Die serielle synchrone Datenübertragung

Bei der seriellen Datenübertragung ist allerdings noch ein wichtiger Punkt zu beachten. Der Datensender sendet nun ein Byte nach dem anderen seriell bitweise zum Datenempfänger aus. Woher weiß der Empfänger nun aber, wann jeweils ein Bit anfängt und wann es aufhört, das heißt, wodurch erkennt der Empfänger, wann er ein Bit korrekt einlesen darf und wann noch nicht (weil noch das vorhergehende Bit auf der Leitung liegt)?

Bei der parallelen Datenübertragung war der Einlesezeitpunkt ganz klar durch das Steuersignal definiert: Bei der fallenden Flanke dieses Signals kann der Empfänger die neuen Daten einlesen. Eine solche Steuerung des Datentransfers fehlt bisher bei der seriellen Datenübertragung.

Deshalb hat man noch eine vierte Leitung eingeführt, auf der ein so genanntes *Taktsignal* zwischen Sender und Empfänger übertragen wird. Dieses Taktsignal steuert mit seinen Flanken die Gültigkeit der einzelnen gesendeten Bits, *Abb. 11.3.1.5.*

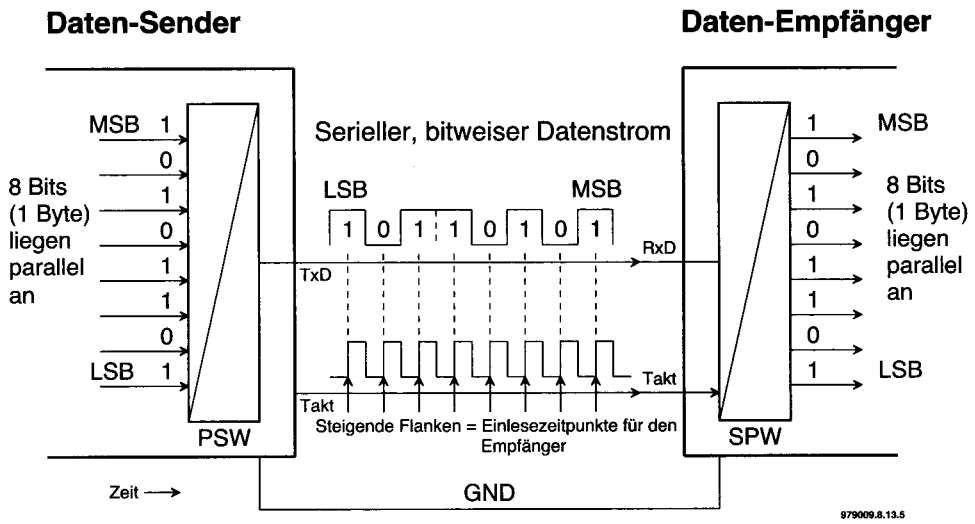


Abb. 11.3.1.5: Die serielle synchrone Datenübertragung

Der Datensender steuert das Taktsignal nun z.B. so, dass bei der steigenden Flanke ein gültiges Datenbit auf der Leitung anliegt. Der Empfänger weiß jetzt: „Bei jeder steigenden Flanke des Taktes kann ich den Zustand auf der Leitung als gültigen Pegelzustand für ein Bit interpretieren und dieses Bit in meinen SPW einlesen."

Durch dieses Taktsignal, auch Synchronisationssignal genannt, erreicht man somit eine *Synchronisierung (einen Gleichlauf)* des Datentransfers zwischen Sender und Empfänger, das heißt, der Datenaustausch zwischen Sender und Empfänger erfolgt zeitlich synchron (gleichlaufend). Die Datenübertragungsgeschwindigkeit wird daher durch die Frequenz des Taktsignals vorgegeben; so kann man Sender und Empfänger optimal aneinander anpassen.

6. Da die Synchronisation zwischen Sender und Empfänger also nicht während der gesamten Datenübertragung aufrechterhalten werden kann, sondern bei jedem zu übertragenen Byte eine erneute Synchronisation erfolgen muss, spricht man daher von einer seriellen *asynchronen* Datenübertragung. Da nach jeder Bitgruppe eine Pause erfolgen muss, nennt man dieses Datentransferverfahren auch „Asynchrones Start-/Stopp-Verfahren".
7. Die bekanntesten Beispiele für serielle asynchrone Datenübertragungsschnittstellen sind die COM-Schnittstellen des PCs: so verläuft z.B. die Kommunikation zwischen dem Mikrocontroller-System und dem Terminal über solch eine asynchrone Verbindung.

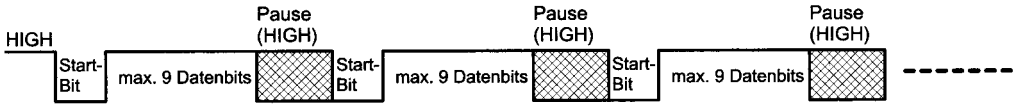


Abb. 11.3.1.9: Der konkrete Ablauf der seriellen asynchronen Datenübertragung

Die Experten-Ecke: „Die serielle Datenübertragung"

Die Darstellung der seriellen asynchronen Datenübertragung erfolgt hier etwas vereinfacht, da auf die Überabtastung im Sender und Empfänger nicht eingegangen wird. Ebenso fehlen Erläuterungen zur seriellen synchronen Datenübertragung mit Zeichensynchronisierung. Dieses Wissen gehört aber schon in die „Spezialisten-Ecke"; für den „Anfänger" ist die Kenntnis der Grundlagen der am häufigsten verwendeten seriellen asynchronen Datenübertragung in vielen Fällen mehr als ausreichend.

Merke: „Die serielle asynchrone Datenübertragung, II"

Die serielle asynchrone Datenübertragung hat sich heute in vielen Bereichen der Datenkommunikation dominierend durchgesetzt. Für diese Datentransfermethode sind mittlerweile wichtige Zusatzbaugruppen, so genannte Modems, entwickelt worden, die eine störsichere Datenübertragung auch über große und größte Entfernungen ermöglichen.

So können die asynchron übertragenen Datenpakete

- über normale Kupferleitungen (Datenleitungen) bis zu 3.000 m und mehr übertragen werden (z.B. bei der TTY-Übertragung),
- über die normale 230V~-Netzleitung in einem gesamten Gebäude verteilt werden,
- über Funk- und Infrarot-Lichtstrecken ohne Kabel versendet werden,
- über Lichtwellenleiter (LWL) besonders störsicher und besonders weit transferiert werden und schließlich
- über das Telefonnetz weltweit verteilt werden.

Der Empfänger

- tastet also alle 15 ms anstatt alle 20 ms die Datenleitung ab,
- dadurch verschiebt sich das gesamte Zeitraster auf der Empfängerseite,
- die dringend benötigte Synchronisation zwischen Sender und Empfänger geht verloren,
- und nach anfangs noch richtig empfangenen Bits werden auf einmal „falsche" (versetzte) Zustände empfangen, da der Empfänger zu falschen (versetzten) Zeiten die Leitung abtastet.

Im Beispiel aus der *Abb. 11.3.1.8* empfängt der Empfänger die Bits:

Start-Bit, 1, 1, 0, 0, 0, 0, 1, 0, ...

anstatt

Start-Bit, 1, 0, 0, 0, 1, 0, 1, 1, ...

Bereits nach dem zweiten Bit entsteht hier ein Fehler auf Grund des Synchronisationsverlustes.

Wie bereits erwähnt, sind die Unterschiede in den Taktfrequenzen zwischen Sender und Empfänger in der Praxis bei weitem nicht so groß wie hier angenommen, aber sie sind auf jeden Fall vorhanden. Das bedeutet, dass ein Empfangsfehler aufgrund der Zeitrasterverschiebung vielleicht erst nach dem 10., 12. oder 20. übertragenen Bit auftritt. Aber auch das ist natürlich nicht tolerierbar.

Daher muss man bei der seriellen asynchronen Datenübertragung mit wesentlichen Einschränkungen leben.

☛ Merke: „Die serielle asynchrone Datenübertragung, I"

Bei der seriellen asynchronen Datenübertragung sind folgende Grundfestlegungen getroffen worden:

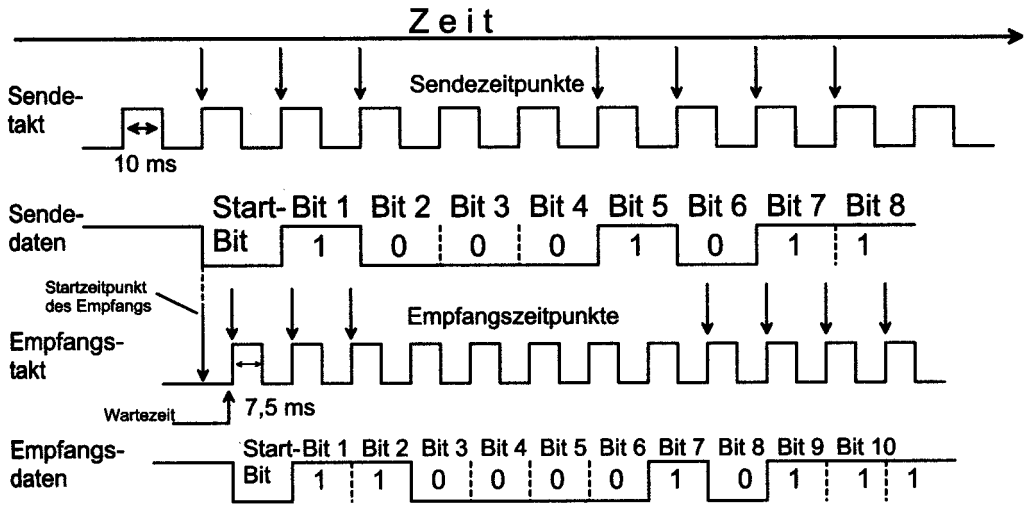
1. Die notwendige Synchronisation zwischen Sender und Empfänger kann maximal (mit „Sicherheitszuschlägen") für 10 Bits aufrechterhalten werden, dann führen die real vorhandenen Frequenztoleranzen langsam (aber sicher) zu einem kritischen Synchronisationsverlust zwischen Sender und Empfänger.
2. Das hat zur Folge, dass man neben dem Start-Bit, das den Empfänger-Baud-Rate-Generator synchronisiert, maximal 9 (Nutz-)Datenbits übertragen kann. Danach muss, durch Aussendung eines neuen Start-Bits, wieder eine neue Synchronisation hergestellt werden, um wieder maximal 9 Bits sicher übertragen zu können.
3. Zwischen den einzelnen übertragenen 9er-Bit-Gruppen muss zur Trennung der einzelnen Gruppen eine notwendige Pause vorhanden sein: Sendedatenleitung fest auf High-Pegel. Dem Empfänger wird es so ermöglicht, sich auf eine neue Synchronisation einzustellen.
4. Die *Abb. 11.3.1.9* zeigt den Ablauf dieser Datenübertragung: Gruppenweises Aussenden von Datenbits, die durch ein Start-Bit synchronisiert und durch eine Pause voneinander getrennt werden.
5. Wenn also z.B. 200 Byte übertragen werden sollen, werden 200 solcher Kombinationen von „Start-Bit, 8 Bit breitem Datenblock, Pause" ausgesendet.

Der Sender arbeitet mit einer Taktfrequenz von 50 Hz, was einer Taktperiode von 20 ms entspricht (2 * 10 ms). Bei jeder steigenden Flanke des Taktsignals sendet er ein Bit aus, hier z.B. die Bitfolge 10001011 ...

Zuerst wird jedoch das Start-Bit gesendet. Dieses wird vom Empfänger erkannt und zur Synchronisation seines eigenen Baud-Rate-Generators benutzt.

Damit der Empfänger die einzelnen Bits auch sicher unterscheiden kann, wartet er eine halbe Taktperiode (10 ms) und liest dann alle 20 ms (bei jeder steigenden Flanke des Taktsignals) den Zustand der Datenleitung (High- oder Low-Pegel) ein.

Durch diese Wartezeit wird erreicht, dass der Empfänger immer in der Mitte eines Bits die Leitung abtastet. Jetzt gelingt es ihm sicher, die gesendete Datenfolge: Start-Bit, 1, 0, 0, 0, 1, 0, 1, 1, ... einzulesen. Somit lässt sich also auch eine serielle Datenübertragung ohne Taktleitung realisieren, wenn hierbei nicht ein *großes Problem* auftreten würde: Man wird es in der Praxis nie erreichen können, dass der Empfänger-Taktgenerator (Baud-Rate-Generator) *100%ig exakt* mit der gleichen Frequenz arbeitet wie der Sender-Taktgenerator. Das ist aber die absolut notwendige Voraussetzung dafür, dass die Datenübertragung einwandfrei funktioniert. Betrachten Sie dazu ein weiteres Beispiel, *Abb. 11.3.1.8.*



Aussenden mit der steigenden Flanke, Empfangen mit der steigenden Flanke

Abb. 11.3.1.8: Der Verlust der Synchronisation zwischen Sender und Empfänger bei der seriellen asynchronen Datenübertragung

Der Sender sendet weiterhin mit einer Taktperiode von 20 ms. Der Empfänger arbeitet aufgrund von Toleranzen, schlechter Einstellung etc. mit einer Taktperiode von 15 ms, und das entspricht einer Taktfrequenz von ca. 67 Hz (so groß sind die Abweichungen in der Praxis zwar nie, aber hiermit lässt sich der „Fehler-Effekt" sehr gut verdeutlichen).

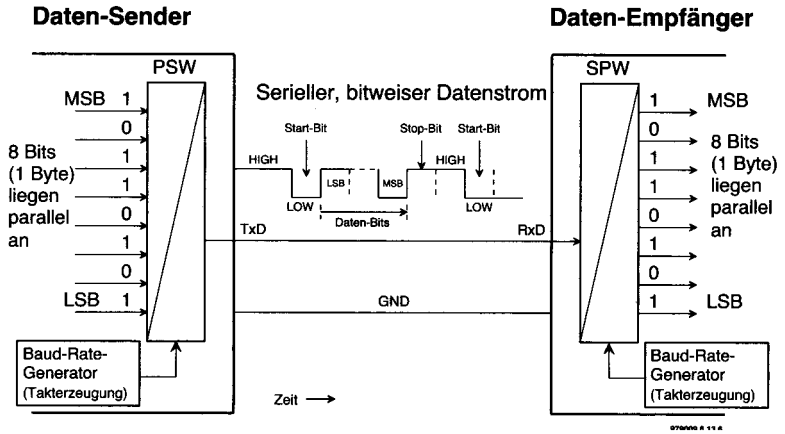
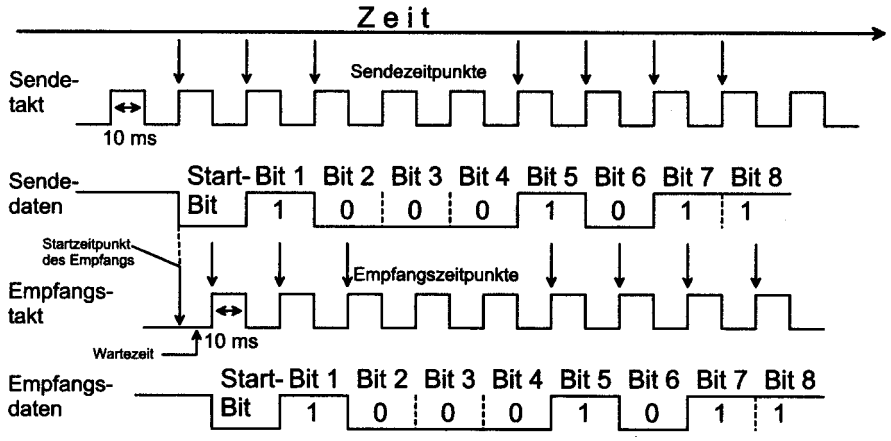


Abb. 11.3.1.6: Die serielle asynchrone Datenübertragung

Für diesen Gleichlauf sorgt nun das zu übertragende Zeichen selber: Im Ruhezustand liegt die Sendedatenleitung immer fest auf High-Pegel. Wenn ein Zeichen gesendet werden soll, erzeugt der Sender zunächst ein so genanntes *Start-Bit*, indem er die Datenleitung auf Low-Pegel zieht. Das erkennt natürlich der Empfänger, der nun weiß, dass ab jetzt einzelne Bits kommen, und zwar im Abstand der von seinem Baud-Rate-Generator erzeugten Taktimpulse, die in ihrer Breite ja ungefähr der Breite der Sendertaktimpulse entsprechen. Der Empfänger besitzt nun ein intern erzeugtes, festes Zeitraster und kann daher die nachfolgenden Bits zu den korrekten Zeitpunkten einlesen.

Ein Beispiel soll dieses verdeutlichen, *Abb. 11.3.1.7.*



Aussenden mit der steigenden Flanke, Empfangen mit der steigenden Flanke

Abb. 11.3.1.7: Die Herstellung der Synchronisation zwischen Sender und Empfänger bei der seriellen asynchronen Datenübertragung

Die Taktleitung selber ist sehr oft bidirektional ausgelegt, das heißt, sie kann von beiden Stationen benutzt werden, um den Datentransfer mit der jeweils anderen Station zu koordinieren, wobei natürlich immer nur eine Station senden und die andere empfangen kann. Eine einzige Taktleitung zwischen beiden Teilnehmern ist also ausreichend.

Merke: „Die serielle synchrone Datenübertragung"

Serielle synchrone Datenübertragung bedeutet, dass neben den seriell zu übertragenen Daten auf einer zusätzlichen 4. Leitung noch serielle Taktinformationen mit übertragen werden, die zeitlich synchron zu den Datenbits verlaufen.

Durch gleichartige Programmierung auf der Sender- und auf der Empfängerseite kann festgelegt werden, wann gültige Daten auf der Datenleitung vorliegen, z.B. entsprechende Programmierung beim Sender und beim Empfänger: „Einlesen eines gültigen Datenbits von der Datenleitung bei der fallenden (steigenden) Flanke des Taktsignals."

Mittels dieser Taktimpulse wird zwischen Sender und Empfänger eine feste Synchronisation (ein fester Gleichlauf) für die gesamte Dauer der Datenübertragung aller Zeichen hergestellt. Daher die Bezeichnung serielle *synchrone* Datenübertragung.

Diese serielle synchrone Datenübertragung ist die schnellste der seriellen Datentransfer-Methoden.

Die einzelnen Bits der zu übertragenden Bytes werden somit als kontinuierlicher serieller Datenstrom *ohne Zwischenpausen* auf die Sendeleitung ausgegeben: Wenn z.B. 200 Bytes übertragen werden sollen, so erscheint auf der Sendeleitung ein zusammenhängender Strom von 200 * 8 Bits = 1.600 Bits.

Die serielle asynchrone Datenübertragung

Für viele Anwendungen ist der Aufwand der seriellen synchronen Datenübertragung noch zu hoch:

- eine zusätzliche vierte Leitung muss installiert werden,
- auf der Senderseite muss ein besonderer Taktgenerator vorhanden sein,
- auf der Empfängerseite muss eine Taktaufbereitungsstufe aufgebaut werden.

Daher hat man als zweite serielle Datenübertragungsmethode die so genannte *serielle asynchrone Datenübertragung* entwickelt, *Abb. 11.3.1.6.*

Sender und Empfänger enthalten hierbei jeweils eine eigene Taktstufe, den *Baud-Rate-Generator*, die unabhängig voneinander arbeiten. Beide Generatoren haben jedoch ungefähr die gleiche Frequenz, da sie quarzgesteuert sind und die benötigten Datenübertragungsquarze mit sehr kleinen Toleranzen hergestellt werden können.

Das grundlegende Problem der seriellen Datenübertragung bleibt zunächst jedoch noch erhalten: Der Empfänger muss wissen, wann ein neues Bit anfängt und wann er es einlesen kann. Es muss also auf jeden Fall eine Synchronisation zwischen Sender und Empfänger hergestellt werden.

Das zu Anfang dieser Lektion vorgestellte Problem:
„Schnelle, sichere und einfache Verteilung von Nutzdaten über größere Entfernungen"
kann daher durch die serielle asynchrone Datenübertragung bestens gelöst werden; daher liegt einer der Schwerpunkte der nachfolgenden Betrachtungen auf dieser Datentransfermethode.

Nach diesen ganzen Betrachtungen über Synchronisation und Verlust derselben braucht der „Programmieranfänger" aber keine bleibenden Frustrationen gegenüber der seriellen Datenübertragung zu entwickeln und zu behalten, denn die Halbleiterindustrie hat eine Menge nützlicher Chips für diese synchronen und asynchronen Datenübertragungsmethoden entwickelt. Diese Bausteine müssen Sie nur noch mit dem µC verbinden, einen Quarz anschließen, und schon haben Sie die komplette Hardware für eine korrekte Datenübertragung. Lediglich einige Software-Programmierungen müssen noch vorgenommen werden, und danach können Sie beliebige Daten austauschen. Bei einigen 8051ern (so z.B. beim 80C537er) haben Sie sogar den Vorteil, dass sich bereits zwei komplette Datenübertragungsbaugruppen ON-Chip befinden, deren Programmierung über entsprechende SFR erfolgt. Bevor wir aber damit beginnen, müssen wir erst noch einige weitere grundlegende Begriffe der seriellen asynchronen Datenübertragung einführen und erklären.

Das UART-Zeichenformat

Die Halbleiterbausteine, die eine serielle Datenübertragung problemlos ermöglichen, haben besondere Namen:

1. **UASRT**-Baustein: **U**niversal **S**ynchronous/**A**synchronous **R**eceiver/**T**ransmitter, also ein Baustein, in dem jeweils ein Empfänger und ein Sender für sowohl die synchrone als auch für die asynchrone serielle Datenübertragung enthalten ist.
2. **UART**-Baustein: **U**niversal **A**synchronous **R**eceiver/**T**ransmitter, also ein Baustein, in dem jeweils ein Empfänger und ein Sender für die asynchrone serielle Datenübertragung enthalten ist.

Sollen nun Nutzdatenbits übertragen werden (meistens in Form von Bytes), so werden die Daten an eine UART übergeben, und diese wandelt die parallelen Daten in einen seriellen Bitdatenstrom um. Das von der UART ausgegebene serielle Zeichen (der serielle Zeichenstrom) hat nun einen besonderen Aufbau, den man *UART-Zeichen (UART-Charakter)* nennt und der genormt ist (DIN 66020), *Abb. 11.3.1.10.*

Nach dem Start-Bit folgen die Datenbits, deren Anzahl meistens programmierbar ist. Es können:

- 5 oder 6 Bits (D0 ... D4 bzw. D0 ... D5) ausgesendet werden. Diese Datenbreite stammt noch aus der Vergangenheit (Fernschreibtechnik) und wird heute kaum noch verwendet.
- 7 Bits (D0 ... D6) ausgesendet werden. Diese Breite wird verwendet, wenn reine ASCII-Zeichen übertragen werden sollen, die ja nur 7 Bit breit sind.
- 8 Bits ausgesendet werden. Das ist die am häufigsten eingestellte Breite, die für den Transfer eines kompletten Bytes benötigt wird.