Fachhochschule Gelsenkirchen
Physikalische Technik

Arbeitsunterlagen zum Praktikum Mikroprozessortechnik II

Versuch 4: Datenübertragung

Inhaltsverzeichnis

Seite

Abbildungsverzeichnis

1 Einleitung

In diesem Versuch werden zwei verschiedene Standards zur Datenübertragung behandelt.
Zunächst wird ein UART (Universal Asynchronous Receiver Transmitter) untersucht. UARTs sind Schnittstellenbausteine zur seriellen Datenübertragung und in vielen Mikrocontrollern bereits implementiert. Zur Datenübertragung werden verschiedene Protokolle verwendet, einige werden im Folgenden vorgestellt. Im praktischen Teil dieses Versuches wird das RS232- Protokoll verwendet.
Des Weiteren wird die synchrone Übertragung mittels des I²C-Bus gezeigt und angewendet . Die I²C-Schnittstelle ist auch auf dem verwendeten Mikrocontroller implementiert.

Im praktischen Teil soll eine Temperaturmessung aufgebaut werden, die Temperatur soll auf dem PC angezeigt werden. Zur Temperaturmessung wird der Sensor DS1621 von der Fa. Dallas Semiconductor mit einer I²C-Schnittstelle eingesetzt. Er soll die gemessenen Temperaturwerte mittels seiner Schnittstelle zum Mikrocontroller ATmega16 übertragen. Der Mikrocontroller soll die Temperaturwerte weiterverarbeiten und mittels der RS232-Schnittstelle zum PC übertragen.

2 Synchrone und Asynchrone Übertragung

2.1 Synchrone Übertragung

Die synchrone Übertragung serieller Daten findet häufig auf Leiterplattenebene, etwa zum Austausch von Daten zwischen verschiedenen integrierten Bausteinen innerhalb einer Schaltung statt. Zum Beispiel zwischen einem Mikrocontroller und mehreren peripheren Schaltkreisen zur Videosignalverarbeitung oder Tonsteuerung über den I^2C- Bus eines Fernsehgerätes.
Bei der synchronen Übertragung serieller Daten wird mit der Übertragung der einzelnen Datenbits auch ein hierzu synchrones Taktsignal vom Sender übertragen. Die Anforderungen an dieses Taktsignal bezüglich des Timings sind dabei weitgehend unkritisch. So kann z. B. bei vielen synchronen Übertragungsprotokollen der Takt zur Anpassung an langsamere Busteilnehmer zeitlich gedehnt werden.

2.2 Asynchrone Übertragung

Hauptanwendungsfeld für die asynchrone Datenübertragung ist in der Regel der Austausch von Daten zwischen räumlich weit getrennten Einheiten mit eigener Intelligenz. Als Beispiel sei hier die Verbindung zwischen einem PC und einem Drucker, einem Modem, einem Programmiergerät oder einem Datenlogger genannt.

3 Asynchrone Datenübertragung über den UART

Bei der asynchronen Übertragung wird kein Taktsignal mit übertragen. Dies stellt strenge Anforderungen an das Timing des Sende- und Empfangskanals. Die Zeitbasis für ein asynchron arbeitendes Datenübertragungssystem ist deshalb in den meisten Fällen quarzstabilisiert.
Da die asynchrone Datenübertragung auch über größere Entfernungen von einigen hundert Metern erfolgen kann, sehen ihre Protokolle keine Übertragung mit den sehr störungsanfälligen TTL-Pegeln vor.

3.1 Die gebräuchlichsten Übertragungsprotokolle

Der UART gibt seine Daten meist als TTL-Pegel aus. Zur Übertragung über größere Distanzen ist dieser Pegel nicht geeignet. Zur Übertragung muss eine Signalanpassung vorgenommen werden. Mit welchen Pegeln das Signal übertragen wird ist im Übertragungsprotokoll festgelegt.

3.1.1 Strom- oder 20 mA-Schnittstelle

Eine sehr störungssichere Schnittstelle für die asynchrone Übertragung bis zu einer Baudrate von 9600 Baud stellt die Strom- oder 20 mA-Schnittstelle dar. Sie ist zwar nicht genormt, hat aber durch ihre einfache Handhabung seit vielen Jahren große Verbreitung gefunden.

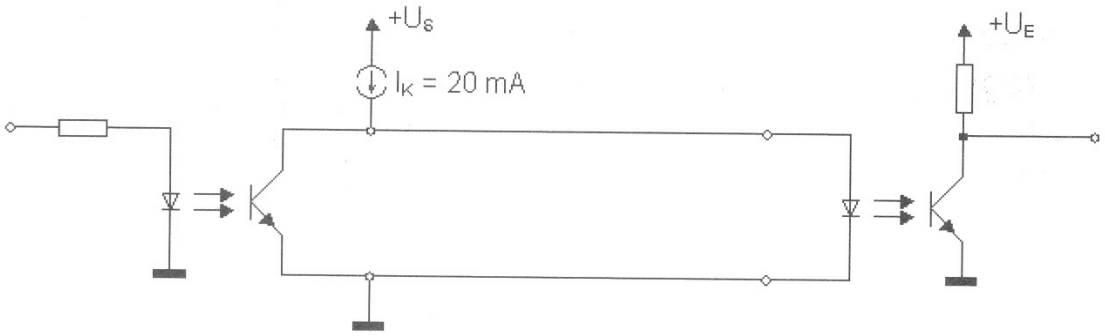


Abbildung 1: Beispiel einer Sendeschleife für die Stromschnittstelle

Die Verbindung zweier Geräte erfolgt hier über eine geschlossene Sende- und eine geschlossene Empfangsschleife. In die Schleifen wird auf der aktiven Seite ein Konstantstrom von 20 mA eingeprägt. Um einen LOW-Pegel zu übertragen, wird der Stromfluss unterbrochen, bei der Übertragung eines HIGH-Pegels bleibt die Schleife geschlossen.
Bei der Stromschnittstelle lässt sich eine Potentialtrennung zwischen Sende- und Empfangsseite über Optokoppler leicht realisieren, wie Abbildung 1 zeigt. Sie ist deshalb auch für weitere Strecken von bis zu 1000 m geeignet und findet im industriellen Einsatz nach wie vor große Verwendung.

3.1.2 RS232C- oder V.24 - Schnittstelle

Die bereits veraltete, aber noch immer weit verbreitete RS232C- oder V.24 – Schnittstelle überträgt LOW- und HIGH-Pegel als Spannungswerte. Auf der Empfangsseite ist dabei einem TTL-HIGH-Pegel (+2 ... +5V) der Spannungsbereich -3 … -15V („MARK") und einem TTL-LOW-Pegel (0 V … +0,8 V) der Spannungsbereich +3 ... +15V („SPACE") zugeordnet. Auf der Sendeseite ist die Untergrenze zum Ausgleich von Spannungsverlusten auf der Leitung auf +5 V bzw. -5 V angehoben. Abbildung 2 zeigt die laut Norm zulässigen Pegel. Für die erforderliche Pegelwandlung von TTL- auf V.24-Standard ist eine Reihe von integrierten Schaltungen erhältlich. Die wohl am weitesten verbreiteten Treiberbausteine für diesen Zweck dürften der MAX232 der Firma *Maxim* oder eines seiner zahlreichen Derivate sein, die aufgrund ihres eingebauten Spannungswandlers mit einer einzigen Betriebsspannung von +5 V auskommen.

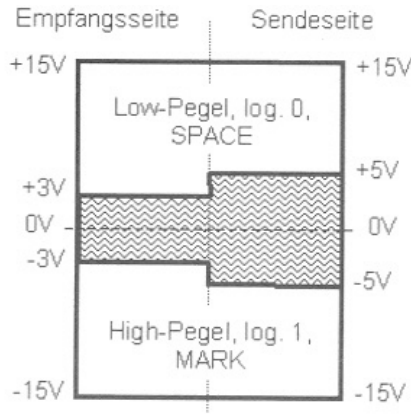


Abbildung 2: Spannungspegel nach RS232C-Norm

Die RS232 bzw. V.24 Norm beschreibt auch die Möglichkeit, den Datenfluss zwischen Sender und Empfänger durch spezielle Freigabe-Signale zu steuern. Diese Signale werden auch „Handshake-Signale" genannt. Zum Aufbau einer Handshake-Verbindung sieht die Norm Steuersignale wie CTS (Clear To Send), RTS (Ready To Send) vor, die aber nicht zwingend verwendet werden müssen. Sie sollten aber auch nicht einfach unbeschaltet bleiben, da dies zu Fehlinterpretationen im Protokoll führen kann. Im einfachsten Fall genügt für die Kommunikation zwischen zwei Geräten eine einfache 3-Draht-Verbindung, bestehend aus den gekreuzten Sende- und Empfangsleitungen TxD und RxD und der Masseleitung. Die nicht benutzten Steueranschlüsse können wie in Abb. 6.3 gezeigt belegt werden.

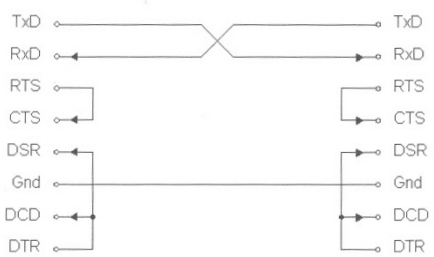


Abbildung 3: Einfache 3-Draht-Verbindung zwischen zwei Geräten

In der Praxis lassen sich mit der RS232C-Schnittstelle größere Übertragungsraten als 19.200 Baud erreichen, aber wegen der galvanischen Verbindung und den dadurch entstehenden Störungen durch Ausgleichsströme auf der Masseleitung sind größere Entfernungen als 15 ... 20 m dann nicht überbrückbar, obwohl die Norm eine Obergrenze von 100 ft = 30,5 m festlegt.

3.1.3 RS423A- oder V.11-Schnittstelle

Die wachsenden Anforderungen an Übertragungsreichweite und -geschwindigkeit haben zu neuen Normen für die asynchrone Datenübertragung geführt. Mit der RS423A- oder V.11-Schnittstelle sind Baudraten bis zu 100.000 Baud und Reichweiten bis zu 1200 m möglich. Verbunden werden Sender und Empfänger bei der RS423A-Schnittstelle über ein mit dem Wellenwiderstand *Z* abgeschlossenes 50 Ω-Koaxialkabel, wie in Abbildung 4 gezeigt.

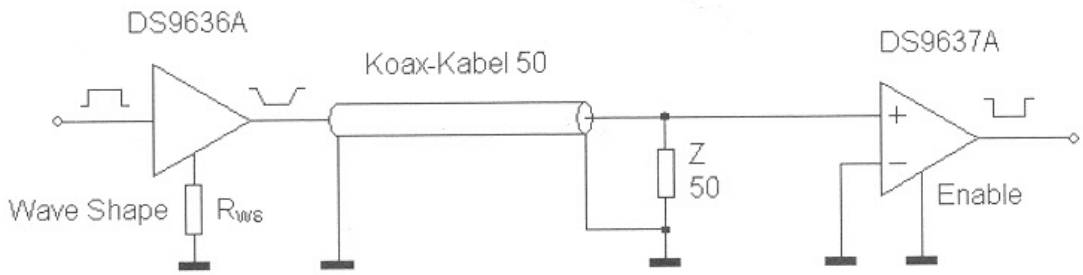


Abbildung 4: Übertragung nach der RS423A-Norm über Koaxialkabel

Treiber für die erforderliche Pegelwandlung von TTL- auf V.11-Standard sind von vielen Herstellern erhältlich. Mögliche Beispiele wären der RS423A-Sender DS9636A und der RS422A / RS423A-Empfänger DS9637A der Firma National Semiconductor.
Mit dem Widerstand R_{WS} kann die Anstiegs- / Abfalldauer der Ausgangsspannung des Treibers DS9636A in den Grenzen von 1 µs bis 100 µs eingestellt werden.
Die RS423A-Schnittstelle arbeitet wie die RS232C-Schnittstelle unsymmetrisch, d.h. die Masseleitung wird als Rückleitung benutzt. In Umgebungen mit hohem Störpegel ist der Einsatz dieser Schnittstelle deshalb nicht sinnvoll.
Wenn keine zu großen Entfernungen zu überbrücken sind, kann die Schaltung nach Abbildung 4 jedoch eine spürbare Verbesserung gegenüber einer RS232C-Verbindung bringen.

3.1.4 RS422A- oder V.10 - Norm

Noch vorteilhafter lässt sich eine Schnittstelle nach der RS422A- oder V.10 - Norm einsetzen. Die Datenübertragung erfolgt hier symmetrisch, d.h. der Sende-Treiber hat eine Brückenausgangsstufe, über die das Signal und das invertierte Signal gleichzeitig ausgegeben werden. Die Information ist hier also nicht in einer Absolutspannung zwischen Ausgang und Masse, sondern in der Differenz zwischen den beiden Ausgangsspannungen enthalten. Durch die Verwendung einer verdrillten Leitung erzielt man hohe Störsicherheiten, da sich eingestreute Störimpulse gegenseitig aufheben.

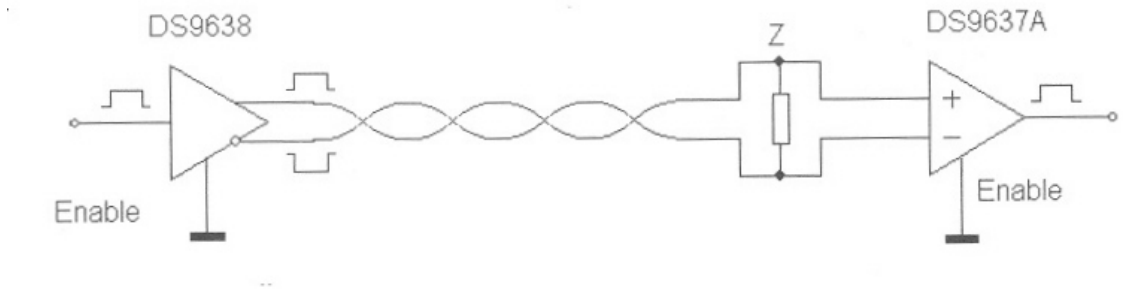


Abbildung 5: Übertragung nach RS422A über verdrillte Leitung

Ein Beispiel verdeutlicht den Sachverhalt: Liegt der Ausgang des Treibers auf +3 V, so liegt sein invertierender Ausgang an der zweiten Leitung entsprechend auf -3 V, die Differenz zwischen beiden Ausgangsspannungen beträgt 6 V. Wenn sich beide Spannungen auf den Leitungen durch einen Störimpuls um +1 V erhöhen, hat die erste Leitung ein Potential von +4 V gegen Masse, die zweite Leitung ein Potential von -2 V gegen Masse, so dass die Differenzspannung nach wie vor 6 V beträgt.
Durch diese Methode sind Baudraten bis zu 10 MBaud und Reichweiten bis zu 1200 m möglich.
Für die RS422A-Schnittstelle bieten viele Hersteller geeignete Treiber für die erforderliche Pegelwandlung von TTL- auf den RS422A -Standard an. Ein mögliches Beispiel für den RS422A-Sender wäre der DS9638 der Firma National Semiconductor. Für den Empfang ist wie bei der RS423A-Norm der DS9637A verwendbar.

3.1.5 Übersicht der Schnittstellenparameter

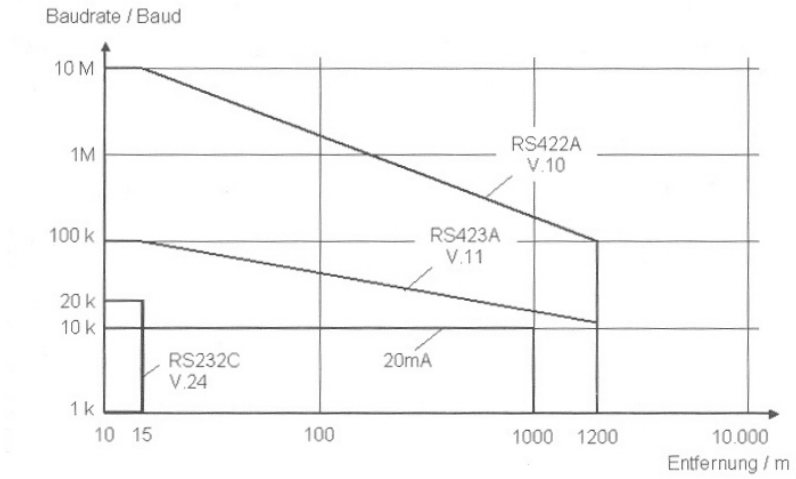


Abbildung 6: Erreichbare Übertragungsrate in Abhängigkeit von der Distanz für die verschiedenen asynchronen Schnittstellennormen

Abbildung 6 zeigt die möglichen Übertragungsraten in Abhängigkeit von der gewünschten Übertragungsentfernung für alle vorgestellten Schnittstellen.

Eine Übersicht der unterschiedlichen Schnittstellenparameter zeigt die folgende Tabelle:

	20mA-Schnittstelle	RS232C	RS423A	RS422A
Übertragungsart	symmetrisch	unsymmetrisch	unsymmetrisch	symmetrisch
Leitungsart	verdrillt	verdrillt	koaxial	verdrillt
maximale Datenrate [Baud]	10k	20k	100k	10M
maximale Leitungslänge [m]	1000	15	1200	1200
max. Ausg.-Signal unbelastet	20 mA	±25 V	±6 V	±6 V Diff.
Ausgangssignal belastet	20 mA	±5 V	±3,6 V	±2 V Diff.
Eingangsempfindlichkeit	10 mA	±3 V	±0,2 V	±0,2 V Diff.

3.2 Das Übertragungsformat der asynchronen Schnittstelle

Nach Definition der V.24-Norm liegt die Datenleitung im Ruhezustand (IDLE) auf log. 1 (MARK). Die Übertragung kann zu einem beliebigen Zeitpunkt beginnen. Um dem Empfänger den Beginn der Übertragung anzuzeigen, wird ein Startbit mit log. 0-Pegel (SPACE) ausgesendet. Danach folgen die Datenbits (meist 7 oder 8, ihre Anzahl wird im Übertragungsprotokoll vereinbart), wobei das niederwertigste Bit (LSB = Least Significant Bit) zuerst ausgesendet wird. Optional kann ein Paritätsbit (PARITY) zur Erhöhung der Datensicherheit angehängt werden. Das Paritätsbit ergänzt die Datenbits in der Weise, dass gerade Parität (die Quersumme aller Bits einschließlich des Paritätsbits ist gerade) oder ungerade Parität (die Quersumme über alle Bits einschließlich des Paritätsbits ist ungerade) erreicht wird. Die gewünschte Art der Parität (keine / gerade / ungerade) muss ebenfalls im Übertragungsprotokoll vereinbart werden. Abgeschlossen wird eine Übertragung durch das Stoppbit mit log. 1 - Pegel (MARK), das bei einigen Controllern mit einer Dauer von 1, 1,5 oder 2 Bitlängen programmiert werden kann. Nach Aussenden des Stoppbits hat die Datenleitung ihren Ausgangsruhezustand wieder erreicht und ist bereit für den nächsten Datentransfer. Abbildung 7 zeigt den zeitlichen Ablauf der Übertragung des Bytes 0xE5 (0b11100101) mit ungerader Parität und einfachem Stoppbit über die V.24-Schnittstelle.

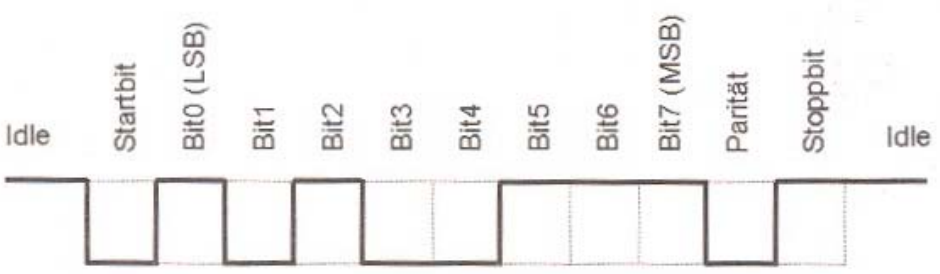


Abbildung 7: Zeitlicher Ablauf der Übertragung des Bytes mit ungerader Parität und einfachem Stoppbit über die V.24-Schnittstelle

Da ein Takt zur Synchronisation der Übertragung nicht mit gesendet wird, kann der Anfang eines Bits nur aus der Messung der vergangenen Zeit seit der fallenden Flanke des Startbits erkannt werden. Es ist deshalb notwendig, bestimmte Baudraten, d.h. Übertragungstaktraten zu vereinbaren, die für die Sende- und die Empfangsseite verbindlich sind. In der Norm für die RS232C-Schnittstelle sind diese Baudraten definiert. Sie müssen präzise eingehalten werden, um Fehler bei der Erkennung der einzelnen Bits zu vermeiden. Abbildung 8 zeigt anhand des obigen Beispiels die Konsequenz, wenn die Abweichung zwischen Sende- und Empfangstakt zu groß wird, und z.B. immer mit der steigenden Taktflanke auf Senderseite ausgesendet und mit der fallenden Taktflanke auf Empfängerseite (also eigentlich genau in der Bit-Mitte) abgetastet würde.

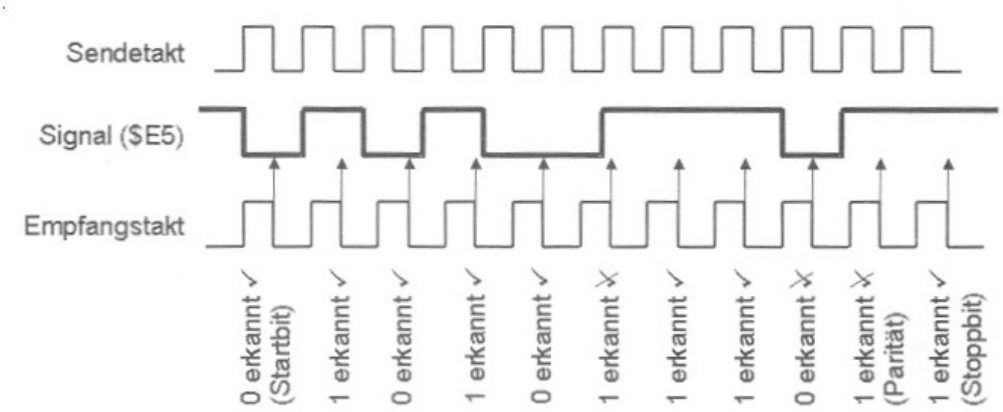


Abbildung 8: Auswirkung einer unterschiedlichen Baudrate auf Sender- und Empfängerseite

In Abbildung 8 werden Bit 4 und Bit 7 sowie das Paritätsbit falsch erkannt. Da wie im obigen Beispiel ungerade Parität vereinbart worden ist, wäre der Fehler zwar anhand des Paritätsbits bemerkt worden, da das als 0x75 erkannte Byte und ein gesetztes Paritätsbit gerade Parität ergeben, wären aber nur zwei Fehler aufgetreten, hätte das Paritätsbit keinen Fehler signalisiert.
Die Abweichung zwischen den Taktraten auf Sender- und Empfängerseite beträgt im Beispiel nach Abbildung 8 drastische 20%, um die Auswirkungen deutlich herauszustellen, aber als Grenzwert für eine zuverlässige Übertragung sollten Abweichungen, die größer als 3 % sind, unbedingt vermieden werden. Auf der sicheren Seite liegt man, wenn die tatsächliche Baudrate von der genormten um nicht mehr als 2 % abweicht.

3.3 Die UART-Hardware des ATmega16

Natürlich könnte man die asynchrone Datenübertragung auch softwaremäßig nachbilden. Da dies aber einen relativ hohen Prozentsatz der CPU-Aktivität in Anspruch nimmt, hat ATMEL ihre meisten Prozessoren mit einem Hardware-UART ausgestattet, der voll-duplex-fähig und damit im Stande ist, Daten gleichzeitig auszusenden und zu empfangen, um so die CPU spürbar zu entlasten.
Für den Betrieb des UARTs sind insgesamt fünf Register im I/O-Bereich relevant. Neben den drei UART Status- und Kontrollregistern UCSRA bis UCSRC zur Steuerung der UART-Funktionen und zur Freigabe bzw. Sperrung der UART-

Interrupts sind dies das UART Datenregister UDR (das physikalisch aus zwei Registern besteht, die unter der selben Adresse angesprochen werden und von denen eines zum Senden und das andere zum Empfang der Daten benutzt wird) und das UART Baud Rate Register UBRR zur Einstellung der gewünschten Baudrate des eingebauten Baudraten-Generators, mit dem sich die gängigsten Baudraten der RS232C-Norm erzeugen lassen.
Der UART kann verschiedene Übertragungsfehler automatisch erkennen und drei verschiedene Interrupts in Zusammenhang mit den UART -Operationen auslösen.

3.3.1 Der UART-Sendeteil (Transmitter)

Abb. 6.9 zeigt ein Blockschaltbild des UART-Sendeteils. Die Datenübertragung wird durch das Schreiben des auszusendenden Bytes in das UART I/O-Datenregister UDR ausgelöst. Wenn das Einschreiben des neuen Bytes in das UDR-Register nach dem Aussenden des Stoppbits des vorhergehenden erfolgt, wird das neue Byte sofort in das Sende-Schieberegister geladen und mit dem Aussenden begonnen.

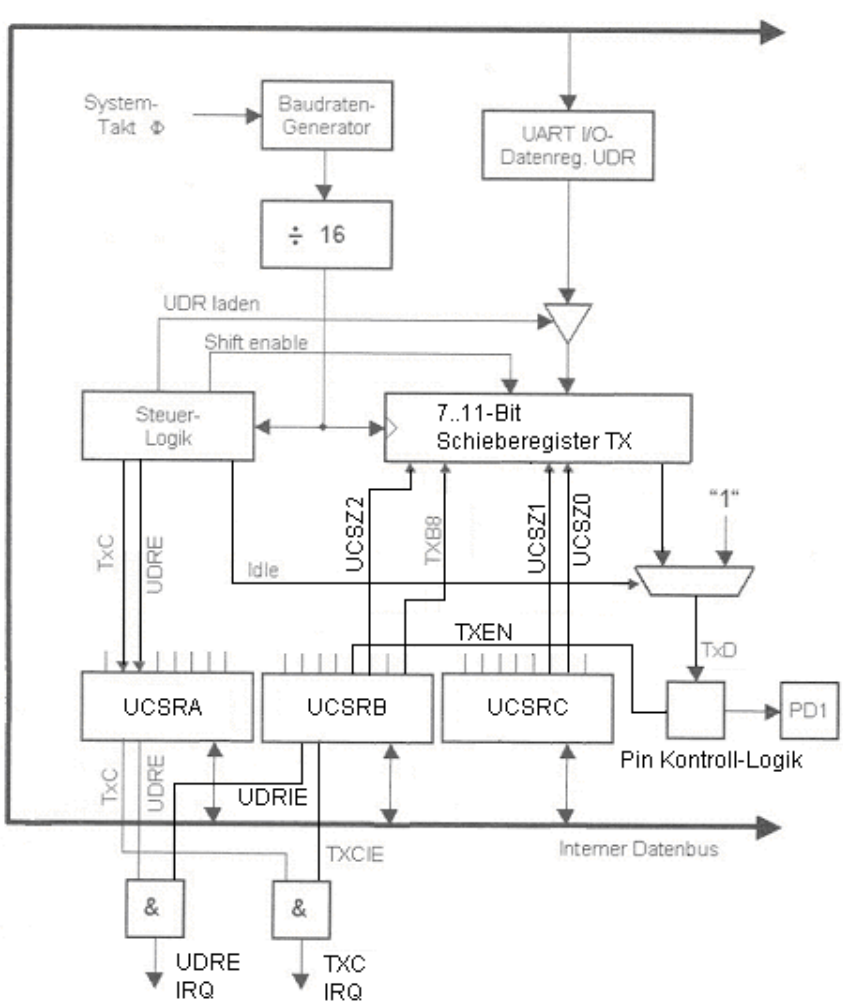


Abbildung 9: Blockschaltbild des UART-Sendeteils (Transmitter)

Trifft das neue Byte im UDR-Register während eines gerade laufenden Transfers ein, so kann es erst dann in das Sende-Schieberegister geladen und mit dem

Aussenden begonnen werden, wenn der laufende Transfer durch das Stoppbit abgeschlossen wurde.
Sobald ein Datenbyte vom UDR in das Sende-Schieberegister TX übertragen wurde, wird das Flag UDRE (UART Data Register Empty) im UART Status-Kontrollregister UCSRA gesetzt und zeigt damit an, dass der UART bereit ist, ein neues Datenbyte in sein UDR-Register zu übernehmen.
Die Gesamtzahl der ausgesendeten Bits kann 7 bis 11 betragen, je nachdem, welche Werte Datenwortlänge mit Hilfe der Bits UCSZ2 bis UCSZ0 in den Status und Kontrollregistern eingestellt ist.
Außer dem Zeichen selbst werden noch zusätzliche Bits gesendet. Sie tragen zusätzliche Informationen und können als Paritätsbit oder als weiteres Stoppbit verwendet werden. Mit der Übernahme des auszusendenden Zeichens vom UDR-Register in die Bits des Schieberegisters wird das Bit 0 (Startbit) des Schieberegisters automatisch gelöscht, das Bit TXB8 aus dem Register UCSRB in Bit 9 des Schieberegisters kopiert, die Datenwortlänge entsprechend gewählt ist und Bit 10 gesetzt (Stoppbit).
Mit dem nächsten Baudraten-Taktimpuls, der auf die Übertragung des auszusendenden Zeichens vom UDR-Register in das Schieberegister folgt, wird Bit 0 des Schieberegisters als Startbit an den Ausgangs-Pin TxD geschoben, gefolgt von den 5 bis. 9 Datenbits (LSB zuerst) und einem Stoppbit. Ist während der laufenden Übertragung ein neues Zeichen in das DDR-Register geladen worden, so wird dieses Zeichen unmittelbar nach dem Aussenden des Stoppbits in das Schieberegister transferiert und eine neue Übertragung gestartet. Mit dem Transfer dieses Zeichens in das Schieberegister wird auch das Flag UDRE im Register UCSRA gesetzt, um anzuzeigen, dass der Transmitter wieder bereit ist, ein neues Zeichen in das UDR-Register aufzunehmen.
Wartet nach dem Aussenden des Stoppbits kein neues Datenwort im UDR-Register auf seine Übertragung, bleibt das durch den Transfer des vorhergehenden Zeichens vom UDR in das Schieberegister gesetzte UDRE-Flag solange auf log. 1-Pegel, bis wieder ein Byte in das UDR-Register eingeschrieben und das UDRE-Flag damit gelöscht wird. Nach dem vollständigen Aussenden des Stoppbits wird das Flag TXC (UART Transmit Complete) im UART Status Register gesetzt, um anzuzeigen, dass ein Datenwort ausgesendet wurde und keine neuen Daten zur Übertragung anstehen.
Über das Bit TXEN im UCSRB kann der Transmitter gesperrt (TXEN = log. 0) oder freigegeben (TXEN = log. 1) werden. Ist der Transmitter gesperrt, kann der I/0- Pin PD 1 als allgemeiner Ein-/Ausgang verwendet werden. Ist er freigegeben, wird der Ausgang des Sende-Schieberegisters mit dem Pin PD1 verbunden, ungeachtet der Einstellung von DDD1 im Datenrichtungsregister DDRD.

3.3.2 Das UART-Empfangsteil (Receiver)

Abbildung 10 zeigt ein Blockschaltbild des UART- Empfangsteils (Receiver).

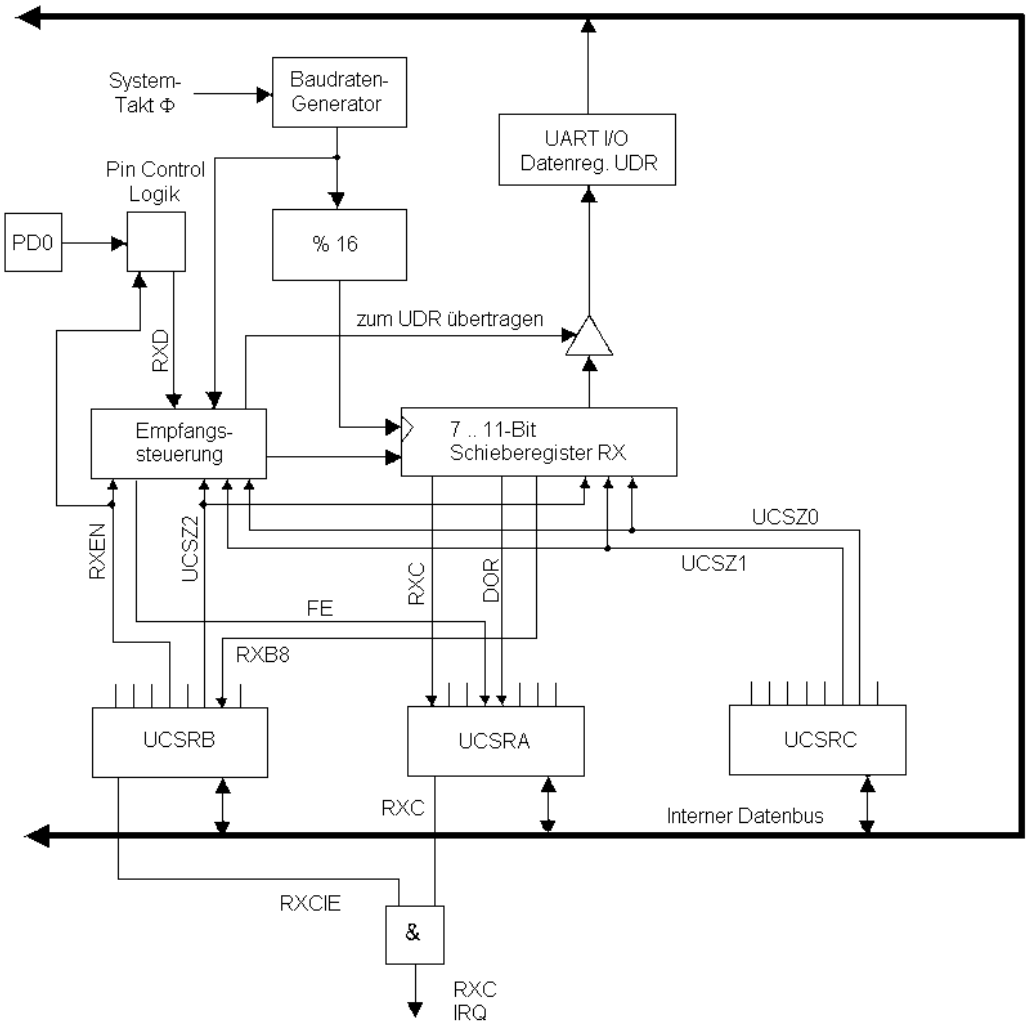


Abbildung 10: Blockschaltbild des UART-Empfangsteils (Receiver)

Die Empfangssteuerung tastet das Eingangssigna an PD0 16-mal während jeder Periode der Baudrate ab. Wird im Ruhezustand (IDLE) ein log. 0-Pegel auf der Empfangsleitung erkannt, so wertet die Logik dies als die fallende Flanke eines Startimpulses und beginnt mit der Sequenz zur Erkennung eines gültigen Startbits. Sowohl zum 8. als auch zum 9. und zum 10. Abtastzeitpunkt - gezählt von dem Abtastzeitpunkt an, an dem die fallende Flanke erkannt wurde - muss die Empfangsleitung auf LOW-Pegel liegen. Wird bei diesen drei Abtastungen zwei- oder dreimal HIGH-Pegel erkannt, wird das vermeintliche Startbit als Störimpuls zurückgewiesen, und die Empfangssteuerung beginnt erneut mit der Suche nach der fallenden Flanke eines Startbits.

Ist ein gültiges Startbit entdeckt worden, werden die seriell eintreffenden Datenbits nacheinander in das Empfangs-Schieberegister eingeschoben. Wie beim Startbit ist zur Biterkennung auch hier der Pegel auf der Empfangsleitung zum 8. wie zum 9. und zum 10. Abtastzeitpunkt, gezählt vom Anfang des betreffenden Bits an, relevant. Der logische Pegel, der bei mindestens zwei der drei Abtastungen

erkannt wurde, wird als Bitwert definiert. Abbildung 11 zeigt den Abtastvorgang bei Empfang eines aus 8 Bit bestehenden Zeichens ohne Paritätsbit.

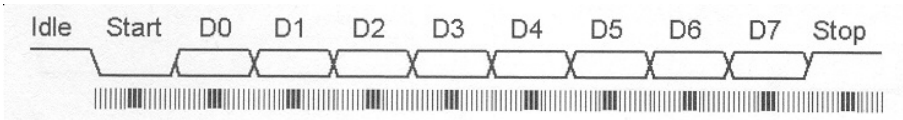


Abbildung 11: Abtastvorgang bei Empfang eines aus acht Bit bestehenden Zeichens

Für das Stoppbit müssen mindestens zwei der drei mittleren Abtastungen eine logische 1 ergeben. Ist dies nicht der Fall, wird das Framing Error Flag FE im Register UCSRB gesetzt, um anzuzeigen, dass das Stoppbit eines eintreffenden Zeichens als falsch erkannt wurde. Das Anwenderprogramm sollte das FE-Flag stets vor dem Auslesen des UDR-Registers überprüfen, um potentiell ungültige Zeichen im Empfangsregister erkennen zu können.
Nach dem Ende eines Datenempfangszyklus wird immer das RXC-Flag im Register UCSRB gesetzt und das eingelesene Zeichen in das Empfangsregister UDR geladen, gleichgültig, ob ein korrektes oder ungültiges Stoppbit erkannt wurde. Wurde mittels Bit der Bits UCSZ2 bis UCSZ0 in den Kontroll- und Status-Registern eine Datenwortlänge von 9 Bit vereinbart, so wird dieses neunte Bit in das RXB8-Bit im Register UCSRB übertragen.
Trifft ein neues Datenwort im Empfangs-Schieberegister ein, bevor ein bereits vorher empfangenes und noch im UDR befindliches Byte ausgelesen wurde, so kann das neue Zeichen nicht in das UDR übertragen werden und geht verloren. In diesem Fall wird das DOR-Flag im Register UCSRA zur Anzeige eines Datenüberlaufs gesetzt. Dieses Flag ist gepuffert, d.h., es wird nach dem Auslesen eines gültigen Zeichens aus dem UDR aktualisiert. Das Anwenderprogramm sollte deshalb das DOR-Flag stets nach dem Auslesen des UDR-Registers überprüfen, um den Verlust eines eingetroffenen Zeichens erkennen zu können.
Über das Bit RXEN im UCSRB kann der Receiver gesperrt (RXEN = log. 0) oder freigegeben (RXEN = log. 1) werden. Ist der Receiver gesperrt, kann der I/O-Pin PDO als allgemeiner Ein-/Ausgang verwendet werden. Ist er freigegeben, wird der Eingang des Receivers mit dem Pin PDO verbunden, ungeachtet der Einstellung von DDD0 im Datenrichtungsregister DDRD.

3.3.3 Die UART-Register

3.3.3.1 Das UART I/O Datenregister UDR

Das UART I/O Datenregister UDR besteht physikalisch aus zwei Registern, die unter derselben Adresse angesprochen werden und von denen eines zum Senden und das andere zum Empfang der Daten benutzt wird. Wenn das UDR ausgelesen wird, erfolgt der Zugriff auf das Empfangsregister, wenn in das UDR geschrieben wird, wird auf das Senderegister zugegriffen.
Das UDR liegt auf dem I/O-Speicherplatz $0C (RAM: 2C) es kann gelesen und beschrieben werden. Nach einem RESET wird es mit 0x00 initialisiert.

Bit	7	6	5	4	3	2	1	0	
	RXB7	**RXB 6**	**RXB5**	**RXB4**	**RXB3**	**RXB2**	**RXB1**	**RXB0**	**UDR (Read**
	TXB7	**TXB 6**	**TXB5**	**TXB4**	**TXB3**	**TXB2**	**TXB1**	**TXB0**	**UDR (Write)**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

3.3.3.2 Das UART Kontroll- und Statusregister A - UCSRA

Das UCSRA informiert das Anwenderprogramm über den Status des UART und die Arbeitsweise kann eingestellt werden. Nach einem RESET werden alle Bits bis auf Bit 5 (UDRE) mit logisch 0 initialisiert. UDRE wird auf logisch 1 gesetzt, um anzuzeigen, dass der Transmitter bereit ist, ein neues Byte zu senden.

Bit	7	6	5	4	3	2	1	0	
	RXC	**TXC**	**UDRE**	**FE**	**DOR**	**PE**	**U2X**	**MPCM**	**UCSRA**
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

RXC-Flag (UART Receive Complete)
Das RXC-Flag (UART Receive Complete) wird ohne Rücksicht auf eventuelle Rahmenfehler (Framing Errors), die möglicherweise während der Übertragung aufgetreten sind, auf logisch 1 gesetzt, wenn ein empfangenes Datenwort vom Empfangs-Schieberegister in das UDR übertragen wurde.
Falls das globale Interrupt-Flag I im Statusregister und das RXCIE-Bit im UCR-Register gesetzt sind, wird die Programmausführung durch ein gesetztes RXC-Flag auf die entsprechende UART-Receive-Complete-Adresse verzweigt. RXC wird durch das Auslesen des UDR-Registers rückgesetzt.
Will das Anwenderprogramm seine Daten per UART-Receive-Complete-Interrupt einlesen, so muss das UDR unbedingt innerhalb der Interrupt-Serviceroutine ausgelesen werden. Andernfalls bleibt das RXC-Flag gesetzt, und die Interruptroutine wird sofort nach Verlassen erneut aufgerufen.

TXC-Flag (UART Transmit Complete)
Das TXC-Flag (UART Transmit Complete) wird auf log. 1 gesetzt, wenn ein Zeichen im Sende-Schieberegister komplett, d. h. einschließlich des Stoppbits gesendet wurde und kein neues Datenbyte im UDR auf sein Aussenden wartet.

Das Flag ist besonders hilfreich im Halb-Duplex - Betrieb, wenn unmittelbar nach dem Senden in den Empfangsmodus umgeschaltet werden muss.
Falls das globale Interruptfreigabebit I im Statusregister und das TXCIE-Bit im UCR-Register gesetzt sind, wird die Programmausführung durch ein gesetztes TXC-Flag auf die entsprechende UART-Transmit-Complete-Adresse des Controllers verzweigt (0x0B).
Beim Eintritt in die Interruptroutine wird TXC hardwaremäßig rückgesetzt. Alternativ kann TXC auch durch Einschreiben einer log. 1 in Bit 6 von USR gelöscht werden.

UDRE-Flag (UART Data Register Empty)
Das UDRE-Flag (UART Data Register Empty) wird auf log. 1 gesetzt, wenn ein Zeichen aus dem UDR in das Sende-Schieberegister übertragen wurde. Dadurch wird dem Anwender signalisiert, dass der Transmitter bereit ist, ein neues Zeichen zur Übertragung aufzunehmen.
Falls das globale Interruptfreigabebit I im Statusregister und das UDRIE-Bit im UCR-Register gesetzt sind, wird die Programmausführung durch ein gesetztes UDRE-Flag auf die entsprechende UART-Data-Register-Empty-Adresse des Controllers verzweigt (0x0A). Diese Interruptroutine wird solange ausgeführt, wie das UDRE-Flag gesetzt ist.
UDRE wird durch das Einschreiben eines Bytes in das DDR-Register rückgesetzt.
Will das Anwenderprogramm seine Daten per UART-Data-Register-Empty-Interrupt ausgeben, so muss innerhalb der Interrupt-Serviceroutine unbedingt ein Zeichen in das UDR geschrieben werden. Andernfalls bleibt das UDRE-Flag gesetzt und die Interruptroutine wird sofort nach Verlassen erneut aufgerufen.
Während des Power-On-Resets wird UDRE auf log. 1 gesetzt, um anzuzeigen, dass der Transmitter bereit ist, ein neues Byte zu senden

FE-Flag (Frame Error)
Das FE-Flag (Frame Error) wird auf log. 1 gesetzt, wenn ein Rahmenfehler (Frame Error) entdeckt wurde. Das ist der Fall, wenn bei den drei mittleren Abtastungen des Stoppbits mehr als einmal eine log. 0 entdeckt und damit das Stoppbit als LOW-Pegel erkannt wurde.
Das FE-Flag wird rückgesetzt, wenn das Stoppbit HIGH-Pegel hat.
Das Anwenderprogramm sollte das FE-Flag stets vor dem Auslesen des DDR-Registers überprüfen, um potentiell ungültige Zeichen im Empfangsregister erkennen zu können.

DOR-Flag (Data Over Run)
Das DOR-Flag (Data Over Run) wird auf log. 1 gesetzt, wenn ein aus dem Empfangs-Schieberegister in das UDR übertragenes Zeichen nicht vor dem nächsten eintreffenden Zeichen ausgelesen wird. Dieses Flag ist gepuffert, d.h., es wird nach dem Auslesen eines gültigen Zeichens aus dem UDR aktualisiert. Das Anwenderprogramm sollte deshalb das DOR-Flag stets nach dem Auslesen des UDR-Registers überprüfen, um den Verlust eines eingetroffenen Zeichens erkennen zu können.
Das DOR-Flag wird rückgesetzt, wenn ein eingelesenes Zeichen in das UDR übertragen wird.

PE-Flag (Parity Error)
Dieses Flag wird gesetzt, wenn bei dem gerade empfangene Zeichen ein Paritätsfehler aufgetreten ist. Aber nur, wenn die Paritätsprüfung auch durch die Bits UPM1:0 aus dem Register UCSRC eingestellt wurde. Sobald das empfangene Zeichen aus dem Register UDR gelesen wurde, ist das Bit PE ungültig. Beim Schreiben in das Register UCSRA muss in das Bit PE immer eine null geschrieben werden.

U2X (Verdopplung der Arbeitsgeschwindigkeit des UART)
Der Vorteiler zur Taktung des UART kann von 16 auf acht gesetzt werden. Hat das Bit U2X den Wert null, so ist der Vorteiler gleich 16. Hat das Bit U2X den Wert eins, so ist der Vorteiler gleich acht.

MPCM (Multiprozessor-Kommunikationsmodus)
Durch Setzen dieses Bits Multiprozessor-Kommunikationsmodus aktiviert. Auf das Senden von Daten hat dieses Bit keinen Einfluss. Dieser Modus bewirkt beim Datenempfang, dass alle Daten, die keine Adressinformation beinhalten, ignoriert werden.

3.3.3.3 Das UART Kontroll- und Statusregister B - UCSRB

Das UCSRB informiert das Anwenderprogramm über den Status des UART und die Arbeitsweise kann eingestellt werden. Nach einem RESET werden alle Bits mit logisch 0 initialisiert.

Bit	7	6	5	4	3	2	1	0	
	RXCIE	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**UCSZ2**	**RXB8**	**TXB8**	**UCSRB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXCIE-Bit (RX Complete Interrupt Enable)
Wenn das RXCIE-Bit (RX Complete Interrupt Enable) und das globale Interrupt-freigabe-Bit I im Statusregister SREG auf logisch 1 gesetzt sind, ist der UART-Receive-Complete-Interrupt freigegeben. Er wird ausgelöst, wenn das Flag RXC im USR gesetzt wird.

TXCIE-Bit (TX Complete Interrupt Enable)
Wenn das TXCIE-Bit (TX Complete Interrupt Enable) und das globale Interruptfreigabebit I im Statusregister SREG auf log. 1 gesetzt sind, ist der UART-Transmit-Complete-Interrrupt freigegeben. Er wird ausgelöst, wenn das Flag TXC im USR gesetzt wird.

UDRIE-Bit (UART Data Register Empty Interrupt Enable)
Wenn das UDRIE-Bit (UART Data Register Empty Interrupt Enable) und das globale Interruptfreigabebit I im Statusregister SREG auf log. 1 gesetzt sind, ist der UART-Data-Register-Empty-Interrupt freigegeben. Er wird ausgelöst, wenn das Flag UDRE im USR gesetzt wird.

RXEN-Bit (Receiver Enable)
Wenn das RXEN-Bit (Receiver Enable) auf log. 1 gesetzt ist, ist der Receiver freigegeben und PD0 wird zum Eingang des UART. Hat RXEN LOW-Pegel, so ist der Empfangsteil des UART gesperrt, und PD0 kann als normaler Ein-/Ausgang verwendet werden.
Wenn RXEN auf log. 0 liegt, können die Flags OR und FE nicht gesetzt werden. Sind diese Flags allerdings gesetzt, werden sie durch das Rücksetzen von RXEN nicht gelöscht.

TXEN-Bit (Transmitter Enable)
Wenn das TXEN-Bit (Transmitter Enable) auf log. 1 gesetzt ist, ist der Transmitter freigegeben und PD1 wird zum Ausgang des UART. Hat TXEN LOW-Pegel, so ist der Sendeteil des UART gesperrt und PD1 kann als normaler Ein-/Ausgang verwendet werden.
Wird TXEN während einer laufenden Sendeoperation auf log. 0 gesetzt, so wird der Transmitter nicht gesperrt, bevor nicht das aktuelle Zeichen im Sende-Schieberegister und das eventuell noch in UDR auf sein Aussenden wartende Zeichen komplett gesendet worden sind.

UCSZ2-Bit (Zeichenlänge)
Zusammen mit den Bits UCSZ1 und UCSZ0 kann die Länge des zu versendenden Zeichens eingestellt werden. Der genaue Zusammenhang wird in CCCC erklärt.

RXB8
RXB8 ist das neunte Datenbit eines eingelesenen Zeichens, wenn mit Hilfe der Bits UCSZ2 bis UCSZ0 eine entsprechende Zeichenlänge eingestellt ist.

TXB8
TXB8 ist das neunte Datenbit eines gesendeten Zeichens, wenn mit Hilfe der Bits UCSZ2 bis UCSZ0 eine entsprechende Zeichenlänge eingestellt ist.

3.3.3.4 Das UART Kontroll- und Statusregister C - UCSRC

Das UCSRC informiert das Anwenderprogramm über den Status des UART und die Arbeitsweise kann eingestellt werden.

Bit	7	6	5	4	3	2	1	0	
	URSEL	**UMSEL**	**UPM1**	**UPM0**	**USBS**	**UCSZ1**	**UCSZ0**	**UCPOL**	**UCSRC**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

URSEL (Register Select)
Das Register UCSRC und UBRRH haben die gleiche Adresse. Durch das Bit URSEL wird eines der Register ausgewählt. Besitzt URSEL den Wert 1, so wird auf das Register UCSRC zugegriffen, ansonsten auf UBRRH.

UMSEL (UART Modus)
Durch dieses Bit kann der UART auch im synchronen Betrieb arbeiten. Ist UMSEL auf den Wert 0 gesetzt, so ist die Betriebsart asynchron. Ist UMSEL auf den Wert 1 gesetzt, so ist die Betriebsart synchron.

UPM1 und UPM0 (Paritätsmodus)
Diese Bits aktivieren und stellen die Art der Paritätsbiterzeugung und –prüfung ein. Falls aktiviert, wird die Sendeeinheit automatisch ein Paritätsbit erzeugen und zusätzlich versenden. Die Empfangseinheit wird die Parität automatisch auswerten und falls ein Paritätsfehler auftritt, das PE-Flag aus dem Register UCSRA setzen. Die folgende Tabelle zeigt die Kombinationen von UPM1 und UPM0 und die die jeweilige Betriebsart.

UPM1	UPM0	Paritätsmodus
0	0	Keine Paritätsüberprüfung
0	1	reserviert
1	0	aktiviert, Parität gerade
1	1	aktiviert, Parität ungerade

USBS (Einstellung des Stopp-Bit)
Dieses Bit legt die Anzahl der Stopp-Bits fest. Ist USBS auf den Wert 0 gesetzt, so wird ein Stopp-Bit ausgegeben. Ist USBS auf den Wert 1 gesetzt, so werden zwei Stopp-Bits ausgegeben. Für die Empfangseinheit ist die Einstellung bedeutungslos.

UCSZ1 und UCSZ0 (Paritätsmodus)
Zusammen mit dem Bit UCSZ2 kann die Länge des zu versendenden Zeichens eingestellt werden. Der genaue Zusammenhang wird in der Tabelle gezeigt:

UCSZ2	UCSZ1	UCSZ0	Zeichenlänge
0	0	0	5-Bit
0	0	1	6-Bit
0	1	0	7-Bit
0	1	1	8-Bit
1	0	0	reserviert
1	0	1	reserviert
1	1	0	reserviert
1	1	1	9-Bit

UCPOL (Receiver Enable)
Dieses Bit wird nur für den synchronen Betrieb benötigt. Bei asynchronem Betrieb sollte das Bit immer auf null gesetzt werden.

3.3.3.5 Das UART Baudraten-Register UBRR

Das UART Baudraten-Register. Es kann gelesen und beschrieben werden. Nach einem RESET wird es mit $00 initialisiert.

Bit	15	14	13	12	11	10	9	8	
	URSEL	**-**	**-**	**-**	**UBRR11**	**UBRR10**	**UBRR9**	**UBRR8**	**UBRRH**
	UBRR7	**UBRR6**	**UBRR5**	**UBRR4**	**UBRR3**	**UBRR2**	**UBRR1**	**UBRR0**	**UBRRL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

URSEL (Register Select)
Das Register UCSRC und UBRRH haben die gleiche Adresse. Durch das Bit URSEL wird eines der Register ausgewählt. Besitzt URSEL den Wert 1, so wird auf das Register UCSRC zugegriffen, ansonsten auf UBRRH. Falls UBRRH ausgelesen wird, wird URSEL immer als null gelesen.

UBRR11 bis UBRR0 (Bautratenbits)
Mit diesen zwölf Bits wird die Baudrate des UART eingestellt. Der eingebaute Baudraten-Generator ist ein Frequenzteiler, der die Baudrate direkt aus dem Systemtakt Φ erzeugt.
Die Baudrate lässt sich nach folgender Gleichung berechnen:

$$f_{Baud} = \Phi / (16\,(UBRR + 1)),$$

wobei:
f_{Baud} : Baud-Rate
Φ : Systemtakt
UBRR : Inhalt des 8 Bit-Baudraten-Registers UBRR (0...4095)

Für die gebräuchlichsten Baudraten sind die Werte von UBRR in der nachfolgenden Tabelle zusammengestellt. Alle Werte, für die sich ein Fehler kleiner 2 % ergibt, sind fett gedruckt.

Baud Rate	1 MHz		%Error	1.8432 MHz		%Error	2 MHz		%Error	2.4576 MHz		%Error
2400	UBRR=	25	**0.2**	UBRR=	47	**0.0**	UBRR=	51	**0.2**	UBRR=	63	**0.0**
4800	UBRR=	12	**0.2**	UBRR=	23	**0.0**	UBRR=	25	**0.2**	UBRR=	31	**0.0**
9600	UBRR=	6	7.5	UBRR=	11	**0.0**	UBRR=	12	**0.2**	UBRR=	15	**0.0**
14400	UBRR=	3	7.8	UBRR=	7	**0.0**	UBRR=	8	3.7	UBRR=	10	3.1
19200	UBRR=	2	7.8	UBRR=	5	**0.0**	UBRR=	6	7.5	UBRR=	7	**0.0**
28800	UBRR=	1	7.8	UBRR=	3	**0.0**	UBRR=	3	7.8	UBRR=	4	6.3
38400	UBRR=	1	22.9	UBRR=	2	**0.0**	UBRR=	2	7.8	UBRR=	3	**0.0**
57600	UBRR=	0	7.8	UBRR=	1	**0.0**	UBRR=	1	7.8	UBRR=	2	12.5
76800	UBRR=	0	22.9	UBRR=	1	33.3	UBRR=	1	22.9	UBRR=	1	**0.0**
115200	UBRR=	0	84.3	UBRR=	0	**0.0**	UBRR=	0	7.8	UBRR=	0	25.0

Baud Rate	3.2768 MHz		%Error	3.6864 MHz		%Error	4 MHz		%Error	4.608 MHz		%Error
2400	UBRR=	84	**0.4**	UBRR=	95	**0.0**	UBRR=	103	**0.2**	UBRR=	119	**0.0**
4800	UBRR=	42	**0.8**	UBRR=	47	**0.0**	UBRR=	51	**0.2**	UBRR=	59	**0.0**
9600	UBRR=	20	**1.6**	UBRR=	23	**0.0**	UBRR=	25	**0.2**	UBRR=	29	**0.0**
14400	UBRR=	13	**1.6**	UBRR=	15	**0.0**	UBRR=	16	2.1	UBRR=	19	**0.0**
19200	UBRR=	10	3.1	UBRR=	11	**0.0**	UBRR=	12	**0.2**	UBRR=	14	**0.0**
28800	UBRR=	6	**1.6**	UBRR=	7	**0.0**	UBRR=	8	3.7	UBRR=	9	**0.0**
38400	UBRR=	4	6.3	UBRR=	5	**0.0**	UBRR=	6	7.5	UBRR=	7	6.7
57600	UBRR=	3	12.5	UBRR=	3	**0.0**	UBRR=	3	7.8	UBRR=	4	**0.0**
76800	UBRR=	2	12.5	UBRR=	2	**0.0**	UBRR=	2	7.8	UBRR=	3	6.7
115200	UBRR=	1	12.5	UBRR=	1	**0.0**	UBRR=	1	7.8	UBRR=	2	20.0

Baud Rate	7.3728 MHz		%Error	8 MHz		%Error	9.216 MHz		%Error	11.059 MHz		%Error
2400	UBRR=	191	**0.0**	UBRR=	207	**0.2**	UBRR=	239	**0.0**	UBRR=	287	-
4800	UBRR=	95	**0.0**	UBRR=	103	**0.2**	UBRR=	119	**0.0**	UBRR=	143	**0.0**
9600	UBRR=	47	**0.0**	UBRR=	51	**0.2**	UBRR=	59	**0.0**	UBRR=	71	**0.0**
14400	UBRR=	31	**0.0**	UBRR=	34	**0.8**	UBRR=	39	**0.0**	UBRR=	47	**0.0**
19200	UBRR=	23	**0.0**	UBRR=	25	**0.2**	UBRR=	29	**0.0**	UBRR=	35	**0.0**
28800	UBRR=	15	**0.0**	UBRR=	16	2.1	UBRR=	19	**0.0**	UBRR=	23	**0.0**
38400	UBRR=	11	**0.0**	UBRR=	12	**0.2**	UBRR=	14	**0.0**	UBRR=	17	**0.0**
57600	UBRR=	7	**0.0**	UBRR=	8	3.7	UBRR=	9	**0.0**	UBRR=	11	**0.0**
76800	UBRR=	5	**0.0**	UBRR=	6	7.5	UBRR=	7	6.7	UBRR=	8	**0.0**
115200	UBRR=	3	**0.0**	UBRR=	3	7.8	UBRR=	4	**0.0**	UBRR=	5	**0.0**

3.4 Code-Beispiele zum Gebrauch des UART

3.4.1 UART_Initialisierung

```
#define BAUD 2400
#define MYUBRR F_CPU/16/BAUD-1
void main( void )
{
:.
USART_Init ( MYUBRR );
:.
}
void USART_Init( unsigned int ubrr)
{
/* Set baud rate */
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 2ststop bit */
UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

3.4.2 Daten senden

```
void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) )
;
/* Put data into buffer, sends the data */
UDR = data;
}
```

3.4.3 Daten empfangen

```
unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !(UCSRA & (1<<RXC)) )
;
/* Get and return received data from buffer */
return UDR;
}
```

4 Serielle Datenübertragung über den I²C-Bus

Die immer größer werdende Komplexität von integrierten Schaltungen führt unter anderem auch zu einer dramatischen Zunahme der Anschluss-Pins, die die Größe des Bausteins bestimmen. Durch die Vielzahl der Verbindungsleitungen sind entsprechend viele Lötstellen und beim Übergang auf eine andere Platine entsprechend viele Steckerpins erforderlich. Dies erhöht die Kosten und die möglichen Störstellen innerhalb des Systems und begrenzt die Anzahl der auf der Platine platzierbaren Bauelemente.
Die hohe Geschwindigkeit, mit der Daten über einen parallelen Bus übertragen werden können, ist aber häufig gar nicht erforderlich. Einige Beispiele für niedrige Datenraten:

- Bedienung der Tastatur: Auch bei schneller Eingabe von maximal fünf bis zehn Zeichen pro Sekunde.
- Ansteuerung eines Displays: bedingt durch die Trägheit des menschlichen Auges sind mehr als zwei bis drei Anzeigenänderungen pro Sekunde nicht sinnvoll.
- Bedienung einer Mechanik: Da mechanische Systeme meist sehr träge sind, kann hier fast immer mit geringen Datenübertragungsraten operiert werden.

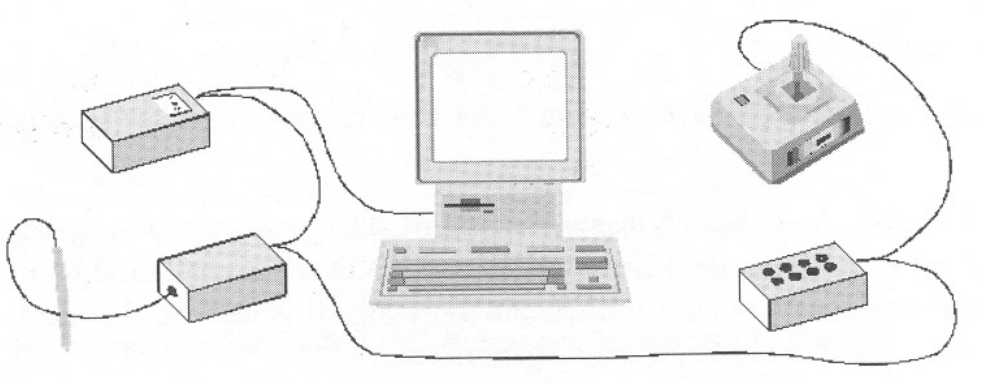


Abbildung 12: Steuerung von Komponenten über den I²C-Bus

In allen hier aufgeführten Fällen könnten die Daten also auch seriell, ein Bit nach dem anderen übertragen werden. Die geringere Übertragungsgeschwindigkeit (die beim SPI-Interface immerhin mehr als 1Million Bit pro Sekunde und beim I²C-Bus bis zu 100.000 Bit pro Sekunde beträgt) tritt hier nicht negativ in Erscheinung.
Aus diesem Grund hat die Fa. Philips den I²C-Bus (Inter-IC-Bus) entwickelt:

Vorteile des I²C-Busses

- ICs können an den Bus angeschlossen oder von ihm entfernt werden, ohne die anderen an den Bus angeschlossenen ICs zu beeinflussen
- Herkömmliche Bus-Interface–Schaltkreise wie Leitungstreiber und – empfänger sind unnötig, da das I²C-Bus-Interface bereits auf dem Chip integriert ist.
- Ein und derselbe I²C-Bus-Schaltkreis kann häufig in vielen anderen unterschiedlichen Anwendungen verwendet werden.

- Die Entwicklungszeit für neue Schaltungen verkürzt sich deutlich, da der Entwickler mit der Funktion des I²C-Busses schnell vertraut wird und I²C-Schaltkreise einfach einzusetzen sind.

I²C-Spezifikationen

- Serieller Betrieb mit nur zwei Leitungen (siehe Abbildung 13), dadurch geringe Anschlusszahl, minimaler Verdrahtungsaufwand.
- Der Wirkungsbereich reicht bis zu drei Metern.
- Single-Master oder Multi-Master Betrieb ist möglich.

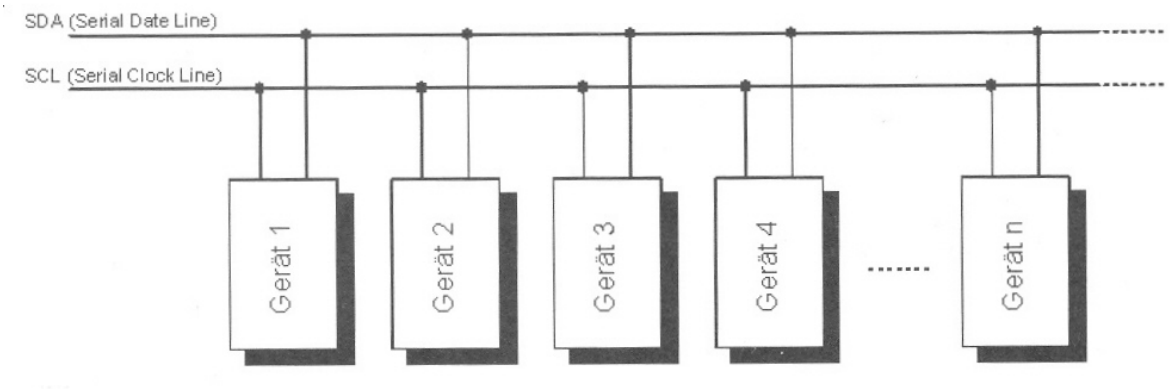


Abbildung 13: Anschluss einzelner Module an den I²C-Bus

4.1 I²C-Funktionsbeschreibung

Die Abbildung 14 zeigt den Aufbau einer I²C-Bus-Verbindung. Der I²C-Bus besteht aus nur zwei Leitungen. Das sind die Leitungen SCL (Serial Clock) und SDA (Serial Data). Serial Clock ist gleich dem Taktsignal. Serial Data ist gleich der Datenleitung. Beide Signale werden im Folgenden beschreiben.

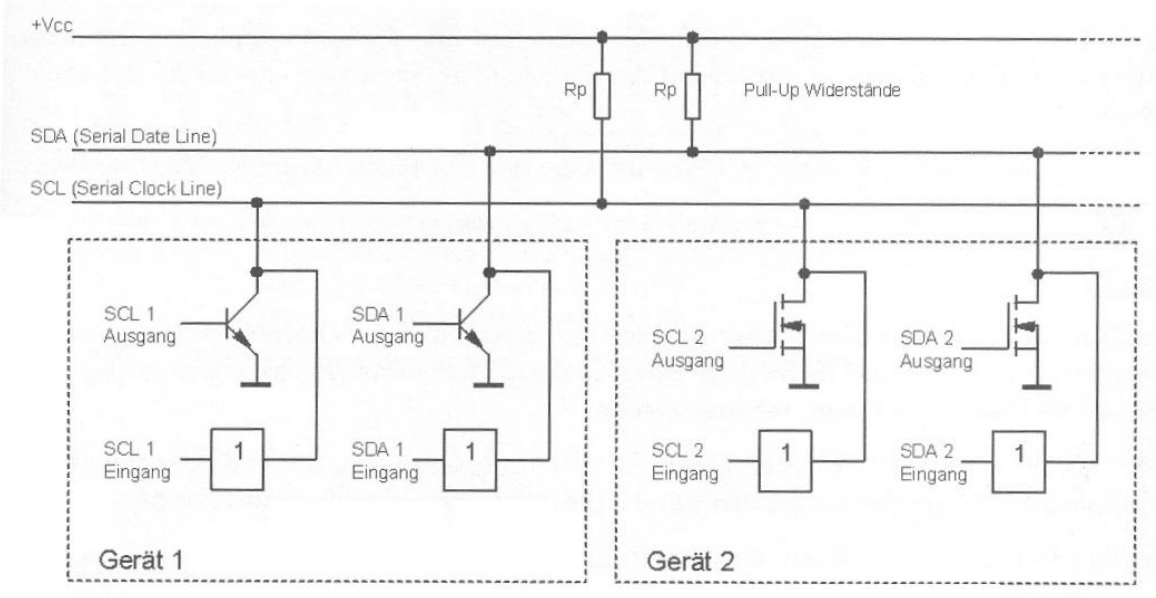


Abbildung 14: Aufbau der I²C-ICs und Belegung des I²C-Busses

4.1.1 SCL (Serial Clock)

Solange keine Daten über den Bus übertragen werden (der Bus frei ist), liegt SCL auf HIGH-Pegel.
Die Taktleitung SCL ist in Open-Drain- bzw. Open-Collector-Struktur ausgelegt (siehe Abbildung 14). Dadurch ist es möglich, die Leitung als Wired-AND zu beschalten, d.h. die Taktleitung ist LOW, wenn mindestens eine Einheit ein LOW-Signal generiert, die Taktleitung ist HIGH, wenn alle Einheiten ein HIGH-Signal generieren.
Die Taktleitung ist über einen Pull-Up-Widerstand mit der Versorgungsspannung verbunden. Bei einer Versorgungsspannung von 5 V sind folgende Pegel definiert:

- LOW-Signal: von 0 bis 1,5 V und
- HIGH-Signal über 3 V.

I²C-Bausteine haben im LOW-Zustand eine maximale Ausgangsspannung von 0,4 V und sind dabei in der Lage, einen Strom von 3 mA aufzunehmen.

Die Taktgeschwindigkeit wird von dem jeweiligen Master vorgegeben. Im vorliegenden Versuch übernimmt der AVR die Rolle des Masters. Die Taktfrequenz f_{SCL} wird vom Systemtakt abgeleitet:

$$f_{SCL} = \frac{\Phi}{16 + 2 \cdot (TWBR) \cdot 2^{TWPS}} \quad , \qquad (1)$$

wobei:
f_{SCL} : Taktfrequenz
Φ : Systemtakt,
TWBR : Inhalt des 8 Bit-TWI-Bitraten-Registers TWBR (0...255) und
TWPS : Inhalt der Bits TWPS1 und TWPS0 aus dem TWI Statusregister (0...3).

4.1.2 SDA (Serial Data)

Solange keine Daten über den Bus übertragen werden, hat die Leitung SDA HIGH-Potential. Während der Datenübertragung überträgt SDA die seriellen Daten. Sie sind in der HIGH-Phase des Taktes gültig und können ihren Zustand nur in der LOW-Phase ändern.
Jede an den Bus angeschlossene Einheit kann während der Datenübertragung entweder Receiver (Empfänger) oder Transmitter (Sender) sein.
Die Datenleitung SDA ist wie SCL in Open-Drain- bzw. Open-Collector-Struktur ausgelegt (siehe Abbildung 14). Dadurch ist es möglich, die Leitung als Wired-AND zu beschalten, d.h. die Taktleitung ist LOW, wenn mindestens eine Einheit ein LOW-Signal generiert, die Taktleitung ist HIGH, wenn alle Einheiten ein HIGH-Signal generieren.
Die Datenleitung wird über einen Pull-Up-Widerstand mit der Versorgungsspannung verbunden. Die Spannungspegel sind wie bei SCL beschrieben definiert.

4.2 Betriebsarten der an den I²C-Bus angeschlossenen Einheiten

4.2.1 Receiver oder Transmitter

Ein an den I²C-Bus angeschlossener Baustein kann als Receiver oder als Transmitter wirken:

- **Receiver**
  Die Einheit empfängt die auf der Datenleitung SDA übertragenen und mit dem Taktsignal auf der SCL-Leitung synchronisierten Daten.

- **Transmitter**
  Die Einheit sendet ihre mit dem Taktsignal auf der SCL-Leitung synchronisierten Daten über die Datenleitung SDA aus.

Ob eine Einheit Transmitter oder Receiver ist, hängt nur von ihren speziellen Eigenschaften ab. Ein ROM-Speicher zum Beispiel kann nur Transmitter sein, während ein LED-Treiber wohl immer Receiver ist. Im Gegensatz dazu kann ein RAM-Speicher oder ein Mikroprozessor sowohl Receiver als auch Transmitter sein.
Die Betriebsweise einer Einheit kann während der laufenden Datenübertragung auch geändert werden. So ist es denkbar, dass z. B. ein Mikroprozessor als Transmitter zunächst Daten an einen RAM-Speicher (Receiver) überträgt. Direkt anschließend liest derselbe Prozessor als Receiver aus dem RAM-Speicher der dann als Transmitter wirkt, Daten aus.

4.2.2 Master oder Slave

Jede an den Bus angeschlossene Einheit kann während der Datenübertragung entweder Master oder Slave sein:

- **Master**
  Einheit, die die Datenübertragung initiiert hat. Insbesondere belegt der Master den Bus, indem er ein Startsignal auf der Leitung SCL generiert und die Kommunikation mit dem Slave aufnimmt. Solange ein Master auf dem Bus tätig ist, kann kein anderer Master aktiv werden.
  Das Taktsignal auf der SCL-Leitung kann nur von einem Master generiert werden.

- **Slave**
  Einheit, die von einem Master angesprochen (adressiert) und zur Datenübertragung aufgefordert wird. Zumindest theoretisch ist es denkbar, dass mehrere Slaves gleichzeitig mit denselben Daten vom Master versorgt werden.

Der Master wird in den meisten Fällen ein Mikroprozessor sein, der entweder mit einer speziellen I²C-Hardware ausgestattet ist oder die Bussteuerung per Software durchführt.

4.2.3 Unterscheidungen

Je nachdem, in welcher Weise der Datentransport über den I²C-Bus erfolgen soll, unterscheidet man zwischen:

- Master-Transmitter (Master, der Daten an den Slave (Receiver) überträgt);
- Master-Receiver (Master, der Daten vom Slave (Transmitter) empfängt);
- Slave-Transmitter (Slave, der Daten an den Master(Receiver)überträgt);
- Slave-Receiver (Slave, der Daten vom Master (Transmitter)empfängt).

4.3 Elektrische Eigenschaften

Die Übertragungsrate wird durch das Taktsignal auf der Leitung SCL bestimmt. Als Obergrenze für den Takt ist eine Frequenz von 100 kHz festgelegt, eine untere Grenze gibt es nicht. Jede Einheit auf dem Bus sollte in der Lage sein, der Taktfrequenz zu folgen. Das Tastverhältnis ist zwar mit 1:1 festgelegt, stellt aber keine zwingende Forderung dar. Maßgebend für eine fehlerfreie Datenübertragung ist vielmehr die Flankensteilheit; die Abfallzeit sowohl für die Daten als auch den Takt darf 300 ns nicht überschreiten. Und hier liegt auch die eigentliche Begrenzung für die Auslegung des Busses: Die kapazitive Belastung der Leitungen darf nicht größer als 400 pF sein. Da die kapazitive Belastung sowohl von der Anzahl der an den Bus angeschlossenen Einheiten als auch von der Leitungslänge abhängt, muss dieser Wert bei dem Entwurf der Hardware berücksichtigt werden (Laut Spezifikation gibt es auch noch einen LOW-Speed- und einen HIGH-Speed-Modus des I²C-Busses, hier wollen wir aber nur den Standard-Modus betrachten).

4.4 Busprotokoll

Kommunizieren mehrere Einheiten über einen Bus miteinander, so ist ein Protokoll erforderlich, das die Datenübertragung auf dem Bus beschreibt und das zu keinem Zeitpunkt eine fehlerhafte Interpretation des Buszustandes zulässt. Bei einem Multi-Master-Bus, wie ihn der I²C-Bus darstellt, muss weiterhin festgelegt werden, wann welcher Master das Recht hat, den Bus zu belegen. Die Bedingungen sind im Busprotokoll geregelt.

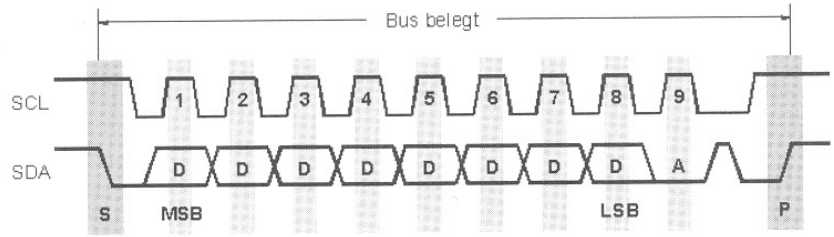


Abbildung 15: Protokoll einer Übertragung eines Bytes über den I²C-Bus

4.4.1 Belegung des Busses

Der Bus-Betrieb und damit seine Belegung wird durch die START-Bedingung eingeleitet.

START-Bedingung
Die beiden Leitungen SCL und SDA befinden sich im HIGH- (Ruhe-) Zustand. Der Master, der eine Datenübertragung beginnen möchte, führt auf der SDA-Leitung einen HIGH → LOW-Wechsel durch.

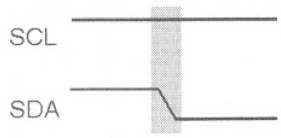


Abbildung 16: START-Bedingung

Nach der START-Bedingung ist der Bus durch den Master, der die START-Bedingung erzeugt hat, belegt und damit für alle anderen Master blockiert.
Die START-Bedingung ist ein eindeutiger Zustand auf dem Bus, weil Pegelwechsel auf der Datenleitung SDA sonst grundsätzlich nur dann zugelassen wird, wenn sich die Taktleitung SCL im LOW-Zustand befindet.
Alle Einheiten auf dem Bus müssen die START-Bedingung erkennen und werden auf Empfang (Slave-Receiver) geschaltet. Die START-Bedingung wird insbesondere auch von einem anderen möglichen Master beachtet, der seinerseits die Absicht hatte, den Bus zu belegen; er stellt seine Busanforderung sofort zurück. Die Einheit, die die START-Bedingung generiert hat, ist momentan der Bus-Master, alle anderen Einheiten sind potentielle Slaves. Der Master ist nun für das Taktsignal verantwortlich und zunächst Transmitter.

4.4.2 Datenübertragung

Nach Erzeugung der START-Bedingung (S) nimmt der Master die Datenübertragung (D) auf. Er bringt die Taktleitung in den LOW-Zustand und ist jetzt in der Lage, die Datenleitung mit der geforderten Bit-Information zu belegen (HIGH oder LOW-Pegel auf SDA). Anschließend wird die Taktleitung wieder in den HIGH-Zustand gebracht (Siehe Abbildung 17).

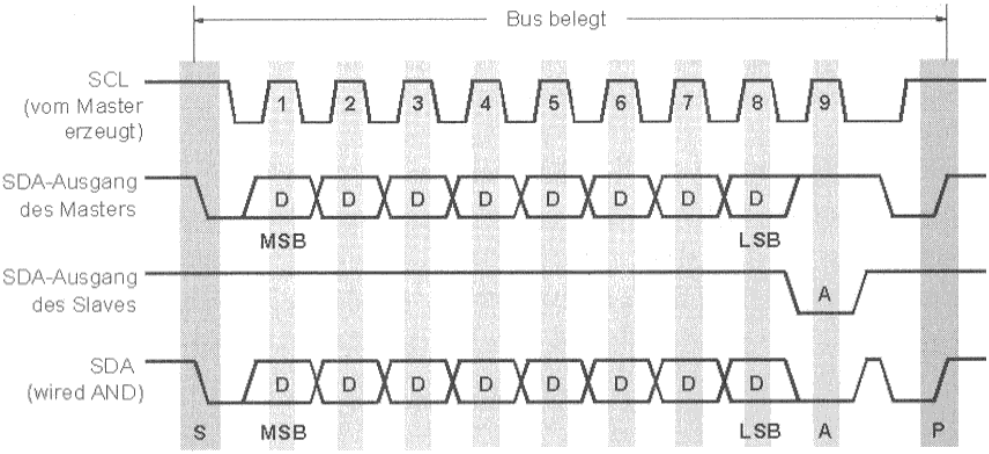


Abbildung 17: Zeitlicher Verlauf einer Datenübertragung beim I²C-Bus

Ein Wechsel der SDA-Leitung darf nur in der LOW-Phase des Taktes SCL erfolgen. SDA muss während der HIGH-Phase des Taktes SCL stabil sein.

Die Datenübertragung wird grundsätzlich byteweise ausgeführt. Das erste übertragene Bit ist das MSB (Most Significant Bit) des Datenbytes. Nach der Ausgabe eines kompletten Datenbytes, die aus acht Taktzyklen besteht, folgt ein Acknowledge-Bit (A).

4.4.3 Acknowledge-Bit

Das Acknowledge-Bit ist eine Reaktion des Receivers auf das empfangene Datenbyte. Es stellt eine Bestätigung (Acknowledge) für die korrekte Datenübertragung dar und ist für den Transmitter ein Zeichen dafür, dass der Receiver physikalisch vorhanden ist und das Byte empfangen hat. Gleichzeitig kann es als Synchronisationssignal betrachtet werden.
Das Acknowledge-Bit wird grundsätzlich vom empfangenen Baustein generiert. Empfängt ein Master von einem Slave mehrere Datenbytes, so quittiert er jedes einzelne Byte mit einem Acknowledge-Bit mit Ausnahme des letzten. Dieses negative Acknowledge zeigt dem Slave an, dass die Datenübertragung beendet ist, und als nächstes eine STOPP- oder eine erneute START-Bedingung folgt.
Zur Übertragung des Acknowledge-Bits generiert der Master einen zusätzlichen Taktimpuls auf SCL, wie in Abbildung 17 zu sehen ist.
Der Receiver gibt ein positives Acknowledge, indem er die Datenleitung SDA auf LOW-zieht, ein negatives Acknowledge ist dementsprechend der HIGH-Zustand der Leitung SDA.
Somit sind für die Übertragung eines Bytes grundsätzlich neun Taktzyklen erforderlich, so dass bei einer Taktfrequenz von 100 kHz eine Übertragungsrate von maximal 11111 Byte / s möglich ist.

Nach der Ausgabe eines Datenbytes und dem Erhalt des Acknowledge-Bits kann die Datenübertragung sofort fortgesetzt werden. Reagiert der Receiver auf die Übertragung eines Bytes mit einem negativen Acknowledge, so sollte der Master die Datenübertragung beenden, indem er den Bus wieder freigibt.

4.4.4 Freigabe des Busses

Der Bus wird durch den Master nach einer kompletten Datenübertragung, die aus einer beliebigen Anzahl von Bytes bestehen kann, wieder freigegeben. Die Freigabe des Busses erfolgt durch eine STOPP-Bedingung (S).

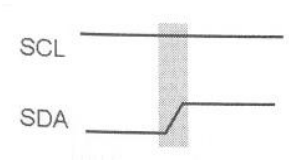


Abbildung 18: STOPP-Bedingung

STOPP-Bedingung
Ein Pegelwechsel von LOW → HIGH auf der SDA-Leitung während die SCL-Leitung HIGH-Pegel führt, wird als STOPP-Bedingung gewertet. Nach Erzeugung der STOPP-Bedingung ist der Bus frei und damit wieder für alle anderen Einheiten zugänglich.
Die STOPP-Bedingung stellt ebenfalls einen eindeutigen Zustand auf dem Bus dar. Alle Einheiten erkennen die STOPP-Bedingung und bereiten sich auf eine neue START-Bedingung vor. Sollte ein Master wegen der zwischenzeitlichen Busbelegung seine Busanforderung zurückgestellt haben, kann er nun erneut ver-

suchen, eine START-Bedingung zu erzeugen und damit den Bus für sich zu gewinnen.
Eine komplette Datenübertragung über den I^2C-Bus besteht grundsätzlich aus einer START-Bedingung, einem oder mehreren Daten-Bytes (jeweils gefolgt von einem Acknowledge-Bit) und einer STOPP-Bedingung.

4.4.5 Repeated Start

Nach der Übertragung eines kompletten Bytes einschließlich des dazugehörigen Acknowledge–Bits kann der Master eine wiederholte Start-Bedingung ausgeben ohne den Bus freizugeben. Da der Bus weiterhin belegt bleibt, ist der Master in der Lage, die Datenübertragung fortzusetzen.

4.4.6 Adressierung der Slaves

Die Auswahl, mit welchem Slave der Master Daten austauschen möchte, wird durch das erste Byte getroffen, das immer als Slave-Adresse definiert ist.

Slave-Adresse
Das erste Byte einer Datenfolge repräsentiert die Adresse eines Slaves. Sie ist eindeutig einem bestimmten Baustein auf dem Bus zugeordnet und hat eine Länge von sieben Bit (Bit 1 bis Bit 7). Theoretisch könnten damit bis zu 128 Slaves adressiert werden. Per Definition haben jedoch einige Slave-Adressen besondere Bedeutung. In der Realität wird ein System aus nicht mehr als 20 Slaves bestehen. Die Slave-Adresse wird aus einem festen und einem variablen Teil gebildet (siehe Abbildung 19).

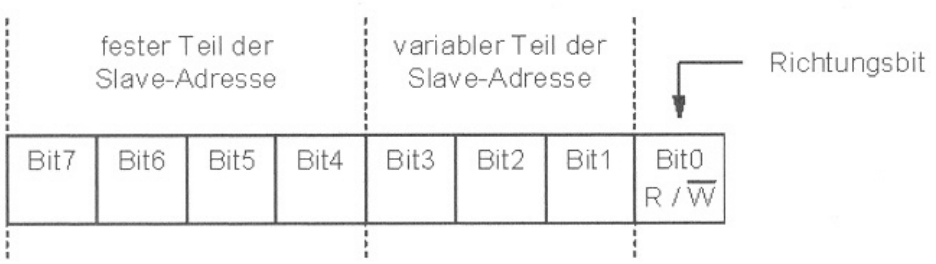


Abbildung 19: Format einer I^2C-Adresse

Fester Teil der Slave-Adresse
Der feste Teil der Adresse spezifiziert eine bestimmte Bausteingruppe und wird vom Hersteller des Chips festgelegt. Seine Länge resultiert aus praktischen Erfahrungen und beträgt in den meisten Fällen 4 Bit (siehe Abbildung 19). Er wird umso kürzer sein, je mehr gleichartige Bausteine in einer Schaltung vorhanden sein könnten. Der feste Teil der Adresse ist in der integrierten Schaltung intern verdrahtet und kann vom Anwender nicht geändert werden.

Variabler Teil der Slave-Adresse
Der variable Teil der Slave-Adresse (siehe Abbildung 19) dient zur Selektierung eines bestimmten Bausteines aus einer Gruppe von gleichartigen Chips, die alle den gleichen festen Anteil der Slave-Adresse besitzen. Dadurch ist sichergestellt, dass mehrere ICs desselben Typs an den Bus angeschlossen werden können. Der variable Teil wird meist durch externe Beschaltung (zusätzliche Pins) vom

Anwender festgelegt.

Datenrichtungsbit
Mit den ersten sieben Bits (Bit 7 ... Bit 1) des 1. Bytes nach der Startbedingung - also der SlaveAdresse - ist der gewünschte Slave eindeutig identifiziert. Das bei der Byte-Übertragung verbleibende achte Bit (Bit 0, siehe Abbildung 19) gibt die Datenrichtung vor. Es kennzeichnet, ob der Slave Daten empfangen oder senden soll.
Hat das Datenrichtungsbit den Wert „1“ (Read), so ist der Master in der Betriebsart Receiver, der Slave in der Betriebsart Transmitter. Hat das Datenrichtungsbit den Wert „0“ (Write), so wird der Master als Transmitter, der Slave als Receiver betrieben.
Die mit der Slave-Adresse gewählte Betriebsart gilt für eine beliebig lange Folge von Bytes. Da alleine der Master bestimmt, wie viele Bytes übertragen werden, verfügen Slaves meist über ein Auto-Inkrement bei der internen Adressierung. So ist es zum Beispiel möglich, einem als Slave-Receiver adressierten RAM beliebig viele Daten zu senden, die dieses in aufeinanderfolgenden Adressen abspeichert. Genauso gut kann ein RAM kontinuierlich ausgelesen werden, indem es der Master als Slave-Transmitter adressiert und die Taktung entsprechend der Anzahl zu lesender Bytes vornimmt.
Die Slave-Adresse wird vom Slave ebenso durch ein Acknowledge-Bit bestätigt, wie ein Datenbyte. Erhält der Master nach der Adressierung ein negatives Acknowledge, so kann er daraus schließen, dass der Slave entweder gar nicht vorhanden ist oder zur Zeit keine Kommunikation zu ihm aufgenommen werden kann - zum Beispiel, weil der Slave gerade mit der Behandlung zeitkritischer Aufgaben beschäftigt ist.

4.5 Beteiligte Register des ATmega16

4.5.1 TWBR – TWI Bitratenregister

Zusammen mit den Bits TWPS1 und TWPS0 aus dem TWI-Statusregister bestimmt dieses Register die Bitrate, falls der AVR als Master agiert. Wenn der AVR über die I²C-Bus als Slave angesprochen wird, ist der Wert dieses Registers ohne Bedeutung. Die genaue Berechnung der Bitrate ist im Abschnitt 4.1.1 erklärt.

Bit	7	6	5	4	3	2	1	0	
	TWBR7	**TWBR6**	**TWBR5**	**TWBR4**	**TWBR3**	**TWBR2**	**TWBR1**	**TWBR0**	**TWBR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

4.5.2 TWCR – TWI Kontrollregister

Mit dem Kontrollregister wird die Schnittstelle konfiguriert und aktiviert bzw. deaktiviert.

Bit	7	6	5	4	3	2	1	0	
	TWINT	**TWEA**	**TWSTA**	**TWSTO**	**TWWC**	**TWEN**	**-**	**TWIE**	**TWCR**
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TWINT (TWI Interrupt Flag)
Dieses Bit wird durch die Hardware gesetzt, wenn die TWI ihre aktuelle Aufgabe beendet hat und auf eine Antwort von der Hardware wartet. Falls das I-Bit im Statusregister und das Bit TWIE im Register TWCR gesetzt sind, wird die MCU zum TWI Interrupt-Vektor verzweigen. Während das Bit TWINT gesetzt ist bleibt die Leitung SCL auf Low. Das Bit TWINT-Flag muss durch die Software zurückgesetzt werden, in dem das Bit mit einer logischen Eins überschrieben wird. Anders als bei vielen anderen Interrupt-Flags wird dieses Bit nicht automatisch nach Abarbeitung seiner ISR zurückgesetzt. Ausserdem ist darauf zu achten, dass durch Löschen des TWINT-Flags ein Arbeitsvorgang der Schnittstelle gestartet wird. Deshalb sollten alle beteiligten Register mit den notwendigen Werten belegt sein, wenn das TWINT-Flag auf null gesetzt wird.

TWEA (TWI Enable Acknowledge Bit)
Das Bit TWEA steuert das Aussenden eines Acknowledge-Impulses. Setzt man TWEA auf eins, wird unter folgendes Voraussetzungen ein Acknowledge-Impuls ausgesendet:

- Der ATmega16 hat seine eigene Slave-Adresse empfangen.
- Ein allgemeiner Aufruf wurde empfangen, wobei das Bit TWGCE aus dem Register TWAR gesetzt ist.
- Im Empfangsbetrieb wurde ein Datenbyte empfangen.

Durch das Setzen von TWEA auf null, kann der Baustein virtuell vom Bus getrennt werden.

TWSTA (TWI Start Condition Bit)
Der Anwender setzt dieses Bit auf eins, wenn er als Master agieren möchte. Die TWI-Hardware überprüft, ob der Bus frei ist. Wenn das der Fall ist, erzeugt er eine Start-Bedingung. Falls der Bus jedoch nicht freigegeben ist, wartet der Baustein mit dem Aussenden der Start-Bedingung bis er die Freigabe des Busses detektiert. TWSTA muss durch den Anwender zurückgesetzt werden, nachdem die Start-Bedingung ausgesendet wurde.

TWSTO (TWI Stop Condition Bit)
Wird dieses Bit während der Baustein als Master agiert auf Eins gesetzt, wird auf dem Bus eine Stopp-Bedingung erzeugt. Wenn die Stopp-Bedingung auf dem Bus erzeugt wurde, wird das Bit TWSTO automatisch zurückgesetzt. Arbeitet der Baustein als Slave, kann durch das Setzen von TWSTO das Bussystem nach einem Fehler wieder in einen definierten Zustand versetzt werden.

TWWC (TWI Write Collision Flag)
Das Flag wird gesetzt, wenn versucht wird, in das Datenregister zu schreiben, während das Bit TWINT auf Null ist. Das Bit wird durch das Schreiben ins Datenregister gelöscht, während das Bit TWINT auf Eins gesetzt ist.

TWEN (TWI Enable Bit)
TWEN ermöglicht die Verwendung von TWI-Operationen und aktiviert die TWI-Schnittstelle. Wenn TWEN auf Eins gesetzt wird, übernimmt die Schnittstelle die Kontrolle über die I/O-Anschlüsse SCL und SDA. Wenn dieses Bit auf Null gesetzt wird, stellt die Schnittstelle sofort ihren Betrieb ein.

TWIE (TWI Interrupt Enable)
Wenn dieses Bit den Wert Eins hat und das I-Bit im Statusregister gesetzt ist, wird die TWI-Interrupt-Anforderung freigegeben, solange das Bit TWINT den Wert Eins hat.

4.5.3 TWSR – TWI Statusregister

Das Kontrollregister bietet neben der Möglichkeit die Bitrate einzustellen die Möglichkeit Kontrollstrukturen einzufügen.

Bit	7	6	5	4	3	2	1	0	
	TWS7	**TWS6**	**TWS5**	**TWS4**	**TWS3**	**-**	**TWPS1**	**TWPS0**	**TWSR**
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

TWS 7:3 (TWI Status 7:3)
Diese fünf Bits zeigen den Busstatus und den Zustand der TWI–Logik an. Die Erklärung der verschiedenen Zustandscodes würde an dieser Stelle zu weit führen. Die Auflistung der Zustandscodes für den Master-Betrieb kann im Datenblatt nachgelesen werden.

TWPS 1:0 (TWI Prescaler Bits 1:0)
Diese Bits können gelesen und geschrieben werden. Zusammen mit dem Inhalt des 8 Bit-TWI-Bitraten-Registers TWBR (0...255) wird die Bitrate eingestellt. Zur Erinnerung ist die Gleichung noch ein Mal dargestellt:

$$f_{SCL} = \frac{\Phi}{16 + 2 \cdot (TWBR) \cdot 2^{TWPS}}$$

, wobei:
f_{SCL} : Taktfrequenz
Φ : Systemtakt,
TWBR : Inhalt des 8 Bit-TWI-Bitraten-Registers TWBR (0...255)
TWPS : Inhalt der Bits TWPS1 und TWPS0 aus dem TWI Statusregister (0...3)

4.5.4 TWDR – TWI Datenregister

Beim Sendebetrieb beinhaltet das Register das als nächstes zu sendende Byte. Im Empfangsbetrieb enthält das Register das zuletzt empfangene Byte. Sobald das TWI–Interrupt-Flag gesetzt ist, kann der Inhalt des Registers nicht überschrieben werden. Der Inhalt des Registers bleibt erhalten, solange das Bit TWINT gesetzt ist.

Bit	7	6	5	4	3	2	1	0	
	TWD7	**TWD6**	**TWD5**	**TWD4**	**TWD3**	**TWD2**	**TWD1**	**TWD0**	**TWDR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

4.5.5 TWAR – TWI Slave Address Register

Falls der AVR im Slave-Betrieb arbeiten soll, muss ihm eine 7-Bit-Adresse zugeteilt werden. Sie wird in den oberen sieben Bits des Registers TWAR gespeichert.
Das gesetzte Bit 0 des Registers ermöglicht den Empfang von allgemeinen Anfragen, die über den Bus am AVR eintreffen. Wir das Bit 0 auf null gesetzt, so werden derartige Anfragen ignoriert.

Bit	7	6	5	4	3	2	1	0	
	TWA6	**TWA5**	**TWA4**	**TWA3**	**TWA2**	**TWA1**	**TWA0**	**TWGCE**	**TWAR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

4.6 Code-Beispiel für die Übertragung eines Bytes

Das folgende Beispiel zeigt Schritt für Schritt, wie bei der Übertragung eines Bytes vom Master (AVR) zu einem Slave vorgegangen werden muss:

Schritt	Beispiel	Kommentar
1	`TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)`	Aussenden der START-Bedingung
2	`while (!(TWCR & (1<<TWINT))) ;`	Abwarten, bis das Flag TWINT gesetzt ist. Das zeigt an, dass eine Start-Bedingung versendet wurde.
3a	`if ((TWSR & 0xF8) != START) ERROR();`	Überprüfung des Status-Registers TWSR. Ausmaskieren der Prescaler-Bits. Falls START ungleich, gehe nach ERROR.

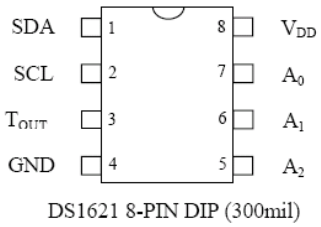
3b	`TWDR = SLA_W;` `TWCR = (1<<TWINT)|(1<<TWEN);`	Lade SLA+W in das TWDR Register. Lösche Bit TWINT in TWCR um die Start-Bedingung zu senden.
4	`while (!(TWCR & (1<<TWINT)))` `;`	Warten auf das TWINT-Flag. Dadurch wird angezeigt, dass SLA+W gesendet und ACK/NACK empfangen wurde.
5a	`if ((TWSR & 0xF8) != MT_SLA_ACK) ERROR();`	Überprüfe den Wert des TWI Status Registers. Ausmaskieren der Prescaler-Bits. Gehe nach ERROR, falls MT_SLA_ACK ungleich.
5b	`TWDR = DATA;` `TWCR = (1<<TWINT) | (1<<TWEN);`	Lade DATA in das TWDR Register. Lösche Bit TWINT im Register TWCR, um das Aussenden der Daten zu starten.
6	`while (!(TWCR & (1<<TWINT)))` `;`	Warten auf das TWINT-Flag. Das zeigt, dass DATA gesendet wurde und ACK/NACK empfangen wurde.
7a	`if ((TWSR & 0xF8) !=MT_DATA_ACK) ERROR();`	Überprüfen des TWI Status Registers. Ausmaskieren der Prescaler-Bits. Gehe nach ERROR, falls MT_SLA_ACK ungleich.
7b	`TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);`	Senden der STOPP-Bedingung

4.7 Temperatursensor mit I²C-Bus

Der Temperatursensor DS1621 bietet eine Temperaturmessung mit einer Auflösung von 9 Bit. Ausgegeben wird die Temperatur des Bauelementes. Der Sensor ist kalibriert und ermöglicht ein direktes, digitales Auslesen der gemessenen Temperatur, ohne dass ein A/D-Wandler benötigt wird. Der Sensor beinhaltet ein digitales Thermometer und verfügt darüber hinaus über Funktionen wie Thermostat, programmierbarem, digitalen Thermostat oder Speicher.
Die Adresse wird gemäß Kapitel 4.4.6 gebildet. Das obere Nibble der Adresse ist 0x9.

4.7.1 Pinbelegung

Die folgende Abbildung 20 zeigt die Pin-Belegung des DS1621 der Fa. Dallas Semiconductor.



PIN DESCRIPTION

SDA	- 2-Wire Serial Data Input/Output
SCL	- 2-Wire Serial Clock
GND	- Ground
T_{OUT}	- Thermostat Output Signal
A0	- Chip Address Input
A1	- Chip Address Input
A2	- Chip Address Input
V_{DD}	- Power Supply Voltage

Abbildung 20: Pin-Belegung des DS1621

An die Anschlüsse V_{DD} und GND wird eine Betriebsspannung von 2,7 V bis 5,5 V gelegt. Die Eingangspins A2 bis A0 bilden den variablen Teil der Adresse und ergeben sich durch die Beschaltung. Ansonsten werden für die I²C- Bus-Anschlüsse SCL und SDA benötigt.

4.7.2 Verwendete Befehle

Um Befehle zum DS1621 zu schreiben, muss der Master die Adresse mit auf Null gesetztem Richtungsbit ausgeben. Nachdem der Master das Acknowledge-Bit empfangen hat, kann der Master den Befehlscode ausgeben. Nach dem Empfang des Code wird der DS1621 ein Acknowledge ausgeben und anschließend kann der Master seine Daten schicken.
Wenn der DS1621 ausgelesen werden soll, muss der Master zunächst auch den entsprechenden Befehlscode ausgeben. Darauf muss ein Restart ausgegeben werden und der Master muss erneut die Adresse des DS1621 mit auf Eins gesetztem Richtungsbit ausgeben.

Umsetzung Starten Set: 0xEE
Dieses Kommando startet eine Temperatur-Umsetzung. Nach der Temperatur-Umsetzung kann das Ergebnis ausgelesen werden. Die Abbildung 21 zeigt den zeitlichen Verlauf der Signale.

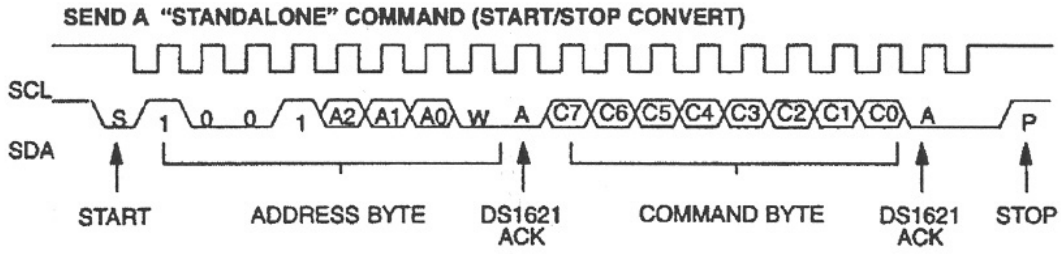


Abbildung 21: Timing-Diagramm bei „Umsetzung starten“

Temperatur lesen Set: 0xAA
Dieses Kommando gibt das Ergebnis der Temperatur-Umsetzung in dem in Kapitel 4.7.3 beschriebenen Format aus. Es werden zwei Bytes gesendet. Die Abbildung 22 zeigt den zeitlichen Verlauf der Signale.

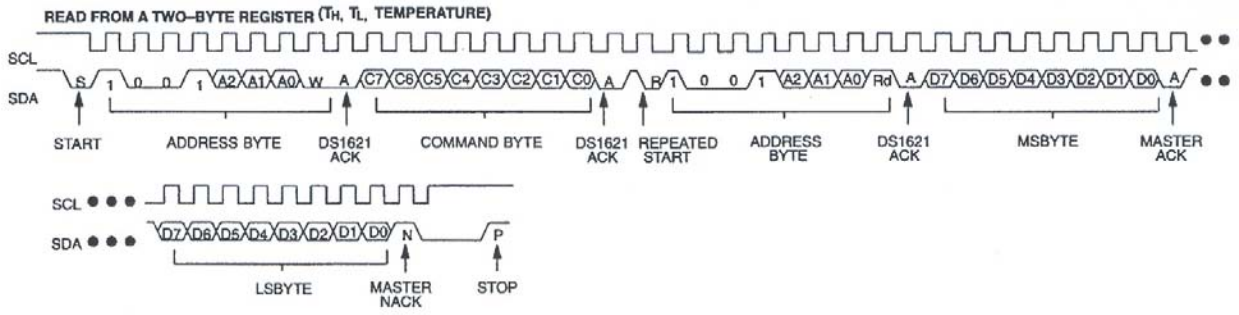


Abbildung 22: Timing-Diagramm bei „Temperatur lesen“

4.7.3 Format des Temperaturwertes

Der Temperaturwert wird in zwei Bytes übertragen. Nach der Umwandlung der unteren sieben Bits des HIGH-Bytes in den Dezimalwert erhält man direkt den Temperaturwert in °C. Das MSB gibt das Vorzeichen des Temperaturwertes an. Bei positiven Werten ist das MSB gleich Null und bei negativen Werten gleich Eins. Das LOW-Byte zeigt die Nachkommastelle mit einer Auflösung von 0,5°C an.

Die folgende Tabelle zeigt diesen Zusammenhang.

Registerinhalt, binär	Registerinhalt, HEX	Temperaturwert in °C
01111101 00000000	7D 00	+125
00011001 00000000	19 00	+25
00000000 10000000	00 80	+0,5
00000000 00000000	00 00	0
11111111 10000000	FF 80	-0,5
11100111 00000000	E7 00	-25
11001001 00000000	C9 00	-55

5 Versuchsdurchführung

5.1 Senden über die UART-Schnittstelle

Geben Sie ein einzelnes Zeichen über die UART-Schnittstelle aus.

5.1.1

Schreiben Sie ein Programm, um ein einzelnes Zeichen über die UART-Schnittstelle auszugeben. Die Baudrate soll 2400 kHz betragen. Die Datenwortlänge soll 10 Bit betragen.

5.1.2

Untersuchen Sie das Ausgabesignal mit dem Oszilloskop indem Sie das Sendesignal am Port PD1 abgreifen. Zeichnen Sie den zeitlichen Verlauf des Signals in das folgende Diagramm ein und beschriften es wie in Abbildung 7.

5.1.3

Speichern Sie ihr Projekt.

5.2 Senden und Empfangen über die UART-Schnittstelle

Schreiben Sie ein Programm, um eine Zeichenkette über die UART-Schnittstelle per Funktionsaufruf auszugeben. Die Funktion soll durch Tastendruck aufgerufen werde.
Bei einem Datenempfang soll der Interrupt RX Complete ausgelöst werden und das Zeichen auf dem Display ausgegeben werden.

5.2.1

Erzeugen Sie zunächst das Programm und laden Sie es in den Programmspeicher des Mikrocontrollers.

5.2.2

Verbinden Sie das STK500 über die Schnittstelle RS232 Spare mit dem PC, in dem Sie die in Abbildung 23 und Abbildung 24 gezeigten Verbindungen vornehmen.

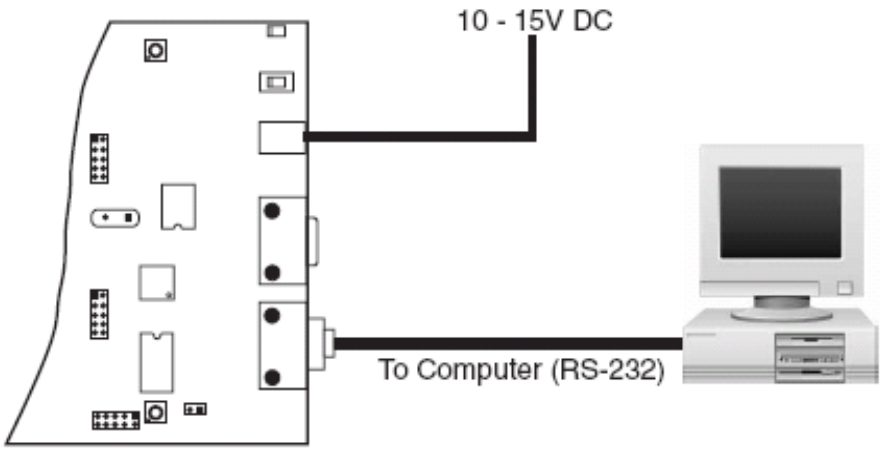


Abbildung 23: Verbindung der RS232 zum PC

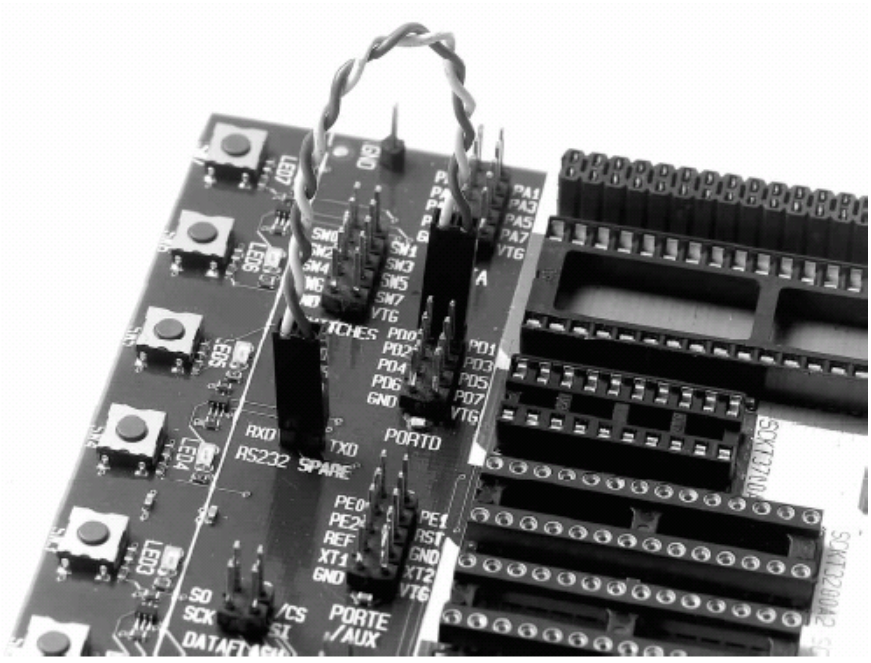


Abbildung 24: Verbindung des UART an die RS232

Starten Sie anschließend auf dem PC das Programm Hyper-Terminal aus der Gruppe Zubehör und dann Kommunikation. Richten Sie eine neue Verbindung über die Schnittstelle COM1 mit den in Abbildung 25 gezeigten Eigenschaften ein.

Abbildung 25: Einstellungen im Programm Hyper-Terminal

Anschließend sollten die Eingaben über die PC-Tastatur auf dem Display ausgegeben werden und bei Aufruf Ihres Unterprogramms zur Zeichenausgabe sollte der Text auf dem PC erscheinen.

5.2.3

Speichern Sie Ihr Programm.

5.3 Temperaturmessung über den I²C-Bus

Schreiben Sie ein Programm, das den Temperaturwert aus dem DS1621 in regelmäßigen Abständen ausliest und das HIGH-Byte auf dem LCD an Port A ausgibt. Der Port PC1 liefert das Signal SDA und PC0 das Signal SCL. Dazu gehen Sie wie folgt vor.

5.3.1

Schließen Sie den DS1621 an Port C an. Die Schreib-Adresse des DS1621 ist auf 0x9E eingestellt. Zum Lesen benutzt man die Adresse 0x9F.

5.3.2

Editieren Sie zunächst das im Anhang dargestellte Programm und laden Sie es in den Programmspeicher des Mikrocontrollers.

5.3.3

Oszillografieren Sie die Bus-Leitungen SDA und SDL. Vergleichen Sie den zeitlichen Verlauf mit den Darstellungen in Abbildung 21 und Abbildung 22.

5.3.4

Übertragen Sie die Temperatur mit der in Kap. 5.2 erstellten Funktion über die RS232-Schnittstelle zum PC und geben Sie die Temperatur wie in Kap. 5.2.2 auf dem PC aus.

6 Anhang

```
// ** ATMEGA16 C-Language File **

// When you program this file use device ATMEGA16

#include <C:/avr/include/avr/io.h>
#include <C:/avr/include/util/delay.h>
#include <C:/avr/include/util/lcd.h>

#define WRITE_ADDRESS 0x9E
#define READ_ADDRESS 0x9F
#define START_CONVERSION 0xEE
#define READ_TEMPERATURE 0xAA


//    Author     : Thomas Massel
//    Company    : FH-Gelsenkirchen
//    Date       : 29.05.2009
//    Comment    : Lösung des Praktikumsversuches µCT
//                 Versuch 4; Aufg. 5.3.2

void ERROR (void );
void TWI_Init(void);
void TWI_Send (void);
void TWI_Start(void);
void TWI_Stopp(void);
void TWI_SendAddr(unsigned char address);
void TWI_SendComm(unsigned char command);
unsigned char TWI_Receive();
void TWI_Rec(void);


// Begin Of Function Main
int main(void)
{
      unsigned char Temperature=0;
      char Buffer[10];
      PORTA=0xFF;                         // Init of Port A
      DDRA=0xFF;
      PORTC=0xFF;                         // Init of Port C
      DDRC=0xFF;
      lcd_init();                         // Init of LCD
      lcd_clear();                        // Erase LCD
      while (1)
      {
            TWI_Init();
            TWI_Send();
            Temperature=TWI_Receive();
            itoa(Temperature, Buffer, 10 );// Convert HEX-Value to String
            lcd_clear();
            lcd_string(&Buffer);          // Display String on LCD
            _delay_ms(2000);              // Time Delay
      }
      return 0;
}
// End of Function main

// Begin of Function TWI_Init
void TWI_Init(void)
{
      TWCR = 0b00000100;                  // Activate TWI
      TWSR = 0;                           // Reset Status and Prescaler
      TWBR = 72;                          // Bit Rate Register 6.25kHz
      _delay_ms(300);
}
// End of Function TWI_Init
```

```
// Begin of Function TWI_send
void TWI_Send(void)
{
      TWI_Start();
      TWI_SendAddr(WRITE_ADDRESS);
      TWI_SendComm(START_CONVERSION);
      TWI_Stopp();
      _delay_ms(5);
}
// End of Function TWI_Send

// Begin of Function TWI_Receive
unsigned char TWI_Receive(void)
{
      unsigned char temp=0;
      TWI_Start();
      TWI_SendAddr(WRITE_ADDRESS);
      TWI_SendComm(READ_TEMPERATURE);
      TWI_Start();
      TWI_SendAddr(READ_ADDRESS);
      TWI_Rec();
      temp= TWDR;
      TWI_Rec();
      TWI_Stopp();
      return temp;
}
// End of Function TWI_Send


//Begin Of Function TWI_Start
void TWI_Start(void)
{
      TWCR=(1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)))
            {
            }
}
//End Of Function TWI_Start

//Begin Of Function TWI_Stopp
void TWI_Stopp(void)
{
            TWCR=(1<<TWINT) |(1<<TWEN) |(1<<TWSTO);
}
//End Of Function TWI_Stopp

//Begin Of Function TWI_SendComm
void TWI_SendComm(unsigned char command)
{
      TWDR=command;
      TWCR=(1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)))
            {
            }
}
//End Of Function TWI_SendComm

//Begin Of Function TWI_SendAddr
void TWI_SendAddr(unsigned char address)
{
      TWDR=address;
      TWCR = (1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)))
            {
            }
}
//End Of Function TWI_SendAddr
```

```
//Begin Of Function TWI_Rec
void TWI_Rec (void)
{
      TWCR=(1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)))
            {
            }
}
//End Of Function TWI_Rec
```

7 Literatur

1) AVR-RISC Mikrocontroller
Wolfgang Trampert
Franzis` Verlag, 2000

2) AVR-Mikrocontroller Praxis
Safinaz Volpe
Elektor Verlag, 2001, 2. Auflage

3) Mikrocomputer-Technik, Vorlesungsskript
Prof. Dr. Lothar Howah
Physikalische Technik, FH Gelsenkirchen, 2004

4) Programmieren in C
Brian W. Kernighan, Dennis M. Ritchie
Hanser Verlag München Wien, 1990

5) Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie
Günter Schmitt
Oldenbourg Wissenschaftsverlag, 2006, 2. Auflage

Außerdem gibt es zahlreiche Links im Internet mit Datenblättern, Anwendungen usw.:

1) http://www.avr-forum.com

2) http://www.avrfreaks.net

3) http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf

4) http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial