4 in 1 row robot -

Designing a structured and modular software architecture

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

Author: Pascal Faatz

General information

Student:	Pascal Fatz 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_E4DIST
Company:	ALTEN Netherlands BV Hurksestraat 45 5652AH, Eindhoven
Technical supervisor	Aniel Shri Aniel.shri@alten.nl +31 (0)6 37 162 867
Business manager	Gijs Haan Gijs.haans@alten.nl +31 (0)6 27 025 966
School:	Fontys University of Applied Sciences The surrounding 1 5612 AP, Eindhoven
School supervisor:	Jeedella SY Jeedella j.jeedella@fontys.nl +31 088 507 81 48

Referenced Documents

ID	Reference	Title	Writer
D1	Motor driver data sheet [1]	ESCON 36/3 EC Servo Controller	Maxon group
D2	Encoder datasheet [2]	Encoder HEDL 5540	Max group
D3	Engine data sheet [3]	Maxon engine EC-i 40	Max group
D4	Servo datasheet [4]	Parallax servo	Parallax
D5	Power supply data sheet [5]	Mean Well LRS-150	Meanwell
D6	Vacuum pump datasheet [6]	SparkFun D2028 pump	SparkFun
D7	Solenoid datasheet [7]	Ada fruit 413 solenoid	Ada fruit
D8	RBG sensor data sheet [8]	Taos TCS3472	Taos
D9	Light barrier design (internal document)	SDD_photodiodeboard.docx	Arjan Verbord
D10	Custom PCB design (internal document)	013-E-0001-D.brd	Jeroen Wilbers

Preface

1-row robot - Designing a structured and modular software architecture The report serves as a completion of the Electrical Engineering course 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. At a renewed acquaintance in 2021, ALTEN turned out to have a nice graduation assignment available. I am grateful for the opportunity they gave me to complete my studies in a professional environment. During the graduation internship from February 1 to July 1, 2022, I was able to use all facilities at the ALTEN office in Eindhoven.

During my graduation I was greatly helped by Aniel Shri, technical supervisor and Gijs Haans, business manager from ALTEN. I would like to thank them for all the information, guidance and feedback they have given me. Jeedella SY Jeedella, my supervisor from Fontys electrical engineering, also contributed to the realization of the assignment. Thanks for this.

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

Then there remains a word of thanks, for you the reader of this report.
I hope you enjoy reading it.

Pascal Fatz

Eindhoven, 7-6-2022

Table of contents

List of Figures

List of tables

Resume

ALTEN is a leading international consultancy company specialized in consultancy & engineering in High Tech development environments. The company has developed a robot that can autonomously play 4-on-1 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 asks for an optimization of the quality and the preparation for an improvement in performance.

Since 2019, various engineers have worked on the robot. As a result, the software architecture has become cluttered and inconsistent. In addition, ALTEN has the desire to expand the hardware in the future. The optimization of the robot requires a complete redesign of the operating system and the replacement of the microcontroller. ALTEN's question is summarized in the following assignment: *up for the operating system of the 4-in-1-row robot, with which all changes and extensions can be facilitated in the future and implement this software architecture on a STM32H7 dual-*

The V-model was used within the project. This has ensured that all necessary project steps have been completed in a structured manner. First, the requirements were established with all stakeholders. The requirements serve as a starting point for the project, determine the choices in the preliminary investigation and serve as validation upon delivery of the project result. The most important requirements relate to a structured and modular software architecture that is logically constructed so that software developers can efficiently manage and upgrade the software. In addition, a Board Support Package (BSP) is requested to control the necessary hardware. Both must be integrated on a STM32H7 dual core microcontroller.

In the preliminary research phase, the interaction between hardware and software was analysed. A redesign 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, several methodological and technical choices were then made. The most important 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 is rebuilt. All steps of the template arc42 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 has been assessed by the stakeholders and after approval, the implementation and test phase have been initiated. In this phase, some software modules necessary for the operating system were implemented and individually tested. After that are some suitability of the software architecture

to demonstrate.

In the validation phase, it is 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. all major requirements met. However, the robot is not working yet. A few modules still need to be designed for this. This can be done easily 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: *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-*

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	Integrated Development Environment
HALL	Hardware Abstraction Layer
MoSCoW	M- Must have , S-Should have, C-Could have, W-
BBV	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
MPU	memory protection unit
PWM	pulse-width modulation

1 Introduction

A 4-in-1-row robot that never loses, ALTEN has it in house..

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

ALTEN has developed a robot in-house that can play 4-on-1 autonomously against a human opponent using an algorithm. The 4 in 1 row robot is built with industrial components to demonstrate the knowledge of different systems. The robot is used as a demonstration unit at fairs and open days, where passers-by can play a round. The player first makes an opening move. Then it is the robot's turn where the control system determines the move, picks up a chip and drops it in the chosen column of the board. The system keeps track of the game progress. 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-in-1-row robot was devised within ALTEN to challenge 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 disjointed software that is difficult to understand and therefore difficult to maintain or expand.

ALTEN wants to guarantee the quality and performance of the 4-in-1-row robot for a longer period of time. This means that the operating system must be easy and robust to adapt. To make this possible, the architecture of the operating system will have to be built up again in a modular way. The redesign of the software architecture offers the possibility to carry out a parallel hardware upgrade, whereby the single-core microcontroller is replaced by 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 operating 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 have been 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 modular and are partly generated automatically. ALTEN's choice of the STM32H7 dual-core microcontroller has 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 starts with a brief introduction to ALTEN (Chapter 2). This is followed by a description of the assignment (Chapter 3). The functioning of the current robot, the requirements of ALTEN and the methodological choices are explained in chapter 4. Subsequently, in chapter 5 the most important components of the software architecture are described. In chapter 6 the implementation and testing are explained and the requirements in chapter 7 are validated. The report concludes with some and conclusions recommendations in chapter 8.

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 1) with a total of 1,200 employees, 91% of whom are engineers [9].

Figure 1 ALTEN Netherlands offices

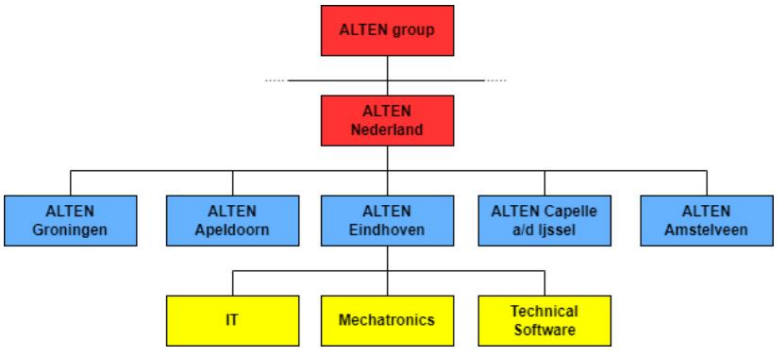


Figure 2 Organization diagram

Technology is the central pillar within ALTEN. Alten offers quality, reliability and innovations in the field of high-tech solutions. ALTEN works on behalf of companies from the automotive, high-tech (manufacturing) 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-in-1-row robot falls within the Mechatronics department. This department consists of consultants with a background in mechatronics, mechanical engineering, electrical engineering and control engineering. All specialisms are represented to provide comprehensive answers to technical questions. The employees are the most important production factor of the company. ALTEN invests in individual development, further expanding expertise and offering 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 appendix II. Action plan, Figure 4). This model has also been used for the 4-in-1-row robot project. The stakeholders are described in Table 1 of Annex II. Plan of approach. During the process, there was communication with and reporting to the technical supervisor, business manager, the customer (ALTEN) and the internship supervisor of Fontys Hogeschool Eindhoven.

3 The command

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 guideline for the project.

3.1 Problem statement

The 4-in-1-row robot is a demonstration unit to attract new customers (technique 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 cluttered and the architecture unclear. In addition, the single-core microcontroller has almost no room to upgrade the hardware.

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

3.2 Objective The

objective of the assignment is twofold. The aim of the assignment is 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 to changing circumstances and wishes. In addition, the basis is laid for improved robot performance through the use 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 in 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 screen and Ethernet connection in the future. In this way, the robot can again prove its worth as a demonstration unit and quality and performance are assured for a longer period of time.

3.3 Design assignment

The command reads as follows:

> *the 4-in-1-row robot, with which all changes and expansions can be facilitated in the future and implement this software architecture on a STM32H7 dual-core*

3.4 Scope and demarcation

- The re-design of the hardware and the mechanical part falls outside the scope of the project of the robot; -

There is no change to the 4-on-1 gameplay; - Within ALTEN, the robot is physically available for further investigation; - A dual-core microcontroller chosen by ALTEN is used; - The standard ALTEN template was used for the software architecture; - The choice of programming language C or C++ is the default 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 redesign of the robot's operating 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 associated with the assignment. These requirements have been formulated in consultation with the stakeholders. The requirements are defined in Table 1. These requirements are performed according to MoSCoW, where the requirements of the results are classified according to M-Must have S- Should have, C- Could have or W-Wo . The priority within the requirements has been determined in consultation with the ALTEN technical supervisor.

Table 1

ID	*Requirements*	MoSCoW
Requirement	UR.1 The architecture of the operating system must be structured and be modular to provide hardware and software developers who are not familiar with the system with 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 structured using diagrams so that software developers and testers can quickly gain insight into the functioning of the software.	Must
UR.4	The relationship between software and hardware should be logically structured using diagrams so that software developers can implement the final software.	Must
UR.5	A Board Support Package (BSP) should be made of the operating system with which the necessary hardware components of the robot can be controlled.	Must
UR.6	The parts 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 must be dual-core microcontroller be integrated.	Must
UR.8	The robot must have a calibration tool that is included with the initialization of the system ensures that the motors in the Z and X become.	Must
UR.9	A Pin-out must be drawn up with a table and diagram to ensure that hardware developers can work out 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 functions.	Could

The requirements define the deliverable result of the project. In the validation (Chapter 7 Validation) the table is run through again with the question whether all requirements have been fulfilled, which ones have not been met and why not.

3.6 Project approach In

order to structure the project, the V-model was chosen. The V-model is a linear development model that is divided into a number of phases. Each phase produces a result, which is required for the next phase. This is repeated for all phases, with each new result growing the system. 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).

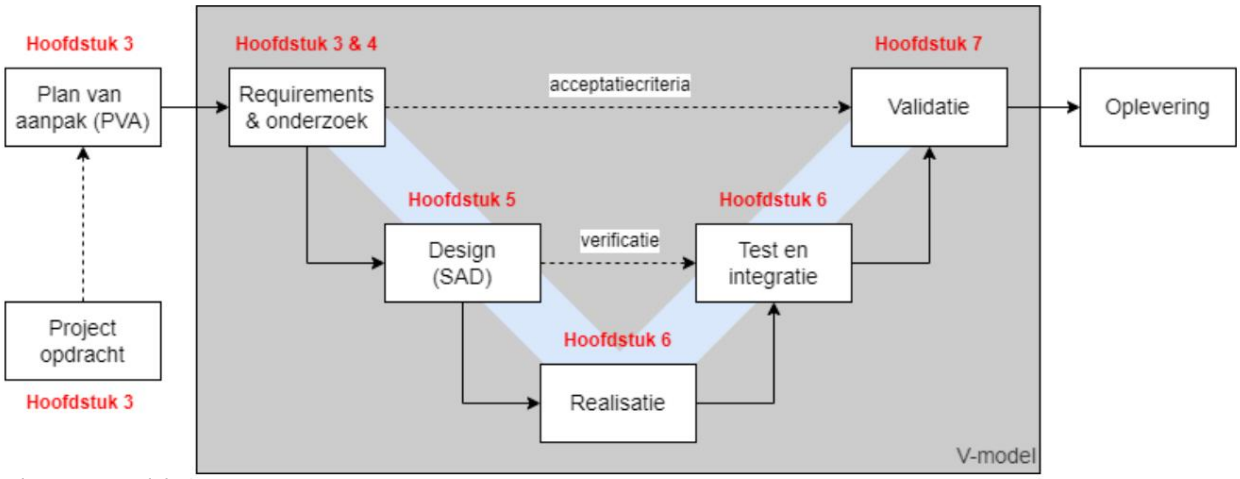


Figure 3 V model

4 Preliminary investigation

The design assignment is divided into a number of research questions that must be answered before the design can begin. The starting point is an existing robot that is controlled by an operating system. A hardware re-design of the robot is beyond the scope of this project. However, you must 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 which (hardware) parts the robot consists of. There are several documents available that record this information. 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 up 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-in-1-row robot

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

4.1.1 Gameplay

The course of 4-in-1-row is fixed. The player and the robot take turns placing chips in the game board until one of them wins, a tie occurs, or foul play occurs.
The behavior of the robot can be described in three phases: a phase in which it is the player's move, a phase in which it is the robot's move, and a phase in which the game is over 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 the electronic hardware components of the 4-in-1-row robot are present within ALTEN. The block diagram (Figure 4) shows which parts make up the robot and how they are connected to the control system. Figure 5 shows a top view of the physical arrangement. Figure 6 shows a bottom view showing the other components. Table 2 describes which components are used and what their function is in the system.

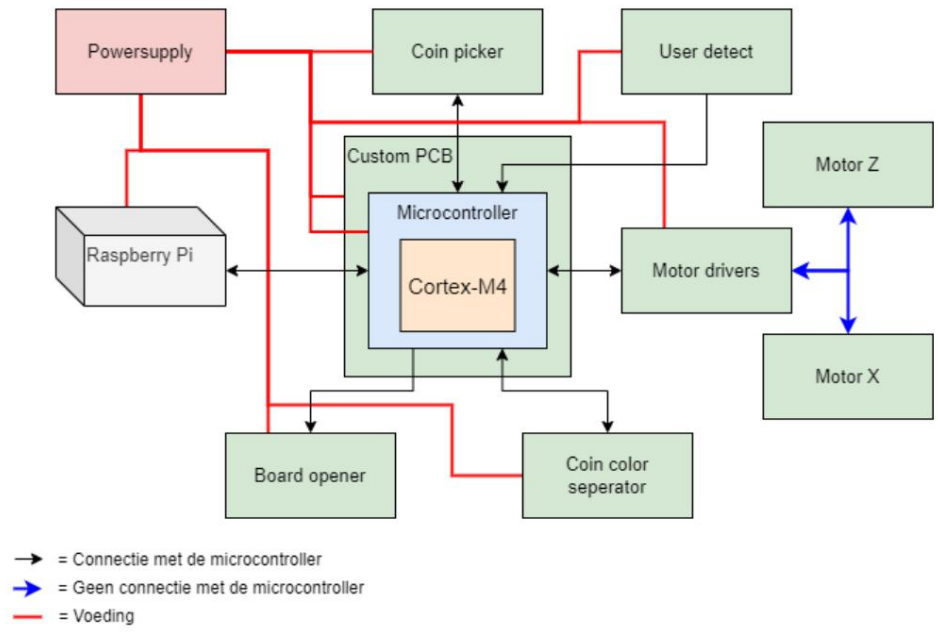


Figure 4 4-in-1-row robot block diagram

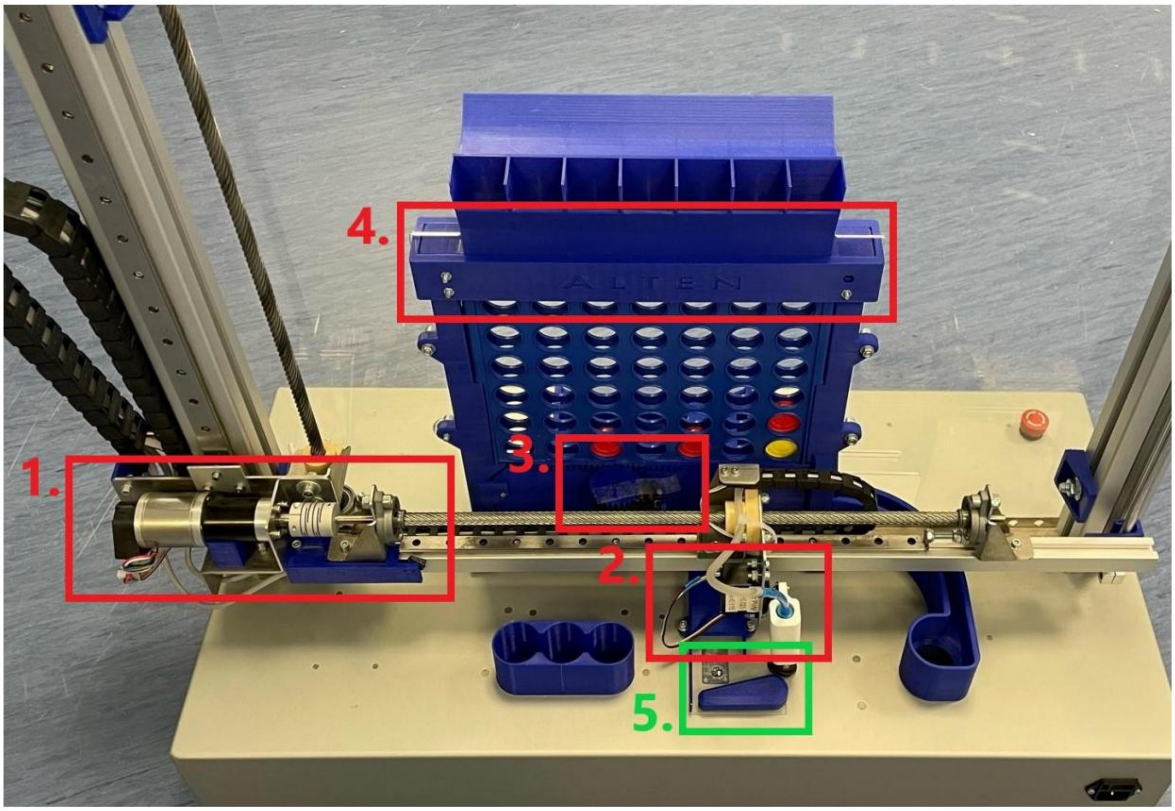


Figure 5 Top view 4-in-1 row

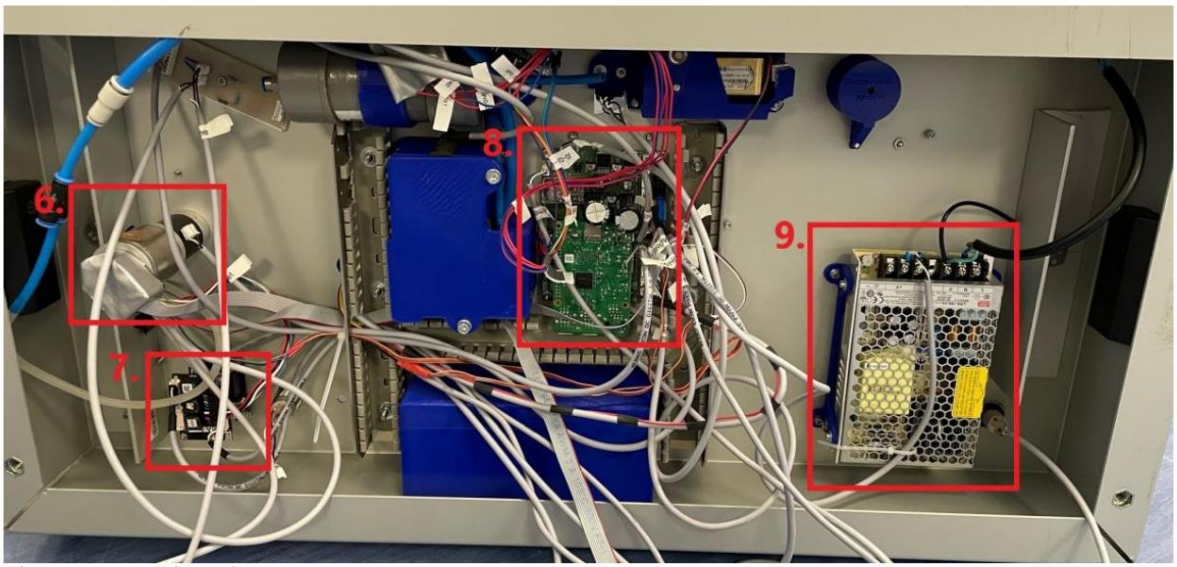


Figure 6 Bottom view 4-in-1 row

Table 2 Components 4-in-1 row

No.	module	Components	Task
1	Engine X	Motor (D3) with a rotary encoder (D2).	The Motor that controls the X-axis ensures that about this axis move.
2	coin picker	Servo (D4) and vacuum gripper (D6).	The vacuum gripper can pick up and release the chips. The servo ensures that the vacuum gripper can move.
3	Board Opener	Servo (D4).	This servo ensures that all chips can fall out of the board at the end of the game.
4	User detect	Light barrier (7x LED with photodiode, D9).	The light barrier is responsible for detecting a token 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 clean-up phase of the robot. The RGB sensor checks the color of a chip and the solenoid can, by means of a flipper, shoot chips at the player's tray.
6	Engine Z	Motor (D3) with a rotary encoder (D2).	The Motor that controls the Z-axis ensures that move.
7	Engine drivers	Motor driver X and Z (D1).	The motor drivers control 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, all signals needed to control the hardware. The Raspberry Pi uses an algorithm to calculate the robot's next move.
9	power supply	(D5)	The powersupply ensures that the entire system is supplied with a power supply.

By going through the datasheets of hardware components, the internal documentation of the hardware and software designed in-house, 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, to have a motivated team and to 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 cluttered or the architecture is simply not present. A few problems can hinder a software project: *1. Non-existence of documentation or outdated documentation.*

> The documentation of the software architecture may lag behind what has been implemented, or there may be no documentation available. Outdated or non-existent software architecture documentation creates complications during the implementation, upgrade or modification of the software because it is not clear how the software is structured and communicates.

2. Obscure documentation.

Existing documentation can also be very confusing. This documentation is often made without any sense of purpose and is made by multiple people without coordination. Cluttered architecture is difficult to understand, maintain, or modify.

3. Too much documentation.

An architecture with too much documentation is also not optimal. This can result in obstacles for future use, because searching or modifying information in the document cannot be done in a structured way.

Ad 1 and 2 apply to the current software architecture of the robot. After consultation with the software architect within ALTEN and the technical supervisor, it was decided to use the arc42 template (Figure 7) to draw up 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 has been optimized for users and engineers. This template is frequently used by the ALTEN software architects. As a result, it is likely that the template meets the demand from the business community. In addition, it ensures that architecture information or important design choices are well described and that the entire architecture is easy to maintain [10].

Figure 7 arc42 logo

4.3 Modularity

Software layers are used in an embedded system. The application of these layers ensures modular software construction. Thanks to the layers, the software can also be used independently of the system on which it is installed. This therefore ensures flexibility. Figure 8 indicates the layers that make up an embedded system.

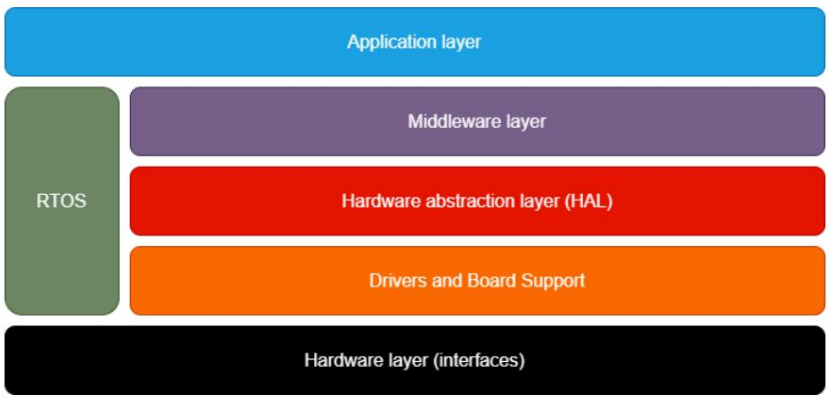


Figure 8 Embedded software layers

The software in each layer is described below. This is from high level to low level.

- *Application layer*

yer. This layer defines it system. This layer is not interchangeable because it is system specific. The Application layer depends on, is managed and run by, the software from the Middleware layer.

- *Middleware layer*

yer 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 know the functionalities of the hardware.
- *Drivers*

  manage between higher software layers and the Hardware layer.
- *Hardware layer*

  t for the layer where the hardware is defined. These are often the physical pins of a microcontroller that go to the appropriate hardware.

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. The Raspberry Pi runs the algorithm to determine the system's next move and 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-in-1-row robot. It can control the robot's hardware. But if expansions are made (such as an 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 already been selected by ALTEN. It concerns the STM32H7 microcontroller, specifically the STM32H755ZIT6U [12] from STMicroelectronics (ST). This is located on the development board Nucleo H755ZI-Q as shown in Figure 9. A development board is ideal for easy and efficient testing of the software being implemented.

Figure 9 Nucleo-H755ZI-Q

ST is a company that specializes in microcontrollers (MCUs), power electronics and flash memory. This project revolves around the microcontroller. ST has a wide variety of microcontrollers available.
From lower cost, robust 8-bit MCUt to 32-bit Arm-based Cortex- with a wide choice of peripherals. The STM32H755ZIT6U dual core microcontroller (in the rest of the document the STM32H7 dual core) belongs to the 32-bit Arm-based Cortex-. The STM32H7 contains a Cortex-M7 and a Cortex-M4 core. The Cortex-M7 can run up to a maximum of 480 MHz and the Cortex-M4 up to a maximum of 240 MHz. Within ST this falls under the category of high-performance microcontrollers. The full list of specifications can be found on ST's site [13].

ALTEN has opted for a dual-core microcontroller that meets the future vision for the 4-in-1-row robot. The STM32H7 dual-core can actually implement all planned upgrades with its two powerful cores (Cortex-M7 and Cortex-M4). As a result, it is an interesting option and a logical step for ALTEN to implement the system in order to guarantee quality and performance. It is also the first time for ALTEN to use a dual-of or concept

Many projects within ALTEN are focused on performance and factors such as cost savings and battery life are less important.

The move from a single core to a STM32H7 dual-core microcontroller is a hardware upgrade that can be implemented simultaneously with a redesign of the software architecture for the 4-in-1 robot.

4.5 Core division It

follows from the requirements that the system will be implemented on a STM32H7 dual-core microcontroller and that real-time processing and game handling will be separated from each other. In the old system, real-time processing runs on a Cortex-M4 and game handling and next move determination is done on the Raspberry Pi. In the new situation, the Cortex-M4 continues to perform real-time processing, since the principle of the 4-on-1 game does not change and adjustments to the robot's functionality are not foreseen. Within the project, the game handling is 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 also move the next move algorithm to the Cortex-M7 in the future. For now, the algorithm will remain on the Raspberry Pi because it is outside the scope of the project. Furthermore, the Cortex-M7 is also suitable for facilitating new hardware expansions.

4.6 STM32CubeIDE

The choice of an STM32H7 dual-core has consequences for the tool required to send written code to the microcontroller. An Integrated Development Environment (IDE) is often used for this. There are several options for an IDE for ST's microcontrollers.

- STM32CubeIDE
- IAR
- Keil

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

IAR and Keil are similar to each other. Both have a free version but the full package costs money. The free version is outdated and has limitations. For example, not every microcontroller can be programmed and there is a maximum size for 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 IAR and Keil as IDEs cannot be programmed on every microcontroller. In addition, the size of the data that makes up a system is limited at IAR and Keil.

One of the biggest advantages of the STM32CubeIDE environment is that it can use ST's HAL. Figure 10 indicates which software layers are automatically generated/used by the STCubeMX software tool.

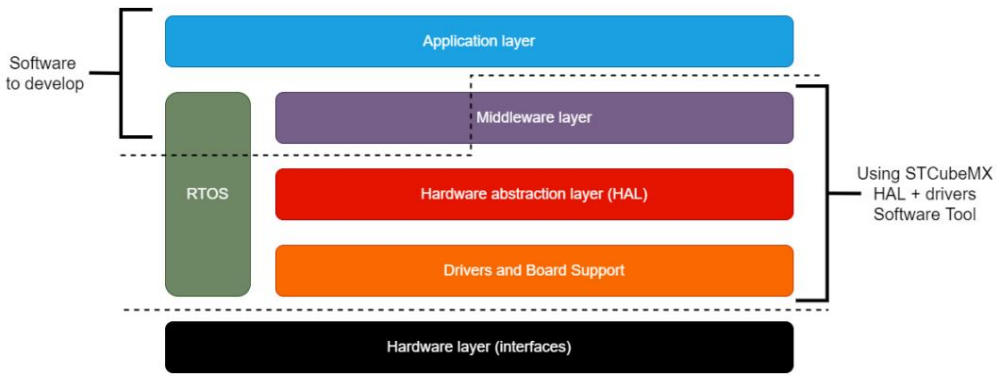


Figure 10 STCubeMX software tool generated layers

All signals and protocols (pin specific configuration) predefined 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 done 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 for ST HAL took 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 specific knowledge of the hardware details. The HAL is supplied by ST. Only the pin-specific configuration is generated. With the ST HAL library, tasks such as UART and i2c are processed by a HAL, so there is no need 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. Studying the robot has made it clear which hardware components the new operating system must take into account. The template arc42 has been chosen for the new software architecture of the operating system. The template is standard for ALTEN projects and it offers a clear, simple and effective way to structure software. Software layers are used to guarantee the modularity of the architecture. Thanks to the layers, the software can be used independently of the system it is on.

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 robot's performance robust in the future, it has been decided to separate game handling and real-time processing. This helps to keep the system clear and ensures that the two cores are used optimally. It is also the first step to deploying the system on a single microcontroller.
Because the Cortex-M7 is very powerful, the algorithm for the next move can be transferred from the Raspberry Pi to the Cortex-M7 without additional measures.

The programming environment chosen is STM32CubeIDE from the same supplier as the dual core microcontroller. The STCubeMX software tool included 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

This chapter discusses the design of the software architecture. The software architecture is the main task of this project. We worked with the template arc42. The template describes how the software is structured, communicates with each other, where data goes and how it behaves when the system is operational (Software Architecture Document, SAD). The following sections highlight the key chapters of the SAD that are critical to implementing the system, with diagrams and accompanying text helping the reader follow the steps. In addition, the design choices are explained. The complete SAD is included in Appendix III. Software Architectural Document.

5.1 System context SAD

The chapter System context within the SAD describes the scope of the project 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 thoroughly understand and describe all interfaces before you can start working on a software architecture document.

First of all, a representation was made of all external hardware interfaces and persons outside the system (see Figure 11). The 4-in-1-row robot does not have any external hardware. The people who use the 4-in-1 row

it's turn is a chip in a column of your choice and gets to see what it maintains the system when the game is over. It provides an update or upgrade if necessary and debugs the system if an error occurs. The N which turns on/off or resets the 4-in-1 robot.

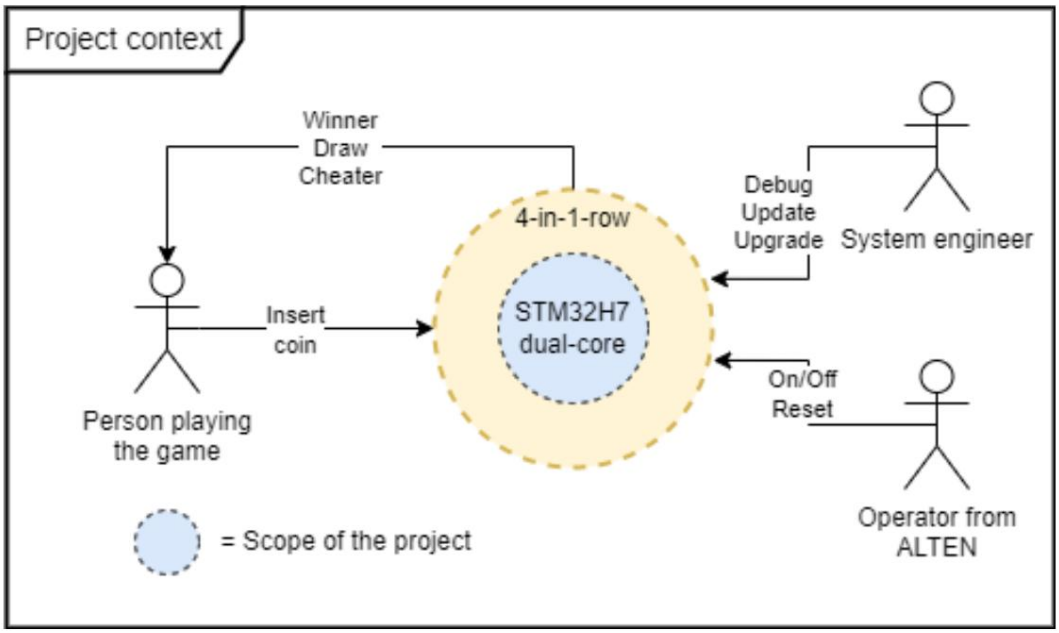


Figure 11 Project context

After the project context has been set up, we looked at what internal hardware interfaces within the system are connected to the project scope (blue circle). The investigation (Preliminary Investigation) prior to the beginning of the software architecture allows a diagram to be drawn which 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 global overview. It is a tactical view to be able to consult with all stakeholders without going too much into technical/operational details.

In addition to the System context, a Technical context has also been set up. The technical context indicates how the internal hardware interfaces are connected to STM32H7 dual-core and what signals/ protocols are used for the connections. A technical context is important for the team that gets to work operationally with the software modules for the hardware. The Technical context is included in Annex III. Software Architectural Document.

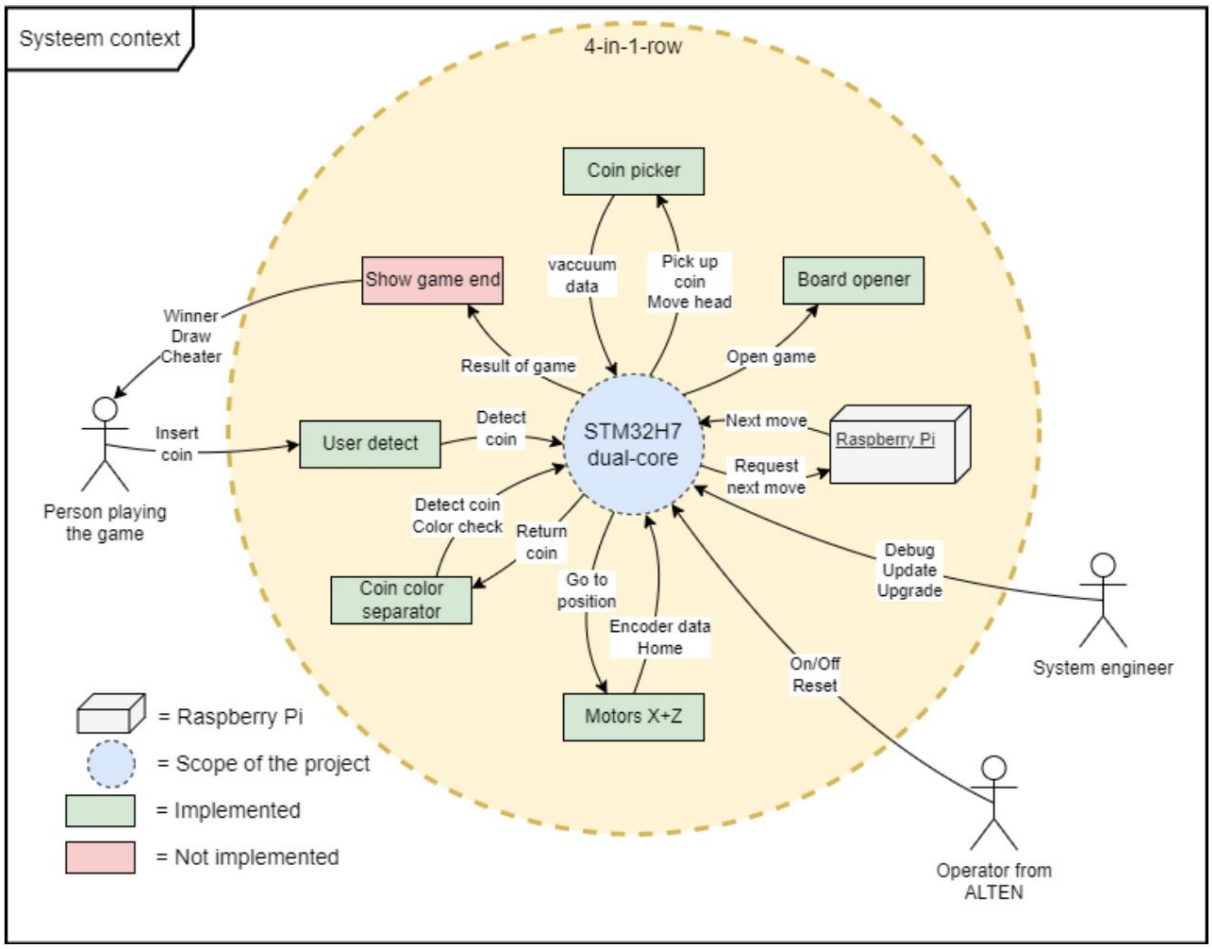


Figure 12 System context

5.2 Building block view SAD

Block V(BBV) of the SAD elaborates on the scope of the project presented in the chapter. The idea of a BBV is to step by step zoom in on the system. This is a good way to analyze a system in a structured way. This makes it clear which software modules make up the system, how they are connected and where they are located. 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). The STM32H7 dual-core is a black box here. In level 1, this STM32H7 dual-core becomes a white box that consists of several black boxes. In level 2, the black boxes from level 1 become white boxes.

During each step, inputs and outputs, connections and assignments of the black box are described. This is the right level of aggregation to discuss with stakeholders without going into too much detail at the implementation level. After BBV has been completed and discussed, the modular software modules can then be set up.

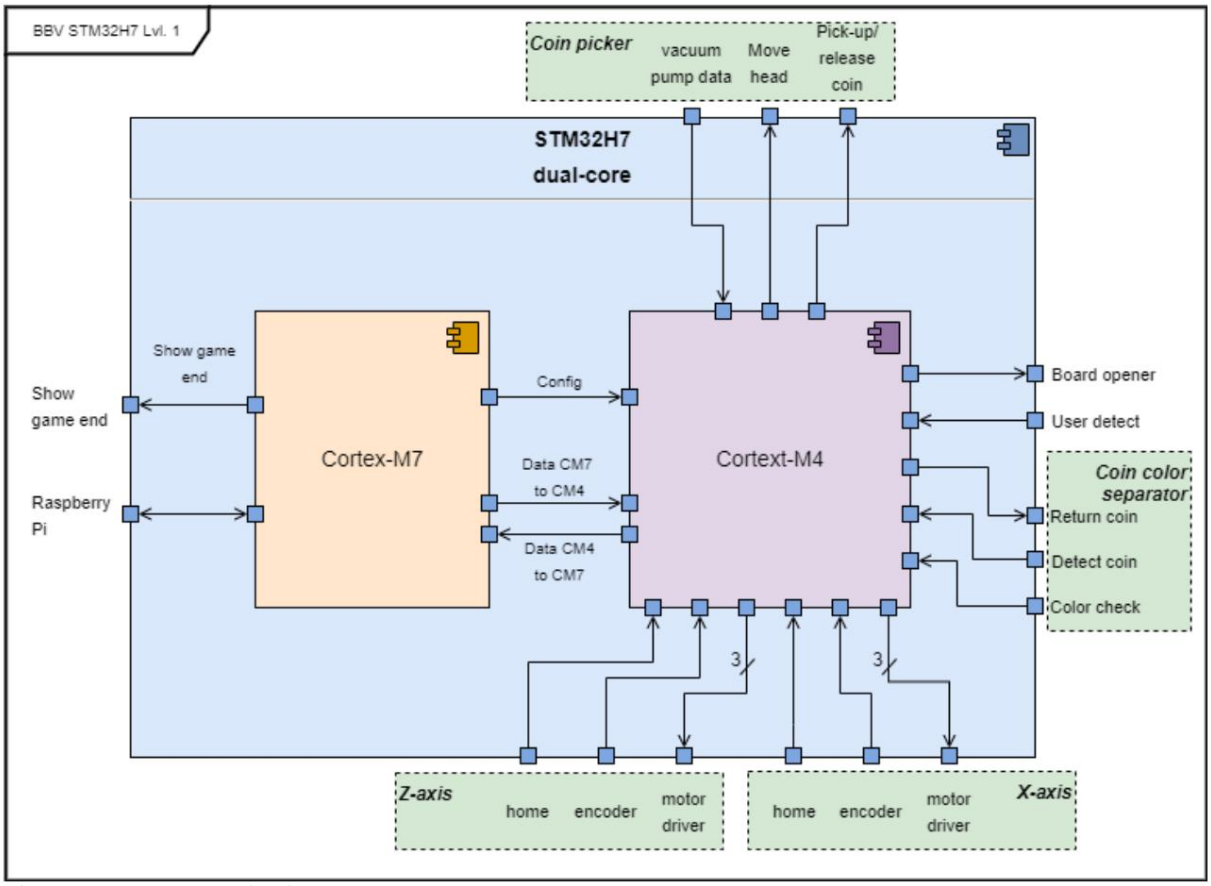


Figure 13 BBV STM32H7 level 1

As shown in Figure 13, level 1 of the BBV is considered the STM32H7 dual-core and the Cortex-M7 and Cortex-. Which one is indicated in the figure hardware interfaces to the Cortext-M4 used 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 represents the dual core communication. The communication is essential for exchanging data between the two cores that execute code in parallel.

In Figure 14 and Figure 15, the Cortex-M7 and Cortex-M4 have been parsed in even more detail (level 2) and -M7, level 2 is considered the lowest level because the blocks within this level are sufficiently representative for the final software modules within the Cortex-M7.

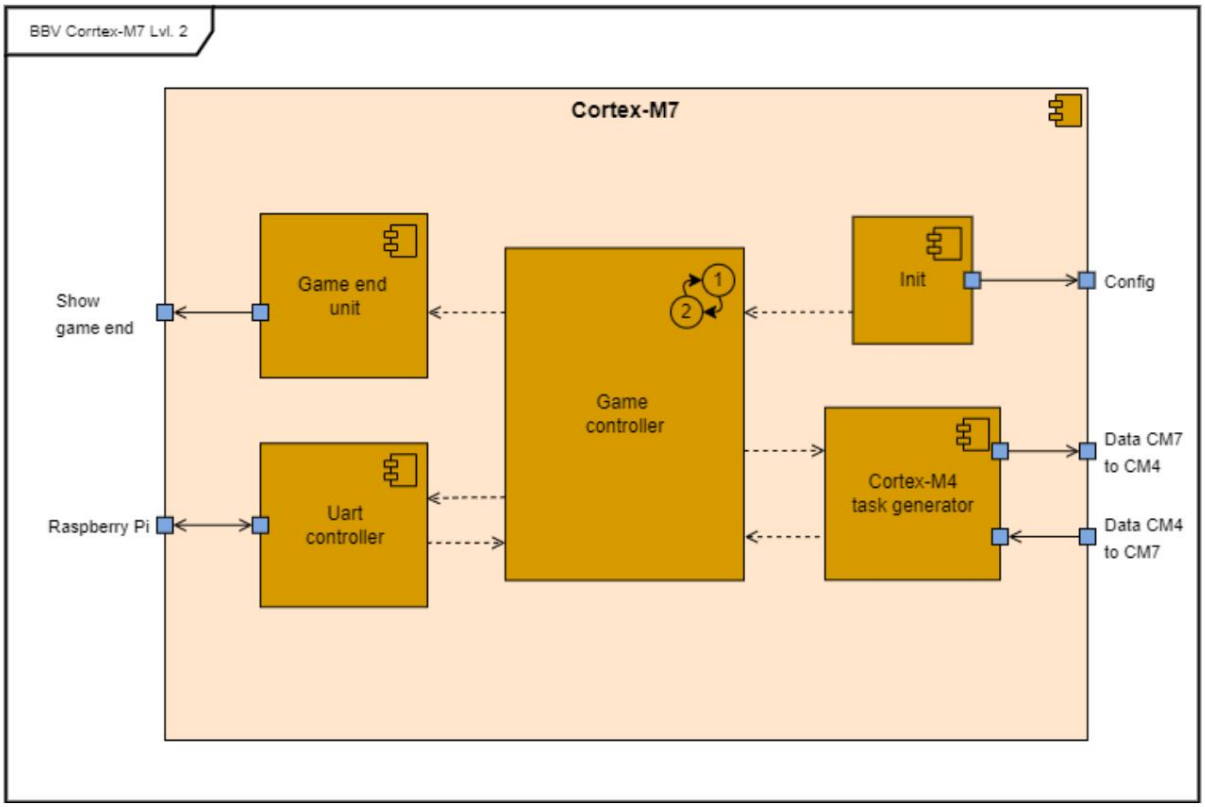


Figure 14 BBV Cortex-M7 level 2

Table 3 Description BBV Cortex-M7 level 2

Black box Description	
Game controller	The Game controller is responsible for the game flow by means of a state machine.
Init	Turned once to initialize and power up 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 communications to an output for the player.

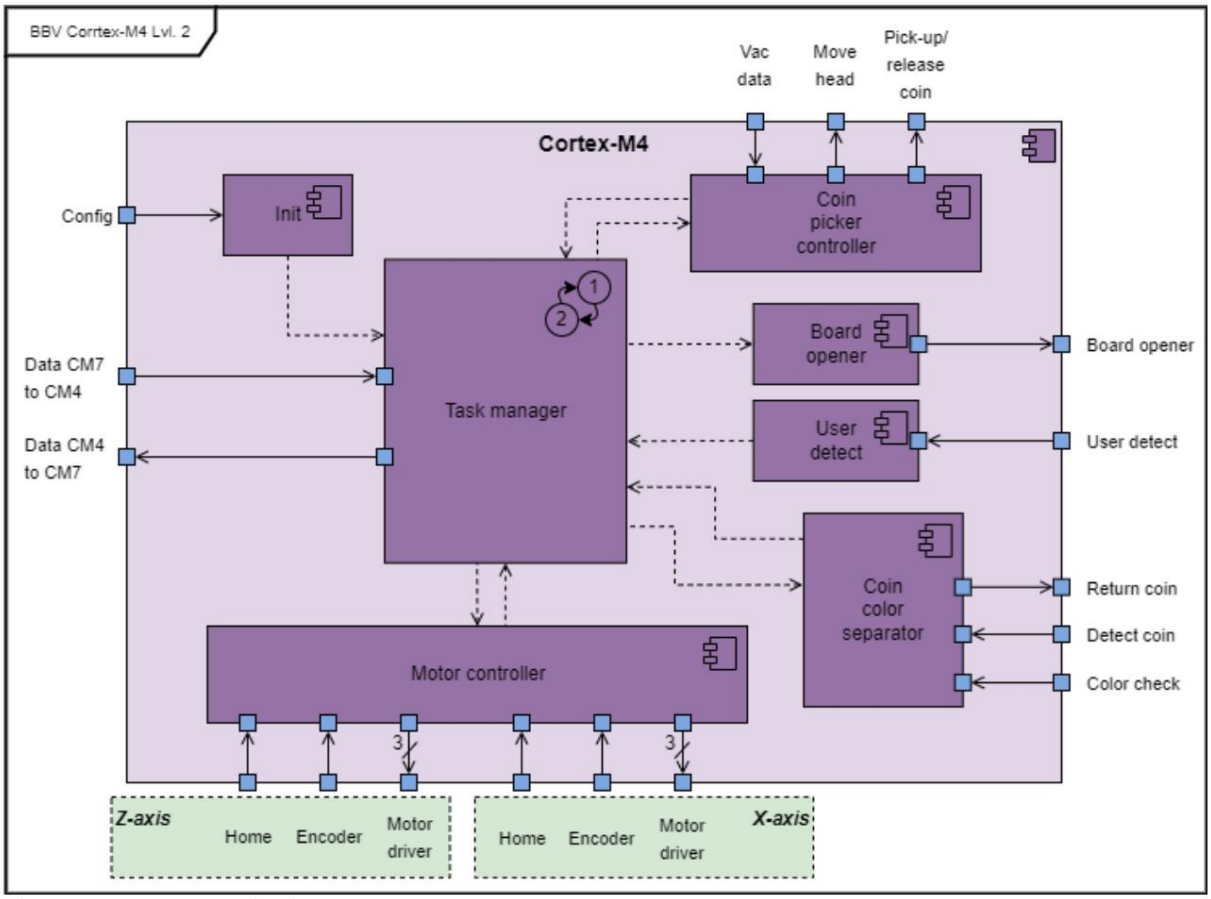


Figure 15 BBV Cortex-M4 level 2

Table 4 Description BBV Cortex-M4 level 2

Black box Description	
Task manager	The Task manager is responsible for the execution of the commands to be executed by the Cortex-M7 by means of a state machine.
Init	Turned once to initialize and power up the Cortex-M4. In this phase, the sensors are tested for presence and the
engine 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 the functions for handling the chip separation.
User detect	The User detect implements all functions for handling the sensors for the player's input.
Board opener	The Board opener implements all functions to handle the opening of the 4-in-1 board when the game is over and the clean-up phase begins.
Coin picker controller	The Coin picker controller implements all the functions for handling the pick up and release of a chip.

For the Cortex-M4 we zoom in even further (level 3) on the following blocks: Coin color separator, Motor controller and Coin picker controller. This is necessary because these blocks contain more important sub software modules for controlling the robot. In Figure 16, the blocks are referenced the full SAD in Annex III. Software Architectural Document.

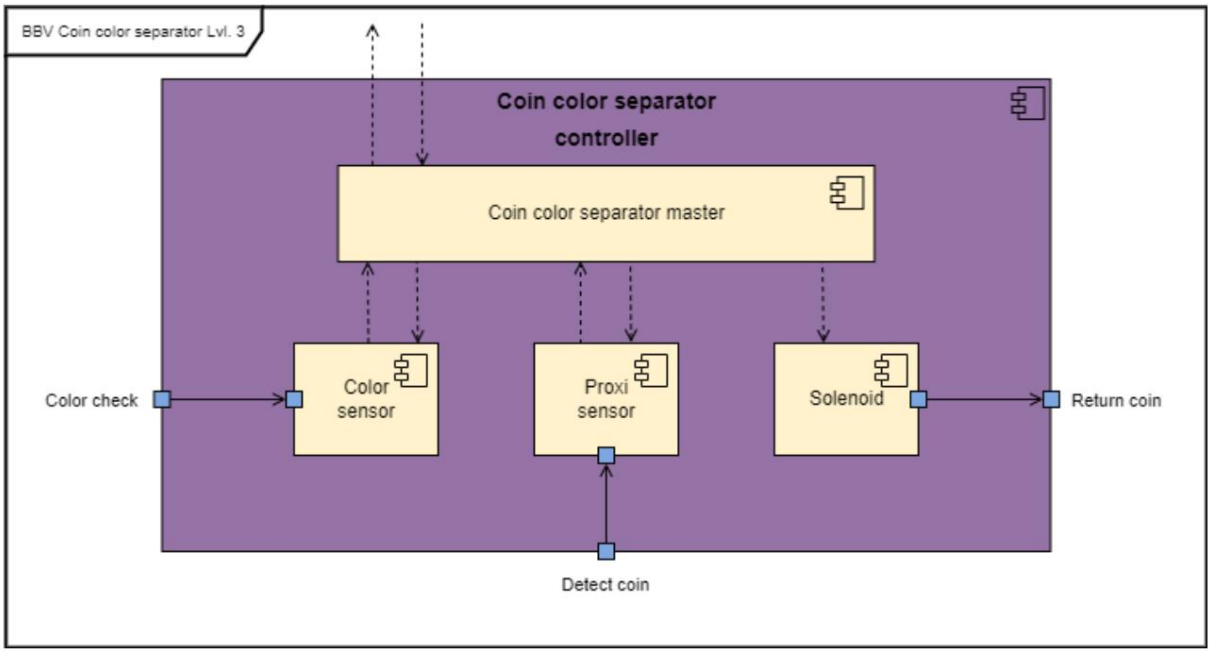


Figure 16 BBV Coin color separator level 3

Table 5 Description BBV Coin color separator level 3

Black box Description	The Coin color separator
Coin color separator master	master is responsible for receiving the sensor data and controlling the solenoid.
color sensor	Requests the data about the color of a chip.
proxy sensor	Gets data if a chip is present.
solenoid	Is responsible for activating the flipper.

5.3 Run-time view SAD

visualizes how the software modules, described in the Building block view chapter, communicate with each other when the system is operational. The diagrams indicate which software modules communicate with each other over 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 also describing all process steps in a model, it becomes clear which data software modules receive and must send out.

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 that 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 sequence of communications exchanged between the software modules to perform the functionality of the state.

The game state machine is shown in Figure 17. It runs on the Cortex-M7 (Figure 14). The first state is the

entire system booted and initialized. As soon as this state is ready it goes

state the system is waiting for the game to start. When the game has started, it is the player's turn to place a chip in the 4-in-1 game and find it

After this, the turn goes to the robot and the state changes to state, the robot performs a calculated move. After each turn, the robot checks whether there is a winner, whether there has been cheating, whether there is a tie. If this is not the case, the turn goes back to the player with the move and the system is in the corresponding state (human move or robot move). Once the game is over, the clean-up state starts and all chips are cleared. The red chips go to the robot and the yellow chips to the player's bin.

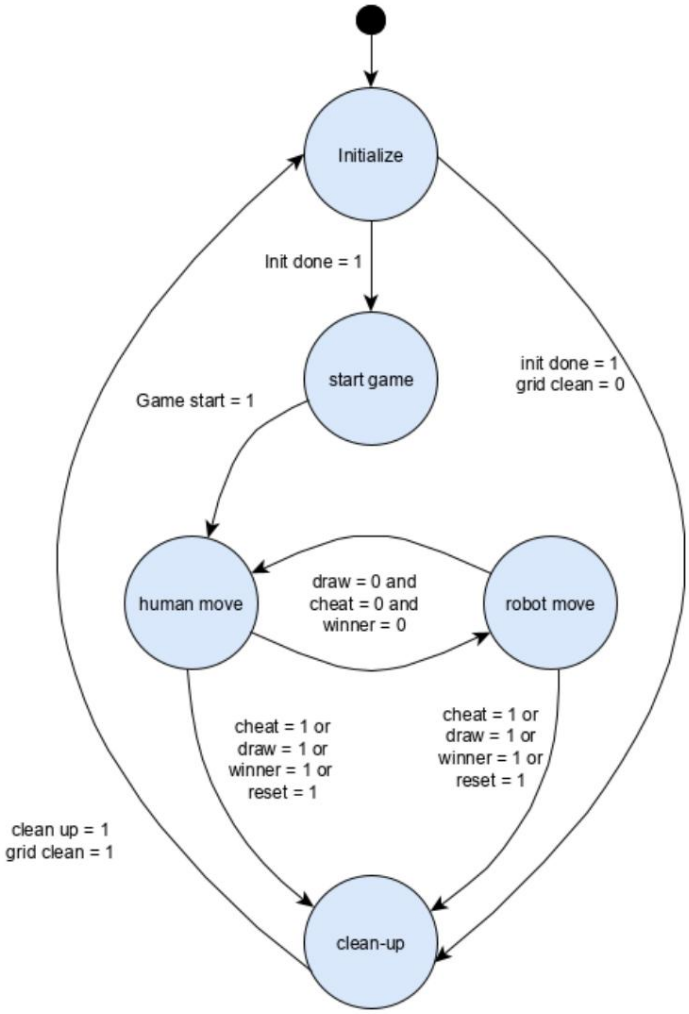


Figure 17 State machine gameplay

The state machine for the real-time processing of the hardware is shown in Figure 18. It runs on the Cortex-M4 in Figure 15). The In this

state, the hardware components are configured and set up. After that has

state waits for the Task manager to receive a task from the Cortex-M7. As soon as a task arrives, the next state is determined based on the task. This

-Any of these states

directs the hardware to complete the task. When the task is completed, the Task manager returns to the Idle state.

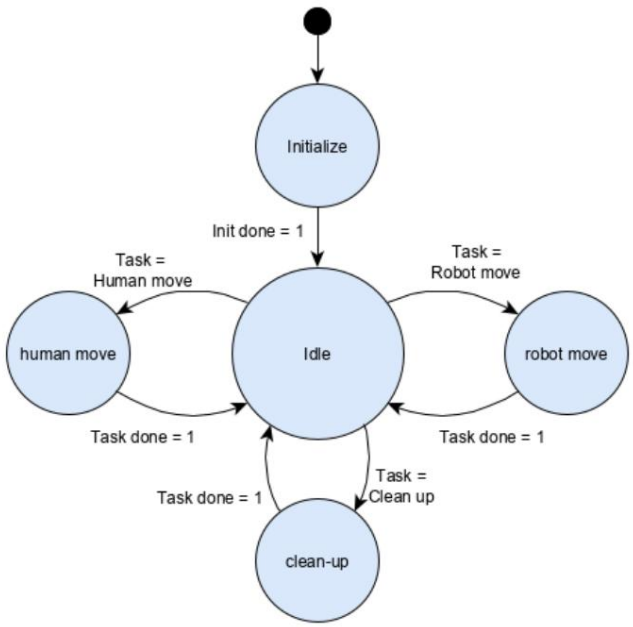


Figure 18 State machine real-time processing

Each state of the state machine in Figure 17 can be represented in a sequence diagram. This is to indicate what happens in each state and how the communication between software modules works. The robot move is worked out in Figure 19. The other sequence diagrams can be found in the SAD included in Appendix III. Software Architectural Document.

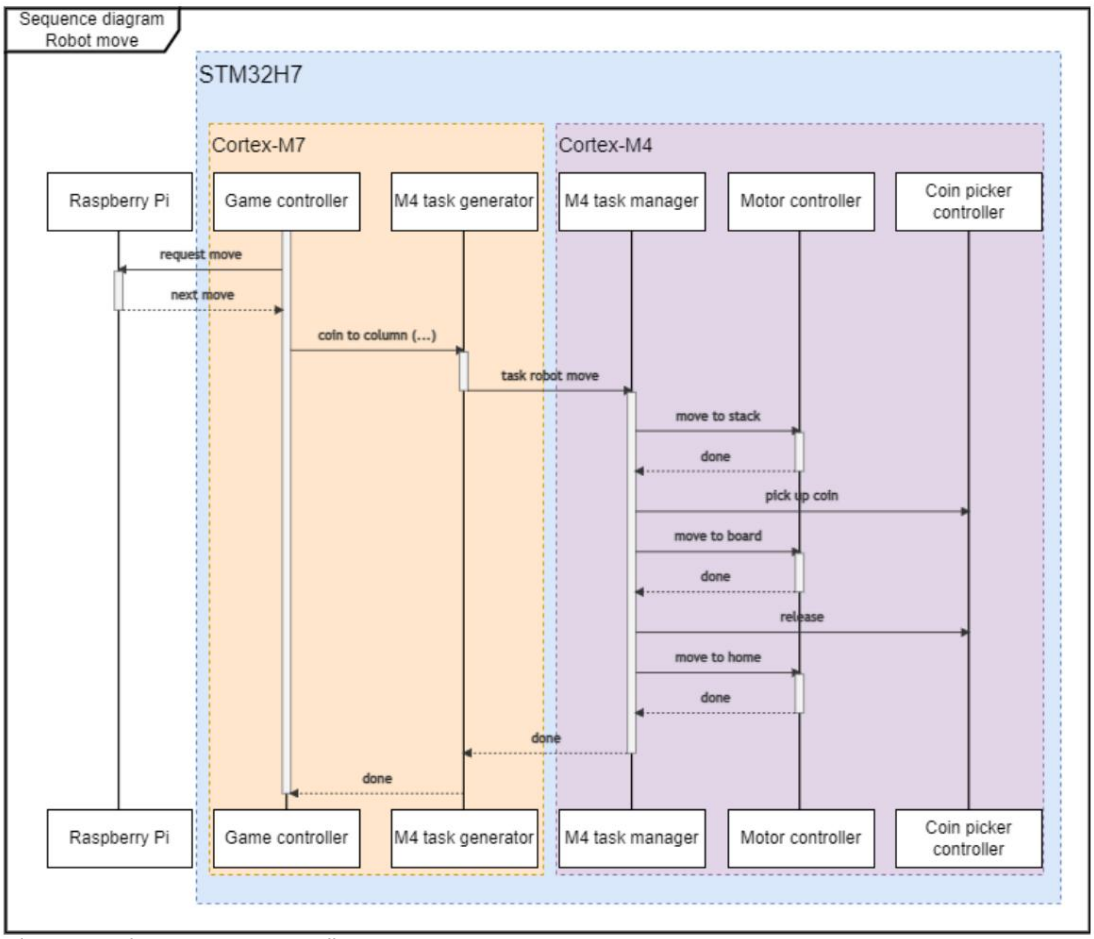


Figure 19 Robot move sequence diagram

The robot move starts with the Game controller requesting the next move from the Raspberry Pi. The Raspberry Pi returns the next move, after which the Game controller tells the M4 task generator which command to make. Once the job is created, the M4 task generator sends the job 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. Once the robot has arrived at the storage, a token must be picked up. Now the robot can move to the board and then release the token. Then Next, the M4 task manager notifies 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

describes what is needed to actually implement the software.
A microcontroller is used for this. The main 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 which signals/ protocols are used (the green pins in Figure 20) to control the robot. A pin cannot carry every type of signal/ protocol. The pins are set so that all signals/ protocols required to run the system are available. There are also pins allocated for Ethernet. This is a feature that is currently not implemented. This is the intention in the future and it is then important that the pins required to use Ethernet are not occupied for other tasks. Full descriptions and diagrams are included in the SAD (Appendix 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. Chapter 5.2 Building block view describes which software modules are necessary to implement a working 4-in-1-row robot. The chapter Modular software modules explains how the modules are interrelated from high level to low level (Application layer to Hardware layer, Figure 8). By using the embedded software layers you create reusable modules.
The HAL and driver layers are representative of this, but the modules from the Middleware can also be reused. The coherence is visualized by means of diagrams. By drawing up these diagrams, it becomes clear what the scope of the various 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, e.g. 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. After all, the modules do not need to be aware of each other's functionalities or tasks in order to do their job.
Because tasks are separated into specific modules in the software, the system meets the required overview and insight and the simplicity of implementation. With a modular structure, several people can work on the system at the same time. The modules have a fixed interface so that each module can be adjusted independently. Thanks to modular software modules, for example, a sensor can be replaced during an upgrade, without having to adjust all the code. Only the module with the sensor-specific details needs to be modified.

Modularity Coin color separator Figure 21 shows the diagram for the software modules of the Coin color separator. The modules from chapter 5.2 Building block view are combined with the standard HAL. The block

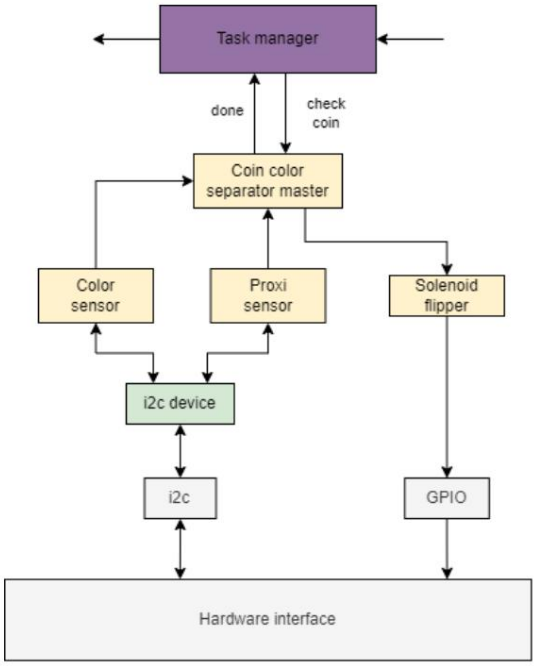


Figure 21 Software layers Coin color separato

software modules where the configuration specific parts are generated. These blocks initialize the General Purpose I/O pins (GPIO) and the i2c bus. The other blocks are software modules that need to be developed. a generic module that is made to ensure that each sensor connected via the i2c bus only needs to indicate whether data needs to be sent or requested. For example, the modules for the sensor do not have to be aware of all the standard functions of an i2c connection, because this is

. By this modular approach, the modules can be reused and it is easy to add an extra module, for example for an extra i2c sensor.

Modularity Cortex-M7 The Cortex-M7 is used for the game definition of the 4-in-1-row robot. A Round robin with interrupt method has been chosen to implement the course of the game in a state machine (Figure 17) (Chapter 6.1). within the Cortex-M7 the module that executes the state machine. The Game controller is responsible for keeping track of the state the system is in and what should be done. Other modules have been chosen for the other tasks of the Cortex-M7. The modules

-le input/output die

the Game controller used to be separated from each other. These modules handle all the 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 All software that controls and handles the hardware runs on the Coretex-M4. To create the required modularity within this core, it was decided to develop a separate software module for each unique hardware block (Figure 15). To properly manage all the software modules of the hardware, gen. By using the functionalities of other components, you can build modularly. The

After all, functionalities do not need to know about each other's existence/status. That information is maintained by the Task Manager. The Task manager will also work through a state machine.

5.6 Conclusion

Drawing up a structured software architecture should lead to a future-proof 4-in-1-row robot. It must be possible to easily implement changes and adjustments to requirements without frustrating the operation of the system. With the preparation of a SAD, the assignment to develop a structured software architecture has been fulfilled.

The arc42 template 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-in-1-row robot can understand how the system works and which software modules are implemented. In addition, modularity ensures 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. Then the BSP modules were tested separately for i2c, UART and Pulse Width Modulation (PWM) communication. Developed in parallel, in which step by step, based on these tests, we worked towards the complete control system of the 4-in-1-row robot.

6.1 Implementation method

Chapter 4.5 Core distribution explains the function of each core. Because we work with a dual-core implementation, some form of dual-core communication will always have to be implemented. For the technical elaboration of this, see section 6.2 Dual-core communication.
In addition, a method must be chosen with which the software will be executed (software loop).
Four possible options have been investigated (Appendix 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.

A Round robin with interrupt implementation method has been chosen within the project. The implementation of the current 4-in-1 robot is also done with a Round robin with interrupt. This 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 motors, chip insertion and dual-core communication, interrupts are needed. The deadline can be caught by means of a timer ISR. What must be taken into account is the false data that can be created by using interrupts. Appendix VI. False data explains in detail how this can be prevented.

6.2 Dual Core Communication

Dual-core communication is a critical 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 necessary. A solution must therefore be devised for the communication and synchronization between the two cores.

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 prevents both cores from writing into memory at the same time.

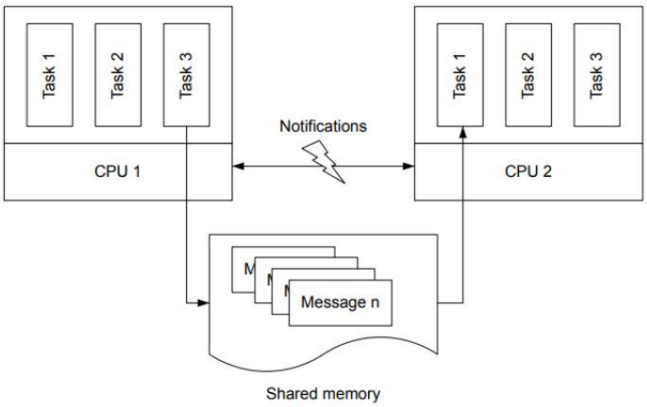


Figure 22 Dual-core communication [17]

6.2.1 Shared memory

The first step in implementing communication is choosing a piece of shared memory that is available and accessible to both cores. The STM32H7 has dual-core memory architecture - This architecture ensures that a large part of the available memory is accessible to both cores. Table 6 indicates that approximately 82% of the memory is readily available for both cores. The Cortex-M4 requires a Master Direct Memory Access (MDMA) controller to access the memory ITCM and DTCM.

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 has been chosen as shared memory. SRAM4 is located in the D3 domain that remains available for both cores, even if one is turned off. The D1 and D2 domains are only available if the corresponding core is active. In addition, SRAM4, with a size of 64Kbytes, has sufficient 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 choosing 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-in-1 implementation. By opting for a notification from one core to the other, consistency is guaranteed and it reduces the time the system would otherwise spend constantly checking whether new data is available. It also provides synchronization.

The STM32H7 dual-core has two solutions. Both solutions use a hardware
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. Figure 23 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 when an interrupt is detected.

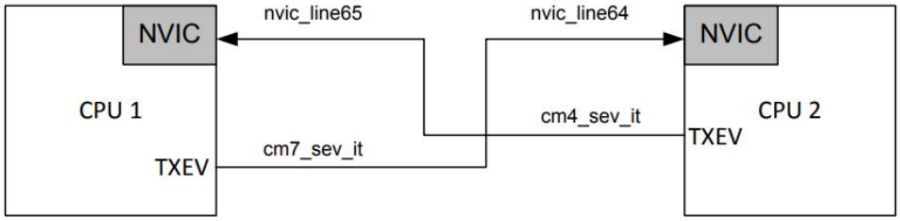


Figure 23 EXTI and SEV dual-core communication [17]

2. *Hardware Semaphore (HSEM) free interrupt.*

The HSEM [18] uses a HSEM controller (Figure 24). An HSEM works with a lock. 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, an interrupt is sent to the other core. In the core receiving the interrupt, an ISR is performed by the NVIC. From then on, the HSEM.

This is a cores not running a task at the same time and it also serves 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 varying commands.

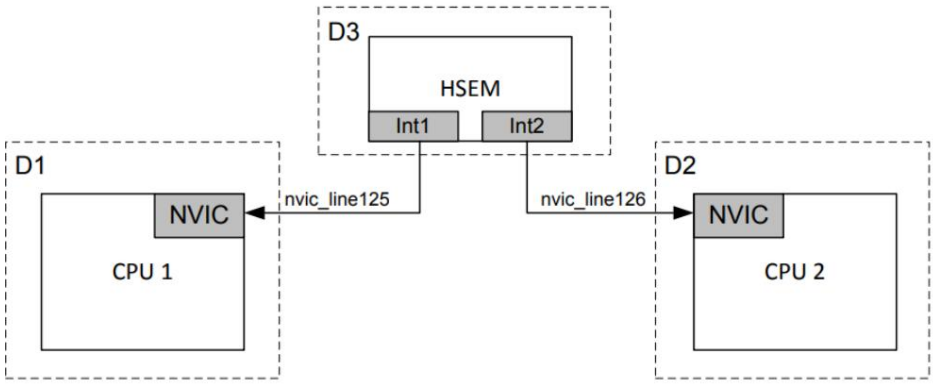


Figure 24 HSEM dual-core communication [17]

Within the project, the mechanism of an HSEM has been 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). This way the Cortex-M4 knows what task to perform when the interrupt arrives. Figure 25 shows the implementation of the HSEM within this project.

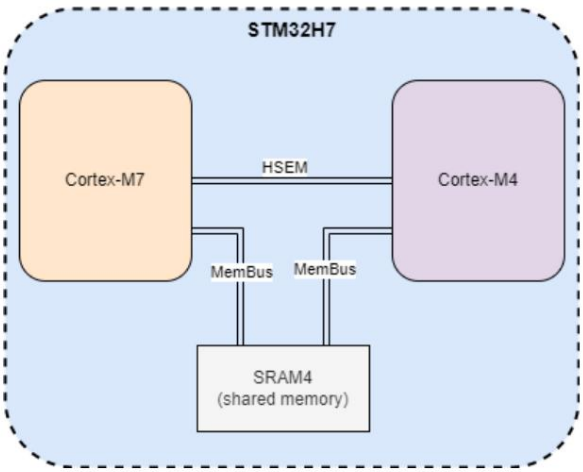


Figure 25 dual-core STM32H7 dual-core implementation

6.2.3 Implementation of dual-core communication

The first part of the BSP concerns implementing the dual-core communication application. Figure 26 gives a schematic overview of the components and how they are connected.

The application is implemented in two ways.

1. Information sharing between the Cortex-M7 and Cortex-M4 is via shared memory (SRAM4);
2. The two cores can send tasks to each other.

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

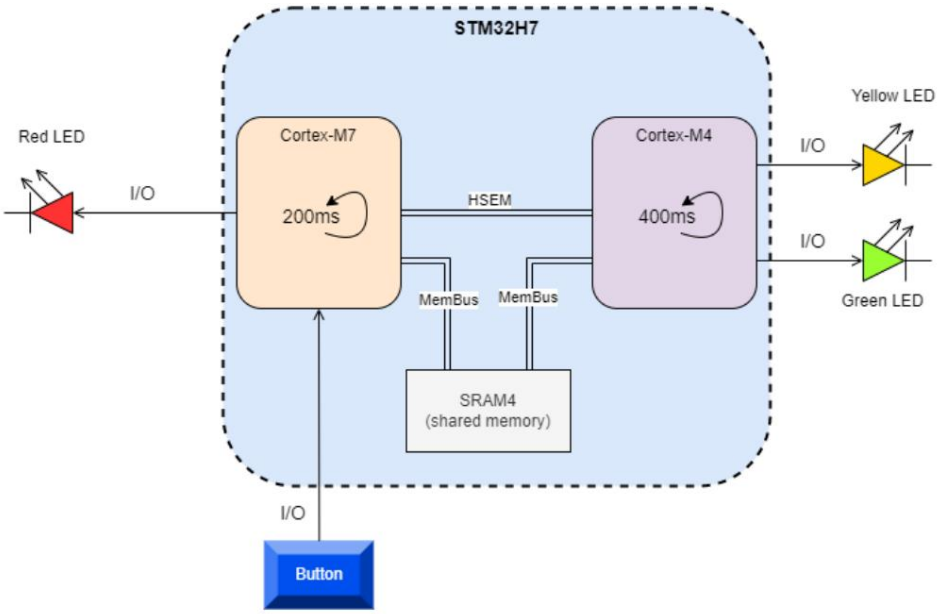


Figure 26 Schematic overview demo dual-core communication

Ad 1) Shared memory unit test
When testing shared memory, the Cortex-M4 will check which value is in SRAM4. Table 7 shows which value is 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 the maximum value is reached (3). After that, the data starts again at 0. In addition, an HSEM is also sent to the Cortex-M4 every time the button is pressed. 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	On
1	Flashes at a frequency of 1.25Hz Off	
	Out	Flashes at a frequency of 1.25Hz
	Blinks at a frequency of 1.25Hz Blinks at a frequency of 1.25Hz	
2 3 unknown	Out	Out

Ad 2) HSEM unit test
During testing, the Cortex-M4 will receive different HSEM. Table 8 shows which HSEM corresponds to a certain pattern of the yellow and green LED. The button connected to the Cortex-M7 ensures that all

Table 8 Possible Patterns HSEM

HSEM	Yellow LED	Green LED
HSEM-1	At	At
HSEM-2	Flashes at a frequency of 1.25Hz Off	
HSEM-3	Out	Flashes at a frequency of 1.25Hz
HSEM-4	Blinks at a frequency of 1.25Hz. Blinks at a	frequency of 1.25Hz **unknown**
	Out	Out

During the implementation of the dual-core communication, a problem of data coherence occurred due to cache. As a solution to this problem, a Memory Protection Unit (MPU) was chosen within the project. An MPU can make part of the memory non-cacheable, which solves the problem. For an explanation see appendix VII. Complication dual-core communication.

With the successful implementation of both tests, the HSEM and shared memory have been shown to meet the requirements of dual-core communication. This completes part of the BSP.

6.3 BSP modules for hardware

In addition to the dual-core communication, unit tests have been performed for i2c, UART and PWM. In the following paragraphs . First, i2c & UART were added to the unit of dual-core communication, with PWM. Step-by-step progress is being made towards the complete operating system of the 4-in-1-row robot.

6.3.1 Dual core i2c UART demo

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

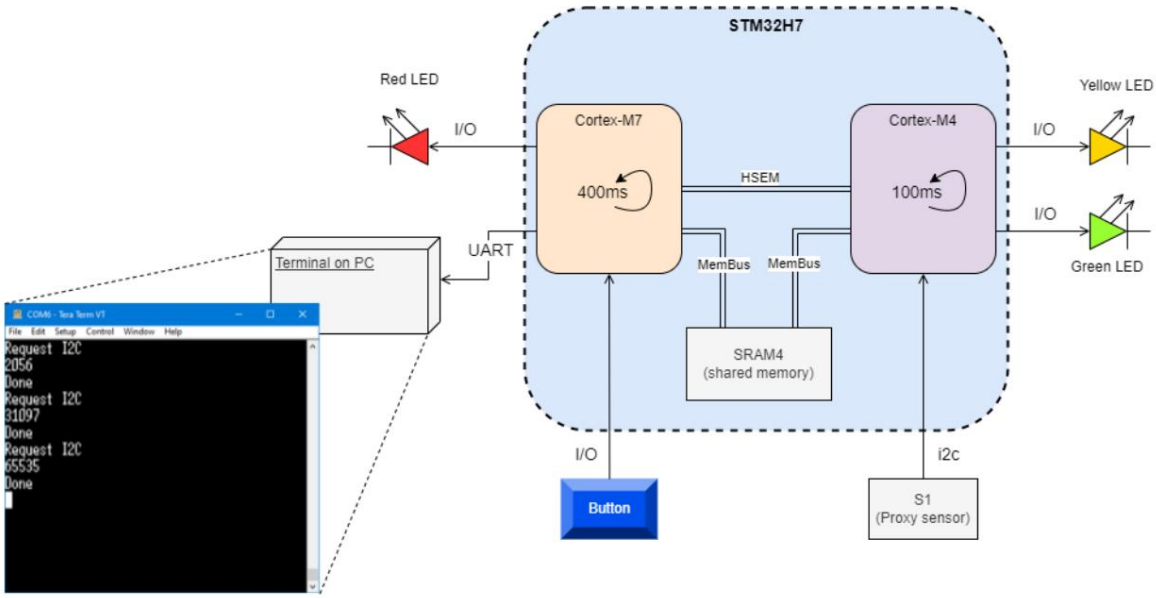


Figure 27 Schematic overview demo dual-core i2c UART

Before the demo was set up, the separate unit tests of i2c and UART were performed. The BSP module i2c dev are generic (see Figure 28). The i2c dev module ensures that any sensor only has to indicate whether data needs to be sent or requested. The module UART controller 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 used in a demo (and in the final operating system).

After it has been shown that the modules i2c and UART function separately, the aim in the demo is to combine them. The Cortex-M7 has a 400ms delay per loop and makes the red LED flash at a frequency of 1.25Hz. The Cortex-M4 has a 100ms delay per loop and makes the green LED blink at a frequency of 2.5Hz. Furthermore, at each interval of the Cortex-M4, the value of the i2c sensor is checked via the BSP module. The sensor is a proximity sensor (S1). If the value of the S1 exceeds a set limit, the yellow LED lights up. When the value of S1 falls below the limit again, the yellow LED goes out.

When the button on the Cortex-M7 is pressed, a request will be sent, via an HSEM, to the Cortex-M4 to request the data from S1. This HSEM is received in the Cortex-M4 and the data is read from S1.
Then the data is placed in SRAM4 and an HSEM is sent to the Cortex-M7 that data is available. When the Cortex-M7 receives the HSEM, the data is read from SRAM4 and displayed on a computer's terminal. The display on the computer is done via UART BSP module. Figure 28 shows the software modules used. The demo has shown that the communication between the different modules is proceeding as expected.

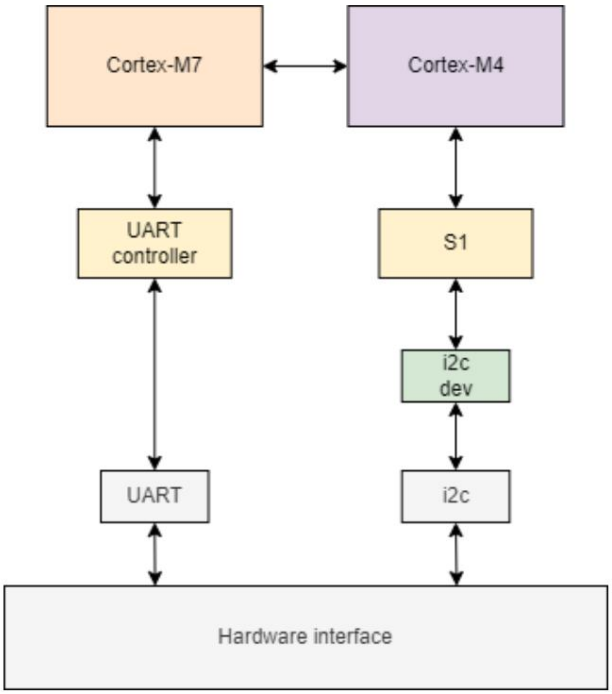


Figure 28 Software layers dual-core i2c UART

6.3.2 Dual core i2c UART PWM demo

The last demo is an extension of the previous demo, with more modules of the 4-in-1-row robot. This test should represent the 4-in-1-row robot as well as possible so that a final version of the control system can be developed based on this demo. Figure 29 gives a schematic overview of the demo.

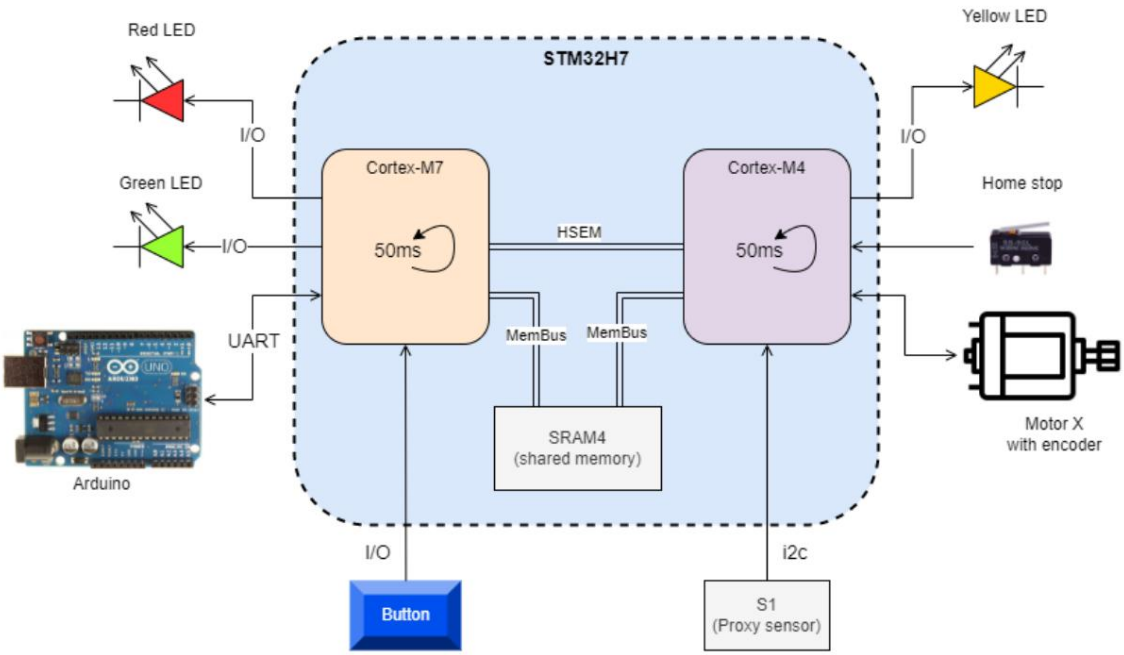


Figure 29 Schematic overview demo dual-core i2c UART PWM

Prior to the demo, the unit test was carried out for the BSP module PWM, which controls servo motors. It is removable and the operation can be easily demonstrated. Then the PWM is applied to the motor which is part of the physical 4-to-1 motors work as expected allowing the BSP module PWM to be applied in the demo.

Now that the PWM module has been shown to function independently, it will be inserted into the demo. The Cortex-M7 has a 50ms delay per loop. The state machine (Figure 17) of the game controller is implemented on the core. The Cortex-M4 has a 50ms delay per loop and the value of the i2c proximity sensor (S1) is checked at each interval. The sensor S1 serves as a representation for the user detect of the 4-in-1-row robot. First, the Cortex-M7 is in the initialize state, the red LED lights up and all functions are initialized. In this state the Cortex-M4 will not know where to drive it

component is physically located, it is necessary to find a home position in advance. This way you can keep track of where the component is located. The motor constantly rotates in one direction until the end position is reached (home stop). This completes the homing sequence and completes the calibration phase. Now the system can proceed to the next state, the Start game state.

In the start game state, the button is used to start the game and the green LED flashes at a frequency of 2.5Hz. The system enters the human move state and it is the player's move. Ortex-M4 randomly generates a value between 1 and 7 to mimic the column of the board. Then this data is written to the shared memory (SRAM4). By releasing a HSEM, the Cortex-M7 is notified of the player's input. The Cortex-M7 will then retrieve the data from SRAM4 and go to the robot move state.

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

By performing this demo, the operating system of the 4-in-1-row robot is well approached. This is an approximation, because not all system components are included. For example, only the motor of the x-axis was used during the implementation, an Arduino was used instead of a Raspberry Pi and the clean-up phase was not taken into account. This need not be an obstacle to demonstrate 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 subsequently operating system of the need to replace current robot.

th interrupt. This is the most appropriate method for deploying software. It is easy to understand, adapt and it is transparent. The system goes through standard phases, making the Round robin with interrupt the appropriate method. Because there are priority tasks, an interrupt is necessary.

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

Par
modules, has been worked towards the complete operating system of the 4-in-1-row robot.
There are sequentially demo core, i2c and UART merged and then a demo where the setup is extended with PWM.
, have shown that the software architecture is applicable and that the design choice for modularity made is the right one. It has also been made plausible that the BSP modules can also be used in other projects.

7 Validation

The requirements are validated in this chapter to check whether the project has been carried out properly. All requirements with before the project can be accepted. In Table 9 the requirements are repeated and checked. 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 provide hardware and software developers who are not familiar with the system with a quick insight into the functioning of the robot.	Must		This has been achieved by means of 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 structured using 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 relationship between software and hardware should 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 parts of the BSP that necessary for the functioning of the robot must be tested independently of each other within the project.	Must		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 in the Z and X	Must		The last demo contains a calibration tool that calibrates the motors at startup (homing sequence).
UR.9	A Pin-out must be drawn up with a table and diagram to ensure that hardware developers can work out a PCB design for the dual core processor in the future.	Should		A pinout has been put together for implementing a custom PCB in the future.

UR.10	The algorithm used on the Raspberry Pi running, must be integrated on the new STM32H7 dual-core.	Could		Due to a lack of time and a lack of knowledge of the algorithm, it was decided early in the project, in consultation with the technical supervisor, not to consider this requirement.
UR.11	A physical demo must demonstrate that the modular construction of the software architecture is applicable and functions.	Could		A demo has been made that shows the operation of all basic functions of the software architecture. Due to a lack of time, it has not (yet) been possible to get the robot fully operational with the new system.

8 Conclusion and recommendations

ALTEN worked on the following assignment description for five months:

4-in-1-row robot, which can facilitate all changes and expansions in the future and implement this software architecture on a STM32H7 dual-

The requirements of the assignment were delivered within the available time. The requested structured and modular software architecture has been realized by drawing up a Software Architecture Document (SAD) based on the arc42 template. The SAD describes how the system works, how it is built up 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

The architecture has a logical structure. First, all software modules needed for the 4-in-1-row robot were set up. These are then subdivided into increasingly 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 has finally been proven in practice.
By building the software architecture according to the template, it is clear to everyone how the system ultimately functions.

These have been implemented step by step and expanded to approach the final system. An approach has been chosen because not all modules are significant in demonstrating that the modular software architecture works. There are possibilities to expand the robot in the future with features that make the 4-in-1-row robot more attractive, smarter and faster.

The step from a single core to a 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 shown that communication between two cores can run smoothly. It is the first time that a dual-core microcontroller is used within ALTEN. That is the next projects. core in

Based on the results, some recommendations can be made. The project can be continued by implementing all other modules for the 4-in-1-row robot. The robot now works sub-optimally because only the necessary modules have been developed, implemented and validated. In addition, the algorithm can be moved from Raspberry Pi to the STM32H7 dual-core microcontroller so that the system runs on a single microcontroller. 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-in-1-row robot with Ethernet or a screen.

Evaluation

During my education I discovered that coding is my passion. So I was somewhat disappointed that coding took second place during my graduation. First a software architecture had to be developed, only then could programming take place. In retrospect, I have learned to appreciate the order. It took me a long time to understand what exactly is underneath

supervisor) and Berend (Software architect) it has become clear to me how to design and visualize software architecture in a user-friendly way. It has led me to learn many new things about reducing complexity and translating an operating system into modules

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

I also learned a lot from the ALTEN employees and my fellow interns. I was able to ask questions and always received good feedback. In doing so, I was challenged to look outside the box. It was a challenge for me to fill a graduation report. I find it difficult to report in detail. Perhaps I should have chosen the English language 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. That's fine with me.

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

Bibliography

[1] https://docs.rs
online.com/5cf2/0900766b811a32c7.pdf.

[2]
https://www.maxongroup.us/medias/sys_master/root/8884124516382/EN-21-488-492.pdf.

[3] -Available:
https://www.maxongroup.com/medias/sys_master/root/8806895386654/13-217-en.pdf.

[4] -00005-
Standard-Servo-Product-Documentation-v2.-462659.pdf.

[5]
https://nl.mouser.com/datasheet/2/260/mwec_s_a0011714497_1-2274579.pdf.

[6]
https://www.sparkfun.com/datasheets/Robotics/Other/spec%20sheet.jpeg.

[7] [online]. Available:
https://nl.mouser.com/datasheet/2/737/Adafruit_05132020_413-1858436.pdf.

[8]
shop.adafruit.com/datasheets/TCS34725.pdf.

[9] www.alten.nl/.

[10]

[11]

[12]
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]
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] ment
tools/stm32cubeide.html#get-software.

[15] -state_machine.

[16]

[17] -core
https://www.st.com/resource/en/application_note/an5617-stm32h745755-and-stm32h747757-lines-interprocessor-communications-stmicroelectronics.pdf.

[18]
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] le: https://nl.wikipedia.org/wiki/Interrupt.

[20] [online]. Available:
https://www.eecs.umich.edu/courses/eecs461/lecture/SWArchitecture.pdf. i/

[21] Cache_(temporary_memory).

[22]
Cache-Coherency-on-Cortex-M7-Based-MCUs-DS90003195A.pdf.

[23] _protection_unit.

Attachments

I. Statement of Originality

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 approach

Plan of approach -

4 in 1 row robot

Version 5
Date: 2/28/2022

ALTEN
Pascal Fatz
2491281

Final version

Version history

Version	Date	Status	Writer	Remark
1	2/9/2022	Concept	Pascal Fatz	First draft version
2	2/10/2022	Concept	Pascal Fatz	Small adjustments to the structure
3	2/14/2022	Concept	Pascal Fatz	Process feedback Gijs
4	2/28/2022	Concept	Pascal Fatz	Process feedback Jeedella
5	2/28/2022	Concept	Pascal Fatz	Process feedback Aniel

Acronyms and abbreviations

Term	Description
BSP	Board Support Package
PvA	Plan of approach

Contents

1 Wallpapers

For my graduation internship in the fourth year of my study Electrical Engineering at the Fontys Hogeschool Eindhoven, I am developing an embedded architecture for the 4-in-1-row robot from ALTEN based on a dual-core STM32H7 processor.

ALTEN is active as a consultancy and engineering organization in various markets of the high-tech sector and IT. Knowledge in the field of technology plays a more important role in this, the central pillar within ALTEN. ALTEN also deploys its specialist knowledge in the field of IT, an important sector in the Netherlands. Quality and reliability, innovations in the field of Big Data and the Internet of Things, are subjects in which ALTEN plays an active role with leading partners in order to strengthen the digital economy of the Netherlands.

ALTEN has three departments: ALTEN IT, Technical Software and Mechatronics. The 4-in-1-row robot falls within the Mechatronics department. WTB, Mechatronics and electrical engineering consultant engineers work in the Mechatronics department. My technical business mentor is Aniel Shri, he himself works as a consultant at ASML and can therefore guide me very well within the assignment and ALTEN. I also have a business manager Gijs Haans, who supports me in the field of personal development and progress.

To give consultants the opportunity to develop their competences of different Mechatronics topics, ALTEN has internal demo projects. An intern or graduate student is often linked to these projects, who then further develops the consultant's ideas and preliminary work. For me, that's the 4-in-1-row robot.

ALTEN has developed a robot in-house that can play 4-on-1 against a human opponent by means of an algorithm (Figure 1). Using industrial components, the 4-in-1-row robot is built to demonstrate the knowledge of different systems. The robot will be used as a demonstration unit at trade fairs and open days, where it will be available for passers-by to play a round. At the front, the player can then place his/her move on the board, after which the 4-in-1-row robot will devise and execute a move. For this purpose, the system keeps track of which moves have been played by its opponent. When the move has been determined by the robot, the combination of the XZ platform with rotating vacuum gripper will pick up a stone and insert it into the game at the chosen spot. Once the game is over or reset, the Match 4 robot will collect all the chips and sort them by color to be ready for the next round.

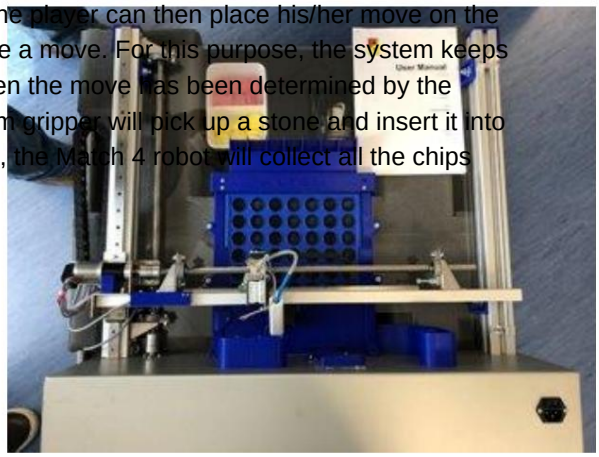


Figure 1 4-in-1 row robot

2 Project results

Objective
Currently the 4-in-1-row robot runs on a Raspberry Pi + STM32 controller. The Raspberry Pi runs the algorithm to determine the system's next move and the STM32 controller runs the operating system. This project was initially used within ALTEN to give consultants who were not at a company during an interim period a challenging project. As a result, several consultants worked on it over a longer period of time. This has resulted in a very confusing and unclear architecture of the software and hardware. This also applies to the operating system of the 4-in-1-row robot.

This unclear architecture makes it very difficult to expand or upgrade the operating system. Because the 4-in-1-row robot is also exhibited at trade fairs and demonstrates what ALTEN has to offer, it is essential that the components and structure grow with market requirements.

A complete redesign of the operating system is necessary to solve the cluttered and unclear architecture and to enable upgrades. For this, choices must be considered and a Software Architecture Document (SAD) must be made. Communication with various components and peripherals is also crucial, for which Board Support Packages (BSP) must be set up on the basis of drawn up diagrams. Then everything has to be converted into a functional embedded low level controller based on a STM32H7 dual core processor.

Problem statement
How do I set up a structured and modular architecture that can facilitate all changes and extensions, after which I can implement this architecture on a STM32H7 dual core processor (Figure 2) for the operating system of the 4-in-1 robot.

Figure 2 STM32H7

Project Result
Flowcharts;
A SAD; A
BSP to be made; Individual
testing of the BSP modules; One core of the
STM32H7 should be used for real-time motion control; The second core of the STM32H7
should be used for machine operation; modules work.

3 Project activities

The 4-in-1-row robot is an existing project. Within this project, the operating system must be redeveloped. By building the architecture from the ground up in a modular way, structure is created in the process. To achieve this, it is important to formulate a clear assignment in advance and to define frameworks. I will do this by applying the V-model.

The V-model is a project method that structures the progress of the project. For each specification or design phase on the left, there is a corresponding integration phase on the right. Each phase on the right side of the project can be verified and validated by the phase on the left (Figure 3).

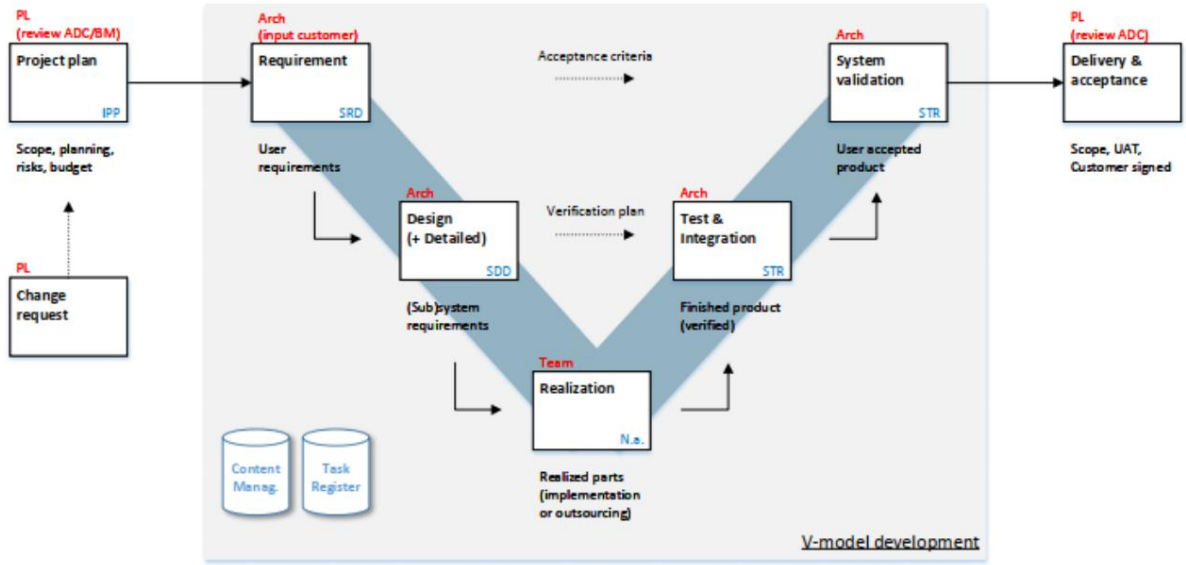


Figure 3 V-model within ALTEN

The first four weeks are all about drawing up the requirements. This is done by consulting with my technical supervisor and discussing and adjusting the expectations and (im)possibilities. Furthermore, existing documents about the 4-in-1-row robot will have to be read. To become familiar with the programming environment to be used (STM32cubeIDE), an STM32 development board can be used to master the basics of an STM32.

Once the assignment has been clearly defined and the requirements have been drawn up, a start can be made on the design phase. This is done by setting up a SAD of the 4 in 1 row robot.
Here diagrams are drawn up to visualize the relationships between the software and hardware.

If the SAD has been approved, the realization phase can be started. The realization phase consists of the architecture implementation of the BSP modules. The SAD is the largest part of my internship and will take up most of my time.

In the test and integration phase, the developed embedded software can actually be tested and implemented on an STM32H7 dual-core processor development board that is available at ALTEN. The 4-in-1-row robot itself is also available and (partly) functional. If part of the setup does not work, a mock-up code must be made to test the specific BSP anyway. A mock-up is a piece of code that allows you to test the operation of a software block without physical components.

The final stage is system validation. For me, this phase is handing in my graduation report and defending my project at the Fontys.

To keep up to date with what the other interns are doing and to indicate if problems arise, a daily stand-up meeting has been scheduled. Here you briefly tell what you did the day before, what you are going to do today and, if applicable, any problems you encounter.
This way everyone is aware of each other's work and people can share knowledge if someone gets stuck. Furthermore, there is a short demo every Friday as a kind of weekly summary. ALTEN consultants are also present here who can give tips or hints if you get stuck.

Unfortunately, due to corona, a timetable has been drawn up for the days that the interns are allowed to be present at the office. This means that people work at the office two days a week and at home two days a week. On Friday there is the possibility to come to the office if this proves to be necessary. You can also consult with the business manager if you need to work with hardware to come to the office for several days.

4 Project boundaries and preconditions

Defining a project properly is very important, so that at the end of the graduation internship you can account for what has and has not been achieved. To begin with, the time and budget are defined. This project has a duration of 100 days and will be carried out within that period.

Starting date:	**01-02-2022**
End date:	**6/31/2022**
Budget:	There is no budget. The product is developed at no cost.

The preconditions provide the further demarcation of the project.
Conditions

The first core of the STM32H7 is programmed using C or C++;
For the second core of the STM32H7, the programming language is free choice;
Flow charts must be drawn up to determine and map the flow through the system;

Diagrams must be made that show the relationship of the hardware to software;

Before a step can be completed, approval must be obtained from the technical manager;
The PvA must be approved by the school;
Documentation (SRD, SAD, Final Report) should be made to track progress and for later research/expansion; Presence at the office on the designated days is mandatory (Monday and Wednesday); Presence at the office outside the designated days must be discussed with the business manager; A timesheet must be completed on time; The project is developed into a proof of concept; The hardware and mechanics of the system are not included in this project; supervisor.

I aim to redesign the operating system within the 100-day internship period according to a structured architecture to the functionality that the old operating system also had. If there is still time left, additional functionalities can be implemented.

The project is completed when all stakeholders are satisfied with the end result. For this project that is: Aniel shri as technical manager, Gijs Haans as business manager, ALTEN Mechatronics department as final responsible, Jeedella SY Jeedella as school mentor and ALTEN as end customer.

Stakeholders

Stakeholders play a major role in the project. It is important to identify who is involved in the project. Figure 4 shows a visual representation of the stakeholders.
Wrong! Reference source not found. indicates which persons have which role within the project and what everyone's share is.

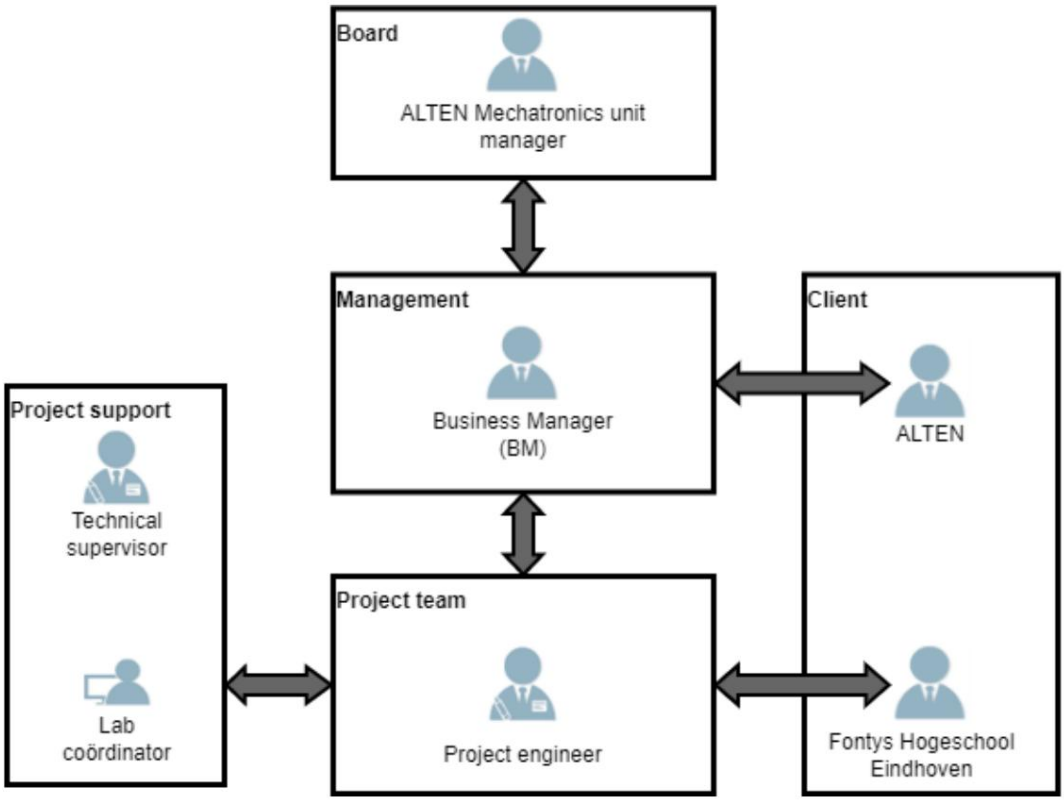


Figure 4 Project organization

Table 1 Distribution of roles

Name	Role	Responsibility
Pascal Fatz	Project engineer	Execute the project according to requirements for the customer (board / management ALTEN)
Aniel Shri	Technical supervisor	Providing technical support to the project team to achieve a good end result, guiding the individual development of the members of
Gijs Haan	Business manager	the project team. The business manager also monitors the process and the planning. The project, process and the student are supervised by Fontys on a technical and individual level. At the end
Jeedella SY Jeedella	Fontys University of Applied Sciences Eindhoven	of 20 weeks, the project is assessed by the Hogeschool supervisor.
Jeroen Wilbers	lab coordinator	Coordinating the safe use of the lab and point of contact for ordering components.
Aniel Shri and Gijs Haans (as client)	ALTEN	Assess the requirements as a customer and ultimately accept the end product.
Chris Kalis	ALTEN Mechatronics unit manager	Client and ultimately responsible for the project

5 Intermediate results

Interim results are important to show that there is progress in the project. The progress within the project is guaranteed by means of the following interim results.

-SRD; -
SAD; -
Approval total system architecture; - Develop a BSP
module for the STM32H7 per module diagram; - Conduct a test per BSP module; -
Distribution chart per core of the STM32H7.

6 Scheduling

It is important to have good planning. I received a template for a Gantt chart from ALTEN. I have filled this in based on what I think I am now subjecting and the tasks that need to be completed (Appendix B). Furthermore, all deadlines are shown in Table 2.

Table 2 Deadlines

deadlines:	Date:
Plan of approach	2/18/2022
SRD concept	11-3-2022
sad concept	3/25/2022
Test report	7/1/2022
Final report	6/7/2022 (subject to change)
Graduation session	6/17/2022 7/18/2022 (subject to change)

7

Because my project involves a redevelopment of the operating system of a 4-in-1-row robot, an upgrade k project must be incorporated in a risk analysis in Appendix A. As can be seen in the graph in Appendix A, the biggest pitfall is the complexity of the projects. Time can also become a trap. Further pitfalls that are not mentioned in the risk analysis are: Illness due to corona, but also corona itself.
Should corona flare up again and working from home become the norm, this could cause delays.
Ordering components can take a long time. Due to the shortage of material in the market, it can take a long time before a component is delivered.

If one of the pitfalls should occur, it is necessary to inform the right people as soon as possible and to ask for help. These are my technical supervisor Aniel about technical pitfalls and my business manager Gijs about person-related pitfalls. In both cases, I must also notify the school of a pitfall. Appropriate solutions will have to be found together with these people and the school.

Appendix A. Risk analysis

Risk analysis	**Print**
4-in-1-row robot	*2/9/2022*

With a risk percentage > 50%, the project should not be carried out in this form.

Category Risk **Time Factor** 1		Value	* Factor	** Gravity	** risk tot.
		J.make choiceJ.			
	Estimated duration of the project	3 - 6 months 1		4	4
2	Does the project have a definite deadline	Yes	2	4	8
3	Is there enough time to complete the project?	Enough	1	4	4
Complexity of the project 4		J.make choiceJ.			
	Number of functional sub-areas involved	3+	3	4	12
5	Number of functional sub-areas that will make use of the results	4	2	2	4
6	Is it an adjustment or a new project?	Major adjustments	2	5	10
7	To what extent will existing responsibilities need to change	Average	2	5	10
8	Are there other projects depending on this project No		0	5	0
9	What will be the attitude of the users	Interested 1		5	5
10	Are there sub-projects, progress depends on the coordination between them	Somewhat	2	3	6
The project management		J.make choiceJ.			
16	Is the project management subject matter expert?	Very knowledgeable	0	3	0
17	How knowledgeable is the project management with regard to project planning	Very knowledgeable	0	3	0
18	How much experience does the project leader have with projects experience like this	Lots of	0	3	0
19	How knowledgeable are the consultants in the area to be investigated	Very knowledgeable	0	5	0
20	How knowledgeable are the subject matter experts in the area to be researched	Very knowledgeable	0	5	0
21	How involved are the responsible line managers the project	Strongly involved 0 in		5	0
22	Is there a good chance that the composition of the project group will change during the project?	Small chance	0	5	0
23	Are standard methods used by the project group	Yes, number 2		4	8

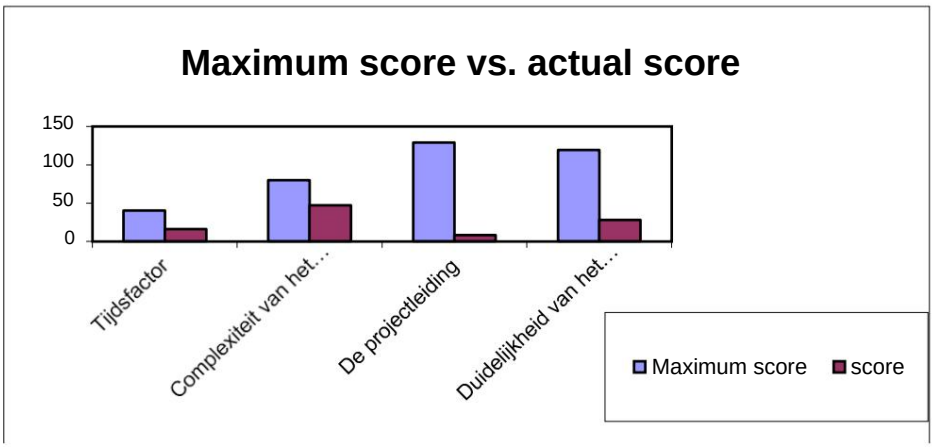
Category	Risk	Value *	Factor **	Gravity **	risk tot.
Clarity of the project 24		Make choice!			
	Are problem and objective sufficiently known to all project members	Yes, everyone	0	5	0
25	Is the research area accurately defined?	Reasonable	2	5	10
26	Is there sufficient demarcation with other projects?	Reasonable	1	4	4
27	Is there sufficient time planned for coordination and decision-making?	Enough	0	4	0
28	Are the preconditions clear?	Most do 1		4	4
29	Are the preconditions restrictive enough?	Reasonable	2	5	10
			Total 99		
			Risk rate * 23%**		

Note: This model only gives a very rough indication of risks

* *Value chosen by project manager.*
** *Height factor and value are fixed.*
*** *Risk percentage is the total score divided by 433 (maximum score) times 100.* Note: this is only an indication

Since the risk percentage gives an overall picture, it is possible that a certain category does provide a high risk. Below is a specification per category to make possible areas for improvement visible.

Category (with maximum score vs actual score)				
Time factor	Maximum	40	Scoring	16
Complexity of the project	Maximum	80	Scoring	47
The project management	Maximum	129	score	
Clarity of the project	Maximum	119	score	8 28



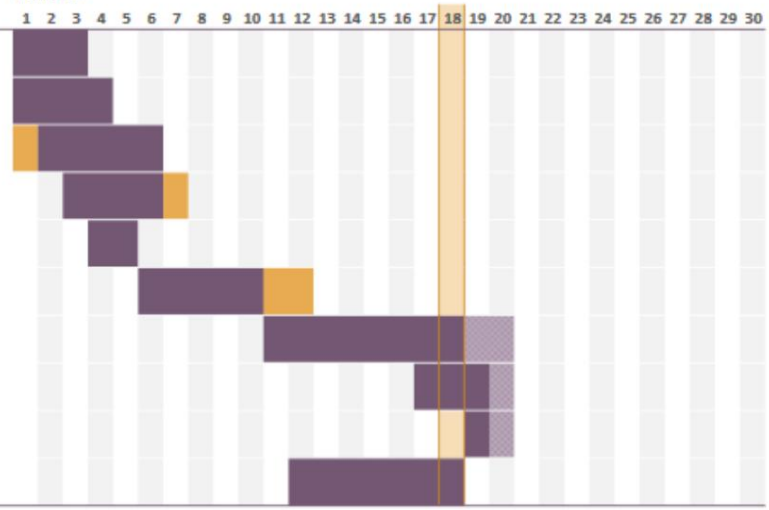
Appendix B. Scheduling

Project Planner

Select a period to highlight at right. A legend describing the charting follows.

Period Highlight: 18 | Plan Duration | Actual Start | % Complete | Actual (beyond plan) | % Complete (beyond plan)

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Plan van aanpak	1	3	1	3	100%
Documentatie doorlezen	1	4	1	4	100%
STM32cube omgeving leren	2	5	1	6	100%
Onderzoek hardware	3	4	3	5	100%
SRD	4	2	4	2	100%
SAD	6	5	6	7	100%
Programmeren BSP's STM32H7	11	10	11	10	80%
Testen	17	4	17	4	80%
Test rapport	19	2	19	2	20%
Eindverslag	12	7	12	7	100%



III. Software Architectural Document

Software Architecture Document

4 in 1 row robot

Version 3
Date: 5/3/2022

ALTEN
Pascal Fatz

Final version

Version history

Version	Date	Status	Writer	Remark
1	3/31/2022	Design	Pascal Fatz	Initial draft to chapter 8
2	4/13/2022	Design	Pascal Fatz	Process feedback Jeedella, Aniel and Berend
3	5/3/2022	Completion	Pascal Fatz	Chapter 8 and beyond

Acronyms and abbreviations

Term	Explanation
BSP	Board Support Package

Referenced Documents

ID	Reference	Title	Date	Writer
D1	SRD_Pascal_Faatz_V3.docx	SRD	3/16/2022	Pascal Fatz

Table of contents

List of figures

List of tables

1 Introduction and goals

ALTEN has developed a robot in-house that can play 4-on-1-row against a human opponent using an algorithm. Using industrial components, the 4-in-1-row robot is built to demonstrate the knowledge of different systems. The robot will be used as a demonstration unit at trade fairs and open days, where it will be available for passers-by to play a round. At the front, the player can then place his/her move on the board, after which the 4-in-1-row robot will devise and execute a move. For this purpose, the system keeps track of which moves have been played by its opponent. When the move has been determined by the robot, the combination of the XZ platform with rotating vacuum gripper will pick up a stone and insert it into the game at the chosen spot. Once the game is over or reset, the Match 4 robot will collect all the chips and sort them by color to be ready for the next round.

Currently, the 4-in-1 robot runs on a Raspberry Pi + STM32 microcontroller (Figure 1). The Raspberry Pi runs the algorithm to determine the system's next move and the STM32 controller runs the operating system. This project was initially used within ALTEN to give consultants who are not on a project in an interim period a challenge. As a result, several consultants worked on it over a longer period of time. This has resulted in a very confusing and unclear architecture of the software and hardware. This also applies to the operating system of the 4-in-1-row robot.

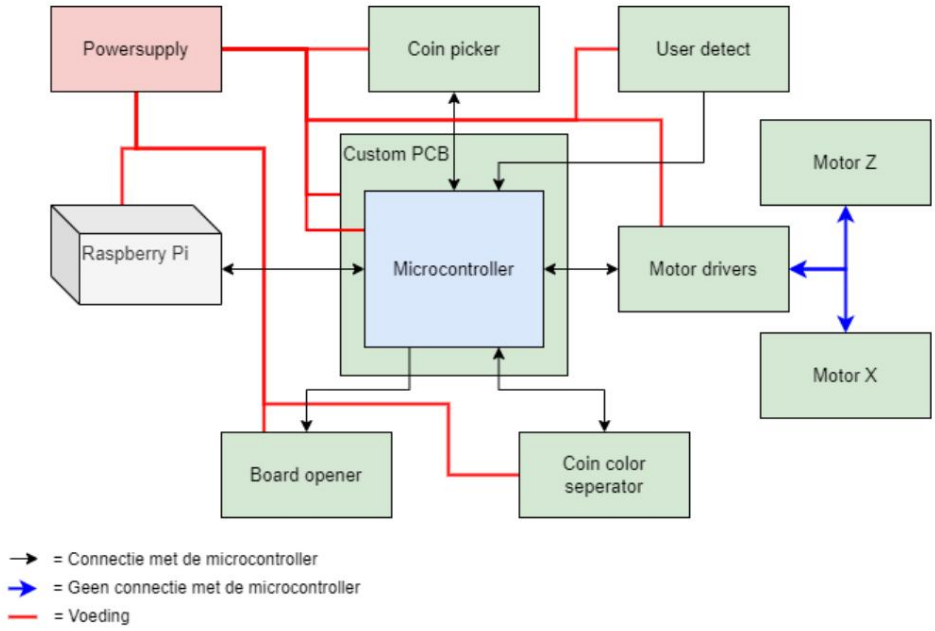


Figure 1 Block diagram 4-in-1 row

1.1 Requirements overview

Due to the unclear architecture, it is very difficult to expand or upgrade the operating system. Because the 4-in-1-row robot is also exhibited at trade fairs and demonstrates what ALTEN has to offer, it is essential that the robot is easy to maintain and can easily be upgraded. This requires a complete redesign of the operating system to solve the cluttered and unclear architecture and to enable upgrades.

One of the main requirements is therefore to introduce a structured architecture and modular implementation. It is also important to draw up state and flow diagrams to visually represent the operation of the system. A BSP (Board Support Package) must also be made.

For the full list of requirements refer to the SRD (D1).

1.2 Quality goals

The quality goals are essential to indicate which quality requirements the system must meet. These goals are also tested at a later stage and checked whether the system meets these requirements.

Table 1 Quality goals

Priority 1	Quality goal	Concrete scenario
	Easy to understand (Structured architecture)	People who are going to work on the 4-in-1-row robot must immediately understand how the hardware and software are connected by reading the architecture.
2	Maintenance and upgrades (Modular construction of software blocks)	People who are going to make an upgrade or modification to the 4-to-1 row must be able to modify or replace software blocks without affecting other parts of the system.
3	Robust	The 4-in-1 row must be reliable and work under all conditions when the system is running.

1.3 Stakeholders

The stakeholders are an important part of the project. They determine the requirements and set requirements for the end product. Figure 2 shows the distribution of stakeholders for this project.

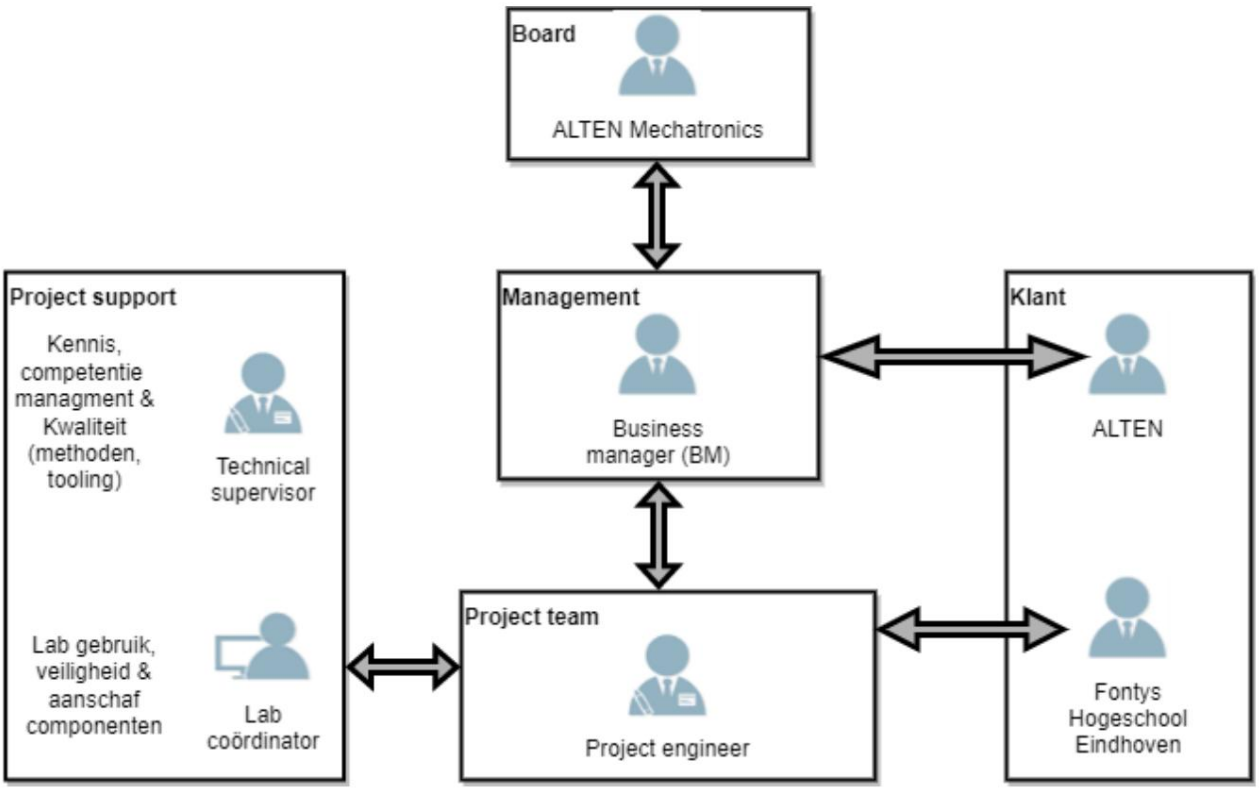


Figure 2 Project organization

Table 2 Stakeholders

Name	Role	Responsibility
Pascal Fatz	Project engineer	Carrying out the project for the relevant customers and board/management.
Aniel Shri	Technical supervisor	Provide technical support to the project team within the project. This involves challenging the team to come up with a good solution and implementation.
Gijs Haan	Business manager	Supporting the project team within the project personal development. Furthermore, the business manager.
Jeedella Jeedella	Fontys University of Applied Sciences Eindhoven	Support the project from school on a technical and personal level. At the end, give an assessment of the project.
Jeroen Wilbers	lab coordinator	Safe use of the lab and the point of contact for orders that need to be made.
Aniel Shri and Gijs Haans (as customer)	ALTEN	Assess the requirements as an end customer and use the end product.
Chris Kalis	ALTEN Mechatronics unit	The management of all project parties within ALTEN. Furthermore, the department is ultimately responsible for the project.

2 Architecture constraints

Table 3 Architecture

constraints **Architecture constraint**	**Description**
A microcontroller is used The choice of the microcontroller is fixed. must therefore be worked.	STM32H755 dual-core nucleo-144 board. microcontroller
The hardware of the 4-in-1 row is fixed. The hardware that is now available is used to write the software.	
The game progression of the 4-on-1 row is fixed.	Because the 4-on-1 row has already been working, the global works are known in advance.
The project must be completed within 100 working days.	The project takes place within a school semester and must therefore be carried out within this time.
The Raspberry Pi passes the next move.	The Raspberrry Pi is the processor that passes the next move to the microcontroller.

3 Project scope and context Figure 3

gives a global representation of the people who work with the system and what they have as input on the system or what they get back from the system. Furthermore, it shows the scope of the project. The scope of the project is the STM32H7 dual-core and its architecture.

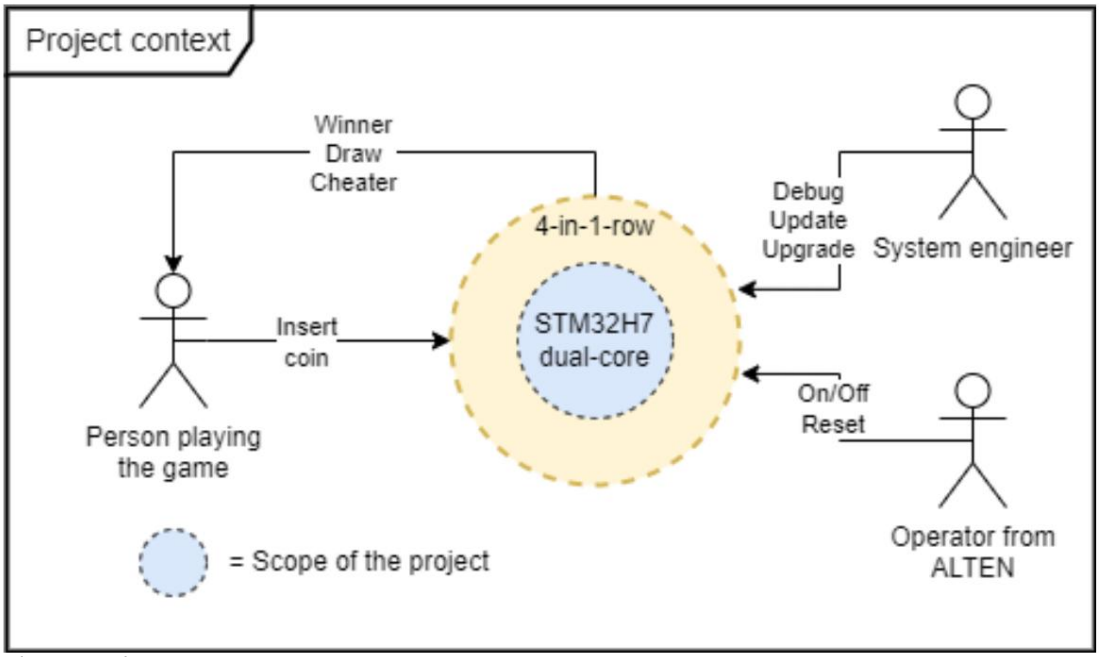


Figure 3 Project context

Table 4 Description Project context

Users	Description
Person playing 4-in-1	The person who plays 4-in-1 has to think carefully when it is his turn to put his chip to defeat the robot. When a move is known, the chip is placed in the 4-in-1 board. Furthermore, at the end of the game, the person can see what the result is.
System engineer	The system engineer can debug the 4-in-1 array if an error occurs. Furthermore, the engineer can implement an update/upgrade.
Operator from ALTEN operating the 4-in-1 row	This person operates the 4-on-1 row at fairs or other occasions. This person can turn the 4-in-1 row on/off or reset it.

3.1 System context

Figure 3 gives a global overview of the people involved in the system. Figure 4 zooms in further on this and provides a visual representation of the subparts of the 4-to-1 row. It also shows the relationship with the STM32H7 dual-core through the inputs and outputs. This is important so that all stakeholders get an idea of what is happening in the system in general terms.

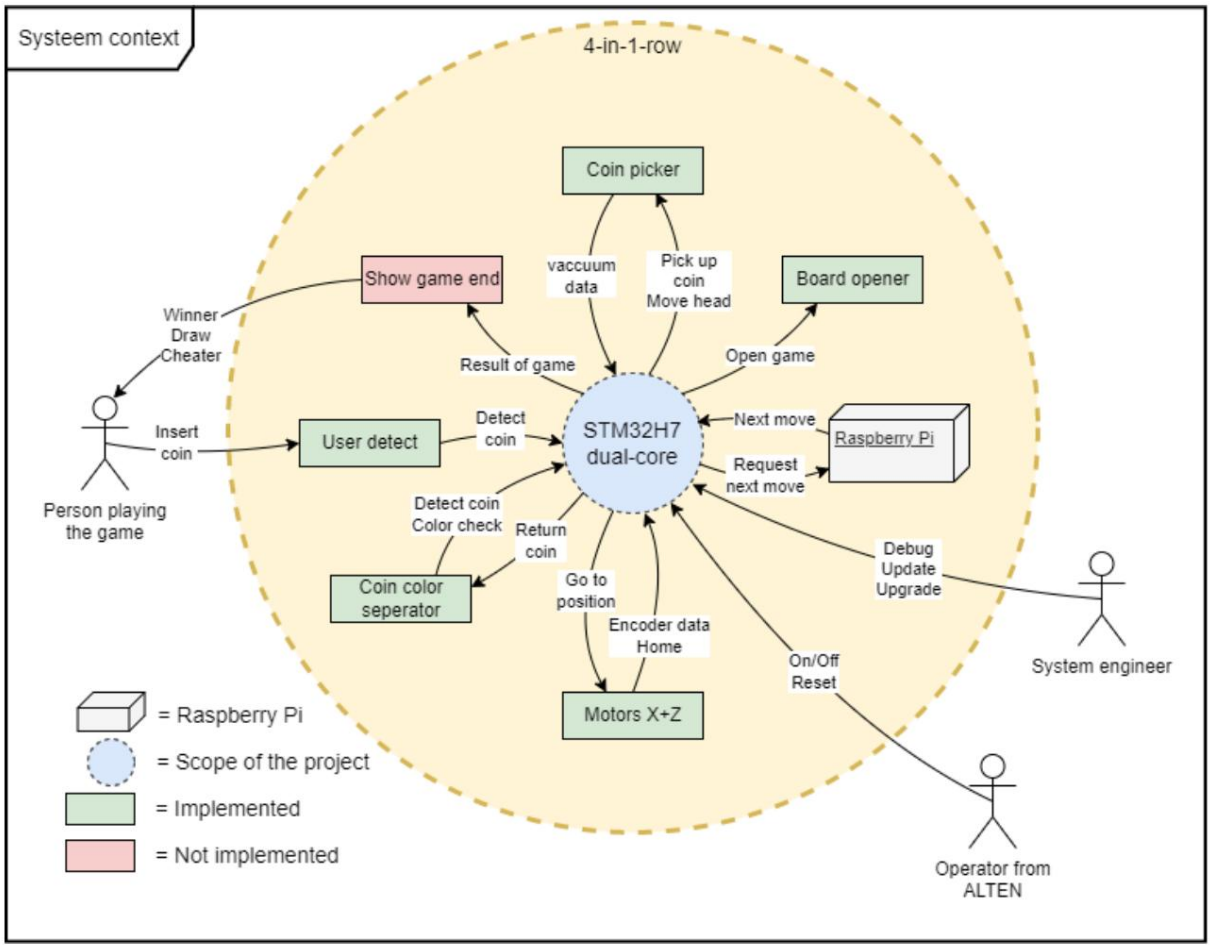


Figure 4 System context

Table 5 Description System context

Hardware block	Function
RaspberryPi	The Raspberry Pi runs an algorithm that determines the robot's next steps.
Engines X+Z	The motors allow the robot to move its vacuum gripper over an X and Z axis of the 4-in-1 row board.
Coin color separator	When the game is over, the coin color separator ensures that each chip is recognized by color. If the chip has the player's color, the chip is shot to the player's bin. If the chip has the color of the robot, the chip is placed in its own stack.
User detect	When the player puts a chip in the game, this is detected. Each column of the 4-in-1 board contains a detection point.
coin picker	The coin picker is the arm of the robot. This is moved by the engines, but can also move itself to put a chip in the 4-in-1-row board. Picking up or releasing a chip is done by means of a vacuum gripper.
Board opener	The board opener causes all chips to fall out of the 4-in-1 board when the game is over.

3.2 Technical context

Figure 5 converts Figure 4 into a description into a technical representation of the relationship between the components and the scope. The connections are not described globally but are shown by what kind of protocol or input/output signal they are connected. This is of interest to the stakeholders who get to work with the hardware or architecture.

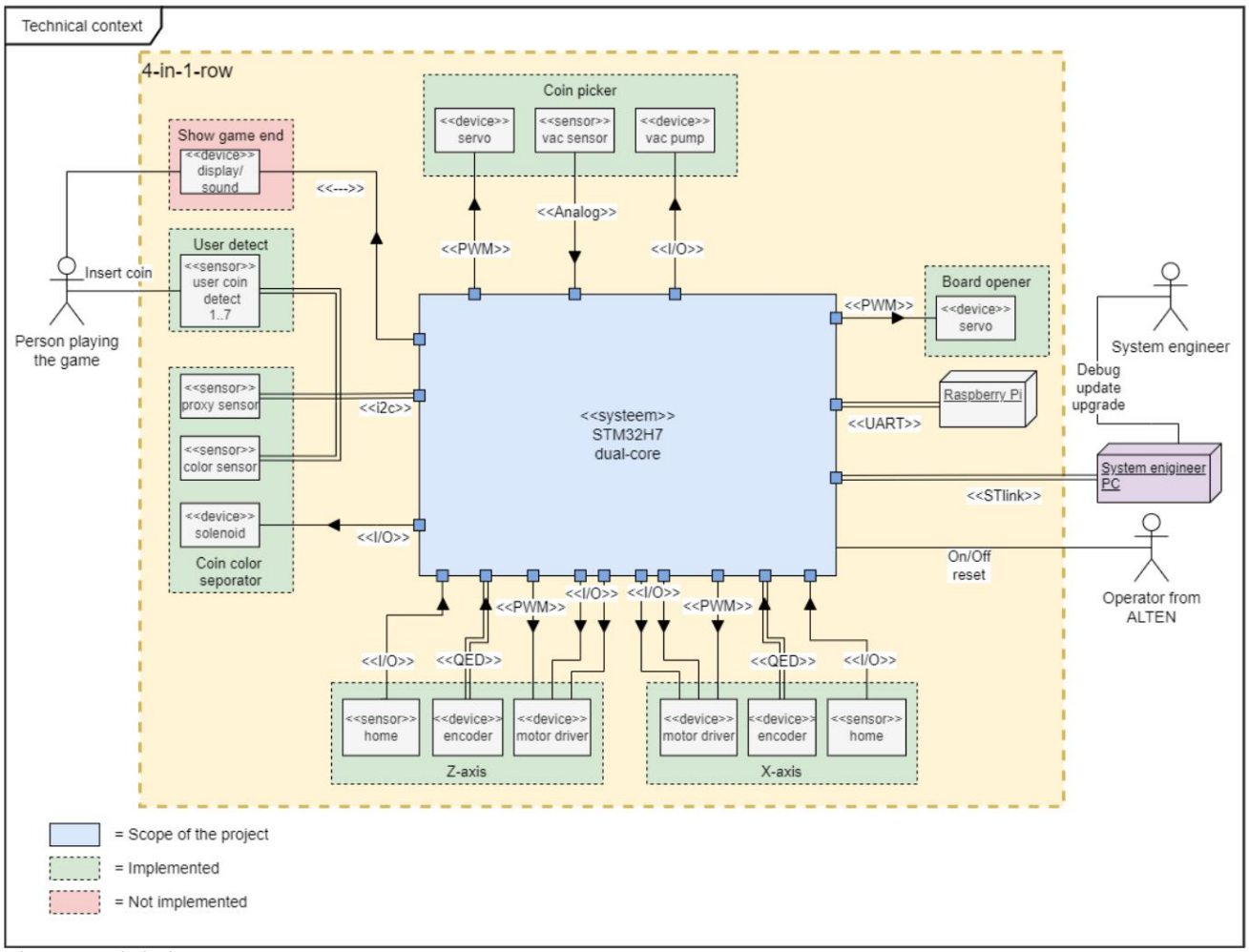


Figure 5 Technical context

Table 6 Description Technical context

Hardware block	Sub blocks	Description	Input to STM32H7	Number of lines	output from STM32H7	Number of lines
raspberry Pi	-	The Raspberry Pi communicates with the STM32H7 through a UART connection.	UART RX	1	UART Tx	1
Motors/ driver X	Motor with encoder	The motor is controlled by a PWM signal, a line for the direction and a line for the ready signal. An encoder is used to determine the position of the motor. The encoder communicates with an A and B line.	QED (A line and B line)	2	1 PWM, 2 I/O	3

	Home/ End stop	The home/end stop indicates when the motor has reached the home point.	IO	1	-	-
Motor/ driver Z	Motor with encoder	See Engine X.	QED (A line and B line)	2	1 PWM, 2 I/O	3
	Home/ End stop	See Engine X.	IO	1	-	-
Coin color separator	proxy sensor	The proxy sensor checks whether a chip is present in the distribution system.	i2c (SDA, SCL)	2	-	-
	Colour sensor	The color sensor checks which color the chip in the distribution system has.	i2c (SDA, SCL)	2	-	-
	Solenoid	When a chip has the player's color, the solenoid shoots the chip to the player's bin by means of a flipper.	-	-	IO	1
User detect	-	The user detect detects when a chip is thrown into the system by the player.	i2c (SDA, SCL)	2	-	-
coin picker	Servo	The servo ensures that the vacuum gripper can be moved.	-	-	PWM	1
	Vacuum pump	The vacuum gripper can suck and release a chip.	-	-	IO	1
	Vac sensor		analog	1	-	-
Board opener	-	The game opener uses a servo to ensure that all chips fall out of the 4-in-1-row board when the game is over.	-	-	PWM	1

4 Solution strategy

Table 7 Solution strategy

goal/requirement	Approach	Link to chapter
structured architecture	This document is used to set up a structured architecture for the software of the 4-in-1 row. The chapters and diagrams in this document make it clear how the 4-to-1 row behaves, the software blocks are connected and the software communicates with the hardware. Furthermore, a top-down approach is used by starting with the requirements. From the requirements, the system is increasingly dissected.	
Modular construction of software blocks	The software is built up modularly to ensure that an upgrade or adjustment is easy to facilitate. First, it is worked out which software blocks there are. Then how these blocks communicate with each other.	(Chapter 5,6,8)
Robust	The 4-in-1 row must operate under normal conditions as predetermined. This can be guaranteed by testing all software blocks individually and extensively together.	

The V-model is used throughout the project. The V-model is a project method that structures the progress of the project. For each specification or design phase on the left, there is a corresponding integration phase on the right. Each stage on the right side of the project can be verified and validated by the stage on the left (Figure 6).

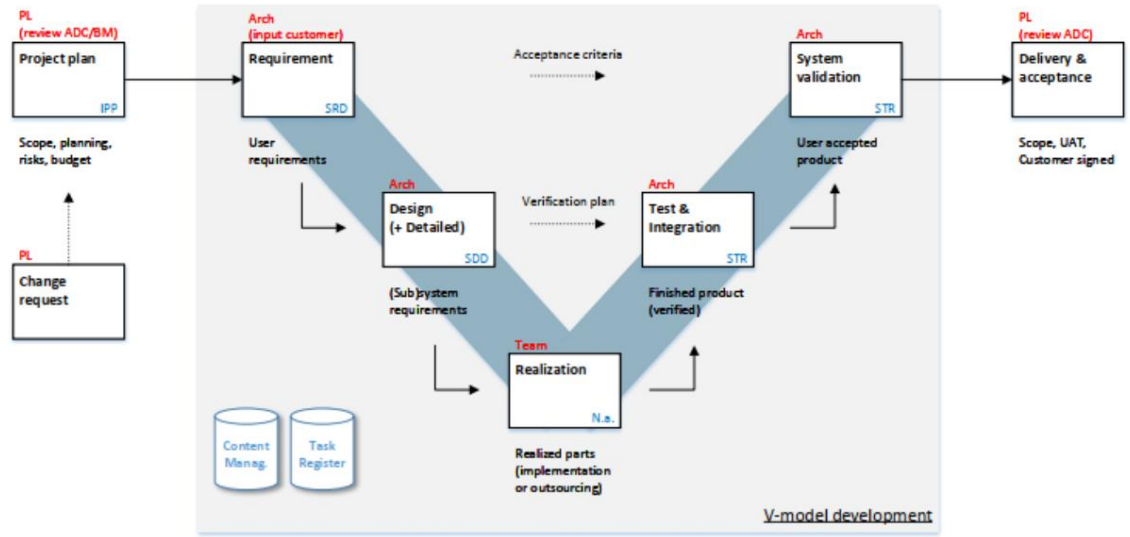


Figure 6 V model

5 Building block view This chapter

takes a closer look at the diagrams presented in chapter 3. The idea of a building block view is to zoom in on the system step by step. This is done through levels. Level 0 for this project is Figure 5 where the STM32H7 dual-core is a black box. In level 1, this STM32H7 dual-core becomes a white box that consists of several black boxes. In level 2, the black boxes from level 1 become white boxes containing black boxes. During each step, the black boxes describe what the responsibility and tasks are. This is a good way to analyze a system in a structured way. It is ideal to consult with the stakeholders on an abstract level without further details on how the implementation will take place. After this, modular software blocks can be created.

5.1 White box STM32H7 dual core

Figure 7 shows level 1 and zooms in on the STM32H7 dual-core. It also shows which core the inputs/ outputs of STM32H7 dual-core go to.

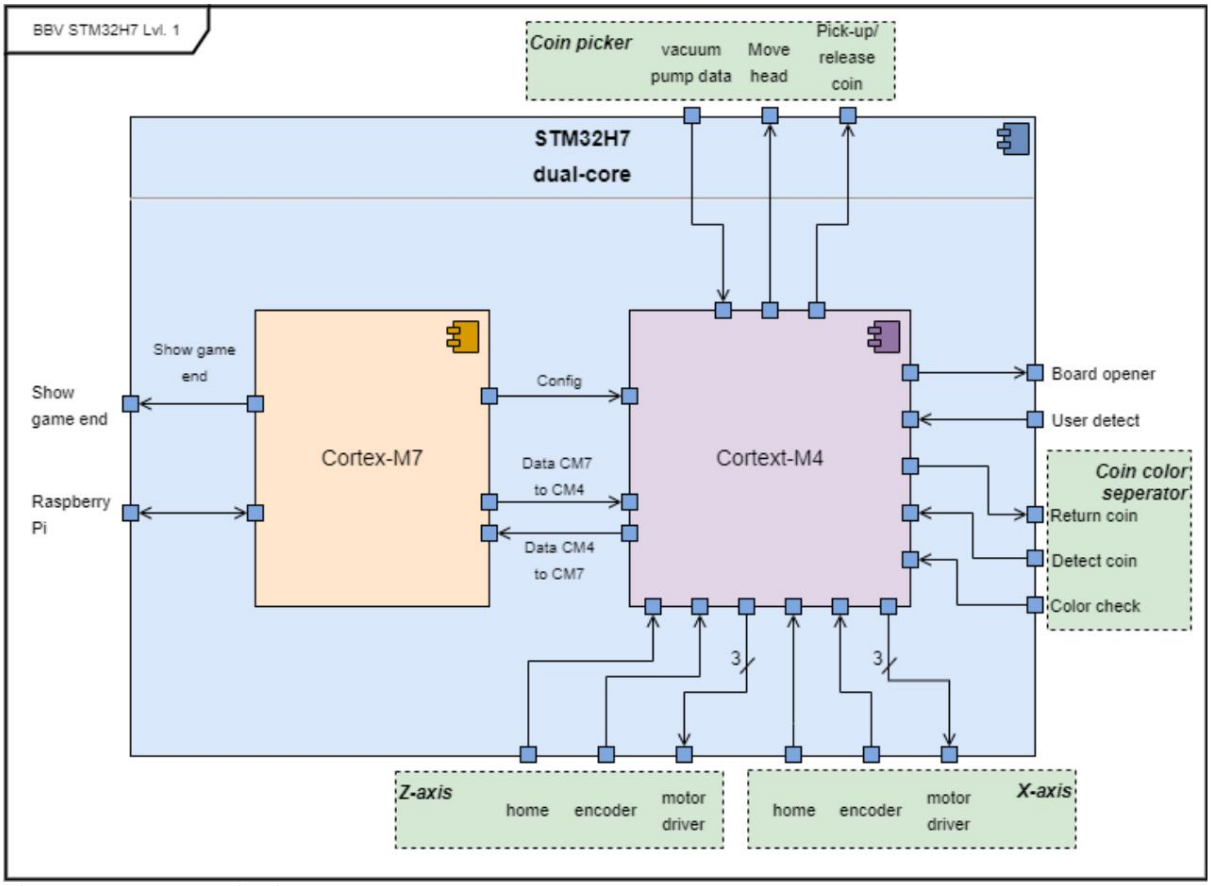


Figure 7 BBV STM32H7 level 1

Table 8 Description BBV STM32H7 level 1

Black box	Description
Cortex-M7	The Cortex-M7 core of the STM32H7 is used for machine handling. This core is responsible for the game flow, communicating with the Raspberry Pi, initializing the whole system and generating an output about the result of the game.
Cortex-M4	The Cortex-M4 core of the STM32H7 is used for real-time motion control. This core is responsible for handling all hardware components such as the Coin picker, Engines, Coin separator, Board opener and User detect.

5.2Level 2

This chapter turns the relevant black boxes from level 1 into white boxes and describes how the internal blocks are connected.

5.2.1 White box Cortex-M7

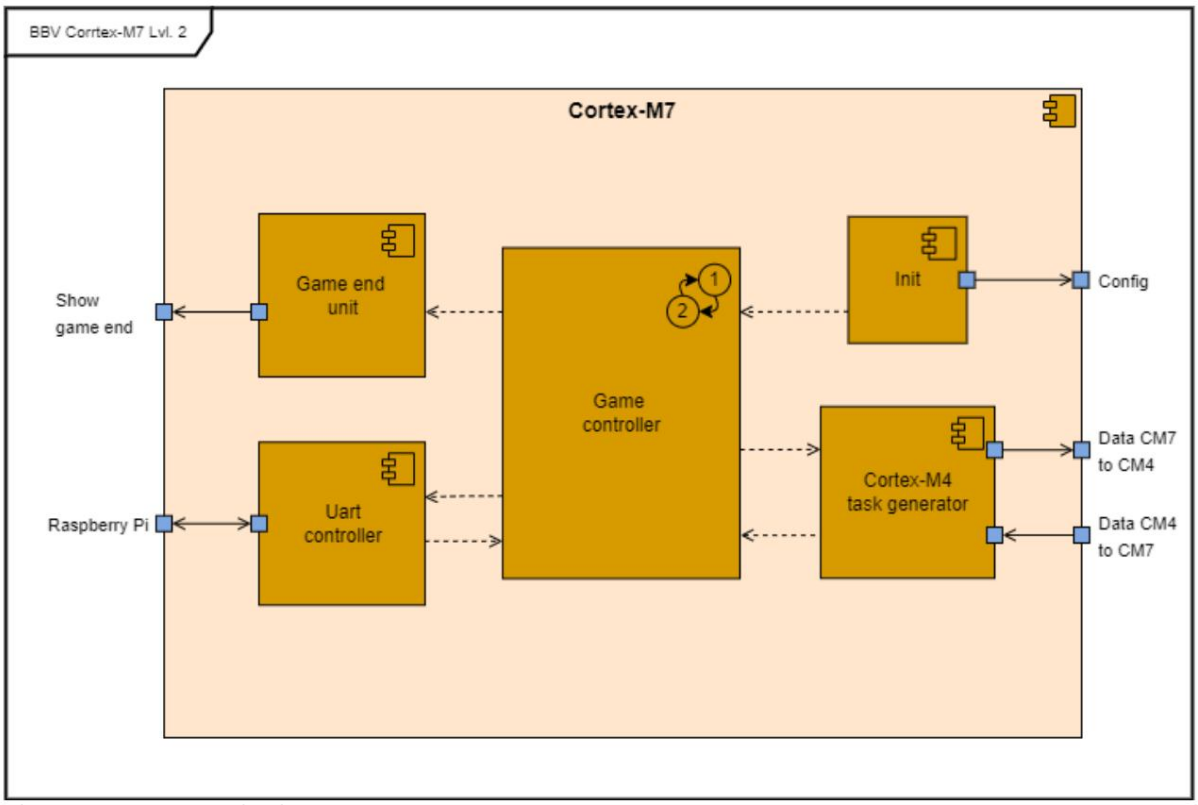


Figure 8 BBV Cortex-M7 level 2

Table 9 Description BBV Cortex-M7 level 2

Black box	Description
Game controller	The Game controller is responsible for the game flow through a state machine.
Init	Runs once to initialize and power up 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 communications to an output for the player.

5.2.2 White box Cortex-M4

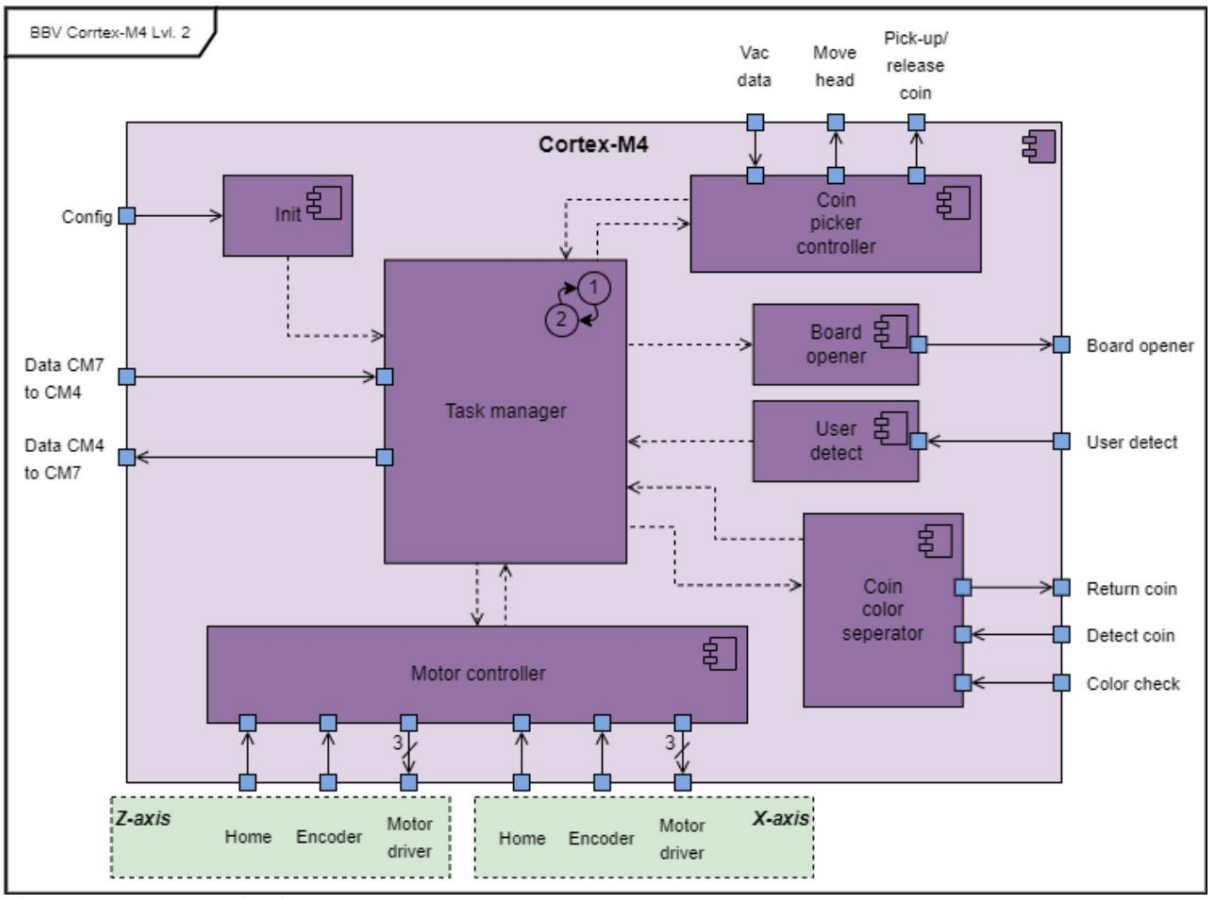


Figure 9 BBV Cortex-M4 level 2

Table 10 Description BBV Cortex-M4 level 2

Black box Description	The Task manager
Task manager	is responsible for the execution of the commands to be executed by the Cortex-M7 by means of a state machine. D the other components are built in a modular way. After all, they don't need to know about each other's existence/status. This information is maintained by the Task Manager.
Init	Runs once to initialize and power up the Cortex-M4. In this phase, the sensors are tested for presence and the protocol is off.
engine controller	The Motor controller implements all communication with the motor drivers and the PID controller.
Coin color separator	The Coin color separator implements all the functions for handling the chip separation.
User detect	The User detect implements all functions for handling the sensors for the player's input.
Board opener	The Board opener implements all functions for handling the opening of the 4-in-1 board when the game is over.
Coin picker controller	The Coin picker controller implements all the functions for handling the pick up and release of a chip.

5.3 Level 3 This

chapter turns the relevant black boxes out of level 2 white boxes and describes how the internal blocks are connected.

5.3.1 White box Motor controller

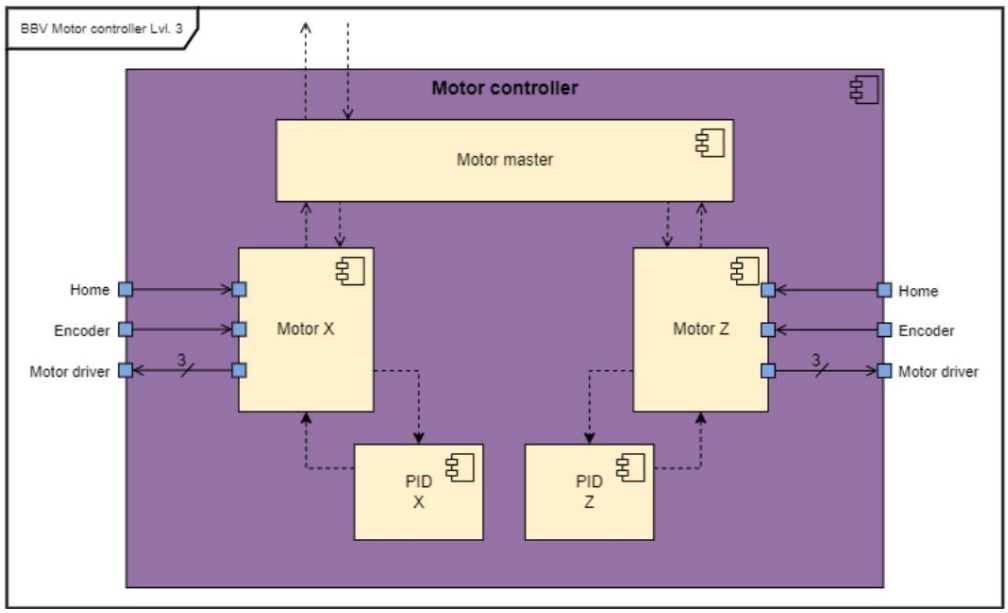


Figure 10 BBV Motor controller level 3

Table 11 Description BBV Motor controller

blackbox	*level 3*
Engine master	**Description** The motor master is responsible for controlling both motors.
Engine X	Motor X controls communication with the motor for the X axis.
PID X	PID X creates the control loop with feedback for motor X.
Engine Z	Motor Z controls communication with the motor for the Z axis.
PID Z	PID Z creates the control loop with feedback for motor Z.

5.3.2 White box Coin color separator

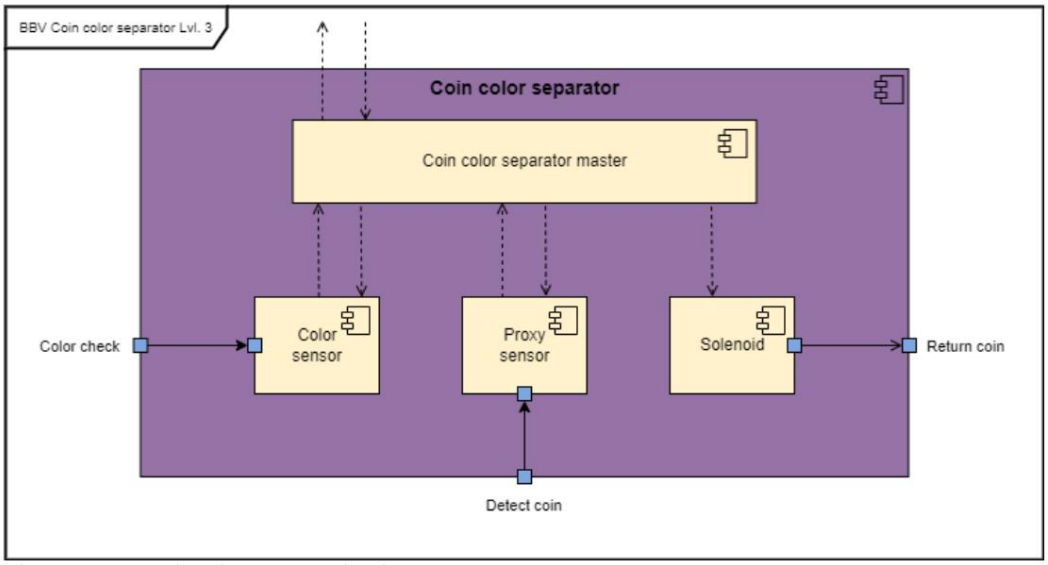


Figure 11 BBV Coin color separator level 3

Table 12 Description BBV Coin color separator level 3

Black box Description	The Coin color separator
Coin color separator master	master is responsible for receiving the sensor data and controlling the solenoid.
color sensor	Requests the data about the color of a chip.
proxy sensor	Gets data if a chip is present.
solenoid	Is responsible for activating the flipper.

5.3.3 White box coin picker

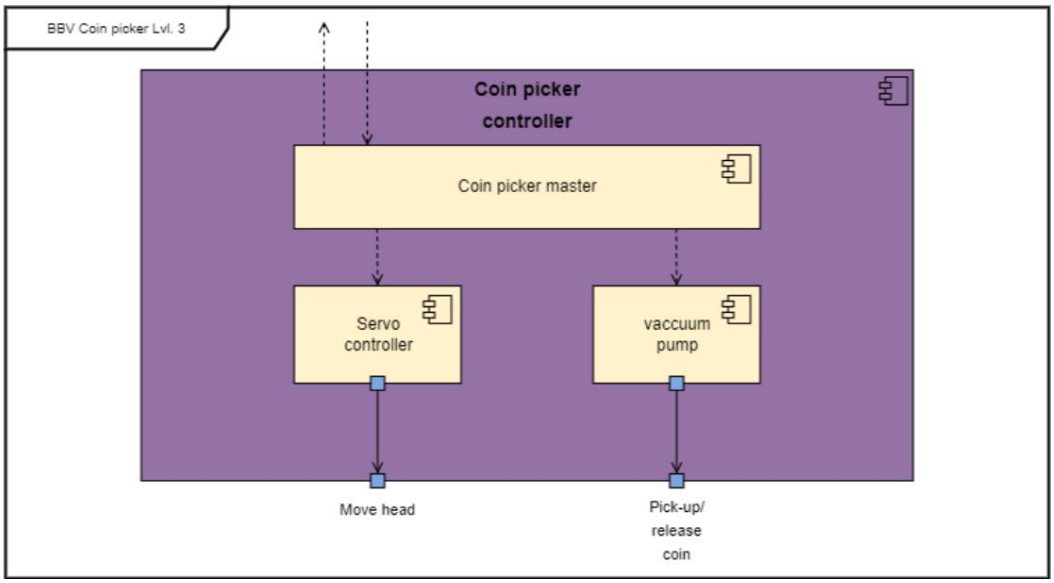


Figure 12 BBV Coin picker level 3

Table 13 Description BBV Coin picker level 3

Black box Description	**Coin picker** The
Coin picker master is responsible for controlling the **master** servo and the vacuum pump of the vacuum gripper.	
Servo controller	Controls the control of the servo motor.
Vaccuum pump	Controls the control of the vaccuum gripper to pick up the chips.

6 Run-time view

The Runtime view describes the concrete behavior and interactions of the building blocks of the

are shown in the subchapters.

6.1 High level overview

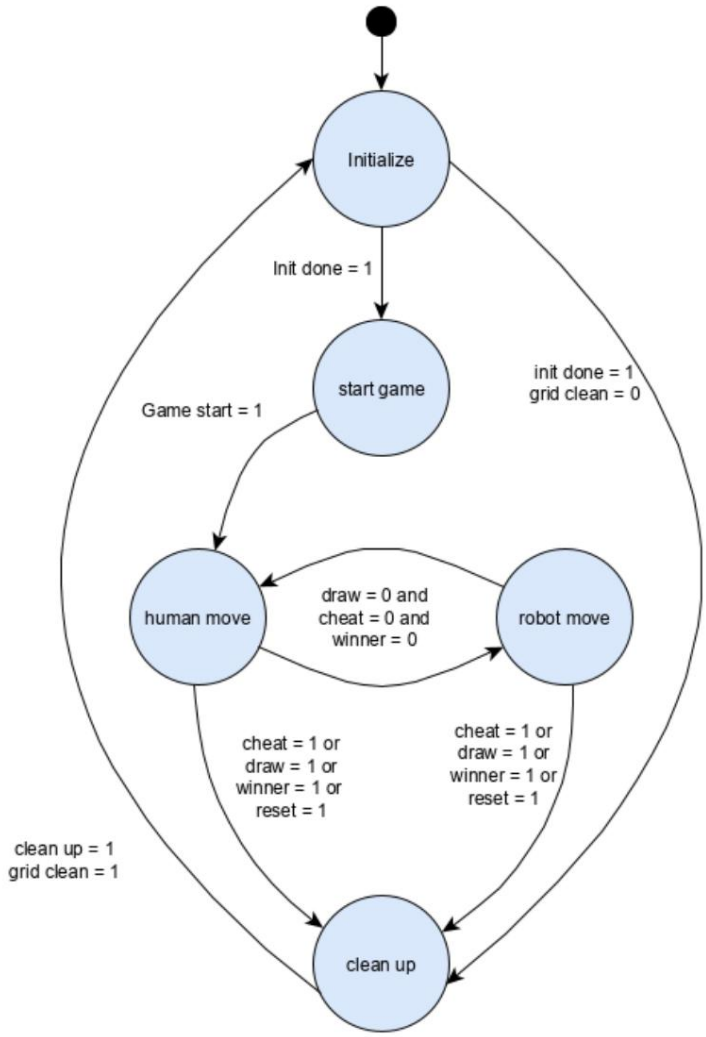


Figure 13 State machine Cortex-M7

The Statemachine in Figure 13 shows how the 4-to-1 rank acts at the highest level. This Statemachine runs on the Cortext-M7 within the game controller and is therefore the main flow of the 4-in-1 row. The first state is the initialize state. In this state, the entire system is booted and initialized. 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 start. When the game has started, it is the player's turn to be the first to put his/her chip in the 4-in-1-row game. Each turn, the robot checks whether there is a winner, cheat or tie. If not, the turn goes to the robot, the robot makes the calculated move. Once the game is over, the clean up state starts and all chips are cleared. The red chips go to the robot and the yellow chips to the player's bin.

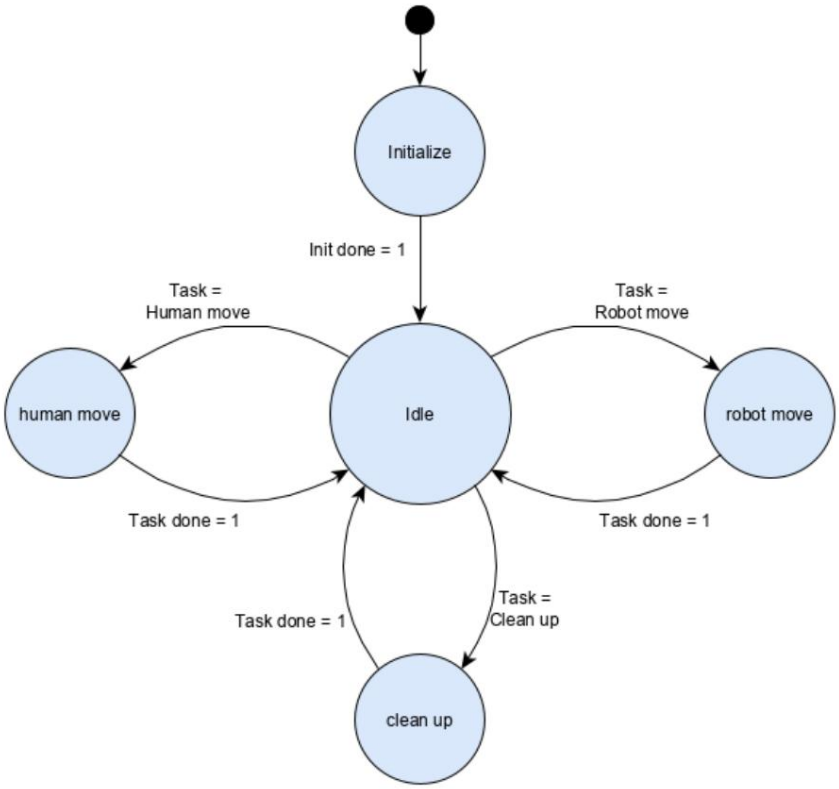


Figure 14 State machine Cortex-M4

As shown in Figure 9, the Cortex-M4 also houses a state machine. This state machine is shown in Figure 14 and runs on the Task manager module. state. In this state, the hardware components are configured and set up. After that, the

the Cortex-M7. As soon as a task arrives, the next state is determined based on the task.
be state. Each of these states directs the
hardware to complete the task. When the task is completed, the Task manager returns to the Idle state.

and clean-
shown in a sequence diagram. A sequence diagram shows how the different building blocks relate to each other over time and what kind of signal they send.

6.2 Robot moves

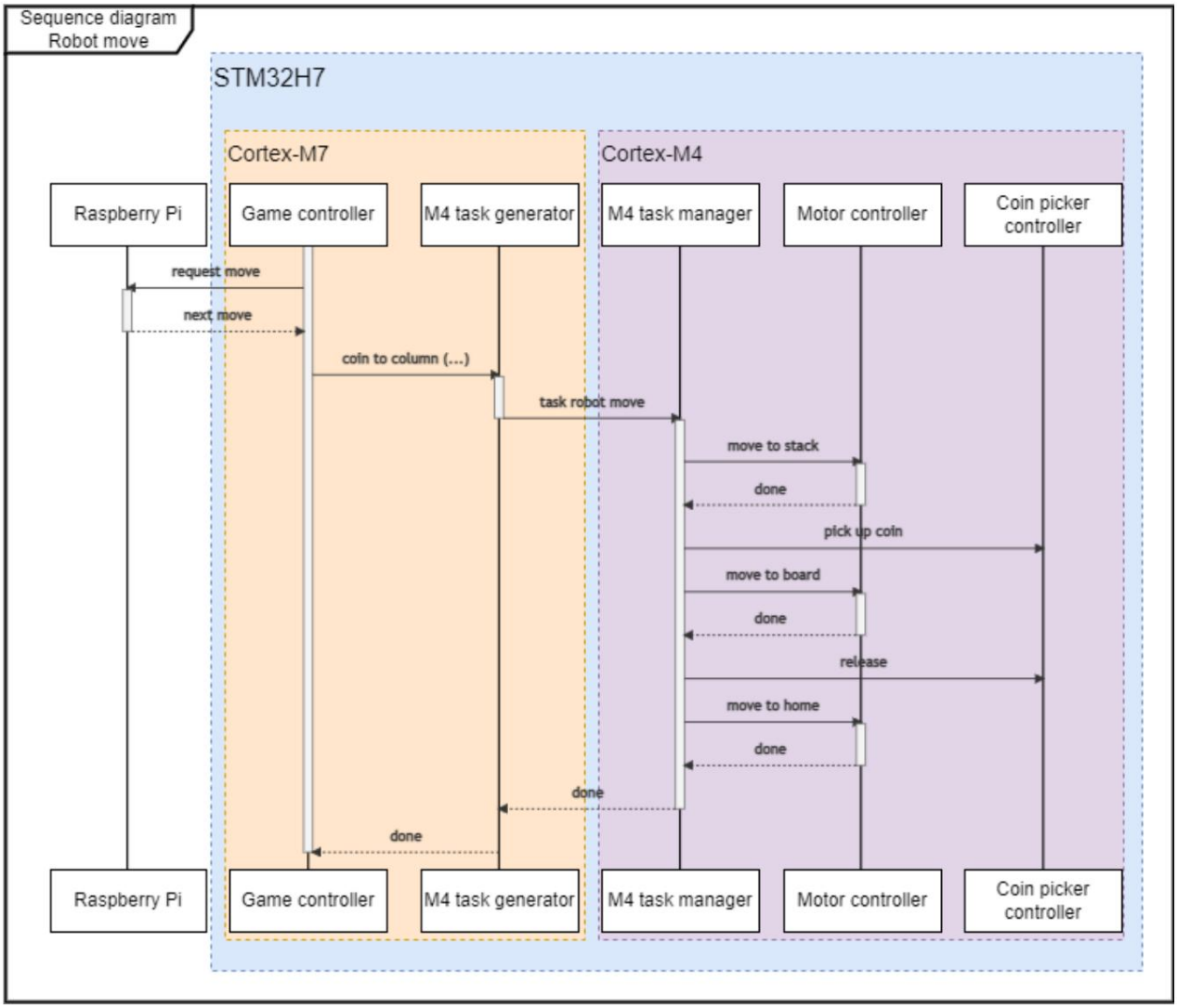


Figure 15 Sequence diagram Robot move

Rin Figure 15 starts with the Game controller requesting the next move from the Raspberry Pi. The Raspberry Pi returns the next move, after which the Game controller tells the M4 task generator what kind of assignment needs to be made. Once the job is created, the M4 task generator sends the job to the M4 task manager on the Cortex-M4. To put a chip in the board, the robot must first go to the place where the chips are stored. Once the robot has arrived at the storage, a token must be picked up. Now the robot can move to the board and then release the chip. When the move has been made, the M4 task manager informs the M4 task generator, which in turn informs the Game controller that the turn is complete. The Game controller goes to the next state.

6.3 Human move

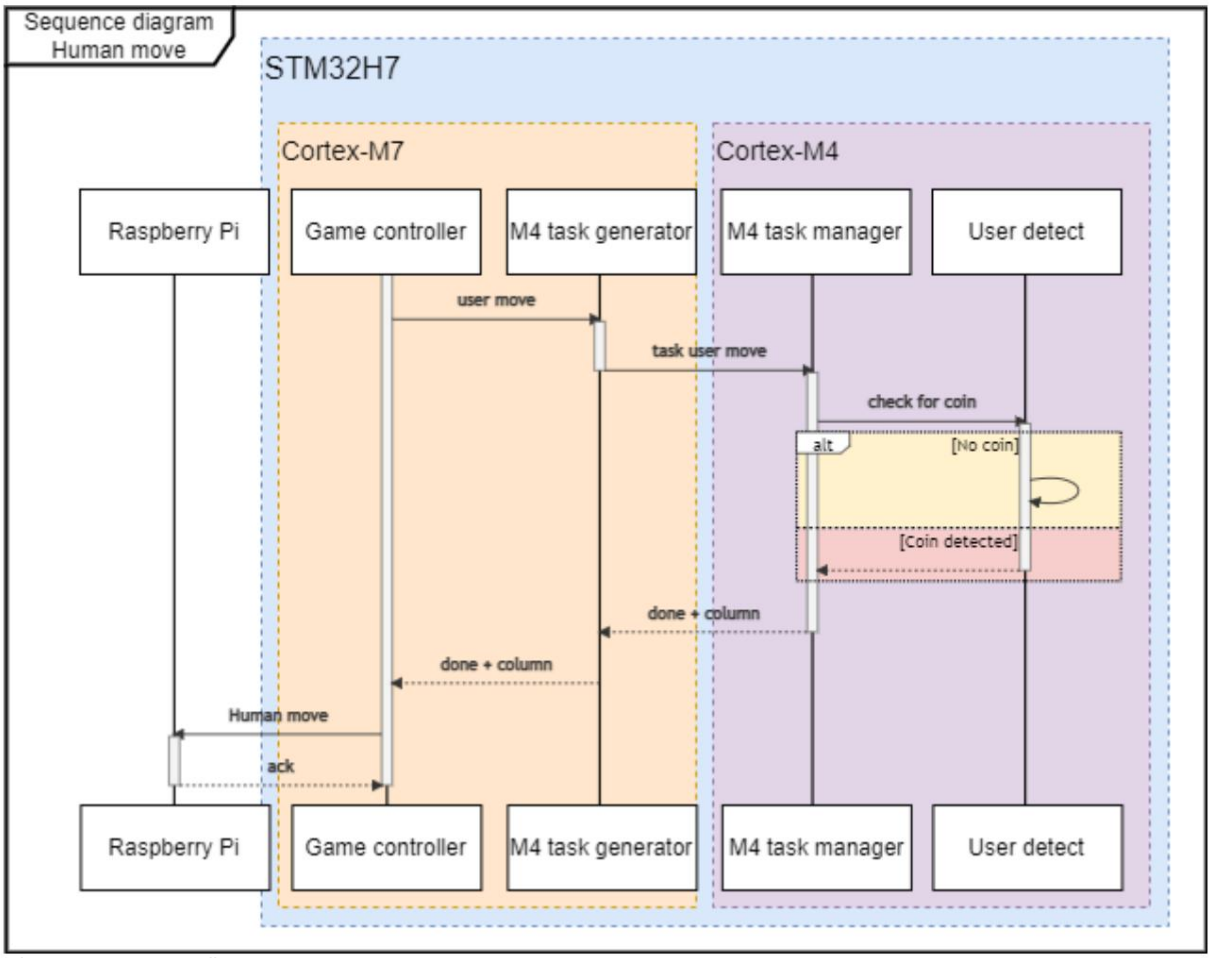


Figure 16 Sequence diagram Human move

Human Figure 16 begins by signaling the player's move to the M4 task generator. The M4 task generator generates the command and sends it to the M4 task manager on the Cortex-M4. The M4 task manager in turn instructs the User detect that a chip is expected. If no chip is played by the player, the system continues to wait for a move. As soon as a chip is played by the player, it is linked back to the M4 task manager. This informs the M4 task generator that a move has been made and in which column the chip is placed. The M4 task generator passes this on to the Game controller, after which it passes on the move to the Raspberry Pi. The Raspberry Pi acknowledges getting the move and sends back whether the game is done or can continue.

6.4 Cleanup

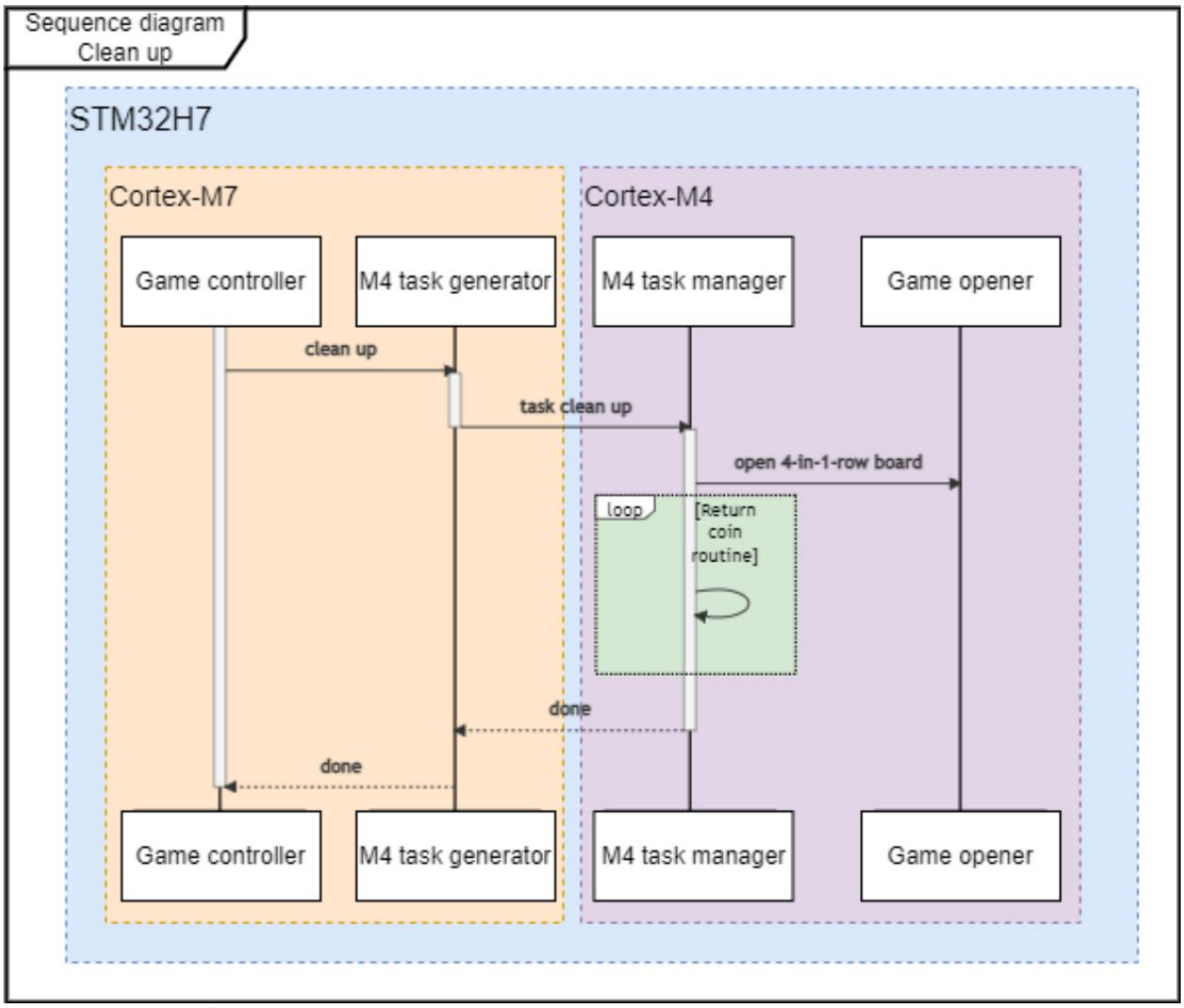


Figure 17 Sequence diagram Clean-up

Figure 17 begins with the game over notification and cleanup routine can start on the M4 task generator. The M4 task generator generates the command and sends it to the M4 task manager on the Cortex-M4. The M4 task manager opens the 4-in-1 board and starts the

is the M4 task manager tells the M4 task generator that the game has been cleaned up. The M4 task generator passes this on to the Game controller, after which the game can start again.

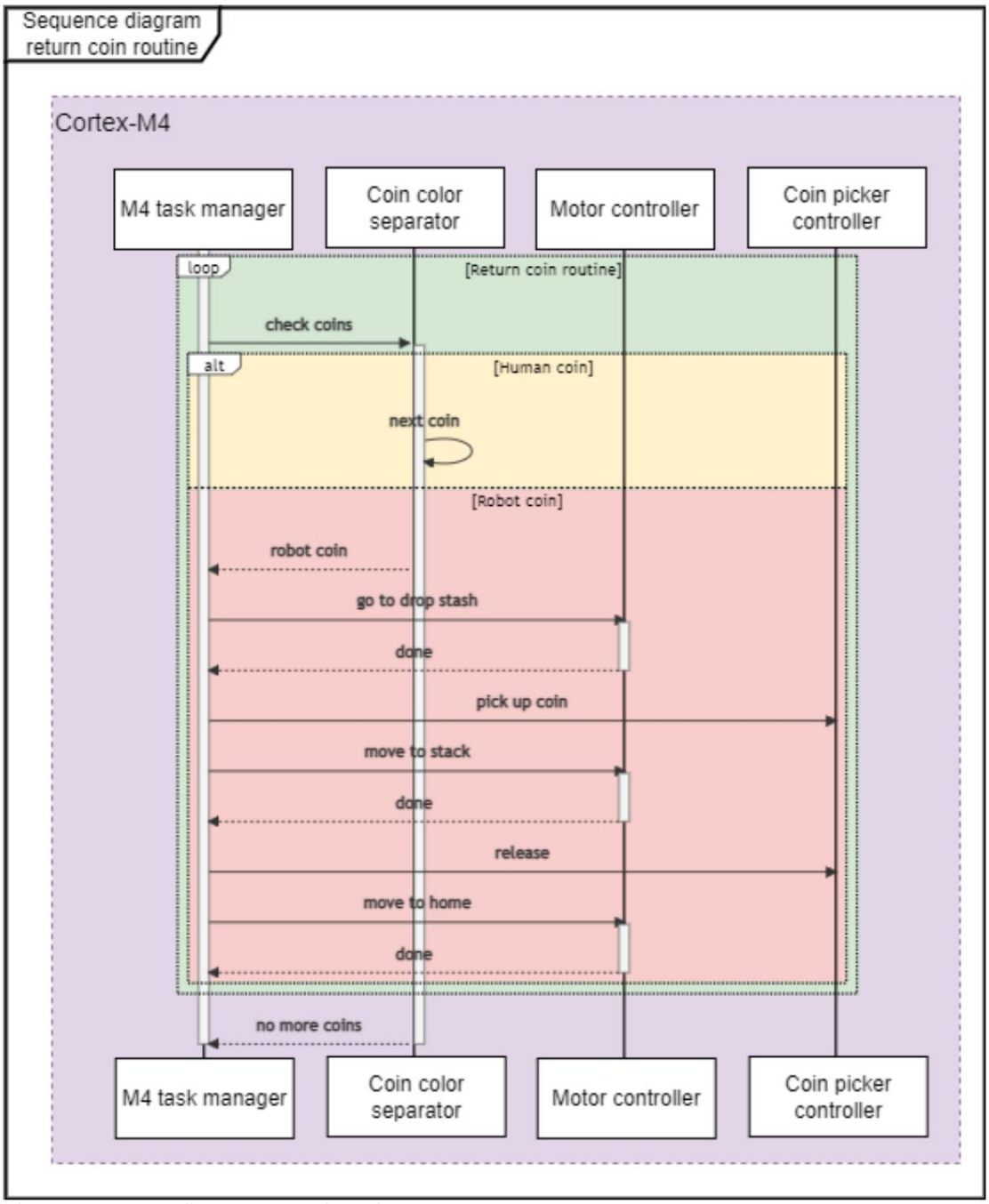


Figure 18 Sequence diagram return coin routine

in Figure 18, part of M4 task Figure 17 and describes how the manager ensures that all chips are cleaned up. First, the M4 task manager sends to the Coin color separator that chips are expected. As soon as a chip is detected, the Coin color separator will check which color this chip has. If the chip belongs to the player, it is shot by a flipper at the player's tray. If the token belongs to the robot, the M4 task manager is informed that it is a token from the robot. The M4 task manager directs the Motor controller to go to the file. Then the token is picked up and the robot moves to its own chip storage. Here the token is released again and the robot moves to its cleared.

7 Deployment View

7.1 NUCLEO-H755ZI-Q

A Nucleo-H755ZI-Q is available to implement the software architecture. This is a development board that taps into the STM32H7 chip. Furthermore, this board has all the necessary pin-outs and hardware to facilitate the implementation of the software blocks. The plan is to eventually switch to a PCB designed by ALTEN for the STM32H7 chip with all the necessary connections. However, this falls outside the scope of this project and therefore the Nucleo H755ZI-Q is used (Figure 19).

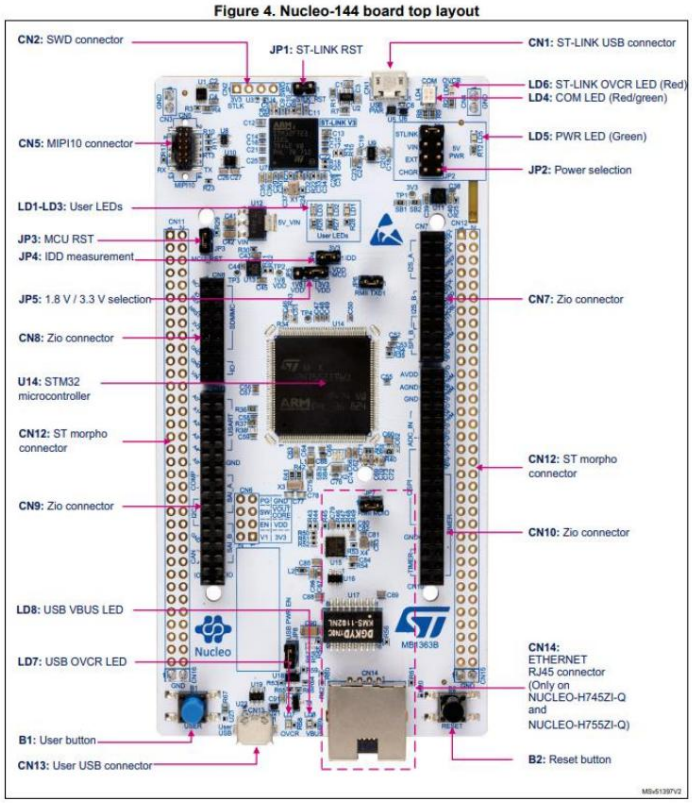


Figure 19 NUCLEO-H755ZI-Q

7.2 Pinout NUCLEO-H755ZI-Q

To realize all the functions of the 4-in-1 row, the pinout in Figure 20 has been composed. This pin out takes into account the function of each pin and what the pin should facilitate. Table 14 lists the function performed by a pin, what kind of signal is on the pin and to which pin the function is connected. As can be seen, pins have also been established for Ethernet. Ethernet is a feature that may be used in the future but is beyond the scope of this project.
But it is very important to assign the function to the pins to prevent those pins from being used for other purposes.

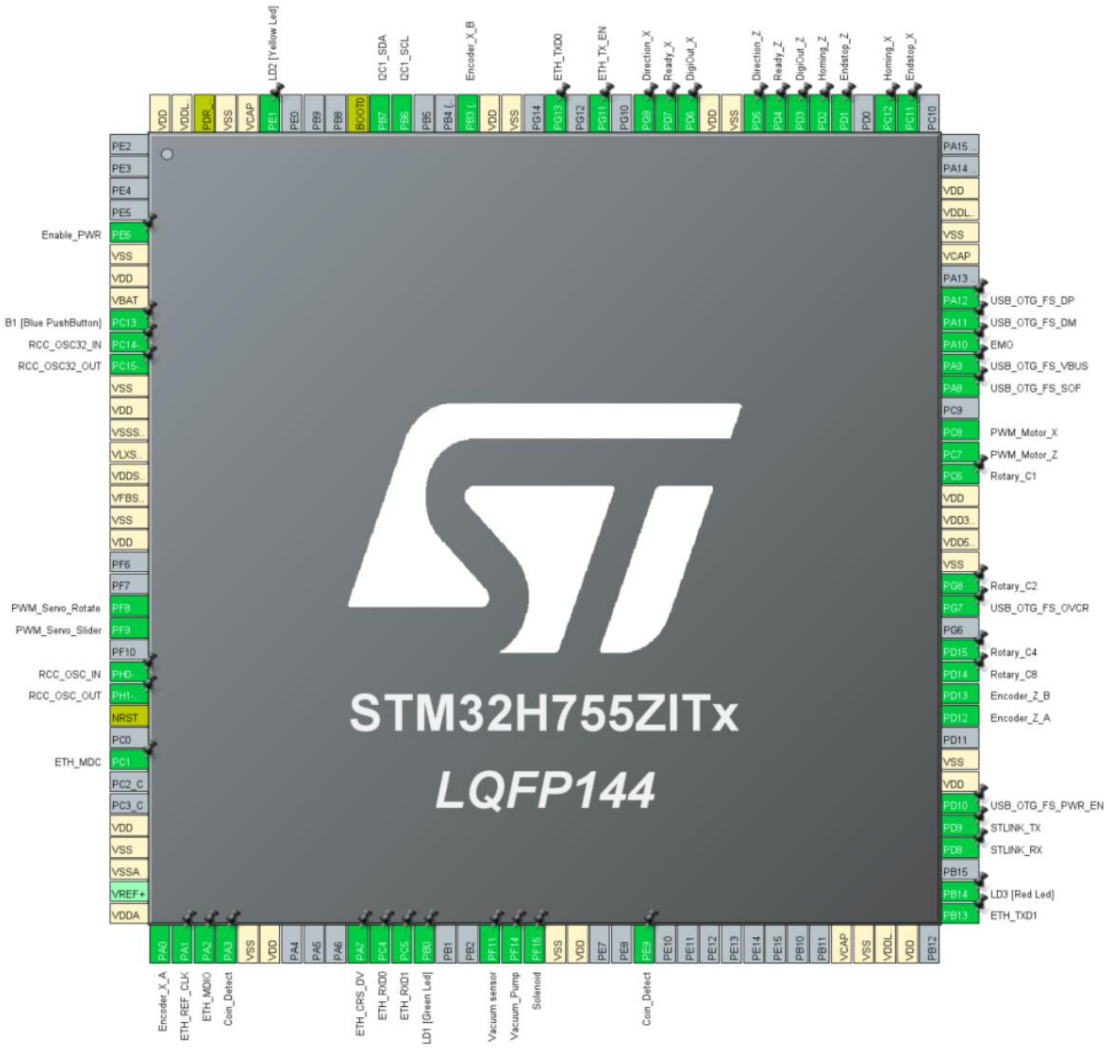


Figure 20 pinout STM32H755ZITx

Table 14 pinout table

function	STM Function	STM Pin	Connector
LD1 (Green)	GPIO	PB0	
LD2 (Yellow)	GPIO	PE1	
LD3 (Red)	GPIO	PB14	
push-button	GPIO	PC13	
Encoder X	a	PA0	
	B	PB3	
Encoder Z	a	PD12	
	B	PD13	
PWM X	PWM timer	PC8	
PWM Z	PWM timer	PC7	

Direction X	GPIO	PG9	
Direction Z	GPIO	PD5	
Ready X	GPIO	PD7	
Ready Z	GPIO	PD4	
DigiOut X	GPIO	PD6	
DigOut Z	GPIO	PD3	
PWM Servo Slider PWM Timer		PF9	
PWM Servo Rotate PWM Timer		PF8	
solenoid	GPIO	PF15	
vacuum pump	GPIO	PF14	
vacuum sensor	ADC	PF11	
Read EMO	GPIO	PA10	
Enable PWR	GPIO	PE6	
Proxint 1	GPIO interrupt	PE9	
I2C	I2C SLC	PB6	
	I2C SDA	PB7	
Coin detect Interrupt GPIO interrupt		PA3	
rotary switch	C1 - GPIO	PC6	
	C2 GPIO	PG8	
	C4 GPIO	PD15	
	C8 - GPIO	PD14	
End stops	X - GPIO	PC11	
	Z-GPIO	PD1	
Homing	X - GPIO	PC12	
	Z-GPIO	PD2	
RPI UART	Tx	PD9	
	Rx	PD8	
Ethernet	ETH_REF_CLK	PA1	
	ETH_MDIO	PA2	
	ETH_CRS_DV	PA7	
	ETH_TXD1	PB13	
	ETH_MDC	PC1	
	ETH_RXD0	PC4	
	ETH_RXD1	PC5	
	ETH_TX_EN	PG11	
	ETH_TXD0	PG13	

7.3 Hardware Layout

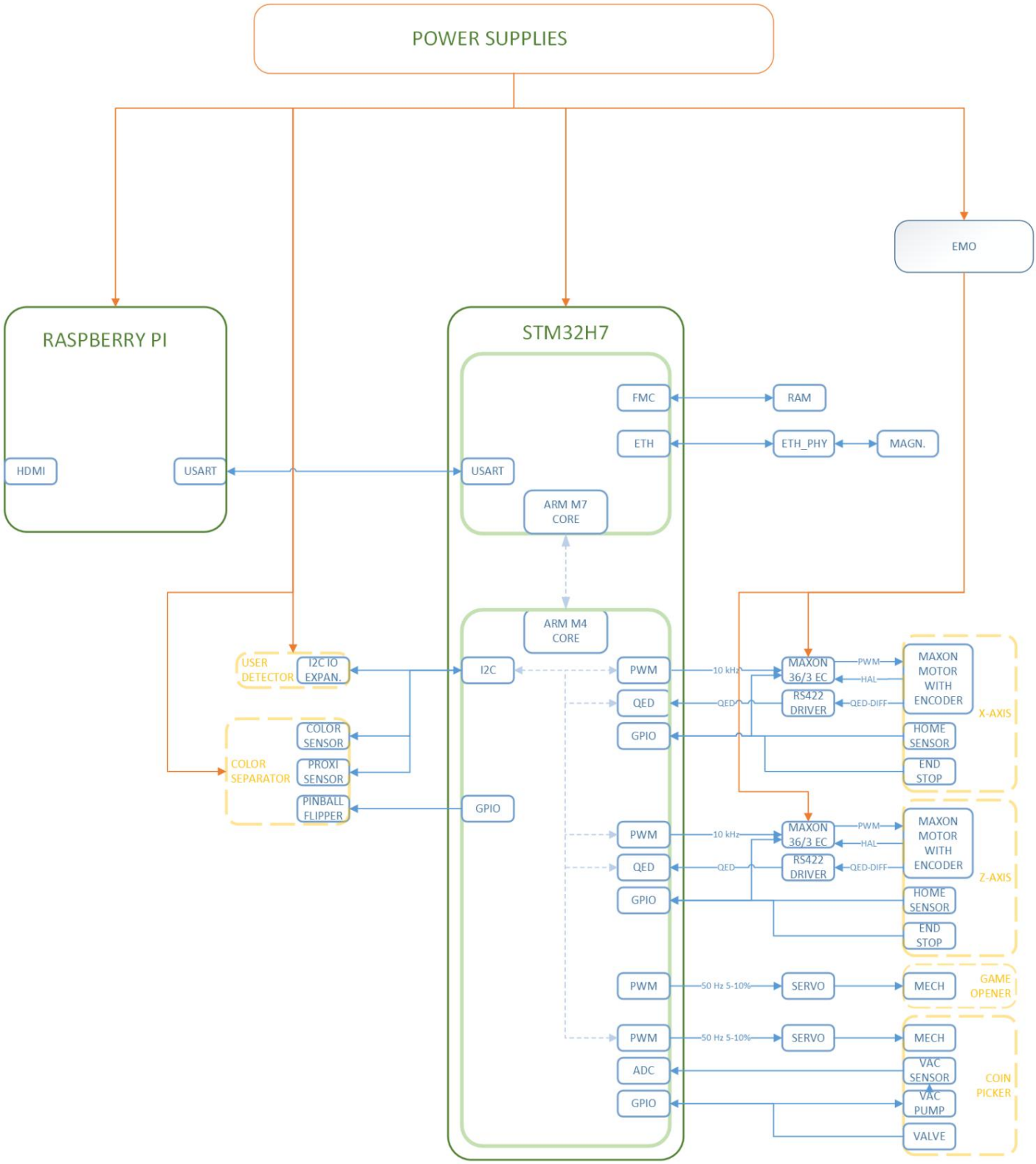


Figure 21 4-in-1-row hardware layout with STM32H7

7.4 Master-Minion relationship

Figure 22 has been drawn up to indicate how the processors relate in a Master-Minion relationship. This shows that the Cortex-M7 is the top master of the system. The game flow runs on the Cortex-M7 and therefore determines the entire game course. The Raspberry Pi and Cortex-M4 are a minion of the Cortex-M7 and perform tasks when the Cortex-M7 asks. The Cortex-M4 is also a master for the hardware. All hardware systems of the 4-in-1 row are a minion of the Cortext-M4 and perform tasks when the Cortex-M4 asks.

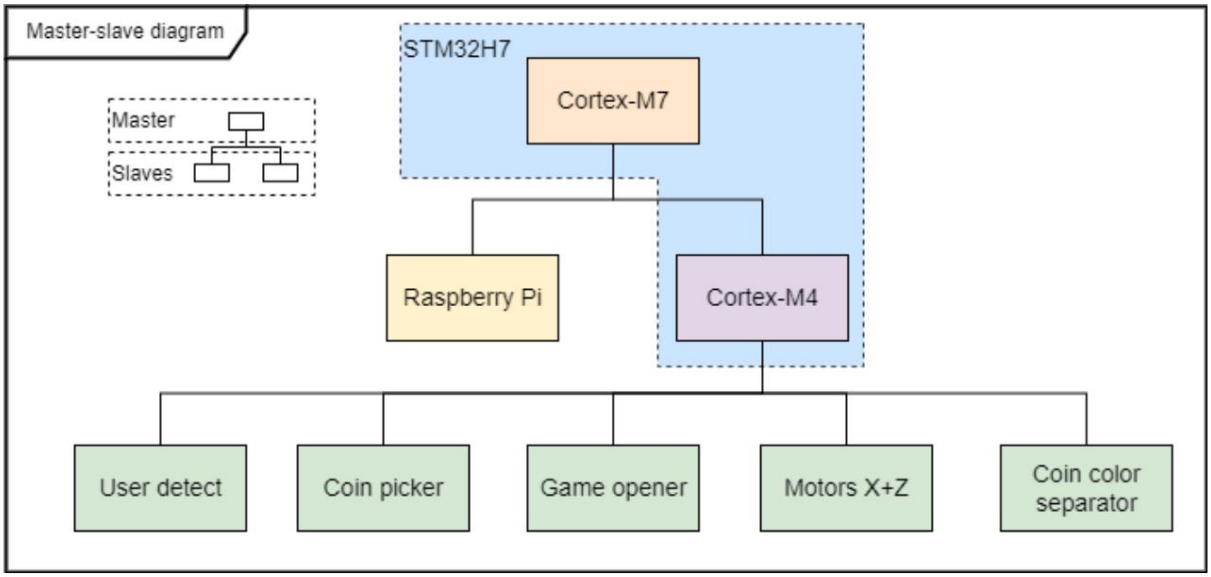


Figure 22 Master-minion relationship

8 Modular software implementation

(Chapter 5) shows which software blocks must be present to implement a fully functional 4-in-1-row robot. It is not shown here how these are related from the top level to the hardware. This is further explained in this chapter using diagrams.

8.1 Cortex-M7 core

In Figure 8 all software blocks of the Cortex-M7 core are shown. To show how the blocks relate to the hardware, the diagram in Figure 23 has been drawn up.

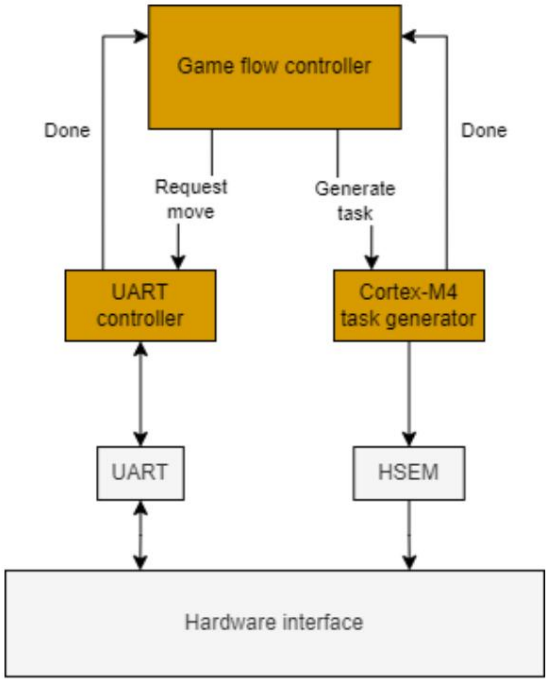


Figure 23 IBD game controller

8.2 Cortex-M4 core

In Figure 9 all software blocks of the Cortex-M4 core are shown. To show how the blocks relate to the hardware, all diagrams have been drawn up in the subchapters.

8.2.1 Engine controller

In Figure 24 shows the diagram of the motors. The blocks for Motor X are the same as for Motor Z. These blocks are shown in green.

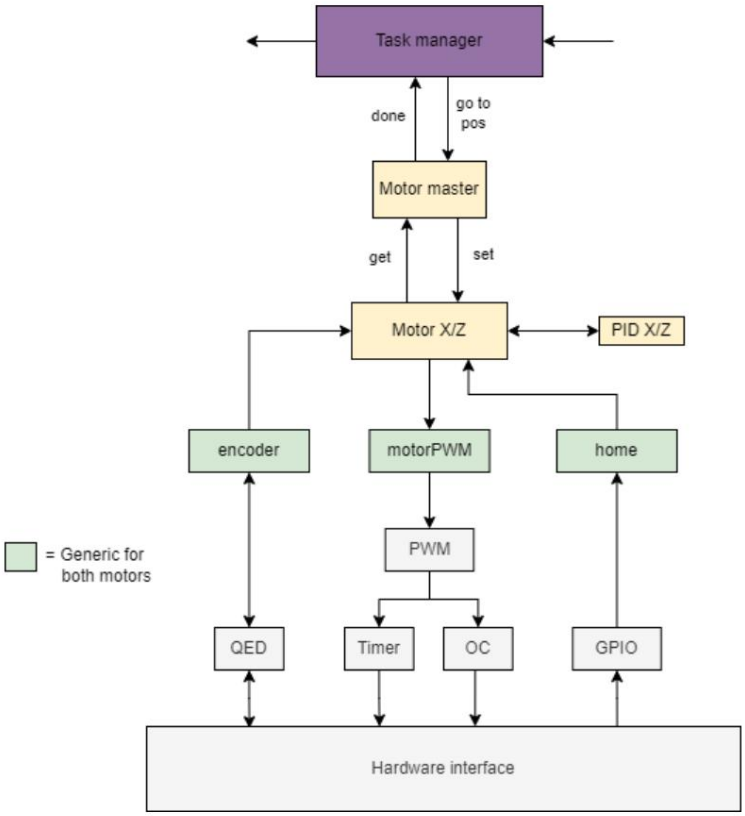


Figure 24 IBD Motor controller

8.2.2 Coin color separator

Figure 25 shows the diagram for the Coin color separator. The i2c device block is a generic block that handles all information for an i2c sensor.

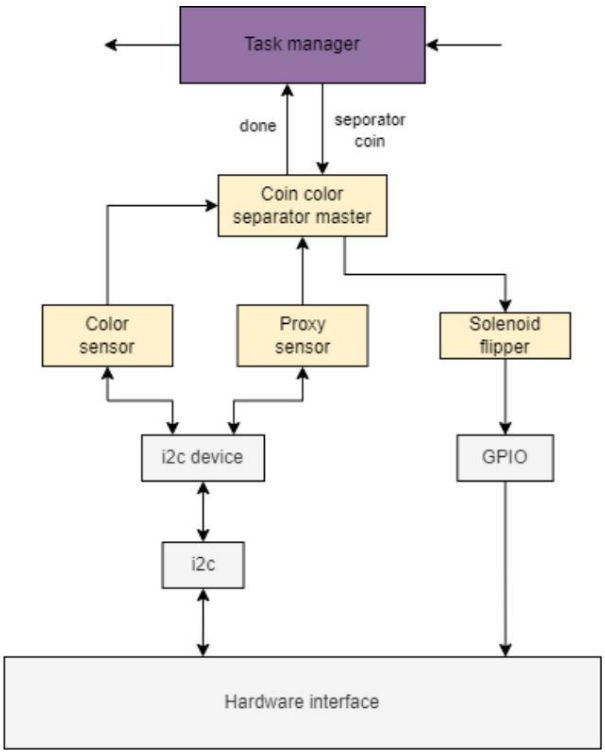


Figure 25 IBD Coin color separator

8.2.3 Coin picker

Figure 26 shows the diagram for the Coin picker.

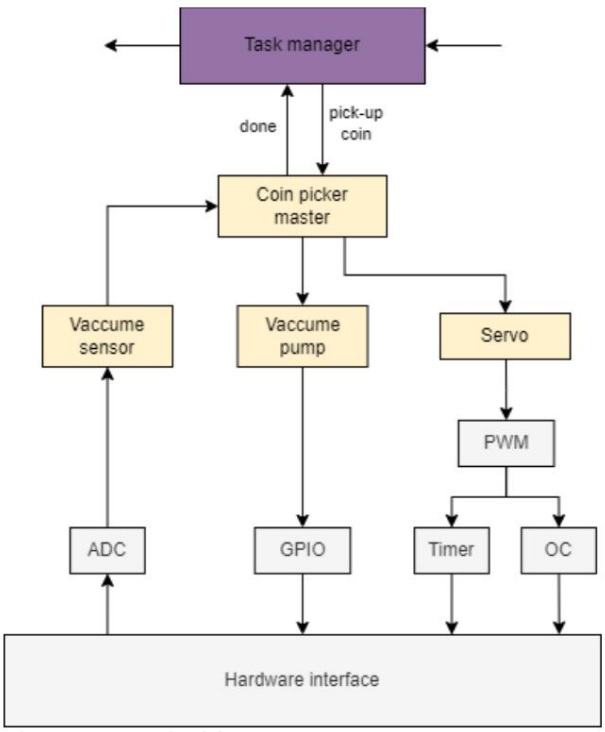


Figure 26 IBD Coin picker

8.2.4 Board opener

Figure 27 shows the diagram for the Board opener.

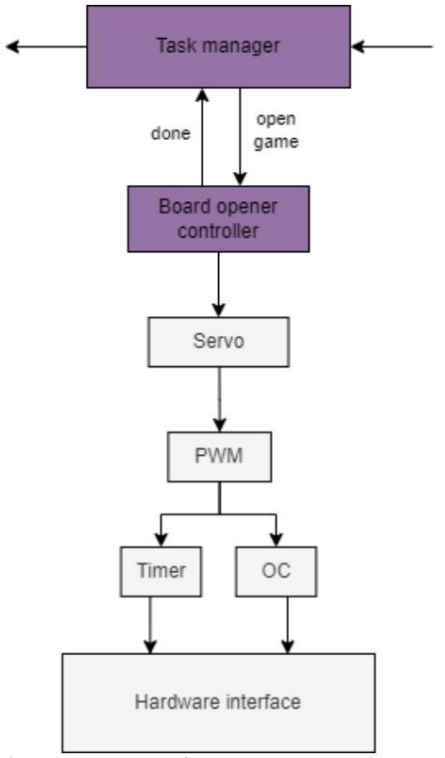


Figure 27 IBD Board opener

8.2.5 User detect

Figure 28 shows the diagram for the User detect.

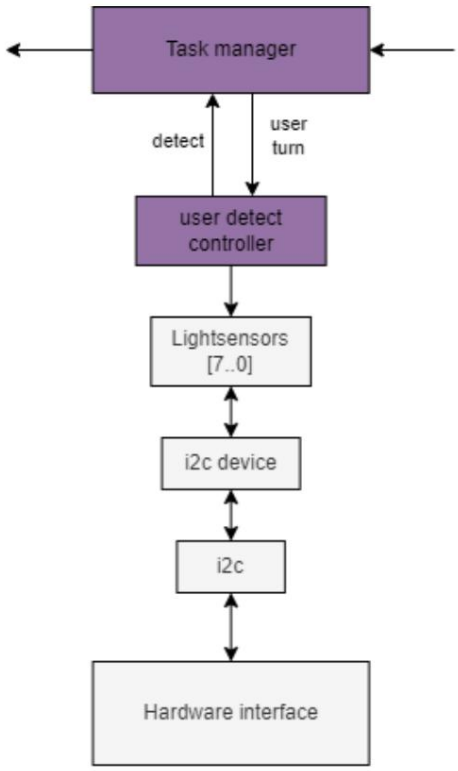


Figure 28 IBD User detect

9 Error handling According

to the architecture described in previous chapters, the system can run smoothly. Run phase anyway, one of the sensors stops responding or the motors fail when a token is brought to the board. As soon as the system gives an error message, it must be dealt with. For safety, this means turning off hardware and giving an indication to the Operator that the system has detected an error. Figure 29 shows how an error within the main states of the Game flow controller is handled.

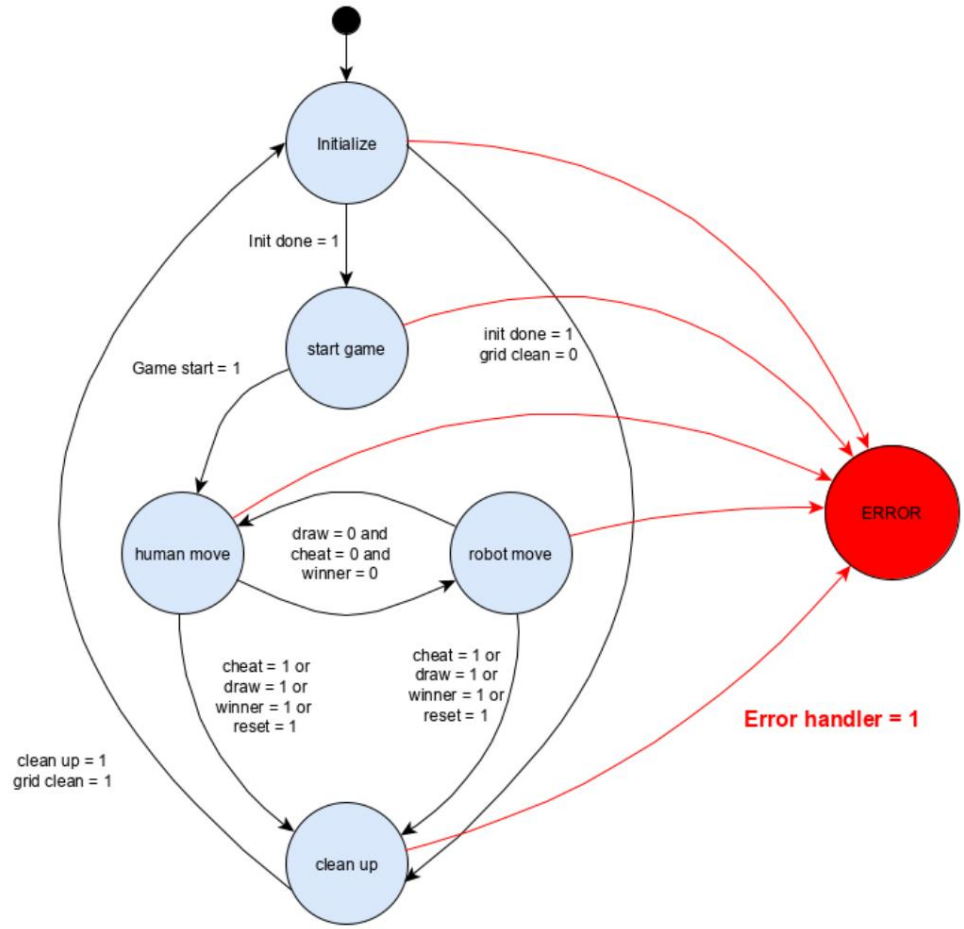
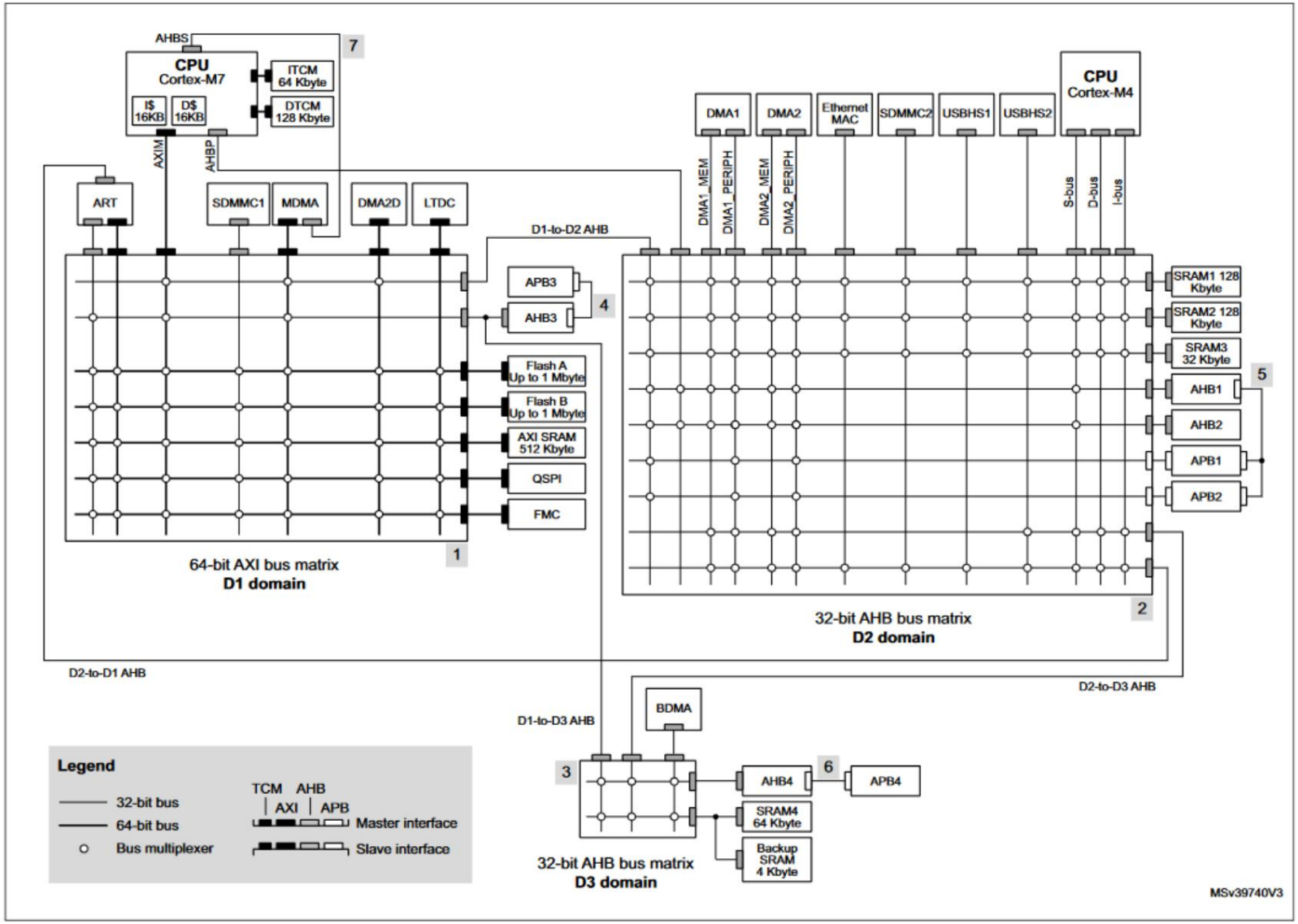


Figure 29 State machine with error handler

le motors and other hardware components plotted within the two cores of the STM32H7 and the system remains in this state until reset.

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

Round robin

The simplest implementation of the Round robin executes tasks one after the other and when they are all executed the system starts again (also called a loop). Figure 30 represents Round robin. Task 1 to 4 are executed serially. There are many examples where this method suffices. Think of a candy 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 executing it (think of the time between pressing a button on the oven and updating the screen) are suitable for a round robin implementation.

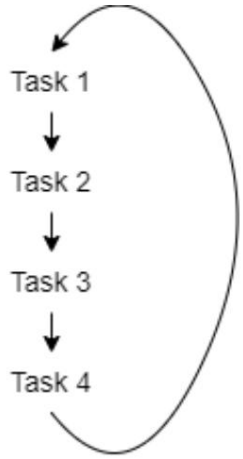


Figure 30 Round robin

In most cases, a round robin method will suffice. The big advantage of the Round robin method is that it is very simple and easy to maintain for small, compact systems with not too many tasks to perform. The method also has limitations. If a device has to perform a task faster than the system can run through the loop, the system will no longer function properly. In the worst case, the total time the system takes over one course is the time of all functions within the tasks added together. A 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 the system to run into problems, for example when the screen refreshes too slowly in an oven. The latter can be limited by having the time-bound tasks come back more often in the course.

Round robin with interrupt

method. Here important tasks within the system are called by an interrupt (often performed by an Interrupt Service Routine (ISR).

A hardware interrupt is an interruption generated by an external signal, such as a button press, timer or data on a bus. As soon as the interrupt occurs, an ISR is performed in the core, where the signal goes.
This ISR interrupts the loop and executes the specific code contained in the ISR. Then the process is picked up again in the loop [19]. Figure 31 is a schematic representation of an ISR.

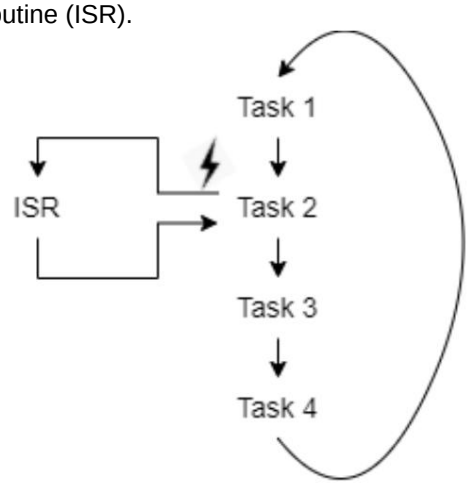


Figure 31 Round robin with interrupt

The big advantage of a round robin with interrupt is that the interrupt ensures that tasks with a high priority are executed immediately and are not dependent on the loop being completed. 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 performing a calculation, which is interrupted by an interrupt that refreshes the data of the calculation, this can lead to incorrect results (Appendix VI. False data). This should be taken into account when implementing this method.

Function Queue Scheduling

uses interrupts just like the Round robin with interrupt. Within this method, the interrupts are assigned a priority level. As soon as an interrupt calls an ISR, it is queued. This queue is passed through the course of

priority. 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 is also possible that an ISR with a low priority can never be executed because an ISR with a higher priority always intervenes.

Real Time Operating System

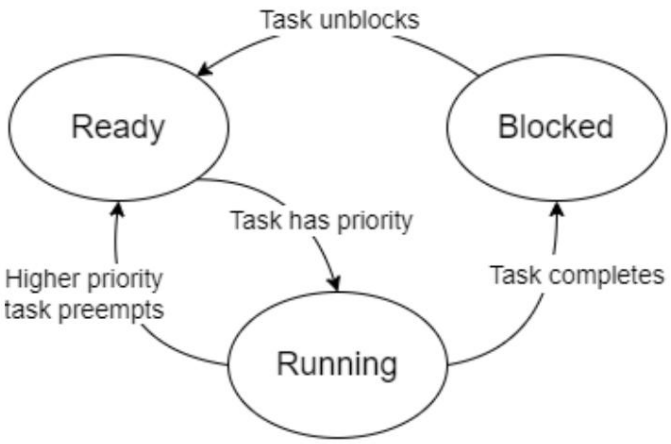
function and priority. No loop is used, making it easy to add and remove tasks. Based on the priority of the tasks, the RTOS plans which task should be executed first. A task can be in one of the following states:

1. **Running**
   The task is being executed by the processor. Only one task can be performed at a time.

2. **Ready**
   All data is present to run the task when the processor is available. Several tasks can be available and the processor determines the order in which the tasks are executed based on the priority.
3. **Blocked**
   it does not yet have all the data to be executed or it is is in an error state.

The part that checks the status of the

Scheduler 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 scheduler is shown in Figure 32. The advantage of an RTOS is that the response time is very short and that the system is flexible. The disadvantage is that it is a complicated method to apply *Figure 32 RTOS scheduler .* 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].



VI. false dates

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 running in the loop (while(1)) at any time. In this case, the interrupt requests 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 occur while the loop is calculating newValue. In this case, the data is adjusted with which the newValue is calculated. 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 that way by the microcontroller. The microcontroller

convert zeros and ones of the binary machine language into readable functions that the microcontroller performs. So Assembly language actually shows how the system acts at registry level. An instruction in the assembly language usually consists of three parts: **1. Label** The memory address where the code is located.

2. Op-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; Adds the data in registers 8 and 9 together, places it
                    result in register 7. r2, r5, r3; Bitwise
and     AND the data from registers 5 and 3, put the result in register 2.

lwz     r6, Ox4(r5); Load the data at the memory address of the sum of register 5 and 0x4 into register 6. lwzx r9, r5, r8;
                    Load the data at the memory address of the sum of registers 5 and 8 into register 9. stwx
r13, r8, r9; Store 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 basic machine activities. The following assembly instructions represent the line of C code **newValue = i2c_sensor offset;** from the first block:

```
lwz     r1, 0(r12); Put the data from i2c_sensor (0(r12)) into register 1. r2, x; Put the value of offset (x) in
li      register 2. r3, r1, r2; Separate the data in registers 1 and 2 and put it
sub
                    result in register 3.
stwx r3, 0(r11); Store the data in memory (0(r11)).
```

This shows that one line of C code consists of several lines of assembly code. This means that not only can the code break between lines of C code, but also the line of C code itself because the assembly code of that line consists of multiple 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 of critical code must be protected from being prevented. Disabling the interrupt before the critical block and enabling it after the critical block prevents false data. This is because the data is first processed before new data can be received. An interrupt has a high priority for a reason, so care must be taken when and where the interrupts are disabled.

VII. Complication dual-core communication

A problem came up while implementing the dual-core communication. During testing, it was not possible to establish communication between both cores. After debugging for a long time, reading through 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 data coherence problem arises.

Cache is memory in which data is temporarily stored for faster access to the data. Essential to using cache is that it be transparent. This means that when retrieving data it is not visible whether the data is being 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 put in the cache memory [21]. Cache has been enabled on the Cortex-M7 within this project to facilitate future upgrades that require speed and efficiency.

Figure 33 Shows schematically how the problem of data coherence occurs during the dual-core communication. First of all, the Cortex-M7 has processed certain data and it needs to be sent to the Cortex M4. The Cortex-M7 writes the data to the predetermined location in the shared memory (SRAM4). In Figure 33 that location is MemX. Because the Cortex-M7 has enabled its cache, the data will be written to the cache first. 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 has been written into the cache 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.

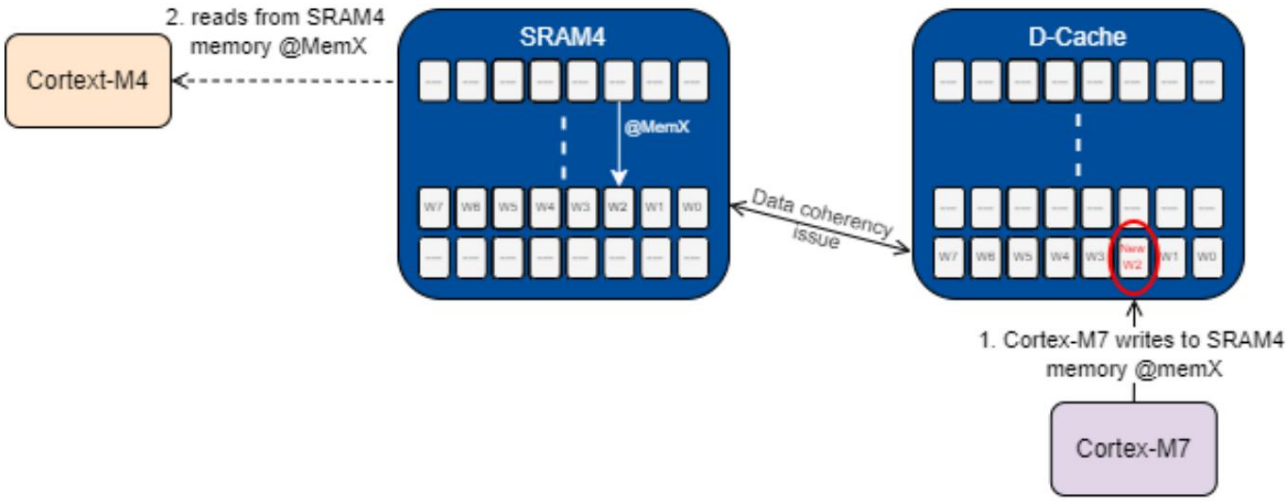


Figure 33 Data coherence problem [22]

Two solutions are conceivable for the problem of data coherence:

1.

-M7 wants to move to a location in memory to write. First, the Cortex-M7 writes to memory but then caches the data

-M7 data will read from memory. As soon as there is new data in memory and the Cortex-M7

the cache will be synchronized with the memory. From this moment on, the new data is also in the cache memory and the Cortex-M7 can access 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 a part of the memory can be adjusted. There is advance

   the Cortex-M7 can write and read directly to this part in memory.

Within this project it was decided to implement an MPU. An MPU is easy to set in advance. It also saves many lines of code that would otherwise be required for implementing

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 #define HSEM_CM7_TO_CM4_MASK      30

__HAL_HSEM_SEMID_TO_MASK(HSEM_CM7_TO_CM4)
#define HSEM_ERROR #define HSEM_ERROR_MASK                 31

__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 #define                          2
HSEM_ROBOT_MOVE_MASK
__HAL_HSEM_SEMID_TO_MASK(HSEM_ROBOT_MOVE)
#define HSEM_HUMAN_MOVE #define                          3
HSEM_HUMAN_MOVE_MASK
__HAL_HSEM_SEMID_TO_MASK(HSEM_HUMAN_MOVE)
#define HSEM_CLEAN_UP #define HSEM_CLEAN_UP_MASK         4

__HAL_HSEM_SEMID_TO_MASK(HSEM_CLEAN_UP)
#define HSEM_CHEAT #define HSEM_CHEAT_MASK                 5

__HAL_HSEM_SEMID_TO_MASK(HSEM_CHEAT)
#define HSEM_COIN_COLUMN #define                         6
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 val = 0;


        HAL_StatusTypeDef retFunc;

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