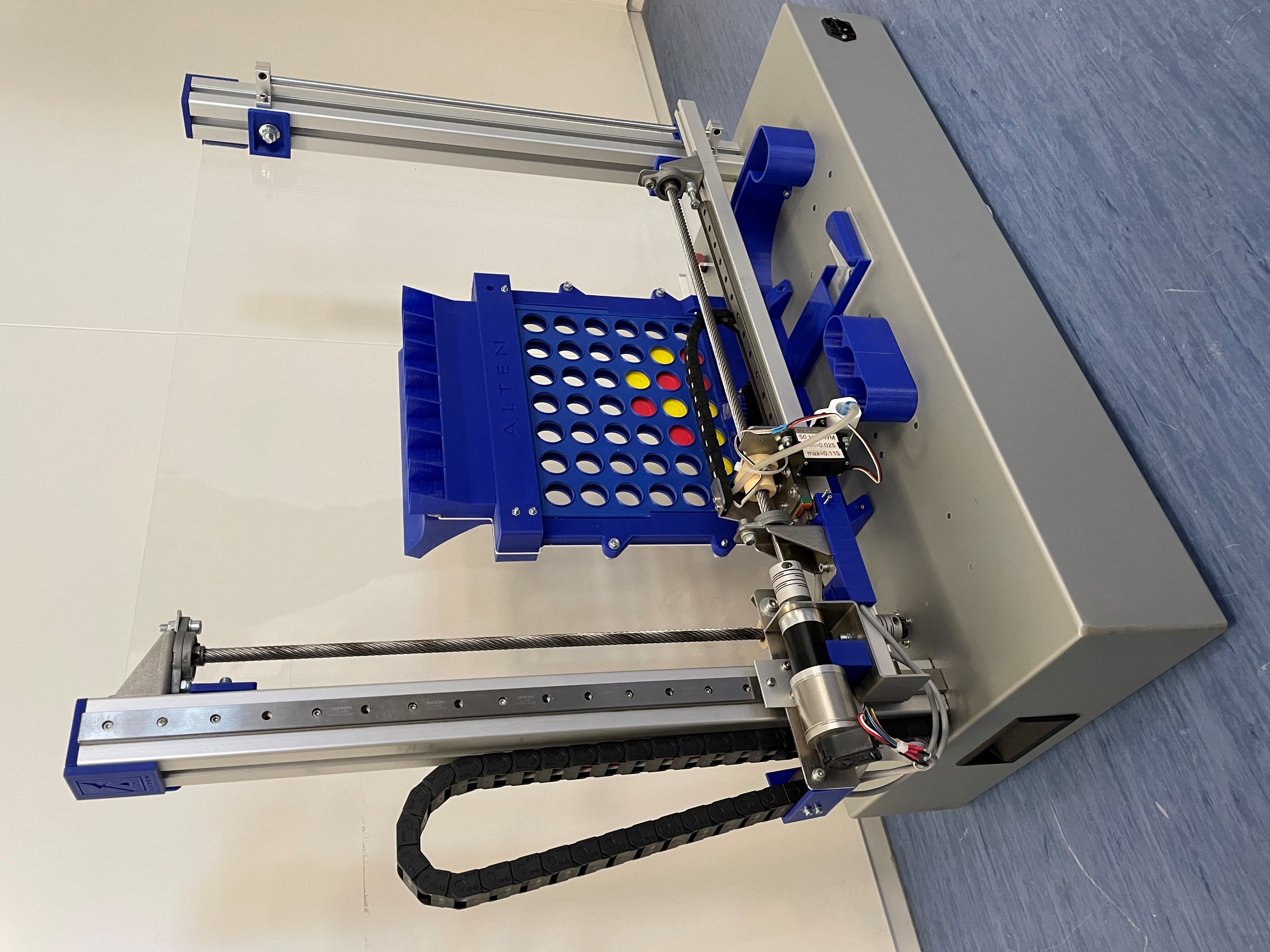
4-op-1-rij robot -

**Het ontwerpen van een gestructureerde en modulaire software architectuur**



Versie: v1.0

Datum: 7-6-2022

Auteur: Pascal Faatz

Algemene informatie

Student: Pascal Faatz

La Traviata 16

5629NN, Eindhoven

[p.faatz@student.fontys.nl](mailto:p.faatz@student.fontys.nl) / [pascal.faatz@alten.nl](mailto:pascal.faatz@alten.nl)

+31 (0)6 43 169 199

Studentnummer : 2491281

Klas: 43\_E4AFST

Bedrijf: ALTEN Nederland B.V.

Hurksestraat 45

5652 AH, Eindhoven

Technisch begeleider Aniel Shri

[Aniel.shri@alten.nl](mailto:Aniel.shri@alten.nl)

+31 (0)6 37 162 867

Business manager Gijs Haans

[Gijs.haans@alten.nl](mailto:Gijs.haans@alten.nl)

+31 (0)6 27 025 966

School: Fontys University of Applied Sciences

De rondom 1

5612 AP, Eindhoven

School begeleider: Jeedella S.Y. Jeedella

[j.jeedella@fontys.nl](mailto:j.jeedella@fontys.nl)

+31 088 507 81 48

**Gerefereerde documenten**

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Referentie | Titel | Schrijver |
|  | Motor driver datasheet [1] | ESCON 36/3 EC Servo Controller | Maxon group |
|  | Encoder datasheet [2] | Encoder HEDL 5540 | Maxon group |
|  | Motor datasheet [3] | Maxon motor EC-i 40 | Maxon group |
|  | Servo datasheet [4] | Parallax Servo | Parallax |
|  | Powersupply datasheet [5] | Mean Well LRS-150 | Mean Well |
|  | Vacuüm pomp datasheet [6] | SparkFun D2028 pump | SparkFun |
|  | Solenoid datasheet [7] | Adafruit 413 solenoid | Adafruit |
|  | RBG sensor datasheet [8] | Taos TCS3472 | Taos |
|  | Lichtsluis design (intern document) | SDD\_photodiodeboard.docx | Arjan Verboord |
|  | Custom PCB design (intern document) | 013-E-0001-D.brd | Jeroen Wilbers |

**Voorwoord**

Voor u ligt het verslag van mijn afstudeeropdracht “4-op-1-rij robot - Het ontwerpen van een gestructureerde en modulaire software architectuur”. Het verslag dient als afronding van de opleiding Elektrotechniek aan de Fontys Hogescholen Eindhoven.

Op het Meet & Match event van Fontys in 2019 ben ik in contact gekomen met mevrouw Romy Wachtmeester van ALTEN Nederland. Bij een hernieuwede kennismaking in 2021 bleek ALTEN een mooie afstudeeropdracht beschikbaar te hebben. Ik ben dankbaar voor de mogelijkheid die zij mij hebben geboden om in de professionele omgeving mijn studie af te ronden. Ik heb gedurende de afstudeerstage van 1 februari tot 1 juli 2022 gebruik mogen maken van alle faciliteiten op het kantoor van ALTEN in Eindhoven.

Gedurende het afstuderen ben ik enorm op weg geholpen door Aniel Shri, technical supervisor en Gijs Haans, businessmanager van uit ALTEN. Ik wil hen hierbij bedanken voor alle informatie, begeleiding en feedback die zij mij hebben gegeven. Ook Jeedella S.Y. Jeedella, mijn begeleider vanuit Fontys elektrotechniek heeft bijgedragen aan de totstandkoming van de opdracht. Dank hiervoor.

Tot slot bedank ik mijn collega stagiaires en de consultants van ALTEN voor hun tijd en medewerking in de beantwoording van mijn vragen en het delen van kennis. Dit heeft mijn opdracht de diepgang gegeven waarna ik opzoek was. Ik kijk terug op een gezellige en leerzame tijd.

Dan rest er nog een woord van dank, voor u de lezer van dit rapport.

Ik wens u veel lees plezier.

Pascal Faatz

Eindhoven, 7-6-2022

Inhoudsopgave

[Samenvatting 7](#_Toc105527361)

[Summary 8](#_Toc105527362)

[Afkortingen 9](#_Toc105527363)

[1 Inleiding 10](#_Toc105527364)

[2 ALTEN 11](#_Toc105527365)

[3 De opdracht 12](#_Toc105527366)

[3.1 Probleemstelling 12](#_Toc105527367)

[3.2 Doelstelling 12](#_Toc105527368)

[3.3 Ontwerpopdracht 12](#_Toc105527369)

[3.4 Scope en afbakening 13](#_Toc105527370)

[3.5 Requirements 13](#_Toc105527371)

[3.6 Project aanpak 14](#_Toc105527372)

[4 Vooronderzoek 15](#_Toc105527373)

[4.1 De 4-op-1-rij robot 15](#_Toc105527374)

[4.1.1 Spelverloop 15](#_Toc105527375)

[4.1.2 Hardware identificatie 15](#_Toc105527376)

[4.2 arc42 17](#_Toc105527377)

[4.3 Modulariteit 18](#_Toc105527378)

[4.4 STM32H7 dual-core 19](#_Toc105527379)

[4.5 Core verdeling 20](#_Toc105527380)

[4.6 STM32CubeIDE 20](#_Toc105527381)

[4.7 Conclusie 21](#_Toc105527382)

[5 Ontwerp software architectuur 22](#_Toc105527383)

[5.1 Systeem context SAD 22](#_Toc105527384)

[5.2 Building block view SAD 23](#_Toc105527385)

[5.3 Runtime view SAD 27](#_Toc105527386)

[5.4 Deployment view SAD 30](#_Toc105527387)

[5.5 Modulaire software modules SAD 30](#_Toc105527388)

[5.6 Conclusie 31](#_Toc105527389)

[6 Implementatie en testen 32](#_Toc105527390)

[6.1 Implementatie methode 32](#_Toc105527391)

[6.2 Dual-core communicatie 32](#_Toc105527392)

[6.2.1 Shared memory 33](#_Toc105527393)

[6.2.2 Notificaties 33](#_Toc105527394)

[6.2.3 Implementatie dual-core communicatie 35](#_Toc105527395)

[6.3 BSP modules voor hardware 36](#_Toc105527396)

[6.3.1 Dual-core i2c UART demo 36](#_Toc105527397)

[6.3.2 Dual-core i2c UART PWM demo 37](#_Toc105527398)

[6.4 Conclusie 38](#_Toc105527399)

[7 Validatie 39](#_Toc105527400)

[8 Conclusie en aanbevelingen 41](#_Toc105527401)

[Evaluatie 42](#_Toc105527402)

[Bibliografie 43](#_Toc105527403)

[Bijlagen 44](#_Toc105527404)

[I. Originaliteitsverklaring 44](#_Toc105527405)

[II. Plan van aanpak 45](#_Toc105527406)

[III. Software Architecturale Document 59](#_Toc105527407)

[IV. Memory en bus architectuur STM32H7 dual-core 93](#_Toc105527408)

[V. Implementatie methodes voor een software loop 94](#_Toc105527409)

[VI. False data 96](#_Toc105527410)

[VII. Complicatie dual-core communicatie 98](#_Toc105527411)

[VIII. Code Dual-core communicatie task generator 100](#_Toc105527412)

[IX. Code i2c module 102](#_Toc105527413)

[X. Code UART module 104](#_Toc105527414)

**Lijst van figuren**

[Figuur 1 Kantoren ALTEN Nederland 11](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527515)

[Figuur 2 Organisatie diagram 11](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527516)

[Figuur 3 V-model 14](#_Toc105527517)

[Figuur 4 blokdiagram 4-op-1-rij robot 15](#_Toc105527518)

[Figuur 5 Bovenaanzicht 4-op-1-rij 16](#_Toc105527519)

[Figuur 6 Onderaanzicht 4-op-1-rij 16](#_Toc105527520)

[Figuur 7 arc42 logo 18](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527521)

[Figuur 8 Embedded software layers 18](#_Toc105527522)

[Figuur 9 Nucleo-H755ZI-Q 19](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527523)

[Figuur 10 STCubeMX software tool gegenereerde layers 20](#_Toc105527524)

[Figuur 11 Project context 22](#_Toc105527525)

[Figuur 12 Systeem context 23](#_Toc105527526)

[Figuur 13 BBV STM32H7 level 1 24](#_Toc105527527)

[Figuur 14 BBV Cortex-M7 level 2 25](#_Toc105527528)

[Figuur 15 BBV Cortex-M4 level 2 26](#_Toc105527529)

[Figuur 16 BBV Coin color seperator level 3 27](#_Toc105527530)

[Figuur 17 State machine spelverloop 28](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527531)

[Figuur 18 State machine real-time processing 28](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527532)

[Figuur 19 Sequence diagram robot move 29](#_Toc105527533)

[Figuur 20 Pin out STM32H7 dual-core 30](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527534)

[Figuur 21 Software layers Coin color separato 31](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527535)

[Figuur 22 Dual-core communicatie [17] 33](#_Toc105527536)

[Figuur 23 EXTI en SEV dual-core communicatie [17] 34](#_Toc105527537)

[Figuur 24 HSEM dual-core communicatie [17] 34](#_Toc105527538)

[Figuur 25 dual-core implementatie STM32H7 dual-core 34](#_Toc105527539)

[Figuur 26 Schematisch overzicht demo dual-core communicatie 35](#_Toc105527540)

[Figuur 27 Schematisch overzicht demo dual-core i2c UART 36](#_Toc105527541)

[Figuur 28 Software layers dual-core i2c UART 37](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527542)

[Figuur 29 Schematisch overzicht demo dual-core i2c UART PWM 37](#_Toc105527543)

[Figuur 30 Round robin 94](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527544)

[Figuur 31 Round robin with interrupt 94](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527545)

[Figuur 32 RTOS scheduler 95](file:///C:\Users\Pascal\OneDrive%20-%20ALTEN%20Group\Documents\School\Afstudeer_Stage\Eindverslag\Afstudeerverslag%202021-2022%20Faatz%20Pascal%202491281.docx#_Toc105527546)

[Figuur 33 Data cohorentie problem [22] 98](#_Toc105527547)

**Lijst van tabellen**

[Tabel 1 Requirements 13](#_Toc105522320)

[Tabel 2 Componenten 4-op-1-rij 17](#_Toc105522321)

[Tabel 3 Omschrijving BBV Cortex-M7 level 2 25](#_Toc105522322)

[Tabel 4 Omschrijving BBV Cortex-M4 level 2 26](#_Toc105522323)

[Tabel 5 Omschrijving BBV Coin color seperator level 3 27](#_Toc105522324)

[Tabel 6 Geheugen toegang [17] 33](#_Toc105522325)

[Tabel 7 Mogelijke patronen gedeelde data 35](#_Toc105522326)

[Tabel 8 Mogelijke patronen HSEM 36](#_Toc105522327)

[Tabel 9 Gevalideerde requirements 39](#_Toc105522328)

# Samenvatting

ALTEN is een internationaal toonaangevend consultancybedrijf gespecialiseerd in advisering & engineering in High Tech ontwikkelomgevingen. Het bedrijf heeft in eigen beheer een robot ontwikkeld die autonoom 4-op-1-rij kan spelen tegen een menselijke tegenstander. De robot is een demonstratie unit die wordt gebruikt op technische beurzen en opendagen van universiteiten en hogescholen om klanten en medewerkers te werven. ALTEN wil de robot blijven gebruiken en vraagt om een optimalisatie van de kwaliteit en de voorbereiding voor een verbetering van de performance.

Sinds 2019 hebben verschillende engineers aan de robot gesleuteld. Hierdoor is de software architectuur onoverzichtelijk en inconsistent geworden. Daarnaast leeft bij ALTEN de wens om de hardware in de toekomst uit te breiden. De optimalisatie van de robot vraagt om een complete re-design van het besturingssysteem en de vervanging van de microcontroller. De vraag van ALTEN is samengevat in de volgende opdracht: *“Stel een gestructureerde en modulaire software architectuur op voor het besturingssysteem van de 4-op-1-rij robot, waarmee alle veranderingen en uitbreidingen in de toekomst gefaciliteerd kunnen worden en implementeer deze software architectuur op een STM32H7 dual-core microcontroller.”*

Binnen het project is gewerkt met het V-model. Dit heeft er voor gezorgd dat alle noodzakelijke projectstappen gestructureerd zijn doorlopen. Eerst zijn met alle stakeholders de requirements vastgesteld. De requirements dienen als uitgangspunt voor het project, zijn bepalend voor de keuzes in het vooronderzoek en dienen ter validatie bij de oplevering van het projectresultaat. De belangrijkste requirements hebben betrekking op een gestructureerde en modulaire software architectuur die logisch is opgebouwd zodat software developers de software efficiënt kunnen beheren en upgraden. Daarnaast wordt een Board Support Package (BSP) gevraagd om de noodzakelijke hardware aan te sturen. Beiden moeten geïntegreerd worden op een STM32H7 dual-core microcontroller.

In de fase van het vooronderzoek is de interactie tussen hard- en software geanalyseerd. Een re-design van de hardware hoort niet tot de scope van het project maar de hardware is onlosmakelijk met het besturingssysteem verbonden. Op basis van het functioneren van het systeem en de requirements zijn vervolgens enkele methodologische en technische keuzes gemaakt. De belangrijkste hebben betrekking op de template voor de software architectuur (arc42), de core verdeling en de IDE (STM32CubeIDE). Bij deze keuzes is gelet op kwaliteit, gebruiksvriendelijkheid en robuustheid.

In de design fase is de software architectuur voor het besturingssysteem opnieuw opgebouwd. Alle stappen van het template arc42 zijn doorlopen en verduidelijkt met diagrammen. Dankzij de diagrammen en de modulaire opbouw is het duidelijk hoe het systeem werkt en kan het eenvoudig beheerd en uitgebreid worden. De design fase is beoordeeld door de stakeholders en na goedkeuring is overgegaan tot de implementatie en test fase. In deze fase zijn enkele, voor het besturingssysteem noodzakelijke, software modules geïmplementeerd en individueel getest. Daarna zijn een aantal modules samengevoegd tot verschillende demo’s om zo de geschiktheid van de software architectuur aan te tonen.

In de validatie fase is getoetst of is voldaan aan de requirements. Het besturingssysteem heeft een complete re-design van de software architectuur gehad en op basis van een BSP is alle hardware aan te sturen met een dual-core microcontroller. De demo’s zijn succesvol doorlopen en daarmee is aan alle belangrijke requirements voldaan. De robot werkt echter nog niet. Daarvoor moeten er nog enkele modules worden ontworpen. Dit kan eenvoudig aan de hand van het BSP en de software architectuur. Daarnaast is het systeem na de upgrade klaar voor een uitbreiding van de hardware, denk aan ethernetverbinding en een beeldscherm.

# Summary

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

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

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

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

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

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

# Afkortingen

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

# Inleiding

*Een 4-op-1-rij robot die nooit verliest, ALTEN heeft hem in huis.*.

Iedereen kent het spel 4-op-1-rij. Het spel wordt gespeeld door twee spelers. De ene speler speelt met rode fiches, de andere met gele. Het spelbord bestaat uit zeven kolommen en zes rijen. Om de beurt gooit een speler één fiche in het spelbord. Het doel van het spel is om als eerste vier fiches aansluitend op een horizontale, verticale of diagonale rij te krijgen.

ALTEN heeft in eigen beheer een robot ontwikkeld die middels een algoritme 4-op-1-rij autonoom kan spelen tegen een menselijke tegenstander. De 4-op-1-rij robot is gebouwd met industriële componenten om de kennis van verschillende systemen te demonstreren. De robot wordt gebruikt als demonstratie unit op beurzen en open dagen, waarbij voorbijgangers een ronde kunnen spelen. Als eerste doet de speler een openingszet. Dan is de robot aan de beurt waarbij het besturingssysteem de zet bepaalt, een fiche oppakt en in de gekozen kolom van het bord laat vallen. Het systeem houdt het spelverloop bij. Zodra het spel is afgelopen, verzamelt de robot alle fiches en sorteert ze op kleur. Het spel is klaar voor de volgende ronde.

De 4-op-1-rij robot is binnen ALTEN bedacht om consultants, die tijdelijk niet op een project zitten, een uitdaging te geven. In 2019 is het eerste prototype opgeleverd. Daarna is het besturingssysteem nog door verschillende engineers geoptimaliseerd en uitgebreid. Dit heeft geresulteerd in onsamenhangende software die moeilijk te begrijpen is en daardoor lastig kan worden onderhouden of is uit te breiden.

ALTEN wil de kwaliteit en de performance van de 4-op-1-rij robot voor langere tijd garanderen. Dit betekent dat het besturingssysteem eenvoudig en robuust moet kunnen worden aangepast. Om dit mogelijk te maken zal de architectuur van het besturingssysteem opnieuw en modulair opgebouwd moeten worden. Het re-design van de software architectuur biedt de mogelijkheid om parallel een hardware upgrade uit te voeren, waarbij de single-core microcontroller wordt vervangen door een dual-core microcontroller. Hierdoor kunnen uitbreidingen worden toegevoegd die de functie van de robot als demonstratie unit onderstrepen. Een gestructureerde en modulaire software architectuur voor het besturingssysteem, in combinatie met een dual core microcontroller maakt dat de robot klaar is voor de toekomst.

De software architectuur moet gebruiksvriendelijk zijn zodat engineers met weinig kennis van hardware toch met de componenten van de robot kunnen werken. Daarom is gekozen voor gestandaardiseerde methoden. Voor het re-design van de software architectuur wordt een template gebruikt die standaard is binnen ALTEN (arc42) en zorgt voor een gestructureerde opbouw van de software m.b.v. diagrammen. De software bestaat uit embedded software layers die modulair zijn en gedeeltelijk automatisch worden gegenereerd. De keuze van ALTEN voor de STM32H7 dual-core microcontroller heeft geleid tot het gebruik van een IDE van dezelfde leverancier, zodat de kwaliteit van de communicatie binnen de core op voorhand is gegarandeerd. De methoden zijn actueel en er is naslagwerk beschikbaar.

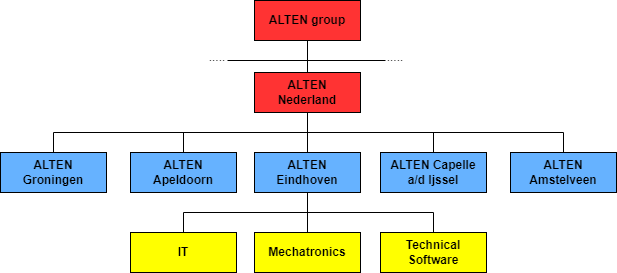
Dit rapport begint met een korte introductie over ALTEN (hoofdstuk 2). Daarna volgt een beschrijving van de opdracht (hoofdstuk 3). Het functioneren van de huidige robot, de requirements van ALTEN en de methodologische keuzes worden toegelicht in hoofdstuk 4. Vervolgens worden in hoofdstuk 5 de belangrijkste onderdelen van de software architectuur beschreven. In hoofdstuk 6 wordt de implementatie en het testen toegelicht aan de hand van enkele demo’s. Uiteindelijk worden de requirements in hoofdstuk 7 gevalideerd. Het rapport wordt afgesloten met enkele conclusies en aanbevelingen in hoofdstuk 8.

# ALTEN

ALTEN is een consultancy en engineering organisatie voor hightech en IT sector. Het bedrijf is in 1988 opgericht en heeft het hoofdkantoor in Frankrijk. ALTEN is een internationaal bedrijf met meer dan 42.000 werknemers in 24 landen. Er zijn 5 kantoren in Nederland (Figuur 1) met totaal 1.200 werknemers waarvan 91% engineer [9].



Figuur 1 Kantoren ALTEN Nederland

Technologie is dé centrale pijler binnen ALTEN. Alten biedt kwaliteit, betrouwbaarheid en innovaties op het gebied van hightech oplossingen. ALTEN werkt in opdracht voor bedrijven uit de automotive, hightech (maak)industrie, defensie, energie, verkeer/ vervoer en telecom sector. De klant is eigenaar van het eindresultaat en zonder toestemming kan niet over opdrachten of het eindresultaat gecommuniceerd worden.

Figuur 2 Organisatie diagram

ALTEN heeft in Eindhoven drie afdelingen: IT, Technical Software en Mechatronics (Figuur 2). Het project voor de 4-op-1-rij robot valt binnen de afdeling Mechatronics. Op deze afdeling zitten consultants met een achtergrond in mechatronica, werktuigbouw, elektrotechniek en regeltechniek. Alle specialismen zijn vertegenwoordigd om technische vragen integraal te beantwoorden. De medewerkers zijn de belangrijkste productiefactor van het bedrijf. ALTEN investeert in individuele ontwikkeling, het verder uitbouwen van expertise en het bieden van een springplank voor de toekomst.

ALTEN hanteert voor elk project hetzelfde organisatiemodel. Binnen dit model is voor elke stakeholder duidelijk wat zijn rol is (zie bijlage II. Plan van aanpak, Figuur 4). Ook voor het project 4-op-1-rij robot is dit model toegepast. De stakeholders zijn beschreven in Tabel 1 van bijlage II. Plan van aanpak. Tijdens het proces is gecommuniceerd met, en gerapporteerd aan de technical supervisor, businessmanager, de klant (ALTEN) en de stagebegeleider van Fontys Hogeschool Eindhoven.

# De opdracht

In dit hoofdstuk worden de probleemstelling, doelstelling en de opdracht beschreven. In paragraaf 3.4 wordt stil gestaan bij de requirements die door de stakeholders zijn geformuleerd. De requirements bepalen het resultaat van het project en worden in hoofdstuk 7 gevalideerd. Het hoofdstuk sluit af met het V-model, dat als leidraad wordt gebruikt voor het project.

## Probleemstelling

De 4-op-1-rij robot is een demonstratie unit om nieuwe klanten (techniek beurzen) of nieuwe medewerkers (opendagen op HBO/ universiteiten) aan te trekken.

De robot heeft tot nu toe goed gefunctioneerd. Het systeem werkt en de hardware is op orde. Maar de software en software architectuur vragen om een revisie. Omdat meerdere consultants over lange periode aan de software hebben geprogrammeerd is het besturingssysteem zeer onoverzichtelijk en de architectuur onduidelijk. Daarnaast heeft de single-core microcontroller bijna geen ruimte meer om een upgrade van de hardware mogelijk te maken.

Omdat de 4-op-1-rij robot representatief is voor de kennis en kunde van ALTEN is het belangrijk dat componenten, software en software architectuur continu doorontwikkeld kunnen worden door verschillende specialisten.

## Doelstelling

Het doel van de opdracht is tweeledig. Met de opdracht wordt beoogd de software en software architectuur van het besturingssysteem van de robot een re-design te geven, zodanig dat de software in de toekomst eenvoudig kan worden aangepast of uitgebreid aan veranderende omstandigheden en wensen. Daarnaast wordt de basis gelegd voor een verbeterde performance van de robot door de toepassing van een dual-core microcontroller. Door middel van een dual core kunnen real-time processing en game-handling van elkaar gescheiden worden en tegelijkertijd parallel draaien.

Voordat aan de opdracht kan worden begonnen is vooronderzoek noodzakelijk om de werking van de robot te doorgronden en om enkele methodologische en ontwerpkeuzes te maken die verband houden met het re-design van de software architectuur.

Toepassing van de dual-core in combinatie met gestructureerde softwarearchitectuur maakt het mogelijk om in de toekomst eenvoudig nieuwe features toe te voegen zoals een beeldscherm en ethernetverbinding. Zo kan de robot als demonstratie unit weer zijn dienst bewijzen en zijn de kwaliteit en performance voor langere tijd verzekerd.

## Ontwerpopdracht

De opdracht luidt als volgt:

*“Stel een gestructureerde en modulaire software architectuur op voor het besturingssysteem van de 4-op-1-rij robot, waarmee alle veranderingen en uitbreidingen in de toekomst gefaciliteerd kunnen worden en implementeer deze software architectuur op een STM32H7 dual-core microcontroller.”*

## Scope en afbakening

* Buiten scope van het project valt het re-design van de hardware en het mechanische deel van de robot;
* Er verandert niets aan het spelverloop van 4-op-1-rij;
* Binnen ALTEN is de robot fysiek beschikbaar voor nader onderzoek;
* Er wordt gewerkt met een door ALTEN gekozen dual-core microcontroller;
* Voor de software architectuur is gebruik gemaakt van de standaard template van ALTEN;
* De keuze voor de programmeertaal C of C++ is de standaard voor de IDE;
* De software architectuur voor het besturingssysteem is een definitieve versie;
* Om design keuzes te verifiëren is een “proof of concept” opgeleverd;
* Er is gevraagd om een re-design van het besturingssysteem van de robot, de implementatie van het volledige systeem wordt niet binnen het project gerealiseerd.

## Requirements

Aan de opdracht zijn een aantal requirements verbonden. Deze requirements zijn geformuleerd in overleg met de stakeholders. In Tabel 1 zijn de requirements gedefinieerd. Deze requirements worden volgens MoSCoW uitgevoerd, waarbij de eisen van de resultaten worden ingedeeld aan de hand van M- Must have , S- Should have, C- Could have of W- Won’t have. In overleg met de technical supervisor van ALTEN is de prioriteit binnen de requirements bepaald.

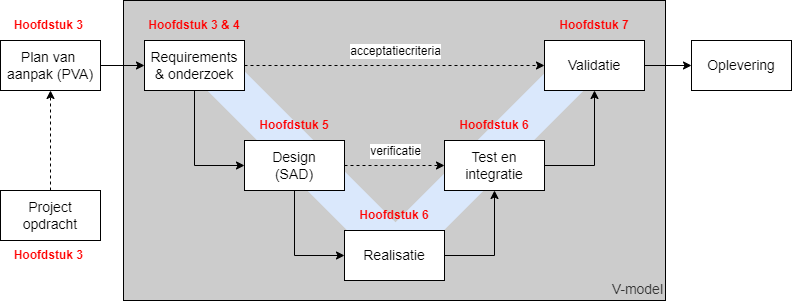
Tabel 1 Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | MoSCoW |
|  | De architectuur van het besturingssysteem moet gestructureerd en modulair zijn om hard- en software developers die niet bekend zijn met het systeem snel inzicht te kunnen geven in het functioneren van de robot. | Must |
|  | De software architectuur moet toekomstbestendig zijn zodat software developers effectief en efficiënt beheer en upgrades kunnen uitvoeren. | Must |
|  | De architectuur van het besturingssysteem dient logisch opgebouwd te zijn aan de hand diagrammen zodat software developers en testers snel inzicht kunnen krijgen in het functioneren van de software. | Must |
|  | De samenhang tussen software en hardware dient logisch opgebouwd te zijn aan de hand van diagrammen zodat software developers de uiteindelijke software kunnen implementeren. | Must |
|  | Er dient een Board Support Package (BSP) gemaakt te worden van het besturingssysteem waarmee de noodzakelijke hardware componenten van de robot aangestuurd kunnen worden. | Must |
|  | De onderdelen van de BSP die noodzakelijk zijn voor het functioneren van de robot moeten binnen het project onafhankelijk van elkaar getest worden. | Must |
|  | Binnen het nieuwe besturingssysteem moet de STM32H7 dual-core microcontroller geïntegreerd worden. | Must |
|  | De robot moet een kalibratie tool hebben die bij de initialisatie van het systeem er voor zorgt dat de motoren in de Z- en X-richting “gehomed” worden. | Must |
|  | Er moet een Pin-out opgesteld worden met een tabel en diagram om er voor te zorgen dat hardware developers in de toekomst een PCB design voor de dual-core processor kunnen uitwerken. | Should |
|  | Het algoritme dat op de Raspberry Pi draait, moet geïntegreerd worden op de nieuwe STM32H7 dual-core. | Could |
|  | Een fysieke demo moet aantonen dat de modulaire opbouw van de software architectuur toepasbaar is en functioneert. | Could |

De requirements definiëren het op te leveren resultaat van het project. In de validatie (hoofdstuk 7 Validatie) wordt de tabel opnieuw doorlopen met de vraag of alle requirements zijn uitgevoerd, welke niet en waarom niet.

## Project aanpak

Om het project gestructureerd te laten verlopen is gekozen voor het V-model. Het V-model is een lineair ontwikkelmodel dat is opgedeeld in een aantal fasen. Elke fase levert een resultaat op, dat nodig is voor de volgende fase. Dit herhaalt zich voor alle fasen, waarbij met elke nieuw resultaat het systeem groeit. Tweede aandachtspunt is dat voor de requirements en het ontwerp aan de linkerzijde een corresponderende validatie respectievelijk integratie/ test bestaat aan de rechterzijde (Figuur 3).



Figuur 3 V-model

# Vooronderzoek

De ontwerpopdracht valt uiteen in enkele onderzoeksvragen die beantwoord moeten worden voordat met het ontwerp kan worden begonnen. Uitgangspunt is een bestaande robot die wordt aangestuurd door een besturingssysteem. Een hardware re-design van de robot ligt buiten de scope van dit project. Echter moet er wel gewerkt worden met de bestaande hardware. Hiermee is de hardware zelf niet buiten scope. Het is belangrijk om vooraf te begrijpen hoe het systeem werkt en uit welke (hardware) onderdelen de robot bestaat. Er zijn verschillende documenten beschikbaar waarin deze informatie is vastgelegd. Daarnaast dient onderzocht te worden welke methoden behulpzaam kunnen zijn bij het re-design van de software en de software architectuur, en hoe de software modulair kan worden opgebouwd. Bovendien moet onderzocht worden wat de context is waarbinnen de gegeven dual-core microcontroller moet functioneren en wat de functionaliteiten zijn. Tot slot worden de requirements van de ontwerpopdracht samengevat in een tabel die aan het einde van de opdracht dienst kan doen als checklist voor de validatie.

## De 4-op-1-rij robot

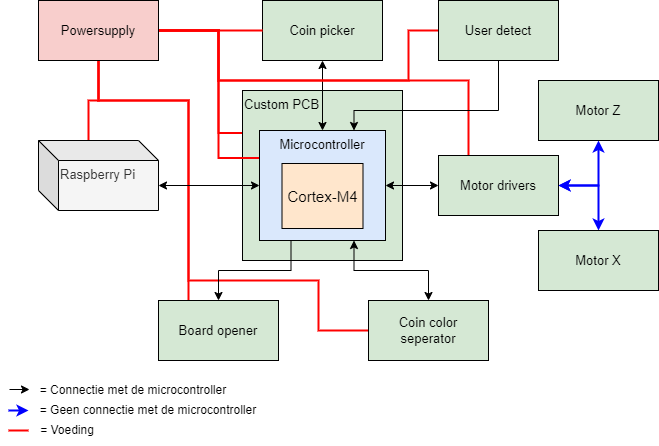
Voor een goed begrip van de huidige robot is in de volgende paragrafen uitgelegd hoe het systeem werkt en uit welke (hardware) onderdelen de robot bestaat.

### Spelverloop

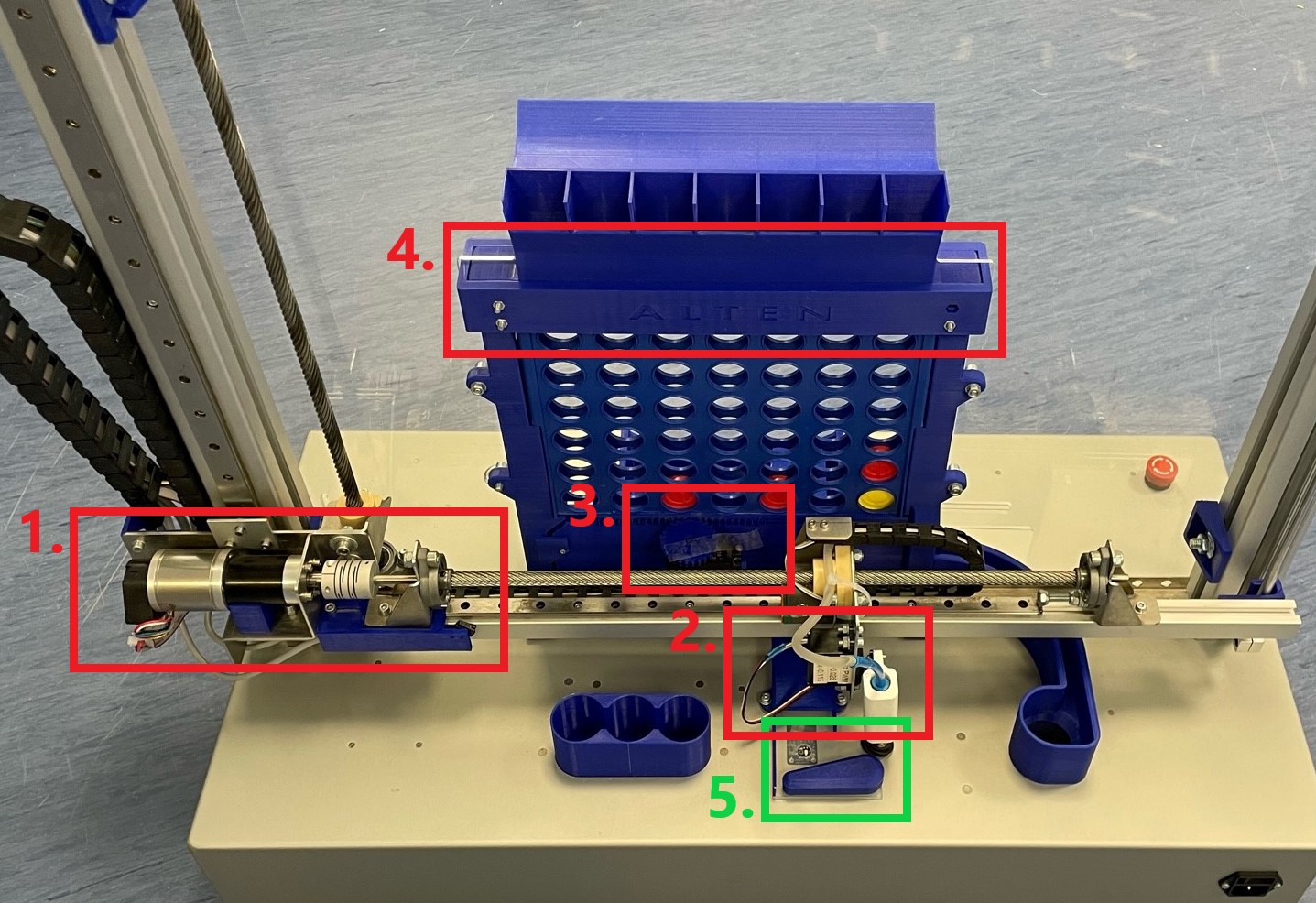
Het spelverloop van 4-op-1-rij staat vast. De speler en de robot doen om de beurt fiches in het spelbord tot een van de twee gewonnen heeft, er een gelijkspel optreedt of er sprake is van vals spel. Het gedrag van de robot kan beschreven worden in drie fases: een fase waarin de speler aan zet is, een fase waarin de robot aan zet is en een fase waarin het spel klaar is en alle fiches opgeruimd worden. Respectievelijk een human fase, een robot fase en een clean-up fase. Deze drie fases liggen aan de basis van het besturingssysteem.

### Hardware identificatie

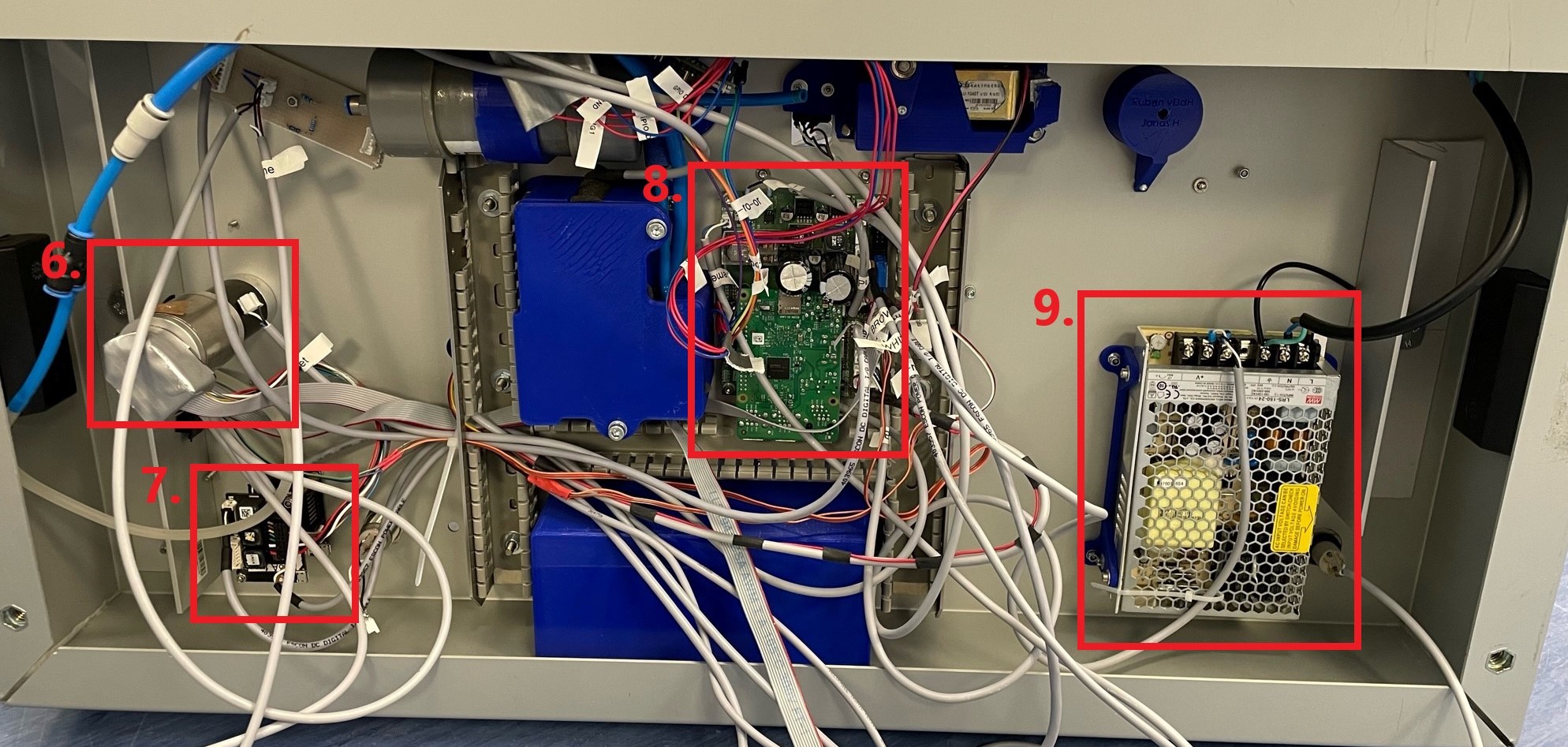
Het mechanische systeem en de elektronische hardware componenten van de 4-op-1-rij robot zijn aanwezig binnen ALTEN. In het blokdiagram (Figuur 4) wordt aangegeven uit welke onderdelen de robot is opgebouwd en hoe deze met het besturingssysteem verbonden zijn. In Figuur 5 is een bovenaanzicht weergegeven van de fysieke opstelling. In Figuur 6 is een onderaanzicht weergegeven die de overige componenten zichtbaar maakt. In Tabel 2 is beschreven welke componenten gebruikt worden en wat de functie is in het systeem.



Figuur 4 blokdiagram 4-op-1-rij robot



Figuur 5 Bovenaanzicht 4-op-1-rij



Figuur 6 Onderaanzicht 4-op-1-rij

Tabel 2 Componenten 4-op-1-rij

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Module | Componenten | Taak |
| 1 | Motor X | Motor (D3) met een rotary encoder (D2). | De Motor die de X-as aanstuurt zorgt ervoor dat de “Coin picker” zich over deze as kan verplaatsen. |
| 2 | Coin picker | Servo (D4) en vacuüm gripper (D6). | De vaccuum gripper kan de fiches oppakken en loslaten. De servo zorgt ervoor dat de vaccuum gripper kan bewegen. |
| 3 | Board opener | Servo (D4). | Deze servo zorgt ervoor dat aan het einde van het spel alle fiches uit het bord kunnen vallen. |
| 4 | User detect | Lichtsluis (7x LED met photodiode, D9). | De lichtsluis is verantwoordelijk voor het detecteren van een fiche dat in het spelbord gedaan is. |
| 5 | Coin color separator | Solenoid (D7) en RGB sensor (D8). | De Flipper en RGB sensor zijn de componenten die tijdens de clean-up fase van de robot actief zijn. De RGB sensor checkt de kleur van een fiche en de solenoid kan, doormiddel van een flipper, fiches naar de bak van de speler schieten. |
| 6 | Motor Z | Motor (D3) met een rotary encoder (D2). | De Motor die de Z-as aanstuurt zorgt ervoor dat de “Coin picker” zich over deze as kan verplaatsen. |
| 7 | Motor drivers | Motor driver X en Z (D1). | De motor drivers regelen het aansturen van de motoren en de communicatie met de microcontroller. |
| 8 | Custom PCB | PCB voor de microcontroller (D10) en een Raspberry Pi shield. | De microcontroller verzorgt de real-time processing, alle signalen die nodig zijn voor de aansturing van de hardware. De Raspberry Pi berekent met een algoritme de volgende zet van de robot. |
| 9 | Powersupply | (D5) | De powersupply zorgt dat het hele systeem voorzien is van een voeding. |

Door de datasheets van hardware componenten, de interne documentatie van de in eigen beheer ontworpen hardware en software te doorlopen, wordt duidelijk hoe de hardware componenten zich gedragen gedurende het spelverloop. Op basis hiervan kunnen de signalen en protocollen voor het besturingssysteem afgeleid worden.

## arc42

Bij het ontwerpen van een software architectuur is het noodzakelijk om een template te gebruiken die alle nodige diagrammen en stappen doorloopt.

Het streven van elk (embedded) software team is om foutloze code te schrijven, een gemotiveerd team te hebben en zorgen dat het systeem efficiënt werkt. Software architectuur speelt hierbij een belangrijke rol. Maar het komt ook voor dat de software architectuur niet goed gedocumenteerd, de code onoverzichtelijk of de architectuur simpelweg niet aanwezig is. Een paar problemen kunnen een software project belemmeren:

1. *Het niet bestaan van documentatie of verouderde documentatie.*

De documentatie van de software architectuur kan achterlopen op wat geïmplementeerd is, maar het kan ook zijn dat er geen documentatie beschikbaar is. Achterhaalde, of niet bestaande, software architectuur documentatie zorgt voor complicaties tijdens het implementeren, upgraden of aanpassen van de software omdat niet duidelijk is hoe de software is opgebouwd en communiceert.

1. *Onoverzichtelijke documentatie.*

Bestaande documentatie kan ook heel onoverzichtelijk zijn. Deze documentatie is vaak gemaakt zonder enig besef van het doel en is gemaakt door meerdere personen zonder coördinatie. Onoverzichtelijke architectuur is moeilijk te begrijpen, onderhouden of aan te passen.

1. *Te veel documentatie.*

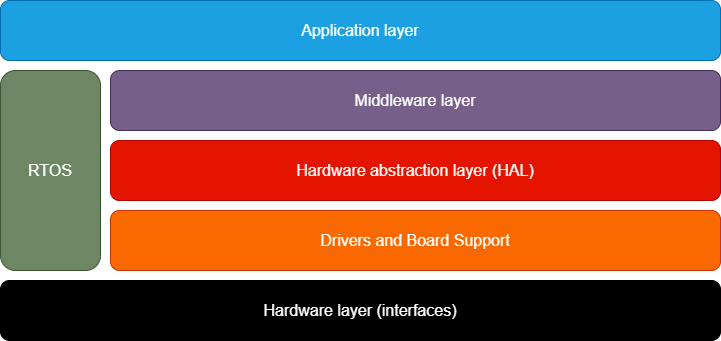
Een architectuur met te veel documentatie is ook niet optimaal. Dit kan resulteren in belemmeringen voor toekomstig gebruik, omdat het zoeken of aanpassen van informatie in het document niet gestructureerd kan gebeuren.

Voor de huidige software architectuur van de robot zijn ad 1 en 2 van toepassing. Na overleg met de software architect binnen ALTEN en de technical supervisor is er voor gekozen om het arc42 template (Figuur 7) te gebruiken voor het opstellen van een software architectuur. Arc42 is een template voor documentatie en communicatie van software of systeem architectuur. Het is een duidelijke, simpele en effectieve manier om structuur aan te brengen in software. Verder is het template geoptimaliseerd voor gebruikers en engineers. Deze template wordt veelvuldig toegepast door de software architecten van ALTEN. Hierdoor is het aannemelijk dat de template beantwoordt aan de vraag vanuit het bedrijfsleven. Daarnaast zorgt het ervoor dat architectuur informatie of belangrijke design keuzes goed beschreven worden en dat de gehele architectuur eenvoudig te onderhouden is [10].

Figuur 7 arc42 logo

## Modulariteit

In een embedded systeem worden software layers gebruikt. Het toepassen van deze layers zorgt voor modulaire software opbouw. Dankzij de layers kan de software tevens onafhankelijk van het systeem waar het op staat gebruikt worden. Dit zorgt dus voor flexibiliteit. Figuur 8 geeft de layers aan waaruit een embedded systeem bestaat.



Figuur 8 Embedded software layers

Hieronder wordt beschreven wat voor software zich in elke laag bevindt. Dit is van high level tot low level.

* *Applicationlayer*

De “Application layer” wordt gezien als een high level software layer. Deze layer definieert het systeem. Deze layer is niet uitwisselbaar want het is systeem specifiek. De Application layer is afhankelijk van, en wordt gemanaged en gedraaid door, de software uit de Middleware layer.

* *Middleware layer*

De “Middleware layer” is de layer die de communicatie tussen de Application layer en HAL, Drivers en Hardware layer regelt. Ook kan de Middleware layer de communicatie faciliteren tussen twee verschillende Application layers.

* *Hardware Abstraction Layer (HAL* [11]*)*

HAL dient voor een geformaliseerde wisselwerking tussen hardware (drivers) en software. HAL zorgt voor een goede communicatie tussen het systeem en de externe en interne hardware, waarbij de implementatie focust op het creëren van abstracte, high level functies. Dit betekent dat software die de functies gebruikt geen kennis hoeft te hebben van de functionaliteiten van de hardware.

* *Drivers*

De laag “Drivers” bevat software libraries die de hardware initialiseren en de toegang managen tussen hogere software lagen en de Hardware layer.

* *Hardware layer*

De “Hardware layer” staat voor de laag waar de hardware gedefinieerd wordt. Dit zijn vaak de fysieke pinnen van een microcontroller die naar de des betreffende hardware gaan.

Door voor het re-design gebruik te maken van layers, in combinatie met het template arc42, is het mogelijk om te voldoen aan de ontwerpopdracht van een gestructureerde en modulaire software architectuur.

## STM32H7 dual-core

Nummer 8 in Tabel 2 verwijst naar het huidige besturingssysteem op basis van een microcontroller en Raspberry-Pi. Op de Raspberry Pi draait het algoritme om de volgende zet van het systeem te bepalen en op de microcontroller draait het besturingssysteem (real-time processing). De huidige microcontroller is een singel-core microcontroller: STM32F303VCT6 met een Cortex-M4. In theorie voldoet deze microcontroller vooralsnog aan de systeem/ performance eisen van het design voor de huidige 4-op-1-rij robot. Het kan de hardware van de robot aansturen. Maar als er uitbreidingen plaatsvinden (zoals ethernetverbinding, beeldscherm of geluidseffecten) schiet deze microcontroller al snel te kort.

Voor de implementatie van de nieuwe software architectuur zal een dual-core microcontroller gebruikt worden. Deze is al eerder door ALTEN uitgekozen. Het gaat om de STM32H7 microcontroller, specifiek de STM32H755ZIT6U [12] van STMicroelectronics (ST). Deze bevind zich op het development board Nucleo-H755ZI-Q zoals weergegeven in Figuur 9. Een development board is ideaal om de software die geïmplementeerd wordt makkelijk en efficiënt te testen.

ST is een bedrijf dat gespecialiseerd is in microcontrollers (MCU’s), vermogenselektronica en flashgeheugen. Binnen dit project draait het om de microcontroller. ST heeft een breed scala aan microcontrollers beschikbaar. Van goedkopere, robuuste 8-bit MCU’s tot 32-bit Arm-based Cortex-M MCU’s met een uitgebreide keuze aan randapparatuur. De STM32H755ZIT6U dual-core microcontroller (in de rest van de document de STM32H7 dual-core) behoort tot de 32-bit Arm-based Cortex-M MCU’s. De STM32H7 bevat een Cortex-M7 en een Cortex-M4 core. De Cortex-M7 kan tot een maximum van 480 MHz draaien en de Cortex-M4 tot een maximum van 240 MHz. Dit valt binnen ST onder de categorie high-performance microcontrollers. De volledige lijst van specificaties kan gevonden worden op de site van ST [13].

Figuur 9 Nucleo-H755ZI-Q

Er is door ALTEN gekozen voor een dual-core microcontroller die tegemoet komt aan de toekomstvisie voor de 4-op-1-rij robot. De STM32H7 dual-core kan met de twee krachtige cores (Cortex-M7 en Cortex-M4) alle upgrades die gepland zijn werkelijk implementeren. Daardoor is het voor ALTEN een interessante optie en logische stap om het systeem hierop te implementeren om zo de kwaliteit en performance te waarborgen. Verder is het voor ALTEN de eerste keer dat er met een dual-core microcontroller gewerkt wordt. Daarmee dient dit project ook als een “proof of concept” voor het gebruik hiervan in volgende “in house” projecten. Veel projecten binnen ALTEN zijn gefocust op performance en factoren als kosten besparing en battery life zijn minder belangrijk.

De stap van een single core naar een STM32H7 dual-core microcontroller is een hardware upgrade die tegelijkertijd kan worden geïmplementeerd met een re-design van de software architectuur voor de 4-op-1-rij robot.

## Core verdeling

Uit de requirements volgt dat het systeem geïmplementeerd wordt op een STM32H7 dual-core microcontroller en dat het real-time processing en de game-handling van elkaar gescheiden zullen zijn. In het oude systeem draait het real-time processing op een Cortex-M4 en wordt de game-handling en het bepalen van de volgende zet uitgevoerd op de Raspberry Pi. In de nieuwe situatie blijft de Cortex-M4 real-time processing uitvoeren, omdat het principe van het spel 4-op-1-rij niet verandert en aanpassingen van de functionaliteit van de robot niet zijn voorzien. Binnen het project wordt de game-handling verplaats van de Raspberry Pi naar de nieuwe Cortex-M7. De Cortex-M7 is veel krachtiger dan de Cortex-M4 wat het mogelijk maakt om in de toekomst ook het algoritme voor de volgende zet naar de Cortex-M7 te verplaatsen. Vooralsnog zal het algoritme op de Raspberry Pi blijven omdat dit buiten de scope van het project valt. Verder is de Cortex-M7 ook geschikt om nieuwe hardware uitbreidingen te faciliteren.

## STM32CubeIDE

De keuze voor een STM32H7 dual-core heeft gevolgen voor de tool die nodig is om geschreven code naar de microcontroller te sturen. Vaak wordt hiervoor een Intergrated Development Enviroment (IDE) gebruikt. Voor de microcontrollers van ST zijn meerdere mogelijkheden voor een IDE.

* STM32CubeIDE
* IAR
* Keil

STM32CubeIDE is de IDE van ST zelf [14]. Deze is gratis en bevat alle opties die relevant zijn voor een microcontroller van ST. Ook bevat het een visuele omgeving waarbij de pin out kan worden ingesteld met signalen en protocollen. Door de STCubeMX software tool die bij de IDE zit wordt automatisch initialisatie/ configuratie software gegenereerd voor signalen en protocollen.

IAR en Keil zijn vergelijkbaar met elkaar. Beiden hebben een gratis versie maar het volledige pakket kost geld. De gratis versie is verouderd en heeft beperkingen. Zo kan niet op elke microcontroller geprogrammeerd worden en is er een maximale grootte voor het software bestand dat ge-upload kan worden.

Er is gekozen voor de IDE van ST, STM32CubeIDE, omdat deze van dezelfde leverancier komt als de STM32H7 dual-core en omdat met IAR en Keil als IDE niet op elke microcontroller kan worden geprogrammeerd. Daarnaast is de omvang van de data waaruit een systeem bestaat bij IAR en Keil begrensd.

Een van de grootste voordelen van de omgeving STM32CubeIDE is dat er gebruik kan worden gemaakt van de HAL van ST. Figuur 10 geeft aan welke software layers automatisch gegenereerd/ gebruikt worden door de STCubeMX software tool.



Figuur 10 STCubeMX software tool gegenereerde layers

Alle signalen en protocollen (pin specifieke configuratie) die van te voren gedefinieerd worden in de setup fase worden door de software tool gegenereerd in de HAL en Driver layers. De opmaak van de code (voor de Middleware, zoals ethernet) wordt ook voor een deel automatisch uitgevoerd door de STCubeMX.

HAL wordt toegepast om de development cycle te verkorten en gebruik te maken van libraries die al voldoende getest zijn. Daarnaast is met de keuze voor ST HAL rekening gehouden met software engineers, die met meerdere microcontrollers werken en de applicatie van het ene platform naar de andere moeten overzetten. Verder is een HAL bij uitstek geschikt om engineers met weinig hardware kennis toch met deze componenten te laten werken. Ze hoeven hiervoor geen specifieke kennis te hebben van de hardware details. De HAL wordt door ST meegeleverd. Alleen de pin-specifieke configuratie wordt gegenereerd. Bij de ST HAL library worden taken zoals UART en i2c verwerkt door een HAL en hoeft er dus niet op register niveau (low level) gewerkt te worden om deze protocollen werkend te krijgen.

## Conclusie

In dit hoofdstuk zijn onderzoeksvragen beantwoord en enkele (methodologische) keuzes gemaakt voor de uitvoering van het project. Met de bestudering van de robot is duidelijk geworden met welke hardwarecomponenten het nieuwe besturingssysteem rekening moet houden. Voor de nieuwe software architectuur van het besturingssysteem is gekozen voor het template arc42. Het template is standaard voor de projecten van ALTEN en het biedt een duidelijke, simpele en effectieve manier om structuur aan te brengen in software. Om de modulariteit van de architectuur te garanderen wordt gebruik gemaakt van software layers. Dankzij de layers kan de software, onafhankelijk van het systeem waar het op staat, gebruikt worden.

Vooraf is bepaald dat de softwarearchitectuur geïmplementeerd moet worden op een dual-core microcontroller. Onderzocht is wat de functionaliteiten zijn. Om de performance van de robot ook in de toekomst robuust te houden is besloten de game-handling en real-time processing van elkaar te gescheiden. Dit helpt het systeem overzichtelijk te houden en zorgt ervoor dat de twee cores optimaal benut worden. Ook is dit de eerste stap om het systeem op één microcontroller te implementeren. Doordat de Cortex-M7 erg krachtig is kan op termijn zonder aanvullende maatregelen het algoritme voor de volgende zet van de Raspberry Pi naar de Cortex-M7 gehaald worden.

Als programmeeromgeving is gekozen voor STM32CubeIDE van dezelfde leverancier als de dual-core microcontroller. Door de STCubeMX software tool die bij de IDE zit wordt automatisch initialisatie/ configuratie software gegenereerd voor signalen en protocollen. Dit versnelt de development cycle voor het project.

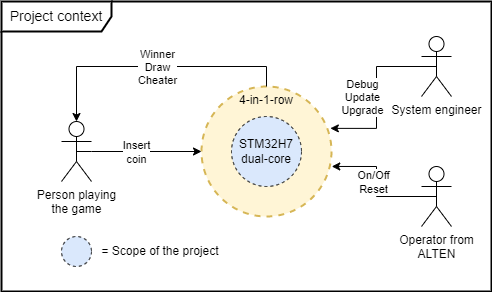
# Ontwerp software architectuur

In dit hoofdstuk komt het ontwerp van de software architectuur aan bod. De software architectuur is de hoofdopdracht van dit project. Er is gewerkt met de template arc42. Aan de hand van de template wordt beschreven hoe de software is opgebouwd, onderling communiceert, waar data naar toe gaat en hoe het zich gedraagt als het systeem operationeel is (Software Architectuur Document, SAD). In de volgende paragrafen worden de belangrijkste hoofdstukken uit het SAD toegelicht die bepalend zijn voor de implementatie van het systeem, waarbij diagrammen en begeleidende tekst de lezer helpen om de stappen te volgen. Daarnaast worden de designkeuzes toegelicht. Het complete SAD is opgenomen in bijlage III. Software Architecturale Document.

## Systeem context SAD

In het hoofdstuk “Systeem context” binnen het SAD, is beschreven wat de scope van het project is en welke externe en interne hardware en personen met het systeem verbonden zijn (Project context). Het is van belang om dit in kaart te brengen omdat er in het design rekening gehouden moet worden met de interfaces van de verschillende externe en interne hardware componenten. Het is cruciaal om alle interfaces goed te doorgronden en te beschrijven voordat begonnen kan worden aan een software architectuur document.

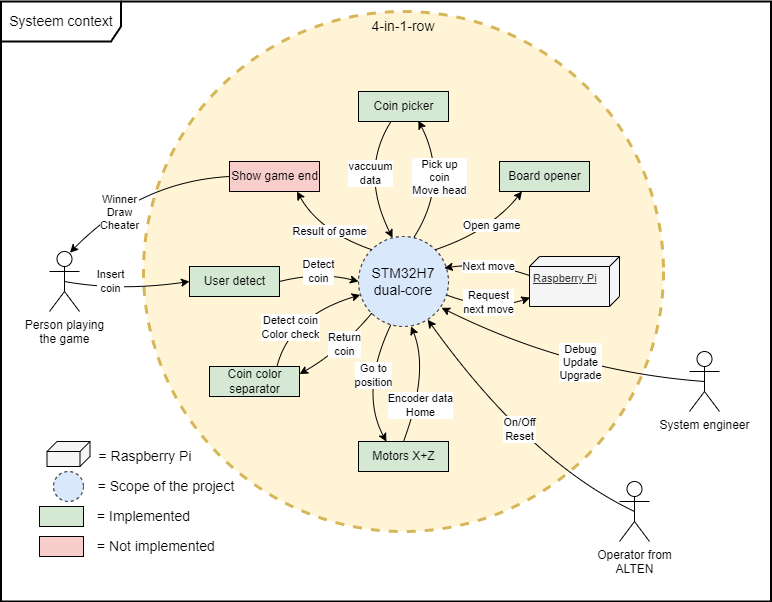
Allereerst is een weergave gemaakt van alle externe hardware interfaces en personen buiten het systeem (zie Figuur 11). Bij de 4-op-1-rij robot is geen externe hardware aanwezig. De personen die de 4-op-1-rij gebruiken zijn “Speler” (Person playing the game), “Systeem engineer” (System engineer) en “Operator”. De “Speler” is de persoon die het spel speelt. Deze werpt elke keer als hij/zij aan de beurt is een fiche in een kolom naar keuze en krijgt als het spel afgelopen is te zien wat het resultaat van het spel is. De “Systeem engineer” is de persoon die het systeem onderhoudt. Deze zorgt voor een update of upgrade als dat nodig is en debugt het systeem als er een fout optreedt. De “Operator” is de persoon vanuit ALTEN die de 4-op-1-rij robot aan/uit zet of reset.



Figuur 11 Project context

Nadat de project context is opgesteld is er gekeken wat voor interne hardware interfaces binnen het systeem verbonden zijn met de project scope (blauwe cirkel). Door het onderzoek (Vooronderzoek) voorafgaand aan het begin van de software architectuur kan een diagram opgesteld worden die is weergegeven in Figuur 12. Dit is de Systeem context. In de Systeem context is te zien welke hardware componenten zijn verbonden met de STM32H7 dual-core en wat voor opdrachten deze ontvangen en terug sturen. Dit geeft een globaal overzicht. Het is een tactische weergave om met alle stakeholders te kunnen overleggen zonder te veel in technische/operationele details te treden.

Naast de Systeem context is er ook een Technische context opgesteld. De technische context geeft aan hoe de interne hardware interfaces zijn verbonden met STM32H7 dual-core en wat voor signalen/ protocollen gebruikt worden voor de verbindingen. Een technische context is van belang voor het team dat operationeel aan de slag gaat met de software modules voor de hardware. De Technische context is opgenomen in bijlage III. Software Architecturale Document.



Figuur 12 Systeem context

## Building block view SAD

Het hoofdstuk “Building Block View”(BBV) van het SAD gaat dieper in op de in het hoofdstuk “Systeem context” gepresenteerde scope van het project. Het idee van een BBV is om stap voor stap op het systeem in te zoomen. Dit is een goede manier om een systeem gestructureerd te ontleden. Zo wordt duidelijk uit welke software modules het systeem is opgebouwd, hoe ze zijn verbonden en waar deze zich bevinden. Afwisselend wordt gewerkt met “black” en “white boxes” waarbij de white boxes de interne details onthullen van de black boxes. Dit wordt gedaan doormiddel van levels. Black boxes zijn modules waarvan alleen de in- en outputs bekend zijn maar de interne details niet. Als een level uitputtend is beschreven kan gekozen worden om een black box verder uit te werken in een nieuwe white box.

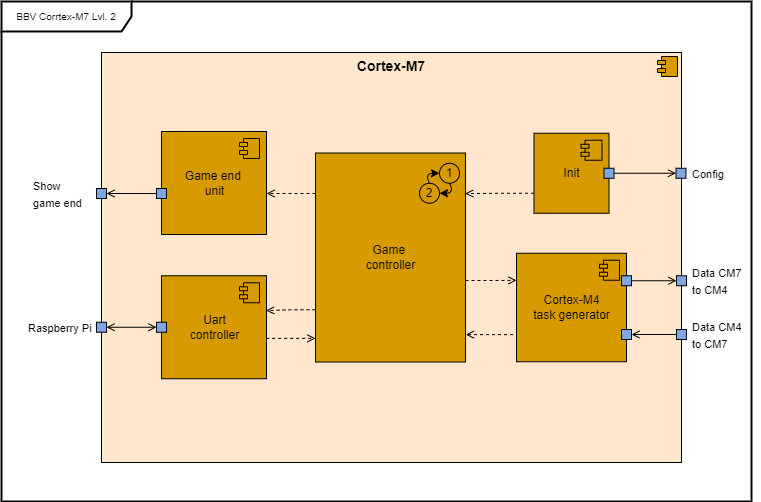
Voor dit project is het eerste level de Systeem context (level 0, Figuur 12). Hierbij is de STM32H7 dual-core een “black box”. In level 1 wordt deze STM32H7 dual-core een “white box” die weer bestaat uit meerdere blokken als “black box”. In level 2 worden de “black boxes” uit level 1 “white boxes”. Tijdens elke stap worden in- en outputs, connecties en opdrachten beschreven van de “black boxes”. Dit is het juiste aggregatieniveau om met de stakeholders in overleg te gaan zonder te veel in detail te treden op implementatie niveau. Nadat BBV is doorlopen en besproken kunnen vervolgens de modulaire software modules opgesteld worden.



Figuur 13 BBV STM32H7 level 1

Zoals weergegeven in Figuur 13 wordt level 1 van de BBV de STM32H7 dual-core beschouwd als “white box” en de Cortex-M7 en Cortex-M4 als “black box”. In het figuur is aangegeven welke hardware interfaces naar de Cortext-M4 gaan die gebruikt worden voor het real-time processing. De Cortex-M7 handelt het spelverloop af en heeft een verbinding met de Raspberry PI. De Raspberry PI rekent doormiddel van een algoritme uit wat de volgende stap van de robot moet zijn en geeft deze vervolgens door aan de Cortex-M7. De verbinding tussen de Cortex-M7 en Cortex-M4 geeft de dual-core communicatie weer. De communicatie is onmisbaar voor het uitwisselen van data tussen de twee cores die parallel van elkaar code uitvoeren.

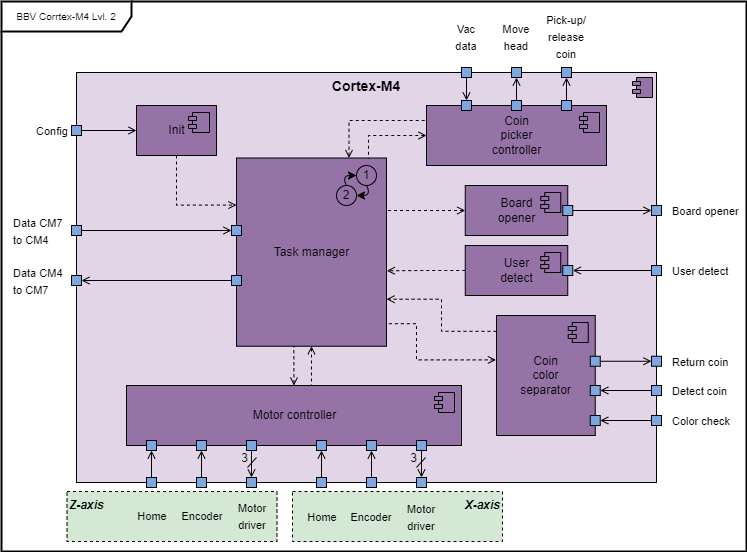
In Figuur 14 en Figuur 15 zijn de Cortex-M7 en Cortex-M4 nog verder in detail ontleed (level 2) en weergeven als een “white box”. Voor de Cortex-M7 wordt level 2 beschouwd als het laagste level omdat de blokken binnen dit niveau voldoende representatief zijn ontleed voor de uiteindelijke software modules binnen de Cortex-M7.



Figuur 14 BBV Cortex-M7 level 2

Tabel 3 Omschrijving BBV Cortex-M7 level 2

|  |  |
| --- | --- |
| Black box | Beschrijving |
| Game controller | De Game controller is verantwoordelijk voor het spelverloop doormiddel van een state machine. |
| Init | Wordt één keer gedraaid om de Cortex-M7 te initialiseren en op te starten. |
| UART controller | De UART controller implementeert alle communicatie via UART. |
| Cortex-M4 task generator | De Cortex-M4 task generator implementeert de communicatie met de Cortex-M4. |
| Game end unit | De Game end unit implementeert alle communicatie naar een output voor de speler. |

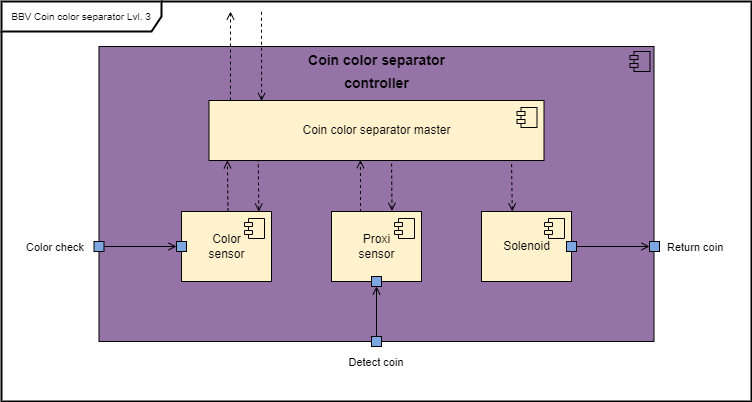


Figuur 15 BBV Cortex-M4 level 2

Tabel 4 Omschrijving BBV Cortex-M4 level 2

|  |  |
| --- | --- |
| Black box | Beschrijving |
| Task manager | De Task manager is verantwoordelijk voor het verloop van de uit te voeren opdrachten van de Cortex-M7 doormiddel van een state machine. |
| Init | Wordt één keer gedraaid om de Cortex-M4 te initialiseren en op te starten. In deze fase worden de sensoren getest op aanwezigheid en de motoren voeren een “homing” protocol uit. |
| Motor controller | De Motor controller implementeert alle communicatie met de motor drivers, encoders, de PID regelaars en home stop. |
| Coin color  separator | De Coin color separator implementeert alle functies voor het afhandelen van het scheiden van de fiches. |
| User detect | De User detect implementeert alle functies voor het afhandelen van de sensoren voor de input van de speler. |
| Board opener | De Board opener implementeert alle functies voor het afhandelen van het open van het 4-op-1-rij bord als het spel klaar is en de clean-up fase begint. |
| Coin picker  controller | De Coin picker controller implementeert alle functies voor het afhandelen van het oppakken en loslaten van een fiche. |

Voor de Cortex-M4 wordt nog verder ingezoomd (level 3) op de volgende blokken: Coin color separator, Motor controller en Coin picker controller. Dit is noodzakelijk omdat deze blokken meer belangrijke sub software modules bevatten voor de aansturing van de robot. In Figuur 16 wordt de Coin color separator weergegeven als “white box”. Voor de overige sub-blokken wordt verwezen naar het volledige SAD in de bijlage III. Software Architecturale Document.



Figuur 16 BBV Coin color seperator level 3

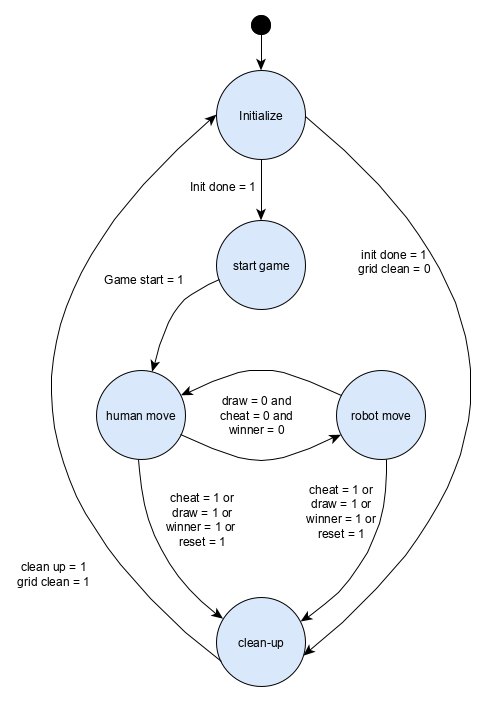
Tabel 5 Omschrijving BBV Coin color seperator level 3

|  |  |
| --- | --- |
| Black box | Beschrijving |
| Coin color separator  master | De Coin color separator master is verantwoordelijk voor het ontvangen van de sensor data en het aansturen van de solenoid. |
| Color sensor | Vraagt de data op over de kleur van een fiche. |
| Proxi sensor | Krijgt data of er een fiche aanwezig is. |
| Solenoid | Is verantwoordelijk voor het activeren van de flipper. |

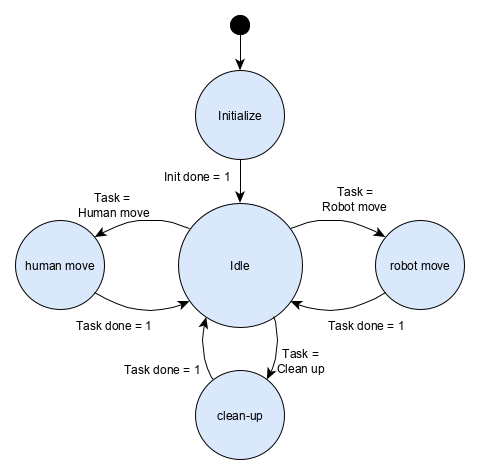
## Runtime view SAD

Het hoofdstuk “Runtime view” visualiseert hoe de, in het hoofdstuk Building block view beschreven, software modules met elkaar communiceren als het systeem operationeel is. De diagrammen geven aan welke software modules wat in de tijd met elkaar communiceren. De diagrammen bestaan uit state machines [15] en sequence diagrammen [16]. De state machine en sequence diagrammen zijn van cruciaal belang om te visualiseren hoe de robot zich gedraagt als het systeem operationeel is. Door daarnaast alle processtappen modelmatig te beschrijven wordt duidelijk welke data software modules ontvangen en moeten uitsturen.

Een state machine is een abstract model voor het gedrag van het systeem. Het model bestaat uit een eindig aantal toestanden (states) waar het systeem zich in kan bevinden en elke toestand heeft een of meerdere overgangen naar volgende states. Deze overgangen worden bepaald door de input die het systeem krijgt. Een sequence diagram toont interacties tussen software modules die in tijdsvolgorde zijn gerangschikt. Het diagram geeft de software modules weer die betrokken zijn bij één van de states van de state machine. Het diagram laat de volgorde van communicatie zien, die tussen de software modules wordt uitgewisseld om de functionaliteit van de state uit te voeren.

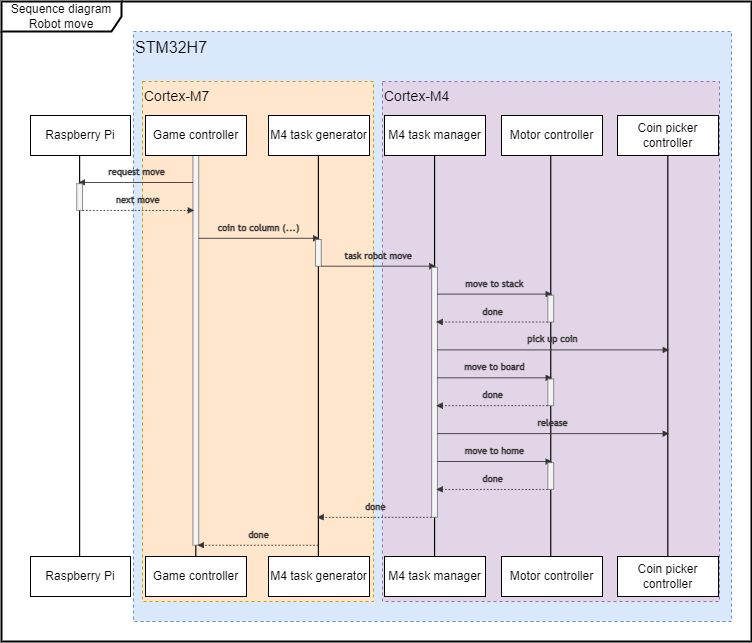
De state machine voor het spelverloop is te zien in Figuur 17. Deze draait op de Cortex-M7 in het blok “Game controller” (Figuur 14). De eerste state is de “initialize” state. In deze state wordt het hele systeem opgestart en geïnitialiseerd. Zodra deze state klaar is gaat het systeem in de “start game” state. In deze state wacht het systeem tot het spel begint. Als het spel gestart is, is het de beurt aan de speler om een fiche in het 4-op-1-rij spel te doen en bevind het systeem zich in de “human move” state. Hierna gaat de beurt naar de robot en wijzigt de state naar “robot move”. In deze state voert de robot een berekende zet uit. Na elke beurt controleert de robot of er een winnaar is, of er vals gespeeld is, of er sprake is van gelijk spel. Als dit niet het geval is gaat de beurt weer naar de speler die aan zet is en bevindt het systeem zich in de bijbehorende state (human move of robot move). Zodra het spel is afgelopen gaat de clean-up state van start en worden alle fiches opgeruimd. De rode fiches gaan naar de robot en de gele naar de bak van de speler.

Figuur 17 State machine spelverloop

De state machine voor het real-time processing van de hardware is te zien in Figuur 18. Deze draait op de Cortex-M4 in het blok “Task manager” (Figuur 15). De eerste state is de “initialize” state. In deze state worden de hardware componenten geconfigureerd en ingesteld. Daarna heeft de Task manager een “Idle” state. In deze state wordt gewacht tot de Task manager een task krijgt van de Cortex-M7. Zodra een task binnenkomt wordt aan de hand van de task de volgende state bepaald. Dit kan de “human move”, “robot move” of “clean-up” state zijn. Elk van deze states stuurt de hardware aan om de task te voltooien. Als de task voltooid is gaat de Task manager terug naar de Idle state.

Figuur 18 State machine real-time processing

Elke state van de state machine uit Figuur 17 kan weergegeven worden in een sequence diagram. Dit is om aan te geven wat er in elke state gebeurt en hoe de communicatie tussen software modules verloopt. In Figuur 19 is de robot move uitgewerkt. De andere sequence diagrammen zijn te vinden in het SAD opgenomen in bijlage III. Software Architecturale Document.



Figuur 19 Sequence diagram robot move

De robot move begint met het opvragen door de Game controller van de volgende zet aan de Raspberry Pi. De Raspberry Pi geeft de volgende zet terug waarna de Game controller aan de M4 task generator doorgeeft welke opdracht er gemaakt moet worden. Zodra de opdracht gemaakt is stuurt de M4 task generator de opdracht naar de M4 task manager op de Cortex-M4. Om een fiche te spelen moet de robot eerst naar de plek waar de fiches zijn opgeslagen. Zodra de robot bij de opslag is aangekomen moet een fiche worden opgepakt. Nu kan de robot naar het bord bewegen om vervolgens het fiche los te laten. Daarna moet de robot weer naar zijn “home” stand. Vervolgens geeft de M4 task manager door aan de M4 task generator dat de beurt voltooid is. Tot slot geeft de M4 task generator het signaal dat de Game controller naar de volgende state kan gaan.

## Deployment view SAD

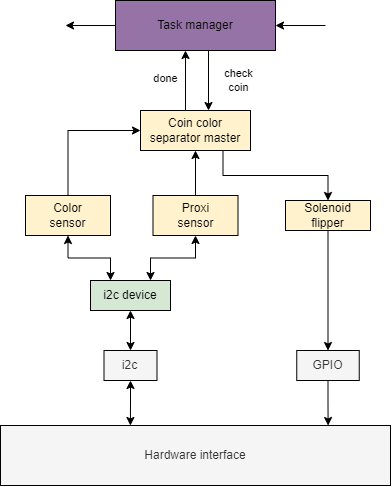
Figuur 20 Pin out STM32H7 dual-core

In het hoofdstuk “Deployment view” wordt beschreven wat er nodig is om de software daadwerkelijk te implementeren. Daarvoor wordt een microcontroller gebruikt. Het belangrijkste component is het development board met de STM32H7 dual-core microcontroller. Verder wordt de pin out gedefinieerd. De pin out geeft aan welke fysieke uitgangen van de microcontroller naar de hardware componenten gaan en ook welke signalen/ protocollen gebruikt worden (de groene pinnen in Figuur 20) om de robot aan te sturen. Een pin kan niet elk type signaal/ protocol uitvoeren. De pinnen zijn zo ingesteld dat alle signalen/ protocollen die nodig zijn om het systeem te laten draaien beschikbaar zijn. Ook zijn er pinnen gealloceerd voor ethernet. Dit is een feature die op dit moment nog niet geïmplementeerd is. In de toekomst is dit wel de bedoeling en dan is het belangrijk dat de pinnen die nodig zijn om ethernet te gebruiken niet bezet zijn voor andere taken. De volledige omschrijvingen en diagrammen zijn opgenomen in het SAD (bijlage III. Software Architecturale Document).

## Modulaire software modules SAD

Op basis van de requirements is gekozen voor een modulaire opbouw van het systeem. In hoofdstuk 5.2 Building block view is beschreven welke software modules noodzakelijk zijn om een werkende 4-op-1-rij robot te implementeren. In het hoofdstuk Modulaire software modules wordt uitgelegd hoe de modules van high level tot low level (Application layer tot Hardware layer, Figuur 8) met elkaar samen hangen. Door gebruik te maken van de embedded software layers creëer je herbruikbare modules. Representatief hiervoor zijn de HAL en driver layers, maar ook de modules uit de Middleware kunnen hergebruikt worden. De samenhang wordt gevisualiseerd aan de hand van diagrammen. Door deze diagrammen op te stellen wordt duidelijk wat de scope van de verschillende software modules is.

Er zijn generieke modules opgesteld om de HAL layer nog meer abstract (high level, Middleware) te maken. Deze generieke software modules van de signalen/ protocollen voor de communicatie met de hardware zijn zo ontworpen, dat ze ook in andere systemen en/ of projecten gebruikt kunnen worden. De benodigde input van generieke modules, bijvoorbeeld i2c\_device, is altijd hetzelfde. De verwerking van de data is hierdoor gestandaardiseerd. Bovendien zijn er modules die systeem specifiek zijn. Deze modules zijn modulair omdat ze functies van elkaar scheiden. Immers, de modules hoeven niet op de hoogte te zijn van elkaars functionaliteiten of taken om hun werk te doen. Doordat taken in de software specifieke modules gescheiden zijn, beantwoordt het systeem aan het gevraagde overzicht en inzicht en de eenvoud van implementatie. Ook kan er met een modulaire opbouw door meerdere mensen tegelijkertijd aan het systeem gewerkt worden. De modules hebben een vaste interface waardoor elke module zelfstandig kan worden aangepast. Door modulaire software modules kan er bijvoorbeeld bij een upgrade een sensor vervangen worden, zonder alle code aan te passen. Alleen de module met de sensor specifieke details hoeft aangepast te worden.

**Modulariteit Coin color seperator**

In Figuur 21 is het diagram weergegeven voor de software modules van de Coin color separator. De modules uit het hoofdstuk 5.2 Building block view worden gecombineerd met de standaard HAL. Het blok “GPIO” en “i2c” behoren tot standaard ST HAL software modules waarbij de configuratie specifieke onderdelen worden gegenereerd. Deze blokken initialiseren de General Purpose I/O pinnen (GPIO) en de i2c bus. De andere blokken zijn software modules die ontwikkeld moeten worden. Het blok “i2c device” is een generieke module die gemaakt wordt om er voor te zorgen dat elke sensor die via de i2c bus verbonden is enkel hoeft aan te geven of er data verstuurd of opgevraagd moet worden. Zo hoeven de modules voor de sensor niet alle standaard functies van een i2c verbinding op de hoogte te zijn want dit wordt afgehandeld door het blok “i2c device”. Door deze modulaire benadering kunnen de modules hergebruikt worden en is het eenvoudig om een extra module, voor bijvoorbeeld een extra i2c sensor, toe te voegen.

Figuur 21 Software layers Coin color separato

**Modulariteit Cortex-M7**

De Cortex-M7 wordt gebruikt voor de spel bepaling van de 4-op-1-rij robot. Er is gekozen voor een Round robin with interrupt methode om het verloop van het spel te implementeren in een state machine (Figuur 17) (hoofdstuk 6.1). De “Game controller” is binnen de Cortex-M7 de module die de state machine uitvoert. De Game controller is verantwoordelijk voor het bijhouden van de state waar het systeem zich in bevindt en, aan de hand van inputs te bepalen wat de volgende state moet worden. Voor de overige taken van de Cortex-M7 zijn andere modules gekozen. De modules “UART controller”, “Cortex-M4 task generator” en de “Game end unit” zorgen ervoor dat alle input/ output die de Game controller gebruikt worden gescheiden van elkaar. Deze modules handelen alle nodige communicatie af tussen de Game controller en de hardware zodat de Game controller alleen deze modules hoeft aan te roepen om een taak uit te voeren.

**Modulariteit Cortex-M4**

Op de Coretex-M4 draait alle software die de hardware aanstuurt en afhandelt. Om binnen deze core de gevraagde modulariteit te creëren is er voor gekozen om voor elk uniek hardware blok (Figuur 15) een eigen software module te ontwikkelen. Om al de software modules van de hardware goed te managen is er voor gekozen om een “Task manager” te implementeren. Door gebruik te maken van een “Task manager” kunnen functionaliteiten van andere onderdelen modulair gebouwd worden. De functionaliteiten hoeven immers niet van elkaars bestaan/status af te weten. Die informatie wordt bijgehouden door de Task Manager. Ook de Task manager zal werken doormiddel van een state machine.

## Conclusie

Het opstellen van een gestructureerde software architectuur moet leiden tot een toekomstbestendige 4-op-1-rij robot. Het moet mogelijk zijn om op een eenvoudige manier wijzigingen en aanpassingen in requirements door te voeren zonder de werking van het systeem te frustreren. Met het opstellen van een SAD is voldaan aan de opdracht om een gestructureerde software architectuur te ontwikkelen. Het template van arc42 heeft er voor gezorgd dat alle benodigde diagrammen, functionaliteiten en design keuzes zorgvuldig gemaakt, beschreven en weergegeven zijn. Elke soft- of hardware developer die met de 4-op-1-rij robot gaat werken kan begrijpen hoe het systeem werkt en welke software modules geïmplementeerd zijn. Daarnaast zorgt modulariteit voor een robuust systeem dat eenvoudig kan worden aangepast. Het resulteert in herbruikbare modules en snellere development tijd.

# Implementatie en testen

Om de software architectuur te testen zijn enkele unit tests en demo’s ontwikkeld. De unit tests valideren de noodzakelijke software modules van de BSP. De demo’s moeten de modulariteit en werking van de software van het besturingssysteem onderschrijven en aantonen dat de juiste design keuzes zijn gemaakt. In eerste instantie is gekozen om het fundament van het BSP te testen, namelijk de dual-core communicatie. Daarna zijn afzonderlijk de BSP modules getest voor i2c, UART en Pulse Width Modulation (PWM) communicatie. Parallel zijn demo’s ontwikkeld waarin stap voor stap, op basis van de geteste en gevalideerde BSP modules, is toegewerkt naar het complete besturingssysteem van de 4-op-1-rij robot.

## Implementatie methode

In hoofdstuk 4.5 Core verdeling is uitgelegd wat de functie is van elke core. Omdat er gewerkt wordt met een dual-core implementatie zal er altijd een vorm van dual-core communicatie geïmplementeerd moeten worden. Voor de technische uitwerking hiervan zie paragraaf 6.2 Dual-core communicatie. Daarnaast moet er een methode gekozen worden waarmee de software uitgevoerd zal worden (software loop). Vier mogelijke opties zijn onderzocht (Bijlage V. Implementatie methodes voor een software loop):

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

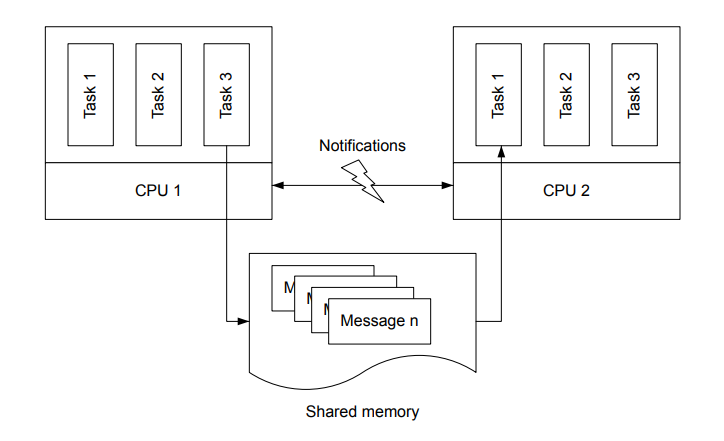
Het uitgangspunt voor een optimaal design is de keuze voor de meest eenvoudige implementatiemethode. De methode moet voldoen aan de performance eisen van het systeem.

Binnen het project is gekozen voor een Round robin with interrupt implementatie methode. De implementatie van de huidige 4-op-1-rij robot is ook gedaan met een Round robin with interrupt. Deze voldoet aan de performance eisen. De volgorde van de fases (state machine) is altijd hetzelfde en kan gemanaged worden door een Round robin. Echter, omdat er prioritaire taken zijn zoals een harde timing deadline voor de control loop van de motoren, de inworp van een fiche en dual-core communicatie zijn interrupts nodig. De deadline kan afgevangen worden d.m.v. een timer ISR. Waar wel rekening mee gehouden moet worden is de false data die kan ontstaan door het gebruik van interrupts. Bijlage VI. False data legt in detail uit hoe dit voorkomen kan worden.

## Dual-core communicatie

Dual-core communicatie is een cruciaal onderdeel van het systeem. De twee cores draaien onafhankelijk van elkaar. Er is standaard geen ingebouwde communicatie tussen de beide cores, ze kunnen geen informatie met elkaar uitwisselen. Omdat de Cortex-M7 de game handling uitvoert en de Cortex-M4 de real-time processing is communicatie nodig. Voor de communicatie en synchronisatie tussen de twee cores moet dus een oplossing bedacht worden.

De basis om communicatie te realiseren is een shared memory. Zoals weergegeven in Figuur 22 wordt er verder gebruik gemaakt van een notificatie lijn om data uit te wisselen tussen de twee cores. De notificatie lijn zorgt er voor dat de toegang tot het gedeeld geheugen gesynchroniseerd is tussen de beide cores. De synchronisatie voorkomt corruptie van de data en dat beide cores tegelijkertijd in het geheugen schrijven.



Figuur 22 Dual-core communicatie [17]

### Shared memory

De eerste stap bij het implementatie van de communicatie is het kiezen van een stuk shared memory dat beschikbaar en toegankelijk is voor beide cores. De STM32H7dual-core heeft als geheugen architectuur een “symmetric memory-mapping”. Deze architectuur zorgt ervoor dat een groot deel van het beschikbare geheugen voor beide cores toegankelijk is. Tabel 6 geeft aan dat ongeveer 82% van het geheugen direct beschikbaar voor beide cores. De Cortex-M4 heeft een Master Direct Memory Access (MDMA) controller nodig om bij het geheugen ITCM en DTCM te komen.

Tabel 6 Geheugen toegang [17]

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

Binnen dit project is SRAM4 gekozen als shared memory. SRAM4 bevindt zich in het D3 domein dat voor beide cores beschikbaar blijft, ook als één van de cores zich in “low-power” modus bevindt of zelfs uit staat. Het D1 respectievelijk D2 domein is alleen beschikbaar als de bijbehorende core actief is. Daarbij heeft SRAM4, met een grootte van 64Kbytes, voldoende ruimte om de data op te slaan. In bijlage IV. Memory en bus architectuur STM32H7 dual-core is een schematisch overzicht opgenomen van alle drie de domeinen.

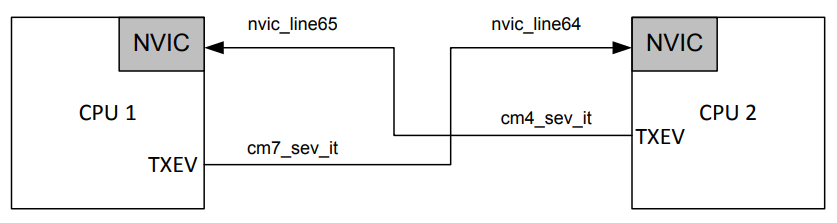
### Notificaties

De tweede stap bij het implementeren van de communicatie is het kiezen van een mechanisme voor het notificeren van de cores. Er zijn twee notificatiemechanismen onderzocht. De keuze wordt bepaald op basis van de meest toepasselijke manier voor de 4-op-1-rij implementatie. Door te kiezen voor een notificatie van de ene core naar de ander wordt de consistentie gegarandeerd en verlaagd het de tijd die het systeem anders kwijt zou zijn aan het constant checken of er nieuwe data beschikbaar is. Verder zorgt het synchronisatie.

De STM32H7 dual-core heeft twee oplossingen. Beide oplossingen gebruiken een “hardware interrupt”. Dit is noodzakelijk voor een real-time notificatie tussen beide cores. De twee oplossingen hebben de volgende eigenschappen:

1. *EXTI software interrupt and event registers, CPU send-event instruction (SEV).*

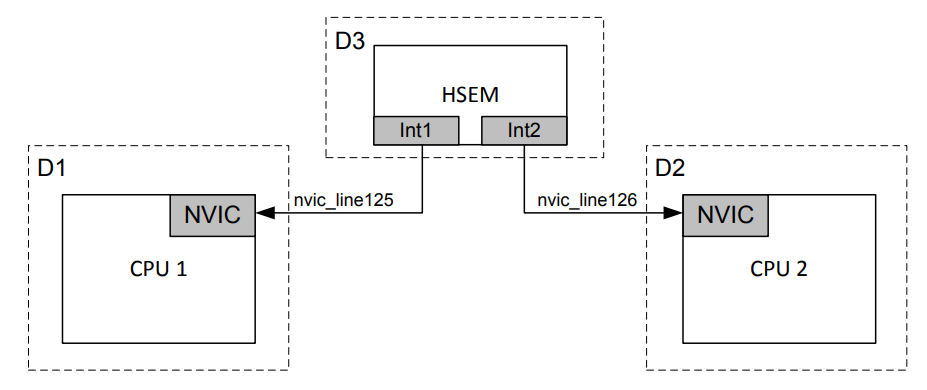
EXTI en SEV zijn eenvoudige interrupts. Figuur 23 laat zien dat beide cores een directe verbinding met elkaar hebben. De lijnen zijn verbonden met de Nested Vector Interrupt Controller (NVIC). De NVIC voert een ISR uit als een interrupt gedetecteerd wordt.



Figuur 23 EXTI en SEV dual-core communicatie [17]

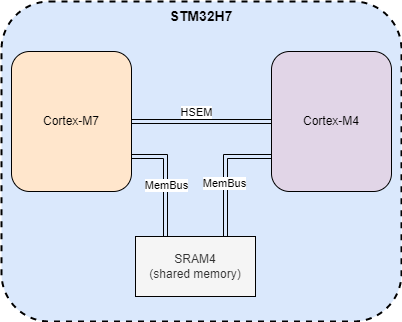
1. *Hardware Semaphore (HSEM) free interrupt.*

De HSEM [18] maakt gebruik van een HSEM controller (Figuur 24). Een HSEM werkt met een “lock/ free” mechanisme. Dit houdt in dat een core een HSEM kan locken. Zolang de HSEM is gelocked kan de andere core geen toegang krijgen tot de HSEM. Pas als de core de HSEM unlocked, wordt er een interrupt naar de andere core gestuurd. In de core die de interrupt ontvangt wordt een ISR uitgevoerd door de NVIC. Vanaf dat moment is de HSEM. Dit is een “handshake” zodat de cores niet tegelijkertijd een taak gaan uitvoeren en het geldt ook als notificatie mechanisme. Verder is het bij een HSEM mogelijk om aan de interrupt een ID tussen de 0 en 31 mee te geven, waardoor een notificatie ook wisselende opdrachten kan initiëren.



Figuur 24 HSEM dual-core communicatie [17]

Binnen het project is gekozen voor het mechanisme van een HSEM. De HSEM heeft dezelfde functionaliteit als EXTI interrupt en SEV maar heeft daarnaast als belangrijk voordeel dat er een ID meegegeven kan worden. De taken die de Cortex-M4 moet uitvoeren worden vooraf gedefinieerd en voorzien van een uniek ID (0-31). Op deze manier weet de Cortex-M4 welke taak uitgevoerd moet worden als de interrupt aankomt. In Figuur 25 is de implementatie van de HSEM binnen dit project weergegeven.



Figuur 25 dual-core implementatie STM32H7 dual-core

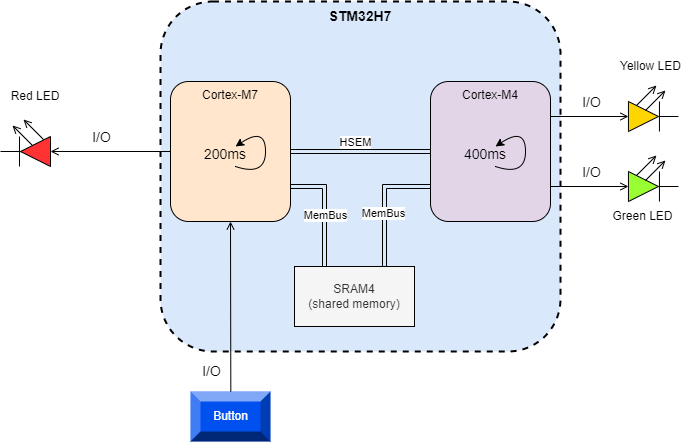
### Implementatie dual-core communicatie

Het eerste onderdeel van de BSP betreft implementeren van de dual-core communicatie applicatie. Figuur 26 geeft een schematisch overzicht van de componenten en hoe ze met elkaar verbonden zijn.

De applicatie is op twee manieren geïmplementeerd.

1. Het delen van informatie tussen de Cortex-M7 en Cortex-M4 loopt via shared memory (SRAM4);
2. De twee cores kunnen doormiddel van verschillende HSEM’s taken naar elkaar sturen.

In beide gevallen heeft de Cortex-M7 een delay van 200ms per software loop en de Cortex-M4 een van 400ms. De rode LED (Cortex-M7) knippert met een frequentie van 2,5Hz.



Figuur 26 Schematisch overzicht demo dual-core communicatie

*Ad 1) Shared memory unit test*

Bij het testen van shared memory zal de Cortex-M4 checken welke waarde in SRAM4 staat. In Tabel 7 is te zien welke waarde in shared memory staat dat zorgt voor het patroon van de gele en groene LED. Elke druk op de button (Cortex-M7) verhoogd de data in SRAM4 tot uiteindelijk de maximale waarde is bereikt (3). Daarna begint de data weer op 0. Daarnaast wordt bij elke druk op de button ook een HSEM verstuurd naar de Cortex-M4. Bij het ontvangen van de HSEM checkt de Cortex-M4 welke data in shared memory staat en voert een patroon uit.

Tabel 7 Mogelijke patronen gedeelde data

|  |  |  |
| --- | --- | --- |
| Data SRAM4 | Gele LED | Groene LED |
| 0 | Aan | Aan |
| 1 | Knippert met een frequentie van 1,25Hz | Uit |
| 2 | Uit | Knippert met een frequentie van 1,25Hz |
| 3 | Knippert met een frequentie van 1,25Hz | Knippert met een frequentie van 1,25Hz |
| onbekend | Uit | Uit |

*Ad 2) HSEM unit test*

Bij het testen zal de Cortex-M4 verschillende HSEM ontvangen. In Tabel 8 is te zien welke HSEM hoort bij een bepaald patroon van de gele en groene LED. De button verbonden met de Cortex-M7 zorgt bij deze test er voor dat alle HSEM’s doorlopen worden.

Tabel 8 Mogelijke patronen HSEM

|  |  |  |
| --- | --- | --- |
| HSEM | Gele LED | Groene LED |
| HSEM-1 | Aan | Aan |
| HSEM-2 | Knippert met een frequentie van 1,25Hz | Uit |
| HSEM-3 | Uit | Knippert met een frequentie van 1,25Hz |
| HSEM-4 | Knippert met een frequentie van 1,25Hz | Knippert met een frequentie van 1,25Hz |
| onbekend | Uit | Uit |

Tijdens de implementatie van de dual-core communicatie is er probleem van data coherentie opgetreden door cache. Als oplossing voor dit probleem is binnen het project gekozen voor een Memory Protection Unit (MPU). Door een MPU kan een deel van het geheugen non-cacheable gemaakt worden wat het probleem verhelpt. Voor uitleg zie bijlage VII. Complicatie dual-core communicatie.

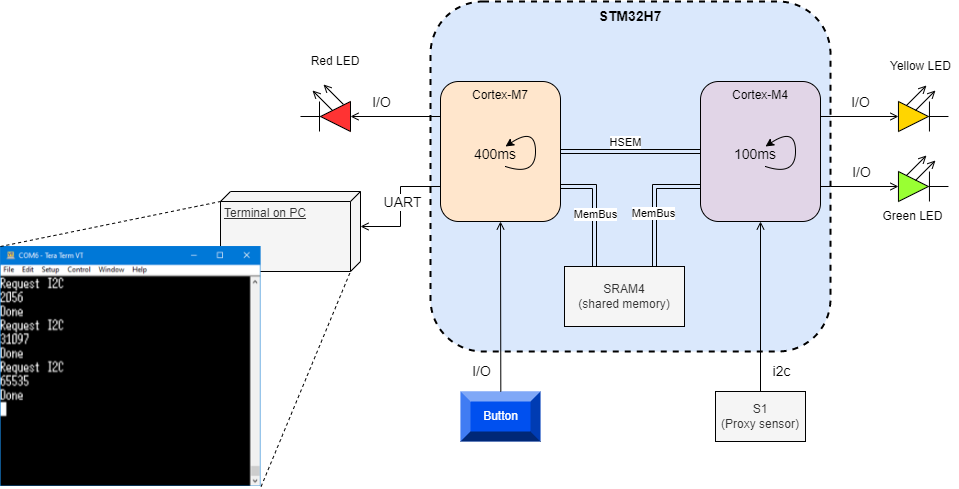
Met het succesvol implementeren van beide tests is aangetoond dat de HSEM en shared memory beantwoorden aan de requiremtents van dual-core communicatie. Hiermee is een onderdeel van het BSP afgerond.

## BSP modules voor hardware

Naast de dual-core communicatie zijn unit tests uitgevoerd voor i2c, UART en PWM. In de volgende paragrafen worden de unit tests samengevoegd tot demo’s. Eerst zijn i2c & UART toegevoegd aan de unit van de dual core communicatie, daarna is de demo verder uitgebreid met PWM. Zo wordt stapsgewijs toegewerkt naar het complete besturingssysteem van de 4-op-1-rij robot.

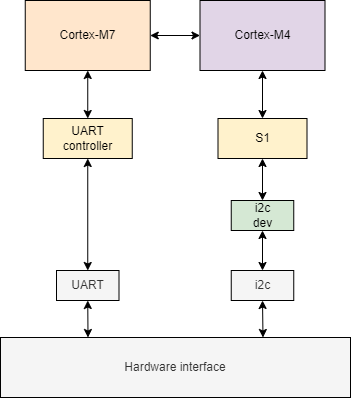
### Dual-core i2c UART demo

De demo is gericht op het aantonen van modulariteit door middel van de implementatie van i2c en UART modules. Figuur 27 geeft een schematisch overzicht van de demo.



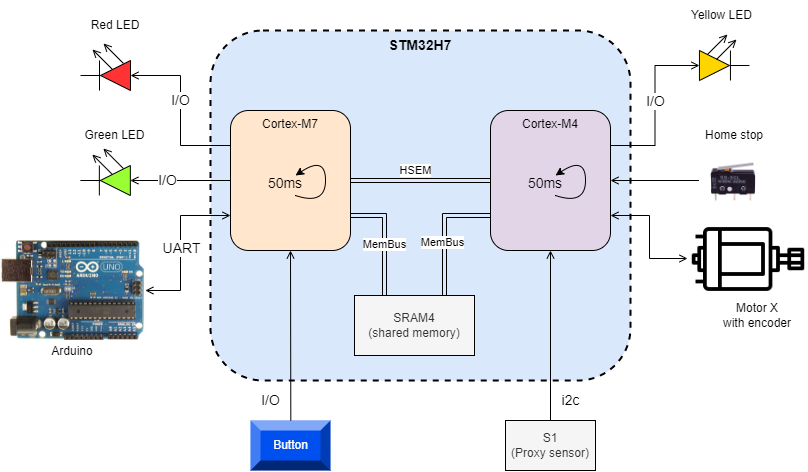
Figuur 27 Schematisch overzicht demo dual-core i2c UART

Voordat de demo is opgesteld zijn eerst de afzonderlijke unit tests uitgevoerd van i2c en UART. De BSP module “i2c dev” en “UART controller” zijn generiek (zie Figuur 28). De module i2c dev zorgt er voor dat elke willekeurige sensor enkel hoeft aan te geven of er data verstuurd of opgevraagd moet worden. De module UART controller zorgt er voor dat data via UART verstuurd en ontvangen kan worden. De i2c unit test is uitgevoerd met een willekeurige i2c en de UART unit test met een connectie met een laptop. Beide unit tests hebben aangetoond dat de BSP modules werken en in een demo (en in het uiteindelijke besturingssysteem) toegepast kunnen worden.

Nadat is aangetoond dat de modules i2c en UART afzonderlijk functioneren is het in de demo de bedoeling om ze samen te voegen. De Cortex-M7 heeft een 400ms delay per loop en laat de rode LED knipperen met een frequentie van 1,25Hz. De Cortex-M4 heeft een 100ms delay per loop en laat de groene LED knipperen met een frequentie van 2,5Hz. Verder wordt bij elk interval van de Cortex-M4 de waarde van de i2c sensor gecheckt via de BSP module. De sensor is een proximity sensor (S1). Als de waarde van de S1 boven een ingestelde grens komt, gaat de gele LED branden. Wanneer de waarde van S1 weer onder de grenswaarde komt gaat de gele LED uit. Als de button op de Cortex-M7 wordt ingedrukt zal er een verzoek, via een HSEM, naar de Cortex-M4 gaan om de data van S1 op te vragen. In de Cortex-M4 wordt deze HSEM ontvangen en wordt de data van S1 uitgelezen. Vervolgens wordt de data in SRAM4 geplaatst en wordt er een HSEM naar de Cortex-M7 gestuurd dat er data beschikbaar is. Als de Cortex-M7 de HSEM ontvangt wordt de data uit SRAM4 gelezen en weer gegeven op de terminal van een computer. Het weergeven op de computer gebeurd via UART BSP module. Figuur 28 geeft de gebruikte software modules weer. De demo heeft aangetoond dat de communicatie tussen de verschillende modules verloopt volgens verwachting.

Figuur 28 Software layers dual-core i2c UART

### Dual-core i2c UART PWM demo

De laatste demo is een uitbreiding op de vorige demo, met meer modules van de 4-op-1-rij robot. Deze test moet zo goed mogelijk de 4-op-1-rij robot representeren zodat naar aanleiding van deze demo een definitieve variant van het besturingssysteem ontwikkeld kan worden. Figuur 29 geeft een schematische overzicht van de demo. 

Figuur 29 Schematisch overzicht demo dual-core i2c UART PWM

Voorafgaand aan de demo is de unit test uitgevoerd voor BSP module PWM, die zorgt voor de aansturing van motoren en servo’s. Deze test is uitgevoerd door eerst één servo aan te sturen. De servo is demontabel en de werking kan eenvoudig aangetoond worden. Daarna is de PWM toegepast op de motor die onderdeel is van de fysieke 4-op-1-rij robot. De servo’s en motoren werken naar verwachting waardoor de BSP module PWM kan worden toegepast in de demo.

Nu is aangetoond dat de module PWM afzonderlijk functioneert, wordt deze ingevoegd in de demo. De Cortex-M7 heeft een 50ms delay per loop. Op de core is de state machine (Figuur 17) van de game controller geïmplementeerd. De Cortex-M4 heeft een 50ms delay per loop en bij elk interval wordt de waarde van de i2c proximity sensor (S1) gecheckt. De sensor S1 dient als representatie voor de user detect van de 4-op-1-rij robot. Eerst bevindt de Cortex-M7 zich in de initialize state, brandt de rode LED en worden alle functies geïnitialiseerd. In deze state zal de Cortex-M4 een “homing sequence” uitvoeren voor de motor X. Omdat de motor niet weet waar het aan te drijven component zich fysiek bevindt is het noodzakelijk om van te voren een home positie te zoeken. Zo kan bijgehouden worden waar het component zich bevindt. De motor draait constant één richting op tot de eindpositie is bereikt (home stop). Hiermee is de homing sequence geslaagd en de kalibratiefase afgerond. Nu kan het systeem door naar de volgende state, de Start game state.

In de start game state wordt de button gebruikt om het spel te starten en knippert de groene LED met een frequentie van 2,5Hz. Het systeem gaat naar de human move state en de speler is aan zet. Door een hand of object voor S1 te houden wordt een “fiche” gedetecteerd. De Cortex-M4 genereert random een waarde tussen 1 en 7 om de kolom van het bord na te bootsen. Vervolgens wordt deze data naar het gedeelde geheugen geschreven (SRAM4). Doordat er een HSEM wordt vrijgegeven wordt de Cortex-M7 op de hoogte gebracht van de input van de speler. De Cortex-M7 zal vervolgens de data ophalen uit SRAM4 en naar de state robot move gaan.

In de state robot move wordt de kolom doorgestuurd naar de Arduino, die de Raspberry Pi representeert. De Arduino stuurt terug in welke kolom de robot zijn fiche moet werpen. Hierop zet de Cortex-M7 deze data in SRAM4 en geeft een HSEM vrij zodat Cortex-M4 de data kan ophalen. Op de Cortex-M4 wordt de motor aangestuurd om naar de positie van de gekozen kolom te gaan op de X-as. Na een delay van een seconde gaat de motor weer naar de home positie. Cortex-M7 krijgt een HSEM dat de robot de zet heeft uitgevoerd en zal weer naar de human state gaan.

Door het uitvoeren van deze demo wordt het besturingssysteem van de 4-op-1-rij robot goed benaderd. Het betreft een benadering, omdat niet alle systeemonderdelen zijn meegenomen. Zo is bij de implementatie alleen de motor van de x-as gebruikt, wordt in plaats van een Raspberry Pi een Arduino gebruikt en is de clean-up fase buiten beschouwing gelaten. Dit hoeft geen belemmering te zijn om modulariteit aan te tonen. Immers, de modules die ontbreken hebben dezelfde principes en protocollen als de gebruikte modules.

## Conclusie

In dit hoofdstuk is beschreven hoe de afzonderlijke modules van de BSP zijn getest en vervolgens zijn samengevoegd in steeds complexere demo’s die uiteindelijk het besturingssysteem van de huidige robot moeten vervangen.

Voor de methode van implementatie is gekozen voor een “Round robin with interrupt. Dit is de meest toepasselijke methode voor het implementeren van software. Het is eenvoudig te begrijpen, aan te passen en het is transparant. Het systeem doorloopt standaard fases waardoor de Round robin with interrupt de toepasselijke methode is. Omdat er prioritaire taken zijn is een interrupt noodzakelijk.

In eerste instantie is gekozen om het fundament van het BSP te testen, namelijk de dual-core communicatie. De communicatie is robuust door het gebruik van de HSEM en makkelijk te onderhouden of aan te passen. Daarna zijn afzonderlijk de BSP modules getest voor i2c, UART en PWM communicatie. De unit tests zijn volgens verwachting verlopen en bewijzen dat de afzonderlijke BSP modules kunnen worden gebruikt voor het besturingssysteem.

Parallel zijn demo’s ontwikkeld waarin stap voor stap, op basis van de geteste en gevalideerde BSP modules, is toegewerkt naar het complete besturingssysteem van de 4-op-1-rij robot. Er zijn opeenvolgend demo’s opgesteld waarin de BSP modules dual-core, i2c en UART zijn samengevoegd en vervolgens een demo waarin de setup is uitgebreid met PWM. De demo’s zijn complementair. De demo’s zijn naar verwachting verlopen, hebben aangetoond dat de software architectuur toepasbaar is en de gemaakte design keuze voor modulariteit de juiste zijn. Verder is aannemelijk gemaakt dat de BSP modules ook gebruikt kunnen worden in andere projecten.

# Validatie

Om te kijken of het project goed uitgevoerd is worden in dit hoofdstuk de requirements gevalideerd. Alle requirements met “Must” moeten helemaal of deels uitgevoerd zijn voordat het project geaccepteerd kan worden. In Tabel 9 worden de requirements herhaald en gecheckt. Groen betekent voltooid, oranje betekent deels voltooid en rood betekent niet voltooid. In een korte beschrijving wordt toegelicht wat nodig was om het resultaat te bereiken.

Tabel 9 Gevalideerde requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Requirement | MoSCoW | 🗹 | Beschrijving |
|  | De architectuur van het besturingssysteem moet gestructureerd en modulair zijn om hard- en software developers die niet bekend zijn met het systeem snel inzicht te kunnen geven in het functioneren van de robot. | Must |  | Doormiddel van het SAD is dit bereikt. |
|  | De software architectuur moet toekomstbestendig zijn zodat software developers effectief en efficiënt beheer en upgrades kunnen uitvoeren. | Must |  | Door design keuzes binnen het SAD is gegarandeerd dat het besturingssysteem in de toekomst een upgrade kan krijgen en betrouwbaar blijft. |
|  | De architectuur van het besturingssysteem dient logisch opgebouwd te zijn aan de hand diagrammen zodat software developers en testers snel inzicht kunnen krijgen in het functioneren van de software. | Must |  | In het SAD is de werking van het systeem gevisualiseerd in diagrammen. |
|  | De samenhang tussen software en hardware dient logisch opgebouwd te zijn aan de hand van diagrammen zodat software developers de uiteindelijke software kunnen implementeren. | Must |  | Het SAD bevat diagrammen die de samenhang tussen software en hardware weergeven. |
|  | Er dient een BSP gemaakt te worden van het besturingssysteem waarmee de noodzakelijke hardware componenten van de robot aangestuurd kunnen worden. | Must |  | De BSP is ontwikkeld en de noodzakelijke hardware componenten van de robot kunnen aangestuurd worden. |
|  | De onderdelen van de BSP die noodzakelijk zijn voor het functioneren van de robot moeten binnen het project onafhankelijk van elkaar getest worden. | Must |  | De demo’s hebben bewezen dat de design keuzes voor de implementatie beantwoorden aan de vraag. |
|  | Binnen het nieuwe besturingssysteem moet de STM32H7 dual-core microcontroller geïntegreerd worden. | Must |  | Met het gebruik van STM32H7 dual-core microcontroller is aan deze eis voldaan. Door de implementatie van de dual-core communicatie kan de dual-core microcontroller informatie uitwisselen tussen beide cores. |
|  | De robot moet een kalibratie tool hebben die bij de initialisatie van het systeem er voor zorgt dat de motoren in de Z- en X-richting “gehomed” worden. | Must |  | In de laatste demo zit een kalibratie tool verwerkt die bij het opstarten de motoren kalibreert (homing sequence). |
|  | Er moet een Pin-out opgesteld worden met een tabel en diagram om er voor te zorgen dat hardware developers in de toekomst een PCB design voor de dual-core processor kunnen uitwerken. | Should |  | Er is een pin-out samengesteld voor het implementeren van een custom PCB in de toekomst. |
|  | Het algoritme dat op de Raspberry Pi draait, moet geïntegreerd worden op de nieuwe STM32H7 dual-core. | Could |  | Door tijdgebrek en gebrek aan kennis van het algoritme is, in overleg met technical supervisor, al vroeg in het project besloten deze requirement buiten beschouwing te laten. |
|  | Een fysieke demo moet aantonen dat de modulaire opbouw van de software architectuur toepasbaar is en functioneert. | Could |  | Er is een demo gemaakt die de werking van alle basisfuncties van de software architectuur laat zien. Het is door tijdgebrek (nog) niet gelukt om de robot met het nieuwe systeem in zijn geheel werkend te krijgen. |

# Conclusie en aanbevelingen

Gedurende vijf maanden is binnen ALTEN gewerkt aan de volgende opdrachtomschrijving:

*“Stel een gestructureerde en modulaire software architectuur op voor het besturingssysteem van de 4-op-1-rij robot, waarmee alle veranderingen en uitbreidingen in de toekomst gefaciliteerd kunnen worden en implementeer deze software architectuur op een STM32H7 dual-core microcontroller.”*

De requirements van de opdracht zijn binnen de beschikbare tijd opgeleverd. De gevraagde gestructureerde en modulaire software architectuur is gerealiseerd door een Software Architectuur Document (SAD) op te stellen aan de hand van de template van arc42. Het SAD beschrijft hoe het systeem werkt, hoe het is opgebouwd met softwaremodules en hoe deze modules met elkaar verbonden zijn en communiceren. De werking van het systeem wordt gevisualiseerd met diagrammen, tabellen en schema’s.

De architectuur kent een logische opbouw. Eerst zijn alle software modules opgesteld die nodig zijn voor de 4-op-1-rij robot. Vervolgens zijn deze onderverdeeld in steeds diepere lagen waarbij ingezoomd is op de onderliggende modules (levels). Nadat alle levels zijn uitgewerkt, is gekeken hoe het systeem zich gedraagt als het operationeel is. Hierbij is de communicatie tussen modules tijdens de operatie beschreven en gevisualiseerd. Als laatste is de samenhang tussen software en hardware weergegeven, waarbij uiteindelijk de modulaire opbouw van het systeem in de praktijk is bewezen. Door de software architectuur volgens de template op te bouwen is het voor iedereen duidelijk hoe het systeem uiteindelijk functioneert.

Aan de hand van demo’s is aangetoond dat de software architectuur daadwerkelijk werkt. Deze zijn stapsgewijs geïmplementeerd en uitgebreid om het uiteindelijke systeem te benaderen. Er is gekozen voor een benadering omdat niet alle modules van betekenis zijn om aan te tonen dat de modulaire softwarearchitectuur werkt. Er zijn mogelijkheden om de robot in de toekomst uit te breiden met features die de 4-op-1-rij robot aantrekkelijker, slimmer en sneller maken.

De stap van een single core naar een STM32H7 dual-core microcontroller is een upgrade die tegelijkertijd is geïmplementeerd met de re-design van de software architectuur. Met de implementatie van de STM32H7 dual-core microcontroller is aangetoond dat de communicatie tussen twee cores goed kan verlopen. Het is de eerste keer dat binnen ALTEN een dual-core microcontroller wordt toegepast. Daarmee is dit project een “prove of concept” voor het gebruik van de dual-core in volgende projecten.

Op basis van de resultaten zijn er enkele aanbevelingen denkbaar. Het project kan een vervolg krijgen door alle overige modules voor de 4-op-1-rij robot te implementeren. De robot werkt nu suboptimaal omdat alleen de noodzakelijke modules zijn ontwikkeld, geïmplementeerd en gevalideerd. Daarnaast kan het algoritme van Raspberry Pi naar de STM32H7 dual-core microcontroller verplaatst worden zodat het systeem op één microcontroller draait. De dual-core processor heeft voldoende ruimte en kracht om dit mogelijk te maken. Bovendien kan onderzocht worden welke softwaremodules toegevoegd moeten worden om de 4-op-1-rij robot uit te breiden met ethernet of een beeldscherm.

# Evaluatie

Tijdens mijn opleiding heb ik ontdekt dat coderen mijn passie is. Ik was dan ook enigszins teleurgesteld dat het coderen tijdens mijn afstuderen op de tweede plaats kwam. Eerst moest een software architectuur worden ontwikkeld, pas daarna kon er geprogrammeerd worden. Achteraf heb ik de volgorde leren waarderen. Het heeft mij veel tijd gekost om te doorgronden wat er precies onder software architectuur wordt verstaan en hoe je zo’n document opzet. Dankzij Aniel (technical supervisor) en Berend (Software architect) is mij duidelijk geworden hoe je software architectuur gebruiksvriendelijk designed en visualiseerd. Het heeft er toe geleid dat ik veel nieuwe dingen heb geleerd over het reduceren van complexiteit en het vertalen van een besturingssysteem in modules en diagrammen. Uiteindelijk heb ik toch nog kunnen programmeren aan de verschillende demo’s.

In de laatste fase van het project heeft een nieuwe consultant (zonder kennis van de 4-op-1-rij) het SAD door genomen. Hij geeft aan dat hij begrijpt hoe het systeem werkt en geïmplementeerd moet worden. Dit geeft mij het gevoel dat de opgestelde software architectuur goed ontworpen en echt toepasbaar is.

Ik heb ook veel geleerd van de medewerkers van ALTEN en mijn collega stagiaires. Ik heb vragen kunnen stellen en altijd goede feedback gekregen. Daarbij werd ik uitgedaagd om buiten de bekende paden te zoeken. Het was voor mij een uitdaging om een afstudeerverslag te vullen. Ik vind het lastig om uitgebreid te rapporteren. Wellicht had ik beter voor de Engelse taal kunnen kiezen omdat veel technische termen in het Engels zijn. Wat betreft mijn communicatieve vaardigheden ben ik uitgedaagd om regelmatig een presentatie te geven aan medewerkers over de voortgang van mijn project. Dat gaat mij prima af.

Als laatste ben ik trots op het feit dat ik na mijn afstuderen bij ALTEN aan de slag mag gaan als consultant bij Mechatronics. Ik kijk daar erg naar uit.

# Bibliografie

|  |  |
| --- | --- |
| [1] | „ESCON 36/3 EC,” Oktober 2012. [Online]. Available: https://docs.rs-online.com/5cf2/0900766b811a32c7.pdf. |
| [2] | „Encoder HEDL 5540,” maart 2021. [Online]. Available: https://www.maxongroup.us/medias/sys\_master/root/8884124516382/EN-21-488-492.pdf. |
| [3] | „Motor EC-i 40,” Mei 2013. [Online]. Available: https://www.maxongroup.com/medias/sys\_master/root/8806895386654/13-217-en.pdf. |
| [4] | „Servo,” Oktober 2011. [Online]. Available: https://nl.mouser.com/datasheet/2/321/900-00005-Standard-Servo-Product-Documentation-v2.-462659.pdf. |
| [5] | „Powersupply,” Oktober 2020. [Online]. Available: https://nl.mouser.com/datasheet/2/260/mwec\_s\_a0011714497\_1-2274579.pdf. |
| [6] | „Vaccuum pomp,” [Online]. Available: https://www.sparkfun.com/datasheets/Robotics/Other/spec%20sheet.jpeg. |
| [7] | „Solenoid,” Maart 2018. [Online]. Available: https://nl.mouser.com/datasheet/2/737/Adafruit\_05132020\_413-1858436.pdf. |
| [8] | „RGB sensor,” Augustus 2012. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf. |
| [9] | „ALTEN Nederland,” [Online]. Available: https://www.alten.nl/. |
| [10] | „arc42,” [Online]. Available: https://arc42.org/why. |
| [11] | „HAL,” [Online]. Available: https://nl.wikipedia.org/wiki/Hardware\_Abstraction\_Layer. |
| [12] | „STM32H755ZI,” [Online]. Available: https://www.st.com/content/st\_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32h7-series/stm32h745-755/stm32h755zi.html. |
| [13] | „ST Microelectronics,” [Online]. Available: https://www.st.com/content/st\_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32h7-series/stm32h745-755/stm32h755zi.html. |
| [14] | „STM32CubeIDE,” [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html#get-software. |
| [15] | „FSM,” [Online]. Available: https://en.wikipedia.org/wiki/Finite-state\_machine. |
| [16] | „Sequence diagram,” [Online]. Available: https://en.wikipedia.org/wiki/Sequence\_diagram. |
| [17] | „ST dual-core communicatie,” [Online]. Available: https://www.st.com/resource/en/application\_note/an5617-stm32h745755-and-stm32h747757-lines-interprocessor-communications-stmicroelectronics.pdf. |
| [18] | „HSEM,” [Online]. Available: https://www.st.com/content/ccc/resource/training/technical/product\_training/group0/2a/6a/df/e1/3b/52/48/b7/STM32H7-System-Hardware\_Semaphore\_HSEM/files/STM32H7-System-Hardware\_Semaphore\_HSEM.pdf/\_jcr\_content/translations/en.STM32H7-System-Hardware\_Semapho. |
| [19] | „Intrerrupt,” [Online]. Available: https://nl.wikipedia.org/wiki/Interrupt. |
| [20] | J. A. Cook en J. S. Freudenberg, „eecs,” 2008. [Online]. Available: https://www.eecs.umich.edu/courses/eecs461/lecture/SWArchitecture.pdf. |
| [21] | „Cache,” [Online]. Available: https://nl.wikipedia.org/wiki/Cache\_(tijdelijk\_geheugen). |
| [22] | „microchip,” [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Managing-Cache-Coherency-on-Cortex-M7-Based-MCUs-DS90003195A.pdf. |
| [23] | „MPU,” [Online]. Available: https://en.wikipedia.org/wiki/Memory\_protection\_unit. |

# Bijlagen

## Originaliteitsverklaring



## Plan van aanpak

Plan van aanpak -

**4-op-1-rij robot**

Versie 5

Datum: 28-2-2022

ALTEN

Pascal Faatz

2491281

Eindversie

## Software Architecturale Document

Software Architectuur Document –

**4-op-1-rij robot**

Versie 3

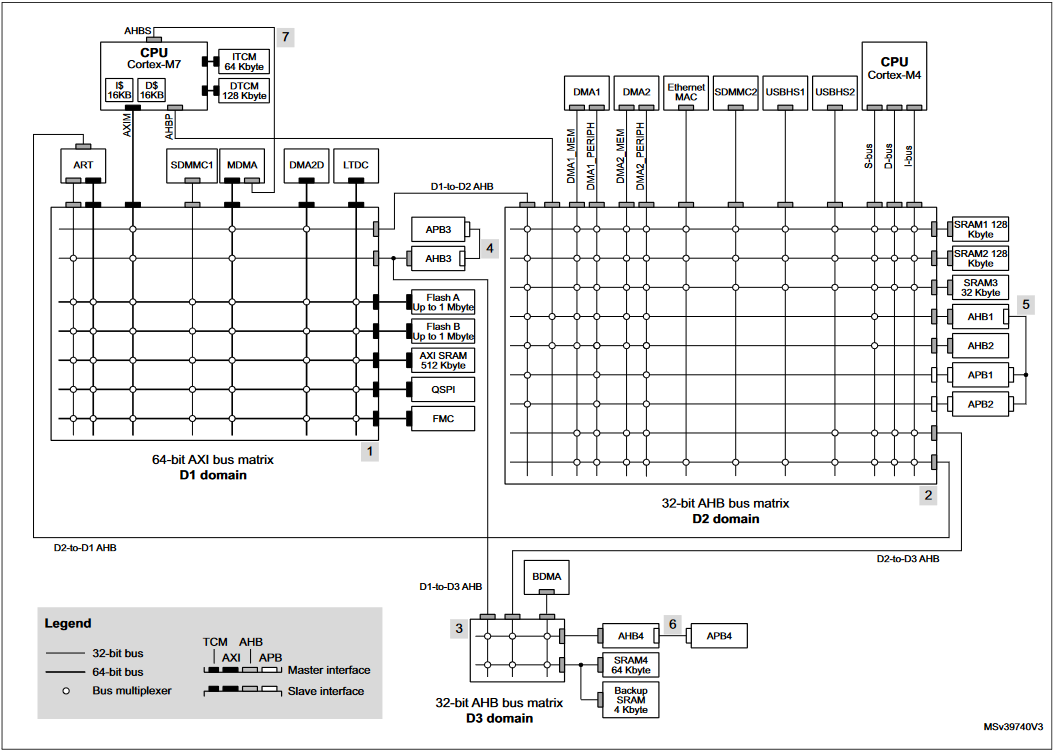
Datum: 3-5-2022

ALTEN

Pascal Faatz

Eindversie

## Memory en bus architectuur STM32H7 dual-core

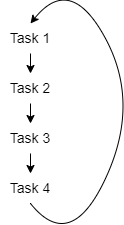


## Implementatie methodes voor een software loop

Het is belangrijk om een goede methode te kiezen om het systeem niet te complex te maken. Vier mogelijke opties zijn onderzocht:

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

**Round robin**

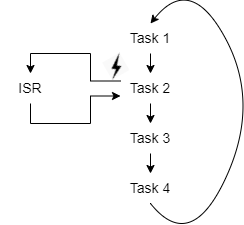
De simpelste implementatie van een software architectuur is de “Round robin”. Round robin voert taken na elkaar uit en als deze allemaal uitgevoerd zijn begint het systeem weer opnieuw (ook wel een loop). Figuur 30 representeert Round robin. Task 1 t/m 4 worden serieel uitgevoerd. Er zijn veel voorbeelden waar deze methode volstaat. Denk aan een snoepautomaat, geldautomaat of een oven. Alle systemen waar de processor genoeg tijd heeft om alle taken te doorlopen en de gebruiker geen vertraging merkt tussen het aanvragen van een taak en het uitvoeren daarvan (denk aan de tijd tussen het drukken van een knop op de oven en het updaten van het scherm) zijn geschikt voor een Round robin implementatie.

In de meeste gevallen volstaat een Round robin methode. Het grote voordeel van de Round robin methode is dat het erg simpel is en goed te onderhouden voor kleine, compacte systemen met niet te veel taken die uitgevoerd moeten worden. De methode kent ook beperkingen. Als een apparaat een taak sneller moet uitvoeren dan dat het systeem door de loop loopt, werkt het systeem niet meer naar behoren. In het slechtste geval is de totale tijd dat het systeem doet over één loop de tijd van alle functies binnen de taken bij elkaar opgeteld. Een Round robin is ook niet robuust. Zodra er een taak wordt toegevoegd aan de loop kan het zijn dat een van de andere taken zijn timing schema niet meer haalt. Hierdoor komt het systeem in de problemen, bijvoorbeeld dat bij een oven het beeldscherm te traag ververst. Dit laatste kan beperkt worden door de tijdgebonden taken vaker terug te laten komen in de loop.

Figuur 30 Round robin

**Round robin with interrupt**

De “Round robin with interrupt” voegt een extra stap toe aan de performance van de Round robin methode. Hier worden belangrijke taken binnen het systeem aangeroepen door een interrupt (vaak een “hardware interrupt”) en uitgevoerd door een Interrupt Service Routine (ISR).



Een hardware interrupt is een onderbreking die gegenereerd wordt door een extern signaal, denk aan een button press, timer of data op een bus. Zodra de onderbreking plaatsvindt wordt er in de core, waar het signaal naar toe gaat, een ISR uitgevoerd. Deze ISR onderbreekt de loop en voert de specifieke code uit die in de ISR staat. Daarna wordt het proces weer opgepakt in de loop [19]. Figuur 31 is een schematische weergave van een ISR.

Figuur 31 Round robin with interrupt

Het grote voordeel van een Round robin with interrupt is dat de interrupt ervoor zorgt dat taken met een hoge prioriteit direct uitgevoerd worden en niet afhankelijk zijn van het doorlopen van de loop. Doordat de methode afgeleid is van de Round robin is de implementatie makkelijk en goed te onderhouden. Door een interrupt in te voegen kan er ook een probleem ontstaan. Als een taak bezig is met een berekening, en die wordt onderbroken door een interrupt die de data van de berekening vernieuwd, kan dat tot verkeerde resultaten leiden (bijlage VI. False data). Hier moet goed rekening mee gehouden worden als deze methode wordt geïmplementeerd.

**Function Queue Scheduling**

De methode “Function Queue Scheduling” maakt net als de Round robin with interrupt gebruik van interrupts. De interrupts worden binnen deze methode voorzien van een prioritair level. Zodra een interrupt een ISR aanroept, wordt deze in een wachtrij gezet. Deze wachtrij wordt door de loop van het systeem uitgevoerd op volgorde van ISR’s met een hoge prioriteit naar ISR’s met een lage prioriteit. Het voordeel van deze methode is dat je prioriteit kan geven aan bepaalde interrupts. Het nadeel is dat deze methode gecompliceerder is dan de hiervoor genoemde methodes. Daarnaast kan ook bij deze methode False data optreden. Ook kan het voorkomen dat een ISR met een lage prioriteit nooit uitgevoerd kan worden omdat er telkens een ISR tussen komt met een hogere prioriteit.

**Real Time Operating System**

Een “Real Time Operating System” RTOS draait op basis van “tasks”. Elke tasks heeft zijn eigen functie en prioriteit. Er wordt geen gebruik gemaakt van een loop waardoor het makkelijk is om tasks toe te voegen en te verwijderen. De RTOS plant op basis van de prioriteit van de tasks welke task eerst uitgevoerd moet worden. Een task kan zich in één van de volgende toestanden bevinden:

1. **Running**

De task wordt uitgevoerd door de processor. Er kan maar één taak tegelijkertijd uitgevoerd worden.

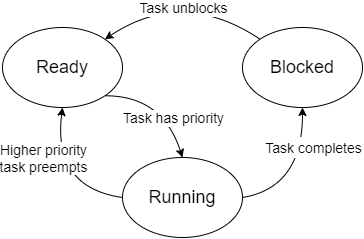
1. **Ready**

Alle data zijn aanwezig om de task uit te voeren wanneer de processor beschikbaar is. Er kunnen meerdere tasks beschikbaar zijn en de processor bepaalt

aan de hand van de prioriteit, in welke volgorde de tasks uitgevoerd worden.

1. **Blocked**

Een task is “blocked” als deze nog niet alle data heeft om uitgevoerd te worden of deze zich in een error state bevindt.



Figuur 32 RTOS scheduler

Het gedeelte dat checkt wat de status is van de tasks heet de “Scheduler”. De instellingen van de Scheduler zijn eenvoudig. Het enige wat deze checkt is de prioriteit van een task en of deze zich in de ready state bevindt. Een task kan zichzelf in de blocked toestand zetten als er geen data is en zichzelf unblocken als de data weer beschikbaar is. Het is dan de taak aan de scheduler om de tasks te verplaatsen, op basis van prioriteit tussen de ready en running state. De werking van de scheduler is weer gegeven in Figuur 32. Het voordeel aan een RTOS is dat de reactie tijd zeer kort is en dat het systeem flexibel is. Het nadeel is dat het een ingewikkelde methode is om toe te passen. Er moet voldoende kennis aanwezig zijn over het systeem, in het bijzonder de timing van de taken. Ook neemt een RTOS veel geheugen in beslag wat in een embedded systeem niet altijd beschikbaar is [20].

## False data

False data kan ontstaan door gebruik te maken van interrupts. Wanneer data gedeeld wordt tussen taken die met een verschillende timing werken moet hier rekening mee gehouden worden. Stel een stuk code dat beschrijft het ophalen van data van een i2c sensor:

**int** i2c\_sensor;

ISR\_read\_i2c(**void**) {

i2c\_sensor = read\_i2c\_Sensor();

}

**int** offset = x;

**int** newValue;

**void** **main**(**void**) {

**while**(**1**) {

...

newValue = i2c\_sensor - offset;

...

}

}

Een interrupt kan de taken die uitgevoerd worden in de loop (while(1)) elk moment onderbreken. In dit geval vraagt de interrupt de data op van een sensor die verbonden is via een i2c bus. Dit gebeurt bijvoorbeeld als er nieuwe data beschikbaar van de sensor. Deze interrupt kan ook plaats vinden op het moment dat de loop bezig is met het berekenen van newValue. In dit geval wordt de data aangepast waarmee de newValue berekent wordt. Hierdoor is niet zeker of de waarde van newValue klopt en ontstaat er false data.

De code die in C geschreven is wordt niet zo uitgevoerd door de microcontroller. De microcontroller voert namelijk binaire machine taal uit. Om het leesbaar te houden kan dit ook in “assembly language” gezet worden. Assembly laguage representeert de niet te begrijpen nullen en enen van de binaire machine taal in leesbare functies die de microcontroller uitvoert. Assembly language geeft dus eigenlijk weer hoe het systeem op register niveau handelt. Een instructie in de assembly language bestaat meestal uit drie onderdelen:

1. **Label**

Het memory adres waar de code zich bevind.

1. **Op-code**

Afkorting voor de instructie die uitgevoerd moet worden.

1. **Operands**

Registers, adressen of data waar de instructie op toegepast wordt.

Hieronder worden een paar voorbeelden gegeven.

add r7, r8, r9; Telt de data in registers 8 en 9 bij elkaar op, plaatst het resultaat in register 7.

and r2, r5, r3; Bitwise AND de data van register 5 en 3,

plaats het resultaat in register 2.

lwz r6, Ox4(r5); Laad de data op het memory adres van de som van register 5

en 0x4 in register 6.

lwzx r9, r5, r8; Laad de data op het memory adres van de som van register 5

en 8 in register 9.

stwx r13, r8, r9; Sla de data in register 13 op in het memory adres van de som van register 8 en 9.

Het belangrijke punt ten opzichte van false data is dat deze assembly handelingen niet onderbroken kunnen worden. Dit is omdat het gaat om fundamentele machine activiteiten. De volgende assembly instructies representeren de regel C code **newValue = i2c\_sensor – offset;** uit het eerste blok:

lwz r1, 0(r12); Zet de data van i2c\_sensor (0(r12)) in register 1.

li r2, x; Zet de waarde van offset (x) in register 2.

sub r3, r1, r2; Haal de data in register 1 en 2 van elkaar af en zet het resultaat in register 3.

stwx r3, 0(r11); Sla de data op in het geheugen (0(r11)).

Hier uit blijkt dat één regel C code uit meerdere regels assembly code bestaat. Dit betekent dat de code niet alleen tussen de regels C code kan worden onderbroken maar ook de regel C code zelf om dat de assembly code van die regel uit meerdere regels bestaat. Dit heeft tot gevolg dat er false data kan ontstaan.

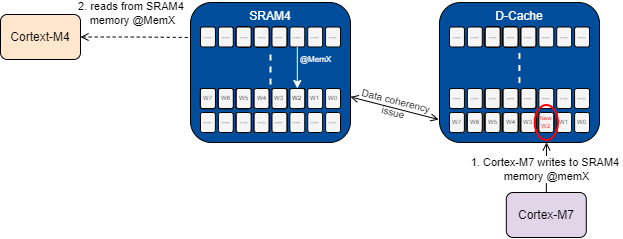
De oplossing voor het voorkomen van false data is simpel. Het deel van de code dat gebruik maakt van data die verworven wordt door een interrupt wordt kritische code genoemd. Het blok kritische code moet beschermt worden voor verhindering. Het uitschakelen van de interrupt voor het kritische blok en het inschakelen ervan na het kritische blok zorgt ervoor dat false data voorkomen wordt. Dit is omdat de data eerst verwerkt wordt voordat er nieuwe data ontvangen kan worden. Een interrupt heeft een hoge prioriteit met een reden dus er moet zeer goed gekeken worden wanneer en waar de interrupts uitgeschakeld worden.

## Complicatie dual-core communicatie

Tijdens het implementeren van de dual-core communicatie is een probleem naar boven gekomen. Bij het testen lukte het niet om communicatie tussen beide cores tot stand te brengen. Na lang debuggen, datasheets doorlezen en informatie op zoeken op het internet werd duidelijk waar het probleem zat. De Cortex-M7 is uitgerust met de mogelijkheid om een gebied in het geheugen te cachen. Ook SRAM4 valt onder dit gebied (Tabel 6). Als de cache is ingeschakeld ontstaat een probleem met de data coherentie.

Cache is geheugen waarin gegevens tijdelijk worden opgeslagen om sneller toegang tot de data te krijgen. Essentieel voor het gebruik van cache is dat het transparant is. Dit houdt in dat bij het ophalen van data niet zichtbaar is of de data uit de originele bron of uit de cache wordt opgehaald. Dit leidt tot het probleem van data coherentie. Door cache in te schakelen is het niet mogelijk om direct naar dit geheugen te schrijven maar wordt de veranderde data eerst in het cache geheugen gezet [21]. Cache is binnen dit project op de Cortex-M7 ingeschakeld om toekomstige upgrades, die snelheid en efficiëntie nodig hebben, te faciliteren.

Figuur 33 Laat schematisch zien hoe het probleem van data coherentie optreedt tijdens de dual-core communicatie. Allereerst heeft de Cortex-M7 bepaalde data verwerkt en moet deze naar de Cortex-M4. De Cortex-M7 schrijft de data naar de van te voren vastgestelde locatie in het gezamenlijk geheugen (SRAM4). In Figuur 33 is dat locatie MemX. Omdat de Cortex-M7 zijn cache heeft ingeschakeld zal de data eerst naar de cache geschreven worden. Zodra de Cortex-M4 een notificatie krijgt dat de Cortex-M7 klaar is met het schrijven van data zal deze de data proberen uit te lezen uit SRAM4. Echter, er is (nog) geen verandering in het geheugenblok van SRAM4 omdat de data van de Cortex-M7 in het cache geheugen is geschreven. Voor het versturen van data van de Cortex-M4 naar de Cortex-M7 is hetzelfde principe van toepassing. De Cortex-M4 schrijft data naar MemX. Zodra de Cortex-M7 een notificatie krijgt dat de Cortex-M4 klaar is met data schrijven probeert deze de data uit te lezen uit SRAM4. Omdat het cache geheugen ingeschakeld is leest de Cortex-M7 uit de cache en niet uit het SRAM4.



Figuur 33 Data cohorentie problem [22]

Voor het probleem van data coherentie zijn twee oplossingen denkbaar:

1. *Gebruik “Cache Clean” en “Cache Invalidate” in code.*

De “Cache Clean” en “Cache invalidate” zijn methodes die voor data coherentie kunnen zorgen. “Cache Clean” wordt gebruikt als de Cortex-M7 naar een locatie in het geheugen wil schrijven. Eerst schrijft de Cortex-M7 naar het geheugen maar wordt de data in de cache gezet. Door “Cache Clean” te gebruiken wordt de data in het geheugen bijgewerkt naar de data die in de cache staat. “Cache Invalidate” wordt gebruikt als de Cortex-M7 data wil uitlezen uit het geheugen. Zodra er nieuwe data in het geheugen staat en de Cortex-M7 deze probeert uit te lezen wordt de data uit de cache gelezen. Als een “Cache Invalidate” gebruikt wordt zal de cache gesynchroniseerd worden met het geheugen. Vanaf dit moment staat de nieuwe data ook in het cache geheugen en heeft de Cortex-M7 toegang tot de data.

1. *Gebruik Memory Protection Unit (MPU) bij de initialisatie.*

Ook een MPU [23] kan gebruikt worden voor data coherentie. Doormiddel van een MPU kan het gedrag van een deel van het geheugen aangepast worden. Er wordt vooraf geconfigureerd of een locatie in het geheugen “cacheable” is. Als cache uitgeschakeld wordt kan de Cortex-M7 direct naar dit deel in het geheugen schrijven en uitlezen.

Binnen dit project is gekozen voor een implementatie van een MPU. Een MPU is eenvoudig vooraf in te stellen. Verder scheelt het veel regels code die anders nodig zijn voor het implementeren van “Cache Clean” en “Cache Invalidate” functies.

## Code Dual-core communicatie task generator

**Common.h beschikbaar voor beide cores.**

/\*

\* common.h

\*

\* Created on: Apr 29, 2022

\* Author: Pascal

\*/

#ifndef INC\_COMMON\_H\_

#define INC\_COMMON\_H\_

#include "stm32h7xx.h"

#define HSEM\_TAKE\_RELEASE(\_id\_) do { HAL\_HSEM\_FastTake((\_id\_)); HAL\_HSEM\_Release((\_id\_), 0); } while (0)

#define HSEM\_WAKEUP\_CPU2 0

#define HSEM\_WAKEUP\_CPU2\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_WAKEUP\_CPU2)

#define HSEM\_CM4\_TO\_CM7 29

#define HSEM\_CM4\_TO\_CM7\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_CM4\_TO\_CM7)

#define HSEM\_CM7\_TO\_CM4 30

#define HSEM\_CM7\_TO\_CM4\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_CM7\_TO\_CM4)

#define HSEM\_ERROR 31

#define HSEM\_ERROR\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_ERROR)

#define HSEM\_CM4\_DONE 1

#define HSEM\_CM4\_DONE\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_CM4\_DONE)

#define HSEM\_ROBOT\_MOVE 2

#define HSEM\_ROBOT\_MOVE\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_ROBOT\_MOVE)

#define HSEM\_HUMAN\_MOVE 3

#define HSEM\_HUMAN\_MOVE\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_HUMAN\_MOVE)

#define HSEM\_CLEAN\_UP 4

#define HSEM\_CLEAN\_UP\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_CLEAN\_UP)

#define HSEM\_CHEAT 5

#define HSEM\_CHEAT\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_CHEAT)

#define HSEM\_COIN\_COLUMN 6

#define HSEM\_COIN\_COLUMN\_MASK \_\_HAL\_HSEM\_SEMID\_TO\_MASK(HSEM\_COIN\_COLUMN)

**static** \_\_attribute\_\_ ((section(".SharedBuffer"), used)) **uint8\_t** SharedBuf[**10**];

#endif /\* INC\_COMMON\_H\_ \*/

**Task generator.c**

/\*

\* task\_Generator.c

\*

\* Created on: May 20 2022

\* Author: Pascal

\*/

#include "task\_Generator.h"

**uint8\_t**\* data;

**void** **initTaskGenerator**(**uint8\_t**\* state, **uint8\_t**\* dataIn){

data = dataIn;

HAL\_HSEM\_ActivateNotification(HSEM\_CM4\_DONE\_MASK);

HAL\_HSEM\_ActivateNotification(HSEM\_COIN\_COLUMN\_MASK);

memset(SharedBuf, **0**, **10**);

}

**void** **taskToDo**(**uint8\_t** task){

**if**(task == TASK\_ROBOT\_MOVE){

memset(SharedBuf, (**int**)(data[**0**]-'0'), **1**);

HSEM\_TAKE\_RELEASE(HSEM\_ROBOT\_MOVE);

}

**if**(task == TASK\_HUMAN\_MOVE){

HSEM\_TAKE\_RELEASE(HSEM\_HUMAN\_MOVE);

}

**if**(task == TASK\_CLEAN\_UP){

HSEM\_TAKE\_RELEASE(HSEM\_CLEAN\_UP);

}

}

**void** **HAL\_HSEM\_FreeCallback**(**uint32\_t** SemMask){

**if**(SemMask == HSEM\_CM4\_DONE\_MASK){

HAL\_HSEM\_ActivateNotification(HSEM\_CM4\_DONE\_MASK);

}

**if**(SemMask == HSEM\_COIN\_COLUMN\_MASK){

HAL\_HSEM\_ActivateNotification(HSEM\_COIN\_COLUMN\_MASK);

}

}

## Code i2c module

**I2c dev.c**

/\*

\* i2c\_dev.c

\*

\* Created on: May 2, 2022

\* Author: Pascal

\*/

#include "i2c\_dev.h"

HAL\_StatusTypeDef **i2c\_CheckDev**(I2C\_HandleTypeDef\* bus, **uint8\_t** DevAddress){

HAL\_StatusTypeDef retFunc;

**uint8\_t** write\_addr = DevAddress << **1**;

retFunc = HAL\_I2C\_IsDeviceReady(bus, write\_addr, **1**, TIME\_OUT);

**return** retFunc;

}

HAL\_StatusTypeDef **i2c\_Transmit**(I2C\_HandleTypeDef\* bus, **uint8\_t** DevAddress, **uint8\_t** MemAddress, **uint8\_t** MemAddSize, **uint8\_t**\* pData, **uint8\_t** pData\_size){

HAL\_StatusTypeDef retFunc;

**uint8\_t** write\_addr = DevAddress << **1**;

retFunc = HAL\_I2C\_Mem\_Write(bus, write\_addr, MemAddress, MemAddSize, pData, pData\_size, TIME\_OUT);

**return** retFunc;

}

HAL\_StatusTypeDef **i2c\_Receive**(I2C\_HandleTypeDef\* bus, **uint8\_t** DevAddress, **uint8\_t** MemAddress, **uint8\_t** MemAddSize, **uint8\_t**\* pData, **uint8\_t** pData\_size){

HAL\_StatusTypeDef retFunc;

**uint8\_t** read\_addr = (DevAddress << **1**) | **0x01**;

retFunc = HAL\_I2C\_Mem\_Read(bus, read\_addr, MemAddress, MemAddSize, pData, pData\_size, TIME\_OUT);

**return** retFunc;

}

**Proxi sensor VCNL4010.c**

/\*

\* VCNL4010.c

\*

\* Created on: May 2, 2022

\* Author: Pascal

\*/

#include "VCNL4010.h"

**void** **VCNL4010\_Init**(**const** VCNL4010\* **const** self){

**uint8\_t** led\_ma = **0x0A**;

**uint8\_t** com\_en = **0x03**;

HAL\_StatusTypeDef retFunc;

retFunc = i2c\_CheckDev(self->bus, self->base\_addr);

**if** (retFunc == HAL\_OK){

i2c\_Transmit(self->bus, self->base\_addr, VCNL4010\_LED\_REG, **1**, &led\_ma, **1**);

i2c\_Transmit(self->bus, self->base\_addr, VCNL4010\_COM\_REG, **1**, &com\_en, **1**);

HAL\_Delay(**1**);

} **else** {

}

}

**uint16\_t** **VCNL4010\_ReceiveProxi**(**const** VCNL4010\* **const** self){

**uint8\_t** buf[**2**];

**uint16\_t** val = **0**;

HAL\_StatusTypeDef retFunc;

retFunc = i2c\_Receive(self->bus, self->base\_addr, VCNL4010\_PROXY\_REG, **1**, buf, **sizeof**(buf));

**if** (retFunc != HAL\_OK){

val = **0**;

} **else** {

val = ((**uint16\_t**)buf[**0**]<<**8**) | buf[**1**];

}

**return** val;

}

VCNL4010 **VCNL4010\_Create**(**uint8\_t** addr, I2C\_HandleTypeDef\* inBus){

VCNL4010 create = { addr, inBus};

**return** create;

}

## Code UART module

**UART\_controller.c**

/\*

\* UART\_controlller.c

\*

\* Created on: Apr 29, 2022

\* Author: Pascal

\*/

#include "UART\_controller.h"

**uint8\_t**\* rxdata;

**void** **Init\_UART\_controller**(UART\_HandleTypeDef\* **const** RPIbus, **uint8\_t**\* data, **uint8\_t**\* substate){

rxdata = data;

HAL\_UART\_Receive\_IT(RPIbus, rxdata, **3**);

srand(**10**);

}

**void** **RPI\_Request\_Move**(UART\_HandleTypeDef\* **const** RPIbus, **uint8\_t** insertColumn){

**int** random = rand() % **7** + **1**;

**uint8\_t** buf[**24**];

sprintf((**char** \*) buf, "UART com. Value : %i**\r\n**", random);

UART\_WriteString(&huart3, (**char** \*)buf);

UART\_WriteValue(RPIbus, random);

}

**void** **HAL\_UART\_RxCpltCallback**(UART\_HandleTypeDef \*huart){

**uint8\_t** buf[**28**];

sprintf((**char**\*) buf, "Received value : %.\*s**\r\n**", **2**, rxdata);

UART\_WriteString(&huart3, (**char** \*)buf);

HAL\_UART\_Receive\_IT(huart, rxdata, **3**);

}

**void** **UART\_WriteString**(UART\_HandleTypeDef\* **const** bus, **char**\* buf){

HAL\_UART\_Transmit(bus, (**uint8\_t**\*)buf, strlen(buf), HAL\_MAX\_DELAY);

}

**void** **UART\_WriteValue**(UART\_HandleTypeDef\* **const** bus, **int** value){

**uint8\_t** buf[**12**];

sprintf((**char**\*) buf, "%i**\r\n**", value);

HAL\_UART\_Transmit(bus, (**uint8\_t**\*)buf, strlen((**const** **char**\*)buf), HAL\_MAX\_DELAY);

}