Bachelor Afstudeer Project BabyShell

Microcontroller, display, opslag, user interface en medische functionaliteit

Paul Bakker
Alexander Jongeling

Supervisors:
dr.ir. Andre Bossche
Jeroen Bastemeijer

Opdrachtgever:
Maurizio Bricola

Technische Universiteit Delft

TU Delft

Samenvatting

In ontwikkelingslanden zijn er nog altijd veel problemen met de gezondheid van zwangere vrouwen en jonge kinderen. Daarom is voor dit Bachelor afstudeerproject BabyShell ontwikkeld: een draagbaar apparaat dat onafhankelijk is van het stroomnet en door middel van een vraag-antwoord proces diagnoses voor medische problemen kan stellen. BabyShell is verder in staat om op het juiste moment herinneringen te geven voor prenatale doktersbezoeken en vaccinaties, en de temperatuur te meten. Deze thesis beschrijft de microcontroller, het display, de opslag, de user interface en de medische functionaliteit van BabyShell. Geconcludeerd kan worden dat er aan de eisen die werden gesteld aan het apparaat is voldaan.

Inhoudsopgave

1

Inleiding

Helaas is kindersterfte nog steeds een groot probleem in deze wereld. In west en centraal Afrika haalt nog steeds minder dan één op de tien kinderen de leeftijd van vijf jaar [1]. Ook sterven er per dag nog 800 vrouwen ten gevolge van zwangerschapscomplicaties die te voorkomen zijn, waarvan 99% in ontwikkelingslanden [2]. Gelukkig heeft er de afgelopen decennia al een significante verbetering plaatsgevonden op het gebied van medische voorzieningen in derdewereldlanden. Vele levens worden gered door bijvoorbeeld mHealth initiatieven [3] die gebruik maken van het GSM netwerk om eigenaars van mobiele telefoons snel en draadloos toegang te geven tot medische diensten. Echter is er nog steeds verbetering mogelijk. Er zijn bijvoorbeeld nog steeds veel mensen in Afrika die geen mobiele telefoon in het bezit hebben [4], en in het arme deel van het continent hebben nog steeds veel mensen geen toegang tot elektriciteit [5]. Hoe kunnen zij voorzien worden van noodzakelijk medische informatie en hulp op het gebied van zwangerschap en jong ouderschap op een zo goedkoop mogelijke manier? Dit is waar BabyShell om de hoek komt kijken.

BabyShell is het eerste draagbare medische apparaat dat als doel heeft om moeders in derdewereldlanden te helpen bij hun zwangerschap en hun jonge kinderen. BabyShell werkt compleet onafhankelijk van het stroomnet en de componentkosten bedragen minder dan 6 dollar om ervoor te zorgen dat zoveel mogelijk mensen zich het apparaat kunnen veroorloven. Door middel van een flowchart zal de gebruiker via een vraag-antwoord proces naar een mogelijke diagnose geleid worden als er iets mis is met het kind. Ook zal BabyShell in staat zijn om taken te verrichten zoals tijd bijhouden, temperatuur opnemen en reminders versturen voor belangrijke gebeurtenissen als prenatale doktersbezoeken en vaccinaties.

Het ontwerpproces van BabyShell en de keuzes die gemaakt zijn tijdens het project zullen beschreven worden in drie verschillende theses met de volgende onderwerpen:

1. Voeding, batterij management en PCB ontwerp
2. RTC en thermometer
3. Microcontroller, display, opslag, user interface en medische functionaliteit

In deze thesis zal het onderwerp ”microcontroller, display, opslag, user interface en medische functionaliteit” worden behandeld. Hierin wordt hoofdzakelijk het volgende besproken:

- Hoe de keuze tot stand is gekomen voor de in deze thesis gebruikte hardware
- Hoe het scherm wordt aangestuurd
- Hoe teksten op een goed leesbare manier op het scherm worden getoond
- Hoe met een beperkt aantal knoppen in combinatie met een bepaalde menustructuur de user interface wordt geïmplemteerd

- Hoe verschillende soorten data op verschillende opslagmedia worden opgeslagen
- Hoe de medische functionaliteit softwarematig op het apparaat is geïmplementeerd

2

Probleemdefinitie

2.1. Opdracht

Zoals in de inleiding is besproken zal er een apparaat moeten worden ontworpen dat als doel het helpen van zwangere vrouwen en moeder met jonge kinderen in derdewereldlanden heeft. Door het doorlopen van flowcharts in het systeem kan de gebruiker tot een mogelijke diagnose komen bij verschillende medische problemen. Ook zal het apparaat herinneringen moeten geven als het tijd is voor de vaccinaties van een kind en een wekelijkse tip geven tijdens de zwangerschap van de moeder om haar bijvoorbeeld te herinneren aan een doktersbezoek. Met een temperatuursensor die gebruikt kan worden om de lichaamstemperatuur te meten en een klok krijgt het apparaat toegevoegd praktisch nut. Dit alles moet het apparaat kunnen zonder afhankelijk te zijn van het stroomnet.

2.2. Programma van eisen

In overleg met de opdrachtgever is een programma van eisen opgesteld, welke de functionele specificaties van het apparaat beschrijft. De eisen die specifiek van toepassing zijn op deze thesis worden dikgedrukt weergegeven.

1. BabyShell is gedurende de levensduur onafhankelijk van het stroomnet en kan minstens 1 uur buiten de zon gebruikt worden bij een volle batterij.
2. De levensduur van het apparaat is minstens 4 jaar.
3. **Het hoofdmenu bestaat uit minstens de twee onderdelen:**
   (a) **Medical (met daaronder de twee subonderdelen ”Pregnancy” en ”New born”)**
   (b) Klok
4. **Wanneer ”Pregnancy” wordt gekozen, moet eenmalig de datum van de laatste menstruatie ingevoerd worden. BabyShell moet dan op basis van de opgeslagen tijd de weken bij gaan houden en de moeder wijzen op prenataal doktersbezoek. BabyShell moet ook wekelijks medische tips geven die toepasselijk zijn in die zwangersschapsperiode volgens bron [6].**
5. **Wanneer ”New born” wordt gekozen, moet eenmalig de geboortedatum ingevoerd worden. Vervolgens moeten er vaccinatie herinneringen voor kinderen tot 3 jaar gegeven worden volgens bron [7].**
6. **Onder ”New born” moet ook een kopje zijn met ”Diagnosis” waar de flowcharts geïmplementeerd worden voor de herkenning van longontsteking en oorinfecties met behulp van input van de gebruiker. Op basis van deze diagnose geeft BabyShell een medisch advies of verwijst naar een medische post.**
7. BabyShell is een draagbaar apparaat met een gewicht van minder dan 500 gram en past in een doos van maximaal 100x100x20mm en minimaal 40x40x5mm.

8. **Het is mogelijk om de software van de BabyShell te updaten met een computer.**
9. **De materiaalkosten van deze versie van BabyShell mogen niet meer dan $6 bedragen.**
10. Een thermometer, geïntegreerd in de BabyShell, maakt het mogelijk voor de gebruiker om de lichaamstemperatuur onder de oksel te meten en te gebruiken bij de diagnose.
11. De behuizing van Babyshell moet in verband met hygiene schoon te maken zijn met een doek met water en/of alcohol en is spatwaterdicht.
12. **BabyShell moet gegevens van tot en met 5 kinderen bij kunnen houden en per kind tijdsafhankelijke tips kunnen geven.**
13. **BabyShell moet per kind een resetfunctie hebben om in het geval van foute of gedateerde invoer te kunnen resetten.**

Technische implementatie

Voor het onderdeel van het project dat in deze thesis besproken wordt, moest er van de volgende hardware gekozen worden welk type er gebruikt ging worden:

- Microcontroller
- Bediening
- Display
- Opslag

De keuzeverantwoording voor elk van deze hardware onderdelen en de technische specificaties worden besproken in het hoofdstuk ”Ontwerpproces Hardware”, zie hoofdstuk 4. Met het maken van de keuze voor de microcontroller is ook besloten in welke programmeertaal het programma werd geschreven.

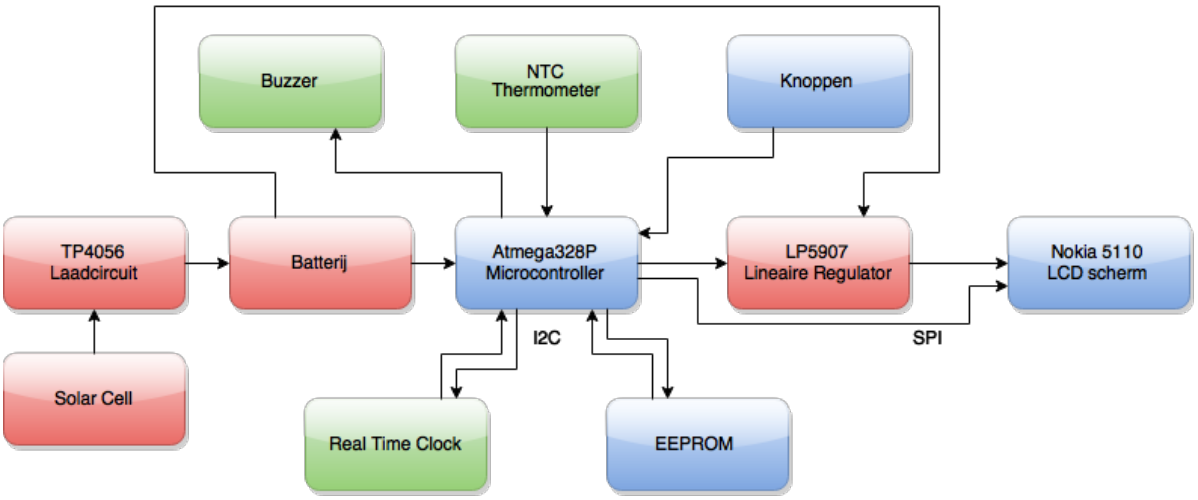
3

Verwant onderzoek

In dit hoofdstuk worden de onderwerpen besproken die buiten deze thesis vallen maar toch van belang waren voor het ontwikkelen van BabyShell.

3.1. Het complete systeem

Zoals reeds beschreven gebeurt het ontwikkelen van BabyShell in drie verschillende groepen. Om goed begrip te krijgen van hoe het het apparaat als geheel werkt, is het belangrijk om het systeem eerst als geheel te beschrijven voordat er wordt ingegaan op de deelgroep specifieke onderwerpen. Een blokschema van BabyShell, in dit geval de versie met zonnepaneel, is hieronder te zien.



Figuur 3.1: Blok schema BabyShell met zonnepaneel

Het project is opgedeeld in drie kleinere deelprojecten:

- Microcontroller, display, opslag, user interface en medische functionaliteit door Paul Bakker en Alexander Jongeling
- Voeding, batterij management en PCB ontwerp door Lars van Leeuwen en Tim Cheung [8]
- Slaapstand, RTC en thermometer door Gerard Hogenhout en Kees Hogenhout [9]

Deze thesis beschrijft het deel van de microcontroller, display, opslag, user interface en medische functionaliteit.

3.1.1. Voeding, batterij management en PCB ontwerp
Dit deelproject heeft gekeken naar de meest geschikte vorm van energievoorziening voor BabyShell. Er is geconcludeerd dat het het beste zou zijn om twee versie te bouwen: één op zonne-energie en één op batterijen. Beide versies voldoen volledig aan de eisen die gesteld zijn door de opdrachtgever. Ook is er onderzoek gedaan naar het mogelijke PCB ontwerp voor BabyShell. Hiervoor zijn twee ontwerpen gemaakt. Beide onderwerpen worden besproken in thesis [8].

3.1.2. Slaapstand, RTC en Thermometer
Dit deelproject heeft onderzoek gedaan naar de temperatuursensor, RTC (Real Time Clock) en de meest energiezuinige slaapstand. De temperatuursensor bestaat uit een NTC weerstand van $10K\Omega$ en wordt uitgelezen met een analoge ingang op de microcontroller. De gebruikte RTC is een DS3231, een zeer energiezuinige RTC met interne oscillator en temperatuurcompensatie. Deze communiceert over het I^2C protocol met de microcontroller.
Ook het in slaapstand brengen van de microcontroller (ook wel Power Down modus genoemd) door middel van de functie *sleepNow()*, wordt besproken in deze thesis [9].

3.2. Zwangerschap en jonge kinderen
De functionaliteit die BabyShell heeft, betreft een specifiek onderwerp: de gezondheid van zwangere vrouwen en jonge kinderen. Binnen de projectgroep was hier weinig kennis over aanwezig. De informatie betreffende dit onderwerp die op het apparaat geïmplementeerd moest worden werd aangeleverd door de opdrachtgever. Omdat er werd gedacht dat enige kennis over dit onderwerp het ontwerpen van het product ten goede zou komen, is er onderzoek naar gedaan.

3.2.1. Zwangerschap
Dankzij de Millenium Development Goals zijn er de afgelopen 25 jaar 50% minder vrouwen doodgegaan aan zwangerschapcomplicaties [10]. Toch sterven er per dag nog 800 vrouwen ten gevolge van zwangerschapscomplicaties die te voorkomen zijn. Hiervan is 99% in ontwikkelingslanden [2]. De meest voorkomende oorzaken van deze sterftes zijn [11]:

- Bloeding tijdens de zwangerschap of tijdens het bevallen
- Complicaties tijdens een abortus
- Bloedvergiftiging
- Hoge bloeddruk

3.2.2. Jonge kinderen en IMCI
Kinderen die jonger zijn dan vijf jaar hebben het minste weerstand tegen ziektes. De meest voorkomende oorzaken van ziekte van het kind zijn [12]:

- Luchtwegeninfecties
- Diarree
- Huidinfecties
- Koorts

De meest voorkomende doodsoorzaken van kinderen jonger dan vijf jaar zijn complicaties voor- en tijdens de geboorte, en longontsteking. Van de vermindering in sterfte van het aantal kinderen in het jaar 2013 ten opzichte van het jaar 2010, was de helft te danken aan de vermindering in het aantal gevallen longontsteking, diarree en mazelen [13].

IMCI (Integrated Management of Childhood Illness) is een strategie die is ontwikkeld door de World Health Organization om op systematische wijze de gezondheid van kinderen jonger dan vijf jaar te verbeteren en kindersterfte tegen te gaan. Het adresseert bekende doodsoorzaken in derdewereldlanden zoals malaria, longontsteking, diarree, mazelen, HIV en ondervoeding [14]. Het ”IMCI Chart Booklet” is een boek dat wordt gebruikt voor het stellen van diagnoses voor vaak voorkomende ziektes en doodsoorzaken.

4

Ontwerpproces Hardware

In dit hoofdstuk wordt beschreven hoe de gebruikte hardware is gekozen en geïmplementeerd.

4.1. Microcontroller

De eisen die gesteld zijn aan de microcontroller zijn:

- De microcontroller moet in staat zijn het hele programma op te slaan en uit te voeren.
- Hij moet een laag stroomgebruik hebben, zowel in actieve modus als slaap modus. De bedoeling is dat de CPU zo snel mogelijk in slaap modus gaat na zijn taken, om de meeste tijd in een staat van weinig stroomgebruik te zijn.
- Hij moet in staat zijn een LCD scherm aan te sturen. Uitgaande van de hieronder besproken keuze van de schermen, moet er dus een SPI of een I^2C interface aanwezig zijn.
- Ook is een I^2C interface nodig voor de RTC (Real Time Clock) en de externe EEPROM.
- Hij moet genoeg I/O pinnen bevatten, waarvan tenminste drie analoge ingangen.
- Hij mag niet dermate veel kosten dat de prijs van het systeem boven de 6 dollar uitkomt.

De volgende opties zijn overwogen:

- De ATmega familie van Atmel, in het bijzonder de ATmega328P
- De MSP430 familie van Texas Instruments
- De PIC familie

De PIC familie heeft een relatief grote community maar met de werking en het programmeren hiervan is geen bestaande ervaring binnen deze projectgroep. Verder is er speciale hardware benodigd voor het flashen van de microcontroller. Hierdoor valt deze optie eigenlijk direct af en wordt hij verder niet meer besproken.

De ATmega familie geniet grote bekendheid door het gebruik in het Arduino computerplatform. De ATmega328P is een erg bekende en vaak gebruikte microcontroller. De ATmega 2560 is een grotere versie van de 328 en is daarom ook opgenomen in de te bekijken mogelijkheden.

De MSP430 familie is een energiezuinige, low-cost microprocessor familie van Texas Instruments. De MSP430F2274 lijkt het meest op de ATmega328P qua beschikbare hoeveelheid FLASH geheugen, ondersteuning van communicatieprotocollen en prijs.

Tabel 4.1: Tabel verschillen tussen diverse microcontrollers [15], [16], [17].

Microcontroller	Max Freq. (MHz)	FLASH (kB)	EEPROM (kB)	SRAM (kB)	GPIO (digital/ analog in)	I^2C	SPI	Current Usage Active @ 3.3V @8MHz (mA)	Current Usage Deepest Sleep @3.3V @8MHz (μA)	Operation Voltage @8MHz (V)	Cost (USD)
ATmega328P	20	32	1	2	23 (15/8)	1	2	3,0	0,25	2,40 - 5,5	1,12 [18]
ATmega 2560	16	256	4	8	86 (72/14)	1	5	6,8	0,25	4,5 - 5,5	3,80 [19]
MSP430F2274	16	32	0	1	32 (20/12)	1	1	2,16	0,1	2,25 - 3,6	1,95 [20]

Zoals in tabel 4.1 te zien is zijn de verschillen tussen deze drie microcontrollers groot. Door de zeer hoge kosten van de ATmega 2560 valt deze eigenlijk al direct af. Daarnaast kent deze microcontroller ook een relatief hoog stroomgebruik.

De ATmega328P heeft door zijn zeer grote oplage (veel gebruikt in de Arduino's) een zeer lage kostprijs. Het stroomgebruik is gemiddeld en er is meer SRAM beschikbaar dan bij de MSP, iets wat later in het ontwerpproces erg waardevol blijkt te zijn. De hoeveelheid FLASH, 32 kB, is genoeg om grote programma's in te verwerken. Daarnaast heeft deze microcontroller ook een kleine hoeveelheid EEPROM tot zijn beschikking. Voor beide ATmega's geldt dat ze dankzij de Arduino een zeer grote community hebben, dus kunnen er genoeg informatiebronnen en libraries gevonden worden om het gebruik te vergemakkelijken. Beide kunnen ook geflashed worden met behulp van de Arduino IDE, iets waar enkele leden van het project al ervaring mee hebben. De taal die gebruikt wordt in Arduino IDE is gebasseerd op C/C++[21].

De MSP430 familie kent deze voordelen niet. Met een omweg zou eventueel gebruik gemaakt kunnen worden van de Arduino IDE [22], maar de community is veel kleiner dan bij de ATmega's. Wanneer het missen van een EEPROM en het kleinere SRAM dan ook nog worden meegenomen in de afweging, weegt het voordeel van een iets lager stroomgebruik hier niet tegenop.

Er is gekozen voor de ATmega328P omdat, zoals hierboven reeds vermeld, de kosten hiervan het laagst zijn, hij twee keer zo veel geheugen heeft als de MSP430, er een grote community voor aanwezig is en er reeds veel kennis over beschikbaar is binnen dit project. Zeker het laatstgenoemde heeft zwaar gewogen in het besluit. De keuze voor deze microcontroller heeft als gevolg dat er een spanning geleverd moet worden die tussen de 2,4V en 5,5V ligt.

4.2. Bediening

Voor interactiviteit met de gebruiker heeft BabyShell een zestal knoppen ter beschikking. Één daarvan is een reset knop die met een naald of paperclip ingedrukt kan worden. De overige vijf knoppen (up, down, back, select en home) zijn voor de besturing van de menu's en om antwoord te kunnen geven op de gestelde vragen.

De eisen die gesteld worden aan de knoppen zijn:

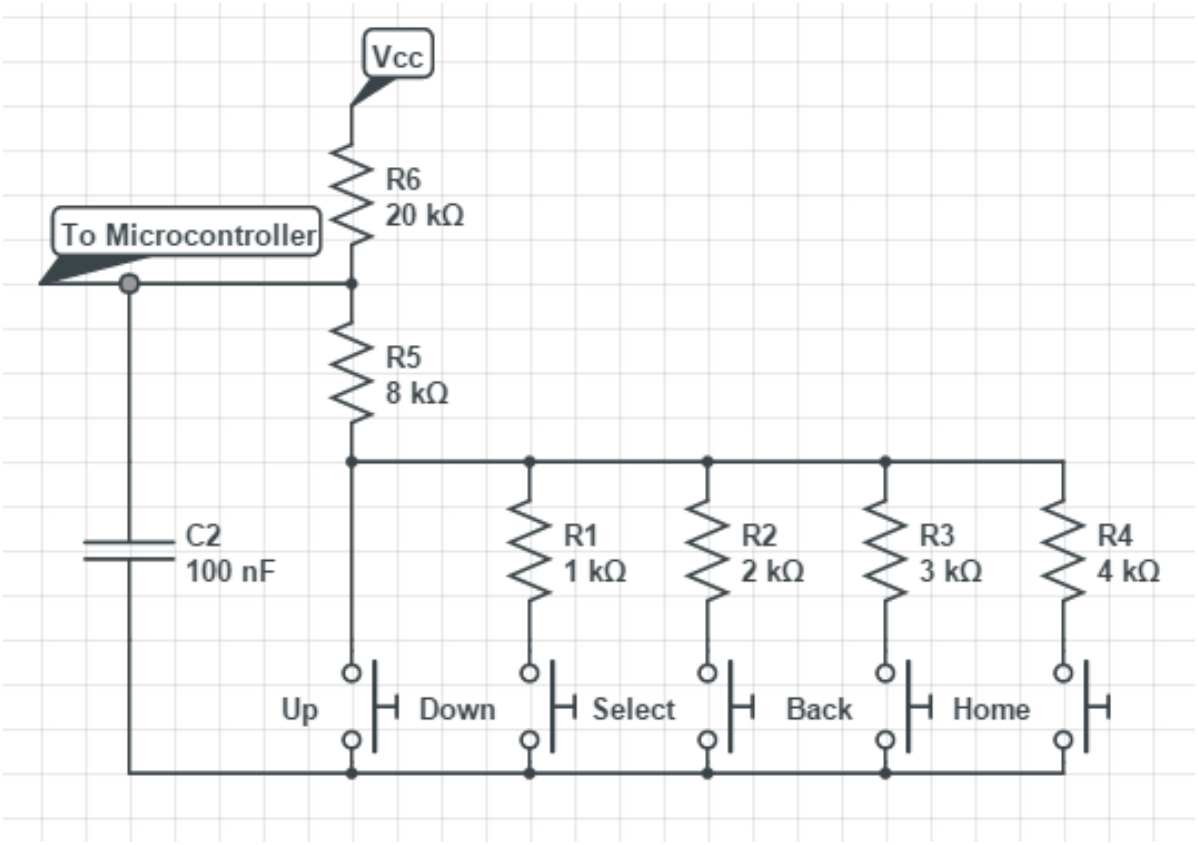
- Het gebruik van weinig I/O poorten op de microcontroller
- Weinig stroomgebruik tijdens indrukken en liefst geen stroomgebruik wanneer er niets ingedrukt wordt
- Goedkoop

De volgende mogelijkheden zijn onderzocht:

- Momentary Tactile Push Button Switch 6*6*5mm, deze zijn zeer goedkoop (1 cent per 6 stuks) en hebben een makkelijke uitlezing
- Silicon Rubber keypad, deze zijn een stuk duurder dan de tactile knoppen, vereisen een eilandenstructuur op de pcb en nemen veel plaats in

Er is gekozen voor de Momentary Tactile knoppen vanwege de simpele uitlezing en de zeer lage kosten. In de behuizing kan deze knop tot een meer gebruiksvriendelijkere knop worden verwerkt.

De ATmega heeft slechts 23 I/O pinnen beschikbaar. Normaliter zouden alle vijf knoppen aan één I/O interrupt pin worden verbonden. Een andere optie is het gebruik van vijf I/O niet-interrupt pinnen in combinatie met één interrupt pin. Omdat beide opties veel I/O pinnen gebruiken, is er gezocht naar een andere oplossing. Deze is gevonden door met verschillende weerstanden de knoppen met één analoge ingang pin te verbinden en gebruik te maken van een software interrupt. Hierbij wordt de ADC gebruikt om de spanningsdeling van elke knop te meten. Het schema is in figuur 4.1 te zien.



Figuur 4.1: Schema van de knoppen

Volgens bron [16] moeten de weerstanden zo gekozen worden dat elke elke knop een spanningsdeling heeft die onder de maximale V_{low} zit ($V_{low_{high}} < 0.3V_{cc}$) ligt, om een interrupt te genereren. Het is echter proefondervindelijk gebleken dat bij waardes onder $0.4V_{cc}$ de buttons ook in staat zijn interrupts te genereren (zie 8).

Daarnaast zijn de weerstanden ook zo gekozen dat bij het indrukken van twee of meerdere knoppen tegelijkertijd het spanningsniveau ver onder het spanningsniveau van de meest lage knop komt. Hierdoor kan dus gedetecteerd worden dat er meerdere knoppen tegelijk ingedrukt worden waarna er geen actie wordt ondernomen. Er is echter in een later stadium gekozen om de weerstanden op te delen in kleine weerstandswaarden en één $8K\Omega$ weerstand. Hierbij is de functie van het detecteren van twee knoppen vervallen (zie 8).

4.3. Display

Het display van BabyShell moet in staat zijn om alle teksten in een zo weinig mogelijk aantal schermen te tonen. Bij het gekozen lettertype, elke letter past binnen 7x5 pixels, moet tenminste het hele hoofdmenu op één scherm passen. Bij lange vragen is het acceptabel wanneer er tussen meerdere

deelschermen gescrold kan worden. Daarnaast dient het scherm zo goedkoop mogelijk te zijn en zo weinig mogelijk stroom te gebruiken. Ook dient het scherm in de zon leesbaar te zijn en moet de gebruikte microcontroller overweg kunnen met de aansturing van het scherm.

Concluderend betekent dat het volgende:

- Het display moet zonder backlight kunnen werken
- Aansturing geschied door I^2C of SPI
- De kosten mogen niet hoger zijn dan 2 dollar
- Het scherm moet bij gebruik niet meer dan 1 mA gebruiken
- Het heeft de voorkeur dat het scherm ook de bestaande tekst blijft tonen na het afsluiten van de datalijnen

Tabel 4.2: LCD display vergelijking [23], [24], [25]

Screen	Chip	Cost (USD)	Size	Data Connection	Readability @ normal textsize	Current Usage Standby (mA)	Voltage Range (V)
Equivalent Nokia 5110 Screen	PCD8544	1,64	48 x 84 pixels	SPI	8 x 6 pixels per character Readable without backlight	0,24	2,7 - 3,3
16x2	Hitachi HD44780	1,12	16 x 2 characters	Parallel 7 or 11 I/O pins Optional I^2C	8 x 6 pixels per character Unreadable without backlight	2,0 + 80(backlight)	2,7 - 5,5
E-Ink		8,00	132 x 64 pixels	Optional SPI	Readable without backlight	0.4	2,4 - 3,7

Zoals te zien in tabel 4.2 zijn er drie verschillende schermen vergeleken. Het onder hobbyisten bekende Nokia 5110 scherm biedt plaats voor 6 regels met elk 14 karakters bij het gekozen lettertype, heeft een lage kostprijs en de aansturing geschied over SPI. Het andere onder hobbyisten bekende scherm, 16x2 LCD, heeft als nadeel dat hij een grote hoeveelheid I/O pinnen gebruikt of een externe I^2C module nodig heeft. Daarnaast is het scherm erg breed en heeft het maar ruimte voor twee regels tekst. Ook is het scherm dik ten opzichte van het Nokia 5110 scherm en zitten er veel loze ruimtes in het zichtbare gebied. Het vereiste gebruik van backlight zorgt voor een flinke toename in het stroomgebruik, iets wat voor een energiezuinig product als BabyShell niet wenselijk is. Het E-Ink screen heeft goede specificaties maar is gewoonweg te duur.

Wanneer de opgenoemde voor- en nadelen bekeken worden, kan geconcludeerd worden dat het Nokia 5110 scherm het beste is voor dit product. Uit testen is ook gebleken dat dit scherm de tekst blijft tonen ook na het afsluiten van de datacommunicatie. Wanneer de ATmega tekst geschreven heeft naar het scherm en de CPU in Power Down modus gaat, blijft het scherm dus nog steeds die tekst tonen.

Als gevolg van deze keuze moeten de SPI poorten op de microcontroller vrijgehouden worden en moet een spanning tussen de 2,7 en 3,3 V geleverd worden.

4.4. Opslag

De ATmega heeft een drietal typen geheugen ingebouwd:

- FLASH wordt gebruikt voor het opslaan van het programma. Dit geheugen kan niet tijdens het draaien van een programma op de ATmega veranderd worden, alleen door te flashen met een computer kan hier data aangepast worden. Dit geheugen is non-volatile. De ATmega heeft hiervan 32kB beschikbaar, waarvan 30kB effectief (de rest is in gebruik door de bootloader).
- In SRAM worden de tijdelijke en globale variabelen opgeslagen. Dit geheugen is volatile. Op de ATmega is 2kB SRAM aanwezig.
- Ten slotte heeft de microprocessor 1kB aan EEPROM geheugen. Hierin kunnen variabelen opgeslagen worden die ook na een reset behouden moeten blijven. Dit is dus non-volatile geheugen. Deze zal, omdat hij zich op de microcontroller zelf bevindt, interne EEPROM genoemd worden.

BabyShell moet in staat zijn alle teksten die nodig zijn voor het menu, de diagnose vragen, de tips en de notificaties op te slaan. Door de grote hoeveelheid tekst is het niet mogelijk dit lokaal op de ATmega op te slaan. Deze heeft immers maar 32kB aan FLASH (waar tevens het programma in op wordt geslagen) en maar 1kB aan interne EEPROM. Om voor meer opslagmogelijkheid te zorgen wordt gebruik gemaakt van een externe EEPROM over de I^2C buslijn. Door I^2C te gebruiken, wat sowieso al gebruikt werd voor de RTC, wordt er bespaard op het gebruik van extra I/O pinnen en extra libraries ten opzichte van bijvoorbeeld parallelle uitlezing.

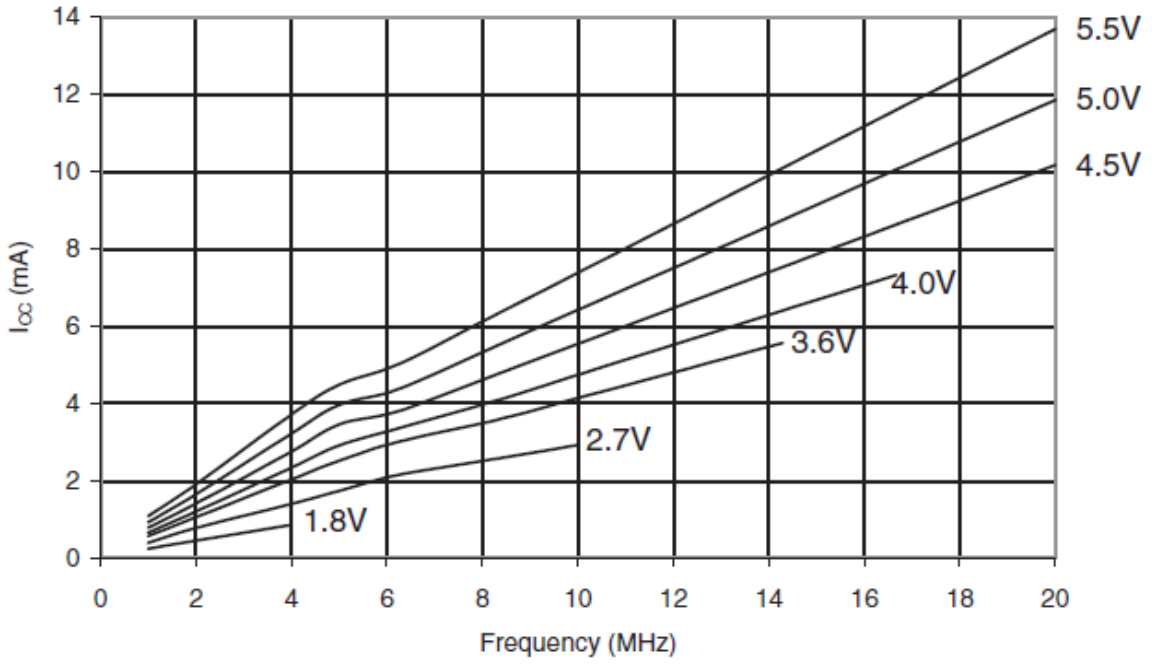
Als EEPROM is gekozen voor een Microchip 24AA256 [26]. Deze 256kbit EEPROM maakt gebruikt van de I^2C lijnen, heeft een kloksnelheid van maximaal 400 kHz, een breed spanningsbereik (1,7 - 5,5 V) en een laag stroomgebruik (max $1\mu A@3.6V$). Deze EEPROM zal in de verdere thesis "externe EEPROM" genoemd worden. In de toekomst kan de goedkopere AT24C256 van Atmel gekozen worden. Deze heeft exact dezelfde pin-out en kan als vervanger dienen [27].

De externe EEPROM gaat gebruikt worden voor het opslaan van grote teksten, zoals de zelfdiagnose-vragen, diagnoses, tips en notificaties. Voor elke tekst wordt een vaste ruimte van 128 bytes gereserveerd in de EEPROM. Dit is de lengte van de langste gebruikte zin met wat extra marge toegevoegd. Met de gebruikte EEPROM betekent dat, dat er plaats is voor 250 teksten.

Omdat in de externe EEPROM alle ruimte verdeeld is in blokken van 128 bytes en het het handigst is deze conventie voor de hele externe EEPROM aan te houden, worden kleinere teksten, zie secties 5.1.3 en 5.4.1, opgeslagen in het FLASH geheugen van de microcontroller. Variabelen die bewaard dienen te blijven tijdens een reset van BabyShell worden opgeslagen in de interne EEPROM.

4.5. Energie besparing

Om de gekozen microcontroller, de ATmega328P, minder stroom te laten gebruiken zou er in plaats van de gebruikelijke klokfrequentie van 16 MHz voor een lagere frequentie van 8 MHz gekozen kunnen worden. Het stroomgebruik zal hierdoor dalen evenals de minimale voedingsspanning. Een bijkomend extra voordeel is dat er bespaard kan worden op een extern kristal (welke nodig is bij een frequentie van 16 MHz), wanneer er gebruik gemaakt wordt van de interne 8 MHz oscillator.



Figuur 4.2: ATmega328P stroomgebruik en spanning tegen frequentie, grafiek [16]

Tabel 4.3: ATmega328P stroomgebruik en minimale spanning tegen frequentie

Frequency [MHz]	$V_{min}[V]$ [28]	$I_{active@3.0V}[mA]$ [16]	$I_{active@3.5V}[mA]$ [16]	$I_{active@4.0V}[mA]$ [16]
16	3.78	Niet Stabiel	Niet Stabiel	7.0
8	2.40	2.8	3.4	4.0

Zoals te zien in bovenstaande tabel zou de ATmega bij een frequentie van 16 MHz zowel een hogere spanning als stroom nodig hebben dan bij 8 MHz. Daar staat tegenover dat hij wel twee keer zo snel werkt als wanneer hij op 8 MHz draait. Men zou kunnen kiezen voor een andere frequentie dan de twee hierboven. Dit zijn echter de twee meest gebruikte frequenties in de Arduino community [29]. Wanneer ervoor gekozen zou worden om buiten deze twee frequenties te werken zou dat betekenen dat een groot deel van de gebruikte libraries aangepast zou moeten worden om ze te laten werken op de afwijkende frequentie. Omdat het stroomgebruik tijdens een Power Down modus (ADC en Watchdogtimer uit) kleiner dan $0.25\mu A$ [16] is, is het zaak zo snel mogelijk na een opdracht in deze modus te gaan om het totale stroomgebruik over een langere tijd te minimaliseren. Hoe dit geïmplementeerd is, kan gevonden worden in thesis van Gerard Hogenhout en Kees Hogenhout [9].

4.6. Flashen en bootloaden

De ATmega328P is alleen met een seriele datalijn te flashen wanneer de juiste bootloader op de microcontroller staat. De bootloader dient aangepast te worden om de ATmega, in plaats van op het normale 16 MHz extern crystal, te laten werken op de 8 MHz interne oscillator om de voordelen die hierboven beschreven zijn te verkrijgen.

Voordat men nieuwe software of een andere bootloader kan uploaden naar de ATmega dienen eerst wat installaties en aanpassingen uitgevoerd te worden. Een handleiding is bijgevoegd en kan gevonden worden in bijlage A.1. Hierin wordt stap voor stap beschreven hoe men de computer en Arduino software geschikt maakt voor het updaten van BabyShell. Een belangrijke aanpassing die gedaan is om de Arduino IDE geschikt te maken voor BabyShell, is de software te laten werken met een zogenaamde ”ATmega328P chip op een breadboard”. Hierdoor wordt er gebruik gemaakt van de interne oscillator op 8 MHz, en wordt de waarde van de Brown-out detectie veranderd. Dit is besproken in sectie 4.5.

Voor het schrijven van een nieuwe bootloader op de ATmega is er een bestand in de map ”mijn documenten/arduino/hardware/” geplaatst. In dit bestand staan de nieuwe ”fuses” (de instellingen van de microcontroller) die specifiek zijn voor een ATmega328P op een breadboard die gebruik maakt van een interne oscillator op 8 MHz. De gebruikte bestanden zijn afkomstig van bron [30], en enigszins aangepast om ze te laten werken op de nieuwe versie van Arduino IDE.

De gebruikte settings zijn te vinden in bijlage A.7. Wanneer er gekeken wordt naar de gebruikte ”fuse” instellingen kan met behulp van bron [31] gezien worden dat de ATmega nu inderdaad op de interne RC oscillator draait en de Brown-out detectie op $2.7V$ gezet is.

4.6.1. Programma flashen

Voordat het ”hoofd” programma op de ATmega geladen kan worden moeten eerst alle teksten in de externe EEPROM geladen worden. Dit kan men doen door het programma genaamd ”Burn_Tekst_v5” te laden op de ATmega. In dit programma staan alle teksten opgeslagen. Deze worden eerst op het FLASH geheugen van de ATmega geladen alvorens deze worden gelezen, naar het geheugen worden geschreven en naar de externe EEPROM worden verstuurd. Er is voor deze complexe methode gekozen omdat de teksten bij elkaar meer RAM geheugen zouden gebruiken dan er beschikbaar is. Er zou ook gekozen kunnen worden om de EEPROM, al dan niet vóór het solderen op de PCB, extern te flashen.

Wanneer dit proces voltooid is kunnen op de seriële datalijn de teksten gelezen worden. Wanneer er ”The flashing is complete if the text above looks fine” op de seriële datalijn verschijnt, is het programma klaar met het schrijven van de EEPROM. Nu kan het ”hoofd” programma geschreven worden op de manier zoals in de handleiding A.1 is beschreven.

5

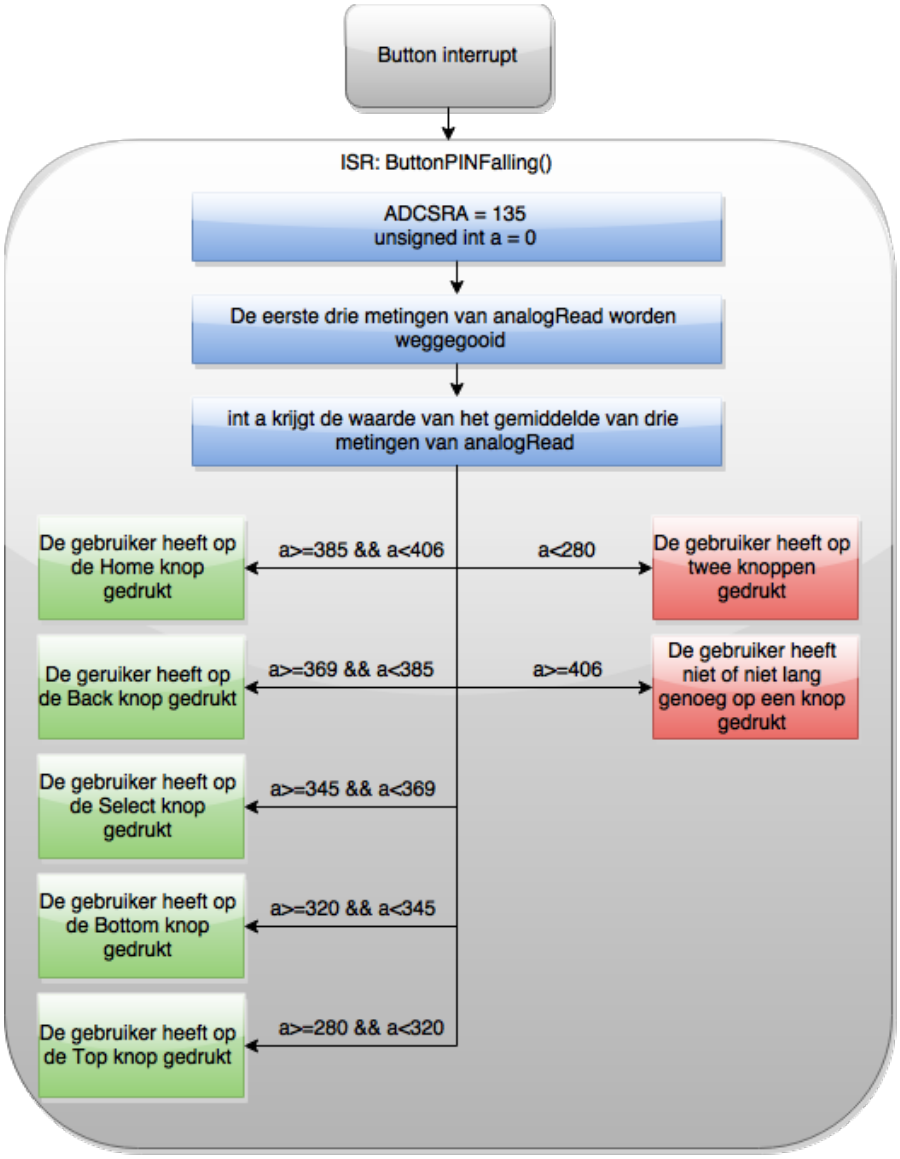
Ontwerpproces Software

In dit hoofdstuk worden verschillende delen van de software besproken. De code is geschreven met behulp van Arduino IDE 1.6.4, welke gebruik maakt van een programmeertaal die gebaseerd is op C/C++ [21].

5.1. User interface

5.1.1. Uitlezing bedieningsknoppen

Zoals besproken in sectie 4.2 geschied het uitlezen van de knoppen via een analoge ingang waar het verschil in spanningsdeling van de knop wordt gemeten. Dit uitlezen gebeurd in een ISR (interrupt service routine), deze is te vinden in bijlage A.2. Zoals te zien in figuur 5.1, wordt na het aanroepen van deze ISR eerst de variabele ”ADCSRA” op 135 gezet. Hiermee wordt de ADC weer geactiveerd nadat deze wegens energiebesparing is uitgezet in de Power Down modus. Hierna wordt de referentie van de analoge ingang op de juiste referentiewaarde (V_{cc} of $1,1V$) gezet. Omdat bekend is en uit proeven is gebleken dat de eerste paar metingen foutief zijn na zo'n verandering in referentiewaarde [32], worden de eerste drie metingen van *analogRead* niet gebruikt. Vervolgens worden er drie metingen gedaan waarbij het gemiddelde wordt genomen. Aan de hand van dit gemiddelde wordt vervolgens gekeken welke knop is ingedrukt. Wanneer bekend is welke knop ingedrukt is wordt de globale variabele behorende bij die knop hoog gemaakt en gaat het programma verder waar het gebleven was voordat het in de Power Down modus ging. Bij het indrukken van de Home knop wordt direct na de ISR het systeem gereset via een softwarereset. Hiermee komt de gebruiker dus direct terug in het hoofdmenu.



Figuur 5.1: De ISR na interrupt door knop

5.1.2. Menustructuur

Hoofdmenu

Het hoofdmenu bestaat uit een vijftal opties: ”Diagnosis”, ”Registration”, ”Clock”, ”Temperature” en ”Settings”. Hoe de menu’s geïmplementeerd zijn kan gelezen worden in sectie 5.1.2. De optie Diagnosis wordt gebruikt om de gebruiker een diagnose te laten stellen. Registration verwijst naar het Registration menu, waar de gebruiker een zwangerschap of kind kan toevoegen en verwijderen, en een overzicht kan bekijken van de vaccinaties van een kind. Clock laat de tijd en datum zien, Temperature de temperatuur die op de temperatuursensor wordt afgelezen. Settings verwijst naar het Settings menu waar de gebruiker de tijd en datum kan instellen en contactinformatie en het versienummer kan bekijken.

Registration menu

In het submenu Registration zijn wederom een vijftal opties te kiezen: ”Add newborn”, ”Remove child”, ”New preg”, ”Remove preg” en ”Vaccinations”.

Add newborn en New preg

Wanneer de optie Add newborn of New preg wordt geselecteerd, wordt de functie *addNewDate(boolean)* aangeroepen. De boolean die als argument wordt meegegeven, geeft aan of de functie is aangeroepen omdat Add newborn of New preg is geselecteerd. *AddNewDate(boolean)* is een functie om de gebruiker een nieuwe datum in te laten voeren, zij het voor het invoeren van een geboortedatum van een kind of de datum dat een vrouw zwanger is geworden. In deze functie worden vier bytes aangemaakt:

- byte selectCounter
- byte selectDay
- byte selectMonth
- byte selectYear

Op het scherm wordt met behulp van de functie *printNewDate(byte, byte, byte, byte, boolean)* de standaard datum ”00/00/2000” (dag/maand/jaar) getoond. Door middel van de byte *selectCounter* wordt bijgehouden welke variabele de gebruiker op dit moment aan het veranderen is. Op het scherm is dit getal in geïnverteerde kleuren te zien. Deze byte wordt geïnitialiseerd op ’0’, wat inhoudt dat het eerste cijfer van de datum ”00/00/2000” wordt veranderd. Voor het veranderen van de datum kan *selectCounter* zes waardes aannemen:

- ’0’, byte *selectDay* wordt met ’10’ verhoogd of verlaagd
- ’1’, byte *selectDay* wordt met ’1’ verhoogd of verlaagd
- ’2’, byte *selectMonth* wordt met ’10’ verhoogd of verlaagd
- ’3’, byte *selectMonth* wordt met ’1’ verhoogd of verlaagd
- ’4’, byte *selectYear* wordt met ’10’verhoogd of verlaagd
- ’5’, byte *selectYear* wordt met ’1’ verhoogd of verlaagd

Het verhogen of verlagen van een getal gebeurt wanneer de gebruiker respectievelijk op de up of down knop drukt. Aan de hand van de waarde van *selectCounter* zal dan een bepaalde byte worden veranderd. Telkens wanneer dit gebeurt, wordt de functie *printNewDate(byte, byte, byte, byte, boolean)* aangeroepen om de nieuwe datum op het scherm te tonen. De gebruiker kan byte *selectDay* niet verder verhogen dan 31, byte *selectMonth* niet verder dan 12 en *selectYear* niet verder dan 99. Wanneer desondanks een verkeerde datum (zoals 31 februari) in wordt gevoerd, wordt dit automatisch door het programma gecorrigeerd. Door te drukken op de select knop wordt de waarde van *selectCounter* met één verhoogd en kan de gebruiker een ander getal veranderen. Door middel van het drukken van de back knop kan de waarde van *selectCounter* met één worden verlaagd. Wanneer de waarde van *selectCounter* ’0’ is en de gebruiker drukt op back, zal terug worden gegaan naar het hoofdmenu. Wordt de waarde van selectCounter meer dan ’5’, dan betekent dat dat de gebruiker alle getallen heeft kunnen veranderen en vervolgens op de select knop heeft gedrukt.

In het geval van Add newborn wordt op dat moment gekeken of er een Child object is dat nog niet in gebruik is (boolean *usedChild*==1, zie 5.2.2). In dat geval wordt de door de gebruiker ingevoerde datum toegekend aan de variabelen van het Child object die de geboortedatum van het kind beschrijven en boolean *usedChild* naar ’1’ veranderd. Ook wordt de functie vacNotification() aangeroepen om te kijken of het kind al oud genoeg is om bepaalde vaccinaties te ontvangen. Ten slotte wordt een alarm ingesteld die zorgt dat BabyShell één keer per maand wakker wordt om te kijken of er notificaties aan de gebruiker moeten worden getoond (zie 5.2.2). Dit is alleen het geval als niet in het apparaat is ingevoerd dat de gebruiker zwanger is. Het alarm staat op dat moment al ingesteld op één keer per week en dat blijft dan onveranderd.

In het geval van New preg wordt op het moment dat de datum door de gebruiker is ingevoerd, deze toegekend aan de variabelen van het Mother object die de datum van het begin van de zwangerschap beschrijven. Vervolgens wordt gekeken of de moeder al lang genoeg zwanger is om bepaalde tips te

ontvangen (zie 5.2.2) en het alarm van de RTC ingesteld op één keer per week. Ten slotte wordt na zowel ”Add newborn” als ”New preg” een tekst op het scherm getoond die de gebruiker eraan herinnert dat de datum op het apparaat goed moet zijn ingesteld.

Remove child en Remove preg

De opties Remove child en Remove preg worden gebruikt voor het resetten van alle variabelen van een bepaald Child of Mother object. Wanneer door de gebruiker de optie Remove child wordt gekozen, wordt de functie *removeChild()* aangeroepen. Deze toont met behulp van de functie *printDateList(byte)* een lijst met de geboortedata die opgeslagen staan in de Child objecten. Wanneer een Child object niet in gebruik is (*usedChild*==0), wordt de tekst ”unused” op het term getoond. Zo worden op het scherm ten alle tijden vijf opties getoond (het maximale aantal kinderen dat in het apparaat kan worden ingevoerd) waaruit de gebruiker kan kiezen. De parameter byte geeft aan welke datum de gebruiker op dat moment heeft geselecteerd. Deze wordt in geïnverteerde kleuren op het scherm weergegeven. Door middel van de up en down knoppen kan de gebruiker de verschillende data selecteren. Wanneer op de select knop wordt gedrukt worden alle variabelen van het Child object met de geselecteerde geboortedatum met behulp van de functie *resetValues()* op ’0’ gezet. Bij Remove preg wordt alleen de datum van het begin van de zwangerschap van de moeder getoond. Wanneer op de select knop wordt gedrukt, worden met behulp van de functie *resetValues()* wederom alle variabelen op ’0’ gezet. Tevens wordt er aan het eind van beide functies gekeken of het alarm van de RTC moet worden uitgeschakeld, wat het geval is indien op dat moment geen enkele zwangerschap en geen enkel kind door de gebruiker is ingevoerd.

Vaccinations

Als de optie Vaccinations wordt geselecteerd, wordt op dezelfde wijze als bij Remove child een lijst met de geboortedata van alle kinderen getoond. Wanneer de gebruiker een geboortedatum selecteert, wordt van het Child object met die geboortedatum de functie *printVaccinations()* aangeroepen. In combinatie met de functie *printScreenVaccinations(byte, byte)* wordt hiermee op het scherm getoond welke vaccinaties het kind heeft gehad en welke niet, en kan de gebruiker door de verschillende vaccinaties heen scrollen. Omdat de kinderen op jonge leeftijd een groot aantal vaccinaties moeten krijgen en veel vaccinaties op hetzelfde moment moeten worden gegeven [7], is gekozen om deze te groeperen per maand waarin ze moeten worden gegeven en dit zo op het scherm weer te geven. Dit wordt gedaan door een bepaalde maand met een nummer, welke staat voor het aantal maanden dat het kind oud moet zijn om een aantal vaccinaties te krijgen, op het scherm te tonen met daarachter een ”yes” of ”no”. Om te weten of er een ”yes” of een ”no” op het scherm moet worden getoond, wordt er gebruik gemaakt van de functie *readBoolean(byte)* van de klasse Child om te kijken of het desbetreffende kind de set vaccinaties al heeft gehad (zie 5.2.2). De gebruiker kan door op de up en down knoppen te drukken de verschillende maanden selecteren. Wanneer hij hij op de select knop drukt, zal de functie *printSingleVac(byte selectCounter)* worden aangeroepen. Met byte *selectCounter* wordt gekeken op welke vaccinatiemaand is gedrukt, om vervolgens de gebruiker te vragen of het kind de vaccinaties (deze woorden in de vraag specifiek benoemd, bijvoorbeeld hepatitis B) van die maand al heeft gehad door de functie *printScreen* van class vacQuestion aan te roepen (zie 5.2.2). Wanneer de vraag is beantwoord door op ”yes” of ”no” te drukken, wordt teruggegaan naar het schema van de vaccinatiemaanden, welke aan de hand van het gekozen antwoord direct is aangepast. Wanneer de gebruiker bij het krijgen van een notificatie (zie 5.2.2) per ongeluk een verkeerd antwoord heeft geselecteerd, kan dat in dit schema dus worden hersteld.

Settings menu

In het Settings menu kan de gebruiker uit vier opties kiezen: ”Time”, ”Date”, ”Contact” en ”Version”.

Bij de optie Time wordt de functie *changeTime()* aangeroepen. Hier kan de gebruiker op vergelijkbare wijze als beschreven bij Add newborn de tijd veranderen, alleen nu worden de bytes *selectMinute* en *selectHour* veranderd. Wanneer de gebruiker voor de vierde keer op de select knop heeft gedrukt, en alle cijfers die op het scherm worden getoond heeft kunnen veranderen, wordt de functie *setDS3231time(byte, byte, byte, byte, byte, byte, byte)* (zie thesis [9]) aangeroepen, waarmee de huidige tijd op het apparaat wordt veranderd. Bij de optie Date wordt de gebruiker op dezelfde wijze als bij Add newborn in staat gesteld de huidige datum in te stellen, met als enige verschil dat ook de dag van de week kan worden ingesteld. Nadat de gebruiker zeven keer op de select knop heeft gedrukt om alle

variabelen te veranderen, wordt wederom de functie *setDS3231time(byte, byte, byte, byte, byte, byte, byte)* aangeroepen om de huidige datum en dag van de week te veranderen. Bij de opties Contact en Version wordt enkel respectievelijk contactinformatie en een versienummer op het scherm getoond.

Class Menu

Voor het implementeren van de menu's in het apparaat is gekozen voor het Class type dat C++ biedt. Hiervoor is gekozen omdat van elk menu hetzelfde type data moet worden opgeslagen, en met deze data dezelfde soort functies moeten worden gebruikt. Het gebruik van het Class type van C++ zorgt dat dit op een overzichtelijke manier kan worden gedaan. De keuze voor classes zorgt ervoor dat wanneer de menustructuur op een later tijdstip nog moet worden uitgebreid, dit op relatief simpele wijze kan worden gedaan door het aanmaken van extra Menu objecten. In de class worden de volgende variabelen opgeslagen:

- char* *menuOptions[6]*, 6 pointers naar de locatie van de tekst die op het scherm moet komen
- byte *numberAnswers*, het aantal ingevulde pointers
- byte *nextOption*, de waarde van het met select gekozen regelnummer

Verder bestaan in de klasse Menu de volgende functies:

- *Menu(char*, char*)*
- *Menu(char*, char*, char*)*
- *Menu(char*, char*, char*, char*)*
- *Menu(char*, char*, char*, char*, char*)*
- *Menu(char*, char*, char*, char*, char*, char*)*
- void *printScreenMenu()*
- void *printMenu()*
- void *setNextOption(byte)*

Class Menu heeft vijf verschillende constructors, zodat er Menu objecten aangemaakt kunnen worden met twee tot zes menu opties. Afhankelijk van hoeveel char arrays bij het aanmaken van het object worden meegegeven, wordt de juiste constructor aangeroepen en hierin de juiste waarde aan byte *numberAnswers* toegekend. *PrintScreenMenu()* is samen met *printMenu(byte)* de functie de ervoor zorgt dat het menu op de juiste manier op het scherm wordt getoond. Wanneer de functie *printScreenMenu* wordt aangeroepen, wordt een byte *answerSelected* aangemaakt welke wordt geïnitialiseerd op '0'. Deze byte geeft aan welke regel op het scherm de gebruiker heeft geselecteerd. Vervolgens wordt de functie *printMenu(byte)* aangeroepen om de menu opties op het scherm te tonen, waarbij de byte die als argument wordt meegegeven bepaald welke optie in geïnverteerde kleuren op het scherm wordt weergegeven om aan te tonen dat deze door de gebruiker is geselecteerd. Met de up en down knoppen kan de gebruiker door de opties heen bewegen. Wanneer op de select knop wordt gedrukt, wordt de functie *setNextOption(byte)* aangeroepen. De byte die als argument wordt meegegeven is *answerSelected* en geeft dus aan welke optie de gebruiker heeft gekozen. Met behulp van de functie wordt dit opgeslagen in de byte *nextOption*.

Het programma zal bij het opstarten van BabyShell direct na de setup de functie *mainMenu()* aanroepen. Hierin wordt het object *mainMenu1* aangemaakt, en de functie *printScreenMenu()* aangeroepen. Pas als de gebruiker op de select knop heeft gedrukt, en dus de byte *nextOption* heeft veranderd, zal met een switch statement worden gekeken naar deze byte (*mainMenu1.nextOption*) en afhankelijk van de waarde een functie worden aangeroepen. Deze functie kan dus ook een andere "menu functie" zijn, waarbij op vergelijkbare wijze verder door het programma kan worden genavigeerd.

5.1.3. Talen

Omdat BabyShell een beperkte hoeveelheid geheugen heeft is er voor gekozen om maar één taal op het apparaat te laden tijdens het flashen. Dit heeft als nadeel dat de taal niet tijdens het gebruik van het apparaat in bijvoorbeeld het Settings menu veranderd kan worden. Om het aanpassen van de taal voor de BabyShell zo gemakkelijk mogelijk te maken, is alle tekst die vertaald zou moeten worden zo centraal mogelijk geplaatst.

Alle teksten die voor de vragen, tips en notificaties benodigd zijn worden opgeslagen in de externe EEPROM. Dit wordt gedaan met de code beschreven in het bestand ”Burn_Tekst_v5” (zie 4.6). In dit bestand staan op de eerste regels alle const char[]’s, gevuld met de tekst voor de vragen, diagnoses, tips en notificaties. Dit ziet er als volgt uit:

```
const char b62[] PROGMEM = {
  "Your child seems to have pneumonia, try to go to a doctor
  or health worker"
};
```

Wanneer voor een andere versie de taal hiervan moet worden aangepast, hoeft enkel de tekst tussen de aanhalingstekens te worden veranderd. Hierbij moet wel worden uitgekeken dat de lengte van de char array de 128 niet overschrijdt. Op deze maximale lengte zijn namelijk de functies die de tekst uit de EEPROM halen en op het scherm tonen gebaseerd (zie 5.3).

De overige teksten die in een bepaalde taal op het apparaat worden weergegeven zijn:

- de menu opties
- de ”aangepaste” antwoorden op de zelfdiagnose vragen
- de feedback die de gebruiker ontvangt na het beantwoorden van de vraag of een kind een bepaalde vaccinatie al heeft ontvangen
- bepaalde woorden wanneer de gebruiker de ”Temperature”, ”Add newborn”, ”New preg”, ”Change time” of ”Change date” functies gebruikt

Ten behoeve van de werkgeheugen besparing is gekozen om al deze teksten in het FLASH geheugen op te slaan. Hiervoor is een structuur gemaakt (zie 5.4.1), welke als bijkomend voordeel heeft dat al deze teksten op één plek, het bestand ”FlashMem.ino”, staan.

Omdat alle teksten dus beschreven staan op twee centrale plekken, is het relatief makkelijk het programma te veranderen in een andere taal.

5.2. Medische software

BabyShell moet in staat zijn om twee medische toepassingen te kunnen uitvoeren: het stellen van een diagnose met behulp van het gebruik van een flowchart, en het geven van tips en notificaties op bepaalde momenten tijdens de zwangerschap van de moeder of gedurende de eerste levensjaren van een kind.

5.2.1. Zelfdiagnose

Een functie van BabyShell is het stellen van diagnoses van veelvoorkomende ziektes bij jonge kinderen in ontwikkelingslanden. Om dit mogelijk te maken is het ”IMCI chartbook” [33] van de World Health Organization op het apparaat geïmplementeerd. Met het IMCI chartbook moet de gebruiker een reeks vragen beantwoorden met betrekking tot de symptomen (bijvoorbeeld: last van ademhaling), waarna aan de hand van de antwoorden een diagnose wordt gesteld en een bijpassend advies wordt gegeven. In overeenstemming met de opdrachtgever is ervoor gekozen op deze versie van BabyShell de symptomen ”last bij ademhaling”, ”diarree” en ”last van het oor” te implementeren.

Class Question

Ook voor het implementeren van deze vragen in het apparaat is gekozen voor het Class type. Voor elke vraag werd daarom een object van de klasse ”Question” gemaakt. Ook van de diagnoses die op het scherm worden getoond na het beantwoorden van de vragen worden ”Question” objecten gemaakt. In de geschreven code kan onderscheid worden gemaakt tussen drie verschillende ”vragen” die als Question object worden opgeslagen:

- een vraag met ”ja of nee” antwoorden
- een vraag met ”aangepaste” antwoorden
- een diagnose: de uitkomst die de gebruiker te zien krijgt na het beantwoorden van de vragen

De klasse Question bevat de volgende variabelen:

- byte *address*
- byte *customAnswer*
- char* *customAnswer1*
- char* *customAnswer2*
- char* *customAnswer3*
- byte *chosenAnswer*

Hierin geeft byte *address* het adres in de EEPROM aan waar de tekst van de vraag staat. Deze structuur is zo gekozen ten behoeve van de geheugenbesparing (zie 4.4). Byte *customAnswer* geeft aan hoeveel ”aangepaste” antwoorden (bijvoorbeeld: minder dan 40 ademhalingen per minuut, tussen de 40 en de 50 ademhalingen per minuut of meer dan 50 ademhalingen per minuut) het object bevat. Deze is nul in het geval van een ”ja of nee” vraag. Ook bij de diagnoses die als Question objecten zijn opgeslagen is deze byte nul. Als de vraag aangepaste antwoorden heeft, worden deze in de drie char*’s *customAnswer(1, 2 of 3)* opgeslagen. Ten slotte wordt in byte chosenAnswer opgeslagen welk antwoord de gebruiker heeft gekozen bij het beantwoorden van de vraag.

De Question class bevat tevens de volgende functies:

- *Question (byte)*
- *Question (byte, char*, char*)*
- *Question (byte, char*, char*, char*)*
- void *setChosenAnswer(byte)*
- void *printScreen()*
- void *printAnswer()*
- void *printOutcome()*

Zoals hierboven te zien heeft class Question drie constructors. Bij het aanmaken van een Question object wordt automatisch de goede constructor aangeroepen, afhankelijk van hoeveel char*’s bij het aanmaken van het object worden meegegeven. De parameter byte staat voor het adres van de tekst van de vraag in het EEPROM. Ook wordt in de constructor een waarde toegekend aan byte *customAnswer*, afhankelijk van welke van de drie constructors is aangeroepen. De parameter char* bij de tweede en derde constructor hierboven zijn voor de aangepaste antwoorden. *SetChosenAnswer* is de functie die wordt aangeroepen op het moment dat de gebruiker antwoord geeft op een vraag. De byte die wordt meegegeven is het gekozen antwoord, waar in het geval van een ”ja of nee” vraag ’0’ staat voor ’nee’, en ’1’ voor ’ja’. In het geval van een vraag met aangepaste antwoorden staat de byte voor het volgnummer van hoe de antwoorden op het scherm getoond werden, en in het geheugen staan

opgeslagen. *PrintScreen()* (zie ook 5.3.1) is de functie die de tekst van de vraag op het scherm toont. Hierbij wordt onderscheid gemaakt tussen een vraag die op één scherm getoond kan worden (maximaal 6 regels), en een vraag die op twee schermen getoond wordt. Door middel van het gebruik van de knoppen up and down is het in dat geval mogelijk om tussen de schermen die verschillende delen van de vraag tonen te wisselen. Wanneer men zich op het scherm bevindt dat het laatste deel van de vraagtekst toont, en nogmaals op down drukt, worden de functie *printAnswer()* aangeroepen. Omdat bij de Question objecten die diagnoses voorstellen het antwoord niet getoond hoeft te worden, wordt door middel van een if statement gekeken of het adres van de betreffende vraag overeenkomt met één van de adressen waar de diagnoses zijn opgeslagen. Als dat het geval is wordt *printAnswer()* niet aangeroepen. *PrintOutcome()* wordt aangeroepen op het moment dat er een diagnose op het scherm moet worden getoond.
PrintAnswer() laat de verschillende antwoorden op de desbetreffende vraag op het scherm zien, waarbij het antwoord dat op dat moment door de gebruiker is geselecteerd met geïnverteerde kleuren wordt weergegeven. Wanneer er op de select knop wordt gedrukt, wordt de functie *setChosenAnswer(byte)* aangeroepen om het geselecteerde antwoord op te slaan en wordt vervolgens de functie afgesloten.

Navigatie door de flowchart

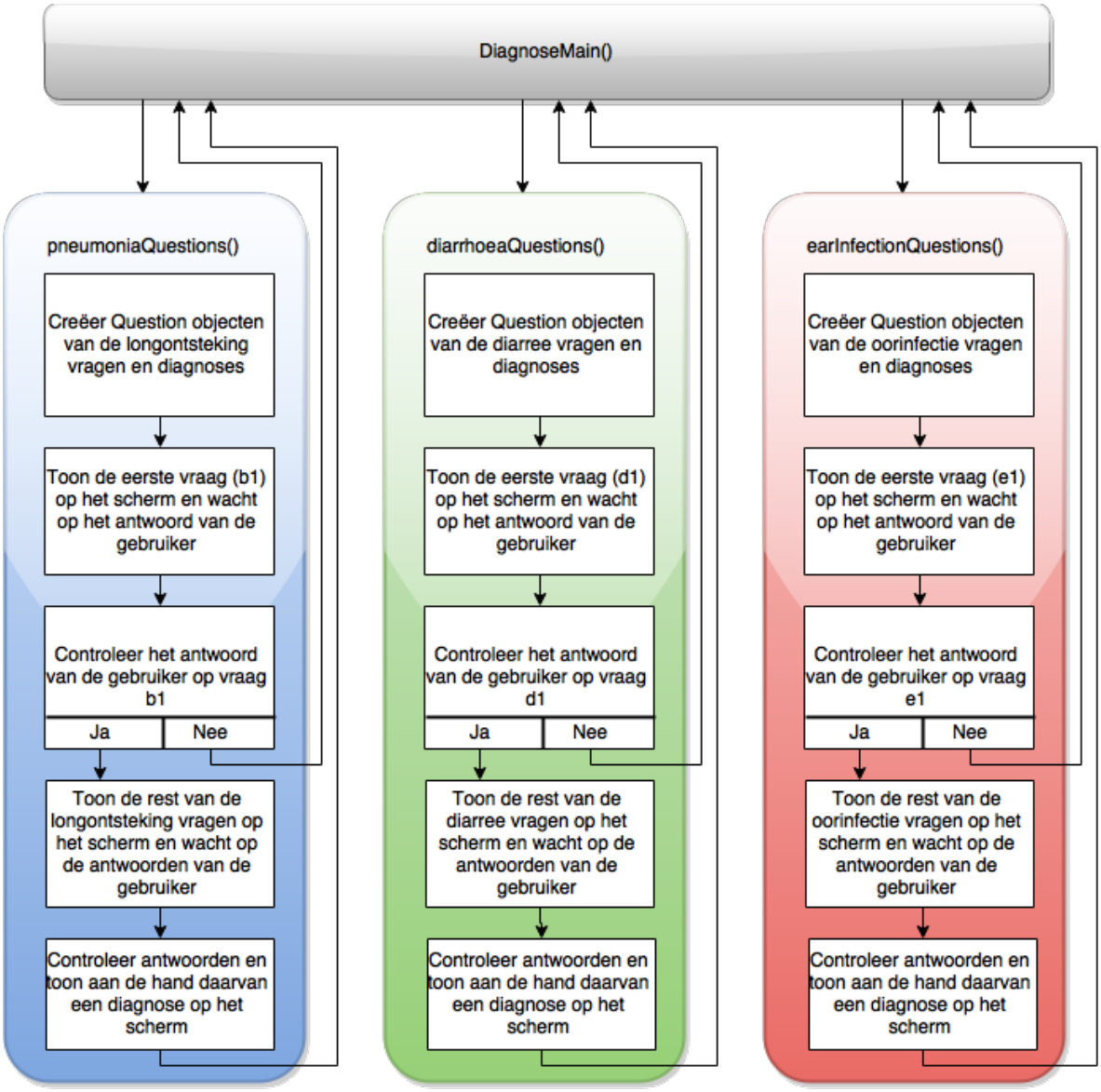
Wanneer de gebruiker in het hoofdmenu ”Diagnosis” selecteert, worden in vooraf bepaalde volgorde alle vragen gesteld om de diagnose te stellen. In het bestand ”Flowchart.ino” staat de functie *DiagnoseMain()* die dan wordt aangeroepen. Deze roept per type symptoom een functie aan (zie 5.4). Hoe deze functie te werk gaat is te zien in afbeelding 5.2. Dit is als volgt en in deze volgorde onderverdeeld:

- void *pneumoniaQuestions()*
- void *diarrhoeaQuestions()*
- void *earInfectionQuestions()*

Deze drie functies zijn qua structuur nagenoeg identiek. Eerst worden er Question objecten voor alle vragen en diagnoses betreffende het symptoom aangemaakt. Hierin wordt het adres, het aantal *customAnswers*, en indien van toepassing de invulling van die *customAnswers*, meegegeven. De eerste vraag is altijd in de vorm van of de gebruiker überhaupt last heeft van de symptomen van de betreffende functie die op het moment wordt aangeroepen. Zo wordt er bij pneumoniaQuestions() als eerst gevraagd of de gebruiker ademhalingsproblemen ervaart. Het antwoord van deze eerste vraag wordt bekeken volgens:

```
if(b1.chosenAnswer == 0)
 {
   return; //no breathing problems,
           //go to next set of questions in DiagnoseMain
 }
```

Hierin stelt b1 het Question object voor. Wanneer b1.chosenAnswer==0, heeft de gebruiker ingevuld dat er deze geen last werd ondervonden van ademhalingsproblemen en kan verder worden gegaan met de andere symptomen. Is dit niet het geval, en heeft de gebruiker aangegeven wél het symptoom te ondervinden, dan worden één voor één de vragen gesteld die bij het symptoom horen. Per vraag wordt het gegeven antwoord opgeslagen, en aan het eind van de functie wordt gekeken hoe deze vragen zijn beantwoord met *chosenAnswer*. Aan de hand van de antwoorden wordt een bepaalde diagnose op het scherm getoond. Kijken welke diagnose van toepassing is wordt gedaan door middel van if-statements.



Figuur 5.2: Werking van de functie DiagnoseMain(), welke de gebruiker door een aantal vragen leidt om een diagnose te kunnen stellen

In de functie *diarrhoeaQuestions()* wordt tevens gewerkt met het optellen van een counter bij antwoorden op bepaalde vragen, omdat sommige te stellen diagnoses afhankelijk zijn van de antwoorden op meerdere specifieke vragen. Ook is het in deze functie zo dat de gebruiker na het beantwoorden van de vragen meerdere diagnoses op het scherm te zien kan krijgen, omdat er afhankelijk van zijn gegeven antwoorden meerdere ”ziektes” vastgesteld kunnen zijn. Dit is geheel volgens bron [33].

Wanneer een diagnose voor een bepaald symptoom is gesteld, wordt teruggegaan naar *DiagnoseMain()* en wordt de functie voor het volgende symptoom aangeroepen. Dit gaat door tot de functies van alle symptomen zijn aangeroepen, en BabyShell alle toepasselijk diagnoses op het scherm heeft getoond.

5.2.2. Tip en notificatie functionaliteit

Een andere medische functie van BabyShell is het geven van tips en notificaties aan de moeder tijdens de zwangerschap, of tijdens de eerste levensjaren van een kind. Voor elke week in de zwangerschap is er een tip die de moeder moet ontvangen, en een aantal keer wanneer een kind een bepaalde leeftijd heeft bereikt moet de moeder eraan worden herinnerd dat het kind gevaccineerd moet worden. Dit

moet worden bijgehouden voor één moeder en maximaal 5 kinderen. Hiervoor zijn voor het apparaat een aantal dingen van belang:

- Het moet weten welke datum en tijd het op dit moment is
- Het moet weten of de moeder zwanger is, en zo ja, welke datum ze zwanger is geworden
- Het moet weten of de moeder kinderen heeft, en van elk kind de geboortedatum

Huidige datum en tijd

Voor het bijhouden van de datum en tijd wordt gebruik gemaakt van een RTC [9]. Bij het hoofdmenu onderdeel ”Settings” kan de gebruiker instellen welke tijd, datum, en dag van de week het op dit moment is (zie 5.1.2). De RTC wordt dan ingesteld en telt door, zodat er wordt bijgehouden welke tijd en datum het nu is. Door het aanroepen van de functie *readDS3231time()* wordt de huidige tijd en datum, opgeslagen in de globale variabelen second, minute, hour, dayofweek, dayofmonth, month en year, ververst.

Mother class

BabyShell moet ten alle tijden weten of de gebruiker van het apparaat zwanger is. Om alle data over de moeder op te slaan, is de class ”Mother” aangemaakt. Hierin staan de volgende variabelen:

- byte *dayPregnant*
- byte *monthPregnant*
- byte *yearPregnant*
- boolean *motherPregnant*

In de eerste drie bytes wordt de datum dat de moeder zwanger is geworden opgeslagen. In boolean *motherPregnant* wordt opgeslagen of de moeder zwanger is of niet. Aangezien BabyShell in staat moet zijn om één moeder te onthouden, wordt er slechts één object (mother1 genaamd) van class Mother aangemaakt. Als dit object in een functie zou worden aangemaakt, zou hij weer verwijderd worden zodra uit deze functie wordt gegaan. Omdat de data van mother1 ten allen tijden bewaard moet blijven, ook als het apparaat uit gaat, en vanuit het hele programma opvraagbaar moet zijn, is mother1 globaal gemaakt. De waardes van de data van het object worden opgeslagen in de interne EEPROM. Wanneer ze worden veranderd, wordt er opnieuw naar de EEPROM geschreven, en wanneer BabyShell is afgesloten en vervolgens opnieuw opstart, worden de waardes van de data weer in mother1 ingelezen en toegekend aan de juiste variabelen van het object. Op deze manier gaat de data van het Mother object nooit verloren. De klasse Mother bevat verder de volgende functies:

- *Mother()*
- void *resetValues()*
- void *setMotherPregnant()*
- void *setDayPregnant()*
- void *setMonthPregnant()*
- void *setYearPregnant()*

Mother() is de constructor. Deze heeft geen argumenten, maar haalt de waardes van de variabelen uit de interne EEPROM van de ATmega (zie 4.4) volgens:

```
Mother()
  {
    motherPregnant = EEPROM.read(36 + 0);
    dayPregnant = EEPROM.read(36 + 1);
    monthPregnant = EEPROM.read(36 + 2);
    yearPregnant = EEPROM.read(36 + 3);
  }
```

Dit betekent dat elke keer dat BabyShell opnieuw wordt opgestart, mother1 opnieuw wordt aangemaakt maar de oude variabelen worden ingelezen waardoor het een exacte kopie is van mother1 voor het afsluiten van het apparaat. De functie *resetValues()* zet de waarde van alle variabelen van het object op '0' en schrijft dit ook naar de interne EEPROM. Dit gebeurt alleen wanneer in het Registration menu huidige zwangerschap wordt verwijderd (zie 5.1.2). De vier *set*-functies worden aangeroepen om de variabelen van het object te veranderen, in het geval dat er in het Registration menu een nieuwe zwangerschap wordt ingevoerd als de gebruiker voor de optie "New preg" heeft gekozen.

Dit alles zorgt ervoor dat ten alle tijden overal kan worden gekeken of en wanneer de moeder zwanger is geworden. Door dit te vergelijken met de huidige tijd gegeven door de RTC (zie [9]) kan worden gekeken hoe lang de moeder al zwanger is en welke tip op dat moment dus moet worden gegeven.

Child class

Ook moet BabyShell ten alle tijden weten of de gebruiker kinderen heeft, hoe oud deze zijn en welke vaccinaties ze al hebben gehad. Hiervoor is wederom een class, genaamd "Child" aangemaakt. In class Child worden van elk object de volgende variabelen opgeslagen:

- byte *dayBirth*
- byte *monthBirth*
- byte *yearBirth*
- byte *childNumber*
- int *_booleanSaver*

Net zoals in de Mother class worden de variabelen geschreven naar de interne EEPROM om te voorkomen dat deze verloren gaan bij het uitvallen van het apparaat. Ook wordt net als bij de Mother class in de drie bytes, *dayBirth*, *monthBirth* en *yearBirth* opgeslagen wanneer het kind is geboren. Omdat er in de interne EEPROM ruimte is gemaakt voor de variabelen van vijf kinderen (het maximale aantal kinderen dat het apparaat op kan slaan) wordt in byte*childNumber* opgeslagen welk van de vijf kinderen het object betreft. De int _booleanSaver is aangemaakt ten behoeve van de geheugenbesparing, immers hebben booleans dezelfde grote als bytes [34]. Door gebruik te maken van de functies bitWrite en bitRead kunnen er tot 8 booleans in één byte en tot 16 booleans in één integer opgeslagen worden. Dit heeft tot gevolg dat het globale geheugengebruik van 5 kinderen met elk 12 booleans, afneemt met 50 bytes. De plaatsing en benaming van de booleans in de int is als volgt:

```
//  in Booleansaver the numbering is:
 //  0 boolean month0;
 //  1 boolean month1;
 //  2 boolean month2;
 //  3 boolean month4;
 //  4 boolean month6;
 //  5 boolean month12;
 //  6 boolean month15;
 //  7 boolean month18;
 //  8 boolean month30;
 //  9 boolean month42;
 //  10 boolean month48;
 //  11 boolean usedChild;
```

De booleans 0 tot en met 10 geven aan of het kind bepaalde vaccinaties wel of niet heeft gehad. Volgens bron [7] zijn in de eerste vijf levensjaren van een kind elf momenten dat een kind bepaalde vaccinaties moet ontvangen. De booleans zijn allen vernoemd naar een bepaalde maand. Het nummer dat achter de maand vermeld staat, staat voor hoe oud het kind moet zijn (in maanden) om die vaccinatieronde te ontvangen. Om dezelfde reden als bij objecten van de Mother class worden objecten van de Child class globaal gemaakt. De waardes van de variabelen staan bij het aanmaken van

de objecten allen op '0'. Pas als in het Registration menu de optie "Add newborn" wordt geselecteerd, en de datum van de geboorte van het kind wordt ingevuld, worden de waardes van deze variabelen veranderd. Op dat moment wordt ook de waarde van boolean 11, de boolean die aangeeft of het het Child object door de gebruiker wordt gebruikt, op '1' gezet. Hierdoor kan het programma van elk Child object nagaan of het in gebruik is. De Child klasse maakt gebruik van de volgende functies:

- *Child()*
- void *resetValues()*
- void *downloadChildValuesFromEeprom(byte)*
- void *setDayBirth()*
- void *setMonthBirth()*
- void *setYearBirth()*
- void *setVacMonth(byte, boolean)*
- void *setUsedChild()*
- boolean *readBoolean(byte)*
- void *printVaccinations()*
- void *printScreenVaccinations(byte, byte)*
- void *printSingleVac(byte)*

De constructor *Child()* wordt aangeroepen op het moment dat er een nieuw Child object wordt aangemaakt. Hierbij worden de waardes van alle variabelen op 0 gezet. De functie *resetValues()* zet de waarde van alle variabelen op 0, deze functie wordt aangeroepen als in het Registration menu de optie "Remove preg" wordt geselecteerd. De functie *downloadChildValuesFromEeprom(byte)* wordt aangeroepen in de setup, bij het opstarten van het apparaat. Deze zorgt dat de waarden van de Child objecten die in de interne EEPROM staan opgeslagen, bij het opstarten van het apparaat weer aan de variabelen van de Child objecten worden toegekend. Dit zorgt ervoor dat de Child objecten na het uitvallen van BabyShell precies hetzelfde zijn als voor het uitvallen. De byte die als argument wordt meegegeven geeft aan van welk Child object de variabelen worden ingelezen. *SetDayBirth()*, *setMonthBirth()* en *setYearBirth()* worden aangeroepen wanneer in het Registration menu door de gebruiker de optie "Add newborn" wordt geselecteerd, en een geboortedatum wordt ingevuld. Op dat moment wordt een waarde gegeven aan de bytes *dayBirth*, *monthBirth* en *yearBirth*. Tevens wordt met de functie *setUsedChild()* de boolean *usedChild* in _booleanSaver op '1' gezet, om aan te geven dat het object door de gebruiker in gebruik is genomen.

Een deel van het geven van de herinneringen aan de gebruiker houdt in dat de gebruiker moet aangegeven of een kind bepaalde vaccinaties al heeft gehad. Wanneer de gebruiker bij deze vraag de optie 'yes' selecteert, wordt de functie *setVacMonth(byte, boolean)* aangeroepen. Deze veranderd een boolean in _booleanSaver. De byte die als argument van de functie wordt meegegeven bepaalt welke boolean van de _booleanSaver wordt veranderd. De boolean die als argument wordt meegegeven bepaalt welke nieuwe waarde deze boolean krijgt. Vervolgens kan de gebruiker door in het Registration menu de optie "Vaccinations" te kiezen, van elk Child object dat in gebruik is (*usedChild*==1) zien welke vaccinaties het kind al heeft gehad en welke niet. Dit wordt gedaan door van het Child object de functie *printVaccinations()* aan te roepen, welke een lijst op het scherm toont van alle mogelijke vaccinaties die het kind kan krijgen, met daarachter 'yes' of 'no' naar gelang door de gebruiker is ingevuld dat het kind de vaccinatie heeft gehad. Met behulp van de functies *printScreenVaccinations(byte, byte)* en *printSingleVac(byte)* kan de gebruiker in deze lijst van elke vaccinatiemaand het gekozen antwoord aanpassen (zie 5.1.2). Om te kijken of het kind een bepaalde vaccinatie al heeft gehad, wordt gebruik gemaakt van de functie *readBoolean(byte)*. Hier kan elke afzonderlijke boolean van int *_booleanSaver* worden uitgelezen, waarbij de byte aangeeft welke boolean moet worden uitgelezen.

Tips en notificaties

De hierboven beschreven manier maakt het voor het apparaat ten alle tijden mogelijk te weten wat de huidige datum en tijd is, of de moeder zwanger is, welke datum ze dit is geworden, hoeveel kinderen ze heeft, en wat hun geboortedatum is. Dit kan worden gebruikt om uit te rekenen hoe oud een kind is of hoe lang een moeder al zwanger is. Op basis hiervan kan het apparaat een tijdsgebonden herinnering geven, waarbij het onderscheid is gemaakt tussen tips en notificaties.

Tips

Tips worden getoond tijdens de zwangerschap van de moeder. Voor elke week van zwangerschap waar de moeder zich in bevindt, krijgt zij een nieuwe tip op het scherm te zien. Deze tips volgen de kalender van bron [6]. De gebruiker hoeft hierbij niet aan te geven of ze wel of niet rekening heeft gehouden met de tip, maar simpelweg op de select knop te drukken om verder te gaan. Voor de tips is een class aangemaakt: TipClass. In TipClass worden de volgende variabelen opgeslagen:

- byte *address*
- byte *weekNumber*

Omdat de tips bestaan uit grote hoeveelheden tekst, worden deze niet op de FLASH of interne EEPROM opgeslagen, maar op de externe EEPROM. De byte *address* geeft aan op welke plek in de externe EEPROM de tekst van het desbetreffende TipClass object staat opgeslagen. Byte *weekNumber* geeft aan in welke week van de zwangerschap de tip moet worden gegeven. TipClass heeft de voglende functies:

- void *set(byte, byte)*
- void *printScreen()*

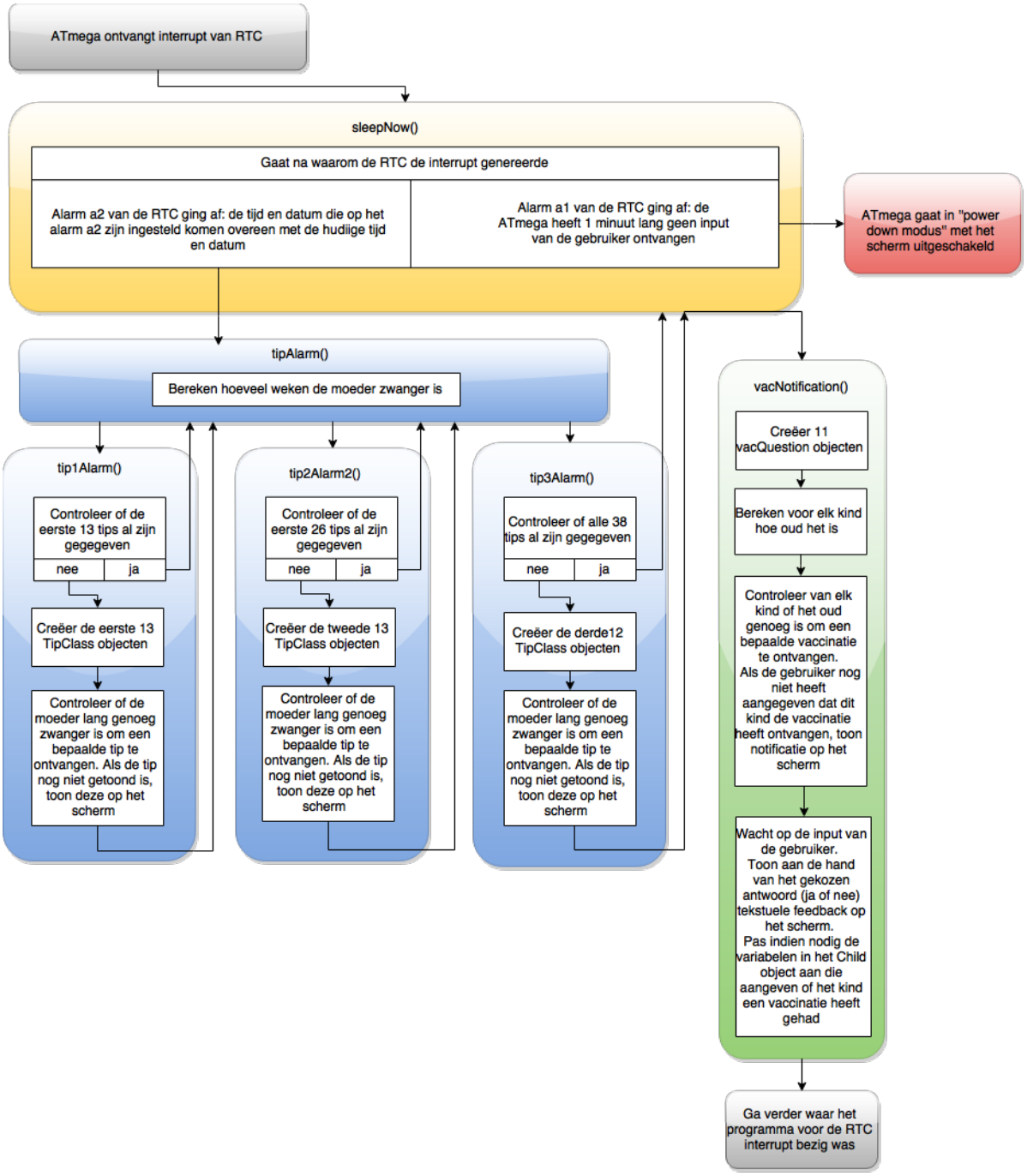
Bij het aanmaken van TipClass objecten wordt de default constructor gebruikt. Omdat het met Arduino IDE niet lukte een array van objecten aan te maken met een custom constructor, waarmee de waardes van de variabelen bij het aanmaken meteen al aan het object worden toegekend, is gebruik gemaakt van de *set(byte, byte)* functie. De eerste byte die als argument bij deze functie wordt meegegeven staat voor de waarde die aan byte *address* wordt toegekend, de tweede voor de waarde die aan byte *weekNumber* wordt toegekend. Op deze manier kan een array van TipClass objecten worden aangemaakt, en direct daarna met de set functie waardes aan de variabelen van elk object worden toegekend. Het voordeel hiervan is dat met een array van TipClass objecten alle objecten een soort volgnummer hebben. Zo kunnen ze later in de functie *notificationAlarm* (zie verderop in deze paragraaf) allemaal op gemakkelijke wijze in een for-loop worden langsgegaan. Met de functie *printScreen()* wordt de tekst van het object die in de externe EEPROM wordt opgeslagen, op het scherm getoond. Pas wanneer door de gebruiker op de select knop wordt gedrukt, wordt deze functie verlaten. Dit is een manier om te controleren of de gebruiker de tip wel heeft gelezen. Voor een gedetailleerde werking van de functie *printScreen()* zie sectie 5.3.1.

TipAlarm

De functie *tipAlarm()* is de functie die kijkt of er op het huidige moment in de tijd een tip moet worden gegeven. Ten behoeven van de werkgeheugen besparing wordt deze functie in feite opgedeeld in drie afzonderlijke functies: *tip1Alarm(int)*, *tip2Alarm(int)* en *tip3Alarm(int)*, welke één voor één worden aangeroepen in *tipAlarm()*. Daarvóór wordt eerst met de functie *readDS3231time()* de huidige tijd aan de globale tijd- en datumvariabelen toegekend zoals beschreven in thesis van Gerard Hogenhout en Kees Hogenhout [9]. Door deze te vergelijken met de bytes *dayPregnant*, *monthPregnant* en *yearPregnant* van het Mother object mother1, wordt uitgerekend hoe lang de moeder al zwanger is. Dit wordt opgeslagen in int *weeksPregnant*, welke vervolgens als argument wordt meegegeven aan de drie tipAlarm(int) functies.

In totaal zijn er 38 TipClass objecten, één voor elke week van de zwangerschap, en zijn de teksten van 38 verschillende tips dus afzonderlijk opgeslagen in de externe EEPROM. In ”deelfuncties” *tip(1,2 of 3)Alarm(int)* wordt door middel van byte *tipCounter* gecontroleerd of nog niet alle tips in de functie op het scherm zijn getoond. Als dit wel het geval is wordt meteen geretourneerd, en verder gegaan

met de volgende deelfunctie. Is dit niet het geval dan wordt een deel van deze 38 TipClass objecten aangemaakt (dertien of twaalf). In een for-loop wordt vervolgens bij elk TipClass object gekeken of de int *weeksPregnant* groter dan of gelijk is aan de TipClass byte *weekNumber*. Dit is namelijk alleen het geval als de moeder al lang genoeg zwanger is om een bepaalde tip te ontvangen. De functie zal in dat geval de *printScreen()* functie van het desbetreffende TipClass object aanroepen, welke de tip op het scherm zal tonen. Pas wanneer de gebruiker op de select knop drukt, verdwijnt de tip weer van het scherm. Tevens wordt na het indrukken van deze knop een globale variabele, byte *tipCounter*, geïncrementeerd. Met deze byte wordt bijgehouden hoeveel tips er al zijn gegeven, en in de hierboven beschreven for-loop wordt hiermee gedetecteerd als een tip al is gegeven, waardoor deze niet voor een tweede keer op het scherm zal verschijnen. De waarde van deze byte wordt ook geschreven naar de interne EEPROM wanneer deze wordt veranderd, en gedownload vanuit de interne EEPROM bij het opstarten van het apparaat, zodat deze bij het uitvallen van het apparaat bewaard blijft.



Figuur 5.3: RTC interrupt

De *tipAlarm()* functie zal als eerst worden opgeroepen wanneer er in het Registration menu de optie ”New preg” wordt geselecteerd, en de gebruiker de datum van de zwangerschap invult. Het kan zijn dat de gebruiker dit pas doet wanneer de moeder al een aantal weken zwanger is. In dat geval zullen een aantal tips achter elkaar op het scherm worden getoond, net zolang tot de byte *weekNumber* van de eerstvolgende te geven tip groter is dan het aantal weken dat de moeder zwanger is.

Op dat moment wordt op de RTC een alarm voor de eerstvolgende maandag om twaalf uur ingesteld door middel van de functie set_a2(flags) (zie [9]). Dit zorgt ervoor dat de RTC op dat moment een interrupt zal geven. Bij het krijgen van een interrupt kijkt de ATmega of deze werd veroorzaakt door het indrukken van een knop (zie 5.1.1) of door de RTC (zie figuur 5.3). Als deze werd veroorzaakt door de RTC, wordt vervolgens gekeken of de RTC deze heeft gegenereerd omdat er een minuut geen

input van de gebruiker is geweest, of omdat het alarm a2 is afgegaan. Dit wordt gedaan in de functie *sleepNow()*, door te controleren of op het moment dat de interrupt wordt gegenereerd, de huidige tijd en datum gelijk zijn aan de tijd en datum die op het alarm ingesteld zijn. Wanneer dit het geval is, zullen de *tipAlarm()* functie en de *vacNotification()* functie (zie verderop in deze paragraaf) worden aangeroepen om te kijken welke tip of notificatie op het moment moet worden gegeven. Het alarm wordt op precies dezelfde datum en tijd gehouden, zodat het over precies een week weer af zal gaan. Pas als uit de globale byte *tipCounter* blijkt dat alle tips zijn gegeven (en dus de moeder niet meer zwanger is), wordt het alarm ofwel op één keer per maand gezet, als er door de gebruiker een kind in het apparaat is ingevoerd, of helemaal uitgezet indien de gebruiker nog geen kinderen in het apparaat heeft ingevoerd.

Notificaties

De structuur voor het geven van notificaties lijkt erg op die voor het geven van tips. Ze worden gebruikt om de gebruiker te notificeren dat een kind gevaccineerd moet worden volgens de kalender van bron [7]. De gebruiker kan vervolgens antwoorden of dit al gebeurd is of niet. Aan de hand van het geselecteerde antwoord krijgt de gebruiker feedback in de vorm van een tekst op het scherm (in de trant van ”goed gedaan” of ”zorg dat u zorgt dat dit alsnog gebeurd”). Voor de notificaties is de klasse vacQuestion aangemaakt met de volgende variabelen:

- byte *address*
- byte *vacMonth*
- boolean *chosenAnswer*

De byte *address* beschrijft wederom het adres in de EEPROM waar de tekst van de vraag staat opgeslagen. Byte *vacMonth* beschrijft het aantal maanden dat het kind oud moet zijn om de vaccinatie te ontvangen. In boolean *chosenAnswer* wordt opgeslagen welk antwoord de gebruiker heeft gekozen nadat er is gevraagd of het kind een bepaalde vaccinatie al heeft gehad. Verder heeft vacQuestion de volgende functies:

- *vacQuestion()*
- void *vacQuestionSet()*
- void *setChosenAnswer()*
- void *printScreen()*
- void *printAnswer()*

VacQuestion() is de constructor en zet boolean chosenAnswer op ’0’. *VacQuestionSet()* wordt aangeroepen na het creëren van een array van vacQuestion objecten om waardes aan de bytes *address* en *vacMonth* toe te kennen. *SetChosenAnswer()* wordt aangeroepen wanneer door de gebruiker wordt geselecteerd of het kind de vaccinatie al heeft gehad, om boolean *chosenAnswer* te veranderen. *PrintScreen()* en *printAnswer()* worden gebruikt om de notificatie en de antwoordkeuzes op het scherm te tonen (zie 5.3.1).

vacNotification

In de functie *vacNotification()* wordt op dezelfde wijze als bij *tipAlarm()* voor elk Child object dat in gebruik is (*usedChild*==1) gekeken hoe oud het kind is en of het al oud genoeg is om een bepaalde vaccinatie te ontvangen. Wanneer dit het geval is, wordt er een tekst op het scherm getoond waarin de gebruiker wordt gevraagd of het kind de vaccinatie al heeft gehad. Deze tekst ziet er bijvoorbeeld als volgt uit:

"Did the child born on 10/10/2010 get the 2nd month vaccinations yet? (RV#1, DTAP#1, HIB#1, PCV#1, IPV#1)"

Als dit het geval is en de gebruiker "ja" selecteert, zal in het Child object de boolean waarin dit wordt bijgehouden worden veranderd (zie 5.2.2). Is dit niet het geval, dan verandert deze waarde niet en zal bij de volgende keer dat de functie *vacNotification()* wordt aangeroepen de notificatie nog een keer op het scherm verschijnen. Na het selecteren van het antwoord van de gebruiker verschijnt er tekstuele feedback op het scherm. Als de gebruiker dan op een knop drukt, wordt de functie afgesloten.

VacNotification() wordt aangeroepen wanneer door de gebruiker in het Registration menu de optie "Add newborn" wordt gekozen. Nadat de geboortedatum is ingevuld wordt gekeken of het kind al oud genoeg is voor bepaalde vaccinaties. Het kan dus zo zijn dat, net als bij de tips voor de moeder, er op dat moment een aantal notificaties achter elkaar op het scherm verschijnen. Op de RTC wordt een alarm ingesteld dat zorgt dat de ATmega één keer per maand (de vijftiende dag, om twaalf uur) een interrupt krijgt. Dit gebeurt alleen wanneer de moeder niet óók zwanger is. In dit geval staat het alarm al ingesteld op één keer per week en wordt dat zo gelaten. Wanneer de ATmega een dergelijke interrupt krijgt roept hij de *tipAlarm()* functies en de *vacNotifcation()* functie aan zoals te zien in figuur 5.3.

5.3. Display software

5.3.1. De tekst op het scherm printen

Het printen van het scherm gebeurt bij de meeste functies op dezelfde manier. Een voorbeeld kan gevonden worden in bijlage A.4. Dit proces kan in meerdere deelprocessen verdeeld worden:

1. Tekst uit EEPROM halen, bewerken en tijdelijk opslaan
2. Tekst tonen op het scherm
3. Te kiezen antwoorden tonen op scherm en keuze opslaan

Tekst uit EEPROM halen, bewerken en tijdelijk opslaan

Om plaats in het geheugen vrij te maken voor een buffer met de grote van het aantal karakters dat op twee schermen past, wordt er allereerst een tweedimensionale char array van 12 x 15 gemaakt. Deze krijgt de naam "*screenBuffer*". De keuze voor de dimensies van deze buffer komen uit het gegeven dat er 6 regels per scherm zijn en er 14 karakters per regel zijn. Het 15de karakter is het "endline" karakter. Vervolgens wordt er een tijdelijke buffer aangemaakt met de naam "*downloadBuffer*". In deze buffer worden de karakters geschreven die uit de externe EEPROM gelezen worden. Wanneer deze buffer gevuld is met de functie *ReadEEPROM*, wordt de functie *PrepareScreenBuffer*(zie 5.3.2) aangeroepen. Hierin worden de woorden uit *downloadBuffer* op een nette manier over de regels van de screenBuffer verdeeld. Hierdoor zijn woorden in *screenBuffer* niet afgehakt aan het einde van een regel, maar verschijnt het woord netjes op de volgende regel. Om geheugenruimte te besparen wordt vervolgens de nu overbodig geworden *downloadBuffer* verwijderd.

Tekst tonen op scherm

De tekst uit de EEPROM is zoals hierboven besproken verdeeld in één of twee schermen nagelang het benodigde aantal regels. Wanneer de functie *PrepareScreenBuffer* aangeroepen wordt, retourneert deze functie het aantal schermen dat benodigd is om de tekst in de buffer te krijgen. Deze waarde krijgt de naam "*amountOfScreens*". Door een variabele genaamd *screenNumber* te maken en deze te laten veranderen van waarde met behulp van de omhoog en omlaag knoppen, kan een schermnummer

gekozen worden. Het gebufferde scherm dat correspondeert met dat nummer wordt vervolgens met de functie *LcdString* naar het scherm verstuurd, waarna de CPU direct in Power Down modus gaat. Wanneer er een knop ingedrukt wordt, wordt de CPU weer wakker, voert aan de hand van de gedrukte knop een actie uit en gaat direct weer slapen.

Te kiezen antwoorden tonen op scherm en keuze opslaan

Wanneer *screenNumber* groter is dan *amountOfScreens*, komt het programma bij de antwoorden terecht. Hier wordt met behulp van *screenNumber* het juiste antwoord geselecteerd en de kleuren van die regel worden geïnverteerd om aan te geven dat die regel geselecteerd is. Wanneer men op de knop "*select*" drukt, wordt de huidige regelnummer opgeslagen als het gegeven antwoord op de vraag (zie 5.2.1).

5.3.2. PrepareScreenBuffer

In deze functie wordt de tekst van een ééndimensionale char array op een nette manier in een tweedimensionale char array geplaatst. Met netjes wordt verstaan:

- Geen woorden worden afgekapt aan het einde van de regel, tenzij één woord langer is dan één regel
- De spaties die kunnen ontstaan aan het begin van de zin worden verwijderd
- De hoeveelheid gebruikte schermen worden geretourneerd aan het einde van de functie

Om dit te bereiken wordt er gebruik gemaakt van de functie *WriteToBuffer*.

WriteToBuffer

De functie *WriteToBuffer* krijgt als argument een pointer naar de *downloadBuffer* en een pointer naar de *screenBuffer* mee. Ook komt het nummer van de laatst geschreven regel in *screenBuffer* en de positie van het laatst gelezen karakter van *downloadBuffer* mee. De functie probeert vervolgens zoveel mogelijk woorden, vanaf de meegekregen positie, uit de *downloadBuffer* in de *screenBuffer* op één regel te krijgen. Wanneer de regel vol is of het woord te lang is én de tekst in *downloadBuffer* is nog niet af, roept *WriteToBuffer* zichzelf aan met een nieuwe waarde van de positie in *downloadbuffer* en het oude regelnummer verhoogd met één. Hierdoor gaat de nieuw aangeroepte *WriteToBuffer* verder waar de oude is gebleven, maar dan wel op de volgende regel in *screenBuffer*. Het resultaat van dit alles is dat wanneer de *downloadBuffer* volledig doorlopen is, de tekst netjes in *screenBuffer* staat.

5.3.3. LcdString

Deze functie is de basis voor het schrijven naar het scherm. De oorsprong van deze functie kan gevonden worden in bron [35]. Er zijn enkele wijzigingen aangebracht om deze functie beter te laten aansluiten op de functionaliteit van BabyShell. Wanneer de functie *LcdString* (tevens te vinden in A.3) aangeroepen wordt, worden er twee argumenten meegegeven. De eerste is een char pointer naar de plaats in het geheugen waar de te printen tekst te vinden is. De tweede bepaalt of de kleuren geïnverteerd moeten worden. Dit is om aan te geven dat de gebruiker een regel heeft geselecteerd. Ook is de functie zo gemaakt dat hij automatisch de rest van de regel vult met spaties wanneer de meegekregen regel korter is dan 14 tekens.

Op enkele momenten in het programma, bijvoorbeeld wanneer er een datum op het scherm moet worden getoond, moet de waarde van een byte op het scherm worden weergegeven. Omdat bij het aanroepen van de functie *LcdString* als eerste argument (de tekst die op het scherm moet worden weergegeven) alleen een char array kan worden meegegeven, moet hiervoor eerst de byte worden omgezet in een char array. Dit wordt gedaan door het gebruik van de functie *byteToCharArray(byte, byte, byte, char*)*.

5.4. Werkgeheugen besparing

De ATmega328P heeft (slechts) 2048 Bytes aan RAM geheugen beschikbaar voor tijdelijke opslag van variabelen. Babyshell heeft grote hoeveelheden tekst die, zoals hiervoor al besproken is, te groot zijn

om in het werkgeheugen te plaatsen. Er is gekozen om de diagnose vragen, de diagnoses, de tips en de notificaties in een extern EEPROM te plaatsen om zowel het FLASH als het RAM geheugen te ontzien.

Om het geheugen gebruik beter te bepalen, en de grootste verbruikers te vinden, is er gebruik gemaakt van de library ”MemoryFree” [36]. De functie *FreeMem()* van deze library retourneert na aanroep de hoeveelheid nog vrij te gebruiken geheugen op dat moment, waarna de waarde met een simpele Serial.print op de datalijn gezet kan worden.

Aanmaken van objecten

Voor het programma zijn een aantal classes gemaakt, welke in secties 5.1 en 5.2 zijn besproken. Het betreft de classes Question, Child, Mother, TipClass en vacQuestion. Van twee van deze classes, Question en TipClass, worden in het programma een groot aantal (respectievelijk 32 en 38) objecten aangemaakt. Bij het ontwerpen van de code voor het deel dat de zelfdiagnose functie van het apparaat mogelijk maakt, heeft dat tot een aantal problemen geleid wat betreft het gebruik van werkgeheugen. De volgende stappen zijn tijdens het programmeren gemaakt om uiteindelijk tot een manier te komen om alle Question objecten te kunnen gebruiken zonder het veroorzaken van geheugenproblemen:

1. Het globaal aanmaken van de Question objecten
2. Het globaal aanmaken van Question objecten en deze stoppen in globale vectoren
3. Het lokaal aanmaken van Question objecten en deze stoppen in globale vectoren
4. Het lokaal aanmaken van alle Question objecten
5. Het lokaal aanmaken van alle Question objecten verdeeld over drie functies

De eerste stap in het proces is geweest om alle Question objecten globaal aan te maken. Op dit moment werd nog niet verondersteld dat het werkgeheugen een beperking op zou leveren. Nog voor het programma werd getest is daarna overgestapt op de tweede versie, het aanmaken van Question objecten om deze vervolgens in een C++ std::vector te stoppen. Door het opslaan van de objecten in vectoren zou er handig gebruik gemaakt kunnen worden van bepaalde functies van de vector, zoals *at*, *size* en *begin*. Voor het gebruik van vectoren in Arduino IDE is echter wel het gebruik van een library vereist. Deze versie van de code leverde bij het testen problemen met het geheugengebruik op, dus werd gekeken naar een andere mogelijkheid om de objecten op te slaan. Bij het testen werd gelet op het werkgeheugengebruik dat door de Arduino IDE tijdens het compileren werd aangegeven. Er werd getest of het lokaal aanmaken van de Question objecten in een functie, om deze vervolgens in een globale vector te stoppen en daarna te verwijderen, ten goede kwam van het geheugengebruik. Er werd gedacht dat dit wellicht een efficiëntere manier was om alle objecten op te slaan. Tijdens het compileren gaf de Arduino IDE aan dat er op deze manier inderdaad minder werkgeheugen werd gebruikt, maar bij het testen bleven de problemen zich voortdoen. Het vermoeden ontstond dat de Arduino IDE het geheugengebruik van de vectoren niet goed kon berekenen. Daarom werd gebruik gemaakt van de library MemoryFree. Hieruit bleek dat de vectoren meer geheugen verbruikten dan de losse objecten en dus werd er afgezien van de keuze voor deze structuur. Dit heeft als bijkomend voordeel dat er geen gebruik van de vector library meer hoeft te worden gemaakt, wat ook scheelt in het geheugengebruik.

De volgende stap was het lokaal aanmaken van alle Question objecten. Alle code die benodigd was voor de diagnose functionaliteit werd vervolgens in dezelfde functie gestopt. Na het gebruik van de functie worden de objecten automatisch weer verwijderd. Met *FreeMem()* werd wederom geconcludeerd dat op deze manier het werkgeheugengebruik te groot was, en het programma na het aanmaken van een bepaald aantal objecten problemen ondervond. Om deze reden is ten slotte gekozen om alle Question objecten te verdelen over drie functies. In elke functie worden er Question objecten aangemaakt van alle vragen en diagnoses betreffende één symptoom (bijvoorbeeld diarree), waarna er wordt gecontroleerd op de te geven diagnose betreffende enkel dat symptoom. Wanneer de functie wordt afgesloten, worden alle Question objecten automatisch verwijderd zodat er weer ruimte is voor een nieuwe set objecten en de functie van het volgende symptoom kan worden aangeroepen. Een dergelijke structuur is later ook geïmplementeerd voor de TipClass, waarbij de objecten worden aangemaakt in drie afzonderlijke functies die één voor één worden aangeroepen (zie 5.2.2).

5.4.1. Teksten en het geheugengebruik

Omdat de Arduino environment alle lokale char arrays in het globale RAM geheugen zet en dit een te groot geheugenverbruik veroorzaakt, moeten ook de lokale teksten zoals gebruikt bij bijvoorbeeld titels van menu opties (zie 5.1.3) in het FLASH geheugen opgeslagen worden. Normaal zou men kiezen dit met de in Arduino IDE gebruikelijke *F()* functie te doen. Deze functie regelt alles m.b.t. teksten in FLASH geheugen plaatsen en het op het juiste moment in het RAM geheugen laden. Omdat deze functie echter niet werkt in de gebruikte functie om tekst op het scherm te tonen, is er een eigen functie genaamd *downloadFromProgmem* geschreven. Deze functie schrijft de tekst uit het FLASH geheugen in de, als pointer meegegeven, char array. Hierdoor wordt vermeden dat deze teksten in het globale RAM geheugen worden gezet. Door deze functie pas aan te roepen vlak vóór het schrijven van het scherm, kan er zo optimaal mogelijk van het geheugen gebruik gemaakt worden.

5.4.2. Diverse

Zoals besproken in sectie 5.3.1 wordt er tijdens het printen van het scherm gebruik gemaakt van de *calloc* en *free* functies om het mogelijk te maken de char array *downloadBuffer* weg te gooien vóórdat de functie eindigt. Dankzij het gebruik van deze functies komen er 128 bytes aan geheugen vrij. Dit heeft als voordeel dat na het verzenden van tekst naar het scherm, in de daarop volgende slaapfunctie het programma een lager geheugengebruik heeft dan wanneer dit niet gedaan zou zijn. In principe gebruikt de *sleepNow()* functie weinig tot geen extra RAM geheugen, ware het niet dat tijdens de Power Down modus het systeem wakker gemaakt kan worden door een notificatie of tip. Dit gaat gepaard met tekst op het scherm printen en dus ook met een groot geheugengebruik.

5.5. Interrupt

De Interrupts worden gebruikt om het systeem uit Power Down modus te halen. Omdat de ATmega328P maar twee hardware interrupt I/O pinnen heeft, wordt gebruik gemaakt van de library ”PinChangeInt versie 2.40 RC2” van bron [37]. Met behulp van #define kunnen één of meerdere ATmega I/O poorten ingesteld worden om te werken met deze library. Omdat de knoppen (zie 4.2) analoog uitgelezen worden, wordt de poort (dat is een onderdeel van de microcontroller waar meerdere I/O pinnen aan aangesloten zijn) die aan de analoge ingangen hangt gebruikt voor de software interrupt. Dit is in dit geval poort C. Om vervolgens geheugen te besparen zijn de andere interrupt lijnen ook aan I/O pinnen aangesloten die op dezelfde poort zitten. Hierdoor wordt er dus maar één poort gebruikt voor alle externe interrups, namelijk poort C. De overige I/O ingangen kunnen nog steeds gebruikt worden voor uitlezing.

In deze library zitten een aantal functies om het geheugengebruik te verminderen. Zo kan men de volgende defines declareren:

- NO_PIN_STATE
- NO_PIN_NUMBER
- DISABLE_PCINT_MULTI_SERVICE

Omdat er geen gebruik gemaakt wordt van het nummer van de laatste pin die geïnterrupt heeft, de status daarvan ook niet gebruikt wordt én er ook niet meerdere interrupts tegelijk verwerkt hoeven te worden kunnen deze opties gedefinieerd worden. Uiteindelijk zijn dus de volgende defines gebruikt:

```
//#define NO_PORTC_PINCHANGES
#define NO_PORTD_PINCHANGES
#define NO_PORTB_PINCHANGES
#define NO_PIN_STATE
#define NO_PIN_NUMBER
#define DISABLE_PCINT_MULTI_SERVICE

#include <PinChangeInt.h>
```

Verdere implementatie van de Power Down modus en interrupts kan gelezen worden in de thesis van Gerard Hogenhout en Kees Hogenhout [9].

5.6. Software testen

5.6.1. DebugUtils library

Om het debuggen gemakkelijker te maken is er gekozen om gebruik van de library ”*DebugUtils*” te maken. Deze library wordt in- of uitgeschakeld door het wel of niet definiëren van ”*#define DEBUG*”. Door ”*DEBUG_PRINT(string)*” aan te roepen in het bestand, wordt op die plaats over de seriele lijn de timer *millis()* (dat is de tijd in milliseconden dat het systeem draaiend is sinds de laatste reset), de functienaam, de bestandsnaam en het regelnummer verstuurd. Ook wordt de meegegeven string verstuurd.
Het grote voordeel van deze library is dat hij met één simpele define helemaal uitgeschakeld kan worden, waarmee in één keer alle *DEBUG_PRINT* opdrachten overgeslagen worden. Tevens wordt op deze manier het geheugengebruik van de library en de daarbij behorende functies naar 0 gebracht.

5.6.2. Versiestructuren

In dit project is een duidelijke versiestructuur gebruikt. De gebruikte benaming is als volgt: ”Naam onderdeel hoofdversie_subversie_subsubversie_naam eigenaar”.

- Naam onderdeel, de naam van het onderdeel van de code voordat alles samengevoegd is, bijvoorbeeld ”Scherm_systeem”.
- Hoofdversie, wanneer er een grote wijziging doorgevoerd wordt, wordt dit nummer verhoogd.
- Subversie, dit zijn alle deelversies binnen de hoofdversie. Hierin werden alle wijziging doorgevoerd die elke persoon had gemaakt in zijn eigen subsubversie.
- Subsubversie en Naam eigenaar, deze twee zijn altijd gekoppeld geweest. Wanneer een persoon een niet geteste wijziging wilde doorvoeren diende die persoon een eigen subsubversie met zijn naam erachter te maken. Wanneer alle wijzigingen werkten en goedgekeurd waren mochten deze wijzigingen doorgevoerd worden in de subversie. Hiervoor diende dan toestemming gevraagd te worden aan de groep, om zogenaamde ”conflicted copies” te vermijden.

Op deze manier werd bij het aanpassen van een werkende subversie altijd een eigen versie aangemaakt, die werd getest tot alle aanpassingen het deden. Wanneer meerdere personen in een eigen versie aanpassingen aan het maken waren, werden deze pas samengevoegd wanneer de afzonderlijke versies elk getest waren en goed werden bevonden. Wanneer deze versies werden samengevoegd tot een nieuwe subversie, werd getest of functies van de oude subversie én alle nieuwe functies naar behoren werkten.

5.6.3. Input van de gebruiker

Bij het testen van BabyShell is het belangrijk dat er geen fouten kunnen optreden doordat de gebruiker op een bepaald moment op een knop druk, terwijl dan niet in het programma staat gedefinieerd wat er met deze input moet worden gedaan. Om deze situatie te voorkomen is het programma op de volgende manier geschreven:

Wanneer de CPU zich niet in de Power Down modus bevindt, is er door middel van ”detachInterrupt” in de functie *ButtonPINFalling()* voor gezorgd dat de buttons geen interrupt meer kunnen genereren. Wanneer de gebruiker de knoppen op dit moment indrukt zal er niks gebeuren. Wanneer het programma zich wél in de Power Down modus bevindt, en de knoppen dus een interrupt kunnen genereren, altijd voor het indrukken van elke knop is gedefinieerd wat het programma moet doen. Wanneer is gedetecteerd dat een bepaalde knop ”HIGH” is gemaakt, wordt de taak die bij het indrukken van deze knop hoort uitgevoerd en de knop weer ”LOW” gemaakt. Op deze manier kan de gebruiker op geen enkel moment in het programma met een druk op een knop een fout van het programma veroorzaken (zie 5.1.1).

5.6.4. Geheugengebruik

Zoals beschreven in sectie 5.4 is het geheugengebruik altijd een groot punt van aandacht geweest tijdens dit project. Ten tijden van schrijven van deze thesis wordt door het programma (versienummer V10_1) in totaal 26.910 bytes (87%) FLASH, 1.049 bytes RAM aan globale variabelen, 36 bytes intern EEPROM en 10.496 bytes extern EEPROM gebruikt.

6

Ethische paragraaf

Als toekomstige ingenieurs en ontwikkelaars van dit apparaat moet er ook gekeken worden naar de ethische implicaties van BabyShell. Een van de ethische vragen is: In hoeverre mogen wij ons als niet-medisch opgeleide mensen bezig houden met deze oplossingen?

Voor het Bachelor afstudeerproject is het de bedoeling dat gedaan wordt alsof de studenten een team van een ingenieursbureau vormen die een opdracht uitvoeren. In die zin zou het ontwikkelen van BabyShell de functionele verantwoordelijkheid van het team zijn. De specificaties waren duidelijk en het was vooral aan de studenten om een apparaat te maken dat aan die specificaties voldeed. De impact van het apparaat zou volgens het separatisme dan niet meer onze verantwoordelijkheid zijn. De informatie en flowcharts die worden gebruikt, zijn afkomstig van de WHO en zijn samengesteld door artsen. Op basis daarvan mag men verwachten dat het redelijk accurate informatie is, maar het kan voorkomen dat een gebruiker een onjuist advies krijgt van het apparaat en daarmee de conditie van de gebruiker door een onjuiste handeling wordt verslechterd. Wie is er in dat geval verantwoordelijk?

Deze vraag is eigenlijk niet alleen relevant voor gebruikers in Afrika, maar voor vrijwel de hele westerse wereld. Ook in Nederland kan men door rond te zoeken op internet snel een zelfdiagnose stellen. Het verschil is echter dat iedereen in Nederland zich ervan bewust is dat deze sites zich niet verantwoordelijk houden voor jouw zelfdiagnose gevonden op een site en dat het vaak toch raadzaam is om naar de arts te gaan voor professioneel advies. Dit heeft als gevolg dat deze vraag ethisch gezien wel speelt in Nederland, maar rechtelijk gezien niet. Ethisch wel, omdat er waarschijnlijk veel mensen zijn die na zelf een diagnose gesteld te hebben op internet niet meer naar de huisarts gaan, maar rechtelijk niet omdat deze mensen waarschijnlijk weinig succes zullen hebben als ze in gevallen van een verkeerde zelfdiagnose de website verantwoordelijk houden.

In Afrika zijn mensen door het gebrek aan medische kennis doorgaans slechter geïnformeerd dan hier en nemen ze het antwoord van BabyShell misschien sneller voor waar aan. De diagnoses van het apparaat zouden dus eigenlijk een stuk accurater moeten zijn, maar zelfs als dit zo zou zijn blijft het stellen van een zelfdiagnose met behulp van vragen een onbetrouwbare methode. Het probleem is echter dat mensen daar vaak geen beschikking hebben tot een arts en dus geen alternatieven hebben. Als er vanuit het utilisme naar deze kwestie gekeken wordt zijn er slechts een paar mogelijkheden als er vanuit wordt gegaan dat mensen sowieso niet naar de arts zullen gaan:

1. Mensen zijn ziek en krijgen de juiste diagnose
2. Mensen zijn ziek en krijgen de verkeerde diagnose
3. Mensen zijn niet ziek en krijgen de juiste diagnose
4. Mensen zijn niet ziek en krijgen de verkeerde diagnose

In geval 2 en 4 treedt er een probleem op. In geval 2 kunnen mensen ziek zijn en een verkeerde diagnose krijgen. Het alternatief zonder BabyShell is dat ze ziek zijn en geen diagnose krijgen omdat ze niet naar de dokter gaan. Wat is in dit geval beter? Als we in het achterhoofd houden dat deze flowcharts zijn bedoeld voor redelijk simpele ziektes zijn de aangeraden remedieën ook niet zo ingrijpend. In het geval van een ingewikkeldere complicatie wordt de gebruiker naar de dokter verwezen. Ze kunnen de eigenlijke ziekte dus niet verergeren. Effectief houd je dus hetzelfde over: er gebeurt niks om de ziekte te verbeteren. Zo is het ook in geval 4.

Kort samengevat zou je dus kunnen zeggen dat het hebben van BabyShell leidt tot kansen op een juiste diagnose en een verkeerde, terwijl het niet hebben van BabyShell en ook geen dokter leidt tot helemaal geen diagnose. Aangezien een verkeerde diagnose voor de simpele ziektes geen erge gevolgen heeft is het dus gerechtvaardigd om toch BabyShell aan te bieden in de Afrikaanse landen. Wel moeten de mensen bewust gemaakt worden dat de BabyShell zeker geen vervanging is voor een arts en wanneer ze zich niet goed voelen ze toch zeker moeten proberen een arts op te zoeken. BabyShell heeft daarom vaak als uiteindelijk advies dat er toch naar een arts wordt verwezen. Het apparaat is namelijk niet alleen maar om mensen zelfdiagnoses te laten stellen, maar ook om ze in te laten zien dat het vaak toch verstandig is om naar een arts te gaan.

7

Resultaten

In het hoofdstuk Probleemstelling zijn een aantal eisen gesteld waaraan het apparaat moet voldoen. Om de eisen die specifiek waren voor deze thesis te bewerkstelligen is er voor de volgende hardware gekozen:

- Microcontroller: Atmel ATmega 328P
- Bediening: Momentary Tactile Push Button Switch 6*6*5mm
- Diplay: Equivalent Nokia 5110 Screen
- Externe opslag: Microchip 24AA256 EEPROM

De code is geschreven met behulp van het programma Arduino IDE 1.6.4. De taal die gebruikt wordt in Arduino IDE is gebasseerd op C/C++ [21]. Om de verschillende vragen, diagnoses, tips en notificaties op een gestructureerde manier op te slaan is veelvoudig gebruik gemaakt van de class structuur die C++ biedt. Lange teksten zijn opgeslagen in de externe EEPROM, om het gebruik van het werkgeheugen en de interne EEPROM te ontzien. De code is op dergelijke wijze geschreven dat het uitbreiden van de medische functionaliteit (meer vragen over ziektes, diagnoses, tips en notificaties) en menustructuur relatief eenvoudig is. Ook is alle tekst die op het scherm wordt getoond zo centraal mogelijk geplaatst, hetgeen het vertalen naar een andere taal eenvoudig maakt.

Van elke eis die gesteld is in de probleemstelling en specifiek van toepassing is op deze thesis kan worden geëvalueerd of deze behaald is:

1. **Het hoofdmenu bestaat uit minstens de twee onderdelen:**
   1.**Medical (met daaronder de twee subonderdelen ”Pregnancy” en ”New born”)**
   2.Klok

   Het hoofdmenu van het prototype van BabyShell bestaat uiteindelijk uit vijf onderdelen: ”Diagnosis”, ”Registration”, ”Clock”, ”Temperature” en ”Settings”. Wanneer de gebruiker de optie Registration selecteert, komt deze in het zogenaamde Registration menu terecht. Hierin kunnen de opties ”Add newborn”, ”Remove child”, ”New preg(nancy)”, ”Remove preg(nancy)” en ”Vaccinations” worden gekozen. Deze structuur is anders dan degene die in eerste instantie is afgesproken, maar in een later stadium van het ontwerp goed bevonden door de opdrachtgever. Er is voor deze andere structuur gekozen omdat de gebruiker uit meer opties kan kiezen dan bij het begin van het project werd gedacht, en een uitgebreidere menu structuur dus was vereist.

2. **Wanneer ”Pregnancy” wordt gekozen, moet eenmalig de datum van de laatste menstruatie ingevoerd worden. BabyShell moet dan op basis van de opgeslagen tijd de weken bij gaan houden en de moeder wijzen op prenataal doktersbezoek. BabyShell moet ook wekelijks medische tips geven die toepasselijk zijn in die zwangersschapsperiode volgens bron [6].**

   Wanneer de gebruiker de optie ”New preg” kiest in het Registration menu, is het mogelijk de datum van de laatste menstruatie in te voeren. Deze datum wordt opgeslagen in de interne EEPROM, zodat bij het uitschakelen of resetten van het apparaat de data niet verloren gaat. Door middel van het instellen van een alarm op de RTC wordt wekelijks een interrupt gegenereerd, waarna wordt berekend hoeveel weken de moeder zwanger is en welke tip ze moet ontvangen. Een deel van deze tips herinnert de zwangere vrouw aan haar doktersbezoeken.

3. **Wanneer ”New born” wordt gekozen, moet eenmalig de geboortedatum ingevoerd worden. Vervolgens moeten er vaccinatie herinneringen voor kinderen tot 3 jaar gegeven worden volgens bron [7].**

   Wanneer de gebruiker de optie ”Add newborn” kiest, is het mogelijk de geboortedatum van een kind in te voeren. Ook deze datum wordt opgeslagen in de interne EEPROM, zodat deze niet verloren gaat bij het uitschakelen of resetten van het apparaat. BabyShell kan de gegevens onthouden van maximaal vijf kinderen. Door middel van het instellen van een alarm op de RTC wordt maandelijks een interrupt gegenereerd waarna wordt gecontroleerd of een van de kinderen al oud genoeg is om bepaalde vaccinaties te ontvangen. Wanneer dit het geval is zal een een notificatie op het scherm verschijnen, waarna de gebruiker aan kan geven of het kind de vaccinaties wel of niet heeft ontvangen. De gebruiker zal deze notificatie eens in de maand blijven ontvangen, totdat er is aangegeven dat het kind de desbetreffende vaccinatie heeft gekregen. Tevens kan in het Registration menu bij de optie ”Vaccinations” van elk kind worden bekeken welke vaccinaties deze al heeft gehad en welke niet.

4. **Onder ”New born” moet ook een kopje zijn met ”Diagnosis” waar de flowcharts geïmplementeerd worden voor de herkenning van longontsteking en oorinfecties met behulp van input van de gebruiker. Op basis van deze diagnose geeft BabyShell een medisch advies of verwijst naar een medische post.**

   In het hoofdmenu van de BabyShell kan de gebruiker kiezen voor de optie ”Diagnosis”, waar de gebruiker door een flowchart wordt geleid door middel van het beantwoorden van vragen betreffende medische problemen. De medische problemen, of symptomen, die op deze versie van BabyShell zijn geïmplementeerd zijn longontsteking, oorinfectie en diarree. Op basis van de geselecteerde antwoorden zal BabyShell een diagnose kiezen en advies op het scherm tonen.

5. **Het is mogelijk om de software van de BabyShell te updaten met een computer.**
   Met een USB to TTL adapter kan met Arduino IDE de software op de ATmega op een eenvoudige wijze geüpdate worden. Wel moet eerst de juiste bootloader op de ATmega geflashed worden. Dit laatste hoeft maar één keer te gebeuren.

6. **De materiaalkosten van deze versie van BabyShell mogen niet meer dan $6 bedragen.**
   Het is gelukt de component kosten van BabyShell, voor zowel de zonnecel versie als de batterij versie, onder de $6.00 te houden. Alle kosten van de gebruikte componenten zijn in de tabellen op de volgende pagina beschreven. Er is hierbij gerekend met productie aantallen van tussen de 500.000 en 1.000.000 stuks.

Tabel 7.2: Kostenoverzicht voor BabyShell met batterij

Naam	Aantal	Prijs/stuk(USD)	Subtotaal(USD)
Thermometer: Druppel lijm	1	0,02	0,02
Thermometer: Plaatje	1	0,01	0,01
Thermometer: NTC	1	0,07	0,07
RTC DS3231	1	0,31	0,31
Buzzer	1	0,20	0,20
Nokia 5110	1	1,29	1,29
Buttons	6	0,01	0,06
EEPROM 24AA256	1	0,09	0,09
Atmega328	1	1,12	1,12
PCB	1	0,17	0,17
CR2450	1	0,75	0,75
Battery Holder	1	0,15	0,15
TLV61220	1	0,35	0,35
Weerstanden	16	0,008	0,13
Condensatoren	6	0,005	0,03
Spoelen	2	0,013	0,03
NMOS	3	0,049	0,15
PMOS	2	0,010	0,02
Male headers	1	0,098	0,10
Female headers	1	0,08	0,08
Casing	1	0,10	0,10
Totaal			5,22

Tabel 7.1: Kostenoverzicht voor BabyShell met zonnecellen

Naam	Aantal	Prijs/stuk(USD)	Subtotaal(USD)
Thermometer: Druppel lijm	1	0,02	0,02
Thermometer: Plaatje	1	0,01	0,01
Thermometer: NTC	1	0,07	0,07
RTC DS3231	1	0,31	0,31
Buzzer	1	0,20	0,20
Nokia 5110 scherm	1	1,29	1,29
Buttons	6	0,01	0,06
EEPROM 24AA256	1	0,09	0,09
Atmega328	1	1,12	1,12
Zonnecel	1	1,0	1,0
PCB	1	0,17	0,17
LIR2450	1	0,70	0,70
TP4056	1	0,07	0,07
LP5907	1	0,14	0,14
Battery Holder	1	0,15	0,15
Weerstanden	16	0,008	0,13
Condensatoren	6	0,005	0,03
Spoelen	1	0,013	0,013
NMOS	1	0,049	0,049
PMOS	0	0,010	0,0
Male headers	1	0,098	0,098
Female headers	1	0,08	0,08
Casing	1	0,10	0,10
Totaal			5,90

De meeste van deze prijzen zijn gevonden op de website `Aliexpress.com`. Ook is een groot deel van deze prijzen gecontroleerd door de leverancier van de opdrachtgever. De mogelijkheid bestaat dat de componenten voor nog lagere prijzen te verkrijgen zijn, daar zou eventueel nog verder onderzoek naar gedaan kunnen worden.

7. **BabyShell moet gegevens van tot en met 5 kinderen bij kunnen houden en per kind tijdsafhankelijke tips kunnen geven.**

   Zoals reeds besproken in dit hoofdstuk is BabyShell in staat de gegevens van vijf kinderen op te slaan en afhankelijk hiervan tijdsafhankelijke tips te kunnen geven. De tips betreffende de kinderen zijn in deze versie van BabyShell beperkt tot het herinneren van de moeder wanneer het kind oud genoeg is voor bepaalde vaccinaties.

8. **BabyShell moet per kind een resetfunctie hebben om in het geval van foute of gedateerde invoer te kunnen resetten.**

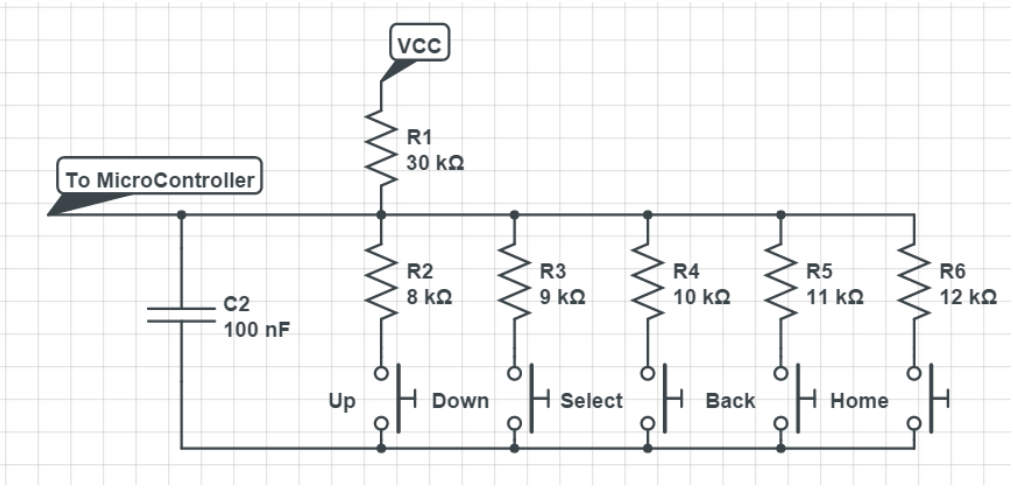
   Wanneer in het Registration Menu voor de optie ”Remove child” wordt gekozen kan de gebruiker kiezen van welk kind de gegevens verwijderd moeten worden. Bij de optie ”Remove preg” is het mogelijk de gegevens van de zwangerschap te verwijderen. Dit wordt gedaan door aan alle variabelen van het Child of Mother object weer de oorspronkelijke waarde van ’0’ toe te kennen.

8

Aanbevelingen

Met dit Bachelor afstudeerproject is een prototype gerealiseerd waarbij de focus is gelegd op het voldoen aan de functionele specificaties van BabyShell zoals deze zijn afgesproken met de opdrachtgever. Gedurende het proces werden er echter (mogelijke) verbeteringen voor het apparaat bedacht. Hoewel veel van de voorstellen hieronder relatief simpel te implementeren zijn, was er richting het einde van het project te weinig tijd om hier nog aandacht aan te besteden. Er moest gefocust worden op de hoofdfunctionaliteiten van het apparaat. Daarom worden de volgende aanbevelingen gedaan, voor wanneer er verder aan het product wordt gewerkt. Mogelijk wordt aan enkele van deze aanbevelingen nog gewerkt na het inleveren van de thesis.

Zoals besproken in hoofdstuk 4.2, is er tijdens het ontwerpen een fout gemaakt met de verdeling van de weerstandswaarden, waardoor de functie van het detecteren van twee knoppen die tegelijk ingedrukt worden is vervallen. Ook zouden de weerstandswaarden zo gekozen moeten worden dat de spanningsdeling altijd onder de $0,3V_{cc}$ komt bij het indrukken van een knop. Dit kan op eenvoudige wijze verholpen worden door een $30k\Omega$ weerstand te nemen als pull-up weerstand en elke knop een eigen weerstand te geven met waardes van $8k$ oplopend tot $12k\Omega$, zoals te zien in figuur 8.1.



Figuur 8.1: Verbeterd schema van de knoppen

In de huidige versie van BabyShell zal om het apparaat in een andere taal te zetten, de teksten in de bestanden ”Burn_Tekst_v5” en ”FlashMem.ino” moeten worden veranderd, zoals besproken in sectie 5.1.3. In feite zouden er dan verschillende versies van deze bestanden kunnen zijn, waarbij voor een bepaalde taal de juiste versie op de ATmega wordt geflashed en in de EEPROM wordt opgeslagen. Hoewel het op deze manier relatief makkelijk is om de code voor de BabyShell naar een andere taal te vertalen, is dit niet ideaal. Ideaal zou zijn wanneer alle talen tegelijkertijd op de ATmega zouden

zijn geflashed, en in de EEPROM zouden zijn opgeslagen. De gebruiker zou dan in het Settings menu de taal kunnen aanpassen. In de code moeten voor deze optie globale variabelen veranderd worden, waardoor in het programma, wanneer er tekst op het scherm wordt getoond, kan worden gecontroleerd in welke taal dat moet. Het probleem van dit voorstel is vooral dat de externe EEPROM een significant grotere geheugencapaciteit (bij twee talen twee keer zoveel) nodig heeft. Ook het FLASH geheugen zou dan zwaarder belast gaan worden. Wanneer de opdrachtgever tegen een meerprijs het waard zou vinden een externe EEPROM met grotere geheugencapaciteit te gebruiken, zou dit voorstel kunnen worden geïmplementeerd.

Wanneer het de bedoeling gaat zijn om de gebruiker de mogelijkheid te geven het apparaat te updaten, of een andere vorm van communicatie met een computer mogelijk te maken, zou het aan te bevelen zijn om een ICT'er een programma te laten schrijven dat dit proces vergemakkelijkt. Om het apparaat nu helemaal compleet te programmeren is het nodig om eerst het programma met de naam "*Burn_Tekst_v5*" te flashen. Hierna kan pas het hoofdprogramma op de ATmega geladen worden. Dit is een omslachtig proces wat vergemakkelijkt moet worden om het updaten mogelijk te maken voor de eindgebruiker.

Bij de optie "Temperature" in het hoofdmenu wordt de temperatuur in graden Celsius getoond. In sommige landen wordt echter ook de eenheid Fahrenheit gebruikt. Een mogelijke toekomstige verbetering voor het apparaat zou zijn dat de gebruiker in het "Settings" menu kan kiezen in welke eenheid de temperatuur wordt getoond.

De optie "Diagnosis" leidt de gebruiker door een aantal vragen welke beantwoord moeten worden voor het stellen van een diagnose. Wanneer de gebruiker een vraag heeft beantwoord, wordt het antwoord opgeslagen en de volgende vraag op het scherm getoond. Indien de gebruiker de vraag per ongeluk fout heeft beantwoord, is er in de huidige versie niet de optie om terug te gaan naar de vorige vraag en dit te veranderen. Wel kan er door het gebruik van de home knop meteen terug worden gegaan naar het hoofdmenu, om de optie "Diagnosis" nogmaals te selecteren en de vragen opnieuw te beantwoorden. In een toekomstige versie van BabyShell zou kunnen worden geïmplementeerd dat door middel van de back knop de gebruiker naar de vorige vraag terug zou kunnen gaan om een foutief gegeven antwoord te corrigeren.

BabyShell maakt voor het gebruik van het apparaat gebruik van slechts vijf knoppen. Hierdoor is het moeilijk om bepaalde dingen op het apparaat in te voeren, zoals bijvoorbeeld een naam. Daarom is er bij de huidige versie van BabyShell gekozen om voor de kinderen geen naam in te voeren, maar enkel de geboortedatum. Wanneer op het scherm een lijst van alle kinderen wordt getoond (bijvoorbeeld bij "Vaccinations"), worden dus alle geboortedata getoond en moet de gebruiker op basis daarvan herkennen welk kind hij of zij wil selecteren. Het zou beter zijn als op dit moment een lijst met namen op het scherm zou worden getoond waaruit de gebruiker kan kiezen. Tijdens het project is het niet gelukt om het invoeren van namen op een gebruikersvriendelijke manier te implementeren, maar wanneer hier meer onderzoek naar gedaan wordt kan er wellicht een manier voor worden gevonden.

Voor kinderen tot en met vier jaar ontvangt de gebruiker notificaties wanneer het kind moet worden gevaccineerd. Dit zijn echter de enige notificaties die de gebruiker ontvangt die het jonge kind betreffen. Net als dat tijdens de zwangerschap de gebruiker een groot aantal tips ontvangt, zouden ook wat meer tips geïmplementeerd kunnen worden voor tijdens de eerste levensjaren van het kind.

Tijdens het opstarten van het apparaat zou BabyShell een beginscherm kunnen tonen, met bijvoorbeeld de tekst "BabyShell" met het versienummer, voordat de gebruiker naar het hoofdmenu gaat.

Wanneer het apparaat ontwaakt uit de Power Down modus, zou op het scherm de klok kunnen worden getoond, waarna bij het indrukken van een knop verder wordt gegaan naar waar de gebruiker vóór de Power Down modus was gebleven. Dit vergroot de functionaliteit van het apparaat, omdat op gemakkelijke wijze BabyShell kan worden gebruikt om even te kijken welke tijd het is. Deze functionaliteit werkt echter wel een hoger stroomgebruik in de hand omdat het appraat vaker uit de Power Down modus moet ontwaken.

9

Conclusie

Aan het begin van dit Bachelor afstudeerproject is door de opdrachtgever een probleem gepresenteerd: de slechte gezondheid van zwangere vrouwen en jonge kinderen in ontwikkelingslanden. Het te ontwikkelen product, BabyShell, moet bijdragen aan het verhelpen van dit probleem. Samen met de opdrachtgever is een programma van eisen opgesteld, met functionele specificaties waaraan het apparaat moet voldoen. BabyShell werkt compleet onafhankelijk van het stroomnet en heeft minder dan 6 dollar aan componentkosten. Door middel van een flowchart kan de gebruiker via een vraag-antwoord proces naar een mogelijke diagnose geleid worden. Ook is BabyShell in staat om taken te verrichten zoals tijd bijhouden, temperatuur opnemen en reminders versturen voor belangrijke gebeurtenissen als prenatale doktersbezoeken en vaccinaties.

Op basis van de eisen die specifiek van toepassing zijn op deze thesis is er een keuze gemaakt voor een aantal hardware componenten (microcontroller, display, bediening en externe opslag), en is gekozen om de code te schrijven met behulp van het programma Arduino IDE 1.6.4. De code is op dergelijke wijze geschreven dat het uitbreiden van de functionaliteit van BabyShell relatief eenvoudig is.

Wanneer wordt gereflecteerd op het programma van eisen, kan worden geconcludeerd dat aan alle eisen die specifiek van toepassing zijn op deze thesis is voldaan. Een enkele eis is op een iets andere wijze geïmplemteerd dan bij het opstellen van het programma van eisen is gedefinieerd. Dit is een gevolg van het feit dat er op dat moment niet goed ingeschat werd hoe de menu structuur het beste geïmplementeerd kon worden. Deze aanpassingen zijn door de opdrachtgever goed bevonden. Tijdens het ontwerpproces zijn ook enkele additionele opties besproken die het apparaat mogelijk nog beter zouden maken, maar wegens een gebrek aan tijd niet meer toegevoegd konden worden. Deze aanbevelingen zijn beschreven, en wanneer verder wordt gewerkt aan BabyShell zou hier naar moeten worden gekeken.

A

Bijlage

A.1. Handleiding Flashen

Om de ATmega 328P te flashen en de bootloader te burnen zijn de volgende aanpassingen vereist:

1. Voor het gebruik van een CP2102 powered usb naar ttl converter is het nodig een driver te installeren. Deze driver heet CP210x_VCP_Windows.zip
   Bron: https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx
2. Voor het gebruik van de UsbTiny bootloader brander is de volgende driver van Adafruit nodig:
   (a) usbtiny_signed_8.zip voor Windows 8
   (b) usbtinyisp_libusb-win32-64_1.2.1.0 voor Windows 7

   Bron: https://learn.adafruit.com/usbtinyisp/drivers
3. De hierbij geleverde mappen met de naam libraries en hardware moeten beide geplaatst worden in de mijn documenten\arduino map.

Alles is getest op Arduino IDE versie 1.6.4 op een x64 Windows 7 pc.

Voordat er geflashed kan worden dient de “ATmega328 on a breadboard (8 MHz internal clock)” in het hulpmiddelen/board menu van Arduino IDE geselecteerd te worden. Verbind de CP2102 met de 5 pinnen van Babyshell met de bijgeleverde kabel. Sluit hierna de CP2102 aan op de computer. Daarna dient de juiste com poort geselecteerd te worden.

Wanneer de bootloader geupdate moet worden, moet eerst zoals hierboven beschreven de juiste board geselecteerd worden, waarna bij programmer USBtinyISP geselecteerd moet worden. Sluit vervolgens de UsbTiny aan op de BabyShell met de 6 pins kabel. Sluit vervolgens de UsbTiny aan op de computer. Vervolgens drukt men op bootloader branden.

A.2. ButtonPINFalling

```
void ButtonPINFalling () {
 ADCSRA = 135; //setting the ADC back in running modus after sleeping

  //if wake up from sleep, with screen off, check for battery levels
  if (INTERRUPT_TYPE == 0)
  {
    powerMode();
  }

#ifdef solarVersion
```

```
  //if the battery levels are too low, we don't wake up from sleep.
  if (battoolow == 1)
  {
    return;
  }
#endif
  analogReference(DEFAULT);
  INTERRUPT_TYPE = 1;
  PCintPort::detachInterrupt(ButtonPin);
  PCintPort::detachInterrupt(RTCPIN);

  unsigned int a;
  for (byte i = 0; i < 3; i++)
  {
    a = analogRead(ButtonPin);// to skip false values;
  }

  a = 0;
  for (byte i = 0; i < 3; i++)
  {
    //to make the measurement more accurate we average over 3 measurements
    a += analogRead(ButtonPin);
  }
  a /= 3;

  if (a > 385 && a < 406 ) //12 k resistor
  {
    ButtonHomePushed = 1;
  }
  else if (a > 369 && a < 385 ) //11 k resistor
  {
    ButtonBackPushed = 1;
  }
  else if (a > 345 && a < 369 ) //10 k resistor
  {
    ButtonSelectPushed = 1;
  }
  else if (a > 320 && a < 345 ) //9 k resistor
  {
    ButtonBottomPushed = 1;
  }
  else if (a > 280 && a < 320 ) //8 k resistor
  {
    ButtonTopPushed = 1;
  }
  else if (a < 280 )
  {
    //TWO buttoms pushed
  }
  else if (a > 406) {
    // no buttum pushed or not long enough
  }
}
```

A.3. LcdString

```
void LcdString(char *characters, boolean invert)
{
  byte AmountOfCharsWritten = 0;
  byte  AmountOfCharsOnLine = 14;
  while (*characters)
  {
    AmountOfCharsWritten++;
    LcdCharacter(*characters++, invert);

  }
  if (AmountOfCharsWritten > 14) {
    /* then the line is longer than the space reserved.
    Switching to two lines to print this answere.
    So need to add spaces until 28 characters */

    AmountOfCharsOnLine = 28;
  }

  while (AmountOfCharsWritten < AmountOfCharsOnLine) {
    /*to fill the rest of the line with spaces.
    so that the following line prints on the next line. */

    if (invert == HIGH) {
      LcdWrite(LCD_D,  0xff);
      for (int index = 0; index < 5; index++)
      {
        LcdWrite(LCD_D,  ~ASCII[0][index]);
      }
      AmountOfCharsWritten++;
    }
    else {
      LcdWrite(LCD_D,  0x00);
      for (int index = 0; index < 5; index++)
      {
        LcdWrite(LCD_D,  ASCII[0][index]);
      }
      AmountOfCharsWritten++;
    }
  }
}

void LcdCharacter(char character, boolean invert)
{
  if (invert == 1)
  {
    LcdWrite(LCD_D, 0xff);
    for (int index = 0; index < 5; index++)
    {
      LcdWrite(LCD_D, ~ASCII[character – 0x20][index]);
    }
  }
  else
  {
```

```
    LcdWrite(LCD_D, 0x00);
    for (int index = 0; index < 5; index++)
    {
      LcdWrite(LCD_D, ASCII[character – 0x20][index]);
    }
  }
}
```

A.4. printScreen

```
void printScreen() {
  char screenBuffer[12][15] = {};
  byte screenNumber = 1;
  byte amountOfScreens = 0;
  byte amountOfAnsweres = 0;
  boolean answered = 0;

  //make temperorarely Download buffer
  char * downloadBuffer;
  downloadBuffer = (char*)calloc( 129, 1);

  //download data from eeprom to buffer;
  ReadEEPROM(this->address, downloadBuffer);
  // devide tekst into multiple lines in ScreenBuffer
  amountOfScreens = PrepareScreenBuffer(downloadBuffer, screenBuffer);

  free(downloadBuffer); //delete download buffer. not needed anymore.

  if (customAnswer == 0) {
    amountOfAnsweres = 2;
  }
  else {
    amountOfAnsweres = customAnswer;
  }

  //process Buttons
  while (answered == LOW) {
    if (ButtonTopPushed == HIGH) {
      if (screenNumber > 1) {
        screenNumber--;
      }
      ButtonTopPushed = LOW;
    }
    else if (ButtonBottomPushed == HIGH) {
      if (screenNumber < (amountOfScreens  + amountOfAnsweres)) {
        screenNumber++;
      }
      ButtonBottomPushed = LOW;
    }
    if (ButtonHomePushed == HIGH) {
      ButtonHomePushed = LOW;
      return;
    }
    if (ButtonBackPushed == HIGH) {
      ButtonBackPushed = LOW;
```

```
    }
    if (screenNumber <= amountOfScreens) { // print question screen
      if (ButtonSelectPushed == HIGH) {
        ButtonSelectPushed = LOW;
      }
      LcdClear();
      for (byte i = (6 * screenNumber - 6); i < (6 * screenNumber); i++) {
        LcdString(screenBuffer[i], 0);
      }
      sleepNow();
    }
    else //print answeres
    {
      if (ButtonSelectPushed == HIGH) {
         // saves selected answer
        setChosenAnswer( screenNumber - amountOfScreens - 1);
        ButtonSelectPushed = LOW;
        return;
      }
      else {
        LcdClear();
        if (address==(5)||address==(6)||address==(7)||address==(16)||
            address==(17)||address==(18)||address==(19)||address==(20)||
            address==(21)||address==(22)||address==(28)||address==(29)||
            address==(30)||address==(31)) {
          //the addresses of the outcomes, they shouldn't print
          //the possible answers but just go to the next question
          //to see which addresses match which questions and diagnoses,
          //refer to file "burn_tekst"
          return;
        }
        else {
          printAnswer(screenNumber - amountOfScreens - 1);
          sleepNow();
        }
      }
    }
  }
}
```

A.5. PrepareScreenBuffer

```
byte PrepareScreenBuffer(char* Buffer, char screenBuffer[12][15]) {
  byte bufferLines = 0;
  byte amountOfScreens = 0;

  bufferLines = WriteToBuffer(Buffer, screenBuffer, 0, 0);
  // split buffer into many screendisplayline buffers
  bufferLines++;

  while (bufferLines > 6) {
  //calculate the amount of screens(thus amount of multiplies of 6 lines)
    bufferLines =  bufferLines - 6;
    amountOfScreens++;
```

```
  }
  amountOfScreens++;
  return amountOfScreens;
}
```

A.6. WriteToBuffer

```
byte WriteToBuffer(char* CharArray, char screenBuffer[12][15],
                   byte previousLastSpace, byte bufferLineNumber)
{
  byte lastSpace = 0;
  boolean End = 0;
  byte i = 0;
  byte k = 0;
  boolean flagLongWord = 0;


  for (i = previousLastSpace; i < previousLastSpace + 15; i++)
  // if character is a blank space or ‘\n‘ or ‘\0‘ then
  // the lastSpace is set at the number of the character in the line.
  {
    if (CharArray[i] == ' ')
    {
      lastSpace = i;
    }
    if (CharArray[i] == '\n' || CharArray[i] == '\0') // \0
    {
      End = 1;
      lastSpace = i;
      break;
    }
  }
  if (lastSpace == 0)
  // if the word is longer than 14 characters, flagLongWord is set high
  {
    lastSpace = previousLastSpace + 14 ;
    flagLongWord = 1;
  }

  screenBuffer[bufferLineNumber];
  for (i = previousLastSpace; i < lastSpace ; i++)
  // filling screenbuffer line:bufferLineNumber with the text out of
  //CharArray until the next word doesn't fit anymore
  {
    screenBuffer[bufferLineNumber][k] = CharArray[i];
    k++;
  }


  for (i = k; i < 14 ; i++) // fill the rest of the line with blank spaces
  {
    screenBuffer[bufferLineNumber][i] = ' ';
  }
```

```
  if (End == 0 )
  // when the end of buffer is not detected in this line, continue filling
  // screenbuffer with new lines until all text in buffer has been used
  {
    if (flagLongWord == 0) {
      bufferLineNumber = WriteToBuffer(CharArray, screenBuffer,
                                ++lastSpace, ++bufferLineNumber);
    }
    else {
      bufferLineNumber = WriteToBuffer(CharArray, screenBuffer,
                                lastSpace, ++bufferLineNumber);
    }
  }

  return bufferLineNumber;
}
```

A.7. ATmega Bootloader

```
atmega328bb.name=ATmega328 on a breadboard (8 MHz internal clock)
atmega328bb.upload.tool=avrdude
atmega328bb.upload.protocol=arduino
atmega328bb.upload.maximum_size=30720
atmega328bb.upload.speed=57600

atmega328bb.bootloader.tool=arduino:avrdude
atmega328bb.bootloader.low_fuses=0xE2
atmega328bb.bootloader.high_fuses=0xDA
atmega328bb.bootloader.extended_fuses=0x05
atmega328bb.bootloader.path=arduino:atmega
atmega328bb.bootloader.file=ATmegaBOOT_168_atmega328_pro_8MHz.hex
atmega328bb.bootloader.unlock_bits=0x3F
atmega328bb.bootloader.lock_bits=0x0F

atmega328bb.build.mcu=atmega328p
atmega328bb.build.f_cpu=8000000L
atmega328bb.build.core=arduino:arduino
atmega328bb.build.variant=arduino:standard
```

Bibliografie

[1] Unicef. Child survival under-five mortality. http://data.unicef.org/child-mortality/under-five.

[2] WHO. Maternal mortality. http://www.who.int/mediacentre/factsheets/fs348/en/.

[3] Vital Wave Consulting. mhealth for development. the opportunity of mobile technology for health-care in the developing world. United Nations Foundation.

[4] The World Bank. Mobile cellular subscriptions (per 100 people). http://data.worldbank.org/indicator/IT.CEL.SETS.P2.

[5] International Energy Agency. Energy access database. http://www.worldenergyoutlook.org/resources/energydevelopment/energyaccessdatabase/.

[6] Babycentre. Pregnancy calendar. http://www.babycentre.co.uk/pregnancy-calendar.

[7] CDC. 2015 recommended immunizations for chichild from birth through 6 years old. http://www.cdc.gov/vaccines/parents/downloads/parent-ver-sch-0-6yrs.pdf.

[8] Lars van Leeuwen; Tim Cheung. Bachelor afstudeer project: Voeding, batterij management en pcb ontwerp. Technical report, TU Delft, 2015.

[9] Gerard Hogenhout; Kees Hogenhout. Bachelor afstudeer project: Rtc en temperatuursensor. Technical report, TU Delft, 2015.

[10] Ole F Norheim *et al.* Avoiding 40% of the premature deaths in each country, 2010–30: review of national mortality trends to help quantify the un sustainable development goal for health. *Lancet*, 385:239–252, 2015.

[11] Joses Muthuri Kirigia *et al.* Indirect cost of maternal deaths in the who african region in 2010. *BMC Pregnancy and Childbirth*, 14:299, 2014.

[12] Sanjana Gupta *et al.* Morbidity among under five children in a rural area of jammu. *JK SCIENCE*, 14:85–88, 2012.

[13] Li Liu *et al.* Global, regional, and national causes of child mortality in 2000–13, with projections to inform post-2015 priorities: an updated systematic analysis. *Lancet*, 385:430–440, 2015.

[14] Jean-Modeste Harerimana *et al.* Effect of shortened integrated management of childhood illness training on classification and treatment of under-five children seeking care in rwanda. *Risk Management and Healthcare Policy*, 7:99–104, 2014.

[15] Texas Instruments. Msp430f2272 datasheet. http://www.ti.com/lit/ds/symlink/msp430f2272.pdf.

[16] Atmel. Atmega328p datasheet. http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf.

[17] Atmel. Atmega2560 datasheet. http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf.

[18] Aliexpress. Atmega328p prijs. http://nl.aliexpress.com/item/VICKO-ATmega328P-AU-ATMEGA328-ATMEGA328P-TQFP-32-ATMEL-MCU-32KB-In-system-Flash-20MHz-1/1913443437.html.

[19] Aliexpress. Atmega2560 prijs. http://nl.aliexpress.com/item/Free-Shipping-10pcs-lots-ATMEGA2560-16AU-ATMEGA2560-ATMEL-TQFP100-new-originaland-rohs/32368142213.html.

[20] Texas Instruments. Msp430 prijs. http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/ultra-low_power/msp430f2x_msp430f4x/products.page#p1130=0.46;4.3&p3100=32;120&p2734=1;8&p299=1;4&p886=24;80.

[21] Arduino. Reference. http://www.arduino.cc/en/Reference/HomePage.

[22] Energia. http://energia.nu/faqs/.

[23] Philips. Pcd8544 datasheet. https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf.

[24] Hitachi. Hd44780u datasheet. https://www.sparkfun.com/datasheets/LCD/HD44780.pdf.

[25] Paul Bakker; Tim Cheung; Gerard Hogenhout; Kees Hogenhout; Alexander Jongeling; Lars van Leeuwen. Bachelor afstudeer project: Literature study. Technical report, TU Delft, 2015.

[26] Microchip. 24aa256/24lc256/24fc256 256k i2c™ cmos serial eeprom. http://ww1.microchip.com/downloads/en/DeviceDoc/20001203U.pdf.

[27] Atmel. Datasheet at24c256. http://www.atmel.com/Images/doc0670.pdf.

[28] Nick Gammon. Power saving techniques for microprocessors. http://www.gammon.com.au/power.

[29] Arduino. Compare board specs. http://www.arduino.cc/en/Products.Compare.

[30] Arduino. From arduino to a microcontroller on a breadboard. http://www.arduino.cc/en/Tutorial/ArduinoToBreadboard.

[31] Engbedded. Engbedded atmel avr®fuse calculator. http://www.engbedded.com/fusecalc.

[32] Arduino. Analogreference. https://www.arduino.cc/en/Reference/AnalogReference.

[33] World Health Organization. *IMCI Chart Booklet*. World Health Organization, 20 Avenue Appia, 1211 Geneva 27, Switzerland, 2014.

[34] CYBERGIBBONS. Arduino misconceptions 7: boolean variables are a native c/avr type. http://cybergibbons.com/arduino/arduino-misconceptions-7-boolean-variables-are-a-native-cavr-type/, April 2013.

[35] kuk; Sylvain Bissonnette. Philips pcd8544 (nokia 3310) driver. http://playground.arduino.cc/Code/PCD8544.

[36] Neil McNeight. Memoryfree library. https://github.com/McNeight/MemoryFree.

[37] Mike Schwager; Chris J. Kiick. Pinchangeint library. https://github.com/GreyGnome/PinChangeInt.