Aachen, April, 2023
SWS: V3/Ü1, ECTS: 6
Professor Dr.-Ing. Stefan Kowalewski
Simon Fonck, M.Sc. RWTH
Alexander Kruschewsky, M.Sc. RWTH

Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

Exercise for
Embedded Systems
Summer Term 2023
Sheet 1: Embedded Systems and Microcontrollers

Exercise 1: Questions

a) What is an embedded system?

Solution:

- A computer system (CPU, memory, software)
- which is integrated into another technical system ”embedding system”
- influences the embedding system such that it behaves in the desired way

b) What is the source of requirements for an embedded system?

Solution:
Requirements are derived from the requirements of the embedd<u>ing</u> system.

c) Embedded systems can be categorized into two groups.

(1) What are these categories?
(2) What are their characteristics?
(3) What kind of hardware is typically used for these categories?
(4) Which programming languages are most dominant in these categories?
(5) Name at least one example for each category!

Solution:

1)	Product Automation	Production Automation
2)	many identical units cost per unit is critical customers aren't experts hardware determines platform	often only one identical unit cost is less critical costumers are close to being experts programming system is more important then the platform
3)	μCs, FPGAs	PLCs, Industrial PCs, Distributed Control Systems
4)	C/C++, Assembler, VHDL, Simulink	Instruction List (IL), SFC, ST, FBD
5)	Carengine Controller Washing Machine Controller Weather Station ...	Chemical Plant Controller Assembly Line Controllers ...

d) What is a microcontroller?

Solution:
A microprocessor with RAM, Permanent memory, Digital I/O and other peripherals.

Exercise 2: Digital I/O
For this and the following exercises we refer to the Atmel ATmega16 microcontroller. A datasheet can be found in the Moodle room.
Assume 8 buttons are connected to Port A and GND. Also assume 8 LEDs are connected to Port B and VCC (using fitting resistors) such that they can be lit.

a) What are the registers that control these ports?

Solution:
DDRA, DDRB: Data direction
PORTA, PORTB: Outputs / pull-up resistors
PINA, PINB: Inputs

b) How should these registers be initialized?

Solution:
DDRA = $(00000000)_2 = (00)_{16}$ Remember: 0 means input
DDRB = $(11111111)_2 = (FF)_{16}$ Remember: 1 means output
PORTA = $(11111111)_2 = (FF)_{16}$ Remember: 1 means Pull-Up resistors (otherwise undefined if button not pressed)
PORTB = $(11111111)_2 = (FF)_{16}$ Remember: 1 sets the LEDs to off (V_{CC} on both sides)
PINA, PINB are read only

c) Write a loop that allows control over the LEDs via the buttons:

(1) On a 1-to-1 basis (pushing button 4 causes LED 4 to be lit).

Solution:

```
while (1) {
    PORTB = PINA;
}
```

This is the simplest solution, since if a button is pressed the PIN has a 0 at that position and a 0 also turns the LED on.

(2) Priority encoder: show the binary coding for the number of the highest button pushed.

Solution:

```
while (1) {
    int i;
    for (i=7; i>=0; i–) {; Starting from highest i potentially improves
performance, due to less loops. However, starting from 0 would also work.

        if(~PINA & (1<<i)){; "~" bitwise inverts PORTB -> Ones where
button is pressed. "<<"is the left shift, so "(1<<i)" gives a One at ith position.
So the if-condition is true if ith button is pressed.

            PORTB=~i; Bitwise invert again, because 0 is on.

            break; Break if we got a one to avoid overwriting by lower
positions (not neccessary if starting for-loop from i=0)

        };
    };
}
```

Note: this means that for example if the 5th button is the highest one pushed, PORTB would be $(11111010)_2$ "binary number", not $(11011111)_2$ (counting starts at 0)

d) What is bouncing? Implement a debouncing method.

Solution:
Bouncing is short signal fluctuations before a signal change.
Example bouncing method:

```
uint8_t first, fail;
count = 3; This is the amount of consequitive samples that need to be one before the
signal change is accepted.

do {
      first = 1 & (PINA >> BTN_PIN); BTN_PIN is the index of the PIN we are
trying to debounce. First is then as the name suggests, the first value of the PIN

      fail = 0;
      for(int i = 0; i < count; i++){ Here we go through the "count" samples that we
need.

            if(first != (1 & (PINA >> BTN_PIN))){ & is a bitwise AND, so in total
we check if the value of PIN at position BTN_PIN is unequal to the first value we
recorded of it.

                  fail = 1;
                  break;
            }
      }
}while(fail) We use this to keep the loop running until we finally do record a proper
signal change. Thus if the program ends we did encounter a proper signal change.
```

Note: There are a lot of possible ways to solve this task.

Exercise 3: Interrupts and Polling

a) Choose Interrupts or Polling for the following scenarios and explain your choice.

(1) The ”change input”-button on a monitor

(2) The wireless-reciever of a garage-opener

(3) The keyboard on a standard desktop

(4) The temperature-sensor of a weather-station

Solution:
1) Interrupts. Due to the rarity of this signal, Polling would waste a lot of CPU ressources.
2) Depends. Again the scarcity of such signals would make Interrupts preferable, however since there might be a lot of signal-noise and erroneous garage-openings would be a security concern Polling would be preferable in that regard.
3) Polling. Interrupts could cause programm pausations in undesired moments. Additionally often we would also like to know if a key was pressed for x amount of seconds which would not be measurable with Interrupts.
4) Polling. This allows for better planability of ressource usage. (I.e. how would one scale Interrupts to get high enough detail without having a lot of interrupts when temperature changes quickly)

b) When is an ISR called and how is it done?

Solution:
Conditions: Global Interrupt Enable Bit, Specific Interrupt Enable Bit and Specific Interrupt Flag need to be 1.

1) Hardware stores the Programm Counter
2) Set Global Interrupt Enable Bit to 0
3) Programm Counter is set to the Look-Up-Table
4) Programm Counter Jumps to ISR
5) Context is stored
6) ISR-Code is Executed
7) Context is restored
8) Hardware restores the Programm Counter

Exercise 4: Timers and Counters

a) What is a counter? What is a timer?

Solution:
Counter: Hardware unit that counts external events (Rising edges, falling edges, arbitrary edges)
Timer: Special counter that counts clock cycles

b) What components does timer 1 of the ATmega16 have? How are they configured?

Solution:
Counter Register TCNT1 (TCNT1H, TCNT1L)
Compare Register OCR1 (OCR1H, OCR1L)
Input Capture Register ICR1 (ICR1H, ICR1L)
Control Register TCCR1 (TCCR1A, TCCR1B)

c) How is the reading and writing of a 16 bit value made atomic?

Solution:
Parallel reading and writing by Reading the Low-Byte and Writing the High-Byte first, then vice-versa. (However disabling ISR is neccessary)

d) What is a watch dog?

Solution:
A watchdog is a timer that counts from an initial value to zero. When zero is reached, the μC is restarted. In the main loop of the program, the watchdog counter is reset to the initial value.This is useful to recover from unintended infinite loops: When the main loop is not executed for too long, the Microcontroller restarts.

e) Why might it be necessary to temporarily disable interrupts when reading 16 bit values?

Solution:
Reading 16 bit values requires two cycles (on 8-bit platforms). Interrupts could occur between those two cycles and could corrupt the 16-bit value that is accessed by the main program.

Exercise 5: Analog Devices

a) What analog devices can be found on a ATmega16?

Solution:
4 PWM channels, an 8-channel-10-bit A/D converter (Successive Approximation Converter) and an Analog comparator

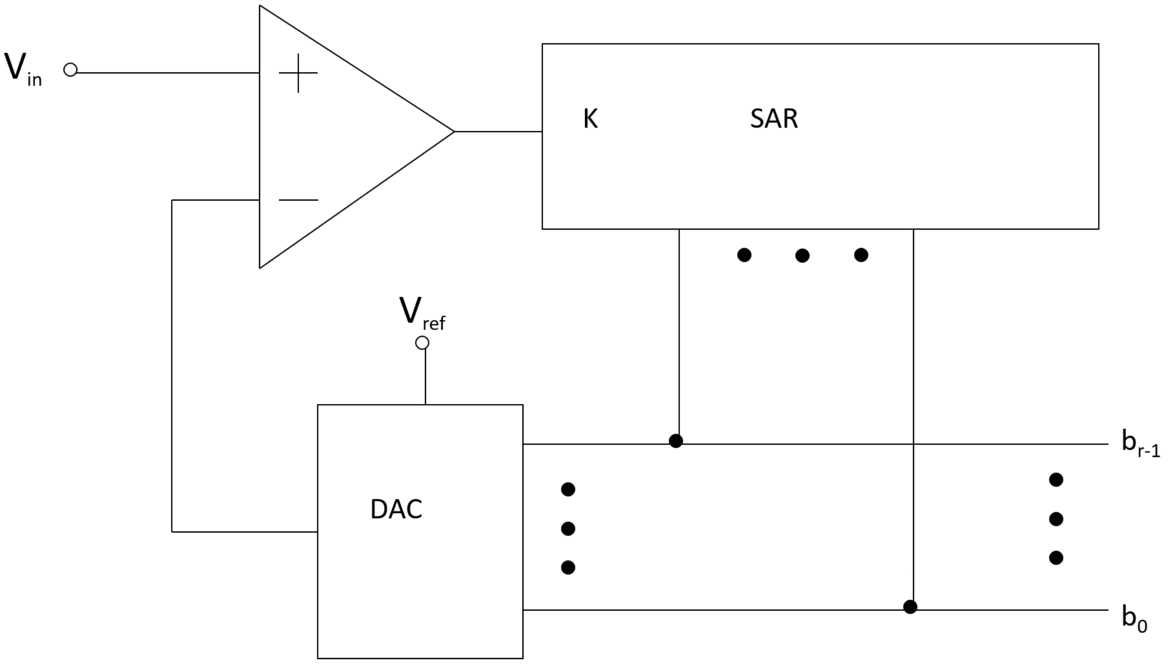
b) What is PWM and how does it work?

Solution:
Pulse width modulation produces a rectangular signal with a configurable frequency and high time. PWM is a hardware feature that usually can produce signals with a much higher frequency than would be possible by producing the signal in software. PWM can be used to reduce the average voltage delivered on a PIN.

The average voltage (also called the value of the signal), can be computed like this: $value = \frac{hightime}{period} \cdot V_{CC}$

c) Sketch a successive approximation converter and explain how it works.

Solution:

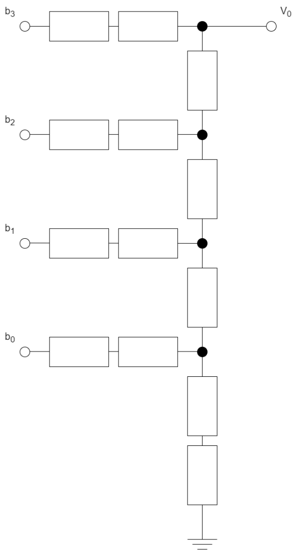


A successive approximation converter performs a binary search through all representable signal levels. In each step, the value of one bit of the digital signal level is determined. The successive approximation converter starts with all level bits set to zero. In the first step, the highest bit is set to one. A DAC produces a signal that corresponds to the current ”guess”. A comparator is used to determine if the signal is greater or smaller than that ”guess”. If it is greater, the bit is left at one, otherwise it is reset to zero. This process is repeated for the second-highest and then all other bits, until all bits are fixed.

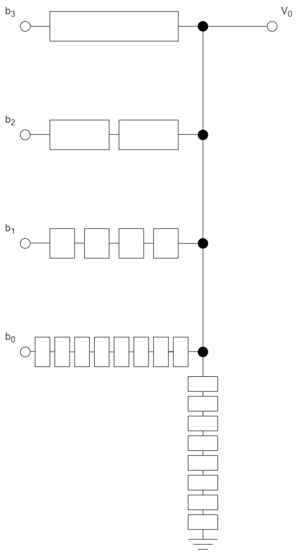
d) Imagine you only have 1 Ohm Resistors available, which cost 10 cents each. What is the minimum achievable cost for a 4Bit R-2R and a 4Bit binary weighted resistor circuit respectively if you can only combine resistors in serial?

Solution:
R-2R: 1,3 Euro (13 resistors)
BWRC: 2,3 Euro (23 resistors)

R-2R

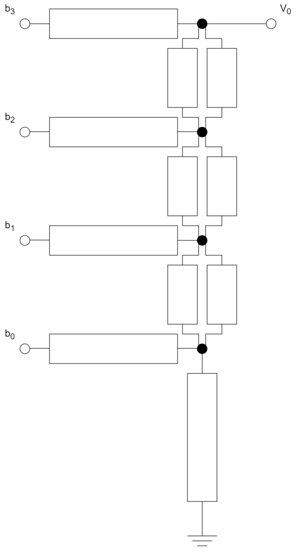


BWRC

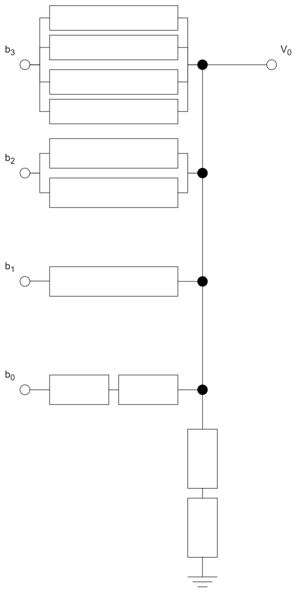


If resistors could also be combined in parallel, both nets would use 11 resistors:

R-2R



BWRC



e) What are the disadvantages of the binary weighted resistor circuit?

Solution:
Many different types resistors or many resistors are needed for BWRC (or many resistors of the same type in series, which is not preferable in practice). This leads to either bad quality of the produced voltage level due to different deviations from the nominal value of the resistances, or to high costs if high-precision resistances are used.

Aachen, May, 2023
SWS: V3/Ü1, ECTS: 6
Professor Dr.-Ing. Stefan Kowalewski
Simon Fonck, M.Sc. RWTH
Alexander Kruschewsky, M.Sc. RWTH

Exercise for
Embedded Systems
Summer Term 2023
Sheet 2: Data Buses

Exercise 1: Basics

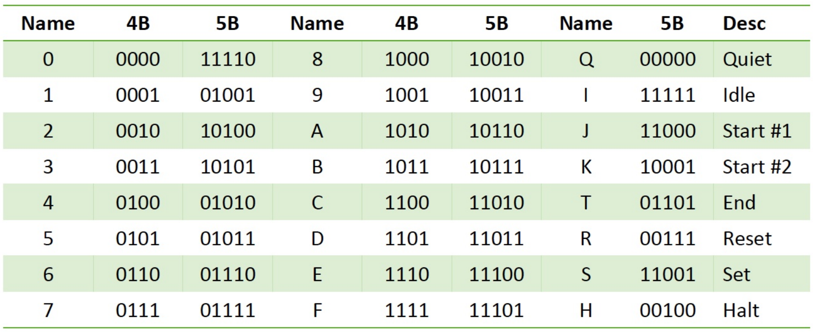
- Name two advantages that the star topology has over the bus topology.

  Solution:

  - Multiple senders at same time
  - Only one single point of failure
  - No collisions
  - No implicit boradcasts

- Use the 4B/5B table introduced in the lecture to encode the message **000000111111**. Then draw the signal diagram using NRZI (that is Differential NRZ with inverted semantics: a level change represents a 1).

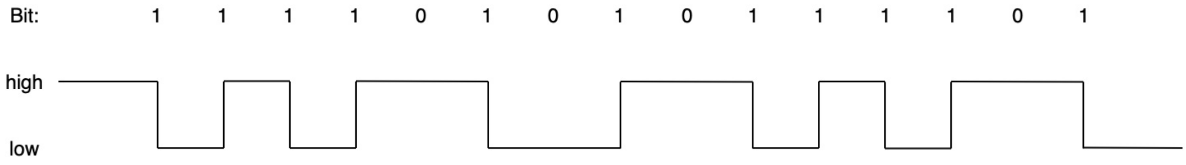
  Solution: First we use the 4B/5B Table to convert our message:



Name	4B	5B	Name	4B	5B	Name	5B	Desc
0	0000	11110	8	1000	10010	Q	00000	Quiet
1	0001	01001	9	1001	10011	I	11111	Idle
2	0010	10100	A	1010	10110	J	11000	Start #1
3	0011	10101	B	1011	10111	K	10001	Start #2
4	0100	01010	C	1100	11010	T	01101	End
5	0101	01011	D	1101	11011	R	00111	Reset
6	0110	01110	E	1110	11100	S	11001	Set
7	0111	01111	F	1111	11101	H	00100	Halt

  We therefore divide the message into 4Byte Chunks: 0000 0011 1111
  and get 11110 10101 11101 after aplying the table.

Now simply use NRZI (A one equals a level change and a 0 equals no level change) and yield the following result:



Having an inverted signal diagram (starting at low and ending at high) is also correct.

- What are the names of the two sublayers of Layer 2? What are their tasks?

  Solution:
  The sublayers of the Data Link Layer (DLL are:)

  - Logic Link Control (LLC)
    - Defines the frame format
    - Defines Adresses / IDs
    - Provides flow control
    - Provides error detection / correction
  - Medium Access Control (MAC)
    - Defines which participant may send with which (Frequency / Code / Time)
    - Defines how collisions are handled (if possible)

Exercise 2: Hamming code

In 1950 Richard Hamming published his famous Hamming code. It's a linear error-correcting code, i.e., you cannot only detect errors, but also fix them. In this code, $2^n - 1$ bit are divided into n parity bits and $2^n - n - 1$ data bits. These bits are numbered from 1 to $2^n - 1$ and bits whose index is a power of two are the parity bits. E.g., for $n = 5$, bits 1, 2, 4, 8, and 16 are parity bits. Each data bit is protected by a unique set of parity bits: decompose the index of a data bit into a sum of unique powers of two; these summands are the parity bits for that data bit.
Example: Data bit 15 is protected by parity bits 1, 2, 4, and 8, because $1 + 2 + 4 + 8 = 15$.

- Calculate the even parity Hamming code of the following bit sequence:
  11001110

  Solution:
  We have 8 data bits, thus we need 4 parity bits. This gives us up to 11 data bits, while 3 parity bits would only give us 4 data bits.
  By putting the parity bits on the powers of two positions we now have:

  ??1?100?1110

  Now we just need to calculate the parity bits:

  - Parity 1: There need to be an even amount of ones out of Positions 1,3,5,7,9,11. Since 3,5,7,9 and 11 have a total of 4 ones, Parity 1 has to be a 0.
  - Parity 2: There need to be an even amount of ones out of Positions 2,3,6,7,10,11. Since 3,6,7,10 and 11 have a total of 3 ones, Parity 2 has to be a 1.
  - Parity 4: There need to be an even amount of ones out of Positions 4,5,6,7,12. Since 5,6,7 and 12 have a total of 1 ones, Parity 4 has to be a 1.
  - Parity 8: There need to be an even amount of ones out of Positions 8,9,10,11,12. Since 9,10,11 and 12 have a total of 3 ones, Parity 8 has to be a 1.

  Example: Since 11=8+2+1, the Position is used with Parity 1,2 and 8 but not 4.

  The result is: 011110011110

- Correct and extract the 11 bit data sequence from this Hamming code protected sequence (even parity):
  100100101100011

  Solution:
  First we mark the positions of the parity bits:

  100**1**001**0**1100011

  Now we need to check if the parity Bits match the data:

  - Parity 1: There need to be an even amount of ones out of Positions 1,3,5,7,9,11,13,15. These positions have a total of 4 ones, so it matches the data.
  - Parity 2: There need to be an even amount of ones out of Positions 2,3,6,7,10,11,14,15. These positions have a total of 4 ones, so it matches the data.
  - Parity 4: There need to be an even amount of ones out of Positions 4,5,6,7,12,13,14,15. These positions have a total of 4 ones, so it matches the data.
  - Parity 8: There need to be an even amount of ones out of Positions 8,9,10,11,12,13,14,15. These positions also have a total of 4 ones, so it matches the data.

  We cannot detect any errors, so we just simply remove the parity bits to get the result.
  The result after correction is: 00011100011

Exercise 3: Inter-Integrated Circuit

- What topology uses I^{2}C?

  Solution:
  Bus Topology

- What does Wired-AND mean?

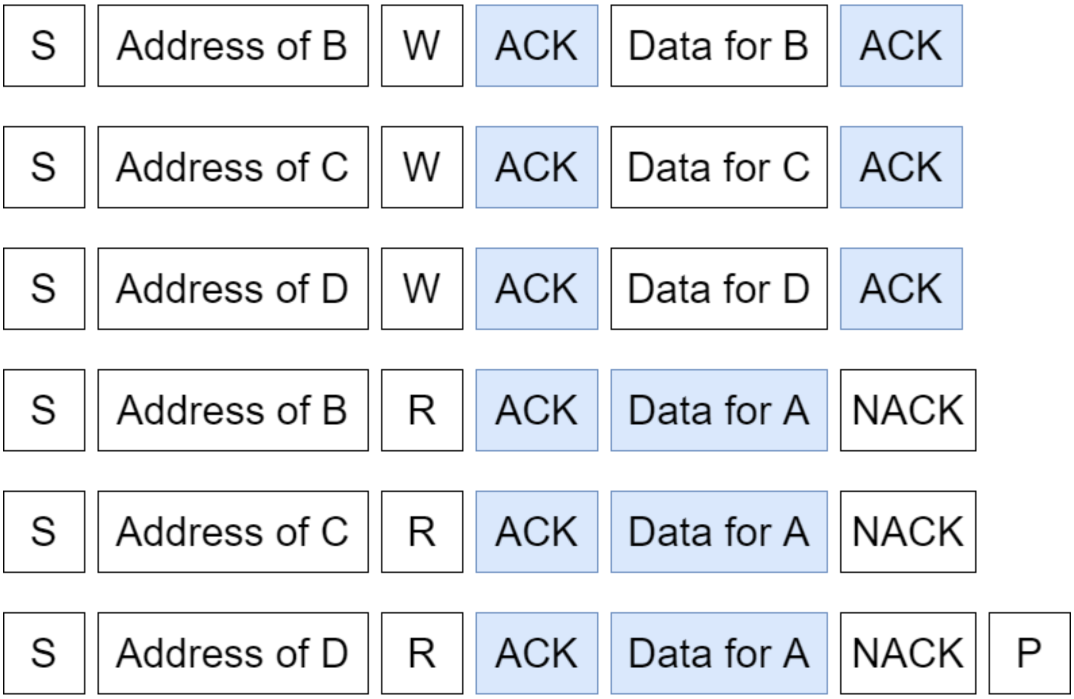
  Solution:
  A zero on the Bus is dominant.

- What is the basic MAC concept? Why are there no collisions when multiple senders start at the same time?

  Solution:
  The basic concept of Master-Slave is that the Master decides who can send at which time, while Slaves can only answer to read requests by Master.
  When there are multiple Masters the dominant 0 ”wins”, thus the Masters with a one at the first bit that differs between the senders stops transmitting.

- Use a block scheme to depict the following communication. A sends three participants B, C, and D a data byte each. Immediately after A sent the third data byte, it starts querying B, C, and D for a one byte response. You may use these blocks ($X \in \{A, B, C, D\}$), but you must indicate who controls the data line for each block:
  - Start-Flag
  - Stop-Flag
  - Address of X
  - Data for X
  - Read-Flag
  - Write-Flag
  - (N)ACK-Flag

Solution:

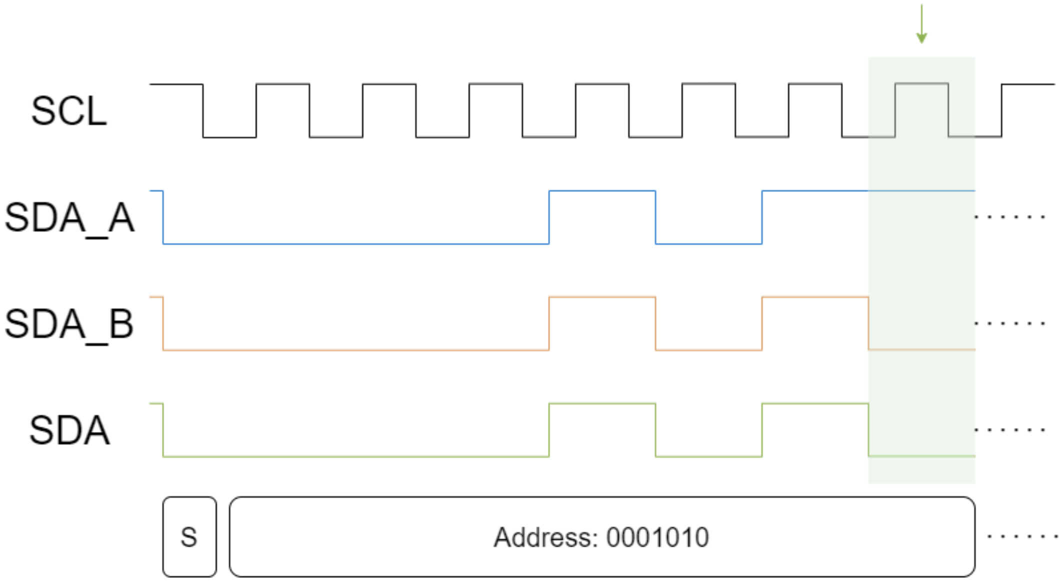


S	Address of B	W	ACK	Data for B	ACK	
S	Address of C	W	ACK	Data for C	ACK	
S	Address of D	W	ACK	Data for D	ACK	
S	Address of B	R	ACK	Data for A	NACK	
S	Address of C	R	ACK	Data for A	NACK	
S	Address of D	R	ACK	Data for A	NACK	P

Blue blocks are the ones not sent by A.
Note: NACK can be sent by Masters to signalize to a Slave that no further Data should be sent.

- A (address = 0x0a) and B (address = 0x0b) want to send messages to each other at the same time. A's message is 0xaa while B's message is 0xbb. Assume their clocks are perfecly synchronous, there is no transmission delay on the bus, and they start at the exact same point in time. Draw SCL, SDA_A, SDA_B, and SDA until the first transmission is complete. Mark the bit that decides who becomes master.

  Solution:

  The green marking shows the deciding Bit.
  The first differing Bit is the last Address-Bit. Thus B wins the arbitration (Due to sending to "0001010") and can keep sending, while A has to wait for B's transmission.

Exercise 4: Masters of Profibus
What is the topology of Profibus (using regular wire) and how is medium access controlled?

Solution:
Profibus uses a Bus Topology.
Medium Access is controlled through Master/Slave and Multiple Masters are managed using a Token Bus.

Aachen, May, 2023
SWS: V3/Ü1, ECTS: 6
Professor Dr.-Ing. Stefan Kowalewski
Simon Fonck, M.Sc. RWTH
Alexander Kruschewsky, M.Sc. RWTH

Exercise for
Embedded Systems
Summer Term 2023
Sheet 3: Programmable Logic Controllers

Exercise 1: Running in circles

- Explain the term *cyclic execution.*

  Solution:
  There are four repeating steps:

  - Internal Checks
  - Reading Hardware Inputs
  - Proram Execution
  - Writing Hardware Outputs

- What are the two worst-case assumptions that are made when calculating the maximum reaction time?

  Solution:
  The two assumptions are that **the input changes just after the inputs are read** and the **maximum cycle time** is reached.

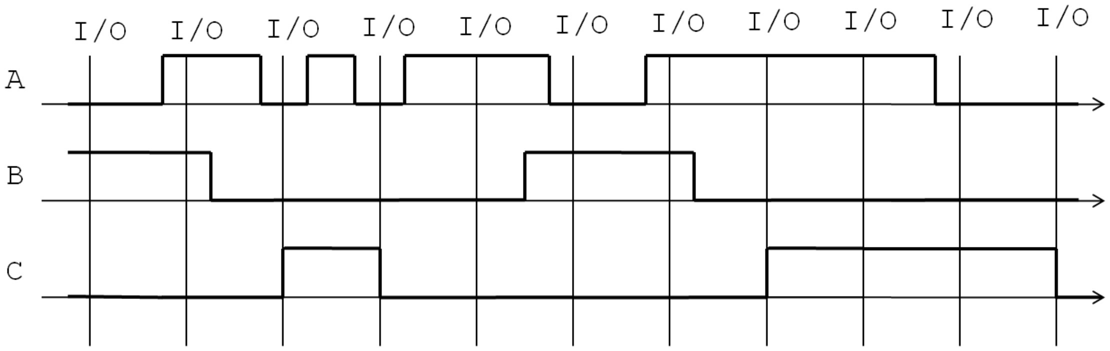
- How are maximum cycle time and maximum reaction time linked?

  Solution:
  Maximum reaction time $\approx 2\cdot$ Maximum Cycle time

Exercise 2: Never too late
Complete the signal diagram below for the following IL program. A and B are inputs while C is an output.

```
LD A
AND B
S C
LDN A
ANDN B
R C
```

Solution:



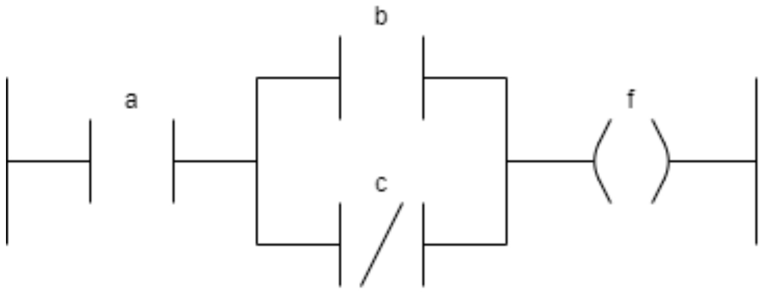
The Programm translates to: $A \wedge B \Rightarrow C = true$ and $\neg A \wedge \neg B \Rightarrow C = false$

You will also notice, that Output seems delayed by one cycle, since after reading the values of A and B the outputs are only written in the next I/O event.

Exercise 3: Fun with functions
Implement the function $f = a \wedge (b \vee \neg c)$ in

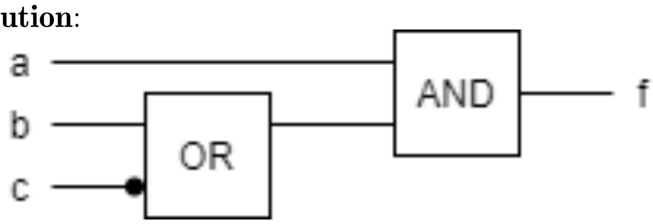
- Ladder diagram

  Solution:



- Function block

  Solution:



- Instruction list

  Solution:

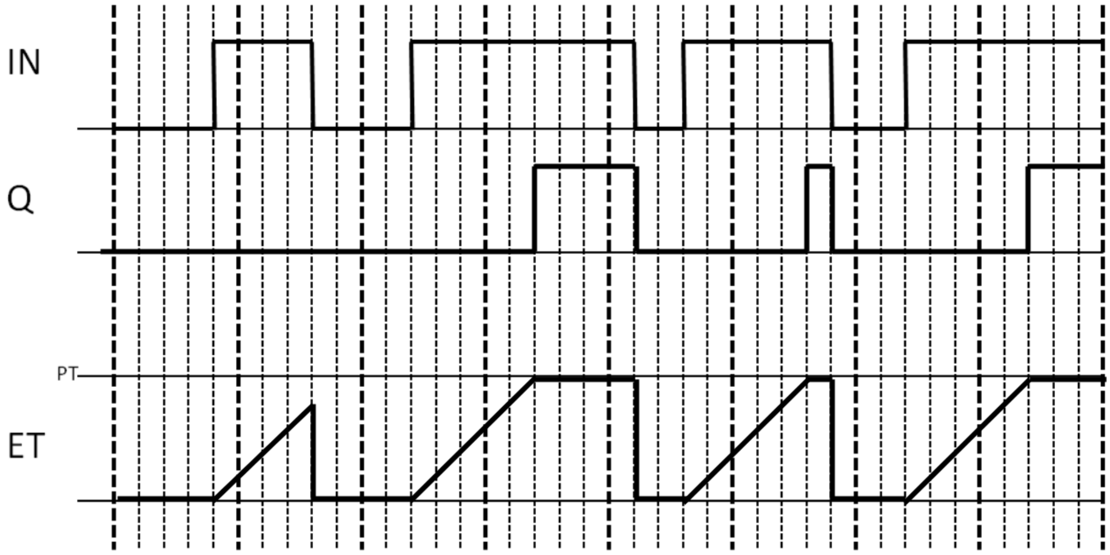
```
LD A
AND (
LD B
ORN C
)
ST F
```

  or

```
LD B
ORN C
AND A
ST F
```

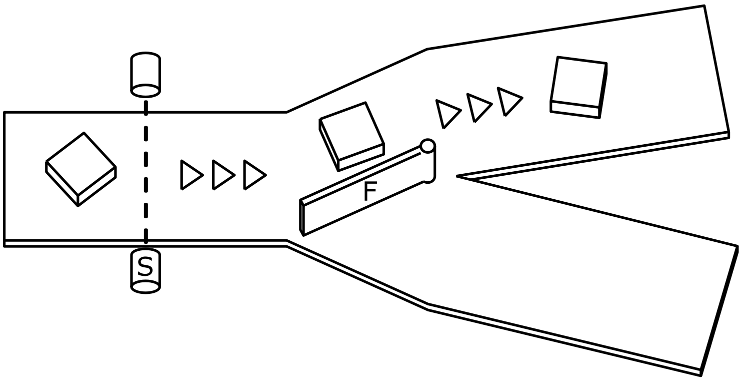
Exercise 4: Good timing
Complete the following diagram of a TON. ET needs five time units to reach PT

Solution:



Q turns 1 after Input stays high for five time units (So when ET reaches PT), but turns off instantly, since this is a Timer-On delay.

Exercise 5: Choosing the right path



The sketch above shows a branch in a conveyor belt. The belt transports objects, which are directed either to the upper or to the lower branch by a separator flap (F). **Write a controller for the flap in IL code with the following behaviour and briefly explain what each line of your code does:**
The flap shall pass the first two objects to the lower branch, then five to the upper one, then two to the lower one, five to the upper, and so on. To count the objects, there is a light barrier switch (S). Assume that there can only be up to one object passing per cycle and that objects passing the light barrier also pass the flap in the same cycle. In other words: When the light barrier variable fires, the object has already passed the flap.
Use 15 instructions or less (it is possible with 6 instructions). Hint: Instead of variables, you can also use number literals as operands. For example, LD 5 sets the CR register to 5.

Use the following variables:

Name	Type	Description
S	input	Equals one for a single cycle when an object passed the light barrier, zero otherwise.
F	output	Controls flap. Directs objects to lower branch when F equals zero, otherwise to the upper branch. Initialized with zero.
C	in/out	Use this to count the objects (increment when S is true). Initialized with zero.

Useful IL instructions:

ADD *X*	Adds an integer *X* to CR.
MOD *X*	Computes CR modulo *X* (for an integer *X*).
RETC	If CR is true, skips the rest of the program (equivalent to jumping to the end of the program).
RETCN	Same as RETCN, but only jumps if CR is false.
GT *X*	Checks if CR is greater than an integer *X*.
LT *X*	Checks if CR is less than an integer *X*.

Solution:

```
LD S // check switch
RETCN // do nothing if no object passed. otherwise continue:
LD C // load the count
ADD 1 // increase count by one
ST C // save the count
GT 1 // is the count greater than one? (in that case, two objects have already passed)
S F // then set the flap to zero
LD C // load counter again
GT 6 // is it greater than 6? Then 7 objects have already passed. Reset to initial state
R F // ...by setting the flap to zero
R C // ... and setting counter to zero
```

There is also an even shorter solution:

```
LD S // check switch
RETCN // do nothing if no object passed
LD C // load count
ADD 1 // increment
MOD 7 // 2+5 = periodicy of 7. "C modulo 7" equals index within the period sequence
ST C // save counter
GT 1 // C >= 2 ? Then upper branch
ST F // set flap to upper or lower branch
```

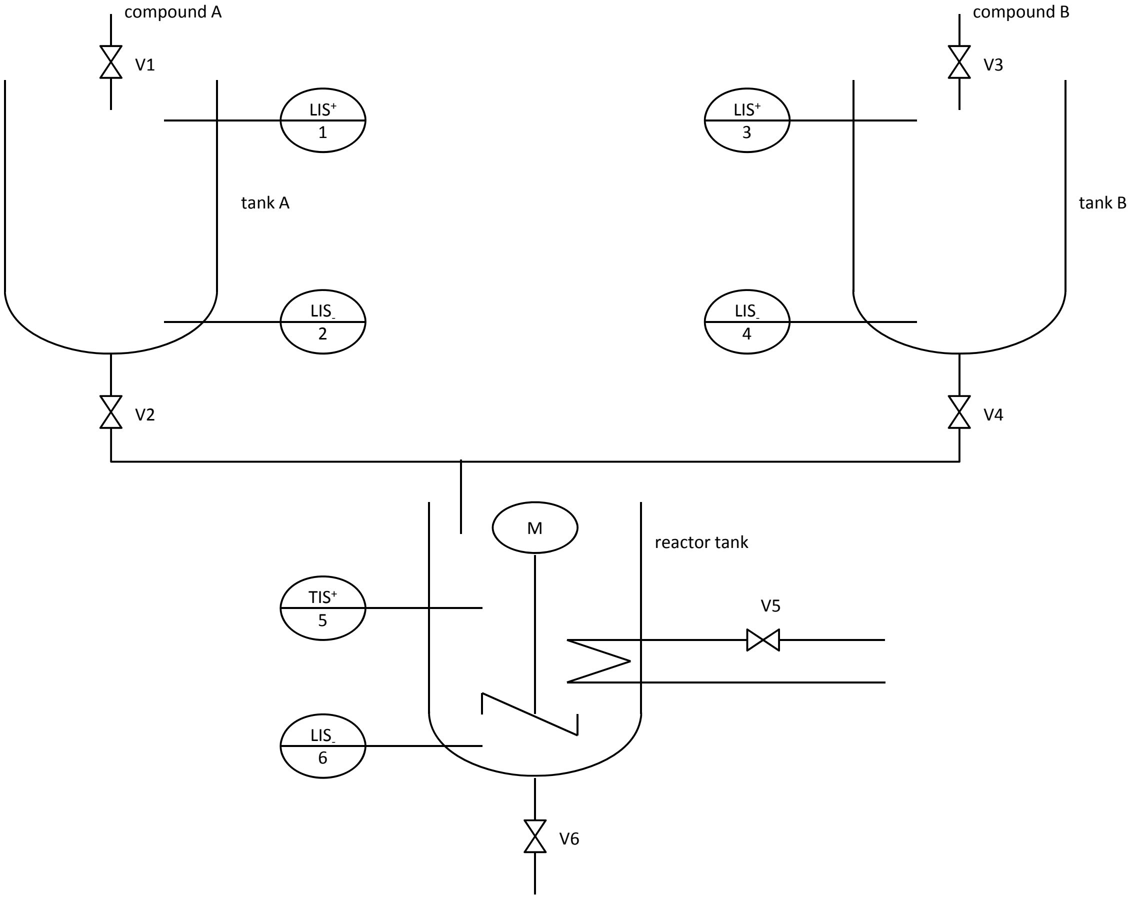
Finally it can further be shortened to:

```
LD C // load count
ADD S // increment if object passed
MOD 7 // 2+5 = periodicy of 7. "C modulo 7" equals index within the period sequence
ST C // save counter
GT 1 // C >= 2 ? Then upper branch
ST F // set flap to upper or lower branch
```

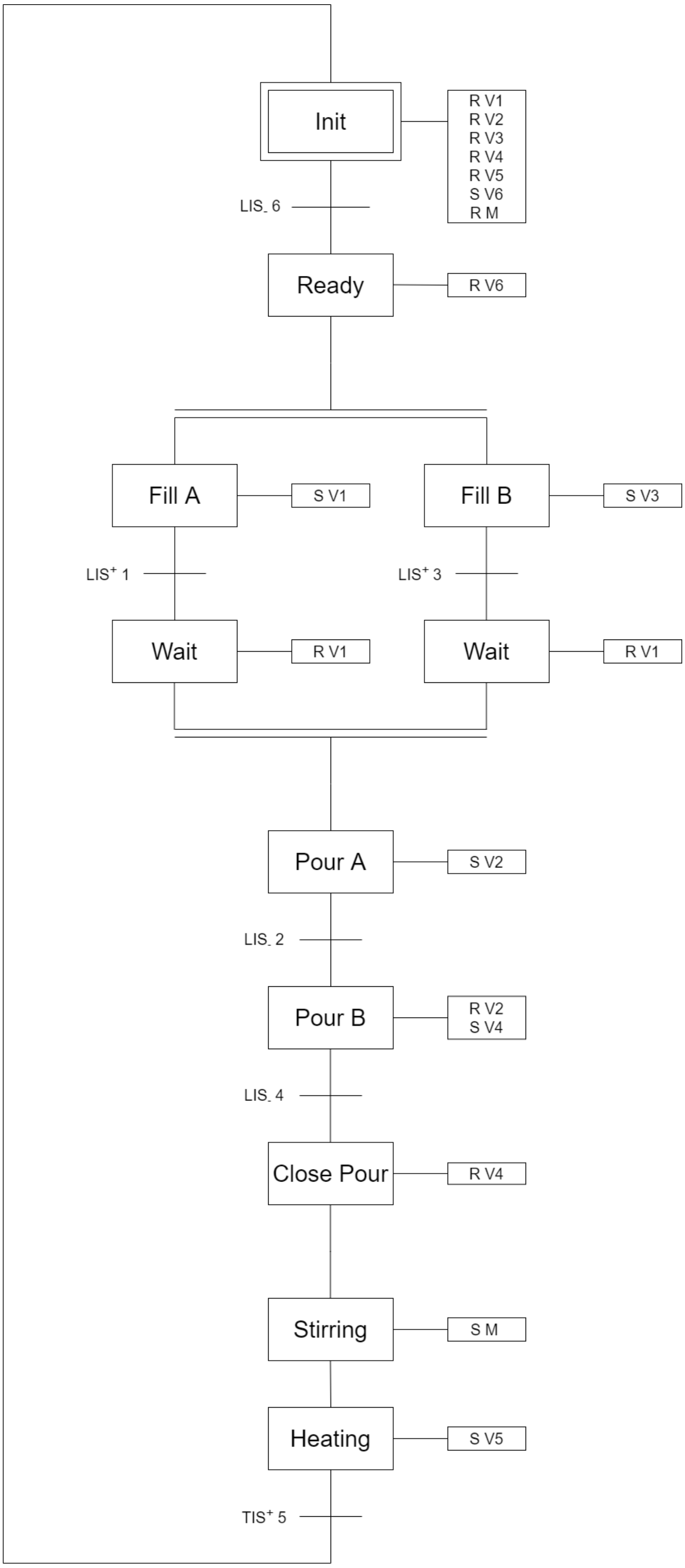
Exercise 6: Secret ingredients
Have a look at the P/ID diagram below. It shows a pipeline and instrumentation diagram of a very simple chemical plant. Two top secret compounds have to be mixed and heated to produce an even more secret compound. As everything is classified your system is completely independent of all other systems. You have been assured that there will always be enough compound A and B to fill your tanks and that you can empty your reactor at any time. You, therefore, only have to follow these important rules:

- The reactor tank can exactly hold the volumes of tank A and tank B combined. There must be no way that the reactor overflows.
- Compound A must never be poured in compound B.
- As timing is highly classified you cannot use timers.
- Tank A and tank B are at the same height. Opening V2 and V4 at the same time might force a backflow causing massive damage to the plant.
- Heating is very delicate and may only be applied when the stirrer is running.
- As soon as the reactor reaches critical temperature the product must be drained.

Create a SFC to control the plant.



Solution:



Init
R V1
R V2
R V3
R V4
R V5
S V6
R M
LIS$_-$ 6
Ready
R V6
Fill A
S V1
Fill B
S V3
LIS$^+$ 1
LIS$^+$ 3
Wait
R V1
Wait
R V1
Pour A
S V2
LIS$_-$ 2
Pour B
R V2
S V4
LIS$_-$ 4
Close Pour
R V4
Stirring
S M
Heating
S V5
TIS$^+$ 5

1) During initialization we need to close every valve that might still be open to avoid liquids going to the wrong tanks or having the heating and stirring on. We also need to drain the reactor tank, especially since it still might contain some residue of compound B.

2) Next the two upper tanks need to be filled with the compounds. Our solution does this in parallel to save time, but it could also be done sequentially. Note however, that closing Valves V1 and V3 has to only depend on the respective LIS+ indicators and not on the finishing of another parallel branch, since fill of the two compounds could take different time.

3) After Filling is done the two compounds can be poured into the reactor tank. Here compound A has to go first as per the rules. Since not stated otherwise, the stirrer could also be running during pour of liquids A and B. A solution where compound A is already poured into the reactor, while tank B is still being filled is also correct.

4) Finally the reaction needs to take place. Here as per the rules the stirrer needs to start before the heating and heating is to be finished when TIS+ 5 is reached.

5) After completion we can simply jump back to the init state, which takes care of draining the reactor and closing everything. But solutions that have a duplicate state for this and jump back to the fill would also work fine, although be less efficient.

Aachen, May, 2023
SWS: V3/Ü1, ECTS: 6
Professor Dr.-Ing. Stefan Kowalewski
Simon Fonck, M.Sc. RWTH
Alexander Kruschewsky, M.Sc. RWTH

Exercise for
Embedded Systems
Summer Term 2023
Sheet 4: Real Time

Exercise 1: Basics

- Name the two requirements needed for real time.

  Solution:
  The requirements are:

  - The Computation is correct
  - The Computation finishes before its deadline

- Expain the terms

  - Hard real time

    Solution:
    The usefulness of a computation is 100% before a deadline, but drops to 0 when the deadline is passed.

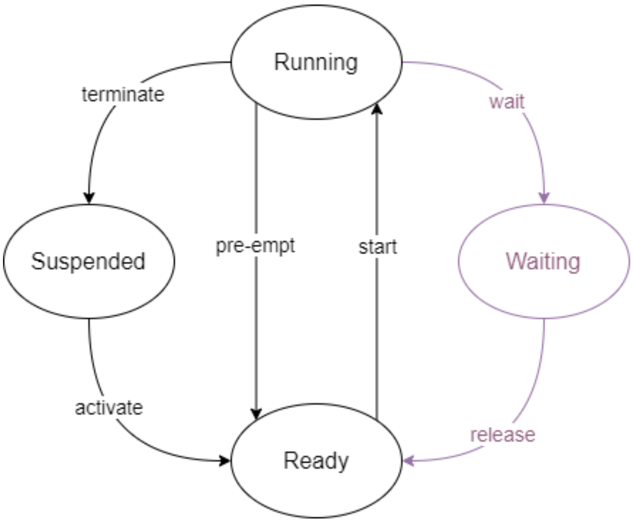
  - Soft real time

    Solution:
    The usefulness of a computation is 100% before a deadline. After the deadline the usefullness is still above 0, but reduces as more time passes.

Exercise 2: OSEK

- Sketch the extended OSEK task model

  Solution:



  Compared to the default OSEK task model, the extension adds the ”Waiting” state colored in purple.

- How many processes and resources are needed for
  - Deadlock
    Solution:
    A Deadlock requires at least 2 Processes and 2 Resources
    A Deadlock occurs when the waiting dependencies of processes form a cycle. For example this occurs, if a Task B would start execution (only one ready) and acquire ressource 1, then be pre-empted by a higher priority Task A which then acquires ressource 2 but also needs ressource 1 before releasing ressource 2. Therefore Task A enters the Waiting state (waiting on B). Now B resumes execution but needs ressource 2 before releasing ressource 1. Thus B now also enters the Waiting state (waiting on A) yielding a cyclic waiting-dependency.
  - Priority Inversion
    Solution:
    Priority Inversion requires at least 3 Processes and 1 Resources
    Priority inversion occurs when a lower priority Task indirectly delays completion of a higher priority Task. For example this occurs if a Task C starts execution (only one ready) and acquires Ressource 1, and is then pre-empted by the highest priority Task A. Task A then needs ressource 1 and enters the Waiting state (waiting on C). Since Task B which has a higher priority than C is now also ready and is executed now. However since C cannot continue to release Ressource 1, this also delays Task A, which would’ve been able to continue and finish sooner, if C would be executed instead of B.

Exercise 3: Real Time and Resources
Given are four tasks that are all executed only once.

Start denotes at which point in time the task enters the ready state.

Run denotes how many time units the task whiches to run without doing any requests.

Req denotes that a tasks requests exclusive access to a system resource.

T denotes that a tasks terminates releasing all resources.

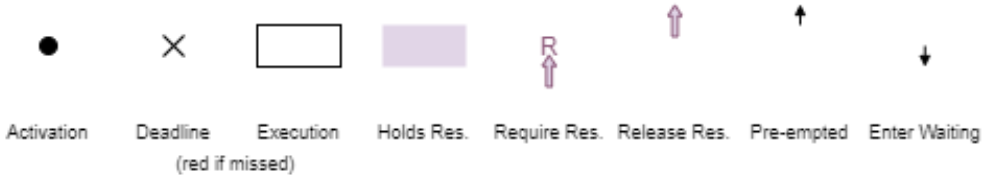
DL denotes the absolute deadline, i.e., the point in time when the computation must be finished.

Schedule these tasks (sorted by prioriy; first task has highest priority)

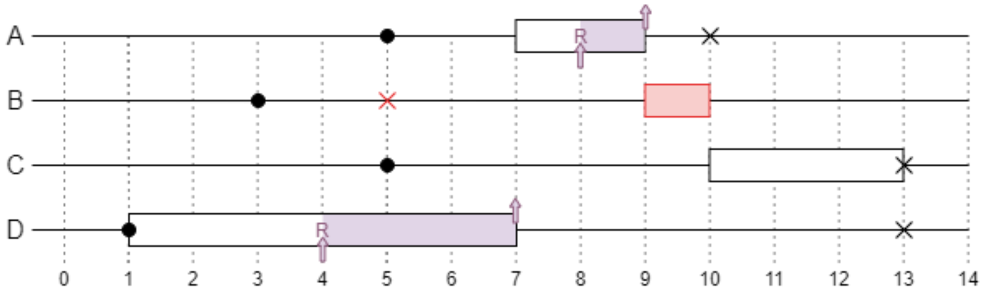
Task	**Execution**	**Deadline (absolute)**
Task A	Start @ 5 | runs 1 | Req. | runs 1 | T	DL @ 10
Task B	Start @ 3 | runs 1 | T	DL @ 5
Task C	Start @ 5 | runs 3 | T	DL @ 13
Task D	Start @ 1 | runs 3 | Req. | runs 3| T	DL @ 13

using

Solution Legend:

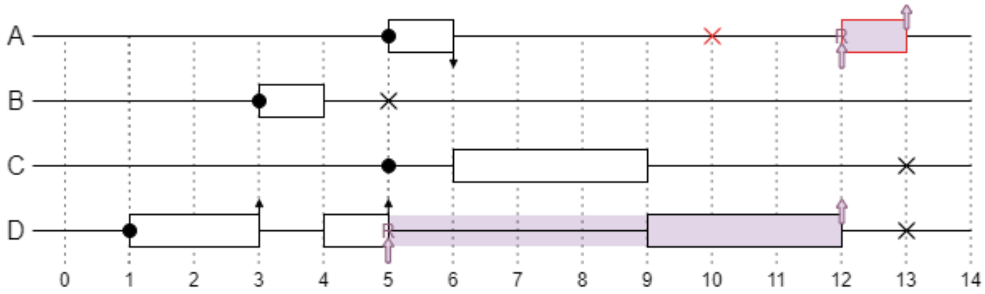


- Cooperative scheduling
  Solution:



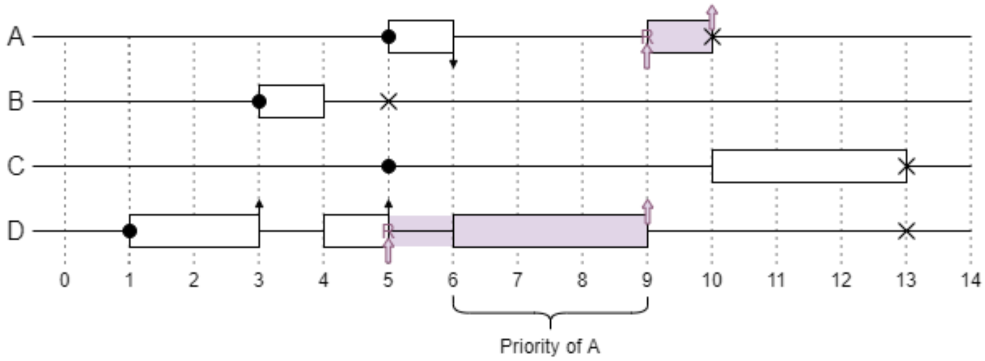
Since this scheduling is non-preemptive, the deadline of Task B is missed.

- Preemptive scheduling
  Solution:



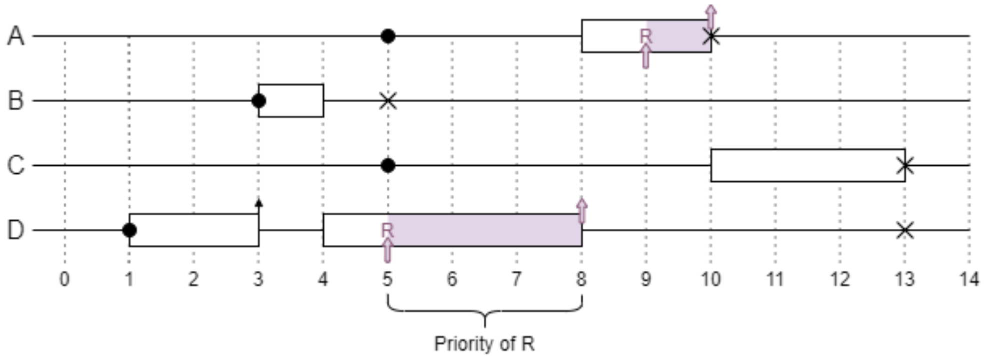
Here we see an example of priority inversion between A and C, caused by D acquiring R before being pre-empted. This causes Task A to miss its deadline.

- with priority inheritance protocol
  Solution:



Here we see an examplary algorithm, that fixes priority inversion. No deadline is missed.

- with priority ceiling protocol
  Solution:

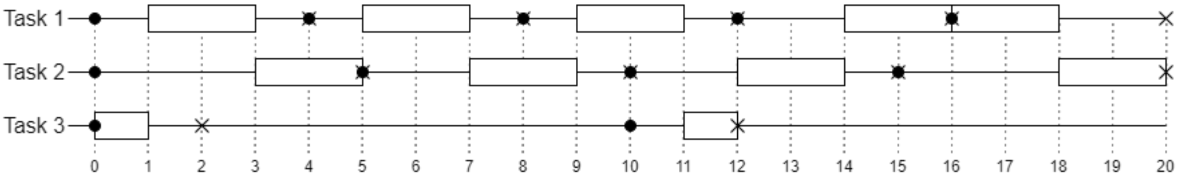


This is a different algorithm that prevents priority inversion. (It also prevents deadlocks)

Exercise 4: Periodic Scheduling
Use earliest deadline first to schedule this task system:

$$(4, 2, 4), (5, 2, 5), (10, 1, 2)$$

Solution:



Since the least common multiple of 4, 5 and 10 is 20 we only need to schedule the first 20 time steps as after that the scheduling would just repeat.

Why is the following task system not schedulable?

$$(3, 2, 2), (6, 2, 7), (10, 3, 10)$$

Solution:
We calculate the Utilization:
$U = \frac{2}{3} + \frac{2}{6} + \frac{3}{10} = 1.3$
Since Utilization is above 1, the task system is not schedulable.

Aachen, June, 2023
SWS: V3/Ü1, ECTS: 6
Professor Dr.-Ing. Stefan Kowalewski
Simon Fonck, M.Sc. RWTH
Alexander Kruschewsky, M.Sc. RWTH

Exercise for

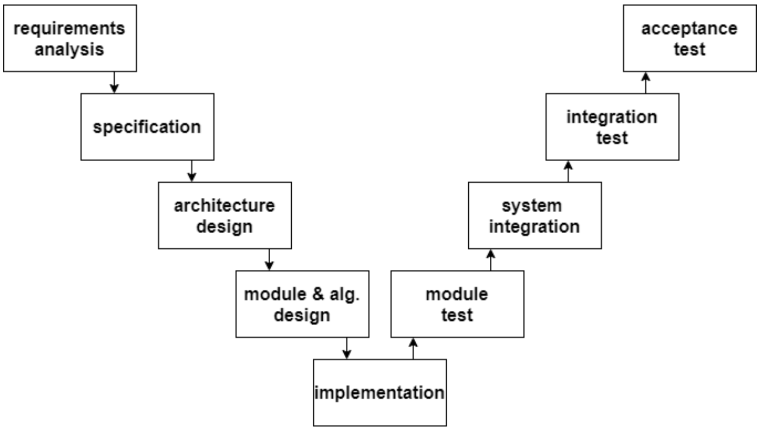
Embedded Systems

Summer Term 2023

Sheet 5: Embedded Software Dev. & Design

Exercise 1: V-Model
Name the missing steps of the V-Model, denoted by A-E in the following graphic:

Solution:



For the remaining three exercises imagine the following scenario:
You are a startup that wants to manufacture an innovative luxury car. However, you are limited in budget and thus cannot afford an assembly line for the production of your vehicle as well as the required parts. Since you firmly believe in the success of your car and want to be able to scale easily with higher demand you start looking for alternative production line designs (Referred to as ”System” further on).

Exercise 2: Types of Requirements
To find the optimal design for your production you start by writing down requirements for it. This yields the following ten requirements.
For each requirement decide whether they are functional or non-functional. If they are non-functional, choose the correct category from the quality tree. (A copy of the quality tree is attached to the end of this exercise)

- The System must fit in an area of 20x20m.
  Solution: **Mounting Space**
  As the name suggests, this is a requirement-type limiting the physical size of the system.

- The Assembly-System should not require an up-front payment higher than 3 million$.
  Solution: **Cost**
  This requirement-type limits the monetary but also labor spendatures for the creation of the system.

- Switching the production to a different vehicle model should take less than 24 manhours and no additional capital investment.
  Solution: **Configurability**
  Differentiating between the different sub-categories of modifiability is not as easy. Most importantly ”configurability” is about the ability of the system to meet changed requirements by a user, so also without physical changes.

- Increasing the Systems capacity twofold should not exceed 100 manhours.
  Solution: **Scalability**
  Scalability is how well the system can change the ”amount of its output”, be it the production rate of an assembly line, the power of an engine or the computing-performance of a CPU.

- Reducing the Systems footprint to an area of 10x10m at the cost of a 60% lower production rate should be doable with less than 100 thousand$ in capital investment.
  Solution: **Adaptability**
  This type is very similar to configurability. The difference being, that Adaptability needs some sort of re-development and usually includes physical changes to the system, while configurability doesn’t.

- The Assembly of a car chassis should take at most 60 minutes.
  Solution: **Performance**
  Requirements measuring the amount or speed of a systems output fall under the category of performance.

- The System must be able to produce parts out of Titanium and Aluminium.
  Solution: **Functional**
  This requirement covers ”what” the system does, not ”how good”.

- The incorporation of additional metal alloys into your production line should be doable in less than 80 manhours.
  Solution: **Extendability**
  This type is about the systems ability to incorparate new requirements.

- The assembly should have tolerances of less than 0.05 millimeters.
  Solution: **Reliability**
  This type is about the systems outputs being as expected / within a tolerance of the perfect result.

- The System must be able to assemble a chassis, all mechanical parts and all body panels of a car.
  Solution: **Functional**
  This requirement also covers ”what” the system does and not ”how good”.

Exercise 3: Good Requirements
A big automative manufacturer is interested in your development and is willing to fund part of its development if the system will also meet their requirements. You ask the manufacturer to send you these and receive the following ten requirements.
For each requirement decide whether there are issues with them. Use the ”Basic requirement issues” discussed in the lecture as a guideline. If there is an issue with a requirement choose one of the following reasons:
(The requirement proposes a solution / the requirement is not checkable / the requirement is not understandable by an average programmer).

- The Assembly-System should use 4 robotic arms.
  Solution: **Proposes Solution**
  This requirement answers the question of ”how” and thus proposes a solution.

- The System should be able to produce parts meeting the ISO 134866-1:1999 standard.
  Solution: **Not Understandable**
  The average developer is not expected to know and be able to work with automotive ISO requirements.

- The System should be able to adapt to other metals.
  Solution: **Not Checkable**
  This requirement is missing time/cost limits. Without those the requirement would always be fulfillable as you could just develop an entirely new system.

- The System should be able to be sold as a product.
  Solution: **Not Checkable**
  What defines the ability to be sold as a product? Technically you could sell anything.

- The Systems operation should not require human interaction 99.99% of time.
  Solution: **Good**
  This is an ”Availability”-type requirement. It is checkable, understandable and not a solution.

- The metal parts should be 3D-printed.
  Solution: **Proposes Solution**
  This requirement also answers the question of ”how” and thus proposes a solution.

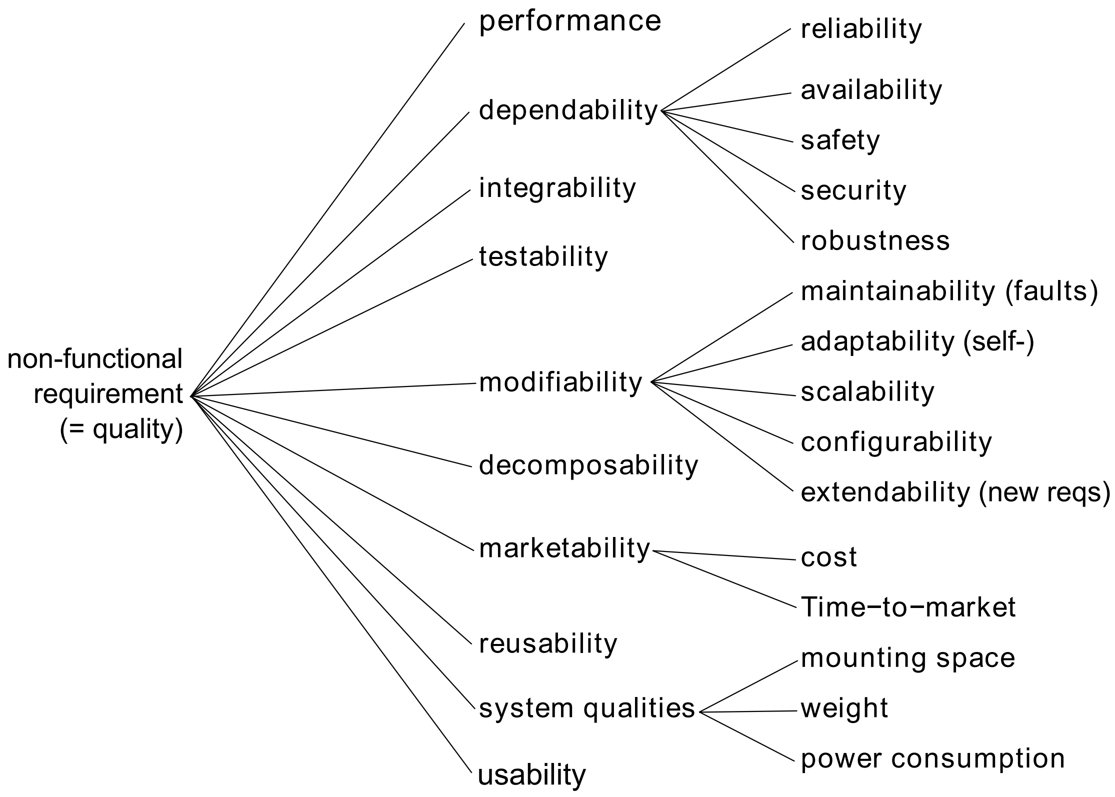
- The Levy should be less than half of the liquid assets available.
  Solution: **Not Understandable**
  The average developer is not expected to know business-slang like ”levy” meaning something like tax or liability. Additionally it also doesn’t specify what for this tax would be, or how high the ”liquid assets” are.

- Production should take less than a week.
  Solution: **Not Checkable**
  This requirement doesn’t specify the production of what should take less than a week. A bolt, a door panel or the whole chassis?

- The parts should be assembled in the center of the machine, one piece at a time.
  Solution: **Proposes Solution**
  This requirement also answers the question of ”how” and thus proposes a solution.

- The System should be operational in temperatures between -10°C and 50°C as well as humidities between 10% and 60%.
  Solution: **Good**
  This is a ”Robustness”-type requirement (Robustness is the ability to operate in changed outside circumstances). It is checkable, understandable and not a solution.

Qualitytree:


non-functional
requirement
(= quality)
performance
dependability
integrability
testability
modifiability
decomposability
marketability
reusability
system qualities
usability
reliability
availability
safety
security
robustness
maintainability (faults)
adaptability (self-)
scalability
configurability
extendability (new reqs)
cost
Time−to−market
mounting space
weight
power consumption