

Übungen zu Grundlagen der systemnahen Programmierung in C (GSPIC) im Sommersemester 2018

2018-05-29

Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme und Betriebssysteme

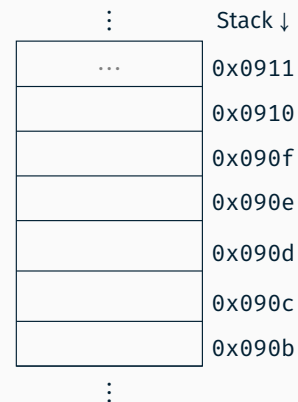


Zeiger & Felder

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

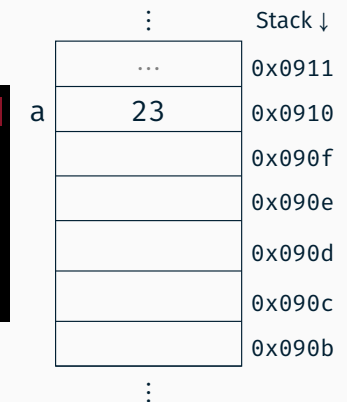
```
01 uint8_t a = 23;  
02 uint8_t b = 42;  
03 uint8_t * p = &a;  
04 *p = 66;  
05 p = &b;  
06 *p -= 40;  
07 uint8_t c = *p;
```



Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;  
02 uint8_t b = 42;  
03 uint8_t * p = &a;  
04 *p = 66;  
05 p = &b;  
06 *p -= 40;  
07 uint8_t c = *p;
```



Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;  
02 uint8_t b = 42;  
03 uint8_t * p = &a;  
04 *p = 66;  
05 p = &b;  
06 *p -= 40;  
07 uint8_t c = *p;
```

	:	Stack ↓
	...	0x0911
a	23	0x0910
b	42	0x090f
		0x090e
		0x090d
		0x090c
		0x090b
	:	

Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

1

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;  
02 uint8_t b = 42;  
03 uint8_t * p = &a;  
04 *p = 66;  
05 p = &b;  
06 *p -= 40;  
07 uint8_t c = *p;
```

	:	Stack ↓
	...	0x0911
a	23	0x0910
b	42	0x090f
p	0x0910	0x090e
p	0x0910	0x090d
		0x090c
		0x090b
	:	

Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;  
02 uint8_t b = 42;  
03 uint8_t * p = &a;  
04 *p = 66;  
05 p = &b;  
06 *p -= 40;  
07 uint8_t c = *p;
```

	:	Stack ↓
	...	0x0911
a	66	0x0910
b	42	0x090f
p	0x0910	0x090e
p	0x0910	0x090d
		0x090c
		0x090b
	:	

Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;  
02 uint8_t b = 42;  
03 uint8_t * p = &a;  
04 *p = 66;  
05 p = &b;  
06 *p -= 40;  
07 uint8_t c = *p;
```

	:	Stack ↓
	...	0x0911
a	66	0x0910
b	42	0x090f
p	0x090f	0x090e
p	0x090f	0x090d
		0x090c
		0x090b
	:	

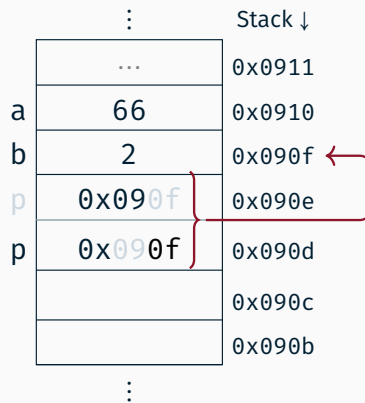
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```



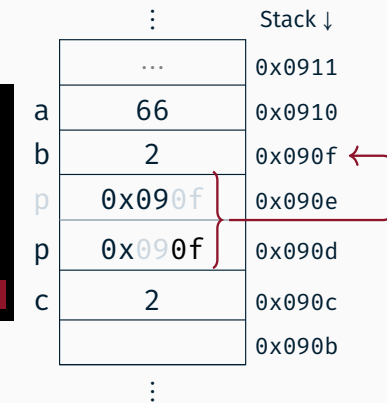
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```



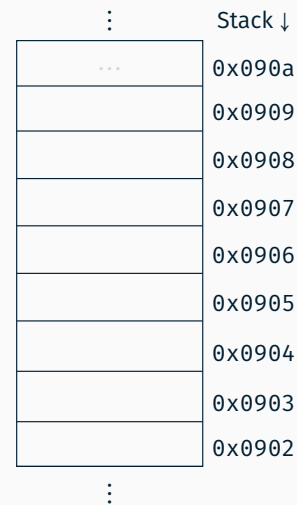
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```

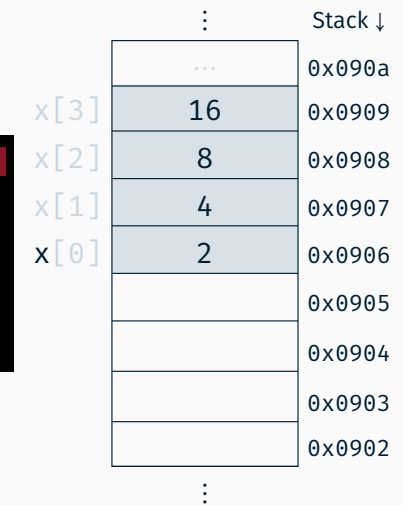


2

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```

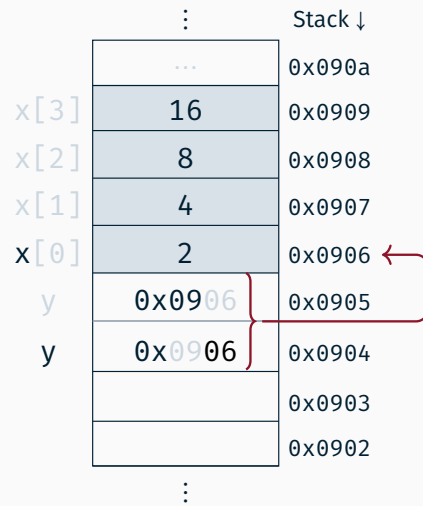


2

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```

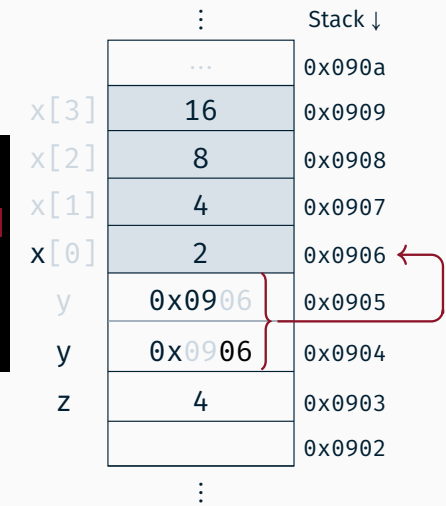


2

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```

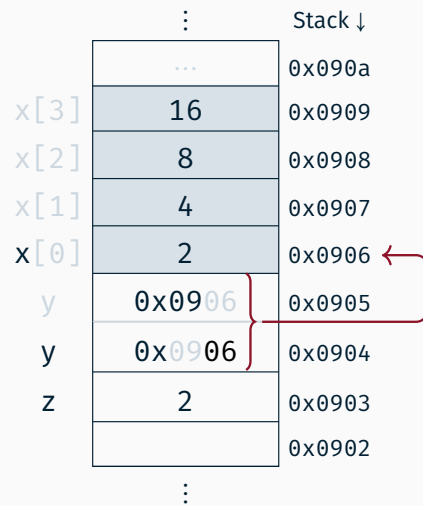


2

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```

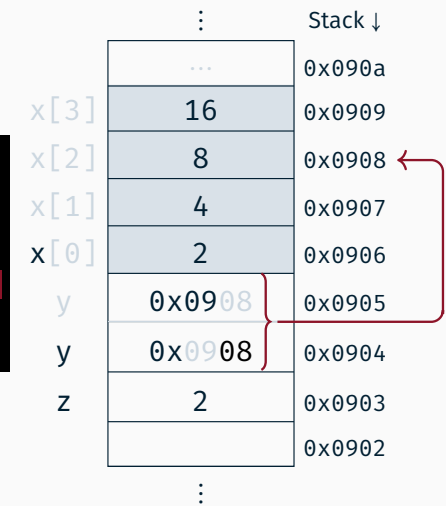


2

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```

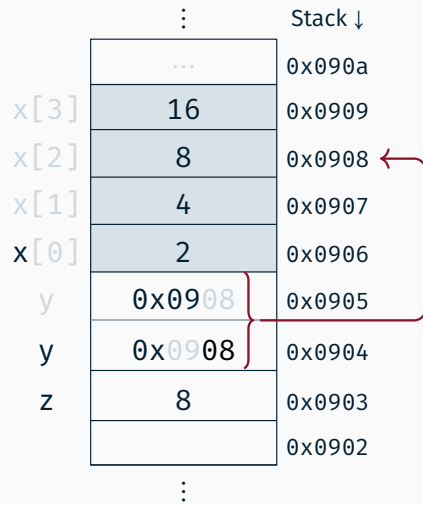


2

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```

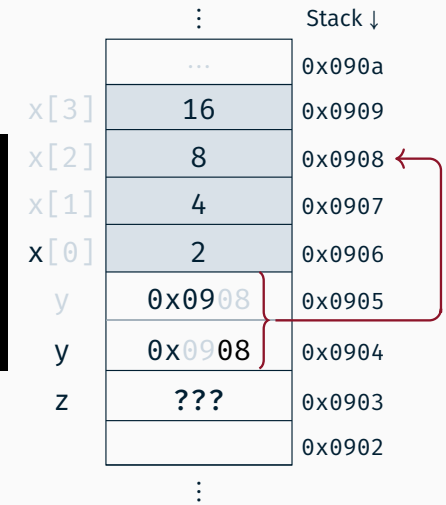
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
    
```



- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

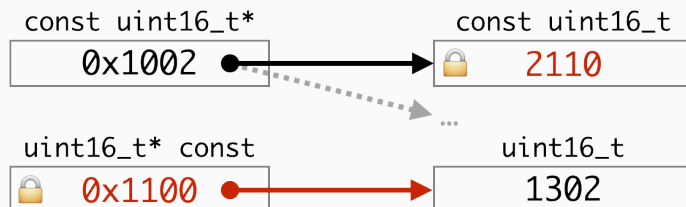
```

08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7]; // !!!
    
```



const uint8_t* vs. uint8_t* const

- `const uint8_t*`
 - ein Pointer auf einen `uint8_t`-Wert, der konstant ist
 - Wert nicht über den Pointer veränderbar
- `uint8_t* const`
 - ein **konstanter Pointer** auf einen (beliebigen) `uint8_t`-Wert
 - Pointer darf nicht mehr auf eine andere Speicheradresse zeigen



Hands-on

- struct für GPS-Koordinaten
- Feld von GPS-Koordinaten
- Call-by-Value vs. Call-by-Reference
- Zeigerarithmetik
- Funktionszeiger

Ausgabe über die serielle Konsole (nach Tastendruck auf BUTTON0)

4

Die serielle Konsole

- erlaubt dem Programm auf dem SPiCboard einfache Kommunikation (Aus- und Eingaben) mit dem PC
- nutzt die serielle Schnittstelle (UART)
- benötigt `#include <console.h>`
- wird durch die Initialisierungsfunktion `sb_console_connect_default();` auf 38400 bit/s (ohne Paritätsbit und mit einem Stoppbit) eingestellt
- ermöglicht Ausgabe über die libspicboard-Funktion `sb_console_putString("Hallo Welt!\n");`
- oder über mächtige Standardbibliotheksfunktionen wie `printf` und `scanf` (aber nicht ganz trivial).

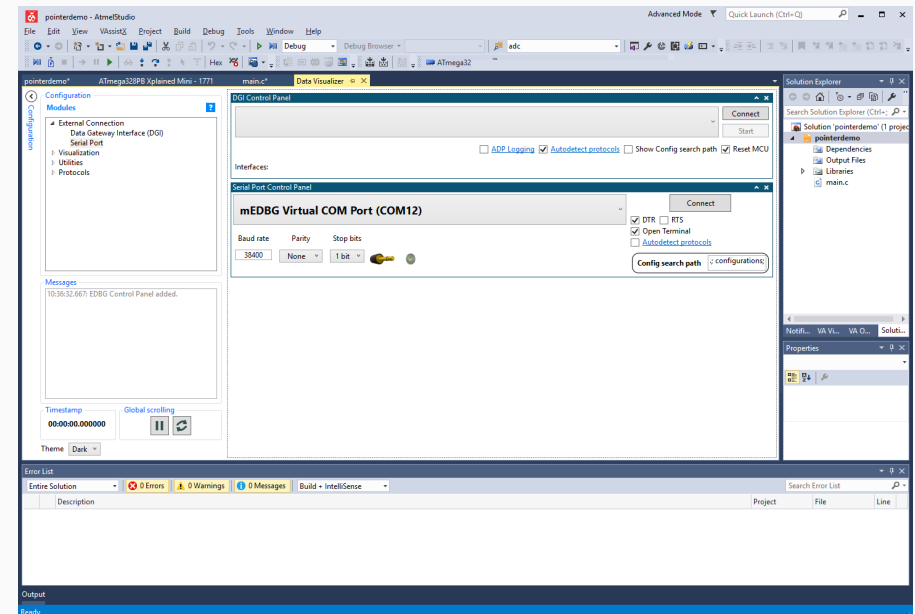
5

Exkurs: Serielle Konsole

Verwendung in ATMEL STUDIO 7

- im Menü *Tools* den *Data Visualizer* starten
- auf der Seite *Configuration* auswählen
- in der Gruppe *Modules* den Punkt *External Connection* erweitern und *Serial Port* auswählen.
- das neu angezeigte *Serial Port Control Panel* anpassen:
 - Verbindung wählen, etwa *mEDBG Virtual COM Port (COM4)*
 - die Übertragungsrate (*Baud rate*) auf *38400* setzen
 - Parität (*Parity*) mit *None* deaktivieren
 - einfaches Stoppbit (*Stop bits* auf *1*)
 - Haken bei *DTR* (und *RTS* auf langsamen Systemen)
 - *Connect* zum verbinden

Exkurs: Serielle Konsole



5

5

Exkurs: Serielle Konsole

The screenshot displays the Atmel Studio IDE with the following components:

- Debugger:** Shows the program execution state. The CPU register window indicates the program counter is at line 07.
- Code Editor:** Contains the following C code:

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#define LED_ON 1
#define LED_OFF 0

static uint8_t pointerDemoVoid()
{
    uint8_t led_on(LED_ON);
    uint8_t led_off(LED_OFF);
    pointerDemo();
}

// Bei Linuxprogrammen haben wir Parameter (argc beinhaltet die Anzahl,
// argv die Werte der Parameter) und einen Rückgabewert
int main(int argc, const char * argv[]) {
    // (Bitte Demo einmalig aus und loslos)
    return pointerDemo();
}

static uint8_t pointerDemo(void) {
    printf("PointerDemo\n");

    // Pointer allzuerst
    printf("%p\n", (int general));

    uint16_t t = 1302;
    printf("t   => %u\n", t);
    printf("t*2 => %u\n", t*2);

    uint16_t *t_ptr = &t;
    printf("t_ptr => %p\n", t_ptr);

    uint16_t t2 = *t_ptr;
    printf("t2   => %u\n", t2);

    t = 3;
    printf("t   => %u\n", t);
    printf("t_ptr => %p\n", t_ptr);
    printf("t2   => %u\n", t2);
    printf("t_ptr => %p\n", t_ptr);

    // const Pointer vs. Pointer const
    printf("const pointer\n");
}
```
- Terminal:** Shows the output of the program:

```
PointerDemo
pointer (in general):
t   => 1302
t*2 => 0x0508
```
- Autos Window:** Shows the current state of variables:

Name	Value	Type
t	1302	uint16_t
t_ptr	0x0516	uint16_t*
- Memory Window:** Shows the memory dump for prog FLASH, starting at address 0x0000.