mit einer zehn Bit Adresse ist jedoch etwas komplizierter, da die zehn Bit nicht auf einmal übertragen werden können. Hier wird zuerst ‘1110XX0‘ gesendet. ‘XX‘ steht für die zwei Höchstwertigen Bit der zehn Bit Adresse. Anschließend Überträgt der Master mit dem zweiten Byte die Restlichen acht Bit der Adresse. Falls ein schreibender Zugriff erfolgen soll, so kann jetzt die Übertragung der eigentlichen Daten beginnen. Wenn man jedoch lesend zugreifen will, besteht das Problem, dass eine Änderung der Übertragungsrichtung an dieser Stelle nicht möglich ist. Dies geht nur nach dem ersten Byte. Deshalb wird nach den 2 Byte der Adresse die Verbindung mit einem *wiederholten START* neu gestartet. Das erste Byte ist jetzt jedoch ‘1110XX1‘, um den lesenden Zugriff anzuzeigen. Durch das vorherige Schreiben der gesamten Adresse und den Wiederholten Start, sprechen wir immer noch die vorher auf den Bus geschriebenen zehn Bit Adresse an. Slaves die den Startzustand auf dem Bus erkennen müssen ihre Bus Logik zurücksetzen, damit der Ordnungsgemäße Empfang der Adresse gewährleistet ist, selbst wenn sich der Bus davor in einem fehlerhaften Zustand befand. Abgesehen von dem Ersten Byte der 10 Bit Adressierung gibt es auch noch weitere Besondere Adressen. Es gibt ein ‘general call‘, dabei werden Daten an mehrere Slaves übertragen. Dies kann auch dazu benutzt werden um bei den Slaves ein Softwarereset zu erzeugen. Jedoch ist die Implementierung des Softwareresets nicht Pflicht bei I2C Bus Teilnehmern. Das START Byte (0x01) hilft Implementation die auf kontinuierlicher Abtastung der Leitungen basieren. Es ermöglicht dem Teilnehmer eine geringere Sampling Rate zu wählen und bei Erkennung des START Bytes in eine hohe zu schalten um auf das wiederholten START im Anschluss reagieren zu können. Das START Byte darf nicht bestätigt werden.

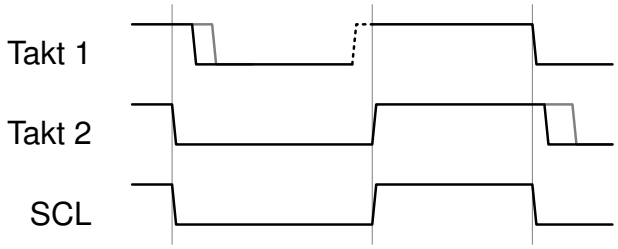

Abbildung 2. Synchronisation mehrerer Taktsignale

B. Sende und Empfangsmodus

Busteilnehmer kennen im Allgemeinen 2 Modi in denen sie operieren können. Den Sende und den Empfängermodus. Ein Master der einen Buszugriff durchführt beginnt immer im Sendemodus. Dadurch kann er die Adresse und die gewünschte Übertragungsrichtung auf den Bus legen. Möchte er jetzt Daten schreiben so ändert sich sein Modus nicht. Slaves hingen befinden sich zu Beginn jeder Übertragung im Empfängermodus. Sie lesen die Adresse vom Bus und bestätigen diese, wenn sie mit ihrer eigenen übereinstimmt. Will der Master einen lesenden Zugriff durchführen, wechselt er nach dem Senden des ersten Byte selbst in den Empfängermodus und der angesprochene Slave in den Sende Modus. Der Slave sendet so lange Daten, bis der Master ihm Anzeigt, dass er die Übertragung beenden möcht. Dies tut er indem er das letzte Empfangene Byte nicht Bestätigt und anschließend STOP oder ein wiederholtes START sendet. Diese Änderung der Übertragungsrichtung ist nur nach dem ersten Byte möglich. Soll die Richtung während der Übertragung geändert werden, muss die Übertragung beendet und eine neue gestartet werden.

V. Arbitrierung in Multimaster Systen

Da es sich beim I2C Bus um einen echten Multimaster Bus handelt kann es passieren, das mehrere Master zur selben Zeit einen Buszugriff durchführen wollen. Mit Hilfe der Bitweisen Arbitierung wird sichergestellt, dass mindestens ein Master seine Nachricht fehlerfrei übertragen kann. Die Lösung des Problems erfolgt in zwei Stufen, zuerst wird der Takt aller beteiligten Master synchronisiert. Anschließend erfolgt die bitweise Abitrierung, die von einem Master gewonnen wird und dieser kann seine Daten übertragen.

A. Synchronisation

Die Synchronisierung des Taktes ist notwendig, da die verschiedenen Master ihr START nicht gleichzeitig gesendet haben und auch der Takt leicht voneinander abweichen kann. Für die Synchronisation wird die logische Und Funktion der SCL Leitung ausgenutzt. Nach dem Schreiben des START Zustands und dem abwarten der Wartezeit, wird SCL von jedem Master auf ‘LOW‘ gesetzt und damit begonnen die ‘LOW‘ Phase abzuzählen. Sobald sie damit fertig sind lassen sie die Leitung aus und versuchen sie somit wieder auf ‘HIGH‘ zu bringen. Solange jedoch mindestens ein Master noch nicht mit dem Abzählen seiner ‘LOW‘ Periode fertig ist bleibt SCL auf ‘LOW‘. Erst wenn auch der letzte Master fertig ist, wechselt SCL auf ‘HIGH‘ und alle Master fangen an diese Periode runter zu Zählen. Somit wird die Länge der ‘LOW‘ Phase vom Langsamsten, die Länge der ‘HIGH‘ Phase vom schnellsten Master bestimmt und der Takt synchronisiert.

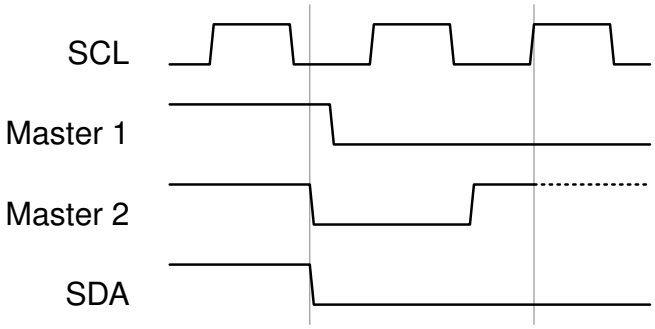

Abbildung 3. Bitweise Arbitrierung

B. Arbitrierung

Um zu entscheiden welche Nachricht letztendlich übertragen wird, nutzt man bitweise Arbitrierung. Dazu schreibt jeder Master die Daten auf den Bus, die er Senden möchte. Während SCL ‘HIGH‘ ist prüft dann jeder Master ob das Level von SDA zu den, von ihm gesendeten Daten, passen. Sollte er feststellen, dass sich SDA und seine Daten unterscheiden, bemerkt er, dass mindestens ein weiterer Master den Bus kontrolliert und gibt den Bus frei. Dies kann nur passieren, falls er eine 1 und ein anderer Master ein 0 sendet. Der andere Master, der die 0 sendet, bemerkt jedoch nicht, dass weiteren Master auf den Bus schreiben und fährt so mit seiner Übertragung fort. Wichtig für einen Master, der auch eine Slave Rolle einnimmt, ist, wenn er die Arbitrierung bereits bei der Adresse verliert in den Slave Modus zu schalten, da der andere Master eventuell auch ihn ansprechen will.

C. Clockstretching

Das Taktsignal beim I2C Bus wird immer vom Master generiert. Slaves haben jedoch eine Möglichkeit darauf Einfluss zu nehmen. Sie können die Taktrate verringern, indem sie, während SCL bereits ‘LOW‘ ist, selbst ‘LOW‘ auf die Taktleitung schreiben. Durch die logische Und-Funktion der Leitungen bleibt SCL auf ‘LOW‘ auch wenn der Master die Leitung auf ‘HIGH‘ setzt. Der Master erkennt dies und ist gezwungen zu Warten, bis der Slave die Taktleitung wieder frei gibt. Die meisten Slaves benötigen dieses Feature nicht, jedoch kann es dort Anwendung finden, wo Daten direkt verarbeitet werden. Clockstretching hat aber auch einen entscheidenden Nachteil. Für den I2C Bus ist kein Timeout definiert, sodass keine minimale Übertragungsrate gewährleistet wird.

VI. Conclusion

Der I2C Bus bietet, durch seine weite Verbreitung und direkte Integration in viele ICs sehr vielseitige Anwendungsmöglichkeiten. Er geht dabei sehr Sparsam mit der Hardware um und benötigt nur zwei Leitungen, Takt und Daten. Die Integration eines I2C Buscontrollers in den Mikrokontroller bringt für Entwickler den Vorteil, dass das Boarddesign stark vereinfacht wird. Es vereinfacht auch die Aktualisierung der Hardware. Um sein System zu verbessern Reicht es oft schon aus einen IC auszutauschen. Da der Buscontroller im IC integriert ist, muss die Ansteuerung des neuen IC’s kaum geändert werden. Der I2C Bus ist kompatibel zum CBUS, sodass es sogar möglich ist SCL und SDA gemeinsam mit dem CBUS zu verwenden was zu weiteren Einsparungen in der Hardware führt. Das I2C Protokoll wird auch weitgehen im System Management Bus verwendet (SMBus), der in vielen modernen PCs zu finden ist. Bei einigen Geräten, beispielsweise den Mikrokontrollern von Atmel, liest man auch von dem so genannten Two Wire Interface (TWI). TWI ist zu I2C weitgehend kompatibel, sodass die meisten I2C fähigen Slaves auch mit dem TWI eingesetzt werden können

ABBILDUNGSVERZEICHNIS

LITERATUR

[1] *AVR315: Using the TWI module as I2C master*, ATMEL, http://www.atmel.com/Images/doc2564.pdf, 2010.
[2] S. Hackländer, W. und Furchtbar, *Datenbus Praktikum*. Elektor-Verlag GMBH, 1995.
[3] *UM10204 I2C-bus specification and user manual*, 6th ed., NXP, http://www.nxp.com/documents/user_manual/UM10204.pdf, 2014.