Technische Hochschule **Rosenheim**
Technical University of Applied Sciences

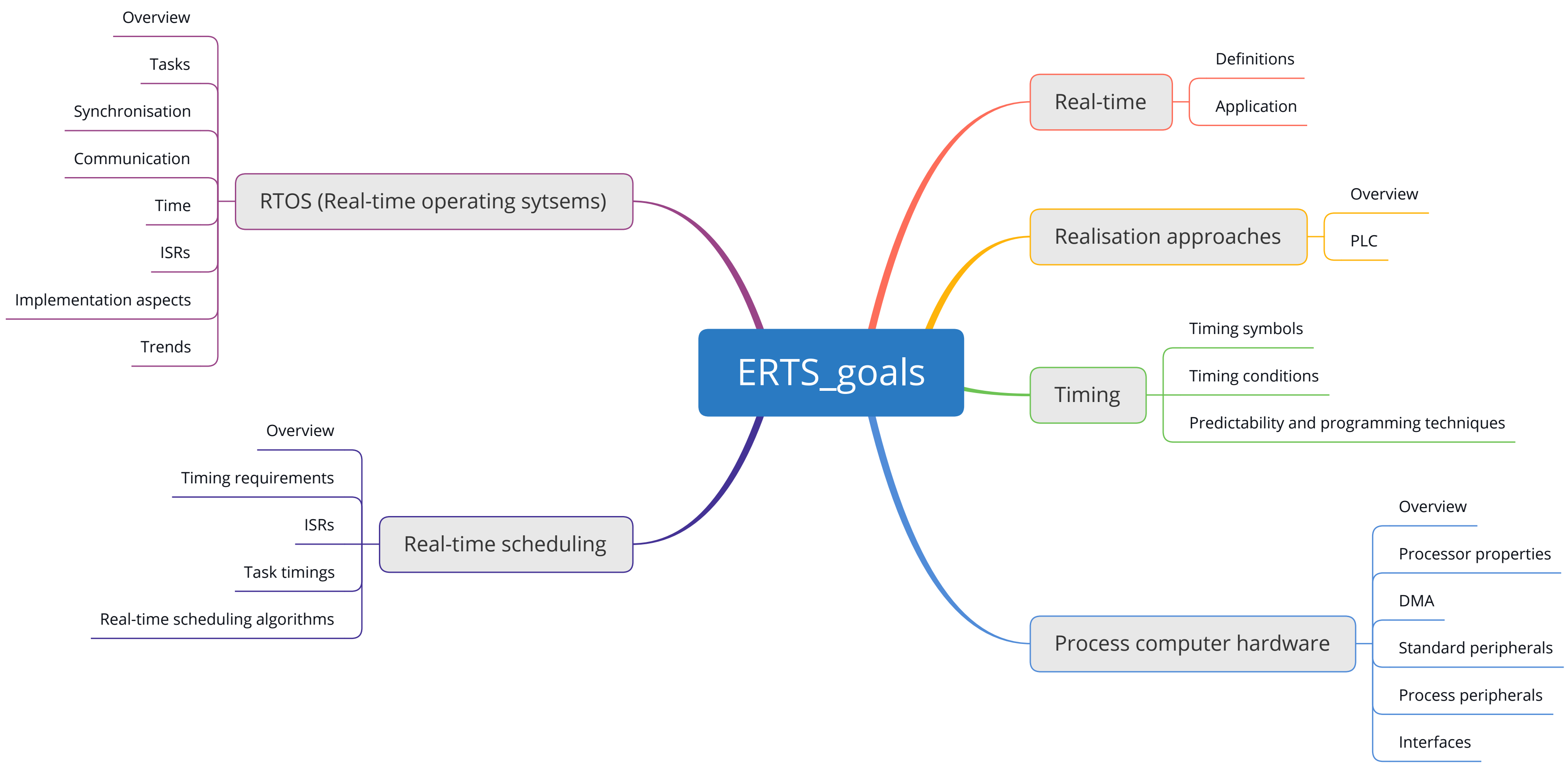
Prof. Dr. Florian Künzner
Technical University of Applied Sciences Rosenheim, Computer Science

ERTS - Embedded real-time systems

ERTS 4 – Process computer HW 1

CAMPUS Rosenheim
Computer Science

Goal



CAMPUS Rosenheim
Computer Science

Goal

ERTS::Process computer hardware 1

- Processor specifics
- Interrupts
- Interrupt interference

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Overview

Process computer hardware topics:

- Processor specifics
- Interrupts
- Standard peripherals: I/O
- Clocks
- DMA
- Process periphery

Important but not yet covered in this lecture:

- Field buses: special bus systems, TSN (time sensitive networking)

CAMPUS Rosenheim
Computer Science

Processor specifics

Architectural features of the hardware (processor + peripherals)

- Pipelining
- Caching
- MMU
- Interrupts
- Direct memory access (DMA)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Pipelining

Multiple instructions are in the pipeline

1. Fetch	2. Decode	3. Execute	time
CMD1			t_1
CMD2	CMD1		t_2
CMD3	CMD2	CMD1	t_3
CMD4	CMD3	CMD2	t_4

$\Rightarrow$ **speeds-up execution**

But: Pipeline flushes and refills for

- jumps/branches
- task changes
- interrupts

$\Rightarrow$ causes **unpredictable runtimes** (variance)

Caching

Processor caches (L1, L2, ...)

Key (real adr.)	Value (data: byte 0 to 3)			
	#0	#1	#2	#3
0x0...0100	0x12	0x34	?	?

⇒ **improves the overall performance**

But: Cache misses occurs (flushes and refills) for

- jumps/branches
- task changes
- interrupts

⇒ causes **unpredictable runtimes** (variance)

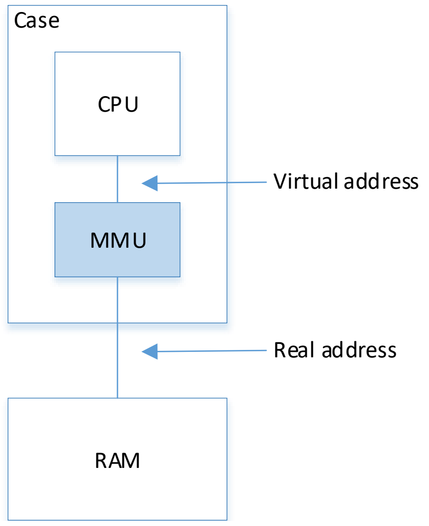
- This can increase the WCET by a factor of 33% [2, Buttazzo, p. 15]
- In a thesis: 10 μs measurement accuracy required, but cache introduced 20 μs uncertainty

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

MMU

The MMU allows paging with all the known security features.



But:

- MMU has to be reprogrammed on process (task) changes
- This introduces some additional processor cycles

CAMPUS Rosenheim
Computer Science

First summary

Modern processors optimise the overall runtime, but not the WCET.

Conclusion:

- **Determining** the **runtime** very accurate is **extremely hard**
- **Adding** a reasonable **buffer** to the **WCET**, according to the processor mechanism
- In practice, $\boldsymbol{U_{\max}} \leq \mathbf{0.3}, \ldots, \mathbf{0.5}$ to have sufficient safety reserves.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Processor

But which processor should we then use?

e.g. ARM Cortex-R

Interrupts

Hardware interrupts occur asynchronously and are therefore unpredictable.

Used for:

- HW communication
- Controlling the standard peripherals
- Process I/O and alarms

Details:

- Different interrupt priorities
- Interrupts preempt application and kernel tasks

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt handling sequence

Typical interrupt handling sequence for a CPU:

1. CPU accepts interrupt
2. Save PC (program counter) and SR (status register); disable interrupts
3. Switch to supervisor mode, save registers, load new registers (typically $< 1\mu s$)
4. Load PC from interrupt vector table
5. If there is no HW for 3, then save registers per software (typically $< 2\mu s$)
6. Execute ISR (interrupt service routine)
7. If 5, then restore all register in software
8. IRET, restore registers, PC, and SR (return to previous task)

Discussion:

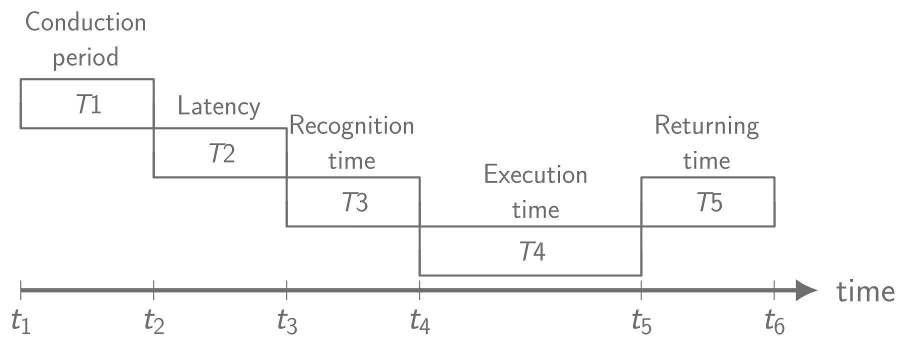
- Depending on the system, interrupts with a higher priority can interrupt ISRs with a lower priority.
- In all cases, ISRs should be short.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt handling timing symbols

Time characteristics according to DIN 66216* (page 2)



- $t_1 \rightarrow$ Interrupt signal arrives at periphery
- $t_2 \rightarrow$ Interrupt arrives CPU
- $t_3 \rightarrow$ Interruptible point in the CPU sequence; save PC, SR, disable interrupts
- $t_4 \rightarrow$ Start of ISR
- $t_5 \rightarrow$ End of ISR: IRET, restore PC, SR, enable interrupts
- $t_6 \rightarrow$ Continue previous task

- $T1 \rightarrow$ Conduction period
- $T2 \rightarrow$ Latency
- $T3 \rightarrow$ Recognition time
- $T4 \rightarrow$ Execution time
- $T5 \rightarrow$ Returning time
- $T_{IRE} \rightarrow$ Reaction time: $T_{IRE} = T1 + T2 + T3$
- $T_{IR} \rightarrow$ Response time: $T_{IR} = T1 + T2 + T3 + T4$
- $T_I \rightarrow$ Interrupt time: $T_I = T3 + T4 + T5$
- $T_{IO} \rightarrow$ Organisation time (overhead): $T_{IO} = T3 + T5$

*The DIN 66216 is outdated and withdrawn.

CAMPUS Rosenheim
Computer Science

Examples

The interrupt timing information are very hardware and process specific.

To get a first impression, we look at some examples*.

*The examples may be outdated, and may need some references.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example 1: Typical interrupt rates

- $T4$ $\rightarrow$ Execution time, depending on HW (clock rate): typically 10...20 instructions: $T4 = 0.5 \ldots 20\mu s$
- T_I $\rightarrow$ Interrupt time: $T_I = T3 + T4 + T5$ typically: $1\mu s < T_I < 220\mu s$
- T_{IO} $\rightarrow$ Organisation time (overhead): typically: $T_{IO} = T3 + T5 \Rightarrow T_{IO} = 0.5 \ldots 200\mu s$ MnP: NXP Coldfire M52259: $T_{IO} = 2400\text{ns}$
- R_{Imax} $\rightarrow$ Maximum interrupt rate: R_{Imax} typically: $4.5\text{kHz} < R_{Imax} < 1\text{MHz}$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example 2: Reaction time

T_{IRE} **depends essentially on the longest non-interruptible phase:**

- Longest hardware instruction
- Longest bus usage
- Longest software function with disabled interrupts

Through an OS kernel, $T2 + T3$ usually increases, because OS kernel routines are often (partially) non-interruptible.

Typical values for T_{IRE}:

- $T_{IRE} = 1\text{ms}$ $\rightarrow$ For bad HW + operating systems combinations
- $T_{IRE} = 0.1, \ldots, 20\mu\text{s}$ $\rightarrow$ For good HW + operating systems combinations
- $T_{IRE} = 1\text{s}$ $\rightarrow$ For non real-time operating systems (e.g. old standard unix system with exotic services)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example 3: Recognition and returning time

$T3$ and $T5$ depends strongly on the hardware:

- Which and how many registers are automatically saved/restored
- FPU enabled $\Rightarrow$ more registers to save/restore?

$T3$ example for: MnP: Motorolla 68000, 16 bit, 16 MHz:

- 44 cycles $\rightarrow$ 2.75 μs (without save registers)
  $$T3 = \frac{44}{16\ \text{MHz}} = \frac{44}{16 \times 10^6} = 2.75\mu\text{s}$$
- 84 cycles $\rightarrow$ 4.25 μs (save 7 registers to stack)
  84 cycles = 44 cycles + (12+4*7 cycles MOVEM)
  $$T3 = \frac{84}{16\ \text{MHz}} = \frac{84}{16 \times 10^6} = 4.25\mu\text{s}$$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example 4: $T1 + T2$

Measured value for $T1 + T2$ on a ColdFire with 80 MHZ:

- $T1 + T2 \rightarrow T1 + T2 = 400, \ldots 800\text{ns}$

Data sheet value for $T1 + T2$ on a NXP: QorIQ P2020, 1200 MHz, RTOS (Integrity, Green Hills)

- $T1 + T2 \rightarrow T1 + T2 = 0.144\mu\text{s}$

CAMPUS Rosenheim
Computer Science

Interrupt interference

Techniques to reduce interrupt interference

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt interference
Timing conditions with interrupts

If time conditions can't be met with interrupts (ISRs), the following methods are possible:

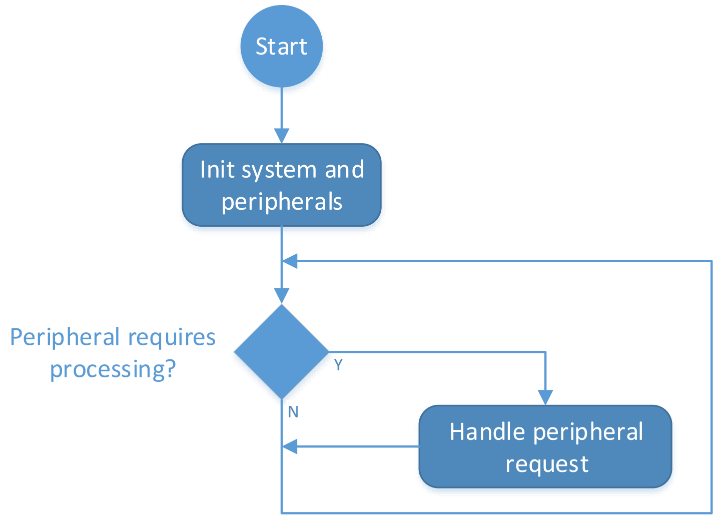
- Reduce interrupt interference:
  - Approach A: Programmed I/O: busy wait or polling
  - Approach B: Periodic hardware tasks
  - Approach C: Delayed interrupt handling (e.g. with RTOS tasks)
- DMA
- Additional or other hardware

CAMPUS Rosenheim
Computer Science

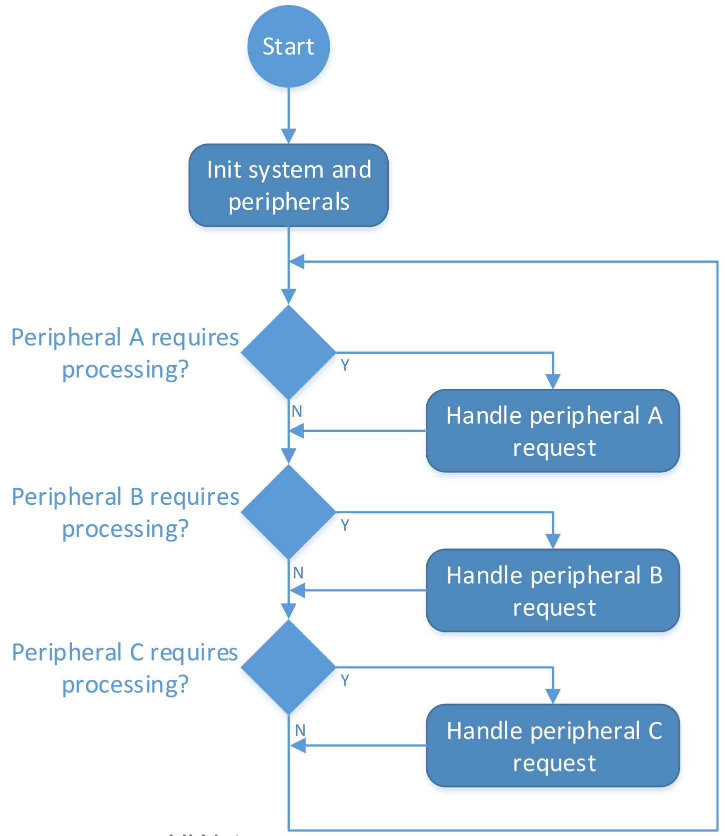
Interrupt interference: Approach A

Programmed I/O: busy wait or polling

One peripheral:


Start
Init system and peripherals
Peripheral requires processing?
Y
N
Handle peripheral request

Multiple peripherals:


Start
Init system and peripherals
Peripheral A requires processing?
Y
N
Handle peripheral A request
Peripheral B requires processing?
Y
N
Handle peripheral B request
Peripheral C requires processing?
Y
N
Handle peripheral C request

CAMPUS Rosenheim
Computer Science

Interrupt interference: Approach A

Programmed I/O: busy wait or polling

Disable all external interrupts

Details:

- Disable all external interrupts (except the timer or systick interrupt)
- Access the hardware directly and perform programmed I/O with busy wait or polling

Pro:

- Introduces a high predictability

Con:

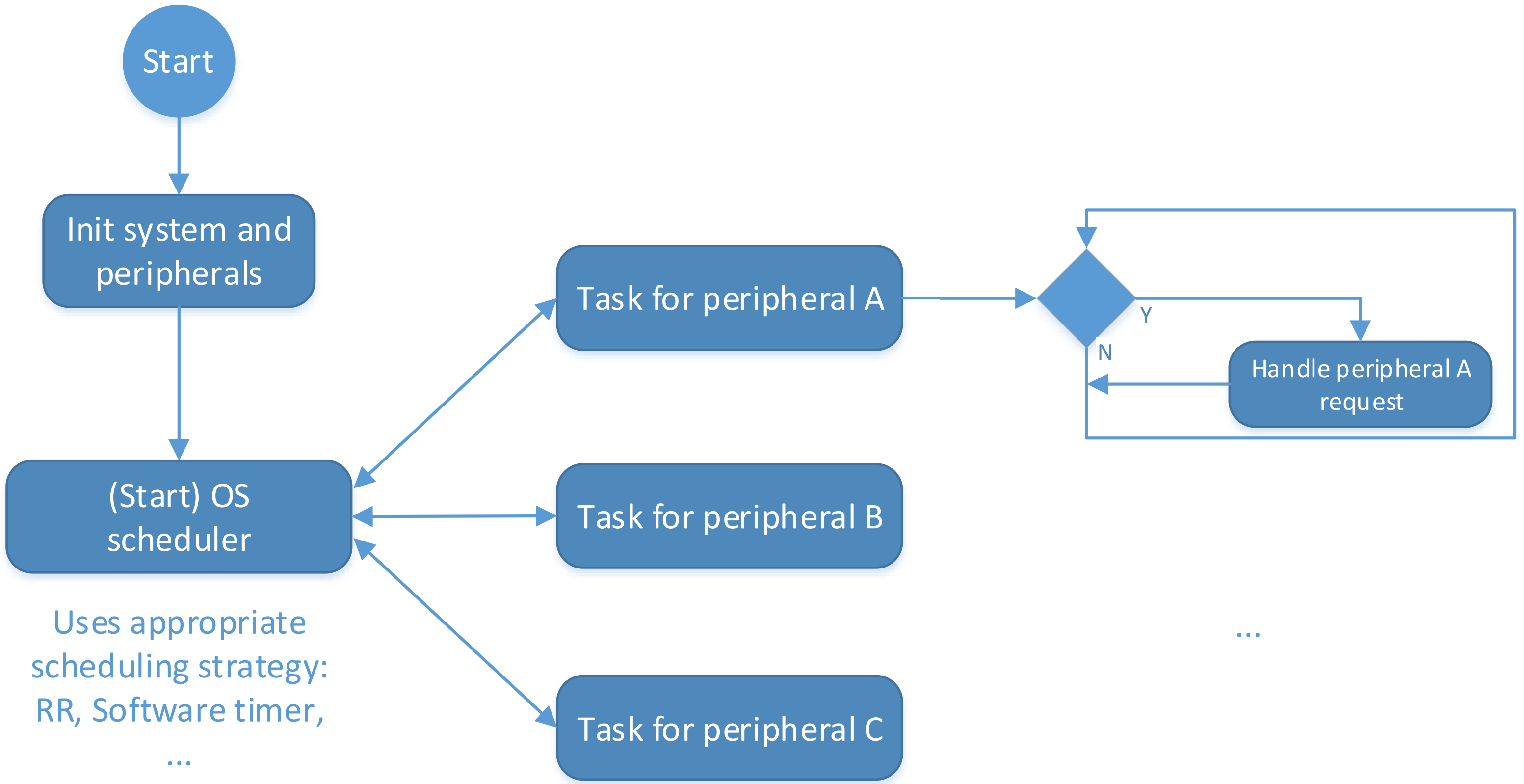
- Low processor efficiency (time spent for busy wait or polling)
- Only possible to poll a few (or one) devices

source: [2, Buttazzo, p. 16–17]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt interference: Approach B
Periodic hardware tasks



Uses appropriate scheduling strategy: RR, Software timer, ...

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt interference: Approach B

Periodic hardware tasks

Use periodic hardware tasks, while external interrupts are disabled

Details:

- Disable all external interrupts (except the timer or systick interrupt)
- Access the hardware with periodic tasks
  - Tasks can be part of the kernel
  - Different tasks for slow/fast devices

Pro:

- Hardware details are encapsulated within kernel tasks

Con:

- Low processor efficiency (time spent for busy wait or polling)

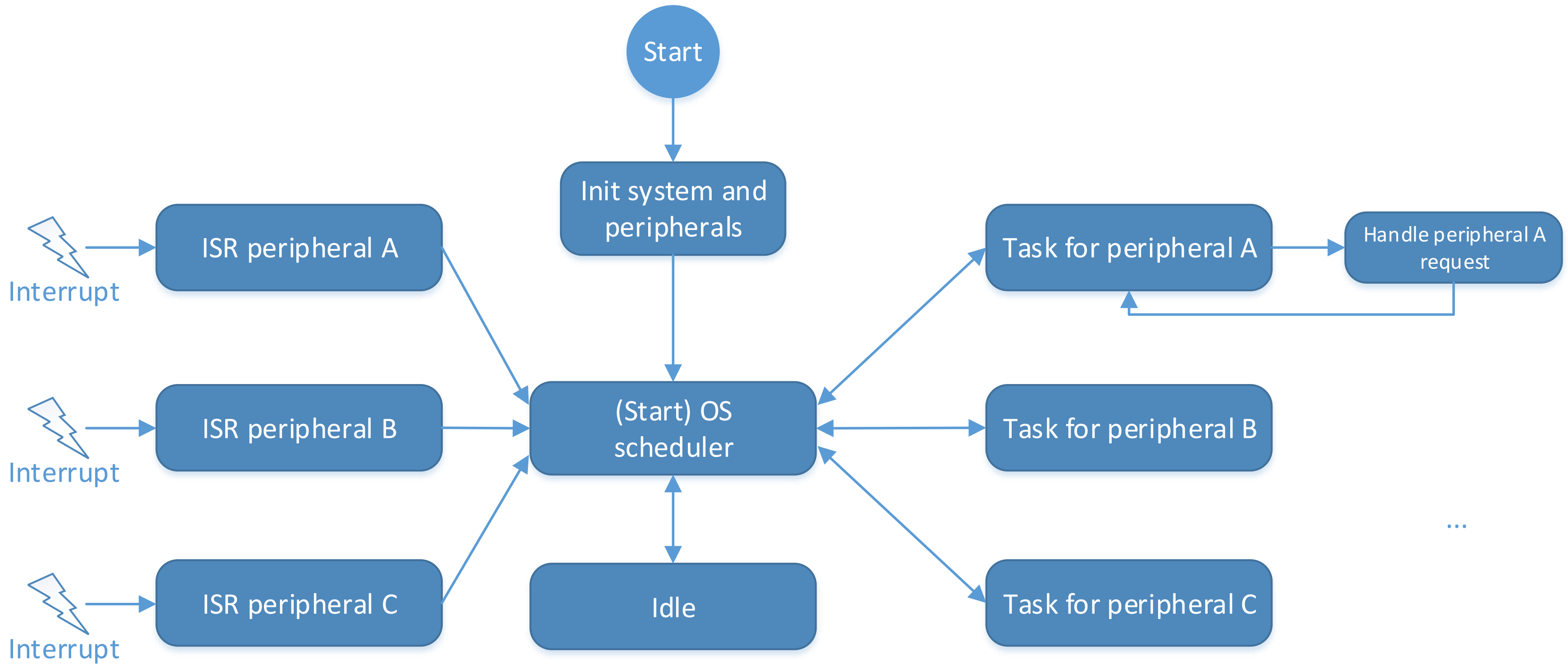
source: [2, Buttazzo, p. 17]

used in: [DRSK89] A. Damm, J. Reisinger, W. Schwabl, and H. Kopetz. The real-time oper- ating system of MARS. Operating System Review, 23(3):141–157, July 1989.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt interference: Approach C
Delayed interrupt handling (e.g. with RTOS tasks)



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt interference: Approach C

Delayed interrupt handling (e.g. with RTOS tasks)

Use ISR to handling task activation

Details:

- All external interrupts kept enabled
- The ISR is very short and only activates a handling task
- The handling task manages the device

Pro:

- Eliminates busy wait and polling overhead
- Unbounded delays introduced by driver tasks are drastically eliminated
- A control task can have a higher priority than a device handling task.

Con:

- Still contains some unbounded delays for the ISRs

source: [2, Buttazzo, p. 17–18]

used in: [TM89] H. Tokuda and C. W. Mercer. ARTS: A distributed real-time kernel. Operating System Review, 23(3), July 1989.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Exercise

Try to find as much timing ($T1 \ldots T5$) information as possible for the Cortex M4 architecture!

References:

- Cortex-M4 Technical Reference Manual: `https://developer.arm.com/documentation/ddi0439/b/`

CAMPUS Rosenheim
Computer Science

Exercise

Cortex M4 architecture: Interrupt handling timing

- $T1 \rightarrow$ Conduction period: $T1 =$
- $T2 \rightarrow$ Latency: $T2 =$
- $T3 \rightarrow$ Recognition time: $T3 =$
- $T4 \rightarrow$ Execution time: $T4 =$
- $T5 \rightarrow$ Returning time: $T5 =$
- $T_{IRE} \rightarrow$ Reaction time:
  $T_{IRE} = T1 + T2 + T3 =$
- $T_{IR} \rightarrow$ Response time:
  $T_{IR} = T1 + T2 + T3 + T4 =$
- $T_I \rightarrow$ Interrupt time: $T_I = T3 + T4 + T5 =$
- $T_{IO} \rightarrow$ Organisation time (overhead): $T_{IO} = T3 + T5 =$

CAMPUS Rosenheim
Computer Science

Summary and outlook

Summary

- Processor specifics
- Interrupts
- Interrupt interference

Outlook

- Digital and analog I/O
- Real-time clocks