Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Hardwarenahe Softwareentwicklung

Programmierung Peripherie, Libraries, CMSIS

V5.1, ©2023 roger.weber@bfh.ch

Lernziele

Sie sind in der Lage:

- Vor- und Nachteile durch die Verwendung von Libraries abzuschätzen.
- Einen GPIO mit Hilfe von Library-Funktionen zu programmieren.
- Einen GPIO durch direkten Zugriff auf die HW-Register zu programmieren.

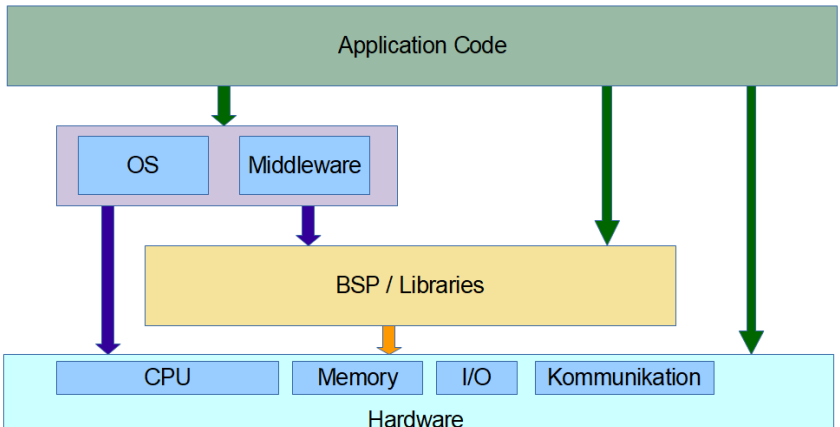
Inhaltsverzeichnis

Übersicht

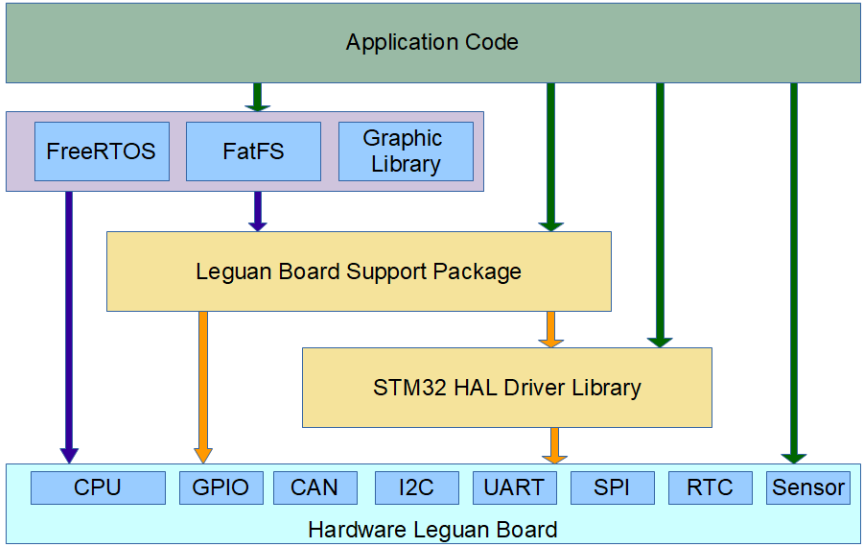
Zugriff auf Hardware-Register

- Auf Peripherie-Bausteine wie serielle Schnittstellen, Timer usw. wird über HW-Register (Memory-Mapped) zugegriffen.
- Es gibt mehrere Möglichkeiten, auf diese HW-Register zuzugreifen:
  - Board Support Package (BSP) des Board-Herstellers.
  - Library des Microcontroller-Herstellers
  - Direkt auf die HW-Register

Hierarchie von Libraries, allgemein



Hierarchie von Libraries, am Beispiel Leguan-Board



Leguan Board Support Package

Übersicht

Aufgaben von Board Support Packages:

- Ansteuerung der Peripherie.
- Vom Hersteller des Boards (Evaluation-Kits, CPU-Module)

Leguan Board Support Package (libleguan)

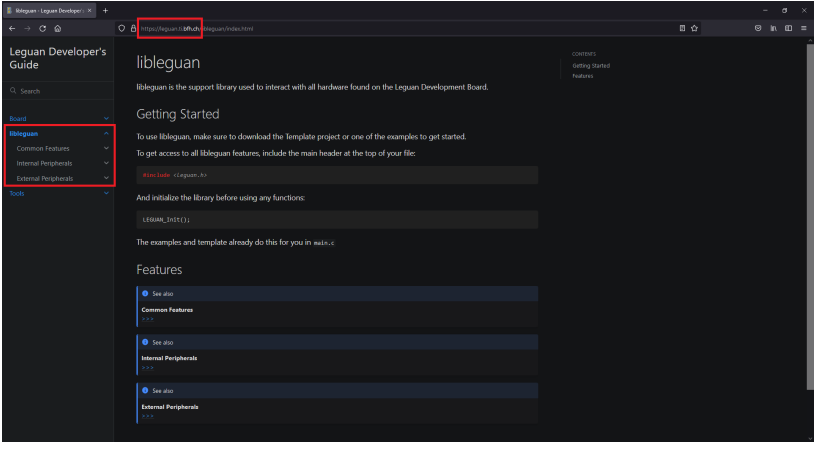
- Von der BFH entwickelt
- Unterstützte Peripherie:
  - 7-Segment, Dipswitch, Buttons, RGB-Led Matrix
  - LCD Display
  - Sensoren

Beispiel BSP Leguan

```c
/* include libleguan*/
#include <leguan.h>

/* Initialize Leguan board */
LEGUAN_Init();

/* read actual value of acceleration sensor */
vector3f_t acceleration;
if (R_SUCCEEDED(SENSOR_GetAcceleration(&acceleration))) {
    float32_t x = acceleration.x * 9.81;
    float32_t y = acceleration.y * 9.81;
    float32_t z = acceleration.z * 9.81;

    // Use acceleration in m/s^2
}
```

Wie finde ich Informationen zum BSP Leguan?

- Auf der Webseite `https://leguan.ti.bfh.ch/`
- Fenster links unter “libleguan“



STM32 HAL Driver Library

Übersicht

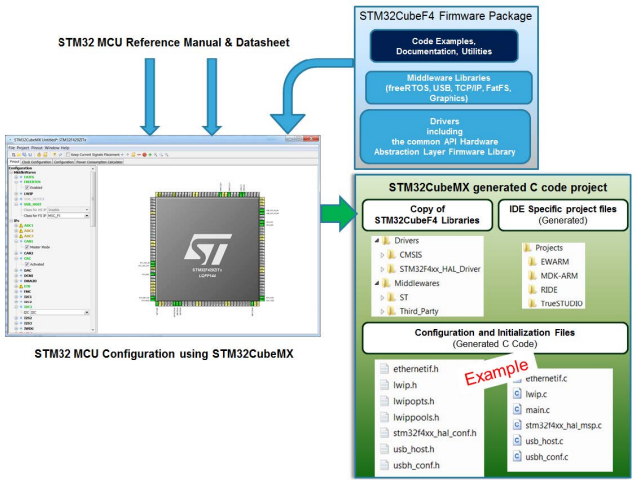
- Von ST Microelectronics für ARM Cortex-Mx CPUs.
- Library-Funktionen für die Ansteuerung der Peripherie des Micrcontrollers.
- Funktionalitäten der HAL Driver Library:
  - CPU
  - GPIO
  - SPI, I2C, USART, Ethernet
  - Timer, Watchdog
  - ADC, DAC
  - DMA
  - SRAM, Flash

Beispiel HAL Driver Library

```
int main(void)
{
    // Initialize HAL
    HAL_Init();

    // Initialize clocks
    SystemClock_Config();
    PeriphCommonClock_Config();

    // Initialize GPIO
    MX_GPIO_Init();

  /* Infinite loop */
  while (1)
  {
      ...
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
  }
}
```

STM32CubeMX Konfigurations-Tool

- Grafisches Tool zur Konfiguration von STM32 Microcontrollern.



CMSIS

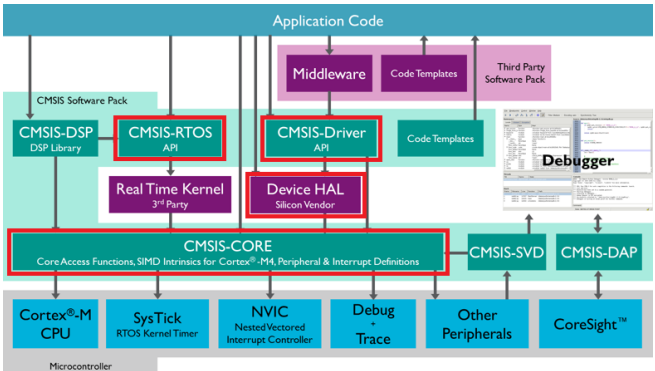
Übersicht

- ARM **C**ortex **M**icrocontroller **S**oftware **I**nterface **S**tandard
- Von ARM definierter Standard für die Programmierung der Cortex-Mx Microcontroller.
- Ziele:
  - Standardisiertes, herstellerunabhängiges Software-Interface.
  - Einheitlicher Programmier-Standard.

Vorteile

→ Portierung von Code auf andere ARM-Microcontroller ist einfach möglich.
→ Kompatible Software-Komponenten unterschiedlicher Hersteller.
→ Unabhängigkeit von der Toolchain.
→ Schneller Lernprozess, Know-How kann auf andere ARM-Microcontroller angewendet werden.

Architektur

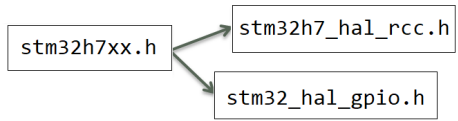


Quelle: siliconlabs.github.io

- CMSIS-CORE: API for the Cortex-Mx processor core and peripherals.
- Device HAL: Driver Library, z.Bsp. von ST.
- CMSIS-Driver: generic peripheral driver interfaces.
- CMSIS-RTOS API: Common API for Real-Time operating systems.

Weitere Definitionen durch CMSIS

File Structure



#include <stm32h7xx.h>

MISRA-C compliance

Tool Independence

```
#if defined ( __GNUC__ )
   #define  __ASM      asm
   #define  __INLINE   inline
#endif
```

Interrupt Service Routines

`typedef enum IRQn`

Suffix «`_Handler`»

Suffix «`_IRQHandler`»

Default-Handler

CMSIS Coding Conventions

Capitel Names for Registers and ASM Instructions

```
GPIOA->MODER
PUSH
```

CamelCase

```
SysTickConfig()
```

Peripheral Prephix

```
GPIO_WriteBit()
```

Data Types

```
#define   __I    volatile const
#define   __O    volatile
#define   __IO   volatile
```

Comments using Doxygen format

```
/**
 * @brief  Enable Interrupt in NVIC Interrupt Controller
 * @param  IRQn_Type  IRQn specifies the interrupt number
 * @return none
 * Enables a device specific interrupt in the NVIC interrupt
 * controller.
 */
Static __INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
}
```

Direkter Zugriff auf die HW-Register

Zugriff auf Hardware-Register in C

- Hardware-Register sind im Memory-Map eingebunden.
- Zugriff aus C mit Hilfe von #define:

```
/* 32-bit Variable auf der Adresse 0x20000008 */
#define reg_32 *((volatile unsigned long *) 0x20000008)
...
reg_32 = 0x12345678;
```

- Zugriff aus C mit Hilfe von const-Pointern:

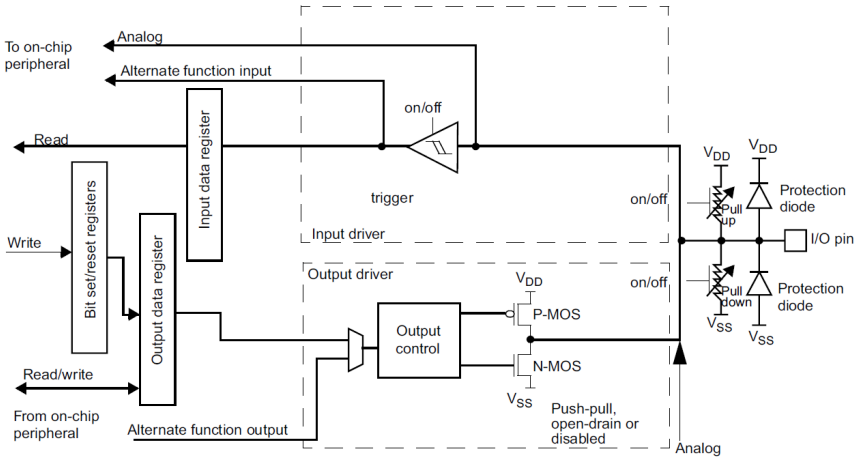
```
/* 32-bit Variable auf der Adresse 0x20000008 */
int* const cp_32 = (int*)0x20000008;
...
*cp_32 = 0x12345678;
```

- Zugriff aus C mit Hilfe von Registerdefinitionen der HAL Driver Library:

```
#include <stm32h7xx.h>
/* GPIO A port output type register, pin 0 push-pull */
GPIOA->OTYPER &= ~(1 << GPIO_PinSource0);
```

GPIO des STM32H743

Ausgangsstufe eines GPIO



Quelle: RM0433 Reference Manual, ST Microelectronics

Register für die Initialisierung des GPIO

Register	Beschreibung	Werte
GPIOx_MODER	Port Mode Register	00 Input 01 General Purpose Output 10 Alternate Function 11 Analog mode
GPIOx_OTYPER	Port Output Type Register	0 Output push-pull 1 Output open-drain
GPIOx_OSPEEDR	Port Output Speed Register	00 Low speed 01 Medium speed 10 Fast speed 11 High speed
GPIOx_PUPDR	Port pull-up/pull-down Register	00 No pull-up/pull-down 01 Pull-up 10 Pull-down 11 Reserved

GPIO Port bit set/reset Register

GPIO port bit set/reset register (GPIOx_BSRR) (x = A to K)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BR[15:0]:** Port x reset I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BS[15:0]:** Port x set I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

Schlussbetrachtung

Schlussbetrachtung

- Vorteile von Libraries
  - Code ist schnell programmiert.
  - Gute Portierbarkeit.
  - Keine low-level Kenntnisse der Hardware erforderlich.

Vorteile

→ schnelle Einarbeitungszeit
→ time to market ist optimal

- Nachteile von Libraries
  - Brauchen zusätzlichen Programmspeicher.
  - Aufrufe sind langsamer als direkter Zugriff auf die HW-Register.