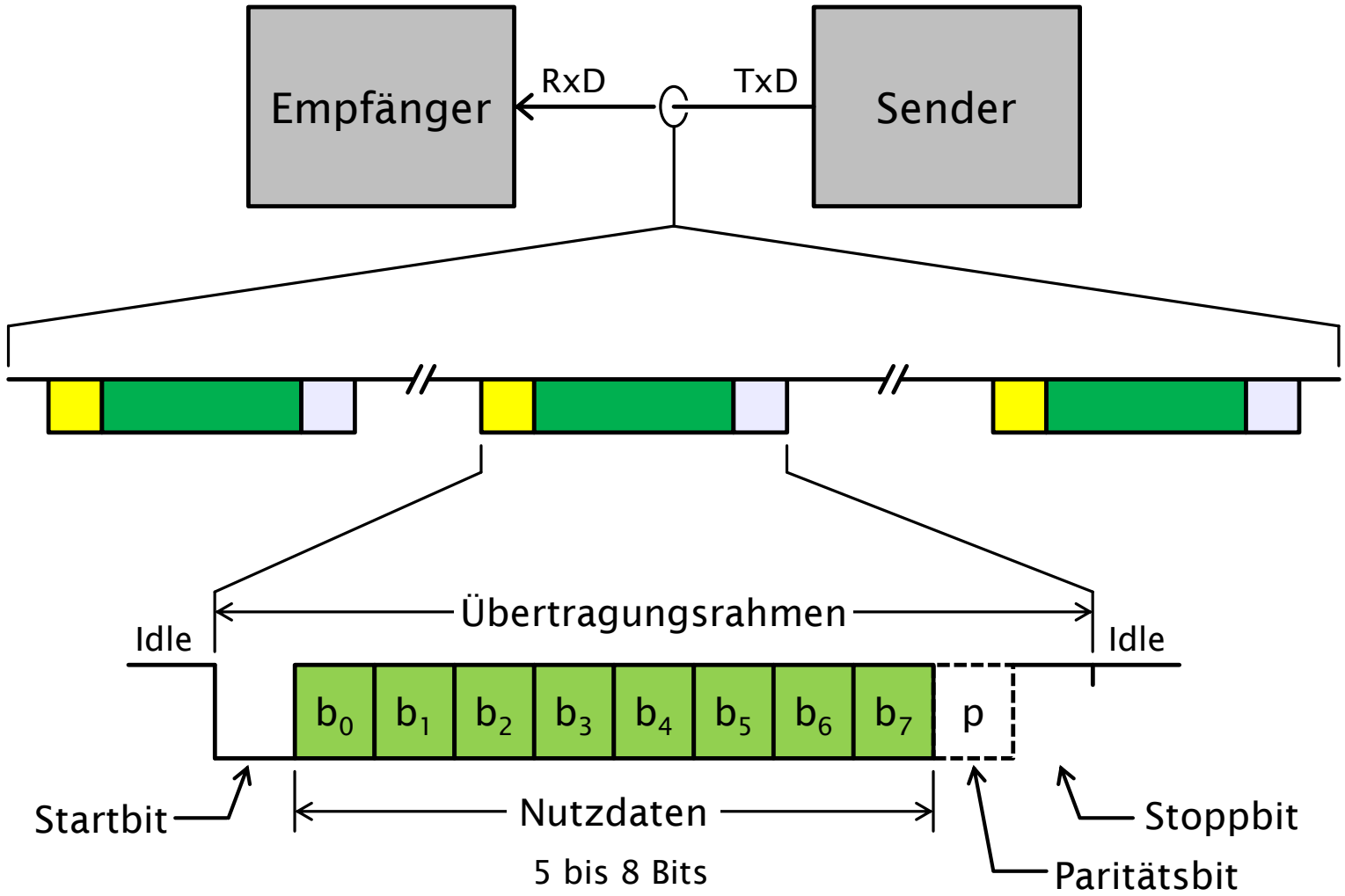
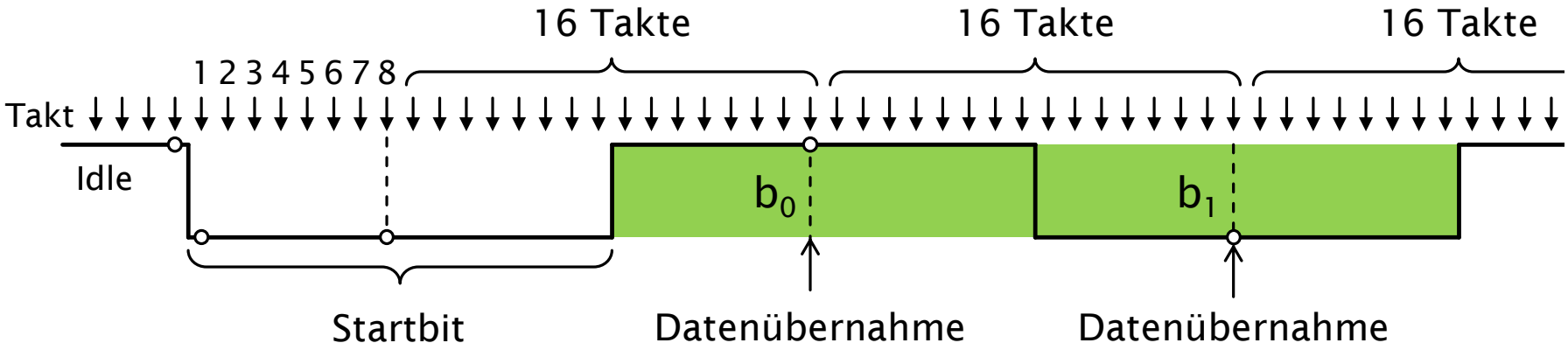
Asynchron-serielle Datenübertragung

- im industriellen Umfeld weit verbreitet:
  - RS232: 1-zu-1-Verbindung
  - RS422/423: 1-zu-N-Verbindung
  - RS485: N-zu-M-Verbindung (Master/Slave), in sog. Feldbussen
- Eigenschaften
  - einfache Realisierbarkeit
  - je eine Datenleitung pro Übertragungsrichtung + Masseleitung
  - Übertragung ohne eigene Taktleitung => im Empfänger Taktrekonstruktion aus dem Datenstrom notwendig
  - zwischen Sender und Empfänger müssen vereinbart sein:
    - Baudrate (Übertragungsgeschwindigkeit)
    - Übertragungsrahmen (Datenformat)
  - Einfache Erkennungsmechanismen des Übertragungsrahmens: ein Startbit und ein Stoppbit (i.a. Start- und Stoppsequenzen)


Empfänger
RxD
TxD
Sender
Übertragungsrahmen
Idle
Idle
b0
b1
b2
b3
b4
b5
b6
b7
p
Startbit
Nutzdaten
5 bis 8 Bits
Stoppbit
Paritätsbit

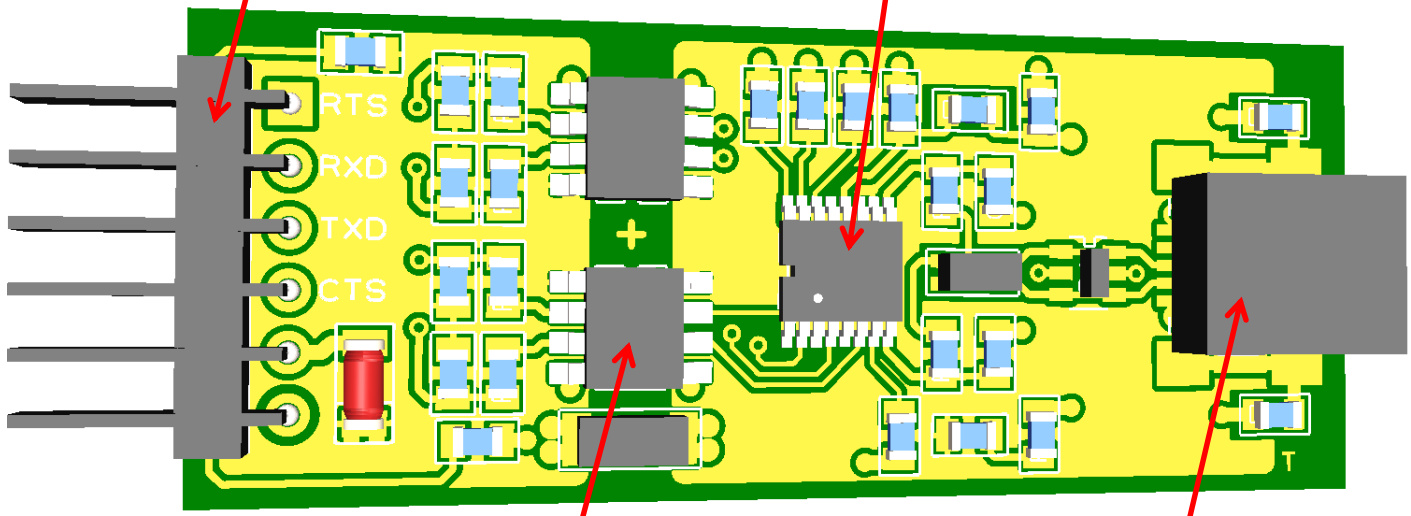
- Datenempfang
  - 16fache Überabtastung des Datenstroms
    - Beispiel: Übertragungsgeschwindigkeit 9600 bps (bits per second)
      => Abtastfrequenz: Takt = 16*9600 = 153,6 kHz
  - Startbiterkennung
    - Übergang von High auf Low
    - Abfrage in der Mitte des Intervalls
      => wenn Signal=Low, dann Startbit erkannt, sonst Fehler



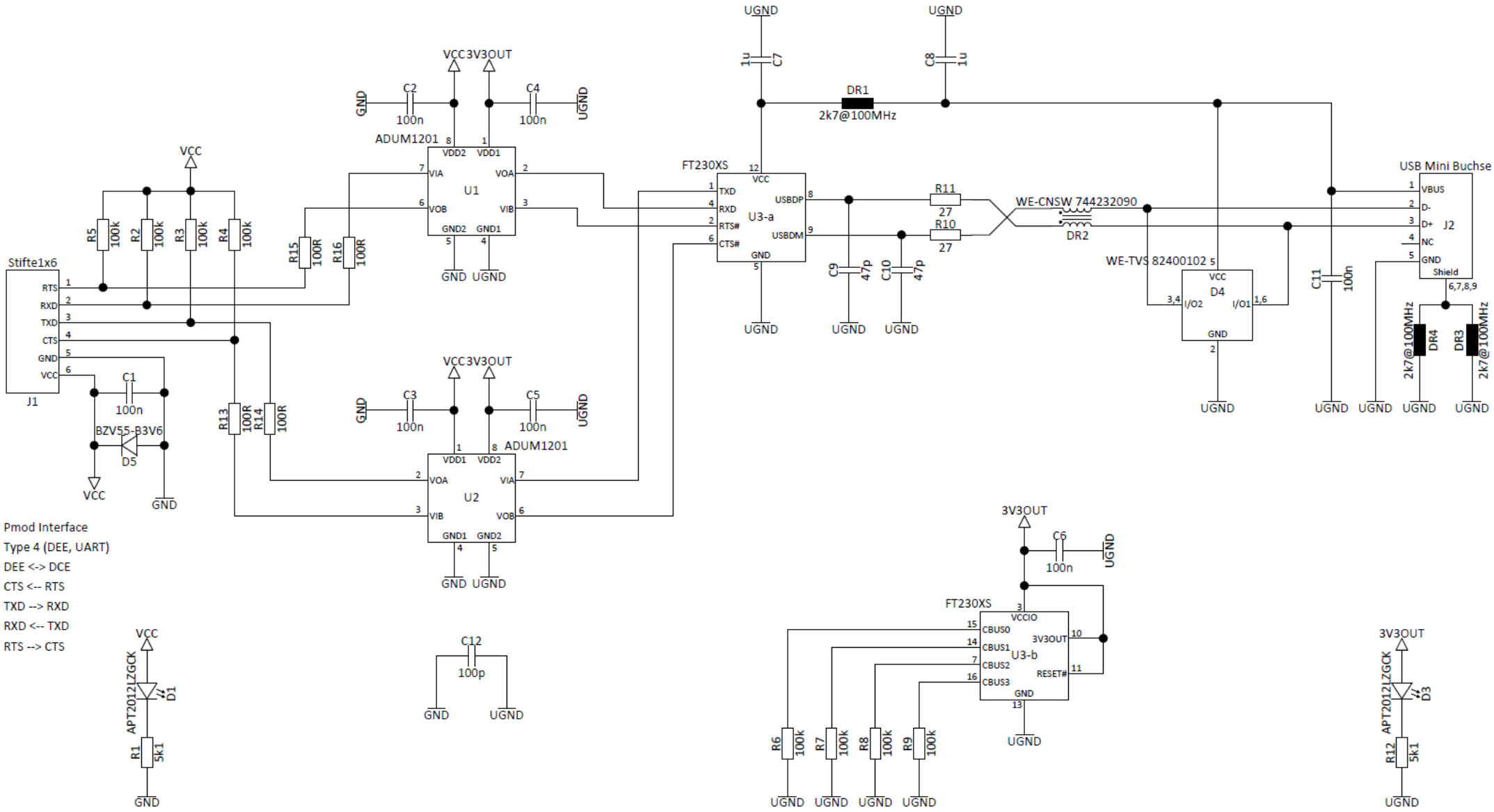
  - Datenübernahme erfolgt in der Mitte eines Bit-Intervalls

UART2USB: Platine

Pin	Signal	Dir.
1	RTS	Out
2	RXD	In
3	TXD	Out
4	CTS	In
5	GND	
6	VCC	

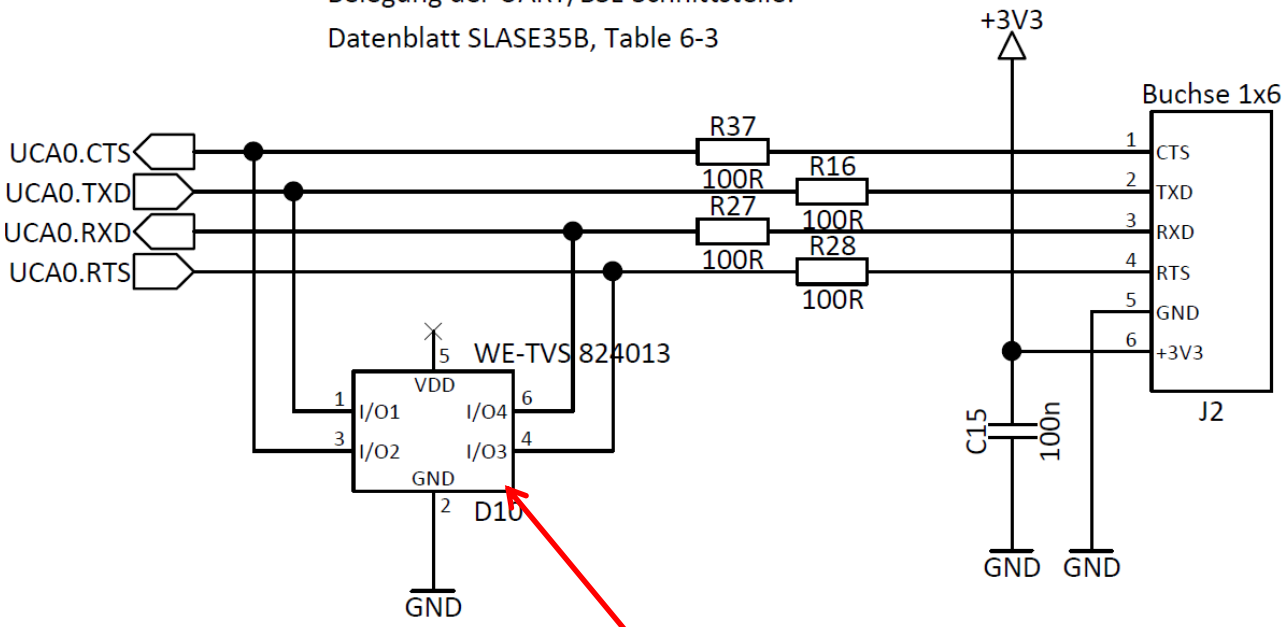


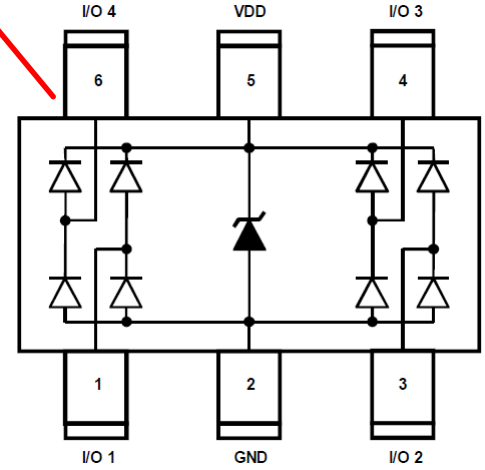
UART2USB: Schaltplan

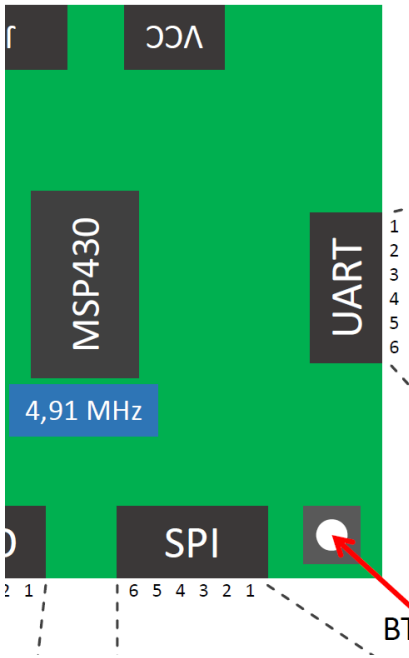


Mainboard: Schnittstelle zu UART2USB

Belegung der UART/BSL-Schnittstelle:
Datenblatt SLASE35B, Table 6-3







PMOD Type 4

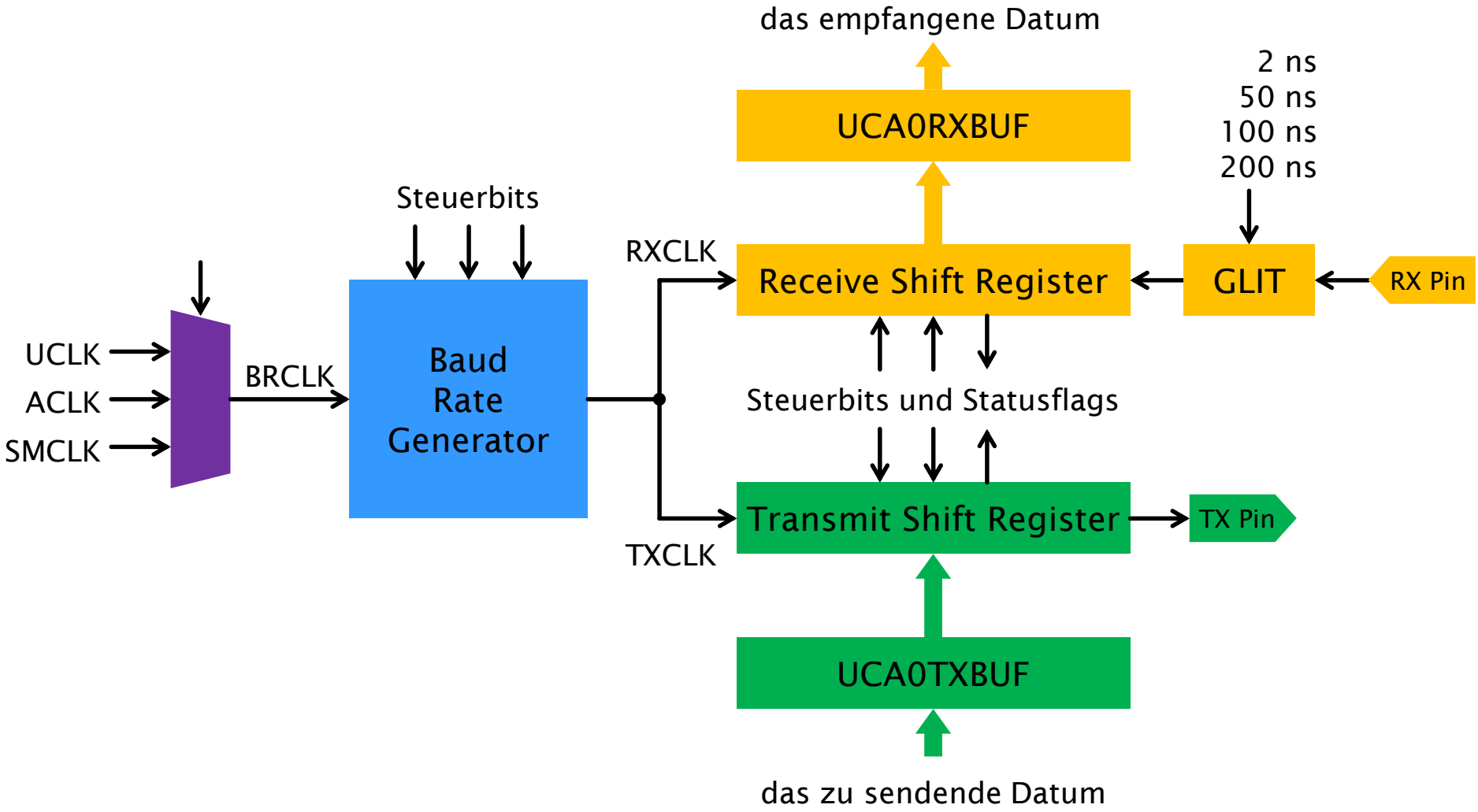
Nr	Pin	MCU
1	P1.4	13
2	UCA0.TXD	23
3	UCA0.RXD	24
4	P1.5	14
5	GND	-
6	+3,3V	-

MSP430: eUSCI

- Enhanced Universal Serial Communication Interface
  - Synchrones Kommunikationsprotokoll SPI (3-/4-Leitungen)
  - Asynchrones Kommunikationsprotokoll UART
  - Erweitertes UART mit automatische Baudratendetektion
  - Protokoll für eine Infrarotschnittstelle IrDA

- MSP430FR5729 hat
  - Zwei Module eUSCI_A0 und eUSCI_A1 für SPI, UART, IrDA
  - Ein Modul eUSCI_B0 für SPI und I2C

USCI_A0: Aufbau



USCI_A0: Initialisierung

- Die Initialisierung eines eUSCI-Moduls für die asynchron-serielle Datenübertragung erfolgt in folgenden Schritten: (am Beispiel der UCA0-Komponente)
  1. Die beiden GPIO-Ports für die Signale RxD und TxD werden passend konfiguriert.
  2. Das Bit UCSWRST (software reset enable) im Steuerregister UCA0CTLW0 wird auf 1 gesetzt. Das führt zu einem Reset-Vorgang im UCA-Modul.
  3. Bei UCSWRST=1 werden restliche Steuerregister des UCA0-Moduls mit geeigneten Werten beladen.
  4. Das Bit UCSWRST im Steuerregister UCA0CTLW0 wird zurückgesetzt.
  5. Das Interrupt-Enable-Flag UCRXIE im Statusregister UCA0IE wird auf 1 gesetzt, sofern erforderlich.

USCI_A0: Initialisierung

- Pins 0 und 1 am Port P2 dienen als TXD und RXD

P2.x	FUNCTION	P2DIR.x	P2SEL1.x	P2SEL0.x
0	GPIO	I:0, O:1	0	0
	TB2.CCI0A TB2.0	0 1	0	1
	UCA0TXD UCA0SIMO	**X(1)** X(1)	1	0
	TB0CLK ACLK	0 1	1	1
1	GPIO	I:0, O:1	0	0
	TB2.CCI1A TB2.1	0 1	0	1
	UCA0RXD UCA0SOMI	**X(1)** X(1)	1	0
	TB0.CCI0A TB0.0	0 1	1	1

(1) Direction controlled by eUSCI_A0 module

Table 18-8. UCAxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCPEN	RW	0h	Parity enable 0b = Parity disabled 1b = Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.
14	UCPAR	RW	0h	Parity select. UCPAR is not used when parity is disabled. 0b = Odd parity 1b = Even parity
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCSPB	RW	0h	Stop bit select. Number of stop bits. 0b = One stop bit 1b = Two stop bits
10-9	UCMODEx	RW	0h	eUSCI_A mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0. 00b = UART mode 01b = Idle-line multiprocessor mode 10b = Address-bit multiprocessor mode 11b = UART mode with automatic baud-rate detection
8	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSSELx	RW	0h	eUSCI_A clock source select. These bits select the BRCLK source clock. 00b = UCLK 01b = ACLK 10b = SMCLK 11b = SMCLK

USCI_A0: Initialisierung

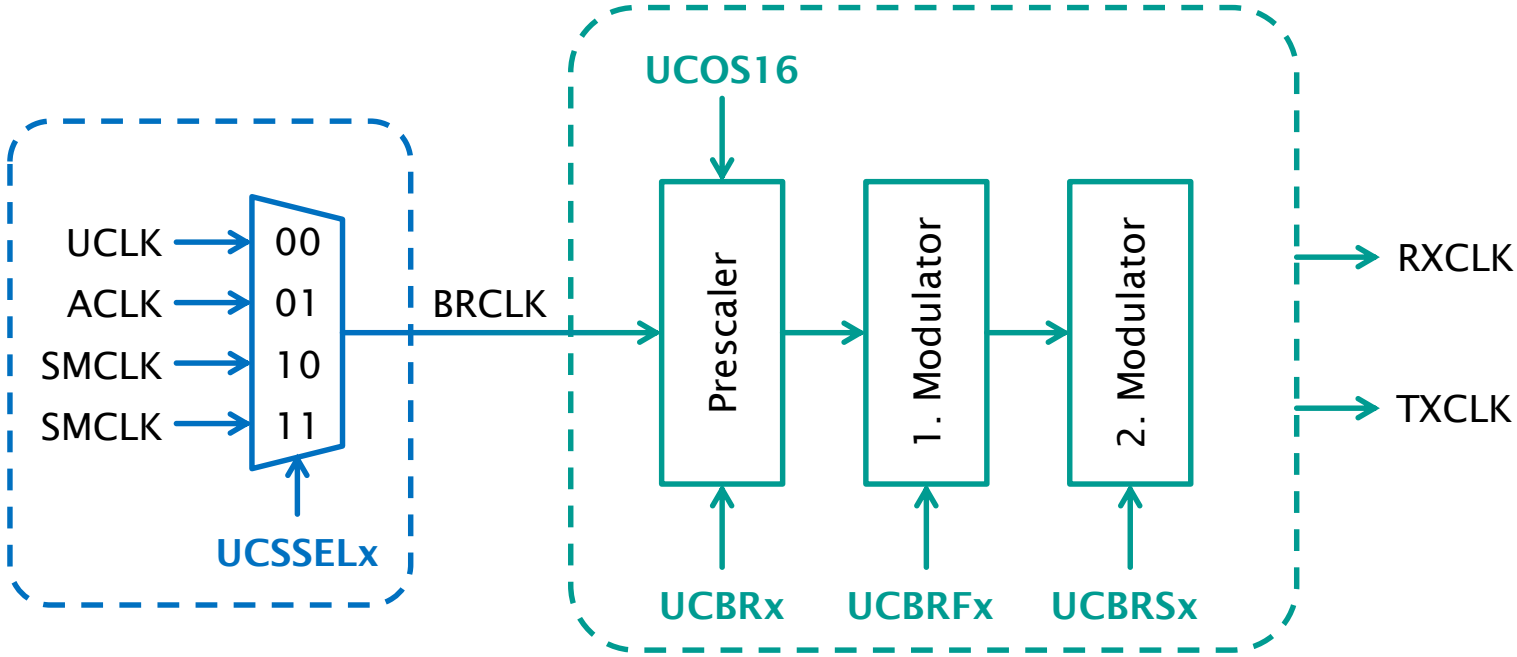
- Auszug auf einer Initialisierungssequenz (noch ohne Baudrate)

Beispiel 9

```
SETBIT(UCA0CTLW0, UCSWRST);  // UCA0 software reset

UCA0CTLW1 = 0x0002;          // deglitch time approximately 100 ns

. . .

UCA0CTLW0 = UCPEN            // enable parity
          | UCPAR            // even parity
          | 0                // LSB first
          | 0                // 8-bit-data
          | 0                // one stop bit
          | UCMODE_0         // UART mode
          | 0                // Asynchronous mode
          | UCSSEL__ACLK     // select clock source: ACLK 614.4 kHz
          | UCRXEIE          // error interrupt enable
          | UCBRKIE          // break interrupt enable
          | 0;               // release the UCA0 for operation

UCA0IE    = 0                // Transmit Complete Interrupt Disable
          | 0                // Start Bit Interrupt Disable
          | 0                // Transmit Interrupt Disable
          | UCRXIE;          // Receive Interrupt Enable
```

USCI_A0: Initialisierung

- Einstellung der Übertragungsgeschwindigkeit (Baudrate)



USCI_A0: Initialisierung

- Einstellung der Übertragungsgeschwindigkeit (Baudrate)
  - mit/ohne 16-fache Überabtastung (UCOS16)
  - Frequenzteiler im Baudratengenerator (UCBRx)
  - Feineinstellung in zwei Modulatorstufen (UCBRFx, UCBRSx)
  - $N = f_{BRCLK}/\text{Baudrate}$

	$N \geq 32$	$N < 32$
UCOS16	$OS_{16} = 1$	$OS_{16} = 0$
UCBRx	$P = N / 16$ $P_t = \text{trunc}(P)$	$P = 0$ $P_t = \text{trunc}(P)$
UCBRFx	$Q = (P - P_t) \times 16$ $Q_t = \text{trunc}(Q)$	$Q = P_t \times 16$ $Q_t = \text{trunc}(Q)$
UCBRSx	$idx = \lceil (Q - Q_t) \times 36 \rceil$ $R = \text{table}[\, idx \,]$	$idx = \lceil (Q - Q_t) \times 36 \rceil$ $R = \text{table}[\, idx \,]$

- Register für Einstellung der Übertragungsgeschwindigkeit

18.4.4 UCAxMCTLW Register

eUSCI_Ax Modulation Control Word Register

Figure 18-15. UCAxMCTLW Register

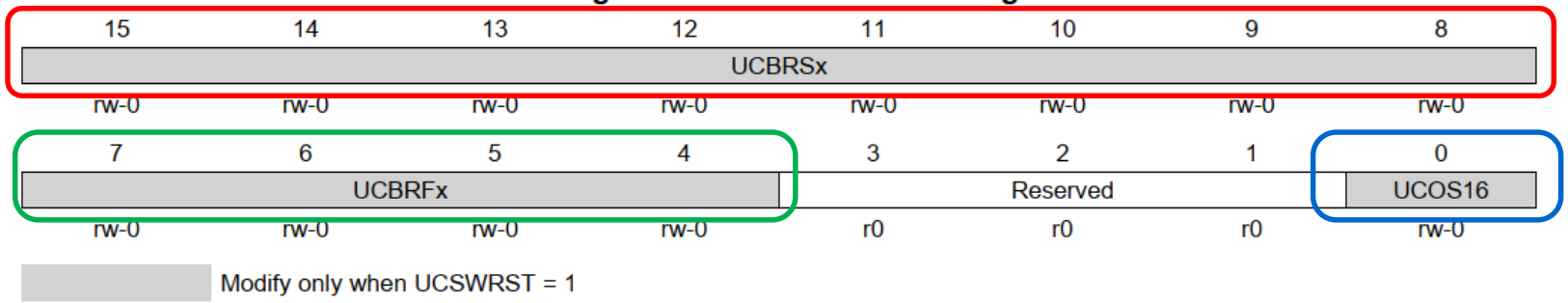


Table 18-11. UCAxMCTLW Register Description

Bit	Field	Type	Reset	Description
15-8	UCBRSx	RW	0h	Second modulation stage select. These bits hold a free modulation pattern for BITCLK.
7-4	UCBRFx	RW	0h	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. The "Oversampling Baud-Rate Generation" section shows the modulation pattern.
3-1	Reserved	R	0h	Reserved
0	UCOS16	RW	0h	Oversampling mode enabled 0b = Disabled 1b = Enabled

- Fall 1: Einstellung der Übertragungsgeschwindigkeit mit folgenden Parametern:
  - Baudrate: 9.6 kb/s
  - f_{BRCLK} = 614.4 kHz
  - N = f_{BRCLK}/Baudrate = 64.0

$N \geq 32$	Register
$OS_{16} = 1$	UCOS16 := 1
$P = N / 16$ $P_t = trunc(P)$	4.0 UCBRx := 4
$Q = (P - P_t) \times 16$ $Q_t = trunc(Q)$	0.0 UCBRFx := 0
$idx = \lceil (Q - Q_t) \times 36 \rceil$ R = table[idx]	0 UCBRSx := 0

USCI_A0: Initialisierung

Beispiel 9

- Programmcode zum Fall 1

```
SETBIT(UCA0CTLW0, UCSWRST);  // UCA0 software reset
UCA0CTLW1 = 0x0002;          // deglitch time approximately 100 ns

UCA0BRW   = 4;               // set clock prescaler for 9600 baud
UCA0MCTLW = 0x00 << 8        // second modulation stage
          | 0x00             // first modulation stage
          | UCOS16;          // enable 16 times oversampling

UCA0CTLW0 = UCPEN            // enable parity
          | UCPAR            // even parity
          | 0                // LSB first
          | 0                // 8-bit-data
          | 0                // one stop bit
          | UCMODE_0         // UART mode
          | 0                // Asynchronous mode
          | UCSSEL__ACLK     // select clock source: ACLK
          | UCRXEIE          // error interrupt enable
          | UCBRKIE          // break interrupt enable
          | 0;               // release the UCA0 for operation

UCA0IE    = 0                // Transmit Complete Interrupt Disable
          | 0                // Start Bit Interrupt Disable
          | 0                // Transmit Interrupt Disable
          | UCRXIE;          // Receive Interrupt Enable
```

- Fall 2: Einstellung der Übertragungsgeschwindigkeit mit folgenden Parametern:
  - Baudrate: 19.2 kb/s
  - f_{BRCLK} = 1.0 MHz
  - N = f_{BRCLK}/Baudrate = 52.0833

$N \geq 32$	Register
$OS_{16} = 1$	UCOS16 := 1
$P = N / 16$ $P_t = trunc(P)$	3.2552 UCBRx := 3
$Q = (P - P_t) \times 16$ $Q_t = trunc(Q)$	4.0833 UCBRFx := 4
$idx = \lceil (Q - Q_t) \times 36 \rceil$ $R = table[idx]$	3 UCBRSx := 0x04

Table 18-4. UCBRSx Settings

Fractional Portion of N	UCBRSx[1]
0.0000	0x00
0.0529	0x01
0.0715	0x02
0.0835	0x04
0.1001	0x08
0.1252	0x10
0.1430	0x20
0.1670	0x11
0.2147	0x21
0.2224	0x22

USCI_A0: Initialisierung

- Fall 3: Einstellung der Übertragungsgeschwindigkeit mit folgenden Parametern:
  - Baudrate: 38.4 kb/s
  - f_{BRCLK} = 8.0 MHz
  - N = f_{BRCLK}/Baudrate = 208.3333

N ≥ 32	Register
OS_{16} = 1	UCOS16 := 1
P = N / 16 P_t = trunc(P)	13.0208 UCBRx := 13
Q = (P – P_t) × 16 Q_t = trunc(Q)	0.3333 UCBRFx := 0
idx = ⌈(Q – Q_t) × 36⌉ R = table[idx]	12 UCBRSx := 0x49

Table 18-4. UCBRSx Settings

Fractional Portion of N	UCBRSx[1]
0.0000	0x00
0.0529	0x01
0.0715	0x02
0.0835	0x04
0.1001	0x08
0.1252	0x10
0.1430	0x20
0.1670	0x11
0.2147	0x21
0.2224	0x22
0.2503	0x44
0.3000	0x25
0.3335	0x49
0.3575	0x4A
0.3753	0x52
0.4003	0x92

USCI_A0: Initialisierung

- Fall 4: Einstellung der Übertragungsgeschwindigkeit mit folgenden Parametern:
  - Baudrate: 9.6 kb/s
  - f_{BRCLK} = 250.0 kHz
  - N = f_{BRCLK}/Baudrate = 26.0417

N < 32	Register
OS_{16} = 0	UCOS16 := 0
P = N P_t = trunc(P)	26.0417 UCBRx := 26
Q = (P – P_t) Q_t = trunc(Q)	0.0417 UCBRFx := 0
idx = trunc(Q_t × 36) R = table[idx]	1 UCBRSx := 0x01

Table 18-4. UCBRSx Settings

Fractional Portion of N	UCBRSx (1)
0.0000	0x00
0.0529	0x01
0.0715	0x02
0.0835	0x04
0.1001	0x08
0.1252	0x10

USCI_A0: Initialisierung

Bit	Feld	Reset	Beschreibung
6	UCFE	0	Framing error flag 0b = No error 1b = Character received with low stop bit
5	UCOE	0	Overrun error flag. This bit is set when a character is transferred into UCAxRXBUF before the previous character was read. 0b = No error 1b = Overrun error occurred.
4	UCPE	0	Parity error flag. 0b = No error 1b = Character received with parity error
3	UCBRK	0	Break detect flag. 0b = No break condition 1b = Break condition occurred.
2	UCRXERR	0	Receive error flag. This bit indicates a character was received with one or more errors. When UCRXERR = 1, on or more error flags, UCFE, UCPE, or UCOE is also set. 0b = No receive errors detected 1b = Receive error detected

- Übertragungsfehler
  - Break wird erkannt, wenn alle Datenbits, das Stoppbit und das Paritätsbit auf =0 sind.
  - Rahmenfehler (Framing Error): Startbit erkannt, Stoppbit nicht vorhanden.
  - Overrun Error: ein empfangenes Byte wurde nicht rechtzeitig abgeholt und durch ein neues überschrieben.
  - Paritätsfehler: Anzahl der Nullen/ Einsen im empfangenen Byte stimmt nicht.
  - Error-Flags werden automatisch gelöscht, wenn das Register UCAxRXBUF gelesen wird.

USCI_A0: Initialisierung

- Interrupt Enable Flags im UCAxIE Register

Bit	Feld	Reset	Beschreibung
3	UCTXCPTIE	0	Transmit complete interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
2	UCSTTIE	0	Start bit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
1	UCTXIE	0	Transmit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE	0	Receive interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

- Interrupt-Vector-Werte im UCAxIV Register

Bit	Feld	Reset	Beschreibung
15-0	UCIVx	0	00h = No interrupt pending 02h = Interrupt Source: Receive buffer full; Interrupt Flag: UCRXIFG; Highest Interrupt Priority 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG 06h = Interrupt Source: Start bit received; Interrupt Flag: UCSTTIFG 08h = Interrupt Source: Transmit complete; Interrupt Flag: UCTXCPTIFG; Lowest Interrupt Priority

- Interrupt Flags im UCAxIFG Register

Bit	Feld	Reset	Beschreibung
3	UCTXCPTIFG	0	Transmit ready interrupt enable. UCTXRDYIFG is set when the entire byte in the internal shift register got shifted out and UCAxTXBUF is empty. 0b = No interrupt pending 1b = Interrupt pending
2	UCSTTIFG	0	Start bit interrupt flag. UCSTTIFG is set after a Start bit was received 0b = No interrupt pending 1b = Interrupt pending
1	UCTXIFG	1	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	0	Receive interrupt flag. UCRXIFG interrupt flag is set each time a character is received a complete character and loaded into UCAxRXBUF 0b = No interrupt pending 1b = Interrupt pending

- Das Flag UCRXIFG und ggf. Error-Flags werden automatisch zurück-gesetzt, wenn das Register UCAxRXBUF gelesen wird.

USCI_A0: Interrupt Service Routine


```
switch (__even_in_range(UCA0IV, 0x08)) {
   case 0x00:  // No interrupts
      ...
      break;
   case 0x02:  // Receive buffer full
      ...
      break;
   case 0x04:  // Transmit buffer empty
      ...
      break;
   case 0x06:  // Start bit received
      ...
      break;
   case 0x08:  // Transmit complete
      ...
      break;
}
```

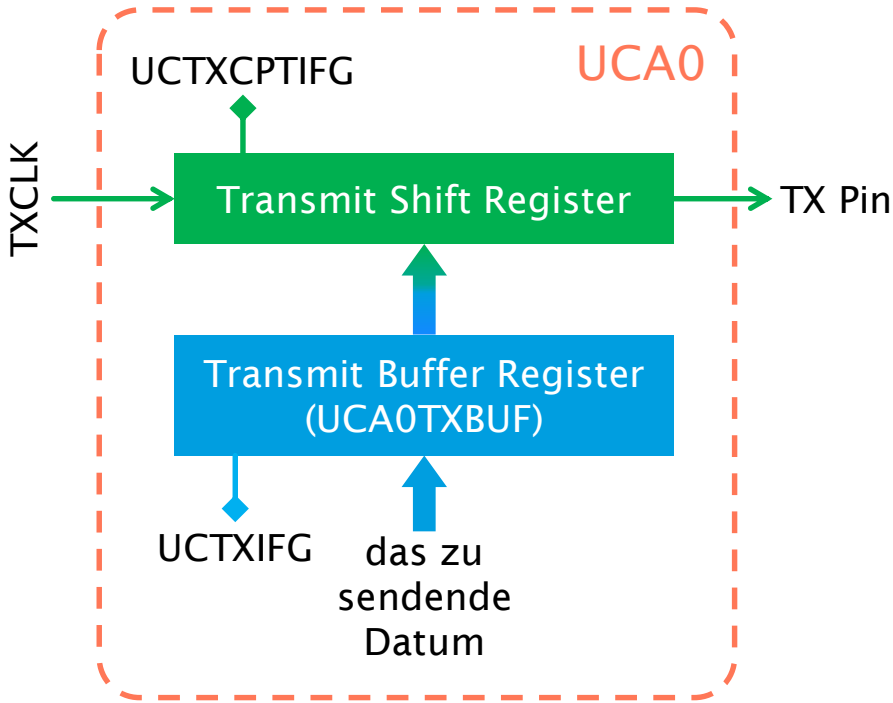
- Die genaue Bestimmung der Interrupt-Quelle erfolgt in einem switch-case-Statement durch die Auswertung des Interrupt-Vektors UCAxIV.
- Damit der C-Compiler den switch-case-Statement nicht als Folge von Vergleichs- und Sprungbefehlen sondern als Tabelle umsetzten kann, muss das Makro __even_in_range() benutzt werden.

USCI_A0: Echo-Funktion per Interrupt

Beispiel 9

```
#pragma vector = USCI_A0_VECTOR
__interrupt Void UCA0_ISR(Void) {
   switch (__even_in_range(UCA0IV, 0x08)) {
      case 0x00:  // Vector 0: no interrupts
         break;

      case 0x02:  // Vector 2: Receive buffer full
         if (TSTBIT(UCA0STATW, UCBRK | UCRXERR)) {
            Char ch = UCA0RXBUF; // dummy read
            return;
         }
         UCA0TXBUF = UCA0RXBUF;
         break;

      case 0x04:  // Vector 4: Transmit buffer empty
         break;
      case 0x06:  // Vector 6: Start bit received
         break;
      case 0x08:  // Vector 8: Transmit complete
         break;
   }
}
```

USCI_A0: Register des Transmitters



- UCA0-Transmitter verfügt über zwei Flags, mit denen er interne Zustände anzeigen kann:
  - UCTXCPTIFG (Transmit complete) zeigt an, dass der gesamte Sendevorgang abgeschlossen ist, d.h. das Transmit-Buffer-Register UCA0TXBUF und das Transmit-Shift-Register sind leer.
  - UCTXIFG (Transmit buffer empty) zeigt an, dass das Transmit-Buffer-Register UCA0TXBUF leer ist.

USCI_A0: Ausgabe einer Zeichenkette

- Die Ausgabe mit Standard-I/O-Funktionen (z.B. printf()) ist auf einem ereignisorientieren System entweder gar nicht möglich oder nur bedingt ausführbar.
- Sollte dennoch die Ausgabe mit Hilfe einer Standard-I/O-Funktion über eine Kommunikationsschnittstelle abgewickelt werden, so muss die Ausgabefunktion entsprechend angepasst werden.

```
UCA0_printf("HalloWorld!\r\n";);
  ...

GLOBAL Int UCA0_printf(const Char * str) {
   if (str EQ NULL) {
      return -1;
   }
   while (*str NE '\0') {
      UCA0TXBUF = *str++;
      while (TSTBIT(UCA0IFG, UCTXIFG) EQ 0);
   }

   while (TSTBIT(UCA0IFG, UCTXCPTIFG) EQ 0);
   return 0;
}
```

Diese Schleife verursacht eine 100%-ige CPU-Last. Sie wird beendet, wenn das Register UCA0TXBUF wieder leer ist.

Hier warten wir noch, bis das Shift-Register leer ist, und somit die Übertragung tatsächlich abgeschlossen ist.

USCI_A0: Ausgabe einer Zeichenkette

- Eine Interrupt-gesteuerte Ausgabefunktion erzeugt deutlich geringe CPU-Last und sie blockiert andere Aufgaben nicht (kooperatives Multitasking)
- Die Ausgabefunktion selbst bereitet den String für die Ausgabe vor und initiert die Übertragung im UART

```
LOCAL const Char * ptr;

GLOBAL Int UCA0_printf(const Char * str) {
   if (str EQ NULL) {
      return -1;
   }

   ptr = str;
   SETBIT(UCA0IFG, UCTXIFG); // set UCTXIFG
   SETBIT(UCA0IE,  UCTXIE);  // enable transmit interrupt
   return 0;
}
```

Der Pointer dient als Übergabeschnittstelle zwischen der Ausgabefunktion und der ISR

Diese Befehlsfolge führt zu einem indirekten ISR-Aufruf!

Beispiel 9.1

USCI_A0: Ausgabe einer Zeichenkette

- Das Versenden der Daten findet dann ausschließlich in der Interrupt-Service-Routine von UART statt.
- Die Übertragung wird beendet, sobald im String das Null-Byte gefunden wird. Der Transmitter wird angehalten, indem sein Interrupt-Enable-Flag (UCTXIE) gelöscht wird.

```
__interrupt Void UCA0_ISR(Void) {
   switch (__even_in_range(UCA0IV, 0x08)) {
      . . .
      case 0x04:  // Vector 4: Transmit buffer empty
         if (*ptr NE '\0') {
            UCA0TXBUF = *ptr++;
            return;
         }
         CLRBIT(UCA0IE, UCTXIE);    // transmit interrupt disable
         Char ch = UCA0RXBUF;       // dummy read
         SETBIT(UCA0IE, UCRXIE);    // receive interrupt enable
         break;
      . . .
}
```

Beispiel 9.1

USCI_A0: Ausgabe einer Zeichenkette

Beispiel 9.1

```
#pragma vector = USCI_A0_VECTOR
__interrupt Void UCA0_ISR(Void) {
   switch (__even_in_range(UCA0IV, 0x04)) {

      case 0x02:  // Vector 2: Receive buffer full
         if (TSTBIT(UCA0STATW, UCBRK | UCRXERR)) {
            Char ch = UCA0RXBUF; // dummy read
            return;
         }
         if (UCA0RXBUF EQ '?') {
            Event_set(EVENT_PRN);
            CLRBIT(UCA0IE, UCRXIE);          // receive interrupt disable
            __low_power_mode_off_on_exit(); // restore Active Mode on return
         }
         break;

      case 0x04:  // Vector 4: Transmit buffer empty
         if (*ptr NE '\0') {
            UCA0TXBUF = *ptr++;
            return;
         }
         CLRBIT(UCA0IE, UCTXIE);    // transmit interrupt disable
         Char ch = UCA0RXBUF;       // dummy read
         SETBIT(UCA0IE, UCRXIE);    // receive interrupt enable
         break;
   }
}
```

USCI_A0: Ausgabe einer Zeichenkette

Beispiel 9.1

```
LOCAL const char msg[] = "HalloWorld!\r\n";

GLOBAL Void main(Void) {
   CS_init();     // set up Clock System
   GPIO_init();   // set up Ports
   Event_init();
   UCA0_init();

   while(TRUE) {
      Event_wait();
      if (Event_tst(EVENT_PRN)) {
         Event_clr(EVENT_PRN);
         UCA0_printf(msg);
      }
   }
}
```