Vorlesungsskript zu „Vertiefung Programmieren“ Programmentwurf

Version 0.2

Dozent: Dipl.-Inf. (FH) Andreas Schmidt

Aufgabenstellung

Entwickeln Sie entsprechend den Anforderungen die Software für das „Underground Mining Monitoring System“. Hierbei basiert die Entwicklung auf dem Nucleo-Board mit STM32G474 Controller und dem PiEye EduShield Aufsatz.

Als Software Basis dient das bereitgestellte VPTemplate Projekt. Das VPTemplate stellt die HAL (von ST) sowie eine zusätzliche HAL (basierend auf PiEye EduShield) bereit.

Für den Programmentwurf sind folgende Artefakte als ZIP Archiv abzugeben

- Quelltext bzw. gesamtes Projektverzeichnis basierend auf dem VPTemplate Projekt (übersetzbar unter Linux mit dem mitgelieferten Makefile)
- Test-Spezifikation und Test-Protokoll für das System (ein Dokument, welches die Testfälle und die Ergebnisse beinhaltet)
  - Die Test-Spezifikation beschreibt Test-Fälle (min. 10 Testfälle) mit einer Beschreibung der einzelnen Test-Schritte, um die Funktionalitäten entsprechend den Anforderungen zu testen
  - Das Test-Protokoll dokumentiert die Ergebnisse eines durchgeführten Tests entsprechend der Test-Spezifikation. Hierbei gibt es als Ergebnis für jeden Testfall: Passed, Conditionally Passed, Failed

Alle Dokumente sind in Englisch zu verfassen. Ebenso muss jedes Dokument ein Titelblatt (mit Namen und Matrikelnummer), ein Inhaltsverzeichnis sowie eine kurze Einführung enthalten.

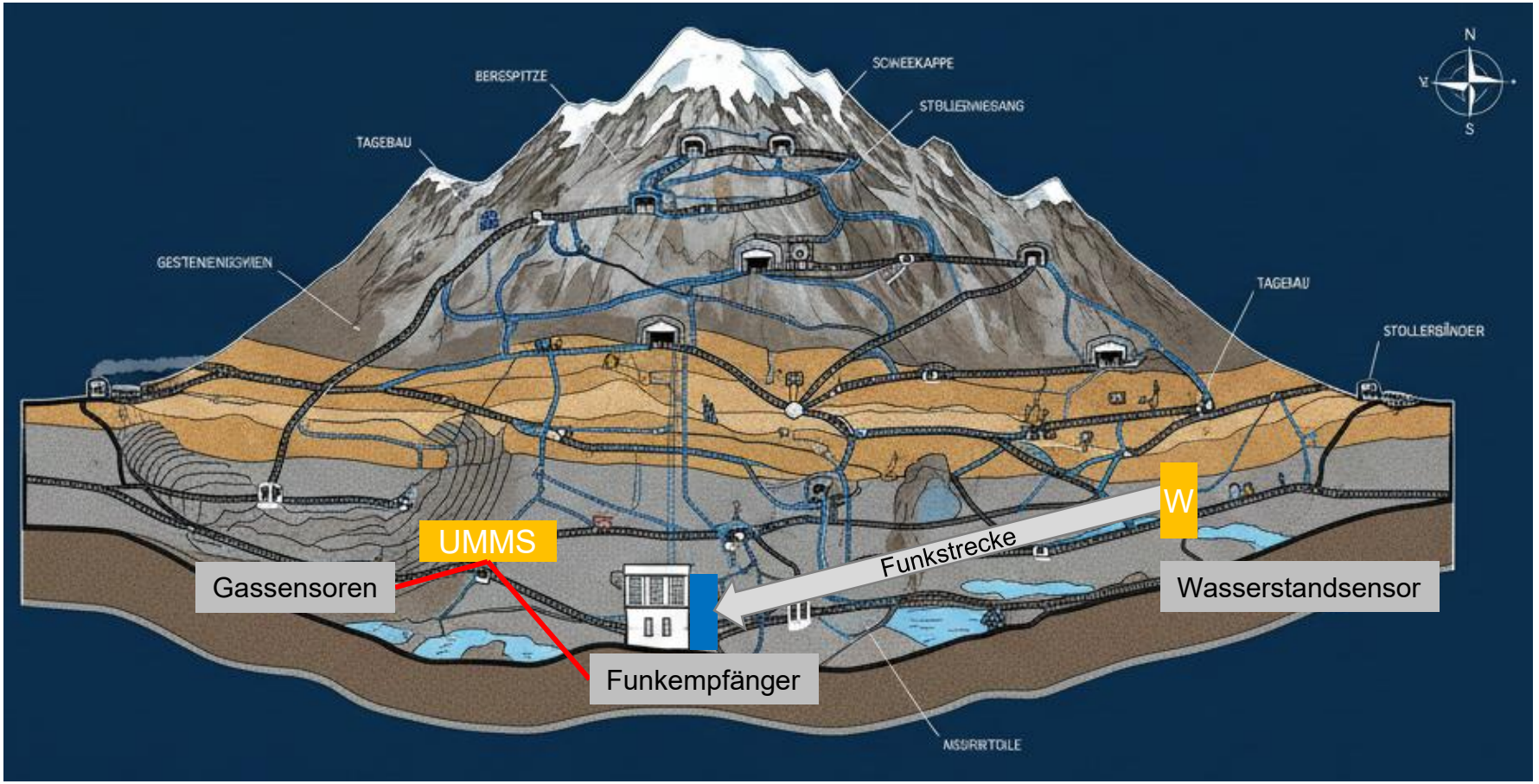
Die Dokumente können auch Bilder/Fotos der Test-Schritte und/oder Test-Ergebnisse erhalten. Diese Bilder sollten aber im Text kurz beschrieben werden. Insbesondere wenn Fotos als Test-Ergebnis Nachweis verwendet werden.

Quelle: intern

Underground **M**ining **M**onitoring **S**ystem

Underground Mining Monitoring System

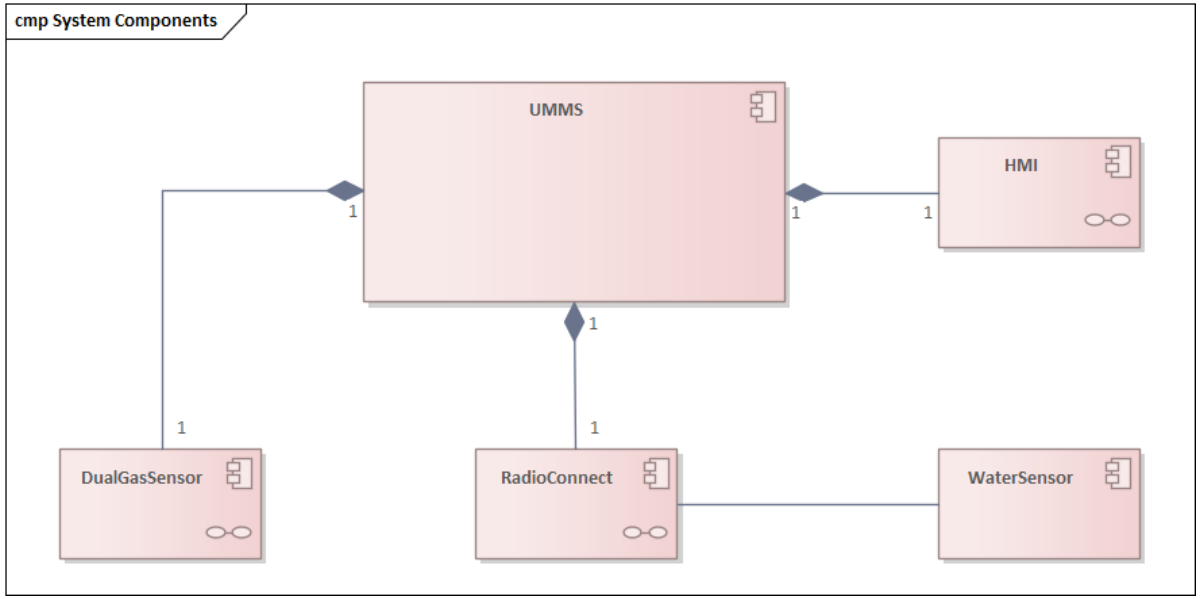
System Context



Quelle: intern

Underground Mining Monitoring System

Main System Components

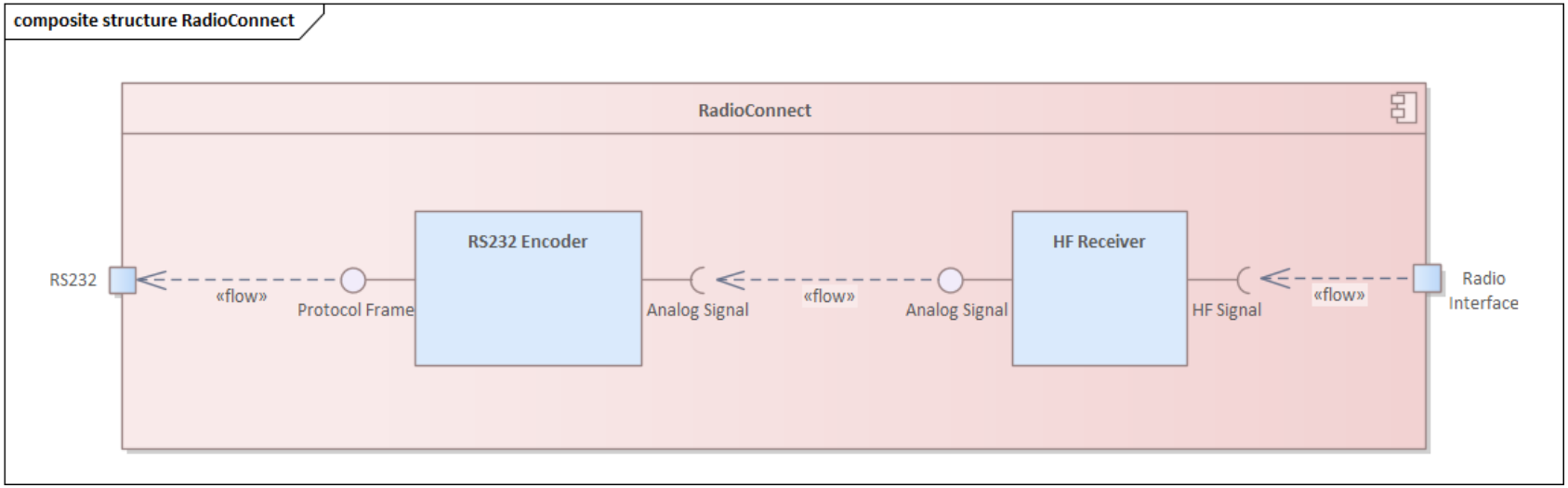


UMMS is the main system component which consists of

- **HMI** – User Interface with Buttons, LEDs and Display
- **DualGasSensor** – Physical Gas-Detection Sensor (redundant)
- **RadioConnect** – RS232 based component to provide WaterSensor data
- **WaterSensor** – Remote component which transmits sensor data via HF

Quelle: intern

Main System Components – Radio Connect

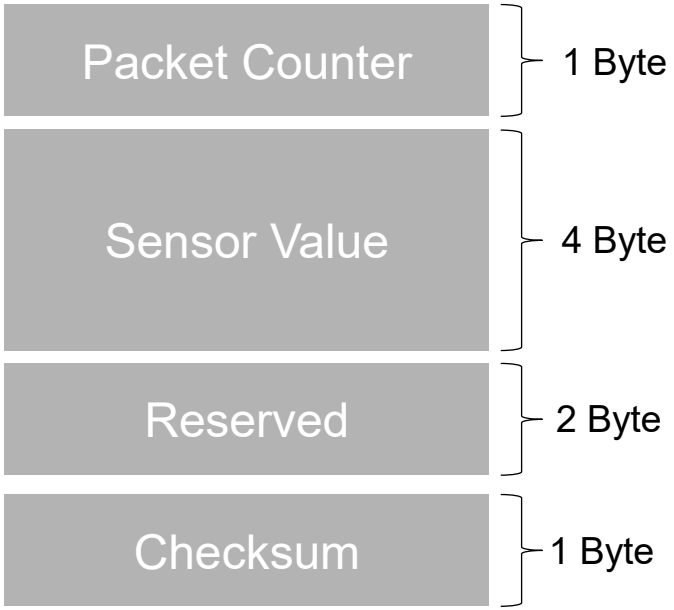


The **RadioConnect** component provides a *HF Receiver* which receives the HF signal of the *WaterSensor* and demodulates it. Hereby, the *HF Receiver* provides an analog voltage which is equivalent to the original *WaterSensor* voltage.

This analog voltage is digitalized by the *RS232 Encoder* and packed in a special data frame and transmitted via a RS232 connection. Hereby, additional information are added to the data frame and transmitted cyclically every 50ms.

Quelle: intern

Main System Components – Radio Connect



Data Frame Format

The data frame consist of 8 data bytes.

Packet Counter – Unsigned 1 Byte value with a counter value of the current transmitted packet

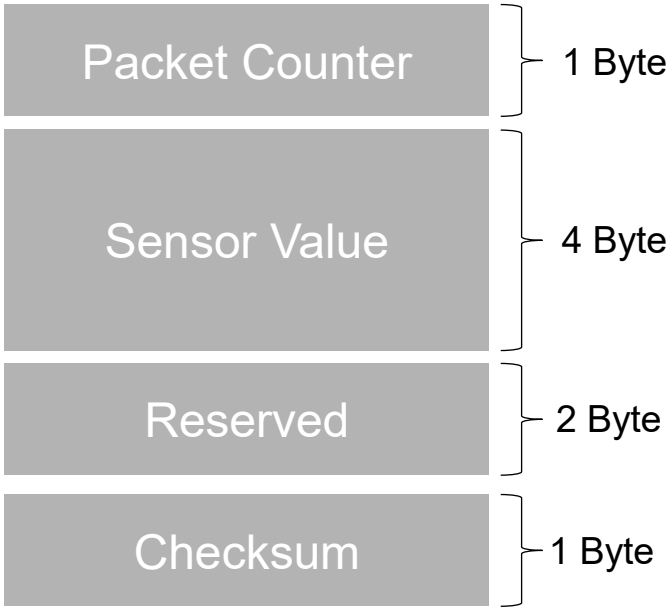
Sensor Value – Unsigned 4 Byte value (little endian) containing the sensor analog voltage value in µV

Reserved – 2 Byte reserved value should be always 0xC0DE

Checksum – Singed 1 Byte value for the checksum of the complete data frame (without the checksum field)

Quelle: intern

Main System Components – Radio Connect



Data Frame Checksum

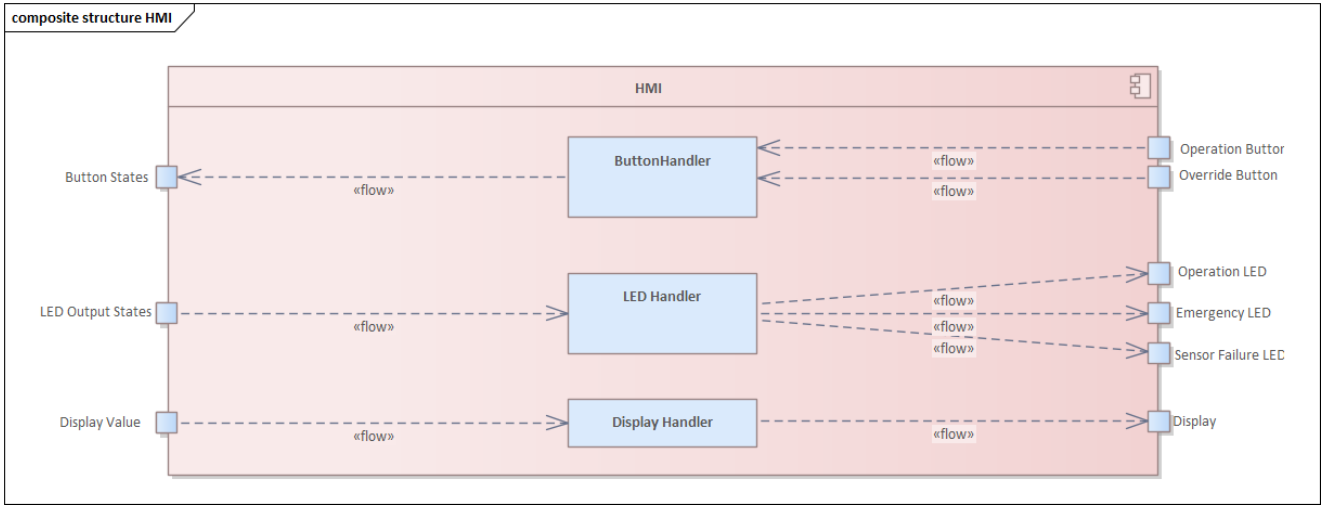
The checksum is calculated from the entire data set, excluding the checksum itself. The data set is summed byte by byte, the least significant byte is taken from the sum, and its two's complement is then calculated.

The two's complement is calculated by inverting the bits of the least significant byte and then adding 1. This can be achieved, for example, by using an exclusive-OR operation with 0xFF and adding 0x01. Thus, 0x00 remains unchanged, 0x01 becomes 0xFF, and so on.

In binary, the two's complement represents a negative number. Since the checksum is therefore the negative sum of the remaining bytes, checking a data set for errors is very simple. One simply sums the individual bytes of a data set, *including* the checksum, and obtains 0x00 as the least significant byte if the data set is correct.

Quelle: intern

Main System Components – HMI

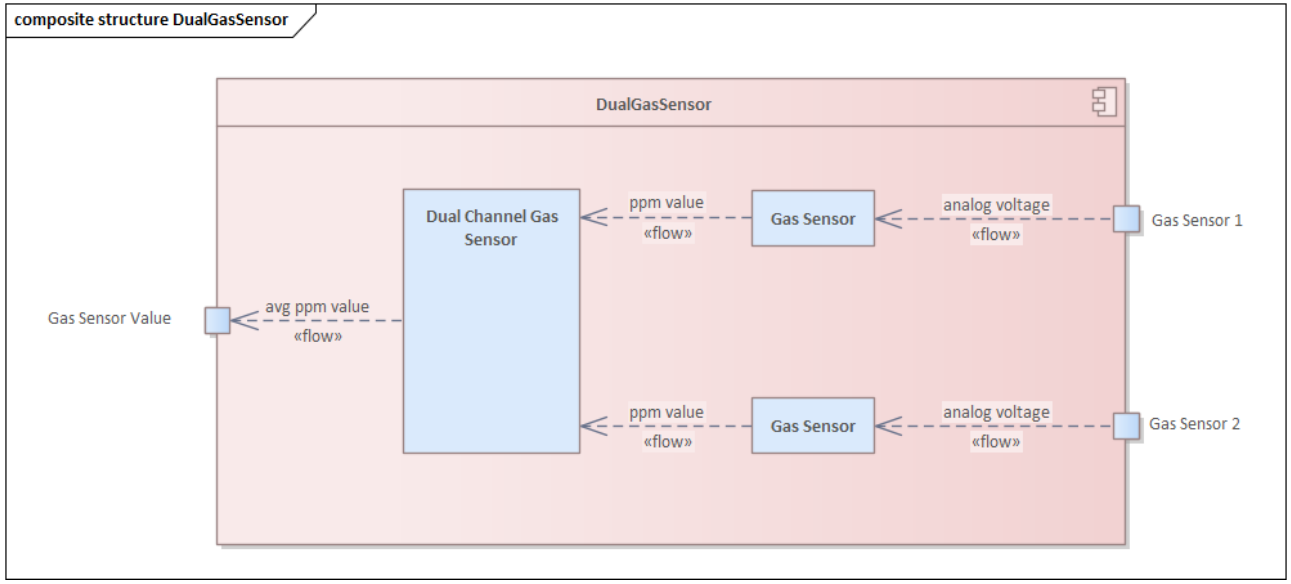


The HMI component is responsible to handle all user interaction with input buttons and indicators like LEDs and 7-segment displays.

From a design perspective, the HMI component should handle the interaction with the corresponding HW components (buttons, LEDs, display) in a centralized way and provide defined interfaces to get and set the logical values for the actual HMI elements.

Quelle: intern

Main System Components – DualChannelGasSensor



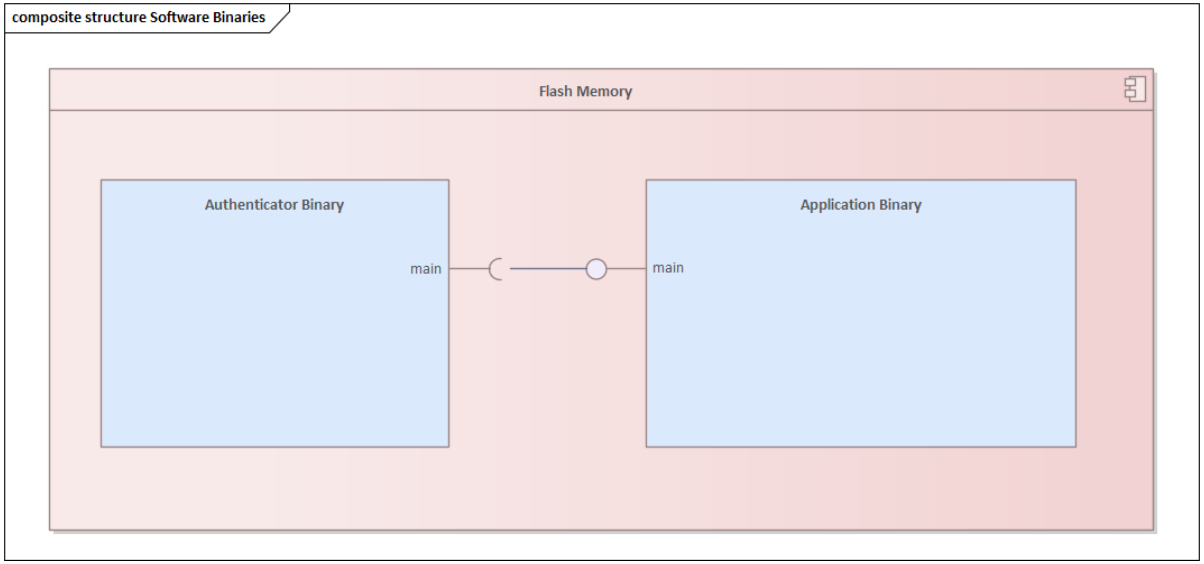
The DualChannelGasSensor component is responsible to read two independent analog gas sensors, convert the analog sensor voltage to a physical measurement in ppm and provide an average ppm value of the two sensors.

Additionally, this component is responsible for checking the sensors for any failures. This includes also an inconsistency between the two sensor values or out of range measurements.

Quelle: intern

Underground Mining Monitoring System

Software Concept



The overall software is split into two different software binaries.

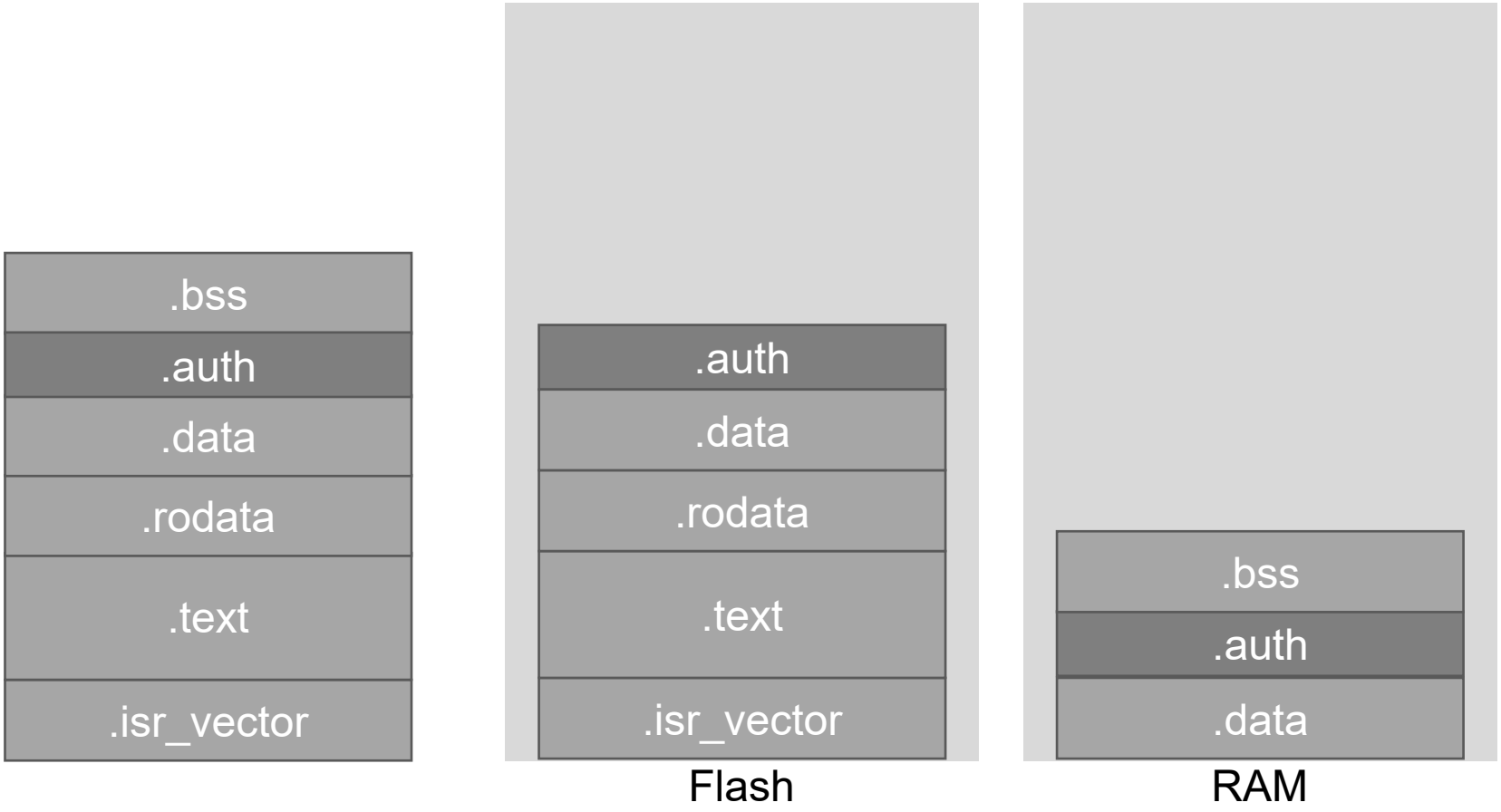
The **Authenticator Binary** is responsible to authenticate the system by receiving a decryption key, decrypting a special software part with the key and checking whether the decrypted sections is correct, the authenticator starts the actual application.

The **Application Binary** contains the actual functionality of the UMMS.
Both binaries reside in different flash memory areas.

Quelle: intern

Underground Mining Monitoring System

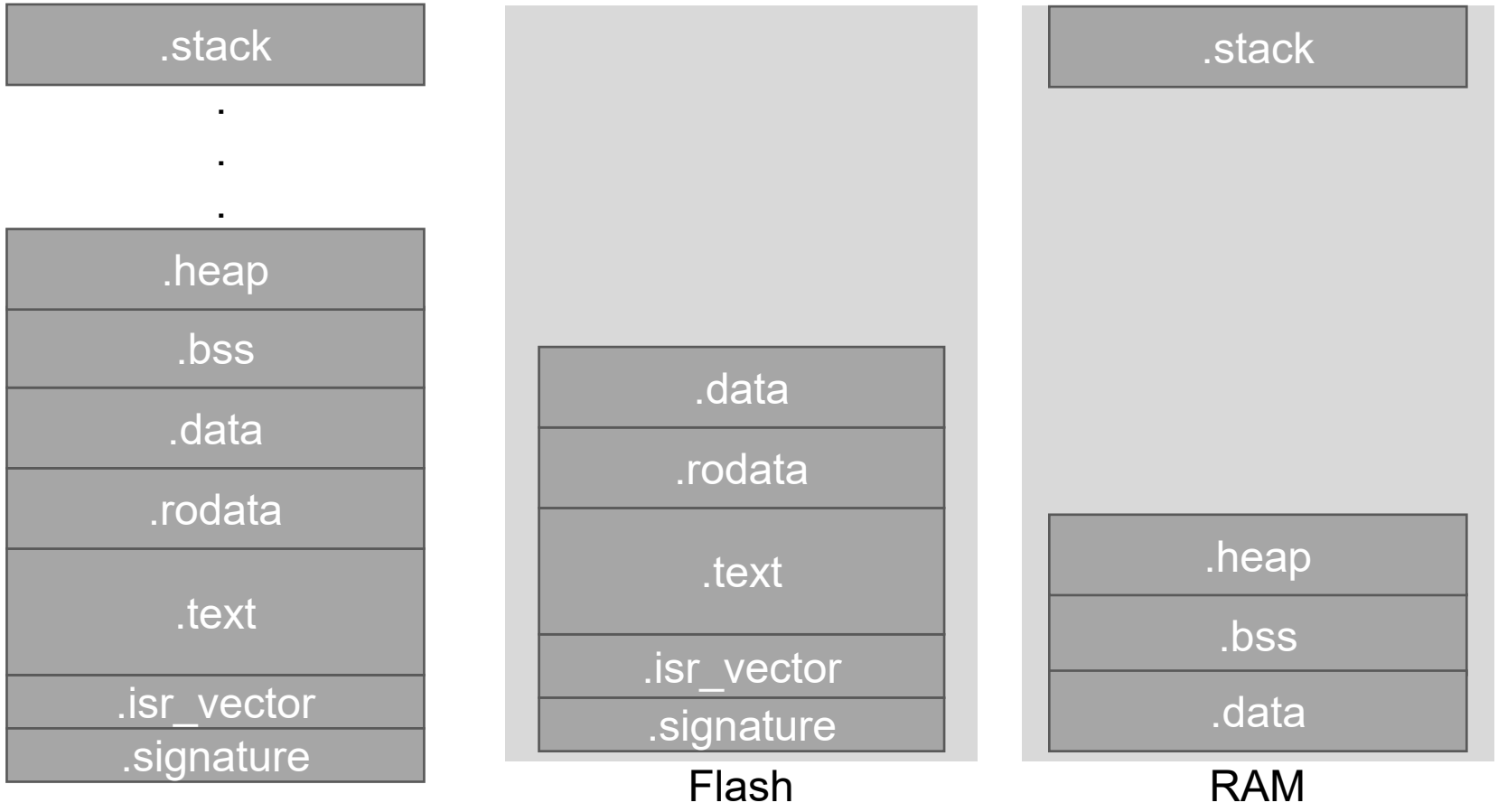
Software Concept – Binary Sections Authenticator



Quelle: intern

Underground Mining Monitoring System

Software Concept – Binary Sections Application



Quelle: intern

Software Concept – Memory Map

Component	Flash Address	Size
Authenticator	0x08000000	64 kiB
Application	0x08010000	128 kiB

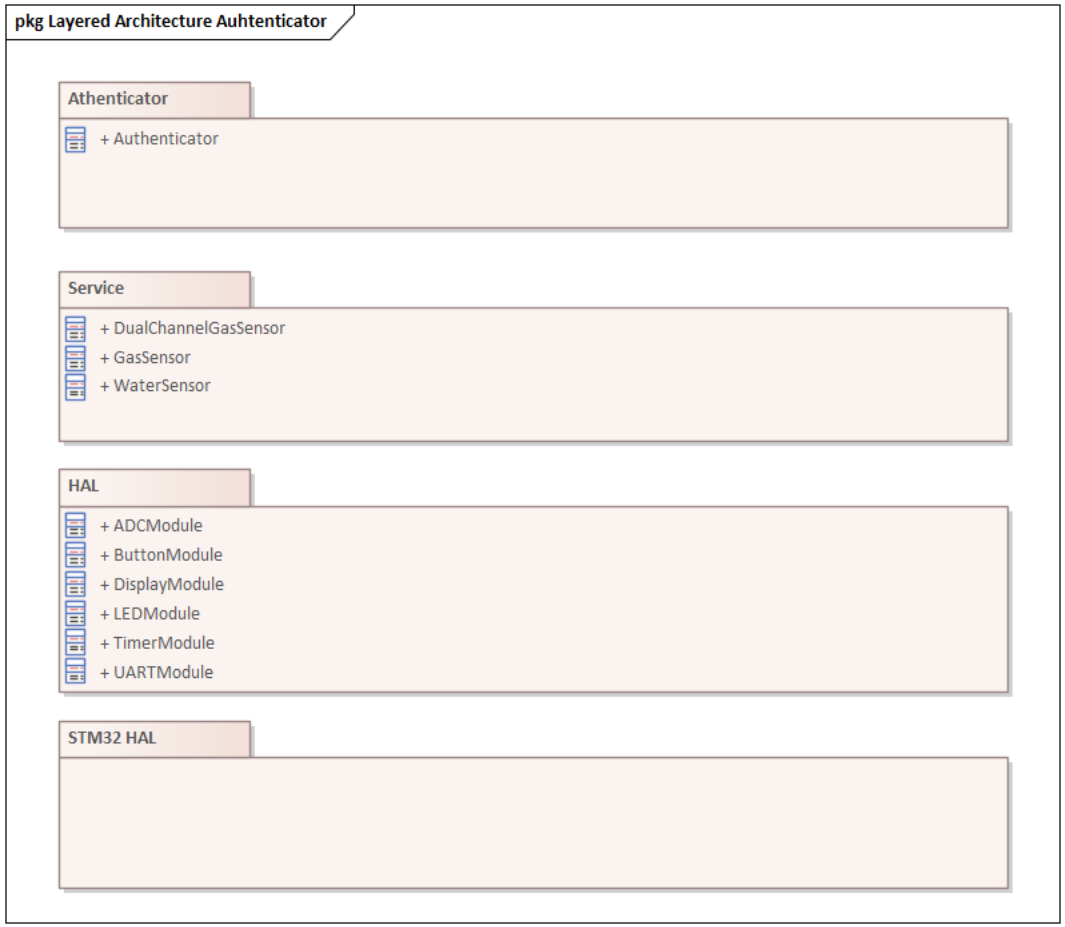
Component	RAM Address	Size
Authenticator	0x20000000	16 kiB
Authenticator Stack (First Address)	End of RAM	
Application	0x20004000	64 kiB
Application Stack (First Address)	0x20014000	4 kiB

The Authenticator doesn‘t use a dedicated stack section. It must only initialize the stack pointer to the correct memory address. Therefore, there is no size constraint for the stack of the Authenticator

The .signature section only contains ‘UMMS’ as signature bytes.

Quelle: intern

Basic Software - Authenticator



The basic software architecture is based on a layer architecture including HAL, Service and Application layer.

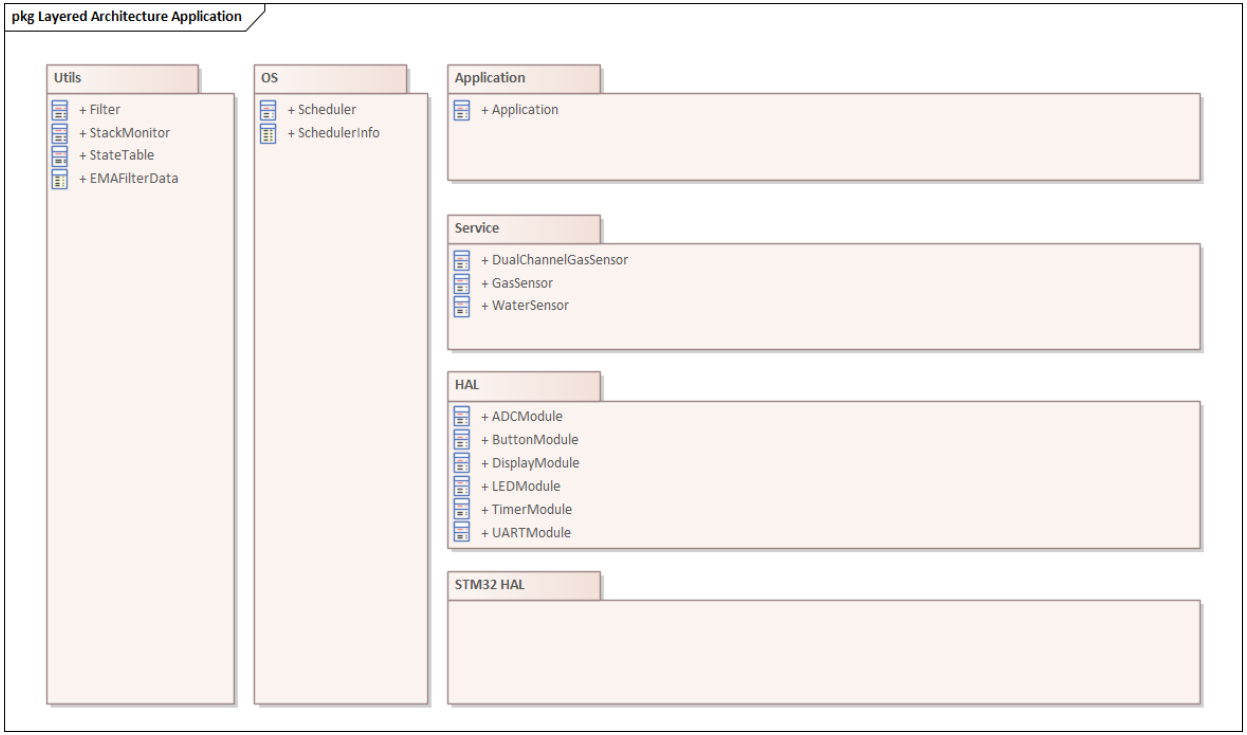
The specific microcontroller abstraction is based on the STM32 HAL Library provided by ST

The Authenticator software does not use any OS module (like the Scheduler) and contains only a simple main() function to receive the decryption key, decrypt the starter code and start the application binary.

Quelle: intern

Underground Mining Monitoring System

Basic Software - Application



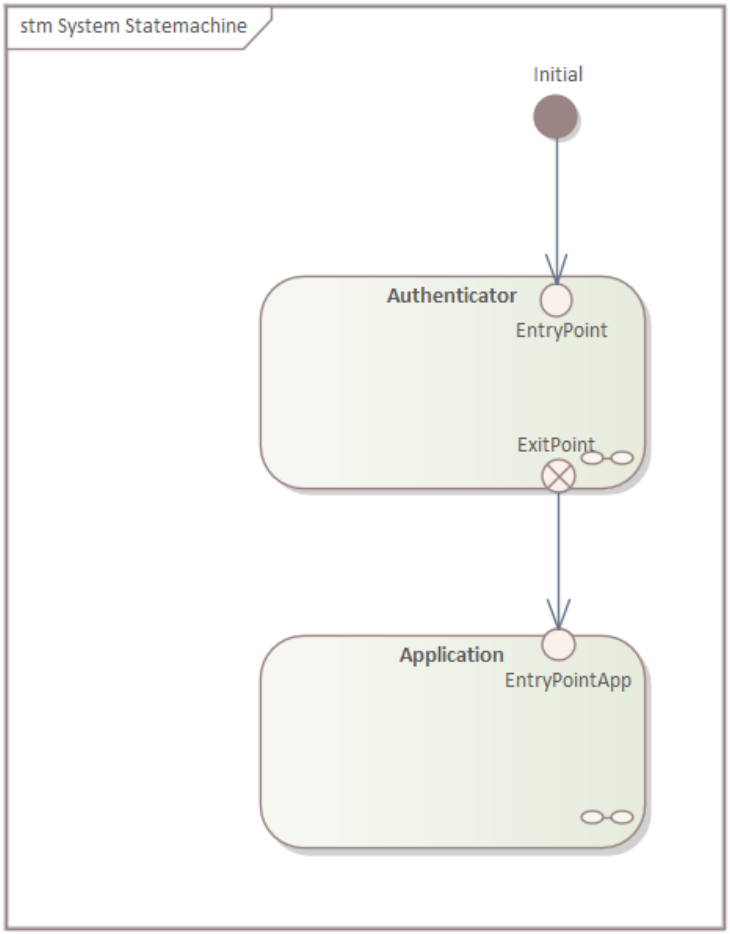
The software is based on a layer architecture including HAL, Service and Application layer.

The specific microcontroller abstraction is based on the STM32 HAL Library provided by ST

The Application contains the main UMMS logic and uses all shown packages to realize the functionality.

Quelle: intern

State Machines – System State Machine

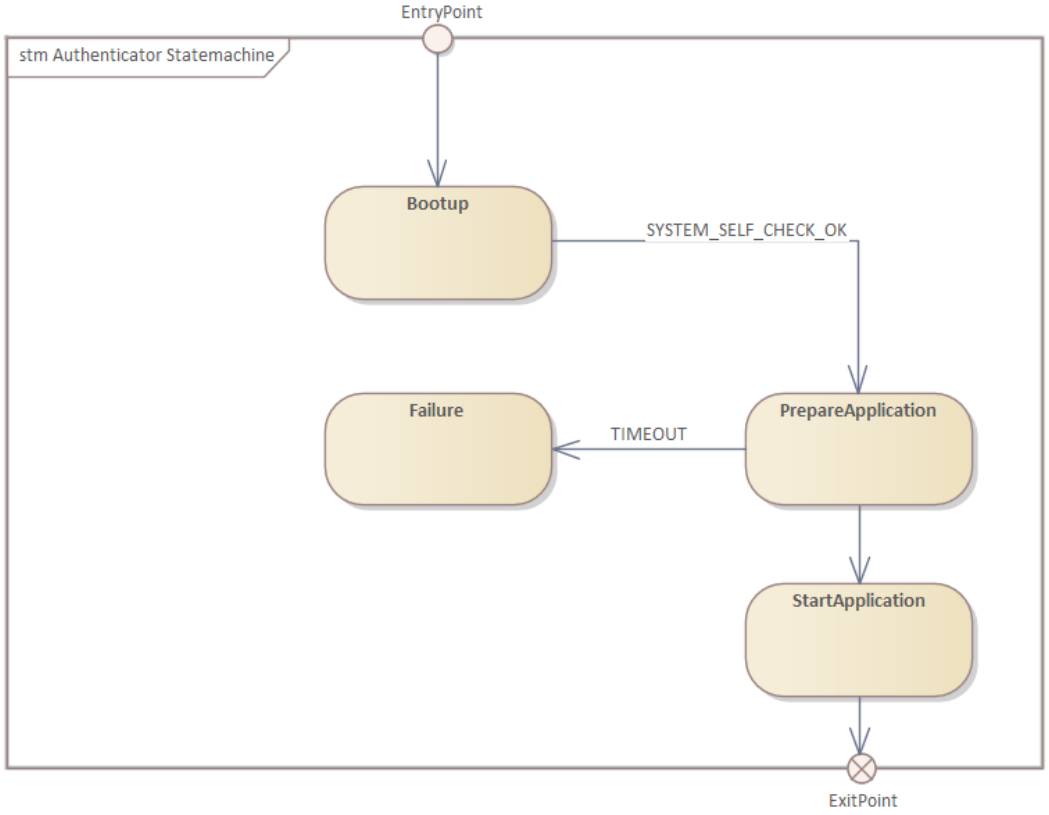


The System State Machine is based on two separate sub-state machines. Each sub-state machine covers an explicit application (Authenticator and Application).

As it is shown in the State Diagram, the system starts (Reset or POR) by entering the Authenticator state machine. This state machine perform the authentication process and if this process was successful, it starts the UMMS Application which contains the shown Application state machine

Quelle: intern

State Machines – Authenticator State Machine



The Authenticator State Machine contains four different states

Bootup: State to cover the bootup process including HW initialization and preparation

Failure: State to cover a system failure during the decryption key receive process (Timeout)

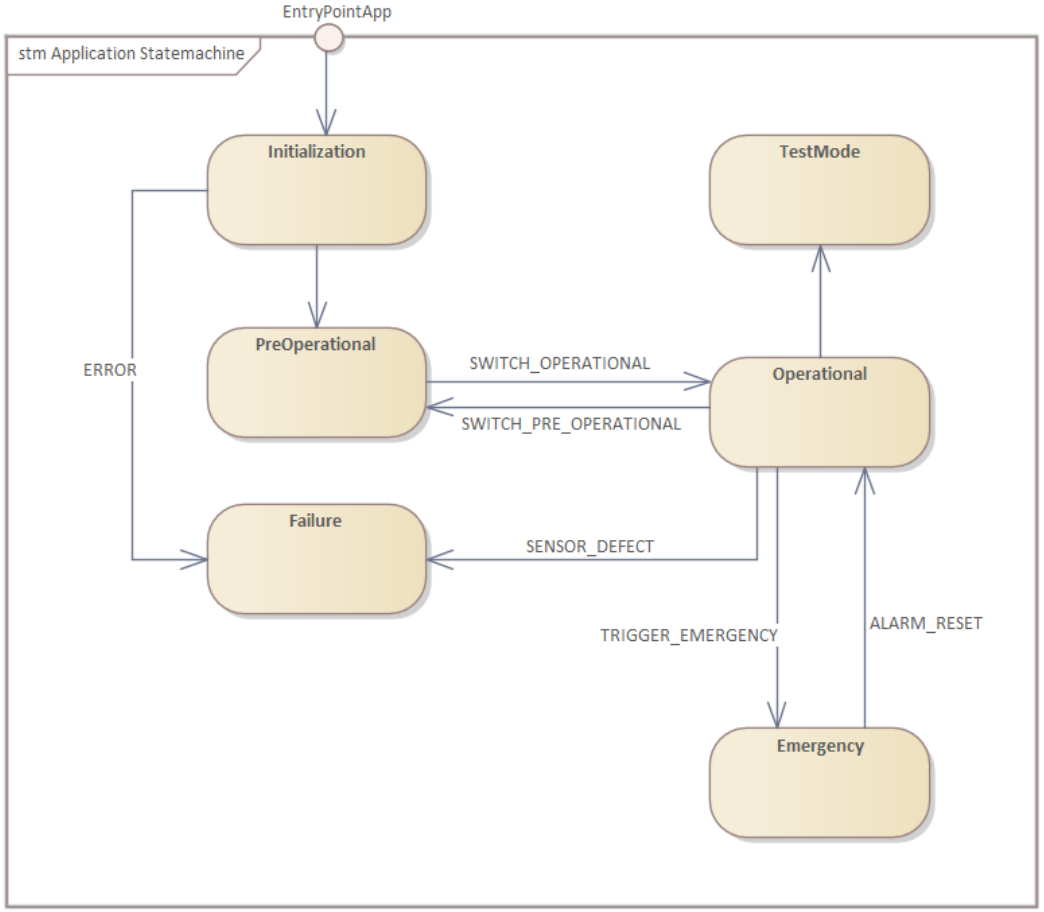
PrepareApplication: State to cover the actual implementation of the Authenticator process including copy of start code and decryption of start code

StartApplication: State to cover the process of starting the application.

Quelle: intern

Underground Mining Monitoring System

State Machines – Application State Machine



Initialization: State to cover the initialization of HW and application.

PreOperational: State to cover the pre-operational state

Failure: State to cover a system failure either during HW initialization or any detected sensor defect

Operational: State to cover the activated monitoring system.

Emergency: State to realize an emergency as soon as the parameterized threshold values for the sensors is exceeded.

TestMode: State to cover a manual test of the alarm system

Quelle: intern

Underground Mining Monitoring System

Authenticator Hardware Mapping

LED D0:	On = Authenticator active, Off = Authenticator not active
LED D1:	Off = Waiting for key, On = 10s Timeout reached
LED D2:	Off = No final timeout, Flashing = 30s Timeout reached
LED D3:	Unused
LED D4:	Unused

Indicators Failure State:

LED D0:	Unused
LED D1:	Unused
LED D2:	Unused
LED D3:	Unused
LED D4:	On = System Failure

Quelle: intern

Application Hardware Mapping

Button Mapping:

Switch Pre-Operational/Operational:	SW1
Activate TestMode:	SW2
Alarm Reset:	B1

Quelle: intern

Underground Mining Monitoring System

Application Hardware Mapping

Indicators Normal Operation:

7-Seg Display S1:	Water-Level (cm) 10^2
7-Seg Display S2:	Water-Level (cm) 10^1
LED D0:	On = Operation Mode, Off = Pre-Operation Mode
LED D1:	Off = No Alarm, On = Warning, Flashing = Emergency
LED D2:	Off = System OK
LED D3:	On = TestMode, Off = No TestMode
LED D4:	On = Sensor Failure, Off = Sensor OK

Indicators Failure State:

LED D0:	Off = No Operation Mode
LED D1:	Unused
LED D2:	On = System Failure
LED D3:	Unused
LED D4:	On = Sensor Failure, Off = Sensor OK

Quelle: intern

Application Hardware Mapping

Sensors:

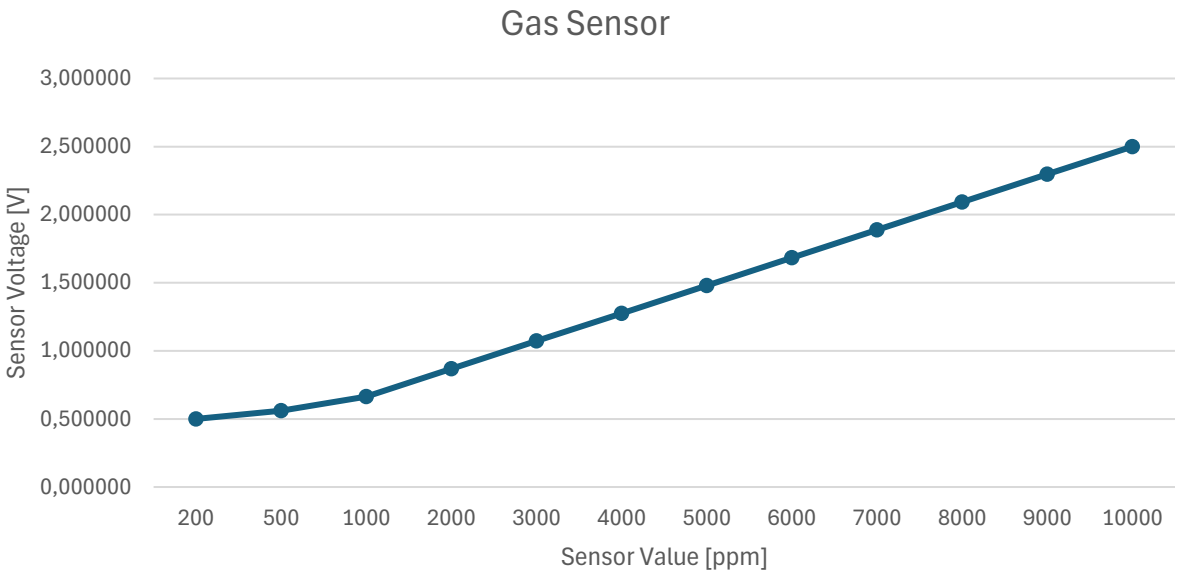
Gas Sensor 1:	POT1
Gas Sensor 2:	POT2
WaterSensor:	UART

Quelle: intern

Sensor Details: Gas Sensor

Measurement:

Measurement Range:	200ppm – 10000ppm
Output Signal:	0,5V – 2,5V
Sensor Defect Signal High:	> 2,5V
Sensor Defect Signal Low:	< 0,5V



Total-Range: 9800 ppm

1ppm = 0,000204082 V = 204,082 µV

Linear correlation between ppm and output voltage

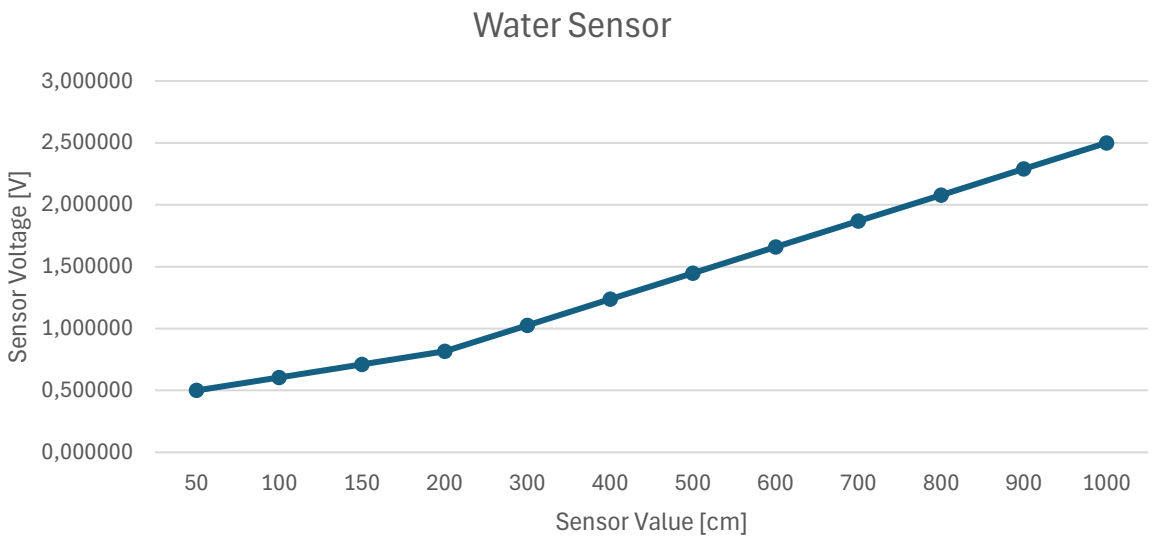
Quelle: intern

Underground Mining Monitoring System

Sensor Details: Water Sensor

Measurement:

Measurement Range:	50cm – 1000cm
Output Signal:	0,5V – 2,5V
Sensor Defect Signal High:	> 2,5V
Sensor Defect Signal Low:	< 0,5V



Total-Range: 950 cm

1cm = 0,002105263 V = 2105,263 µV = 2,105263 mV

Linear correlation between water level and output voltage

Quelle: intern

Functionality Authenticator

After power-up, the Authenticator must initialize the needed HW components. This includes the following peripherals:

- UART to receive decryption key
- LEDs to show status information

As soon as the HW initialization has finished and no errors have been detected, the Authenticator should enter the PrepareApplication state.

The PrepareApplication state should wait for an ‘A’ character via the UART interface. This ‘A’ character initiates the receive process of the decryption key.

- If the ‘A’ character is not received within 15s, the system shall switch to the Failure state.

If the ‘A’ character is received, the Authenticator expects to receive a decryption key with a maximum length of 8 bytes via the UART interface. Hereby, the termination of the decryption key is the ‘\n’ character.

- If the decryption key hasn’t been completely received after 10s, the Authenticator should signalize the first timeout stage by turning on the LED D1.
- If the decryption key hasn’t been completely received after 30s, the Authenticator should signalize the second timeout stage by flashing the LED D1.
- If the decryption key hasn’t been completely received after 45s, the Authenticator should switch to the failure state.

Leaving the Failure state is only via a reset possible.

Quelle: intern

Functionality Authenticator

If the Authenticator has successfully received the decryption key, it should copy the .auth section from the flash memory to the RAM memory section. Hereby, the symbols provided by the Linker should be used.

The .auth section only contains a single function verify() which checks a signature byte of the application and calls a function of the application.

Remark: The function to start the application must be called via a memory address and not by a typical function call. Hereby, it is recommended to define a function pointer type, create a variable of that function pointer, set the function pointer variable to the correct address of the StartHandler() function of the application and then call the function via the function pointer.

Remark: The address of the StartHandler() function in the application binary must be provided hard-coded to the Authenticator. Since the Application has a fixed address in flash memory, this address can be used to enter the application respectively call the StartHandler().

Remark: The StartHandler() function of the application must remap the Vector Table to its own vector table and also run the most part of the reset handler code to initialize the .data and .bss segment of the application.

Quelle: intern

Functionality Authenticator

If the copy process of the .auth section was successful, the Authenticator should decrypt the .auth section in RAM by using the decryption key and a simple XOR operation.

After the decryption of the .auth section, the Authenticator should switch to StartApplication state.

The StartApplication state should call the verify() function in the .auth section. This function checks a signature byte inside the application flash memory and if the signature byte is correct, is calls the StartHandler() function of the Application.

The build process of the Authenticator must include the encryption of the .auth section. Hereby, it is recommended to build the Authenticator ELF binary as usual, extract the .auth section into a raw binary file with the arm-none-eabi-objcopy tool. After that, the raw binary is encrypted with the provided Encryption-Script. As soon as the raw binary file is encrypted, the .auth section in the ELF binary must be replaced with the raw binary content. This can be achieved with the tool arm-none-eabi-objcopy and updating the .auth section.

For debugging purposes, the encryption process during the build and the decryption process during runtime should be enabled/disabled depending on the build configuration. For this, a pre-processor define ENABLE_ENCRYPTION should be used.

Quelle: intern

Functionality Application

After power-up and after the initialization of the HAL components and the scheduler, the system shall enter Bootup State and perform sequentially the following checks:

- Check DualChannelGas Sensor for valid values
- Check WaterSensor for valid values

Only if all checks were successful, the system shall enter Pre-Operational Mode. If any of the checks fail, the system shall enter the Failure state

In Per-Operational state, the use can press button SW1 to enter Operational Mode. As long as the system is in pre-operational mode, no sensor thresholds are checked and no alarms are triggered. Also, no checks for valid sensor signals are performed in Pre-Operational state.

In Operational state, the system shall verify the sensor values and check them against pre-defined thresholds.

- Gas Sensor value > 3000ppm for 5 seconds → Warning by flashing LED D1
- Gas Sensor value > 5000ppm for 3 seconds → Switch to Emergency mode
- Water Sensor value > 250cm for 10 seconds → Warning by flashing LED D1
- Water Sensor value > 300cm for 5 seconds → Switch to Emergency mode

If the application is on Operation state and the user presses SW1, the application shall switch back to Pre-Operational state.
If the application has triggered an alarm and switched to Emergency state, the user can reset that alarm by pressing the button B1 and the application shall switch back to Operational state.

Leaving the Failure state is only via a reset possible.

Quelle: intern

Functionality Application

In Operational state, the application shall display the current WaterSensor value on the 7-segment display

In any other state, the Application should display on both 7-segment display the ‚-' (dash) symbol

The reception of data via the UART (WaterSensor) is only enabled in Operational state. Nevertheless, the functionality to receive the data and handle timeout shall be in a separate module (e.g., WaterSensor) and called from the 10ms task.

If the WaterSensor module didn‘t receive data packets for more than 150ms, the system shall trigger a SENSOR_DEFECT event.

If the WaterSensor module received a data packet, it must verify the following conditions:

- The packet counter is incremented by one compared to the previous received data packet. Consider also the „roll-over“ of the 8-bit counter
- If the packet counter has not the expected value, a SENSOR_DEFECT event is triggered
- The CRC of the data packet must be checked. If the CRC doesn‘t match, a SENSOR_DEFECT event must be triggered.

The DualChannelGasSensor module shall check for a parameterized inconsistency between the to different gas sensor. If one of the gas sensor values is more than 10% different from the other, a SENSOR_DEFECT event shall be triggered.

If the application detects a stack corruption, it shall switch to Failure state and turn on all LEDs

Quelle: intern

Design Constraints

The system shall filter the analog input signals from POT1 and POT2 according the following rules

- POT1: Exponential Moving Average Filter with appropriate filter constant
- POT2: Exponential Moving Average Filter with appropriate filter constant

The system shall debounce the digital inputs for the buttons by 50ms

The system shall use the following task cycles

- 10ms: Analog and Digital Input (incl. UART) Processing
- 50ms: Main Application and State Machine
- 250ms: Health-Monitoring (Stack-Monitoring)

It is not allowed to use floating point calculations (neither SW Emulated nor HW FPU)

A clear layered architecture shall be used. The usage of modules to create a flexible and modular software is highly recommended

The application state machine shall be implemented using a tabled based state machine approach

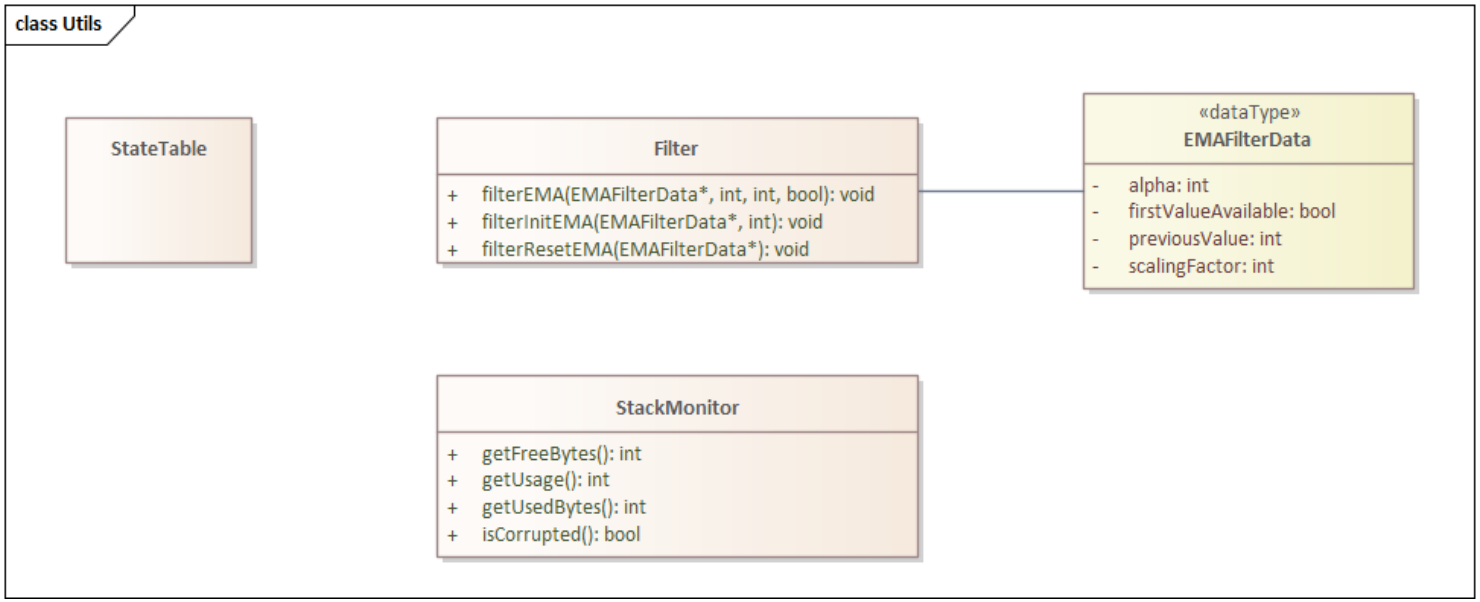
The authenticator state machine shall be implemented using a switch-case state machine approach

The application shall provide a module to monitor the current stack usage respectively the free stack memory and also checks whether the stack is corrupted. Therefore, the startup code for the application must prepare the .stack section accordingly.

Quelle: intern

Design Constraints

The StackMonitor Module shall provide implementations for the functions shown in the Class Diagram. The same applies for the Filter Module.



Quelle: intern