ONDERZOEK

VAN REAL-TIME OPERATING SYSTEMS

afstudeerstageproject

PROCENTEC

Auteur: Maria Brodskaya

Datum: 08-09-2014

Stagebedrijf: PROCENTEC

Bedrijfsbegeleider: Edward Dumay

Onderwijsinstelling:

Haagse Hogeschool, Delft

Opleiding: Technische Informatica

Samenvatting

Dit verslag beschrijft mijn onderzoek naar bestaande Real-Time Operating Systems.

Dit onderzoek heb ik uitgevoerd voor het bedrijf PROCENTEC, dat is gespecialiseerd in industriële technologieën. Een van de productlijnen die het bedrijf heeft ontwikkeld is COMbricks. Dit is een device dat communicatie tussen twee verschillende protocollen mogelijk maakt. Om dit product verder te ontwikkelen wilde PROCENTEC een Real-Time Operating System in de software van COMbricks integreren. Om het meeste geschikte Real-Time Operating System te kunnen kiezen moest onderzoek worden gedaan.

Mijn onderzoek bestond uit twee delen: een theoretisch en een praktisch onderzoek.

Tijdens het theoretische onderzoek stelde ik de eisen aan het systeem vast. Deze eisen waren gebaseerd op de eigenschappen van de gebruikte hardware in COMbricks en op algemene handige kernmerken van de Real-Time Operating Systems, zoals technische ondersteuning, populariteit en prijs. Aan het eind van het theoretische onderzoek kwamen drie Real-Time Operating Systems naar voren: ThreadX, MicroC/OS-II en FreeRTOS/OpenRTOS. Deze systemen voldeden het best aan de vastgestelde eisen.

Voor het praktische onderzoek maakte ik een demoapplicatie om de top-drie-systemen te testen. ThreadX viel af voor het onderzoek, want het bleek om verschillende redenen onmogelijk om een trialversie van dit systeem te krijgen. MicroC/OS-II en FreeRTOS werden getest. Voor deze twee systemen maakte ik een wrapper om eenvoudig te kunnen overschakelen van het ene systeem naar het andere. Bij het ontwikkelen en testen van de wrapper kwamen voordelen en nadelen van MicroC/OS-II en FreeRTOS naar voren.

Aan het eind van dit onderzoek werd FreeRTOS/OpenRTOS als het meeste geschikte systeem gekozen. Bij deze keuze werd rekening gehouden met de aanwezige nadelen en de prijsfactor.

Dit verslag bestaat uit dertien hoofdstukken. In de eerste vier hoofdstukken zijn het bedrijf, het probleem, het doel van opdracht en de keuze van ontwikkelmethode beschreven. Het vijfde hoofdstuk gaat over het theoretische onderzoek. De vijf hoofdstukken daarna zijn aan de ontwikkeling van de demoapplicatie en de wrapper gewijd. In hoofdstuk elf wordt de uiteindelijke keuze voor het systeem toegelicht. De laatste twee hoofdstukken bevatten mijn conclusie en de evaluatie van het project.

Inhoudsopgave

1. Inleiding

Ter afsluiting van mijn opleiding Technische Informatica aan de Haagse Hogeschool volgde ik een afstudeerstage bij het bedrijf PROCENTEC.

PROCENTEC ontwikkelt en levert producten en diensten om productieprocessen bij klanten in de industrie te optimaliseren. Het bedrijf is specialist in onder meer PROFIBUS-technologie. PROFIBUS (Proces Field Bus) is een standaard voor veldbuscommunicatie in de automatiseringstechniek en wordt toegepast om apparatuur met uiteenlopende functies, zoals sensoren, controllers en remote I/O, in één systeem te integreren. PROFIBUS werkt met twee communicatieprotocollen met verschillende snelheid en doorgiftetechniek, PA en DP. Voor een naadloze aansluiting tussen PA- en DP-netwerken voert PROCENTEC een speciale productlijn, genaamd COMbricks.

Aan de PA-kant werkte COMbricks tot dusver op de datalinklaag sequentieel. Dat kon bij uitbreiding van de aansluitmogelijkheden — waar het bedrijf mee bezig was — tot lange wachttijden leiden. In de industrie is de snelheid van de communicatie tussen de automatiseringsapparatuur cruciaal. Daarom wilde PROCENTEC de processen op de datalinklaag van de COMbricks PA link parallelliseren. Het tot dusver toegepaste besturingssysteem liet dat niet toe. Het bedrijf wilde overstappen op een bestaand Real-Time Operating System (RTOS).

Afstudeeropdracht

De vraag voor PROCENTEC was: welk van de bestaande RTOS'en is het meest geschikt voor integratie in de bestaande software van de COMbricks PA link?

Mijn afstudeeropdracht luidde dan ook: selecteer in een vergelijkend theoretisch onderzoek en een aansluitend praktijkonderzoek het RTOS dat beschikt over de beste functionaliteit die noodzakelijk is voor de COMbricks PA link.

Afstudeerverslag

In dit verslag geef ik het werkproces tijdens mijn afstudeerstage weer. Ik beschrijf hoe ik mijn doelen heb bereikt, welke problemen er zijn voorgekomen en hoe ik die heb opgelost.

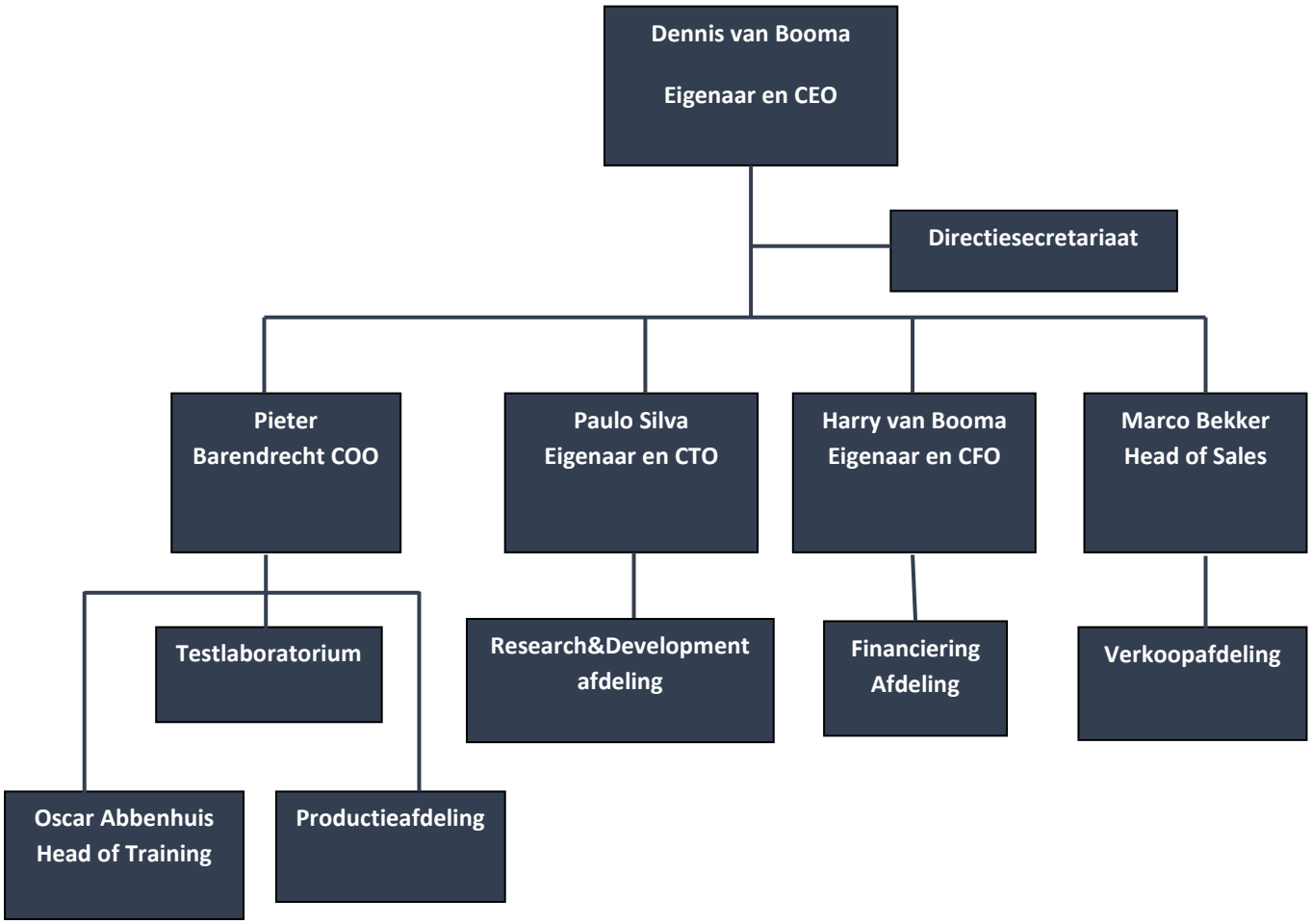
Na deze inleiding beschrijft hoofdstuk twee het bedrijf PROCENTEC, zijn doelen en organisatiestructuur. Hoofdstuk drie beschrijft de opdracht en uitdagingen die ik moest oppakken. Hoofdstuk vier gaat over ontwikkelmethode en planning. De daarop volgende zeven hoofdstukken beschrijven hoe ik het onderzoek heb uitgevoerd in meerdere iteraties, de uitkomsten van het onderzoek, de implementatiestappen en testresultaten. Hoofdstuk twaalf analyseert de werkzaamheden en beschrijft of ik het doel van het project heb bereikt en alle problemen heb opgelost. In hoofdstuk dertien wordt het werk aan het project beoordeeld: hier beschrijf ik welke competenties ik tijdens het afstudeerproject heb toegepast en welke nieuwe vaardigheden ik heb opgedaan.

2. Opdrachtgevende organisatie

Mijn afstudeerstageopdracht voerde ik uit voor het bedrijf PROCENTEC. PROCENTEC, opgericht in 1998, is specialist in industriële procesautomatisering met behulp van onder mee PROFIBUS-technologie. Het bedrijf verzorgt ook trainingen en ondersteuning van eindgebruikers. Het bedrijf oriënteert zich op de wereldmarkt. Er zijn officiële distributeurs in negen landen.

PROCENTEC heeft twee kantoren: een hoofdkantoor in Wateringen en een verkoopkantoor in Duitsland. Op het hoofdkantoor werken ruim 30 mensen. Er zijn vijf afdelingen: research & development, verkoop, financiën, productie en het testlaboratorium (zie organigram 1).

Organigram 1: Organisatiestructuur van PROCENTEC



De opdrachtgever voor mijn afstudeerproject was de afdeling research & development. Het hoofd van deze afdeling, Paulo Silva, was de bedrijfsmentor van dit project. Mijn begeleider was Edward Dumay, ingenieur van research & development.

De opdrachtnemer van het project was de Haagse Hogeschool. De projectbegeleider vanuit de school was de heer Smeets.

3. Opdrachtomschrijving

PROCENTEC maakt devices waarin het bedrijf de PROFIBUS-technologie toepast.

PROFIBUS (Proces Field Bus) is een standaardprotocol voor veldbuscommunicatie in de automatiseringstechniek. Er zijn twee varianten:

- PROFIBUS DP (Decentralized Peripher). Deze variant wordt gebruikt voor productieautomatisering. Het voornaamste kenmerk van deze variant is de hoge snelheid.
- PROFIBUS PA (Process Automation). Deze variant wordt gebruikt voor procesautomatisering. Hij voldoet aan alle speciale eisen voor procesautomatisering, zoals intrinsiek veilige doorgiftetechnieken, betrouwbare datadoorgifte en interoperabiliteit (standaardisatie van functies van het apparaat). Het voornaamste kenmerk van deze variant is betrouwbaarheid.

De PA- en DP-protocollen verschillen van elkaar op de fysieke laag, zie tabel 3.1.

Tabel 3.1: Verschillen op de fysieke laag tussen PROFIBUS DP en PROFIBUS PA

	PROFIBUS DP	**PROFIBUS PA**
Doorgiftesnelheid	tot 12 Mbits	31,25kBits
Doorgiftetechniek	RS485	IEC 61158-2

Koppeling van PA en DP via COMbricks

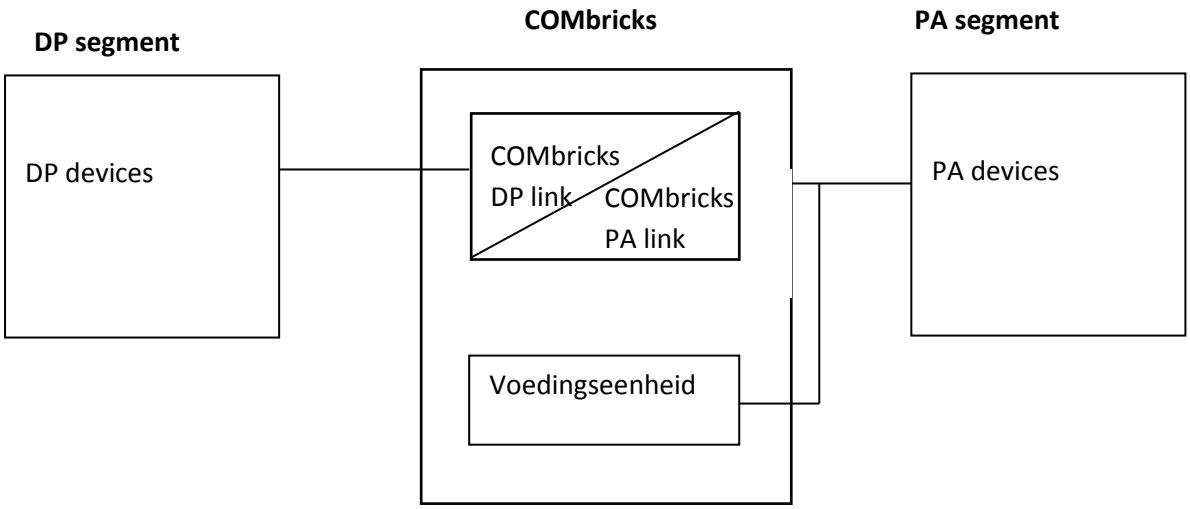
Om PROFIBUS PA- en DP-segmenten aan elkaar te koppelen heeft PROCENTEC onder meer de productlijn COMbricks ontwikkeld. COMbricks heeft de volgende taken:

- elektrische voeding van het PA-bussegment;
- berichtenconversie;
- baudrate-aanpassing;
- elektrische isolatie tussen het beveiligde en het onbeveiligde bussegment.

COMbricks: onderdelen

Een COMbricks-module bestaat uit drie onderdelen: een COMbricks PA link, een COMbricks DP link en een voedingseenheid. Zie tekening 3.1.

Tekening 3.1: COMbricks



COMbricks PA link: functies en componenten

Mijn opdracht draaide om de COMbricks PA link. Die heeft de volgende functionaliteit:

- communicatie met de DP master;
- berichtenconversie (RS485 => IEC 61158-2 doorgiftetechniek);
- baudrate-aanpassing (12 Mbits =>31,25kBits);
- communicatie met de PA devices.

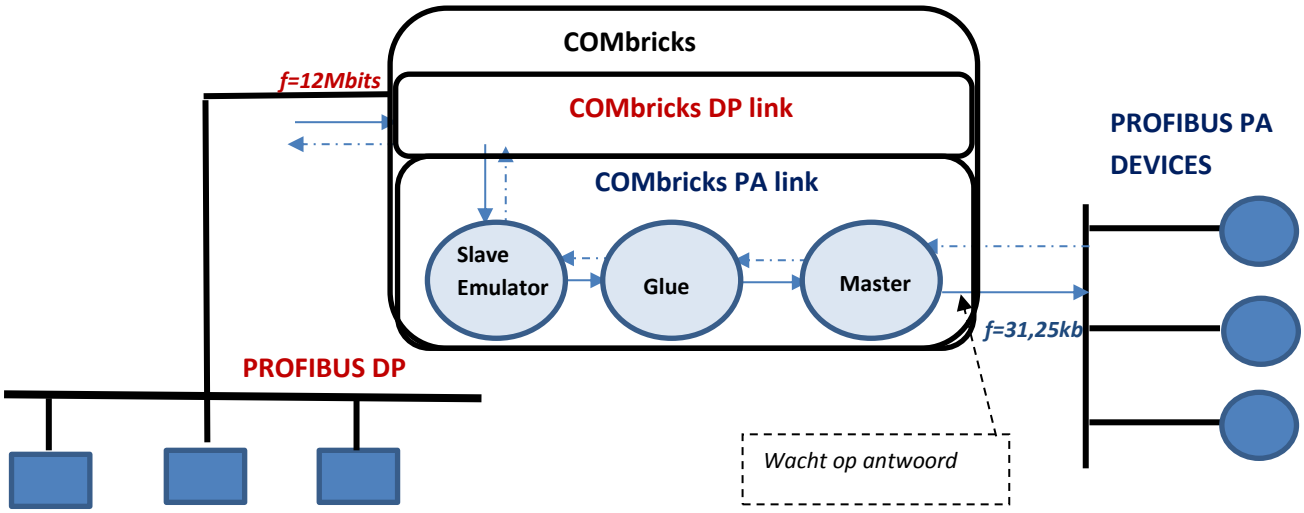
De COMbricks PA link draait op een Microblaze-processor en heeft een geïntegreerde FPGA-schakeling.

De software van de COMbricks PA link voorziet in een timer, scheduler en queues, maar niet in ondersteuning door een besturingssysteem. De software bestaat uit drie onderdelen: de Slave Emulator, Glue en de Master.

- De Slave Emulator emuleert de slaves aan de kant van de DP link en communiceert met de DP master.
- Glue converteert berichten en baudrates van de DP link naar de PA link.
- De Master communiceert met het netwerk van slaves op de PA link.

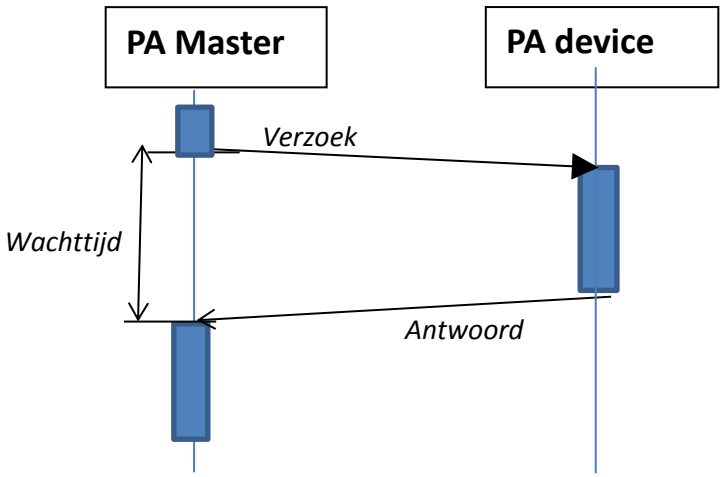
Zie tekening 3.2.

Tekening 3.2: COMbricks DP/PA link



De communicatieprocessen tussen de Master en de PA devices zijn sequentieel, zie tekening 3.3. Op de datalinklaag in de Master wordt gewacht op een antwoord van een slave.

Tekening 3.3: Communicatieproces tussen de Master van COMbricks PA link en PA slave



Parallellisering met RTOS

Bij aanvang van mijn stage kon één netwerk met maximaal 32 PA devices via COMbricks PA link met de devices van het DP segment communiceren. (Dit maximum hield verband met de voedingscapaciteit.) De wachttijd op de datalinklaag in de Master van de PA devices was acceptabel. PROCENTEC werkte aan de ontwikkeling van een nieuwe coupler device met meerdere Masters om meerdere netwerken van PA devices tegelijk op het DP segment te kunnen aansluiten. Daarmee zou de wachttijd een kritische factor in het systeem worden. Doordat de Master interne processen sequentieel op de datalinklaag uitvoerde, konden vertragingen ontstaan.

Om vertragingen te voorkomen wilde het bedrijf de interne processen van de Master van de COMbricks PA link parallelliseren. De tot dusver toegepaste software van COMbricks PA link liet dat niet toe. Ter vervanging daarvan wilde het bedrijf overstappen op een Real-Time Operating System (RTOS) en dit in de software van COMbricks PA link integreren.

Afstudeeropdracht

De vraag voor PROCENTEC was: welke van de bestaande RTOS'en is het meest geschikt voor integratie in de bestaande software van de COMbricks PA link?

PROCENTEC gaf mij in week 6 van 2014 de opdracht om deze vraag te beantwoorden. Al snel bleek dat de prijzen van RTOS'en sterk uiteenlopen. Het bedrijf wilde in het begin van de integratie niet een duur systeem gebruiken, maar de mogelijkheid hebben om het ene systeem eenvoudig door het andere te vervangen. Daarom besloten we om een wrapper te maken met een demoapplicatie voor een geselecteerde top-drie van RTOS'en. Afhankelijk van het gebruikte systeem zou deze wrapper de functies van het huidige systeem aanroepen. Hierbij konden we dan testen hoe gemakkelijk de RTOS'en zich lieten integreren in de ontwikkelomgeving, of ze intuïtief hanteerbaar waren, wat de kwaliteit van de bronnen was en hoe makkelijk informatie te vinden was om problemen op te lossen.

Mijn afstudeeropdracht luidde dan ook: selecteer in een vergelijkend theoretisch onderzoek uit de bestaande RTOS'en een top-drie. Voer vervolgens een praktisch onderzoek uit door een demoapplicatie te schrijven en op basis van een test het RTOS te kiezen dat beschikt over de beste functionaliteit die noodzakelijk is voor de COMbricks PA link.

De resultaten van de afstudeerstageopdracht waren voor PROCENTEC:

1. een rapport van het onderzoek naar bestaande RTOS'en;
2. een demoapplicatie van drie gekozen RTOS'en met functionaliteit die voor de COMbricks PA link nodig was;
3. een wrapper voor de top-drie RTOS'en.

Deze resultaten moest ik binnen 20 weken (uiterlijk in week 26) opleveren. De code van de demoapplicatie moest duidelijk commentaar bevatten.

Binnen 17 weken (uiterlijk in week 23) moest ik bij de Haagse Hogeschool het afstudeerverslag met de beschrijving van het werkproces inleveren.

Gebruikte hardware en software

De uitvoering van het project vereiste de volgende hard- en software.

Hardware:
1. PC
2. Internettoegang
3. Board met Microblaze-processor (processor waarop de COMbricks PA link draait)

Software:
1. Besturingssysteem: Windows 7
2. Eclipse-based IDE
3. Smart GIT voor de ontwikkeling van de demoapplicatie

4. Ontwikkelmethode en planning

Dit hoofdstuk beschrijft hoe ik aan het begin van het project een ontwikkelmethode koos en de planning vaststelde.

4.1 Keuze van een ontwikkelmethode

Bij de keuze van mijn ontwikkelmethode paste ik de Klassieke Contingentieanalyse[1] toe om de risico's van het project te minimaliseren. Ik analyseerde de volgende situationele factoren:

- **de projectgrootte**;
  Hierbij ging het om de kosten van het project, de duur van het project en het aantal betrokken ontwikkelaars.
- **de mate van gestructureerdheid**;
  Dit is de mate waarin de oplossing van het probleem in een procedure of programma kan worden beschreven. In dit geval lagen de eisen waaraan de demoapplicatie moest voldoen, vooraf niet vast; ze hingen af van de uitkomsten van het theoretische onderzoek. De enige onderzoeksvraag was: welk RTOS is voor PROCENTECs doelen het meest geschikt?
- **taakinzicht bij de gebruikers**;
  De demoapplicaties die tijdens het project werden ontwikkeld, waren bedoeld voor de software-ingenieurs van PROCENTEC.
- **deskundigheid van de ontwikkelaar in het probleemdomein**;
  Voordat ik met dit afstudeerproject begon, had ik geen ervaring met PROFIBUS.
- **de veranderlijkheid van de probleemruimte**.
  De onderzoeksvraag van het project lag vanaf het begin vast: welk RTOS is voor PROCENTECs doelen het meest geschikt?

Bij de uitvoering het afstudeerproject kon ik niet vooraf de eisen voor de RTOS'en vaststellen. Want de eisen waren afhankelijk van de resultaten van het theoretische onderzoek dat eerst werd uitgevoerd. Het bouwen van de demoapplicatie was afhankelijk van deze eisen. Daarom was de mate van gestructureerdheid van het afstudeerproject laag. Bovendien moest ik rekening houden met mijn geringe kennis van bestaande RTOS'en. Voor het begin van dit project had ik alleen met FreeRTOS gewerkt. Ik had ook geen kennis met PROFIBUS. Dus de deskundigheid van de ontwikkelaar in het probleemdomein was laag.

De resultaten van de analyse staan in tabel 4.1.

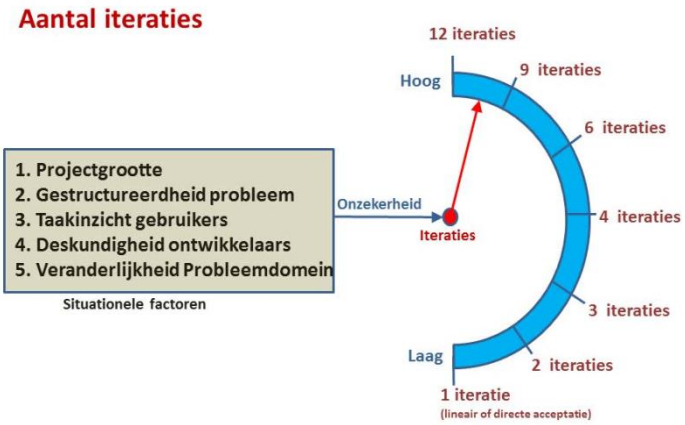
[1] Systeemontwikkelmethoden. Sheets van Henk van den Bosch.

Tabel 4.1: Factoren in het afstudeerproject volgens de Klassieke Contingentieanalyse

Soort	**Gradatie**	**Aandeel onzekerheid**
Projectgrootte	Klein	Laag
Mate van gestructureerdheid	Zwak gestructureerd	Hoog
Taakinzicht gebruikers	Volledig	Laag
Deskundigheid ontwikkelaars in het probleemdomein	Laag	Hoog
Veranderlijkheid probleemruimte	Laag	Laag

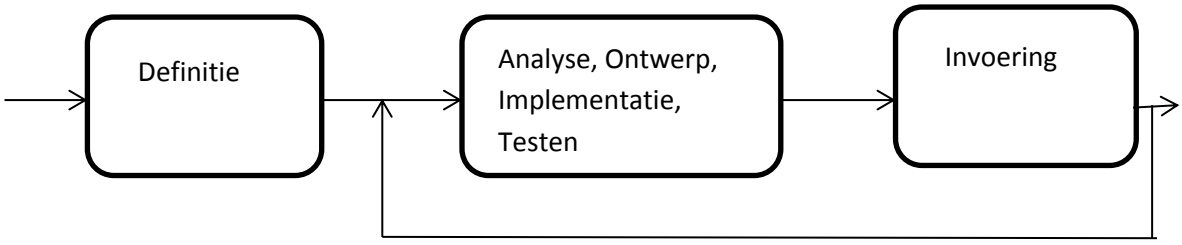
Het project had een onzekerheid van 40% , dus de optimale strategie voor dit project was de iteratieve strategie, met 3-4 iteraties erin. Zie schema 4.1.

Schema 4.1: Contingentieanalyse. Aantal Iteraties.



De definitiefase hoefde niet iteratief te zijn, want de uitgangsvraag en de probleemruimte veranderden niet. De invoeringsstrategie is in de loop van het project wel geëvolueerd, zie schema 4.2.

Schema 4.2: Evolutionaire invoering



De conclusies uit de analyse van de opdracht kwamen terug in het Plan van Aanpak: zie Bijlage II. Dit plan was door de opdrachtgever goedgekeurd.

4.2 Planning

Voor de uitvoering van het afstudeerstageproject koos ik de methode van Iterative Application Development (IAD). Deze methode geeft de mogelijkheid om het onderzoek naar de RTOS'en stapsgewijs uit te voeren.

IAD is een iteratieve ontwikkelmethode, die voor niet-gestructureerde systemen geschikt is. IAD heeft drie iteratieve fasen van ontwikkeling:

- *Definitiefase*
- *Pilotontwikkelingsfase:*
    - *Analyse*
    - *Ontwerp*
    - *Implementatie*
    - *Testen*
- *Invoeringsfase*

Voor het afstudeerproject plande ik twee iteraties. De eerste iteratie besteedde ik aan het theoretische onderzoek, in de tweede iteratie plande ik het praktische onderzoek. Zie tabel 4.2:

Tabel 4.2: Fasen in het project

Het onderzoek								
Iteratie 1 – Theoretisch onderzoek van RTOS'en				**Iteratie 2 – Praktisch onderzoek van RTOS'en**				
Definitiefase	*Analysefase*	*Ontwerpfase*	*Selectiefase*					*Invoeringsfase*
				Analysefase	*Ontwerpfase*	*Implementatiefase*	*Testfase*	

Iteratie 1, theoretisch onderzoek

Het doel van het theoretische onderzoek was om te komen tot een top-drie van RTOS'en die in het huidige COMbricks PA link-systeem kunnen worden geïntegreerd. Hiervoor moest ik de functionaliteit van de COMbricks PA link onderzoeken. Op basis van dit onderzoek moest ik de functionele eisen voor het RTOS vaststellen. Daarna moest ik de managers van PROCENTEC interviewen om de niet-functionele eisen voor het RTOS te bepalen, zoals de prijsgrenzen en de soorten licenties. Vervolgens moest ik naar de specificaties van de verschillende RTOS'en kijken. Op basis van deze stappen koos ik een top-drie en bepaalde ik de alternatieven.

- *Definitiefase van het project*
  Hierin bouwde ik kennis op van de bestaande RTOS'en via internet en boeken en koos ik de onderzoeksmethode.
- *Analysefase* van de eerste iteratie
  Hierin stelde ik de functionele en niet-functionele eisen voor een RTOS vast.
- *Ontwerpfase* van de eerste iteratie
  Hierin kende ik aan de vastgestelde eisen prioriteiten toe.
- *Selectiefase* van de eerste iteratie
  Hierin beoordeelde ik of de RTOS'en voldeden aan de vastgestelde eisen en bepaalde ik op basis daarvan de top-drie RTOS'en.

Iteratie 2, praktisch onderzoek

Het doel van het praktische onderzoek was om te laten zien in welke mate de top-drie RTOS'en aan de vastgestelde eisen voldeden en zich lieten integreren in de COMbricks PA link. Daarvoor moest ik demoversies van de geselecteerde RTOS'en bij de ontwikkelaars opvragen en een demoapplicatie maken op basis van de eisen aan RTOS-functionaliteit. Ten slotte analyseerde ik de testresultaten en concludeerde ik op basis van deze analyse of het doel van het onderzoek was bereikt.

- *Analysefase* van de tweede iteratie
  Hierin stelde ik de functionaliteiten van het RTOS vast die voor de COMbricks PA link nodig zijn en die ik zou gaan demonstreren. Ik bedacht ook testcases voor de demoapplicatie.
- *Ontwerpfase* van de tweede iteratie
  Hierin maakte ik het ontwerp van de demoapplicatie en van de wrapper voor de RTOS'en.
- *Implementatiefase* van de tweede iteratie
  Hierin implementeerde ik de demoapplicatie en de wrapper.
- *Testfase* van de tweede iteratie

  Hierin testte ik de demoapplicatie en de wrapper en analyseerde ik de testresultaten.
- *Invoeringsfase van het project*.
  Hierin heb analyseerde ik de resultaten van het theoretische en praktische onderzoek. Op basis van deze analyse koos ik het RTOS dat voor de COMbricks PA link het best geschikt is.

WBS

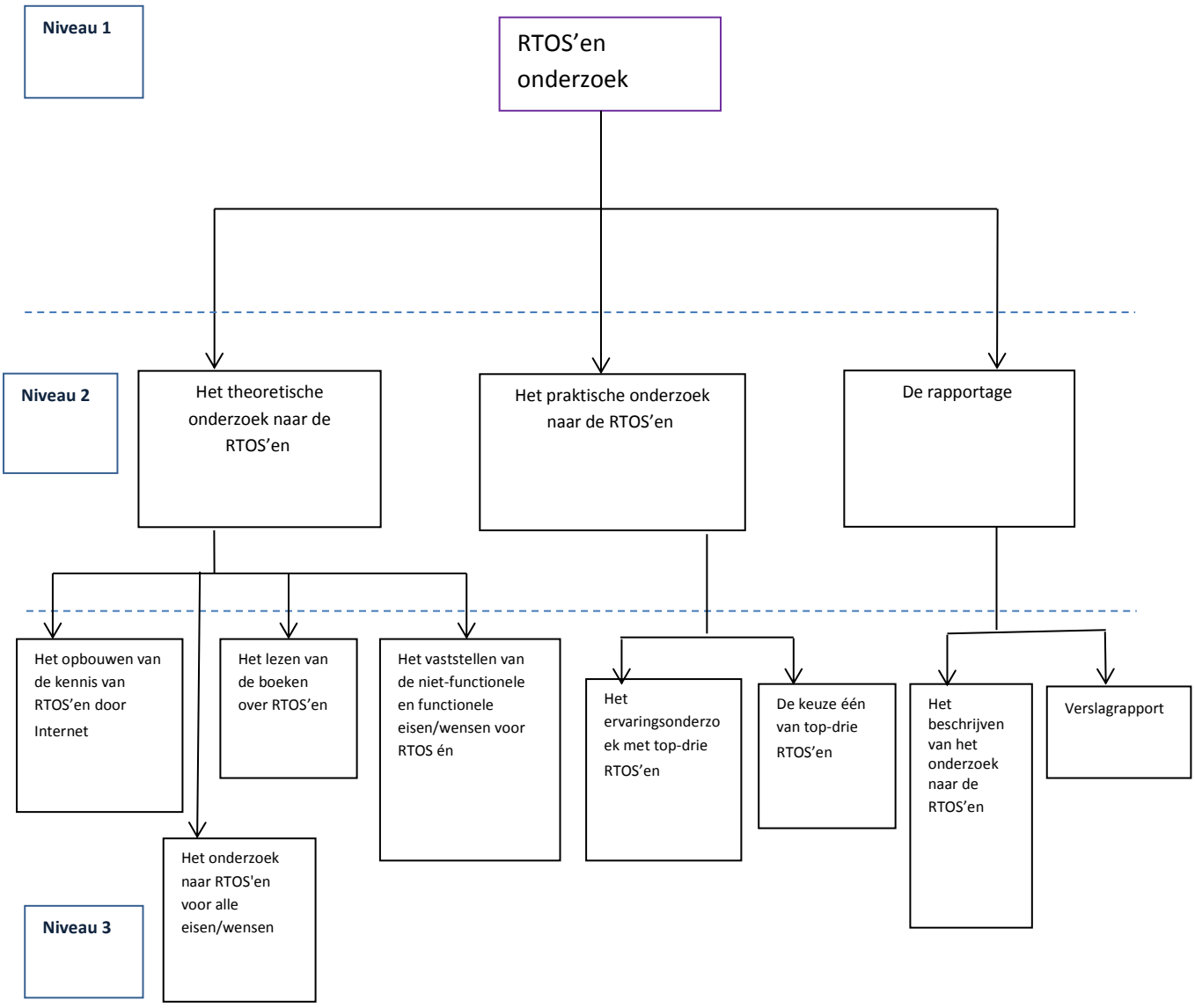
PROCENTEC werkt voor de planning van projecten met de *Work Breakdown Structure*, WBS.

"*WBS is a hierarchical and incremental decomposition of the project into phases, deliverables and work packages. It is a tree structure, which shows a subdivision of effort required to achieve an objective; for example a program, project, and contract.*"[1]

Het was wenselijk dat ik het afstudeerproject ook volgens deze methode plande. In schema 4.3 staat het WBS-plan tot het derde detailniveau. In Bijlage III staat de gedetailleerde WBS-planning, waaraan de tijdsplanning en de fasen van het project waren gekoppeld.

[1] http://en.wikipedia.org/wiki/Work_breakdown_structure 22 februari 2014

Schema 4.3: Work Breakdown Structure voor het project



5. Theoretisch onderzoek naar RTOS'en

Dit hoofdstuk beschrijft de eerste iteratie van het stageproject: het theoretische onderzoek naar de bestaande RTOS'en. Het resultaat van deze iteratie was de opstelling van een top-drie van RTOS'en.

In tabel 5.1 staan de activiteiten die ik in de eerste drie fasen van de eerste iteratie heb uitgevoerd.

Tabel 5.1: Doelen per fase van de eerste iteratie

Fase	Doelen
Analysefase	Informatie over bestaande RTOS'en verzamelen en onderzoeksmethode bepalen; de verzamelde informatie analyseren; functionele en niet-functionele eisen vaststellen
Ontwerpfase	Met de MoSCoW-methode prioriteiten aan eisen toekennen
Selectiefase	De RTOS'en die aan de vastgestelde eisen voldoen, vergelijken en wegen

5.1 Iteratie 1: theoretisch onderzoek naar RTOS'en. Definitiefase

Voor deze fase had ik twee weken uitgetrokken. Het doel van deze fase was om informatie over bestaande RTOS'en te verzamelen.

Momenteel bestaan er meer dan 100 RTOS'en. Ik wil beginnen met een algemene definitie van RTOS.

"*Een realtimebesturingssysteem (Eng: real-time operating system, RTOS) is een besturingssysteem waarin de realtime-aspecten de nadruk hebben. Dit wil zeggen dat het besturingssysteem taken kan uitvoeren op door de gebruiker aangeduide tijdstippen en met een door de gebruiker opgegeven prioriteit.*"[1]

De meeste informatie over de RTOS'en kan op internet worden gevonden. Sommige RTOS'en hebben een officiële website met een beschrijving en forums erin. Over de populaire RTOS'en zijn boeken geschreven.

Als de niet-technische kenmerken zijn bekeken, kunnen de RTOS'en in twee categorieën worden verdeeld: commerciële en niet-commerciële. Voor dit project zijn de RTOS'en in beide categorieën onderzocht, want er zijn weinig niet- commerciële RTOS'en.

5.2 Iteratie 1: theoretisch onderzoek naar RTOS'en. Analysefase

Het doel van deze fase was de functionele en niet-functionele eisen voor het RTOS vast te stellen. Tijdens de analyse van de RTOS'en keek ik naar twee belangrijke gebieden:

1. Functionaliteit: het RTOS heeft de functionaliteit die noodzakelijk is voor de uitvoering van de taken van COMbricks.
2. Betrouwbaarheid: het RTOS heeft invloed op de betrouwbaarheid van COMbricks.

[1] http://nl.wikipedia.org/wiki/Realtimebesturingssysteem 12 maart 2014

Basisvereisten voor het RTOS

Om de basisvereisten voor het RTOS vast te stellen analyseerde ik het probleemdomein van het project. Ik onderzocht de functionaliteit van de COMbricks PA link en besprak de vereisten met de begeleider vanuit het bedrijf.

De COMbricks PA link is een adapter tussen twee verschillende protocollen: PA en DP, zie Tekening 1.1. De taken van de COMbricks PA link zijn:

- communiceren met de master aan de kant van de DP link;
- commando's naar de slaves aan de kant van de PA link doorgeven;
- berichten van de slaves aan de kant van de PA link ontvangen;
- deze berichten naar de master van de DP link doorsturen.

Dus het RTOS moet communicatie- en synchronisatietools hebben. De commando-uitwisseling in de COMbricks PA link gaat via één resource, die moet worden beschermd.

Hier volgen vier eisen uit. Het RTOS moet:

Eis 1. De Mutex-functie ondersteunen om de verdelen resources te kunnen beveiligen.

Eis 2. De Semaphoren-functie ondersteunen om processen te kunnen synchroniseren.

Eis 3. De Message queue-functie ondersteunen om data te kunnen uitwisselen.

Eis 4. Beschikken over het Preemptive scheduling-algoritme[1] en de mogelijkheid om prioriteiten aan de processen toe te wijzen, om processen te kunnen parallelliseren.

COMbricks werkt op de Microblaze-processor, dus de volgende eis is:

Eis 5. Het RTOS moet de Microblaze-processor ondersteunen.

Betrouwbaarheid van het RTOS

Uit gesprekken met de begeleiders bleek dat de betrouwbaarheid van het RTOS heel belangrijk is voor het bedrijf.

Bijna elke bron[2] over RTOS verdeelt RTOS'en in twee categorieën: *hard* en *soft*. Een *hard* RTOS moet een bepaalde taak in een beperkte tijd uitvoeren; als het RTOS de deadline niet haalt, is dat catastrofaal voor het hele systeem. Dit type wordt toegepast in systemen waarmee grote financiële belangen of mensenlevens gemoeid zijn, zoals de derivatenhandel, medische apparatuur en de luchtvaart. In een *soft* RTOS is een te late reactie onwenselijk, maar zijn de gevolgen van eventuele vertragingen overkomelijk. Dit geldt bijvoorbeeld in de industrie, het toepassingsgebied van COMbricks. In ons project voldeed dus een *soft* RTOS.

De factoren die de betrouwbaarheid van het RTOS bepalen, zijn:

1. De professionele technische ondersteuning van een RTOS. Als tijdens de ontwikkeling van het systeem een bug of een probleem voorkomt, geeft deze factor de mate van support bij het zoeken van de oplossing weer. De ontwikkelaar blijft niet alleen met problemen zitten.
2. De beschikbaarheid van de broncode van een RTOS. Deze factor geeft de mogelijkheid weer om de functies van het RTOS als dat nodig is grondiger te onderzoeken. Met toegang tot de

[1] Zie Begrippenlijst, Bijlage I
[2] http://www.ni.com/white-paper/3938/en/ 22 maart 2014

broncode kan een ontwikkelaar goed begrijpen hoe de functie in het systeem is ingebouwd. Deze mogelijkheid verhoogt de kwaliteit van het ontwikkelde systeem dat het RTOS gebruikt, en daarmee de betrouwbaarheid van het ontwikkelde systeem.

3. Aanwezigheid van het industriecertificaat. Afhankelijk van de gebruikssector worden verschillende eisen gesteld aan de betrouwbaarheid van de RTOS'en. Bij bijvoorbeeld gebruik in de avionica (in de luchtvaart) moet het RTOS 100% betrouwbaar zijn. Om de betrouwbaarheid van het RTOS te garanderen zijn certificaten voor de RTOS'en ontwikkeld. Deze certificaten zijn een directe indicator voor de betrouwbaarheid van het RTOS. Uit onderzoek tijdens de definitiefase bleek dat sommige commerciële RTOS'en gecertificeerd zijn. In tabel 5.2 staan de standaarden van de certificaten voor de RTOS'en.

Tabel 5.2: Standaarden van certificaten

Certificaat	Omschrijving	Niveaus
IEC 61508	Internationale standaard voor de industrie	SIL 1 t/m SIL 4
IEC 62304	Internationale standaard voor medische apparatuur	
DO-178B **ED12-B**	Internationale standaard voor avionica	

COMbricks wordt in de industrie gebruikt. Hier heeft het gedrag van het RTOS invloed op het werk van de bestuurde apparaten, maar levert het geen gevaar op voor mensen en mensenlevens. Daarom is op dit project de industriestandaard IEC 61508 van toepassing. Dit certificaat heeft vier niveaus van Safety Integrity Levels (SIL), die de betrouwbaarheid van het systeem bepalen. Zie tabel 5.3.

Ik besprak het SIL-niveau met de begeleider vanuit het bedrijf besproken. Het derde niveau voorziet dat het systeem een faalkans heeft van minder dan 10^{-7} per uur of minder dan 10^{-3} per gebruik. Dit bepaalt meer dan 99,9% van de betrouwbaarheid van het systeem. Het was wel wenselijk dat het systeem zo hoog mogelijke betrouwbaarheid zou hebben. Daarom besloten we om voor het RTOS niveau 3 te kiezen.

Tabel 5.3: Safety Integrity Levels[1]

SIL	**Systemen die minder dan één maal per jaar gebruikt worden: gemiddelde kans op een falen per gebruik**			**Systemen die één maal per jaar of vaker, of doorlopend gebruikt worden: kans op een gevaarlijk falen per uur**
1	≥ 1 % tot < 10 %	ofwel	≥ 10^{-2} tot < 10^{-1}	≥ 10^{-6} tot < 10^{-5}
2	≥ 0,1 % tot < 1 %	ofwel	≥ 10^{-3} tot < 10^{-2}	≥ 10^{-7} tot < 10^{-6}
3	≥ 0,01 % tot < 0,1 %	ofwel	≥ 10^{-4} tot < 10^{-3}	≥ 10^{-8} tot < 10^{-7}
4	< 0,01 %	ofwel	< 10^{-4}	< 10^{-8}

Hieruit volgen eisen aan de betrouwbaarheid van het systeem:

Eis 6. Het RTOS moet technische ondersteuning hebben.
Eis 7. De broncode van het RTOS moet beschikbaar zijn.
Eis 8. Het RTOS moet het industriecertificaat IEC-61508 SIL3 hebben.

[1] http://nl.wikipedia.org/wiki/Safety_Integrity_Level 25 maart 2014

Simpele integratie van het RTOS

Belangrijk was ook dat het RTOS eenvoudig in de bestaande systeem moest kunnen worden geïntegreerd. Om de factoren te bepalen die de integratie van een RTOS in het bestaande systeem vereenvoudigden, moest ik de huidige omgeving van het bestaande systeem analyseren. Het bestaande systeem was in Windows ontwikkeld met de GCC-compiler. Het bedrijf had licenties voor Windows en geen reden om naar een ander besturingssysteem over te stappen. Het huidige systeem was in Eclipse-based IDE ontwikkeld en het was heel wenselijk om deze IDE niet te veranderen. Hieruit volgden de volgende eisen aan het RTOS:

Eis 9. Het RTOS moet de GCC-compiler ondersteunen.

Eis 10. De software onder het RTOS moet op een Windows-host kunnen worden ontwikkeld.

Eis 11. De Eclipse-based IDE moet met het RTOS worden gebruikt.

Populariteit van het RTOS

Tijdens de ontwikkeling van systemen raadpleegt een developer vaak forums, boeken en websites. Een breed gebruik van een RTOS maakt het werk aan het project makkelijker. Hieruit volgden:

Eis 12. Er moet op internet en in boeken voldoende informatie over het RTOS beschikbaar zijn.

Eis 13. Het systeem moet niet heel nieuw zijn en minimaal al drie jaar bestaan.

Standaardisatie van het RTOS

Het was handig als het RTOS compatibel zou zijn met POSIX.

"POSIX, de afkorting van Portable Operating-System Interface, is een standaard waaronder de IEEE-1003 standaarden en delen van ISO/IEC 9945 vallen. POSIX is een handelsmerk van de IEEE en The Open Group. Veel op Unix gebaseerde besturingssystemen houden zich geheel of grotendeels aan de POSIX-standaard. Daardoor is het mogelijk om programma's die op een van deze systemen zijn ontwikkeld en zich conformeren aan de door de Portable Applications Standards Committee ontwikkelde POSIX-standaard, ook te compileren en gebruiken op een ander systeem."[1]

De volgende eis maakte de ontwikkeling van het systeem makkelijker:

Eis 14. Het RTOS moet POSIX-conform zijn.

Het gebruik in de toekomst van het RTOS

In de toekomst wil PROCENTEC het geselecteerde RTOS in andere systemen implementeren en kunnen gebruiken met andere devices. De andere systemen die het bedrijf heeft ontwikkeld, maken gebruik van Borland- en IAR-compilers en werken met de ARM-processor. Hieruit volgden als eisen:

Eis 15. Het RTOS moet de IAR-compiler ondersteunen.

Eis 16. Het RTOS moet de Borland-compiler ondersteunen.

Eis 17. Het RTOS moet de ARM-processor ondersteunen.

Prijs van het RTOS

Er zijn commerciële en enkele niet-commerciële RTOS'en. Voor dit project heb ik beide categorieën

[1] http://nl.wikipedia.org/wiki/POSIX 23 maart 2014

onderzocht. De prijs van het RTOS kwam als weegfactor in de implementatiefase aan de orde. De reden daarvoor was dat de prijs alleen bekend kon worden na contact met de ontwikkelaar van het RTOS. Daarom besloot ik om eerst de RTOS'en te selecteren die aan de eisen voldoen en daarna de prijs ervan bij de ontwikkelaars op te vragen.

Samenvatting

Aan het eind van de analysefase kon ik na overleg met de begeleider vanuit het bedrijf en met zijn goedkeuring de volgende functionele en niet-functionele eisen aan het RTOS vaststellen:

Functionele eisen:

Eis 1. Het RTOS moet de Mutex-functie ondersteunen.

Eis 2. Het RTOS moet de Semaphoren-functie ondersteunen.

Eis 3. Het RTOS moet de Message queue-functie ondersteunen.

Eis 4. Het RTOS moet beschikken over het Preemptive scheduling-algoritme.

Niet-functionele eisen:

Eis 5. Het RTOS moet de Microblaze-processor ondersteunen.

Eis 6. Het RTOS moet technische ondersteuning hebben.

Eis 7. De broncode van het RTOS moet beschikbaar zijn.

Eis 8. Het RTOS moet het industriecertificaat IEC-61508 SIL3 hebben.

Eis 9. Het RTOS moet de GCC-compiler ondersteunen.

Eis 10. De software onder het RTOS moet op een Windows-host kunnen worden ontwikkeld.

Eis 11. De Eclipse-based IDE moet met het RTOS worden gebruikt.

Eis 12. Er moet op internet en in boeken voldoende informatie over het RTOS beschikbaar zijn.

Eis 13. Het systeem moet niet heel nieuw zijn en minimaal al drie jaar bestaan.

Eis 14. Het RTOS moet POSIX-conform zijn.

Eis 15. Het RTOS moet de IAR-compiler ondersteunen.

Eis 16. Het RTOS moet de Borland-compiler ondersteunen.

Eis 17. Het RTOS moet de ARM-processor ondersteunen.

5.3 Iteratie 1: theoretisch onderzoek naar RTOS'en. Ontwerpfase

Dit onderzoek was onderdeel van een groot project van het bedrijf. Het einddoel van dit project was de integratie van het RTOS in een nieuw device dat op de software van de COMbricks PA link was gebaseerd. Tijdens de prioritering moest ik met dit einddoel rekening houden.

In de vorige fase had ik de eisen aan het RTOS vastgesteld. Natuurlijk waren niet alle eisen even belangrijk. Het doel van deze fase was om aan de eisen prioriteiten toe te kennen.

Prioritering van de eisen

De eisen die voor het RTOS waren vastgesteld, hadden een verschillend gewicht. De volgende stap van het onderzoek was de verschillende weegfactoren of prioriteiten voor de eisen vast te stellen om te bepalen aan welke eisen het RTOS moest voldoen en welke eigenschappen van het RTOS alleen wenselijk waren.

Daarvoor zijn er verschillende methodes, zoals MoSCoW[1] en Multicriteria-analyse[2]. Er is geen principieel verschil tussen deze twee methodes. Voor het bedrijf maakte het niet uit welke methode ik koos, zolang de weegfactoren maar terecht aan de eisen waren toegekend. Ik koos voor dit project de MoSCoW-methode omdat ik er ervaring mee had.

De MoSCoW-methode stelt prioriteiten aan de eisen van het systeem. In de naam van de methode staan de letters M, S, C en W voor: *Must haves*, *Should haves*, *Could haves* en *Won't need to haves.*

- **M – must haves**:
  als het RTOS niet voldoet aan deze eis, is het onmogelijk in het huidige systeem te integreren en kan het project het einddoel niet bereiken.
- **S – should haves**:
  als het RTOS niet voldoet aan deze eis, is integratie mogelijk en kan het project het einddoel bereiken, maar kost dit ten eerste veel extra tijd en geld tijdens de ontwikkeling en kan dit ten tweede een slechte invloed hebben op de kwaliteit van het eindproduct.
- **C – could haves**:
  als het RTOS voldoet aan eisen in deze categorie verloopt de integratie makkelijker en wordt tijd bespaard, maar ook in het andere geval kan het project op tijd worden uitgevoerd.
- **W – won't need to haves:**
  eisen in deze categorie vergroten de mogelijkheden om het project in toekomst uit te breiden, maar zijn voor het einddoel van het huidige project niet belangrijk.

In het vervolg noem ik de vereisten in de categorieën *Must haves* en *Should haves* "eisen" en die in de categorieën *Could haves* en *Won't need to haves* "wensen".

Dit leverde de volgende hiërarchie van eisen op:

[Must haves]

Functionele eisen:

- Het RTOS moet de Mutex-functie ondersteunen.
- Het RTOS moet de Semaphoren-functie ondersteunen.
- Het RTOS moet de Message queue-functie ondersteunen.
- Het RTOS moet beschikken over het Preemptive scheduling-algoritme.

Deze eisen betroffen de functionaliteit van het RTOS: ze waren noodzakelijk om de processen van COMbricks te parallelliseren, wat het doel was van de integratie van het RTOS in COMbricks. Om deze reden vielen deze eisen onder de categorie *Must haves*.

[1] http://www.coleyconsulting.co.uk/moscow.htm 27 maart 2014
[2] "Multicriteria Analysis in Engineering", R.B. Statnikov J.B. Matusov**,** Kluwer Academic Publishers, 2002

Niet-functionele eisen:

- Het RTOS moet de Microblaze-processor ondersteunen.

Ook deze eis viel onder de categorie *Must haves*, want de COMbricks PA link werkt op basis van de Microblaze-processor. Als het RTOS niet voldeed aan deze eis, moest er een volledig nieuw device worden ontwikkeld.

[Should haves]

Niet-functionele eisen:

- Het RTOS moet technische ondersteuning hebben.
- De broncode van het RTOS moet beschikbaar zijn.

Deze eisen verhogen de betrouwbaarheid van het RTOS, ze zijn een soort garantie van de kwaliteit van het eindproduct. Er bestond een risico dat tijdens de ontwikkeling van het systeem problemen zouden opduiken die het bedrijf niet zelf, of niet snel kon oplossen. In zo'n situatie is technische support en toegang tot de broncode noodzakelijk. Deze eisen vielen onder de *Should haves*.

- Het systeem moet niet heel nieuw zijn en minimaal drie jaar bestaan.
- Er moet op internet en in boeken voldoende informatie over het RTOS beschikbaar zijn.

Deze eisen bieden een garantie dat de ontwikkelaar eenvoudig technische informatie over het RTOS kan vinden en geven een indicatie van de mate van de verspreiding van het gebruik. Als het RTOS niet aan deze eisen voldeed, bestond er een groot risico dat het project een vertraging opliep. Hierom vielen deze eisen onder de categorie *Should haves*.

- Het RTOS moet de GCC-compiler ondersteunen.
- De software onder het RTOS moet op een Windows-host kunnen worden ontwikkeld.

Als het RTOS niet aan deze eisen voldeed, kon veel tijd verloren gaan met het aanpassen van de code van COMbricks aan een andere compiler of een ander besturingssysteem. Om deze reden vielen deze eisen onder de categorie *Should haves*.

[Could haves]

- ***Niet-functionele wensen:***
- Het RTOS kan POSIX-conform zijn.

Als het RTOS aan deze eis voldoet is dat wenselijk en behulpzaam, maar ook in het andere geval kan het project op tijd worden uitgevoerd. Daarom viel deze eis onder de categorie *Could haves*.

- Het RTOS kan het industriecertificaat IEC-61508 SIL3 hebben.

Aan de ene kant is deze eis een garantie voor de kwaliteit van het RTOS, aan de andere kant zijn deze certificaten belangrijk voor systemen waarin fouten levensbedreigend kunnen zijn, zie 5.3. Aanwezigheid van deze certificaten verhoogt de prijs van het RTOS, waarmee ook rekening moet worden gehouden: certificaten zijn heel wenselijk, maar verhogen het budget van het project. Daarom viel deze eis onder de categorie *Could haves*.

- De Eclipse-based IDE kan met het RTOS worden gebruikt.

Voldoening aan deze eis bespaart tijd, maar heeft geen invloed op de kwaliteit van het eindproduct. Daarom viel deze eis onder de categorie *Could haves*.

[Won't need to haves]

Niet-functionele wensen:

- Het RTOS kan de IAR-compiler ondersteunen.
- Het RTOS kan de Borland-compiler ondersteunen.
- Het RTOS kan de ARM processor ondersteunen.

Deze eisen waren niet noodzakelijk voor het huidige project, maar wel toepasbaar op de toekomstige projecten. Hierom vielen deze eisen onder de categorie *Won't need to Have*.

5.4 Iteratie 1: theoretisch onderzoek naar RTOS'en. Selectiefase

Tijdens de definitiefase heb ik een lijst met de bestaande RTOS'en gemaakt, zie Bijlage IV. Momenteel bestaan er ruim 100 RTOS'en. Het doel van deze fase was om drie RTOS'en te kiezen op basis van de vastgestelde eisen. Ik heb de selectie in vijf stappen gemaakt:

1. Alle RTOS'en die aan één of meer *Must haves* niet voldeden, vielen af.
2. Alle na stap 1 overgebleven RTOS'en die aan meer dan één *Should have* niet voldeden, vielen af.
3. De na stap 2 overgebleven RTOS'en kregen punten voor de mate waarin ze aan de wensen voldeden.
4. De na stap 2 overgebleven RTOS'en kregen, afhankelijk van de prijs, een bepaalde weegfactor.
5. Ik vergeleek alle weegfactoren en koos de top-drie RTOS'en.

Stap 1: selectie op *Must haves*

Ik zocht informatie over de systemen op de officiële websites van de RTOS'en, op Wikipedia en op de officiële site van Xilinx, de ontwikkelaar van Microblaze; de links staan in Bijlage IV.

Na de eerste stap bleef een lijst met elf RTOS'en over, zie tabel 5.4.

Tabel 5.4: De RTOS'en die voldoen aan de Must haves (in alfabetische volgorde)

RTOS	Officiële website
Embox	https://code.google.com/p/embox/
FreeRTOS	http://www.freertos.org/
Micrium µC/OS-II	http://micrium.com/
Micrium µC/OS-III	http://micrium.com/
NORTi	http://www.mispo.co.jp/
Nucleus RTOS v1.15	http://www.mentor.com/embedded-software/nucleus/
OpenRTOS	http://www.highintegritysystems.com/openrtos/
SafeRTOS	http://www.highintegritysystems.com/safertos/
ThreadX	http://rtos.com/products/threadx/
uClinux/Petalinux 2.6	http://www.xilinx.com/tools/petalinux-sdk.htm
Unison RTOS	http://www.rowebots.com/products/unison_rtos

Stap 2: selectie op *Should haves*

Hiervoor zocht ik informatie op internet; voor sommige details nam ik contact op met de ontwikkelaars van de RTOS'en. Het resultaat van dit onderzoek staat in Tabel 5.5. Als een systeem voldoet aan de eis, staat er de letter *V* bij; *N/V* betekent dat een systeem niet aan de eis voldoet.

De eisen in de categorie *Should haves* waren heel kritisch voor het project. Niet voldoen aan één eis was een groot nadeel voor het systeem. Een systeem dat één "niet voldoen" had, kreeg weegfactor "--"; de RTOS'en die aan alle eisen voldeden, kregen weegfactor "++". De systemen die aan meer dan één eis, niet voldoen valt uit het onderzoek en krijgen `0`. In tabel 5.5 staan de weegfactoren die zijn toegevoegd na vergelijking met de *Should haves*.

Tabel 5.5: RTOS'en en Should haves (alfabetisch)

RTOS	GCC	Windows	Techn. ond.	Broncode	Documentatie	≥ 3 jaar	Weging
Embox	V	V	N/V	V	N/V	N/V	0
FreeRTOS	V	V	N/V	V	V	V	--
Micrium µC/OS-II	V	V	V	V	V	V	++
Micrium µC/OS-III	V	V	V	V	V	V	++
NORTi	V	V	N/V	V	N/V	V	0
Nucleus RTOS v1.15	V	V	V	V	N/V	V	--
OpenRTOS	V	V	V	V	V	V	++
SafeRTOS	V	V	V	V	V	V	++
ThreadX	V	V	V	V	V	V	++
uClinux/Petalinux 2.6	V	N/V	V	V	N/V	V	0
Unison RTOS	V	N/V	V	V	N/V	V	0

Na de tweede stap bleven zeven RTOS'en interessant:

1. FreeRTOS: ontwikkelaar is Richard Barry & FreeRTOS Team
2. Micrium µC/OS-II: ontwikkelaar is Micrium
3. Micrium µC/OS-III: ontwikkelaar is Micrium
4. Nucleus RTOS v1.15: ontwikkelaar is Mentor Graphics
5. OpenRTOS: eigenaar is WITTENSTEIN Group
6. SafeRTOS: eigenaar is WITTENSTEIN Group
7. ThreadX: ontwikkelaar is Express Logic

De gezochte informatie werd gedetailleerder en steeds moeilijker op internet te vinden. Daarom voerde ik vanaf dit moment een intensieve e-mailwisseling met de ontwikkelaars van de RTOS'en. Twee van de geselecteerde RTOS'en zijn functioneel identiek: FreeRTOS en OpenRTOS. In feite is OpenRTOS de commerciële kloon van FreeRTOS met technische ondersteuning.

Stap 3: weging van wensen

De *Could haves* en *Won't need to haves* zijn wel wenselijk maar niet noodzakelijk. Ze vormen een extra weegfactor, maar geen *knock-out*vereiste.

De RTOS'en die bij vergelijking aan het maximale aantal wensen voldeden, kregen factor "++". De systemen die aan een minimumaantal wensen voldeden, kregen factor "--". Daarna kregen, afhankelijk van de mate van afwijking van de maximale en de minimale score, de andere RTOS'en

respectievelijk de weegfactoren “+” of “-”. De uitkomst van deze stap staat in tabel 5.6.

Tabel 5.6: RTOS’en en wensen

RTOS	**POSIX**	**EC-61508 SIL3**	**Eclipse**	**IAR**	**Borland**	**ARM**	**Weging**
FreeRTOS	V	N/V	V	V	N/V	V	+
Micrium µC/OS-II	V	V	V	V	V	V	++
Micrium µC/OS-III	V	N/V	V	N/V	N/V	V	–
Nucleus RTOS v1.15	N/V	V	N/V	N/V	N/V	V	– –
OPENRTOS	N/V	N/V	V	V	N/V	V	–
SafeRTOS	N/V	V	V	V	N/V	V	+
ThreadX	V	V	V	V	V	V	++

Stap 4: weging op prijs

De prijs van de RTOS’en (licenties, technische support en royalty’s) werd ook een weegfactor. Het RTOS met de hoogste prijs kreeg weegfactor “--”, het RTOS met de laagste prijs kreeg weegfactor “++”. Daarna kregen, afhankelijk van de mate van afwijking van de hoogste en laagste prijzen, de andere RTOS’en respectievelijk de weegfactoren “+” of “-”. De samenvatting van het prijzenonderzoek staat in tabel 5.7.

Tabel 5.7: Prijzenoverzicht van de RTOS’en

RTOS	**Single-project**	**Multi-project**	**Techn. support**	**Royalty’s**	**Weging**
FreeRTOS	gratis	gratis	-	gratis	++
Micrium µC/OS-II	$7.500	$37.500	$1.500 / $7.400**	gratis	–
Micrium µC/OS-III	$7.500	$37.500	$1.500 / $7.400**	gratis	–
Nucleus RTOS v1.15	€6.000*	€12. 500*		niet gratis	–
OpenRTOS	€2.625	€4.500	€750/€900**	gratis	+
SafeRTOS	-	€65.000	€13.000	gratis	– –
ThreadX	€12.500	€37.500	€2.500 / € 7.450**	gratis	–

*per user
**afhankelijk van de licentie: eerste prijs voor single-project-, tweede prijs is voor multi-projectlicentie

De prijs was een heel belangrijke factor voor het bedrijf. Het budget voor het RTOS was niet vastgesteld, maar de prijs moest wel acceptabel zijn. Na een gesprek met de begeleider viel SafeRTOS af, want dit systeem is heel duur.

Stap 5: Finale weging en bepaling top-drie

Hier telde ik de toegekende “-” en “+” op en koos ik op basis van het maximumaantal “+” en het minimumaantal “-“ de top-drie RTOS’en voor dit project. De resultaten staan in tabel 5.8.

Tabel 5.8: Vergelijking van de RTOS’en op weegfactoren

Plaats	RTOS	Weging op eisen	Weging op wensen	Prijs
1ste	**ThreadX**	++	++	–
2de	**Micrium µC/OS-II**	++	++	–
2de	**OpenRTOS**	++	–	+
4de	**FreeRTOS**	– –	+	++
5de	**Micrium µC/OS-III**	++	– –	–
6de	**Nucleus RTOS v1.15**	– –	– –	–

De beste RTOS'en voor het project waren ThreadX of Micrium µC/OS-II, want zij voldeden voor 100% aan alle eisen. Maar deze systemen waren te duur voor het bedrijf.

Het beste RTOS om te kopen was OpenRTOS, want dit voldeed aan alle eisen en aan een aantal wensen. De prijs voor dit systeem is niet hoog en zoals eerder opgemerkt is OpenRTOS de commerciële variant van FreeRTOS. Dat was heel handig voor PROCENTEC, want zo zou het mogelijk zijn om prototypen te maken met het gratis FreeRTOS en vervolgens over te stappen naar de commerciële versie met de technische support: OpenRTOS.

Op basis van dit onderzoek was het handig om drie systemen te testen: ThreadX, OpenRTOS/FreeRTOS en Micrium µC/OS-II. Met de begeleider sprak ik af om in ieder geval, onafhankelijk van de testresultaten, de wrapper voor deze systemen te maken.

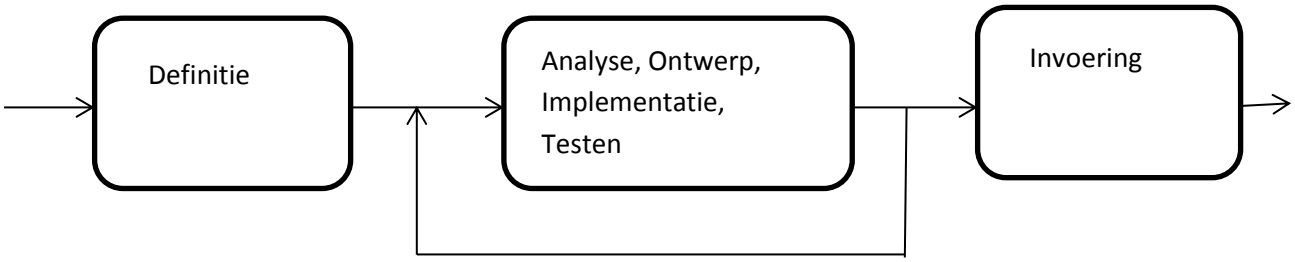
6. Heroverweging ontwikkelstrategie

Als tweede iteratie had ik het praktische onderzoek van de drie top-RTOS'en gepland. Tijdens het praktische onderzoek moest ik een demoapplicatie maken om de top-drie RTOS'en te testen. Aan het begin van mijn stageproject had ik daarvoor één iteratie gepland. Tijdens het praktische onderzoek maakte ik twee versies van de demoapplicatie en een wrapper voor de RTOS'en. Dit paste niet in één iteratie, daarom veranderde ik de strategie voor de projectontwikkeling en het aantal iteraties. Aan het begin van dit hoofdstuk beschrijf ik de wijzingen die ik in de projectontwikkeling heb aangebracht.

Ontwikkelingsstrategie

Het doel van het project was de keuze van meeste geschikte RTOS voor de COMbricks PA link. De keuze moest gebaseerd zijn op het theoretische en het praktische onderzoek. De invoeringsfase moest het geselecteerde RTOS en de motivering voor deze keuze naar voren brengen. Hiervoor moesten het theoretische en het praktische onderzoek helemaal klaar zijn. De invoeringsfase was dus niet iteratief. De Snel-ontwikkelen-strategie was het meest geschikt, zie schema 6.1.

Schema 6.1: Snel-ontwikkelen-strategie



Aantal iteraties

Tijdens het praktische onderzoek had ik vier milestones te ontwikkelen, elk met een Analyse-, Ontwerp-, Implementatie- en Testfase. Zodoende vormde de ontwikkeling van elke milestone één iteratie van het project. Vanwege bevatte het theoretische onderzoek de één iteratie en het praktische onderzoek de vier iteraties uit tabel 6.1.

Tabel 6.1: Iteraties van het onderzoek van de RTOS'en

Iteraties	Doelen	
Iteratie 1	Het kiezen van de top-drie RTOS´en	Theoretisch onderzoek
Iteratie 2	Ontwikkeling van Demoapplicatie 1.0	Praktisch onderzoek
Iteratie 3	Ontwikkeling van Demoapplicatie 1.0 met seriële interrupt	
Iteratie 4	Ontwikkeling van Demoapplicatie 2.0	
Iteratie 5	Ontwikkeling van Wrapper voor FreeRTOS/OpenRTOS, MicroC/OS-II en ThreadX	

Per hoofdstuk beschrijf ik de iteraties van het praktische onderzoek in meer detail.

7. Ontwikkeling demoapplicatie 1.0

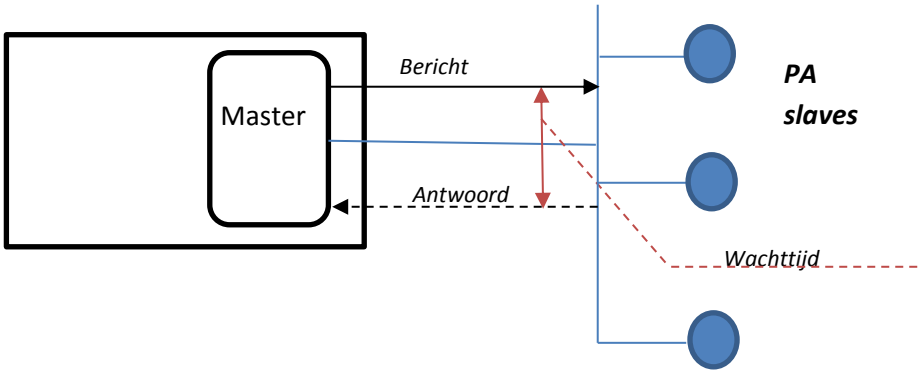
Het doel van deze iteratie was de demoapplicatie 1.0 te maken die de functionaliteit van de RTOS'en moest aantonen. Tijdens de Analysefase stelde ik de eisen voor de demoapplicatie vast. Tijdens de Ontwerpfase maakte ik de diagrammen voor de demoapplicatie en bedacht ik het testplan. In de Implementatiefase implementeerde ik de demoapplicatie. In de Testfase voerde ik de tests van de demoapplicatie 1.0 uit.

7.1 Iteratie 2: ontwikkeling demoapplicatie 1.0. Analysefase

In deze fase bepaalde ik welke functies van het RTOS voor de COMbricks PA link nodig waren.

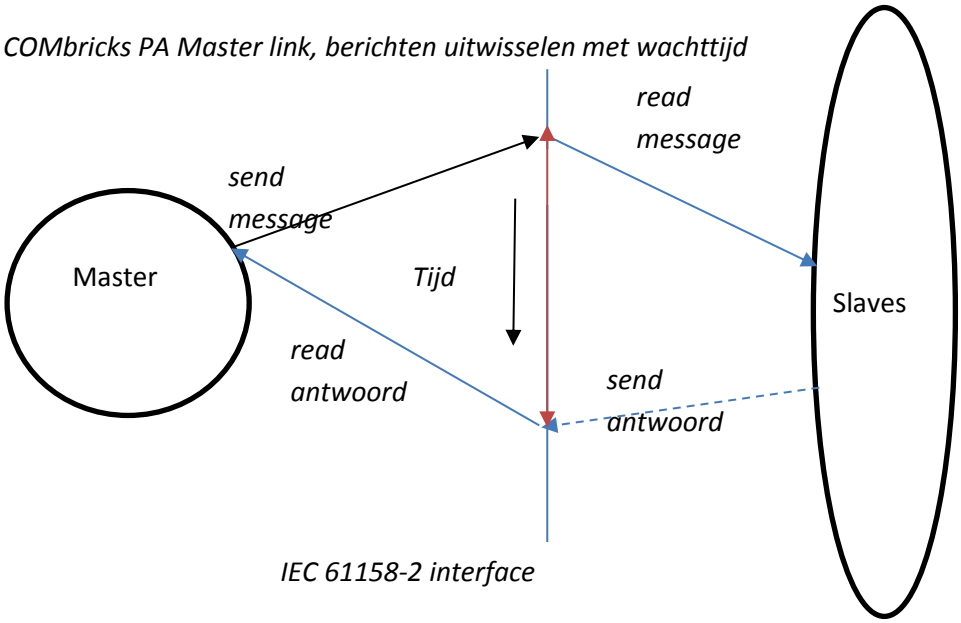
Om de vereiste functionaliteit van het RTOS voor de demoapplicatie te bepalen moest ik naar de functionaliteit van de COMbricks PA link kijken. De reden waarom het bedrijf het bestaande RTOS wilde integreren, lag in mogelijke vertragingen die bij sequentiële uitvoering van de interne processen van de COMbricks PA link op de datalinklaag konden optreden. De Master van de COMbricks PA link stuurt berichten naar de PA slaves en wacht op antwoord, zie tekening 7.1.

Tekening 7.1: COMbricks PA link - Master



De uitwisseling van berichten tussen de COMbricks PA link Master en slaves gaat via een IEC 61158-2 interface, zie tekening 7.2.

Tekening 7.2: COMbricks PA Master link, berichten uitwisselen met wachttijd



Het integreren van het RTOS in de software van de COMbricks PA link is nodig om de processen te kunnen parallelliseren. De Master van de COMbricks PA link moest via de IEC 61158-2 interface een bericht naar de slaves kunnen sturen en zonder op antwoord te wachten andere processen kunnen uitvoeren. Zodra een antwoord binnenkomt, geeft het systeem het proces van de Master de tijd om dit antwoord te lezen. Hiervoor moesten de synchronisatietools van de geselecteerde RTOS'en worden gebruikt, zoals prioritering van taken, timer events en queues. De verschillende processen van de COMbricks PA link moesten één communicatie-interface kunnen gebruiken. Om een resource te beveiligen die door meer dan één taak wordt gebruikt, is implementatie van Mutex nodig.

Het belangrijkste doel van de COMbricks PA link is communiceren met PA devices. Om dit proces te simuleren bedacht ik in de demoapplicatie 1.0 twee taken die via twee queues met elkaar communiceerden. De eerste taak simuleert de Master van de COMbricks PA en de tweede de PA devices. De eerste taak luistert periodiek naar de UART-poort[1] en stuurt de input via de queue naar de andere taak. De tweede taak verandert het bericht en stuurt het terug via de tweede queue naar de eerste taak. De eerste taak hoeft niet te wachten totdat de tweede taak helemaal zal afhandelen.

In de bestaande software van de COMbricks PA link was een time-out geïmplementeerd. Als de Master van de COMbricks PA link niet binnen een bepaalde tijd antwoord kreeg, gaf het systeem een foutmelding. Dit moest ik ook in de demoapplicatie implementeren. Daarvoor gebruikte ik de systeemtimer. De eerste taak van de demoapplicatie moet het tijdsverschil tussen het sturen van het bericht en het krijgen van het antwoord meten. Als dit verschil groter is dan de time-out, wordt een foutmelding geprint.

De enige gedeelde resource van de COMbricks PA link is de communicatie-interface. In de demoapplicatie 1.0 simuleerde deze interface een seriële UART-poort. Tijdens elk event moeten de taken een melding over het event printen via de seriële UART-poort die door Mutex kon worden beveiligd.

Aan het eind van de Analysefase stelde ik de volgende functionele eisen vast waaraan het RTOS in de demoapplicatie 1.0 moest voldoen:

Taken:

Eis 1. In de demoapplicatie 1.0 moeten taken geconfigureerd worden.

Eis 2. De demoapplicatie 1.0 moet het Preemptive algoritme[1] van het RTOS demonstreren.

Eis 3. De demoapplicatie 1.0 moet taken met verschillende prioriteiten bevatten.

Eis 4. De demoapplicatie 1.0 moet periodieke taken bevatten.

Queues:

Eis 5. In de demoapplicatie 1.0 moeten queues geconfigureerd worden.

Eis 6. In de demoapplicatie 1.0 moeten functies geïmplementeerd worden zoals Receive from the Queue en Send to the Queue.

Eis 7. In de demoapplicatie 1.0 moeten processen gesynchroniseerd worden met gebruikmaking van Queues.

Timerconfiguratie:

[1] Zie Begrippenlijst, Bijlage I

Eis 8. In de demoapplicatie 1.0 moet een systeemtimer worden gebruikt.

Eis 9. In de demoapplicatie 1.0 moet de time-out van het systeem geconfigureerd worden.

Mutex:

Eis 10. In de demoapplicatie 1.0 moet Mutex geconfigureerd worden**.**

Eis 11. In de demoapplicatie 1.0 moet een resource door Mutex worden beveiligd.

Op grond van deze systeemeisen maakte ik in de volgende fasen de demoapplicatie 1.0. Dat deed ik met FreeRTOS, de gratis variant van OpenRTOS.

OpenRTOS, ThreadX en MicroC/OS-II waren de drie RTOS'en die uit het theoretische onderzoek naar voren waren gekomen. Voor ThreadX en MicroC/OS-II kon ik een gratis demoversie aanvragen, maar de tests zou ik dan binnen de beperkte gebruiksduur van één maand moeten afronden. Met FreeRTOS kon ik onbeperkt testen.

7.2 Iteratie 2: ontwikkeling demoapplicatie 1.0. Ontwerpfase

In deze fase maakte ik een gedetailleerd ontwerp van de demoapplicatie 1.0 en de diagrammen daarvoor. Daarna bedacht ik de testcases voor de tests van de demoapplicatie 1.0 en stelde ik op basis daarvan het testplan op.

De afdeling Development & Research van PROCENTEC gebruikte Yourdon-diagrammen[1] om systemen te beschrijven. De eis van het bedrijf was dat ik voor dit project ook Yourdon-diagrammen gebruikte.

De Yourdon-notatie bevat drie typen van diagrammen[2]:

1) *dataflow*-diagrammen;
2) *state-transition*-diagrammen;
3) entiteit-relatiediagrammen.

Het *dataflow*-diagram biedt een grafische voorstelling van de informatiestromen binnen een bepaald systeem. Dit diagram beschrijft de processtappen, informatieveranderingen en de context waarin een proces actief is. Het *state-transition*-diagram beschrijft de toestanden van de processen in het systeem en de overgangen daarvan naar andere toestanden. Het *entiteit-relatiediagram* geeft een grafisch overzicht van de relaties tussen de objecten in het systeem. Het Yourdon-model is hiërarchisch, de diagrammen zijn opgebouwd van het hoogste niveau naar de lagere. Hoe lager het niveau, des te meer het systeem is gedetailleerd. In tabel 7.1 staat de beschrijving van de Yourdon-symbolen die ik in dit project heb gebruikt.

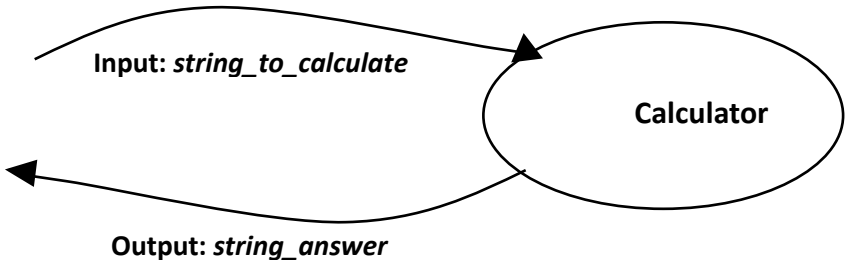
[1] Zie Begrippenlijst, Bijlage I
[2] "Structured Development for Real-Time Systems", Paul T. Ward, Stephen J. Meller, 1985

Tabel 7.1: Yourdon-symbolen

Symbool	Beschrijving
Entiteiten	Externe subjecten die met het systeem interageren.
	Een proces in het systeem.
	De stromen; zij vormen de verbindingen tussen de processen.
Data Store	Een verzameling gegevens in rusttoestand.

Ik besprak de eisen voor de demoapplicatie 1.0 met de begeleider. Hij gaf de tip om een demoapplicatie met een duidelijk doel te maken. Ik besloot om een rekenmachine te maken, zie schema 7.1. De gebruiker voert via de terminal twee decimale of hexadecimale getallen in voor berekening en krijgt als output een antwoord. Ik noemde het systeem *Calculator*.

Schema 7.1: Contextdiagram voor Calculator 1.0



Tijdens de Analysefase bedacht ik de globale logica van de demoapplicatie 1.0. De demoapplicatie moest twee processen/taken bevatten.

De functies van de eerste proces/taak waren:

1) luisteren naar een UART-poort
2) de aankomsttijd van het bericht onthouden
3) de correctheid van het verzoek controleren
4) het bericht via een queue naar de andere taak sturen
5) het antwoord uit een andere queue lezen
6) de time-out checken
7) het antwoord printen

De functies van de tweede proces/taak waren:

1) het bericht uit de eerste queue uitlezen
2) het bericht verwerken
3) de berekening uitvoeren
4) het bericht via de tweede queue terugsturen

Tijdens de Ontwerpfase besloot ik om in de demoapplicatie 1.0 meer taken en queues in te voegen. De redenen daarvoor waren:

1. Een demoapplicatie met slechts twee processen/taken ontwerpen is heel chaotisch, want één proces moet veel verschillende functionaliteiten hebben, zoals bericht uitlezen, input checken, aankomsttijd onthouden, hexadecimale getallen naar decimalen en terug converteren. Daardoor wordt het hoofddoel van elk proces onduidelijk.
2. Het gedrag van elk RTOS in verschillende situaties moest worden bekeken, zoals:
    - taken sturen berichten tegelijkertijd naar één queue;
    - één taak moet verschillende queues uitlezen.
3. In een demoapplicatie met twee taken is slechts één van de twee volgende varianten mogelijk:
    - de taken hebben dezelfde prioriteit; of
    - de taken hebben verschillende prioriteit.

    Maar het was noodzakelijk om het gedrag van het systeem te bekijken bij uitvoering van taken met dezelfde én verschillende prioriteiten.

Om deze redenen voegde ik vijf taken toe aan de demoapplicatie 1.0:

- *Control_task (*functies *2, 5, 6, 7* van de eerste taak en *1* van de tweede taak*),*
- *Calc_HEX_task(*functie *2* en *4* van de tweede taak*)*,
- *UART_task(*functies *1, 4* van de eerste taak*)*,
- *Interpreter_task* (functie *3* van de eerste taak en *2* van de tweede taak),
- *Calc_DEC_task* (functie *2* en *4* van de tweede taak*)*.

De belangrijkste belasting van *Calculator*-systeem lag op de *Control_task*: deze taak beheert de verzoeken van de gebruiker en de antwoorden van het systeem. De *UART_task* moet snel worden uitgevoerd, waarom deze taak de hoogste prioriteit kreeg. De andere taken kregen lagere prioriteiten met allemaal eenzelfde niveau. Elk van deze taken had een hoofdfunctie in de demoapplicatie en demonstreerde een bepaalde functionaliteit van het RTOS, zie tabel 7.2.

Voor het uitwisselen van berichten tussen de taken creëerde ik vijf queues:

- *queue_UART* (voor de communicatie tussen *UART_task* en *Control_task*),
- *queue_IN* (voor de communicatie tussen *Control_task* en *Interpreter_task*),
- *queue_OUT* (voor de communicatie tussen *Calc_DEC_task, Calc_HEX_task, Interpreter_task en Control_task*),
- *queue_DEC* en
- *queue_HEX* (voor de communicatie tussen *Interpreter_task* en *Calc_DEC_task, Calc_HEX_task* respectievelijk).

Om de time-out te checken heb ik zesde queue gemaakt:

- *queue_State*, die de aankomsttijd van de berichten bewaart.

Tabel 7.2: Doelen van de taken in Calculator 1.0

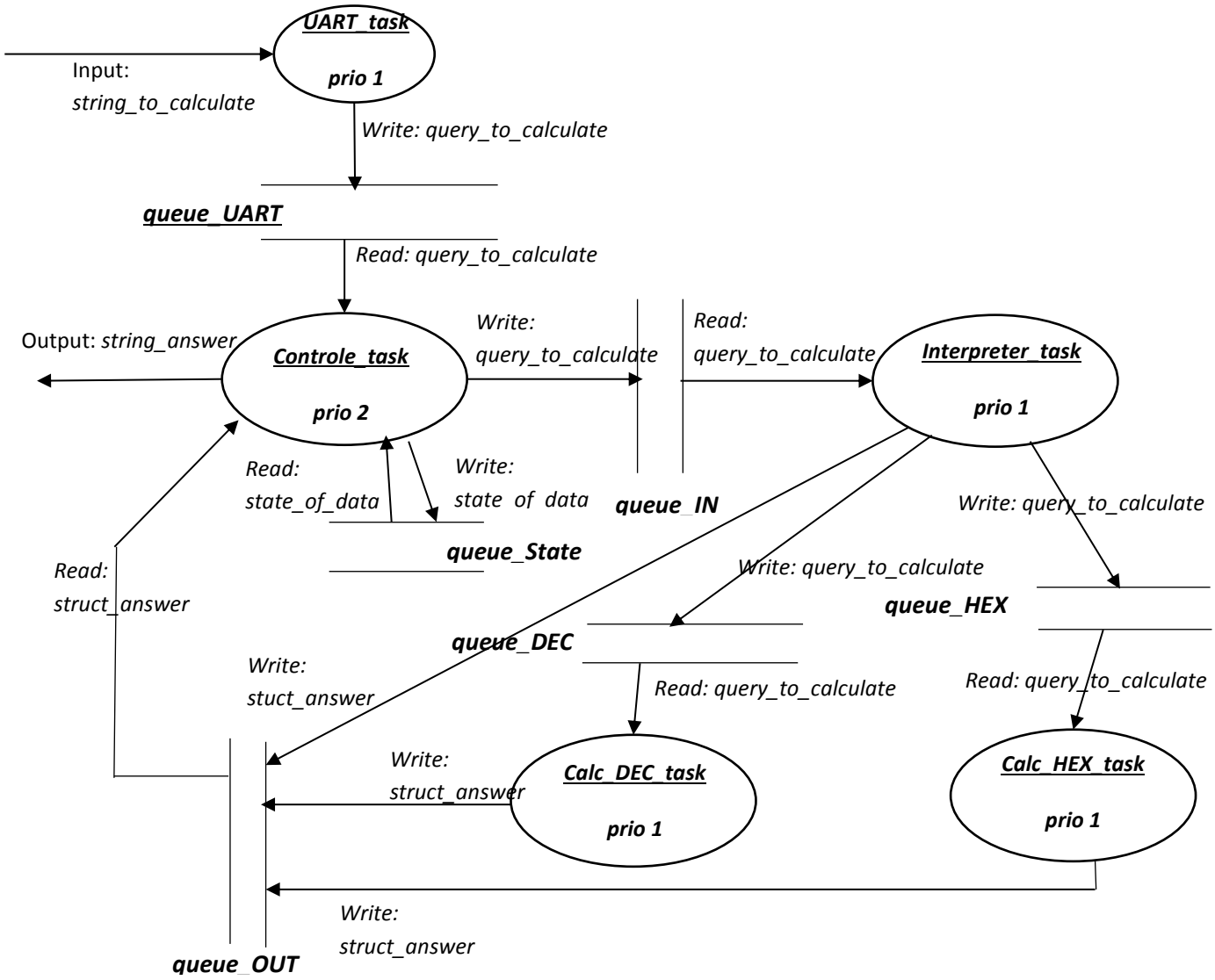
Taak	Hoofdfunctie in Calculator	Gedemonstreerde functionaliteit van het RTOS
UART_task	Deze taak leest de UART-poort, geeft een identificatienummer voor het bericht en stuurt het bericht naar de *Control_task* via *queue_UART.*	• Een taak uitvoeren in een bepaalde periode.
Control_task	Deze taak leest een bericht vanuit de *queue_UART*, neemt de aankomsttijd van het bericht en stuurt het ID van het bericht met de tijd naar de *queue_State*. Vervolgens stuurt ze het bericht verder naar de *queue_IN*. Daarna leest deze taak de *queue_OUT*, controleert de tijd en print het gelezen bericht of een time-out melding.	• Lezen vanuit twee queues. • Timer gebruiken. • Deze taak heeft de hoogste prioriteit en demonstreert de synchronisatie van de taken met behulp van de queues en timer. • Demonstreert het Preemptive algoritme van het RTOS.
Interpreter_task	Deze taak leest berichten vanuit de *queue_IN,* checkt of ze correct zijn ingetypt, converteert ze naar een andere structuur en stuurt ze naar de *queue_DEC* en/of *queue_HEX*, of stuurt een foutmelding over een niet correct bericht naar de *queue_OUT*.	• De taken sturen de berichten tegelijkertijd naar één queue. • Deze taken hebben gelijke prioriteiten en demonstreren het Round-Robin scheduling algoritme[1] (als dit in het RTOS is geïmplementeerd).
Calc_DEC_task	Deze taak leest de berichten vanuit de *queue_DEC,* berekent de query, converteert het getal naar decimaal en stuurt het antwoord naar de *queue_OUT*.	
Calc_HEX_task	Deze taak leest de berichten vanuit de *queue_HEX,* berekent de query, converteert het getal naar hexadecimaal en stuurt het antwoord naar de *queue_OUT*.	

Twee taken van de demoapplicatie 1.0 moeten periodiek worden uitgevoerd. De *UART_task* moet periodiek de UART-poort proberen te lezen en de *Control_task* moet periodiek de tijd checken. De perioden van deze taken werden door de begeleider vastgesteld: voor *UART_task* op 20msec en voor *Control_task* op 500msec (time-out van de COMbricks PA link).

Alle taken van de demoapplicatie 1.0 communiceren met elkaar via de queues *queue_UART*, *queue_IN*, *queue_STATE*, *queue_DEC*, *queue_HEX* en *queue_OUT.* De communicatie tussen de taken is beschreven in het dataflow-diagram voor Calculator, zie schema 7.2.

[1] Zie Begrippenlijst, Bijlage I

Schema 7.2: Calculator 1.0 - laag 1: dataflow-diagram



De uitvoering van de taken van de demoapplicatie 1.0 heeft de volgende volgorde:

- De gebruiker voert een string in voor berekening via een terminal. Deze string bevat een teken dat het numerieke systeem van de input bepaalt (decimaal en/of hexadecimaal), twee getallen, een operand tussen de getallen en het numeriek systeem (decimaal of/en hexadecimaal) waarin hij een antwoord wil krijgen, zie Tabel 7.3.
- *UART_task* leest de input via de UART-poort, geeft een identificatienummer en converteert de inputstring naar de structuur *query_to_calculate*. Deze structuur bestaat uit twee getallen van -2^{31} tot en met $(+2^{31}-1)$ en een andere structuur, *packet_header*. In de structuur *packet_header* staan de operand, een teken voor het numerieke systeem van input, een teken voor het numerieke systeem van output en het ID van het bericht. Tijdens de conversie moeten de andere processen worden geblokkeerd om de integriteit van de input te bewaren.
- Daarna stuurt de *UART_task* de gemaakte structuur naar de *queue_UART.* De *UART_task* heeft een lage prioriteit, maar elke 20msec onderbreekt het systeem de andere taken met dezelfde prioriteit en keert het terug naar deze taak om te proberen de UART-poort te lezen.

Tabel 7.3: Calculator 1.0, string voor berekening

Name of message		***query_to_calculate***						
Type		***String***						
Example		*D356*25 c*						
Type of data in the string	Type of numbers	Number one	Operand	Number two	Space	Type of answer	Enter	
Max size of string	8 bits	11 bytes	8 bits	11 bytes	8 bits	8 bits	8 bits	Total: 216 bits
Possible input	(H) | (h) | (D) | (d) H/h – hexadecimaal; D/d – decimaal;	Max number: **2147483647** in dec **7FFFFFFF** in hex Min number: - **2147483648** in dec - **80000000** in hex	(*) | (/) | (+) | (-)	Max number: **2147483647** in dec **7FFFFFFF** in hex Min number: - **2147483648** in dec - **80000000** in hex	"Space" ASCII: 20	(H) | (h) | (D) | (d) |(C) | (c) H/h – hexadecimaal; D/d – decimaal; C/c – decimaal en hexadecimaal.	"Enter" ASCII: 13	

- De *Control_task* leest de *queue_UART*.
    - Als de *queue_UART* leeg is, wordt de *Control_task* voor 500msec geblokkeerd en geeft ze de processor tijd om andere taken uit te voeren. Als het bericht in *queue_UART* komt, wordt de *Control_task* onmiddellijk gedeblokkeerd.
    - Wanneer het bericht binnen de *queue_UART* komt, neemt de *Control_task* de aankomsttijd van het bericht, maakt een structuur – *state_of_data* en stuurt deze structuur naar de *queue_STATE*. De structuur *state_of_data* bevat het ID van het bericht en de aankomsttijd van het bericht.
    - Daarna stuurt de *Control_task* het bericht – *query_to_calculate*, naar de *queue_IN*.
    - Vervolgens checkt de *Control_task* de *queue_OUT*; als het bericht in de *queue_OUT* staat, wordt het door de *Control_task* gelezen; de tijd van uitvoering wordt gecheckt. De uitvoeringstijd moet korter dan 500msec zijn. Als er geen tijdvertraging is, wordt het bericht geprint; anders print de *Control_task* een foutmelding.
    - De *Control_task* heeft de hoogste prioriteit en na een event (een bericht komt in de *queue_UART* of de 500msec gaan voorbij) wordt deze taak meteen uitgevoerd.
- De *Interpreter_task* leest de *queue_IN*.
    - Het doel van de *Interpreter_task* is de fouten in het bericht te controleren en te bepalen voor welke volgende taak het bericht is bedoeld. Als een gebruiker een fout maakt tijdens het intypen van de query, stuurt de *Interpreter_task* een structuur: *struct_answer*. De structuur *struct_answer* bevat een string met foutmelding of antwoord en het ID van het bericht.
    - Wanneer er geen fouten in het bericht zitten, stuurt de *Interpreter_task* het bericht verder naar de *queue_DEC* en/of de *queue_HEX*, afhankelijk van het gewenste numerieke systeem van output. Het teken voor het numerieke systeem van output staat in de header van het bericht.

    - De *Interpreter_task* heeft een lage prioriteit, dezelfde als de *UART_task*. De *Interpreter_task* leest de *queue_IN* uit, als de queue leeg is dan wordt de *Interpreter_task* geblokkeerd.
- De *Calc_DEC_task* leest de *queue_DEC*.
    - Als de *queue_DEC* leeg is wordt de *Calc_DEC_task* geblokkeerd. Anders wordt het bericht uit de *queue_DEC* gelezen.
    - Het doel van de *Calc_DEC_task* is de query van de gebruiker te berekenen.
    - Daarna checkt de taak het numerieke systeem van de getallen. Als de getallen in hexadecimaal zijn, converteert de *Calc_DEC_task* het antwoordgetal naar decimaal.
    - Vervolgens converteert deze taak het antwoord naar een string en stuurt de antwoordstring met het ID van het bericht naar de *queue_OUT*.
    - Deze taak heeft lage prioriteit, dezelfde als *UART_task*.
- *De Calc_HEX_task* heeft bijna dezelfde functionaliteit als de *Calc_DEC_task*. Het verschil is dat de *Calc_HEX_task* de *queue_HEX* leest en decimale getallen naar hexadecimaal converteert.

In de demoapplicatie 1.0 moesten de taken na elk event (zoals: queue is uitgelezen, bericht is naar queue gestuurd, taak is aan de beurt om uitgevoerd te worden) een bericht over dit event printen. Voor het printen werd de seriële poort gebruikt. Om deze resource te beveiligen moest een Mutex gecreëerd worden. Ik heb de printfuncties ontworpen die door Mutex zijn beveiligd.

Tijdens de Ontwerpfase heb ik de Yourdon-diagrammen voor Calculator 1.0 gemaakt. Ze staan in de Bijlage V. In dit hoofdstuk tijdens de beschrijving van de demoapplicatie 1.0 heb ik de namen van verschillende structuren gebruikt. De beschrijving van deze structuren staan in Bijlage V.

De demoapplicatie 1.0 toont een aantal noodzakelijke functionaliteiten aan. De beschrijving staat in tabel 7.4.

Tabel 7.4: Calculator 1.0. Gebruikte functionaliteit van RTOS

	Functionaliteit		Beschrijving
Eis 1	In de demoapplicatie 1.0 moeten taken geconfigureerd worden	*UART_task; Control_task; Interpreter_task; Calc_DEC_task; Calc_HEX_task;*	Deze taken zijn gecreëerd en geconfigureerd. Elke taak heeft een eigen handle.
Eis 2	De demoapplicatie 1.0 moet het Preemptive algoritme van het RTOS demonstreren	*UART_task; Interpreter_task; Calc_DEC_task; Calc_HEX_task*	Deze taken hebben dezelfde prioriteiten. Ze worden om de beurt uitgevoerd dankzij het Preemptive algoritme.
Eis 3	De demoapplicatie 1.0 moet taken met verschillende prioriteiten bevatten	*Control_task; UART_task; Interpreter_task; Calc_DEC_task; Calc_HEX_task*	De *Control_task* heeft de hogere prioriteit dan de andere taken
Eis 4	De demoapplicatie 1.0 moet periodieke taken bevatten	*UART_task*	Deze taak wordt elke 20msec uitgevoerd.
Eis 5	In de demoapplicatie 1.0 moeten queues geconfigureerd worden	*queue_UART; queue_IN; queue_OUT; queue_STATE; queue_DEC; queue_HEX*	Deze queues zijn gecreëerd en geconfigureerd. Elke queue heeft een eigen handle.
Eis 6	In de demoapplicatie 1.0 moeten functies geïmplementeerd worden zoals Receive from the Queue en Send to the Queue	*UART_task; Control_task; Interpreter_task; Calc_DEC_task; Calc_HEX_task;*	In deze taken zijn de functies geïmplementeerd die de berichten vanuit queues van de demoapplicatie lezen en naar de queues sturen.
Eis 7	In de demoapplicatie 1.0 moeten processen gesynchroniseerd worden met gebruikmaking van Queues	*Control_task: queue_UART en queue_OUT*	De *Control_task* heeft de hoogste prioriteit. Deze taak wordt door de queues voor 500msec geblokkeerd. Daardoor krijgen de taken met lagere prioriteiten processortijd voor de uitvoering.
Eis 8	In de demoapplicatie 1.0 moet een systeemtimer worden gebruikt	*Control_task*	Deze taak neemt de aankomsttijd van het bericht en vergelijkt deze tijd met de tijd van het uitkomen van het bericht. Als het antwoord niet binnen 500msec komt, geeft het systeem een time-outmelding. Daarvoor wordt de systeemtimer gebruikt.
Eis 9	In de demoapplicatie 1.0 moet een time-out van het systeem geconfigureerd worden		
Eis 10	In de demoapplicatie 1.0 moet Mutex geconfigureerd worden		In *Calculator* systeem moet een Mutex gecreëerd worden die de seriële poort beveiligt tijdens het printen.
Eis 11	In de demoapplicatie 1.0 moet een resource door Mutex worden beveiligd		

Voorbereiding op de testfase

Om te kijken of *Calculator 1.0* voldeed aan alle eisen, moest ik het systeem testen. In deze fase maakte ik het testplan. Ik verdeelde de tests in twee categorieën:

1. Tests van de correcte werking van de demoapplicatie 1.0 als rekenmachine: deze tests controleren of de demoapplicatie de correcte berekeningen uitvoert en bij de incorrecte input een foutmelding geeft.
2. Tests van de correcte werking van de demoapplicatie als realtime-systeem: deze tests controleren of de demoapplicatie aan de vastgestelde eisen voldoet.

Het testplan staat in Bijlage VIII. Na de Implementatiefase moesten deze testen worden uitgevoerd. In het hoofdstuk 7.4 staat de beschrijving van de uitvoering van de testen. Volledige resultaten staan in Bijlage IX.

7.3 Iteratie 2: ontwikkeling demoapplicatie 1.0. Implementatiefase

De demoversies van ThreadX en MicroC/OS-II hebben een tijdbeperking van één maand. Daarom besloot ik om het niet-commerciële FreeRTOS te gebruiken om de demoapplicatie te ontwikkelen. In de Implementatiefase implementeerde ik de demoapplicatie 1.0 met FreeRTOS.

FreeRTOS

Ik downloadde FreeRTOS voor Microblaze van http://www.freertos.org/. De software van de COMbricks PA link was in Xilinx IDE ontwikkeld. Daarom was de volgende stap het integreren van FreeRTOS naar deze ontwikkelomgeving. Op de website van FreeRTOS staat een stappenplan voor het creëren van een FreeRTOS-project in Xilinx IDE. Na uitvoering van deze stappen mislukte het bouwen van het nieuwe project. Het bleek dat de scripts van FreeRTOS die de makefile gebruikte, verwezen naar een verouderde versie van Xilinx. Om het FreeRTOS-project in Xilinx IDE versie 14.7 te creëren moest ik de bestanden met de scripts *freertosv210.tcl* en *freertosv210.mld* aanpassen. Het volledige stappenplan voor de integratie van FreeRTOS in Xilinx 14.7 staat in Bijlage VI.

Na het creëren van het nieuwe FreeRTOS-project in Xilinx 14.7 ontwikkelde ik demoapplicatie 1.0.

Voor de realisatie van de demoapplicatie creëerde ik vijf taken: *vUART_task, vControl_task, vInterpreter_task, vCalc_DEC_task* en *vCalc_HEX_task*. De *Control_task* beheert de verzoeken van de gebruiker en de antwoorden van *Calculator*. Als het verzoek of het antwoord binnenkomt, moet deze taak onmiddellijk reageren. Daarom heeft deze taak de hoogste prioriteit. De prioriteiten van de andere taken, *vUART_task, vInterpreter_task, vCalc_DEC_task* en *vCalc_HEX_task*, zijn lager en aan elkaar gelijk. Voor de communicatie van deze taken creëerde ik zes queues, zoals beschreven in het hoofdstuk Ontwerpfase. Een korte beschrijving van de queues staat in Bijlage VII. De queues zijn gecreëerd met API-functie *xQueueCreate()*:

```
xQueueCreate( uxQueueLength, uxItemSize)
```

waarin de eerste parameter de lengte en de tweede parameter de maat van één item in de queue is.

Verder beschrijf ik per taak welke functies en welke configuratie van FreeRTOS ik heb gebruikt, hoe de taak is gesynchroniseerd en welke prioriteit de taak heeft.

vUART_task()

De prioriteit van deze taak is 1. Deze taak was met API-functie `xTaskCreate()` gecreëerd:

```
xTaskCreate( vUART_task, "tUart", 1000, NULL, 2, &xHandle_UART)
```

Hierin is de eerste parameter de pointer naar de functie, de tweede is de naam van de taak, de derde is benodigde stack-omvang, de vierde is de parameter die in de taak kan worden meegegeven, de vijfde is de prioriteit van de taak en de zesde is de handle van de taak. De omvang van de stack van de taken bepaalde ik eerst ongeveer; daarna corrigeerde ik de omvang met behulp van de diagnosetaak, die het gebruik van de stack per taak weergeeft.

- ***Periodiek uitgevoerd***

Deze taak wordt periodiek uitgevoerd: elke 20msec na de laatste uitvoering (als er geen taken zijn in toestand *Ready* met hogere prioriteit). Daarvoor wordt de FreeRTOS-functie *vTaskDelayUntil()* gebruikt.

```
vTaskDelayUntil( &xRunTime, 20/portTICK_RATE_MS );
```

Hierin is de eerste parameter de tijd van de laatste uitvoering van de functie en de tweede parameter een periode van uitvoering. De tweede parameter moet in kernel ticks zijn. Maar met behulp van constante *portTICK_RATE_MS* kunnen milliseconden naar kernel ticks worden geconverteerd en andersom. *portTICK_RATE_MS* is gelijk aan (1000/*configTICK_RATE_HZ*). *configTICK_RATE_HZ* is een gespecificeerde frequentie van de kernel ticks (in Hz) in *FreeRTOSConfig.h*. De frequentie kan worden veranderd via Xilinx:

Board Support Package Settings >> freertos >> systmr_interval;

Voor het gebruik van de functie *vTaskDelayUntil()* moet de configuratiefile van FreeRTOS handmatig worden aangepast. In het bestand *freertos_v2_1_0.tcl* moet de parameter *INCLUDE_vTaskDelayUntil* op 1 worden gezet:

```
xput_define $config_file "INCLUDE_vTaskDelayUntil" "1"
```

- ***Luisteren naar de UART-poort***

Deze taak luistert de hele time-slicing naar de UART-poort. Daarvoor wordt de Microblaze-functie gebruikt: *XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR)*. Hierin is *XPAR_RS232_UART_1_BASEADDR* het adres van de UART-poort. Deze functie retourneert een byte die in de UART-poort komt. Het nadeel van deze functie is dat de applicatie gedurende de hele timeslice op deze functie hangt als er geen input is geweest.

- ***Lengte van het bericht controleren en het bericht naar de structuur converteren***

Wanneer een teken de seriële poort binnenkomt, schort deze taak alle andere taken op met de FreeRTOS-functie *vTaskSuspendAll()*. Vervolgens leest de *UART_task* het verzoek vanuit de seriële poort. Tijdens het lezen controleert de *UART_task* meteen de lengte van de input — die mag niet meer dan 24 tekens bevatten — en converteert de gelezen tekens naar de structuur *QUERY_TO_CALCULATE*. Daarna worden de andere taken gedeblokkeerd met de FreeRTOS-functie *xTaskResumeAll()*.

- ***Nieuw verzoek naar de queue_UART sturen***

Ten slotte stuurt de *UART_task* de structuur QUERY_TO_CALCULATE naar de *queue_UART* met de API-functie *xQueueSendToBack().*

```
xQueueSendToBack(xQueue_UART, &input, xTicksToWait);
```

Hierin is de eerste parameter de handle van queue, de tweede de pointer naar te verzenden item en de derde de wachttijd. Als de queue vol is, wordt de taak voor de wachttijd geblokkeerd en als binnen deze tijd de queue wordt vrijgemaakt, krijgt de taak CPU-tijd om een bericht naar de queue te sturen. Dit gebeurt als er geen taken met hogere prioriteiten in de toestand *Ready* zijn. Als de parameter *xTicksToWait 0* is, wordt de taak niet geblokkeerd; als deze parameter is gedefinieerd als *portMAX_DELAY*, gaat de taak oneindig lang op de queue wachten. In de demoapplicatie 1.0

configureerde ik de laatste parameter als 0. Want de *Control_task* met hoogste prioriteit leest de *queue_UART*, dus de *queue_UART* kan niet meer dan één bericht bevatten.

vControl_task()

De prioriteit van deze taak is gelijk aan 2. Deze taak is de ontvanger van de berichten vanuit drie queues: *queue_UART, queue_STATE* en *queue_OUT*.

- ***De queue_UART lezen***

De *Control_task* leest een bericht vanuit de *queue_UART*. Hiervoor wordt de functie `xQueueReceive(xQueue_UART, &input, xTicksToWaitQueue)` gebruikt. In deze functie is de eerste parameter de handle van queue*,* de tweede is de pointer naar het ontvangen item en de derde is de wachttijd. Als er geen berichten in de queue zijn, wordt de taak voor de wachttijd geblokkeerd. In de *Control_task* is de wachttijd gelijk aan 500msec, want elke 500msec moet de *Control_task* de time-out checken.

- ***Status van berichten bewaren***

Wanneer het bericht in de *queue_UART* komt, wordt de *Control_task* gedeblokkeerd. Ze wijst de inkomende bericht een uniek *ID* toe. Daarna onthoudt de *Control_task* de aankomsttijd van de berichten met behulp van API-functie `xTaskGetTickCount()`, maakt een structuur *STATE_OF_DATA* met de *ID* en aankomsttijd van het bericht erin, en stuurt deze structuur naar de *queue_STATE*.

- ***Time-out checken***

Elke 500msec neemt de *Control_task* de actuele tijd op en checkt het verschil tussen de actuele tijd en de aankomsttijd van de berichten die al in de *queue_STATE* zijn bewaard. Daarvoor gebruikt de *Control_task* de API-functie:

```
xQueuePeek(xQueue_STATE, &msg_state, xTicksToWait );
```

Deze functie leest het item vanuit de queue, maar het item wordt niet uit de queue verwijderd. In deze functie is de eerste parameter de handle van de queue, de tweede de pointer naar het ontvangen item en de derde de wachttijd. De *Control_task* moet niet door deze queue geblokkeerd worden, daarom is de wachttijd gelijk aan 0.

Als het verschil tussen de aankomsttijd van een bericht en de actuele tijd meer is dan 500msec, print het systeem een time-out-foutmelding en het bericht wordt uit de *queue_State* verwijderd.

- ***Nieuw verzoek naar de queue_IN sturen***

Wanneer het bericht in de *queue_UART* komt, leest *Control_task* het bericht, neemt de aankomsttijd op, bewaart de *ID* van het bericht met de aankomsttijd in de *queue_STATE* en stuurt het bericht naar de *queue_IN.*

- ***De queue_OUT lezen***

De Control-taak leest de *queue_OUT.* Als het antwoord in de *queue_OUT* komt, vergelijkt de *Control_task* de aankomsttijd van het antwoord en de aankomsttijd van het bericht met dezelfde *ID* vanuit de *queue_STATE*. Als het tijdsverschil kleiner is dan 500msec, print de *Control_task* een antwoord. Anders print de *Control_task* een foutmelding. Het bericht in ieder geval wordt uit de *queue_State* verwijderd.

vInterpreter_task()

Deze taak heeft prioriteit 1.

- ***De queue_OUT lezen***

Deze taak is de *Receiver* van de *queue_IN.* Als de *queue_IN* leeg is, is de *Interpreter_task* geblokkeerd. Wanneer het bericht binnen de *queue_IN* komt en er geen taken met hogere prioriteiten in de toestand *Ready* zijn, leest de *Interpreter_task* het bericht vanuit de queue.

- ***Het verzoek om calculatie checken***

Daarna roept de Interpreter-taak de functie *check_string_to_calculate()* aan.

```
QUERY_TO_CALCULATE check_string_to_calculate(STRUCT_QUERY query_struct)
```

De input van deze functie is *STRUCT_QUERY*. Deze functie controleert het bericht op inputfouten en converteert daarna de *STRUCT_QUERY* naar *QUERY_TO_CALCULATE*. De structuur *QUERY_TO_CALCULATE* is de returnwaarde van deze functie.

- ***Het verzoek voor calculatie naar de queue_DEC of/en queue_HEX sturen***

Vervolgens stuurt de *Interpreter_task*, als er in de input geen fouten zitten, de *QUERY_TO_CALCULATE* naar *queue_DEC* of/en *queue_HEX*, afhankelijk van `query_to_calc.header.output_radix.` Anders stuurt de *Interpreter_task* een foutmelding naar de *queue_OUT*.

vCalc_DEC_task () en *vCalc_HEX_task()*

Deze taken hebben prioriteit 1. Deze taken zijn *Receiver*s van respectievelijk de *queue_DEC* en de *queue_HEX*. Deze taken blijven in geblokkeerde status tot er een bericht in de queue komt. Deze taken voeren de berekeningen uit. *Calc_DEC_task* berekent decimale getallen, *Calc_HEX_task* berekent hexadecimale getallen. Na de berekening sturen deze taken de antwoorden naar de *queue_OUT*.

7.4 Iteratie 2: ontwikkeling demoapplicatie 1.0. Testfase

In de Testfase voerde ik de testen uit die ik in de Ontwerpfase had gepland. De volledige beschrijving van de testresultaten met de printscreen van de output staat in Bijlage VIII.

Eerst testte ik de correcte werking van de demoapplicatie 1.0 als rekenmachine.

Test 1 Correcte berekeningen

> Tijdens deze test werden de verzoeken ingevoerd die in tabel 1 in Bijlage VIII staan. De testresultaten waren correct.

Test 2 Foutmeldingen bij incorrecte input door de gebruiker

> Tijdens deze test werden de testcases gebruikt die in tabel 2 in Bijlage VIII staan. De testresultaten waren correct.

Daarna voerde ik de tests uit die de eisen aan de functionaliteit van het RTOS controleerden.

Test 3 Controle op de correcte configuratie van de taken

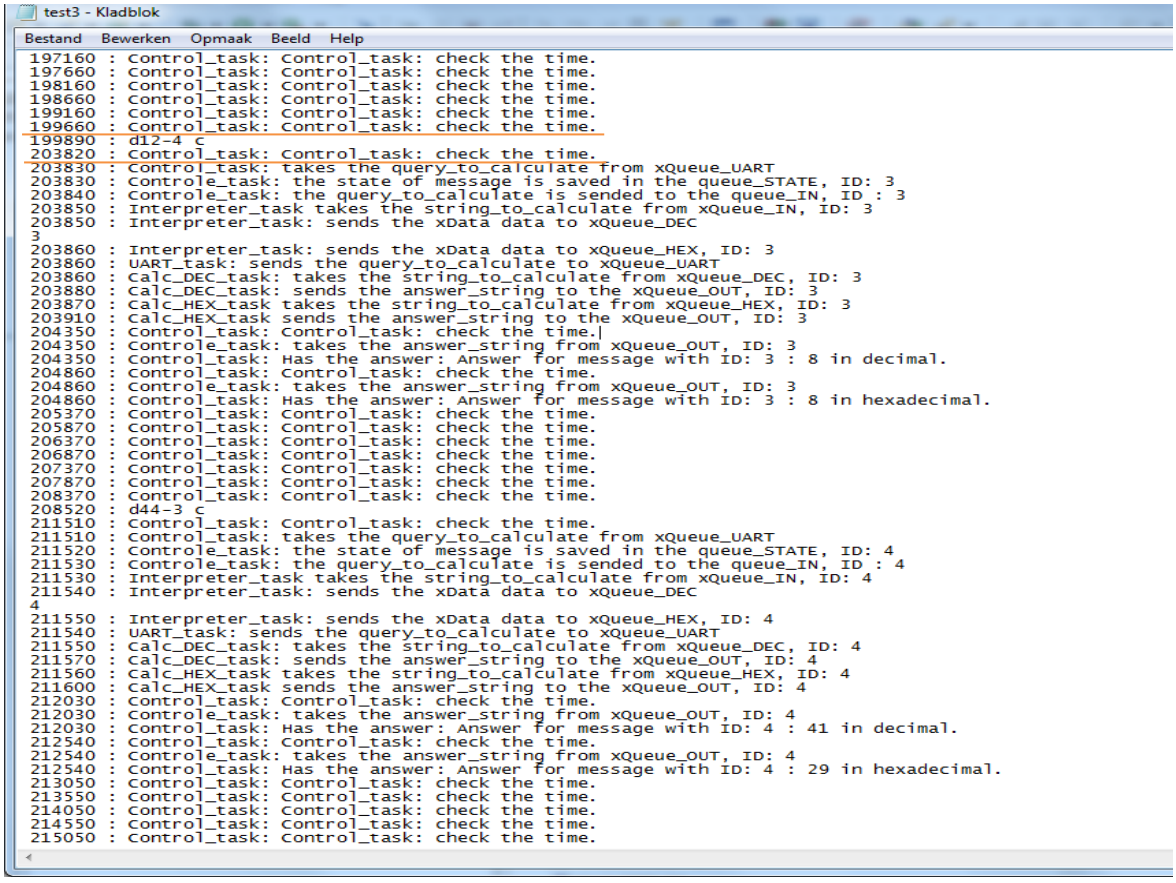
Hiervoor gebruikte ik de testcases die in Tabel 3 in Bijlage VIII staan. De resultaten van de tests waren correct.

Test 4 Test van het Preemptive algoritme en verschillende prioriteiten

Hiervoor gebruikte ik de testcases die in Tabel 4 in Bijlage VIII staan. Het resultaat van testcase 4.1 was correct.

Het resultaat van testcase 4.2 was niet goed: de output staat in printscreen 7.1. Verwacht was dat de *Control*_task elke 500msec de uitvoering van andere taken zou onderbreken om de tijd te checken. De oorzaak dat het systeem niet binnen de deadline bleef, lag in de implementatie van *UART_task*. Deze taak wachtte tot het eind van de input, andere taken bevonden zich in de geblokkeerde status.

Printscreen 7.1: Printscreen van test: synchronisatie van taken met verschillende prioriteiten



Test 5 Controle op de uitvoering van een taak in een bepaalde periode

Bij de opstelling van deze testcase moet de *UART_task* de melding "*Current time.UART_task: is active*" printen wanneer ze runt. Maar deze opstelling is onmogelijk, want in de *UART_task* is de functie *XUartLite_RecvByte()* gebruikt. Bij de eerste uitvoering roept de taak deze functie aan en blijft daarin hangen totdat er input binnenkomt of de slicing-time op is. Na 20msec komt de *UART_task* terug binnen deze functie. Om de periodieke uitvoering te checken heb ik de functie *XUartLite_RecvByte()* verwijderd. De resultaten van deze test waren correct.

De resultaten van de volgende tests waren correct:

Test 6. Controle op de correcte configuratie van de queues.

Test 7. Controle op het gebruik van de functies Receive from the Queue, Send to the Queue.

Test 8. Controle op de synchronisatie van de taken met gebruikmaking van queues.

Test 9. Controle op de reactie van het systeem op de niet-gehaalde deadline.

Test 10. Controle op de correcte configuratie van Mutex.

Test 11. Controle op de beveiliging van de UART-poort door Mutex.

De resultaten van de testcase 11.1 was correct. De testcase 11.2 voerde ik niet uit, want ik kon geen incorrecte configuratie voor Mutex bedenken. De functie die Mutex creëert, heeft geen inputparameters die incorrect zouden kunnen worden geconfigureerd.

Test 12. Stresstest.

De resultaten van de stresstest waren onvoldoende. De demoapplicatie las niet alle data uit het bestand. De demoapplicatie miste het eind van de input en bleef hangen in afwachting van het eind van de input.

In de volgende iteraties probeerde ik de oplossingen te vinden om de fouten van de testcase 4.2 en Stresstest te verbeteren.

8. Implementatie seriële interrupt

In deze iteratie heb ik geprobeerd de demoapplicatie 1.0 te verbeteren op basis van de analyse van de onvoldoende testresultaten. In de Analysefase analyseerde ik de onvoldoende resultaten en bedacht ik een oplossing daarvoor die tegemoet zou komen aan de eisen aan de demoapplicatie. Daarna paste ik in de ontwerpfase de oplossing in de Yourdon-diagrammen aan. In de Implementatiefase implementeerde ik deze oplossing.

8.1 Iteratie 3: implementatie seriële interrupt. Analysefase

De testfase van de tweede iteratie leverde onvoldoende resultaten op bij twee tests: de stresstest en de periodieke uitvoering van de *Control_task*. De oorzaak van de onvoldoende resultaten ligt in de implementatie van *UART_task*. Wanneer de *UART_task* het begin van de input krijgt, blokkeert ze andere taken tot het eind van de input binnenkomt. Daardoor wordt de *Control_task* niet periodiek. Bij snelle input van een grote hoeveelheid data kan het systeem een deel daarvan verliezen. Om deze problemen te voorkomen kan in *Calculator* een seriële interrupt worden geïmplementeerd.

De grootste voordelen van de seriële interrupt zijn:

- lezen van de UART-poort neemt alleen CPU-tijd als er data binnenkomen;
- de interrupt heeft in alle RTOS'en standaard de grootste prioriteit. Daardoor worden, als er data in de seriële poort komen, alle andere taken onmiddellijk geblokkeerd en komt de input binnen.

Bij de implementatie van de seriële interrupt wordt de *UART_task* door de interrupt-routine vervangen en de functionaliteit van de *UART_task* door de interrupt-routine overgenomen. De functionaliteit van de interrupt-routine moet zo compact mogelijk zijn. In *Calculator* 1.0 werd het verzoek van de gebruiker gedeeltelijk gecontroleerd binnen de *UART_task*. De *UART_task* omvatte een groot aantal functionaliteiten:

- de UART-poort lezen;
- als er input binnenkomt, andere taken blokkeren en wachten op het eind van de input;
- een string maken;
- lengte en getallen van het verzoek controleren;
- het verzoek naar de *queue_UART* sturen.

Bij de implementatie van de seriële interrupt minimaliseerde ik de functionaliteit van de interrupt-routine. Er bleven twee functies over:

1. de UART-poort lezen;
2. het inkomende teken naar de *queue_UART* sturen.

De andere functionaliteit van de *UART_task* werd in de demoapplicatie 1.0 overgenomen door de *Control_task.*

De analysefase resulteerde ik in de volgende eisen:

Eis 1. In de demoapplicatie 1.1 moet een seriële interrupt worden geïmplementeerd.

Eis 2. De interrupt-routine moet als minimale functionaliteit bevatten:

- de UART-poort lezen;
- het inkomende teken naar de *queue_UART* sturen.

Eis 3. De *Control_task* moet extra functionaliteit bevatten:

- een string maken;
- lengte en tekens van het verzoek controleren.

8.2 Iteratie 3: implementatie seriële interrupt. Ontwerpfase

In de Analysefase stelde ik de eisen aan de demoapplicatie met seriële interrupt vast. In deze fase bedacht ik de functies die ik moest implementeren om *Calculator* aan die eisen te laten voldoen.

Eis 1. In de demoapplicatie moet een seriële interrupt worden geïmplementeerd.

Daarvoor moest ik de volgende functies realiseren in de code van de demoapplicatie 1.1:

1. een functie die de seriële interrupt initialiseert. Binnen deze functie moest ik de interrupt handle creëren. Daarna moest ik deze handle als interrupt handle installeren en ten slotte de interrupt inschakelen;
2. de functie die de interrupt-routine bevat.

Eis 2. De interrupt-routine moet als minimale functionaliteit bevatten:

- de UART-poort lezen;
- het inkomende teken naar de *queue_UART* sturen.

Om aan deze eis te voldoen moet de functie die de interrupt afhandelt de volgende functies aanroepen:

1. `XUartLite_ReadReg()`: Xilinx-functie die de UART-poort leest;
2. `xQueueSendFromISR()`: FreeRTOS-functie die een bericht vanuit de interrupt-routine naar de queue stuurt.

Eis 3. De *Control_task* moet extra functionaliteit bevatten:

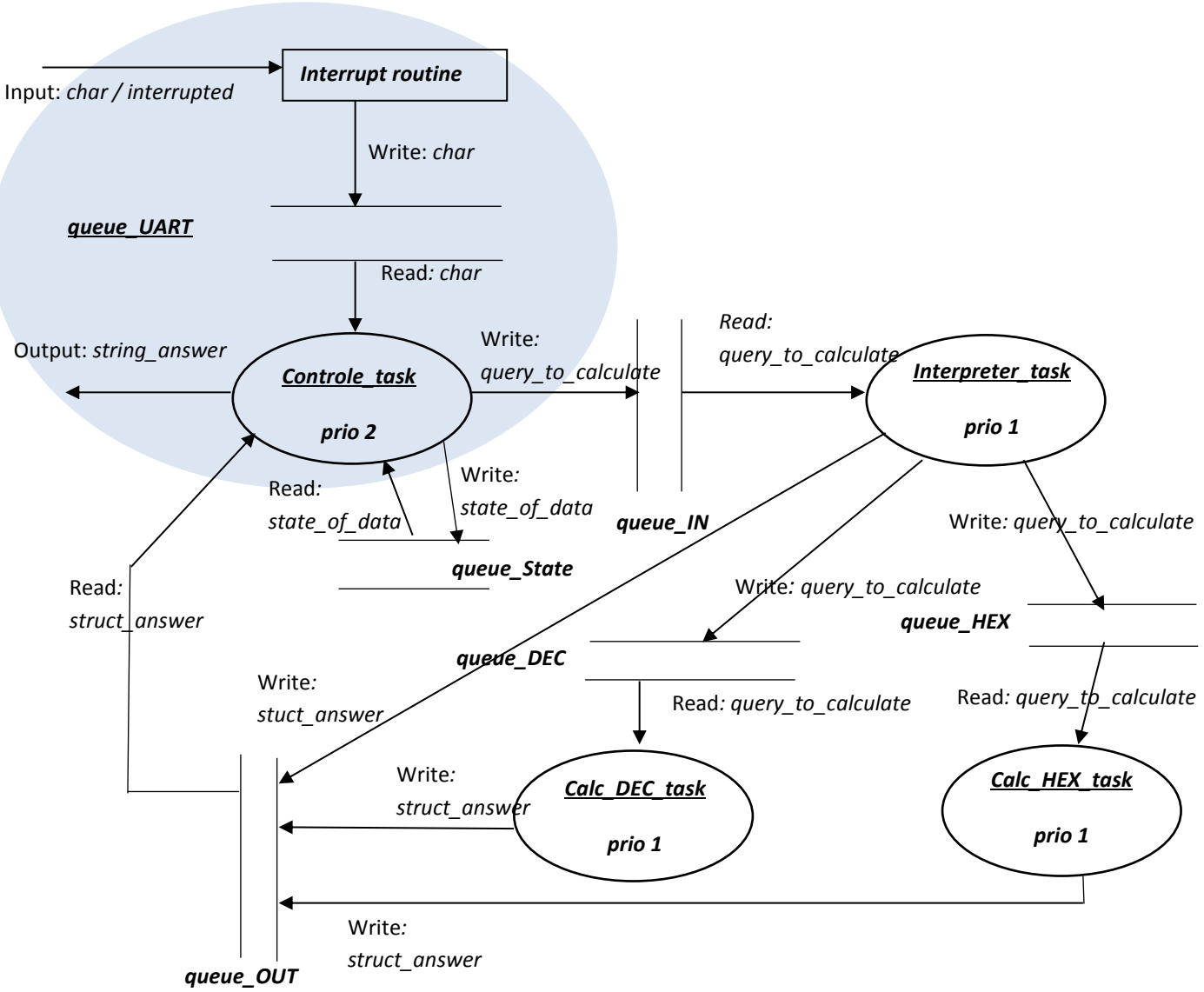
- een string maken;
- lengte en tekens van het verzoek controleren.

Om aan deze eis te voldoen moest ik in de *Control_task* van de demoapplicatie 1.1 de volgende functies implementeren:

1. een functie die de structuur *query_to_calculate* van de ingelezen tekens uit de *queue_UART* maakte. In deze functie moest ik ook de controle op correcte input implementeren;
2. beveiliging van deze functie: ofwel door de API-functie `vTaskSuspendAll()` (deze functie schort alle taken in de applicatie op, maar de interrupt blijft ingeschakeld), ofwel doordat de prioriteit van de *Control_task* de hoogste zou zijn.

In deze fase voerde ik de veranderingen in de demoapplicatie ook in de Yourdon-diagrammen door. Door de implementatie van de seriële interrupt kwam de *UART_ta*sk te vervallen en veranderde het *dataflow*-diagram van Calculator 1.0, zie schema 6.1 De *Control_task* kreeg de extra functionaliteit, zie Bijlage X.

Schema 8.1: Demoapplicatie 1.1 - laag 1: dataflow-diagram



8.3 Iteratie 3: implementatie seriële interrupt. Implementatiefase

In deze fase moest ik de functies implementeren die ik in de Analysefase had bepaald.

1. Voor de functie die de interrupt handle creëert, gebruikte ik de Xilinx-functie:

   ```
   XUartLite_SetRecvHandler( XUartLite *, XUartLite_Handler, void *);
   ```

   Deze functie bepaalt de handle die na een event wordt aangeroepen. De input van deze functie is een pointer naar de XUartLite-variabele, een pointer naar de functie die de interrupt behandelt en de bovenste laag van de callback-referentie die wordt teruggegeven als de callback-functie wordt aangeroepen.

2. Voor de functie die de XuartLite_Handler pointer als interrupt handle installeert, gebruikte ik de FreeRTOS-functie:

*BaseType_t xPortInstallInterruptHandler(XPAR_INTC_0_UARTLITE_1_VEC_ID, XInterruptHandler, void *)*

Deze functie maakt de connectie tussen de ID van de interrupt source en de handle. De input van deze functie is de ID van de interrupt source, de pointer naar de interrupt handle en de callback-referentie. Deze functie retourneert 1 als de connectie correct is gemaakt, anders 0.

3. De functie die de interrupt inschakelt, roept twee functies aan.
   De eerste functie is een Xilinx-functie:

   XUartLite_EnableIntr(XPAR_RS232_UART_1_BASEADDR)

   Deze functie activeert de device interrupt. De input is het base adres van de device.
   De tweede functie is een FreeRTOS-functie:

   vPortEnableInterrupt(XPAR_INTC_0_UARTLITE_1_VEC_ID)

   Dit is een API-functie die de interrupt in de interrupt controller activeert. De inputparameter van deze functie is de ID van de interrupt source.
   De constanten *XPAR_RS232_UART_1_BASEADDR en XPAR_INTC_0_UARTLITE_1_VEC_ID* moeten in het bestand *xparameters.h* worden gedefinieerd. Tijdens de implementatie bleek dat de source ID - *XPAR_INTC_0_UARTLITE_1_VEC_ID* niet was gedefinieerd. De parameters die in het bestand *xparameters.h* zijn gedefinieerd, zijn afhankelijk van de FPGA-configuratie van Microblaze. Dus kon de interrupt niet alleen softwarematig worden geconfigureerd; er moest een hardwarekoppeling worden gemaakt. Dit kon alleen een andere medewerker doen die bezig was met de FPGA-configuratie van Microblaze. De implementatie van de interrupt werd daarmee afhankelijk van de beschikbaarheid van de collega die heel druk bezig met andere projecten was. Daarom besloot ik om een taak te maken die een emulatie van de interrupt zou kunnen zijn.

9. Ontwikkeling demoapplicatie 2.0

Tijdens tweede iteratie was de demoapplicatie 1.0 ontwikkeld. De demoapplicatie 1.0 was de *Calculator* die de basisset van functies van FreeRTOS aantoonde. Bij het testen van de applicatie bleek dat de *Calculator* tijdens de uitvoering niet de deadline haalde. De oorzaak lag in de implementatie van *UART_task* die naar de seriële poort luisterde. Om deze te verbeteren probeerde ik in de derde iteratie een seriële interrupt te integreren. Dat lukte niet, want de interrupt kon ik niet alleen softwarematig implementeren. Het doel van de vierde iteratie was om een alternatieve oplossing voor implementatie van de *UART_task* te vinden. Bovendien besloot ik na het gesprek met mijn begeleider om de functionaliteit van de demoapplicatie uit te breiden.

9.1 Iteratie 4: ontwikkeling demoapplicatie 2.0. Analysefase

Het doel van deze fase was de eisen voor de demoapplicatie 2.0 vast te stellen. Versie 2.0 is een uitbreiding van versie 1.0. Daarom gelden de eisen die voor versie 1.0 waren vastgesteld, ook voor versie 2.0; ze staan in tabel 9.1.

Tabel 9.1: Eisen aan de demoapplicatie 2.0

	Eis
Taken	
Eis 1	In de demoapplicatie 2.0 moeten taken geconfigureerd worden.
Eis 2	De demoapplicatie 2.0 moet het Preemptive algoritme van het RTOS demonstreren.
Eis 3	De demoapplicatie 2.0 moet taken met verschillende prioriteiten bevatten.
Eis 4	De demoapplicatie 2.0 moet periodieke taken bevatten.
Queues	
Eis 5	In de demoapplicatie 2.0 moeten queues geconfigureerd worden.
Eis 6	In de demoapplicatie 2.0 moeten functies worden geïmplementeerd zoals Receive from the Queue, Send to the Queue.
Eis 7	In de demoapplicatie 2.0 moeten processen gesynchroniseerd worden met gebruikmaking van Queues.
Configuratietimer	
Eis 8	In de demoapplicatie 2.0 moet de systeemtimer worden gebruikt.
Eis 9	In de demoapplicatie 2.0 moet time-out van het systeem geconfigureerd worden.
Mutex	
Eis 10	In de demoapplicatie 2.0 moet Mutex geconfigureerd worden.
Eis 11	In de demoapplicatie 2.0 moet een resource worden beveiligd door middel van Mutex.

In de *UART_task* van de demoapplicatie 1.0 was de Xilinx-functie *XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR)* gebruikt om naar de seriële poort te luisteren. De applicatie hangt de hele slicing-time in deze functie. Als er input binnenkomt, hangt de demoapplicatie tot het eind van de input. Deze implementatie van de *UART_task* had de volgende onwenselijke gevolgen:

- CPU-tijd wordt niet effectief gebruikt;
- periodieke taken worden niet periodiek;
- als omvangrijke data heel snel binnenkomen, mist het systeem een deel van de data.

De oplossing van deze problemen lag in de configuratie van een seriële interrupt of in wijziging van de implementatie van de taak op zo'n manier dat de taak een interrupt simuleert. Hieruit leidde ik de eerste extra eis aan de demoapplicatie 2.0 af.

Eis 12. De implementatie van de taak die naar de seriële port luistert, moet de interrupt simuleren.

FreeRTOS heeft diagnosefuncties. Door aanroeping van deze functies kan het systeem de toestand van de taak, vrije plekken in de queues, door de taken gebruikte stack en CPU-tijd, huidige

prioriteiten van de taken en de runtime van de taken laten zien. Gebruik van deze functies is noodzakelijk om het gedrag van het systeem tijdens de ontwikkeling te controleren. Deze functies moest ik in de demoapplicatie 2.0 toevoegen.

Eis 13. De demoapplicatie 2.0 moet gebruikmaken van diagnostische functies als Task state List en Queue state.

De demoapplicatie 1.0 gebruikte de FreeRTOS-timerfunctie om de tijd bij te houden. De variabele die de ticks van het systeem bewaart, is van het type *unsigned long*. De timer wordt gereset als $(2^{32} - 1)$ ticks voorbij zijn. Dat is na ongeveer 49 dagen. De productie van PROCENTEC wordt in de automatiseringsindustrie gebruikt en moet continu werken op lange termijn; 49 dagen is te kort voor de tijdhoudervariabele. Daaruit vloeide voort:

Eis 14. In de demoapplicatie 2.0 moet een softtimer worden geïmplementeerd, waarin de variabele die de tijd bewaart wordt uitgebreid tot het type *unsigned long long* $(2^{64} - 1)$.

In de demoapplicatie 1.0 werden de taken door de queues gesynchroniseerd. Sommige taken lezen de berichten uit twee queues die aan het synchronisatieproces moeten deelnemen. Daarvoor moet een set van queues gemaakt worden. In FreeRTOS bestaat een functie waarmee een set van queues kan worden gecreëerd. Het event binnen de set kan in de demoapplicatie 2.0 worden gebruikt als synchronisatietool.

Eis 15. In de demoapplicatie 2.0 moet een set van queues worden gecreëerd om de taken te synchroniseren.

De begeleider vanuit het bedrijf wilde de functionaliteit van de demoapplicatie uitbreiden. In de COMbricks PA link is de *Syslog*-functie geïmplementeerd. Deze functie print, afhankelijk van de configuratie, de meldingen van systeem. Dankzij deze functionaliteit heeft de gebruiker de mogelijkheid om de output te configureren, wat noodzakelijk is om het systeem te debuggen als er een probleem optreedt.

Eis 16. In de demoapplicatie 2.0 moet de systeemdiagnose op een seriële poort worden geïmplementeerd. Het gaat om een soort logbericht waarin staat wat waar en wanneer is gebeurd tijdens de uitvoering van het programma. In het bericht moet staan:

- de ID van het proces;
- het soort event;
- de tijd.

Deze berichten kunnen met een seriële terminal (zoals Hyperterminal) worden ontvangen en gelogd.

In de software van COMbricks PA link is één functie geïmplementeerd die voor het printen van output verantwoordelijk is. In de demoapplicatie 1.0 was ook een functie geïmplementeerd die de output print. Maar dit was geen efficiënt gebruik van CPU-tijd, want het printen van output kostte tijd. Het creëren van een aparte queue voor outputberichten is de meest efficiënte oplossing daarvoor, want er is weinig tijd nodig om het bericht naar de queue te sturen. Daarna kan, als het systeem niet met belangrijke taken bezig is, een aparte taak met laagste prioriteit deze queue lezen en de outputberichten printen.

Eis 17. De demoapplicatie 2.0 moet een aparte queue bevatten om de outputberichten te verzamelen. Daarnaast moet een taak worden geïmplementeerd die deze berichten uitprint als de processor vrij is.

De gedetailleerde opstelling van de demoapplicatie werd vervolgens in de Ontwerpfase bedacht op basis van de eisen.

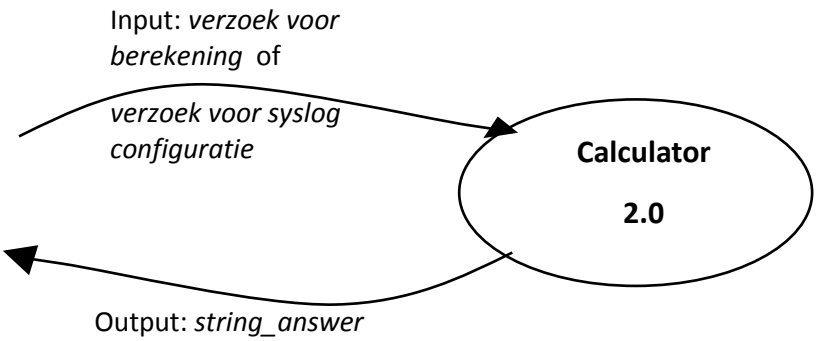
Op basis van deze systeemeisen ontwierp ik in de volgende fase de demoapplicatie 2.0.

9.2 Iteratie 4: ontwikkeling demoapplicatie 2.0. Ontwerpfase

In deze fase werkte ik de nieuw vastgestelde eisen aan de demoapplicatie 2.0 uit. Deze eisen veranderden de logica van de demoapplicatie en moesten terugkomen in de systeemdiagrammen. Daarna bedacht ik de testcases om te checken of de demoapplicatie 2.0 aan de eisen voldeed.

De demoapplicatie 2.0 was een uitgebreide variant van de demoapplicatie 1.0. Opnieuw moest een *Calculator* worden gecreëerd, maar de functionaliteit van het systeem werd uitgebreid door eis 16. Door deze eis kan de gebruiker ook configureren wat voor diagnose naar de seriële port van de terminal wordt gestuurd. De uitgebreide functionaliteit van de demoapplicatie staat in schema 9.1.

Schema 9.1: Contextdiagram – Calculator 2.0



In de vorige fase had ik de eisen aan de demoapplicatie 2.0 vastgesteld. Het doel van deze fase was om deze eisen in het systeem te ontwerpen. De eisen 1 tot en met 11 vormden een overlap met de demoapplicatie 1.0. Hier beschrijf ik alleen de uitbreiding van de demoapplicatie.

Eis 12. De implementatie van de taak die naar de seriële poort luistert, moet de interrupt simuleren.

Om deze eis in de demoapplicatie te realiseren, moest ik de logica van de *UART_task* veranderen. Ten eerste mocht de taak niet hangen als er geen berichten in de seriële poort kwamen. Ten tweede moest ik het creëren van de query-structuur uit deze taak verwijderen. De nieuwe implementatie van de *UART_task* moet periodiek de seriële poort checken en die, als er geen input binnenkomt, meteen blokkeren. Als er input binnenkomt, moet de taak de input naar de *queue_UART* sturen en de poort weer blokkeren voor 20msec. De nieuwe implementatie van de *UART_task* lost twee problemen op:

- CPU-tijd wordt niet effectief gebruikt;
- de periodieke taken worden niet periodiek.

Het probleem dat tijdens de stresstest voorkwam, kon alleen met configuratie van de seriële interrupt worden opgelost. Want de interrupt reageert op het event in de seriële poort, namelijk de binnenkomende input, maar in de nieuwe implementatie reageert de *UART_task* nog steeds op het time-event, namelijk het verstrijken van 20msec.

Het flowdiagram voor de *UART_task* werd opnieuw gemaakt, zie Bijlage X.

Eis 13. De demoapplicatie 2.0 moet gebruikmaken van diagnostische functies als Task state List en Queue state.

Om deze eis in de demoapplicatie te realiseren creëerde ik een nieuwe taak, de *Diagnose_task*, die de FreeRTOS-diagnosefuncties aanroept. Deze taak moet periodiek worden uitgevoerd en de toestanden van het systeem laten zien.

Eis 14. In de demoapplicatie 2.0 moet een softtimer geïmplementeerd worden om de variabele die de tijd bijhoudt, uit te breiden tot type *unsigned long long*.

Om de demoapplicatie aan deze eis te laten voldoen, moet een softtimer in de demoapplicatie gecreëerd worden. De callback-functie van de softtimer moet zorgen dat de variabele die de systeemticks telt, niet overloopt.

Eis 15. In de demoapplicatie 2.0 moet een set van queues worden gecreëerd om de taken te synchroniseren.

Daarvoor moet de FreeRTOS-functie worden gebruikt die een set van queues creëert. Deze functie wordt gerealiseerd in de taken die meer dan één queue lezen*: Control_task* en *Syslog_task*.

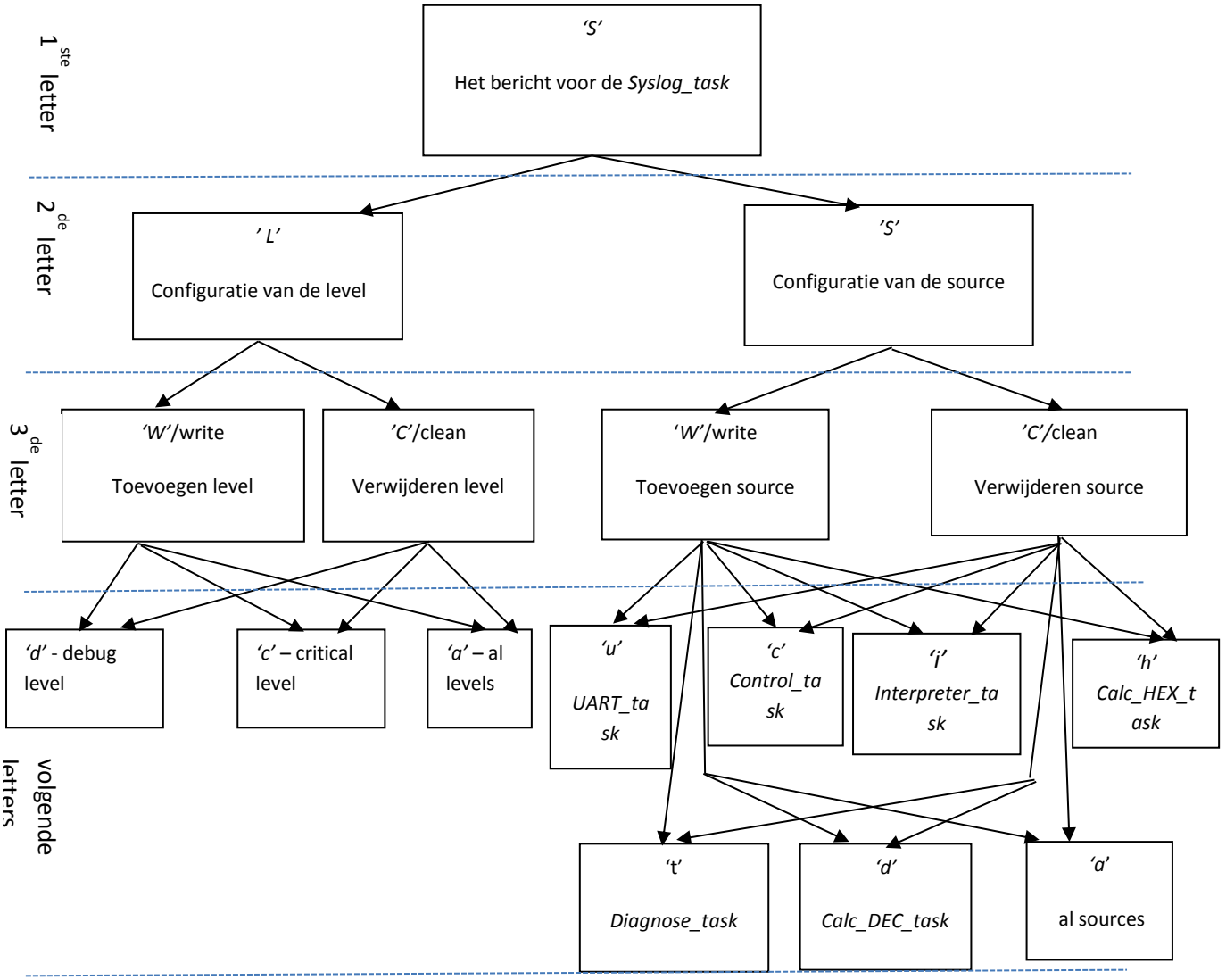
Eis 16. In de demoapplicatie 2.0 moet de systeemdiagnose op een seriële poort worden geïmplementeerd, voor logberichten die bevatten:

- de ID van het proces;
- het soort event;
- de tijd.

Deze berichten kunnen met een seriële terminal (zoal Hyperterminal) worden ontvangen en gelogd.

Ik bedacht in de demoapplicatie 2.0 een extra taak: *Syslog_task*. Het doel van deze taak was de configuratie van de output van de taken. De gebruiker kan level en source van de output configureren. De levels kunnen zijn: critical en debug. Als het critical level is geconfigureerd, bestaat de output van het systeem uit foutmeldingen. Bij het debug level bestaat de output uit de events binnen het systeem. De taken van het systeem zijn sources. Via de *Syslog_task* kan de gebruiker de events configureren van de taak die hij wil zien. Het configuratiebericht van de gebruiker komt via de seriële poort naar de Control_task en wordt daarna naar de *queue_Syslog* verzonden. De Syslog_task leest de berichten uit de *queue_Syslog*, controleert of de query correct is en configureert de output van het systeem. De interpretatie van het configuratieverzoek staat in schema 9.2. In tabel 9.2 staan twee voorbeelden van de interpretatie van het configuratieverzoek van de gebruiker.

Schema 9.2: Configuratiebericht – interpretatie



Tabel 9.2: Voorbeelden van de interpretatie van het configuratieverzoek

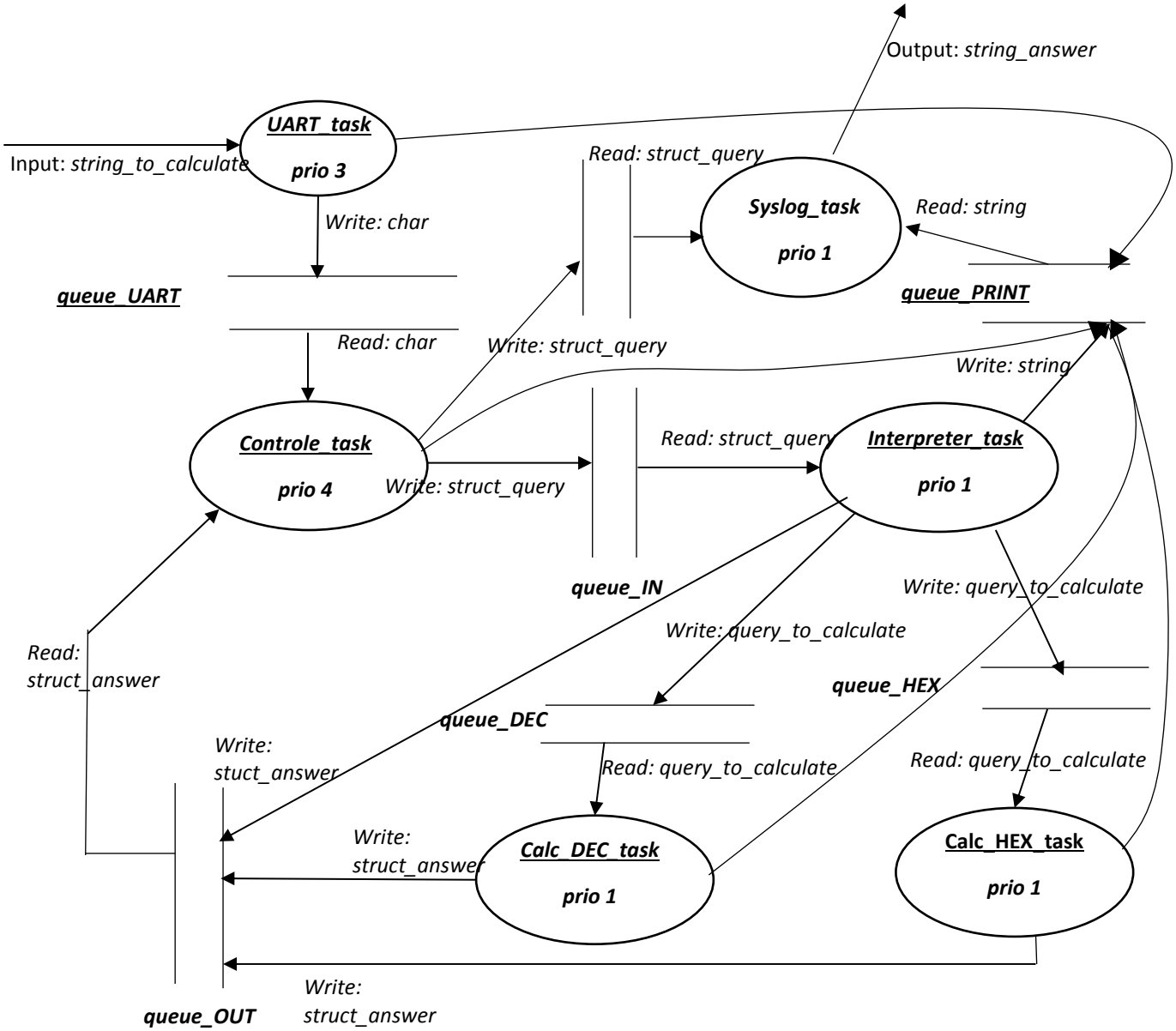
Configuratieverzoek 1	**"SSCd"**			
Teken	**S**	**S**	**C**	**d**
Interpretatie	Het bericht voor de *Syslog_task*	Configuratie van de source	Verwijderen source	*Calc_DEC_task*
Resultaat	Bij de output worden geen meldingen van de *Calc_DEC_task* weergegeven.			
Configuratieverzoek 2	**"SLCc"**			
Teken	**S**	**L**	**C**	**c**
Interpretatie	Het bericht voor de *Syslog_task*	Configuratie van de level	Verwijderen level	critical level.
Resultaat	Bij de output worden geen foutmeldingen van het systeem weergegeven			

Eis 17. De demoapplicatie 2.0 moet een aparte queue bevatten om outputberichten te verzamelen. Daarnaast moet de taak worden geïmplementeerd die deze berichten uitprint als de processor vrij is.

Om deze eis te realiseren voegde ik in de demoapplicatie 2.0 nog een extra functionaliteit aan de *Syslog_task* toe. Voor het printen van systeemberichten roepen de taken van het systeem de Syslog-functie aan. Deze functie is de voor het RTOS geadapteerde analogie van de Syslog-functie in de COMbricks PA link. Deze functie maakt een string van het bericht en stuurt, als de systeemoutput voor dit bericht is geconfigureerd, deze string naar de *queue_PRINT*. Deze functie is met Mutex beveiligd. De *Syslog_task* leest dit bericht uit en print het op de terminal. Als de *queue_PRINT* en *queue_SYSLOG* leeg zijn, wordt de *Syslog_task* geblokkeerd. Om dit te realiseren wordt een set van deze queues geïmplementeerd.

Toevoeging van extra functionaliteit in *Calculator* 2.0 leidt ook tot veranderingen in het ontwerp van de *UART_task* en de *Control_task*. De functionaliteit van de *UART_task* was verminderd: ze checkt de seriële port, leest het teken en stuur de tekens naar de *UART_queue*. In de *Control_task* wordt de functie toegevoegd die de inputtekens naar een structuurquery converteert. Voor de demoapplicatie 2.0 ontwierp ik nieuwe diagrammen, zie Bijlage XI. In schema 9.3 staat het dataflow-diagram van de demoapplicatie 2.0.

Schema 9.3: Dataflow-diagram demoapplicatie 2.0



Prioriteiten

In deze fase moest ik ook de prioriteiten van de taken bepalen. In FreeRTOS wordt de prioriteit als een getal toegevoegd: hoe groter het getal, hoe hoger de prioriteit.

De belangrijkste taak van de demoapplicatie 2.0 is de *Control_task*. Deze taak moet de inputtekens van de gebruiker converteren naar de query-structuur, de time-out van het systeem checken en de circulatie van de berichten binnen het systeem regelen. Deze taak moet echt periodiek uitgevoerd worden en mag niet worden onderbroken door de andere taken. Om deze reden moet deze taak de hoogste prioriteit hebben: '4'.

De *UART_task* is een periodieke taak van het systeem; de periode van uitvoering is 20msec. Deze taak moet alle andere taken (behalve de *Control_task*) onderbreken om de UART-poort te checken. Dus de *UART_task* heeft prioriteit '3'.

De *Diagnose_task* is een optionele taak van het systeem en wordt ook periodiek uitgevoerd, met een lange periode van 5sec. De accuratesse van de periodieke uitvoering is niet erg belangrijk ten opzichte van de *Control_task* en *UART_task*; deze taak krijgt de prioriteit '2'.

De andere taken zijn niet periodiek en worden door de queues gesynchroniseerd. Ze krijgen alle de laagste prioriteit, '1'. In tabel 9.3 staan de aan de taken toegekende prioriteiten.

Tabel 9.3: Prioriteiten van de taken in de demoapplicatie 2.0

Taak	Prioriteit
Control_task	4
UART_task	3
Diagnose_task	2
Interpreter_task	1
Calculate_DEC_task	1
Calculate_HEX_task	1
Syslog_task	1

Testen

Om te controleren of de demoapplicatie 2.0 aan de nieuwe gestelde eisen voldoet, moest ik voor elke eis testcases bedenken. In de testfase zou ik deze tests uitvoeren. Ik zou de correcte werking van de demoapplicatie als rekenmachine niet meer testen, want de functies die de berekening uitvoeren had ik in de demoapplicatie 2.0 niet veranderd.

In deze iteratie probeerde ik de problemen te oplossen die tijdens het testen van de demoapplicatie 1.0 naar voren komen. Hier noem ik de testcases die moeten laten zien of de problemen waren opgelost. De andere testcases staan in Bijlage XII.

- **Test 1.**

Eis 12. De implementatie van de taak die naar de seriële poort luistert, moet de interrupt simuleren.

Hier voerde ik de tests uit waarop de demoapplicatie 1.0 onvoldoende scoorde: twee stresstesten en de test op de verschillende prioriteiten van de taken. Ik verwachtte hier niet dat de eerste stresstest bij de nieuwe implementatie van de *UART_task* voldoende resultaat zou opleveren. Want in de nieuwe implementatie reageerde de *UART_task* op een time event, maar om het stresstestprobleem op te lossen moest deze taak op een input event reageren. Dat kon alleen bij implementatie van de seriële interrupt. De tweede stresstest wordt uitgevoerd met vertraging van 1msec per char. De testcases staan in tabel 9.4.

Tabel 9.4: Testcases voor test 1

Testcase	***Doel***	***Opstelling***	***Verwacht resultaat***
1.1	Controleren of het Preemptive algoritme van FreeRTOS correct werkt.	Elke 500msec checkt de *Control_task* de time-out. Elke 500msec print de *Control_task* de melding: "*Huidige time*. *Control_task: check the time."*	De melding: "*Huidige time*. *Control_task: check the time."* wordt elke 500msec geprint.
1.2 Stresstest 1	Controleren hoe *Calculator* systeem zich gedraagt als de gebruiker de input heel snel invoert.	*Calculator* wordt gestart. Een tekstbestand met 100 verzoeken wordt naar het systeem gestuurd.	Het systeem scoort onvoldoende. De input wordt gedeeltelijk uitgelezen.
1.3 Stresstest 2	Het doel van deze test is het controleren hoe draagt zich het Calculator systeem als de input van de gebruiker heel snel is ingevoerd	*Calculator* wordt gestart. Een tekstbestand met 100 verzoeken wordt naar het systeem gestuurd. Maar op de Terminal wordt geïnstalleerd dat elke char met 1msec vertraging binnen komt.	Het systeem berekent alle verzoeken.

9.3 Iteratie 4: ontwikkeling demoapplicatie 2.0. Implementatiefase

In deze fase creëerde ik de demoapplicatie 2.0. Hiervoor had ik de zeven taken en zeven queues gecreëerd die in het hoofdstuk Ontwerpfase zijn beschreven. De korte beschrijving van de queues staat in Bijlage XIII. In Bijlage XIV staan de definities van de structuren die in de demoapplicatie zijn gebruikt. De taken creëerde ik in een apart bestand. Het overzicht van de bestanden van de demoapplicatie staat in Bijlage XV.

Verder beschrijf ik de functies die nieuw of veranderd waren ten opzichte van versie 1.0.

1. vUART_task()

Aan deze taak was de Xilinx-functie toegevoegd:

XUartLite_IsReceiveEmpty(XPAR_RS232_UART_1_BASEADDR);

Deze functie checkt de seriële poort. Als er geen berichten zijn, blokkeert ze zich voor 20msec. Zodra er een bericht binnenkomt, leest de *UART_task* dit en stuurt het naar de *queue_UART*. Anders dan in versie 1.0 converteert in 2.0 de *UART_task* niet het bericht naar de structuur, maar geeft ze de input meteen via de *queue_UART* door naar de *Control_task*.

2. vControl_task()

Deze taak wordt periodiek uitgevoerd en door de *queue_UART* en *queue_OUT* geblokkeerd. Als deze queues leeg zijn, blijft de *Control_task* in geblokkeerde staat gedurende 500msec. Om dit te realiseren creëerde ik een set van queues met de functie:

```
Handle_qControl = xQueueCreateSet( COMBINED_LENGTH_CONTROL_SET);
```

De parameter van deze functie is de totale lengte van de queues in de set. De retourneerde waarde is de handle van de queue. Om een set van queues te creëren moest ik het configuratiebestand van FreeRTOS aanpassen. De parameter `configUSE_QUEUE_SETS` in het bestand *FreeRTOS.h* moest handmatig op één worden gezet. Na het creëren van de set werden de *queue_UART* en *queue_OUT* aan deze set toegevoegd met de functie `xQueueAddToSet(handle van queue, handle van set)`. Vervolgens werd de FreeRTOS-functie aangeroepen:

```
xActivatedMemberControl = pr_QueueSelectFromSet( Handle_qControl,xTicksToWaitSET);
```

Hierin is de eerste parameter de handle van de set en de tweede de wachttijd. In de demoapplicatie 2.0 is deze tijd 500msec. Deze functie retourneert de handle van de geactiveerde queue. Als de *queue_UART* wordt geactiveerd, leest de *Control_task* een teken vanuit de queue. Als de *queue_OUT* wordt geactiveerd, leest de *Control_task* het bericht uit de *queue_OUT* en vergelijkt de aankomsttijd van het antwoord met de tijd die in de buffer staat. Als binnen 500msec de queues niet worden geactiveerd, neemt de *Control_task* de huidige tijd en vergelijkt die met de tijd van de verzonden berichten die in de buffer staan. Als het verschil meer dan 500msec is, roept de *Control_task* de Syslog-functie aan en stuurt een time-out-foutmelding. Anders blokkeert de taak zich weer voor 500msec.

In de demoapplicatie 2.0 werd extra functionaliteit aan de *Control_task* toegevoegd. Wanneer in de *queue_UART* het teken 'Enter' komt (het eind van het verzoek), converteert de *Control_task* de tekens naar de structuur *STRUCT_QUERY* en bepaalt de volgende ontvanger van het bericht: de *Interpreter_task* of de *Syslog_task*.

3. vSyslog_task()

Dit is een nieuwe taak in de demoapplicatie 2.0. De functionaliteit van deze taak is:

- de output van het systeem configureren;
- de output printen.

Deze taak is de *ontvanger* van twee queues: *queue_SYSLOG* en *queue_PRINT*. Als deze queues leeg zijn, blijft de *Syslog_task* geblokkeerd. In deze taak wordt de set met de *queue_Syslog* en *queue_PRINT* gecreëerd.

Als *queue_SYSLOG* is geactiveerd en andere taken met hogere prioriteiten in geblokkeerde staat zijn, leest de *Syslog_task* het configuratiebericht vanuit *queue_SYSLOG*, controleert dit en configureert een syslog output message. De configuraties staan in tabel 9.5.

Tabel 9.5: Configuraties van syslog output message

Level	
DEBLEV_DEBUG	Alle events weergeven
DEBLEV_CRITICAL	Alle systeemfouten weergeven
Source	
DEBSRC_UART	De messages vanuit de *UART_task* worden geprint op het ingestelde level
DEBSRC_CONTROL	De messages vanuit de *Control_task* k worden geprint op het ingestelde level
DEBSRC_SYSLOG	De messages vanuit de *Syslog_task* worden geprint op het ingestelde level
DEBSRC_INTERPR	De messages vanuit de *Interpreter_task* worden geprint op het ingestelde level
DEBSRC_DEC	De messages vanuit de *Calc_DEC_task* worden geprint op het ingestelde level
DEBSRC_HEX	De messages vanuit de *Calc_HEX_task* worden geprint op het ingestelde level
DEBSRC_DIAGNOSE	De messages vanuit de *Diagnose_task* worden geprint op het ingestelde level
DEBSRC_ALL	De messages vanuit alle taken worden geprint op het ingestelde level

Als *queue_PRINT* is geactiveerd en andere taken met hogere prioriteiten in geblokkeerde staat zijn, leest de *Syslog_task* het bericht vanuit *queue_PRIN*T en print dit bericht op een console.

In de demoapplicatie 2.0 werd de nieuwe functie *SysLog()* geïntegreerd. Deze functie werkt de output van het systeem uit.

```
void Syslog(int sLevel, long long sSource, char *text, ...);
```

Deze functie is in de software van COMbricks PA link gebruikt. Deze functie heeft de volgende inputparameters:

- *sLevel*: het niveau van het syslog-bericht. In de demoapplicatie 2.0 kan dit debug level, critical level of allebei zijn.
- *sSource*: de source van het syslog-bericht. In de demoapplicatie 2.0 is elke taak een aparte source.
- **tekst*: een variabele (melding).

De Syslog-functie converteert de berekende waarde naar de string, neemt de aankomsttijd en stuurt de output naar de *queue_PRINT.* De Syslog- functie is beveiligd door Mutex. Mutex is gecreëerd met de API-functie *xSemaphoreCreateMutex().* De returnwaarde van deze functie is de handle van Mutex. In het begin van de Syslog-functie wordt Mutex genomen en aan het eind van deze functie wordt Mutex vrijgemaakt met de API-functies

```
xSemaphoreTake(handle of Semaphore, xTicksToWait);
```

```
xSemaphoreTake(handle of Semaphore).
```

De eerste parameter van de functies is de handle van Mutex en de tweede parameter van de eerste functie is de wachttijd. Binnen deze tijd wordt de taak die op Mutex wacht geblokkeerd. In de demoapplicatie is deze parameter als *portMAX_DELAY* gedefinieerd, dus is de wachttijd oneindig.

Om Mutex te gebruiken moet het configuratiebestand van FreeRTOS worden veranderd. De parameter *configUSE_MUTEXES* in het bestand *FreeRTOS.h* moet via de Board Support Package Settings op één worden gezet.

4. vDiagnose_task()

Deze taak werd in de demoapplicatie 2.0 geïmplementeerd. Deze taak wordt periodiek uitgevoerd: 1 keer per 5sec. Daarvoor is de FreeRTOS-functie *vTaskDelayUntil()* gebruikt. Wanneer de *Diagnose_task* is uitgevoerd, wordt de FreeRTOS-functie *uxTaskGetSystemState(..)* aangeroepen.

```
UBaseType_t uxTaskGetSystemState( TaskStatus_t *, const UBaseType_t, uint32_t *);
```

De eerste parameter van deze functie is een pointer naar een array van TaskStatus-structuren. Deze structuur is in de broncode van FreeRTOS gedefinieerd.

```
typedef struct xTASK_STATUS
{
```

`TaskHandle_t xHandle;` - de handle van de taak;

`const char *pcTaskName;` - de pointer naar de naam van de taak;

`UBaseType_t xTaskNumber;` - het unieke nummer van de taak;

`eTaskState eCurrentState;` - de huidige status van de taak;

`UBaseType_t uxCurrentPriority;` - de huidige prioriteit van de taak;

`UBaseType_t uxBasePriority;` - de prioriteit van de taak bij het creëren;

`uint32_t ulRunTimeCounter;` - de totale tijd dat de taak in de toestand Running is;

`uint16_t usStackHighWaterMark;` - het aantal stacks van de taak die niet zijn gebruikt; de rest van de stack;

```
} TaskStatus_t;
```

De tweede parameter van deze functie is het aantal bestaande taken. Deze parameter wordt verkregen door de FreeRTOS-functie `uxTaskGetNumberOfTasks()`. Deze functie retourneert het aantal bestaande taken.

De derde parameter is de pointer naar een getal van het type unsigned integer. In deze parameter wordt de totale tijd van de uitvoering van de applicatie geschreven.

Na uitvoering van deze functie wordt de verkregen *xTASK_STATUS* per taak als tabel geprint. Om deze API-functie te gebruiken moet de configuratie van FreeRTOS en broncode veranderd worden.

1. Via Xilinx moet de parameter *configUSE_TRACE_FACILITY* op '1' worden gezet:

   Board Support Package Settings >> freertos >> use_trace_facility;

2. In het bestand *freertos_v2_1_0.tcl* moet de parameter `configGENERATE_RUN_TIME_STATS` op '1' worden gezet:

   ```
   xput_define $config_file "configGENERATE_RUN_TIME_STATS" "1"
   ```

3. In het bestand *FreeRTOS.h* moeten de parameters `INCLUDE_eTaskGetState` en `INCLUDE_uxTaskGetStackHighWaterMark` op '1' worden gezet:

   ```
   #define INCLUDE_uxTaskGetStackHighWaterMark 1
   #define INCLUDE_eTaskGetState 1
   ```

4. In het bestand *FreeRTOS.h* moeten twee nieuwe functies en een runtime counter worden gedefinieerd:

```
extern volatile unsigned long ulHighFrequencyTimerTicks;
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() (ulHighFrequencyTimerTicks=0UL)
#define portGET_RUN_TIME_COUNTER_VALUE() ulHighFrequencyTimerTicks
```

5. In de broncode van FreeRTOS *port.c* en in de timer interrupt handle `vPortTickISR()` moet een variabele-counter worden toegevoegd:
   `ulHighFrequencyTimerTicks ++;`

Daarna wordt in de *Diagnose_task* de FreeRTOS-functie *uxQueueSpacesAvailable(handle van queue)* voor elke queue aangeroepen. De input van deze functie is de handle van de queue, de output is het actuele aantal vrije plaatsen in de queue.

Om overflow van de interne timer te voorkomen werd in de demoapplicatie 2.0 een softtimer geïntegreerd. De softtimer werd gecreëerd met de API-functie die de timer handle retourneert:

```
xTimerCreate(const char * const pcTimerName, const TickType_t xTimerPeriodInTicks, const UBaseType_t uxAutoReload, void * const pvTimerID, TimerCallbackFunction_t pxCallbackFunction);
```

De parameters van deze functie zijn: *pcTimerName* (naam van de timer), *xTimerPeriodInTicks* (periode van de timer), *uxAutoReload* (reload de timer of niet), *pxCallbackFunction* (timer-callback-functie). In de callback-functie was een algoritme gerealiseerd om overflow van de timer te voorkomen. De periode van de timer was 1msec geconfigureerd, en autoreload was aangezet.

Daarna werd de timer gestart met de API-functie:

```
xTimerStart( xTimer, xTicksToWait );
```

De parameters van deze functie zijn de handle van de timer en de wachttijd. In de demoapplicatie 2.0 was voor de softtimer geen wachttijd geconfigureerd.

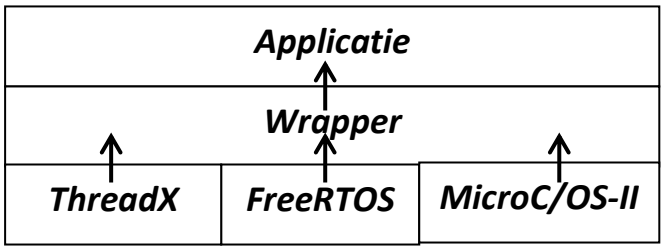
9.4 Iteratie 4: ontwikkeling demoapplicatie 2.0. Testfase

In deze fase testte ik de demoapplicatie 2.0. De testcases bedacht ik in de Ontwerpfase van deze iteratie en ze staan beschreven in Bijlage XII. De resultaten van de tests waren correct, zoals verwacht. Hier wil ik nog eens benadrukken dat de nieuwe implementatie van de *UART_task* niet het probleem van de stresstest oploste. De stresstest leverde geen voldoende resultaat op; tijdens te snelle input raakte de applicatie een aantal gegevens kwijt. Dit probleem kan alleen met implementatie van een seriële interrupt worden opgelost. In Bijlage XVI staan de resultaten van de tests.

10. Ontwikkeling wrapper

Tijdens het theoretische onderzoek koos ik een top-drie van RTOS'en: ThreadX, MicroC/OS-II en FreeRTOS. Het bedrijf besloot om een wrapper voor deze drie systemen te maken. De wrapper geeft de mogelijkheid een applicatie met wrapperfuncties maken die werkt met elke van deze drie systemen, zie tekening 10.1.

Tekening 10.1: Het werk van de wrapper



Tijdens de Analysefase van deze iteratie stelde ik de functionaliteit vast die de wrapper moest bieden. De vaststelling leidde ik af uit de eisen aan de demoapplicatie en uit het gesprek met de begeleider. Deze functionaliteit kwam in de eisen aan de wrapper terug. Daarna bepaalde ik in de Ontwerpfase welke functies van de drie RTOS'en ik in de wrapper moest opnemen. Tijdens deze fase moest ik ook de regels voor de naamgeving van de wrapperfuncties bepalen.

Vervolgens moest ik de compatibiliteit van de functies van de verschillende RTOS'en bekijken en oplossingen bedenken voor de complexe compatibiliteit van de functies. Aan het eind van de Ontwerpfase maakte ik het testplan voor de wrapper . In de Implementatiefase herdefinieerde ik de functies van de verschillende RTOS'en voor de wrapper. Tijdens de testfase voerde ik de tests uit.

10.1 Iteratie 5: ontwikkeling wrapper. Analysefase

In deze fase moest ik de functionaliteit bepalen die de wrapper moest bieden. In Hoofdstuk 9.1 had ik de eisen vastgesteld aan de functionaliteit van het RTOS die in de laatste versie van de demoapplicatie 2.0 moest worden aangetoond. Deze eisen reflecteerden de noodzakelijke functionaliteit van de COMbricks PA link, die dus ook golden bij het vaststellen van de functionaliteit van de wrapper. In tabel 10.1 staan de eisen aan de functionaliteit van RTOS en afhankelijk daarvan staan de functies beschreven die moesten worden gewrapt.

Tabel 10.: Functionaliteit van de wrapper

Eisen aan de demoapplicatie 2.0		Functies die de wrapper moet bevatten
Eis 1	In de demoapplicatie 2.0 moeten taken geconfigureerd worden.	• Creëren van een taak • Beheren van een taak • Starten van scheduler • Verwijderen van een taak
Eis 2	De demoapplicatie 2.0 moet het Preemptive algoritme van het RTOS demonstreren.	
Eis 3	De demoapplicatie 2.0 moet taken met verschillende prioriteiten bevatten.	
Eis 4	De demoapplicatie 2.0 moet periodieke taken bevatten.	
Eis 5	In de demoapplicatie 2.0 moeten queues geconfigureerd worden.	• Creëren van een queue • Schrijven in een queue • Lezen vanuit een queue • Hardware-timerfuncties • Beheren van een queue
Eis 6	In de demoapplicatie 2.0 moeten functies worden geïmplementeerd zoals Receive from the Queue, Send to the Queue.	
Eis 7	In de demoapplicatie 2.0 moeten processen gesynchroniseerd worden met gebruikmaking van Queues.	
Eis 8	In de demoapplicatie 2.0 moet de systeemtimer worden gebruikt.	• Time beheren
Eis 9	In de demoapplicatie 2.0 moet time-out van het systeem geconfigureerd worden.	
Eis 10	In de demoapplicatie 2.0 moet Mutex geconfigureerd worden.	• Creëren van Mutex • Beheren van Mutex
Eis 11	In de demoapplicatie 2.0 moet een resource worden beveiligd door middel van Mutex.	
Eis 12	De implementatie van de taak die naar de seriële port luistert, moet de interrupt simuleren. In de demoapplicatie werd niet de interrupt geïmplementeerd, maar in de toekomst kan in COMbricks PA link de seriële interrupt worden toegevoegd.	Bij sommige RTOS'en kunnen API-functies niet vanuit de interrupt-routine worden aangeroepen. Daarvoor zijn in deze RTOS'en speciale functies gebruikt. Deze functies moeten ook gewrapt worden.
Eis 13	De demoapplicatie 2.0 moet gebruikmaken van diagnostische functies als Task state List en Queue state.	Voor deze eis moeten de functies van RTOS worden gebruikt die de toestand van queues en taken kunnen laten zien.
Eis 15	In de demoapplicatie 2.0 moet een set van queues worden gecreëerd om de taken te synchroniseren.	• Creëren van een set • Beheren van een set

De eisen 14, 16 en 17 worden aan de demoapplicatie gesteld, niet aan de gebruikte RTOS-functionaliteit. Daarom worden ze niet genoemd bij de opstelling van de eisen aan de wrapper.

Tijdens het gesprek met de begeleider vroeg hij aandacht voor de synchronisatietool van de RTOS'en: Semaphoren, die in de demoapplicatie waren niet gebruikt. De wrapper moet ook de functies van het creëren en beheren van Semaphoren bevatten.

- Creëren van counter-Semaphoren
- Beheren van counter-Semaphoren

Elk RTOS heeft een scheduler: een besturingsmechanisme van processen. Het werk van de scheduler houdt verband met de hardwaretimer-interrupt. De gebruiker van de RTOS'en heeft de mogelijkheid om delen van een applicatie (critical sections) tegen de systeeminterrupt te beveiligen met behulp

van functies die scheduler stoppen of de interrupt uitschakelen. De wrapper moet deze functionaliteit ook bieden.

Op basis van de noodzakelijke functionaliteit van de wrapper kwamen er eisen aan wrapperfuncties op.

Eisen aan de wrapper:

Eis 1. De wrapper moet de functie bevatten die de scheduler start.

Eis 2. De wrapper moet de functie bevatten die een taak creëert.

Eis 3. De wrapper moet de functie bevatten die een taak verwijdert.

Eis 4. De wrapper moet de functie bevatten die de uitvoering van een taak opschort.

Eis 5. De wrapper moet de functie bevatten die de uitvoering van een taak hervat.

Eis 6. De wrapper moet de functie bevatten die de count van de hardwaretimer leest.

Eis 7. De wrapper moet de functie bevatten die een taak voor een vaste tijd blokkeert.

Eis 8. De wrapper moet de functie bevatten die de prioriteit van een taak verandert.

Eis 9. De wrapper moet de functie bevatten die een queue creëert.

Eis 10. De wrapper moet de functie bevatten die een item uit de queue leest met het verwijderen van item vanuit de queue.

Eis 11. De wrapper moet de functie bevatten die een item naar de queue zendt.

Eis 12. De wrapper moet de functie bevatten die de context van de queue leegmaakt.

Eis 13. De wrapper moet de functie bevatten die een softwaretimer creëert.

Eis 14. De wrapper moet de functie bevatten die een softwaretimer start.

Eis 15. De wrapper moet de functie bevatten die een software timer stopt.

Eis 16. De wrapper moet de functie bevatten die een software timer verwijdert.

Eis 17. De wrapper moet de functie bevatten die Mutex creëert.

Eis 19. De wrapper moet de functie bevatten die Mutex verwijdert.

Eis 20. De wrapper moet de functie bevatten die Mutex opneemt.

Eis 21. De wrapper moet de functie bevatten die Mutex vrijlaat.

Eis 22. De wrapper moet de functie bevatten die een item uit de queue vanuit de interruptroutine leest.

Eis 23. De wrapper moet de functie bevatten die een item vanuit interruptroutine naar de queue zendt.

Eis 24. De wrapper moet de functie bevatten die de huidige toestand van een taak weergeeft.

Eis 25. De wrapper moet de functie bevatten die hoeveelheid van resterende stack van een taak weergeeft.

Eis 26. De wrapper moet de functie bevatten die de prioriteit van een taak weergeeft.

Eis 27. De wrapper moet de functie bevatten die de naam van een taak weergeeft.

Eis 28. De wrapper moet de functie bevatten die algemene informatie over een taak weergeeft: de huidige toestand, de omvang van de resterende stack, de prioriteit, de naam en de CPU- tijd gedurende welke een taak is uitgevoerd.

Eis 29. De wrapper moet de functie bevatten die het aantal vrije plaatsen in een queue weergeeft.

Eis 30. De wrapper moet de functie bevatten die een set van de queues creëert.

Eis 31. De wrapper moet de functie bevatten die een queue aan de set toevoegt.

Eis 32. De wrapper moet de functie bevatten die geactiveerde queue vanuit de set kiest.

Eis 33. De wrapper moet de functie bevatten die een Counting Semaphore creëert.

Eis 34. De wrapper moet de functie bevatten die een Semaphore verwijdert.

Eis 35. De wrapper moet de functie bevatten die een Semaphore opneemt.

Eis 36. De wrapper moet de functie bevatten die een Semaphore vrijlaat.

Eis 37. De wrapper moet de functie bevatten die de scheduler opschort.

Eis 38. De wrapper moet de functie bevatten die het werk van de scheduler hervat.

Eis 39. De wrapper moet de functie bevatten die de systeeminterrupts uitschakelt.

Eis 40. De wrapper moet de functie bevatten die de systeeminterrupts aanzet.

Naamgeving voor de wrapper

De namen van de wrapperfuncties besprak ik met de begeleider. Hij vroeg om Engelstalige namen. Hij vond het ook zeer wenselijk dat de naam aan het doel van de functie voldeed. Tijdens het gesprek besloot ik om voor de namen van de functies de letters met onderstreepje '*pr_*' van PROCENTEC toe te voegen en voor de namen van de typen van variabelen de letters '*pr*' toe te voegen. Uit dit gesprek kwamen de volgende eisen voort:

Eis 41. De namen van de wrapperfuncties moeten in het Engels zijn.

Eis 42. De namen van de wrapperfuncties moeten duidelijk laten zien wat de functie doet.

Eis 43. Voor de namen van de wrapperfuncties moeten de letters met onderstreepje '*pr_*' staan.

Eis 44. Voor de namen van de typen van variabelen moeten de letters '*pr*' staan.

10.2 Iteratie 5: ontwikkeling wrapper. Ontwerpfase

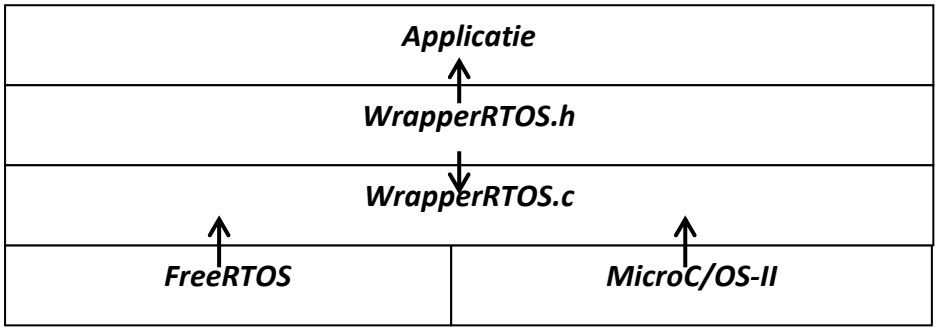
Tijdens deze fase moest ik de drie RTOS'en, ThreadX, MicroC/OS-II en FreeRTOS, dieper onderzoeken om een logische wrapper te kunnen bouwen. De eerste stap daarvoor was de integratie van deze systemen in de Xilinx 14.7 omgeving. Tijdens het ontwikkeling van de demoapplicatie 1.0 was FreeRTOS al in de Xilinx 14.7 omgeving geïntegreerd. Het stappenplan voor integratie van FreeRTOS staat in Bijlage VI. De integratie van MicroC/OS-II lijkt op de integratie van FreeRTOS, het stappenplan van de integratie van MicroC/OS-II staat in Bijlage XVII.

ThreadX biedt in de trialversie van het systeem een voor Big Endian[1] gecompileerde library, die vanuit http://rtos.com/products/threadx/ kan worden gedownload. Om de functies van ThreadX te gebruiken moet een applicatie naar deze library worden gekoppeld. Aan de andere kant moet een applicatie worden gekoppeld aan de library's die de FPGA-configuratie bevatten. De library's van de FPGA-configuratie zijn gecompileerd voor Little Endian[15]. De ThreadX library en de library's van de FPGA configuratie zijn dus niet compatibel.

Om dit probleem op te lossen heb ik met het technische ondersteuningsteam van ThreadX gecommuniceerd via e-mail. Deze uitwisseling duurde ongeveer een maand en ten slotte bood het ondersteuningsteam volledige toegang tot de broncode van ThreadX aan voor het testen. Maar om de toegang te krijgen moest PROCENTEC een bepaalde conventie ondertekenen. Volgens deze conventie zou PROCENTEC verantwoordelijk zijn voor non-proliferatie van de broncode van ThreadX. Dit voorstel had bepaalde risico's voor PROCENTEC. Zo had een toevallige distributie van de code op het internet een groot probleem voor het bedrijf kunnen vormen. PROCENTEC vond deze risico's onnodig, gezien de hoge prijs van het systeem, en besloot om het testen van ThreadX te weigeren.

Ik besloot dus om de wrapper voor twee RTOS'en te maken: FreeRTOS en MicroC/OS-II. Het idee van de wrapper is dat de applicatie de wrapperfuncties gebruikt die in het headerbestand van de wrapper zijn gedefinieerd. Deze functies zijn in het bronbestand van de wrapper geïmplementeerd. Afhankelijk van het geactiveerde RTOS worden de wrapperfuncties van MicroC/OS-II of FreeRTOS toegepast. Het headerbestand van de wrapper is opgenomen in de applicatiebestanden en het bronbestand van de wrapper. De bestanden van MicroC/OS-II en FreeRTOS zijn in het bronbestand van de wrapper opgenomen, zie tekening 10.2. Zodoende ziet de applicatie de functies van de wrapper en niet de functies van FreeRTOS en MicroC/OS-II.

Tekening 10.2: De verbinding van de wrapper met de RTOS'en en een applicatie



Het grootste verschil tussen FreeRTOS en MicroC/OS-II is dat in FreeRTOS het Round-robin scheduling algoritme is geïmplementeerd en in MicroC/OS-II niet. Dit betekent dat met gebruik van FreeRTOS verschillende taken dezelfde prioriteiten kunnen hebben en dat met gebruik van MicroC/OS-II taken verschillende prioriteiten moeten hebben. Als een applicatie compatibel moet zijn met beide RTOS'en moet elke taak van een applicatie een verschillende prioriteit hebben. Het idee was de wrapper met de demoapplicatie 2.0 te testen; daarvoor veranderde ik de prioriteiten van de taken in de demoapplicatie 2.0.

[1] Zie Begrippenlijst, Bijlage I

Functies van FreeRTOS en MicroC/OS-II voor de wrapper

In de Analysefase waren de eisen voor de wrapper vastgesteld. In deze fase bepaalde ik op basis van die eisen de functies van de RTOS'en. Tijdens de Ontwerpfase zocht ik de functies van FreeRTOS en MicroC/OS-II op die dezelfde actie uitvoeren. Dit betekent niet dat functies met dezelfde actie identiek zijn, want ze hebben verschillende parameters, returnwaardes en implementatie. In dit stadium analyseerde ik deze functies en vond ik mogelijke richtingen voor de oplossingen.

De volledige beschrijving van de wrapperfuncties per eis staat in Bijlage XVIII. Hier schetste ik de wrapperfuncties waarvan ik het ontwerp het meest interessant en complex vond. Bovendien worden hier ook de functies genoemd waarvan tijdens de ontwikkeling de nadelen van FreeRTOS of MicroC/OS-II naar voren kwamen.

Eis 2. De wrapper moet de functie bevatten die een taak creëert.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_TaskCreate(..)*	
Functie	*xTaskGenericCreate(..)*	*OSTaskCreateExt(..)*
Inputparameters	1. void pointer naar de taak; 2. naam van de taak; 3. stackdiepte; 4. pointer naar de data die kan als de parameter van de taak worden gebruikt; 5. prioriteit; 6. pointer naar de taakhandle; 7. pointer naar *top-of-stack* van de taak; 8. (optioneel) pointer naar memorylocatie;	1. void pointer naar de taak; 2. pointer naar de data die kan als de parameter van de taak worden gebruikt; 3. pointer naar *top-of-stack* van de taak; 4. prioriteit; 5. id == prioriteit; 6. pointer naar *bottom-of-stack* van de taak; 7. stackdiepte; 8. (optioneel) pointer naar memorylocatie; 9. taakspecifieke opties;
Returnwaarden	foutmelding	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

De functie in FreeRTOS heeft acht parameters, de functie in MicroC/OS-II heeft negen parameters. De zesde parameter van de MicroC/OS-II-functie is niet noodzakelijk voor de wrapperfunctie; hij kan tijdens de implementatie van de wrapperfunctie berekend worden, omdat de diepte van de stack en pointer van de *top-of-stack* van de taak bekend zijn. De andere parameters van de functies zijn compatibel. De handle van de taak voor MicroC/OS-II kan prioriteit van de taak zijn.

Eis 4. De wrapper moet de functie bevatten die de utvoering van een taak opschort.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_TaskSuspend(..)*	
Functie	*vTaskSuspend(..);*	*OSTaskSuspend(..);*
Inputparameters	- pointer naar de taakhandle;	- prioriteit;
Returnwaarden	geen	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

De functie van FreeRTOS heeft geen returnwaarde, de functie van MicroC/OS-II heeft een foutmelding als returnwaarde. Dit probleem kan op twee manieren worden opgelost:

1. De wrapperfunctie moet een returnwaarde hebben, in het geval van FreeRTOS kan ze altijd true retourneren.
2. De wrapperfunctie moet geen returnwaarde hebben, in het geval van MicroC/OS-II kan ze de returnwaarde negeren.

De handle van de taak werd als prioriteit gedefinieerd voor MicroC/OS-II.

Ik heb de eerste variant voor de wrapper gekozen om dit probleem op te lossen. Veel functies van FreeRTOS en MicroC/OS-II vereisen een bepaalde configuratie van het systeem. Tijdens de implementatie van de wrapperfunctie kan de configuratie van het systeem gecontroleerd worden. Als het in de configuratie van het systeem niet correct is om de functie te gebruiken, kan de wrapperfunctie de configuratiefout retourneren.

Eis 9. De wrapper moet de functie bevatten die een queue creëert.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_QueueCreate(..)*	
Functie	*xQueueGenericCreate(..);*	*OSQCreate(..);*
Inputparameters	1. het aantal elementen in de queue; 2. de grootte van één element; 3. type van de queue (constant);	1. pointer naar *top-of-stack* van de queue; 2. het aantal elementen in de queue;
Returnwaarde	pointer naar de handle van de queue	pointer naar de handle van de queue
Verschil in implementatie	dynamische geheugentoewijzing	statische geheugentoewijzing

MicroC/OS-II is op zo'n manier gebouwd dat de gebruiker voor het creëren van een queue of een taak het geheugen statisch moet toewijzen. In het geval van de queue moet de gebruiker een array in de applicatie creëren. De lengte van de array moet gelijk zijn aan het aantal elementen in de queue en de elementen in de array moeten dezelfde zijn als de elementen in de queue. Daarna gebruikt het systeem het toegewezen geheugen van de array voor de queue.

FreeRTOS wijst het geheugen dynamisch toe. Het systeem kent de lengte en de grootte van de elementen van de queue die wordt gecreëerd. Op basis van deze kennis wijst het geheugen dynamisch toe binnen de implementatie van de functie *xQueueGenericCreate()*.

Dit probleem kan worden opgelost als de wrapperfunctie drie parameters heeft: het aantal elementen in de queue, de grootte van één element, de pointer naar *top-of-stack* van de queue. Voor MicroC/OS-II wordt de grootte van één element niet gebruikt, voor FreeRTOS wordt de pointer naar *top-of-stack* van de queue niet gebruikt. Maar in de applicatie moeten de arrays voor het creëren van een queue worden gedefinieerd. Deze toepassing moet in de demoapplicatie 2.0 terugkomen. Deze oplossing is niet optimaal maar ik kon geen andere vinden.

De dynamische geheugentoewijzing is een nadeel van FreeRTOS, want de gebruiker kan de verdeling van het geheugen niet controleren.

Eis 10. De wrapper moet de functie bevatten die een item uit de queue leest.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_QueueReceive(..)*	
Functie	*xQueueGenericReceive(..);*	*OSQPend(..);*
Inputparameters	- handle van de queue; - pointer naar het geheugen voor data; - wachttijd; - constante die bepaalt of het element vanuit queue moet worden verwijderd of niet;	- handle van de queue; - wachttijd; - pointer naar variabele van fout;
Returnwaarde	foutmelding	- pointer naar het geheugen voor data;
Verschil in implementatie	De functie kopieert het element vanuit de queue naar het adres van het geheugen van de data die als parameter was doorgegeven.	De functie retourneert de pointer naar het geheugen van de element in de queue.

Om deze functies te combineren moet een wrapperfunctie gecreëerd worden die een foutmelding als returnwaarde heeft en drie inputparameters: handle van de queue, pointer naar het geheugen voor data en wachttijd. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet de inhoud van returnpointer van de functie *OSQPend()* naar pointer van het geheugen voor data gekopieerd worden. Op dezelfde manier worden ook de andere functies met hetzelfde probleem ontworpen.

Eis 24. De wrapper moet de functie bevatten die de huidige toestand van de taak weergeeft.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_TaskGetStatus(..)*	
Functie	*eTaskGetState(..);*	*OSTaskQuery(..);*
Inputparameters	- handle van de taak;	- prioriteit van de taak; - pointer naar TCB-blok[1];
Returnwaarde	toestand van de taak	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier wordt de wrapperfunctie gecreëerd die de handle van de taak als inputparameter heeft en de toestand van de taak als returnwaarde. De handle van de taak voor MicroC/OS-II werd als prioriteit gedefinieerd. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet het TCB-blok gecheckt worden. Het TCB-blok bevat de huidige toestand van de taak die als returnwaarde wordt teruggegeven. De gebruiker heeft vrije toegang tot het TCB-blok in MicroC/OS-II. Dit is een nadeel van MicroC/OS-II, want zo kan de gebruiker daar per ongeluk iets veranderen, wat de werking van het systeem kan verstoren.

Eis 28. De wrapper moet de functie bevatten die algemene informatie over een taak weergeeft: de huidige toestand, de omvang van de resterende stack, de prioriteit, de naam en de CPU-tijd gedurende welke een taak is uitgevoerd.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_TaskGetState(..)*	
Functie	*uxTaskGetSystemState(..);*	*OSTaskQuery(..);*
Inputparameters	- pointer naar het *TaskStatus_t* structuur; - grootte van array, is gelijk aan het aantal aangemaakte taken; - pointer naar variabele die de algemene tijd van de werking van het systeem bevat;	- prioriteit van de taak; - pointer naar TCB-blok;
Returnwaarde	de array van *TaskStatus_t*-structuur die de informatie over de taken bevat;	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Deze functies zijn niet compatibel, want FreeRTOS heeft een functie die het aantal gecreëerde taken retourneert. Dit getal is gebruikt als tweede parameter van de FreeRTOS-functie. In MicroC/OS-II bestaat geen functie die het aantal gecreëerde taken bepaalt. De enige oplossing is in de implementatie van wrapperfunctie voor FreeRTOS de volgende functie aan te roepen: *eTaskGetState(); uxTaskGetStackHighWaterMark(); uxTaskPriorityGet(); pcTaskGetTaskName()*.

Om de CPU-tijd waarin de taak is uitgevoerd te weten te komen, moet de functie aan de broncode van FreeRTOS toegevoegd worden. Het TCB-blok van FreeRTOS bevat de runtime van de taak. Het TCB-blok is gesloten voor de gebruiker. In de broncode van FreeRTOS kan de functie worden geïmplementeerd die het TCB-blok van de taak ziet en de runtime van de taak retourneert.

Eis 30. De wrapper moet de functie bevatten die een set van queues creëert.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_QueueCreateSet(..)*	
Functie	*xQueueCreateSet(..);*	niet geïmplementeerd

De mogelijke oplossing beschreef ik verder.

Eis 31. De wrapper moet de functie bevatten die een queue aan de set toevoegt.

RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_QueueAddToSet(..)*	
Functie	*xQueueAddToSet(..);*	niet geïmplementeerd

De mogelijke oplossing beschreef ik verder.

Eis 32. De wrapper moet de functie bevatten die uit de set een geactiveerde queue kiest.

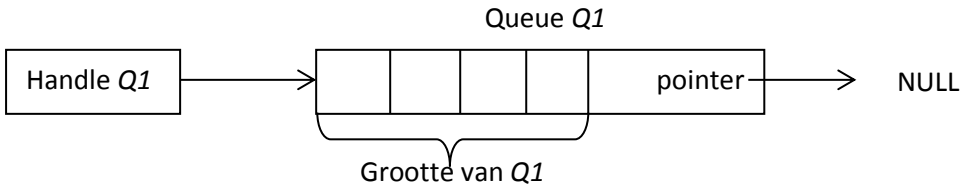
RTOS	**FreeRTOS**	**MicroC/OS-II**
Wrapperfunctie	*pr_QueueSelectFromSet(..)*	
Functie	*xQueueSelectFromSet(..);*	niet geïmplementeerd

De set van queues is niet geïmplementeerd in MicroC/OS-II. Deze functie is belangrijk binnen het systeem, ze geeft de mogelijkheid om een taak synchroniseren die uit meerdere queues leest. Ik heb gekeken hoe deze functie is gerealiseerd in FreeRTOS. Elke queue heeft een structuur waarin alle informatie over de queue staat. Deze structuur wordt aangemaakt bij het creëren van de queue. In FreeRTOS heeft deze structuur een pointer naar de queue-structuur, die standaard nul is. De

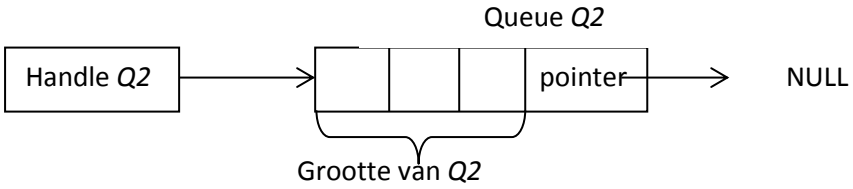
FreeRTOS-functie: *xQueueCreateSet(..)* creëert een nieuwe queue(queueSET). De grootte van deze nieuwe queue(queueSET) is de totaalomvang van de queues(Q1, Q2, ...) die de set zal bevatten. Als een queue aan de set is toegevoegd met de functie *xQueueAddToSet(..)*, wordt de pointer binnen de queue-structuur gelijkgesteld aan de handle van de set. De functie *xQueueSelectFromSet(..)* leest vanuit de queueSET en retourneert de handle van de geactiveerde queue in de set (de queue die een item binnenkrijgt). Wanneer in een van de queues die aan de set zijn toegevoegd een element komt, wordt deze queue naar set(queueSET) gestuurd, zie schema 10.1.

Schema 10.1: Werking van de set functies in FreeRTOS

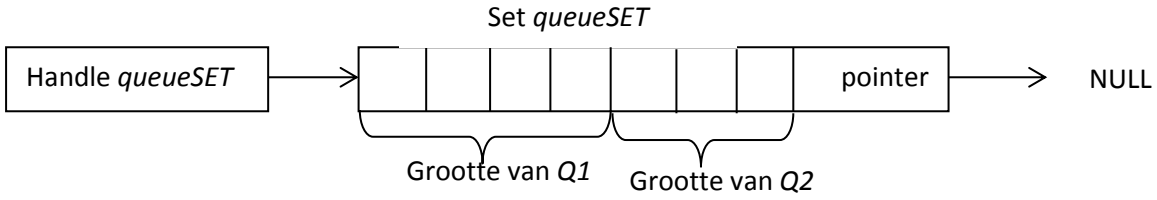
1. *Creëren van queue 1 xQueueGenericCreate(..)*



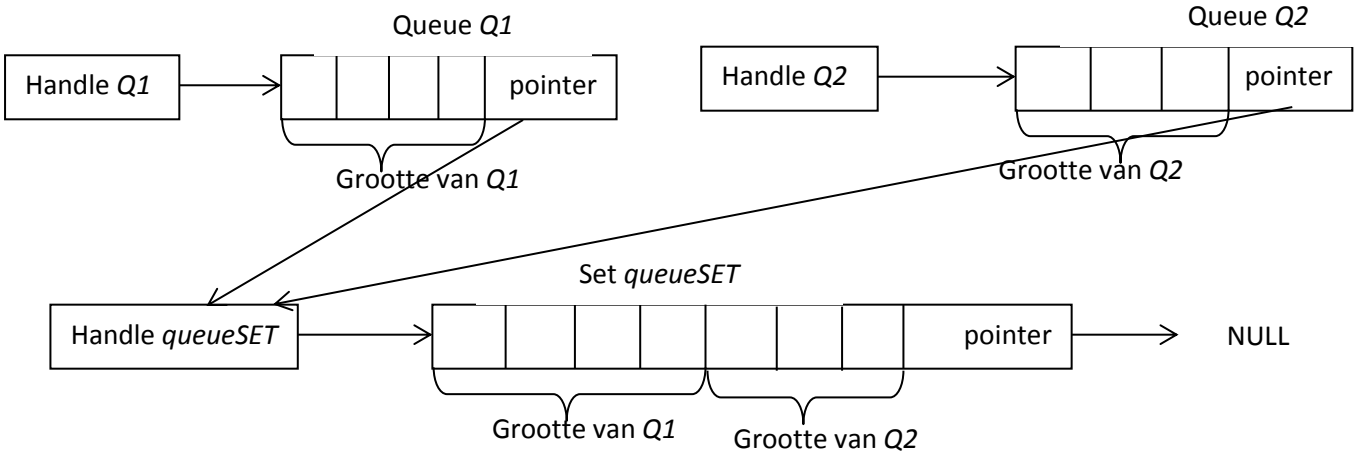
2. *Creëren van queue 2 xQueueGenericCreate(..)*



3. *Creëren van set xQueueCreateSet(..)*

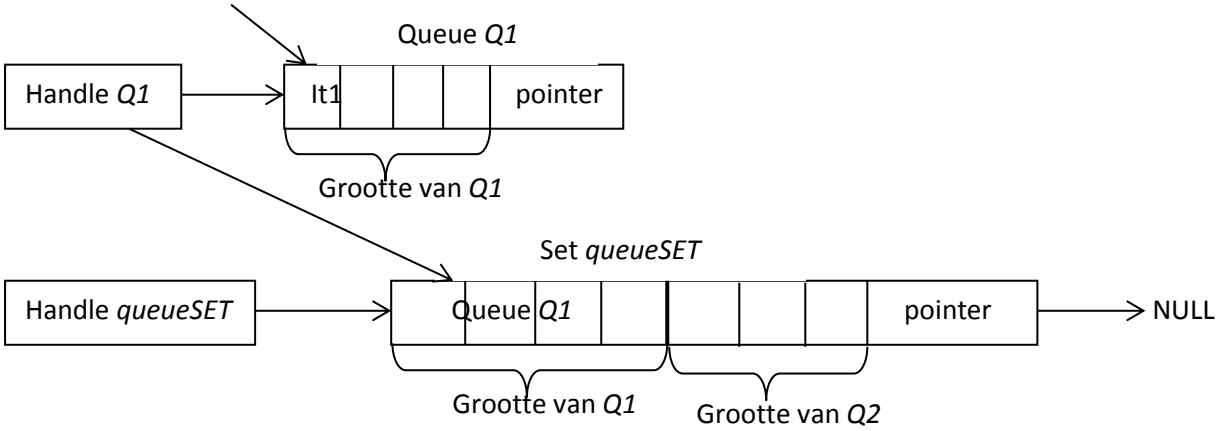


4. *Toevoegen de queue 1 en queue 2 in de set xQueueAddToSet(..)*



5. Kiezen welke queue in de set een data is gekregen in de set: `xQueueSelectFromSet(..)`

Een element is naar queue 1 gestuurd.



Op dezelfde manier kunnen de setfuncties in MicroC/OS-II worden gerealiseerd. Maar dat vereist veranderingen in de broncode van MicroC/OS-II.

Testplan

Na het bepalen van de functies die de wrapper moet bevatten heb ik het testplan gemaakt, zie Bijlage XIX. De testcases vloeiden voort uit de vastgestelde eisen voor de wrapper. Elke testcase controleerde of de wrapper voldeed aan de vastgestelde eisen. De tests werden uitgevoerd tijdens de Testfase van de vijfde iteratie.

Tijdens deze fase bleek dat de implementatie van de functies van de wrapper nieuwe implementaties vereist van de queue, TCB-blokken en sommige functies van MicroC/OS-II en FreeRTOS. Dit probleem kon op twee manieren opgelost worden:

1. Structuren en functies in de wrapper laten overlappen.
2. De broncode van de RTOS'en veranderen.

Dit probleem besprak ik met mijn begeleider. Ik koos de tweede oplossing, want de realisatie van de eerste oplossing zou veel tijd kosten. De veranderingen realiseerde ik in aparte bestanden. Deze bestanden moest ik in plaats van de bronbestanden van het RTOS gebruiken.

10.3 Iteratie 5: ontwikkeling wrapper. Implementatiefase

Tijdens deze fase werd de wrapper geïmplementeerd. Ik creëerde het bestand *wrapper_rtos.h*, waarin alle wrapperfuncties waren gedefinieerd. Deze functies werden in *wrapper_rtos.c* geïmplementeerd. De wrapper staat in de Bijlage XX.

De eerste stap in het bouwen van de wrapper was de herdefiniëring van de typen variabelen. De typen variabelen waren in het headerbestand gedefinieerd. De namen van de typen waren volgens eis 44: ''Voor de namen van de typen variabelen moeten de letters '*pr*' staan.'' Zo werd bijvoorbeeld het type *char* als *prInt8* gedefinieerd en *unsigned char* als *prInt8U*. Om queues, taken, semaphoren, timers en Mutex te beheren moesten de handles van deze tools worden gebruikt. In de wrapper werden alle handles als type *void** gedefinieerd, tijdens de implementatie van de functies werd een cast gebruikt die de handle converteerde naar het juiste type voor het systeem.

Tijdens de implementatie van de wrapper kwam nog een technisch verschil tussen FreeRTOS en MicroC/OS-II naar voren. Om FreeRTOS te starten moeten alleen de taken worden gecreëerd en scheduler worden gestart. Maar MicroC/OS-II vereist de initialisatie van interrupt controller, hardwaretimer en systeem zelf, wat in FreeRTOS binnen het systeem ontstaat wanneer de scheduler wordt gestart. Voor de initialisatie van het systeem werden de functies in de wrapper gedefinieerd en geïmplementeerd voor MicroC/OS-II. Voor FreeRTOS waren deze functies leeg, maar ze moesten wel in de demoapplicatie 2.0 staan.
MicroC/OS-II initialisatie:

```
void pr_SistemInit()
{
        #ifdef USING_microCOS_II
                OSInit();
        #endif /* USING_microCOS_II */
}
```

Initialisatie van interrupt controller en timer:

```
void pr_BSP_InitIO()
{
        #ifdef USING_microCOS_II
                if (OS_TASK_STAT_EN > 0)
                {
                        BSP_InitIO();
                        OSStatInit();
                }
                else
                        BSP_InitIO();
        #endif /* USING_microCOS_II */
}
```

Voor het gebruik van veel functies van FreeRTOS en MicroC/OS-II moesten de configuratiebestanden van de RTOS'en worden aangepast. De juiste configuratie is in de implementatie van de wrapperfuncties gecheckt met het *if*-statement.

Verder beschrijf ik de implementatie van de wrapperfuncties die in de Ontwerpfase waren vastgesteld. De implementatie van de functies die in tabel 9.5 staan was makkelijk, want ze gebruiken functies van FreeRTOS en MicroC/OS-II die aan elkaar gelijk zijn.

Tabel 9.5: Wrapperfuncties die makkelijk te realiseren waren

	Wrapperfunctie	Beschrijving
1	`void pr_DISABLE_INTERRUPTS();` `void pr_ENTER_CRITICAL();`	De functie die interrupts uitschakelt.
2	`void pr_ENABLE_INTERRUPTS();` `void pr_EXIT_CRITICAL();`	De functie die interrupts aanschakelt.
3	`void pr_TaskSuspendAll();`	De functie die de scheduler opschort.
4	`void pr_TaskResumeAll();`	De functie die het werk van de scheduler hervat.
5	`void pr_TaskDelete(prTaskHandle pr_THandle);`	De functie die een taak verwijdert.
6	`prErrorMessage pr_TaskSuspend(prTaskHandle pr_THandle);`	De functie die de utvoering van een taak opschort.
7	`prErrorMessage pr_TaskResume(prTaskHandle pr_THandle);`	De functie die de utvoering van een taak hervat.

8	`prErrorMessage pr_TaskDelay(prTickType prTimeTowait);`	De functie die een taak voor een vaste tijd blokkeert.
9	`prTickType pr_TaskGetTickCount();`	De functie die de count van de hardwaretimer leest.
10	`prErrorMessage pr_QueueSend(prQueueHandle prQHandle, void *prItem);`	De functie die een item naar de queue zendt.
11	`prErrorMessage pr_QueueReset(prQueueHandle prQHandle);`	De functie die de context van de queue leegmaakt.
12	`prSemaphoreHandle pr_CreateSemaphore(prInt16U prMaxCount);`	De functie die een Counting Semaphore creëert.
13	`prErrorMessage pr_DeleteSemaphore(prSemaphoreHandle rSHandle);`	De functie die een Semaphore verwijdert.
14	`prErrorMessage pr_SemaphoreTake(prSemaphoreHandle prSHandle, prTickType prTimeOut);`	De functie die een Semaphore opneemt.
15	`prErrorMessage pr_SemaphoreGive(prSemaphoreHandle prSHandle);`	De functie die een Semaphore vrijgeeft.
16	`prMutexHandle pr_CreateMutex();`	De functie die Mutex creëert.
17	`prErrorMessage pr_DeleteMutex(prMutexHandle prMHandle);`	De functie die Mutex verwijdert.
18	`prErrorMessage pr_MutexTake(prMutexHandle prMHandle, prTickType prTimeOUT);`	De functie die Mutex opneemt.
19	`prErrorMessage pr_MutexGive(prMutexHandle prMHandle);`	De functie die Mutex vrijgeeft.
20	`prErrorMessage pr_QueueSendFromISR(prQueueHandle prQHandle, void* pr_Item);`	De functie die een item vanuit een interruptroutine naar de queue zendt.
21	`prErrorMessage pr_TimerStart(prTimerHandle prTmHandle);`	De functie die een softwaretimer start.
22	`prErrorMessage pr_TimerStop(prTimerHandle prTmHandle);`	De functie die een softwaretimer stopt.
23	`prErrorMessage pr_TimerDelete(prTimerHandle prTmHandle);`	De functie die een softwaretimer verwijdert.
24	`prInt32U pr_qQuery(prQueueHandle prQHandle);`	De functie die het aantal vrije plaatsen in een queue weergeeft.
25	`const prInt8* pr_TaskGetStatus(prTaskHandle prtHandle);`	De functie die de huidige toestand van de taak weergeeft.
26	`prInt32 pr_TaskGetStack(prTaskHandle prtHandle);`	De functie die omvang van de resterende stack van een taak weergeeft.
27	`prInt32 pr_TaskGetPriority(prTaskHandle prtHandle);`	De functie die de prioriteit van een taak weergeeft.
28	`const prInt8* pr_TaskGetName(prTaskHandle prtHandle);`	De functie die de naam van een taak weergeeft.
29	`prTickType pr_GetfTICK_RATE();`	De functie die het aantal milliseconden in één tick berekent.

Anders dan in FreeRTOS retourneren veel functies in MicroC/OS-II geen foutmelding maar geven ze de pointer naar de code error als parameter van de functie. Dit probleem was opgelost tijdens Ontwerpfase. Zie bijvoorbeeld de functie die Mutex opneemt. Ik creëerde de wrapperfunctie die de handle van Mutex als inputparameter en een foutmelding als returnwaarde heeft. Binnen de implementatie voor FreeRTOS riep ik de API-functie aan die dezelfde parameter en returnwaarde heeft, voor MicroC/OS-II definieerde ik de variabele voor de foutmelding en gaf ik de pointer naar deze variabele als inputparameter van de MicroC/OS-II-functie. Daarna checkte ik deze parameter en afhankelijk van het resultaat bepaalde ik de returnwaarde:

```
prErrorMessage pr_MutexTake(prMutexHandle prMHandle, prTickType prTimeOUT)
{
        #ifdef USING_FreeRTOS
                prErrorMessage prError = 0;
                prError = xSemaphoreTake(prMHandle, prTimeOUT);
                if(prError == pdPASS)
                        return prOK;
                else
                        return prERROR;
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                prErrorMessage prError = 0;
                OSMutexPend((OS_EVENT*)prMHandle, prTimeOUT, &prError);
                if(prError == OS_ERR_NONE)
                        return prOK;
                else
                        return prERROR;
        #endif /* USING_microCOS_II */
        return prCONF_ERROR;
}
```

Verder beschrijf ik de wrapperfuncties waarin tijdens de implementatie problemen naar voren kwamen.

1. De wrapperfunctie die een taak creëert: *pr_TaskCreate(..)*

   Tijdens de implementatie voor **FreeRTOS:**

   - werd er een API-functie die een taak creëert, aangeroepen.

   Tijdens de implementatie voor **MicroC/OS-II:**

   - werd de handle van de taak gelijkgesteld aan de prioriteit van de taak;
   - moest in het configuratiebestand van MicroC/OS-II worden bepaald hoe de stack van het geheugen groeit. Hij kan van laag naar hoog en andersom groeien. Afhankelijk van deze configuratie wordt de *top-of-stack* bepaald (of het hoogste geheugenadres, of het laagste). Daarna moeten bij verschillende configuraties verschillende parameters voor de MicroC/OS-II-functie worden doorgegeven. Dus tijdens de implementatie moet deze configuratie gecheckt worden en afhankelijk daarvan de functie van MicroC/OS-II anders aangeroepen worden;
   - anders dan in FreeRTOS hoeven in MicroC/OS-II de taken geen namen te hebben.

   Om een taak een naam te geven moet een speciale functie van MicroC/OS-II aangeroepen worden. Dat werd gerealiseerd in de implementatie van deze wrapperfunctie.

2. De functie die de prioriteit van een taak verandert is *pr_TaskPrioritySet(..)*

   Tijdens de implementatie voor **FreeRTOS:**

   - werd er een API-functie die de prioriteit van een taak verandert, aangeroepen.

   Tijdens de implementatie voor **MicroC/OS-II:**

   - werd de handle van de taak gelijkgesteld aan de nieuwe prioriteit.
3. De functie die een queue creëert is *pr_QueueCreate(..)*.
   - Het probleem met de implementatie van deze functie was in de Ontwerpfase opgelost.

   Deze wrapperfunctie heeft drie inputparameters: het aantal elementen in de queue, de grootte van één element en de pointer naar de *top-of-stack* van de queue. De FreeRTOS API-functie gebruikt twee parameters: het aantal elementen in de queue en de grootte van één element. De MicroC/OS-II-functie gebruikt twee andere parameters: de grootte van één element en de pointer naar de *top-of-stack* van de queue.

4. De functie die een item uit de queue leest is *pr_QueueReceive(..)* en de functie die een item uit de queue vanuit de interruptroutine leest is *pr_QueueReceiveFromISR(..)*.

   Tijdens de implementatie voor **FreeRTOS:**

   - werden er API-functies aangeroepen die een item uit de queue lazen en die een item uit de queue vanuit de interruptroutine lazen.

   Tijdens de implementatie voor **MicroC/OS-II:**

   - werd het probleem met deze functie in de Ontwerpfase opgelost.

   De functie van MicroC/OS-II die een item uit de queue leest, retourneert het adres van het element in de queue. Daarna moet de inhoud van dit adres met behulp van de functie *memcpy()* worden gekopieerd naar het adres dat als inputparameter van de wrapperfunctie was doorgegeven. Maar om deze functie te kunnen gebruiken moet de omvang van het item bekend zijn. Daarom hebben de wrapperfuncties *pr_QueueReceive(..)* en *pr_QueueReceiveFromISR(..)* de omvang van elementen als derde parameter.

5. De functie die een softwaretimer creëert is *pr_TimerCreate(..)*.

   Tijdens de implementatie voor **FreeRTOS:**

   - werd de API-functie die een softwaretimer creëert aangroepen.

   Tijdens de implementatie voor **MicroC/OS-II:**

   - riep de wrapperfunctie de MicroC/OS-II-functie *OSTmrCreate(),* aan. Maar om deze functie te activeren moest de broncode van MicroC/OS-II veranderd worden. In het bestand *os_cpu_.c* moest aan de functie *void OSTimeTickHook(void),* de functie *OSTmrSignal()* worden toegevoegd.
6. De functie die alle informatie over een taak geeft is *prTaskGetState(..)*.

   Het probleem met de implementatie van deze wrapperfunctie voor **FreeRTOS:**

   - was al besproken tijdens de Ontwerpfase. FreeRTOS heeft:
     - API-functies die de huidige toestand van de taak weergeven: *eTaskGetState();*

- die de omvang van de resterende stack van een taak weergeven, *uxTaskGetStackHighWaterMark();*
- die de prioriteit van een taak weergeven: *uxTaskPriorityGet();*
- en die de naam van een taak weergeven: *pcTaskGetTaskName()*.

Maar FreeRTOS heeft geen functie die de runtime van de taak weergeeft. Deze functie moet in de broncode geïmplementeerd worden. Daarvoor moeten de volgende stappen worden gezet:

1. Via Xilinx moet de parameter *configUSE_TRACE_FACILITY* op '1' worden gezet:

Board Support Package Settings >> freertos >> use_trace_facility;

2. In het bestand *freertos_v2_1_0.tcl* moet de parameter *configGENERATE_RUN_TIME_STATS* op 1 worden gezet*:*

```
xput_define $config_file "configGENERATE_RUN_TIME_STATS" "1"
```

3. In het bestand *FreeRTOS.h* moeten de parameters *INCLUDE_eTaskGetState, INCLUDE_uxTaskGetStackHighWaterMark* op '1' worden gezet:

```
#define INCLUDE_uxTaskGetStackHighWaterMark 1

#define INCLUDE_eTaskGetState 1
```

4. In het bestand *FreeRTOS.h* moeten twee nieuwe functies en een counter voor runtime worden gedefinieerd:

```
extern volatile unsigned long ulHighFrequencyTimerTicks;

 #define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()
(ulHighFrequencyTimerTicks=0UL)

#define portGET_RUN_TIME_COUNTER_VALUE() ulHighFrequencyTimerTicks
```

5. In de broncode van FreeRTOS *port.c* moet in de timer interrupt handle *vPortTickISR()* de variabele-counter worden toegevoegd:

```
ulHighFrequencyTimerTicks ++;
```

6. In het bestand *tasks.c* moet de functie worden gecreëerd:

```
#if ( configGENERATE_RUN_TIME_STATS == 1)
  uint32_t uxTaskGetRunTime(TaskHandle_t xTaskToQuery )
  {
          vTaskSuspendAll();
          TCB_t *pxTCB;
          pxTCB = prvGetTCBFromHandle( xTaskToQuery );
          xTaskResumeAll();
          return pxTCB->ulRunTimeCounter;
  }
  #endif
```

7. In het bestand *task.h* moet deze functie gedefinieerd worden:

```
uint32_t uxTaskGetRunTime(TaskHandle_t xTaskToQuery );
```

Na deze veranderingen in de broncode werden de FreeRTOS-functies en nieuwe functies in de wrapperfunctie aangeroepen.

Tijdens de implementatie voor **MicroC/OS-II:**

- wordt een MicroC/OS-II-functie aangeroepen die alle informatie over de taak geeft: *OSTaskQuery(..)*.

7. De functie die een set creëert: *pr_QueueCreateSet(..)*, de functie die queues aan de set toevoegt: *pr_QueueAddToSet(..*), de functie die een queue vanuit de set selecteert: *pr_QueueSelectFromSet(..)*.

   Tijdens de implementatie voor **FreeRTOS:**

   - worden API- functies aangeroepen die alle informatie over de taak geven: *xQueueCreateSet(..),xQueueAddToSet(..),xQueueSelectFromSet(..)*.

   Voor de implementatie voor **MicroC/OS-II** geldt**:**

   - de set is niet geïmplementeerd in MicroC/OS-II. Tijdens de Ontwerpfase was de implementatie van de set functies in FreeRTOS onderzocht. Op dezelfde manier heb ik de set functies in MicroC/OS-II geïmplementeerd. Daarvoor moesten de volgende veranderingen in de broncode van MicroC/OS-II gedaan worden:

   1. In het bestand *ucos_ii.*h moest aan de definitie van *OS_EVENT* structuur de pointer toegevoegd worden:

   ```
   struct os_event *QueueSetContainer;
   ```

   2. In het bestand *os_q.c*, moest aan de functie *OS_EVENT *OSQCreate (void **start, INT16U size)*, een regel binnen het *if*-statement(*if (pevent != (OS_EVENT *)0*)) worden toegevoegd:

   ```
   pevent->QueueSetContainer = 0u;
   ```

   3. In het bestand *os_q.c* moest aan de functie *OSQPost (OS_EVENT *pevent, void *pmsg)*, voor het laatste aanroepen van *OS_EXIT_CRITICAL();* de code toegevoegd worden:

   ```
   if(pevent->QueueSetContainer!=0u)
   {
           OSQPost(pevent->QueueSetContainer, pevent);
   }
   ```

   Daarna in *wrapper_rtos. c* waren de drie functies geimplementeerd.

1. De functie die een set creëert:

```
prQueueSetHandle pr_QueueCreateSet(prInt32U prSetQueuesLength)
    {
      #ifdef USING_FreeRTOS
            if ( configUSE_QUEUE_SETS == 1 )
                  return xQueueCreateSet(prSetQueuesLength);
      #endif /* USING_FreeRTOS */

      #ifdef USING_microCOS_II
            if(OS_Q_EN > 0u)
            {
                  OS_EVENT* prHandle= 0u;
                  void* set[prSetQueuesLength];
                  prHandle = OSQCreate(&set[0], prSetQueuesLength );
                  return prHandle;
            }
      #endif /* USING_microCOS_II */
      return prCONF_ERROR;
     }
```

2. De functie die een queue in de set toevoegt:

```
prErrorMessage pr_QueueAddToSet(prQueueHandle prQHandle, prQueueSetHandle
prQSHandle)
{
  #ifdef USING_FreeRTOS
        prErrorMessage prError = 0;
        if ( configUSE_QUEUE_SETS == 1 )
        {
                prError =  xQueueAddToSet(prQHandle, prQSHandle);
                if(prError == pdPASS)
                        return prOK;
                else
                        return prERROR;
        }
  #endif /* USING_FreeRTOS */

  #ifdef USING_microCOS_II
        if((OS_Q_ACCEPT_EN > 0u)&&(OS_Q_EN > 0u))
        {
                INT32S  prReturn;
                #if OS_CRITICAL_METHOD == 3u
                OS_CPU_SR  cpu_sr = 0u;
                #endif
                OS_ENTER_CRITICAL();
                if( ( ( OS_EVENT* ) prQHandle )->QueueSetContainer != 0
                {
                        prReturn = prERROR;
                }
                else
                {
                  ((OS_EVENT*)prQHandle)->QueueSetContainer = prQSHandle;
                  prReturn = prOK;
                }

                OS_EXIT_CRITICAL();

                return prReturn;
        }
  #endif /* USING_microCOS_II */
  return prCONF_ERROR;
}
```

3. De functie die de geactiveerde queue select vanuit de set:

```
prQueueSetMemberHandle pr_QueueSelectFromSet(prQueueSetHandle prQSHandle,
                                             prTickType prTimeOUT)
{
      #ifdef USING_FreeRTOS
            if ( configUSE_QUEUE_SETS == 1 )
                  return xQueueSelectFromSet(prQSHandle, prTimeOUT);
      #endif /* USING_FreeRTOS */

      #ifdef USING_microCOS_II
            if(OS_Q_EN > 0u){
                  OS_EVENT *prReturn = 0u;
                  INT8U err = 0;
                  prReturn = OSQPend(prQSHandle, prTimeOUT, &err);
                  return prReturn;
            }
      #endif /* USING_microCOS_II */
      return prCONF_ERROR;
}
```

In deze fase was de wrapper gemaakt. In de volgende fase werden de testen uit het testplan uitgevoerd.

10.4 Iteratie 5: ontwikkeling wrapper. Testfase

In deze fase voerde ik de tests uit die ik tijdens Ontwerpfase had bedacht. Voor het testen van de wrapper gebruikte ik de demoapplicatie. De testresultaten waren voldoende. De beschrijving van de testresultaten staan in Bijlage XX.

11. Invoeringsfase

Dit was de slotfase van mijn onderzoekproject. Tijdens deze fase probeerde ik de sterke en zwakke kanten van de bestudeerde RTOS'en te identificeren en op basis daarvan één systeem te kiezen voor toekomstig gebruik in de COMbricks PA link.

Het project bevatte een theoretisch en een praktisch onderzoek naar bestaande RTOS'en. Uit het theoretische onderzoek kwamen drie RTOS'en naar voren: ThreadX, MicroC/OS-II en OpenRTOS/ FreeRTOS (FreeRTOS is de gratis versie van OpenRTOS). Tijdens het praktische onderzoek viel ThreadX af, vanwege problemen met het verkrijgen van een werkbare trialversie van het systeem voor het testen.

In het praktische deel van het project maakte ik een wrapper voor MicroC/OS-II en FreeRTOS. Tijdens de ontwikkeling van de wrapper moest ik diepgaand onderzoeken hoe deze beide RTOS'en systeemfuncties implementeren. Zo heb ik de voor- en nadelen van MicroC/OS-II en OpenRTOS/ FreeRTOS in kaart kunnen brengen. In dit hoofdstuk noem ik deze voor- en nadelen per systeem.

FreeRTOS

Voordelen:

1. In OpenRTOS/ FreeRTOS is het Round-robin scheduling algoritme gerealiseerd, wat het mogelijk maakt om taken met dezelfde prioriteiten te gebruiken.
2. De functies en structuren van OpenRTOS/ FreeRTOS die het werk van de scheduler kunnen beïnvloeden, zijn afgeschermd. De gebruiker heeft geen toegang tot deze functies vanuit de applicatiecode, wat het systeem beschermt tegen accidentele schade.

Nadelen:

1. De instelling van sommige configuratieparameters vereist aanpassing van de broncode van OpenRTOS/ FreeRTOS.
2. OpenRTOS/ FreeRTOS gebruikt dynamische toewijzing van het geheugen. Zo wijst bijvoorbeeld de API-functie *`xQueueGenericCreate()`* het geheugen aan gecreëerde queues toe met de functie *`malloc()`*, wat tot grote fragmentatie van het geheugen[1] kan leiden; deze functie werkt niet deterministisch[1].
3. De API-functie *`xQueueGenericReceive()`* van FreeRTOS, die elementen vanuit een queue leest, kopieert die van de queue naar het adres dat als parameter was doorgegeven. Daarvoor wordt de functies *memcpy()* gebruikt. Deze functie checkt niet of de grootte van het element minder is dan of gelijk is aan de grootte van de bestemmingsbuffer. Daardoor kan deze functie tot buffer overflow leiden, waardoor het systeem crasht.

MicroC/OS-II

Voordelen:

1. MicroC/OS-II gebruikt statische toewijzing van het geheugen.
2. In MicroC/OS-II zijn geheugenbeheerfuncties geïmplementeerd die het geheugen tegen de fragmentatie beschermen. Deze functies zijn deterministisch.

Nadelen:

1. In MicroC/OS-II is het Round-robin scheduling algoritme niet gerealiseerd, wat het onmogelijk maakt om taken met dezelfde prioriteiten te gebruiken.

2. In MicroC/OS-II is de set van de queues niet gerealiseerd. Deze functie is belangrijk als een taak met twee of meer queues gesynchroniseerd moet worden. Implementatie van de setfuncties vereist grote aanpassingen in de broncode.
3. In MicroC/OS-II is geen functie geïmplementeerd waardoor een taak met vaste periode wordt uitgevoerd. De realisatie van deze functie vereist aanpassingen in de broncode.
4. Alle systeemfuncties en structuren van MicroC/OS-II zijn open. Daardoor heeft de gebruiker vrije toegang tot de systeemfuncties en structuren vanuit applicatie. Dit is gevaarlijk voor de software, want de gebruiker kan daar per ongeluk iets aan veranderen.

Het grootste nadeel van OpenRTOS/FreeRTOS is de dynamische geheugentoewijzing. Daardoor moet tijdens het werk met FreeRTOS de ontwikkelaar heel voorzichtig zijn met het gebruik van functies die een Queue, Semaphore of Mutex creëren.

Het grootste nadeel van MicroC/OS-II is het ontbreken van het Round-robin scheduling algoritme. Tijdens het werk met dit systeem moet de ontwikkelaar opletten dat elk proces is gesynchroniseerd met bepaalde tools of events.

Bij het kiezen van het RTOS moest ook rekening worden gehouden met de prijs van het systeem. FreeRTOS is gratis maar heeft geen technische ondersteuning. OpenRTOS is de commerciële versie van FreeRTOS, heeft technische support en kost €2.625. MicroC/OS-II kost $7.500.

Alle deze omstandigheden besprak ik met de begeleider. Uiteindelijk werd FreeRTOS gekozen om in de COMbricks PA link te integreren. Als er technische ondersteuning nodig is kan het bedrijf eenvoudig naar OpenRTOS overstappen.

12. Conclusie

Ik heb mijn afstudeerproject uitgevoerd bij het bedrijf PROCENTEC. Dit bedrijf is onder meer specialist in PROFIBUS-technologie en producent van de COMbricks PA link. De afstudeeropdracht was het kiezen van het beste bestaande RTOS voor integratie in de bestaande software van de COMbricks PA link.

Aan het begin van het project koos ik een ontwikkelmethode. Daarvoor gebruikte ik de Klassieke Contingentieanalyse. Op basis van deze analyse koos ik Iterative Application Development (IAD) als methode voor het project.

Het project bevatte vijf iteraties. Tijdens de eerste iteratie deed ik theoretisch onderzoek naar de bestaande RTOS'en. Voor dit onderzoek stelde ik eisen en wensen voor de RTOS'en vast met behulp van de MoSCoW-methode. Aan het eind van deze iteratie koos ik een top-drie van RTOS'en: FreeRTOS/OpenRTOS, ThreadX en MicroC/OS-II, die het meest geschikt waren voor integratie.

Tijdens de restende iteraties deed ik praktisch onderzoek naar deze top-drie van RTOS'en.

In de tweede en vierde iteratie ontwikkelde ik de demoapplicatie 1.0 en de demoapplicatie 2.0. Deze demoapplicaties toonden aan dat de drie RTOS'en beschikten over de functionaliteit die voor de COMbricks PA link nodig was.

De derde iteratie besteedde ik aan de implementatie van de seriële interrupt in de demoapplicatie. Dat lukte niet door de FPGA-configuratie. ThreadX viel af tijdens het praktische onderzoek, want het was onmogelijk om een werkbare trialversie van dit systeem te krijgen voor het testen.

In de vijfde iteratie maakte ik een wrapper voor FreeRTOS en MicroC/OS-II. Daarna veranderde ik de functies van de demoapplicatie 2.0 naar wrapperfuncties voor het testen. Vervolgens draaide ik de demoapplicatie eerst onder FreeRTOS en daarna onder MicroC/OS-II.

Tijdens het praktische onderzoek naar de RTOS'en moest ik FreeRTOS en MicroC/OS-II diepgaander bestuderen. Zo bepaalde ik de voor- en nadelen van MicroC/OS-II en FreeRTOS. Op basis van deze nadelen en voordelen, en rekening houdend met de prijs van RTOS'en, koos ik FreeRTOS als het meeste geschikte systeem voor integratie in de COMbricks PA link software.

13. Evaluatie

Het doel van dit hoofdstuk is de beoordeling van het werkproces. Hier evalueer ik de uitgevoerde werkzaamheden.

Om de doelen van dit project te bereiken had ik op basis van de Klassieke Contingentieanalyse als methode Iterative Application Development (IAD) gekozen. Aan het begin van project had ik twee iteraties gepland. De eerste iteratie was voor het theoretische onderzoek en de tweede iteratie was voor het praktische onderzoek.

Tijdens de ontwikkeling van het product bleek de gekozen methode daar niet goed bij te passen. Aan het begin van het praktische onderzoek werd duidelijk dat één iteratie niet genoeg was voor het praktische onderzoek, want tijdens het praktische onderzoek moesten vier tussenproducten worden ontwikkeld. Daarom verhoogde ik aantal iteraties naar vijf. Elke iteratie moest een bepaald product opleveren:

1. Iteratie 1: keuze van top-drie RTOS'en
2. Iteratie 2: demoapplicatie 1.0
3. Iteratie 3: demoapplicatie 1.0 met seriële interrupt
4. Iteratie 4: demoapplicatie 2.0
5. Iteratie 5: wrapper voor top-drie RTOS'en

De tussenproducten voldeden bij oplevering niet allemaal:

1. De demoapplicatie 1.0 met seriële interrupt heb ik niet geïmplementeerd, want de implementatie van de seriële interrupt vereiste veranderingen in de FPGA-configuratie. Deze veranderingen moest een collega uitvoeren, maar die was te druk met andere projecten om hier tijd voor te hebben.
2. De wrapper heb ik alleen voor twee RTOS'en gemaakt: MicroC/OS-II en FreeRTOS. Het derde systeem ThreadX viel in het praktische onderzoek af, want om de trialversie van ThreadX te krijgen moest het bedrijf bepaalde verplichtingen op zich nemen. Deze verplichtingen brachten risico's voor het bedrijf mee, wat onwenselijk was.

Het doel van het afstudeerproject was het kiezen van een geschikt RTOS om in de bestaande software van de COMbricks PA link te integreren. Dit doel is bereikt. De bestaande RTOS'en zijn onderzocht. Op basis van dit onderzoek zijn de nadelen en voordelen van de systemen naar voren gekomen. Daarna is op basis van de analyse van nadelen en voordelen FreeRTOS/OpenRTOS als meest geschikte systeem gekozen.

Tijdens mijn studie heb ik een aantal competenties verworven. Veel van deze competenties kwamen terug in het werkproces aan de stageopdracht.

Aan het begin van het project had ik een Plan van Aanpak geschreven. Hierin legde ik de risico's, de grenzen van het project, de activiteiten die ik moest uitvoeren en de beoogde eindresultaten vast. Hiervoor was competentie G1 belangrijk.

G1 Praktische aspecten hanteren projecten

In de analysefase van elke iteratie heb ik het probleemdomein van de stageopdracht geanalyseerd. Daarvoor had ik gespreken met de bedrijfsbegeleider, om de behoeften van het bedrijf te weten te komen. Daarna onderzocht ik de functionaliteit van de COMbricks PA link. Op basis van deze

gesprekken en het onderzoek heb ik de functionele en niet-functionele eisen voor de demoapplicaties en de wrapper vastgesteld. Hierbij heb ik de competenties A1, A3 en A5 laten zien.

A1 Analyseren van het probleemdomein

A3 Achterhalen van behoeften van belanghebbenden

A5 Opstellen van systeemeisen

In de ontwerpfases van het project bedacht ik de logica van de demoapplicaties en de wrapper. Op basis van de eisen tekende ik de Yourdon-diagrammen. Deze diagrammen gebruikte ik vervolgens tijdens de implementatie van de demoapplicaties. Hiermee heb ik competentie C8 laten zien.

C8 Ontwerpen van een technisch informatiesysteem

In de implementatiefases van het project heb ik de demoapplicatie 1.0, de demoapplicatie 2.0 en de wrapper geïmplementeerd. Hierbij heb ik de competentie D16 laten zien.

D16 Het realiseren van software

Het grootste deel van het project bestond uit het testen van de ontwikkelde software. Daarvoor heb ik het testplan gemaakt, de verwachtte resultaten bepaald en de testresultaten geanalyseerd. Hier heb ik competentie D17 laten zien.

D17 Testen van softwaresystemen

12. Bronnenlijst

1. Jean J.Labrosse. MicroC/OS-II. The real-time kernel. Second Edition. USA, CMP Books, 2002.
2. William Stallings. Operating systeems. Vierde editie. Vertaling: Siem de Wit. Engeland, Pearson Education Benelux, 2003.
3. Derek J.Hatley, Imtiaz A Pirbhai. Strategies for Real-Time System Specification. USA, Dorset House Publishing Co., 1987, p. 41-74.
4. Stephen J.Mellor, Paul T.Ward. Structured development for Real-Time Systems. USA, Prentice-Hall, 1986, p. 37-76.
5. Travis Swicegood. Pragmatic Guide to Git. USA, PragmaticProgrammers, 2010, p. 4-52.
6. Richard Barry. FreeRTOS Reference Manual - API Functions and Configuration Options. Real Time Engineers Ltd. 2014.
7. MULTI: Developing for ThreadX. USA, Green Hills Software, Inc., 2005.
8. Edward L. Lamie. Real-Time Embedded Multithreading Using ThreadX. Second Edition. USA, Newnes, 2009.
9. Xilinx. LogiCORE IP MicroBlaze Micro Controller System (2012). Geraadpleegd op maart 2014 via http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf
10. Ufa State Aviation Technical University (2009). Real-Time Systems. Hoorcolleges. Geraadpleegd op februari 2014 via http://www.studfiles.ru/dir/cat32/subj528/file13346.html
11. Martin Hoffmann, Christoph Borchert (2014). Effectiveness of Fault Detection Mechanisms in Static and Dynamic Operating System Designs. Geraadpleegd op maart 2014 via https://www4.cs.fau.de/Publications/2014/hoffmann_14_isorc.pdf
12. Faheem Sheikh, Dan Driscoll. Mentor Graphics (2009). Measuring RTOS performance:WHAT? WHY? HOW? Geraadpleegd op maart 2014 via http://www.fortronic.it/user/file/A%26VElettronica/MeasuringRTOS_mentorpaper_69305.pdf
13. Gökhan Ugurel, Cüneyt F. Bazlamaçcı (2011). Context Switching Time and Memory Footprint Comparison of Xilkernel and µC/OS-II on MicroBlaze. Geraadpleegd op maart 2014 via http://www.emo.org.tr/ekler/00a7192d9abd186_ek.pdf
14. IAR Systems (2011). How to choose an RTOS. Geraadpleegd op februari 2014 via http://www.iar.com/Global/Resources/Seminars/How_to_choose_an_RTOS.pdf
15. Andrey Kurniz(2011). FreeRTOS - operating system for microcontrollers. Geraadpleegd op april 2014 via http://kit-e.ru/assets/listalka/Kurniz/Kurniz.pdf
16. Samson (1999). PROFIBUS-PA. Geraadpleegd op februari, april, mei 2014 via http://www.samson.de/pdf_en/l453en.pdf
17. http://en.wikipedia.org/wiki/Profibus Geraadpleegd op februari 2014.
18. Xilinx(2014).Spartan-6 FPGA Family. Geraadpleegd op februari 2014 via http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/
19. http://nl.wikipedia.org/wiki/MicroBlaze Geraadpleegd op februari 2014.
20. http://rtos.com/products/threadx/MicroBlaze Geraadpleegd op februari, mei 2014.
21. http://www.freertos.org/ Geraadpleegd op februari, maart, april, mei 2014.
22. http://micrium.com/rtos/ Geraadpleegd op februari, mei 2014.
23. http://www.highintegritysystems.com Geraadpleegd op februari 2014.

24. Michael Barr**(2002).** Special Report: Choosing an RTOS. Geraadpleegd op februari 2014 via http://www.embedded.com/electronics-blogs/other/4024563/Special-Report-Choosing-an-RTOS
25. Colin Walls (2009). Choosing the right RTOS: A life or death decision. Geraadpleegd op februari 2014 via http://embedded-computing.com/articles/choosing-right-rtos-life-death-decision/

Bijlage I. Begrippenlijst

Big Endian – "*Big-endian systems store the most significant byte of a word in the smallest address and the least significant byte is stored in the largest address*."[1]

FIFO – "*is an acronym for First In, First Out, a method for organizing and manipulating a data buffer, where the oldest (first) entry, or 'head' of the queue, is processed first. It is analogous to processing a queue with first-come, first-served behaviour: where the people leave the queue in the order in which they arrive*."[2]

LIFO – *"(last in, first out) refers to the way items stored in some types of data structures are processed. By definition, in a LIFO-structured linear list, elements can be added or taken off from only one end"*[3]

Little Endian – "*Little-endian systems store the least significant byte in the smallest address*."[4]

Preemptive algoritme – "*de processen worden onderbroken tijdens hun uitvoer zodat de scheduler een ander proces kan hervatten. De processen kunnen onderbroken worden.*"[5]

Round-Robin scheduling algoritme -"*is one of the algorithms employed by process and network schedulers in computing. As the term is generally used, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive).*"[6]

TCB-blok – "*Task Control Block, is a data structure in the operating system kernel containing the information needed to manage a particular process*."[7]

Yourdon-diagram - het is een data flow diagram. "*Het Data Flow Diagram (DFD), is een grafische representatie van de gegevensstroom, de dataflow, door een informatiesysteem. Het is een veel gebruikte techniek binnen de gestructureerde analyse.*"[8]

UART-poort – "*A universal asynchronous receiver/transmitter..., is a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as EIA, RS-232, RS-422or RS-485.*
The universal designation indicates that the data format and transmission speeds are configurable. The electric signaling levels and methods (such as differential signaling etc.) are handled by a driver circuit external to the UART."[9]

---

[1] http://en.wikipedia.org/wiki/Endianness 28 juni 2014
[2] http://en.wikipedia.org/wiki/FIFO 22 mei 2014
[3] http://en.wikipedia.org/wiki/LIFO_(computing) 22 mei 2014
[4] http://en.wikipedia.org/wiki/Endianness 28 juni 2014
[5] http://nl.wikipedia.org/wiki/Scheduling 9 maart 2014
[6] https://en.wikipedia.org/wiki/Round-robin_scheduling 9 maart 2014
[7] http://www.ustudy.in/node/5473 28 augustus 2014
[8] http://nl.wikipedia.org/wiki/Data_flow_diagram 12 april 2014
[9] http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter 20 september 2014

Bijlage II. Het Plan van Aanpak

Plan van Aanpak

Afstudeerstageproject

PROCENTEC

De Haagse Hogeschool

Maria Brodskaya 11104945@hhs.nl

15-02-2014

Delft

1. Achtergrond

De opdrachtgever van het stageproject is het bedrijf PROCENTEC. PROCENTEC specialiseert zich in alle producten en diensten van Profibus- en Profinettechnologieën. Profibus (Process Field Bus) is een standaard voor veldbus communicatie in de automatiseringstechniek. Profibus wordt gebruikt om automatiseringsapparatuur zoals sensoren, controllers, in één systeem te combineren. Profinet is een standaard voor industriële automatisering met het gebruik van het computernetwerk. Het bedrijf ontwikkelt producten om de productieprocessen van eindgebruikers te optimaliseren. PROCENTEC maakt componenten die nodig zijn om een industrieel meet- en stuurbaar netwerk in te richten. De producten van PROCENTEC zijn toepasbaar op allerlei installaties zowel de fabrieks- als utiliteitsvloer. PROCENTEC biedt ook training en ondersteuning aan de eindgebruikers.

PROCENTEC heeft twee kantoren: het hoofdkantoor in Wateringen en een verkoopkantoor in Duitsland. In het hoofdkantoor werken ongeveer 30 mensen. Er zijn vijf afdelingen: research & development-, verkoop-, financies-, administratie- , productieafdeling en testlaboratorium. Het bedrijf oriënteert zich op de markt zowel binnen Nederland als buiten. Zij heeft officiële distributeurs in negen landen.

De opdrachtgever is Research & Development afdeling van PROCENTEC. Het hoofd van deze afdeling is Paulo Silva, hij is de bedrijfsmentor van dit project. De bedrijfsbegeleider van het afstudeerproject is Edward Dumay. Hij is ingenieur van de R&D afdeling van het bedrijf.

De opdrachtnemer van het project is de Haagse Hogeschool. Dit project wordt door de vierde jaren studente van de Haagse Hogeschool en stagiair van PROCENTEC – Maria Brodskaya uitgevoerd. De projectbegeleider van dit project vanuit de school is meneer Smeets.

Het doel van dit document is het project definiëren, het duidelijk maken van de behoeften van de opdrachtgever, de scope van het project en de planning van de werkzaamheden van dit project bepalen. In dit document wordt de opdracht van het project geformuleerd, de activiteiten en producten die tijdens het project moeten worden uitgevoerd, worden vastgesteld. Daarna worden de projectgrenzen, randvoorwaarden en risico’s bepaald.

2. Projectopdracht en doelstellingen

In dit hoofdstuk wordt het doel van het project beschreven en de opdracht van het project geformuleerd. De doelstelling geeft aan welke problemen de opdrachtgever met dit project wil oplossen. De projectopdracht beschrijft wat er moet worden gedaan om deze problemen op te lossen.

2.1 Doelstellingen

Een van de productenline die door het bedrijf PROCENTEC is ontwikkeld, is COMbricks. De COMbricks maakt de naadloze communicatie op hoge snelheid tussen PROFIBUS DP en PROFIBUS PA mogelijk. PROFIBUS DP (Decentralized Peripherals) en PROFIBUS PA (Process Automation) zijn verschillende communicatieprotocollen. De communicatieprocessen COMbricks PA link zijn sequentieel op de datalinklaag. Wanneer meerdere veldbussen via één COMbricks communiceren, ontstaat er een vertraging in het systeemwerk, omdat alle processen op datalinklaag sequentieel zijn uitgevoerd. Daarom wil PROCENTEC in de software van COMbricks een standaard Real-Time Operating System integreren, dat mogelijkheid biedt om de processen te parallelliseren. Hieruit komt het doel van de afstudeeropdracht – het kiezen van een bestaande Real Time Operating Systeem op basis van het onderzoek en aan de hand ervan een demoapplicatie maken. Het systeem moet aan de eisen van PROCENTEC voldoen.

2.2 Projectopdracht

De opdracht van het project is het onderzoeken van de bestaande Real-Time Operating Systeem op basis van de vastgestelde eisen. Daarna moet de demoapplicatie met de gekozen Real Time Operating Systeem voor COMbricks worden geschreven. De demoapplicatie moeten gecontroleerd en getest worden.

De resultaten van de afstudeerstageopdracht moeten zijn:

4. Het onderzoeksrapport van de bestaande Real Time Operating Systemen
5. De demoapplicatie voor het splitsen van de processen van de COMbricks PA Link

Alle resultaten moeten worden bereikt binnen 20 weken. De code van de demoapplicatie moet duidelijke commentaar bevatten.

3.Projectactiviteiten

Tijdens het project moeten bepaalde activiteiten worden uitgevoerd. Deze activiteiten omvatten diverse typen van werkzaamheden zoals het onderzoeken, het schrijven van documenten, het verzamelen en bestuderen van informatie, het programmeren, het testen en het bespreken van het project met de begeleiders.

1. **Documenten**

a. Plan van Aanpak
b. Verslag

2. **Verzamelen en bestuderen van informatie**

a. Het onderzoek en de analyse van de bestaande Real Time Operating Systemen en hun mogelijkheden
b. Het onderzoeken van de Microblaze processor
c. Het onderzoeken van de Profibus PA protocol
d. Het onderzoeken van de Profibus DP protocol
e. Het onderzoeken van de COMbricks functies
f. Het vaststellen van de eisen voor een Real Time Operating Systeem

3. **Programmeren**

a. Het maken van demoapplicatie die de werking van de features van het geselecteerde Real Time Operating Systeem, die nodig zijn voor de COMbricks PA Link aantonen

4. **Testen**

a. Het testen van demoapplicatie

5. **Bespreken**

a. Vergaderingen met stagebegeleider – meneer Smeets
b. Vergaderingen met bedrijfsstagebegeleider – meneer Dumay

6. **Presentatie**

a. De eindpresentatie

4. Projectgrenzen en randvoorwaarden

Het afstudeerproject start op 10 februari 2014, de deadline van het project is 27-06-2014. De projectontwerper moet 40 uur per week aan het project besteden.

4.1 Projectgrenzen

De resultaten van het project worden in drie categorieën verdeeld: de resultaten die binnen de projectgrenzen, op de grenzen en buiten de grenzen van het project liggen. De verdeling in deze categorieën is op de volgende principes gebaseerd:

- Binnen de grenzen van het project liggen de resultaten die binnen 20 weken met beloofde kwaliteit moeten worden bereikt.
- Op de grenzen van het project liggen de resultaten die erg wenselijk, maar afhankelijk van bepaalde voorwaarden zijn, zoals regels en financiering. Ze kunnen onder bepaalde voorwaarden worden uitgevoerd.
- De resultaten die buiten de grenzen van het project liggen, worden uitgevoerd als er tijd genoeg voor is.

Binnen de grenzen:

1. Plan van Aanpak

2. Het kiezen van een Real Time Operating Systeem

3. Verslag

4. De demoapplicatie, die de werking van de features van het geselecteerde Real Time Operating Systeem die nodig zijn voor de COMbricks PA Link aantonen

5. Testen van de demoapplicatie

Op de grenzen:

Een start maken met het implementeren van het geselecteerde Real Time Operating Systeem in de software van de COMbricks PA link.

Buiten de grenzen:

De volledige implementatie van het gekozen Real Time Operating Systeem in de software van de COMbricks PA link.

4.2 Randvoorwaarden

De uitvoering van het project vereist een aantal hard- en software.

Hardware:

1. PC

2. Internet toegang

3. Het boord met Microblaze processor

Software:

1. Windows operating system

2. Installatie van het geselecteerde RTOS

3. Eclipse-based IDE

5. Producten

Tijdens het afstudeer project moeten de volgende producten ingeleverd worden:

Bij het bedrijf:

1. Plan van Aanpak
2. Verslag
3. De demoapplicatie, die de werking van de features van het geselecteerde Real Time Operating Systeem die nodig zijn voor de COMbricks PA Link aantonen

Bij de school:

1. Plan van Aanpak

2. Verslag

3. Eindpresentatie

6. Kwaliteit

Een van de activiteiten van de afstudeerstageopdracht is het vaststellen van de eisen voor het RTOS. Deze eisen worden gebruikt om de kwaliteit van het eindresultaat te beoordelen. Het geselecteerde RTOS en de demoapplicatie moeten aan de vastgestelde eisen voldoen.

Om de correcte eisen voor het RTOS en prototypen te formuleren zullen de volgende stappen worden genomen:

1. De functionaliteit van de COMbricks PA link moet worden onderzocht om te bepalen welke eisen ze van het RTOS vereist
2. De eisen worden met de ontwikkelaars van de COMbricks PA link besproken
3. De eisen worden met de begeleider en/of bedrijfsmentor besproken.

Voor de controle van de kwaliteit van de demoapplicatie wordt het testplan geschreven. De verwachte testresultaten worden bepaald door de eisen van het systeem. De ontwikkelde prototypen worden volgens dit testplan getest en de gevonden fouten worden geanalyseerd en verbeterd.

7. Projectorganisatie

Het afstudeerproject gaat 20 weken door, de deadline van het project is 27 juni 2014. De opdrachtgever is het bedrijf PROCENTEC - http://www.procentec.com/. Dit project wordt uitgevoerd door de studente van de Haagse Hogeschool Maria Brodskaya. Zij gaat in het bedrijf PROCENTEC haar afstudeerstage lopen. Tussen het bedrijf en de stagiair wordt een overeenkomst gesloten. De stagiair moet 800 uren aan het project besteden – 40 uur per week. Hieronder staan de persoonlijke gegevens.

Naam	E-mail adres	mobile
Maria Brodskaya	maria.brodskaya@gmail.com 11104945@hhs.nl	062-4701050

De begeleiders van dit project zijn meneer Smeets aan de kant van de school en meneer Dumay aan de kant van het bedrijf.

Tijdens de projectontwikkeling gaat de communicatie tussen de projectontwerper en de stagebegeleider vanuit de school meneer Smeets via e-mail door. Bij de Haagse Hogeschool moeten de volgende producten worden ingeleverd: Plan van Aanpak, Verslag. Deze documenten moeten door hem worden goedgekeurd.

Wekelijks moet het project met de bedrijfsbegeleider - meneer Dumay, worden besproken. Bij het bedrijf moeten de volgende producten worden ingeleverd: het Plan van Aanpak, het Onderzoeksrapport, demoapplicatie. Het Plan van Aanpak moet door het bedrijf worden goedgekeurd en ondertekend.

8. Ontwikkelmethode

In dit hoofdstuk wordt een aantal factoren van het project geanalyseerd met behulp van de contingentieanalyse. Op basis van dit analyse wordt de strategie van de ontwikkeling van het project bepaald en dan de ontwikkelmethode gekozen.

8.1 Strategieanalyse

Bij de keuze van mijn ontwikkelmethode paste ik de Klassieke Contingentieanalyse[1] toe om de risico's van het project te minimaliseren. Ik analyseerde de volgende situationele factoren:

- **de projectgrootte**;
  Hierbij ging het om factoren als de kosten van het project, de duur van het project en het aantal betrokken ontwikkelaars.
- **de mate van gestructureerdheid**;
  Dit is de mate waarin de oplossing van het probleem in een procedure of programma kan worden beschreven. In dit geval lagen de eisen waaraan de demoapplicatie moest voldoen, vooraf niet vast; ze hingen af van de uitkomsten van het theoretische onderzoek. De enige onderzoeksvraag was: welk RTOS is voor PROCENTECs doelen het meest geschikt?
- **taakinzicht bij de gebruikers**;
  De demoapplicaties die tijdens het project werden ontwikkeld, waren bedoeld voor de software-ingenieurs van PROCENTEC.
- **deskundigheid van de ontwikkelaar in het probleemdomein**;
  Voordat ik met dit afstudeerproject begon, had ik geen ervaring met PROFIBUS.
- **de veranderlijkheid van de probleemruimte**;
  De onderzoeksvraag van het project lag vanaf het begin vast: welk RTOS is voor PROCENTECs doelen het meest geschikt?

De resultaten van de analyse staan in tabel 4.1.

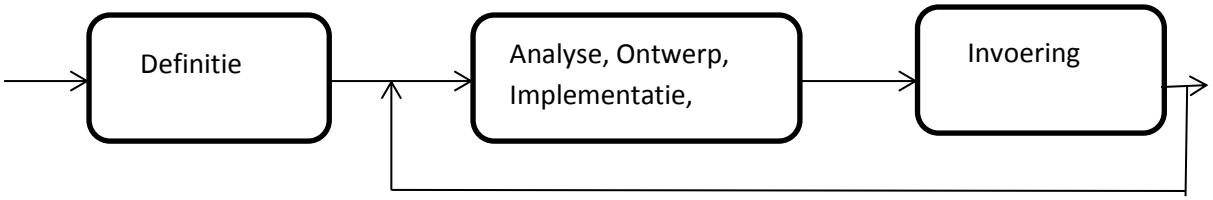
Tabel 4.1 Factoren in het afstudeerproject volgens de Klassieke Contingentieanalyse

Soort	Gradatie	Aandeel onzekerheid
Projectgrootte	Klein	Laag
Mate van gestructureerdheid	Zwak gestructureerd	Hoog
Taakinzicht gebruikers	Volledig	Laag
Deskundigheid ontwikkelaars in het probleemdomein	Laag	Hoog
Veranderlijkheid probleemruimte	Laag	Laag

Het project had een onzekerheid van 40% , dus de optimale strategie voor dit project was de iteratieve strategie. De definitiefase hoefde niet iteratief te zijn, want de uitgangsvraag en de probleemruimte veranderden niet. De invoeringsstrategie is in de loop van het project wel geëvolueerd, zie schema 4.1. Ik ben uitgekomen op meer dan de twee geplande iteraties.

[1] Systeemontwikkelmethoden. Sheets van Henk van den Bosch.

Schema 4.1 Evolutionaire invoering



De conclusies uit de analyse van de opdracht kwamen terug in het Plan van Aanpak: zie bijlage I. Ik heb dit Plan van Aanpak ter goedkeuring voorgelegd aan de opdrachtgever.

8.2 Iteratieve Application Development - IAD

IAD is de iteratieve ontwikkelmethode, die voor grote systemen met onstabiele, en niet-gestructureerde probleemruimte geschikt is. IAD heeft drie iteratieve fasen van de ontwikkeling:

- *Definitiefase*
- *Pilotontwikkelingsfase:*
    - *Analyse*
    - *Ontwerp*
    - *Implementatie*
    - *Testen*
- *Invoeringsfase*

Tijdens de *Definitiefase* worden de doelen van het ontwikkelde systeem onderzocht en de eisen van de opdrachtgever genoemd. Hier worden ook de projectgrenzen en randvoorwaarden bepaald. De volgende *Pilotontwikkelingsfase* bevat vier stappen: *Analyse, Ontwerp, Implementatie* en *Testen.* In de *Analysefase* worden systeemeisen gedetailleerd en tot het deelsysteem verwerkt. In de *Ontwerpfase* worden de deelsystemen uitgewerkt. In de *Implementatiefase* en *Testfase* moet het deelsysteem geprogrammeerd en getest worden. De laatste fase van de IAD-methode is de *Invoeringsfase*, waarin het ontwikkelde systeem operationeel zal worden gemaakt.

9. Planning

Om het project te ontwikkelen wordt de Iteratieve Application Development methode gekozen. Volgens deze methode wordt het project in drie delen opgesplitst: definitiefase, pilotontwikkelingsfase en invoeringsfase. Respectievelijk ervan wordt de globale planning van het project gebouwd, zie Tabel 2. Verder wordt de wekelijkse planning gemaakt.

Tabel 2 Planning

Fase	Aantal Weken	Inhoud	Activiteiten
Definitiefase	2	In deze fase moet het doel van het project worden vastgesteld, risico's en grenzen worden bepaald en de globale eisen van het systeem geformuleerd.	• Plan van Aanpak schrijven • Plan van het werkproces maken • Vergaderingen met stagebegeleider vanuit het bedrijf • Onderzoek van de niet-functionele kenmerken van RTOS
Pilotontwikkelingsfase: • Analyse • Ontwerp • Implementatie • Testen	15	In deze fase moet het onderzoek van een bestaande RTOS, Microblaze processor worden uitgevoerd. De globale eisen moeten worden verwerkt in functionele en niet-functionele eisen voor het systeem. Op basis van de eisen moet het RTOS worden gekozen. De architectuur voor de demoapplicatie moet worden gebouwd. De demoapplicatie moet gedeeltelijk worden uitgewerkt en getest. Tijdens deze fase moet de demoapplicatie geprogrammeerd worden, verschillende soorten testen worden uitgevoerd, de gevonden fouten worden verbeterd.	• Verzamelen en bestuderen van informatie over RTOS, Microblaze, COMbricks • Het vaststellen van de niet-functionele- en functionele eisen voor RTOS • Het kiezen van het RTOS • Testcode voor het gekozen RTOS schrijven • Het schrijven van het onderzoeksrapport • Het schrijven van het concept van het verslag • Vergaderingen met begeleider en bedrijfsmentor • Demoapplicatie schrijven • Het testen van de demoapplicatie
Invoeringsfase	3	Hier moeten alle documenten worden afgemaakt. Nogmaals moet een systeemtest uitgevoerd worden. Er moet een eindpresentatie over het project worden gegeven.	• Testen • Verslag afmaken • Presentatie maken

10. Risico's

In dit hoofdstuk worden de risico’s van het project bepaald. De risico’s worden in de interne en externe risico’s opgesplitst. Interne risico’s zijn direct afhankelijk van de projectactiviteiten en kunnen worden verminderd. Externe risico's zijn met betrekking tot externe omstandigheden en kunnen in geringe mate gecontroleerd worden.

Interne risico’s

Definitieve deadline

Er bestaat een risico dat het project niet binnen de vastgestelde tijd kan worden afgerond. Om deze risico te voorkomen wordt de optimale ontwikkelmethode gekozen op basis van de contingentieanalyse. Dan wordt de planning vastgesteld.

Onvoldoende competenties

De Profibus- en Profinettechnologieen zijn helemaal nieuw voor de student. Er bestaat een risico dat door onvoldoende kennis het eindresultaat niet gaat voldoen aan de gestelde kwaliteit. Om deze risico te voorkomen wordt een iteratieve methode voor het afstudeerproject gekozen. Dankzij dat kan de kwaliteit van het product in elke iteratie worden gecontroleerd.

Externe risico’s

Afhankelijkheid van de hardware

Het project is zeer afhankelijk van de hardware. Defecte hardware kan de reden zijn van vertraging van het project. Om de invloed van dit risico te verminderen moet de opdrachtgever onmiddellijk ter kennis worden gesteld.

Bijlage III. WBS-planning

	Projectfases	WBS-koppeling			Activiteiten	Tijd
		Niveau 2	Niveau 3	Niveau 4	Niveau 5	In weken
Iteratie 1	Definitiestudie	Aanpak van het project			- Opdracht formuleren; - Planning maken; - Ontwikkelingsmethode kiezen; - PvA schrijven; -de algemene functionaliteit van PA-protocol onderzoeken; De algemene functionaliteit van de COMbrocks onderzoeken;	1
	Definitiefase	Het onderzoek naar de functionaliteiten van de RTOS-en	Het opbouwen van de kennis van RTOS-en door Internet		- Algemene informatie over RTOS-en zoeken; -Informatie over de RTOS-en op websites van de RTOS-ontwikkelaars vinden; -forums lezen;	2
			Het lezen van de boeken over RTOS-en		-Onderzoeken welke boeken er zijn over RTOS-en; -Meest populaire boeken doorlezen; -Belangrijke kenmerken van de RTOS-en bepalen	
	Analysefase	Het opstellen van de eisen/wensen	Het vaststellen van niet-functionele eisen voor een RTOS			1
			Het vaststellen van functionele eisen voor een RTOS			
	Ontwerpfase	Het opstellen van de eisen/wensen	Het verdelen van de functionele- en niet-functionele eisen op vier categorieën met behulp van MoSCoW methode	Probleemdomein analyseren	-Onderzoeksrapport schrijven;	1
				De COMbricks functionaliteit analyseren		
	Implementatief ase	Het onderzoek naar de RTOS-en	Het onderzoek naar RTOS-en voor alle eisen/wensen	Het onderzoek naar RTOS-en die aan de eisen in de categorie Must Have voldoen	-Onderzoeksrapport schrijven; -Het concept van het verslag beginnen - Contact opnemen met de ontwikkelaars van de RTOS-en; -De voldoeningskwaliteiet aan de eisen analyseren; -De resultaat van het theoretische onderzoek met de begeleider bespreken; -Verslag schrijven	1
				Het onderzoek naar RTOS-en die aan de eisen in de categorie Should Have voldoen		
				Het onderzoek naar RTOS-en die aan de eisen in de categorieën Could Have en Won't Have voldoen		
				Kwaliteitsbeoordeling van de voldoening aan de eisen/wensen		
				De drie top-RTOS-en bepalen		
Iteratie	Analysefase	Het onderzoek naar de RTOS-en	Het ervaringsonderzoek met de drie top-RTOS-en	De features vaststellen die voor deCOMbricks PA-link nodig zijn	-De bestaande code van de COMbricks PA-link onderzoeken; -De features van het RTOS met de begeleider bespreken; -Verslag- en onderzoeksrapport schrijven; - Kennismaken met Xilinx IDE;	1
				Vaststellen van de features die getest gaan worden		
				De factoren bepalen die tijdens de testen onderzocht worden		
	Ontwerpfase	Het onderzoek naar de RTOS-en	Het ervaringsonderzoek met de drie top-RTOS-en	De design van de demoapplicaties maken	-Use-cases maken; -Sequentiediagrammen maken; -Wrapper logica bedenken -Verslag- en onderzoeksrapport schrijven;	3
				De design maken van de wrapper		
	Implementatief ase	Het onderzoek naar de RTOS-en		Implementeren van de demoapplicaties	- Integreren de gratis versie van het RTOS in Xilinx IDE; -Implementeren van de demoapplicaties met de gratis versie van het RTOS; -Integreren van de drie top-RTOS-en in Xilinx IDE; - De testapplicatie implementeren met de drie top-RTOS-en ; -Wrapper implementeren; -Verslag afmaken;	7
				Wrapper voor RTOS maken		

2	Testen	Het onderzoek naar de RTOS-en	De keuze één van de drie top-RTOS-en	Testen uitvoeren De analyse van de implementatie RTOS maken	-Testplan maken; -Checklist maken; -Testen uitvoeren; -De resultaten analyseren	2
	Invoeringsfase	Het onderzoek naar de RTOS-en	De keuze één van de drie top-RTOS-en	De keuze van het RTOS maken op basis van de analyse	-De analyse documenteren; - De keuze met de begeleider bespreken;	1 week

Bijlage IV. De lijst met de bestaande RTOS-en

N	RTOS	Officiële website
1	**Abassi**	http://www.code-time.com/products.html
2	**AMOS**	http://www.alphamicro.com/
3	**AMX RTOS**	http://www.kadak.com/rtos/rtos.htm
4	**uKOS**	http://www.ukos.ch/
5	**ARTOS**	http://www.locamation.nl/solutions/substation-automation-hmv/products
6	**Atomthreads**	http://atomthreads.com/
7	**AVIX**	http://www.avix-rt.com/
8	**BeRTOS**	http://www.bertos.org/
9	**BRTOS**	https://code.google.com/p/brtos/
10	**CapROS**	http://www.capros.org/
11	**ChibiOS/RT**	http://www.chibios.org/dokuwiki/doku.php
12	**ChorusOS**	http://www.oracle.com/technetwork/documentation/legacy-op-sys-193044.html#os
13	**ChronOS**	http://chronoslinux.org/wiki/Main_Page
14	**CMX RTOS**	http://www.cmx.com/
15	**CoActionOS**	http://coactionos.com/
16	**cocoOS**	http://www.cocoos.net/
17	**Concurrent CP/M**	http://www.cpm.z80.de/
18	**Concurrent DOS**	--
19	**Contiki**	https://www.sics.se/contiki
20	**CooCox CoOS**	http://www.coocox.org/CoOS.htm
21	**Deos**	http://www.ddci.com/products_deos.php
22	**DioneOS**	http://www.elesoftrom.com.pl/en/os/index.php
23	**DSP/BIOS**	http://www.ti.com/tool/dspbios
24	**DSPnano RTOS**	http://www.rowebots.com/products/dspnano_rtos
25	**DuinOS**	https://code.google.com/p/duinos/
26	**eCos**	http://ecos.sourceware.org/
27	**eCosPro**	http://www.ecoscentric.com/ecos/ecospro.shtml
28	**embOS**	http://www.segger.com/embos.html
29	**Embox**	https://code.google.com/p/embox/
30	**Embkernel**	http://sourceforge.net/p/embkernel/wiki/Home/
31	**ERIKA Enterprise**	http://erika.tuxfamily.org/drupal/
32	**EUROS**	http://www.euros-embedded.com/v3/index.php/en/
33	**EROS**	http://www.eros-os.org/eros.html
34	**Femto OS**	http://www.femtoos.org/
35	**FreeOSEK**	http://opensek.sourceforge.net/
36	**FreeRTOS**	http://www.freertos.org/
37	**FunkOS**	http://funkos.sourceforge.net/
38	**HeartOS**	http://www.ddci.com/products_heartos.php
39	**Helium**	http://helium.sourceforge.net/
40	**HP-1000/RTE**	http://www.hp.com/products1/rte/tech_support/documentation/
41	**Hybridthreads**	http://hthreads.csce.uark.edu/wiki/About_Hthreads
42	**INTEGRITY**	http://www.ghs.com/products/rtos/integrity.html
43	**IntervalZero RTX**	http://www.intervalzero.com/
44	**INtime**	http://www.tenasys.com/tenasys-products/intime-rtos-family/overview-rtos
45	**ITRON, µITRON**	http://www.t-engine.org/
46	**ISIX**	http://bryndza.boff.pl/index.php?dz=rozne&id=isixrtos
47	**ioRTOS**	http://www.spaciode.com/
48	**iRTOS**	http://irtos.sourceforge.net/
49	**KolibriOS**	http://kolibrios.org/nl/
50	**Lepton**	https://code.google.com/p/lepton/

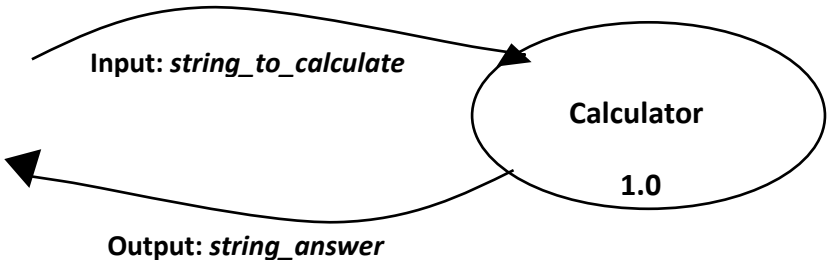
51	**LithOS**	http://www.xtratum.org/
52	**LynxOS**	http://www.lynuxworks.com/rtos/
53	**MaRTE OS**	http://marte.unican.es/
54	**MenuetOS**	http://www.menuetos.net/
55	**Micrium µC/OS-II**	http://micrium.com/
56	**Micrium µC/OS-III**	http://micrium.com/
57	**Microsoft Invisible Computing (MMLite)**	http://research.microsoft.com/en-us/um/redmond/projects/invisible/
58	**mLithOS**	http://www.xtratum.org/
59	**MQX**	http://www.freescale.com/webapp/sps/site/homepage.jsp?code=MQX_HOME&tid=vanMQX
60	**Nano-RK**	http://www.nanork.org/projects/nanork/wiki
61	**Neutrino**	http://www.qnx.com/products/index.html
62	**Nucleus OS**	http://www.mentor.com/embedded-software/nucleus/
63	**Nut/OS**	http://www.ethernut.de/en/firmware/nutos.html
64	**NuttX**	http://www.on-time.com/rtos-32.htm
65	**OPENRTOS**	http://www.highintegritysystems.com/openrtos/
66	**OSA**	http://wiki.pic24.ru/doku.php/en/osa/ref/intro
67	**OSE**	http://www.enea.com/ose
68	**OS-9**	http://www.microware.com/
69	**OSEK**	http://www.osek-vdx.org/
70	**Partikle**	http://www.xtratum.org/
71	**picoOS**	http://picoos.sourceforge.net/
72	**Portos**	http://www.portos.org/
73	**PowerTV**	http://www.cisco.com/web/about/ac49/ac0/ac1/ac259/scientificatlanta.html
74	**Prex**	http://prex.sourceforge.net/
84	**Protothreads**	http://dunkels.com/adam/pt/index.html
85	**QNX**	http://www.qnx.com/
86	**Q-Kernel-Free**	http://www.quasarsoft.com/
87	**Q-Kernel-Pro**	http://www.quasarsoft.com/
88	**QP**	http://www.state-machine.com/qp/
89	**ReaGOS**	http://www.obp.fi/2009/products/reagos/index.php
90	**REAL/32**	
91	**Real-time Linux**	http://www.osadl.org/Realtime-Linux.projects-realtime-linux.0.html
92	**RIOT**	http://www.riot-os.org/
93	**RMX**	http://www.tenasys.com/rmx
94	**RTAI**	https://www.rtai.org/
95	**RTEMS**	http://www.rtems.com/
96	**rt-kernel**	http://www.rt-labs.com/rt-kernel_hw.shtml
97	**RT-Thread**	http://www.rt-thread.org/
98	**RTXC Quadros**	http://www.quadros.com/products/operating-systems
99	**SafeRTOS**	http://www.highintegritysystems.com/safertos/
100	**Salvo**	http://www.pumpkininc.com/
101	**SCIOPTA**	http://www.sciopta.com/
102	**scmRTOS**	http://scmrtos.sourceforge.net/ScmRTOS
103	**SDPOS**	http://www.sdpos.org/
104	**SHaRK**	http://shark.sssup.it/kernel.shtml
105	**silRTOS**	http://spanidea.com/products.php
106	**Sirius RTOS**	http://www.spaceshadow.com/products.php
107	**SMX RTOS**	http://www.smxrtos.com/index.html
108	**SOOS Project**	http://www.ingelec.uns.edu.ar/rts/soos/
109	**Symbian OS**	http://licensing.symbian.org/
110	**SYS/BIOS**	http://www.ti.com/tool/sysbios
111	**Talon DSP RTOS**	http://www.blackhawk-dsp.com/products/
112	**TargetOS**	http://www.blunkmicro.com/os.htm

113	**T-Kernel**	http://www.t-engine.org/
114	**ThreadX**	http://www.rtos.com/page/product.php?id=2
115	**THEOS**	http://www.theos-software.com/
116	**Trampoline Operating System**	http://trampoline.rts-software.org/
117	**TNKernel**	http://www.tnkernel.com/
118	**Transaction Processing Facility**	http://www-03.ibm.com/software/products/en/ztransaction-processing-facility
119	**TUD:OS**	http://demo.tudos.org/eng_about.html
120	**Unison RTOS**	http://www.rowebots.com/products/unison_rtos
121	**uSmartx**	http://usmartx.sourceforge.net/
122	**µTasker**	http://www.utasker.com/
123	**u-velOSity**	http://www.ghs.com/products/micro_velosity.html
124	**velOSity**	http://www.ghs.com/products/velosity.html
125	**VxWorks**	http://www.windriver.com/products/vxworks/
126	**Windows CE**	http://www.microsoft.com/windowsembedded/en-us/windows-embedded.aspx
127	**XRTOS**	http://www.esystech.com.br/english/produtos/xkernel/XKernel.php
128	**xPC Target**	http://www.mathworks.nl/products/simulink-real-time/index.html
129	**MontaVista Linux**	http://www.mvista.com/
130	**uOS**	https://code.google.com/p/uos-embedded/wiki/about
131	**PrKernel**	http://www.esol.com/
132	**uClinux/Petalinux 2.6**	http://www.xilinx.com/tools/petalinux-sdk.htm
133	**NORTi/ulTRON**	http://www.mispo.co.jp/

Bijlage V. Yourdan-diagrammen voor de demoapplicatie 1.0

System: Calculator - Context:

The purpose of *Calculator* system is: give the answer for the calculation expression.



	Calculator							
	performs the operations between two numbers (max 32 bits number) and gives the answer in hexadecimal or decimal radix (depends on the user's query)							
action	'*' - multiplying		'/' - division		'+' addition		'- 'subtraction	
Numeral system	Hexadecimal –H/h	Decimal – D/d	Hexadecimal H/h	Decimal D/d	Hexadecimal H/h	Decimal D/d	Hexadecimal H/h	Decimal D/d

The systems uses next types of message:

1. ***string_to_calculate*** – the input of user's calculation expression through the terminal;
2. ***query_to_calculate* –** the calculation expression used by the system (added the packet_header);
3. ***packet_header* -** framework necessary for the identification of the message
4. ***sruct_answer* –** the answer used by the system (added the *packet_header*);
5. ***string_answer*** – the output answer through the terminal;

Name of message		***string_to_calculate***					
Type		string					
Example		D356*25 c					
Type of data in the string	Type of numbers	Number one	operand	Number two	Space	Type of answer	Enter
Max size of string	8 bits	11 bytes	8 bits	11 bytes	8 bits	8 bits	8 bits

6. ***state_of_data*** – stores the ID and the time of received;

Totaal	216 bits						
Possible input	(H) | (h) | (D) | (d)	Max number: **2147483647** in dec ***7FFFFFFF***in hex Min number: - **2147483648** in dec - ***80000000*** in hex	(*) | (/) | (+) | (-)	Max number: **2147483647** in dec **7FFFFFFF** in hex Min number: - **2147483648** in dec - **80000000** in hex	"Space" ASCII: 20	(H) | (h) | (D) | (d) |(C) | (c)	"Enter" ASCII: 13

Name of message	***query_to_calculate***		
Type	structure		
Example	{{1,'*','h',32}, 126, 673}		
Type of data in the *query_to_calculate*	header	First number -integer	Second number
Name in the structure	header	opr_1	opr_2
Max size of *query_to_calculate*	48 bits	32bits	32 bits

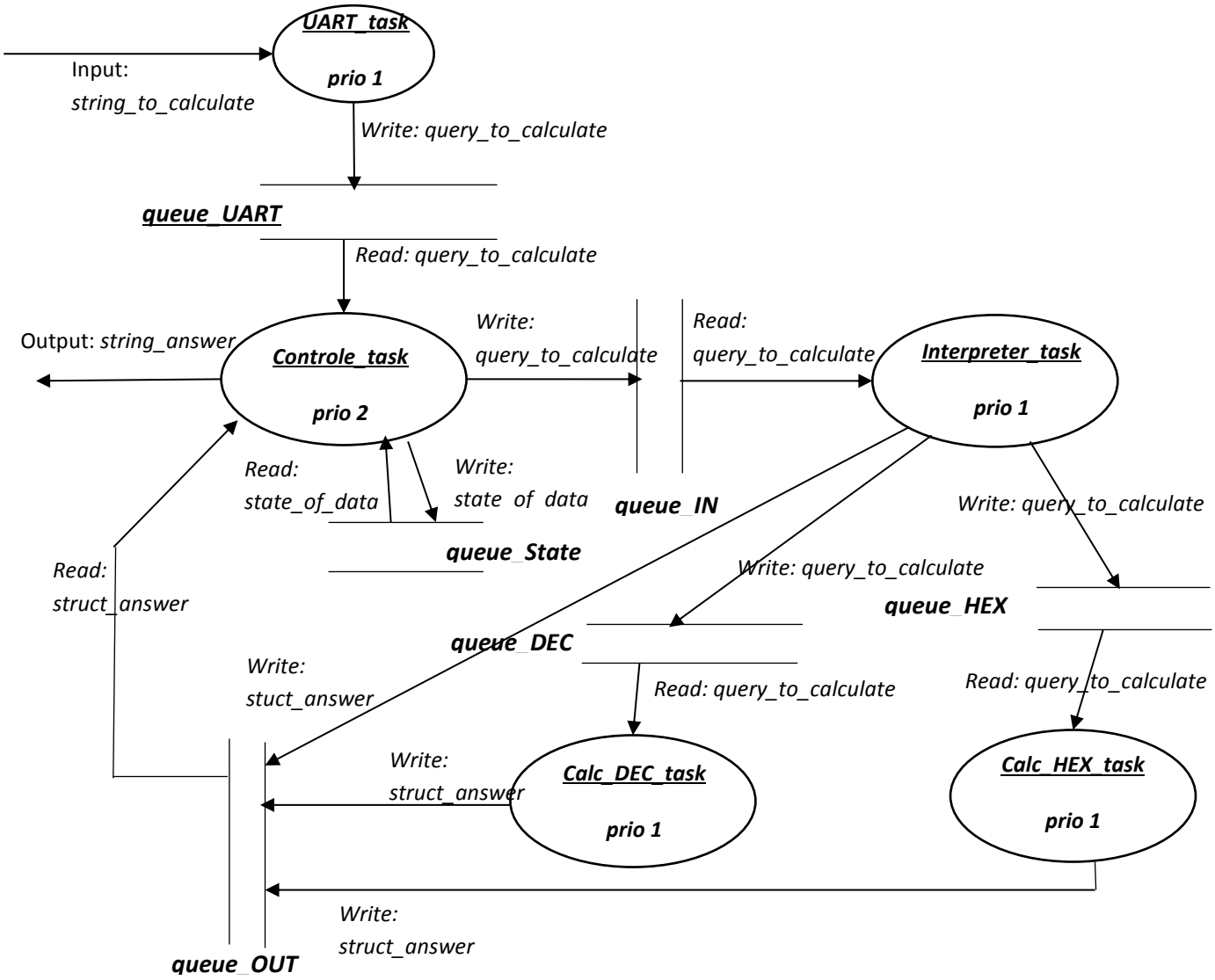
Name of message		***packet_header***			
type		structure			
Example		{1,'*','d','h',16}			
Type of data in the *packet_header*	uint16	char	char	char	uint8 /unsigned char
Name in the structure	ID	com	*input_radix*	*output_radix*	*data_length*
significance	ID	command	Radix of input numbers	Radix of output answer	length of the data
Max size of *packet_header*	16 bits	8 bits	8 bits	8 bits	8 bits

Name of message	***sruct_answer***		
Type	structure		
Example	{{1,'*','h',32}, 84798}		
Type of data in the sruct_answer	*packet_header*	string	
Name in the structure	*header*	*string_answer*	
Max size of the sruct_answer	48 bits	60 bits	
Type of answer		Calculated number	Error message
		Max number: 9223372036854775807 in dec 7FFFFFFFFFFFFFF in hex Min number: -9223372036854775808in dec - 8000000000000000 in hex	"The numeral system is not correct" "The operand is not correct" "The number is too large" "Must be enter two numbers" "The input is too large" Max length: is 27 bits

Name of message	***state_of_data***	
Type	structure	
Example	{2, 4567}	
Type of data in the *state_of_data*	uint16	uint32
Name in the structure	ID	start_time

System: Calculator 1.0 - Layer 1: Data flow diagram

System: Calculator 1.0 - Layer 2:



1.1 UART_task:

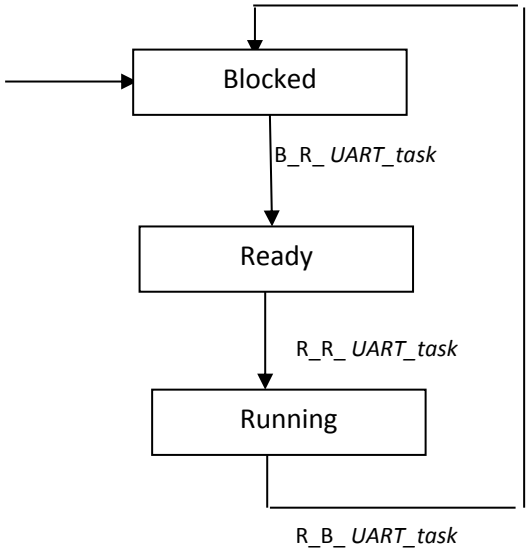
The purpose of the ***UART_task***:

- listen the UART-port if came a message
- convert string to the *query_to_calculate*
- transfer it to the system.

Processes of *UART_task*	Explanations

-

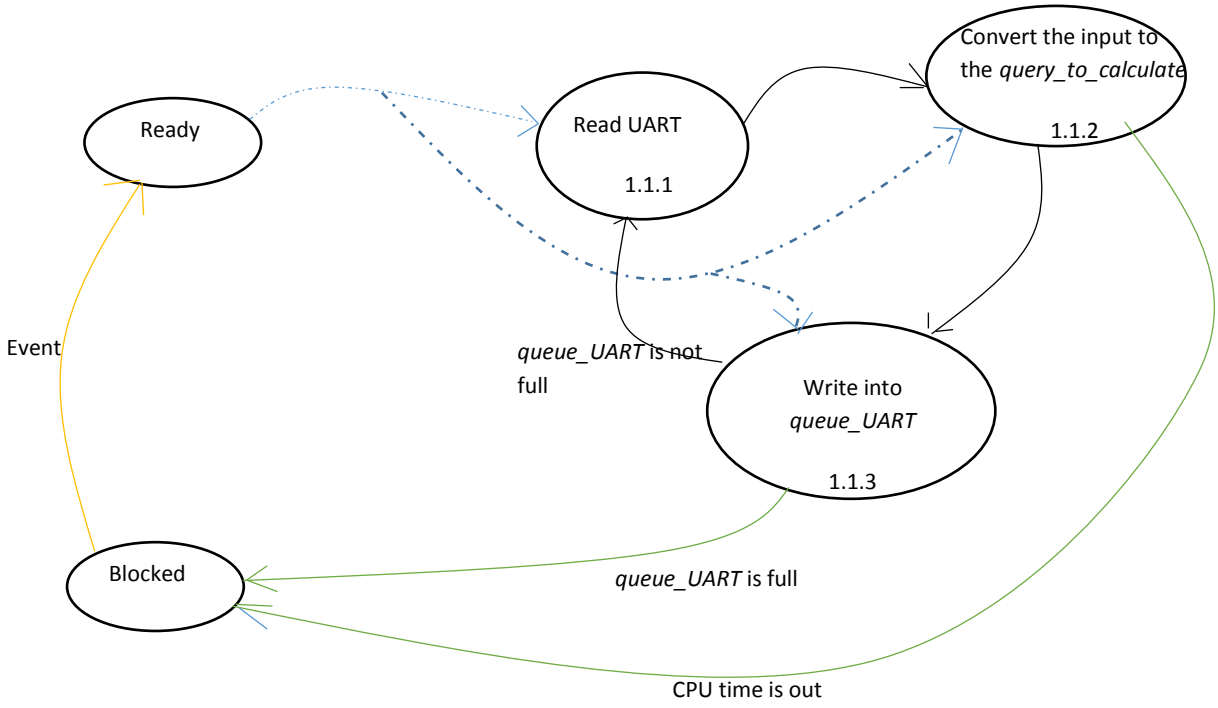
State transition diagram and action table of the *Uart_task*:



	Current State	Event	Actie	Condition	Next state	Transition
1	Blocked	-CPU gives the time; -the another task frees the place in *queue_UART*;		-20msec have passed - *UART_task* was waiting *queue_UART*;	Ready	B_R_ *UART_task*
2	Ready	Came the turn	- continues last stopped operation; -write to *queue_UART*;	-*USART_task* was unblocking the CPU time; -*UART_task* was unblocking *queue_UART*;	Run	R_R_*UART_task*
3	Run	- CPU time is out -*queue_UART* is full	-- stop operation; -- the *UART_task* wait the *queue_UART*;	 - *UART_task* blocked for a set time = 20msec;	Blocked	R_B_*UART_task*

Read UART-port	Every 20 miliseconds the system tries to read UART port
Convert the string to the structure: *query_to_calculate*	- Create the *header_packet;* - Create the *query_to_calculate*
Write the *query_to_calculate* to *queue_UART*	
Processes of *UART_task*	**Explanations**

Running state of: *UART_task*

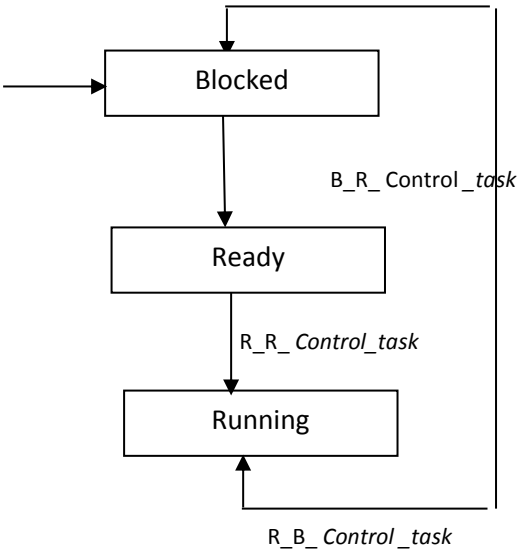


1.2 Control_task

The purpose of the *Control_task:*

- read *queue_UART*;
- save the time of received message;
- save the state of message in *queue_State*;
- send the *query_to_calculate* to the *queue_IN;*
- read *struct_answer* from *queue_OUT*;
- control of the ID's of resulted messages;
- time-out control – if no answer after 500msec => time is out => print error massage

State transition diagram and action table of the *Control_task:*

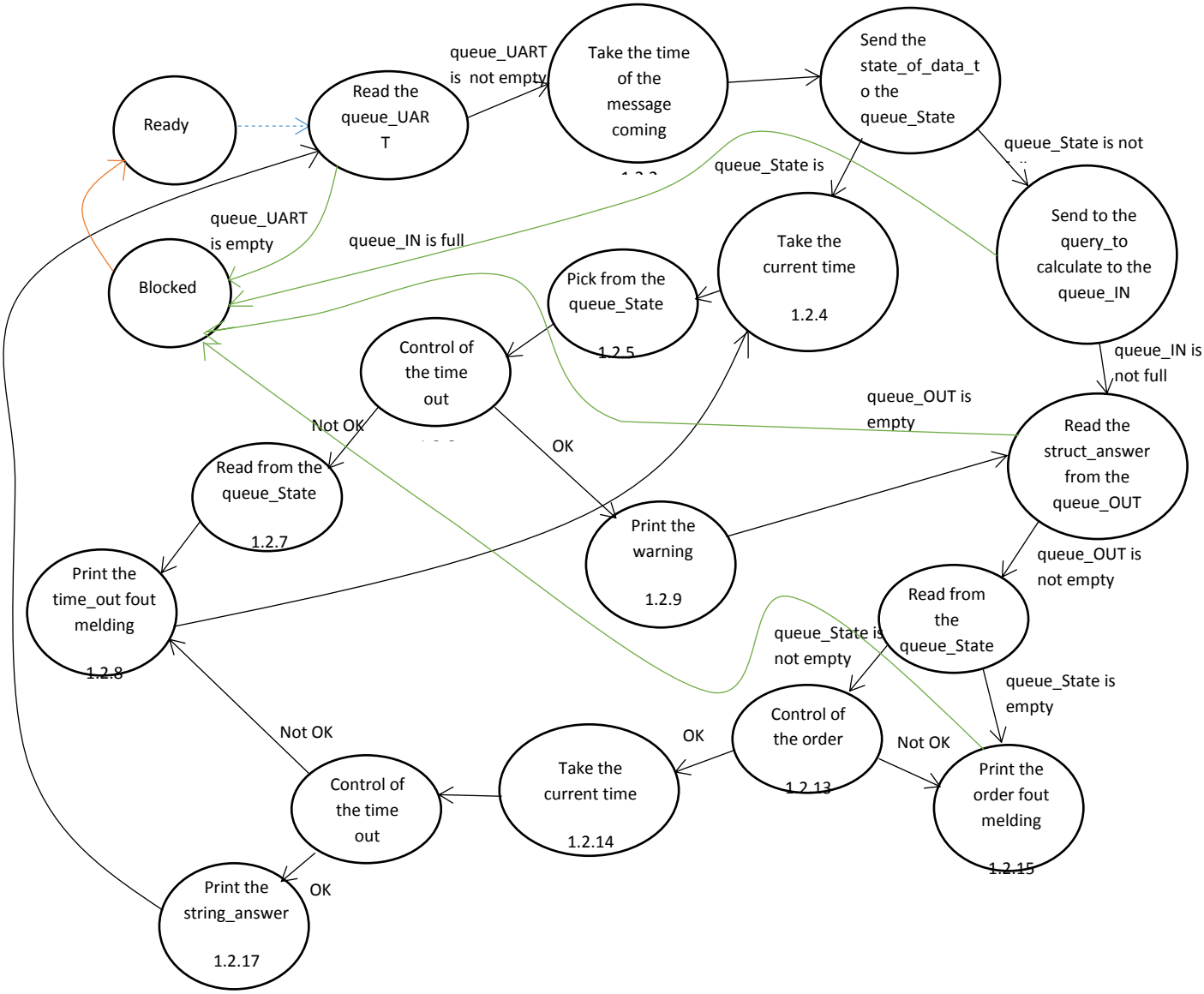


	Current State	Event	Action	Condition	Next state	Transition
1	Blocked	-CPU gives the time; - another task writes to *queue_UART*; - another task frees the place in *queue_IN*; - another task writes to *queue_OUT;*		- *Control_task* was waiting *queue_UART;* - *Control_task* was waiting *queue_IN*; - *Control_task* was waiting *queue_OUT;*	Ready	B_R_control_task
2	Ready	Came the turn	- continues last stopped operation; -read from *queue_UART*; -write to *queue_IN*; - read from *queue_OUT;*	- *Control_task* was unblocking the CPU time; - *Control_task* was unblocking *queue_UART;* - *Control_task* was unblocking *queue_IN;* - *Control_task* was unblocking *queue_OUT*;	Run	R_R_control_task
3	Run	- CPU time is out -order of the answers is not correct - *queue_UART* is empty - *queue_IN* is full - *queue_OUT* is empty	- stop operation; - *Control_task* wait *queue_UART*;	- print the error message	Blocked	R_B_control_task

Processes of ***Control_task***	Explanations
Read *queue_UART*	
Take the time when the *query_to_calculate* is received	
Save the status of the messages	In *queue_State* are saved the ID and time of coming of the message
Write *query_to_calculate* to *queue_IN*	
Read *queue_OUT*	
Control of the order of answer	If ID of the answer does not match of the order => print error message (is compared with a stored state)
Control of the time out	If for 0.5sec no result was received for the *query_to calculate* => print error message
Print *string_answer*	Call of the function *UART_print_string(char*)*

```
UART_print_string(char* str){
        Take_mutex();
        Printf("%s", str);
        Give_mutex();
}
```

Running state of the *Controle_task*



	Current State	Event	Actie	Condition	Next state	Transition

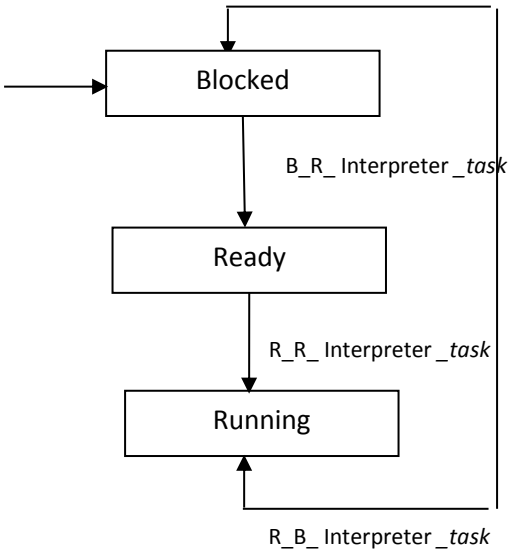
1.3 Interpreter_task

The purpose of the **Interpreter_task**:

Processes of *Interpreter_task*	Explanations
Read *queue_IN*	
Check the *query_to_calculate*	- *query_to_calculate.header.ID* must be int16_t; - *query_to_calculate.header.com* must be: '+' or '-' or '*' or '/'; - *query_to_calculate.header.input_radix* must be: 'H'/'h' or 'D'/'d'; - *query_to_calculate.header.output_radix* must be: 'H'/'h' or 'D'/'d' or 'C'/'c'; - *query_to_calculate.header.length* must be max 64 - *query_to_calculate.opr_1* must be max int32_t; - *query_to_calculate.opr_2* must be max int32_t;
Write to the *queue_DEC*	If the *query_to_calculate.header.output_radix* is 'D'/'d' or 'C'/'c'
Write to the *queue_HEX*	If the If the *query_to_calculate.header.output_radix* is 'H'/'h' or 'C'/'c'
Write to the *queue_OUT*	If are errors in the *query_to_calculate*

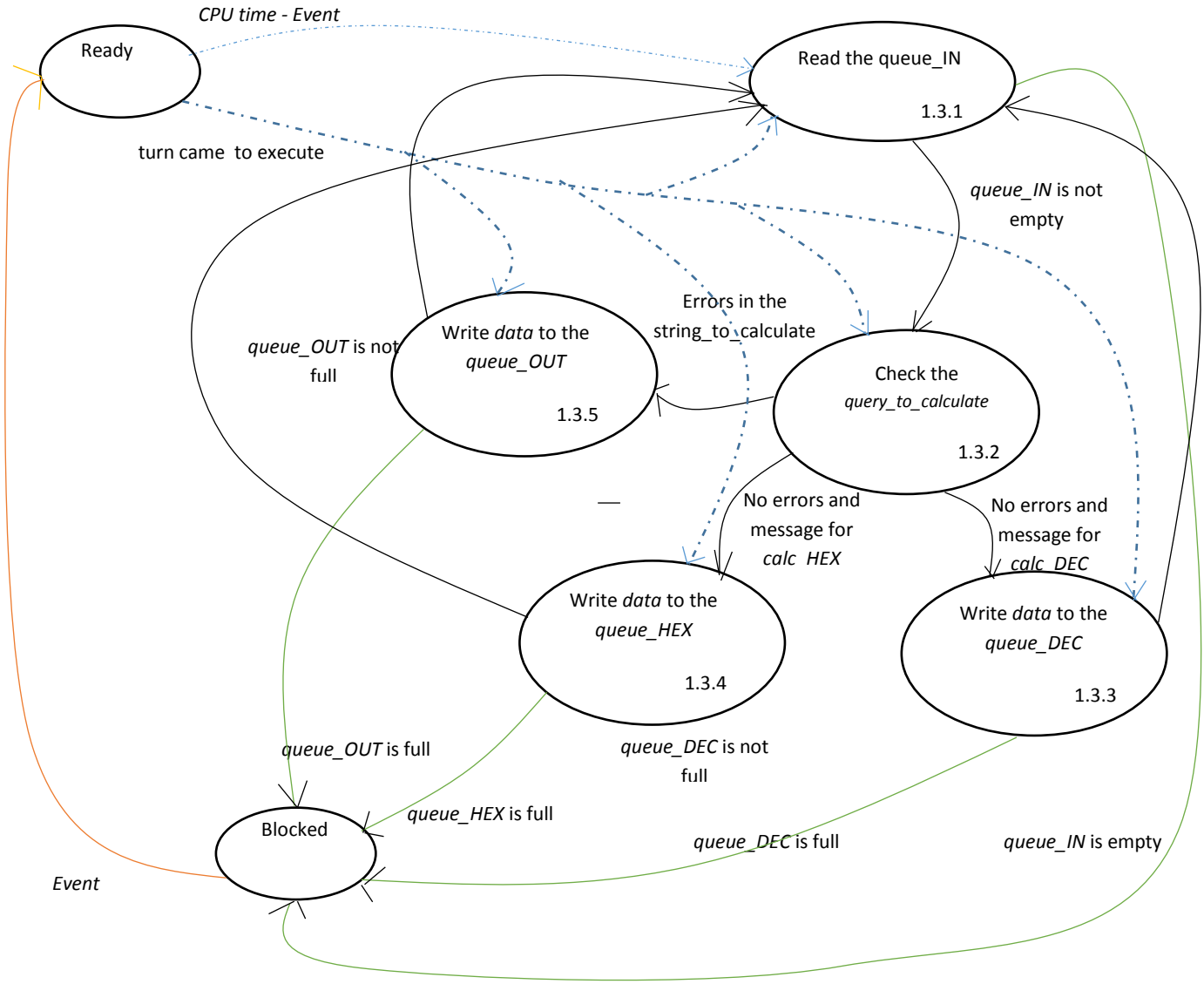
- read from the queue_IN;
- control query_to_calculate fields: if error =>send the struct_answer with the error message to the queue_OUT;
- send the query_to_calculate to queue_DEC or/and queue_HEX (depends on the user's request)

State transition diagram and action table of the *Interpreter_task*:



1	Blocked	---CPU gives the time; ---The another task writes to queue_IN; --the another task frees the place in the queue_DEC; --the another task frees the place in the queue_HEX; --the another task frees the place in the queue_OUT;		--the *Interpreter_task* was waiting the queue_IN; -- the *Interpreter_task* was waiting the queue_DEC; -- the *Interpreter_task* was waiting the queue_HEX; -- the *Interpreter_task* was waiting the queue_OUT;	Ready	B_R_Interpreter_task
2	Ready	Came the turn	-- continues last stopped operation; --read from the queue_IN; --write to the queue_OUT; --write to the queue_DEC; --write to the queue_HEX;	-- the *Interpreter_task* was unblocking the CPU time; -- the *Interpreter_task* was unblocking the queue_IN; -- the *Interpreter_task* was unblocking the queue_OUT; -- the *Interpreter_task* was unblocking the queue_DEC; -- the *Interpreter_task* was unblocking the queue_HEX;	Run	R_R_Interpreter_task
3	Run	-- CPU time is out --queue_IN is empty --queue_OUT is full --queue_DEC is full --queue_HEX is full	-- stop operation; -- the *Interpreter_task* wait the queue_IN; -- the *Interpreter_task* wait the queue_OUT; -- the *Interpreter_task* wait the queue_DEC; -- the *Interpreter_task* wait the queue_HEX;	--the *Interpreter_task* blocked for a set time = 10msec; --the *Interpreter_task* blocked for a set time = 10msec; --the *Interpreter_task* blocked for a set time = 10msec; --the *Interpreter_task* blocked for a set time = 10msec;	Blocked	R_B_Interpreter_task

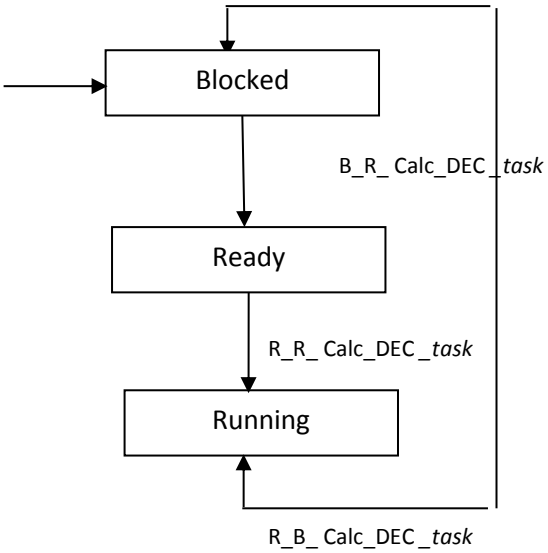
Running state of the Interpreter_task



1.4 Calc_DEC_task

The purpose of the **Calc_DEC_task**: calculation of the user's query and convert the answer to the decimal numeral system.

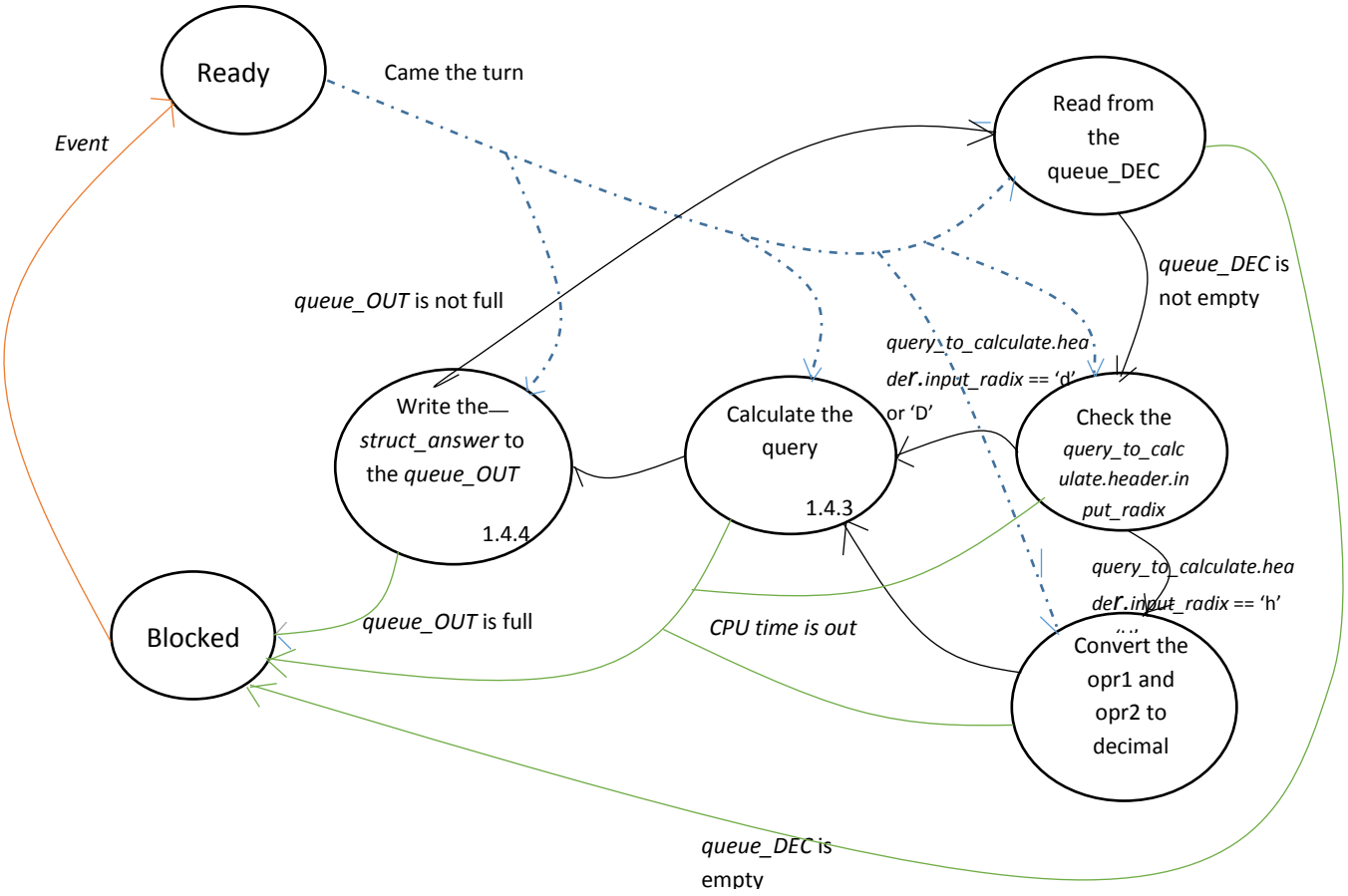
State transition diagram and action table of the *Calc_DEC_task*:



	Current State	Event	Actie	Condition	Next state	Transition
1	Blocked	---CPU gives the time; ---The another task writes to queue_DEC; --the another task frees the place in the queue_OUT;		--the Calc_DEC_task was waiting the queue_DEC; -- the Calc_DEC_task was waiting the queue_OUT;	Ready	B_R_Calc_DEC_ task
2	Ready	Came the turn	-- continues last stopped operation; --read from the queue_DEC; --write to the queue_OUT;	-- the Calc_DEC_task was unblocking the CPU time; -- the Calc_DEC_task was unblocking the queue_DEC; -- the Calc_DEC_task was unblocking the queue_OUT;	Run	R_R_Calc_DEC_ task
3	Run	-- CPU time is out --queue_DEC is empty --queue_OUT is full	-- stop operation; -- the Calc_DEC_task wait the queue_DEC; -- the Calc_DEC_task wait the queue_OUT;	--the Calc_DEC blocked for a set time = 10msec; --the Calc_DEC blocked for a set time = 10msec;	Blocked	R_B_Calc_DEC_ task

Processes of Calc_DEC_task	Explanations
Read *queue_DEC*	
Check the *query_to_calculate.header.input_radix*	if *query_to_calculate.header.input_radix* == 'h' or 'H' => convert to decimal
Calculate the query	
Write to the *queue_OUT*	**sruct_answer**

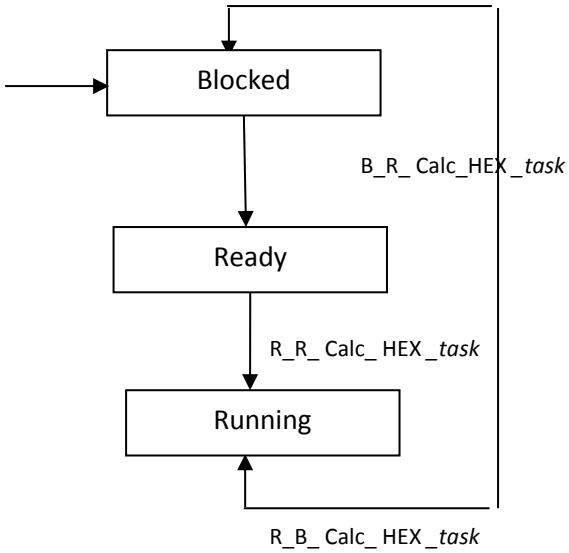
Running state of the Calc_DEC_task



1.5 Calc_HEX_task

The purpose of the **Calc_HEX_task**: calculation of the user's query and convert the answer to the hexadécimal numeral system.

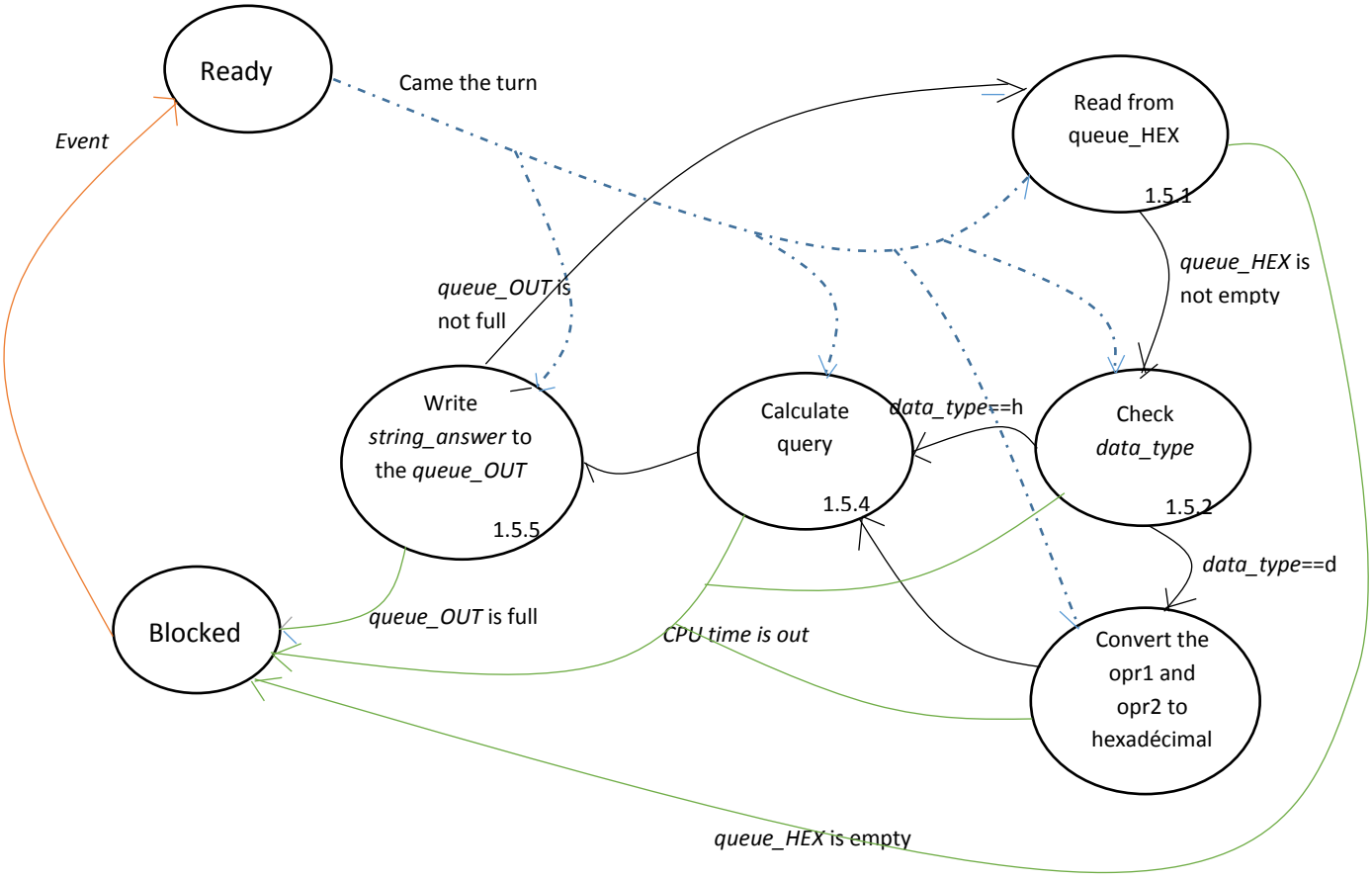
State transition diagram and action table of the *Calc_HEX_task*:



	Current State	Event	Actie	Condition	Next state	Transition
1	Blocked	---CPU gives the time; ---The another task writes to queue_HEX; --the another task frees the place in queue_OUT;		 -- Calc_HEX_task was waiting queue_HEX; -- Calc_HEX_task was waiting queue_OUT;	Ready	B_R_Calc_HEX_task
2	Ready	Came the turn	-- continues last stopped operation; --read from queue_HEX; --write to queue_OUT;	-- Calc_HEX_task was unblocking CPU time; -- Calc_HEX_task was unblocking queue_HEX; -- Calc_HEX_task was unblocking queue_OUT;	Run	R_R_Calc_HEX_task
3	Run	-- ended CPU time --queue_HEX is empty --queue_OUT is full	-- stop operation; -- Calc_HEX_task wait queue_HEX; -- Calc_HEX_task wait queue_OUT;	 -- Calc_HEX blocked for a set time = 10msec; -- Calc_HEX blocked for a set time = 10msec;	Blocked	R_B_Calc_HEX_task

Processes of Calc_DEC_task	Explanations
Read *queue_HEX*	
Check *query_to_calculate.header.input_radix*	if *query_to_calculate.header.input_radix* == 'H'/'h' => convert to décimal
Calculate the query	
Write to *queue_OUT*	*struct_answer*

Running state of the Calc_HEX_task



Queues description

Name of the queue	Type of element	Naam of element	Size of one element	Size of queue
Queue_UART	*structure*	*query_to_calculate*	216 bits	5 elements
Queue_State_data	*structure*	*state_of_data*	16 bits + sizeof(portTickType)	25 elements
Queue_IN	*structure*	*query_to_calculate*	112 bits	10 elements
Queue_DEC	*structure*	*query_to_calculate*	112 bits	10 elements
Queue_HEX	*structure*	*query_to_calculate*	112 bits	10 elements
Queue_OUT	*structure*	*struct_answer*	108bits	10 lements

Bijlage VI. Integratie van FreeRTOS 8.0 naar Xilinx SDK 14

FreeRTOS 8.00/ Xilinx SDK 14.7 Integration

1. Download FreeRTOS8.0.0 from http://sourceforge.net/projects/freertos/files/latest/download?source=files

2. Unzip the source code into a suitable directory, for example - .C:/project/workspace

3. Execute the `CreateProjectDirectoryStructure.bat` batch file that is located in the `../workspace/FreeRTOSV8.0.0/FreeRTOS/Demo/MicroBlaze_Spartan-6_EthernetLite/SDKProjects/RTOSDemoSource` directory.

4. Change the version *standalone_v3_01_a* at *standalone_v3_11_a* in the files:

4.1 Change in "`freertosv210.mld`" file that is located in the

`../workspace/FreeRTOSV8.0.0/FreeRTOS/Demo/MicroBlaze_Spartan-6_EthernetLite/KernelAwareBSPRepository/bsp/freertos_v2_00_a/data`:

- *"OPTION DEPENDS = (standalone_v3_11_a);"*

4.2 Change " in "freertosv210.tcl" file that is located in the

`../workspace/FreeRTOSV8.0.0/FreeRTOS/Demo/MicroBlaze_Spartan-6_EthernetLite/KernelAwareBSPRepository/bsp/freertos_v2_00_a/data`:

- *"set standalone_version standalone_v3_11_a "*
- *"set makeconfig [open "../standalone_v3_11_a/src/config.make" w]"*
- *"set filename [file join "../standalone_v3_11_a/src" "microblaze_interrupts_g.c"]"*

5. Make the link from the `KernelAwareBSPRepository` to the `Xilinx workspace:`

   - From within the SDK, click *Xilinx Tools > Repositories* to open the *Repositories* tab in the Preferences dialog box

- To add a local or global repository, click *New*. Browse to and select the `FreeRTOS\Demo\MicroBlaze_Spartan-6_EthernetLite\KernelAwareBSPRepository` directory
- Click *Rescan Repositiries*
- Click *Apply* and *Ok*

6. Create a FreeRTOS BSP from within the SDK, click *File > New > Xilinx Board Support Package* to open the New Board Support Package Project dialog. If a hardware project has not already been defined, you will be prompted to select one before the New Board Support Package Project dialog opens. Select "freertos" as the Board Support Package OS, before clicking *Finish*. The Board Support Package Settings dialog will open, in which the FreeRTOS parameters can be adjusted as required.

7. Once the FreeRTOS BSP has been created, it can be referenced by other SDK projects: *Properties > Project References > FreeRTOS BSP*.

Bijlage VII. De beschrijving van de queues in de demoapplicatie 1.0

Naam van queue	**Type van items**	***Lengte***	***Beschrijving***
xQueue_UART	*Structuur QUERY_TO_CALCULATE*	*1*	*Sender: UART_task, prio =1* *Receiver: Control_task, prio =2* In de huidige implementatie van de demoapplicatie is het genoeg om de lengte van deze queue gelijk aan 1 te maken, want de *Receiver* heeft een hogere prioriteit dan de *Sender*. Als een item binnen de queue komt, schakelt de scheduler meteen over naar de *Receiver*-taak.
xQueue_OUT	*Structuur STRUCT_ANSWER*	*1*	*Sender: Interpreter_task, prio =1;* *Calc_DEC_task, prio = 1;* *Calc_HEX_task, prio =1;* *Receiver: Control_task, prio =2;* De *Senders* hebben dezelfde prioriteit. Deze prioriteit is lager dan de prioriteit van *Receiver*. Als een item in de queue komt schakelt scheduler meteen over naar de *Receiver*, daarom is een lengte van 1 voor deze queue genoeg.
xQueue_STATE	*Structuur STATE_OF_DATA*	*25*	*Sender: Control_task, prio=2;* *Receiver: Control_task, prio=2;* Het doel van deze queue is de status van de verzonden verzoeken te bewaren. Het antwoord op het verzoek moet binnen 500msec komen. Als dat niet gebeurt geeft het systeem een foutmelding en verwijdert het de status van het verzoek vanuit de *queue_STATE.* Daarom moet er de *queue_STATE* genoeg plaats zijn om verzoeken die binnen 500msec komen te bewaren. De nieuwe verzoeken komen via de *UART_task*, die elke 20msec de UART-poort probeert uit te lezen. Het maximale aantal plaatsen in de *queue_STATE* moet dus gelijk zijn aan 500/20 = 25.
xQueue_IN	*Structuur QUERY_TO_CALCULATE*	*10*	*Sender: Control_task, prio =2;* *Receiver: Interpreter_task, prio = 1;* De *Sender* heeft een hogere prioriteit dan de *Receiver*. De *Sender* kan sneller items versturen dan de *Receiver* ze ontvangt. De queue moet groot genoeg zijn om meerdere items te bewaren.
xQueue_DEC	*Structuur QUERY_TO_CALCULATE*	*5*	*Sender: Interpreter_task, prio =1;* *Receiver: Calc_DEC_task, prio =1;* De *Sender* heeft dezelfde prioriteit als de *Receiver*. De *Sender* kan meer dan één item per slice-time versturen. De queue moet groot genoeg zijn om meerdere items te bewaren.
xQueue_HEX	*Structuur QUERY_TO_CALCULATE*	*5*	*Sender: Interpreter_task, prio =1;* *Receiver: Calc_HEX_task, prio =1;* De *Sender* heeft dezelfde prioriteit als de *Receiver*. De *Sender* kan meer dan één item per slice-time versturen. De queue moet groot genoeg zijn om meerdere items te bewaren.

Bijlage VIII. Het testplan voor de demoapplicatie 1.0

Tests van de demoapplicatie 1.0 als rekenmachine

- **Test 1.** De demoapplicatie moest de correcte berekeningen maken. Om dit te controleren bedacht ik testcases die alle mogelijke combinaties van numerieke systemen voor input en output van het verzoek bevatten. De testcases en te verwachten resultaten staan in Tabel 1.

Tabel 1: Testcases om correcte berekeningen te controleren

Testcase	***Opstelling***	***Verwacht resultaat***
1.1	*Input: d48+32 d*	*In decimaal: 80*
1.2	*Input: d4*11 d*	*In decimaal: 44*
1.3	*Input: d10000-20 D*	*In decimaal: 9980*
1.4	*Input: d48/4 d*	*In decimaal: 12*
1.5	*Input: d48+-32 h*	*In hexadecimaal: 10*
1.6	*Input: d2*100 h*	*In hexadecimaal: c8*
1.7	*Input: d20--35 h*	*In hexadecimaal: 37*
1.8	*Input: D48/-2 h*	*In hexadecimaal: -18*
1.9	*Input: h12+10 h*	*In hexadecimaal: 22*
1.10	*Input: h20*14 h*	*In hexadecimaal: 280*
1.11	*Input: hd*100 h*	*In hexadecimaal: d00*
1.12	*Input: h123/3 h*	*In hexadecimaal: 61*
1.13	*Input: d148+32 c*	*In decimaal: 180* *In hexadecimaal: b4*
1.14	*Input: d4*16 c*	*In decimaal: 64* *In hexadecimaal: 40*
1.15	*Input: d133--20 c*	*In decimaal: 153* *In hexadecimaal: 98*
1.16	*Input: D155/-5 c*	*In decimaal: 31* *In hexadecimaal: -1f*
1.17	*Input: H148+32 C*	*In decimaal: 348* *In hexadecimaal: 15c*
1.18	*Input: Hdd*0 c*	*In decimaal: 0* *In hexadecimaal: 0*
1.19	*Input: h133--55 C*	*In decimaal: 392* *In hexadecimaal: 188*
1.20	*Input: h1425/3 c*	*In decimaal: 1719* *In hexadecimaal: 6b7*
1.21	*Input: d-23+10 c*	*In decimaal: -13* *In hexadecimaal: -d*
1.22	*Input: h-2*10 h*	*In hexadecimaal: -20*

- **Test 2.** De demoapplicatie moest op incorrecte input een foutmelding geven. Om dit te controleren bedacht ik testcases die alle mogelijke combinaties van inputfouten bevatten. De testcases en te verwachten resultaten staan in Tabel 2.

Tabel 2: Testcases voor incorrecte input van het verzoek van de gebruiker

Testcase	***Opstelling***	***Verwacht resultaat***
2.1	*Input: d 48+32 d*	foutmelding over incorrecte input
2.2	*Input: d4832 h*	foutmelding over incorrecte input
2.4	*Input: d58-32h*	foutmelding over incorrecte input
2.5	*Input: d8*132h*	foutmelding over incorrecte input
2.6	*Input: z8*132 h*	foutmelding over incorrecte input
2.7	*Input: h8*132 k*	foutmelding over incorrecte input
2.8	*Input: d*132 h*	foutmelding over incorrecte input
2.9	*Input: h17/*132 h*	foutmelding over incorrecte input
2.10	*Input: h17/c*	foutmelding over incorrecte input
2.11	*Input: hd0* d*	foutmelding over incorrecte input
2.12	*Input: h111111111*	foutmelding over incorrecte input
2.13	*Drukt continu op de knop:* *Input: 33333333333333333333*	foutmelding na 15 tekens
2.14	*Input: hchhhh*	foutmelding over incorrecte input
2.15	*Input: d12&11 c*	foutmelding over incorrecte input

Tests van de demoapplicatie 1.0 om de eisen aan de RTOS-functionaliteit te controleren

- **Test 3.**
  Eis 1. In de demoapplicatie 1.0 moeten de taken geconfigureerd worden

In Calculator 1.0 creëerde ik vijf taken. Daarvoor gebruikte ik de functie *xTaskCreate(..)*. Deze functie heeft twee returnwaarden:

- ✓ *pdPASS*: de taak is met succes gecreëerd; Calculator 1.0 geeft een melding dat de taken succesvol gecreëerd zijn;
- ✓ *pdFALSE*: creëren van de taak is mislukt; Calculator 1.0 geeft een foutmelding.

De testcases staan in tabel 3.

Tabel 3: Testcases voor het controleren van de correcte configuratie van de taken

Testcase	***Opstelling***	***Verwacht resultaat***
3.1	Alle taken worden met correcte parameters gecreëerd.	Melding van het systeem: "*Main: The Tasks of the Calculator system are created successfully."*
3.2	De *UART_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The UART_ task has not created."*
3.3	De *Control_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Control_ task has not created."*
3.4	De *Interpreter_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Interpreter_ task has not created."*
3.5	De *Calc_DEC_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Calc_DEC_task has not created."*
3.6	De *Calc_HEX_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Calc_HEX_task has not created."*

- **Test 4.**

  Eis 2. De demoapplicatie 1.0 moet het Preemptive algoritme van het RTOS demonstreren

  Eis 3. De demoapplicatie 1.0 moet taken met verschillende prioriteiten bevatten

Het doel van deze tests is het Preemptive algoritme van FreeRTOS te demonstreren. In *Calculator 1.0* heeft de *Control_task* de hoogste prioriteit, hoger dan andere taken. De *Control_task* wordt gedeblokkeerd door drie typen van events:

- ✓ een bericht in de *queue_UART*;
- ✓ een bericht in de *queue_OUT*;
- ✓ een tijd-event: elke 500msec moet de *Control_task* gedeblokkeerd worden om de timeout te checken.

Na een van deze events moet bij het Preemptive algoritme de uitgevoerde taak onmiddellijk geblokkeerd worden en de *Control_task* wordt uitgevoerd. De testcases om het werk van het Preemptive algoritme te controleren staan in Tabel 4.

Tabel 4: Testcases: het testen van het Preemptive algoritme en verschillende prioriteiten

Testcase	***Opstelling***	***Verwacht resultaat***
4.1	De *UART_task* stuurt het bericht in de *queue_UART* en print daarna de melding: "*UART_task: sends the query_to_calculate to xQueue_UART*" Wanneer de *Control_taak* het bericht vanuit de *queue_UART* heeft gelezen, print de taak melding: "*Control_task: takes the query_to_calculate from xQueue_UART."*	Bij deze testcase is de volgorde van de meldingen in Calculator 1.0 belangrijk. Bij het Preemptive algoritme wanneer het bericht in de *queue_UART* komt, moet de *Control_task* onmiddellijk uitgevoerd worden en de uitvoering van *UART_task* wordt gestopt. Daarom de meldingen moeten in de volgende volgorde komen: Eerst: "*Control_task: takes the query_to_calculate from xQueue_UART."* Daarna: "*UART_task: sends the query_to_calculate to xQueue_UART.*"
4.2	Elk 500msec checkt de *Control_task* de timeout. Elke 500msec print de *Control_task* de melding: "*Current time*. *Control_task: check the time."*	De melding: "*Control_task: check the time"* wordt elke 500msec geprint. Alle andere taken worden geblokkeerd.

- **Test 5.**
  Eis 4. De demoapplicatie 1.0 moet periodieke taken bevatten

In de demoapplicatie 1.0 moesten twee taken periodiek worden uitgevoerd: *UART_task* en *Control_task*. *UART_task* moest elke 20msec na de laatste uitvoering geactiveerd worden. De *Control_task* moest elke 500msec de tijd checken, de laatste werd in de vorige testcase gecheckt. In Tabel 5 staan de testcases voor de *UART_task*.

Tabel 5: Testcase: testen van de periodiciteit van de uitvoering van UART_task

Testcase	***Opstelling***	***Verwacht resultaat***
5.1	Wanneer de *UART_task* loopt, print ze de melding: "*Current time.UART_task: is active*". Wanneer *UART_task* klaar is met de uitvoering, print ze de melding: "*Current time. UART_task: the end of the execute*".	Het verschil tussen de tijd bij de melding:"*Current time. UART_task: the end of the execute*" en de volgende melding: "*Current time. UART_task: is active*" is 20msec.

- **Test 6.**
  Eis 5. In de demoapplicatie 1.0 moeten de queues geconfigureerd worden

In de demoapplicatie 1.0 werden zes queues gecreëerd. Daarvoor werd de functie xQueueCreate*(..)* gebruikt. Deze functie heeft twee returnwaarden:

- ✓ een pointer naar de queue handle als de taak met succes is gecreëerd; *Calculator 1.0* geeft een melding dat de queues succesvol gecreëerd zijn;
- ✓ NULL pointer als het creëren van de taak is mislukt; ; *Calculator 1.0* geeft een foutmelding.

De gebruikscases staan in tabel 6.

Tabel 6: Testcases voor het controleren van de queue-configuratie

Testcase	***Opstelling***	***Verwacht resultaat***
6.1	Alle queues worden met correcte parameters gecreëerd.	Melding van het systeem: "*Main*: *The Queues of the Calculator system were created successfully."*
6.2	De *queue_UART* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_UART has not created"*
6.3	De *queue_STATE* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_STATE has not created"*
6.4	De *queue_IN* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_IN has not created"*
6.5	De *queue_OUT* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_OUT has not created"*
6.6	De *queue_HEX* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_HEX has not created"*
6.7	De *queue_DEC* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_DEC has not created"*

- **Test 7.**

  Eis 6. In de demoapplicatie 1.0 moeten functies geïmplementeerd worden zoals Receive from the Queue, Send to the Queue

Alle berichten worden via de queues uitgewisseld. Daarvoor zijn de FreeRTOS-functies *xQueueReceive(..)* en *xQueueSendToBack(..)* gebruikt. Deze functies hebben twee returnwaarden:

- ✓ *pdTRUE* als de operatie is gelukt; *Calculator 1.0* geeft een melding dat het bericht succesvol gestuurd/ontvangen is;
- ✓ *pdFALSE* als de operatie is mislukt; *Calculator 1.0* geeft een foutmelding.

De testcases voor het controleren van de uitvoering van deze functies staan in Tabel 7.

Tabel 7: Testcases voor het controleren van de uitvoering van de functies xQueueReceive(..) en xQueueSendToBack(..)

Testcase	***Opstelling***	***Verwacht resultaat***
7.1	De *Control_task* bevat functie *xQueueReceive(..)* die de berichten uit *queue_UART* leest. Voor deze test wordt deze functie geblokkeerd. Daarna worden twee verzoeken ingevoerd. De *UART_task* stuurt alle verzoeken naar de *queue_UART* met de functie *xQueueSendToBack(..).* De returnwaarde van deze functie wordt gecontroleerd.	Na het tweede verzoek raakt de *queue_UART* vol. *Calculator 1.0* geeft de foutmelding: "*UART_task: xQueue_UART is full*".
7.2	In de *Interpreter_task* wordt de functie *xQueueReceive(..)* aangeroepen. Deze functie probeert de *queue_IN* uit te lezen. Als de *queue_IN* leeg is, wordt de taak geblokkeerd. In deze testcases wordt de returnwaarde van de functie *xQueueReceive(..)* gecontroleerd. Daarvoor wordt de code van de *Interpreter_task* met regel die het melding over de toestand van de *queue_IN* print, toegepast.	Als de *queue_IN* leeg is, geeft *Calculator 1.0* de foutmelding "*Interpreter_task: xQueue_IN is empty*."

- **Test 8.**

Eis 7. In de demoapplicatie 1.0 moeten processen gesynchroniseerd worden met gebruikmaking van Queues.

De taken *Interpreter_task, Calc_DEC_task* en *Calc_HEX_task* zijn met de queues gesynchroniseerd. Ze worden geactiveerd als er een bericht komt in de queue dat ze moeten uitlezen. Dus ze nemen geen processortijd als er geen events zijn. Om deze synchronisatie te realiseren, wordt in de functie *xQueueReceive(..)* een oneindige wachttijd geconfigureerd. In deze test bekeek ik wat er gebeurde als deze synchronisatie werd doorbroken. De testcases voor deze tests staan in Tabel 8.

Tabel 8: Testcases: *Het controleren van* de s*ynchronisatie van processen die gebruik maken van queues*

Testcase	***Opstelling***	***Verwacht resultaat***
8.1	In de taken *Interpreter_task, Calc_DEC_task* en *Calc_HEX_task* wordt in de functies *xQueueReceive(..)* de wachttijd op 0 geconfigureerd. Aan het begin en aan het eind van de taak van de uitvoering print elke taak respectievelijk de meldingen: *"Name_task: is active."* en *"Name_task: has ended."* Vervolgens wordt *Calculator 1.0* systeem gestart zonder dat de gebruiker een verzoek invoert.	In het FreeRTOS is het Round-Robin scheduling algoritme geïmplementeerd. Voor elke taak met dezelfde prioriteit geeft het systeem een bepaalde tijd voor de uitvoering: time-slicing. Bij deze testcase wordt verwacht dat *Calculator* de volgende meldingen print. Binnen de eerste time-slicing (als de *Interpreter_task* als eerste is gecreëerd): "*Interpreter_task: is active."* *"Interpreter_task: has ended."* *"Interpreter_task: is active."* *"Interpreter_task: has ended."* … Binnen de tweede time-slicing (als de *Calc_DEC_task* als tweede is gecreëerd): "*Calc_DEC_task: is active."* *"Calc_DEC_task: has ended."* *"Calc_DEC_task: is active."* *"Calc_DEC_task: has ended."* …
8.2	In de taken *Interpreter_task, Calc_DEC_task* en *Calc_HEX_task* wordt in de functies *xQueueReceive(..)* de wachttijd op oneindig geconfigureerd. Aan het begin en aan het eind van de taakuitvoering print elke taak respectievelijk de meldingen: *"Name_task: is active."* en *"Name_task: has ended."* Vervolgens wordt *Calculator 1.0* gestart zonder dat de gebruiker een verzoek invoert.	Hier wordt verwacht dat de taken door de queue worden geblokkeerd tot het event optreedt. Bij deze testcase wordt verwacht dat *Calculator 1.0* één keer de volgende meldingen print: "*Interpreter_task*: is active." "*Calc_DEC_task*: is active." "*Calc_HEX_task*: is active."

- **Test 9.**
  Eis 8. In de demoapplicatie 1.0 moet een systeem timer worden gebruikt
  Eis 9. In de demoapplicatie 1.0 moet time-out van het systeem geconfigureerd worden

In de *Control_task* van de demoapplicatie 1.0 werd de timeout gecheckt. *Calculator 1.0* moest een foutmelding geven als het antwoord op het verzoek niet binnen een deadline van 500msec kwam. Daarvoor gebruikte de demoapplicatie de systeemtimer. Het doel van de test was te controleren of het systeem reageerde op de niet-gehaalde deadline. De testcases staan in Tabel 9.

Tabel 9: Testcases: *Testen dat het systeem reageert op de niet-gehaalde deadline*

Testcase	***Opstelling***	***Verwacht resultaat***
9.1	In de *Calc_DEC_task* wordt de functie *xQueueSendToBack(..)* geblokkeerd. Deze functie stuurt het antwoord naar de *queue_OUT*. Vervolgens wordt het verzoek ingevoegd: *d122+2 d*.	Na 500msec geeft *Calculator 1.0* de foutmelding "*Controle_task: Time is out"*.
9.2	Om één verzoek van de gebruiker te berekenen heeft *Calculator 1.0* 30msec nodig. De timeout van het systeem is 500msec. In deze testcase wordt de timeout van het systeem verminderd tot 10msec.	Het systeem haalt nooit de deadline, 10msec na invoering van het verzoek komt de foutmelding "*Controle_task: Time is out"*.

- **Test 10.**
  Eis 10. In de demoapplicatie 1.0 moet Mutex geconfigureerd worden

In de demoapplicatie 1.0 werd Mutex geconfigureerd met de FreeRTOS-functie: *xQueueCreateMutex(..).* Deze functie heeft twee returnwaarden:

- ✓ een pointer naar de Mutex handle als de taak met succes is gecreëerd; *Calculator 1.0* geeft een melding dat de Mutex succesvol gecreëerd is;
- ✓ NULL pointer als het creëren is mislukt; *Calculator 1.0* geeft een foutmelding.

De testcases staan in tabel 10.

Tabel 10:Testcases voor het controleren van de Mutex-configuratie

Testcase	***Opstelling***	***Verwacht resultaat***
10.1	De Mutex is gecreëerd.	Melding van het systeem: "*Main*: *The Mutex is created successfully."*
10.2	De Mutex wordt incorrect gecreëerd.	Foutmelding van het systeem: "*Main*: *The Mutex has not created."*

- **Test 11.**
  Eis 11. Beveiliging van een resource door middel van Mutex

In de demoapplicatie 1.0 moet elke taak meldingen printen. Daarvoor wordt de seriële poort gebruikt. De toegang tot de seriële poort wordt door de Mutex beveiligd. Het doel van deze testcases is om te bekijken hoe effectief Mutex de seriële poort beveiligt. De testcases zijn in tabel 11 beschreven.

Tabel 11: Testcases voor het controleren van de beveiliging van de seriële port door Mutex

Testcase	***Opstelling***	***Verwacht resultaat***
11.1	De toegang tot de seriële port is door Mutex beveiligd.	Elk geprinte melding is afgerond
11.2	De toegang tot de seriële port is niet door Mutex beveiligd.	*Calculato 1.0* print afgebroken meldingen.

- **Test 12. Stresstest**

Doel. Het doel van deze test is te controleren hoe *Calculator*-systeem zich gedraagt als de gebruiker de input heel snel invoert. De testcase zijn in tabel 12 beschreven.

Tabel 12: Testcase voor de stresstest

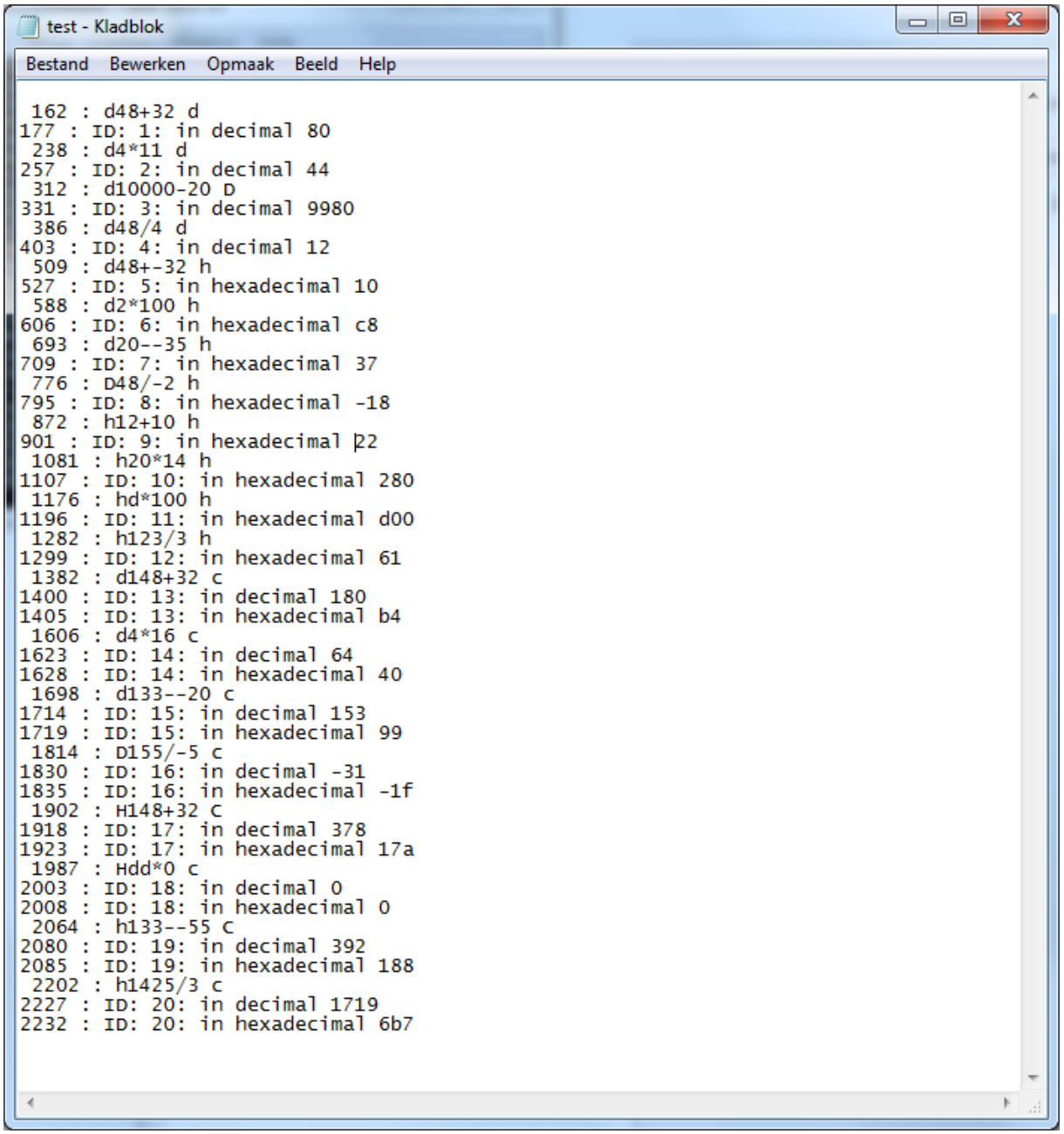
Testcase	***Opstelling***	***Verwacht resultaat***
12.1	Calculator 1.0 wordt gestart. Een tekstbestand met 100 verzoeken wordt naar het systeem gestuurd.	*Calculator* berekent alle verzoeken vanuit het bestand en geeft de antwoorden.

Bijlage IX. Resultaten van het testen van de demoapplicatie 1.0

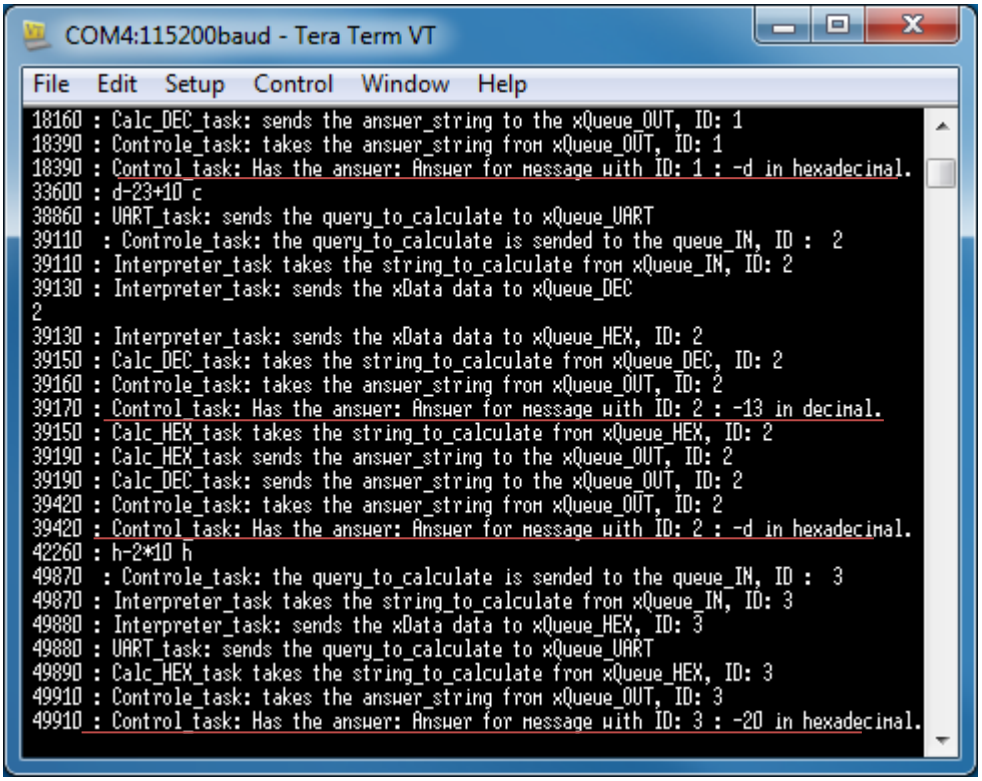
Test 1. Het testen van de correcte berekeningen.

***Test*case**	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
1.1	*Input: d48+32 d*	*In decimaal is 80*	*Correct /*zie printscreen 1
1.2	*Input: d4*11 d*	*In decimaal is 44*	*Correct /*zie printscreen 1
1.3	*Input: d10000-20 D*	*In decimaal is 9980*	*Correct /* zie printscreen 1
1.4	*Input: d48/4 d*	*In decimaal is 12*	*Correct /* zie printscreen 1
1.5	*Input: d48+-32 h*	*In hexadecimaal is 10*	*Correct /* zie printscreen 1
1.6	*Input: d2*100 h*	*In hexadecimaal is c8*	*Correct /* zie printscreen 1
1.7	*Input: d20--35 h*	*In hexadecimaal is 37*	*Correct /* zie printscreen 1
1.8	*Input: D48/-2 h*	*In hexadecimaal is -18*	*Correct /* zie printscreen 1
1.9	*Input: h12+10 h*	*In hexadecimaal is 22*	*Correct /* zie printscreen 1
1.10	*Input: h20*14 h*	*In hexadecimaal is 280*	*Correct /* zie printscreen 1
1.11	*Input: hd*100 h*	*In hexadecimaal is d00*	*Correct /* zie printscreen 1
1.12	*Input: h123/3 h*	*In hexadecimaal is 61*	*Correct /* zie printscreen 1
1.13	*Input: d148+32 c*	*In decimaal is 180* *In hexadecimaal is b4*	*Correct /* zie printscreen 1
1.14	*Input: d4*16 c*	*In decimaal is 64* *In hexadecimaal is 40*	*Correct /* zie printscreen 1
1.15	*Input: d133--20 c*	*In decimaal is 153* *In hexadecimaal is 99*	*Correct /* zie printscreen 1
1.16	*Input: D155/-5 c*	*In decimaal is -31* *In hexadecimaal is -1f*	*Correct /* zie printscreen 1
1.17	*Input: H148+32 C*	*In decimaal is 378* *In hexadecimaal is 17a*	*Correct /* zie printscreen 1
1.18	*Input: Hdd*0 c*	*In decimaal is 0* *In hexadecimaal is 0*	*Correct /* zie printscreen 1
1.19	*Input: h133--55 C*	*In decimaal is 392* *In hexadecimaal is 188*	*Correct /* zie printscreen 1
1.20	*Input: h1425/3 c*	*In decimaal is 1719* *In hexadecimaal is 6b7*	*Correct /* zie printscreen 1
1.21	*Input: d-23+10 c*	*In decimaal is -13* *In hexadecimaal is -d*	*Correct /* zie printscreen 1.1
1.22	*Input: h-2*10 h*	*In hexadecimaal is -20*	*Correct /* zie printscreen 1.1

Printscreen 1: De output van Test 1 – Testcases 1.1 t/m1.20



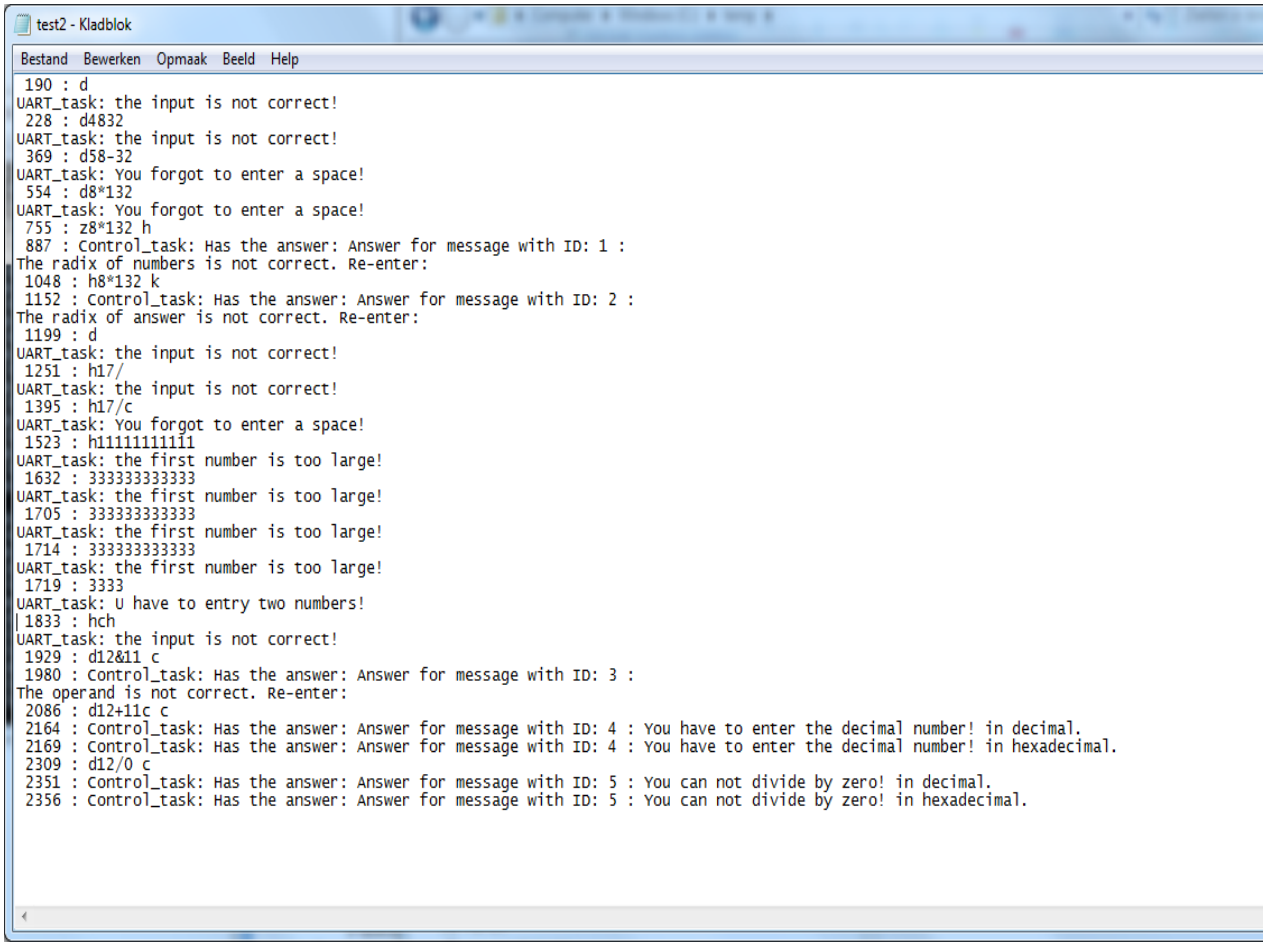
Printscreen 1.1: De output van Test 1 – Testcases 1.20 en 1.21



Test 2. Foutmeldingen van de incorrecte input van de gebruiker.

***Test*case**	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
2.1	*Input: d 48+32 d*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.2	*Input: d4832 h*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.4	*Input: d58-32h*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.5	*Input: d8*132h*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.6	*Input: z8*132 h*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.7	*Input: h8*132 k*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.8	*Input: d*132 h*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.9	*Input: h17/*132 h*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.10	*Input: h17/c*	foutmelding over incorrecte input	*Correct /zie* printscreen 2
2.11	*Input: hd0* d*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.12	*Input: h111111111*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.13	*Drukt continu op de knop:* *Input:33333333333333333333*	Na 15 symbolen het systeem moet foutmelding geven	*Correct* / zie printscreen 2
2.14	*Input: hchhhh*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.15	*Input:d12&11 c*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.16	*Input:d12+11c c*	foutmelding over incorrecte input	*Correct* / zie printscreen 2
2.17	*Input:d12/0 c*	foutmelding over delen op zero	*Correct* / zie printscreen 2

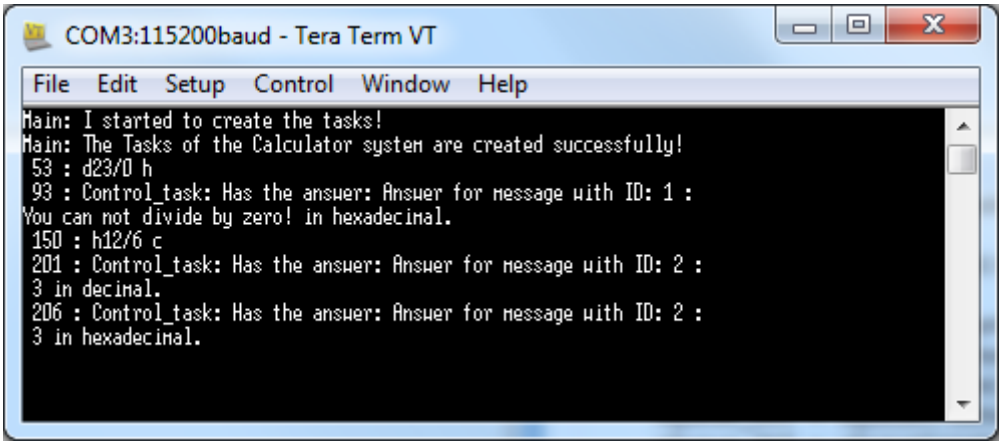
Printscreen 2: De output van de test 2 – Testcases 2.1 t/m 2.17.



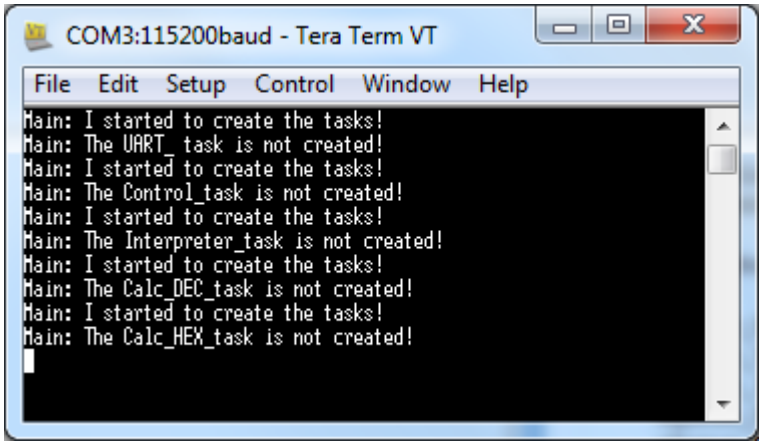
Test 3. Het controleren van de correcte configuratie van de taken.

Testcase	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
3.1	Alle taken worden met correcte paarmeters gecreëerd.	Melding van het systeem: "*Main: The Tasks of the Calculator system are created successfully."*	*Correct /*zie printscreen 3
3.2	De *UART_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The UART_ task is not created."*	*Correct /* zie printscreen 4
3.3	De *Control_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Control_ task is not created."*	*Correct /* zie printscreen 4
3.4	De *Interpreter_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Interpreter_ task is not created."*	*Correct /* zie printscreen 4
3.5	De *Calc_DEC_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Calc_DEC_task is not created."*	*Correct /* zie printscreen 4
3.6	De *Calc_HEX_ task* wordt incorrect gecreëerd.	Foutmelding: "*Main: The Calc_HEX_task is not created."*	*Correct /* zie printscreen 4

Printscreen 3: De output van de test 3 - Tastcase 3.1



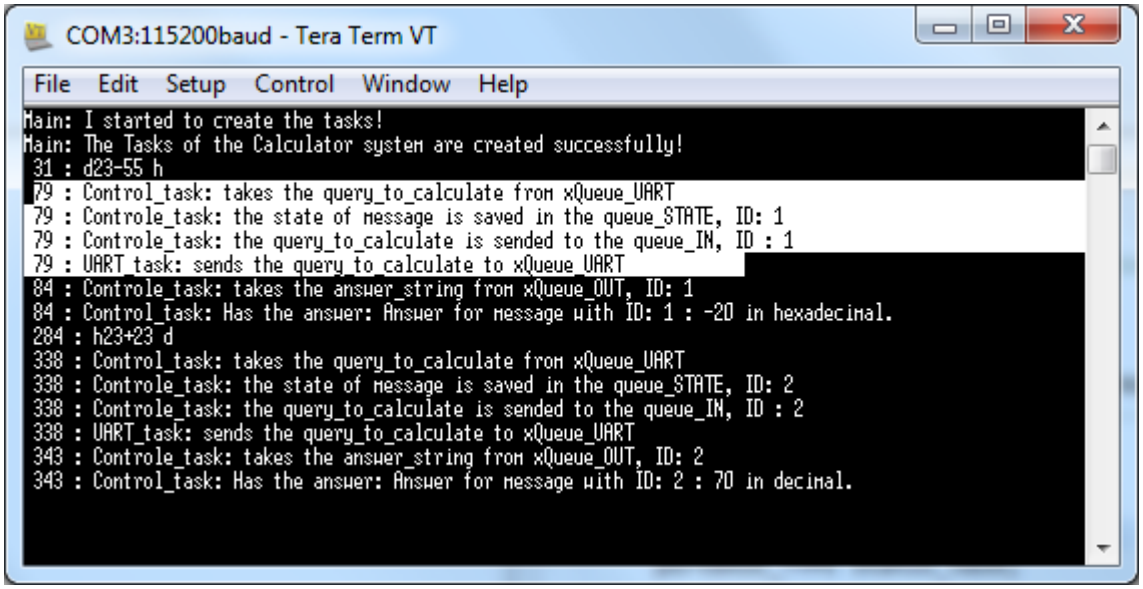
Printscreen 4: De output van de test 3 - Testcases 3.2 t/m 3.6

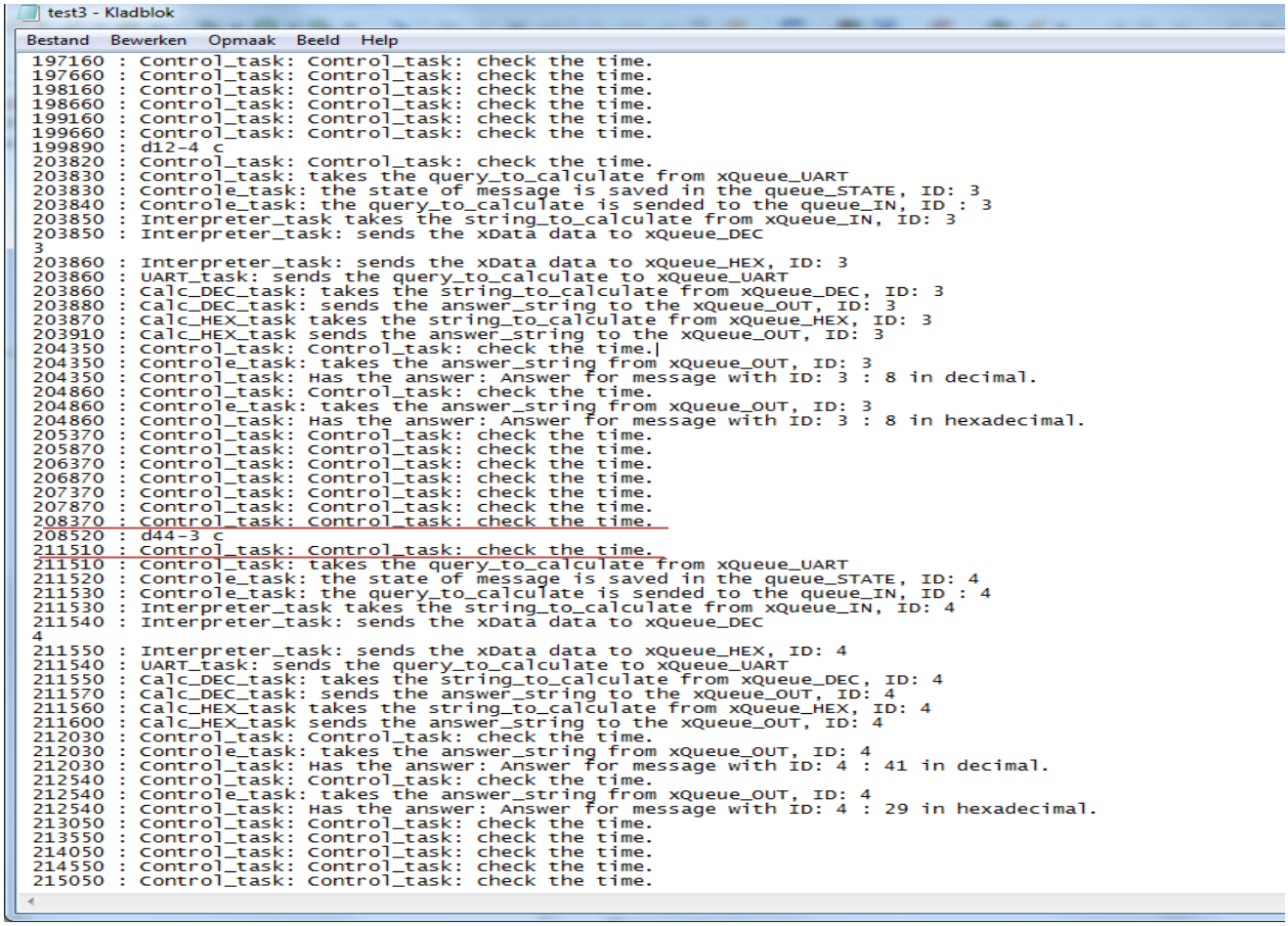


Test 4. Het testen het testen van het Preemptive algoritme en verschillende prioriteiten

Testcase	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
4.1	De *UART_taak* stuurt het bericht in de *queue_UART* en daarna print de melding: "*UART_task: sends the query_to_calculate to xQueue_UART*" Wanneer de *Control_taak* het bericht vanuit de *queue_UART* is gelezen, dan print de taak melding: "*Control_task: takes the query_to_calculate from xQueue_UART.*"	Bij deze test case is belangrijk de volgorde van de meldingen van het *Calculator* systeem. Eerst: "*Control_task: takes the query_to_calculate from xQueue_UART.*" Daarna: "*UART_task: sends the query_to_calculate to xQueue_UART*"	*Correct /*Zie Printscreen 5
4.2	Elk 0.5sec checkt de *Control_task* timeout. Elk 0.5sec print de *Control_task* de melding: "*Huidige time*. *Control_task: check the time.*"	De melding: "*Control_task: check the time.*" Moet elke 0.5 sec wordt geprint. Alle andere taken moeten worden afgebroken.	*Niet Correct* Zie Printscreen 6

Printscreen 5: De output van de test 4 - Testcase 4.1



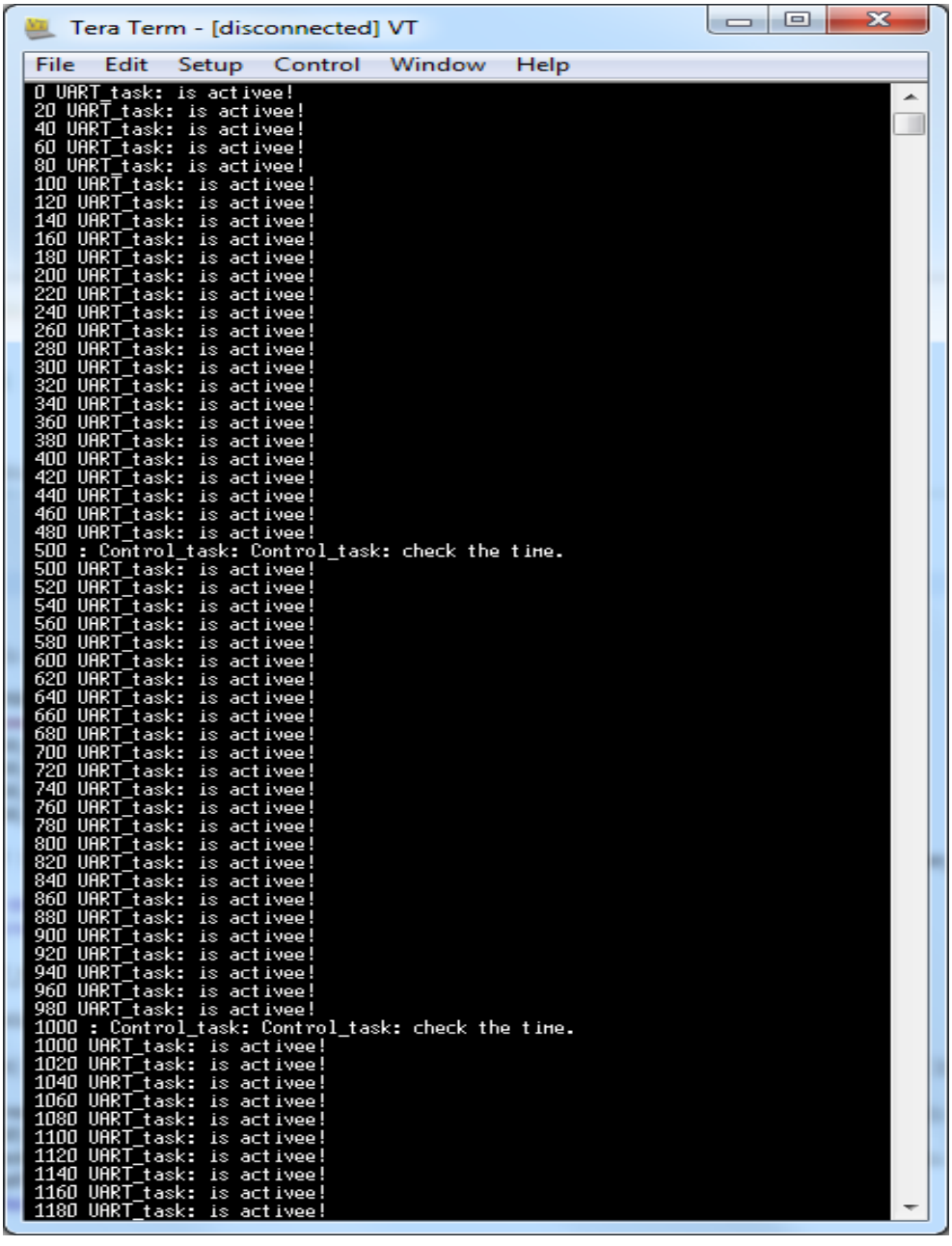


Printscreen 6: De output van de test 4 - Testcase 4.2

Test 5. Het testen van de periodiciteit van de uitvoering van UART_task.

Testcase	*Opstelling*	*Verwacht resultaat*	*Resultaat*
5.1	Wanneer de *UART_taak* loopt, print ze de melding: "*Current time.UART_task: is active*". Wanneer *UART_taak* klaar is met de uitvoering, print ze de melding: "*Current time. UART_task: the end of the execute*".	Het verschil tussen de tijd bij de melding:"*Current time. UART_task: the end of the execute*" en de volgende melding: "*Current time. UART_task: is active*" is 20msec.	*Correct* Zie Printscreen 7

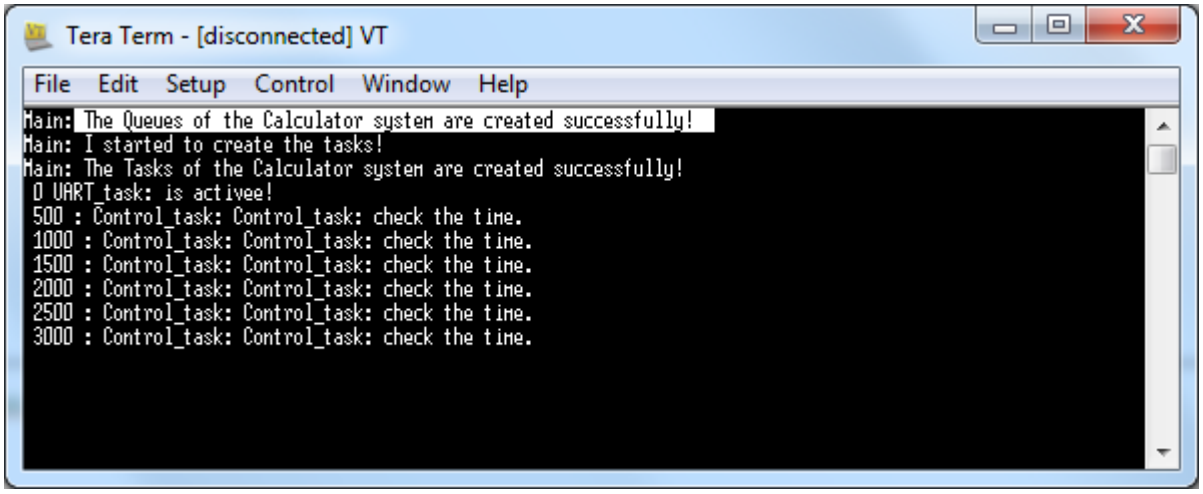
Printscreen 7: De output van de test 5 - Testcase 5.1



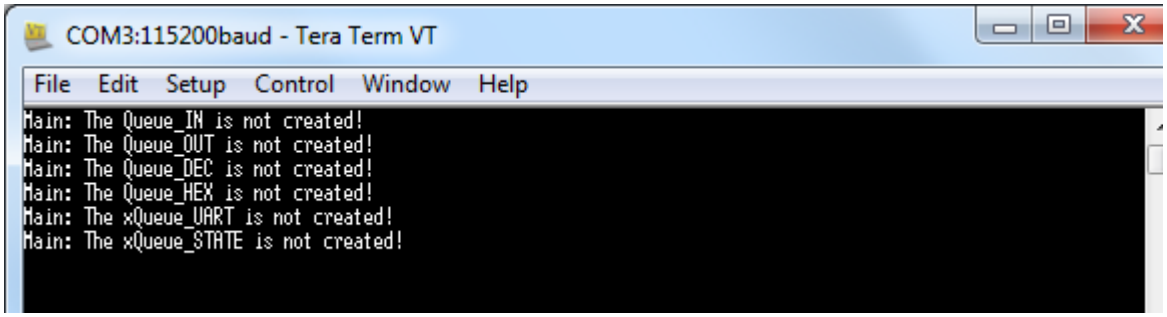
Test 6. Het controleren van de queue-configuratie

Testcase	*Opstelling*	*Verwacht resultaat*	*Resultaat*
6.1	Alle queues worden met correcte paarmeters gecreëerd.	Melding van het systeem: "*Main*: *The Queues of the Calculator system are created successfully."*	*Correct* Zie Printscreen 8
6.2	De *queue_UART* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_UART is not created"*	*Correct* Zie Printscreen 9
6.3	De *queue_STATE* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_STATE is not created"*	*Correct* Zie Printscreen 9
6.4	De *queue_IN* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue_IN is not created"*	*Correct* Zie Printscreen 9
6.5	De *queue _OUT* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue _OUT is not created"*	*Correct* Zie Printscreen 9
6.6	De *queue _ HEX* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue _HEX is not created"*	*Correct* Zie Printscreen 9
6.7	De *queue _ DEC* wordt incorrect gecreëerd.	Foutmelding: "*Main*: *The queue _DEC is not created"*	*Correct* Zie Printscreen 9

Printscreen 8: De output van de test 6 - Testcase 6.1



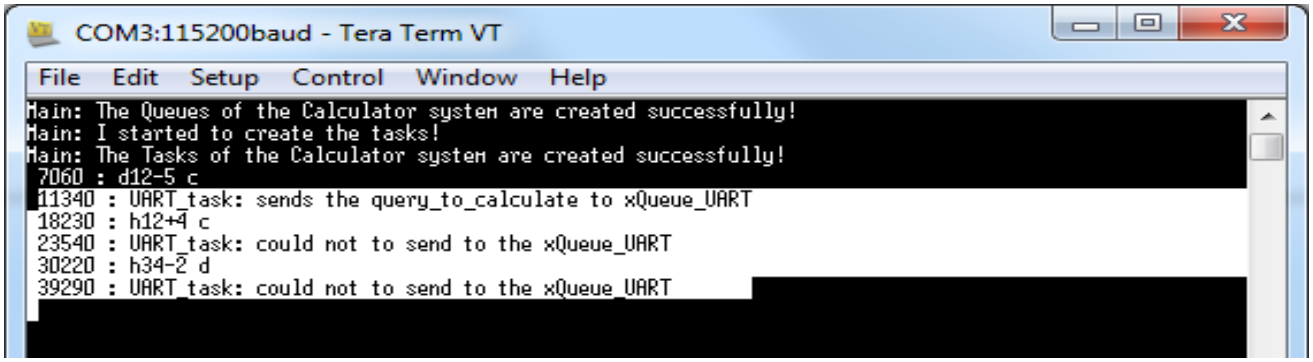
Printscreen 9: De output van de test 6 - Testcases 6.2 t/m 6.7



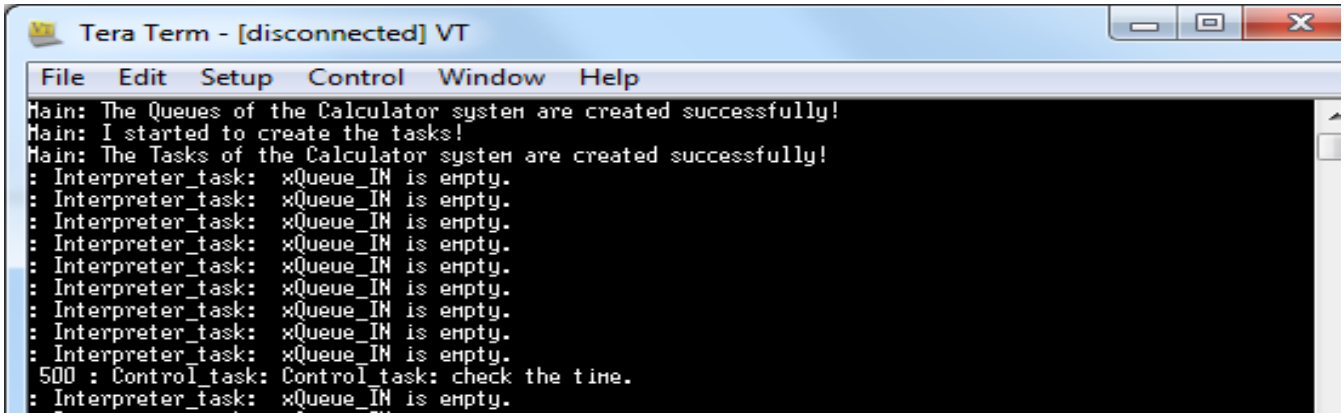
Test 7. Het controleren van de uitvoering van de functies *xQueueReceive(..)* en *xQueueSendToBack(..)*

Testcase	*Opstelling*	*Verwacht resultaat*	*Resultaat*
7.1	De *Control_task* bevat functie *xQueueReceive(..)* die de berichten uit *queue_UART* leest. Voor deze test wordt deze functie geblokkeerd. Daarna worden twee verzoeken ingevoerd. De *UART_task* stuurt alle verzoeken naar de *queue_UART* met de functie *xQueueSendToBack(..).* De returnwaarde van deze functie wordt gecontroleerd.	Na het tweede verzoek raakt de *queue_UART* vol. *Calculator* 1.0 geeft de foutmelding: "*UART_task: could not to send to the xQueue_UART*".	*Correct* Zie Printscreen 10
7.2	In de *Interpreter_task* wordt de functie *xQueueReceive(..)* aangeroepen. Deze functie probeert de *queue_IN* uit te lezen. Als de *queue_IN* leeg is, wordt de taak geblokkeerd. In deze gebruikscases wordt de returnwaarde van de functie *xQueueReceive(..)* gecontroleerd.	Als de *queue_IN* leeg is, geeft *Calculator* de foutmelding "*Interpreter_task: xQueue_IN is empty*."	*Correct* Zie Printscreen 11

Printscreen 10: De output van de test 7 - Testcases 7.1



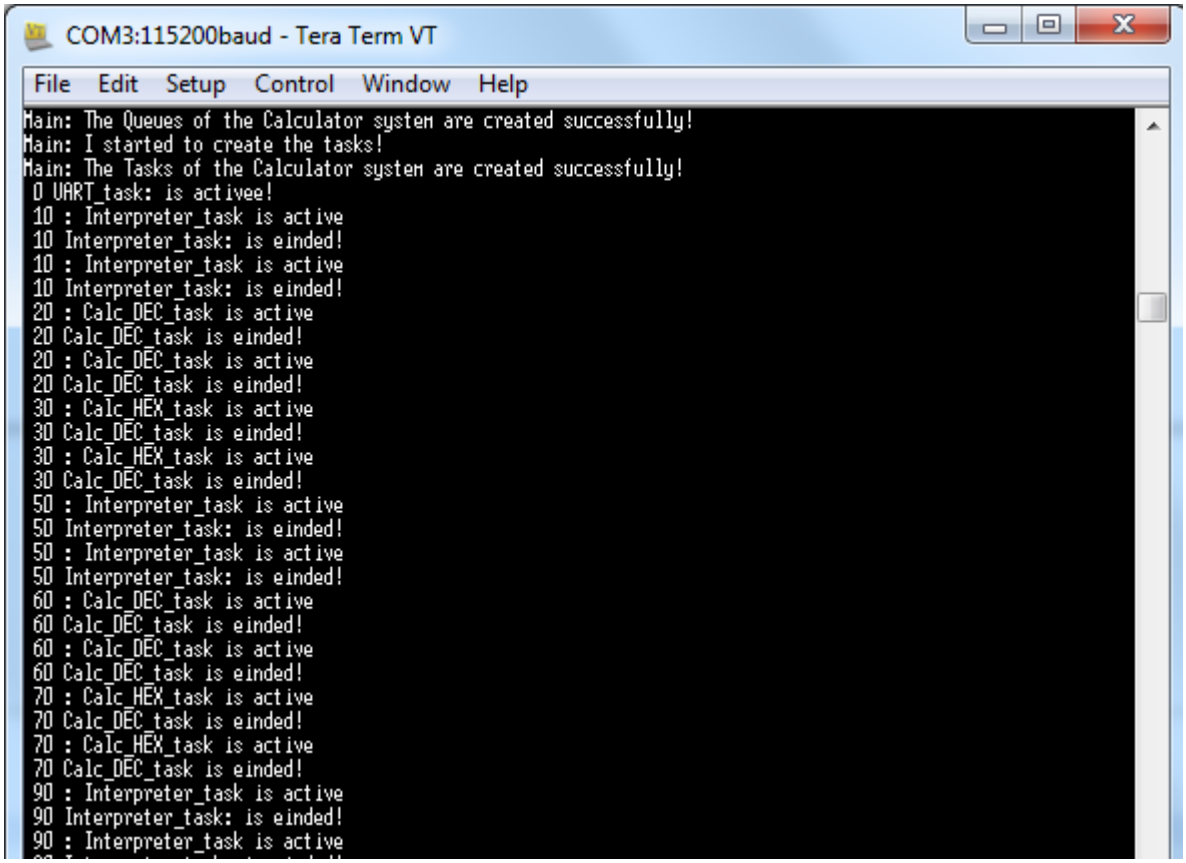
Printscreen 11: De output van de test 7 - Testcases 7.2



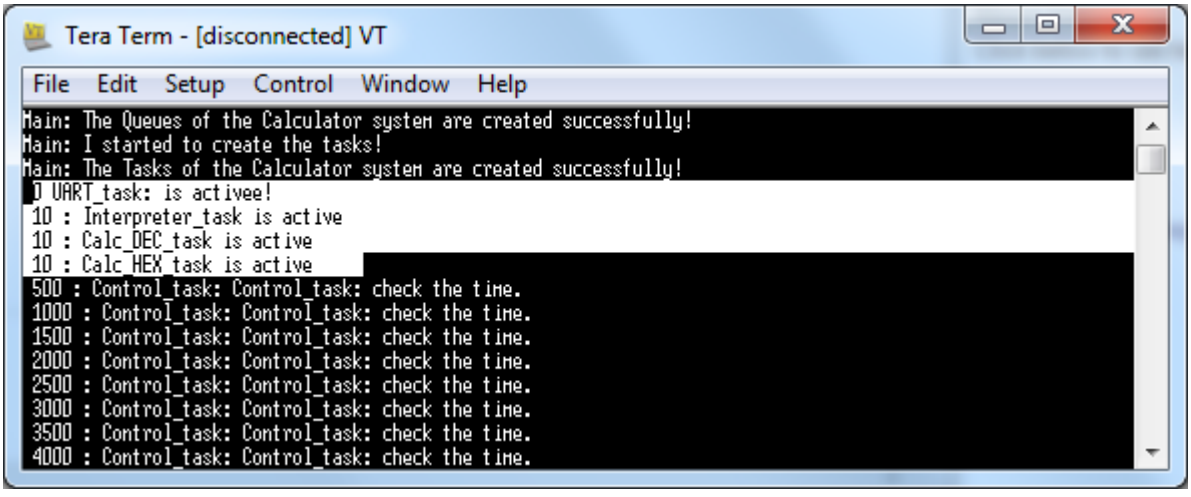
Test 8. Het controleren van de synchronisatie van processen die gebruik maken van queues

Testcase	***Opstelling***	***Verwacht resultaat***	***Resultaat***
8.1	In de taken *Interpreter_task, Calc_DEC_task* en *Calc_HEX_task* wordt in de functies *xQueueReceive(..)* de wachttijd op 0 geconfigureerd. Aan het begin en aan het eind van de taak van de uitvoering print elke taak respectievelijk de meldingen: *"Name_task: is active."* en *"Name_task: has ended."* Vervolgens wordt *Calculator* systeem gestart zonder dat de gebruiker een verzoek invoert.	In het FreeRTOS is het Round-Robin scheduling algoritme geïmplementeerd. Voor elke taak met dezelfde prioriteit geeft het systeem een bepaalde tijd voor de uitvoering: time-slicing. Bij deze gebruikscase wordt verwacht dat Calculator de volgende meldingen print. Binnen de eerste time-slicing (als de *Interpreter_task* als eerste is gecreëerd): "*Interpreter_task*: is active." "*Interpreter_task*: *has ended*." "*Interpreter_task*: is active." "*Interpreter_task*: *has ended*." … Binnen de tweede time-slicing (als de *Calc_DEC_task* als tweede is gecreëerd): "*Calc_DEC_task*: is active." "*Calc_DEC_task*: *has ended*." "*Calc_DEC_task*: is active." "*Calc_DEC_task*: *has ended*." …	*Correct* Zie Printscreen 12
8.2	In de taken *Interpreter_task, Calc_DEC_task* en *Calc_HEX_task* wordt in de functies *xQueueReceive(..)* de wachttijd op oneindig geconfigureerd. Aan het begin en aan het eind van de taakuitvoering print elke taak respectievelijk de meldingen: *"Name_task: is active."* en *"Name_task: has ended."* Vervolgens wordt *Calculator* gestart zonder dat de gebruiker een verzoek invoert.	Hier wordt verwacht dat de taken door de queue worden geblokkeerd tot het event optreedt. Bij deze gebruikscase wordt verwacht dat *Calculator* systeem één keer de volgende meldingen print: "*Interpreter_task*: is active." "*Calc_DEC_task*: is active." "*Calc_HEX_task*: is active."	*Correct* Zie Printscreen 13

Printscreen 12: De output van de test 8 - Testcases 8.1



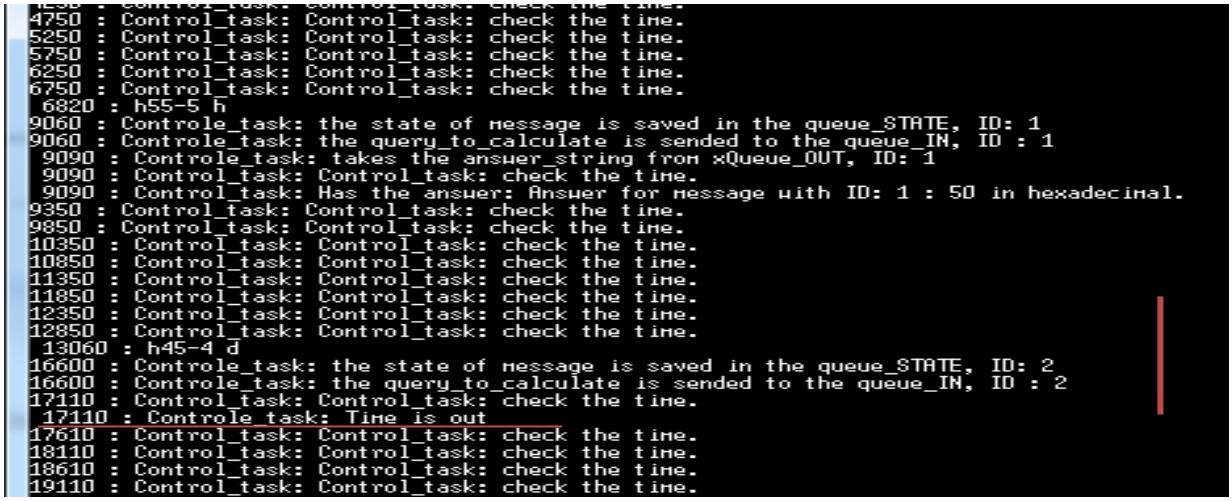
Printscreen 13: De output van de test 8 - Testcases 8.2



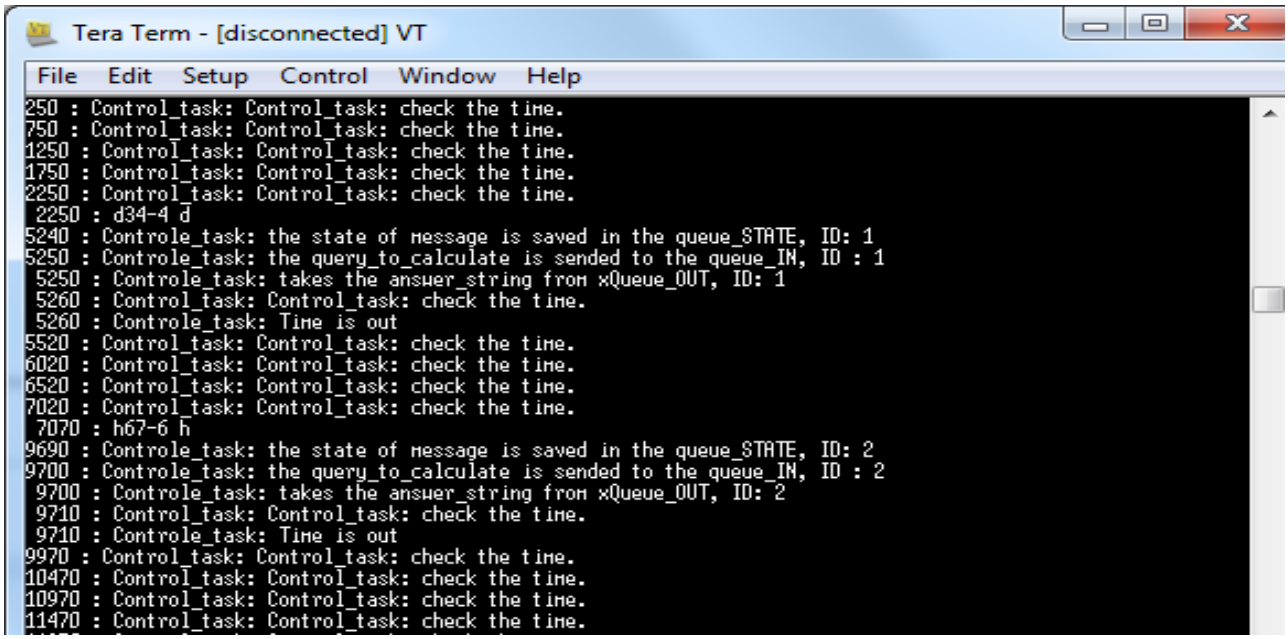
Test 9. Testen dat het systeem reageert op de niet-gehaalde deadline

Testcase	*Opstelling*	*Verwacht resultaat*	*Resultaat*
9.1	In de *Calc_DEC_task* wordt de functie *xQueueSendToBack(..)* geblokkeerd. Deze functie stuurt het antwoord naar de *queue_OUT*. Vervolgens wordt het verzoek ingevoegd: *d34-4 d*.	Na 500msec geeft *Calculator* de foutmelding "*Controle_task: Time is out*".	*Correct* Zie Printscreen 14
9.2	Om één verzoek van de gebruiker te berekenen heeft *Calculator* 30msec nodig. De timeout van het systeem is 500msec. In deze testcase wordt de timeout van het systeem verminderd tot 10msec.	Het systeem haalt nooit de deadline, 10msec na invoering van het verzoek komt de foutmelding "*Controle_task: Time is out*".	*Correct* Zie Printscreen 15

Printscreen 14: De output van de test 9 - Testcase 9.1



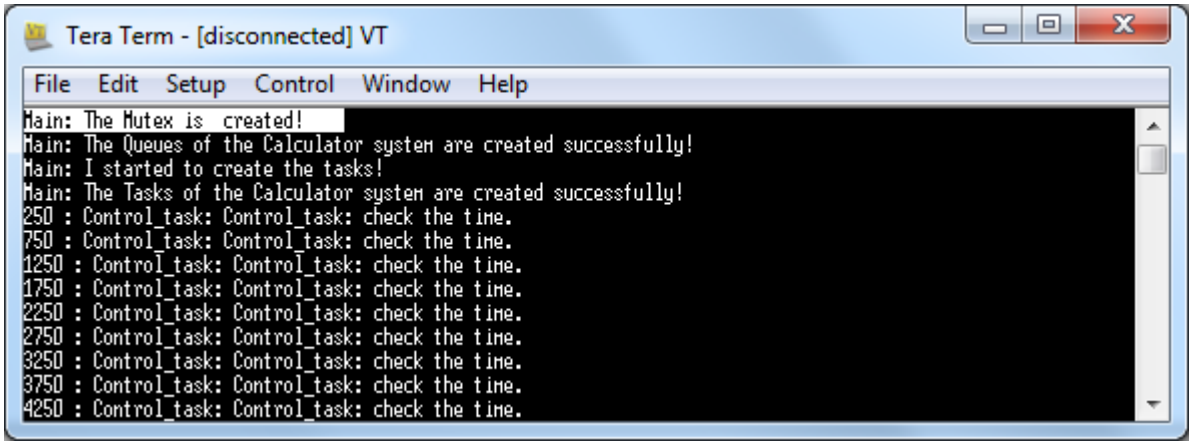
Printscreen 15: De output van de test 9 - Testcase 9.2



Test 10. Het controleren van de Mutex-configuratie.

Testcase	*Opstelling*	*Verwacht resultaat*	*Resultaat*
10.1	De Mutex is gecreëerd.	Melding van het systeem: "*Main*: *The Mutex is created successfully.*"	*Correct* Zie Printscreen 16

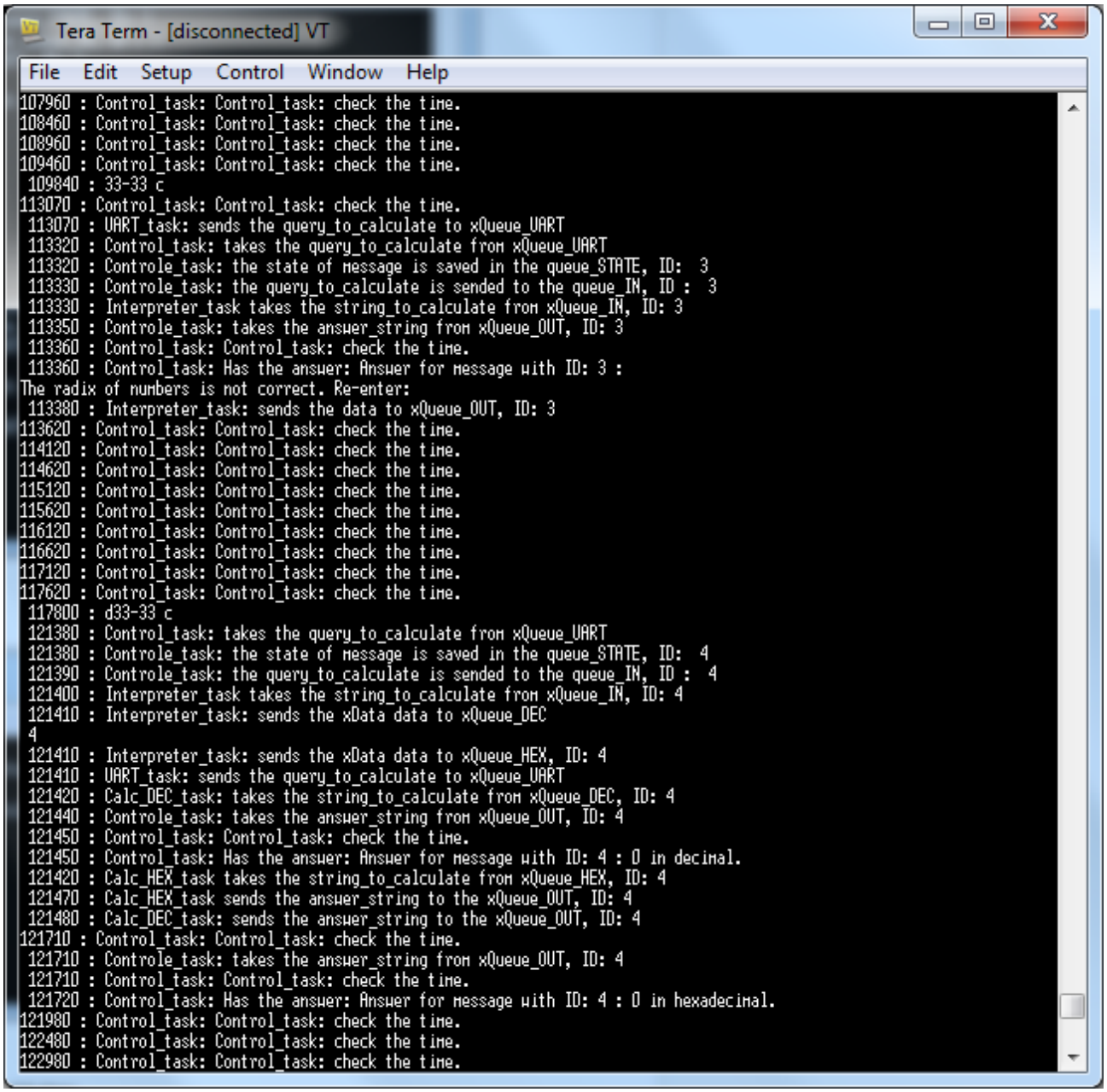
Printscreen 16: De output van de test 10 - Testcase 10.1



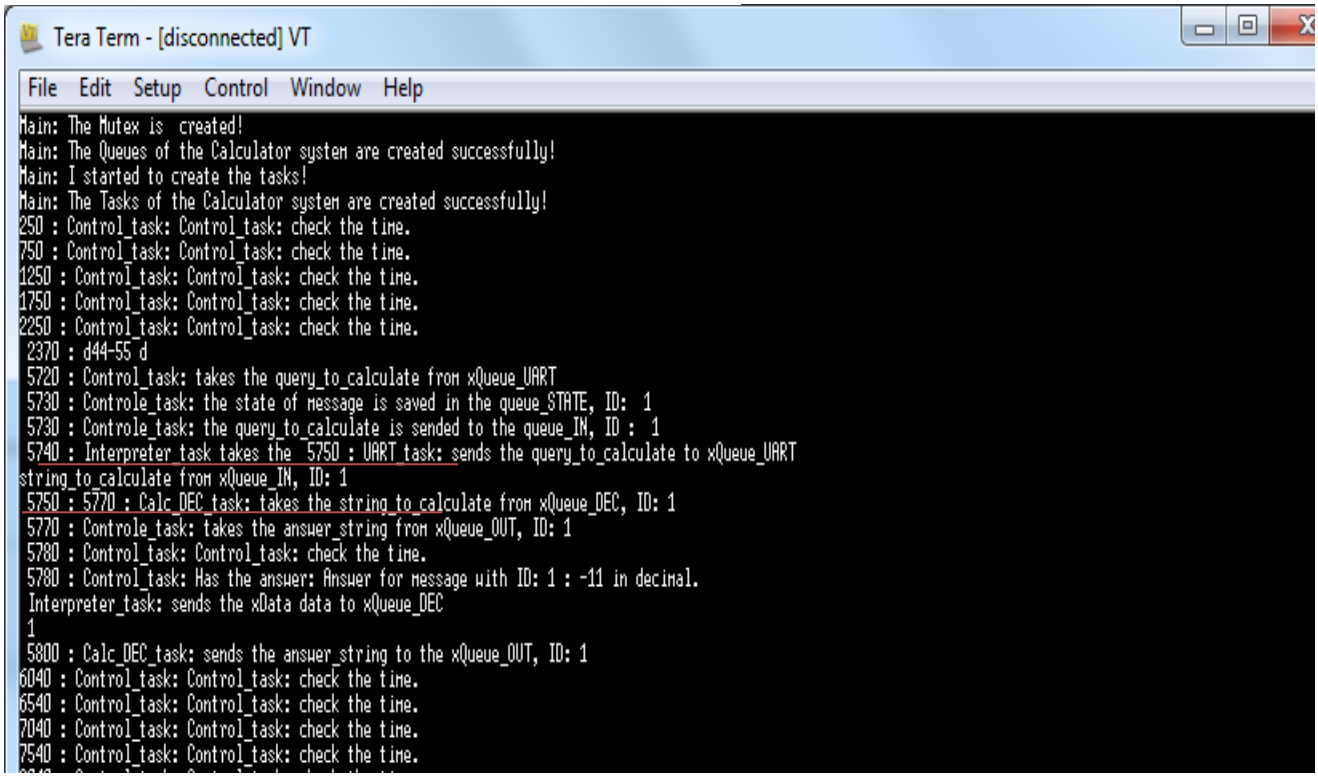
Test 11. Het controleren van de beveiliging van de seriële port door Mutex.

Testcase	*Opstelling*	*Verwacht resultaat*	*Resultaat*
11.1	De toegang tot de seriële port is door Mutex beveiligd.	Elk geprinte melding is afgerond	*Correct* Zie Printscreen 17
11.2	De toegang tot de seriële port is niet door Mutex beveiligd.	*Calculator* print afgebroken meldingen.	*Correct* Zie Printscreen 18

Printscreen 17: De output van de test 11 - Testcase 11.1



Printscreen 18: De output van de test 11 - Testcase 11.2

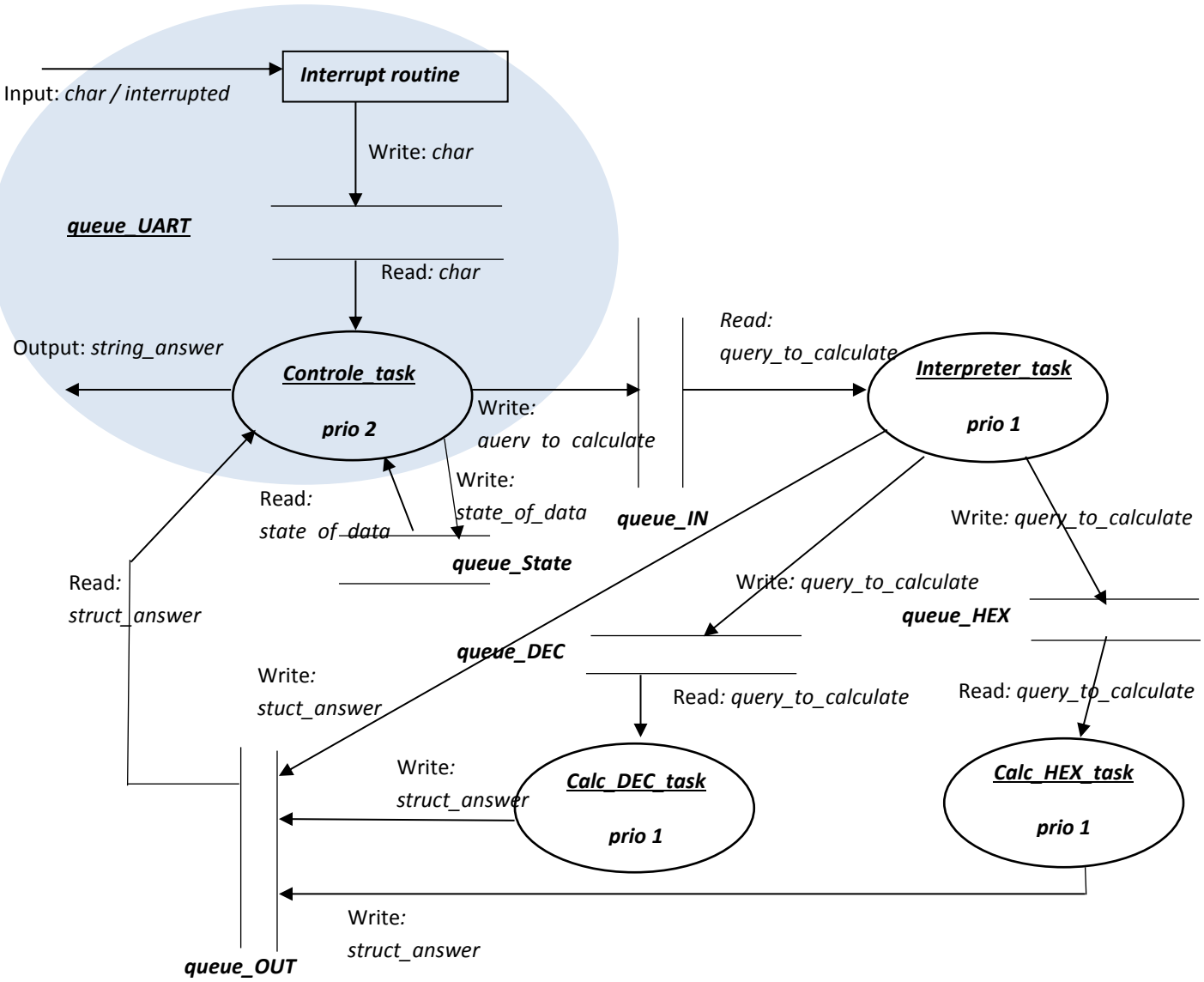


Test 12. Stresstest.

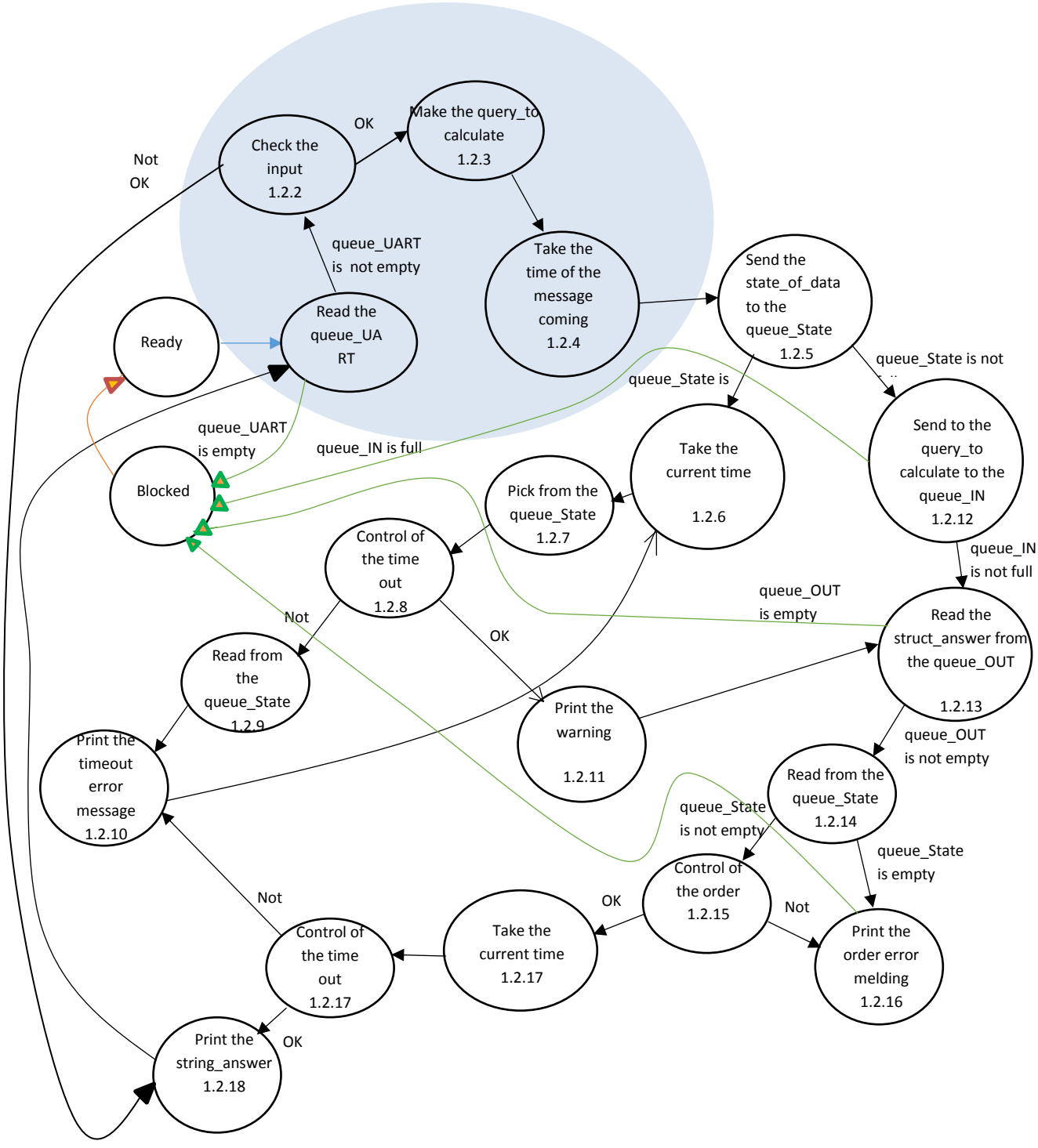
Testcase	*Opstelling*	*Verwacht resultaat*	*Resultaat*
12.1	*Calculator* wordt gestart. Een tekstbestand met 20 verzoeken wordt naar het systeem gestuurd.	*Calculator* berekent alle verzoeken vanuit het bestand en geeft de antwoorden.	*Niet Correct*

Bijlage X. Yourdan-diagrammen voor de demoapplicatie 1.1

System: Calculator 1.1 - Layer 1: Data flow diagram



Running state: Program Flow diagram: *Control_task*



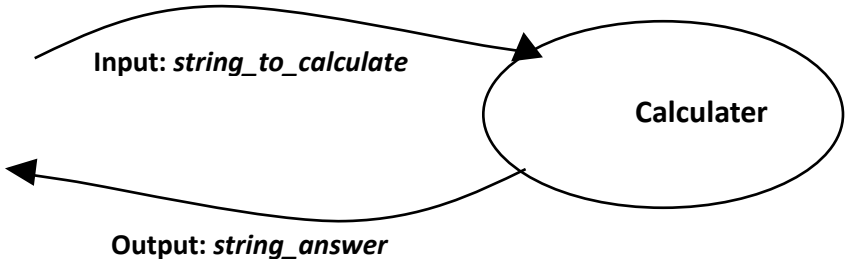
Bijlage XI. Yourdan-diagrammen voor de demoapplicatie 2.0

Systeem: *Calculator 2.0* Context:

The purpose of Calculator 2.0 is: gives the answer for the calculation expression. This system also allows you to configure the message of the tasks execution.

	Calculator							
	performs the operations between two numbers (max 32 bits number) and gives the answer in hexadecimal or decimal radix (depends on the user's query)							
Action	'*' - multiplying		'/' - division		'+' addition		'- 'subtraction	
Numeral system	Hexadecimal –H/h	Decimal – D/d	Hexadecimal H/h	Decimal D/d	Hexadecimal H/h	Decimal D/d	Hexadecimal H/h	Decimal D/d

Context diagram:



The systems uses next types of message:

1. STRUCT_QUERY – the structure with string for calculate or configuration and PACKET_HEADER;
2. QUERY_TO_CALCULATE – the structure with the two numbers for calculate and PACKET_HEADER;
3. PACKET_HEADER- framework, necessary for the identification of the message
4. STRUCT_ANSWER – the structure with the answer string and PACKET_HEADER;

Name of message	***QUERY_TO_CALCULATE***					
example	{{ PACKET_HEADER }, 126, 673,'0', '+', 'h'}					
data in the structure	PACKET_HEADER	First number	Second number	Code error	Operand	Radix of input
Name in the structure	header	opr_1	opr_2	code_error	com	Input_radix
Type of data in the structure	PACKET_HEADER	Int32_t	Int32_t	char	char	char
size	32 bits	32 bits	32 bits	8 bits	8 bits	8 bits
Possible input		Max number: 2147483647 in dec 7FFFFFFF in hex Min number: - 2147483648 in dec - 80000000 in hex	Max number: 2147483647 in dec 7FFFFFFF in hex Min number: - 2147483648 in dec - 80000000 in hex	'0' - no errors; '1' - input_id is not correct; '2' - the number is not correct; '3'- the Input_radix is not correct; '4' - the operand is not correct;	'-' '+' '/' '*'	'h' or 'H' – hexadecimal; 'd' or 'D' - decimal

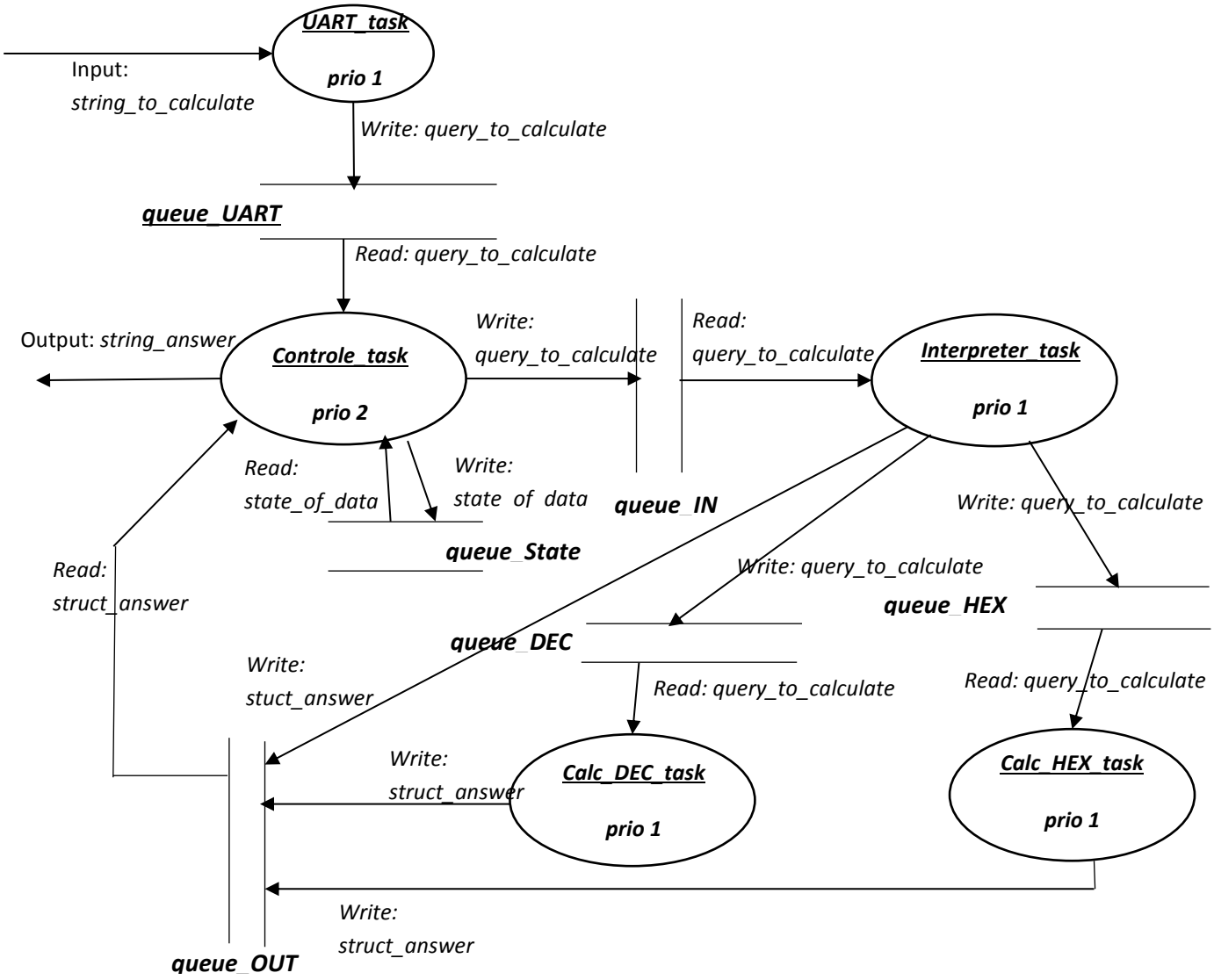
Name of message	***STRUCT_QUERY***	
Type	structure	
example	{{ PACKET_HEADER }, 126*673 H}	
Type of data in the structure	*PACKET_HEADER* - it contains *ID , input_id and data length*	String with the expression to calculate or with the expression to syslog configuration
Name in the structure	*header*	*string_query[25]*
Max size of structure	32 bits	25 bytes

Name of message	***PACKET_HEADER***		
type	structure		
example	{1,'d',15}		
Type of data in the packet_header	uint16_t	char	uint8_t
Name in the structure	ID	input_id	data_length
Significance	ID	Desired radix of output numbers or query for syslog configuration	Het number of chars in the data
Max size of packet_header	16 bits	8 bits	8 bits
Possible data	identification number	'h' or 'H' – hexadecimal; 'd' or 'D' – decimal; 'c' or 'C' – hexadecimal and decimal; 'S' - de query for syslog configuration;	The numbers of chars in the data

Name of message	***STRUCT_ANSWER***		
Type	structure		
Example	{{1,'h',5}, 84798}		
Type of data in the sruct_answer	packet_header	string	
Name in the structure	header	string_answer	
Max size of *sruct_answer*	32 bits	30 bytes	
Type of answer		Calculated number	
		Max number: 9223372036854775807 in dec 7FFFFFFFFFFFFFFF in hex Min number: -9223372036854775808in dec - 8000000000000000 in hex	"The input id of numbers is not correct. Re-enter:" "The numbers are not correct. Re-enter:" "The output radix of numbers is not correct. Re-enter:" "The operand is not correct. Re-enter:"

System: Calculator 2.0 - Layer 1: Data flow diagram

The global propose of this system is calculate of the query of the user. This system allows to see internal work processes at the at various levels. The internal processes that will be displayed depend upon the syslog configuration. This will be later described in more detail.



Systeem: Calculator 2.0 - Layer 2:

1.1 *UART_task*: This task will be replaced by the interrupt routine.

The purpose of the UART_task:

- listen the UART-port, if came a char -> sends to the *queue_UART*

This task has the priority 3

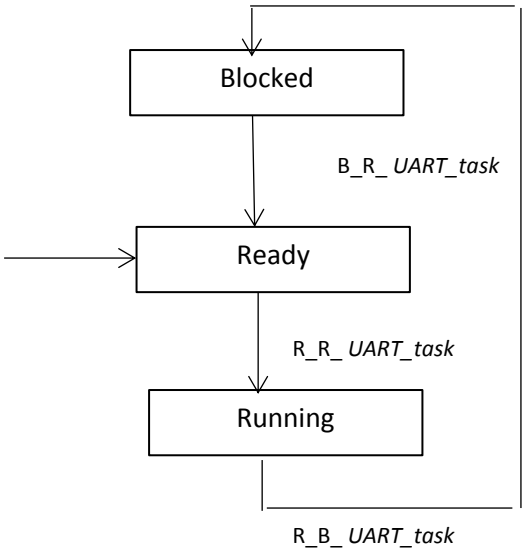
The task is blocked if no events.

Events:

1. Timer event: every 20msec
2. The input cames to the UART-port

Processes of *UART_task*	**Explanations**
Check the UART register	Every 20 milliseconds the task checks the input register of the UART port
Read UART-port	The task tries to read UART port
Write the char to *queue_UART*	The task writes the input char to *queue_UART*

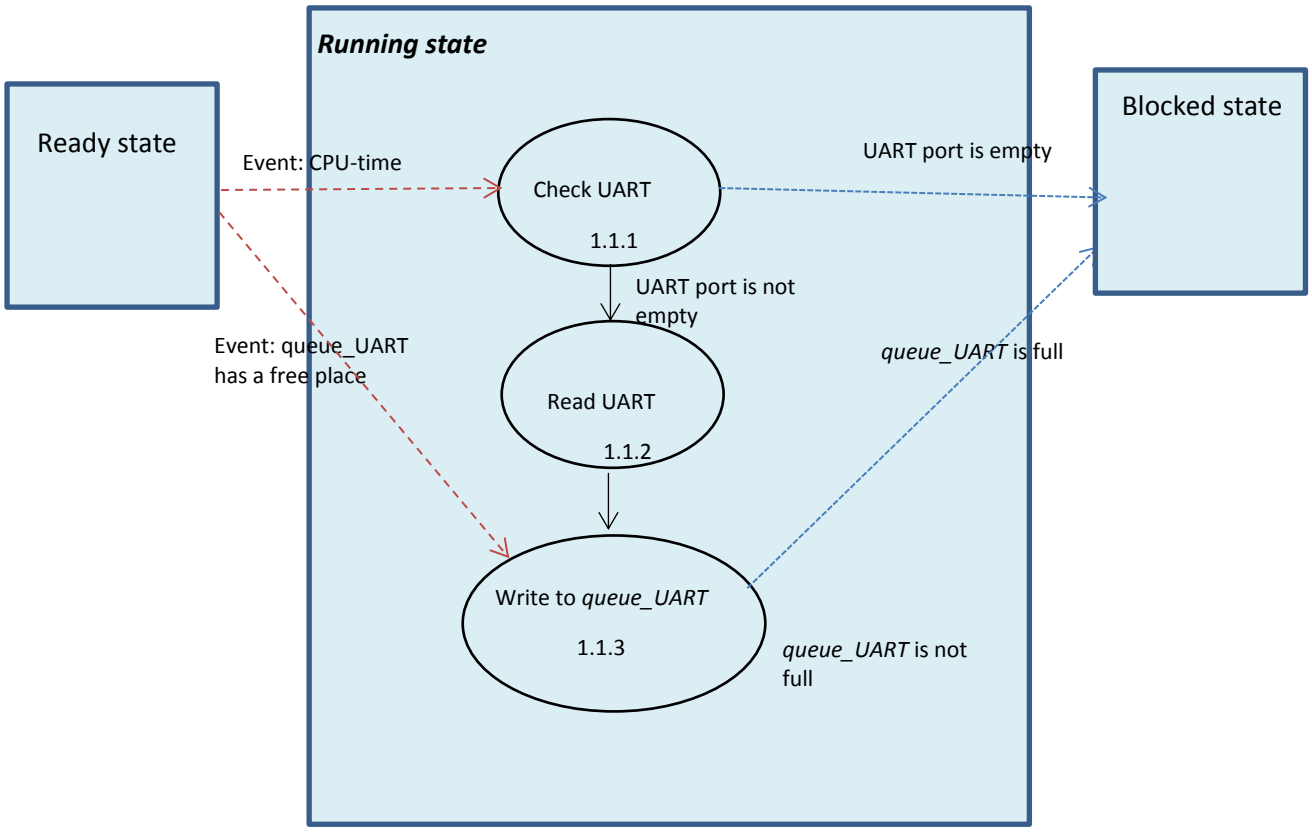
State transition diagram of Uart_task:



State Table of Uart_task:

	Current State	Event	Action	Condition	Next state	Transition
1	Blocked	-- CPU gives the time; --the another task frees the place in *queue_UART*;		-- 20msec have passed -- the *UART_task* has waiting *queue_UART;*	Ready	B_R_ *UART_task*
2	Ready		-- continues last stopped operation; --write to the queue_UART;	-- *UART_task* was unblocking CPU time; -- *UART_task* was unblocking *queue_UART*;	Run	R_R_*UART_task*
3	Run	-- CPU time is out -- queue_UART is full	-- stop operation; -- *UART_task* wait *queue_UART*;	-- *UART_task* blocked	Blocked	R_B_*UART_task*

Running state: Program Flow diagram: UART_task



1.2 Control_task.

The purpose of *Control_task*:

- read *queue_UART*;
- check the length of the message string: if > MAX -> error message and query deleted;
- checks *input_id* ->determines the destination: is syslog configuration query or query to calculate;
- takes the time
- convert the received chars to structure *STRUCT_QUERY*;
- fills in the fields of the header of the message*: ID, input_id, length*.
- if the query to syslog configuration - > send *struct_query* to *queue_SYSLOG*;
- if *query_to_calculate* -> send *struct_query* to *queue_IN*;
- read *struct_answer* from *queue_OUT*;
- time out control – if ((no answer after 0.5 sec=> time is out => syslog error message

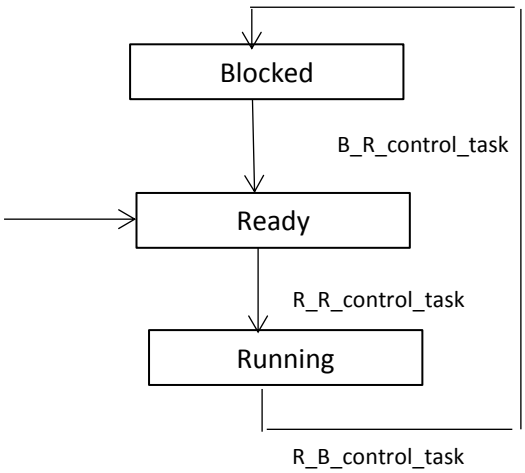
This task has a **priority 3**. This task is in the blocked state until an event occur.

Events:

1. The char came to *queue_UART*;
2. *struct_answer* came to *queue_OUT*
3. Timer event: every 0.5sec

Processes of *Control_task*	**Explanations**
Read *queue_UART*	The task reads *queue_UART* until came the char '\r' binnen or the input contains more then 26 chars
Check length of the message string	If input>26 → syslog the error messge
Convert the received chars to the structure STRUCT_QUERY;	Make *STRUCT_QUERY*: Initialisation of the HEADER: {*ID, input_id, length*}; *string_query* ;}
Checks *input_ID*	Check the first char of the message: If first cahr == 's' → this message for *queue_SYSLOG*; else → this message for *queue_IN*;
Take the time of the message	
Write the time with ID of the message into the buffer	
Write *struct_query* to *queue_IN*	If query for *queue_IN*;
Write *struct_query* to *queue_SYSLOG*	If query for *queue_SYSLOG*;
Read *queue_OUT*	If *queue_OUT* is not empty => read *queue_OUT*
Control of the time out	if ((no answer after 0.5 sec)&&(*mes_counter!=0)*=> time is out => print error message
Syslog *string_answer* to *queue_PRINT*	

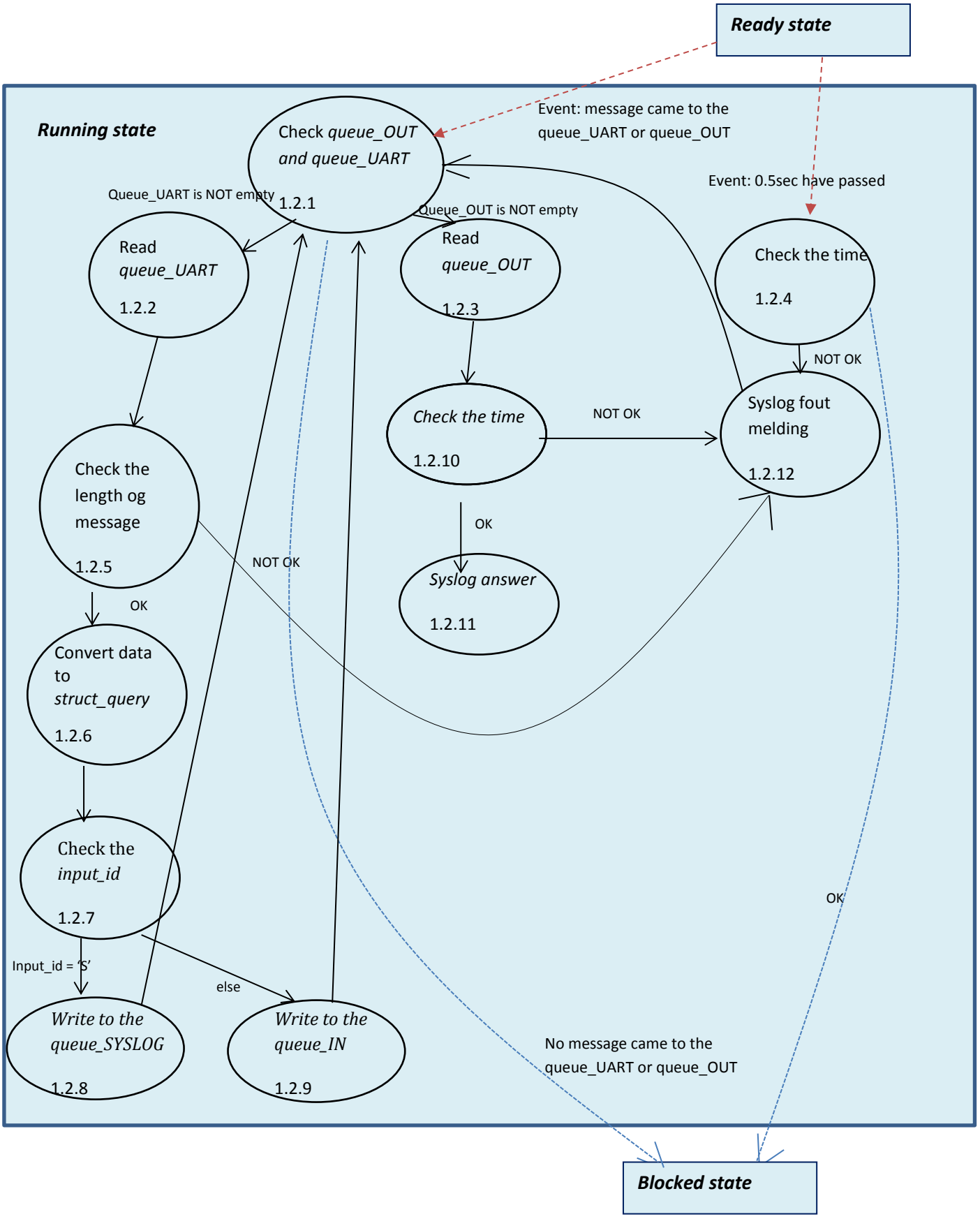
State transition diagram of Control_task:



State Table of Control_task:

	Current State	**Event**	**Action**	**Condition**	**Next state**	**Transition**
1	Blocked	---The another task writes to *queue_UART*; ---The another task writes to *queue_OUT*; -- CPU gives the time ;		-- 0.5sec have passed	Ready	B_R_control_task
2	Ready	--emergence of an element in *queue_UART;* --emergence of an element in *queue_OUT;* -- *queue_OUT* is empty meer then 0.5 sec;	--read from *queue_UART*; -- read from *queue_OUT*; -- check the time;	-- *Control_task* was unblocking *queue_UART*; -- *Control_task* was unblocking *queue_ OUT* ;	Run	R_R_control_task
3	Run	--*queue_UART* is empty && *queue_OUT* is empty	-- *Control_task* wait *queue_UART* || *Control_task* wait *queue_OUT*;		Blocked	R_B_control_task

Running state: Program Flow diagram: Control_task



1.6 Syslog_task

The propose of this task is configured the syslog mode.

- read *queue_SYSLOG;*
- check *struct_query.strung_query:* if error = > error melding;
- syslog configured;
- read *queue_PRINT*
- print the message

This task has the **priority 1** and the task is blocked if are no events.

Events:

1. The data came in *queue_SYSLOG*
2. The data came to *queue_PRINT*

Processes of *Syslog_task*:	**Explanations**
Read queue_ *SYSLOG*	
Convert *string_query* to configured commands and Check *struct_query*	If error = > print error message;
Configure the syslog	

The first char in *struct_query.string_query* determines what to be configured: source or level;

The second char in *struct_query.strung_query* determines how to be configured: clean or write;

	Configuration mode			
struct_query.strung_query[0]	'L' - level		'S' - source	
struct_query.strung_query[1]	'W' - write	'C' - clean	'W' - write	'C'- clean
Example	"LW…" – level write; "SW…" – source write; "LC…" – clean level; "SC…" – clean source;			

If *struct_query.strung_query[0][1]== "SW..":*

Third char and next in *struct_query.strung_query[2]…[6]*							
UART_task	**Control_task**	**Interpreter_task**	**Calc_DEC_task**	**Calc_HEX_task**	**Syslog_task**	**Diagnose_task**	**Al tasks**
Print message for *UART_task*	Print message for *Control_task*	Print message for *Interpreter_task*	Print message for *Calc_DEC_task*	Print message for *Calc_HEX_task*	Clear *Syslog_task* from syslog	Clear *Diagnose_task* from syslog	Print message for *al tasks*
'u'	**'c'**	**'i'**	**'d'**	**'h'**	**'s'**	**'t'**	**'a'**

If *struct_query.strung_query[0][1]== "SC..":*

Second char and next in *struct_query.strung_query[2]…[6]*							
UART_task	**Control_task**	**Interpreter_task**	**Calc_DEC_task**	**Calc_HEX_task**	**Syslog_task**	**Diagnose_task**	**Al tasks**
Clear *UART_task* from syslog	Clear *Control_task* from syslog	Clear *Interpreter_task* from syslog	Clear *Calc_DEC_task* from syslog	Clear *Calc_HEX_task* from syslog	Clear *Syslog_task* from syslog	Clear *Diagnose_task* from syslog	Clear *al tasks* from syslog
'u'	**'c'**	**'i'**	**'d'**	**'h'**	**'s'**	**'t'**	**'a'**

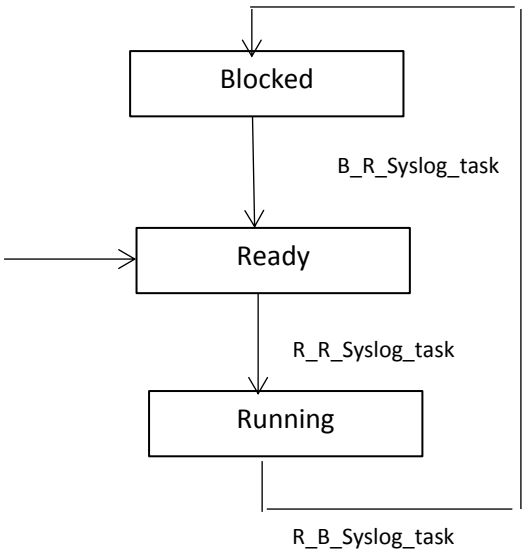
if *struct_query.strung_query[0][1]== "LW..":*

Second char and next in *struct_query.strung_query[2]…[6]*		
CRITTCAL level	Debug Level	Al levels
'c'	**'d'**	**'a'**
Print error messges	Print messages in debug mode	Print error and debug messages

if *struct_query.strung_query[0][1]== "LC..":*

Second char and next in *struct_query.strung_query[2]…[6]*		
CRITTCAL level	Debug Level	Al levels
'c'	**'d'**	**'a'**
Not print error messges	Not print messages in debug mode	Print nothing

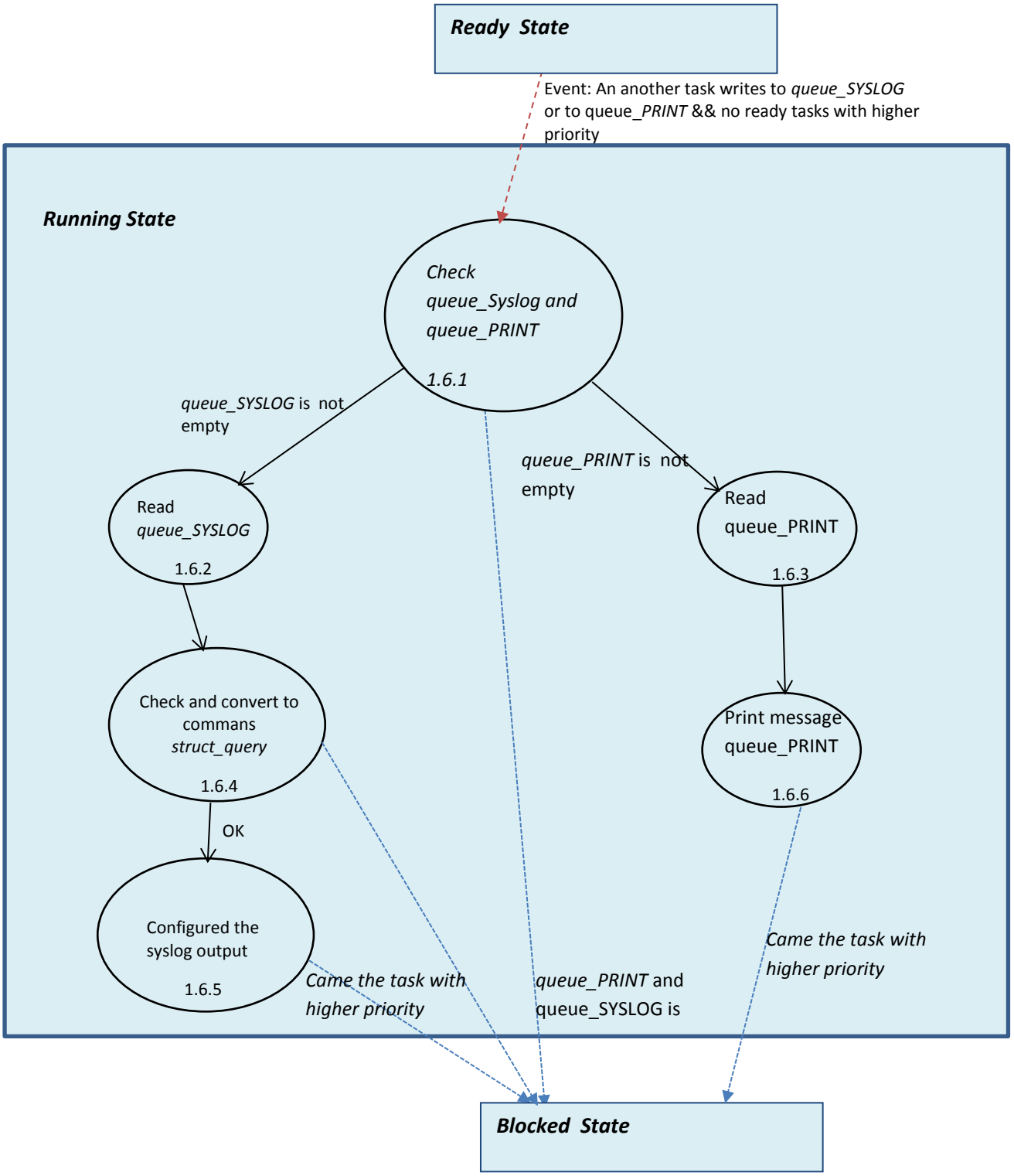
State transition diagram of Syslog_task:



State Table of Syslog_task

	Current State	**Event**	**Actie**	**Condition**	**Next state**	**Transition**
1	Blocked	-- An another task writes to *queue_SYSLOG*; -- An another task writes to *queue_PRINT*;		No another tasks with higher priority	Ready	B_R_Syslog_task
2	Ready	Came the turn	--read from *queue_ SYSLOG;* --read from *queue_ PRINT ;*		Run	R_R_ Syslog _task
3	Run	-- *queue_SYSLOG* is empty; -- *queue_PRINT* is empty; -- scheduler event;	-- stop operation;	-- *Syslog_task* blocked ;	Blocked	R_B_ Syslog _task

Running state: Program Flow diagram: Syslog_task



1.3 Interpreter_task

The purpose of *Interpreter_task:*

- read from *queue_IN;*
- control *struct_queury*: if error =>send the *struct_answer* with the error message to *queue_OUT*;
- send *query_to_calculate* to *queue_DEC* or/and *queue_HEX* (depends on the user's request)

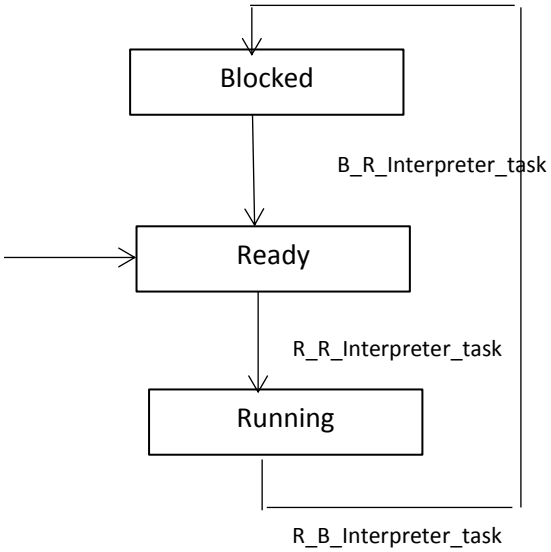
This task has **priority 1**. The task is blocked if are no events.

Events:

1. The data came to *queue_IN*

Processes of *Interpreter_task*	**Explanations**
Read *queue_IN*	
Check *struct_queury*	If there are errors => write the fout melding to *queue_OUT*
Convert *struct_queury* to the structuur *query_to_calculate*	If no errors in *struct_queury*
Write to *queue_DEC*	If *query_to_calculate.header.radix_id* is 'D'/'d' or 'C'/'c' and *struct_query.header.code_error=0*
Write to *queue_HEX*	If the If *query_to_calculate.header*. *header.radix_id* is 'H'/'h' or 'C'/'c' and *struct_query.header.code_error=0*
Write to *queue_OUT*	If there are errors in *struct_query*

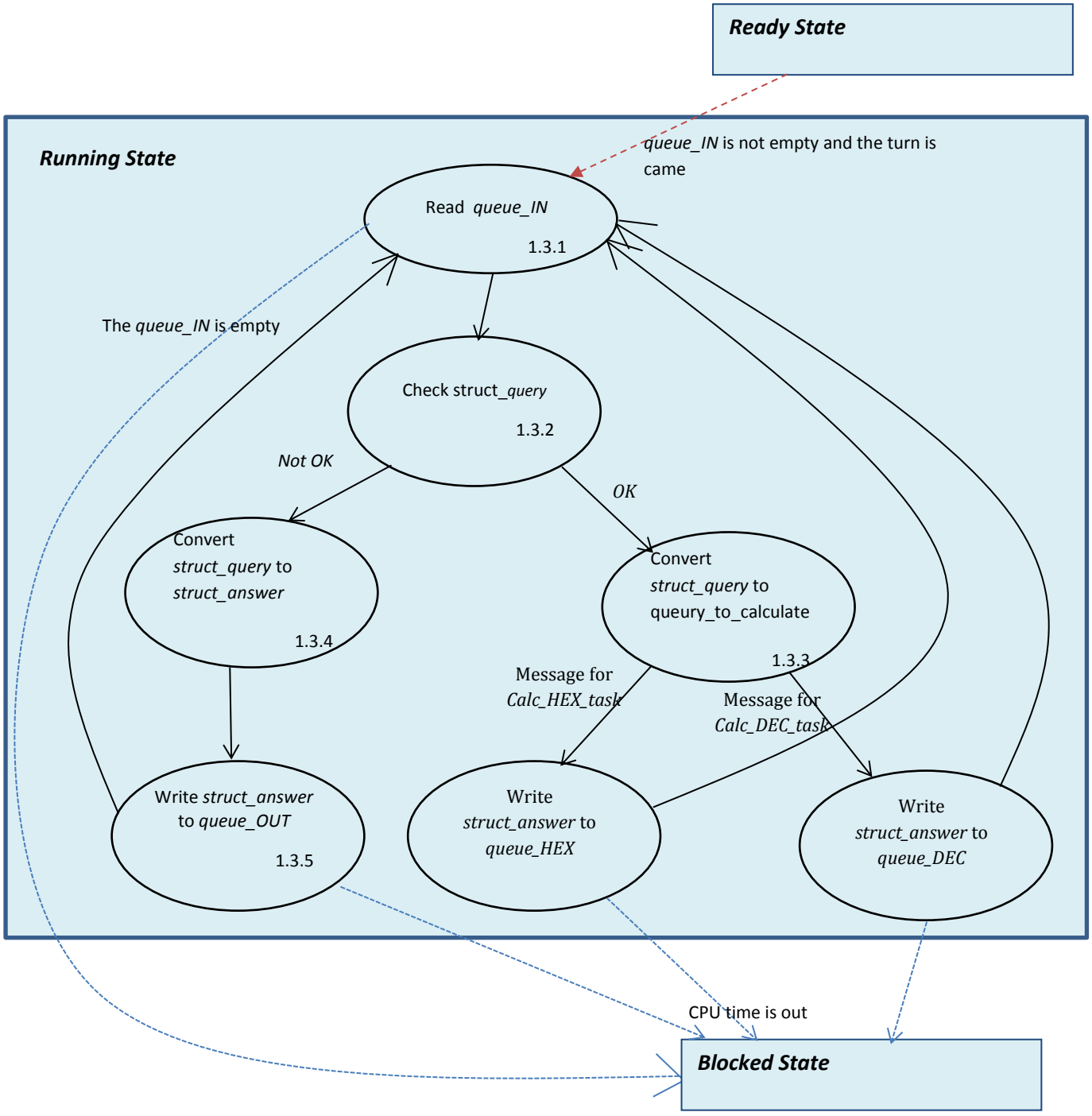
State transition diagram of Interpreter_task:



State Table of *Interpreter_task*

	Current State	Event	Actie	Condition	Next state	Transition
1	Blocked	---The another task writes to *queue_IN* and the turn is came;		-- *Interpreter_task* was waiting *queue_IN* and no tasks in Ready state met higher priority;	Ready	B_R_Interpreter_task
2	Ready	Came the turn	--read from *queue_IN*;		Run	R_R_Interpreter_task
3	Run	-- CPU time is out --queue_IN is empty	-- stop operation; -- the *Interpreter_task* wait queue_IN;	-- *Interpreter_task* blocked ;	Blocked	R_B_Interpreter_task

Running state: Program Flow diagram: *Interpreter_task*



1.4 Calc_DEC_task

The purpose of *Calc_DEC_task*: calculation of the user's query and convert the answer to the decimal numeral system.

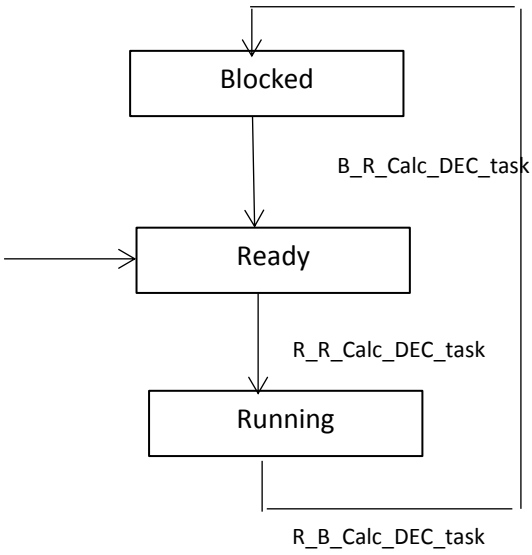
This task has priority 1. The task is blocked if are no events.

Events:

1. The element came to *queue_DEC*;

Processes of *Calc_DEC_task*	Explanations
Read *queue_DEC*	
Check *query_to_calculate.input_radix*	if *query_to_calculate.input_radix* == 'h' or 'H' => convert to decimal
Calculate the query	
Write to *queue_OUT*	*sruct_answer*

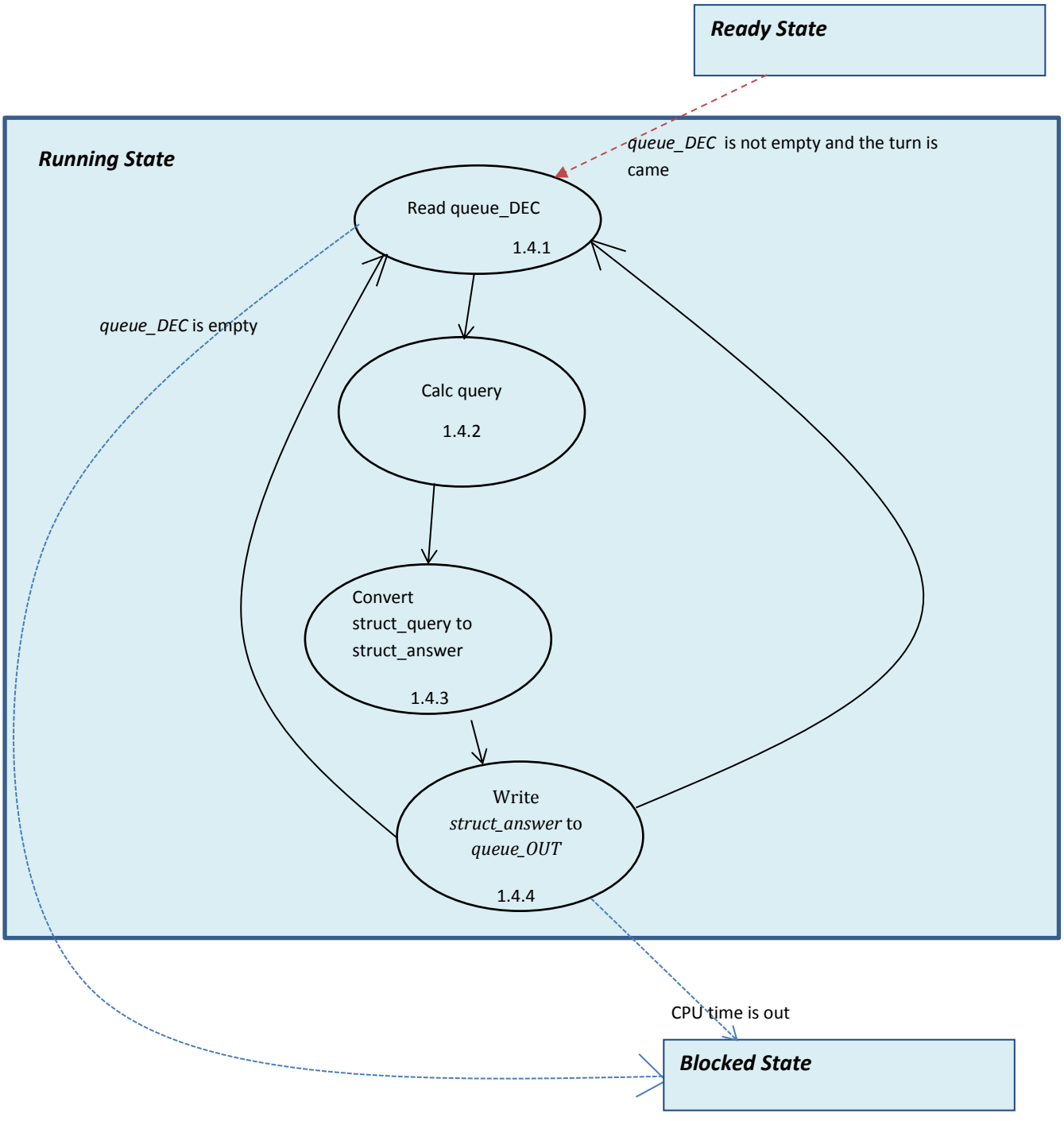
State transition diagram of *Calc_DEC_task:*



State Table of the *Calc_DEC_task*

	Current State	Event	Actie	Condition	Next state	Transition
1	Blocked	--The message came to *queue_DEC and the turn came*;		-- *Calc_DEC_task* was waiting *queue_DEC* and no tasks in Ready state met higher priority;	Ready	B_R_Calc_DEC_task
2	Ready	Came the turn	--read from queue_DEC;		Run	R_R_Calc_DEC_task
3	Run	-- CPU time is out --queue_DEC is empty	-- stop operation; -- Calc_DEC_task wait queue_DEC;	-- Calc_DEC blocked ;	Blocked	R_B_Calc_DEC_task

Running state: Program Flow diagram: *Calc_DEC_task*



1.4 *Calc_HEX_task*

The purpose of *Calc_HEX_task*: calculation of the user's query and convert the answer to the hexadecimal numeral system.

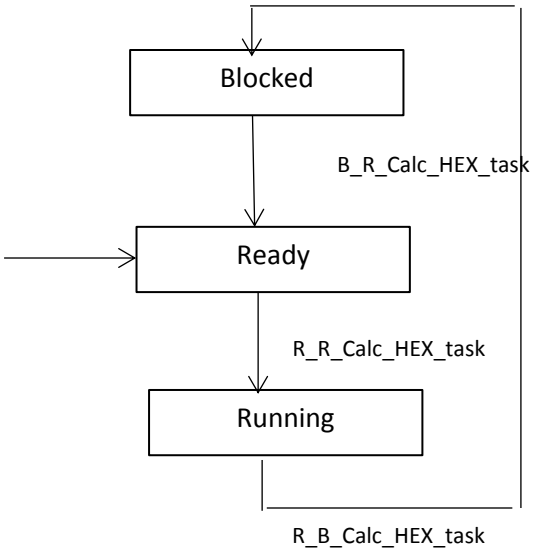
This task has priority 1. The task is blocked if are no events.

Events:

2. The element came to *queue_HEX*;

Processes of *Calc_HEX_task*	**Explanations**
Read *queue_HEX*	
Check *query_to_calculate.input_radix*	if *query_to_calculate.input_radix* == ‘d’ or ‘D’ => convert to hexadecimal
Calculate the query	
Write to *queue_OUT*	*sruct_answer*

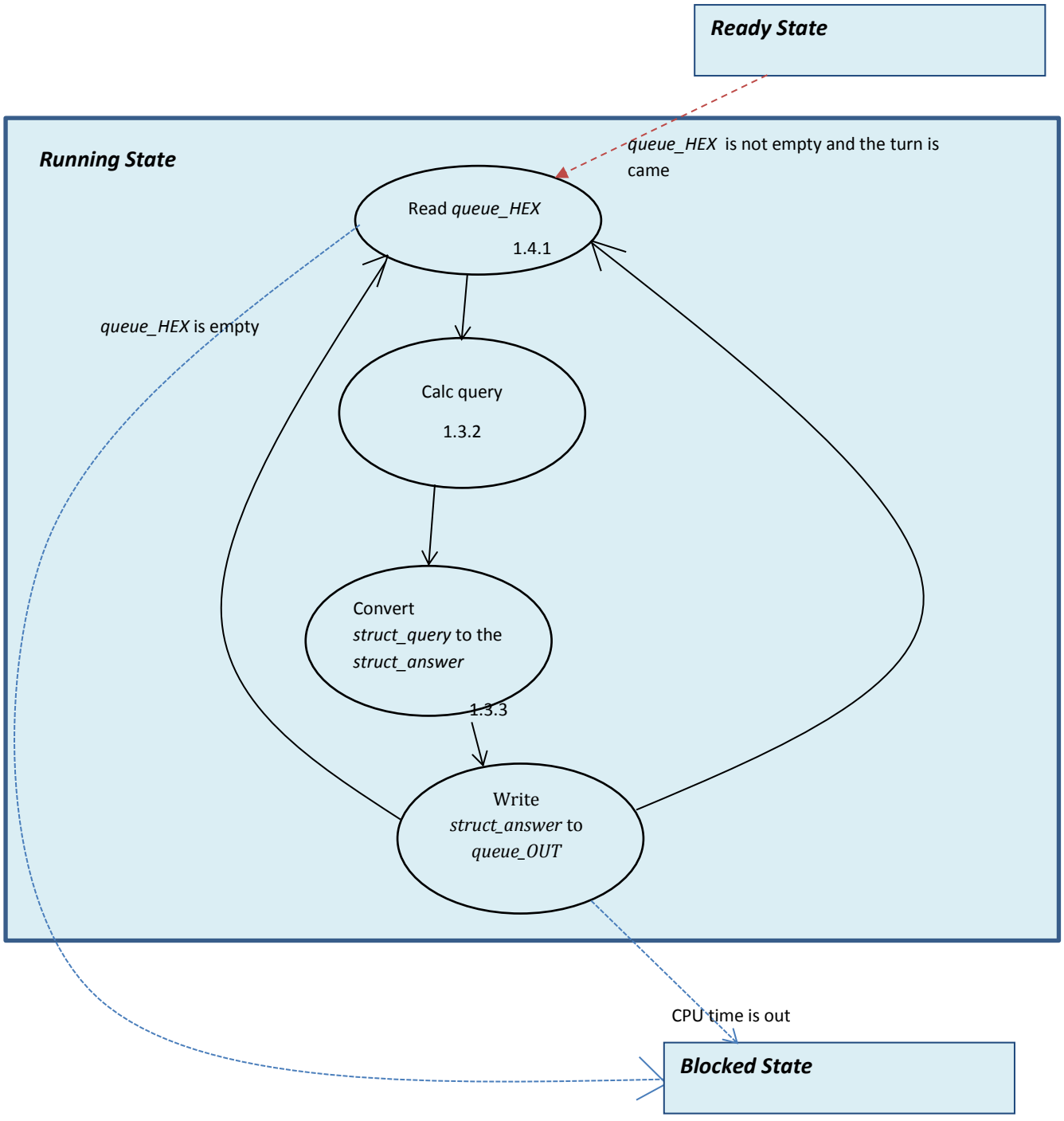
State transition diagram of Calc_HEX_task:



State Table of Calc_HEX_task

	Current State	**Event**	**Actie**	**Condition**	**Next state**	**Transition**
1	Blocked	--The message came to *queue_HEX and the turn came*;		-- *Calc_HEX_task* was waiting *queue_ HEX* and no tasks in Ready state met higher priority;	Ready	B_R_Calc_ *HEX* _task
2	Ready	Came the turn	--read from *queue_DEC;*		Run	R_R_Calc_ *HEX* _task
3	Run	-- CPU time is out --queue_ *HEX* is empty	-- stop operation; -- *Calc_ HEX _task* wait *queue_ HEX*;	-- Calc_ *HEX* blocked ;	Blocked	R_B_Calc_ *HEX* _task

Running state: Program Flow diagram: *Calc_HEX_task*

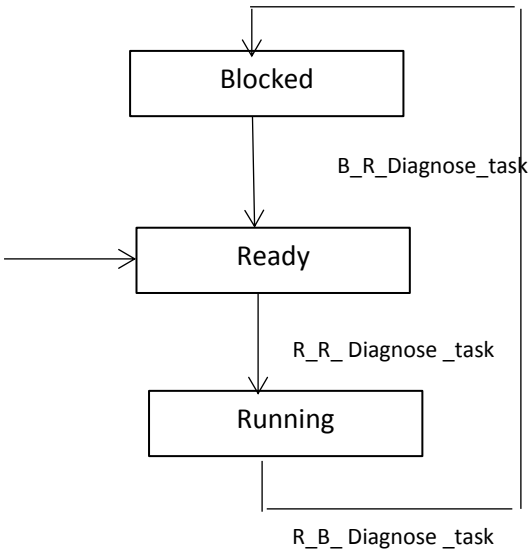


1.5 Diagnose_task

The purpose of *Diagnose_task*: show of the systems status. This task has priority 2. The task is performed every 5 seconds

Processes of *Calc_DEC_task*	Explanations
Read *the states of the exising tasks*	- Runtime in %; - Absolute runtime; - Prioriy of the task; - Current state of the task; - residue of the stack
Check the queues	- check free places in the existing queues;

State transition diagram of Diagnose_task:



State Table of the Diagnose_task

	Current State	Event	Actie	Condition	Next state	Transition
1	Blocked	5 seconds have passed since the last execution			Ready	B_R_Calc_HEX_task
2	Ready	Came the turn			Run	R_R_Calc_HEX_task
3	Run	process has finished executing			Blocked	R_B_Calc_HEX_task

Bijlage XII. Het testplan voor de demoapplicatie 2.0

- **Test 1.**

Eis 12. De implementatie van de taak die naar de seriële poort luistert, moet de interrupt simuleren

Hier voerde ik de tests uit waarop de demoapplicatie 1.0 onvoldoende scoorde: de stresstest en de test op de verschillende prioriteiten van de taken. Ik verwachtte hier niet dat de stresstest bij de nieuwe implementatie van de *UART_task* voldoende resultaat zou opleveren. Want in de nieuwe implementatie reageerde de *UART_task* op een time event, maar om het stresstestprobleem op te lossen moest deze taak op een input event reageren. Dat kon alleen bij implementatie van de seriële interrupt. De testcases staan in tabel 1.

Tabel 1: Testcases voor test 1

Testcase	***Doel***	***Opstelling***	***Verwacht resultaat***
1.1	Controleren of het Preemptive algoritme van FreeRTOS correct werkt.	Elke 500msec checkt de *Control_task* de time-out. Elke 500msec print de *Control_task* de melding: "*Huidige time*. *Control_task: check the time."*	De melding: "*Huidige time*. *Control_task: check the time."* wordt elke 500msec geprint.
1.2 Stresstest	Controleren hoe *Calculator* systeem zich gedraagt als de gebruiker de input heel snel invoert.	Het *Calculator* systeem wordt gestart. Een tekstbestand met 20 verzoeken wordt naar het systeem gestuurd.	Het systeem scoort onvoldoende. De input wordt gedeeltelijk uitgelezen.

- **Test 2.**

Eis 13. De demoapplicatie 2.0 moet gebruikmaken van diagnostische functies als Task state List en Queue toestand.

Hier controleerde ik of de diagnostische functies van de FreeRTOS juist waren geconfigureerd. Bovendien moest de *Diagnose_task* periodiek worden uitgevoerd: elke 5sec; ook dit moest ik checken. De testcases staan in tabel 2.

Tabel 2: Testcases voor test 2

Testcase	***Doel***	***Opstelling***	***Verwacht resultaat***
2.1	Controleren of de diagnosefuncties correct werken.	De demoapplicatie 2.0 wordt gestart. De output van de demoapplicatie wordt op de terminal weergegeven.	Op de terminal worden de toestanden, prioriteiten, gebruik van CPU-tijd, stack van taken en lengte van queues uitgeprint.
2.2	Controle op de periodieke uitvoering van de *Diagnose_task*. (T = 5sec)	De demoapplicatie 2.0 wordt gestart. De output van de demoapplicatie wordt op de terminal weergegeven. De output moet ook de tijd van het bericht bevatten.	Elke 5sec wordt de toestanden van het systeem uitgeprint.

- **Test 3.**

Eis 14. In de demoapplicatie 2.0 moet een softtimer geïmplementeerd worden om de variabele die tijd de houdt, uit te breiden tot type *unsigned long long.*

Hier moeten worden gecontroleerd of de softwaretimer correct is geconfigureerd en gestart en of de callback-functie van de softtimer zorgt dat de timer counter niet overloopt. De testcases staan in tabel 3.

Tabel 3: Testcases voor test 3

Testcase	***Doel***	***Opstelling***	***Verwacht resultaat***
3.1	Controleren of de timer correct is geconfigureerd.	De demoapplicatie 2.0 wordt gestart. De output bevat de aankomsttijd van het bericht. De tijd wordt twee keer opgenomen: • Eerste keer met API-functie van FreeRTOS • Tweede keer met de functie die de tijd van de softtimer geeft	De tijd van de softtimer en de tijd van de interne timer zijn gelijk.
3.2	Controleren of de callback-functie zorgt dat de timer counter niet overloopt.	De beginwaarde van de timercounter wordt gelijk aan de $(2^{32} - 6)$ geïnitialiseerd. De demoapplicatie 2.0 wordt gestart. De output van de demoapplicatie wordt op de terminal weergegeven. De output moet ook de tijd van de softtimer bevatten.	De tijd loopt gewoon door en wordt niet gereset na elke 5sec.

- **Tets 4.**

Eis 15. In de demoapplicatie 2.0 moet een set van queues worden gecreëerd om de taken te synchroniseren.

Het doel van deze test is het controleren dat de sets van de queues correct zijn gecreëerd. In de demoapplicatie 2.0 werden twee sets van queues gecreëerd, in de *Control_task* en in de *Syslog_task*. Daarvoor werd een FreeRTOS-functie gebruikt met als returnwaarden:

- een pointer naar de set handle als de sets met succes zijn gecreëerd; *Calculator* 2.0 geeft een melding dat de sets succesvol gecreëerd zijn;
- een NULL pointer als het creëren van de sets is mislukt; *Calculator* 2.0 geeft een foutmelding.

De testcases staan in tabel 4.

Tabel 4: Testcases voor test 4

Testcase	***Opstelling***	***Verwacht resultaat***
4.1	Alle sets worden met correcte paarmeters gecreëerd.	Melding van het systeem: "*Control_task*: *The set of the queues is created successfully."* "*Syslog_task*: *The set of the queues is created successfully."*
4.2	*De set van de queues in de Control_task* wordt incorrect gecreëerd.	Foutmelding: "*Control_task*: *The set is not created"*
4.3	*De set van de queues in de Syslog_task* wordt incorrect gecreëerd.	Foutmelding: "*Syslog_task*: *The set is not created"*

- **Test 5.**

Eis 16. In de demoapplicatie 2.0 moet de systeemdiagnose op een seriële poort worden geïmplementeerd, voor logberichten die bevatten:
- de ID van het proces;
- het soort event;
- de tijd.

Deze berichten kunnen met een seriële terminal (zoal Hyperterminal) worden ontvangen en gelogd.Hier moet worden gecheckt of de *Syslog_task* de configuratie van de gebruiker correct installeert. Als de gebruiker tijdens de input van het configuratieverzoek een fout maakt, moet het systeem een foutmelding geven. De testcases staan in de tabel 5.

Tabel 5: Testcases voor test 5

Testcase	***Opstelling***	***Verwacht resultaat***
5.1	*Input: SLWa* *Input: SSWa* *Hier wordt de code veranderd om te controleren dat de gebruiker verschillende levels van de output kan configureren. Hier wordt de fout op het critical level tijdelijk geprogrammeerd.*	Alle taken van het systeem printen na elk event een melding. Als er een systeemfout voorkomt, wordt die ook uitgeprint.
5.2	*Input: SLCc* *Input: SSWa*	Alle taken van het systeem printen na elk event een melding. Als er een systeemfout voorkomt, wordt die niet uitgeprint.
5.3	*Input: SLCd* *Input: SLWc* *Input: SSWa*	Als er een systeemfout voorkomt, wordt die uitgeprint. De meldingen na elk event worden niet geprint.
5.4	*Input: SLWa* *Input: SSCa*	Geen output van het systeem.
5.5	*Input: SLCa* *Input: SSWa*	Geen output van het systeem.
5.6	*Input: SLWa* *Input: SSCa* *Input: SSWu*	De meldingen van de *UART_task* worden geprint op Debug- en Critical-niveau.
5.7	*Input: SSWc*	De meldingen van de *UART_task* en *Control_task* worden geprint op Debug- en Critical-niveau
5.8	*Input: SSWi*	De meldingen van de *UART_task*, *Control_task* en *Interpreter_task* worden geprint op Debug- en Critical-niveau
5.9	*Input: SSCi* *Input: SSCu* *Input: SSCc* *Input: SSWh*	De meldingen van de *Calc_HEX_task* worden geprint op Debug- en Critical-niveau.
5.10	*Input: SSCh* *Input: SSWd*	De meldingen van de *Calc_DEC-task* worden geprint op Debug- en Critical-niveau
5.11	*Input: SSCd* *Input: SSWt*	De meldingen van de *Diagnose_task* worden geprint op Debug- en Critical-niveau.
5.12	*Input: SSCt*	Geen output van het systeem.
5.13	*Input: SSWa* *Input: sSCa*	Foutmelding over incorrecte input.
5.14	*Input: SsCa*	Foutmelding over incorrecte input.
5.15	*Input: SlCa*	Foutmelding over incorrecte input.
5.16	*Input: SLca*	Foutmelding over incorrecte input.
5.17	*Input:SLwa*	Foutmelding over incorrecte input.
5.18	*Input:SSwa*	Foutmelding over incorrecte input.
5.19	*Input:SSca*	Foutmelding over incorrecte input.
5.20	*Input:SSWA*	Foutmelding over incorrecte input.
5.21	*Input:SSCU*	Foutmelding over incorrecte input.
5.22	*Input:SLCD*	Foutmelding over incorrecte input.
5.23	*Input:SLWC*	Foutmelding over incorrecte input.

Bijlage XIII. De beschrijving van de queues in de demoapplicatie 2.0

Naam van queue	**Type van items**	***Lengte***	***Beschrijving***
xQueue_UART	*char*	*20*	*Sender: UART_task* *Receiver: Control_task* In de huidige implementatie van de demoapplicatie is het genoeg om de lengte van deze queue gelijk aan één te maken, want de *Receiver* heeft een hogere prioriteit dan de *Sender* en als een item binnen de queue komt , schakelt de scheduler meteen over naar de *Receiver* taak. Maar in de toekomst wordt deze taak door de interrupt vervangen, die heeft een hogere prioriteit dan de andere taken. Daarom werd de lengte van de queue gelijk gemaakt aan 20.
xQueue_OUT	*STRUCT_ANSWER*	*1*	*Sender: Interpreter_task;* *Calc_DEC_task;* *Calc_HEX_task;* *Receiver: Control_task* De *Senders* hebben dezelfde prioriteit. Deze prioriteit is lager dan de prioriteit van *Receiver*. Als een item in de queue komt dan schakelt scheduller meteen *over* naar de *Receiver*, daarom is de lengte van één voor deze queue genoeg.
xQueue_SYSLOG	*STRUCT_QUERY*	*10*	*Sender: Control_task* *Receiver: Syslog_task* De *Sender* heeft een hogere prioriteit dan de *Receiver*. De *Sender* kan sneller items versturen dan de *Receiver* hen ontvangt. De queue moet groot genoeg zijn om meerdere items te bewaren.
xQueue_IN	*STRUCT_QUERY*	*10*	*Sender: Control_task* *Receiver: Interpreter_task* De *Sender* heeft een hogere prioriteit dan de *Receiver*. De *Sender* kan sneller items versturen dan de *Receiver* hen ontvangt. De queue moet groot genoeg zijn om meerdere items te bewaren.
xQueue_DEC	*QUERY_TO_CALCULATE*	*5*	*Sender: Interpreter_task* *Receiver: Calc_DEC_task* De *Sender* heeft dezelfde prioriteit als de *Receiver*. De *Sender* kan meer dan één item per slice time versturen. De queue moet groot genoeg zijn om meerdere items te bewaren.
xQueue_HEX	*QUERY_TO_CALCULATE*	*5*	*Sender: Interpreter_task* *Receiver: Calc_HEX_task* De *Sender* heeft dezelfde prioriteit als de *Receiver*. De *Sender* kan meer dan één item per slice time versturen. De queue moet groot genoeg zijn om meerdere items te bewaren.
xQueue_PRINT	*string*	*40*	*Sender: Interpreter_task; Control_task; UART_task; Calc_DEC_task; Calc_HEX_task; Diagnose_task* *Receiver: Syslog_task;* De *Syslog_task* heeft de laagste prioriteit. De queue moet genoeg ruimte hebben om meerdere items te bewaren.

Bijlage XIV. De definities van de structuren in de demoapplicatie 2.0

```
typedef struct
{
	uint16_t ID; //identification number;
	uint8_t data_length; //the length of data;
	char input_id;//address of input and the radix of output numbers:

} PACKET_HEADER;


typedef struct
{
	PACKET_HEADER header; //information about the query
	char string_answer[60];//the output answer
}STRUCT_ANSWER;

typedef struct
{
	PACKET_HEADER header; //information about the query
	char string_query[24];//the output answer
} STRUCT_QUERY;

typedef struct
{
	PACKET_HEADER header; //information about the query
	int32_t opr_1;//first number
	int32_t opr_2; //second number
	char com;//command
	char code_error;//error of input
	char input_radix; //radix of input numbers
}QUERY_TO_CALCULATE;
```

Bijlage XV. De bestandenbeschrijving van de demoapplicatie 2.0

Bestanden	Beschrijven
FreeRTOS-main.c	main-bestaand. Dit bestand bevat de void *main()*, waarin alle taken en queues gecreëerd en gecheckt worden.
uart.c uart.h	Deze bestanden bevatten de *UART_task*. In het uart.h staat declaratie, in het uart.c staat de definitie van deze taak.
control.h control.c	De *Control_task* is in deze bestanden gedeclareerd en gedefinieerd.
syslog.h syslog.c	In deze bestanden is de *Syslog_task* gedeclareerd en gedefinieerd. Hier waren de functies *set_syslog()* en *syslog()* gerealiseerd. De functie *set_syslog()* configureert de output. De functie *syslog()* maakt een string vanuit query naar output en stuurt deze string naar de *queue_PRINT*.
interpreter.h interpreter.c	In deze bestanden is de *Interpreter_task* gedeclareerd en gedefinieerd. Hier was de functie *check_string_to_calculate()* gerealiseerd. Deze functie checkt de structuur *struct_query* en converteert het naar structuur *query_to_calculate.*
calculate.h calculate.c	In deze bestanden zijn de *Calc_DEC_task* en Calc_HEX_task gedeclareerd en gedefinieerd. Hier waren de functies *my_utoa ()* en *my_itoa()* gerealiseerd. De functie *my_utoa ()* converteert het integer decimaal getal naar de string. De functie *my_itoa()* converteert het decimaal getal naar een hexadecimaal en naar de string.
diagnose.h diagnose.c	In deze bestanden is de *Diagnose_task* gedeclareerd en gedefinieerd.
timer.h timer.c	In deze bestanden was de softwaretimer gerealiseerd.
Includes.h	In dit bestand staan include alle noodzakelijke bestanden voor de demoapplicatie.
Define.h	In dit bestand zijn alle constanten, structuren en globale variabelen gedefinieerd.

Bijlage XVI. De resultaten van het testen van de demoapplicatie 2.0

- **Test 1.**

Eis 12. Het implementatie van de taak die de seriële port luistert, moet de interrupt simuleren.

Hier worden de testen uitgevoerd die waren niet voldoende tijdens het testen van de eerste versie van de demoapplicatie. Het is de stresstest en test van de verschillende prioriteiten van de taken. Hier is niet verwacht dat de stresstest de voldoende resultaat zal hebben bij de nieuwe implementatie van de UART taak. Want in de nieuwe implementatie regeert UART taak op de time event, maar om de stresstest probleem op te lossen moet deze taak op de input event reageren. Dat kan alleen bij de implementatie van de seriële interrupt.

Testcase	**Doel**	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
1.1	Het controleren dat de preemptive algoritme van de FreeRTOS werkt correct. De Control taak heeft de hogere prioriteit dan de andere taken (behalve Diagnose taak)	Minimum elk 0.5sec checkt de *Control_task* time-out. Elk 0.5sec de *Control_task* print de melding: "*Huidige time*. *Control_task: check the time."*	De melding: "*Control_task: check the time."* moet tenminste elke 0.5 sec wordt geprint. Alle andere taken moeten worden afgebroken als de Diagnose taak uit is.	*Correct* Printscreen 1.1
1.2 Stresstest 1	Het doel van deze test is het controleren hoe draagt zich het *Calculator* systeem als de input van de gebruiker heel snel is ingevoerd	Het Calculator systeem wordt opgestart. Het tekst bestand met 20 verzoeken wordt naar het systeem gestuurd.	Het is verwacht dat het systeem niet voldoende resultaat zal hebben. De input wordt gedeeltelijk uitgelezen.	*Correct* Printscreen 1.2
1.3 Stresstest 2	Het doel van deze test is het controleren hoe draagt zich het Calculator systeem als de input van de gebruiker heel snel is ingevoerd	Het Calculator systeem wordt opgestart. Het tekst bestand met 100 verzoeken wordt naar het systeem gestuurd. Maar op de Terminal wordt geïnstalleerd dat elke char met 1msec vertraging binnen komt.	Het is verwacht dat het systeem alle verzoekingen berekent.	*Correct* Printscreen 1.3

Testcase 1.1. Opmerkingen:

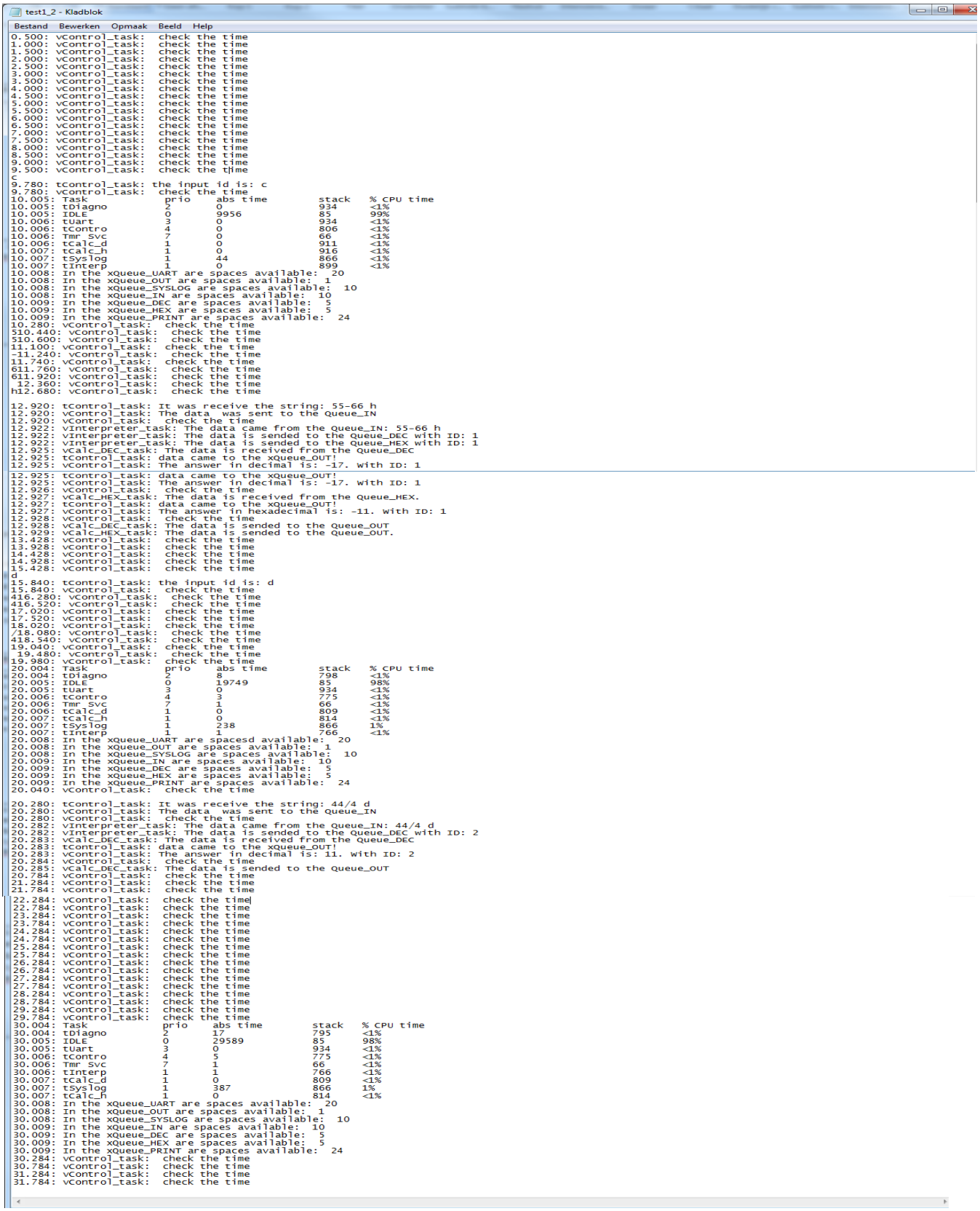
In de demoapplicatie wordt de timeout gecontroleerd in de volgende gevallen:

- Elke 500 msec
- Elke keer als het bericht komt in de *queue_UART* of *queue_OUT*;

Dus de verwachtte resultaat is dat de timeout checking wordt tenminste één keer per 0.5 secunde wordt uitgevoerd. De resultaten van deze test is correct. De output van deze test staat op

Printscreen 1.1

Printscreen 1.1: De output van Test 1 – Test-cases 1.1

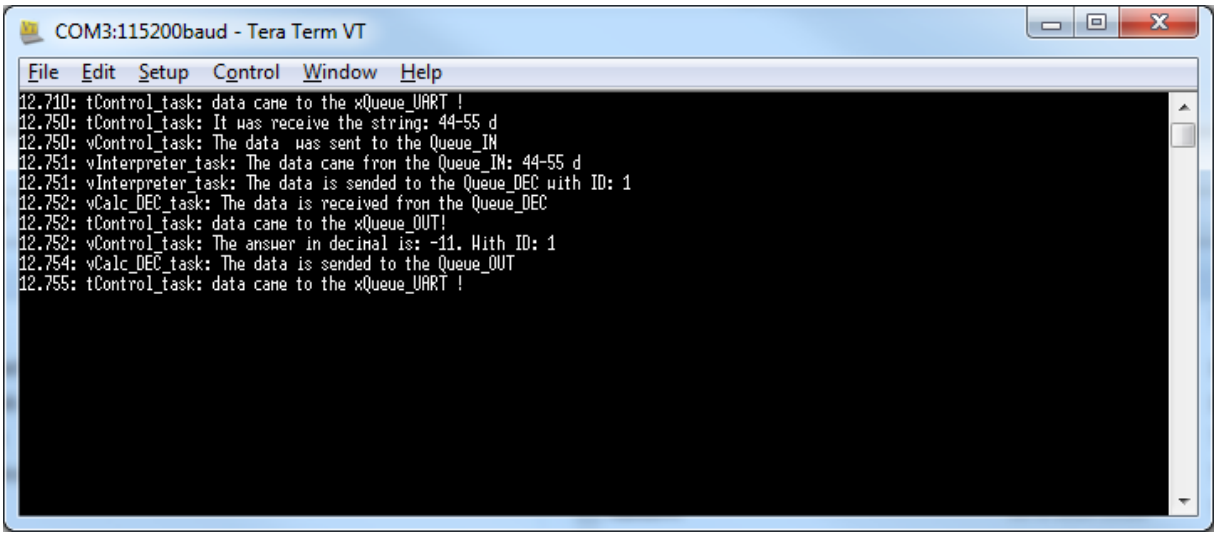


```
test1_2 - Kladblok
Bestand  Bewerken  Opmaak  Beeld  Help
0.500: vControl_task:  check the time
1.000: vControl_task:  check the time
1.500: vControl_task:  check the time
2.000: vControl_task:  check the time
2.500: vControl_task:  check the time
3.000: vControl_task:  check the time
3.500: vControl_task:  check the time
4.000: vControl_task:  check the time
4.500: vControl_task:  check the time
5.000: vControl_task:  check the time
5.500: vControl_task:  check the time
6.000: vControl_task:  check the time
6.500: vControl_task:  check the time
7.000: vControl_task:  check the time
7.500: vControl_task:  check the time
8.000: vControl_task:  check the time
8.500: vControl_task:  check the time
9.000: vControl_task:  check the time
9.500: vControl_task:  check the time
c
9.780: tControl_task: the input id is: c
9.780: vControl_task:  check the time
10.005: Task            prio    abs time        stack   % CPU time
10.005: tDiagno         2       0               934     <1%
10.005: IDLE            0       9956            85      99%
10.006: tUart           3       0               934     <1%
10.006: tContro         4       0               806     <1%
10.006: Tmr Svc         7       0               66      <1%
10.006: tCalc_d         1       0               911     <1%
10.007: tCalc_h         1       0               916     <1%
10.007: tSyslog         1       44              866     <1%
10.007: tInterp         1       0               899     <1%
10.008: In the xQueue_UART are spaces available:  20
10.008: In the xQueue_OUT are spaces available:  1
10.008: In the xQueue_SYSLOG are spaces available:  10
10.008: In the xQueue_IN are spaces available:  10
10.009: In the xQueue_DEC are spaces available:  5
10.009: In the xQueue_HEX are spaces available:  5
10.009: In the xQueue_PRINT are spaces available:  24
10.280: vControl_task:  check the time
510.440: vControl_task:  check the time
510.600: vControl_task:  check the time
11.100: vControl_task:  check the time
-11.240: vControl_task:  check the time
11.740: vControl_task:  check the time
611.760: vControl_task:  check the time
611.920: vControl_task:  check the time
 12.360: vControl_task:  check the time
h12.680: vControl_task:  check the time

12.920: tControl_task: It was receive the string: 55-66 h
12.920: vControl_task: The data  was sent to the Queue_IN
12.920: vControl_task:  check the time
12.922: vInterpreter_task: The data came from the Queue_IN: 55-66 h
12.922: vInterpreter_task: The data is sended to the Queue_DEC with ID: 1
12.922: vInterpreter_task: The data is sended to the Queue_HEX with ID: 1
12.925: vCalc_DEC_task: The data is received from the Queue_DEC
12.925: tControl_task: data came to the xQueue_OUT!
12.925: vControl_task: The answer in decimal is: -17. With ID: 1
12.925: tControl_task: data came to the xQueue_OUT!
12.925: vControl_task: The answer in decimal is: -17. With ID: 1
12.926: vControl_task:  check the time
12.927: vCalc_HEX_task: The data is received from the Queue_HEX.
12.927: tControl_task: data came to the xQueue_OUT!
12.927: vControl_task: The answer in hexadecimal is: -11. With ID: 1
12.928: vControl_task:  check the time
12.928: vCalc_DEC_task: The data is sended to the Queue_OUT
12.929: vCalc_HEX_task: The data is sended to the Queue_OUT.
13.428: vControl_task:  check the time
13.928: vControl_task:  check the time
14.428: vControl_task:  check the time
14.928: vControl_task:  check the time
15.428: vControl_task:  check the time
d
15.840: tControl_task: the input id is: d
15.840: vControl_task:  check the time
416.280: vControl_task:  check the time
416.520: vControl_task:  check the time
17.020: vControl_task:  check the time
17.520: vControl_task:  check the time
18.020: vControl_task:  check the time
/18.080: vControl_task:  check the time
418.540: vControl_task:  check the time
19.040: vControl_task:  check the time
 19.480: vControl_task:  check the time
19.980: vControl_task:  check the time
20.004: Task            prio    abs time        stack   % CPU time
20.004: tDiagno         2       8               798     <1%
20.005: IDLE            0       19749           85      98%
20.005: tUart           3       0               934     <1%
20.006: tContro         4       3               775     <1%
20.006: Tmr Svc         7       1               66      <1%
20.006: tCalc_d         1       0               809     <1%
20.007: tCalc_h         1       0               814     <1%
20.007: tSyslog         1       238             866     1%
20.007: tInterp         1       1               766     <1%
20.008: In the xQueue_UART are spacesd available:  20
20.008: In the xQueue_OUT are spaces available:  1
20.008: In the xQueue_SYSLOG are spaces available:  10
20.009: In the xQueue_IN are spaces available:  10
20.009: In the xQueue_DEC are spaces available:  5
20.009: In the xQueue_HEX are spaces available:  5
20.009: In the xQueue_PRINT are spaces available:  24
20.040: vControl_task:  check the time

20.280: tControl_task: It was receive the string: 44/4 d
20.280: vControl_task: The data  was sent to the Queue_IN
20.280: vControl_task:  check the time
20.282: vInterpreter_task: The data came from the Queue_IN: 44/4 d
20.282: vInterpreter_task: The data is sended to the Queue_DEC with ID: 2
20.283: vCalc_DEC_task: The data is received from the Queue_DEC
20.283: tControl_task: data came to the xQueue_OUT!
20.283: vControl_task: The answer in decimal is: 11. With ID: 2
20.284: vControl_task:  check the time
20.285: vCalc_DEC_task: The data is sended to the Queue_OUT
20.784: vControl_task:  check the time
21.284: vControl_task:  check the time
21.784: vControl_task:  check the time
22.284: vControl_task:  check the time
22.784: vControl_task:  check the time
23.284: vControl_task:  check the time
23.784: vControl_task:  check the time
24.284: vControl_task:  check the time
24.784: vControl_task:  check the time
25.284: vControl_task:  check the time
25.784: vControl_task:  check the time
26.284: vControl_task:  check the time
26.784: vControl_task:  check the time
27.284: vControl_task:  check the time
27.784: vControl_task:  check the time
28.284: vControl_task:  check the time
28.784: vControl_task:  check the time
29.284: vControl_task:  check the time
29.784: vControl_task:  check the time
30.004: Task            prio    abs time        stack   % CPU time
30.004: tDiagno         2       17              795     <1%
30.005: IDLE            0       29589           85      98%
30.005: tUart           3       0               934     <1%
30.006: tContro         4       5               775     <1%
30.006: Tmr Svc         7       1               66      <1%
30.006: tInterp         1       1               766     <1%
30.007: tCalc_d         1       0               809     <1%
30.007: tSyslog         1       387             866     1%
30.007: tCalc_h         1       0               814     <1%
30.008: In the xQueue_UART are spaces available:  20
30.008: In the xQueue_OUT are spaces available:  1
30.008: In the xQueue_SYSLOG are spaces available:  10
30.009: In the xQueue_IN are spaces available:  10
30.009: In the xQueue_DEC are spaces available:  5
30.009: In the xQueue_HEX are spaces available:  5
30.009: In the xQueue_PRINT are spaces available:  24
30.284: vControl_task:  check the time
30.784: vControl_task:  check the time
31.284: vControl_task:  check the time
31.784: vControl_task:  check the time
```

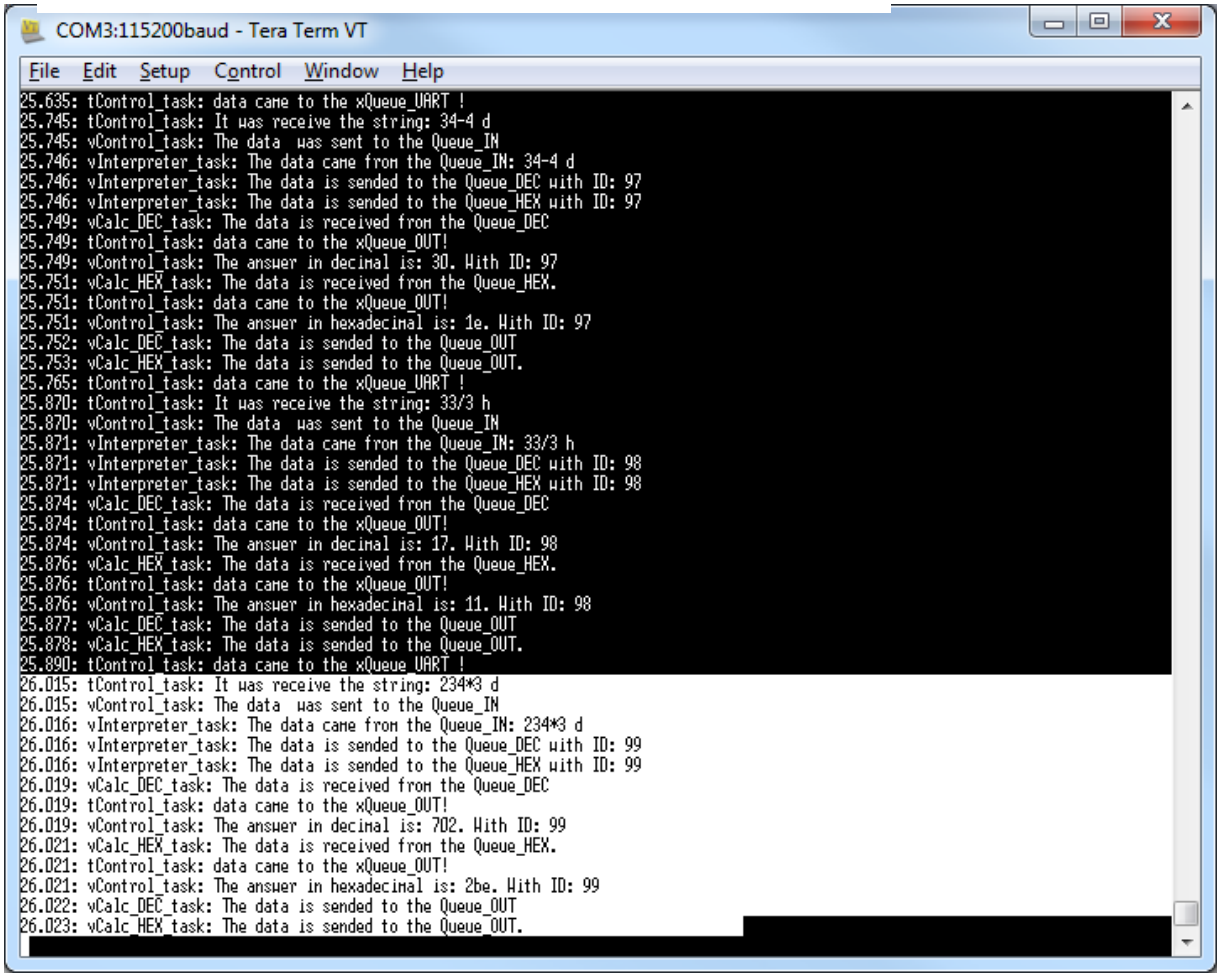
Test-cases 1.2 en 1.3. Opmerkingen:

Het was niet verwacht dat de resultaten van deze test voldoende zullen zijn. Want de UART taak die leest de seriele port, reageert op de tijdevent maar niet op de inputevent. Om deze veronderstelling te controleren was de testcase 1.2 met normale instellingen van Terminal uitgevoerd en de testcase 1.3 was bij de Terminal geinstaleerd de inputvertraging – 1msec elke char. De resultatenvan de eerste test waren niet voldoende, van de tweede waren alle verzoeken berekend. De output van deze testen staan op Printscreen 1.2 en 1.3. Uit de output van de eerste test kan worden laten gezien dat de demoapplicatie misde het eind van de input – 'Enter', doordat het verzoek was niet verder uitgewerkt. Bij de tweede test werden 99 verzoeken gestuurd en bij Printscreen 1.3 staat de output van de laatste deel van de uitvoering van het verzoek. Het uitvoering was correct, alle verzoekingen waren berekend. Dus de resultaat van de testcase 1.3 is voldoende.

Printscreen 1.2: De output van Test 1 – Testcases 1.2



Printscreen 1.3: De output van Test 1 – Testcases 1.3



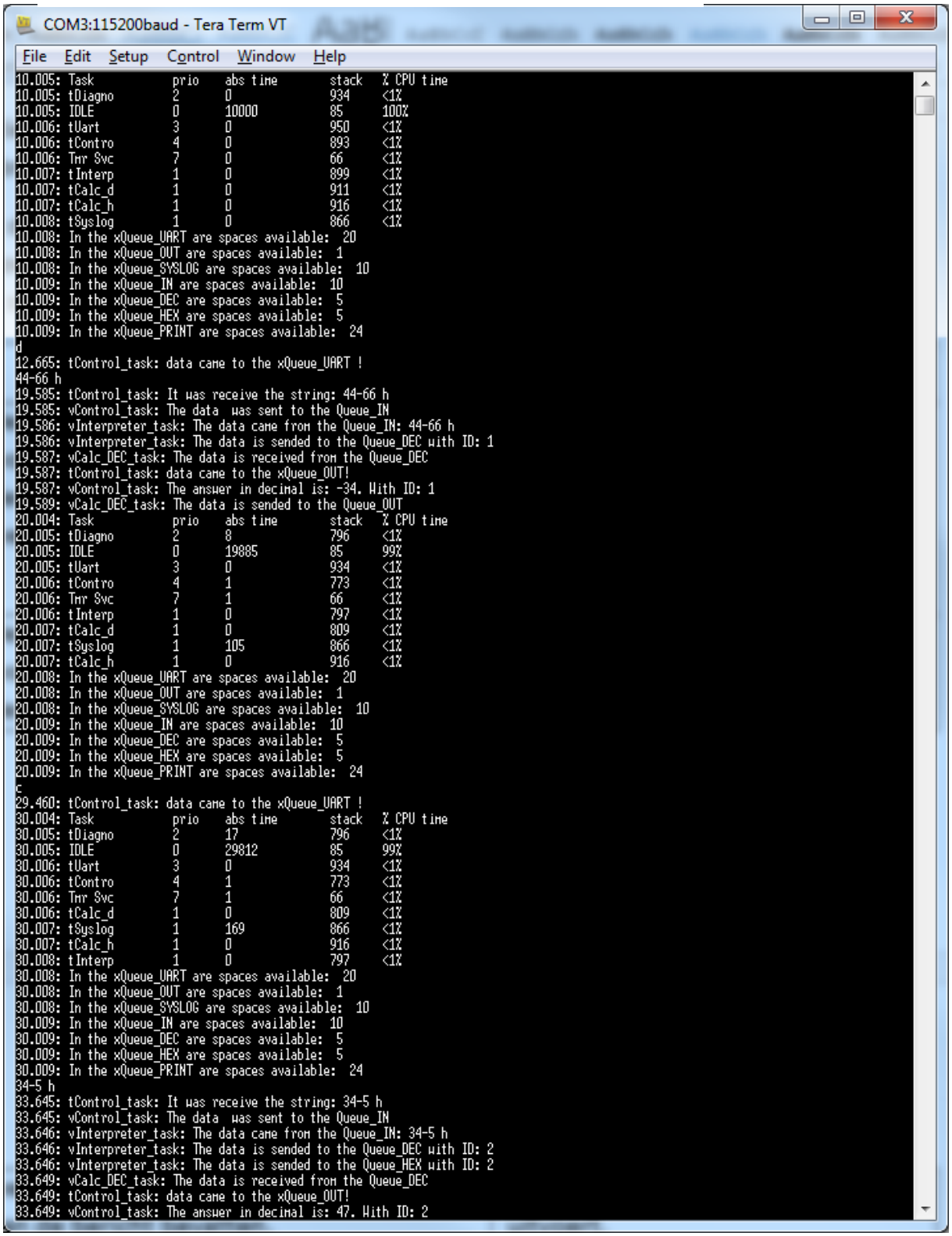
- **Test 2.**

Eis 13. De demoapplicatie 2.0, moet de diagnostische functies zoals Task state List en Queue status.

Hier worden gecontroleerd dat de diagnostische functies van de FreeRTOS juist zijn geconfigureerd. Bovendien de *Diagnose_task* moet periodiek uitvoeren.Tijdens het test was periode gelijkaan 10 secunde gedefineerd, deze stelling moet ook gecheckt worden.

Testcase	**Doel**	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
2.1	Het controleren dat de diagnose functies werken correct in de demoapplicatie – versie 2.	De tweede versie van de demoapplicatie worden opgestart. De output van de demoapplicatie wordt op de terminal weergegeven.	Er worden op de terminal de staten, prioriteiten, het gebruikt van CPU tijd en stack van de taken en hoe volle queues zijn uitgeprint.	*Correct* Printscreen 2
2.2	De controle van de periodieke uitvoering van de Diagnose taak. (T = 10 sec)	De tweede versie van de demoapplicatie worden opgestart. De output van de demoapplicatie wordt op de terminal weergegeven. De output moet ook de tijd van de bericht bevatten.	Het is verwacht dat het elke 10 secunde worden de stae van het systeem uitgeprint, wat betekend dat de Diagnose taak periodiek uitvoert.	*Correct* Printscreen 2

Printscreen 2 De output van Test 2 – Test-cases 2.1 en 2.2



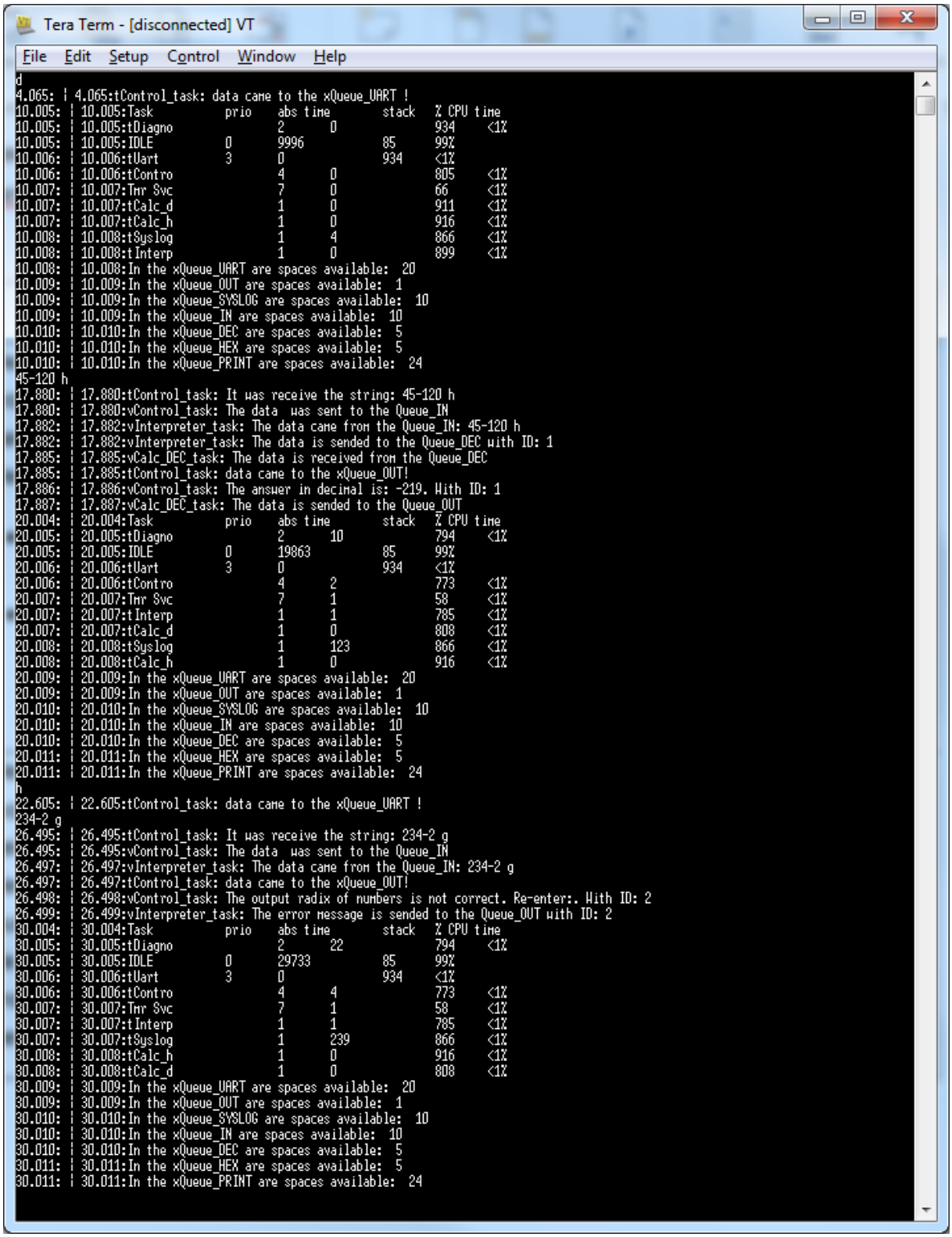
Test 3.

Eis 14. In de demoapplicatie 2.0, moet de softtimer geïmplementeerd worden om de variabele die tijd houdt, tot *unsigned long long* type uitgebreid.

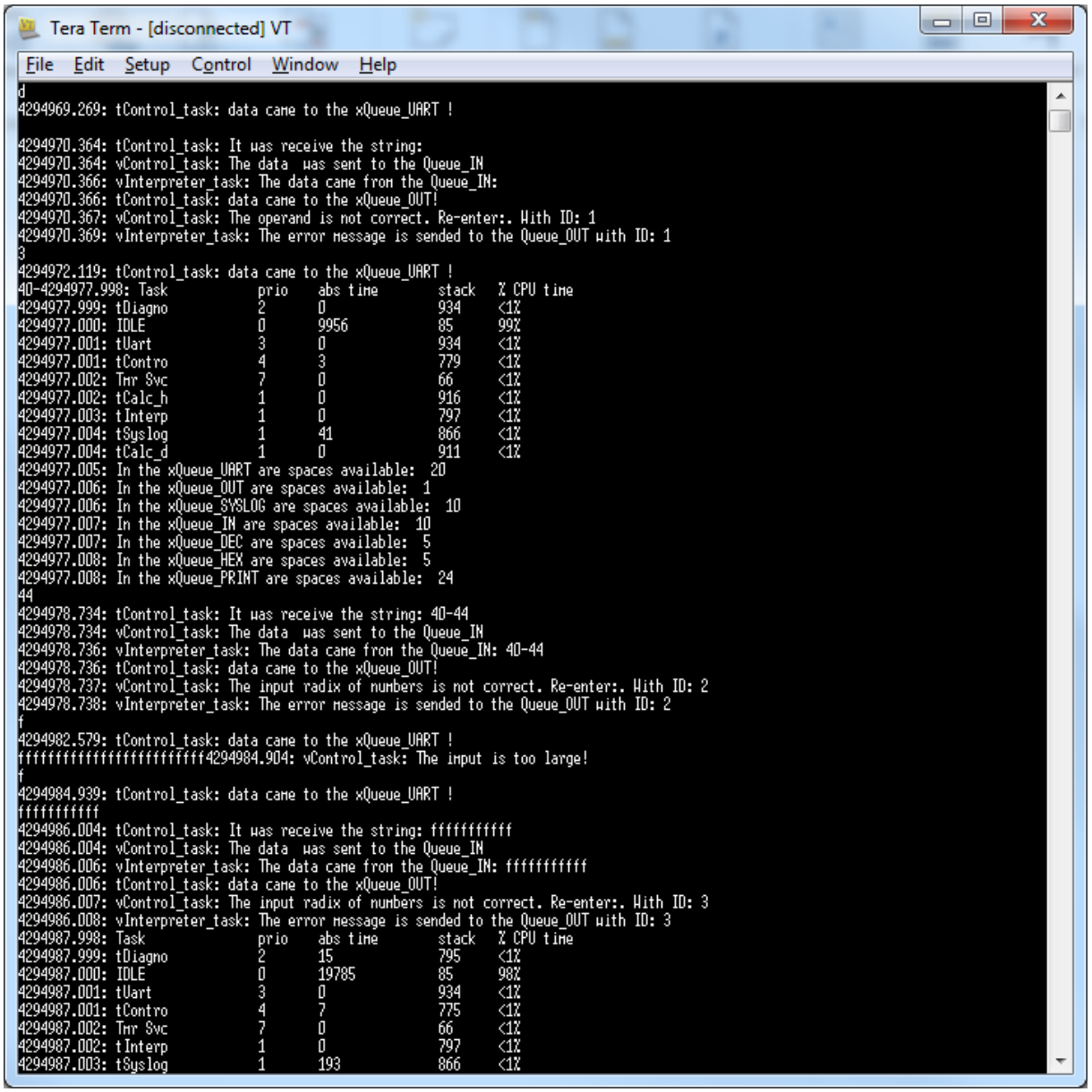
Hier moeten worden gecontroleerd dat de software timer correct geconfigureerd, gestaart en de callback functie van de softtimer zorgt voor dat de timer counter niet overflow gaat.

Testcase	**Doel**	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
3.1	Het controleren dat de timer correct is geconfigureerd.	De tweede versie van de demoapplicatie worden opgestart. De output moet de aankomsttijd van het bericht bevatten. De tijd wordt twee keer aangenomen: • Eerste keer met met de functie die geeft de tijd van de softtimer: *GetSystemTicks();* • Tweede keer met API functie van de FreeRTOS: *xTaskGetTickCount();*	De tijd van de softtimer en tijd interne timer moet gelijk zijn.	*Correct* Printscreen 3.1
3.2	De controle dat e callback functie zorgt voor dat de timer counter niet overflow gaat.	De beginwaarde van de timercounter wordt gelijk aan de $(2^{32} - 6)$ geinitialiseerd. De tweede versie van de demoapplicatie worden opgestart. De output van de demoapplicatie wordt op de terminal weergegeven. De output moet ook de tijd van de softtimer bevatten.	Het is verwacht dat de tijd loopt gewoon door en wordt niet gereset over 5 seconde.	*Correct* Printscreen 3.2

Printscreen 3.1: De output van Test 3 – Testcase 3.1



Printscreen 3.2: De output van Test 3 – Testcase 3.2



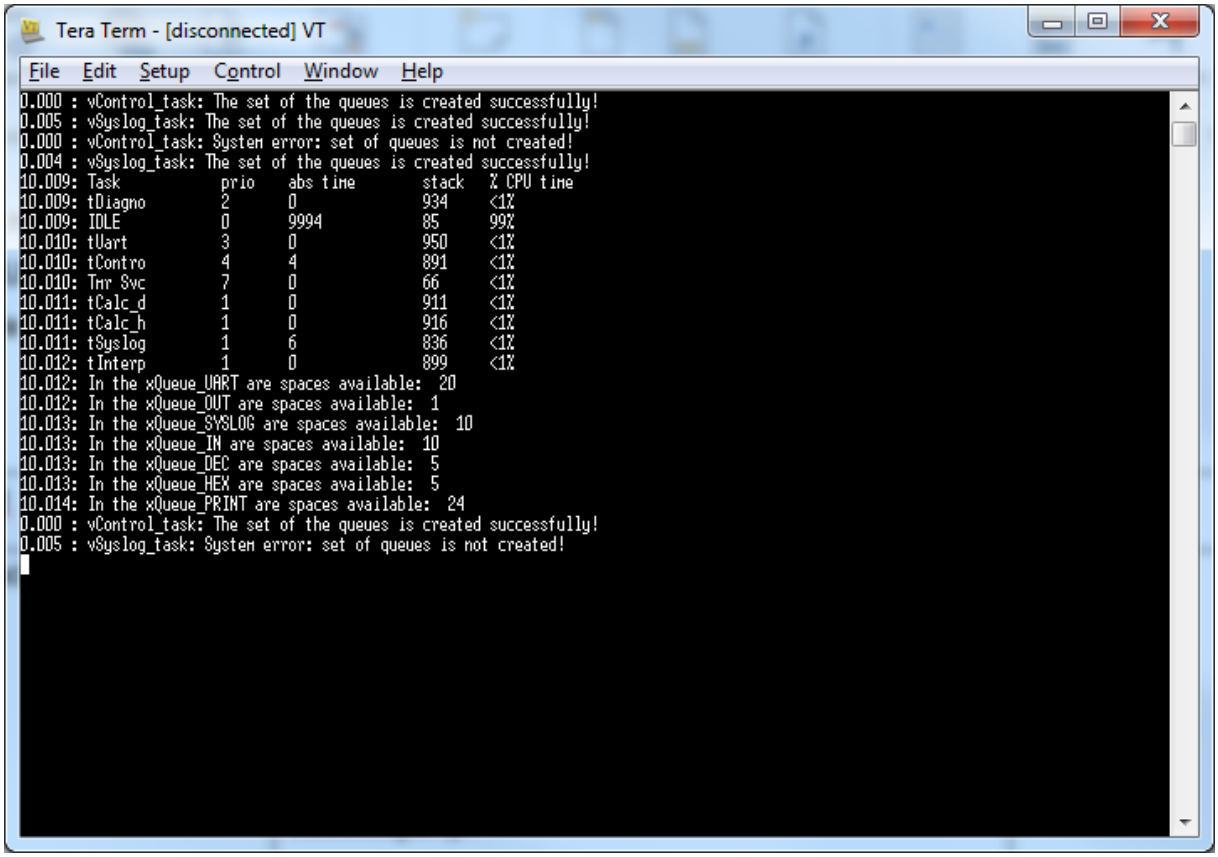
- **Tets 4.**

Eis 15. In de demoapplicatie - versie 2, moet de set van de queues worden gecreëerd om de taken te synchroniseren.

In de demoapplicatie 2.0 werden twee sets van de queues gecreëerd in de *Control_task* en in de *Syslog_task*. Daarvoor werd de FreeRTOS functie gebruikt. Deze functie heeft als een return waarde een pointer naar de set handle. Als het creëren is mislukt is de return waarde NULL pointer. Na het creëren van de sets worden de return waarde gecheckt. Als het NULL is, geeft het Calculator 2.0 een foutmelding, als het niet NULL is dan geeft het Calculator 2.0 een melding dat de sets succesvol gecreëerd werden.

Test-Case	*Opstelling*	*Verwachtte resultaat*	*Resultaat*
4.1	Alle sets worden met correcte paarmeters gecreëerd.	Melding van het systeem: "*Control taak*: *The set of the queues is created successfully.*" "*Syslog taak*: *The set of the queues is created successfully.*"	*Correct* Printscreen 4
4.2	*De set van de queues in de Control_task* wordt incorrect gecreëerd.	Foutmelding: "*Control taak*: *The set is not created*"	*Correct* Printscreen 4
4.3	*De set van de queues in de Syslog_task*wordt incorrect gecreëerd.	Foutmelding: "*Syslog taak*: *The set is not created*"	*Correct* Printscreen 4

Printscreen 4: De output van Test 4 – Test-cases 4.1, 4.2 en 4.3



- **Test 5.**

Eis 16. In de tweede versie van de demoapplicatie moet de systeem diagnose op een seriële port worden geïmplementeerd. Dat is een soort van de log bericht waarin staat wat, waar en wanneer is gebeurd tijdens de uitvoering van het programma. In het bericht moet staan:

- De ID van proces
- Soort van event
- Tijd

Deze berichten kunnen met een seriële terminal (zoal Hyperterminal) worden ontvangen en gelogd.

Hier moet worden gecheckt dat de *Syslog_task* correct de configuratie van de gebruiker installeert. Als tijdens de input van de configuratie verzoek de gebruiker een fout gemaakt, het systeem moet foutmelding geven.

***Test*case**	***Opstelling***	***Verwachtte resultaat***	***Resultaat***
5.1	*Input: SLWa* *Input: SSWa* *Hier wordt de code veranderd om te controleren dat de gebruiker kan verschillende levels van de output configureren. Hier wordt de fout op de critical level tijdelijk geprogrammeerd.*	Alle taken van het systeem moeten na elke event een melding printen. Als er een systeem fout komt, ze moet ook uitgeprint worden.	*Correct*
5.2	*Input: SLCc* *Input: SSWa*	Alle taken van het systeem moeten na elke event een melding printen. Als er een systeem fout komt, ze moet niet uitgeprint worden.	*Correct*
5.3	*Input: SLCd* *Input: SLWc* *Input: SSWa*	Als er een systeem fout komt, ze moet uitgeprint worden. De meldingen na elke event worden niet geprint.	*Correct*
5.4	*Input: SLWa* *Input: SSCa*	Geen output van het systeem	*Correct*
5.5	*Input: SLCa* *Input: SSWa*	Geen output van het systeem	*Correct*
5.6	*Input: SLWa* *Input: SSCa* *Input: SSWu*	De meldingen van de UART taak moeten worden geprint op de Debug en Critical niveau	*Correct*
5.7	*Input: SSWc*	De meldingen van de UART en Control taak moeten worden geprint op de Debug en Critical niveau	*Correct*
5.8	*Input: SSWi*	De meldingen van de UART, Control en Interpreter taken moeten worden geprint op de Debug en Critical niveau	*Correct*
5.9	*Input: SSCi* *Input: SSCu* *Input: SSCc* *Input: SSWh*	De meldingen van de Calculate _HEX taak moeten worden geprint op de Debug en Critical niveau	*Correct*
5.10	*Input: SSCh* *Input: SSWd*	De meldingen van de Calculate _DEC taak moeten worden geprint op de Debug en Critical niveau	*Correct*
5.11	*Input: SSCd* *Input: SSWt*	De meldingen van de Diagnose taak moeten worden geprint op de Debug en Critical niveau	*Correct*
5.12	*Input: SSCt*	Geen output van het systeem	*Correct*
5.13	*Input: SSWa* *Input: sSCa*	foutmelding over incorrecte input	*Correct*
5.14	*Input: SsCa*	foutmelding over incorrecte input	*Correct*
5.15	*Input: SlCa*	foutmelding over incorrecte input	*Correct*
5.16	*Input: SLca*	foutmelding over incorrecte input	*Correct*
5.17	*Input:SLwa*	foutmelding over incorrecte input	*Correct*
5.18	*Input:SSwa*	foutmelding over incorrecte input	*Correct*
5.19	*Input:SSca*	foutmelding over incorrecte input	*Correct*
5.20	*Input:SSWA*	foutmelding over incorrecte input	*Correct*
5.21	*Input:SSCU*	foutmelding over incorrecte input	*Correct*
5.22	*Input:SLCD*	foutmelding over incorrecte input	*Correct*
5.23	*Input:SLWC*	foutmelding over incorrecte input	*Correct*

De output van deze testen staan in de bestanden: *test_Syslog_conf.txt* en *test_Syslog_fout_conf. Txt*

Bijlage XVII. Integratie van MicroC/OS-II naar Xilinx SDK 14

MicroC/OS-II/ Xilinx SDK 14.7 Integration

1. Download *µC/OS-II (Kernel)*
   from http*://micrium.com/downloadcenter/*

2. Make the link from the `Micrium_Microblaze_uCOS-II-AXI` to the `Xilinx workspace:`

   - From within the SDK, click *Xilinx Tools > Repositories* to open the *Repositories* tab in the Preferences dialog box
   - To add a local or global repository, click *New*. Browse to and select the directory `Micrium_Microblaze_uCOS-II-AXI\lib\lib`
   - Click *Rescan Repositiries*
   - Click *Apply* and *Ok*

3. Create a uCOS-II BSP from within the SDK, click *File > New > Xilinx Board Support Package* to open the New Board Support Package Project dialog. If a hardware project has not already been defined, you will be prompted to select one before the New Board Support Package Project dialog opens. Select " uCOS-II" as the Board Support Package OS, before clicking *Finish*. The Board Support Package Settings dialog will open, in which the MicroC/OS-II parameters can be adjusted as required.

4. Once the MicroC/OS-II BSP has been created, it can be referenced by other SDK projects: *Properties > Project References >* uCOS-II *BSP*.

Bijlage XVIII. Het ontwerpen van de wrapper functies.

Eis 1. De wrapper moet de functie bevatten die scheduler start.

Wrapper functie	*pr_StartScheduler()*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vTaskStartScheduler()*	*OSStart()*
Inputparameters	geen	geen
Returnwaarden	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei functies hebben geen parameters en geen returnwaarden. Er zijn geen problemen voor de implementatie van deze functies.

Eis 2. De wrapper moet de functie bevatten die een taak creëert.

Wrapper functie	*pr_TaskCreate(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xTaskGenericCreate(..)*	*OSTaskCreateExt(..)*
Inputparameters	9. void pointer naar de taak; 10. naam van de taak; 11. stackdiepte; 12. pointer naar de data die kan als de parameter van de taak worden gebruikt; 13. prioriteit; 14. pointer naar de taakhandle; 15. pointer naar *top-of-stack* van de taak; 16. (optioneel) pointer naar memorylocatie;	10. void pointer naar de taak; 11. pointer naar de data die kan als de parameter van de taak worden gebruikt; 12. pointer naar *top-of-stack* van de taak; 13. prioriteit; 14. id == prioriteit; 15. pointer naar *bottom-of-stack* van de taak; 16. stackdiepte; 17. (optioneel) pointer naar memorylocatie; 18. taakspecifieke opties;
Returnwaarden	foutmelding	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

De functie van FreeRTOS heeft acht parameters, de functie van MicroC/OS-II heeft negen parameters. De zesde parameter van de MicroC/OS-II-functie is niet noodzakkelijk voor de wrapperfunctie; hij kan tijdens de implementatie van de wrapperfunctie berekend worden, omdat de diepte van de stack en pointer van de *top-of-stack* van de taak bekend zijn. De andere parameters van de functies zijn compatibel. De handle van de taak voor Microc/OS-II kan prioriteit van de taak zijn.

Eis 3. De wrapper moet de functie bevatten die een taak verwijdert.

Wrapper functie	*pr_TaskDelete(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vTaskDelete(..);*	*OSTaskDel(..);*
Inputparameters	- pointer naar de taakhandle;	- prioriteit;
Returnwaarden	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Als de handle van de taak als prioriteit wordt gedefineerd voor MicroC/OS-II, dan er is geen probleem voor de implementatie: de functies hebben dezelfde parameters en geen returnwaarden.

Eis 4. De wrapper moet de functie bevatten die de utvoering van een taak opschort.

Wrapper functie	*pr_TaskSuspend(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vTaskSuspend(..);*	*OSTaskSuspend(..);*
Inputparameters	- pointer naar de taakhandle;	- prioriteit;
Returnwaarden	geen	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

De functie van FreeRTOS heeft geen returnwaarde, de functie van MicroC/OS-II heeft een foutmelding als returnwaarde. Dit probleem kan op twee manieren worden opgelost:

3. De wrapperfunctie moet een returnwaarde hebben, in het geval van FreeRTOS kan ze altijd true retourneren.
4. De wrapperfunctie moet geen returnwaarde hebben, in het geval van MicroC/OS-II kan de returnwaarde genegeerd worden.

De handle van de taak werd als prioriteit gedefineerd voor MicroC/OS-II.

Eis 5. De wrapper moet de functie bevatten die de utvoering van een taak hervat.

Wrapper functie	*pr_TaskResume(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vTaskResume (..);*	*OSTaskResume (..);*
Inputparameters	- pointer naar de taakhandle;	- prioriteit;
Returnwaarden	geen	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

De functie van FreeRTOS heeft geen returnwaarde, de functie van MicroC/OS-II heeft een foutmelding als returnwaarde. Dit probleem kan op twee manieren worden opgelost:

1. De wrapperfunctie moet een returnwaarde hebben, in het geval van FreeRTOS kan ze altijd true retourneren.
2. De wrapperfunctie moet geen returnwaarde hebben, in het geval van MicroC/OS-II kan de returnwaarde genegeerd worden.

De handle van de taak werd als prioriteit gedefineerd voor MicroC/OS-II.

Eis 6. De wrapper moet de functie bevatten die de count van de hardwaretimer leest.

Wrapper functie	*pr_TaskGetTickCount()*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xTaskGetTickCount(..);*	*OSTimeGet(..);*
Inputparameters	geen	geen
Returnwaarden	tick count	tick count
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei functies hebben geen parameters en dezelfde returnwaarden. Er zijn geen problemen voor de implementatie van deze functies.

Eis 7. De wrapper moet de functie bevatten die een taak voor vaste tijd blokkeert.

Wrapper functie	*pr_TaskDelay(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vTaskDelay(..);*	*OSTimeDly(..);*
Inputparameters	- wachttijd;	- wachttijd;
Returnwaarden	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei functies hebben dezelfde parameters en geen returnwaarden. Er zijn geen problemen voor de implementatie van deze functies.

Maar deze functies zijn niet periodiek, de wachttijd is niet de periode van uitvoering van de taak. De tussentijd van de uitvoeringen van de taak is afhankelijk van de tijd wanneer deze functies worden aangeroepen. De tijd van de volgende uitvoering van de taak wordt berekend:

$$T_r = t_0 + t_p;$$

waar T_r de tijd is van de volgende uitvoering van de taak, t_0 de tijd van aanroepen van de functie en t_p de wachttijd.

In FreeRTOS is de functie *vTaskDelayUntil()* geïmplementeerd om periodieke uitvoering van de taak te realiseren. Om deze functie in MicroC/OS-II te implementeren moet de broncode van MicroC/OS-II worden aangepast. De tijd van de volgende uitvoering van de taak moet worden berekend op volgende manier:

$$T_r = t_0 + t_p;$$

$$t_0 = T_r;$$

waar T_r de tijd is van de volgende uitvoering van de taak, t_p – tijd van de laatste uitvoering van de taak en t_p de wachttijd.

Eis 8. De wrapper moet de functie bevatten die prioriteit van een taak verandert.

Wrapper functie	*pr_TaskPrioritySet(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vTaskPrioritySet(..);*	*OSTaskChangePrio(..);*
Inputparameters	- pointer naar de taakhandle; - nieuwe prioriteit;	- huidige prioriteit van de taak; - nieuwe prioriteit;
Returnwaarde	geen	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

De handle van de taak voor MicroC/OS-II werd als prioriteit gedefineerd. Tijdens implementatie van deze functie voor MicroC/OS-II moet de handle van de taak aan de nieuwe prioriteit gekoppeld worden.

De functie van FreeRTOS heeft geen returnwaarde, de functie van MicroC/OS-II heeft een foutmelding als returnwaarde. Dit probleem kan op twee manieren worden opgelost:

1. De wrapperfunctie moet een returnwaarde hebben, in het geval van FreeRTOS kan ze altijd true retourneren.
2. De wrapperfunctie moet geen returnwaarde hebben, in het geval van MicroC/OS-II kan de returnwaarde genegeerd worden.

Eis 9. De wrapper moet de functie bevatten die een queue creëert.

Wrapper functie	*pr_QueueCreate(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueGenericCreate(..);*	*OSQCreate(..);*
Inputparameters	1. het aantal elementen in de queue; 2. de grootte van één element; 3. type van de queue (constant);	1. pointer naar *top-of-stack* van de queue; 2. het aantal elementen in de queue;
Returnwaarde	pointer naar de handle van de queue	pointer naar de handle van de queue
Verschil in implementatie	dynamische geheugentoewijzing	statische geheugentoewijzing

MicroC/OS-II is op zo'n manier gebouwd dat de gebruiker voor het creëren van een queue of een taak het geheugen daarvoor statisch moet toewijzen. In het geval van de queue moet de gebruiker een array in de applicatie creëren. De lengte van de array moet gelijk zijn aan het aantal elementen in de queue en de elementen in de array moeten dezelfde zijn als de elementen in de queue. Daarna gebruikt het systeem het toegewezen geheugen van de array voor de queue.
FreeRTOS wijst het geheugen dynamisch toe. Het systeem kent de lengte en de grootte van de elementen van de queue die wordt gecreëerd. Op basis van deze kennis wijst het geheugen dynamisch toe binnen de implementatie van de functie *xQueueGenericCreate()*.
Dit probleem kan worden opgelost als de wrapperfunctie drie parameters heeft: het aantal elementen in de queue, de grootte van één element, de pointer naar *top-of-stack* van de queue. Voor MicroC/OS-II wordt de grootte van één element niet gebruikt, voor FreeRTOS wordt de pointer naar *top-of-stack* van de queue niet gebruikt. Maar in de applicatie moeten de arrays voor het creëren van een queue worden gedefineerd. Deze toepassing moet in de demoapplicatie 2.0 terugkomen. Deze oplossing is niet optimaal maar ik kon geen andere vinden.

De dynamische geheugentoewijzing is een nadeel van FreeRTOS, want de gebruiker kan de verdeling van het geheugen niet controleren.

Eis 10. De wrapper moet de functie bevatten die een item uit de queue leest.

Wrapper functie	*pr_QueueReceive(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueGenericReceive(..);*	*OSQPend(..);*
Inputparameters	- handle van de queue; - pointer naar het geheugen voor data; - wachttijd; - constante die bepaalt of het element vanuit queue moet worden verwijderd of niet;	- handle van de queue; - wachttijd; - pointer naar variabele van fout;
Returnwaarde	foutmelding	- pointer naar het geheugen voor data;
Verschil in implementatie	De functie kopieert het element vanuit de queue naar het adres van het geheugen van de data die als parameter was doorgegeven.	De functie retourneert de pointer naar het geheugen van de element in de queue.

Om deze functies te combineren moet een wrapperfunctie gecreëerd worden die een foutmelding als returnwaarde heeft en drie inputparameters: handle van de queue, pointer naar het geheugen voor data en wachttijd. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet de inhoud van returnpointer van de functie *OSQPend()* naar pointer van het geheugen voor data gekopieerd worden.

Eis 11. De wrapper moet de functie bevatten die een item naar de queue zendt.

Wrapper functie	*pr_QueueSend(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueGenericSend(..);*	*OSQPost(..);*
Inputparameters	- handle van de queue; - pointer naar het geheugen voor data; - wachttijd; - variabele die type queue bepaalt: FIFO of LIFO[1];	- handle van de queue; - pointer naar het geheugen voor data;
Returnwaarde	foutmelding	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

In de functie van FreeRTOS *xQueueGenericSend()* staat de wachttijd als parameter. Deze parameter bepaalt hoe lang de taak moet worden geblokkeerd als de queue vol is. In MicroC/OS-II kan een taak niet worden geblokkeerd als de taak een element wil zenden naar een queue die vol is. Dus moet in de wrapperfunctie de wachttijd gelijk aan nul zijn. Het type van de queues was met begeleiders besproken en er was besloten om de queue met FIFO-volgorde te gebruiken, dus de vierde parameter is constant.

[1] Zie Begrippenlijst, Bijlage I

Eis 12. De wrapper moet de functie bevatten die de context van de queue leegmaakt.

Wrapper functie	*pr_QueueReset(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueReset(..);*	*OSQFlush(..);*
Inputparameters	- handle van de queue;	- handle van de queue;
Returnwaarde	foutmelding	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei de functies hebben dezelfde parameters en dezelfde returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Eis 13. De wrapper moet de functie bevatten die een softwaretimer creëert.

Wrapper functie	*pr_TimerCreate(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xTimerCreate(..);*	*OSTmrCreate(..);*
Inputparameters	- naam van de softwaretimer; - periode van de timer; - variabele die bepaalt of de timer periodiek is of voor één keer (one-shot) wordt gecreëerd; - pointer naar argument voor timerfunctie; - pointer naar de timerfunctie;	- variabele die de tijd voor one-shot timer bepaalt of delay voor periodieke timer; - periode van de timer; moet voor one-shot timer gelijk zijn aan nul; - variabele die bepaalt of de timer periodiek is of voor één keer (one-shot) wordt gecreëerd; - pointer naar de timerfunctie; - pointer naar argument voor timerfunctie; - naam van de softwaretimer; - pointer naar variabele van fout;
Returnwaarde	handle van de timer	handle van de timer
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei functies hebben parameters die compatibel met elkaar zijn en dezelfde returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Eis 14. De wrapper moet de functie bevatten die een softwaretimer start.

Wrapper functie	*pr_TimerStart(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xTimerStart(..);*	*OSTmrStart(..);*
Inputparameters	- handle van de timer;	- handle van de timer; - pointer naar variabele van fout;
Returnwaarde	foutmelding	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier kan de wrapperfunctie gecreëerd worden met een foutmelding als returnwaarde en de handle van de timer als inputparameter. Binnen de implementatie van deze functie voor MicroC/OS-II wordt de tweede parameter van de functie *OSTmrStart()* gecheckt en afhankelijk van het resultaat de returnwaarde van wrapperfunctie bepaald.

Eis 15. De wrapper moet de functie bevatten die een softwaretimer stopt.

Wrapper functie	*pr_TimerStop(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xTimerStop(..);*	*OSTmrStop(..);*
Inputparameters	- handle van de timer; - wachttijd op het stoppen;	- handle van de timer; - variabele die bepaalt of de timerfunctie voor het stoppen moet worden aanroepen; - pointer naar nieuw argument voor de timerfunctie; - pointer naar foutcode;
Returnwaarde	foutmelding	true of false
Verschil in implementatie	niet belangrijk	niet belangrijk

De extra functionaliteit van deze functies die FreeRTOS en MicroC/OS-II bieden kan niet compatibel zijn. Dus de enige opplossing is deze functionaliteit te negeren. Hier kan een wrapperfunctie gecreëerd worden met een foutmelding als returnwaarde en de handle van de timer als inputparameter.

Eis 16. De wrapper moet de functie bevatten die een softwaretimer verwijdert.

Wrapper functie	*pr_TimerDelete(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xTimerDelete(..);*	*OSTmrDel(..);*
Inputparameters	- handle van de timer; - wachttijd;	- handle van de timer; - pointer naar foutcode;
Returnwaarde	foutmelding	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier kan de wrapperfunctie gecreëerd worden met een foutmelding als returnwaarde en de handle van de timer als inputparameter. De extra functionaliteit van deze functies die FreeRTOS biedt wordt genegeerd.

Eis 17. De wrapper moet de functie bevatten die Mutex creëert.

Wrapper functie	*pr_CreateMutex()*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xSemaphoreCreateMutex();*	*OSMutexCreate(..);*
Inputparameters	geen	- prioriteit van de Mutex; - pointer naar foutcode;
Returnwaarde	handle van Mutex	handle van Mutex
Verschil in implementatie	niet belangrijk	niet belangrijk

MicroC/OS-II biedt extra functionaliteit voor deze functie: de prioriteit voor Mutex bepalen. FreeRTOS heeft deze mogelijkheid niet, Mutex in FreeRTOS heeft altijd hoogste prioriteit. Daarom wordt de wrapperfuntie gemaakt zonder inputparameters en de handle van Mutex als returnwaarde. In de implementatie van deze functie voor MicroC/OS-II wordt de eerste parameter bepaald als de maximale prioriteit.

Eis 19. De wrapper moet de functie bevatten die Mutex verwijdert.

Wrapper functie	*pr_DeleteMutex(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vQueueDelete(..);*	*OSMutexDel(..);*
Inputparameters	- handle van Mutex;	- handle van Mutex; - variabele die bepaalt onder welke voorwaarden Mutex kan worden verwijderd; - pointer naar foutcode;
Returnwaarde	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

MicroC/OS-II biedt extra functionaliteit voor deze functie. De gebruiker kan deze functie zo configureren dat de Mutex niet kan worden verwijderd als hij door een taak wordt verwacht. Maar deze functionaliteit is niet mogelijk in FreeRTOS. Daarom moet in de implementatie van de wrapperfunctie voor MicroC/OS-II de tweede parameter als constante worden doorgegeven. Deze constante bepaalt dat Mutex altijd moet worden verwijderd.

Eis 20. De wrapper moet de functie bevatten die Mutex opneemt.

Wrapper functie	*pr_MutexTake(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xSemaphoreTake(..);*	*OSMutexPend(..);*
Inputparameters	- handle van Mutex; -wachttijd;	- handle van Mutex; - wachttijd; - pointer naar foutcode;
Returnwaarde	foutmelding	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier kan de wrapperfunctie gecreëerd worden met een foutmelding als returnwaarde en de handle van de timer en de wachttijd als inputparameters. Binnen de implementatie van deze functie voor MicroC/OS-II wordt de derde parameter van de functie *OSMutexPend()* gecheckt en afhankelijk van de resultaat wordt de returnwaarde van de wrapperfunctie bepaald.

Eis 21. De wrapper moet de functie bevatten die Mutex vrijlaat.

Wrapper functie	*pr_MutexGive(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xSemaphoreGive(..);*	*OSMutexPost(..);*
Inputparameters	- handle van Mutex;	- handle van Mutex;
Returnwaarde	foutmelding	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei functies hebben dezelfde parameters en returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Eis 22. De wrapper moet de functie bevatten die een item uit de queue vanuit de interruptroutine leest.

Wrapper functie	*pr_QueueReceiveFromISR(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueReceiveFromISR(..);*	*OSQAccept(..);*
Inputparameters	- handle van queue; - pointer naar het geheugen voor data; -variabele die bepaalt of de taak die een plaats in de queue verwacht, moet worden gedeblokkeerd of niet;	- handle van queue; - pointer naar foutcode;
Returnwaarde	foutmelding	- pointer naar het geheugenadres voor data;
Verschil in implementatie	De functie kopieert het element vanuit de queue naar het adres van het geheugen van de data dat als parameter was doorgegeven.	De functie retourneert de pointer naar het geheugen van het element in de queue.

Hier zit eenzelfde probleem als bij 10. De opplossing is dezelfde. De extra functionaliteit van de FreeRTOS-functie wordt genegeerd, want die is onmogelijk binnen MicroC/OS-II. De derde parameter van de FreeRTOS-functie wordt als constante doorgegeven. Deze constante bepaalt dat de wachtende taak moet worden gedeblokkeerd.

Eis 23. De wrapper moet de functie bevatten die een item vanuit de interruptroutine naar de queue zendt.

Wrapper functie	*pr_QueueSendFromISR(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueGenericSendFromISR(..);*	*OSQPost(..);*
Inputparameters	- handle van de queue; - pointer naar het geheugen voor data; - variabele die bepaalt of de taak die het element in de queue verwacht, moet worden gedeblokkeerd of niet; - variabele die het type queue bepaalt: FIFO of LIFO;	- handle van de queue; - pointer naar het geheugen voor data;
Returnwaarde	foutmelding	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

De extra functionaliteit van FreeRTOS functie is genegeerd, want die is onmogelijk binnen MicroC/OS-II. De derde parameter van de FreeRTOS-functie wordt als constante doorgegeven. Deze constante bepaalt dat de wachtende taak moet worden gedeblokkeerd. Het type queues was met begeleiders besproken en er was besloten om de queue met FIFO-volgorde te gebruiken, dus de vierde parameter is een constante.

Eis 24. De wrapper moet de functie bevatten die de huidige toestand van de taak weergeeft.

Wrapper functie	*pr_TaskGetStatus(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*eTaskGetState(..);*	*OSTaskQuery(..);*
Inputparameters	- handle van de taak;	- prioriteit van de taak; - pointer naar TCB-blok[1];
Returnwaarde	toestand van de taak	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier wordt de wrapperfunctie gecreëerd die de handle van de taak als inputparameter heeft en de toestand van de taak als returnwaarde. De handle van de taak voor MicroC/OS-II werd als prioriteit gedefineerd. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet het TCB-blok gecheckt worden. Het TCB-blok bevat de huidige toestand van de taak die als returnwaarde wordt teruggegeven. De gebruiker heeft vrije toegang tot het TCB-blok in MicroC/OS-II. Dit is een nadeel van MicroC/OS-II want zo kan de gebruiker daar per ongeluk iets veranderen, wat het werk van het systeem kan beschadigen.

Eis 25. De wrapper moet de functie bevatten die de omvang van de resterende stack van een taak weergeeft.

Wrapper functie	*pr_TaskGetStack(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*uxTaskGetStackHighWaterMark(..);*	*OSTaskQuery(..);*
Inputparameters	- handle van de taak;	- prioriteit van de taak; - pointer naar TCB-blok[1];
Returnwaarde	vrije stack	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier wordt de wrapperfunctie gecreëerd die de handle van de taak als inputparameter heeft en de vrije stack van de taak als returnwaarde. De handle van de taak voor MicroC/OS-II werd als prioriteit gedefinieerd. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet het TCB-blok gecheckt worden. Het TCB-blok bevat de omvang van de stack en de omvang van de gebruikte stack van de taak, het verschil hiertussen wordt als returnwaarde teruggegeven.

Eis 26. De wrapper moet de functie bevatten die de prioriteit van een taak weergeeft.

Wrapper functie	*pr_TaskGetPriority(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*uxTaskPriorityGet(..);*	*OSTaskQuery(..);*
Inputparameters	- handle van de taak;	- prioriteit van de taak; - pointer naar TCB-blok;
Returnwaarde	prioriteit van de taak	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier wordt de wrapperfunctie gecreëerd die de handle van de taak als inputparameter heeft en de prioriteit van de taak als returnwaarde. De handle van de taak voor MicroC/OS-II werd als prioriteit

[1] Zie Begrippenlijst, Bijlage I

gedefinieerd. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet het TCB-blok gecheckt worden. Het TCB-blok bevat de prioriteit van de taak die als returnwaarde wordt teruggegeven.

Eis 27. De wrapper moet de functie bevatten die de naam van een taak weergeeft.

Wrapper functie	*pr_TaskGetName(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*pcTaskGetTaskName(..);*	*OSTaskQuery(..);*
Inputparameters	- handle van de taak;	- prioriteit van de taak; - pointer naar TCB-blok;
Returnwaarde	naam van de taak	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier wordt de wrapperfunctie gecreëerd die de handle van de taak als inputparameter heeft en het naam van de taak als returnwaarde. De handle van de taak voor MicroC/OS-II werd als prioriteit gedefinieerd. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet het TCB-blok gecheckt worden. Het TCB-blok bevat het naam van de taak die als returnwaarde wordt teruggegeven.

Eis 28. De wrapper moet de functie bevatten die algemene informatie over een taak weergeeft: de huidige toestand, de omvang van de resterende stack, de prioriteit, de naam en de CPU- tijd gedurende welke een taak is uitgevoerd.

Wrapper functie	*pr_TaskGetState(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*uxTaskGetSystemState(..);*	*OSTaskQuery(..);*
Inputparameters	- pointer naar het *TaskStatus_t* structuur; - grootte van array, is gelijk aan het aantal aangemaakten taken; - pointer naar variabele die de algemene tijd van de werking van het systeem bevat;	- prioriteit van de taak; - pointer naar TCB-blok;
Returnwaarde	de array van *TaskStatus_t*-structuur die de informatie over de taken bevat;	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Deze functies zijn niet compatibel, want FreeRTOS heeft een functie die het aantal gecreëerde taken retourneert. Dit getal is gebruikt als tweede parameter van de FreeRTOS-functie. In MicroC/OS-II bestaat geen functie die het aantal gecreëerde taken bepaalt. De einige opplossing is in de implementatie van wrapperfunctie voor FreeRTOS de volgende functie aan te roepen: *eTaskGetState(); uxTaskGetStackHighWaterMark(); uxTaskPriorityGet(); pcTaskGetTaskName().*

Om de CPU- tijd gedurende welke de taak is uitgevoerd te weten, moet de functie in de broncode van FreeRTOS toegevoegd worden. Het TCB-blok van FreeRTOS bevat de runtime van de taak. Het TCB-blok is gesloten voor de gebruiker. In de broncode van FreeRTOS kan de functie worden geïmplementeerd die het TCB-blok van de taak ziet en de runtime van de taak retourneert.

Eis 29. De wrapper moet de functie bevatten die het aantal vrije plaatsen in een queue weergeeft.

Wrapper functie	*pr_qQuery(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*uxQueueSpacesAvailable(..);*	*OSQQuery(..);*
Inputparameters	- pointer naar de handle van de queue;	- pointer naar de handle van de queue; - pointer naar de structuur die de informatie over de queue bevat;
Returnwaarde	aantal vrije plaatsen in een queue	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier wordt de wrapperfunctie gecreëerd die de handle van de queue als inputparameter heeft en de naam van de taak als returnwaarde. In de implementatie van de wrapperfunctie voor MicroC/OS-II moet de structuur die de informatie over de queue bevat, gecheckt worden. Deze structuur bevat de grootte van de queue en het aantal gebruikte plaatsen in de queue. Het verschil hiertussen kan als returnwaarde worden terugegeven.

Eis 30. De wrapper moet de functie bevatten die een set van de queues creëert.

Wrapper functie	*pr_QueueCreateSet(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueCreateSet(..);*	niet geïmplementeerd
Inputparameters	- de totale lente van de queues in de set;	
Returnwaarde	pointer naar de sethandle	
Verschil in implementatie	niet belangrijk	

De mogelijke oplossing bescheef ik verder.

Eis 31. De wrapper moet de functie bevatten die een queue aan de set toevoegt.

Wrapper functie	*pr_QueueAddToSet(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueAddToSet(..);*	niet geïmplementeerd
Inputparameters	- pointer naar de queue handle; - pointer naar de sethandle;	
Returnwaarde	foutmelding	
Verschil in implementatie	niet belangrijk	

De mogelijke oplossing bescheef ik verder.

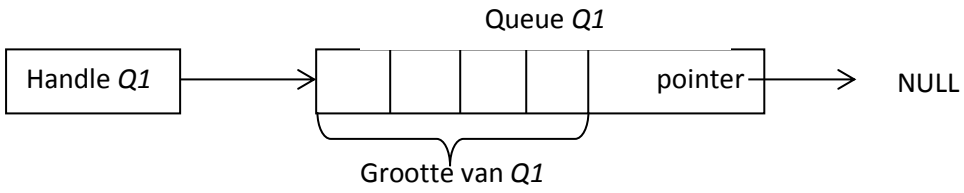
Eis 32. De wrapper moet de functie bevatten die een geactiveerde queue vanuit de set kiest.

Wrapper functie	*pr_QueueSelectFromSet(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xQueueSelectFromSet(..);*	niet geïmplementeerd
Inputparameters	- pointer naar de queue-handle; - wachttijd;	
Returnwaarde	- pointer naar de handle van de geactiveerde queue;	
Verschil in implementatie	niet belangrijk	

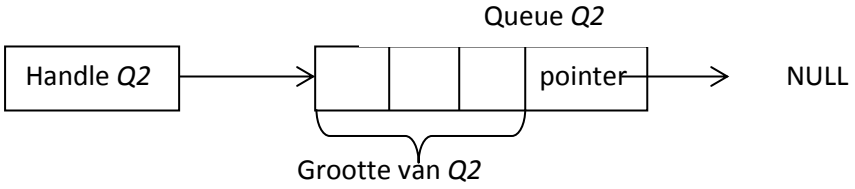
De set van queues is niet geïmplementeerd in MicroC/OS-II. Deze functie is een belangrijke functie van het systeem, ze geeft de mogelijkheid om een taak synchroniseren die uit meerdere queues leest. Ik heb gekeken hoe deze functie is gerealiseerd in FreeRTOS. Elke queue heeft een structuur waarin alle informatie over de queue staat. Deze structuur wordt aangemaakt bij het creëren van de queue. In FreeRTOS heeft deze structuur een pointer naar de queue-structuur, die bij default nul is. De FreeRTOS-functie: *xQueueCreateSet(..)* creëert een nieuwe queue(queueSET). De grootte van deze nieuwe queue(queueSET) is de totaal grootte van de queues(Q1, Q2, ...) die de set zal bevatten. Als een queue aan de set is toegevoegd met de functie: *xQueueAddToSet(..)*, wordt de pointer binnen de queue-structuur gelijk aan de handle van de set gesteld. De functie *xQueueSelectFromSet(..)* leest vanuit de queueSET en returneert de handle van geactiveerde queue in de set (queue die krijgt een item binnen). Wanneer in een van de queues die aan de set zijn toegevoegd een element komt, wordt deze queue naar set(queueSET) gestuurd, zie schema 9.1.

Schema 9.1 De realisatie van het werk van de set functies in FreeRTOS

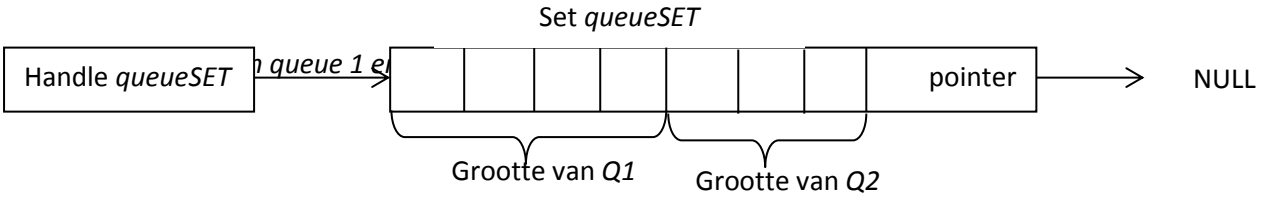
4. *Creëren van queue 1 xQueueGenericCreate(..)*



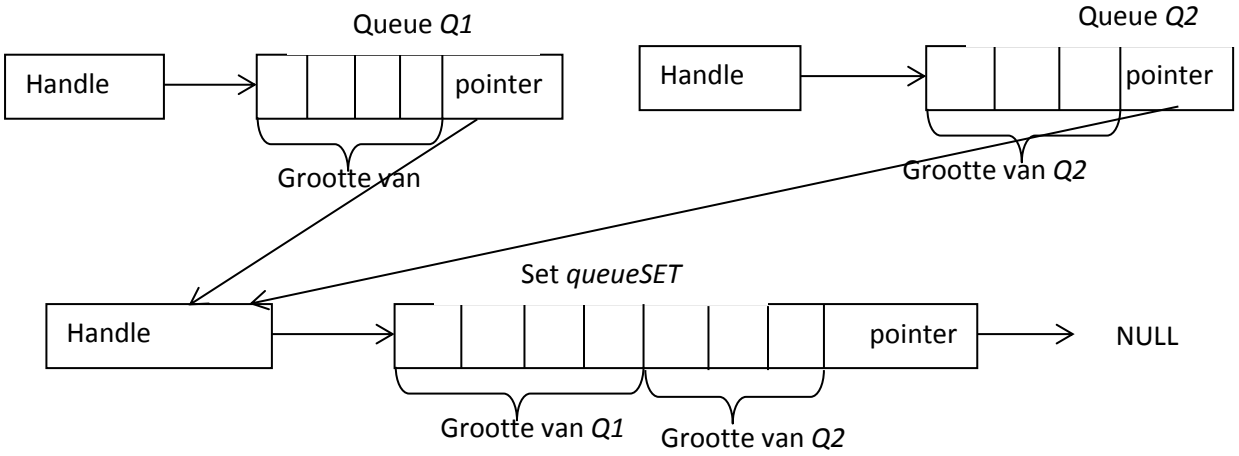
5. *Creëren van queue 2 xQueueGenericCreate(..)*



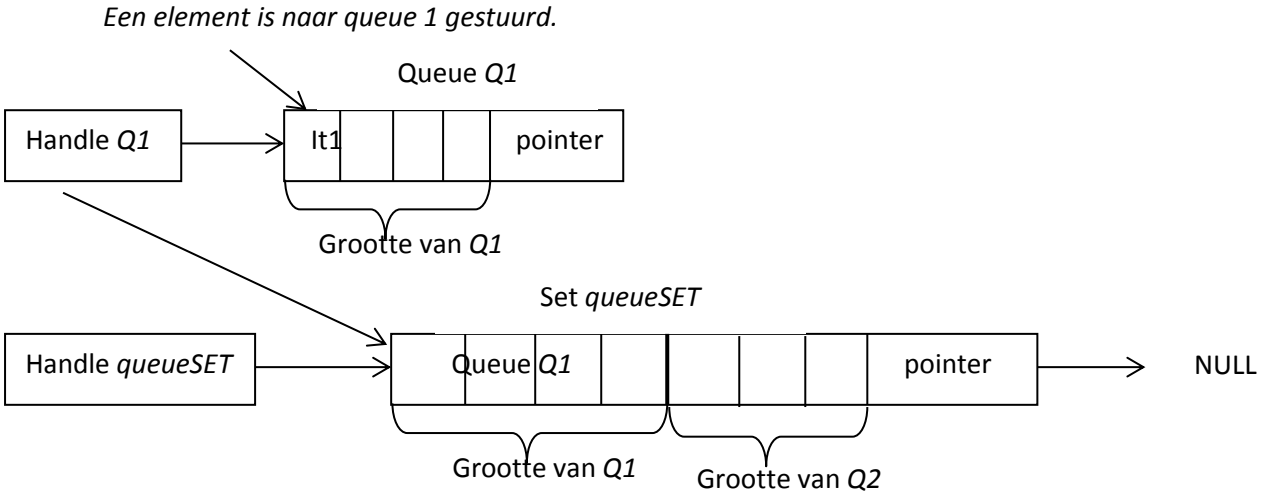
6. *Creëren van set xQueueCreateSet(..)*



6. *Toevoegen de queue 1 en queue 2 in de set `xQueueAddToSet(..)`*



7. Kiezen van welke queue is een data gekregen in de set: *`xQueueSelectFromSet(..);`*



Op dezelfde manier kunnen de setfuncties in MicroC/OS-II worden gerealiseerd. Maar dat vereist veranderingen in de broncode van MicroC/OS-II.

Eis 33. De wrapper moet de functie bevatten die Counting Semaphore creëert.

Wrapper functie	*`pr_CreateSemaphore(..)`*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*`xQueueCreateCountingSemaphore(..);`*	*`OSSemCreate(..);`*
Inputparameters	- maximale telwaarde van de counter die kan worden bereikt; - begintelwaarde van de counter;	- maximale telwaarde van de counter die kan worden bereikt;
Returnwaarde	pointer naar de handle van de semaphore;	pointer naar de handle van de semaphore;
Verschil in implementatie	niet belangrijk	niet belangrijk

Tijdens implementatie van de wrapperfunctie voor FreeRTOS moet de tweede parameter van de functie *`xQueueCreateCountingSemaphore`*`()` als nul worden doorgegeven, want MicroC/OS-II biedt deze functionaliteit niet.

Eis 34. De wrapper moet de functie bevatten die een Semaphore verwijdert.

Wrapper functie	*pr_DeleteSemaphore(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vQueueDelete(..);*	*OSSemDel(..);*
Inputparameters	- pointer naar de handle van de semaphore;	- pointer naar de handle van de semaphore; - variabele die bepaalt onder welke voorwaarden Mutex kan worden verwijderd; - pointer naar foutcode;
Returnwaarde	foutmelding	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

MicroC/OS-II biedt extra functionaliteit voor deze functie. De gebruiker kan deze functie zo configureren dat een Semaphore niet kan worden verwijderd als hij door een taak wordt verwacht. Maar deze functionaliteit is niet mogelijk in FreeRTOS. Daarom moet in de implementatie van de wrapperfunctie voor MicroC/OS-II de tweede parameter als constante worden doorgegeven. Deze constante bepaalt dat Semaphore altijd verwijderd moet worden.

Eis 35. De wrapper moet de functie bevatten die een Semaphore opneemt.

Wrapper functie	*pr_SemaphoreTake(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xSemaphoreTake(..);*	*OSSemPend(..);*
Inputparameters	- pointer naar de handle van de semaphore; -wachttijd;	- pointer naar de handle van de semaphore; - wachttijd; - pointer naar foutcode;
Returnwaarde	foutmelding	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Hier kan de wrapperfunctie gecreëerd worden met een foutmelding als returnwaarde, de handle van de Semaphore en de wachttijd als inputparameters. Binnen de implementatie van deze functie voor MicroC/OS-II wordt de derde parameter van de functie *OSSemPend()* gecheckt en wordt afhankelijk van het resultaat de returnwaarde van de wrapperfunctie bepaald.

Eis 36. De wrapper moet de functie bevatten die een Semaphore vrijlaat.

Wrapper functie	*pr_SemaphoreGive(..)*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xSemaphoreGive(..);*	*OSSemPost(..);*
Inputparameters	- pointer naar de handle van de semaphore;	- pointer naar de handle van de semaphore;
Returnwaarde	foutmelding	foutmelding
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei functies hebben dezelfde parameters en returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Eis 37. De wrapper moet de functie bevatten die de scheduler opschort.

Wrapper functie	*pr_TaskSuspendAll()*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*vTaskSuspendAll(..);*	*OSSchedLock(..);*
Inputparameters	geen	geen
Returnwaarde	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei de functies hebben geen parameters en geen returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Eis 38. De wrapper moet de functie bevatten die het werk van de scheduler hervat.

Wrapper functie	*pr_TaskResumeAll()*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*xTaskResumeAll(..);*	*OSSchedUnlock(..);*
Inputparameters	geen	geen
Returnwaarde	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei de functies hebben geen parameters en geen returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Eis 39. De wrapper moet de functie bevatten die de systeeminterrupts uitschakelt.

Wrapper functie	*pr_ENTER_CRITICAL()*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*taskENTER_CRITICAL(..);*	*OS_ENTER_CRITICAL(..);*
Inputparameters	geen	geen
Returnwaarde	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei de functies hebben geen parameters en geen returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Eis 40. De wrapper moet de functie bevatten die de systeeminterrupts aanzet.

Wrapper functie	*pr_EXIT_CRITICAL()*	
RTOS	**FreeRTOS**	**MicroC/OS-II**
Functie	*taskEXIT_CRITICAL(..);*	*OS_EXIT_CRITICAL(..);*
Inputparameters	geen	geen
Returnwaarde	geen	geen
Verschil in implementatie	niet belangrijk	niet belangrijk

Allebei de functies hebben geen parameters en geen returnwaarden. Er zijn geen problemen voor de implementatie van de wrapperfunctie.

Bijlage XIX. Testplan voor de wrapper

Testcase		Eis	Opstelling	Verwachte resultaat
1	Eis 2	De wrapper moet de functie bevatten die een taak creëert. *pr_TaskCreate()*	In de demoapplicatie waren acht taken gecreëerd met de functie*: pr_TaskCreate()*. Tijdens de test moet de demoapplicatie opgestart worden. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Alle taken moeten gecreëerd worden eerst met FreeRTOS en daarna met MicroC/OS-II.
2	Eis 3	De wrapper moet de functie bevatten die een taak verwijdert. *pr_TaskDelete().*	In de demoapplicatie was de *Creator_taak* gecreëerd. Deze taak creëert de andere taken en daarna verwijdert zichzelf met de functie: *pr_TaskDelete().*Tijdens de test moet de demoapplicatie opgestart worden. Deze test moet worden uitgevoerd eerst onder FreeRTOS MicroC/OS-II en daarna onder MicroC/OS-II.	FreeRTOS: In de statetabel moet duidelijk zijn dat de Creator taak is verwijderd. MicroC/OS-II: In de statetabel moet duidelijk zijn dat de Creator taak is niet bestaat.
3	Eis 8	De wrapper moet de functie bevatten die prioriteit van een taak verandert. *pr_TaskPrioritySet()*	Hier moet in de demoapplicatie de functie: *pr_TaskPrioritySet()* die de priority van, bijvoorbeeld, de UART taak verandert, worden toegevoegd, en daarna de statetabel moet worden uitgeprint. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II. Onder FreeRTOS wordt de priority van de UART taak van 3 naar 4 veranderd. Onder MicroC/OS-II wordt de priority van de UART taak van 10 naar 15 veranderd.	In de uitprint van de statetabel moet de priority van de UART taak veranderd worden.
4	Eis 4	De wrapper moet de functie bevatten die de utvoering van een taak opschort. *pr_TaskSuspend()*	Hier moet in de demoapplicatie de functie: *pr_TaskSuspend(),* worden toegevoegd die, bijvoorbeeld , de *Interpreter_taak* opschort. Daarna moet de toestand van de *Interpreter_taak* worden uitgeprint, vervolgens na het printen van de toestand wordt de *Interpreter_taak* hervat en weer de toestand van de *Interpreter_taak* wordt uitgeprint. De opschortte functie wordt in de *main* toegevoegd en de resumefunctie: : *pr_TaskResume(),*	Eerst moet de toestand van de *Interpreter_taak Suspend* zijn.Na het opscorten moet de toestand van *Interpreter_taak* naar *Blocked* verandert worden.
	Eis 5	De wrapper moet de functie bevatten die de utvoering van een taak hervat. *pr_TaskResume()*		

			wordt in de *Diagnose_taak* toegevoegd, na het uitprinten van de toestanden. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder het MicroC/OS-II.	
5	Eis 7	De functie die een taak vertraagt op de bepaalde periode. *pr_TaskDelayUntil()*	De functie: *pr_TaskDelayUntil(),* is gebruikt in de *UART_taak* en *Diagnose_taak*. Hier moet naar de periodiciteit van de uitvoering van deze functies worden gekeken. Daarvoor moet in de *UART_taak* een functie worden toegevoegd die per elke uitvoering van de taak een melden moet printen, dat de taak runt. De periode van de uitvoering moet 0.02 secunde zijn. De *Diagnose_taak* moet de statetabel elke 5 secunde printen. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat de uitvoering van de *UART_taak* en *Diagnose_taak* periodiek zal zijn, met de respectievelijke perioden van 0.02 en 5 secunden.
6	Eis 6	De wrapper moet de functie bevatten die de count van de hardware timer leest. *pr_TaskGetTickCount()*	In de *Syslog_taak* van de demoapplicatie gebruikt de softtimer om de tijd van de berichten bij te houden. Tijdens deze test moet de functie van interne timer: *pr_TaskGetTickCount(),* naast de softtimer worden aangeroepen. De tijd van de interne timer moet ook worden uitgeprint. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat de tijd die door de interne timer wordt gekregen, en de tijd van de softtimer, gelijk zullen zijn.
7	Eis 9	De wrapper moet de functie bevatten die een queue creëert. *pr_QueueCreate()*	In de demoapplicatie zijn zeven queues gecreëerd met de functie: *pr_QueueCreate()*. Tijdens deze test moet de demoapplicatie opgestart worden. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat de melding over het succesvolle creëren van de queues wordt aangetoond.
8	Eis 10	**.**De wrapper moet de functie bevatten die een item uit de queue leest. *pr_QueueReceive()*	In de taken van demoapplicatie 2.0: *Control_taak, Interpreteter_taak, Syslog_taak, Calc_DEC_taak en Calc_HEX_taak* is de wrapperfunctie geïmplementeerd: *pr_QueueReceive(),* die de data uit de queues leest. Na het lezen van de queues moet het bericht uitgeprint	Het is verwacht dat na het invoegen van het verzoek de *Control_taak, Interpreteter_taak, Syslog_taak, Calc_DEC_taak en Calc_HEX_taak* de bericht met uitgelezen data uit de

			worden. Tijdens deze test worden de volgende verzoeken ingevoerd: c444-5556 d SSCi Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	queues worden uitgeprint.
9	Eis 11	De wrapper moet de functie bevatten die een item naar de queue zendt. *pr_QueueSend()*	In de taken van demoapplicatie 2.0: UART_taak, *Control_taak,*, *Interpreteter_taak*, *Syslog_taak, Calc_DEC_taak en Calc_HEX_taak* is de wrapperfunctie geïmplementeerd: *pr_QueueSend(),* die een bericht naar de queue zendt. Na het zenden van de data naar de queue moet het bericht erover uitgeprint worden. Tijdens deze test worden de vogende verzoeken ingevoerd: c444-5556 d SSCi Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Het is verwacht dat na het invoegen van het verzoek, worden de UART_taak, *Control_taak,*, *Interpreteter_taak*, *Syslog_taak, Calc_DEC_taak en Calc_HEX_taak* de bericht dat de data naar de queue was gestuurd, uitgeprint.
10	Eis 12	De wrapper moet de functie bevatten die de context van de queue leeg maakt. *pr_QueueReset()*	Hier wordt de functie die een queue opnieuw zet*: pr_QueueReset(),* en de functie die toestand van de queue aantoont geïmplementeerd in de *Interpreteter_taak*. In de *Calc_DEC_taak en Calc_HEX_taak* worden de ontvangen functies geblokkeerd. Daardoor de *queue_HEX* en *queue_DEC* worden overgelopen. Wanneer de queues vol zullen zijn, wordt de reset functie vanuit *Interpreteter_taak* aangeroepen en de queues moeten weer leeg zijn. Hier worden zes verzoeken gemaakt: c444-5556 d c444-5556 h c444+5556 d c444*5556 d c5556 /444d c17-17 h Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat tijdens deze test het aantal plaatsen in de *queue_HEX* en *queue_DEC* wordt verminderd en na het aanroepen van de reset functie wordt het aantal plaatsen in de queues weer gelijk aan 5.
11	Eis 33	De wrapper moet de functie	Hier wordt in de demoapplicatie een	FreeRTOS: Hier is

		bevatten die een Counting Semaphore creëert. *pr_CreateSemaphore()*	Semaphore gecreëerd worden. De *UART_taak* neemt de semaphore met de functie: *pr_SemaphoreTake(),* maar na de uitvoering geeft niet ze de semaphore terug. De *Control_taak* moet deze semaphore ook nemen voor de uitvoering. Maar de semophore is bezet door de *UART_taak.* Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.	verwacht dat de toestand *Control_taak* wordt na het input niet veranderd. De toestand van *Control_taak* blijft *Blocked* en programma gaat niet input verwerken. MicroC/OS-II: Hier is verwacht dat de toestand *Control_taak* wordt veranderd naar *WAIT_OF_SEMAPHORE*.
	Eis 35	De wrapper moet de functie bevatten die een Semaphore opneemt. *pr_SemaphoreTake()*		
12	Eis 36	De wrapper moet de functie bevatten die een Semaphore vrijlaat. *pr_SemaphoreGive()*	Hier wordt in de demoapplicatie een Semaphore gecreëerd worden. De *UART_taak* neemt de semaphore,na de uitvoering geeft ze de semaphore terug met functie: *pr_SemaphoreGive()*. De *Syslog_taak* moet deze semaphore ook nemen voor de uitvoering. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht de volgende uitprint: "*UART_task takes the semaphore"* *"UART_task gives the semaphore"* *"Control_task takes the Semaphore"* *….* *"Control_task gives the semaphore"*
13	Eis 34	De wrapper moet de functie bevatten die een Semaphore verwijdert. *pr_DeleteSemaphore()*	Hier wordt de test N11 weer uitgevoerd. Maar voor het eind van de uitvoering van *UART_taak* wordt de semaphore verwijderd. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht een foutmelding: "*Control_task: kan not to take the semaphore."*
14	Eis 17	De wrapper moet de functie bevatten die Mutex creëert. *pr_CreateMutex()*	Aan het begin van de uitvoering van Syslog functie neemt ze Mutex. Aan het eind van de uitvoering geeft ze Mutex terug. Voor deze test voegde ik de functie `pr_DeleteMutex(xMutex)` in *Syslog_task* toe, na het syslog configuratie verzoek.. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Na de invoering van het syslog configuratie request verwacht hier een foutmelding: "*vSyslog_task_task: kan not to take the Mutex."*
	Eis 19	De wrapper moet de functie bevatten die Mutex verwijdert. *pr_DeleteMutex()*		
15	Eis 20	De wrapper moet de functie bevatten die Mutex vrijlaat. *pr_MutexGive()*	In de demoapplicatie is de Mutex gebruikt om de *Syslog* functie te beveiligen. Tijdens de test wordt de Mutex uit Syslog functie verwijderd. De diagnose_task wordt uitgevoerd met	Hier is verwacht dat de output chaotisch uit.
	Eis 18	De wrapper moet de functie		

		bevatten die Mutex neemt. *pr_MutexTake()*	het periode 0,1sec en *UART_task* wordt uitgevoerd met het periode 0.02sec. Bij de elke uitvoering *UART_task* moet printen: "*UART_task*: Running" en *Diagnose_task*: *"Diagnose_task running"* Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	
16	Eis 13	De wrapper moet de functie bevatten die een software timer creëert. *pr_TimerCreate()*	In de demoapplicatie wordt de softtimer gecreëerd, met de functie: *pr_TimerCreate()* en opgestart met de functie: *pr_TimerStart()* . Bij de output is de tijd van de softtimer weergegeven. Bij deze test worden acht verzoeken gemaakt. Na de tweede verzoek wordt de functie aangeroepen die de softtimer stoppen: *pr_TimerStop().* Na de vierde verzoek wordt de functie aangeroepen die de timer weer starten. Daarna na de zesde verzoek wordt de functie aangeroepen die de softtimer delete: *pr_TimerDelete(),* en meteen opnieuw starten. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier wordt de volgende output verwacht: Voor de tweede verzoek: wordt de output geprint met de tijd erbij; Na de tweede verzoek: wordt de output geprint maar de tijd die bij de output staat, wordt niet veranderd. Na de vierde verzoek: wordt de output geprint met de tijd erbij; Na de zesde verzoek: eerst wordt een foutmelding uitgeprint worden, dat de timer niet kan worden opgestart. Daarna wordt de output geprint maar de tijd die bij de output staat, wordt niet veranderd.
	Eis 14	De wrapper moet de functie bevatten die een software timer opstart. *pr_TimerStart()*		
	Eis 15	De wrapper moet de functie bevatten die een software timer stopt. *pr_TimerStop()*		
	Eis 16	De wrapper moet de functie bevatten die een software timer verwijdert. *pr_TimerDelete()*		
17	Eis 29	De wrapper moet de functie bevatten die een set van de queue's creëert. *pr_QueueCreateSet()*	In de demoapplicatie werd de set van de queues gecreëerd in de *Control_taak* en *Syslog_taak*, met de functie: *pr_QueueCreateSet().* Tijdens deze test eerst moet de demoapplicatie laten worden gedraaid. Daarna moet de functie die de set creëert geblokkeerd en de applicatie wordt weer gedraaid. Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.	Hier wordt verwacht dat eerst de demoapplicatie normaal gaat uitvoeren, dus de berekeningen worden uitgevoerd, time controle tenminste een keer per 0.5 secunde en foutmeldingen. Daarna als de setfuncties worden geblokkeerd, dan de periodiek van de Control taak wordt gebroken, en de Syslog taak kan niet optimaal worden
	Eis 30	**.**De wrapper moet de functie bevatten die een queue in de set voegt toe. *pr_QueueAddToSet()*		
	Eis 31	De wrapper moet de functie bevatten die geactieveerde queue vanuit de set kiest.		

		pr_QueueSelectFromSet()		gesynchroniseerd door de twee queues: queue PRINT en queue Syslog. (Onder optimale synchronisatie is begrepen dat met het gebruik van de set van de queues Syslog taak wordt geblokkeerd voor oneindig tijd tot de data komt in de queue Syslog of queue PRINT binnen. Zonder het gebruikt van de set van de queues dat niet mogelijk is, want de taak zal dan wachten de event alleen in één queue).
18	Eis 28	De wrapper moet de functie bevatten die het aantal vrije plaatsen in een queue weergeeft. *pr_qQuery()*	In de *Diagnose_taak* van de demoapplicatie 2.0 staat de functie die het aantal vrije plaatsen in de queues weergeeft. Tijdens deze test wordt de receiver functie van de *queue_IN* geblokkeerd. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat het aantal plaatsen in de *queues_IN* wordt verminderd na elk verzoek.
19	Eis 24	De wrapper moet de functie bevatten die hoeveelheid van resterende stack van een taak weergeeft. *Pr_TaskGetStack()*	In de demoapplicatie in de Diagnose taak was de functie geïmplementeerd die de maat van de resterende stack van de bepaalde taak toont aan. De totale maat van de stack van de taak wordt bij het creëren van de taak gedefinieerd. Tijdens deze test wordt de demoapplicatie vijf keer opgestart. Elke keer wordt de maat van de stack van de elke taak voor 100 groter gedefinieerd. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat de maat van de resterende stack van de taken wordt voor 100 verhogen per elk opstarten.
20	Eis 25	De wrapper moet de functie bevatten die de prioriteit van een taak weergeeft. *prTaskGetPriority()*	In de demoapplicatie 2.0 in de Diagnose_taak was de functie geïmplementeerd die de prioriteit van de bepaalde taak aantoont. De prioriteiten zijn niet veranderd	FreeRTOS: Hier is verwacht dat de functie: *prTaskGetPriority()* toont *eerst de prioriteiten van Control_taak* - 4 en van

			tijdens de uitvoering van de demoapplicatie 2.0. Tijdens deze test worden de prioriteiten van de taken worden wel veranderd. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II. Onder FreeRTOS wordt de priority van de *Control_taak* van 4 naar 3 veranderd en de priority van de *Diagnose_taak* van 2 naar 4. Onder het MicroC/OS-II wordt de priority van de *UART_taak* van 10 naar 15 veranderd en van de *Control_taak* van 9 naar 11.	*Diagnose_taak* - 2 , na de verandering toont ze prioriteit van *Control_taak* - 3 en van *Diagnose_taak* – 4. MicroC/OS-II: Hier is verwacht dat de functie: *prTaskGetPriority()* toont eerst de prioriteiten van *UART_taak – 10 en van Control_taak* – 8, na de verandering toont ze prioriteit van *UART_taak* - 11 en van *Control_taak* – 9.
21	Eis 26	De wrapper moet de functie bevatten die de naam van een taak weergeeft. *prTaskGetName()*	In de demoapplicatie in de *Diagnose_taak* was de functie geïmplementeerd die de naam van de taak aantoont. Tijdens deze test wordt de demoapplicatie opgestart. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat de namen van alle taken worden weergegeven op de terminal.
22	Eis 23	De wrapper moet de functie bevatten die de huidige toestand van de taak weergeeft. *prTaskGetStatus()*	In de demoapplicatie in de *Diagnose_taak* was de functie geïmplementeerd die de toestand van de taak aantoont. Tijdens deze test wordt de demoapplicatie opgestart. Voor deze test wordt in de *Diagnose_taak* de functie toegevoegd, die de *Control_taak* opschort. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat de tustanden van alle taken worden op de terminal weergegeven. Na het aangeroepen van de opschortte functie moet de toestand van de *Controle_taak Suspend* geworden.
23	Eis 27	De functie die alle informatie over de taak aan toont. De informatie bevat de volgende kenmerken van de taak: - Toestand; - Naam; - De maat van de resterende stack; - Huidige prioriteit; - Runtime;	In de demoapplicatie in de Diagnose_taak was de functie geïmplementeerd die alle informatie over de taak aantoont. Tijdens deze test wordt de demoapplicatie opgestart. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	Hier is verwacht dat één keer per 5 secunde wordt volledig state van elke taak aangetoond. De runtime van de *Syslog_taak* en *Diagnose_taak* moet tijdens de uitvoering van de demoapplicatie groeien.
24	Eis 33	De wrapper moet de functie bevatten die de scheduler opschort. *pr_TaskSuspendAll()*	Tijdens eerste uitvoering van de *Diagnose_taak* van de demoapplicatie 2.0 wordt de functie aangeroepen die scheduler opschot.	Tijdens deze test is verwacht dat na het aanroepen van de functie: *pr_TaskSuspendAll()*

	Eis 34	De wrapper moet de functie bevatten die het werk van de scheduler hervat. *pr_TaskResumeAll()*	Deze functie stopt de scheduler, maar interrupt blijft enable. Daarna gaat de Diagnose_taak door, aan het eind van de Diagnose taak wordt de functie aangeroepen, die scheduler opstart . De Diagnose taak moet tien keer de current tijd weergeven. Voor het printen van de output wordt *xil_printf()* functie gebruikt. De UART taak die de hogere prioriteit heeft, moet elke 0.02sec "*UART_task running*" printen. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II. Voor de overzicht van de resultaat wordt de periode van de *Diagnose_taak* gelijk aan 0.2 sec aangezet.	worden alle taken opgeschort. Er wordt alleen de Diagnose_taak uitgevoerd. Tijdens eerste tien uitvoeringen zal de *UART_taak* de *Diagnose_taak* niet breken. Over 10 uitvoering van de *Diagnose_taak* wordt scheduler weer opgestaart en de uitvoering van de *Diagnose_taak* wordt door de *UART_taak* gebroken. De tijd van het systeem moet doorgaan dus na de eerste tien uitvoering van de *Diagnose_taak* moet de *UART_taak* een vertraging hebben.
25	Eis 38	De wrapper moet de functie bevatten die de systeems interrupts schakelt uit. *pr_ENTER_CRITICAL()*	Tijdens deze test in Diagnose_taak van de demoapplicatie wordt tijdens de eerste uitvoering de functie *pr_ENTER_CRITICAL()* aangeroepen. Daarna gaat de *Diagnose_taak* door, aan het eind van de *Diagnose_taak* wordt de functie *pr_EXIT_CRITICAL()* aangeroepen. In de Diagnose taak wordt tien keer de functie: *pr_TaskGetTickCount()* toegevoegd. Voor het printen van de output wordt *xil_printf()* functie gebruikt. De UART_taak die de hogere prioriteit heeft, moet elke 0.02sec "*UART_task running*" printen. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.	De tijd die tien keer wordt uitgeprint, moet dezelfde blijven, want alle interrupten zijn disable. Ook is verwacht helemaal geen output van de andere taken, want de scheduler van het RTOS wordt ook gestopt. Na tien uitvoeringen van de *Diagnose_taak* moet de tijd doorlopen. De tijd van het systeem moet niet doorgaan dus na de eerste tien uitvoering van de *Diagnose_taak* moet de *UART_taak* geen vertraging hebben. Voor de overzicht van de resultaat wordt de periode van de *Diagnose_taak* gelijk aan 0.1 sec aangezet.
	Eis 39	De wrapper moet de functie bevatten die de systeemsinterrupts zet aan. *pr_EXIT_CRITICAL()*		

25	Eis 1	De wrapper moet de functie bevatten die scheduler start op. *pr_StartScheduler()*	In de demoapplicatie in de *main* is de functie *pr_StartScheduler()* aangeroepen. Deze test moet worden uitgevoerd eerst onder het MicroC/OS-II en daarna onder FreeRTOS.	De demoapplicatie runt correct.

Bijlage XX. Wrapper

```
/*
 * wrapper_rtos.h
 */

#ifndef WRAPPER_RTOS_H_
#define WRAPPER_RTOS_H_

/*#define USING_FreeRTOS*/
#define USING_microCOS_II

/*
*********************************************************************************************
*                                            ERROR CODE
*********************************************************************************************
*/
#define prOK               1                /*no errors*/
#define prERROR            -1               /*error*/
#define prCONF_ERROR       0                /*configuration error*/

/*
*********************************************************************************************
/*
*********************************************************************************************
*                                                PRIORITY's
*********************************************************************************************
*      PRIO_1 - highest, PRIO_10 - lowest
*/
int    pr_get_prio(int prio);


#define PRIO_1             pr_get_prio(1)
#define PRIO_2             pr_get_prio(2)
#define PRIO_3             pr_get_prio(3)
#define PRIO_4             pr_get_prio(4)
#define PRIO_5             pr_get_prio(5)
#define PRIO_6             pr_get_prio(6)
#define PRIO_7             pr_get_prio(7)
#define PRIO_8             pr_get_prio(8)
#define PRIO_9             pr_get_prio(9)
#define PRIO_10           pr_get_prio(10)
*********************************************************************************************
*                                            DATA TYPES
*********************************************************************************************
*/

#define prInt8             char
#define prInt8U            unsigned  char
#define prInt16U           unsigned short
#define prInt16            short
#define prInt32U           unsigned int
#define prInt32            int
#define prLongLongU        unsigned long long
#define prLongLong         long long
#define prSTK              prInt32U          /* stack of task entry     */
#define prTickType         prInt32U          /* type for timeout */
#define prErrorMessage     prInt8            /* type for error messages */

/* handle of the task!FreeRTOSConfig.h:configENABLE_BACKWARD_COMPATIBILITY == 1*/
typedef void*
         prTaskHandle;
```

```
/* handle of the queue!FreeRTOSConfig.h:configENABLE_BACKWARD_COMPATIBILITY == 1*/
typedef void*
         prQueueHandle;


typedef void*
         prSemaphoreHandle;                  /* handle of the semaphore*/
typedef void*
         prMutexHandle;              /* handle of the mutex*/
typedef void*
         prTimerHandle;              /* handle of the timer*/
typedef void*
         prQueueSetHandle;           /* handle of the set of queue's*/
typedef void*
         prQueueSetMemberHandle;     /* handle of the members in the set*/
typedef struct{                      /* contains the task's data*/
                            prInt8* tName;
                            prInt8* tStatus;
                            prInt32U tSTKsize;
                            prInt32U tPrio;
                            prInt32U tRunTime;
                  } prTASK_STATE;

/*
*********************************************************************************************
*                                                CONSTANTS
*********************************************************************************************
*/
#define prITSELF                    0
#define pr_Null                     0x00
#define prTRUE                      1
#define prFALSE                     0
#define prNull                      0
#define prOS_TMR_OPT_PERIODIC       prTRUE    /* for periodical software timer  */
#define prOS_TMR_OPT_ONE_SHOT       prFALSE   /* for one shot software timer  */

/*max timeout: 0xffffffffUL or 0xffff if configUSE_16_BIT_TICKS == 1 in FreeRTOSConfig.h */
#define prMAX_DELAY         0xffffffffUL
#define prTICK_RATE_MS      prTICK_RATE


/*
*********************************************************************************************
*                                             SYSTEMS FUNCTIONS
*********************************************************************************************
* Starts the multitasking process.  Before you can call pr_StartScheduler(), you MUST have
*called OSInit()
*              and you MUST have created at least one task.
*/
void pr_SistemInit();
void pr_StartScheduler();           /* starts scheduler running */
void pr_DISABLE_INTERRUPTS();       /* interrupts disable */
void pr_ENABLE_INTERRUPTS();        /* interrupts enable  */
void pr_ENTER_CRITICAL();   /* interrupts disable and stop scheduler */
void pr_EXIT_CRITICAL();    /* interrupts enable and start scheduler */
void pr_TaskSuspendAll();   /* suspends the scheduler !!leaves interrupts enabled */
void pr_TaskResumeAll();            /* resumes the scheduler */

/* Initialize the interrupt controller and Initialize the timers / uC/OS-II /*/
void pr_BSP_InitIO();
void pr_BSP_IntDisAll();   /*Disable all interrupts at the interrupt controller/uC/OS-II*/


/*
*********************************************************************************************
*                                             TASKS FUNCTIONS
```

```
*******************************************************************************************
*/
/*pr_TaskCreate() - creates a task;
* input: (void*)pr_taskcode - pointer to the task void;
*        (const prInt8 *)pr_Name - task name;
*        (prInt16U)prStackDepth - stack depth;
*        (void*)pr_Parameters - pointer to the data, which is used as input parameters for
*the task;
*        (prInt32U)prPriority - priority of the task;
*        (prSTK*)pr_STK - pointer to the tasks's top-of-stack;
*        (prTaskHandle*)pr_THandle - pointer to the task handle;
* output: error message;
*/
prErrorMessage pr_TaskCreate(void *pr_taskcode, const prInt8* pr_Name, const prInt16U
prStackDepth, void *pr_Parameters, prInt32U prPriority,  prTaskHandle *pr_THandle,  prSTK *
pr_STK);

/* pr_TaskDelete() - deletes a task; input:task handle (prITSELF - itself)
* !FreeRTOSConfig.h: INCLUDE_vTaskDelete == 1 ; !!os_cfg.h: OS_TASK_DEL_EN == 1;
*!!os_cfg.h: OS_TASK_CHANGE_PRIO_EN = 1
 */
void pr_TaskDelete(prTaskHandle);

/* pr_TaskPrioritySet() - changes priority; input: task handle (prITSELF - itself); new
*priority;
* !FreeRTOSConfig.h: INCLUDE_vTaskPrioritySet = 1
*/
prErrorMessage pr_TaskPrioritySet(prTaskHandle *pr_THandle, prInt32U prNewPriority);

/*pr_TaskResume() - changes Suspended state of the task to Ready; input: task handle;
 * !FreeRTOSConfig.h: INCLUDE_vTaskSuspend = 1; !!os_cfg.h: OS_TASK_SUSPEND_EN = 1
 */
prErrorMessage pr_TaskResume(prTaskHandle);

/*pr_TaskSuspend() - places the task into Suspended state; input:task handle (prITSELF -
*itself);
* !!leaves interrupts enabled!FreeRTOSConfig.h: INCLUDE_vTaskSuspend = 1;!!os_cfg.h:
*OS_TASK_SUSPEND_EN = 1
*/
prErrorMessage pr_TaskSuspend(prTaskHandle pr_THandle);

/*
*******************************************************************************************
*                                          TIME FUNCTIONS
*******************************************************************************************
*/
/*pr_TaskDelay() - locks the task for a fix number ticks;
* input: number of the clock ticks;
* !FreeRTOSConfig.h:INCLUDE_vTaskDelay = 1;
* Not periodically. It depends on the time of calling this function
*/
prErrorMessage pr_TaskDelay(prTickType);

/*pr_TaskGetTickCount() -  returns the number of ticks of hw timer; output: tick count
*(prTickType);
*  !!os_cfg.h: OS_TIME_GET_SET_EN = 1
*/
prTickType pr_TaskGetTickCount();

/*pr_TaskDelayUntil() - unlocks the task after a fix number ticks
* input:
*        time of the last execution - defined by itself, must be set once as the current
*time
*        number of ticks - absolute period
* !FreeRTOSConfig.h:INCLUDE_vTaskDelayUntil = 1;!!os_cfg.h: OS_TIME_GET_SET_EN = 1
* Periodically.
```

```
*/
void pr_TaskDelayUntil(prTickType* const, prTickType);

/*
*********************************************************************************************
*                                                QUEUE FUNCTIONS
*********************************************************************************************
*/
/* pr_QueueCreate() - creates a queue;
* input:
*  (prInt16U)prQueueLength - max number of item in the queue;
*        pr_Queue - pointer to the queue's top-of-stack;
*        prSizeElement - size of element in the queue;
* output:
*        (prQueueHandle)prQHandle - queue handle;
*/
prQueueHandle pr_QueueCreate(prInt16U , void* , prInt32U );

/* pr_QueueReceive() - reads an item from the queue;
          * input:
          *        (prQueueHandle)prQHandle - queue handle;
          *        (void *)pr_Item - pointer to memory address for item;
          *        (prTickType)prTimeOut - amount of time to wait (in clock ticks);
          *        (prInt32U)prSizeItem - size of the item
          * output:
          *        (prErrorMessage)prError - error message: if prError = prOK - is OK
          */
prErrorMessage pr_QueueReceive(prQueueHandle prQHandle,  void*  prItem, prTickType
prTimeOut, prInt32U prSizeItem);

/*pr_QueueSend() - sends an item to the back of the queue;
* input:
*        (prQueueHandle)prQHandle - queue handle;
*        (void *)pr_Item - pointer to the memory address of data;
*  output is error message, if prError = prOK - is OK
*  !!os_cfg.h: OS_Q_POST_EN = 1
*/
prErrorMessage pr_QueueSend(prQueueHandle prQHandle,  void*  prItem);

/*pr_QueueReset() - empties the queue;
* input:
*  (prQueueHandle)prQHandle - queue handle;
* output:(prErrorMessage)error message
*/
prErrorMessage  pr_QueueReset(prQueueHandle);

/*
*********************************************************************************************
*                                                SEMAPHORE FUNCTIONS
*********************************************************************************************
*/
/* pr_CreateSemaphore() - creates a counting semaphore;
*  input:
* (prInt16U)prMaxCount - is the initial value for the semaphore, to specify how many
*resources are available;
*  output: handle of the semaphore;
*/
prSemaphoreHandle pr_CreateSemaphore(prInt16U);

/* pr_DeleteSemaphore() - deletes a semaphore;
* input:
*        (prSemaphoreHandle)prSHandle - semaphore handle;
*  output:(prErrorMessage)error message
*/
prErrorMessage     pr_DeleteSemaphore(prSemaphoreHandle);
```

```
/* pr_SemaphoreTake() - obtains a semaphore;
* input:
*        (prSemaphoreHandle)prSHandle - semaphore handle;
*        (prTickType)prTimeOUT - amount of time to wait - number of the clock ticks;
* output:(prErrorMessage)error message
*/
prErrorMessage pr_SemaphoreTake(prSemaphoreHandle, prTickType);

/*pr_SemaphoreGive() gives the semaphore; input: semaphore handle; output: error code */
prErrorMessage     pr_SemaphoreGive(prSemaphoreHandle);

/*
*********************************************************************************************
*                                                MUTEX FUNCTIONS
*********************************************************************************************
*/
/* pr_CreateMutex() - creates of a mutex;
* output:
*                  handle of mutex;
*!!FreeRTOSConfig.h: configUSE_MUTEXES = 1;
*/
prMutexHandle pr_CreateMutex();

/* pr_DeleteMutex() - deletes a mutex;
* input:
*        (prMutexHandle)prMHandle - mutex handle;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
*/
prErrorMessage pr_DeleteMutex(prMutexHandle);

/* pr_MutexTake() - takes the mutex; input: mutex handle, time to wait;
output:(prErrorMessage)error message; */
prErrorMessage pr_MutexTake(prMutexHandle, prTickType);

/* pr_MutexGive() -  gives the semaphore; input: mutex handle; output:
(prErrorMessage)error message; */
prErrorMessage pr_MutexGive(prMutexHandle);

/*
*********************************************************************************************
*                                                FUNCTIONS FOR ISR
*********************************************************************************************
*/
/*pr_QueueReceiveFromISR() receives item from the queue, called from ISR routine;
* the task that is waiting for an item in the queue will be unblocked if its priority is
*higher or equal to the interrupted task;
* input:
*        queue handle;
*        pointer to the memory address for data;
*        (prInt32U)prSizeItem - size of the item
*  output:(prErrorMessage)error message
*/
prErrorMessage pr_QueueReceiveFromISR(prQueueHandle prQHandle, void* pr_Item, prInt32U
prSizeItem);

/* pr_QueueSendFromISR() - sends item to the back of the queue from ISR routine;
* the task that is waiting for an item in the queue will be unblocked if its priority is
*higher or equal to the interrupted task;
* input:
*        (prQueueHandle)prQHandle - queue handle;
*        (void*)pr_Item - pointer to data;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
*/
prErrorMessage pr_QueueSendFromISR(prQueueHandle, void*);

/*
```

```
*********************************************************************************************
*                                                SOFTWARE TIMER
*********************************************************************************************
*/
/*pr_TimerCreate() this function creates the software timer;
* input:
*   (prInt8 *)prName -   Is a  string that allows you to give a name to your timer;
*  (prTickType)prPeriod - Specifies the amount of time it will take before the timer
*expires.
*  const prInt8U prMode -   prOS_TMR_OPT_PERIODIC: defines type of the timer (periodically)
*                           prOS_TMR_OPT_ONE_SHOT: defines type of the timer (one-shot)
*  (void *)pr_Arg  -   the argument passed to the callback function when the timer *expires
*or is terminated.
* (void *)pr_CallBackFunction - specifies the address of a function (optional) that you
*want to execute when the timer expire;
* output: (prTimerHandle) handle of a timer;
*
*********************************************************************************************
*      THIS FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE FOR MicroC/OS-II
*
*                                                SOFTWARE TIMER
*    1. Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*    2. Must be defined in the code: OS_TASK_TMR_STK_SIZE and OS_TASK_TMR_PRIO
*    3. Must be defined frequency of timers update through OS_TMR_CFG_TICKS_PER_SEC in the
*        os_cfg.h
*    4. Must be enabled hooks in the os_cfg.h
*    6. In the file os_cpu_c.c:
*        in the function void  OSTimeTickHook (void):
*                 add:
*                           OSTmrSignal();
*********************************************************************************************
*/
prTimerHandle pr_TimerCreate(prInt8*, prTickType, const prInt8U, void*, void*);

/* pr_TimerStart() - starts the software timer;
* input:
*         (prTimerHandle)prTmHandle - the handle of the timer;
*         (prTickType)prTimeOut - amount of time to wait in clock ticks;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
* !!!Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*/
prErrorMessage pr_TimerStart(prTimerHandle);

/* pr_TimerStop() - stops the software timer;
* input:
*         (prTimerHandle)prTmHandle - the handle of the timer;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
* !!!Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*/
prErrorMessage pr_TimerStop(prTimerHandle);

/* pr_TimerDelete() - deletes the software timer;
* input:
*        (prTimerHandle)prTmHandle - the handle of the timer;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
* !!!Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*/
prErrorMessage  pr_TimerDelete(prTimerHandle);

/*
*********************************************************************************************
*                                                SET OF QUEUES
*********************************************************************************************
*    THIS FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE FOR MicroC/OS-II
*
*!!!!To use the set of queues functions, should be amended in the next source code:
```

```
         * 1. in the file ucos_ii.h
         *        in the definition of the OS_EVENT structure:
         *                add a line:
         *                        struct os_event *QueueSetContainer;
         * 2. in the file os_q.c
         *        in the function OS_EVENT  *OSQCreate (void    **start, INT16U    size)
         *        add a line inside the if-statement (if (pevent != (OS_EVENT *)0)):
         *                        pevent->QueueSetContainer = 0u;
         * 3. in the file os_q.c
         *        in the function INT8U  OSQPost (OS_EVENT *pevent, void *pmsg)
         *                add a code before the last call to OS_EXIT_CRITICAL(); :
         *                        if(pevent->QueueSetContainer!=0u)
         *                        {
         *                                OSQPost(pevent->QueueSetContainer, pevent);
         *                        }
*********************************************************************************************
*/

/* pr_QueueCreateSet() - creates a set of queue's; input: (prInt32U)total length of the
*queue's;
* output:set handle;!FreeRTOSConfig.h:configUSE_QUEUE_SETS = 1; os_cfg.h: OS_Q_ACCEPT_EN =1
*and OS_Q_EN = 1
*/
prQueueSetHandle pr_QueueCreateSet(prInt32U length);

/* pr_QueueCreateSet() - adds queue to the set; input: queue handle, set handle;
* output:(prErrorMessage)error message;  if prError = prOK - is OK;
* !FreeRTOSConfig.h:configUSE_QUEUE_SETS = 1; os_cfg.h: OS_Q_ACCEPT_EN  = 1 and OS_Q_EN = 1
*/
prErrorMessage pr_QueueAddToSet(prQueueHandle, prQueueSetHandle);

/*pr_QueueSelectFromSet() - selects queue from the set; input: set handle, max time to
*wait;
*output: prQueueSetMemberHandle;
*!FreeRTOSConfig.h:configUSE_QUEUE_SETS = 1; os_cfg.h: OS_Q_ACCEPT_EN  = 1 and OS_Q_EN = 1
*/
prQueueSetMemberHandle pr_QueueSelectFromSet(prQueueSetHandle, prTickType);

/*
*********************************************************************************************
*                                              DIAGNOSE FUNCTIONS
*********************************************************************************************
*********************************************************************************************
*                                              QUEUE DIAGNOSE FUNCTION
*********************************************************************************************
*/

/*pr_qQuery() - returns the number of free spaces at the queue; input: queue handle;
*output: number of the free spaces in the queue;*/
prInt32U pr_qQuery(prQueueHandle);

/*
*********************************************************************************************
*                                              TASK DIAGNOSE FUNCTION
*********************************************************************************************
*
* Function prTaskGetState(): returns  the current state of the task
* input: handle of the task;
* output: string the current state of the task;
* !!!!Must be defined in the FreeRTOSConfig.h: INCLUDE_uxTaskGetStackHighWaterMark  = 1;
os_cfg.h: OS_TASK_QUERY_EN  == 1
*/
const prInt8* pr_TaskGetStatus(prTaskHandle);

/*  Function pr_TaskGetStack():returns the size of not used stack;
* input: handle of the task;
```

```
* output: the size of not used stack;
* !!!!Must be defined in the FreeRTOSConfig.h: INCLUDE_uxTaskGetStackHighWaterMark = 1;
* os_cfg.h: OS_TASK_QUERY_EN = 1; OS_TASK_PROFILE_EN = 1; OS_TASK_CREATE_EXT_EN = 1;
*/
prInt32 pr_TaskGetStack(prTaskHandle);

/*  Function pr_TaskGetPriority(): returns the priority of the task;
* input: handle of the task;
* output: the priority of the task;
* !!!! Must be defined in the FreeRTOSConfig.h: INCLUDE_uxTaskPriorityGet = 1; os_cfg.h:
*OS_TASK_QUERY_EN  == 1
*/
prInt8U pr_TaskGetPriority(prTaskHandle);

/* Function pr_TaskGetName(): returns the name of the task;
* input: handle of the task;
* output: the name of the task;
* !Must be defined in the FreeRTOSConfig.h: INCLUDE_pcTaskGetTaskName  = 1; os_cfg.h:
*OS_TASK_NAME_EN  = 1, OS_TASK_QUERY_EN  = 1
*/
const prInt8* pr_TaskGetName(prTaskHandle);

/* Function prTaskGetState(): returns all information about the task;
* input: handle of the task;
*        pointer to the prTASK_STATE;
* output:(prErrorMessage)error message;
* !!Must be defined in the os_cfg.h: OS_TASK_QUERY_EN  = 1, OS_TASK_NAME_EN = 1,
*OS_TASK_CREATE_EXT_EN = 1, OS_TASK_PROFILE_EN = 1
* !!Must be defined in the FreeRTOSConfig.h: INCLUDE_pcTaskGetTaskName  = 1; os_cfg.h:
*OS_TASK_NAME_EN  = 1, OS_TASK_QUERY_EN  = 1,
* INCLUDE_uxTaskPriorityGet = 1
*
*       THIS FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE FOR FreeRTOS
*
* FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE
* 1. In the file tasks.c must be created the function:
*         #if ( configGENERATE_RUN_TIME_STATS == 1)
*                 uint32_t uxTaskGetRunTime(TaskHandle_t xTaskToQuery )
*                 {
*                          vTaskSuspendAll();
*                          TCB_t *pxTCB;
*                          pxTCB = prvGetTCBFromHandle( xTaskToQuery );
*                          xTaskResumeAll();
*                          return pxTCB->ulRunTimeCounter;
*
*                 }
*                 #endif
* 1. In the file task.h must be defined the function:
*        uint32_t uxTaskGetRunTime(TaskHandle_t xTaskToQuery );
*/
prErrorMessage pr_TaskGetState(prTaskHandle , prTASK_STATE*);
/*

*********************************************************************************************
*                                              EXTRA FUNCTION
*********************************************************************************************
*/
/*prGetConTICK_RATE() - returns the number millisecond per one ticks; is used to convert
milliseconds to ticks and vice versa*/
prTickType pr_GetTICK_RATE();

#endif /* WRAPPER_RTOS_H_*/
```

wrapper_rtos.c

```
/*
 * wrapper_rtos.c
 */


#include "wrapper_rtos.h"
#ifdef USING_FreeRTOS
                    /* Kernel includes. */
                    #include "FreeRTOS.h"
                    #include "task.h"
                    #include "queue.h"
                    #include "timers.h"
                    #include "semphr.h"
/*hardware timer interrupt configurations
    -   -   -   -

*/
#endif /* USING_FreeRTOS */

#ifdef USING_microCOS_II
          /* Kernel includes. */
          #include "ucos_ii.h"
          #include "app_cfg.h"
          #include "cpu.h"
          #include "xintc_l.h"
          #include  "bsp_mC_OS.h"
          #include  "os_cpu.h"
          #include <string.h>
          #include "semphr.h"
/*

*********************************************************************************
 *                                                  CONSTANTS

*********************************************************************************
*/
        #define prConfigTICK_RATE_HZ     OS_TICKS_PER_SEC/* The number of ticks in one second  */
        #define  BSP_TMR_VAL            (XPAR_CPU_M_AXI_DP_FREQ_HZ / OS_TICKS_PER_SEC)

/*hardware timer interrupt configurations
    -   -   -   -

*/
prInt16U pr_SetTaskHandle(prInt16U * prTHandle, prInt32U);
prInt32U pr_GetPrio(prInt32U * prtHandle);

/*set de handle of the task / microCOS_II */
prInt16U pr_SetTaskHandle(prInt16U * prTHandle, prInt32U prPriority)
{
          *prTHandle = (prInt16U)prPriority;
          return *prTHandle;
}
/*get the prio from handle */
prInt32U pr_GetPrio(prInt32U * prtHandle)
{
        prInt32U prio = (prInt32U)prtHandle;
        return prio;
}


#endif /* USING_microCOS_II */
*get priority;
 * 1 is highest, 10 is lowest;
 * limited for 10 priority's
 */
int pr_get_prio(int prio)
{
        #ifdef USING_FreeRTOS
                if(configMAX_PRIORITIES>10)
```

```
		{
			switch(prio)
			{
				case 1: return 10; break;
				case 2: return 9; break;
				case 3: return 8; break;
				case 4: return 7; break;
				case 5: return 6; break;
				case 6: return 5; break;
				case 7: return 4; break;
				case 8: return 3; break;
				case 9: return 2; break;
				case 10: return 1; break;
				default: return prCONF_ERROR; break;
			}
		}
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		if(OS_LOWEST_PRIO>16)
		{
			switch(prio)
			{
				case 1: return 7; break;
				case 2: return 8; break;
				case 3: return 9; break;
				case 4: return 10; break;
				case 5: return 11; break;
				case 6: return 12; break;
				case 7: return 13; break;
				case 8: return 14; break;
				case 9: return 15; break;
				case 10: return 16; break;
				default: return prCONF_ERROR; break;
			}
		}
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;

}


/* start scheduler running */
void pr_StartScheduler()
{
	#ifdef USING_FreeRTOS
		vTaskStartScheduler();
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		OSStart();
	#endif /* USING_microCOS_II */
}

/* interrupts disable */
void pr_DISABLE_INTERRUPTS()
{
	#ifdef USING_FreeRTOS
		taskDISABLE_INTERRUPTS();
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		microblaze_disable_interrupts();
	#endif /* USING_microCOS_II */
}

/* interrupts enable  */
void pr_ENABLE_INTERRUPTS()
{
	#ifdef USING_FreeRTOS
		taskENABLE_INTERRUPTS();
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
```

```
                microblaze_enable_interrupts();
        #endif /* USING_microCOS_II */
}

/* uC/OS-II initialization. FreeRTOS does not require initialization. It is defined only for
microS/OS-II */
void pr_SistemInit()
{
        #ifdef USING_microCOS_II
                OSInit();
        #endif /* USING_microCOS_II */
}

/* interrupts disable and stop scheduler */
void pr_ENTER_CRITICAL()
{
        #ifdef USING_FreeRTOS
                taskENTER_CRITICAL();
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                #if (OS_CRITICAL_METHOD == 3u)
                        OS_CPU_SR  cpu_sr = 0u;
                #endif
                OS_ENTER_CRITICAL();
        #endif /* USING_microCOS_II */
}

/* interrupts enable and start scheduler */
void pr_EXIT_CRITICAL()
{
        #ifdef USING_FreeRTOS
                taskEXIT_CRITICAL();
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                #if (OS_CRITICAL_METHOD == 3u)
                        OS_CPU_SR  cpu_sr = 0u;
                #endif
                OS_EXIT_CRITICAL();
        #endif /* USING_microCOS_II */
}

/* suspend the scheduler !!leaves interrupts enabled */
void pr_TaskSuspendAll()
{
        #ifdef USING_FreeRTOS
                vTaskSuspendAll();
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                OSSchedLock();
        #endif /* USING_microCOS_II */
}

/* resume the scheduler */
void pr_TaskResumeAll()
{
        #ifdef USING_FreeRTOS
                xTaskResumeAll();
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                OSSchedUnlock();
        #endif /* USING_microCOS_II */
}

/* Initialize the interrupt controller and Initialize the timers  uC/OS-II*/
void pr_BSP_InitIO()
{
        #ifdef USING_microCOS_II
                if (OS_TASK_STAT_EN > 0)
                {
                        BSP_InitIO();
```

```
                        OSStatInit();
                }
                else
                        BSP_InitIO();
        #endif /* USING_microCOS_II */
}

/*Disable all interrupts at the interrupt controller / uC/OS-II /*/
void pr_BSP_IntDisAll()
{
        #ifdef USING_microCOS_II
                BSP_IntDisAll();
        #endif /* USING_microCOS_II */
}

/*
*********************************************************************************************************
*                                            TASKS FUNCTIONS
*********************************************************************************************************
*/
/*pr_TaskCreate() - creates a task;
* input: (void*)pr_taskcode - pointer to the task void;
*        (const prInt8 *)pr_Name - task name;
*        (prInt16U)prStackDepth - stack depth;
*        (void*)pr_Parameters - pointer to the data, which is used as input parameters for the task;
*        (prInt32U)prPriority - priority of the task;
*        (prSTK*)pr_STK - pointer to the tasks's top-of-stack;
*        (prTaskHandle*)pr_THandle - pointer to the task handle;
* output: error message;
*/
prErrorMessage pr_TaskCreate(void *pr_taskcode, const prInt8* pr_Name, const prInt16U prStackDepth,
void *pr_Parameters, prInt32U prPriority,  prTaskHandle* prTHandle,  prSTK *  pr_STK)
{
 #ifdef USING_FreeRTOS
  BaseType_t error = 0;
  error = xTaskGenericCreate(pr_taskcode, pr_Name, prStackDepth, pr_Parameters, prPriority, prTHandle,
(StackType_t * const)pr_STK, NULL);
  if(error == pdPASS)
        return prOK;
  else
        return prERROR;
 #endif /* USING_FreeRTOS */

 #ifdef USING_microCOS_II

  prInt8U error = 0;
  prInt16U id;
  id = pr_SetTaskHandle((prInt16U*)prTHandle, prPriority);
  if (OS_STK_GROWTH == 1)      /* Stack grows from HIGH to LOW memory on Xilinx Microblaze */
  {
        error = OSTaskCreateExt((void *)pr_taskcode, (void *)pr_Parameters,
       (prInt32U*)&pr_STK[TASK_STK_SIZE - 1], (prInt8U)prPriority, (prInt16U)id,
       (prInt32U*)&pr_STK[TASK_STK_SIZE - prStackDepth], (const prInt32U)prStackDepth, (void *)0,
       OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
        if (OS_TASK_NAME_EN == 1)
                OSTaskNameSet (prPriority,  (prInt8U*)pr_Name,  NULL);
                if(error == OS_ERR_NONE)
                {
                        return prOK;
                }
                else
                        return prERROR;
  }
  else    /* Stack grows from LOW to HIGH memory on Xilinx Microblaze */
  {
        error = OSTaskCreateExt(pr_taskcode, pr_Parameters, &pr_STK[TASK_STK_SIZE - prStackDepth],
       (prInt8U)prPriority, id, &pr_STK[TASK_STK_SIZE], prStackDepth, NULL, OS_TASK_OPT_STK_CHK |
       OS_TASK_OPT_STK_CLR);
        if (OS_TASK_NAME_EN == 1)
                OSTaskNameSet (prPriority,  (prInt8U *)pr_Name,  NULL);
        if(error == OS_ERR_NONE)
                return prOK;
        else
                return prERROR;
```

```
  }
 #endif /* USING_microCOS_II */
 return prCONF_ERROR;
}


/* pr_TaskDelete() - deletes a task; input:task handle (prITSELF - itself)
* !FreeRTOSConfig.h: INCLUDE_vTaskDelete == 1 ; !!os_cfg.h: OS_TASK_DEL_EN == 1; !!os_cfg.h:
*OS_TASK_CHANGE_PRIO_EN = 1
*/
void pr_TaskDelete(prTaskHandle prTHandle)
{
          #ifdef USING_FreeRTOS
                    if (INCLUDE_vTaskDelete == 1)
                    {
                              if(prTHandle == prITSELF)
                                        vTaskDelete(NULL);
                              else
                                        vTaskDelete(prTHandle);
                    }
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prInt32U handle = (prInt32U)prTHandle;
                    if (OS_TASK_DEL_EN == 1)
                    {
                              if(prTHandle == prITSELF)
                                        OSTaskDel((INT8U)OS_PRIO_SELF);
                              else
                                        OSTaskDel((prInt8U)handle);
                    }

          #endif /* USING_microCOS_II */

}

/* pr_TaskPrioritySet() - changes priority; input: task handle (prITSELF - itself); new priority;
 * !FreeRTOSConfig.h: INCLUDE_vTaskPrioritySet = 1
 */
prErrorMessage pr_TaskPrioritySet(prTaskHandle* prTHandle, prInt32U prNewPriority)
{
          #ifdef USING_FreeRTOS
                    if (INCLUDE_vTaskPrioritySet == 1)
                    {
                              if(prTHandle == prITSELF)
                                        vTaskPrioritySet(NULL, prNewPriority);
                              else
                                        vTaskPrioritySet(prTHandle, prNewPriority);
                              return prOK;
                    }
                    else
                              return prCONF_ERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    if (OS_TASK_CHANGE_PRIO_EN > 0u)
                    {
                              prInt8U error = 0;
                              error = OSTaskChangePrio((prInt8U)*prTHandle, (prInt8U)prNewPriority);
                              if(error == OS_ERR_NONE)
                              {
                                        *prTHandle = prNewPriority;
                                         return prOK;
                              }
                              else
                                        return prERROR;
                    }
                    else
                              return prCONF_ERROR;
          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}

/*pr_TaskResume() - changes Suspended state of the task to Ready; input: task handle;
```

```
 * !FreeRTOSConfig.h: INCLUDE_vTaskSuspend = 1; !!os_cfg.h: OS_TASK_SUSPEND_EN = 1
 */
prErrorMessage pr_TaskResume(prTaskHandle prTHandle)
{
	#ifdef USING_FreeRTOS
		if (INCLUDE_vTaskSuspend == 1)
		{
			vTaskResume(prTHandle);
			return prOK;
		}
		else
			return prCONF_ERROR;
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		if (OS_TASK_SUSPEND_EN > 0u)
		{
			prInt8U error = 0;
			error = OSTaskResume((OS_STK)prTHandle);
			if(error == OS_ERR_NONE)
				return prOK;
			else
				return prERROR;
		}
		else
			return prCONF_ERROR;
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;
}

/*pr_TaskSuspend() - places the task into Suspended state; input:task handle (prITSELF - itself);
* !!leaves interrupts enabled!FreeRTOSConfig.h: INCLUDE_vTaskSuspend = 1;!!os_cfg.h:
*OS_TASK_SUSPEND_EN = 1
*/
prErrorMessage pr_TaskSuspend(prTaskHandle prTHandle)
{
	#ifdef USING_FreeRTOS
		if (INCLUDE_vTaskSuspend == 1)
		{
			if(prTHandle == prITSELF)
				vTaskSuspend(NULL);
			else
				vTaskSuspend(prTHandle);
			return prOK;
		}
		else
			return prCONF_ERROR;
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		if (OS_TASK_SUSPEND_EN > 0u)
		{
			prInt8U error = 0;
			if(prTHandle == prITSELF)
				error = OSTaskSuspend(OS_PRIO_SELF);
			else
				error = OSTaskSuspend((OS_STK)prTHandle);
			if(error == OS_ERR_NONE)
				return prOK;
			else
				return prERROR;
		}
		else
			return prCONF_ERROR;
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                          TIME FUNCTIONS
*********************************************************************************************************
*/
/*pr_TaskDelay() - locks the task for a fix number ticks;
```

```
* input: number of the clock ticks;
* !FreeRTOSConfig.h:INCLUDE_vTaskDelay = 1;
* Not periodically. It depends on the time of calling this function
 */
prErrorMessage pr_TaskDelay(prTickType prTimeTowait)
{
	#ifdef USING_FreeRTOS
		if (INCLUDE_vTaskDelay == 1)
		{
			vTaskDelay(prTimeTowait);
			return prOK;
		}
		else
			return prCONF_ERROR;
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		OSTimeDly(prTimeTowait);
		return prOK;
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;
}

/*pr_TaskGetTickCount() -  returns the number of ticks of hw timer; output: tick count (prTickType);
*  !!os_cfg.h: OS_TIME_GET_SET_EN = 1
*/
prTickType pr_TaskGetTickCount()
{
	#ifdef USING_FreeRTOS
		return xTaskGetTickCount();
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		if (OS_TIME_GET_SET_EN > 0u)
			return OSTimeGet();
		else
		    return prCONF_ERROR;
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;
}

/*pr_TaskDelayUntil() - unlocks the task after a fix number ticks
* input:
*	time of the last execution - defined by itself, must be set once as the current time
*	number of ticks - absolute period
* !FreeRTOSConfig.h:INCLUDE_vTaskDelayUntil = 1;!!os_cfg.h: OS_TIME_GET_SET_EN = 1
* Periodically.
*/
void pr_TaskDelayUntil(prTickType* const prPreWakeTime, prTickType prWakeTime)
{
	#ifdef USING_FreeRTOS
		if ( INCLUDE_vTaskDelayUntil == 1 )
		{
			vTaskDelayUntil((TickType_t * const)prPreWakeTime, (const
TickType_t)prWakeTime);

		}

	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		if (OS_TIME_GET_SET_EN > 0u)
		{
			OSTimeUntilDly((INT32U * const)prPreWakeTime, prWakeTime);

		}


	#endif /* USING_microCOS_II */

}

/*
*********************************************************************************************************
```

```
*                                               QUEUE FUNCTIONS
*********************************************************************************************
*/
/* pr_QueueCreate() - creates a queue;
* input:
*  (prInt16U)prQueueLength - max number of item in the queue;
*          pr_Queue - pointer to the queue's top-of-stack;
*          prSizeElement - size of element in the queue;
* output:
*          (prQueueHandle)prQHandle - queue handle;
*/
prQueueHandle pr_QueueCreate(prInt16U prLengteQueue, void *prStart, prInt32U prSizeElement)
{
          #ifdef USING_FreeRTOS
                    return xQueueGenericCreate( prLengteQueue, prSizeElement, queueQUEUE_TYPE_BASE);
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    return (OS_EVENT*)OSQCreate(prStart, prLengteQueue);
          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}

/* pr_QueueReceive() - reads an item from the queue;
* input:
*          (prQueueHandle)prQHandle - queue handle;
*          (void *)pr_Item - pointer to memory address for item;
*          (prTickType)prTimeOut - amount of time to wait (in clock ticks);
*          (prInt32U)prSizeItem - size of the item
* output:
*          (prErrorMessage)prError - error message: if prError = prOK - is OK
*/
prErrorMessage pr_QueueReceive(prQueueHandle prQHandle,  void*  prItem, prTickType prTimeOut, prInt32U
prSizeItem)
{
          #ifdef USING_FreeRTOS

                    prErrorMessage prError = 0;

                    prError = xQueueGenericReceive(prQHandle, prItem, prTimeOut, prFALSE);
                    if(prError == pdPASS)
                              return prOK;
                    else
                              return prERROR;

          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    void* item =NULL;
                    prInt8U prError = 0;
                    item = OSQPend (prQHandle,  prTimeOut,  &prError);
                    if(prError == OS_ERR_NONE){
                              #if (OS_CRITICAL_METHOD == 3u)
                                        OS_CPU_SR  cpu_sr = 0u;
                              #endif
                              OS_ENTER_CRITICAL();

                              (void ) memcpy( ( void * ) prItem, ( void * ) item, prSizeItem );
                              OS_EXIT_CRITICAL();
                              return prOK;
                    }
                    else
                              return prERROR;
          #endif  /*USING_microCOS_II */
          return prCONF_ERROR;
}

/*pr_QueueSend() - sends an item to the back of the queue;
* input:
*          (prQueueHandle)prQHandle - queue handle;
*          (void *)pr_Item - pointer to the memory address of data;
*  output is error message, if prError = prOK - is OK
*  !!os_cfg.h: OS_Q_POST_EN = 1
*/
```

```
prErrorMessage pr_QueueSend(prQueueHandle prQHandle,  void *  prItem)
{
          #ifdef USING_FreeRTOS
                    prErrorMessage prError = 0;
                    prError = xQueueGenericSend( prQHandle, prItem, prNull, queueSEND_TO_BACK);
                    if(prError == pdPASS)
                              return prOK;
                    else
                              return prERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prErrorMessage prError = 0;
                    if (OS_Q_POST_EN > 0u)
                    {
                              prError = OSQPost((OS_EVENT*)prQHandle, prItem);
                              if(prError == OS_ERR_NONE)
                                        return prOK;
                              else
                                        return prERROR;
                    }
                    else
                              return prCONF_ERROR;
          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}

/*pr_QueueReset() - empties the queue;
* input:
*  (prQueueHandle)prQHandle - queue handle;
* output:(prErrorMessage)error message
*/
prErrorMessage  pr_QueueReset(prQueueHandle prQHandle)
{
          #ifdef USING_FreeRTOS
                    prErrorMessage prError = 0;
                    prError = xQueueGenericReset(prQHandle, prTRUE);
                    if(prError == pdPASS)
                              return prOK;
                    else
                              return prERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prErrorMessage prError = 0;
                    if (OS_Q_POST_EN > 0u)
                    {
                              prError = OSQFlush((OS_EVENT*)prQHandle);
                              if(prError == OS_ERR_NONE)
                                        return prOK;
                              else
                                        return prERROR;
                    }
                    else
                              return prCONF_ERROR;
          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                          SEMAPHORE FUNCTIONS
*********************************************************************************************************
*/
/* pr_CreateSemaphore() - creates a counting semaphore;
*  input:
*  (prInt16U)prMaxCount - is the initial value for the semaphore, to specify how many resources are
*available;
*  output: handle of the semaphore;
*/
prSemaphoreHandle pr_CreateSemaphore(prInt16U prMaxCount)
{
          #ifdef USING_FreeRTOS
                    if (configUSE_COUNTING_SEMAPHORES == 1)
```

```
                        return xQueueCreateCountingSemaphore(prMaxCount, prNull);
                else
                        return prCONF_ERROR;
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                return (OS_EVENT*)OSSemCreate(prMaxCount);
        #endif /* USING_microCOS_II */
        return prCONF_ERROR;
}

/* pr_DeleteSemaphore() - deletes a semaphore;
* input:
*       (prSemaphoreHandle)prSHandle - semaphore handle;
*  output:(prErrorMessage)error message
*/
prErrorMessage      pr_DeleteSemaphore(prSemaphoreHandle prSHandle)
{
        #ifdef USING_FreeRTOS
                vQueueDelete(prSHandle);
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                prInt8U prError = 0;
                if (OS_SEM_DEL_EN > 0u)
                {
                        OSSemDel((OS_EVENT*)prSHandle, OS_DEL_ALWAYS, &prError);
                        if(prError == OS_ERR_NONE)
                                return prOK;
                        else
                                return prERROR;
                }
                else
                        return prCONF_ERROR;
        #endif /* USING_microCOS_II */
        return prCONF_ERROR;
}

/* pr_SemaphoreTake() - obtains a semaphore;
* input:
*       (prSemaphoreHandle)prSHandle - semaphore handle;
*       (prTickType)prTimeOUT - amount of time to wait - number of the clock ticks;
* output:(prErrorMessage)error message
*/
prErrorMessage pr_SemaphoreTake(prSemaphoreHandle prSHandle, prTickType prTimeOut)
{
        #ifdef USING_FreeRTOS
                prErrorMessage prError = 0;
                prError = xSemaphoreTake(prSHandle, prTimeOut);
                if(prError == pdPASS)
                        return prOK;
                else
                        return prERROR;
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                prInt8U prError = 0;
                OSSemPend((OS_EVENT*)prSHandle, prTimeOut, &prError);
                if(prError == OS_ERR_NONE)
                        return prOK;
                else
                        return prERROR;
        #endif /* USING_microCOS_II */
        return prCONF_ERROR;
}

/*pr_SemaphoreGive() gives the semaphore; input: semaphore handle; output: error code */
prErrorMessage      pr_SemaphoreGive(prSemaphoreHandle prSHandle)
{
        #ifdef USING_FreeRTOS
                prErrorMessage prError = 0;
                prError = xSemaphoreGive(prSHandle);
                if(prError == pdPASS)
                        return prOK;
```

```
                    else
                              return prERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prInt8U prError = 0;
                    prError = OSSemPost((OS_EVENT*)prSHandle);
                    if(prError == OS_ERR_NONE)
                              return prOK;
                    else
                              return prERROR;
          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                          MUTEX FUNCTIONS
*********************************************************************************************************
*/
/* pr_CreateMutex() - creates of a mutex;
* output:
*                   handle of mutex;
*!!FreeRTOSConfig.h: configUSE_MUTEXES = 1;
*/
prMutexHandle pr_CreateMutex()
{
          #ifdef USING_FreeRTOS
                    if ( configUSE_MUTEXES == 1 )
                              return xSemaphoreCreateMutex();
                    else
                              return prCONF_ERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prInt8U prError = 0;
                    prMutexHandle prMHandle;
                    prMHandle = OSMutexCreate(prOS_MAX_PRIO, &prError);
                    if(prError == OS_ERR_NONE)
                              return (OS_EVENT*)prMHandle;
                    else
                    {
                              prMHandle = pr_Null;
                              return prMHandle;
                    }

          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}

/* pr_DeleteMutex() - deletes a mutex;
* input:
*          (prMutexHandle)prMHandle - mutex handle;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
*/
prErrorMessage pr_DeleteMutex(prMutexHandle prMHandle)
{
          #ifdef USING_FreeRTOS
                    vQueueDelete(prMHandle);
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prInt8U prError = 0;
                    if (OS_MUTEX_DEL_EN > 0u)
                    {
                              OSMutexDel((OS_EVENT*)prMHandle, OS_DEL_ALWAYS, &prError);
                              if(prError == OS_ERR_NONE)
                                        return prOK;
                              else
                                        return prERROR;
                    }
                    else
                              return prCONF_ERROR;
          #endif /* USING_microCOS_II */
```

```
	return prCONF_ERROR;
}

/* pr_MutexTake() - takes the mutex; input: mutex handle, time to wait; output:(prErrorMessage)error
message; */
prErrorMessage pr_MutexTake(prMutexHandle prMHandle, prTickType prTimeOUT)
{
	#ifdef USING_FreeRTOS
		prErrorMessage prError = 0;
		prError = xSemaphoreTake(prMHandle, prTimeOUT);
		if(prError == pdPASS)
			return prOK;
		else
			return prERROR;
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		prInt8U prError = 0;
		OSMutexPend((OS_EVENT*)prMHandle, prTimeOUT, &prError);
		if(prError == OS_ERR_NONE)
			return prOK;
		else
			return prERROR;
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;
}

/* pr_MutexGive() -  gives the mutex; input: mutex handle; output: (prErrorMessage)error message; */
prErrorMessage pr_MutexGive(prMutexHandle prMHandle)
{
	#ifdef USING_FreeRTOS
		prErrorMessage prError = 0;
		prError = xSemaphoreGive(prMHandle);
		if(prError == pdPASS)
			return prOK;
		else
			return prERROR;
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		prInt8U prError = 0;
		prError = OSMutexPost((OS_EVENT*)prMHandle);
		if(prError == OS_ERR_NONE)
			return prOK;
		else
			return prERROR;
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                            FUNCTIONS FOR ISR
*********************************************************************************************************
*/

/*pr_QueueReceiveFromISR() receives item from the queue, called from ISR routine;
* the task that is waiting for an item in the queue will be unblocked if its priority is higher or
*equal to the interrupted task;
* input:
*	queue handle;
*	pointer to the memory address for data;
*	(prInt32U)prSizeItem - size of the item
*  output:(prErrorMessage)error message
*/
prErrorMessage pr_QueueReceiveFromISR(prQueueHandle prQHandle, void* prItem, prInt32U prSizeItem)
{
  #ifdef USING_FreeRTOS
   prErrorMessage prError = 0;
   prInt32 prPriorityTaskWoken = prTRUE;
   prError = xQueueReceiveFromISR(prQHandle, prItem, (BaseType_t * const) &prPriorityTaskWoken);
   if(prError == pdPASS)
	return prOK;
   else
```

```
        return prERROR;
  #endif /* USING_FreeRTOS */

  #ifdef USING_microCOS_II
   prInt8U prError = 0;
   if((OS_Q_ACCEPT_EN > 0u)&&(OS_Q_EN > 0u) && (OS_MAX_QS > 0u))
   {
        void* item =NULL;
        item = OSQAccept((OS_EVENT*)prQHandle, &prError);
        if(prError == OS_ERR_NONE)
        {
                #if (OS_CRITICAL_METHOD == 3u)
                        OS_CPU_SR  cpu_sr = 0u;
                #endif
                OS_ENTER_CRITICAL();
                (void ) memcpy( ( void * ) prItem, ( void * ) item, prSizeItem );
                OS_EXIT_CRITICAL();
                return prOK;
        }

        else
                return prERROR;
        }
   else
       return prCONF_ERROR;
  #endif /* USING_microCOS_II */
  return prCONF_ERROR;
}

/* pr_QueueSendFromISR() - sends item to the back of the queue from ISR routine;
* the task that is waiting for an item in the queue will be unblocked if its priority is higher or
*equal to the interrupted task;
* input:
*         (prQueueHandle)prQHandle - queue handle;
*         (void*)pr_Item - pointer to data;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
*/
prErrorMessage pr_QueueSendFromISR(prQueueHandle prQHandle, void* prItem)
{
        #ifdef USING_FreeRTOS
                prErrorMessage prError = 0;
                prInt32 prPriorityTaskWoken = prTRUE;
                prError = xQueueGenericSendFromISR(prQHandle, prItem, (BaseType_t * const)
&prPriorityTaskWoken, queueSEND_TO_BACK);
                if(prError == pdPASS)
                        return prOK;
                else
                        return prERROR;
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                prErrorMessage prError = 0;
                if(OS_Q_ACCEPT_EN > 0u)
                {
                        prError = OSQPost((OS_EVENT*)prQHandle, prItem);
                        if(prError == OS_ERR_NONE)
                                return prOK;
                        else
                                return prERROR;
        }
        else
                return prCONF_ERROR;
        #endif /* USING_microCOS_II */
                return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                          SOFTWARE TIMER
*********************************************************************************************************
*/
/*pr_TimerCreate() this function creates the software timer;
* input:
*  (prInt8 *)prName -   Is a  string that allows you to give a name to your timer;
```

```
*  (prTickType)prPeriod - Specifies the amount of time it will take before the timer expires.
*  const prInt8U prMode - prOS_TMR_OPT_PERIODIC: defines type of the timer (periodically)
*                         prOS_TMR_OPT_ONE_SHOT: defines type of the timer (one-shot)
*  (void *)pr_Arg  -   the argument passed to the callback function when the timer expires or is
         terminated.
*  (void *)pr_CallBackFunction - specifies the address of a function (optional) that you want to
          execute when the timer expire;
* output: (prTimerHandle) handle of a timer;
*
*********************************************************************************************
*            THIS FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE FOR MicroC/OS-II
*
*                                          SOFTWARE TIMER
*    1. Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*    2. Must be defined in the code: OS_TASK_TMR_STK_SIZE and OS_TASK_TMR_PRIO
*    3. Must be defined frequency of timers update through OS_TMR_CFG_TICKS_PER_SEC in the os_cfg.h
*    4. Must be enabled hooks in the os_cfg.h
*    6. In the file os_cpu_c.c:
*          in the function void  OSTimeTickHook (void):
*                    add:
*                              OSTmrSignal();
*********************************************************************************************
*/
prTimerHandle pr_TimerCreate(prInt8* prName, prTickType prPeriod, const prInt8U prMode, void* pr_Arg,
void* pr_CallBackFunction)
{
 #ifdef USING_FreeRTOS
  if(prMode == prOS_TMR_OPT_PERIODIC)
  {
        return xTimerCreate(prName, prPeriod, prTRUE, pr_Arg,(TimerCallbackFunction_t)
        pr_CallBackFunction);
  }
  else
  {
        return xTimerCreate(prName,  prPeriod,  prFALSE,  pr_Arg, (TimerCallbackFunction_t)
        pr_CallBackFunction);
  }

 #endif /* USING_FreeRTOS */

 #ifdef USING_microCOS_II
  if (OS_TMR_EN > 0)
  {
    prTimerHandle prTmHandle;
    prInt8U prError = 0;
    if(prMode == prOS_TMR_OPT_PERIODIC)
    {
        prTmHandle = OSTmrCreate((prInt32U)0, prPeriod,  OS_TMR_OPT_PERIODIC,
        (OS_TMR_CALLBACK)pr_CallBackFunction, pr_Arg, (prInt8U*)prName, &prError);
        if(prError == OS_ERR_NONE)
          return (OS_TMR*)prTmHandle;
        else
       {
          prTmHandle =pr_Null;
          return prTmHandle;
       }
       }
          else
      {
        prTmHandle = OSTmrCreate(prPeriod, (prInt32U)0, OS_TMR_OPT_ONE_SHOT,
        (OS_TMR_CALLBACK)pr_CallBackFunction, pr_Arg, (prInt8U*)prName, &prError);
        if(prError == OS_ERR_NONE)
                  return (OS_TMR*)prTmHandle;
        else
        {
                  prTmHandle =pr_Null;
                  return prTmHandle;
        }
      }
  }
 #endif /* USING_microCOS_II */
 return prCONF_ERROR;
}
```

```
/* pr_TimerStart() - starts the software timer;
* input:
*          (prTimerHandle)prTmHandle - the handle of the timer;
*          (prTickType)prTimeOut - amount of time to wait in clock ticks;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
* !!!Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*/
prErrorMessage pr_TimerStart(prTimerHandle prTmHandle)
{
          #ifdef USING_FreeRTOS
                    prErrorMessage prError = 0;
                    prError = xTimerStart(prTmHandle, prNull);
                    if(prError == pdPASS)
                              return prOK;
                    else
                              return prERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prInt8U prError = 0;
                    if (OS_TMR_EN > 0)
                    {
                              OSTmrStart((prTimerHandle)prTmHandle, &prError);
                              if(prError == OS_ERR_NONE)
                                        return prOK;
                              else
                                        return prERROR;
                    }
          #endif /* USING_microCOS_II */
                    return prCONF_ERROR;
}

/* pr_TimerStop() - stops the software timer;
* input:
*          (prTimerHandle)prTmHandle - the handle of the timer;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
* !!!Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*/
prErrorMessage pr_TimerStop(prTimerHandle prTmHandle)
{
          #ifdef USING_FreeRTOS
                    prErrorMessage prError = 0;
                    prError = xTimerStop(prTmHandle, prNull);
                    if(prError == pdPASS)
                              return prOK;
                    else
                              return prERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prInt8U prError = 0;
                    if (OS_TMR_EN > 0)
                    {
                              OSTmrStop(prTmHandle, 0u, (void*)0, &prError);
                              if(prError == OS_ERR_NONE)
                                        return prOK;
                              else
                                        return prERROR;
                    }
          #endif /* USING_microCOS_II */
                    return prCONF_ERROR;
}

/* pr_TimerDelete() - deletes the software timer;
* input:
*          (prTimerHandle)prTmHandle - the handle of the timer;
* output:(prErrorMessage)error message;  if prError = prOK - is OK
* !!!Must be defined in the os_cfg.h: OS_TMR_EN  == 1
*/
prErrorMessage  pr_TimerDelete(prTimerHandle prTmHandle)
{
          #ifdef USING_FreeRTOS
                    prErrorMessage prError = 0;
```

```
                    prError = xTimerDelete(prTmHandle, prNull);
                    if(prError == pdPASS)
                              return prOK;
                    else
                              return prERROR;
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    prInt8U prError = 0;
                    if (OS_TMR_EN > 0)
                    {
                              OSTmrDel((OS_TMR *)prTmHandle, &prError);
                              if(prError == OS_ERR_NONE)
                                        return prOK;
                              else
                                        return prERROR;
                    }
          #endif /* USING_microCOS_II */
                    return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                          SET OF QUEUES
*********************************************************************************************************
*               THIS FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE FOR MicroC/OS-II
*
*!!!!To use the set of queues functions, should be amended in the next source code:
* 1. in the file ucos_ii.h
*         in the definition of the OS_EVENT structure:
*                   add a line:
*                             struct os_event *QueueSetContainer;
* 2. in the file os_q.c
*         in the function OS_EVENT  *OSQCreate (void    **start, INT16U    size)
*                   add a line inside the if-statement (if (pevent != (OS_EVENT *)0)):
*                             pevent->QueueSetContainer = 0u;
* 3. in the file os_q.c
*         in the function INT8U  OSQPost (OS_EVENT *pevent, void *pmsg)
*                   add a code before the last call to OS_EXIT_CRITICAL(); :
*                             if(pevent->QueueSetContainer!=0u)
*                             {
*                                       OSQPost(pevent->QueueSetContainer, pevent);
*                             }
*********************************************************************************************************
*/
/* pr_QueueCreateSet() - creates a set of queue's; input: (prInt32U)total length of the queue's;
* output:set handle;!FreeRTOSConfig.h:configUSE_QUEUE_SETS = 1; os_cfg.h: OS_Q_ACCEPT_EN  = 1 and
*OS_Q_EN = 1
*/
prQueueSetHandle pr_QueueCreateSet(prInt32U prSetQueuesLength)
{
        #ifdef USING_FreeRTOS
                if ( configUSE_QUEUE_SETS == 1 )
                        return xQueueCreateSet(prSetQueuesLength);
        #endif /* USING_FreeRTOS */

        #ifdef USING_microCOS_II
                if(OS_Q_EN > 0u)
                {
                        OS_EVENT* prHandle= 0u;
                        void* set[prSetQueuesLength];
                        prHandle = OSQCreate(&set[0], prSetQueuesLength );
                        return prHandle;
                }
          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}


/* pr_QueueCreateSet() - adds queue to the set; input: queue handle, set handle;
 * output:(prErrorMessage)error message;  if prError = prOK - is OK;
 * !FreeRTOSConfig.h:configUSE_QUEUE_SETS = 1; os_cfg.h: OS_Q_ACCEPT_EN  = 1 and OS_Q_EN = 1
 */
prErrorMessage pr_QueueAddToSet(prQueueHandle prQHandle, prQueueSetHandle prQSHandle)
```

```
{
	#ifdef USING_FreeRTOS
		prErrorMessage prError = 0;
		if ( configUSE_QUEUE_SETS == 1 )
		{
			prError =  xQueueAddToSet(prQHandle, prQSHandle);
			if(prError == pdPASS)
				return prOK;
			else
				return prERROR;
		}
	#endif /* USING_FreeRTOS */

	#ifdef USING_microCOS_II
		if((OS_Q_ACCEPT_EN > 0u)&&(OS_Q_EN > 0u))
		{
			INT32S  prReturn;
			#if OS_CRITICAL_METHOD == 3u
		OS_CPU_SR  cpu_sr = 0u;
			#endif
			OS_ENTER_CRITICAL();
			if( ( ( OS_EVENT* ) prQHandle )->QueueSetContainer != 0
			{
				prReturn = prERROR;
			}
			else
			{
			  ((OS_EVENT*)prQHandle)->QueueSetContainer = prQSHandle;
			  prReturn = prOK;
			}

			OS_EXIT_CRITICAL();

			return prReturn;
		}
	#endif /* USING_microCOS_II */
	return prCONF_ERROR;
}

/*pr_QueueSelectFromSet() - selects queue from the set; input: set handle, max time to wait;
 * output: prQueueSetMemberHandle;
 * !FreeRTOSConfig.h:configUSE_QUEUE_SETS = 1; os_cfg.h: OS_Q_ACCEPT_EN  = 1 and OS_Q_EN = 1
 */
prQueueSetMemberHandle pr_QueueSelectFromSet(prQueueSetHandle prQSHandle, prTickType prTimeOUT)
{

		#ifdef USING_FreeRTOS
			if ( configUSE_QUEUE_SETS == 1 )
				return xQueueSelectFromSet(prQSHandle, prTimeOUT);
		#endif /* USING_FreeRTOS */

		#ifdef USING_microCOS_II
			if(OS_Q_EN > 0u){
				OS_EVENT *prReturn = 0u;
				INT8U err = 0;
				prReturn = OSQPend(prQSHandle, prTimeOUT, &err);
				return prReturn;
			}
		#endif /* USING_microCOS_II */
		return prCONF_ERROR;


}


/*
*********************************************************************************************************
*                                          DIAGNOSE FUNCTIONS
*********************************************************************************************************
*********************************************************************************************************
*                                          QUEUE DIAGNOSE FUNCTION
*********************************************************************************************************
*/
```

```
/*pr_qQuery() - returns the number of free spaces at the queue; input: queue handle; output: number of
the free spaces in the queue;*/
prInt32U pr_qQuery(prQueueHandle prQHandle)
{
          #ifdef USING_FreeRTOS
                              return uxQueueSpacesAvailable(prQHandle);
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                              OS_Q_DATA prqdata;
                              prErrorMessage prError = 0;
                              prError = OSQQuery((OS_EVENT*)prQHandle, &prqdata);
                              if(prError==OS_ERR_NONE)
                              {
                                        return(prqdata.OSQSize - prqdata.OSNMsgs);
                              }
                              else
                                        return prERROR;
          #endif /* USING_microCOS_II */
          return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                          TASK DIAGNOSE FUNCTION
*********************************************************************************************************
*
* Function pr_TaskGetState(): returns  the current state of the task
* input: handle of the task;
* output: string the current state of the task;
* !!!!Must be defined in the FreeRTOSConfig.h: INCLUDE_uxTaskGetStackHighWaterMark  = 1; os_cfg.h:
*OS_TASK_QUERY_EN  == 1
*/
const prInt8* pr_TaskGetStatus(prTaskHandle prtHandle)
{
  #ifdef USING_FreeRTOS
    if ( INCLUDE_eTaskGetState == 1 )
    {
          eTaskState prDataTaskState;
          prDataTaskState = eTaskGetState(prtHandle);
          switch(prDataTaskState)
          {
                    case eRunning: return "RUNNING";
                    case eReady: return "READY";
                    case eBlocked: return "BLOCKED";
                    case eSuspended: return "SUSPENDED";
                    case eDeleted: return "DELETED";
                    default: return "Can not to determine the status of the task";
          }
    }
  #endif /* USING_FreeRTOS */

  #ifdef USING_microCOS_II
    if (OS_TASK_QUERY_EN > 0)
    {
          prInt8U prError = 0;
          OS_TCB prDataSystem;
          prInt8U prio;
          prio = pr_GetPrio(prtHandle);
          prError = OSTaskQuery(prio, &prDataSystem);
          if(prError==OS_ERR_NONE)
          {
                    switch(prDataSystem.OSTCBStat)
                    {
                              case OS_STAT_RDY: return "READY";
                              case OS_STAT_SUSPEND: return "SUSPEND";
                              case OS_STAT_Q: return "WAIT OF QUEUE";
                              case OS_STAT_MBOX: return "WAIT OF MAILBOX";
                              case OS_STAT_SEM: return "WAIT OF SEMAPHORE";
                              case OS_STAT_MUTEX: return "WAIT OF MUTEX";
                              case OS_STAT_FLAG: return "WAIT OF FLAG";
                              default: return "Can not to determine the state of the task!\n\r";
                    }
          }
```

```
          else if(prError == OS_ERR_PRIO)
          {
                    return "The task must be exist";
          }
          else
          {
                    return "Can not to get the state of the task";
          }
    }
  #endif /* USING_microCOS_II */
  return "Configuration problems!";
}

/*  Function pr_TaskGetStack():returns the size of not used stack;
* input: handle of the task;
* output: the size of not used stack;
* !!!!Must be defined in the FreeRTOSConfig.h: INCLUDE_uxTaskGetStackHighWaterMark = 1;
* os_cfg.h: OS_TASK_QUERY_EN = 1; OS_TASK_PROFILE_EN = 1; OS_TASK_CREATE_EXT_EN = 1;
*/
prInt32 pr_TaskGetStack(prTaskHandle prtHandle)
{
  #ifdef USING_FreeRTOS
    if ( INCLUDE_uxTaskGetStackHighWaterMark == 1  )
    {
          return uxTaskGetStackHighWaterMark(prtHandle);
    }
  #endif /* USING_FreeRTOS */
  #ifdef USING_microCOS_II
    if ((OS_TASK_CREATE_EXT_EN > 0u)&&(OS_TASK_PROFILE_EN > 0u))
          {
                    prInt8U prError = 0;
                    OS_TCB prDataSystem;
                    prInt8U prio;
                    prio = pr_GetPrio(prtHandle);
                    prError = OSTaskQuery(prio, &prDataSystem);
                    if(prError==OS_ERR_NONE)
                    {
                      return (prInt32U)(prDataSystem.OSTCBStkSize-(prDataSystem.OSTCBStkUsed)/4);
                    }
                    else
                    {
                      return prERROR;
                    }
          }
  #endif /* USING_microCOS_II */
return prCONF_ERROR;
}

/* Function pr_TaskGetPriority(): returns the priority of the task;
* input: handle of the task;
* output: the priority of the task;
* !!!! Must be defined in the FreeRTOSConfig.h: INCLUDE_uxTaskPriorityGet = 1; os_cfg.h:
*OS_TASK_QUERY_EN  == 1
*/
prInt8U pr_TaskGetPriority(prTaskHandle prTHandle)
{
          #ifdef USING_FreeRTOS
                    if ( INCLUDE_uxTaskPriorityGet == 1  )
                    {
                              return uxTaskPriorityGet(prtHandle);
                    }
          #endif /* USING_FreeRTOS */

          #ifdef USING_microCOS_II
                    if (OS_TASK_QUERY_EN > 0u)
                    {
                              prInt8U prError = 0;
                              OS_TCB prDataSystem;
                              prInt8U prio;
                              prio = pr_GetPrio(prtHandle);
                              prError = OSTaskQuery(prio, &prDataSystem);
                              if(prError==OS_ERR_NONE)
                              {
                                        return (prDataSystem.OSTCBPrio);
```

```
                                }
                                else
                                {
                                        return prERROR;
                                }
                        }
                #endif /* USING_microCOS_II */
                return prCONF_ERROR;
}

/* Function pr_TaskGetName(): returns the name of the task;
* input: handle of the task;
* output: the name of the task;
*!Must be defined in the FreeRTOSConfig.h: INCLUDE_pcTaskGetTaskName  = 1;os_cfg.h: OS_TASK_NAME_EN=1,
*OS_TASK_QUERY_EN  = 1
*/
const prInt8* pr_TaskGetName(prTaskHandle prtHandle)
{
                #ifdef USING_FreeRTOS
                        if ( INCLUDE_pcTaskGetTaskName == 1  )
                        {
                                return pcTaskGetTaskName(prtHandle);
                        }
                #endif /* USING_FreeRTOS */

                #ifdef USING_microCOS_II
                        if ((OS_TASK_NAME_EN  > 0)&&(OS_TASK_QUERY_EN > 0))
                        {
                                prInt8U prError = 0;
                                OS_TCB prDataSystem;
                                prInt8U prio;
                                prio = pr_GetPrio(prtHandle);
                                prError = OSTaskQuery(prio, &prDataSystem);
                                if(prError==OS_ERR_NONE)
                                {
                                        return (const prInt8*)(prDataSystem.OSTCBTaskName);
                                }
                                else
                                {
                                        return "Can not to get the name of the task";
                                }
                        }
                #endif /* USING_microCOS_II */
                        return "Configuration problems!";
}

/* Function prTaskGetState(): returns all information about the task;
* input: handle of the task;
*         pointer to the prTASK_STATE;
* output:(prErrorMessage)error message;
* !!Must be defined in the os_cfg.h: OS_TASK_QUERY_EN =1,OS_TASK_NAME_EN = 1,OS_TASK_CREATE_EXT_EN=1,
*OS_TASK_PROFILE_EN = 1
* !!Must be defined in the FreeRTOSConfig.h: INCLUDE_pcTaskGetTaskName  = 1;
*os_cfg.h:OS_TASK_NAME_EN=1, OS_TASK_QUERY_EN  = 1,
* INCLUDE_uxTaskPriorityGet = 1
*
*                THIS FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE FOR FreeRTOS
*
* FUNCTIONS IS IMPLEMENTED WITH THE CHANGE OF THE SOURCE CODE
* 1. In the file tasks.c must be created the function:
*          #if ( configGENERATE_RUN_TIME_STATS == 1)
*                  uint32_t uxTaskGetRunTime(TaskHandle_t xTaskToQuery )
*                  {
*                          vTaskSuspendAll();
*                          TCB_t *pxTCB;
*                          pxTCB = prvGetTCBFromHandle( xTaskToQuery );
*                          xTaskResumeAll();
*                          return pxTCB->ulRunTimeCounter;
*
*                  }
*                  #endif
* 1. In the file task.h must be defined the function:
*         uint32_t uxTaskGetRunTime(TaskHandle_t xTaskToQuery );
*/
```

```
prErrorMessage pr_TaskGetState(prTaskHandle prtHandle, prTASK_STATE* prtdata)
{
 #ifdef USING_FreeRTOS
  if(INCLUDE_eTaskGetState  == 1)
  {
          eTaskState tstate;
          tstate = eTaskGetState(prtHandle);
          switch(tstate)
          {
                    case eRunning: prtdata->tStatus = "RUNNING"; break;
                    case eReady: prtdata->tStatus = "READY";  break;
                    case eBlocked: prtdata->tStatus = "BLOCKED"; break;
                    case eSuspended: prtdata->tStatus = "SUSPENDED"; break;
                    case eDeleted: prtdata->tStatus = "DELETED"; break;
                    default: prtdata->tStatus = "Can not to determine the status of the task"; break;
          }
  }
  else
  {
          return prCONF_ERROR;
  }
  if(INCLUDE_pcTaskGetTaskName  == 1)
  {
          prtdata->tName = pcTaskGetTaskName(prtHandle);
  }
  else
  {
          return prCONF_ERROR;;
  }
  if(INCLUDE_uxTaskGetStackHighWaterMark  == 1)
  {
          prtdata->tSTKsize = uxTaskGetStackHighWaterMark(prtHandle);
  }
  else
  {
           return prCONF_ERROR;
  }
  if ( INCLUDE_uxTaskPriorityGet == 1)
  {
          prtdata->tPrio = uxTaskPriorityGet(prtHandle);
  }
  else
  {
          return prCONF_ERROR;
  }
  if ( configGENERATE_RUN_TIME_STATS == 1)
  {
          prtdata->tRunTime = uxTaskGetRunTime(prtHandle);
  }
  else
  {
          return prCONF_ERROR;
  }

  return prOK;
#endif /* USING_FreeRTOS */

#ifdef USING_microCOS_II
  if (OS_TASK_QUERY_EN > 0)
  {
          prInt8U prError = 0;
          OS_TCB prDataTask;
          prInt8U prio;
          prio = pr_GetPrio(prtHandle);
          prError = OSTaskQuery(prio, &prDataTask);
          if(prError==OS_ERR_NONE)
          {
                    if (OS_TASK_NAME_EN == 1)
                    {
                              prtdata->tName = (prInt8*)prDataTask.OSTCBTaskName;
                    }
                    else
                    {
                              return prCONF_ERROR;
```

```
                    }

                    switch(prDataTask.OSTCBStat)
                    {
                        case OS_STAT_RDY: prtdata->tStatus = "READY"; break;
                        case OS_STAT_SUSPEND:  prtdata->tStatus = "SUSPEND"; break;
                        case OS_STAT_Q:  prtdata->tStatus = "WAIT OF QUEUE"; break;
                        case OS_STAT_MBOX:  prtdata->tStatus = "WAIT OF MAILBOX"; break;
                        case OS_STAT_MUTEX:  prtdata->tStatus = "WAIT OF MUTEX"; break;
                        case OS_STAT_FLAG:  prtdata->tStatus = "WAIT OF FLAG"; break;
                        default:  prtdata->tStatus = "Can not to determine the status of the task";
                        break;
                    }
                    prtdata->tPrio = prDataTask.OSTCBPrio;
                    if ((OS_TASK_CREATE_EXT_EN > 0u)&&(OS_TASK_PROFILE_EN > 0u))
                    {
                        prtdata->tSTKsize = (prDataTask.OSTCBStkSize - (prDataTask.OSTCBStkUsed)/4);
                    }
                    else
                    {
                        return prCONF_ERROR;
                    }

                    prtdata->tRunTime = prDataTask.OSTCBCtxSwCtr;
            }
            else
            {
                    return prERROR;
            }
            return prOK;
    }
  #endif /* USING_microCOS_II */
  return prCONF_ERROR;
}

/*
*********************************************************************************************************
*                                          GET CONFIGURATION SETTINGS FUNCTION
*********************************************************************************************************
*/
/*prGetConfTICK_RATE() - returns the number millisecond per one ticks; is used to convert milliseconds
to ticks and vice versa*/
prTickType pr_GetTICK_RATE()
{
            return ( ( prTickType ) 1000 / prConfigTICK_RATE_HZ);
}
```

Bijlage XXI. Test resultaten van de wrapper

- **Test 1.**

Opstelling:

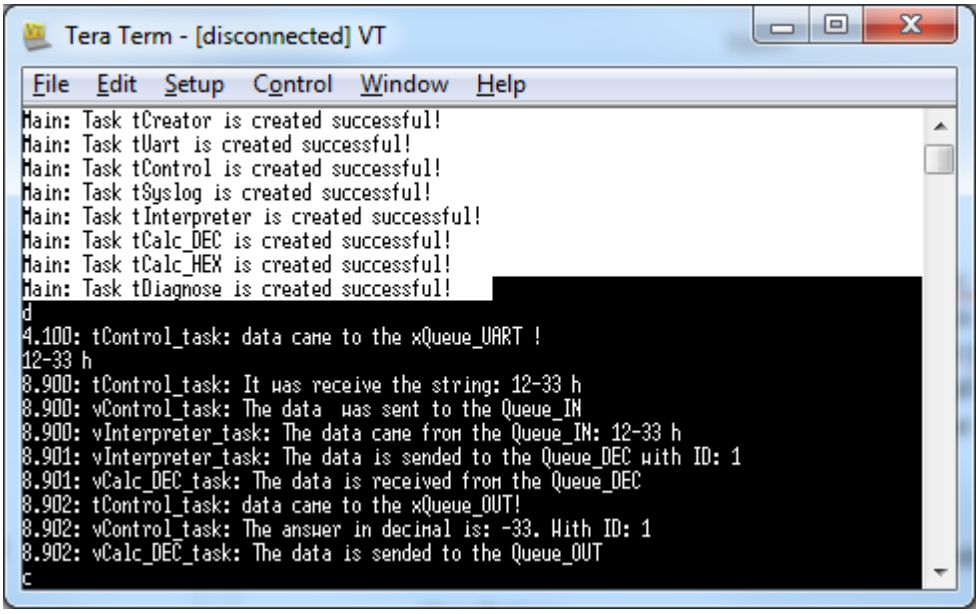
In de demoapplicatie waren acht taken gecreëerd met de functie*: pr_TaskCreate()*. Tijdens de test moet de demoapplicatie opgestart worden. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

Verwachtte resultaat:

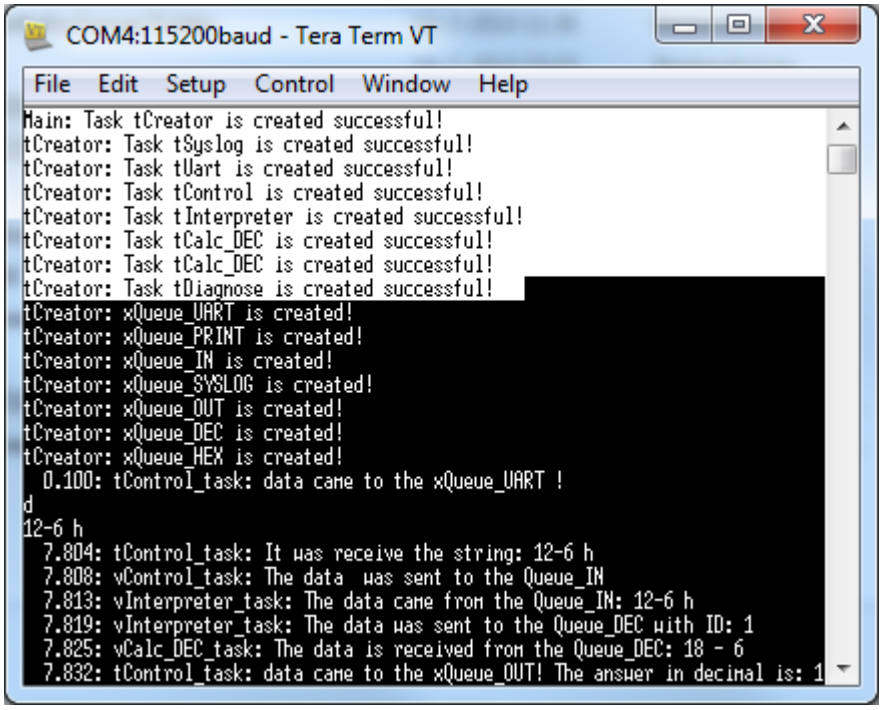
Alle taken moeten gecreëerd worden eerst met FreeRTOS en daarna met MicroC/OS-II.***Resultaten:***

1. FreeRTOS, zie Printscreen 1.1
   Correct
2. MicroC/OS-II, zie Printscreen 1.2
   Correct

Printscreen 1.1: Output van Test 1 - FreeRTOS



Printscreen 1.2: Output van Test 1 – MicroC/OS-II



- **Test 2.**

Opstelling:

In de demoapplicatie was de *Creator_taak* gecreëerd. Deze taak creëert de andere taken en daarna verwijdert zichzelf met de functie: *pr_TaskDelete().*Tijdens de test moet de demoapplicatie opgestart worden. Deze test moet worden uitgevoerd eerst onder FreeRTOS MicroC/OS-II en daarna onder MicroC/OS-II.

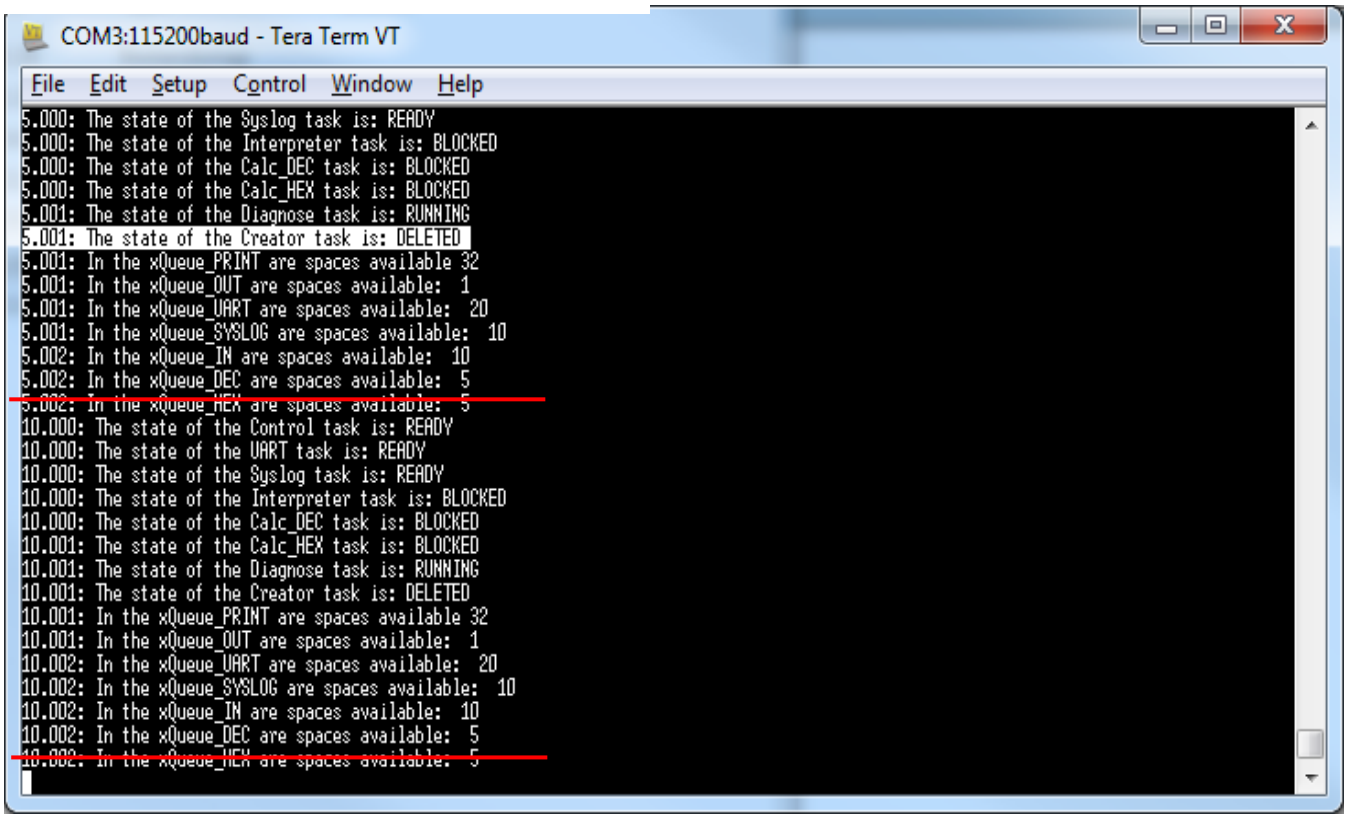
Verwachtte resultaat:

FreeRTOS: In de statetabel moet duidelijk zijn dat de Creator taak is verwijderd.
MicroC/OS-II: In de statetabel moet duidelijk zijn dat de Creator taak is niet bestaat.

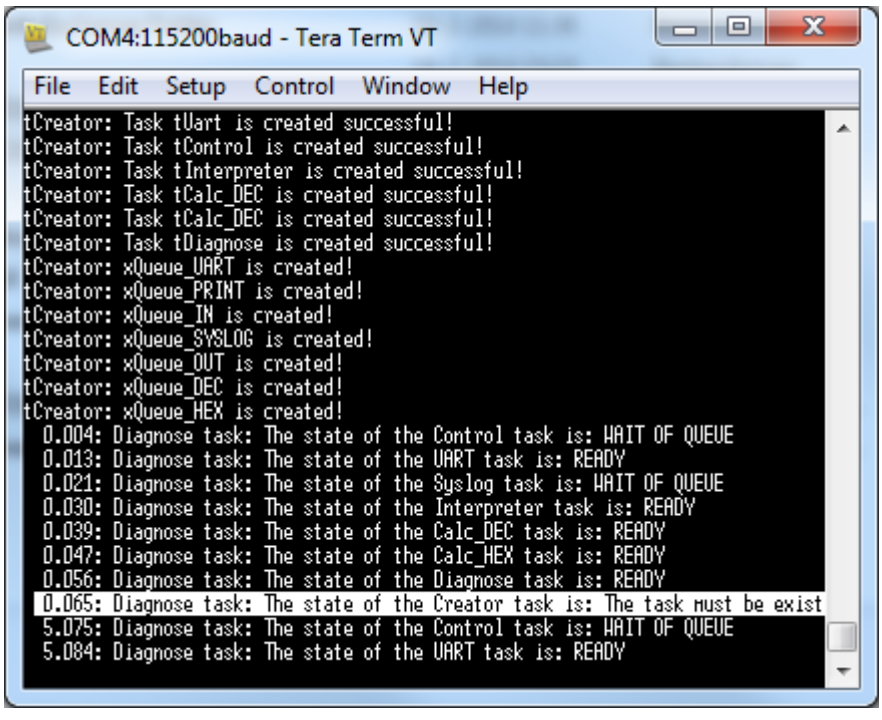
Resultaten:

1. FreeRTOS, zie Printscreen 2.1
   Correct
2. MicroC/OS-II, zie Printscreen 2.2
   Correct

Printscreen 2.1: Output van Test 2 - FreeRTOS



Print Screen 2.2: Output van Test 2 – MicroC/OS-II



- **Test 3.**

Opstelling:

Hier moet in de demoapplicatie de functie: *pr_TaskPrioritySet()* die de priority van, bijvoorbeeld, de UART taak verandert, worden toegevoegd, en daarna de statetabel moet worden uitgeprint. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II. Onder FreeRTOS wordt de priority van de UART taak van 3 naar 4 veranderd. Onder MicroC/OS-II wordt de priority van de UART taak van 10 naar 15 veranderd.

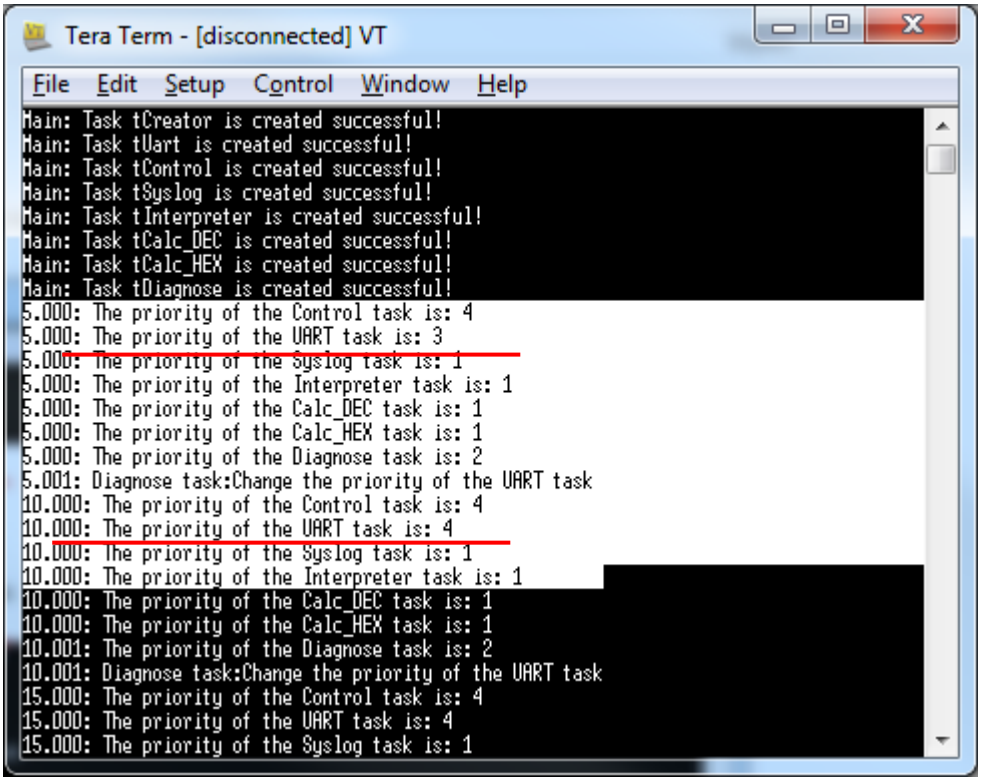
Verwachtte resultaat:

In de uitprint van de statetabel moet de priority van de UART taak veranderd worden.

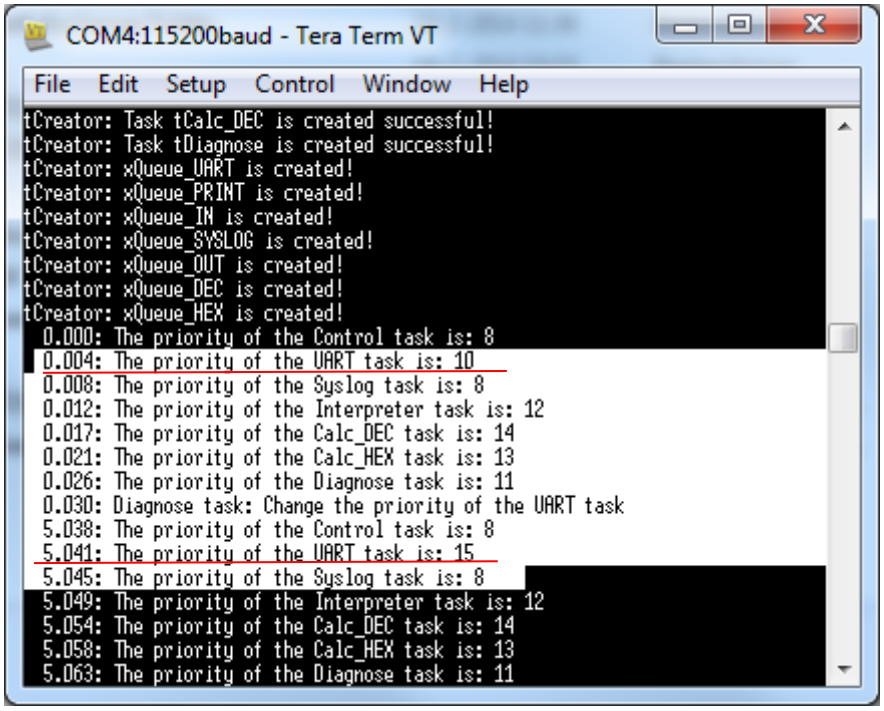
Resultaten:

1. FreeRTOS, zie Printscreen 3.1
   Correct
2. MicroC/OS-II, zie Printscreen 3.2
   Correct

Printscreen 3.1: Output van Test 3 - FreeRTOS



Printscreen 3.2: Output van Test 3 – MicroC/OS-II



- **Test 4.**

Opstelling:

Hier moet in de demoapplicatie de functie: *pr_TaskSuspend(),* worden toegevoegd die, bijvoorbeeld , de *Interpreter_taak* opschort. Daarna moet de toestand van de *Interpreter_taak* worden uitgeprint, vervolgens na het printen van de toestand wordt de *Interpreter_taak* hervat en weer de toestand van de *Interpreter_taak* wordt uitgeprint. De opschortte functie wordt in de *main* toegevoegd en de resumefunctie: *pr_TaskResume(),* wordt in de *Diagnose_taak* toegevoegd, na het uitprinten van de toestanden. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder het MicroC/OS-II.

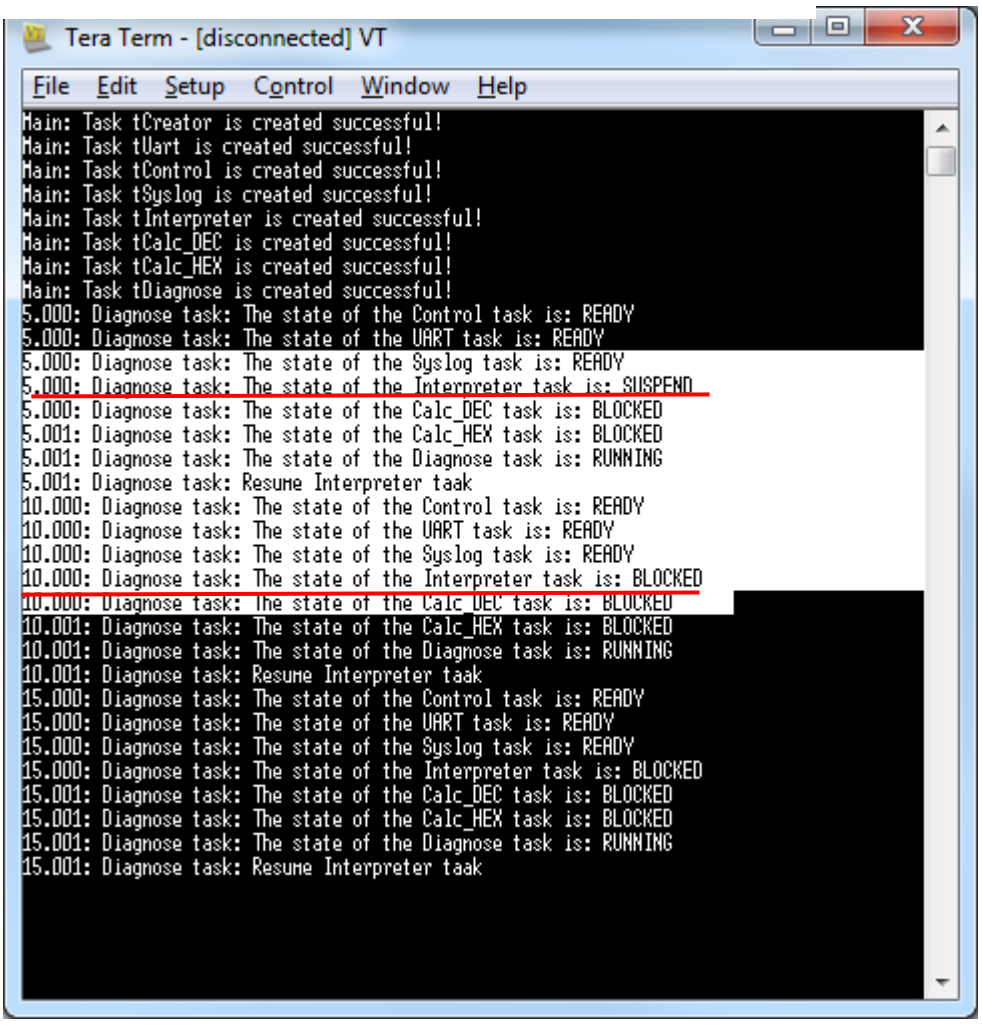
Verwachtte resultaat:

Eerst moet de toestand van de *Interpreter_taak Suspend* zijn.Na het opscorten moet de toestand van *Interpreter_taak* naar *Blocked* verandert worden.

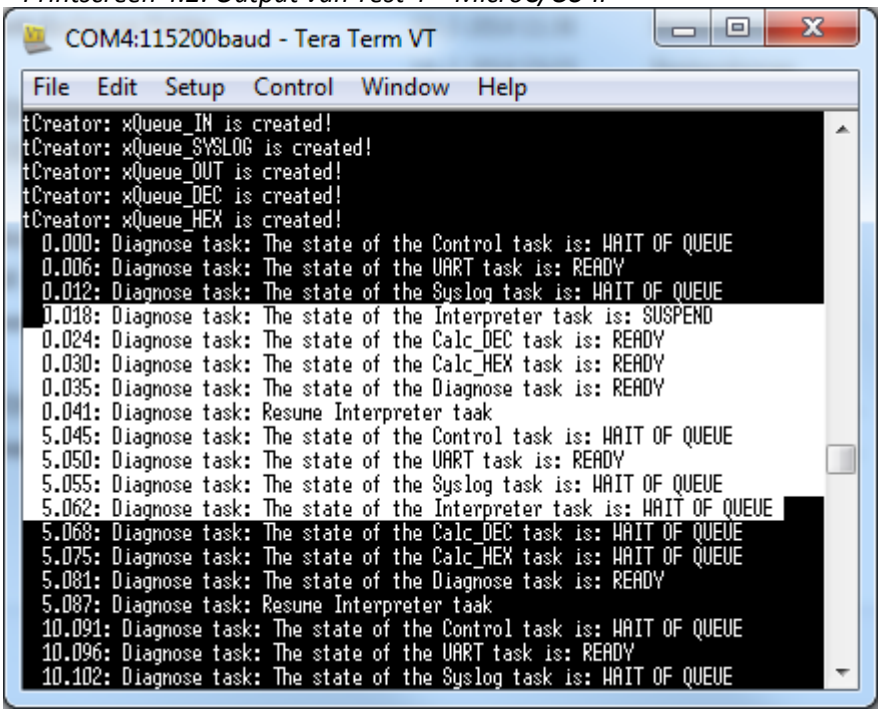
Resultaten:

1. FreeRTOS, zie Printscreen 4.1
   Correct
2. MicroC/OS-II, zie Printscreen 4.2
   Correct

Printscreen 4.1: Output van Test 4 - FreeRTOS



Printscreen 4.2: Output van Test 4 – MicroC/OS-II



- **Test 5.**

Opstelling:

De functie: *pr_TaskDelayUntil(),* is gebruikt in de *UART_taak* en *Diagnose_taak*. Hier moet naar de periodiciteit van de uitvoering van deze functies worden gekeken. Daarvoor moet in de *UART_taak* een functie worden toegevoegd die per elke uitvoering van de taak een melden moet printen, dat de taak runt. De periode van de uitvoering moet 0.02 secunde zijn. De *Diagnose_taak* moet de statetabel elke 5 secunde printen. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

Verwachtte resultaat:

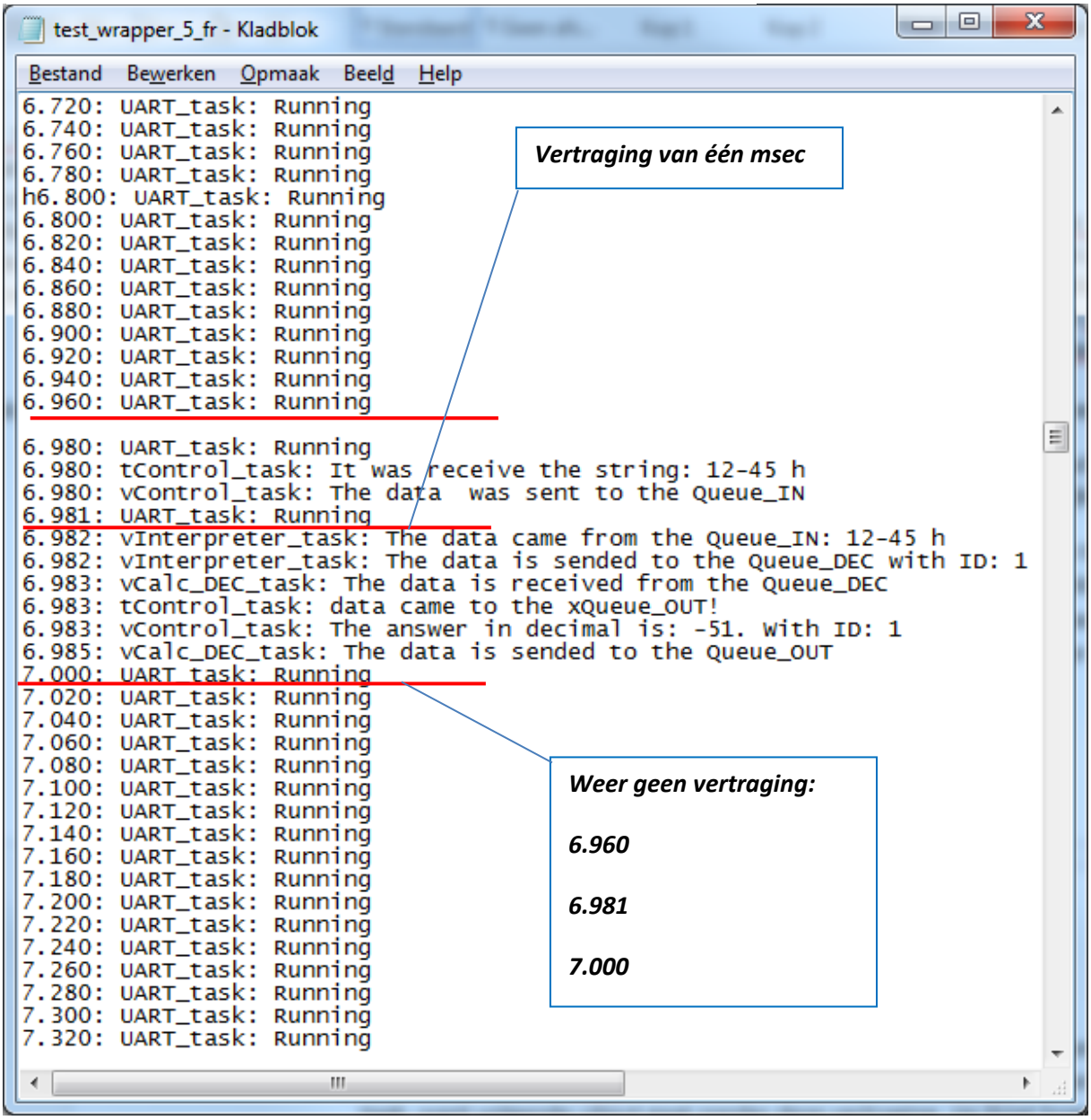
Hier is verwacht dat de uitvoering van de *UART_taak* en *Diagnose_taak* periodiek zal zijn, met de respectievelijke perioden van 0.02 en 5 secunden.

Resultaat

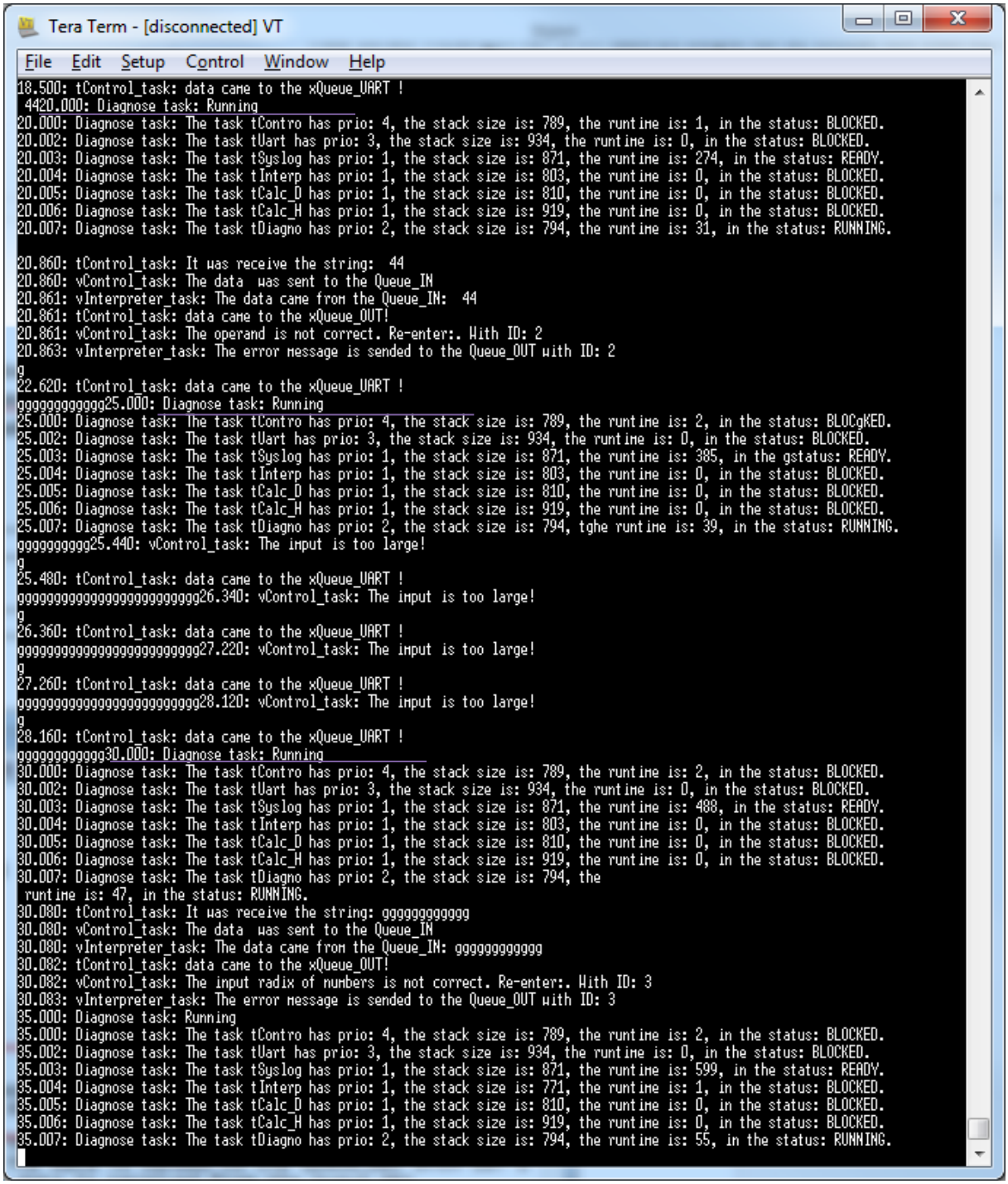
Ik heb deze test uitgevoerd met meldingen van het runnen van *UART_taak* en *Diagnose_taak*. *UART_taak* print de meldingen elke 0.02 secunde en de output staat in het bestand *test_wrapper_5_fr.txt* en *test_wrapper_5_micro.txt*. Om deze test overzichtelijke te maken voor de Diagnose taak, heb ik nog een keer deze test uitgevoerd maar zonder meldingen van *UART_taak*. De output van de tweede test staat bij Print Screen 5.1b en 5.2b. De resultaten van de testen zijn voldoende. Allebei taken voeren periodiek uit. Soms kon ik zien de vertraging bij de output van 1 msec, maar het is vertraging van de *Syslog_taak*, die is door de Mutex beveiligd, en niet van de *UART_taak*, want volgende uitput gaat zonder deze vertraging, zie Print Screen 5.1a, 5.2a.

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	test_wrapper_5_fr.txt Printscreen 5.1a,b	***V***
MicroC/OS-II	Printscreen 5.2a,b	***V***

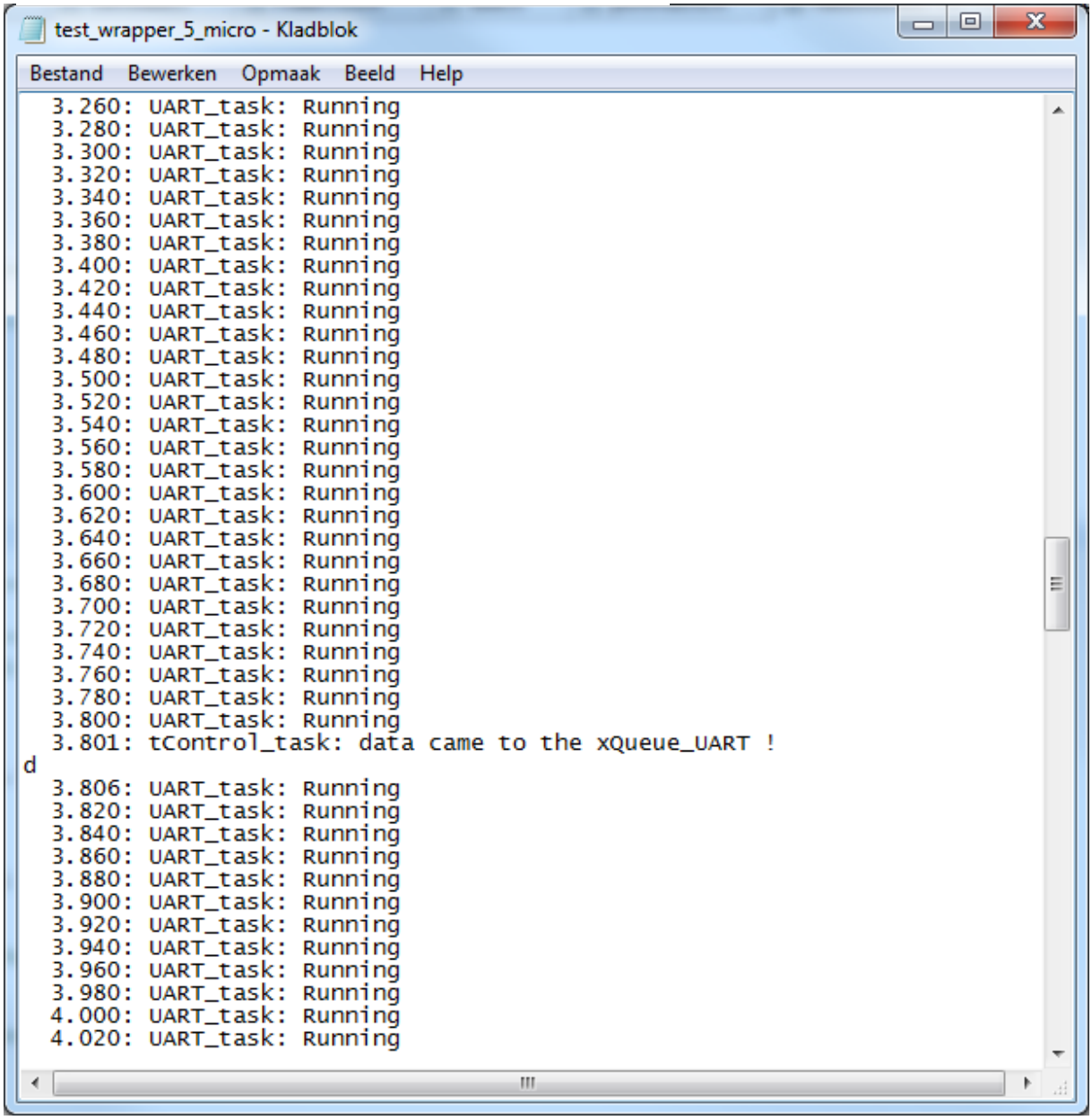
Printscreen 5.1: Output van Test 5.1a - FreeRTOS



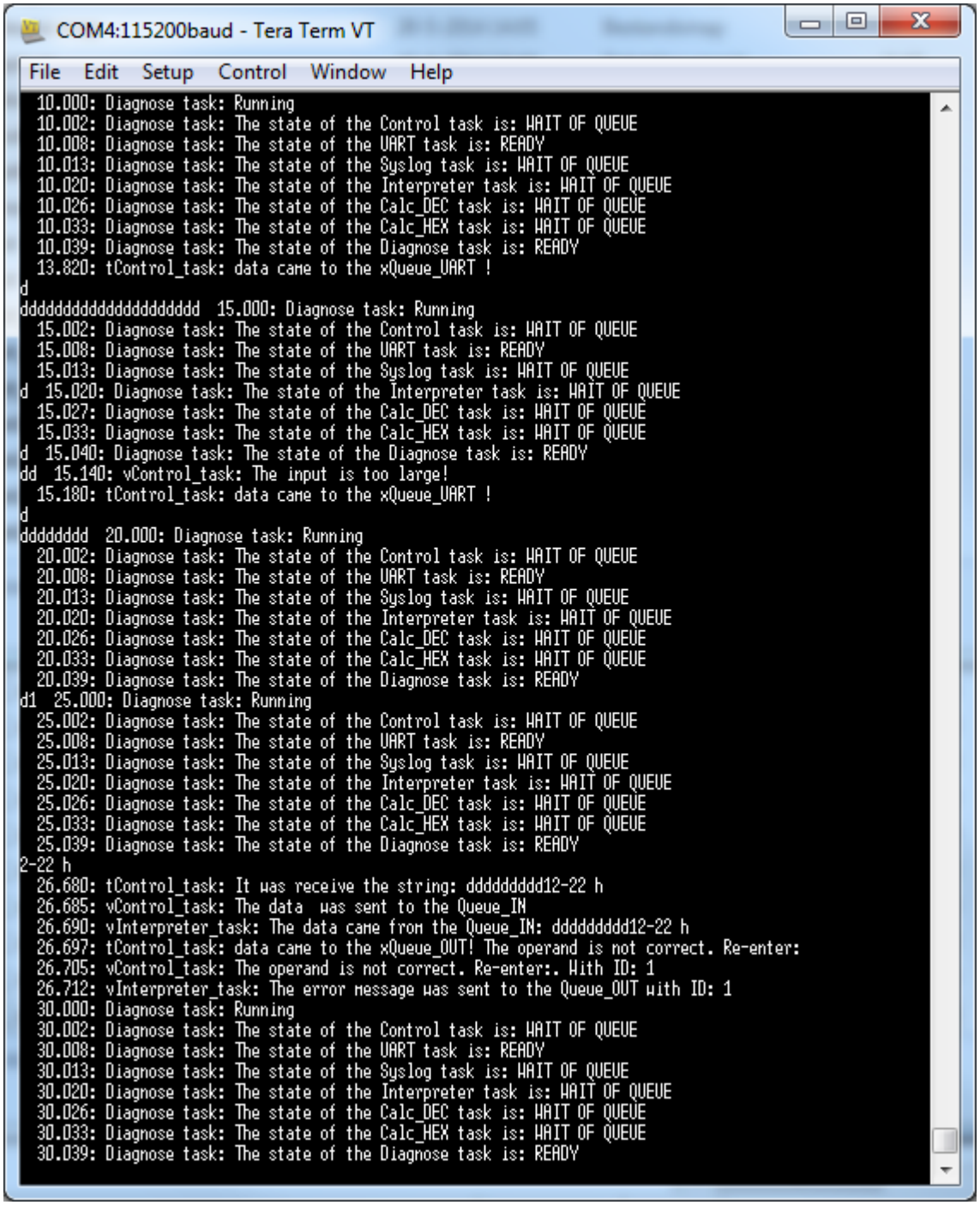
Printscreen 5.1: Output van Test 5.1b - FreeRTOS



Printscreen 5.2a: Output van Test 5 – MicroC/OS-II



Printscreen 5.2b: Output van Test 5 – MicroC/OS-II



- **Test 6.**

Opstelling:

In de *Syslog_taak* van de demoapplicatie gebruikt de softtimer om de tijd van de berichten bij te houden. Tijdens deze test moet de functie van interne timer: *pr_TaskGetTickCount(),* naast de softtimer worden aangeroepen. De tijd van de interne timer moet ook worden uitgeprint. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

Verwachtte resultaat:

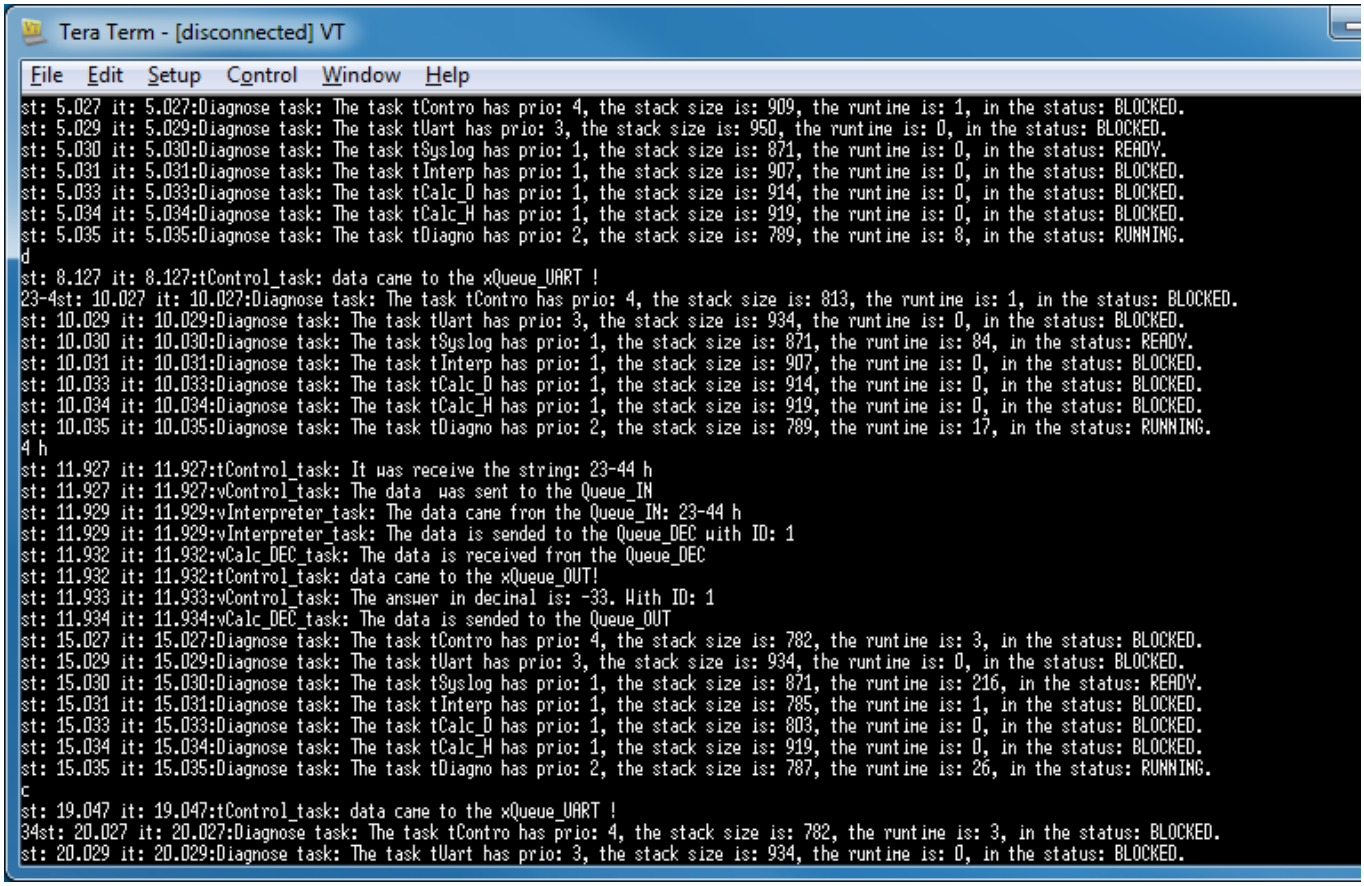
Hier is verwacht dat de tijd die door de interne timer wordt gekregen, en de tijd van de softtimer, gelijk zullen zijn.

Resultaat

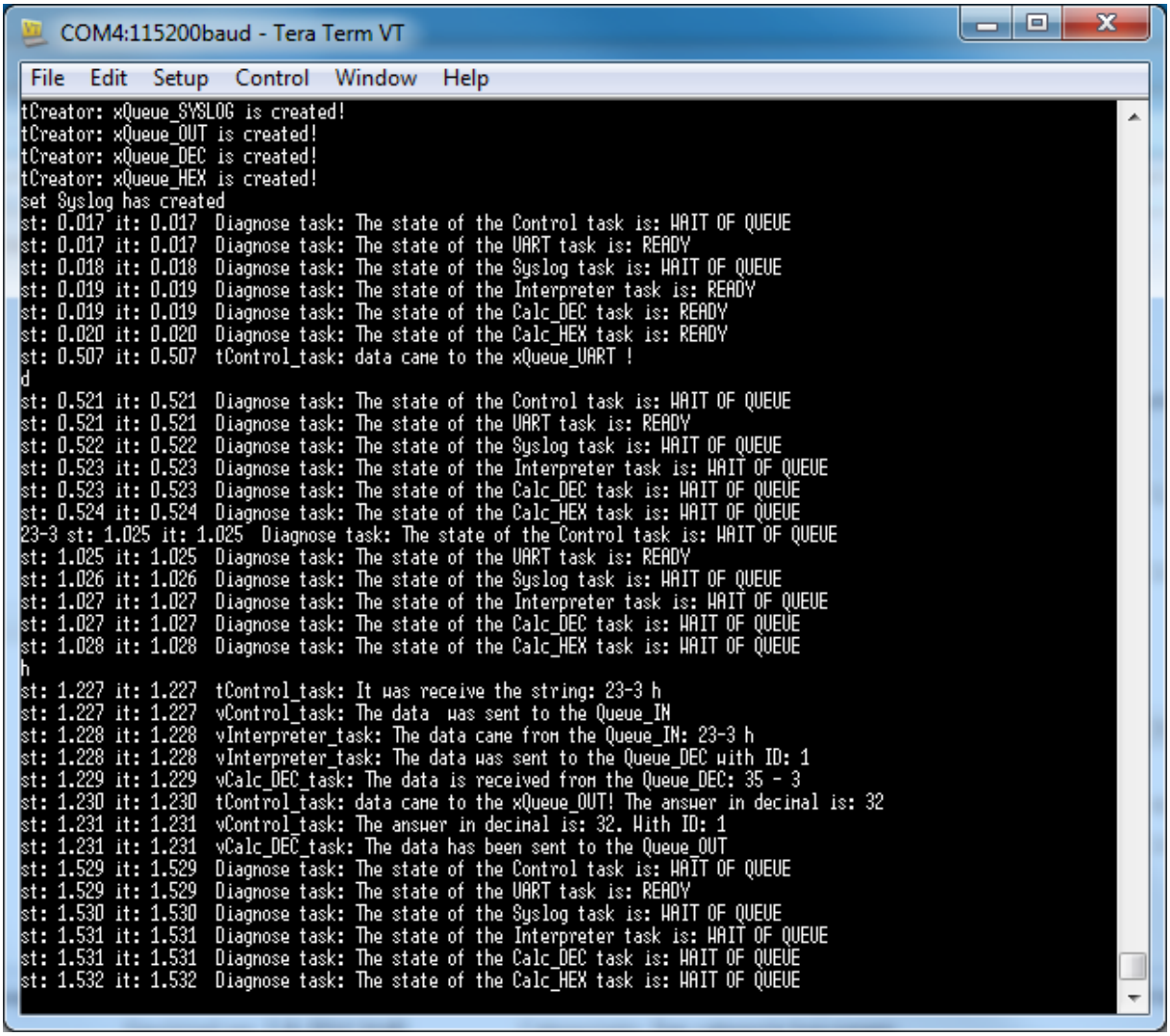
Op Printscreen 6.1 en 6.2 na *'st'* staat de tijd van de softtimer, na *'it'* staat de output van interne timer.

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 6.1	***V***
MicroC/OS-II	Printscreen 6.2	***V***

Printscreen 6.1: Output van Test 6 - FreeRTOS



Printscreen 6.2:. Output van Test 6 – MicroC/OS-II



- **Test 7.**

Opstelling:

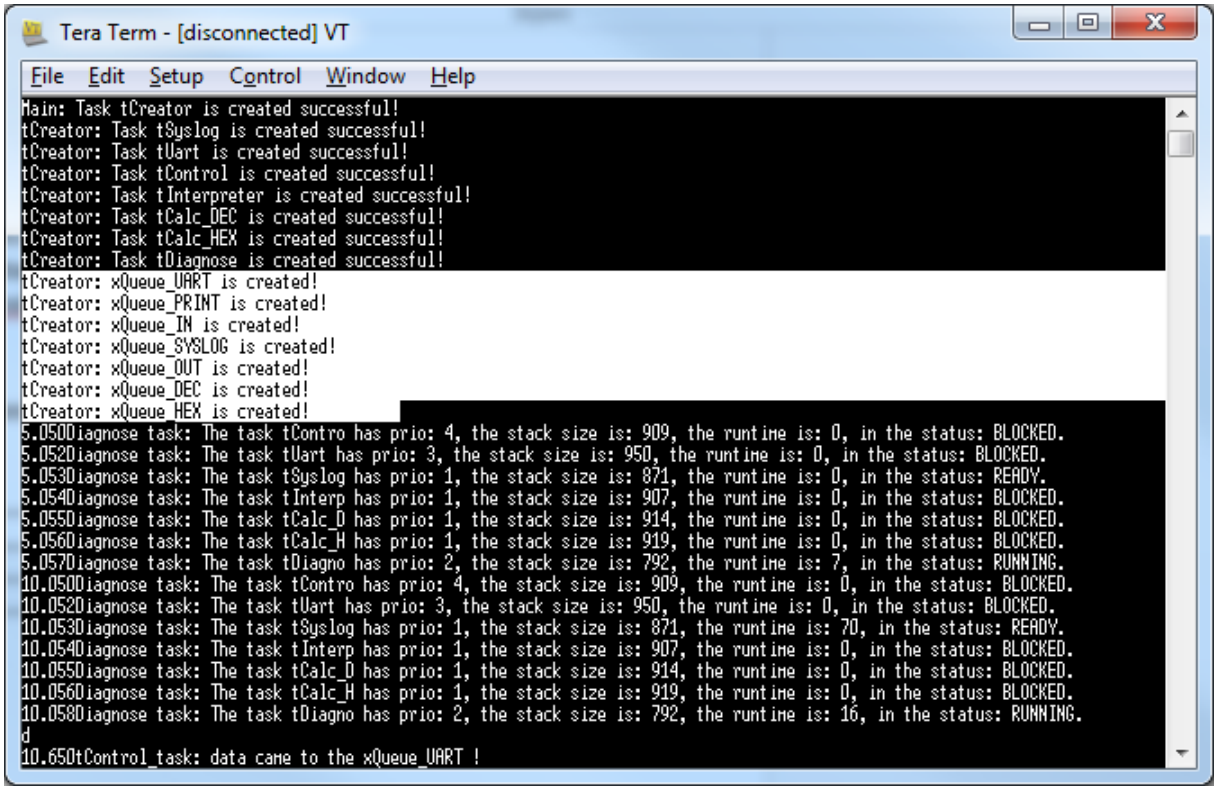
In de demoapplicatie worden zeven queues gecreëerd worden met de functie: *pr_QueueCreate()*. Tijdens de test moet de demoapplicatie opgestart worden. Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

Verwachtte resultaat:

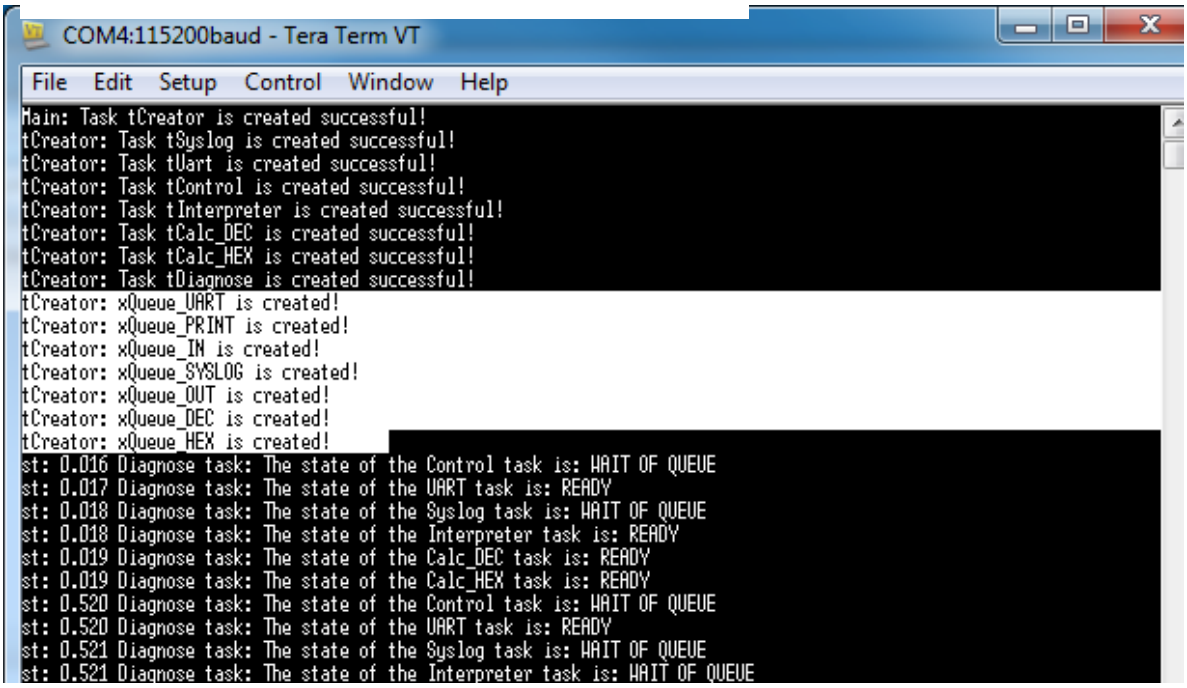
Hier is verwacht dat de melding over het succesvolle creëren van de queues wordt aangetoond.

RTOS	*Printscreen*	*Resultaat*
FreeRTOS	Printscreen 7.1	***V***
MicroC/OS-II	Printscreen 7.2	***V***

Printscreen 7.1: Output van Test 7 - FreeRTOS



Printscreen 7.2: Output van Test 7 – MicroC/OS-II



- **Test 8.**

Opstelling:

In de demoapplicatie in de taken: *Control_taak*, *Interpreter_taak*, *Syslog_taak*, *Calc_DEC_taa*k en *Calc_HEX_taak* , is de functie geïmplementeerd: *pr_QueueReceive(),* die de data uit de queues leest. Na het lezen van de queues moet het bericht uitgeprint worden. Tijdens deze test worden de verzoeken ingevoerd:
c444-5556 d
SSCi
Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

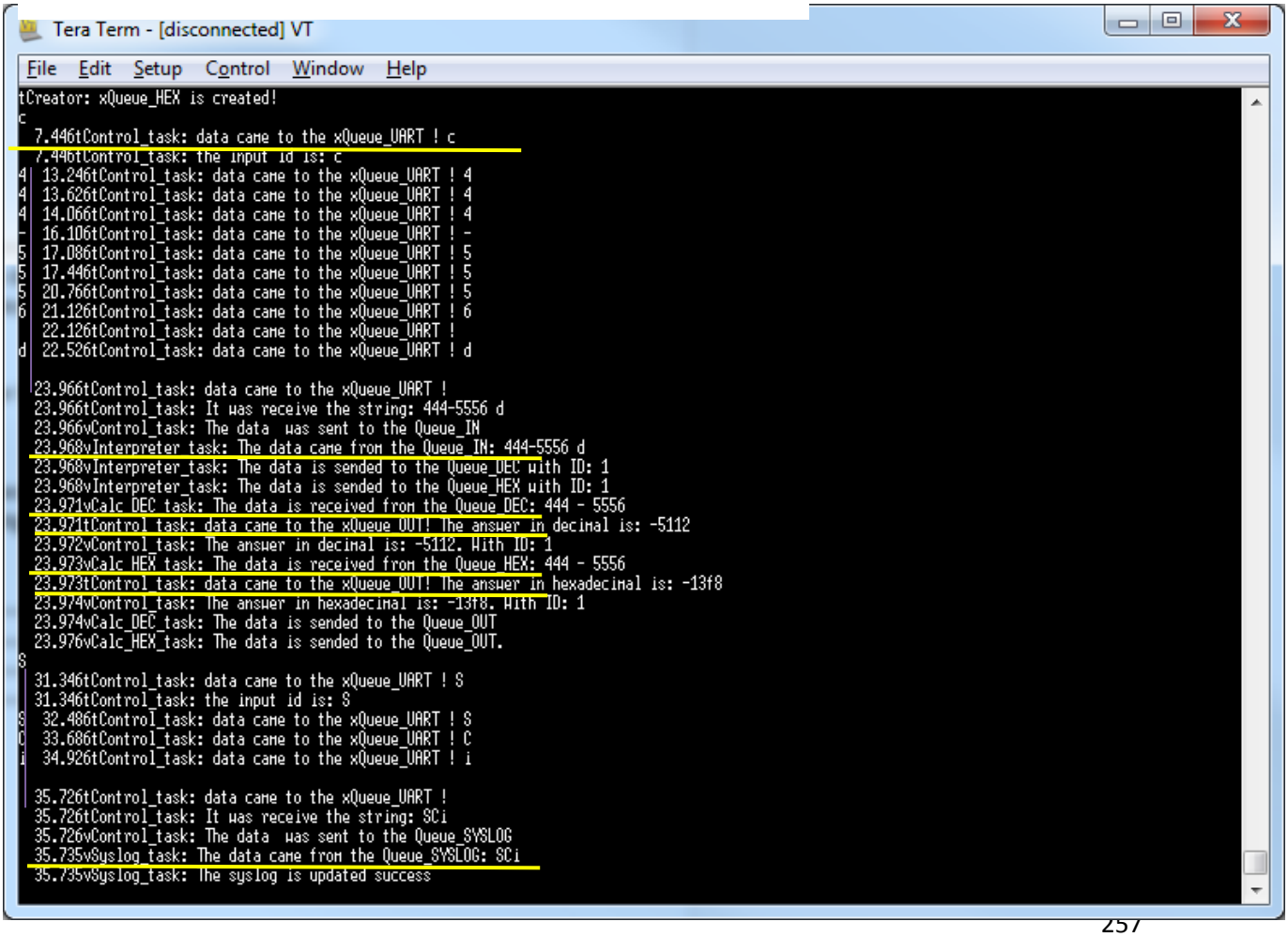
Verwachtte resultaat:

Het is verwacht dat na het invoegen van het verzoek de *Control_taak, Interpreteter_taak, Syslog_taak, Calc_DEC_taak en Calc_HEX_taak* de bericht met uitgelezen data uit de queues worden uitgeprint.

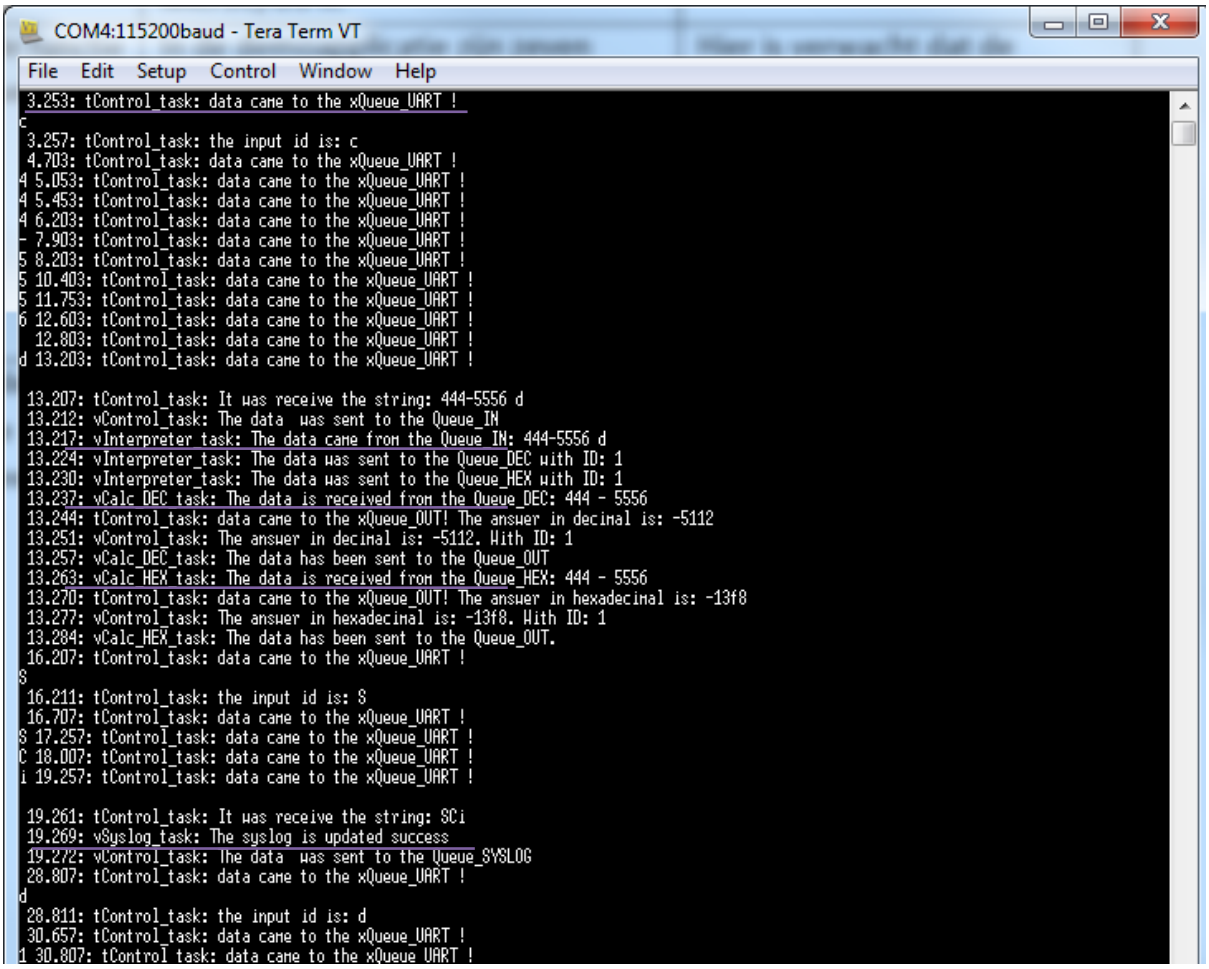
Resultaat

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	*Printscreen* 8.1	***V***
MicroC/OS-II	*Printscreen* 8.2	***V***

Printscreen 8.1: Output van Test 8 - FreeRTOS



Printscreen 8.2: Output van Test 8 – MicroC/OS-II



- **Test 9.**

Opstelling:

In de taken van demoapplicatie 2.0*: UART_taak*, *Control_taak,*, *Interpreteter_taak*, *Syslog_taak, Calc_DEC_taak en Calc_HEX_taak* is de wrapperfunctie geïmplementeerd: *pr_QueueSend(),* die een bericht naar de queue zendt. Na het zenden van de data naar de queue moet het bericht erover uitgeprint worden. Tijdens deze test worden de vogende verzoeken ingevoerd:
c444-5556 d
SSCi
Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II FreeRTOS.

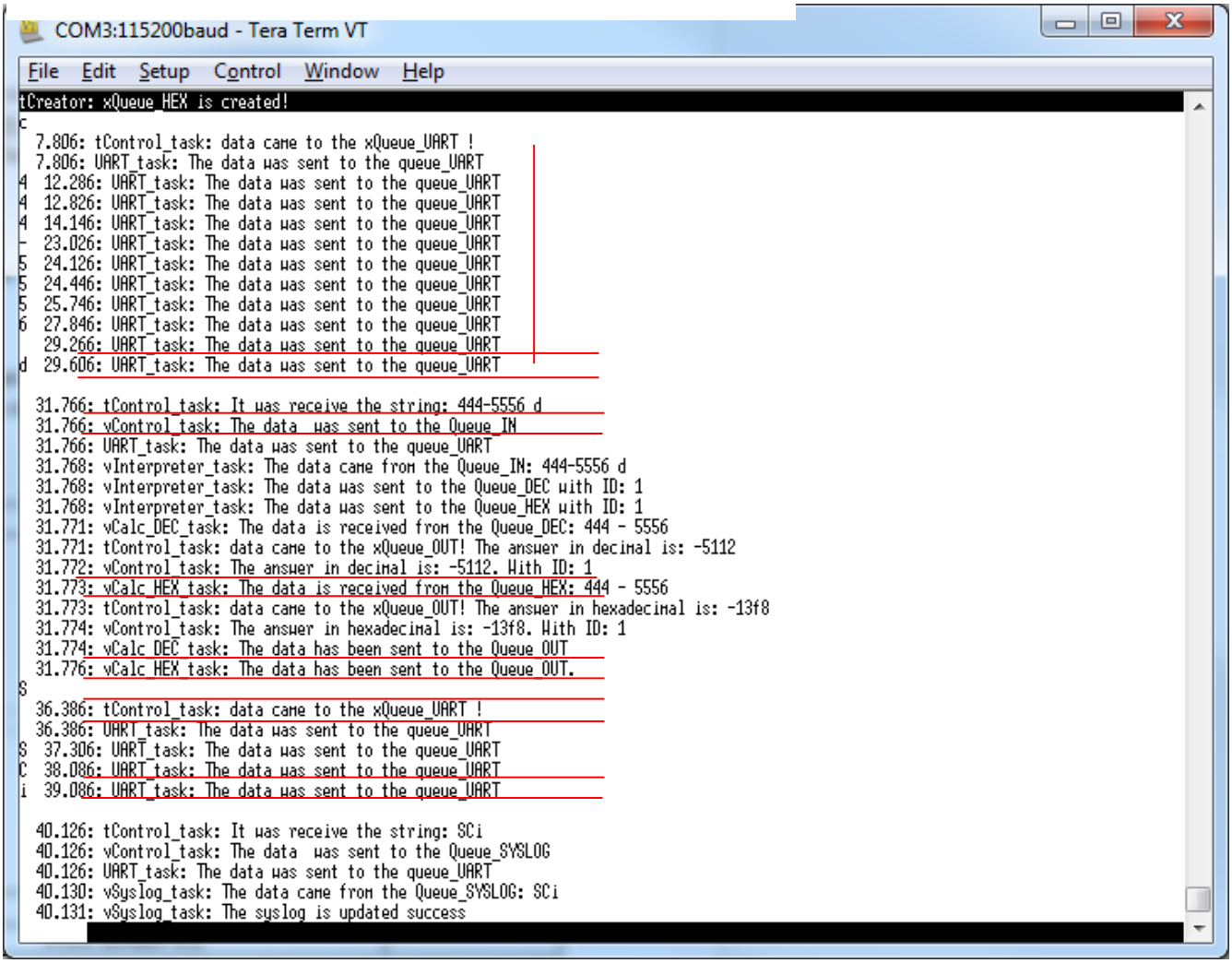
Verwachtte resultaat:

Het is verwacht dat na het invoegen van het verzoek, worden de UART_taak, *Control_taak,*, *Interpreteter_taak*, *Syslog_taak, Calc_DEC_taak en Calc_HEX_taak* de bericht dat de data naar de queue was gestuurd, uitgeprint.

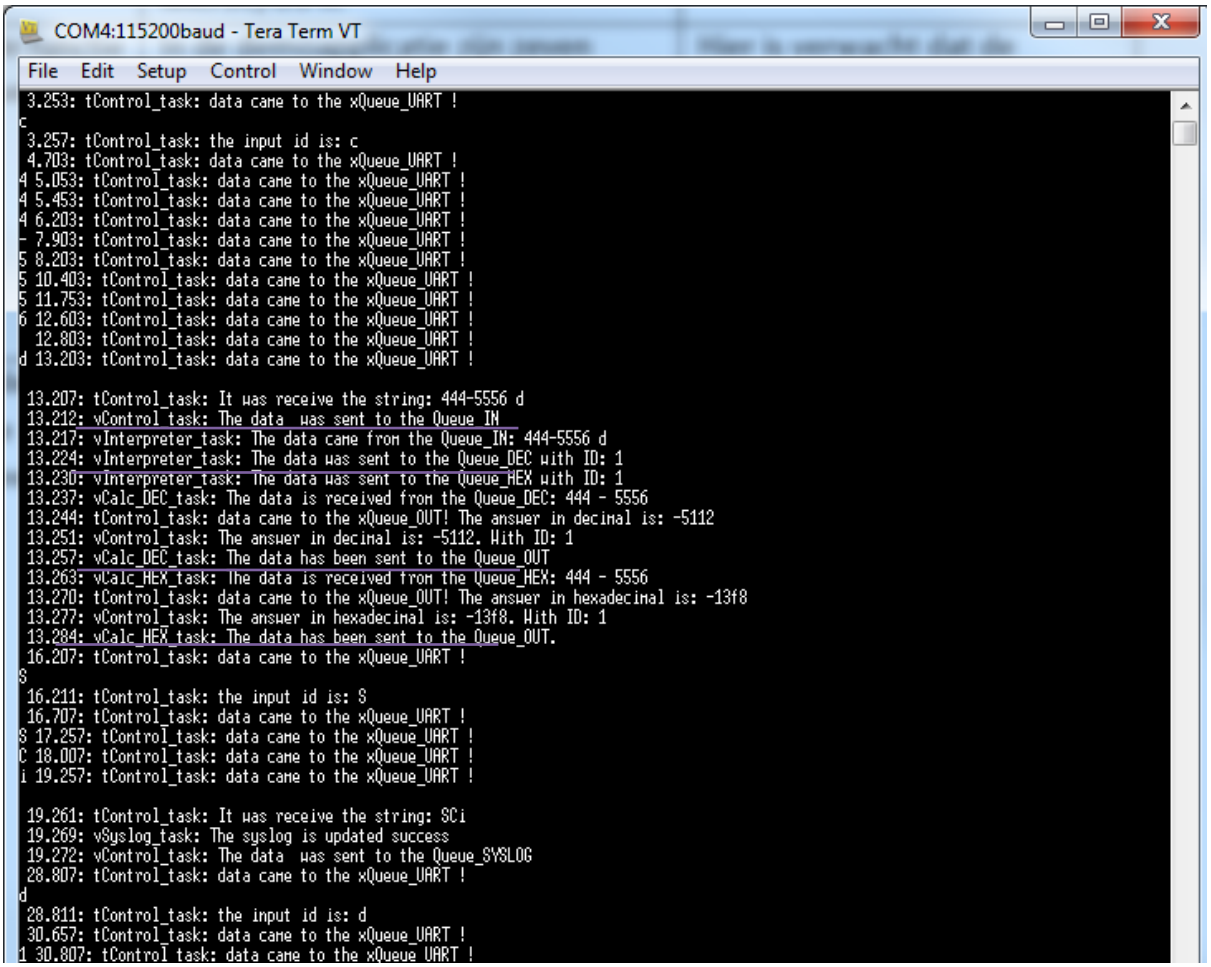
Resultaat

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 9.1	***V***
MicroC/OS-II	Printscreen 9.2	***V***

Printscreen 9.1: Output van Test 9 - FreeRTOS



Printscreen 9.2: Output van Test 9 – MicroC/OS-II



- **Test 10.**

Opstelling:

Hier wordt de functie die een queue opnieuw zet*: pr_QueueReset()* en de functie die status van de queues aantoont geïmplementeerd. In de Calc_DEC en Calc_HEX worden de ontvangen functies geblokkeerd daardoor worden de queues HEX en DEC overgelopen. Wanneer de queues vol zullen zijn, worden de reset functie vanuit Interpreter taak aangeroepen en de queues moeten weer leeg zijn.
Hier worden zes verzoeken gemaakt:
c444-5556 d
c444-5556 h
c444+5556 d
c444*5556 d
c5556 /444d
c17-17 h
Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II FreeRTOS.

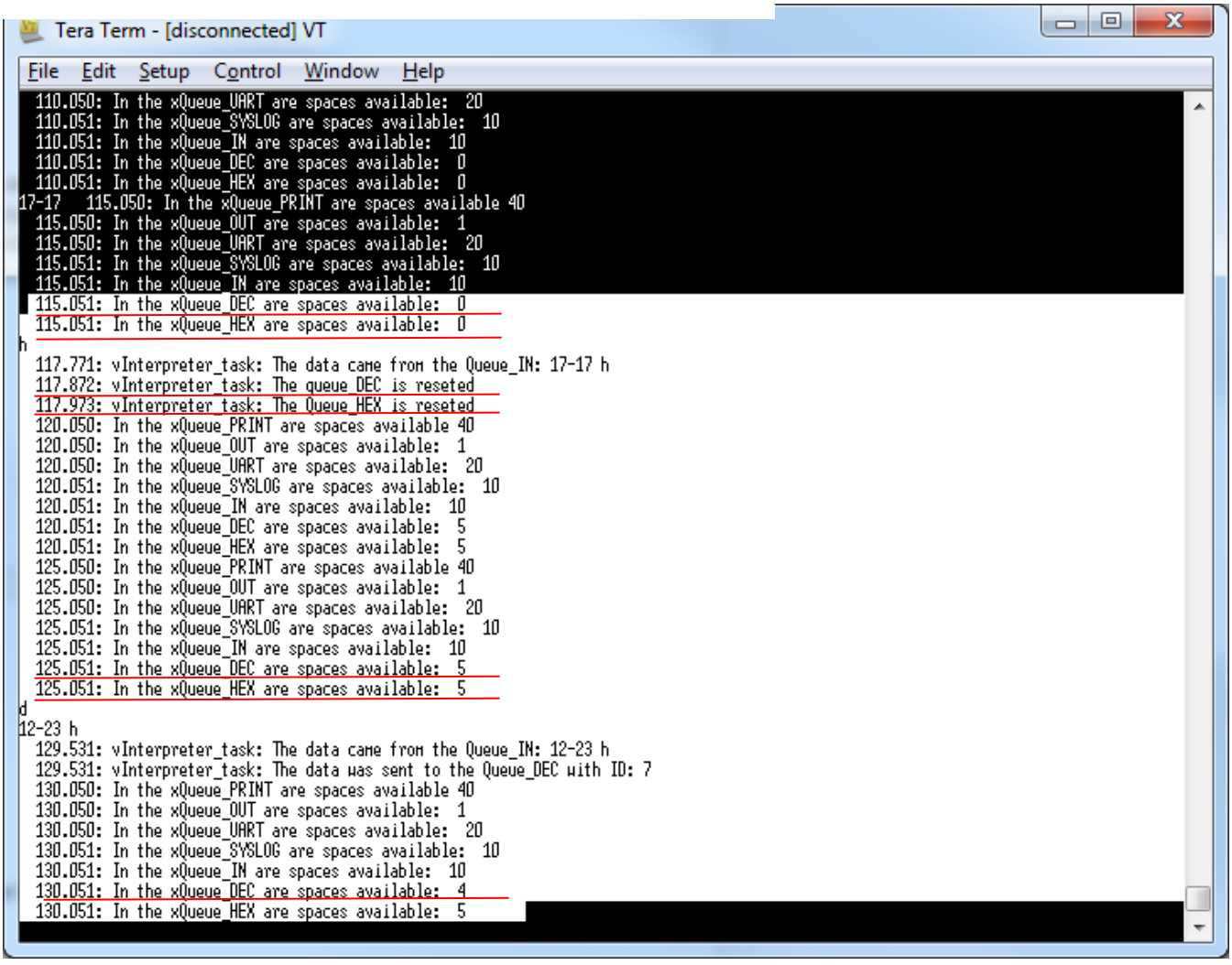
Verwachtte resultaat:

Hier is verwacht dat tijdens deze test het aantal plaatsen in de queues HEX en DEC wordt verminderd en na het aanroepen van de reset functie wordt het aantal plaatsen in de queues weer gelijk aan 5.

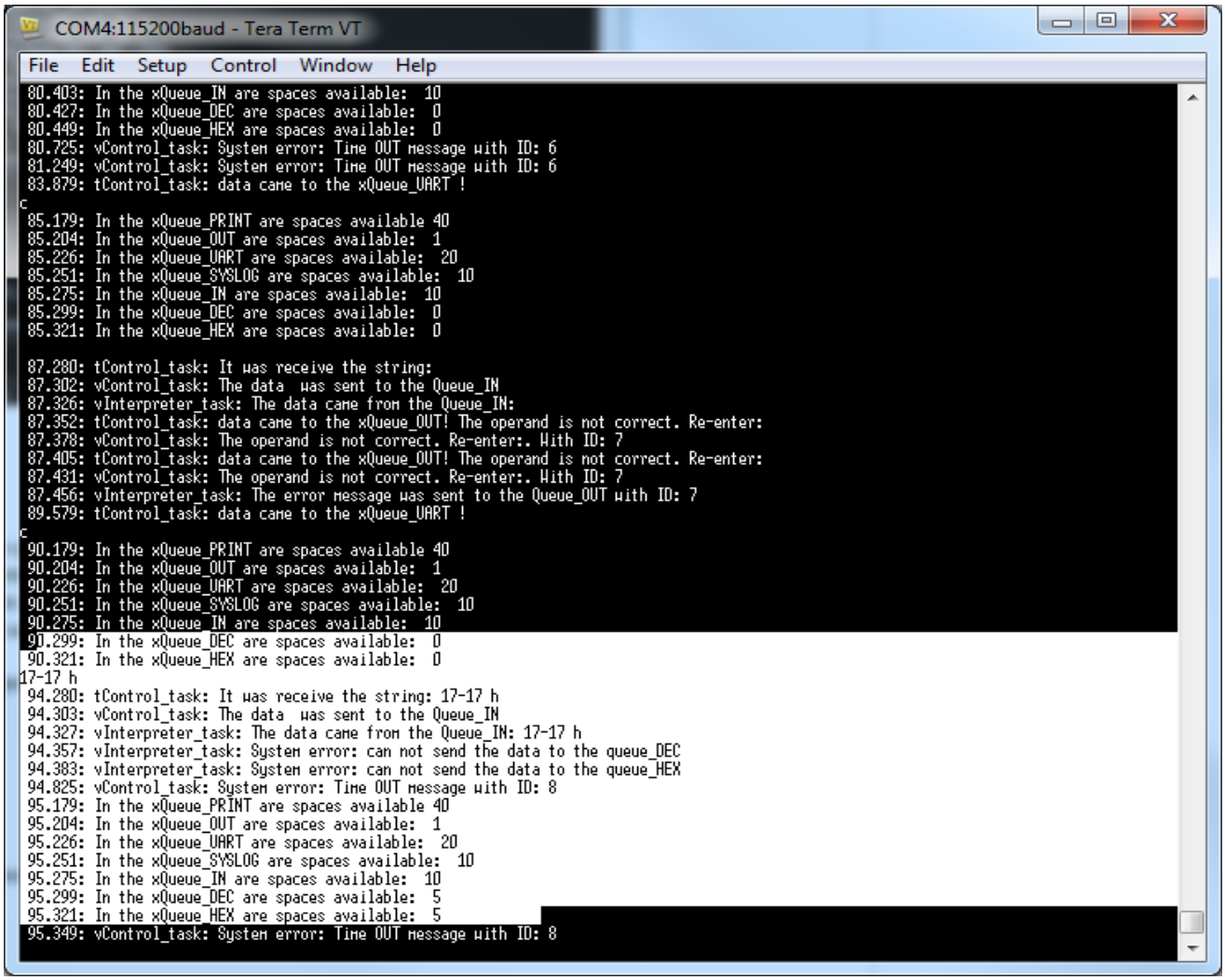
Resultaat

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 10.1	***V***
MicroC/OS-II	Printscreen 10.2	***V***

Printscreen 10.1: Output van Test 10 - FreeRTOS



Printscreen 10.2: Output van Test 10 – MicroC/OS-II



- **Test 11.**

Opstelling:

Hier wordt in de demoapplicatie een Semaphore gecreëerd worden. De *UART_taak* neemt de semaphore, na de uitvoering geeft ze de semaphore niet terug. De *Control_taak* moet deze semaphore ook nemen voor de uitvoering. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

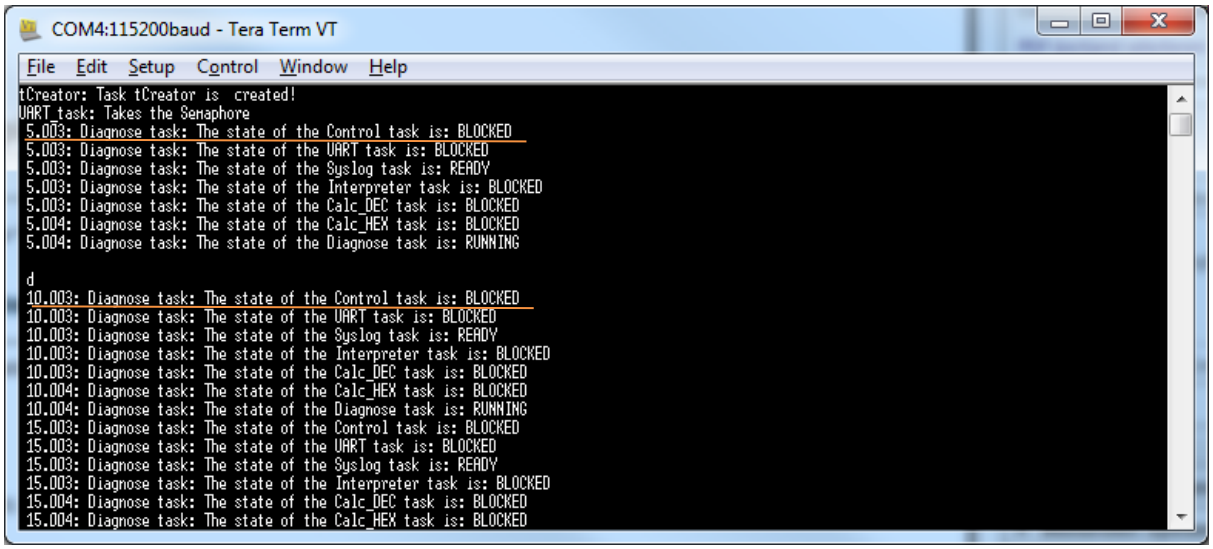
Verwachtte resultaat:

FreeRTOS: Hier is verwacht dat de toestand *Control_taak* wordt na het input niet veranderd. De toestand van *Control_taak* blijft *Blocked* en programma gaat niet input verwerken.
MicroC/OS-II: Hier is verwacht dat de toestand *Control_taak* wordt veranderd naar *WAIT_OF_SEMAPHORE*.

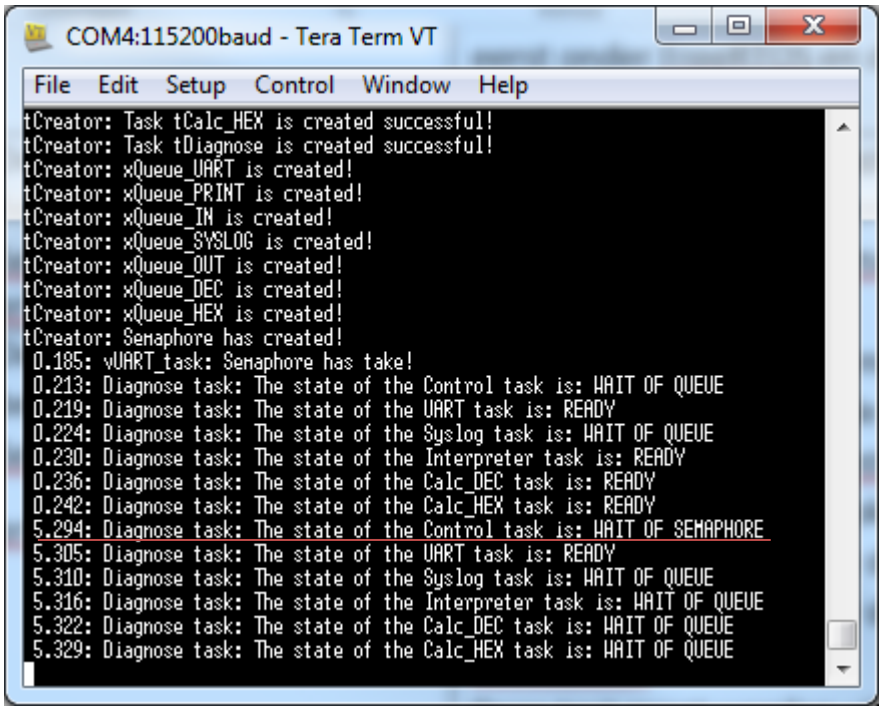
Resultaat

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Print Screen 11.1	***V***
MicroC/OS-II	Print Screen 11.2	***V***

Printscreen 11.1: Output van Test 11 – FreeRTOS



Printscreen 11.2: Output van Test 11 – MicroC/OS-II



- **Test 12.**

Opstelling:

Hier wordt in de demoapplicatie een Semaphore gecreëerd worden. De *UART_taak* neemt de semaphore, na de uitvoering geeft ze de semaphore terug met functie: *pr_SemaphoreGive()*. De *Control_taak* moet deze semaphore ook nemen voor de uitvoering. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

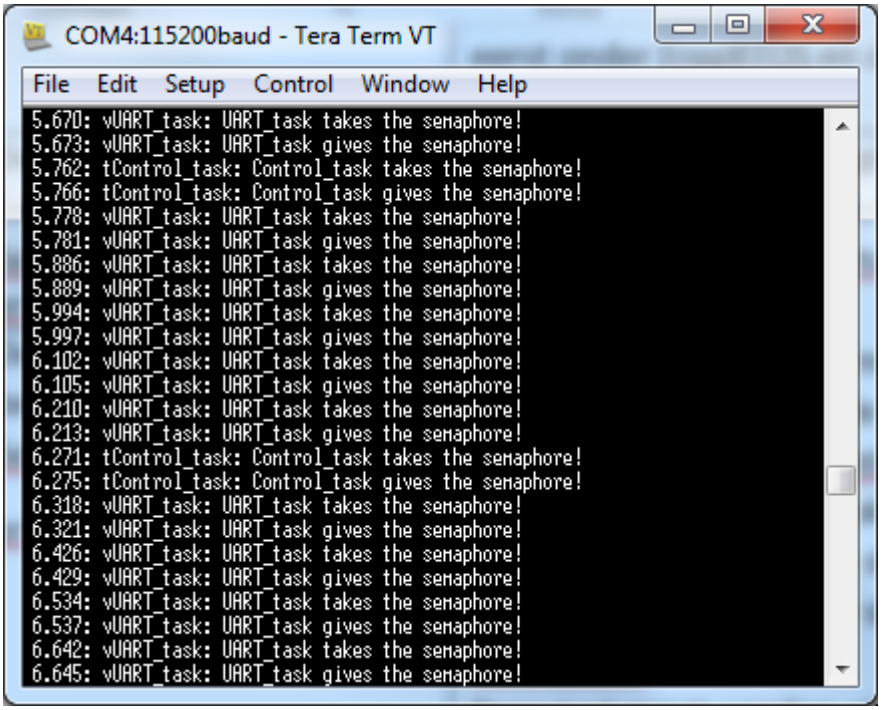
Verwachtte resultaat:

Hier is verwacht de volgende uitprint:
"*UART_task takes the semaphore"*
"UART_task gives the semaphore"
"Control_task takes the Semaphore"
….
"Control_task gives the semaphore"

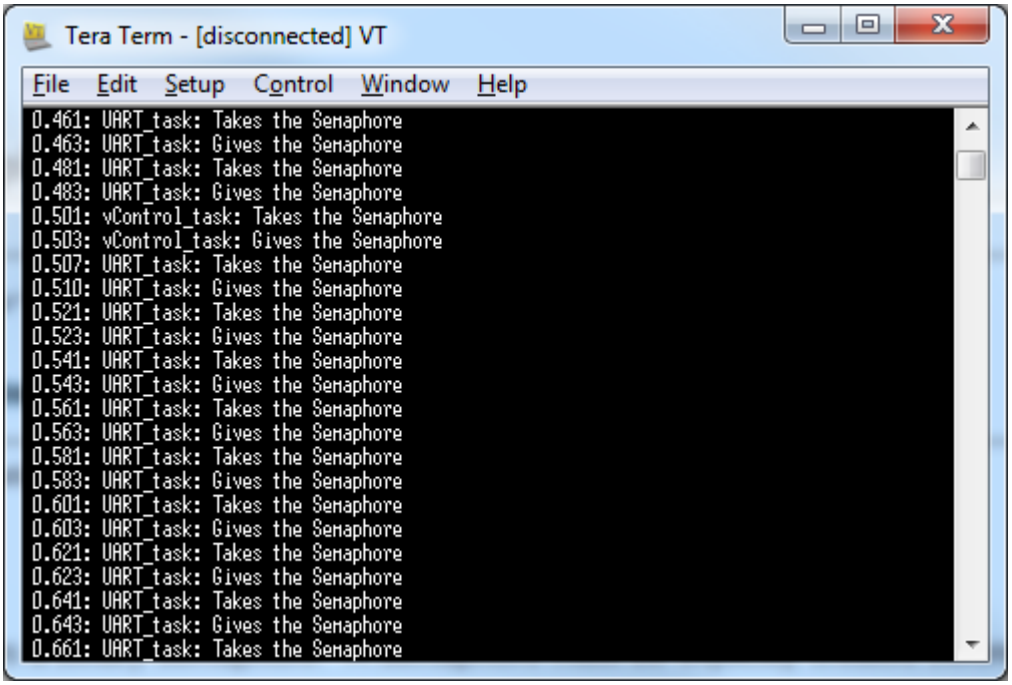
RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Print Screen 12.1	***V***
MicroC/OS-II	Print Screen 12.2	***V***

Resultaat

Printscreen 12.1: Output van Test 12 – FreeRTOS



Printscreen 12.2: Output van Test 12 – MicroC/OS-II



- **Test 13.**

Opstelling:

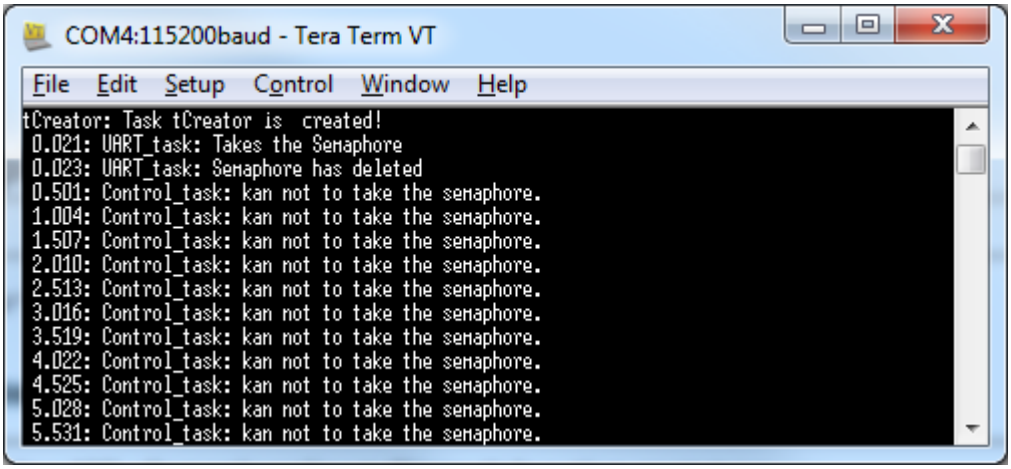
Hier wordt de test N11 weer uitgevoerd. Maar voor het eind van de uitvoering van *UART_taak* wordt de semaphore verwijderd. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

Verwachtte resultaat:

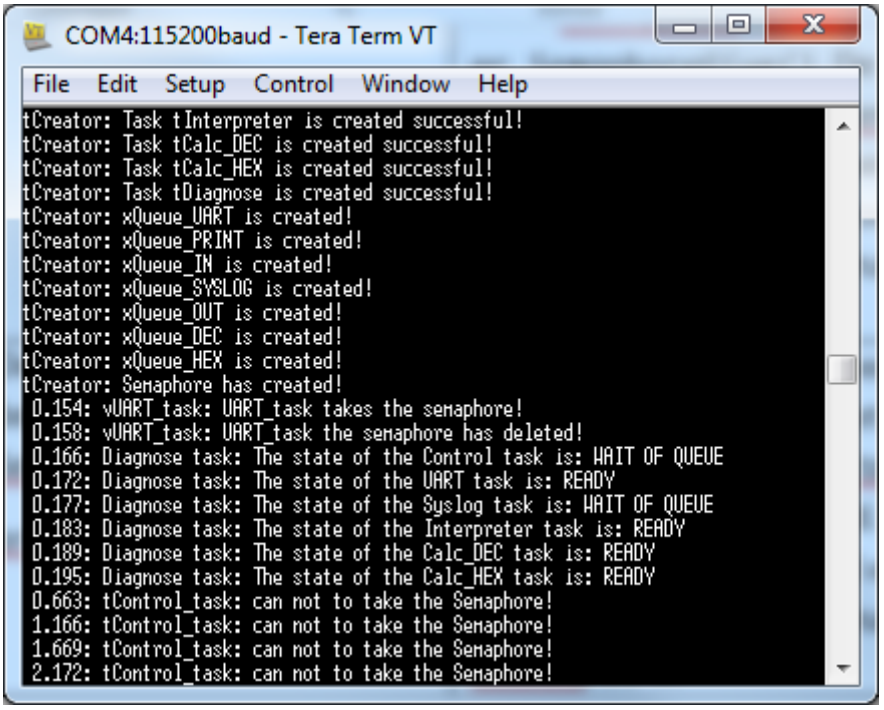
Hier is verwacht een foutmelding:
"*Control_task: kan not to take the semaphore."*
Resultaat:

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Print Screen 13.1	***V***
MicroC/OS-II	Print Screen 13.2	***V***

Printscreen 13.1: Output van Test 13 – FreeRTOS



Printscreen 13.2: Output van Test 13 – MicroC/OS-II



- **Test 14.**

Opstelling:

Aan het begin van de uitvoering van Syslog functie neemt ze Mutex. Aan het eind van de uitvoering geeft ze Mutex terug. Voor deze test voegde ik de functie `pr_DeleteMutex(xMutex)` in *Syslog_task* toe, na het syslog configuratie verzoek.. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

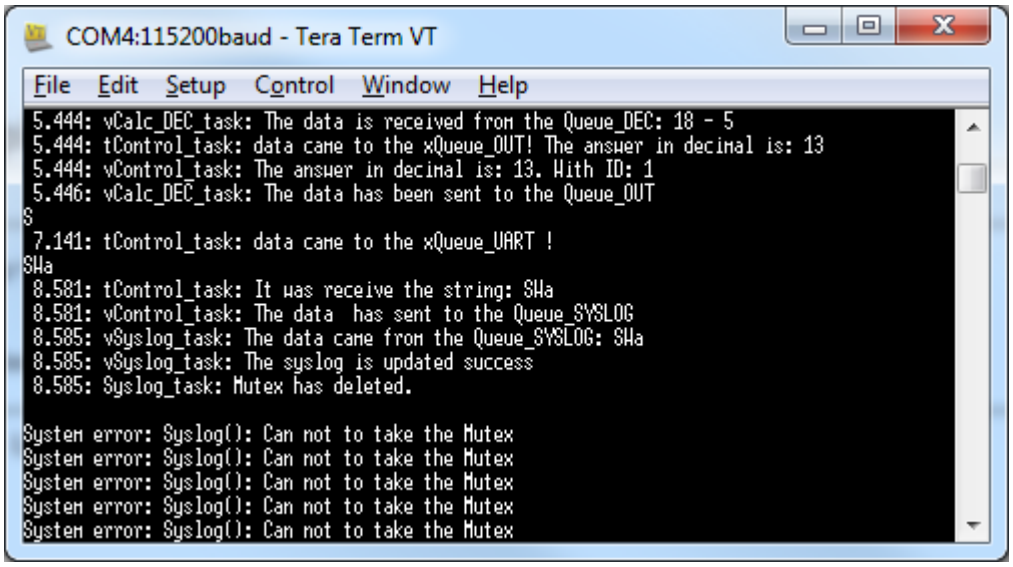
Verwachtte resultaat:

Na de invoering van het syslog configuratie request verwacht hier een foutmelding:
"*vSyslog_task_task: kan not to take the Mutex.*"

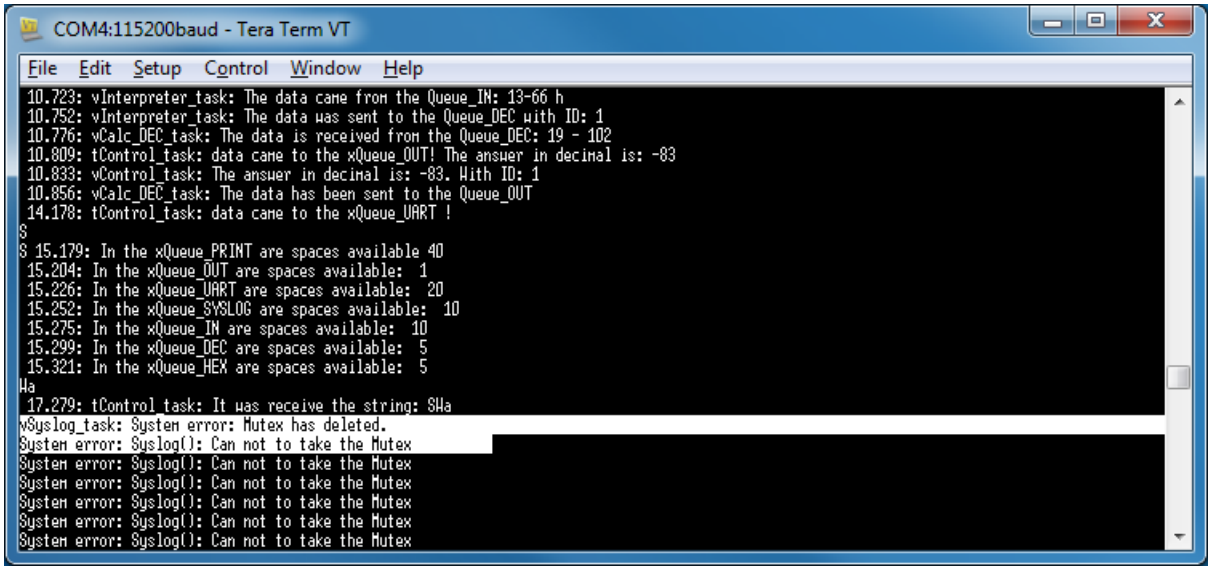
Resultaat:

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Print Screen 14.1	***V***
MicroC/OS-II	Print Screen 14.2	***V***

Printscreen 14.1: Output van Test 14 – FreeRTOS



Printscreen 14.2: Output van Test 14 – MicroC/OS-II



- **Test 15.**

Opstelling:

In de demoapplicatie is de Mutex gebruikt om de *Syslog* functie te beveiligen.
Tijdens de test wordt de Mutex uit Syslog functie verwijderd. De diagnose_task wordt uitgevoerd met het periode 0,1sec en *UART_task* wordt uitgevoerd met het periode 0.02sec. Bij de elke uitvoering *UART_task* moet printen: "*UART_task*: Running" en *Diagnose_task*: *"Diagnose_task running"*
Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

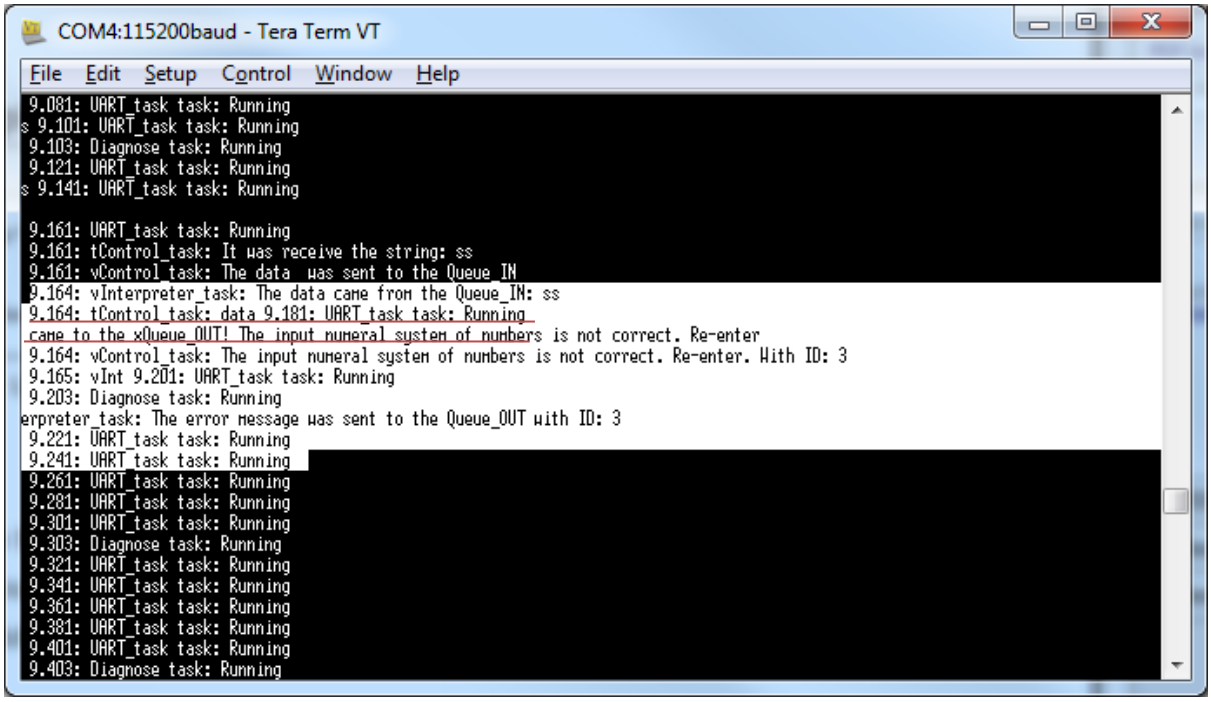
Verwachtte resultaat:

Hier is verwacht dat de output chaotisch uit.

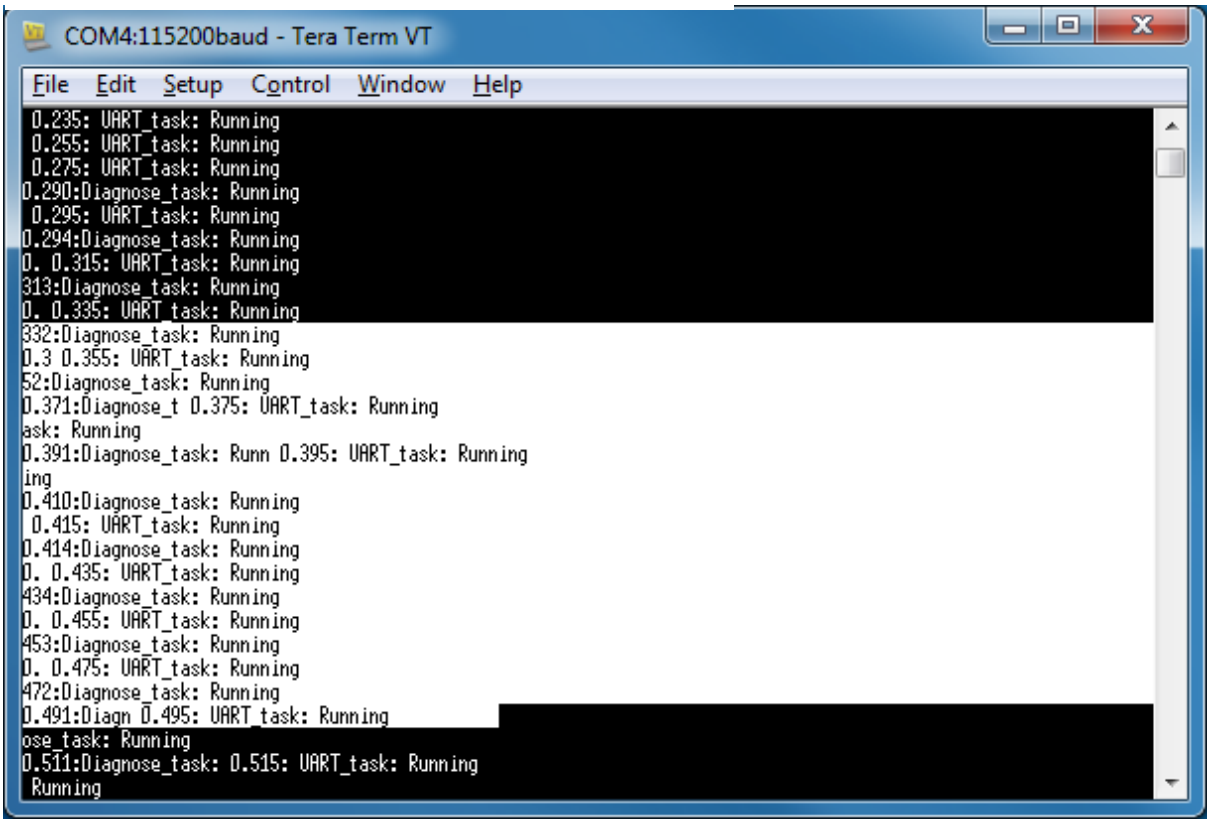
Resultaat:

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Print Screen 14.1	***V***
MicroC/OS-II	Print Screen 14.2	***V***

Printscreen 15.1: Output van Test 15 – FreeRTOS



Printscreen 15.2: Output van Test 15 – MicroC/OS-II



- **Test 16.**

Opstelling:

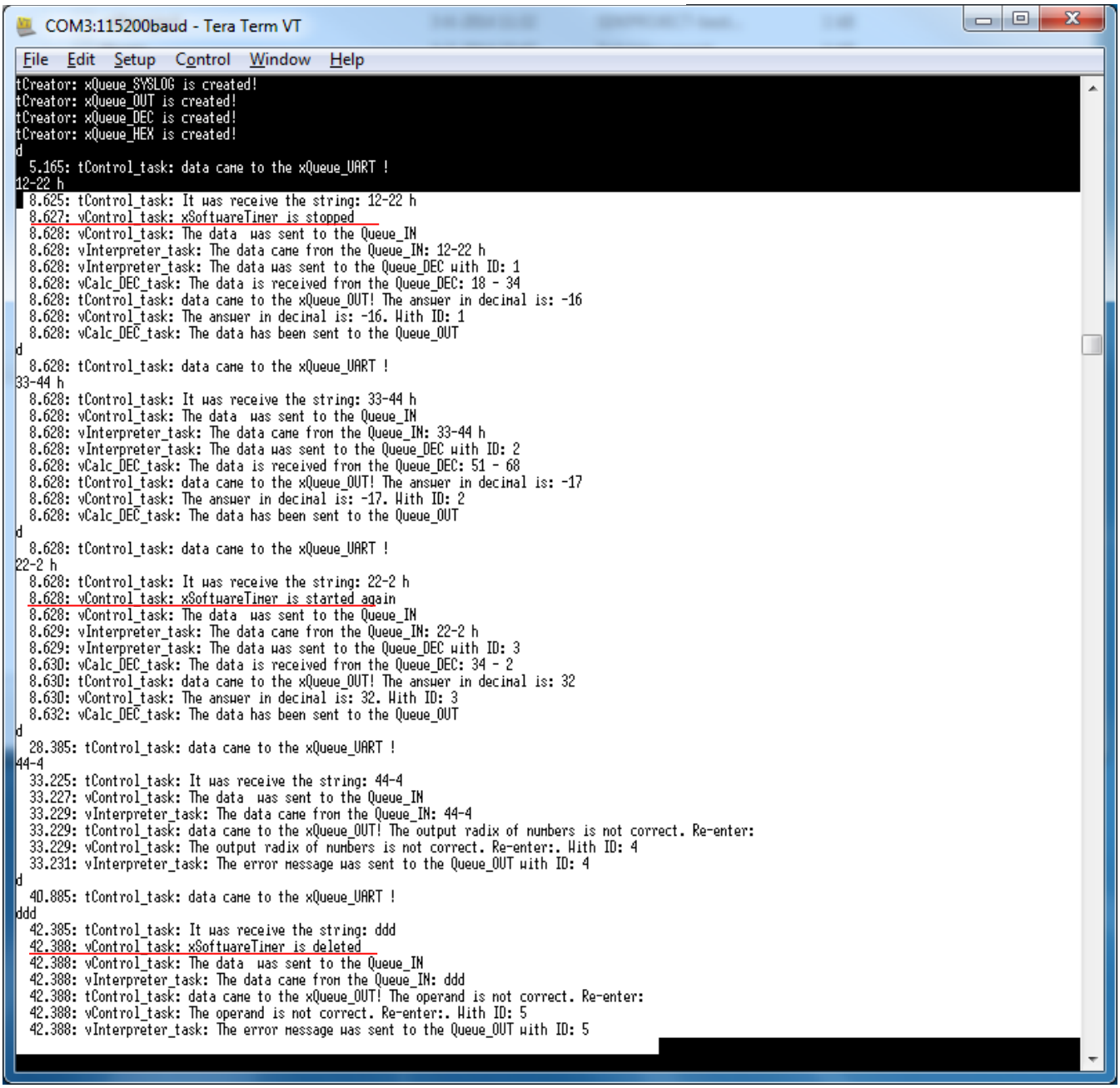
In de demoapplicatie werd de softtimer gecreëerd, met de functie: *pr_TimerCreate()* en opgestart met de functie: *pr_TimerStart()* . Bij de output is de tijd van de softtimer weergegeven. Bij deze test worden acht verzoeken gemaakt. Na de tweede verzoek wordt de functie aangeroepen die de softtimer stoppen: *pr_TimerStop().* Na de vierde verzoek wordt de functie aangeroepen die de timer starten. Daarna na de zesde verzoek wordt de functie aangeroepen die de softtimer delete: *pr_TimerDelete()*.
Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

Verwachtte resultaat:

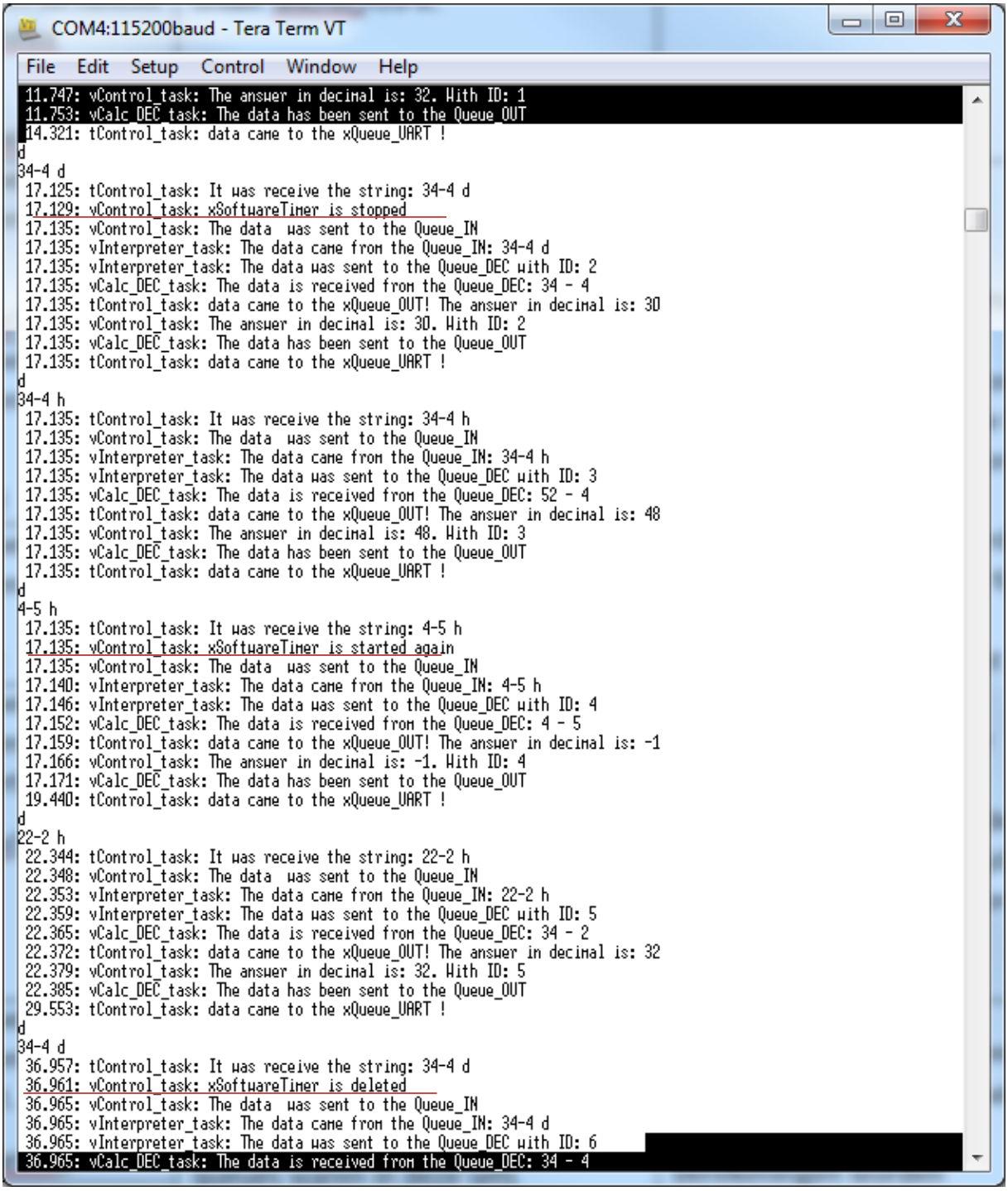
Hier wordt de volgende output verwacht:
Voor de tweede verzoek: wordt de output geprint met de tijd erbij;
Na de tweede verzoek: wordt de output geprint maar de tijd die bij de output staat, wordt niet veranderd.
Na de vierde verzoek: wordt de output geprint met de tijd erbij;
Na de zesde verzoek: wordt de output geprint maar de tijd die bij de output staat, wordt niet veranderd.
Resultaat

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 16.1	***V***
MicroC/OS-II	Printscreen 16.2	***V***

Printscreen 16.1. Output van Test 16 - FreeRTOS



Printscreen 16.2: Output van Test 16– MicroC/OS-II



- **Test 17.**

Opstelling:

In de demoapplicatie werd de set van de queues gecreëerd in de *Control_taak* en *Syslog_taak*, met de functie: *pr_QueueCreateSet().* Tijdens deze test eerst moet de demoapplicatie laten worden gedraaid. Daarna moet de functie die de set creëert geblokkeerd en de applicatie wordt weer gedraaid.
Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

Verwachtte resultaat:

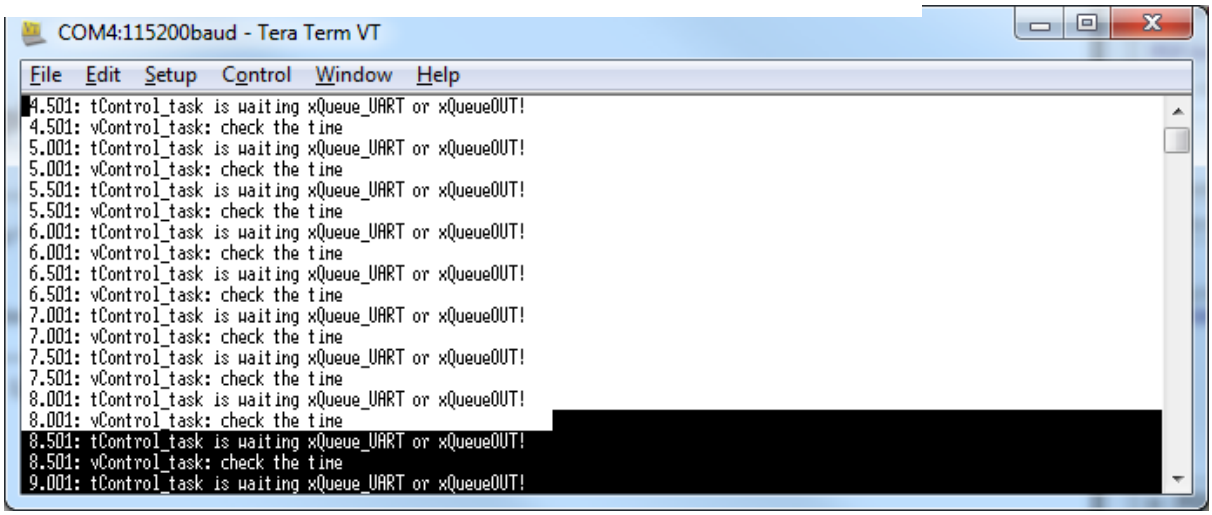
Hier wordt verwacht dat eerst de demoapplicatie normaal gaat uitvoeren, dus de berekeningen worden uitgevoerd, time controle tenminste een keer per 0.5 secunde en foutmeldingen. Daarna als de setfuncties worden geblokkeerd, dan de periodiek van de Control taak wordt gebroken, en de Syslog taak kan niet optimaal worden gesynchroniseerd door de twee queues: queue PRINT en queue Syslog. (Onder optimale synchronisatie is begrepen dat met het gebruik van de set van de queues Syslog taak wordt geblokkeerd voor oneindig tijd tot de data komt in de queue Syslog of queue PRINT binnen. Zonder het gebruikt van de set van de queues dat niet mogelijk is, want de taak zal dan wachten de event alleen in één queue).

Resultaat

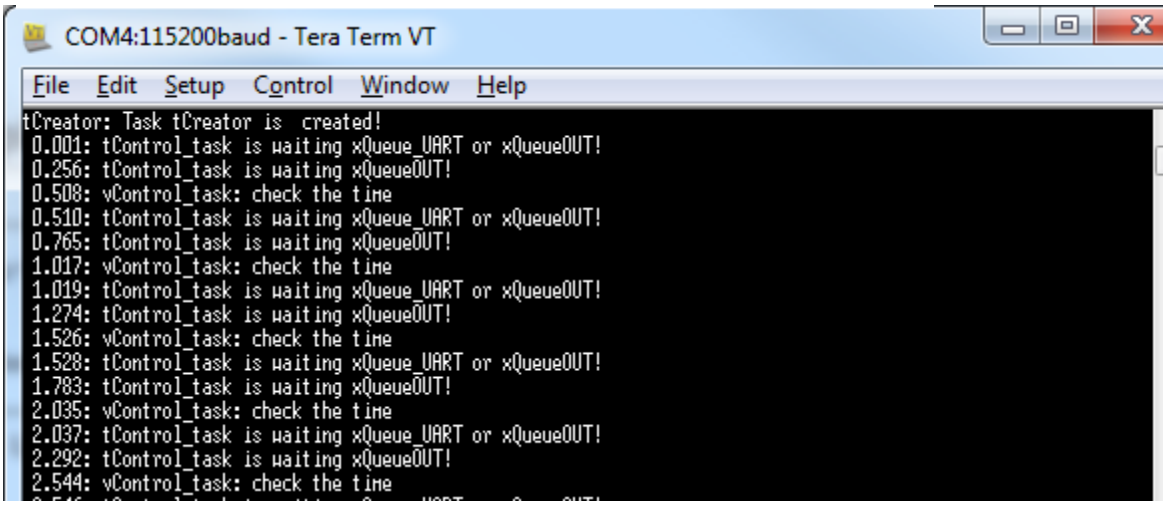
Zonder het gebruikt van de set van queues is onmogelijk om de Syslog en Control taken op data event te synchroniseren. Ik kan alleen als synchronisatie tool de tijd event gebruiken. Zonder het gebruik van de set zal de Control taak 250msec de data in de queue UART wachten en daarna 250msec data in de queue OUT wachten. Met deze installatie bewaar ik de periodieke tijd controle maar het is niet optimaal, want als de taak wacht de queue OUT en de data komt in de queue UART dan reageert de taak op de event allen over 250 msec, zie Print Screen 17.1a) en 17.1 b)

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Print Screen 17.1a) – met set van de queues Print Screen 17.1b) – zonder set van de queues	***V*** ***V***
MicroC/OS-II	Print Screen 17.2a)– met set van de queues Print Screen 17.2b) – zonder set van de queues	***V*** ***V***

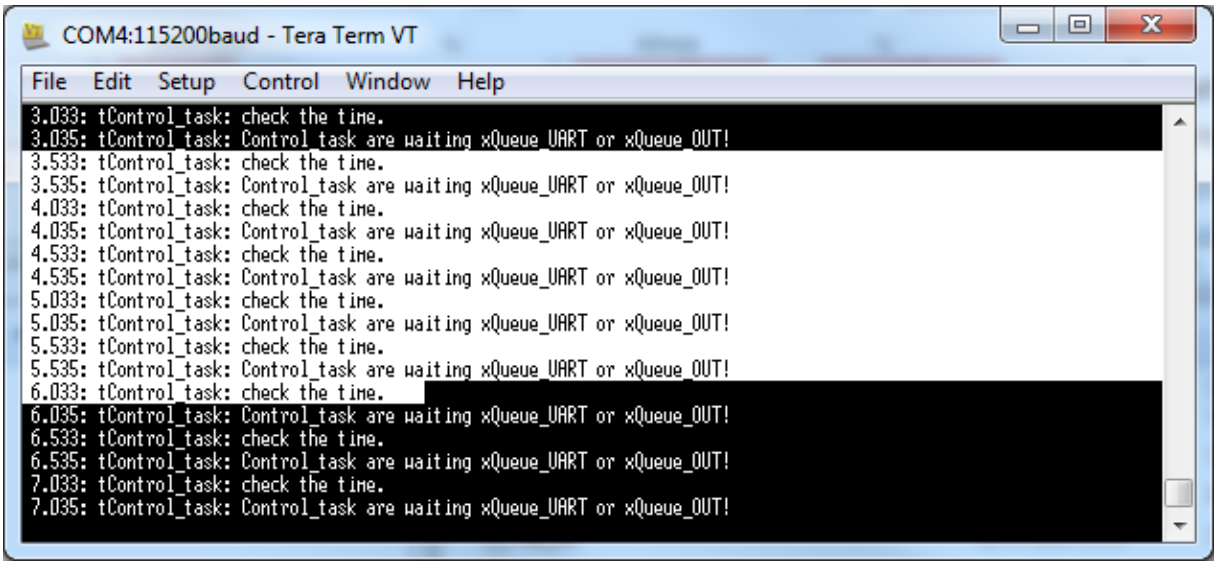
Printscreen 17.1a: Testcase 1.Output van Test 17– Freertos



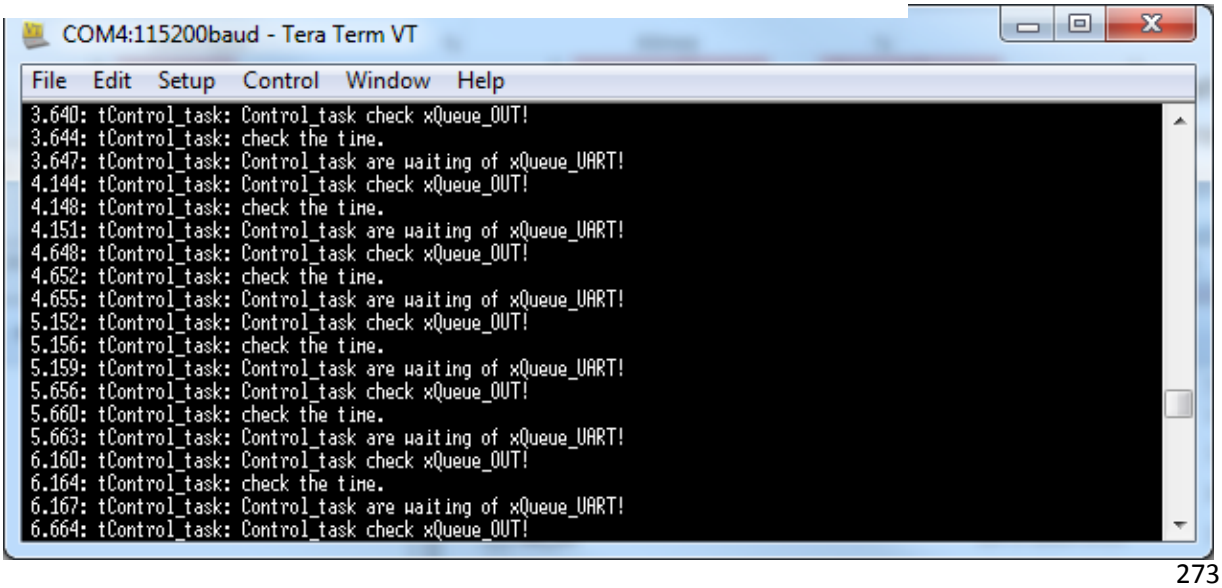
Printscreen 17.2b: Testcase 2.Output van Test 17– Freertos



Printscreen 17.2a: Testcase 1.Output van Test 17– MicroC/OS-II



Printscreen 17.2b: Testcase 2.Output van Test 17– MicroC/OS-II



- **Test 18.**

Opstelling:

In de demoapplicatie in de Diagnose taak was de functie geïmplementeerd die het aantal vrije plaatsen in de bepaalde queue aantoont, daarvoor werd de functie gebruikt: *pr_qQuery().* Tijdens deze test wordt de receiver functie van de queue IN geblokkeerd.
Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

Verwachtte resultaat:

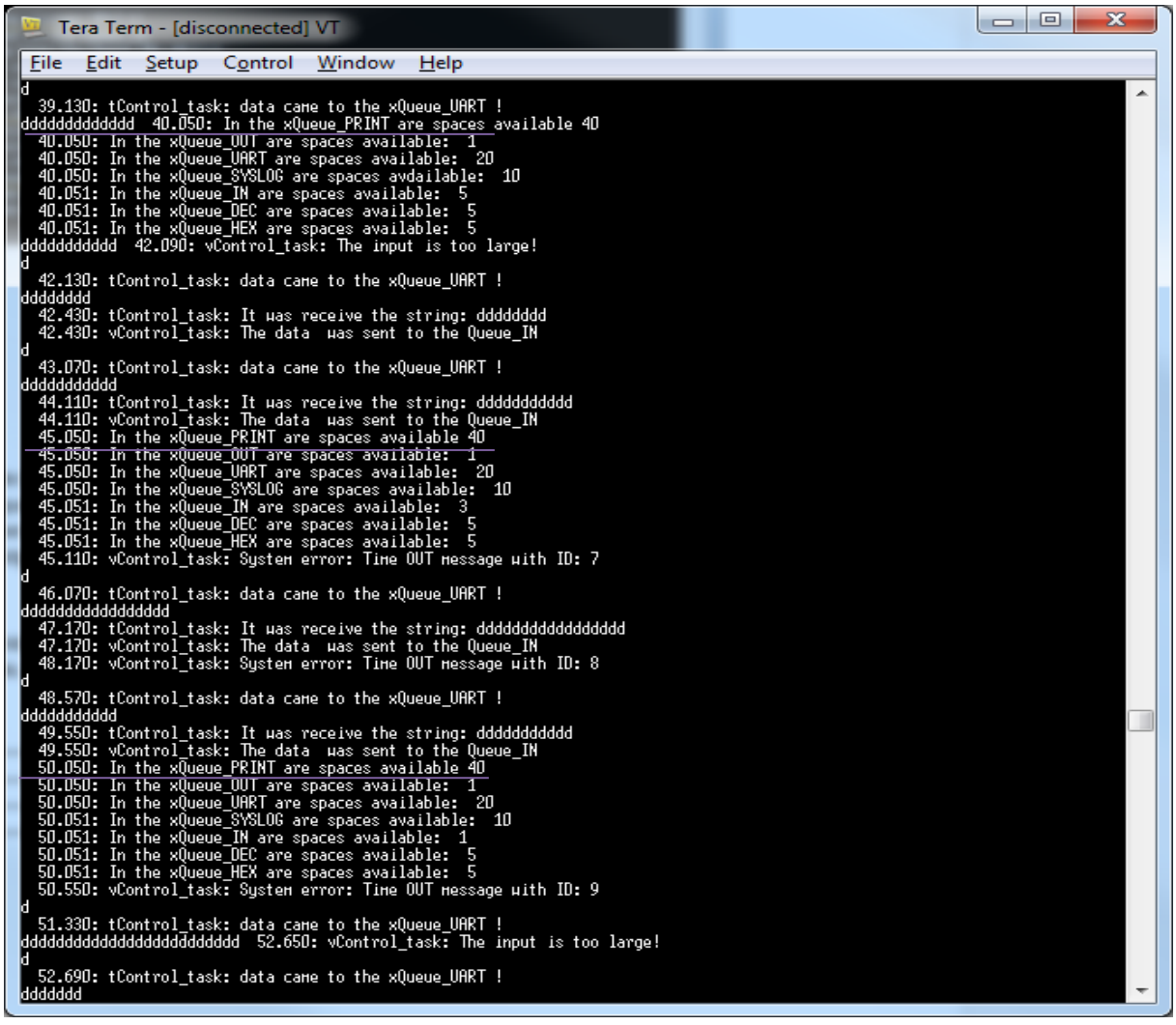
Als de functie correct werkt dan is het verwacht dat het aantal plaatsen in de queue IN wordt verminderd na elk verzoek.

Resultaat

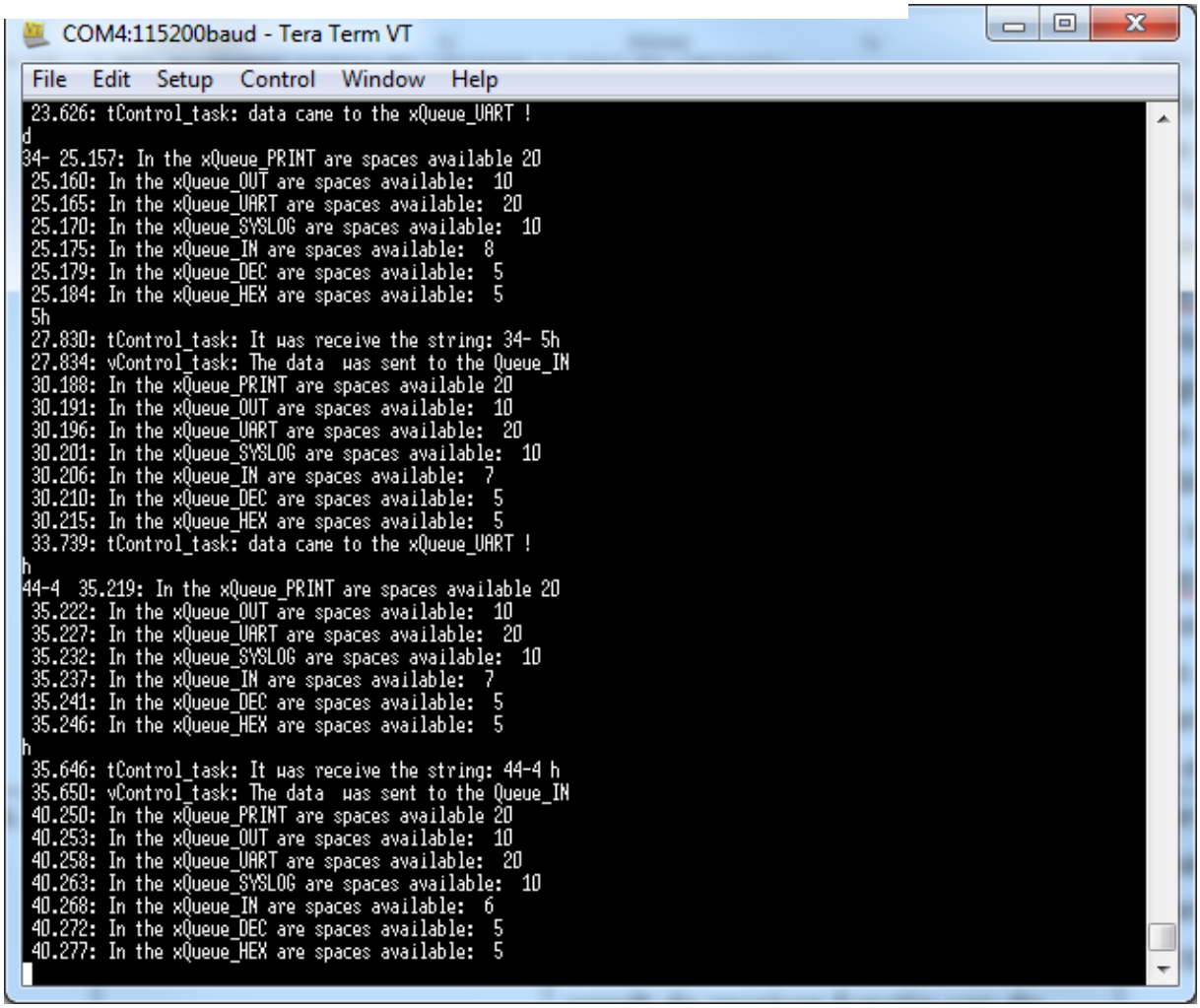
De wrapper functie: *pr_qQuery()* werkt correct.

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 18.1	***V***
MicroC/OS-II	Printscreen 18.2	***V***

Printscreen 18.1: Output van Test 18 - FreeRTOS



Printscreen 18.2:.Output van Test 18– MicroC/OS-II



- **Test 19.**

Opstelling:

In de demoapplicatie in de Diagnose taak was de functie geïmplementeerd die de maat van de resterende stack van de bepaalde taak toont aan: *prTaskGetStack()* . De totale maat van de stack van de taak wordt bij het creëren van de taak gedefinieerd. Tijdens deze test wordt de demoapplicatie vijf keer opgestart. Elke keer wordt de maat van de stack van de elke taak voor 100 groter gedefinieerd.
Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

Verwachtte resultaat:

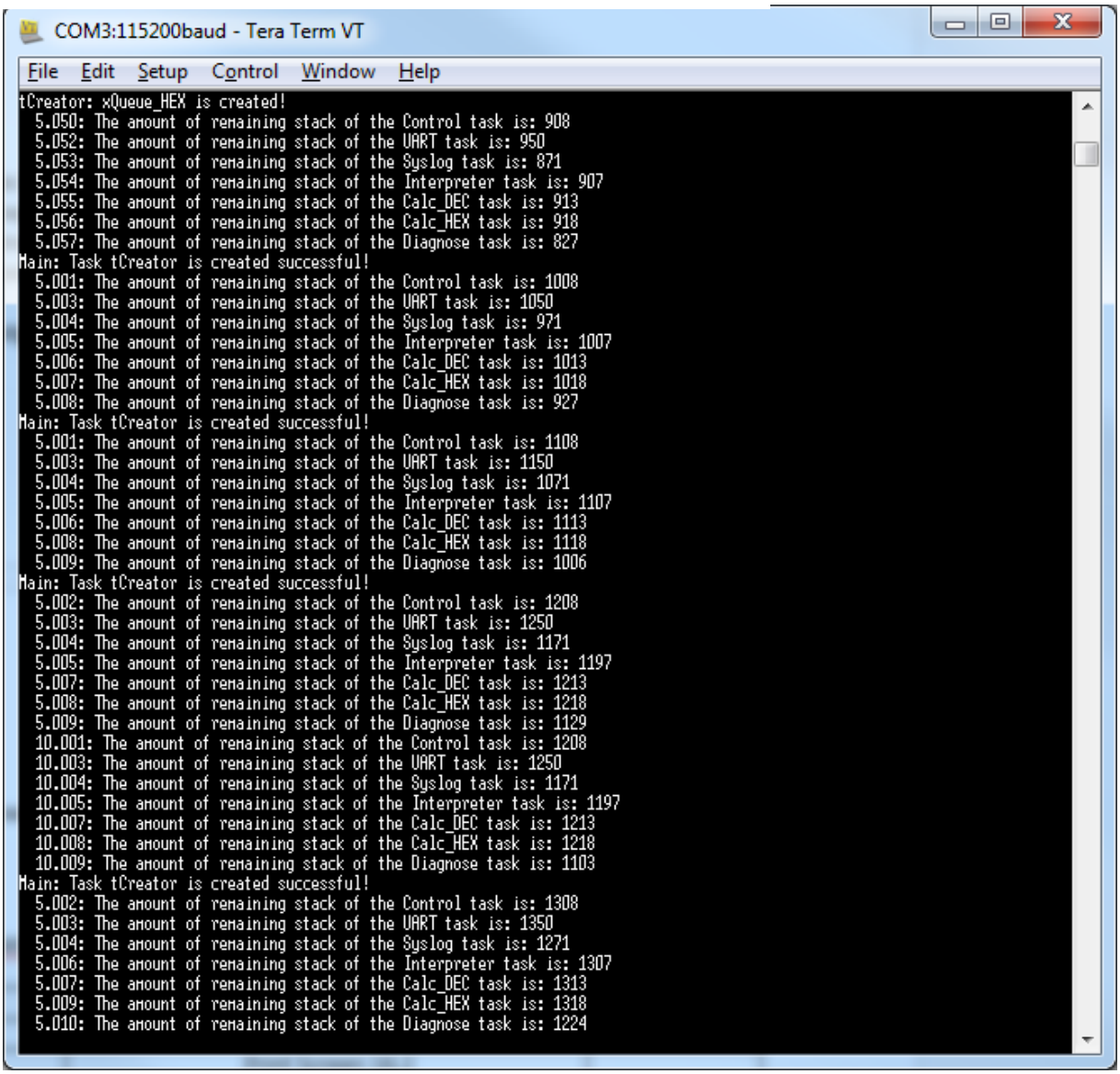
Hier is verwacht dat de maat van de resterende stack van de taak wordt voor 100 verhogen per elk opstarten.

Resultaat

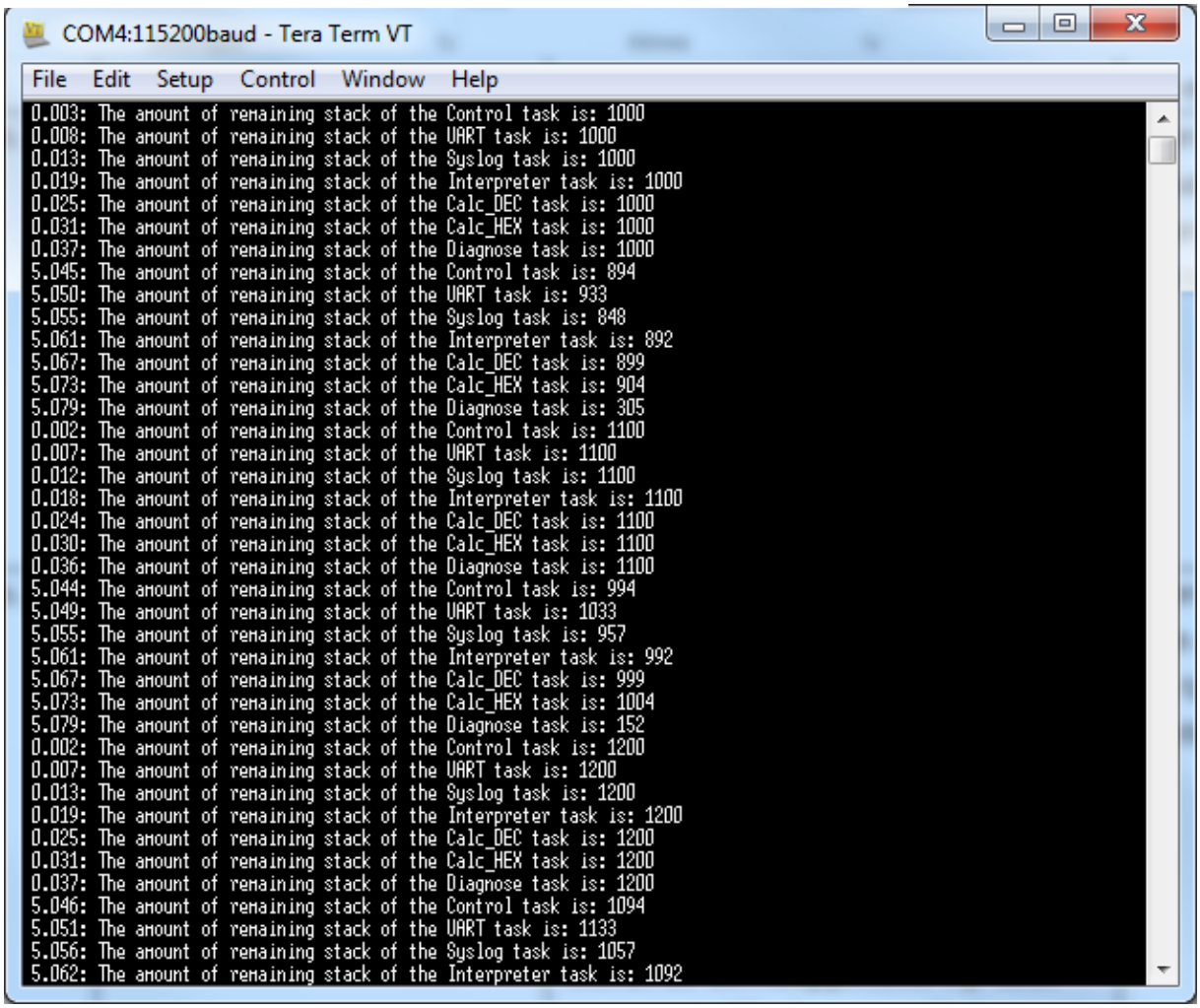
De wrapper functie *prTaskGetStack()* werkt correct.

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 19.1	***V***
MicroC/OS-II	Printscreen 19.2	***V***

Printscreen 19.1: Output van Test 19 - FreeRTOS



Printscreen 19.2:.Output van Test 19– MicroC/OS-II



- **Test 20.**

Opstelling:

In de demoapplicatie in de Diagnose taak was de functie geïmplementeerd die de prioriteit van de bepaalde taak aantoont: *prTaskGetPriority().* De prioriteiten zijn niet veranderd tijdens de uitvoering van de demoapplicatie 2.0. Tijdens deze test worden de prioriteiten van de taken worden wel veranderd.
Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II. Onder het MicroC/OS-II wordt de priority van de UART taak van 10 naar 11 veranderd en van de Control taak van 8 naar 9. Onder het FreeRTOS wordt de priority van de Control taak van 4 naar 3 veranderd en de priority van de Diagnose taak van 2 naar 4.

Verwachtte resultaat:

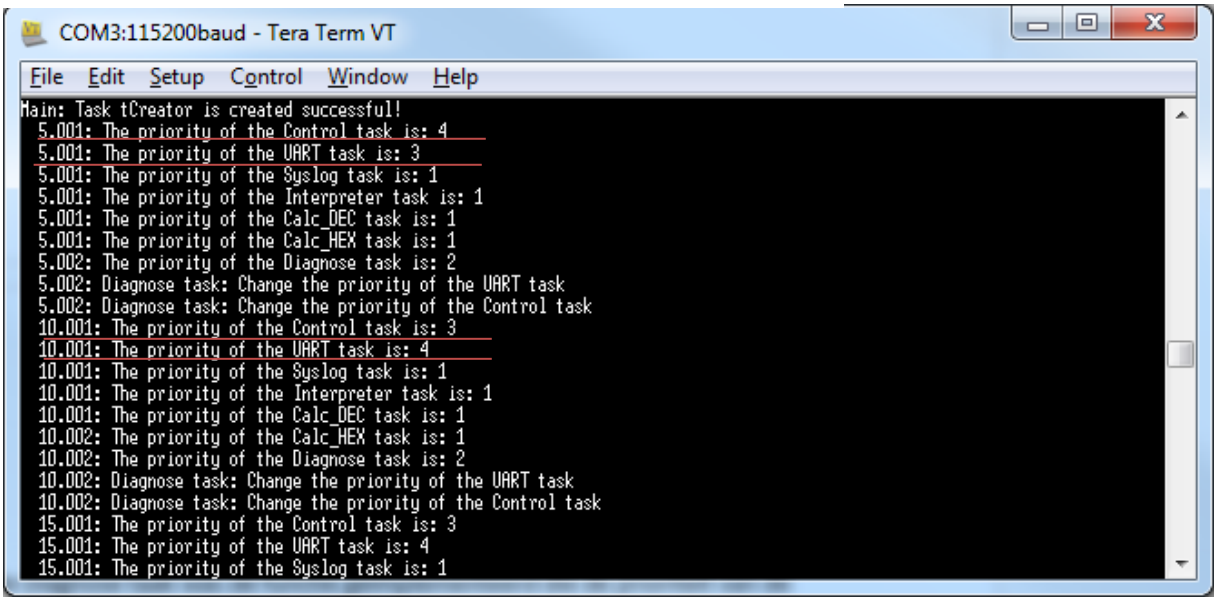
Hier is verwacht dat de prioriteiten van de UART taak, Control en Diagnose taak die worden aangetoond, worden veranderd na het uitroepen van de functie die de prioriteiten van de taken veranderd

Resultaat

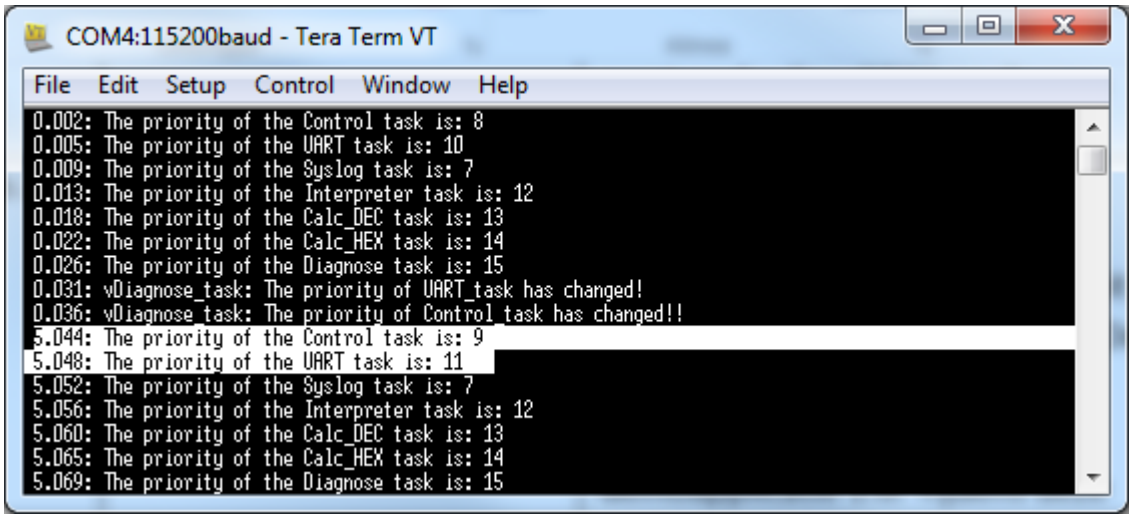
De wrapper functie *prTaskGetPriority()* werkt correct.

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 20.1	***V***
MicroC/OS-II	Printscreen 20.1	***V***

Printscreen 20.1: Output van Test 20- FreeRTOS



Printscreen 20.2:.Output van Test 20– MicroC/OS-II



- **Test 21.**

Opstelling:

In de demoapplicatie in de Diagnose taak was de functie geïmplementeerd die de naam van de taak aantoont: *prTaskGetName().* Tijdens deze test wordt de demoapplicatie opgestart. Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.
Verwachtte resultaat:

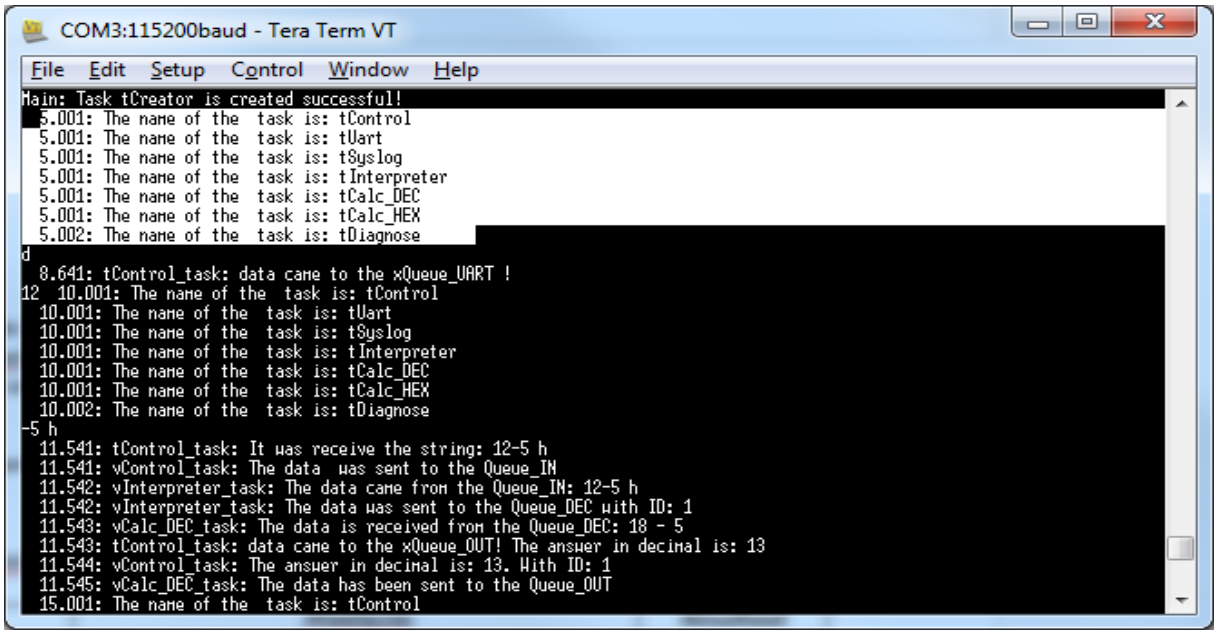
Hier is verwacht dat de namen van alle taken worden weergegeven op de terminal.

Resultaat

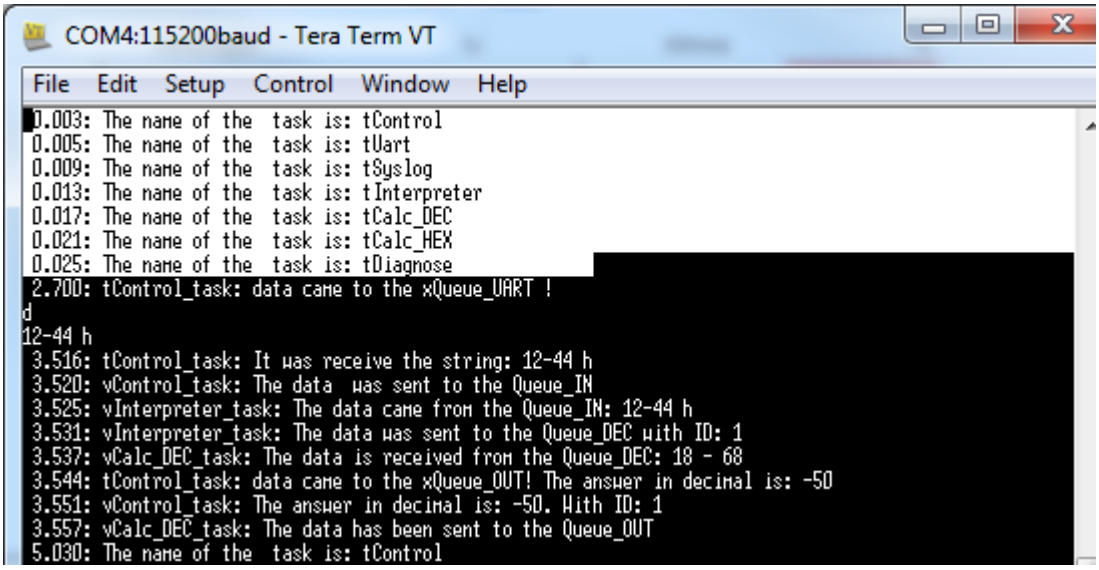
De wrapper functie *prTaskGetName()* werkt correct.

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 21.1	***V***
MicroC/OS-II	Printscreen 21.2	***V***

Printscreen 21.1 Output van Test 21 - FreeRTOS



Printscreen 21.2:.Output van Test 21– MicroC/OS-II



- **Test 22.**

Opstelling:

In de demoapplicatie in de Diagnose taak was de functie geïmplementeerd die de status van de taak aantoont: *prTaskGetStatus().* Tijdens deze test wordt de demoapplicatie opgestart. Tidens deze test wordt in de Diagnose taak de functie toegevoegd, die de Control taak opschort. Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

Verwachtte resultaat:

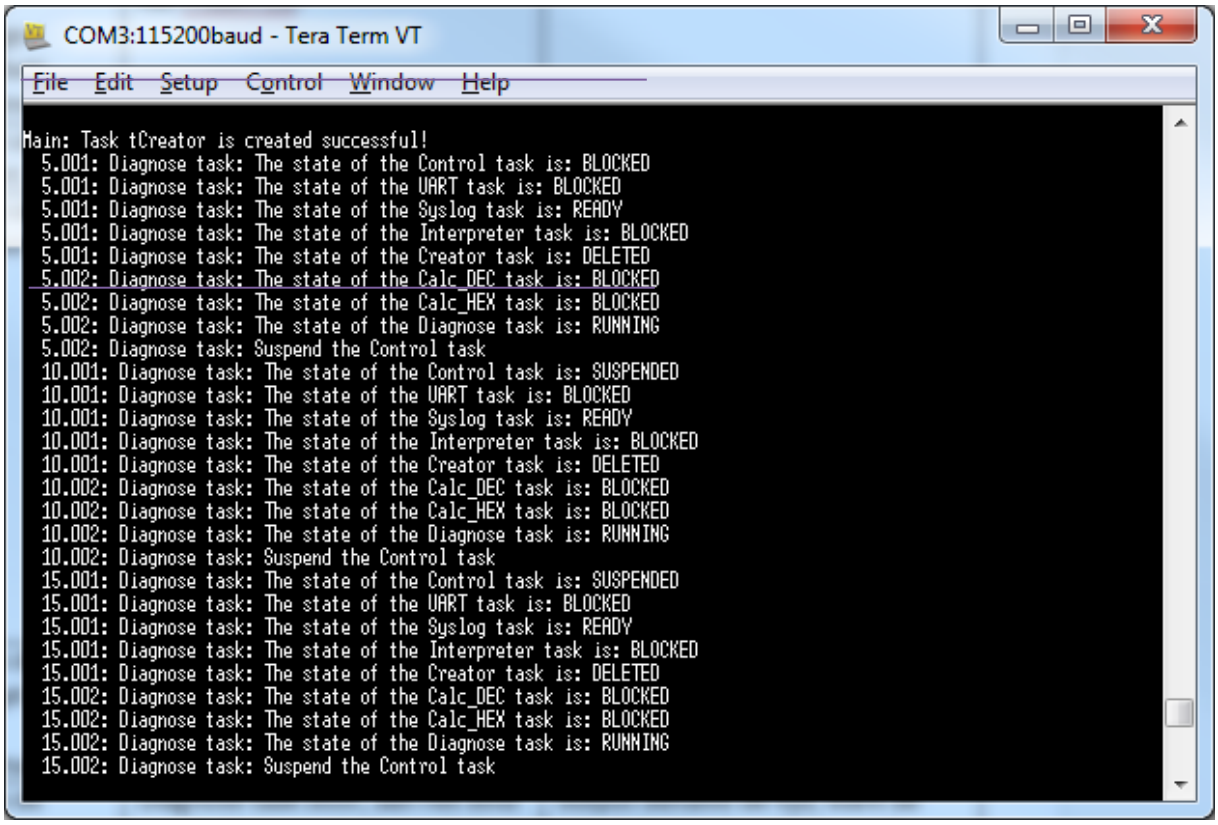
Hier is verwacht dat de statussen van alle taken worden op de terminal. Na het aangeroepen van de opschortte functie moet de status van de Controle taak *Suspend* zijn.

Resultaat

De wrapper functie : *prTaskGetStatus()* werkt correct.

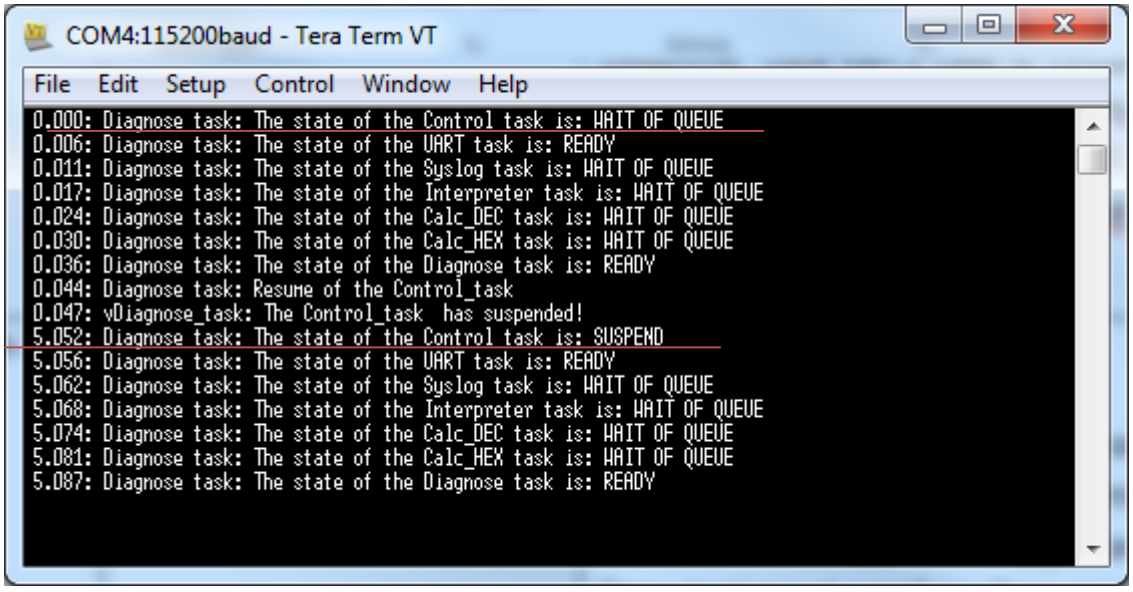
RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 22.1	***V***
MicroC/OS-II	Printscreen 22.2	***V***

Printscreen 22.1: Output van Test 22 - FreeRTOS



z

Printscreen 22.2:.Output van Test 22– MicroC/OS-II



- **Test 23.**

Opstelling:

In de demoapplicatie in de Diagnose_taak was de functie geïmplementeerd die alle informatie over de taak aantoont. Tijdens deze test wordt de demoapplicatie opgestart. Deze test moet worden uitgevoerd eerst onder FreeRTOS en daarna onder MicroC/OS-II.

Verwachtte resultaat:

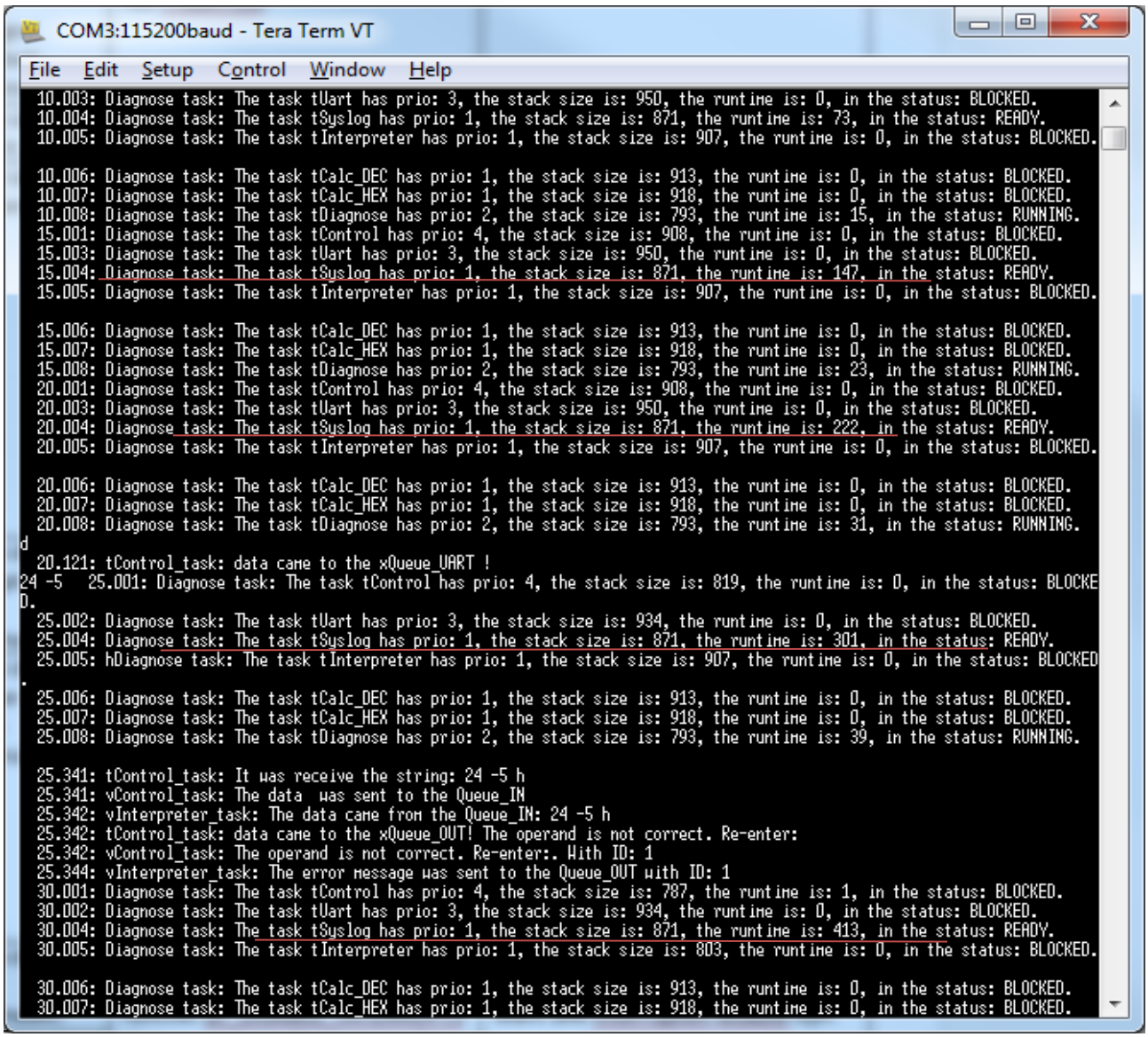
Hier is verwacht dat één keer per 5 secunde wordt volledig state van elke taak aangetoond. De runtime van de Syslog en Diagnose taak moet tijdens de uitvoering van de demoapplicatie groeien.

Resultaat

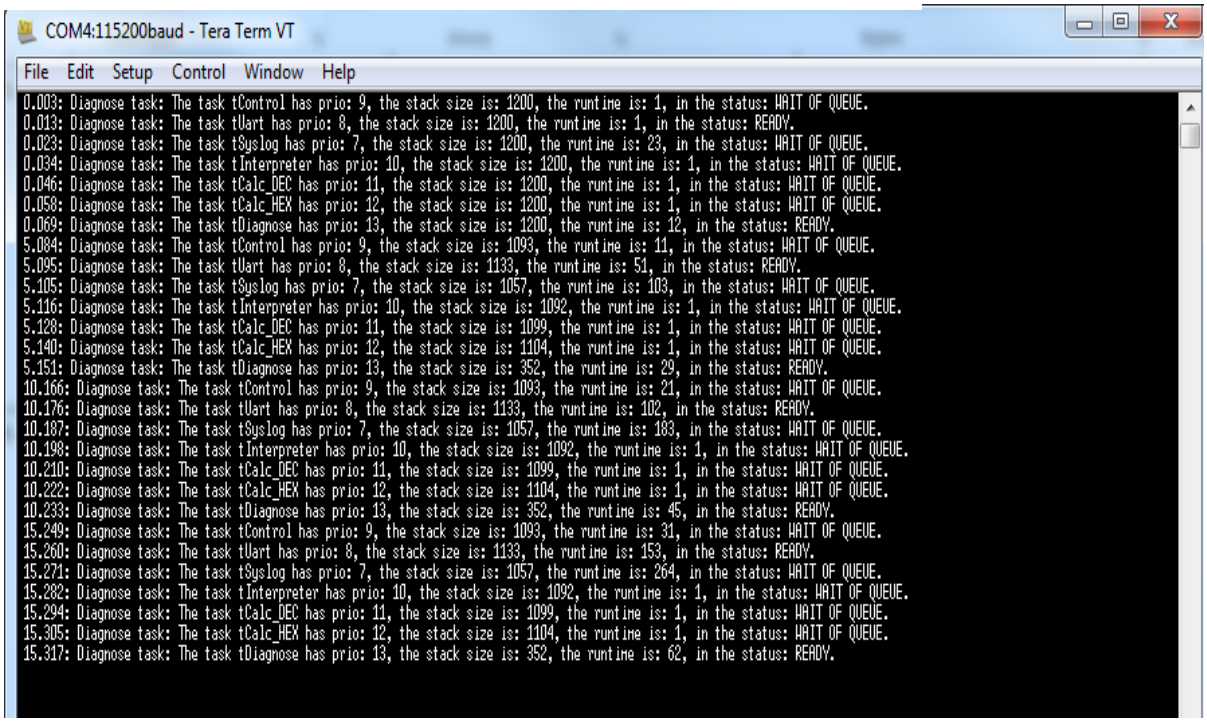
RTOS	*Printscreen*	***Resultaat***
FreeRTOS	Printscreen 23.1	***V***
MicroC/OS-II	Printscreen 23.2	***V***

De wrapper functie : *prTaskGetState()* werkt correct.

Printscreen 23.1 Output van Test 23 - FreeRTOS



Printscreen 23.2:.Output van Test 23– MicroC/OS-II



- **Test 24.**

Opstelling:

Tijdens eerste uitvoering van de Diagnose taak van de demoapplicatie wordt de opschortte functie aangeroepen: *pr_TaskSuspendAll().* Deze functie stopt de scheduler, maar interrupt blijft enable. Daarna gaat de Diagnose taak door, aan het eind van de Diagnose taak wordt de functie *pr_TaskResumeAll()* aangeroepen. . In de Diagnose taak wordt tien keer de functie: *pr_TaskGetTickCount()* toegevoegd. Voor het printen van de output wordt *xil_printf()* functie gebruikt. De UART taak die de hogere prioriteit heeft, moet elke 0.02sec "*UART_task running*" printen. Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II. Voor de overzicht van de resultaat wordt de periode van de Diagnose taak gelijk aan 0.2 sec aangezet.

Verwachtte resultaat:

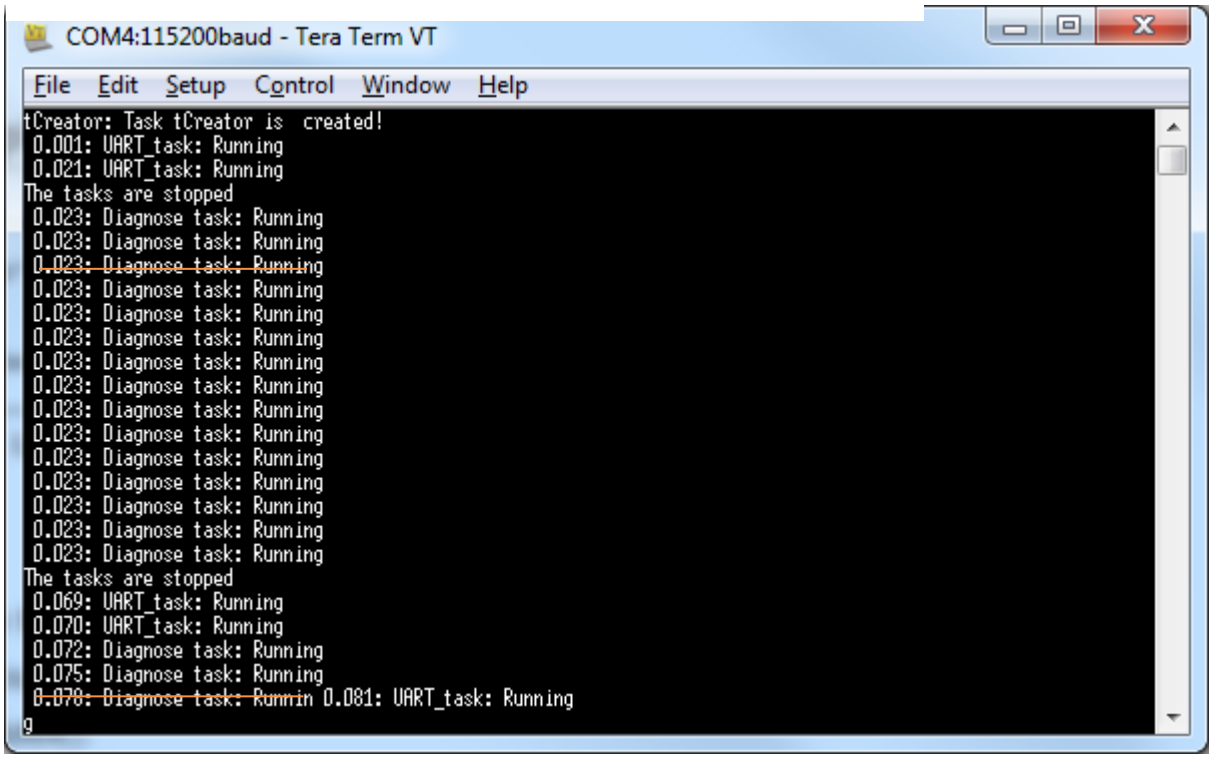
Tijdens deze test is verwacht dat na het aanroepen van de functie: *pr_TaskSuspendAll()* worden alle taken opgeschort. Er wordt alleen de Diagnose taak wordt uitgevoerd. Tijdens eerste tien uitvoeringen zal de UART taak de Diagnose taak niet breken. Over 10 uitvoering van de Diagnose taak wordt scheduler weer opgestaart en de uitvoering van de Diagnose taak wordt door de UART taak gebroken. De tijd van het systeem moet doorgaan dus na de eerste tien uitvoering van de Diagnose taak moet de UART taak een vertraging hebben.

Resultaat

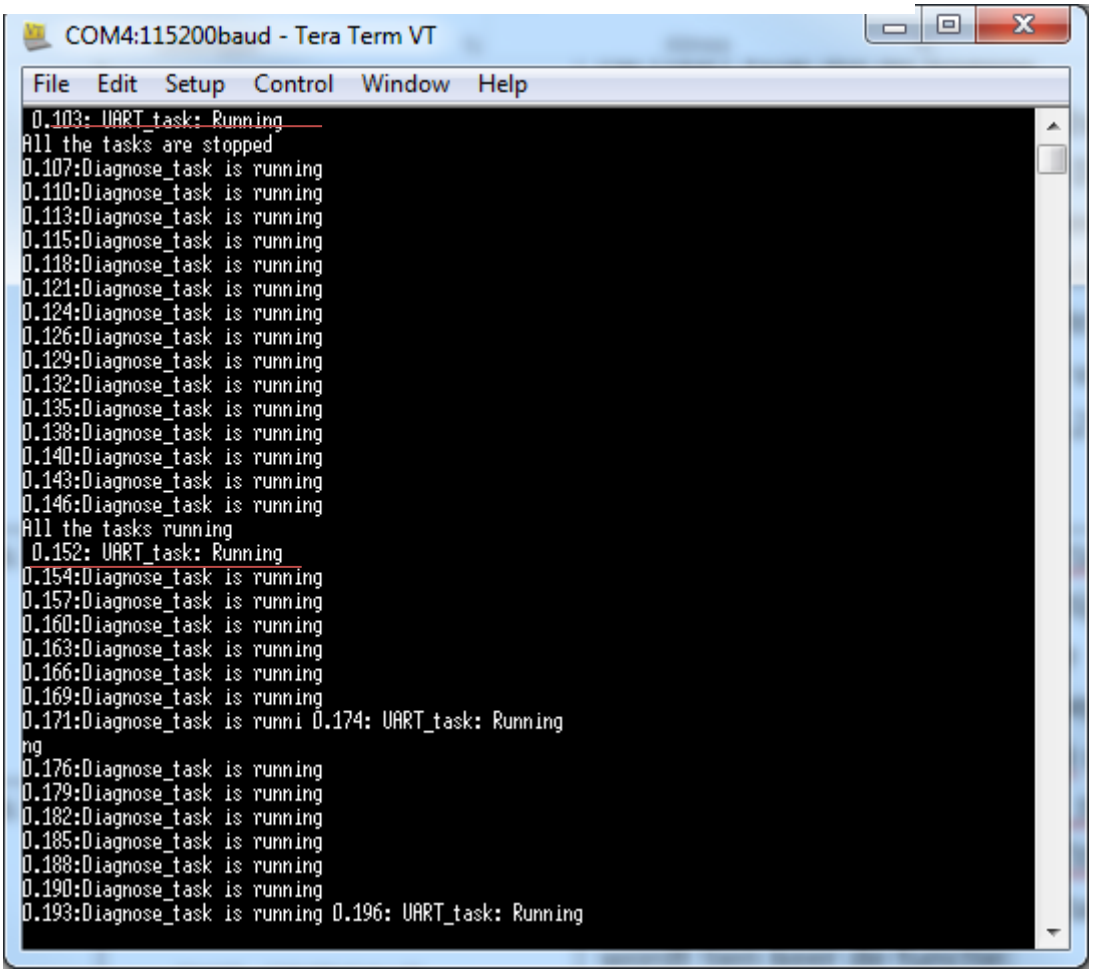
De wrapper functies : *pr_TaskSuspendAll() en pr_TaskResumeAll()* werken correct.

RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Printscreen 24.1	***V***
MicroC/OS-II	Printscreen 24.2	***V***

Printscreen 24.1:Output van Test 24– FreeRTOS



Printscreen 24.2:Output van Test 24– MicroC/OS-II



- **Test 25.**

Opstelling:

Tijdens deze test In Diagnose taak van de demoapplicatie wordt tijdens de eerste uitvoering de functie *pr_ENTER_CRITICAL()* aangeroepen. Daarna gaat de Diagnose taak door, aan het eind van de Diagnose taak wordt de functie *pr_EXIT_CRITICAL()* aangeroepen. In de Diagnose taak wordt tien keer de functie: *pr_TaskGetTickCount()* toegevoegd. Voor het printen van de output wordt *xil_printf()* functie gebruikt. De UART taak die de hogere prioriteit heeft, moet elke 0.02sec "*UART_task running*" printen. Deze test moet worden uitgevoerd eerst onder het FreeRTOS en daarna onder het MicroC/OS-II.

Verwachtte resultaat:

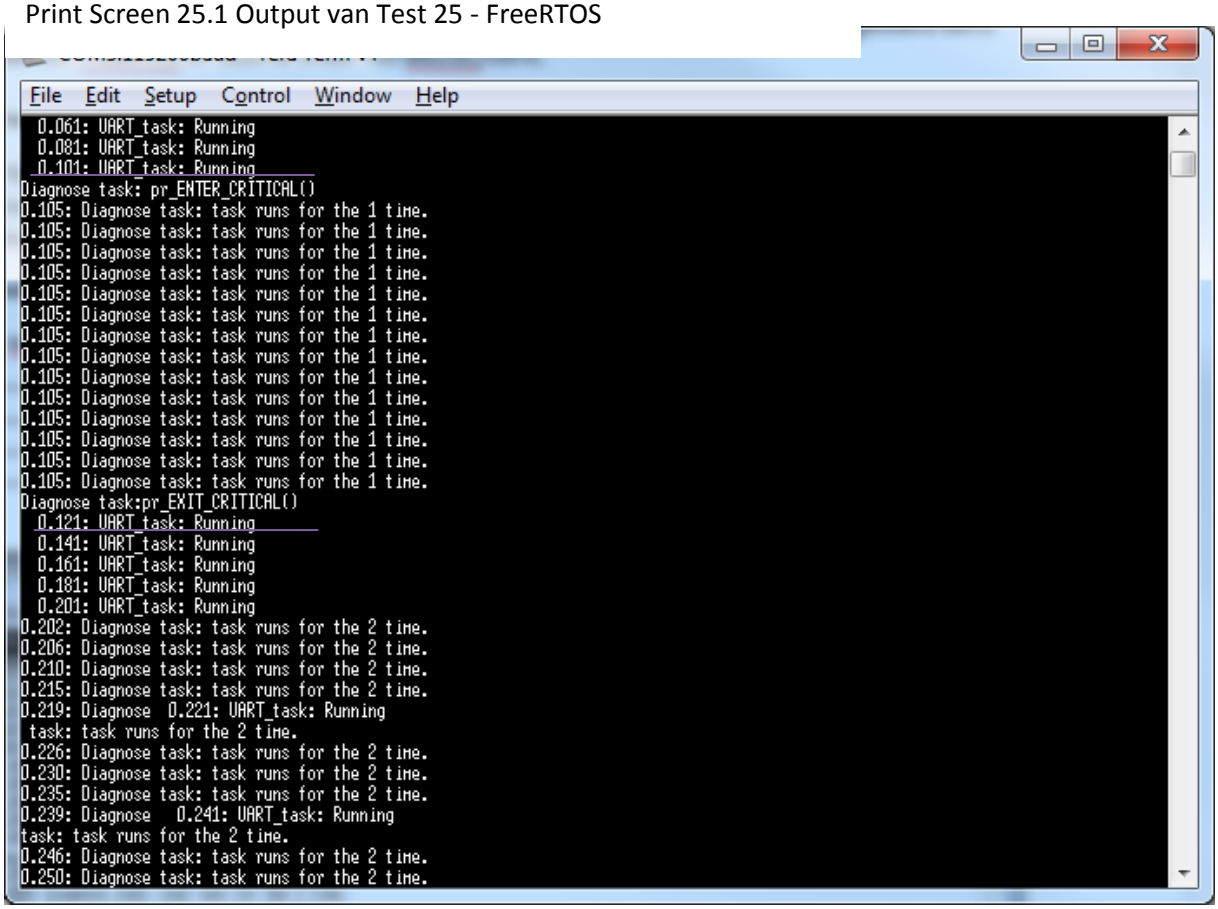
De tijd die tien keer wordt uitgeprint, moet dezelfde blijven, want alle interrupten zijn disable. Ook is verwacht helemaal geen output van de andere taken, want de scheduler van het RTOS wordt gestopt. Tijdens de volgende uitvoering van de Diagnose taak wordt de tijd veranderd. De tijd van het systeem moet niet doorgaan dus na de eerste tien uitvoering van de Diagnose taak moet de UART taak geen vertraging hebben. Voor de overzicht van de resultaat wordt de periode van de Diagnose taak gelijk aan 0.1 sec aangezet.

Resultaat

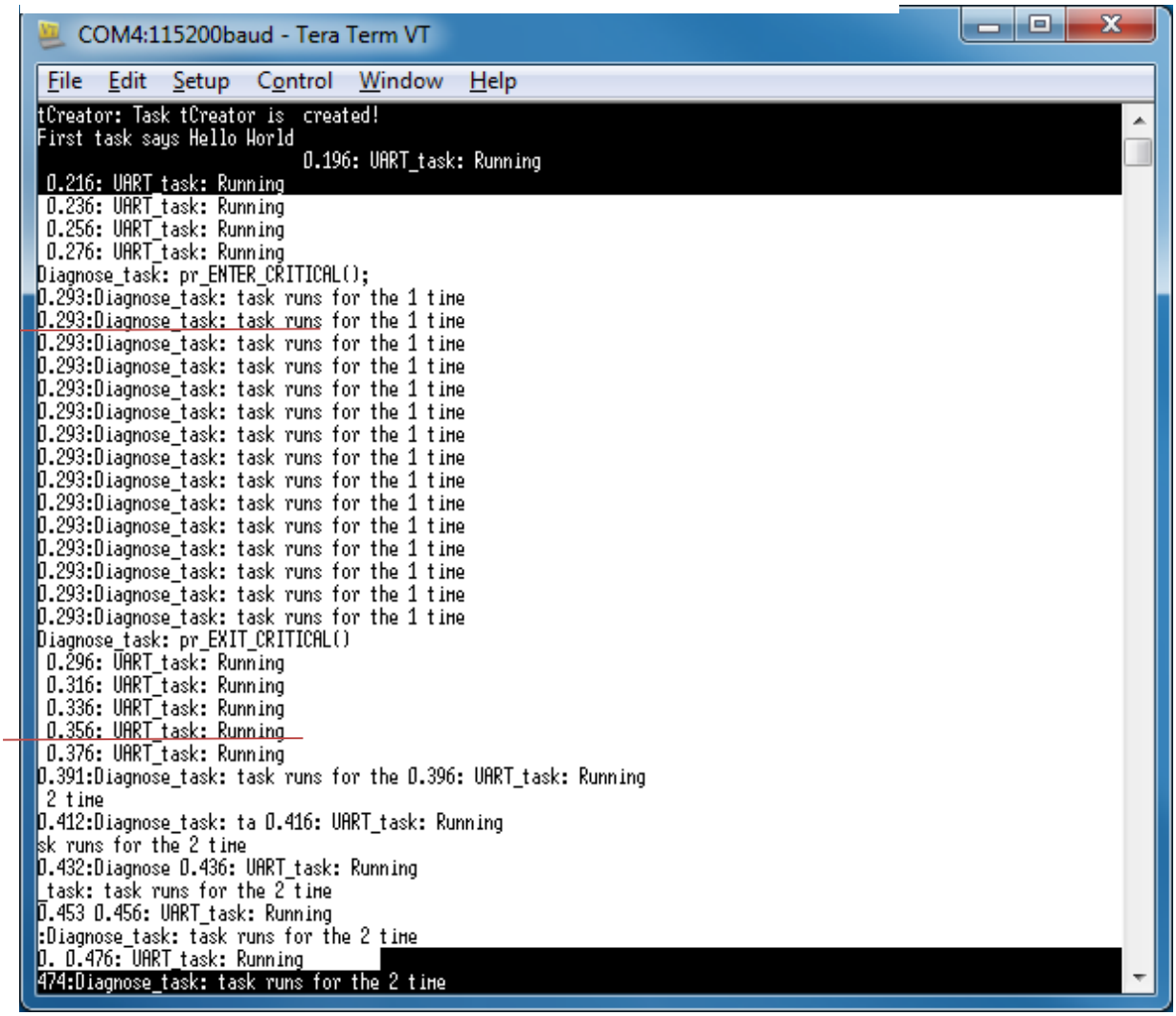
RTOS	***Printscreen***	***Resultaat***
FreeRTOS	Print Screen 25.1	***V***
MicroC/OS-II	Print Screen 25.2	***V***

De wrapper functies : *pr_ENTER_CRITICAL() en pr_EXIT_CRITICAL()* werken correct.

Print Screen 25.1 Output van Test 25 - FreeRTOS



Printscreen 25.2:Output van Test 25– MicroC/OS-II



Bijlage XXII. Afstudeerplan

Afstudeerplan

Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2014-1.1 (start uiterlijk 10 februari 2014)
Startdatum uitvoering afstudeeropdracht: 10-02-2014
Inleverdatum afstudeerdossier volgens jaarrooster: 6 juni 2014

Studentnummer: 11104945
Achternaam: mw Brodskaya
Voorletters: M
Roepnaam: Maria
Adres: Lumumbasingel 13
Postcode: 2622 ED
Woonplaats: Delft
Telefoonnummer: 015-2853233
Mobiel nummer: 062-4701050
Privé emailadres: maria.brodskaya@gmail.com

Opleiding: Technische Informatica
Locatie: Delft
Variant: voltijd

Naam studieloopbaanbegeleider: -
Naam begeleidend examinator: J.J. Smeets
Naam tweede examinator: J.J. Visser

Naam bedrijf: Procentec
Afdeling bedrijf: Research and Development
Bezoekadres bedrijf: Turfschipper 41
Postcode bezoekadres: 2292 JC
Postbusnummer:
Postcode postbusnummer:
Plaats: Wateringen
Telefoon bedrijf: 017-4671800
Telefax bedrijf: 017-4671801
Internetsite bedrijf: www.procentec.com

Achternaam opdrachtgever: dhr Dumay
Voorletters opdrachtgever: E
Titulatuur opdrachtgever
Functie opdrachtgever: Head of R&D
Doorkiesnummer opdrachtgever:
Email opdrachtgever: aonderwater@procentec.com

Achternaam bedrijfsmentor: dhr Silva
Voorletters bedrijfsmentor: P
Titulatuur bedrijfsmentor:
Functie bedrijfsmentor: R&D ingenieur
Doorkiesnummer bedrijfsmentor:
Email bedrijfsmentor: edumay@procentec.com

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:

Het veranderen van het eigen operationele systeem van de COMbricks naar standaard Real Time Operation Systeem.

Opdrachtomschrijving

1. Bedrijf

Het bedrijf Procentec specialiseert zich in alle producten en diensten van Profibus en Profinet technologie. Profibus (Process Field Bus) is een standaard voor veldbus communicatie in de automatiseringstechniek. Profibus wordt gebruikt om automatiseringsapparatuur, zoals sensoren, controllers, in één systeem te combineren. Profinet is een standaard voor industriële automatisering met het gebruik van computernetwerk. Het bedrijf ontwikkelt producten om productie-processen van eindgebruikers te optimaliseren. Procentec maakt componenten die nodig zijn om een industrieel meet- en stuurbaar netwerk in te richten. De producten van Procentec is toepasbaar op allerlei installaties zowel de fabrieks- als utiliteitsvloer. Procentec biedt ook training en ondersteuning aan de eindgebruikers.

Procentec heeft twee kantoren: het hoofdkantoor in Wateringen en een verkoopkantoor in Duitsland. In het hoofdkantoor werken ongeweer 30 mensen. Het bedrijf oriënteert zich op de markt zowel binnen Nederland als buiten. Zij heeft officiële distributeurs in negen landen.

2. Probleemstelling

Een van de producten die door het bedrijf Procentec is ontwikkeld, is COMbricks. De COMbricks maakt de naadloze communicatie op hoge snelheid tussen PROFIBUS DP en PROFIBUS PA mogelijk. PROFIBUS DP (Decentralized Peripherals) en PROFIBUS PA (Process Automation) zijn verschillende communicatieprotocollen. De software van de COMbricks PA Link heeft een eigen gemaakt operating systeem dat processen scheduled, het berichten verkeer regelt tussen processen en andere functies. Als meer veldbussen via één COMbricks worden communiceren dan ontstaat een vertraging in het systeemwerk, omdat alle processen sequentieel zijn uitgevoerd. Daarom wil Procentec het eigen operating systeem vervangen door een standaard Real Time Operation Systeem, dat mogelijkheid biedt om de processen te parallelliseren.

3. Doelstelling van de afstudeeropdracht

Het doel van de afstudeeropdracht is het kiezen van een bestaande Real Time Operation Systeem op basis van het onderzoek. Het systeem moet aan het eisen van Procentec voldoen.

4. Resultaat

De resultaten van de afstudeerstagestageopdracht zijn:

1. Het onderzoeksrapport over bestaande Real Time Operation Systemen
2. Het prototype voor het splitsen van de processen van de COMbricks PA Link

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

Tijdens de afstudeerstage moeten een aantal activiteiten worden uitgevoerd om dit project tot een goed einde te kunnen brengen.

1. Definitiefase – 15 dagen

- Plan van aanpak schrijven
- Onderzoek van het PROFIBUS PA protocol
- Onderzoek van de functionaliteit van de COMbricks

2. Ontwerpfase – 35 dagen

- Vaststellen van de features die nodig zijn voor de COMbricks PA Link bij de toepassing van een Real Time Operation Systeem.
- Vaststellen van de eisen voor een Real Time Operation Systeem
- Onderzoek en analyze van de bestaande Real Time Operation Systemen en hun mogelijkheden
- Keuzen van de Real Time Operation Systemen
- Het onderzoeksrapport schrijven
- Het concept van het verslag schrijven

3. Implementatiefase – 35 dagen

- Maken testaplicatie's, die de werking van de features van het geselecteerde Real Time Operationg Systeem die nodig zijn voor de COMbricks PA Link aantonen
- Een start maken met het implementeren van het geselecteerde Real Time Operation Systeem in de software van de COMbricks PA link.
- Het verslag schrijven

4. Evaluatie - 15 dagen

- Testen
- Analyseren van de testresultaten
- Het verslag afmaken

6. **Op te leveren (tussen)producten**

Tijdens de stageperiode moeten voor het bedrijf Procentec de volgende producten worden opgeleverd:

- Plan van Aanpak.
- Onderzoeksrapport van de bestaande Real Time Operation Systemen.
- Het prototype voor het splitsen van de processen van de COMbricks PA Link

1. Te demonstreren competenties en wijze waarop

A1 Analyseren van het probleemdomein
A3 Achterhalen van behoeften van belanghebbenden

A5 Opstellen van systeemeisen
Deze competenties zullen terug worden gevonden in het opstellen van een plan van aanpak en het vaststellen van de systeemeisen.

C8 Ontwerpen van een technisch informatie systeem **D16** Het realiseren van software **D17** Testen van software systemen Tijdens de stage moet de software worden ontwikkeld, die moeten worden realiseerd en getest.

G1 Praktische aspecten hanterenin projecten
Deze competentie is belangerijk bij het opstellen van het PvA.