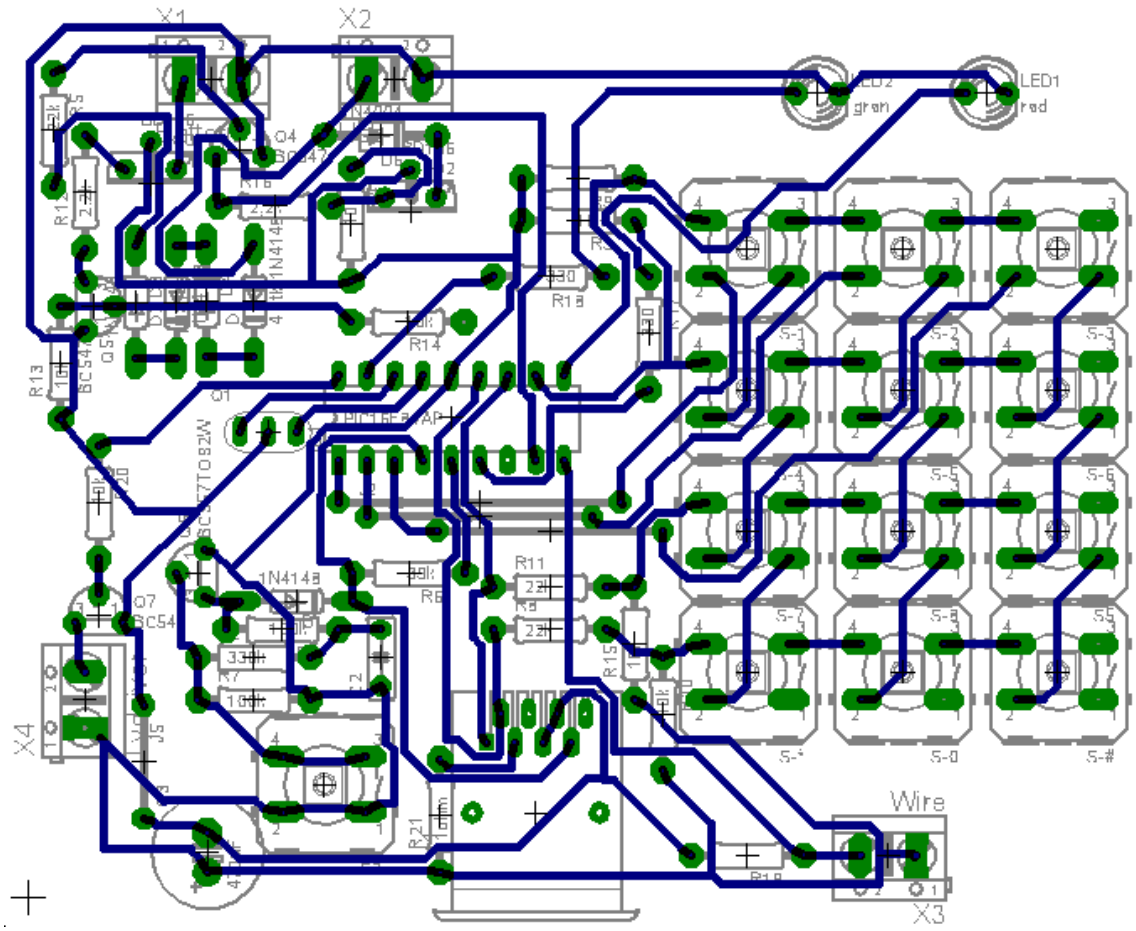
USB Alarm



Fag :	El-teknik A
Navn :	Bent Arnoldsen
Skole :	Holstebro Tekniske Gymnasium
Periode :	Uge 43 til 48 2005
Aflevering :	29. november 2005

Indholdsfortegnelse:

Indledning:

Denne rapport er fremstillet som et eksempel på hvordan en rapport til faget El-teknik A (Design og Produktion med el som speciale) på 3. år af HTX-uddannelsen kan se ud.

Ideen stammer fra et projekt som Gitte Bundgaard lavede på 1. år i 2005. Ideen er taget med tilladelse fra Gitte, men hovedparten af dokumentationen er lavet som eksempel af Bent Arnoldsen.

Ideen.

Selve ideen med produktet var, at fremstille en alarm til en bærbar computer, som var simpel at anvende, men alligevel svær at snyde. Alarmen skulle betjenes med en kode, den skulle larme op hvis alarmen gik og den skulle kunne sikre at man ikke bare tog den bærbar computer og vandrede af med den.

Problemformulering.

- Hvorfor er det nødvendigt at sikre sine ting med alarm?
- Hvordan skal alarmen udformes?
- Hvordan løses forsynings-problematikken?
- Hvordan fremstilles tastaturet?
- Hvilken type lydgiver skal der bruges?
- Hvordan lagrer man koden til alarmen?

Problemanalyse.

Baggrunden for at der er et stigende salg i alarmer er den tiltagende kriminalitet, hvor der stjæles mere og mere[1]. Det er ikke bare i virksomheder og private hjem at der er set en stigning i tyverier, men også på mere eller mindre offentlige steder som skoler og andre institutioner.

På Holstebro HTX er det ikke et stort problem endnu, men vi kan se tendensen efterhånden som vi bliver en større afdeling, og vi ikke kender hinanden på sammen måde, som da det var en mindre afdeling med kun 150 elever i alt. En anden del af problemet er at der er mange flere der har bærbare computere, og de bliver anvendt alle steder.

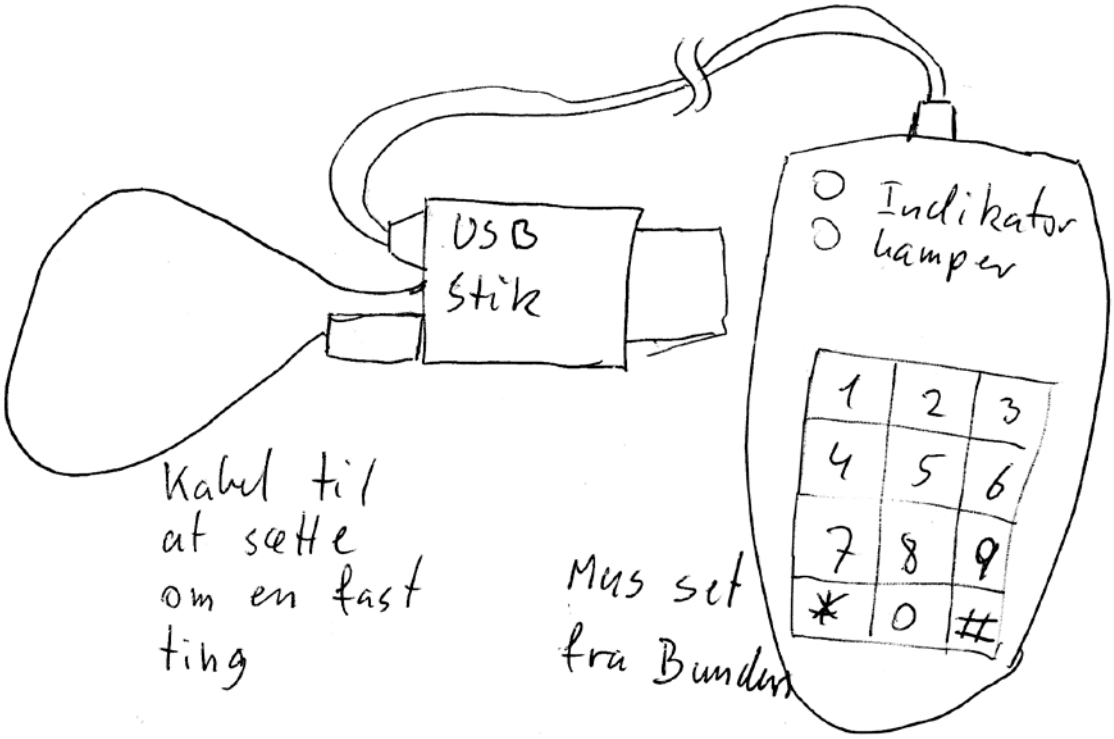
Ud fra disse overvejelser vil det være fornuftigt at fremstille et prisbilligt produkt, der kan anskaffes og anvendes af folk med bærbare computere.

Det er ikke kun den økonomiske del af det at miste sin computer, men også det at man mister mange timers arbejde, hvis ens bærbare computer bliver stjålet.

[1] Her skal henvises til en kilde. Jeg har i denne rapport valgt ikke at lave det arbejde, men det er denne del, der kan dække det tværfaglige aspekt.

Udformningen af alarmen vil i dette projekt blive lavet som en simpel prototype, dog med vægten lagt på at tingene skal fungere. De tekniske dele af projektet vil blive problemanalyseret hen gennem den tekniske beskrivelse i rapporten.

Et af de mere kreative forslag, der var oppe at vende under produktudviklingen var at kombinere alarmen med en mus. Det kunne laves ud fra følgende skitse:



Figur 1. Håndskitse af produktide.

Ideen var at man normalt altid slutter en mus til sin bærbare, når man skal arbejde lidt mere. Det der så skulle udvides med var at sætte et alarm-kabel fast i USB-stikket, så det ikke generede arbejdet med musen. Dette kabel skulle sidde fast i USB-stikket i den ene ende, og i have et stik i den anden ende, som så skulle sættes i bagenden af USB-stikket, for at sikre den bærbare. Kablet skulle have en længde, så det kunne komme rundt om en fast ting (stoleryg, radiator eller lign.). Kablet behøver ikke at være særligt kraftigt, da alarmen skulle lyde, hvis kablet bliver afbrudt eller stikket trukket ud.

På undersiden af musen skulle der så være indfældet et tastatur, så man kunne slå alarmen til og fra, og også nogle indikatorlamper, så betjeningen bliver brugervenlig. Det kunne evt. også være et display, men det ville sikkert fordyre produktet. Tastaturen kunne være en folie-type, så det både er prisbilligt og heller ikke ville genere den normale betjening af musen.

Hovedparten af elektronikken kunne så ligge i musen, og blive forsynet via USB-porten.

Valg af løsnings-strategi.

Jeg har valgt at opbygge og teste opstillingen på fumlebræt, specielt forsyningsdelen skal testes sammen med programmet, inden det lægges på print, men også lydgiver og

wire-interface skal testes. Det hele skal ende i et prototypeprint, men ikke sammen med en mus.

Krav og specifikationer.

Alarmen skal kunne køre på de 5 V fra en USB-port.
Alarmen skal også kunne virke på batteri, gerne generelt anskaffelige batterier.
Wiren skal kunne tilsluttes og udløse alarmen.
Alarmen skal udløses når USB-forsyningen afbrydes.
Alarmen skal give et lydsignal når alarmen udløses.
Alarmen skal kunne frakobles med en 4-cifret pinkode.
Der skal være mulighed for at ændre pinkoden.
Betjeningen skal være så simpel som muligt.

Tidsplan.

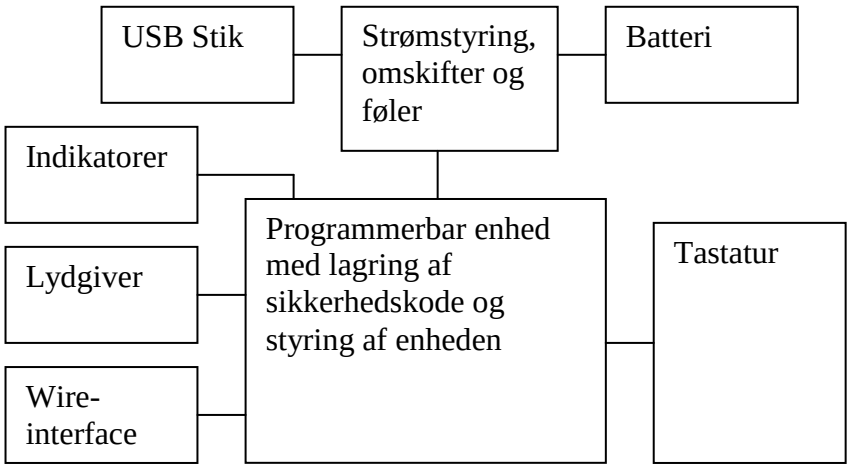
Tidplanen er lavet for at kunne overskue projektforløbet, og holde styr på hvilke aktiviteter der skal udføres, og hvad der stadig mangler at blive lavet.

Aktivitet	**Uge 43**	**Uge 44**	**Uge 45**	**Uge 46**	**Uge 47**	**Uge 48**
Planlægning og analyse	ooooooooo xxxxxx		xx			
Konstruktion af elektronik	xxx	ooo ooo xxxxxx				
Opbygning på fumlebræt		ooo xxx	ooo xxx			
Test på fumlebræt		ooooooooo xxx	oooooo xxx			
Print-udlægning				ooooooooo xxxxxx	xxx	
Test af printet					oooo xxxxxx	
Programmering af alarmen		xxx	oooooo xxxxxx		oooooo xxx	xxx
Test af software		xxx	ooo x x xx			oooooo xxxxxxx
Rapport-skrivning	ooooooooo xxx	ooo xxx	ooo	ooo xxx	ooo	oooooo xxx

o – Planlagt aktivitet
x – Gennemført aktivitet

Blokdiagram.

For at få et overblik over den elektriske del, så er der her skitseret et blokdiagram over den elektriske del af projektet. De forskellige dele vil blive uddybet yderligere hen gennem rapporten.



Figur 2. Blokdiagram over hele USB alarmen

USB Stikket skal sørge for at forsyne alarmen med strøm. I denne version er der ingen kommunikation med USB-porten.

Batteriet skal forsyne USB-alarmen, hvis man trækker USB stikket ud, så alarmen stadig virker, selvom man forsøger at snyde den på denne måde.

Strømstyringen skal sammen med den programmerbare enhed sørge for at der er strøm til USB alarmen, når vi ønsker det. Samtidigt med skal den sørge for at vi kun bruger batteriet når det er højst nødvendigt.

Den **Programmerbare enhed** er den der skal lave alle funktionerne i alarmen.

Tastaturet skal sørge for betjenings interfacet. Det skal her være muligt at slå alarmen til og fra, og der skal også være mulighed for at ændre koden.

Lydgiveren siger sig selv, den skal sige noget, når alarmen går, og det skal gerne være højt.

Wire-interface skal være det der føler om wiren er brudt.

Indikatorlamperne skal være en del af betjeningen, så brugeren kan finde ud af hvordan alarmen skal betjenes.

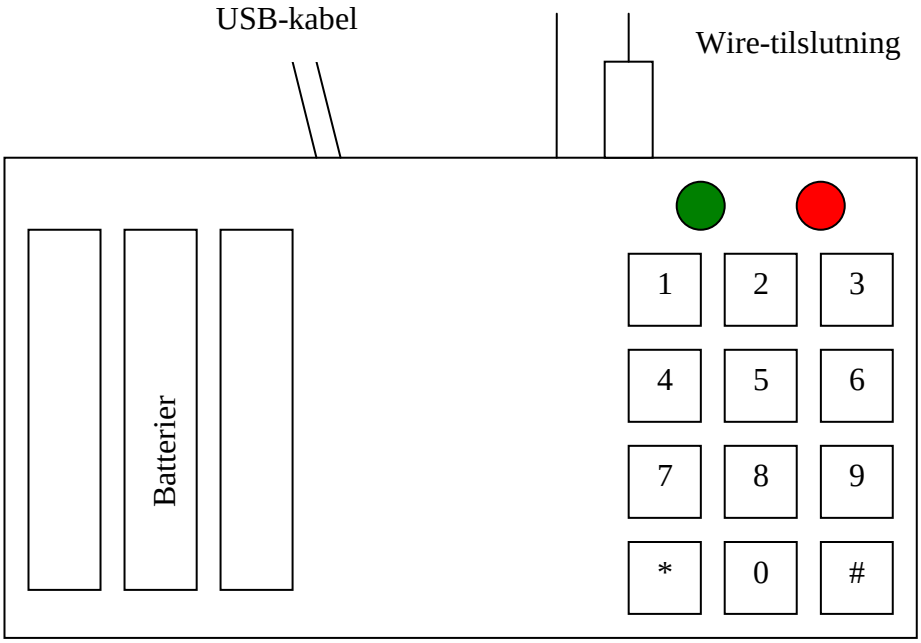
Hen gennem rapporten vises de del-diagrammer, der dokumenteres. Det samlede diagram kan ses i bilag 1 bagerst i rapporten.

Den fysiske udformning af prototypen.

Ideen med at indbygge alarmen i en mus blev hurtigt droppet, da det ville give problemer med pladsen i en mus, når det skulle laves som almindelige leadede komponenter.

I stedet har jeg valgt at lave det som et selvstændigt print, men stadig tilsluttet til USB-porten. Tilslutningen af wiren er lavet med simle headers på printet, og så er der brugt et jack-stik til den ende af wiren, så den let kan kobles til og fra.

Det layout jeg har gået efter er nogenlunde som vist her:



Figur 3. Layout-skitse for prototypen.

Tastaturet skal placeres til venstre, så man kan resten af enheden, når man taster med højre hånd. Lysdioderne placeres over tastaturet, så enheden også kan betjenes af venstrehåndede uden problemer.

Ud over det er der kun tilslutningen af USB-kabel og wiren der skal tages hensyn til, og det er de ret frie rammer for.

Valg af programmerbar enhed.

Der findes mange forskellige programmerbare enheder, men da den vi arbejder med i el-teknik på Holstebro HTX typisk er en PIC-16F84, og der ikke er nogen specielle krav til hvad processoren skal kunne, ud over at der skal være benforbindelser nok til at den kan klare det, men der skal vi lige igennem alle de andre blokkes krav for at kunne sikre det.

Valg af programmeringssprog.

Programmeringssproget til enheden vil kunne klares med JAL. Der er et simpelt anvendeligt sprog.

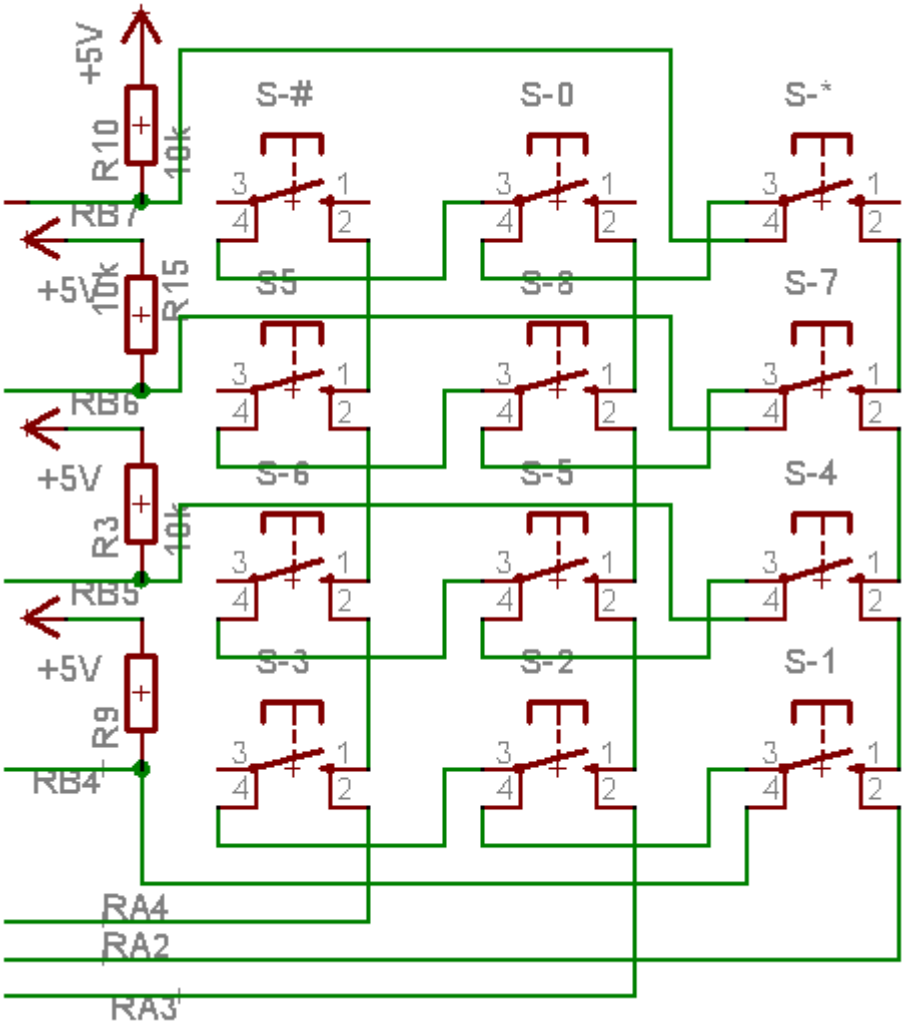
Tastaturet.

Da jeg har valgt at arbejde med en tal-kode, så skal der være 10 taster på tastaturet, og til at få lagt kode ind og nulstille osv. vil det være praktisk med 12 taster (en * og en #

tast), det giver 12 taster, der kan placeres i 3 x 4 som på en telefon, for at gøre det lettere at betjene.

Til den elektriske opkobling er der valgt er matrix-struktur, da de 12 taster så kan aflæses med 3 udgange og 4 indgange, altså i alt 7 I/O fra PIC-kredsen. Opkoblingen af tastaturet og det tilhørende softwaremodul er taget fra El-teknik[2] hjemmesiden. Tilslutningen til softwaren er beskrevet under softwarebeskrivelsen.

Diagrammet til tastaturet ser ud som følger:



Figur 4. Diagram over Tastatur.

Som det kan ses på diagrammet, så er tasterne tegnet lidt rodet, men hvis man følger fra port-navnene, så er RA2 tilsluttet tast-1, RA3 til tast-2 og RA4 til tast-3, og RB4 – RB7 aflæser også i rækkefølge. På printet er tasterne selvfølge lagt således at de ligger som på et normalt tastatur (som vist på figur 3).

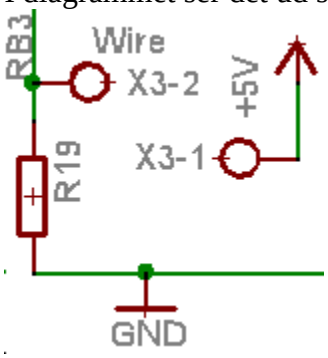
Wire-interfacet.

Wire-interfacet skal bare give et signal ind i PIC'en, der angiver om wiren er intakt og sat i stikket eller ej.

[2] Kilde: **El-teknik** http://www.holstebrohtx.dk/bar/index.php?pkt=1371630 set 26/11 2005.

Det gøres ganske simpelt ved at forbinde den ene ende af wiren til +5V og den anden ende til en indgang RB3, som så skal have en pull-down til stel, for at indgangen kan se forskel på signalet.

I diagrammet ser det ud som følger:



Figur 5. Diagram over Wire-interface.

Beregninger til Wire-interface.

Der er ikke de store beregninger til wire-interfacet. Der er valgt en pull-down-modstand på 10k. Det giver følgende strømforbrug:

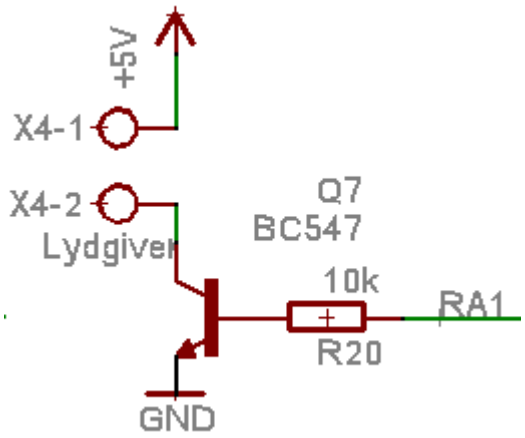
$$I_{19} = \frac{5V}{10k} = 0{,}5mA$$

Lydgiver.

Til den endelige alarm skal der vælges en type lygiver, der kan give noget der ligner 110dB, for at det kan virke afskræmmende og alarmere andre personer, men til prototypen er der bare valgt en lydgiver der kan sige noget ved 5V.

Den type der er valgt trækker ca. 25 mA, og det ligger lige omkring grænsen for hvad en PIC-port kan levere. Derfor er der valgt at sætte en transistor på til at forstærke strømmen.

Transistoren arbejder som switch, og er koblet op på følgende måde:



Figur 6. Diagram til Lydgiver.

Lydgiveren er placeret uden for printet ved hjælp af et Header-stik X4.

Beregninger til Lydgiveren.

I databladet for en BC547 kan man aflæse at h_{FE} er mindst 100 gange, og at V_{BE} er ca. 0,7V. Det giver følgende beregninger:

$$I_{R20} = \frac{U_{RA1-ON} - V_{BE}}{R20} = \frac{5V - 0{,}7V}{10k\Omega} = 0{,}43mA$$

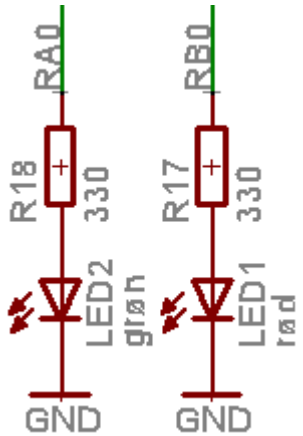
$$I_{B7} = I_{R20}$$

$$I_{C7} = I_{B7} \cdot h_{FE} = 0{,}43mA \cdot 100 = 43mA$$

Da lydgiveren kun trækker 25 mA, så er transistoren Q7 helt ON.

Indikatorer med lysdioder.

Det naturlige valg til at få en rød og en grøn lampe, der bare skal virke som indikatorlampe er lysdioder. Det giver følgende diagram:



Figur 7. Diagram over indikatorerne med Lysdioder.

Til at styre strømmen i lysdioderne anvendes en modstand til hver.

Beregninger til indikatorerne med lysdioder.

Der er lidt forskel på en rød og en grøn lysdiode, nemlig den spænding der ligger over lysdioden, når den lyser, hvor den røde er på ca. 1,5 V og den grønne er på ca. 2V. Lysdioderne drives direkte af PIC'ens udgang, og udgangen kan levere ca. 5V.

Der vælges en strøm på ca. 10 mA i lysdioderne, da det normalt giver et fornuftigt lys. Det giver følgende beregninger:

$$R_{17} = \frac{5V - 1{,}5V}{10mA} = 350\Omega$$

$$R_{18} = \frac{5V - 2V}{10mA} = 300\Omega$$

Begge modstande afrundes til standardværdien 330Ω.

Forsynings-problematikken.

Når man analyserer på det, så skal forsyningen kunne håndtere 3 forskellige tilstande:

1. Når enheden er forsynet fra USB-porten.
2. Når spændingen fra USB-porten forsvinder, men alarmen skal lyde, så skal den forsynes fra batteriet.
3. Når alarmen har lydt længe nok, så skal enheden slukke for sig selv, for ikke at bruge alt batteriet.

Overvejelser omkring batteridrift.

Hvis man skulle lave en produktionsmodel af USB-alarmen, så skulle den laves sådan at den kan arbejde med et opladeligt batteri, eller at den kan fungere på en kondensator, så længe alarmen skal lyde.

Dette ville dog betyde at man skulle lave et ladekredsløb, og også sikre at batteriet blev motioneret, i stedet for at der bare blev ladet på det.

I stedet har jeg valgt at lave denne del med almindelige batterier. Der er valgt at de bare skulle være 3 stk. 1,5 V AA celler, ud fra de overvejelser der kommer senere med hensyn til forsyningsspændingen.

Det er selvfølgelig ikke en holdbar løsning i en produktionsmodel, men til prototypen illustrerer det meget fint funktionen.

Tilslutning til PIC'en.

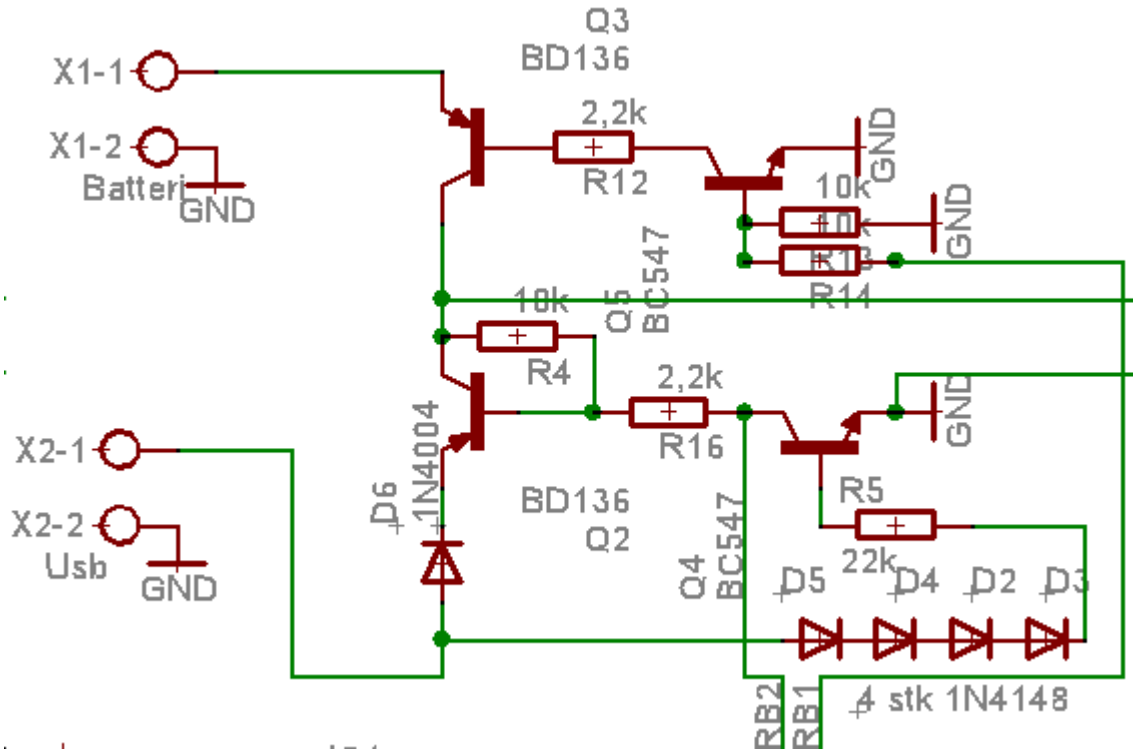
For at kunne håndtere de tre tilstande med de tider der skal lægges ind, så skal PIC'en have besked om der er forsyning fra USB-porten, og den skal kunne tænde og slukke for batteriforsyningen.

Signalet til PIC'en om der er forsyning fra USB-porten er bygget sammen med den switch der forsyner PIC'en fra USB-porten. For at der ikke skal løbe strøm baglæns er

der placeret en diode før switchen. Denne diode er også med til at signalet til PIC'en hurtigere.

Endelig skal batteriet kunne forsyne PIC'en, men det skal også være sådan at batteriet ikke belastes, når der er slukket for det.

Det har resulteret i følgende kredsløb:



Figur 8. Diagram for forsyningskredsløbet.

Beregninger på forsyningskredsløbet.

Den første forudsætning vi skal gøre er, hvor meget strøm hele kredsløbet trækker. Selve PIC'en trækker ikke mere end 10 mA, de to lysdioder trækker max. 20 mA og lydgiveren skal have 25 mA. Hvis der afsættes 15 mA til resten, så trækker hele kredsløbet i alt 70 mA.

Hvis man læser i databladet for PIC'en[3], så kan den arbejde med til 4V, så vi kan tåle at tabe ca. 1 V fra USB-porten. Der falder ca. 0,7 V over D6 og ca. 0,2 V over Q2 ved den de 70 mA, hvis USB-porten ikke kommer under 4,9 V, så holder forudsætningen. Ligeledes kan batteriet komme ned på 4,2 V, da der også kun falder 0,2 V over Q3.

Beregningerne på Q2 og Q3 kan gøres fælles, hvis man regner på Q2 der får den mindste spænding.

R16 og R12 kan hermed sættes til 2,2 kΩ.

[3] Kilde: Microchip, http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf Set 26/11-2005.

$I_{Q2} := 0.07A$

$h_{FE2} := 40$ Aflæst i databladet for BD136

$I_{R16} := \frac{I_{Q2}}{h_{FE2}}$

$V_{USB} := 4.9V$ Mindsteværdien som forklaret i teksten

$V_{BE2} := 0.7V$ Fra databladet på en BD136

$V_{CE4} := 0.2V$ Fra databladet på en BC547

$R_{16} := \frac{V_{USB} - V_{BE2} - V_{CE4}}{I_{R16}}$ $R_{16} = 2.286 \times 10^{3}\,\Omega$

R4 på 10 kΩ sidder der for at trække BR2 høj, når Q4 slukker fordi USB-forsyningen forsvinder. Grunden til at PIC'en kan fortsætte med at fungere er, at der på forsyningen sidder en kondensator på 470 µF, der kan lever strøm til PIC'en i et meget kort tidsrum, mere om hvordan det løses under beskrivelsen af programmet.
De 10 kΩ er blot taget som en erfaringsværdi, ud fra at den ikke må trække for stor strøm, og at den skal være lille nok til at trække høj inden for en fornuftig tid.

For at få signalet om at USB-forsyningen ind til PIC'en så hurtigt som muligt, så er Q4 lavet så den slukker ved ca. 3 V. Beregningerne af R5 starter ved strømmen i R16, som er beregnet før:

$R16 := 2200\Omega$

$I_{R16} := \frac{V_{USB} - V_{BE3} - V_{CE4}}{R16}$ Strømmen i R16 beregnes ud fra den valgte modstand

$V_{BE4} := 0.6V$ Spændingen lige når transistoren slukker

$h_{FE4} := 200$ Fra databladet på BC547

$V_{D} := 0.6V$ Fra databladet på 1N4148

$I_{R5} := \frac{I_{R16}}{h_{FE4}}$

$R5 := \frac{V_{USB} - 4 \cdot V_{D} - V_{BE4}}{I_{R5}}$ $R5 = 2.09 \times 10^{5}\,\Omega$

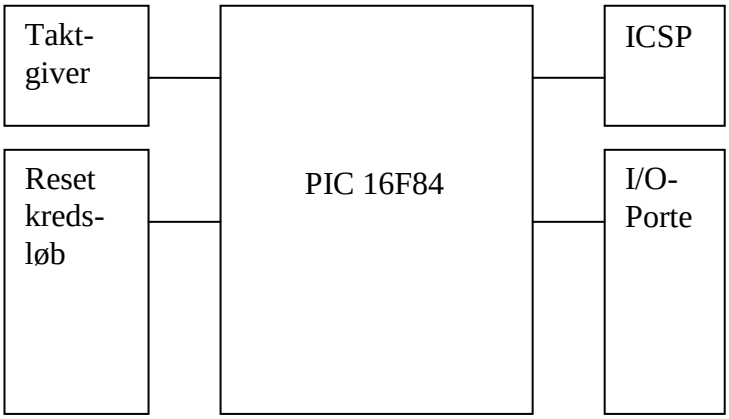
For at være sikker på at Q4 tænder ordentligt sættes R5 til 22 kΩ.

R13 og R14 er bare sat til 10 kΩ ud fra erfaring. R14 skal blot sikre at Q5 tænder ordentligt når RB1 går høj. R13 sidder der for at sikre at Q5 slukker helt, så vi ikke trækker noget strøm fra batteriet, når det ikke er meningen vi skal bruge strøm fra det.

Den programmerbare enhed.

Jeg har valgt at anvende en PIC 16F84 ud fra at det er den vi anvender i undervisningen til daglig. Oplysningerne om PIC'en er overvejende taget fra el-teknik hjemmesiden[4] og fra databladet til PIC'en[5], så jeg vil kun henvise specielt til disse kilder, når jeg har værdier og kredsløb taget direkte herfra.

Den programmerbare enhed består af en række blokke som illustreret:



Figur 9. Blokdiagram over den programmerbare enhed.

PIC 16F84 er hjertet i den programmerbare enhed. Det er den programmerbare controller.

Taktgiveren er den der angiver hvor hurtigt PIC'en skal afvikle instruktionerne. Det er en 4MHz resonator, den vil ikke blive omtalt yderligere.

Reset kredsløbet er det kredsløb der skal sørge for at PIC'en starter i adresse 0, når kredsløbet tændes.

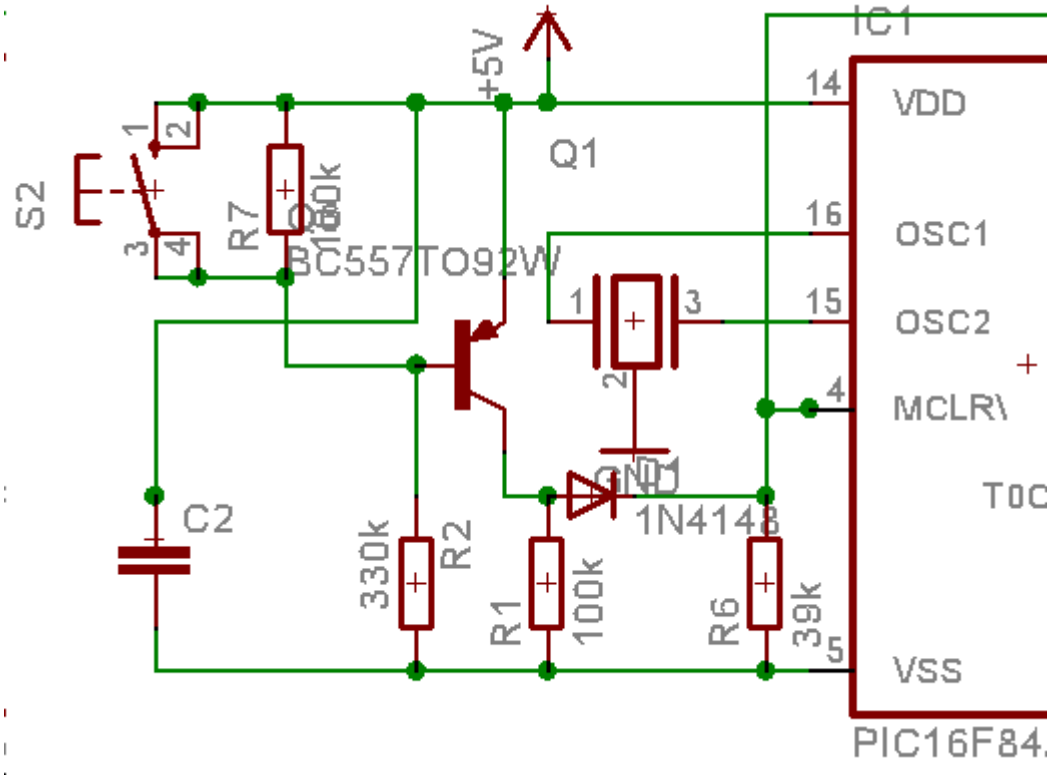
I/O portene er dem der har været omtalt hele vejen ned gennem teksten.

ICSP er In Circuit Serial Programming, der betyder at vi kan programmere PIC'en uden at tage den ud af kredsløbet.

[4] Kilde. El-teknik: http://www.holstebrohtx.dk/bar/index.php?pkt=1371 set 27/11-2005.
[5] Kilde. Microchip, http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf Set 26/11-2005.

Reset kredsløbet.

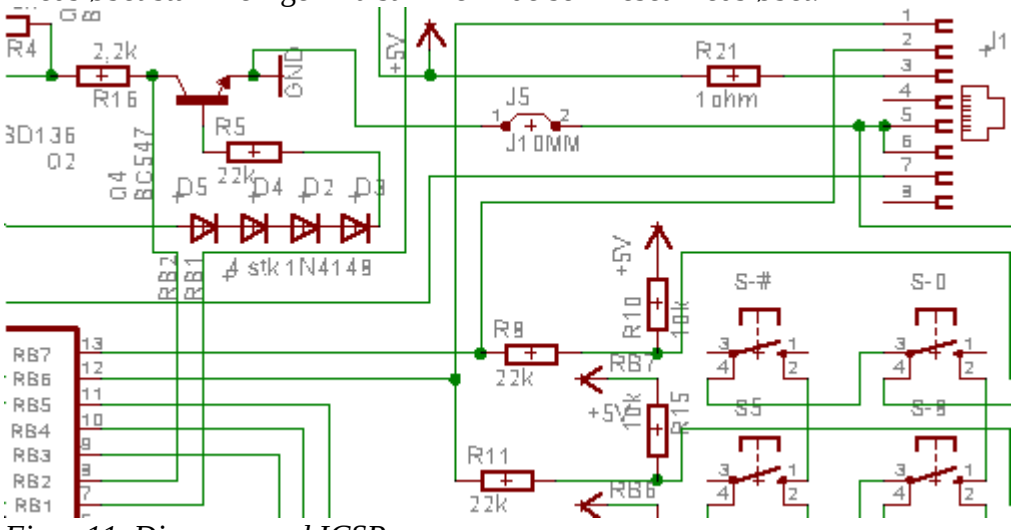
Kredsløbet er taget direkte fra el-teknik hjemmesiden[6] i den version, der kan arbejde sammen med ICSP. På diagrammet ses også resonatoren der sidder på PIC'ens OSC1 og OSC2.



Figur 10. Diagram over resetkredsløbet.

ICSP

Kredsløbet stammer igen fra samme kilde som reset kredsløbet.



Figur 11. Diagram med ICSP.

De to modstande R8 og R11 gør at RB6 og RB7 gør at tastaturet ikke genererer ICSP.

[6] Kilde: El-teknik http://www.holstebrohtx.dk/bar/index.php?pkt=137131 Set 27/11 2005

I/O porte.

Portene på PIC'en er omtalt der hvor de anvendes, men for at danne et overblik er de listet op her, med kommentarer.

Portben	**Type**	**Anvendelse**
RA0 og RB0	Output	Indikator Lysdioder – Lys ved høj
RA1	Output	Lydgiver – Aktiv høj
RA2 – RA4	Output	Tastatur MUX-ben
RB1	Output	Tænder batteriforsyning – Aktiv høj
RB2	Input	Føler om der er USB-forsyning – Lav når der er forsyning.
RB3	Input	Wire interface – Lav når wiren er brudt
RB4 – RB7	Input	Tastatur Sense-ben. RB6 og RB7 bruges også i ICSP

Lagring af alarmkode.

Når alarmen programmeres, så skal der ligge en standardkode i den, men for at alarmen kan bruges fornuftigt, så skal det være sådan at brugeren selv kan lægge sin egen kode ind i alarmen.

Den måde koden kan gemmes på er ved at placere den i EEPROM-en inde i PIC'en, som det kan ses i databladet for PIC'en[7].

Softwaren til PIC'en.

Programmeringssproget det er valgt til softwaren er JAL, der er brugt i undervisningen i el-teknik. JAL stammer fra en hollandsk programmør, der manglede et simpelt sprog til PIC, og derfor lavede en compiler til det. Compileren er freeware og kan hentes på hans hjemmeside[8], men er også placeret på el-teknik hjemmesiden.

Under udviklingen af softwaren er anvendt JAL Edit, der også er et freewareprogram, der er udviklet af Sunish Issac. Programmet kan hentes på hans hjemmeside[9], eller på el-teknik hjemmesiden. Grunden til at bruge editoren er, at der er code completion, syntax highlight og mange andre gode faciliteter.

Grundlæggende krav til softwaren.

Softwaren skal selvfølgelig kunne håndtere alarmen. Det siger sig selv, men for at det skal kunne lade sig gøre, så kræver det at softwaren kan håndtere flere ting samtidigt.

Dette gøres ved at bruge polling, som beskrevet på el-teknik hjemmesiden[10]. Det betyder bare at man løber hurtigt rundt i en loop og spørger på de forskellige ting der skal serviceres. Hvis det sker hurtigt nok, så betyder det, at brugeren ser det som om det sker samtidigt.

---

[7] Kilde: Microchip http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf Set 26/11 2005
[8] Kilde: Wouter van Ooijen http://www.voti.nl/jal/free.html Set 27/11 2005
[9] Kilde: JAL Edit http://jaledit.sunish.net/ Set 27/11 2005
[10] Kilde: El-teknik http://www.holstebrohtx.dk/bar/index.php?pkt=13718 Set 27/11 2005

Betjeningen af alarmen.

Betjeningen af alarmen skal ske via tastaturet, og det brugeren kan se er de to lysdioder, og så kan brugeren selvfølgelig også høre alarmen.

Når alarmen tændes, så er det meningen at den skal indikere at den er tændt, og slå alarmen til.
Det at den er tændt indikeres ved at den grønne lysdiode tændes 1 sekund og at den røde blinker 3 gange og så er tændt. Alarmen skal så lyde, hvis man bryder wiren, eller hvis man trækker USB-stikket ud, og den skal så lyde i 2 minutter, hvorefter den skal slukke. Hvis der er power på (fra USB) og wiren stadig er brudt, så vil alarmen fortsætte. Hvis den er gået på batteri, så skal den slukke for alarmen.

Ved normal brug skal man selvfølgelig også kunne pille alarmen fra uden at den hyler. Det gør man ved at taste koden. Hvis det er den rigtige kode, så slukkes den røde lysdiode og den grønne tændes, så det indikeres at alarmen er slået fra og kan afbrydes.

Brugeren skal også kunne skifte passwordet. Det gøres ved at slå alarmen fra og så taste * efterfulgt af det nye password og så #. Hvis det gøres korrekt, så blinker den grønne lysdiode 5 gange, og alarmen slår til igen.

Passwordet er begrænset til at være en 4 cifret pinkode, og den skal være 4 cifre (altså ikke mindre antal).

Det ovenstående er hvad brugeren har brug for at vide om alarmen for at kunne anvende den.

Specielle krav til softwaren.

Der er også nogen mere specielle krav til softwaren, for at alarmen skal kunne fungere.

Der er hele håndteringen af batteriet, der gør at der ikke må være nogen ting der tager lang tid i softwaren, men at der hele tiden skal tjekkes på om der er strøm fra USB-porten, og hvis den forsvinder, så er det vigtigt at der slås over til batteridrift inden den kondensator C3, der holder forsyningen, er ladet for meget af. Jeg har ikke målt på tiden, men det ser ud til at det fungere.

Ud over det, så skal softwaren holde nogenlunde styr på tiden der går, men det er slet ikke på niveau med et ur, det er bare til at alarmen skal lyde i 2 minutter, og at blinkene skal tage 1 sekund for et helt blink (tændt ½ sekund, slukket ½ sekund).

Variabler der bruges på tværs i softwaren.

De forskellige dele af softwaren skal arbejde sammen, og det sker hovedsageligt ved hjælp af variabler. De vigtigste variabler beskrives her.

Der er de 4 variabler der indeholder hver sit ciffer i passwordet:

```
var       byte      kode_1
var       byte      kode_2
var       byte      kode_3
var       byte      kode_4
```

Variablen `mode` angiver om alarmen er aktiveret. 0 betyder aktiveret og 1 betyder slået fra. Variablen `alarm` angiver om der er detekteret en situation hvor alarmen skal lyde. True betyder at den skal lyde og false det modsatte.

```
var         byte        mode = 0
var         bit         alarm = false
```

Lysdioderne styres de fire følgende variabler, to til hver lysdiode. Bit-variablerne siger om lysdioden skal være tændt eller slukket. Det gør de kun hvis blinkvariablerne er 0. Hvis man sætter blinkvariablerne til et tal, så blinker den pågældende lysdiode indtil variablen er talt ned til nul. For at få en lysdiode til at blinke 3 gange sætter man blinkvariablen til 6.

```
var         byte        groen_blink = 0
var         byte        roed_blink = 0
var         bit         groen       = true
var         bit         roed        = false
```

Variablen `halv_sec_tick` er en variabel der aflæses i alle rutiner der bruger noget med tid. Den er true i det gennemløb hvor der er gået ½ sekund og falsk i resten af gennemløbene.

```
var         bit         halv_sec_tick
```

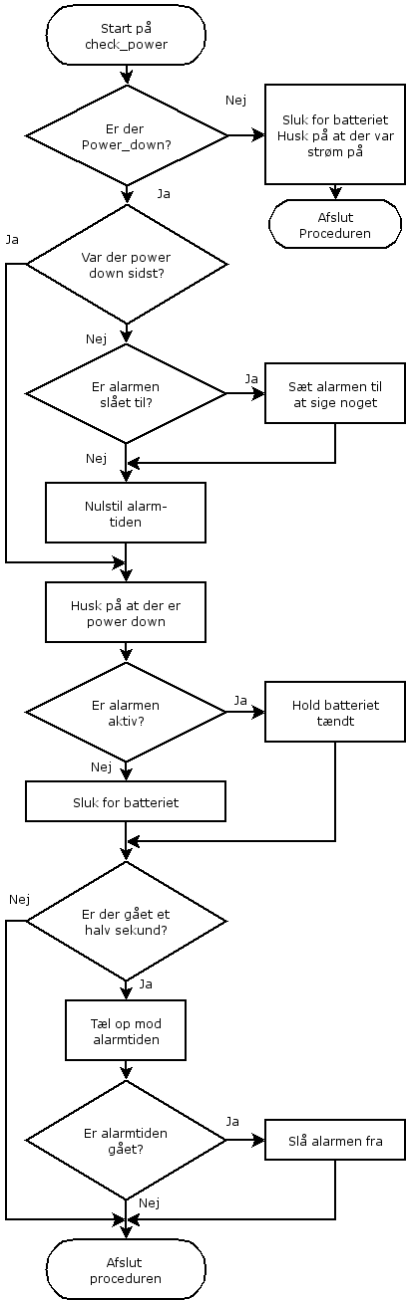
Håndtering af Batteriet og alarmen.

USB-power, wire og batteriet skal alle spille sammen med alarmen.

Proceduren check_power tester om der stadig er strøm på. check_power er illustreret på flowchartet her ved siden af.

Det man skal huske på, når man læser dette blokdiagram er at proceduren kaldes meget tit.

Den måde batteriet tændes på er at alarmen skal være aktiv (mode = 0). Den første gang man kommer igennem efter strømmen er forsvundet slås alarmen til, så den siger noget og alarmtiden nulstilles. Hvert halve sekund tælles tiden op indtil alarmtiden nås, så slås alarmen fra. Da der skal være alarm for at batteriet holdes tændt, så vil den slukke efter at alarmtiden er gået.



Koden til `check_power` ser ud som følger:

```
procedure check_power is
     if power_down then -- Der er ikke spænding fra USB-porten
          -- Hvis strømmen fejler, sæt alarm, om nødvendigt
          if last_power then
               if mode == 0 then
                    alarm = true
               end if
               alarm_count = 0
          end if
          last_power = false
     else
          battery = low -- sluk batteriet
          last_power = true
          return
     end if

     -- Hvis alarmen er slået fra, så sluk for enheden
     if mode == 1 then
          battery = low
     else
          battery = high
     end if

     if halv_sec_tick then
          alarm_count = alarm_count + 1
          -- sluk for batteriet og afstil alarmen , efter ventetid
          if alarm_count == alarm_tid then
               mode = 1
          end if
     end if
end procedure
```

I Proceduren `sound_alarm` sker der det, at lydgiveren tændes og slukkes hvert halve sekund, for at gøre mere opmærksom på sig selv, hvis der er alarm. Hvis der er strøm fra USB-porten, så tælles der på alarmtiden, så alarmen bliver slået fra.
Hvis wiren brydes, så er der alarm.

```
-- Få alarmen til at sige noget, og håndter Wire-inputtet
procedure sound_alarm is
     if alarm then
          if halv_sec_tick then alarm_out = ! alarm_out end if
          if ! power_down then
               if halv_sec_tick then
                    alarm_count = alarm_count + 1
                    if alarm_count == alarm_tid then
                         alarm = false
                         -- alarmen går igen hvis wiren stadig er afbrudt
                    end if
               end if
          end if
     else
          alarm_out = false
     end if
     if (! wire) & (! alarm) then
          alarm = true
     end if
end procedure
```

Tiden i softwaren.

Mange ting i softwaren er baseret på tid.

Nogen dele skal kaldes så tit som muligt fordi de skal kunne reagere hurtigt. Andre dele er baseret på halve sekunder. Alle ting bliver kaldt i hvert gennemløb, som det ses her i hoved-loopet:

```
-- Kør i hovedloopet med de ønskede rutiner
forever loop
     styr_tid
     laes_taster
     LED
     check_power
     sound_alarm
end loop
```

Den måde det er lavet på er, at der tælles hvor mange gange der er løbet rundt i hoved-loopet, og der tælles op til 10 ms. Når der er gået 10 ms, så tælles en anden tæller op til ½ sekund. Når tælleren på et ½ sekund løber over, så sættes variablen `halv_sec_tick` til sand. Den er så kun sand gennem ét gennemløb af hovedloopet.
På den måde kan alle procedurer tjekke på `halv_sec_tick` og håndtere tiden ud fra det, f.x. kan man komme op på 1 minut ved at tælle til 120.

Den første tælling op til 10 ms er på 26. Denne værdi er fundet eksperimentelt ved at måle tiden på alarmen med et stopur, og så justere den værdi indtil det passer.

Man skal være opmærksom på at hvis man ændrer i programmet, så det tager mere eller mindre tid, så er det nødvendigt at rette på denne tæller, der er erklæret i toppen som følger:

```
-- Antal gange den skal rundt, for at der er gået 10 ms
const      count_til_10ms = 26
```

Selve proceduren `styr_tid` der holder styr på de forskellige dele i tids-håndteringen ser ud som følger:

```
-- Procedure der sørger for at halv_sec_tick er true i eet gennemløb
hvert halve sekund
procedure styr_tid is
     halv_sec_tick = false
     count_10ms = count_10ms + 1
     if count_10ms == count_til_10ms then
          count_10ms = 0
          count_halv_sec = count_halv_sec + 1
          if count_halv_sec == 50 then
               count_halv_sec = 0
               halv_sec_tick = true
          end if
     end if
end procedure
```

Tastatur.

Aflæsningen af tastaturet sker ved hjælp af tast-modulet fra el-teknik hjemmesiden[11] hvor modulet ligger som en ZIP-fil.

Modulet hentes ind med følgende kodelinie:

```
include tast    -- Biblioteket med interface til multilplexed tastatur
```

I tast-modulet ligger endnu en include-fil, hvor definitionerne på tasterne ligger. Det er her der defineres hvilke I/O ben der skal anvendes. Der skal også defineres direction på alle benene. Det ser ud som følger:

```
var   bit   Mux_A       is   pin_a2   -- første multiplexer output
var   bit   Mux_B       is   pin_a3   -- andet multiplexer output
var   bit   Mux_C       is   pin_a4   -- tredie multiplexer output
var   bit   Sense_1     is   pin_b4   -- Sense-ledning på første række
var   bit   Sense_2     is   pin_b5   -- Sense-ledning på anden række
var   bit   Sense_3     is   pin_b6   -- Sense-ledning på tredie række
var   bit   Sense_4     is   pin_b7   -- Sense-ledning på fjerde række

-- HUSK også at rette diretion benene !!!
var   bit   Mux_A_dir     is   pin_a2_direction
var   bit   Mux_B_dir     is   pin_a3_direction
var   bit   Mux_C_dir     is   pin_a4_direction
var   bit   Sense_1_dir   is   pin_b4_direction
var   bit   Sense_2_dir   is   pin_b5_direction
var   bit   Sense_3_dir   is   pin_b6_direction
var   bit   Sense_4_dir   is   pin_b7_direction
```

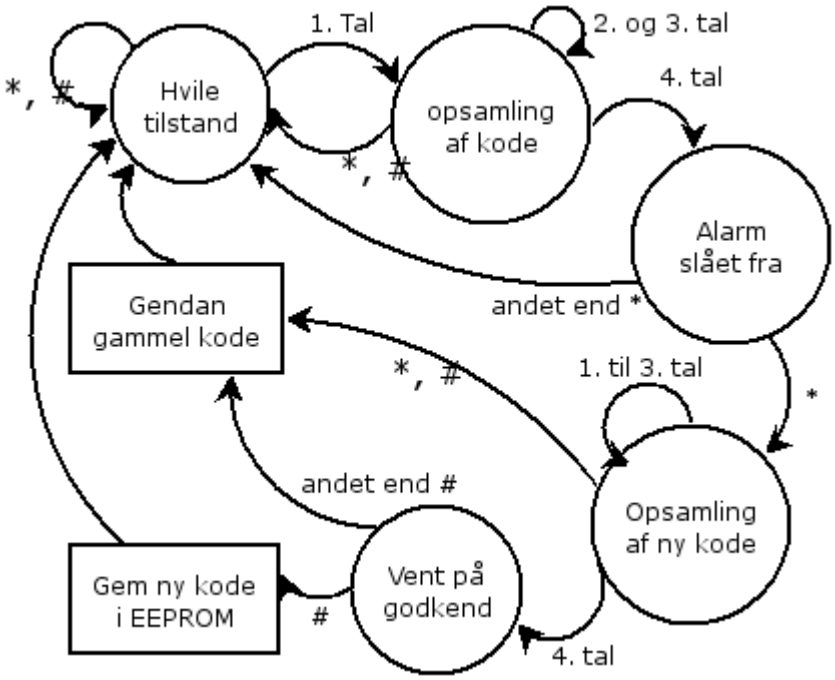
Nede i koden kaldes modulets kode med følgende linie, hvor variablen temp får tasteværdien, så man ikke mister en indtastning, da den kun kommer en gang ved hvert tryk på tasten:

```
  temp = tast
```

Det er svært at beskrive forløbet i koden, men hvis man beskriver det der sker omkring indtastningerne som tilstande, så giver det mere mening.
Cirklerne i diagrammet er de forskellige tilstande programmet kan være i, hvor programmet starter i hviletilstanden, når der PIC'en starter. Herefter er det indtastningerne der ændrer de forskellige tilstande.

Ved korrekt indtastning kan man se, at man kan slå alarmen fra ved at taste de 4 rigtige tal i koden, så man ender i tilstanden med alarmen slået fra.
Når man har gjort det, så kan man indtaste * og så 4 nye tal, der så vil blive den nye kode. Koden skal godkendes ved at man taster #, så koden gemmes. Hvis man taster noget forkert i denne sekvens, så bliver den gamle kode reetableret og den ender i hviletilstanden igen, hvor alarmen også er aktiv.

[11] El-teknik http://www.holstebrohtx.dk/bar/index.php?pkt=1371630 Set 27/11 2005



Det eneste der ikke vises i dette tilstandsdiagram er, at hvis man venter for lang tid (30 sekunder) i en eller anden tilstand, så vil den ende i hvile-tilstanden. Alarmen vil således blive slået til igen, hvis man ikke gør andet. Dette klares af variablen passiv_count.

LED.

Som beskrevet under de variabler der anvendes på tværs, så er der 4 variabler der styrer de to lysdioder.

Proceduren der håndterer lysdioderne ser ud som følger:

```
-- Håndtering af rød og grøn lysdiode
procedure LED is
     -- blink-håndtering på begge lysdioder
     if halv_sec_tick then
          if roed_blink > 0 then
               roed_blink = roed_blink - 1
               roed_LED = ! roed_LED
          end if
          if groen_blink > 0 then
               groen_blink = groen_blink - 1
               groen_LED = ! groen_LED
          end if
     end if

     -- eller konstant niveau
     if groen_blink == 0 then groen_LED = groen end if
     if roed_blink == 0 then roed_LED = roed end if
end procedure
```

Som det blev beskrevet ved variablerne, så er det blink-variablen der har højest prioritet. Sættes de til et tal, så vil lysdioden blinke indtil blink-variablen er talt ned til 0 med ½

sekunds skridt. Det vil sige at skal der blinkes i 10 sekunder, så sættes den til 20, go så går resten af sig selv.

Når lysdioden er færdig med at blinke, så er det de to andre variabler der bestemmer om lysdioden lyser eller ej.

Lagring af pin-koden.

Som omtalt så bliver pin-koden lagret i PIC'ens EEPROM. Den første pin-kode der ligger i programmet er lagret i starten på følgende linie:

```
Pragma eedata 1, 2, 3, 4          -- Koden der brændes i som default
```

Når PIC'en starter efter en reset, så læses den kode i EEPROM'en, det gøres i følgende procedure, så den lægges i kodevariablerne:

```
-- Procedure der henter det gemte password fra EE-PROM
procedure read_EE_kode is
     eeprom_get(0, kode_1)
     eeprom_get(1, kode_2)
     eeprom_get(2, kode_3)
     eeprom_get(3, kode_4)
end procedure
```

Den procedure kaldes også hvis brugeren af en eller anden grund fejler i indtastningen af en ny pinkode, fordi den nye kode bliver lagret i kodevariablerne mens indtastningen af den nye kode sker.

Hvis det går godt med at indtaste en ny kode, så skal koden gemmes i EEPROM'en, og det gøres i den følgende procedure:

```
-- Procedure der gemmer det nye password i EE-PROM
procedure new_password is
     eeprom_put(0, kode_1)
     eeprom_put(1, kode_2)
     eeprom_put(2, kode_3)
     eeprom_put(3, kode_4)
     alarm_aktiv
     groen_blink = 10
end procedure
```

Test.

Produktet er blevet testet løbende, specielt under udviklingen af softwaren.
Funktionerne er testet ud fra de ønskede funktioner, og der er også lagt vægt på at testen har omfattet fejlbetjening.

Videreudvikling.

Ved afsluttet udviklingsforløb, så er der selvfølgelig altid ting man ved der kan gøres bedre.

Watchdog i softwaren.

En af de relativt simple ting der kan laves er at der mangler en watchdog i softwaren.

Princippet i en watchdog er at man relativt tit skal udføre en instruktion der holder "hunden" vågen. Hvis man ikke gør det, så resetter PIC'en.
Ideen i det er, at softwaren ikke kan låse nogen steder, uden at der genstartes.

Ladekredsløb.

Som omtalt under overvejelserne omkring batteridriften, så vil det være naturligt at erstatte det faste batteri med et opladeligt, specielt når det skal være noget der bare skal virker hele tiden.

Når det så også skal virke hele tiden, så er opladelige batterier ikke det letteste at håndtere, alt efter hvilken type batteri man ville vælge (NiCd, Nikkel Metalhydrid eller andet). Om man vil lade PIC'en arbejde med opladningen eller det kan løses rent elektrisk er lidt afhængigt af hvilken type det er, men det vil under alle omstændigheder betyde en større rekonstruktion af forsynings-kredsløbet.

Produktionsmodning.

Hvis USB-alarmen skal sættes i produktion skal hele produktet igennem møllen. Det ville være naturligt at lave bl.a. ladekredsløbet som beskrevet, men der ville også være god økonomi i at lave tingene om til SMD (overflademonterede komponenter), da det vil være produktionsmæssigt billigere, men også komponentprisen er lavere, hvis man snakker fornuftige stk.-tal

Konklusion:

Jeg har fået konstrueret en alarm der kan fungere i den prototypeversion jeg har ønsket.
Jeg har fået konstrueret og afprøvet alle funktioner.
Produktet kan selvfølgelig udvikles yderligere, og det vil helt klart være nødvendigt med en del produktudvikling, hvis alarmen skal sættes i produktion.
Softwaren har jeg næsten fået færdig til en produktions-version, med mindre der skal laves et ladekredsløb, der skal fungere via softwaren.

Kildeliste:

El-teknik. el.holstebrohtx.dk er den hjemmeside der anvendes i undervisningen på Holstebro HTX. Der er samlet undervisningsplaner, undervisningsnoter, datablade og en del teori omkring el-teknik. Sitet opdateres af Bent Arnoldsen der underviser i el-teknik.

JAL Edit. www.sunish.net er en personlig hjemmeside for en inder, der bl.a. har udviklet en editor til JAL sproget.

Microchip. www.microchip.com er hjemmeside for dem der producerer PIC kredsen vi anvender, og et meget bredt udbud af andre microcontrollere.

Wouter van Ooijen. http://www.voti.nl/e_index.html er en hjemmeside, hvor jeg først og fremmest har hentet JAL-compileren. Wouter har en stor erfaring inden for PIC og programmering af dem. Denne erfaring øser han ud af på hans hjemmeside, sammen med at han har en webshop, hvor han handler med forskellig elektronik, specielt omkring PIC'en.

Bilag 1.
Samlet Diagram.