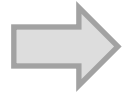




Microcontroller Programming (2)

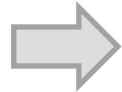
Gerald Kupris, 15.10.2013

Lectures Microcontroller Programming WS2013/14



- 08.10.2013 Microcontroller, Programming and Debugging Interfaces
- 15.10.2013 Reading and Writing of Registers
- 22.10.2013 I/O-Pins, Reading and Writing of Single Bits
- 29.10.2013 Clock Generation, CPU und Computing Power
- 05.11.2013 Interrupts
- 12.11.2013 No lecture !**
- 19.11.2013 Memory
- 26.11.2013 Timer and PWM, Watchdog Timer
- 03.12.2013 Analog to Digital Converter
- 10.12.2013 Serial Interfaces: SPI, IIC and UART
- 17.12.2013 Additional Explanation of the Freescale Cup Cars
- 14.01.2014 Project Work on the Freescale Cup Cars
- 21.01.2014 Project Work on the Freescale Cup Cars

Hands-On Workshops Microcontroller Programming



08.10.2013 Workshop 1: Preparation of the Work Place

15.10.2013 Workshop 2: Loading and Debugging of Programs

22.10.2013 Workshop 3: Using the GPIO Pins

29.10.2013 Workshop 4: Clock Generation and Calculations

05.11.2013 Workshop 5: Interrupts

12.11.2013 No Workshop !

19.11.2013 Workshop 6: Using the Flash Memory

26.11.2013 Workshop 7: Timer and Pulse Width Modulation (PWM)

03.12.2013 Workshop 8: Analog to Digital Conversion

10.12.2013 Workshop 9: Serial Communication

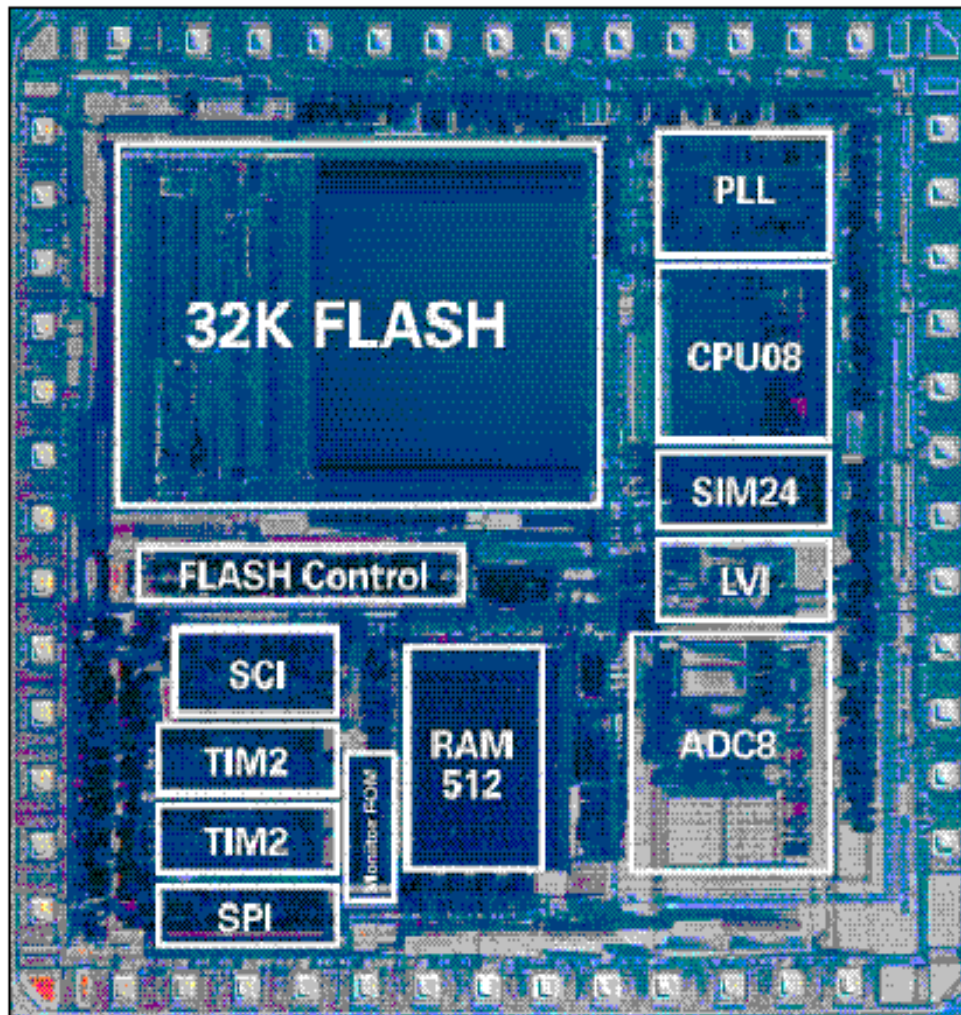
17.12.2013 Project work on the Freescale Cup Cars

14.01.2014 Project work on the Freescale Cup Cars

21.01.2014 Project work on the Freescale Cup Cars

Participation on all workshops is required for admittance to the final project!

MC68HC908GP32 Microcontroller (MCU)



Register

In computer architecture, a register is a small amount of storage.

Registers are normally measured by the number of bits they can hold, for example, an "8-bit register" or a "32-bit register".

Registers can be used to:

- hold data (data registers)
- configure something (configuration register)
- display the status of something (status register)

Registers can have the access type „Read“ or „Write“ or both of them. More sophisticated access types are: „Read only“ or „Write once“ ...

Register Set of a processor

A **processor register** is available as part of a CPU or other digital processor. Such registers are (typically) addressed by mechanisms other than main memory and can be accessed more quickly. Almost all computers, load-store architecture or not, load data from a larger memory into registers where it is used for arithmetic, manipulated, or tested, by some machine instruction.

A processor often contains several kinds of registers, that can be classified accordingly to their content or instructions that operate on them.

The number of registers available on a processor and the operations that can be performed using those registers has a significant impact on the efficiency of code generated by optimizing compilers.

The multitude of all registers are called the **Register Set** of a processor.

Different Types of Processor Registers

Data registers can hold numeric values such as integer and floating-point values, as well as characters, small bit arrays and other data.

Address registers hold addresses and are used by instructions that indirectly access primary memory.

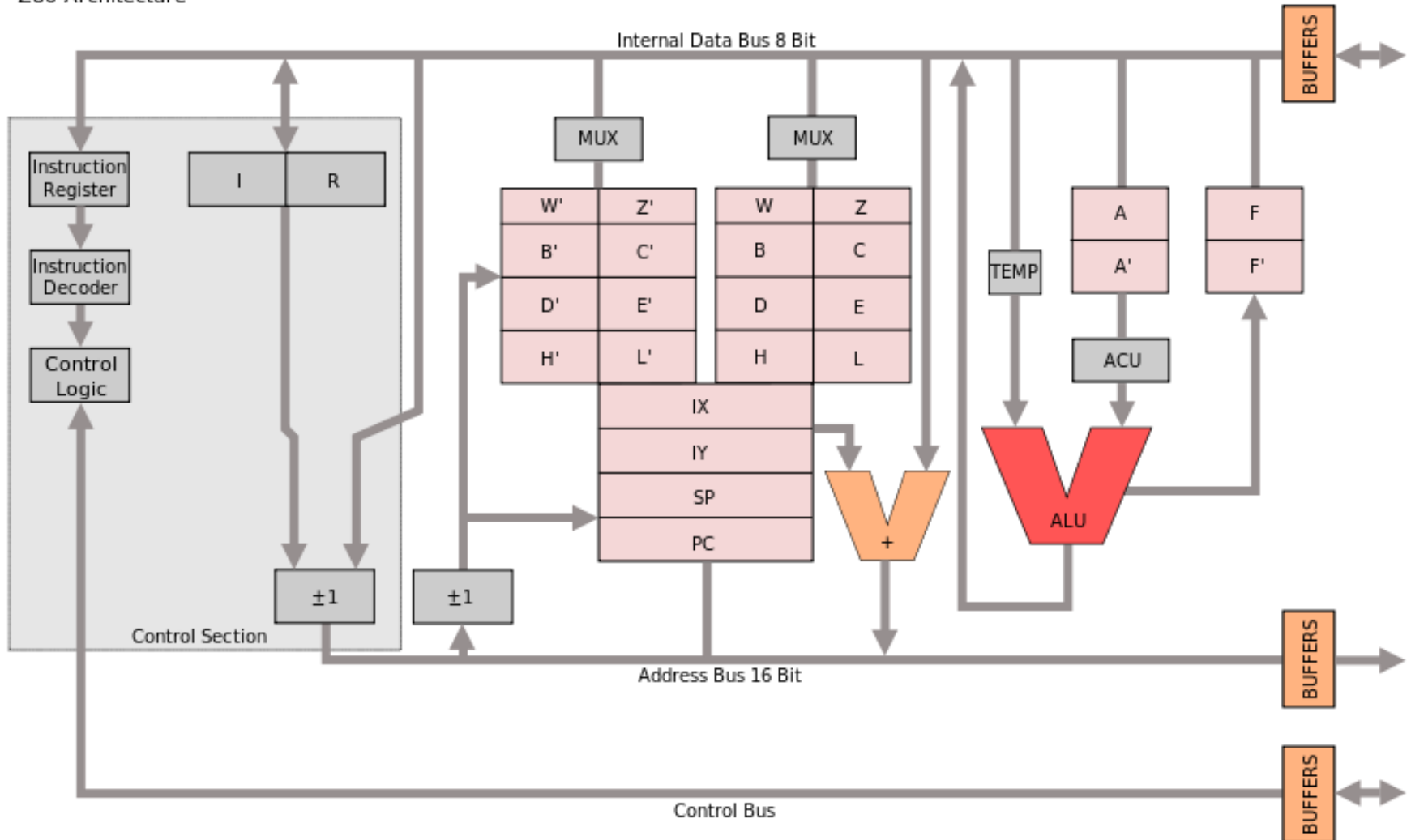
Conditional registers hold truth values often used to determine whether some instruction should or should not be executed.

General purpose registers (GPRs) can store both data and addresses, i.e., they are combined Data/Address registers.

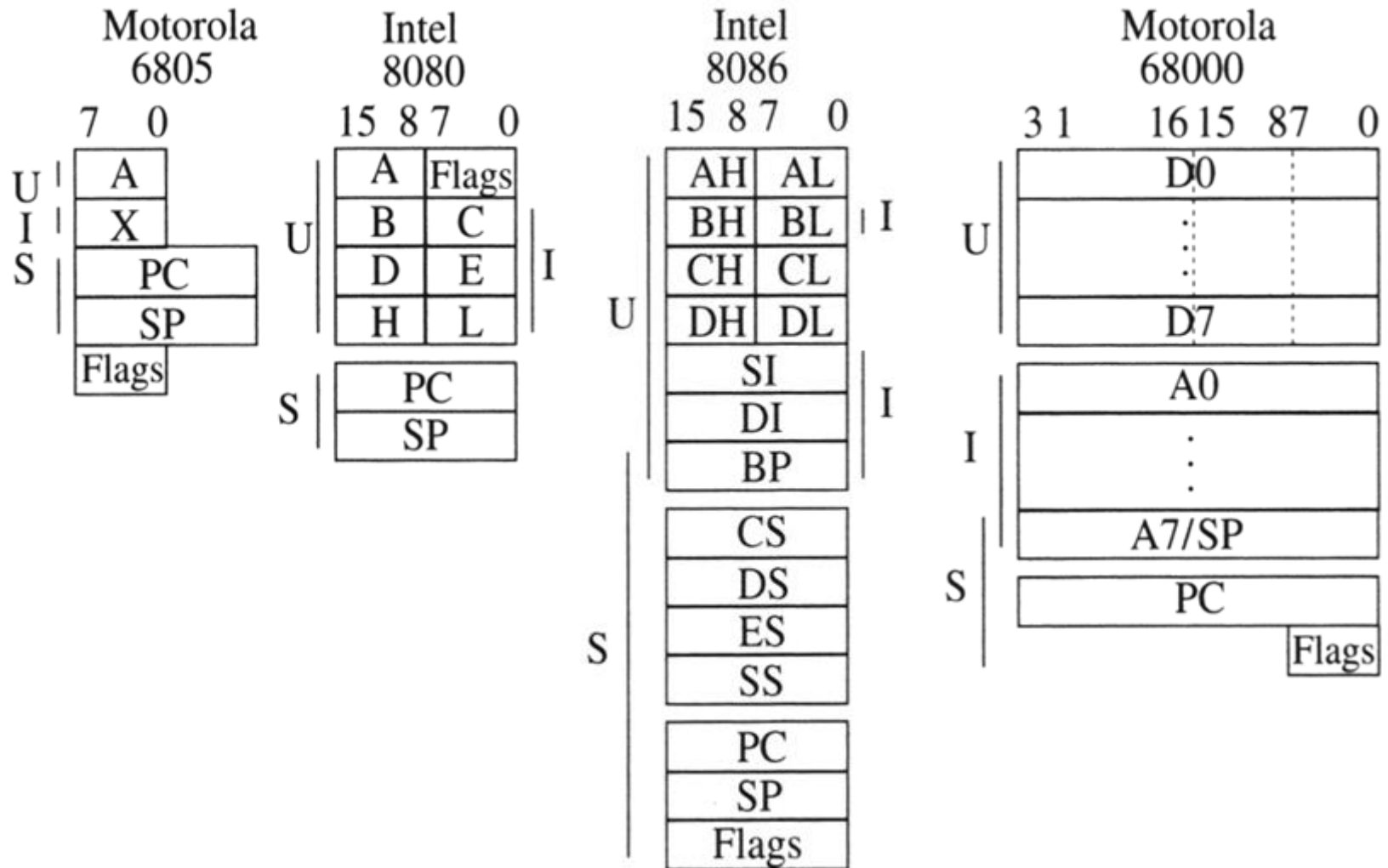
Special purpose registers (SPRs) hold program state; they usually include the **program counter** (instruction pointer) and **status register** (processor status word).

Architecture of the Z80 microprocessor

Z80 Architecture



Register Sets of historical Processors



Register Set of the ARM Cortex M3 Processor

Name	Functions (and Banked Registers)
R0	General-Purpose Register
R1	General-Purpose Register
R2	General-Purpose Register
R3	General-Purpose Register
R4	General-Purpose Register
R5	General-Purpose Register
R6	General-Purpose Register
R7	General-Purpose Register
R8	General-Purpose Register
R9	General-Purpose Register
R10	General-Purpose Register
R11	General-Purpose Register
R12	General-Purpose Register
R13 (MSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)
R13 (PSP)	
R14	Link Register (LR)
R15	Program Counter (PC)
xPSR	Program Status Registers
PRIMASK	
FAULTMASK	
BASEPRI	
CONTROL	

Low Registers

High Registers

Special Registers

Interrupt Mask Registers

Control Register

Peripheral Registers

A **peripheral register** is part of the peripheral block outside of the CPU.

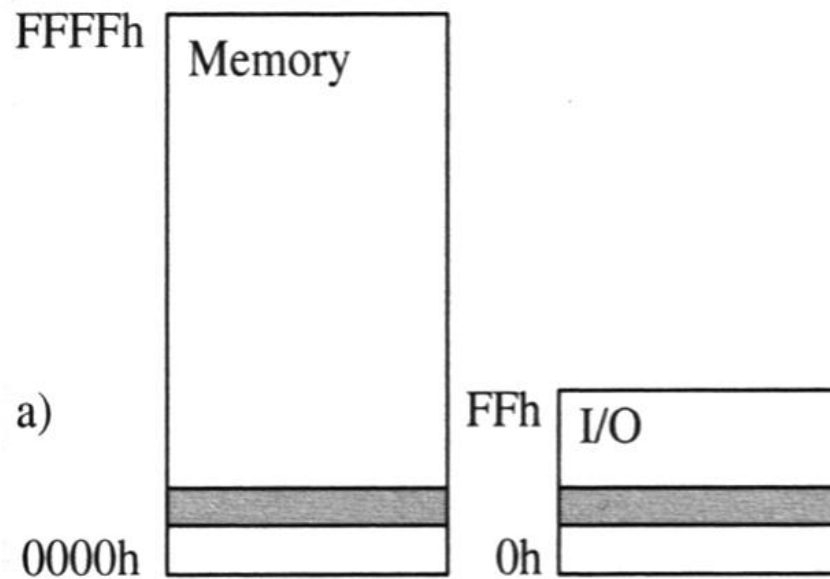
For example: serial interface, timer, A/D converter ...

The peripheral registers are used to:

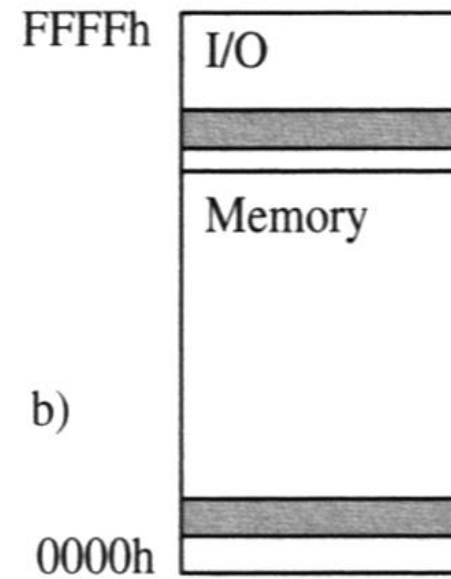
- configure the peripheral itself,
- read the status of the peripheral,
- exchange data with the peripheral.

Peripheral registers are often **memory mapped**, so the programmer can access these registers with C language instructions.

Address Spaces



a) Separated address spaces



b) Memory mapped I/O

Address Space of ARM Cortex-M3/M4 in general

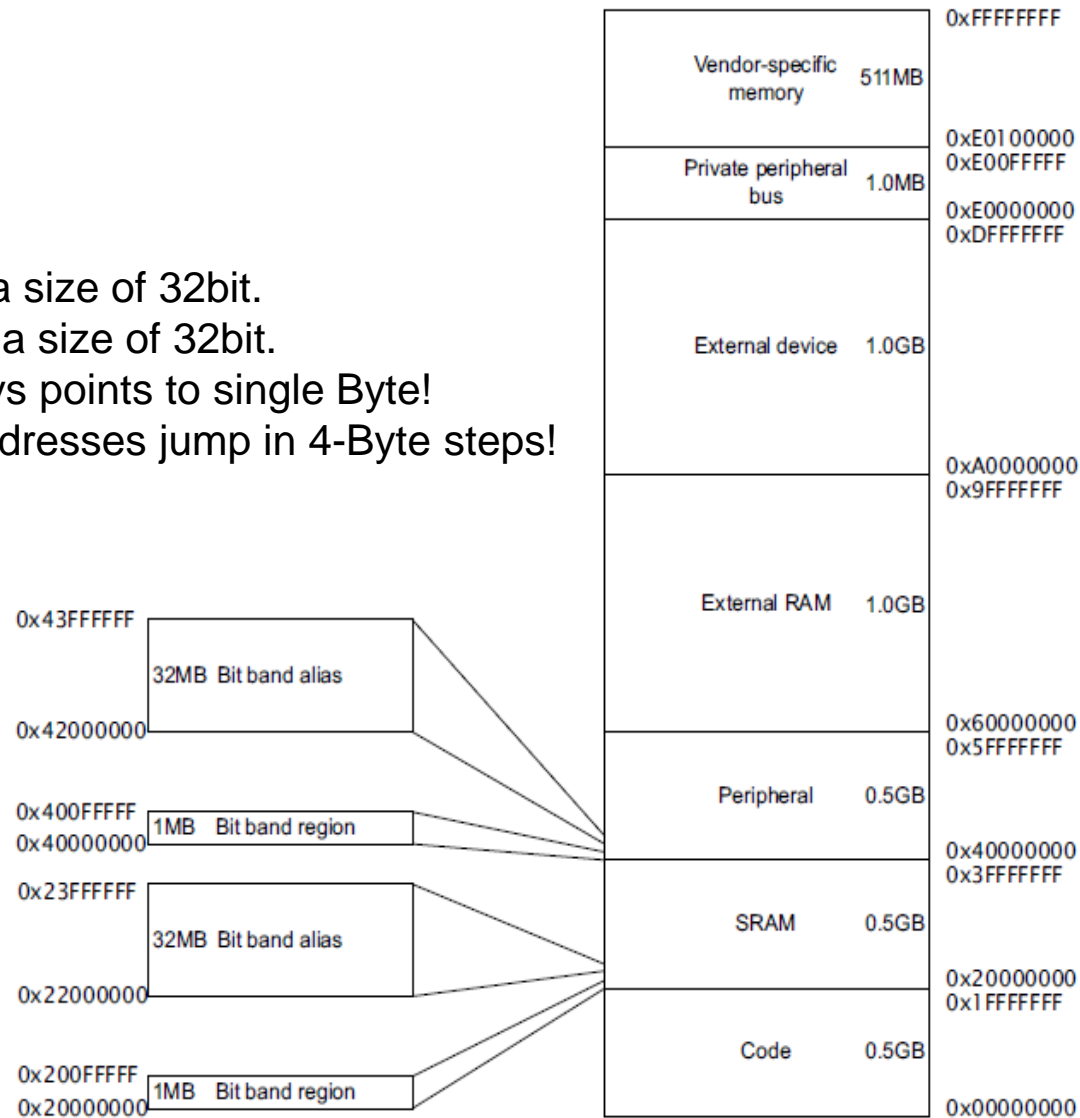
Please note:

Addresses have a size of 32bit.

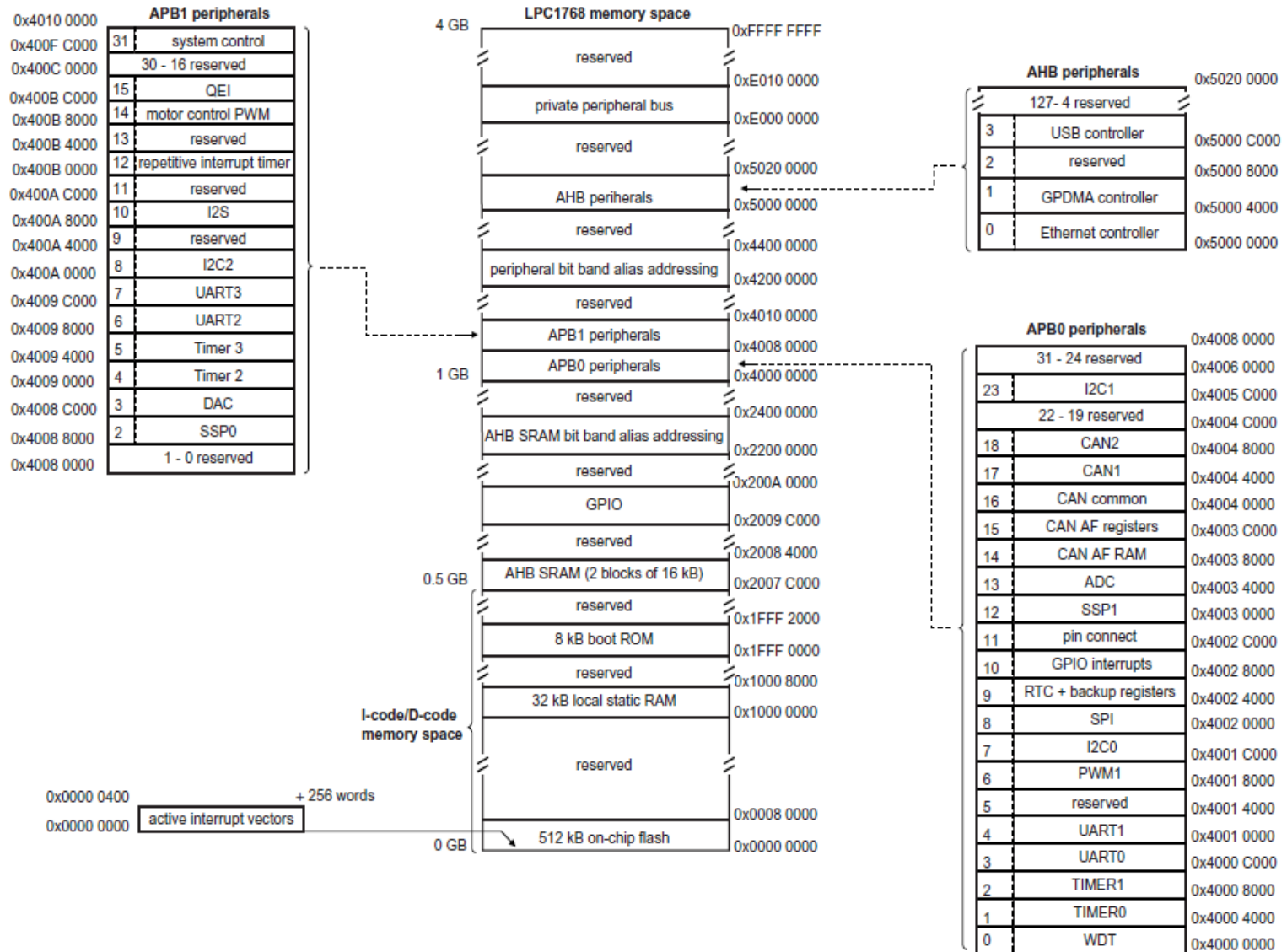
Data words have a size of 32bit.

An address always points to single Byte!

Therefore, the addresses jump in 4-Byte steps!



Address Space of the LPC17xx (Cortex M3, NXP)



Address Space of the Kinetis K60 (1)

System 32-bit Address Range	Destination Slave	Access
0x0000_0000–0x0FFF_FFFF	Program flash and read-only data (Includes exception vectors in first 1024 bytes)	All masters
0x1000_0000–0x13FF_FFFF	<ul style="list-style-type: none"> • For MK60DN256ZVLQ10: Reserved • For MK60DX256ZVLQ10: FlexNVM • For MK60DN512ZVLQ10: Reserved • For MK60DN256ZVMD10: Reserved • For MK60DX256ZVMD10: FlexNVM • For MK60DN512ZVMD10: Reserved 	All masters
0x1400_0000–0x17FF_FFFF	For devices with FlexNVM: FlexRAM For devices with program flash only: Programming acceleration RAM	All masters
0x1800_0000–0x1FFF_FFFF	SRAM_L: Lower SRAM (ICODE/DCODE)	All masters
0x2000_0000–0x200F_FFFF	SRAM_U: Upper SRAM bitband region	All masters
0x2010_0000–0x21FF_FFFF	Reserved	–
0x2200_0000–0x23FF_FFFF	Aliased to SRAM_U bitband	Cortex-M4 core only
0x2400_0000–0x3FFF_FFFF	Reserved	–

Address Space of the Kinetis K60 (2)

System 32-bit Address Range	Destination Slave	Access
0x4000_0000–0x4007_FFFF	Bitband region for peripheral bridge 0 (AIPS-Lite0)	Cortex-M4 core & DMA/EzPort
0x4008_0000–0x400F_EFFF	Bitband region for peripheral bridge 1 (AIPS-Lite1)	Cortex-M4 core & DMA/EzPort
0x400F_F000–0x400F_FFFF	Bitband region for general purpose input/output (GPIO)	Cortex-M4 core & DMA/EzPort
0x4010_0000–0x41FF_FFFF	Reserved	–
0x4200_0000–0x43FF_FFFF	Aliased to peripheral bridge (AIPS-Lite) and general purpose input/output (GPIO) bitband	Cortex-M4 core only
0x4400_0000–0x5FFF_FFFF	Reserved	–
0x6000_0000–0x7FFF_FFFF	FlexBus (External Memory - Write-back)	All masters
0x8000_0000–0x9FFF_FFFF	FlexBus (External Memory - Write-through)	All masters
0xA000_0000–0xDFFF_FFFF	FlexBus (External Peripheral - Not executable)	All masters
0xE000_0000–0xE00F_FFFF	Private peripherals	Cortex-M4 core only
0xE010_0000–0xFFFF_FFFF	Reserved	–

Register Manipulation

Memory-mapped I/O is easy to use in C

Pointer dereferences are quick

- Any address in the memory space can be addressed via a pointer and dereferenced
- The compiler sets the address at compile-time!

You need only figure out how to set the correct address manually

- The details are processor / compiler-specific

Important Documents

K60 Sub-Family Reference Manual (K60P144M100SF2RM)
1800 pages!

Kinetis Peripheral Module Quick Reference (KQRUG)
with software examples KINETIS512_SC.zip

K60 Sub-Family Reference Manual

Supports: MK60DN256ZVLQ10, MK60DX256ZVLQ10,
MK60DN512ZVLQ10, MK60DN256ZVMD10, MK60DX256ZVMD10,
MK60DN512ZVMD10



Document Number: K60P144M100SF2RM
Rev. 6, Nov 2011

Please note:

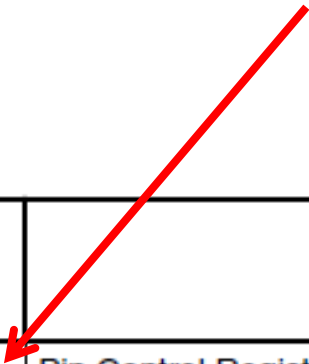
Addresses have a size of 32bit.

Data words have a size of 32bit.

An address always points to single Byte!

Therefore, the addresses jump in 4-Byte steps!

PORT memory map



Absolute address (hex)	Register name	Width (in bits)	Access	Reset value
4004_9000	Pin Control Register n (PORTA_PCR0)	32	R/W	0000_0000h
4004_9004	Pin Control Register n (PORTA_PCR1)	32	R/W	0000_0000h
4004_9008	Pin Control Register n (PORTA_PCR2)	32	R/W	0000_0000h

Port A in the K60 Sub-Family Reference Manual

Table 10-35. GPIO Signal Descriptions

Chip signal name	Module signal name	Description	I/O
PTA[31:0] ¹	PORTA[31:0]	General purpose input/output	I/O

PORT memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
4004_9028	Pin Control Register n (PORTA_PCR10)	32	R/W	0000_0000h	11.4.1/264

Addresses: 4004_9000h base + 0h offset + (4d × n), where n = 0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0							IS	0				IRQC				UK	0				MUX			0	DSE	ODE	PFE	0	SPE	PE	PS
W								w1c																		DSE	ODE	PFE		SPE	PE	PS
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Pin Control Register PCR 10

11.4.1 Pin Control Register n (PORTx_PCRn)

For PCR1 to PCR5 of the port A, bit 0, 1, 6, 8, 9, 10 reset to 1; for the PCR0 of the port A, bit 1, 6, 8, 9, 10 reset to 1; in other conditions, all bits reset to 0.

Addresses: 4004_9000h base + 0h offset + (4d × n), where n = 0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0							$\frac{1}{0}$	0				IRQC				$\frac{1}{0}$	0				MUX			0	DSE	ODE	PFE	0	SRE	PE	PS
W								w1c																		DSE	ODE	PFE		SRE	PE	PS
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
PORTA_PCR10=(0|PORT_PCR_MUX(1));
```

```
#define PORT_PCR_MUX_MASK                0x700u
#define PORT_PCR_MUX_SHIFT                8
#define PORT_PCR_MUX(x)
((uint32_t)((uint32_t)(x))<<PORT_PCR_MUX_SHIFT)&PORT_PCR_MUX_MASK)
```

0x700u = 11100000000

Header File MK60DZ10.h

The header file **MK60DZ10.h** reflects the specific features of the concrete microcontroller and is included in our c code.

#include „MK60DZ10.h“

#include <MK60DZ10.h>

#include with angel brackets **< >** searches in the specified standard include search path. Angle brackets are often used to include libraries.

#include with quotation marks **„ “** searches in the normal program source path.

Analysis of **MK60DZ10.h**

Start of the file main.c

used for printf outputs

definition of the concrete microcontroller

```
/*  
 * main implementation: use this 'C' sample to create your own  
 */  
#include <stdio.h>  
#include "derivative.h" /* include peripheral declarations */  
  
#define GPIO_PIN_MASK          0x1Fu  
#define GPIO_PIN(x)            (((1)<<(x & GPIO_PIN_MASK)))  
  
...
```

next week more information about this!!

File derivative.h

```
/*  
 * Note: This file is recreated by the project wizard whenever  
 *       the MCU is changed and should not be edited by hand  
 */  
  
/* Include the derivative-specific header file */  
#include <MK60DZ10.h>
```



information about the microcontroller used

File MK60DZ10.h

```
/*
** #####
** Processors:           MK60DN512ZVLL10
**                       MK60DX256ZVLL10
**                       MK60DN256ZVLL10
**                       MK60DN512ZVLQ10
**
** . . .
**
** Compilers:           Freescale C/C++ for Embedded ARM
**                       GNU ARM C Compiler
**                       IAR ANSI C/C++ Compiler for ARM
**
** Reference manual:    K60P144M100SF2RM, Rev. 5, 8 May 2011
** Version:             rev. 1.1, 2011-06-15
**
** . . .
```

Look at the file!

Example: in the blink program in main.c

```
// Set PTA10, PTA11, PTA28, and PTA29 (connected to LED's) for GPIO
PORTA_PCR10=(0|PORT_PCR_MUX(1));
...
```

```
#define PORT_PCR_REG(base,index) ((base)->PCR[index])
```

```
#define PORTA_PCR10
```

```
PORT_PCR_REG(PORTA_BASE_PTR,10)
```

```
#define PORTA_BASE_PTR ((PORT_MemMapPtr)0x40049000u)
```

0x4004_9000	73	Port A multiplexing control
-------------	----	-----------------------------

Bit fields

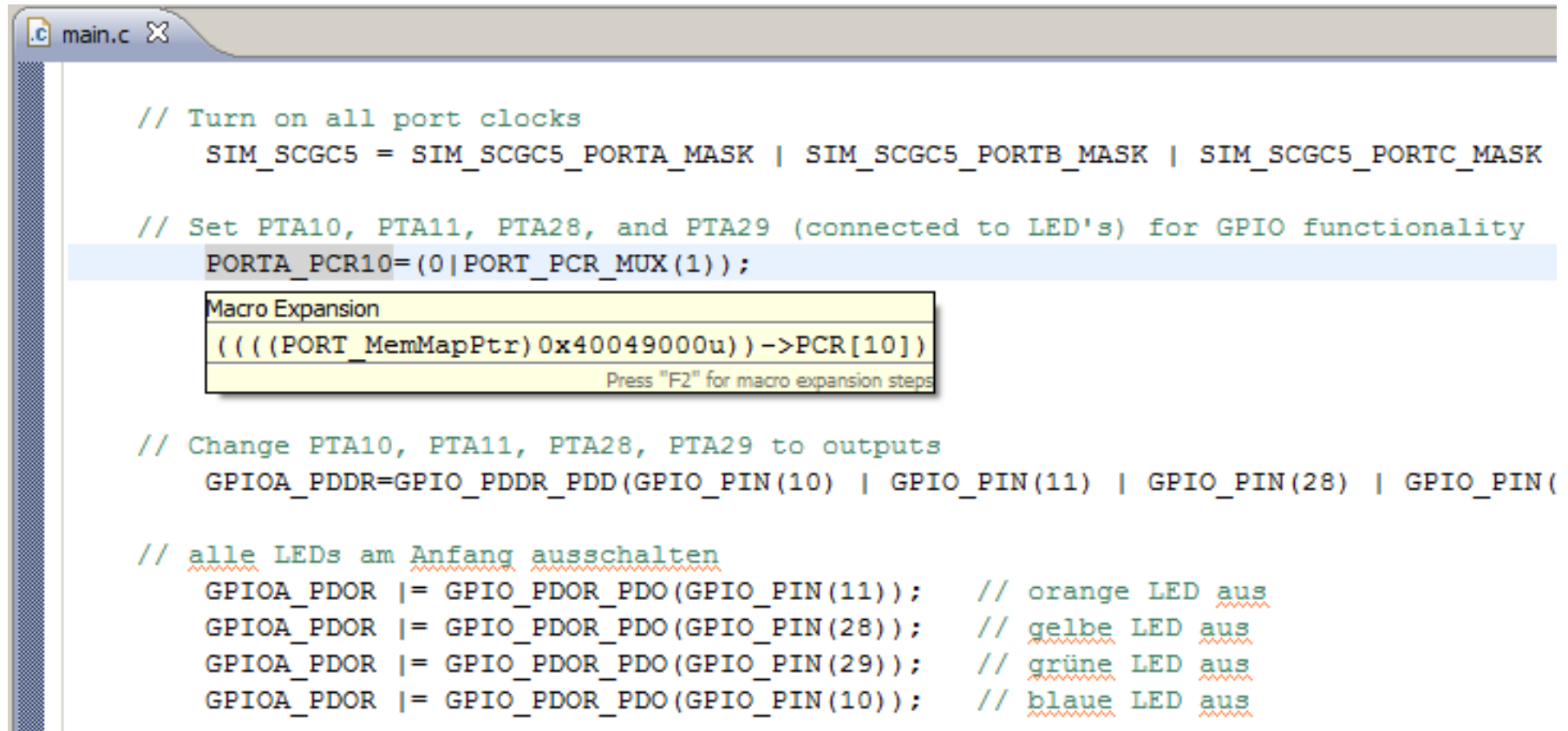
A bit field is set up with a structure declaration that labels each field and determines its width.

A bit field is used in computer programming to store multiple, logical, neighboring bits, where each of the sets of bits, and single bits can be addressed. A bit field is most commonly used to represent integral types of known, fixed bit-width. A well-known usage of bit-fields is to represent a set of bits, and/or series of bits, known as flags. For example, the first bit in a bit field can be used to determine the state of a particular attribute associated with the bit field.

Implementation in MK60DZ10.h

```
/** PORT - Peripheral register structure */
typedef struct PORT_MemMap {
    uint32_t PCR[32];           **< Pin Control Register n, array of
    uint32_t GPCLR;             **< Global Pin Control Low Register,
    uint32_t GPCHR;             **< Global Pin Control High Register,
    uint8_t RESERVED_0[24];
    uint32_t ISFR;              /**< Interrupt Status Flag Register,
    uint8_t RESERVED_1[28];
    uint32_t DFER;              /**< Digital Filter Enable Register,
    uint32_t DFCR;              /**< Digital Filter Clock Register, of
    uint32_t DFWR;              /**< Digital Filter Width Register, of
} volatile *PORT_MemMapPtr;
```

Port A PCR 10 in the Editor window



```

main.c X
// Turn on all port clocks
SIM_SCGC5 = SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTC_MASK

// Set PTA10, PTA11, PTA28, and PTA29 (connected to LED's) for GPIO functionality
PORTA_PCR10=(0|PORT_PCR_MUX(1));

// Change PTA10, PTA11, PTA28, PTA29 to outputs
GPIOA_PDDR=GPIO_PDDR_PDD(GPIO_PIN(10) | GPIO_PIN(11) | GPIO_PIN(28) | GPIO_PIN(

// alle LEDs am Anfang ausschalten
GPIOA_PDOR |= GPIO_PDOR_PDO(GPIO_PIN(11)); // orange LED aus
GPIOA_PDOR |= GPIO_PDOR_PDO(GPIO_PIN(28)); // gelbe LED aus
GPIOA_PDOR |= GPIO_PDOR_PDO(GPIO_PIN(29)); // grüne LED aus
GPIOA_PDOR |= GPIO_PDOR_PDO(GPIO_PIN(10)); // blaue LED aus
  
```

Macro Expansion
 (((PORT_MemMapPtr) 0x40049000u) ->PCR[10])
 Press "F2" for macro expansion steps

Port A PCR 10 in the Debug window (Register view)

The screenshot shows the Debug window with the Register view selected. The register list displays the following data:

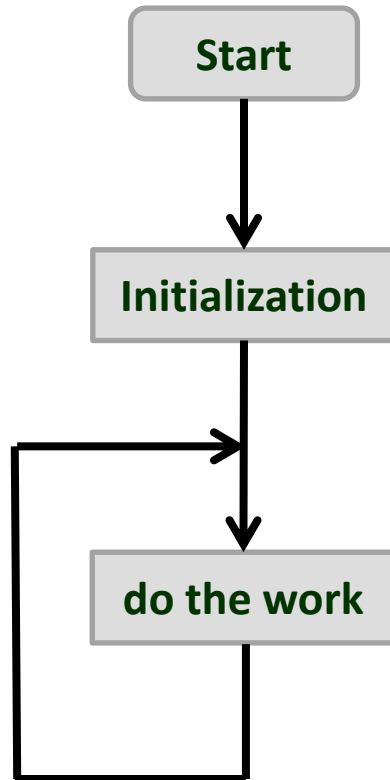
Name	Value	Location
PORTA_PCR8	0x0	0x40049020
PORTA_PCR9	0x0	0x40049024
PORTA_PCR10	0x100	0x40049028
PORTA_PCR11	0x100	0x4004902c
PORTA_PCR12	0x0	0x40049030
PORTA_PCR13	0x0	0x40049034

Below the register list, the Bit Fields section shows the bit pattern for the selected register (PORTA_PCR10):

00000000 0 0000 0000 0 0000 001 0 0 0 0 0 0 0

A tooltip for the bit field [10:8] [Pin Mux Control] is visible, indicating the function of the bits 10 through 8.

Typical flow of an embedded program



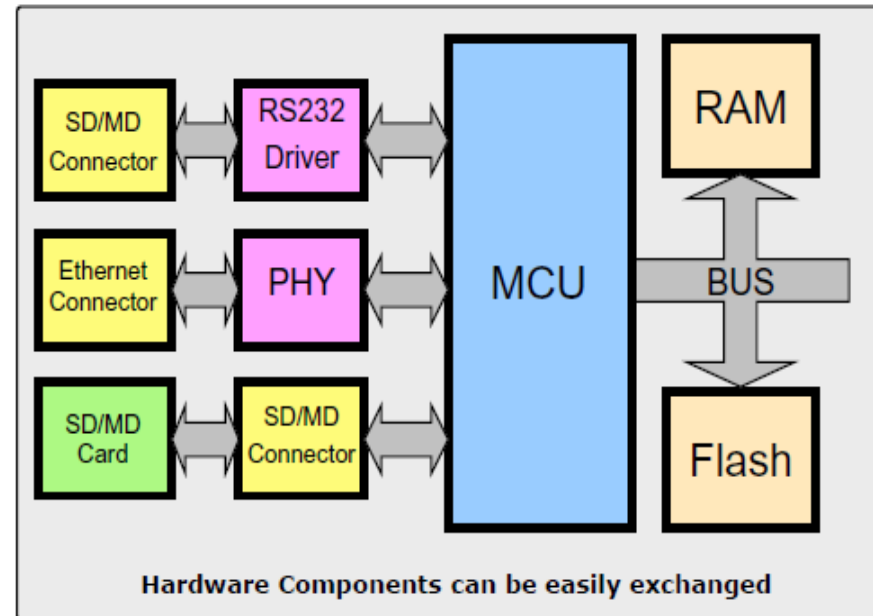
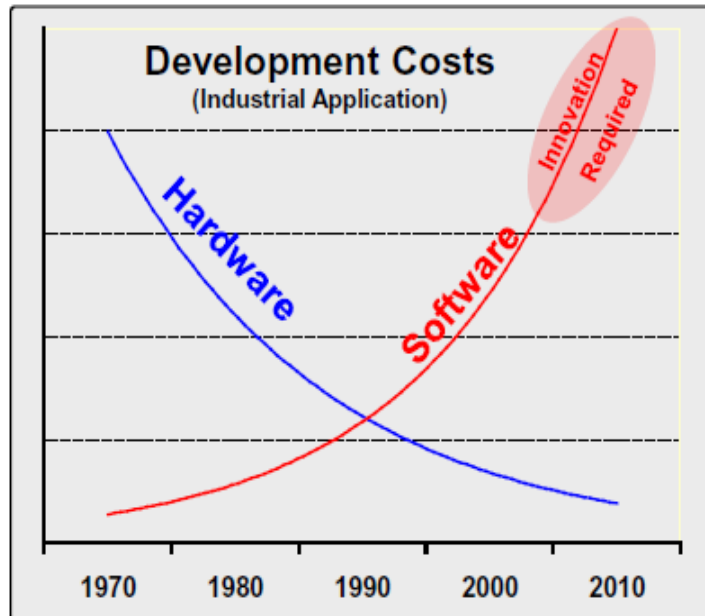
switch the system on

configuring the system by
writing to the necessary registers

```
while (1)
{
    . . . . .
}
```

„Superloop“

Software Complexity



- Well-known issues that drive software costs
 - Increasing product requirements that are implemented by software
 - Hardware problems tend to become compensated by software
- Up to now software components cannot be easily exchanged

A Microcontroller Software Interface Standard is Required!

CMSIS

Cortex Microcontroller Software Interface Standard

- ▶ **CMSIS supports all Cortex-Mx based micro controllers**
- ▶ **Will be expanded for future family members**
- ▶ **Defined in co-operation with silicon and software vendors**

- ▶ **Software layers**
 - **Core Peripheral Access Layer:**
 - contains name definitions, address definitions and helper functions to access core registers and peripherals.
 - It defines also an device independent interface for RTOS Kernels that includes debug channel definitions.
 - Available for Cortex-M0 and Cortex-M3.
 - **Middleware Access Layer:**
 - Provides common methods to access peripherals for the software industry.
 - The Middleware Access Layer is adapted by the Silicon Vendor for the device specific peripherals used by middleware components.
 - Middleware access layer is currently available for: Ethernet, UART, SPI

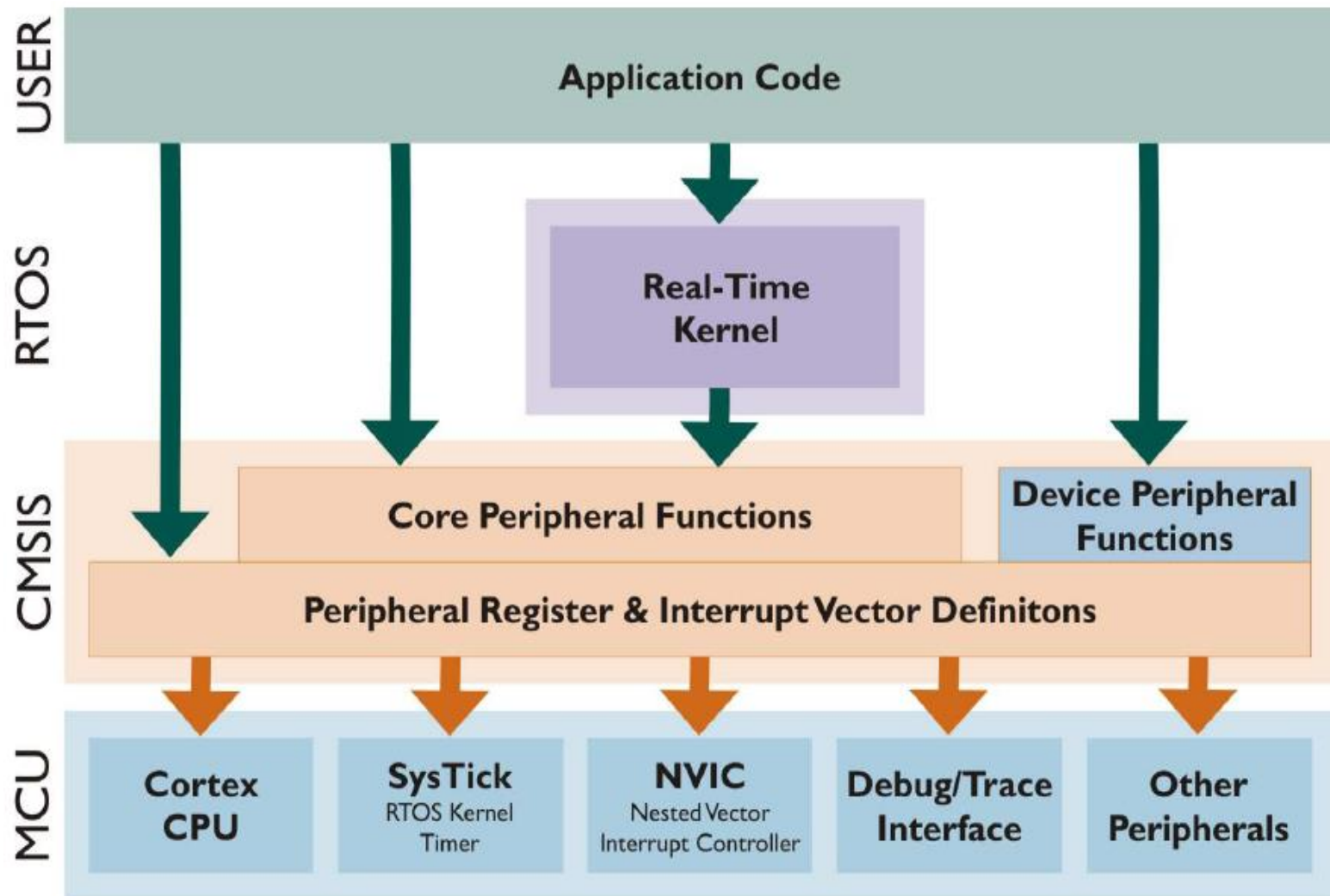
- ▶ **Expanded by Silicon partners with:**
 - **Device Peripheral Access Layer:** provides definitions for all device peripherals
 - **Access Functions for Peripherals (optional):** provides additional helper functions for peripherals

CMSIS Usage

- ▶ **CMSIS defines for a Cortex-Mx Microcontroller System:**
 - A common way to access peripheral registers and a common way to define exception vectors.
 - The register names of the Core Peripherals and the names of the Core Exception Vectors.
 - An device independent interface for RTOS Kernels including a debug channel.
 - Interfaces for middleware components (TCP/IP Stack, Flash File System).

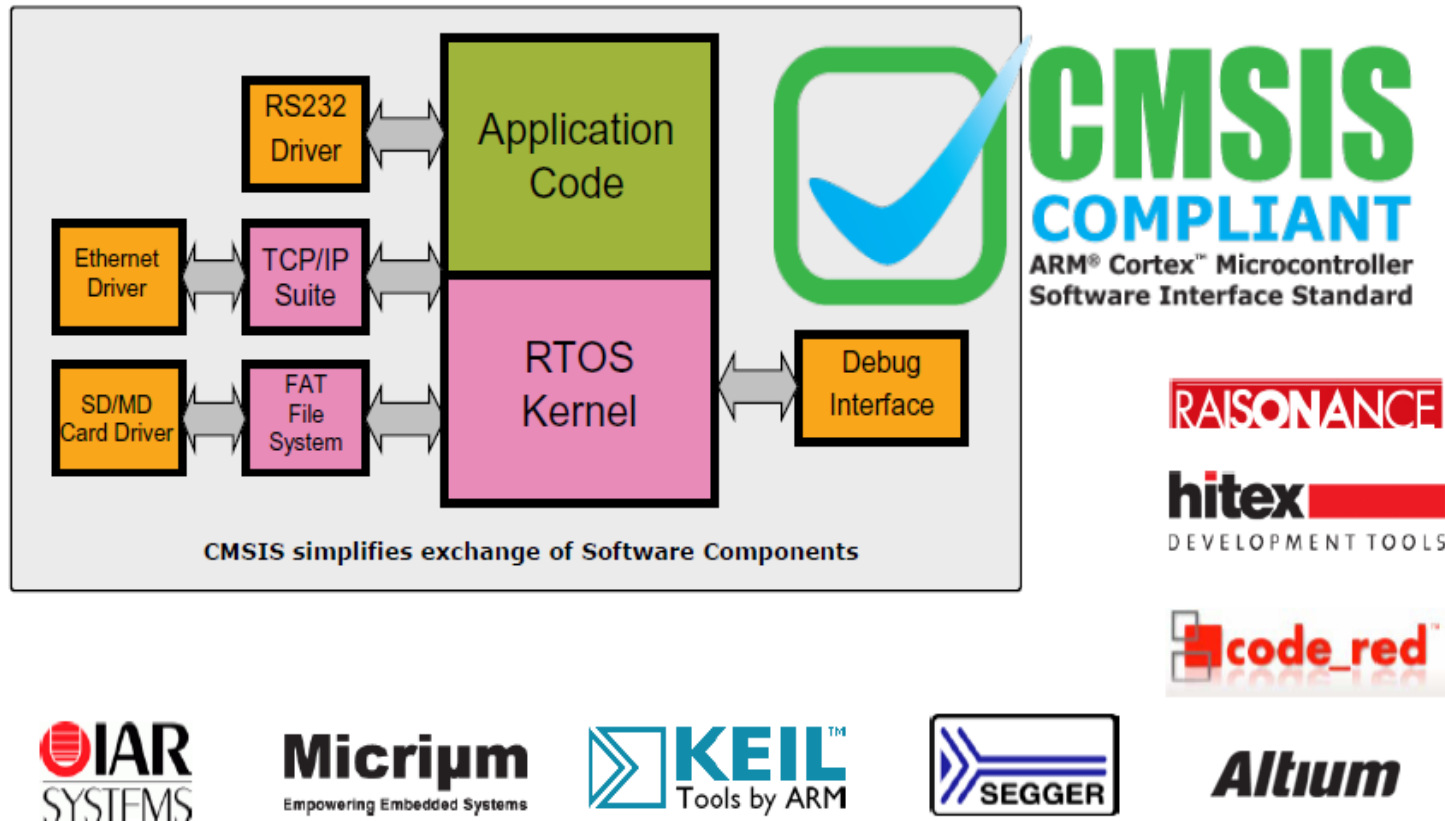
- ▶ **By using CMSIS compliant software components, the user can easier re-use template code. CMSIS is intended to enable the combination of software components from multiple middleware vendors.**

CMSIS Structure



CMSIS enables code Re-Use

<http://www.keil.com/dd/docs/arm/freescale/kinetis/mk60dz10.h>



The **Cortex Microcontroller Software Interface Standard (CMSIS)**
enables deployment of software components to physical Microcontroller Devices

Literature and Links

<http://www.arm.com/products/processors/cortex-m/cortex-m3.php>

<http://www.arm.com/products/processors/technologies/instruction-set-architectures.php>

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=K60

K60 Sub-Family Reference Manual

Kinetis Peripheral Module Quick Reference

Cortex-M4 Technical Reference Manual

<http://www.arm.com/products/processors/cortex-m/index.php>



Technische Hochschule Deggendorf – Edlmairstr. 6 und 8 – 94469 Deggendorf