Internetuppkopplat reglersystem

Styrning av temperatur i en slowcooker

Christoffer Karlsson

ISYS16 – Projekt inom inbyggda system, 40yhp

2017-06-13, VT2017

Handledare: Staffan Johansson

Förord

Den här rapporten ingår som ett moment i kursen Projekt inom inbyggda system, 40yhp. Kursen är en del av den 1-åriga YH-utbildningen Utvecklare inom inbyggda system, som drivs av EC Utbildning. Jag är tacksam över att ha fått den magnifika möjligheten att utföra det här projektet på plats hos Neava AB i Luleå. Ett stort tack förstås till min handledare tillika VD på Neava, Staffan Johansson. Vill även passa på att tacka farsgubben, Ove Karlsson. Han hjälpte till med inbyggnaden och bidrog till att ge projektet en polerad finish.

Innehållsförteckning

1 Sammanfattning

2 Inledning

2.1 Bakgrund

En Crock-pot (eller långkokare/slowcooker) är en gryta som är designad för att sjuda maten under lång tid. Den typiska slowcookern består av ett keramiskt kokkärl som omges av ett stålhölje med ett inbyggt värmelement. Exakt funktionalitet varierar beroende på modell, men ofta så ges möjligheten att välja några fasta effektlägen, och någon form av timer.

2.2 Uppgift

En Crock-pot ska med hjälp av ett inbyggt system utrustas med temperaturreglering, med målet att temperaturen ska hållas så jämn som möjligt. Det ska vara möjligt att observera och styra temperaturen lokalt, och systemet ska även vara uppkopplat med Bluetooth och WiFi för fjärrstyrning. Som ett följdsteg, när detta är implementerat, så ska någon form av autentiseringssystem läggas till. Med all funktionalitet avklarad så är det sen aktuellt att titta på hur möjligheterna att paketera och miniatyrisera systemet, givet att tid finns över.

Projektet innefattar alla steg i utvecklingsprocessen. Det innebär planering, specifiering, design, utförande och dokumentation.

Problemformuleringen ger upphov till en del frågor som är viktiga att besvara för att kunna slutföra projektet. Vilken utrustning behövs? Hur ska temperaturregleringen hanteras? Hur ska användaren kunna interagera med systemet? Hur ska systemet kommunicera med Bluetooth/WiFi?

2.3 Utvecklingsmiljö

Utvecklingsmiljöerna som har använts är Arduino IDE och Geany. Arduino IDE är en lättviktsmiljö för C++, som är speciellt anpassad för att jobba med Arduino. Varje program i Arduino IDE består av en sketchfil i .ino-format. En sketchfil fungerar som en vanlig källfil till C/C++, med vissa modifikationer. Viktigast är att istället för en main-funktion så ska en sketch innehålla två andra funktioner: void setup(void), och void loop(void). Funktionen setup anropas när mikrokontrollern startas upp, och loop innehåller huvudprogrammet.

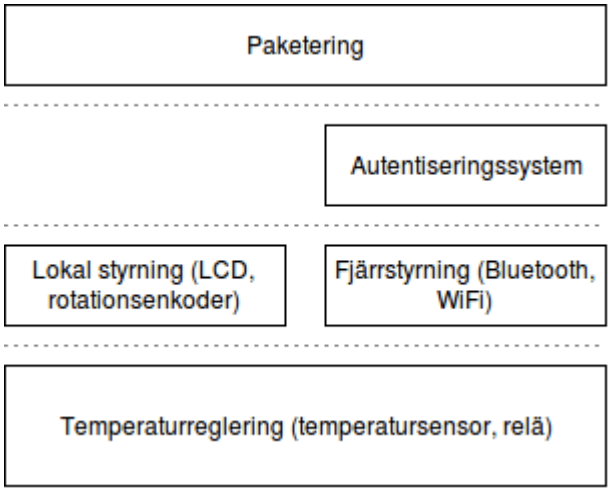
Arduino IDE underlättar processen att kompilera och ladda upp ett program, men som utvecklingsmiljö så är den ganska begränsad. Det saknas autocorrect, möjlighet att formatera texten osv. Jag fann det därför vara en bra idé att använda en extern utvecklingsmiljö, och enbart använda Arduino IDE för att ladda upp koden. Som extern utvecklingsmiljö har jag använt Geany. Det är en minimalistisk IDE med stöd för ett flertal programmeringsspråk, däribland C/C++.

3 Utförande

3.1 Metod

Min arbetsmetod har varit botten-upp och skett komponent för komponent. Varje komponent är möjlig att testa självständigt, och kan sen sammanfogas som ett byggnadsblock i programmet. Nästa steg blir att inrikta sig på hur de här byggnadsblocken ska interagera med varandra. Med t.ex. temperatursensorn och LCD-displayen integrerade i projektet så finns möjligeten att visa temperaturen på displayen. Allt eftersom så klarar programmet av att utföra komplexare uppgifter, tills önskad funktionalitet är uppnådd.

De olika momenten som ingår i projektet kan illustreras med en hierarkisk trappstegsmodell, enligt Figur 1. För att kunna komma till de övre stegen så måste de lägre implementeras först. Genom att ha modellen i åtanke så vet man i varje steg av utvecklingen vilket område som bör vara fokuset för tillfället. T.ex. så behövs ju först fungerande temperaturreglering innan det finns möjlighet att styra den.



Figur 1: Trappstegsmodell

3.2 Material

För utvecklingssyftet så bedömde jag att en Arduino Uno standardmodell var mer än tillräcklig. Den är lätt att jobba med, jag beräknade att den har nog med GPIO-pinnar för att alla komponenterna ska rymmas, och man har tillräckligt stort svängrum i prestandaväg för att man ska slippa oroa sig för att minnet tar slut eller att programmet ska bli för långsamt.
Ett alternativ till en vanlig Arduino Uno hade varit en Arduino Uno WiFi, som inkluderar en inbyggd ESP8266 för WiFi. Jag valde istället att lägga till en ESP-01 modul, som är en breakout board till ESP8266. Det är en mer modulär lösning som inte är lika hårt knuten till Arduino Unon.

Temperaturen i en slowcooker kan överstiga 90°C vid högsta effektläget. Det ställer krav på en temperatursensor som kan hantera ett stort temperaturspann. Förutom det så är det absolut

nödvändigt att sensorn även är vattentät. Valet föll på en DS18B20 temperatursensor från Luxorparts som uppfyller alla dessa egenskaperna.

Jag hade inga precisa planer på exakt hur det lokala användargränssnittet skulle fungera. Därför fokuserade jag på komponenter som tillåter stor flexibilitet, för att sen konstruera en design på dem. En rotationsenkoder är lämplig för att ställa in t.ex. ett börvärde på temperaturen, då den ger möjlighet att snabbt välja mellan ett stort urval av möjliga värden. Enkodern inkluderar även en tryckknapp, vilket kan användas för att t.ex. acceptera ett alternativ eller liknande.
En 16x2 LCD-skärm erbjuder en lagomt stor display, och med mjukvara ges full kontroll över vad och när saker ska visas upp på skärmen.

Eftersom slowcookern drivs med en nätspänning på 220V så måste den vara elektriskt isolerad från resten av systemet. Den vanligaste lösningen här är att driva den med ett relä. Kjell & CO hade ett rätt reguljärt elektromagnetiskt relä som man valde att införskaffa.

Sammanfattat så köptes följande komponenter, med allt inhandlat direkt från Kjell & CO:

- Arduino Uno Rev. 3.
- HC-06 Bluetooth-modul.
- ESP-01 WiFi-modul.
- Luxorparts LCD-display 2x16 tecken.
- Keyes rotationsenkoder KY-040.
- Keyes relämodul KY-019.
- Luxorparts temperatursensor (DS18B20)
- Breadboard
- Kopplingssladdar 40-pack

Förutom detta så fanns även tillgång till resistorer, kondensatorer, och andra mindre komponenter med olika värden.

Vid ett senare tillfället inhandlades en Crock-pot från Elgiganten. Här behövdes inte mer än en modell med väldigt grundläggande funktioner, eftersom själva grytan och värmeelementet är det enda som är av intresse. Valet blev en Crock-pot SCV400. Den är rätt barskalad, och i funktionsväg så erbjuder den enbart tre aktiva effektlägen (High, Low, Warm).
Den simpla konstruktionen gör att man kan använda sig av Crock-poten som den är, utan att behöva utföra några invasiva ingrepp på den. Istället för att integrera regleringssystemet direkt med Crock-poten, så kan det göras som en extern modul. Man kopplar bara in strömsladden från Crock-poten och styr om den ska få ström eller inte. Man är då inte heller låst till att använda sig av just Crock-poten. Så länge man har en apparat med någon form av värmelement, som styrs via ett vanligt nättutag, så skulle vi kunna få en fungerade reglering.

3.3 Temperaturreglering

3.3.1 On/off-reglering

Den första modellen som användes för temperaturreglering var on/off-reglering, som är snabb och enkel att implementera. Crock-poten har ett flertal effektlägen, men antas alltid lämnas i högsta effektläget (om den är påslagen), för att man ska kunna nå högsta möjliga temperatur om det behövs. Reläet styr värmelementet, som befinner sig i antingen fullt påslaget eller avslaget läge. Vi har ett börvärde som temperaturen i processen ska hålla sig till, och ett litet intervall kring börvärdet med hysteres. Skulle temperaturen sjunka under intervallet så slås reläet på, och stiger temperaturen över intervallet så slås reläet av. Någon form av intervall är önskvärt för att reläet inte ska slå om för frekvent, och leda till att det slits ut i förtid.

Diagram 1 Visar hur processen beter sig med regleringen påslagen. Vi har ett börvärde på 60°C, och ett intervall på $\pm 0{,}3\,°C$ som temperaturen ska hålla sig inom. Kurvan visar tydliga periodiska svängningar, där främst den maximala temperaturen (62°C) ligger långt över acceptabla värden. Det beror på att keramikkärlet har hög förmåga att magasinera värme, vilket ger hög termisk tröghet i processen. Det förvärras även att att det saknas möjlighet att kyla ner vattnet; det finns en gas, men ingen broms. Vi måste vänta på att processen kyler ner sig själv innan elementet slås på igen. Hur som helst så är avvikelsen från börvärdet tillräckligt stort för att on/off-reglering inte ska vara acceptabelt, utan det behövs en bättre lösning.

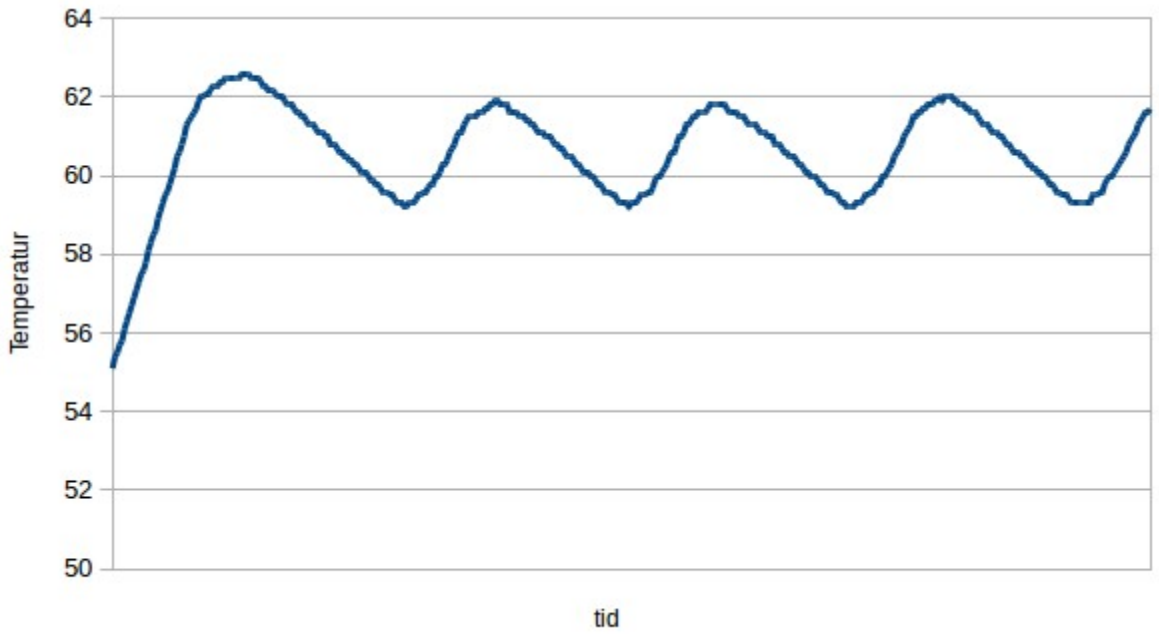


Diagram 1: On/off-reglering, 60°C

3.3.2 Proportionell reglering

Största problemet med on/off-reglering är att det saknas möjlighet att styra graden av uppvärmning. Man kan göra en analogi med att köra bil; Föreställ att man har en hastighetsgräns på 100km/h och att man stampar plattan i mattan tills man nått den hastigheten. Efter det släpper man helt gasen, och gasar på igen först när man kommit en bit under 100km/h. Som man kan föreställa sig så skulle det

bli en väldigt ryckig biltur. För att fortsätta på analogin så skulle man kunna göra hastighetskurvan mycket slätare genom att iaktta självbehärskning och styra exakt *hur* mycket man trycker ner gasen. Ju närmare man kommer 100km/h, desto mer släpper man på den. Vad man får då är proportionell reglering.
Återkopplat till det nuvarande problemet så skulle man vilja skicka PWM-signaler till värmelementet, och styra storleken på duty cyclen beroende på hur nära börvärdet man är. För ett visst börvärde $r(t)$ som processen ska hålla, och ett ärvärde $y(t)$, så ges det uppmätta felet $e(t)$ av

$$e(t)=r(t)-y(t) \tag{1}$$

Utifrån felet så justeras då längden $p(t)$ på duty cyclen till

$$p(t)=\begin{cases} 0, & u(t)<0 \\ 1, & u(t)>1 \\ u(t), & annars \end{cases} \tag{2}$$

Där

$$u(t)=K_p e(t) \tag{3}$$

Storheten K_p är den proportionella förstärkningen, en parameter som anger hur starkt systemet ska reagera på avvikelser i temperatur. Det optimala värdet på K_p beror på egenskaperna hos processen, och behöver justeras efter behov.

Att åstadkomma något som liknar en PWM-signal med ett elektromagnetiskt relä är inte möjligt. Reläet skulle behöva switcha med alldeles för hög frekvens, och väldigt snabbt bli utbränt. Här är ett halvledarrelä/Solid State-relä (SSR) en bättre lösning. Omkopplingen i ett SSR sker med en optokopplare, och slits inte ut vid användning.

Egenskaperna hos ett SSR begränsar dock fortfarande vilka frekvenser som är rimliga på PWM-signalen. Reläet kopplar om vid nollgenomgång på växelström, vilket ger en maximal omkopplingstid på ½ cykel. Med en växelström på 50Hz så blir då den maximala omkopplingstiden 10ms. Det är alltså orimligt att ha en period som är kortare än 10ms. Lämpligen så bör signalens period vara en multipel av 10ms. Genom att dela in signalen i intervall på 10ms så får vi ingen fasförskjutning när vi kopplar om, och kan vara säkra på att omkopplingstiden inte varierar.

Jag bedömde att en period på 100ms (10Hz) var en bra kompromiss. En längre period ger ett större omfång av möjliga värden på duty cyclen, men en för lång period gör att uppvärmningen blir för ojämn. 10Hz är fortfaranade en väldigt låg frekvens för en PWM-signal, men det är acceptabelt i det här fallet. Hade vi istället haft en motor som styrdes med så låg frekvens skulle man få väldigt ryckiga rörelser och stora energiförluster då motorn frekvent startar och stannar upp igen. Stenkärlet

har däremot som tidigare nämnt stor termisk tröghet; skulle vi stänga av värmelementet i någon tiondels sekund så skulle inte det märkas i processen.

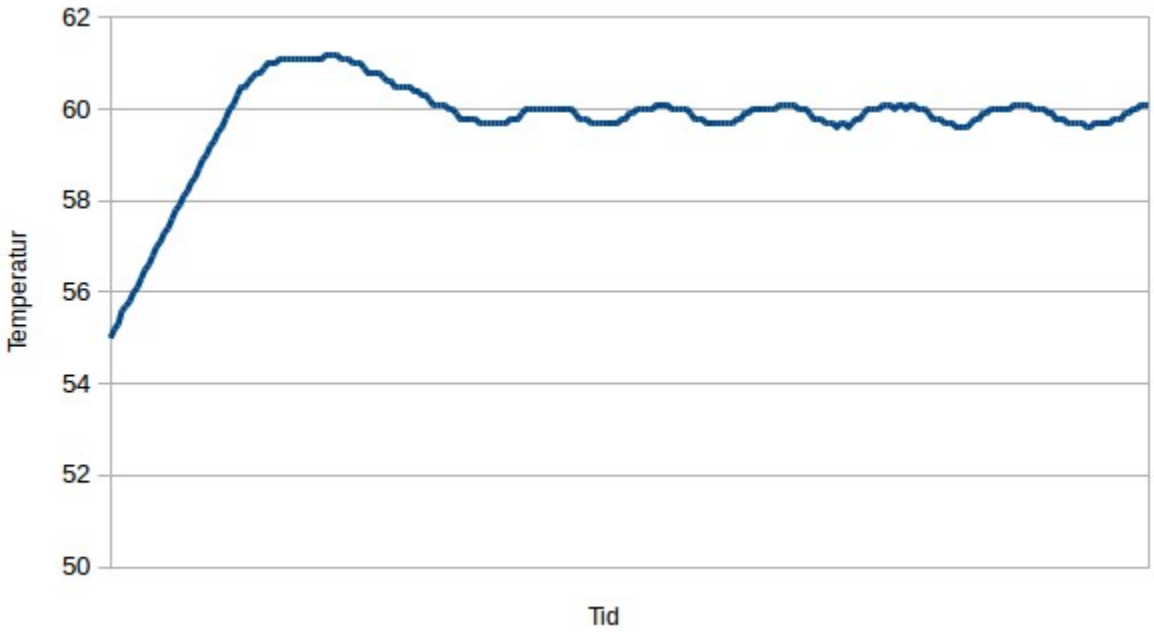


Diagram 2: Proportionell reglering, 60°C, $K_p = 0.15$

Med proportionell reglering så fås som förväntat mer stabila resultat, vilket syns i Diagram 2. Den proportionella förstärkningen här är lite låg, vilket medför att den genomsnittliga temperaturen ligger en hårsmån under 60°C. Det är fortfarande väldigt nära det önskade värdet, och fullt acceptabelt.

3.3.3 Temperatursensor

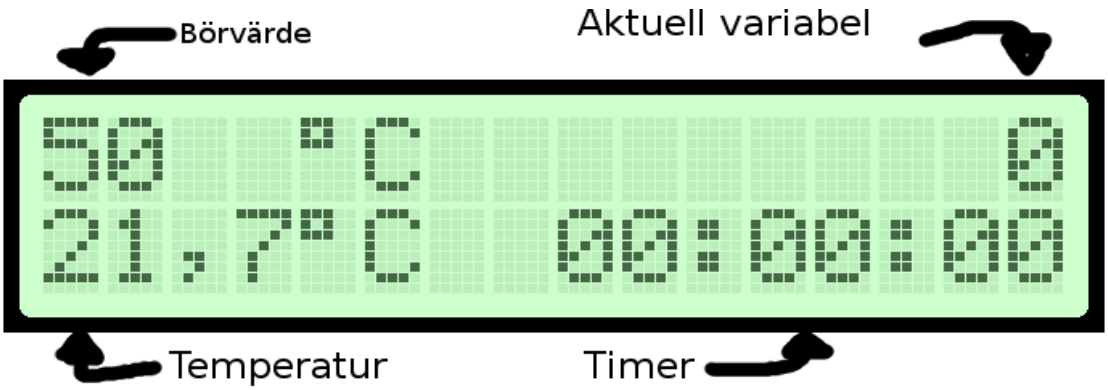
Till skillnad från många andra temperatursensorer så kommunicerar DS18B20 med 1-wire, istället för att ge en analog signalspänning. Jag använde mig av Arduinos 1-wire-bibliotek för lager 1-kommunikationen, och ett externt bibliotek (Arduino Library for Maxim Temperature Integrated Circuits) som sköter lager 2 och skickar instruktioner till sensorn.

För att kunna få korrekta data så behövs ett PullUp-motstånd på 4,7kΩ mellan dataingången och matningsspänningen. Att erhålla temperaturdatan tar även en ansenlig tid (750ms), vilket är ett problem om programmet ska blockera tills läsningen är klar. Problemet löstes genom att låta programmet skicka en förfrågan till sensorn om att läsa data, och sen fortsätta köra på som vanligt. När en sekund har gått läser programmet den uppmätta temperaturen och uppdaterar.

3.4 Lokal styrning

Det lokala användargränssnittet använder sig av en 16x2 LCD-display och en rotationeenkoder med en tryckknapp. Den första designen som brukades var väldigt ad hoc och mest lämpat för testningssyfte. Figur 2 visar hur den såg ut. Några viktiga storheter visas på skärmen, och uppe i högre hörnet syns en indexvariabel, som indikerar vilket värde på skärmen som användaren kan påverka genom att vrida på rotationsenkodern. Genom att trycka på knappen på enkodern så kan användaren växla mellan variablerna. De variabler som är möjliga att påverka är

0) Börvärdet
1) Antalet timmar
2) Antalet minuter
3) Antalet sekunder
4) Reglering på/av



Figur 2: LCD, första versionen.

Det är förstås ett förvirrande system som förmodligen bryter mot varenda designregel i boken, så det blev tillbaka till ritbordet senare under projektet när tid fanns. Jag ville ha något som är lättare att använda, och som är lätt att utöka om man vill ha mer funktionalitet. Viktigt att ha i åtanke var också att displayen enbart kan visa upp 16X2 tecken. Lösningen blev att implementera ett navigerbart menysystem.

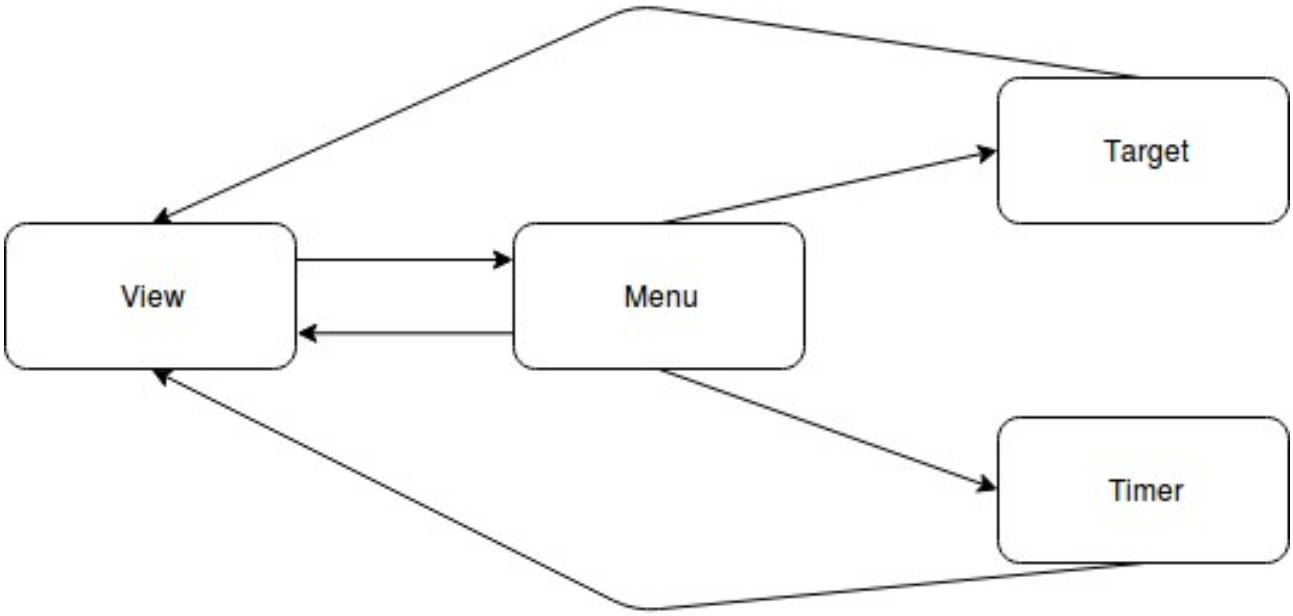
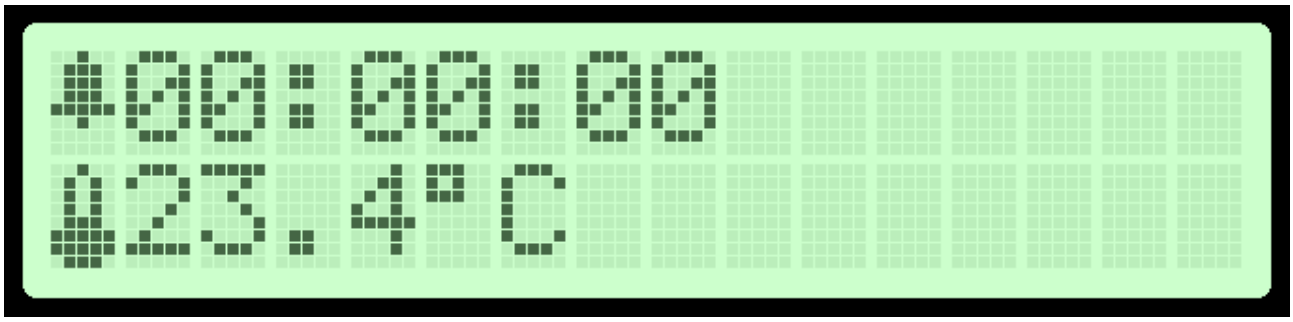


Diagram 3: Menysystemet, tillståndsdiagram.

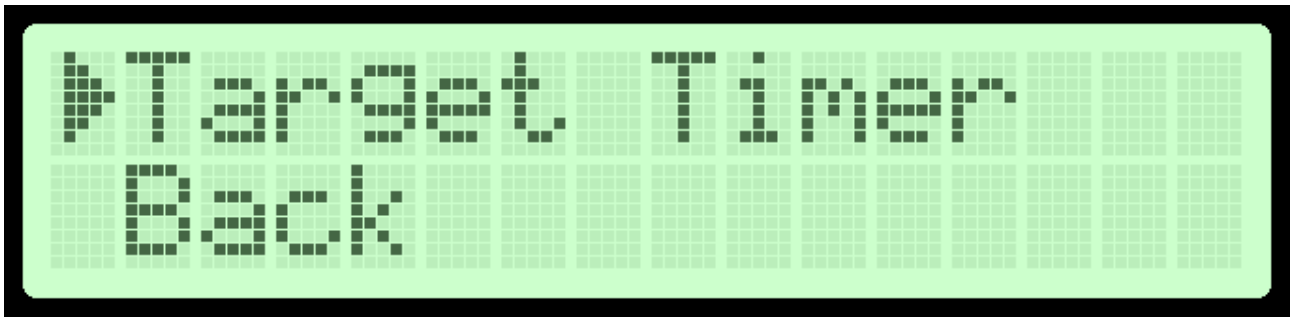


Figur 3: LCD view

Ursprungligen så visar LCD-displayen den aktuella temperaturen respektive timern, som i Figur 3. Displaysystemet fungerar som en state machine, och är illustrerad med möjliga övergångar i Diagram 3. Användaren kan nå nästa tillstånd genom tryckknappen på rotationsenkodern. Varje tillstånd representeras av en instans av structen lcd_state:

```
struct lcd_state
{
     void (*init)(void); //called when state initialized
     void (*mode_func)(enum direction); // called on rotary encoder
     void (*update)(void); // called every update cycle
     struct lcd_state (*state_func)(void); //gives next state
};
```

Menu-tillståndet ger användaren tillgång till en meny, som ser ut som i Figur 4. Genom att vrida på rotationenkodern så kan användaren växla mellan alternativen i menyn, och trycker på knappen för att acceptera valet och gå till nästa tillstånd. Möjlighet finns att ändra börvärde (Target), ställa in timern (Timer), eller att gå tillbaka (Back). Användaren kan därefter ställa in önskat värde för att sen gå tillbaka till View-läget.



Figur 4: LCD menu

3.5 Fjärrstyrning

3.5.1 Koppling och strömaspekter

Både BT- och WiFi-modulen kommunicerar med UART. Det var något jag inte hade ponerat på i förhand, och visade sig bli ett stort problem av ett flertal anledningar. Arduino Uno har enbart en serieport i hårdvaran, och använder man den så tappar man förmågan att utnyttja USB-anslutningen till Arduinon, som använder sig av samma port. Det går då inte att använda seriellmonitorn för debugging, och det är inte heller möjligt att ladda upp program.
Som tur är så har Arduino ett bibliotek för att hantera seriell kommunikation med mjukvara. Det löste tidigare nämnda problem men gav samtidigt upphov till nya. Även om biblioteket officiellt ska stödja baud rates upp till 115200 så såg inte verkligheten ut så. Arduino Uno har en klockfrekvens på 16MHz, vilket visade sig innebära alldeles för tight timing. Det enda man fick tillbaka var skräpdata, även när man hade satt korrekt baud rate. Den skräpdatan skiljer sig inte heller från den man får när man försöker prata med fel baud rate, vilket gjorde felet rätt besvärligt att diagnostisera. I slutändan så åtgärdade jag problemet genom att använda hårdvaruporten och instruera modulen att sänka sin baud rate.
Ett annat problem med biblioteket är att det enbart är kapabelt att lyssna på en mjukvaruport samtidigt. Data som kommer in från den porten man lyssnar på hamnar i en buffer, och eventuell data som kommer in från andra portar kastas bort. Det är inte möjligt att veta i förväg från vilken modul data kommer att anlända från härnäst, så det går inte att garantera att datan inte går förlorad. Enda sättet att åtgärda detta, som jag kunde komma på, var att låta en modul använda hårdvaruporten och låta den andra använda en mjukvaruport. För att fortfarande skriva ut debugginginformation samtidigt som man testade modulerna så skrev jag ut datan på porten där jag använde Bluetooth.

Såväl BT- som WiFi-modulen använder sig av 3.3V-signalering utan att vara 5V-toleranta, och i fallet med WiFi-modulen så kräver den även att matningsspänningen ska vara på 3.3V. Signalspänningen från mikrokontrollerns GPIO-pinnar till modulernas RX-pinnar är hanterbar med spänningsdelare, men matningsspänningen till WiFi-modulen utgör ett stort hinder. Den vill ha relativt stora mängder ström, med toppar på mer än 200mA vid dataöverföring över WiFi. 3.3V-pinen på Arduino Uno har en kapacitet på 100mA. Det är inte tillräckligt, och leder till frekventa omstarter på modulen. I tidigare stadier av projektet så löstes detta genom att låta modulen få

matning från ett externt nätaggregat. För en mer långsiktig och portabel lösning så köptes en LM2596 spänningsregulator från Kjell & CO.

3.5.2 Programtolk

Fjärrstyrning av systemet över Bluetooth och WiFi sker med en textbaserad programtolk, som man kan komma åt med t.ex. en terminalemulator. Ett Command Line Interface (CLI) är kanske bland de enklare och snabbare sätten här att få till ett gränssnitt som en människa kan använda. Programtolken ger tillgång till systemet genom att användaren får åtkomst till viktiga variabler i mikrokontrollern, som t.ex. börvärdet. Grammatiken som implementeras är simpel:

$$S \rightarrow get\ VARIABLE \mid set\ VARIABLE\ VALUE \tag{4}$$

Kommandona get och set ger möjlighet att läsa, respektive skriva, till en variabel. VARIABLE är namnet på den variabel man vill ha åtkomst till, och VALUE är ett giltigt värde, som beror på vilken typ av variabel man vill komma åt.

Varje variabel, så som de är representerade i programtolken, består av en struct:

```
struct variable
{
     void *addr; //address to variable
     char *symbol; //string representation in interpreter
     enum permission perm; //read, write or both
     enum token_id type; //what type the variable should be treated as
     int min; //minimum value, if number
     int max; //maximum value, if number
};
```

Medlemsvariabeln perm består av flaggor som anger vilka rättigheter vi har att läsa och skriva den tillhörande variabeln i programtolken:

```
enum permission
{
     P_R  = 0x01, // read
     P_W  = 0x02, // write
     P_RW = 0x03, // equals P_R & P_W
};
```

I vissa fall är det befogat att enbart göra en variabel läsbar eller skrivbar. T.ex. så är det möjligt att läsa av den aktuella temperaturen, men att ändra temperaturen kan ju enbart göras indirekt genom att ställa in börvärdet. Det är även möjligt att ange lösenordet som ska användas för att ansluta genom WiFi, men för säkerheten är det förstås inte möjligt att visa lösenordet.

3.5.3 Autentisering

Som standard så använder sig Bluetooth-modulen av en PIN-kod, som den kräver innan den parar ihop sig med den andra enheten. Det är en form av autentisering som kan anses tillräcklig för

Bluetooth, då räckvidden för standarden bara ligger på ungefär 100 meter. För WiFi var dock någon form av autentiseringssystem nödvändigt att utveckla. Ett lösenordssystem är en vanlig lösning som ger hyfsad säkerhet. Om användaren ansluter genom nätverket får denne ange ett lösenord. Skulle användaren ange rätt lösenord så ges tillgång till programtolken som vanligt. IP-adressen hos användaren loggas för en annan nivå av autentisering, och användaren räknas som inloggad tills uppkopplingen timar ut (10 minuter).

3.6 Inbyggnad

Med de överordnade målen uppfyllda så fanns fortfarande tid kvar till en inbyggnad av systemet. Förutom att presentera systemet på ett snyggt sätt för användaren så undviker man att exponera sladdar och känsliga komponenter.

Under utvecklingen så har alla komponenter kopplats ihop på en breadboard. Snabbaste sättet att få ihop en inbyggnad är då helt enkelt att stoppa ner breadboarden med komponenter i en låda. Med nöd och näppe så var det möjligt att trycka ner allt i en låda på 160x240x90mm. Lådan har en DC-plugg för en 12V-adapter, som går till strömregulatorn som förser systemet med matning på 5V och 3.3V. Reläkontakterna går till ett vägguttag som är monterat på lådan, där det är möjligt att koppla in Crock-poten.

Det här är ju en fungerande lösning, men det är möjligt att göra mycket mer. Främst alla sladdar är skrymmande. Vanligast, speciellt i kommersiella projekt, är att man konstruerar ett mönsterkort med ledningar där man sedan löder fast komponenterna. Mönsterkorten designas oftast i ett CAD-program, innan mönstret skickas i Gerberformat till en tillverkare.

Vid skrivande stund så är designen till mönsterkortet klart, men inte producerat än.

4 Slutsats

Alla uppgifter i projektet är uppfyllda. Temperaturreglering fungerar som önskat, och systemet kan styras genom ett flertal kanaler. I slutet fanns även tid över att jobba med paketeringen av systemet, vilket låg över förväntan.

4.1 Framtida arbete

Jag har inte gjort några större ansträngningar för att minimera strömförbrukningen. Det är ett område som ofta kan vara viktigt i ett inbyggt system, men här bedömde jag att den är en låg prioritet. Det är möjligt att t.ex. byta ut till strömsnålare kompontenter och att aktivera strömsparläge på mikrokontrollern, men strömförbrukningen domineras till stor del av Crock-poten. De eventuella vinningar man skulle kunna göra är inte försumbara, men inte heller överväldigande.

Kontaktstudsar är till viss del hanterade i mjukvaran. Resultatet är dock inte perfekt. Speciellt enkodern är benägen till kontaktstudsar. Enkodern har två pinnar (D1 och D2) som avgör när och i vilken riktning enkodern roteras. I ursprungsläget är både D1 och D2 låg. När man sen börjar rotera så kommer D1 och D2 gå till högt. Genom att notera vilken pinne som går från låg till hög först så är det möjligt att veta åt vilket håll enkodern roterar. När man slutför rotationen så kommer värdet

på pinnarna gå från hög till låg igen, i samma ordning. Ett exempel på möjligt vågformsdiagram syns i Diagram 4. Det är många moment som ska hanteras på kort tid, och det är lätt att kontaktstudsar ställer till det.

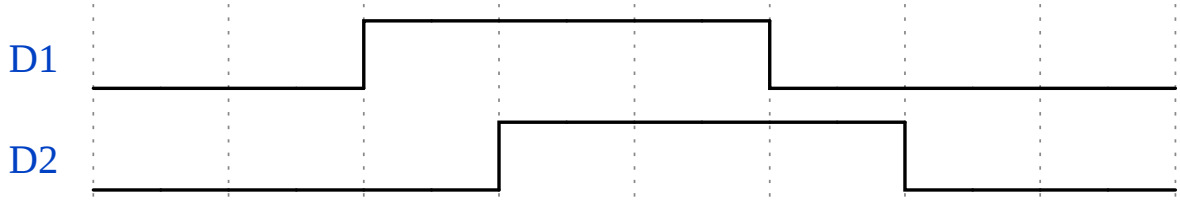


Diagram 4: Vågformsdiagram för medurs rotation

Vill man hantera kontaktstudsarna ordentligt så måste man ta till en hårdvarulösning. Det finns flera sätt att göra detta på, t.ex. med en kondensator och en Schmitt trigger.

Många slowcookers, främst i den övre prisklassen, har möjligheten att köra program. Man skulle t.ex. kunna köra ett program på ”låg” effekt i 3 timmar, gå över till ”hög” effekt i ytterligare 3 timmar och sen stanna i ett varmhållningsläge. Viktiga frågeställningar om man vill gå den vägen är hur man ska integrera ett programläge med resten av systemet, och hur användaren ska kunna välja och eventuellt göra sina egna program.

Vill man utveckla WiFi- och Bluetooth-interfacet vidare så skulle en GUI kunna vara ett alternativ. Det är något som hade ökat användbarheten drastiskt, men som ligger utanför omfattningen på den aktuella utbildningen. För att slippa göra förändringar i koden så är det fullt möjligt att använda sig av den existerande programtolken, och lägga GUIn som ett lager ovanför den. Men programtolken skulle även kunna modifieras, så den jobbar med siffror eller annan data, istället för med strängar. Det finns ingen större anledning att ha ett strängbaserat system när människor inte är involverade, och C är inte ett programmeringsspråk som är optimalt lämpat för att hantera strängar.

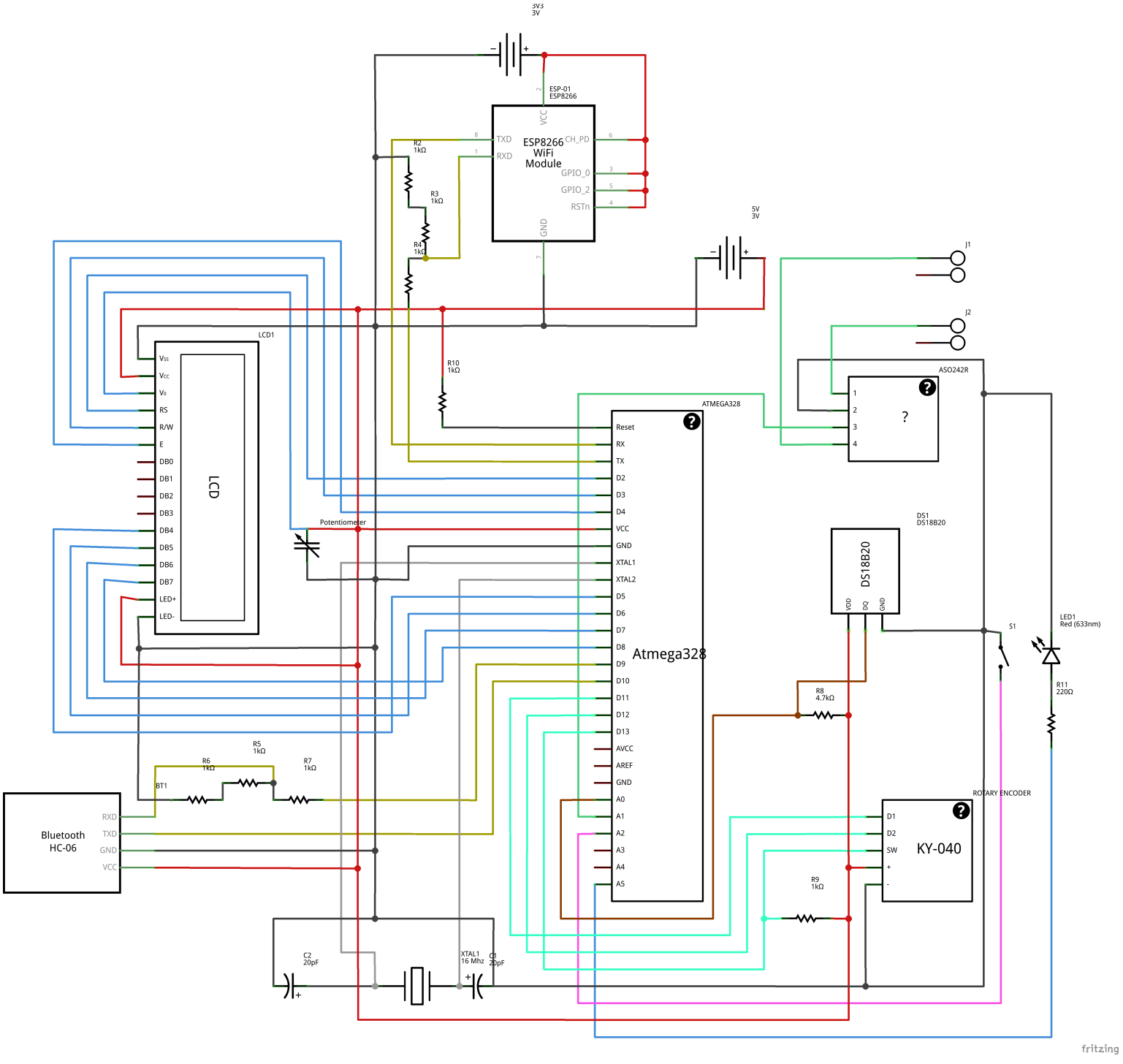
Autentiseringssystemet är rätt primitivt, och för att kunna vara brukbart i ett kommersiellt projekt så måste det förbättras. Det är ett lösenordsbaserat system där lösenordet skickas över i klartext, vilket är en stor sårbarhet då någon skulle kunna sniffa upp trafiken. Den naturliga lösningen här är att kryptera datan innan man skickar över den.

För tillfället så behöver inbyggnadslådan två strömkällor; en på 12V DC respektive en på 220V AC. Idealt hade varit om man hade kunnat driva allt från 220V. För det så skulle man vara tvungen att lägga till en likriktare, och sänka spänningen till acceptabla nivåer.

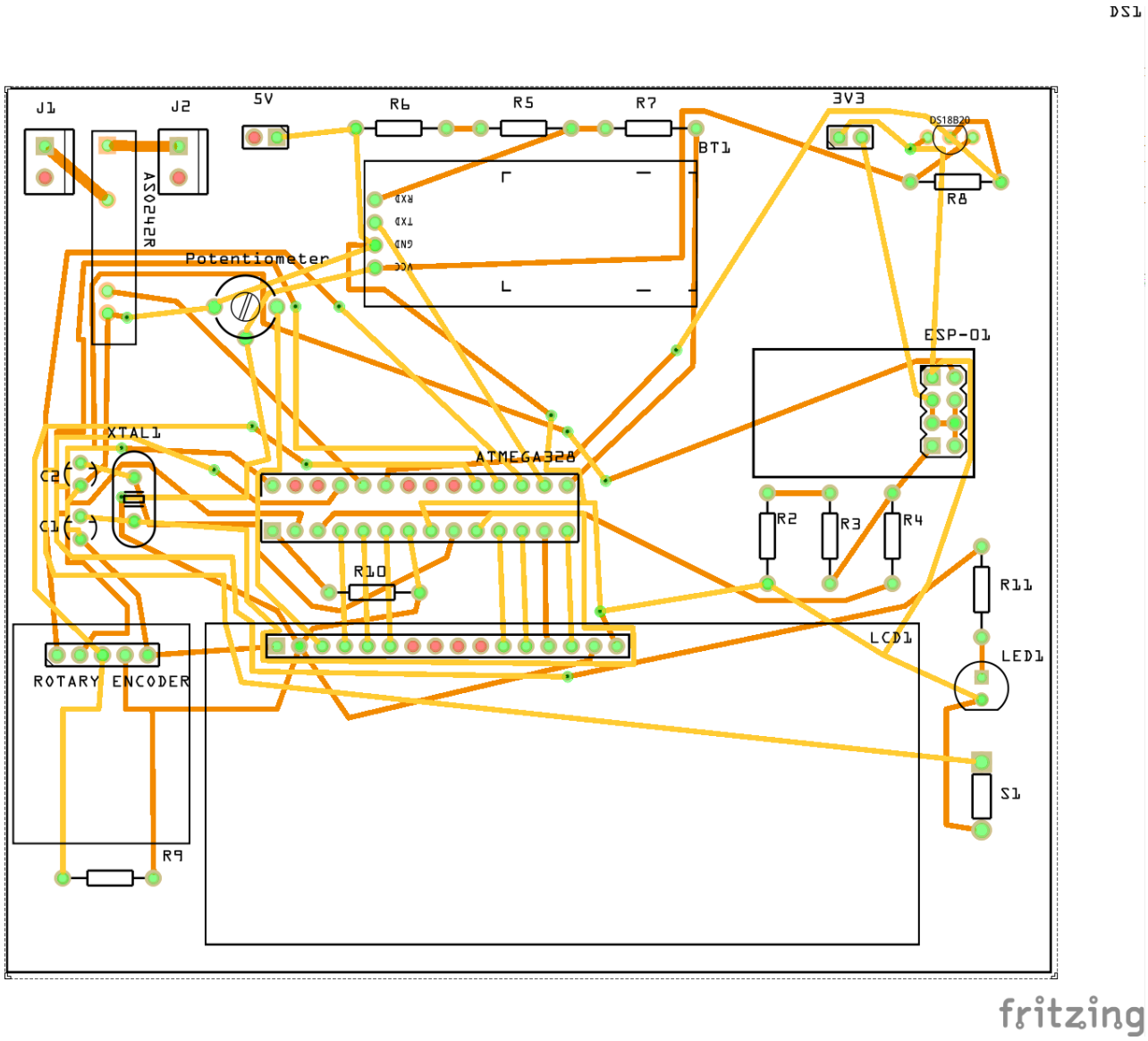
5 Bilagor

5.1 Kopplingsschema

Notera att i kopplingsschemat nedan så dras ledningarna till pinnarna på själva mikrokontrollern (Atmega328P-PU), som skiljer sig från layouten på Arduino Uno.



5.2 PCB


DS1
J1
J2
5V
R6
R5
R7
3V3
DS18B20
BT1
R8
AS0242R
Potentiometer
ESP-01
XTAL1
C2
C1
ATMEGA328
R2
R3
R4
R10
R11
LCD1
LED1
ROTARY ENCODER
S1
R9
fritzing