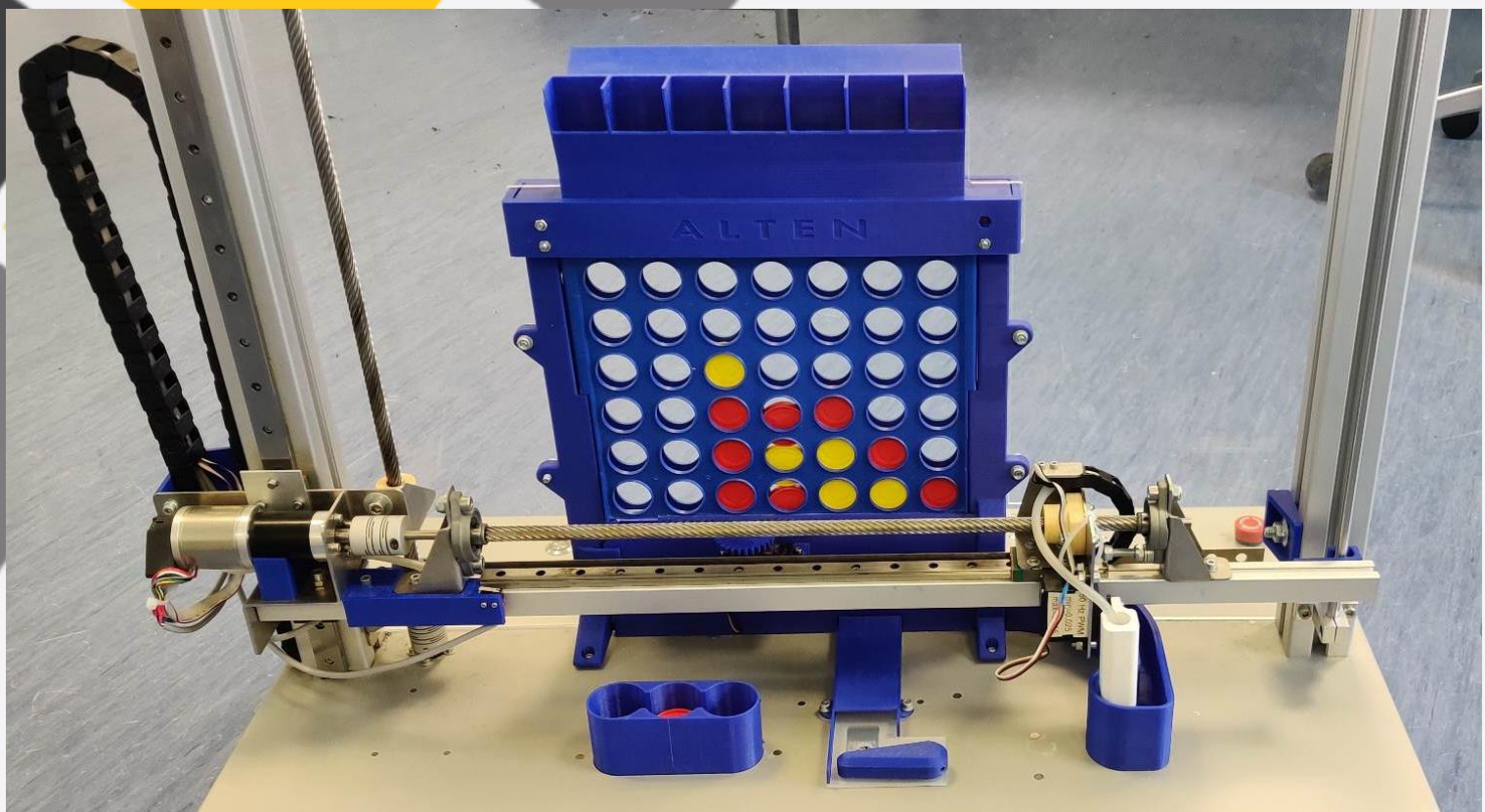
Designing an Autonomous Robot-Player for Connect-4

Author: Boris Ivanov

Date: 06/06/2023



Document data:

Author: Boris Ivanov
Student number: 2969300
E-mail: 357544@student.fontys.nl
Date: 06 June 2023
Place: Eindhoven, the Netherlands

Company Data:

Name: ALTEN Nederland
Address: Hurksestraat 45, 5652 AH Eindhoven
Company supervisor: Michael van der Velden
Telephone: 0402563080
E-mail: Michael.van.der.velden@alten.nl

University Data:

Name: Fontys University of Applied Science
Address: Nexus Building (ER, De Rondom 1, 5612 AP Eindhoven)
School supervisor: Michal Mikołajczyk
Telephone: 0618592672
E-mail: m.mikolajczyk@fontys.nl

Approved and signed by the company supervisor:

Date: 06-06-2023

Signature:

Foreword

This is an internship report on ‘Designing an Autonomous Robot-Player for Connect-4’. This project has been realized at ALTEN by Boris Ivanov on behalf of the educational program Electrical & Electronic Engineering at Fontys University of Applied Sciences in Eindhoven. The project and this report were realized from February 2023 through June 2023.

During November, I was contacted by a business manager, Gosia Szabla, and through the course of several interviews an assignment, from my current business manager, Gijs Haans, was shown to me and it was perfectly in line with what I wanted to do. I am grateful for the presented opportunity.

During my internship, I was guided by my technical supervisor Michael van der Velden, who has expert knowledge in software design and embedded system. I would also like to thank Gijs Haans, and my Fontys supervisor Michal Mikołajczyk, for without their guidance at crucial times in the internship, it wouldn’t have been such a resounding success.

I would like to express my sincere appreciation to my fellow interns and the consultants at ALTEN for devoting their time and collaborating with me, answering my inquiries, and sharing their knowledge. Their invaluable contributions have significantly enriched the depth and quality of my assignment.

And that leaves us with final words of thanks to you, the reader.

I wish you a pleasant reading experience!

Boris Ivanov

06/06/2023

Table of Contents

Summary

ALTEN, a global technology consulting and engineering company, specializes in providing projects specifically designed for their client's needs. They also have in-house projects used to train their consultants and interns on new skills. The products of which, are usually used for demonstration purposes at trade fairs, university open days and such. For this purpose, the Connect-4 robot was developed.

The initial concept of the robot came to be somewhere in 2019. A lot of different engineers have worked on it, each with their own style of writing code. As a result, the architecture became unclear and inconsistent. ALTEN knew that a redesign would be imminent, however, they decided that this would be a great opportunity to change the micro-controller of the system to a more powerful one so it can sustain even more tasks in the future.

In the projects, after ALTEN decided to upgrade the robot, the architecture was built up, and the hardware was designed. Here comes the main topic of this assignment. To create the embedded system that would run the robot as it had before, but based on the new software architecture, while using as much as possible from the older software that was previously developed. All the while researching methods to validate the system and finding possible fail points. In short, to make the system operational, because since the redesign it hadn't functioned properly.

This new phase of development was started based on the V-model and was hence continued on it. Since most of the verification portion of the task was complete, coding and validation were the main focuses of this assignment according to the V-model. That is not to say that there was no designing left to be done. Plenty in the system had to be restructured and redesigned to fit the new architecture, as well as completely redesigning the main gameplay logic of the system. The communication protocols had to be rewritten and adapted, the sensors of the system had to be set up, and to be later integrated into the higher-level modules. The validation phase included research into software testing and what that should entail, followed by shaping that knowledge into a test report that could be used for the Connect-4 machine.

After this assignment, the machine functions at a desirable level for basic operation, but with room for a lot of improvement and new opportunities. Like the addition of a screen and the usage of ethernet to communicate with the wide internet.

List of abbreviations

Acronym	Description
BSP	Board Support Package
CM4	Cortex-M4
CM7	Cortex-M7
CubeIDE	Integrated Development Environment for STM32
DMA	Direct Memory Access
EXTI	External Interrupt/Event Controller
FSM	Finite State Machine
HAL	Hardware Abstraction Layer
HMI	Human-Machine Interface
HSEM	Hardware Semaphore
I2C	Inter-Integrated Circuit (pronounced as “eye-squared-C”)
IDE	Integrated Development Environment
IR	Infrared
IT	Information Technology
MCU	Microcontroller Unit

List of figures

List of tables

I. Introduction

The Connect-4 game is a two-player connection game, and as the name suggests, four pieces in a row have to be connected in any direction, for any one player to win. ALTEN thought to robotize one of the players and power it with artificial intelligence so that it can determine the next move of the game. This move, of course, would have to be facilitated by some kind of mechanical and electrical design so that the robot can also play the moves.

The robot's initial concept came into being around 2019, as a way for ALTEN's idle consultants to have a project to work on and develop their skills further. Since the prototype, the system has undergone many changes from a lot of engineers. The latest adaptation is the addition of a dual-core microcontroller, that can facilitate the full needs of the system and upgrades for the future.

The previous iteration of the system was on a single-core microcontroller, so a redesign of the software architecture was imminent. This graduation report is an in-depth exploration of the implementation of the embedded system of the Connect-4 robot. By an embedded system one should understand that it is essentially, the connecting layer between the electrical design and the underlying building parts of the robot, like its actuators and sensors. Apart from the software, part of the assignment is to research methods to validate the system and find potential issues that the robot might have.

In this report, you will learn about the company in Chapter **II**, about the project and the assignment in Chapter **III**. Following that is the research completed for this internship and the specifications discussed, in **IV** and **V** respectively. The system design and its realization are described in Chapter **VI** and Chapter **VII**. The completion of the initial requirements is explained in Chapter **VIII**. Chapter **IX** is the conclusion part of the assignment, with recommendations in Chapter **X** being the last chapter of this report.

II. About the Company

ALTEN is a global technology consulting and engineering firm. They provide research projects for technical and information systems divisions in the industrial, telecommunications, and service sectors. Their focus is on the conception and research for the technical divisions. Additionally, ALTEN provides networks and telecom architectures, as well as the development of IT systems for the information departments [1]. As far as industries that rely on ALTEN for their business include, but are not limited to, telecommunications, computer systems, networking, multimedia, energy & life sciences, finance, defence, aviation, and information systems [2].

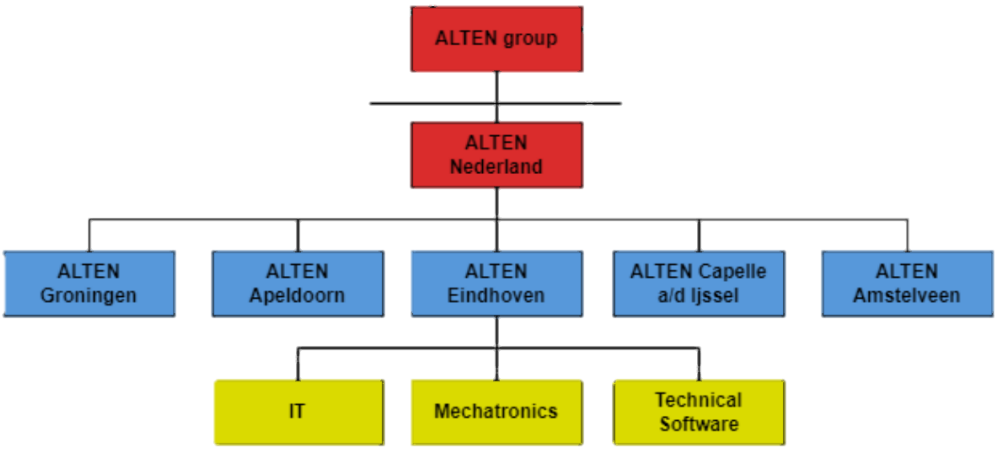


FIGURE 1: ORGANOGRAM

1. Company background information

Established in France in 1988, ALTEN is a global engineering and technology consulting firm with locations in 30 nations. ALTEN had 54,100 employees and earned 3.78 billion euros in revenue in 2022. 45% of the group's business is conducted in the French market [1].
Within the Netherlands, their expertise is in the following categories: ALTEN IT, Technical Software and Mechatronics, with the “Connect-4” project falling within the Mechatronics department.
Technical software focuses on embedded systems, simulation & modelling, monitoring & control, and business critical systems. This includes anything from banking systems to traffic light control [3].
ALTEN provides end-to-end software engineering solutions, including software design, development, testing, integration, and maintenance, to its clients across industries.
Mechatronics supports its clients by developing and improving its products with the latest improvements in technology. ALTEN's mechatronics services include designing and prototyping complex systems, simulation and modelling, control systems development, system integration, and testing and validation. The company has a team of experienced engineers who work closely with clients to understand their requirements and develop custom solutions that meet their needs [4].
This project is part of ALTEN’s in-house projects, which are often used to develop new skills for consultants or the ones of interns. Since ALTEN wants to demonstrate their competence in the field of motion systems it wanted to create a demonstrator around this. The Connect-4 (Four Up, 4-in-a-row) robot was developed for demos at trade fairs and open days at universities. The robot game is meant to demonstrate the knowledge of the consultants at ALTEN, and it is therefore developed with industrial components.

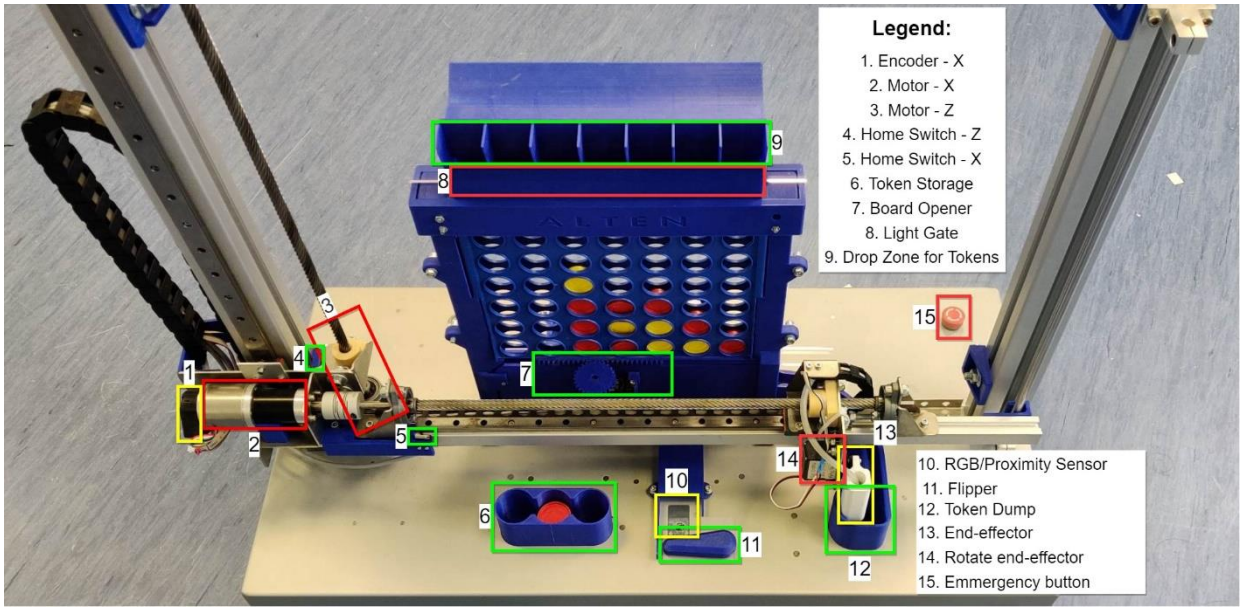
III. Project description and assignment

In this chapter, insight is provided in the form of a comprehensive overview of the project's background, problem description, assignment, project scope, and boundary conditions. The chapter delves into the project's approach, including the development phases and the application of the V-model verification method. Additionally, it discusses any unforeseen issues that arose during the project, offering an understanding into the challenges faced and their impact on the overall project execution.

1. Project background information

My graduation internship for Fontys Hogescholen will be conducted at ALTEN, with my task being to realize as many as possible software blocks and verify the whole system to the best of my abilities. This all would be done to a Connect-4 robot player, shown in Figure 2 and Figure 3, which has had its software architecture previously designed but not verified. There are some existing demonstration codes that showcase some functions of the robot. They are to be discussed in chapter 2. Investigating the microcontroller and the existing software.
The game itself is fairly simple to play. There is a seven-by-six rack board, with slots at each side for the two players to enter their tokens. A red one for the robot player and a yellow one for the human player. The first player to connect four tokens in a row in any direction wins. **Figure 2**: The Connect-4 Robot and **Figure** 3 explain what each part of the robot is and where the operator/developer might find them.

FIGURE 2: THE CONNECT-4 ROBOT



The whole process, of playing the game, should be completely autonomous. After the player token has been placed in the idle robot, it can decide its next move based on a difficulty setting. To be able to play the game, the 4-in-a-row robot is equipped with numerous parts that help it achieve its task. The big ones are the two motors for movement in the X and Z direction, together with their encoders and

home/end switches. Additionally, it has two servos, one to rotate the end-effector and another to open the board for resetting the game state. Also, the robot has an RGB (colour) sensor and a flipper to be able to sort and distribute the tokens to the correct sides. The robot's end-effector is equipped with a vacuum pump, vacuum sensor, and valve to be able to pick up tokens. Finally, the machine can detect when and where a token is dropped, through a series of Infra-Red (IR) sensors on the entrance of the board.

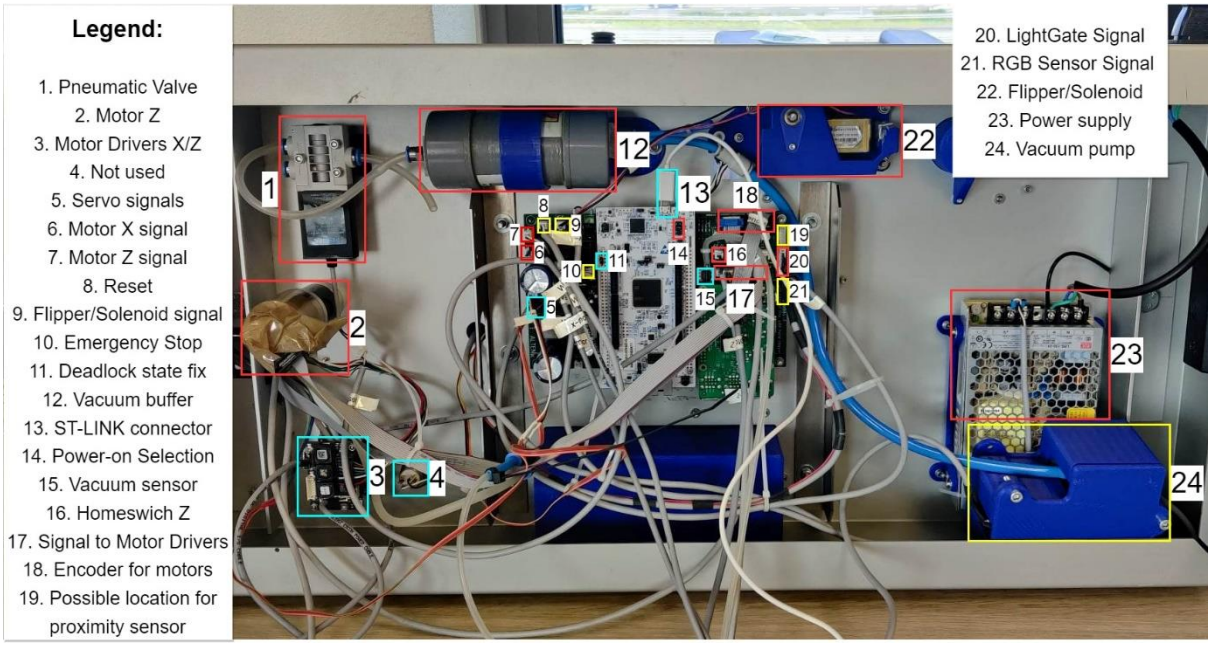


FIGURE 3: UNDERSIDE OF THE CONNECT-4 ROBOT

The project has existed for several years, and several major changes have occurred during its existence. The one that concerns the current state, is the change of micro-controller used in the system. Before, the system used a single-core processor, but with constant improvements in functionality and new additions, the system started to become slower and unresponsive. Therefore, it was decided that a new processor will be put into the system. The dual-core STM32H755 is more powerful than its predecessor and fits with the newer requirements. The initial idea was that one core would be responsible for the real-time processing, while the other core would be the "primary" core and it will delegate tasks and take care of the higher-level logic like the game decisions, displaying results and more.
Another major change is the new PCB made to accommodate the new controller, together with rewiring the system, and another set of demo software projects on specific sections of the system.

2. Problem description

With the newly added dual-core processor the system had to undergo a major restructure of its software architecture and its PCB design. These two tasks were undertaken by previous interns. However, the software architecture wasn't realized or verified due to time constraints. Several "demos" were made to showcase some parts of the architecture working together. However, neither demonstration code has been extensively verified or documented. More information about the demonstration projects will be included in Chapter IV.
The above-mentioned software projects have different functionality. One is the initialization-homing

procedure, another one is low-level code about different peripherals on the STM32, and finally the communication and rudimentary data exchange between the two cores. Additionally, a legacy code of the old system exists.
None of the projects adhere to the newly created software architecture, therefore, all the software so far has to be reviewed and assessed if it is suitable for the new system.

I would like to mention that during the internship, there were some unforeseen challenges related to the latest software project. Its delivery was delayed since the code wasn't uploaded on the intranet of the company. Additionally, after analysing the code, it was determined that it didn't function as described. This led to focusing longer on documenting and optimizing the available software and devoting more time to develop a thorough test plan. Additionally, during normal operation and debugging of the robot, some elements from the control signal circuitry for the Vacuum Pump and the Flipper/Solenoid burned, because the MOSFETs in the circuit were placed in the wrong direction. However, this occurred at a later stage of the internship, and the software for the module existed and was tested before.

3. Assignment

The assignment is to realize and verify the existing software architecture of the Connect-4 robot player, based on both the older system, and the newer Cortex-M7 core (as discussed in IV. Research). The primary objective is to validate the functionality of the designed architecture by implementing it and thus, bringing the robot to an operational level. This entails writing additional code to support the newly designed software modules and designing high-level logic for different system sub-modules. Further, this involves creating libraries for sensors and communication, for the operations of different peripherals, implementing low-level data manipulation to program different devices, and investigating testing techniques which validate the system.

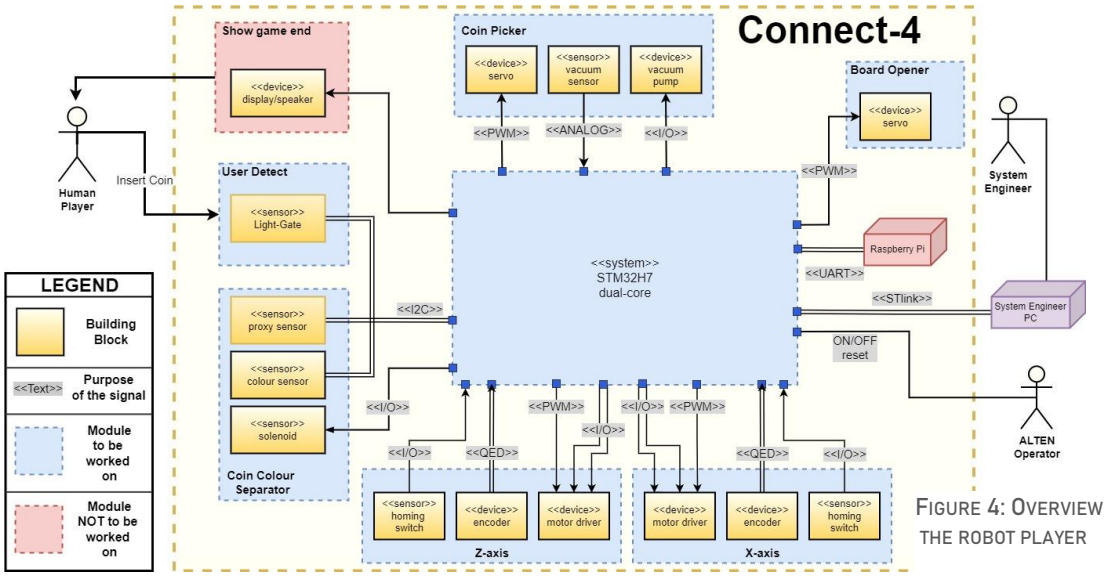


FIGURE 4: OVERVIEW OF THE ROBOT PLAYER

What can be seen in Figure 4 is the overview of the system, with the different controlling methods required by each peripheral. Blue signifies that the module needs work to be brought to a functional level. Red means that this is not implemented on the system or will not be worked upon during this project.

The objects in the blue-dashed blocks are the different sub-modules of the system. Each inner block of the sub-modules describes the hardware used to achieve the task, coloured in darker yellow. For example, the servo is a device that facilitates the rotation of the end-effector, and is part of the coin picker module, the signal that controls the servo is called PWM and it comes from the MCU.

The ultimate goal is to bring the Connect-4 robot player to an operational level by integrating and validating the various elements, paving the way for potential advancements and a seamless human-machine interface.

4. Project scope

The project is concerned with the re-evaluation (and if needed redesign) and implementation of the previously designed software architecture. The dual-core communication is worked out, but the rest of the modules have to be implemented. The programming language will be C.

TABLE 1: PROJECT BOUNDARIES

Project boundaries	**Within Scope ?**
Implement software modules *[1]*	Yes
Redesign software modules *[1]*	Yes
Research ethernet communication	Yes
Implementing ethernet communication	No
Redesign hardware/mechanics	No
Changes to the gameplay	No

[1] -- For further clarification look at Table 5: Modules origins and their initial conditions

5. Boundary condition

Boundary conditions are essential to ensure that a project is completed within the specified limits and to prevent any unwanted consequences. In the context of the Connect-4 robot player project, several boundary conditions should be considered. These include hardware limitations, time constraints, and more.
The project must be completed within the allotted timeframe, and deadlines for each stage of the project must be established to ensure timely completion. The nature of merging different software projects at different points of completion is usually time-consuming. The hardware limitations of the robot player must be considered during the design and implementation of the software. As mentioned, there are some faulty components, like some transistors, and the vacuum system, and not all of the systems intended for the robot are implemented, like end-switches for example.

6. Project approach:

The project will follow the normal V-model development procedure. However, since the project has been under development for quite some time, a big part of the verification phase is complete. The system and the architecture of the said system have been designed, together with parts of the different lower-levelled modules and their software implementations.
A part of this project will be the validation of the already made design choices and system/sub-systems (refer to VI) , through different means of testing (unit, module, integration) and a varied assortment of techniques (black-box, white-box, happy-path, worst-case) and through the designing of newer modules that are to be integrated into the system.
Additionally, to ensure the continuity of the projects and their future development and support, it will be necessary to fully document the code used from previous projects (where the code itself is completely undocumented), to make sure that future developers are not hindered. And as such, big portions of the work on that will be included in the appendix with the code itself.

The V-model is a development model that emphasizes the importance of testing and verification throughout the development process. It is suitable for the Connect-4 project because it involves a complex system with multiple components that must be integrated and tested thoroughly.

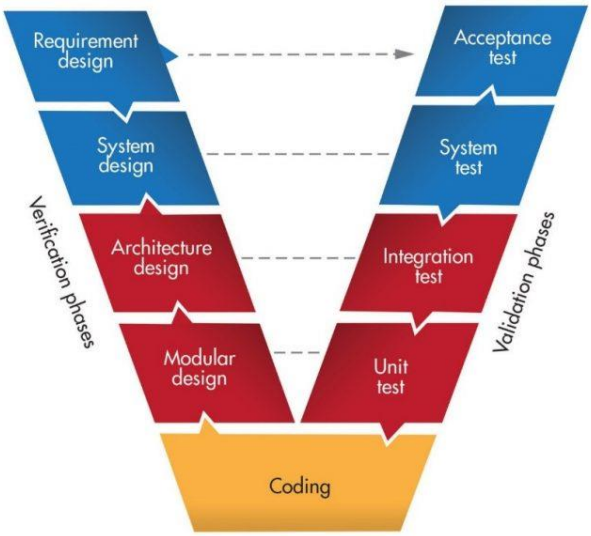


FIGURE 5: THE V-MODEL

IV. Research

The research needed for the completion of this project is varied and layered. Several topics needed investigating, to ensure a thorough evaluation of the robot and at the end, to make it operational. The topics are as follows: baseline research of the STM32 controllers and how they operate, familiarization with the existing software project and new architecture, and how to test software on embedded systems. The following pages briefly introduce the system architecture that was designed when the project was handed over, then followed by the baseline research of the STM32H microcontroller and later the projects and the discoveries about them. Followed by the necessary knowledge to improve the main gameplay logic and the validation of systems research.

1. A look at the Connect-4 Robot Player through its software architecture

The previous designer chose to describe the system on several levels. Each of them obstructs more and more the low-level controls of the peripherals and the registers of the microcontroller until level 1 is reached, which would be considered the highest level of logic in the system. In total there are 3 layers to the software architecture, each of them describing the different modules needed to make the system functional. Level 1 has the highest abstraction, and level 3 has the lowest. By designing and implementing from the lowest level, a clear path to completion is presented. Furthermore, by building up the lower-levelled blocks and testing them, the stability of the system can be verified better, and debugging can be done more easily when building up the more complex blocks.

The first level of the architecture describes that the core Cortex-M7 (referred to as CM7) will take care of the game-handling logic, like the next-move decision, delegating tasks to the other core, and the bulk of the additions for the future will be done on this core. It will be the primary core of the system, while Cortex-M4 (referred to as CM4) will be the secondary core of the system. It will take care of the real-time processing and it will act upon tasks given from Cortex-M7. The core will drive the motors, separate, and pick the tokens and more. All of these functions are to be expanded upon by the following levels of the architecture and through the addition of more software blocks.

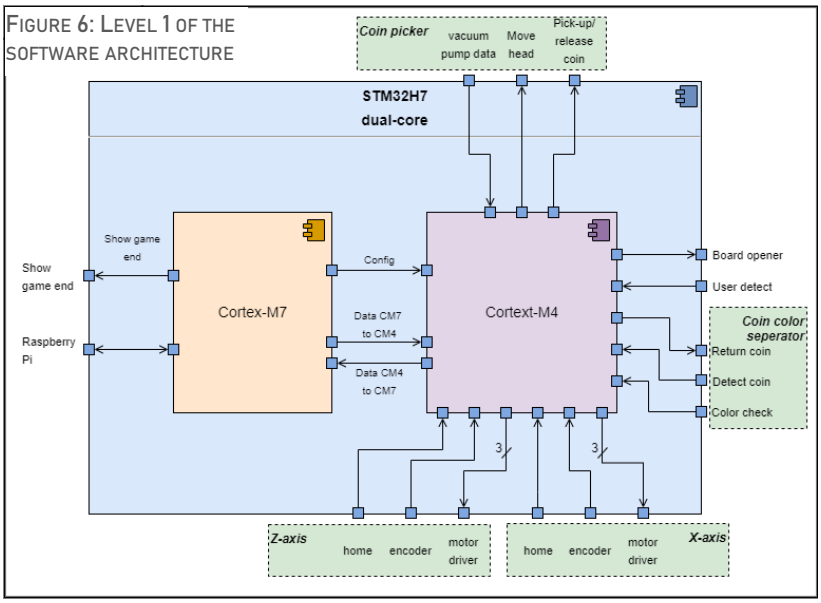


FIGURE 6: LEVEL 1 OF THE SOFTWARE ARCHITECTURE

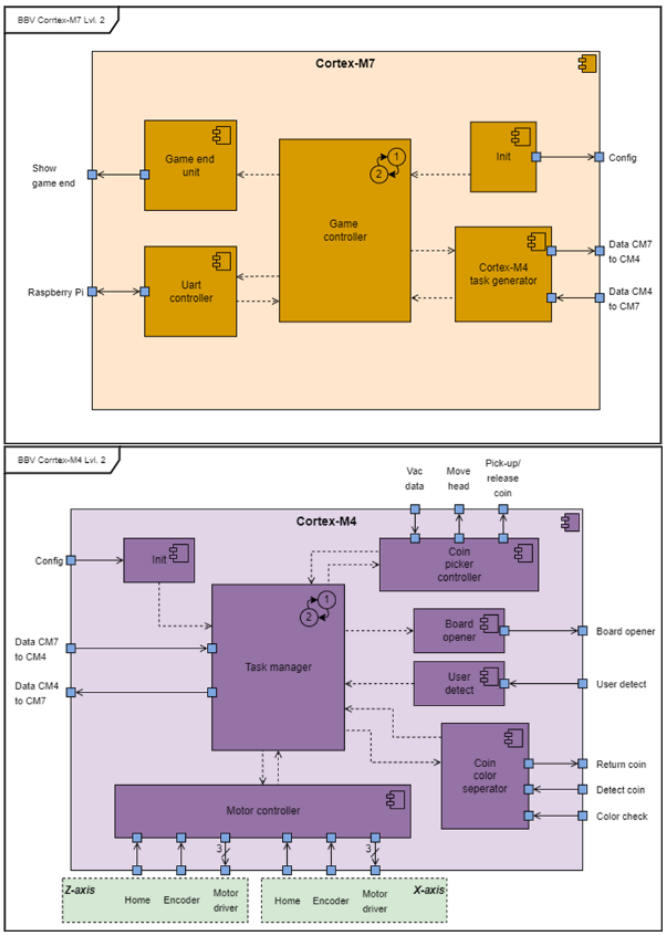


FIGURE 7: LEVEL 2 OF THE SOFTWARE ARCHITECTURE FOR BOTH CORES

The second level is as deep as it goes for Cortex-M7 since the rest of the functionality is out of scope for this project and is for future upgrades. However, this doesn't mean that there's no work to be done on this level. The game controller is the main finite state machine of the Connect-4 robot player, and together with the task generator, they are responsible for the whole gameplay loop. A part of the task for this assignment is to improve that state machine.

For Cortex-M4 the second layer describes another set of controllers, the hardware of the robot. It is apparent that another level would be needed to explain the full functionality of these blocks. However, on this level, it can be clear how the game operates. There are modules for picking up the tokens, bringing them from place to place, a module to separate user and robot tokens to their respective places, a token detector, a board opening module and the main controller of this layer, the task manager.

Layer 3 is the last one from the software architecture. The blocks there describe the lowest-level components that make up the system. For example, the blocks Motor X and Motor Z, together with PID X and PID Z, or block controlling the vacuum pump or the variety of sensors present in the system. As mentioned before, the description of this architecture is the work of a previous assignment and further detail is saved due to brevity [5]. However, a part of this current assignment is to evaluate how good the architecture will be in practice.

2. Investigating the microcontroller and the existing software

In parallel with the software architecture research, the Nucleo-144 board containing STM32H745/55 was investigated. Necessary in order to grasp the basic principles behind its operation and how the boards are set up for development.

STM has a proprietary IDE (STM32CubeIDE) with which the board could be programmed and is what's been used by the previous designers of the system. It has the unique feature of being able to configure all the features that the microcontroller has to offer, and auto-generates code for the initialization of the device. This is a newer addition to the CubeIDE, and such a configuration file does not exist for the legacy code of

```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

FIGURE 8: USER CODE MARKING

the Connect-4 system. Because parts of the code are auto-generated great attention needs to be paid to several matters.

First and most important is to only write code in the designated places within the project files, which wasn't the case for all of the demo projects received. When the developers' input is required to program a feature that should exist in the auto-generated sections of code, that section where it should be placed is marked as " USER CODE BEGIN/END".

Secondly, the generated code and the programming of the microcontroller are made easier through the STM-provided library, HAL (Hardware Abstraction Layer). It provides a simple, generic set of APIs (application programming interfaces) to interact with higher-level logic, while abstracting the low-level complexities of hardware [6].

A good starting point to understand how the system works on a low-level is to look into the devices it uses to function and by referring the dual-core architecture. In **Figure 4** some of them could be noticed, like the PWM and I2C.

The full list of peripherals in use and their functions are as follows:

- GPIO, General Purpose Input/Output Pins, which control the external devices.
- NVIC, Nested Vectored Interrupt Controller, one for each core, to take care of interrupts.
- RCC, Reset and Clock Control, to set the internal clocks.
- ADC, Analog to Digital Converter, to transform the vacuum sensor data.
- TIM, Timers, multiples of which control the PWMs, for the motors and servos, and the encoders.
- ETH, Ethernet connection for future upgrades.
- I2C, the I2C communication protocol, facilitates the connection with some sensors.
- UART, the Universal Asynchronous Receiver/Transmitter protocol, facilitate communication with the Raspberry Pi and the Operator of the system.

Through the inspection of the software projects the necessary peripheral devices could be identified and compared. On the intranet of ALTEN, three projects could be found initially:

- One is called the "*Legacy Project*", the initial working version from the old STM32F.
- Of the other two found, one is called "*Connect-4 Demo*", which has code for the Task Generator based on **Figure 9**: Send-Event Instruction notification mechanism through HSEM (Hardware Semaphores) and the basic FSM on the game logic described in the software architecture design.
- The other one is called *"Dual-Core Communication Demo",* which has code that describes the data exchange between the two cores, through the use of HSEM, and a shared buffer.
- The 4th project was received from the person responsible for the last internship project after the start of this internship, and it is called the *"Initialization Demo"*. It is based on the "Connect-4 Demo" project, but doesn't include all its functionalities, and only boots up the system.

Due to a lack of documentation, most of the logic of the projects was hard to follow, but since success had been achieved with the demo projects, it was important to learn from them and deem appropriate what is available for use and what was not. It's important to note, that only the Initialization Demo was able to run on the current machine due to hardware configurations, meaning that the old codes had limited capabilities to be tested/debugged on the current hardware.

3. Gameplay Logic Improvement

Based on a recommendation from a previous report of the system, which stated that the main logic of the system needs improvement, research into what that meant was done. Contact was established with the person who had worked on it since he is a consultant at ALTEN. The following information was collected: The state machine was basic and needed expansion and improvement, to set-up a new project that adheres to the file structure dictated by the architecture.

To understand how the state machine could be improved, first the topics of synchronization and hardware semaphores have to be explained. Synchronization is a key component of the state machines of the robot, since there are two cores that operate in parallel. The cores will have to communicate with each other and exchange some data. In the following paragraphs the questions of how one core knows that the other has written the data to memory and how does it know when to look in the memory will be answered. A hardware semaphore is a synchronization primitive [8], used in projects with multiple cores to synchronize processes together. In general, it is used to control access to a common resource to the cores. And there are several types, however for this implementation, the only focus will be on binary semaphores. That is, a semaphore that has only two states: locked or unlocked. When a semaphore is locked, if another process wants to access the same resource currently occupied by the semaphore, it has to wait for it to finish and unlock the resource before it can access it.

From the previous work done on this topic, a location in SRAM4 memory was found that is available for the hardware semaphores. That is, a special place in memory where the data could be exchanged between the two cores in a safe and atomic manner [9]. This would be where the data for which column to play, or at which column the user has dropped the token, would have to be stored.

Additionally, in the "Connect-4 Demo" it is shown how the HSEMs are used like notifications, which in essence establish a way for the two cores to know what the other one is doing. Applicable in this case since it is desired for one core to be considered a Primary-core and the other one Secondary-Core.

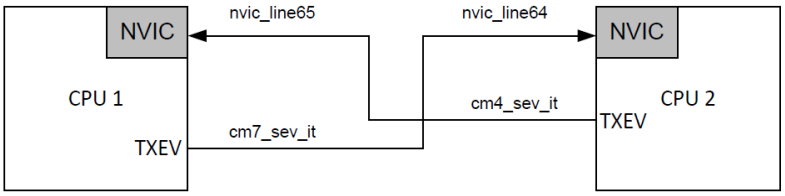


FIGURE 9: SEND-EVENT INSTRUCTION NOTIFICATION MECHANISM [7]

Through the notifications in the cores: states could be advanced and synchronized to each other; operations could be performed on time. As such, one core will send notifications with commands, while

the other will send notifications with status updates. One such notification “task” exists as an example in one of the projects. The rest have to still be designed.

To sum up the improvements, with the notification method, the Cortex-M4 will know when to look into the memory to read off the location where it has to play its next move. Also, through locking of the HSEM, it is ensured that only one core has access to any peripheral at any given time. And also, to expand the current tasks and give a clear meaning to their functions. Add tasks which describe the system to a fuller extent and finally implement the design of the FSM.

4. Validating the current system

Another part of the assignment is to validate the current system and create a test plan for current and future use. In order to achieve this, several principles from software testing have to be used. For example, unit tests had to be created, as well as functional tests of each block from the software architecture. Furthermore, a happy-path test was also drafted and finally acceptance tests, to judge if the system does what it is supposed to according to the client’s needs.

Unit tests are tests that are designed to find defects and verify the functionality of the software [9]. They are concerned with the smallest possible block that the system uses to run. In this case, they are the STM32H peripherals like timers, UART and I2C, ADC etc... One evaluates for correct initialization of the separate modules and then for their core functionalities. In the case of timers that would be if they correctly count on each clock cycle if they generate interrupts if needed and if multiple timers work correctly as expected. Each module has its own unique features, for the full test plan, one can refer to D. TRD, Test Report Document.

Following the unit tests, are the functionality tests. They are the ones that assess the functionality of the higher-level blocks from the software architecture and judge if they are in accordance with the functional requirements of the system [10]. In the case of the Connect-4 robot player, the tests are conducted to see if the modules themselves work separately as expected regarding the requirements each of them has towards the bigger workings of the entire system.

Finally, a happy-path test should be conducted to check if the expected gameplay loop is behaving as, it should according to the user requirements. Such a test is intended to simulate how the end user will use the machine. The purpose isn’t to break the functionality of the machine, but to see if the machine works as intended by the design [11].

V. Specification

To ensure the success of the project and to meet the client's expectations, a set of user requirements and specifications has been developed. These requirements are defined according to the MoSCoW principle, which classifies them into four priority levels: Must, Should, Could and Will. This approach helps steer the development process and ensures that the most important requirements are met, while allowing flexibility for additional enhancements and features as time permits.

TABLE 2: USER REQUIREMENTS

ID	Requirement	Explanation	Priority
UR.1	The user shall be notified when the game ends.	The system should provide a clear indication to the user when the game has ended, either because one player has won, or because the game has ended in a draw.	Could
UR.2	The robot must detect a cheating player and respond by resetting the game.	A cheating player is someone who plays out of their turn, or someone who inserts two coins or more at once in one or several columns.	Must
UR.3	A Board Support Package (BSP) must be made of the operating system with which the necessary hardware components of the robot can be controlled.	BSP must be developed for the operating system, which will allow the necessary hardware components of the robot to be controlled. This will ensure that the system is able to operate reliably and consistently.	Must
UR.4	The insertion of a game token in an arbitrary column shall be detected by the photodiodes and IR sensors.	The system should be able to detect the insertion of a game token in any column using photodiodes and IR sensors. This will ensure that the robot is able to accurately detect the user's moves and respond accordingly.	Must
UR.5	The system is able empty the playfield, separate the tokens by colour and prepare itself for the next game.	The system should be able to automatically empty the playfield at the end of each game, separate the tokens by colour, and prepare itself for the next game. This could involve moving the tokens to a sorting base, as specified in the following sub-requirements.	Must
UR5.1	After a game, the tokens must move to the sorting base, by emptying the game board column by column.	In order to avoid obstruction during clearing the board game and make the token checking principle easier.	Must
UR5.2	From the sorting base, the yellow and red tokens shall be sorted and returned to their belonging base – on the user side.	The tokens must be sorted by colour at the sorting base and returned to their belonging base on the user side. This will ensure that the system is ready for the next game.	Must
UR5.3	A flipper will shoot the human (yellow) tokens back to their base.	This sub-requirement specifies that a flipper must be used to shoot the yellow tokens back to their base. This will ensure that the system is fully automated, and the user does not need to manually retrieve the tokens.	Must

UR.6	The robot head should be controlled to the desired X and Z position within 1.5mm accuracy	This requirement specifies that the robot head must be able to move to the desired X and Z position with a high degree of accuracy.	Should
UR.7	The robot end effector should suck up tokens by actuating the pressure air pump.	Research needs to be done on the sucking power w.r.t. the tokens.	Should
UR.8	The robot end effector must release the token at a given position to insert the token into board.	The robot must be capable of precise positioning and releasing of the token to ensure it goes into the correct slot.	Must
UR.9	The algorithm running on the Raspberry Pi could be integrated on the new STM32H7 dual core.	This means that the software running on the robot could be optimized for performance and power efficiency using the new hardware. The integration of the algorithm on a new platform may require modifications to the code and additional testing to ensure proper functioning.	Could
UR.10	Research the feasibility and capabilities of ethernet connectivity for future upgrades.	The system in the future will have need of higher speed of communications and bigger data flow, in order to keep track of high scores and any other data.	Could

VI. System Design

The following chapter provides a detailed exploration of the system's design, focusing on two crucial elements: the design of the first level of the architecture, which involves the creation of a finite state machine to handle the intricacies of the game, and the outline of Level 2 and Level 3, encompassing the fundamental building blocks that form the backbone of the machine.

1. The Design of Level 1: The finite state machine

One of the derivative tasks of improving the system was to improve the main gameplay logic of the Connect-4 robot. The one developed from the architecture was only to show the initial idea of how it might look in the end. The software modules that facilitate this are the ***Game Controller*** and ***Task Generator*** from Cortex-M7, and the ***Task Manager*** from Cortex-M4.

The initial idea of the redesign was to make the finite-state machines match as much as possible and add more states and clear definitions of what should happen in each state and why. Done so, that the synchronization can be represented and understood better by future developers and to be more accurate to the system's inner workings. Additionally, when more functionalities are added to certain stages of the game (game-end, cheat detected), it will make those additions easier to implement and debug. These new states are the ***cheat*** and ***clean-up*** states. Look at Table 3 and
Table 4 for further elaboration on the definition of all states in both cores.

The new states, require new tasks to be designed as well. Tasks and the aforementioned Figure 9: Send-Event Instruction notification mechanism are one and the same when looking at the code, just at different abstraction levels. The semaphores, which are the directive through which the tasks are done, are used to signal to each core when it should transition states.
Because CM-7 should be regarded as the primary core, the secondary should only execute whatever is needed only when the primary requests it. Through the notification system, we can identify what HSEM has been activated, and trigger a specific state change tied to the HSEM. Through this action, the CM-4 is now in the correct state to execute whatever CM-7 needs to do. Another semaphore will be activated, this time from CM-4 when it is done with its current action, in order to notify CM-7 that it is done with the work. Then CM-7 can transition states and decide what needs to happen next and repeat this whole cycle until a winner emerges.

More detailed information on the states can be found in the Table 3 and
Table 4. And the design of the FSM, can be seen in Figure 11: The Main FSM of each core and explained in The Design of Level 1: The finite state machine.

TABLE 3: THE PURPOSE OF EACH STATE, ITS TRIGGERS AND OUTPUTS FOR CORTEX-M7

Cortex-M7 States	Purpose of state	Input HSEM	Output HSEM
Initialize	To initialize CM7 and send an initialization signal to CM4	CM4_DONE	CM4_INIT
Start Game	Set game variables, request difficulty (when/if implemented), inform user that game is ready	None	None
User Move	Sends signal to CM4 which will legitimize any user input token. Future: Sends column in which user plays to RaspPi	CM4_DONE	USER_TURN
Idle	Checks for win conditions before advancing to next state	None	None
Robot Move	Sends signal to CM4 in which column robot should play Future: Request move from RaspPi	CM4_DONE	ROBOT_TURN
Cheat Detected	State to take care of necessary actions for this special case. Notify user.	CHEATER	None
Game End	State to take care of necessary actions for this special case. Notify user.	CM4_DONE	GAME_END
Clean-up	Prepares the game board for next player	CHEATER	None

TABLE 4: THE PURPOSE OF EACH STATE, ITS TRIGGERS AND OUTPUTS FOR CORTEX-M4

Cortex-M4 States	Purpose of state	Input HSEM	Output HSEM
Initialize	Initializes all necessary signals for operation of all sub-modules	None	CM4_DONE
Idle	A state of rest for the peripherals. Waiting for a task from CM7	CM4_INIT ROBOT_TURN USER_TURN GAME_END CLEAN_UP	
User Move	Waits for a token drop, and records it to memory	None	CM4_DONE
Robot Move	Moves to a location read from memory	None	CM4_DONE
Cheat Detected	A state existing to simplify the graph and understating of the concept that a cheat could be detected from any state. For this purpose, it is a floating state that is entered immediately upon a cheat detection.	None	CHEATER
Game End	Resets the game.	None	CM4_DONE
Clean-up	Executes the appropriate actions to clean up the game board.	None	CM4_DONE

2. The Outline of Level 2 and Level 3: The building blocks

In addition to the FSM improvements, a lot of software modules need to be adapted from the old software projects to perform desirably in the new architecture. To be able to describe the initial situation in more detail, a table was created to keep the structure concise. A list of all modules that build up the machine is made, it shows if the module already exists and needs a redesign, where it originates from or if the module is yet to be designed. This information was gathered by meticulously debugging and researching the various project that is named.

TABLE 5: MODULES ORIGINS AND THEIR INITIAL CONDITIONS

Software Modules	**To be improved**	**To be designed**	**Details**
Motor X	✓		Basic structure in *„Initialization Demo“,* however it wasn't functioning at all.
Motor Z	✓		Basic structure in *„Initialization Demo“,* however it wasn't functioning at all.
PID X	✓		Exists in the *„Legacy Code“*, due to time constrains explain in the introduction it was omitted in this design.
PID Z	✓		Exists in the *„Legacy Code“*, due to time constrains explain in the introduction it was omitted in this design.
Motor Master	✓		When the building blocks were fixed, the simple functions of the block worked.
Encoder	✓		Tested and working as expected.
Home-Switch	✓		Tested and working as expected.
Motor Driver	✓		Existing settings on their proprietary software however, some of them had to be fine-tuned.
Colour Sensor		✓	Program the sensor.
Proxy Sensor		✓	Program the sensor. However, no physical connection exists on the machine.
Solenoid	✓		Tested and working as expected, before the break.
Token Separator Master	✓		The higher logic needs to be adapted.
Servo Controller		✓	Calculations of timers existed, however the software had to be designed.
Vacuum Pump	✓		Software exists from *„Initialization Demo“*, however there's mechanical issues with air leaks and not being able to hold enough pressure to hold the tokens.
Token Picker Master		✓	The logic of the module needs to be designed.
User Detector		✓	The logic needs to be designed.
Board Opener		✓	The logic needs to be designed.
Init-CM4		✓	The logic needs to be designed.
Init-CM7		✓	The logic needs to be designed.
Task Manager	✓		Rudimentary version exists in *„Connect-4 Demo“*, expansion of functionality needed.
Game Controller	✓		Rudimentary version exists in *„Connect-4 Demo“*, expansion of functionality needed.
Task Generator	✓		Rudimentary version exists in *„Connect-4 Demo“*, expansion of functionality needed.
UART controller	✓		Rudimentary version exists in *„Connect-4 Demo“*, expansion of functionality needed.
Game end unit		✓	The logic needs to be designed.

VII. Realization of the system

This chapter delves into the software modules of the system implemented within the Connect-4 project. This chapter focuses on two significant aspects: the optimization of existing modules and the detailed design of new modules.
Additionally, the modules marked in Table 5: Modules origins and their initial conditions are divided into two groups. Ones that had some basic structure before this project assignment, and ones that need to be designed completely.

1. Optimization of the existing modules

The modules in the "*To be improved*" category had to be optimized, as mentioned before, to function as necessary and document the code itself to make sure that further development is not hindered by undocumented code. A coding standard was also introduced in the face of BSD/Allman [12], for the sake of readability and maintainability. The optimization wasn't only introducing a coding standard and fixing the mistakes in the code, but also introducing definitions for values for even more modularity, breaking down the functions into smaller ones so that only simple steps are executed per call. Making the debugging process easier and bringing up the reusability of certain modules and operations.

The section will start with looking at the movements of the machine. Both motor blocks had the correct PWM settings in their respective timers and a semi-functioning structure however, a lot of the values used were wrong and through testing, they had to be adjusted, alongside the logic of the block itself. The optimization included changing the lower-level functions to ones that make more logical sense and show the flow of the code more accurately, documenting the code thoroughly, unifying the variables since a lot of them were different data types, and readjusting the threshold values to make the motors move and additionally map out all of the possible movements that could be done by the robot at any point of operation. Except, some of the Z-axis movement since the pump doesn't function and accurate guesses cannot be made about precise positions.

The motor master block also received expansions, mostly communication-based, so that it now gives status updates to the machine operator on its actions. This makes for easier debugging when playing around with the speed of the motor or when something crashes, to know exactly when and where. Additionally, this information could be used at a later stage when a screen is added to the machine.
The motor controller is concerned with the specific positions that the machine will have to traverse. They have also been mapped out, however, because there's no end-switch, the robot crashes if it reaches the end of the X-axis, and the end is one of the positions needed to be reached to be able to grab tokens from the cleaning dump location. A simple push towards the home location solves the issue for now, but an end-switch would be a far better solution. Additionally, the crash could be prevented when the PID controllers are implemented in the motor control loop.

The software issues, however, weren't leading in solving the problems of the whole movement module of the robot. There was a lot of necessary troubleshooting on the motor drivers, encoders, and the backplane's connectors since initially the programming of them was wrong and the signals received were very off. Some internal settings of the motors had to be changed, like the addition of a larger current offset on the X-axis motor.

Following that insight into the movement of the robot, a dive into the sensors of the system will be made. The user detector system, the RBG sensor, and the proximity sensor use the I2C protocol to communicate with the microcontroller. There's a written example in the *Connect-4 Demo* that shows how the proximity sensor uses the I2C to get the value from the sensor. Unfortunately, this is the only sensor that isn't physically connected to the robot through the backplane PCB. However, the example could be used to work out how the communication works and afterwards to get the needed data.
The user detector module increments a value in memory that stores the number of tokens in each column. It does that when it recognizes a token has been dropped. By way of detecting the emitted light from the diodes in the light gate, when that same light connection is broken, something must be passing by. And various function is implemented to show in which column the token was dropped if there are multiple insertions at a time. This section of code is adapted from the "*Legacy Code*". It is important to mention that 4 out of the 8 inputs of the I/O expander facilitating the light-gate are swapped, which results in the detection of a token from column 2 appearing on the input of column 1 and vice versa. The same is the case for columns 3 and 4. This is fixed in the software, but important to note. The main complication of the module was programming the sensor correctly with its initial settings. And beforehand, fixing some issues in the I2C library from older projects. Instructions were mostly taken from the datasheet of the PCA9554 device.

2. The detailed design of the new modules

The other category is the modules that needed to be designed from the ground up. The modules that reside on level 3 of the architecture are mostly the basic functionality of the respective devices. Most of the design here is done based on recommendations from the datasheets of each device. The calculations for the servo control come from Parallax [13]. For the RGB sensor the design is also based on the datasheet [14] and information from the *"Legacy Code"*.

The board opener is part of the cleaning task. A servo motor controls the position of a piece that opens the bottom side of the board. The existing code was not functional and had to be redesigned. The servo positioning calculation had to be re-mapped, alongside the positions for each column. Due to the physical construction of the opener piece, the opening is not linear, and through trial and error, all the positions were found.

The token picker master is mainly concerned with the end-effector of the robot which has another servo in it, and it also facilitates the "grabbing" function of the robot, however, because the issues of the vacuum pump are rooted in its hardware, the fix for that is out of scope for this assignment. The positions of its servo are mapped out, and a structure is set up for when the VAC is fixed.

The initialization modules of each core are pretty self-explanatory. They need to facilitate the initiation of all needed control signals, variables, structures, handles, peripheral devices, and sensors that will be needed in the smooth running of the respective core. They also need to initialize any intermediate functions that are also required in the same processes.

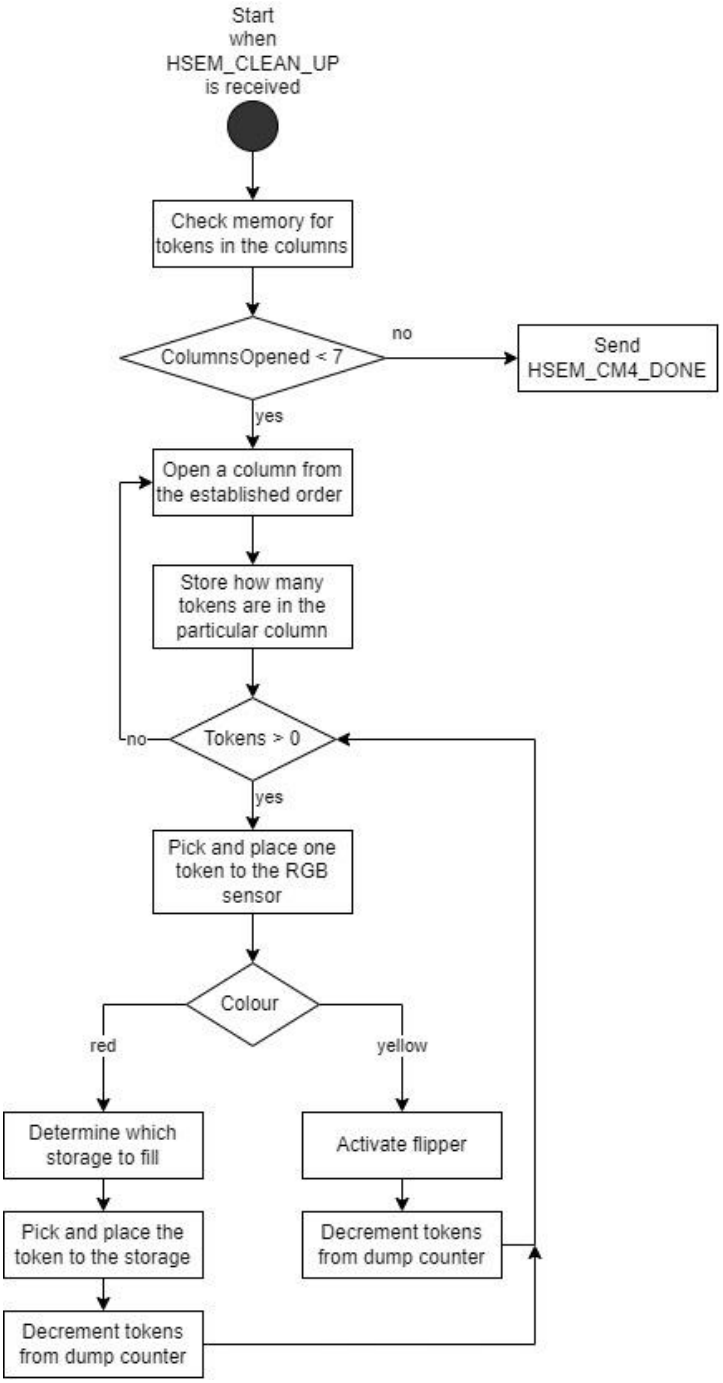


FIGURE 10: TOKEN SEPARATOR CONTROLLER

The token colour separator module has to make decisions based on the colour and proximity sensors. First, the sensor had to be programmed with initial settings like powering on and the value of its integration time. After that, the reading of the colours happens from the registers on the device. The data from them is transformed according to information from the datasheet and the *Legacy Code* [14]. This is then used to determine if the colour of the token is red or yellow through several software functions. Upon detection a user message is sent, and based on the colour two actions can happen as shown in the flowchart.

The design of this module has been created, but not yet implemented in the system, due to time constraints. However, the structure of the software for its implementation is available.

The state machine for both cores have been designed according to the tables in the previous chapters. And the design could be seen in Figure 11: The Main FSM of each core.

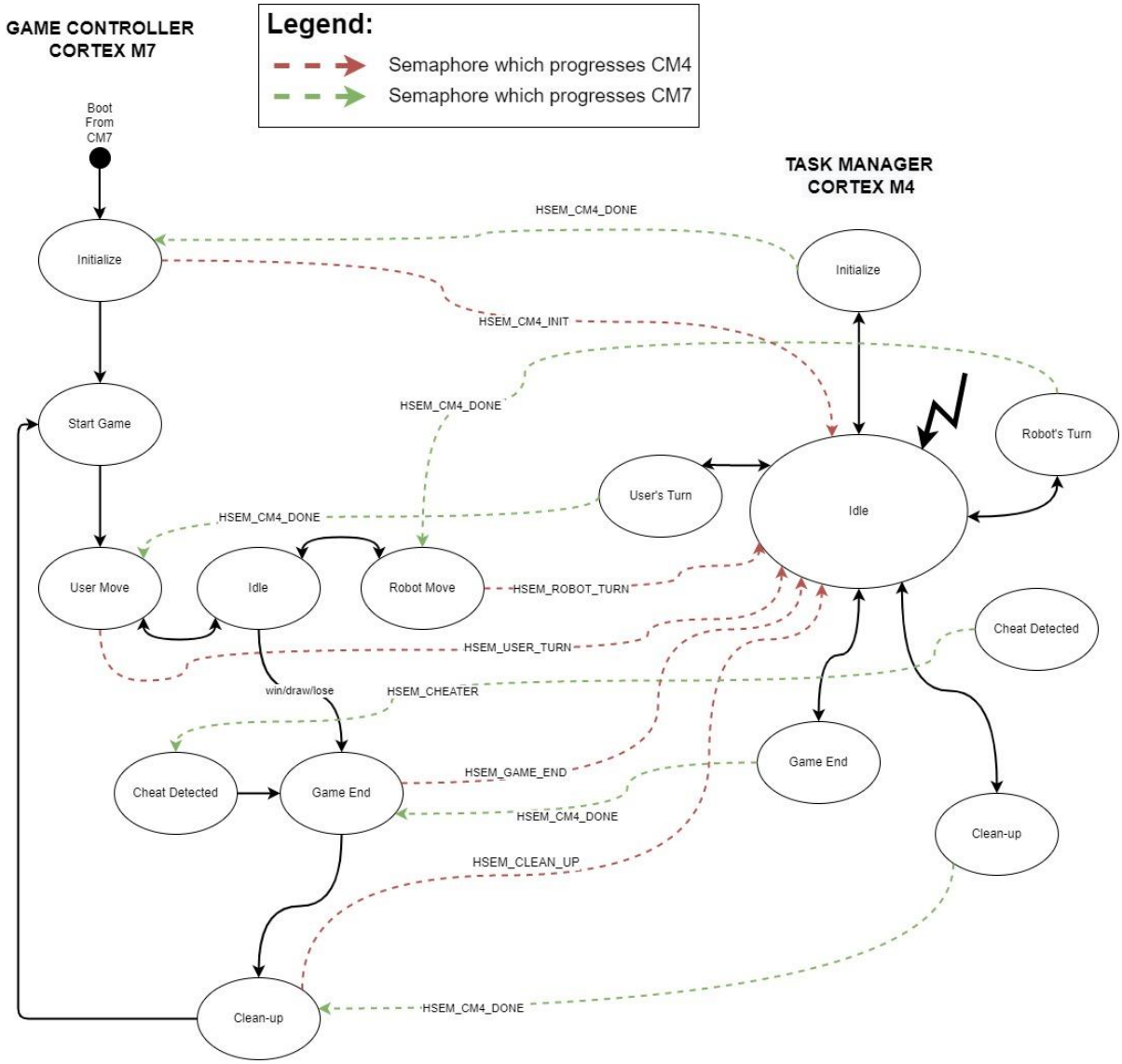


FIGURE 11: THE MAIN FSM OF EACH CORE

The execution of the state machine is facilitated by three separate modules of the software. Of these three modules, the task generator will be examined first, as it is the simplest and shortest module. The new addition to it is that it has more tasks in the form of more HSEMs added to its software.

The other two modules, the task generator and the game controller are the blocks that support the FSM themselves. Their transitions, condition checks, and the execution of the different **Table** 3: The purpose of each state, its triggers and outputs for Cortex-M7. The state machines have only been designed, and their implementation will be done in the time between the submission of this report and the presentation.

The UART controller required more work. It had to be fixed since the output was jumbled characters. The settings of the UART itself were fine, however, the function that sent the message had several

issues. Several new functions were created to cater to the needs of different types of messages that need to be sent. Namely, ones that include data and its volume. Then they were included throughout the example codes written about the locations stored in the motor master code file. It is the intention for such a message to exist after each important action that the machine makes.

This is helpful for two reasons, firstly it helps to debug while programming some features, to know exactly where the mistake occurs, and secondly, it gives more information about the operation and intention of the machine to the operator, which at a later stage could be conveyed to the user via the future development of an HMI.

3. Software issues during realization

In the following section some of the issues that were found during the project and their solutions will be presented.

During the initial set-up of the project, while examining the .ioc files of the different projects, an option to migrate the project to the newer version of STM32CubeMX appeared or to continue with the old settings of the project. Initially, the project was migrated, however, that created several problems, the root of which was the RCC and the power configuration supply source. When the project was migrated, these settings changed to ones that were undesired for the system, and when tried to program the core like that, the system entered what is known as a deadlock state [15]. Then a manual hardware reset has to be performed on the board. For that purpose, two pins were soldered which can be seen in Figure 3 point 11. More detailed information about unlocking the core can be gained from reading the user manual of the Nucleo Board [15].

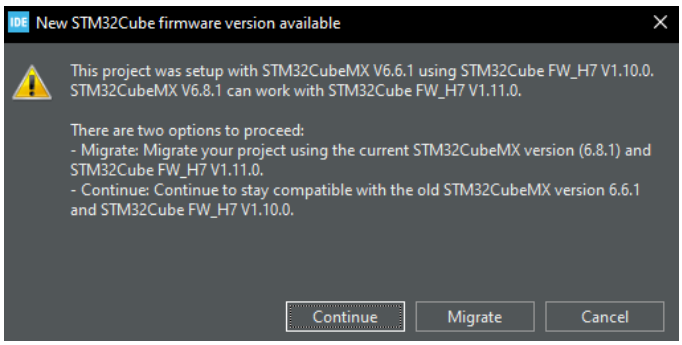


FIGURE 12: FIRMWARE VERSION OPTIONS

To sum up this whole section, the base-level modules of the system are implemented and working on an operational level. There are still a lot of features that can be added to the system to make it fully functional. And there's room for a lot of further future improvement.

VIII. Verification and validation

This chapter will be about the completion status of the initial user requirements that were set out.

ID	Requirement	Explanation	Priority	✓
UR.1	The robot must detect a cheating player and respond by resetting the game.	A cheating player is someone who plays out of their turn, or someone who inserts two coins or more at once in one or several columns.	Must	
UR.2	A Board Support Package (BSP) must be made for the operation of the robot.	BSP must be developed for the robot, which will allow the necessary hardware components of the robot to be controlled.	Must	
UR.3	The insertion of a game token in an arbitrary column shall be detected by the photodiodes and IR sensors.	The system should be able to detect the insertion of a game token in any column using photodiodes and IR sensors.	Must	
UR.4	The system is able empty the playfield, separate the tokens by colour and prepare itself for the next game.	The system should be able to clean up automatically.	Must	
UR4.1	After a game, the tokens must move to the sorting base, by emptying the game board column by column.	In order to avoid obstruction during clearing the board game and make the token checking principle easier.	Must	
UR4.2	From the sorting base, the yellow and red tokens must be returned to their appropriate place.	The tokens must return to their belonging base on the user side or in the storage space of the robot.	Must	
UR4.3	A flipper will shoot the yellow tokens back to their base.	This sub-requirement specifies that a flipper must be used to shoot the yellow tokens back to their base.	Must	
UR.5	The robot end-effector should be able to traverse to the desired X and Z position within 2mm accuracy.	This requirement specifies that the robot head must be able to move to the desired X and Z.	Should	
UR.6	The robot end effector should suck up tokens by actuating the pressure air pump.	Research needs to be done on the sucking power w.r.t. the tokens.	Should	
UR.7	The robot must release the token at a given position to insert the token into board.	To ensure that the token goes into the correct slot.	Must	
UR.8	The algorithm running on the Raspberry Pi could be integrated on the new STM32H7 dual core.	This means that the software running on the robot could be optimized for performance and power efficiency using the new hardware.	Could	
UR.9	Research the feasibility and capabilities of ethernet connectivity for future upgrades.	The system in the future will have need of higher speed of communications and bigger data flow.	Could	

The green-marked requirements are fully complete. The orange colour marks completion to a high degree, not counted as green due to hardware failure. Red means that the requirements haven't been fulfilled due to low priority or due hardware failure.

For UR. 4, the sub-requirement of UR 4.3 couldn't be completed because of the aforementioned circuitry fault for the flipper. Hence the uncompleted status of UR. 4 and the red mark for UR 4.3, however, as also mentioned, the software structure is built for whenever the flipper works. Similar reasoning for UR.7. Because its main functionality depends on the vacuum module, for which during the course of the internship it was determined that the issue would take a lot more work to fix than anticipated. Mostly because the issues were hardware-related, the issue was left for another engineer to fix.

IX. Conclusions

To sum up the results of the internship assignment, the implementation of the architecture was successful, to the desired degree of basic operation of the entire system. All critical sub-modules for the gameplay are designed and implemented, but not complete to their full potential, because thorough and extensive development of software takes a lot of time. This could be considered a successful result since the robot was completely unfunctional as a whole at the beginning of the assignment, and currently, it is working completely, only without the highest level of logic having been applied yet.

Future upgrades will have the opportunity to be added much more easily than before, due to the modular approach taken with the development of the software. The variety of old projects with different functions were all combined into one project with all of their functionalities working together. Various bugs were fixed across all of them, alongside optimizing specific sections of code, adapting them for the new architecture and the new microcontroller and adding comments with explanations for all major logic in the software.

Alongside the software implementation, research was done into what would be needed to evaluate in order to traverse along the V-model that the project was set to follow. A test report was created, which is attached in the appendix, alongside the code that I have written.

X. Recommendations

There are a lot of improvements that could be done to the system, but in this excerpt, the focus will be on ones found during the implementation phase of the internship and ones that directly hinder the performance of the robot immediately. The issues are a mix of both electrical and mechanical nature.

The vacuum pump cannot build enough pressure to hold the token, and from further inspection, several points of possible failure were seen. The cable's end at the end-effector seems to easily slip out since no right places it tightly in the spot it goes. The cable for the pump is long and winding and there may be a leak either somewhere in it, or on the other connection point as well. Furthermore, the control circuitry for the vacuum pump burned, due to incorrect placement of the MOSFETs, and they have to be replaced. Also, two other MOSFETs burned, and they are for the flipper/solenoid and the valve for the vacuum module.

The proximity sensor is not connected to the back plain PCB. And while the focus is on the PCB, the connector types sometimes pop the cables out of the sockets, so if there's a future revision of the PCB, more secure connectors should be used. Either that or some cable management of the cabinet could fix the issue since currently, it is very hard to navigate anything there.

Due to development time, the PID controllers for the motors couldn't be integrated with the new codebase, and their implementation would solve a lot of potential issues with the movement. Additionally, end switches still have to be added on the axis so that the robot knows when the end has been reached. Also, on the Z-axis motor, a plastic holder for the cable broke, which makes the structure lag a little due to its weight, currently, they are strung together with bolts and zip-ties. However, a 3D-printed module would be much better, the old one had thin walls that over time gave up to the pressure.

In the software, a lot of notes have been left, marked with "TODO" which shows a special icon in the IDE, and these are places where I thought something could be done better, but due to time limitations, I couldn't have done it in time.

Some error handling exists in the system, but not nearly enough to be substantial and to be able to recover the system.

Evaluation

During my study of electrical engineering, I discovered that my drive for technology and engineering came from designing embedded systems. My time at Fontys helped me learn the necessary material needed for me to become an embedded developer.

Naturally, I pursued an internship in that field, and I am so grateful that I discovered ALTEN. The whole experience from start to finish has been nothing but pleasant, though not free of challenges, as is the nature of engineering.

Within the company I had many colleagues that I could learn from and ask for guidance, which was crucial when I was dealing with such a large software project. There was an initial bump of not having all the correct software projects in possession, and during this time I focused more on the validation part of the system and research into it. As I've had no previous experience in test engineering, I had the opportunity to meet with a Technical Manager at ALTEN, and have a conversation with them about how I can approach the matter best. At that moment I saw and understood the great benefit of working as a consultant, and that was that you can really easily continue to develop your own knowledge on different matters of engineering. Also there were plenty of people within the department that I worked at that could give me constructure feedback for my work, which only makes its quality improve.

Bibliography

[1] "ALTEN - 2022 report," 2022-11-02.

[2] "Alten - Services," ALTEN, [Online]. Available: https://www.alten.com/services/. [Accessed 03 2023].

[3] ALTEN, "Technical Software," ALTEN, 2023. [Online]. Available: https://www.alten.nl/en/technical-software/.

[4] ALTEN, "Mechatronics," ALTEN, 2023. [Online]. Available: https://www.alten.nl/en/mechatronics/.

[5] P. Faatz, "Software Architecture Document," https://redmine.alten.nl/projects/in-a-row/repository/192/revisions/758/show/51.%20Software%20Engineering/1.%20Repo/trunk/1.%20Design/Dual-core%20low%20level%20design, 2022.

[6] ST, "Description of STM32H7 HAL and low-layer drivers | Introduction," [Online]. Available: https://www.st.com/resource/en/user_manual/um2217-description-of-stm32h7-hal-and-lowlayer-drivers-stmicroelectronics.pdf.

[7] STMicroelectronics, "AN5617," in *EXTI controller and send-event instruction*.

[8] OpenCSF, "Computer Systems Fundamentals," [Online]. Available: https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/SynchOverview.html.

[9] R. B. E. v. V. Dorothy Graham, "2.2.1 Component Testing," in *Foundations of software testing - ISTQB Certifications*.

[10] R. B. E. v. V. Dorothy Graham, "2.3.1 Functional testing," in *Foundations of software testing - ISTQB Cerfification*.

[11] TestLodge. [Online]. Available: https://blog.testlodge.com/what-is-happy-path-testing/.

[12] catb, "The on-line hacker Jargon File. 4.4.7.," [Online]. Available: http://www.catb.org/jargon/html/I/indent-style.html.

[13] PARALLAX, "Parallax Standard Servo (#900-00005)," 2022.

[14] TAOS, "TCS3472 COLOR LIGHT-TO-DIGITAL CONVERTER WITH IR FILTER," 2012.

[15] STMicroelectronics, "UM2408 - 6.4.8".

[16] STMicroelectronics, "AN5617," in *2.2 Working techniques*.

Appendix

A. Originality Declaration

Declaration of originality

of the Bachelor's thesis entitled:

Designing an Autonomous Robot-Player for Connect-4

Hereby I declare that this submitted Bachelor's thesis, with the title as mentioned above, is original and I also declare that I personally have written this Bachelor's thesis.

In order to write this Bachelor's thesis, I did the necessary investigations (as a part of my graduation project).

In case of the use of material not written by myself (parts of articles, reference books, graphs, etc.) I will indicate (by using the mark '**') this material in my Bachelor's thesis. I will also state the sources of these contributions (author, name of the magazine, source of the graph, etc.) in the list of literature (part of this Bachelor's thesis).

Date: 06/06/2023

Students name: Boris Nikolaev Ivanov

Students signature: 

Fontys
University of Applied Sciences

Graduation
Report

B. Project plan

Plan of Approach

Designing an Autonomous Robot Player for Connect-4

Name of client: ALTEN
Name of supervisor: Michael van der Velden
Publication date: 6-Jun-23

Version History			
Version	***Date***	***State***	***Comment***
1	10-02-2023	Draft	First draft set-up
2	13-02-2023	Draft	Put main bodies of text
3	16-02-2023	Finalizing draft	Correction from the feedback from the technical supervisor
4	03-03-2023	Final version	Correction after the supervisors meeting
5	06-03-2023	Final version	Correction from feedback

Acronyms and Abbreviations	
Term	**Explanation**
PoA	Plan of approach
BSP	Board support package
OS	Operating system

Referenced documents				
ID	**Reference**	**Title**	**Date**	**Author**

Index

1. Background information:

My graduation internship for Fontys Hogeschool will be conducted at the company ALTEN, with my task being to realize an embedded software architecture on an STM32H7 processor, for their 4-in-a-row robot, which was previously designed by another graduation project.

ALTEN is an international Technology and IT consultancy and Engineering company. Originally ALTEN was created in 1988 in France and currently they span over thirty countries with over 54100 employees, having established themselves in all the major sectors: Aeronautics & Space, Defence & Naval, Security, Automotive, Rail, Energy, Life Sciences, Finance, Retail, Telecommunications and Services.

Within the Netherlands, their expertise falls within the following categories: ALTEN IT, Technical Software and Mechatronics. The 4-in-a-row project falls within the Mechatronics department. Where my technical supervisor, Michael van der Velden, is also working as a consultant for ASML. With over 20 years of experience in the electrical engineering industry and weekly progress meetings, he has enough technical knowledge to guide me successfully through the project. Additionally, I have an appointed business manager, Gijs Haans, who is responsible for our personal development within the company and progress as engineers.

ALTEN has in-house projects, which are often used to develop new skills for consultants or the ones of interns. The 4-in-a-row (Connect4, Four Up) robot was developed for demos at trade fairs and open days at universities. The robot game is meant to demonstrate the knowledge of the consultants of ALTEN, and it is therefore developed with industrial components.

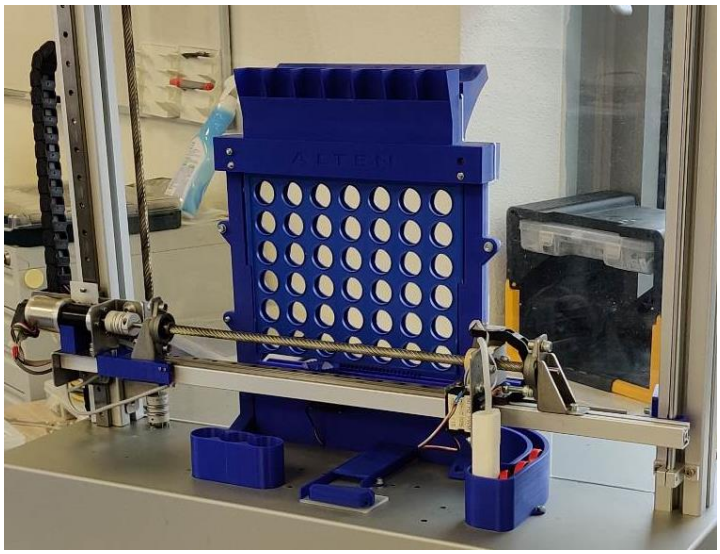


FIGURE 13: CONNECT4 ROBOT

The game is simple, there is a seven-by-six rack board, with slots at each spot for two coloured tokens. A red one and a yellow one. The first player to Connect 4 tokens in any direction wins. In our case, one player is a human, the other one is a robot. It is a completely autonomous process. After a token has been placed in the idle robot, the machine can calculate its next move based on a difficulty setting. To be able to execute everything, the 4-in-a-row robot is equipped with ‘X’ and ‘Z’ plane motors, a rotating vacuum gripper, and a routine to clear the board and reset the tokens.

2. Project results:

2.1 Goals of the project:

Implement the previously designed software architecture for the new STM32H755ZIT6U controller. Previously this system and its last iteration were running on a single Cortex-M4 core. Which was not powerful enough to provide resources for the software and hardware expansions ALTEN wanted to introduce to it. Therefore, they decided to upgrade the system with a dual-core processor. With this new requirement, a new architecture was designed and its feasibility was proven with a demo code.

My task will involve designing the modules to make the system reliable and functional to the best of its capacity. That will involve writing code for the needed modules, improving and adapting flowcharts/logic, and redesigning modules that do not function as expected from the software architecture. Further steps will include the further optimization of the BSP and testing on the robot itself. Moreover, research on ethernet communication with the robot will also be investigated, as a secondary goal.

The project will not involve integrating the game-logic part (the module where the next decision for the robot is made), on the Cortex-M7 core, but it will keep its function as it is currently.

2.2 Problem definition:

As mentioned before, the system has had an upgrade in hardware and because of that a new architecture was designed. My task is to implement the new dual-core architecture by validating/testing the design and by implementing the necessary software modules to make the 4-in-a-row robot perform more reliably. Secondary to that goal is the research of ethernet communication with the system. That will ensure that future upgrades of the system have a starting point to go off of.

Within the software architecture, I have complete freedom of design and choice in implementing the functions and how they are programmed. Alongside the ability to re-evaluate the designed modules and if needed redesign them after agreement with the client.

2.3 Description of the project result:

1. Designed software modules from the architecture design.
2. Tested modules on the Connect4 robot.
3. Research on ethernet communication for the system.

2.4 Design model

The V-Model is a useful tool to manage and deliver projects. By using this model, the student can ensure that their project is completed systematically and efficiently, and that the requirements set out at the beginning of the project are met.

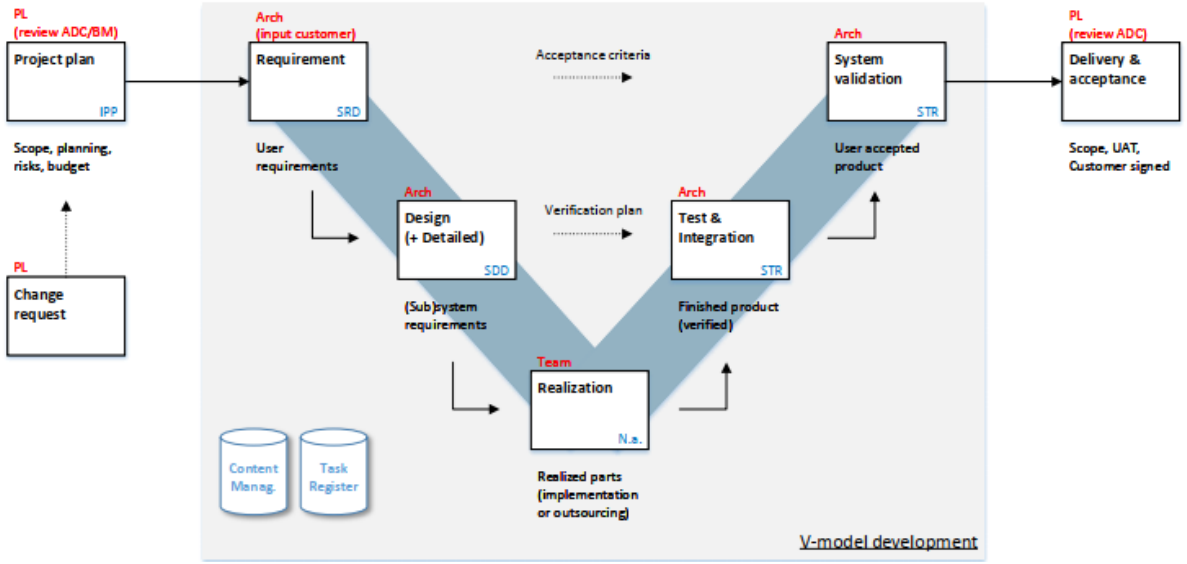


FIGURE 14: THE V-MODEL

3. Project activities:

1. Research key concepts about STM32
    a. Memory, Timers, Interrupts, Communication protocols, etc.
2. Redesigning flowcharts for software modules
3. To study and get familiar with the environment of the STM32 controller
4. Programming modules from the architecture
5. Implementing modules from the architecture on the hardware

The programming done on this project will be in C.

4. Project boundaries:

The project is concerned with the re-evaluation (and if needed redesign) and implementation of the previously designed software architecture. The dual-core communication is worked out, but the rest of the modules(blocks) have to be implemented. More modules are building up than the ones seen in the diagrams below.

TABLE 6: SCOPE OF PROJECT

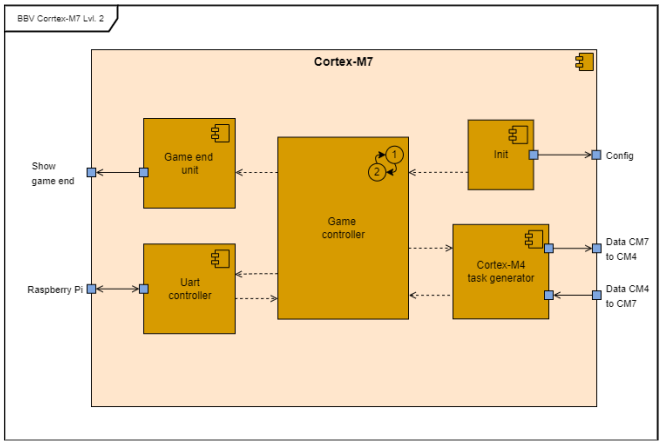


FIGURE 15: CORTEX-M7 BLOCK DIAGRAM

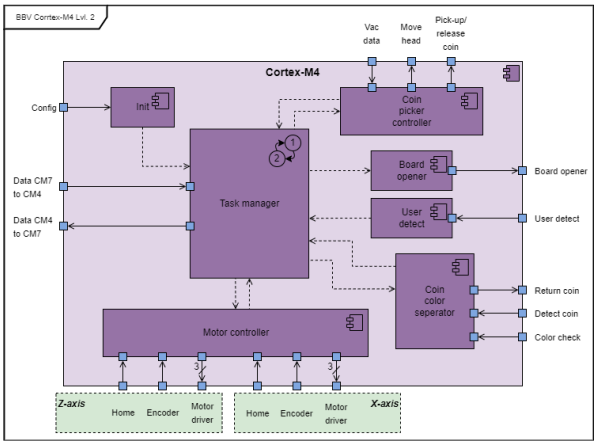


FIGURE 16: CORTEX-M4 BLOCK DIAGRAM

Project boundaries	**Within Scope ?**
Implement software modules	Yes
Redesign software modules	Yes
Research ethernet communication	Yes
Implementing ethernet communication	No
Redesign hardware/mechanics	No
Changes to the gameplay	No

5. Intermediate milestones:

1. System Requirements Document / System Design Document
2. Implementation of modules
3. Test Plan

6. Planning:

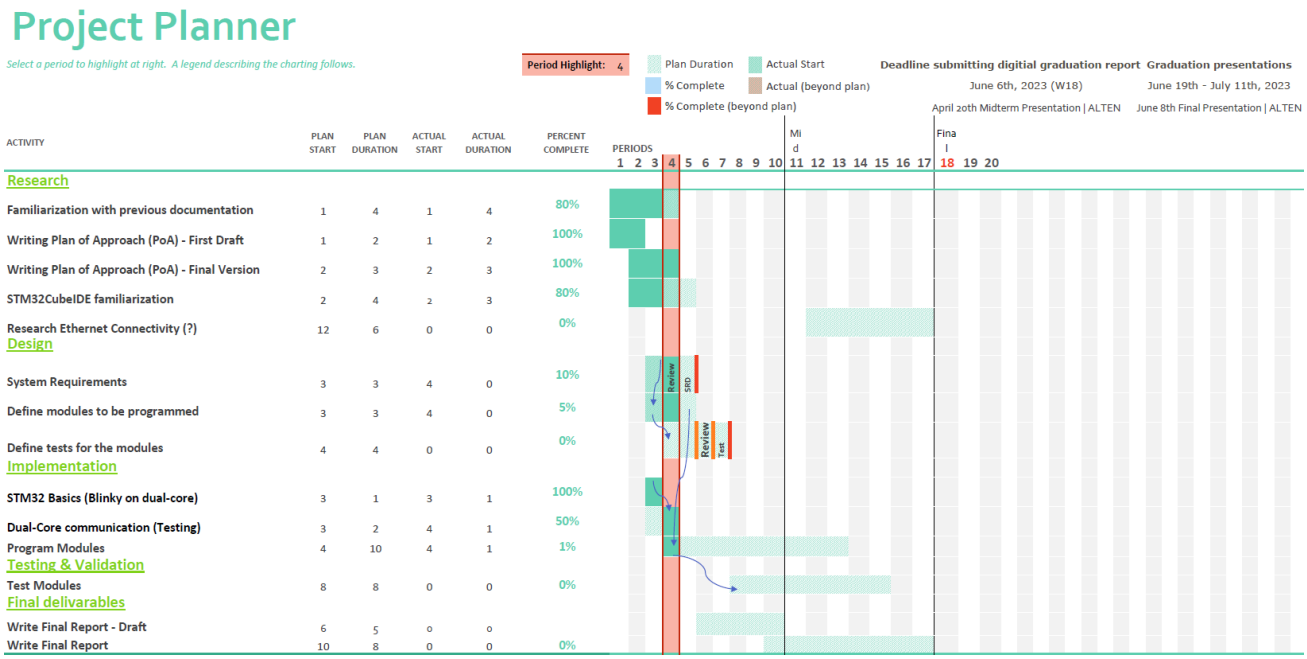


FIGURE 17: PLANNING OF THE PROJECT

7. Risks:

Risk description	Likelihood	Impact	Risk
Context: Hardware failure Event: Component malfunction	2	4	
Context: Software bugs Event: Software modules not working as expected	2	4	
Context: Integration issues Event: Unresponsive while testing new software modules	4	2	
Context: Time constrain Event: Failure to complete tasks due to poor time management and/or unexpected events	2	2	
Context: Unclear scope of project Event: Unclear user requirements	1	3	
Context: Wrong logic in state machines Event: Wrong behaviour of the system	1	1	

TABLE 7: RISK LIST

Likelihood	Consequences of impact			
	1	2	3	4
4	4	8	12	16
3	3	6	9	12
2	2	4	6	8
1	1	2	3	4

TABLE 8: QUALITATIVE RISK ANALYSIS MATRIX

C. Software

Colour Sensor

```c
 * colour_sensor.c
 * Hardware Device:TCS3472.c
 *
 *  Created on: 26 May 2023
 *      Author: Boris Ivanov
 */

#include "level_3/colour_sensor.h"

int red = 0;
int yellow = 0;
int rgb_error = 0;

TCS3472 TCS3472_Create(uint8_t addr, I2C_HandleTypeDef *handle)
/* Structure with the address and I2C handle of the RGB Sensor */
{
    //create structure of RBG sensor data
    TCS3472 sensor_data =
    { addr, handle };
    return sensor_data;
}

void rgb_init(const TCS3472 *const self)
{
    // command to turn on the device [0x03] sent to register [0x80]
    rgb_send(self, RGB_COMMAND_REG | RGB_REG_ENABLE,
            RGB_ENABLE_PON | RGB_ENABLE_AEN);

    HAL_Delay(2);

    // Set timing register (ADC integration time)
    // Integration time can be set in steps of 2.4 ms.
    // 0xFF = 2.4 ms
    // 0x00 = 700 ms
    // 0xEE = 238; (256 - 238) * 2.4 = 43.2 ms

    //  ATIME = 0xEE; sent to TIMING register [0x81]
    rgb_send(self, RGB_COMMAND_REG | RGB_REG_TIMING, 0xEE);
    HAL_Delay(10);
    send_msg((uint8_t*) "\r!RGB-Sensor Initialised!\n\r");

}

int rgb_read_sensor(const TCS3472 *const self)
/* rgb_read_sensor: Function which interprets the result from RGB sensor
 * legacy code
 * @param1 self: An address to the structure of the device
 *
 * RETURNS:
 * robotCoin = 0 --> The token is yellow
 * robotCoin = 1 --> The token is red
 * robotCoin = 3 --> The token is not present
 */
{
    struct Color sens_RGBOut;
    int robotCoin = -2;         // arbitrary value to enter while loop
    int t_it_RGB = 0;           // integration time ? interrupt time ? Can't figure out name
    uint16_t hue = 0;

    while (robotCoin == -2) // loop until any return is reached
    {
        sens_RGBOut = queryRGBSensor(self); // store the structure of colours values from sens

        // get hue value (dominant wavelength) out of RGB sensor readings
        hue = getHue(sens_RGBOut.r, sens_RGBOut.g, sens_RGBOut.b);

        if (hue >= 35 && hue <= 75)
        {
            robotCoin = 0;
            yellow++;
        }
        if (hue >= 340 && hue <= 360)
        {
            robotCoin = 1;
            red++;
        }

        t_it_RGB++;
        HAL_Delay(2);
        if (t_it_RGB > 24)
        {
            // no token present
            robotCoin = 3;
            rgb_error++;
        }
    }
    return robotCoin;
}
```

```c
void rgb_send(const TCS3472 *const self, uint8_t regAddress, uint8_t data)
/*  rgb_send: Function which sends commands to the RGB sensor
 *
 * @param1 self: the structure which holds information about the RGB sensor
 * @param2 regAddress: the register to which the data will be written to
 * @param3 data: the data to be written in the register
 */
{
    i2c_Transmit(self->handle, self->dev_addr, regAddress, 1, &data, 1);
}

struct Color queryRGBSensor(const TCS3472 *const self)
/*
 * queryRGBSensor: retrieve information from sensor
 * Retrieves 16-bit number for red, green and blue
 *
 * @param1 self: An address to the structure of the device
 *
 * Returns:
 *  Colour struct containing r, g and b
 */
{

    /* Read RGB values */
    uint8_t low = 0; // temporary
    uint8_t high = 0; // temporary
    HAL_StatusTypeDef dev_Status;
    struct Color color;

    // Read red value; LOW and HIGH channels
    dev_Status = i2c_Receive(self->handle, self->dev_addr,
            RGB_COMMAND_REG | RGB_RED_LOW, 1, &low, sizeof(low));
    dev_Status = i2c_Receive(self->handle, self->dev_addr,
            RGB_COMMAND_REG | RGB_RED_HIGH, 1, &high, sizeof(high));
    if (dev_Status != HAL_OK) // if device is not OK
    {
        color.r = 0; //output 0
    }
    else
    {
        color.r = (high << 8) | low; // combine both local values into the output struct
    }

    // Read green value; LOW and HIGH channels
    dev_Status = i2c_Receive(self->handle, self->dev_addr,
            RGB_COMMAND_REG | RGB_GREEN_LOW, 1, &low, sizeof(low));
    dev_Status = i2c_Receive(self->handle, self->dev_addr,
            RGB_COMMAND_REG | RGB_GREEN_HIGH, 1, &high, sizeof(high));
    if (dev_Status != HAL_OK) // if device is not OK
    {
        color.g = 0; //output 0
    }
    else
    {
        color.g = (high << 8) | low; // combine both local values into the output struct
    }

    // Read blue value; LOW and HIGH channels
    dev_Status = i2c_Receive(self->handle, self->dev_addr,
            RGB_COMMAND_REG | RGB_BLUE_LOW, 1, &low, sizeof(low));
    dev_Status = i2c_Receive(self->handle, self->dev_addr,
            RGB_COMMAND_REG | RGB_BLUE_HIGH, 1, &high, sizeof(high));
    if (dev_Status != HAL_OK) // if device is not OK
    {
        color.b = 0; //output 0
    }
    else
    {
        color.b = (high << 8) | low; // combine both local values into the output struct
    }
    return color;
}
```

Token Colour Separator

```c
/*
 * token_colour_separator.c
 *
 *  Created on: Apr 24, 2023
 *      Author: Boris Ivanov
 */

#include "level_2/token_colour_separator.h"

int separate_tokens(const TCS3472 *const self)
/* separate_tokens: Function which separates tokens to robot/user storage
 *
 * Shoots yellow token, or gives signal to move red token
 *
 */
{
    int value_rgb = 0;

    value_rgb = rgb_read_sensor(self); // store sensor readings

    switch (value_rgb)
    {
    case 0:     // yellow token
        send_msg((uint8_t*)"\rThe colour of the token is YELLOW!\n\r");

        /* uncomment when FLIPPER is fixed
        set_Flipper();
        HAL_Delay(5);
        reset_Flipper();
        break;
        */
        break;

    case 1:     // red token
        send_msg((uint8_t*)"\rThe colour of the token is RED!\n\r");
        break;

    case 3:     // no token present
        send_msg((uint8_t*)"\rNo token is present!\n\r");
        break;

    default:
        send_msg((uint8_t*)"\rDEF:No token is present!\n\r");
        break;
    }
    return value_rgb;
}
```

Init CM4

```c
/*
 * init_CM4.c
 *
 *  Created on: Apr 24, 2023
 *      Author: Boris Ivanov
 */

/* Includes ------------------------------------------------------------------*/

#include "level_2/init_CM4.h"


void start_PWM(void)
/* start_PWM: Function which initialises the PWM TIMERs 2,13,14. */
{
    HAL_TIM_PWM_Start(&htim13, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim14, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
}

void init_Movement(void)
/* start_Movement: Function which initialises the motors and servos */
{
    initMotors();
    set_Enable_Power();
    start_PWM();
}

void init_MX_init(void)
/* start_MX_init: Function which initialises the STM32H peripherals */
{
    MX_ADC1_Init();
    MX_I2C1_Init();
    MX_TIM2_Init();
    MX_TIM3_Init();
    MX_TIM4_Init();
    MX_TIM7_Init();
    MX_TIM13_Init();
    MX_TIM14_Init();

    HAL_TIM_Base_Start_IT(&htim7); // activate interrupt for TIM7
}

void init_Start_Up(void)
/* init_Start_Up: Function to initialise the whole system and set inital values to PWMs
 *  The microcontroller peripherals. The Motors and Servos.
 *  Homes the servos.
 *
 *  TODO: Add output to each init func, to catch errors
 */
{
    send_msg((uint8_t*)"\r!Initialising Micro-controller Signals!\n\r");
    init_MX_init();
    HAL_Delay(50);

    send_msg((uint8_t*)"\r!Initialising movement signals!\n\r");
    init_Movement();
    HAL_Delay(50);

    send_msg((uint8_t*)"\rHoming Motors\n\r");
    HAL_Delay(50);

    HomeMotors(1, 1); // homing motors
    HAL_Delay(50);

    send_msg((uint8_t*)"\r########Motors Homed########\n\r");
    HAL_Delay(50);

    send_msg((uint8_t*)"\rHoming servos\n\r");
    HAL_Delay(50);

    set_Slide_Servo(SLIDE_CLOSED);
    set_Rotate_Servo(ROTATE_NEUTRAL);
    send_msg((uint8_t*)"\r########Servos Homed########\n\r");
    HAL_Delay(50);

}
```

Motor X

```
 *
 *  Created on: Jun 20, 2022
 *      Author: Pascal
 *
 *  Edited on: 11 May, 2023
 *      Author: Boris Ivanov
 */
#include "level_3/motor_x.h"

/* TODO: Move to Variables.c */
uint32_t counterX;
double position_mm_X;
int msg_counter_x = 0;                                  // Counter limit for messages

uint8_t i_X = 0;
uint8_t once_X = 1;

void initMotorX()
/* Initialises the necessary timers for motor X */
{
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
    HAL_TIM_Encoder_Start_IT(&htim3, TIM_CHANNEL_ALL);
}

uint8_t homeMotorX()
/* homeMotorX: Moves the motor to the home position */
{

    __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_3, 130);   // set PWM of motor

    set_Direction_X();                                  // counter-clockwise  | towards HOME TODO:Change
name
    set_Ready_X();                                      // enables motor X TODO:Change name

    while (!get_Homing_X())
    {

    }
    __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_3, 0);     // set PWM of motor
    reset_Ready_X();                                    // disables motor
    HAL_Delay(1000);
    /* Initialise variables */
    position_mm_X = 0.0;
    counterX = 0;
    i_X = 0;
    __HAL_TIM_SET_COUNTER(&htim3, 0);                   // reset timer
    return 1;
}

uint8_t move_to_posX(double posX)
/* move_to_posX: Moves the motor to a position X cm away from the home position
 *
 * @param posX: Centimetres away from the home position
 *  RANGE posX:  TODO: Determine it
 */
{
    counterX = __HAL_TIM_GET_COUNTER(&htim3);                   // get timer value
    position_mm_X = (double) ((counterX / 3855) + (i_X * 17));  // get position
    double delta = posX - position_mm_X;                        // calculate delta
    __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_3, 130);           // adjust speed (prev val 140) 120 is a
bit slow, but safe speed for not crashing during testing

    set_Ready_X();                                              // enable motor
    while (abs(delta) > 1.5)                                        // accuracy of movement TODO: prove
it
    {
        if (delta > 0){
            reset_Direction_X();}                               // clockwise | towards END
        else if (delta < 0){
            set_Direction_X();}                                 // counter-clockwise | towards HOME
        else{
            break;}

        counterX = __HAL_TIM_GET_COUNTER(&htim3);               // update counter
        /* Some Magic */
        if (counterX > 61680 && delta > 0 && once_X){
            once_X = 0;}
        else if (counterX < 3855 && delta > 0 && !once_X){
            once_X = 1, i_X += 1;}
        else if (counterX < 3855 && delta < 0){
            once_X = 0;}
        else if (counterX > 61680 && delta < 0 && !once_X){
            once_X = 1, i_X -= 1;}
        /* End of Magic */

        position_mm_X = (double) ((counterX / 3855) + (i_X * 17));  // update position
        delta = posX - position_mm_X;                               // update delta

        // limit msg sending every Nth operation
        if (msg_counter_x % 200000 == 0)                            // TODO: move 10 to a define freq of
msging
        {
            send_msg_data((uint8_t*)"\rCurrently @ Pos X: %d\n\r", (int)position_mm_X);
        }
        msg_counter_x++;

    }
    __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_3, 0);                 // set PWM to 0
    reset_Ready_X();
    return 1;
}
```

Motor Controller

```c
/*
 * motor_master.c
 *
 *  Created on: Jun 20, 2022
 *      Author: Pascal
 */
#include "level_3/motor_master.h"

void initMotors()
{
    initMotorX();
    initMotorZ();
}

uint8_t move_to_X_and_Z(int16_t posX, int16_t posZ)
{
    move_to_posX(posX);
    send_msg((uint8_t*)"\r**** Motor X is at the final position! ****\n\r");

    move_to_posZ(posZ);
    send_msg((uint8_t*)"\r**** Motor Z is at the final position! ****\n\r");

    return 1;
}

uint8_t HomeMotors(uint8_t homeX, uint8_t homeZ)
{
    send_msg((uint8_t*)"\r#### HOMING Motor X №###\n\r");
    if (homeX)
    {
        homeMotorX();
    }
    send_msg((uint8_t*)"\r######## Motor X is at HOME ########\n\r");

    HAL_Delay(250);
    send_msg((uint8_t*)"\r#### HOMING Motor Z ####\n\r");
    if (homeZ)
    {
        homeMotorZ();
    }
    send_msg((uint8_t*)"\r######## Motor Z is at HOME ########\n\r");
    return 1;
}
/*
 * To see how the code bellow function
 * simply paste it into the main.c file of CM4
 * and run it.
 */

// Fully working set up to get the arms to the top
// and simulate dropping down a token in each column
/*...
 */

// Fully worked out "go to dump base" and lower to pick up tokens
/*...
 */

// Fully worked out "go above flipper" and lower the token
/*...
 */

// Fully worked out "go to store location" and lower into each one
/* ...
 */
```

User Detector

```c
/*
 * user_detector.c
 *  hardware device: PCA9554
 *
 *  Created on: Apr 24, 2023
 *      Author: Boris Ivanov
 */

#include "level_2/user_detector.h"

int coinInsertDetection(void)
/**
 * coinInsertDetection: check with light gates if coin is inserted
 *
 * returns:
 *  0: no move
 *  1-7: inserted into column n
 *  9: error
 */
{
    int output = 0;

    /* Query the processor with the I2C connection for the light gates */
    int gateNew = queryLightGate();

    if (gateNew == 9)
    {
        detect_error++;
        return 9;
    }

    /* Query the logic: if the same as the prior gate, not debounced yet.  */
    if (gateNew == -1 && gateOld <= 7 && gateNew != gateOld)
    {
        output = gateOld;
    }

    gateOld = gateNew;

    return output;
}

int whatStackToFill(void)
/**
 * whatStackToFill: determine which storage stack to fill
 *
 * Always start filling the first stack, then the second, finally the last.
 * Each stack can hold max. 7 chips
 *
 * Returns:
 *  -1: All stacks are full
 *  0-2: Number of stack
 */
{
    int stack = 2;
    for (int i = 0; i < 3; i++) // loop through all options
    {
        if (mem_StorageStack[i] < 7) // assert if not filled
        {
            stack = i; // if not filled, use this stack
            break;
        }
    }
    return stack;
}

int whatStackToEmpty(void)
/**
 * whatStackToEmpty: determine which storage stack to empty
 *
 * Always start emptying the last stack, then the second, finally the first.
 *
 * Returns:
 *  -1: All stacks are empty
 *  0-2: Number of stack
 */
{
    int stack = 2;
    for (int i = 2; i >= 0; i--) // loop through all options
    {
        if (mem_StorageStack[i] > 0) // assert if not empty
        {
            stack = i; // if not empty, use this stack
            break;
        }
    }
    return stack;
}

void init_coinDetector(void)
{
    HAL_StatusTypeDef dev_Status;
    uint8_t data = 0xFF; // ensures all ports are INPUT configured

    // Program command byte:
    // configuration register [0x03] sets all ports as inputs (1)
    dev_Status = i2c_Transmit(&hi2c1, CD_ADD, 0x03, 1, &data, 1);

    if (dev_Status != HAL_OK) // if device is not OK
    {
        send_msg((uint8_t*) "\r¡User-Detector Initialisation FAILED¡\n\r");
        return;
    }
    else
    {
        send_msg((uint8_t*) "\r!User-Detector Initialised!\n\r");
    }
}
```

```c
int queryLightGate(void)
/**
 * queryLightGate: read coin detector value.
 *
 * Reads coin detector using I2C. Returns column where something is present.
 *60000-1
 * Returns:
 * -2: Error from PCA9554
 * -1: All gates are free
 * 1-7: Stack where gate is blocked
 * 9: More than one gate is blocked
 */
{
    uint8_t cd;

    HAL_StatusTypeDef dev_Status;
    /* Retrieve input data from
     * Single Registers:
     0100 0001 - 0x41
     0100 0011 - 0x43
     0100 0101 - 0x45
     0100 0111 - 0x47
     0100 1001 - 0x49
     0100 1011 - 0x4B
     0100 1101 - 0x4D
     0100 1111 - 0x4F
     */
    // Program command byte: Reading the input port [0x00] register
    dev_Status = i2c_Receive(&hi2c1, CD_ADD, 0x00, 1, &cd, sizeof(cd));

    if (dev_Status != HAL_OK)
    {
        return -2;
    }
    else
    {
        /* Remove LSB: is always 1 */
        cd &= ~0x01;

        /* Check if empty: */
        if (!cd)
            return -1;

        if (cd == 0x02)
        {
            return 7;
        }
        if (cd == 0x04)
        {
            return 6;
        }
        if (cd == 0x08)
        {
            return 5;
        }
        if (cd == 0x10)
        {
            return 3;   // flipped in hardware
        }
        if (cd == 0x20)
        {
            return 4;   // flipped in hardware
        }
        if (cd == 0x40)
        {
            return 1;   // flipped in hardware
        }
        if (cd == 0x80)
        {
            return 2;   // flipped in hardware
        }

        //More than one coin
        if ((cd >> 1) & 1)
        {
            mem_Board[7 - 1]++;
        }
        if ((cd >> 2) & 1)
        {
            mem_Board[6 - 1]++;
        }
        if ((cd >> 3) & 1)
        {
            mem_Board[5 - 1]++;
        }
        if ((cd >> 4) & 1)
        {
            mem_Board[3 - 1]++;
        }
        if ((cd >> 5) & 1)
        {
            mem_Board[4 - 1]++;
        }
        if ((cd >> 6) & 1)
        {
            mem_Board[1 - 1]++;
        }
        if ((cd >> 7) & 1)
        {
            mem_Board[2 - 1]++;
        }
        sens = 0;
        return 9;
    }
}
```

Servo Controller

```c
/*
 * servo_controller.c
 *
 *  Created on: Apr 24, 2023
 *      Author: Boris Ivanov
 */

#include "level_3/servo_controller.h"

uint32_t t_pulse_rotate = 0;
uint32_t t_pulse_slide = 0;
uint32_t CCR_value_rotate = 0;
uint32_t CCR_value_slide = 0;

void set_Rotate_Servo(float angle)
/* set_Rotate_Servo: Sets the angle of the servo controlling the end-effector rotator
 *
 * @param angle: Angle of rotation in degrees
 * Acceptable values between 1 - 23 degrees
 */
{
    t_pulse_rotate = 10 * ((uint32_t) angle) + CONTROL_PULSE_MIN;   // from data sheet of Parallax
Standard Servo
    CCR_value_rotate = (t_pulse_rotate - CONTROL_PULSE_MIN)
            * (TIMER_COUNTER_PERIOD) / (CONTROL_PULSE_RANGE);       // mapping the values of us to the
range of Timer Counter
    TIM13->CCR1 = CCR_value_rotate;                                 // applying value to TIM
}
void set_Slide_Servo(float angle)
/* set_Slide_Servo: Sets the angle of the servo controlling the board opening slider
 *
 * @param angle: Angle of rotation in degrees
 * Acceptable values between 1 - 25 degrees
 */
{
    t_pulse_slide = 10 * ((uint32_t) angle) + CONTROL_PULSE_MIN;    // from data sheet of Parallax
Standard Servo
    CCR_value_slide = (t_pulse_slide - CONTROL_PULSE_MIN)
            * (TIMER_COUNTER_PERIOD) / (CONTROL_PULSE_RANGE);       // mapping the values of us to the
range of Timer Counter
    TIM14->CCR1 = CCR_value_slide;                                  // applying value to TIM
}

// Fully programmed positions of board opener and end-effactor rotator
/*
        // THIS order of opening has to be preserved
        send_msg((uint8_t*)"\rOpening Column 2\n\r");
        set_Slide_Servo(OPEN_COL_2);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rOpening Column 6\n\r");
        set_Slide_Servo(OPEN_COL_6);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rOpening Column 7\n\r");
        set_Slide_Servo(OPEN_COL_7);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rOpening Column 4\n\r");
        set_Slide_Servo(OPEN_COL_4);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rOpening Column 1\n\r");
        set_Slide_Servo(OPEN_COL_1);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rOpening Column 5\n\r");
        set_Slide_Servo(OPEN_COL_5);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rOpening Column 3\n\r");
        set_Slide_Servo(OPEN_COL_3);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rOpening board fully\n\r");
        set_Slide_Servo(SLIDE_OPEN);
        HAL_Delay(1000);

        send_msg((uint8_t*)"\rClosing board fully\n\r");
        set_Slide_Servo(SLIDE_CLOSED);
        HAL_Delay(1000);

        set_Rotate_Servo(ROTATE_NEUTRAL);
        HAL_Delay(1000);
        set_Rotate_Servo(ROTATE_TO_STACK);
        HAL_Delay(1000);
        set_Rotate_Servo(ROTATE_TO_DROP);
        HAL_Delay(1000);
        set_Rotate_Servo(ROTATE_NEUTRAL);
 */
```

D. TRD, Test Report Document

The tests in this document are grouped by the level in accordance with the software architecture designed for the Connect-4 robot. The tests for Low-Level (peripherals of STM32H) unit and component testing were omitted from this plan due to time limitations. However, a basic draft was created for future reference.

Functionality Tests

Test Cases	Test Conditions	Done
Level 2 – Cortex-M4		
Initialization	**1.** Verify that the initialization sets up the system to be ready for operation, including configuring the peripherals, initializing the modules.	1.
Task Manager	1. Test the initialization and configuration of the module. 1.1. Test if the module can read the tasks from memory and trigger the state transition. 2. Verify that the task manager can detect, report, and recover from errors.	1. 1.1 2.
Motor Controller	**1.** Test the initialization and configuration of the module and both motors. 1.1. Verify that the module is set up to receive signals from the encoders, and home/end switches. 1.2. Test if the module moves the motors in both X and Z directions. 1.3. Test if the PWM signal controls the motors effectively 1.4. Test if the module can read the position from the encoders. 1.5. Test that the home/end-switches send the correct interrupt and stop the motor. **2.** Verify that the Motor Controller module can detect, report, and recover from errors.	1. 1.1 1.2 1.3 1.4 1.5 2.
Token colour separator controller	**1.** Test the initialization and configuration of the module. 1.1. Test the module is correctly set up to control the RGB and proximity sensor and the flipper. 1.2. Test if the module detects the colour of tokens (red and yellow). 1.3. Test if the module detects the proximity of the token.	1. 1.1 1.2 1.3 1.4

	1.4. Test the activation of the flipper. **2.** Verify that the Token Colour Separator Controller module can detect, report, and recover from errors.	2.
Token picker controller	**1.** Test the initialization and configuration of the controller. 1.1. Test if the controller can move the end-effector servo, read the vacuum sensor, and control the vacuum valve. 1.2. Test if the vacuum pump generates enough pressure to pick up a token and transport it. 1.3. Test if the positions of all different pick-up/drop-off points are correct. **2.** Verify that the module can detect, report, and recover from errors.	1. 1.1 1.2 1.3 2. 3.
User Detector	**1.** Test the initialization and configuration of the module. 1.1. Verify that the module can read data from the light-gate circuit. **2.** Verify that the module can detect, report, and recover from errors.	1. 1.1 2.
Board Opener	**1.** Test the initialization and configuration of the module. 1.1. Test that the servo motors can open the board column by column. 1.2. Test that the Task Manager can send commands for opening and closing the board. **2.** Verify that the Board Opener module can detect, report, and recover from any errors.	1. 1.1 1.2 2.
Level 2 – Cortex-M7		
Initialization	**1.** Verify that the initialization sets up the system to be ready for operation, including configuring the peripherals, initializing the modules.	1.
Game controller	**1.** Verify that the Game Controller module is correctly set up to manage the overall game logic and flow. 1.1. Test the module's ability to keep and update the game state, including the board state and player turns. 1.2. Test the module's ability to detect win, loss, or draw conditions. 1.3. Test the state transitions of the controller. **2.** Verify that the Game Controller can detect, report, and recover from any errors.	1. 1.1 1.2 1.3 2.
CM4 Task Generator	**1.** Test the initialization and configuration of the module. 1.1. Test the CM4 Task Generator's ability to create tasks based on the game state and requests from other modules. 1.2. Verify that the CM4 Task Generator module receives correct game state updates and next-move decisions from the Game Controller module.	1. 1.1 1.2

	2. Verify that the CM4 Task Generator module can detect, report, and recover from any errors	2.
Game end block	**1.** Test the initialization and configuration of the module. 1.1. Test if the module is able to and handle a win/lose/draw condition for either the human player or the robot player. **2.** Verify that the module can detect, report, and recover from any errors.	1. 1.1 2.
UART controller	**1.** Test the initialization and configuration of the UART Controller module. 1.1. Test the UART's ability to transmit/receive data to and from external blocks. 1.2. Test the debug environment created through UART * **2.** Verify that the UART controller can detect, report, and recover from any errors.	1. 1.1 1.2 2.

*If created and discussed that it is reasonable to do so.

System Timing

It is important for future improvements of the system and separate modules to know the timing of specific operations and action. It would be wise for them to be divided into the same structure as the one set up in the above tests and the architecture. This would be more useful for the Cortex-M4 at the moment, and therefore I have compiled a brief list of timings that would be useful to know.

- Record how long picking the tokens takes.
- Record how long releasing the tokens takes.
- Record how fast a token dropping is recognized.
- Test reaction to multiple tokens insertion in the same column (cheat move).
- Test reaction to multiple tokens insertion in multiple columns (cheat move).
- Test reaction to when a token is inserted in the wrong player state (cheat move).
- Test the time needed to run the length of the axis of the X and Z motors **pre-PID** controller implementation (current system).
- Test the time needed to run the length of the axis of the X and Z motors **post-PID** controller implementation.
- Record how long it takes to clear the full board.

Happy-Path Test

This is a full test of the system that involves the whole gameplay loop as one would normally (without cheat moves) expect it to run.

This includes verifying that the robot moves the tokens accurately, the sensor (RGB, proxy, IR/PD) detect the correct positions when tokens are inserted and the correct colour when cleaning the board, the game logic (FSM) functions properly and transitions correctly based on the current state, and the

communication between the two cores. The standard rules of gameplay are applied when playing against the robot. The human player is not trying to cheat at any point.

1. Power on the Connect-4 robot and ensure all components are functioning properly.
2. Initiate a game session and confirm that the robot accurately detects and responds to player moves.
3. Place tokens in a strategic manner to create a winning condition for the robot.
4. Verify that the robot correctly identifies the winning condition and declares itself as the winner.
5. Reset the game board and repeat the process multiple times, testing the different win conditions, to ensure consistent and accurate performance.

Worst-Case Test

A worst-case test is designed to evaluate the performance and behavior of a system under challenging or extreme conditions. It aims to push the system to its limits or simulate scenarios that are expected to cause failures or suboptimal outcomes. This type of testing helps identify potential weaknesses, vulnerabilities, bugs and overall general unwanted behavior.

Several examples would be to simulate an already high-density tokens on the board and check how the AI of the robot responds and also if the entire system can analyze complex patterns. Another one could be to create a scenario where the winning condition could only be met on the last possible move of the complete set. Make tests that try and break the error handling that is to be implemented.