



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 / 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 +31 (0)6 37 162 867
Business manager	Gijs Haans 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 +31 088 507 81 48



Gerefereerde documenten

Id	Referentie	Titel	Schrijver
D1	Motor driver datasheet [1]	ESCON 36/3 EC Servo Controller	Maxon group
D2	Encoder datasheet [2]	Encoder HEDL 5540	Maxon group
D3	Motor datasheet [3]	Maxon motor EC-i 40	Maxon group
D4	Servo datasheet [4]	Parallax Servo	Parallax
D5	Powersupply datasheet [5]	Mean Well LRS-150	Mean Well
D6	Vacuüm pomp datasheet [6]	SparkFun D2028 pump	SparkFun
D7	Solenoid datasheet [7]	Adafruit 413 solenoid	Adafruit
D8	RBG sensor datasheet [8]	Taos TCS3472	Taos
D9	Lichtsluis design (intern document)	SDD_photodiodeboard.docx	Arjan Verboord
D10	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 hernieuwde 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 Gis 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
SUMMARY	8
AFKORTINGEN	9
1 INLEIDING	10
2 ALTERN	11
3 DE OPDRACHT	12
3.1 PROBLEEMSTELLING	12
3.2 DOELSTELLING	12
3.3 ONTWERPOPPDRACHT	12
3.4 SCOPE EN AFBAKENING	13
3.5 REQUIREMENTS	13
3.6 PROJECT AANPAK	14
4 VOORONDERZOEK	15
4.1 DE 4-OP-1-RIJ ROBOT	15
4.1.1 Spelverloop	15
4.1.2 Hardware identificatie	15
4.2 ARC42	17
4.3 MODULARITEIT	18
4.4 STM32H7 DUAL-CORE	19
4.5 CORE VERDELING	20
4.6 STM32CUBEIDE	20
4.7 CONCLUSIE	21
5 ONTWERP SOFTWARE ARCHITECTUUR	22
5.1 SYSTEEM CONTEXT SAD	22
5.2 BUILDING BLOCK VIEW SAD	23
5.3 RUNTIME VIEW SAD	27
5.4 DEPLOYMENT VIEW SAD	30
5.5 MODULAIRE SOFTWARE MODULES SAD	30
5.6 CONCLUSIE	31
6 IMPLEMENTATIE EN TESTEN	32
6.1 IMPLEMENTATIE METHODE	32
6.2 DUAL-CORE COMMUNICATIE	32
6.2.1 Shared memory	33
6.2.2 Notificaties	33
6.2.3 Implementatie dual-core communicatie	35
6.3 BSP MODULES VOOR HARDWARE	36
6.3.1 Dual-core I2C UART demo	36
6.3.2 Dual-core I2C UART PWM demo	37
6.4 CONCLUSIE	38
7 VALIDATIE	39
8 CONCLUSIE EN AANBEVELINGEN	41
EVALUATIE	42
BIBLIOGRAFIE	43
BIJLAGEN	44
I. ORIGINALITEITSVERKLARING	44
II. PLAN VAN AANPAK	45
III. SOFTWARE ARCHITECTURALE DOCUMENT	59

IV.	MEMORY EN BUS ARCHITECTUUR STM32H7 DUAL-CORE	93
V.	IMPLEMENTATIE METHODES VOOR EEN SOFTWARE LOOP	94
VI.	FALSE DATA.....	96
VII.	COMPLICATIE DUAL-CORE COMMUNICATIE	98
VIII.	CODE DUAL-CORE COMMUNICATIE TASK GENERATOR	100
IX.	CODE I2C MODULE	102
X.	CODE UART MODULE	104

Lijst van figuren

Figuur 1	Kantoren ALTERN Nederland	11
Figuur 2	Organisatie diagram.....	11
Figuur 3	V-model.....	14
Figuur 4	blokdiagram 4-op-1-rij robot.....	15
Figuur 5	Bovenaanzicht 4-op-1-rij.....	16
Figuur 6	Onderaanzicht 4-op-1-rij	16
Figuur 7	arc42 logo	18
Figuur 8	Embedded software layers	18
Figuur 9	Nucleo-H755ZI-Q	19
Figuur 10	STCubeMX software tool gegenereerde layers.....	20
Figuur 11	Project context	22
Figuur 12	Systeem context	23
Figuur 13	BBV STM32H7 level 1	24
Figuur 14	BBV Cortex-M7 level 2	25
Figuur 15	BBV Cortex-M4 level 2	26
Figuur 16	BBV Coin color separator level 3	27
Figuur 17	State machine spelverloop	28
Figuur 18	State machine real-time processing	28
Figuur 19	Sequence diagram robot move.....	29
Figuur 20	Pin out STM32H7 dual-core	30
Figuur 21	Software layers Coin color separato	31
Figuur 22	Dual-core communicatie [17]	33
Figuur 23	EXTI en SEV dual-core communicatie [17]	34
Figuur 24	HSEM dual-core communicatie [17]	34
Figuur 25	dual-core implementatie STM32H7 dual-core	34
Figuur 26	Schematisch overzicht demo dual-core communicatie	35
Figuur 27	Schematisch overzicht demo dual-core i2c UART	36
Figuur 28	Software layers dual-core i2c UART	37
Figuur 29	Schematisch overzicht demo dual-core i2c UART PWM	37
Figuur 30	Round robin	94
Figuur 31	Round robin with interrupt.....	94
Figuur 32	RTOS scheduler	95
Figuur 33	Data cohorentie problem [22]	98

Lijst van tabellen

Tabel 1	Requirements	13
Tabel 2	Componenten 4-op-1-rij	17
Tabel 3	Omschrijving BBV Cortex-M7 level 2	25
Tabel 4	Omschrijving BBV Cortex-M4 level 2	26
Tabel 5	Omschrijving BBV Coin color separator level 3	27
Tabel 6	Geheugen toegang [17]	33
Tabel 7	Mogelijke patronen gedeelde data	35
Tabel 8	Mogelijke patronen HSEM	36
Tabel 9	Gevalideerde requirements	39



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



1 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 onsaamhangende 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.

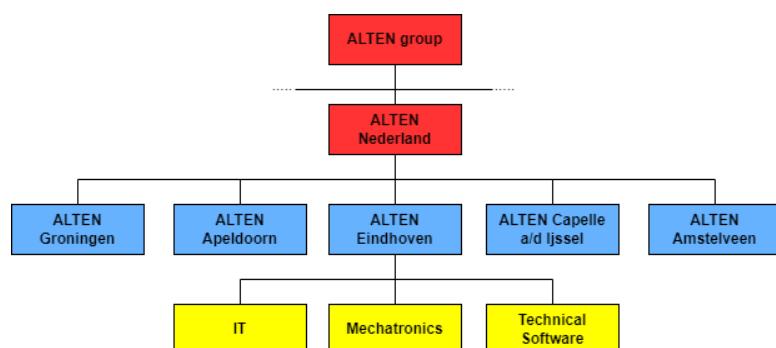
Dit rapport begint met een korte introductie over ALTEN (hoofdstuk 2). Daarna volgt een beschrijving van de opdracht (hoofdstuk 3). Het functioneren van de huidige robot, de requirements van ALTEN en de methodologische keuzes worden toegelicht in hoofdstuk 4. Vervolgens worden in hoofdstuk 5 de belangrijkste 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.

2 ALTERN

ALTERN is een consultancy en engineering organisatie voor hightech en IT sector. Het bedrijf is in 1988 opgericht en heeft het hoofdkantoor in Frankrijk. ALTERN 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 ALTERN Nederland



Figuur 2 Organisatie diagram

Technologie is dé centrale pijler binnen ALTERN. ALten biedt kwaliteit, betrouwbaarheid en innovaties op het gebied van hightech oplossingen. ALTERN 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.

ALTERN 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. ALTERN investeert in individuele ontwikkeling, het verder uitbouwen van expertise en het bieden van een springplank voor de toekomst.

ALTERN 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 (ALTERN) en de stagebegeleider van Fontys Hogeschool Eindhoven.



3 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.

3.1 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.

3.2 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.

3.3 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.”

3.4 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.

3.5 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.

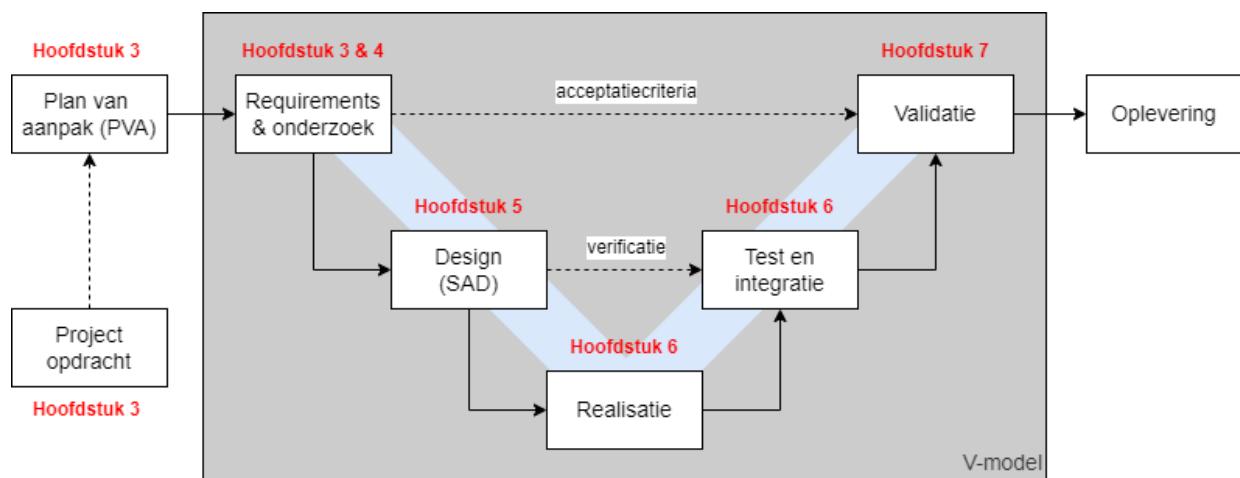
Tabel 1 Requirements

ID	Requirement	MoSCoW
UR.1	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
UR.2	De software architectuur moet toekomstbestendig zijn zodat software developers effectief en efficiënt beheer en upgrades kunnen uitvoeren.	Must
UR.3	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
UR.4	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
UR.5	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
UR.6	De onderdelen van de BSP die noodzakelijk zijn voor het functioneren van de robot moeten binnen het project onafhankelijk van elkaar getest worden.	Must
UR.7	Binnen het nieuwe besturingssysteem moet de STM32H7 dual-core microcontroller geïntegreerd worden.	Must
UR.8	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
UR.9	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
UR.10	Het algoritme dat op de Raspberry Pi draait, moet geïntegreerd worden op de nieuwe STM32H7 dual-core.	Could
UR.11	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.

3.6 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

4 Vooronderzoek

De ontwerpopdracht valt uiteen in enkele onderzoeks vragen 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 modular 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.

4.1 De 4-op-1-rij robot

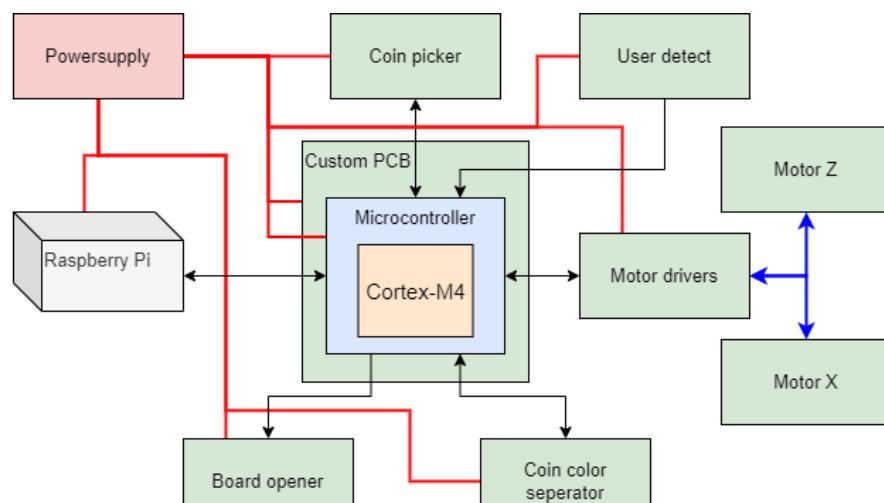
Voor een goed begrip van de huidige robot is in de volgende paragrafen uitgelegd hoe het systeem werkt en uit welke (hardware) onderdelen de robot bestaat.

4.1.1 Spelverloop

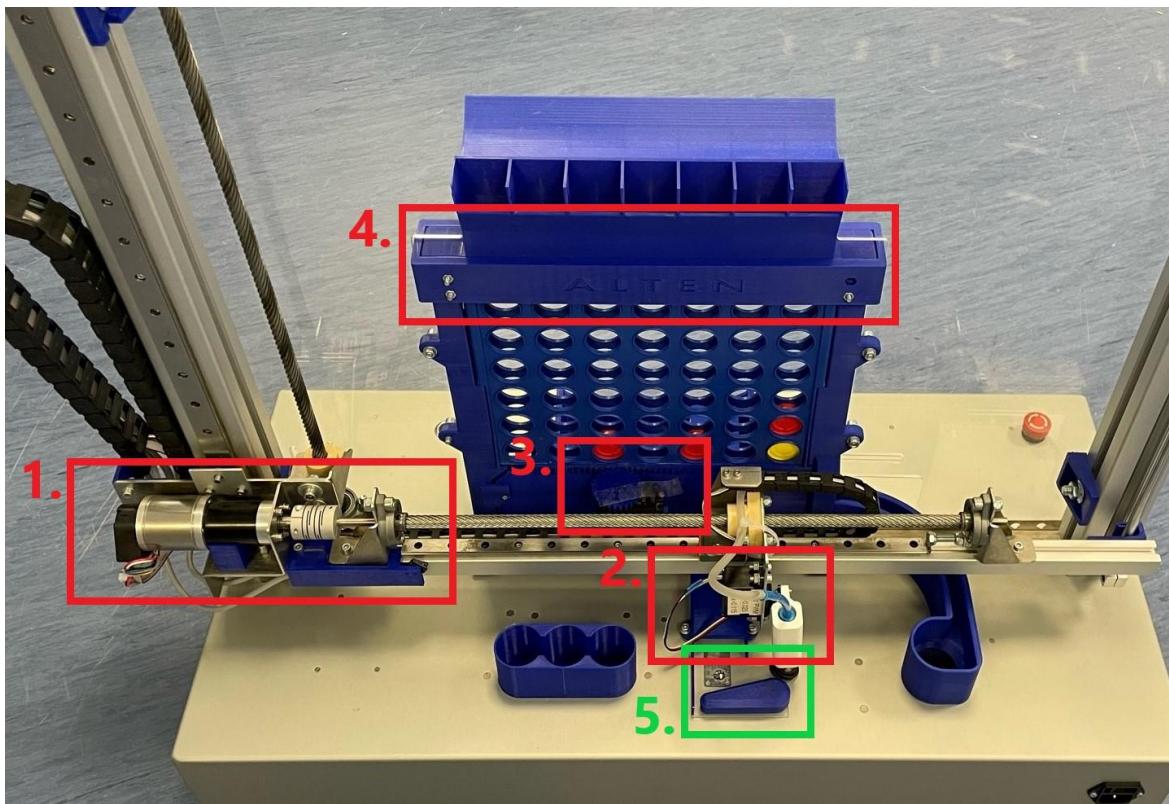
Het spelverloop van 4-op-1-rij staat vast. De speler en de robot doen om de beurt fiches in het spelbord tot een van de twee gewonnen heeft, er een gelijkspel optreedt of er sprake is van vals spel. Het gedrag van de robot kan beschreven worden in drie fasen: een fase waarin de speler aan zet is, een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

4.1.2 Hardware identificatie

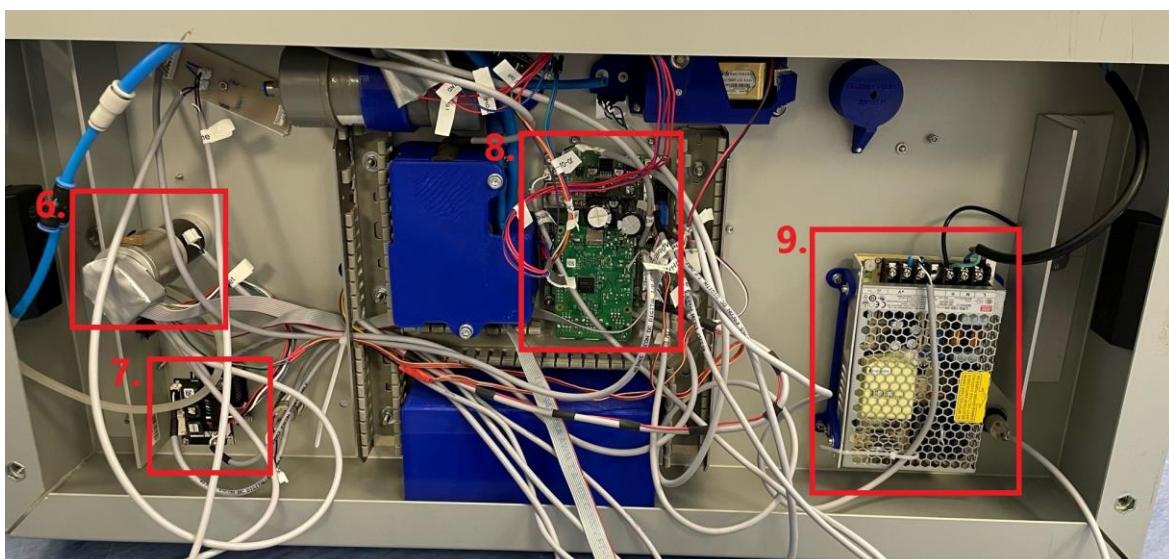
Het mechanische systeem en de elektronische hardware componenten van de 4-op-1-rij robot zijn aanwezig binnen ALLEN. In het blokdiagram (Figuur 4) wordt aangegeven uit welke onderdelen de robot is opgebouwd en hoe deze met het besturingssysteem verbonden zijn. In Figuur 5 is een bovenaanzicht weergegeven 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.

4.2 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.

2. *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.

3. *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.

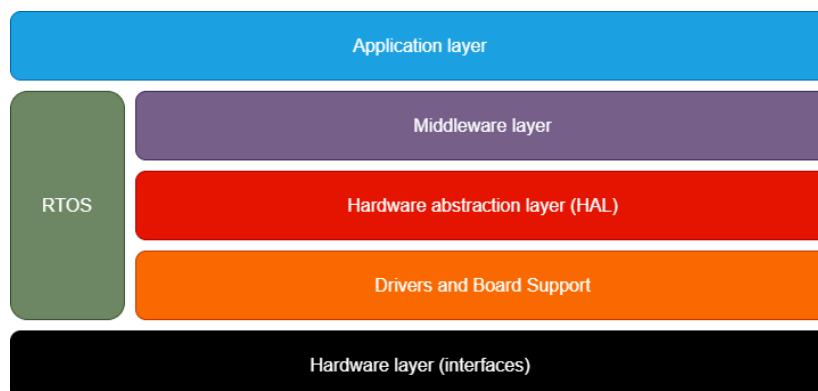
Voor de huidige software architectuur van de robot zijn ad 1 en 2 van toepassing. Na overleg met de software architect binnen ALTERN en de technical supervisor is er voor gekozen om het arc42 template (Figuur 7) te gebruiken voor het opstellen van een software architectuur. Arc42 is een template voor 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 ALTERN. 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

4.3 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.

4.4 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 ALLEN uitgekozen. Het gaat om de STM32H7 microcontroller, specifiek de STM32H755ZIT6U [12] van STMicroelectronics (ST). Deze bevindt 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), vermagenselektronica 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 ALLEN 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 ALLEN een interessante optie en logische stap om het systeem hierop te implementeren om zo de kwaliteit en performance te waarborgen. Verder is het voor ALLEN 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 ALLEN 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.

4.5 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 uitbreidings te faciliteren.

4.6 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 Environment (IDE) gebruikt. Voor de microcontrollers van ST zijn meerdere mogelijkheden voor een IDE.

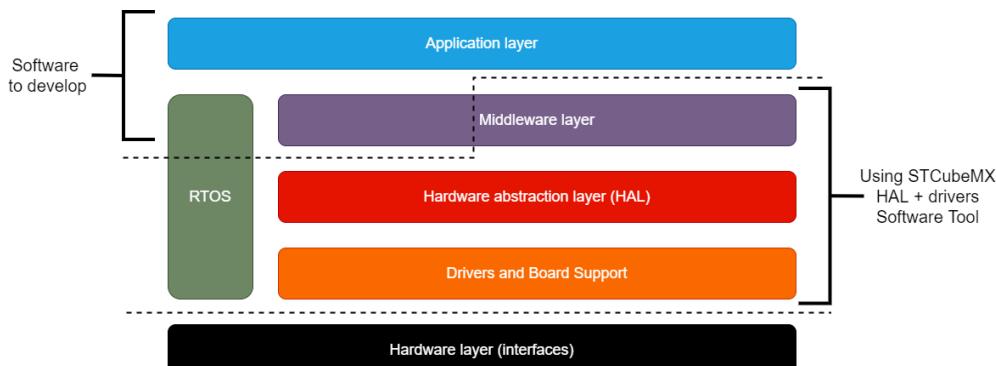
- STM32CubeIDE
- IAR
- Keil

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.

4.7 Conclusie

In dit hoofdstuk zijn onderzoeks vragen 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 ALLEN 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.

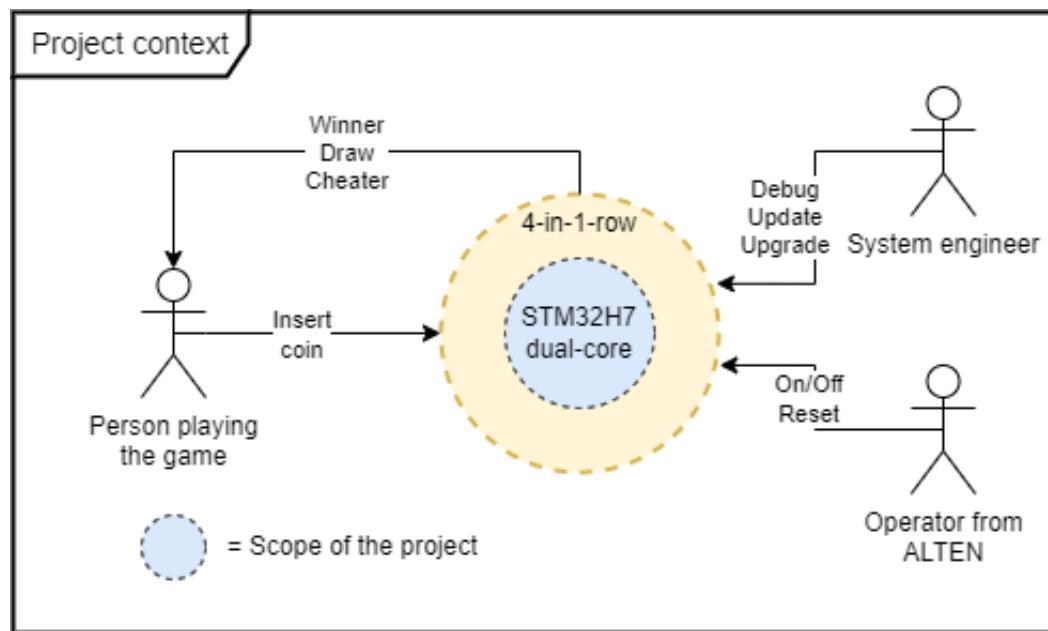
5 Ontwerp software architectuur

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.

5.1 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.

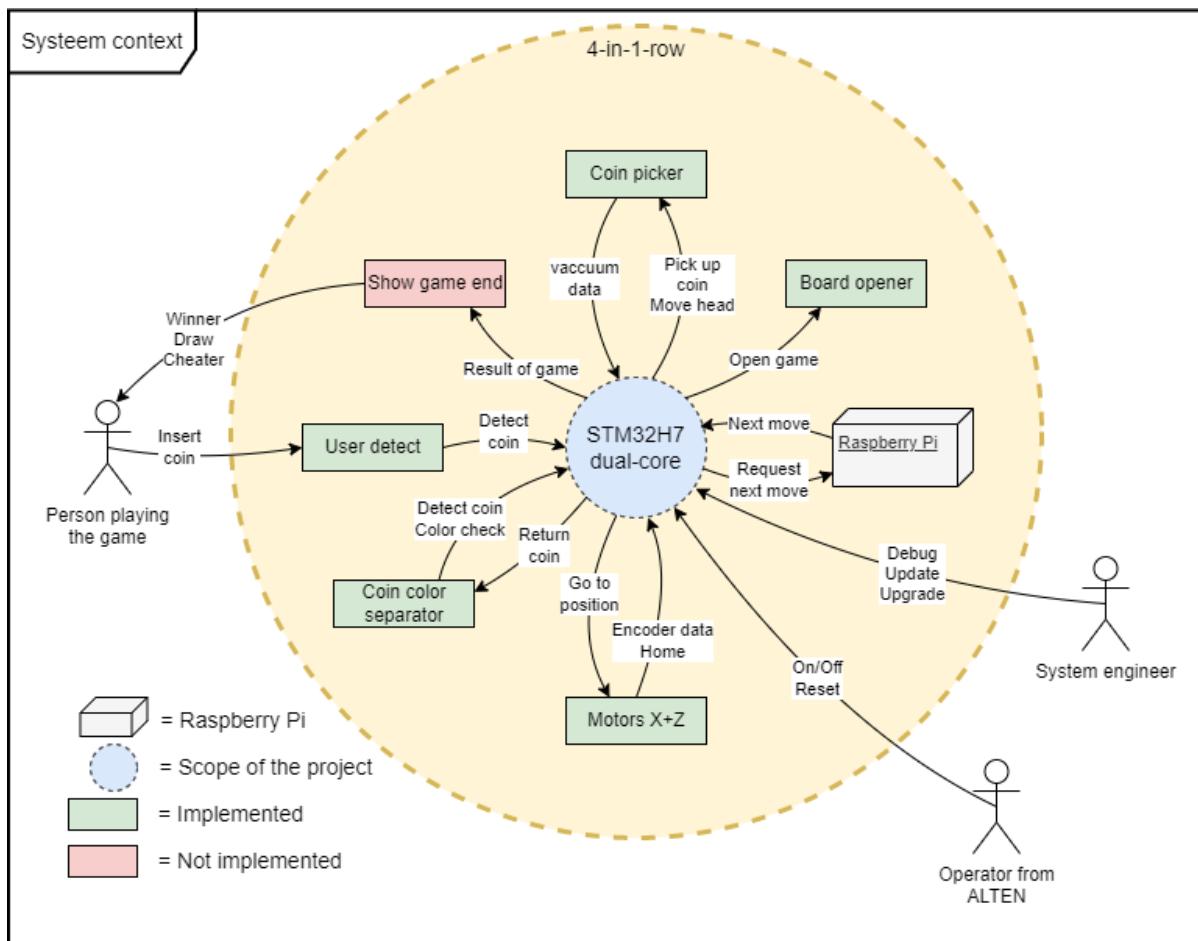
Allereerst is een weergave gemaakt van alle externe hardware interfaces en personen buiten het systeem (zie Figuur 11). Bij de 4-op-1-rij robot is geen externe hardware aanwezig. De personen die de 4-op-1-rij gebruiken zijn "Speler" (Person playing the game), "Systeem engineer" (System engineer) en "Operator". 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.

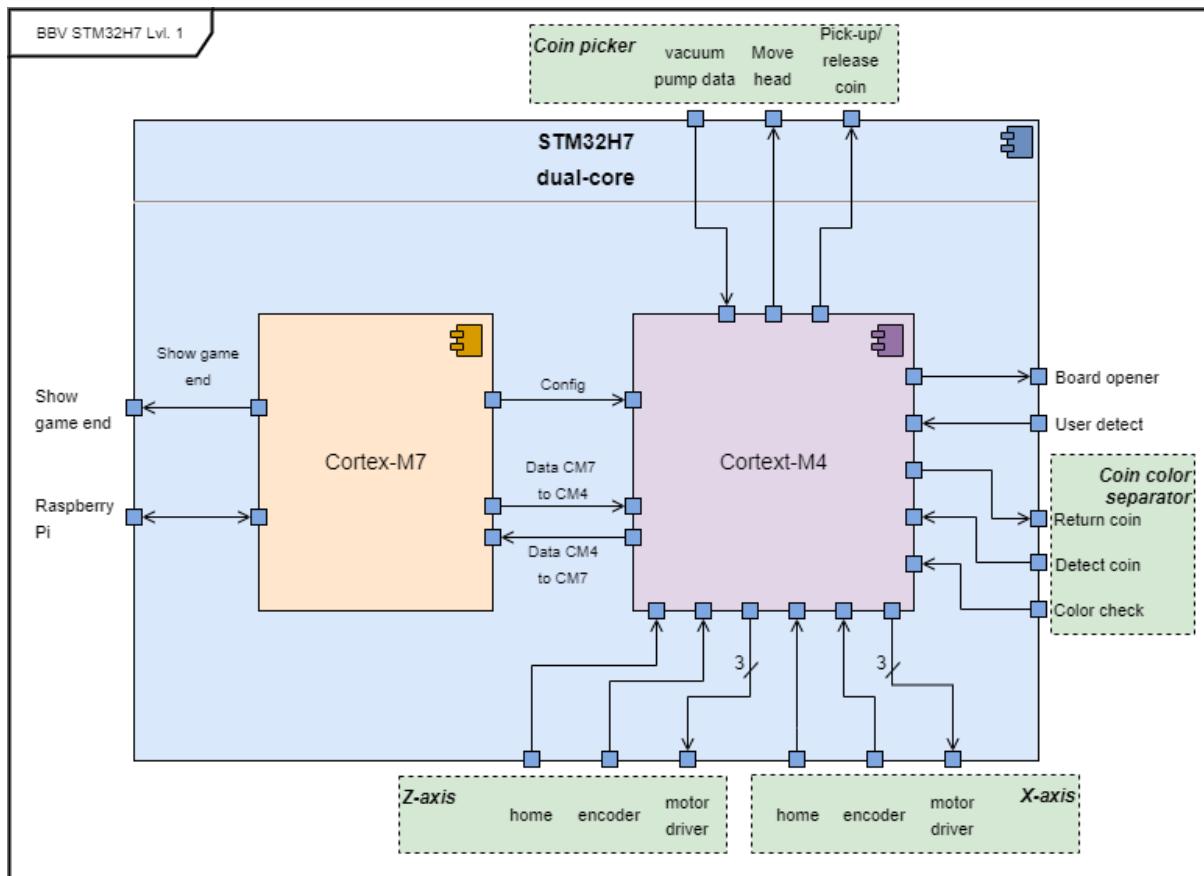


Figuur 12 Systeem context

5.2 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.

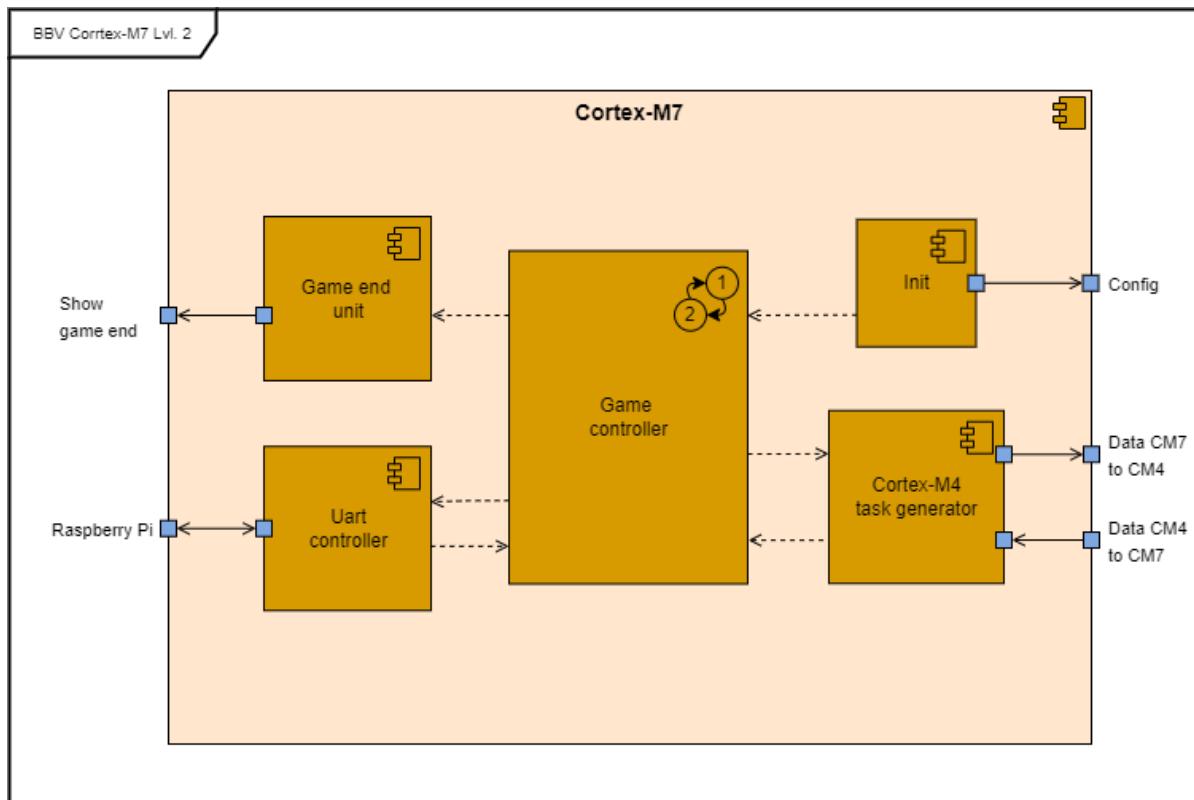
Voor dit project is het eerste level de Systeem context (level 0, Figuur 12). Hierbij is de STM32H7 dual-core een “black box”. In level 1 wordt deze STM32H7 dual-core een “white box” die weer bestaat uit meerdere blokken als “black box”. In level 2 worden de “black boxes” uit level 1 “white boxes”. Tijdens elke stap worden in- en outputs, connecties en opdrachten beschreven van de “black boxes”. Dit is het juiste aggregatienniveau 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 Cortex-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.

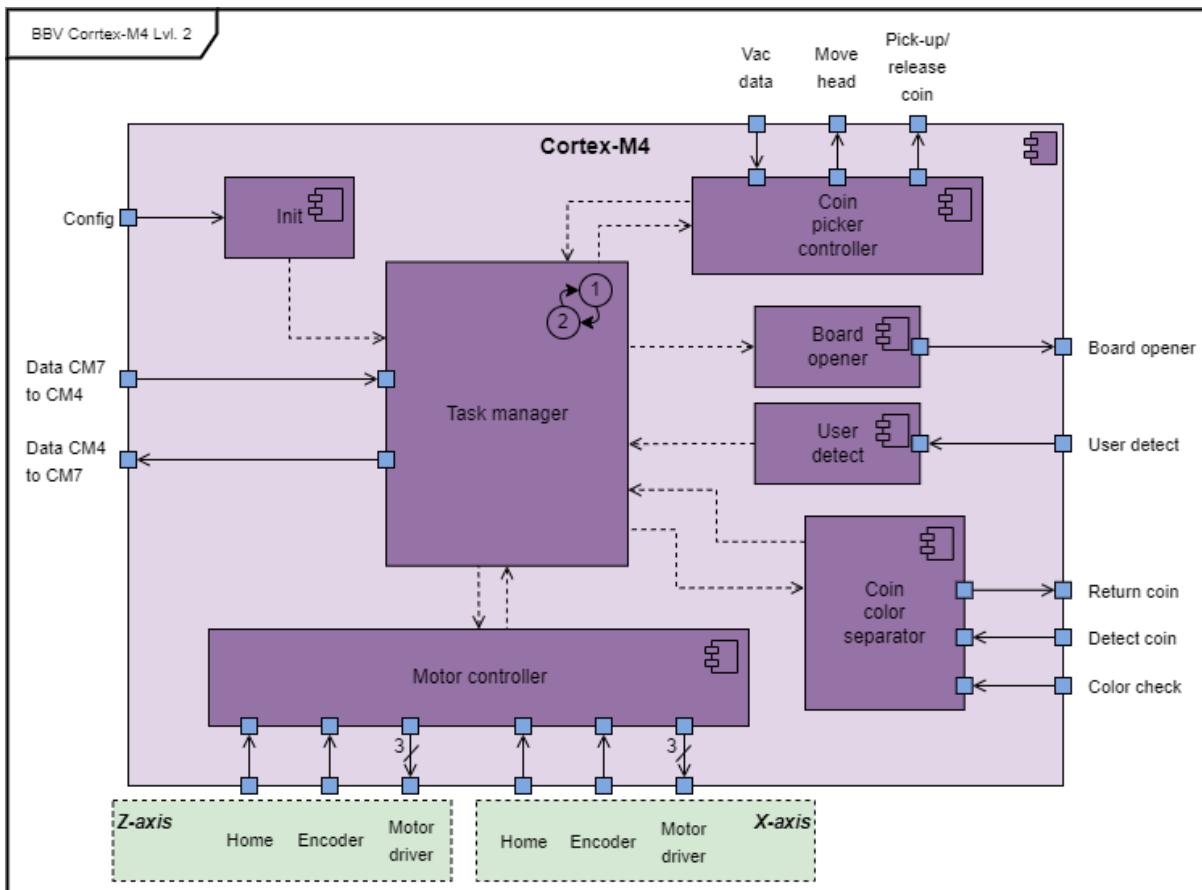
In Figuur 14 en Figuur 15 zijn de Cortex-M7 en Cortex-M4 nog verder in detail ontleed (level 2) en weergegeven als een “white box”. Voor de Cortex-M7 wordt level 2 beschouwd als het laagste level omdat de blokken binnen dit niveau voldoende representatief zijn ontleed voor de uiteindelijke software modules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

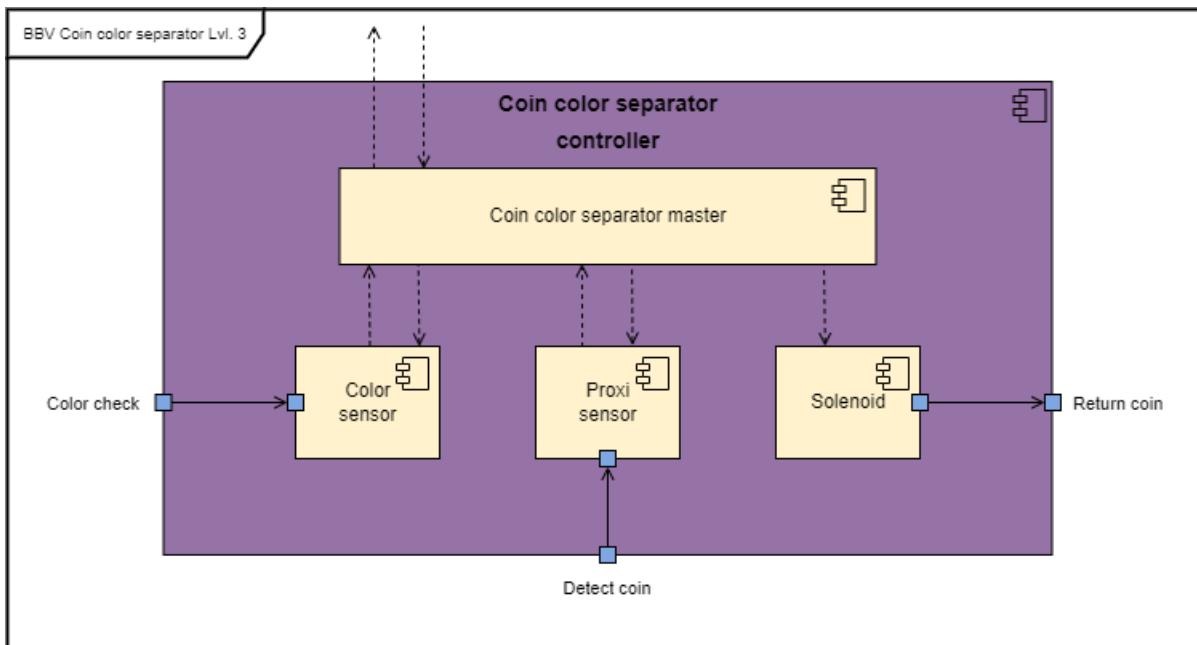


Figuur 15 BBV Cortex-M4 level 2

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 openen 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 separator level 3

Tabel 5 Omschrijving BBV Coin color separator 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.

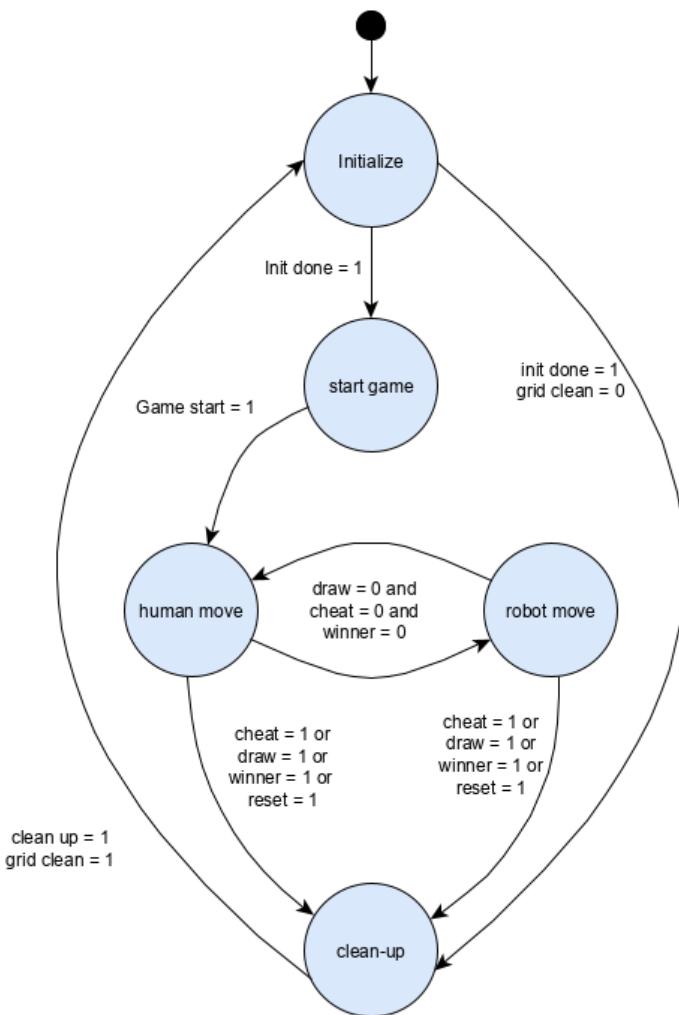
5.3 Runtime view SAD

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 uitsluiten.

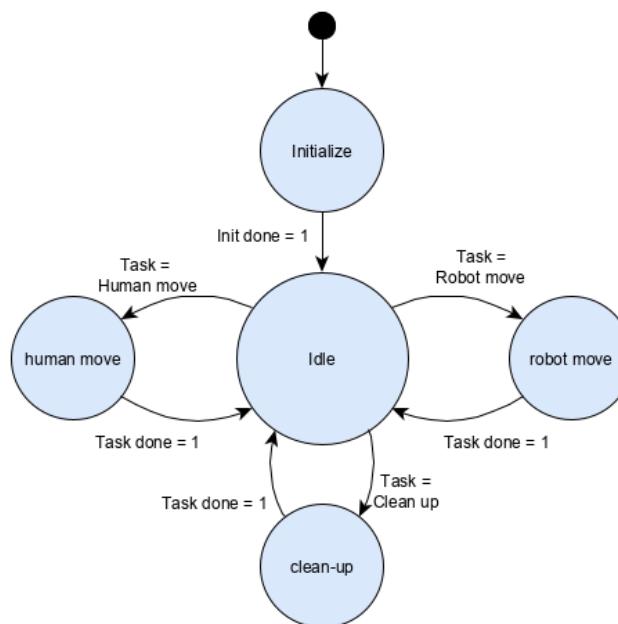
Een state machine is een abstract model voor het gedrag van het systeem. Het model bestaat uit een eindig aantal toestanden (states) waar het systeem zich in kan bevinden en elke toestand heeft een of meerdere overgangen naar volgende states. Deze overgangen worden bepaald door de input die het systeem 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 bevindt 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.

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 binnentkomt 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.

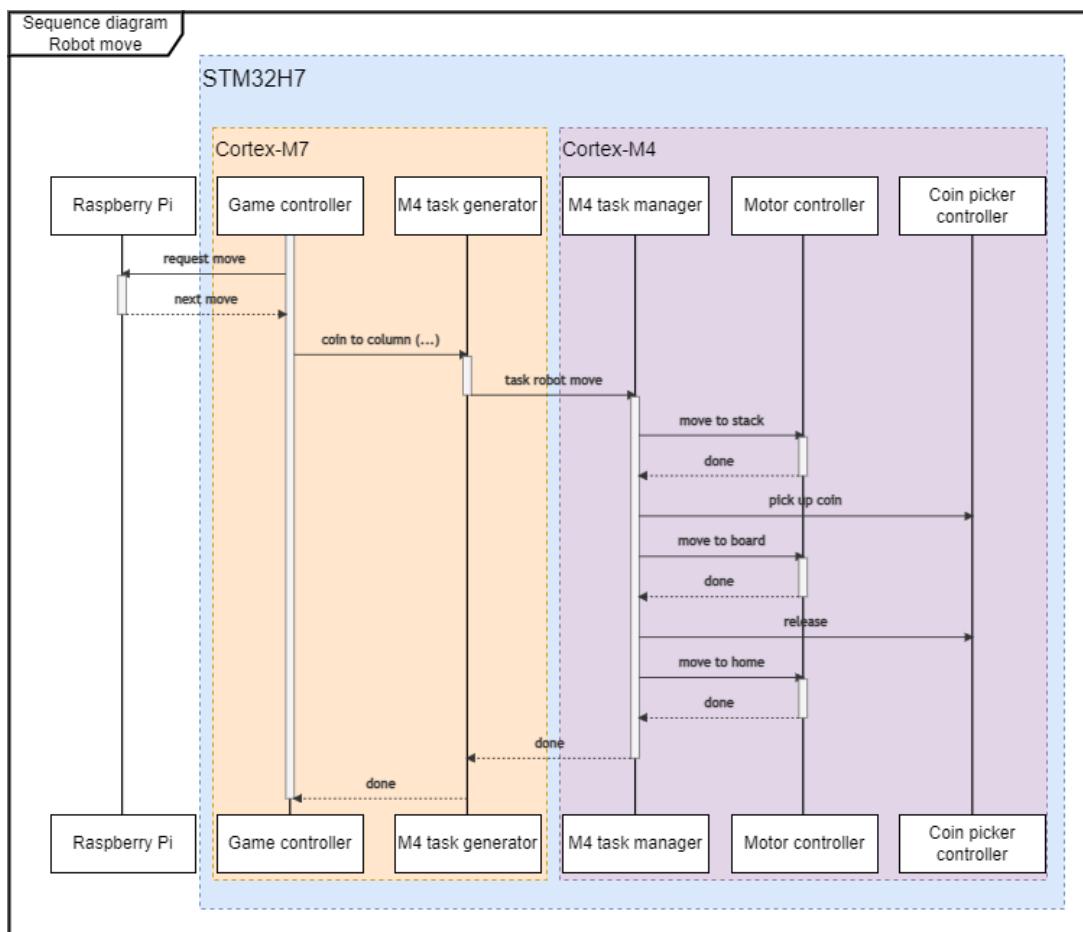


Figuur 17 State machine spelverloop



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.

5.4 Deployment view SAD

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).



Figuur 20 Pin out STM32H7 dual-core

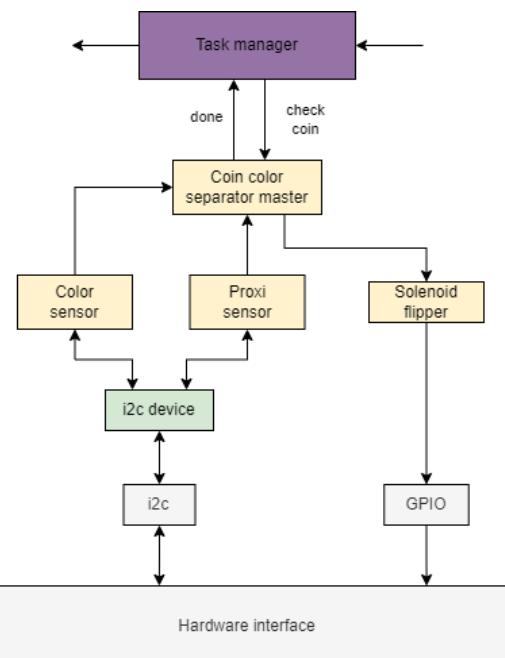
5.5 Modulaire software modules SAD

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 modular 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 separator

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 separator

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 elkaar bestaan/status af te weten. Die informatie wordt bijgehouden door de Task Manager. Ook de Task manager zal werken doormiddel van een state machine.

5.6 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.

6 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 toegewerkten naar het complete besturingssysteem van de 4-op-1-rij robot.

6.1 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)

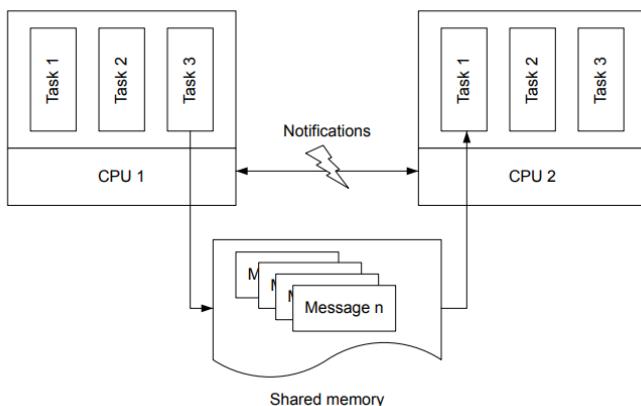
Het uitgangspunt voor een optimaal design is de keuze voor de meest eenvoudige implementatiemethode. De methode moet voldoen aan de performance eisen van het systeem.

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.

6.2 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]

6.2.1 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				

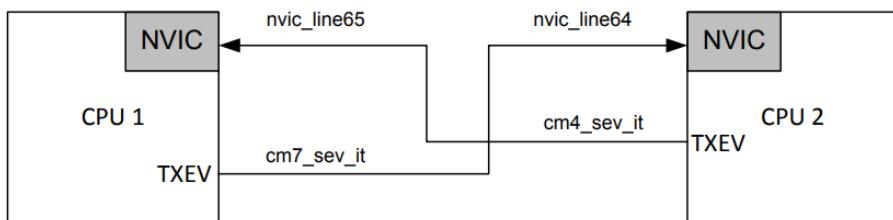
Binnen dit project is SRAM4 gekozen als shared memory. SRAM4 bevindt zich in het D3 domein dat voor beide cores beschikbaar blijft, ook als één van de cores zich in “low-power” modus bevindt of zelfs uit staat. Het D1 respectievelijk D2 domein is alleen beschikbaar als de bijbehorende core actief is. 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.

6.2.2 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:

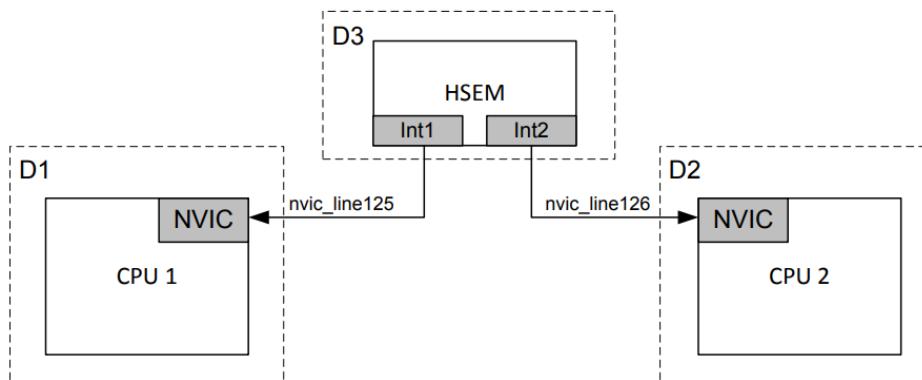
1. *EXTI software interrupt and event registers, CPU send-event instruction (SEV).*
EXTI en SEV zijn eenvoudige interrupts. Figuur 23 laat zien dat beide cores een directe verbinding met elkaar hebben. De lijnen zijn verbonden met de Nested Vector Interrupt Controller (NVIC). De NVIC voert een ISR uit als een interrupt gedetecteerd wordt.



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

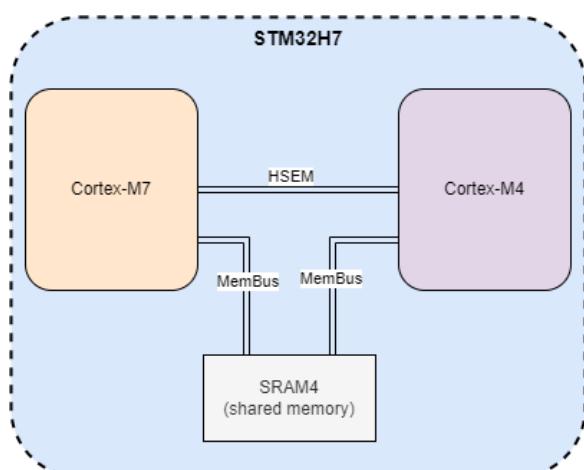
2. Hardware Semaphore (HSEM) free interrupt.

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

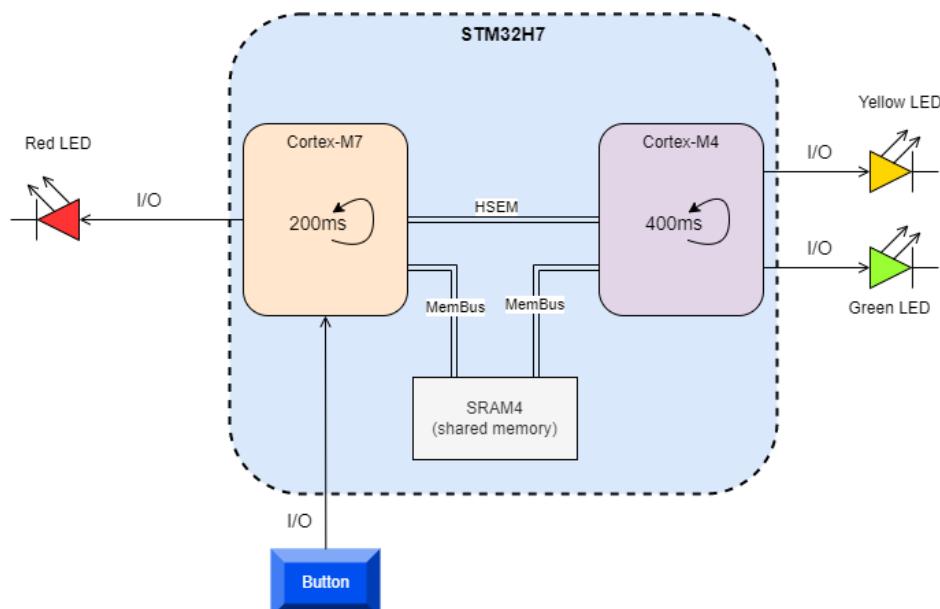
6.2.3 Implementatie dual-core communicatie

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) verhoogt 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.

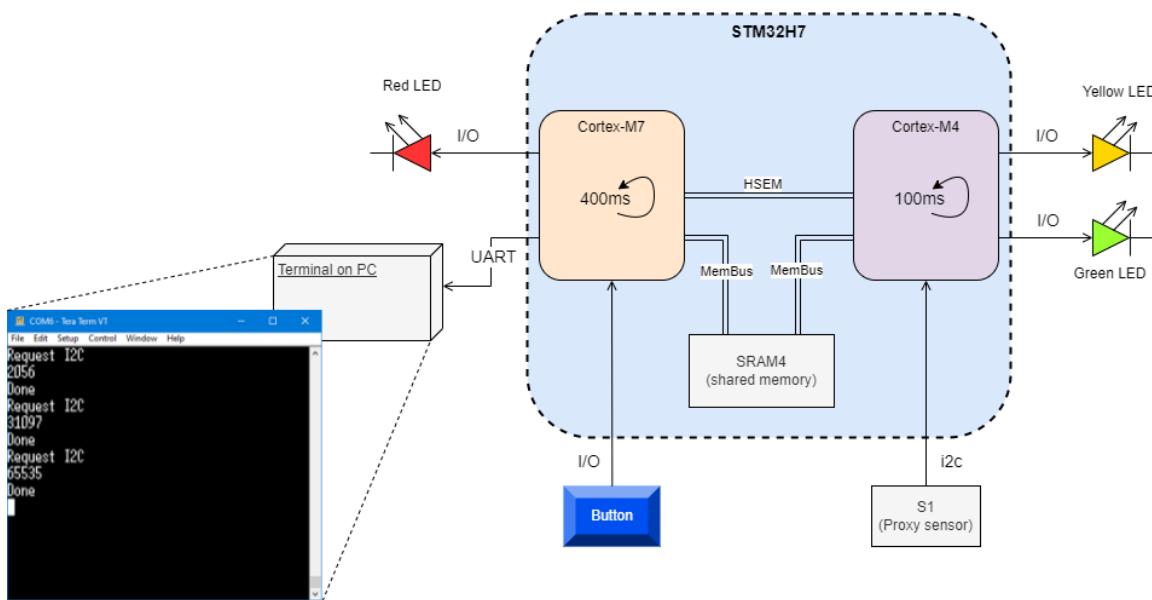
Met het succesvol implementeren van beide tests is aangetoond dat de HSEM en shared memory beantwoorden aan de requirements van dual-core communicatie. Hiermee is een onderdeel van het BSP afgerond.

6.3 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.

6.3.1 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.



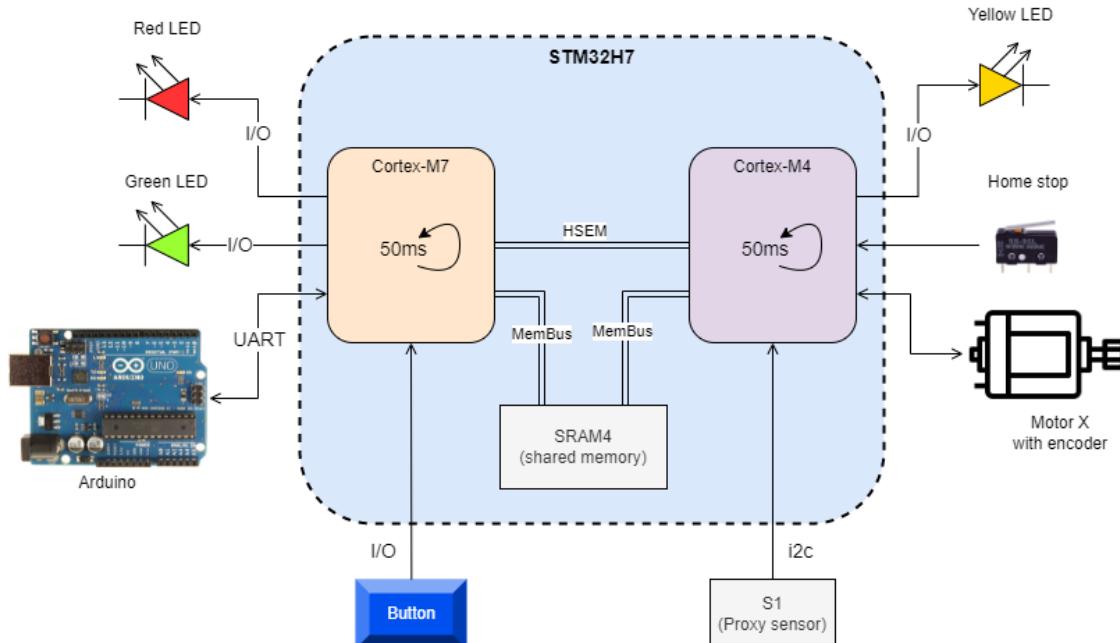
Figuur 27 Schematisch overzicht demo dual-core i2c UART

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.

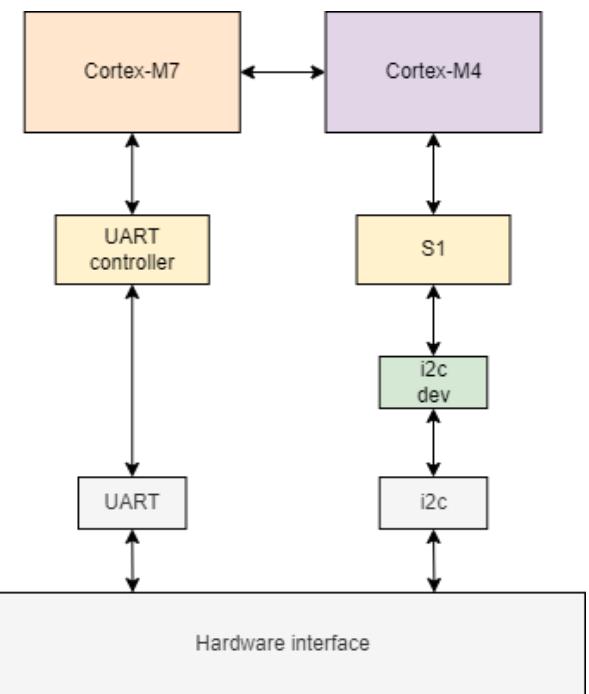
6.3.2 Dual-core i2c UART PWM demo

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.



Figuur 28 Software layers dual-core i2c UART

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 representeren. 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.

6.4 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.

7 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	<input checked="" type="checkbox"/>	Beschrijving
UR.1	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.
UR.2	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.
UR.3	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.
UR.4	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.
UR.5	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.
UR.6	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.
UR.7	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.
UR.8	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).
UR.9	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.



UR.10	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.
UR.11	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.



8 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] „Vacuum 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_\(tijdelijk_geheugen\)](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

I. Originaliteitsverklaring

Versie: 6-6-2022 15:45



ORIGINALITEITSVERKLARING

bij het afstudeerrapport met de titel :

4-op-1-rij robot -
Het ontwerpen van een gestructureerde en modulaire software architectuur

Hierbij verklaar ik dat het ingeleverde rapport zoals hierboven is genoemd, origineel * is: het is door mij, de ondergetekende, persoonlijk opgesteld en opgemaakt.
Om dit stuk te kunnen opstellen heb ik zelf de benodigde onderzoeken uitgevoerd.
Daar waar ik gebruik heb gemaakt van andermans werk, heb ik dat aangegeven bij het betreffende stuk tekst ** en in de literatuurlijst.

Datum : 7-6-2022

Naam : Pascal Faatz

Handtekening student:

- * De Hogeschool heeft de beschikking over controlesoftware m.b.t. originaliteit. Zij behoudt zich het recht om deze software in voorkomende gevallen in te zetten
- ** Letterlijk overgenomen werk dient meteen vóór die tekst begint, te zijn voorzien van de bronvermelding: de titel van het werk waaruit geciteerd wordt alsmede naam van de auteur.
- *** Verplicht opnemen in het verslag

Het betreft hier:

Hoofdstuk	Geschreven door
1 t/m 8	Pascal Faatz



II. Plan van aanpak

Plan van aanpak - **4-op-1-rij robot**

Versie 5
Datum: 28-2-2022

ALTEN
Pascal Faatz
2491281

Eindversie



Versie geschiedenis

Versie	Datum	Status	Schrijver	Opmerking
1	9-2-2022	Concept	Pascal Faatz	Eerste concept versie
2	10-2-2022	Concept	Pascal Faatz	Kleine aanpassingen aan de opbouw
3	14-2-2022	Concept	Pascal Faatz	Feedback Gijs verwerken
4	28-2-2022	Concept	Pascal Faatz	Feedback Jeedella verwerken
5	28-2-2022	Concept	Pascal Faatz	Feedback Aniel verwerken

Acroniemen en afkortingen

Term	Beschrijving
BSP	Board Support Package
PvA	Plan van Aanpak



Inhoud

1 Achtergronden	4
2 Projectresultaten	5
3 Projectactiviteiten	6
4 Projectgrenzen en randvoorwaarden	8
5 Tussenresultaten	10
6 Planning	10
7 Risico's	11
Appendix A. Risico analyse	12
Appendix B. Planning	14
Figuur 1 4-op-1-rij robot	4
Figuur 2 STM32H7.....	5
Figuur 3 V-model binnen ALTEN	6
Figuur 4 Project organisatie	9
Tabel 1 Rolverdeling.....	9
Tabel 2 Deadlines.....	10

1 Achtergronden

Voor mijn afstudeerstageperiode in het vierde jaar van mijn studie Elektrotechniek aan de Fontys Hogeschool Eindhoven ontwikkel ik een embedded architectuur voor de 4-op-1-rij robot van ALLEN op basis van een dual-core STM32H7 processor.

ALLEN is als consultancy en engineering organisatie werkzaam in uiteenlopende markten van de hightech sector en de IT. Kennis op het gebied van technologie speelt hierbij een belangrijkere rol, dé centrale pijler binnen ALLEN. ALLEN zet haar specialistische kennis ook in op het gebied van IT, een belangrijke sector in Nederland. Kwaliteit en betrouwbaarheid, innovaties op het gebied van Big Data en Internet of Things, het zijn onderwerpen waarin ALLEN een actieve rol speelt met toonaangevende partners om zo de digitale economie van Nederland te versterken.

ALLEN heeft drie afdelingen: ALLEN IT, Technical Software en Mechatronics. De 4-op-1-rij robot valt binnen de afdeling Mechatronics. Op de Mechatronics afdeling werken WTB, Mechatronica en elektrotechniek consultant engineers. Mijn technische bedrijfsmentor is Aniel Shri, hij werkt zelf als consultant bij ASML en kan mij hierdoor zeer goed begeleiden binnen de opdracht en ALLEN. Verder heb ik ook een business manager Gijs Haans, hij ondersteunt mij op het gebied van persoonlijke ontwikkeling en voortgang.

Om consultant de mogelijkheid te geven om hun competenties van verschillende Mechatronica onderwerpen te ontwikkelen, heeft ALLEN interne demo projecten. Vaak wordt er een stagiair of afstudeerde aan deze projecten gekoppeld die dan de ideeën en voorwerk van de consultant verder uitwerkt. Voor mij is dat de 4-op-1-rij robot.

ALLEN heeft in eigen beheer een robot ontwikkeld die middels een algoritme 4-op-1-rij kan spelen tegen een menselijke tegenstander (Figuur 1). Met behulp van industriële componenten wordt de 4-op-1-rij robot gebouwd om de kennis van verschillende systemen te demonstreren. De robot zal onder andere op beurzen en open dagen gebruikt worden als demonstratie unit, waarbij deze beschikbaar is voor voorbijgangers om een ronde te spelen. Aan de voorzijde kan de speler dan zijn/haar zet plaatsen op het bord, waarna de 4-op-1-rij robot een zet zal bedenken en uitvoeren. Hiervoor houdt het systeem bij welke zetten door zijn tegenstander gespeeld zijn. Wanneer de zet door de robot bepaald is zal de combinatie van het X-Z platform met roterende vacuüm gripper een steen op pakken en op de gekozen plek in het spel invoeren. Zodra het spel afgelopen of gereset is, zal de 4-op-1-rij robot alle fiches verzamelen en op kleur sorteren om zo klaar te zijn voor de volgende ronde.



Figuur 1 4-op-1-rij robot

2 Projectresultaten

Doelstelling

Momenteel draait de 4-op-1-rij robot op een Raspberry Pi + STM32 controller. Op de Raspberry Pi draait het algoritme om de volgende zet van het systeem te bepalen en op de STM32 controller draait het besturingssysteem. Dit project is binnen ALTEN in eerste instantie gebruikt om consultants die op een tussen periode niet bij een bedrijf zaten toch een uitdagend project te geven. Hierdoor is er, over een langere periode, door meerdere consultants aan gewerkt. Dit heeft geresulteerd in een zeer onoverzichtelijke en onduidelijke architectuur van de software en hardware. Dit geldt dus ook voor het besturingssysteem van de 4-op-1-rij robot.

Door deze onduidelijke architectuur is het zeer lastig om het besturingssysteem uit te breiden of een upgrade toe te passen. Omdat de 4-op-1-rij robot ook op beurzen staat en aantoont wat ALTEN in huis heeft is het van noodzaak dat de componenten en structuur met de markteisen meegroeien.

Een complete re-design van het besturingssysteem is nodig om de onoverzichtelijke en onduidelijke architectuur op te lossen en upgrades mogelijk te maken. Hiervoor moeten keuzes overwogen worden en een Software Architectuur Document (SAD) gemaakt worden. Communicatie met verschillende componenten en randapparatuur is ook cruciaal, hiervoor dienen Board Support Packages (BSP) opgezet te worden aan de hand van opgestelde diagrammen. Daarna moet alles worden omgezet in een functionele embedded low level controller op basis van een STM32H7 dual-core processor.

Probleemstelling

Hoe stel ik een gestructureerde en modulaire architectuur op die alle veranderingen en uitbreidingen kan faciliteren waarna ik deze architectuur kan implementeren op een STM32H7 dual-core processor (Figuur 2) voor het besturingssysteem van de 4-op-1-rij robot.



Figuur 2 STM32H7

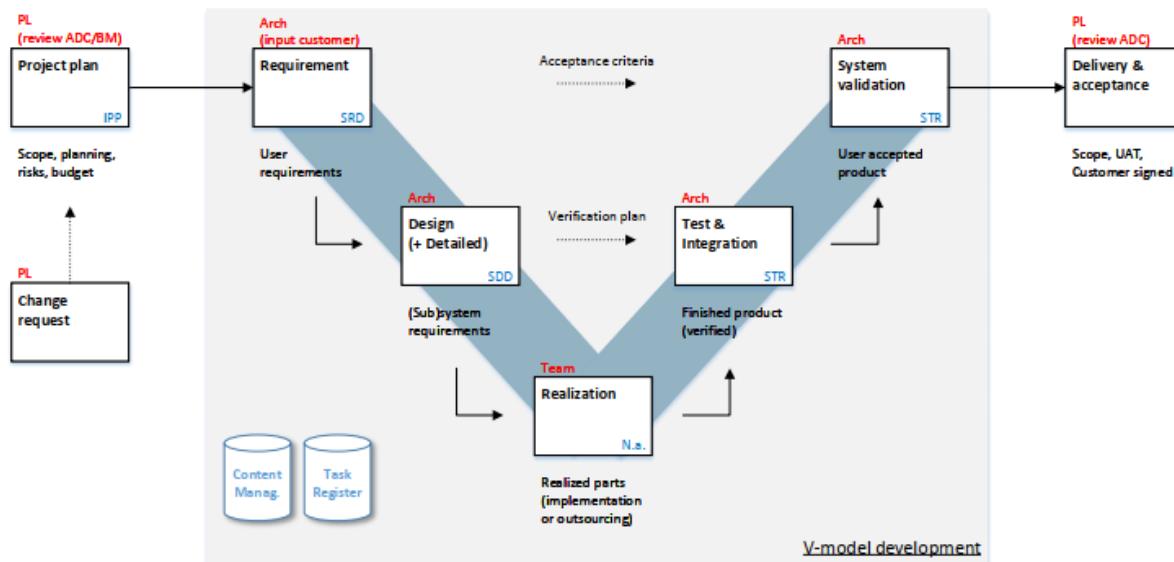
Projectresultaat

- Flowcharts;
- Een SAD;
- Er een BSP gemaakt te worden;
- Het individueel testen van de BSP modules;
- Een core van de STM32H7 dient gebruikt te worden voor real-time motion control;
- De tweede core van de STM32H7 dient gebruikt te worden voor machine handeling;
- Demo's om aan te tonen dat de verschillende BSP modules werken.

3 Projectactiviteiten

De 4-op-1-rij robot is een bestaand project. Binnen dit project moet het besturingssysteem opnieuw worden ontwikkeld. Door de architectuur van de grond af modulaire op te bouwen ontstaat er structuur in het proces. Om dit te realiseren is het belangrijk om vooraf een heldere opdracht te formuleren en kaders af te bakenen. Dit ga ik doen door het V-model toe te passen.

Het V-model is een project methode die structuur aanbrengt in de doorloop van het project. Voor elke specificatie- of ontwerpfase aan de linkerzijde, is een corresponderende integratiefase aan de rechterzijde. Elke fase aan de rechterzijde van het project kan geverifieerd en gevalideerd worden door de fase aan de linkerzijde (Figuur 3).



Figuur 3 V-model binnen ALTEN

De eerste vier weken staan in het teken van de requirements opstellen. Dit wordt gedaan door goed te overleggen met mijn technisch begeleider en de verwachtingen en (on)mogelijkheden te bespreken en bij te stellen. Verder zal er bestaande documenten over de 4-op-1-rij robot gelezen moeten worden. Om bekend te worden met de te gebruiken programmeer omgeving (STM32cubeIDE) kan er een STM32 development board gebruikt worden om de basis van een STM32 onder de knie te krijgen.

Als de opdracht duidelijk is omschreven en de requirements opgesteld zijn, kan er een start worden gemaakt aan de design fase. Dit wordt gedaan door een SAD op te stellen van de 4-op-1-rij robot. Hierin worden diagrammen opgesteld om de verhoudingen tussen de software en hardware te visualiseren.



Als het SAD is goed gekeurd kan er aan de realisatie fase begonnen worden. De realisatie fase bestaat uit de architectuur implementeren van de BSP modules. Het SAD is het grootste deel van mijn stage en zal de meeste tijd in beslag nemen.

In de test en integratie fase kan de ontwikkelde embedded software ook daadwerkelijk getest en geïmplementeerd worden op een STM32H7 dual-core processor development board die aanwezig is bij ALTEN. Ook de 4-op-1-rij robot zelf is beschikbaar en (deels) functioneel. Mocht er een deel van de opstelling niet werken dient er een mock-up code gemaakt te worden om het specifieke BSP toch te testen. Een mock-up is een stuk code waarmee je de werking van een software blok kan testen zonder fysieke componenten.

De laatste fase is systeem validatie. Voor mij is deze fase het inleveren van mijn afstudeerverslag en het verdedigen van mijn project op de Fontys.

Om op de hoogte te blijven wat de andere stagiaires doen en om aan te geven als problemen ontstaan is er dagelijks een stand-up meeting ingepland. Hierin vertel je kort wat je de dag ervoor hebt gedaan, wat je vandaag gaat doen en indien van toepassing problemen waar je tegen aanloopt. Zo is iedereen op de hoogte van elkaars werk en kunnen mensen kennis delen als er iemand vast loopt. Verder is er elke vrijdag een korte demo als een soort week samenvatting. Hier zijn ook consultants van ALTEN aanwezig die tips of hints kunnen geven als je vastloopt.

Helaas is er in verband met corona wel een tijdschema opgesteld voor de dagen dat de stagiaires op kantoor aanwezig mogen zijn. Dit betekent dat er twee dagen in week op kantoor gewerkt wordt en twee dagen thuis. Op vrijdag is er de mogelijkheid om naar kantoor te komen als dit nodig blijkt te zijn. Verder kan er worden overlegd met de business manager als er met hardware gewerkt moet worden om meerdere dagen naar kantoor te komen.



4 Projectgrenzen en randvoorwaarden

Een project goed afbakenen is erg belangrijk, zodat je aan het eind van de afstudeerstage kan verantwoorden wat er wel en niet bereikt is. Om te beginnen wordt de tijd en budget afgebakend. Dit project heeft een loop tijd van 100 dagen en zal daarbinnen uitgevoerd worden.

Begin datum: **01-02-2022**

Einddatum: **31-6-2022**

Budget: Er is geen budget. Het product wordt ontwikkeld zonder kosten.

De randvoorwaarden verzorgen de verdere afbakening van het project.

Randvoorwaarden

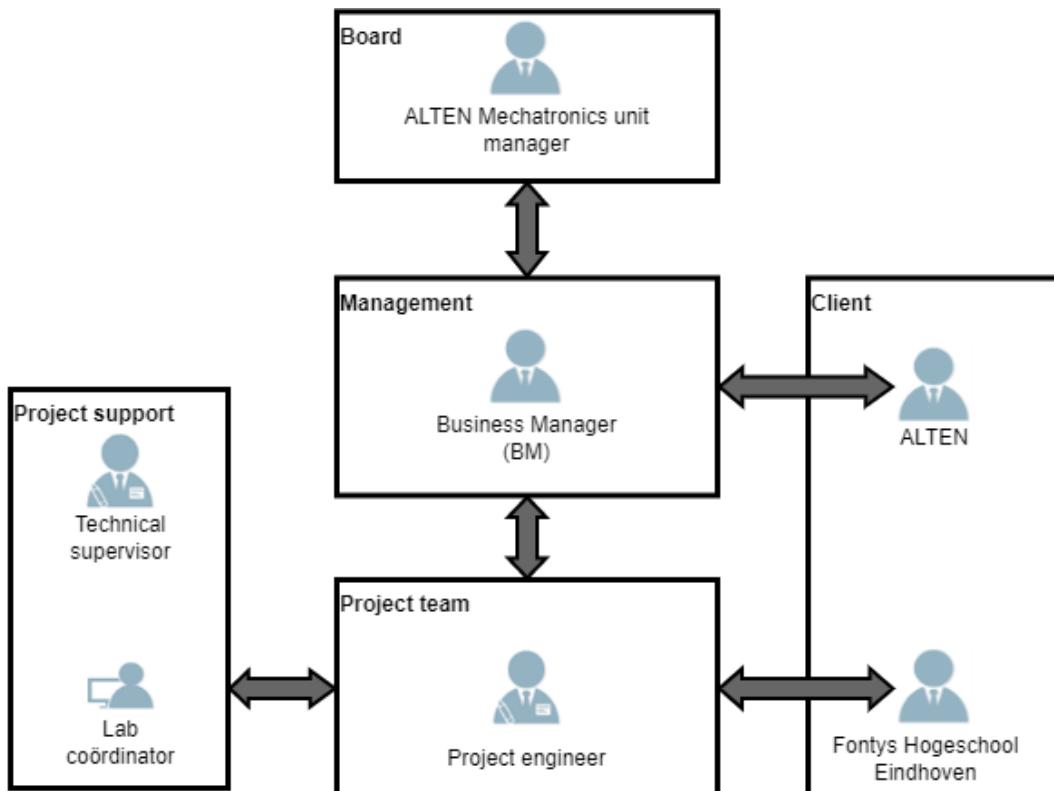
- Voor de eerste core van de STM32H7 wordt er geprogrammeerd op basis van C of C++;
- Voor de tweede core van de STM32H7 is de programmeertaal vrije keuze;
- Er dienen Flowcharts opgesteld te worden om de flow door het systeem te bepalen en in kaart te brengen;
- Er dienen diagrammen gemaakt te worden die de relatie van de hardware met software weergeeft;
- Voor een stap afgerond kan worden moet er goed keuring komen van de technisch manager;
- Het PvA moet goed gekeurd worden door school;
- Documentatie (SRD, SAD, Eindverslag) dient gemaakt te worden om de voortgang bij te houden en voor later onderzoek/uitbreiding;
- Aanwezigheid op kantoor op de daartoe aangewezen dagen is verplicht (maandag en woensdag);
- Aanwezigheid op kantoor buiten de daartoe aangewezen dagen moet overlegd worden met de business manager;
- Een timesheet dient op tijd ingevuld te worden;
- Het project wordt uitgewerkt tot een prove of concept;
- De hardware en mechanica van het systeem wordt niet in dit project meegenomen;
- De demo's van de BSP dienen goed gekeurd te worden door de technisch supervisor.

Ik streef ernaar om binnen de stageperiode van 100 dagen het besturingssysteem volgens een gestructureerde architectuur te herontwerpen naar de functionaliteit die het oude besturingssysteem ook had. Mocht er nog tijd over zijn dan kunnen extra functionaliteiten worden geïmplementeerd.

Het project is afgerond als alle stakeholders tevreden zijn over het eindresultaat. Voor dit project is dat: Aniel shri als technisch manager, Gijs Haans als business manager, ALTEN Mechatronics afdeling als eind verantwoordelijke, Jeedella S.Y. Jeedella als schoolmentor en ALTEN als eind klant.

Stakeholders

Stakeholders spelen een grote rol in het project. Het is van belang om in kaart te brengen wie betrokken is bij het project. In Figuur 4 is een visuele representatie te zien van de stakeholders. **Fout! Verwijzingsbron niet gevonden.** geeft aan welke personen welke rol hebben binnen het project en wat het aandeel van iedereen is.



Figuur 4 Project organisatie

Tabel 1 Rolverdeling

Naam	Rol	Verantwoordelijkheid
Pascal Faatz	Project engineer	Het project uitvoeren volgens requirements voor de klant (board/ management ALTEN)
Aniel Shri	Technical supervisor	Technische support leveren aan het projectteam om tot een goed eindresultaat te komen
Gijs Haans	Business manager	Begeleiden van de individuele ontwikkeling van de leden van het project team. Verder bewaakt de business manager het proces en de planning
Jeedella S.Y. Jeedella	Fontys Hogeschool Eindhoven	Het project, proces en de student worden vanuit Fontys begeleid op technisch en individueel gebied. Aan het einde van 20 weken wordt het project door de Hogeschool begeleider beoordeeld.
Jeroen Wilbers	Lab coördinator	Coördineren van het veilig gebruik van het lab en aanspreekpunt voor bestellingen van componenten.
Aniel Shri en Gijs Haans (als klant)	ALTEN	Beoordelen als klant de requirements en accepteren uiteindelijk het eindproduct.
Chris Kalis	ALTEN Mechatronics unit manager	Opdrachtgever en eindverantwoordelijk voor het project



5 Tussenresultaten

Tussenresultaten zijn belangrijk om aan te tonen dat er progressie zit in het project. Doormiddel van de volgende tussenresultaten wordt de voortgang binnen het project gewaarborgd.

- SRD;
- SAD;
- Goedkeuring totale systeem architectuur;
- Per module diagram een BSP module voor de STM32H7 ontwikkelen;
- Per BSP module een test uitvoeren;
- Verdelingsschema per core van de STM32H7.

6 Planning

Het is belangrijk om een goede planning te hebben. Vanuit ALTEN heb ik een template gekregen voor een Gantt chart. Deze heb ik ingevuld op basis van wat ik nu denk te onderwerpen en de taken die voltooid moeten worden (Appendix B). Verder zijn alle deadlines weergegeven in Tabel 2.

Tabel 2 Deadlines

Deadlines:	Datum:
Plan van Aanpak	18-2-2022
SRD concept	11-3-2022
SAD concept	25-3-2022
Test rapport	1-7-2022
Eindverslag	7-6-2022 (onder voorbehoud)
Afstudeerzitting	17-6-2022 – 18-7-2022 (onder voorbehoud)



7 Risico's

Omdat het binnen mijn project gaat om een herontwikkeling zijn de risico's te over zien. Alleen het besturingssysteem van 4-op-1-rij robot moet een upgrade krijgen. De algemene risico's van het project zijn verwerkt in een risicoanalyse in Appendix A. Zoals te zien in de grafiek van Appendix A is de grootste valkuil de complexiteit van het project. Ook tijd kan een valkuil worden. Verdere valkuilen die niet benoemd zijn in de risicoanalyse zijn: Ziekte door corona maar ook corona zelf. Mocht corona weer oplaaien en thuis werken de norm worden dan kan dit voor vertraging zorgen. Het bestellen van componenten kan lang duren. Door het te kort aan materiaal in de markt kan het lang duren voor een component geleverd wordt.

Als een van de valkuilen mocht plaats vinden is het van noodzaak om zo snel mogelijk de juiste personen te informeren en om hulp te vragen. Dit zijn mijn technisch begeleider Aniel over technische valkuilen en mijn business manager Gijs over persoon gerelateerde valkuilen. In beide gevallen moet ik school ook op de hoogte stellen van een valkuil. Samen met deze personen en school zal een passende oplossingen gevonden moeten worden.

Appendix A. Risico analyse

Risico-analyse	Print
4-op-1-rij robot	9-2-2022

Bij een risicopercentage > 50%, dient het project niet in deze vorm worden uitgevoerd.

Categorie	Risico	Waarde *	Factor **	Zwaarte **	Risicotot.
Tijdsfactor <small>J. maak keuze J.</small>					
1	Geschatte looptijd van het project	3 - 6 maanden	1	4	4
2	Kent het project een definitieve deadline	Ja	2	4	8
3	Is de tijd voldoende om het project te realiseren	Voldoende	1	4	4
Complexiteit van het project <small>J. maak keuze J.</small>					
4	Aantal functionele deelgebieden dat betrokken is	3+	3	4	12
5	Aantal functionele deelgebieden dat gebruik gaat maken van de resultaten	4	2	2	4
6	Gaat het om een aanpassing of een nieuw project	Grote aanpassingen	2	5	10
7	In hoeverre zullen bestaande verantwoordelijkheden moeten wijzigen	Gemiddeld	2	5	10
8	Zijn er andere projecten afhankelijk van dit project	Nee	0	5	0
9	Wat zal de houding zijn van de gebruikers	Geïnteresseerd	1	5	5
10	Zijn er deelprojecten, is de voortgang afhankelijk van de coördinatie hiertussen	Enigszins	2	3	6
De projectleiding <small>J. maak keuze J.</small>					
16	Is de projectleiding materiedeskundig	Zeer deskundig	0	3	0
17	Hoe deskundig is de projectleiding mbt de projectplanning	Zeer deskundig	0	3	0
18	Hoeveel ervaring heeft de projectleider met projecten als deze	Veel ervaring	0	3	0
19	Hoe deskundig zijn de adviseurs op het te onderzoeken gebied	Zeer deskundig	0	5	0
20	Hoe deskundig zijn de materiedeskundigen op het te onderzoeken gebied	Zeer deskundig	0	5	0
21	Hoe betrokken zijn de verantwoordelijke lijnmanagers bij het project	Sterk betrokken	0	5	0
22	Is de kans groot dat de samenstelling van de projectgroep wijzigt tijdens het project	Kleine kans	0	5	0
23	Worden door de projectgroep standaardmethoden gebruikt	Ja, een aantal	2	4	8

Categorie	Risico	Waarde *	Factor **	Zwaarte **	Risicotot.
Duidelijkheid van het project					
24	Zijn probleem en doelstelling voldoende bekend bij alle projectleden	Ja, iedereen imaak keuze!	0	5	0
25	Is het onderzoeksgebied nauwkeurig vastgelegd	Redelijk	2	5	10
26	Is er voldoende afbakening met andere projecten	Redelijk	1	4	4
27	Is er voldoende tijd gepland voor afstemming en besluitvorming	Voldoende	0	4	0
28	Zijn de randvoorwaarden duidelijk	De meeste wel	1	4	4
29	Werken de randvoorwaarden beperkend genoeg	Redelijk	2	5	10
					Totaal 99
					Risicopercentage *** 23%

Opmerking: Dit model geeft slechts een zeer grove indicatie van risico's

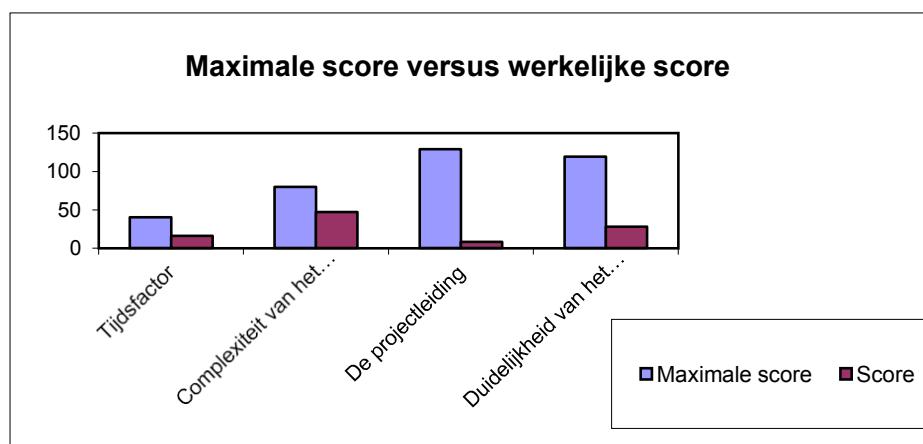
* Waarde gekozen door projectleider.

** Hoogte factor en waarde staan vast.

*** Risicopercentage is de totaalscore gedeeld door 433 (maximale score) maal 100. Let op: dit is slechts een indicatie

Aangezien het risico-percentage een totaalbeeld geeft, kan het zijn dat een bepaalde categorie wel voor een hoog risico zorgt. Hieronder een specificatie per categorie om eventuele verbeterpunten zichtbaar te maken.

Categorie (met maximale score versus werkelijke score)				
Tijdsfactor	Maximaal	40	Score	16
Complexiteit van het project	Maximaal	80	Score	47
De projectleiding	Maximaal	129	Score	8
Duidelijkheid van het project	Maximaal	119	Score	28



Appendix B. Planning

Project Planner

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

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

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE	PERIODS																												
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Plan van aanpak	1	3	1	3	100%																													
Documentatie doorlezen	1	4	1	4	100%																													
STM32cube omgeving leren	2	5	1	6	100%																													
Onderzoek hardware	3	4	3	5	100%																													
SRD	4	2	4	2	100%																													
SAD	6	5	6	7	100%																													
Programmeren BSP's STM32H7	11	10	11	10	80%																													
Testen	17	4	17	4	80%																													
Test rapport	19	2	19	2	20%																													
Eindverslag	12	7	12	7	100%																													



III. Software Architecturale Document

Software Architectuur Document – **4-op-1-rij robot**

Versie 3
Datum: 3-5-2022

ALTEN
Pascal Faatz

Eindversie



Versie geschiedenis

Versie	Datum	Status	Schrijver	Opmerking
1	31-3-2022	Opzet	Pascal Faatz	Eerste opzet tot hoofdstuk 8
2	13-4-2022	Opzet	Pascal Faatz	Verwerken feedback Jeedella, Aniel en Berend
3	3-5-2022	Afronding	Pascal Faatz	Hoofdstuk 8 en verder

Acroniemen en afkortingen

Term	Uitleg
BSP	Board Support Package

Gerefereerde documenten

Id	Referentie	Titel	Datum	Schrijver
D1	SRD_Pascal_Faatz_V3.docx	SRD	16-3-2022	Pascal Faatz



Inhoudsopgave

1	INTRODUCTIE EN GOALS	5
1.1	REQUIREMENTS OVERZICHT.....	5
1.2	QUALITY GOALS	6
1.3	STAKEHOLDERS	6
2	ARCHITECTUUR BEPERKINGEN	8
3	PROJECT SCOPE EN CONTEXT	9
3.1	SYSTEEM CONTEXT	10
3.2	TECHNISCHE CONTEXT	11
4	OPLOSSINGSSTRATEGIE	13
5	BUILDING BLOCK VIEW.....	14
5.1	WHITE BOX STM32H7 DUAL-CORE.....	14
5.2	LEVEL 2.....	15
5.2.1	White box Cortex-M7	15
5.2.2	White box Cortex-M4	16
5.3	LEVEL 3.....	17
5.3.1	White box Motor controller	17
5.3.2	White box Coin color separator	18
5.3.3	White box Coin picker	18
6	RUNTIME VIEW	19
6.1	HIGH LEVEL OVERZICHT.....	19
6.2	ROBOT MOVE.....	21
6.3	HUMAN MOVE	22
6.4	CLEAN UP	23
7	DEPLOYMENT VIEW	25
7.1	NUCLEO-H755ZI-Q.....	25
7.2	PIN-OUT NUCLEO-H755ZI-Q	26
7.3	HARDWARE LAY-OUT	28
7.4	MASTER-MINION VERHOUDING.....	29
8	MODULAIRE SOFTWARE IMPLEMENTATIE	30
8.1	CORTEX-M7 CORE	30
8.2	CORTEX-M4 CORE	31
8.2.1	Motor controller.....	31
8.2.2	Coin color separator	32
8.2.3	Coin picker.....	32
8.2.4	Board opener.....	33
8.2.5	User detect	33
9	FOUTAFHANDELING.....	34

Lijst met figuren

Figuur 1 Blok diagram 4-op-1-rij	5
Figuur 2 Project organisatie	6
Figuur 3 Project context	9
Figuur 4 Systeem context	10
Figuur 5 Technical context	11
Figuur 6 V-model.....	13
Figuur 7 BBV STM32H7 level 1	14
Figuur 8 BBV Cortex-M7 level 2	15
Figuur 9 BBV Cortex-M4 level 2	16
Figuur 10 BBV Motor controller level 3	17
Figuur 11 BBV Coin color seperator level 3.....	18
Figuur 12 BBV Coin picker level 3	18
Figuur 13 State machine Cortex-M7	19
Figuur 14 State machine Cortex-M4	20
Figuur 15 Sequence diagram Robot move	21
Figuur 16 Sequence diagram Human move	22
Figuur 17 Sequence diagram Clean-up	23
Figuur 18 Sequence diagram return coin routine.....	24
Figuur 19 NUCLEO-H755ZI-Q	25
Figuur 20 pin-out STM32H755ZITx	26
Figuur 21 Hardware lay-out 4-op-1-rij met STM32H7.....	28
Figuur 22 Master-minion verhouding	29
Figuur 23 IBD game controller	30
Figuur 24 IBD Motor controller	31
Figuur 25 IBD Coin color seperator	32
Figuur 26 IBD Coin picker	32
Figuur 27 IBD Board opener	33
Figuur 28 IBD User detect.....	33
Figuur 29 State machine met error handler	34

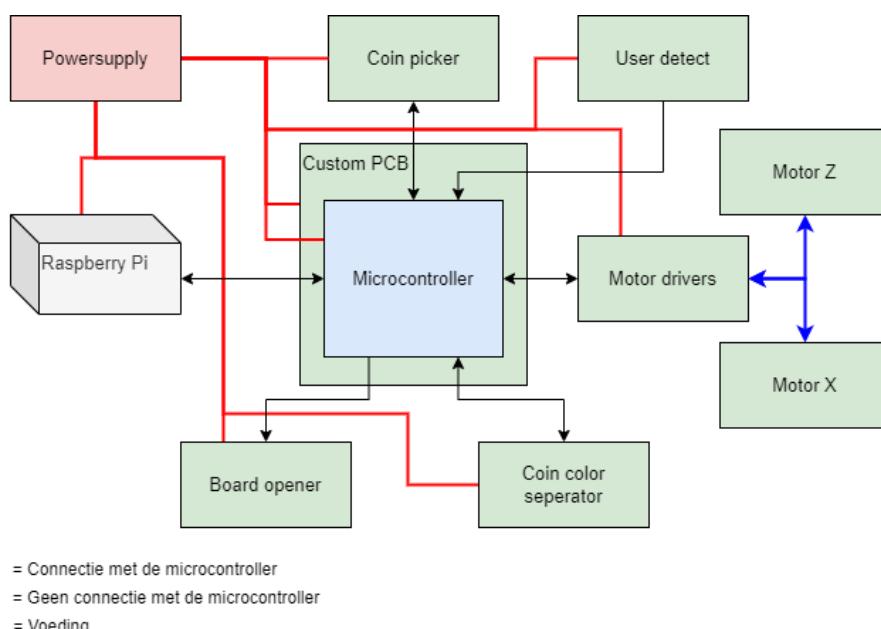
Lijst met tabellen

Tabel 1 Quality goals	6
Tabel 2 Stakeholders	7
Tabel 3 Architectuur beperkingen	8
Tabel 4 Beschrijving Project context.....	9
Tabel 5 Beschrijving Systeem context	10
Tabel 6 Beschrijving Technical context.....	11
Tabel 7 Oplossingstrategie	13
Tabel 8 Beschrijving BBV STM32H7 level 1	14
Tabel 9 Beschrijving BBV Cortex-M7 level 2	15
Tabel 10 Beschrijving BBV Cortex-M4 level 2	16
Tabel 11 Beschrijving BBV Motor controller level 3.....	17
Tabel 12 Beschrijving BBV Coin color seperator level 3	18
Tabel 13 Beschrijving BBV Coin picker level 3	18
Tabel 14 pin-out tabel	26

1 Introductie en goals

ALLEN heeft in eigen beheer een robot ontwikkeld die middels een algoritme 4-op-1-rij kan spelen tegen een menselijke tegenstander. Met behulp van industriële componenten wordt de 4-op-1-rij robot gebouwd om de kennis van verschillende systemen te demonstreren. De robot zal onder andere op beurzen en open dagen gebruikt worden als demonstratie unit, waarbij deze beschikbaar is voor voorbijgangers om een ronde te spelen. Aan de voorzijde kan de speler dan zijn/haar zet plaatsen op het bord, waarna de 4-op-1-rij robot een zet zal bedenken en uitvoeren. Hiervoor houdt het systeem bij welke zetten door zijn tegenstander gespeeld zijn. Wanneer de zet door de robot bepaald is zal de combinatie van het X-Z platform met roterende vacuüm gripper een steen op pakken en op de gekozen plek in het spel invoeren. Zodra het spel afgelopen of gereset is, zal de 4-op-1-rij robot alle fiches verzamelen en op kleur sorteren om zo klaar te zijn voor de volgende ronde.

Momenteel draait de 4-op-1-rij robot op een Raspberry Pi + STM32 microcontroller (Figuur 1). Op de Raspberry Pi draait het algoritme om de volgende zet van het systeem te bepalen en op de STM32 controller draait het besturingssysteem. Dit project is binnen ALLEN in eerste instantie gebruikt om consultants die op een tussen periode niet op een project zitten toch een uitdaging te geven. Hierdoor is er, over een langere periode, door meerdere consultants aan gewerkt. Dit heeft geresulteerd in een zeer onoverzichtelijke en onduidelijke architectuur van de software en hardware. Dit geld dus ook voor het besturingssysteem van de 4-op-1-rij robot.



Figuur 1 Blok diagram 4-op-1-rij

1.1 Requirements overzicht

Door de onduidelijke architectuur is het zeer lastig om het besturingssysteem uit te breiden of een upgrade toe te passen. Omdat de 4-op-1-rij robot ook op beurzen staat en aantoon wat ALLEN in huis heeft is het van noodzaak dat de robot goed te onderhouden is en makkelijk een upgrade kan ondergaan. Hiervoor is een complete re-design van het besturingssysteem nodig om de onoverzichtelijke en onduidelijke architectuur op te lossen en upgrades mogelijk te maken.

Een van de hoofd requirements is dan ook om een gestructureerde architectuur en modulaire implementatie in te voeren. Verder is het van belang state- en flowdiagrammen op te stellen om de werking van het systeem visueel weer te geven. Ook moet er een BSP (Board Support Package) gemaakt worden.

Voor de volledige lijst van requirements refereert aan het SRD (D1).

1.2 Quality goals

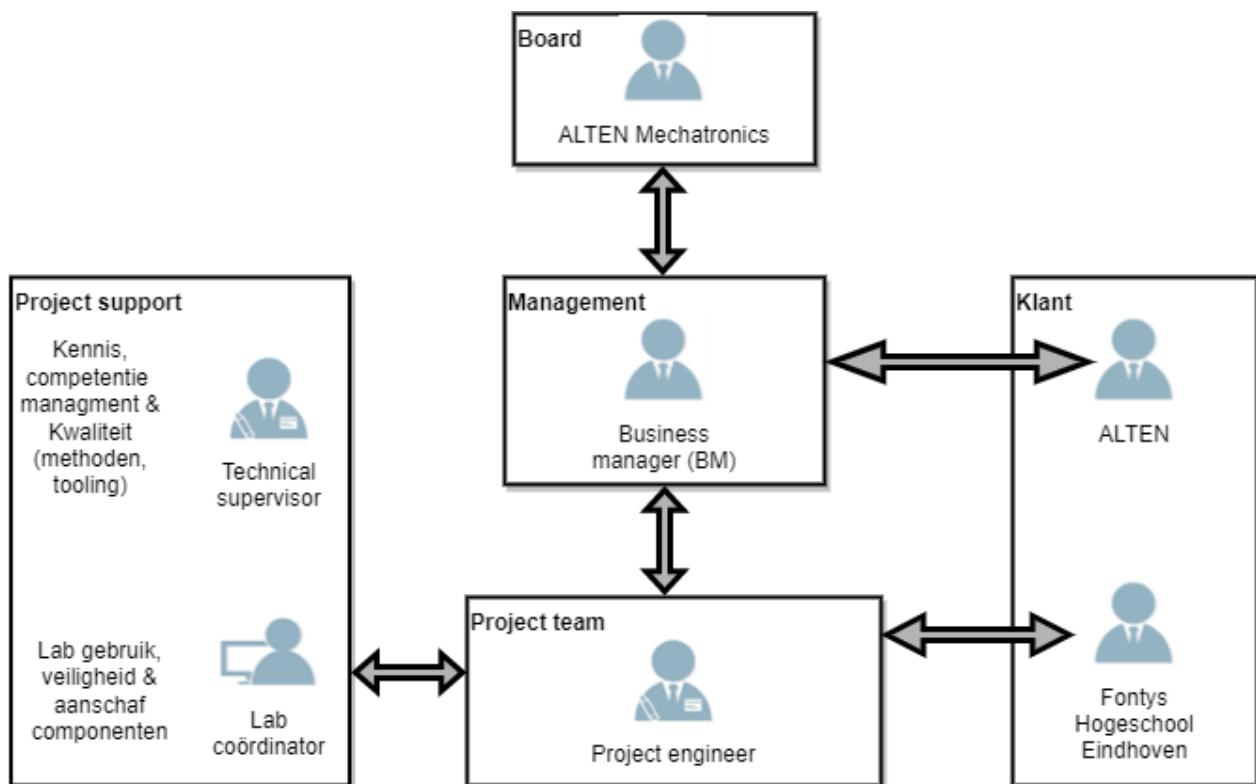
De quality goals zijn essentieel om aan te geven aan welke kwaliteitseisen het systeem moet voldoen. Deze goals worden in een later stadium ook getest en gecheckt of het systeem hier aan voldoet.

Tabel 1 Quality goals

Prioriteit	Quality goal	Concreet scenario
1	Makkelijk te begrijpen (Gestructureerde architectuur)	Mensen die aan de 4-op-1-rij robot gaan werken moet met het lezen van de architectuur in een keer snappen hoe de hardware en software samen hangt.
2	Onderhoud en upgrades (Modulaire opbouwen van software blokken)	Mensen die een upgrade of aanpassing gaan doen aan de 4-op1-rij moeten software blokken kunnen aanpassen of vervangen zonder dat dit andere delen van het systeem beïnvloed.
3	Robuust	De 4-op-1-rij moet betrouwbaar zijn en werken onder alle omstandigheden wanneer het systeem draait.

1.3 Stakeholders

De stakeholders zijn een belangrijk onderdeel van het project. Zij bepalen de requirements en stellen eisen aan het eind product. In Figuur 2 is de verdeling van stakeholders voor dit project weergegeven.





Tabel 2 Stakeholders

Naam	Rol	Verantwoordelijkheid
Pascal Faatz	Project engineer	Het project uitvoeren voor de betreffende klanten en board/management.
Aniel Shri	Technical supervisor	Binnen het project technische steun geven aan het project team. Dit houdt het uitdagen van het team om tot een goede oplossing en uitwerking te komen.
Gijs Haans	Business manager	Binnen het project persoonlijke ontwikkeling ondersteunen van het project team. Verder houdt de business manager.
Jeedella Jeedella	Fontys Hogeschool Eindhoven	Het project vanuit school ondersteunen op technisch en persoonlijk vlak. Aan het eind een beoordeling geven over het project.
Jeroen Wilbers	Lab coördinator	Het veilig gebruik maken van het lab en het aanspreekpunt voor bestelling die gedaan moeten worden.
Aniel Shri en Gijs Haans (als klant)	ALTEN	Als eindklant de requirements beoordelen en het eindproduct gebruiken.
Chris Kalis	ALTEN Mechatronics unit	De leiding over alle project partijen binnen ALTEN. Verder is de afdeling eindverantwoordelijke voor het project.

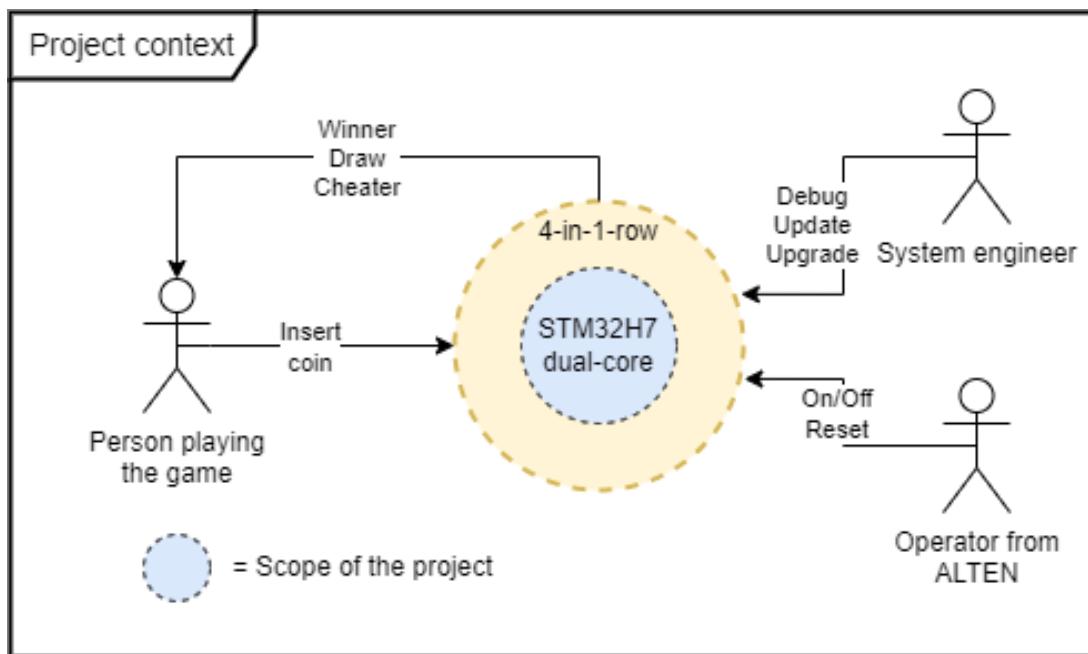
2 Architectuur beperkingen

Tabel 3 Architectuur beperkingen

Architectuur beperking	Omschrijving
Er wordt gebruik gemaakt van een STM32H755 dual-core nucleo-144 board.	De keuze van de microcontroller staat vast. Met deze microcontroller moet dus gewerkt worden.
De hardware van de 4-op-1-rij staat vast.	De hardware die er nu is wordt gebruikt om de software voor te schrijven.
Het spel verloop van de 4-op-1-rij staat vast.	Omdat de 4-op-1-rij al werkend is geweest is de globale werken van te voren bekend.
Het project moet binnen 100 werkdagen worden uitgevoerd.	Het project vind plaats binnen een school semester en moet dus binnen deze tijd worden uitgevoerd.
De Raspberry Pi geeft de volgende zet door.	De Raspberry Pi is de processor die de volgende zet door geeft aan de microcontroller.

3 Project scope en context

Figuur 3 geeft een globale weergave van de personen die met het systeem werken en wat zei als input hebben op het systeem of wat ze van het systeem terug krijgen. Verder toont het de scope van het project. De scope van het project is de STM32H7 dual-core en de architectuur daarvan.



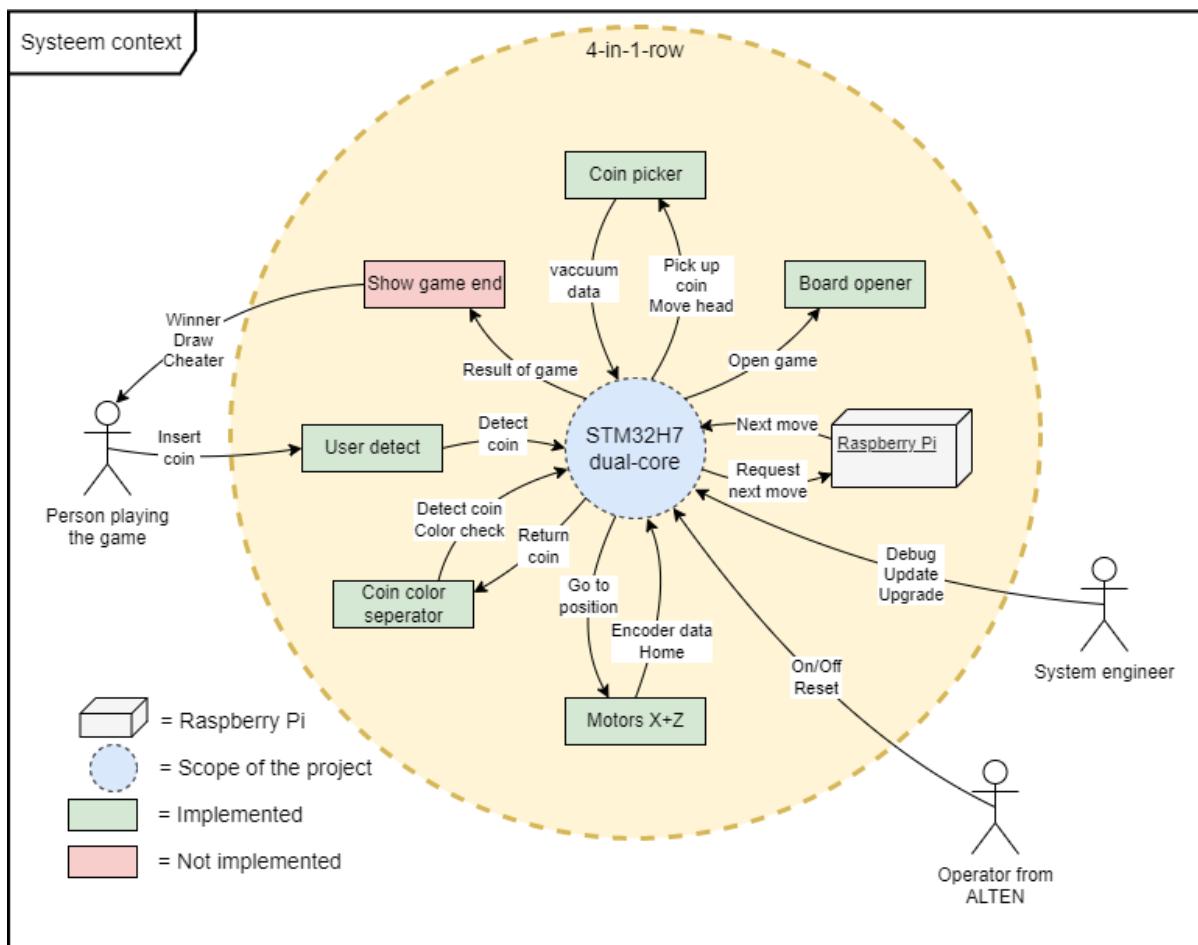
Figuur 3 Project context

Tabel 4 Beschrijving Project context

Gebruikers	Beschrijving
Persoon die 4-op-1-rij speelt	De persoon die 4-op-1-rij speelt moet als het zijn beurt is goed nadenken kolom hij zijn fiche wil doen om de robot te verslaan. Als een zet bekent is wordt het fiche in het 4-op-1-rij bord gestopt. Verder kan de persoon aan het einde van het spel zien wat het resultaat is.
Systeem engineer	De systeem engineer kan de 4-op-1-rij debuggen als er een fout optreedt. Verder kan de engineer een update/upgrade implementeren.
Operator vanuit ALTEN die de 4-op-1-rij bedient	Deze persoon bedient de 4-op-1-rij op beurzen of anderen gelegenheden. Deze persoon kan de 4-op-1-rij aan-/uitzetten of hem resetten.

3.1 Systeem context

Figuur 3 geeft een globaal overzicht van de betrokken personen bij het systeem. Figuur 4 zoomt hier verder op en geeft een visuele representatie van de subonderdelen van de 4-op-1-rij. Ook laat het de relatie zien met de STM32H7 dual-core door middel van de inputs en outputs. Dit is van belang zodat alle stakeholders een idee krijgen wat er in grote lijnen binnen het systeem gebeurt.



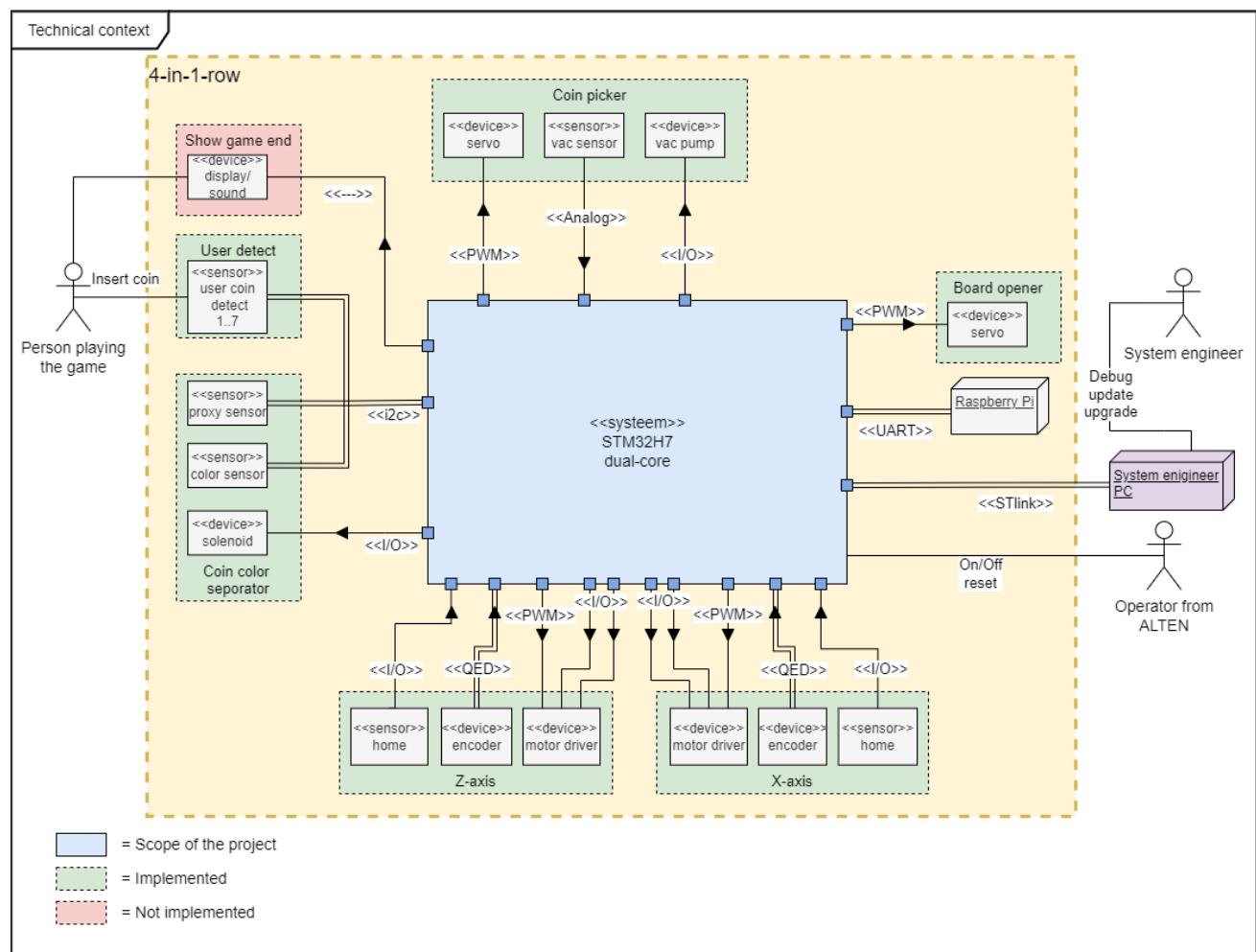
Figuur 4 Systeem context

Tabel 5 Beschrijving Systeem context

Hardware blok	Functie
Raspberry Pi	Op de Raspberry Pi draait een algoritme dat de volgende stappen van de robot bepaalt.
Motors X+Z	De motoren zorgen ervoor dat de robot zijn vaccuum gripper over een X- en Z-as van het 4-op-1-rij bord kan bewegen.
Coin color separator	Als het spel is afgelopen zorgt de coin color separator ervoor dat elk fiche op kleur herkent wordt. Heeft het fiche de kleur van de speler dan wordt het fiche naar de bak van de speler geschoten. Heeft het fiche de kleur van de robot dan wordt het fiche in de eigen stapel gedaan.
User detect	Als de speler een fiche in het spel doet wordt dit gedetecteerd. Elke kolom van het 4-op-1-rij bord bevat een detectie punt.
Coin picker	De coin picker is de arm van de robot. Deze wordt verplaats door de motoren maar kan zelf ook bewegen om een fiche in het 4-op-1-rij bord te doen. Een fiche oppakken of los laten gebeurt doormiddel van een vaccuum gripper.
Board opener	De board opener zorgt ervoor dat alle fiches uit het 4-op-1-rij bord vallen als het spel is afgelopen.

3.2 Technische context

Figuur 5 zet Figuur 4 om in een beschrijving om in een technische representatie van de verhouding tussen de componenten en de scope. De verbindingen wordt niet globaal beschreven maar weergegeven door welke soort protocol of in-/output signaal ze verbonden zijn. Dit is van belang voor de stakeholders die aan de slag gaan met de hardware of architectuur.



Figuur 5 Technical context

Tabel 6 Beschrijving Technical context

Hardware blok	Sub-blokken	Omschrijving	Input naar STM32H7	Aantal lijnen	Output van STM32H7	Aantal lijnen
Raspberry Pi	-	De Raspberry Pi communiceert met de STM32H7 doormiddel van een UART verbinding.	UART Rx	1	UART Tx	1
Motors/driver X	Motor met encoder	De motor wordt aangestuurd door een PWM signaal, een lijn voor de richting en een lijn voor het ready signaal. Om de positie van de motor te bepalen is een encoder gebruikt. De encoder communiceert met een A en B lijn.	QED (A lijn en B lijn)	2	1 PWM, 2 I/O	3

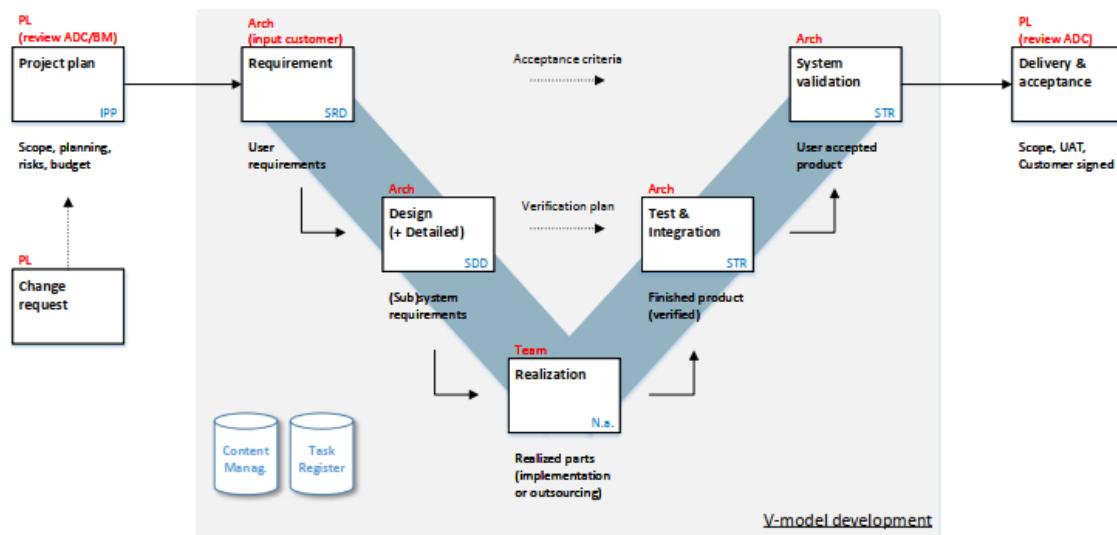
	Home/ End stop	De home/end stop geeft aan als de motor op het home punt gekomen is.	I/O	1	-	-
Motor/ driver Z	Motor met encoder	Zie Motor X.	QED (A lijn en B lijn)	2	1 PWM, 2 I/O	3
	Home/ End stop	Zie Motor X.	I/O	1	-	-
Coin color separator	Proxy sensor	De proxy sensor checkt of er een fiche aanwezig is in het verdeel systeem.	i2c (SDA, SCL)	2	-	-
	Color sensor	De color sensor checkt welke kleur het fiche heeft die in het verdeel systeem ligt.	i2c (SDA, SCL)	2	-	-
	Solenoid	Als een fiche de kleur van de speler heeft schiet de solenoid d.m.v. een flipper het fiche naar de bak van de speler.	-	-	I/O	1
User detect	-	De user detect detecteert als er door de speler een fiche in het systeem geworpen wordt.	i2c (SDA, SCL)	2	-	-
Coin picker	Servo	De servo zorgt ervoor dat de vaccuum gripper bewogen kan worden.	-	-	PWM	1
	Vaccum pump	De vaccuum gripper kan een fiche vast zuigen en loslaten.	-	-	I/O	1
	Vac sensor		Analoog	1	-	-
Board opener	-	De game opener zorgt doormiddel van een servo dat alle fiches uit het 4-op-1-rij bord vallen als het spel is afgelopen.	-	-	PWM	1

4 Oplossingsstrategie

Tabel 7 Oplossingstrategie

Goal/requirement	Aanpak	Link naar hoofdstuk
Gestructureerde architectuur	Om een gestructureerde architectuur voor de software van de 4-op-1-rij op te zetten wordt gebruik gemaakt van dit document. Aan de hand van de hoofdstukken en diagrammen in dit document wordt duidelijk hoe de 4-op-1-rij zich gedraagt, de software blokken samenhangen en de software met de hardware communiceert. Verder wordt er een top-down approach gebruikt door te beginnen bij de requirements. Vanaf de requirements wordt het systeem steeds verder ontleed.	
Modulaire opbouwen van software blokken	De software wordt modulaire opgebouwd om er voor te zorgen dat een upgrade of aanpassing makkelijk te faciliteren is. Eerst wordt uitgewerkt welke software blokken er zijn. Daarna hoe deze blokken met elkaar communiceren.	(Hoofdstuk 5,6,8)
Robuust	De 4-op-1-rij moet onder normale omstandigheden werken zoals van te voren vast gesteld. Dit kan worden gegarandeerd door alle software blokken individueel en samen uitgebreid te testen.	

Binnen het hele project wordt gebruik gemaakt van het V-model. Het V-model is een project methode die structuur aanbrengt in de doorloop van het project. Voor elke specificatie- of ontwerpfase aan de linkerzijde, is een corresponderende integratiefase aan de rechterzijde. Elke fase aan de rechterzijde van het project kan geverifieerd en gevalideerd worden door de fase aan de linkerzijde (Figuur 6).



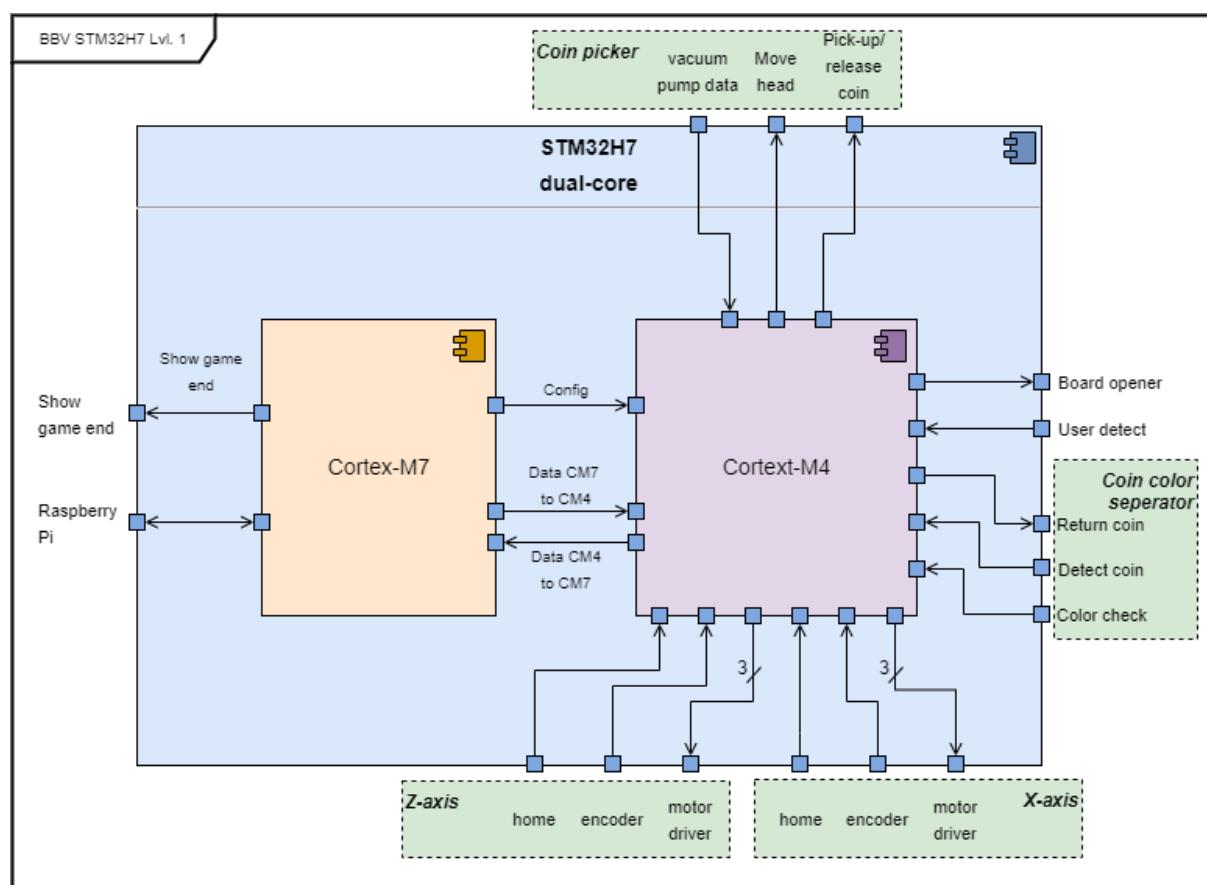
Figuur 6 V-model

5 Building block view

Dit hoofdstuk gaat dieper in op de in hoofdstuk 3 gepresenteerde diagrammen. Het idee van een building block view is om op het systeem stap voor stap verder in te zoomen. Dit wordt gedaan door middel van levels. Level 0 is voor dit project is Figuur 5 waarbij de STM32H7 dual-core een black box is. In level 1 wordt deze STM32H7 dual-core een white box die weer bestaat uit meerdere black boxes. In level 2 worden de black boxes uit level 1 white boxes met daarin black boxes. Tijdens elke stap worden de black boxes beschreven wat de verantwoordelijkheid en taken zijn. Dit is een goede manier om een systeem gestructureerd te ontleden. Het is ideaal om in overleg te gaan met de stakeholders op een abstract niveau zonder verdere details op hoe de implementatie plaats gaat vinden. Hierna kunnen er modulaire software blokken gecreëerd worden.

5.1 White box STM32H7 dual-core

Figuur 7 geeft level 1 weer en zoomt in op de STM32H7 dual-core. Ook toont het naar welke core de in-/outputs van STM32H7 dual-core gaan.



Figuur 7 BBV STM32H7 level 1

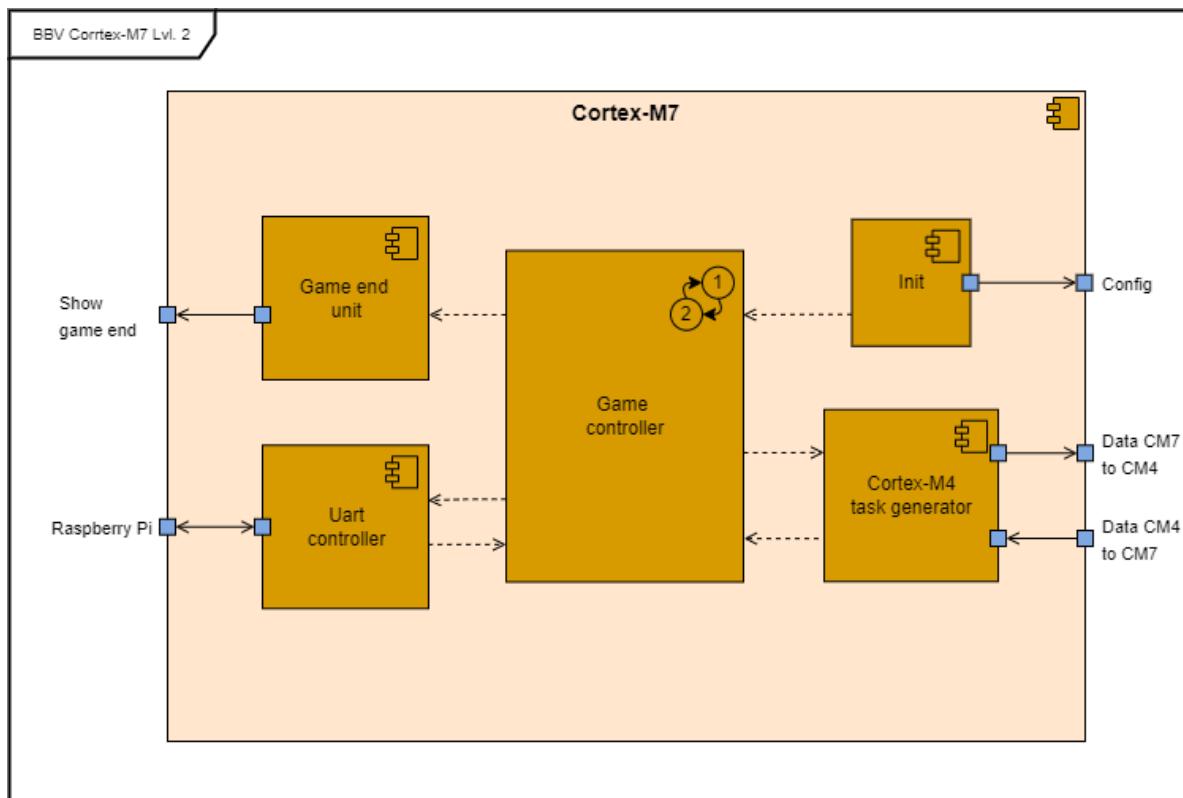
Tabel 8 Beschrijving BBV STM32H7 level 1

Black box	Beschrijving
Cortex-M7	De Cortex-M7 core van de STM32H7 wordt gebruikt voor de machine handling. Deze core is verantwoordelijk voor het spelverloop, de communicatie met de Raspberry Pi, Het initialiseren van het hele systeem en een output genereren over het resultaat van het spel.
Cortex-M4	De Cortex-M4 core van de STM32H7 wordt gebruikt voor de real-time motion control. Deze core is verantwoordelijk voor het afhandelen van alle hardware componenten zoals de Coin picker, Motoren, Coin seporator, Board opener en User detect.

5.2 Level 2

Dit hoofdstuk maakt van de relevante blackboxes uit level 1 white boxes en beschrijft hoe de interne blokken verbonden zijn.

5.2.1 White box Cortex-M7

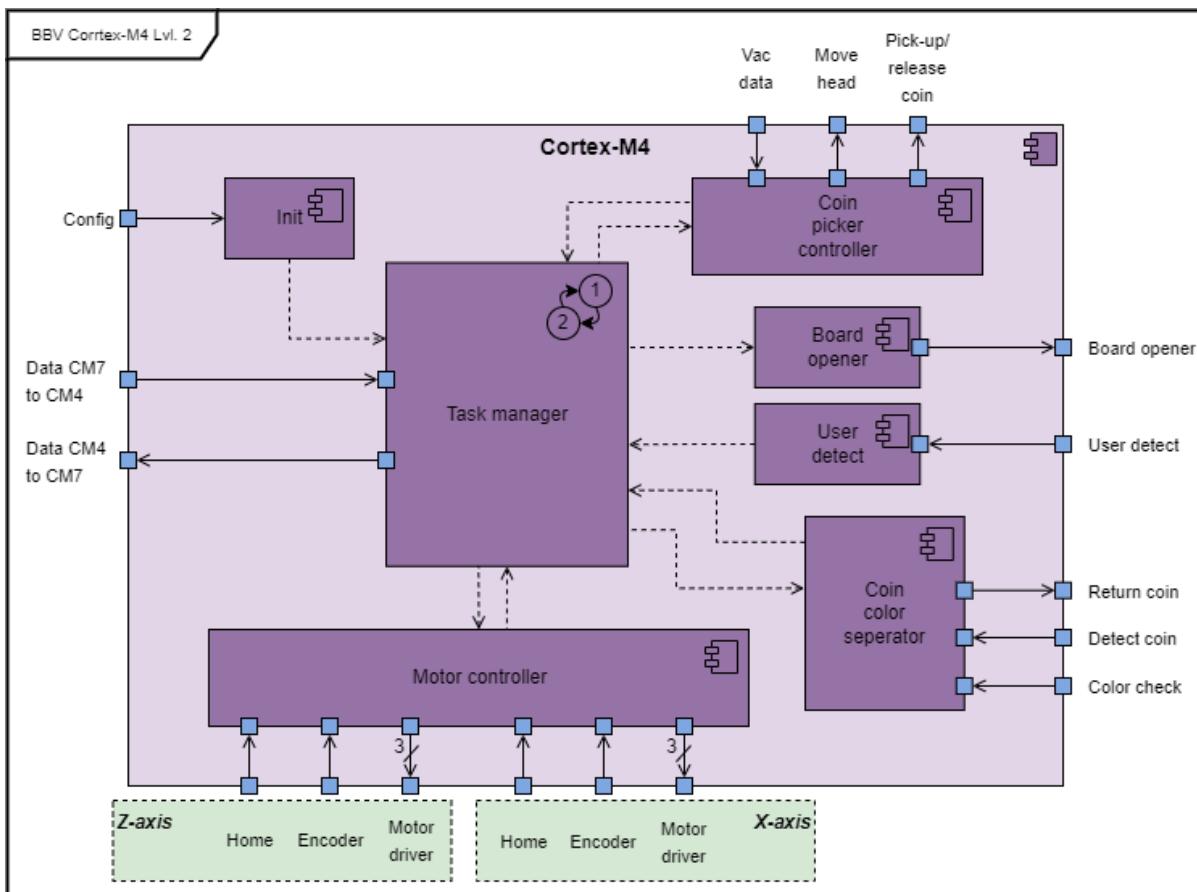


Figuur 8 BBV Cortex-M7 level 2

Tabel 9 Beschrijving 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 een 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.

5.2.2 White box Cortex-M4



Figuur 9 BBV Cortex-M4 level 2

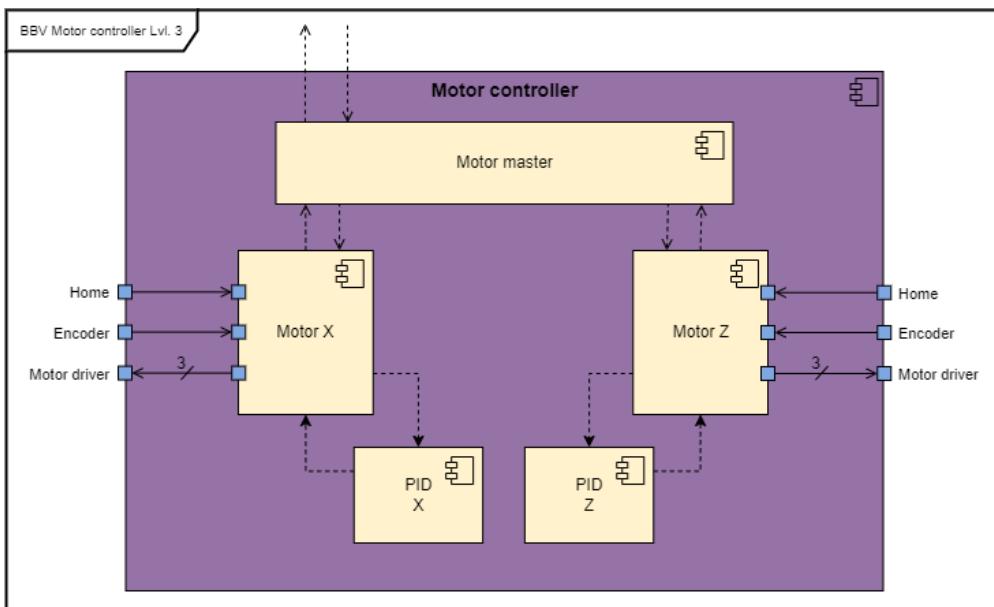
Tabel 10 Beschrijving BBV Cortex-M4 level 2

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. Door gebruik te maken van een "Task manager" kan de functionaliteit van de andere onderdelen modulair gebouwd worden. Die hoeven immers niet van elkaar bestaan/status af te weten. Die informatie wordt door de Task Manager bijgehouden.
Init	Wordt een 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 en de PID regelaar.
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 openen van het 4-op-1-rij bord als het spel klaar is.
Coin picker controller	De Coin picker controller implementeert alle functies voor het afhandelen van het oppakken en loslaten van een fiche.

5.3 Level 3

Dit hoofdstuk maakt van de relevante blackboxes uit level 2 white boxes en beschrijft hoe de interne blokken verbonden zijn.

5.3.1 White box Motor controller

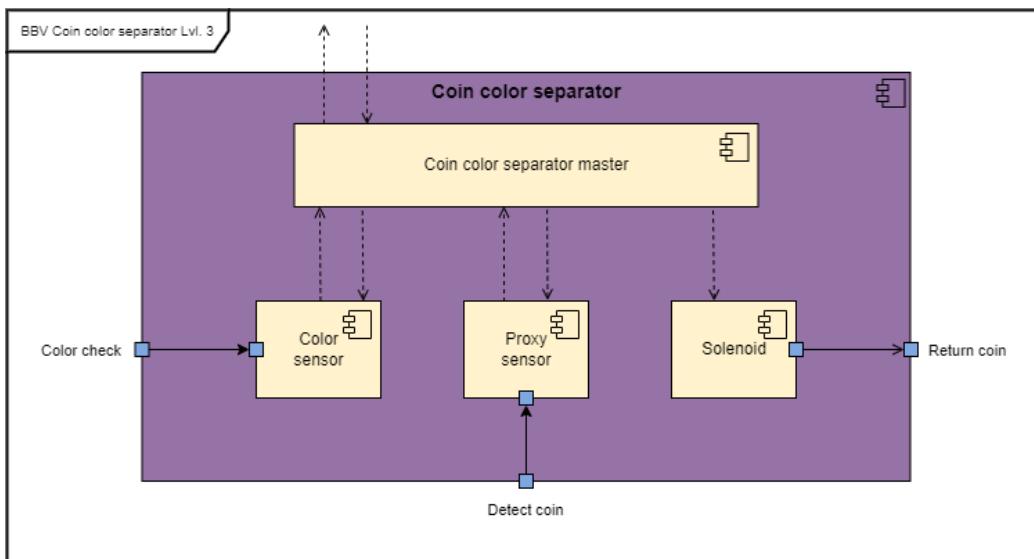


Figuur 10 BBV Motor controller level 3

Tabel 11 Beschrijving BBV Motor controller level 3

Black box	Beschrijving
Motor master	De motor master is verantwoordelijk voor het aansturen van beide motoren.
Motor X	Motor X regelt de communicatie met de motor voor de X-as.
PID X	PID X creëert de control loop met feedback voor motor X.
Motor Z	Motor Z regelt de communicatie met de motor voor de Z-as.
PID Z	PID Z creëert de control loop met feedback voor motor Z.

5.3.2 White box Coin color separator

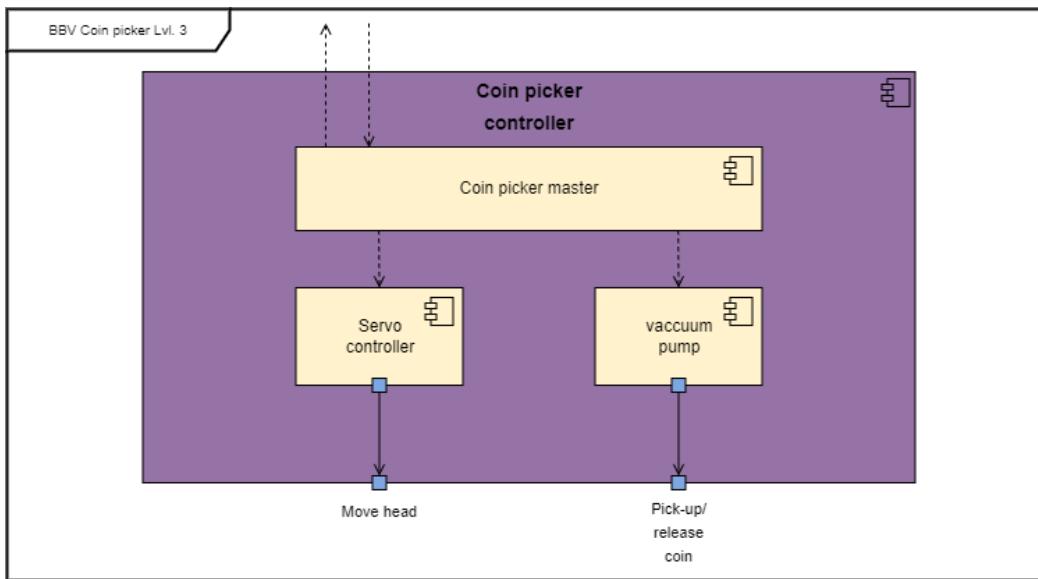


Figuur 11 BBV Coin color separator level 3

Tabel 12 Beschrijving BBV Coin color separator 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.
Proxy sensor	Krijgt data of er een fiche aanwezig is.
Solenoid	Is verantwoordelijk voor het activeren van de flipper.

5.3.3 White box Coin picker



Figuur 12 BBV Coin picker level 3

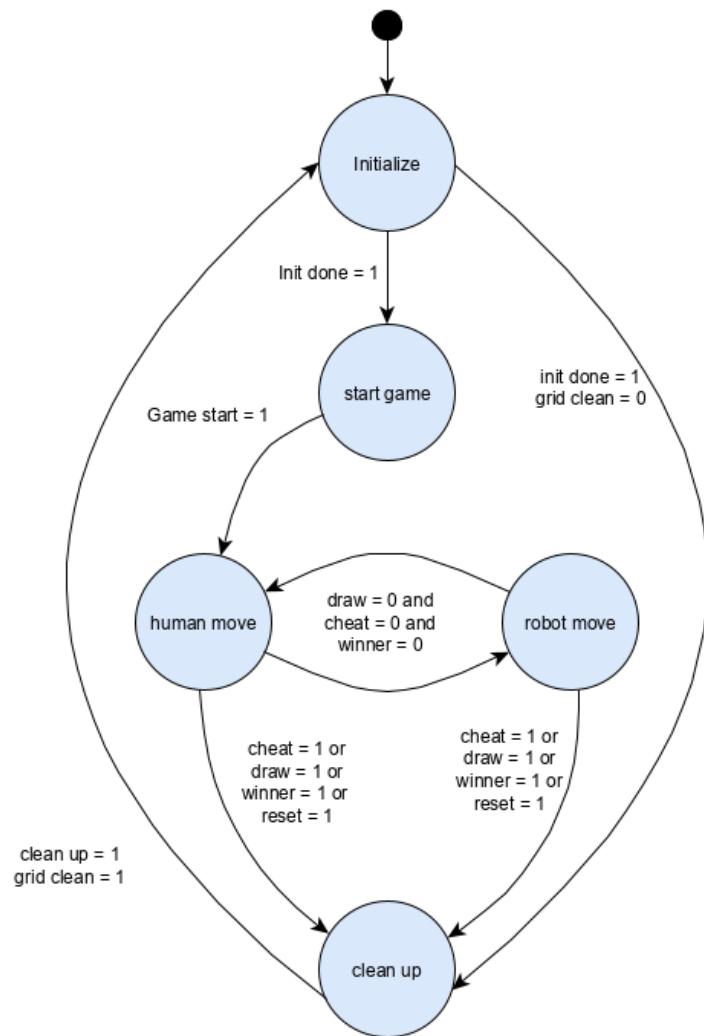
Tabel 13 Beschrijving BBV Coin picker level 3

Black box	Beschrijving
Coin picker master	De Coin picker master is verantwoordelijk voor het aansturen van de servo en de vaccuum pomp van de vaccuumgripper.
Servo controller	Regelt de aansturing van de servo motor.
Vaccuum pump	Regelt de aansturing van de vaccuum gripper om de fiches op te pakken.

6 Runtime view

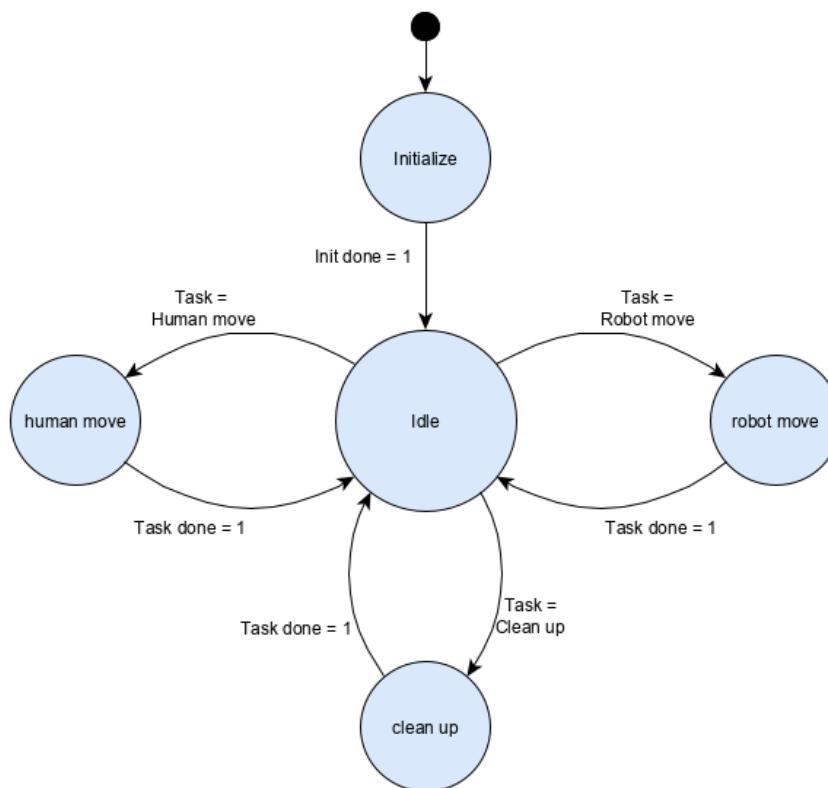
De Runtime view beschrijft het concrete gedrag en de interacties van de building blocks van het systeem. Dit wordt gedaan door middel van een highlevel statemachine (Figuur ...) en scenario's die worden weer gegeven in de subhoofdstukken.

6.1 High level overzicht



Figuur 13 State machine Cortex-M7

De Statemachine in Figuur 13 geeft aan hoe de 4-op-1-rij op het hoogste niveau handelt. Deze Statemachine draait op de Cortex-M7 binnen de game controller en is daarmee de main flow van de 4-op-1-rij. 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 als eerste zijn/haar fiche in het 4-op-1-rij spel te doen en bevind het systeem zich in de “human move” state. Elke beurt controleert de robot of er een winnaar, vals gespeeld of gelijk spel is. Als dit niet het geval is gaat de beurt naar de robot en gaat het systeem naar de state “robot move”. In deze state voert de robot de berekende zet uit. 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.

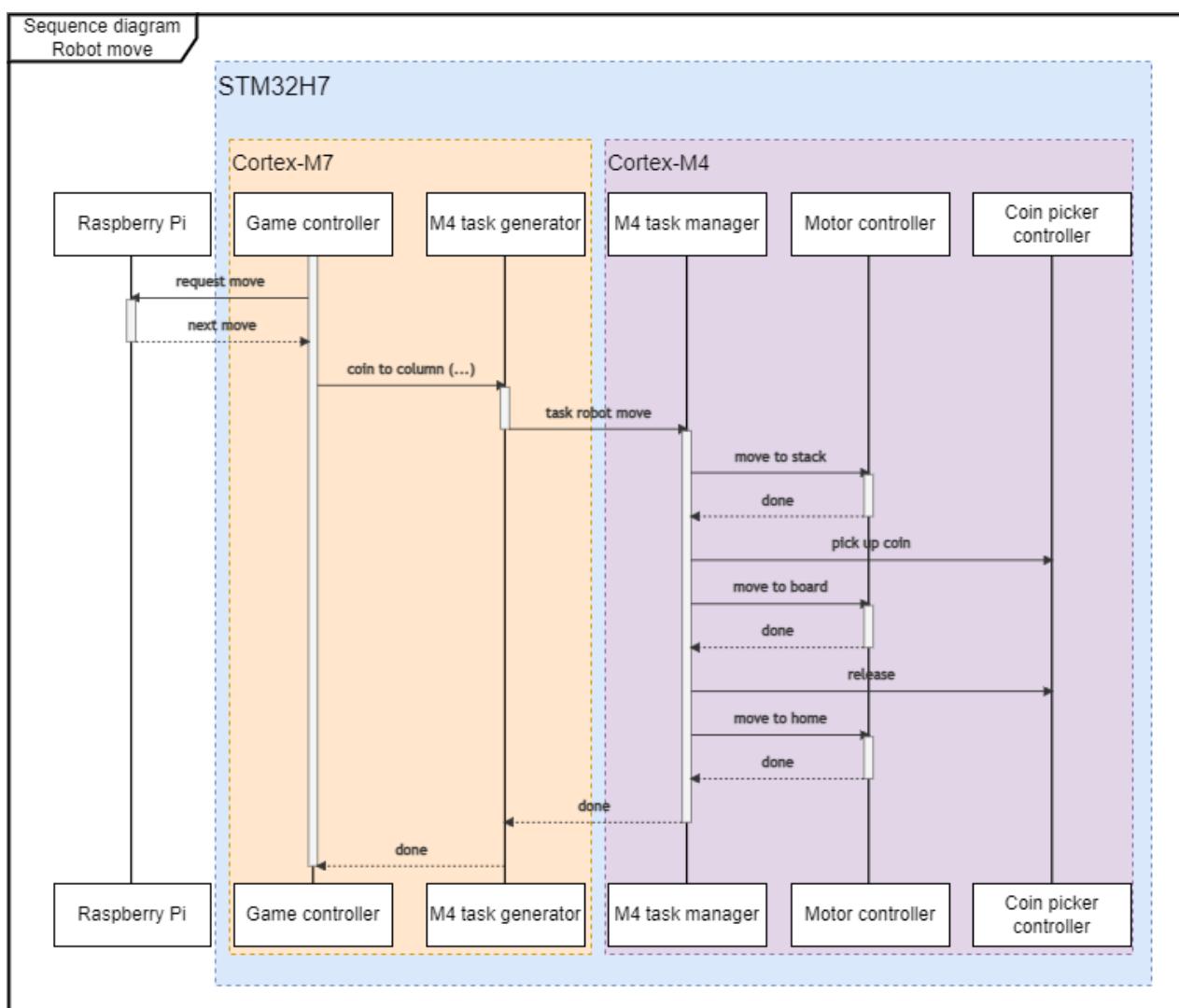


Figuur 14 State machine Cortex-M4

Zoals te zien in Figuur 9 bevindt zich op de Cortex-M4 ook een statemachine. Deze state machine is weergegeven in Figuur 14 en draait op de module Task manager. 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.

In de volgende subhoofdstukken worden de states "robotmove", "human move" en "clean-up" weergegeven in een sequence diagram. Een sequence diagram geeft aan hoe de verschillende building blocks zich tot elkaar verhouden door de tijd en wat voor signaal ze sturen.

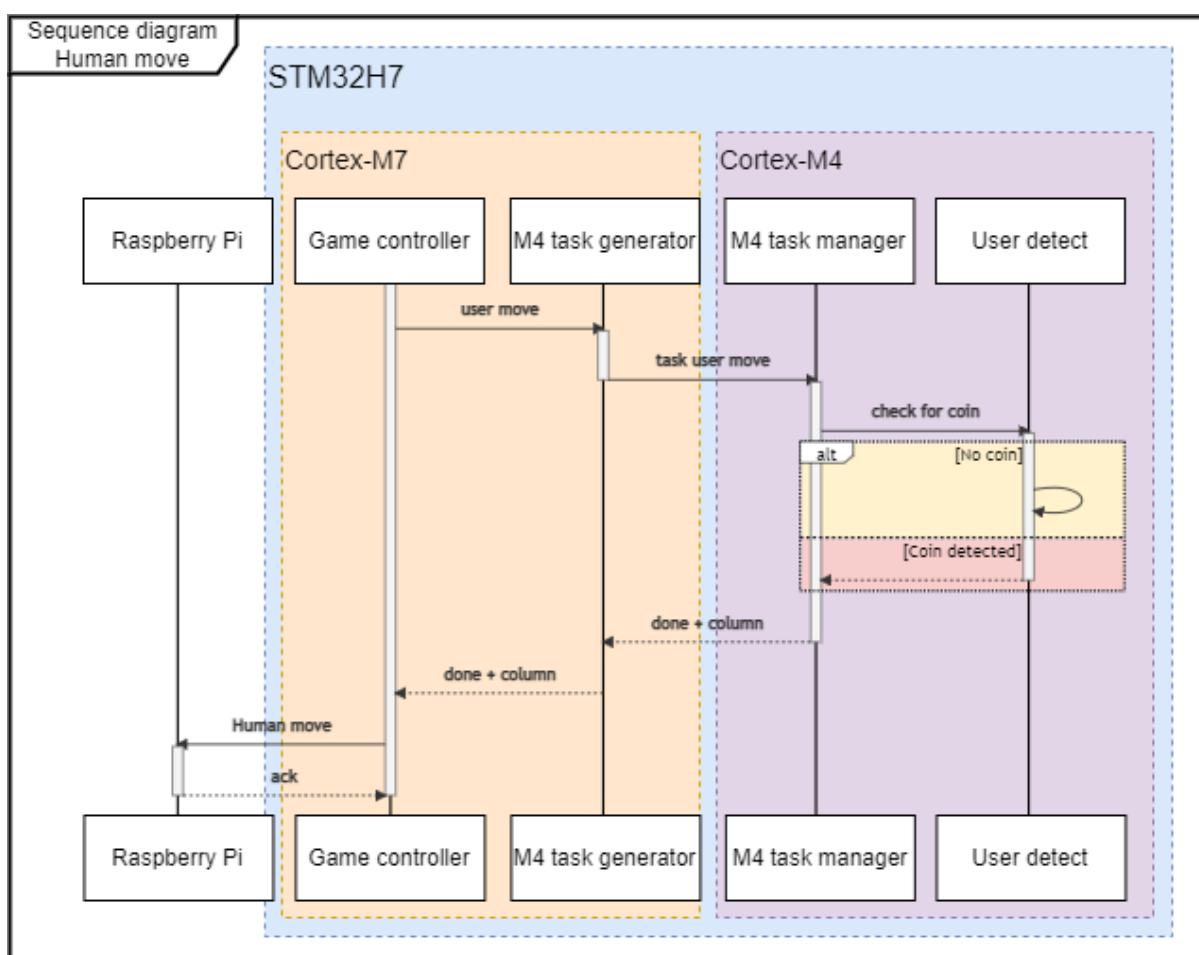
6.2 Robot move



Figuur 15 Sequence diagram Robot move

De "Robot move" in Figuur 15 begint met het opvragen van de volgende zet aan de Raspberry Pi door de Game controller. De Raspberry Pi geeft de volgende zet terug waarna de Game controller aan de M4 task generator doorgeeft wat voor 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 in het bord te doen 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. Tot slot moet de robot weer naar zijn "home" stand. Als de zet is uitgevoerd geeft de M4 task manager door aan de M4 task generator die op zijn beurt weer doorgeeft aan de Game controller dat de beurt voltooid is. De Game controller gaat zo naar de volgende state.

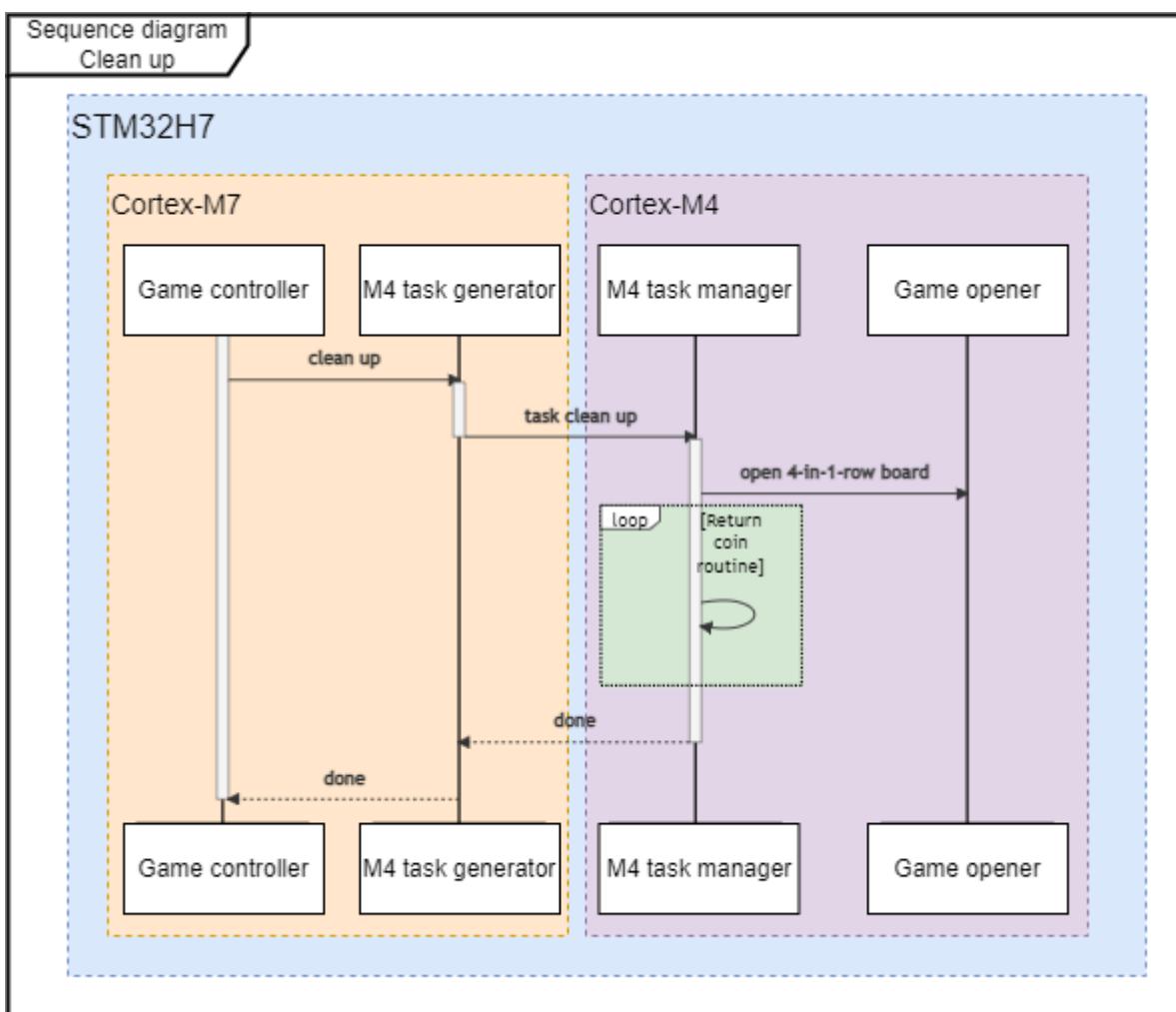
6.3 Human move



Figuur 16 Sequence diagram Human move

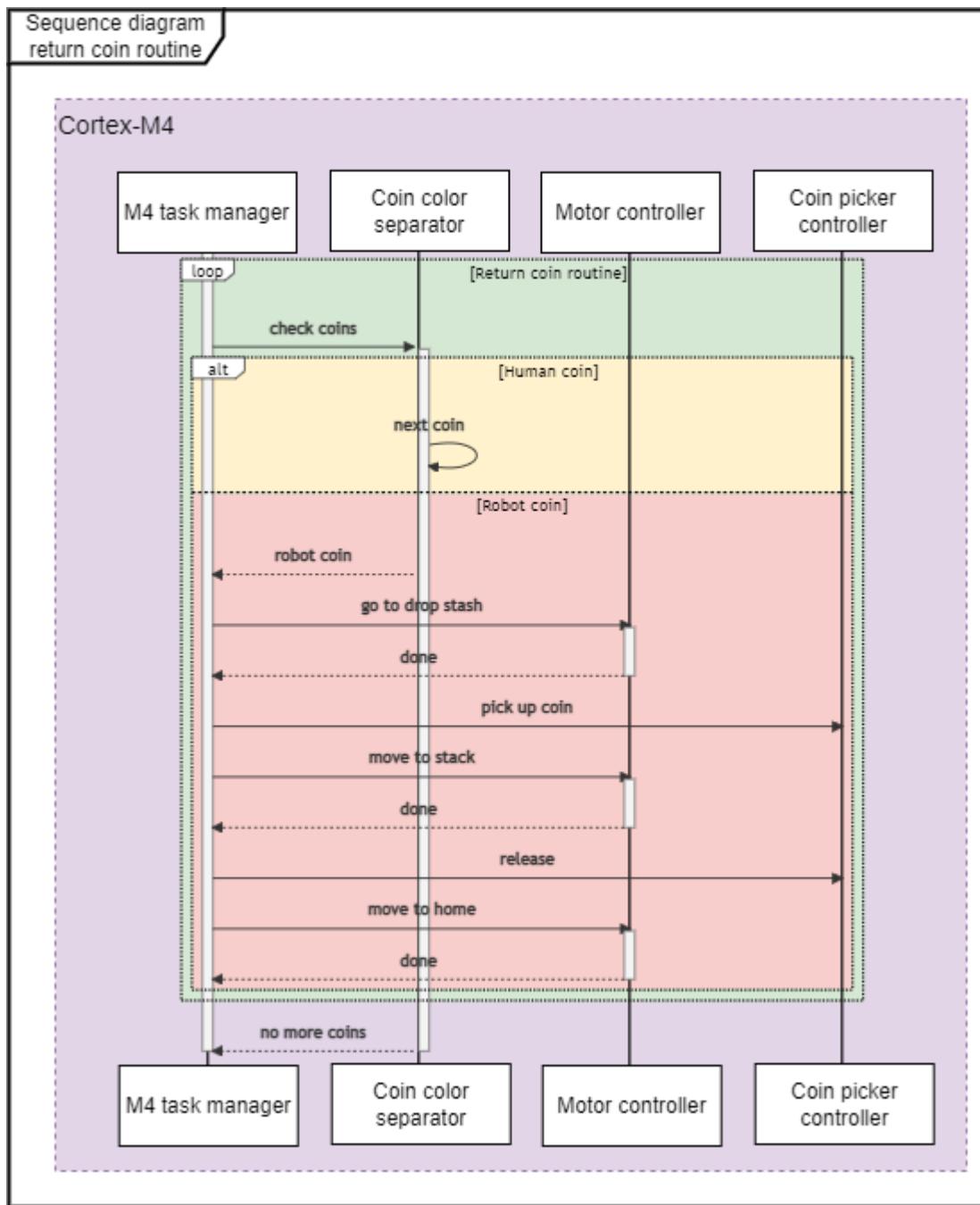
De "Human move" in Figuur 16 begint met het doorgeven dat de speler aan zet is aan de M4 task generator. De M4 task generator genereert de opdracht en stuurt deze naar de M4 task manager op de Cortex-M4. De M4 task manager op zijn beurt geeft de instructie aan de User detect dat er een fiche verwacht wordt. Als er geen fiche wordt gespeeld door de speler blijft het systeem wachten op een zet. Zodra er een fiche gespeeld wordt door de speler wordt dit terug gekoppeld aan de M4 task manager. Deze geeft door aan de M4 task generator dat er een zet gedaan is en in welke kolom het fiche is gestopt. De M4 task generator geeft dit door aan de Game controller waarna deze de zet doorgeeft aan de Raspberry Pi. De Raspberry Pi bevestigt het krijgen van de zet en stuurt terug of het spel klaar is of verder kan.

6.4 Clean up



Figuur 17 Sequence diagram Clean-up

De “Clean up” in Figuur 17 begint met het doorgeven dat het spel afgelopen is en de opruim routine kan beginnen aan de M4 task generator. De M4 task generator genereert de opdracht en stuurt deze naar de M4 task manager op de Cortex-M4. De M4 task manager opent het 4-op-1-rij bord en start de “Return coin routine” deze loop wordt uitgevoerd tot dat alle fiches opgeruimd zijn. Zodra dat gebeurt is geeft de M4 task manager door aan de M4 task generator dat het spel opgeruimd is. De M4 task generator geeft dit door aan de Game controller waarna het spel opnieuw kan beginnen.



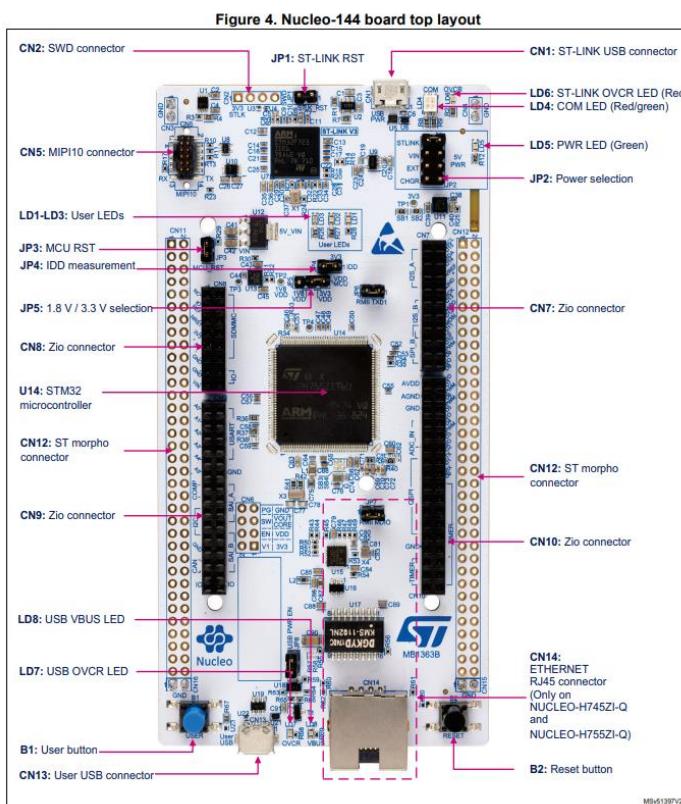
Figuur 18 Sequence diagram return coin routine

De "Return coin routine" in Figuur 18 is onderdeel van de "Clean up" uit Figuur 17 en beschrijft hoe de M4 task manager er voor zorgt dat alle fiches worden opgeruimd. Eerst stuurt de M4 task manager naar de Coin color separator dat er fiches verwacht worden. Zodra er een fiche gedetecteerd wordt gaat de Coin color separator kijken welke kleur dit fiche heeft. Als het fiche van de speler is wordt deze door een flipper naar de bak van de speler geschoten. Als het fiche van de robot is wordt de M4 task manager geïnformeerd dat het gaat om een fiche van de robot. De M4 task manager stuurt de Motor controller aan om naar het fiche toe te gaan. Vervolgens wordt het fiche op gepakt en beweegt de robot naar zijn eigen fiches opslag. Hier wordt het fiche weer losgelaten en beweegt de robot naar zijn "home" stand. Deze loop wordt uitgevoerd tot alle fiches zijn opgeruimd.

7 Deployment View

7.1 NUCLEO-H755ZI-Q

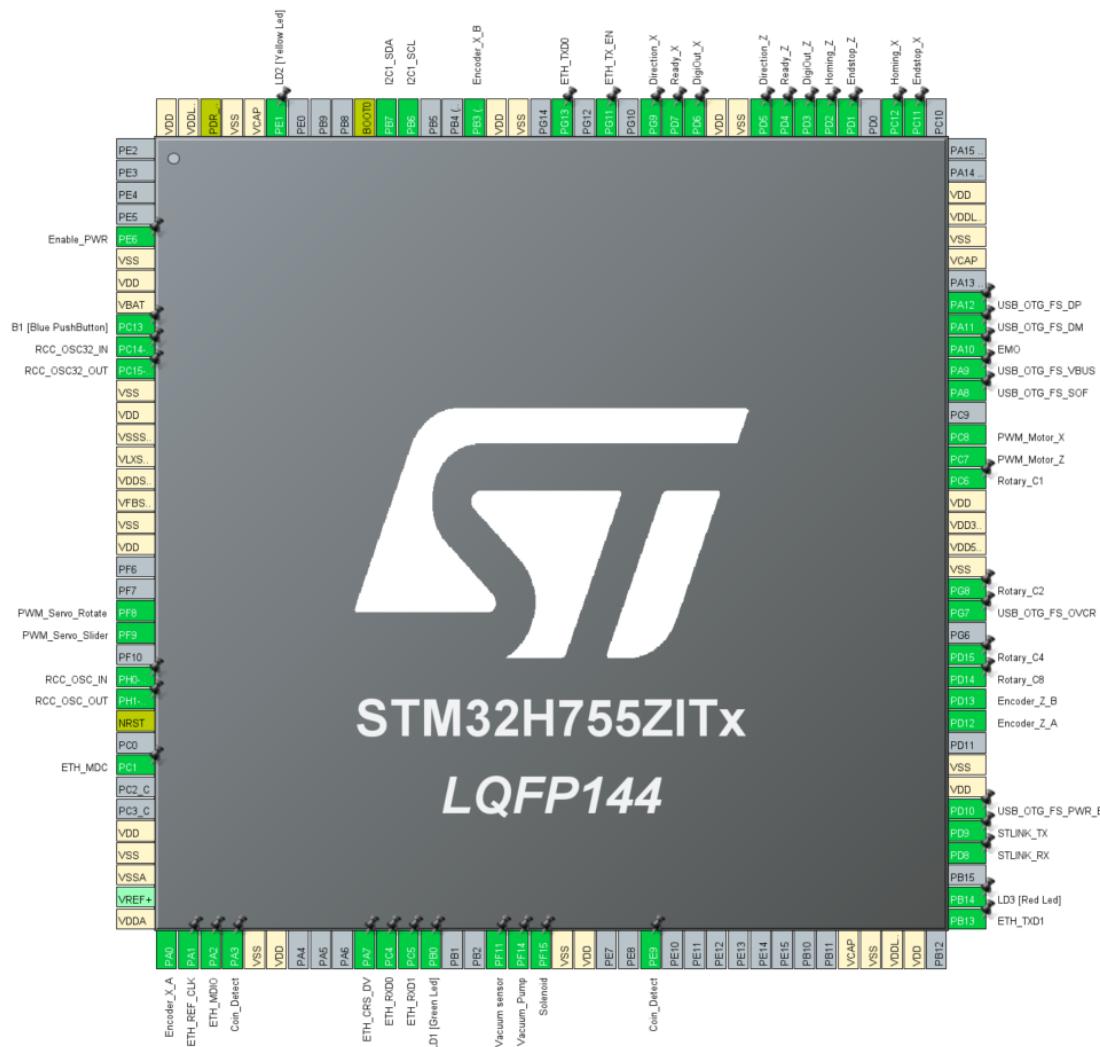
Om de software architectuur te implementeren is een Nucleo-H755ZI-Q aanwezig. Dit is een development board met de STM32H7 chip aanboord. Verder heeft dit bord alle benodigde pin-outs en hardware om het implementeren van de software blokken te faciliteren. Het plan is om op ten duur over te gaan naar een door ALLEN ontworpen PCB voor de STM32H7 chip met alle benodigde connecties. Dit valt echter buiten de scope van dit project en daarom wordt er gewerkt met de Nucleo-H755ZI-Q (Figuur 19).



Figuur 19 NUCLEO-H755ZI-Q

7.2 Pin-out NUCLEO-H755ZI-Q

Om alle functies van de 4-op-1-rij te realiseren is de pin-out in Figuur 20 samengesteld. In deze pin-out is rekening gehouden met de functie van elke pin en wat de pin moet faciliteren. In Tabel 14 staat de functie die uitgevoerd wordt door een pin, wat voor signaal er op de pin staat en aan welke pin de functie verbonden is. Zoals te zien zijn er ook pinnen vastgesteld voor ethernet. Ethernet is een functie die in de toekomst misschien gebruikt gaat worden maar buiten de scope van dit project valt. Maar het is wel van groot belang om de functie wel toe te wijzen aan de pinnen om te voor komen dat die pinnen gebruikt worden voor andere doeleinden.



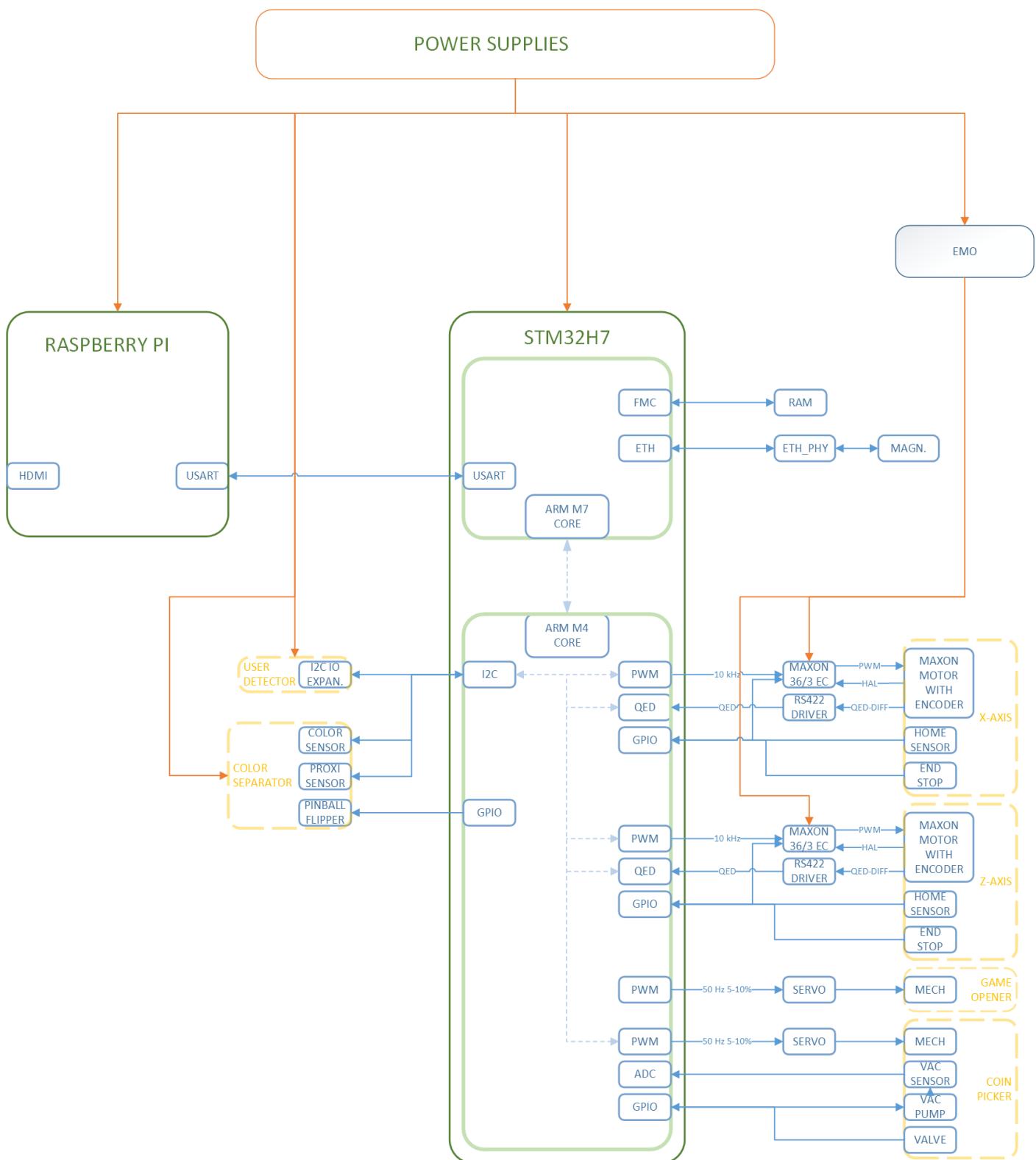
Figuur 20 pin-out STM32H755ZITx

Tabel 14 pin-out tabel

Function	STM Function	STM Pin	Connector
LD1 (Green)	GPIO	PB0	
LD2 (Yellow)	GPIO	PE1	
LD3 (Red)	GPIO	PB14	
Push button	GPIO	PC13	
Encoder X	A	PA0	
	B	PB3	
Encoder Z	A	PD12	
	B	PD13	
PWM X	PWM Timer	PC8	
PWM Z	PWM Timer	PC7	

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

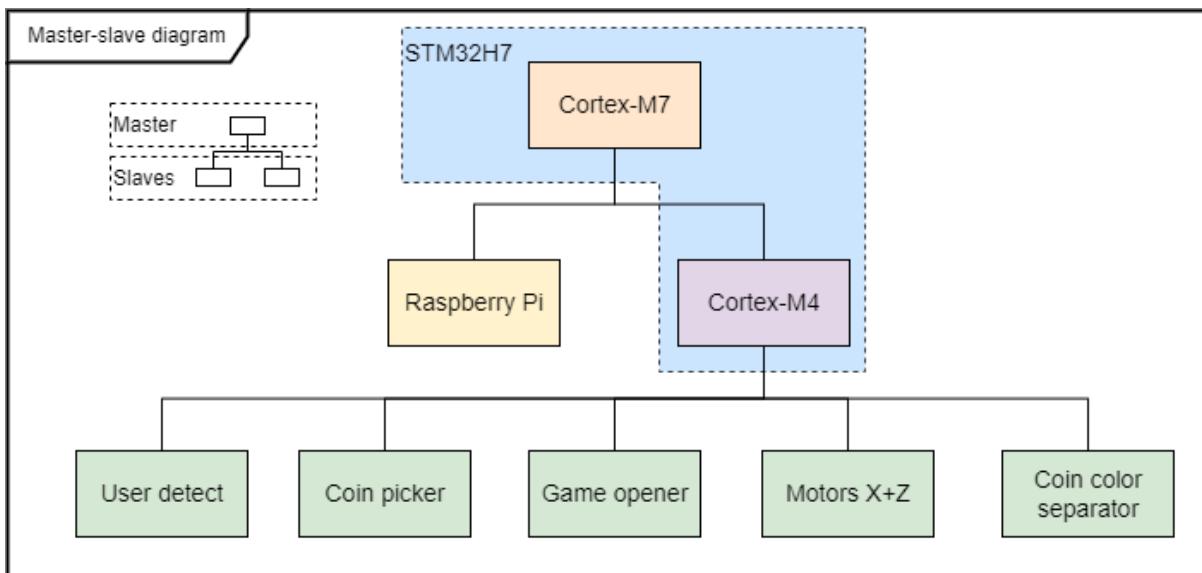
7.3 Hardware lay-out



Figuur 21 Hardware lay-out 4-op-1-rij met STM32H7

7.4 Master-Minion verhouding

Om aan te geven hoe de processoren zich verhouden in een Master-Minion verhouding is Figuur 22 opgesteld. Hierin is te zien dat de Cortex-M7 de top-master van het systeem is. Op de Cortex-M7 draait de game flow en die bepaalt dus het volledige spel verloop. De Raspberry Pi en Cortex-M4 zijn een minion van de Cortex-M7 en voeren taken uit als de Cortex-M7 dat vraagt. De Cortex-M4 is ook een master voor de hardware. Alle hardware systemen van de 4-op-1-rij zijn een minion van de Cortex-M4 en voeren taken uit als de Cortex-M4 dat vraagt.



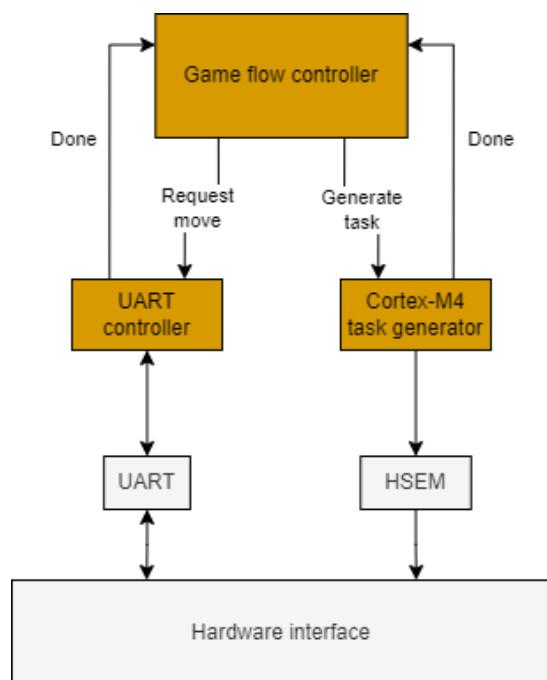
Figuur 22 Master-minion verhouding

8 Modulaire software implementatie

(Hoofdstuk 5) geeft weer welke software blokken aanwezig moeten zijn om een volledig werkende 4-op-1-rij robot te implementeren. Hier is niet weergegeven hoe deze van het top-level tot de hardware samen hangen. Dat wordt in dit hoofdstuk verder uitgelegd aan de hand van diagrammen.

8.1 Cortex-M7 core

In Figuur 8 worden alle software blokken van de Cortex-M7 core weergegeven. Om weer te geven hoe de blokken zich verhouden tot de hardware is het diagram in Figuur 23 opgesteld.



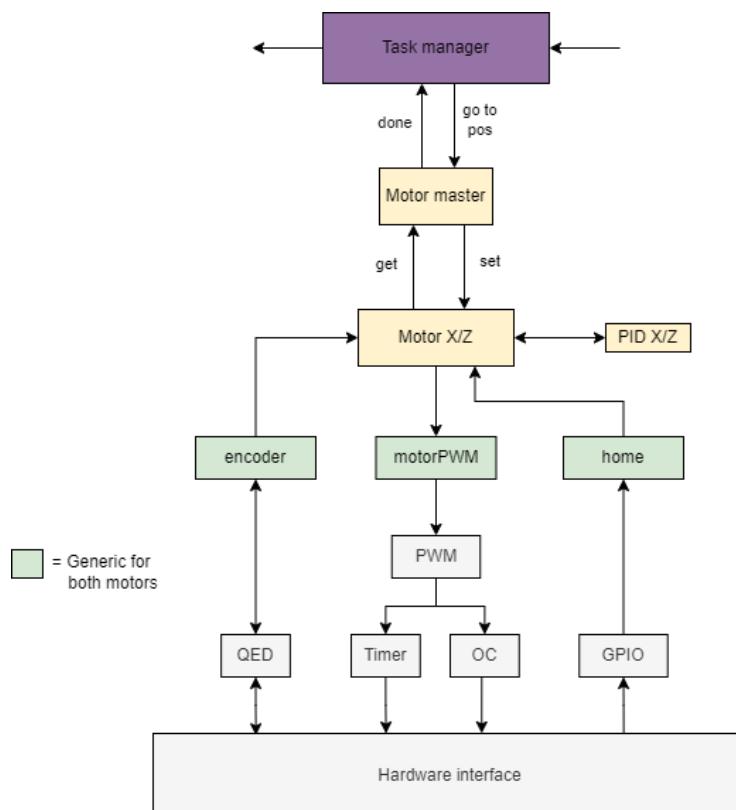
Figuur 23 IBD game controller

8.2 Cortex-M4 core

In Figuur 9 worden alle software blokken van de Cortex-M4 core weergegeven. Om weer te geven hoe de blokken zich verhouden tot de hardware zijn alle diagrammen in de subhoofdstukken opgesteld.

8.2.1 Motor controller

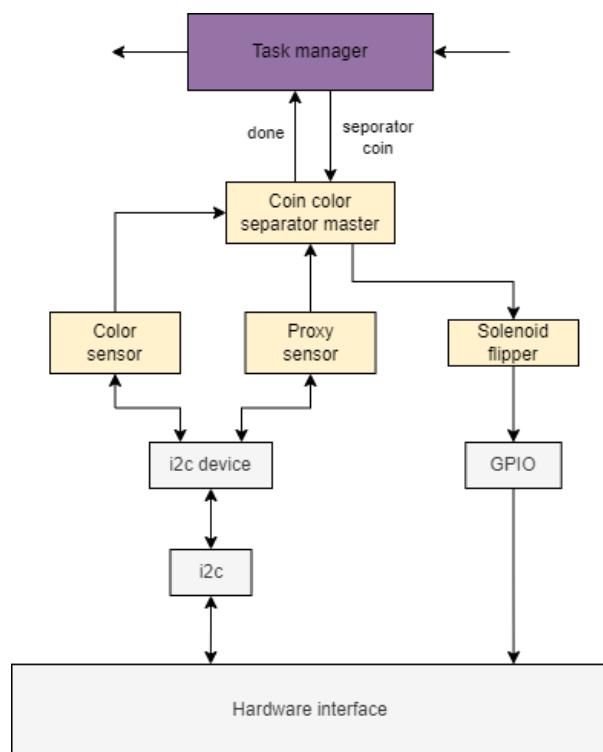
In Figuur 24 geeft het diagram van de motoren weer. De blokken voor Motor X zijn hetzelfde als voor Motor Z. Deze blokken zijn in weergegeven in het groen.



Figuur 24 IBD Motor controller

8.2.2 Coin color separator

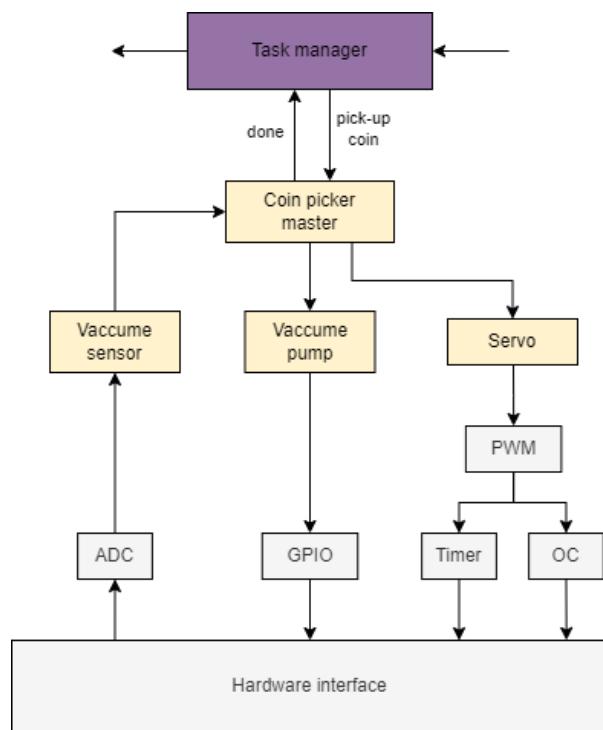
Figuur 25 geeft het diagram weer voor de Coin color separator. Het i2c device blok is een generiek blok dat alle informatie voor een i2c sensor afhandelt.



Figuur 25 IBD Coin color separator

8.2.3 Coin picker

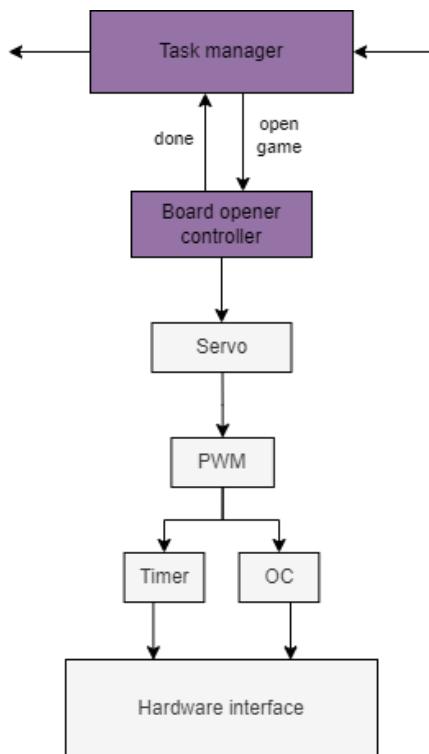
Figuur 26 geeft het diagram weer voor de Coin picker.



Figuur 26 IBD Coin picker

8.2.4 Board opener

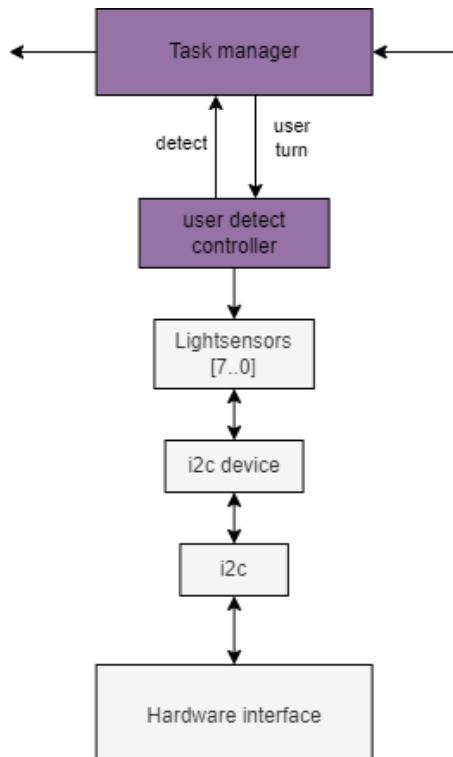
Figuur 27 geeft het diagram weer voor de Board opener.



Figuur 27 IBD Board opener

8.2.5 User detect

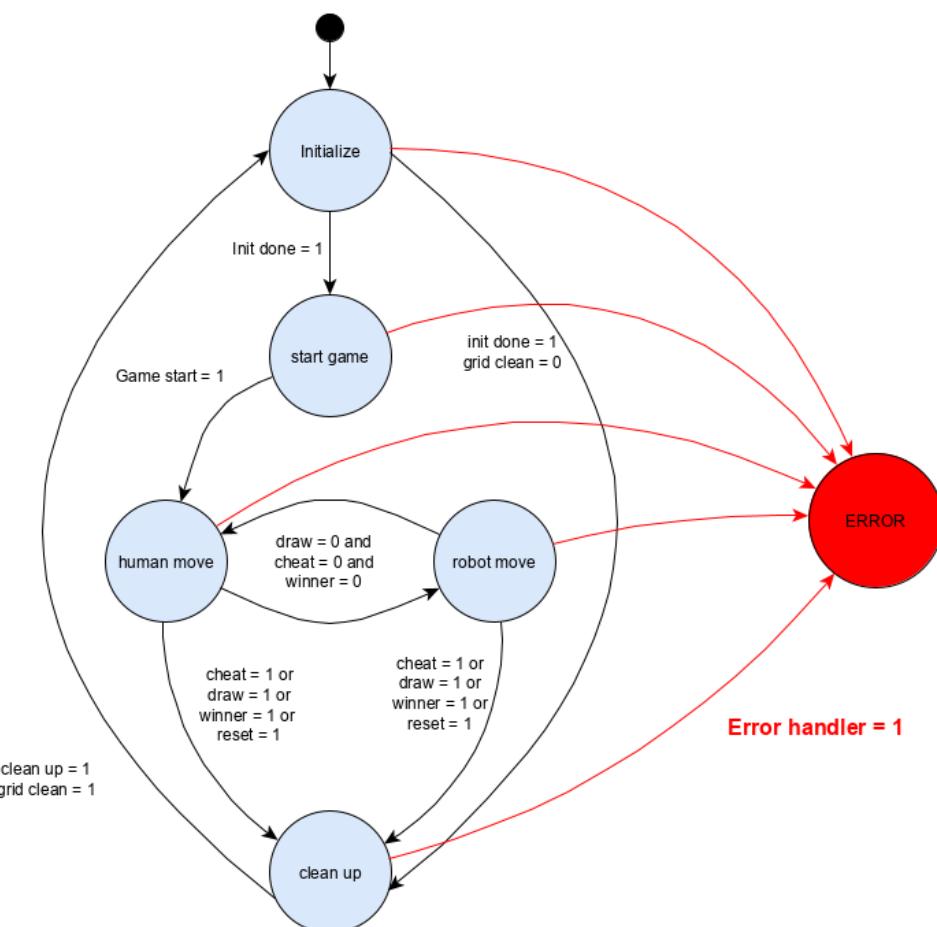
Figuur 28 geeft het diagram weer voor de User detect.



Figuur 28 IBD User detect

9 Foutafhandeling

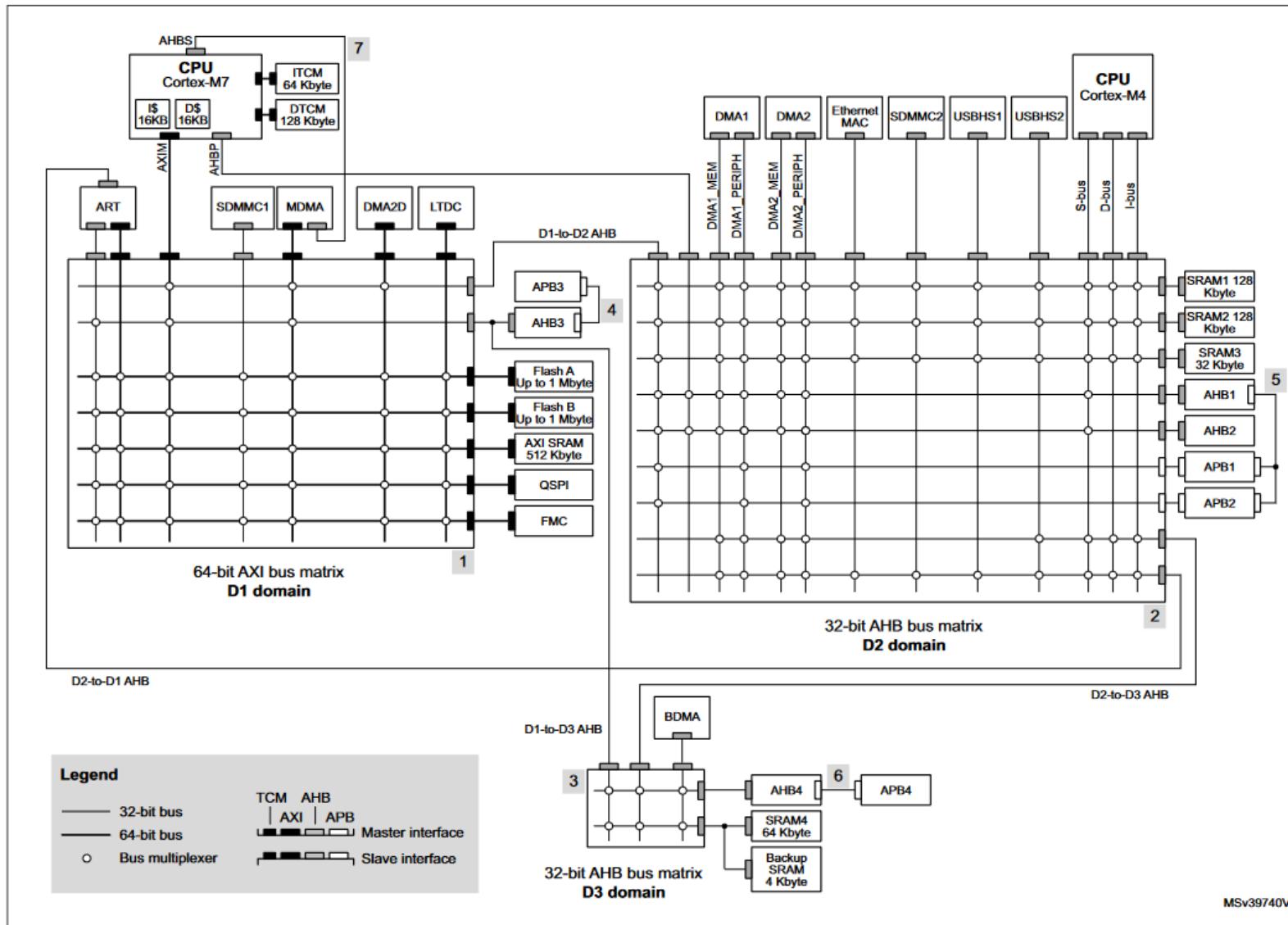
Volgens de architectuur die in voor gaande hoofdstukken is beschreven kan het systeem vloeiend draaien. Toch bestaat er een kans dat er een fout optreedt. Als de motoren niet goed de “homing” fase uitvoeren, een van de sensoren reageert niet meer of de motoren falen als een fiche naar het bord wordt gebracht. Zodra het systeem een error melding geeft moet deze afgehandeld worden. Dit betekent voor de veiligheid hardware uitzetten en een indicatie geven aan de Operator dat het systeem een fout heeft gedetecteerd. Figuur 29 laat zien hoe een error binnen de hoofd states van de Game flow controller wordt afgehandeld.



Figuur 29 State machine met error handler

In de “ERROR” state worden alle motoren en andere hardware componenten uitgezet binnen de twee cores van de STM32H7 en blijft het systeem in deze state tot deze gereset wordt.

IV. Memory en bus architectuur STM32H7 dual-core



V. Implementatie methodes voor een software loop

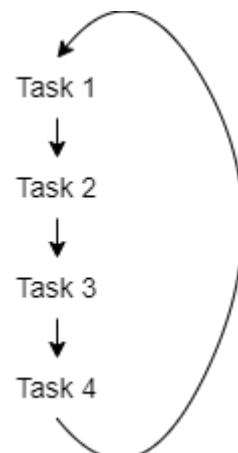
Het is belangrijk om een goede methode te kiezen om het systeem niet te complex te maken. Vier mogelijke opties zijn onderzocht:

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 represeneert 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 vervaart. Dit laatste kan beperkt worden door de tijdgebonden taken vaker terug te laten komen in de loop.



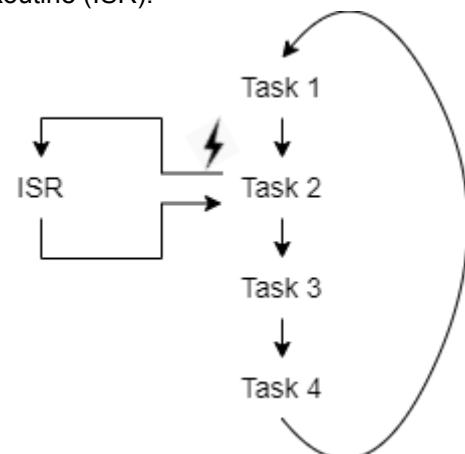
Figuur 30 Round robin

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.

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.



Figuur 31 Round robin with interrupt

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 aanroeft, 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.

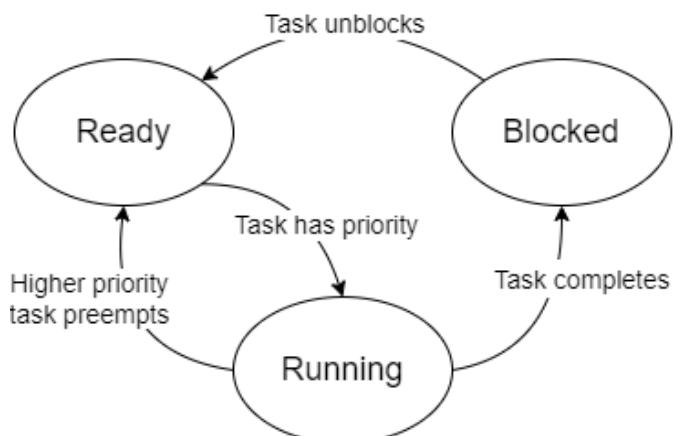
2. Ready

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.

3. Blocked

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

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].



Figuur 32 RTOS scheduler

VI. 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 bevindt.

2. Op-code

Afkorting voor de instructie die uitgevoerd moet worden.

3. 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, 0x4(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
```

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)).
```

Hieruit 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.

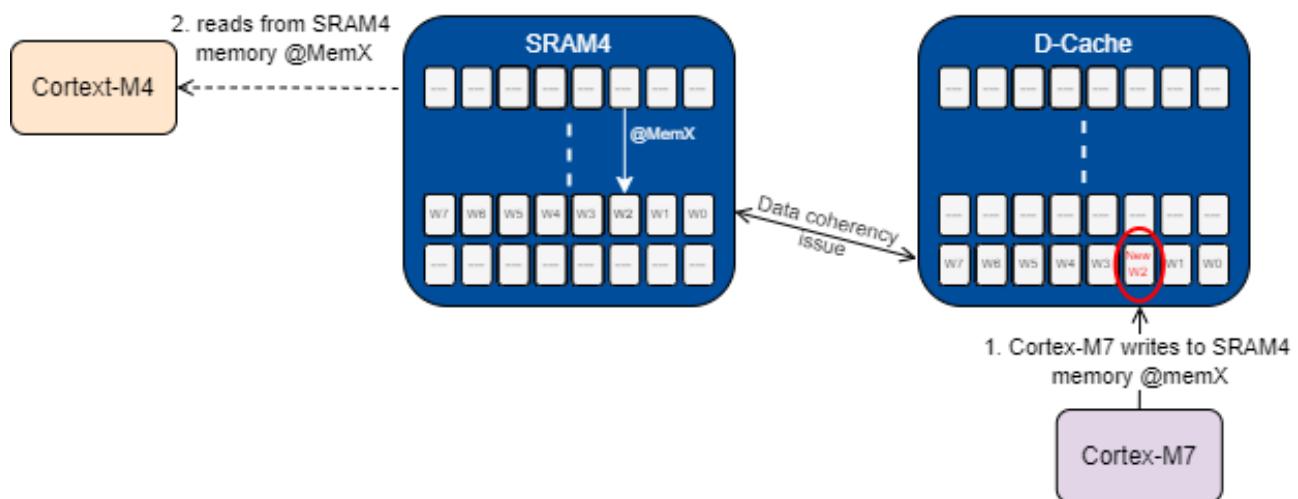
De oplossing voor het voorkomen van false data is simpel. Het deel van de code dat gebruik maakt van data die verworven wordt door een interrupt wordt kritische code genoemd. Het blok kritische code moet beschermt worden voor verhindering. Het uitschakelen van de interrupt voor het kritische blok en 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.

VII. 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 probleem [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.
2. *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 ingestellen. Verder scheelt het veel regels code die anders nodig zijn voor het implementeren van “Cache Clean” en “Cache Invalidate” functies.



VIII. 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);
    }
}
```



IX. Code i2c module

i2c dev.c

```
/*
 * i2c_dev.c
 *
 * Created on: May 2, 2022
 * Author: Pascal
 */

#include "i2c_dev.h"

HAL_StatusTypeDef i2c_CheckDev(I2C_HandleTypeDef* bus, uint8_t DevAddress) {
    HAL_StatusTypeDef retFunc;
    uint8_t write_addr = DevAddress << 1;
    retFunc = HAL_I2C_IsDeviceReady(bus, write_addr, 1, TIME_OUT);
    return retFunc;
}

HAL_StatusTypeDef i2c_Transmit(I2C_HandleTypeDef* bus, uint8_t DevAddress,
                               uint8_t MemAddress, uint8_t MemAddSize, uint8_t* pData, uint8_t
                               pData_size) {
    HAL_StatusTypeDef retFunc;
    uint8_t write_addr = DevAddress << 1;
    retFunc = HAL_I2C_Mem_Write(bus, write_addr, MemAddress,
                                MemAddSize, pData, pData_size, TIME_OUT);
    return retFunc;
}

HAL_StatusTypeDef i2c_Receive(I2C_HandleTypeDef* bus, uint8_t DevAddress,
                             uint8_t MemAddress, uint8_t MemAddSize, uint8_t* pData, uint8_t
                             pData_size) {
    HAL_StatusTypeDef retFunc;
    uint8_t read_addr = (DevAddress << 1) | 0x01;
    retFunc = HAL_I2C_Mem_Read(bus, read_addr, MemAddress, MemAddSize,
                               pData, pData_size, TIME_OUT);
    return retFunc;
}
```



Proxi sensor VCNL4010.c

```
/*
 * VCNL4010.c
 *
 * Created on: May 2, 2022
 *      Author: Pascal
 */

#include "VCNL4010.h"

void VCNL4010_Init(const VCNL4010* const self) {
    uint8_t led_ma = 0x0A;
    uint8_t com_en = 0x03;
    HAL_StatusTypeDef retFunc;
    retFunc = i2c_CheckDev(self->bus, self->base_addr);

    if (retFunc == HAL_OK) {
        i2c_Transmit(self->bus, self->base_addr, VCNL4010_LED_REG,
1, &led_ma, 1);
        i2c_Transmit(self->bus, self->base_addr, VCNL4010_COM_REG,
1, &com_en, 1);
        HAL_Delay(1);
    } else {
    }
}

uint16_t VCNL4010_ReceiveProxi(const VCNL4010* const self) {
    uint8_t buf[2];
    uint16_t val = 0;
    HAL_StatusTypeDef retFunc;

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

VCNL4010 VCNL4010_Create(uint8_t addr, I2C_HandleTypeDef* inBus) {
    VCNL4010 create = { addr, inBus};
    return create;
}
```



X. Code UART module

UART_controller.c

```
/*
 * UART_controller.c
 *
 * Created on: Apr 29, 2022
 * Author: Pascal
 */

#include "UART_controller.h"

uint8_t* rxdata;

void Init_UART_controller(UART_HandleTypeDef* const RPIbus, uint8_t* data,
                           uint8_t* substate) {
    rxdata = data;
    HAL_UART_Receive_IT(RPIbus, rxdata, 3);
    srand(10);
}

void RPI_Request_Move(UART_HandleTypeDef* const RPIbus, uint8_t
insertColumn) {
    int random = rand() % 7 + 1;
    uint8_t buf[24];
    sprintf((char*) buf, "UART com. Value : %i\r\n", random);
    UART_WriteString(&huart3, (char*)buf);
    UART_WriteValue(RPIbus, random);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    uint8_t buf[28];
    sprintf((char*) buf, "Received value : %.s\r\n", 2, rxdata);
    UART_WriteString(&huart3, (char*)buf);
    HAL_UART_Receive_IT(huart, rxdata, 3);
}

void UART_WriteString(UART_HandleTypeDef* const bus, char* buf) {
    HAL_UART_Transmit(bus, (uint8_t*)buf, strlen(buf), HAL_MAX_DELAY);
}

void UART_WriteValue(UART_HandleTypeDef* const bus, int value) {
    uint8_t buf[12];
    sprintf((char*) buf, "%i\r\n", value);
    HAL_UART_Transmit(bus, (uint8_t*)buf, strlen((const char*)buf),
HAL_MAX_DELAY);
}
```