Moderne Low-Level Treiberprogrammierung

CMSIS, MCAL und Co. – Low-Level-Treiber von der Stange

Renate Schultes
MicroConsult GmbH

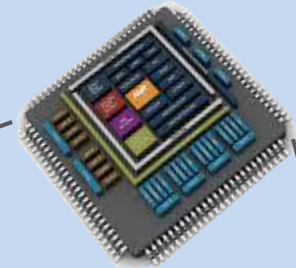
- Die Embedded-Welt
- Klassische LLD-Programmierung
- Software-Schichten-Modelle
- CMSIS, MCAL und Co
- Zusammenfassung

Die Embedded-Welt

- Klassische LLD-Programmierung
- Software-Schichten-Modelle
- CMSIS, MCAL und Co
- Zusammenfassung

- Für die Erfüllung einer vorgegebenen Aufgabe wird ein passender Mikrocontroller ausgewählt.

- Mikrocontroller stellen unterschiedlichste Ressourcen zur Verfügung:
  - ⇨ eine / mehrere CPU(s)
  - ⇨ Programmspeicher
  - ⇨ Datenspeicher
  - ⇨ Peripheriemodule

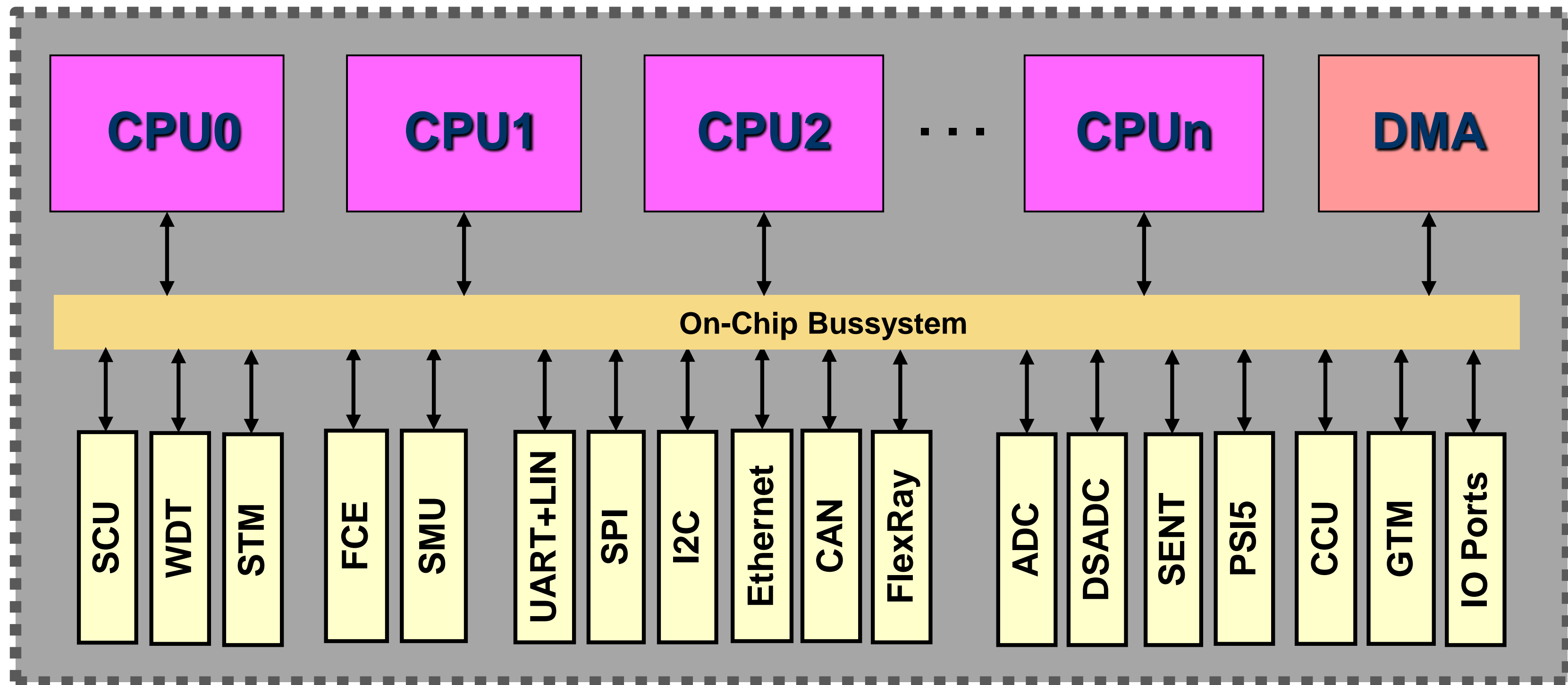


CPU0 | CPU1 | CPU2 | . . . | CPUn | DMA

On-Chip Bussystem

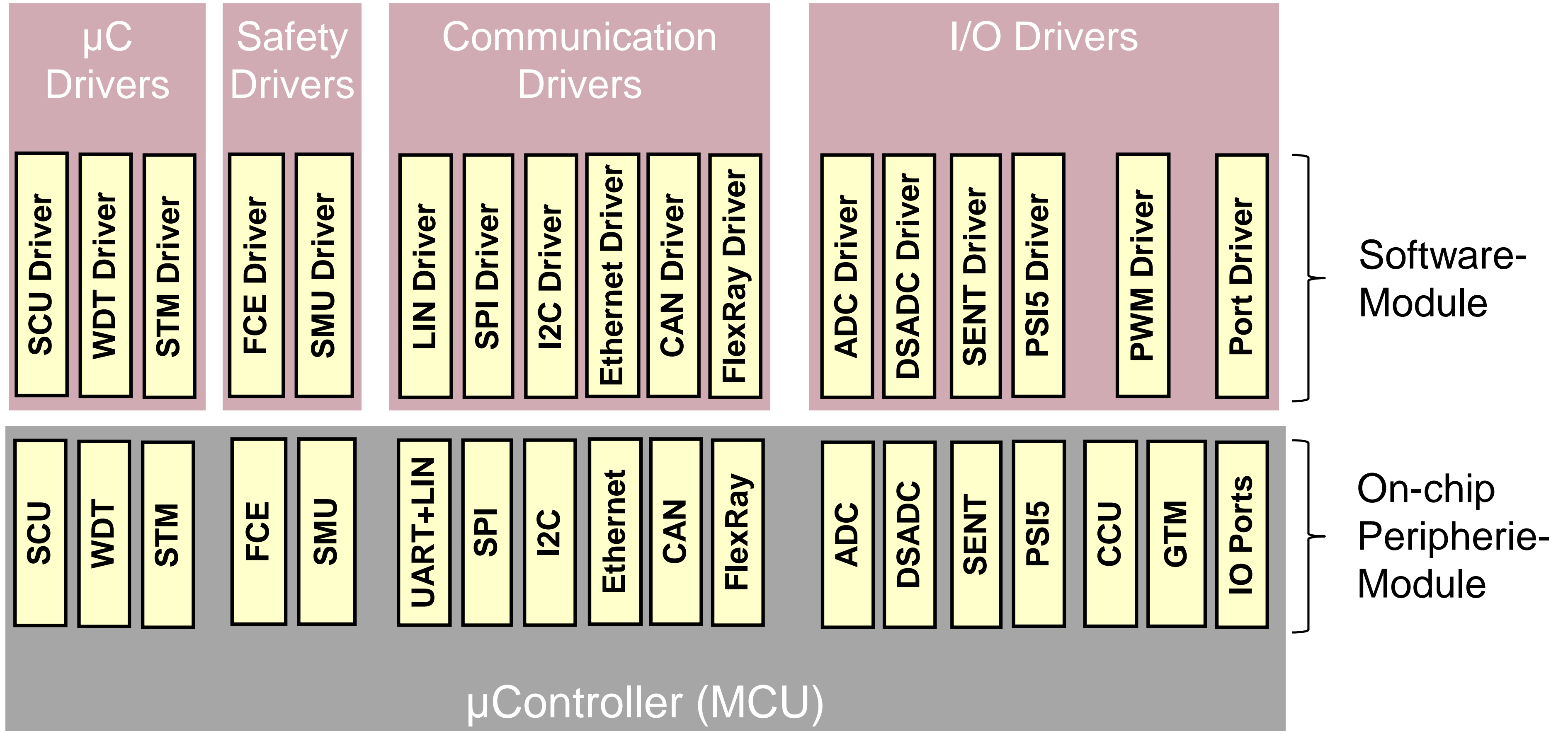
SCU | WDT | STM | FCE | SMU | UART+LIN | SPI | I2C | Ethernet | CAN | FlexRay | ADC | DSADC | SENT | PSI5 | CCU | GTM | IO Ports

MICROCONSULT



- Alle Module des Bausteins müssen abhängig von der Aufgabenstellung richtig und vollständig initialisiert werden.

  ⇨ Detailliertes Hardware-Knowhow ist erforderlich!


µC Drivers
Safety Drivers
Communication Drivers
I/O Drivers
SCU Driver
WDT Driver
STM Driver
FCE Driver
SMU Driver
LIN Driver
SPI Driver
I2C Driver
Ethernet Driver
CAN Driver
FlexRay Driver
ADC Driver
DSADC Driver
SENT Driver
PSI5 Driver
PWM Driver
Port Driver
Software-Module
SCU
WDT
STM
FCE
SMU
UART+LIN
SPI
I2C
Ethernet
CAN
FlexRay
ADC
DSADC
SENT
PSI5
CCU
GTM
IO Ports
On-chip Peripherie-Module
µController (MCU)

- Embedded-Softwareentwicklung ist ein Spagat zwischen verschiedenen Software-Qualitätsmerkmalen.

- Software-Qualitätsmerkmal **Effizienz**

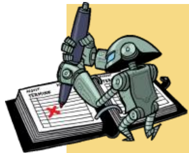
  ⇨ Ressourcen sparend programmieren
  - Codegröße
  - Speicherbedarf
  - Laufzeit
  - Stromverbrauch

- Software-Qualitätsmerkmal **Wiederverwendbarkeit**

  ⇨ Wiederverwendbarkeit spart Entwicklungszeit

MICROCONSULT

- Die Embedded-Welt



- **Klassische LLD-Programmierung**
- Software-Schichten-Modelle
- CMSIS, MCAL und Co
- Zusammenfassung

XE16xREGS.H

```
9903
9904  // Baud Rate Generator Register H
9905  #define U0C0_BRGH              (*((uword volatile far *) 0x20401E))
9906
9907  // Baud Rate Generator Register L
9908  #define U0C0_BRGL              (*((uword volatile far *) 0x20401C))
9909
9910  // Bypass Data Register
9911  #define U0C0_BYP               (*((uword volatile far *) 0x204100))
9912
9913  // Bypass Control Register H
9914  #define U0C0_BYPCRH            (*((uword volatile far *) 0x204106))
9915
9916  // Bypass Control Register L
9917  #define U0C0_BYPCRL            (*((uword volatile far *) 0x204104))
9918
9919  // Channel Configuration Register
9920  #define U0C0_CCFG              (*((uword volatile far *) 0x204018))
9921
9922  // Channel Control Register
9923  #define U0C0_CCR               (*((uword volatile far *) 0x204010))
9924
9925  // IOput Control Register n
9926  #define U0C0_DX0CR             (*((uword volatile far *) 0x204020))
```


MicroConsult

USIC0.C

```
134     U0C0_KSCFG       =  0x0003;       // load U0C0
135                                       // register
136
137     _nop_();  // one cycle delay
138
139     _nop_();  // one cycle delay
140
141     ///  -----------------------------------------------------------------------
142     ///  Configuration of the U0C0 Channel Control Register (Mode & Interrupts):
143     ///  -----------------------------------------------------------------------
144     /// - The USIC channel is disabled
145     /// - The parity generation is disabled
146
147     U0C0_CCR         =  0x0000;       // load U0C0 channel control register
148
149     //   initializes the Universal Serial Interface Channel (USIC) U0C0
150
151     U0C0_ASC_vInit();
152
153     ///  -----------------------------------------------------------------------
154     ///  Configuration of the U0C0 Channel Control Register (Mode & Interrupts):
155     ///  -----------------------------------------------------------------------
156     ///  - ASC (SCI, UART) Protocol is selected
157     ///  - The parity generation is disabled
158
159     U0C0_CCR         =  0x4002;       // load U0C0 channel control register
```

© MicroConsult - Microelectronics Consulting & Training GmbH

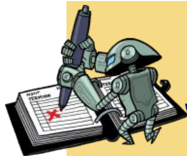
Auszug aus dem Applikationsfile MAIN.C

MAIN.C

```c
141
142  // ----------------------------------------------------------
143  // Initialization of the Peripherals:
144  // ----------------------------------------------------------
145
146  // initializes the Parallel Ports
147  IO_vInit();
148
149  // initializes the Analog / Digital Converter  (ADC0)
150  ADC0_vInit();
151
152  // initializes the Analog / Digital Converter (ADC1)
153  ADC1_vInit();
154
155  // initializes the MultiCAN Module (CAN)
156  CAN_vInit();
157
158  // initializes the USIC0 Module
159  USIC0_vInit();
```

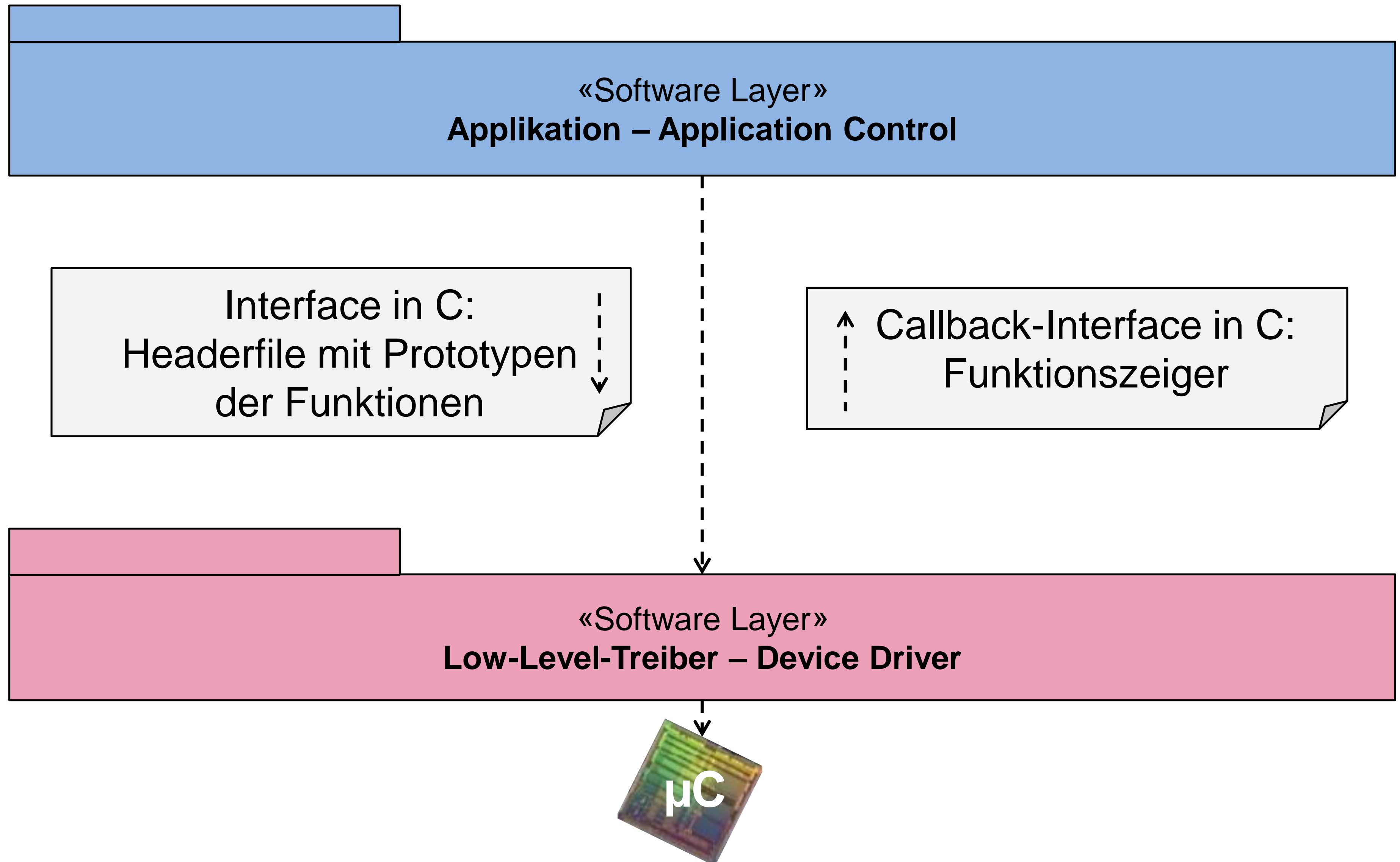

MICROCONSULT

- Die Embedded-Welt
- Klassische LLD-Programmierung

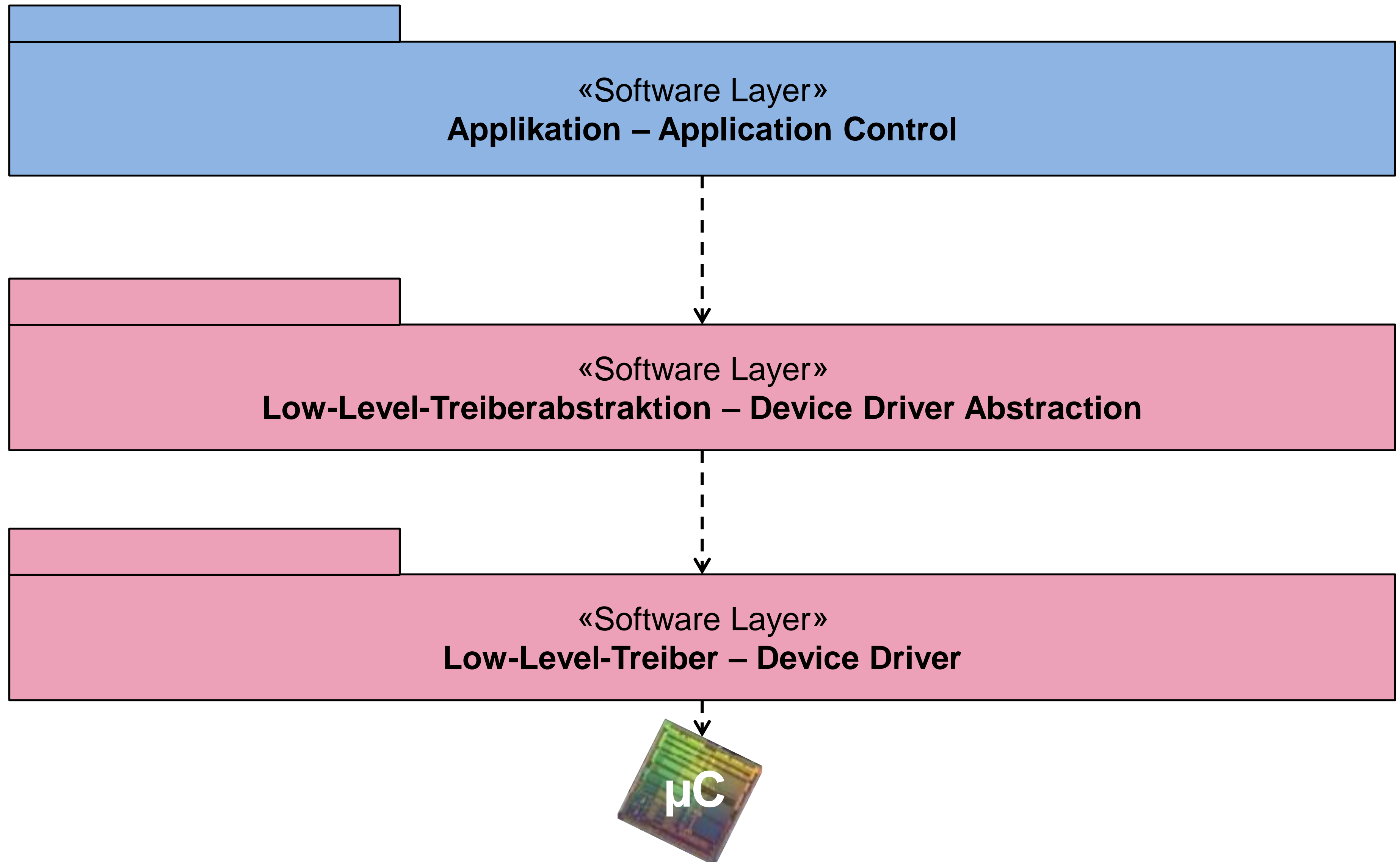


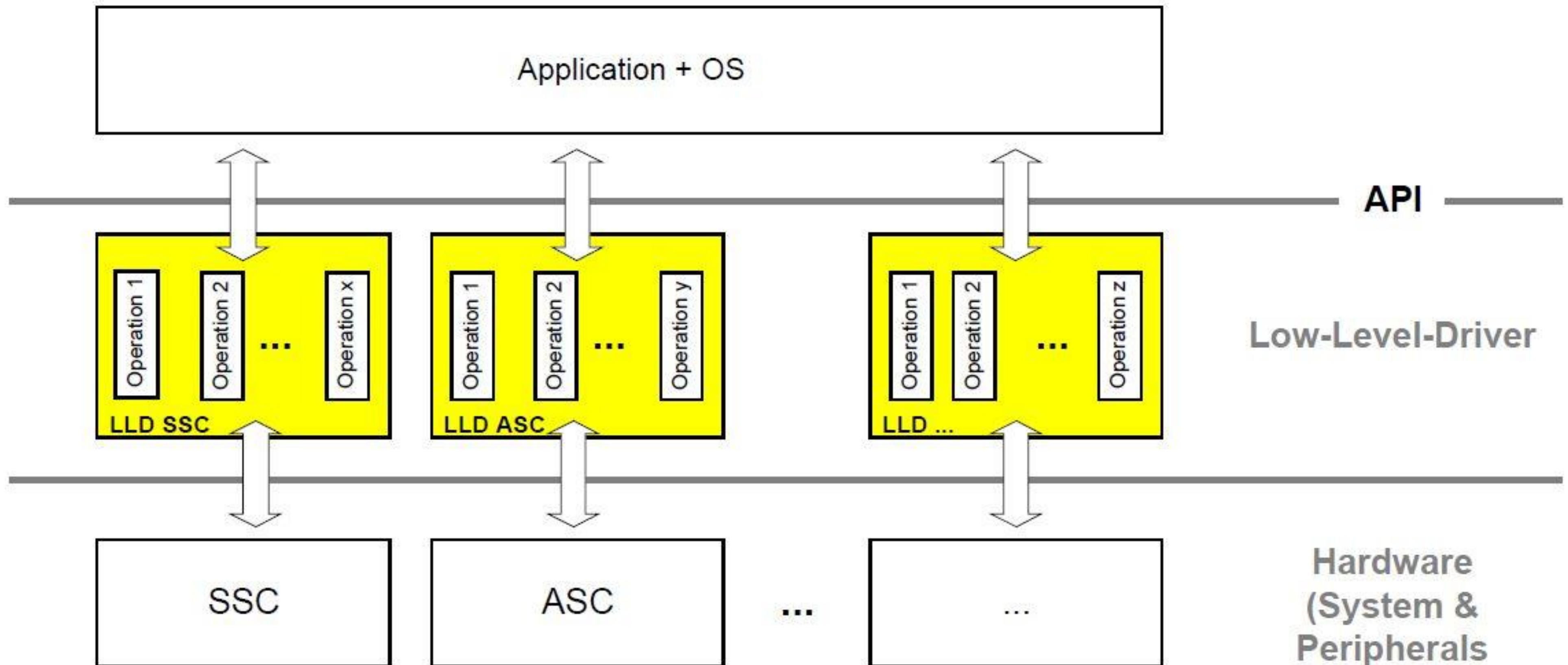
- **Software-Schichten-Modelle**
- CMSIS, MCAL und Co
- Zusammenfassung

2-Schichten-Modell

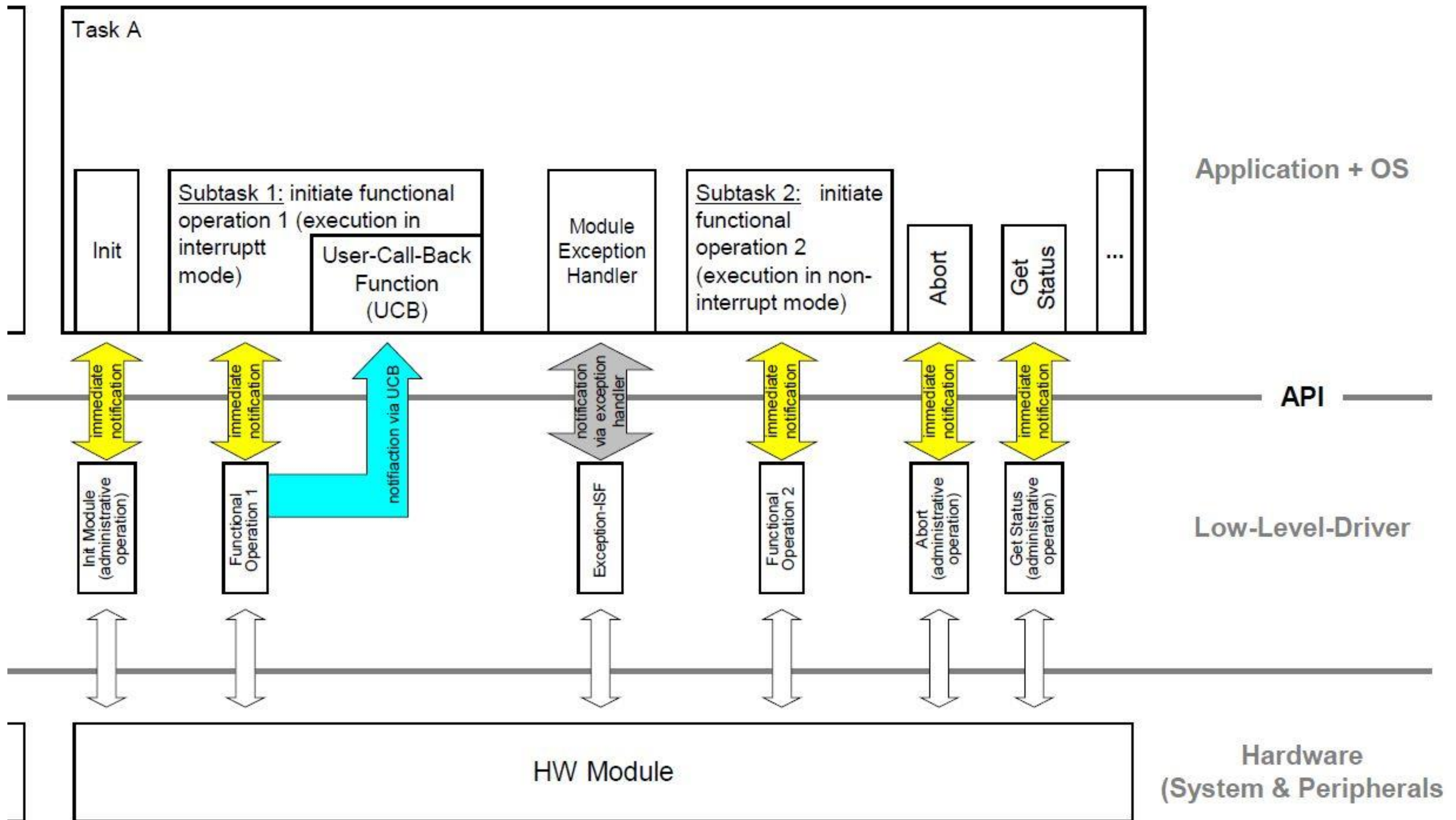


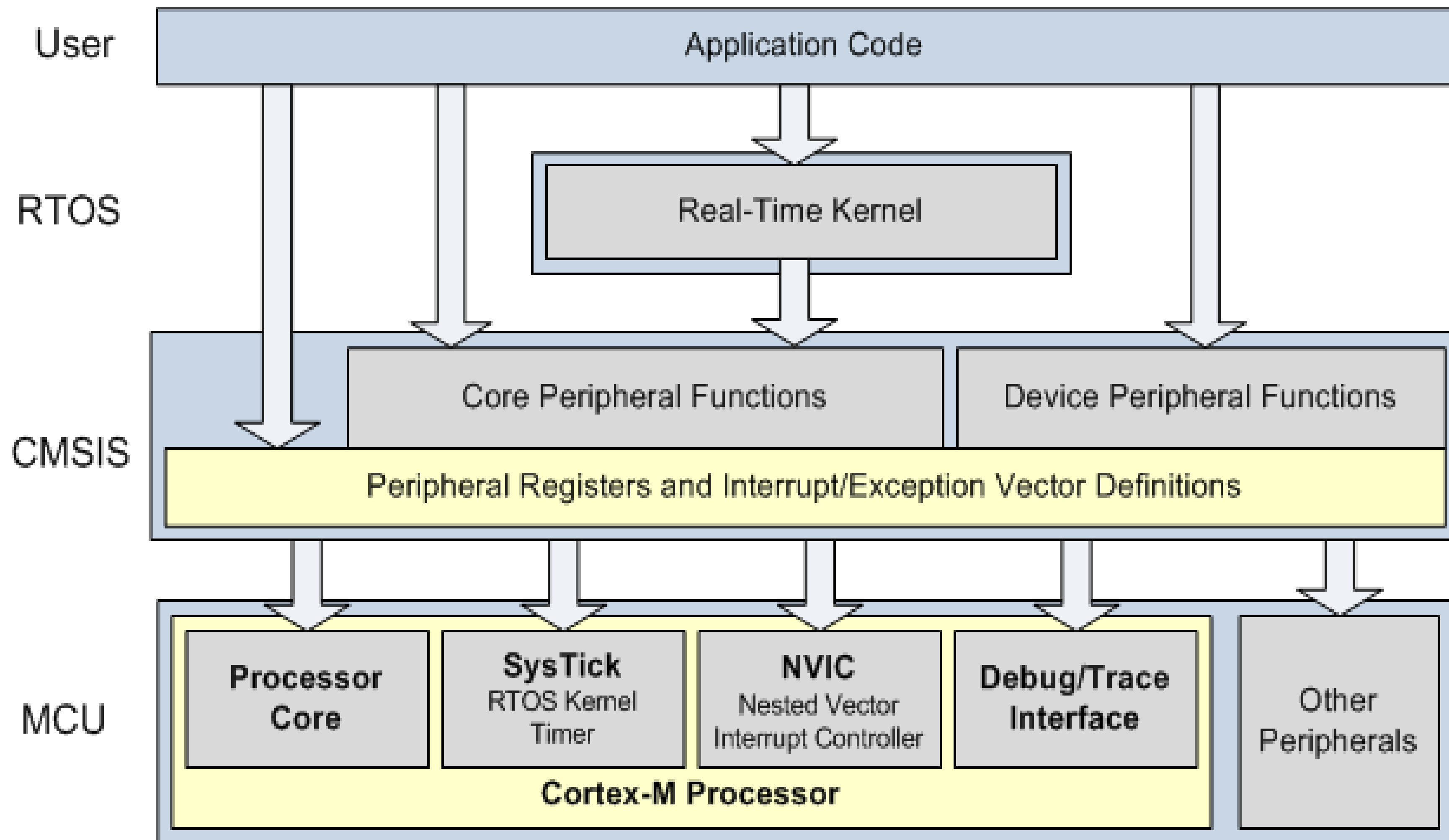
3-Schichten-Modell

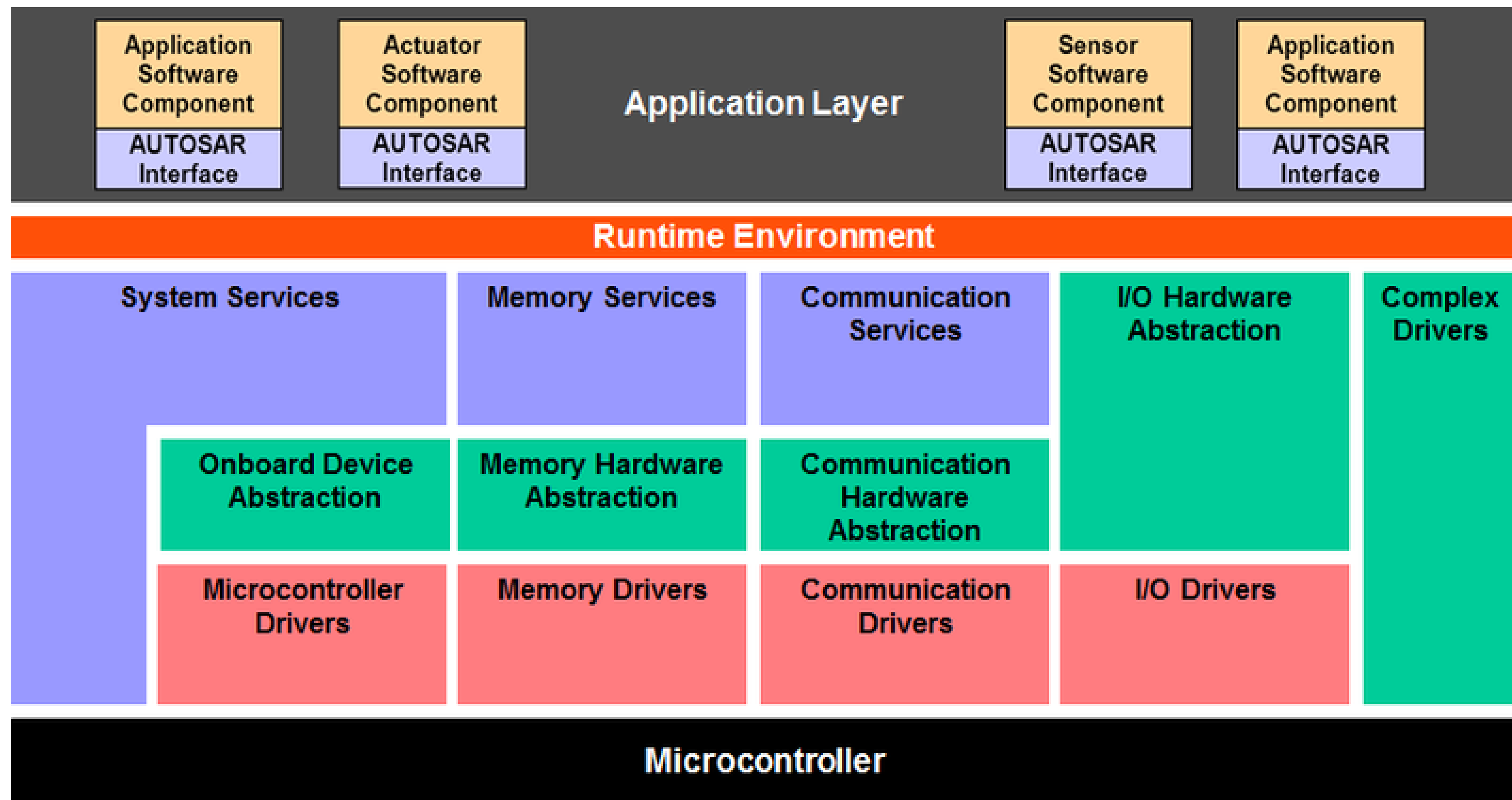




Application + OS
API
Operation 1
Operation 2
...
Operation x
LLD SSC
Operation 1
Operation 2
...
Operation y
LLD ASC
Operation 1
Operation 2
...
Operation z
LLD ...
Low-Level-Driver
SSC
ASC
...
...
Hardware (System & Peripherals

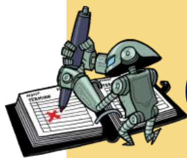

Task A
Init
Subtask 1: initiate functional operation 1 (execution in interruptt mode)
User-Call-Back Function (UCB)
Module Exception Handler
Subtask 2: initiate functional operation 2 (execution in non-interrupt mode)
Abort
Get Status
...
Application + OS
immediate notification
notifiaction via UCB
notification via exception handler
API
Init Module (administrative operation)
Functional Operation 1
Exception-ISF
Functional Operation 2
Abort (administrative operation)
Get Status (administrative operation)
Low-Level-Driver
HW Module
Hardware (System & Peripherals


User
Application Code
RTOS
Real-Time Kernel
CMSIS
Core Peripheral Functions
Device Peripheral Functions
Peripheral Registers and Interrupt/Exception Vector Definitions
MCU
Processor Core
SysTick
RTOS Kernel Timer
NVIC
Nested Vector Interrupt Controller
Debug/Trace Interface
Other Peripherals
Cortex-M Processor



MicroConsult

- Die Embedded-Welt
- Klassische LLD-Programmierung
- Software-Schichten-Modelle



- **CMSIS, MCAL und Co**
- Zusammenfassung

CMSIS: Cortex Microcontroller Software Interface Standard

Standard: Allgemein anerkannte einheitliche Vorgehensweise

MCAL: Microcontroller Abstraction Layer

HAL: Hardware Abstraction Layer

AUTOSAR: AUTomotive Open System ARchitecture

API: Application Programmers Interface

CMSIS – Device Specific Driver Packs

Pack Installer - C:\Keil_v5\ARM\PACK

File Packs Window Help

Device: Infineon - XMC4000

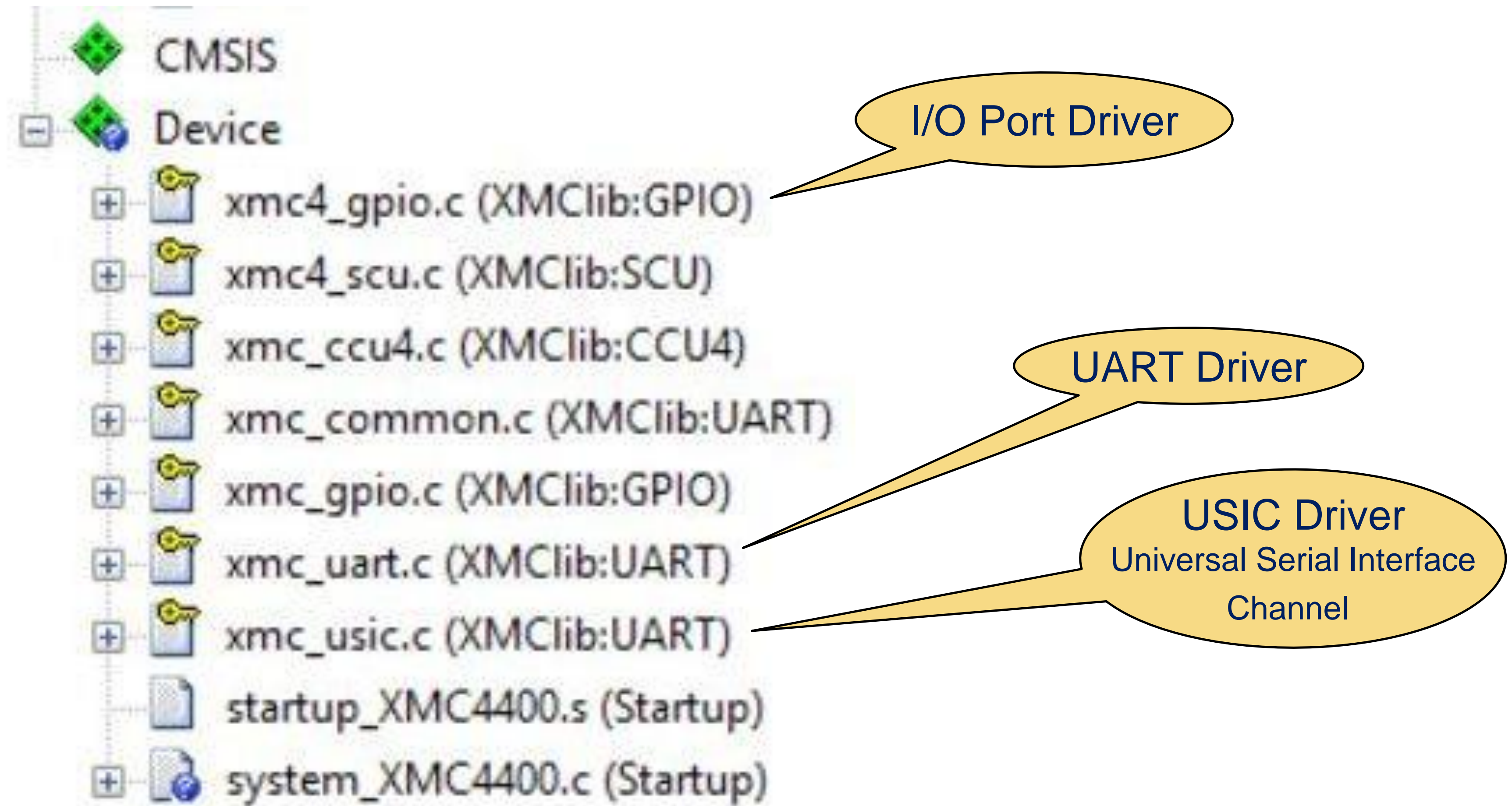
Device	Summary
All Devices	4401 Devices
ABOV Semiconductor	10 Devices
Active-Semi	4 Devices
Ambiq Micro	6 Devices
Analog Devices	12 Devices
ARM	46 Devices
Atmel	335 Devices
Cypress	425 Devices
GigaDevice	99 Devices
HDSC	24 Devices
Holtek	100 Devices
Infineon	166 Devices
TLE98xx Series	20 Devices
XMC1000	106 Devices
XMC4000	40 Devices
Lapis Semiconductor	2 Devices

Pack	Action	Description
Device Specific	2 Packs	XMC4000 selected
Infineon::XMC4000_DFP	Up to da...	Infineon XMC4000 Series Device Support, Drivers and Examples
Oryx-Embedded::Midd...	Install	Middleware Package (CycloneTCP, CycloneSSL and CycloneCrypt
Generic	20 Packs	
ARM::AMP	Install	Software components for inter processor communication (Asymr
ARM::CMSIS	Update	CMSIS (Cortex Microcontroller Software Interface Standard)
ARM::CMSIS-Driver_Val...	Install	CMSIS-Driver Validation
ARM::CMSIS-FreeRTOS	Install+	Bundle of FreeRTOS for Cortex-M and Cortex-A
ARM::CMSIS-RTOS_Vali...	Install	CMSIS-RTOS Validation
ARM::mbedClient	Install	ARM mbed Client for Cortex-M devices
ARM::mbedTLS	Install	ARM mbed Cryptographic and SSL/TLS library for Cortex-M devi
ARM::minar	Install	mbed OS Scheduler for Cortex-M devices
Huawei::LiteOS	Install	Huawei LiteOS kernel Software Pack
Keil::ARM_Compiler	Update	Keil ARM Compiler extensions for ARM Compiler 5 and ARM Cor
Keil::Jansson	Install	Jansson is a C library for encoding, decoding and manipulating J!
Keil::MDK-Middleware	Update	Middleware for Keil MDK-Professional and MDK-Plus

Output

Update available for Infineon::XMC1000_DFP (installed: 2.5.1, available: 2.7.1)
Update available for Keil::ARM_Compiler (installed: 1.3.0, available: 1.3.3)
Update available for Keil::MDK-Middleware (installed: 7.4.0, available: 7.4.1)

Completed requested actions ONLINE


CMSIS
Device
xmc4_gpio.c (XMClib:GPIO)
xmc4_scu.c (XMClib:SCU)
xmc_ccu4.c (XMClib:CCU4)
xmc_common.c (XMClib:UART)
xmc_gpio.c (XMClib:GPIO)
xmc_uart.c (XMClib:UART)
xmc_usic.c (XMClib:UART)
startup_XMC4400.s (Startup)
system_XMC4400.c (Startup)
I/O Port Driver
UART Driver
USIC Driver
Universal Serial Interface Channel

USIC Channel SFR Struktur (1..)

```
typedef struct XMC_USIC_CH
{
  __I  uint32_t  RESERVED0;
  __I  uint32_t  CCFG;        /**< Channel configuration register*/
  __I  uint32_t  RESERVED1;
  __IO uint32_t  KSCFG;       /**< Kernel state configuration register*/
  __IO uint32_t  FDR;       /**< Fractional divider configuration register*/
  __IO uint32_t  BRG;       /**< Baud rate generator register*/
  __IO uint32_t  INPR;        /**< Interrupt node pointer register*/
  __IO uint32_t  DXCR[6];     /**< Input control registers DX0 to DX5.*/
  __IO uint32_t  SCTR;        /**< Shift control register*/
  __IO uint32_t  TCSR;

  union {
    __IO uint32_t  PCR_IICMode; /**< I2C protocol configuration register*/
    __IO uint32_t  PCR_IISMode; /**< I2S protocol configuration register*/
    __IO uint32_t  PCR_SSCMode; /**< SPI protocol configuration register*/
    __IO uint32_t  PCR;       /**< Protocol configuration register*/
    __IO uint32_t  PCR_ASCMode; /**< UART protocol configuration register*/
  };
  __IO uint32_t  CCR;       /**< Channel control register*/
  __IO uint32_t  CMTR;        /**< Capture mode timer register*/

  union {
    __IO uint32_t  PSR_IICMode; /**< I2C protocol status register*/
    __IO uint32_t  PSR_IISMode; /**< I2S protocol status register*/
    __IO uint32_t  PSR_SSCMode; /**< SPI protocol status register*/
    __IO uint32_t  PSR;       /**< Protocol status register*/
    __IO uint32_t  PSR_ASCMode; /**< UART protocol status register*/
  };
...
```


MicroConsult

USIC Channel SFR Struktur (..2)

...

```
  __O  uint32_t  PSCR;       /**< Protocol status clear register*/
  __I  uint32_t  RBUFSR;     /**< Receive buffer status register*/
  __I  uint32_t  RBUF;       /**< Receive buffer register*/
  __I  uint32_t  RBUFD;      /**< Debug mode receive buffer register*/
  __I  uint32_t  RBUF0;      /**< Receive buffer 0*/
  __I  uint32_t  RBUF1;      /**< Receive buffer 1*/
  __I  uint32_t  RBUF01SR;     /**< Receive buffer status register*/
  __O  uint32_t  FMR;      /**< Flag modification register*/
  __I  uint32_t  RESERVED2[5];
  __IO uint32_t  TBUF[32];     /**< Tranmsit buffer registers*/
  __IO uint32_t  BYP;      /**< FIFO bypass register*/
  __IO uint32_t  BYPCR;      /**< FIFO bypass control register*/
  __IO uint32_t  TBCTR;      /**< Transmit FIFO control register*/
  __IO uint32_t  RBCTR;      /**< Receive FIFO control register*/
  __I  uint32_t  TRBPTR;     /**< Transmit/recive buffer pointer register*/
  __IO uint32_t  TRBSR;      /**< Transmit/receive buffer status register*/
  __O  uint32_t  TRBSCR;     /**< Transmit/receive buffer status clear register*/
  __I  uint32_t  OUTR;       /**< Receive FIFO output register*/
  __I  uint32_t  OUTDR;      /**< Receive FIFO debug output register*/
  __I  uint32_t  RESERVED3[23];
  __O  uint32_t  IN[32];     /**< Transmit FIFO input register*/
} XMC_USIC_CH_t;
```


MicroConsult

UART Initialisierungsstruktur

```
/*********************************
 * DATA STRUCTURES
 *******************************************/

/**
 * UART initialization structure
*/
typedef struct XMC_UART_CH_CONFIG
{
  uint32_t baudrate;
  uint8_t data_bits;

  uint8_t frame_length;

  uint8_t stop_bits;
  uint8_t oversampling;
  XMC_USIC_CH_PARITY_MODE_t parity_mode;

} XMC_UART_CH_CONFIG_t;
```


MicroConsult

UART API

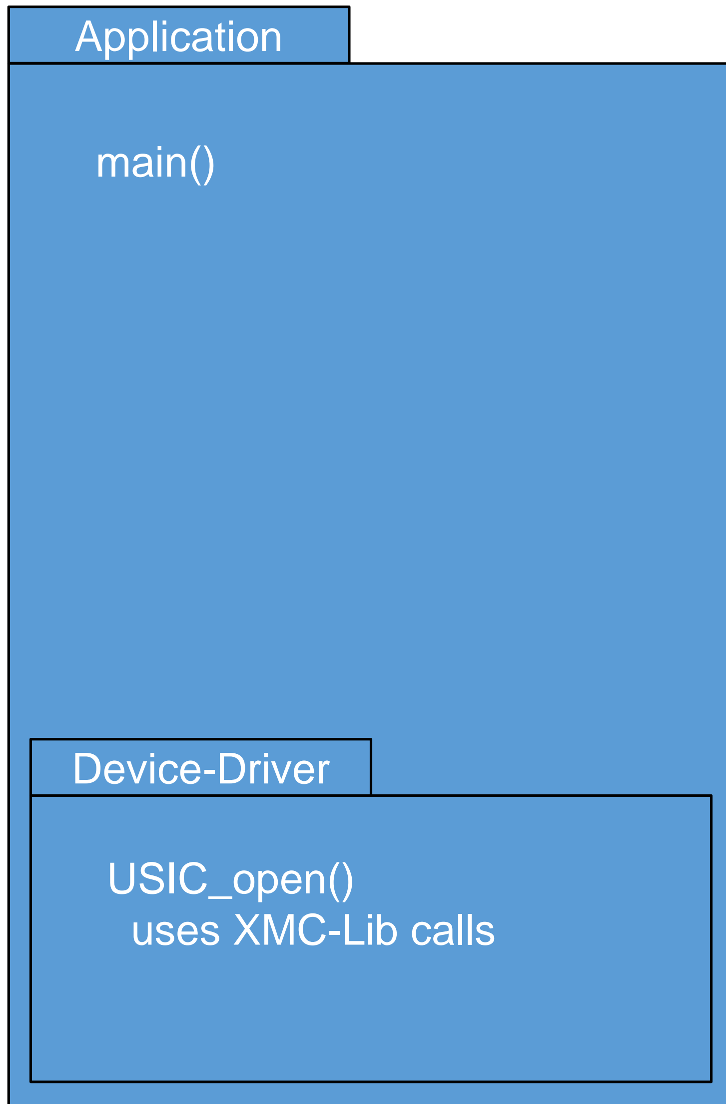
```
 * \par<b>Related APIs:</b><BR>
 * XMC_UART_CH_Start(), XMC_UART_CH_Stop(), XMC_UART_CH_Transmit()\n\n\n
 */
```

```
void XMC_UART_CH_Init(XMC_USIC_CH_t *const channel,
                      XMC_UART_CH_CONFIG_t *const config);
```

```
 * @param channel Constant pointer to USIC ch
 *          \b Range: @ref XMC_UART0_CH0, @re                          ),@ref
 * @return None
 *
 * \par<b>Description</b><br>
 * Sets the USIC channel operation mode to UART mode.\n\n
 * CCR register bitfield \a Mode is set to 2(UART mode). This API should be called after configuring
 * the USIC channel. Transmission and reception can happen only when the UART mode is set.
 * This is an inline function.
 *
 * \par<b>Related APIs:</b><BR>
 * XMC_UART_CH_Stop(), XMC_UART_CH_Transmit()\n\n\n
 */
__STATIC_INLINE void XMC_UART_CH_Init(XMC_USIC_CH_t *const channel)
{
  channel->CCR = (uint32_t)(((channel->CCR) & (~USIC_CH_CCR_MODE_Msk)) | (uint32_t)XMC_USIC_CH_OPERATING_MODE_UART);
}
```


MicroConsult

Beispiel: Hardcoded UART 0 Channel 0 Konfiguration mithilfe der XMCLib



- Main ruft **`USIC_open()`** ohne Parameter auf
- Es gibt keinen Hardware Abstraction Layer
- Wahl der Schnittstelle und Konfiguration ist hardcoded

Beispiel: Hardcoded UART 0 Channel 0 Konfiguration mithilfe der XMCLib

❶ ②

```
#include "xmc_uart.h"
#include "xmc_gpio.h"
#include "xmc_usic.h"

void USIC_open( void )
{
   XMC_UART_CH_CONFIG_t uart_config =
   {
     .baudrate        = 19200,
     .data_bits       = 8,
     .frame_length    = 8,
     .stop_bits       = 1,
     .oversampling    = 16,
     .parity_mode     = XMC_USIC_CH_PARITY_MODE_NONE
   };
   XMC_GPIO_CONFIG_t rx_config =
   {
    .mode            = XMC_GPIO_MODE_INPUT_TRISTATE,
    .output_level    = XMC_GPIO_OUTPUT_LEVEL_HIGH,
    .output_strength = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SOFT_EDGE
   };
   XMC_GPIO_CONFIG_t tx_config =
   {
    .mode            = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT2,
    .output_level    = XMC_GPIO_OUTPUT_LEVEL_HIGH,
    .output_strength = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SOFT_EDGE
   };
```

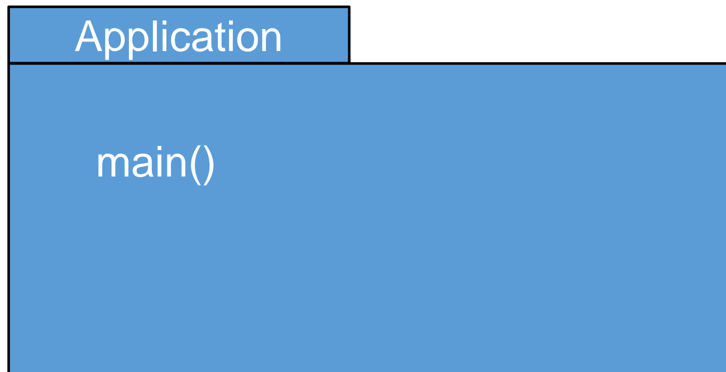
Beispiel: Hardcoded UART 0 Channel 0 Konfiguration mithilfe der XMCLib

1 2

```
    /*Initialize and configure UART0 on channel 0 */
    XMC_UART_CH_Init(XMC_UART0_CH0, &uart_config);
    /*Configure RX*/
    XMC_GPIO_Init(P1_4, &rx_config);
    /*Configure TX*/
    XMC_GPIO_Init(P1_5, &tx_config);
    /*Set input source path*/
    XMC_USIC_CH_SetInputSource(XMC_UART0_CH0, XMC_USIC_CH_INPUT_DX0, 1U);
    /*Configure transmit FIFO*/
    XMC_USIC_CH_TXFIFO_Configure(XMC_UART0_CH0, 16U, XMC_USIC_CH_FIFO_SIZE_16WORDS, 1U);
    /*Configure receive FIFO*/
    XMC_USIC_CH_RXFIFO_Configure(XMC_UART0_CH0, 0U, XMC_USIC_CH_FIFO_SIZE_16WORDS, 15U);
    /*Start UART */
    XMC_UART_CH_Start(XMC_UART0_CH0);
}

int main( void )
{
    /*UART0_CH0 */
    USIC_open();
    /*Write an A */
    XMC_UART_CH_Transmit(XMC_UART0_CH0, 'A');
}
```

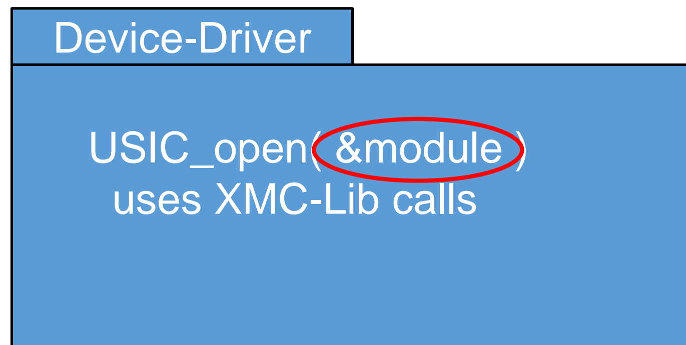

MICROCONSULT

Beispiel: Init Struktur stellt Konfigurationsdaten zur Verfügung



- Main füllt Init Struktur nach Bedarf und
- ruft **`USIC_open()`** mit Init Struktur auf

- Direkte Abhängigkeit vom Device Layer
- Es gibt keinen Hardware Abstraction Layer



- Das Device Driver Modul wird durch die Init Struktur parametrisiert
  z.B.
  - Schnittstelle USIC 0 Channel 0
  - Baudrate 19600
  - …

MicroConsult

Beispiel: Init Struktur stellt Konfigurationsdaten zur Verfügung

main.c

Definition in USIC.h

```
#include "usic.h"

int main(void)
{    USIC_MODULE_t module =
     {    .usic_module                = XMC_UART0_CH0,
          .usic_ch_input              = XMC_USIC_CH_INPUT_DX0,
          .usic_ch_inp_source         = 1U,
          .usic_ch_fifo_size          = XMC_USIC_CH_FIFO_SIZE_16WORDS,
          .uart_config                =
          { .baudrate         = 19200,
             .data_bits       = 8,
             .frame_length    = 8,
             .stop_bits       = 1,
             .oversampling    = 16,
             .parity_mode     = XMC_USIC_CH_PARITY_MODE_NONE
          }
     };
     /*open UART with settings in module struct */
     USIC_open( &module );
     /*Write an A */
     XMC_UART_CH_Transmit(module.usic_module, 'A');

     while(1){}
}
```


MicroConsult

Beispiel: Init Struktur stellt Konfigurationsdaten zur Verfügung

usic.h

1 2 3 4

```
#ifndef __USIC_H
#define __USIC_H

#include "xmc_uart.h"
#include "xmc_gpio.h"
#include "xmc_usic.h"

typedef struct
{
    XMC_USIC_CH_t *usic_module;                    //XMC_USIC1_CH0
    XMC_USIC_CH_INPUT_t usic_ch_input;             //XMC_USIC_CH_INPUT_DX0
    uint8_t usic_ch_inp_source;                    //(1U)
    XMC_USIC_CH_FIFO_SIZE_t usic_ch_fifo_size      //FIFO_SIZE_16WORDS
    XMC_UART_CH_CONFIG_t uart_config;

} USIC_MODULE_t;

//prototype
void USIC_open( USIC_MODULE_t *const module );

#endif //__USIC_H
```


MICROCONSULT

Beispiel: Init Struktur stellt Konfigurationsdaten zur Verfügung

Teil1 usic.c

1 2 3 4

```
void USIC_open( USIC_MODULE_t *const module )
{   /*Initialize and configure UART0 on channel 0 */
    XMC_UART_CH_Init(module->usic_module, &module->uart_config);

    /*Configure RX*/
    {   XMC_GPIO_CONFIG_t rx_config =
        {
            .mode               = XMC_GPIO_MODE_INPUT_TRISTATE,
            .output_level       = XMC_GPIO_OUTPUT_LEVEL_HIGH,
            .output_strength    = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SOFT_EDGE
        };
        XMC_GPIO_Init(P1_4, &rx_config);
    }

    /*Configure TX*/
    {   XMC_GPIO_CONFIG_t tx_config =
        {
            .mode               = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT2,
            .output_level       = XMC_GPIO_OUTPUT_LEVEL_HIGH,
            .output_strength    = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SOFT_EDGE
        };
        XMC_GPIO_Init(P1_5, &tx_config);
    }
    …
```

Beispiel: Init Struktur stellt Konfigurationsdaten zur Verfügung

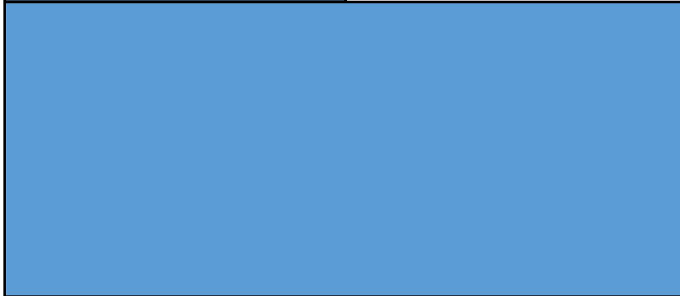
Teil2 usic.c

1 2 3 4

```
…

/*Set input source path*/
XMC_USIC_CH_SetInputSource(   module->usic_module
                            , module->usic_ch_input
                            , module->usic_ch_inp_source);

/*Configure transmit FIFO*/
XMC_USIC_CH_TXFIFO_Configure(  module->usic_module
                             , 16U
                             , module->usic_ch_fifo_size
                             , 1U);

/*Configure receive FIFO*/
XMC_USIC_CH_RXFIFO_Configure(  module->usic_module
                             , 0U
                             , module->usic_ch_fifo_size
                             , 15U);

/*Start UART */
XMC_UART_CH_Start( module->usic_module);
}
```


MicroConsult

Example: HAL Driver

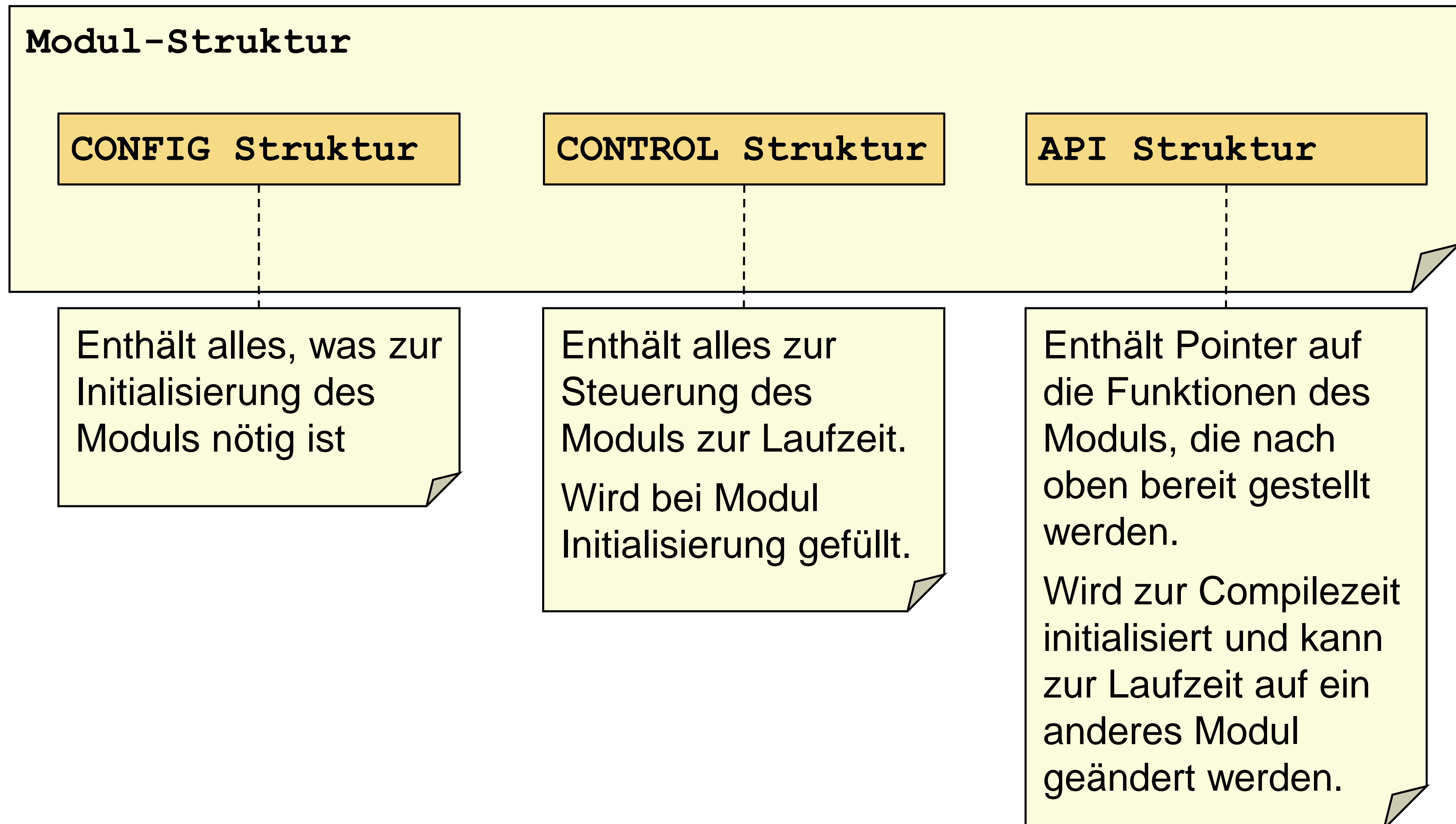


- Main füllt CONFIG Struktur nach Bedarf und
- ruft HAL Funktionen über API Struktur auf

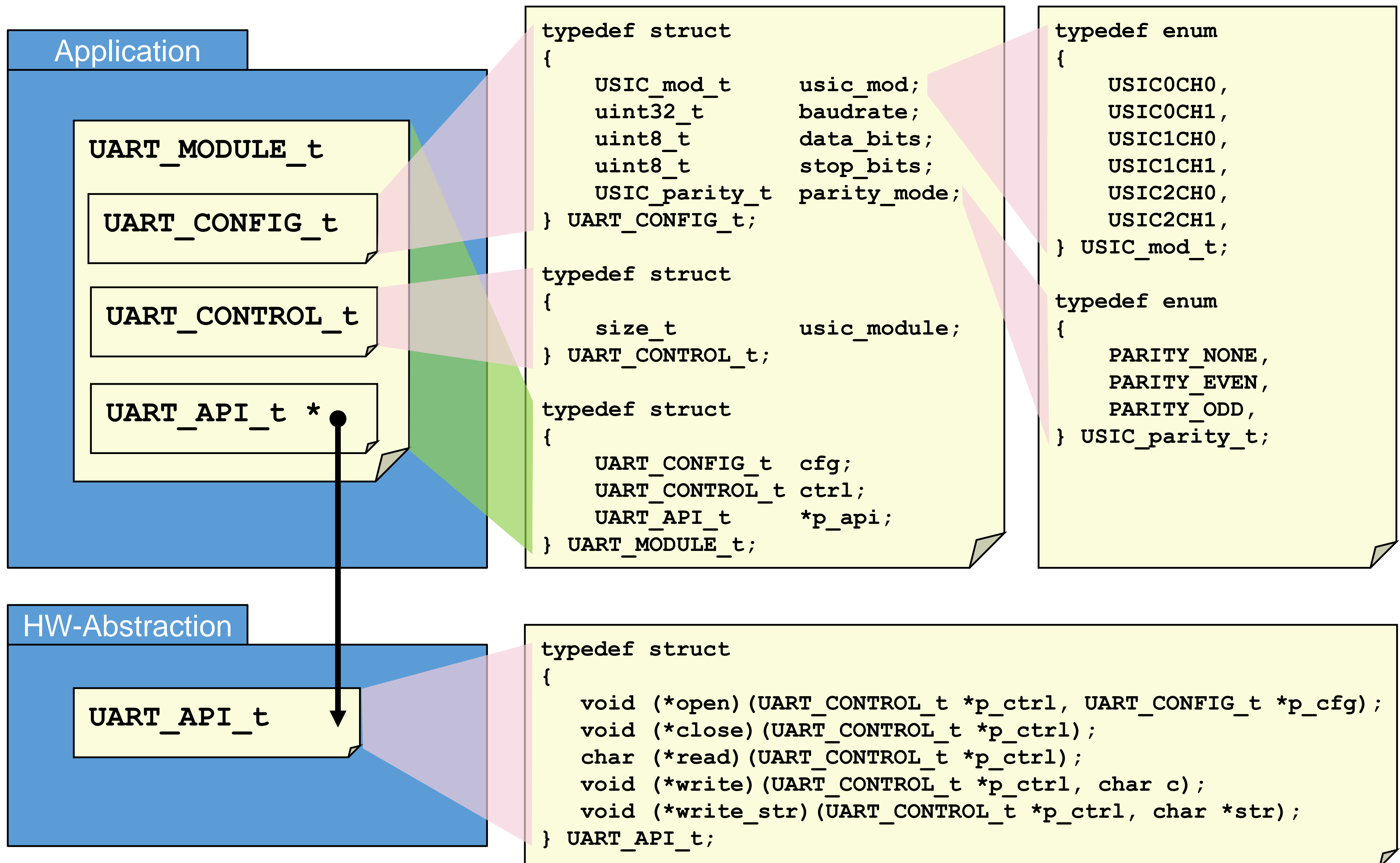
- HAL Driver stellt generisches Interface bereit,
  das unabhängig vom Device Driver ist
- Kapselung durch CONFIG, CONTROL und API Struktur

- Das Device Driver Modul wird durch die Init Struktur parametrisiert
  z.B.
  - Schnittstelle USIC 0 Channel 0
  - Baudrate 19600
  - …

Die Modul-Struktur enthält alles, was ein weiter oben liegendes Modul von diesem Modul wissen muss.



Example: UART-HAL Driver



Beispiel: UART-HAL Driver

main.c

1 2 3 4 5

```c
#include "uart.h"

int main(void)
{
    UART_MODULE_t module =
    {
        .cfg.usic_mod       = USIC0CH0,
        .cfg.baudrate       = 19200,
        .cfg.data_bits      = 8,
        .cfg.stop_bits      = 1,
        .cfg.parity_mode    = PARITY_NONE,
        .p_api              = &uart_api,
    };

    /*open UART with settings in config struct */
    module.p_api->open( &module.ctrl, &module.cfg);

    /*Write an A */
    module.p_api->write( &module.ctrl, 'A' );

    while(1){}
}
```


MicroConsult

Beispiel: UART-HAL Driver

uart.c

1 2 3 4 5

```
#include "uart.h"
#include "USIC.h"
//global API structure initialisation
UART_API_t uart_api =
{
    .open      = UART_open,
    .close     = UART_close,
    .read      = UART_read,
    .write     = UART_write,
    .write_str = UART_write_str,
};

void UART_write(UART_CONTROL_t *p_ctrl, char c)
{   XMC_UART_CH_Transmit((XMC_USIC_CH_t *) p_ctrl->usic_module , c);
}

void UART_write_str(UART_CONTROL_t *p_ctrl, char *str)
{
    while( *str != '\0' )
    {   UART_write(p_ctrl, *str);
        str++;
    }
}
   …
```


MicroConsult

Beispiel: UART-HAL Driver

uart.c

1 2 3 4 5

```
...
void UART_open(UART_CONTROL_t *p_ctrl, UART_CONFIG_t *p_cfg)
{
    USIC_MODULE_t module =
    {
        //.usic_module       = XMC_UART0_CH0, //part of UART_CONFIG_t
        .usic_ch_input       = XMC_USIC_CH_INPUT_DX0,
        .usic_ch_inp_source = 1U,
        .usic_ch_fifo_size  = XMC_USIC_CH_FIFO_SIZE_16WORDS,
        .uart_config         =
        {
            //.baudrate      = 19200,    //part of UART_CONFIG_t
            //.data_bits     = 8,        //part of UART_CONFIG_t
            .frame_length    = 8,
            //.stop_bits     = 1,        //part of UART_CONFIG_t
            .oversampling    = 16
            //.parity_mode   = XMC_USIC_CH_PARITY_MODE_NONE
        }
    };

    ...
```


MicroConsult

Beispiel: UART-HAL Driver

uart.c

1 2 3 4 5

```
…
//fill USIC module with content of p_cfg
//translation of UART enums into USIC enums avoids dependencies
switch( p_cfg->usic_mod )
{   case USIC0CH0:
        module.usic_module = XMC_UART0_CH0;
        break;
    case USIC0CH1:
        module.usic_module = XMC_UART0_CH1;
        break;
    case USIC1CH0:
        module.usic_module = XMC_UART1_CH0;
        break;
    case USIC1CH1:
        module.usic_module = XMC_UART1_CH1;
        break;
    case USIC2CH0:
        module.usic_module = XMC_UART2_CH0;
        break;
    case USIC2CH1:
        module.usic_module = XMC_UART2_CH1;
        break;
}
…
```

Beispiel: UART-HAL Driver

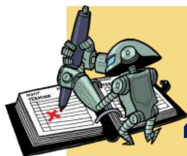
uart.c

1 2 3 4 5

```
    …
    //continue to fill USIC module with content of p_cfg
    module.uart_config.baudrate  = p_cfg->baudrate;
    module.uart_config.data_bits = p_cfg->data_bits;
    module.uart_config.stop_bits = p_cfg->stop_bits;
    switch( p_cfg->parity_mode )
    {
        case PARITY_NONE:
            module.uart_config.parity_mode = XMC_USIC_CH_PARITY_MODE_NONE;
            break;
        case PARITY_EVEN:
            module.uart_config.parity_mode = XMC_USIC_CH_PARITY_MODE_EVEN;
            break;
        case PARITY_ODD:
            module.uart_config.parity_mode = XMC_USIC_CH_PARITY_MODE_ODD;
            break;
    }

    //open USIC channel with filled module structure
    USIC_open( &module );

    //set control structure content for use in other functions
    p_ctrl->usic_module = (size_t) module.usic_module;
}
```


MicroConsult

- Die Embedded-Welt
- Klassische LLD-Programmierung
- Software-Schichten-Modelle
- CMSIS, MCAL und Co



Zusammenfassung

Zusammenfassung

- **Die Embedded-Welt**
  - Spezielle Produkte, die sowohl von der Hardwarearchitektur als auch vom Softwaredesign speziell an ihre Aufgaben angepasst werden.
- **Klassische Low-Level-Treiberprogrammierung**
  - Detailliertes Hardware-Knowhow ist erforderlich, um die Peripheriemodule abhängig von der Aufgabenstellung richtig und vollständig nutzen zu können.

- **Software-Schichten-Modelle**
  - Durch die saubere Trennung der verschiedenen Softwarekomponenten wird die Wiederverwendbarkeit wesentlich erleichtert.

- **CMSIS, MCAL und Co**
  - Das Knowhow der Entwickler verlagert sich von detaillierten Kenntnissen der Peripheriemodule hin zu Software Interfaces und abstraktem Hardware-Knowhow.

https://www.microconsult.de

Experience Embedded

Professionelle Schulungen, Beratung und Projektunterstützung

Suche Go

KONTAKT | DE | EN

TRAINING & BERATUNG | DIENSTLEISTUNGEN | FACHWISSEN | UNTERNEHMEN | EVENTS | BLOG

EXPERIENCE EMBEDDED

MicroConsult ist Ihr Partner für Embedded Systems Engineering - professionelle Schulungen, Beratung und Projektunterstützung.

Vom Mikrocontroller bis zum Systemdesign, Sie profitieren von unserer jahrzehntelangen Erfahrung. Maßgeschneiderter Knowhow-Transfer für Ihre Projektziele – nah am Menschen, nah an der Praxis.

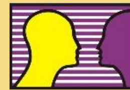
Embedded-Software: Analyse, Design, Architektur
Die wichtigsten Erfolgsfaktoren für Ihre Embedded-Projekte
Training & Coaching | Fachwissen

Echtzeit: Embedded-Programmierung, Betriebssysteme
Expertenwissen für Ihre Software-Implementierung
Training & Coaching | Fachwissen

Mikrocontroller: Multicore, Singlecore, Peripherie
Das richtige Bausteinwissen führt Sie in kürzester Zeit zu Ihrer Applikation
Training & Coaching | Fachwissen

Test, Qualität und Safety von Embedded-Software
Mit modernen Methoden erreichen Sie Ihre vorgegebenen Qualitätsziele
Training & Coaching | Fachwissen

Renate Schultes
MicroConsult GmbH
Trainer & Coach
Mikrocontroller Hardware & Software



r.schultes@microconsult.de
www.microconsult.de