MICROCONTROLLER

Einführung

Eigenschaften

- Steuer- und Regelungsaufgaben
- Vielfältige Peripheriemodule
- „kleine“ Rechenleistung
- Kein Benutzerinterface

Simulation	Emulation
Ausführung auf dem Host	Ausführung auf dem Zielsystem

Zahlensysteme

Dezimalsystem	Binärsystem	Hexadezimalsystem
63	0b0011‘1111	0x3F

Negative Zahlen -> Zweierkomplement
Vorzeichenwechsel = Alle Bits invertieren und 1 dazuzählen

Speicherorganisation

Standard = 8bit

ROM	EEPROM	Flash	RAM
nur lesen	electric erasable		flüchtig
R	R/((W))	R/(W)	R/W

16-Bit Speicherung (z.B. von 2233h)

Big Endian		Little Endian		
3F1		3F1		msb = most significant bit
3F2	22h (msb)	3F2	33h (lsb)	lsb = least significant bit
3F3	33h (lsb)	3F3	22h (msb)	

Addressierungsarten, Delay

Data Direct Addressing (direkt von Register)

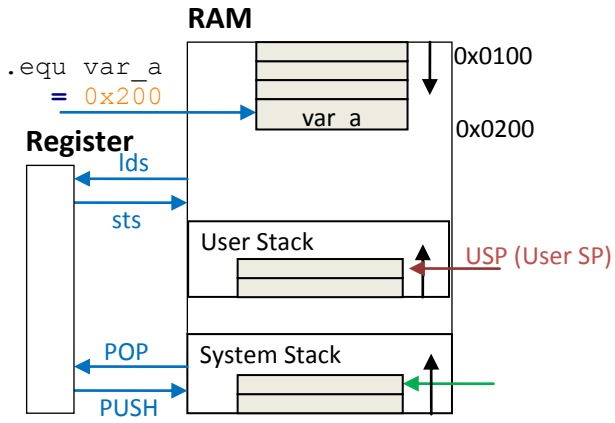
```
ldi r16, 0xA6
STS var_a, r16
```

Data Indirect Addressing (über X-Register)

```
LDI XL, LOW (0x0500)
LDI XH, HIGH (0x0500)
ST X, r16
```

Stack, Subroutinen, Clock

RAM + System Stack (Stapel)



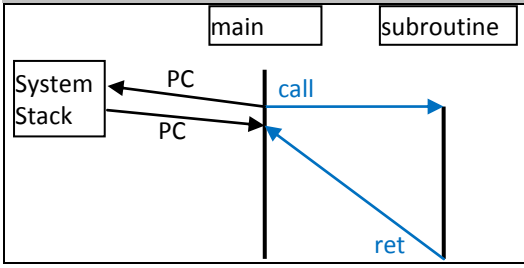
lds	
sts	
pop r16	1. StackPointer +1 2. Wert an SP -> r16
push r17	1. Wort von r17 an 0xFF 3. StackPointer -1

Initialiserung des System Stacks

```
ldi r16, HIGH(RAMEND)
out sph, r16
ldi r16, LOW(RAMEND)
out spl, r16
```

hat 16 Bit, für lokale Variablen und Parameter
RAMENDE = Konstante die vordefiniert ist

Subroutine



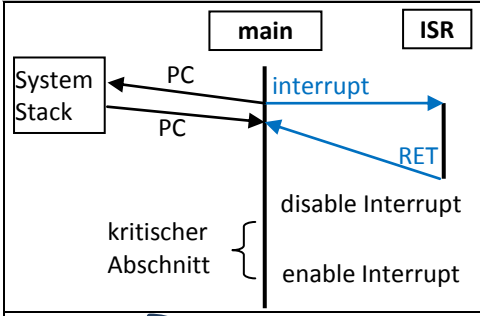

```
  call routine

routine:
  push r14
  ...
  pop r14
  ret
```

- gleich viele push wie pop

Interrupts, Timer0

Interrupts (Unterbrechungen)

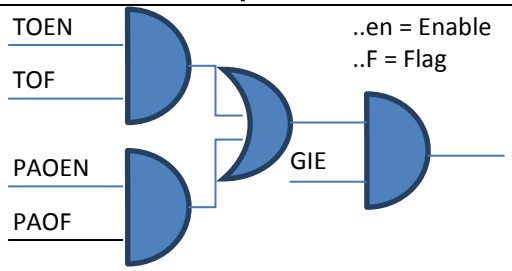


Allgemein

- ist erreignissgesteuert, entspricht event handling
- können jederzeit unterbrechen
- Statusregister mit IN und OUT sichern und wiederherstellen

ISR (Interrupt Service Routine)

- definierte Stelle anspringen
- kein returnwert



GIE = Global interrupt enable

Ablauf

alte Instruktion wird fertig ausgeführt
GIE Bit wird gelöscht -> keine weiteren Interrupts
PC (Programmcounter) wird in Stack geschrieben (initialisiert)
Interrupt Vektor wird geladen (Programm Address)
Hardware löscht interrupt-flag
jrmp zu ISR (Verzweigung, muss nicht geschrieben werden)
SREG, in allen Register auf den Stack retten (Statusregister)
(opt.) GIE setzen für Nesting (Interrupt unterbricht Interrupt), falls dies möglich sein soll
ISR ausführen (Code)
Stack -> Statusregister (gerettete Register aus Stack holen)
RETI (Code um zurückzuspringen)
PC <- Stack (Programmcounter wird vom Stack gehohlt
GIE wird wieder gesetzt (um wieder interrupts zu erlauben

Timer 0 (S.102)

8-bit Timer (0-255)

TCCRnA und RCCRnB, um Timer zu initialisieren
Prescalor,
Overflow-Interrupt, wenn timer von 255 auf 0.

mit COM: ob OC0A Bit gesetz werden soll (0 beim initialisieren)
CS00, CS01, CS02 (Clock source),
0 = timer läuft nicht, 1 = timer läuft (für Prescalor)
Zählvariable (TCNT0)

berechnen von Startwert

Timer 0 OVF (OverFlow) Frequenz

1 Tick berechnen: 1/f_Timer = prescalor/f_sys
f_Timer = f_sys/Prescalor

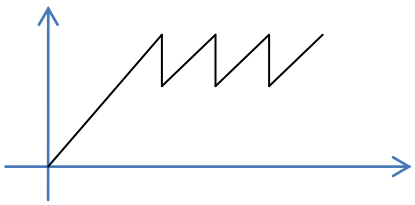
wieviele Ticks

Bsp prescailor = 1:
1 Tick = 1/ 8 000 000Hz
80 000 Ticks für 10ms

Problem: grösser als 8 bit
Lösung: prescailor auf 1024

1 Tick = 1024/8 000 000 = 128us
10/0.128=78.125 Tick => 78Ticks

bei 0xFF kommt der Overflow -> den Start ändern 256-78 = **178**



PORT E = 4-7 Pin 4-7
PORT D = 0-3 Pin 0-3 (Externer Interrupt)

Low-Power Design (Sleep Modus, Airthmetik)

Benützung des Timer 3 für den Messtakt (S130, S181)

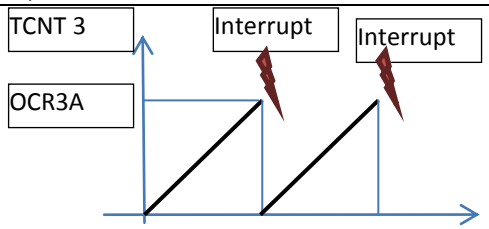
$f_{Interrupt} = \frac{f_{sys}}{prescaler * (OCRnA + 1)}$	$OCRnA = \frac{f_{sys}}{prescaler * f_{interrupt}} - 1 = \frac{8MHz}{1024 * 1} \approx 7812$

$\rightarrow 16Bit\ Zähler$

CTC Mode (Clear Timer On Compare Match)

einstellen über TCCR1A und TTCR1B – WGM11, WGM1, WGM13 und WGM12

```
TCCR3A = 0;
TCCR3B = 0;
TCCR3B |= (1<<WGM32); //CTC Mode
TCCR3B |=(1<<CS32); //prescaler = 256
TIMSK3 |= (1<<OCIE3A); //OC Interrupt enable
OCR3A=31249; //Interrupt Takt = 1sec
```



UART

RS232

- bit-serielle asynchrone Schnittstelle
- an Protokoll halten
- zeichenorientiert (ein Zeichen nach dem Anderen)

Übertragungsmodi

Synchron	Asynchron
Sender und Empfänger haben den gleichen Takt	Daten werden unregelmässig übertragen

Nullterminiert: Jeder String hat hinten eine Null.

Treiber

= Zugriff auf Hardware

Idee: sollte unabhängig von Hardware sein

Pointer

- enthält als Wert eine Speicheradresse

JAVA	C
`Complex c1 = new Complex();`	`struct Complex c;` `stuct Complex *c1 = &c;`

Initialisieren

`int *j;  //Pointer auf Integer`	`int* j;`
`int *j = &i; // inkl. Zuweisung  "&" vor dem i liefert die Adresse`	

Beispiel

```
int i;
int *j = &i; //deklaration und Adressen-Zuweisung
*j=100;   //i hat jetzt den Wert 100 (Dereferenzieren)
i = 100;     //bewirkt dasselbe
int k = *j;  //k hat jetzt auch den Wert 100
```

Array

C hat keine Array-Bereichsüberschreitungs-Prüfung

überschreiten eines Index ausserhalb überschreibt fremenden Speicher

Strings

konstanten Strings

```
char * str = "hello";
char str [] = "hello";  //identisch, str enthält Adresse des konst. Strings
```

String kopieren

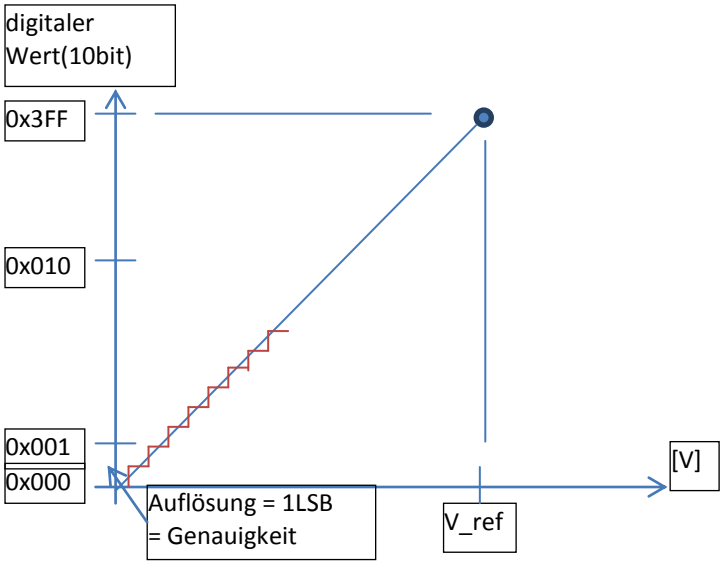
Bibliothek: `<string.h>`

Code: `char * strcpy(char *s1, const char *s2);`

Erklärung: Kopiert String s2 in die Adresse von s1, return ist ebenfalls s1

ADC – Analog digital Wandler

10 bit, 0-V_CC;
Running or Single Conversion
Noice Canceler = Sleep Mode (S. 320)
funktionert wenn single Conversion eingestellt ist.
Sobald der Noice Canceler gestartet wird, startet die Konvertierung. (S.51)
Autotrigger
ADMUX = welcher Kanal gemessen wird, welche Versorgung



$$Z = \frac{V_{in}}{V_{ref}} * (2^{10} - 1)$$

$$1LSB = \frac{V_{Ref}}{1023} = 0.0048$$

CRC (zyklische Redudanzprüfung)

Methode, um durch eine Prüfsumme Fehler bei einer Datenübertragung zu finden.
Angefügter Wert, der keinen Informationsgehalt besitz.
redundant
Verfahren mittels Polynomdivision, CRC = Modulo

Struktur im receive

struct {
char count;
unsigned temp;
int crc;
} rxFrame
	Achtung falsche Reihenfolge von CRC
	
	deshalb:
	
	char data[3];
	data[0] = rxFrame.temp;
	data[1] = rxFrame .crc>>8;
	data[2] = rxFrame.crc;
	
	calcCRC16(data,3); // muss null sein fals Temp richtig angekommen

count temp LSB crc MSB crc	

Vorsicht!!!
Achtung, beim Prüfen der Temperatur den Interrupt ausschalten, das sonst der Interrupt das Data auffüllt.

deshalb

Beim Berechnen der CRC den Interrupt ausschalten.

Übersicht

Register File	I/O Memory	DataSpace	Program Memory
00 5 16 31	0 63	Data 0x0000 var_a 0x0100 RAMEND	0x0000 FLASHEND
`ldi r16,5`	`in r17,PINB` `out PORTA,r22`	`.equ var_a = 0x200`	

Instruktionen

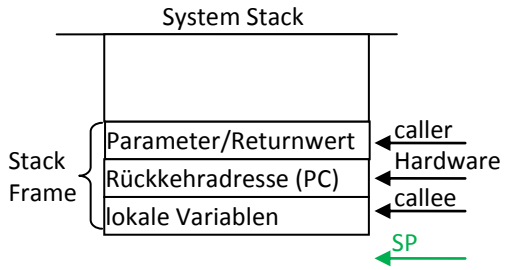
Fast alle	16Bit	passen in Programm Memory
Wenige	32Bit	brauchen zwei Programmwörter

Verweise

`jmp`	`call`	jump/call irgendwo	direkt, ohne Bedingung
`rjmp`	`rcall`	jump/call in Nähe (schneller)	relativ, ohne Bedingung
BRNE		Branch if not equal	mit Bedingung

bei Sprung, muss ein NOP (No Operation) ausgeführt werden

Lesen und schreiben der lokalen Variablen



Variante System Stack
(Y-Register als Frame Pointer nutzen)
schreiben: SD Y+q, Rr
lesen LD Rd, Y+q

SP+3 -> geht nicht
Y+q -> geht

Variante User Stack (wir benützten diese Variante)
lokale Variablen und Parameter in User Stack schreiben
benötigt einen User Stack Pointer (USP) -> Y-Register
achten, dass der User Stack nicht in besetzte Bereiche wächst

X,Y,Z - Register

```
LDI XL, LOW (0x0500)
LDI XH, HIGH (0x0500)
LD r17,X
```

Y=R28,R29

(1<<5) = 0010 0000