SmartPram

-Den intelligente barnevogn

Gruppe 8

Projektrapport

Afleveringsdato: 20-12-2013

Vejleder: Michael Alrøe

Navn	**Studienummer**	**Underskrift**
David Buhauer	201270749	
Jonas Jussila	201270916	
Kristian Kristensen	201271039	
Mads Krogsgaard	201270392	
Anders Lassen	201270933	
Kjeld Laursen	201270042	
Sander Louring	201270763	
Joachim Larsen	201270761	

1 Resumé - (Joachim)

Dette projekt er en del af et semesterprojekt, som skal inkludere adskillige aspekter af dette semesters generelle læringsmål. De krav, som projektet skal indeholde er, at projektet skal gøre brug af mere end en sensor og mere end en aktuator, samtidigt med, at brugerinteraktion skal være til stede. Disse nævnte dele, er selve formålet for projektet. Vores specifikke produkt er udviklet ud fra ideen, om en automatisk barnevogn og vil have nogle automatiske og manuelle funktioner. Selve produktnavnet er "Smart-Pram", og dette produkt er i stand til at vugge en baby i søvn, ved hjælp af rolige vug samt en roterende uro – alt sammen noget, som kontrolleres af brugeren via et touch-display. Smart-Pram er udstyret med en mikrofon og en temperatursensor. Vi har valgt at implementere trådløs forbindelse mellem vores Pram og touch displayet via SPI-protokollen. Selve barnevognen får sin spændingsforsyning fra et påmonteret batteri. Softwaren er i dette projekt skrevet på to platforme. Brugerside-delen er generelt set opbygget ud fra princippet "event-driven" programmering, som gør brug af både events og tråde. Barnevognsdelens software tager derimod mere udgangspunkt i generel hardwarenær og strukturel C-kode. Igennem projektet har vi arbejdet med en iterativ fremgangsmetode – en fremgangsmåde, som sikrer resultater i høj kvalitet. Ydermere er der blevet sørget for individuelle tests af hver enkelt hardware -og softwaredel, før der blev gået videre med næste stykke arbejde. Noget som minimerer ophobning af fejl i sidste ende. Vi har benyttet Scrum som værktøj til at holde styr på folks enkelte arbejde, hvilket har gjort det individuelle arbejde mere attraktivt og samtidig mere effektivt. Vores produkt er implementeret således, at det opfylder kravene givet for dette projekt. Produktet lever op til de for semesterprojektet opstillede krav.

2 Abstract - (Mads)

This project is an outcome of a semester project, which includes several aspects of this semester's general learning requirements. The project requirements state that there must be two or more sensors, as well as two or more actuators, operational in the system. At the same time, it must interact with the user/users through user interaction. The mentioned so far is the general purpose of this project. Our specific product has been developed on the idea of an automatic nanny pram, which have several automatic and semi-automatic functions. Our product's name is "Smart-Pram" and it is able to lull a baby to sleep with gentle rocking and by the use of a mobile – all controlled by the user through a touch-display. Smart-Pram is equipped with a microphone and a temperature sensor. We have chosen to implement wireless connection between our Pram and the touch display through a SPI device. The Pram-side of the product draw its power from a battery. The software is written on two platforms. The user-side unit has generally been written in an event-driven structure, using both events and threads. The pram-side unit has been written in a more sequential way where a close line to hardware is met. Generally throughout the project, we have been working with an iterative approach – an approach that ensures high quality results. Furthermore, we ensured a high quality product with individual testing of any piece of hardware or software before continuing with the next piece. By the use of Scrum we managed to keep track of people's work – making the individual work more attractive and less inefficient. Our product is implemented fulfilling the requirements of this project – at the same time it has proven itself to be working. With our product we are able to help parents nurturing their baby.

Indholdsfortegnelse

3 Forord - (Joachim & Mads)

Dette projekt er baseret på projektoplægget udstykket af skolen og er et skoleprojekt i forbindelse med 3. semester E3PRJ på Ingeniørhøjskolen Aarhus.

I denne opgave skal notationen forstås således, at navnet ude for titlen repræsenterer den eller de personer, der har arbejdet på det pågældende emne. Navnet/navnene i parenteserne repræsenterer den eller de personer, der har skrevet dette afsnit og været en del af det. I de tilfælde hvor et afsnit kun er efterfulgt af en parentes med navnet/navnene, har personerne i den parentes skrevet selve afsnittet, men alle har været med til arbejdet.

Der er blevet anvendt forskellige indskrivningsmetoder af gruppen, som vil blive forklaret her. I Zigbee afsnittet er Analyse, Design og Implementering blevet skrevet ud i et.

For Pram delen er Analyse og Design blevet skrevet samlet og Implementering efterfølgende. For Devkit er Analyse, Design og Implementering blevet skrevet i tre separate punkter. Begrundelsen for dette er, at det har givet bedst mening for de enkelte delpunkter at blive skrevet således.

4 Indledning - (Joachim)

Emnet for dette projekt har været frit, men med visse forudsatte krav og fokusområder. Emnet er valgt af gruppen og er trådløs kommunikation og styring af en barnevogn – en intelligent barnevogn, hvor forældrene skal kunne monitorere deres baby. Grunden til at emnet faldt på dette var, at vi i gruppen så en væsentlig mulighed for at udvide princippet omkring pasning af en baby. Især på de tidspunkter, hvor babyen oftest ligger uden opsyn – nemlig i en barnevogn. I forvejen findes der selvfølgelig babyalarmer, men disse giver ikke en fuldt ud beskyttende tilværelse, hverken for babyen eller forældrene. Ved at overvåge mere end bare lyden fra babyen og ydermere lade forældrene interagere herpå, kan der højst sandsynligt opnås en større sikkerhed for babyen. Overvågning går ud på at opsamle data fra babyen, herunder lyd og temperatur. Disse sendes trådløst til en komponent med brugergrænseflade, som moren eller faren så kan skride til handling ud fra. Ud fra disse parametre kan forældrene vælge at starte manuel vugning af barnevognen eller start af en uro. En automatisk funktion er også mulig og vil selv kunne styre de to aktuatorer ud fra de forskellige parametre. Dette system styres af en central kontroller, der via SPI-protokollen kommunikere med en transceiever, der trådløst kommunikerer med en tilsvarend transceiever på barnevognen, som igen via SPI-protokollen taler med en kontroller, der kontrollerer diverse sensorer og aktuatorer. Denne rapport vil tage udgangspunkt i en prototype af en barnevogn med den nævnte funktionalitet implementeret.

5 Opgaveformulering - (Sander) Kristian & Jonas

Som nybagt forælder vil man blive præsenteret for mange nye opgaver for at passe bedst muligt på sit barn. Pludselig skal man vænne sig til at skulle op midt om natten og trøste en grædende baby og derved miste sin egen søvn, man skal skifte ble og man skal se sig selv med en barnevogn hvor end man går hen. Selvom der naturligvis også følger overvejende mange dejlige stunder med, så må man tilvænne sig en ny livsstil, hvor man måske ikke altid har så meget tid til sig selv, som man har været vant til.

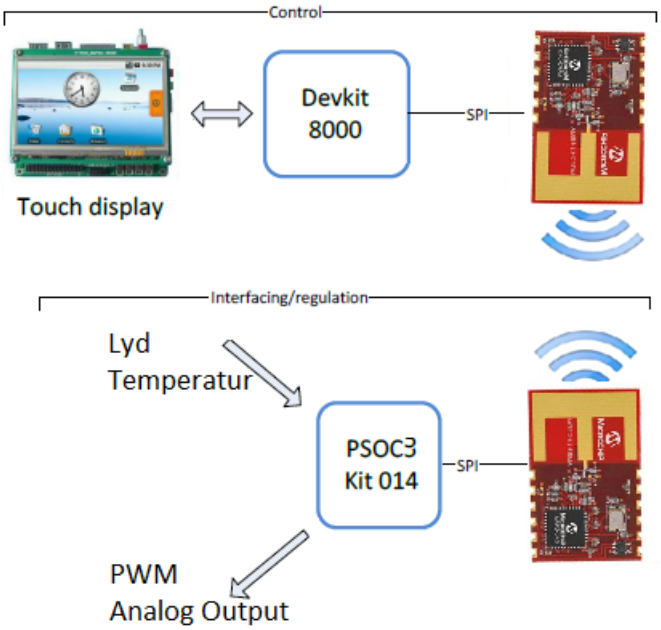
Figur 1: SmartPram

Derfor ser mange forældre sig efter produkter, der kan lette dette arbejde, så de kan få lidt mere tid til sig selv, samtidig med at de kan være sikre på, at deres baby ligger trygt og forsvarligt. Mange har f.eks. valgt at anskaffe sig en babyalarm, så de kan høre, når deres baby græder, men med hjælp fra moderne teknologi stopper mulighederne ikke her. Det er i den forbindelse, at vi præsenterer: SmartPram – Den intelligente barnevogn.

SmartPram er en barnevogn, der er udstyret med en række målere og motorer der kan automatisere forskellige opgaver. Barnevognen har en mikrofon, en temperaturmåler, et hygrometer og en bevægelsessensor, der alle sender oplysninger trådløst til en håndholdt kontrolenhed, der er udstyret med et brugervenligt touch display. Med denne kontrolenhed kan man derved holde øje med, om ens barn er vågnet og græder, om det ligger uroligt, om temperaturen i barnevognen bør ændres, eller om det trækker op til regn. Kontrolenheden giver på baggrund af disse oplysninger mulighed for at starte en række funktioner i barnevognen gennem kontrolinterfacet. Hvis barnet er vågnet kan man starte en beroligende uro, således at barnet kan falde i søvn igen, og hvis dette ikke er nok, kan en motor sættes i gang med at vugge barnet. Dette kan enten gøres manuelt, eller et automatisk program kan igangsættes. Temperaturmåleren og hygrometeret kan give oplysninger om, om man bør gå ud til barnevognen og skifte dyneplacering eller påsætte et regnslag. SmartPram kan altså gøre livet nemmere for forældre til babyer, uden at man mister nærværet, da man altid kan holde øje med barnet fra enheden, og derved kan man få overskuddet til at fokusere mere på de glade stunder med barnet.

6 Systembeskrivelse - (Mads)

Systemet eksisterer for, at kunne få en baby til at falde i søvn/berolige den. Brugeren har mulighed for at definere, hvordan systemet skal fungere via sit Devkit8000. Der kan indstilles vuggetid, urotid og lydstyrke, som systemets automatiske funktioner reagerer på. Yderligere kan brugeren indstille den temperatur-værdi som han/hun ønsker barnevognen max/min må være. Brugeren kan tænde og slukke for den automatiske funktion eller selv køre en cyklus af motorerne ved at trykke på de manuelle knapper.



Figur 2: Systembeskrivelse

Kommunikationen mellem de to enheder foregår vha. ZigBee moduler som anvender SPI protokollen. De to ZigBee moduler fungerer således, at når de modtager data, sender de et interrupt til den enhed, som de sidder ved, som indlæser de data, der er kommet fra den anden enhed.

7 Krav - (Mads & Sander)

Vores system skal via ZigBee kunne kommunikere imellem de to enheder henholdsvis Devkit8000 og PSoC3. Devkit8000 tilgås af Bruger og PSoC3 af Baby. Dette fremgår af figur 3.

Figur 3: Usecasediagram

Man skal kunne tænde og slukke for systemet samt køre uro og vugning i manuel mode og automatisk mode. Systemet skal hente data fra Baby og informere brugeren om dette, og brugeren skal kunne indstille grænseværdier. Systemet behandler de data, som den får fra Baby, og kører vugning/uro automatisk, såfremt dette er aktiveret. Kort sagt skal vores prototype være i stand til at opfylde disse funktionelle krav (Der refereres til Use Casene i projektdokumentationen afsnit 1.5) i et passende omfang, som så testes ud fra vores kravspecifikation.

8 Projektafgrænsning og fremtidigt arbejde - (Sander)

Da formålet med projektet har været at løse et problem på en fagligt relevant måde – men ikke at skulle sælge produktet efterfølgende, har vi i gruppen fra starten af valgt, at det færdige produkt skulle være i form af en funktionel prototype. Dette er også vurderet på baggrund af de på forhånd stillede krav om, at produktet skulle bestå af PSoC3 og Devkit8000, hvilket ikke direkte egner sig til et kommercielt produkt. Dette har medført, at vi ikke nødvendigvis har forsøgt at finde de billigste komponenter til produktet, men at fokus har været på funktionalitet. Dog har der hele vejen igennem været et stor fokus på at lave gode og strukturerede løsninger, som man senere vil kunne arbejde videre på.

Projektet lægger op til mange muligheder for udvidelse i form af flere typer af sensorer og aktuatorer og i form af opdateret software til at håndtere disse moduler. Vi valgte fra start af at inddele projektet i flere delleveringer, hvilket kan ses i afsnittet "implementeringsliste" i kravspecifikationen afsnit 1.4. Her fokuserede vi som udgangspunkt kun på den første dellevering, der består af grundkonceptet med en mikrofon og en temperaturføler samt en vuggemotor og en uromotor. Disse moduler er valgt til første dellevering, da de er nok til at give et funktionelt produkt, der viser de essentielle principper i SmartPram-konceptet. Til fremtidigt arbejde med de resterende delleveringer kan bl.a. nævnes en fugtmåler, en vægtmåler og et temperaturreguleringssystem, der vil kunne udvide automatiseringsfunktionaliteten i produktet.

Visse projektafgrænsninger opstod også undervejs i projektet - særligt i form af valget om ikke at implementere trådløs overførsel af lyd fra PSoC3 til Devkit8000. Det var nemlig oprindeligt planen, at lyden skulle streames mellem de to enheder, men da arbejdet begyndte, viste det sig at være en større opgave end først antaget. De første planer var at afspille lyden fra Devkit8000 gennem en DAC, men dette ville dog kræve en meget stor timings-præcision, og det ville desuden være uhensigtsmæssigt at foretage interrupts på DevKittet så ofte, som live streaming af lyd ville kræve. Vi valgte derfor at droppe denne feature, da den ville kræve for meget arbejde i forhold til dens bidrag til det samlede projekt. Den vigtigste del af lydoverførslen er nemlig, at systemet skal kunne registrere, hvis der opstår et for højt lydniveau, så derfor valgte vi at analysere lyden på PSoC3 og blot overføre en værdi for lydniveauet. Dette gav os mulighed for at fokusere mere på features, der har større betydning for brugeroplevelsen.

9 Projektbeskrivelse

9.1 Projektgennemførelse - (Mads)

Tidsplan for gruppe 8

nr:	Kategori	Opgave:	Status:	Prioritet:	Planlagt Start:	Faktisk start	Planlagt slut:	Faktisk slut:
1	Dokumentation	Projektformulering	Afsluttet	Normal	10. september 2013	10. september 2013	16. september	10. september 2013
2	Administration	Overordnet Tidsplan	Afsluttet	Lav	10. september 2013	10. september 2013	16. september	10. september 2013
3	Dokumentation	Kravspecifikation	Afsluttet	Høj	17. september 2013	17. september 2013	1. Oktober	10. Oktober 2013
4	Dokumentation	Accepttest	Afsluttet	Høj	17. september 2013	17. september 2013	1. Oktober	10. Oktober 2013
5	Dokumentation	Teknologiløsninger	Afsluttet	Høj	24. september 2013	17. september 2013	1. Oktober	10. Oktober 2013
6	Dokumentation	Systemarkitektur for SW/HW	Afsluttet	Normal	24. september 2013	3. Oktober 2013	22. Oktober	14. November 2013
7	Software design	Udvikling af Software	Afsluttet	Normal	10. oktober 2013	10. oktober 2013	11. december	12. december
8	Hardware design	Udvikling af Hardware	Afsluttet	Normal	10. oktober 2013	10. oktober 2013	11. december	12. december
9	Test	Udførelse af Accepttest	Afsluttet	Normal	26. november 2013	16. December 2013	10. december	16. December 2013
10	Dokumentation	Aflevering af projekt	Afsluttet	Normal	10. december 2013			20. December 2013
11	1. Review	Accepttest og Kravspec.	Afsluttet	Høj	11. oktober 2013	11. oktober 2013	11. oktober	11. oktober 2013
12	2. Review	Punkt 6 + V1.0 Af punkt 7+8	Afsluttet	Høj	15. november 2013	15. November 2013	15. November	15. November 2013
13	Test	Modulsammenkobling	Afsluttet	Høj	12. december 2013	12. december 2013	12. december	13. december 2013
14	Dokumentation	Projektrapport	Afsluttet	Høj	1. december 2013	1. december 2013	19. december	19. december

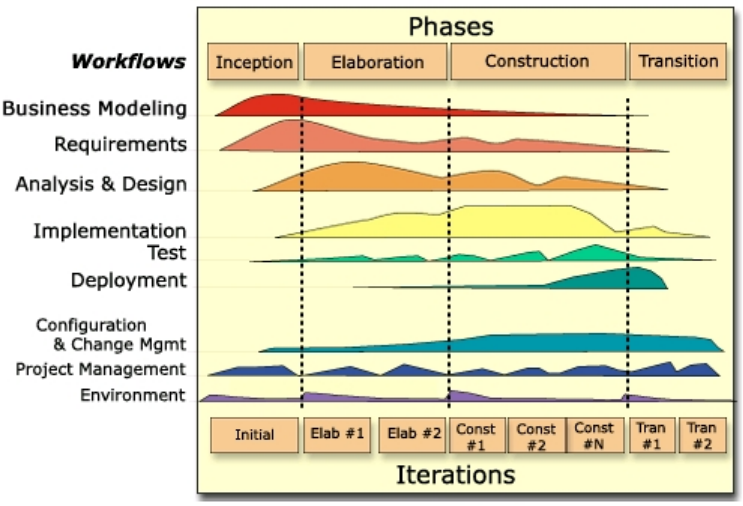
Figur 4: Tidsplan.

Vi er i gruppen startet ud med at anvende en tidsplan, som har givet os mulighed for at overholde vores deadlines for review og hjulpet med at skabe et overblik over projektet. Efter 2. review, begyndte vi at benytte Scrum i et større omfang til prototypeudviklingen og projektrapport. Dette har primært været for at holde styr på projekts indhold (Der henvises til afsnit 9.2.2, der fortæller om Scrum som et af projekts metoder). Vi har benyttet os af tre sprint, hvoraf det første var implementering, det andet omhandlede rapport, mens det sidste sprint omhandlede dokumentation. De to sidste sprint var mest omfattende, men de viste sig virkelig at være en kæmpe fordel, idet det gav en særdeles god struktur. Hvis man kigger på tidsplanen ser man, at vi generelt har været gode til at overholde de planlagte punkter, men nogle er skredet lidt. Som udgangspunkt regnede vi med, at der skulle være fem mand omkring software og tre mand omkring hardware. Det viste sig dog, at hardwaren og softwaren generelt havde et større sammenspil end forventet, så folk har generelt arbejdet med begge dele.

9.2 Metoder

9.2.1 RUP - (Joachim)

Rational Unified Process. Denne proces er en iterativ proces, der overordnet består af fire faser: forberedelse(inception), uddybelse(elaboration), konstruktion(construction) og overdragelse (transition).



Figur 5: RUP Process

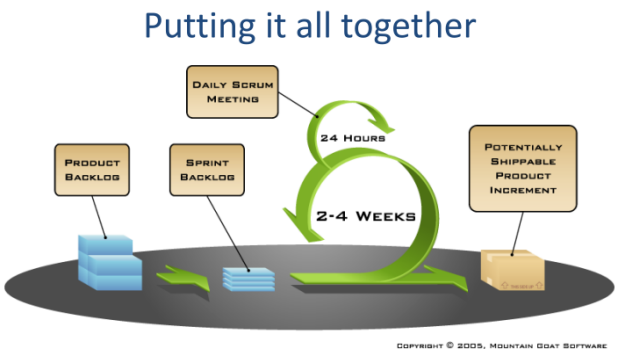
Der er i dette projekt kun benyttet de tre første fase. Den sidste fase (overdragelse) er ikke med, da vores produkt ikke skal overdrages til produktion. Der er derimod udelukkende tale om en prototype. Hele essensen ved denne iterative arbejdsproces er, at alle faser foregår i iterationer – altså gentagelser. Dette er modsat vandfaldsmodellen, hvor en arbejdsgang (workflow) afsluttes, før der gås videre til det næste. I RUP arbejdes der med hver arbejdsgang i hver fase – dog i vidt forskellige omfang (som det også ses af figur 5).

Det er netop denne arbejdsproces, som vi i denne projektgruppe har lagt os op ad. Her har vi nemlig arbejdet med forskellige arbejdsgange i de forskellige faser. Analyse og design er bare et eksempel. Her har man i den uddybende fase (elaboration) arbejdet meget med dette og næsten færdiggjort det, men man foretager stadigvæk ændringer her i konstruktionsfasen. Det samme scenarie gør sig gældende ved krav. Kravene fastlås gerne i den forberedende fase (inception), men ændres typisk i de efterfølgende faser også – et krav kunne fx være for højt sat, eller et krav kunne helt mangle. Alt dette er noget, som vi helt og aldeles har benyttet os af, og som vi har haft stor glæde af i dette projektforløb. Det har givet meget mere mening for os at anvende denne iterative arbejdsproces frem for fx den mere fastlåste vandfaldmodel.

9.2.2 Scrum - (Kristian)

Ved projektledelsen af dette projekt benyttede vi os af Scrum.

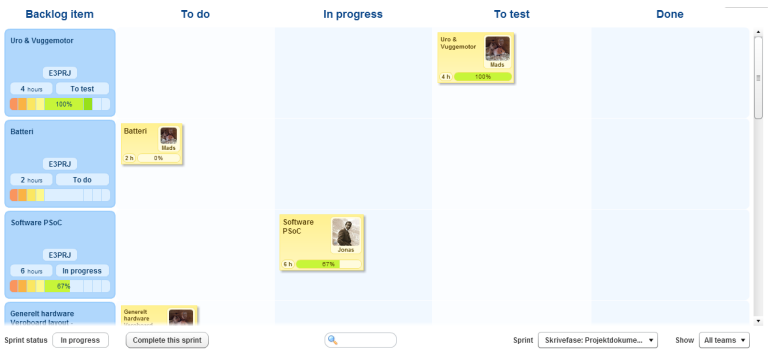
Ved projekter hvor der benyttes SCRUM betragtes gruppen der skal udvikle et produkt, som et selvorganiserende team.



Figur 6: Scrum model

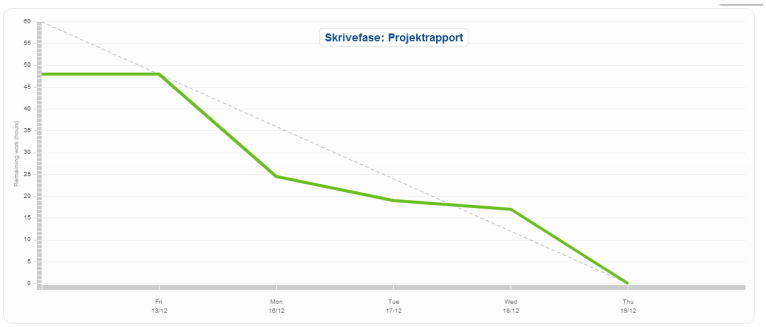
Indbyrdes i teamet havde vi roller, som var med til at strukturere gruppen og styre projektarbejdet i den rigtige retning. Kundens krav opstilles i en Product Back-log, som udgør delelementer der tilsammen giver det færdige produkt. SCRUM har som udgangspunkt en Product Backlog, og var her, at vi skrev alle arbejdsopgaver som software- og hardwareudvikling. Disse blev inddelt efter prioritering og fik estimeret en tid. Til hver Sprint oprettede vi en Sprint Backlog, som vi tidsestimerede og uddelte til de enkelte medlemmer i Teamet. Under hver Backlog Item/Sprint Backlog angav vi hvor lang tid hver opgave tog. Arbejdsprocessen til at opnå målet var igennem Sprint. Vi blev enige i gruppen om at lave korte og intense Sprint på en uge ad gangen. Sprint blev både brugt under udviklings-, implementerings- og dokumentationsfasen til at få udført alle opgaverne i vores Product Backlog. Vi benyttede os af et program kaldet Scrumwise[1]. Dette er et interaktivt værktøj, som har alle de ønskede funktioner, som vi benytter os af ved SCRUM projektledelse. I sammenhold med arbejdsprocessen igennem sprintet er det muligt ved hjælp af en Burndown Chart i Scrumwise at se, hvor langt vi var med processen, som det ses i figur 8.

Gennemarbejdning af vores projekt lægger meget op til at benytte sig af SCRUM, da vi først startede med at opstille os nogle forventninger og delelementer, som sammensat skulle give os en løsning på vores problemformulering. Efterhånden som vi kom længere ind i projektet, fandt vi ud af, at der skulle ændringer og tilføjelser til vores projekt. Det er her SCRUMs fleksibilitet og mulighed for udvikling og test af hver delopgave i sprint – og efterfølgende ændringer, som gør Scrum til en agil og forandringstilpassende proces.



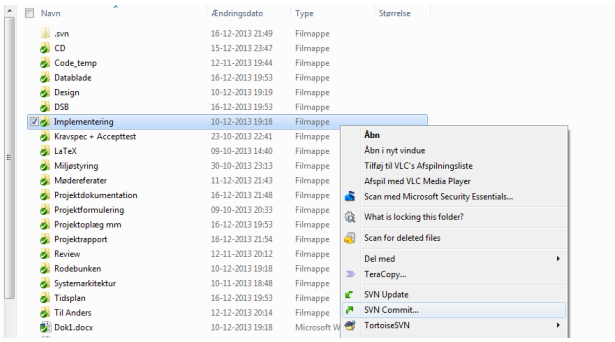
Figur 7: Billedeeksempel på tastboard under et sprint

[1] www.scrumwise.com



Figur 8: Screendump af burndown for seneste sprint

9.2.3 SVN - (Joachim)



Figur 9: Billedudsnit af SVN, der viser funktionerne "SVN Update" samt "SVN Commit".

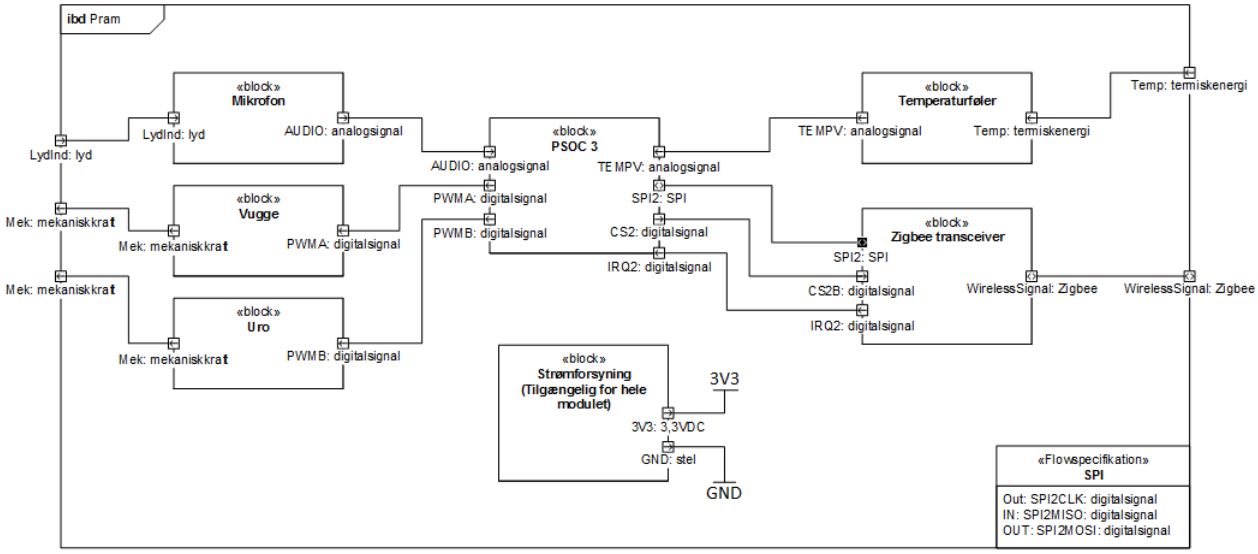
Et afgørende softwareversionerings- og revisionskontrolsystem, som vi har anvendt for vores projekt, har været Apache Subversion bedre kendt som SVN. Programmet har givet vores projektgruppe mulighed for at holde vores filer synkroniseret. Redskabet har skabt en nem måde at dele informationer og filer på i vores projekt - noget som især er fordelagtigt ved mange gruppemedlemmer som i vores gruppe. Der holdes hermed styr på, hvad folk laver. En anden stærk egenskab ved SVN er versionstyring. Kort fortalt så bliver der i SVN gemt kopier af alle versioner af ens dokumenter/kildekode, således at man altid vil være i stand til at gå tilbage til tidligere versioner, hvis nu noget skulle være gået galt.

9.3 Analyse og design HW - Joachim, Mads & Kristian

Det allerførste der blev gjort var at opstille kredsløb på et teoretisk niveau, som havde til formål at løse de enkelte opgaver ud fra IBD'et. Herefter blev der regnet teoretisk på komponentværdier mm. til de enkelte blokke under Pram (se figur 10), hvorefter kredsløbene blev indtegnet i Multisim. Når dette var gjort, gik turen til udvikling/realisering på fumlebræt – her blev der tilpasset en del, herunder komponentstørrelser. Testprogrammer skrevet på PSoC3 og målinger på oscilloskop førte i sidste ende til fungerende dele, som i helhed fungerede som Pram. Der blev efterfølgende bygget Veroboard-print af kredsløbene. Yderligere information om de enkelte delblokke, som Pram består af, kan findes i HW-implementeringen. Uddybende dokumentation findes i projektdokumentations-dokumentet. PSoC'en har udover generel mikrocontroller-funktionalitet også diverse analoge og digitale komponenter såsom OpAmps, SPI, I2C og meget mere.

9.3.1 Pram - Joachim, Mads & Kristian

Overordnet set er der tiltænkt en PSoC3 som værende en master, der styrer de enkelte aktuatorer/sensorer samt ZigBee transceiveren, der udgør slaven i denne del. Alt dette udgør altså selve Pram-delen(barnevognen). Pram'en kan overordnet ses som en slave, der styres af controlleren, der primært består af DevKit8000, der agerer master. Se figur 10 for opbygningen af Pram – vist med et IBD.



Figur 10: IBD for blokken Pram

Nedenfor beskrives de enkelte blokke af det ovenstående IBD, med henblik på designprocessen.

Mikrofon - Joachim & Mads
Denne enheds funktion er at opfange babygråd. Udfordringen lå i, at få en mikrofon til at sende fornuftig "lyd" ind på PSoC3'en, hvor denne lyd efterfølgende skulle behandles. En mikrofon har en spænding som output – et output som varierer med lyd. Dette analoge output vil som udgangspunkt svinge om nul. Her lå det første problem altså. Mikrofonens arbejdsniveau skulle tilpasses, således, at det ville stemme overens med PSoC3'ens arbejdsområde. Forskellige ideer/teorier blev drøftet, og vi kom til slut frem til en løsning. Løsningen lå i, at hæve mikrofonens arbejdsområde, således at dennes output, vil svinge omkring den halve forsyningsspænding til PSoC3'en. Hermed opnås maksimalt udnyttelse af arbejdsområdet. Denne fastsættelse er noget, som er meget væsentligt for udbygningen af vores produkt. Skulle der ske en eventuel fejl med mikrofonen, i et færdigt produkt hos kunden, skal denne nemlig nemt og enkelt kunne udskiftes, uden andre former for ændringer/tilføjelser på produktet.

Til behandling af den analoge spænding, som kommer fra mikrofonen(og som afbilleder lyd), bliver der anvendt en PSoC3. Det analoge signal fra mikrofonen er af en meget lille størrelse, hvorfor dette signal derfor skal forstærkes. Her bruges et forstærkertrin i PSoC3'en. Det forstærkede signal kommes efterfølgende ind i en ADC, der konverterer vores analoge spænding til diskrete værdier, som herefter kan behandles via softwaren. Ideen er nemlig, at kigge på lydniveauet fra mikrofonen, og hermed kunne skelne mellem baggrundsstøj mm. og babygråd. Hermed skal softwaren kunne reagere på, om babyen græder eller ej.

Vugge - Joachim & Mads
Denne aktuator har til formål, at vugge barnevognen med en vis styrke. Der skulle her designes en motorstyring, som ud fra et signal fra PSoC3'en, kan starte en DC-motor. Dette driver trin, blev tiltænkt, at skulle fungere som en "kontakt", der trækker motoren til nul. En transistor er her benyttet (se afsnit 9.7.1). Motoren skulle herefter monteres således, at dens rotation blev omformet til mekaniske svingninger, der således ville rykke i barnevognen. Her fik vi hjælp af Rasmus fra værkstedet, og vi fik konstrueret et mekanisk træk, som helt fint udmunder i rolige mekaniske svingninger, der roligt vugger barnevognen.

Uro - Joachim & Mads
Denne aktuator har til formål, at drive uroen. Igen skulle motorstyring benyttes, og der blev besluttet, at benytte samme motorstyring, som ved vugge-motoren, selvom denne vugge-motor er af en mindre størrelse. Dette er gjort for enkelthedens skyld. Der opretholdes derfor en generel struktur(design), som gør eventuelle fremtidige tilpasninger nemmere – en indsættelse af en større motor, ville derfor ikke kræve et nyt driver trin her

Temperaturføler - Kristian
Temperaturfølerens opgave er, at informere brugeren om den aktuelle temperatur i barnevognen. Den resistive temperaturmåler NTC thermistor er det komponent i vores opstilling hvis modstandsværdi ændres afhængigt af temperatur. NTC thermistoren eller Negative Temperature Coefficent, thermistoreren er en elektronisk enhed, hvis elektriske modstand varierer invers proportionalt til den omgivne temperatur. Det vil sige, at modstandsværdien for NTC thermistoren falder som temperaturen stiger og omvendt. Den valgte NTC thermistor har en værdi på 100 $k\Omega$. En Wheatstone bridge er meget velegnet til at måle små ændringer i modstandsværdi. Det grundlæggende princip er at have to ballancerede spændingsdelere, og måle den differentielle spændingsforskel mellem de to spændingsdelere. Rent måleteknisk er det attraktivt at måle en differentiel spændingsvariation, fremfor at måle absolutte spændingsniveauer. Vi benytter

en instrumenteringsforstærker til at sammenligne spændingen i de to spændingsdelere. Dette er den analog del af temperaturmåleren. I konverteringen til digitalt signal benyttes PSoC3, og en ADC til konvertering af spændingen. Denne digitale værdi omregnes til en tilsvarende temperatur-værdi ved hjælp af en algoritme som er fremfundet ved regression af flere målinger. Som udgangspunkt er spændingsforskellen på udgangen af instrumenteringsforstærkeren 0 V og dette er ved ca. 25 °C, hvis temperaturen stiger, vil der på udgangen være en negativ spænding. Af hensyn til vores PSoC3's arbejdsområde har vi hævet arbejdspunktet for at kunne se både temperatur stigning og fald. Imellem spændingsforsyningen og Wheatstone bridgen er der placeret et lavpas filter. Dette forsikrer os, at der ikke vil komme noget støj fra motorer ind i denne del af kredsløbet og forsage fejlmålinger.

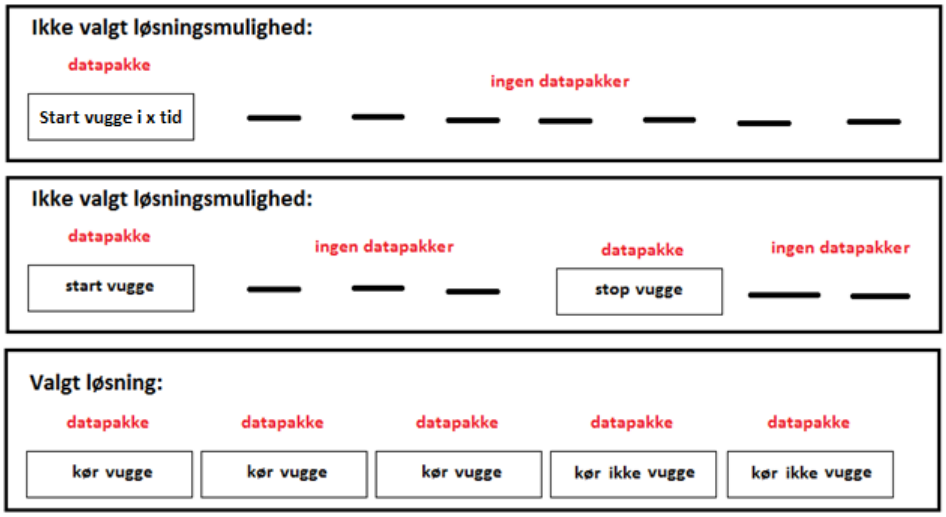
ZigBee transceiver - Kjeld
Dette modul står får den trådløse kommunikation mellem Controller og Pram. Valget faldt på Zigbee transceiver modulerne, da en i gruppen havde kendskab til disse i forvejen. Zigbee modulerne har en god rækkevide på 50-100m, og kan køre på flere forskellige frekvensbånd. Hardware-mæssigt er de nemme at implementere, da de blot benytter en SPI-bus, og nogle få andre kontrolsignaler.

9.4 Analyse SW - (David & Sander), Kjeld, Jonas og Anders

SmartPram systemets software er som bekendt opdelt til at køre delvist på DevKittet og delvist på PSoC3'en. Det var på forhånd givet, at DevKittet via sin touchskærm skal fungere som grænseflade til brugerens interaktion, mens PSoC3'en skal fungere som grænseflade til sensorer og aktuatorer. Men herimellem opstod det designmæssige valg, om på hvilke enheder de forskellige softwareklasser skal køres. Vi valgte at tage det designvalg, at DevKittet fungerer som den intelligente enhed, hvor hovedparten af logikken køres, mens PSoC3'en mere fungerer som en forlængerarm for DevKittet, og at PSoC3'en derfor bare gør, hvad den får besked på uden selv at tage de store be-slutninger.

Et vigtigt element for at få dette til at spille sammen er kommunikationen imellem de to enheder. Da vi ønsker, at DevKittet hele tiden skal kunne holde brugeren opdateret med de nyeste temperatur- og lyd-målinger, sender PSoC3'en blot disse oplysninger i datapakker, der kommer i et fast interval, så længe systemet er aktivt.

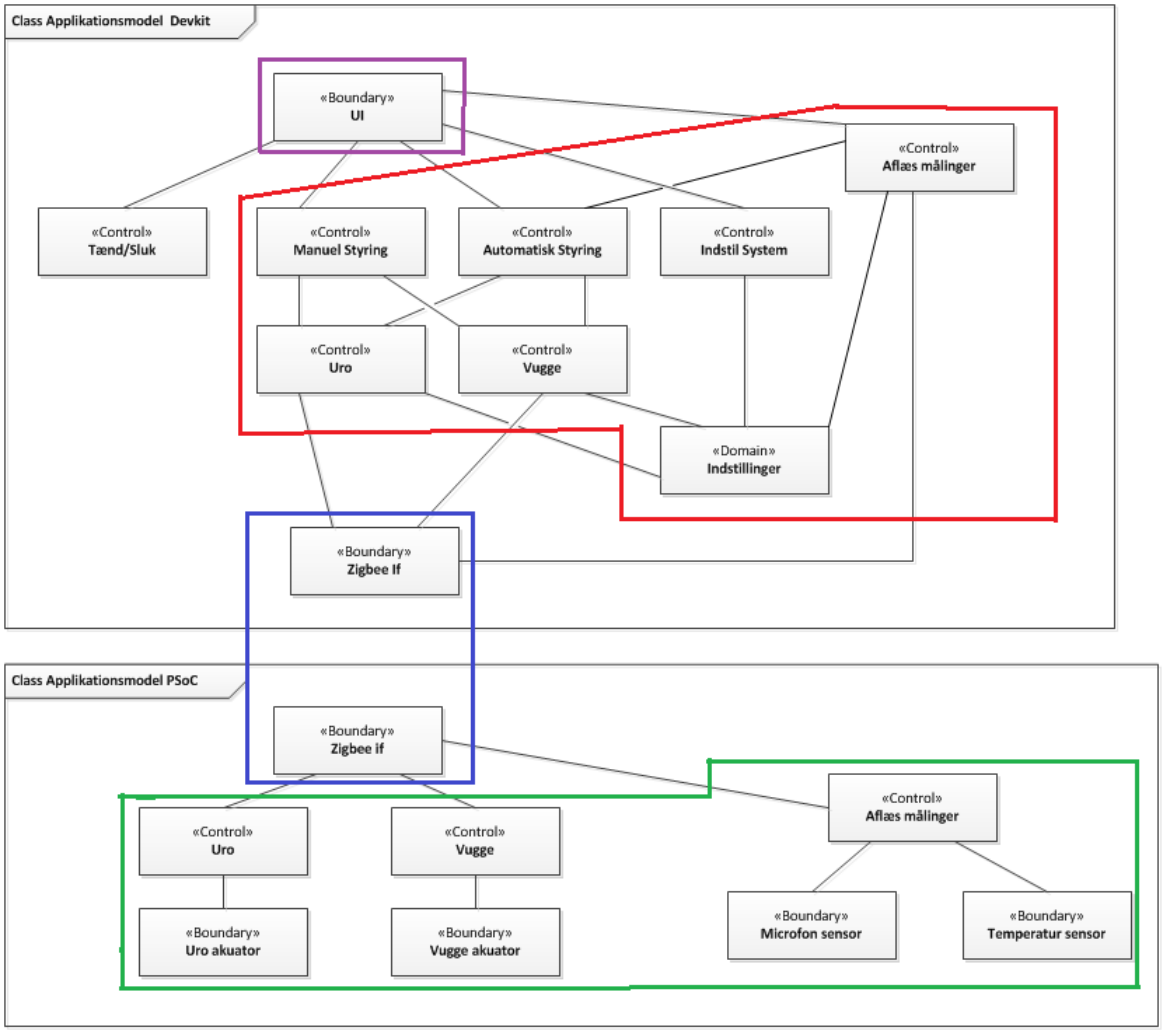
Kommunikationen fra Devkit8000 til PSoC3 står for at styre, om vuggen og uroen skal køre (og med hvilken styrke), eller om de ikke skal. Dette kunne være udført på flere forskellige måder. En løsning kunne være, at man sendte en startbesked til PSoC3'en, når f.eks. vuggen skulle starte sammen med en tidsstørrelse, og at PSoC3 selv ville stoppe vuggen, når denne tid var gået. En anden mulighed, som var den vi valgte, er at DevKittet står for at sende en startbesked, når vuggen skal starte, og at DevKittet står for at holde øje med tiden for så at sende en stopbesked, når tiden er gået. Derved skal PSoC3'en slet ikke kende til tiderne, men skal bare tage imod de kommandoer, som den får tilsendt, hvilket stemmer bedre overens med vores filosofi om, at PSoC3'en blot er en forlængerarm. Vi valgte en løsning, hvor DevKittet konstant sender datapakker til PSoC3'en med et bestemt tidsinterval mellem hver pakke, hvorved en evt. mistet datapakke ikke vil medføre så store konsekvenser. Denne løbende dataudveksling fungerer også som en slags Watch dog, der kan få PSoC3'en til at gå i stand by mode, hvis den ikke længere modtager data. Dette er illustreret på figur 11.



Figur 11: Alternative løsninger og valgte løsning

Dette medfører naturligvis også en større dataudveksling, hvilket ikke er et strømsparende tiltag. Set i bakspejlet ville en løsning med modtagelsesbekræftelse nok have været mere optimal, men vi foretog valget, da vi stadig regnede med at streame lyden fra PSoC3 til Devkit8000, hvorved de ekstra krævede ressource for løbende at sende beskeder fra Devkit8000 til PSoC3 ville have været forholdsvist små.

Softwarearbejdet blev fra start af inddelt i mindre grupper, hvor en opdeling gav god mening. Vi valgte overordnet set at dele grupperne imellem systemets grænseflader, således at en gruppe tog sig af softwaren til PSoC3 (Mads og Jonas), en tog sig af grænsefladen til zigBee på både PSoC3 og Devkit8000 (Kjeld), en gruppe tog sig af softwaren på Devkit8000 (David og Sander), og en to sig af det grafiske QT user interface på Devkit8000 (Anders). På den måde var gruppernes individuelle arbejde ikke direkte afhængigt af hinanden i den indledende fase, da vi i de grupper, der grænsefladem æssigt ligger op mod hinanden havde aftalt, hvordan bindeledende skulle kunne forbindes. Denne inddeling er illustreret på figur 12.



Figur 12: Inddeling af softwarearbejde. Lilla(Anders), Rød(Sander og David), Blå(Kjeld) og Grøn(Mads, Kristian, Joachim og Jonas).

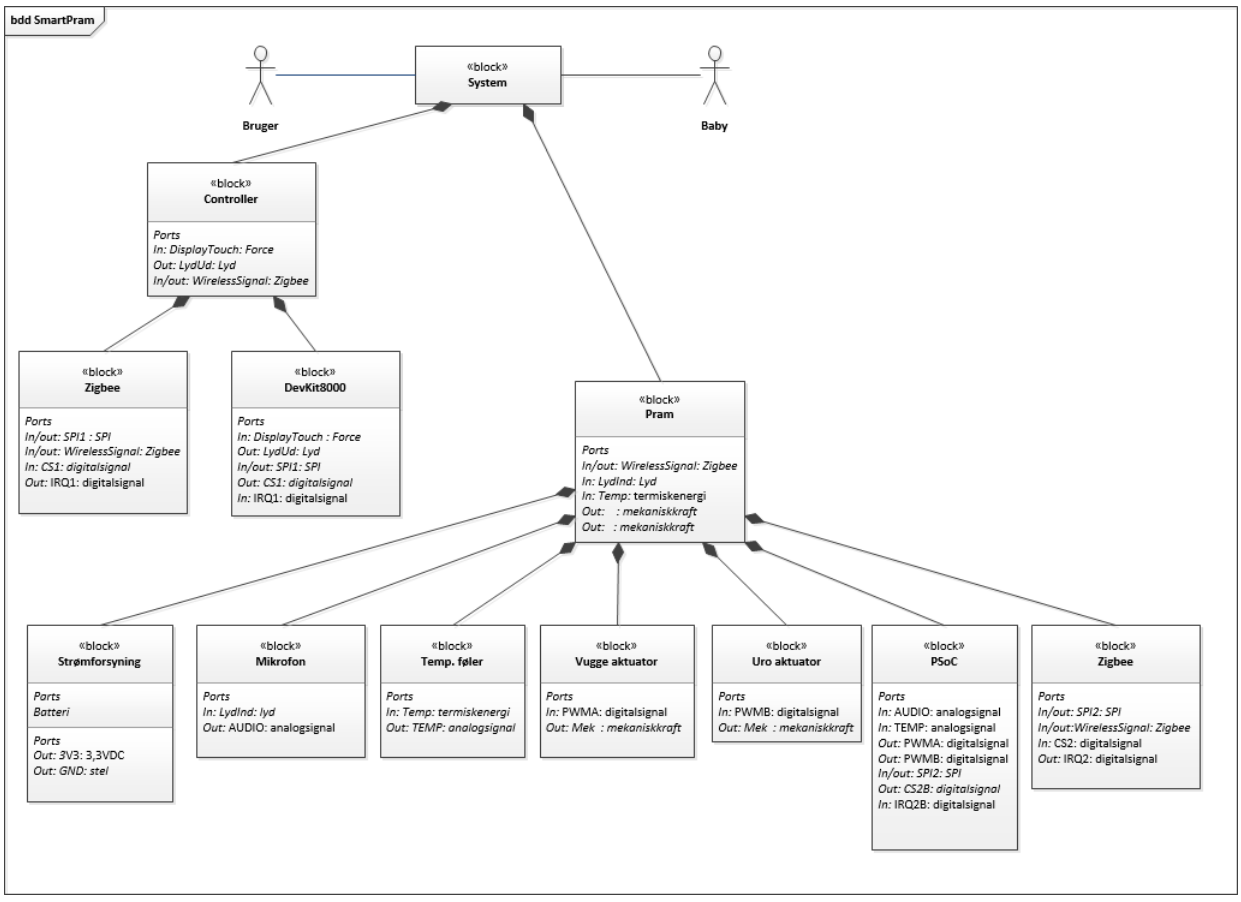
I figur 12 ses inddeling af softwarearbejdet ud fra en gennemarbejdet applikationsmodel. Vi har tilføjet en UIController til UI-grænsefladen, hvilket dog ikke ses direkte på tegningen, og denne fungerer som en bro fra QT-grænsefladen til messageQueue-systemet på DevKit8000. UIControlleren er også blevet brugt til debug.

9.5 Systemarkitektur (Joachim & Mads)

Her bliver systemets sammensætning beskrevet. Systemarkitekturen beskriver hvordan de enkelte dele er koblet sammen og kommunikerer med hinanden. Den giver et godt overblik til projektarbejdet og over, hvordan systemet virker. Systemarkitektturen er overordnet beskrevet med et bdd, som ses på figur 13. De enkelte blokke vil efterfølgende blive beskrevet nærmere.

9.5.1 Block Definitions diagram

Bdd som viser den logiske sammenhæng mellem de forskellige blokke i systemet. Bruger kan tilgå System, som består af Controller og Pram. Bruger kan via Controlleren tilgå resten af systemet. Controlleren består af et Devkit8000 og et Zigbee-modul. Igennem Zigbee-modulet opnås der trådløs forbindelse til PSoC3'en. Denne enhed behandler data, der kommer fra Devkit8000, og styrer aktuatorene her ud fra. Devkit8000 modtager også information fra PSoC3, når data er klar på PSoC3, herunder lyd fra mikrofon og temperatur fra temperaturføler.



Figur 13: Block definitions diagram.

Controller
Controllerdelen er den blok, der indeholder DevKit8000 og en ZigBee transceiver, hvilket ses på IBD'et på figur 14. Denne blok står for brugergrænsefladen til Bruger. Det er her, at Bruger kan kommunikere med Pram-blokken. De enkelte underblokke er beskrevet nedenfor.

ZigBee
Er en HW-enhed, som kan sende information trådløst til en anden Zigbee. Den bruges til at sende information imellem PSoC3 og Devkit800.

Devkit8000
DevKit8000 består af en microcontroller med tilhørende periferihardware til elektrisk interface til Zigbee. DevKit8000 afvikler en række SW-komponenter og Bruger tilgår dette via en touchskærm, som er en del af enheden.

Pram
Pram er den blok, der består af PSoC3'en og en Zigbee transceiver. Denne blok sørger for diverse målinger på barnevognen og igangsættelse af diverse aktuatorer.

Strømforsyning
Komponent, der vil være i stand til, at supplere alle HW-enheder på Pram med en passendeforsyningsspænding.

Mikrofon
Er en HW-enhed, der opfanger lyd i barnevognen.

Temperaturføler
Er en HW-enhed, der måler temperaturen i barnevogne.

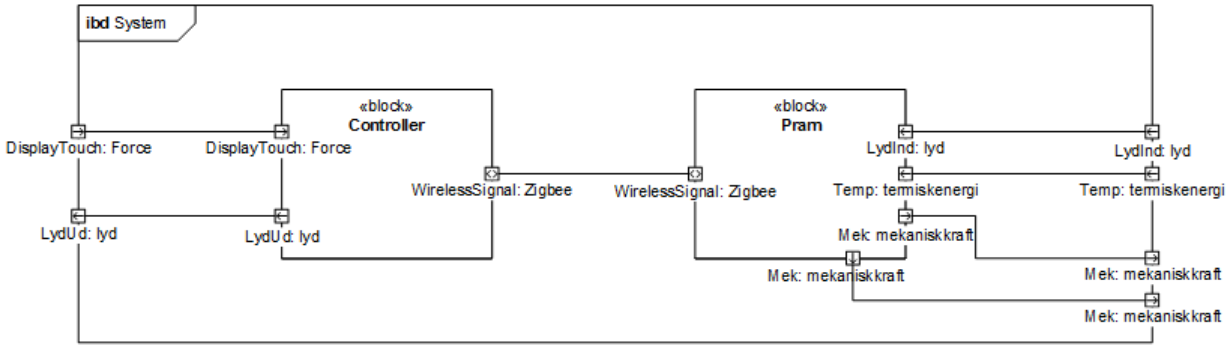
Vuggeaktuator
Er en HW-enhed, som kan vugge barnevognen vha. motorkraft med henblik på at få Baby til at slappe af.

Uroaktuator
Er en HW-enhed, som kan dreje rundt vha. motorkraft med henblik på at få Baby til at føle sig tryg.

PSoC3
PSoC3'en består af en microcontroller med tilhørende periferihardware til elektrisk interface til uro, vugge, mikrofon, temp. føler og Zigbee. PSoC3'en afvikler en række SW-komponenter som beskrevet i PSoC3'ens applikationsmodel i afsnit 2.8 i dokumentationen.

9.5.2 Internt Blok diagram

Ibd for det overordnede system ses i figur 14.



Figur 14: Internt Block Diagram.

Med dette Ibd fås et overblik over den generelle anvendelse af systemet, og dennes forbindelse til omverdenen.

9.5.3 Datapakker

PSoC3→Devkit8000

1byte Temperatur	1byte Lydstyrke

Figur 15: Software datapakke fra PSoC3 til Devkit8000.

Vi sender to værdier over til Devkit8000. Den første værdi er den aktuelle temperatur, der er i barnevognen, og den anden værdi er den lydstyrke, der er blevet analyseret i barnevognen af PSoC3'en, som bliver sendt over til Devkittet.

Devkit8000→PSoC3

1byte Vuggehastighed	1byte Urohastighed

Figur 16: Software datapakke fra Devkit8000 til PSoC3.

Vuggehastighed beskriver, om vuggen skal startes eller stoppes og med hvilken hastighed. Urohastighed beskriver, om vuggen skal startes eller stoppes og med hvilken hastighed. Vuggehastighed og Urohastighedens besked fungerer som "watchdogs".

9.6 SW Design

9.6.1 Devkit8000 - David & Sander

Designfasen af softwaren til DevKittet blev initieret ved at tage udgangspunkt i applikationsmodellen fra systemarkitekturen i afsnit 9.5. Her blev forefundet en tydelig klasseopdeling i kontrol-, domæne- og grænsefladeklasser, som var allokereret til DevKittet. Et logisk træk var her at bibeholde denne klasseopdeling i selve softwaredesignet ved at oprette tilsvarende klasser i et objektorienteret domæne. Derved fås et logisk og overskueligt udgangspunkt at programmere ud fra.

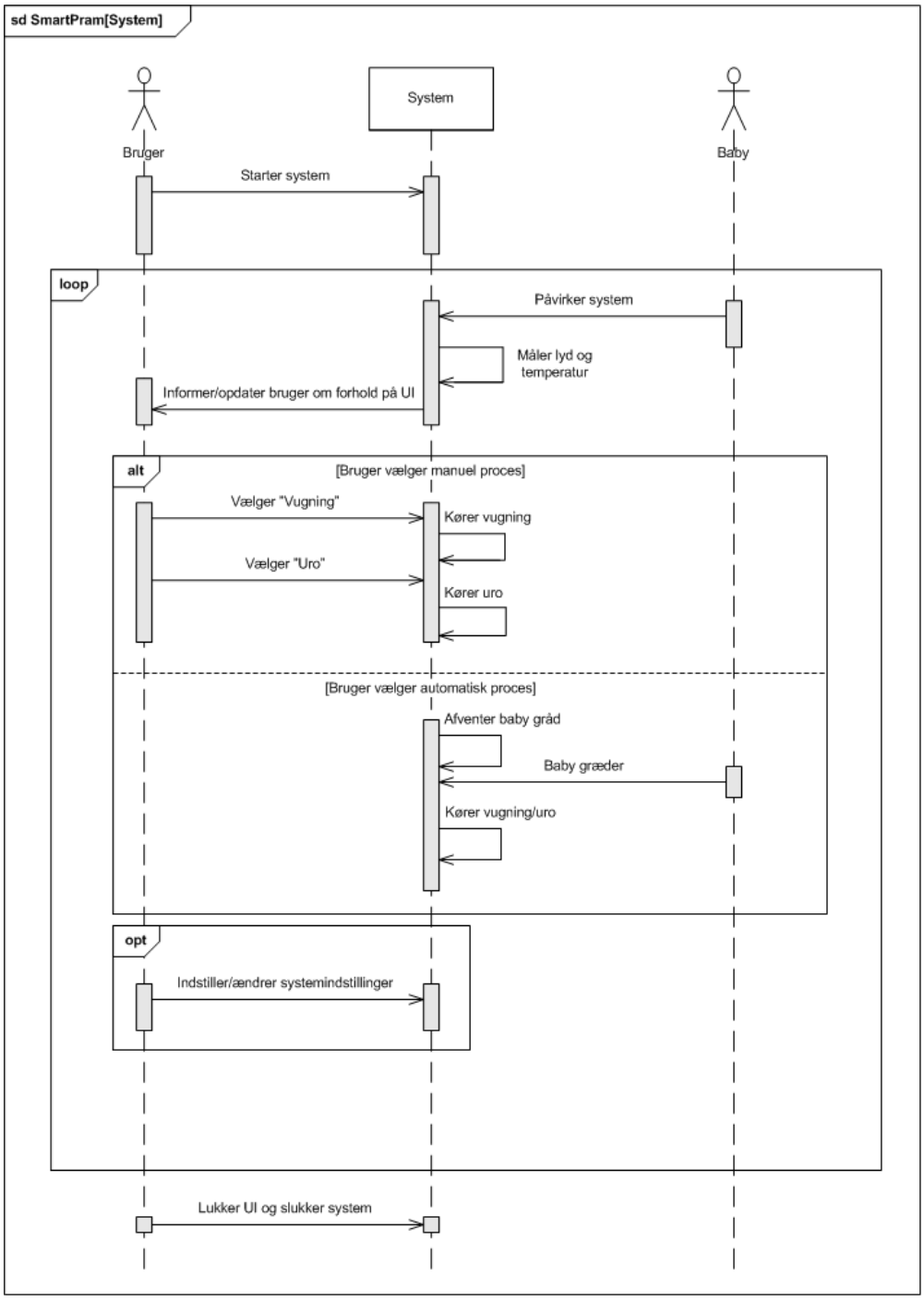
Ser man på kravene fra kravspecifikationen i afsnit 7, fremgår det, at flere opgaver skal kunne udføres på samme tid, såsom at vuggen og uroen skal kunne styres på samme tid med, at temperatur- og lydinformationer indsamles og behandles, alt imens user interfacet skal kunne tage imod input fra brugeren. Dette lægger meget op til et flertrådet program, hvilket også er den tilgang, vi har valgt at tage.

Af figur 17 ses det overordnede flow af beskeder, der sker i systemet. Her ses de mange veje, systemet kan gå, mens det modtager inputs fra Bruger samtidigt med, at der hele tiden skal holde øjes med, hvornår baby græder, da dette påvirker systemet yderligere. Sekvensdiagrammet i figur 17 er meget overordnet, men der refereres til projektdokumentationen afsnit 4.1 for yderligere detaljer vedrørende flowet af systemet.

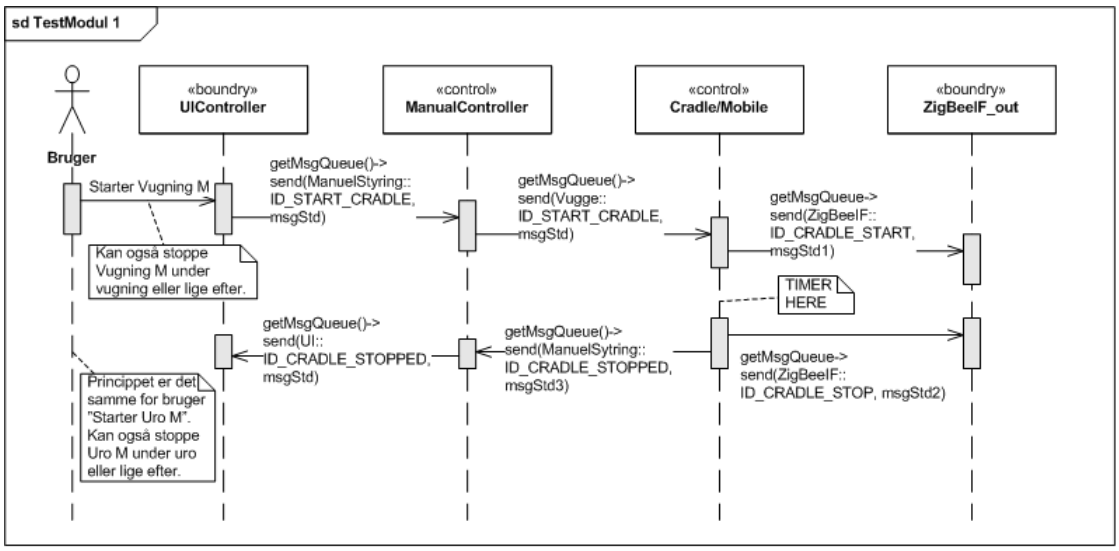
Da systemet skal foretage mange ting på én gang og parallelt, tog vi det valg, at hver klasse skal have sin egen tråd, og den næste designmæssige beslutning lå i at bestemme et system til kommunikationen mellem disse tråde.

Valget faldt på at benytte et eventbaseret MessageQueue-system, da dette ville spare os for en del timingproblemer, som der ofte kan være ved direkte brug af mutex-låsninger. Et beskedsystem ligger også i god tråd med, at DevKittet benytter et user interface, hvor et knaptryk skaber en event, der sætter gang i en ønsket tråd. På den måde vil klassernes objekter blot stå og vente på at modtage beskeder, så længe der ikke sker noget. Ligeledes kan en indkommen besked fra ZigBee-grænsefladen også sætte gang i systemet.

Vi benytter et objektorienteret MessageQueue-system, hvor hver tråd også består af et objekt med metoder og attributter, hvilket gør det muligt at knytte det flertrådede system sammen med de forskellige klasser fra applikationsmodellen, samtidig med at det bliver mere struktureret.



Figur 17: Sekvensdiagram for SmartPram system



Figur 18: Eksempel på brug af MessageQueue-systemet - her vist testmodul 1.

Et eksempel på vores brug af det objektorienterede MessageQueue-system ses i figur 18, som er et udsnit af testmodul 1 - Forklaring heraf kan ses i software implementeringen for Devkit8000 i afsnit 9.8.1.

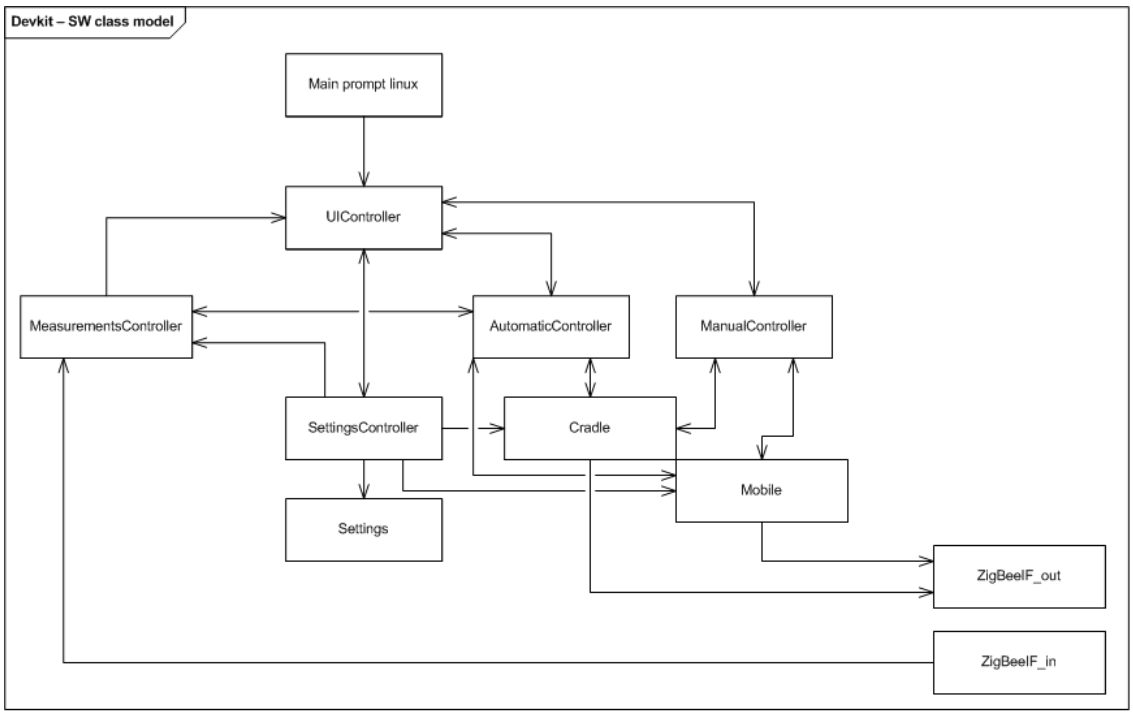
MessageQueue-systemet baserer sig på et OSAPI, vi har valgt at tage udgangspunkt i. Dette OSAPi er på forhånd givet og implementeret i kurset Indledende System Engineering med Søren Hansen som underviser. Grunden til at vi har valgt at tage udgangspunkt i dette er, at OSAPI tilbyder mange faciliteter såsom mutex-låsning og automatisk oprettelse af tråde, hvilket netop er, hvad vi har brug for i projektets softwareafdeling. Yderligere tilbyder den også en timerfunktion (mere herom i implementeringsdelen i afsnit 9.8.1), som kan benyttes i forbindelse med vugge/urotiden samt sendetiden fra ZigBee_out-klassen.

Vi bruger kun OSAPI af den grund, at vi kan bruge dens funktionalitet til at oprette tråde og gøre dem sikre samt benytte en timerfunktion, da en sleep() funktion ikke er god at bruge timingsmæssigt på target.

Vi vurderer, at et message distributionsystem, hvor en beskedcentral sørger for videreleveringen af beskederne, ville have været en endnu bedre løsning til opgaven, da vi med vores løsning er nødt til at opsætte en hel del referencer imellem objekterne. Vi lærte dog først til dette koncept på et sent tidspunkt i forløbet, og vi valgte derfor at holde os til det, vi allerede var startet på.

Vi har lagt meget vægt på, at der skal være en tydelig forskel på kontrol-, domæne- og grænsefladeklasser, og at disse arbejder på vidt forskellige måder. Det er derfor kontrolklasserne, der står for al udregning og logik, mens grænsefladeklasserne så vidt muligt blot modtager eller videresender disse informationer. Selvom dette medfører, at der skal sendes en del flere beskeder rundt imellem klasserne, og at disse beskeder indeholder flere oplysninger, så giver det et langt mere logisk design at arbejde ud fra.

Selve designafsnittet i projektdokumentationen afsnit 4.1 omfatter udover klassebeskrivelser en række sekvensdiagrammer for denne beskedkommunikationen imellem klasserne, og vi vurderer, at disse er den bedste måde at beskrive softwaren på, og at det giver et mere overskueligt udgangspunkt for selve implementeringen.



Figur 19: Klassediagram model med associationer.

I figur 19 ses et overordnet klassediagram med associationer mellem klasserne.
Se projektdokumentation afsnit 4.1.1 for et detaljeret klassediagram med OSAPi'ets beskeder og trådoprettelser som yderligere association til klasserne.

9.6.2 PSoC3 - Jonas

The PSoC's role is to control the pram itself. It will gather data through sensors, send and receive data wirelessly, as well as affect the physical world through actuators. The software for these tasks is written in PSoC Creator 3, in the programming language C.

The main requirements for the PSoC's functionality were as follows:

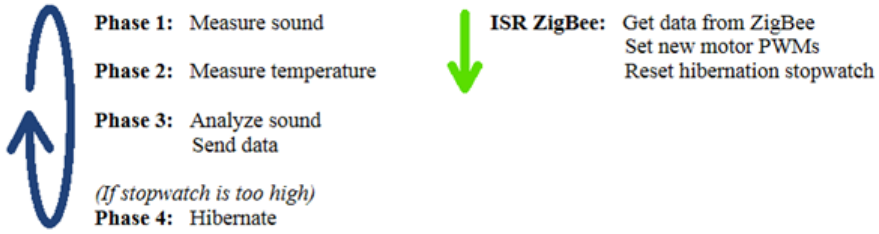
- Measure and analyze sound level.
- Measure and analyze temperature.
- Send data (sound level & temperature) to the DevKit.
- Receive data (cradle- & mobile PWM) from the DevKit.

The ZigBee driver was coded to trigger an interrupt on the PSoC when a new transmission has arrived from DevKit. The program should then read the two PWM values and update the motor PWMs accordingly. It was considered beneficial to execute all this in the ISR ZigBee, thus being able to keep the rest of the program running as an eternal loop.

The PSoC only receives information about the motor PWMs from the DevKit, no start or stop signal. For that reason, it was desirable to make the system hibernate if connection to

the DevKit was lost. A stopwatch was added to the design, reset only in the ZigBee ISR, incremented in every round of the main program loop, and if too high, sends the system into hibernation.

The following sketch shows the basic architecture of the PSoC software.



Figur 20: Basic architecture of the PSoC software.

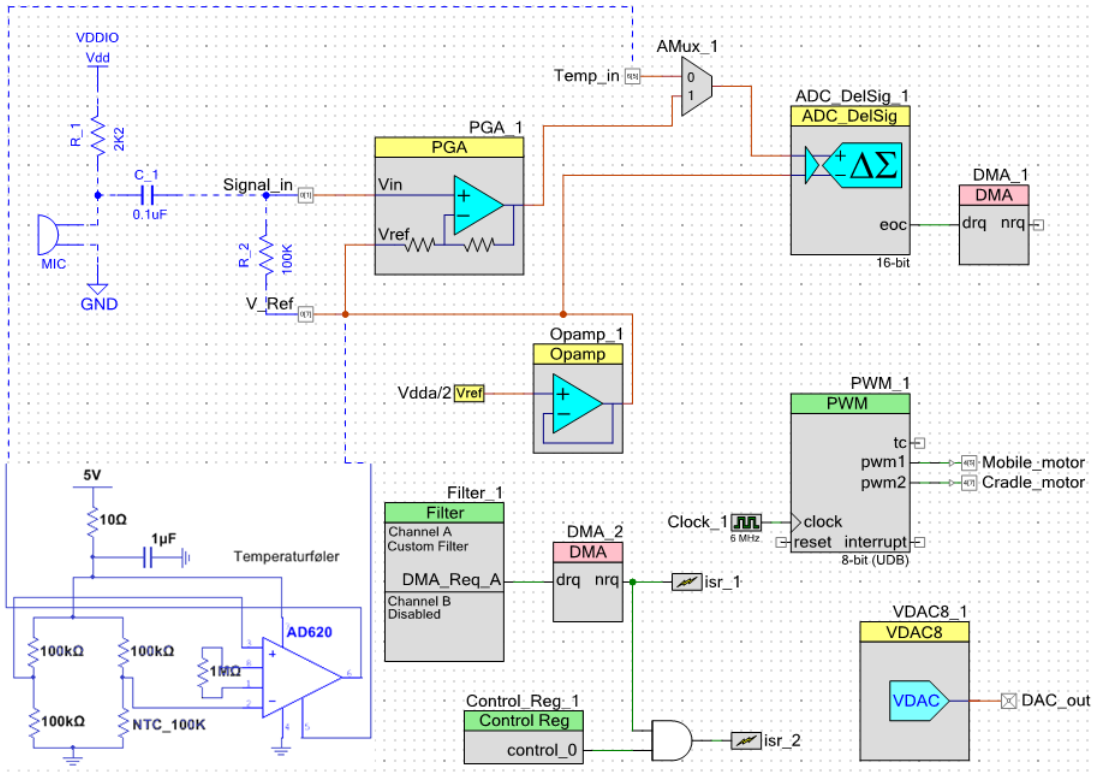
Having set the basic structure of the PSoC software, there were a few points that needed planning before starting the implementation:

- The PSoC 3 only has one ADC-converter, meaning the temperature signal and the sound signal must share the ADC input. The Analog Mux component on the PSoC was chosen to solve this problem.

9.7 HW implementering

9.7.1 Pram - Mads, Joachim og Kristian

Nedenfor beskrives de enkelte blokke i Pram-delen, med henblik på implementeringsprocessen.



Figur 21: PSoC3 top design

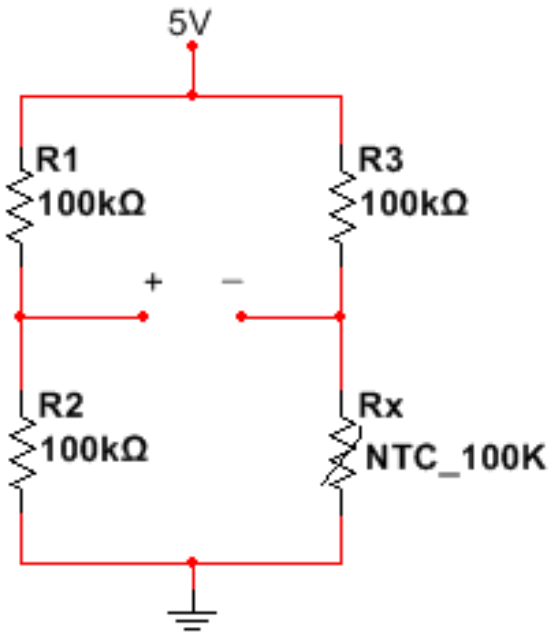
Mikrofon - Joachim & Mads
Dette kredsløb funktion er at fange lydsignaler fra babyen, der ligge i vores barnevogn. Denne er er koblet til ground(GND) og en inputspænding på 3,3V. Input spændingen løber først igennem en 2,2 $k\Omega$ modstand. Mikrofonens output føres igennem en større elektrolyt kondensator, således, at DC-spændingen fra forsyningen fjernes, da denne ikke bør være en del af output'et. Signalet føres herefter ind på PSoC3'en. Dette signal, er dog løftet med den halve forsyningsspænding – 1,65V – da PSoC3'ens arbejdsområde netop ligger mellem 0-3,3V. Hermed vil mikrofonens output svinge omkring 1,65V. Dette kan så ses, som mikrofonens nye reference. I PSoC3 bliver signalet forstærket væsentligt. Forstærkningen er således, at det højest mulige output fra mikrofon, ikke kommer over 3,3V, og hermed heller ikke kommer under 0V. Den højest mulige forstærkning, som OpAmps i PSoC3 kan sættes til, er 50(for ikke-inverterende). Dette er også den valgte forstærkning her. Signalet fra mikrofonen er nemlig meget lille. Herefter føres signalet ind på en ADC. Dennes reference er også sat til de 1,65V. For at undgå konvertering af aliasering, som er de frekvenser, der ligger over den halve samplingsfrekvens, bør der som udgangspunkt benyttes et skarpt analogt filter foran ADC'en. Dette er dog ikke nødvendigt i vores tilfælde, da ADC'en vi benytter, er af typen Delta Sigma. Denne har sin egen form for aliaseringsfilter, som bunder i dens hurtige oversampling. Det er derfor denne oversamplingsfrekvens der danner grænsen. Og da vores signal på ingen måde vil nærme sig dennes halve samplingsfrekvens, har vi valgt at lade ADC'en stå udelukkende for denne anti-aliasering.

Vuggemotor - Joachim & Mads
Aktuatoren er blevet dimensioneret efter den batterispænding, som der er til rådighed på barnevognen fra vores 12V scooter-batteri. Overgangen fra PSoC3'en til aktuatoren går igennem en MOSFET som vha. et PWM signal fra PSoC3'en, på ca. 23kHz, kan styre motorens omdrejningshastighed og hermed vuggestyrken af barnevognen. For at beskytte kredsløbet er aktuatoren forsynet med en diode til at tilbagekoble den strøm der bliver genereret når aktuatorens spænding frakobles. Vi har valgt at aktuatoren kun skal kunne styres med en PWM på 100% da den ellers havde lidt svært ved at vugge barnevognen ordentligt. Vugningsfunktionen bliver styret af PSoC3'en som sætter hastigheden. PSoC3'en får sine start/stop signaler af ZigBee'ens interrupts som får sin data fra DevKittet.

Uromotor - Joachim & Mads
Den har ligeså en diode der beskytter mod tilbagekoblingsstrømmen fra aktuatoren. Denne aktuatorer er forsynet fra en spændingsregulator der forsyner den med 5V. Overgangen fra PSoC3'en til aktuatoren foregår på samme måde som vuggemotoren men er forsynet fra en anden PWM pin. Da aktuatoren er gearet kører den på 100% PWM da vi har vurderet det er en passende omdrejningshastighed. Uro motoren får sine kommandoer på samme måde som Vuggen. Dvs. PSoC3'en styrer Uroen og PSoC3'ens funktionskald sættes af DevKittet via ZigBee.

Temperaturføleren - Kristian
Denne blok består hovedsageligt at hardware, hvor en Wheatstone bridge benyttes til differentiel sammenligning. Et signal på 5V sendes fra vores batteri hen over et lavpasfilter og en Wheatstone bridge. En NTC thermistor på $100k\Omega$ fungerer som en af de fire resistorer i Wheatstone bridgen. Dette er en resistiv modstand som ændrer sin modstandsværdi afhængigt af temperaturen. Den angivne værdi på vores NTC thermistor informerer om dennes ohmske værdi ved stuetemperatur (25 °C).

Ud fra denne information indsatte vi de tre $100k\Omega$ som nu har designet Wheatstone bridgen. De to knudepunkters spænding vil nu blive sendt ind i en instrumenteringsforstærker AD620, som har til opgave at sammenligne de to spændinger og sende spændingsforskellen ind på en ADC på PSoC3.



Figur 22: Wheatstone bridge

Ved stigning i temperatur vil den ohmske værdi falde og dette vil resultere i et spændingsfald på thermistor-siden. Den nu differentielle forskel vil resultere i en spændingsforskel mellem de to knudepunkter og denne spændingsforskel vil via AD620 blive sendt ind i vores ADC. Vi har valgt, at AD620 kun skal udsende den faktiske forskel, derfor er der en forstærkning på 1 – hvilket forklarer den $1M\Omega$ modstand, som sidder imellem ben 1 og ben 8. Hvis der ikke er nogen spændingsforskel på + og - indgangene, vil der blive sendt 0V ind på vores ADC, men hvis temperaturen falder, vil spændingen på den negative indgang stige og dette vil resultere i en negativ spænding.

Vores PSoC har et arbejdsområde på 0-3,3V af denne grund er arbejdsområdet løftet til 1,65V via instrumenteringsforstærkeren. Arbejdsområdet for vores ADC er ±1,024V indenfor dette spænd, jf. kravspecifikationen, skal der kunne måles værdier fra $-10\,^{\circ}\mathrm{C}$ til $+60\,^{\circ}\mathrm{C}$. Den spændingsforskel, som er blevet digitaliseret via ADC har en tilsvarende temperatur, algoritmen for omregningen fra en ADC-værdi til temperatur er lavet ud fra en regression af målepunkter. Vi benyttede et varmeskab, og med et interval på $10\,^{\circ}\mathrm{C}$ aflæste vi den tilsvarende værdi i PSoC fra ADC. Ud fra disse målepunkter vurderede vi, at det var mest passende med en lineær regression, og denne blev så vores algoritme til omregning fra ADC-værdi til temperatur, som bliver sendt til vores DevKit8000.

Foran Wheatstone bridgen er der implementeret en lavpas led til at forsikre der ingen støj er, hvis motoren skulle påvirke denne del af kredsløbet. Dette lavpasled består af en 10Ω modstand og 1uF kondensator, den har en knævfrekvens på

$$f_c = \frac{1}{2*\pi*R*C} = \frac{1}{2*\pi*10\Omega*1*10^{-6}F} = 15,9kHz$$

, hvilket skulle dæmpe al støj som motorerne kunne ske at frembringe. Vores ADC har et arbejdsområde der er fastsat til ±1,024V=2,048V. Denne spænding er uddeles på 16 bit. Vi finder nu opløsningen: 2,048V/2^{16} =31,25uV

Ved sammenhold med den udregnede samlede støjspænding, som fremfindes til at være 9,163uV ses det, at den samlede støjspænding maksimalt kan udgøre ca. 1/3 af hvert step – det kan derfor konkluderes, at støjspændingen ikke er af relevans.

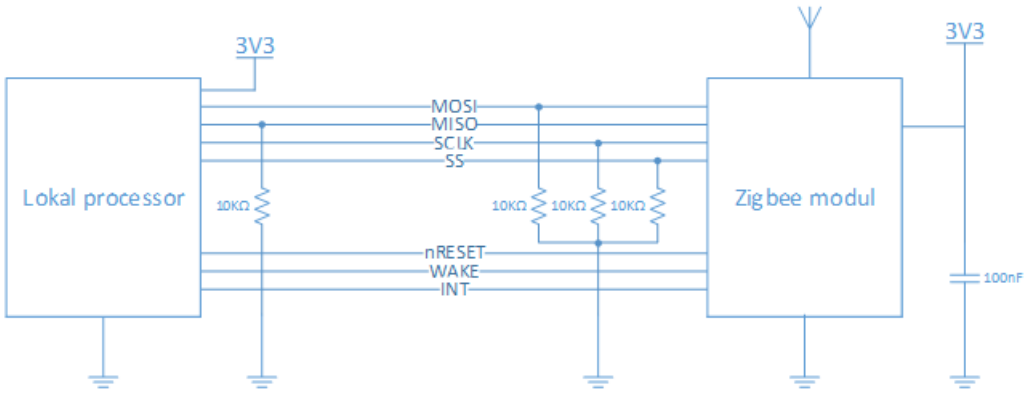
Spændingskilde - Joachim & Mads
Drivkraften på barnevognen kommer fra et scooter-batteri på 12V. Hele kredsløbet er som det første blevet beskyttet med en 1,6A sikring. Herefter fordeles spændingen ud til en spændingsregulator, 12V forsyning til vugge motoren og PSoC3'en får også direkte 12V. Sidstnævnte er blevet verificeret med databladet.

9.7.2 ZigBee - Kjeld

Fra et tidligt stadie i projektet blev det ønsket at der var trådløs kommunikation mellem Controller og Pram, og til at løse denne opgave blev Zigbee moduler foreslået. Zigbee modulerne er komplette enheder som tilgås via en standard SPI port, og skal derudover blot have 3,3 VDC til forsyning. Zigbee modulerne er fra producenten Microchip og har betegnelsen MRF24J40MA, og er baseret på IEEE standarden 802.15.4, som specificere hele det fysiske lag og dataprotokollen. Det eneste brugeren skal gøre for at bruge Zigbee modulet er at konfigurere den og forberede den data man ønsker at sende fra den.

Måden det blev valgt at gribe det an på, var at lave et lille teknologi projekt i sig selv, for at bevise at det kunne fungere, men det havde også den fordel at det kastede en funktionel driver af sig til PSoC3 processoren. Derfra var det blot at pakke den ind i en Linux Device Driver, og så fungerede den også på Devkit8000. I de følgende afsnit vil design og implementeringen af Zigbee modulet blive mere detaljeret beskrevet.

Hardware setup
Selve interfacet mellem Zigbee modulet og den lokale processor er relativt lige til, den består blot af selve SPI bussen hvor Zigbee modulet er slaven, og så tre kontrol signaler. I figur 23 herunder kan man se hvor simpelt det er:



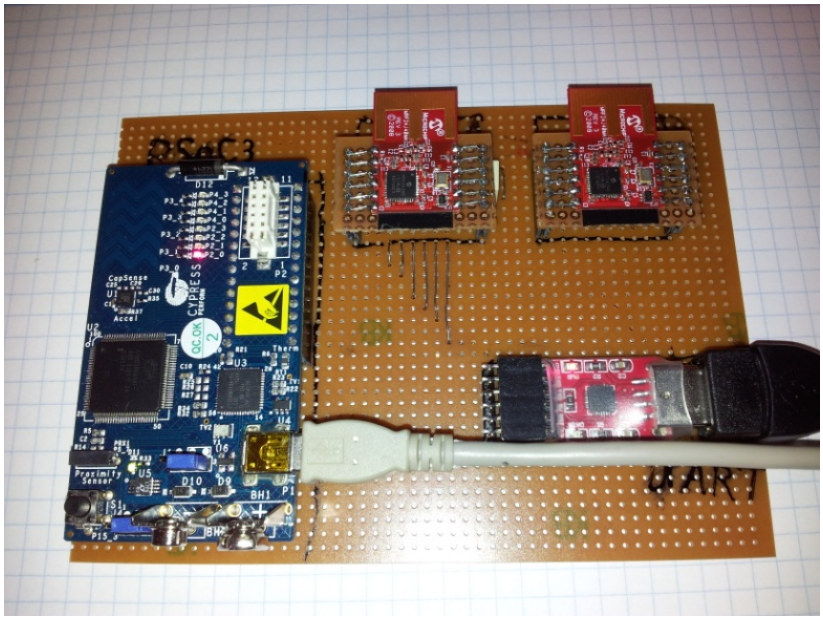
Figur 23: Standard ZigBee modul interface.

De enkelte signaler er beskrevet i dokumentationen under afsnittet "Zigbee hardware".

På figur 23 ses også fire modstande og en kondensator. Modstandende er der blot for at hjælpe på støjimmunitet, og placeres tæt ved indgangs porten hvor signalet skal hen. Kondensatoren er blot god designskik, og er lokal afkobling på forsyningen til Zigbee modulet, og placeres så tæt på Zigbee modulet som muligt.

Driver design
En ting er hardware, men Zigbee modulet fungerer ikke uden noget software til at styre den, hvilket vil sige en software driver, og med to forskellige platforme, PSoC3 og Devkit8000, så bliver det selvfølgelig til to drivere. Idet PSoC3 processoren umiddelbart var den nemmeste at gå til, var det oplagt at begynde på den, hvor de første forsøg på at få Zigbee modulet under kontrol blev forsøgt.

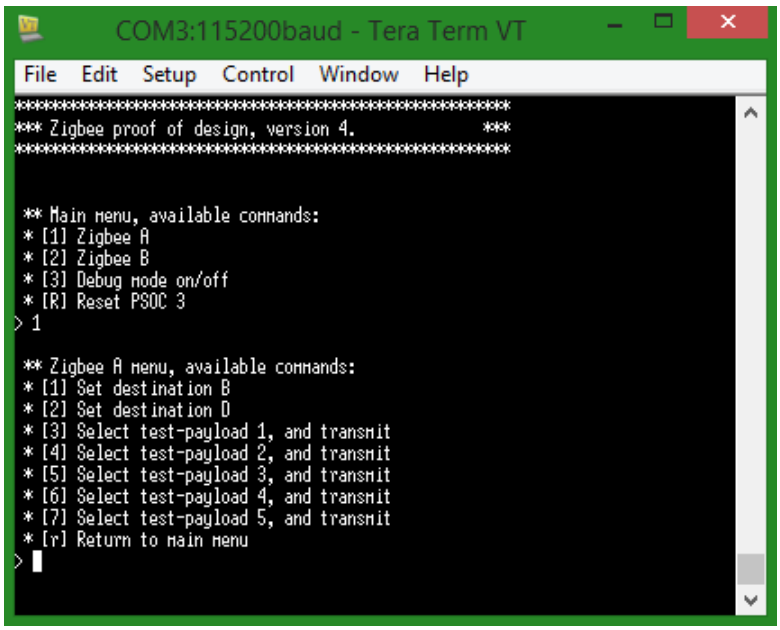
Til forsøgs formål blev et specielt print lavet til PSoC3 modulet som kunne have to Zigbee moduler tilkoblet, og derudover også kunne have et UART til USB modul tilkoblet. UART til USB modulet gjorde det muligt at få beskeder ud fra PSoC3 modulet mens et testprogram kørte, og aflæse dem i et terminalprogram. Denne lille detalje var guld værd, idet det også gjorde det muligt at sende kommandoer den anden vej, hvilket alt sammen hjalp til med at finde alle de små finurligheder og andre forhindringer som blokerede fremdrift i driver designet. I figur 24 er der et billede af det specielle print brugt under start fasen.



Figur 24: Billede af det specielle testprint.

Som man kan se på billedet, er der to Zigbee moduler, og det er der en rigtig god grund til. Det at skrive en driver til en enhed man kun lige er ved at lære at kende er svært nok, men at skulle skrive to, en til hver platform samtidig, er ganske enkelt bare svært. Men hvis man nu kunne nøjes med at skrive en driver til den ene platform, og verificere at den virker først, så bliver det betydeligt nemmere at designe driveren til den anden platform bagefter, for nu har man noget at gå ud fra. Dette var hele ideen bag dette print, og for at Zigbee modulet har en modpart at sende til, ja så sætter man da bare et Zigbee modul mere på.

På baggrund af disse tanker kom dette specielle print til eksistens, og det var i den grad en stor hjælp, også til sidst i projektet hvor den kunne bruges til at sniffe med på kommunikationen mellem de rigtige enheder. I figuren herunder kan man se et screen dump fra terminal programmet Tera Term, hvor man kan se noget af menu strukturen i testprogrammet og nogle af kommandoerne man kan sende tilbage til den.

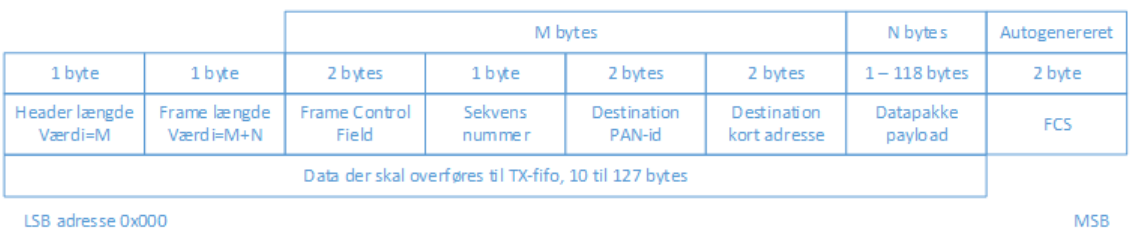


Figur 25: Screen dump af terminal programmet i brug.

Chippen MRF24J40, er hjertet i Zigbee modulet. For at man kan bruge den succesfuldt kræver det forståelse for dens virkemåde, og det får man kun ved at læse og bruge dens vejledninger og anbefalinger, og selvfølgelig at lave små forsøg med den. Der skal ikke lægges skjul på at eureka øjeblikket var knap to uger om at komme, før der succesfuldt kunne sendes en datapakke fra det ene Zigbee modul til det andet. Men der var uden nogen form for adressering, og hvis man var uheldig med sit valg af test data til datapakken man forsøgte at sende, så kom datapakken aldrig frem på modtagersiden. Denne ustabilitet bundede i den manglende forståelse af hvordan en datapakke skal formateres.

Datapakken
Når man ønsker at sende en datapakke fra et Zigbee modul til et andet, så skal datapakken være formateret korrekt, ellers bliver datapakken erklæret ugyldig på modtagersiden og smidt ud. I IEEE standarden 802.15.4-2006 kapitel 7.2 er det specificeret hvordan en datapakke skal formateres korrekt. Når man kigger i det, står det klart at der er flere forskellige muligheder, blandt andet om man ønsker at afsenderens adresse skal sendes med, om der bruges en kort eller en lang adresse, og om der skal bruges nogen former for kryptering. Til vores formål blev det valgt at bruge kort adresse, ingen kryptering, og at der ingen grund var til at afsender adressen blev sendt med.

Først skal datapakken konstrueres for afsendersiden og kopieres ned i det lokale Zigbee moduls TX-fifo. I figur 26 ses det, hvordan datapakken som skal sendes ned i Zigbee modulets TX-fifo ser ud.



Figur 26: TX-fifo datastruktur.

Selve brugerens data er den del af datapakken som kaldes "Datapakke payload", og som man ser er der ret mange ekstra ting der skal med, bare for at kunne sende en enkel byte brugerdata. De enkelte datafelter er beskrevet i dokumentationen under sektionen "Zigbee datapakke".

Med datapakken på plads i TX-fifo'en kan man nu give Zigbee modulet kommandoen der aktiver den trådløse transmission.

På modtagersidens Zigbee modul bliver den modtagende datapakke fra det trådløse netværk lagt i RX-fifo'en, hvis altså destinations PAN-id, korte adresse, og at dens egen generering af FCS feltet ellers matcher. Man kan se at den modtagne datapakke næsten er det samme som det sendte, med få ændringer, og i figur 27 kan man se hvordan datapakke formateringen ser ud når det ankommer til RX-fifo'en.

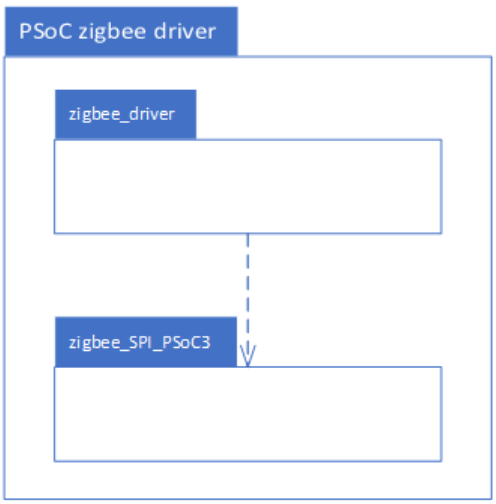
	M bytes				N bytes	Autogenereret		
1 byte	2 bytes	1 byte	2 bytes	2 bytes	1 – 118 bytes	2 byte	1 byte	0 - 1 byte
Frame længde Værdi=M+N+2	Frame Control Field	Sekvens nummer	Destination PAN-id	Destination kort adresse	Datapakke payload	FCS	LQI	RSSI
Data der skal overføres fra RX-fifo, 12 til 130 bytes								

LSB adresse 0x300 MSB

Figur 27: RX-fifo datastruktur.

Som det ses er der et par ændringer i den modtagne datapakke kontra den afsendte datapakke. De enkle datafelter og hvad der er ændret, er beskrevet i dokumentationen under sektionen "Zigbee datapakke".

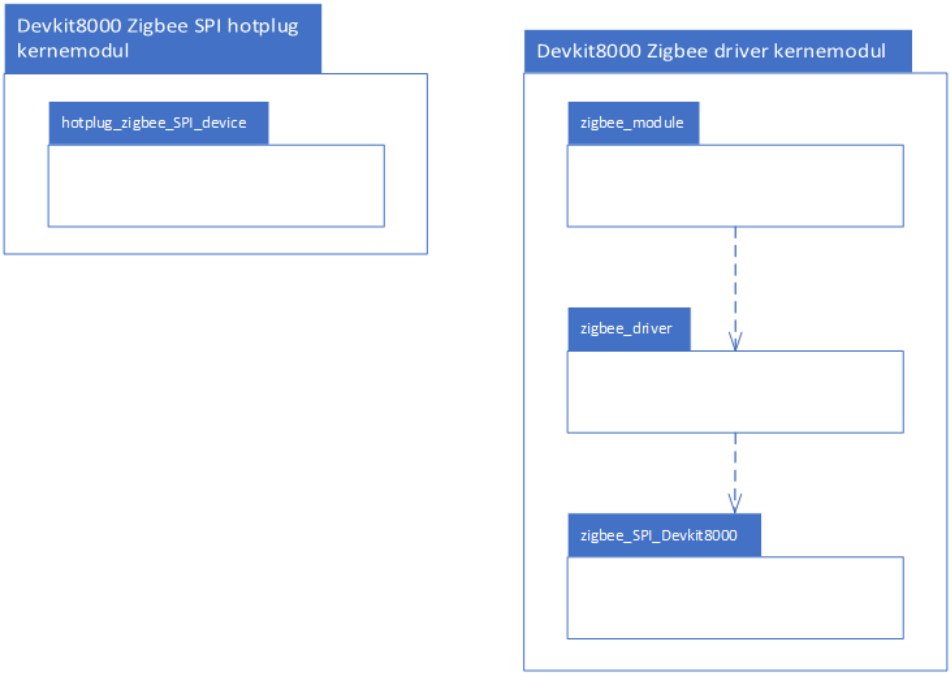
Zigbee modul driverne
For at gøre driveren til selve Zigbee driveren så generel og uafhængig af platform som muligt, så er alle hardwarenære funktioner, inklusiv brugen af SPI bussen, taget ud af driveren og lagt i et underliggende lag. Dette resultere i et lag opdelt design af driveren, med et abstraktions niveau hvor alle hardware relevante funktioner er i det nederste lag, og så bygges selve driveren af Zigbee modulet oven på. I figur 28 kan man se driverpakken til PSoC3 processoren.



Figur 28: ZigBee driver pakke til PSoC3.

Med driveren på plads til PSoC3 processoren kunne den testes på det specielle testprint, hvor vi

succesfuldt kunne sende testdata fra det ene Zigbee modul til det andet, og nu skulle driveren blot porteres til Devkit8000. Devkit8000 er baseret på en OMAP processor, men hvad er mere vigtigt er at den kører styresystemet Linux, hvilket vil sige at driveren til Zigbee modulet skal skrives som en Linux Device Driver. Principielt var det blot et spørgsmål om at konfigurer dens hardware og så ellers genbruge selve Zigbee driveren fra PSoC3 pakken, blot hvor den bruger Devkit8000's funktioner i stedet. En figur af driverpakkerne til Devkit8000 kan ses i figur 29.



Figur 29: ZigBee driver pakke til Devkit8000.

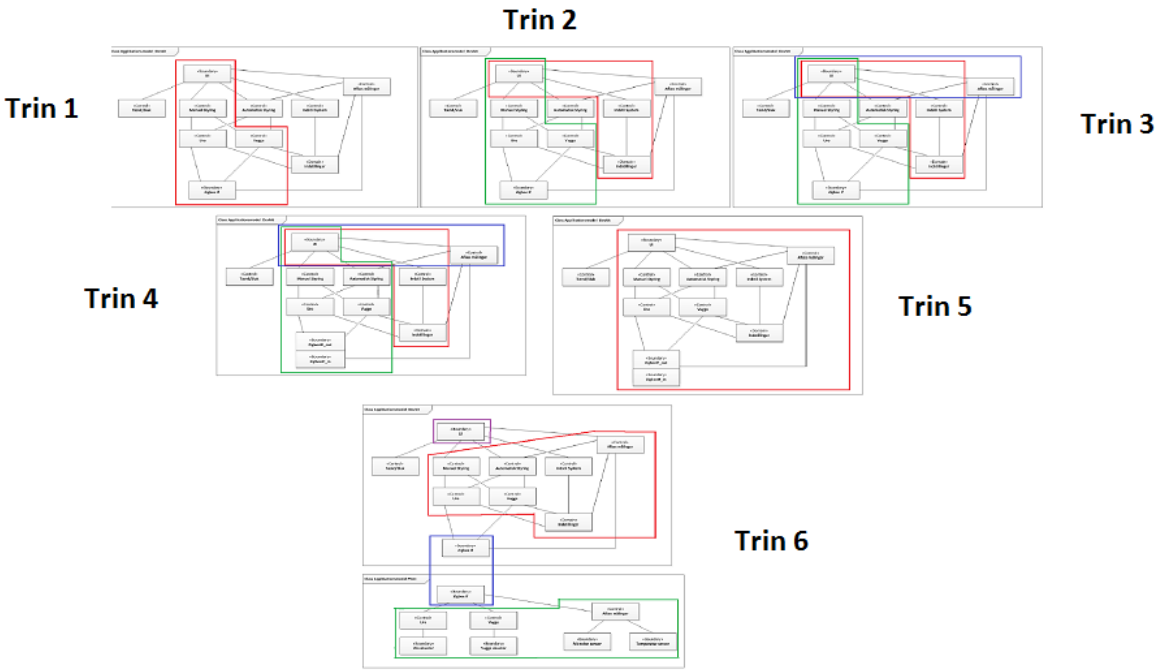
Som det ses i figur 29 har den også en del i pakken der hedder "zigbee_driver", og det er præcist den samme som fra pakken til PSoC3 processoren, den bruger blot bare et andet underliggende lag til at styre hardwaren. Pakken har dog også et lag oven på selve Zigbee driveren, "zigbee_module", og dette er selve Linux Device Driveren som den pakkes ind i, og det er herfra der oprettes fil noder i Linux styresystemet som er tilgængelig for resten af systemet.

9.8 SW implementering

9.8.1 Devkit8000 - David & Sander

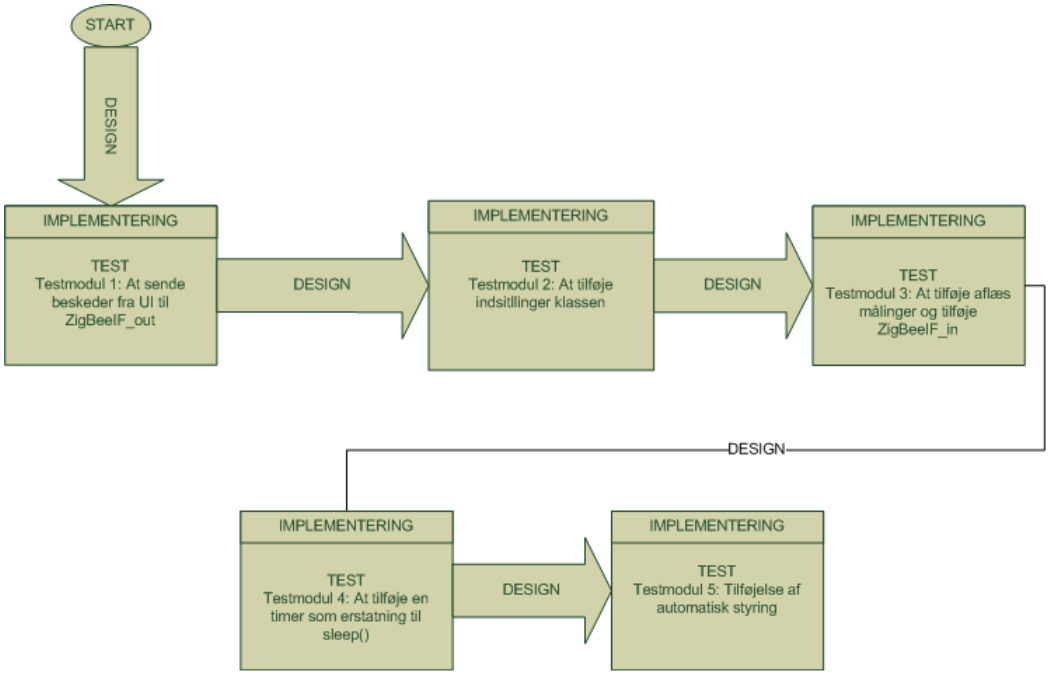
I stedet for først at designe alt for så derefter at implementere det og teste det, benyttede vi en iterativ arbejdsproces, hvor en mindre del af softwarens funktionalitet blev lavet ad gangen med både design, implementering og test. Denne cyklus blev så gentaget flere gange, hvor en ny funktion blev tilføjet ad gangen oven i det allerede implementerede. Dette blev vi bl.a. rådet til at gøre af vores vejleder, da vi på denne måde ikke skulle fokusere på så mange ting af gangen, samtidig med, at fejl blev lettere at lokalisere.

I figur 30 ses de forskellige lag af de mindre dele af softwarens funktionalitet. Figuren lægger ikke vægt på, at man skal kunne se detaljerne i den, men det er blot princippet i vores måde at udføre en iterativ proces på. Hvert lag blev testet, hvorefter et nyt lag blev testet og tilsidst blev alle grænseflader testet. Først blev grænsefladen til UIControlleren programmeret. Dernæst koblede vi QT's grænseflade på og dernæst koblede vi ZigBee og PSoC3 grænsefladen sammen. Denne arbejdsproces dannede tilsammen et hierarki med Devkit8000 grænsefladen i trin 1, 2 og 3 og ZigBee opkoblingen sammen med Devkit8000 og PSoC3 i trin 4, 5 og 6.



Figur 30: Iterativ process hieraki

Selvom vi på dette tidspunkt hverken havde QT user interfacet eller ZigBee grænsefladen til rådighed, implementerede vi i vores del også en ZigBeeoutput klasse uden nogen implementeret sendefunktion, og en UIcontrollerklasse, som fungerede som bindeledet til QT, og fra sidstnævnte kunne vi sende beskeder ned til de andre klasser i systemet ved at kalde denne klasses metoder (de samme metoder, som QT senere skulle tilgå). Vi valgte at implementere udskrifter til terminalen i alle beskedhåndteringerne, så vi kunne følge med i, om beskedudvekslingerne fortsatte som beskrevet i sekvensdiagrammerne, eller om kæden knækkede undervejs.



Figur 31: Iterativ process med design, implementering og test.

Selve implementeringen af softwaren på Devkittet blev som sagt udført som en iterativ arbejdsproces. Vi valgte at dele SW designfasen op i testmoduler, som det ses i figur 31, da implementeringen på denne måde kunne understøtte en iterativ arbejdsproces. Ved at bruge en iterativt metode kan man starte helt fra bunden med nogle basale tests og derefter udvide mere flere moduler. Dette gør, at det er nemt at vedligeholde samt debugge koden. Testmodul 1 var en simpel beskedudveksling fra UIController gennem ManualController, Uro, Vugge samt ZigBee boundrien. En illustration af dette testmodul er vist i figur 18, der vedrører eksemplet omkring brugen af MessageQueues. Dette testmodul viste, at det var muligt at få en strøm af basale beskedudvekslinger helt fra da UIController forespurgte om en vugning eller uro til at outputtet på ZigBee boundrien kunne bekræfte, at den kunne sende en 'start vugge' kommando til PSoC3.

Til bl.a. vugge-og urotiden brugte vi i første omgang en sleep funktion til at sætte tråden til at sove i den tid, der nu svarede til denne tid. Men for ikke at bremse beskedmodtageren mens tråden sover pga. sleep-funktionen, indførte vi et nyt begreb kaldt for "MessageKicker". Denne sørgede for at vække den sovende tråd hvert sekund og tælle dennes interne tæller op, indtil tælleren nåede den ønskede tid. Dette fungerede uden problemer på host, men vi erfarede, at timingen på target var anderledes, da den ca. var 1/3del bagud i forhold til host. Dette var selvfølgelig ikke acceptabelt, og vi konkluderede derfor, at vi måtte bruge en timer interrupt i stedet. OSAPI tilbød netop denne facilitet. Derfor brugte vi en funktion derfra, som direkte tilgik en timer.

I næste trin i testmodul 2 implementerede vi så systemindstillingsklassen og lavede et system, så man kan ændre på forskellige indstillinger og gemme disse til en fil. Indstillingsklassen sender i dette træk også beskeder videre til de klasser, som skal bruge disse indstillinger. I testmodul 3 implementerede vi klassen til at aflæse målinger, som var koblet sammen med UIcontroller-klassen og ZigBeeIF_in-klassen (bemærk at vi valgte at dele ZigBee-klassen op i to, da dette i implementeringsfasen viste sig at være en bedre løsning). I testmodul 4 tilføjede vi en timer som erstatning til sleep() funktionen. Denne test viste betydelig bedre timing på host såvel som på target. I testmodul 5 skulle klassen til den automatiske styring tilføjes, hvor denne klasse skulle arbejde sammen med hele fire af de andre klasser. Her skulle vi få ZigBee interfacet, der nu var delt op i to, til at kunne køre parallelt med hinanden. På denne måde kunne automatisk styring køre sin tråd i takt med at MeasurementsController, som var den klasse, der tog imod nye målinger, også kunne modtage og analysere input data, som automatisk styring kunne reagere på.

Selvom vi naturligvis stødte på flere udfordringer i forløbet, mener vi, at vores nøje tilrettelagte tilgangsvinkel var ganske fornuftig og gjorde hele processen mere håndterbar.

9.8.2 Devkit8000 Touch Interface - Anders

I projekt oplægget står der beskrevet at et krav til produktet skal have brugerinteraktion, og vi har i kravspecifikationen lavet det første udkast af hvordan det kunne se ud. Dette ses i projektdokumentationen afsnit 1.8.

Til udvikling af dette Touch interface er programmet, Qt Creator blevet brugt, senere refereret til som qt. Qt er et udviklingsværktøj på lige fod med Visual Studio som vi kender fra de tidligere semestre. Grunden til at qt er blevet valgt er ud fra at vi dette semester har haft en del programmering på en Linux platform og qt blev anbefalet af faglærerne til design af UI. Qt indeholder en lang række værktøjer som gør det muligt at opbygge en bruger interface ved "drag and drop" af knapper og andre moduler. Ud fra kravspecifikationens design udkast, er bruger interfacet blevet til. Implementering og testning af disse moduler er blevet gjort iterativt. Dog har vi ikke rigtig lært hvordan man designer et bruger interface så det er blevet gjort efter bedste overbevisning og et tilstands diagram er blevet lavet, dog løbende ændret for til slut at tilpasse de mange ændringer til det færdige produkt. Tilstands diagrammet kan ses i projektdokumentationen afsnit 2.12.

I gruppen mener vi at bruger interfacet er en vigtig del af produktet, da vores produkt afhænger meget af brugerinteraktion og feedback til brugeren, så bruger kan agere efter behov, og vi har derfor valgt at sætte en mand med hovedansvaret for denne post. Som tidligere nævnt så giver qt, et "drag and drop" interface hvor man kan trække de ønskede moduler ind i dens design flade og man har derefter mulighed for at programmere hvert enkelt modul efter ønske. F.eks. har vi valgt at bruge knapper til aktivering af manuel og automatisk styring. Når man har med brugerinterface at gøre så kan man ikke komme uden om at det jo er eventstyret og dermed brugeren som starter disse events via touch skærmen, hvilket gør os i stand til at sikre os at der ikke sker noget uventet.

Kort beskrivelse af bruger-interfacet
Brugerinterfacet består af to vinduer, hvor det ene er Hovedmenuen hvor brugeren kan aktivere manuel og automatisk styring samt se løbende hvad temperatur inde i barnevognen er og hvad lydniveauet er inde i vognen. Der er indbygget en timer som hvert 1 sekund kalder et event som henter data og opdatere dermed displayet på Devkittet. Dertil er der en knap som gør det muligt at åbne systemindstillinger, dette gøres ved at trykke på knappen "SystemIndstillinger" hvilket tager en videre til et andet vindue, og her kan man indstille hvor langt tid barnevognen skal vugge mm. Der er desuden en default knap som indstiller systemet til standard indstillinger, og knappen gem, som tager en tilbage til hovedmenu og ligeledes gemmer indstillinger ned i en fil. Til slut knappen annuller som tager en tilbage til hovedmenuen uden at gemme indstillingerne.

Til implementeringen blev der oprettet en klasse ved navn, UiController som bliver brugt som vores bindeled, en såkaldt bro kobling. Når et qt event sker, kaldes der en funktion på denne bro som så går videre i systemet, hvilket der ikke beskrives yderligere her.

Til testning af brugerinterfacet er der for hver gang et nyt modul er blevet designet, blevet testet det givne modul i et separat Qt projekt. Dette er gjort for at sikre en bedre tilgang til softwaren hvilket også gør os i stand til at følge udviklingen hele vejen til det færdige design og lige så meget for at sikre at modulet bliver brugt på den rigtige måde og at det fungerer uden for det samlede Qt projekt. Testning af denne del af projektet har hængt meget sammen med resten af softwaren der kører på devkittet og vil derfor blive beskrevet mere der.

9.8.3 PSoC3 - (Jonas) Mads & Joachim

The basic structure of the program has been determined. Recall from the design chapter, there is an eternal loop running through the four phases. This is implemented as a for(;;)-loop around all the four phases, with each phase represented by a while-loop.

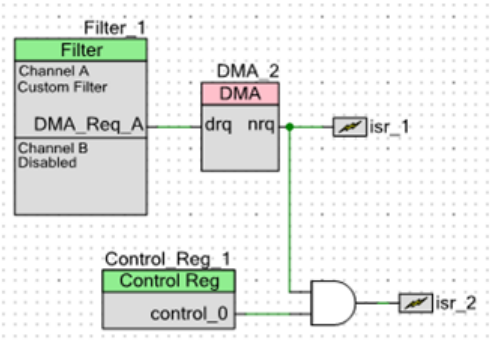
```
void main()
{
    Init();

    for(;;)
    {
        // Phase 1
        // Phase 2
        // Phase 3
        // Phase 4
    }
```

Figur 32: Main phases

The phases were implemented in iterations, preventing too big changes in code between every running version of the program. For debugging purposes a UART USB device was used in combination with the UART component in PSoC Creator, making it possible to output data to a computer running Tera Term. The digital to analog converter component on the PSoC was also used in testing, and still remains in the design, though not being accessed in the final code.

Phase 1

In each run through the while-loop, peak-to-peak values are calculated from the microphone input. When a new sample is ready in the ADC, it is sent to a DMA, transferring this data further to the filter. From the filter, the sample enters another DMA, transferring it to a variable holding the sample. When the latter DMA has saved the sample, an interrupt service routine (isr_2) is triggered. The control register makes sure the program is ready for a new sample, before triggering the ISR.

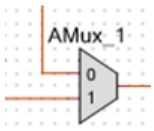


Figur 33: Components for phase 1

In this routine, the sample is evaluated, checking if it is the highest or lowest value so far. If so, it is saved in the max / min value variable respectively. For every 20 elements in the sound array, a new peak-to-peak value is calculated. This continues until an array is full. It should now contain 20 values, representing 1 second of sound all together. The program advances to phase 2.

Phase 2

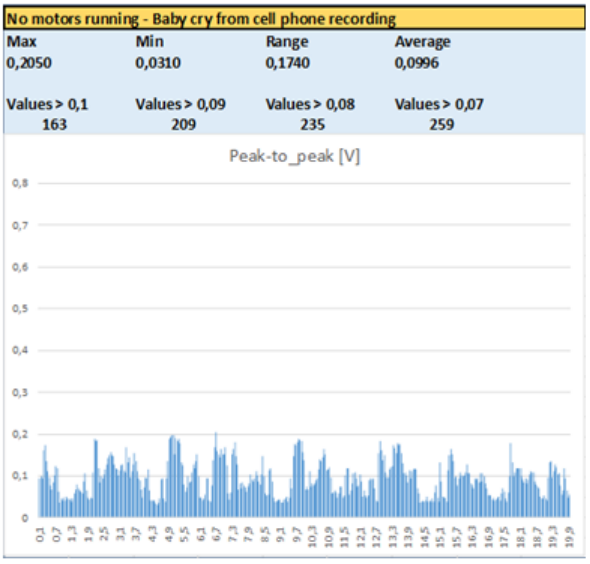
This phase's while-loop is run through once. First, the MUX is accessed, changing the ADC input to thermometer. The temperature voltage is then recorded and converted into a temperature in °C, using the analyzeTemp() function. Before moving on to phase 3, the MUX is accessed to change the ADC input back to microphone.



Figur 34: A mux component

Phase 3

The array with sound information is analyzed, returning a value between 0 and 100 representing the noise level. The analysis focuses on the average peak-to-peak value of the sampled sound, but also scales the noise level based on some other qualities, such as range and value counts. The following graph shows part of the results from the sound testing. This testing was used to make the analyzeSound() function more useful. Tests were made with and without a baby crying, making it possible to differentiate between the two recordings, and adapting our analysis function thereafter. This is discussed in length in the project documentation.




```
while( phase == 3 )
{
    soundResult = analyzeSound( soundArray, SOUND_SIZE );
```

Figur 35: Phase 3

This phase further sends the analyzed sound and temperature information to the DevKit. This is implemented by calling functions in the ZigBee Driver. The program is then told to move back to phase 1.

Phase 4

Every time phase 3 is completed, the program might go into phase 4. This will only happen if the hibernation stopwatch is too high (remember this is reset upon receiving a transmission from DevKit). If this phase is entered, both motors are sent a PWM value of zero, putting them to a halt. The program then does nothing until it receives a new transmission from DevKit.

ISR ZigBee:

As discussed, the ZigBee module triggers an interrupt on the PSoC upon receiving a new transmission from DevKit. However, the same interrupt is triggered when a transmission is sent successfully to DevKit. The first thing that needs to happen in the ISR ZigBee should thus be checking the type of interrupt triggered. In the case of an incoming transmission interrupt, the data is read from ZigBee. Then the new PWM values are sent to the PWM component, and further out to the motor pins.

9.9 Resultater - (Anders)

Forventede mål:

Forventning:	**Beskrivelse:**	**Resultat:**
Trådløs data kommunikation	Data kommunikation ved hjælp af Zigbee	Systemet fungerer ved brug af trådløs kommunikation mellem Devkit8000 og PSoC3
Data indsamling	Lydniveau og grader celsius indsamles via. Sensorer indbygget i barnevognen, til PSoC3	Der modtages data fra PSoC3's sensorer
Data behandling	PSoC3 behandler den indsamlede data og sender værdier til Devkit8000 vha. den trådløse kommunikation	Den indsamlede data bliver behandlet og videresendt
Bruger interaktion	Det skal være muligt for brugeren af systemet at have et grafisk interface at benytte sig af	Bruger har muligheden for at benytte et grafisk interface
Systemkontrol	Når bruger benytter grafisk interface forventes det at systemet udfører brugerens valg	Systemet opfylder brugerens forventning når det grafiske interface bliver anvendt.
System feedback	Systemet bliver løbende opdateret med de indsamlede data og vises på den grafiske brugerflade	Systemet modtager løbende data og opdaterer den grafiske brugerflade ca. hvert sekund
System interaktion	Systemet snakker internt sammen ved hjælp af beskedkøer	Systemet benytter beskedkøer og det virker efter hensigten.

Trådløs data kommunikation
Vi havde fra starten sat os det mål at kunne sende data trådløst mellem vores 2 moduler og vi har opnået et tilfredsstillende resultat på dette punkt. Der bliver brugt SPI mellem PSoC3 og Zigbee og SPI mellem Zigbee og Devkit8000, som gør, at vi opfylder et af projektoplæggets krav.

Dataindsamling
Til data indsamling har vi ved hjælp af en temperatur sensor og en mikrofon været i stand til at opsamle data og modtage på PSoC3. Projektoplægget beskriver, at der skal benyttes sensorer, hvilket derfor opfylder dette punkt.

Databehandling
Den indsamlede data bliver på PSoC3'en behandlet for så at blive sendt via til devkittet via den trådløse kommunikation. Der modtages de samme værdier på Devkittet, som sendes fra PSoC3'ens side, og vi er tilfredse med dette resultat.

Brugerinteraktion
Der er blevet bygget et brugerinterface hvor brugeren har mulighed for at aktiver og ændrer på systemet efter ønske. Dog var ønsket at dette grafiske interface skulle starte op selv når devkittet bliver startet, men på grund af problemer med en journald fil som påtog sig 97% af devkittets cpu kraft, gjorde at vi på et tidspunkt fravalgte dette og måtte fokusere på produktet og opnå et resultat som vi kunne være tilfredse med. Derfor må dette program startes med en computer.

Systemkontrol
Systemet udfører den forventede opgave, som brugeren betrykker det, og ligeledes når bruger trykker for stop, stopper vugning/uro som forventet. Systemets kontrol opfører sig som forventet og er et tilfredsstillende resultat for gruppen.

System feedback
Devkittet modtager løbende data fra PSoC3'ens sensorer og bliver vist på brugerinterfacet så brugeren kan følge med i hvad temperaturen er i barnevognen, og hvor høj lyden er omkring baby. Dog er lyden i procent, men i forhold til hvad. Vi mener at dette punkt kunne have været gjort bedre, og at lyden i procent ikke giver den store mening for brugeren, da brugeren ikke har nogen information om hvad 100% procent lyd betyder, udover at det højst sandsynligt er højt. Dog er det en skala, der kan indstilles efter behov, men burde nok have været anderledes.

Systeminteraktion
Internt i systemet bliver der på devkittets side benyttet beskeder og beskedkøer til kommunikation mellem de 9 forskellige tråde systemet benytter sig af. Vi har i undervisningen lært om tråde og kommunikationen imellem og mener derfor, at det er fagligt brugbart. Vi har ved testning bekræftet at kommunikationen mellem trådene sker og at det derfor er et tilfredsstillende resultat for systemet.

Overordnet
Vi synes i gruppen, at produktet overordnet fungerer, som det kunne forventes. Dog finder vi også nogle ting, som kunne laves om eller burde laves om, men på et eller andet tidspunkt blev vi nødt til at beslutte os for, at vi ikke kunne nå længere for projektet. For at tage en ting, som vi er specielt stolte over, så er det vores trådløse kommunikation.

9.10 Opnåede erfaringer

David

I projektarbejdet har jeg arbejdet med softwaren på Devkittet, hvor jeg har været med til at implementere et MessageQueue system, som baserer sig på det, vi har lært i kurset "Indledende System Engineering". Min rolle har været at strukturere designet til den overordnet software men også til den detaljerede del til Devkittet og dens implementering. Jeg har også haft en mindre rolle, hvor jeg har været sekretær og fik lavet mødereferater. Ydermere har jeg også hjulpet med planlægningen og gjort gruppen opmærksom på deadlines. Jeg har opnået et par erfaringer, som jeg synes har hjulpet rigtig meget på overblikket af projektet, hvilket kan give en optimeret ydelse i fremtidige projekter. I min del af softwareafdelingen sammen Sander Louring lærte og anerkendte vi, hvor nemt og overskueligt det var at udføre en iterativ proces med testmoduler, som blev bygget ovenpå hinanden lidt efter lidt. Samarbejdet gjorde også at læringen og forståelse for, hvordan de små brikker i systemet hængt sammen var meget god.

Designvalget som beskrevet i afsnit 9.6.1 gjorde, at man nemt kunne se sammenhængen i mellem diagrammer og beskrivelser og selve implementeringen. Det var tydeligt at ved hjælp af et MessageQueue system, blev afstanden fra design til implementering betydelig mindre. Dette havde den konsekvens, at selve programmeringsdelen til Devkittet var meget mere håndgribeligt. Generelt kan jeg sige, at vi var i en tidlig fase, hvor ansvarsområderne blev fordelt ud, og vi havde styr på, hvem der skulle lave hvad.

Mads

Læringen for mig på dette semester har bestået af at tilegne mig mere viden omkring udviklingsfasen fra design af et selvvalgt produkt og dets kravspecifikation. Det har været et lærerigt semester med en masse udfordringer bestående af at opretholde et samarbejde samtidigt med at man individuelt lavede en projektdel til projektet. Jeg har primært beskæftiget mig med at lave hardwaren på barnevognen og har arbejdet rigtig tæt sammen med at udvikle den software der ligger på PSoC3'en. Det var egentlig planen at jeg skulle udvikle softwaren på PSoC3'en da den var så hardware nær, men det blev der lavet om på. Det har været sjovt at skulle implementere den viden man fik i fagende i sit projekt, men også en udfordring da der har været mindre hardware end software på dette projekt. Til gengæld har jeg beskæftiget mig meget med at administrere projektgruppen og taget mødereferater f.eks. og også stået for en del flere punkter i projektrapporten. Projektet har taget form stille og roligt i løbet af de sidste par måneder men har udviklet sig hurtigt til sidst eftersom vi først fik tildelt viden om hvordan vi kan løse nogle af projektets problemer i dette stadie. Det har presset os i sidste del af projektfasen men vi har nået at lave et produkt der lever op til de forventninger vi havde i starten af projektet. Så det har været et lærerigt semester der har lært mig mere om hvordan en ingeniør tilegner sig viden, bearbejder den og for udviklet et endeligt produkt!

Joachim

Jeg har personligt opnået en større erfaring inden for det, at arbejde med hardware på et software-niveau. Her taler jeg selvfølgelig om Cypress' PSoC3. Denne har åbnet op for en masse nye muligheder, og det ses virkelig hvor anvendelig sådan en type mikrocontroller-kit kan være. Denne kan bl.a. spare en for en masse tid og arbejde, idet fysiske komponenter nu er komponenter inden for PSoC3'ens rammer(komponenter indsættes og programmeres i selve PSoC3 softwaren). Især grænsesnittet fra den fysiske verden til PSoC3'ens interne komponenter,

er der hvor jeg har fået større erfaring, da det primært har været mit arbejdsområde i dette projekt. På et overordnet plan, har jeg også fået en større forståelse for SPI-protokollen og dens anvendelse i den virkelige verden. Ydermere har mit forhold til SysML-diagrammer ændres sig markant. Disse er ikke længere nogle diagrammer, som bare skal implementeres (som på 2. semester, hvor fokus lå herpå) for alt i produktet, men derimod guidelines til en selv, der giver overskuelighed og forståelse – noget som implementeres i et passende omfang. Når SysML netop bruges i et passende omfang, er dette et rigtig stærkt værktøj, til at få struktur og adfærd på ens produkt. Det er især stærkt i starten af projektet.

Kjeld

I dette projekt kom der endelig mere fokus på teknologien, hvilket jeg har savnet lidt i de foregående projekter. Mit største bidrag til dette projekt var at foreslå at bruge trådløse Zigbee moduler, for jeg har brugt noget lignende tidligere i min karriere, så hvor svært kunne det være? Efter som det var mit forslag at bruge disse moduler, bestilte vi nogle hjem, og så påtog jeg mig ansvaret at finde ud at bruge dem og skrive driverne til dem. Nu skulle det vise sig at der er sket lidt siden sidst jeg brugte Zigbee moduler, blandt andet var disse moduler baseret på en helt andet og noget mere avanceret chip end sidst jeg havde fat i dem. Dette præsenterede selvfølgelig lidt af en udfordring, men i bund og grund var det jo bare at lære dens grænseflade at kende. Yderlige skulle der oparbejdes en forståelse for datapakkestrukturen som er beskrevet i IEEE standarden 802.15.4, som beskriver dette. Men det hele lykkedes, og et par udmærkede driver blev skrevet til dem, lige til at bruge. Alt i alt er jeg meget tilfreds med resultat af vores projekt. Det var en stor fornøjelse at se da Zigbee modulerne blev sat i printene og driverne indlæst af det overordnede software, og så se det bare virkede.

Anders

Efter min opfattelse så har dette semesters projekt fungeret bedre end sidste års semester. Sidste år var jeg i en gruppe hvor et par stykker i ikke arbejdede særlig godt sammen og ikke rigtig bidrog lige meget til projektet. På dette semester synes jeg vi internt i gruppen har været gode til at arbejde og aftale fra uge til uge hvad der skulle laves og det er blevet lavet med få undtagelser. Folk har været gode til at sige til hvis opgaven har været for stor og havde brug for hjælp. I projektet har mit arbejde været at stå for at lave brugerinterface på devkittets touch skærm, og jeg begik den dumhed at tro at Qt, som er programmet der er blevet brug, vær noget man bare lige lærte på en dags tid og jeg kunne ikke have været mere forket på den. Jeg synes jeg har fået nogle gode erfaringer med hvordan et brugerinterface er opbygget og hvor stort et arbejde der ligger i opbyggelsen af samme. Dog ville det ikke være skidt med lidt undervisning på semesteret da semester projektet ligger direkte op til at have et grafisk brugerinterface da der i projekt oplægget direkte står at produktet skal have bruger interaktion og at man så har en touch skærm til rådighed.

Kristian

Igennem projektarbejdet har jeg arbejdet med både NTC thermistor, Wheatstone bridge, PSoC3 ADC og instrumenteringsforstærker mm. Hovedsageligt har jeg investeret tid i hardware og har fået meget større erfaring indenfor kredsløbsanalyse, specielt når det kommer til støj og offset-analyse. Jeg har fundet ud af, hvordan en NTC thermistor fungerer, hvilke fysiske egenskaber den har og hvordan den interagerer med omverden. Hvilken betydning og usikkerhed der ligger i at analysere på enheder som temperatur, fra den fysiske verden. Wheatstone bridgen har jeg brugt til at måle de små ændringer i modstandsværdi, og igennem en instrumenter-

ingsforstærker sende den differentielle spænding ind i en ADC. Instrumenteringsforstærkeren AD620 har jeg skulle sætte ind i, hvordan den fungerer, hvilke pins den har og hvilken betydning de har for outputtet. Jeg har fundet ud af hvordan man designer et input signal til PSoC3 og ADC, så man får mest muligt arbejdsområde - det er vigtigt, at hæve arbejdspunktet til Vdda/2.

Igennem semestret har jeg stået for administrationen af SCRUM – hvor backlog og sprints skulle opdateres løbende. Det har været godt til at holde styr på hvor langt folk var kommet med deres del af de enkelte delopgaver til projektet. Nogle af elementerne i software har været svært at designe på forhånd og få skrevet ind i backloggen – det kunne have været en god idé at bede alle medlemmerne i softwareudvikling om, at blive mere enige om et design inden de går i gang med implementering.

Opgaven var meget frit stillet, så derfor har den lagt op til, at vi selv skulle ud og opsøge viden som ligger udenfor skolens undervisning. Jeg har bl.a. benyttet mig at Torben og Heidi på ASE til design af den mekaniske vuggekonstellation – selv boret og skruet for at få en velfungerede SmartPram prototype.

Sander

I dette projekt har jeg udover at arbejde med de indledende projektfaser fokuseret på design, implementering og test af softwaren på DevKit sammen med David Buhauer. Dette har givet mig en bedre forståelse for vigtigheden i at arbejde ud fra velovervejede designvalg, som her, hvor vi besluttede os for at benytte et messageQueue-system til kommunikationen imellem tråde. Den iterative angrebsvinkel, som vi havde på udviklingen af softwaren, ser jeg også som et meget effektivt værktøj, der gjorde hele processen mere overskuelig og sikrede os et mere struktureret resultat, og denne arbejdsmetode vil jeg bestemt benytte i fremtidige projekter.

Af egne styrker vil jeg selv vurdere, at jeg er god til at bevare et godt overblik over strukturen i en kode, og at jeg af denne grund også er hurtig til at finde på løsninger på de designmæssige problemer, der måtte opstå. Jeg føler desuden, at jeg er god til at se på en projektformulering fra en indledende fase og forvandle denne til konkrete opgaver, der skal implementeres.

Jeg er selv blevet godt tilfreds med resultatet og arbejdet til at nå dertil, og jeg synes generelt, at gruppens måde at arbejde på har fungeret rigtig godt. I forhold til sidste semesterprojekt føler jeg, at dette projekt er gået mere smertefrit, da vi alle løbende får bedre erfaringer med det at lave projekt og derfor også bliver bedre til at gribe det an.

Jonas

Through the project I have mainly worked on the PsoC 3 software, controlling the pram in the system. It has been a useful experience, leading to insight into the PSoC hardware and the software controlling it. Throughout the project, I have collaborated with the hardware group in finding the best solutions for the software, as well as for the interfaces to the hardware itself. This semester project was new to the group, in the way that this is the first semester project where we could choose what to work with ourselves. We got to experience the entirety of developing a system, from coming up with the ideas, planning and implementing them, to organizing the roles within the group. As the project requirements were quite open, the group relied upon seeking knowledge independently. After coming up with the main requirements for the PSoC's functionality, I got to exercise this, reading datasheets, watching video tutorials

and seeking other information on the PSoC and matching software. I think this is one of the most important qualities to learn for an engineering student, as knowledge, technology and programming languages change all the time. Using this approach to finding information, I have learned a lot about the PSoC 3 and PSoC Creator. What I find especially useful, are the integrated hardware components of the PSoC. The fact that these are accessible in the graphical user interface in PSoC Creator, makes the control of inputs from sensors and output to actuators much more simple. Another thing I take away from this project is an improved understanding of how the software and hardware work together, and of the interfaces between these.

9.11 Udviklingsværktøjer - Joachim

Kubuntu
Kubuntu, der er en afledt udgave af Ubuntu, er i dette projekt, blevet benyttet i et bredt omfang. Det er herfra, at vores Devkit8000 er blevet programmeret. Linux-miljøet har altså spillet en større rolle i projektet.

Devkit8000
Devkit8000 er som bekendt det target, der er arbejdet med, og som har sørget for funktionaliteten(brugergrænsefladen) til brugeren af systemet.

PSoC3
Dette stykke hardware har suppleret vores færdige produkt med adskillige funktionaliteter. Det er PSoC3'en, som har direkte kontakt til de forskellige sensorer, motorer, aktuatorer mm., og som formidler/behandler disse analoge/digitale data. Ydermere er kommunikation via SPI også sat op i PSoC3. Dette stykke hardware har altså været en essentiel del af vores samlede produkt.

PSoC Creator 3.0 Component Pack 7 PsoC Creater 3.0 er blivet benyttet til at skrive alt software for PSoC3.

Multisim 12
Multisim har været brugt til, at designe kredsløb inden for HW, herunder de forskellige delkredsløb, som med de enkelte sensorer at gøre. Multisim er ydermere blevet brugt til simulering af visse dele i HW-designfasen.

Mathcad15
Mathcad er blevet brugt I et bredt omfang, til generelle matematiske beregninger. Her har Mathcad været et rigtig stærkt værktøj.

QTCreator 2.7.0
Qtcreator er et program som gør det muligt at opbygge en bruger grænseflade, som tager brug af event driven programmering.

LaTex TexMaker
LaTex værktøjet er blevet brugt til layouttet og (David) har været med til at finpudse selve rapporten og dokumentationen. Dette har været et yderst godt værktøj til at holde styr på hvert afsnit og figure, som der skulle refereres til.

10 Konklusion (Jonas, Joachim & Mads)

Vi har formået at lave et projekt, som opfylder de vigtigste krav, som var stillet. Det er lykkedes os at få en trådløs styring mellem Devkit8000 og PSoC3. Ved at tilgå DevKit'et, kan brugeren sætte gang i vugning af barnevognen eller en roterende uro. Disse vil også kunne igangsættes af DevKit via den automatiske styring. Ydermere kan systemet måle temperatur og lydniveau ved barnevognen for så at sende dette tilbage til DevKit. Vi har ikke formået at streame lyden og afspille denne på DevKit'et. Begrænsningen lå i, at Devkit ikke kan afspille lyden fra PSoC'en. Det blev i stedet besluttet, at lyden skulle analyseres ovre på PSoC, og en værdi, som repræsenterer lydniveauet, sendes over til DevKit'et. Selv om streaming af lyd ville have være et godt element at have med, vurderede vi, at en værdi for lydniveauet ville være tilstrækkeligt for at afgøre, om babyen græder.

Brugergrænsefladen var tiltænkt at skulle starte op automatisk, men dette er ikke lykkedes pga. tidspres. Vores resultater af projektet stemte næsten ellers overens med den accepttest, som vi fik designet tidligt i projektet. Vi har hermed formået at levere et produkt, som vi selv mener, at vi kan være stolte af. Med hensyn til forbedringer af projektet, har vi udarbejdet en fremtidig-implementeringsliste, som forefindes i projektdokumentationen. Heri indgår der ideer om, hvorledes man kan videreudvikle produktet.