ÜBUNG: Parameterübergabe über Stack (1)

Schreiben Sie ein Unterprogramm, welches folgende Berechnung durchführt:

Erg = (z1+z2)*(z3+z4)

- Die Parameter z1 … z4 sollen über den Stack übergeben werden.
- Das Ergebnis soll über r0 zurückgeben werden.
- Alle verwendeten Register sollen gerettet und wieder restauriert werden.

10.2.3 Lokaler Speicher für Unterprogramme

10.2.3.1 Was ist „lokaler Speicher“ ?

Lokaler Speicher ist …

Speicherplatz, der bereitgestellt wird, wenn das Unterprogramm startet und der wieder freigegeben wird, wenn das Unterprogramm endet.

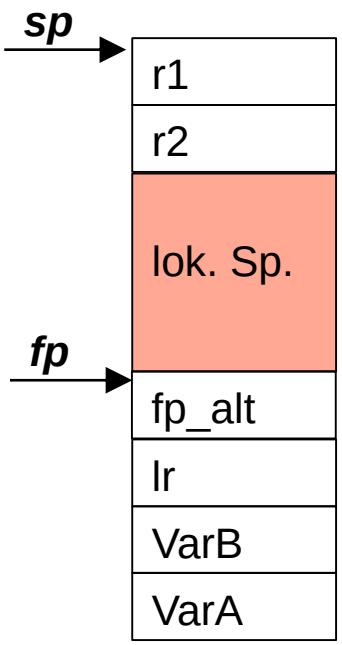
Vorteile:
- effiziente Speichernutzung
- Kapselung interner Daten
- es können gleichzeitig mehrere Instanzen eines Unterprogramms existieren ohne sich gegenseitig zu beinflussen (z.B. Multitasking, rekursive Unterprogramme)

10.2.3.2 Realisierung

```
# Aufrufendes Programm
    str     r4, [sp, #-4]!      ; PUSH VarA    (oder mit
    str     r3, [sp, #-4]!      ; PUSH VarB     push {…})
    bl      MySubroutine
    add     sp, #8              ; Stack korrigieren
```

Über den Framepointer fp kann man bequem auf den lokalen Speicher zugreifen !

```
MySubroutine
    push    {fp, lr}        ; fp und lr retten
    mov     fp , sp         ; aktuelle Stackpos. merken
    sub     sp, #16         ; 16 Byte lok. Speicher (n x 8 Byte)
    push    {r1-r2}         ; PUSH,  Register retten
    ldr     r2, [fp, #8]    ; [r2] ← VarB
    ldr     r1, [fp, #12]   ; [r1] ← VarA
    …….                     ; irgendwas berechnen ……
    …….                     ; und Ergebnis → [r0]
    pop     {r1-r2}         ; POP,  Register restaurieren
    mov     sp , fp         ; sp unter lok. Speicher setzen
    pop     {fp,lr}         ; POP  fp u. lr restaurieren
    bx      lr
```



ÜBUNG: Parameterübergabe über Stack (2) und lok. Speicher

Schreiben Sie ein Unterprogramm, welches folgende Berechnung durchführt:

$$Erg = a\ (1 + b^2 + b^3 + b^4)$$

- Die Parameter a und b (32 Bit Worte) sollen über den Stack übergeben werden.
- Das Ergebnis soll über r0 zurückgeben werden.
- Alle verwendeten Register sollen gerettet und wieder restauriert werden.
- Für Zwischenergebnisse soll 16 Byte lokaler Speicher bereitgestellt und (mindestens teilweise) genutzt werden.

11. ARM : Ein- und Ausgabeschnittstellen und -befehle

- Einführung
- Memory-Mapped-Ein-/Ausgabe
- Polling
- Steuerregister
- Beispiel: General Purpose Input/Output Port (GPIO)

11.1 Einführung

11.1.1 Basisschnittstellen

Für die Interaktion des Prozessors mit der Außenwelt (Ein-/Ausgabe von Signalen) werden Schnittstellen benötigt.

Grundlegende Schnittstellen (oft auch englisch „Interfaces“) sind beispielsweise:

- **GPIO** (General Purpose Input/Output) :
  - Ansteuerung von LED, Lampen, Relais, Signalleitungen, ...
  - Abfrage von Schaltern, Tastaturen, binären Signalpegeln, ...

- **ADC** (Analog Digital Converter)
  - Messen einer analogen Eingangsspannung (z.B. Messwertaufnehmer)
  - Abtasten (Samplen) von Zeitsignalen (z.B. von Musik)

- **DAC** (Digital Analog Converter)
  - Umwandeln einer Digitalzahl in eine analoge Ausgangsspannung
  - Ausgabe von Tönen und Musik

11.1.2 Komplexere Schnittstellen

- **UART** (*Universal Asynchronous Receiver Transmitter*) :
  - serielle Schnittstelle
  - EIA 485 (früher RS 232)

- **PWM** (*Pulse Width Modulation*)
  - z.B. Ansteuerung von Motoren,

- **I2C** (*Inter-Intergrated Circuit*)
  - 2-Draht-Bus, Master-Slave-Bus,
  - für einfache Gerätekommunikation
  - Beispiel: LEGO-NXP-Roboter für Sensoren und Antriebe

- **CAN** (*Controller Area Network*)
  - Feldbussystem (bis 40 m bei 1 MBit/s*, bis zu 110 Teilnehmer, Linienstruktur)
  - Beispiel: Standard bei der Kfz-internen Gerätekommunikation

- **Ethernet**
- **USB**

11.2 I/O-Ports (GPIO)

Die Ein- und Ausgabe von Werten sowie die Programmierung der I/O-Bausteine erfolgt über die Steuer- und Ein-/Ausgaberegister der I/O-Bausteine.
Diese Register werden ähnlich wie Speicher über Adressen angesprochen.

Zwei Prinzipien sind üblich:

- **Memory-mapped-I/O**:
  - I/O-Baustein und Speicher teilen gemeinsam den Adressraum.
  - Die Register der I/O-Bausteine werden wie Speicher angesprochen (→ *ldr, str*).
  - Beispiel: ARM-Cortex, Motorola 68000

- **I/O-mapped:**
  - Die I/O-Bausteine haben einen eigenen Adressraum.
  - Die Register der I/O-Bausteine werden über eigene Befehle angesprochen (→ *in* / *out*)
  - Beispiel: Intel IA-32-Prozessoren

11.3 Reaktion auf Eingangsereignisse

Für die Detektion von Eingangsereignissen (z.B. Tastendruck) sowie die daran anschließende Reaktion darauf können verschiedene Mechanismen angewendet werden:

- **Polling**:
  - Zyklisches Abfragen eines Eingangswertes, solange bis eine bestimmte Bedingung erfüllt ist

- **Interrupt:**
  - asynchrone Unterbrechung der normalen Programmausführung durch EA-Bausteine über spezielle Steuerleitungen (Interruptleitung):

    → Sicherung des aktuellen Prozessorzustandes (Register, Flags)

    → Ausführung der Interruptroutine

    → Wiederherstellung des gespeicherten Prozessorzustandes

    → Fortführung des unterbrochenen Programms

11.4 Ein-/Ausgabe

11.4.1 Cortex-M4 : General-Purpose-Input/Output-Ports

- 9 Ports mit je 16 Bit : GPIO A, GPIO B, ... GPIO I
- Bitweise als Input- oder Outputports programmierbar
- verschiedene I/O-Taktraten programmierbar 2 MHz, 25 MHz, 50 MHz, 100 MHz
- Adressbereiche der Ports

0x4002 2000 - 0x4002 23FF	GPIOI
0x4002 1C00 - 0x4002 1FFF	GPIOH
0x4002 1800 - 0x4002 1BFF	GPIOG
0x4002 1400 - 0x4002 17FF	GPIOF
0x4002 1000 - 0x4002 13FF	GPIOE
0x4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	GPIOC
0x4002 0400 - 0x4002 07FF	GPIOB
0x4002 0000 - 0x4002 03FF	GPIOA

Initialisierungssequenz der Ports beim Cortex-M4

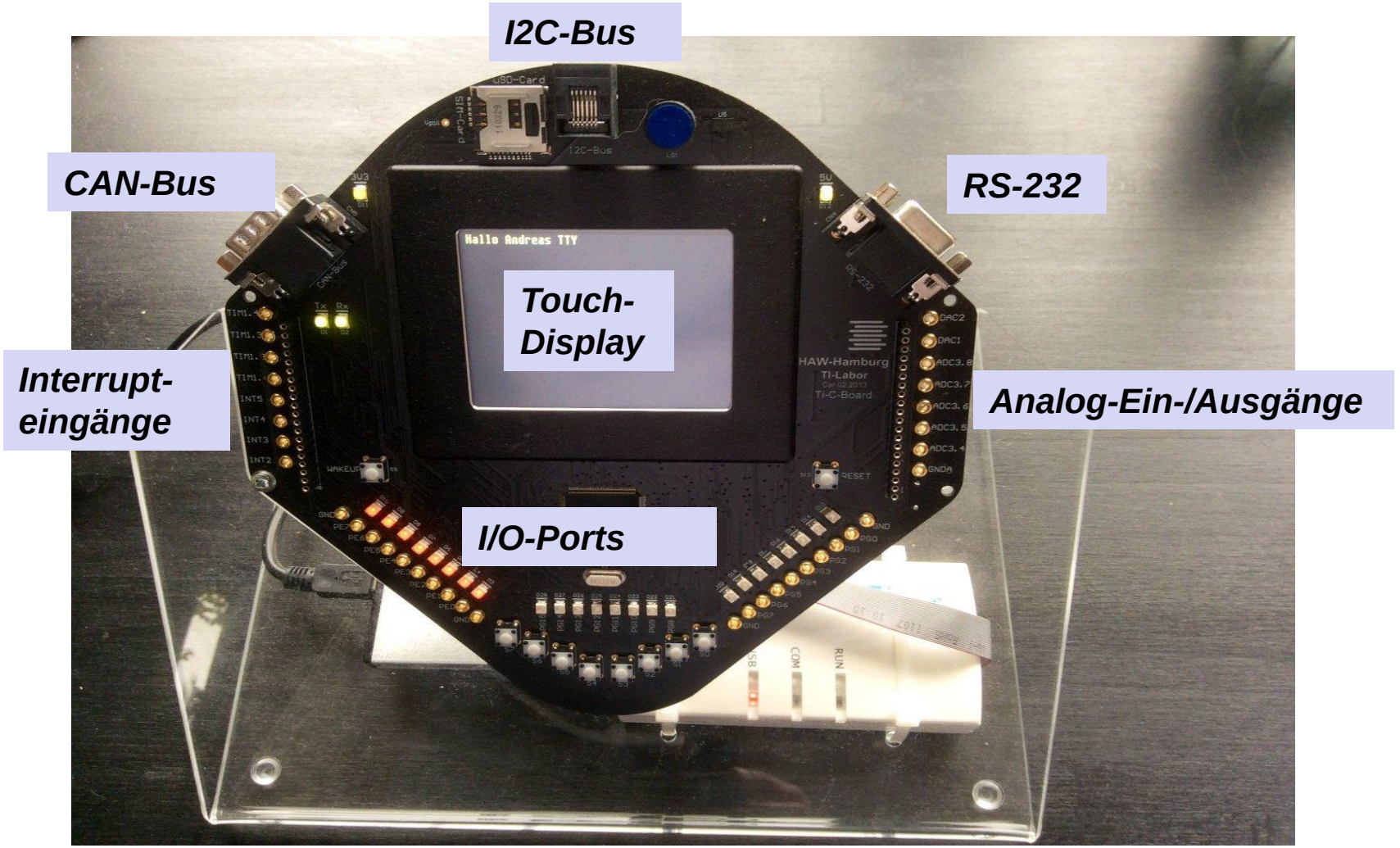
1. Takt der verwendeten Ports einschalten
2. Zu jedem Port
   - verwendete Pins festlegen
   - Modus festlegen (Input, Output)
   - Ausgangsbeschaltung festlegen (Pullup, Pulldown, ...)
   - Taktrate festlegen

Beim TI-Board steht für die korrekte Initialisierung der Ports die Funktion

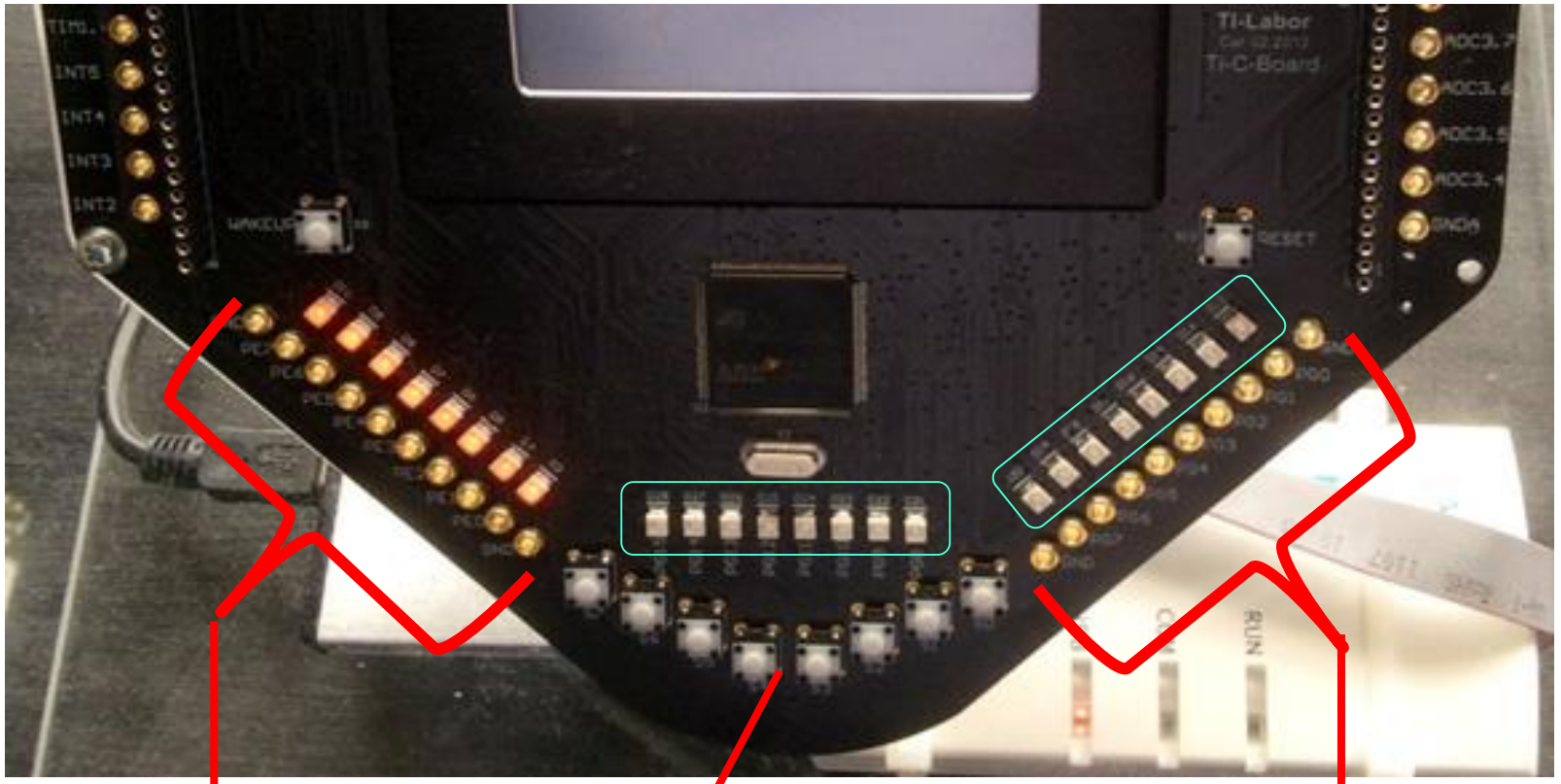
`Init_TI_Board()` → `Init_IO()`

zur Verfügung.

11.4.1 TI-Board



- Port GPIO_E: 8 Eingänge, auf Buchsen und LED herausgeführt (Bit 0 ... 7)
- Port GPIO_G: 8 Ausgänge, auf Buchsen herausgeführt (0 ... 7)
  16 Ausgänge, auf LED herausgeführt (0 ... 15)
- Initialisierung mit `Init_TI_Board()`
- 50 MHz I/O-Taktrate
- **Achtung**: offene Eingänge haben den Wert 1



Eingänge 0 .... 7 des **GPIO_E**

Taster zu den Eingängen

Ausgänge 0 .... 7 des **GPIO_G**

16 LED zu den Ausgängen 0 ... 15

11.4.2 Ein-/Ausgabe auf den GPIO-Ports

Die Ansteuerung bzw. das Auslesen eines 16-bit-GPIO-Ports erfolgt über Register mit den folgenden Adressen:

Lesen von Port E

GPIO_E_PIN ((= 0x 40 02 10 10) : Lesen der Eingangswerte von GPIO_E

Schreiben auf Port G

GPIO_G_PIN (= 0x 40 02 18 10) : Lesen der aktuellen Ausgangswerte

GPIO_G_SET (= 0x 40 02 18 18) : Setzen von Ausgangswerten

GPIO_G_CLR (= 0x 40 02 18 1A) : Löschen von Ausgangswerten

Beipiel:

2_0000000000000001 → GPIO_G_SET setzt Bit 0
und lässt alle anderen Bits unverändert

2_0000000000000001 → GPIO_G_CLR löscht Bit 0
und lässt alle anderen Bits unverändert

Beispiel: GPIO_E lesen und auf GPIO_G ausgeben (Eingabe reflektieren)

1. Portadressen definieren

```
PERIPH_BASE      equ       0x40000000
AHB1PERIPH_BASE  equ       (PERIPH_BASE + 0x00020000)

;blaue LEDs: output
GPIOG_BASE       equ       (AHB1PERIPH_BASE + 0x1800)
GPIO_G_SET       equ       GPIOG_BASE + 0x18
GPIO_G_CLR       equ       GPIOG_BASE + 0x1A
GPIO_G_PIN       equ       GPIOG_BASE + 0x10

;rote LEDs / Taster: input
GPIOE_BASE       equ       (AHB1PERIPH_BASE + 0x1000)
GPIO_E_PIN       equ       GPIOE_BASE + 0x10
```

2. Asm-Programm

```
Reflect_PortE_to_PortG PROC
loop
            ldr  r0, =GPIO_E_PIN     ; Eingangsadr. laden
            ldrb r3, [r0]            ; Eingangswert lesen

            mov  r2, #0xFFFF         ; Löschbits laden
            ldr  r1, =GPIO_G_CLR     ; Adr. CLR-Register laden
            strh r2, [r1]            ; LED 0...15 löschen

            ldr  r1, =GPIO_G_SET     ; Adr. SET-Register laden
            strh r3, [r1]            ; LED setzen

            b   loop
ENDP
```