Implementering Softdrink-Automat

Fag:	Projekt – E1PRJ1
Emne:	Implementering – Softdrink-Automat
Gruppe:	6
Dato:	20. marts 2006
Medlemmer:	Benjamin Sørensen, Jacob Nielsen, Klaus Eriksen, Mikkel Larsen og Tomas Stæhr Hansen

Indholdsfortegnelse

1. Indledning

1.1 Formål

Formålet med denne rapport er, at dokumentere implementationen. Først vil vi beskrive opbygningen af rapporten.

Vi starter med realisering og test af hardwaren, afsnit 2. Derefter beskrives realisering og test af softwaren, afsnit 3. Afsnit 4 omhandler integrationen mellem hardwaren og softwaren. Afsnittet gennemgår hvordan hardwaren og softwaren er koblet sammen til det endelige produkt. Vi kommer også ind på hvordan produktet er testet og finjusteret.

1.2 Referencer

Dokumenter:
Kravspecifikation
Struktureringsrapport
T-136: Softdrink-Automat – Tekniske Specifikationer til doseringsmekanik

1.3 Revisionshistorie

11. juni 2003 revisionsnr.: 1.0

1.4 Bilag

Bilag 1: Komponentplacering
Bilag 2: Motorberegninger
Bilag 3: Kode
Bilag 4: Test af HWInterface

2. Hardware

2.1 Indledning

I dette afsnit vil vi se nærmere på hvordan vi har opbygget de forskellige kredsløb, og hvordan komponenter er valgt.

For at få automaten til at kommunikere med software i PC'en, var det nødvendig at lave et interface mellem automaten og I/O-kortet.

Veroboardet var givet på forhånd og er et hulprint, som har forbindelse til både automat og PC.

Kravet til veroboardet var at komponenterne skulle monteres med wirewrap. Ud over dette har vi bestemt at der ikke skulle tilsluttes ekstern spændingsforsyning til veroboardet, og at vi bruger positiv logik.

Vi har delt veroboardet op i fire hovedpunkter:

- Væskemålere
- Motorstyring
- Temperaturføler
- Knapper + dioder

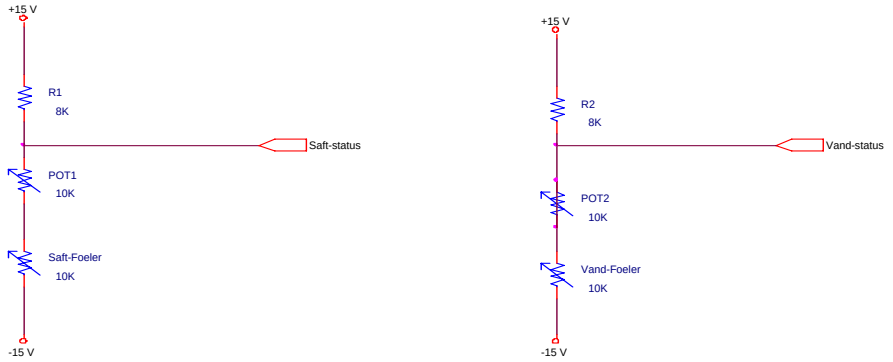
Disse punkter vil bliver uddybet i efterfølgende afsnit.

Værktøjer:

- Wirewrap udstyr
- Komponenter
- Ledninger
- Måleudstyr
- PC med I/O-kort

2.2 Implementering

2.2.1 Væskeføler



Figur 1: Kredsløb for hhv. saft- og vandmåler.

Beskrivelse

På automaten sidder to væskemålere. En til saft og en til vand. Disse målere er hver opbygget af et 10 kΩ potmeter. Da I/O-kortet kun kan aflæse en spændingsændring på dens A/D indgange, bliver vi nødt til at konvertere modstandsændringen om til en spændingsændring.
Vi har valgt at lave konverteringen med en simpel spændingsdeler. Det har vi gjort da føleren ikke er så følsom for belastning, og at målingen ikke behøver at være præcis.

Beregninger

Udsvinget på følerne er på max 2 kΩ. De to yderpunkter for følerne er derfor 0-2 kΩ, og 8-10 kΩ. Ved hjælp af potmetrene POT1 og POT2 (se figur 1), kan vi justere kredsløbet sådan at den samlede modstandsændring POT + Føler altid vil svinge i området 8-10 kΩ. Derfor har vi valgt at have en top modstand R1 og R2 på 8 kΩ. Outputtet vil nu svinge fra 0 V til 1.667 V.

Outputtet er givet ved følgende beregning: $\frac{30V}{8k\Omega + 10k\Omega} * 10V - 15V = 1.667V$

Dette ser måske ikke ud af meget, da A/D indgangen på indstikskortet, står til at modtage 0-5 V. Men med en 12 bit konverter, har man 4096 step fra 0-5 V, hvilket giver 4096*(1.667 V / 5 V) ca. 1365 step fra 0 - 1.667 V. Og 1365 step er rigeligt i denne sammenhæng.
Vi hævede dog intervallet via POT1 og POT2 sådan at spændingen går fra 1 - 2.667 V. Dette skyldes, at hvis en af målerne får skiftet sit interval område f.eks. pga. slitage, og spændingen som følge af dette falder til under 0 V på outputtet, vil vi ikke kunne registrere det. A/D konverteren er indstillet til området 0-5 volt.

2.2.2 Motor

Beskrivelse

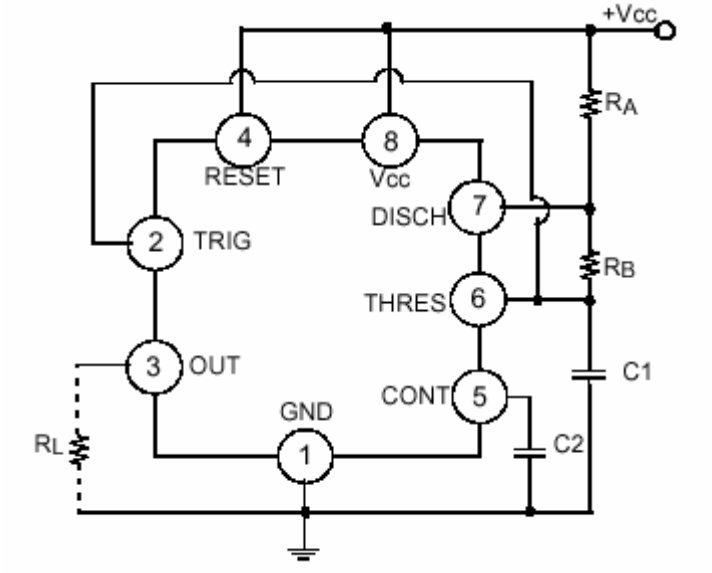
Den elektro-mekaniske konstruktion af koptransporten er en sammenføjning af følgende funktionelle enheder:

1. En kopbærende enhed bestående af en platform og en bøjle.
2. En transport-orienterende enhed bestående af hjul og skinner
3. En krafttransmitterende enhed som er en skruespindel med en stigning på 5 mm/omdrej.
4. En drivende enhed som er en stepmotor (SAIA UHD23/45 12Volt) med 7,5 grader pr. step. Den maximale start/stop stepfrekvens angives i T-136 til at være forventelig 100 step/sek. Motoren er delt i en motor A og motor B som forsynes med hver sit styresignal. Motorens retning styres ved at vende fasen på styresignalerne + / - 90 grader.
5. En drivkraftsregulerende enhed som er et full-step driverkredsløb der ikke kan sættes højimpedant, dvs. det hele tiden trækker strøm.

Indledende overvejelser

Koptransporten:
Koptransporten byder ikke på særlige vanskeligheder. Der er ikke den store enerti at overvinde ved start og stop så motoren kan sættes i gang med konstant styrefrekvens. Accelereret start (softstart) øger imidlertid driftstabiliteten, mindsker peak-effekt og giver mulighed for at opnå mindre transporttider eller mulighed for at anvende en spinklere (og billigere) motor.

Clockgenerering:
Valget faldt på kredsen NE 555. Figur 2 viser hvorledes kredsen kan anvendes som astabil multivibrator.

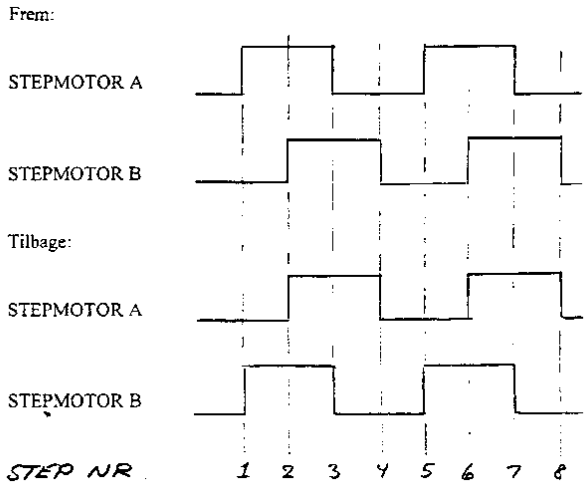


Figur 2: NE555 forbundet som astabil multivibrator med fast trigger/treshold niveau.

Opbygning af styresignal

Det ses af Figur 3 at motorsignalerne kan kodes således (retning → motor A, motorB)

Frem:	00 →	10 →	11 →	01 → etc.
Tilbage:	00 →	01 →	11 →	10 → etc.



Figur 3: Krævede styresignaler til kontrol af motor reproduceret fra T-136.

Der er to tilstandsvariable og der er anvendt Gray-kodning. Vi valgte at anvende en PEEL nemlig 22CV10 som vi kender fra undervisningen i DTM1. ABEL-koden findes på næste side.

```
MODULE motorstyring
TITLE 'motorstyring'

declarations
CLOCK,RUN,UP pin 1,2,3;                          "RUN=START, UP=FREM

A,B                     pin 18,19 istype 'reg';
FREM,TILBAGE,BIAS       pin 20,21,22 istype 'com';

c,x=.c.,.x.;

SSTATE      = [A,B];
S0          = [0,0];
S1          = [1,0];
S2          = [1,1];
S3          = [0,1];

Equations

SSTATE.clk=CLOCK;

BIAS = !RUN;

state_diagram SSTATE

State S0: IF !RUN THEN S0
            ELSE IF UP THEN S1
                        ELSE S3;
state S1: IF !RUN THEN S1
            ELSE IF UP THEN S2
                        ELSE S0;
state S2: IF !RUN THEN S2
            ELSE IF UP THEN S3
                        ELSE S1;
state S3: IF !RUN THEN S3
            ELSE IF UP THEN S0
                        ELSE S2;

Truth_table  ([RUN,UP]    -> [FREM,TILBAGE])
            [0,0]         -> [0,0];
            [0,1]         -> [0,0];
            [1,0]         -> [0,1];
            [1,1]         -> [1,0];


test_vectors // udeladt her!

END combi
```

Forklaring til ABEL koden

Input:

CLOCK, RUN og UP	Henholdsvis Clock, signal til motorstart (positiv logik) og retningssignal. RUN er forsynet med en 100 kΩ pull-down modstand for at forhindre utilsigtet igangsætning.

Output:

A,B	Styresignal til motor A og motor B
FREM,TILBAGE	Signaler til dioderne hhv. frem og tilbage på styringspanelet BIAS Går lav når motoren startes og styrer kontrolkredsløbet som beskrives i det følgende.

softstart

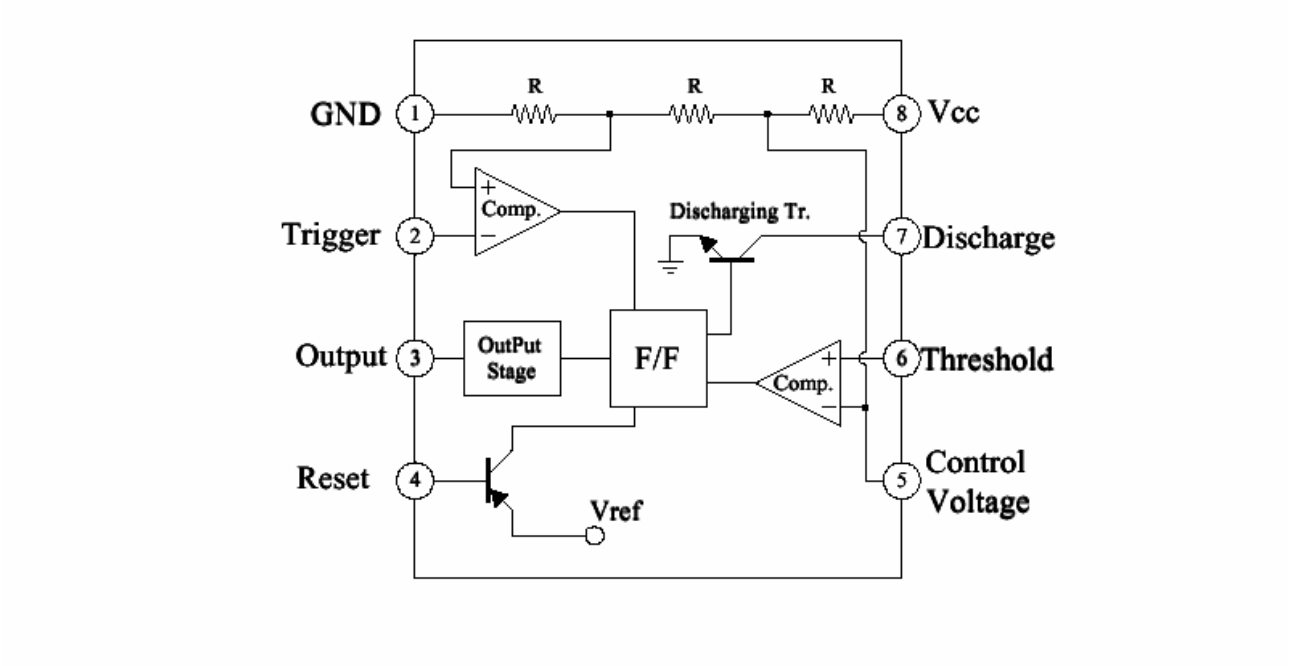
En stepmotor mister steps hvis trækkraften er for lav i relation til modstanden i systemet. Modstanden er summen af friktion og enerti. Hvis motoren sættes i gang med en høj frekvens er der risiko for at motoren "staller" pga. enertien. I praksis viste det sig at enertien var af lille betydning idet forskellen mellem den maksimale startfrekvens og den højeste stabile "march-frekvens" var minimal. Vi valgte alligevel at realisere et sparsomt accelerationskredsløb.

Digital Softstart

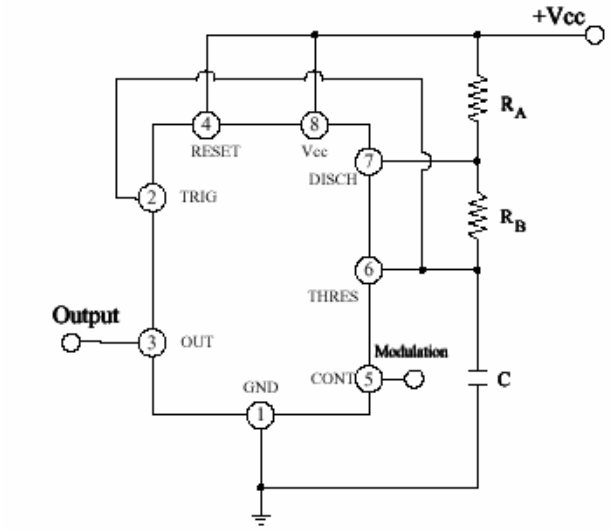
Ideen var at neddele en højfrekvent clock efter en tabel tilpasset det bevægelige system, dvs. tilpasset motorens moment ved forskellige hastigheder, elementer af inerti og friktion samt slip i transmissionen. Dette system kunne kodes ind i to 22CV10'ere styret af to forskellige clock's. Den ene kreds "tællerkredsen" kunne clockes med konstant høj frekvens og være brændt med en 8 bit tæller med preset og RCO (Ripple carry out). Hvis denne kreds eksempelvis presettes til at tælle ned fra 127 kan RCO'et fra 0 (00000000) bruges som clocksignal. Hvis kredsen herefter presettes med 89, 71, 60…..30,30,30.. så har man en accelereret clock. Den anden kreds "signalkredsen" kunne indeholde kodningen af motorsignalerne samt en mindre tæller og et register til presettet af tællerkredsen foruden start/stop og frem/tilbage funktionaliteten.
Men det blev for omfattende for en overflødig funktion.

Analog softstart

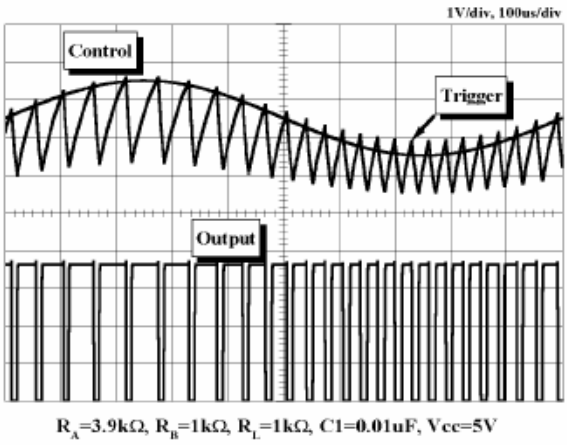
NE555 giver mulighed for at variere pulslængden ved at påtrykke en kontrolspænding over den indbyggede spændingsdeler. Dette giver mulighed for at lade clockfrekvensen følge eks. opladningen af en kondensator ved motorens igangsættelse. Ved at påtrykke en kontrolspænding (control voltage, pin 5, figur 4) flyttes på både trigger- og thresholdniveauerne på de to indbyggede komparatorer. De tre modstande R i kredsen har en målt seriemodstand på ca. 12 kΩ dvs. 3 * 4 kΩ. Spændingen på pin 5 er derfor ubelastet 2/3 af Vcc. Ved påtrykning af en lavere spænding stiger clockfrekvensen fordi afstanden mellem trigger og threshold formindskes hvorved opladningstiden formindskes. Ved højere spænding sker i analogi hermed det omvendte. Påtrykning af 0 V kræver en strøm på 5 V / 4 kΩ ca. 1,2 mA . Denne strøm kan leveres af en operationsforstærker brugt som spændingsfølger.



Figur 4: Internt blokdiagram for NE555.



Figur 5: NE 555 som astabil multivibrator med modulerende input.



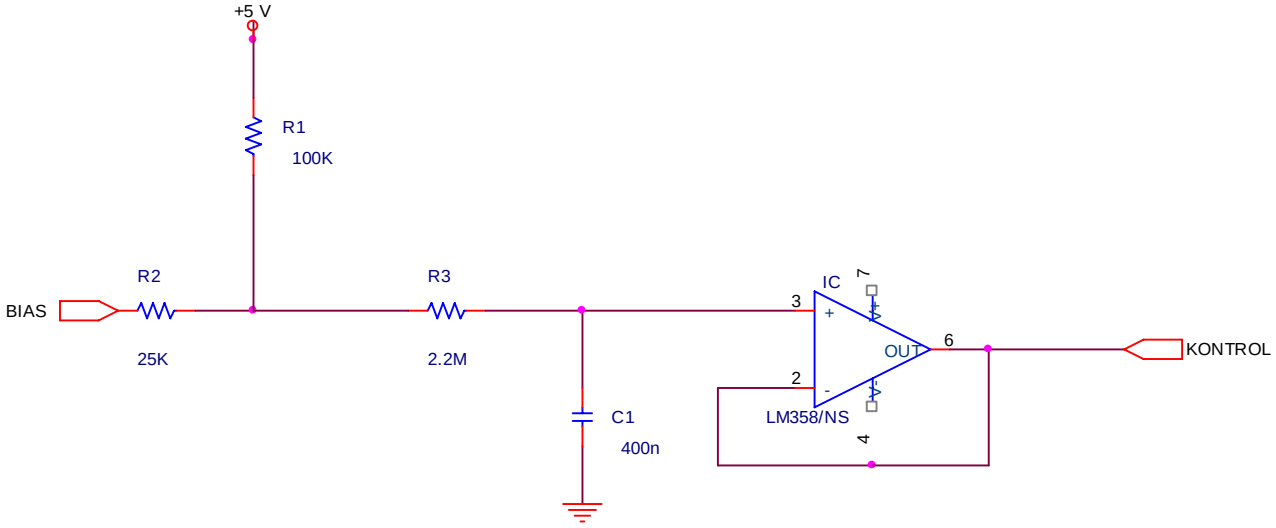
Figur 6: Øverst: Kontrolspænding. Nederst: Det resulterende output .

Opbygning af kontrolkredsløb

Ved kontrolspænding = Vcc bliver mulitvibratorens opladningstid uendelig. Dermed bliver clockfrekvensen uendelig høj. Vi sætter maksimum kontrolspænding til Vcc dvs. 5 V. Når kontrolspænding går mod 0 V, dvs. multivibratorens jord, går opladningstiden mod 0 og periodetiden går således mod afladningstiden som alene afhænger af R_B. Vi vælger 1 V som minimum kontrolspænding for at få en passende bred puls. Spændingsdeleren 25kΩ/100kΩ sørger for at spændingen foran 2.3 MΩ modstanden bliver 1/5 Vcc ved 0 V på BIAS. BIAS (pin 22 på

22CV10 kredsen) er den inverterede af det høj-logik signal der starter motoren. Når motoren gives startsignalet falder kontrolspændingen i takt med kondensatorens afladning. Operationsforstærkeren er forbundet som spændingsfølger og skal kun levere strøm.

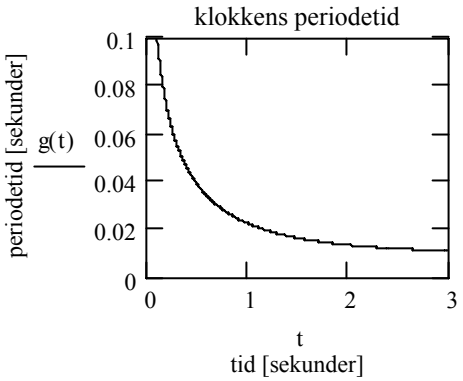
Beregninger på kredsløbet findes i bilag 2. Det skal bemærkes at pludselige retningsskift under 1 sekund efter stop må undgås idet omladningen af kredsløbet tager ca. 1 sekund og motoren derfor vil starte ved en højere hastighed.



Figur 7: Kredsløb til opbygning af kontrolspænding til astabil multivibrator.

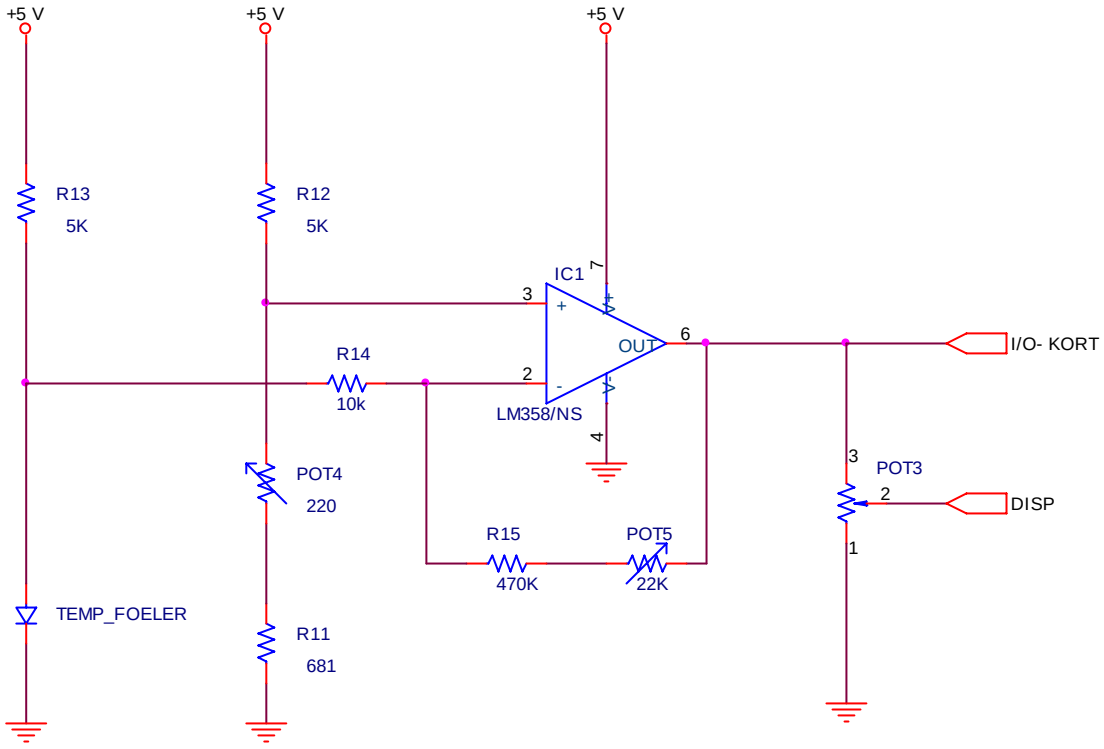
Valg af komponenter til astabil multivibrator

Modstandende R_A og R_B (figur 5) er valgt så duty cycle bliver omtrent 50:50 ved kontrolspænding 1 V, dvs. ved marchhastighed. Dette er egentlig ikke påkrævet. Det væsentlige er, at R_B er tilpas lav så afladningen af kondensatoren er så hurtig at kontrolfunktionen har betydning. Beregninger findes i bilag 2.



Figur 8: Klokkens periodetid som funktion af tid fra motorstart.

2.2.3 Temperatur



Figur 9: Kredsløb for temperaturføleren

Beskrivelse

Automaten har kun en diode som temperaturføler. Det er derfor krævet at lave et kredsløb som registrerer ændringerne i dioden og konverter dem til spændingsændringer. Vi sender derefter disse spændingsændringer videre til A/D konverteren på I/O-kortet i pc'en og displayet på automaten.

Beregninger

Føleren består af en helt almindelig diode 1N4148. Dioden har den egenskab at spændingen over den ændrer sig lineært med ca. -2mV per grad celsius. Denne spændingsændring fremkommer dog kun hvis der påføres en strøm, derfor R13. Vi har set på karakteristikken for dioden, og fandt arbejdspunktet på 1 mA passende. Vi har derfor valgt R13 værdi til 5 kΩ da (5 V - 0.6 V)/5 kΩ ≈ 1 mA.

Dioden er meget følsom, og at sende denne spændingsændring direkte over i A/D konverteren på indstikskortet, ville påvirker resultatet. Vi har derfor valgt at bruge en standard opsætning med en operationsforstærker.

Dioden giver 2 mV ændring per grad, og displayet viser 1 grad per 10 mV. Vi er derfor nødt til at forstærke signalet 5 gange. Dette signal skal også føres over til A/D konverteren på I/O-kortet, men da dette kun vil gå fra 0-300 mV kunne man efterfølgende forstærke signalet for at få en bedre

opløsning. Det vil sige først forstærke signalet med 5, hvorefter det bliver forstærket igen. Dette kan dog laves smartere. Vi har valgt kun at forstærke signalet en gang, og føre dette signal over til A/D konverteren. Ud over dette bliver dette signal spændingsdelt, så displayet får en spænding på 10 mV per grad.

Vi har valgt at have et 1:10 forhold mellem det vi sender til displayet og A/D konverteren. Dette betyder vi skal have en forstærkning på 50 gange. Forstærkningen bestemmes med forholdet mellem POT5+R15 og R14 (Figur 9). Potmetret er isat grundet justering af forstærkningen da flere af komponenterne har en tolerance på 5 %. Vi har valgt ca. 500 kΩ til 10 kΩ ohm.

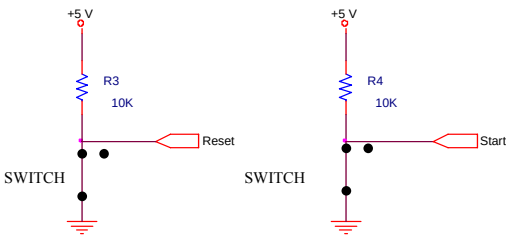
Da vi bruger en operationsforstærker har vi mulighed for at sætte et offset. Offset betyder hvilken startværdi outputtet skal have. Vi ønsker at outputtet på operationsforstærkeren, er lineært med temperaturen og at 1 grad svarer til 100 mV. Da spændingen over dioden er ca. 0,6 V, skal vi have ca. 0,6 volt på operationsforstærkerens andet input. Modstanden R12's værdi er valgt således at den skal strøm begrænse, og 5 kΩ ohm giver 1 mA, hvilket er tilladeligt. Vi har derefter valgt at tage en 680 Ω modstand R11 i serie med et potmeter på 220 (POT4). Derved kan vi kalibrere offsetværdien præcist. R11 er givet ved:

$$\frac{5V}{5k\Omega + X} * X = 0{,}6V \Rightarrow X = 682\Omega$$

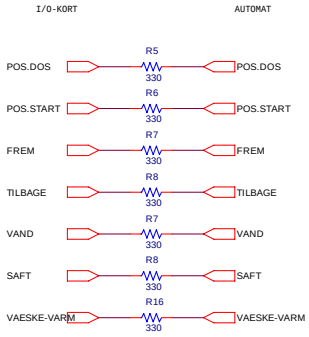
Outputtet fra operationsforstærkeren bliver nu sendt over til A/D konverteren på I/O-kortet, samt delt med en spændingsdeler bestående af et potmeter POT3 på 10 kΩ, sådan at de to output præcis er i forholdet 1:10. Værdien på 10 kΩ er valgt med henblik på at den skulle være så stor den ikke trak for meget strøm, og alligevel så lille at værdierne ikke bliver trykket ned når outputtene bliver belastede.

2.2.4 Knapper + dioder

Knapper:



Dioder:



Figur 10: Kredsløb for hhv. Knapper og dioder

Beskrivelse

Figur 10 viser kredsløbene for knapper og dioder. Disse kredsløb er vigtige men ikke store nok til at få deres eget afsnit.

Beregninger

Knapper

Da vi valgte at bruge positiv logik, og kontakterne start og reset er fast forbundet til stel, var mulighederne begrænsede. Når knapperne ikke er aktiveret er kontakterne sluttet til stel, og 0 V sendes videre. Ved aktivering af knapperne afbrydes kontakterne og outputtet er nu 5 V. Ulempen ved denne opstilling er at der løber strøm når knapperne ikke er aktiveret, men vi har valgt formodstande på 10 kΩ hvilket giver en strøm på (5 V / 10 kΩ) = 0,5 mA, som er acceptabelt.

Lysdioder

Lysdioderne bliver styret af logiske output, som er ca. 5 V. En lysdioder bruger kun ca. 2 V, og vil prøve at trække de 5 V ned, ved at trække en stor strøm. Dette vil belaste den kreds der giver outputtet, og måske brænde lysdioden af. En lysdiode lyser fint med 10 mA. Dette betyder vi skal have en formodstand på (3V / 10 mA) = 300 Ω. Vi har valgt 330 Ω for at være på den sikre side.

2.3 Grænseflader

Grænsefladerne mellem hardwaren og omverdenen er beskrevet i struktureringsrapport bilag 1.

2.4 Komponentplacering

Komponentplacering

Se bilag 1

Komponentliste

R1-R2	8 kΩ
R3-R4	100 kΩ
R5-R10	330 Ω
R11	681 Ω
R12-R13	5 kΩ
R14	10 kΩ
R15	470 kΩ
R16	330 Ω
R17	10 kΩ
R18	100 kΩ
R19	100 kΩ
POT1-POT3	10 kΩ
POT4	220 Ω
POT5	22K Ω
POT6	470 kΩ
C1-C2	10 nF
IC1	LM358

IC2	22V10
IC3	LM555

2.5 Modultest

Væskefølere

1. Kalibrere outputtet til at give 0 V ud ved minimum væske højde (Senere ændret til en 1 V, dog uden modultest).
2. Hæver væskestanden til maksimum væske højde, aflæser spændingsændringen.

Motoren

Ved lave steprater bevæger motoren langsomt, hvilket forøger softdrink-rutinens varighed; ved høje steprater falder motorens moment og der opstår risiko for stop af koptransport. Clockfrekvensen justeres ved at trimme potentiometer 6 (bilag 1: pot6) så motorens steprate ligger i det ovenfor skitserede interval.

Temperatur

1. Spændingen over dioden er 0,6 V.
2. Spændingen ændrer sig lineær med 2 mV per grad.
3. Offset indstilles til 0,6 V
4. Forstærkningen er på 50 gange
5. Kalibrere output til at være 0 V ved 0 grader celsius.
6. Sætte temperaturen til 20 grader celsius og kalibrere forstærkningen præcis.
7. Indstille spændingsdeleren til at give forholdet 1:10.

Knapper + dioder
Pga. disse modulers enkle opbygning udføres ikke tests på dem.

3. Software

3.1 Indledning

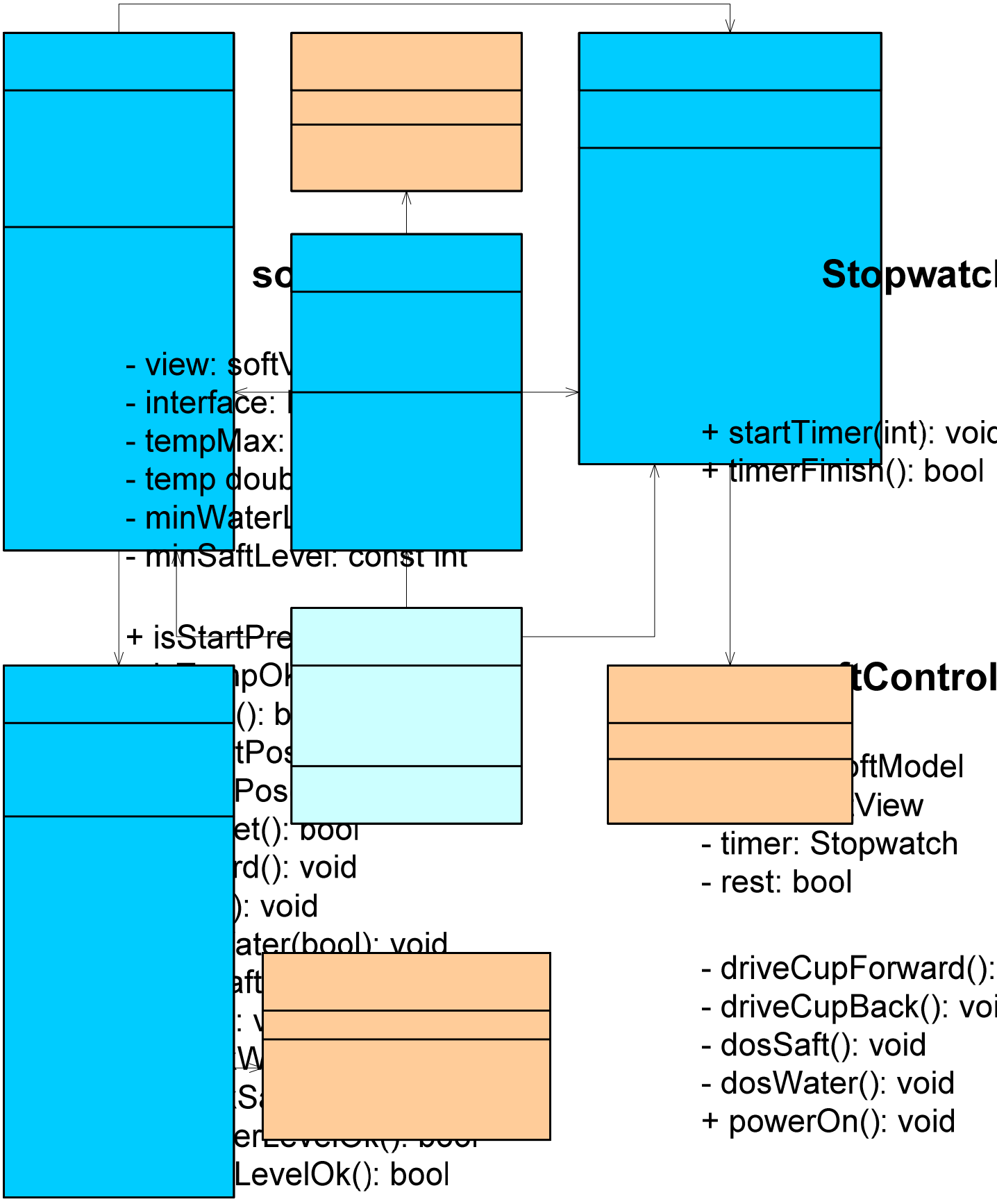
Implementeringen af softwaren er foretaget i C++.
Vi har anvendt følgende programmer:

- MS Visual Studio.net
- MS C/C++ compiler
- Emacs
- Gnu C/C++ compiler

Koden er compilet med MS C/C++ compiler. Da vi i softView bruger klassen consoleGraphic, som inkludere windows.h, så kræves en MS compiler. Programmet kan derfor kun køre på en Windows maskine.

3.2 Omstrukturering

Undervejs i implementeringsforløbet har vi foretaget en række ændringer i forhold til det, vi lagde os fast på i struktureringsfasen. Ændringer kan være tilføjelse af ny funktionalitet, flytning af funktionalitet mellem klasserne eller fjernelse af overflødig funktionalitet.
Vi har derfor måtte udarbejde et nyt klassediagram.



Figur 11: Klassediagram MVC

Vi vil nu gennemgå de enkelte ændringer.

3.2.1 Ændringer

softController:
Tilføjet:
- Bool reset, blivet sat når man trykker reset, og bliver nulstillet når den afbrudte softdrinkrutine afslutter.
- Timer, bruges under dosering. Sørger for at vi doserer i et præcist antal milisekunder.

softModel:
Fjernet:
- Bool reset, er flyttet til softControlleren
- Bool start, er helt fjernet.

Tilføjet:
- Double temp: indeholder temperaturen i grader celsius
- Const int minWaterLevel, sætter minimum grænsen for vand level
- Const int minSaftLevel, sætter minimum grænsen for saft level
- isDosPos(), returnerer true hvis vi er i doseringsposition, ellers false
- stop(), stopper motoren
- checkWater(), returnerer vandhøjden
- checkSaft(), returnerer safthøjden
- isWaterLevelOk(), returnerer true hvis vandhøjden er over minWaterLevel
- isSaftLevelOk(), returnerer true hvis safthøjden er over minSaftLevel

Ændret:
- dosWater() -> dosWater(bool), gives true som argument starter dosering, gives false stopper dosering
- dosSaft() -> dosSaft(bool), gives true som argument starter dosering, gives false stopper dosering

softView:
Tilføjet:
- setSaftLevel(), bruges til at vise saftniveauet grafisk
- setWaterLevel(), bruges til at vise vandniveauet grafisk
- setStatusMsg(), bruges til at udprinte en status meddelelse.

Fjernet:
- Bool startPressed
- Bool cup
- Bool startPos
- Bool Forward
- Bool Back
- Bool waterHot
- Bool saft
- Bool water
- Int temp, vi har fjernet alle bools, da de er overflødige.

Ændret:
- setTemp(int) -> setTemp(double), da temperaturen er repræsenteret ved en double

HWInterface:

Tilføjet:
- stopWater(), stopper vand doseringen
- stopSaft(), stopper saft doseringen
- getWater(), returnerer vandniveauet
- getSaft(), returnerer saftniveauet
- DBC(), omsætter en værdi mellem 0 og 255 til et 8 bit tal, som bliver lagt i et int-array.
- Clear(), sætter 0 på den digitale udgang, og derved sætter 0 på DOUT0->DOUT7.

Ændret:
- getTemp(int) -> getTemp(double), da temperaturen er repræsenteret ved en double

3.3 Overordnet beskrivelse
Vi vil ikke gennemgå koden slavisk, men i det efterfølgende kapitel beskriver vi hoved-funktionerne. Vi vil dog først forsøge, at give en tekstuel beskrivelse af de enkelte klasser's funktioner. Som beskrevet i struktureringsrapporten er programmet opbygget omkring en model-view-controller struktur.
Programmets main befinder sig i softDrink.cpp. Denne klasse danner instanser af de andre objekter, og kalder til sidst powerOn funktionen i softController.cpp.
SoftController.cpp indeholder funktioner til at styre automaten.
PowerOn styrer hændelsesforløbet under fremstilligen af en softdrink.
driveCupForward styrer koptransporten fra startposition til doseringsposition
driveCupBack styrer koptransporten fra doseringsposition til startposition.
dosWater styrer doseringenen af vand.
dosSaft styrer doseringen af saft.
Styringerne er baseret på en række kald til softModel.cpp
softModel.cpp er bindeleddet mellem controlleren og hardware interfacet. Vi vil ikke gennemgå funktionerne i modellen, da de er forholdvis simple.
softView.cpp sørger for et grafisk output på skærmen. Både modellen og controlleren updatere viewet.
Vi vil gå i dybden med nogle udvalgte funktioner.

3.4 Beskrivelse af hovedfunktioner

`void softController::powerOn()` (koden er vedlagt i bilag 3)
Den overordnede styring af automaten foregår her fra. Når funktionen kaldes, er automaten "tændt". Funktionen består af en uendelig løkke som gør følgende:
1. Først spørger vi modellen om vandniveauet er ok, derefter om saft niveauet er ok.
2. Så spørger vi om der er trykket på start.
3. Hvis der er trykket start, spørger vi modellen, om kravene for at lave en soft drink er opfyldt:
   - Er der en kop, hvis ikke, gøres brugeren opmærksom derpå.
   - Er temperaturen ok
   - Er kopholder i start position

4. Derefter køres koppen til doseringsposition
5. Der doseres vand og derefter saft.
6. Koppen returnerer så til start position, og vi venter på at brugen tager koppen ud af holderen.

Undervejs sker en række opdateringer af viewet.

`void softController::driveCupForward()` (koden er vedlagt i bilag 3)
Denne rutine kører koppen fra start position til doseringsposition. Funktionen gør følgende:
Vi beder modellen starte motoren, og går i følgende løkke:

1. Så længe vi ikke er i doseringsposition og der ikke er trykket reset gøres følgende:
   - Spørg modellen om der er trykket på reset
   - Hvis true, så beder vi modellen stoppe motoren, og vi sætter reset boolen (som indikerer automaten er i reset mode).
2. Hvis der ikke bliver trykket på reset, hopper vi ud af løkken fordi kopholderen er nået til doseringspositionen, og vi beder modellen stoppe motoren.

Undervejs sker en række opdateringer af viewet.

`void softController::dosWater()` (koden er vedlagt i bilag 3)
Denne rutine søger for vanddoseringen. Funktionen gør følgende:

a. Først undersøges om reset boolen er sat, eller der er trykket på reset på automaten. Hvis true, så gør vi ingenting. Ellers går vi igang med doseringen.
1. Først spørger vi modellem om vandniveauet. På baggrund af vandniveauet beregner vi doseringstiden. Formlen er beskrevet i afsnit 5.4.
2. Vi sætter timeren til det ønskede antal milisekunder, starter doseringen, og går i denne løkke:
   - Så længe timeren ikke er færdig spørger vi modellen om der er trykket på reset.
   - Hvis der er trykket reset, så beder vi modellen stoppe vand doseringen
   - Sætter timeren til 3000 milisekunder, og venter på timeren er færdig (Vi venter med andre ord 3 sekunder, for at opsamle dryp).
   - Vi sætter reset boolen
3. Hvis der ikke bliver trykket på reset, så beder vi modellen stoppe doseringen.

`void HWInterface::DBC(int value)` (koden er vedlagt i bilag 3)
Denne funktion bruges til at mappe en værdi mellem 0 – 255 til et 8 bit tal. Det binære tal gemmes i bitArray. Vi fylder bitArrayet op sådan:

- Hvis value er større eller lig 2^{i-1} sættes bitArray[i-1] = 1 og vi trækker fra value 2^{i-1}, ellers sættes bitArray[i-1] = 0;

Dette gøres for alle 8 indgange i bitArray.

`void HWInterface::driveForward()` (koden er vedlagt i bilag 3)
Denne funktion starter motoren i fremadgående retning. Funktionen gør følgende:

- Motoren startes ved at sætte 1. bit på digital out. For at køre frem sættes 2. bit til 1. Det vil sige vi skal sætte binært: 11 på digital out. Det gøres ved at kalde DigitalOutput(3) i ad512drv.cpp.

`void softView::setWaterLevel(int)` (koden er vedlagt i bilag 3)

Denne operation har til formål at udskrive vandniveauet i softdrink-automaten på skærmen. Det har vi valgt at gøre med henblik på fejlfinding, da det ikke er muligt at aflæse væskestanden på selve automaten.
Operationen fungerer på den måde, at den placerer cursoren på det rigtige sted på skærmen og sætter skriftfarven til hvid. Her får vi skrevet "Vand: " ud på skærmen.
Herefter bliver skriftfarven sat til rød, og vi begynder den egentlige udskrift af vandniveauet. Vi får vandniveauet som en værdi af typen int. Hvis den viser sig at være mellem 0 og 650 (i input-værdi) er vi under vores minimumshøjde for vandstanden, og automaten vil da ikke kunne starte doseringsrutinen. Den vil da udskrive en rød streg for at indikere, at automaten mangler vand (det vil også komme som en besked i statusfeltet, men det bliver ikke styret fra denne klasse). Hvis niveauet er højere end minimumshøjden, får vi i stedet skrevet en rød firkant ud (vi benytter tegnet med hex-værdien \xDB fra ASCII-tegnsættet). Herefter checker operationen om niveauet er højere end nogle foruddefinerede værdier, og hvis det er tilfældet skriver den endnu en firkant ud. Efterhånden som vi kommer længere op af skalaen, vil firkanterne skifte farve. Fra rød over i gul og til sidst til grøn. Hele skalaen er inddelt i 12 områder. Derved får vi en sammenhængende bar skrevet ud, som illusterer vandniveauet i softdrink-automaten (ser evt. figur 12).

`void softView::setStatusMsg(char [],bool,bool)` (koden er vedlagt i bilag 3)
Denne operation er som beskrevet tidligere, først blevet tilføjet til softView klassen under implementeringsforløbet. For det første opstod der et behov for at kunne udskrive en fejlbesked eller statusbesked til skærmen. For det andet ville vi gerne have muligheden for at indikere et stop vha. et bip i PC-speakeren.
Operationen starter med at producere bippet i PC-speakeren, hvis inputparametren beep har værdien true. Herefter bliver cursoren sat i den rigtige position og sat til rød. Hvis inputparametren on er true, vil der blive skrevet en rød firkant, for at indikere at der er en besked i statusdisplayet. Ydermere vil karakter-arrayet char[] fra inputtet blive skrevet ud i statusdisplayet. Det bliver skrevet ud vha. to for-løkker indeni hinanden for at give den effekt at teksten kører ind på skærmen fra højre.
Hvis on er false, vil der i stedet blive fjernet en evt. prik i diodefeltet og teksten i statusdisplayet vil blive slettet af en tom streng.

`bool softModel::isTempOk()` (koden er vedlagt i bilag 3)
Denne funktion undersøger om væsketemperaturen er acceptabel. Funktionen gør følgende:

1. Først hentes temperaturen, i celsius.
2. Temperaturen sammenlignes med tempMax.
    a. Hvis temperaturen er for høj, så beder vi hardware interface om at tænde diode D7, se Kravspecifikation afsnit 4.1.
    b. Ellers sørger vi for at slukke dioden
3. Vi returnere true, hvis temperaturen er ok og false, hvis temperaturen er for høj.

Undervejs opdateres viewet.

3.5 Test
Sideløbende med implementeringen har vi testet koden. Da compileren tjekker, at klasserne kan arbejde sammen, altså gennem korrekt parameteroverførsel, så burde vores program virke fint. Den

eneste klasse der kommunikerer med omverden er HWInterface.cpp, som taler med driveren til IO-kortet. Da funktionerne i HWInterface er afhængig af, de input IO-kortet får, så kan vi ikke teste vores program, medmindre hardwaren er tilsluttet. Det har vi klaret ved at modificere HWInterface.cpp til ikke at spørge I/O-kortet, men bare returnere konstanter. Derved har vi kunnet teste resten af programmet. Konklusionen på de tests er, at så længe funktionerne i HWInterface.cpp returnerer gyldige værdier[1], så er resten af programmet stabilt.
For at hele programmet er stabilt, må vi altså sikre os, at HWInterface virker som forventet.
Derfor har vi udført en modultest at klassen.

3.5.1 Test af HWInterface

Vi har skrevet et separat test program til at teste HWInterface klassen imod. Programmet kan bruges til at lave en gennemgående test af hele klassen, men kan også specialiseres til kun at teste dele af klassen. Koden er vedlagt som bilag 3.
Koden er delt op i forskellige tests. Ved at udkommentere kode, kan man vælge hvilken test man vil køre. Som eksempel har vi valgt testen:

```
// --- Test af følere + knapper    --- //
cout <<"\nIs @ StartPos?"<<testInterface.checkStartPos();
cout <<"\nIs @ dosPos?"<<testInterface.checkDosPos();
cout <<"\nWhats the temperature: "<<testInterface.getTemp();
cout <<"\nIs cup present?"<<testInterface.checkCup();
cout <<"\nIs reset pressed?"<<testInterface.checkReset();
cout <<"\nIs start pressed?"<<testInterface.checkStart();
_getch();
// ---  --- //
```

Testen undersøger om der er en kop, hvor koppen er, vandtemperaturen og hvilke knapper der er trykket på. Vi har udkommenteret de andre tests.
Vi tester under følgende forhold:
Kop er i holder.
Kopholder er i startposition
Vi holder start inde
Temperaturen er manuelt aflæst til 20,4 ºC.

Resultat.

```
k:\from e to e\E1_6\PRJ\4. Test\HWItest\Debug\HWItest.exe
Is @ StartPos?1
Is @ dosPos?0
Whats the temperature: 20
Is cup present?1
Is reset pressed?0
Is start pressed?1
```

Resultatet må siges at være præcis som forventet.
Vi har testet resten af HWInterface.cpp, og vedlagt resultaterne af disse test i bilag 4.

[1] Med gyldige værdier menes, at et kald til en funktion som tester på den digitale indgang skal returnere true eller false, og et kald til en funktion, som tester en analog indgang returnerer en integer imellem 0 og 4095.

3.5.2 Test af softView

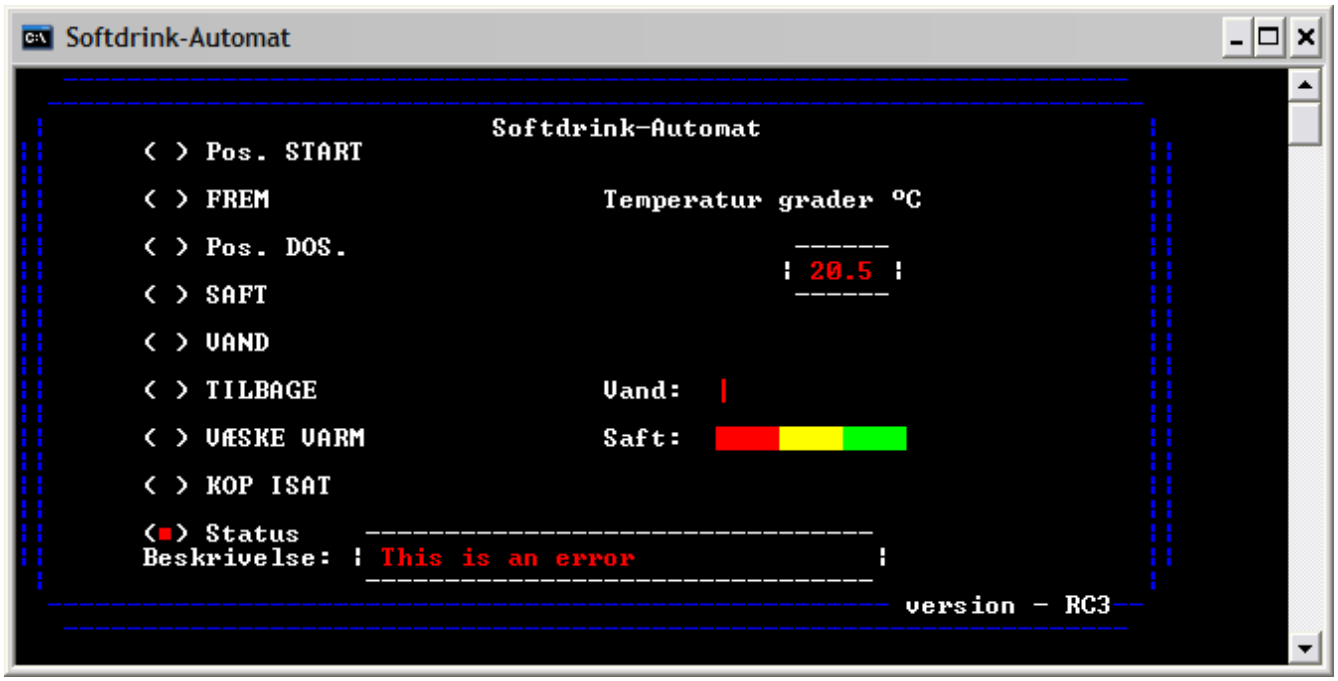
Vores softView klasse har til opgave at komme med en grafisk repræsentation af softdrink-automaten. Den skal være et værktøj til diagnosticering og fejlretning. Den er opbygget med de samme dioder og målere som selve automaten plus et par ekstra, som ikke er at finde på automaten.

Pga. dens formål har vi tilstræbt at skærmudskriften lignede automaten så meget som muligt. Derfor har testen af denne klasse primært gået ud på at checke kvaliteten af udskriften. Dvs. om de forskellige elementer var på de rigtige pladser, og om målerne stemte overens med automatens. Derudover har vi implementeret et status-felt, som vi har testet med forskellige længder af tekst-strenge inkl. en tom streng når feltet skulle slettes, samt et alarm-bip i PC-speakeren ved de beskeder hvor det var relevant.
Testen indeholder følgende:

```
// --- Test af udskrift + status    --- //
softView testView;
testView.setWaterLevel(500);
testView.setSaftLevel(1000);
char array[] = "This is an error";
testView.setStatusMsg(array, true, true);
_getch();
// ---  --- //
```

Testen opretter et objekt af klassen softView og vi får derved printet grunddisplayet ud. Herefter testes udseendet af forskellige værdier for vandniveau og saftniveau, og til sidst tester vi statusdisplayet + bip.

Resultat:



Figur 12: Skærmudskrift

Resultatet var som ønsket og der kom desuden et bip i PC-speakeren.

4. Integration mellem hardware og software

4.1 Integrationstest

Vi vil i dette afsnit beskrive hvordan vi har integreret hardware og software. Vi starter med at beskrive hvordan vi har simuleret hardware for at teste softwaren, og hvordan automaten er simuleret, for at kunne teste hardwaren. Til sidst beskrives hvordan sammenkoblingen af hardwaren og softwaren forløb.

4.1.2 Humusoft test boks

Vi har lånt en Humusoft test boks af værkstedet. Men den har vi kunnet simulere hardware og automat for at kunne teste hardwaren.

Billede 1: Humusoft testboks

Test-boksen giver mulighed for at aflæse outputtet fra I/O-kortet samt sætte de forskellige inputs i testøjemed. Vi kørte forskellige tests på den for at simulere en kørsel af selve automaten. Da vi først gik i gang med at teste på kortet, gjorde vi en antagelse ang. måden kortet håndterer indgangene. Vi antog, at DIN0 (Digital Indgang 0) svarede til den mindst betydende bit i vores inputværdi, DIN1 var den næste bit osv. Men det viste sig at kortet fungerede lige præcis omvendt og altså satte DIN0 som den mest betydende bit (MSB) og DIN7 som den mindst betydende (LSB). Det betød for os, at vi måtte ændre på den operation, der omsætter input-værdien fra I/O-kortet til et bit array, så den sorterede indgangene i den rigtige rækkefølge.
Efter denne første komplikation kunne vi afprøve hele softdrink-rutinen, hvilket forløb tilfredsstillende. Eftersom softwaren tidligere var blevet testet med faste værdier, var der ikke nogen overraskelser på den front. Med test-boksen var vi i stand til at simulere tryk på start, reset, kørsel frem og tilbage samt indikering af koppens position.

4.1.3 Simulatorboks

Vi har fået en simulator boks stillet til rådighed. Denne boks kan simulere selve automaten, og kan bruges som erstatning for denne ved test af hardware og software. Simulatorboksen er elektrisk ækvivalent med automaten. Den indeholder stepmotor og driverkredsløb, temperaturdisplay, dioder svarende til frontpanelet på automaten, samt afbrydere, der fungerer som automatens positionsindikatorer.
Vi har brugt simulatorboksen til at teste og fejlfinde på følgende funktioner:

- Tryk på start
- Tryk på reset
- Motorstyring
- Diodestyring
- Temperaturudlæsning
- Den samlede softdrink-rutine

4.1.4 Automat

Efter den indledende fejlfinding på simulatorboksen skulle hardwaren og softwaren fejlfindes og justeres på de funktioner, som ikke kunne testes på simulatorboksen:

- Motorhastighed
- Væskestandsmålinger
- Dosering af vand og saft

4.2 Kalibrering

Dette afsnit omhandler kalibrering af hardware og software for at sikre korrekt funktion af softdrink-automaten i henhold til kravspecifikationen.

4.2.1 Kalibrering af temperatur

Fra veroboardet får vi en værdi på den analoge indgang mellem 0 og 5 volt. Denne værdi skal konverteres til noget brugbart i computeren. Når vi fra computeren læser på analoge indgang, får vi en værdi mellem 0 og 4096. Der er altså en 12 bit opløsning. Konverteringen sker ved at finde LSB [volt/bit] = 5/4096. Temperaturen fås ved at gange LSB på den værdi der aflæses på den analoge indgang. Da 1 volt svarer til 10 grader, skal der ganges en faktor 10 på.
Den endelige formel er:

$$\frac{Ana\log Input \cdot 5}{4096} \cdot 10 = \text{temperatur i } ^\circ\text{C}$$

Vi har testet kalibreringen af temepraturen. Resultatet findes i testdokumentationen afsnit 2.1 accepttest: 3.1.1.4.

4.2.2 Kalibrering af væskemålere

På den analoge input kan vi i teorien læse en værdi mellem 0 og 4096. Som beskrevet i afsnit 2.2.1 har vi 1365 steps som opløsning for væskemålerne. I praksis læser vi en værdi mellem 650 og 2015. Vi har valgt ikke at konvertere denne værdi til liter (eller en anden enhed) da det ikke er nødvendigt. Vi bruger værdien fra den analoge indgang til at beregne doseringstiden. Derudover bruges den til udskrivning af væskeniveauet på skærmen i form at en bar. Til disse formål behøver værdien ingen enhed .

4.2.3 Kalibrering af dosering

For at kunne dosere den korrekte mængde vand og saft, skal vi vide noget om automatens evne til at dosere ved forskellige væskestande. Kort beskrevet så giver en bestemt åbningstid af ventilerne forskellige mængder væske afhængig af væskestanden i beholderen.
Vi ønsker altså at beregne en funktionsforskrift, der beskriver hvor lang tid den pågældende ventil skal være åben for at vi får den ønskede mængde væske som en funktion af inputtet fra væskemåleren.

Nedenstående tabel viser tider for hhv. saft og vand.

Saftniveau [inputværdi]	Doseringstid [ms]
650	850
900	750
1065	650
1230	600
1360	575
1475	550

Vandniveau [inputværdi]	Doseringstid [ms]
650	4200
860	3700
1020	3250
1170	3000
1275	2800
1380	2650

Tabel 1: Doseringstider (af en færdig softdrink) for saft og vand

Doseringstiderne er fremkommet ved at justere doseringstiden i softwaren indtil den doserede mængde var korrekt: Hhv. 25 [ml] saft og 125 [ml] vand.

Doseringstiden som funktion af væskestandsinputtet er tilnærmelsesvis lineær (i det område vi benytter). Det vil sige, at vi vil beregne en ret linie af formen f(x) = ax + b med en negativ hældningskoefficient. For at finde denne hældningskoefficient bruger vi de to yderpunkter fra målingerne:

Saft:

$$Hældningskoefficient = \frac{\Delta tid}{\Delta niveau} = \frac{550 - 850}{1475 - 650} = -0{,}364$$

Skæring med y-aksen:
f(650) = -0,364 * 650 + b = 850 => b ≈ 1086

Funktion:
f(x) = 1086 – 0,364 * x , hvor x er saftniveauet repræsenteret ved en decimal inputværdi

Hvis vi sammenligner denne funktion med vores oprindelige måledata kan vi tilpasse den lidt bedre og vi ender derved med:

f(x) = 1065 – 0,36 * x , hvor x er saftniveauet repræsenteret ved en decimal inputværdi

Vand:

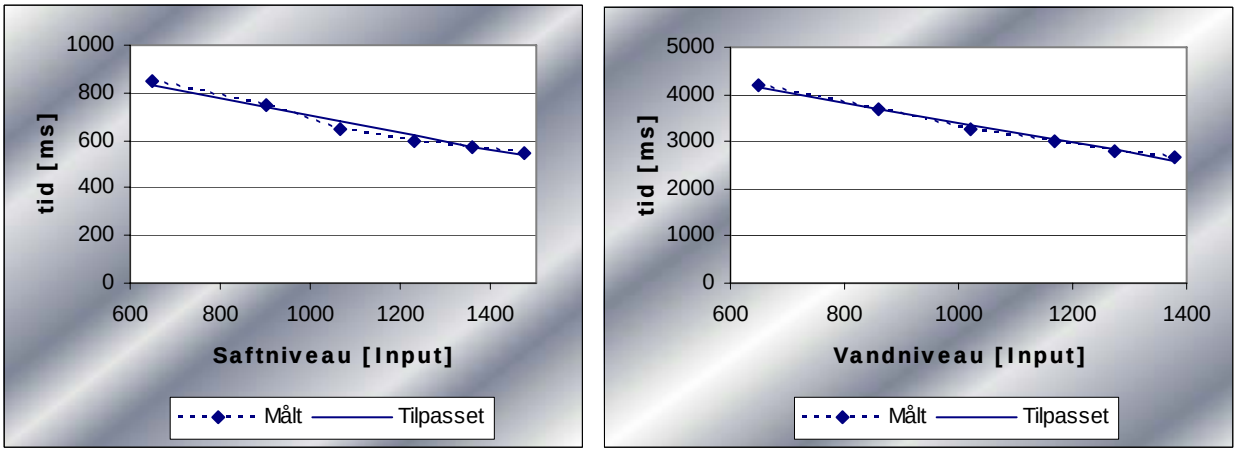
$$Hældningskoefficient = \frac{\Delta tid}{\Delta niveau} = \frac{2650 - 4200}{1380 - 650} = -2{,}12$$

Skæring med y-aksen:
f(650) = -0,364 * 650 + b = 4200 => b ≈ 5578

Funktion:
f(x) = 5578 – 2,12 * x , hvor x er vandniveauet repræsenteret ved en decimal inputværdi

Igen kan vi lave en approksimation:

f(x) = 5525 – 2,12 * x , hvor x er vandniveauet repræsenteret ved en decimal inputværdi



Figur 13: Kurver for målt og beregnet vand- og saftdosering

De to fundne forskrifter indsættes i softwaren og benyttes til beregning af doseringstider.
Vi fandt desuden at en efterdrypstid på ca. 3 sekunder var tilfredsstillende. Det betyder to ting for vores software:

1. Koppen skal først køre tilbage 3 sekunder efter endt dosering.
2. Ved reset under dosering ventes 3 sekunder inden tilbagekørsel.

5. Underskrift

Implementeringsrapporten er færdiggjort d. 11. juni 2003

_____

_____

_____

_____

_____

Gruppe 6