CHALMERS

Firmware till en Dyklampa
- Utveckling av programvara

Firmware for a dive light
- Developing software

Examensarbete inom Mekatronik

MIKAEL ISAKSSON

MICHAEL GUSTAFSSON
Institutionen för Signaler och System

Examinator Bertil Thomas

Avdelningen för Reglerteknik, Automation och Mekatronik

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2014

Förord

Detta examensarbete innefattar 15p och pågick från november 2013 till slutet av januari 2014. Företaget Nanight sökte efter personer som kunde fortsätta utvecklingen av ett av deras projekt, en egenutvecklad dyklampa. Då detta examensarbete handlar främst om programmering passade det vår utbildning. Vi läser till högskoleingenjörer inom Mekatronik på Chalmers tekniska högskola. Utbildningen involverar programmering i olika språk men även mekanik och elektronik. Utbildningen är så bred för att vi ska kunna se helheten av en produkt och inte bara enskilda delar. Detta ger ett helhetsbegrepp kring produktutveckling med mekaniska system som styrs av programmerbar elektronik.

Vi vill båda rikta ett stort tack till alla, både på Nanight och på Chalmers, som varit hjälpsamma i utformningen av arbetet.

Ett särskilt tack går till vår handledare på företaget Ulf Backudd, men även Per Ohlson och Anders Nilsson som var mycket hjälpsamma när vi körde fast.

Ett stort tack går även till vår handledare på Chalmers Göran Hult som kom med värdefulla synpunkter och idéer.

Av Mikael Isaksson och Michael Gustafsson

Sammanfattning

Företaget Nanight håller på att utveckla en dyklampa, och de sökte efter någon som kunde utveckla mjukvaran till projektet. Hårdvaran som användes var en mikroprocessor av typen PIC10F322. Processorn ska skicka ut en pulsbreddsmodulerad signal, dvs. en signal som slås av och på inom ett bestämt intervall, för att hålla lampan tänd. Lampan ska leverera maximalt ljus och dra så lite ström som möjligt. Programmet ska även kunna ta reda på vilken batterityp som används, och med ledning av det varna användaren i god tid innan batterierna tar slut. För att ytterligare spara på batterierna och förlänga livslängden på dem så ska processorn gå in i ett strömsparläge som kraftigt reducerar den förbrukade strömmen.

För att genomföra detta utfördes flera tester på olika batterityper i detta projekt Alkaliska, NiMH samt Li-Ion, för att få fram hur spänningen varierade under urladdningen. Det visade sig att de olika batterierna hade tillräckligt skild urladdningskarakteristik så att de kunde skiljas åt genom att mäta deras polspänning när de sattes in i lampan. Efter det gjorde programmet kontinuerliga tester av polspänningen för att se om värdena hade sjunkit under de rekommenderade värdena baserat på vilken typ av batteri som användes. Om detta hände började lampan blinka ungefär en gång per minut.
Olika ljusstyrkor på lampan jämfördes för att ta fram ett bra förhållande på förbrukad ström kontra ljusstyrka. Efter flera tester med den riktiga hårdvaran visade det sig att med en PWM-pulskvot på 75% fick vi bra upplysning med låg strömförbrukning.

De uppgifter som var satta som mål för programmet att kunna utföra uppfylldes.

Kostnadsoptimering har inte behandlats i rapporten. Ej heller anpassning av elektronik eller mekanik.

LED, PWM, lampa, A/D omvandling, avbrott

Summary

The company Nanight is in the process of developing a diving lamp, and they were searching for someone to develop the software for the project. The hardware used was a microcontroller PIC10F322. The microcontroller will send out a pulse width modulated signal, ie a signal that alternates between on and off, to keep the lamp lit. The lamp will deliver as much light as possible while using as little power as possible. The program should also be able to determine which battery type that is being used, and using that information to warn the user before the batteries are depleted. To save the power of the batteries even further and prolong their life the microcontroller will go into a power save mode that will reduce the used power significantly.

To be able to see how the voltage changed during the discharge of the batteries, several tests was conducted. It turned out that the different batteries had different enough discharge characteristics so they could be differentiated by their terminal voltage when they were inserted into the lamp. After that the program did continuous test of the terminal voltage to see if the voltage had sunk below the recommended minimum values for the different batteries. If that happens, the lamp will start blinking about once a minute.

Different brightness produced by the lamp was compared to get a good grip on the relationship between brightness and used current. After several tests on the real hardware we found that the best was when the duty cycle of the PWM was at 75%.

We did not handle any economics, like the cost of components. We also didn’t modify any electronics.

LED, PWM, light, ADC, interrupt

Innehållsförteckning

Beteckningar

MCU - Microcontroller
PWM - Pulse-Width-Modulation
ADC - Analog to Digital Converter (AD Omvandlare)
SoC - State of Charge
PR2 - Heltal som Timer2 räknar upp till
PWM1DCH - Register som innehåller pulskvot som ett binärt tal
Fosc – Mikroprocessorns frekvens
Tosc - 1/FOSC
NOP – No Operation, en rad programkod som inte gör någonting
Lux – SI-enheten för illuminans (belysning)

1 Inledning

1.1 Bakgrund

Under nattdyk bör man ha minst en lampa per person med sig då sikten är mycket begränsad. Även om man dyker under dagen kan det behövas en dyklampa då ljusnivån under vatten snabbt kan förändras. Det kan även finnas mörkare utrymmen som behöver extra ljus för att kunna utforskas. Dyklampor kan vara dyra och sakna en del praktiska funktioner. Därför håller konsultföretaget Nanight på att utveckla en dyklampa från grunden. De var av åsikten att de dyklampor som kan köpas på dagens marknad var dyra, och att de kunde bygga billigare och bättre lampor själv. De har dock inte tid att jobba med den på heltid så de sökte efter personer som kunde hjälpa till att utveckla främst mjukvaran till lampan.

1.2 Syfte

Detta uppdrag består av att ta fram mjukvara till den mikroprocessor som styr lampan så att man med hjälp av pulsbreddsmodulering kan få så starkt ljus som möjligt samtidigt som strömförbrukningen minimeras för längsta möjliga drifttid. Mikroprocessorn ska även mäta batterinivån och ge en varning genom att t.ex. blinka lampan för att varna användaren en stund innan det är dags att byta batterier. Det ska även finnas funktioner för att bestämma ljusstyrkan genom att använda på/av knappen.

1.3 Avgränsningar

Vi har inte gå in närmare på den ekonomiska delen av utvecklingen. Kostnaden för de olika komponenterna samt om lampan kan kostnadsoptimeras behandlades inte. Vi kommer inte heller behandla utvecklingen av plast och metalldetaljer till lampan. Vi har heller inte berört de elektroniska komponenterna eller försökt rita om el-schemat. De mekaniska komponenterna på lampan har inte behandlats.

1.4 Precisering av frågeställningen

I samarbete med Nanight togs ett antal mål fram som ska uppnås.

1. Lampan ska drivas av en PWM-signal och dra så lite ström som möjligt, men leverera tillräckligt mycket ljus.
2. Processorn ska kunna ta emot en analog insignal och omvandla den till ett digitalt värde för att kunna beräkna batteriets spänning.
3. Processorn ska kunna gå in i ett strömsparläge och dra mycket lite ström när lampan inte används.
4. Processorn ska kunna väckas från strömsparläget via ett knappavbrott.
5. Lampan ska leverera optimalt upplevt ljus.
6. Processorn ska kunna analysera när batteriet börjar bli tomt, och återge det i någon form.
7. Processorn ska kunna känna igen batteritypen som används och ändra inställningar med ledning av det.
8. Vid påsättning av lampan ska ljuset dimmas på, vid avstängning ska lampan dimmas av.

2 Teori

2.1 Allmänt om ljusstyrka

Människans öga kan bara uppfatta ljus med våglängder mellan 380 nm och 770 nm. Därför bör en lampa som används för upplysning producera så mycket ljus som möjligt inom detta intervall.
Lampan ska producera så mycket synligt ljus som möjligt med så lite förbrukad energi som möjligt
Ljusutbyte är ett mått på hur bra en ljusskälla är på att producera synligt ljus. Det är en kvot mellan ljusflöde och ineffekt. Vi vill uppnå ett högt ljusflöde med en låg ineffekt.

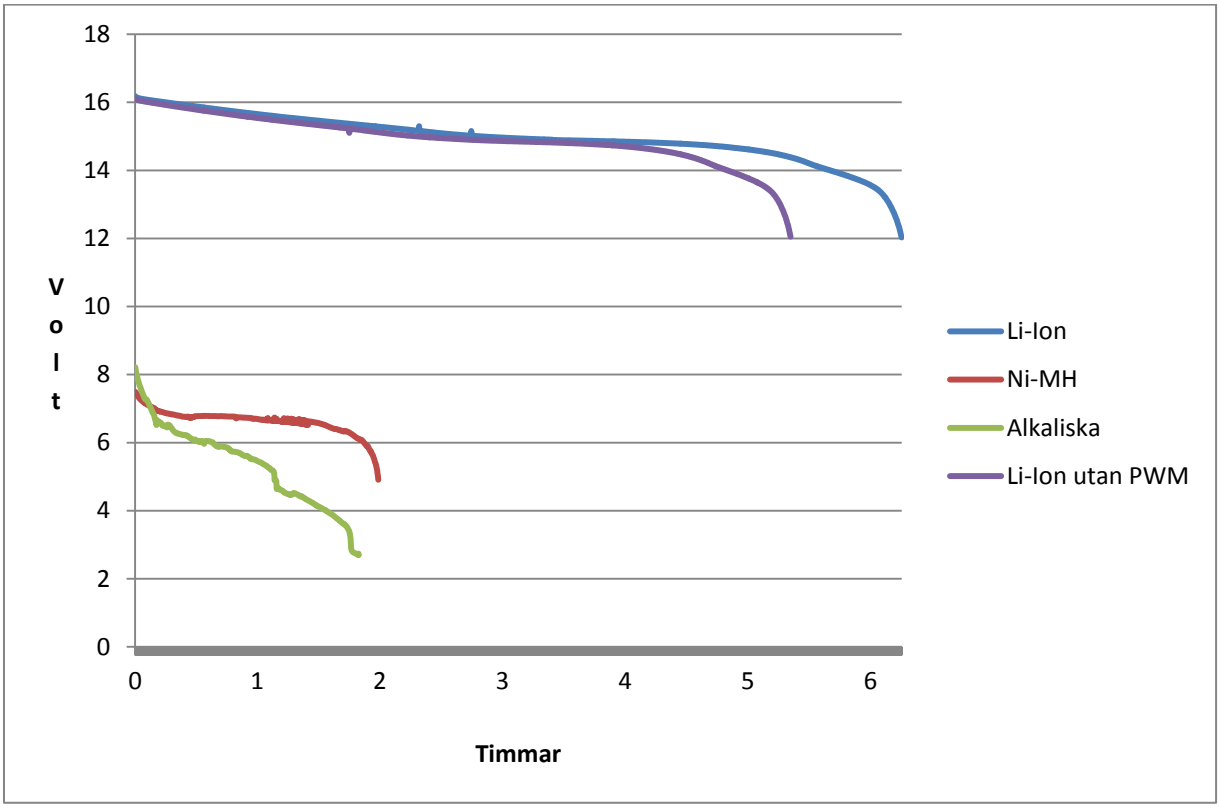
De lampor som finns på marknaden har oftast ljusstyrka angiven i lumen, ett sådant värde har inte kunnat tas fram eftersom det kräver mätning i en s.k. integrationssfär. Sådan utrustning är mycket dyr så för att utföra ett sådant test måste man vända sig till t.ex. Sveriges Tekniska Forskningsinstitut (SP).

2.2 Avläsning av State of Charge

Det finns flera metoder för att beräkna ett batteris State-of-charge (SoC) varav två var aktuella att använda.

Direkt strömmätning
Denna går ut på principen att laddningen som finns i batteriet är lika med strömmen som flödade in vid laddning multiplicerat med tiden som laddningen tog. Detta fungerar enklast om laddningen och urladdningen av batteriet är linjära dvs då ingående och utgående ström är konstant. Det går även att beräkna batteriets laddning om strömmen varierar. Då används tidintergralen av strömmen.[1] Det kan även skapa problem att använda en timer då dessa kan ändras vid en återställning av programmet eller processorn. I så fall tappar man räkningen på hur lång tid lampan har lyst och därmed kan man inte längre göra en bra uppskattning på kvarvarande lystid.

Spänningsbaserad uppskattning
I denna metod används batteriets polspänning som ett enkelt mått på kvarvarande laddning i batteriet. Då flera olika faktorer så som temperatur, batteriets ålder, urladdningshastighet och aktuell spänning kan göra så att batteriets polspänning varierar behövs dessa kompenseras för. Ett problem med denna metod är att det är svårt att uppskatta hur mycket laddning som finns kvar i batteriet eftersom de är designade att ge en så konstant spänning som möjligt. Lösningen på detta problem blir att göra urladdningar på de olika batterierna för att kunna avgöra vid vilken spänning som batteriet övergår från att urladdas relativt linjärt till att dyka mer och mer.[1]



Figur 2.1 Urladdning av olika batterityper

Denna metod valdes eftersom hårdvaran var utformad på så sätt att en femtedel av batterispänningen kunde läsas av på ett av processorns ben. Detta möjliggjordes via en spänningsdelare (se bilaga 2 för elschema). Batterispänningen kunde på så sätt enkelt användas i programmet. Det krävdes ej heller komplicerade uträkningar eller någon timer.

2.3 Allmänt om hårdvara, mjukvara och utvecklingsmiljö

Programmet som tillhandahölls i början av projektet var en skelettkod med bara de viktigaste definitionerna skrivna, samt ett kort program som kunde blinka en lampa på en specifik port med en viss frekvens.
För att testa programmet samt utvecklingsmiljön byggdes en enkel krets bestående av processorn, några motstånd samt en lysdiod kopplad till en port.

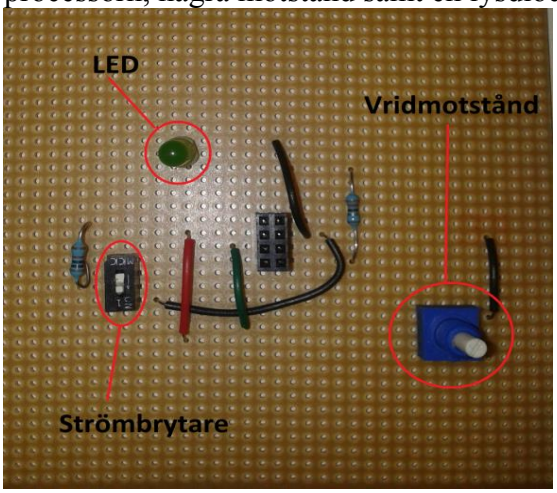


Bild 1 Testhårdvara

Programmet MPLAB 8.92 med kompilatorn XC8 användes för att skriva och kompilera koden.

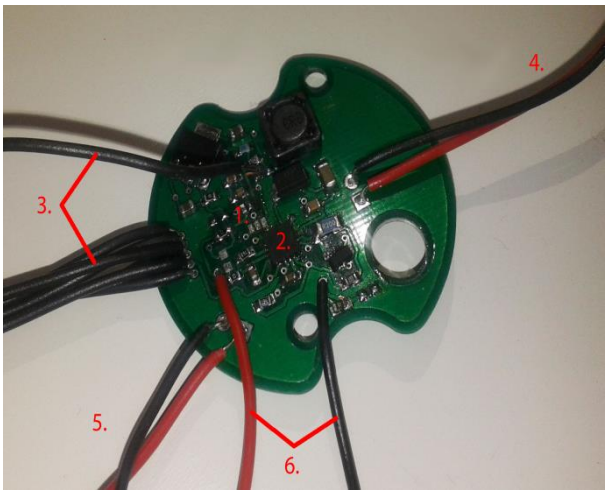


Bild 2 Prototyphårdvara

För att testa koden mot hårdvaran utan att använda själva processorn användes debuggern MPLAB ICD 3. För mer avancerad testning användes ett prototypkretskort enligt bild 2.

1. Här ska processorn sitta
2. DC/DC till LED
3.Kablarna från debuggern som går till de olika benen på processorn
4.Kablar till LED
5.Strömförsörjningskablar
6.Kablar till strömbrytare

För att testa de olika batterityperna användes Voltcraft 8500 som är en apparat som kan ladda och ladda ur olika typer av batterier på ett säkert sätt. Den kan även kopplas till en dator för att få grafer på urladdningen/laddningen över tid. Då vi arbetade både med NiMh- och Li-Ionbatterier som kan skadas om de urladdas under ett visst värde var detta en värdefull tidsbesparare då batterierna inte behövde övervakas under urladdning.

2.4 Olika batterityper

Lampan ska kunna använda flera olika typer av batterier. Dessa tre är de vi har inkluderat i mikroprocessorns programmering.

NiMH står för nickel-metallhydrid och är ett återuppladdningsbart batteri som började utvecklas i slutet av 1960 talet. Sedan 1991 har deras kapacitet fördubblats och deras livslängd ökats. Spänningen över batteriet är ungefär 1,2V vid användning. Det gäller dock att inte urladda dessa batterier till en för låg spänning, ungefär 1,1V-0,9V, då de kan bli skadade och deras kapacitet försämras.[3]

Li-Ion står för litium-jon och är ett återuppladdningsbart batteri. Det började utvecklas redan 1912 men användes inte kommersiellt förrän tidigt 1970 tal. Spänningen är ungefär 3,6V vilket gör att man behöver ett mindre antal celler än t.ex. alkaliska batterier för att uppnå samma spänning. Dessa batterier behöver förses med en skyddskrets för att förhindra överladdning eller att spänningen i batteriet blir för låg då detta kan resultera i skador på batteriet.[2]

De alkaliska batterier som användes kan inte laddas igen och deras spänning är cirka 1,5V men sjunker vid användning.

3. Metod

Hela arbetet utfördes på Nanights kontor i Göteborg där det fanns möjlighet till en bra dialog med handledaren på företaget, och det fanns även tillgång till en prototypverkstad där vi kunde bygga enkla testkretsar. Den hårdvaran som skulle användas fanns också där. Närheten till företaget gjorde så att inblandade parter enkelt kunde hållas uppdaterade om framstegen i projektet samt att en bra dialog kunde föras om utvecklingen av de olika programfunktionerna.

Arbetet började med att sätta sig in i ritningen till hur hårdvaran såg ut samt bli bekanta med den utvecklingsmiljö som skulle användas för att skriva koden till projektet. Detta gjordes genom ett enkelt färdigskrivet program.

Utvecklingen fortsatte med en egenutvecklad krets för snabb kodtestning. Den bestod av en port för debugheadern, en lysdiod, en strömbrytare och en potentiometer för att kunna variera spänningen på en port. Syftet med att variera spänningen var att kontrollera AD-omvandlarens funktion.

För att kunna testa koden mot den riktiga lampan kopplades debugheadern mot ett kretskort utan processor.

När en funktion fungerade som den skulle utökades programmet med ytterligare funktioner.

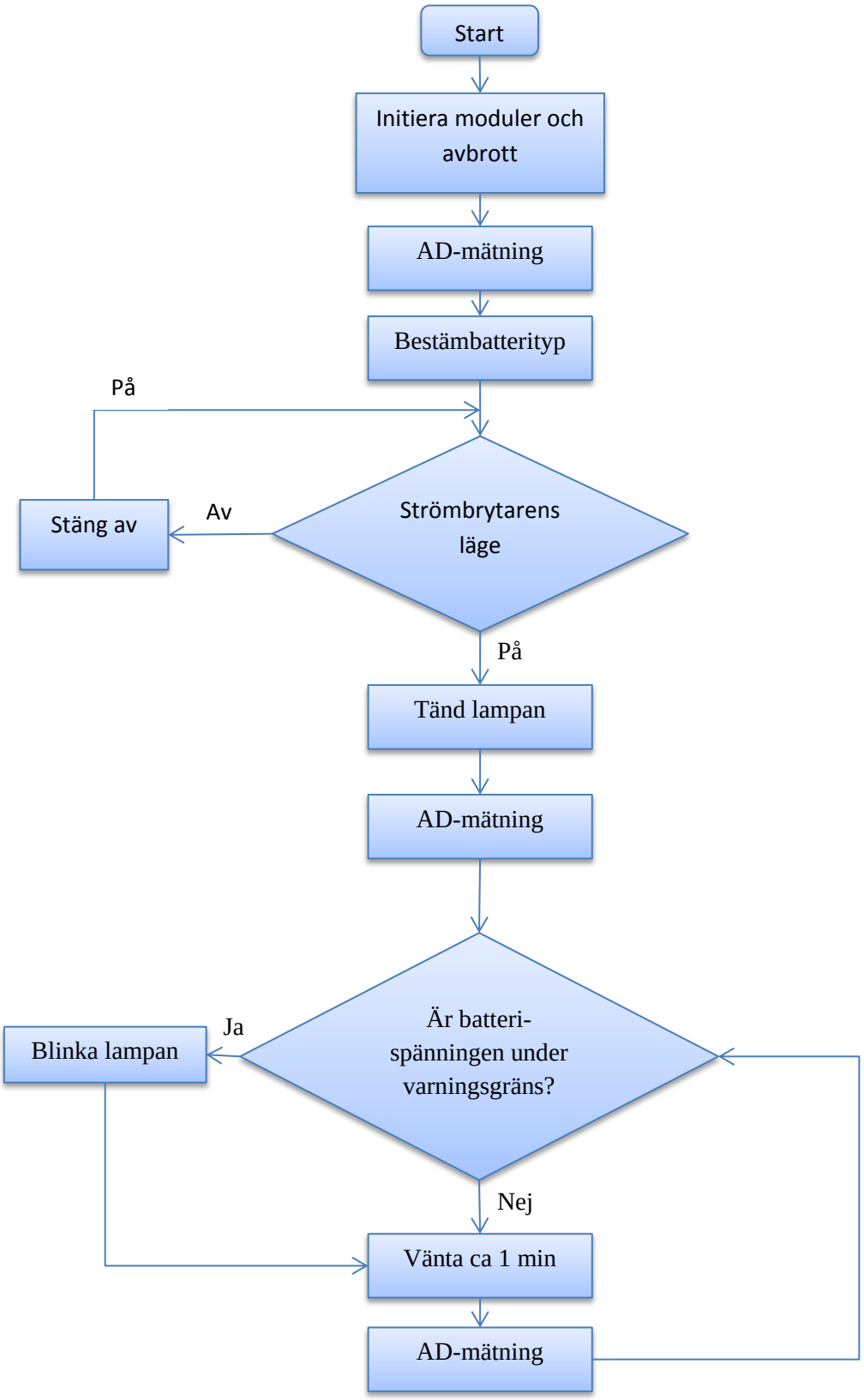
I vissa steg behövdes små tester göras för att jämföra ljusstyrkan på lampor. Ett kontrolltest för att se om ljusstyrkan ökade linjärt gjordes. Ett efterföljande experiment i ett mörkt rum gjordes för att få en uppskattning på upplevd ljusstyrka och om fullt påslagna ljus gav lika mycket.

För att ta reda på när olika typer av batteri närmar sig urladdning så genomfördes en rad urladdnings tester på NiMH, Li-Ion och alkaliska batterier. Dessa tester utfördes på den riktiga hårdvaran med samma pulskvot på PWM-funktionen. En kort test genomfördes även för att se hur strömförbrukningen skilde sig åt om lampan lyste genom en kontant spänning istället för en PWM-signal

4. **Programmets uppbyggnad**

4.1 Programflöde

När processorn slås på genom att ett batteri sätts in eller att strömbrytaren slås på kommer den först att aktivera sin PWM-funktion, AD-omvandlare och avbrottsfunktion. Om strömbrytaren är i av-läge kommer processorn omedelbart gå in i lågenergiläge. Om strömbrytaren är i på-läge kommer lampan slås på och en mätning av batterispänningen kommer utföras för att avgöra vilken typ av batteri som driver lampan samt vad det aktuella batteriets polspänning är. Under drift kommer batterispänningen mätas periodiskt för att utifrån vilken typ av batteri som används i god tid kunna varna användaren. Detta görs genom att blinka en gång per minut ifall spänningen blir för låg. Under tiden detta pågår väntar avbrottsrutinen på ett avbrott genererat från lampans strömbrytare för att kunna stänga av lampan oavsett vad programmet håller på med.



Figur 4.1 Programflöde

4.2 PWM-styrning

Signalen till lampan ska vara av typen PWM dvs. pulsbreddsmodulerad, vilket betyder att den alternerar mellan på och av inom ett bestämt intervall. Detta sparar ström, vilket leder till ökad batteritid. Den upplevda ljusstyrkan kan även lätt bestämmas genom att variera perioden då lampan är på respektive av.

Arbetet började med att bestämma en pulskvot, vilket är hur stor del av varje period som lampan är påslagen. Detta gjordes genom att använda formler hämtade från processorns datablad. enligt ekv 1,2 och 3.

Ekv 1 $PWM - Period = (PR2 + 1) * 4 * Tosc * (TMR2\ Prescale\ value)$ ekv 1

Ekv 2 $Pulsbredd = (PWM1DCH{:}PWM1DCL < 7{:}6 >) * Tosc * (TMR2\ Prescale\ value)$

Ekv 3 $Pulskvot = \frac{(PWM1DCH:PWM1DCL<7:6>)}{4*(PR2+1)}$

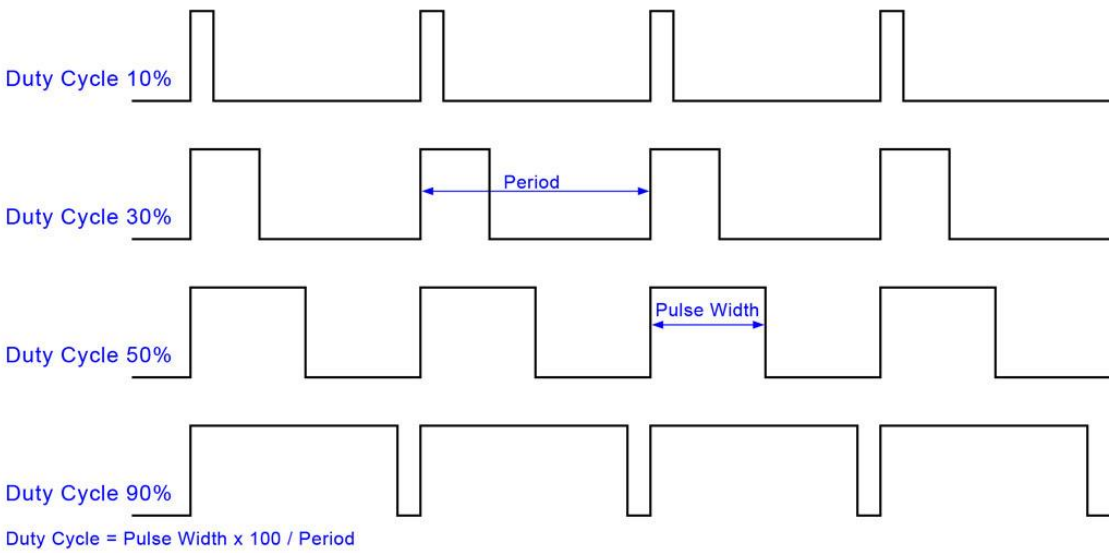
Tosc = 1/Fosc

PR2 = 255

Med Fosc = 1 MHz så har vi Tosc = 1/1MHz = 0,000001s

vilken ger en PWM-period på 256 * 4 * 0,000001 * 4 = 0,004096s
Eftersom frekvensen är 1/periodtiden, f = 1/T, så är frekvensen som lampan blinkar med 1/0,004096 = 244 Hz
Denna höga frekvens gör att människans öga inte upplever något flimmer utan bara ett stadigt ljus (4)

max pulsbredd är 255 * 0,000001 * 4 = 0,00102s



Figur 4.2 Förklaring av begreppen Period, Pulskvot och Pulsbredd

För att PWM-modulen ska fungera optimalt och kunna styras enkelt behöver den initieras på ett specifik sätt enligt den aktuella processorns datablad. (5). Den processor som användes, PIC10F322, initierades på följande vis:

1. Sätt den använda pinnen, i detta fall RA0, till input
2. Nollställ PWM1CON registret.
3. Ladda PR2 registret med PWMs periodvärde
4. Nollställ PWM1DCH och PWM1DCHL
5. Konfigurera och starta Timer2:
   Nollställ TMR2IF, Timer2s interruptflagga
   Lägg in Timer2s prescalevärde i T2CKPS
   Starta Timer2 genomn att ettställa TMR2ON
6. Sätt på RA0 pinnen och vänta tills Timer2 overflowar då TMR2IF sätts
7. Sätt RA0 till output och ettställ PWM1OE
8. Starta PWM-modulen genom att ettställa PWM1EN

Detta behövde konverteras till C kod i en initieringsfunktion.

```
//Initierar PWM
void InitPWM(void)     //Initiera PWM
{
TRISA  = 0x01;   // Stäng av output på RA0
PWM1CON = 0x00;    // Nollställ PWM1Con
PR2   = 0xFF;      // Ställ in period på Timer2
PWM1DCH = 0x00;    //Nollställer båda duty-cycle registerna
PWM1DCL = 0x00;
TMR2 = 0x00;    //Nollställer timer2
TMR2IF = 0;              //Cleara tmr2if flaggan
TMR2ON = 1;    //startar timer2
TRISA = 0x04;    //Slå på pwm pin output
PWM1EN = 1;    //Aktivera PWM-modulen
PWM1OE = 1;    //Aktivera output på PWM-modulen
T2CON = 0b00000101;         //bit0-1 sätter prescaler till 4
}
```

För att variera ljusstyrkan på lampan så ändrar man pulskvot. För att få mer ljus ökar man pulskvotn och för att få mindre så minskas den. För att testa om beräkningarna stämde och gav ett godkänt resultat användes kretsen i Bild 2. Kretsen fungerade som en förenklad version av den riktiga hårdvaran för att lättare kunna upptäcka fel och för att se om funktioner fungerade som det var tänkt.

4.3 Avbrottsrutin

Då programmet ska kunna utföra viktiga funktioner, som att försättas i och startas ur strömsparningsläge, medan andra programfunktioner körs, behövs en funktion som avbryter programmet och kör nödvändig kod. Processorn PIC10F322 har flera olika sätt att orsaka ett avbrott, t ex vid AD-omvandling, overflow på Timer0 där overflow i detta fall betyder att registret överskrider sitt maximala värde, externa avbrottskällor eller avbrott beroende på förändringar på portar.
Då oscillatorn som styr Timer0 är inaktiv i sleep enligt databladet kan Timer0 inte användas som en källa till avbrott. Ett externt avbrott kan heller inte användas då den bara finns kopplad till en specifik port, RA2, på mikroprocessorn som redan är kopplad till A/D-mätning. A/D-omvandlingsavbrottet är inte optimal att använda då vi måste starta en A/D omvandling varje gång vi vill väcka lampan. Det avbrott som har valts är avbrott beroende på förändringar vilket läser av om en specifik port har ändrat sitt värde från högt till lågt eller vice versa. Om detta har inträffat aktiveras en avbrottsflagga i IOCAF-registret som korresponderar mot den specifika porten i och som lätt kan avläsas i programmet. Detta fungerar även då mikroprocessorn är försatt i sleepläge. Detta avbrott valdes för att den kan kopplas till vilken port som helst på mikroprocessorn och är enkel att använda och kontrollera.

För att använda avbrott behöver de initieras. Detta görs på följande sätt:

```
IOCIE = 1;        //Aktivera avbrott vid lägesändring av brytare.
IOCAP3 = 1;           //Aktivera avbrott på stigande flank.
IOCAN3 = 1;       //Aktivera avbrott på fallande flank.
GIE = 1;          //Aktivera avbrott
```

Ökande eller fallande flank bestämmer när avbrottet utlöses. Vi har valt båda då vi vill att avbrottet ska kunna göras åt båda hållen med switchen som används. Ett problem som kan uppstå är att avbrott kan genereras flera gången om brytaren stabiliserar sig långsamt på ett värde.

För att vidare hantera avbrott så behövs en avbrottsfunktion som programmet hoppar till så fort ett avbrott har hänt. I den funktionen anropas de uppgifter som behöver utföras. Eftersom programmet inte fortsätter utföra sina andra uppgifter medan det befinner sig i avbrottsfunktionen så har programmet utformats så att avbrottsfunktionen anropar olika funktioner baserat på vad som utlöste avbrotten. Då ett knapp-avbrott används är det viktigt att vänta tills knappen är stabil innan man vidtar åtgärder eftersom en knapp kan svänga mycket mellan högt och lågt värde en period efter att den har ändrat läge. Denna korta väntetid bör inte göras i avbrottsfunktionen eftersom då hela programmet blir låst under den tiden. När sedan avbrottsrutinen är klar aktiveras möjligheten till avbrott igen.

```
//Avbrottsfunktion
void interrupt ISR()
{
    if(IOCAF3 == 1)//kolla om interrupt-on-change har triggat
    {
    IOCAF3 = 0;      //nollar interruptflaggan
    GIE = 1;         //aktiverar globala interrupts
    if(SWITCH == 0)          //är switchen i på läge?
        {
        wake_up();
        }
        if(SWITCH == 1)      //är switchen i av läge?
                        {
                        shutdown();
                        }
                }

}
```

4.4 A/D Omvandling

För att mäta batteriets polspänning och därmed typ och kvarvarande laddning användes processorns analog till digital-omvandlare(ADC) där spänningen på en viss port, i detta fall RA2, omvandlas till ett 8-bitars digitalt värde. Detta värde kan sedan användas för att t.ex. varna användaren när batterinivån börjar bli låg. Användaren ska varnas genom att lampan börjar blinka med jämna mellanrum när spänningen fallit under en viss nivå. För att A/D konverteringen ska fungera behövs en viss konfigurering. (5)

1. Konfigurera port genom att sätta den till input med TRISA-registret och sätt porten till analog med ANSEL-registret
2. Ställ in konverteringsklocka, inputkanal och slå på ADC-modulen.(ADCON)
3. Konfigurera ADC avbrott.
4. Vänta på att ADC-modulens kondensator laddas till samma spänningsnivå som inputkanalen.
5. Starta konverteringen genom att sätta GO/DONE-biten.
6. Vänta på att konverteringen är färdig genom att antingen vänta på att GO/DONE-biten nollställs eller att ADC-avbrottflaggan aktiveras.
7. Läser in resultatet i ADRES-registret.
8. Nollställ ADC-avbrottflaggan.

Detta skrevs i en initeringsfunktion som startade en konvertering på värdet som lästes in på port RA2. Det färdiga resultatet läggs automatiskt in i registret ADRES.

```
//AD-omvandlingsfunktion
void initAD(void)
{                       //1.Konfigurera portar
TRISA2 = 1;             //Sätt RA2 input
ANSA2 = 1;                      //Sätt RA2 till analog
ADCON = 0b00001001;    //2.Konfigurera ADC-modul, Fosc/2
                        //Välj ADC konverteringsklocka
                                        //Välj inputkanal för ADC
                        //Slå på ADC-modul
__delay_us(2);                  //4.Vänta nödvändig tid
GO_nDONE = 1;                   //5.Starta konvertering genom att sätta GO/DONE bit.
                        //6.Vänta på att ADC är färdig genom att
                        //   undersöka GO/DONE-bit
while(1)
   {
     if (ADIF == 1)             //Är AD omvandlarens avbrottsflagga satt?
      {
       ADON = 0;        //Om ja omvandlingen är klar, fortsätt
      break;
      }
   }
  ADIF = 0;             //8.Nollställ ADC avbrottsflagga
}
```

Så som hårdvaran är designad skall port RA2 på processorn utföra denna uppgift. För att testa om A/D-omvandlingen fungerade löddes en potentiometer fast på testhårdvaran enligt Bild 1 och ett kort program skrevs som varierade lysdiodens ljusstyrka beroende på resultatet av A/D omvandlingen. Programmet fungerade på så sätt att det 8-bitarsvärde som gavs från A/D-omvandlingen sattes som pulskvot för PWM-signalen vilket resulterade i en dimminingseffekt på LED lampan som lätt indikerade om A/D-omvandlingsfunktionen fungerade som planerat. AD-konverteringen lagras i ett register ADRES som kan användas även utanför konverteringsfunktionen.

ADC Clock Period (TAD)		Device Frequency (FOSC)			
ADC Clock Source	ADCS<2:0>	16 MHz	8 MHz	4 MHz	1 MHz
FOSC/2	000	125 ns(1)	250 ns(1)	500 ns(1)	2.0 µs
FOSC/4	100	250 ns(1)	500 ns(1)	1.0 µs	4.0 µs
FOSC/8	001	0.5 µs(1)	1.0 µs	2.0 µs	8.0 µs(2)
FOSC/16	101	1.0 µs	2.0 µs	4.0 µs	16.0 µs(2)
FOSC/32	010	2.0 µs	4.0 µs	8.0 µs(2)	32.0 µs(2)
FOSC/64	110	4.0 µs	8.0 µs(2)	16.0 µs(2)	64.0 µs(2)
FRC	x11	1.0-6.0 µs(1,3)	1.0-6.0 µs(1,3)	1.0-6.0 µs(1,3)	1.0-6.0 µs(1,3)

Figur 4.3 ADC väntetid

Diagram ovan från processorns datablad visar hur lång konverteringsklockan ska vara. Då processorn körs med 1 MHz och klockkällan är Fosc/2 så är den valda delaytiden 2 mikrosekunder.

4.5 Strömbesparning

Då lampan kommer vara avstängd en stor del av tiden så behövs en strömsparningsfunktion som låter processorn gå i vila och förbruka minimal ström så att batterierna inte laddas ur snabbt. Detta görs genom att anropa en inbyggd sleepfunktion i mikroprocessorn som stänger av de flesta funktioner. Detta minskar strömförbrukningen markant. För att minska strömförbrukningen ytterligare finns det vissa åtgärder man kan ta i programmet, bl.a. att stänga av funktioner så som ADC och att sätta alla portar till output och slå av deras pull-ups så de inte drar ström i onödan.
Vid de senaste testerna på riktig hårdvara var strömförbrukningen nere på cirka 30 µA i sleepläge. Om batteriet är på 2600 mAh ger detta en standbytid på strax under 10 år vid fulladdat batteri om man bortser från batteriets självurladdning.

Kompilatorn som användes har en inbyggd funktion för att försätta en processor i sleep, SLEEP(). Vad denna funktion gör är att anropa en assemblerfunktion, asm (“sleep”), som försätter processorn i strömsparläge.
För att testa om processorn verkligen gick in i sleep så kodades lampan att tändas precis efter sleepkommandot utfördes. Om lampan fortsätter att vara släckt har processorn gått in i sleep. Med den riktiga hårdvaran mättes även strömförbrukningen för att kontrollera om funktionen fungerade som den skulle.

Ett problem vi stötte på när vi utvecklade sleepfunktionen var att en komponent i hårdvaran förbrukade ström så att processorns strömförbrukning i lågenergiläget inte kunde mätas. Efter att viss elektronik bytts ut märktes en stor skillnad.
När programmet väcks ur sleep kommer det utföra nästa kommando, vilket är en NOP, samt hoppa till interruptrutinen för att bestämma vad som väckte processorn.

4.6 Ljusoptimering

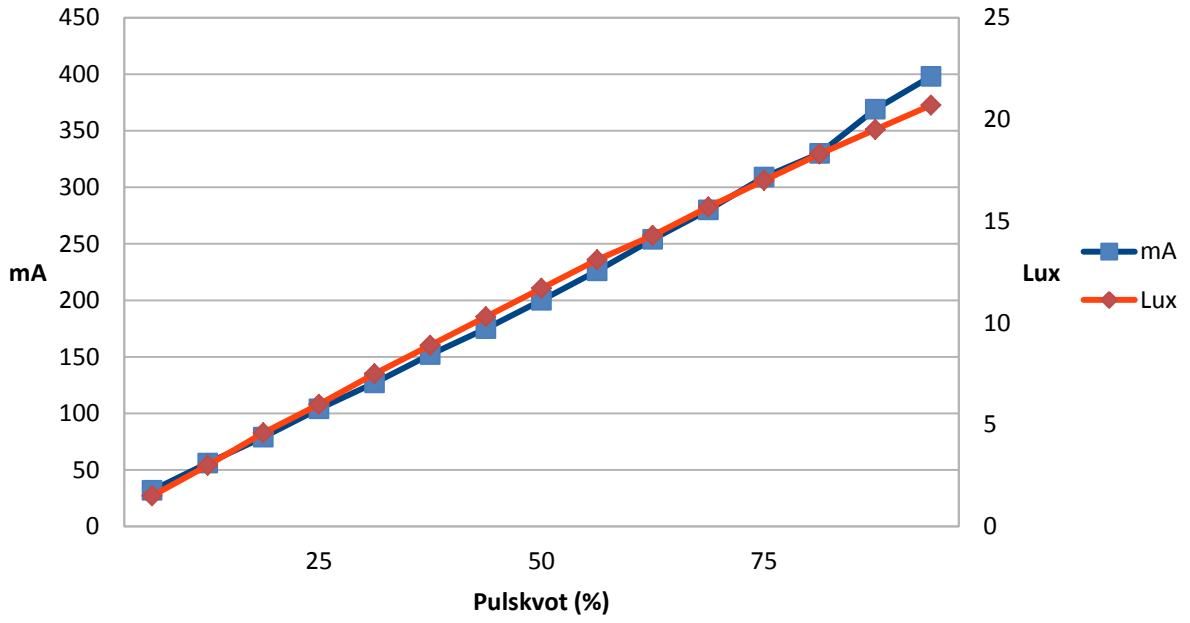
Ljuset från lampan ska optimeras så att en bra balans mellan strömförbrukning och ljusstyrka uppnås. LED-lampan som används ska ge tillräckligt mycket ljus med så låg strömförbrukning som möjligt. För att få en preliminär uppfattning av hur olika PWM-signaler påverkar det upplevda ljuset så gjordes två enkla experiment.

En lux-mätare placerades ungefär en cm ifrån lampans LEDs i ett upplyst rum. Detta för att snabbt kunna jämföra de olika ljusstyrkorna mot varandra. I detta korta test visade det sig att ökningen i belysning var uppskattningsvis linjär enligt figur 4.4.

Ett uppföljande experiment gjordes där lampan riktades mot en LUX mätare på en vägg 25 cm bort i ett upplyst rum. Detta för att se om resultaten från det första testet stämde överens med de nya resultaten, vilket de gjorde. Dessa experiment ger inga bra värden på hur mycket lampan själv lyser upp ett rum, utan syftet var att kunna få en snabb jämförelse av hur de olika ljusnivåerna förhåller sig till varandra. Vi kunde även se att strömförbrukningen inte var helt linjär när vi ökade lampans ljusstyrka, utan var högre mot slutet.

Då vi vill ha hög ljusutbyte kontra låg strömförbrukning så kan vi se att det inte lönar sig att vara vid max ljusintensitet utan lite under då det ger mer upplysning men för mindre ström än max.

Ett test genomfördes där lampan ställdes på golvet i ett mörkt rum och riktade den mot taket, bredvid lampan ställde vi en ljusmätare. Vi stegade sedan upp pulskvot med 10 hexadecimalt åt gången för att få fram en kurva där ljusstyrka och strömförbrukning kunde ställas i relation till varandra. Vi kom fram till att pulskvot lämpligast bör ligga vid 70-90%, vilket är mellan B0 och E0. Den pulskvot som valdes var C0, vilket blir 75%.



Figur 4.4 Resultat av ljusstester i mörkt rum

4.7 Batteritest

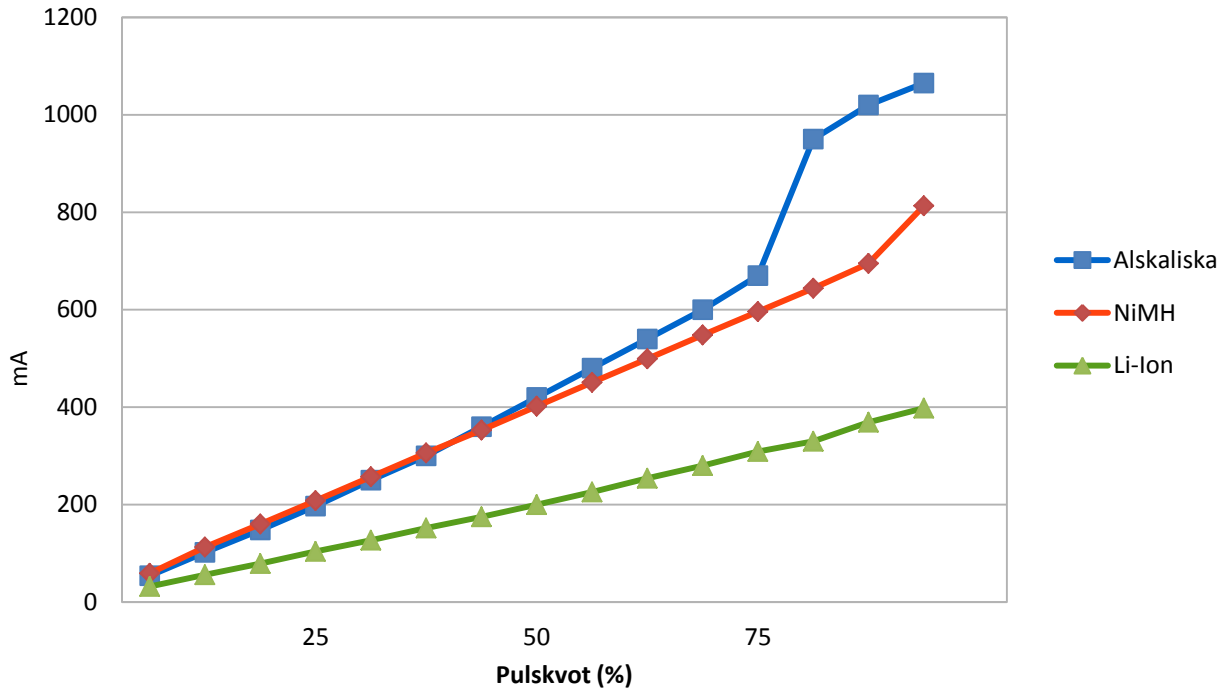
Då de olika batterierna skiljer sig i urladdningskarakteristik gjordes kontrollerande urladdningar av de olika batterierna för att få fram deras individuella spänningskurvor. Först gjordes ett test med batteriladdaren ALC 8500 med en last på 1A för att få en uppskattning för hur batterierna laddas ur. Senare gjordes tester för respektive batteri kopplat till själva lampan med en pulskvot på 81,6% vilket motsvarar D0 i hexadecimal. Detta gjordes dels för att se hur batterierna laddas ur under verkliga förhållanden och dels för att se hur lång drifttid som kan förväntas från varje batteri. De drifttider som framkom när batterierna var kopplade till hårdvaran var:

För att driva lampan användes 6 alkaliska AA-batterier, 6 NiMh AA-batterier och 4 Li-Ion celler.

Drifttid för respektive batteri:
Alkaliska: 1h 55 min
NiMH: 2h
Li-Ion: 6h 20min

Grafen i *Figur 2.1* ger oss bra uppskattning om ungefär vid vilken spänning batterierna kommer sluta att fungera. Då kan en spänning väljas så att användaren varnas när spänningen är ungefär 20 min från att ta slut.

Eftersom de alkaliska batterierna och NiMH batterierna är relativs lika i spänningsnivåer så testades först att bryta båda batterierna vid ungefär 6V. Detta ger en bra säkerhetsmarginal till NiMH batterierna då det kan vara skadligt för deras inre kemi att gå lägre. Problemet med detta är att man tappar en stor del av kapaciteten från de alkaliska batterierna vilket gör att detta inte är en användbar metod. Detaljer som skiljer de alkaliska och NiMH batterierna åt är deras startspänning samt lutningen på deras urladdningskurvor. Alternativet att mäta deras startspänning för att skilja dem åt valdes.



Figur 4.5 Strömförbrukning av olika batterier i hårdvaran

Li-Ion
Väljs om spänningen ligger över 12 volt vid uppstart
Blinkar vid 20-30 min kvar vid 13.9 V
NiMH
Väljs om spänningen ligger mellan 7.5V och 7V vid uppstart
Blinkar vid 20-30 min kvar vid 6.4V
Alkaliska
Väljs om spänningen ligger mellan 9.5V och 7.8V vid uppstart
Blinkar vid 20-30 min kvar vid 4V

Hoppet i strömförbrukning vid långa arbetscykler för alkaliska batterierkan bero på att verkningsgraden för den LED driver som användes (LT3517) minskar vid lägre spänningar.

Andra metoder som kunde användas var t.ex. att mäta hur lång tid lampan är på och jämföra den tiden med en framräknad tid. Detta användes inte pga. av att fel kan uppstå om t.ex. batteriet kopplas ur och processorns program startar om, vilket skulle leda till att räknaren börjar om från sitt ursprungsläge.

Batteriets spänningsnivå behöver övervakas så att användaren kan bli varnad i god tid innan lampan slocknar så att batterierna kan laddas eller ersättas med nya. Detta görs genom att mikroprocessorn kontrollerar spänningen i batteriet med jämna mellanrum. Spänningen i de litium-jon batterier som används i detta projekt sjunker linjärt till en början för att sedan falla drastiskt på slutet. Det är en stund innan detta plötsliga fall som användaren bör varnas så inte lampan inte slocknar utan förvarning under ett pågående dyk.
Detta gjordes genom att först analysera batterierna genom att ladda ur dem och se hur laddning över tid förändrades.

4.8 Batterimätning

Efter det behövdes en metod så processorn kunde övervaka batteriets spänning i realtid. Funktionen för att konvertera batteriets spänning till processorn fanns färdig sedan tidigare då den tagits fram för AD-omvandling.
Funktionen battery_level() är den som tar reda på det aktuella spänningsvärdet på port RA2. Den jämför sedan det värdet med de inprogrammerade varningsnivåerna som togs fram i Batteritest kapitlet.
Spänningen kommer in i port RA2 via en spänningsdelning mellan en 5k ohm och en 20k ohm resistor. Detta ger

$$Uadc = \left(\frac{5}{25}\right) * U$$

4.9 Batteritypsalgoritm

När batterierna kopplas in i lampan för första gången kommer en funktion mäta deras polspänning och därifrån avgöra vilken av de 3 olika typerna av batteri som används, Alkaliska, NiMh eller Li-Ion.

4.10 Extrafunktioner

En extrafunktion som inte hann implementeras men som skrevs klart är en funktion för att ge användaren möjlighet att själv kunna ställa in ljusstyrkan. Genom att strömbrytaren förs till av-läget börjar lampan dimma ned. Om användaren under denna period igen för strömbrytaren till på-läget genereras ett avbrott som stannar dimmningen och lampan bibehåller den ljusstyrka den hade när avbrottet skedde. Lampan kommer stanna kvar på denna ljusnivå tills dess att switchen slås av och lampan dimmas ned ytterligare till en lägre nivå eller stängs av helt.
Om denna funktion kommer finnas med i den färdiga produkten är inte säkert i.o.m. att det är ganska vanligt för dykare att kommunicera med varandra genom att blinka lampor, något som blir besvärligt om lampan är långsam på att stängas av och på.

5 Slutsats

5.1 Mjukvaran

Resultatet av detta arbete blev en fungerande kod för att kunna styra lysdiodernas ljusstyrka genom att generera en pulsbreddsmodulerad signal från en mikroprocessor till en styrkrets som i sin tur skickar signalen vidare till dioderna. Processorn kan även mäta spänningsnivån på batteriet genom att konvertera spänningens analoga värde till ett 8-bitars digital värde. För att användaren ska kunna styra lampan finns ett vred med en magnet som påverkar en hallsensor.

5.2 Ljusstyrkan

Att mäta ljusstyrkan exakt visade sig svårt eftersom nödvändig utrustning för ett sådant test saknades. En uppskattning kunde dock göras med hjälp av en luxmätare i ett mörkt rum där mätare och lampa lades på golvet och ljuset som reflekterades från tak och väggar mättes. Dessa värden bör inte jämföras med värden från andra lampor utan bara användas som ett inbördes mått på de olika ljusstyrkorna.

En funktion för att dimma lampan med hjälp strömbrytaren skapades, men eftersom det kan vara viktigt för dykare att kommunicera genom att blinka sina lampor lyckades vi inte skapa något bra sätt att implementera denna funktion med befintlig hårdvara.

För att ha en så låg strömförbrukning som möjligt men ändå få en bra ljusstyrka valdes en pulskvot på 75%, detta för att undvika den plötsliga ökning i strömförbrukning som sker med högre pulskvoter när lampan drivs av alkaliska batterier.

5.3 Framtida Arbete

Ett framtida arbete kan vara att implementera så att användaren kan välja mellan dimning av lampan eller vanligt av/på genom någon input från strömbrytaren. Processorn som användes hade inte mycket utrymme kvar så det fanns inte mycket utrymme för fler mjukvarufunktioner. Ett förslag är att byta till en större processor, eftersom då finns det plats för fler funktioner. Kretskortet har även utrymme för fler komponenter, tex en accelerometer kan implementeras för ett simpelt användargränssnitt.

6 Referenser och Källor

[1] http://www.mpoweruk.com/soc.htm (Acc 2014-01-20)

[2] http://batteryuniversity.com/learn/article/is_lithium_ion_the_ideal_battery(Acc 2014-01-20)

[3] http://batteryuniversity.com/learn/article/Nickel_based_batteries (Acc 2014-01-21)

[4] http://www.digikey.com/us/en/techzone/lighting/resources/articles/how-to-dim-an-led.html (Acc 2014-01-21)

[5] http://ww1.microchip.com/downloads/en/DeviceDoc/41585A.pdf (Acc 2013-11-20)

[6] http://d32zx1or0t1x0y.cloudfront.net/2011/06/atmega168a_pwm_02_lrg.jpg (Acc 2014-01-13)

6.1 Ekvationer

Ekv 1 $PWM - Period = (PR2 + 1) * 4 * Tosc * (TMR2\ Prescale\ value)$ ekv 1

Ekv 2 $Pulsbredd = (PWM1DCH{:}PWM1DCL < 7{:}6 >) * Tosc * (TMR2\ Prescale\ value)$

Ekv 3 $Arbetscyckel = \frac{(PWM1DCH:PWM1DCL<7:6>)}{4*(PR2+1)}$

PR2 = register som Timer2 räknar upp till

Tosc = 1/Fosc

Fosc = klockoscillatorns frekvens

PWMxDCH = PWM-pulskvot

Bilaga 1 – Main programmet

```
/*
 * File:   main.c
 * Author: Michael Gustafsson & Mikael Isaksson
 *
 * Skapad den 11 November 2013
 */

#include <stdio.h>
#include <stdlib.h>
#include "GenericTypedefs.h"
#include <pic10f322.h>
#include <pic.h>
#include <xc.h>

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

// CONFIG
#pragma config FOSC = INTOSC    // Oscillator Selection bits
#pragma config BOREN = OFF      // Brown-out Reset Enable (Brown-out Reset disabled)
#pragma config WDTE = OFF       // Watchdog Timer Enable (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = OFF   // MCLR Pin Function Select bit
#pragma config CP = OFF         // Code Protection bit (memory code protection is disabled)
#pragma config LVP = OFF        // Low-Voltage Programming Enable (High-voltage on
MCLR/VPP must be used for programming)
#pragma config LPBOR = ON       // Brown-out Reset Selection bits (BOR enabled)
#pragma config BORV = LO        // Brown-out Reset Voltage Selection (Brown-out Reset
Voltage (Vbor), low trip point selected.)
#pragma config WRT = OFF        // Flash Memory Self-Write Protection (Write protection off)

#define  _XTAL_FREQ 1000000     //nödvändig för delay, värdet är klockfrekvensen

#define POWER_ON RA1            //sätter på dc/dc, RA1 på riktig hårdvara
#define SWITCH RA3          //knapp till input, RA3 på riktig hårdvara

char TYP = 0;

//Initierar processorns PWM funktion
void initPWM(void);

//Dimmar på lampan
void wake_up(void);

//Initierar AD konvertering
void initAD(void);
```

```
//Initierar interrupts
void init_interrupt(void);

//Dimmar av lampan och går in i sleep
void shutdown(void);

//Testar batterinivå, och vidtar nödvändiga åtgärder
void battery_level(void);

//Innehåller inställningar
void init(void);

//Kollar batterityp
void battery_type(void);

/*****************************************************************************/
int main(void)
{
	init();
	int counter = 0;
	battery_type();	//räknar ut vilken typ av batteri som används
	if(SWITCH == 0)		//dimmar på lampan om switch är 0
		{
		wake_up();
		battery_level();	//mäter polspänningen över batterierna
		}

	if(SWITCH == 1)
		{
		shutdown();
		}
	while(1)
		{
		if(TMR0IF == 1)	//kollar batteriets spänning varje minut
			{
			counter++;
			if(counter == 225)		//225 ca 60s
				{
				battery_level();
				counter = 0;
				}
			TMR0IF = 0;
				}

			}

			return EXIT_SUCCESS;
}
```

```
/*************************************************************************/
//Vaknar MCUn från sleep samt dimmar på lampan
void wake_up(void)
{
	__delay_ms(50);		//button debouncer
	if(SWITCH == 0)
		{
		if(PWM1DCH < 0xC0)	//så att programmet inte går in här
			{			//om inte lampan ska tändas
			WPUA = 0b00001111;
			POWER_ON = 1;
			char i = 0;
			while(i < 0xC0)   //dimmar på lampan
				{
				i++;
				PWM1DCH = i;
				__delay_ms(1);
				}
			}
		}
}

/*************************************************************************/
//Stänger sakta av lampan och går in i sleep
void shutdown(void)
{
	__delay_ms(50);			//debouncer
	if(SWITCH == 1)
		{
		char i = PWM1DCH;
		while(i > 0 )		//dimmar lampan, när den blir noll ska det brytas
			{
			i--;
			PWM1DCH = i;
			__delay_ms(1);
			}

			WPUA = 0x00;
			POWER_ON = 0;
			SLEEP();
			NOP();		//sleep kommer utföra denna instruktion om GIE = 1
					//sen hoppa till interrupt vid uppvaknande
		}
}
```

```
/*********************************************************************/
//Allmäna inställningar
void init()
{
	ANSELA = 0x00;		//1 så är analog I/O enable /0 så är digital I/O enabled
	LATA = 0x00;
	TRISA = 0x04;		//sätt tris-registerna till I/O (1 = input/0 = output)
	WPUA = 0b00001111;
	OPTION_REG = 0b01010111;	//Aktivera Weak Pull-up samt timer0
	VREGPM1 = 1;		//low power mode i sleep
	initPWM();
	OSCCON = 0b00111001;		//ger en klockfrekvens på 1 MHz bit 6-4
	POWER_ON = 1;		//sätter på dc/dc


	init_interrupt();
}

/*********************************************************************/
//Initierar PWM
void initPWM(void)	//Initiera PWM
{

 TRISA  = 0x01;		// Stäng av output på RA0
 PWM1CON = 0x00;	// Nollställ PWM1Con
 PR2   = 0xFF;		// Ställ in period på Timer2
 PWM1DCH = 0x00;
 PWM1DCL = 0x00;
 TMR2 = 0x00;		//nollställer timer2
 TMR2IF = 0;		//Cleara tmr2if flaggan
 TMR2ON = 1;		//startar timer2
 TRISA = 0x04;		//Slå på pwm pin output
 PWM1EN = 1;		//Aktivera PWM-modulen
 PWM1OE = 1;		//Aktivera output på PWM-modulen
 T2CON = 0b00000101;	//bit0-1 sätter prescaler till 4

}

/*********************************************************************/
//Initiering av interrupts
void init_interrupt(void)
{
	IOCIE = 1;		//enable interrupt on change
	IOCAN3 = 1;		//falling edge interrupt
	IOCAP3 = 1;
	GIE = 1;		//Enable global interrupt
}
```

```
/*********************************************************************/
//Initierar AD konverteringen samt utför den
//och lägger ut värdet på ADRES
void initAD(void)
{
ADIF = 0;
                    //Konfigurera portar
TRISA2 = 1;       //Sätt RA2 input
ANSA2 = 1;        //Sätt RA2 till analog
ADCON = 0b00001001;       //Konfigurera ADC-modul
                          //Välj ADC konverteringsklocka, Fosc/2
                          //Välj inputkanal för ADC
                          //Slå på ADC-modul
__delay_us(2);            //Vänta nödvändig tid
GO_nDONE = 1;             //Starta konvertering genom att sätta GO/DONE bit.
                          //Vänta på att ADC konvertering är färdig genom att
                          // undersöka GO/DONE-bit
while(1)
        {
         if(ADIF == 1) //Undersöker om AD omvandlarens Interruptflagga är satt (bit 6)
        {
        ADON = 0;
        break;
        }
        }
   ADIF = 0;              //8.Nollställ ADC interruptflagga (om interrupts är påslagna)

}
```

```
/*********************************************************************/
//Kollar batterinivån varje minut och blinkar lampan om spänningen är för låg
void battery_level(void)
{
	int variabel = 0;
	initAD();
	if(TYP == 1)			//1 = Li-ion batteri
		{
		if ( ADRES < 217 )	//217 mostsvarar ungefär 13.9 V
			{		//vilket ger 20-30 min tid kvar
			variabel = 1;
			}
		else
			variabel = 0;
		}
	if(TYP == 2)			//2 = NiMH batteri
		{
		if (ADRES < 110 )	//sätta denna så att den mostsvarar 6.4V
			{			//ungefär 20 min innan batteriet ger upp
			variabel = 1;
			}
		else
			variabel = 0;
		}
	if(TYP == 3)		//3 = Alkaliska icke laddningsbara batteri
		{
		if (ADRES < 95 )   //börja varna vid 4 volt, cirka 20 min kvar
			{
			variabel = 1;
			}
		else
			variabel = 0;
		}
//Om spänningen på batteriet faller under det specifika batteriets
//lägstanivå, blinka lampan
	if (variabel == 1 )
		{
		variabel = 0;

		PWM1DCH = 0x00;
		__delay_ms(500);
		PWM1DCH = 0xC0;
		}
}
```

```
/*************************************************************************/
//Läser av vilken typ av batteri som används och lägger in i variabeln TYP
void battery_type(void)
{
	initAD();

	if(ADRES > 190)					//190 är 12.0 Volt in
		{					//222 är 14.0 Volt in
		TYP = 1;			//1 = Li-ion batteri
		}
	if(ADRES < 136 && ADRES > 130)	//mindre än 7.5V och större än 7V
		{
		TYP = 2;			//2 = NiMH batteri
		}
	if(ADRES < 166 && ADRES > 140)	//mindre än 9.5V men större än 7.8V
		{
		TYP = 3;			//3 = Alkaliska icke laddningsbara batteri
		}

}

/*************************************************************************/
//Interruptrutin
void interrupt ISR()
{
	if(IOCAF3 == 1) //kolla om interrupt-on-change har triggat
		{
		IOCAF3 = 0;		//nollar interruptflaggan
		GIE = 1;		//sätterpå globala interrupts
		if(SWITCH == 0)		//är switchen i på läge?
			{
			wake_up();

			}
		if(SWITCH == 1)		//är switchen i av läge?
			{
			shutdown();
			}

		}
}
```

Bilaga 2 - EL-schema