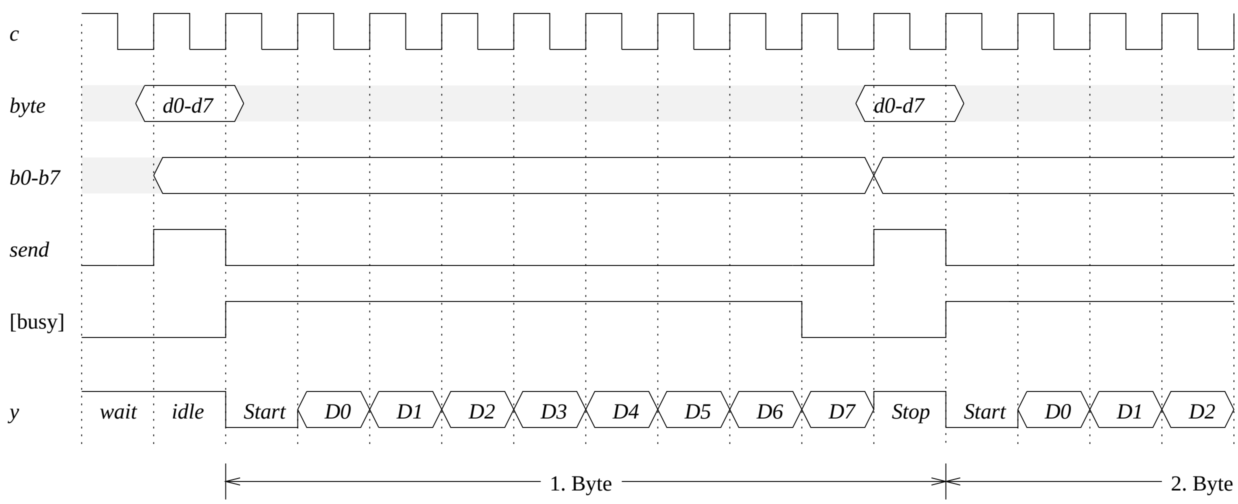
1 0	**Praxistutorium Grundzüge der Informatik B1**	**Aufgabenblatt 6**

Aufgabe 6.1 Serielle Schnittstelle – RS232 Sender

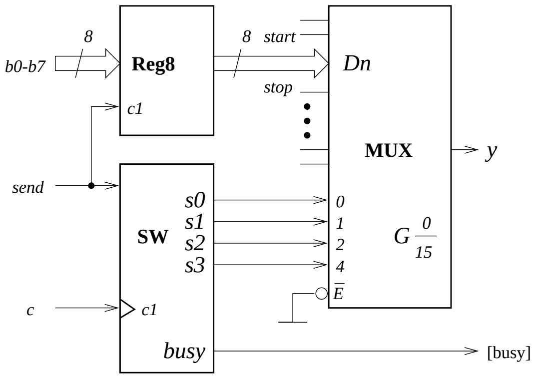
Die Verbindung von (oder zwischen) Rechnern zu verschiedenen Datenendgeräten wie Maus, Modem, Drucker, Terminal, ..., erfolgt vielfach über (bit-)serielle V24/RS232-Schnittstellen. Um ein Byte zu senden, wird je ein Datenbit pro Taktperiode in der Reihenfolge

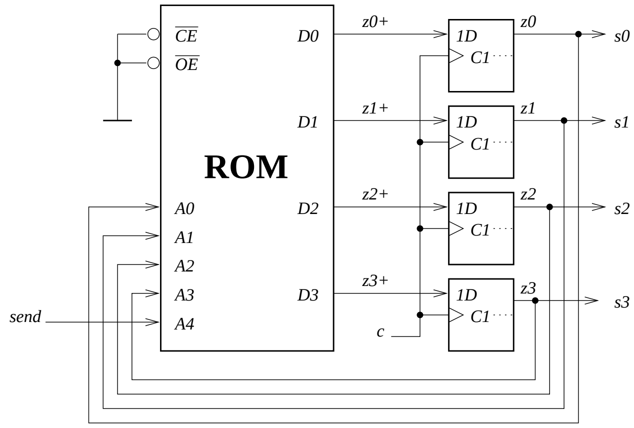
$$
\begin{aligned}
y_{n-1} &= 0 \quad \mathit{Startbit} \\[1em]
y_{n+0} &= d_0 \\
y_{n+1} &= d_1 \\
y_{n+2} &= d_2 \\
y_{n+3} &= d_3 \\
y_{n+4} &= d_4 \\
y_{n+5} &= d_5 \\
y_{n+6} &= d_6 \\
y_{n+7} &= d_7 \\[1em]
y_{n+8} &= 1 \quad \mathit{Stopbit}
\end{aligned}
$$

gesendet. Den Datenbits wird dabei ein Startbit vorangestellt und ein Stopbit wird angehängt. Es werden also zehn Bits für ein Byte gesendet. Wird nichts gesendet, so ist $y = 1$ konstant. Die Taktfrequenz bestimmt die Datenübertragungsrate. Start- und Stopbit dienen einem RS232-Empfänger zur Synchronisation. Das Zeitdiagramm eines RS232-Senders ist in folgender Abbildung skizziert:



Um ein Byte zu senden, wird dieses an die Dateneingänge $b_0 \ldots b_7$ angelegt, anschließend erfolgt durch Setzen von $send := 1$ die Sendeaufforderung. Während des Sendevorgangs ist $busy = 1$, Sendebereitschaft wird durch $busy = 0$ signalisiert. Im *Handshake-Betrieb* erfolgt eine Sendeaufforderung nur wenn $busy = 0$. Der Automat wartet in einem Zustand *LEERLAUF (engl. idle)* auf eine Sendeaufforderung. Ein zusätzlicher Zustand *WARTEN (engl. wait)* stellt sicher, daß *send* erst einmal $send = 0$ werden muß, um eine neue Sendesequenz zu starten.





Die Eingabe der zu übertragenden Bits kann mit zwei `HexSwitch`es geschehen.
Die Auswahl der zu sendenden Bits soll mit Hilfe eines Multiplexers (MUX) erfolgen, indem Bit für Bit in Abhängigkeit von Automatenzuständen ausgewählt wird.

a) Zustandsgraph für Schaltwerk SW
Entwerfen Sie anhand des dargestellten Impulsdiagramms einen geeigneten Zustandsgraphen mit zwölf Zuständen für einen Moore-Automaten.

Die Erzeugung des Signals *busy* durch den Automaten ist optional – d.h. die Realisierung eines geeigneten Schaltnetzes dafür. Die Ansteuerung der Steuereingänge s_0, s_1, s_2, s_3 des Multiplexers kann direkt – d.h. ohne ein zusätzliches Ausgangsschaltnetz – über die Zustände des Automaten erfolgen, wenn eine geeignete Zuordnung zwischen den acht Ausgängen des Pufferspeichers und den Dateneingängen des Multiplexers gewählt wird.

b) Zustandsfolgetabelle
Definieren Sie eine Zustandscodierung und übertragen Sie diese in eine Zustandstabelle und bestimmen Sie aus der Tabelle den Inhalt für einen Mikroprogrammspeicher (ROM).

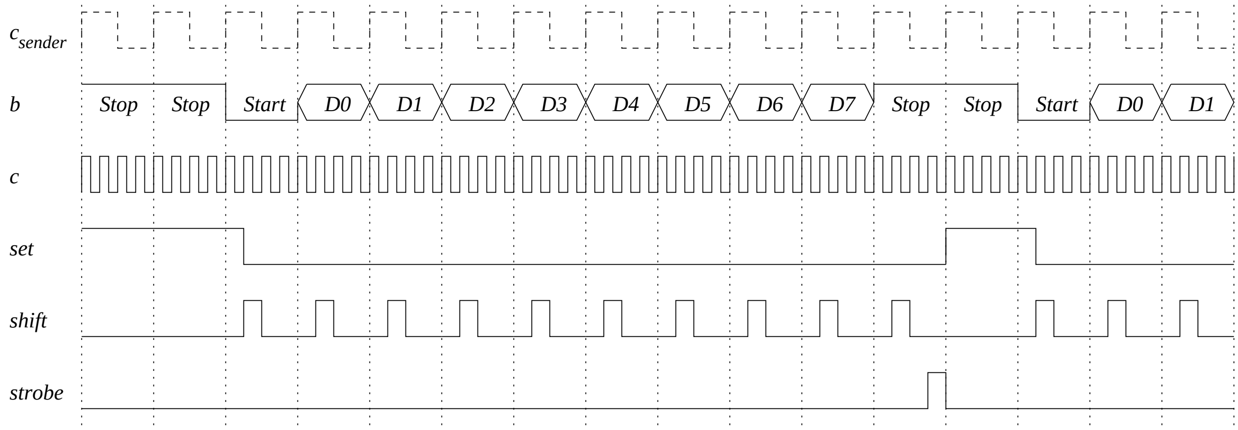
c) Gesamtschaltwerk
Realisieren Sie das mikroprogrammgesteuerte Gesamtschaltwerk.

d) Taktfrequenz
Mit welcher Taktfrequenz muß ein Automat mit dem dargestellten Zeitverhalten getaktet werden, um eine Übertragungsrate von 9600 bit/s zu erreichen?

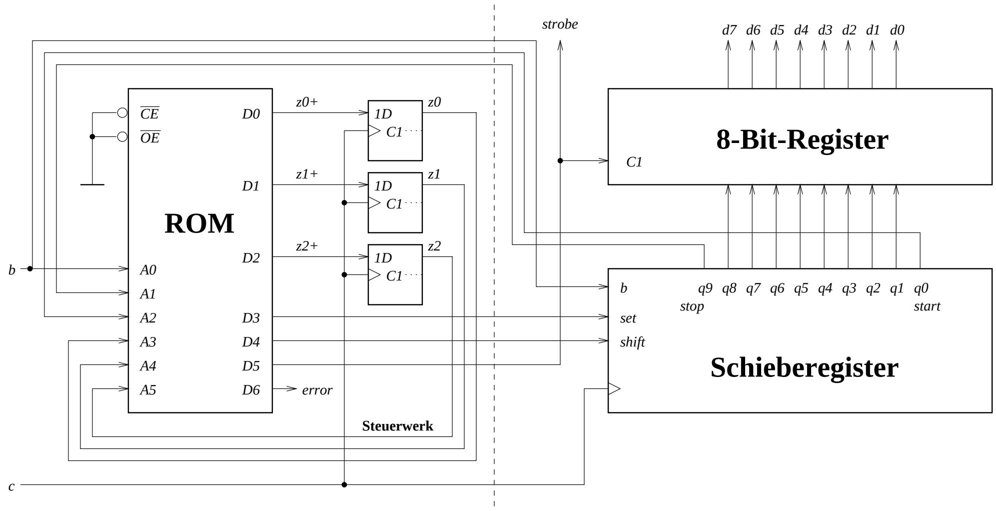
e) Handshake-Zeitverhalten
Wie würde sich das Zeitverhalten ändern, wenn das zum Handshake ausgewertete Signal *busy* nicht schon bei der Auswahl des letzten Datenbits D_7, sondern erst nach vollständig abgeschlossener Sendesequenz zu Null ($busy := 0$) würde?

Aufgabe 6.2 Serielle Schnittstelle – RS232 Empfänger (optional)

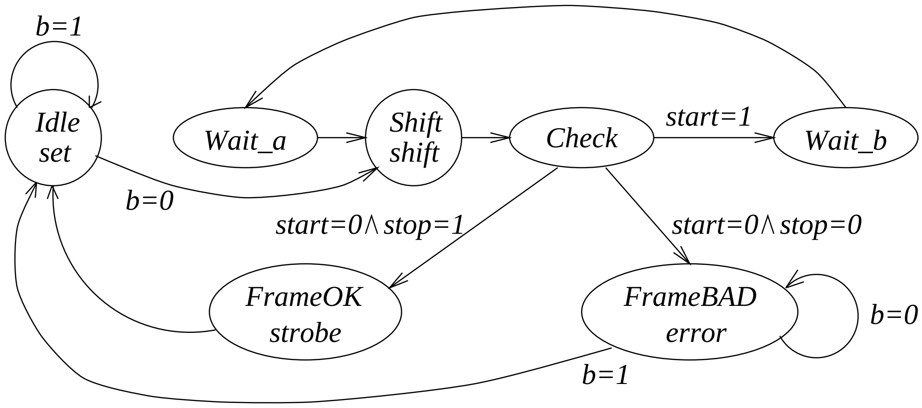
In der folgenden Aufgabe soll ein RS232-Empfänger als synchrones Schaltwerk realisiert werden. Der im Sender verwendete Takt steht dabei im Empfänger nicht mehr direkt zur Verfügung. Der Empfänger muß sich auf den Bit-Strom synchronisieren. Diese Art der bitsequentiellen Datenübertragung wird daher als asynchron bezeichnet. Die Synchronisation kann gemäß dem folgenden Impulsdiagramm erfolgen:



Die Frequenz des Taktes *c* für den Empfänger ist viermal höher als die Taktfrequenz des Senders der vorangegangenen Aufgabe. Der Empfänger wartet auf ein Startbit, d.h. eine fallende Flanke des RS232-Signals. Damit diese Art der Synchronisation eindeutig funktioniert, muß das RS232-Signal mindestens für die Dauer eines vollständigen Bytes den Zustand *eins* haben. Nach dem empfangenen Startbit werden die noch folgenden neun Bits mit der vereinbarten Baudrate abgetastet. Nachdem zehn Bits abgetastet wurden, wird die Gültigkeit des empfangenen Datenstroms überprüft: Das Start-Bit muß *Null* und das Stop-Bit muß *Eins* sein.

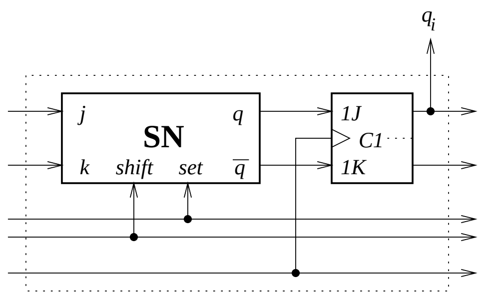


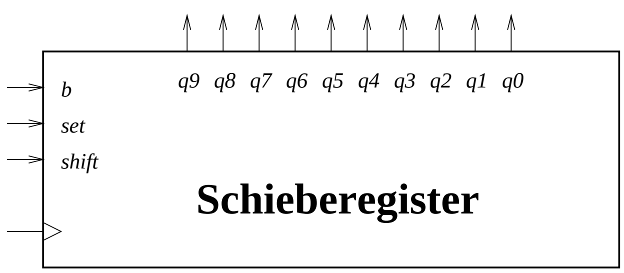
Wurde ein Byte korrekt empfangen, so wird dies durch das Signal *strobe* signalisiert. Ein Fehler bezüglich des Start- und Stop-Bits kann durch ein Signal *error* signalisiert werden. Die hohe Taktfrequenz im Zusammenhang mit den Notwendigkeit, die Zahl der empfangenen Bits zählen zu müssen, erfordert einen Automaten mit vielen (10 Bits a' 4 Takte $\approx$ 40) Zuständen.
Durch die Kopplung von zwei Automaten (Steuerwerk und Schieberegister) läßt sich die Zahl der benötigten – d.h. zu codierenden – Zustände deutlich reduzieren:



Das Schieberegister dient dabei sowohl zur Aufnahme der Datenbits als auch zum Zählen der empfangenen Datenbits. Um die empfangenen Datenbits zählen zu können, müssen alle Schieberegister-Bits während des Wartens auf ein Startbit *eins* gesetzt (*set*) werden. Da das Startbit *null* ist, wird dieses, nachdem zehn Bits in das Schieberegister geschoben wurden, am Ausgang q_0 deutlich.

Das Steuerwerk wartet solange im Zustand *Idle*, bis das Datensignal $b = 0$ ist, d.h. der Automat wartet auf das Startbit. Im Zustand $Shift$ wird ein Bit in das Schieberegister geschoben. Im Zustand $Check$ überprüft der Automat anhand der Schieberegisterbits q_0 und q_9 die Vollständigkeit und Gültigkeit des im Schieberegister gespeicherten Bitstroms. Bei korrekt empfangenem Byte geht der Automat in den Zustand $FrameOK$ und anschließend wieder in den Zustand $Idle$. Bei fehlerhaft empfangenem Byte wartet der Automat im Zustand $FrameBAD$ erst einmal auf ein Stopbit $b = 1$, um anschließend in den Zustand $Idle$ zu gehen. Die Schleife der Zustände $Wait\_a$, $Shift$, $Check$ und $Wait\_b$ wird mit der Baudrate zyklisch durchlaufen; die Frequenz, mit welcher der Zustand $Shift$ auftritt, ist gleich der Baudrate.





Das Schieberegister ist aus 10 gleichen *Schiebezellen* aufgebaut. Über die beiden Eingänge *set* und *shift* wird das Schieberegister synchron vom Steuerwerk kontrolliert.

a) Schiebezelle
Realisieren Sie das Schaltnetz SN der Schiebezelle derart, daß dieses die folgenden Funktionen erfüllt:

shf	set	Funktion		Kommentar
0	0	$q = 0$	$\overline{q} = 0$	*Halten*
0	1	$q = 1$	$\overline{q} = 0$	*Setzen*
1	0	$q = j$	$\overline{q} = k$	*Schieben*
1	1	$q = *$	$\overline{q} = *$	

b) Schieberegister
Realisieren Sie mit Hilfe der zuvor realisierten Schiebezelle das im folgenden benötigte 10-Bit-Schieberegister mit synchronem Set-Eingang.

c) Zustandsübergangstabelle
Erstellen Sie eine Zustandsübergangstabelle für das Steuerwerk gemäß dem dargestellten Graphen. Verwenden Sie dazu folgende Zustandscodierung:

Zustand	z_2	z_1	z_0
Idle	0	0	0
Wait_a	0	0	1
Shift	0	1	0
Check	0	1	1
Wait_b	1	0	0
FrameOK	1	0	1
FrameBAD	1	1	0
XXX	1	1	1

Achten Sie außerdem auf die vorgegebene Beschaltung der benötigten Signale.

d) Mikroprogramm
Erstellen Sie ausgehend von der Zustandsübergangstabelle ein Mikroprogramm, d.h. eine Tabelle mit dem Inhalt des ROMs.