Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Hardwarenahe Softwareentwicklung

Startup, Schnittstelle C/Assembler

V5.1, ©2023 roger.weber@bfh.ch

Lernziele

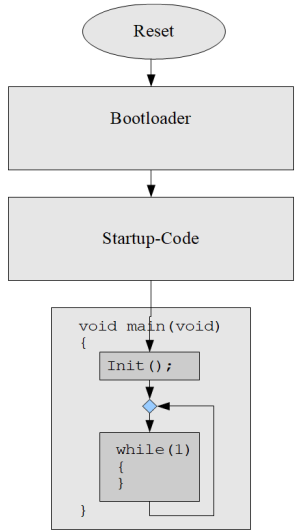
Sie sind in der Lage:

- Den Aufstartvorgang eines Embedded Systems zu erklären.
- Den Startup-Code zu modifizieren.
- Programme in C und Assembler gemischt zu schreiben.

Inhaltsverzeichnis

Der Aufstartvorgang

Ablauf des Aufstartvorgangs



Reset

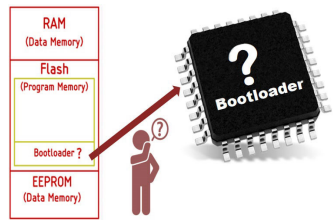
- Der Reset bringt den Microcontroller in einen definierten Anfangszustand.
- Der Reset ist als Vektor in der Vektor-Tabelle eingetragen.

- Welche Ereignisse können einen Reset auslösen?

Bootloader

Die Aufgaben des Bootloaders sind:

- Initialisieren der wichtigsten Register (Chip-Select, Memory-Controller).
- Download von neuen Firmware-Versionen.
- Laden des Betriebssystem (falls vorhanden).
- Laden der Applikation vom Flash (oder anderen nichtflüchtigen Speichermedien) in den Arbeitsspeicher (RAM / TCM) falls erforderlich.
- Start der Applikation.
- Der Bootloader ist optional und nicht bei allen Embedded Systems vorhanden.



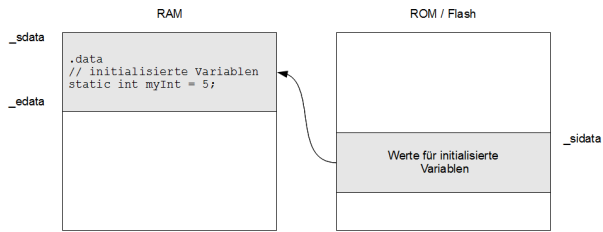
Startup-Code

- Der Startup-Code ist in Assembler programmiert.
- Dateinamen: oft “crt0.s“ oder “startup.s“
- Der Startup-Code eines Microcontrollers muss typischerweise folgende Aufgaben erledigen:
  - Definition der Vektor-Tabelle
  - CPU-Register initialisieren
  - Stack initialisieren
  - Initialisierte globale Variablen mit dem Initialisierungswert laden.
  - Nichtinitialisierte globale Variablen auf 0 setzen.
  - optional: Board-Support-Package (BSP) initialisieren

Initialisierte globale Variablen

- Der Startup-Code kopiert die Werte für die initialisierten globalen Variablen vom Flash ins RAM.

```
/* Copy the data segment
    initializers from flash to
    SRAM */
    LDR   r0, =_sdata
    LDR   r1, =_edata
    LDR   r2, =_sidata
    MOVS  r3, #0
    B     LoopCopyDataInit

CopyDataInit:
    LDR   r4, [r2, r3]
    STR   r4, [r0, r3]
    ADDS  r3, r3, #4

LoopCopyDataInit:
    ADDS  r4, r0, r3
    CMP   r4, r1
    BCC   CopyDataInit
```



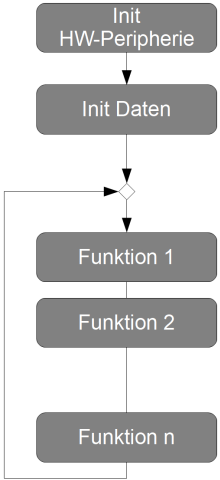
Nichtinitialisierte globale Variablen

- Der Startup-Code löscht die nichtinitialisierten globalen Variablen

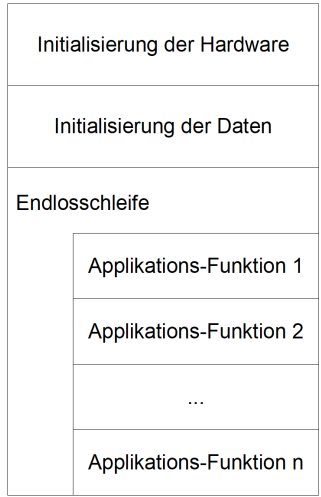
```
/* Zero fill the bss segment. */
    LDR  r2, =_sbss
    LDR  r4, =_ebss
    MOVS r3, #0
    B    LoopFillZerobss

FillZerobss:
    STR  r3, [r2]
    ADDS r2, r2, #4

LoopFillZerobss:
    CMP  r2, r4
    BCC  FillZerobss
```

Die main-Schleife

Ablauf des Hauptprogramms

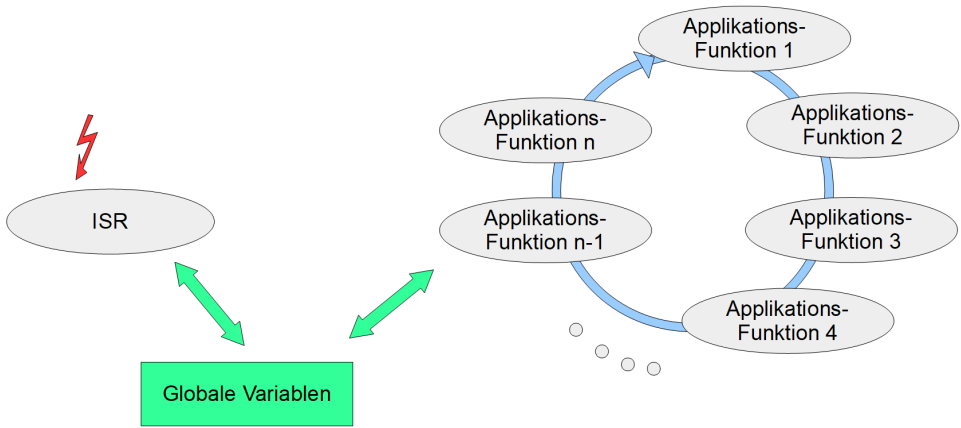


Activity-Diagramm / UML



Nassi-Shneiderman Diagramm

Hauptprogramm mit Interrupts



Projekte in C und Assembler

Übersicht

- Bei sehr hardware-spezifischen Aufgaben kann ein Teil der Software in Assembler programmiert werden:
  - Portierung von Betriebssystemen: Durchführung des Context-Switch.
  - Rechenintensive Algorithmen, um die Hardware (z.B. Multiply-Accumulate Einheit) optimal zu nutzen.
  - Häufig durchlaufene Funktionen, um die System-Performance zu erhöhen.
  - Zugriffe auf CPU-Register, die in C gar nicht adressiert werden können.
- Dabei muss insbesondere berücksichtigt werden:
  - Aufruf von Assembler-Subroutinen aus C-Funktionen und umgekehrt.
  - Parameterübergabe und Rückgabewerte.
  - Verwendung von C-Variablen in Assembler und umgekehrt.
  - Inline-Assembler

Aufruf von Assembler-Subroutinen und C-Fkt.

- Assembler-Subroutinen können aus C-Code und C-Funktionen aus Assembler-Code aufgerufen werden.

- C-Code

```
extern int asmsub(int a1);

int cfunc(int c1){
    ...
    return(0);
}

int main(void){
    ...
    res = asmsub(5);
    ...
}
```

- Assembler-Code

```
    .global asmsub
    .extern cfunc
    .text

asmsub:
    PUSH {lr}
    ...
    MOV  r0,#5   @ set parameter for cfunc
    BL   cfunc   @ call cfunc
    ...
    POP  {pc}    @ return
```

Parameterübergabe, verwendete Register

Es ist zwingend die AAPCS einzuhalten (siehe auch Kapitel Subroutinen)!

- Die ersten vier Parameter werden in r0 bis r3 übergeben.
- Die restlichen Parameter werden (in umgekehrter Reihenfolge) auf dem Stack übergeben.
- Rückgabewerte werden im Register r0 zurückgegeben.
- Verwendete Register (ausser r0 bis r3 und r12) müssen gerettet werden.
- r13 / sp: Stack-Pointer
- r14 / lr: Link-Register
- r15 / pc: Program-Counter

Verwendung gemeinsamer Variablen

- Globale Variablen können in C und Assembler gemeinsam genutzt werden.

- C-Code

```c
/* global variables */
uint32_t c_var = 0;
extern uint32_t asm_var;


void ctest_var(void)
{
    ...
    asm_var = 0x22;
    c_var = 0x55;
}
```

- Assembler-Code

```
    .global  asm_var
    .extern  c_var
    ...
    .data
asm_var: .word 0

    .text
asmtest_var:
    LDR r0,=c_var        @ c_var = 0x44
    MOV r1,#0x44
    STR r1,[r0]
    LDR r0,=asm_var      @ asm_var = 0x11
    MOV r1,#0x11
    STR r1,[r0]
    MOV pc,lr            @ return
```

Inline Assembler

- Assembler-Zeilen können direkt in den C-Code eingefügt werden.
- Beispiel:

```
#define     NOP \
            asm volatile("MOV r0,r0");
```

- Inline-Assembler ist hilfreich für:
  - Ausführung von ARM-Instruktionen, die vom C-Compiler nicht unterstützt werden.
  - Zugriff auf CPU-Register sowie Coprozessor-Register.
  - Beispiele, die Sie kennen: SQRT der FPU, enable / disable Interrupts
  - Inline-Assembler ist nach ANSI-C nicht definiert und ist abhängig vom Compiler! Andere Schreibweisen: "ASM", "__asm" oder "ASMLINE".
  - Achtung: ungewollte Seiteneffekte! Vorsicht ist geboten.
- Inline-Assembler nur bis ca. drei Assembler-Instruktionen verwenden, sonst besser eine Assembler-Subroutine schreiben.
- Weiterführende Informationen: Kapitel 11.6 "Inline Assembler" im Skript.

Inline Assembler [2]

Schreiben Sie folgenden Code in C mit Hilfe von Inline Assembler (asm volatile) :

- Ein Makro ENABLE_INTERRUPTS, welches die Interrupts einer Cortex-M7 CPU global aktiviert (Tipp: Register PRIMASK löschen, Instruktion “CPSIE i“).
- Ein Makro DISABLE_INTERRUPTS, welches die Interrupts einer Cortex-M7 CPU global deaktiviert (Tipp: Register PRIMASK setzen, Instruktion “CPSID i“).