DigiRuler
Digitala Projekt 2005

Johan Norén E–03
Ted Strandberg E–02

Handledare: Bertil Lindvall

17 maj 2005

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Abstract

In many different situations it may be important to know the width and length of an object. There is today different kinds of methods to measure these parameters with many kinds of sensors and actuators.

This paper describes the development of a size measuring device that can be an alternative to the already existing methods. The device is built around a Motorola 68008 processor an uses a CCD camera and some elementary image analysis.

Innehåll

1 Specifikation **2**

2 Komponenter **3**
2.1 Kontrolldel . . . 3
2.1.1 Processor . . . 3
2.1.2 Systemklocka . . . 3
2.1.3 Minne . . . 3
2.1.4 Display . . . 3
2.1.5 Knappar . . . 4
2.1.6 Logik . . . 4
2.1.7 Buffert . . . 4
2.2 Kameradel . . . 5
2.2.1 CCD-kamera . . . 5
2.2.2 A/D-omvandlare . . . 5
2.2.3 Video Sync Separator . . . 5
2.2.4 Bildminne . . . 5
2.2.5 Räknare . . . 5
2.2.6 Oscillator . . . 5

3 Hårdvarukonstruktion **6**

4 Logik **7**
4.1 Adressering . . . 7
4.2 Avbrott . . . 7
4.3 A/D-omvandling . . . 9

5 Mjukvarukonstruktion **10**

6 Förbättringsförslag **11**

7 Sammanfattning **12**

A Kretsschema **13**
A.1 Kontrolldel . . . 13
A.2 Kameradel . . . 14

B Lattice-kod **15**
B.1 styrlogik.abl . . . 15

C Programkod **20**
C.1 digiruler.c . . . 20
C.2 pmain.68k . . . 25
C.3 avben.68k . . . 27
C.4 exp5.68k . . . 28
C.5 exp2.68k . . . 29

D Systemklocka **30**

E Bilder **31**

Inledning

Inom industrin är mätning av längd, bredd, tjocklek och läge några av de viktigaste mätningarna som görs. Det finns en rad olika sätt att bestämma dessa parametrar på; induktiva givare, kapacitiva givare, resistiva givare, trådtöjningsgivare, fotoceller med mera. Flera av dessa har nackdelar, några är känsliga mot elektromagnetiska störningar och andra klarar inte av att användas i vissa miljöer.

Tanken med detta projekt var att göra ett alternativ till befintliga metoder och skapa en beröringsfri givare som ej påverkas av yttre störningar som elektromagnetiska fält, smuts etc. Detta gjordes med hjälp av en CCD-kamera och en digital konstruktion.

Projektet genomfördes i kursen Digitala Projekt 8p under våren 2005 vid Lunds Tekniska Högskola.

1 Specifikation

Projektet går ut på att vi vill kunna mäta ett föremåls approximativa storlek genom att beräkna dess största bredd och dess största höjd. Detta skall göras med en CCD-kamera, en processor och diverse andra digitala komponenter.

Principen för mätningen bygger på att två olika bilder tas med kameran, en bild tagen utan mätobjekt och en med. Därefter kommer en jämförelse mellan de två bilderna att ske. De pixlar som har ändrats från den först tagna bilden är de som ingår i mätobjektet och man kan sedan, utifrån de ändrade pixlarna, beräkna föremålets storlek.

När beräkningen är färdig presenteras resultatet, det vill säga bredd och höjd, på en display. Applikationen innehåller också två knappar, en resetknapp för att nollställa och en knapp för att starta bildtagningen.

Avgränsningar

För att kunna beräkna vad en pixel motsvarar i verklig längd krävs det att vi vet avståndet mellan kameran och mätobjektet. För att inte göra bygget allt för komplicerat är ett fast avstånd mellan mätobjekt och kameran förbestämt för beräkningar i programmet. Detta avstånd ska således också hållas vid mätning.

Ljuskällor i omgivningen kan göra att det bildas skuggor som kan störa mätningarna. För att slippa detta krävs att den starkaste ljuskällan ligger nära applikationen då bilden tas.

Ytterligare några avgränsningar görs angående mätningen för att begränsa projektet:

- Mätobjektet ska vara mycket mörkare än bakgrunden
- Bakgrundens nyans skall vara relativt homogen
- Bakgrunden skall vara plan (för att undvika skuggor etc.)
- Mätobjektet skall vara ett sammanhängande stycke
- Mätobjektet måste vara större än en viss storlek

2 Komponenter

När komponenter valdes var det tvunget att tänka på prestanda och tidskrav. Efter hand som bygget fortlöpte uppkom behov av ytterligare komponenter som inte hade uppmärksammats vid kretsdesignen. Konstruktionen delades upp i två delar, en kontrolldel och en kameradel.

I detta kapitel finns beskrivning av alla komponenter i dessa två delar samt deras funktion.

2.1 Kontrolldel

Kontrolldelen innehåller i princip en dator uppbyggd runt motorola-processorn. I denna del sker alla uträkningar av bilderna och kommunikation med tryckknappar och display etc. Denna del och kameradelen hålls åtskilda med buffrar som gör att det går att stänga av adressbuss och databuss från kontrolldelen till bildminnet. Detta måste göras så att inte kontrolldelen stör när A/D-omvandlaren skickar information till bildminnet.

2.1.1 Processor

Vi valde Motorola 60008 som processor. Denna är just bara en processor och har inte inbyggt minne som till exempel en AVR, vilket innebär att man måste ha externt programminne och ramminne. 68008 har en databuss på 8 bitar och en adressbuss på 20 bitar vilket innebär att den kan adressera 1 Mbyte. För att undvika kollision på databussen måste alla enheter kunna gå i tri-state, det vill säga deras ut- och ingångar blir högimpediva.

- Motorola 68008 Microprocessor

2.1.2 Systemklocka

För att kunna klocka vårt system behövs en klocka. Denna byggdes med hjälp av en kristall och några kringkomponenter.
Ritning på klockan kan ses i appendix D.

2.1.3 Minne

För att lagra programmet som processorn ska köra valdes ett EPROM på 8 kbyte. Dessutom behövs ett ram-minne för att lagra variabler och stack i.
Eftersom beräkningar på många pixlar ska göras så tog vi i ordentligt och valde ett SRAM på 32 kbyte.

- 27C64 8 kbyte x 8-bit parallell EPROM
- 43256 32 kbyte x 8-bit parallell CMOS SRAM

2.1.4 Display

Vi ska bara kunna skriva ut längd och bredd på objektet, så därför räcker det bra med en vanlig 16x2 LCD. Denna följer standarden för alfanumeriska displayer, är väldokumenterad och enkel att använda.

- 16x2 LCD Alfanumerisk teckendisplay

2.1.5 Knappar

För att kunna ta insignaler från användaren behövs knappar. Det behövs en resetknapp för att nollställa hela konstruktionen och en knapp för att ta bild.

2.1.6 Logik

Eftersom periferikretsarna ska kunna gå i tri-state så måste vi kunna generera deras Chip Select-signal. Detta görs med någon sorts logik som kan avkoda vilken krets som processorn vill kontakta. Dessutom behövs det genereras avbrott till processorn samt kunna styra A/D-omvandlingen. Detta leder till många logiska uttryck som skulle bli omfattande att bygga med grindar. Därför valde vi en programmerbar logikkrets från Lattice som har 64 in/utgångar och är lätt att programmera.

- ispLSI 1032E Programmable Logic

2.1.7 Buffert

Adressbussen och databussen ska endast ha kontakt med bildminnet då processorn ska läsa bilden ur minnet. Övrig tid ska kameradelen vara bortkopplad så att A/D-omvandling kan ske utan kollissioner på bussarna.Detta löses genom att ha buffertar mellan kontrolldell och kameradel. Dessa fungerar som en kanal som bara öppnas när processorn anropar bildminnet.

- 74HC541 Buffert

2.2 Kameradel

2.2.1 CCD-kamera

Kameran som används är en svartvit CCD-kamera. CCD står för Coupled Charged Device.

2.2.2 A/D-omvandlare

Bilden från kameran kommer ut som en analog signal. Därför måste den A/D-omvandlas så att den kan sparas undan för behandling. Det är viktigt att bildtagningen går så fort som möjligt, och således valde vi en höghastighets A/D-omvandlare.

- TDA8703 8-bit High Speed A/D Converter

2.2.3 Video Sync Separator

För att kunna börja A/D-omvandla när en ny bild börjar från kameran använder vi en krets som ger en utsignal när detta händer. När denna signal kommer igen så börjar nästa bild och vi kan sluta A/D-omvandla. På detta sätt får vi in rätt information för en bild.

- LM1881 Video Sync Separator

2.2.4 Bildminne

Den kodade bilden från CCD-kameran lagras i ett separat bildminne och kan därifrån läsas av processorn. Ett krav på minnet är att det ska vara snabbt så att kodningen av bilden blir så snabb som möjligt. Storleken på minnet är svårt att uppskatta men 128 kbyte bör räcka.

- 43100 128 kbyte x 8bit High Speed CMOS SRAM

2.2.5 Räknare

Datan från A/D-omvandlaren sparas på bildminnet, men varje gång en ny pixel är kodad så ska den in på en ny adress i bildminnet. Adressen till rätt plats i bildminnet räknas upp av en räknare som styrs av logikkretsen.

- 54HC590 8-bitars räknare

2.2.6 Oscillator

I testfasen vill vi kunna ställa in frekvensen på samplingen av bilden. Detta för att kunna se vad som blir lagom antal pixlar för att kunna räkna ut objektets storlek. För att kunna variera frekvensen så använder vi en EXO-krets. Denna har en grundfrekvens som kan delas ner genom att ställa in ett binärt tal på tre inputben. Lägsta frekvensen är grundfrekvensen delat med 256.

- EXO-3 20 MHz Variabel Oscillator

3 Hårdvarukonstruktion

Innan hårdvaran byggdes ritades ett kopplingsschema över hela konstruktionen. Detta ritades i Power Logic och var mycket värdefullt att ha vid själva bygget. Schemat kan ses i appendix A.

Konstruktionen byggdes på ett förborrat och företsat laborationskort med hjälp av lödning och virning. Virtekniken användes för att underlätta så mycket som möjligt vid eventuella felkopplingar och ändringar. De digitala komponenterna hölls avskilda från de analoga, detta för att undvika störningar från de senare nämnda. Avkopplingar gjorde för varje krets för att minska risken för oönskade strömmar i jorplanet.

Först byggdes kontrolldelen vilket i stort gick ut på att bygga en dator baserad på Motorolas 68008-processor. Mycket tid gick åt till att vira alla signaler, särskilt data- och adressbuss. Lattice-kretsen monterades på kortet och alla styrsignaler kopplades in på avsedda pinnar. Dessutom skulle interfacet för att programmera den kopplas ihop. Lattice-kretsen programmeras med speciell programvara till PC.
Kameradelen var lite mer komplicerad att bygga och konstruera. Räknaren för att räkna upp adressen till bildminnet fick konstrueras av tre stycken räknare. För att kaskadkoppla dessa krävdes lite knep för att få dem att räkna rätt. För att kunna A/D-omvandla en hel bild var det tvunget att konstruera en tillståndsmaskin och detta gjordes enklast i Lattice-kretsen.

För att kunna testa de olika periferienheterna användes ett utvecklingssytem för 68008, utvecklat av it-institutionen vid LTH. Detta system underlättade mycket då man kunde prova ut enhet efter enhet och direkt korrigera fel.

4 Logik

4.1 Adressering

För att processorn ska få kontakt med rätt periferienhet så måste den adressera den. Då ska denna enhet få chip-select signal och bli aktiv. För att implementera detta använde vi oss av den programmerbara logikkretsen från Lattice.
EPROMet är på 8 kbyte, det vill säga processorn behöver 13 pinnar för att adressera detta. Värre är det för RAM-minnet och bildminnet, dessa behöver 17 pinnar för att adresseras, vilket lämnar 3 pinnar över för att avkoda dem. I tabellen nedan ses hur adresspinnarna skall vara ställda för att processorn ska få access till just den enheten. Ur denna tabell härleds sen enkelt logiska uttryck för chip-select signalerna till de olika enheterna.
Koden finns i appendix B.1

Enhet	*A18*	*A17*	*A16*	*A15*	*A14*	*A13*	*A12 .. A0*	*Adress*
EPROM	0	0	0	0	0	0	-	0x00000
LCD, instr.	0	0	0	0	0	1	-	0x02000
LCD, data	0	0	0	0	0	1	-	0x02001
Parallellport	0	0	0	1	0		-	0x06000
SRAM	0	1	0	-	-	-	-	0x10000
Bildminne	0	1	1	-	-	-	-	0x20000

- = don't care

4.2 Avbrott

När man ska göra en mätning så ska två bilder tas. När man är redo att ta första bilden trycker man på en knapp. Då genereras ett avbrott till processorn som i sin tur kan skicka en signal på parallelporten till Latticekretsen om att det är dags att starta A/D-omvandlingen. När den första bilden är tagen skickas ett avbrott till processorn. Denna är då beredd att ta emot ett knapptryck till för att starta bildtagningen av bild nummer två.

Processorn har tre avbrottsnivåer 7, 5 och 2. Nummer 7 tar lite längre tid att behandla, och därför valdes nivå 5 och 2. När man vill generera ett avbrott skickar man två signaler, IPL0/2 och IPL1 som kodar vilket nivå avbrottet har. Det finns två sorters avbrott, vektoriserat avbrott och autovektor avbrott. När man har periferienheter som inte kan svara med en adress till en vektor som innehåller avbrottsprogrammet så får man använda autovektor. Med autovektor kan man i mjukvaran bestämma var i koden man ska hoppa vid ett visst avbrott. När man vill använda autovektor så måste man skicka en signal $\overline{VPA}$ till processorn för att tala om att det är autovektoravbrott det är frågan om.

Processorn har en tre-bitars utgång för att berätta i vilket tillstånd den är just nu. Dessa tre bitar kallas FC0, FC1 och FC2. När alla tre ger hög utgång så är processorn i *interrupt aknowledge*, vilket innebär att den är redo att ta emot ett avbrott.

Av detta kan vi skapa ett antal logiska uttryck för att kunna generera rätt

avbrott. Som insignaler fås knapptryck (avbrott 2), bildtagning klar (avbrott 5) samt {FC0,FC1,FC2}. Som utsignaler fås $\overline{VPA}$ och {IPL0/2,IPL1}.
Koden kan ses i appendix B.2

4.3 A/D-omvandling

När en bild ska A/D-omvandlas så ska först en pixel samplas sen läggas in på rätt ställe i bildminnet. Adressen genereras av räknarna. Dessutom ska ett avbrott genereras när bildtagningen är klar. För att få ihop detta är det ett antal olika kontrollsignaler som ska genereras vid rätt tidpunkt. Det behövdes en tillståndsmaskin. Denna implementerades i Lattice/ABEL.
Här följer en förklaring av de olika tillstånden:

S_0 : Starttillstånd. När signalen START_PICTURE skickas från processorn hoppar maskinen till nästa tillstånd.

S_1 : Kretsen väntar på att vsync-signalen ska komma, det innebär att en ny bild börjar.

S_2 : Samplingen börjar. Räknaren startas och writesignal ges till bildminnet.

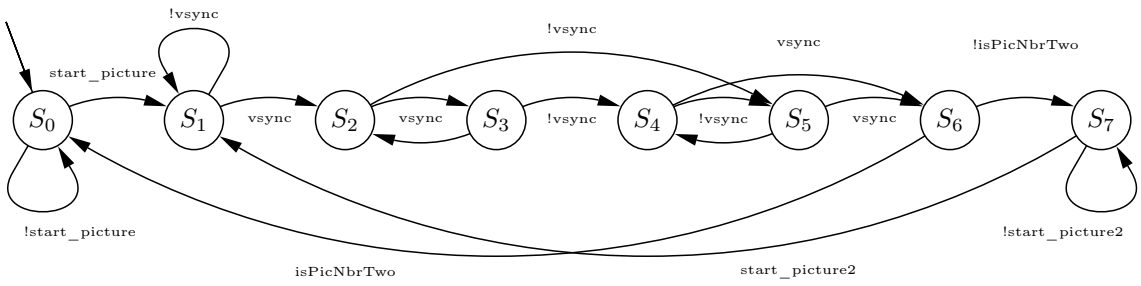
S_3 : Räknaren haltas och den aktuella pixeln skrivs in i minnet. Maskinen stannar i denna loop mellan S_2 och S_3 tills vsync inte gäller.

$S_{4,5}$: Här händer samma sak som i S_2 och S_3. När signalen vsync kommer igen hoppar maskinen till S_6.

S_6 : Samplingen av en bild är klar. Ett avbrott skickas till processorn som kan ge användaren klartecken för att ta nästa bild. En variabel, isPicNbrTwo, sätts första gången till 1. Sen hoppar maskinen till S_7. När maskinen hamnar här nästa gång ser den att den varit här innan och det är den andra bilden som är tagen. Nästa tillstånd blir då S_0.

S_7 : Maskinen väntar i detta tillstånd tills den får signalen START_PICTURE2 och hoppar då till S_1.

I figur 1 kan grafen för tillståndsmaskinen ses. Koden kan ses i appendix B.3.



Figur 1: Graf för tillståndsmaskin

5 Mjukvarukonstruktion

Mjukvaran till processorn skrevs i assembler och ANSI C.
Även vid mjukvarukonstruktionen var utvecklingssystemet till stor nytta vid felsökningar. Detta har funktioner för att sätta brytpunkter i programmet vilket användes flitigt! Programmeringen av mjukvarn var utan tvivel den lättaste delen i projektet, med undantag av rutinen för att räkna ut objektets storlek.. Tyvärr tog hårdvarukonstruktionen för mycket tid och det fanns ingen tid över till att skriva denna rutin. Detta kommer dock att göras efter kursens slut. Programkoden kan ses i appendix C.

6 Förbättringsförslag

För att kunna beräkna vad en pixel motsvarar i verklig längd krävs det, som tidigare nämnts, att vi vet avståndet mellan kameran och mätobjektet.
Till applikationen gjordes avgränsningen att bara mäta på ett fast avstånd som var förbestämt och inlagt i programmet. För att förbättra applikationen kunde man tänka sig att kunna mäta på alla avstånd genom att knappa in avståndet. Ytterligare en förbättring hade varit att bygga in en avståndsmätare i applikationen. Manuell mätning kunde då slopas helt och vi kunde även mäta föremålets djup.

För att skuggan av objektet inte skall göra ljusändringar i bakgrunden krävs det att den starkaste ljuskällan ligger i närheten av mätobjektet. Ett bra alternativ hade varit att ha någon sorts blixt som placerade skuggan bakom mätobjektet och då inte påverkar bilden.

För att kunna ställa in avståndet så att hela objektet ryms innanför kamerans område hade en inbyggd bildskärm av något slag underlättat.

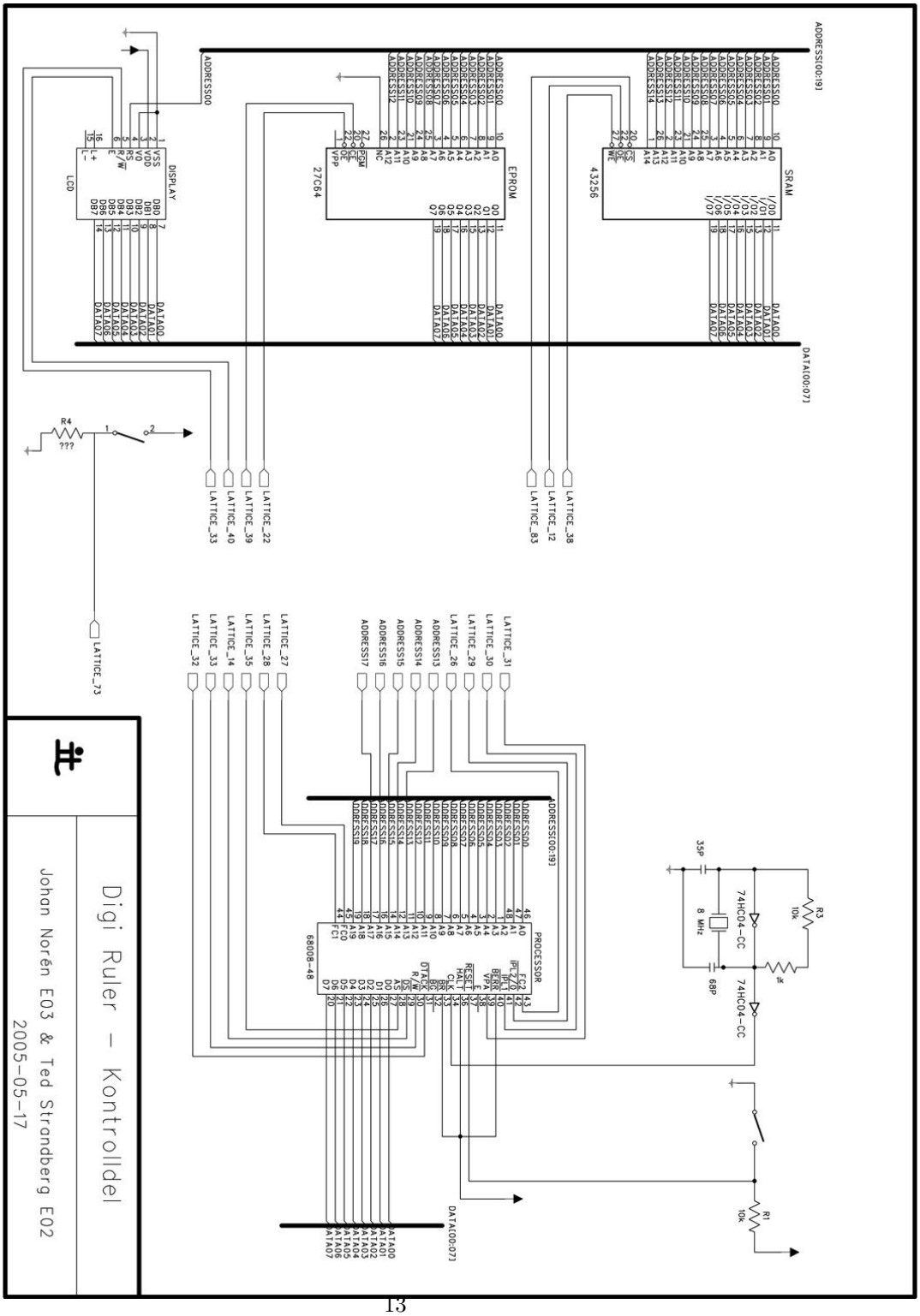
7 Sammanfattning

Projektet innebar mycket mer arbete än vad vi först trodde, och då hade vi ändå trott att det skulle innebära massor av jobb. Det uppkom hela tiden nya problem som vi inte hade tänkt på. Dessa skulle ha en lösning och det innebar alltså mer tid. Efter många svettiga och mycket intressanta timmar har vi nu en konstruktion som nästan fungerar och vi är nöjda med att vi kom så här långt. Genom att läsa kursen Digitala Projekt så har vi ökat på våra digitalteknik-kunskaper med några hundra procent, utan tvivel! Det är roligt att få omsätta teori i praktik och verkligen få se en riktigt produkt växa fram.
Kursen rekommenderas verkligen för dem som vill få en större förståelse för digitala kretsar och deras konstruktion.

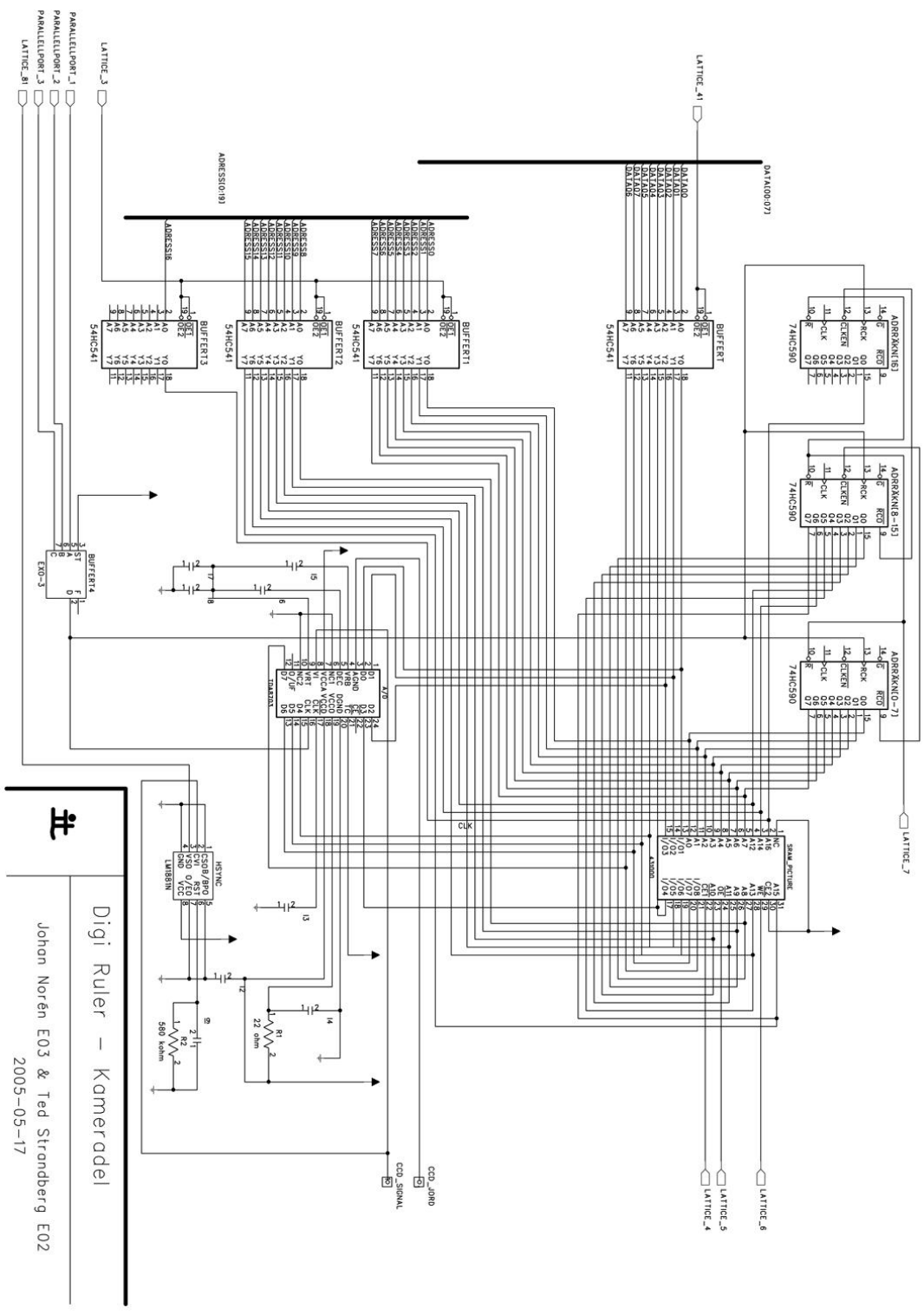
Slutligen vill vi tacka vår handledare Bertil Lindvall som hjälpt oss med våra många och dumma(?) frågor!

A Kretsschema

A.1 Kontrolldel



A.2 Kameradel



B Lattice-kod

B.1 styrlogik.abl

```
"Digitala Projekt 2005
"Johan Norén & Ted Strandberg
"2005-05-17

module styrlogik;

" input pins
A13, A14, A15, A16, A17 pin 8, 47, 48, 49, 50;  "Adresspinnar 13-17"
DS pin 14;
AS pin 35;
RW pin 33;


vsync pin 81;                           "Vsync-signal, aktivt hög"

START_PICTURE pin 80;                   "pinne 4 på parallellport,
                                        "för att starta bildtagning av bild 1

START_PICTURE2 pin 60;                  "pinne 6 på parallellport,
                                        "för att starta bildtagning av bild 2

isPicNbrTwo pin 51;                     "boolean för att hålla reda på om det
                                        "bild nummer ett eller två som tas
clock pin 84;                           "Klocksignal från EXO-3


" output pins

OE_EPROM pin 22;                        "Output enable EPROM, aktivt låg"
CE_EPROM pin 39;                        "Chip-enable EPROM, aktivt låg"


CS_SRAM pin 83;                         "Chip-select SRAM, aktivt låg"
OE_SRAM pin 12;                         "Output-enable SRAM, aktivt låg"
WE_SRAM pin 38;                         "Write-enable SRAM, aktivt låg"
CE_BILDRAM pin 4;                       "Chip enable bildmine, aktivt låg"
CE_BILDRAM_STATEMACHINE pin 74;
OE_BILDRAM pin 5;                       "Output enable bildminne, aktivt låg"
WE_BILDRAM pin 6;                       "Write enable bildminne, aktivt låg"

E_DISPLAY pin 9;                        "Chip-select display"
OE_BUFFERT pin 41;                      "Output enable databuffert"
OE_ADRESSBUFFERT pin 3;                 "output enable adressbuffert"
OE_LATCH pin 45;                        "Output enable latch"
```

```
OE_PARALLELLPORT pin 77;                "Ouput enable parallelport (latch),
                                        "aktivt låg"
LE_PARALLELLPORT pin 79;

CE_AD pin 76;                           "Chip enable A/D-omvandlare, aktivt låg"

OE_COUNTER pin 78;                      "output enable räknare
CCKEN_COUNTER pin 82;                   "count enable räknare
CCLR_COUNTER pin 7;                     "nollställning räknare

DTACK_PROC pin 32;                      "DTACK till processorn"

"------------------------- Pinnar för avbrottshantering ---------------------
VPA pin 31;                             "VPA - autovektor
FC0 pin 27;
FC1 pin 28;
FC2 pin 26;
IPL02 pin 29;                           "IPL0/2
IPL1 pin 30;                            "IPL1
KNAPP_TA_BILD pin 73;                   "Avbrottssignal frŇn knapp
                                        "- ta bild, aktivt hög
BILD_KLAR pin 72;                       "Avbrottsignal från A/D-omvandling
                                        "- bild klar, aktivt hög

"-------------- Tillståndsvariabler för tillståndsmaskinen -------------------

q3 node istype 'REG_D,BUFFER,KEEP';
q2 node istype 'REG_D,BUFFER,KEEP';
q1 node istype 'REG_D,BUFFER,KEEP';
q0 node istype 'REG_D,BUFFER,KEEP';

sreg = [q3,q2,q1,q0];

"Tillståndsvärden
A = 0;
B = 1;
C = 2;
D = 3;
E = 4;
F = 5;
G = 6;
H = 7;
I = 8;

equations

        "Ekvationer för chipselect

        !CE_EPROM = (!A17 & !A16& !A15 & !A14 & !A13 & !AS);
        !OE_EPROM = (!DS & RW);
```

```
        !CS_SRAM = (!A17 & A16 & !A15 & !AS);
        !OE_SRAM = (!DS & RW);
        !WE_SRAM = (!DS & !RW);

        !CE_BILDRAM = (A17 & !AS) # (CE_BILDRAM_STATEMACHINE);
        !OE_BILDRAM = (!DS & RW);
        !WE_BILDRAM = (!DS & !RW);

        E_DISPLAY = !A17 & !A16& !A15 & !A14 & A13 & !DS;

        !OE_BUFFERT = (A17 & !AS);

        !OE_ADRESSBUFFERT = (A17 & !AS);

        !OE_LATCH = (!A17 & !A16& !A15 & A14 & !A13 & !AS)


        OE_PARALLELLPORT = 0;

        LE_PARALLELLPORT = !A17 & !A16 & !A15 & A14 & A13 & !DS;

        DTACK_PROC = !(!CE_EPROM # !CS_SRAM # !E_DISPLAY # !CE_BILDRAM #
        !OE_BUFFERT # !OE_LATCH # !LE_PARALLELLPORT);


        "Ekvationer för tillståndsmaskinen
        [q3,q2,q1,q0].clk = clock;


        "Ekvationer för avbrotthantering
        !VPA= FC0 & FC1 & FC2 & !AS;
        !IPL02 = BILD_KLAR;
        !IPL1 =  KNAPP_TA_BILD & !BILD_KLAR;

state_diagram sreg;

        State A:

                CCLR_COUNTER = 1;
                CE_AD = 1;
                CE_BILDRAM_STATEMACHINE = 1;
                BILD_KLAR = 0;

                IF (START_PICTURE) THEN B;
                ELSE A;


        State B:
```

```
        IF (vsync) THEN C;
        ELSE B;

State C:


        OE_COUNTER = 0;

        goto D;

State D:

        IF (vsync) THEN E WITH {CE_AD = 0;
                                CE_BILDRAM_STATEMACHINE = 0;
                                WE_BILDRAM = 0;
                                CCKEN_COUNTER = 1;}
        ELSE G;

State E:

        IF (vsync) THEN D WITH {CE_AD = 1;
                                CE_BILDRAM_STATEMACHINE = 1;
                                WE_BILDRAM = 1;
                                CCKEN_COUNTER = 0;}
        ELSE F;


State F:

        CE_AD = 1;
        CE_BILDRAM_STATEMACHINE = 1;
        WE_BILDRAM = 1;
        CCKEN_COUNTER = 0;

        IF (vsync) THEN H WITH{CE_AD = 1;
                               CE_BILDRAM_STATEMACHINE = 1;
                               WE_BILDRAM = 1;
                               CCKEN_COUNTER = 1;}

        ELSE G;

State G:

        CE_AD = 0;
        CE_BILDRAM_STATEMACHINE = 0;
        WE_BILDRAM = 0;
        CCKEN_COUNTER = 1;

        IF (vsync) THEN H WITH{CE_AD = 1;
                               CE_BILDRAM_STATEMACHINE = 1;
```

```
                                        WE_BILDRAM = 1;
                                        CCKEN_COUNTER = 1;

                ELSE F;


        State H:

                BILD_KLAR = 1;

                IF (isPicNbrTwo) THEN A WITH {CCLR_COUNTER = 0;
                                              isPicNbrTwo = 0;}

                IF (!isPicNbrTwo) THEN I WITH {isPicNbrTwo = 1;}
                ELSE H;



        State I:

                IF START_PICTURE2 THEN B;

                ELSE I;

end styrlogik;
```

C Programkod

C.1 digiruler.c

```
/*****************************************************************************
                            digiruler.c
                          ------
   Detta ar huvudprogrammet som ar skrivet i ANSI C. Exekveringen av hela
   programpaketet borjar i pmain.68k (lage __main).

   exp2()  och exp5() anropas fran assemblyprogrammet exp2.68k
   resp. exp5.68k vid avbrott.

   _avben() anropar avben.68k vilket tillater avbrott fran PI/T.
                          ------
  2005-05-17
  Johan Norén - e03jno@efd.lth.se
  Ted Strandberg - e02ts@efd.lth.se
*****************************************************************************/
unsigned short int *disp_cmd;   /* Display instruktioner*/
unsigned short int *disp_data;  /* display data */
unsigned short int *sram;       /* Ramminne */
unsigned short int *picram;     /* Bildminne*/
unsigned short int *parallel;   /*Parallellport*/

unsigned short int interrupt2;
unsigned short int secondPic;

/* main: huvudprogram */
main(){
        disp_cmd = (unsigned short int *) 0x2000;
        disp_data = (unsigned short int *) 0x2001;
        sram = (unsigned short int *) 0x10000;
        picram = (unsigned short int *) 0x20000;
        parallel = (unsigned short int *) 0x6000;

        interrupt2 = 0;
        secondPic = 0;

        init();                                 /* initsiering */
        _avben();               /* tillåt avbrott */
        loop: goto loop;        /* evig loop */

}

/************************ Bildhantering ****************************/
/* takePicNbrOne: ta första bilden  */
takePicNbrOne(){
        lcd_clear();
        lcd_write_ln1("Capturing image");
```

```
        lcd_write_ln2("number one...");
        wait_variable(0x3e8);
        *parallel=0x0f;
        return(0);
}

/* takePicNbrTwo: ta andra bilden  */
takePicNbrTwo(){
        lcd_clear();
        lcd_write_ln1("Capturing image");
        lcd_write_ln2("number two...");
        wait_variable(0x3e8);
        *parallel=0x27;
        return(0);
}

/* calculate: räkna ut höjd och bredd  */
calculate(){

        /* massor av kod...
        ....
        ....
        .... */

        /* För syns skull så länge...*/
        lcd_clear();
        lcd_write_ln1("Calculating...");
        wait_variable(0x41a);
        lcd_clear();
        lcd_write_ln1("Height: ?? mm");
        lcd_write_ln2("Width : ?? mm");

        wait_variable(0x41a);
        lcd_clear();
        lcd_write_ln1("Press button to");
        lcd_write_ln2("take first pic.");
        interrupt2 = 0;
        return(0);
}

/*************************** System*******************************/

/* init: Initiering och välkomsttext  */
init(){
        *parallel=0x07; /* ställer in parallellporten , ställer EXO-kretsens frekvens */
        lcd_init(); /*Starta initiering av LCD*/
        lcd_write_ln1("Digi Ruler v 0.1");
        lcd_write_ln2("----------------");
        wait_variable(0x41a);
        lcd_clear();
```

```
        lcd_write_ln1("Press button to");
        lcd_write_ln2("take first pic.");
        }

/************************ LCD-display ****************************/

/* lcd_init: Funktion för att initiera LCD */
lcd_init() {
        wait();
        *disp_cmd=0x30;
        wait();
        *disp_cmd=0x30;
        wait();
        *disp_cmd=0x30;
        wait();
        *disp_cmd=0x1;
        wait();
        *disp_cmd=0x38;
        wait();
        *disp_cmd=0xf;
        wait();
        *disp_cmd=0x6;

        return(0);

}

/* lcd_clear: rensar display */
lcd_clear(){
        wait();
        *disp_cmd=0x1;
        return(0);
}

/* lcd_write: skriv sträng till rad 1 på lcd */
lcd_write_ln1(char str[]){
    int i=0;
        while (str[i]!='\0')
    {
        wait();
                *disp_data=str[i];
        i++;
    }
        return(0);

}

/* lcd_write: skriv sträng till rad 2 på lcd */
lcd_write_ln2(char str[]){
    int i=0;
```

```
        wait();                 /* Vänta */
        *disp_cmd=0xc0; /* Hoppa till andra raden */
        while (str[i]!='\0')    /* Skriv tecken från strängen */
    {                                           /* tills returntecken påträffas */
                wait();
                *disp_data=str[i];
        i++;
    }
        return(0);

}

/* wait: väntrutin */
wait(){
        int k=0;
        do{
                k++;
        }while(k <=0x200);

}

/* wait_variable: väntrutin med ställbar delay  */
wait_variable(int count){
        int i;
        int a;
        for (i=0;i<=count;i++){
                for (a=0;a<=count;a++){
                }
        }
}

/***************************** Avbrottsrutiner ***********************/

/* exp2: avbrottsprogram för avbrottsnivå 2 - knapptryck */
exp2(){
        if ((interrupt2 == 0) && (secondPic == 0)) {
                interrupt2 = 1;
                takePicNbrOne();
                return;
        }
        else if ((interrupt2 == 0) && (secondPic == 1)) {
                interrupt2 = 1;
                takePicNbrTwo();
                return;
        }
        else return;
}

/* exp5: avbrottsprogram för avbrottsnivå 5 - bild klar */
exp5(){
```

```
        *parallel=0x07; /*nollställer parallellport */

        if (secondPic == 0){
                secondPic = 1;
                lcd_clear();
                lcd_write_ln1("First picture");
                lcd_write_ln2("completed.");
                wait_variable(0x250);
                lcd_clear();
                lcd_write_ln1("Put object in");
                lcd_write_ln2("front of screen");
                wait_variable(0x280);
                lcd_clear();
                lcd_write_ln1("and press button");
                lcd_write_ln2("to take 2:nd pic");
                interrupt2 = 0;
                return;
        }
        else if(secondPic == 1) {
                secondPic = 0;
                calculate();
                return;
        }
        else return;
}
```

C.2 pmain.68k

```
*****************************************************************************
*                        pmain.68k
*                        ---------
*        Detta assemblyprogram
*        initierar stackpekaren,
*        den globala datapekaren {data},
*        avbrottsvektorn,
*        nollstaller alla register
*        samt gor ett subrutinanrop till _main.
*
*****************************************************************************
        xref    _main           Refererat till symbolen _main = main i test.c
M_main  equ     *
        section init
        xdef    __main          Definierar __main
__main
        xref    data            Segmentet data som ar den globala data-arean.
        xref    __exp2          Symbolen for avbrottshanteraren vid avbrott 4
        xref    __exp5

        movea.l #$00013ffe,A7   Stackpekare till 7ffc.
        move    #$2700,SR       Supervisor och inga avbrott tillatna.
        lea     data,A5         ladda A5 med global datapekare.
        suba.l  A6,A6           Nollstall frame pointer.
        lea.l   $00000068,A4    Ladda avbrottsvektorn
        move.l  #__exp2,(A4)
        lea.l   $00000074,A4    Ladda avbrottsvektorn
        move.l  #__exp5,(A4)
        suba.l  A4,A4
        suba.l  A3,A3
        suba.l  A2,A2
        suba.l  A1,A1
        suba.l  A0,A0
        clr.l   D7
        clr.l   D6
        clr.l   D5
        clr.l   D4
        clr.l   D3
        clr.l   D2
        clr.l   D1
        clr.l   D0
        jsr     _main           Hoppa till _main vilket ar i filen test.c

* Om nagot gar fel sa kommer programmet hit vid ett return fran _main
* och hamnar i en evig loop.
errloop
        jmp     errloop
```

```
*  Slut pa inits.
*
*  Org kommandot ar ett direktiv till assemblern, att allt efterfoljande
*  skall borja laggas pa den adress som anges. I detta fall ar det SP med
*  borjan pa adress 0 (dc.l => 4 byte) och sedan vardet pa PC.
*
        org     $0
        dc.l    $00007ffc
        dc.l    __main
*  Avbrottsvektorerna far ratt varde som ovan.
*
*       org     $70
*       dc.l    __exp4          PI/T on it68 board
* Nedanstaende definierar __main som startadress.
        end     __main
```

C.3 avben.68k

```
*********************************************************************
*                            avben.68k
*                            ---------
*           Ett program som tillater avbrottsniva 4 och hogre.
*           Anropas fran test.c med funktionen _avben().
*

*********************************************************************
        xdef    __avben             Definierar ett lage __avben.

__avben equ     *

        move    #$2100,SR           Tillat avbrott
        rts                         atervand till test.c
        end
```

C.4 exp5.68k

```
*************************************************************************
*                           exp5.68k
*                           --------
*           Avbrottshanterare till avbrottsniva 5
*
*************************************************************************

        xdef    __exp5              Definierar ett lage __exp5.
        xref    _exp5               Refererar till lage _exp5.

__exp5  equ     *

        movem.l D0-D7/A0-A7,-(A7)   Spara alla register!!
        jsr     _exp5               Hoppa till _exp4 i test.c.
        movem.l (A7)+,D0-D7/A0-A7   Aterstall ala register.
        rte                         Avsluta avbrott.
        end
```

C.5 exp2.68k

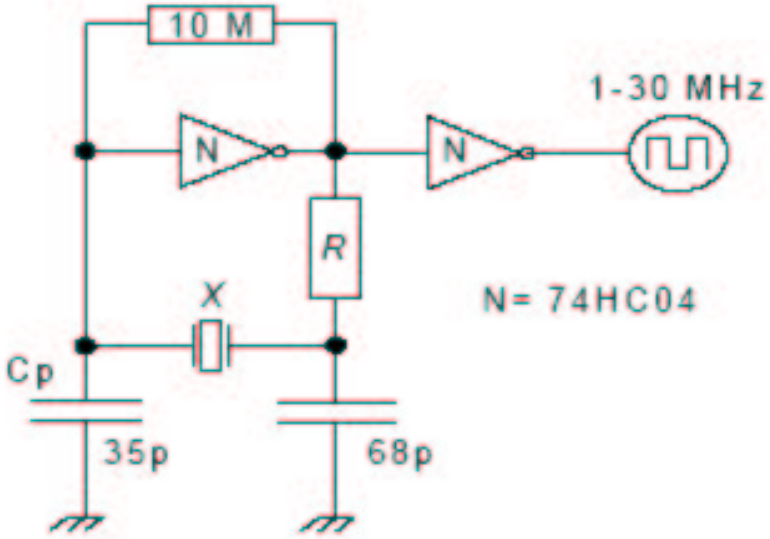
```
*************************************************************************
*                           exp2.68k
*                           --------
*           Avbrottshanterare till avbrottsniva 2
*
*************************************************************************

        xdef    __exp2              Definierar ett lage __exp2.
        xref    _exp2               Refererar till lage _exp2.

__exp2  equ     *

        movem.l D0-D7/A0-A7,-(A7)   Spara alla register!!
        jsr     _exp2               Hoppa till _exp4 i test.c.
        movem.l (A7)+,D0-D7/A0-A7   Aterstall ala register.
        rte                         Avsluta avbrott.
        end
```

D Systemklocka



E Bilder

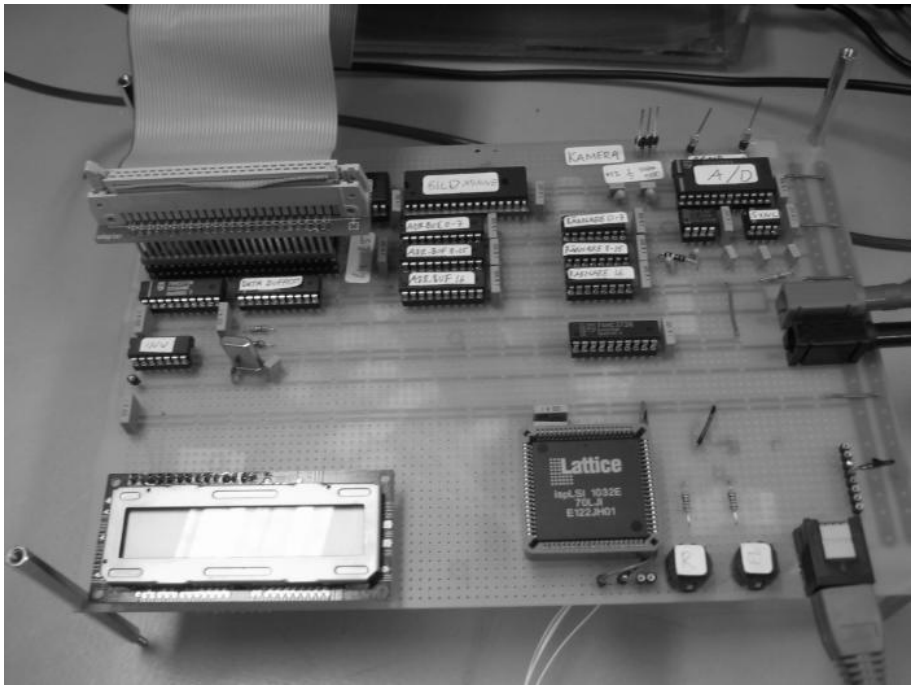


Bild 1: Laborationskort

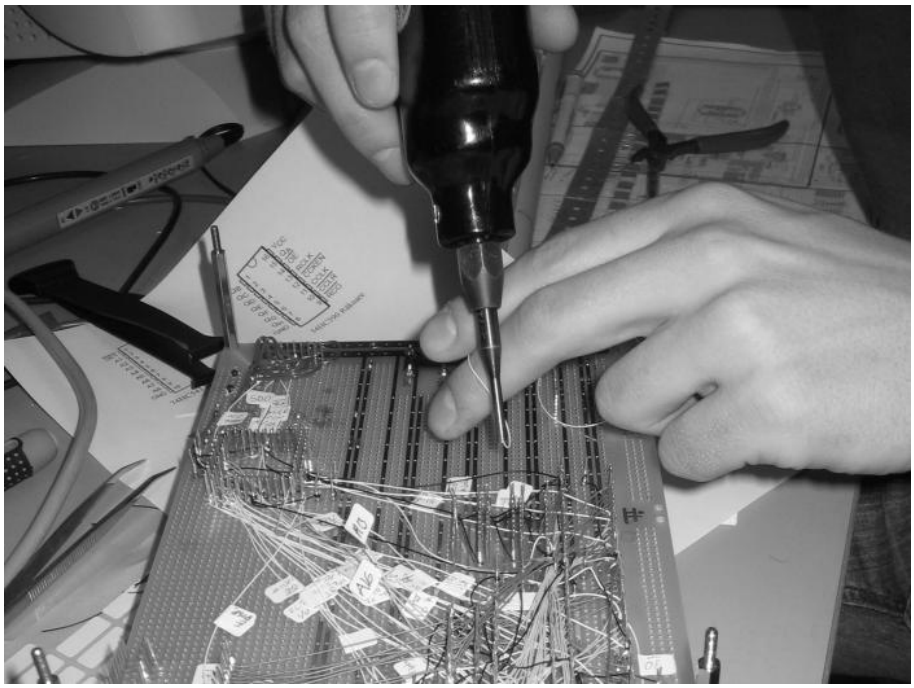


Bild 2: Virning

Bild 3: Hårt arbetande teknologer

Bild 4: Virning är terapi!