Technische Hochschule Rosenheim
Technical University of Applied Sciences

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

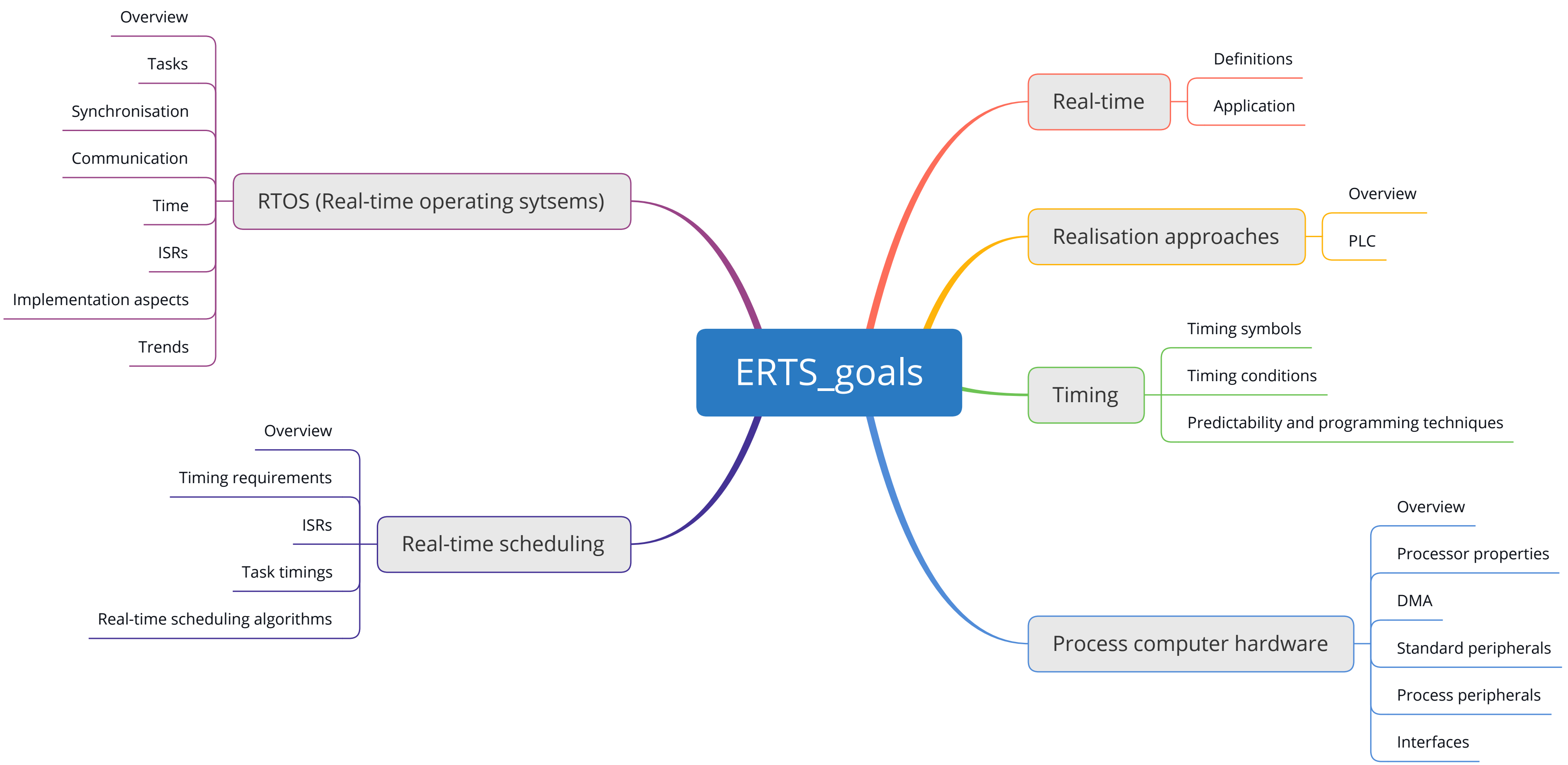
ERTS - Embedded real-time systems

ERTS 3 – Real-time basics

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal

ERTS::Real-time basics

- Timing symbols
- Timing conditions
- Predictability

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

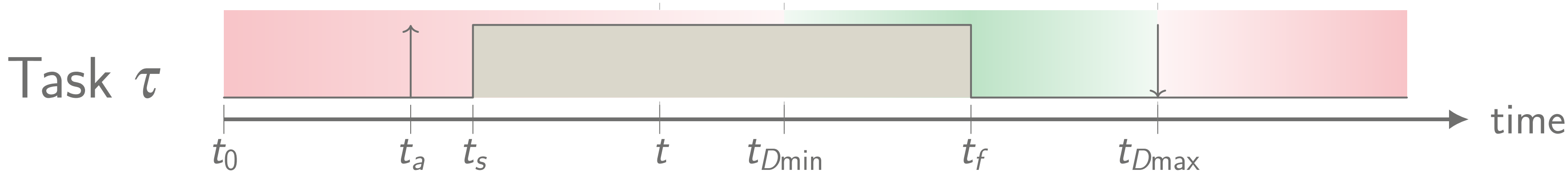
Timing symbols

- Real-time: extended timing basics
- Real-time requirements: period and rate
- CPU/μC core allocation
- Waiting and execution time
- Best and worst case execution times

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Real-time: extended timing basics

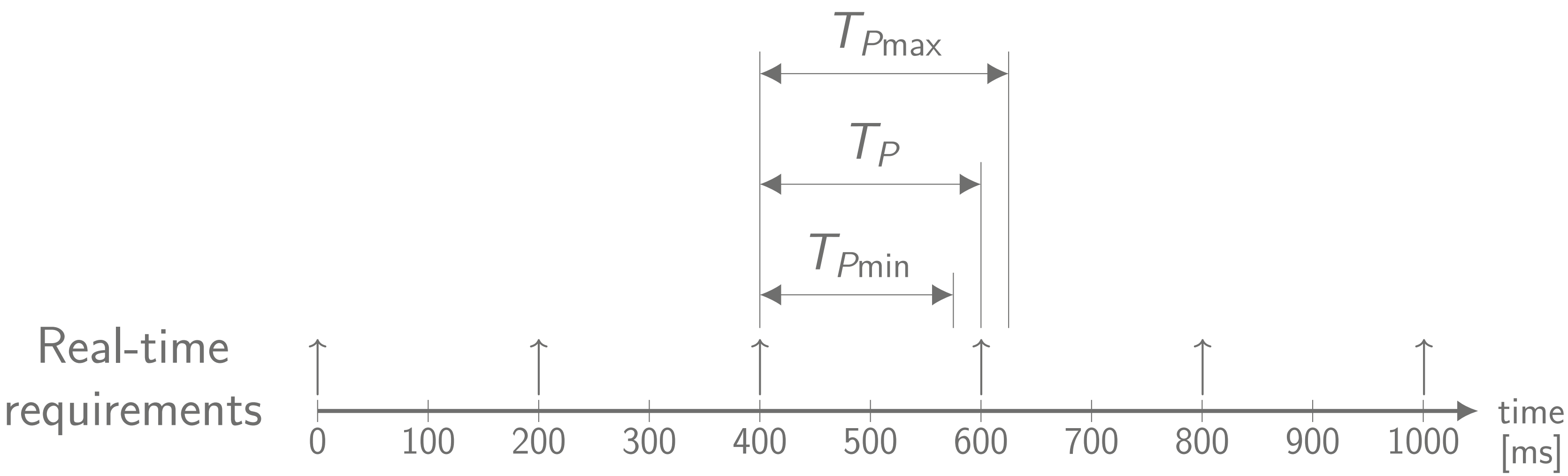


- t $\rightarrow$ Current time
- t_a $\rightarrow$ Arrival time of task τ: task becomes ready for execution
- t_s $\rightarrow$ Start time of task τ
- t_f $\rightarrow$ Finish time of task τ
- t_{Dmin} $\rightarrow$ Minimum allowable deadline (reaction time)
- t_{Dmax} $\rightarrow$ Maximum allowable deadline (reaction time)
- T_E $\rightarrow$ Execution (computation) time: $T_E = t_f - t_s$
- T_R $\rightarrow$ Response time: $T_R = t_f - t_a$
- T_L $\rightarrow$ Lateness: $T_L = t_f - t_{Dmax}$ (usually negative)
- T_T $\rightarrow$ Tardiness or exceeding time: $T_T = \max(0, T_L)$
- T_{RR} $\rightarrow$ Remaining response time: $T_{RR} = t_{Dmax} - t$
- T_{RE} $\rightarrow$ Remaining execution time: $T_{RE} = t_f - t$
- T_X $\rightarrow$ Laxity or slack time: $T_X = (t_{Dmax} - t) - T_{RE}$ or $T_{X_a} = (t_{Dmax} - t_a) - T_E$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Real-time requirements: period and rate

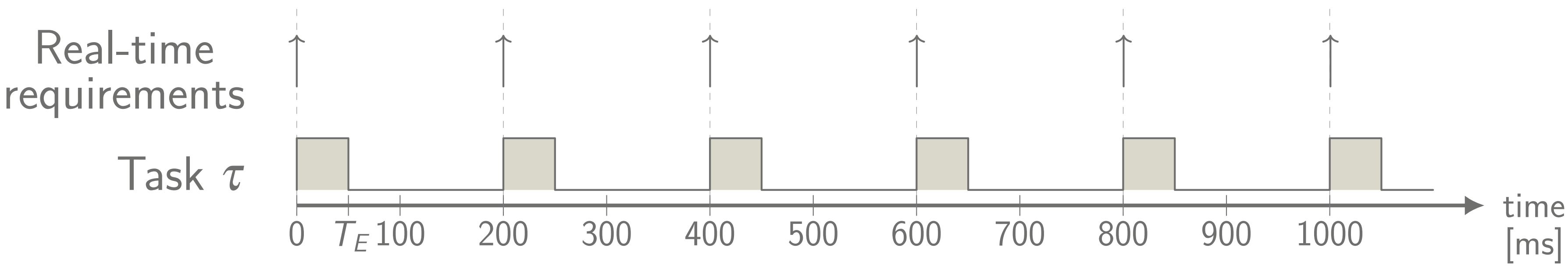


- T_P $\rightarrow$ Period, time between realtime requirements
- $T_{P\min}$ $\rightarrow$ Minimum period
- $T_{P\max}$ $\rightarrow$ Maximum period
- R $\rightarrow$ Rate $R = \frac{1}{T_P}$
- $R_{\max}$ $\rightarrow$ Maximum rate $R_{\max} = \frac{1}{T_{P\min}}$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

CPU/μC core allocation

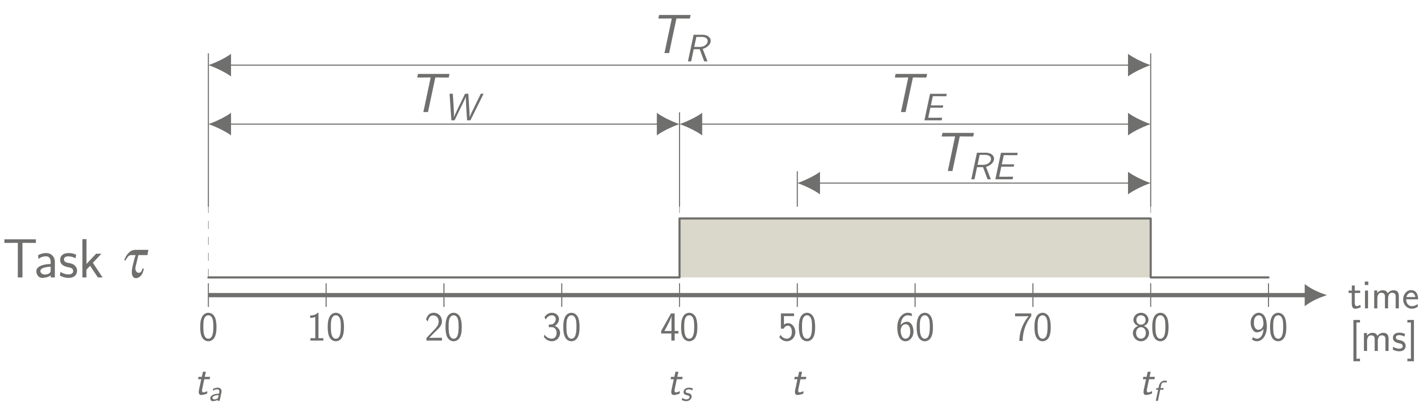


- T_E $\rightarrow$ Execution time:
  The CPU/μC core is allocated for 50 ms: $T_E = 50$ ms
- U $\rightarrow$ Utilisation $U = \frac{T_E}{T_P}$
- U_{max} $\rightarrow$ Utilisation $U_{max} = \frac{T_{Emax}}{T_{Pmin}}$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Waiting and execution time

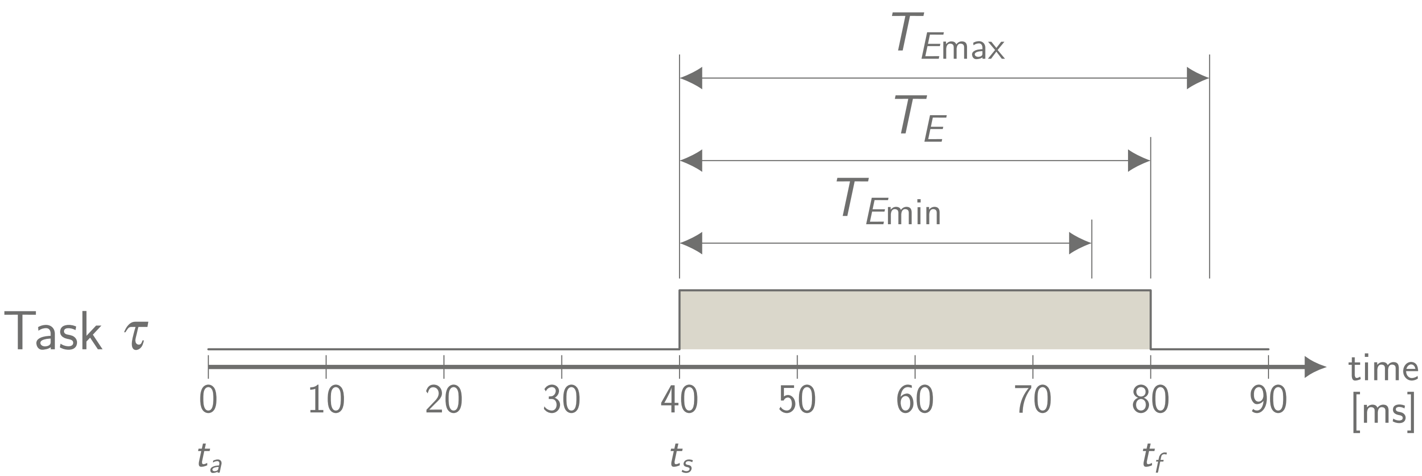


- T_W $\rightarrow$ Wait time: $T_W = t_s - t_a$
- T_E $\rightarrow$ Execution (computation) time: $T_E = t_f - t_s$
- T_R $\rightarrow$ Response time: $T_R = t_f - t_a$ or $T_R = T_W + T_E$
- T_{RE} $\rightarrow$ Remaining execution time: $T_{RE} = t_f - t$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Best and worst case execution times



Best case execution time (BCET):

- Minimum execution time $T_{E\min}$
- The BCET is important if a $t_{D\min}$ exist

Determine BCET:

- Analytical: Shortest path through source code (determine number of cycles)
- Experimental: Measure runtime (system should be load-free)

Worst case execution time (WCET):

- Maximum execution time $T_{E\max}$
- The WCET is important if a $t_{D\max}$ exist

Determine WCET:

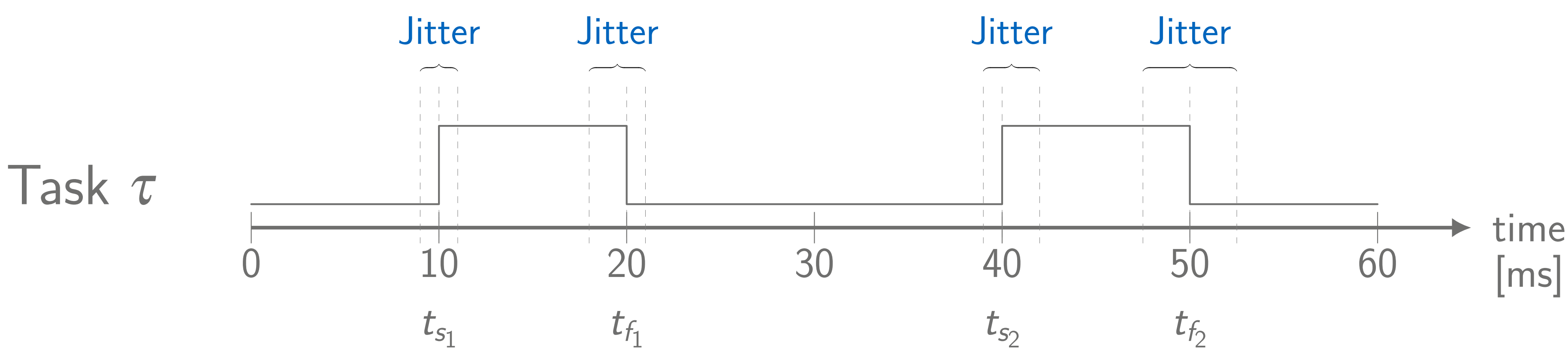
- Analytical: Longest path through source code (determine number of cycles)
- Experimental: Measure runtime (system can/should be under load)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Jitter

The jitter is the deviation of a periodic signal.



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Timing conditions

Real-time systems have to guarantee:

- Correctness of the sequential parts of the program
- Correctness of tasking (including synchronisation and communication)
- Timeliness (Rechtzeitigkeit)

CAMPUS Rosenheim
Computer Science

Timeliness

Hard to proof formally:

- For periodic, mixed known and unknown spontaneous alarms
- Always use the worst case: $\Rightarrow$ WCET for the analysis
- Runtime of program required (estimation, measure, time analysis programs)

Determine timeliness via heuristics:

- Measure/determine WCET
- Calc/determine maximal utilisation U_{max}
- In practice, $\boldsymbol{U_{\text{max}}} \leq \mathbf{0.3}, \ldots, \mathbf{0.5}$ to have sufficient safety reserves.

For higher utilisations:

- Use mathematical methods (e.g. scheduling theory)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example 1

Printer street: Fa. Bosch-Rexroth

Given:

- Paper speed: $v = 25$ m/s
- Required print accuracy: 0.25 mm

Question:

- Determine the period: T_P
- What is the required real-time requirement?

Proposed solution:

- $v = \frac{s}{t} \Rightarrow t = \frac{s}{v} = \frac{0.25 \text{ mm}}{25 \text{ m/s}} = \frac{0.25 \text{ mm}}{25000 \text{ mm/s}} = 0.00001 \text{ s} \Rightarrow 10\ \mu\text{s}$

$\Rightarrow$ $T_P = 10\ \mu$s

- The printing has to be finished within 10 μs

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example 2
Signal sampling

Given:

- Maximum frequency of signal: $f_{\text{max}} = 1$ kHz

Question:

- Determine sampling frequency f_s, considering $f_s > 10 \times f_{max}$
- Determine the period: T_P
- What is the required real-time requirement?

Proposed solution:

- $f_s = 10 \times 1 \text{ kHz} = 10 \text{ kHz}$

$\Rightarrow$ $T_P = \frac{1}{10 \text{ kHz}} = 100\ \mu s$

- A measurement has to be taken every 100 μs

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

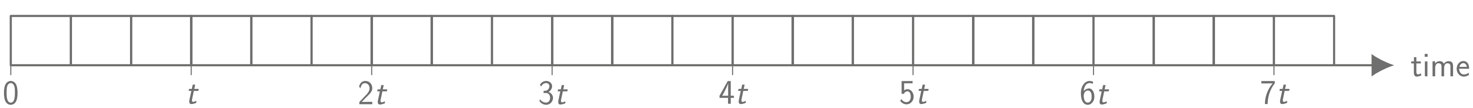
Example 3
Multiple time-delayed actions

Multiple actions should be run in a time-delayed manner:

- Action A_1 every t_1 s (e.g. temperature acquisition)
- Action A_2 every t_2 s (e.g. recording of pressure values)
- Action A_3 every t_3 s (e.g. recording of switch positions)

In every action, values have to be: read, scaled, calculated, and stored into a buffer.

It now applies: $t_1 \approx 1t; t_2 \approx 3t; t_3 \approx 6t$



The time conditions sometimes lead to an uneven utilisation of the system, and all $6t$, a peak load occurs.

Real-time constraints are:

- **violated**, if $\sum_{i=1}^{3} T_{E_{A_i}} > 1t$
- **met**, if $\sum_{i=1}^{3} T_{E_{A_i}} \leq 1t$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Exercise: bike computer

Consider a bike computer for a racing bicycle.

[source: biker-boarder.de]

Basic calculations:

- Determine the real-time requirements
  - Tire diameter → circumference
  - Max. speed
- Determine periods T_P, $T_{P\min}$, and $T_{P\max}$
- Maximum allowable deadline $t_{D\max}$
- Determine rate R and $R_{\max}$

Questions:

- What max. execution time $T_{E\max}$ is allowed, if the utilisation is $U_{\max} \leq 0.5$
- What time constraints do we have: hard, firm, or soft?
- Is a minimum allowable deadline $t_{D\min}$ required?

source idea: [1, Quade, c. 2]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Exercise: bike computer

Solution proposal

Basic calculations:

- Real-time requirements
  - Tire diameter: 68 cm
  - Circumference: 1200 mm – 2300 mm
  - Max. speed: $v = 150\ \text{km/h} = \frac{150000\ \text{m}}{60 \times 60\ \text{s}} \approx 41.67\ \text{m/s}$

$\Rightarrow$ $T_P = \frac{68\ \text{cm} \times \pi}{41.67\ \text{m/s}} = \frac{0.68\ \text{m} \times \pi}{41.67\ \text{m/s}} = 0.05124\ \text{s} \approx 51.24\ \text{ms}$

$\Rightarrow$ $T_{P\min} = \frac{1200\ \text{mm}}{41.67\ \text{m/s}} = \frac{1.2\ \text{m}}{41.67\ \text{m/s}} = 0.02879\ \text{s} \approx 28.8\ \text{ms}$

$\Rightarrow$ Min speed: $v = 0\ \text{km/h}; \Rightarrow T_{P\max} = \infty$

$\Rightarrow$ $t_{D\max} = T_{P\min} \approx 28.8\ \text{ms}$

$\Rightarrow$ $R = \frac{1}{T_P} = \frac{1}{51.24\ \text{ms}} = \frac{1}{0.05124\ \text{s}} = 19.51$

$\Rightarrow$ $R_{\max} = \frac{1}{T_{P\min}} = \frac{1}{28.8\ \text{ms}} = \frac{1}{0.0288\ \text{s}} = 34.72$

Proposed solution:

- $T_{E\max} = t_{D\max} * U_{\max} = 28.8\ \text{ms} * 0.5 = 14.4\ \text{ms}$
- Time constraint: **firm** or soft
- A $t_{D\min}$ is not required

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Predictability

One of the most important properties that a hard real-time system should have is predictability [SR90].

Goal:

- Guarantee in advance that all critical timing constraints are met

Depends on:

- Architectural features of the hardware (process + peripherals)
- Mechanisms and policies adopted in the kernel
- Programming language and techniques used to implement real-time application

source: [2, Buttazzo, p. 13]

[SR90] J. A. Stankovic and K. Ramamritham. What is predictability for real-time systems? Journal of Real-Time Systems, 2, 1990.

Overview of affects on predictability 1/2

Architectural features of the hardware (processor + peripherals)

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

Mechanisms and policies adopted in the kernel

- Scheduling algorithm
- Synchronisation and communication mechanism
- Types of semaphores
- Memory management policy (swapping)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Overview of affects on predictability 2/2

A problem of programming languages:

- It's not possible to define timing constraints for certain tasks or functions (C, C++, Ada)

Programming techniques used to implement real-time application

- Fixed-size integer types
- Multiplication over division
- Constexpr
- Fixed size data structures
- `Static_assert`
- Constant loop iterations
- Avoid conditional execution
- Init at startup

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Fixed-size integer types

Prefer fixed-size integer over standard integer types

C/C++

```c
#include <inttypes.h> //int8_t, uint8_t, int16_t, uint16_t, ...
#include <stdlib.h>   //EXIT_SUCCESS

int main(void) {
  //prefer this
  int8_t   value_s8_bit   = INT8_C(1);
  uint8_t  value_u8_bit   = UINT8_C(1);
  int16_t  value_s126_bit = INT16_C(1);
  uint16_t value_u126_bit = UINT16_C(1);
  //...

  //over
  int  value_s = 1; //how much bits are used?
  uint value_u = 1U; //how much bits are used?

  return EXIT_SUCCESS;
}
```

Problem:

- How much bits are used?
- How long take operations like +,-,...

Advantage:

- More clear how much bits are used for a variable
- Easier to derive computation time

Multiplication over division

Prefer multiplication over division where possible

C/C++

```c
#include <inttypes.h> //uint13_t, ...
#include <stdlib.h>   //EXIT_SUCCESS

int main(void) {
  //example 1:
  uint32_t value = UINT32_C(1000);
  //the div can be replaced
  value = value / 2;
  //by a shift
  value = value >> 1;


  //example 2:
  uint32_t j = 0, i = 0, min_length = 30;
  //the div can be replaced
  if ((j - i) / 2 >= min_length) {}
  //by a mul
  if ((j - i) >= min_length * 2) {}

  return EXIT_SUCCESS;
}
```

Problem:

- A division (DIV instruction) may cause different cycle times
- E.g. Cortex M4: SDIV/UDIV can take 2 to 12 cycles

Advantage:

- More stable computation time

Disadvantage:

- Harder to read and understand

CAMPUS Rosenheim
Computer Science

Constexpr

Use constexpr for a guaranteed compile-time constant

C++

```
#include <cinttypes> //uint13_t, ...
#include <cstdlib>   //EXIT_SUCCESS

int main(void) {
  //prefer
  constexpr uint32_t value1 = (3+3)*1000;

  //over
  const     uint32_t value2 = (3+3)*1000;

  return EXIT_SUCCESS;
}
```

Problem:

- Is the constant part of the code, or is it computed during runtime?

Advantage:

- Guaranteed compile-time constant

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Fixed size data structures

Try to avoid dynamic data structures

C++

```
#include <cinttypes> //uint13_t, ...
#include <cstdlib>   //EXIT_SUCCESS
#include <array>
#include <vector>

int main(void) {
  //prefer
  std::array<uint8_t, 3U> stl_arr{0U,1U,2U};

  //over
  std::vector<uint8_t> stl_vector{0U,1U,2U};
  //or the old-fashioned
  uint8_t array[3U] = {0U,1U,2U};

  return EXIT_SUCCESS;
}
```

Problem:

- The size of `std::vector` may not be known at compile time and grow/shrink over time

Advantage:

- The size of `std::array` is known at compile time
- As comfortable as every other STL container

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Static_assert

Use `static_assert` for compile time checks

C/C++

```c
#include <inttypes.h> //uint8_t,
#include <stdlib.h>   //EXIT_SUCCESS
#include <assert.h>   //static_assert
#include <stdio.h>    //printf

int main(void) {
  const uint8_t version = UINT8_C(5);

  //prefer
  static_assert(version >= 4, "Requires at least version 4");

  //over
  if(version < 4){
      printf("Error: Requires at least version 4");
      exit(EXIT_FAILURE);
  }

  return EXIT_SUCCESS;
}
```

Advantage:

- Avoids error paths
- Asserted at compile time

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Constant loop iterations

Try to always use same number of iterations

C/C++

```
const uint8_t LEN = 10;
uint8_t data[LEN];

{ //prefer
  int8_t found_index = -1;
  for(uint8_t i = 0; i < LEN; ++i){
    if(data[i] >= 0 && found_index == -1){
        found_index = i;
    }
  }
}

{ //over
  uint8_t i = 0;
  for(i = 0; i < LEN; ++i){
      if(data[i] >= 0){
          break;
      }
  }
  int8_t found_index = i >= LEN ? -1 : i;
}
```

Problem:

- If a loop is exited early (depending on some data), the number of iterations may differ from cycle to cycle.

Advantage:

- Always performs the same number of iterations

Disadvantage:

- May not feel intuitive when writing the code

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Avoid conditional execution

Avoid conditional execution where possible

C/C++

```
#include <inttypes.h> //uint8_t
#include <stdlib.h>   //EXIT_SUCCESS

int main(void) {
  const uint8_t LEN = 10;
  uint8_t data[LEN];

  for(uint8_t i = 0; i < LEN; ++i){
    //prefer
    uint8_t val = data[i] > 0 ? 5 : 0;
    data[i] += val;

    //over
    if(data[i] > 0){
        data[i] += 5;
    }
  }

  return EXIT_SUCCESS;
}
```

Problem:

- The code inside the `if` is not always executed

Advantage:

- The same code is always executed

Hints:

- This is often possible in mathematical calculations with a neutral element

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Init at startup

Try to initialise everything on startup

C/C++

```
#include <inttypes.h> //uint8_t
#include <stdlib.h>   //EXIT_SUCCESS
#include <stdbool.h>  //bool: true/false

int main(void) {
  const uint8_t LEN = 5;

  //prefer
  uint8_t data[LEN] = {0}; //init
  //over
  uint8_t* p = NULL;

  while(true){
    data[3] = UINT8_C(5); //usage

    //over
    if(p == NULL){
        p = (uint8_t*)calloc(LEN, sizeof(uint8_t)); //init
    }
    p[3] = UINT8_C(5); //usage
  }

  return EXIT_SUCCESS;
}
```

Problem:

- Allocation of heap memory may take some (unpredictable) time

Advantage:

- Initialisation is done in the initialisation phase, before the cyclic processing starts
- Usage is predictable

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary and outlook

Summary

- Timing symbols
- Timing conditions
- Predictability

Outlook

- Process computer hardware