

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



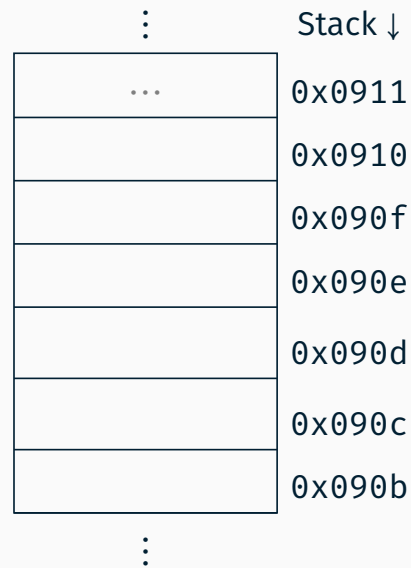
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

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;
```

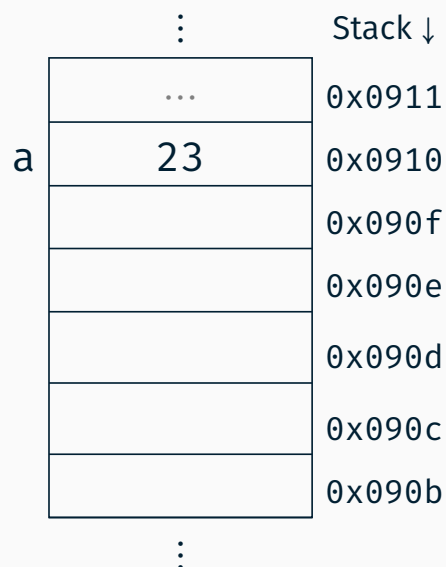


1

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;
```



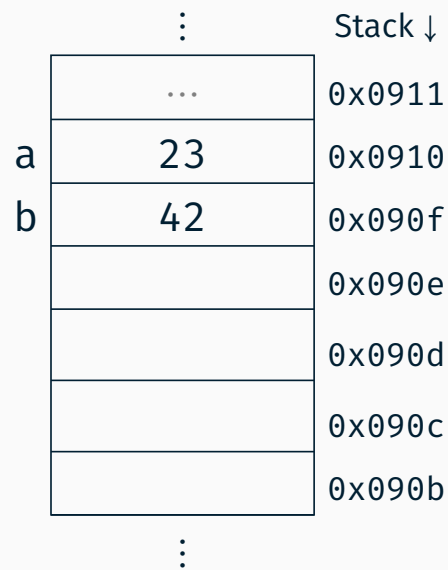
Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

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;
```



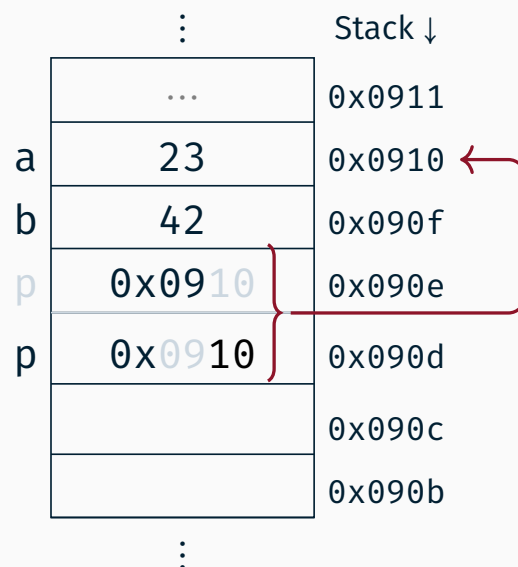
Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

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;
```



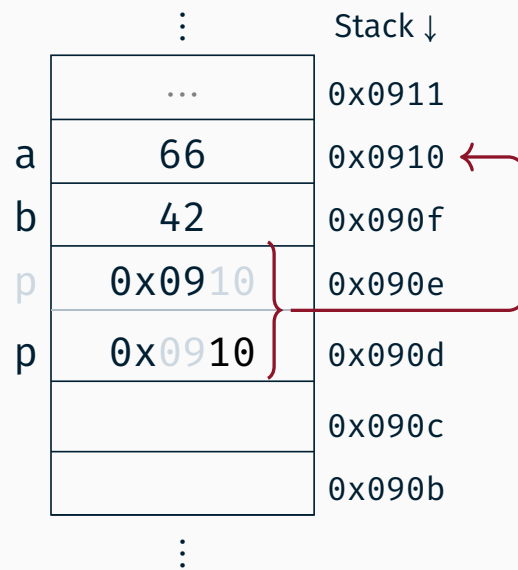
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

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;
```



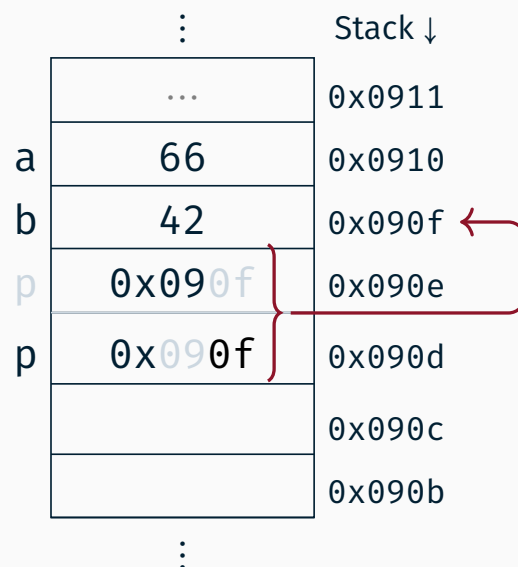
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

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;
```



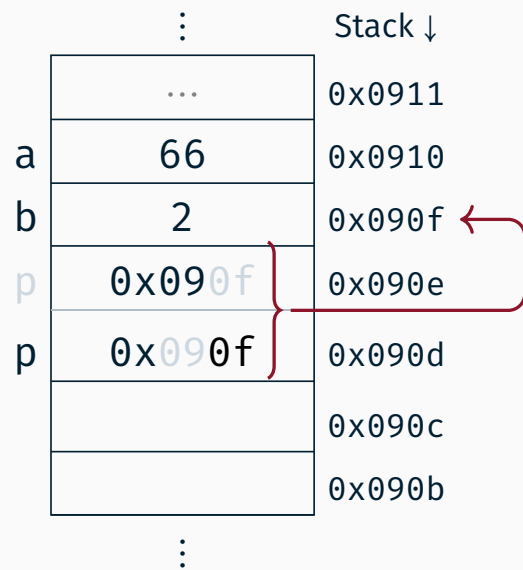
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

1

Wiederholung: 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;
```



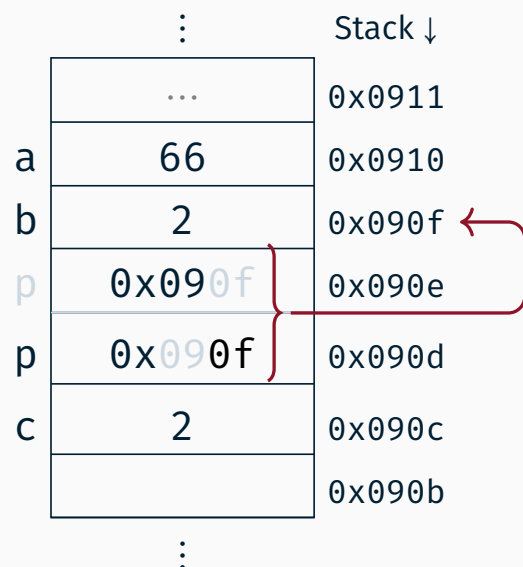
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

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;
```



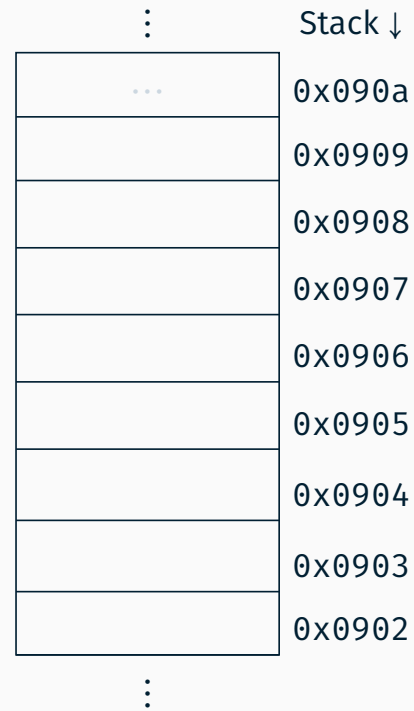
Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

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)`

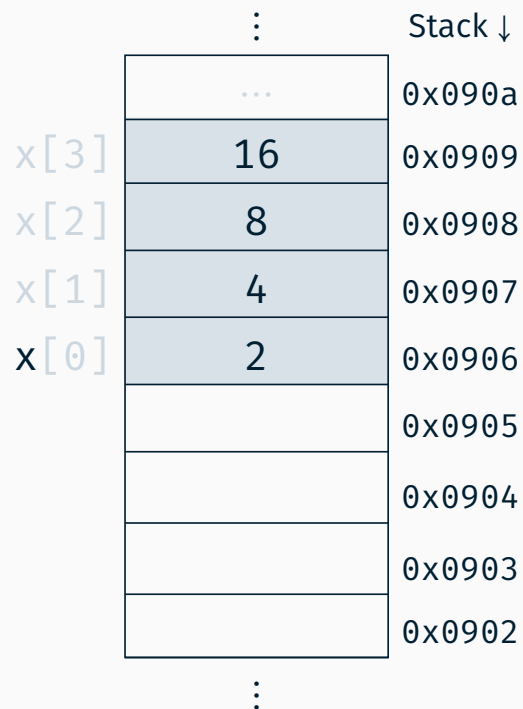
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



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)`

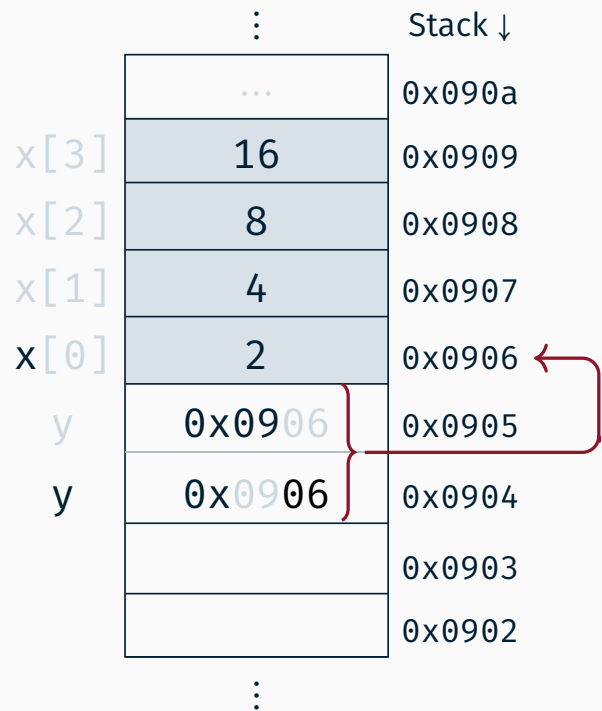
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



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)`

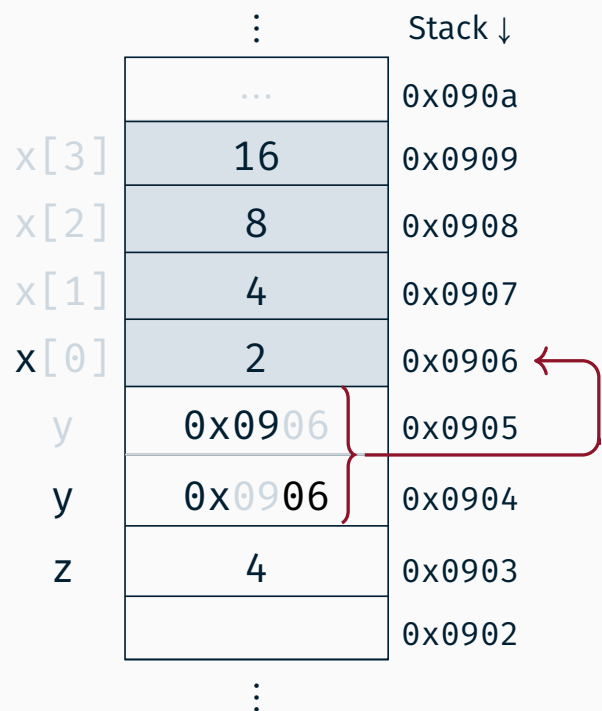
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



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)`

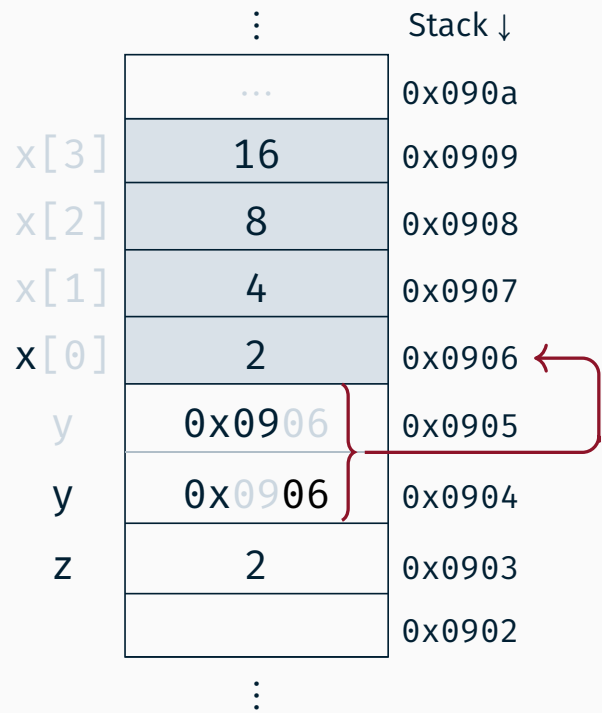
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



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];
```

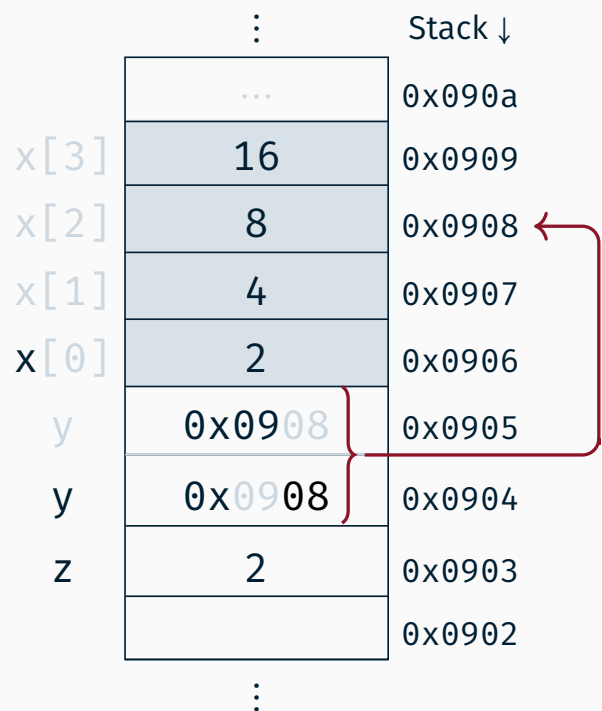


2

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];
```

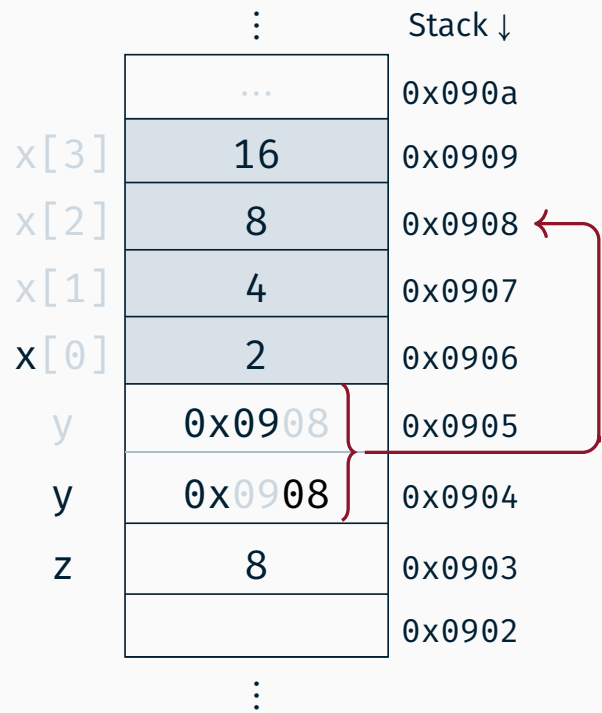


2

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];
```

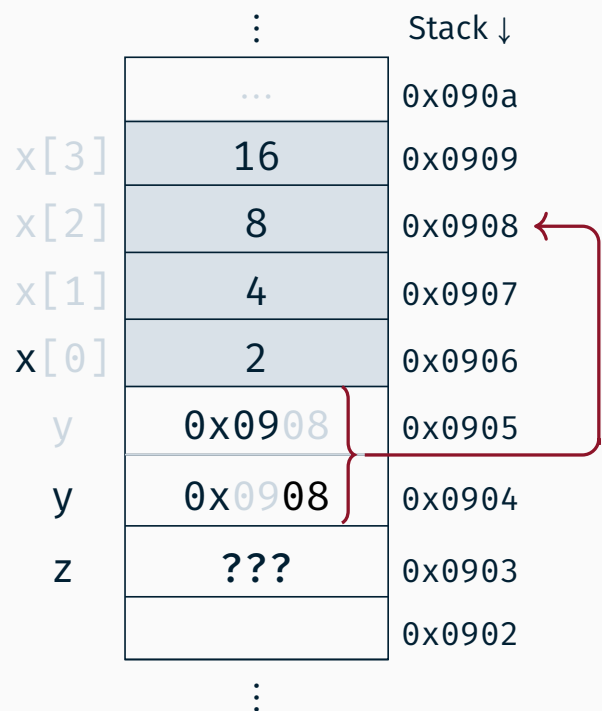


2

Wiederholung: Felder

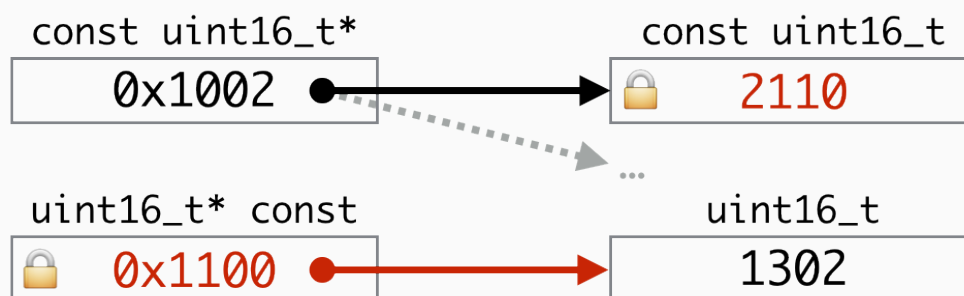
- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
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

- `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

Exkurs: Serielle Konsole

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).

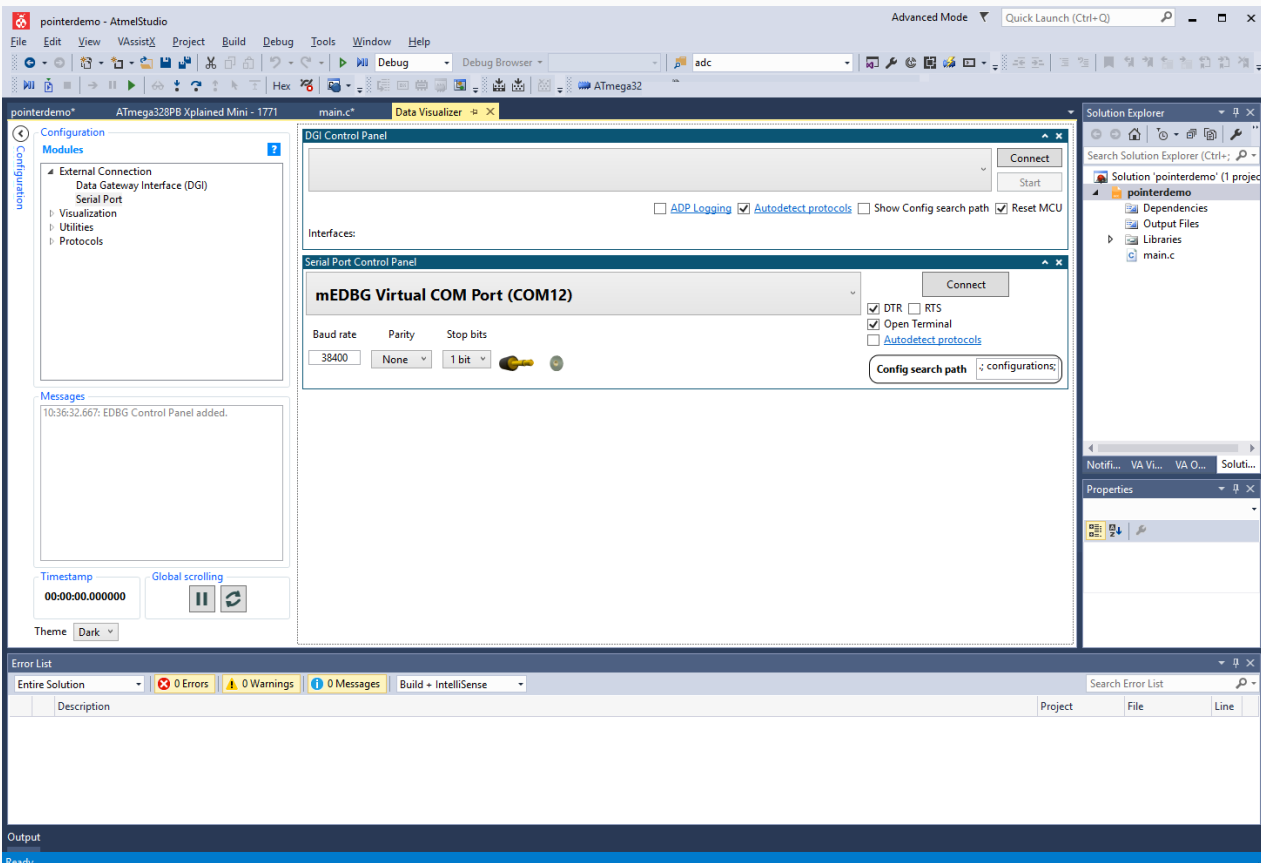
5

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 Stoppsbit (*Stop bits* auf *1*)
 - Haken bei *DTR* (und *RTS* auf langsamen Systemen)
 - *Connect* zum verbinden

5

Exkurs: Serielle Konsole



5

Exkurs: Serielle Konsole

The screenshot displays the Atmel Studio IDE in Advanced Mode, debugging a program named 'pointerdemo'. The main window shows the C source code for 'main.c'. The code defines a function 'pointerDemo' that prints a pointer value and then demonstrates pointer arithmetic. The 'main' function calls 'pointerDemo' and prints the values of variables 't' and 't_ptr'.

```
pointerdemo (Debugging) - AtmelStudio
File Edit View Assistant Project Build Debug Tools Window Help
Debug Browser - adc
ATmega328PB debugWIRE on mEDBG (ATML2523041800001771)
Data Visualizer - X
pointerdemo main.c Disassembly
pointerDemo
static uint8_t pointerDemo(void)
sb_led_on(GREEN);
sb_led_off(YELLOW);
pointerDemo();
}
#else
// Bei Linuxprogrammen haben wir Parameter (argc beinhaltet die Anzahl,
// argv die Werte der Parameter) und einen RÜCKGABEWERT
int main(int argc, const char * argv[]) {
// Führe Demo einmalig aus und beende
return pointerDemo();
}
#endif
static uint8_t pointerDemo(void){
printf("[PointerDemo]\n");
// Pointer allgemein
printf("\npointer (in general):\n");
uint16_t t = 1302;
printf("t => %u\n", t);
printf("&t => %p\n", &t);
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("\nconst pointer:\n");
}
Autos
Name Value Type
t 1302 uint16_t
t_ptr 0x0516 uint16_t*
Memory 4
Memory: prog FLASH Address: 0x0000,prog
prog 0x0000 0c 94 5b 00 0c 94 67 04 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x0015 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x002a 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x003f 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x0054 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x0069 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x007e 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x0093 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c 94 83 00 0c
prog 0x00a8 0c 94 83 00 0c 94 83 00 0c 94 83 00 d6 04 11 24 1f be cf ef d8
```

The Terminal window shows the output of the program:

```
[PointerDemo]
pointer (in general):
t => 1302
&t => 0x0508
```

The Autos window shows the values of variables 't' (1302) and 't_ptr' (0x0516). The Memory window shows the program's memory layout, including the 'prog FLASH' and 'Memory 4' windows.