

NeoPixel (UART + DMA)

Donnerstag, 27. November 2025

14:31

Serielle Bussysteme

Allgemeine Fakten

Die Kommunikation zwischen zwei Systemen bedingt, dass diese ein identisches Verständnis von der Zeit haben, so dass der Sender mit einem vorgegebenen Intervall Daten ausgeben/senden und der Empfänger dieses mit dem identischen Intervall abtasten/empfangen kann.

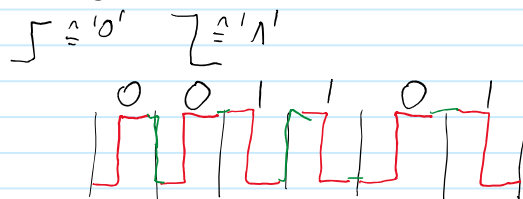
Zur Darstellung gibt es zwei möglich Verfahren:

Synchrone Datenübertragung:

Parallel zum Datensignal gibt es ein Taktsignal, welches dem Sender und Empfänger mitteilt, wann Daten auf den Datenbus zu schreiben/zu lesen sind. Dieses Taktsignal kann gleichermaßen als eigenständige Leitung aber auch im Datensignal kodiert sein (Leitungscode).

Beispiel Manchesterkodierung (Ethernet 100MBit)

Bei der Manchesterkodierung entspricht eine Null-Eins Folge einer logisch 0 und eine Eins-Null Folge einer logischen 1



Asynchrone Datenübertragung:

Hier gibt es kein separates Taktsignal, stattdessen senden/empfangen die Teilnehmer mit ihrer eigenen 'Uhr' und nutzen den Fakt aus, dass für kleine Zeitabschnitte der Gangunterschied tolerierbar ist.

Zur Synchronisation der Uhren wird ein Startsignal ausgesendet (bei UART die fallende Flanke des Startbits), zu welchen beide Teilnehmer ihre Uhr auf 0 setzen. Die max. Paketlänge ergibt sich aus der tolerierbaren Taktabweichung.

Sender und Empfänger müssen 'händisch' auf die identische Taktrate eingestellt werden. Da die Prozessorfrequenz und die Baudrate nicht ganzzahlige Vielfache sind, sind allein deswegen Taktabweichungen vorhanden, so dass die Datenlänge zumeist sehr kurz ist.

UART Universal Asynchron Receiver/Transmitter

In Verbindung mit UART sind div. weitere Begriffe verbunden

RS232

Chip für USB zu UART Konvertierung --> Das wäre der FT232 Baustein von FTDI, der hierzu gerne genutzt wird und im Namen den Begriff 232 trägt. Dieser beinhaltet einen USB nach UART Konverter, der Ausgabepegel ist jedoch TTL

RS Recommend Standard 232

Definiert Stecker DB9/DB25

Basiert auf UART Datenübertragung

Definiert die Datenübertragung über Spannungen

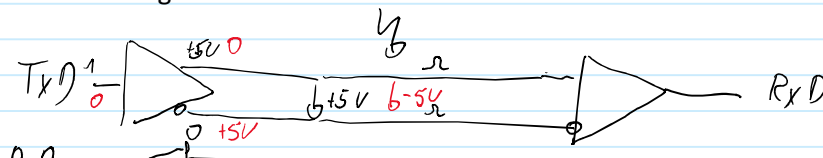
'0' -> +3V...+15V

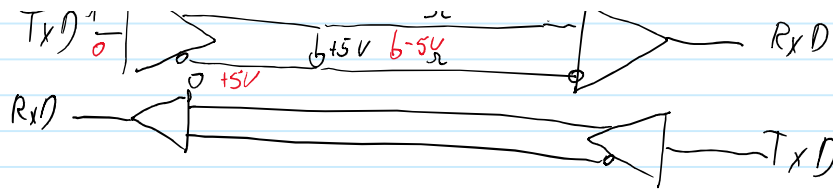
'1' -> -3V...-15V

RS422

Basiert auf UART Datenübertragung

Die Sende und Empfangsleitung ist eine zweiadrigte Datenleitung, auf welcher die Daten Differentiell gesendet werden

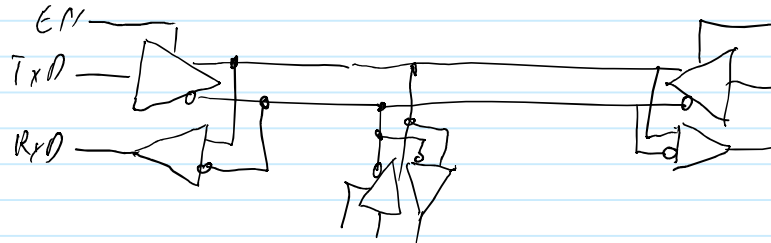




Ähnlich zu RS232, nur dass anstatt der unipolaren Spannung nun eine Differentielle Spannung übertragen wird.

Identisch zu RS232 ist eine Adressierung der Teilnehmer nicht notwendig (Punkt zu Punkt Übertragung)

RS485



Ähnlich zu RS422, nur dass der Transceiver deaktiviert werden kann und somit mehrere Teilnehmer angeschlossen werden können.

HalbDuplex Übertragung!

--> Im NXT Baustein vorhanden an SensorPort 4, bei welcher der Treiber mit PA7 aktiviert/deaktiviert wird

UART

gemeinsame Baudrate (Taktrate)

1200, 2400, 4800, 9600 Baud, 115200Baud

Rahmenformat

- Startbit (fest 1 Bit, welches 0 ist)
- Datenbits (5,6,7,8,9) Datenbits
- Parity (Optional, Prüfsumme, so dass der Empfänger prüfen könnte, ob die Datenbits korrekt übertragen wurden)

☐ Not Parity

☐ Even Parity: Drive to 0 when the number of 1 in data ist even

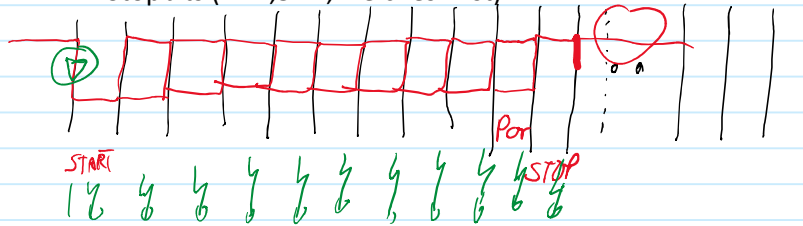
0 1 0 1 1 0 0 0 1
0 1 0 1 1 0 0 1 0

Odd Parity: Gegenteil zu Even

Mark Parity: Paritätsbit ist immer 1

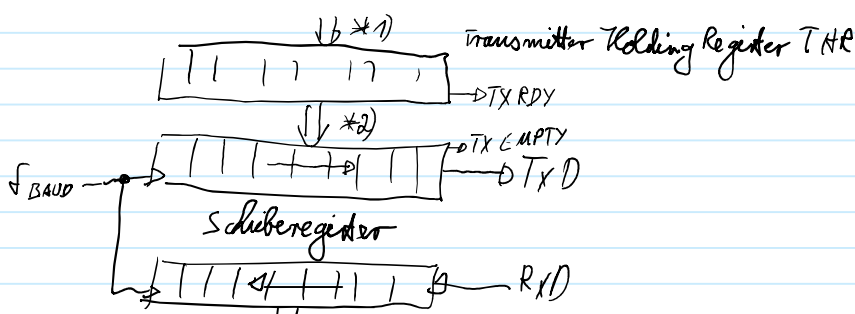
Space Parity: Paritätsbit ist immer 0

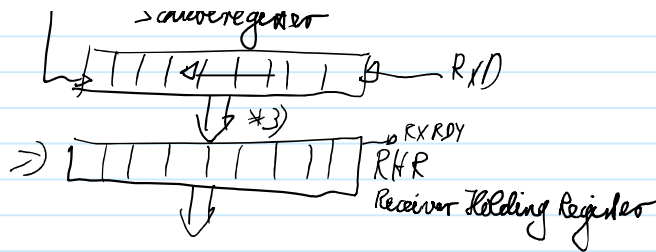
- Stopbits (1 1,5 2, welches 1 ist)



UART gibt nur dieses Rahmenformat vor. Welche Bedeutung die übertragenen Bytes hat, hängt vom nächst höheren Protokoll ab.

Der prinzipielle Aufbau einer seriellen Schnittstelle sieht wie folgt aus





- *1) In das TransmitterHoldingRegister THR darf nur geschrieben werden, wenn dieses Leer ist (erkennbar an TXRDY), andernfalls würde der alte Wert überschrieben und nicht gesendet werden. Wenn beim Schreiben in das THR Register das Schieberegister ebenfalls leer ist, wird der Inhalt sofort in das Schieberegister übertragen und das THR Register könnte den nachfolgend zu sendenden Wert aufnehmen.
- *2) Nach senden des Stopbits prüft das Schieberegister, ob ein neuer Wert im THR steht. Wenn ja, wird dessen Inhalt in das Schieberegister zum Senden übertragen. Wenn nein, wird der Sendeprozess beendet und ein Idle Signal (Pegel = 1) gesendet.
- *3) Wenn das Stopbit empfangen wurde und die Prüfsumme (sofern vorhanden) korrekt ist, wird der Inhalt aus dem Schieberegister in das ReceiverHoldingRegister RHR übertragen. Wenn das RHR nicht rechtzeitig gelesen wird, würde mit diesem Schreiben der zuvor empfangene Wert überschrieben/verloren gehen.

Figure 31-9. Transmitter Status

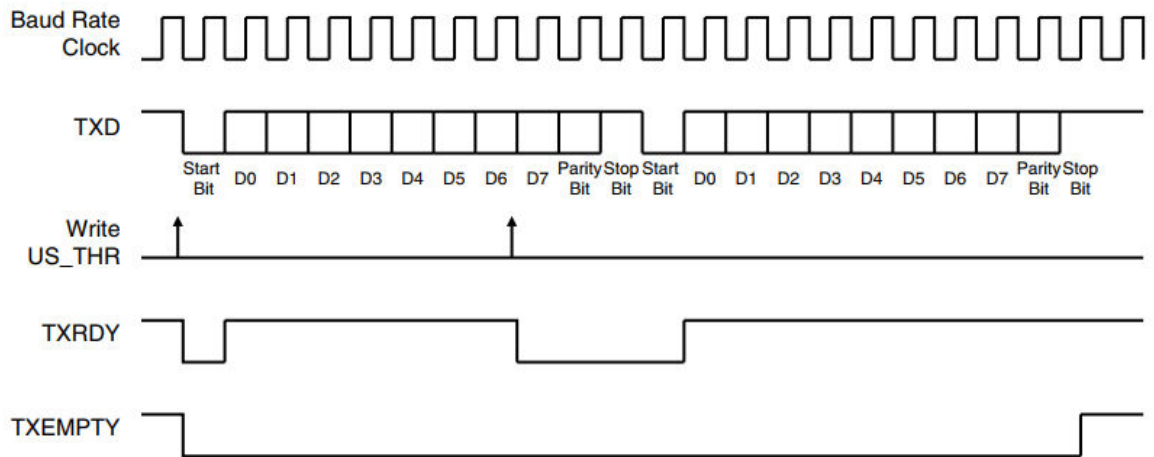
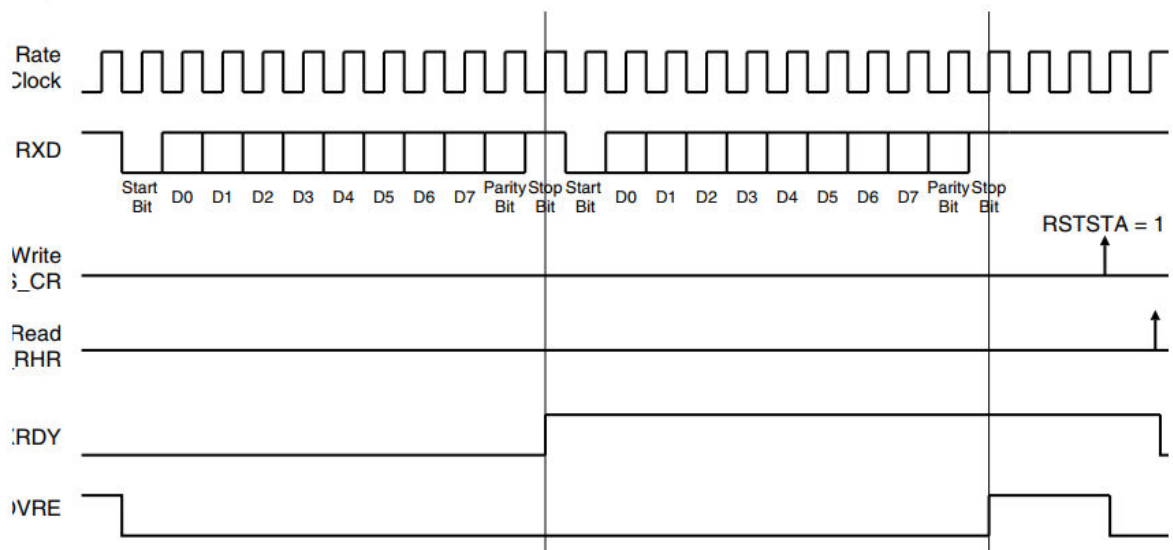


Figure 31-13. Receiver Status



LED

Leitungsband

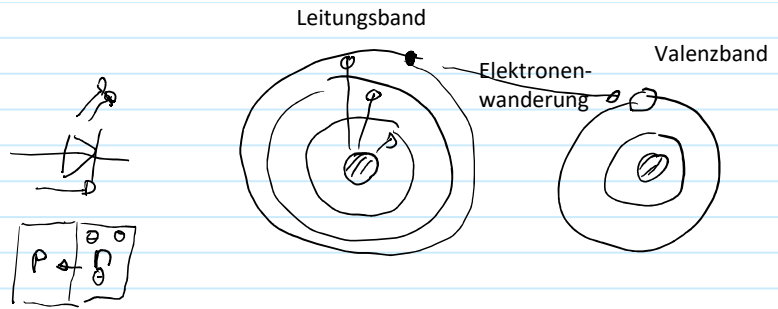


Elektronen-

Valenzband

LED

Light Emitting Diode



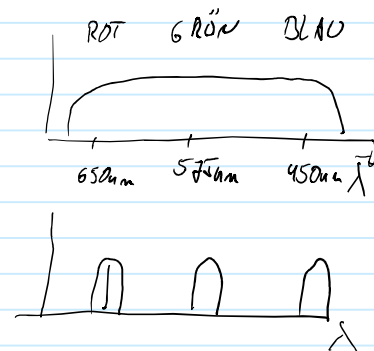
Verunreinigt mit einem Material mit 3 Elektronen in der Außenschale

Verunreinigt mit einem Material mit 5 Elektronen in der Außenschale

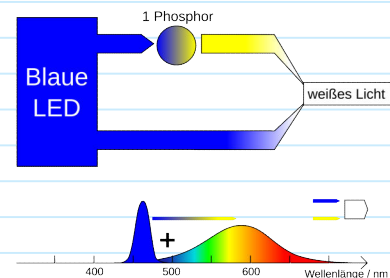
Bei der 'Elektronenwanderung' eines Elektrons von dem weiter entfernten Leitungsband in das näher zum Atomkern gelegene Valenzband wird Energie freigesetzt, welche sich in Form von Licht (einer Wellenlänge) und Wärme äußert!

Weißes Licht

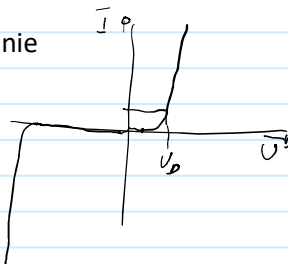
- Mischung aus allen Grundfarben



- Mischung aus Blau und einer durch die Phosphorschicht erzeugten Farbbereich



Strom-/Spannungskennlinie



Die Durchlassspannung U_D , ab welcher Strom die LED fließt ist vom Halbleitermaterial und von der Farbe abhängig

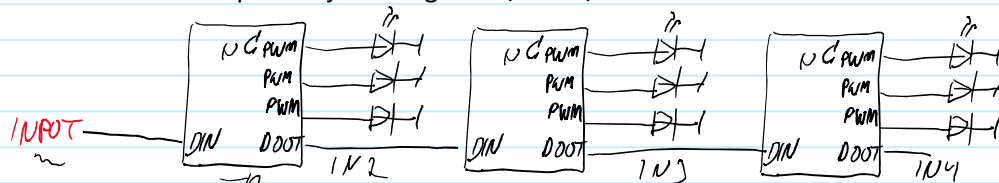
Rote-LED: 1,6..2,2V

Gelb/Grüne-LED: 1,9..2,5V

Blaue-LED: 2,7..3,5V

NeoPixel

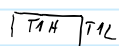
besteht aus einem μC und je einer grünen/roten/blauen LED



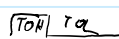
je LED 256-Helligkeitsstufen möglich, LED's werden über PWM angesteuert!

Konfiguriert wird das NeoPixel über eine serielle Schnittstelle

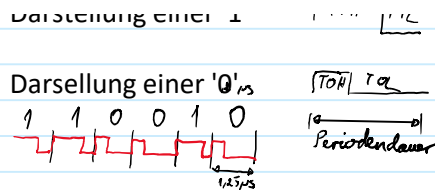
Darstellung einer '1'



Darstellung einer '0'



Bit	Impulsdauer	Pausendauer
'1'	T1H=0,8 μ s	T1L=0,45 μ s
'0'	T0H=0,4 μ s	T0L=0,85 μ s



'1'	T1H=0,8µs	T1L=0,45µs
'0'	T0H=0,4µs	T0L=0,85µs

Alle Zeiten $\pm 0,15\mu s$

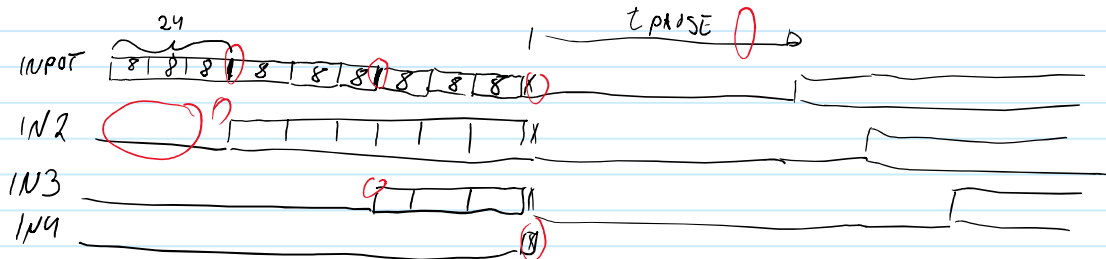
Die Daten zur Farbdarstellung werden im Rohformat übertragen:

G7 G6 G5 G4 G3 G2 G1 G0 R7 R6 R5 R4 R3 R2 R1 R0 B7 B6 B5 B4 B3 B2 B1 B0

erste zu übertragendes Bit

letztes zu übertragendes Bit

Im 'zurückgesetzten' Zustand konsumiert die erste LED die ersten 24-Bit, d.h. die empfangen Daten werden nicht an den Ausgang weitergereicht. Mit dem Empfang des 25. Bit reicht das NeoPixel diese und die folgenden Bits an den Datenausgang weiter.



Das Rücksetzen in den zurückgesetzten Zustand erfolgt durch Einhaltung eines RET Codes (Pausenzeit) $>50\mu s$. Mit Ablauf der Pausenzeit aktualisieren alle NeoPixel erst ihre LED's

Darstellung des seriellen Signals

Variante 1) PWM

Nach jeden Zählerdurchlauf muss das UpdateRegister mit dem passenden Tastverhältnis für das als übernächstes zu sendendes Bit geladen.

Bedeutet hier, spätestens nach $1,25\mu s$ muss das Register neu geladen werden

48MHz Taktfrequenz $1/48MHz$ $1,25\mu s / (1/48MHz) = 60$ Maschinensprachebefehle Zeit für Update

Variante 2) BitBang

Jedes Bit wird per Software erzeugt

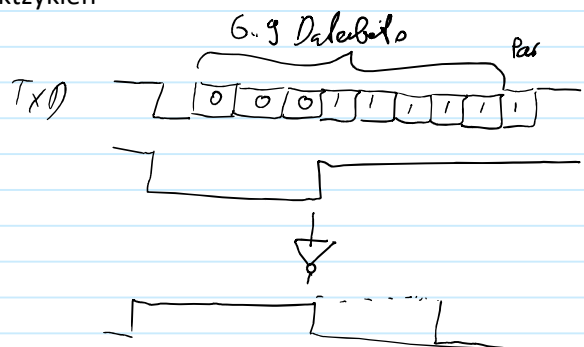
SODR=PA2;

for(); ~ 40 Taktzyklen

CODR=PA2;

for(); ~ 20 Taktzyklen

Variante 3) UART



Frage: Baudrate, Anzahl Datenbits, ggf. Parität, Länge Stopbits zur Darstellung einer '0' oder '1'

MCK=47923200 \rightarrow 48000000

Vorteiler = 1

TBIT= $1/MCK/2=41,6ns$ $1,25\mu s / 41,6ns = 30$ Bit KO, Anzahl der Bits zu groß!

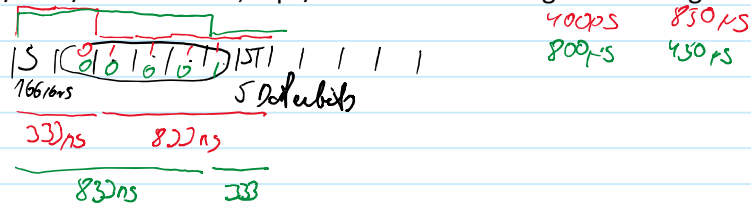
Vorteiler = 8

TBIT= $1/MCK/8=167ns$ $1,25\mu s / 167ns = 8$ Bit Mögliche Lösung

400ps 850µs

Vorteiler = 8

TBIT=1/MCK/8=167ns 1,25µs/167ns= 8 Bit Mögliche Lösung



Vorteiler 8

5 Datenbits '0' -> 0b01111 '1' -> 0b00001

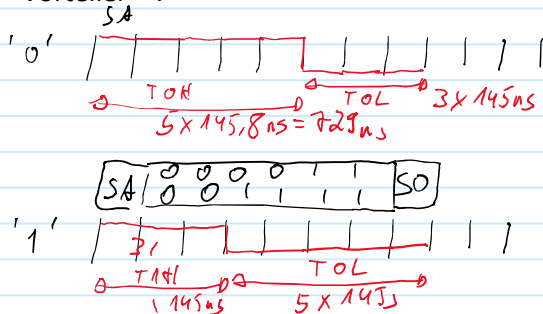
Keine Parität

1 Stopbit

Besser

TBIT=7/MCK=145,8ns 1,25µs/145,8ns=8.9 Bit Mögliche Lösung

Vorteiler = 7



```
for(int led=0;led<8;led++)
```

```
for(int lauf=0;lauf<24;lauf++)
```

```
while(TXRDY==0);
```

```
THR= 0b001111 / 0b000011
```

Datenstruktur zur Darstellung der Farbe

```
typedef union {  
    struct {uint8_t dummy; //Bits 00..07 in rgb  
            uint8_t blue; //Bits 08..15 in rgb  
            uint8_t red; //Bits 16..23 in rgb  
            uint8_t green; //Bits 24..31 in rgb  
    };  
    uint32_t rgb;  
} ws2812_composition_t;
```

```
ws2812_composition_t led;
```

```
led.blue=50;
```

```
led.red=20;
```

```
led.green=30;
```

```
for(int lauf=0; lauf<24; lauf++)
```

```
while(TXRDY==0);
```

```
//Hier hatte ich einen Fehler eingebaut
```

```
//THR = led.rgb & 0x00800000 ? 0b001111 : 0b000011
```

```
//Korrekt
```

```
THR = led.rgb & 0x80000000 ? 0b001111 : 0b000011
```

```
led.rgb<=1;
```

```
#define ROT (ws2812_composition_t){.red=255,.blue=0,.green=0}
```

```
led=ROT;
```

DMA (Direct Memory Access)

Grundidee: Datentransport wird in der Software vielseitig genutzt (memcpy() strcpy(),...)

Die Realisierung einer memcpy Funktion sieht wie folgt aus (am Beispiel von ARM32 Befehlen)

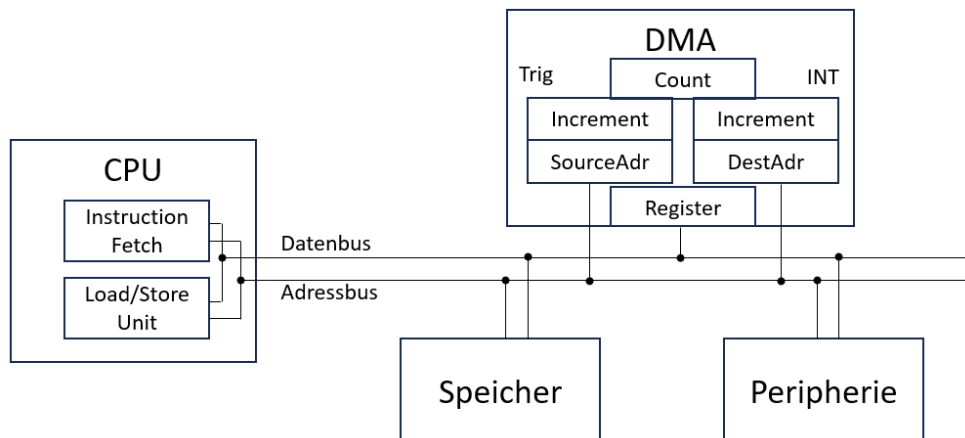
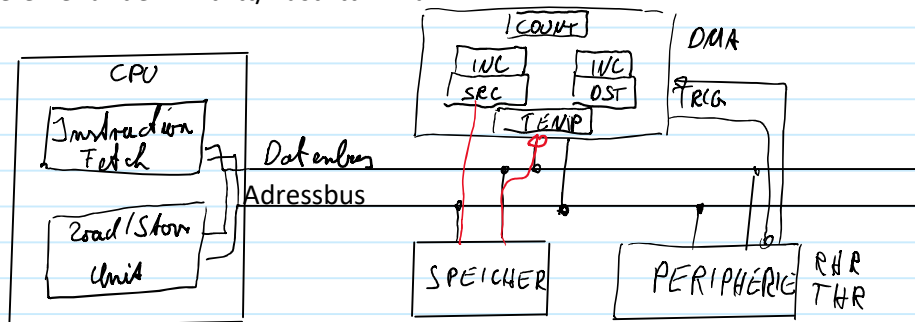
```
memcpy(dst, src, count)
//R0 beinhaltet dst
//R1 beinhaltet src
//R2 beinhaltet count
add r2,r2,r1    //r2=src+count
sub r2,r2,#1
```

goon:

```
ldr r3,[r1]++4  //r3=*src++
str r3,[r0]++4  //*dst++=r3
cmp r1,r2       //for( ; src<src_end; )
bne goon
```

Eine in Software realisierte Kopieraktion benötigt mindestens 4 Maschinenbefehle.

Wenn die Kopieraktion auf Basis eines Peripherie Interrupts angestoßen wird, so kommen hierzu noch die Befehle für den Eintritt/Austritt hinzu.



Zwei verschieden Betriebsarten

Blockdatentransfer (identisch zu memcpy())

Explicit Adressing / Two Cycle Transfer

- 1) Softwareseitigen Start
- 2) DMA legt Inhalt des SRC-Register auf den Adressbus, gibt eine Leseanforderung an das RAM, Speicher den über den Datenbus gelesenen Speicherinhalt im Temp-Register zwischen und erhöht das SRC-Register
- 3) DMA legt DST-Register auf den Adressbus, den Inhalt des TEMP-Registers auf den Datenbus und fordert den Speicher auf, den Wert vom Datenbus zu speichern und erhöht das DST-Register
- 4) Count wird dekrementiert, wenn nicht null, dann gehe zu 2). Wenn Null dann ggf. Interrupt

Einzeldatentransfer (zum Nachladen von Senderegistern , zum Sichern von Empfangsregistern)

Implicit Adressing / Single Bus Transfer

- 1) gestartet durch ein Ereignis der Peripherie (z.B. Empfang eines neuen Zeichens, Bekanntgabe, dass das Senderegister einen neuen Wert aufnehmen kann).
- 2a) DMA fordert Peripherie über dedizierte Leitung auf, den Inhalt des RHR Register auf den Datenbus zu

legen und legt ergänzend den Inhalt des DST-Registers auf den Adressbus und fordert den Speicher, den Wert zu speichern. anschließend wird das DST-Register inkrementiert

2b) DMA legt SRC-Register auf Adressbus und fordert den Speicher auf, den dazugehörigen Inhalt auf den Datenbus zu legen, und fordert die Peripherie auf, den Inhalt vom Datenbus im THR Register zu speichern. Anschließend wird das SRC-Register inkrementiert

3) Count wird dekrementiert, Wenn 0, dann Interrupt,

Anstatt 4 Maschinensprachebefehle wird der Datentransport nun in 1/2 Taktzyklen durchgeführt!

Ergänzend zum Auslösen eines Interrupts kann bei Erreichen von count = 0 die DMA-Einheit auch selbständig ihre DST/SRC/INC/COUNT-Register aus Schattenregister nachladen

Da sowohl CPU als auch DMA den Bus benutzen, ist eine Arbitrierung notwendig. Während eines DMA-Transfers kann die CPU keine Speicherzugriff (Instruction Fetch, Load/Store befehl) tätigen.

Der AT91SAM7... beinhaltet 2 USART-Bausteine. Mit den Port 4 des NXT ist der USART0 verbunden!

Zugriff auf die PDC Register wahlweise über

```
#define AT91C_US0_TCR ((AT91_REG *) 0xFFFC010C) // (PDC_US0) Transmit Counter Register

AT91_REG US_RPR; // Receive Pointer Register
AT91_REG US_RCR; // Receive Counter Register
AT91_REG US_TPR; // Transmit Pointer Register
AT91_REG US_TCR; // Transmit Counter Register
AT91_REG US_RNPR; // Receive Next Pointer Register
AT91_REG US_RNCR; // Receive Next Counter Register
AT91_REG US_TNPR; // Transmit Next Pointer Register
AT91_REG US_TNCR; // Transmit Next Counter Register
AT91_REG US_PTCR; // PDC Transfer Control Register
AT91_REG US_PTSR; // PDC Transfer Status Register
} AT91S_USART, *AT91PS_USART;
```