Programmieren von Mikrokontrollern

Gliederung

1. Mikrokontroller und Ihre Einsatzgebiete

1.1 Mikrokontroller Anwendungen

Fernbedienungen
Radios - Sendereinstellung
Fernseher - Sendereinstellung - Bildeinstellung - Videotext
Videorecorder
Kameras
Telefone
Wetter Stationen
Waschmaschinen
Trockener
Mikrowelle
Kassen
Waagen
Zugangssysteme - Schranken - Codeschlösser
Heizungsregelungen - Gebäudemanagement
Ladegeräte
Wechselrichter
Autos
Straßenbahnen
Eisenbahnen
Flugzeuge
Drucker, Scanner, Tastaturen, Festplatten, Messgeräte ...

1.2 Vorstellung einiger Mikrokontroller

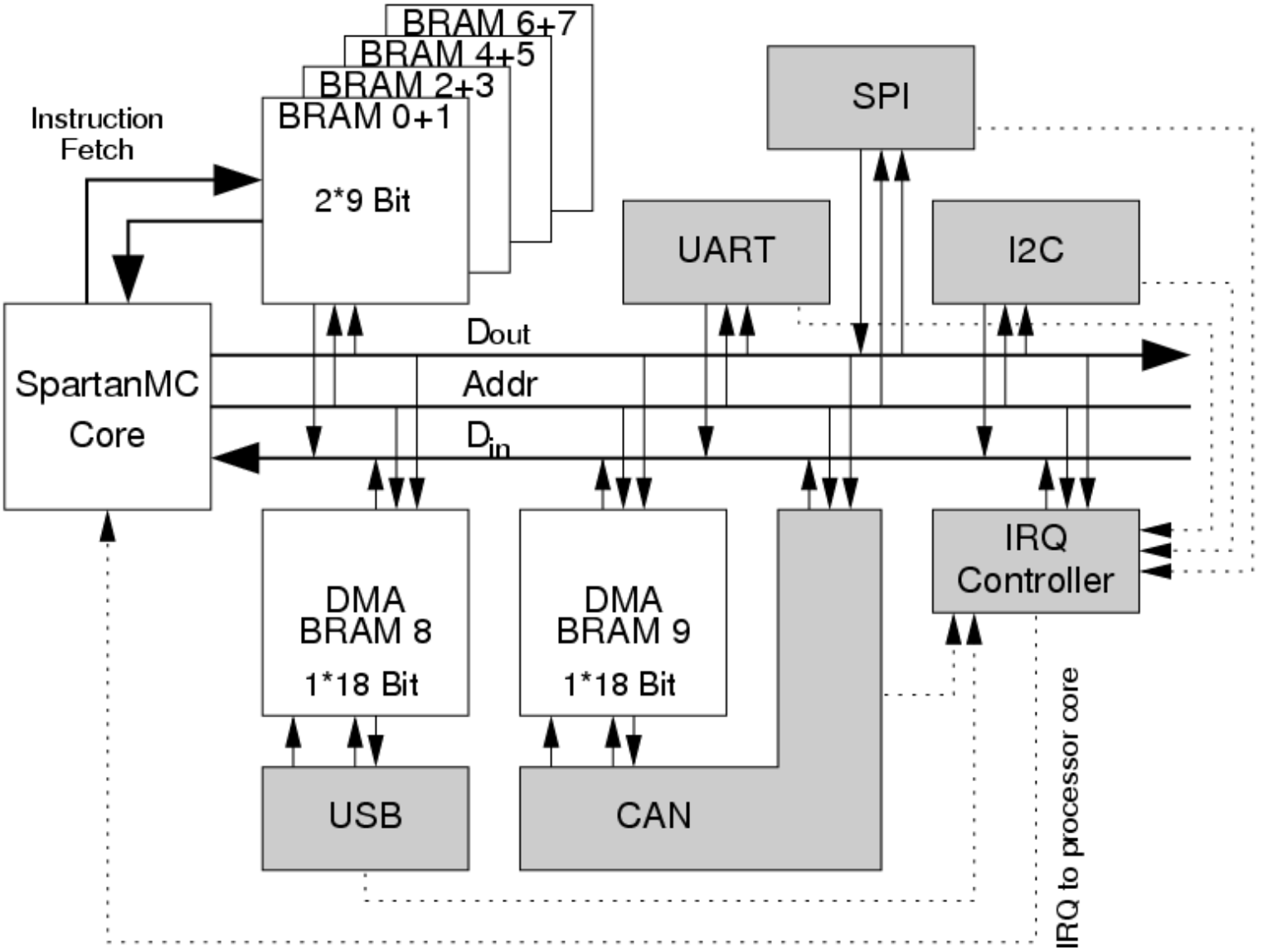
Kurz nach der Einführung der ersten Mikroprozessoren Anfang der 70ger hat man den Einsatz in universellen programmierbaren Steuerungen als wichtiges Anwendungsfeld erkannt. Daraufhin wurden in der Mitte der 70ger die ersten Mikrokontroller von Intel, und später von Motorola und Zilog angeboten.

8051 Familie	Intel
68hc11 Familie Nachfolger ist MC9S12 (hc12) Familie	Motorola
Z8	Zilog

In vielen aktuellen Steuerungen werden Mikrokontroller der folgenden Hersteller eingesetzt.

AVR	ATMega			Atmel
ARM	cortex	Raspberry Pi	odroid	ARM
PIC	http://www.microchip.com/			Microchip
MSP430				TI

2. Der Mikrokontroller SpartanMC



2. Der Mikrokontroller SpartanMC (2)

- Spartan 3e FPGAs schon ab 2$ erhältlich
- Einsatz in eingebetteten Systemen bei niedrigen Kosten möglich
- Vorhandene SoC Kits mit 8 Bit und 32 Bit Prozessoren kommen sehr schnell an Ihre Grenzen oder benötigen sehr große FPGAs
- SpartanMC soll diese Lücke schließen
  - Einsatz schon auf den kleinen FPGAs möglich
  - 18 Bit um Blockram und Multiplizierer maximal zu nutzen
  - Erstellung einer Konfiguration ohne FPGA Wissen möglich
  - große Vielfalt an Standard und Spezial Peripherie
  - mehrfache Implementierung der Peripherie möglich
  - multiprozessor Systeme sind möglich

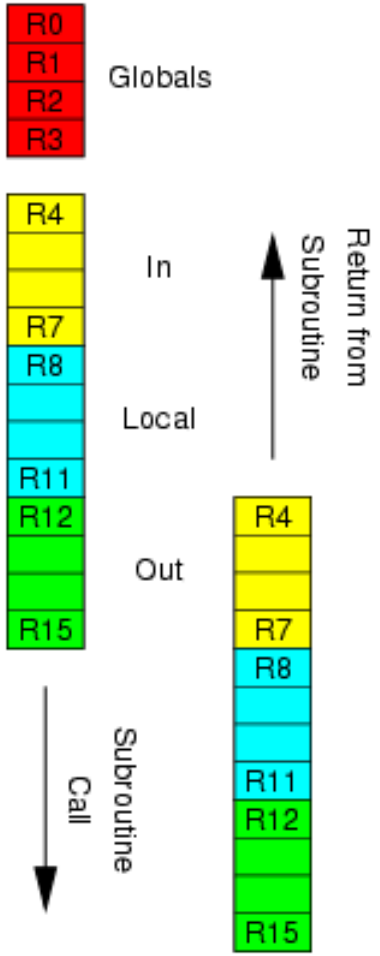
2. Der Mikrokontroller SpartanMC (3)

- Nachteile einer SoC Lösung gegenüber Mikrokontroller
  - kleine Mikrokontroller sind noch billiger als 2$
  - Sie benötigen zur Zeit noch weniger Strom (Batterieeinsatz)
  - ADC und DAC sind vorhanden
- Vorteile einer SoC Lösung
  - Konfiguration kann genau an die Aufgabe angepasst werden
  - Nachträgliche Änderungen und Anpassungen sind möglich
  - unterschiedliche Signalpegel an den Interfaces möglich
  - Skalierung der Leistung durch Wahl des FPGA
  - Leistung steigt mit jeder neuen FPGA Generation
  - keine unterschiedlichen Mikrokontroller in einem Produkt notwendig (in Fernsehern werden viele unterschiedliche Mikrokontroller eingesetzt)

2. Der Mikrokontroller SpartanMC (4)

- Der SpartanMC ist als 18 Bit RISC implementiert
- Die schnellen Blockrams werden für die Register und den Speicher eingesetzt
- Kein Cache Speicher notwendig
- Registerfenster um Registerspeicher voll zu nutzen
- Im Speicher wird ein Port zum lesen der Befehle und das zweite für den Datentransfer eingesetzt (Harvard Architektur)
- Befehle können nur auf 18 Bit Adressen stehen
- Datenzugriff auf obere und untere 9 Bit von den 18 Bit möglich
- Daher ist Befehlsadresse*2 gleich Datenadresse
- 18 Bit Datenzugriff immer nur auf gerade Adressen möglich

2.1 Die Register und der Speicher



- SpartanMC kann 16 Register direkt ansprechen
- r0 bis r3 global immer erreichbar
- r4 bis r15 können bei einem Unterprogramm aufruf oder Interrupt mit einem Offset von 8 umgeschaltet werden
- Rücksprung schaltet das Registerfenster wieder um eine Ebene zurück
- In den unteren 7 Bit des SFR_STATUS Register steht die Nummer des Registerfenster
- 16 + 126*8 = 1024 Register realisiert
- Im Hauptprogramm mit r4 bis r11 arbeiten

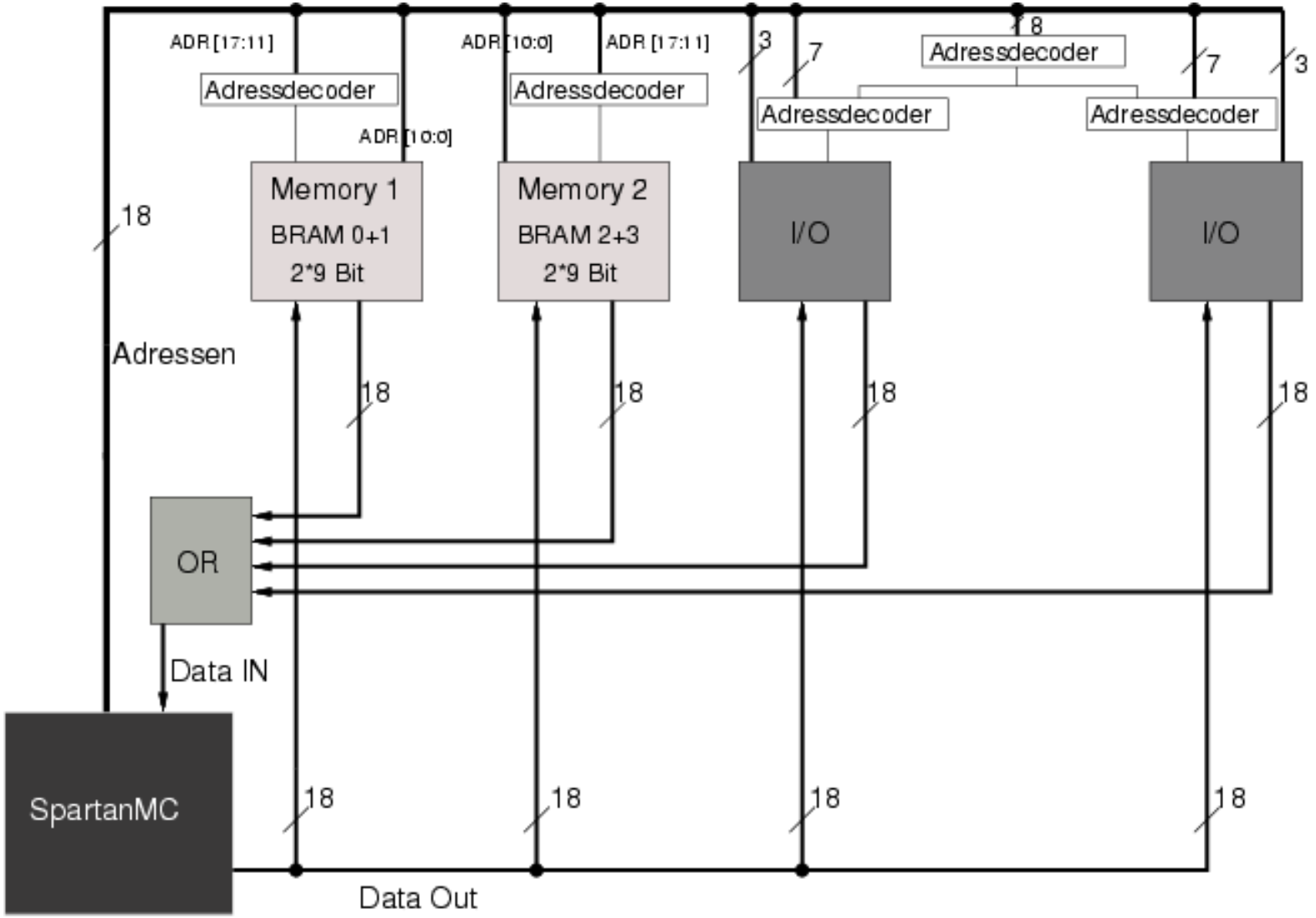
2.1 Die Register und der Speicher (2)

- UP und ISR sollten r8 bis r15 nutzen
- In r11 steht die Rückspungadresse vom UP oder ISR
- r0 bis r3 für spezielle Operanden (Stackpointer)
- Es können 4 Werte in r12 bis r15 an UP übergeben werden
- Im UP sind diese Werte in r4 bis r7
- Damit auch Rückgabe von 4 Ergebnissen des UP
- 127 geschachtelte UP/ISR Aufrufe ohne Stack möglich
- Register sind in einem 18 Bit Blockram des FPGA realisiert

2.1 Die Register und der Speicher (3)

- Neben den Universalregistern gibt es noch 6 *Special function* Register 0=SFR_STATUS, 1=SFR_LEDS, 2=SFR_MUL, 3=SFR_CC, 4=SFR_IV und 5=SFR_TR
- $SFR\_STATUS_{6:0}$ = RegBase – Nummer des Registerfenster
- SFR_STATUS_{7} = MM – ADR_{17} am Speicher für den Datenzugriff
- SFR_STATUS_{8} = EI – Freigabe der Interrupts
- SFR_LEDS – verbunden mit 7 Segment Anzeige oder LEDs
- SFR_MUL – oberen 18 Bit bei der Multiplikation
- SFR_CC_{0} – *Condition code* Bit
- SFR_IV – Startadresse der Interruptverarbeitung
- SFR_TR – Startadresse der TRAP-Tabelle (teilbar durch 256)

2.1 Die Register und der Speicher (4)



2.1 Die Register und der Speicher (5)

- Ein Speicherblock wird mit zwei 9 Bit Blockrams realisiert
- Speicherblock hat 2k Worte mit 18 Bit
- getrenntes Schreiben in die unteren oder oberen 9 Bit des 18 Bit Speicherwortes möglich
- Anzahl der Speicherblöcke begrenzt durch die Anzahl der Blockrams des verwendeten FPGA
- Blockrams sind immer selektiert, nur Output wird mit (SSR) Signal auf Festwertregister (belegt mit 0) umgeschaltet
- DATA_IN Bus wird durch Zusammenschaltung aller Speicher blöcke und Peripherie Module über OR-GATE realisiert
- Zweites OR-GATE am Befehlslesebus bei mehr als einem Speicherblock notwendig

2.2 Die Adressierungsarten des SpartanMC

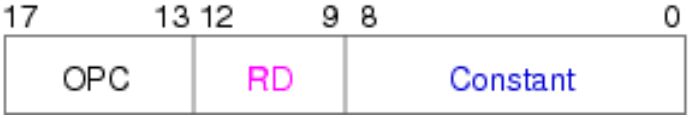
- Basis des Befehlssatz: modifizierter DLX-Befehlssatz mit nur 2 Adressen (Die DLX ähnelt dem MIPS-Prozessor)
- Die Register werden direkt adressiert
- Konstanten werden mit 9 Bit direkt angegeben (-256 bis 255 oder 0 bis 511)
- Speicher Zugriff: Register indirekt mit Displacement (Offset)
  - Displacement nur positiv und nur 5 Bit (0 bis 31)
  - Einschränkung notwendig, da nur 18 Bit/Befehl

2.3 Die Befehle des SpartanMC

- 4 Formate mit 2 Operanden in 18 Bit
- R (Register) Format


```
add   r4,   r6
subu  r12,  r14
```

- I (Immediate) Format

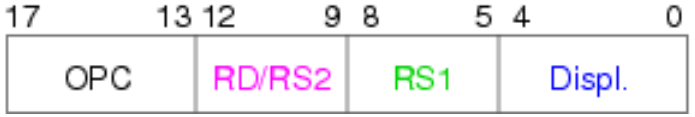



```
addi  r5,   -'0'
bnez  r7,   loop
```

- J (Jump) Format


```
jals  unterprogramm1
beqzc loop2
```

- M (Memory) Format




```
s18   6(r4),  r3
l9    r10,   0(r12)
```

2.3 Die Befehle des SpartanMC (2)

Hauptmatrix der Befehls Register Bits IR 17 - 13								
IR 17-13	**..000**	**..001**	**..010**	**..011**	**..100**	**..101**	**..110**	**..111**
00...	**Spezial1**	**Spezial2**	J	JALS	BEQZ	BNEZ	BEQZC	BNEZC
01...	ADDI	MOVI	LHI	SIGEX	ANDI	ORI	XORI	MULI
10...	L9	S9	L18	S18	SLLI	--- *)	SRLI	SRAI
11...	SEQI	SNEI	SLTI	SGTI	SLEI	SGEI	IFADDUI	IFSUBUI
Submatrix SPEZIAL 1 der Befehls Register Bits IR 4 - 0								
IR 4-0	**..000**	**..001**	**..010**	**..011**	**..100**	**..101**	**..110**	**..111**
00...	--- *)	--- *)	--- *)	--- *)	SLL	MOV	SRL	SRA
01...	SEQU	SNEU	SLTU	SGTU	SLEU	SGEU	--- *)	--- *)
10...	--- *)	--- *)	--- *)	--- *)	--- *)	--- *)	CBITS	SBITS
11...	--- *)	--- *)	--- *)	--- *)	--- *)	--- *)	--- *)	NOT
Submatrix SPEZIAL 2 der Befehls Register Bits IR 4 - 0								
IR 4-0	**..000**	**..001**	**..010**	**..011**	**..100**	**..101**	**..110**	**..111**
00...	RFE	TRAP	JR	JALR	JRS	JALRS	--- *)	--- *)
01...	--- *)	--- *)	--- *)	--- *)	--- *)	--- *)	--- *)	--- *)
10...	ADD	ADDU	SUB	SUBU	AND	OR	XOR	MUL
11...	SEQ	SNE	SLT	SGT	SLE	SGE	MOVI2S	MOVS2I

- *) Code wird nicht benutzt
- rot geschriebene Befehle werden in der Zukunft nicht unterstützt

2.3.1 Arithmetikbefehle

- **add** **Rd**, **Rs** R Rd <-- Rd+Rs; if (overflow) cc=1
- **addu** **Rd**, **Rs** R Rd <-- Rd+Rs
- **addi** **Rd**, **Imm** I Rd <-- Rd+IR_8^9##$IR_{8:0}$
- **ifaddui** **Rd**, **Imm** I if (cc==1) Rd <-- Rd+0^9##$IR_{8:0}$

CC darf nicht erst im vorhergehenden Befehl gesetzt werden.

- **sub** **Rd**, **Rs** R Rd <-- Rd-Rs; if (overflow) cc=1
- **subu** **Rd**, **Rs** R Rd <-- Rd-Rs
- **ifsubui** **Rd**, **Imm** I if (cc==1) Rd <-- Rd-0^9##$IR_{8:0}$

CC darf nicht erst im vorhergehenden Befehl gesetzt werden.

2.3.1 Arithmetikbefehle (2)

- **mul** **Rd**, **Rs** R SFR_MUL##Rd <-- Rd*Rs
- **muli** **Rd**, **Imm** I SFR_MUL##Rd <-- Rd*IR_8^9##$IR_{8:0}$

div muss mit Software realisiert werden.

- **seq** **Rd**, **Rs** R if (Rd==Rs) cc=1 else cc=0
- **sequ** **Rd**, **Rs** R if (Rd==Rs) cc=1 else cc=0
- **seqi** **Rd**, **Imm** I if (Rd==IR_8^9##$IR_{8:0}$) cc=1 else cc=0

- **sne** **Rd**, **Rs** R if (Rd!=Rs) cc=1 else cc=0
- **sneu** **Rd**, **Rs** R if (Rd!=Rs) cc=1 else cc=0
- **snei** **Rd**, **Imm** I if (Rd!=IR_8^9##$IR_{8:0}$) cc=1 else cc=0

2.3.1 Arithmetikbefehle (3)

- **slt** **Rd**, **Rs** R if (Rd<Rs) cc=1 else cc=0
- **sltu** **Rd**, **Rs** R if (Rd<Rs) cc=1 else cc=0
- **slti** **Rd**, **Imm** I if (Rd<IR_8^9##$IR_{8:0}$) cc=1 else cc=0

- **sle** **Rd**, **Rs** R if (Rd<=Rs) cc=1 else cc=0
- **sleu** **Rd**, **Rs** R if (Rd<=Rs) cc=1 else cc=0
- **slei** **Rd**, **Imm** I if (Rd<=IR_8^9##$IR_{8:0}$) cc=1 else cc=0

2.3.1 Arithmetikbefehle (4)

- **sgt** **Rd**, **Rs** R if (Rd>Rs) cc=1 else cc=0
- **sgtu** **Rd**, **Rs** R if (Rd>Rs) cc=1 else cc=0
- **sgti** **Rd**, **Imm** I if (Rd>IR_8^9##$IR_{8:0}$) cc=1 else cc=0

- **sge** **Rd**, **Rs** R if (Rd>=Rs) cc=1 else cc=0
- **sgeu** **Rd**, **Rs** R if (Rd>=Rs) cc=1 else cc=0
- **sgei** **Rd**, **Imm** I if (Rd>=IR_8^9##$IR_{8:0}$) cc=1 else cc=0

2.3.2 Logikbefehle

- **and** **Rd, Rs** R Rd <-- Rd & Rs
- **andi** **Rd, Imm** I Rd <-- Rd & 0^9##$IR_{8:0}$

- **or** **Rd, Rs** R Rd <-- Rd | Rs
- **ori** **Rd, Imm** I Rd <-- Rd | 0^9##$IR_{8:0}$

- **xor** **Rd, Rs** R Rd <-- Rd ^ Rs
- **xori** **Rd, Imm** I Rd <-- Rd ^ 0^9##$IR_{8:0}$

- **not** **Rd, Rs** R Rd <-- !Rs

2.3.2 Logikbefehle (2)

- **sll** **Rd, Rs** R cc##Rd <-- Rd << Rs
- **slli** **Rd, Imm** I cc##Rd <-- Rd << 0^9##$IR_{8:0}$

- **srl** **Rd, Rs** R cc##Rd <-- Rd >> Rs
- **srli** **Rd, Imm** I cc##Rd <-- Rd >> 0^9##$IR_{8:0}$

- **sra** **Rd, Rs** R cc##Rd <-- Rd $>>_a$ Rs
- **srai** **Rd, Imm** I cc##Rd <-- Rd $>>_a$ 0^9##$IR_{8:0}$

2.3.2 Logikbefehle (3)

- Schiebebefehle – 2 Konfigurationsvarianten der CPU
  - es wird immer nur um ein Bit verschoben (wenig hardware)
  - es kann beliebig verschoben werden (*Barrel-shifter*)
- einfachem Schieben
  - Wert in Rs oder im Imm wird ignoriert
  - Mehrfaches Schieben durch Schleifen

2.3.3 Transportbefehle

- **mov Rd, Rs** R Rd <-- Rs
  MOV notwendig, da nur 2 Operanden

- **movi Rd, Imm** I Rd <-- 0^9##$IR_{8:0}$
  Werte von 0 ... 511 (9 Bit)

- **lhi Rd, Imm** I Rd <-- $IR_{8:0}$##0^9
  - Mit LHI und ORI werden 18 Bit geladen
  - Mit LHI werden durch 512 teilbaren Werte geladen

2.3.3 Transportbefehle (2)

- **l9** **Rd, disp(Rs)** M Rd <-- 0^{9}##M[disp+Rs]
- **l18** **Rd, disp(Rs)** M Rd <-- M[disp+Rs]##M[disp+Rs+1]

- **s9** **disp(Rs)**, **Rs2** M M[disp+Rs] <-- Rs2
- **s18** **disp(Rs)**, **Rs2** M M[disp+Rs]##M[disp+Rs+1] <-- Rs2

- **l9** und **s9** zur besseren Speicherauslastung bei Zeichenketten
- In C wird mit **struct** diese Adressierung besonders unterstützt. Programme werden damit deutlich kürzer. Beim SpartanMC kann ein **struct** aber nur über 32 Byte genutzt werden, da **disp** nur 5 Bit hat. Die Verwendung von **struct** ist im lcc18 nur für globale Variable möglich.

2.3.3 Transportbefehle (3)

- **movi2s** **SfrNr, Rs** R SFR_reg <-- Rs
- **movs2i** **Rd, SfrNr** R Rd <-- SFR_reg
  SfrNr wird im R Format immer auf die Bits von Rs1 gespeichert, auch bei MOVI2S

- **sigex** **Rd, BitNr+1** I Rd <-- $Rd_{BitNr}^{17-BitNR}$##$Rd_{BitNr:0}$
  BitNr+1 darf nur 16, 9 oder 8 sein

- Werden in C die Datentypen ohne den Vorsatz **unsigned** eingesetzt, dann wird beim Laden eines Wertes kleiner 18 Bit immer der Befehl **sigex** eingefügt.

2.3.4 Verzweigebefehle

- Die folgenden 2 Befehle haben einen **Delay-Slot**.
- **beqz** **Rd**, **label** I PC=PC+1 if(Rd==0) PC <-- label
- **bnez** **Rd**, **label** I PC=PC+1 if(Rd!=0) PC <-- label

- Das cc darf nicht erst im vorherigen Befehl gesetzt werden!
- **beqzc** **label** J if(cc==0) PC <-- label
- **bnezc** **label** J if(cc!=0) PC <-- label

- **j** **label** J PC <-- label

2.3.4 Verzweigebefehle (2)

- **Unterprogramm und ISR Arbeit**
- **trap** **Nummer** R RegBase<--RegBase+1;R11<--PC+1;
  PC<--SFR_TR$_{17:8}$##Nummer
- **jals** **label** J RegBase<--RegBase+1;r11<--PC+1;PC<--label
- Die folgenden 5 Befehle haben einen **Delay-Slot**.
- **jalrs** **Rs** R PC=PC+1;RegBase<--RegBase+1;r11<--PC+1;
  PC<--Rs
- **jrs** **Rs** R PC=PC+1;PC<--Rs;RegBase<--RegBase-1
- **rfe** **Rs** R PC=PC+1;PC<--Rs;RegBase<--RegBase-1
  Erzeugt ein Signal für den Interruptkontroller
- **jalr** **Rs** R PC=PC+1;r11<--PC+1;PC<--Rs
- **jr** **Rs** R PC=PC+1;PC<--Rs (Für **CASE** Verzweigung)

2.3.4 Verzweigebefehle (3)

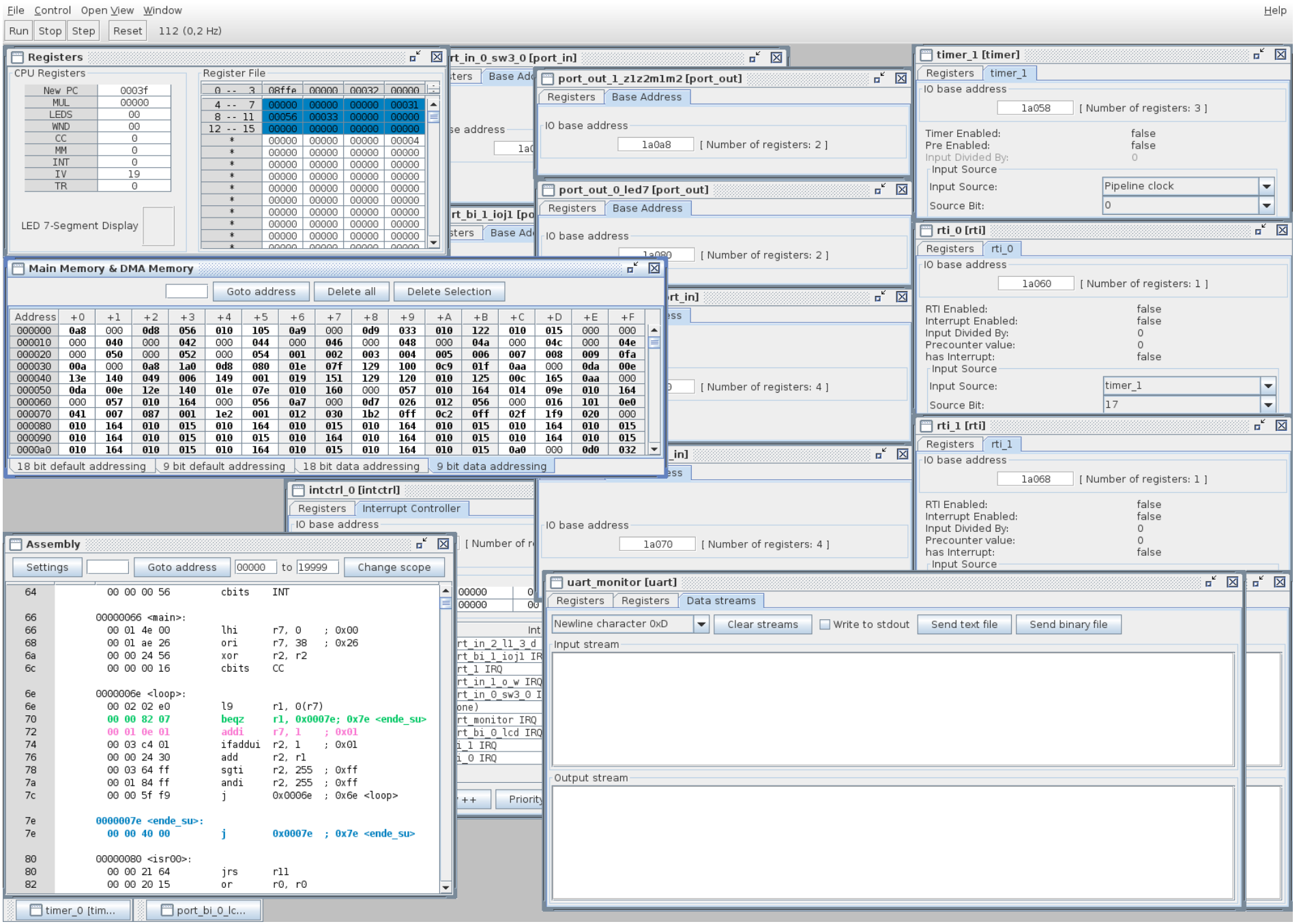
- Der **Delay-Slot** wird bei Befehlen eingesetzt, die den linearen Ablauf der Befehlsabarbeitung verlassen, also nicht mit dem Folgebefehl weiter arbeiten.
- Dies ist meist notwendig, da die Bedingung für den Sprung erst im nächsten Takt vorliegt.
- Um keine Pipeline Verluste zu haben, wird der Befehl hinter dem Sprung noch so abgearbeitet, als ob er davor steht, also auch mit dem alten Register Fenster!
- Der Befehl wird zunächst mit einem NOP (or r0,r0) belegt.
- Bei der Optimierung wird aus der Befehlsfolge vom letzten Einsprung bis zum Sprung ein Befehl gesucht, der die Sprung-bedingung nicht beeinflusst und hinter den Sprung an die Stelle des NOP in den **Delay-Slot** verschoben.
- Der **Delay-Slot** wird in einer Schleife immer mit abgearbeitet.

2.3.5 Steuerbefehle

- Befehle zum ein- und ausschalten von Statusbits
  - CC – BitNr = 0 = SFR_CC_0
  - MM – BitNr = 1 = SFR_STATUS_7
  - INT – BitNr = 2 = SFR_STATUS_8

- **cbits** **BitNr** R SFR_bit <-- 0
- **sbits** **BitNr** R SFR_bit <-- 1

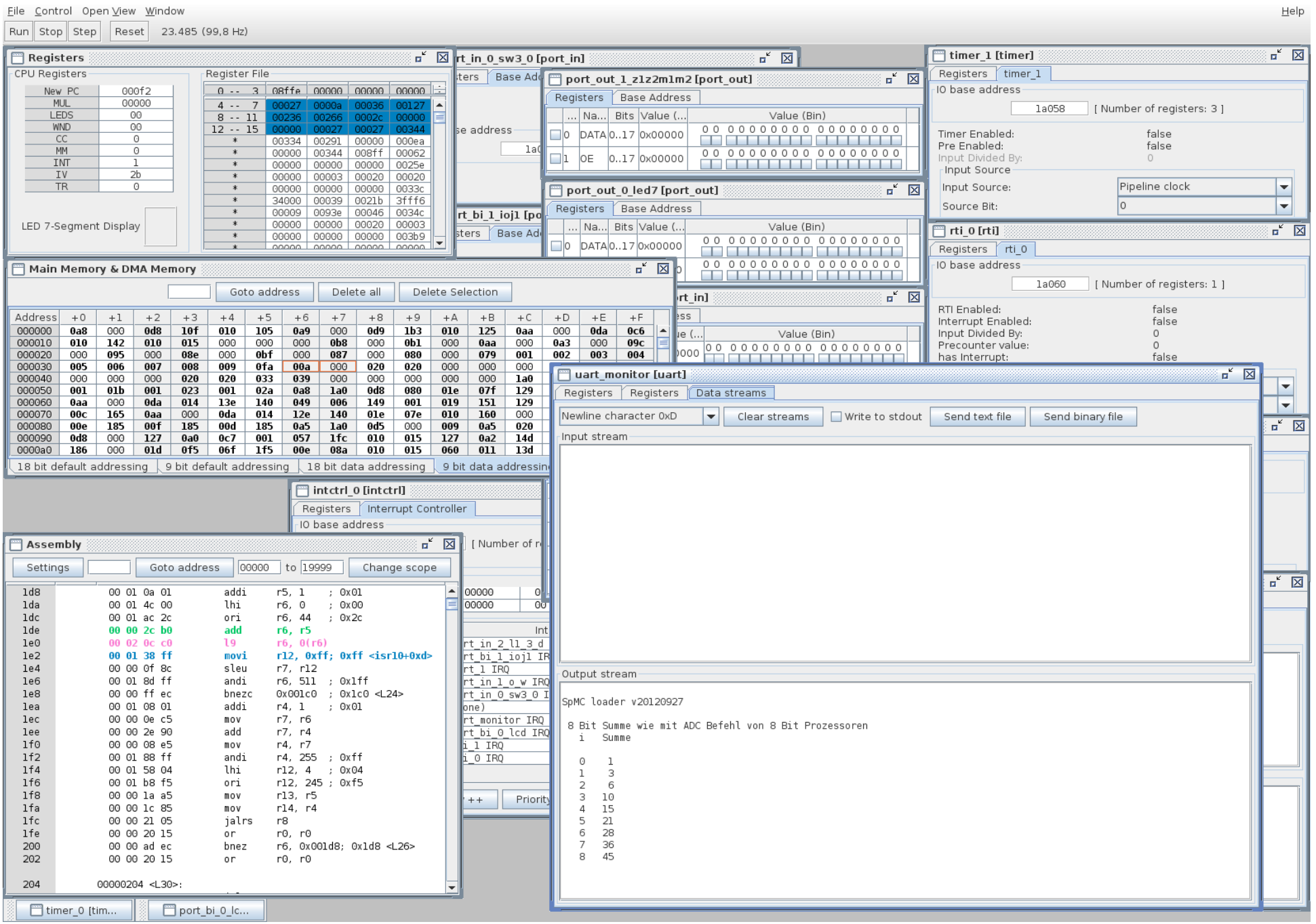
```
; 8 Bit Summe über eine Byte Liste bilden, Ende bei 0 (Im Simulator testen) (summe8a.s)
;
            .data
;
liste1:     .byte   1,2,3,4,5,6         ;Byte liegen im 9 Bit Abstand. Wird mit L18 von
            .byte   7,8,9,250,10,0      ;liste1 geladen steht im Rd 0x0202
;
.include    "../lib_obj/src/interrupt/interrupt.s"
            .text
;
.global     main
main:       lhi     r7, %hi(liste1)
            ori     r7, %lo(liste1)     ;Zeiger auf Liste1 in r7
            XOR     R2, R2              ;Summe löschen
            cbits   CC                  ;CC = 0 setzen
loop:       L9      R1, 0(R7)           ;Byte laden
            BEQZ    R1, ende_su         ;0 gefunden --> ENDE
            ADDI    R7, 1               ;Zeiger + 1 (wird im Delay-Slot ausgeführt)
            IFADDUI R2, 1               ;8 Bit Übertrag vom letzten ADD addieren
            ADD     R2, R1              ;Summe bilden
            SGTI    R2, 255             ;CC = 1 wenn größer 255
            ANDI    R2, 255             ;Nur 8 Bit übrig lassen
            J       loop
ende_su:    J       ende_su             ;Summe in R2
;
.global     isr00
isr00:      jrs     r11
            or      r0, r0
```


File Control Open View Window
Help
Run Stop Step Reset 112 (0,2 Hz)
Registers
CPU Registers
New PC 0003f
MUL 00000
LEDS 00
WND 00
CC 0
MM 0
INT 0
IV 19
TR 0
LED 7-Segment Display
Register File
Main Memory & DMA Memory
Goto address
Delete all
Delete Selection
18 bit default addressing
9 bit default addressing
18 bit data addressing
9 bit data addressing
Assembly
Settings
Goto address
00000 to 19999
Change scope
timer_1 [timer]
IO base address
1a058 [Number of registers: 3]
Timer Enabled: false
Pre Enabled: false
Input Divided By: 0
Input Source
Input Source: Pipeline clock
Source Bit: 0
port_out_1_z1z2m1m2 [port_out]
IO base address
1a0a8 [Number of registers: 2]
port_out_0_led7 [port_out]
IO base address
rti_0 [rti]
IO base address
1a060 [Number of registers: 1]
RTI Enabled: false
Interrupt Enabled: false
Input Divided By: 0
Precounter value: 0
has Interrupt: false
Input Source
Input Source: timer_1
Source Bit: 17
rti_1 [rti]
IO base address
1a068 [Number of registers: 1]
RTI Enabled: false
Interrupt Enabled: false
Input Divided By: 0
Precounter value: 0
has Interrupt: false
Input Source
intctrl_0 [intctrl]
Interrupt Controller
IO base address
1a070 [Number of registers: 4]
uart_monitor [uart]
Registers
Data streams
Newline character 0xD
Clear streams
Write to stdout
Send text file
Send binary file
Input stream
Output stream
timer_0 [tim...
port_bi_0_lc...

```
/*
 * 8 Bit Summe wie mit ADC Befehl von 8 Bit Prozessoren (summe8l.c)
 */
#include <system/peripherals.h> // Registerstrukturen aller installierten Interfaces
//                                 I/O-Adessen und Konstanten des Systems
#include <uart.h>
#include <stdio.h>
#include <interrupt.h>
/*
 * Konstanten aus der "system/peripherals.h"
 */
#define  ANZ_ISR      i_bits
/*
 * Zuordnung der Interrupts
 */
#define  isr_rti0     isr00    // Sekunden Interupt von RTI0
#define  isr_rti1     isr01    // Interrupt von RTI1 aller 4 Sekunden
#define  isr_def01    isr02    // port_bi_0_LCD.intr - kein Interrupt
#define  isr_def02    isr03    // uart_monitor.intr
#define  isr_def03    isr04    // rot_0.intr
#define  isr_def04    isr05    // port_in_0_SW3_0.intr
#define  isr_def05    isr06    // port_in_1_O_W.intr
#define  isr_uart1    isr07    // uart_1.intr
#define  isr_def06    isr08    // port_bi_1_ioj1.intr
#define  isr_def07    isr09    // port_in_2_L1_3_D.intr
#define  isr_def08    isr10    // interrupt_lowactiv

    unsigned char su[12]   = {1,2,3,4,5,6,7,8,9,250,10,0}; // zu addierende Werte

/*
 * 8 Bit Summe wie mit ADC Befehl von 8 Bit Prozessoren
 */
```

```
void main() {

interrupt_disable();
uart_wait_idle(UART_MONITOR);
UART_MONITOR->ctrl_stat = UART_RX_EN|UART_TX_EN|UART_DATA_LEN_8|UART_BPS_115200;
stdio_uart_open(UART_MONITOR);
interrupt_enable();

   while   (1) {

       printf("\r\n 8 Bit Summe wie mit ADC Befehl von 8 Bit Prozessoren\
               \r\n   i   Summe\r\n");

       unsigned int  z    = 0;     // Index fuer su
       unsigned int  supu = 0;     // Summe 18 Bit
       unsigned int  sum  = 0;     // Summe  8 Bit
       unsigned int  su2  = 1;     // aktueller Summand
       while (su2 > 0) {
            su2  = su[z];
            if (supu > 255) {      // war die letzte Summe > 255
                 sum  = sum + 1;   // Uebertrag addieren
            }
            supu = sum + su2;      // neue Summe
            sum  = supu & 255;     // auf 8 Bit beschneiden
       printf("\r\n %3d  %3d", z, sum);
   //  printf("  %5x",sum);
       z++;
       }
   unsigned char    urd  = getchar();
   }
}
```

```
/* Funktionen zur Interrupt Behandlung */
// UART1
void isr_uart1(int reg12, int reg13, int reg14, int reg15) {
       printf("isr_uart1 PC=%05x CC=%01x\r\n",reg12,reg14);
       }

// rti00.intr
void isr_rti0(int reg12, int reg13, int reg14, int reg15) {
       printf("isr_rti00 PC=%05x CC=%01x\r\n",reg12,reg14);
       }

// rti01.intr
void isr_rti1(int reg12, int reg13, int reg14, int reg15) {
       printf("isr_rti01 PC=%05x CC=%01x\r\n",reg12,reg14);
       }

// fuer alle nicht genutzten ISR-Signale
#if    ANZ_ISR >= 4
void isr_def01(int reg12, int reg13, int reg14, int reg15) {
       printf("isr_def01 PC=%05x CC=%01x\r\n",reg12,reg14);
       }
#endif
// fuer alle nicht genutzten ISR-Signale
#if    ANZ_ISR >= 5
void isr_def02(int reg12, int reg13, int reg14, int reg15) {
       printf("isr_def02 PC=%05x CC=%01x\r\n",reg12,reg14);
       }
#endif
```

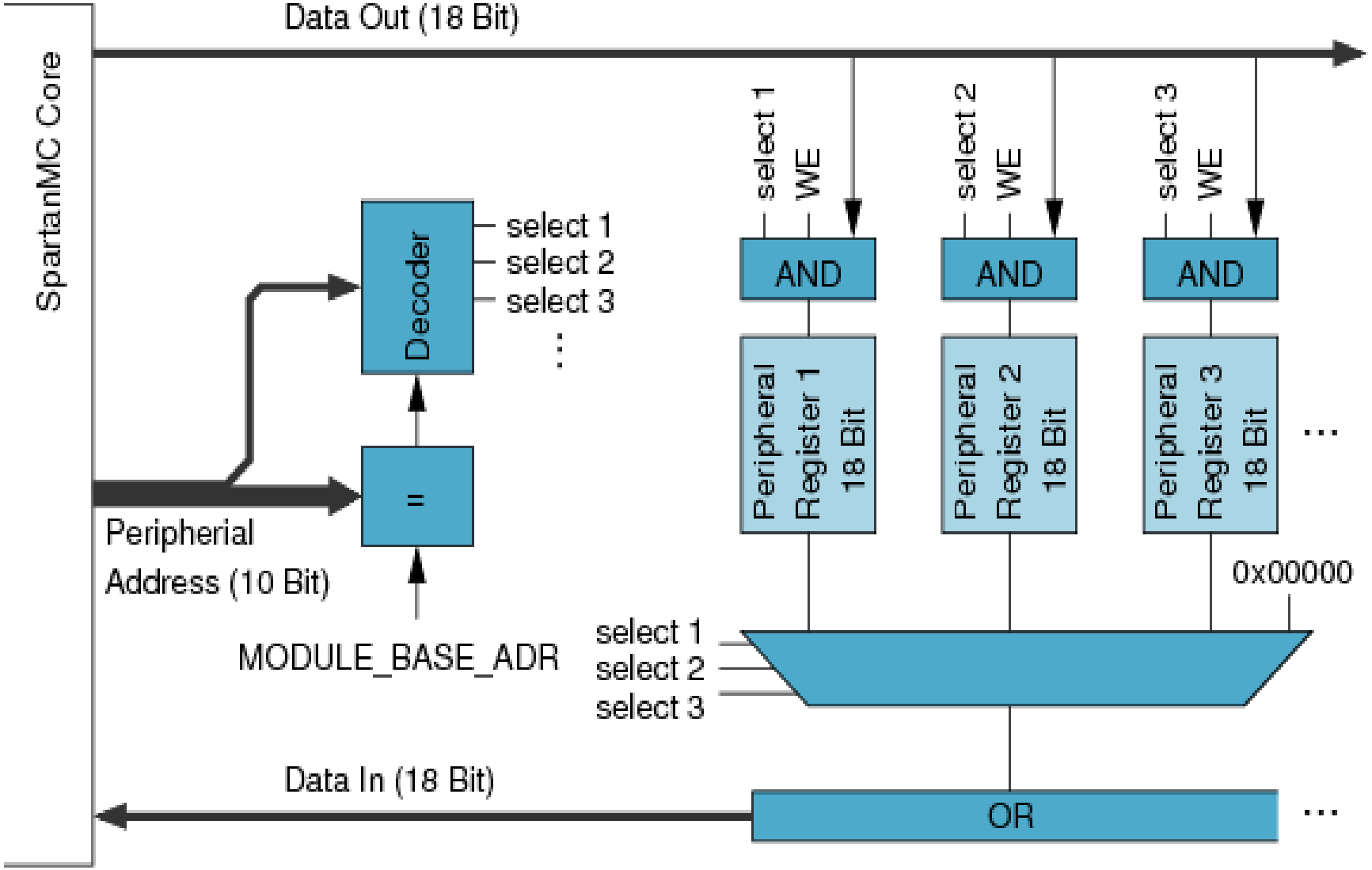

File Control Open View Window
Help
Run Stop Step Reset 23.485 (99,8 Hz)
Registers
CPU Registers
New PC 000f2
MUL 00000
LEDS 00
WND 00
CC 0
MM 0
INT 1
IV 2b
TR 0
LED 7-Segment Display
Register File
0 -- 3 08ffe 00000 00000 00000
4 -- 7 00027 0000a 00036 00127
8 -- 11 00236 00266 0002c 00000
12 -- 15 00000 00027 00027 00344
Main Memory & DMA Memory
Goto address
Delete all
Delete Selection
18 bit default addressing
9 bit default addressing
18 bit data addressing
Assembly
Settings
Goto address
00000 to 19999
Change scope
1d8 00 01 0a 01 addi r5, 1 ; 0x01
1da 00 01 4c 00 lhi r6, 0 ; 0x00
1dc 00 01 ac 2c ori r6, 44 ; 0x2c
1de 00 00 2c b0 add r6, r5
1e0 00 02 0c c0 l9 r6, 0(r6)
1e2 00 01 38 ff movi r12, 0xff; 0xff <isr10+0xd>
1e4 00 00 0f 8c sleu r7, r12
1e6 00 01 8d ff andi r6, 511 ; 0x1ff
1e8 00 00 ff ec bnezc 0x001c0 ; 0x1c0 <L24>
1ea 00 01 08 01 addi r4, 1 ; 0x01
1ec 00 00 0e c5 mov r7, r6
1ee 00 00 2e 90 add r7, r4
1f0 00 00 08 e5 mov r4, r7
1f2 00 01 88 ff andi r4, 255 ; 0xff
1f4 00 01 58 04 lhi r12, 4 ; 0x04
1f6 00 01 b8 f5 ori r12, 245 ; 0xf5
1f8 00 00 1a a5 mov r13, r5
1fa 00 00 1c 85 mov r14, r4
1fc 00 00 21 05 jalrs r8
1fe 00 00 20 15 or r0, r0
200 00 00 ad ec bnez r6, 0x001d8; 0x1d8 <L26>
202 00 00 20 15 or r0, r0
204 00000204 <L30>:
intctrl_0 [intctrl]
Registers
Interrupt Controller
IO base address
port_out_1_z1z2m1m2 [port_out]
port_out_0_led7 [port_out]
timer_1 [timer]
IO base address
1a058 [Number of registers: 3]
Timer Enabled: false
Pre Enabled: false
Input Divided By: 0
Input Source
Input Source: Pipeline clock
Source Bit: 0
rti_0 [rti]
IO base address
1a060 [Number of registers: 1]
RTI Enabled: false
Interrupt Enabled: false
Input Divided By: 0
Precounter value: 0
has Interrupt: false
uart_monitor [uart]
Registers
Registers
Data streams
Newline character 0xD
Clear streams
Write to stdout
Send text file
Send binary file
Input stream
Output stream
SpMC loader v20120927
8 Bit Summe wie mit ADC Befehl von 8 Bit Prozessoren
i Summe
0 1
1 3
2 6
3 10
4 15
5 21
6 28
7 36
8 45
timer_0 [tim...
port_bi_0_lc...

```
Willkommen zu minicom 2.5

Optionen: I18n
Übersetzt am Feb 24 2011, 11:25:55.
Port /dev/ttyS0

Drücken Sie CTRL-A  Z für Hilfe zu speziellen Tasten

SpMC loader v20120927

 8 Bit Summe wie mit ADC Befehl von 8 Bit Prozessoren
   i   Summe

   0    1
   1    3
   2    6
   3   10
   4   15
   5   21
   6   28
   7   36
   8   45
   9   39
  10   50
  11   50
 8 Bit Summe wie mit ADC Befehl von 8 Bit Prozessoren
   i   Summe

   0    1
   1    3
   2    6
   3   10
   4   15
   5   21
   6   28
   7   36
   8   45
   9   39
  10   50
  11   50
 CTRL-A  Z = Hilfe |115200 8N2 | NOR | Minicom 2.5    | VT102 | Online 00:01
```

2.4 Einige wichtige Interface des SpartanMC

- Interface werden wie Speicher angesprochen (Memory Maped)
- Jedes Interface kann ein oder mehrere Ports am OR_GATE belegen
- Geräteregister reservieren immer 18 Bit
- I/O Basisadresse befindet sich hinter dem letzten Speicherblock
- Von Adresse 0x00000 beginnt der Speicher
- DMA Puffer liegen unterhalb der I/O Basisadresse und oberhalb des Speichers
- Interruptsignale müssen im Gerät bis zur Annahme gespeichert werden
- Interruptsignale müssen unterdrückbar sein (nach RESET)

Interface des SpartanMC



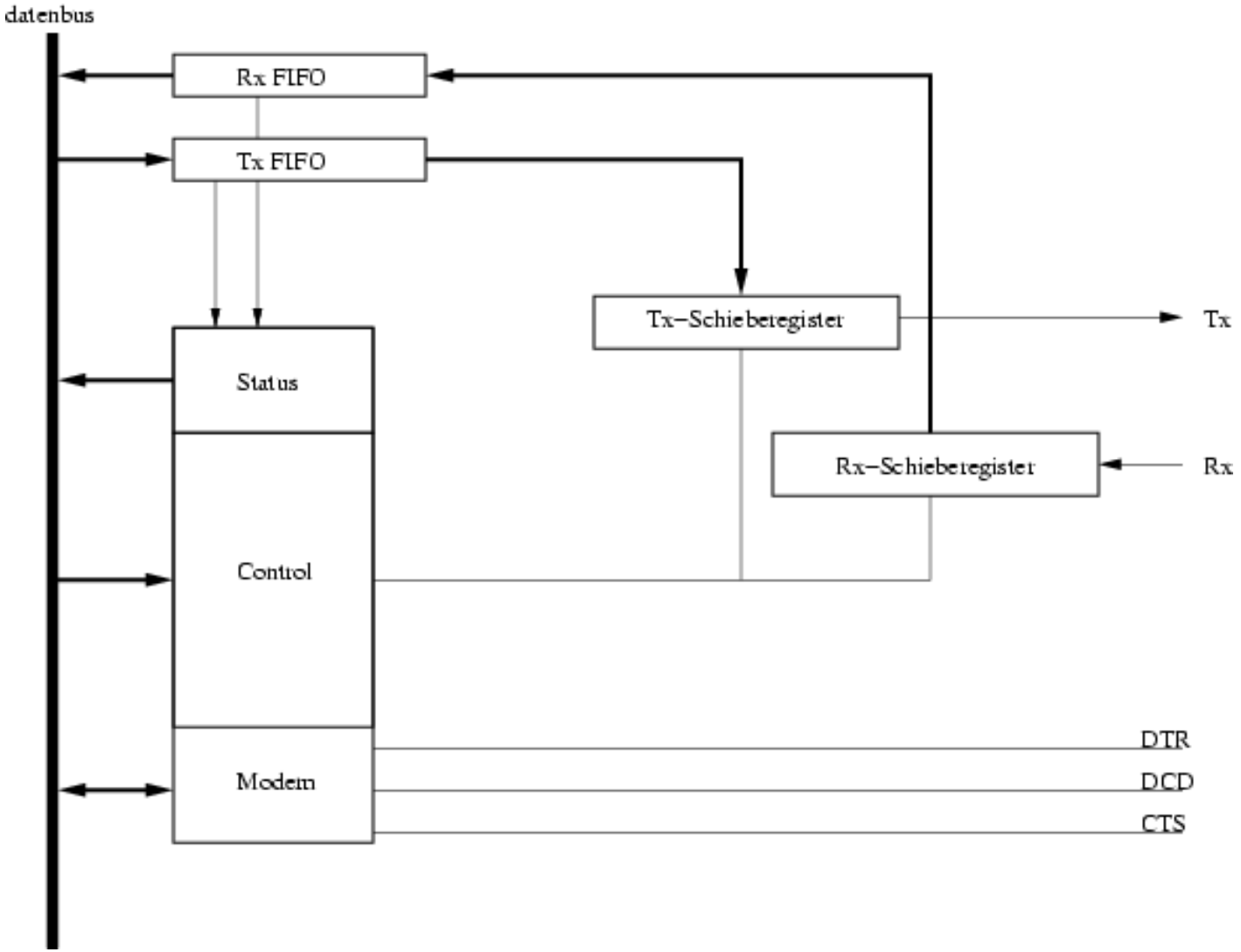
2.4.1 Parallele Ein- und Ausgabe

- Zur Abfrage von Schaltern und Tastern
- Zur Ansteuerung von Lampen, LEDs, Relais, ...
- Für Geräte mit bidirektionalen Bussen (LCD)
- 1 bis 18 Bit Eingabe (Flanken Erkennung / Interrupt möglich)
- 1 bis 18 Bit Ausgabe
- 1 bis 18 Bit Ein-/Ausgabe (Open Drain Output möglich)
- Jedes der 3 Ports kann mehrfach installiert werden

2.4.2 Serielle Ein- und Ausgabe

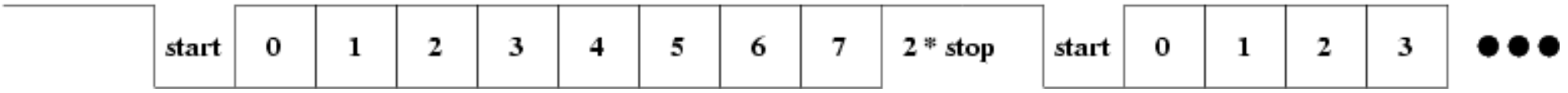
- Zur Kommunikation mit anderen Rechnern und Geräten oft serielle Interfaces
- Sie haben nur wenige Leitungen
- Es gibt asynchrone und synchrone Interface
  - UART – asynchron
  - SPI – synchron
  - IIC – synchron

2.4.2.1 Die UART des SpartanMC



2.4.2.1 Die UART des SpartanMC (2)

- FIFO für Rx und Tx getrennt einstellbar (auf 8 voreingestellt)
- Modem Steuerleitungen können eingerichtet werden (voreingestellt sind keine Steuerleitungen)
- 15 Datenratren von 50 Baud bis 115200 Baud darunter auch
  - 31250 Baud für MIDI Realisierungen
  - 7812,5 Baud zum Booten von 68hc11 Kontrollern
- 5, 6, 7 oder 8 Datenbit
- Paritätsbit gerade/ungerade oder ohne
- 1 oder 2 Stopbits



2.4.2.1 Die UART des SpartanMC (3)

- Senden von BREAK möglich
- Interrupt für Rx und Tx getrennt möglich
- Erkennung von Rx Datenverlust
- Erkennung von Paritätsfehlern
- Erkennung von Framefehlern (Break, Leitungsunterbrechung, unterschiedliche Datenlängen oder Datenraten)

```
#ifndef __UART_H
#define __UART_H

#include <stdint.h>

// Status Signale
#define UART_RX_EMPTY    (1<<0)
#define UART_RX_FULL     (1<<1)
#define UART_TX_EMPTY    (1<<2)
#define UART_TX_FULL     (1<<3)
#define UART_TX_IRQ_PRE  (1<<4)
#define UART_TX_IRQ_FLAG (1<<5)
#define UART_RX_P_ERR    (1<<6)
#define UART_RX_F_ERR    (1<<7)
#define UART_RX_D_ERR    (1<<8)
#define UART_M_DCD       (1<<9)
#define UART_M_CTS       (1<<10)
#define UART_M_DSR       (1<<11)
#define UART_RX_CLK      (1<<12)
#define UART_RX_STOP     (1<<13)
#define UART_TX_CLK      (1<<14)
#define UART_RST_UART    (1<<15)     // UART noch im RESET, wenn = 1
#define UART_TX_STOP     (1<<16)

// Steuersignale
#define UART_RX_EN       (1<<0)
#define UART_TX_EN       (1<<1)
#define UART_PARI_EN     (1<<2)
#define UART_PARI_EVEN   (1<<3)      // 0 = ungerade / 1 = gerade
#define UART_TWO_STOP    (1<<4)      // 0 = ein / 1 = zwei Stopbits

#define UART_DATA_LEN_5  (4<<5)      // 0x00080
#define UART_DATA_LEN_6  (5<<5)      // 0x000A0
#define UART_DATA_LEN_7  (6<<5)      // 0x000C0
#define UART_DATA_LEN_8  (7<<5)      // 0x000E0
```

```
#define UART_BPS_115200  (0<<9)      // 0x00000
#define UART_BPS_57600   (1<<9)      // 0x00200
#define UART_BPS_38400   (2<<9)      // 0x00400
#define UART_BPS_31250   (3<<9)      // 0x00600 MIDI Datenrate
#define UART_BPS_19200   (4<<9)      // 0x00800
#define UART_BPS_9600    (5<<9)      // 0x00A00
#define UART_BPS_4800    (6<<9)      // 0x00C00
#define UART_BPS_2400    (7<<9)      // 0x00E00
#define UART_BPS_1200    (8<<9)      // 0x01000
#define UART_BPS_600     (9<<9)      // 0x01200
#define UART_BPS_300     (10<<9)     // 0x01400
#define UART_BPS_150     (11<<9)     // 0x01600
#define UART_BPS_75      (12<<9)     // 0x01800
#define UART_BPS_50      (13<<9)     // 0x01A00
#define UART_BPS_7812    (14<<9)     // 0x01C00 Boot 68hc11 mit 7812,5 Baud
#define UART_RX_IE       (1<<13)
#define UART_TX_IE       (1<<14)
#define UART_TX_BREAK    (1<<15)
// Modem Outputs und Richtung
#define UART_DTR_DSR     (1<<0)
#define UART_RTS_CTS     (1<<1)
#define UART_DCD         (1<<2)
#define UART_DCD_OUT     (1<<3)
#define UART_RTS_OUT     (1<<4)
#define UART_DTR_OUT     (1<<5)
// Rückgabewerte für non blocking read
#define UART_OK          0
#define UART_NO_DATA     1
typedef struct {
    volatile uint18_t   status;     // read
    volatile uint18_t   rx_data;    // read (Reset Rx Interrupt)
    volatile uint18_t   tx_data;    // write
    volatile uint18_t   ctrl_stat;  // write (or read = status & Reset Tx Interrupt)
    volatile uint18_t   modem;      // write (optional)
} uart_regs_t;
#endif /*UART_H_*/
```

```
#include <system/hardware.h>         // I/O-Adessen und Konstanten des Systems
#include <system/peripherals.h>     // Registerstrukturen aller installierten Interfaces
#include <uart.h>
#include <stdio.h>
#include <stdio_uart.h>
#include <interrupt.h>

     unsigned int  u_stat;
     unsigned int  uw   = 0;         // Tx-UART bereit, wenn = 1
     unsigned int  ur   = 0;         // Rx-UART bereit, wenn = 1 (neue Daten auf urd)
     unsigned char urd  = ' ';       // Rx-UART, letztes Zeichen
     unsigned char e    = 0;         // keine Eingabe
     unsigned char tx   = ' ';       // zu sendendes Zeichen

/* Funktione zur Interrupt Behandlung von UART2 */
void isr01() {
     u_stat = UART_1->status;
     if ((u_stat & UART_RX_EMPTY) == 0) {
          urd = UART_1->rx_data;     // Daten lesen setzt Rx Interrupt zurueck
          ur  = 1;                   // neues Zeichen Empfangen
          }
     if ((u_stat & UART_TX_IRQ_FLAG) != 0) {
          uw = 1;                    // Tx bereit zum senden melden
          u_stat =UART_1->ctrl_stat; // Tx Interrupt zurueck setzen
          }
     }
// Anzeige der Werte von UART2
void uart2anz(void) {
     unsigned int  txa;
     unsigned int  rxa;
     txa  =    tx;
     rxa  =    urd;
     printf("\r %3x  %3x", txa, rxa);
     }
```

```
/*
 * An UART2 muss fuer den Test Rx mit Tx gebrueckt werden. (Pin 2 und 3 am DB9 oder DB25)
 */
void main() {

    interrupt_disable();

    UART_MONITOR->ctrl_stat    = UART_RX_EN|UART_TX_EN|UART_TWO_STOP|UART_DATA_LEN_8|
                                 UART_BPS_115200;
    stdio_uart_open(UART_MONITOR);
    UART_1->ctrl_stat          = UART_RX_EN|UART_TX_EN|UART_TWO_STOP|UART_DATA_LEN_8|
                                 UART_BPS_50|UART_RX_IE|UART_TX_IE;
    interrupt_enable();

    printf("\r\nFortlaufend Senden und Empfangen mit UART2\
            \r\n Tx   Rx\r\n");


    while    (1) {

        if (uw == 1) {
            uart2anz();
            uw = 0;                // Vor dem Senden ruecksetzen
            UART_1->tx_data = tx;  // Zeichen senden --> Interrupt
            tx = tx + 1;
            if (tx >= 0xff) tx = 0x0;
            }
        if (ur == 1) {
            ur = 0;
            uart2anz();
            }
        }
}
```

Einige Auszüge aus der Übersetzungsliste des Programms

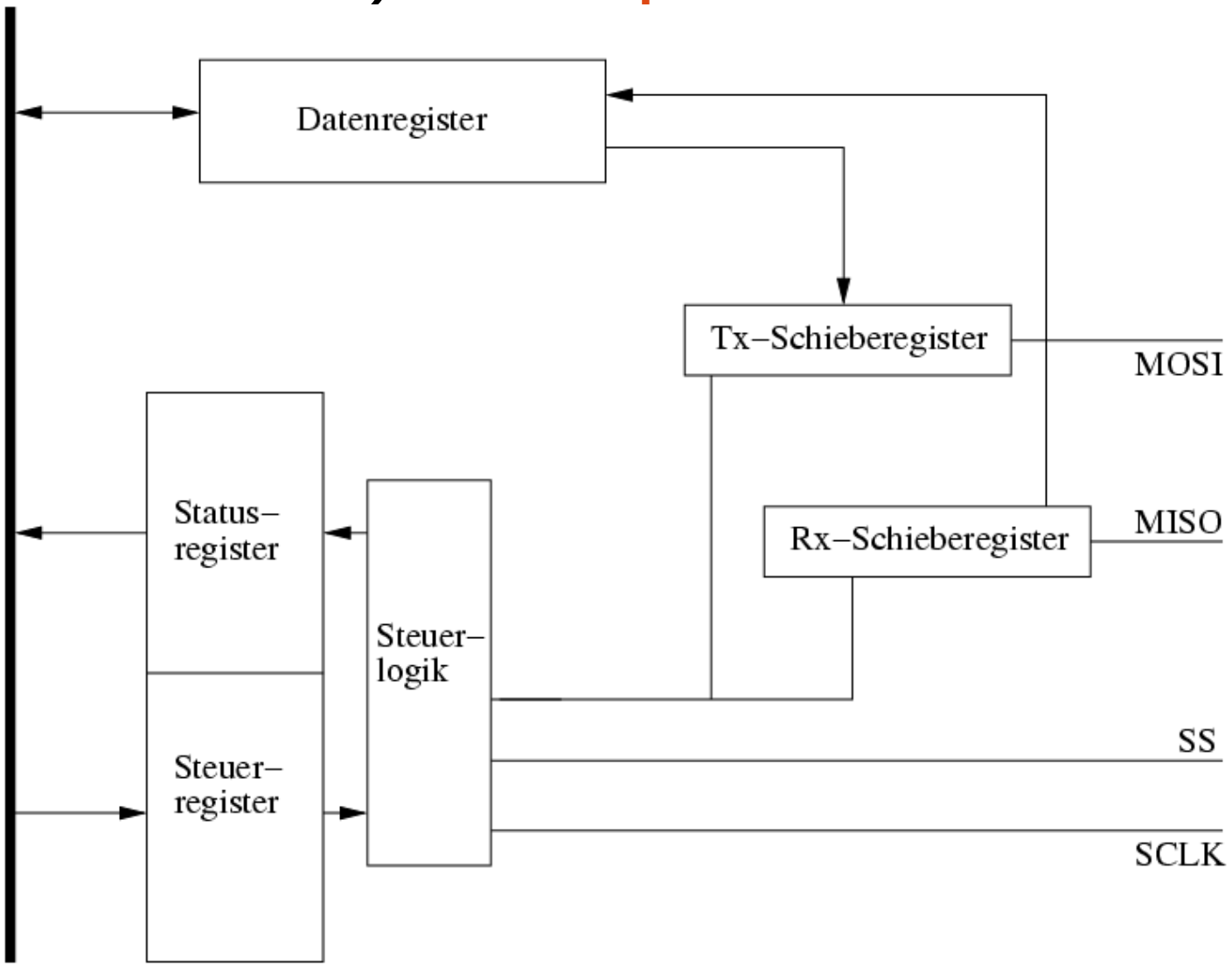
```
;
; Befehlsfolge für die Initialisierung
;
000001ce <main>:
 1ce: 00 01 01 fe  addi    r0, -2  ; 0x1fe
 1D0: 00 02 64 00  s18     0(r0), r2
 1D2: 00 00 61 36  jals    0x0043e ; 0x43e <interrupt_disable>
 1D4: 00 01 59 a0  lhi     r12, 416   ; 0x1a0
 1D6: 00 01 b8 00  ori     r12, 0  ; 0x00
 1D8: 00 00 61 36  jals    0x00444 ; 0x444 <uart_wait_idle>
 1Da: 00 01 59 a0  lhi     r12, 416   ; 0x1a0
 1dc: 00 01 b8 00  ori     r12, 0  ; 0x00
 1De: 00 01 28 f3  movi    r4, 0xf3   ; 0xf3 <isr07+0x9>
 1E0: 00 02 69 86  s18     6(r12), r4
 1E2: 00 00 61 3c  jals    0x0045a ; 0x45a <stdio_uart_open>
 1E4: 00 01 49 a0  lhi     r4, 416 ; 0x1a0
 1E6: 00 01 a8 10  ori     r4, 16  ; 0x10
 1E8: 00 01 4a 3d  lhi     r5, 61  ; 0x3d
 1Ea: 00 01 aa f3  ori     r5, 243 ; 0xf3
 1Ec: 00 02 6a 86  s18     6(r4), r5
 1Ee: 00 00 61 25  jals    0x00438 ; 0x438 <interrupt_enable>
 1F0: 00 01 58 04  lhi     r12, 4  ; 0x04
 1F2: 00 01 b8 c6  ori     r12, 198   ; 0xc6
 1F4: 00 01 3a 00  movi    r13, 0x00  ; 0x0 <LOADER_UART_LIGHT>
 1F6: 00 00 1d a5  mov     r14, r13
 1F8: 00 00 61 5a  jals    0x004ac ; 0x4ac <printf_>
```

```
000000ea <isr07>:
  ea:   00 01 5d a0    lhi     r14, 416   ; 0x1a0
  ec:   00 01 bc 10    ori     r14, 16    ; 0x10
  ee:   00 02 59 c0    l18     r12, 0(r14)
  f0:   00 01 5a 00    lhi     r13, 0     ; 0x00
  f2:   00 01 ba 32    ori     r13, 50    ; 0x32
  f4:   00 00 1f 85    mov     r15, r12
  f6:   00 01 9e 01    andi    r15, 1     ; 0x01
  f8:   00 03 1e 00    seqi    r15, 0     ; 0x00
  fa:   00 02 79 a0    s18     0(r13), r12
  fc:   00 00 c0 09    beqzc   0x0010e    ; 0x10e <L2>
  fe:   00 02 5f c2    l18     r15, 2(r14)
 100:   00 01 5c 00    lhi     r14, 0     ; 0x00
 102:   00 01 bc 30    ori     r14, 48    ; 0x30
 104:   00 02 3f c0    s9      0(r14), r15
 106:   00 01 5c 00    lhi     r14, 0     ; 0x00
 108:   00 01 bc 2e    ori     r14, 46    ; 0x2e
 10a:   00 01 28 01    movi    r4, 0x01   ; 0x1 <SB_PORT_BI_1_IOJ1_OD_OUTPUT>
 10c:   00 02 69 c0    s18     0(r14), r4
```

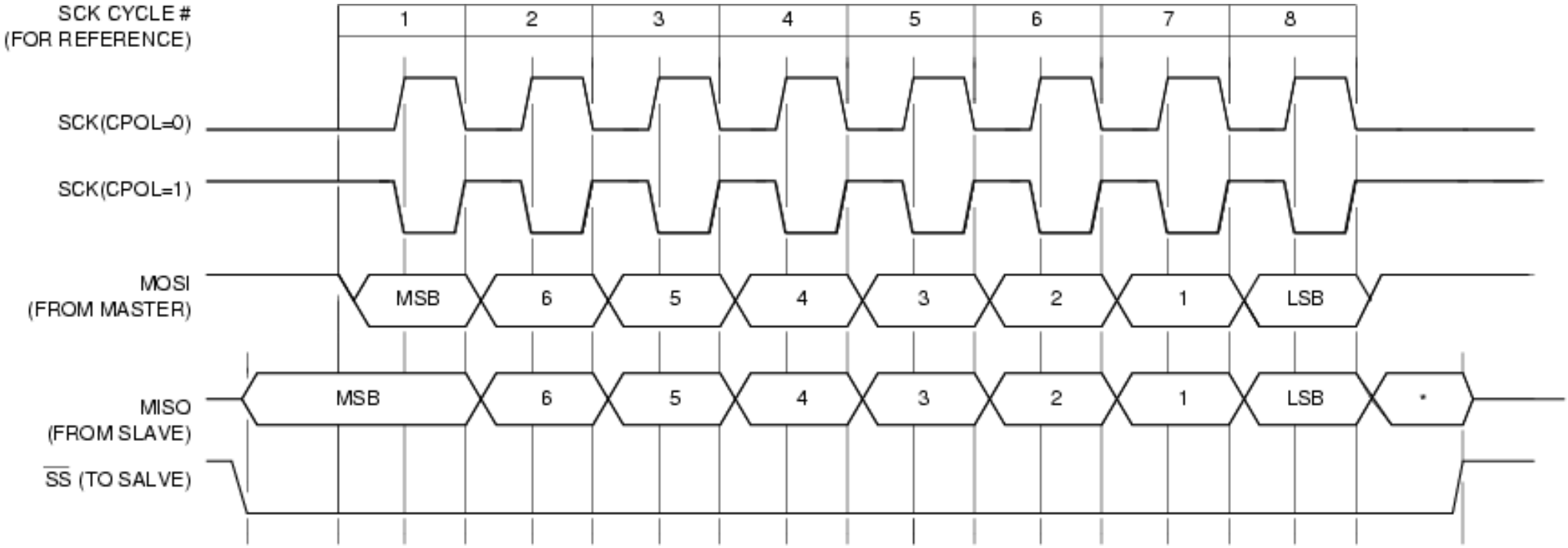
```
0000010e <L2>:
 10e:   00 01 98 20    andi    r12, 32    ; 0x20
 110:   00 00 98 0a    beqz    r12, 0x00124   ; 0x124 <L4>
 112:   00 00 20 15    or      r0, r0
 114:   00 01 58 00    lhi     r12, 0     ; 0x00
 116:   00 01 b8 2c    ori     r12, 44    ; 0x2c
 118:   00 01 28 01    movi    r4, 0x01   ; 0x1 <SB_PORT_BI_1_IOJ1_OD_OUTPUT>
 11a:   00 02 69 80    s18     0(r12), r4
 11c:   00 01 59 a0    lhi     r12, 416   ; 0x1a0
 11e:   00 01 b8 10    ori     r12, 16    ; 0x10
 120:   00 02 59 86    l18     r12, 6(r12)
 122:   00 02 79 a0    s18     0(r13), r12

00000124 <L4>:
 124:   00 00 21 64    jrs     r11
 126:   00 00 20 15    or      r0, r0
```

```
00000416 <uart_read>:
 416:   00 02 58 80    l18     r12, 0(r4)
 418:   00 01 98 01    andi    r12, 1 ; 0x01
 41a:   00 00 b9 fe    bnez    r12, 0x00416   ; 0x416 <uart_read>
 41c:   00 00 20 15    or      r0, r0
 41e:   00 02 48 82    l18     r4, 2(r4)
 420:   00 00 21 64    jrs     r11
 422:   00 00 20 15    or      r0, r0

00000424 <uart_write>:
 424:   00 02 58 80    l18     r12, 0(r4)
 426:   00 01 98 08    andi    r12, 8 ; 0x08
 428:   00 00 b9 fe    bnez    r12, 0x00424   ; 0x424 <uart_write>
 42a:   00 00 20 15    or      r0, r0
 42c:   00 02 6a 84    s18     4(r4), r5
 42e:   00 00 21 64    jrs     r11
 430:   00 00 20 15    or      r0, r0
```

2.4.2.2 Das serielle Peripherie-Interface (SPI) des SpartanMC

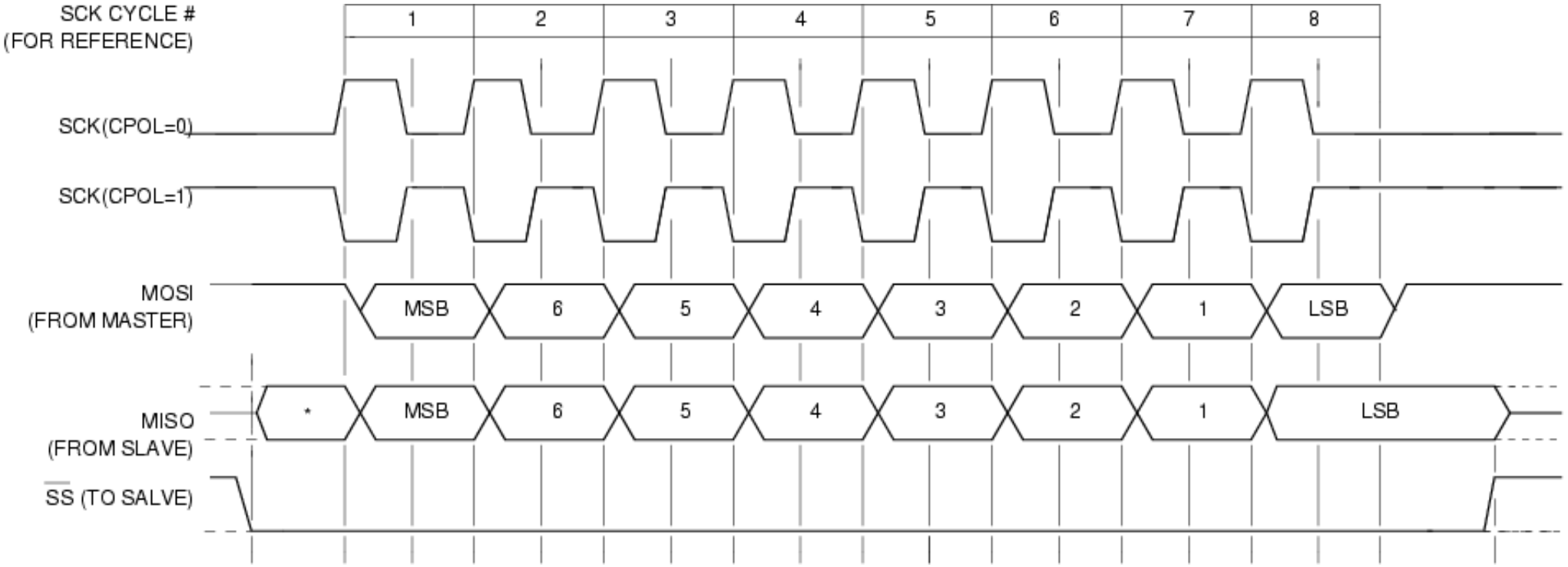


2.4.2.2 Das serielle Peripherie-Interface (SPI) (2)



- Polarität CPOL bei CPHA = 0

2.4.2.2 Das serielle Peripherie-Interface (SPI) (3)



- Polarität CPOL bei CPHA = 1

2.4.2.2 Das serielle Peripherie-Interface (SPI) (4)

- SPI basiert auf dem Master-Slave-Prinzip
- Ein Master kommuniziert immer nur mit einem Slave
- 15 Slave-Geräte an einen Master möglich
- vier Anschlüsse: Takt, Slave-Select, Daten-Ein- und Ausgang
- Beim Master sind Takt und Slave-Select Ausgänge
- Alle 4 Phasenlagen von Takt zu Daten mit CPOL und CPHA einstellbar
- Über die Datenleitungen MISO(Master In Slave Out) oder MOSI(Master Out Slave In) wird mit jedem Takt ein Bit in beide Richtungen gleichzeitig übertragen
- Breite des Datenwort von 1 bis 16 Bit einstellbar

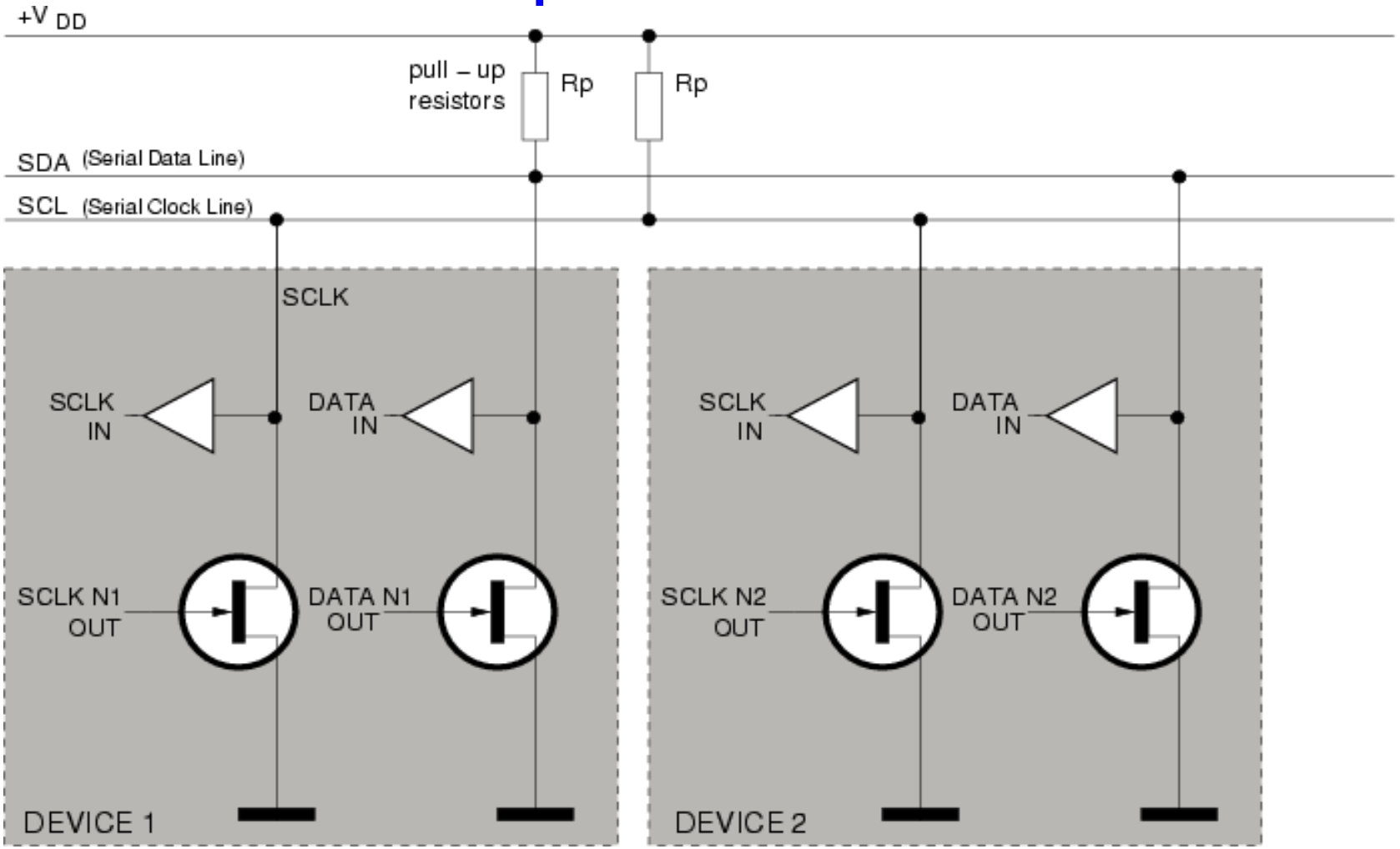
```
;
;
; Unterprogramm für SPI Datentransfer
;
; r12       zu sendende Daten (intern r4)
; r13       empfangene Daten (intern r5)
; r3        IO Basisadresse
;
spitfr:     s18       2(r3),    r4        ; Daten senden
            lhi       r8,       1         ; Maske für Statusbit
spi01:      l18       r9,       0(r3)     ; Status lesen
            and       r9,       r8
            beqz      r9,       spi01     ; Transfer noch nicht zu ende
            or        r0,       r0        ; NOP
            jrs       r11
            l18       r5,       4(r3)     ; Daten lesen im Delay-Slot
;
;
;
```

```
unsigned int spi_readwrite(spi_t* spi, unsigned int data){
	unsigned int result;
	spi->spi_data_out = data;
	while(IS_CLEAR(spi->spi_status,SPI_MASTER_CTRL_TRANS_EMPTY)){}
	result =  spi->spi_data_in;
	return result;
}

void spi_write(spi_t* spi, unsigned int data){
	while(IS_CLEAR(spi->spi_status,SPI_MASTER_CTRL_TRANS_EMPTY));
	spi->spi_data_out = data;
}

void spi_activate(spi_t* spi, unsigned int device){
	register    int temp;
	temp                = spi->spi_control & ~SPI_MASTER_CTRL_SLAVE;
	spi->spi_control = temp | ((device<<4)&SPI_MASTER_CTRL_SLAVE);
	while(IS_CLEAR(spi->spi_status,SPI_MASTER_CTRL_SS_SET));     // warten bis ss Signal
aktiv ist
}

void spi_deactivate(spi_t* spi){
	while(IS_CLEAR(spi->spi_status,SPI_MASTER_CTRL_TRANS_EMPTY)){}       // end of last
                                                                                // spi_write
	UNSET(spi->spi_control,(SPI_MASTER_CTRL_SLAVE|SPI_MASTER_CTRL_SS_ON));
	while(IS_SET(spi->spi_status,SPI_MASTER_CTRL_SS_ON));
}
```

```
#include <spi.h>
void spi_set_bitcnt(spi_t *spi, unsigned int bitcnt){
	UNSET(spi->spi_control ,SPI_MASTER_CTRL_BITCNT);
	SET(spi->spi_control , ((16 - bitcnt)<<8)&SPI_MASTER_CTRL_BITCNT);
}

#include <spi.h>
void spi_set_div(spi_t *spi, unsigned int div){
	UNSET(spi->spi_control ,SPI_MASTER_CTRL_DIV);
	SET(spi->spi_control , (div<<12)&SPI_MASTER_CTRL_DIV);
}

#include <spi.h>
void spi_set_cpah(spi_t *spi, unsigned int cpah){
	UNSET(spi->spi_control ,SPI_MASTER_CTRL_CPHA);
	SET(spi->spi_control , (cpah<<3)&SPI_MASTER_CTRL_CPHA);
}

#include <spi.h>
void spi_set_cpol(spi_t *spi, unsigned int cpol){
	UNSET(spi->spi_control ,SPI_MASTER_CTRL_CPOL);
	SET(spi->spi_control , (cpol<<2)&SPI_MASTER_CTRL_CPOL);
}
```

Weitere Funktionen für das SpartanMC SPI Interface finden Sie hier unter der Überschrift:
„C-Quellen der SPI-Include Funktionen“.

2.4.2.3 Das IIC Interface (I^2C) des SpartanMC



- Bus Prinzip der I^2C Master Slave Anordnung

2.4.2.3 Das IIC Interface (I^2C)

- Ein Master und mehrere Slave an einem Bus
- Auswahl des Slave durch Adresszyklen
- Slave mit 7-Bit- und 10-Bit-Adressen möglich
- Taktrate mit 16-bit-Vorteiler aus Systemtakt
- Datenregister für auszuführende Operation oder Daten
- Operationen: Startsymbol, Stopsymbol, Schreiben, Lesen
- Multi Master durch Arbitrierung

```
; Unterprogramme zum Lesen und Schreiben des i2c Master
; ======================================
; Vor dem Senden das Datenregister lesen
;                       Aufruf    Intern
;  IN:      Wert tx     r12       r4
;  IN:      io_base     r15       r7
;  OUT:     ack_fail    r14       r6
;  OUT:     wert rx     r13       r5
i2c_rx:   l18         r5,       4(r7)          ; Datenregister lesen
; Senden eines Bytes an den Master
;                       Aufruf    Intern
;  IN:      Wert        r12       r4
;  IN:      io_base     r15       r7
;  OUT:     ack_fail    r14       r6
i2c_tx:   s18         4(r7),    r4             ; Wert an i2c ausgeben
;
; Warten bis i2c wieder bereit ist
;                       Aufruf    Intern
;  IN:      io_base     r15       r7
;  OUT:     ack_fail    r14       r6     = 0 --> kein Fehler
i2c_wa:   L18         r6,       2(r7)          ; i2c status
          andi        r6,       0x180          ; 0x180 Maske für ir_sig und ack_fail
          beqz        r6,       i2c_wa         ; warten, bis ir_sig = 1
          or          r0,       r0             ; NOP
          jrs         r11
          andi        r6,       0x100          ; ack_fail (im Delay-Slot)
;
```

```
#include <i2c_master.h>

int i2c_master_reada(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int anz, unsigned int * datar){
	// Adresse senden
	i2c_m->txr	= (slave_addr << 1) | 1;   // bit0 = 1 ist read
	i2c_m->cmd	= I2C_STA | I2C_WR ;
	while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
	i2c_m->cmd	= I2C_IACK;
	if ((i2c_m->stat & I2C_RXACK) != 0) return 1;	//I2C_NO_ACK bei ADR read;

	while (anz >1) {
		i2c_m->cmd	= I2C_RD | I2C_ACK;
		while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
		i2c_m->cmd	= (I2C_IACK );

		*datar		= i2c_m->rxr;
		anz		= anz -1;
		datar++;
	}
	// letztes Datenbyte empfangen und STOP
	i2c_m->cmd	= I2C_RD | I2C_ACK | I2C_STO;
	while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
	i2c_m->cmd	= (I2C_IACK );

	*datar		= i2c_m->rxr;
	return I2C_OK;
}
```

```
#include <i2c_master.h>

int i2c_master_readn(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int anz, unsigned int * datar){
	// Adresse senden
	i2c_m->txr	= (slave_addr << 1) | 1;   // bit0 = 1 ist read
	i2c_m->cmd	= I2C_STA | I2C_WR ;
	while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
	i2c_m->cmd	= I2C_IACK;
	if ((i2c_m->stat & I2C_RXACK) != 0) return 1;	//I2C_NO_ACK bei ADR read;

	while (anz >1) {
		i2c_m->cmd	= I2C_RD | I2C_ACK;
		while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
		i2c_m->cmd	= (I2C_IACK );

		*datar		= i2c_m->rxr;
		anz		= anz -1;
		datar++;
	}
	// letztes Datenbyte empfangen und STOP
	i2c_m->cmd	= I2C_RD | I2C_NAK | I2C_STO;
	while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
	i2c_m->cmd	= (I2C_IACK );

	*datar		= i2c_m->rxr;
	return I2C_OK;
}
```

```
#include <i2c_master.h>

int i2c_master_wr_st(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int anz, unsigned int * dataw, unsigned int stop){
	// Adresse senden
	i2c_m->txr	= (slave_addr << 1) | 0;   // bit0 = 0 ist write
	i2c_m->cmd	= I2C_STA | I2C_WR ;
	while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
	i2c_m->cmd	= I2C_IACK;
	if ((i2c_m->stat & I2C_RXACK) != 0) return 2;	//I2C_NO_ACK bei ADR write
	unsigned int	err	= 3;
	while (anz >1) {
		i2c_m->txr	= *dataw;
		i2c_m->cmd	= I2C_WR ;
		while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
		i2c_m->cmd	= I2C_IACK;
		anz--;
		dataw++;
		if ((i2c_m->stat & I2C_RXACK) != 0) return err;	//I2C_NO_ACK im write data;
		err++;
	}
	// letztes Datenbyte senden und STOP
	i2c_m->txr	= *dataw;
	i2c_m->cmd	= I2C_WR | stop ;
	while ((i2c_m->stat & I2C_TIP) == I2C_TIP);
	i2c_m->cmd	= I2C_IACK;
	return ((i2c_m->stat & I2C_RXACK) != 0) ? err : I2C_OK;	//err = I2C_NO_ACK im write data;
}
```

```
#include <i2c_master.h>

int i2c_master_write(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int anz, unsigned int
* dataw){
	return i2c_master_wr_st(i2c_m, slave_addr, anz, dataw, I2C_STO);
}

#include <i2c_master.h>

int i2c_master_wr_rda(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int anzw,
unsigned int * dataw, unsigned int anzr, unsigned int * datar){
	unsigned int	err;
	err	= i2c_master_wr_st(i2c_m, slave_addr, anzw, dataw, 0);	// write ohne stop
	if (err != I2C_ACK) return err; //I2C_NO_ACK im wr_st;
	return i2c_master_reada(i2c_m, slave_addr, anzr, datar);
}

#include <i2c_master.h>

int i2c_master_wr_rdn(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int anzw,
unsigned int * dataw, unsigned int anzr, unsigned int * datar){
	unsigned int	err;
	err	= i2c_master_wr_st(i2c_m, slave_addr, anzw, dataw, 0);	// write ohne stop
	if (err != I2C_ACK) return err; //I2C_NO_ACK im wr_st;
	return i2c_master_readn(i2c_m, slave_addr, anzr, datar);
}
```

```
/*
 * Schreiben eines Byte des m24c08
 */
int i2c_m_write(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int memad, unsigned int dataw){
	volatile unsigned int slad= slave_addr | (memad >>8);  // A9 und A8 von memad einblenden
	volatile unsigned int madl	= memad & 0xff;			// A7 bis A0 von memad
	volatile unsigned int wrda[2]	= {madl, dataw};
	volatile unsigned int* pwrda	= &wrda;
	return i2c_master_write(i2c_m, slad, 2, pwrda);
}

/*
 * Lesen eines Byte des m24c08 (das Byte muss mit NAK beantwortet werden)
 */
int i2c_m_read(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int memad, unsigned int datar){
	volatile unsigned int slad= slave_addr | (memad >>8);  // A9 und A8 von memad einblenden
	volatile unsigned int madl	= memad & 0xff;			// A7 bis A0 von memad
	volatile unsigned int* pmadl	= &madl;
	volatile unsigned interr;
	err	= i2c_master_wr_rdn(i2c_m, slad, 1, pmadl, 1, datar);  // rd mit NAK beim letzten Byte
	return err;
}
```

```
/*
 * Lesen von 16 Byte des m24c08 (das letzte Byte muss mit NAK beantwortet werden)
 * (die unteren 4 Bit von mamad werden geloescht!)
 */
int i2c_m_read16(struct i2c_master *i2c_m, unsigned int slave_addr, unsigned int memad, unsigned
int datar){
	volatile unsigned int slad = slave_addr | (memad >>8);  // A9 und A8 von memad einblenden
	volatile unsigned int madl	= memad & 0xf0;		// A7 bis A0 von memad (durch 16 teilbar)
	volatile unsigned int* pmadl	= &madl;
	volatile unsigned int err;
	err	= i2c_master_wr_rdn(i2c_m, slad, 1, pmadl, 16, datar);// rd mit NAK beim letzten Byte
	return err;
}
```

Weitere Funktionen für das SpartanMC IIC Interface finden Sie hier unter der Überschrift:
„Funktionen für den Anwender aus der Datei "i2c_master.h"“.

2.4.3 Programmierbare Zeitgeber

- Es gibt 5 verschiedene Zeitgeber und einen Vorteiler (TIMER)
- **Timer** – Vorteiler für beliebige kleine Frequenz aus Systemtakt
- **Timer** kann kaskadiert werden
- **Timer** hat 18 Bit Zähler mit einstellbarem Endwert und ausschaltbarem Vorteiler von 2^1 bis 2^8 einstellbar.
- Zähler kann gelesen und geschrieben werden
- **Timer** kann für Wartezeiten eingesetzt werden

```
timer       =       0x1a050 * 2         ; IO Basisadresse
ti_c        =       0 * 2               ; Offset Steuerregister
ti_d        =       1 * 2               ; Offset maximaler Timerwert
ti_v        =       2 * 2               ; Offset aktueller Timerwert
ti_init     =       0x3                 ; mit VT=2^1 bei 25MHz 0,04us*2^1= 0,08us
ti_end      =       262143              ; 262143*0,08us = 20,97144ms
; Timer initialisieren
            lhi     r5,   %hi(timer)
            ori     r5,   %lo(timer)    ; IO Basisadresse
            movi    r6,   ti_init
            s18     ti_c(r5), r6        ; Timer initialisieren
            lhi     r6,   %hi(ti_end)
            ori     r6,   %lo(ti_end)   ; 20,97144ms
            s18     ti_d(r5), r6        ; Timer immer bis ti_end laufen lassen.
; Unterprogramm Wait mit input r13 (= intern r5) = Timer Zyklen
;             ====                        ============
wait:       lhi     r8,   %hi(timer)
            ori     r8,   %lo(timer)    ; IO Basisadresse
            xor     r9,   r9
            s18     ti_v(r8), r9        ; Start der Zeit
            or      r0,   r0            ; NOP
wait0:      l18     r9, ti_v(r8)
            sgeu    r9,   r5
            or      r0,   r0            ; NOP
            beqzc   wait0               ; Wert ist < r5
            jrs     r11
            or      r0, r0              ;NOP
```

2.4.3 Programmierbare Zeitgeber (3) RTI

- **RTI** Erzeugung von Interrups mit einer einstellbaren Frequenz
- **RTI** Vorteiler von 2^0 bis 2^{15}
- **RTI** Takt Eingang wählbar bei der Systemkonfiguration
- **RTI** Takt Quellen:
  - Systemtakt
  - DCM des FPGA
  - TIMER Modul
  - weiterer RTI Modul
- **RTI** Beispiel für Sekunden Takt:
  - Kette von 25 MHz --> TIMER Bit 17 --> RTI --> Interrupt
  - TIMER mit Vorteiler 2^4 und Zählerendwert 195313
  - RTI mit Vorteiler 2^3
  - 195313 * 128 = 25000064

```
; (2^17 + 64240) * 2^7    = 195312 * 128      = 24999936
; (2^17 + 64241) * 2^7    = 195313 * 128      = 25000064
; Eingetellt werden:
; Timer    Wert      = 195313
; Timer    Vorteiler = 2^4
; RTI      Vorteiler = 2^3          --> rtiout = 1 Hz
timer     =         0x1a058 * 2    ; Basisadresse TIMER
ti_c      =         0              ; Offset Steuerregister
ti_d      =         2              ; Offset maximaler Timerwert
ti_en     =         1              ; Timer freigabe Bit
ti_vt     =         2              ; Timer Vorteiler freigabe Bit
ti_v4     =         0xC            ; Vorteiler 2^4
rti       =         0x1a060 * 2    ; Basisadresse RTI
rtiini    =         0xf            ; RTI freigabe, Interrupt Freigabe und VT = 2^3
;
          lhi       r4,       %hi(timer)
          ori       r4,       %lo(timer)
          movi      r6,       ti_en | ti_vt | ti_v4
          s18       ti_c(r4), r6              ; Timer initialisieren
          lhi       r6,       %hi(195313)
          ori       r6,       %lo(195313)
          s18       ti_d(r4), r6              ; Timer immer bis 195313 laufen lassen.
;
          lhi       r4,       %hi(rti)
          ori       r4,       %lo(rti)
          movi      r6,       rtiini          ; RTI initialisieren
          s18       0(r4),    r6
```

2.4.3 Programmierbare Zeitgeber (5)

- Die Programmierung der Zeitgeber ist auch in C möglich
- Ein RTI Interrupt wird im folgenden Programm in C behandelt
- Ein Assembler Startupcode stellt dazu für alle Interrupts einen Prozedur Aufruf bereit
- Es gibt Prozeduren zur Freigabe und zum Sperren des Interrupts
- Es gibt Prozeduren zur Ausgabe auf das SFR_LED Register
- Der Zugriff auf den Timer und RTI wird über vorbereitete Strukturen realisiert
- Das Programm verändert die 7 LEDs im Sekunden Abstand
- Es kann auch über den „Startup Loader“ geladen werden

```
/*
 * SpartanMC LED Sekunden Zaehler
 * ==============================
 */

#include <system/peripherals.h>
#include <interrupt.h>
#include <led7.h>

unsigned int i;

/* Funktionen zur Interrupt Behandlung */
void isr00() {
    volatile int rtictrl;

    rtictrl = RTI_0->ctrl;          //Interrupt ruecksetzen

    i++;
    if (i>=128) i=0;
    led7_set(i);
    }
```

```
void isr01() {
        i = i;
       }

int main() {
        interrupt_disable();

       // Impuls mit einer Periodendauer von 0,125s aus 25MHz erzeugen. Fuer den Sekundenimpuls
       // an RTI0 bringen die Vorteiler vom TIMER = 2^4 und RTI0 = 2^3 den Wert 16 * 8 = 128.
       // Der Rest muss vom Haupteiler des Timer kommen und muss die Bedingung
       // 2^17 < Wert < (2^18 -1) erfüllen.
       //       2^17                        = 131072
       //       int(25000000 / 128)+1       = 195313  erfüllt die Bedingung
       //       2^18 – 1                    = 262143
       // Die Periodendauer des Impuls am TIMER Ausgang ergibt sich damit zu:
       // (1 / 25000000 Hz)*(int(25000000 / 128)+1) * 16 = 0,00000004s * 195313 * 16 = 0,12500032s

        TIMER_1->control= TIMER_EN|TIMER_PRE_EN|(TIMER_PRE_VAL*(4-1));
        // TI_PRE_VAL = 2^4
        TIMER_1->limit   =((25000000 / 128)+1);
        RTI_0->ctrl = RTI_EN|RTI_EN_INT|(RTI_PRE_VAL*3);      // RTI_PRE_VAL = 2^3
        interrupt_enable();

        while (1);
}
```

```
__asm__("\n.include \"const.s\"");

void led7_set(int i){
        __asm__("MOVI2S       SFR_LEDS, %0"::"r"(i));
}

int led7_get(){
        int i;
        __asm__("MOVS2I       %0,   SFR_LEDS":"=r"(i));
        return i;
}
```

```
/*
          00000
        5      1
        5      1
        5      1
         66666
        4      2
        4      2
        4      2
         33333
*/
// Ziffern                0   1   2   3   4   5   6   7   8   9   A   b   c   d   E   F
const unsigned char led7[16] ={
0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x58,0x5E,0x79,0x71};

void led7_hex(int i){
        unsigned char out;
        if(i <=15) {
                out = led7[i];
        } else {
                out = 0;
        }
        __asm__("MOVI2S      SFR_LEDS, %0"::"r"(out));
}
```

2.4.3 Programmierbare Zeitgeber (8)

Die Include Dateien timer.h und rti.h

```
#define TIMER_EN      (1<<0)
#define TIMER_PRE_EN  (1<<1)
#define TIMER_PRE_VAL (1<<2)       // *0 fuer 2^1  bis *7 fuer 2^8

typedef struct timer {
    volatile unsigned int control;
    volatile unsigned int limit;
    volatile unsigned int value;
} timer_regs_t;
```

```
#define RTI_EN        (1<<0)
#define RTI_EN_INT    (1<<1)
#define RTI_PRE_VAL   (1<<2)       // *0 fuer 2^0  bis  *15 fuer 2^15

typedef struct rti {
    volatile unsigned int ctrl;
} rti_regs_t;
```

2.4.3 Programmierbare Zeitgeber (9)

- Zeitmessung mit **Timer Capture** bis zur Flanke oder dem Zustand eines Signals
- Impulserzeugung oder Signalverzögerung mit Timer Compare
  - Impulbreitenmodulation möglich
- Zählen von Impulsen bis zu einem Signal mit **Timer Pulsakku**
- **Systemüberwachung** durch **Timer Watchdog**
  - Watchdog löst RESET aus
  - Beim Neustart wird im Statusregister der Alarm erkannt
  - zum Rückstellen ist ein Code notwendig
  - Code wird bei der Systemkonfiguration festgelegt

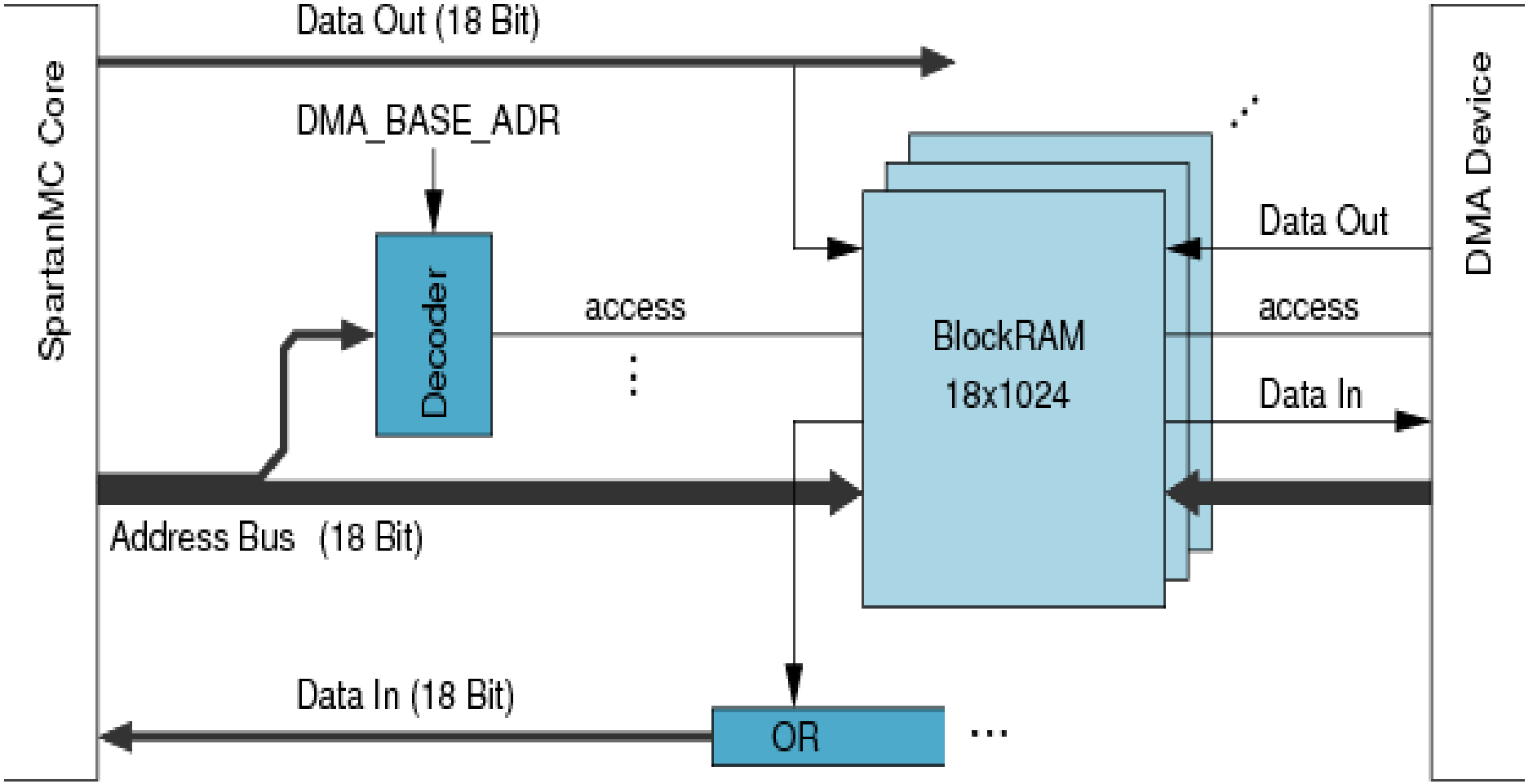
```
void main(void) {

      // Initialisierung der Geraete
      interrupt_disable();
      data2->wdctrl    = WATCHDOG_0->control;     // Reset Status des WD merken
#ifndef    SB_UART_LIGHT_MONITOR_INTERRUPT_SUPPORTED
      uart_wait_idle(UART_MONITOR);
      UART_MONITOR->ctrl_stat    = (UART_BPS_115200|UART_DATA_LEN_8|UART_TWO_STOP|UART_RX_EN|
UART_TX_EN);
      stdio_uart_open(UART_MONITOR);
#else
      stdio_uart_open(UART_LIGHT_MONITOR);
#endif

      TIMER_1->control= TIMER_EN|TIMER_PRE_EN|(TIMER_PRE_VAL*(VT_TIMER1-1));
      // TIMER_PRE_VAL = 2^4  (Fuer VSIM 2^1)
      TIMER_1->limit  = LI_TIMER1;               // getesteter Wert fuer eine Sekunde

      RTI_0->ctrl     = RTI_EN|RTI_EN_INT|(RTI_PRE_VAL*VT_RTI0);
      // RTI_PRE_VAL = 2^3 (Fuer VSIM 2^1)

      RTI_1->ctrl     = RTI_EN|RTI_EN_INT|(RTI_PRE_VAL*2);
      // RTI_PRE_VAL = 2^2

      ROT_0->edgsel   = ROT_RIGHT|ROT_LEFT|ROT_PUSH;
      // Positive Flanke fuer die Signale einstellen
      ROT_0->ie  = ROT_RIGHT|ROT_LEFT|ROT_PUSH;
      // Interrupt fuer die Signale freigeben
      ROT_0->counter  = 0;                       // Schrittzaehler auf Startwert setzen
```

```
      // Initialisierung beendet
      interrupt_enable();
      if (data2->wdctrl == 0) {
            printf("\f\r\n");          // Loesche Terminal
            printf(startMeldung);
            data2->pgnr     = 0;
            test_timer_rti();
            }
      else {
            if (data2->wdzyk > data2->zmax) data2->zmax = data2->wdzyk;
            if (data2->wdzyk < data2->zmin) data2->zmin = data2->wdzyk;
            printf("\33\[u\t\t\t WDT-Alarm bei \33\[1m%6u\33\[0m Zyklen",data2->wdzyk);
            // Die ESC-Sequenz \33\[u setzt den CU an die mit \33\[s gespeicherte Position.
            // Wenn diese Sequenz nicht arbeitet, dann hier durch \r\33\[2A ersetzen!
            printf(" (min=\33\[1;32m%6u\33\[0;37m max=\33\[1;31m%6u\33\[0;37m)\r\33\
[5A",data2->zmin,data2->zmax);
            // \33\[1m  -- Zeichen Fett   \33\[0m -- Zeichen normal
            // Die ESC-Sequenz \33\[5A setzt den CU um 5 Zeilen nach oben.
            data2->wdzyk    = ZYKSTART;
      }

      while (1) {
            data2->pgnr     = 1;
            test_wdt();
            data2->pgnr     = 2;
            test_rotta();
      }
}
```

```
/* Test des Watchdog Timers
 *
 * Die Schleife while (i > 0) i--; hat im GCC zur Zeit 6 Befehle.
 * Bei maximal 262143 = 0x3ffff Schleifenzyklen ergibt das ein Zeit von:
 * (1 / 35937500 Hz) * 6 * 262143 = 0,000000027826086956s * 6 * 262143 = 0,0437664834783s
 * Die Zeit des Watchdog Timers muss kleiner sein, damit ein Alarm ausgeloest
 * wird. Die Zeit ergibt sich zu (1 / 35937500 Hz) * VT * limit
 * (1 / 35937500 Hz) * 128 * 12000 = 0,0427408695652s ist kleiner bei 6 Befehlen!
 */
void test_wdt(void) {
volatile unsigned int i     = data2->wdzyk;
volatile unsigned int wdcnt = 0;
volatile unsigned char     d    = ' ';
     printf("\r\n\n2. Test des Watchdog Timers\r\n   Ende mit ESC\r\n");
     WATCHDOG_0->limit    = 12000;   // 262143 = 0x3ffff
     WATCHDOG_0->control  = WATCHDOG_EN | WATCHDOG_PRE_EN | WATCHDOG_PRE_128;
     printf(" limit = %5u ctrl = %5x\r\n\33\[s",WATCHDOG_0->limit,WATCHDOG_0->control);
     // Die ESC-Sequenz \33\[s merkt sich die CU-Position.
     while (d != 0x1b) {
          d = uart_getstat();
          i = data2->wdzyk;
          WATCHDOG_0->val_rst  = 0x12345; // Reset Pin
          while (i > 0) i--;    // Schleife wird durch Watchdog Timer abgebrochen!
          wdcnt = WATCHDOG_0->val_rst;
          WATCHDOG_0->val_rst  = 0x12345; // Reset Pin
          data2->wdzyk++;
          printf(" wdcnt = %5u\r",wdcnt);
          if (data->aktzeit != data->i) {
               putzeit();
               data->aktzeit = data->i;
          }
     }
     WATCHDOG_0->control  = 0;
}
```

```
Timer, RTI, Watchdog, LED, UART und Rotationstaster Test in C

1. Zeit auf der Konsole anzeigen
   Ende bei jeder Eingabe
   Stellen mit dem Rotationstaster oder
CTRL+B Stunden mit + stellen
CTRL+C Minuten mit + stellen
CTRL+D Sekunden mit + stellen
CTRL+E Ende Stellen

 Aktuelle Zeit: 12:08:23  Hallo ich bin SpartanMC 18

2. Test des Watchdog Timers
   Ende mit ESC
 limit = 12000 ctrl =    1B
 wdcnt = 11968  12:10:02 WDT-Alarm bei 25579 Zyklen (min=255979 max=255983)
SpMC loader v20120927

3. Test des Rotationstaster Moduls

   ESC = Programm Ende

0    0    1    1    2    2    3
0....5....0....5....0....5....0
                       ^


2. Test des Watchdog Timers
   Ende mit ESC
 limit = 12000 ctrl =    1B
 wdcnt = 11972  13:08:19 WDT-Alarm bei 255981 Zyklen (min=255956 max=255986)
SpMC loader v20120927
```

2.4.4 DMA Interfaces des SpartanMC



2.4.4 DMA Interfaces des SpartanMC
2.4.4.1 USB Interface

- Das USB 1.1 des SpartanMC wird vollständig auf der FPGA realisiert.
  - Es sind nur 3 Widerstände extern notwendig
  - In einem auf 18 Bit konfigurierten Blockram sind alle Register und Puffer realisiert
  - Ein Port des Blockram ist mit dem Datenbus verbunden
  - Am zweiten Port ist das USB Interface angeschlossen
  - Es gibt keine I/O Register
- Das USB 2.0 des SpartanMC realisiert nur ein 8 Bit UTMI Interfaces für den SMSC 3250
  - die Puffer sind mit Blockrams realisiert
  - es besitzt I/O Register für die Steuerung
- Leistungsmessung mit USB Ping Pong

2.4.4.2 CAN Interface

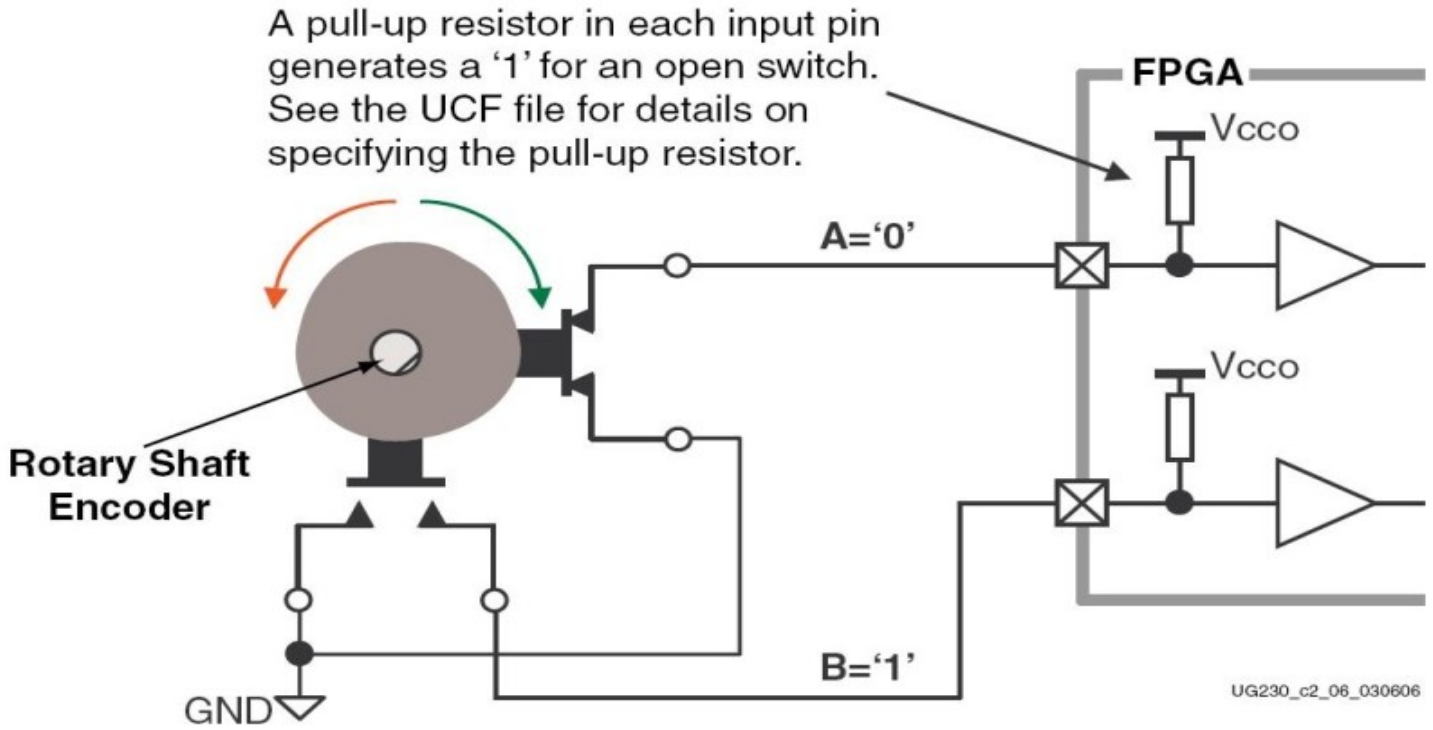
- Der CAN-Bus (Controller Area Network) ist ein asynchrones, serielles Bussystem
- 1983 von Bosch für die Vernetzung von Steuergeräten in Automobilen entwickelt
- bei 1 MBit/s 40 m, bei 500 kBit/s 100 m und bei 125 kBit/s 500 m möglich
- Modifikationen: CANopen, DeviceNet, J1939 (NMEA2000 und ISOBUS), CleANopen, SafetyBUS p, TTCAN, CANaerospace, ARINC 825, EnergyBus
- Überschneidung des Einsatz von SoC und CAN
- CAN Telegramme werden in DMA Blockram abgelegt
- CAN hat 15 I/O-Register

2.4.5 Spezial Interfaces für den SpartanMC

- In SoC Lösungen ist es möglich Spezial Interface mit Hardware zu unterstützen ohne dadurch zusätzliche Kosten zu verursachen.
- Die Verwendung in der Software wird dadurch einfacher.
- Es braucht keine Systemleistung für so einfache Dinge wie Endprellen verschwendet werden.
- Für den SpartanMC wird ein immer breiteres Spektrum solcher Interface zur Verfügung gestellt.
- Die meisten Lösungen unterstützen den Roboter Einsatz.

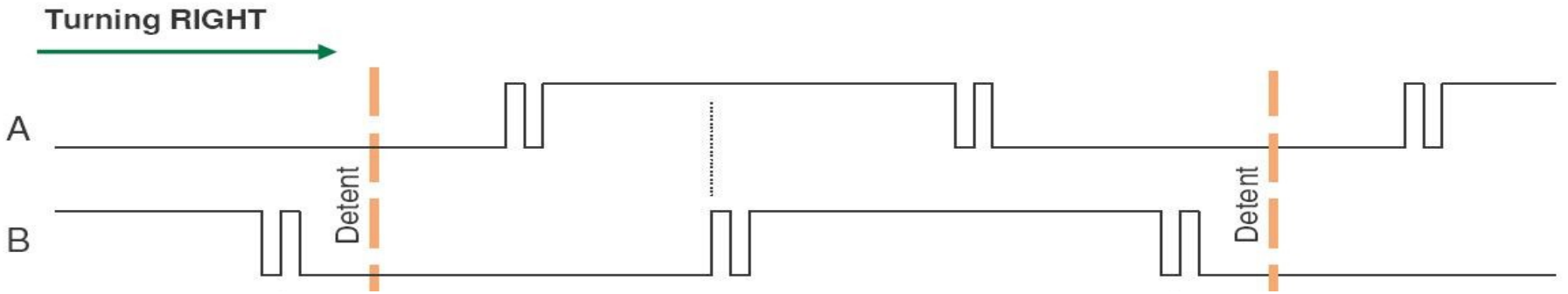
2.4.5.1 Rotationstaster (Sensor)

- Zur Erkennung der Drehrichtung und Drehzahl
- Zur Auswahl in „Ein stell Menüs“
- Zur Erkennung von Bewegungen mechanischer Teile



2.4.5.1 Rotationstaster (Sensor) 2

- Das Interface entprellt die Kontakte A und B und erzeugt daraus je ein Signal für rechts oder links Drehung.
- Alle diese Signale und das Drücken der Taste können Interrupt auslösen.
- Ein 18 Bit Zähler wird beim rechts drehen inkrementiert und beim links drehen decrementiert.
- Der Zähler kann zu jeder Zeit mit jedem beliebigen 18 Bit Wert geschrieben werden.

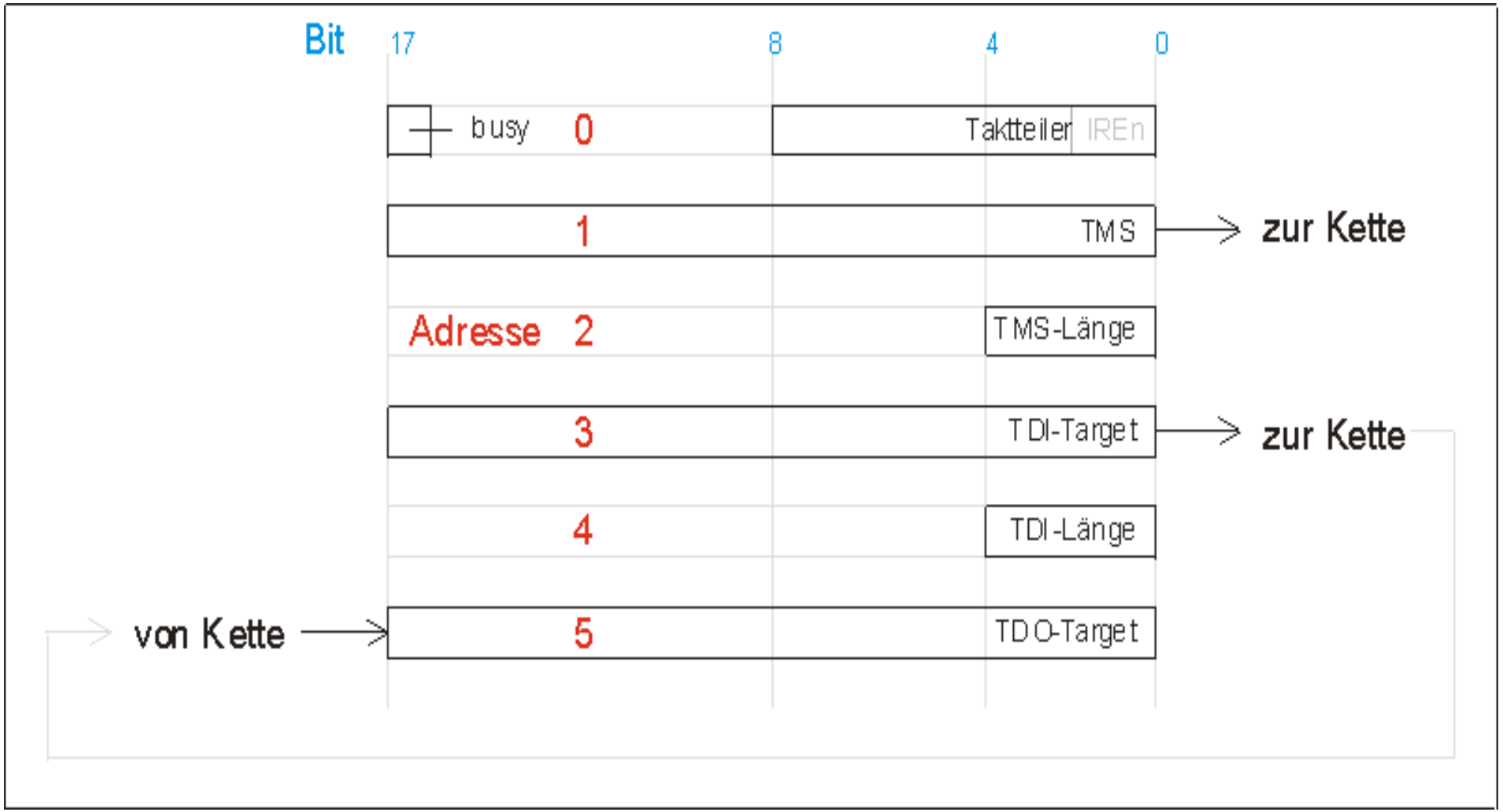


```
#include <system/peripherals.h>
#include <interrupt.h>


// Rotationstaster
void isr04() {
	volatile int	dat;
	volatile int	cnt;
	volatile int	irstat;
	irstat	= ROT_0->ir_stat;
	// Rechts und Links werden nach dem lesen von data geloescht
	dat	= ROT_0->data;		// Interrupt wird rueckgesetzt
	cnt	= ROT_0->counter;	// Zaehler laden
	if ((irstat & ROT_PUSH) != 0) {	// war Taster gedrueckt ?
		cnt  = 0;
		ROT_0->counter = 0;	// Schrittzaehler loeschen
		}
	}
```

2.4.5.2 JTAG Interface

- Viele Systeme haben Heute ein JTAG Interface.
- Zum Testen und Debuggen von elektronischer Hardware.
- Zum Einrichten und Auslesen programmierbarer Logik.
- Zur Startinitialisierung von programmierbarer Logik.
- Zum Datenaustausch zwischen verschiedenen Komponenten.
  - TDI Schieberegister zum Senden von Daten an ein JTAG Interface.
  - TMS Schieberegister zum Selektieren des Testmode.
  - TDO Schieberegister zum Empfang der Daten vom JTAG Interface.

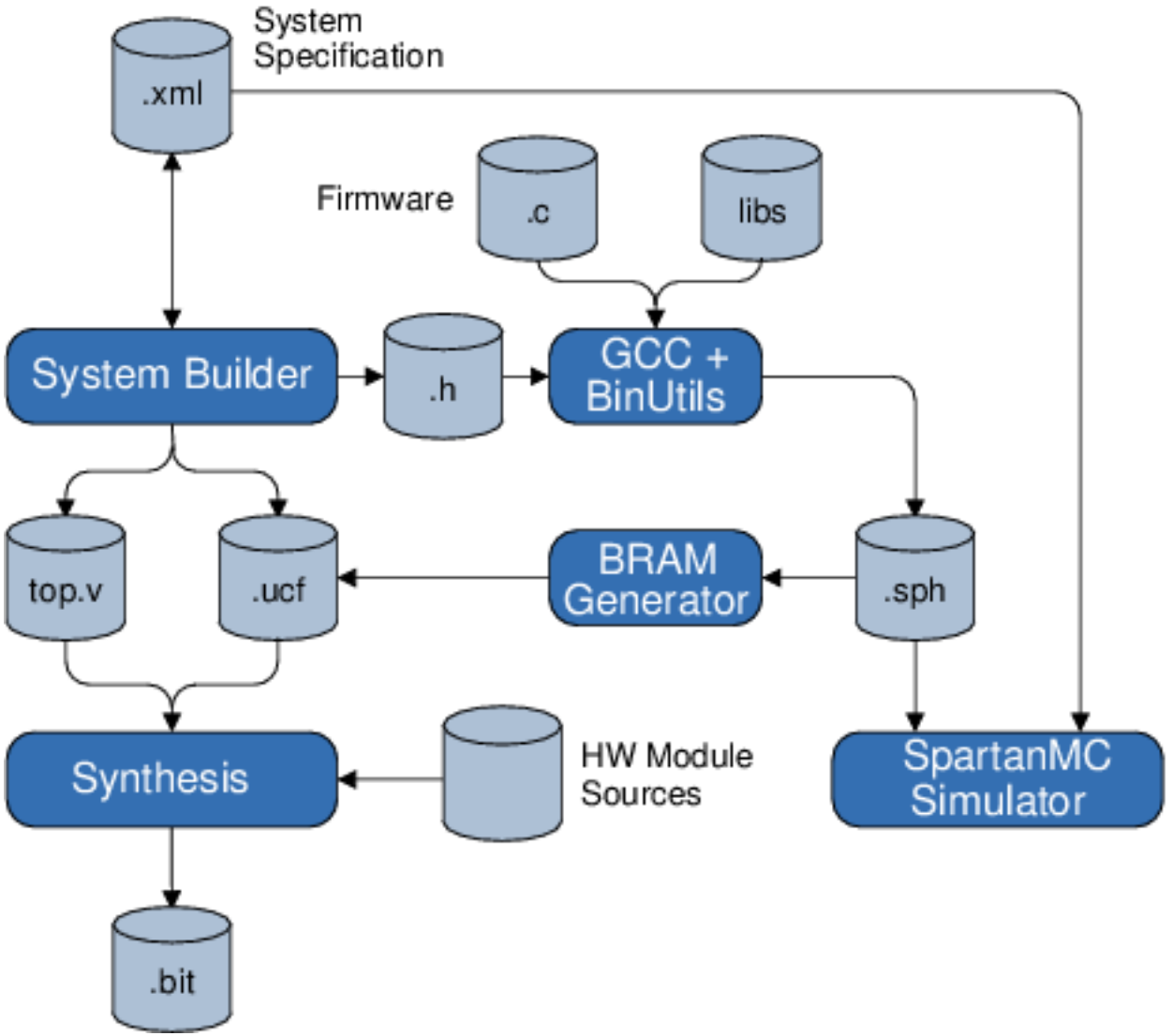
2.4.5.2 JTAG Interface (2)



2.4.5.3 Schrittmotoren Interface

- Das Interface übernimmt das Beschleunigen und Bremsen des Motors
- Die Anzahl der Beschleunigungsphasen (typisch 5) ist generierbar
- Nach der Initialisierung muss nur die Anzahl der Schritte und die Richtung eingestellt werden
- Es werden 3 Interrupts ausgelöst für:
  - Topspeed erreicht
  - Abbremsung beginnt
  - Stopp Zustand erreicht

3 Werkzeuge zur Programmierung und Konfiguration des SpartanMC



3.1 Der Assembler

Der Assembler ist Bestandteil der GNU – Binutils welche wie der GCC für den SpartanMC angepasst wurden. Alle Informationen dazu können also den im Internet vorhandenen Dokumentationen entnommen werden.

http://en.wikipedia.org/wiki/GNU_Assembler
http://wwwpub.zih.tu-dresden.de/~ss17/wiki/www.mr.inf.tu-dresden.de/wiki/tiki-index0af9.html

Auf den Seiten finden Sie alle wichtigen Informationen zur Nutzung des Assembler, Linker und aller anderen Tools. Für den SpartanMC ist eigentlich nur die Syntax der Assembler Direktiven von Interesse, um die List-Dateien zu verstehen oder selbst geschriebene Assemblerprogramme zu erstellen. Der Aufruf der Tools erfolgt automatisch durch das Entwicklungssystem für den SpartanMC. Da sowohl die Hardware als auch die wichtigsten Komponenten der Tool-Kette selbst im Team erstellt wurden, können natürlich auch in allen Teilen Fehler sein. Wenn etwas nicht funktioniert kann der Fehler im eigenen Programm, im Compiler oder auch in der Hardware sein. Um den Fehler zu finden ist dann fast immer die Assemblerliste des übersetzten C-Programms notwendig. Mit der Befehlsbeschreibung der Maschinenbefehle kann man dann überprüfen ob die aus dem C-Programm entstandene Befehlsfolgen auch wirklich das machen, was mit dem C-Programm realisiert werden soll. Ist das der Fall, muss herausgefunden werden welcher Befehl in der Hardware nicht das macht, was in der Beschreibung steht oder welches Ein- Ausgabegerät nicht richtig arbeitet. Ein Verständnis für die Assemblersprache des SpartanMC ist bei der Fehlersuche also unbedingt notwendig.

3.1.1 Direktiven

Direktive	Bedeutung	Beispiel
.ascii	Deklarierung von Zeichenketten	msg: .ascii "Hello, world!\r\n" ; Die Zeichenkette hat 15 Byte
.asciz	Deklarierung von Zeichenketten, die automatisch mit einer binären 0 abgeschlossen werden.	msg: .ascii "Hello, world!\r\n" ; Die Zeichenkette hat 16 Byte
.equ .set =	Einem symbolichen Namen einen Wert zuweisen. Dem Symbol egon wird in allen 3 Varianten der Wert 20 zugewiesen.	.equ egon, 10 * 2 .set egon, 0x14 egon = 0b10100
wert = .	Dem Symbol wert wird der Wert des aktuellem Adresspointers zugeordnet	msg: .ascii "Hello, world!\r\n" len = . - msg ; Die Länge der Zeichenkette wird len zugeordnet
.data	Beginn eines Feldes zur Deklaration von Programmvariablen und Konstanten.	
.w18	Reservierung eines Datenelements mit 18 Bit	.w18 egon ; Auf der Speicherzelle steht der Wert 20
.text	Beginn eines Feldes mit Maschinen Befehlen.	

3.1.1 Direktiven (2)

Direktiven	Bedeutung	Beispiel
.skip	Reservierung einer Anzahl von Byte im Datenbereich	buffer: .skip 512 ; es werden 512 Byte reserviert
.align	Auffüllen des Speichers mit Byte bis zur nächsten möglichen Adresse für den dahinter folgenden Datentyp	.asciz „0123“ .align .w18 0x3ffff

Sonderzeichen in Zeichenketten	**In Zeichenkette .ascii[z] einfügen**
****	****
“	**\“**
0xa <Line Feed>	**\n**
0x9 <Horizontal Tabulator>	**\t**
0x8 <Backspace>	**\b**
0xd <Carrriage Return>	**\r**
0xc <Form Feed>	**\f**

3.1.2 Include

- Include Dateien können Makros oder ganze .text Segmente und/oder .data Segmente enthalten. Der Inhalt der inc-Datei wird dann immer an der Stelle der a18-Datei eingefügt, an welcher sich der Include-Aufruf befindet. Der Aufruf erfolgt entweder:
- **.include "Dateiname.inc"** es wird der Pfad des inc-Files wie folgt angegeben, oder ohne Pfad, dann steht die zu verwendete inc-Datei im Verzeichnis der C Header Dateien. .include "../lib_obj/src/interrupt/interrupt.s" bindet in ein Programm den Quellkode der Interruptbearbeitng ein, wie er als Objekt in C-Programmen eingebunden wird.

3.1.2 Include (2)

Mit dem Befehl

.include „Verz\test3.inc“

ist es möglich auf inc-Dateien zuzugreifen, welche sich in Unterverzeichnissen befinden. Geladen werden können:

- Konstanten des SpartanMC.
- Konstanten für die Uart und andere Interface.
- Eine Interruptbehandlung.
- Funktionen.
- Macros für den SpartanMC.
- Startup Assembler Programme zum laden in C-Programmen.

3.2 Der GCC
3.2.1 Konsolen Funktionen

- 1. **uart_wait_idle(UART_MONITOR);**

  Die Funktion wartet auf den Ruhezustand einer UART, deren Adresse mit dem Namen der Uart im jConfig als Argument (hier UART_MONITOR) in den Klammern angegeben wird. Die Funktion wartet, bis die Schieberegister von Rx und Tx sowie RESET Funktionen abgeschlossen sind. Danach kann dann bei unidirektionalen Datenübertragungen die Datenrichtung geändert werden oder eine andere Datenrate, oder andere um Initialisierungen vorgenommen werden. Bei Verwendung der UART_LIGHT darf diese Funktion nicht aufgerufen werden. Wird die Funktion beim Systemstart vor der 1. Verwendung der Uart nicht aufgerufen, dann werden die Daten verfälscht, wenn der Ruhezustand noch nicht eingenommen wurde!

3.2 Der GCC
3.2.1 Konsolen Funktionen (2)

- 2. **stdio_uart_open(UART_MONITOR);**

  Die Funktion verknüpft die Uart mit der Adresse UART_MONITOR mit den STDIO Operationen, die in den folgenden Funktionen beschrieben werden.

- 3. **putchar(zeichen);**

  Ausgabe eines 8 Bit Wertes an das STDIO Gerät. Der folgende Code sendet eine ASCII "0".

```
unsigned char   zeichen;

zeichen = 0x30;

putchar(zeichen);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (3)

- 4. **zeichen = getchar();**

  Warten auf die Eingabe eines 8 Bit Wertes vom STDIO Gerät. Der folgende Code übergibt den eingegebenen Wert der Variablen "zeichen".

```
unsigned char   zeichen;
zeichen = getchar();
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (4)

- 5. **getchar_nb(&zeichen);**

  Testen, ob am STDIO Gerät ein Zeichen eingegeben wurde. Im folgenden Beispiel liefert die definierte Funktion eine 0x0 wenn nichts eingegeben wurde oder den Code des gesendeten Zeichens.

```
// z=uart_getstat() // z=0 keine Eingabe
unsigned char uart_getstat(void){
        unsigned char   zeichen = 0;
        getchar_nb(&zeichen);
        return zeichen;
}
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (5)

- 6. **printf("\r\n Wert = 0x%6x\n",out_value);**

  Senden von Zeichenketten und Anzeige von Werten auf dem STDIO Gerät. Es wird die übliche C-Funktion realisiert, es können aber nur maximal 2 Werte pro Aufruf angezeigt werden. Die Anzeige von Werten im LONG Format ist nicht möglich!

```
unsigned int out_value = 0x12345;
printf("\r\n\t  Hallo ich bin SpartanMC 18   \r");
printf("\r\n\t  SpartanMC 18 TU-Dresden InfMR\r");
printf("\r\n\t  Wert = 0x%6x\n",out_value);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (6)

- 7. **print_lx(**unsigned long wert, unsigned int anz**);**

  Anzeigen von long Werten (36 Bit) hexadezimal auf dem STDIO Gerät. Mit dem 2. Parameter wird die Anzahl der Anzeigestellen festgelegt. Es sind Werte von 0 bis 9 möglich. Bei 9 werden immer alle Stellen angezeigt und bei 0 nur die Stellen ungleich 0. Bei 1 wird nur die Stelle 16^0 immer angezeigt.

```
#include <io_funktionen.h>
unsigned long out_value = 0x123456789;
printf("\r\n 36 Bit Wert = 0x");
print_lx(out_value, 1);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (7)

- 8. **print_ld(**unsigned long wert, unsigned int sig, unsigned int anz**);**

  Anzeigen von long Werten (36 Bit) dezimal auf dem STDIO Gerät. Mit dem 3. Parameter wird die Anzahl der Anzeigestellen festgelegt. Es sind Werte von 0 bis 11 möglich. Bei 11 werden immer alle Stellen angezeigt und bei 0 nur die Stellen ungleich 0. Bei 1 wird nur die Stelle 10^0 immer angezeigt. Mit sig=0 wird wert ohne Vorzeichen angezeigt. Für ungleich 0 wird das Vorzeichen und der Betrag angezeigt.

```
#include <io_funktionen.h>

unsigned long out_value = -34359738368;

printf("\r\n 36 Bit Wert = ");

print_ld(out_value, 1, 1);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (8)

- 9. **get_lx(**unsigned int pos**);**

  Eingabe von long Werten (36 Bit) hexadezimal mit dem STDIO Gerät. Mit dem Parameter pos wird die Anzahl der Eingabestellen festgelegt, bis zu der nach ENTER ein TAB auf die Konsole ausgegeben wird. Mit BS kann die Eingabe korrigiert werden.

```
#include <io_funktionen.h>

unsigned long in_value;

printf("\r\n 36 Bit Wert = 0x");

in_value = get_lx(4);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (9)

- 10. **get_ld(**unsigned int sig, unsigned int pos**);**

  Eingabe von long Werten (36 Bit) dezimal mit dem STDIO Gerät. Mit dem Parameter pos wird die Anzahl der Eingabestellen festgelegt, bis zu der nach ENTER ein TAB auf die Konsole ausgegeben wird. Mit BS kann die Eingabe korrigiert werden. Mit sig = 0 wird ein Wert ohne Vorzeichen beginnen mit einer Ziffer erwartet. Bei ungleich 0 muss die Eingabe mit „+“ oder „-“ anfangen.

```
#include <io_funktionen.h>

unsigned long in_value;

printf("\r\n 36 Bit Wert = ");

in_value = get_ld(1, 4);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (10)

- 11. **get_18(**char *meld, unsigned int pos**);**

  Eingabe von Werten (18 Bit) hexadezimal mit dem STDIO Gerät. Mit dem Parameter pos wird die Anzahl der Eingabestellen festgelegt, bis zu der nach ENTER ein TAB auf die Konsole ausgegeben wird. Mit BS kann die Eingabe korrigiert werden. Der 1. Parameter ist eine Zeichenkette zur Eingabeaufforderung. Bei der Eingabe von Werten > 0x3ffff wird eine Fehlermeldung angezeigt und erneut zur Eingabe aufgefordert.

```
#include <io_funktionen.h>

unsigned long in_value;

in_value = get_18(„\r\n 18 Bit = 0x“, 4);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (11)

- 12. **get_18d(**char *meld, unsigned int sig, unsigned int pos**);**

  Eingabe von Werten (18 Bit) dezimal mit dem STDIO Gerät. Mit dem Parameter pos wird die Anzahl der Eingabestellen festgelegt, bis zu der nach ENTER ein TAB auf die Konsole ausgegeben wird. Mit BS kann die Eingabe korrigiert werden. Der 1. Parameter ist eine Zeichenkette zur Eingabeaufforderung. Bei der Eingabe von Werten die mehr als 18 Bit benötigen wird eine Fehler-meldung angezeigt und erneut zur Eingabe aufgefordert. Mit sig = 0 wird ein Wert ohne Vorzeichen beginnen mit einer Ziffer erwartet. Bei ungleich 0 muss die Eingabe mit „+“ oder „-“ anfangen.

```
#include <io_funktionen.h>

unsigned long in_value;

in_value = get_18d(„\r\n 18 Bit = “, 1, 4);
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (6)

```
/* Print 36 Bit Hexadezimal mit Vorgabe der Anzeigestellen
 * (nur wenn die voran gehenden Stellen 0 sind, werden sie nicht angezeigt) */
void   print_lx(unsigned long wert, unsigned int anz) {
    unsigned long  mask4high  = 0xf00000000;
    unsigned long  tetrade;
    unsigned int   stellen;
    unsigned int   pos = 9 - anz;
    unsigned int   pos0no  = 0;    // Merken, wenn Stelle != 0 gefunden wurde.
    unsigned char  zeichen;
    for (stellen = 0; stellen < 9; stellen++) {
        tetrade = wert & mask4high;
        wert    = wert << 4;       // naechten 4 Bit nach high
        tetrade = tetrade >> 32;  // in die unteren 4 Bit verschieben
        zeichen = tetrade;
        if (zeichen > 9) {
            zeichen = zeichen + 7; // 10 + 7 + 0x30 = 0x41
        }
        if ((stellen >= pos) || (zeichen != 0) || (pos0no != 0)) {
            pos0no  = 1;
            zeichen = zeichen + 0x30;
            putchar(zeichen);
        }
    }
}
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (7)

```
/* Input 36 Bit Hexadezimal */
unsigned long   get_lx(unsigned int pos) {
        unsigned long   wert    = 0;
        unsigned int    ok      = 0;
        unsigned int    stellen = 0;
        unsigned char   zeichen;
        do {
                zeichen = getchar();
                if (zeichen == 0x0d) ok =1;                     // ENTER
                if ((zeichen == 0x08) && (stellen > 0)) {       // BS
                        stellen--;
                        wert    = wert >> 4;
                        putchar(0x8);
                        putchar(' ');
                        putchar(0x8);
                        }
                if (stellen < 9) {
                        if ((zeichen >= '0') && (zeichen <= '9')) {
                                wert    = wert << 4;
                                wert    = wert + (zeichen - '0');
                                stellen++;
                                putchar(zeichen);
                                }
```

3.2 Der GCC
3.2.1 Konsolen Funktionen (7)

```
            // Wenn keine Ziffer auf A bis F und a bis f pruefen
                        if (zeichen >= 'A') {
                                    // Kleine Buchstaben in grosse wandeln
                                    zeichen = zeichen & 0xdf;
                                    if (zeichen <= 'F') {
                                                wert    = wert << 4;
                                                wert    = wert + (zeichen - 0x37);
                                                stellen++;
                                                putchar(zeichen);
                                                }
                                    }
                        }
            }
            while (ok == 0);                    // Eingabe beendet ?
            if (stellen < pos) putchar(0x9); // TAB
            return wert;
}
```

```
void main() {
    interrupt_disable();
    uart_wait_idle(UART_MONITOR);
    UART_MONITOR->ctrl_stat   = UART_RX_EN|UART_TX_EN|UART_TWO_STOP|
                                UART_DATA_LEN_8|UART_BPS_115200;
    stdio_uart_open(UART_MONITOR);
    interrupt_enable();
    printf("\r\n\n 36 Bit Summen berechnen\
        \r\n   i \tSummand 1 \tSummand 2 \tSumme 9 Stellen\t\
                 Summe mit mindestens einer Stelle\r\n");
    unsigned int   z   = 0;           // Index fuer Sumanden
    unsigned long  sum = 0;           // Summe     36 Bit
    unsigned long  su1 = 0;           // Summand 1 36 Bit
    unsigned long  su2 = 0;           // Summand 2 36 Bit
    while  (1) {
        printf("\r\n %3d\t0x",z);
        su1 = get_lx(6);
        printf("\t+ 0x");
        su2 = get_lx(4);
        printf("\t= 0x");
        sum = su1 + su2;
        print_lx(sum, 9);
        printf("\t = 0x");
        print_lx(sum, 1);
        z++;
    }
}
```

36 Bit Summen berechnen

i	Summand 1		Summand 2	Summe 9 Stellen	Summe mit einer Stelle
0	0x0	+	0x0	= 0x000000000	= 0x0
1	0x1	+	0x1	= 0x000000002	= 0x2
2	0x12	+	0x12	= 0x000000024	= 0x24
3	0x123	+	0x123	= 0x000000246	= 0x246
4	0x1234	+	0x1234	= 0x000002468	= 0x2468
5	0x12345	+	0x12345	= 0x00002468A	= 0x2468A
6	0x123456	+	0x123456	= 0x0002468AC	= 0x2468AC
7	0x1234567	+	0x1234567	= 0x002468ACE	= 0x2468ACE
8	0x123456789	+	0x123456789	= 0x2468ACF12	= 0x2468ACF12
9	0x23456789A	+	0x23456789A	= 0x468ACF134	= 0x468ACF134
10	0x3456789AB	+	0x3456789AB	= 0x68ACF1356	= 0x68ACF1356
11	0x456789ABC	+	0x456789ABC	= 0x8ACF13578	= 0x8ACF13578
12	0x56789ABCD	+	0x56789ABCD	= 0xACF13579A	= 0xACF13579A
13	0x6789ABCDE	+	0x6789ABCDE	= 0xCF13579BC	= 0xCF13579BC
14	0x789ABCDEF	+	0x789ABCDEF	= 0xF13579BDE	= 0xF13579BDE
15	0x89ABCDEF0	+	0x89ABCDEF0	= 0x13579BDE0	= 0x13579BDE0
16	0x				

```
void main() {
    interrupt_disable();
    uart_wait_idle(UART_MONITOR);
    UART_MONITOR->ctrl_stat   = UART_RX_EN|UART_TX_EN|UART_TWO_STOP|
                                UART_DATA_LEN_8|UART_BPS_115200;
    stdio_uart_open(UART_MONITOR);
    interrupt_enable();
    printf("\r\n 36 Bit Summen berechnen\r\n   i \tSummand 1 \tSummand 2\
           \tSumme 9 Stellen\t Summe dezimal\r\n");
    unsigned int   z   = 0;        // Index fuer Sumanden
    unsigned long  sum = 0;        // Summe     36 Bit
    unsigned long  su1 = 0;        // Summand 1 36 Bit
    unsigned long  su2 = 0;        // Summand 2 36 Bit
    while  (1) {
        printf("\r\n %3d\t",z);
        su1 = get_ld(1, 7);        // dez Eingabe
        printf("\t+ ");
        su2 = get_ld(1, 5);        // dez Eingabe
        printf("\t= 0x");
        sum = su1 + su2;
        print_lx(sum, 9);          // hex Ausgabe
        printf("\t = ");
        print_ld(sum, 1, 11);      // dez Ausgabe
        z++;
    }
}
```

36 Bit Summen berechnen

```
i    Summand 1        Summand 2        Summe 9 Stellen   Summe dezimal

0    +5               + +5             = 0x00000000A     = +00000000010
1    -5               + +10            = 0x000000005     = +00000000005
2    +20              + +20            = 0x000000028     = +00000000040
3    +34359738360     + +7             = 0x7FFFFFFFF     = +34359738367
4    +34359738360     + +8             = 0x800000000     = -34359738368
5    -34359738367     + +7             = 0x800000008     = -34359738360
6    +20              + +0             = 0x000000014     = +00000000020
7
```

3.2 Der GCC
3.2.2 Funktion für 36 Bit Ergebnis vom MUL

```
__asm__("\n.include \"const.s\"");
/*
 * Multiplikation mit 36 Bit Produkt aus 18 Bit * 18 Bit
 */

long mul36(int fa1, int fa2){
              long prod;
   unsigned int   prodl;
   prodl = fa1;
   __asm__("MUL      %0,      %1":"+r"(prodl):"r"(fa2));
   __asm__("MOVS2I   %0,      SFR_MUL":"=r"(prod));
   prod  = prod << 18;     // SFR Inhalt nach HIGH schieben
   prod  = prod | prodl;   // LOW Einblenden
   return   prod;          // 36 Bit Produkt
}
```

```
void main() {

    interrupt_disable();
    uart_wait_idle(UART_MONITOR);
    UART_MONITOR->ctrl_stat   = UART_RX_EN|UART_TX_EN|UART_TWO_STOP|
                                UART_DATA_LEN_8|UART_BPS_115200;
    stdio_uart_open(UART_MONITOR);
    interrupt_enable();

    printf("\r\n\n MUL 18 Bit * 18 Bit = 36 Bit\r\n");

    while  (1) {
        unsigned long  prod    = 0;        // Produkt   36 Bit
        unsigned int   fa1     = 0;        // Faktor  1 18 Bit
        unsigned int   fa2     = 0;        // Faktor  2 18 Bit

        fa1 = get_18("\r\n Faktor 1 18 Bit = 0x", 6);
        fa2 = get_18("\r\n Faktor 2 18 Bit = 0x", 3);

        prod   = mul36(fa1, fa2);

        printf("\r\n Produkt  36 Bit = 0x");
        print_lx(prod, 1);

        printf("\r\n");
    }
}
```

```
/*
 * Input 18 Bit Hexadezimal
 */
unsigned long  get_18(char *meld, unsigned int pos) {
   unsigned long  wert18   = 0;
   do {
      printf(meld);
      wert18   = get_lx(pos);    // 36 Bit Eingabefunktion
      if (wert18 > 0x3ffff) printf("Wert > 18 Bit!");
   }
   while (wert18 > 0x3ffff);
   return   wert18;
}
```

```
MUL 18 Bit * 18 Bit = 36 Bit

Faktor 1 18 Bit = 0x2
Faktor 2 18 Bit = 0x4
Produkt  36 Bit = 0x8

Faktor 1 18 Bit = 0x20
Faktor 2 18 Bit = 0x20
Produkt  36 Bit = 0x400

Faktor 1 18 Bit = 0x222
Faktor 2 18 Bit = 0x222
Produkt  36 Bit = 0x48C84

Faktor 1 18 Bit = 0x4444
Faktor 2 18 Bit = 0x4444
Produkt  36 Bit = 0x12343210

Faktor 1 18 Bit = 0x
```

```
void main() {
   interrupt_disable();
   uart_wait_idle(UART_MONITOR);
   UART_MONITOR->ctrl_stat = UART_RX_EN|UART_TX_EN|
         UART_TWO_STOP|UART_DATA_LEN_8|UART_BPS_115200;
   stdio_uart_open(UART_MONITOR);
   interrupt_enable();
   printf("\r\n\n MUL 18 Bit * 18 Bit = 36 Bit Dezimal\r\n");
   while (1) {
      unsigned long  prod  = 0;  // Produkt   36 Bit
      unsigned int   fa1   = 0;  // Faktor  1 18 Bit
      unsigned int   fa2   = 0;  // Faktor  2 18 Bit
      fa1   = get_18d("\r\n Faktor 1 18 Bit = ", 1, 6);
      fa2   = get_18d("\r\n Faktor 2 18 Bit = ", 1, 3);
      prod  = mul36(fa1, fa2);
      printf("\r\n Produkt  36 Bit = ");
      print_ld(prod, 1, 1);
      printf("\r\n");
   }
}
```

```
/*
 * Input 18 Bit Dezimal
 */
long   get_18d(char *meld, unsigned int sig, unsigned int pos) {
    long   wert18 = 0;
    do {
        printf(meld);
        wert18 = get_ld(sig, pos); // 36 Bit Eingabefunktion
        if (((sig == 0) && (wert18 > 262143)) || ((sig != 0) &&
                ((wert18 > 131071) || (wert18 < -131072)))){
            printf("\tWert > 18 Bit!");
        }
    }
    while (((sig == 0) && (wert18 > 262143)) || ((sig != 0) &&
                ((wert18 > 131071) || (wert18 < -131072))));
    return wert18;
}
```

```
MUL 18 Bit * 18 Bit = 36 Bit Dezimal

Faktor 1 18 Bit = +3
Faktor 2 18 Bit = -4
Produkt  36 Bit = -12

Faktor 1 18 Bit = +131071
Faktor 2 18 Bit = +131071
Produkt  36 Bit = +17179607041

Faktor 1 18 Bit = +131072      Wert > 18 Bit!
Faktor 1 18 Bit = +131073      Wert > 18 Bit!
Faktor 1 18 Bit = -131072
Faktor 2 18 Bit = -131072
Produkt  36 Bit = +17179869184

Faktor 1 18 Bit = -131073      Wert > 18 Bit!
Faktor 1 18 Bit = +1024
Faktor 2 18 Bit = +2
Produkt  36 Bit = +2048

Faktor 1 18 Bit =
```

3.2 Der GCC
3.2.3 Funktionen für das SFR_LED Register

```
#include <led7.h>

/*
 * Werte an 7 LEDs oder 7 Segmentanzeige am SFR_LED ausgeben.
 */
unsigned charwert1  = 0x7f;
unsigned charwert2  = 0;
unsigned charwert3  = 0xd;

led7_set(wert1);    // Alle 7 LEDs einschalten

wert2  = led7_get();// Wert2 ist jetzt auch 0x7f

led7_hex(wert3);    // „d“ wird auf 7 Segmentanzeige angezeigt
led7_hex(16);       // Bei Werten größer 15 ist die Anzeige aus-
                    // geschaltet
```

;