Programming Embedded Devices with C

Sommerakademie in Leysin
AG 2 – Effizientes Rechnen

Johannes Zeitler

Friedrich-Alexander-Universität Erlangen/Nürnberg

August 2016

Inhalt

- Einleitung
- Eigenschaften der Sprache C
- Grundlagen der Sprache C – Codebeispiele
- Zeiger
- Hardwarenahe Programmierung eines ATmega32
- Programmierbeispiel

Einleitung

Anforderungen an eingebettete Systeme

- Energieverbrauch
- Echtzeitverarbeitung
- Abmessungen
- Kosten

Inhalt

- Einleitung
- **Eigenschaften der Sprache C**
- Grundlagen der Sprache C – Codebeispiele
- Zeiger
- Hardwarenahe Programmierung eines ATmega32
- Programmierbeispiel

Eigenschaften der Sprache C

Eigenschaften

- imperative Sprache → Gliederung in Funktionen und Variablen
- kleine Menge an Schlüsselwörtern → einfache Compiler
- direkte Speicherzugriffe
- fast keine Fehlerprüfung zur Laufzeit
- nicht typsicher
- Modularisierung auf Dateiebene

Eignung von C für Eingebetette Systeme

- Laufzeiteffizienz
  - Code läuft direkt auf dem Prozessor
  - Keine Prüfungen auf Programmierfehler zur Laufzeit
  - Keine Prüfung der Datenzugriffe zur Laufzeit
- Platzeffizienz
  - Code und Daten lassen sich kompakt speichern
- Direktheit
  - direkter Zugriff auf Speicher und Register
- Portabilität
  - Compiler für JEDE Plattform vorhanden

Inhalt

- Einleitung
- Eigenschaften der Sprache C
- **Grundlagen der Sprache C – Codebeispiele**
- Zeiger
- Hardwarenahe Programmierung eines ATmega32
- Programmierbeispiel

Grundlagen der Sprache C - Codebeispiele

Struktur eines C-Programms

```
#include <display.h>                    // Bibliotheken einbinden

int zahl = 42;                          // Globale Variablen

void addAndShow(int a, int b) {         // Sub-Funktionen
       int sum = a + b;
       displayShow(sum);
}

int main(void) {                        // Main-Funktion
       addAndShow(zahl, 17);

       while(1) {}
}
```

Datentypen

C-Standard				
char	short	int	long	long long
>= 8 bit	>= 16 bit	>= 16 bit	>= 32 bit	>= 64 bit

stdint.h			
uint8_t	0 … 255	int8_t	-128 … +127
uint16_t	0 … 65.535	int16_t	-32.768 … + 32.767
uint32_t	0 … 4.294.967.295	int32_t	-2.147.483.648 … +2.147.483.647
uint64_t	0 … 1,8*10^19	int64_t	-9.2*10^18 … +9.2*10^18

Leerer Datentyp: void

außerdem: float, double, long double, _Bool, _Complex, _Imaginary

Literalformen

```
int a = 42;          // dezimal
int b = 0x2A;        // 0x…: hexadezimal
int c = 052;         // 0…: oktal
```

Zeichenfolgen in C

```
#include <stdio.h>

char[] string = "Hello World!\n";

int main(void) {
	printf(string);
	return 0;
}
```

Bitweise Operatoren

&	bitweises "und"
|	bitweises "oder"
^	bitweises "Exklusiv-Oder"
~	bitweise Inversion
<<	bitweises Linksschieben
>>	bitweises Rechtsschieben

Bit	7	6	5	4	3	2	1	0
x = 115	***0***	***1***	***1***	***1***	***0***	***0***	***1***	***1***
~x	1	0	0	0	1	1	0	0
7	***0***	***0***	***0***	***0***	***0***	***1***	***1***	***1***
x & 7	0	0	0	0	0	0	1	1
x | 7	0	1	1	1	0	1	1	1
x ^ 7	0	1	1	1	0	1	0	0
x << 2	1	1	0	0	1	1	0	0

Inhalt

- Einleitung
- Eigenschaften der Sprache C
- Grundlagen der Sprache C – Codebeispiele
- **Zeiger**
- Hardwarenahe Programmierung eines ATmega32
- Programmierbeispiel

Zeiger

Zeiger - Syntax

- Zeigervariable enthält als Wert die Adresse einer anderen Variable
- Adressoperator &x
- Verweisoperator *y
- (*(&x)) ≡ x

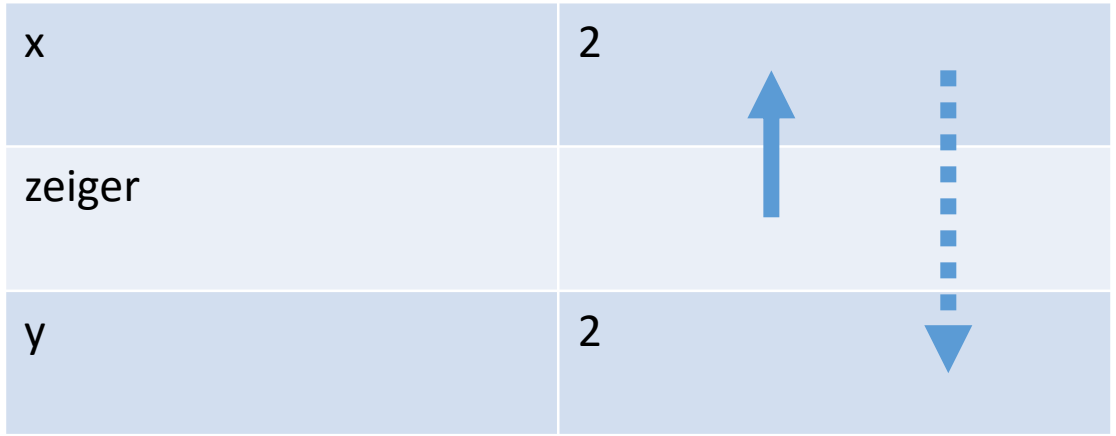
```
int x = 2;

int *zeiger;

zeiger = &x;

int y;

y = *zeiger;
```



Zeiger - call by reference

```
void tausche (int *a, int *b) {   // tausche erhält als Parameter zwei Zeiger
       int tmp;                   // Puffer
       tmp = *a;                  // Wert von Speicher a in den Puffer schreiben
       *a = *b;                   // Wert von Speicher b in Speicher a schreiben
       *b = tmp;                  // Puffer in Speicher b schreiben
}

int main() {
       int z1 = 1;
       int z2 = 2;
       swap(&z1, &z2);            // Adressen von z1 und z2 übergeben
}
```

Zeiger auf Funktionen

```
void mehrfachAusfuehren( void (*aufgabe)(void) , uint16_t anzahl) {
	for(uint16_t i = 0; i < anzahl; i++) {
		aufgabe();
	}
}

void blinken(void) {
	ledAn();
	warten(500);
	ledAus();
	warten(500);
}

int main(void)
{
	mehrfachAusfuehren(blinken, 20);
	while(1) {}
}
```

Malloc

- Reservierung eines zusammenhängenden Speicherblocks

```
int main(void) {
	char* m = malloc(50);		// Speicher für 50 chars bereitstellen
	m[10] = 42;
	// …
	free(m);			// reservierten Speicher wieder freigeben
}
```

Inhalt

- Einleitung
- Eigenschaften der Sprache C
- Grundlagen der Sprache C – Codebeispiele
- Zeiger
- **Hardwarenahe Programmierung eines ATmega32**
- Programmierbeispiel

Hardwarenahe Programmierung eines ATmega32

Zugriff auf Hardware-Register

```
#include <avr/io.h>          // Bibliothek einbinden

int main(void)
{       // Hardware initialisieren (LED an Port D Pin 7, active low)
        DDRD |= (1<<7);      // Port D Pin 7 ist Output
        // (* (volatile uint8_t*) ( 0x31 ) ) |= (1<<7);

        PORTD |= (1<<7);     // Pin 7 high -> LED ist aus
        // (* (volatile uint8_t*) ( 0x32 ) ) |= (1<<7);

        // „Hallo Welt“ – LED anschalten
        PORTD &= ~(1<<7);    // Port D Pin7 low -> LED an
        // (* (volatile uint8_t*) ( 0x32 ) ) &= ~(1<<7);

        while(1) {}   }      // Endlosschleife
```

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$31	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$32	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0

Externe Interrupts

```
#include <avr/interrupt.h>
#include <avr/io.h>
ISR( INT0_vect) { … }              // Installation der Interrupt-Service-Routine

int main(void) {
      DDRD &= ~(1<<PD2);                  // Taster an PD2 ist Input
      PORTD |= (1<<PD2);                  // PD2 hat Pull-Up-Widerstand

      MCUCR |= (1<<ISC01) | (1<<ISC00); // Interrupt wird bei steigender Flanke
                                          // ausgelöst
      // ( *(volatile uint8_t*) (0x55) ) |= (1 | 2);

      GICR |= (1<<INT0);                  // Aktivierung der Interrupt-Quelle INT0
      // ( *(volatile uint8_t*) (0x5B) ) |= (1<<6);

      sei(); …}                           // Interrupts global zulassen
```

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$55	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
$5B	GICR	INT1	INT0	INT2	-	-	-	IVSEL	IVCE

Programmierbeispiel

Vielen Dank für Eure Aufmerksamkeit!