- Architectural set-up
- State Machine Improvements

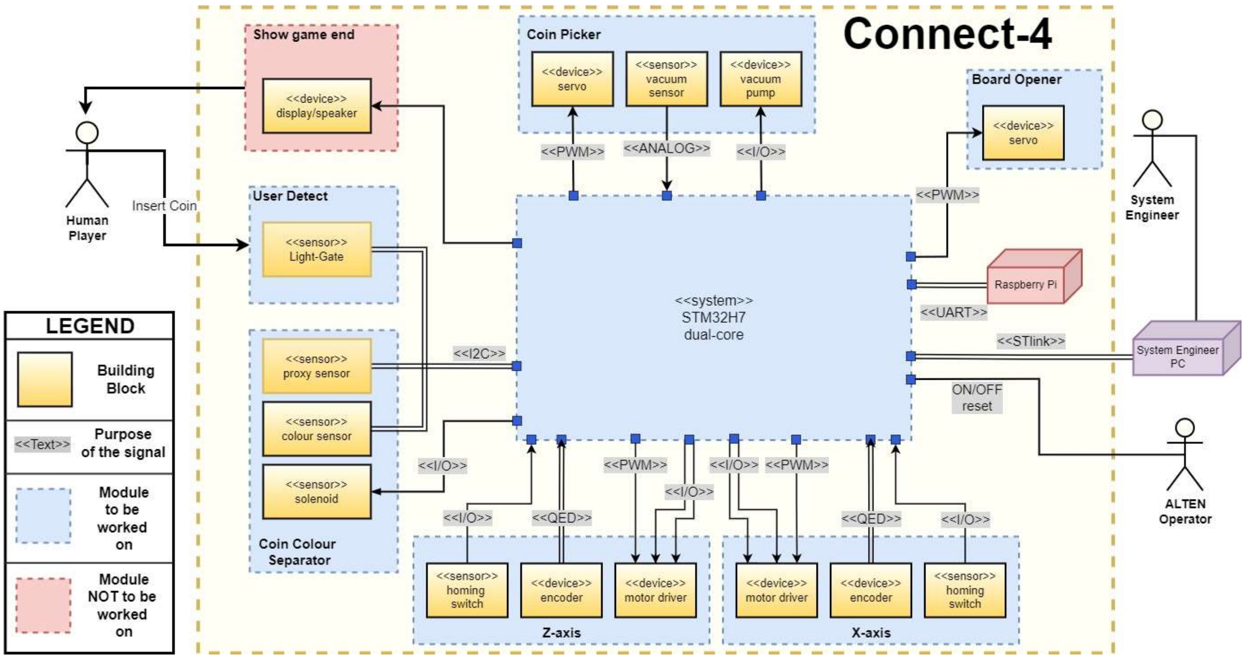
2. Research (divide the information into chapters later)

- SAD / STM32 HAL - BASELINE
- Existing Projects – DIVING INTO THE EXISITNG WORK
  - Dual-Core Communication Demo [DataExchg Blinky] (Pascal)
  - Dual-Core Connect-4 Demo [Proximity & TaskGenerator & MotorX Code] (Pascal)
  - Dual-Core Initialization [Based on C4-Demo adds MotorZ, GPIO controls] (Laurens)
  - Legacy low-level code
- Hardware semaphores and their role in GameController-CM7, CM4TaskGenerator-CM7, TaskManager-CM4 → My own design
- Mapping legacy code to new architecture (figuring out how to blend all the projects together) → My own work

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

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, which are comprised of the already mentioned “smaller” 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.



1

2 FIGURE 4: OVERVIEW OF THE ROBOT PLAYER

3

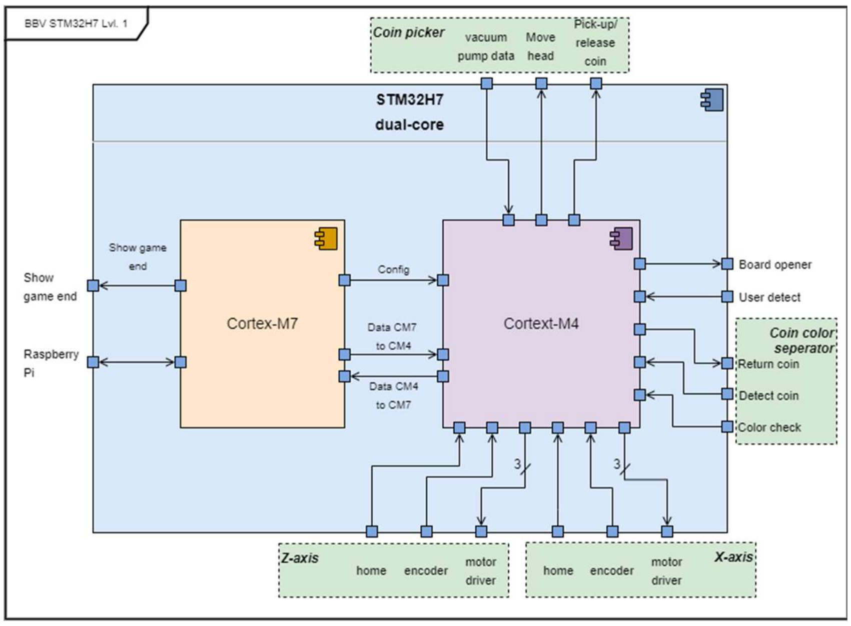


FIGURE 5: LEVEL 1 OF THE SOFTWARE ARCHITECTURE 18

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.

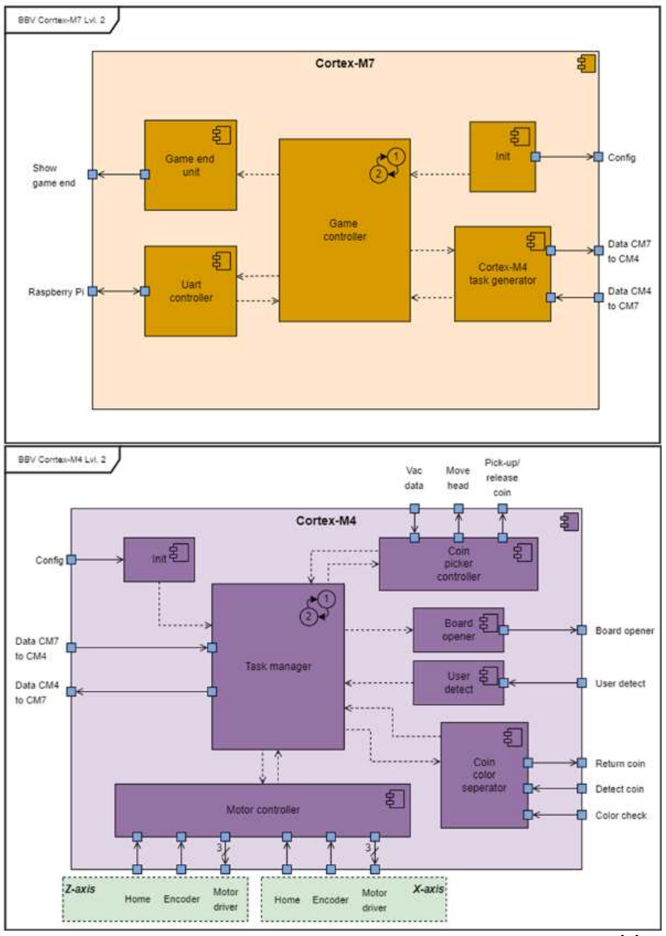


FIGURE 6: LEVEL 2 OF THE SOFTWARE ARCHITECTURE FOR BOTH CORES

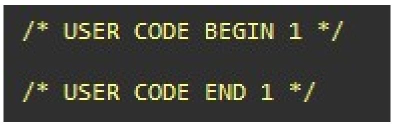
The second level is as deep as it goes for Cortex-M7, since the rest of the functionality is out of scope for this project and is for future upgrades. However this doesn't mean that there's no work to be done on this level. The game controller is the main finite state machine of the Connect-4 robot player, 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, 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".



```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

FIGURE 7: USER CODE MARKING

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 Figure 4 some of them could be noticed, like the PWM and I2C.

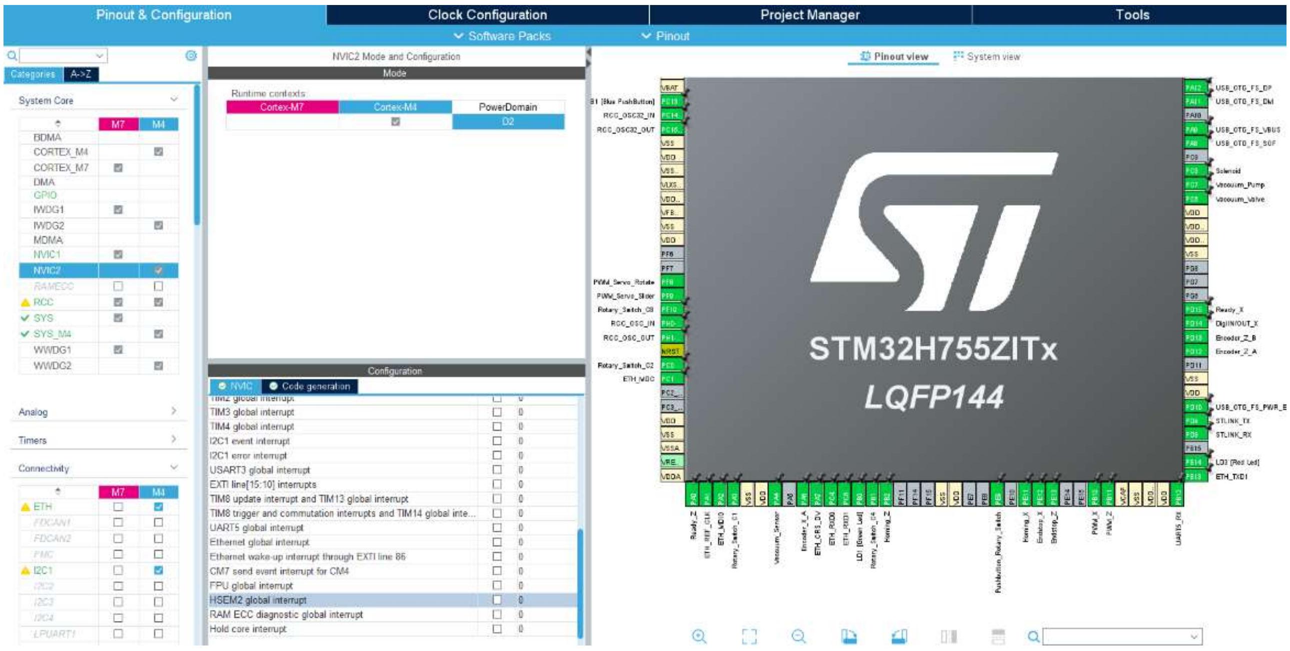
The full list of peripherals in use and their functions are as follows: [to copy acronyms to top]

- 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 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 (Hardware Semaphores) 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 and it adds the functionality to move the motors and to perform a homing routine.
Due to lack of documentation, most of the logic of the codes 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.



System Core

	M7	M4
BDMA		
CORTEX_M4		✓
CORTEX_M7	✓	
DMA		
GPIO		
IWDG1	✓	
IWDG2		✓
MDMA		
NVIC1	✓	
NVIC2		✓
RAMECC	☐	☐
RCC	✓	✓
SYS	✓	
SYS_M4		✓
WWDG1	✓	
WWDG2		✓

Connectivity

	M7	M4
ETH	☐	✓
FDCAN1	☐	☐
FDCAN2	☐	☐
FMC	☐	☐
I2C1	☐	✓
I2C2	☐	☐
I2C3	☐	☐
I2C4	☐	☐
LPUART1	☐	☐

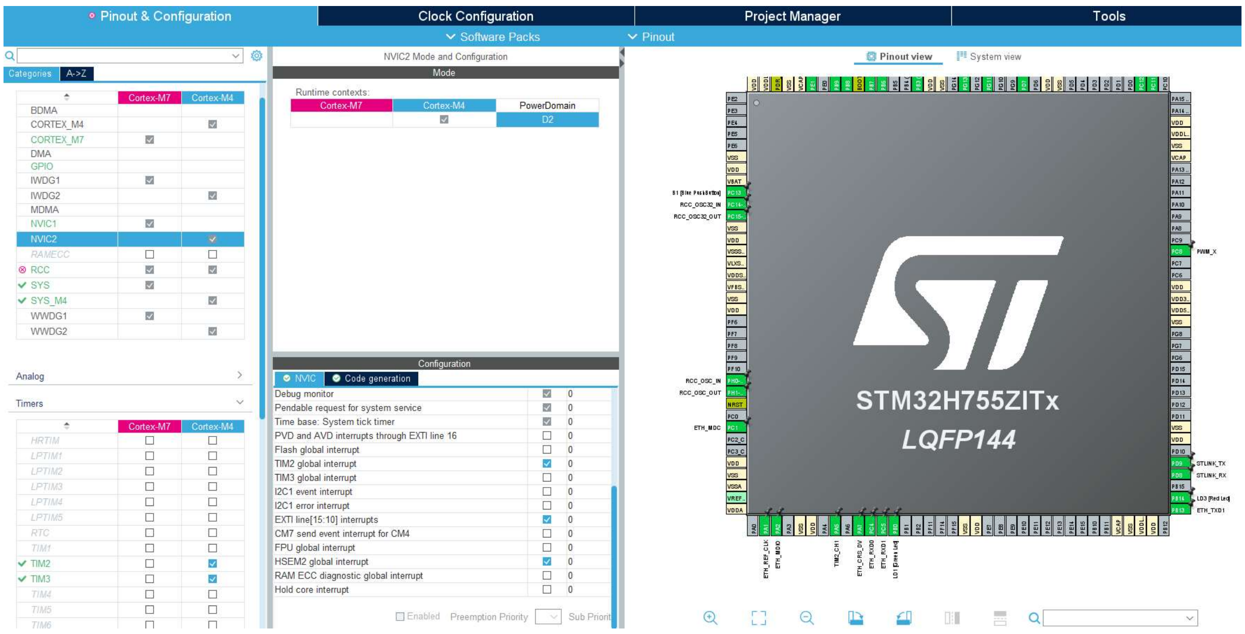
Runtime contexts

Cortex-M7	Cortex-M4	PowerDomain
	✓	D2

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

FIGURE 8: THE .IOC CONFIGURATION FILE OF LATEST IMPLEMENTATION

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



	Cortex-M7	Cortex-M4
BDMA		
CORTEX_M4		✓
CORTEX_M7	✓	
DMA		
GPIO		
IWDG1	✓	
IWDG2		✓
MDMA		
NVIC1	✓	
NVIC2		✓
RAMECC	☐	☐
RCC	✓	✓
SYS	✓	
SYS_M4		✓
WWDG1	✓	
WWDG2		✓

	Cortex-M7	Cortex-M4
HRTIM	☐	☐
LPTIM1	☐	☐
LPTIM2	☐	☐
LPTIM3	☐	☐
LPTIM4	☐	☐
LPTIM5	☐	☐
RTC	☐	☐
TIM1	☐	☐
TIM2	☐	✓
TIM3	☐	✓
TIM4	☐	☐
TIM5	☐	☐
TIM6	☐	☐

NVIC2 Mode and Configuration

Mode

Runtime contexts:

Cortex-M7	Cortex-M4	PowerDomain
	✓	D2

Configuration

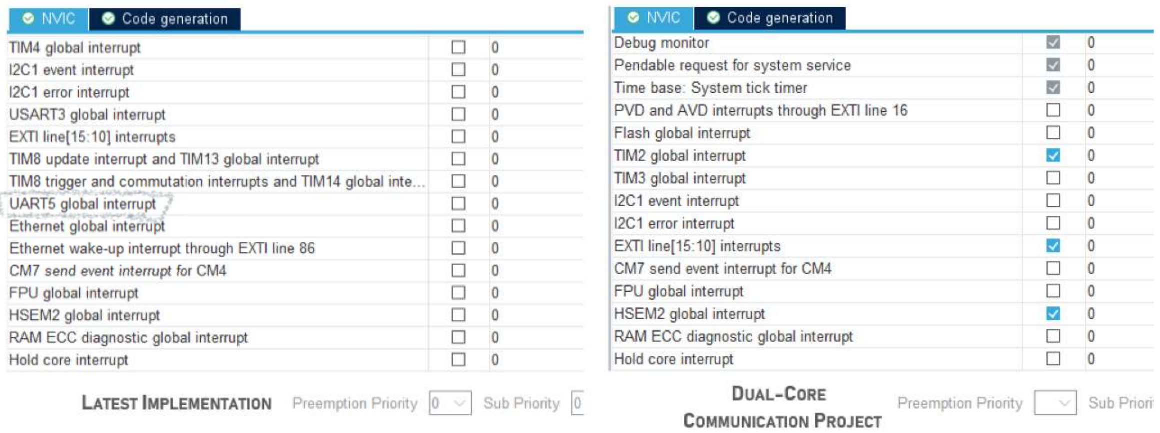
NVIC | Code generation

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

☐ Enabled Preemption Priority Sub Priority

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.


NVIC | Code generation

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 Preemption Priority 0 Sub Priority 0

NVIC | Code generation

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 Preemption Priority Sub Priori

FIGURE 10: COMPARISON OF SETTINGS ACROSS PROJECTS

The presence of options was later linked to their enabling in different sections of the configuraion. 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 I started the internship. 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.

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

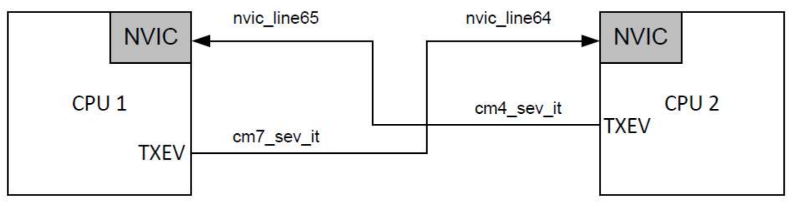


FIGURE 11: SEND-EVENT INSTRUCTION NOTIFICATION MECHANISM [8]

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; operations could be performed on time. As such, one core will send notifications with commands, while the other will send notification with status updates.

To sum up, 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 that through locking, it is ensured that only one core has access to a peripheral.

The improvements to be made in the game logic are to expand the current tasks and give a clear meaning to their functions. Add tasks which describe the system in a better and to a fuller extent. Implement the design.

3. Results

4. Conclusions

V Specification

Make sure:

A. *Clear definition of the test cases and their outcome*
B. *MoSCoW model for realistic overview*
C. *Can be described in a SRD (System Requirement Document) and attached*
D. *In this chapter you describe the outcome of the SRD*

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*

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)*

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

D. Confidentiality Declaration (optional)

E. SRD, System Requirements Document (optional)

F. SDD, System Design Document (optional)

G. MDD, Module Design Document (optional)

H. TRD, Test Report Document (optional)