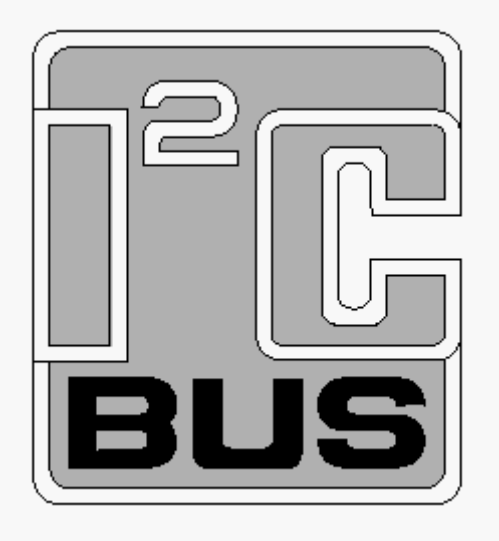
Der I²C-Bus



Bearbeitet von:

Thomas Finke, EL5
thomas.finke@gmx.de

Inhaltsverzeichnis

1. Entstehungsgeschichte des I²C-Bus

Der I²C-Bus wurde Anfang der 80er Jahre von Philips entwickelt. Das ursprüngliche System wurde zuerst als Batterie-Kontrollinterface verwendet, später aber zu einem einfachen Bussystem erweitert.
Im Vergleich zu den, normalerweise verwendeten, parallelen Bussystemen hat der I²C-Bus den Vorteil, dass er nur zwei Leitungen (sowie Ground) benötigt und dadurch die Bauteile weniger Anschlusspins benötigen, die Gehäuse dadurch kleiner werden und die Layouts einfacher werden. Was sich alles positiv in den Kosten niederschlägt.
I²C-Bus oder IIC-bus steht für Inter-IC-Bus. Also ein Bus, der dafür gedacht ist IC's untereinander zu verbinden. Dieser Bus wird heutzutage hauptsächlich in Geräten der Unterhaltungsindustrie, wie Fernseher, Stereoanlagen oder auch Computern eingesetzt, um die internen Komponenten miteinander zu verbinden. Ein Fernsehgerät besitzt z.B. einen Mikrocontroller mit I²C-Schnittstelle, ein I²C-EEPROM, einen Tuner-Baustein mit I²C-Interface und einen Videotextdecoder mit I²C-interface. In modernen PC's wird der I²C-Bus auch teilweise dazu verwendet um die Informationen über Lüfterdrehzahl oder Temperaturen zu erfassen.
Zu Beginn des Busses gab es nur Bausteine von Philips, mit der Zeit kamen aber immer mehr andere Chiphersteller, wie Infineon, Intel oder Texas Instruments hinzu und die Produktsparte der I²C-Bausteine umfasst mittlerweile Speicherbausteine, AD- und DA-Wandler, Tuner, Videotextdecoder, Echtzeituhren, Temperaturfühler und viele andere.

2. Technik der I²C-Bus

2.1 Hardware-Struktur

Der Bus besteht aus zwei Leitungen, SDA und SCL. Diese beiden Leitungen sind mit allen I²C-Geräten verbunden.
SCL (Serial Clock) ist die Taktleitung, welche zur Synchronisation des Datentransfers verwendet wird. SDA (Serial Data) ist die Datenleitung.
Als dritte Leitung wird jedoch noch die Masse bzw. die 0V als Bezugspotential benötigt.
Alle Geräte besitzen „Open Collector" oder „Open Drain" Ausgänge. Dies bedeutet, dass ein Gerät zwar eine Leitung nach „Low" ziehen kann, diese aber nicht auf „High" zwingen kann. Um High-Signale zu erreichen besitzen die SDA- und SCL-Leitung Pull-Up-Widerstände nach +5V oder +3,3V.
Im normalen Modus arbeitet der Bus mit 100 kbit/s, man kann diesen jedoch auch in einen langsamen Modus (10 kbit/s), den Fast-Mode (400 kbit/s) oder den Highspeed-Mode (3,4 Mbit/s) schalten.
Jeder Busteilnehmer besitzt eine eindeutige Adresse und kann als Sender oder Empfänger arbeiten. Zusätzlich zu dieser Sender/Empfänger-Struktur wird das Master/Slave-Konzept angewandt. Hierbei ist das Gerät welches den Datentransfer startet der Master und hat somit die Aufgabe das Taktsignal zu generieren, alle anderen Busteilnehmer sind demnach Slaves. Weiterhin handelt es sich bei dem I²C-Bus um einen echten Multi-Master-Bus. Das heißt, es gibt eine Kollisionserkennung und Arbitrierung um Datenverluste zu vermeiden, welche durch gleichzeitiges Senden von mehreren Mastern auftreten könnten.
Durch die oben genannten „Open Collector"- bzw. „Open Drain"-Ausgänge besitzt der Bus auch eine „wired-and"-Funktionalität. Dies bedeutet, dass wenn auch nur ein Ausgang LOW ist, die ganze Leitung auf LOW liegt. Auf den Vorteil dieser „wired-and"-Funktionalität gehe ich später noch genauer ein.

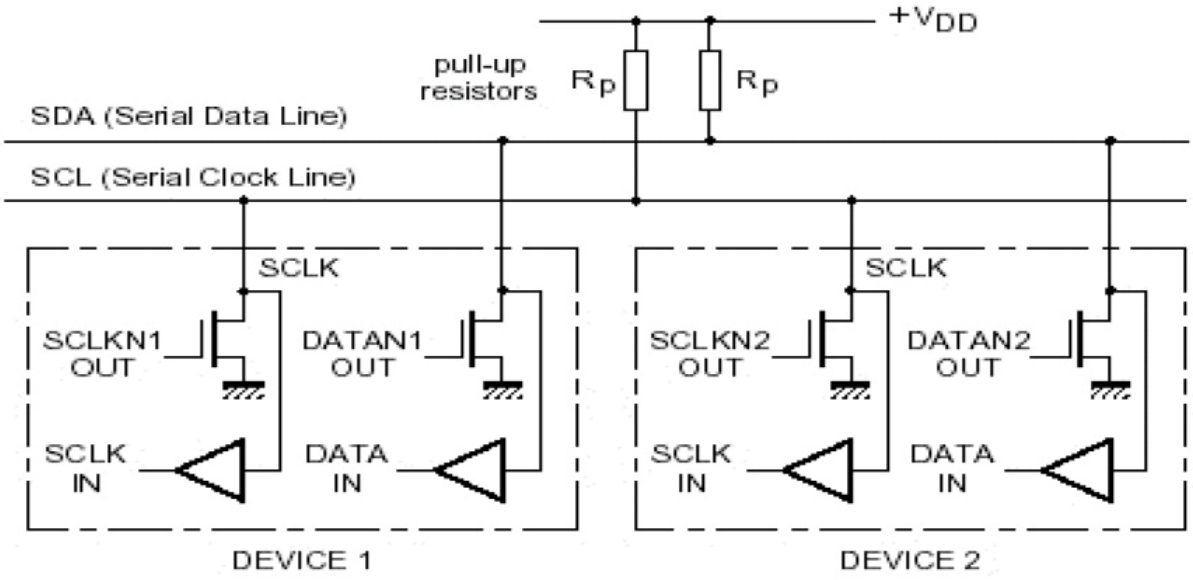


Abbildung 1: Aufbau der typischen Busstruktur

2.2 Das I²C-Protokoll

Um den Vorgang der Datenübertragung auf dem I²C-Bus zu erläutern, möchte ich zuerst kurz zeigen, welche Schritte zur Datenübertragung notwendig sind.

1. Es findet kein Datenaustausch auf dem Bus statt, der Bus ist frei. Das heißt alle Geräte sind gleichberechtigt.
2. Ein Gerät möchte einen Datentransfer starten, dazu sendet es die START-Kondition. Dadurch wird dieses Gerät automatisch zum Master und alle anderen Busteilnehmer werden zu Slaves und hören die Daten auf dem Bus ab.
3. Der Master sendet die Adresse des gewünschten Kommunikationspartners. Alle Slaves lesen diese Adresse ein und vergleichen diese mit ihrer eigenen Busadresse. Wenn ein Gerät nicht die gesuchte Adresse besitzt, wartet es solange bis eine STOP-Kondition über den Bus gesendet wird. Das Gerät, welches jedoch die gesuchte Adresse besitzt, sendet ein Acknowledge zurück an den Master.
4. Wenn der Master das Acknowledge empfangen hat, beginnt er mit der Datenübertragung. Nachdem alle Daten übertragen wurden, sendet der Master eine STOP-Kondition. Dadurch wird die Übertragung beendet und der Bus wieder freigegeben.

Die Datenübertragung auf dem I²C-Bus findet in Datenpaketen zu je 8 Bit statt, nach jedem dieser Datenpakete sendet der Slave ein Acknowledge an den Master um den Empfang zu bestätigen.

2.2.1 Die START- und STOP-Kondition

Die START-Kondition wird erzeugt, indem der Master zuerst die SDA-Leitung auf LOW legt und anschließend auch die SCL-Leitung auf LOW zieht.
Bei der STOP-Kondition wird zuerst die SCL-Leitung freigegeben und anschließend die SDA-Leitung.

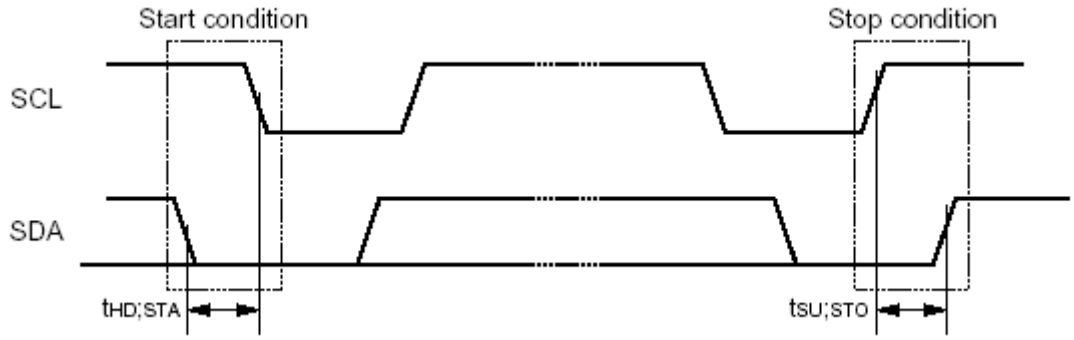


Abbildung 2: START- und STOP-Kondition

Eine STOP-Kondition darf jederzeit gesendet werden. Wenn dies eintrifft, wird die Bus-Control-Logic bei allen Busteilnehmern zurückgesetzt und jedes Gerät wartet auf eine erneute START-Kondition.

2.2.2 Übertragung der Daten

Nach der START-Kondition können die Daten übertragen werden. Dabei wird zuerst die Geräteadresse übertragen und danach die Nutzdaten. Die Daten auf SDA sind nur gültig wenn SCL auf HIGH liegt.
Zum Übertragen eines Bits setzt der Master die Datenleitung SDA auf den entsprechenden Wert und schaltet SCL auf HIGH. Damit sind die Daten auf SDA gültig und können vom Slave eingelesen werden. Anschließend zieht der Master SCL wieder auf LOW und erklärt die Daten auf SDA somit für ungültig und kann anschließend die Daten auf SDA ändern.

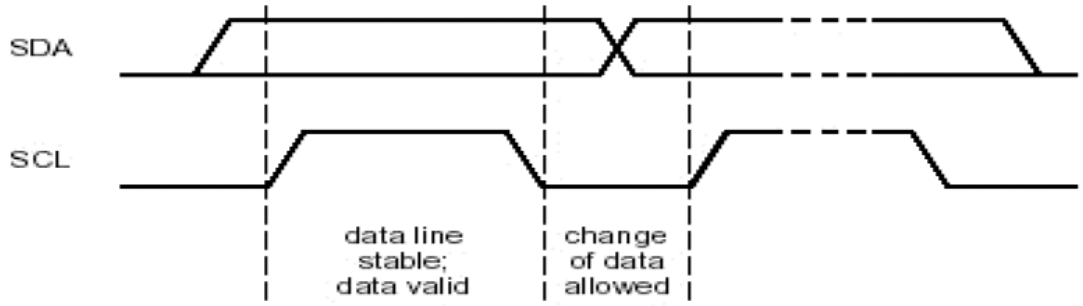


Abbildung 3: Übertragung eines Bits

2.2.3 Synchronisation des Taktsignals

Damit der Datentransfer auf dem Bus problemlos abläuft muss sichergestellt sein, dass auch der langsamste Busteilnehmer den Takt korrekt verarbeiten kann. Dafür wird die Taktsynchronisation verwendet.
Für diese Funktion wird wieder die bekannte „wired-and“-Funktionalität des Busses verwendet und zwar folgendermaßen. Eine fallende Flanke auf SCL veranlasst alle betroffenen Busteilnehmer einen internen Zähler zu starten und ihren SCL-Ausgang auf LOW zu setzen. Wenn dieser interne Zähler abgelaufen ist, d.h. wenn der Baustein die Verarbeitung abgeschlossen hat, setzt der betroffene Baustein seinen SCL-Ausgang wieder auf HIGH. Nun greift die „wired-and“-Funktionalität mit ein, denn die Taktleitung geht erst

wieder auf HIGH, wenn der letzte Zähler abgelaufen ist und wenn der letzte Busteilnehmer aufhört SCL auf LOW zu ziehen. Somit müssen alle schnelleren Geräte warten, bis auch das langsamste Gerät die Verarbeitung beendet hat. Bei der darauf folgenden steigenden Flanke der Taktleitung besteht somit kein Unterschied mehr zwischen dem Takt der einzelnen Busteilnehmer und dem Takt auf dem Bus und die beteiligten Geräte starten einen internen HIGH-Zähler. Das erste Gerät, dessen HIGH-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 mit der längsten LOW-Phase und dessen HIGH-Phase durch das Gerät mit der kürzesten HIGH-Phase bestimmt wird.

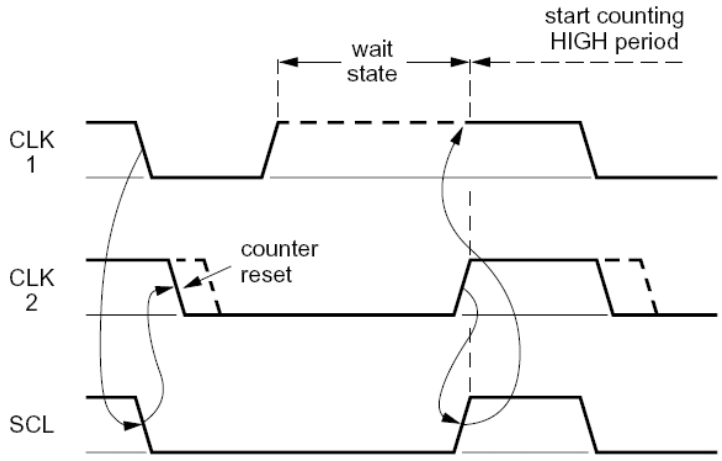


Abbildung 4: Taktsynchronisation

2.2.4 Arbitrierung

Die bereits erwähnte „wired-and“-Funktionalität des I²C-Busses ist Voraussetzung für den so genannten Multimasterbetrieb. Dies bedeutet, dass mehrere Master auf dem Bus arbeiten können, ohne dass es zu Datenkollisionen kommt. Um eine korrekte Datenübertragung zu gewährleisten, findet vor der eigentlichen Datenübertragung ein Auswahlprozess (Arbitrierung) statt. Bei diesem Auswahlprozess erhält ein bestimmter Master die Buszuteilung, während alle anderen Master abgeschaltet werden und auf das Ende des Datentransfers warten. Erst wenn der Master, welcher die Arbitrierung „gewonnen“ hat, den Bus wieder frei gibt, können die anderen Master versuchen die Buszuteilung zu bekommen. Solange die Master bei der Arbitrierung die gleichen Daten senden, kann keine Entscheidung stattfinden. Wird jedoch gleichzeitig versucht, von einem Master eine „1“, von einem zweiten eine „0“ auf den Bus zu geben, so stellt sich auf diesem, wegen der „wired-and“-Funktionalität, LOW-Pegel ein und somit wird eine „0“ empfangen. Wenn der zugehörige Taktimpuls nun HIGH wird, vergleichen beide Master den Pegel auf der Datenleitung SDA mit ihrem internen Ausgangspegel. Der Master, der hier einen Unterschied feststellt, schaltet seine Ausgangsstufe sofort ab und wartet auf die STOP-Kondition um danach ein erneutes Senden zu versuchen. Der Master, der die Arbitrierung „gewonnen“ hat, sendet dagegen weiter seine Daten.
Im Allgemeinen geschieht dieser Vorgang schon beim Adresstransfer. Der „verlierende“ Master schaltet dann sofort auf die Betriebsart „Slave“ um, da er vom „gewinnenden“ Master auch adressiert werden könnte.

2.3 Adressierung

Jedes Gerät am I²C-Bus besitzt eine eindeutige Adresse, mit welcher es am Bus identifiziert werden kann. Im Normalfall besteht eine solche Adresse aus 7 Adressbits und einem R/W-Bit, mit welchem gesteuert wird, ob der Master lesend oder schreibend auf den Baustein zugreifen will. Mit diesen 7 Adressbits sind 128 Zustände möglich und somit könnte man theoretisch 128 Bausteine am Bus adressieren.
Es gibt jedoch eine „General Call Address“, diese besteht aus 8 Nullbits und hat die Aufgabe alle Busteilnehmer auf einmal zu adressieren. Wenn diese „General Call Address“ auf den Bus gegeben wird, bekommen alle Slaves die Aufgabe, die Daten vom Master mitzuhören.
Die restlichen 127 möglichen Bauteiladressen wurden jedoch sehr schnell knapp und somit wurde ein „Extended Addressing Mode“ eingeführt. Dieser ermöglicht Adressen von bis zu 10 Bit.
Um diesen erweiterten Adressierungsmodus zu verwenden gibt es folgende Regelung. Die Adresse 1111 0xxx ist für den „Extended Addressing Mode“ reserviert, d.h. wenn diese Adresse über den Bus gesendet wird, wissen die Busteilnehmer, dass das folgende Datenbyte keine „Nutzdaten“ enthält, sondern die restlichen Bit der Adresse.
Durch dieses Verfahren wird die Abwärtskompatibilität zu Geräten, die den „Extended Addressing Mode“ nicht unterstützen, sichergestellt, da diese Geräte die längeren Adressen einfach ignorieren.

	reservierte Adresse					Adresse		R/$\overline{W}$		Adresse, 2.Teil					Geräte-Sub-Adresse			
Start	1	1	1	1	0	x	x	1	ACK	x	x	x	x	x	x	x	x	ACK

Abbildung 5: Extended Addressing Mode

Es gibt aber noch weitere Adressen, die für spezielle Funktionen reserviert sind.
Hier ein kurzer Überblick:

SLAVE ADDRESS	R/$\overline{W}$ BIT	DESCRIPTION
0000 000	0	General call address
0000 000	1	START byte(1)
0000 001	X	CBUS address(2)
0000 010	X	Reserved for different bus format(3)
0000 011	X	Reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	Reserved for future purposes
1111 0XX	X	10-bit slave addressing

Abbildung 6: Reservierte Adressen

	Adresse				Geräte-Sub-Adresse			R/$\overline{W}$		Daten								
Start	0	1	0	0	x	x	x	1	ACK	x	x	x	x	x	x	x	x	Stop

Abbildung 7: Lesender Zugriff auf einen PCF8574

2.4 Die Geschwindigkeitsmodi

2.4.1 Normaler Geschwindigkeitsmodus

Der Standard-I²C-Bus arbeitet mit einer Geschwindigkeit von 100 kbit/s.

2.4.2 Fast-Mode

Da der I²C-Bus am Anfang nur zum Austausch von Status-Informationen vorgesehen war, reichten damals die 100 kbit/s aus. Mit der Zeit kamen allerdings immer mehr Bausteine hinzu und der Bus wird heutzutage auch zur Übertragung von Text und Daten verwendet. Dazu reichen allerdings die 100 kbit/s nicht mehr aus und somit bietet der Fast-Mode eine maximale Datenübertragungsrate von 400 kbit/s.
Im Vergleich zum „normalen Modus“ wurden weder das Bus-Protokoll, noch die physikalischen Parameter geändert und somit sind alle Fast-Mode-Geräte abwärtskompatibel.
Heutzutage sollten alle modernen Bausteine diesen Geschwindigkeitsmodus unterstützen.

2.4.3 Highspeed-Mode

Für spezielle RAM-Bausteine, die noch höhere Datenübertragungsraten forderten, wurde der Highspeed-Mode mit 3,4 MBit/s eingeführt.
Geräte des Fast- oder Normal-Mode kann man nur zusammen mit Highspeed-Mode Geräte an einem Bus betreiben, wenn man die Geschwindigkeit auf 100 bzw. 400 kbit/s heruntersetzt. Da sich, außer Arbitrierung und Synchronisation, nichts am Busprotokoll geändert hat, ist auch der Highspeed-Mode abwärtskompatibel.
Man kann aber auch Normal- / Fast-Mode Geräte und Highspeed-Geräte zusammen in einem System verwenden, ohne auf die Geschwindigkeitsvorteile zu verzichten. Hierfür verwendet man einen Master, der für den Normal- / Fast-Mode zwei Pins besitzt und für den Highspeed-Mode zwei extra Pins. Die Konvertierung zwischen den beiden Bussen muss in diesem Fall der Master übernehmen.

3. Adresstabelle verschiedener Busteilnehmer

	LSB							
MSB	000x	001x	010x	011x	100x	101x	110x	111x
0010	SAA5240 SAA4700 SAF1134 SAF1135	SAA5240 SAA5241 SAA5243 SAA5244 SAA5246 SAA9041 SAA4700 SAF1134 SAA1135	SAA5240 SAB9070 SAF1134 SAA5244	SAF1134 SAF1135 SAF1135	SAA5252 SAA9020	SAA9020	SAA9020	SAA9020
0011	SAA7250 PCB5020 PCB5021 PCB5032	SAA7250 PCB5020 PCB5021 PCB5032			SAA1136 PCF1810	PCF1810	PCF1810 SAA1770	PCF1810 SAA1770
0100	SAA1137 PCD4430 SAA7194 PCF8574 TDA8444	SAA1137 PCD4430 SAA7194 PCF8574 TDA8444	PCA1070 PCF8574 TDA8444	PCA1070 PCF8574 TDA8444	PCD3311 PCD3312 PCF8574 TDA8444	PCD3311 PCD3312 PCF8574 TDA8444	SAB3028 PCF8574 TDA8444	PCD5002 PCF8574 TDA8444
0101								
0110								
0111	PCF8576 SAA1064 PCF8574A	PCF8576 SAA1064 PCF8574A	PCF8577 SAA1064 PCF8574A	PCF8577A SAA1064 PCF8574A	PCF8578 PCF8579 PCF8574A	PCF8566 PCF8574A	PCF8566 PCF8574A	
1000	TEA6320 TEA6330 TDA8420 TDA8421 TDA8960 NE5751	TDA8424 TDA8425 TDA8426 TDA8420 TDA8421 TDA9860 NE5751	TDA8425 TDA8415 TDA8416 TDA8440 TDA8940 TDA8417 TDA6360 TDA8480	TDA6360 TDA8480				
1001	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591
1010	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582 PCF8583	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582 PCF8583	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582
1011	SAA7199 PCF8570	SAA7199 SAA7152 PCF8570	TDA8416 PCF8570	PCF8570	TDA2518 SAA7186 PCF8570	PCA8510 PCA8516 PCF8570	SAA7186 PCF8570	SAA9065 PCF8570
1100	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010 TEA6000 TEA6100	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010 TSA6057 PCA8516 TSA6060 TSA6061	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010 TSA6057 PCA8516 TSA6060 TSA6061	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010
1101	TDA8443 PCF8573	TDA8443 PCF8573	TDA8443 PCF8573	TDA8443 PCF8573	TDA8443 UMA1000 TDA1551	TDA8443 UMA1000	TDA8443 UMA1000 PCD4440	TDA8443 UMA1000 PCD4440
1110	SAA7192	SAA7192						

4. Quellenangaben

http://www.computerbase.de/lexikon/I2C

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

http://www.embedded.com/story/OEG20010718S0073

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

http://homepages.which.net/~paul.hills/Software/I2C/I2C.html

http://www.semiconductors.philips.com/markets/mms/protocols/i2c/

http://vhimpe.crosswinds.net/electronics/i2cfaq/i2c.html

http://www.bbs-winsen.de/GoBlack/ETechnik/digitalt/i2c/signale.html

http://www.semiconductors.philips.com/acrobat_download/literature/9398/39340011.pdf

http://www.elektronik-kompendium.de/public/borchers/i2c/whatis.htm#10bit

http://www.elektronik-kompendium.de/public/borchers/i2c/addresses.htm