4-on-1-row robot -

# Designing a structured and modular software architecture

Version:v1.0
Date: 7-6-2022

Author: Pascal Faatz

# General information

Student:Pascal Faatz

La Traviata 16

5629NN, Eindhoven

p.faatz@student.fontys.nl / pascal.faatz@alten.nl

+31 (0)6 43 169 199

Student number : 2491281

Class: 43_E4AFST

Company:ALTEN Nederland B.V.

Hurksestraat 45

5652 AH, Eindhoven

Technical supervisorAniel Shri

Aniel.shri@alten.nl

+31 (0)6 37 162 867

Business managerGijs Haans

Gijs.haans@alten.nl

+31 (0)6 27 025 966

School:Fontys University of Applied Sciences

The all around 1

5612 AP, Eindhoven

School supervisor:Jeedella S.Y. Jeedella

j.jeedella@fontys.nl

+31 088 507 81 48

# Referenced documents

| Id | Reference | Title | Writer |
|---|---|---|---|
| **D1** | Motor driver datasheet [1] | ESCON 36/3 EC Servo Controller | Maxon group |
| **D2** | Encoder datasheet [2] | Encoder HEDL 5540 | Maxon group |
| **D3** | Motor datasheet [3] | Maxon motor EC-i 40 | Maxon group |
| **D4** | Servo datasheet [4] | Parallax Servo | Parallax |
| **D5** | Powersupply datasheet [5] | Mean Well LRS-150 | Mean Well |
| **D6** | Vacuum pump datasheet [6] | SparkFun D2028 pump | SparkFun |
| **D7** | Solenoid datasheet [7] | adafruit 413 solenoid | Adafruit |
| **D8** | RBG sensor datasheet [8] | Taos TCS3472 | Taos |
| **D9** | Lichtsluis design (internal document) | SDD_photodiodeboard.docx | Arjan Verboord |
| **D10** | Custom PCB design (internal document) | 013-e-0001-d.brd | Jeroen Wilbers |

# Foreword

In front of you is the report of my graduation project "4-on-1-row robot - Designing a structured and modular software architecture". The report serves as a completion of the Electrical Engineering programme at the Fontys Hogescholen Eindhoven.

At the Meet & Match event of Fontys in 2019 I came into contact with Mrs. Romy Wachtmeester of ALTEN Netherlands. During a renewed acquaintance in 2021, ALTEN turned out to have a nice graduation assignment available. I am grateful for the opportunity theyhave offered me to complete my studies in the professional environment. During the graduation internship from 1 February to 1 July 2022, I was able to use all facilities at the ALTEN office in Eindhoven.

During the graduation I was helped enormously by Aniel Shri, technical supervisor and Gijs Haans, business manager from ALTEN. I would like to thank himfor all the information, guidance and feedback they have given me. Also Jeedella S. Y. Jeedella, my supervisor from Fontys electrical engineering, contributed to the realization of the assignment. Dank for this.

Finally, I would like to thank my fellow interns and the consultants of ALTEN for their time and cooperation in answering my questions and sharing knowledge. This gave my assignment the depth after which I was looking. I look back on a pleasant and educational time.

That leaves a word of thanks, for you the reader of this report.
I wish you a lot of reading pleasure.

Pascal Faatz

Eindhoven, 7/06/2022

# Table of contents

## Lices of figures

## List of tables

## Summary

ALTEN is an internationally leading consultancy company specialized in consulting & engineering in High Tech development environments. The company has developed a robot in-house that can autonomously play 4-in-1 row against a human opponent. The robot is a demonstration unit that is used at technical fairs and open days of universities and colleges to recruit customers and employees. ALTEN wants to continue using the robot and requires an optimization of the quality and the preparation for an improvement in performance.

Since 2019, several engineers have been tinkering with the robot. As a result, the software architecture has become unclear and inconsistent. In addition, ALTEN has a desire to expand the hardware in the future. The optimization of the robot requires a complete re-design of the control system and the replacement of the microcontroller. ALTEN's question is summarized in the following assignment: *"Establish a structured and modular software architecture for the operating system of the 4-on-1-row robot, with which all changes and extensions can be facilitated in the future and implement this software architecture on an STM32H7 dual-core microcontroller."*

The project worked with the V-model. This has ensured that all necessary project steps have been completed in a structured manner. First, the requirements were determined with all stakeholders. The requirements serve as a starting point for the project, determine the choices in the preliminary research and serve as validation when the project result is delivered. The main requirementsrelate to a structured and modular software architecture that is logically constructed so that software developers can efficiently manage and upgradethe software. In addition, a Board Support Package (BSP) is requested to control the necessary hardware. Both must be integrated on an STM32H7 dual-core microcontroller.

In the preliminary research phase, the interaction between hardware and software was analysed. A re-design of the hardware is not part of the scope of the project, but the hardware is inextricably linked to the operating system. Based on the functioning of the system and the requirements, a number of methodological and technical choices were then made. The main ones relate to the template for the software architecture (arc42), the core distribution and the IDE (STM32CubeIDE). These choices are based on quality, user-friendliness and robustness.

In the design phase, the software architecture for the operating system was rebuilt. All steps of the arc42 template have been completed and clarified with diagrams. Thanks to the diagrams and the modular structure, it is clear how the system works and can be easily managed and expanded. The design phase was assessed by the stakeholders and after approval proceeded to the implementation and test phase. In this phase, some software modules necessary for the operating system were implemented and individually tested. After that, a number of modules were merged into different demos to demonstrate the suitability of the software architecture.

In the validation phase, it was tested whether the requirements have been met. The operating system has had a complete re-design of the software architecture and based on a BSP all hardware can be controlled with a dual-core microcontroller. The demos have been successfully completed and all important requirements have been met. However, the robot is not working yet. For this, a few modules still need to be designed. This can easily be done on the basis of the BSP and the software architecture. In addition, after the upgrade, the system is ready for an expansion of the hardware, such as an Ethernet connection and a screen.

## Summary

ALTEN is a company that works worldwide and outsources its consultants to projects in the high-tech sector. The company made its own robot that can autonomously play 4-in-1-row against human players. It serves as a demonstration unit on technical fairs or on open days of universities to attract new customers and employees. ALTEN wants to keep using the robot and requests an optimization of quality and a preparation for better performance.

Since 2019 different engineers have worked on the robot. Because of this, the software architecture became unclear and inconsistent. Furthermore, there is a wish to upgrade the hardware in the future. A complete redesign of the software architecture is needed to optimize the robot, and the current single-core microcontroller is swapped out for a dual-core. The following assignment can summarize the project: *"Set up a structured and modular software architecture for the operating system of the 4-in-1-row robot that can facilitate all the needs for changes and upgrades in the future and implement the software architecture on an STM32H7 dual-core microcontroller."*

The V-model is used within the project. This model helps create all the necessary steps to go through a project in a structured manner. First, in consultation with the stakeholders, the requirements are set. The requirements are there to validate the result, base research questions on, and as a starting point for the project. The most critical requirements relate to a structured and modular software architecture built in a logical way that software developers can easily manage and upgrade the software. Also, a Board Support Package (BSP) needs to be made to control the essential hardware. Both need to be integrated on an STM32H7 dual-core microcontroller.

In the research phase, the interaction between hardware and software is analyzed. A hardware redesign is not part of the project. However, the hardware is inseparable from the operating system. Based on the requirements and functions of the robot, a few technical and methodological choices have been made. The most important is for a software architectural template (arc42), the distribution of software modules on the dual-core, and the IDE (STM32CubeIDE). Attention has been paid to quality, usability, and robustness.

In the design phase, the software architecture of the operating system needs to be rebuilt. The steps of the arc42 template are followed, and diagrams are made to clarify the software architecture. Because of the diagrams and modular structure, the working of the system is straightforward, and it can easily be managed and expanded. The stakeholders approve the design phase. After that, the implementation and test phase can start. A few modules are chosen during the tests to check if the software architecture is suitable to be used. For this, demos are created.

The last phase is the validation phase. In this phase, a check is done to see if all the requirements are met. The complete software architecture of the operating system has undergone a redesign, and all hardware can be controlled using a BSP on the dual-core microcontroller. The demos have been successful, so all-important requirements have been checked. However, the robot is not fully operating yet. Some modules still need to be implemented. This implementation can quickly be done using the BSP and the software architecture. After the upgrade of the microcontroller, the system is ready for hardware upgrades in the future. Think about an end-game screen or an ethernet connection.

## Abbreviations

| Term | Explanation |
|---|---|
| SAD | Software Architecture Document |
| BSP | Board Support Package |
| ST | STMicroelectronics |
| IDE | Intergrated Development Enviroment |
| HALL | Hardware Abstraction Layer |
| MoSCoW | M- Must have , S- Should have, C- Could have, W- Won't have |
| BAC | Building Block View |
| GPIO | General Purpose I/O |
| RTOS | Real Time Operating System |
| ISR | Interrupt Service Routine |
| MDMA | Master Direct Memory Access |
| NVIC | Nested Vector Interrupt Controller |
| HSEM | Hardware Semaphore |
| EXTI | External interrupt/ event controller |
| SEV | CPU send-event instruction |
| MPH | Memory Protection Unit |
| PWM | Pulse Width Modulation |

# 1 Introduction

*A 4-on-1 row robot that never loses, ALTEN has it at home. .*

Everyone knows the game 4-in-1 row. The game is played by two players. One player plays with red chips, the other with yellow ones. The game board consists of seven columns and six rows. A player takes turns throwing one chip into the game board. The goal of the game is to be the first to have four chips adjacent to a horizontal, vertical or diagonal row.

ALTEN has developed a robot in-house that can play autonomously against a human opponent 4-on-1 row using an algorithm. The 4-on-1 row robot is built with industrial components to demonstrate the knowledge of different systems. The robot is used as a demonstration unit at trade fairs and open days, where passers-by can play a round. First, the player makes an opening move. Then it's the robot's turn where the control system determines the move, picks up a chipand drops it into the chosen column of the board. The system keeps track of the gameplay. Once the game is over, the robot collects all the chips and sorts them by color. The game is ready for the next round.

The 4-on-1-row robot was devised within ALTEN to give a challenge to consultants who are temporarily not on a project. The first prototype was delivered in 2019. After that, the operating system was optimized and expanded by various engineers . This has resulted in incoherent software that is difficult to understand and therefore difficult to maintain or expand .

ALTEN wants to guarantee the quality and performance of the 4-on-1 row robot for a longer period of time. This means that the operating system must be easy and robustly adaptable. To make this possible, the architecture of the control system will have to be rebuilt and modular. The re-design of the software architecture offers the possibility to perform a hardware upgrade in parallel, replacing the single-core microcontroller with a dual-core microcontroller. This allows extensions to be added that underline the robot's function as a demonstration unit. A structured and modular software architecture for the control system, in combination with a dual core microcontroller, makes the robot ready for the future.

The software architecture must be user-friendly so that engineers with little knowledge of hardware can still work with the components of the robot. That is why standardized methods were chosen. For the re-design of the software architecture, a template is used that is standard within ALTEN (arc42) and ensures a structured structure of the software using diagrams. The software consists of embedded software layers that are modulair and are generated partially automatically. The choice of ALTEN for the STM32H7 dual-core microcontroller led to the use of an IDE from the same supplier, so that the quality of communication within the core is guaranteed in advance. The methods are up-to-date and reference work is available.

This report begins with a brief introduction to ALTS (Chapter 2). This is followed bya description of the assignment (Chapter 3). The functioning of the current robot, the requirements of ALTEN and the methodological choices are explained in chapter 44 Thenchapter 55 most important parts of the software architecture. In chapter 6 , the implementation and testing is explained on the basis of some demos. Finally,the requirements in Chapter 7 are validated. The report concludes with some conclusions and recommendations in Chapter8.

# 2 ALTEN

ALTEN is a consultancy and engineering organization for the high-tech and IT sector. The company was founded in 1988 and is headquartered in France. ALTEN is an international company with more than 42,000 employees in 24 countries. There are 5 offices in the Netherlands (Figure 1with a total of 1. 200 employees of which 91% engineer[9] .

*Figure 1 Offices ALTEN Netherlands*

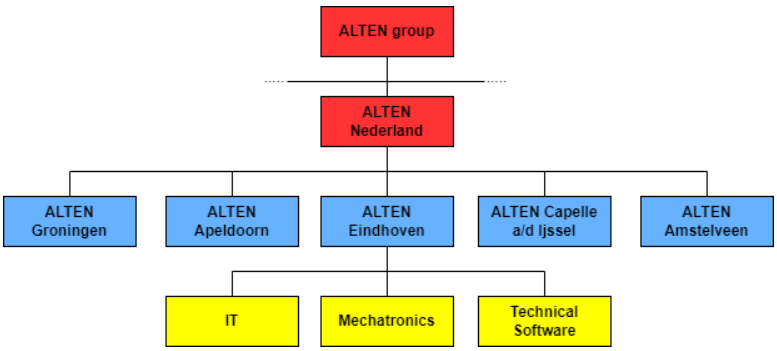



*Figure 2 Organization diagram*

Technology is the central pillar within ALTEN. Alten offers kWality, reliability and innovations in the field of high-tech solutions. ALTEN Works commissioned for companies from the Automotive, high-tech (make)industry, defence, energy, traffic/ transport and telecom sector. The customer is the owner of the end result and without permission it is not possible to communicate about assignments or the end result.

ALTEN has three departments in Eindhoven: IT, Technical Software and Mechatronics (Figure 2). The project for the 4-on-1-row robot falls within the Mechatronics department. This department includes consultants with a background in mechatronics, mechanical engineering, electrical engineering and control engineering. All specialisms are represented to answer technical questions integrally. The employees are the most important production factor of the company. ALTEN invests in individual development, the further development of expertise and the provision of a springboard for the future.

ALTEN uses the same organizational model for every project. Within this model, it is clear to each stakeholder what his role is (see Annex II. Plan of action, Figure 4). This model has also been used for the 4-on-1-row robot project. The stakeholders are described in Table 1 of Annex II. Plan of action. During the process, communication was made with and reported to the technical supervisor, business manager, the customer (ALTEN) and the internship supervisor of Fontys Hogeschool Eindhoven.

# 3 The assignment

This chapter describes the problem statement, objective and assignment. Section 3.4 discusses the requirements formulated by the stakeholders. The requirements determine the result of the project and are validated in chapter 7. The chapter closes with the V-model, which is used as a guide for the project.

## 3.1 Problem definition

The 4-on-1-row robot is a demonstration unit to attract new customers (technology fairs) or new employees (open days at HBO / universities).

The robot has functioned well so far. The system works and the hardware is in order. But the software and software architecture require an overhaul. Because several consultants have programmed the software over a long period of time, the operating system is very unclear and the architecture unclear. In addition, the single-core microcontroller has almost no space left to allow an upgrade of the hardware.

Because the 4-on-1-row robot is representative of ALTEN's knowledge and skills, it is important that components, software and software architecture can be continuously developed by various specialists.

## 3.2 Objective

The purpose of the assignment is twofold. The assignment aims to redesign the software and software architecture of the robot's operating system, in such a way that the software can be easily adapted or expanded in the future in the event of changing circumstances and wishes. In addition, the basis is laid for an improved performance of the robot through the application of a dual-core microcontroller. By means of a dual core, real-time processing and game handling can be separated from each other and run parallel at the same time.

Before the assignment can be started, preliminary research is necessary to understand the operation of the robot and to make some methodological and design choices related to the re-design of the software architecture.

Application of the dual-core in combination with structured software architecture makes it possible to easily add new features such as a screenand Ethernet connection in the future. In this way, the robot can once again prove its service as a demonstration unit and the quality and performance are assured for a longer period of time.

## 3.3 Design assignment

The assignment is as follows:

> *"Set up a structured and modular software architecture for the control system of the 4-on-1-row robot, with which all changes and extensions in the future can be facilitated and implement this software architecture on an STM32H7 dual-core microcontroller."*

## 3.4 Scope and demarcation

- Outside scope of the project is the re-design of the hardware and the mechanical part of the robot;
- Nothing changes in the gameplay of 4-in-1 row;
- Within ALTEN, the robot is physically available for further research;
- We work with a dual-core microcontroller chosen by ALTEN;
- For the software architecture, the standard template of ALTEN was used;
- The choice of the C or C++ programming language is the standard for the IDE;
- The software architecture for the operating system is a final version;
- To verify design choices, a "proof of concept" has been delivered;
- A re-design of the robot's control system has been requested, the implementation of the entire system is not realized within the project.

## 3.5 Requirements

There are a number of requirements attached to the assignment. These requirements have been formulated in consultation with the stakeholders. Table 1 definesTable 1 the requirements. These requirements are carried out according to MoSCoW, whereby the requirements of the results are classified according to M- Must have, S- Should have, C- Could have or W- Won't have. In consultation with alten's technical supervisor, the priority within the requirements has been determined.

*Table 1 Requirements*

| ID | Requirement | MoSCoW |
|---|---|---|
| UR.1 | The architecture of the operating system must be structured and modular in order to give hardware and software developers who are not familiar with the system a quick insight into the functioning of the robot. | Must |
| UR.2 | The software architecture must be future-proof so that software developers can perform effective and efficient management and upgrades. | Must |
| UR.3 | The architecture of the operating system must be logically constructed on the basis of diagrams so that software developers and testers can quickly gain insight into the functioning of the software. | Must |
| UR.4 | The coherence between software and hardware must be logically constructed on the basis of diagrams so that software developers can implement the final software. | Must |
| UR.5 | A Board Support Package (BSP) must be made of the operating system with which the necessary hardware components of the robot can be controlled. | Must |
| UR.6 | The components of the BSP that are necessary for the functioning of the robot must be tested independently of each other within the project. | Must |
| UR.7 | Within the new operating system, the STM32H7 dual-core microcontroller must be integrated. | Must |
| UR.8 | The robot must have a calibration tool that, when initializing the system, ensures that the motors are "homed" in the Z and X direction. | Must |
| UR.9 | A Pin-out must be drawn up with a table and diagram to ensure that hardware developers can develop a PCB design for the dual-core processor in the future. | Should |
| UR.10 | The algorithm running on the Raspberry Pi must be integrated on the new STM32H7 dual-core. | Could |
| UR.11 | A physical demo must demonstrate that the modular structure of the software architecture is applicable and functioning. | Could |

The requirements define the result to be delivered by the project. In the validation (chapter 7Validation

## 3.6 Project approach

In order to ensure a structured project, the V-model was chosen. The V-model is a linear developmentmodel that is divided into a number of phases. Each phase produces a result, which is necessary for the next phase. This repeats itself for all phases, with each new result the system growing. The second point of attention is that for the requirements and the design on the left there is a corresponding validation or integration / test on the right (Figure 3).

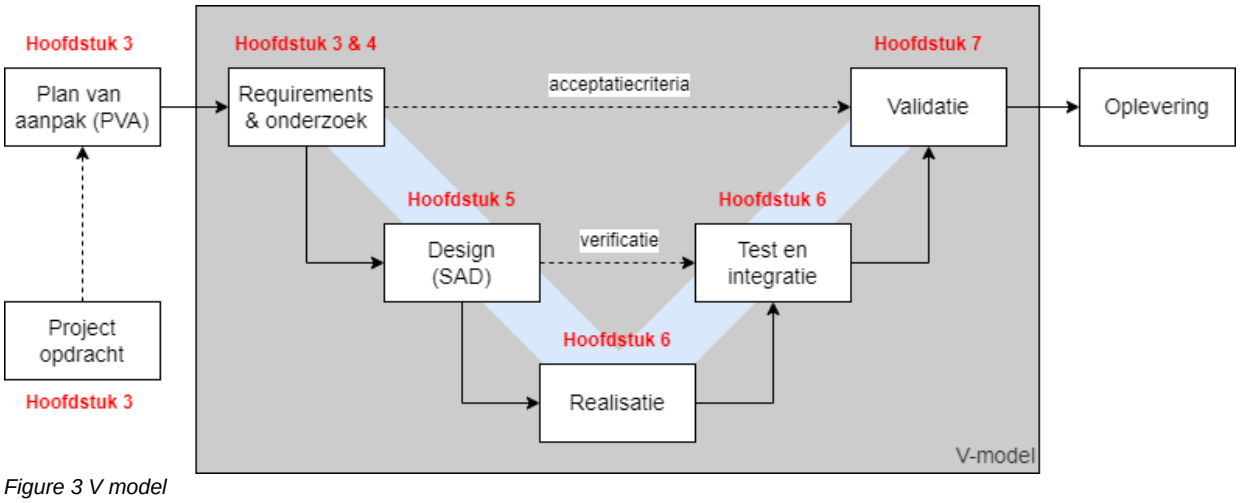



*Figure 3 V model*

# 4 Preliminary research

The design assignment is divided into a number of research questions that need to be answered before the design can be started. The starting point is an existing robot that is controlled by an operating system. A hardware re-design of the robot is outside the scope of this project. However, you have to work with the existing hardware. This means that the hardware itself is not out of scope. It is important to understand in advance how the system works and what (hardware) parts the robot consists of. There are several documents available in which this information is recorded. In addition, it should be investigated which methods can be helpful in the re-design of the software and the software architecture, and how the software can be built modularly. In addition, it must be investigated what the context is in which the given dual-core microcontroller must function and what the functionalities are. Finally, the requirements of the design assignment are summarized in a table that can serve as a checklist for validation at the end of the assignment.

## 4.1 The 4-on-1-row robot

For a good understanding of the current robot, the following paragraphs explain how the system works and what (hardware) parts the robot consists of.

### 4.1.1 Gameplay

The game course of 4-in-1 row is fixed. The player and the robot take turns placing chips in the game board until one of the two has won, a draw occurs or there is foul play. The behavior of the robot can be described in three phases: a phase in which the player is on the move, a phase in which the robot is on and a phase in which the game is finished and all chips are cleared. Respectively, a human phase, a robot phase and a clean-up phase. These three phases form the basis of the operating system.

### 4.1.2 Hardware identification

The mechanical system and electronic hardware components of the 4-on-1-row robot are present inALTEN. The block diagram (Figure 4 4) shows which parts the robot is made up of and how they are connected to the control system. Figure Figure 5 a top view of the physical setup. Figure Figure 6 a bottom view that makes the other components visible. Table Table 2 describes which components are used and what the function is in the system.

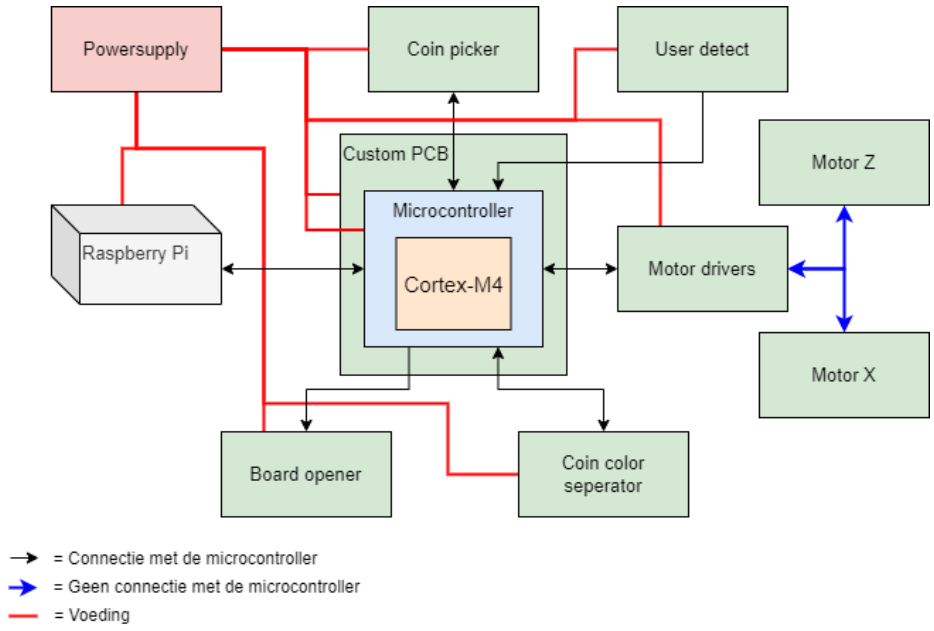



*Figure 4 4 block diagram 4-on-1-row robot*

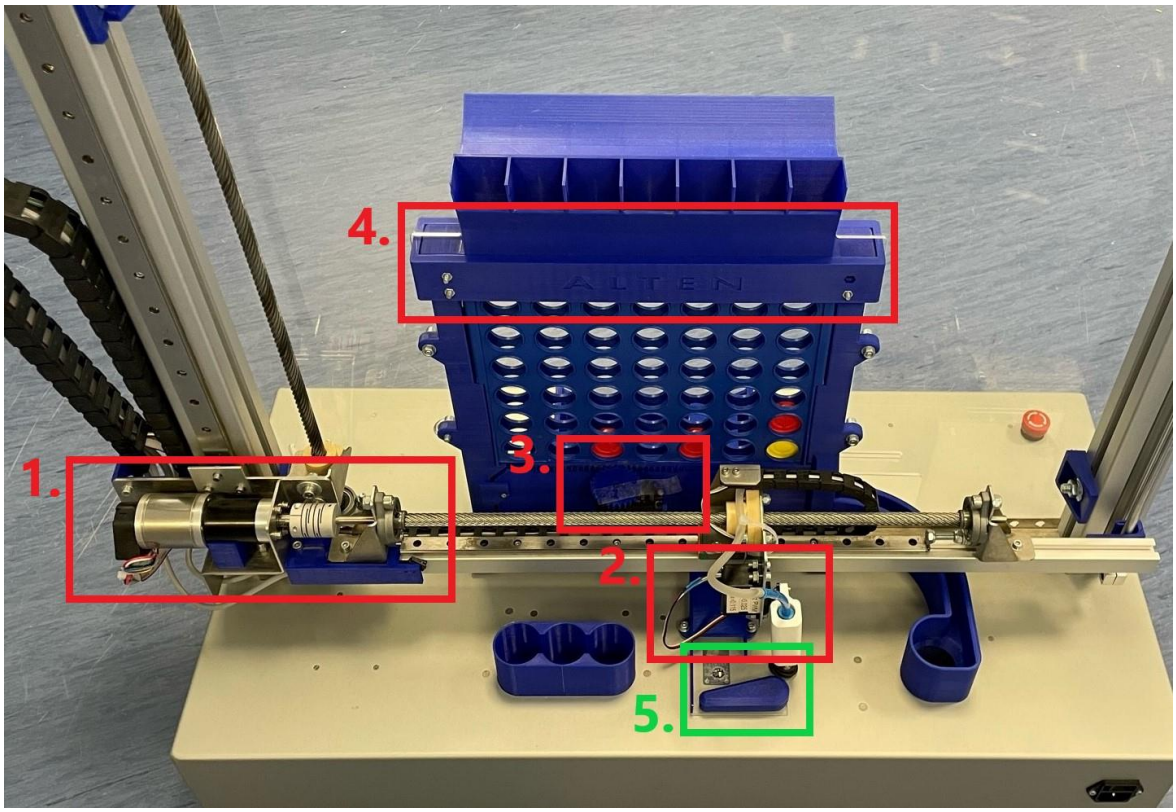



*Figure 5 Top view 4-on-1 row*

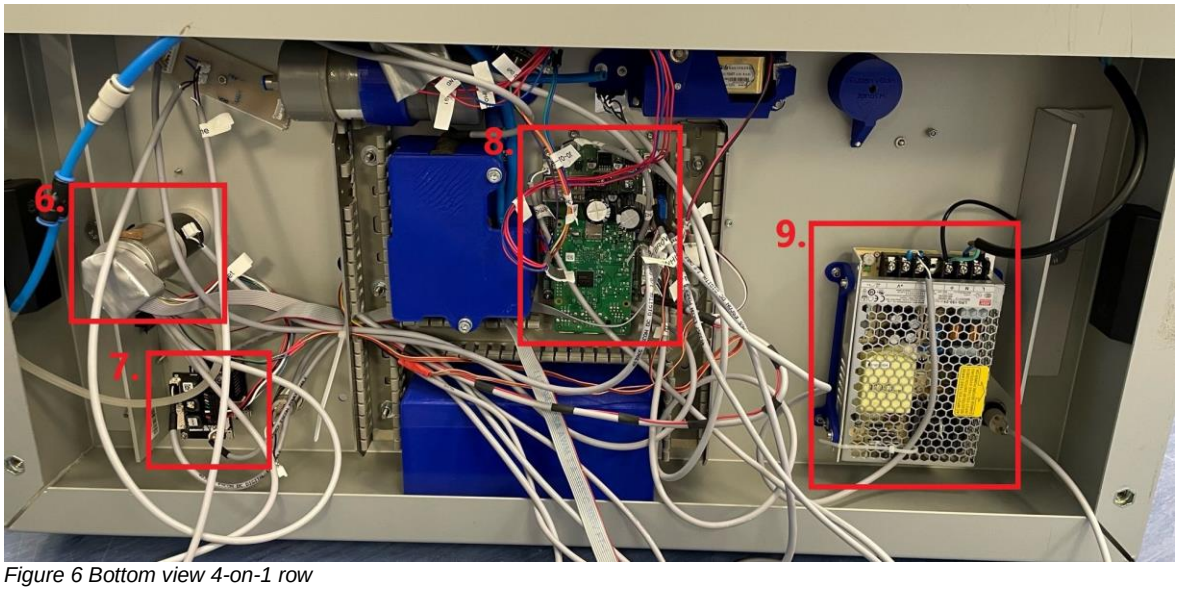



*Figure 6 Bottom view 4-on-1 row*

*Table 2 Components 4-on-1 row*

| No. | Module | Components | Task |
|---|---|---|---|
| **1** | Engine X | Motor (D3with a rotary encoder (D2). | The motor that controls the X-axis ensures that the "Coin picker" can move around this axis. |
| **2** | Coin picker | Servo (D4and vacuum gripper (D6). | The vaccuum gripper can pick up and release the chips. The servo ensures that the vaccuum gripper can move. |
| **3** | Board opener | Servo (D4). | This servo ensures that at the end of the game all chips can fall out of the board. |
| **4** | User detect | Light lock (7x LED with photodiode, D9. | The light lock is responsible for detecting a chip that has been placed in the game board. |
| **5** | Coin color separator | Solenoid (D7 and RGB sensor (D8). | The Flipper and RGB sensor are the components that are active during the c lean-up phase of the robot. The RGB sensor checks the color of a chip and the solenoid can, by means of a pinball, shoot chips to the player's box. |
| **6** | Motor Z | Motor (D3with a rotary encoder (D2). | The motor that controls the Z-axis ensures that the "Coin picker" can move around this axis. |
| **7** | Engine drivers | Motor driver X and Z (D1). | The motor drivers control the control of the motors and the communication with the microcontroller. |
| **8** | Custom PCB | PCB for the microcontroller (D10) and a Raspberry Pi shield. | The microcontroller takes care of the real-time processing, allsignals that are necessary for the control of the hardware. The Raspberry Pi uses an algorithm to calculate the robot's next move. |
| **9** | Powersupply | D5 | The power supply ensures that the entire system is supplied with a power supply. |

By going through the datasheets of hardware components, the internal documentation of the in-house designed hardware and software, it becomes clear how the hardware components behave during the course of the game. Based on this, the signals and protocols for the operating system can be derived.

## 4.2 arc42

When designing a software architecture, it is necessary to use a template that goes through all the necessary diagrams and steps.

The aim of every (embedded) software team is to write error-free code, have a motivated team and ensure that the system works efficiently. Software architecture plays an important role in this. But it also happens that the software architecture is not well documented, the code is unclear or the architecture is simply not present. A few problems can hinder a software project:

1. *The non-existence of documentation or outdated documentation.*
   The documentation of the software architecture may lag behind what has been implemented, but it may also be that no documentation is available. Outdated, or non-existent, software architecture documentation creates complications while implementing, upgrading or modifying the software because it is not clear how the software is structured and communicates.

2. *Cluttered documentation.*
   Existing documentation can also be very confusing. This documentation is often created without any awareness of purpose and is created by multiple individuals without coordination. Cluttered architecture is difficult to understand, maintain, or adapt.
3. *Too much documentation.*
   An architecture with too much documentation is also not optimal. This can result in barriers to future use, as searching for or modifying information in the document cannot be done in a structured way.

For the current software architecture of the robot, AD 1 and 2 apply. After consultation with the software architect within ALTEN and the technical supervisor is for it Chosen to use the Arc42 template (Figure 7) to be used to create a software architecture. Arc42 is a template for documentation and communication of software or system architecture. It is a clear, simple and effective way to structure software. Furthermore, the template is optimized for users and engineers. This template is frequently used by the software architects of ALTEN. This makes it plausible that the template meets the demand from the business community. In addition, it ensures that architectuur informatie whether important design choices are well described and that the entire architecture is easy to maintain [10].

*Figure 7 arc42 logo*

## 4.3 Modularity

In an embedded system, software layers are used. The application of these layers ensures modular software construction. The software can also be used independently of the system on which it is located. This ensures flexibility. Figure 8 shows the layers that make up an embedded system.

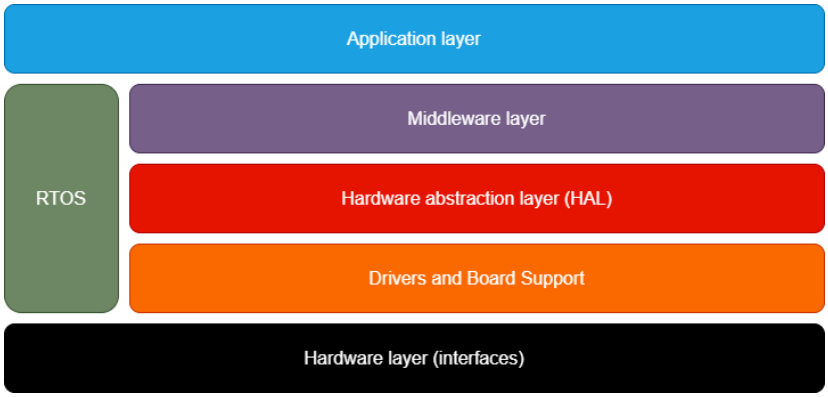



*Figure 8 Embedded software layers*

The following describes what kind of software is in each layer. This is from high level to low level.

- *Applicationlayer*
  The "Application layer" is seen as a high level software layer. This layer defines the system. This layer is not interchangeable because it is system specific. The Application layer depends on, and is managed and run by, the software from the Middleware layer.
- *Middleware layer*
  The "Middleware layer" is the layer that controls the communication between the Application layer and HAL, Drivers and Hardware layer. The Middleware layer can also facilitate communication between two different Application layers.

- *Hardware Abstraction Layer (HAL*[11] *)*
  HAL serves for a formalized interaction between hardware (drivers) and software. HAL ensures good communication between the system and the external and internal hardware, with the implementation focusing on creating abstract, high-level functions. This means that software that uses the functions does not need to have knowledge of the functionalities of the hardware.
- *Drivers*
  The "Drivers" layer contains software libraries that initialize the hardware and manage access between higher software layers and the Hardware layer.
- *Hardware layer*
  The "Hardware layer" stands for the layer where the hardware is defined. These are often the physical pins of a microcontroller that go to the hardware in question.

By using layers for the re-design, in combination with the template arc42, it is possible to meet the design assignment of a structured and modular software architecture.

## 4.4 STM32H7 dual-core

Number 8 in Table 2 refers to the current operating system based on a microcontroller and Raspberry-Pi. On the Raspberry Pi runs the algorithm to determine the next move of the system and on the microcontroller runs the operating system (real-time processing). The current microcontroller is a single-core microcontroller: STM32F303VCT6 with a Cortex-M4. In theory, this microcontroller still meets the system/ performance requirements of the design for the current 4-on-1-row robot. It can control the robot's hardware. But if expansions take place (such as Ethernet connection, screen or sound effects), this microcontroller quickly falls short.

A dual-core microcontroller will be used for the implementation of the new software architecture. This has been chosen by ALTEN before. It concerns the STM32H7 microcontroller, specifically the STM32H755ZIT6U[12] from STMicroelectronics (ST). This is located on the development board Nucleo-H755ZI-Q as shown inFigure 9. A development board is ideal for easy and efficient testing of the software that is implemented.

*Figure 9 Nucleo-H755ZI-Q*

ST is a company specializing in microcontrollers (MCU's), power electronics and flash memory. This project is all about the microcontroller. ST has a wide range of microcontrollers available. From cheaper, robust 8-bit MCU's tot 32-bit Arm-Home Cortex-M MCUs with eand Extensive choice of peripherals. The STM32H755ZIT6U dual-core microcontroller (in the rest of the document the STM32H7 dual-core) belongs to the 32-bit Arm-Home Cortex-M MCUs. The STM32H7 contains a Cortex-M7 and a Cortex-M4 Core. The Cortex-M7 jug up to a maximum of 480 Mhz turn and the Cortex-M4 up to a maximum of 240 Mhz. This falls within ST under the category of high-performance microcontrollers. The full list of specifications can be found on the ST site [13].

ALTEN has chosen a dual-core microcontroller that meets the futurevision for the 4-on-1-row robot. With the two powerful cores (Cortex-M7 and Cortex-M4), the STM32H7 dual-core can actually implement all the upgrades that are planned. This makes it an interesting option and logical step for ALTEN to implement the system in order to guarantee quality and performance. Furthermore, it is the first time for ALTEN that a dual-core microcontroller is used. This project also serves as a "proor or concept" for its use in subsequent "in-house" projects. Many projects within ALTEN are focused on performance and factors such as cost savings and battery life are less important.

The step from a single core to an STM32H7 dual-core microcontroller is a hardware upgrade that can be implemented simultaneously with a re-design of the software architecture for the 4-on-1-row robot.

## 4.5 Core distribution

It follows from the requirements that the system will be implemented on an STM32H7 dual-core microcontroller and that the real-time processing and game handling will be separate. In the old

system, the real-time process sing runs on a Cortex-M4 and the game handling and determining the next move is carried out on the Raspberry Pi. In the new situation, the Cortex-M4 continues to perform real-time processing, because the principle of the game 4-in-1 row does not changeand adjustments to the functionality of the robot are not foreseen. Within the project, the game handling will be moved from the Raspberry Pi to the new Cortex-M7. The Cortex-M7 is much more powerful than the Cortex-M4, which makes it possible to move the algorithm for the next move to the Cortex-M7 in the future. For the time being, the algorithm will remain on the Raspberry Pi because it is outside the scope of the project. Furthermore, the Cortex-M7 is also suitable to facilitate new hardware expansions.

## 4.6 STM32CubeIDE

The choice of an STM32H7 dual-core affects the tool needed to send written code to the microcontroller. An Intergrated Development Enviroment (IDE) is often used for this. For ST's microcontrollers, there are several possibilities for an IDE.

- STM32CubeIDE
- IAR
- Keil

STM32CubeIDE is the IDE of ST itself. This is free and contains all the options relevant to an ST microcontroller. It also contains a visual environment where[14]the pin out can be set with signals and protocols. The STCubeMX software tool that comes with the IDE automatically generates initialization/ configuration software for signals and protocols.

IAR and Keil are similar to each other. Bothn have a free version but the full package costs money. The free version is outdated and has limitations. For example, it is not possible to program on every microcontroller and there is a maximum amountfor the software file that can be uploaded.

ST's IDE, STM32CubeIDE, was chosen because it comes from the same supplier as the STM32H7 dual-core and because with IAR and Keil as IDE it is not possible to program on every microcontroller. In addition, the size of the data that makes up a system at IAR and Keil is limited.

One of the biggest advantages of the STM32CubeIDE environment is that the HAL of ST can be used. Figure 10 shows which software layers are automatically generated/ used by the STCubeMX software tool.

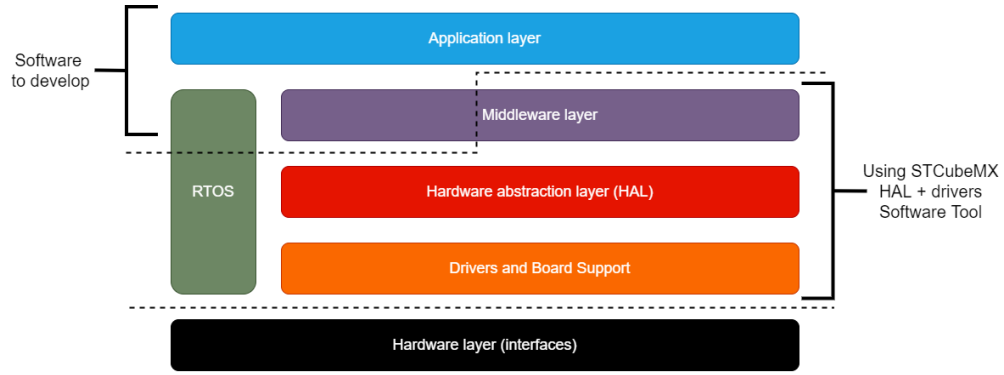



*Figure 10 STCubeMX software tool generated layers*

All signals and protocols (pin specific configuration) that are defined in advance in the setup phase are generated by the software tool in the HAL and Driver layers. The formatting of the code (for the Middleware, such as ethernet) is also partly performed automatically by the STCubeMX.

HAL is used to shorten the development cycle and to use libraries that have already been sufficiently tested. In addition, the choice of ST HAL takes into account software engineers, who work with multiple microcontrollers and have to transfer the application from one platform to another. Furthermore, a HAL is ideally suited to allow engineers with little hardware knowledge to work with these components. They do not need to have any specific knowledge of the hardware details. The HAL is provided by ST. Only the pin-specific configuration is generated. At the ST HAL library, tasks such as UART and i2c are processed by a HAL and therefore do not have to work at register level (low level) to get these protocols working.

## 4.7 Conclusion

In this chapter, research questions have been answered and some (methodological) choices have been made for the implementation of the project. With the study of the robot, it has become clear which hardware components the new operating system must take into account. For the new software architectof the operating system, the template arc42 was chosen. The template is standard for ALTEN's projects and it offers a clear, simple and effective way to structure software. To guarantee the modularity of the architecture, software layers are used. The software can be used independently of the system on which it is located.

It is predetermined that the software architecture must be implemented on a dual-core microcontroller. The functionalities have been investigated. In order to keep the performance of the robot robust in the future, it was decided to separate the game handling and real-time processing. This helps to keep the system clear and ensures that the two cores are optimally used. This is also the first step to implement the system on one microcontroller. Because the Cortex-M7 is very powerful, the algorithm for the next move from the Raspberry Pi to the Cortex-M7 can eventually be retrieved without additional measures.

STM32CubeIDE from the same supplier as the dual-core microcontroller was chosen as the programming environment . The STCubeMX software tool that comes with the IDE automatically generates initialization/ configuration software for signals and protocols. This speeds up the development cycle for the project.

# 5 Design software architecture

In this chapter the design of the software architecture is discussed. The software architecture is the mainassignment of this project. We worked with the template arc42. The template describes how the software is built up, communicates with each other, where data goes and how it behaves when the system is operational (Software Architecture Document, SAD). The following sections explain the main chapters of the SAD that determine the implementation of the system, whilediagrams and accompanying text help the reader to follow the steps. In addition, the design choices are explained. The complete SAD is set out in Annex III. Software Architectural Document.

## 5.1 System context SAD

In the chapter "System context" within the SAD, it is described what the scope of the project is and which external and internal hardware and people are connected to the system (Project context). It is important to map this out because the design must take into account the interfaces of the various external and internal hardware components. It is crucial to understand and describe all interfaces before starting a software architecture document.

First of all, a representation of all external hardware interfaces and people outside the system was created (see Figure 11). The 4-on-1 row robot has no external hardware. The people who use the 4-on-1 row are "Person" (Person playing the game), "System engineer" and "Operator". The "Player" is the person who plays the game. He/she throws a chip in a column of his/her choice every time it's his/her turn and gets to see what the result of the game is when the game is over. The "System Engineer" is the person who maintains the system. It provides an update or upgrade if needed and debugs the system if an error occurs. The "Operator" is the person from ALTEN who turns on/off or resets the 4-on-1-row robot .

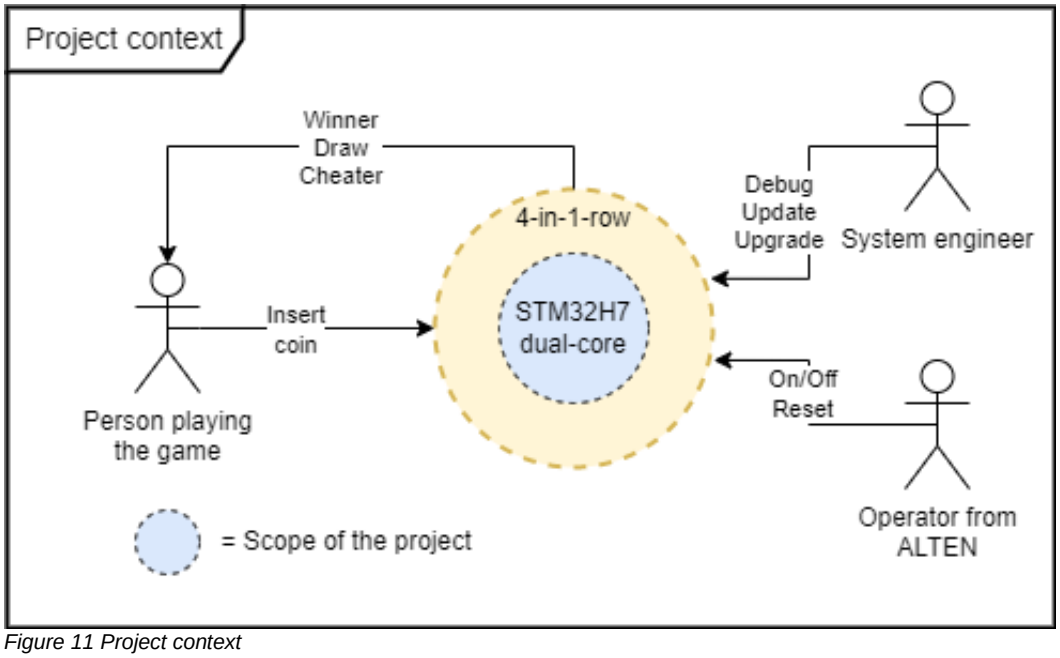



*Figure 11 Project context*

After the project context has been drawn up, we looked at what kind of internal hardware interfaces within the system are connected to the project scope (blue circle). The research (Preliminary research) prior to the start of the software architecture allows a diagram to be drawn up that is shown in Figure 12. This is the System context. In the System context you can see which hardware components are connected to the STM32H7 dual-core and what kind of commands they receive and send back. This gives a generaloverview. It is a tactical representation to be able to consult with all stakeholders without going into too much technical/operational details.

In addition to the System context, a Technical context has also been drawn up. The technical context indicates how the internal hardware interfaces are connected to STM32H7 dual-core and what kind of

signal and/or protocols are used for the connectionas well. A technical context is important for the team that works operationally with the software modules for the hardware. The Technical context is set out in Annex III. Software Architectural Document.

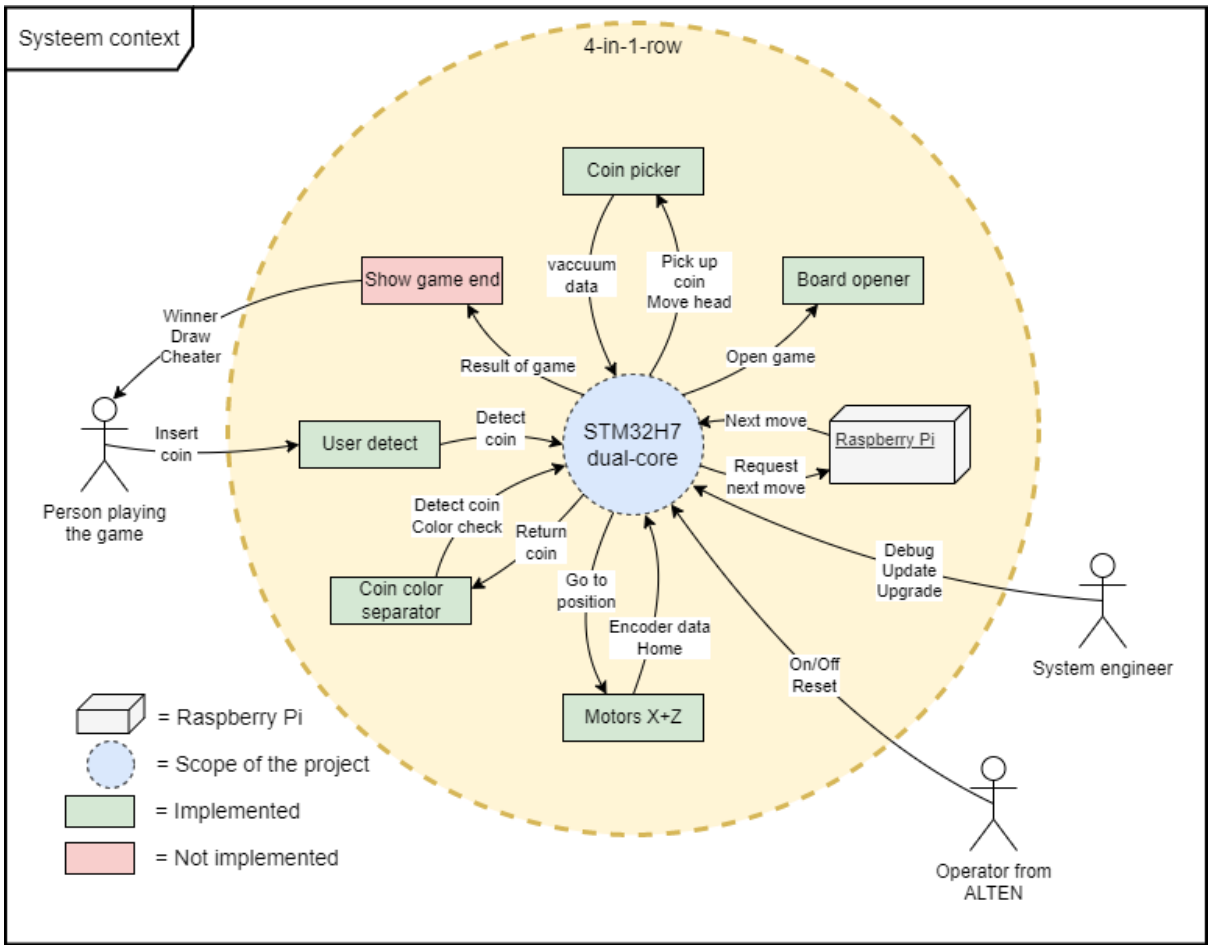



*Figure 12 System context*

## 5.2 Building block view SAD

The chapter "Building Block View" (BBV) of the SAD goes deeper into the scope of the project presented in the chapter "System context". The idea of a BBV is to zoom in on the system step by step. This is a good way to dissect a system in a structured way. This makes it clear which software modules the system is made of, how they are connected and where they are located. Alternately we work with "black" and "white boxes" where the white boxes reveal the internal details of the black boxes. This is done through levels. Black boxes are modules of which only the inputs and outputs are known , but the internal details are not. If a level has been described exhaustively, you can choose to further develop a black box into a new white box.

For this project, the first level is the System context (level 0, Figure 12). Here the STM32H7 dual-core is a "black box". In level 1, this STM32H7 dual-core becomes a "white box" that again consists of several blocks as "black box". In level 2, the "black boxes" from level 1 become "white boxes". During each step, inputs and outputs, connections and assignments of the "black boxes" are described. This is the right level of aggregation to consult with stakeholders without going into too much detail at implementation level. After BBV has been completed and discussed, the modular software modules can then be drawn up.

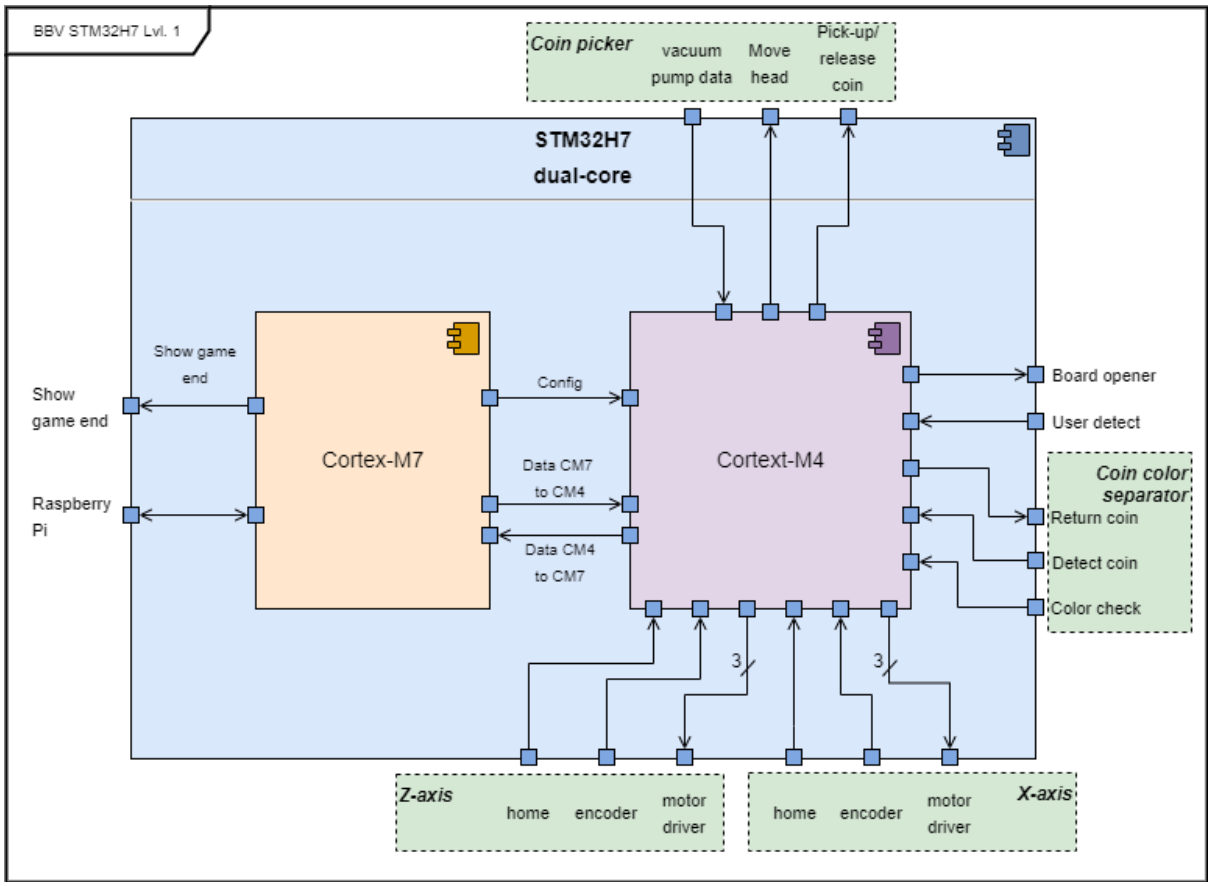



*Figure 13 BBV STM32H7 level 1*

As shown in Figure 13 level 1 of the BBV considers the STM32H7 dual-core to be a "white box" and the Cortex-M7 and Cortex-M4 to be a "black box". The figure shows which hardware interfaces go to the Cortext-M4 that is usedand for real-time processing. The Cortex-M7 handles the gameplay and has a connection to the Raspberry PI. The Raspberry PI uses an algorithm to calculate what the robot's next step should be and then passes it on to the Cortex-M7. The connection between the Cortex-M7 and Cortex-M4 reflects the dual-core communication. Thecommunication is indispensable for exchanging data between the two cores that execute code in parallel from each other.

Figure1415, Figure15displayed as a "white box". For the Cortex-M7, level 2 is considered the lowest level because the blocks within this level are sufficiently representative of the final software modules within the Cortex-M7.

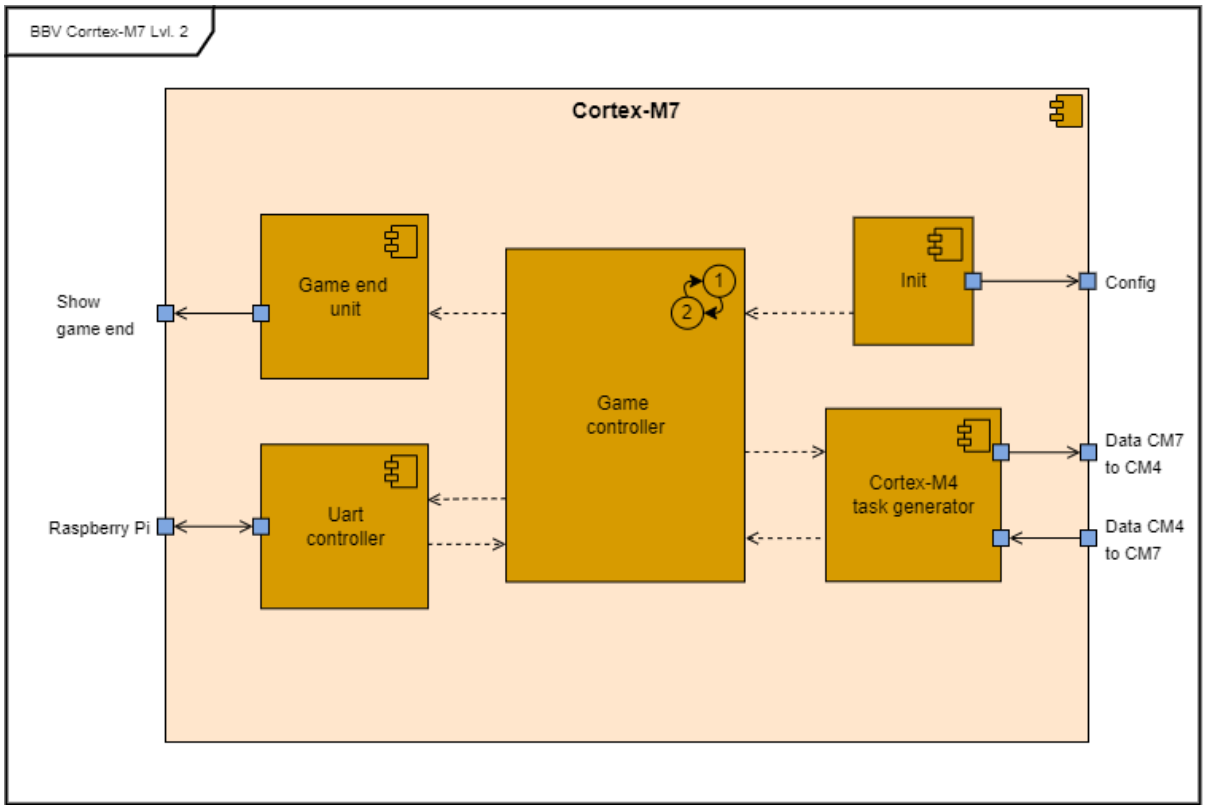



*Figure14 BBV Cortex-M7 level 2*

*Table 3 Description BBV Cortex-M7 level 2*

| Black box | Description |
|---|---|
| **Game controller** | The Game controller is responsible for the course of the game by means of a state machine. |
| **Init** | Rotated once to initialize and boot the Cortex-M7. |
| **UART controller** | The UART controller implements all communication via UART. |
| **Cortex-M4 task generator** | The Cortex-M4 task generator implements communication with the Cortex-M4. |
| **Game end unit** | The Game end unit implements all communication to an output for the player. |

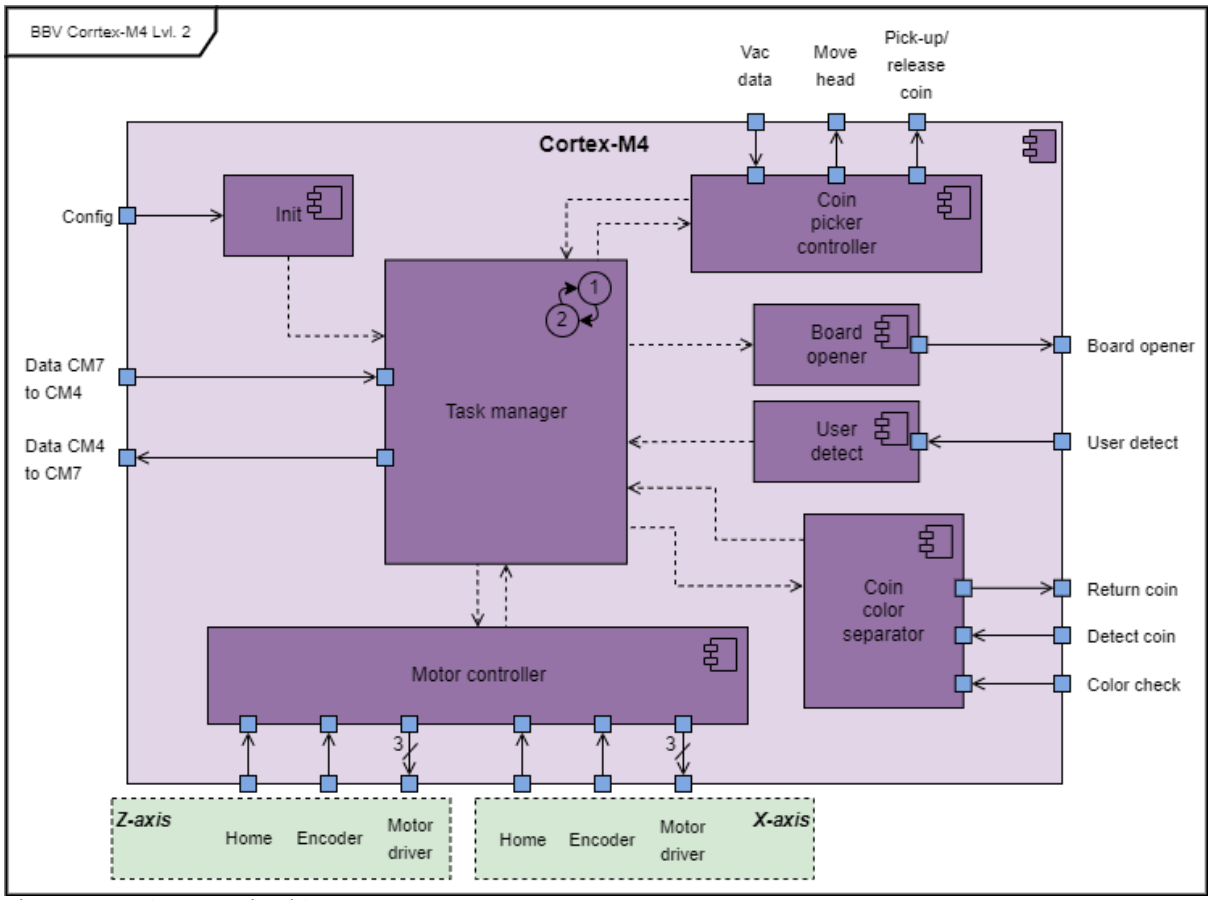



*Figure15 BBV Cortex-M4 level 2*

*Table 4 Description BBV Cortex-M4 level 2*

| Black box | Description |
|---|---|
| **Task manager** | The Task manager is responsible for the progress of the tasks to be executed of the Cortex-M7 by means of a state machine. |
| **Init** | Rotated once to initialize and boot the Cortex-M4. In this phase, the sensors are tested for presence and the motors perform a "homing" protocol. |
| **Motor controller** | The Motor controller implements all communication with the motor drivers, encoders, the PID controllers and home stop. |
| **Coin color Separator** | The Coin color separator implements all functions for handling the separation of the chips. |
| **User detect** | The User detect implements all functions for handling the sensors for the input of the player. |
| **Board opener** | The Board opener implements all the functions for handling the open of the 4-on-1-row board when the game is finished and the clean-up phase begins. |
| **Coin picker controller** | The Coin picker controller implements all functions for handling the picking up and release of a chip. |

For the Cortex-M4, the following blocks are zoomed in even further (level 3): Coin color separator, Motor controller and Coin picker controller. This is necessarybecause these blocks contain more important sub software modules for controlling the robot. In Figure16 the Coin color separator is shown as "white box". For the remaining sub-blocks, reference is made to the full SAD in Annex III. Software Architectural Document.

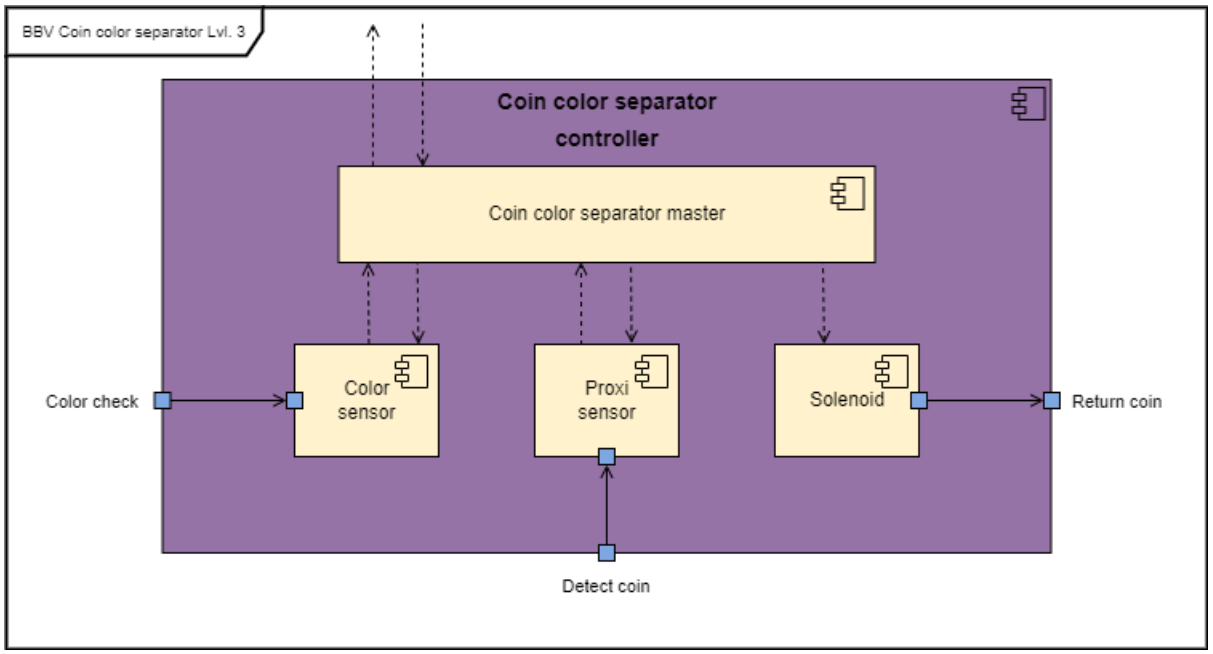



*Figure16 BBV Coin color seperator level 3*

*Table 5 Description BBV Coin color seperator level 3*

| Black box | Description |
|---|---|
| **Coin color separator master** | The Coin color separator master is responsible for receiving the sensor data and controlling the solenoid. |
| **Color sensor** | Requests the data on the color of a sheet. |
| **Proxi sensor** | Receives data whether a sheet is present. |
| **Solenoid** | Is responsible for activating the flipper. |

## 5.3 Runtime view SAD

The chapter "Runtime view" visualizes how the software modules the chapter Building block view communicate with each other when the system is operational. The diagrams g to which software modules communicate with each other in time. The diagrams consist of state machines[15] and sequence diagrams[16]. The state machine and sequence diagrams are crucial to visualize how the robot behaves when the system is operational. By describing all process steps in a model way, it becomes clear which data software modules receive and must send.

A state machine is an abstract model for the behavior of the system. The model consists of a finite number of states that the system can be in, and each state has one or more transitions to subsequent states. These transitions are determined by the input the system receives. A sequence diagram shows interactions between software modules arranged in time order. The diagram shows the software modules involved in one of the states of the state machine. The diagram shows the order of communication exchanged between the software modules to perform the functionality of the state.

The state machine for the game play can be seen in Figure 17. It runs on the Cortex-M7 in the block "Game controller" (Figure14). The first state is the "initialize" state. In this state, the entire system is booted and initiated. As soon as this state is ready, the system goes into the "start game" state. In this state, the system waits for the game to begin. When the game is started, it is the player's turn to one chip in the 4-on-1 row game and the system is in the "human move" state. Hereafter the turn goes to the robot and changes the state to "robot move". In this state, the robot performs one calculated expand. After each turn, the robot checks whether there is a winner, whether there has been cheating, whether there is a tie. If not, the turn goes back to the player it's up to you and locatedt the system is located in the Associated State (human move or robot move). Once the game is over The Clean-up state of start and all chips will be cleaned up. The red chips go to the robot and the yellow ones to the player's box.

The state machine for real-time processing of the hardware can be seen in Figure 18. This runs on the Cortex-M4 in the block"Task manager" (Figure15). The first state is the "initialize" state. In this state be the hardware components configured and set. After that, the Task manager a "Idle" state. In this state, we wait until the Task manager a task gets from the Cortex-M7. Once a task enters on the basis of the task determined the next state. This jug the "human move", "robot move" or "clean-up" state be. Each of these States controls the hardware to task to complete. If the task completed the Task manager back to the Idle State.

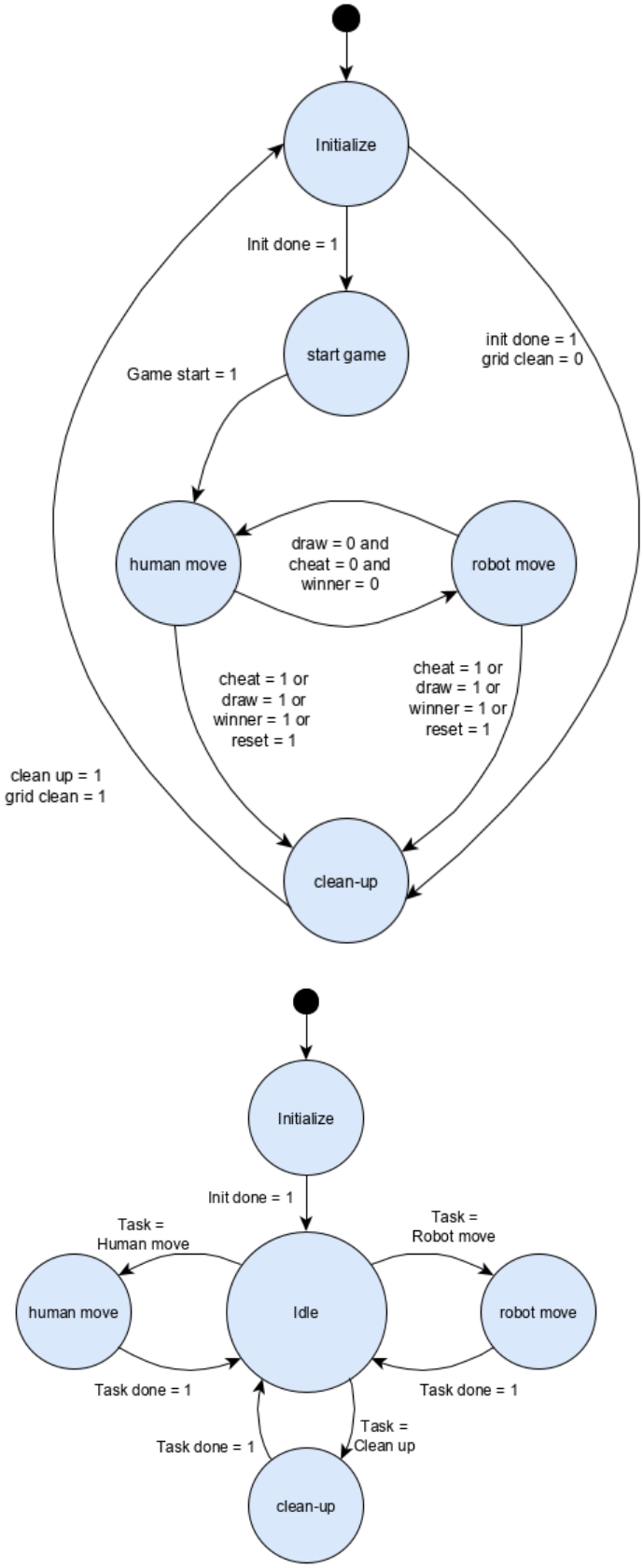



*Figure 18 State machine real-time processing*

Figure 17 can be shown in a sequence diagram. This is to indicate what happens in each state and how the communication between software modules goes. In Figure 19 the robot move is elaborated. Theother sequence diagrams can be found in the SAD listed in Annex III. Software Architectural Document.

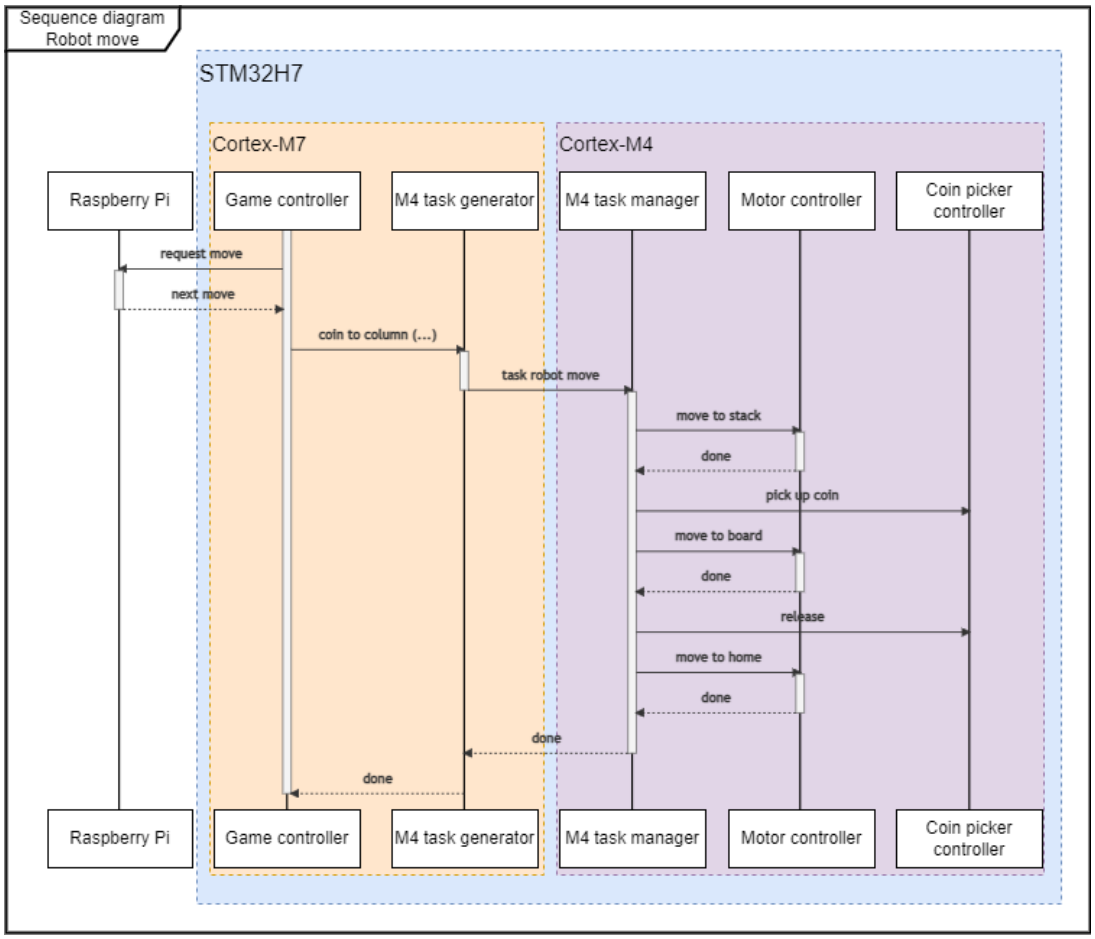



*Figure 19 Sequence diagram robot move*

The robot move starts with the Game controller requesting the next move to the Raspberry Pi. The Raspberry Pi returns the next move after which the Game controller passes to the M4 task generator whatevery command needs to be made. Once the command is created, the M4 task generator sends the command to the M4 task manager on the Cortex-M4. To play a chip , the robot must first go to the place where the chips are stored. As soon as the robot has arrived at the storage, a sheet must be picked up. Now the robot can move to the board and then release the chip. After that, the robot has to go back to its "home" position. Then the M4 task manager informs the M4 task generator that the turn is complete. Finally, the M4 task generator gives the signal that the Game controller can go to the next state.

## 5.4 Deployment view SAD

The chapter "Deployment view" describes what is needed to actually implement the software. A microcontroller is used for this. The most important component is the development board with the STM32H7 dual-core microcontroller. Furthermore, the pin out is defined. The pin out indicates which physical outputs from the microcontroller go to the hardware components and also whiche signals / protocols are used (the green pins in Figure 20) to control the robot. A pin cannot output every type of signal/protocol. The pins are set up in such a way that all signals / protocols needed to run the system are available. There are also pins allocated for ethernet. This is a feature that has not yet been implemented. In the future this is the intention and then it is important that the pins needed to use Ethernet are not occupied for other tasks. The full descriptions and diagrams are given in the SAD (Annex III. Software Architectural Document).


*Figure 20 Pin out STM32H7 dual core*

## 5.5 Modular software modules SAD

Based on the requirements, a modular structure of the system was chosen. In Chapter 5. 2 describes which software modules are necessary to implement a working 4-on-1-row robot. The chapter Modular software modules explains how the modules from high level to low level (Application layer to Hardware layer, Figure 8) relate to each other. By using the embedded software layers, you create reusable modules. Representative for this are the HAL and driver layers, but also the modules from the Middleware can be reused. The coherence is visualized on the basis of diagrams. By drawing up these diagrams, it becomes clear what the scope of the different software modules is.

Generic modules have been set up to make the HAL layer even more abstract (high level, Middleware). These generic software modules of the signals/ protocols for communication with the hardware are designed in such a way that they can also be used in other systems and / or projects. The required input from generic modules, for example i2c_device, is always the same. The processing of the data is therefore standardized. In addition, there are modules that are system specific. These modules are modular because they separate functions from each other. After all, the modules do not have to be aware of each other's functionalities or tasks to do their job. Because tasks in the software are separated from specific modules, the system meets the required overview and insight and the simplicity of implementation. With a modular structure, several people can also work on the system at the same time. The modules have a fixed interface so that each module can be adjusted independently. Modular software modules, for example, allow a sensor to be replaced during an upgrade, without changing all the code. Only the module with the sensor-specific details needs to be adjusted.

**Modularity Coin color seperator**

In Figure 21 is the diagram displayed for the software modules of the Coin Color Separator. The modules from the chapter 5.2 are combined with the standard HAL. The block "GPIO" and "i2c" belong to standard ST HAL software modules where the configuration becomes specific components Generated. These blocks initialize the GEneral Purpose I/O pins (GPIO) and the i2c bus. The other blocks are software modules that cultured should be. The block "i2c device" is a generice module that youMakes becomes to for it ensure that each sensor connected via the i2c bus only has to indicate whether data needs to be sent or requested. For example, the Modules for the sensor not all standard functions to be aware of an i2c connection because this becomes handled by the block "i2c device". By this Modular approach can be the Modules reused and is it simple to get an extra module, for example for an extra i2c sensor.

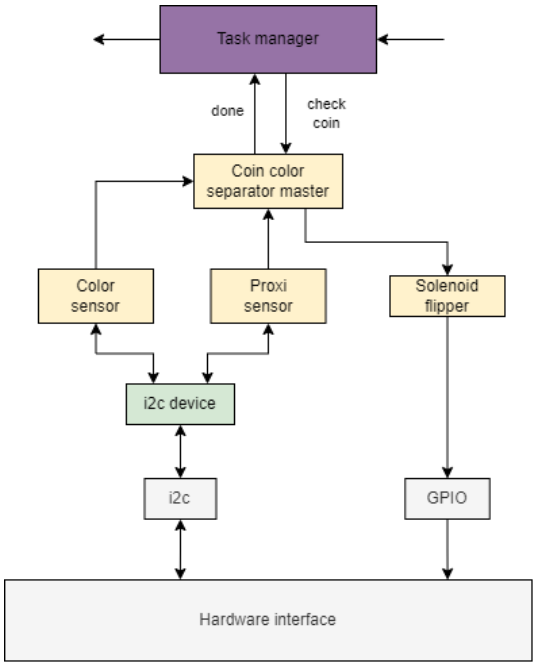



*Figure 21 Software layers Coin color separato*

**Modularity Cortex-M7**

The Cortex-M7 is used for the game determination of the 4-on-1-row robot. A Round robin with interrupt method was chosen Figure 17) (chapter 6.1). The "Game controller" is the module within the Cortex-M7 that runs the state machine. The Game controller is responsible for keeping track of the state the system is in and, using inputs, determining what the next state should be. Other modules have been chosen for the other tasks of the Cortex-M7 . The modules "UART controller", "Cortex-M4 task generator" and the "Game end unit" ensure that all input/output used by the Game controller are separated from each other. These modules handle all necessary communication between the Game controller and the hardware so that the Game controller only needs to call these modules to perform a task.

**Modularity Cortex-M4**

The Coretex-M4 runs all the software that controls and handles the hardware. In order to create the requested modularity within this core, it was decided to develop its own software Figure15for each unique hardware block (Figure 15). In order to properly manage all the software modules of the hardware, it was decided to implement a "Task manager". Dto use a "Task manager" to have functionalityand other components to be built modularly. After all, the functionalities do not have to know about each other's existence/status. That information is maintained by the Task Manager. The Task manager will also work by means of a state machine.

## 5.6 Conclusion

Drawing up a structured software architecture should lead to a future-proof 4-on-1-row robot. It should be possible to easily make changes and adjustments to requirements without frustrating the operation of the system. By drawing up a SAD, the assignment to develop a structured software architecture is fulfilled. The template of arc42 has ensured that all necessary diagrams, functionalities and design choices have been carefully made, described and displayed. Every software or hardware developer who will work with the 4-on-1-row robot can understand how the system works and everysoftware module is implemented. In addition, modularity creates a robust system that can be easily adapted. It results in reusable modules and faster development time.

# 6 Deployment and testing

To test the software architecture, some unit tests and demos have been developed. The unit tests validate the necessary software modules of the BSP. The demos must endorse the modularity and operation of the operating system software and demonstrate that the right design choices have been made. Initially, it was decided to test the foundation of the BSP, namely dual-core communication. After that, the BSP modules were tested separately for i2c, UART and Pulse Width Modulation (PWM) communication. In parallel, demos have been developed in which, step by step, based on the tested and validated BSP modules, we worked towards the complete control system of the 4-on-1-row robot.

## 6.1 Implementation method

Core distributionsection 6. 2Dual-core communicationIn addition, a method must be chosen with which the software will be executed (software loop). Four possible options have been examined (Annex V. Implementation methods for a software loop

1. Round robin
2. Round robin with interrupt
3. Function Queue Scheduling
4. Real Time Operating System (RTOS)

The starting point for an optimal design is the choice of the simplest implementation method. The method must meet the performance requirements of the system.

Within the project, a Round robin with interrupt implementation method was chosen. Theimplementation of the current 4-on-1-row robot has also been done with a Round robin with interrupt. It meets the performance requirements. The order of the phases (state machine) is always the same and can be managed by a Round robin. However, because there are priority tasks such as a hard timing deadline for the control loop of the engines, the throw-in of a chip and dual-core communication, interrupts are required. The deadline can be captured by means of a timer ISR. What must be taken into account is the false data that can arise from the use of interrupts. Annex VI. False data

## 6.2 Dual-core communication

Dual-core communication is a crucial part of the system. The two cores run independently of each other. By default, there is no built-in communication between the two cores, they cannot exchange information with each other. Because the Cortex-M7 performs the game handling and the Cortex-M4 the real-time processing, communication is needed. Communication and synchronisation between the two cores must therefore be resolved.

The basis for realizing communication is a shared memory. As shown in Figure 22 , a notification line is also used to exchange data between the two cores. The notification line ensures that access to the shared memory is synchronized between the two cores. The synchronization prevents corruption of the data and that both cores write in memory at the same time.

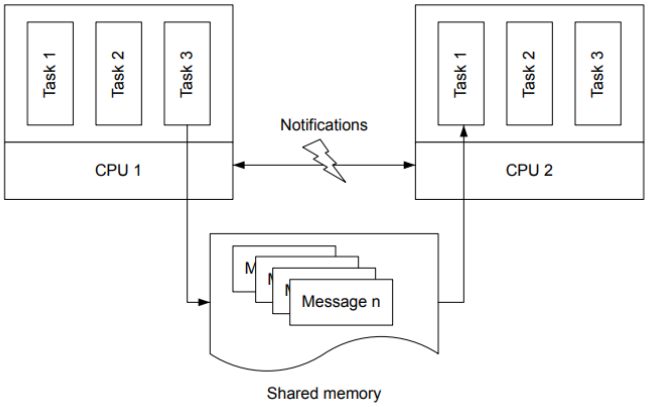



*Figure 22 Dual-core communication[17]*

### 6.2.1 Shared memory

The first step in implementing the communication is to choose a piece of shared memory that is available and accessible for both cores. The STM32H7dual-core has a "symmetric memory mapping" as a memory architecture. This architecture ensures that a large part of the available memory forboth cores is accessible. Table 6 indicates that about 82% of the memory is readily available for both cores. The Cortex-M4 needs a Master Direct Memory Access (MDMA) controller to access the ITCM and DTCM memory.

*Table 6 Memory Access[17]*

| Core | Cortex-M7 | | | Cortex-M4 | | | Cortex-M7/4 | |
|---|---|---|---|---|---|---|---|---|
| | D1 domain | | | D2 domain | | | D3 domain | |
| | ITCM | DTCM | AXISRAM | SRAM1 | SRAM2 | SRAM3 | SRAM4 | BKSRAM |
| Cortex-M7 | Yes | | Yes (cacheable) | | | | | |
| Cortex-M4 | Indirect (via MDMA) | | Yes | | | | | |

Within this project, SRAM4 was chosen as shared memory. SRAM4 is located in the D3 domain that remains available for both cores , even if oneof the cores is in "low-power" mode or even off. The D1 or D2 domains are only available if the corresponding core is active. In addition, SRAM4, with a size of 64Kbytes, has enough space to store the data. In Annex IV. Memory and bus architecture STM32H7 dual-core is a schematic overview of all three domains.

### 6.2.2 Notifications

The second step in implementing the communication is to choose a mechanism for notifying the cores. Two notification mechanisms have been investigated. The choice is determined based on the most appropriate way for the 4-on-1 row deployment. By choosing a notification from one core to another, consistency is guaranteed and it reduces the time that the system would otherwise spend on constantly checking whether new data is available. Furthermore, it ensures synchronization.

The STM32H7 dual-core has two solutions. Both solutions use a "hardware interrupt". This is necessary for a real-time notification between both cores. The two solutions have the following properties:

1. *EXTI software interrupt and event registers, CPU send-event instruction (SEV).*
   EXTI and SEV are simple interrupts. Figure23 shows that both cores have a direct connection to each other. The lines are connected to the Nested Vector Interrupt Controller (NVIC). The NVIC performs an ISR if an interrupt is detected.

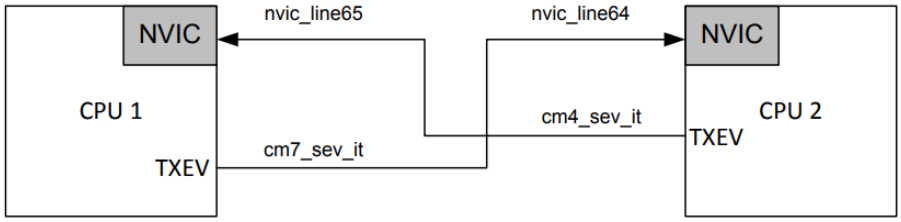



*Figure23 EXTI and SEV dual-core communication[17]*

2. *Hardware Semaphore (HSEM) free interrupt.*
   The HSEM uses an [18]HSEM controller (Figure24). An HSEM works with a "lock/ free" mechanism. This means that a core can lock an HSEM. As long as the HSEM is locked, the other core cannot access the HSEM. Only when the core unlocks the HSEM is an interrupt sent to the other core . In the core that receives the interrupt, an ISR is executed by the NVIC. From the moment on, the HSEM. This is a "handshake" so that the cores do not perform a task at the same time and it also counts as a notification mechanism. Furthermore, with an HSEM it is possible to give the interrupt an ID between 0 and 31, so that a notification can also initiate changing commands.

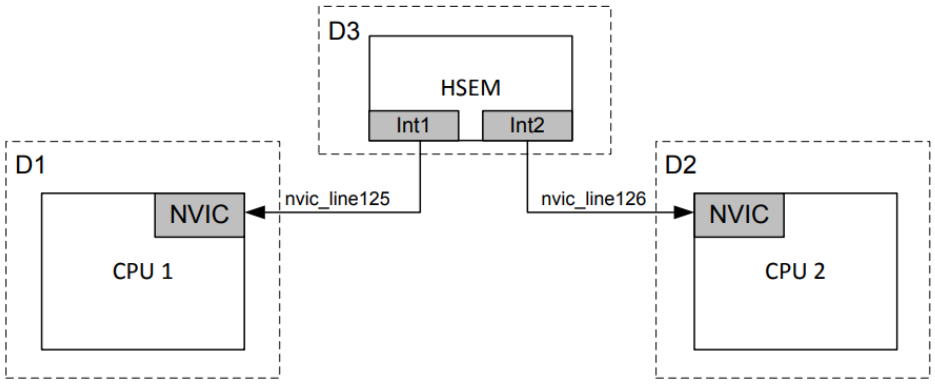



*Figure24 HSEM dual-core communication[17]*

Within the project, the mechanism of an HSEM was chosen. The HSEM has the same functionality as EXTI interrupt and SEV but also has the important advantage that an ID can be given. The tasks that the Cortex-M4 must perform are predefined and provided with a unique ID (0-31). In this way, the Cortex-M4 knows which task to perform when the interrupt arrives. Figure Figure25implementation of the HSEM within this project.

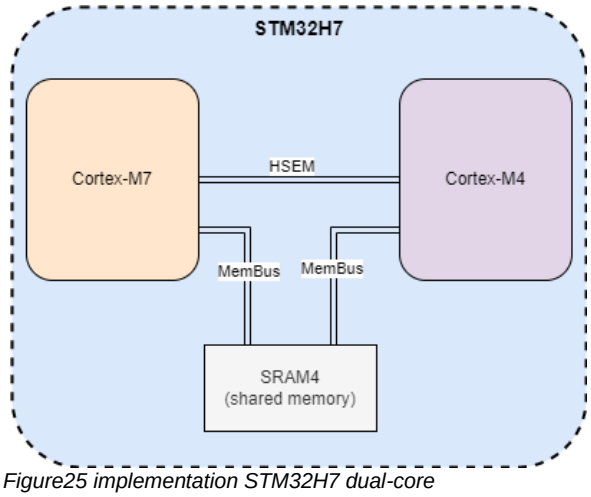



*Figure25 implementation STM32H7 dual-core*

### 6.2.3 Implementation of dual-core communication

The first part of the BSP involves implementing the dual-core communication application. Figure 26

provides a schematic overview of the components and how they are connected to each other.

The application is implemented in two ways.

1. The sharing of information between the Cortex-M7 and Cortex-M4 is done via shared memory (SRAM4);
2. The two cores can peer at each other through different HSEM's tasks.

In both cases, the Cortex-M7 has a delay of 200ms per software loop and the Cortex-M4 one of 400ms. The red LED (Cortex-M7) flashes at a frequency of 2.5Hz.

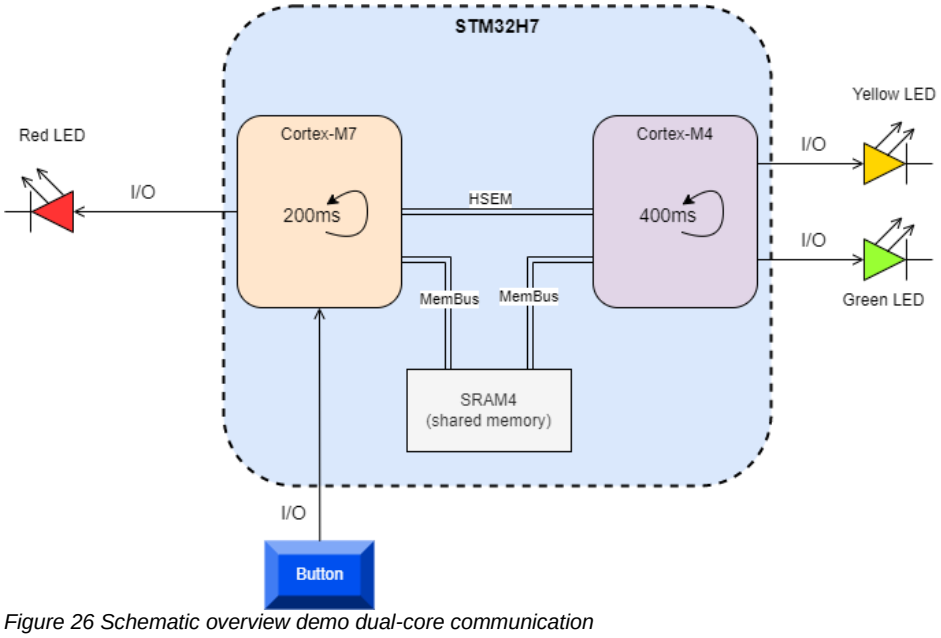



*Figure 26 Schematic overview demo dual-core communication*

*Ad 1) Shared memory unit test*
When testing shared memory, the Cortex-M4 will check which value is in SRAM4. Table Table 7 value in shared memory that creates the pattern of the yellow and green LED. Each press of the button (Cortex-M7) increases the data in SRAM4 until finally the maximum value is reached (3). After that, the data starts again at 0. In addition, with every press of the button, an HSEM is also sent to the Cortex-M4. When receiving the HSEM, the Cortex-M4 checks which data is in shared memory and executes a pattern.

*Table 7 Possible patterns of shared data*

| Data SRAM4 | Yellow LED | Green LED |
|---|---|---|
| **0** | At | At |
| **1** | Flashes at a frequency of 1.25Hz | From |
| **2** | From | Flashes at a frequency of 1.25Hz |
| **3** | Flashes at a frequency of 1.25Hz | Flashes at a frequency of 1.25Hz |
| **unknown** | From | From |

*Ad 2) HSEM unit test*
Upon testing, the Cortex-M4 will receive different HSEM. Table Table 8 which HSEM belongs to a particular pattern of the yellow and green LED. The button connected to the Cortex-M7 ensures that all HSEMs are passed through this test.

*Table 8 Possible patterns of HSEM*

| HSEM | Yellow LED | Green LED |
|---|---|---|
| **HSEM-1** | At | At |
| **HSEM-2** | Flashes at a frequency of 1.25Hz | From |
| **HSEM-3** | From | Flashes at a frequency of 1.25Hz |
| **HSEM-4** | Flashes at a frequency of 1.25Hz | Flashes at a frequency of 1.25Hz |
| **unknown** | From | From |

During the implementation of the dual-core communication, there was a problem of data coherence due to cache. As a solution to this problem,the project has chosen a Memory Protection Unit (MPU). An MPU can make part of the memory non-cacheableComplication of dual-core communication

With the successful implementation of both tests, it has been shown that the HSEM and shared memory meet the requiremtents of dual-core communication. This concludes a subsectionof the BSP.

## 6.3 BSP modules for hardware

In addition to the dual-core communication, unit tests were carried out for i2c, UART and PWM. In the following paragraphs, the unit tests are merged into demos. First i2c & UART were added to the dual core communication unit, then the demo was further expanded with PWM. For example, we work on a tape-by-tap basis to the complete control system of the 4-on-1-row robot.

### 6.3.1 Dual-core i2c UART demo

The demo is aimed at demonstrating modularity through the implementation of i2c and UART modules. Figure 27 gives a schematic overview of the demo.

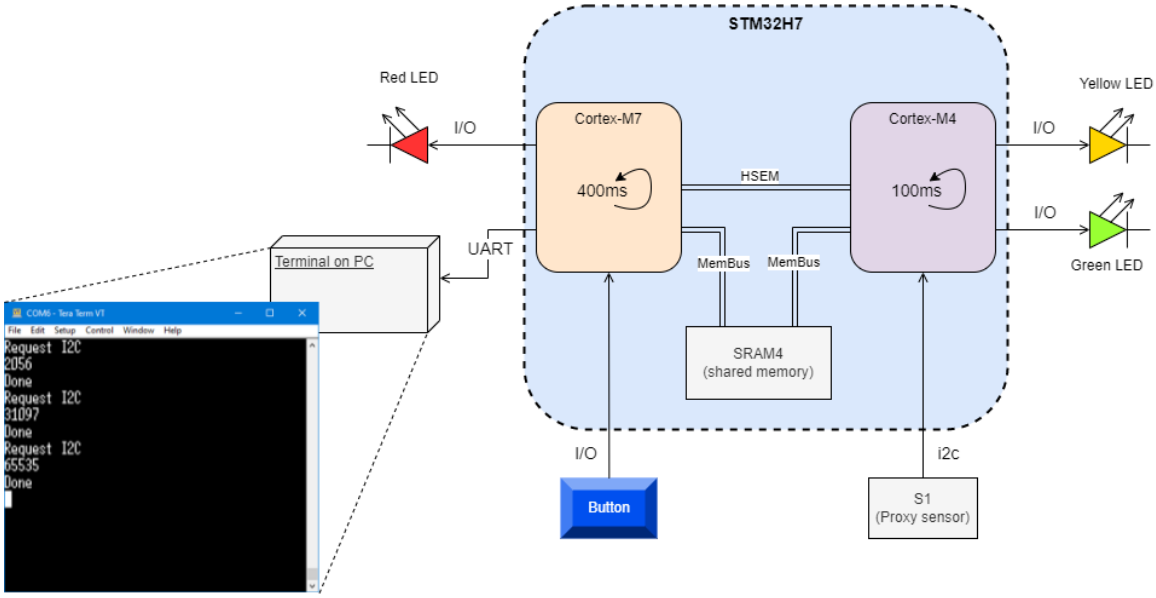



*Figure 27 Schematic overview demo dual-core i2c UART*

Before the demo was drawn up, the separate unit tests of i2c and UART were first carried out. The BSP module "i2c dev" and "UART controller" are generic (seeFigure 28). The i2c dev module ensures that any sensor only has to indicate whether data needs to be sent or requested. The UART controller module ensures that data can be sent and received via UART. The i2c unit test was performed with a random i2c and the UART unit test with a connection to a laptop. Both unit tests have shown that the BSP modules work and can be applied in a demo (and in the final control system).

After it has been shown that the modules i2c and UART function separately, the intention in the demo is to merge them. The Cortex-M7 has a 400ms delay per loop and flash the red LED at a frequency of 1.25Hz. The Cortex-M4 has a 100ms delay per loop and release the green LED blink with a frequency of 2.5Hz. Furthermore becomes bee any interval of the Cortex-M4 the value of the i2c sensor checked via the BSP module. The sensor is a Proximity sensor (S1). If the value of the S1 above a set limit, the yellow LED lights up. When the value from S1 again falls below the limit value goes the yellow LED from. If the Button on the Cortex-M7 there will be a verzoek, via an HSEM, to the Cortex-M4 go to the data of S1 to be requested. In the Cortex-M4, this HSEM is received and the data from S1 is read out. The data is then placed in SRAM4 and an HSEM is sent to the Cortex-M7 that there data is available. When the Cortex-M7 receives the HSEM, the data from SRAM4 is read and displayed on the terminal of a computer. Displaying it on the computer happened via UART BSP module. Figure 28 indicates the used software modules weather. The demo has shown that the communication between the different modules is going as expected.

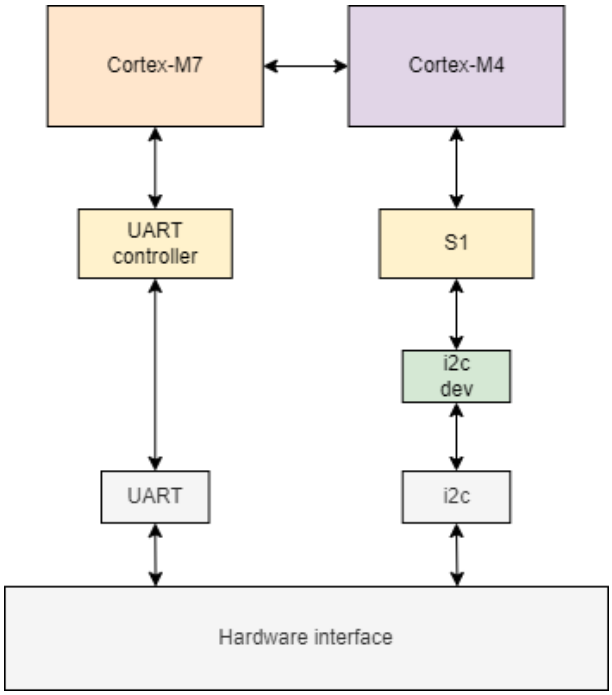



*Figure 28 Software layers dual-core i2c UART*

### 6.3.2 Dual-core i2c UART PWM demo

The final demo is an extension of the previous demo, with more modules of the 4-on-1 row robot. This test must represent the 4-on-1 row robot as well as possible so that a definitive variant of the operating system can be developed as a result of this demo. Figure 29 provides a schematic overview of the demo.

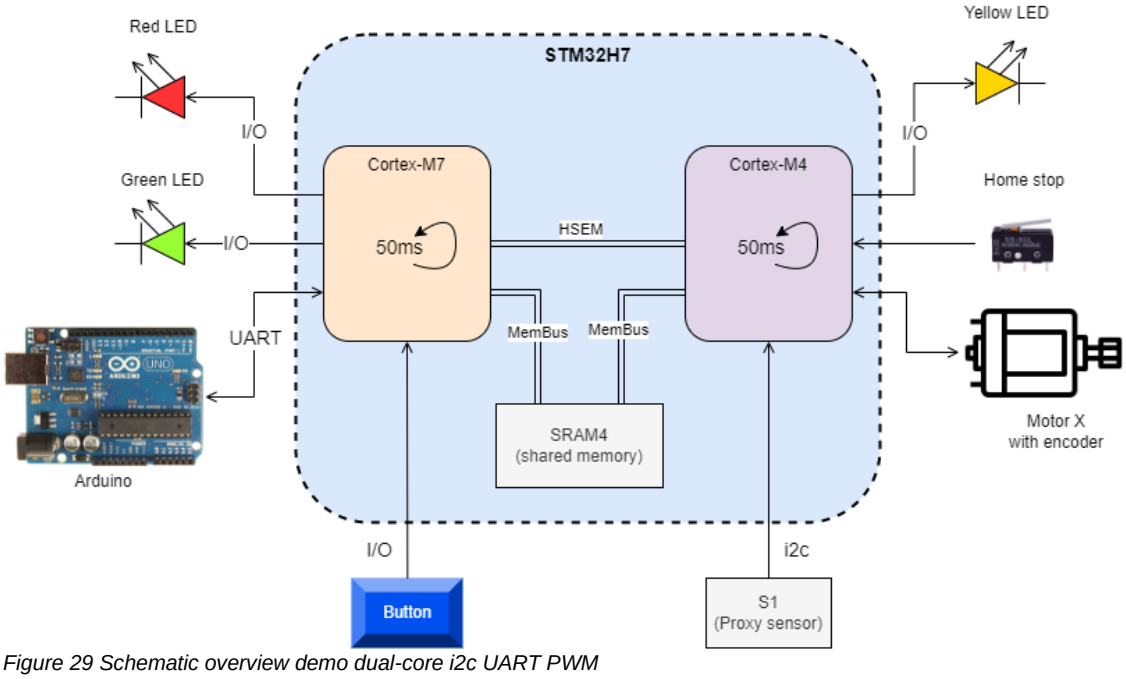



*Figure 29 Schematic overview demo dual-core i2c UART PWM*

Prior to the demo, the unit test was carried out for BSP module PWM, which takes care of the control of motorand servos. This test was carried out by first controlling one servo. The servo is demountable and the operation can easily be demonstrated. After that, the PWM was applied to the motor that is part of the physical 4-on-1 row robot. The servos and motors are expected to work so that the BSP module PWM can be applied in the demo.
Now that the PWM module has been shown to function separately, it will be inserted into the demo. The Cortex-M7 has a 50ms delay per loop. Op the core is the state machine (Figure Figure 17) of the

game controller implemented. The Cortex-M4 has a 50ms delay per loop and at each interval the value of the i2c proximity sensor (S1) is checked. The sensor S1 serves as a representation for the user detect of the 4-on-1-row robot. First, the Cortex-M7 is in the initialize state, the red LED is lit and all functions are initialized. In this state, the Cortex-M4 will perform a "homing sequence" for the motor X. Because the motor does not know where the component to be driven is physically located, it is necessary to look for a home position in advance. This way it can be kept track of where the component is located. The engine constantly turns in onedirection until the end position is reached (home stop). With this, the homing sequence is successful and the calibration phase is completed. Now the system can move on to the next state, de Start game state.

In the start game state, the button is used to start the game and the green LED flashes with a frequency of 2.5Hz. The system goes to the human move state and the player is on it. By holding a hand or object in front of S1, a "chip" is detected. The Cortex-M4 randomly generates a value between 1 and 7 to mimic the column of the board. This data is then written to the shared memory (SRAM4). Because an HSEM is released, the Cortex-M7 is informed ofthe player's input. The Cortex-M7 will then retrieve the data from SRAM4 and go to the state robot move.

In the state robot move, the column is forwarded to the Arduino, which represents the Raspberry Pi. The Arduino sends back in which column the robot should throw its chip. The Cortex-M7 then puts this data in SRAM4 and releases an HSEM so that Cortex-M4 can retrieve the data. On the Cortex-M4, the motor is controlled to go to the position of the chosen column on the X-axis. After a delay of one second, the engine goes back to the home position. Cortex-M7 will receive an HSEM that the robot has made the move and will go back to the human state.

By performing this demo , the operatingsystem of the 4-on-1-row robot is well approached. It is an approach, because not all system components are included. For example, only the engine of the x-axis was used in the implementation, an Arduino is used instead of a Raspberry Pi and the clean-up phase was not taken into account. This does not have to be an obstacle to demonstrating modularity. After all, the modules that are missing have the same principles and protocols as the modules used.

## 6.4 Conclusion

This chapter describes how the individual modules of the BSP were tested and then merged into increasingly complex demos that eventually replace the operating system of the current robot.

For the method of implementation, a "Round robin with interrupt. This is the most appropriate method for deploying software. It's easy to understand, customize, and it's transparent. The system goes through standard phases, making the Round robin with interrupt the applicable method. Because there are priority tasks, an interruption is necessary.

Initially, it was decided to test the foundation of the BSP, namely dual-core communication. The communication is robust due to the use of the HSEM and easy to maintain or adjust. After that, the BSP modules were tested separately for i2c, UART and PWM communication. The unit tests went as expected and prove that the individual BSP modules can be used for the operating system.

In parallel, demos have been developed in which, step by step, based on the tested and validated BSP modules, we worked towards the complete control system of the 4-on-1-row robot. Successive dmo's have been drawn up in which the BSP modules dual-core, i2c and UART have been merged and then a demo in which the setup has been extended with PWM. The demos are complementary. The demos went as expected, showed that the software architecture is applicable and the design choice made for modularity is the right one. It has also been demonstrated that the BSP modules can also be used in other projects.

# 7 Validation

To see if the project has been carried out properly, the requirements are validated in this chapter. All requirements with "Must" must be fully or partially implemented before the project can be accepted. Table 9 repeats Table 9 and checks the requirements. Green means completed, orange means partially completed and red means not completed. A short description explains what was needed to achieve the result.

*Table 9 Validated requirements*

| ID | Requirement | MoSCoW | ☑ | Description |
|---|---|---|---|---|
| UR.1 | The architecture of the operating system must be structured and modular in order to give hardware and software developers who are not familiar with the system a quick insight into the functioning of the robot. | Must | | This has been achieved through the SAD. |
| UR.2 | The software architecture must be future-proof so that software developers can perform effective and efficient management and upgrades. | Must | | Design choices within the SAD guarantee that the operating system can be upgraded in the future and remains reliable. |
| UR.3 | The architecture of the operating system must be logically constructed on the basis of diagrams so that software developers and testers can quickly gain insight into the functioning of the software. | Must | | In the SAD, the operation of the system is visualized in diagrams. |
| UR.4 | The coherence between software and hardware must be logically constructed on the basis of diagrams so that software developers can implement the final software. | Must | | The SAD contains diagrams that show the relationship between software and hardware. |
| UR.5 | A BSP must be made of the operating system with which the necessary hardware components of the robot can be controlled. | Must | | The BSP has been developed and the necessary hardware components of the robot can be controlled. |
| UR.6 | The components of the BSP that are necessary for the functioning of the robot must be tested independently of each other within the project. | Must | | The demos have proven that the design choices for the implementation meet the demand. |
| UR.7 | Within the new operating system, the STM32H7 dual-core microcontroller must be integrated. | Must | | With the use of STM32H7 dual-core microcontroller, this requirement is met. The implementation of the dual-core communication allows the dual-core microcontroller to exchange information between both cores. |
| UR.8 | The robot must have a calibration tool that, when initializing the system, ensures that the motors are "homed" in the Z and X direction. | Must | | The last demo contains a calibration tool that calibrates the engines at start-up (homing sequence). |
| UR.9 | A Pin-out must be drawn up with a table and diagram to ensure that hardware developers can develop a PCB design for the dual-core processor in the future. | Should | | A pin-out has been put together for implementing a custom PCB in the future. |
| UR.10 | The algorithm running on the Raspberry Pi must be integrated on the new STM32H7 dual-core. | Could | | Due to lack of time and lack of knowledge of the algorithm, it was decided, in |

| | | | | |
|---|---|---|---|---|
| | | | | consultation with the technical supervisor, early in the project to disregard this requirement. |
| UR.11 | A physical demo must demonstrate that the modular structure of the software architecture is applicable and functioning. | Could | | A demo has been made that shows the operation of all the basic functions of the software architecture. Due to lack of time, it has not (yet) been possible to get the robot working in its entirety with the new system. |

# 8 Conclusion and recommendations

For five months, ALTEN worked on the following assignment description:
*"Set up a structured and modular software architecture for the control system of the 4-on-1-row robot, with which all changes and extensions in the future can be facilitated and implement this software architecture on an STM32H7 dual-core microcontroller."*

The requirements of the assignment have been delivered within the available time. The requested structured and modular software architecture was realized by drawing up a Software Architecture Document (SAD) based on the template of arc42. The SAD describes how the system works, how it is constructed with software modules and how these modules are connected and communicate with each other. The operation of the system is visualized with diagrams, tables and diagrams.

The architecture has a logical structure. First, all software modules needed for the 4-on-1-row robot were drawn up. These are then divided into deeper and deeper layers, zooming in on the underlying modules (levels). After all levels have been worked out, we looked at how the system behaves when it is operational. The communication between modules during the operation is described and visualized. Finally, the connection between software and hardware is shown, whereby the modular structure of the system was ultimately proven in practice. By building the software architecture according to the template, it is clear to everyone how the system ultimately functions.

On the basis of demos, it has been demonstrated that the software architecture actually works. These have been implemented step by step and expanded to approximate the final system. An approach was chosen because not all modules are significant to demonstrate that the modular software architecture works. There are possibilities to expand the robot in the future with features that make the 4-on-1 row robot more attractive, smarter and faster.

The step from a single core to an STM32H7 dual-core microcontroller is an upgrade that was implemented simultaneously with the re-design of the software architecture. With the implementation of the STM32H7 dual-core microcontroller, it has been demonstrated that communication between two cores can run smoothly. It is the first time that a dual-core microcontroller is used within ALTEN. This makes this project a "prove of concept" for the use of the dual-core in subsequent projects.

Based on the results, some recommendations are conceivable. The project can be followed up by implementing all other modules for the 4-on-1-row robot. The robot now works suboptimally because all and the necessary modules have been developed, implemented and validated. In addition, the alreadygorhythm can be moved from Raspberry Pi to the STM32H7 dual-core microcontroller so that the system runs op onemicrocontroller. The dual-core processor has enough space and power to make this possible. In addition, it can be investigated which software modules need to be added to expand the 4-on-1 row robot with Ethernet or a screen.

## Evaluation

During my education I discovered that coderen is my passion. I was therefore somewhat disappointed that coding came second during my graduation. First a software architecture had to be developed, only then could programming be done. In retrospect, I learned to appreciate the order. It took me a long time to understand what exactly is meant by software architecture and how to set up such a document. Thanks to Aniel (technical supervisor) and Berend (Software architect), it became clear to me how to design and visualize software architecture in a user-friendly way. It has led me to learn a lot of new things about reducing complexity and translating an operating system into modules and diagrams. In the end, I was still able to program the various demos.

In the final phase of the project, a new consultant (without knowledge of the 4-in-1 row) went through the SAD. He indicates that he understands how the system works and needs to be implemented. This gives me the feeling that the drafted software architecture is well designed and really applicable .

I also learned a lot from the employees of ALTEN and my fellow interns. I've been able to ask questions and always get good feedback. In doing so, I was challenged to look outside the known paths. It was a challenge for me to fill out a graduation report. I find it difficult to report extensively. I should have chosen the English language better because many technical terms are in English. In terms of my communication skills, I have been challenged to regularly give a presentation to employees about the progress of my project. I'm fine with that.

Finally, I am proud of the fact that after graduating at ALTEN I can start working as a consultant at Mechatronics. I'm looking forward to that.

# Bibliography

[1] "ESCON 36/3 EC," October 2012. [Online]. Available: https://docs.rs-online.com/5cf2/0900766b811a32c7.pdf.

[2] "Encoder HEDL 5540," March 2021. [Online]. Available: https://www.maxongroup.us/medias/sys_master/root/8884124516382/EN-21-488-492.pdf.

[3] "Engine EC-i 40," May 2013. [Online]. Available: https://www.maxongroup.com/medias/sys_master/root/8806895386654/13-217-en.pdf.

[4] "Servo," October 2011. [Online]. Available: https://nl.mouser.com/datasheet/2/321/900-00005-Standard-Servo-Product-Documentation-v2.-462659.pdf.

[5] "Powersupply," October 2020. [Online]. Available: https://nl.mouser.com/datasheet/2/260/mwec_s_a0011714497_1-2274579.pdf.

[6] "Vaccuum pump," [Online]. Available: https://www.sparkfun.com/datasheets/Robotics/Other/spec%20sheet.jpeg.

[7] "Solenoid," March 2018. [Online]. Available: https://nl.mouser.com/datasheet/2/737/Adafruit_05132020_413-1858436.pdf.

[8] "RGB sensor," August 2012. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf.

[9] "ALTEN Netherlands," [Online]. Available: https://www.alten.nl/.

[10] "arc42," [Online]. Available: https://arc42.org/why.

[11] "HAL," [Online]. Available: https://nl.wikipedia.org/wiki/Hardware_Abstraction_Layer.

[12] "STM32H755ZI," [Online]. Available: https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32h7-series/stm32h745-755/stm32h755zi.html.

[13] "ST Microelectronics," [Online]. Available: https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32h7-series/stm32h745-755/stm32h755zi.html.

[14] "STM32CubeIDE," [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html#get-software.

[15] "FSM," [Online]. Available: https://en.wikipedia.org/wiki/Finite-state_machine.

[16] "Sequence diagram," [Online]. Available: https://en.wikipedia.org/wiki/Sequence_diagram.

[17] "ST dual-core communication," [Online]. Available: https://www.st.com/resource/en/application_note/an5617-stm32h745755-and-stm32h747757-lines-interprocessor-communications-stmicroelectronics.pdf.

[18] "HSEM," [Online]. Available: https://www.st.com/content/ccc/resource/training/technical/product_training/group0/2a/6a/df/e1/3b/52/48/b7/STM32H7-System-Hardware_Semaphore_HSEM/files/STM32H7-System-Hardware_Semaphore_HSEM.pdf/_jcr_content/translations/en.STM32H7-System-Hardware_Semapho.

[19] "Intrerrupt," [Online]. Available: https://nl.wikipedia.org/wiki/Interrupt.

[20] J. A. Cook and J. S. Freudenberg, "eecs," 2008. [Online]. Available: https://www.eecs.umich.edu/courses/eecs461/lecture/SWArchitecture.pdf.

[21] "Cache," [Online]. Available: https://nl.wikipedia.org/wiki/Cache_(tijdelijk_geheugen).

[22] "microchip," [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Managing-Cache-Coherency-on-Cortex-M7-Based-MCUs-DS90003195A.pdf.

[23] "MPU," [Online]. Available: https://en.wikipedia.org/wiki/Memory_protection_unit.

# Attachments

## I. Originality statement

Versie: 6-6-2022 15:45

Fontys
Hogescholen

**ORIGINALITEITSVERKLARING**

bij het afstudeerrapport met de titel :

4-op-1-rij robot -
Het ontwerpen van een gestructureerde en modulaire software architectuur

Hierbij verklaar ik dat het ingeleverde rapport zoals hierboven is genoemd, origineel * is: het is door mij, de ondergetekende, persoonlijk opgesteld en opgemaakt.
Om dit stuk te kunnen opstellen heb ik zelf de benodigde onderzoeken uitgevoerd.
Daar waar ik gebruik heb gemaakt van andermans werk, heb ik dat aangegeven bij het betreffende stuk tekst ** en in de literatuurlijst.

Datum : 7-6-2022

Naam : Pascal Faatz

Handtekening student:

.....

* De Hogeschool heeft de beschikking over controlesoftware m.b.t. originaliteit. Zij behoudt zich het recht om deze software in voorkomende gevallen in te zetten
** Letterlijk overgenomen werk dient meteen vóór die tekst begint, te zijn voorzien van de bronvermelding: de titel van het werk waaruit geciteerd wordt alsmede naam van de auteur.
*** Verplicht opnemen in het verslag

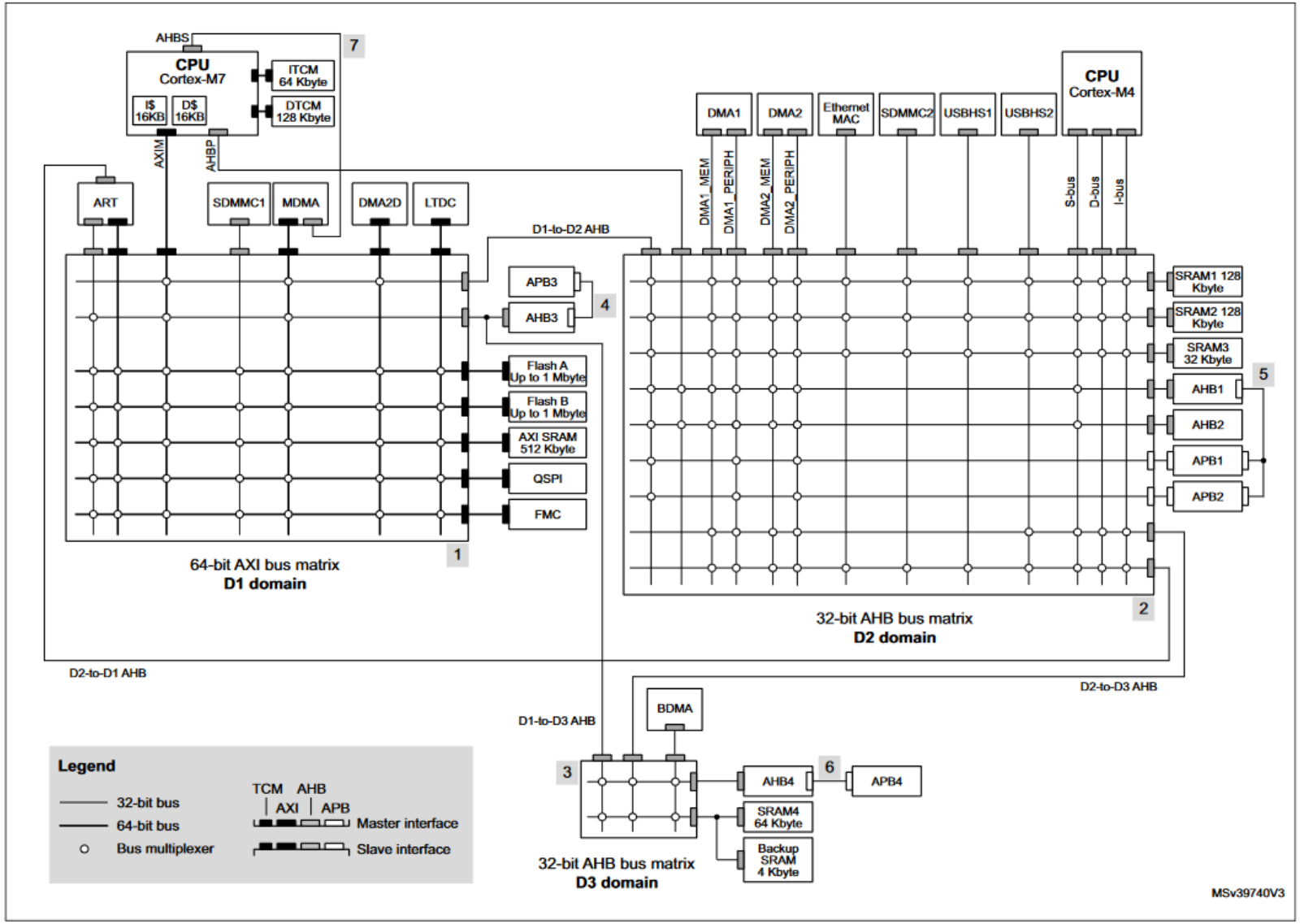
Het betreft hier:

| Hoofdstuk | Geschreven door |
|---|---|
| 1 t/m 8 | Pascal Faatz |
| | |
| | |
| | |

## II. Plan of action

## III. Software Architectural Document

## IV. Memory and bus architecture STM32H7 dual-core



## V. Implementation methods for a software loop

It is important to choose a good method so as not to make the system too complex. Four possible options have been explored:

1. Round robin
2. Round robin with interrupt
3. Function Queue Scheduling
4. Real Time Operating System (RTOS)

**Round robin**

The simplest implementation of a software architecture is the "Round robin". Round robin performs tasks one after the other and when they have all been executed, the system starts again (also called a loop). Figure 30 represents Round robin. Tasks 1 through 4 are run serially. There are many examples where this method is sufficient. Think of a vending machine, ATM or an oven. All systems where the processor has enough time to go through all the tasks and the user does not notice a delay between requesting a task and performing it (think of the time between pressing a button on the oven and updating the screen) are suitable for a Round robin implementation.

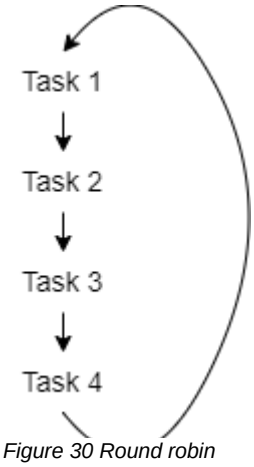



*Figure 30 Round robin*

In most cases, a Round Robin method. The great advantage of the Round Robin The method is that it is very simple and easy to maintain for small, compact systems with not too many tasks that need to be performed. The method also has limitations. If a device has to perform a task faster than the system goes through the loop, the system stops working properly. At worst, the total time the system does is about one run time of all functions within the tasks added together. One Round Robin is also not robust. As soon as a task is added to the loop, one of the other tasks may no longer meet its timing schedule. This causes problems for the system, for example that the screen refreshes too slowly in the case of an oven. The latter can be limited by having the time-bound tasks come back more often in the course.

**Round robin with interrupt**

The "Round robin with interrupt" adds an extra step to the performance of the Round robin method. Here, important tasks within the system are called by an interrupt (often a "hardware interrupt") and performed by an Interrupt Service Routine (ISR).

One hardware Interrupt is an interruption generated by an external signal, such as a button Press, timer or data on a bus. Once the interruption locationfindt is in the Core, where the signal goes, an ISR is performed. This ISR interrupts the course and executes the specific code contained in the ISR. Afterwards the process is resumed in the course of the [19]. Figure 31 is a schematic representation of an ISR.

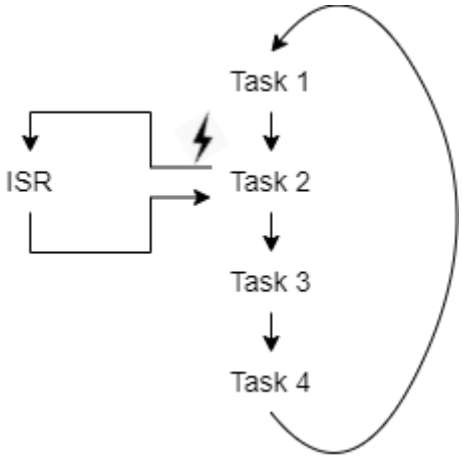



*Figure 31 Round robin with interrupt*

The big advantage of a Round robin with interrupt is that the interrupt ensures that tasks with a high priority are performed immediately and are not dependent on going through the loop. Because the method is derived from the Round robin, the implementation is easy and easy to maintain. Inserting an interrupt can also cause a problem. If a task is in the process of a calculation, and it is interrupted by an interrupt that refreshes the data of the calculation, this can lead to incorrect results (Appendix VI. False data

**Function Queue Scheduling**

The "Function Queue Scheduling" method, like the Round robin with interrupt, uses interrupts. The interrupts are provided with a priority level within this method. As soon as an interrupt calls an ISR, it is queued. This queue runs through the course of the system in order from high-priority ISRs to low-

priority ISRs. The advantage of this method is that you can give priority to certain interrupts. The disadvantage is that this method is more complicated than the methods mentioned above. In addition, False data can also occur with this method. It can also happen that an ISR with a low priority can never be executed because an ISR always intervenes with a higher priority.

**Real Time Operating System**

A "Real Time Operating System" RTOS runs on the basis of "tasks". Each task has its own function and priority. No loop is used, making it easy to add and remove tasks. Based on the priority of the tasks, the RTOS determines which task should be carried out first. A task can be in one of the following states:

1. **Running**
   The task is performed by the processor. Only one task can be performed at a time.
2. **Ready**
   All data is present to perform the task when the processor is available. Multiple tasks may be available and the processor determines
   based on the priority, in which order the tasks are executed.
3. **Blocked**
   A task is "blocked" if it does not yet have all the data to be executed or it is in an error state.

The part that checks the status of the tasks is called the "Scheduler". The scheduler's settings are simple. The only thing it checks is the priority of a task and whether it is in the ready state. A task can put itself in the blocked state when there is no data and unblock itself when the data is available again. It is then up to the scheduler to move the tasks, based on priority between the ready and running state. The operation of the schedulerFigure 32. The advantage of an RTOS is that the response time is very short and that the system is flexible. The downside is that it is a complicated method to apply. There must be sufficient knowledge about the system, in particular the timing of the tasks. An RTOS also takes up a lot of memory, which is not always available in an embedded system[20].

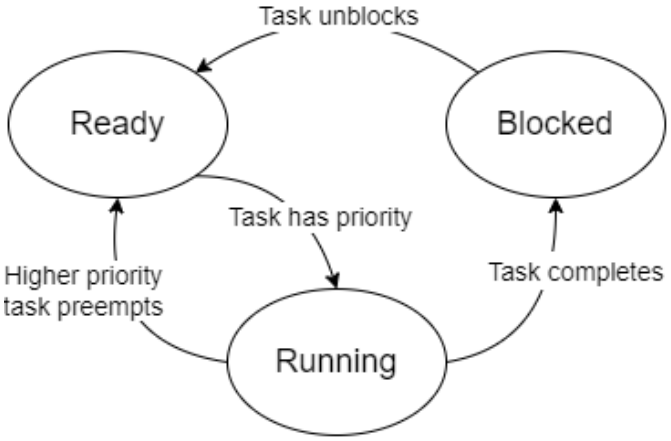



*Figure 32 RTOS Scheduler*

## VI. False data

False data can be created by using interrupts. When data is shared between tasks that work with different timing, this must be taken into account. Suppose a piece of code that describes retrieving data from an i2c sensor:

```
Int i2c_sensor;

ISR_read_i2c(Void) {
        i2c_sensor = read_i2c_Sensor();
}

int offset = x;
Int newValue;

void main(void) {
        while(1) {
                ...
                newValue = i2c_sensor - offset;
                ...
        }
}
```

An interrupt can interrupt the tasks performed in the course (while(1)) at any time. In this case, the interrupt requests the data from a sensor connected via an i2c bus. This happens, for example, when new data is available from the sensor. This interrupt can also take place when the loop is busy calculating newValue. In this case, the data with which the newValue is calculated is adjusted. As a result, it is not certain whether the value of newValue is correct and false data is created.

The code written in C is not executed as such by the microcontroller. The microcontroller performs binary machine language. To keep it readable, this can also be put in "assembly language". Assembly laguage represents the incomprehensible zeros and ones of the binary machine language in readable functions that the microcontroller performs. Assembly language actually reflects how the system acts at the registry level. An assembly language instruction usually consists of three parts:

1. **Label**
   The memory address where the code is located.
2. **On-code**
   Abbreviation for the instruction to be executed.
3. **Operands**
   Registers, addresses or data to which the instruction is applied.

A few examples are given below.

```
add R7, R8, R9; Add the data in registers 8 and 9 together, place the result in register 7.
and R2, R5, R3; Bitwise AND the data of register 5 and 3,
                Place the result in register 2.
LWZ R6, OX4(R5);    Laad the data on the memory address  or the sum of register 5
                    and 0x4 in register 6.
lwzx r9, r5, r8;    Laad de data on the memory address of the sum ofan register 5
                    and 8 in register 9.
STWX R13, R8, R9; Save the data in register 13 in the memory address of the sum of
        registers 8 and 9.
```

The important point with respect to false data is that these assembly operations cannot be interrupted. This is because it involves fundamental machine activities. The following assembly statements represent the line C code **newValue = i2c_sensor – offset;** from the first block:

```
lwz     R1, 0(R12);     Put the data of i2c_sensor (0(r12)) in register 1.
li      r2, x;  Put the value of offset (x) in register 2.
subparagraphs (R3), R1, R2;     Remove the data in register  1 and 2 from each other and put the
                                result in register 3.
STWX R3, 0(R11);        Save the data to memory (0(r11)).
```

This shows that one line of C code consists of several lines of assembly code. This means that the code can be interrupted not only between lines C code but also the line C code itself because the assembly code of that line consists of several lines. As a result, false data can arise.

The solution to preventing false data is simple. The part of the code that uses data acquired by an interrupt is called critical code. The block critical code must be protected from being prevented. Disabling the interrupt for the critical block and enabling it after the critical block ensures that false data is prevented. This is because the data is first processed before new data can be received. An interrupt has a high priority for a reason, so it is necessary to look very carefully at when and where the interrupts are switched off.

## VII. Complication of dual-core communication

During the implementation of the dual-core communication, a problem has surfaced. During testing, communication between the two cores was not possible. After a long debugging, reading datasheets and searching for information on the internet, it became clear where the problem was. The Cortex-M7 is equipped with the ability to cache an area in memory. SRAM4 also falls under this area (Table 6). If the cache is enabled, a problem with the data coherence arises.

Cache is memory in which data is temporarily stored to gain faster access to the data. Essential for using cache is that it is transparent. This means that when retrieving data, it is not visible whether the data is retrieved from the original source or from the cache. This leads to the problem of data coherence. By enabling cache it is not possible to write directly to this memory, but the changed data is first cached in the cache memory. Cache has been enabled within this project on the Cortex-M7 to facilitate future upgrades that require speed and efficiency. [21]

Figure 33 shows schematically how the problem of data coherence occurs during dual-core communication. First of all, the Cortex-M7 has processed certain data and it has to go to the Cortex-M4. The Cortex-M7 writes the data to the predetermined location in the shared memory (SRAM4). In Figure 33 this is the MemX location. Because the Cortex-M7 has enabled its cache, the data will first be written to the cache. As soon as the Cortex-M4 receives a notification that the Cortex-M7 has finished writing data, it will try to read the data from SRAM4. However, there is no change in the memory block of SRAM4 (yet) because the data from the Cortex-M7 is cached memory. The same principle applies to sending data from the Cortex-M4 to the Cortex-M7. The Cortex-M4 writes data to MemX. As soon as the Cortex-M7 receives a notification that the Cortex-M4 has finished writing data, it tries to read the data from SRAM4. Because the cache memory is enabled, the Cortex-M7 reads from the cache and not from the SRAM4.

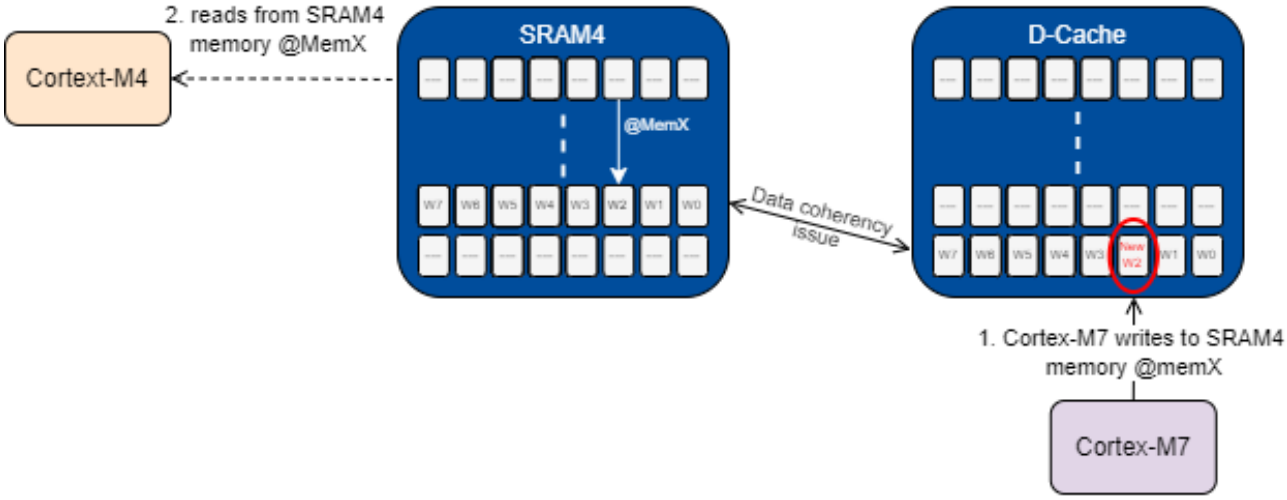



*Figure 33 Data cohorentie problem[22]*

For the problem of data coherence , two solutions are conceivable:

1. *Use "Cache Clean" and "Cache Invalidate" in code.*
   The "Cache Clean" and "Cache invalidate" are methods that can ensure data coherence." Cache Clean" is used when the Cortex-M7 wants to write to a location in memory. First, the Cortex-M7 writes to the memory but the data is cached. By using "Cache Clean", the data in the memory is updated to the data that is in the cache. "Cache Invalidate" is used when the Cortex-M7 wants to read data from memory. As soon as there is new data in the memory and the Cortex-M7 tries to read it, the data is read from the cache. If a "Cache Invalidate" is used, the cache will be synchronized with the memory. From this moment on, the new data is also in the cached memory and the Cortex-M7 has access to the data.
2. *Use Memory Protection Unit (MPU) in initialization.*
   An MPU [23] can also be used for data coherence. By means of an MPU, the behavior of part of the memory can be adjusted. It is preconfigured whether a location in memory is "cacheable". If cache is disabled, the Cortex-M7 can write directly to this part in the memory and read it out.

Within this project, an implementation of an MPU was chosen. An MPU is easy to set up in advance. Furthermore, it saves many lines of code that are otherwise needed to implement "Cache Clean" and "Cache Invalidate" functions.

## VIII. Code Dual-core communication task generator

**Common.h available for both cores.**

```
/*
 * common.h
 *
 * Created on: Apr 29, 2022
 * Author: Pascal
 */

#ifndef INC_COMMON_H_
#define INC_COMMON_H_

#include "stm32h7xx.h"

#define HSEM_TAKE_RELEASE(_id_) do { HAL_HSEM_FastTake((_id_));
HAL_HSEM_Release((_id_), 0); } while (0)

#define HSEM_WAKEUP_CPU2 0
#define HSEM_WAKEUP_CPU2_MASK
__HAL_HSEM_SEMID_TO_MASK(HSEM_WAKEUP_CPU2)

#define HSEM_CM4_TO_CM7 29
#define HSEM_CM4_TO_CM7_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_CM4_TO_CM7)
#define HSEM_CM7_TO_CM4 30
#define HSEM_CM7_TO_CM4_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_CM7_TO_CM4)
#define HSEM_ERROR 31
#define HSEM_ERROR_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_ERROR)

#define HSEM_CM4_DONE 1
#define HSEM_CM4_DONE_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_CM4_DONE)
#define HSEM_ROBOT_MOVE 2
#define HSEM_ROBOT_MOVE_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_ROBOT_MOVE)
#define HSEM_HUMAN_MOVE 3
#define HSEM_HUMAN_MOVE_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_HUMAN_MOVE)
#define HSEM_CLEAN_UP 4
#define HSEM_CLEAN_UP_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_CLEAN_UP)
#define HSEM_CHEAT 5
#define HSEM_CHEAT_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_CHEAT)
#define HSEM_COIN_COLUMN 6
#define HSEM_COIN_COLUMN_MASK __HAL_HSEM_SEMID_TO_MASK(HSEM_COIN_COLUMN)

static __attribute__ (section(". SharedBuffer"), used)) uint8_t SharedBuf[10];

#endif /* INC_COMMON_H_ */
```

**Task generator.c**

```
/*
 * task_Generator.c
 *
 * Created on: May 20 2022
 * Author: Pascal
 */

#include "task_Generator.h"
```

```
uint8_t* data;

void initTaskGenerator(uint8_t* state, uint8_t* dataIn){
        data = dataIn;
        HAL_HSEM_ActivateNotification(HSEM_CM4_DONE_MASK);
        HAL_HSEM_ActivateNotification(HSEM_COIN_COLUMN_MASK);
        memset(SharedBuf, 0, 10);
}

void taskToDo(uint8_t task){
        if(task == TASK_ROBOT_MOVE){
                memset(SharedBuf, (int)(data[0]-'0'), 1);
                HSEM_TAKE_RELEASE(HSEM_ROBOT_MOVE);
        }
        if(task == TASK_HUMAN_MOVE){
                HSEM_TAKE_RELEASE(HSEM_HUMAN_MOVE);
        }
        if(task == TASK_CLEAN_UP){
                HSEM_TAKE_RELEASE(HSEM_CLEAN_UP);
        }
}

void HAL_HSEM_FreeCallback(uint32_t SemMask){
        if(SemMask == HSEM_CM4_DONE_MASK){
                HAL_HSEM_ActivateNotification(HSEM_CM4_DONE_MASK);
        }
        if(SemMask == HSEM_COIN_COLUMN_MASK){
                HAL_HSEM_ActivateNotification(HSEM_COIN_COLUMN_MASK);
        }
}
```

# IX. Code i2c module

**I2c dev.c**

```
/*
 * i2c_dev.c
 *
 * Created on: May 2, 2022
 * Author: Pascal
 */

#include "i2c_dev.h"

HAL_StatusTypeDef i2c_CheckDev(I2C_HandleTypeDef* bus, uint8_t DevAddress){
        HAL_StatusTypeDef retFunc;
        uint8_t write_addr = DevAddress << 1;
        retFunc = HAL_I2C_IsDeviceReady(bus, write_addr, 1, TIME_OUT);
        return retFunc;
}

HAL_StatusTypeDef i2c_Transmit(I2C_HandleTypeDef* bus, uint8_t  DevAddress,
uint8_t MemAddress, uint8_t  MemAddSize, uint8_t* pData,  uint8_t
pData_size){
        HAL_StatusTypeDef retFunc;
        uint8_t write_addr = DevAddress << 1;
        retFunc = HAL_I2C_Mem_Write(bus, write_addr, MemAddress,
MemAddSize, pData, pData_size, TIME_OUT);
        return retFunc;
}

HAL_StatusTypeDef i2c_Receive(I2C_HandleTypeDef* bus, uint8_t  DevAddress,
uint8_t MemAddress, uint8_t  MemAddSize, uint8_t* pData,  uint8_t
pData_size){
        HAL_StatusTypeDef retFunc;
        uint8_t read_addr = (DevAddress << 1) |  0x01;
        retFunc = HAL_I2C_Mem_Read(bus, read_addr, MemAddress, MemAddSize,
pData, pData_size, TIME_OUT);
        return retFunc;
}
```

**Proxi sensor VCNL4010.c**

```
/*
 * VCNL4010.c
 *
 * Created on: May 2, 2022
 * Author: Pascal
 */

#include "VCNL4010.h"

void VCNL4010_Init(const VCNL4010* const self){
	uint8_t led_ma = 0x0A;
	uint8_t com_en = 0x03;
	HAL_StatusTypeDef retFunc;
	retFunc = i2c_CheckDev(self->bus, self->base_addr);

	if (retFunc == HAL_OK){
		i2c_Transmit(self->bus, self->base_addr, VCNL4010_LED_REG,
1, &led_ma, 1);
		i2c_Transmit(self->bus, self->base_addr, VCNL4010_COM_REG,
1, &com_en, 1);
		HAL_Delay(1);
	} else {
	}
}

uint16_t VCNL4010_ReceiveProxi(const VCNL4010* const self){
	uint8_t buf[2];
	uint16_t fall = 0;
	HAL_StatusTypeDef retFunc;

	retFunc = i2c_Receive(self->bus, self->base_addr,
VCNL4010_PROXY_REG, 1, buf, sizeof(buf));
	if (retFunc != HAL_OK){
		fall = 0;
	} else {
		fall = ((uint16_t)buf[0]<<8) | buf[1];
	}
	return val;
}

VCNL4010 VCNL4010_Create(uint8_t addr, I2C_HandleTypeDef* inBus){
	VCNL4010 create = { addr, inBus};
	return create;
}
```

## X. Code UART module

**UART_controller.c**

```
/*
 * UART_controlller.c
 *
 * Created on: Apr 29, 2022
 * Author: Pascal
 */

#include "UART_controller.h"

uint8_t* rxdata;

void Init_UART_controller(UART_HandleTypeDef* const RPIbus, uint8_t* data,
uint8_t* substate){
        rxdata = data;
        HAL_UART_Receive_IT(RPIbus, rxdata, 3);
        Srand(10);
}

void RPI_Request_Move(UART_HandleTypeDef* const RPIbus, uint8_t
insertColumn){
        int random = rand() % 7 + 1;
        uint8_t buf[24];
        sprintf((char *) buf, "UART com. Value : %i\r\n", random);
        UART_WriteString(&huart3, (char *)buf);
        UART_WriteValue(RPIbus, random);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
        uint8_t buf[28];
        sprintf((char*) buf, "Received value : %.*s\r\n", 2, rxdata);
        UART_WriteString(&huart3, (char *)buf);
        HAL_UART_Receive_IT(Huart, RxData, 3);
}

void UART_WriteString(UART_HandleTypeDef* const bus, char* buf){
        HAL_UART_Transmit(bus, (uint8_t*)buf, strlen(buf), HAL_MAX_DELAY);
}

void UART_WriteValue(UART_HandleTypeDef* const bus, int value){
        uint8_t buf[12];
        sprintf((char*) buf, "%i\r\n", value);
        HAL_UART_Transmit(bus, (uint8_t*)buf, strlen((const char*)buf),
HAL_MAX_DELAY);
}
```