C0 Wiederholung

Die Grundlagen der Assembler-Programmierung aus dem Modul B sollen anhand einiger Programmieraufgaben gefestigt werden.

Kurze Zusammenfassung Modul B

- **Mechanische Schalter** an Eingängen ohne hardwaremäßige Entprellung müssen softwaremäßig mit Hilfe einer Zeitschleife **entprellt** werden (Pull-Up-Widerstände nicht vergessen; mit PIN und nicht mit PORT einlesen!).
- Bei der Benutzung von Unterprogrammen und Interrupt-Behandlungsroutinen (ISR) muss der **Stapel initialisiert** werden! Am Besten die 4 Zeilen zur Initialisierung im Template nie löschen.
- Auf die Arbeitsregister kann üblicherweise im gesamten Programm zugegriffen werden. Man spricht dann von **globalen Variablen**. Variablen die aber nur innerhalb eines Unterprogramms benötigt werden, sollten kein wertvolles Arbeitsregister blockieren. Innerhalb von Unterprogrammen und Interruptroutinen soll man mit **lokalen Variablen** arbeiten. Um Arbeitsregister für lokale Variablen zu befreien wird ihr Inhalt auf dem Stapel mit dem Befehl **push** zwischengespeichert. Vor dem Verlassen des Unterprogramms bzw. der Routine wird der Inhalt dann wieder ins Arbeitsregister zurückgespeichert (**pop**).
- Globale Variablen, die zur Übergabe von Daten an Unterprogramme dienen werden **Parameter** genannt. Am besten benutzt man immer die gleichen Arbeitsregister für die Parameterübergabe (in diesem Kurs das Doppelregister W (**r24**:**r25** bzw. **WL**:**WH**)). Diese Arbeitsregister dürfen dann nicht auf den Stapel gerettet werden. Bei mehreren Parametern (z.B. Tabellen) sollte auf Speicherzellen im SRAM zurückgegriffen werden.
- Interrupts müssen einzeln für die lokale Baugruppe und global (**sei**) freigeschaltet werden. Zusätzlich muss der Interruptvektor mit einem Sprungbefehl zur ISR initialisiert werden.
- In einer Interruptroutine müssen zwingend alle Register die in der ISR verwendet werden, sowie auch das Statusregister **SREG**, auf dem Stapel gesichert (**push**) und von diesem wiederhergestellt (**pop**) werden. Die ISR schließt mit einem **reti** ab!
- Es existieren beim ATmega32A drei externe Interrupts (**INT0**, **INT1**, **INT2**) an **PD2**, **PD3** und **PB2**. Im SF-Register **MCUCR** (bzw. **MCUCSR** für **INT2**) wird festgelegt, ob der Interrupt auf eine positive Flanke, eine negative Flanke oder eine beliebige Flanke bzw. einen Pegel (nur **INT0** und **INT1**) reagiert. Erstaunlicherweise reicht es das **MCUCR**-Register (bzw. **MCUCSR**) zu initialisieren, damit die Interruptflags bei auftretender Flanke bereits gesetzt werden, sogar wenn der externe Interrupt noch gar nicht freigeschaltet ist!
- Wartet der Controller in einer Warteschleife auf ein Ereignis, so bezeichnet man das als **Polling**. Polling hat den Vorteil einer sequentiellen Programmierung, ist aber nicht sehr effektiv, da der Controller blockiert wird. Wenn möglich sollten

deshalb **Interrupts** eingesetzt werden. Mit Interrupts ist Multitasking möglich, da einzelne Baugruppen unabhängig von Prozessor arbeiten können und beim Auftreten eines wichtigen Ereignisses dieses durch eine Unterbrechung mitteilen. Die Programmierung mittels Interrupts erhöht allerdings die Komplexität der Programme.

- Die **serielle Schnittstelle** (EIA-232) wird auch heute noch sehr häufig eingesetzt. Damit eine Kommunikation richtig ablaufen kann, muss das **gleiche Datenformat** (Anzahl der Datenbits, gleiche Parität, gleiche Anzahl von Stoppbits) und die **gleiche Übertragungsgeschwindigkeit** (Baud bzw. Bit/s) verwendet werden. Werden zwei Endgeräte (z.B. PC und Controller) eingesetzt, so müssen die Leitungen gekreuzt werden (Null-Modem-Kabel). Oft werden bei proprietären Lösungen auch Steuerleitungen (z.B. **CTS**, **RTS**) eingesetzt. Diese sind richtig anzuschließen.
  Die Schnittstelle arbeitet mit anderen Pegeln wie der Controller!! Es muss ein Schnittstellen-Wandlerbaustein (z.B. MAX232) eingesetzt werden, da sonst der Controller zerstört werden kann.
- Die meisten ATmega-Controller besitzen eine Hardware-Schnittstelle. Diese wird mit **UART** oder **USART** bezeichnet (*Universal Synchronous/Asynchronous Receiver/Transmitter*). Die USART des ATmega32A wird mit Hilfe von 4 SF-Registern initialisiert. Für die Baudrate wird ein Teilerwert in dem Doppelregister **UBRRH**:**UBRRL** abgelegt wird. Das Datenformat wird im Register **UCSRC** eingestellt. Sender und Empfänger sowie die Interrupts werden in **UCSRB** eingeschaltet. Beim Polling werden die Flags im Statusregister **UCSRA** abgefragt. Das Datenregister heißt **UDR**.
- Daten werden am Pin **PD0** (**RxD**) eingelesen, und am Pin **PD1** (**TxD**) ausgegeben. Wurde der Sende- bzw. Empfängerbaustein eingeschaltet so ist das entsprechende Pin automatisch richtig als Eingang oder Ausgang initialisiert.
- Die Übertragungsgeschwindigkeit der seriellen Schnittstelle wird beim Controller durch Teilung seiner Taktfrequenz eingestellt. Für eine sichere Übertragung über die serielle Schnittstelle sollte ein externer Quarz verwendet werden und die Abweichung zur erwünschten Übertragungsgeschwindigkeit nach Datenblatt 0,5 %[1] nicht überschreiten, da sonst häufiger Fehler auftreten können. Bei kurzen Leitungen können aber ohne weiteres höhere Abweichungen (±2 %) toleriert werden.

1 Nach EIA-232 Standard sollte die Schnittstelle bis zu einer Abweichung von ±4% bei Baudraten unter 19200 Baud funktionieren.

Software-UART

Ein wichtiger Schritt bei der Programmierung ist die Fehlersuche (*Debugging*). Für die Ausgabe von Fehlermeldungen oder Registerinhalten ist die serielle Schnittstelle erste Wahl. Leider ist die einzige serielle Hardware-Schnittstelle oft schon vom Projekt in Beschlag genommen. Eine Lösung ist die Verwendung eines AVR-Controllers mit zwei seriellen Hardware-Schnittstellen wie der zum ATmega32A pinkompatible und sparsame ATmega644p[2]. (das „p“ m

Eine einfachere alternative Lösung ist eine Software-Emulation der Schnittstelle. Sie hat zudem folgende Vorteile:

- Jedes beliebige Pin kann zum Senden verwendet werden kann. Dies gilt auch beim Empfang mittels Polling.
- Jede Baudrate ist bei guter Programmierung mit geringer Abweichung realisierbar unabhängig vom verwendeten Quarz.

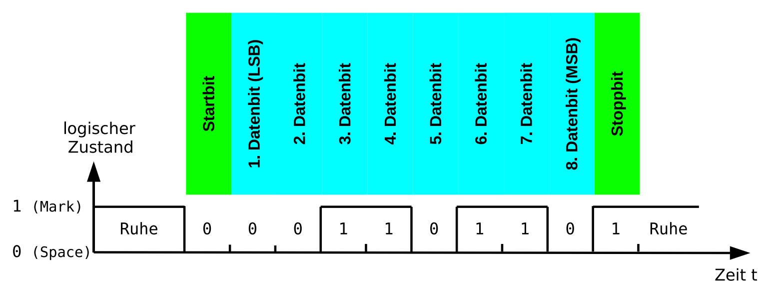
Natürlich gibt es auch Nachteile, da sonst keine Hardware-UARTs existieren würden:

- Bei der Emulation kann der Controller während des Sendens oder des Empfangs mittels Polling sonst keine Aufgaben wahrnehmen.
- Bei hohen Übertragungsgeschwindigkeiten müssen eventuell spezielle Unterprogramme mit entsprechenden Zeitschleifen geschrieben werden um die Fehlerquote klein zu halten.
- Der Code fällt umfangreicher aus.

Software-UART zum Senden.

Das folgende Unterprogramm zeigt eine mögliche Lösung. Die Baudrate kann mittels einer Zuweisung beliebig eingestellt werden. Das Datenformat beträgt fest 8N1, es sind also 10 Bit zu senden. Nach dem Startbit kommt das Zeichen (LSB *first*) und dann ein Stoppbit.

Beispiel:



2 Beim ATmega644p ist darauf zu achten, dass die SF-Register sich teilweise im SRAM befinden. Sie müssen dann mit **`sts`** und **`lds`** anstatt von **`out`** und **`in`** adressiert werden. **`sbi`**, **`cbi`**, **`sbic`** und **`sbis`** können dann nicht verwendet werden. Es muss auf die Bitmaskierung zurückgegriffen werden.

Es wird eine 16 Bit Zeitschleife verwendet. Zur Berechnung des Anfangswerts des Schleifenzähler G wird die genaue Formel benutzt:

$$G = \frac{t_{Bit} - t_T}{4t_T} = \frac{t_{Bit}}{4t_T} - \frac{1}{4}$$

Mit

$$t_{Bit} = \frac{1}{Baud} \quad und \quad t_T = \frac{1}{Takt}$$

ergibt sich:

$$G = \frac{Takt}{4Baud} - \frac{1}{4}$$

Beim Startbit und beim Stoppbit benötigt der **cbi** bzw. der **sbi**-Befehl einen Taktzyklus, so dass ein Viertel (Zeit eines Taktzyklus) abgezogen werden muss. Zur Berechnung des **Zeitschleifenzähler**s ergibt sich damit für das **Start-** bzw. das **Stoppbit** die folgende Formel:

$$G_{Start} = G_{Stopp} = \frac{Takt}{4Baud} - \frac{2}{4}$$

Im Hauptteil des Unterprogramms wird das Zeichen, das sich im Register **r24**[3] befinden muss, 8 mal (Zähler in **r25**) mit dem Schiebebefehl **lsr** nach Rechts geschoben. Das niederwertigste Bit (2^0) wird dabei in das Übertragsbit (**C**arry, Statusregister **SREG**) geschoben. Ein bedingter Sprungbefehl bewirkt dann, dass die Leitung bei einer Null auf Masse bzw. bei einer Eins auf VCC gezogen wird. Nach 8 Schiebeoperationen wurde dann das ganze Zeichen versendet.

Da bei der Verzweigung, je nach Bit eine unterschiedliche Anzahl von Befehlen verwendet wird, muss ein **nop**-Befehle als Zeitausgleich eingefügt werden. Die Verarbeitung der Daten benötigt 8 Taktzyklen. Für den **Zeitschleifenzähler** der **8 Datenbit** ergibt sich folgende Formel:

$$G_{Datenbit} = \frac{Takt}{4Baud} - \frac{9}{4}$$

Da die Berechnungen in Assembler mittels einer Ganzzahldivision erfolgt, findet keine Rundung statt und es kann ein Fehler von 4 Taktzyklen (1 Schleifendurchgang) entstehen. Dies ist bei einer Baudrate von 115200 Baud nicht hinnehmbar, da ein Fehler von 4 Taktzyklen/139 Taktzyklen*100 = 2,9 % entstehen könnte. Um dies zu vermeiden wird die Berechnung mit 10 multipliziert (und später wieder durch 10 geteilt). Durch eine Addition von 5 kann dann eine richtige Rundung durchgeführt werden.

3 Das Doppelregister **r24**:**r25** ist hier für die Parameterübergabe reserviert und wird deshalb auch nicht auf den Stapel gerettet.

Erweiterte Formeln für den Zeitschleifenzähler:

$$G_{Start} = G_{Stopp} = \frac{\frac{Takt*10}{4Baud} - \frac{2*10}{4} + 5}{10}$$

$$G_{Datenbit} = \frac{\frac{Takt*10}{4Baud} - \frac{9*10}{4} + 5}{10}$$

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;       Zuweisungen
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.EQU    TXDDR   = DDRC
.EQU    TXPORT  = PORTC
.EQU    TXPNr   = 1

.EQU    Baud    = 115200
.EQU    Takt    = 16000000

;-------------------------------------------------------------------------------
;       Sende 1 Byte (r24) ueber die serielle Schnittstelle (8N1)
;-------------------------------------------------------------------------------
SSNDC:  push    XL              ;rette verwendete Register
        push    XH

        sbi     TXDDR,TXPNr     ;Ausgang
        sbi     TXPORT,TXPNr    ;Leitung auf 1
        ldi     r25,8           ;Zaehler = 8

        cbi     TXPORT,TXPNr    ;Startbit senden
        ;Zeitschleife fuer das Startbit
        ldi     XL,LOW(((((10*Takt)/(4*Baud))-((2*10)/4)+5)/10))
        ldi     XH,HIGH(((((10*Takt)/(4*Baud))-((2*10)/4)+5)/10))
SSNDC1: sbiw    XL,1
        brne    SSNDC1
        ;Sende Zeichen LSB first!
SSNDC2: lsr     r24             ;Bit in den Uebertrag (Carry) schieben
        brcs    SSNDC3          ;Carry-Bit testen
        cbi     TXPORT,TXPNr    ;Null
        rjmp    SSNDC4          ;zur Zeitschleife
SSNDC3: sbi     TXPORT,TXPNr    ;Eins
        nop                     ;Zeitausgleich
SSNDC4: ;Zeitschleife fuer ein Datenbit
        ldi     XL,LOW(((((10*Takt)/(4*Baud))-((9*10)/4)+5)/10))
        ldi     XH,HIGH(((((10*Takt)/(4*Baud))-((9*10)/4)+5)/10))
SSNDC5: sbiw    XL,1
        brne    SSNDC5
        dec     r25             ;falls nicht alle Bits durch, dann naechstes Bit
        brne    SSNDC2

        nop                     ;Zeitausgleich für Sprungbefehl (MSB)
        sbi     TXPORT,TXPNr    ;Stoppbit senden
        ;Zeitschleife fuer das Stoppbit
        ldi     XL,LOW(((((10*Takt)/(4*Baud))-((2*10)/4)+5)/10))
        ldi     XH,HIGH(((((10*Takt)/(4*Baud))-((2*10)/4)+5)/10))
SSNDC6: sbiw    XL,1
        brne    SSNDC6

        pop     XH              ;Register wiederherstellen
        pop     XL
        ret                     ;zurueck
```

✎ **C000** a) Zeichne das Flussdiagramm zum Unterprogramm.

b) Das obige Unterprogramm befindet sich in der Bibliothek "**SR_UART_SOFT_sample.asm**".
Schreibe ein Hauptprogramm zum Testen des Unterprogramms. Gib dem Hauptprogramm den Namen "**C000_UART_SOFT_test1.asm**".

c) Welche maximale Abweichung ergibt sich bei einer Baudrate von 115200 Baud und bei einer Baudrate von 9600 Baud? Sind diese Abweichungen hinnehmbar? Begründe.

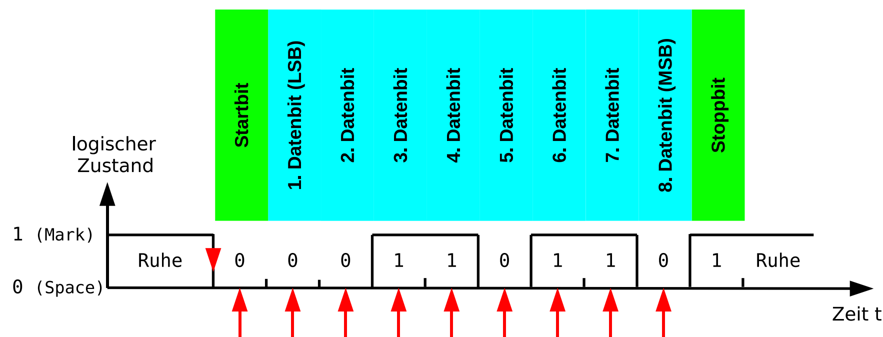
✎ **C001** Die Bibliothek soll um ein ein Unterprogramm mit dem symbolischen Namen **SSNDC11** für eine feste Baudrate von 115200 Baud (16 MHz Quarz) erweitert werden. Es ist eine einfache 8-Bit Zeitschleife zu verwenden. Ein Zeitfehler soll mit **nop**- Befehlen ausgeglichen werden.

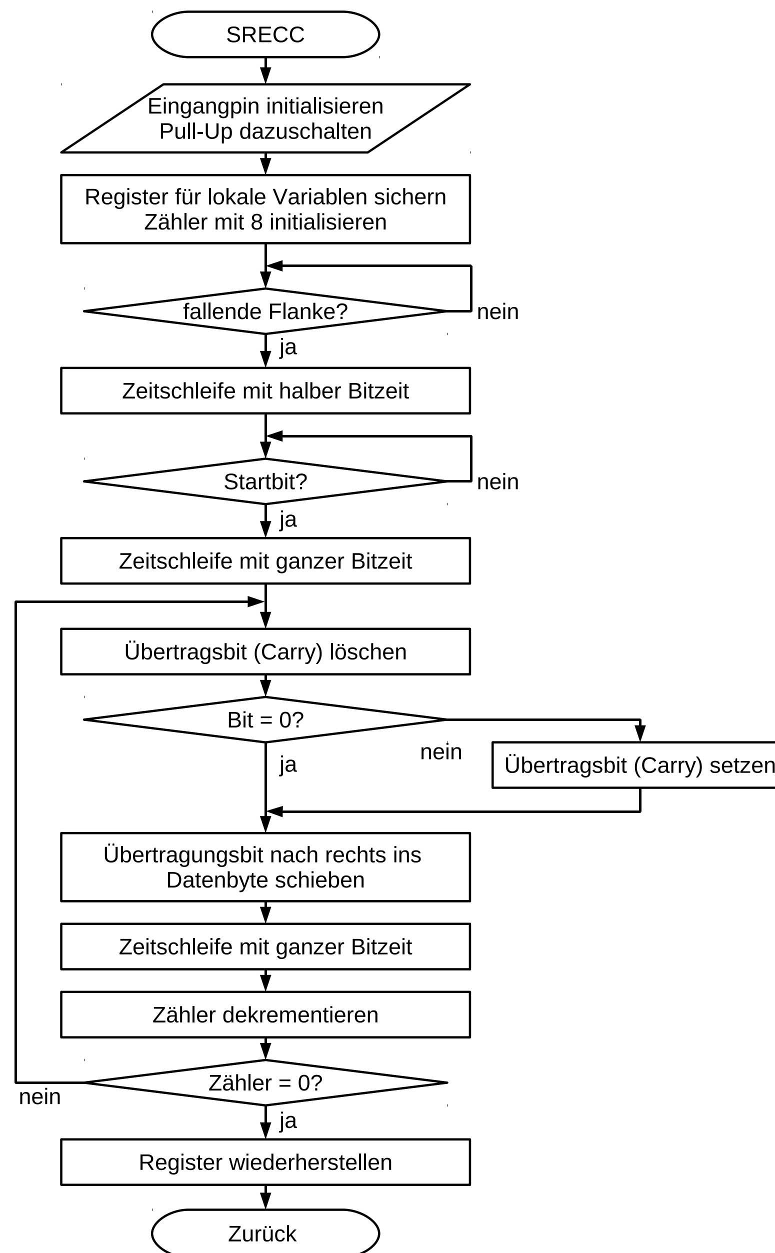
a) Teste das Unterprogramm. Nenne die Bibliothek "**SR_UART_SOFT.asm**" und das Hauptprogramm "**C001_UART_SOFT_test2.asm**".

b) Sende das Zeichen **'U'** und miss die Baudrate mit dem Oszilloskop. Der Fehler darf maximal 0,5 % betragen!

Software-UART zum Empfangen.

Ein Unterprogramm für die Bibliothek soll ähnlich wie beim Senden den Empfang mittels Software ermöglichen. Das Abtasten soll zur Mitte der Bitzeit erfolgen. Die Schiebeoperation und die Berechnung der Zeitschleifen erfolgt ähnlich wie beim Senden.





✎ **C002** Die Bibliothek soll um ein ein Unterprogramm mit dem symbolischen Namen **`SRECC`** erweitert werden. Programmiere und teste das Unterprogramm mit 115200 Baud. Im Hauptprogramm wird dazu das von der PC-Tastatur empfangene Byte als Echo zurück zum PC geschickt wird. Nenne das Hauptprogramm "**`C002_UART_SOFT_test3.asm`**".

Software-UART mit Interrupt

Besonders beim Empfang wird der Controller lange blockiert, da nicht gewusst ist, wann Daten eintreffen werden. Hier bietet es sich an einen externen Interrupt zu opfern um auf eintreffende Daten reagieren zu können.

✎ **C003** Die Bibliothek soll um ein eine Interruptroutine mit dem symbolischen Namen **ISRSRC** erweitert werden. Die negative Flanke des Startbit an **PB2** (**INT2**) soll den Empfang des Datenbyte auslösen. Programmiere und teste die ISR. Im Hauptprogramm wird dazu das von der PC-Tastatur empfangene Byte als Echo zurück zum PC geschickt wird. Nenne das Hauptprogramm "**C003_UART_SOFT_test4.asm**".

Bemerkungen: Falls möglich sollte beim Software-UART und auch beim Debuggen immer mit der höchstmöglichen Baudrate gearbeitet werden. Setzt man die serielle Schnittstelle zum Debuggen ein, so muss man darauf achten das Timing des eigentlichen Programms nicht beeinflusst wird.

Um weiter Arbeitszeit des Controllers einzusparen können die Zeitschleifen durch Timerinterrupts ersetzt werden.

Durch Mehrfachtastung könnten Fehler entdeckt werden und die Übertragung bei großen Leitungslängen (bzw. hoher Störstrahlung) verbessert werden.

Wie schon erwähnt besitzt der pinkompatible ATmega644p zwei Hardware-USARTs. Er bietet aber auch die Möglichkeit an jedem beliebigen Pin einen externen Interrupt auszulösen und somit interruptgesteuert Daten zu empfangen.