Das Registermodell

1. Modell

1.1 Einen vollständigen Überblick über den Aufbau des Attiny2313 erhält man auf S. 3 des *ATMEL Attiny 2313*-Manuals. Hier beschränken wir uns auf das Zusammenspiel von Mikroprozessor und Peripherie über die I/O-Ports. Als Peripherie betrachten wir hier den bereits bekannten Port B und die **USART** (**U**niversal **S**ynchron **A**synchronous **R**eceiver/**T**ransmitter), welche für die serielle Kommunikation zuständig ist.

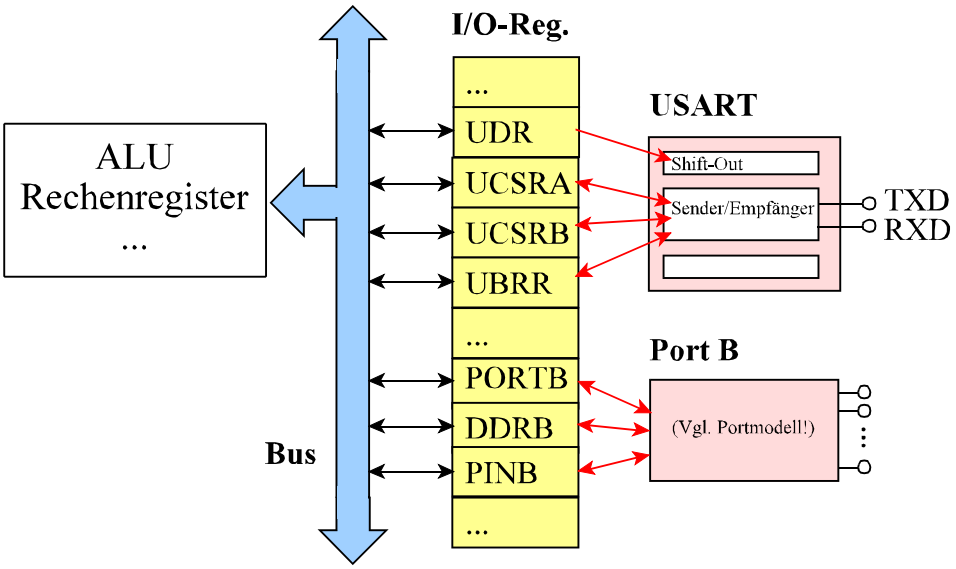
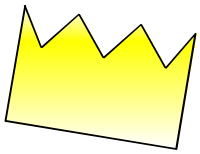


Abb. 1: Vereinfachtes Modell des Attiny2313

1.2 Das Funktionsprinzip könnte man nennen: Teile und herrsche. Der Mikroprozessor tauscht Daten über den BUS mit den I/O-Registern aus. Dadurch werden die damit verbundenen Peripherieeinheiten (z. B. Port B oder USART) gesteuert und kontrolliert. Um ein Byte über die serielle Schnittstelle zu senden, muss z. B.



- die USART mit dem Kontrollregister UCSRB eingeschaltet werden,
- die Baudrate mit dem Register UBRR festgelegt werden,
- das Byte in das UDR-Register geschrieben werden.

Während die ersten beiden Aktionen lediglich die USART initialisieren, wird durch die letzte Aktion der eigentliche Sendevorgang ausgelöst. Die Übertragung wird alleine von der USART durchgeführt, die natürlich auch Zugriff auf die entsprechenden I/O-Ports hat. Der Mikroprozessor kann sich derweil anderen Aufgaben widmen; diese Aufgabenteilung bedeutet natürlich eine starke Entlastung des Mikroprozessors. Da bei einer Baudrate von 9600 Baud die Übertragung eines Bytes ca 1 ms dauert, kann der Mikroprozessor in dieser Zeit (bei einer Taktfrequenz von 4 MHz) ca. 2000 Befehle (mit je 2 Taktzyklen) abarbeiten und sich damit anderen Herrscher-Aufgaben widmen.

Wichtige Bemerkung: Der Mikroprozessor wird durch BASCOM mit üblichen Strukturelementen programmiert (if ... then ... else, do ... loop, ...). Die Peripherie lässt sich über die I/O-Ports steuern. Es gibt zwar auch zahlreiche BASCOM-Befehle, die solche Aufgaben übernehmen (z. B. print, config...). Diese sind aber in ihrer Wirkungsweise manchmal etwas kompliziert und wenig systematisch; außerdem verschleiern sie die eigentliche Funktionsweise. Und nicht zuletzt führt der direkte Zugriff auf die I/O-Register zu deutlich kompakterem Maschinencode.

2. Register-Tabelle

Einen Überblick über die I/O-Register des Attiny liefert die Register-Tabelle (Tabelle 1 auf der nächsten Seite). Schon bekannt sind uns die Register PortB, PinB und DDRB. Registeradressen beginnen bei der Adresse $0020 bzw. $0000, je nachdem ob man die Rechenregister mitzählt oder nicht. Alle I/O-Register haben mehr oder weniger verständliche Namen - ebenso deren einzelne Bits (s. u.).

Die für uns wichtigen Register der USART haben die Adressen $0029 bis $002C: UBRR(L), UCSRB, UCSRA, UDR. Welche Bedeutung diese Bytes und Bits haben, werden wir gleich sehen. Vorher aber noch eine Bemerkung dazu, wie BASCOM mit diesen I/O-Registern umgeht.

3. Bits und Bytes der I/O-Register bei BASCOM

3.1 Schreibweise

Wie geht man mit der Registertabelle um? Am Beispiel des bekannten Registers PORTB wollen wir dies darlegen:

Adresse	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Seite
$0038	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	57

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	7
0x3E (0x5E)	Reserved	–	–	–	–	–	–	–	–	
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
0x3C (0x5C)	OCR0B	Timer/Counter0 – Compare Register B								77
0x3B (0x5B)	GIMSK	INT1	INT0	PCIE	–	–	–	–	–	59
0x3A (0x5A)	EIFR	INTF1	INTF0	PCIF	–	–	–	–	–	61
0x39 (0x59)	TIMSK	TOIE1	OCIE1A	OCIE1B	–	ICIE1	OCIE0B	TOIE0	OCIE0A	78, 109
0x38 (0x58)	TIFR	TOV1	OCF1A	OCF1B	–	ICF1	OCF0B	TOV0	OCF0A	78
0x37 (0x57)	SPMCSR	–	–	–	CTPB	RFLB	PGWRT	PGERS	SELFPRGEN	155
0x36 (0x56)	OCR0A	Timer/Counter0 – Compare Register A								77
0x35 (0x55)	MCUCR	PUD	SM1	SE	SM0	ISC11	ISC10	ISC01	ISC00	52
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	36
0x33 (0x53)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	76
0x32 (0x52)	TCNT0	Timer/Counter0 (8-bit)								77
0x31 (0x51)	OSCCAL	–	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	25
0x30 (0x50)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	73
0x2F (0x4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1BO	–	–	WGM11	WGM10	104
0x2E (0x4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	***107***
0x2D (0x4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								108
0x2C (0x4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								108
0x2B (0x4B)	OCR1AH	Timer/Counter1 – Compare Register A High Byte								108
0x2A (0x4A)	OCR1AL	Timer/Counter1 – Compare Register A Low Byte								108
0x29 (0x49)	OCR1BH	Timer/Counter1 – Compare Register B High Byte								109
0x28 (0x48)	OCR1BL	Timer/Counter1 – Compare Register B Low Byte								109
0x27 (0x47)	Reserved	–	–	–	–	–	–	–	–	
0x26 (0x46)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	27
0x25 (0x45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								109
0x24 (0x44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								109
0x23 (0x43)	GTCCR	–	–	–	–	–	–	–	PSR10	81
0x22 (0x42)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	108
0x21 (0x41)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	41
0x20 (0x40)	PCMSK	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	61
0x1F (0x3F)	Reserved	–	–	–	–	–	–	–	–	
0x1E (0x3E)	EEAR	–	EEPROM Address Register							15
0x1D (0x3D)	EEDR	EEPROM Data Register								16
0x1C (0x3C)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	16
0x1B (0x3B)	PORTA	–	–	–	–	–	PORTR2	PORTA1	PORTA0	57
0x1A (0x3A)	DDRA	–	–	–	–	–	DDA2	DDA1	DDA0	57
0x19 (0x39)	PINA	–	–	–	–	–	PINA2	PINA1	PINA0	57
0x18 (0x38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	57
0x17 (0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	57
0x16 (0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	57
0x15 (0x35)	GPIOR2	General Purpose I/O Register 2								20
0x14 (0x34)	GPIOR1	General Purpose I/O Register 1								20
0x13 (0x33)	GPIOR0	General Purpose I/O Register 0								20
0x12 (0x32)	PORTD	–	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	57
0x11 (0x31)	DDRD	–	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	57
0x10 (0x30)	PIND	–	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	57
0x0F (0x2F)	USIDR	USI Data Register								144
0x0E (0x2E)	USISR	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0	145
0x0D (0x2D)	USICR	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC	146
0x0C (0x2C)	UDR	UART Data Register (8-bit)								129
0x0B (0x2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM	129
0x0A (0x2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	131
0x09 (0x29)	UBRRL	UBRRH[7:0]								133
0x08 (0x28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	149
0x07 (0x27)	Reserved	–	–	–	–	–	–	–	–	
0x06 (0x26)	Reserved	–	–	–	–	–	–	–	–	
0x05 (0x25)	Reserved	–	–	–	–	–	–	–	–	
0x04 (0x24)	Reserved	–	–	–	–	–	–	–	–	
0x03 (0x23)	UCSRC	–	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	132
0x02 (0x22)	UBRRH	–	–	–	–	UBRRH[11:8]				133
0x01 (0x21)	DIDR	–	–	–	–	–	–	AIN1D	AIN0D	150
0x00 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Tabelle 1

Der Eintrag in der Registertabelle sagt uns folgendes:

- Das Register PORTB hat die Adresse $0038.
- Die einzelnen Bits dieses Registers haben die Bezeichner Bit7, Bit6, ...
- Nähere Informationen findet man im Manual auf S. 57.

Um Werte in dieses Register zu schreiben, gibt es verschiedene Möglichkeiten. Um z. B. den Wert 85 in das Register PORTB zu schreiben, haben wir bisher den Befehl

```
PORTB = 85
```

benutzt. Da das PORTB-Register die Adresse $38 hat, ist dies gleichwertig zu

```
out $38, 85
```

Die erste Schreibweise hat einen wichtigen Vorteil: Sie ermöglicht über die Punkt-Schreibweise einen einfachen Zugriff auf die einzelnen Bits eines Registers: Mit `PORTB.5 = 1` wird z. B. das Bit 5 auf den Wert 1 gesetzt; die anderen Bits dieses Registers bleiben dabei unverändert. Die Bitnummer kann auch durch den Bitbezeichner - in diesem Fall `PORTB5` ersetzt werden. Statt `PortB.5` kann man also auch `PORTB.PORTB5` schreiben. In diesem Fall bringen die Bitbezeichner keine Erleichterung; weiter unten werden wir den Vorteil dieser Schreibweise jedoch erkennen.

3.2 Die Register-Definitionsdatei `Attiny2313.dat`

Die Inhalte der Register-Tabelle sind in Register-Definitionsdateien festgehalten. Da die verschiedenen Mikrocontroller sich in dem einen oder anderen Register unterscheiden können, gibt es für jeden Mikrocontroller eine eigene Register-Definitionsdatei. Im Falle des Attiny2313 heißt sie `Attiny2313.dat`. Sie können diese Datei mit einem einfachen Editor studieren.

4. Die USART

4.1 Vereinfachter Aufbau

Wird ein Wert in das Register **UDR** (**U**SART **D**ata **R**egister) geschrieben, so wird er in das interne Shift-Out-Register der USART übernommen und über den TXD-Ausgang gesendet. In ähnlicher Weise gelangen Daten, die über den RXD-

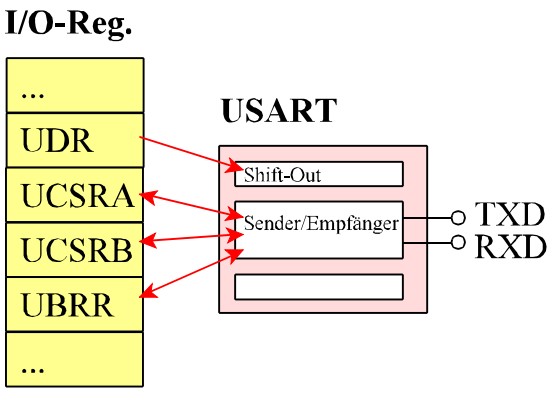


Abb. 2: Die USART und ihre Register

Eingang empfangen werden, über einen anderen Puffer in das UDR-Register und können dort ausgelesen werden. Da Senden und Empfangen recht ähnlich ablaufen, schauen wir uns nur einen dieser beiden Vorgänge genauer an, und zwar das Senden.

4.2 Steuer- und Kontrollregister der USART

Der Sendevorgang wird durch die Register UCSRA, UCSRB, UCSRC und UBRR kontrolliert. Dabei ist das Register UCSRC schon für alle uns interessierenden Fälle so voreingestellt, dass wir uns nicht mehr darum kümmern müssen. Die USART-Register werden ab S. 129 im Manual beschrieben.

Beginnen wir mit dem Register **UBRR** (**U**SART **B**aud **R**ate **R**egister). Der hier einzutragende Wert ist nicht die Baudrate selbst; vielmehr kennzeichnet er nur die Baudrate auf der Grundlage der benutzten Taktfrequenz. Eine Formel zur Berechnung des UBRR-Wertes findet man auf S. 114 des Manuals. Leichter ist es allerdings auf die Tabelle 57 der S. 135 zurückzugreifen. Dort finden wir:

```
UBRR = 25       ' für f = 4,0 MHz und baud = 9600
```

Das schreiben wir sinnvollerweise in den Initialisierungsteil.

Betrachten wir nun zunächst das Register **UCSRB** (**U**SART **C**ontrol and **S**tatus **R**egister **B**). Auf S. 131 des Manuals finden wir:

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	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Jedes einzelne Bit von Register UCSRB hat eine spezielle Bedeutung, die durch eine Abkürzung gekennzeichnet ist. Das TXEN-Bit kennzeichnet z. B., ob der Sender (**T**ransmitter) eingeschaltet (**en**abled) ist oder nicht. Die unterste Zeile macht deutlich: Der Standardwert (Initial Value) von TXEN ist 0, d. h. der Sender ist ausgeschaltet. In der Zeile darüber erkennen wir, dass dieses Bit gelesen und geschrieben werden kann.

Hier nun die Bedeutung der im Moment wichtigen Bits (Für die restlichen sei auf das Manual verwiesen. Einige davon werden wir allerdings noch im Zusammenhang mit Interrupts kennenlernen.):

Wichtige Bits des UCSRB-Registers

RXEN: Schaltet den Empfänger ein (RXEN = 1) oder aus (RXEN = 0); Port D.0 (RXD) wird als Eingang konfiguriert.

TXEN: Schaltet den Sender ein (TXEN = 1) oder aus (TXEN = 0); Port D.1 (TXD) wird als Ausgang konfiguriert.

Wichtige Bits des UCSRA-Registers

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	UPE	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	

UDRE: (**U**SART **D**ata **R**egister **E**mpty) UDRE = 1 (Startwert!) zeigt an, dass das Register UDR leer ist und somit bereit ist, einen neuen Wert zu übernehmen. Solange UDRE = 0 ist, wird jeder Versuch, einen Wert ins UDR zu schreiben, ignoriert. (Weitere Erläuterungen s. u.)

TXC: (**T**ransmit **C**omplete) TXC = 1 zeigt an, dass die Daten aus dem Shift-Out-Puffer vollständig übertragen wurden, vgl. unten 4.3.2

RXC: (**R**eceive **C**omplete) RXC = 1, wenn sich im 2(!)-Byte-Eingangspuffer (in Abb. 4 nur angedeutet) ein ungelesener Wert befindet. RXC wird automatisch gelöscht, wenn alle Daten über [Variablenname] = UDR gelesen wurden.

Der Ablauf eines Sendevorgangs im Detail: UDR und Shift-Out sind zunächst leer (UDRE = 1). Nun wird ein erster Wert ins UDR geschrieben; dieser wird sofort(!) ins Shift-Out geschoben, d. h. UDR ist gleich wieder leer und UDRE = 1. Jetzt wird ein nächster Wert nach UDR geschrieben (das geht, weil noch UDRE = 1 ist). Shift-Out ist aber noch längst nicht leer, weil serielle Übertragung relativ lange dauert. Der Wert von UDR kann noch nicht ins Shift-Out übernommen werden; UDR wird durch UDRE = 0 gesperrt: Ein weiterer (dritter) Schreibversuch wird von UDR ignoriert. Erst wenn Shift-Out entleert ist (durch das vollständigen Senden des ersten Bytes), wird der Inhalt von UDR ins Shift-Out übernommen; dort steht jetzt der zweite Wert. UDR ist jetzt leer und UDRE steht somit wieder auf 1. Der zweite Wert wird jetzt gesendet; der dritte Wert ist verloren gegangen (so wie alle weiteren, die vom UDR ignoriert wurden).

Will man nun vermeiden, dass Werte für die Übertragung verloren gehen, muss man mit der Übergabe dieser Werte an das UDR hinreichend lange warten: Dazu gibt es verschiedene Möglichkeiten:

1. Man setzt einen waitms-Befehl ein: Nach einigen Millisekunden sollte ein Byte schließlich übertragen sein!

2. Man wartet solange, bis UDRE = 1 ist.

Die zweite Lösung ist natürlich eleganter und (hinsichtlich des Zeitbedarfs) auch ökonomischer.

4.3 Beispiele

4.3.1

Die Zahlen von 0 bis 255 sollen über die serielle Schnittstelle übertragen werden - wenig spektakulär, aber lehrreich. Das Programm finden Sie auf der nächsten Seite. Hier wurden nicht die Nummern der einzelnen Kontrollbits, sondern stellvertretend deren Bezeichner benutzt. Dies erleichtert sowohl das Schreiben wie auch das Lesen ungemein; denn wer kann sich schon merken, dass das UDRE-Bit die Nummer 5 des UCSRA-Registers ist.

Nach der Übertragung des Programms auf den Attiny startet es, und jetzt haben wir noch 5 Sekunden Zeit, das Terminal-Programm des Uploaders zu aktivieren: Lasche Terminal anklicken, Empfangene Daten als Zahl anzeigen, Empfang starten...

Zügig werden die einzelnen Daten übertragen. In Wirklichkeit ist die Übertragung noch etwas rascher als sie zu beobachten ist. Das Terminalprogramm ist bei der Darstellung langsamer als die Daten ankommen. Glücklicherweise besitzt die USART des PCs einen recht großen Puffer, in welchem die empfangenen Daten zwischengelagert werden.

Fragen:

1. Wie kann man die letzte Aussage testen?

2. Was geschieht, wenn man das WarteAufUSART-Unterprogramm weglässt?

3. Was geschieht, wenn man in dem WarteAufUSART-Unterprogramm jeweils nur 100µs wartet? (Benutzen Sie: waitus 100.) Welche Wartezeit ist für eine einwandfreie (d. h. vollständige) Übertragung mindestens nötig?

```
Declare Sub Warteaufusart
Dim I As Byte

Ucsrb.txen = 1                'USART-Sender aktivieren;
Ubrr = 25                     'Baudrate, vgl. Tabelle S. 135 des Manuals

'******************** Hauptprogramm ***********************

Wait 5          'Noch 5 s Zeit, um Terminal nach dem Upload zu aktivieren

For I = 0 To 255
  Udr = I                     'Schleifenindex ins UDR
  Call Warteaufusart
Next I

End

'******************* Unterprogramme ***********************

Sub Warteaufusart
  Do
  Loop Until Ucsra.udre = 1   'warten bis UDR leer
End Sub
```

4.3.2 Das Warten kann auch über das Status-Bit TXC kontrolliert werden. Allerdings muss das TXC-Bit jedesmal von Hand gelöscht werden, bevor ein Wert in das UDR geschrieben wird. Das Löschen geschieht nicht, wie man vielleicht erwarten würde, durch UCSRA.TXC = 0, sondern durch UCSRA.TXC = 1(!), vgl. Manual S. 129 ("... it can be cleared by writing a one into its bit location."). Dagegen erfolgt das Löschen des RXC-Bits automatisch, vgl. 4.2.

4.4 Das Empfangen

Das Empfangen von Daten über die serielle Schnittstelle läuft fast genauso ab wie das Senden - lediglich die Richtung des Datenstroms ist umgekehrt: Die Daten werden über die RXD-Leitung in einem Puffer zwischengespeichert. Dieser Shift-In-Puffer ist vom Typ FIFO (First In - First Out); er besteht aus 2(!) Bytes. Durch den BASCOM-Befehl x = UDR wird ein Wert aus diesem Puffer in das UDR geschoben und in der Variablen x gespeichert.

Abb. 53 auf S. 111 des Manuals gewährt einen genaueren Einblick in die Verdrahtung zwischen den verschiedenen Puffern und dem UDR-Register. Hier wird deutlich, dass das UDR eigentlich aus zwei verschiedenen Registern besteht, welche sich aber eine gemeinsame Adresse ($002C) teilen. Bei einem Schreibvorgang (UDR = ...) wird automatisch auf das Sende-UDR (mit dem Shift-Out-Puffer verbunden) zugegriffen, beim Lesevorgang auf das Empfangs-UDR (mit dem Shift-In-Puffer verbunden).

5. Aufgaben

5.1 Testen Sie das Programm aus 4.3.1 aus (USART6.BAS): Wie verhält es sich, wenn man die USART einzuschalten "vergisst"? Was geschieht, wenn die Baudratenkennung BRR um 1, 2 oder 5 Einheiten nach oben oder unten von 25 abweicht? Versuchen Sie auch, das Warte-Unterprogramm mithilfe des TXC-Bits zu schreiben.

5.2 Schreiben Sie ein Programm, welches eine einzige Zahl, die vom Terminal gesendet wird, auf Port B ausgibt. Benutzen Sie wieder die USART-Register.

5.3 Schreiben Sie das Cäsar-Verschlüsselungs-Programm mithilfe der USART-Register. Versuchen Sie auch, im Manual die Bedeutung des RXC-Registers zu recherchieren.

6. Zusammenfassung

Durch den direkten Zugriff auf die Kontroll-und Status-Register haben wir die volle Kontrolle über die Mikrocontrollerperipherie. Wenn man nur auf die BASCOM-Befehle zurückgreift, gelingt dies nicht immer zufriedenstellend. Außerdem führen die BASCOM-Befehle zu umfangreicherem Maschinencode. Sie lassen sich auch nicht so einfach auf andere Programmiersprachen wie Pascal oder auch Assembler übertragen.

Das Beispiel mit der USART hat (hoffentlich) auch deutlich gemacht, dass durch den Umgang mit den I/O-Registern klar wird, wie die Vorgänge im Innern der USART eigentlich ablaufen und gesteuert werden können. Befehle wie $baud = ... und print ... hingegen lassen dies nicht erkennen.