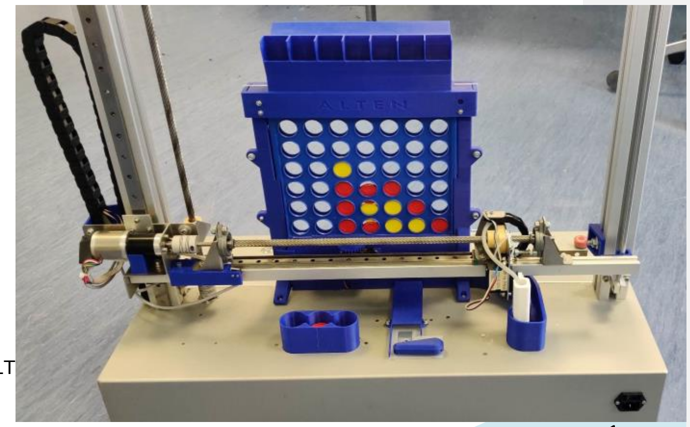
Graduation Report

Submitted to: Fontys University of Applied Sciences

June 2023

Boris Ivanov



Document data:

Author: Boris Ivanov
Student number: 2969300
E-mail: 357544@student.fontys.nl
Date: 16 May 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:

Signature:

Foreword

[PLACEHOLDER]

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 educational program Electrical & Electronic Engineering at Fontys University of Applied Sciences in Eindhoven. The project and this report were realized in the period of February 2023 – June 2023.

I was guided by my mentor Michael van der Velden.

[Continue]

Table of Contents

Summary

[written at the end]

List of abbreviations

[continually updated]

ACRONYM	DESCRIPTION
IT	Information Technology

List of figures & tables

[continually updated]

[continually updated]

I. Introduction

[written at a later stage]

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].

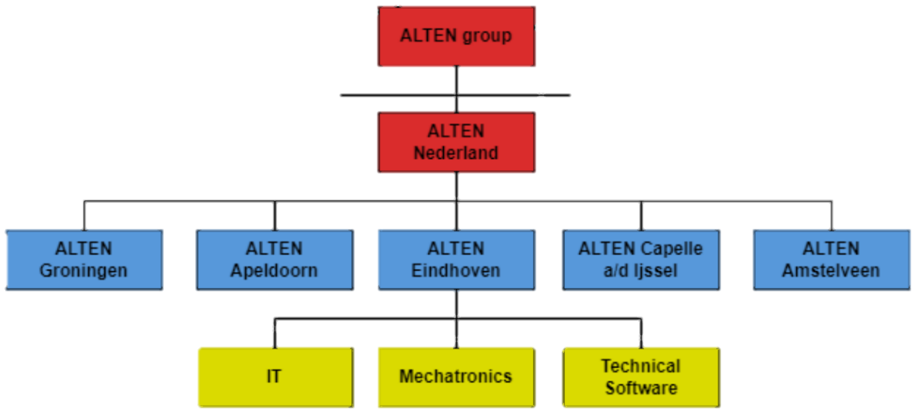


FIGURE 1: ORGANOGRAM

1. 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 you give all the details about the project assignment, in clear sections. The reader comes to know everything about the following areas:

- *What is the initial situation? What is there to miss? Why is that a problem? What are the unintended consequences?*
- *What is the purpose of the project? What does the client achieve with it? What is the desired end situation?*
- *What is, by virtue of the two previous points, the precise assignment description?*

In this chapter you clearly describe what does belong to and does not belong to your assignment. So if the company wants you to use a certain design method or apply a specific technique (FPGA, microcontroller, protocols, etc.) the reader can find it here.

If the assignment in the course of the project changes, this will be explained in this chapter.

So again:

A. *Successful problem definition, means clear goal of the project*
B. *Defining boundaries of the project. Boundaries are more conditions that must be met. E.g. if there is enough budget.*
C. *What is out of scope? For example, out of scope is: sw development is not part of the project.*
D. *Make sure that the final results/solutions could be verified if possibleda,*

1. Project background

My graduation internship for Fontys Hogeschool 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, 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 at later chapters.

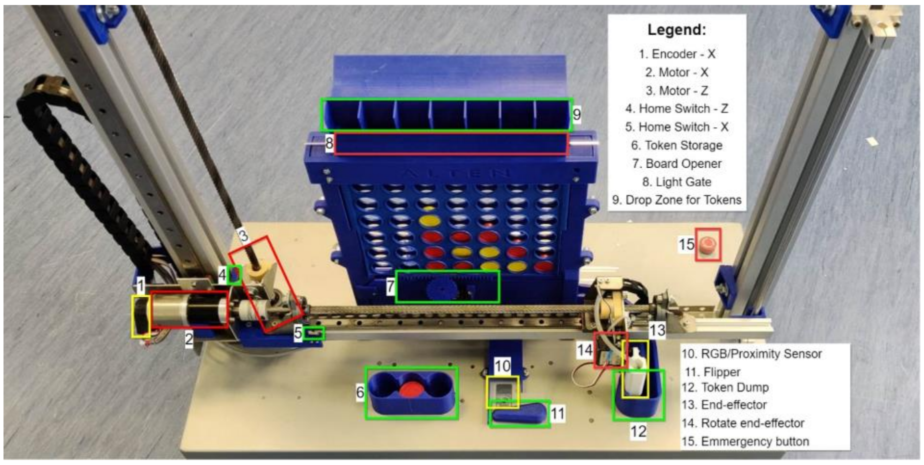


FIGURE 2: THE CONNECT-4 ROBOT

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.

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 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 IR sensors on the entrance of the board.

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.

3. Assignment

The assignment, therefore, is to merge old with the new and evaluate if the previous systems work as intended. Write additional code that supports the operation of the Connect-4 robot player. That would be any block from the newer core Cortex-M7 (described in Chapter VI.), and the main game logic. Additional tasks include, but not limited to: designing high-level logic for different system sub-modules from the previously designed architecture, designing libraries for sensors (RGB sensor, IR sensor) and peripherals (GPIO, motors, encoders, etc.), implementing low-level logic (EXTI, NVIC, HSEM, etc.).
To sum up, the task is to bring the robot to an operational level by designing and implementing the necessary elements and validating the previous work done.

Additionally, ethernet communication with the systems should be investigated. This would be the starting ground for future upgrades of the system. This would have to facilitate communication with the internet, the transfer and receiving of data to keep high scores, current player’s turn, a human machine interface, etc.

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.

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

Table 1: Project boundaries

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, there are several boundary conditions that 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 project at different points of completion is usually very time-consuming.
The hardware limitations of the robot player must be considered during the design and implementation of the software. The STM32H7 processor has a limited amount of memory, there are a lot of other hardware components with varied points of failure.

6. Project approach:

i. Development phases

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 Chapter VI) , through different means of testing (unit, module, integration) and a varied assortment of techniques (black-box, white-box, happy-path) and through the designing of newer modules that are to be integrated into the system.

ii. Verification method (V-model)

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.

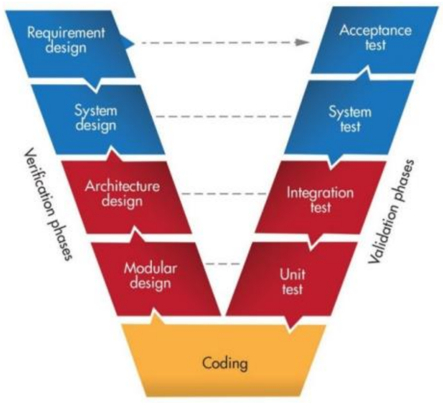


FIGURE 3: THE V-MODEL

IV. Research

1. Research objectives

- Validation of the previous software designs
- Architectural set-up
- State Machine Improvements

2. Research

The research needed for 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, how to test software on embedded systems and finally ethernet communication [if included].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. About more detailed information on the architecture, one may request access to the SAD (Software Architecture Document) [5]

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

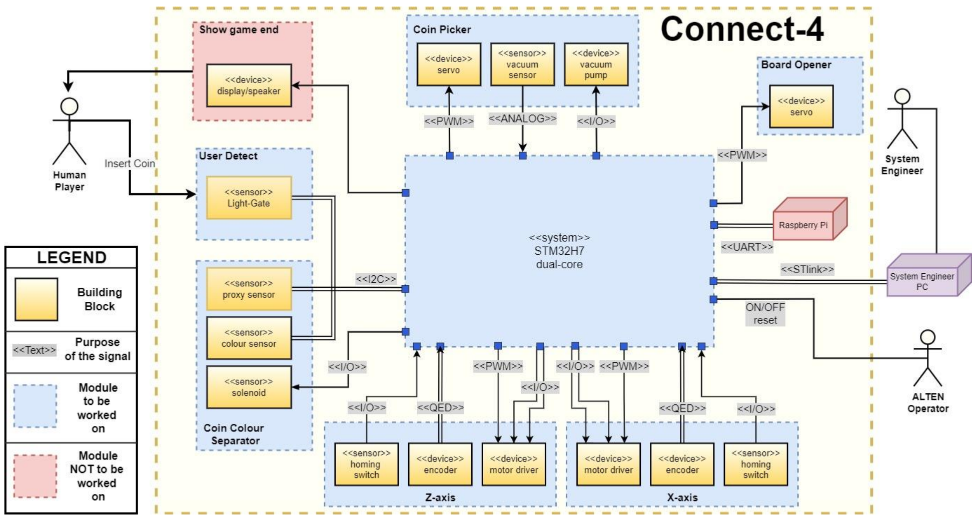


FIGURE 4: OVERVIEW OF THE ROBOT PLAYER

The previous designer chose to describe the system by several different levels of abstraction. Abstraction, in this case being how obstructed are the low-level controls of the peripherals and the registers of the microcontroller. 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, level 3 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. What can be seen in figure X is the overview of the system, with the different controlling methods required by each peripheral. Blue signifies that the module needs work upon. Red means that this is not implemented on the system and 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.

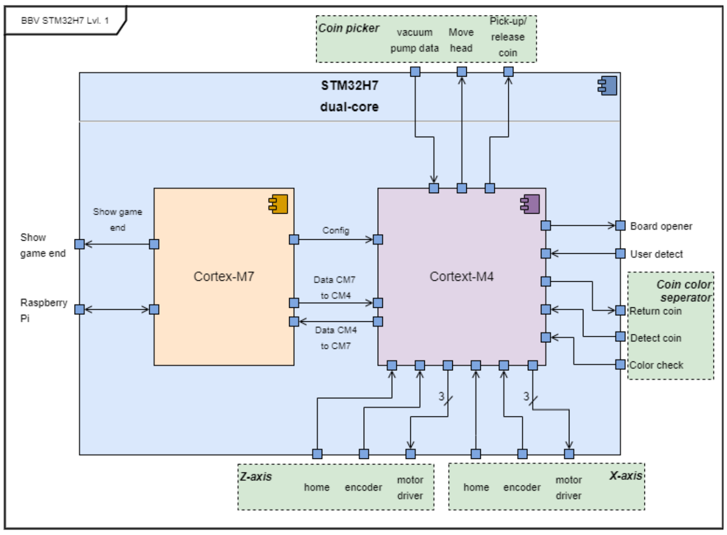


FIGURE 5: LEVEL 1 OF THE SOFTWARE ARCHITECTURE

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.

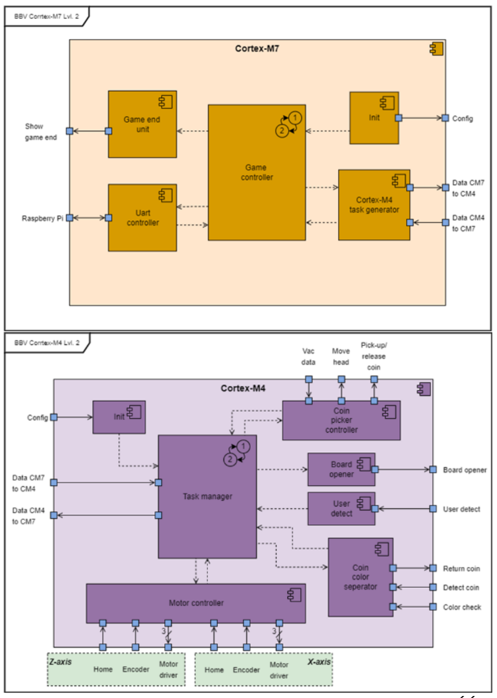


FIGURE 6: LEVEL 2 OF THE SOFTWARE ARCHITECTURE FOR BOTH CORES

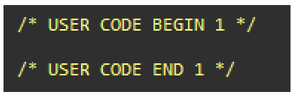
The second level is as deep as it goes for Cortex-M7, since the rest of the functionality is out of scope for this project and is for future upgrades. However this doesn't mean that there's no work to be done on this level. The game controller is the main finite state machine of the Connect-4 robot player, and together with the task generator, they are responsible for the whole gameplay loop. A part of the task for this assignment is to imporve 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 a place to place, a module to separates 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 architeture is the work of a previous assignment and further detail is saved due to brevety [5]. However, a part of this current assignment is to evaluate how good the architecture will be in practice, and during design.

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,



```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

FIGURE 7: USER CODE MARKING

and such a configuration file does not exist for the legacy code of 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 at is marked as “ USER CODE BEIGN/END”.
Secondly, the generated code and the programing of the microcontroller is 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 **Error! Reference source not found.** some o f them could be noticed, like the PWM and I2C.

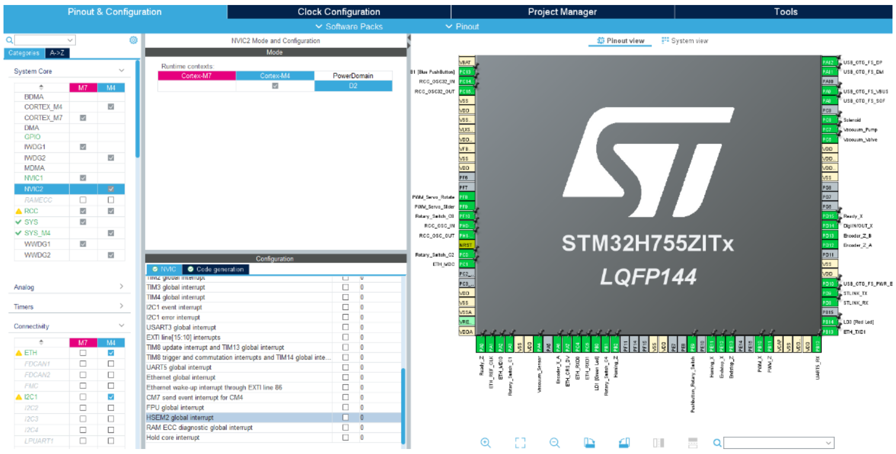
The full list of peripherals in use and their functions are as follows: [to copy acronyms to top]

- GPIO, General Purpose Input/Output Pins, which controls 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, which facilitates the connection with some sensors.
- UART, the Universal Asynchronous Receiver/Transmitter protocol, which 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, one is called “*Connect-4 Demo”*, which has code for the Task Generator based on the HSEM (Hardware Semaphores) notifications and the basic FSM on the game logic described in the SAD , and 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 previous intern after the start of the internship, and it is called the *“Initialization Demo”*. It is based on the “Connect-4 Demo” project, but doesn't include all its functionalities, however it adds the functionality to move the motors and to perform a homing routine. Furthermore, initially the “Initialization Demo” wasn't running on the system due to the problems described in, [include chapter], and when those issues were overcome the code still didn't run on the system. Through more research, it was concluded that a lot of the logic in the project was wrong and had to be modified.
Due to lack of documentation, most of the logic of the projects was hard to follow, but since success has been achieved with the demo projects, it was important to learn from them and see what is available for use. 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 on the hardware as it stands right now.

When inspecting the “Initialization Demo”, the configuration file of the micro-controller opens as well. Here the developer can configure all the features that the STM32H755 has to offer.

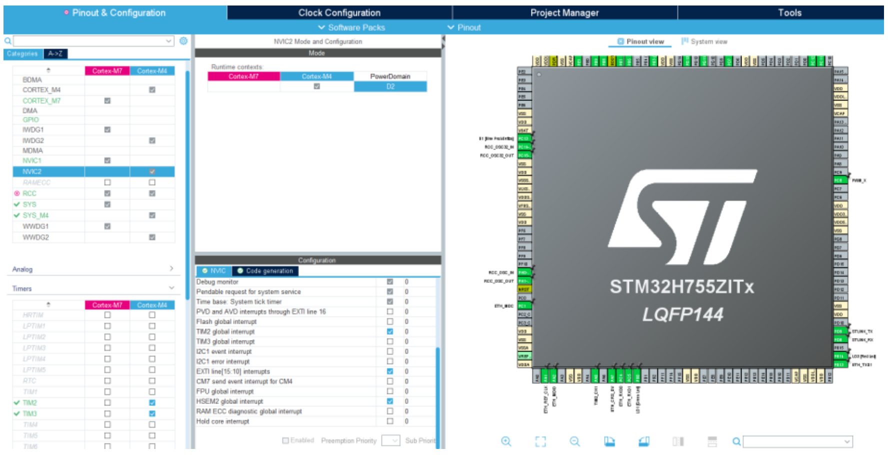


TIM2 global interrupt	☐	0
TIM3 global interrupt	☐	0
TIM4 global interrupt	☐	0
I2C1 event interrupt	☐	0
I2C1 error interrupt	☐	0
USART3 global interrupt	☐	0
EXTI line[15:10] interrupts	☐	0
TIM8 update interrupt and TIM13 global interrupt	☐	0
TIM8 trigger and commutation interrupts and TIM14 global inte...	☐	0
UART5 global interrupt	☐	0
Ethernet global interrupt	☐	0
Ethernet wake-up interrupt through EXTI line 86	☐	0
CM7 send event interrupt for CM4	☐	0
FPU global interrupt	☐	0
HSEM2 global interrupt	☐	0
RAM ECC diagnostic global interrupt	☐	0
Hold core interrupt	☐	0

On the left side is the pinout view, with user input names for the functions of different pins. In the middle

Figure 8: The .IOC Configuration File of Latest Implementation

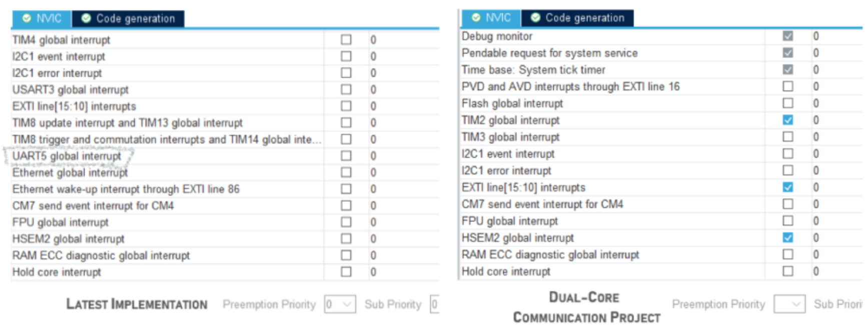
are the different options each feature has. And on the right side, are the categories themselves. As evident, there is a lot to go through, and even more so, because several projects are involved. Here comes the first challenge of the merging several projects.
If one is to look at a project of different functionality, for example how the “Dual-Core communication” is constructed, we can notice several differences with the configuration from figure X.



Debug monitor	✓	0
Pendable request for system service	✓	0
Time base: System tick timer	✓	0
PVD and AVD interrupts through EXTI line 16	☐	0
Flash global interrupt	☐	0
TIM2 global interrupt	✓	0
TIM3 global interrupt	☐	0
I2C1 event interrupt	☐	0
I2C1 error interrupt	☐	0
EXTI line[15:10] interrupts	✓	0
CM7 send event interrupt for CM4	☐	0
FPU global interrupt	☐	0
HSEM2 global interrupt	✓	0
RAM ECC diagnostic global interrupt	☐	0
Hold core interrupt	☐	0

Figure 9: The .IOC file of a Dual-Core Communication Project

First off, the pinout is completely different, but for now this isn't important since the project in figure 5 is the one that includes the most up to date layout of the pins. However, if we were to focus on the configuration settings of the separate features, it can be noticed that both the presence of some settings and their enabling is different.



TIM4 global interrupt	☐	0
I2C1 event interrupt	☐	0
I2C1 error interrupt	☐	0
USART3 global interrupt	☐	0
EXTI line[15:10] interrupts	☐	0
TIM8 update interrupt and TIM13 global interrupt	☐	0
TIM8 trigger and commutation interrupts and TIM14 global inte...	☐	0
UART5 global interrupt	☐	0
Ethernet global interrupt	☐	0
Ethernet wake-up interrupt through EXTI line 86	☐	0
CM7 send event interrupt for CM4	☐	0
FPU global interrupt	☐	0
HSEM2 global interrupt	☐	0
RAM ECC diagnostic global interrupt	☐	0
Hold core interrupt	☐	0

LATEST IMPLEMENTATION

Debug monitor	✓	0
Pendable request for system service	✓	0
Time base: System tick timer	✓	0
PVD and AVD interrupts through EXTI line 16	☐	0
Flash global interrupt	☐	0
TIM2 global interrupt	✓	0
TIM3 global interrupt	☐	0
I2C1 event interrupt	☐	0
I2C1 error interrupt	☐	0
EXTI line[15:10] interrupts	✓	0
CM7 send event interrupt for CM4	☐	0
FPU global interrupt	☐	0
HSEM2 global interrupt	✓	0
RAM ECC diagnostic global interrupt	☐	0
Hold core interrupt	☐	0

DUAL-CORE COMMUNICATION PROJECT

FIGURE 10: COMPARISON OF SETTINGS ACROSS PROJECTS

The presence of options was later linked to their enabling in different sections of the configuration. For example, if *UART5* was never configured to begin with (as is the case in the dual-core project), you will never see the option for a *UART5 global interrupt* in the NVIC settings.
Next, it's observed that there are features existing in both projects, that are enabled on one, but not the other. For example, look at the HSEM and EXTI line interrupts. This is a bit more impactful to the merging of the demo projects into a single one, since the lack of their presence means that the auto-generated code for them won't exist. And with it, the place where the developer can implement their required functions. Which in turn will negate the added code, if a new auto-generation is needed, which is mandatory every time a feature of the controller is configured or modified. As mentioned, everything in the generated files that resided outside of the aforementioned "USER CODE" places, will disappear.
For the current implementation this means the following: Since it will cost too much time to manually scrape through all the different options, when a feature is found not present in the current project, but is in a demonstration project, only then will it be investigated where and how to configure it, and then add the necessary code for it.

Here, I believe it is appropriate to mention that the latest software project before my own implementation was received 7 weeks after the internship started. This was an unforeseen delay in communication from the previous intern, who also hadn't uploaded the software in question in the intranet of ALTEN. Which resulted in spending more time on developing tests for the system, which are to be described in a later chapter [include chapter] and researching the previous projects more, the results from which are described in [add chapter].

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 very simple 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. How does one core know that the other has written the data to memory ? How does the other core know when to look in the memory ?

A hardware semaphore is a synchronization primitive, 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 basic 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 [7]. This would be where the data for which column to play at, or at which column the user has dropped the token, would have to happen.
Additionally, in the "Connect-4 Demo" it is shown how the HSEMs are used for 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. 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 notification with status updates. One such notification "task" exists as an example in one of the projects. The rest have to still be designed.

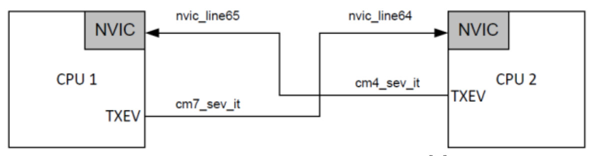


FIGURE 11: SEND-EVENT INSTRUCTION NOTIFICATION MECHANISM [8]

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, it is ensured that only one core has access to a peripheral at any given time. Additionally, to expand the current tasks and give a clear meaning to their functions. Add tasks which describe the system in a fuller extent. Implement the design.

3. Results

4. Conclusions

V1 Specification

ID	Requirement	Explanation	Priority
UR.1	The user should be able to start and play a game of Connect-4 against the robotized opponent without operator intervention.	The system should be fully automated, allowing the user to start and play a game of Connect-4 against the robot without any human intervention. The robot should be able to detect the user's moves and respond accordingly with its own moves.	Must
UR.2	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.3	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. Or one who tries to input a wrong object.	Must
UR.4	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.5	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.6	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
UR6.1	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
UR6.2	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
UR6.3	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	Must

Commented [IB1]: A could because such a system doesn't exist as of now. No display, buzzer. A special sequence could be made and a new token could be 3D-printed, alongside maybe a special place to signal to the user that the game has ended.
But it felt a bit less important than the other features.

		automated, and the user does not need to manually retrieve the tokens.	
UR.7	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.8	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.9	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.10	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
UR11	ETHERNET		

VI System Design

Here you can e.g. introduce or start applying and the top-down structured design of your project.

A. *Can be described in a SDD (System Design Document) and attached to the report*
B. *In this chapter you describe the outcome of the SDD*

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 beginning idea of how it might look like. The software modules that facilitate this are the ***Game Controller*** and ***Task Generator*** from Cortex-M7, and the ***Task Manager*** from Cortex-M4.

First, compared to the old FSM [5], the new one was made to match on each core. Done so the synchronization can be represented and understood better by future developers, when more functionality will be added to certain stages of the game. Also, more states were added, for the same reasons as the last modifications. These states are the ***cheat*** and ***clean-up*** state, as well as a revised meaning for the game end task, and clear intentions for the rest. Look at table X for further elaboration.

The new states, require new tasks to be designed as well. Tasks and the aforementioned HSEM notifications (figure11), are the same when looking at the code, just at different abstraction levels. The semaphores in the code 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 doing. 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 decide what needs to happen next, send another notification, and go on like this until the game ends.

TABLE 2: THE PURPOSE OF EACH STATE, ITS TRIGGERS AND OUTPUTS FOR CORTEX-M7

Cortex-M7 States	**Purpose of state**	**Transition From**	**Transition To**	**Input HSEM**	**Output HSEM**
Initialize	To initialize CM7 and send an initialization signal to CM4	Boot of system	Start Game	CM4_DONE	CM4_INIT
Start Game	Set game variables, request difficulty (when/if implemented), inform user that game is ready	Initialize	User Move	None	None
User Move	Sends signal to CM4 which will legitimize any user input token. Future: Sends column in which user plays to RaspPi	1. Start Game 2. Idle	Idle	CM4_DONE	USER_TURN
Idle	Checks for win conditions before advancing to next state	User/Robot Move	1. User/Robot Move 2. Game End	None	None
Robot Move	Sends signal to CM4 in which column robot should play Future: Request move from RaspPi	Idle	Idle	CM4_DONE	ROBOT_TURN
Cheat Detected	State to take care of necessary actions for this special case. Notify user.	Interrupt based from CM4 ???	Game End	CHEATER	None
Game End	State to take care of necessary actions for this special case. Notify user.	1.Idle 2. Cheat	Clean-up	CM4_DONE	GAME_END
Clean-up	Prepares the game board for next player	Game End	Start Game	CHEATER	None

In the end, the figure bellow describes the communication and exchange of information of each core. More detailed information on the states/tasks and their meaning can be found in the Annex[to be attached later].

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

[include something about research into RGB, Lightgate, but more about the communication with them, even thought the coms libraries are already written, the logic for the modules themselves is lacking.]

[also mention the UART debug-controller EITHER how its designed, or for a FUTURE IMPROVEMENT]

VII Detailed Design/ Module Design

A. *Can be described in a MDD (Module Design Document) and attached to the report*

B. *In this chapter you describe the outcome of the SDD regarding the design (calculations, simulations, schematics, software)*

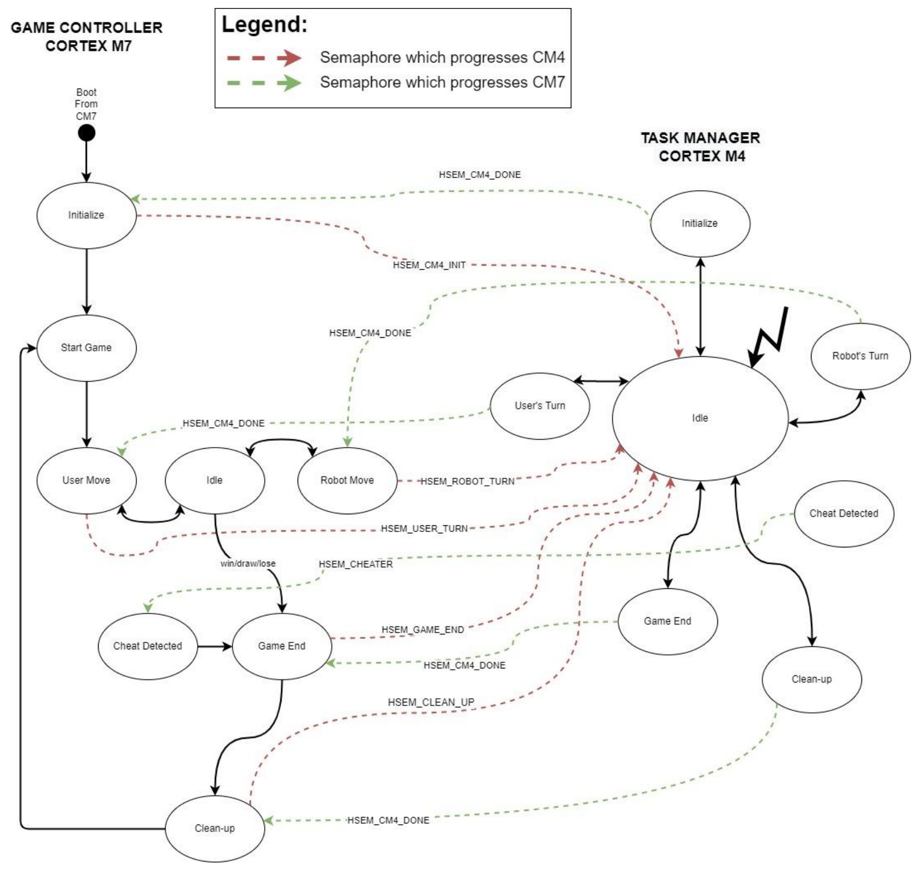


FIGURE 12: THE MAIN FSM OF EACH CORE

[problems with power setting (SMPS) and deadlock state of the controller due version of softwares]

Format used for coding (maintainability)

VIII Realization

A. *Can be described in a MDD (Module Design Document) and attached to the report*

B. *In this chapter you describe the outcome of the SDD regarding the building of the prototype*

C. *The test plan and test report of the MDD can be used as source*

IX Verification and validation

1. Test set-up

2. Test results

X Result analysis

XI Conclusions

The reader should be able to understand this chapter even when he, immediately after he has read the introduction and chapters, has skipped all intermediate: make sure you connect the content within the conclusion chapter!

The reader who has read the whole report, should encounter no new information in this last chapter, indeed: he must be able to predict what it says! In this chapter the results are compared with the initial assignment (requirements/specifications) 15

and conclusions are drawn. Do not draw conclusions that are not underpinned with previous mentioned results. Conclusions coming out of the blue are not acceptable!

Recommendations (could be a separate chapter) tell the reader what should be improved or still has to be done in order to complete the assignment

This last chapter has no figures or lists. The maximum length is one page.

XII Recommendations

Evaluation

This is not a chapter, and therefore has no number and no sections. Just like the foreword or preface the evaluation is a personal part of the report and you can write this component also in the 'I' form. You reflect on the experiences you have had during the project. You oversee the whole journey and you discuss what you've learned. You describe what you've found and what you remember as your most "teachable or valuable moments" i.e.: when did the error(s) or problem(s) occur and why; especially how you've solved the problems and again emphasize that!

This is not the place to settle outstanding accounts. But suppose there was a profound reorganization at your Department, where many people are transferred or dismissed, then of course this has influenced your work, and then you need to mention this. But do this carefully, without offending somebody.

Finally it is advised to take some time to look back at and evaluate your study. First compare your graduation time, subjects, needed skills, needed knowledge, etc. to that what you have learned at Fontys Engineering. Which subjects, courses, practical's and projects were helpful or even indispensable. Also you could advice how to change the curriculum of Fontys Engineering from every possible view point. Adding or deleting subjects and/or courses, change practical's, change the way of teaching, you name it.

This will help Fontys Engineering to keep the curriculum updated and in that way Fontys Engineering is able to educate the engineer of the future!

To be clear: this part is not often written in (business) reports. But some universities do want this part to show your competences and your (positive) critical view on your education. Fontys Electrical Engineering is happy with this separate chapter as a learning experience for the study.

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, "Reference Manual STM32H755," in *11. Hardware semaphore (HSEM)*.

Attachments

A. Original assignment

B. Project plan

C. Originality Declaration

E. SRD, System Requirements Document (optional)

F. SDD, System Design Document (optional)

H. TRD, Test Report Document (optional)