Institutionen för teknik, matematik och datavetenskap – TMD

EXAMENSARBETE
2004:RT02

IQ-Mower: Styrning och positionering av mobil robot

Micael Karlsson

HÖGSKOLAN
TROLLHÄTTAN • UDDEVALLA

EXAMENSARBETE

IQ-Mower: Styrning och positionering av mobil robot

Micael Karlsson

Sammanfattning

Examensarbetet inleddes med att inhämta kunskaper i att programmera Atmel AVR microcontroller i programspråket C. För att minska och snabba upp positioneringsberäkningen omarbetades en tidigare utvecklad algoritm så att denna minskades i komplexitet och antal beräkningssteg. För att få kommunikation med användaren har program skapats för att använda LCD-display och telefontangentbord. Dessa kopplas till AVR microcontrollern som fungerar som en hjärna i systemet. AVR ATmega128L är den krets som används då den har tillräckligt antal IO-pinnar för att kunna kommunicera med den övriga utrustningen. För kommunikationen mellan AVR- och FPGA-kretsarna används ett adress- och databussprotokoll. Databussprotokollet är halv duplex medan adressbussen är simplex med AVR-kretsen som sändare.

Tester av programvaran har gjort med hjälp av utrustning som byggts för detta ändamål. Testerna visade att både positioneringsalgoritmen och kommunikationsprotokoll enligt tidigare fastställd specifikation.

Utgivare:	Högskolan Trollhättan/Uddevalla, Institutionen för teknik, matematik och datavetenskap, Box 957, 461 29 Trollhättan Tel: 0520-47 50 00 Fax: 0520-47 50 99 Web: www.htu.se
Examinator:	Peder Carlsson, HTU
Handledare:	Peder Carlsson, HTU
Nivå:	Fördjupningsnivå 1 **Poäng**: 10
Datum:	2004-06-10
Nyckelord:	Positionering, AVR, Mobil robot, C-programmering, Trilateration

DEGREE PROJECT

IQ-Mower: Control and positioning of a mobile robot

Micael Karlsson

Summary

This degree project started of with gaining knowledge in programming the ATMEL AVR microcontroller in C. In order to reduce the amount of code and an increase in speed the algorithm that have been used in an earlier project is no longer used because it is too complex. The new one is a lot easier to implement in C-code and it is faster.

To communicate with the user a LCD-display and a 16-key keyboard is used. This peripheral equipment is connected to the AVR microcontroller who is functioning as master in the system. To control the electric and cutting motors as well as all the other hardware the AVR is connected to a FPGA-circuit, functioning as slave. For the communication between the AVR- and FPGA-circuits an address- and databusprotocol are used. The databus is half duplex while the addressbus is simplex with he AVR-circuit as sender.

Testing of the program has shown that the positioningalgorithm is functioning very well. And the communication protocol that has been written in C is working and has been tested with equipment specialy designed for this purpose.

Publisher:	University of Trollhättan/Uddevalla, Department of Technology, Mathematics and Computer Science, Box 957, S-461 29 Trollhättan, SWEDEN Phone: + 46 520 47 50 00 Fax: + 46 520 47 50 99 Web: www.htu.se		
Examiner:	Peder Carlsson, HTU		
Advisor:	Peder Carlsson, HTU		
Subject:	Electrical Engineering	**Language**:	Swedish
Level:	Advanced	**Credits**:	10 Swedish, 15 ECTS credits
Number:	2004:E000	**Date**:	Juni 10, 2004
Keywords	Positioning, AVR, Mobile robot, C-programming, Trilateration		

Förord

Denna rapport är det skriftliga dokumentet av mitt examensarbete om 10 poäng som avslutar Magisterutbildningen inom robotteknik, 40p. För att tillgodogöra sig rapporten fullt ut bör läsaren ha grundläggande kunskaper i C-programmering, elektronik, till viss del i mikrodatorer samt matematik.

Ett tack till min handledare och examinator Peder Carlsson och Gunne Anderson som hjälpt mig med utlåning av allt från LCD-display till motstånd.

Jag vill särskilt rikta ett varmt tack till Mattias Ottoson som ställt upp och svarat på alla mina frågor om C-programmering och initiering av LCD-displayer.

Innehållsförteckning

Bilagor

Figurförteckning

Nomenklatur

F_i	Fyr *i*
x_i	x-koordinat för fyr *i*
y_i	y-koordinat för fyr *i*
y_{ij}	Ekvation för skärningslinjen mellan fyr *i* och *j*
k_{ij}	Riktningskefficient för skärningslinjen mellan fyr *i* och *j*
A_{ij}	Höjdkoefficient för skärningslinjen mellan fyr *i* och *j*
FPGA	***F***ield ***P***rogrammable ***G***ate ***A***rray
VHDL	***V***ery High Speed Integrated Circuit ***H***ardware ***D***escription ***L***anguage
0x0F	Anger att ett tal är skrivet på hexadecimalt format, dvs basen 16, i detta fall just det decimala talet 15

1 Inledning

Var är jag? Vart skall jag? Hur tar jag mig dit?

Det är dessa frågor som navigering, inte bara av mobila robotar, skall ge svar på. Frågan ”Var är jag?” besvaras av det positioneringssystem som används oavsett om det är av typen absolut eller relativ. De två andra frågorna besvaras av navigeringssystemet som beroende av användningsområde fattar beslut om hur roboten skall förflytta sig.

Det finns ett antal självgående gräsklippare på marknaden men dessa har det gemensamt att alla är beroende av att användaren antingen gräver ner eller fäster någon form av slinga runt området som skall klippas samt runt rabatter och andra områden som ej skall klippas. Detta ger naturligtvis ett system som är betydligt mer stelbent än det system som kommer att vara slutprodukten där detta examensarbete är en del. Största skillnaden är att IQ-Mower, som är aktuellt produktnamn, kommer att vara helt oberoende av olika former av slingor. Istället används radiofyrar för att beräkna gräsklipparens position. Innan gräsklipparen kan klippa en ny gräsmatta behöver användaren med assistans av, i IQ-Mower, inbyggd mjukvara sätta ut fyrar så att dessa placeras på bästa sätt. Därefter markeras de områden som skall klippas respektive ej klippas. Efter detta kommer det ej att behöva göras något mer för att klippa gräsmattan.

Om gräsmattan ändras eller om områden som ej skall klippas ändras behövs inget flyttande av slingor, utan det enda som krävs är att mata in det nya området och gräsklipparen är redo att klippa den nya gräsmattan.

Ett annan fördel är att det kommer att finnas inbyggd intelligens i gräsklipparen som kommer att beräkna hur gräsmattan skall klippas på bästa sätt, ej som de som finns på marknaden som klipper mer eller mindre slumpmässigt.

1.1 Bakgrund

För att det överhuvudtaget skall vara intressant att använda någon form av mobil robot eller autonomt fordon skall det kunna förflytta sig från en punkt till en annan utan att någon människa styr eller riktar fordonet mot slutpunkten. Med andra ord behövs ett stabilt positionerings- respektive navigeringssystem. Längre fram i rapporten kommer några olika positioneringssystem att diskuteras.

2 Syfte och mål

Att med hjälp av en AVR microcontroller styra en gräsklippare så att denna i slutänden skall kunna klippa en godtycklig yta. För positionering av gräsklipparen används minst tre aktiva fyrar, antalet är beroende dels av ytan som skall klippas dels av om det finns hinder som skärmar av fyr eller gräsklippare.

För att gräsklipparen skall kunna kommunicera med användaren används en LCD-display samt tangentbord av typen telefontangentbord med 16 knappar. Funktioner för initiering och avkodning av dessa skall skrivas i programspråket C. Ett adress- och databuss protokoll för kommunikation mellan hårdvaran kommer att skapas.

Dessutom kommer en snabbare och enklare positioneringsalgoritm tas fram och testas på en AVR microcontroller.

3 Förutsättningar

En gräsklippare finns på institutionen samt fyrar och en FPGA-krets som används för motorstyrning och inmätning av avstånd till fyrarna. Den programvara som kommer att användas är gratis och används dessutom på HTU. Övrig utrustning har köpts in av författaren utom den låda med bland annat LCD-display och tangentbord som lånats på HTU.

3.1 Avgränsningar

Huvuddelen av arbetet ligger i att skapa ett program i programmeringsspråket C för att positionera och styra gräsklipparen. Ingen hårdvaruprogrammering i VHDL eller byggande av fyrar ingår i detta examensarbete. Den FPGA-krets som används kommer att behandlas som en ”blackbox”, det vill säga: Skicka ett kommando för att få en viss händelse utförd, hur detta går till internt tas det ingen hänsyn till.

4 Positionering

För att bestämma ett fordons position finns ett antal olika metoder som brukar delas in i två grupper, absolut och relativ positionering [1]. För relativ positionering, det vill säga vilken sträcka som fordonet färdats kan delas in i två undergrupper, vägmätning (eng. Odometry) och tröghetsnavigering (eng. Inertial navigation).

De felkällor som kan uppstå vid vägmätning delas in i två grupper systematiska och ickesystematiska fel[3], dessa är till exempel:

- Systematiska fel
  - Olika hjuldiameter
  - Skalfel
  - Verklig hjulbas skiljer sig från nominell hjulbas
  - Felvinklade hjul
  - Sensorfel
- Ickesystematiska fel
  - Ojämt underlag
  - Kör över oväntade föremål på marken

- Slirande hjul som kan bero på:
  - Halt underlag
  - Hjulspinn
  - Slirar vid för snabb sväng
  - Inre och yttre krafter
  - Hjul förlorar tillfälligt kontakt med underlaget

Vägmätning är inte den effektivaste metoden utomhus på till exempel gräsmattor som kan vara både hala och ojämna, men i innomhusmiljöer fungerar det bra tillsammans med kompass[4].
För absolut navigering, en position ges, finns flera olika metoder några av dessa är:

- Olika former av kompass
- Aktiva fyrar
- Global Positioning System, GPS
- Landmärkesnavigering

Det finns andra metoder men dessa torde vara de mest använda och de som har undersökts i den litteraturstudie som ingår i detta examensarbete.

De ovan nämnda metoderna har naturligtvis alla sina för och nackdelar [1], se vidare under kapitel 4.6 för en kort summering av de olika metoderna.

4.1 Kompass

På marknaden finns ett antal olika kompasser i varierande prislägen allt från hobbyrobotar till industriapplikationer. De vanligaste formerna av kompasser är:

- Mekaniskt magnetkompass
- Fluxgate kompass
- Hall-effekt kompass
- Magnetoresistiva kompasser
- Magnetoelastiska kompasser

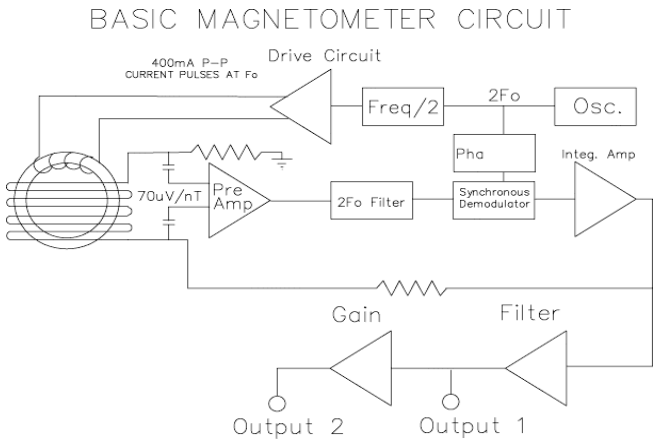
4.1.1 Mekaniska magnetkompass

En mekanisk magnetkompass läser helt enkelt av jordens magnetfält och med hjälp av det bestäms positionen av fordonet. För montering i mobila robotar saluför t.ex.

Honeywell ett antal olika två-axliga kompasser i prisklasser från $89.00[1] upp till $699.00.

4.1.2 Fluxgate kompass

Fluxgate är en sensor som känner av jordens magnetfält. I figur 3.1 nedan visas ett kopplingsschema för en fluxgatekompass. Detta är dock ingen ny uppfinning utan den konstruerades under andra världskriget för att upptäcka lågt flygande flygplan.



Figur 4.1 Kopplingsschema Fluxgate kompass [2]

4.2 Aktiva fyrar

Att använda aktiva fyrar som navigationshjälpmedel är vanligast på flygplan och båtar så väl som i kommersiella mobila robotsystem. Nackdelarna med aktiva fyrar är naturligtvis att det krävs god noggrannhet då fyrarna placeras ut samt att visst underhåll krävs. Det finns två olika tekniker för att beräkna positionen triangulering och trilateration.

Triangulering går ut på att vinkeln till fyrarna mäts och med hjälp av dessa beräknas positionen. Med denna metod krävs det med andra ord noggrann placering av fyrar samt att fordonet är utrustat med en exakt utrustning för vinkelmätning då ett litet vinkelfel kan generera stora fel i positionen. Vanliga metoder för triangulering [1] är:

[1] Priserna hämtade från Honeywells webplats 2004-04-13
http://shop.ssec.honeywell.com/shopdisplayproducts.asp?id=6&subcat=12&cat=HMR3xxx+%2F+HMR4xxx+%2D+Products+and+Accessories

- Geometrisk triangulering

  En metod med begränsningar som att positionering kan endast göras då fordonet befinner sig innanför den triangel som de aktiva fyrarna bildar samt vissa områden utanför denna men dessa är dock svåra att bestämma och starkt beroende av hur vinklarna är definierade.

- Geometrisk skärning av cirklar

  En metod som ger stora fel då fyrar och robot ligger på eller nära samma cirkel.

- Newton-Raphsons metod

  Denna metod misslyckas med att producera en position om robotens utgångsposition och orientering är allt för långt ifrån dess egentliga position och orientering.

Med andra ord bör alltså minst två av dessa metoder kombineras för att på så vis erhålla en stabilare och säkrare positioneringsmetod.

Den andra metoden som använder aktiva fyrar är som nämnts tidigare, trilateration. Trilateration använder istället avståndet till fyrarna för att beräkna positionen oftast görs detta med någon form av högfrekvent signal.

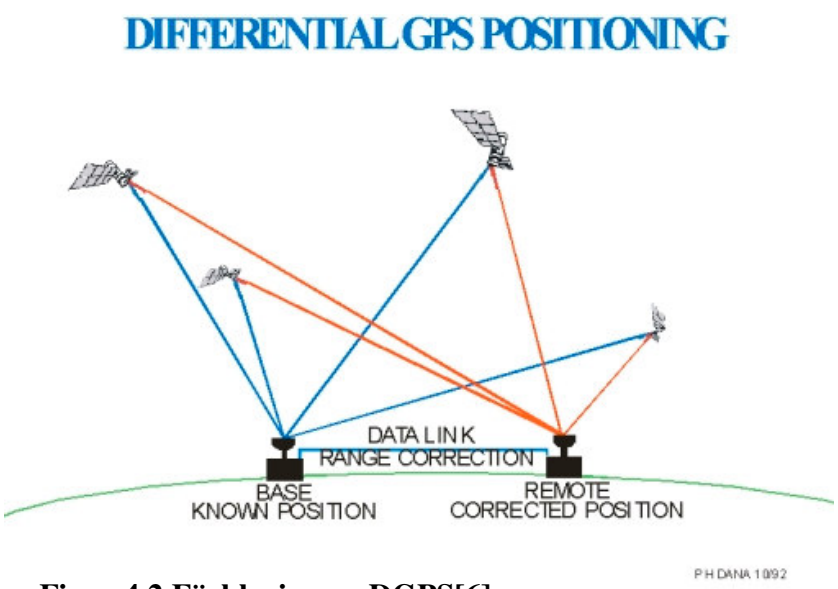
I aktuellt projekt anropas fyrarna med en fyrspecifik signal varefter fyren svarar och med hjälp av den tid som åtgår beräknas avstånden till de tre fyrarna.

Den metod som används i detta projekt använder som en del andra metoder cirkelns eller sfärens ekvation, beroende på om det är i 2 eller 3 dimensioner som positionen skall bestämmas. En noggrannare beskrivning av algoritmen redovisas i kapitel 4.

4.3 Global Positioning System, GPS

GPS grundades och finansierades av U.S. Department Of Defence (DOD) och sköts av amerikanska försvaret. Sammanlagt består GPS-systemet av 24 sateliter i omloppsbanor runt jorden. För att bestämma position används minst tre sateliter vilket producerar en position med ett fel som varierar beroende på källan, allt från ett tiotal meter upp till cirka 100[2]. Anledningen till att GPS ger ett visst fel beror till viss del på att en störsignal, eller SA (Selective Avaliability) adderas till de signaler som används av civila GPS-mottagare. SA lägger till ett varierande tidsfel till klocksignalen som ger upphov till det positionsfel som uppstår. Den positioneringsmetod som används är trilateration. Om Differentiell GPS, DGPS, används minskar felet betydligt. Anledningen till detta är att felet mäts på kända platser/positioner varifrån korrigeringsdata sänds till GPS-mottagaren. En förklaring av DGPS visas i figur 3.2 nedan.

[2] Då uppgifterna varierar kraftigt anges inga exakta mått på felet, (*författarens kommentar)*



Figur 4.2 Förklaring av DGPS[6]

4.4 Landmärkesnavigering

Landmärken, det vill säga objekt som roboten kan känna igen med hjälp av sensorer, Dessa landmärken kan vara geometriska former som trianglar, cirklar eller rektanglar. För att ge roboten ytterligare information kan dessa innehålla någon form av streckkod.

Landmärken kan delas in i två olika grupper, dels naturliga, dels artificiella landmärken. Fördelen med de naturliga är att det inte krävs någon form av ingrepp i miljön. Men det är inte alltid som förhållandena är optimala som för de artificiella där kontrast- och formförhållanden kan optimeras. Andra fördelar med artificiella landmärken är priset, exakt kunskap om form och storlek.

Nackdelar med alla former av landmärkesnavigering är bland annat att bestämma en exakt position om roboten ej befinner sig nära eller i en bra vinkel, alltså en vinkel i förhållande till landmärket som gör det möjligt att identifiera landmärket och bestämma fordonets position.

Den största nackdelen är dock den processorkraft som krävs för visionbaserad igenkänning av landmärken.

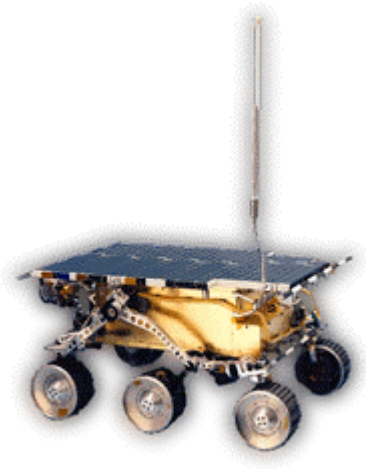
4.5 Visionbaserad navigering[8]

Att använda någon form av kamera för att navigera har funnits under ett flertal år. Under 1980-talet och första halvan av 1990-talet hade visionbaserad navigering av mobila robotar en framskjuten plats.

När vision används för navigering utomhus används den oftast för att undvika hinder, hitta landmärken, bygga upp och uppdatera kartor samt uppskatta robotens position.

Ofta delas navigering med vision in i två kategorier, navigering i strukturerad respektive ostrukturerad omgivning. I strukturerad omgivning innebär det ofta någon form av vägföljning vilket betyder att roboten kan följa en väg eller skilja på vägbanor för att förflyttas längs vägbanan och naturligtvis undvika eventuella hinder som kan finnas. Exempel på ett liknande system är t.ex. Navlab[3] som används i olika former av fordon för att känna igen människa av olika storlekar och kroppsställningar[9] eller att styra fordon[4] av olika storlekar.

I ostrukturerade miljöer som t.ex. Månen eller mars används systemet för att upptäcka hinder, en av de på senare år mest kända exemplen är troligen Mars Pathfinder Rover (Sojourner), se figur 3.3 nedan.



Figur 4.3 Sojourner[10]

4.6 Summering av positioneringsmetoder

Det finns ytterligare metoder än de som nämnts i föregående kapitel men dessa är inte så vanliga i kommersiella sammanhang. Ibland kombineras en relativ och en absolut positioneringsmetod för att bygga in säkerhet i positioneringssystemet. I detta examensarbete är tanken att bygga in en modul för att göra en rimlighetsbedömning av de mätvärden som fås från mät/styr-kretsen, se vidare under kapitel 5. Tanken är att beräkna ett uppskattat värde utifrån föregående mätvärden och fordonets riktning. Om dessa, det beräknade och inmätta värde, befinner sig inom ett visst område kan den nya positionsbestämningen ses som riktig. Om den avviker för mycket kan en ny positionsbestämning göras.

[3] http://www.navlab.org/
[4] video som visar detta finns på: http://www.navlab.org/navlab_video.html

I och med att datorkraften ökat, ökar naturligtvis möjligheterna att utveckla allt mer avancerade navigeringsalgoritmer som adaptiva eller baserade på en neuro-fuzzy[5].

Det kan konstateras att den metod som skall användas är till stor del beroende av vilken miljö som fordonet skall navigera i. En annan synpunkt är naturligtvis kostnaden. Att använda DGPS eller vanlig GPS i en innomhusmiljö är inget att rekommendera eftersom satelitsignalerna till stor del störs ut av den omgivande byggnaden, det kan faktiskt räcka med lövverk för att störa signalen. På samma sätt kan det ses som olämpligt att välja vägmätning på en ojämn och kanske hal gräsmatta.

En relevant fråga i sammanhanget är: Varför aktiva fyrar och trilateration?
Svaret är att metoden är relativt billig och noggrann kombination av utrustning och metod under förutsättning att området ej är för stort då ultraljud har begränsad räckvidd. Att området ej innehåller allt för mycket som avskärmar ljud eller att allt för många robotar använder samma fyrar. Dessa nackdelar är för detta projekt inget negativt eftersom det handlar om en gräsklippare vilket leder till begränsade ytor som skall täckas, gräsmattor är relativt öppna ytor samt att det endast finns en gräsklippare.
Andra tungt vägande skäl för detta val är att fyrar finns tillgängliga och att positioneringsalgoritmen är framtagen av författaren. Detta ger alltså låga kostnader och inga problem med anpassning av mjuk- resp hårdvara då denna är konstruerad tidigare och konstrueras under detta examensarbete.

5 Positioneringsalgoritm

Den algoritm som arbetats fram för detta projekt bygger på metoden räta linjer[7]. De nackdelar som finns i metoden har till största del undvikits genom en ny angreppsmetod. I stället för att utgå från en förenklad uppställning av fyrar har en mer allmän uppställning studerats, se figur 4.1, detta leder till att de ekvationer som erhålls är litet mer komplicerade.

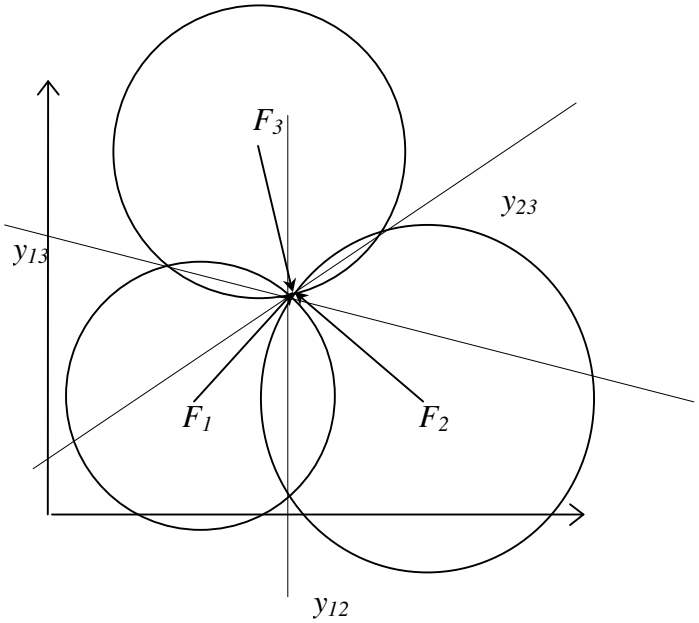
Längden från respektive fyr till roboten ger radien i de tre cirklarna som används för att matematiskt beräkna positionen.

Ekvationerna för fyrarna 1, 2 och 3 är som följer:

$$F_1 : (x - x_1)^2 + (y - y_1)^2 = L_1^{\,2} \tag{4.1}$$

$$F_2 : (x - x_2)^2 + (y - y_2)^2 = L_2^{\,2} \tag{4.2}$$

$$F_3 : (x - x_3)^2 + (y - y_3)^2 = L_3^{\,2} \tag{4.3}$$



Figur 5.1 Metoden Räta linjer

Därefter utförs subtraktionerna F_1-F_2, F_1-F_3 och F_2-F_3 vilket ger ekvationerna för skärningslinjer mellan cirkelparen.

$$y_{12} = -\frac{x_2 - x_1}{y_2 - y_1} x + \frac{L_1^{\,2} - L_2^{\,2} - x_1^{\,2} + x_2^{\,2} - y_1^{\,2} + y_2^{\,2}}{2(y_2 - y_1)} \tag{4.4}$$

$$y_{13} = -\frac{x_3 - x_1}{y_3 - y_1} x + \frac{L_1^{\,2} - L_3^{\,2} - x_1^{\,2} + x_3^{\,2} - y_1^{\,2} + y_3^{\,2}}{2(y_3 - y_1)} \tag{4.5}$$

$$y_{23} = -\frac{x_3 - x_2}{y_3 - y_2} x + \frac{L_2^{\,2} - L_3^{\,2} - x_2^{\,2} + x_3^{\,2} - y_2^{\,2} + y_3^{\,2}}{2(y_3 - y_2)} \tag{4.6}$$

Förenkla ekvationerna 4.4 – 4.6:

$$y_{12} = k_{12}x + A_{12} \tag{4.7}$$

$$y_{13} = k_{13}x + A_{13} \tag{4.8}$$

$$y_{23} = k_{23}x + A_{23} \tag{4.9}$$

För att lösa ut x sätts följande likhet upp:

$$y_{12} = y_{13} \quad \Rightarrow \quad k_{12}x + A_{12} = k_{13}x + A_{13} \tag{4.10}$$

Ur 4.10 löses x ut och ger resultatet:

$$x = \frac{A_{13} - A_{12}}{k_{12} - k_{13}} \tag{4.11}$$

x insättes i 4.9:

$$y = k_{23}x + A_{23} \tag{4.12}$$

Detta ger alltså positionen för roboten under förutsättning att cirklarna skär eller tangerar varandra. Om detta ej inträffar behöver ytterligare beräkningar utföras, dessa redovisas i Appendix A.

6 Positionerings- och styrsystem

Positionerings- och styrsystem består av en överordnad AVR microcontroller[5] och en underordnad FPGA-krets[6], dessa är sammankopplade med hjälp av en adress- och databuss samt två kontrollbitar för att ge information till FPGA om den skall läsa eller skriva till databussen, som är 12-bitar. På adressbussen skickas kommandon från AVR om FPGA skall adressera fyrar eller om motorer skall köras framåt, bakåt, svänga vänster, svänga höger eller stanna. Adressbussen används även för att skicka kommandon så att FPGA lägger ut längder till fyrar på databussen.

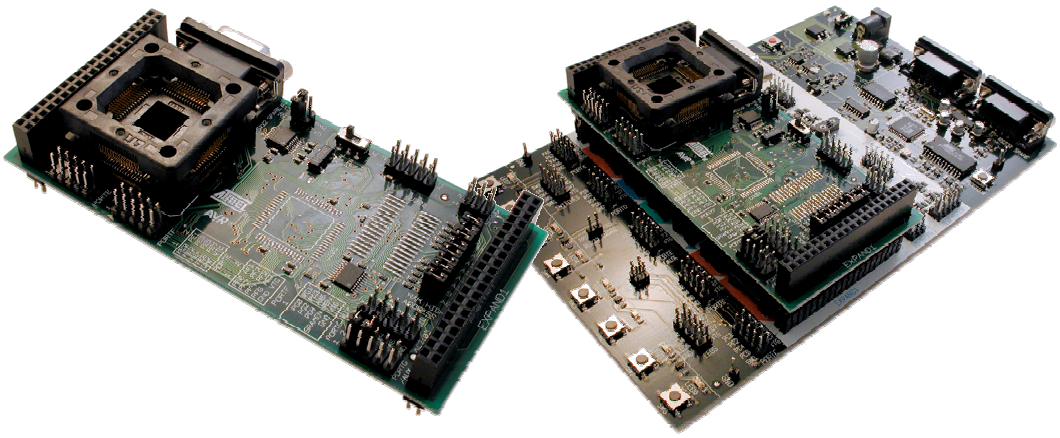
Fördelen med att använda en FPGA är att det är en oerhört flexibel krets som kan programmeras enligt användarens önskemål, mycket snabb då den kod som skrivits byggs upp som elektriska kretsar.

[5] I fortsättningen benämns denna endast AVR
[6] Denna benämns i fortsättningen endast FPGA

7 Hårdvara

Den AVR microcontroller som används är ATmega128L en krets i TQFP64-kapsel. Det finns 53 st I/O-pinnar som kan växla mellan att vara in- eller utgång beroende på behov[11]. Till denna krets kopplas tangentbord av typen telefontangentbord 16 knappar och en LCD-display 4x40 tecken av märket Seiko. Det skulle visa sig att denna display övergavs till förmån för en som finns på HTU, se vidare under kapitel 6.3.2.

ATmega128L monteras i STK501, ett färdigbyggt system, som finns att inhandla på t.ex. www.elfa.se, se figur 6.1 nedan.



Figur 6.1 STK501 (vänster) och STK501 monterad på STK500

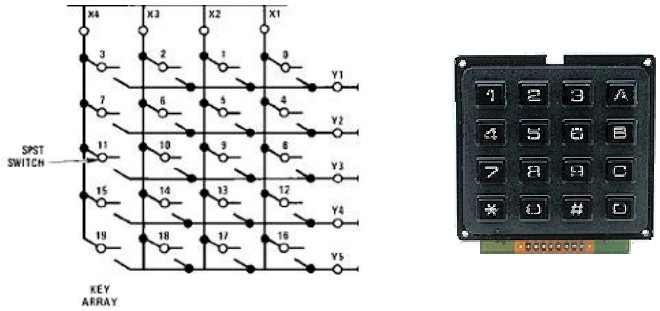
Varje del av hårdvaran testas var för sig, detta är mycket viktigt för att undanröja problem eller fel i hårdvaran. Först då varje del av hårdvaran fungerar helt enligt önskemål bör delar kopplas samman till större enheter. Om testning ej utförs och problem uppstår kommer dessa problem troligen att bli betydligt mer komplexa och svårare att överblicka.

I detta examensarbete är den största möjligheten till fel att både FPGA och AVR försöker skriva till databussen samtidigt vilket kan resultera i märkliga fel som kan vara svåra att spåra till vilken krets som försöker skriva vid fel tillfälle.

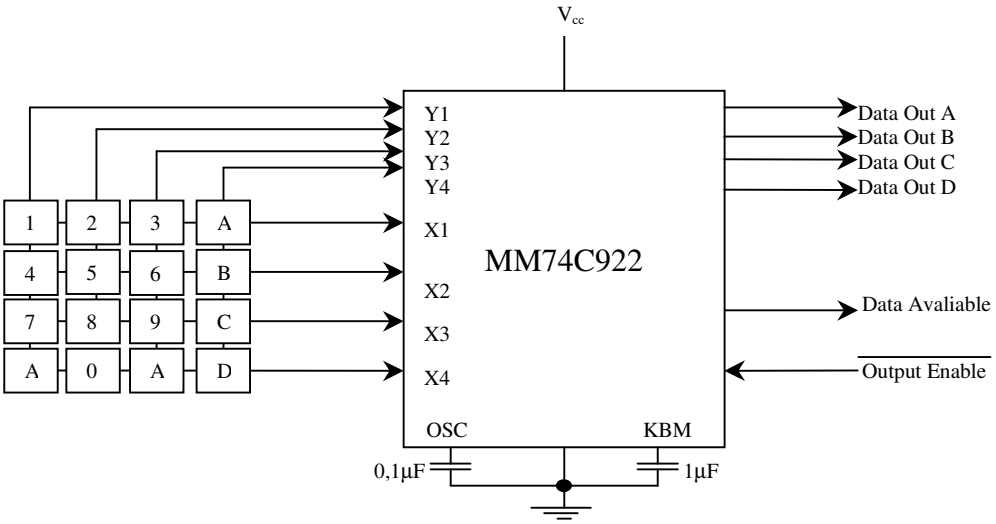
Det kan ses som om det är att vara försiktig i överkant men då problemen hopas och är svåra att överblicka som en noggrann testning skulle ha undanröjt problemen. Därmed ej sagt att denna metod är felfri och ger en krets som alltid fungerar men problemen som uppstår tenderar att bli mindre komplexa och enklare att reda ut och avhjälpa.

7.1.1 Tangentbord

För att underlätta programmering och minska antalet portar används avkodaren MM74C922[7] för tangentbordet. Denna krets fungerar i korthet så att den känner av då en tangent trycks ned, då går *Data avaliable* hög, därefter skickas en nolla från AVR till *!Output Enable*[8] varefter avkodaren lägger ut tangentkoden för nedtryckt tangent på *Data out*. Se figurerna 6.2, för uppbyggnad av och bild på tangentbord, och 6.3 för schematisk inkoppling av kretsen.



Figur 7.2 a)Tangentbordets uppbyggnad [12], b) samt bild hämtad från www.elfa.se

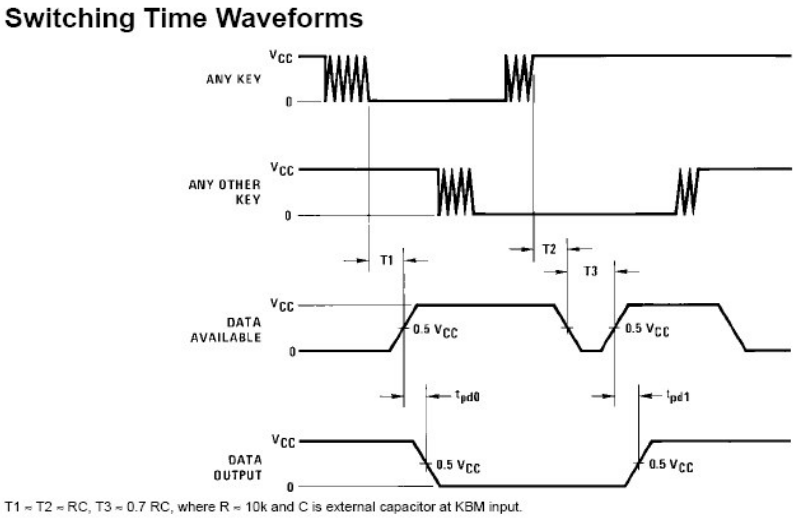


Figur 7.3 Kopplingsschema för telefontangentbord

Kretsen MM74C922 kräver endast två utomstående komponenter, två kondensatorer. Kondensatorn kopplad till KBM, Keybounce Mask, undanröjer problemet med kontaktstutsar dvs. samma tangent registreras flera gånger trots att användaren bara

[7] För utförligare information hänvisas till databladet: http://www.elfa.se/pdf/73/737/07371131.pdf
[8] *!Output Enable* är alltså inverterad ingång

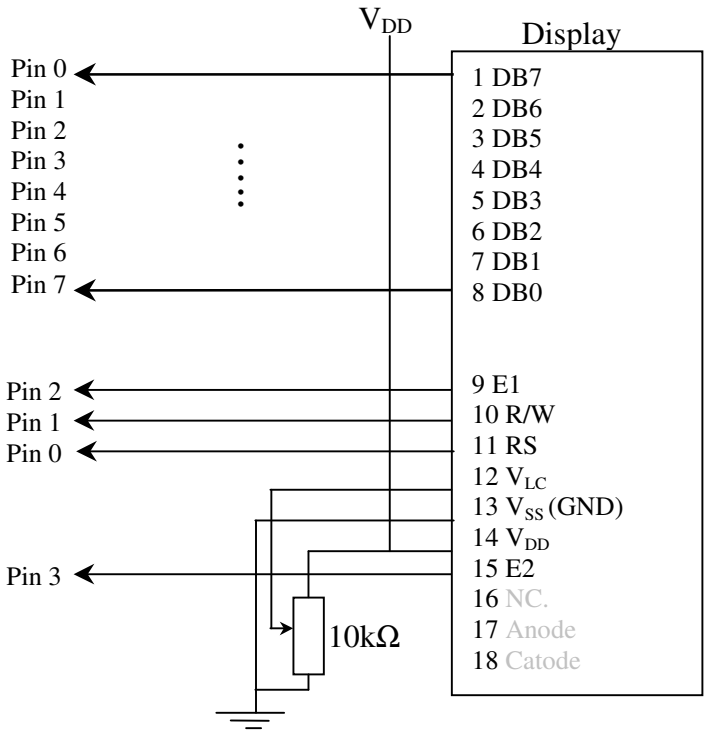
tryckt ner tangenten en gång. Med en kondensator på 1 µF fås alltså tiden T_1 och T_2 till 10 ms, se figur 6.4 nedan.



Figur 7.4 Eliminering av kontaktsdutsar i MM74C922[12]

7.1.2 LCD-display

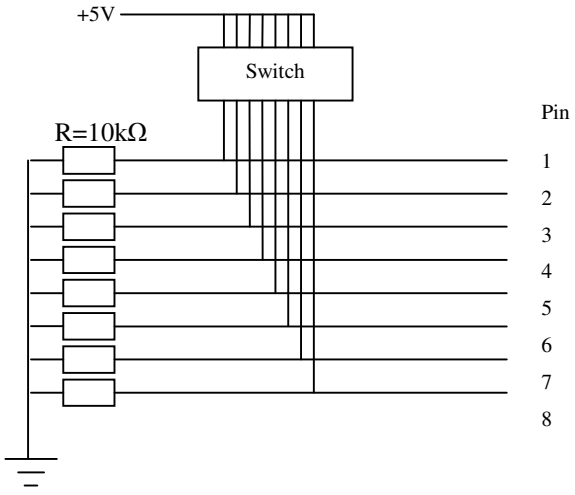
För inkoppling av display krävs en yttre komponent, en vridpotentiometer på 10kΩ. Under förutsättning att Atmels Starterkit STK500 används kan drivspänning, V_{CC} och jord, GND, användas från pin 9 och 10 på portuttag. Om denna ej används krävs extern spänningskälla på +5V samt drivspänning för de flytande kristallerna. För kopplingsschema se figur 6.5 nedan. Observera att detta kopplingsschema gäller för Seiko LCD-display 4x40(rader x tecken) med drivkrets Samsung KS0066. Observera också att pinnarna 17 och 18, se figur 6.5, används endast för displayer men bakgrundsbelysning. E1 och E2 används för att styra vilken del av displayen som skall skrivas till då denna display kan ses som 2 stycken 2x40 displayer som det är alltså möjligt att skriva till en eller båda samtidigt.



Figur 7.5 Inkoppling av Seiko LCD-display 4x40 tecken

7.1.3 Switchar för testning av adress-och databussprotokoll

En enkel uppkoppling av switchar, se figur 6.6, kommer att användas för att testa mjukvaran. Dessa fungerar som helt vanliga brytare. I figur nedan visas kopplingsschema för 8 switchar, om fler behövs är det enkelt att utöka antalet.



Figur 7.6 Kopplingsschema för switchar för test av adress- och databussprotokoll

7.2 Kommunikation mellan AVR- och FPGA-kretsarna

Den kommunikation som behövs mellan AVR och FPGA går som sagts tidigare ut på att AVR skickar kommandon till FPGA om den skall adressera fyrar, styra driv- eller klippmotorer. För att köra roboten framåt läggs koden ”000001” ut på adressbussen samt kontrollbitarna *enadress* ges ”1” och *rw* ”0”. Detta betyder att FPGA skall läsa från adressbussen och i sin tur adressera motorerna som skall driva roboten framåt med en mjuk acceleration annars finns risk att roboten stegras då drivhjulen sitter långt bak på roboten.

Adressering och inmätning av längder till fyrarna går till på likartat sätt, dock med den skillnaden att AVR läser av en ”tillståndsvektor” och då denna har värdet 0x01 skickar AVR kommando till FPGA att lägga ut detta på databussen.

För en sammanställning av de kommandon som kan skickas till FPGA hänvisas till Appendix C.

7.3 Mjukvara

Den programvara som används för att skriva och kompilera C-kod är WinAVR (uttalas ”whenever”) som är freeware precis som den editor, Programmers Notepad 2 som ingår i det installationspaket som installeras[9]. Den enda nackdelen som upptäckts med detta ”paket” är att det kräver lite mer av programmeraren än vad till exempel kommersiella utvecklingspaket gör. Det ingår ingen windowsbaserad kompilator och kodskrivaren kan på så vis styra exakt hur kompilatorn skall kompilera koden. I korthet är det makefile-filen som ger kompilatorn information om hur C-koden skall kompileras vilken sorts filer som skall genereras samt vilken AVR-modell som koden skall kompileras för, eftersom det finns processorknuten kringutrustning till olika AVR.

Programmers Notepad 2 har en trevlig funktion som kan tyckas borde vara standard i editorer, det går att minimera funktioner, loopar och liknande som uppskattas då programmen ökar i storlek. Det fungerar på samma sätt som katalogstrukturen i Windows utforskaren men istället för ”mappar” är det funktioner och loopar som kan minimeras eller maximeras.

För att överföra den kompilerade programkoden till AVR som sitter i Atmels starter kit STK500 eller expansionsmodulen STK501 beroende på AVR-modell används AVR Studio 4.08 från Atmel, också detta är freeware. Det filformatet som används kallas coff för AVR Studio 3.xx och extended coff då AVR Studio 4.xx används. Fördelen med detta format är att det ger möjligheten att simulera C-koden och följa det som sker i koden rad för rad och se hur det påverkar portar, räknare med mera. Möjligheten finns att se hur lång tid det tar att exekvera en funktion eller ett enskilt kommando. Programmen som används är stabila, speciellt AVR Studio, och är enkla att ”komma in igång med”.

[9] Mer information finns på följande URL: http://winavr.sourceforge.net/index.html

7.3.1 Styr- och positioneringsfunktioner

Den viktigaste delen i programvaran som tagits fram är dels positioneringsfunktion dels de funktioner som styr gräsklipparen. Nedan visas ett flödesschema för hur flödet kan fungera för att klippa en gräsmatta. De delar som utelämnats är beräkning av hur gräsklipparen skall navigera då detta ej ingår i examensarbetet. Denna del kan naturligtvis enkelt infogas i koden.

7.3.2 Tangentbord

I kapitel 7.1.1 beskrivs hur hårdvaran för tangentbordet är uppbyggd, i detta kapitel ges en förklaring till hur mjukvaran är konstruerad. Då *Data avaliable* går hög sätts *!Output Enable* till logiskt låg nivå därefter krävs en kort stund innan data läses av för att kretsen MC74C992 skall hinna lägga ut tangentpositionen, ej vilken tangent som tryckts ned.

Nästa steg är alltså att tolka vilken tangent som tryckts ned för att kunna göra något meningsfullt med informationen som att mata ut den till displayen, starta klippning eller något annat.

Kopplingsschema för telefontangentbordet och avkodningskretsen MM74C922 visas i figur 6.3. För pinnkonfiguration hänvisas till kretsens datablad[12].

7.3.3 LCD-display

Det första som måste göras då displayen startas eller efter en reset är att initiera den, dvs hur den skall fungera i fråga om antal rader, vilket teckenformat, skall data skickas med fyra eller åtta bitar, skall det finnas markör och skall den blinka med mera. Denna information hämtas från tillverkaren av displayen eller styrkretsen och kan variera mellan olika tillverkare.

Första valet av LCD display var alltså av märket Seiko, denna övergavs dock då initieringen inte fungerade trots upprepade försök och oräkneligt antal tester med olika delaytider mellan initieringsstegen och olika kodkombinationer. Det var lätt att förstå hur Sisyfos upplevde sin situation då det hela tiden dök upp nya problem[10] som gjorde att det krävdes att börja om från början gång på gång.

Seiko-displayen övergavs för en mindre display som används på HTU av flera anledningar.

Den display som används på HTU är en 2x8 display med kontrollkrets HD44780 och den kan initieras enligt följande:

- 8-bitars data
- 2 raders display
- Tecken uppbyggda av 5x8 punkter

[10] I stället för ett stenblock som hela tiden ramlade tillbaka ned

- Blinkande markör
- Automatiskt flytta markören till höger efter inmatat tecken
- Rensa displayen

Det finns naturligtvis ett stort antal olika möjligheter för hur en display kan initieras, dessa redovisas i databladet för displayen eller kontrollkretsen.

För programkod se Appendix C.

7.3.4 Kommunikation mellan AVR och FPGA

Som tidigare nämnts är det AVR som är master/överordnad vilket ger att den styr kommunikation och hur FPGA skall fungera. Det protokoll som finns hur kommandon skall skickas emellan kretsarna använder en databuss på 12 bitar, en adressbuss med 6 bitar samt en bit för om FPGA skall läsa eller skriva samt en för att visa vilken krets (det har tidigare funnits två FPGA) som skall utföra kommandot. Anledningen till att denna finns kvar är att ge möjligheten att koppla till andra kretsar. Vad det blir är vid detta tillfälle endast spekulationer men ett antagande är GPS, GSM eller annan utrustning för att kunna fjärrstyra gräsklipparen eller följa den via Internet eller att uppgradera befintlig programvara i AVR, som kan omprogrammeras i kretsen under förutsättning att pinnarna GND, !RESET, SCK, MOSI, MISO finns att tillgå, hur detta går till finns beskrivet i kretsens datablad.

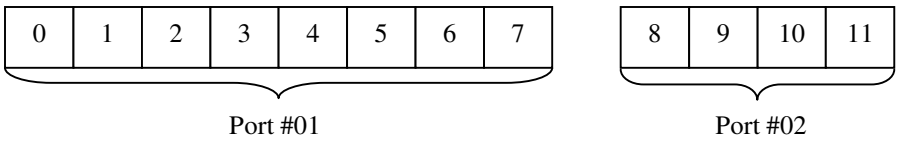
I appendix C redovisas de kommandon som kan skickas till FPGA, nedan visas ett exempel i pseudokod för hur avståndet till en fyren hämtas in till AVR:

1. Kommando läggs ut på adressbuss (enadress = rw = 0)
2. Vänta
3. Sätt enadress = ”1” och rw = ”0” för att FPGA skall lägga ut data på intern databuss
4. Vänta
5. Sätt enadress = ”1” och rw = ”1” för att FPGA skall lägga ut data på databussen
6. Vänta
7. Läs av databuss

Psuedokoden ovan förutsätter att det tidigare i programflödet skickats kommando till FPGA att mäta in avstånd till tre fyrar. Detta görs på följande vis:

1. Kommando för inmätning av avstånd läggs ut på adressbuss.
2. Vänta
3. Sätt enadress = ”1”och rw = ”0”
4. Vänta
5. Lägg ut vilka fyrar som skall ”anropas” av FPGA via databuss
6. Vänta

Det problem som kan uppstå är då databussen läses av eftersom denna är 12 bitar och AVR har portar som är 8 bitar. Men detta kan lösas genom att göra en förhållandevis enkel summation.



Figur 7.7 Databuss sedd från AVR

I figur 6.7 ovan visas hur databussen ses från AVR, de första åtta bitarna kan enkelt läsas av, däremot måste de sista fyra bitarna multipliceras med 256 för att erhålla deras rätta värde eftersom de är de fyra första bitarna representerar de värdena 1,2,4,8 inte 256,512,1024 och 2048 som de gör från FPGA.

I C-kod blir det: `L = PINF+(PINC&0x0F)*256;`

`PINC&0x0F` betyder att bitarna 4-7 maskas bort, det är alltså en vanlig logisk AND-funktion där värdet på bitarna 0-3 är oförändrade medan de sista fyra sätts till noll oavsett deras värde innan operationen.

7.3.5 Testning av mjukvara

För att spara tid och undvika problem bör mjukvaran testas ”bit för bit” det vill säga att funktion för funktion testas var för sig. Anledningen till detta är främst att undvika problem då något inte fungerar. Det kan vara svårt att avgöra var problemet ligger och ändringar kan göras som förvärrar problemet. Andra svårigheter kan uppstå i kommunikationen mellan AVR och FPGA vilket gör att felsökning blir komplicerad.

Av dessa anledningar testas programvaran först i programvaran AVRStudio, därefter då funktionaliteten kunnat verifieras kommer koden att överföras till AVR och där testas med hjälp av ett antal switchar som simulerar databuss och dioder för utport från AVR.

Den enda ändringen som måste göras i programmet är att förlänga vänte-tiden mellan varje steg i kommunikationen detta görs dock enkelt med hjälp av en delayfunktion. Utportar kopplas till lysdioder och inportar till switchar och med denna uppkoppling kan man simulera och verifiera att programmet fungerar som det skall. Först efter detta är det lämpligt att koppla ihop alla delar av hårdvaran och testa att allt fungerar som det skall.

Första steget i testningen är att verifiera att programflödet är korrekt så att då avstånden till fyrarna mäts in av FPGA väntar AVR på att statusvektorns LSB, den minst signifikanta biten, ettställs innan kommando ges för att FPGA skall mata ut längden till första, andra, respektive tredje fyren på databussen.

När detta är vidimerat kommer tre olika fall att testas för att konstatera att all programvara fungerar som den skall, först därefter kommer de olika delarna att kopplas ihop för att testas.

En simulerad uppställning av fyrar har testats i tre olika positioner för roboten, se Appendix D för testdata.

8 Resultat

Fungerande funktioner för att avkoda tangentbord, initiera och skriva ut på display finns, dock ej den som var tänkt från början. Den display som används räcker dock för att mata ut den information som behövs för att fullfölja detta examensarbete. En algoritm för positionering finns och fungerar samt ett adress- och databussprotokoll för att läsa in längder vid positionering är skrivet.

Tyvärr har det inte blivit tester med robot och fyrar men trots detta kan det visas att den programvara som tagits fram kommer att fungera.

9 Diskussion

Den programvara som tagits fram fungerar enligt målsättning och har i tester visat sig fungera väl. Det har inte lagts ner tid på att optimera koden utan funktionen har varit det viktigaste.

Det enda som inte fallit ut enligt målbeskrivningen är den display av märket Seiko som från början skulle användas. Detta beror på att det inte finns obegränsat med tid och omprioriteringar gjordes för att kunna fortsätta och avsluta examensarbetet på ett för övrigt lyckat sätt.

Då det på marknaden inte för tillfället finns en produkt med specifikationer som IQ-Mower kan det konstateras att den fyller en delvis ny nisch av intelligenta fordon för konsumenter. Det enda som krävs är att användaren ”lär” IQ-mower var den skall och inte skall klippa. Möjligheten att placera ut flera fyrar och att lagra data för ett antal gräsmattor gör att IQ-mower kommer att vara betydligt mer flexibel än de produkter som nu finns på marknaden som alla behöver slingor för att markera var den får klippa och inte.

En annan fördel med en gräsklippare som navigerar med hjälp av aktiva fyrar eller liknande utrustning är att den kan klippa betydligt längre innan den måste ta sig tillbaka till laddningsstationen. Detta beror på att den endast behöver gå den kortaste vägen från positionen då laddningsnivån i batteriet blir för låg. Då stängs klippmotorerna av och gräsklipparen åker tillbaka till laddningsstationen för att ladda upp batteriet. De andra gräsklipparna som använder slingor måste ta sig till en slinga och därifrån följa den tills den hittar laddningsstationen. Den kan alltså behöva färdas ganska länge innan den hittar laddningsstationen och detta påverkar naturligtvis hur lång tid den kan klippa mellan två laddningar.

Om eller när denna produkt sätts i serieproduktion kan det vara lämpligt att designa elektronikenhet som är väl skyddad för omgivningen i form av fukt och gräs som kan skada känslig elektronik. Denna enhet bör konstrueras på ett sådant sätt att den är lätt att uppgradera eller byta ut.

10 Slutsatser

Den övergripande slutsatsen blir att examensarbetet har uppfyllt målen och skall därför betraktas som lyckat.

Det bör finnas i åtanke att det i framtiden kan bli så att elektronikenheten utökas med ytterligare enheter som utökar användarvänligheten, kanske styra den via ett webbaserat gränssnitt, uppgradera programvaran eller följa gräsklipparen via videokamera.

11 Rekommendationer till fortsatt arbete

Ett första steg i ett fortsätt arbete kan vara att bygga en bas för all elektronik så att denna utgör en bas för gräsklipparen och därefter kan det finnas möjlighet att konstruera programvara för att använda någon form av intelligent planering av hur en godtycklig gräsmatta klipps. Med detta följer ett antal intressanta problem som:

- Hur lagras data för klippt, respektive oklippt yta?
- Vilket mönster skall gräsmattan klippas i? Spiralmönster? Utifrån och in? Inifrån och ut?
- Skall gräsmattan delas in i kända former och vilken/vilka former skall användas? Kvadrater? Rektanglar? Polygoner? Trianglar?
- Hur skall den behandla rabatter med mera. Köra runt? Något annat mönster?
- Vilken form av intelligens skall användas. Fuzzy logic? Neurala nätverk? Egentillverkad metod?

Källförteckning

1 Borenstein, J (1997). *Mobile Robot Positioning: Sensors and Techniques*, Journal of Robotic Systems.

2 Strangeway , Robert J. (2001). *Fluxgate Magnetometer for NASA/GSFC ST5 Mission* [Elektronisk]. Tillgänglig: <http://www-ssc.igpp.ucla.edu/st5/design.html>

3 Mäkelä, Hannu, (2001). *Outdoor navigation of mobile robots* [Elektronisk]. Tillgänglig: <http://lib.hut.fi/Diss/2001/isbn9512259117/isbn9512259117.pdf>

4 Suksakulchai, S, Thongchai, S. (2000) *Mobile Robot Localization using an Electronic Compass for Corridor Environment* [Elektronisk]. Tillgänglig: <http://eecs.vanderbilt.edu/CIS/PAPERS/2000/Ss_SMC00.pdf>

5 Nefti, S., Oussalah, M (2000). *Intelligent Adaptive Mobile Robot Navigation*

6 *http://www.ngs.noaa.gov/CORS/CorsPP/WA-SlideShow/sld012.htm*

7 Karlsson, M. (2004). [Ej publicerad] *Positioneringsalgoritm för intelligent fordon*, HTU

8 DeSouza, G, Kak, A (2002). *Vision for Mobile Robot Navigation: A Survey.* IEEE Transactions on pattern analysis and machine intelligence, vol. 24, no. 2

9 Liang Zhao, (2001). *Dressed Human Modeling, Detection, and Parts Localization.* [Elektronisk]. The Robotics Institute, Carnegie Mellon University Tillgänglig: < http://www.ri.cmu.edu/pubs/pub_3767.html >

10 Tillgänglig på: http://mpfwww.jpl.nasa.gov/MPF/index1.html. Kirk Goodall kirk.goodall@jpl.nasa.gov, Mars Web Engineer

11 Datablad, (2004). *8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash* [Elektronisk]. ATMEL®, Tillgänglig: <http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf >

12 Datablad (2001). *MM74C992 16-key encoder, MM74C923 20-key encoder*Fairchild Semiconductor, Tillgänglig: <http://www.fairchildsemi.com/pf/MM/MM74C922.html>

Övriga datablad:

Samsung KS0066: URL: http://www.data-modul.de/de/service/download/nonedit/download.html?kategorie=Display%20Technology%20Download

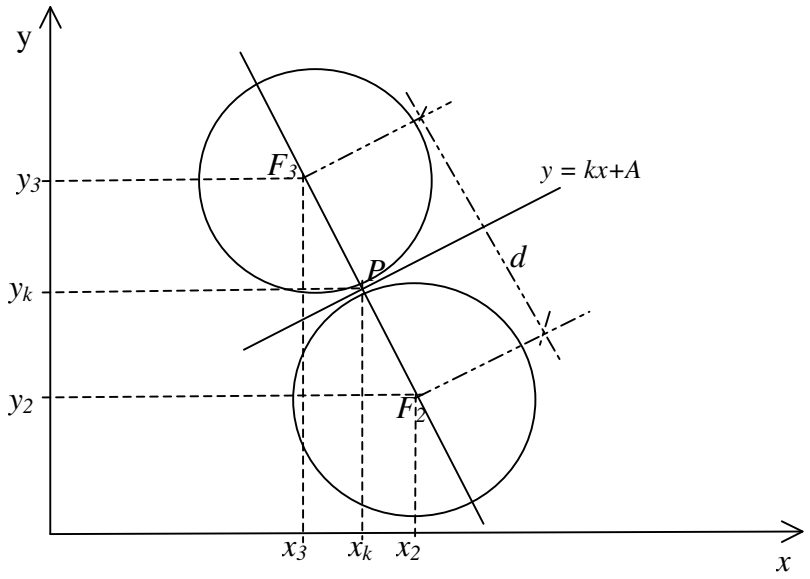
Seiko LCD-display: URL: http://www.elfa.se/pdf/75/07554991.pdf

Hitachi HD44780, < http://www.eio.com/hd44780.pdf>

A Utökning av algoritmen: Räta linjer

Vid vissa positioner kan det inträffa att något cirkelpar ej skär varandra, se figur A.1 nedan. Vid dessa tillfällen är det önskvärt att använda en alternativ beräkning av skärningslinjen. Det enda antagandet som gjorts är att den verkliga skärningslinjen ligger mitt i gapet mellan cirklarna, det vill säga båda längderna har reducerats i längd av samma storlek och tecken på bruset.

Längderna är alltså avståndet från gräsklipparen till respektive fyr, radien för cirklarna.



Figur A.1 Skärningslinje då cirklar ej skär varandra

$$d = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} \tag{A.1}$$

Avståndet F_2P betecknas hädanefter som L_{2+}

$$L_{2+} = L_2 + \frac{d - (L_2 + L_3)}{2} \tag{A.2}$$

Betrakta trianglarna $F_3F_2(x_3y_2)$ och $PF_2(x_ky_2)$ i figur A.1 ovan. Dessa är likformiga vilket ger följande likhet:

$$\frac{d}{y_3 - y_2} = \frac{L_{2+}}{y_k - y_2} \tag{A.3}$$

y_k löses ur A.3 varefter x_k beräknas:

$$y_k = y_{2+} \frac{L_{2+}(y_3 - y_2)}{d} \tag{A.4}$$

$$x_k = x_2 - \sqrt{L_{2+}{}^2 - (y_k - y_2)^2} \tag{A.5}$$

$$k_{23} = -\frac{x_3 - x_2}{y_3 - y_2} \tag{A.6}$$

$$A_{23} = y_k - k_{23} x_k \tag{A.7}$$

Skärningslinjen för de två övriga beräknas på samma sätt som redovisats ovan varför dessa beräkningar inte tas upp, utan endast resultatet redovisas nedan.

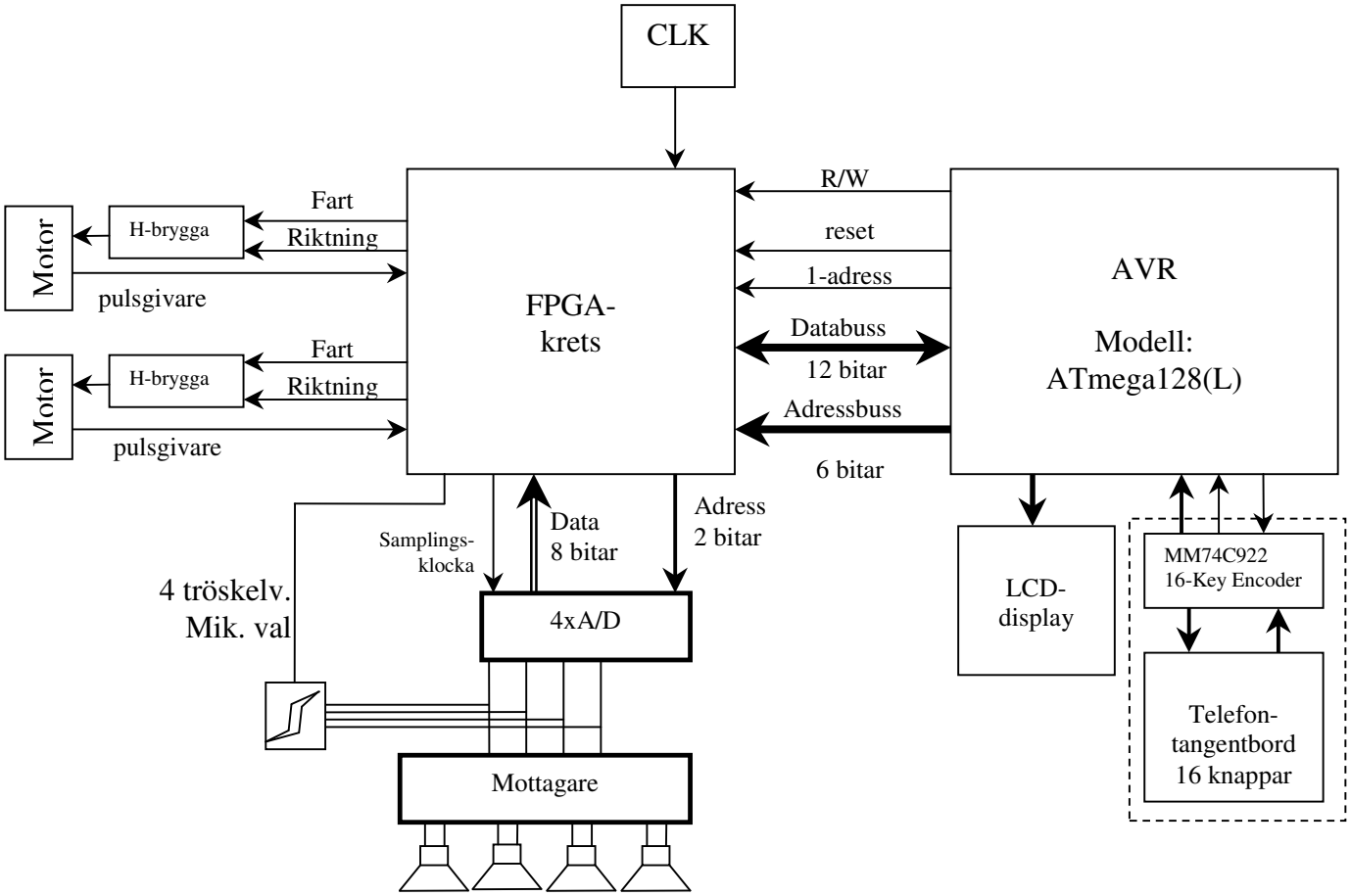
$$k_{12} = -\frac{x_2 - x_1}{y_2 - y_1} \tag{A.8}$$

$$A_{12} = y_k - k_{12}(x_1 + L_{1+}(x_2 - x_1)) \tag{A.9}$$

$$k_{13} = -\frac{x_3 - x_1}{y_3 - y_1} \tag{A.10}$$

$$A_{13} = y_k - k_{13} x_k \tag{A.11}$$

B Blockschema för hårdvaran



Figur B.1 Blockschema för hårdvaran

I figur B.1 ovan visas en schematisk bild över hur hårdvaran kopplas ihop. För en tydligare bild av hur LCD-display och tangentbord kopplas in hänvisas till kapitlen 5.1.1 och 5.1.2.

C Flödesschema och programkod

Nedan redovisas en sammanställning av kommandon som används i kommunikationen mellan AVR och FPGA. Variabeln *enadress* anger vilken krets som adresseras, då det för tillfället endast finns en FPGA krets är denna alltid ”1” men den finns med för möjlighet att bygga ut systemet med ytterligare funktioner i framtiden. Slutligen variabeln *rw* (=read/write) helt enkelt talar om för FPGA om den skall läsa av adressbussen eller skriva till databussen.

Tabell C.1 Kommandon för kommunikationen AVR-FPGA

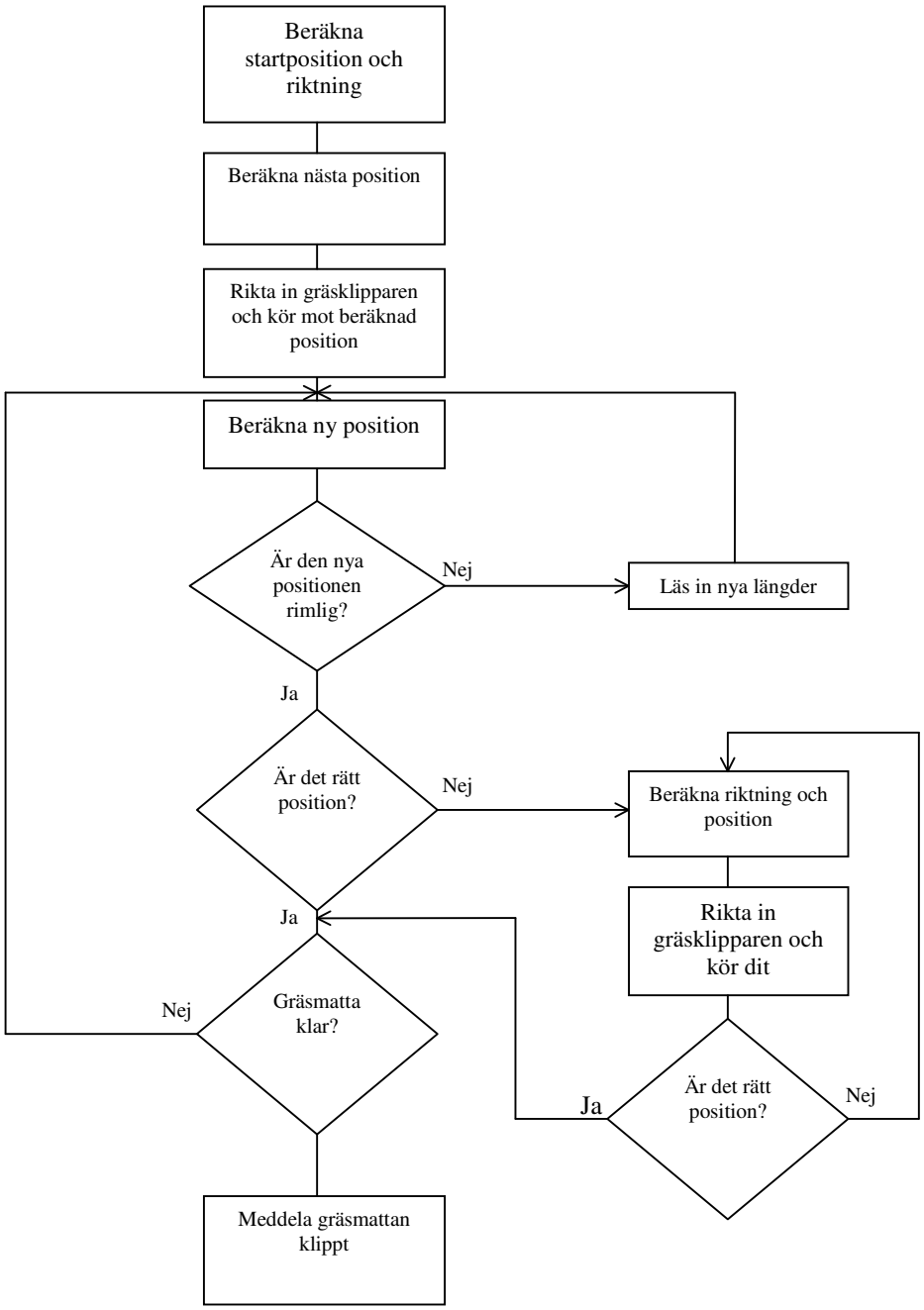
Kommando på adressbuss Då *enadress* =”1” och *rw*=”0”	Beskrivning
011111	Nollställning av motorer
000000	Tvärstanna motorer
000001	Framåt båda motorer
000010	Backa båda motorer
000011	Sväng höger
000100	Sväng vänster
000101	Starta klippmotorer
000110	Stoppa klippmotorer
110001	Längd 1 läggs ut på intern databuss
110010	Längd 2 läggs ut på intern databuss
110011	Längd 3 läggs ut på intern databuss
111111	Tillståndsvektor läggs ut på intern databuss
100001	Inmätning av fyrar, FPGA läser av databuss vilka fyrar som gäller
enadress=”1” och *rw*=”1”	
110001	Längd 1 läggs ut på databuss
110010	Längd 2 läggs ut på databuss
110011	Längd 3 läggs ut på databuss
111111	Tillståndsvektor läggs ut på databuss

Förutom de kommandon som redovisas i tabell C.1 ovan finns ett för att skicka information om vilka fyrar som skall adresseras det görs på följande sätt:

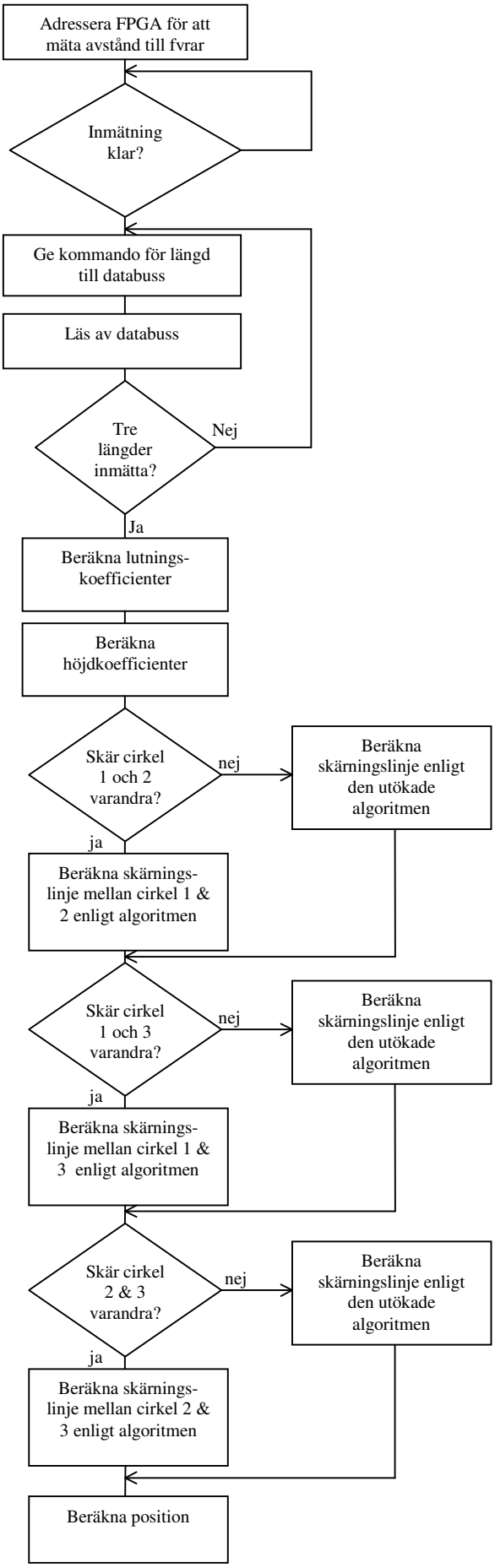
1. Sätt adressbuss till 100001 och enadress och r/w till ”0”
2. Vänta
3. Sätt databuss till utport
4. Vänta
5. Sätt enadress till ”1”
6. Vänta
7. Lägg ut fyrkod på databuss
8. Vänta
9. Sätt databus till inport

Därefter kommer FPGA att adressera angivna fyrar och invänta svar för att beräkna avståndet till fyren. Om en fyr inte svarar läggs det ut 12 ettor på databussen då denna fyrs längd läses in från AVR detta ger det hexadecimala värdet 0xFFF alltså ett tecken på att vidta en åtgärd som att på nytt anropa samma fyrar

I figur C.1 nedan visas ett flödesschema för hur det skulle kunna gå till för att klippa en gräsmatta. Denna figur är som synes övergripande och varje block innehåller varierande mängder steg men om alla tas med ger det ingen överblick utan blir svårtolkat.



Figur C.1 Flödesschema för styrning och positionering



Figur C.2 Flödesschema för inläsning och positionsberäkning

I figur C.2 ovan redovisas endast inläsning och positionering, den del som detta examensarbete behandlar. Från sidan C:6 och framåt redovisas den kod som skrivits för att kommunicera med FPGA och positionera roboten samt skicka information till användaren via en LCD-display. Koden för att avkoda tangentbordet finns med men används dock inte då denna del av programvaran egentligen inte behöver kommunicera med användaren annat än att visa positionen och avstånd till fyrar.
I det läge då denna programvara används skall eller kan användaren egentligen inte påverka resultatet eftersom fyrarnas position är fast och gräsklipparen håller på att klippa. Informationen som skickas till LCD-displayen är snarare något som kan liknas vid en statusrapport.
Nedan redovisas de funktioner som används för att initiera LCD-displayen som används på HTU. De kommandon som används, t.ex: `PORTD = 0x38;` hämtas från styrkretsens datablad.

```
void LCD_ini(void)
 {
 delay(LCD_Start);
 PORTD = 0x38;     //8-bit mod, 2 rader, 5x8 punkters.
 E_Klocka();

 PORTD = 0x0F;     //Display on/off, blinkande markör på.
 E_Klocka();
 PORTD = 0x06;     //Flyttar markör till höger efter utskrift.
 E_Klocka();
 PORTD = 0x01;     //Rensar displayen.
 E_Klocka();
 }

void E_Klocka(void)
 {
 //Används för att skicka kommandon till LCD:
 delay(E_Clock);
 PORTA=0x4;
 delay(E_Clock);
 PORTA=0x0;
 }
```

Funktionen nedan används då åtta tecken skrivits ut på displayen som fungerar som två stycken 1x8 teckens displayer.

```
void Rad2(void)
 {
 //Gör det möjligt att skriva ut på rad två:
 PORTC=0xC0;
 E_Klocka();
 }
```

Nedan redovisas fullständig programkod för tangentbord, LCD-display, Adress- och Databussprotokoll.

```
/*****************************************************************
Chip:           ATmega128L
Datum:          2004-05-20
Skrivit av:     Micael Karlsson
Funktion:
Kommunicerar med FPGA-krets för att mäta in längder till tre fyrar.
Därefter görs en positionsbestämmning och skriver ut positionen. Om
någon av fyrarna ej svarar får den längden värdet 4095 och en
varning
skrivs ut. Inmätningen fortsätter dock men i Position() testas om
något
av längderna är just 4095 och då avbryts/stoppas positioneringen och
värdet "No position" skrivs ut.
---------------------------------
------- Portkonfiguration -------
---------------------------------
Port A Kontroll-bitar LCD
Port D Databitar LCD DB7-DB0
Port C Tangentbord    (ej inkopplat i detta program)
Port B Adressbuss FPGA      (PORTB.0 = enadress,  PORTB.1 = rw,
PORTB.2 = AD0,
                                  PORTB.3 = AD1,       PORTB.4 =
AD2, PORTB.5 = AD3,
                                  PORTB.6 = AD4,       PORTB.7 =
AD5.)
Port C Databuss FPGA  (DA11-D4)
Port F Databuss FPGA  (DA3-DA0)
*****************************************************************/

#include <math.h>
#include <avr/io.h>
#include <stdlib.h>
#include <inttypes.h>


#define LCD_Start 10000          //Delay innan LCD initieras.
#define E_Clock 1000        //Fördröjning för kommando och data till
LCD.
#define AdressDelay 2       //Fördröjning för inläsning av längder.
#define StatVectorDelay 3   //Fördröjning för statusvektor.
#define TextDelay 5              //Delaytid för textutmatning.

//För kommandot dtostre();
//Om resultat i print_float() önskas på formen "[-]d.ddde177dd",
//istället för formen "[-]d.ddd" som nu är resultatet
#define   DTOSTR_ALWAYS_SIGN 0x01       //Ger plus/minustecken
#define   DTOSTR_UPPERCASE 0x04         //Ger E istället för e


#define statevector_0 0xFC  //statevector -> intern1.
#define statevector_1 0xFF  //statevector -> databuss.

//Kommandon som läggs ut på adressbussen då
//längder till tre olika fyrar skall läsas in:
#define dist1_0 0xC4        //Kommando för längd 1
#define dist1_1 0xC5        //längd 1 -> intern1.
```

```
#define dist1_2 0xC7        //längd 1 -> databuss.
#define dist2_0 0xC8        //Kommando för längd 2
#define dist2_1 0xC9        //längd 2 -> intern1.
#define dist2_2 0xCB        //längd 2 -> databuss.
#define dist3_0 0xCC        //Kommando för längd 3
#define dist3_1 0xCD        //längd 3 -> intern1.
#define dist3_2 0xCF        //längd 3 -> databuss.

//Funktionsdeklarationer:
//Delayfunktioner:
void delay(int);
void delay_1ms(void);
void delay_10ms(void);
void delay_100ms(void);
void delay_1s(void);

void AVR_ini(void);               //Initierar ATmega128L.
void LCD_ini(void);               //Initierar HTUs LCD.
void E_Klocka(void);              //Används för att skicka kommando
till LCD.
void E_RS_klocka(void);           //Används för att skicka data till
LCD.
void Clear_Display(void);         //Rensar LCD.
void Rad2(void);                        //Skriver ut på andra raden
(tecken 9-16).
void keyboard(void);              //Avkodar tangentbord.
void Distans(void);               //Mäter in längder.
void Position(void);              //Beräknar positionen.
void print_text(int);             //Skriver ut text.
void print_int(int);              //Skriver ut heltal.
void print_float(float,int,int,int);    //Skriver ut flyttal.
int StateVector(int);             //Läser av statusvektorn
void send_beacons(int,int,int);   //Skickar info. om vilka fyrar som
                                  //skall adresseras av FPGA.
//-------------------------------------------------------------------
//------------------------ Globala variabler ----------------------
float L_1=123.45,L_2=123.45,L_3=123.45;
//-------------------------------------------------------------------
//-------------------------------------------------------------------
void main(void)
    {
    AVR_ini();
    LCD_ini();
    print_text(1);
    delay(TextDelay);
    Clear_Display();
    Position();
    while(1)
          {
          }
    }
//-------------------------------------------------------------------
void AVR_ini(void)
    {
    //Initiering av ATmega128L
    //--------------------------
    // Port A Kontroll-bitar LCD:
    DDRA=0xFF;  //skall vara 0xFF
    PORTA=0x00;
```

```
// Port D Databitar LCD DB7-DB0:
DDRD=0xFF;  //skall vara 0xFF
PORTD=0x00;

// Port C Tangentbord:
DDRC=0x00;//DDRC=0x20;
PORTC=0x00;

// Port B   //Adressbuss: AVR - FPGA
DDRB=0xFF;
PORTB=0x00;


// Port E //Databuss: FPGA 0-7
PORTE=0x00;
DDRE=0x00;

// Port F   //Databuss: FPGA 8-11
DDRF=0x00;
PORTF=0x00;

// Port G   //Används ej!
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0
ASSR=0x00;
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter 2
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
```

```
    OCR3AL=0x00;
    OCR3BH=0x00;
    OCR3BL=0x00;
    OCR3CH=0x00;
    OCR3CL=0x00;

    // External Interrupt(s)
    EICRA=0x00;
    EICRB=0x00;
    EIMSK=0x00;

    // Timer(s)/Counter(s) Interrupt(s)
    TIMSK=0x00;
    ETIMSK=0x00;
    //Watchdog:
    WDTCR=0x00;
    WDTCR=0x00;
    /* Write logical one to WDCE and WDE */
    WDTCR = (1<<WDCE) | (1<<WDE);
    /* Turn off WDT */
    WDTCR = 0x00;
    // Analog Comparator
    SFIOR=0x00;
    ACSR=0x80;
    }
//-----------------------------------------------------------------
void LCD_ini(void)
    {
    //Initiering av LCD-display logisk:2x8, fysiskt 1x16:
    delay(2);
    PORTD = 0x38;     //8-bit mod, 2 rader, 5x8 punkters.
    E_Klocka();
    PORTD = 0x0F;     //Display on/off, blinkande markör på.
    E_Klocka();
    PORTD = 0x06;     //Flyttar markör åt höger efter utskrift.
    E_Klocka();
    PORTD = 0x01;     //Rensar displayen.
    E_Klocka();
    delay(3);
    }
//-----------------------------------------------------------------
void delay_1ms(void)
    {
    //Fördröjning på 1 ms vid frekvensen 3,59 MHz:
    int ii,steps=18500;
    for(ii=0;ii<steps;ii=ii+1){}
    }
//-----------------------------------------------------------------
void delay_10ms(void)
    {
    //Fördröjning på 10 ms vid frekvensen 3,59 MHz:
    int ii,steps=10;
    for(ii=0;ii<steps;ii=ii+1)
          {
          delay_1ms();
          }
    }
//-----------------------------------------------------------------
void delay_100ms(void)
```

```
    {
    //Fördröjning på 100 ms vid frekvensen 3,59 MHz:
    int ii,steps=100;
    for(ii=0;ii<steps;ii=ii+1)
          {
          delay_1ms();
          }
    }
//---------------------------------------------------------------
void delay_1s(void)
    {
    //Fördröjning på 1 s vid frekvensen 3,59 MHz:
    int ii,steps=1000;
    for(ii=0;ii<steps;ii=ii+1)
          {
          delay_1ms();
          }
    }
//---------------------------------------------------------------
void delay(int val)
    {
    //Inparametern "val" anger fördröjningens längd:
    if(val==1)
          {delay_1ms();}
    if(val==2)
          {delay_10ms();}
    if(val==3)
          {delay_100ms();}
    if(val==4)
          {delay_1s();}
    if(val==5)
          {delay_1s();delay_1s();}
    }
//---------------------------------------------------------------
void E_Klocka(void)
    {
    //Används för att skicka kommandon till LCD:
    delay_1ms();
    PORTA=0x4;        //E=1, RS=0, RW=0.
    delay_1ms();
    PORTA=0x0;        //E=0, RS=0, RW=0.
    }
//---------------------------------------------------------------
void E_RS_klocka(void)
    {
    //Används för att skriva ut tecken i LCD:
    delay_1ms();
    PORTA=0x5;        //E=1, RS=0, RW=1.
    delay_1ms();
    PORTA=0x0;        //E=0, RS=0, RW=0.
    delay(2);
    }
//---------------------------------------------------------------
void Clear_Display(void)
    {
    //Rensar LCD och återgår till position 0:
    PORTD=0x01;
    E_Klocka();
    delay(2);
```

```
    }
//---------------------------------------------------------------------
void Rad2(void)
    {
    //Gör det möjligt att skriva ut på rad två:
    PORTD=0xC0;
    E_Klocka();
    delay(2);
    }
//---------------------------------------------------------------------
void print_text(int textnr)
    {
    char text1[] = "** IQ-Mower **";
    char text2[]="New Lawn(1=N,2=Y)?";
    char text3[]="Nr. of beacons?";
    char text4[]="X= ";
    char text5[]="Y= ";
    char text6[]="L1= ";
    char text7[]="L2= ";
    char text8[]="L3= ";
    char text9[]="Position:";
    char text10[]="(";
    char text11[]=")";
    char text12[]=":";
    char text13[]=" Get L1:";
    char text14[]=" Get L2:";
    char text15[]=" Get L3:";
    char text16[]=" Get State";
    char text17[]=" Wrong value!";
    char text18[]="No position!";
    char text19[]=" ";
    char text20[]=" A";
    char text21[]=" B";
    char text22[]=" C";
    char text23[]=" D";
    char text24[]=" 0";
    char text25[]=" 1";
    char text26[]=" 2";
    char text27[]=" 3";
    char text28[]=" 4";
    char text29[]=" 5";
    char text30[]=" 6";
    char text31[]=" 7";
    char text32[]=" 8";
    char text33[]=" 9";
    char text34[]=" #";
    char text35[]=" *";
    char text36[]=" .";

    long temp=0, length;
    char *textin;
    if (textnr == 1)
          {length = sizeof(text1);
          textin = &text1[0];}
    else if (textnr == 2)
          {length = sizeof(text2);
          textin = &text2[0];}
    else if (textnr == 3)
          {length = sizeof(text3);
```

```
      textin = &text3[0];}
else if (textnr == 4)
      {length = sizeof(text4);
      textin = &text4[0];}
else if (textnr == 5)
      {length = sizeof(text5);
      textin = &text5[0];}
else if (textnr == 6)
      {length = sizeof(text6);
      textin = &text6[0];}
else if (textnr == 7)
      {length = sizeof(text7);
      textin = &text7[0];}
else if (textnr == 8)
      {length = sizeof(text8);
      textin = &text8[0];}
else if (textnr == 9)
      {length = sizeof(text9);
      textin = &text9[0];}
else if (textnr == 10)
      {length = sizeof(text10);
      textin = &text10[0];}
else if (textnr == 11)
      {length = sizeof(text11);
      textin = &text11[0];}
else if (textnr == 12)
      {length = sizeof(text12);
      textin = &text12[0];}
else if (textnr == 13)
      {length = sizeof(text13);
      textin = &text13[0];}
else if (textnr == 14)
      {length = sizeof(text14);
      textin = &text14[0];}
else if (textnr == 15)
      {length = sizeof(text15);
      textin = &text15[0];}
else if (textnr == 16)
      {length = sizeof(text16);
      textin = &text16[0];}
else if (textnr == 17)
      {length = sizeof(text17);
      textin = &text17[0];}
else if (textnr == 18)
      {length = sizeof(text18);
      textin = &text18[0];}
else if (textnr == 19)
      {length = sizeof(text19);
      textin = &text19[0];}
else if (textnr == 20)
      {length = sizeof(text20);
      textin = &text20[0];}
else if (textnr == 21)
      {length = sizeof(text21);
      textin = &text21[0];}
else if (textnr == 22)
      {length = sizeof(text22);
      textin = &text22[0];}
else if (textnr == 23)
```

```
          {length = sizeof(text23);
          textin = &text23[0];}
    else if (textnr == 24)
          {length = sizeof(text24);
          textin = &text24[0];}
    else if (textnr == 25)
          {length = sizeof(text25);
          textin = &text25[0];}
    else if (textnr == 26)
          {length = sizeof(text26);
          textin = &text26[0];}
    else if (textnr == 27)
          {length = sizeof(text27);
          textin = &text27[0];}
    else if (textnr == 28)
          {length = sizeof(text28);
          textin = &text28[0];}
    else if (textnr == 29)
          {length = sizeof(text29);
          textin = &text29[0];}
    else if (textnr == 30)
          {length = sizeof(text30);
          textin = &text30[0];}
    else if (textnr == 31)
          {length = sizeof(text31);
          textin = &text31[0];}
    else if (textnr == 32)
          {length = sizeof(text32);
          textin = &text32[0];}
    else if (textnr == 33)
          {length = sizeof(text33);
          textin = &text33[0];}
    else if (textnr == 34)
          {length = sizeof(text34);
          textin = &text34[0];}
    else if (textnr == 35)
          {length = sizeof(text35);
          textin = &text35[0];}
    else if (textnr == 36)
          {length = sizeof(text36);
          textin = &text36[0];}
          else
          ;

    for(temp=0;temp<length-1;temp=temp+1)
          {
          PORTD=*(textin+temp);
          E_RS_klocka();
          if(temp==7)
                {Rad2();}
          }
    }
//---------------------------------------------------------------------
void print_float(float tal,int NrOfDigits,int NrOfDecimals,int
Disp2)
    {
    //Skriver ut ett flyttal på LCD-display:
    char tal1[10];
    long length,temp;
```

```
    char *textin;
    //För formatet: "[-]d.ddde177dd":
    //dtostre(tal,tal1,DTOSTR_ALWAYS_SIGN,DTOSTR_UPPERCASE);
    //För formatet: "[-]d.ddd":
    dtostrf(tal,NrOfDigits,NrOfDecimals,tal1);
    textin = &tal1[0];
    length = sizeof(tal1);
    for(temp=0;temp<NrOfDigits;temp=temp+1)
          {
          PORTD=*(textin+temp);
          E_RS_klocka();
          if(temp==(7-Disp2)){Rad2();}
          }
    }
//---------------------------------------------------------------------
void print_int(int tal)
    {
    //Skriver ut ett heltal på LCD-display:
    char tal1[5];
    long length,temp;
    char *textin;

    itoa(tal,tal1,10);
    textin = &tal1[0];
    length = sizeof(tal1);
    for(temp=0;temp<length-1;temp=temp+1)
          {
          PORTD=*(textin+temp);
          E_RS_klocka();
          }
    }
//---------------------------------------------------------------------
void Position(void)
    {
    //Beräknar positionen med hjälp av metoden "Räta linjer":
    double k12=1.23,k13=1.24,k23=1.25;  //Lutnings-
    double A12=1.23,A13=1.23,A23=1.23;  //och höjdkoefficienter.
                                        //för skärningslinjer.
    float x1,x2,x3,y1,y2,y3,L1,L2,L3;   //x- och y-koordinat samt
                                        //längder
                                        //till resp. fyr.
    float Lp,xa,ya;                     //Variabel om cirklar ej
                                        //skär varann.
    double d12,d13,d23;                 //Avstånd mellan fyrar.
    float X,Y;                          //Aktuell position.
    int ii=0;
    Distans();
    x1 = 1;
    y1 = 1;
    x2 = 21;
    y2 = 2;
    x3 = 1;
    y3 = 15.5;
    if((L_1==4095)|(L_2==4095)|(L_2==4095))
          {
          Clear_Display();
          print_text(18);
          delay(TextDelay);
          }
```

```
else
      {
      L1 = L_1/100;
      L2 = L_2/100;
      L3 = L_3/100;
      //Lutningskoefficienter:
      k12 =  -(x2-x1)/(y2-y1);
      k13 =  -(x3-x1)/(y3-y1);
      k23 =  -(x3-x2)/(y3-y2);
      //Avstånd mellan fyrar:
      d12 = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
      d13 = sqrt((x1-x3)*(x1-x3)+(y1-y3)*(y1-y3));
      d23 = sqrt((x2-x3)*(x2-x3)+(y2-y3)*(y2-y3));

      if(y1==y2)
            {//Nödvändigt för att algoritmen ej skall haverera:
            y1 = y1-0.05;
            y2 = y2+0.05;
            }
      if ((L1+L2) >= d12)
      {//Om cirklarna för fyr 1 & 2 skär varandra:
      A12 = (L1*L1-L2*L2-x1*x1+x2*x2-y1*y1+y2*y2)/(2*(y2-y1));
      }
      if((L1+L2) < d12)
            {//Om cirklarna för fyr 1 & 2 ej skär varandra:
            Lp = L1+(d12-(L1+L2))/2;
            xa  = x1+Lp*(x2-x1);
            ya = sqrt(Lp*Lp-((xa-x1)*(xa-x1)));
            A12 = ya-k12*xa;
            }
      if ((L1+L3) >= d13)
      {//Om cirklarna för fyr 1 & 3 skär varandra:
      A13 = (L1*L1-L3*L3-x1*x1+x3*x3-y1*y1+y3*y3)/(2*(y3-y1));
      }
      if((L1+L3) < d13)
            {//Om cirklarna för fyr 1 & 3 skär varandra:
            Lp  = L1+(d13-(L1+L3))/2;
            ya  = y1+Lp*(y3-y1)/d13;
            xa  = x1+sqrt(Lp*Lp-((ya-y1)*(ya-y1)));
            A13 = ya-k13*xa;
            }
      if ((L2+L3) >= d23)
      {//Om cirklarna för fyr 2 & 3 skär varandra:
      A23=(L2*L2-L3*L3-x2*x2+x3*x3-y2*y2+y3*y3)/(2*(y3-y2));
      }
      if((L2+L3) < d23)
            {//Om cirklarna för fyr 2 & 3 ej skär varandra:
            Lp  = L2+(d23-(L2+L3))/2;
            ya  = y2+Lp*(y3-y2)/d23;
            xa  = x2+sqrt(Lp*Lp-((ya-y2)*(ya-y2)));
            A23 = ya-k23*xa;
            }
            //Beräknar positionen:
            X = (A13-A12)/(k12-k13);
            Y = k23*X+A23;
            //Skriver ut positionen:
            Clear_Display();
            print_text(9);
            delay(TextDelay);
```

```
          Clear_Display();
          for(ii=0;ii<4;ii=ii+1)
                {
                //skriver ut x- och y-koordinaterna 4 ggr:
                print_text(4);
                print_float(X,6,4,3);
                delay(TextDelay);
                Clear_Display();//LCD_ini();
                print_text(5);
                print_float(Y,6,4,3);
                delay(TextDelay);
                Clear_Display();//LCD_ini();
                }
          }
    }
//---------------------------------------------------------------------
void send_beacons(int fyr1,int fyr2,int fyr3)
    {
    //Adressera FPGA för att mäta längderna till tre fyrar
    //som anges som siffervärden i funktionsanropet:
    PORTB=0x84;
    DDRF=0xFF;
    DDRC=0x0F;
    delay(AdressDelay);
    PORTB=0x85;
    delay(AdressDelay);
    PORTF=fyr1+fyr2*16;
    PORTC=fyr3;
    delay(AdressDelay);
    DDRF=0x00;
    DDRC=0x00;
    delay(AdressDelay);
    }
//---------------------------------------------------------------------
void Distans(void)
    {
    //Läser in tre längder från FPGA och skriver ut värdet på
    //displayen:
    print_text(16);
    while(StateVector(0x1)==0)
          {delay(StatVectorDelay);}
    //Läsa in längd #01:
    Clear_Display();
    print_text(13);
    PORTB = dist1_0;
    delay(AdressDelay);
    PORTB = dist1_1;
    delay(AdressDelay);
    PORTB = dist1_2;
    delay(AdressDelay);
    L_1 = (PINF+(PINC&0x0F)*256); //Läsa in längd 1.
    if(L_1==4095)
          {
          //Detta betyder att fyren ej gett svar:
          Clear_Display();
          print_text(17);
          delay(TextDelay);
          }
    Clear_Display();
```

```
    print_text(6);
    print_int(L_1);
    delay(TextDelay);
    //Läsa in längd #02:
    Clear_Display();
    print_text(14);
    PORTB = dist2_0;
    delay(TextDelay);
    PORTB = dist2_1;
    delay(TextDelay);
    PORTB = dist2_2;
    delay(TextDelay);
    L_2 = (PINF+(PINC&0x0F)*256); //Läsa in längd 2.
    if(L_2==4095)
    {
    //Detta betyder att fyren ej gett svar:
    Clear_Display();
    print_text(17);
    delay(TextDelay);
    }
    Clear_Display();
    print_text(7);
    print_int(L_2);
    delay(TextDelay);
    //Läsa in längd #03:
    Clear_Display();
    print_text(15);
    PORTB = dist3_0;
    delay(AdressDelay);
    PORTB = dist3_1;
    delay(AdressDelay);
    PORTB = dist3_2;
    delay(AdressDelay);
    L_3 = (PINF+(PINC&0x0F)*256); //Läsa in längd 3.
    if(L_2==4095)
          {
          //Detta betyder att fyren ej gett svar:
          Clear_Display();
          print_text(17);
          delay(TextDelay);
          }
    Clear_Display();
    print_text(8);
    print_int(L_3);
    delay(TextDelay);
    }
//-----------------------------------------------------------------
int StateVector(int tal)
    {
    //Då tillståndsvektorn antar 0x01 betyder
    //det att alla tre längdet är inmätta:

    PORTB = 0xFC;           //statevector.
    delay(AdressDelay);
    PORTB = 0xFD;           //statevector ut på intern1.
    delay(AdressDelay);
    PORTB = 0xFF;           //statevector ut på databuss.
    delay(AdressDelay);
    Clear_Display();
```

```
    if((PINF&0x01) == tal)
          {
          //statevector ==> "längder klara"
          return(1);
          }
    else
          {return(0);}
    }
//-------------------------------------------------------------------
void keyboard(void)
    {
    unsigned char keypress;
    PORTC = PORTC|0x20;
    while((PINC&0x10)==0x0){}
          PORTC = PORTC&0xD0;
          delay(2);
          keypress = PINC&0x0F;
          PORTC = PORTC|0x20;
          if(keypress==0)   //"A"
                {print_text(20);}
          if(keypress==1)   //"3"
                {print_text(27);}
          if(keypress==2)   //"2"
                {print_text(26);}
          if(keypress==3)   //"1"
                {print_text(25);}
          if(keypress==4)   //"B"
                {print_text(21);}
          if(keypress==5)   //"6"
                {print_text(30);}
          if(keypress==6)   //"5"
                {print_text(29);}
          if(keypress==7)   //"4"
                {print_text(28);}
          if(keypress==8)   //"C"
                {print_text(22);}
          if(keypress==9)   //"9"
                {print_text(33);}
          if(keypress==10)  //"8"
                {print_text(32);}
          if(keypress==11)  //"7"
                {print_text(30);}
          if(keypress==12)  //"D"
                {print_text(23);}
          if(keypress==13)  //"#"
                {print_text(34);}
          if(keypress==14)  //"0"
                {print_text(33);}
          if(keypress==15)  //"*"
                {print_text(35);}
          while((PINC&0x10)==0x10){}
    }
```

D Test av programvara

För att testa att programmet fungerade tillfredsställande gjordes nedanstående test som utföll till belåtenhet.
Delmomenten var:

1. Adress- och databussprotokoll
2. Inläsning av data från databuss
3. Positioneringsfunktion
4. Sammankoppling av 1-3

Fyrarnas position är:

Fyr 1: (1,1)
Fyr 2: (21,2)
Fyr 3: (1,15,5)
Positioner beräknades i tre olika fall:

Tabell D.1 Avstånd till fyrar i decimalt och binärt format

Avstånd till fyr:	Fall 1(m)		Fall 2 (m)		Fall 3(m)	
1	$11,3_{10}$	10001101010_2	$9,3_{10}$	1110100010_2	$21,4_{10}$	100001011100_2
2	$9,4_{10}$	1110101100_2	$14,7_{10}$	10110111110_2	$13,6_{10}$	10101010000_2
3	$20,0_{10}$	11111010000_2	$10,3_{10}$	10000000110_2	$16,1_{10}$	11001001010_2

Först beräknades positionen I Matlab där en testfunktion gjordes eftersom möjligheten finns att enkelt rita upp cirklar och skärningslinjer för att på så sätt verifiera att positionen är korrekt.

Nästa steg var att med hjälp av switchar, som nämnts tidigare, simulera FPGA och databussen mellan kretsarna.

Positionsberäkningen stämde överrens i mycket hög grad se tabell D.2 nedan.

Tabell D.2 Jämförelse mellan Matlab och AVR

Fall	Matlab (m)		AVR (m)	
1	x=12,115	y=-1,140	x=12,115	y=-1,110
2	x= 7,456	y= 7.574	x=7,456	y=7,574
3	x=17.144	y= 15.103	x=17,144	y=15,103

Det kan alltså konstateras att överensstämmelsen är mycket god, att positionen stämmer till tre decimaler är bra eftersom det i det verkliga systemet finns ett visst fel redan i andra decimalen, det kan bli centimeterfel i inmätning av avståndet till fyrar. Felet kan bero på störningar och ett visst mätfel som finns med då ett analogt mätvärde omvandlas till binärt. Detta sker dock i FPGA varför noggrannare analys av eventuella felkällor ej diskuteras i detta examensarbete.

Däremot kan det konstateras att då fyrar sätts ut är det endast den första fyrens position som är exakt placerad eftersom denna placeras i origo, de övriga fyrarna kommer att mätas in vilket ger samma mätfel som diskuterats ovan. Hur mycket eller om detta påverkar klippresultatet är en intressant fråga som tyvärr ej får sitt svar i denna rapport.