Robocar

Digitala Projekt 8p

2006-05-22

Johan Olsson E-01
Filip Oredsson E-01

Handledare: Bertil Lindvall

Abstract

In our project we wanted to build a car that could follow you around the corridors. Or just follow another moving object such as a ball or other moving robot.
This paper describes the development of a movement recognizing car named Robocar. The device is build around a Motorola 68008 processor and uses a CCD camera and basic picture analysis for movement recognizing.

Innehållsförteckning

1. Inledning

Vår uppgift var att konstruera en bil som med hjälp av en ccdkamera kan detektera rörelser och följa dessa. En A/D omvandlare behövde användas för att sampla fram bilderna ccd kameran gav och sedan skicka två bilder tagna i följd men med en kort paus sinsemellan till minnet. Bildtagning och inläsning i minnet styrs helt av en lattice krets men efter bilderna är inlästa i minnet tar processorn en MC68008 över kontrollen och kör därefter en bildanalys för att avgöra om bilderna som tagits skiljer sig markant åt och om så är fallet var i bilden en förändring har skett. Signaler skickas sen till bilens motorer som ser till att bilen följer efter den eventuella rörelsen.

Detta projekt har lärt oss mycket om grunderna inom konstruktion av hårdvara fungerar och det har varit nyttigt att få mer insikt i hur bland annat A/D-omvandling, bussar och kontrollsignaler fungerar.

2. Specifikation

Projektet går ut på att få en mekanisk bil att följa rörelser med hjälp av en CCD kamera. Tanken är att konstruktionen ska kunna känna av var i omgivningen rörelsen sker och sedan följa efter denna. Detta ska göras med hjälp av en bildanalys som jämför två bilder tagna av CCD kameran. Konstruktionen är uppbyggd kring en Motorola 68008 processor som tar hand om bildanalysen. Bildtagning och minneslagring är uppbyggd av programmerbar logik i en latticekrets.

3. Utrustning

3.1 Motorola 68008

68008 är en 32 bitars processor med 8-bits databuss. Den kan adressera 1Mb minne och valdes för att den var mer flexibel än AVR AT Mega16. I vårt projekt krävs stor flexibilitet genomgående i alla komponenter och framförallt ett större minne än AVR kan tillhandahålla.

3.2 CCD

Kameran vi använde heter VCM 3612/00. Den ger svart-vit video enligt PAL standard.

3.3 A/D-omvandling

A/D-omvandlingen är till för att omvandla videosignalen ifrån CCD-kameran till en digitalt samplad signal som kan läggas in i minnet. Adressen där varje bildpixel ska läggas bestäms av en inbyggd adressräknare i latticekretsen.

A/D-omvandlaren vi valde heter TDA8703 och är en 8-bitars omvandlare som kan sampla i 40 MHz. Den är speciellt bra för videoomvandling enligt databladet och har ett dynamiskt område på ingången som motsvarar standarden för PAL. Detta var anledningen till att denna A/D-omvandlare valdes samt att det skulle gå så fort som möjligt att generera bilden. Dock räckte det gott med 10Mhz visade det sig och därför utnyttjade vi inte A/D-omvandlarens hela kapacitet.

3.4 Minne

Vi använde oss av ett och samma minne för både programapplikationer och bildlagring. Därför behövdes ett hyfsat stort minne och vi valde därför ett 512x8 Parallell CMOS SRAM. På grund av att vi drog ner samplingshastigheten till 10MHz istället för 40MHz utnyttjades inte hela minnet, egentligen hade det räckt med 128kb.

3.5 Latticekrets

Bildtagningen och lagringen av bilderna i minnet sköts av latticekretsen. Större delen av programmeringen har därför skett i ABEL-HDL i latticekretsen. På grund av att vi behövde många I/O portar krävdes en stor krets. Vi använde oss av en ”Lattice 1032E High-Density Programmable Logic” som har 84 ben och tillräckligt många I/O portar för applikationen.

3.6 Motorer och styrkrets

För att göra kretskortet mobilt använde vi oss av en färdig modul som innehåller två likströmsmotorer, utväxling och två hjul. För att driva likströmsmotorerna användes en ”L298 Dual full-bridge driver” som är en H-brygga med digitala styrsignaler.

3.7 Videosync-separator

För att veta när varje bild och varje rad börjar så användes en LM1881, vilket är en krets som plockar ut digitala vsync och hsync från den analoga videosignalen.

3.8 Parallellportar

I konstruktionen behövdes det parallellportar på två ställen. En för att hålla styrsignalerna till tillståndsdiagrammet och bryggan, och en för att hålla utdatan från A/D-omvandlaren under hela skrivcykeln.

3.9 Spänningsregulatorer

Eftersom vi använde ett batteri som ger ut 6V så behövdes två regulatorer för att omvandla spänningen till 12 V till kameran respektive 5 V till övriga kretsar.

3.10 Klockgenerator

Vi använde oss av en EXO-3 för att generera en klocksignal på 10 MHz.

4 Konstruktion av hårdvara

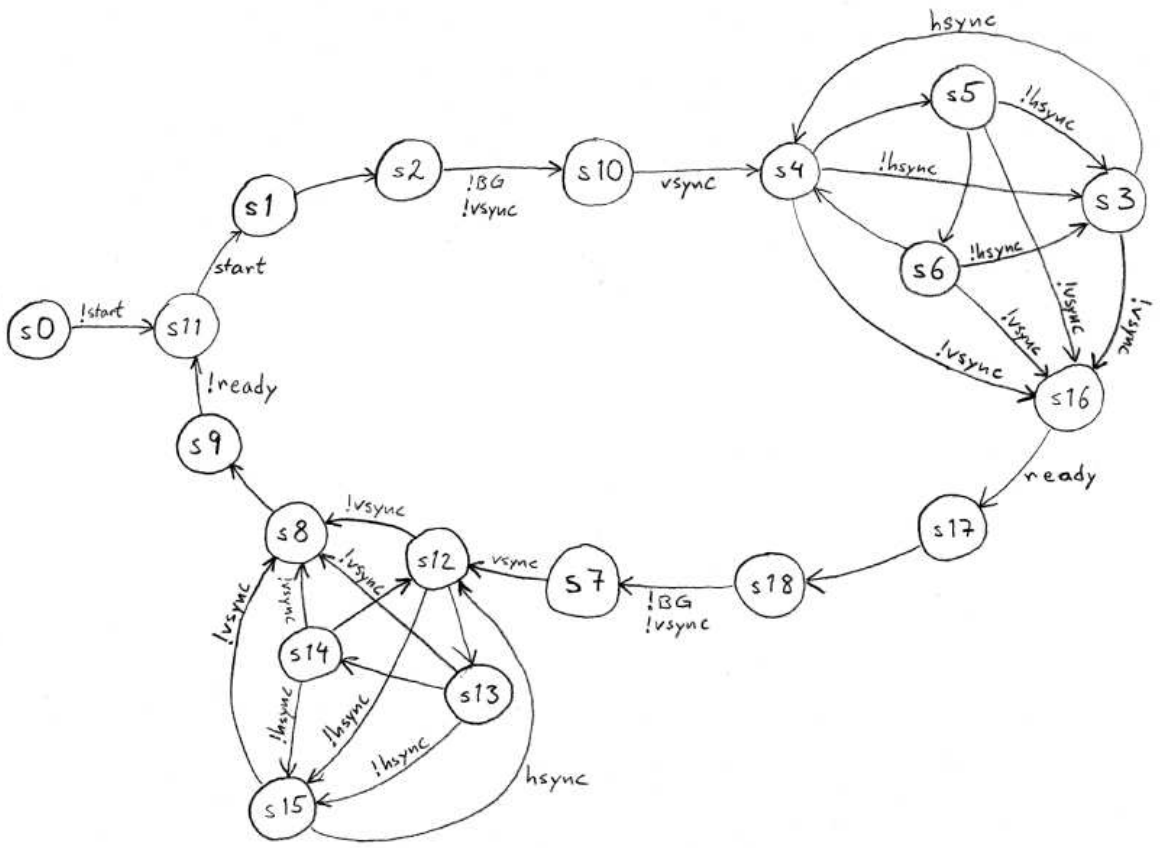
Först av allt ritades ett kretsschema upp i Powerlogic för att hålla reda på alla kopplingar som senare skulle viras mellan de olika komponenterna på kretskortet. Det gällde dock att ha huvudet med sig och inte lita fullt ut på kretsschemats komponenter då deras pinnar inte alltid stämde överens med databladen över komponenterna. Trots detta var kretsschemat oumbärligt för att hålla reda på alla kopplingar. Schemat kan ses i appendix A.

På det förborrade och företsade labrationskortet som tillhandahölls byggdes först en ”minidator” baserad på motorolas 68008-processor och minnet. Detta var en av de mest tidskrävande delarna av projektet då helst inga virningar fick bli fel. Den noggranna uppbyggnaden gav dock utdelning och efter att konstruktionen var

färdigvirad visade det sig att vi kunde skriva till och läsa från minnet utan problem. Testerna utfördes med hjälp av utvecklingssystemet it68 och efter klartecken därifrån kopplades kamera och motorer in och testades så att även de var rätt kopplade.

5 Lattice

Som nämnts ovan innehåller Latticekretsen merparten av koden som styr våran bil. En tillståndsmaskin för att ta två bilder och lagra dessa på två olika platser i minnet började efter hand ta form. Först försökte vi snåla in på de olika tillstånden och återanvända skrivningsloopen för de olika bilderna. Detta visade sig dock bli problematiskt då vi av någon anledning inte kunde få latticekretsen att hålla en variabel för om det var bild ett eller bild två som behandlades. Över huvud taget fick vi slumpartade felhopp i tillståndsmaskinen så fort vi försökte få latticekretsen att avgöra vilken av två vägar som skulle väljas då en skapad kontrollsignal styrde. Vi tror att störningar då registren i latticekretsen slår till kan ha haft en del i detta. Efter att ha lagt mycket till synes bortkastad tid på att göra ett litet och smart tillståndsdiagram för att styra bildhanteringen övergav vi tillslut denna tanke och började bygga ett stort och förhoppningsvis robust tillståndsdiagram som inte skulle kunna hoppa fel. Resultatet kan ses i *figur 5.1* nedan och koden finns i *appendix B*. För att få en möjlighet att jämföra två bilder där en förändring haft tid att ske görs en paus i diagrammet efter bild ett och databussen ges för enkelhetens skull tillbaks till processorn som väntar en kort stund innan lattice återfår den. Latticekretsen fortsätter sen med att lagra bild två i minnet innan processorn får tillbaks databussen igen.



Figur 5.1 *Tillståndsdiagrammet för bildtagning.*

5.1 Förklaring till tillståndsdiagrammet

S0:S11:S1:S2: Under dessa tillstånd nollställs adressräknaren och en bus request skickas till processorn. När bus granted fåtts från processorn fortsätter tillståndsmaskinen till S10.

S10: Vsync inväntas innan bildsamplingen börjar för att samplingen inte ska börja i mitten av en bild.

S4:S5:S6:S3: Bild nummer ett samplas och sparas i minnet. S3 ser till att Hsync signalen inte lagras som bildinformation.

S16:S17:S18: Bild ett är klar och lagrad i minnet. Nu ges kontrollen över till processorn som väntar en kort stund innan lattice återfår kontrollen och hoppar vidare till S7.

S7: Har samma funktion som S10, se ovan.

S12:S13:S14:S15: Bild nummer två samplas och sparas i minnet. S15 ser till att Hsync signalen inte lagras som bildinformation.

S8:S9: Bild två är klar och lagrad i minnet. Kontrollen ges tillbaks till processorn.

6 Konstruktion av mjukvara

Eftersom hela bildlagringsprogrammet gjordes i ABEL-HDL i latticekretsen återstod det inte mycket programmering när det väl blev dags att skriva C-kod. Vad som behövdes var framförallt en algoritm för att jämföra de två tagna bilderna och avgöra om och i så fall var i bilderna störst förändring föreligger. Efter att denna algoritm tagit reda på var i bilden största förändringen skett tar en ny algoritm över som skickar en signal till motorerna om att köra rakt fram alt svänga höger eller vänster beroende på vilken inparameter som givits från den förra algoritmen.
Även vid konstruktionen av mjukvaran var utvecklingssystemet it68 till stor nytta. Här gavs möjligheten att sätta brytpunkter i programmet vilket underlättade felsökningen avsevärt. C-koden kan ses i appendix C.

7 Slutsatser

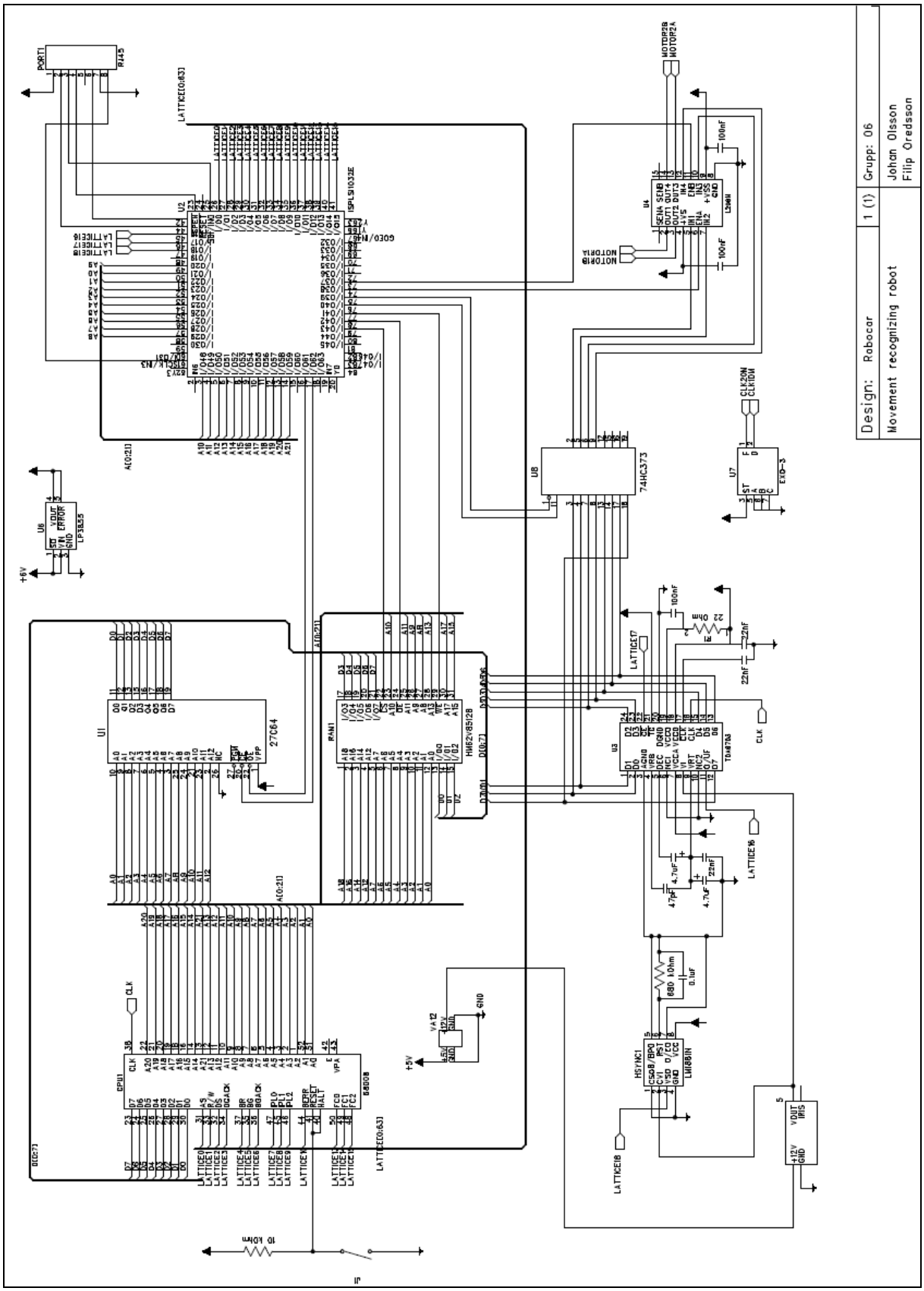
Förhoppningen med projektet var att kunna få våran bil att följa en tydlig rörelse i en annars tom korridor vilket kändes som en rimlig uppgift. Tyvärr stötte vi på ett större problem i A/D-omvandlingen än vi trodde från början. Trots massvis av avkopplingar runt alla komponenter fick vi problem med stora störningar. Detta resulterade i att bilderna som lästes in i minnet bara består av brus. Vi har lagt åtskilliga timmar på att försöka lösa just detta specifika problem. Bland annat genom att använda ett kort matlabscript som efter en del modifiering nu klarar av att läsa in en minnesarea och visa innehållet som en bild.
För att få konstruktionen att fungera som planerat krävs förmodligen en nyare och bruståligare teknologi.

8 Sammanfattning

Vårat projekt har krävt mycket tid men det har varit en fruktansvärt rolig och lärorik resa. Efter många timmar i labbet har vi löst många problem som satt griller i våra huvuden, tyvärr lyckades vi inte riktigt lösa alla. Resultatet är ändock något vi är mycket nöjda med, så även om konstruktionen inte fungerar riktigt som vi vill känns målet närmare än nånsin. Vi har under kursens gång lärt oss massvis om digitalteknik och att omsätta teori i praktiken har varit mycket spännande. Kursen digitala projekt rekommenderar vi därför varmt.

Vi vill även ge ett varmt tack till Bertil Lindvall för att alltid ha funnits tillhands med såväl teknisk som pizzarelaterad vägledning.

Appendix A


Design: Rabocar
1 (1)
Grupp: 06
Movement recognizing robot
Johan Olsson
Filip Oredsson

Appendix B

```
MODULE test

"Input pins

        B19,B18,B17,B16,B15,B14,B13,B12 pin 14,15,18,29,35,37,38,59;
        AS,RW,DS pin 26,27,28;
        BG pin 31;
        vsync pin 47;                           "Vsync from camera
        hsync pin 39;                           "Hsync from camera
        clock20 pin 20;                         "20 MHz clock på pin 66!
        start pin 75;                           "Startsignal for statediagram
        ready pin 69;                           "Readysignal from processor
        oe_abus pin 60;                         "Adressbus control
        oe_par2,le_par2 pin 74,73;              "oe och le för parallellporten som håller A/D omvandlarens data


"Output pins

        cs_prom,oe_prom pin 17,16;              "chip-select och output-enable to EPROM
        cs_ram,oe_ram,we_ram pin 79,78,77;      "chip-select, output-enable och write-enable to RAM
        cs_AD pin 46;                           "chip-select to ADC.
        le_par pin 76;                          "latch-enable for paralellport before bridge (active high)
        oe_par pin 68;                          "(active low)
        DTACK pin 32;
        BR pin 30;
        oe_addr pin 70;                         "Addresbus control

"Variables for state diagram

        q4 pin 40 istype 'REG_D,BUFFER,KEEP';   "Vit
        q3 pin 83 istype 'REG_D,BUFFER,KEEP';   "Svart
        q2 pin 82 istype 'REG_D,BUFFER,KEEP';   "Blå
        q1 pin 81 istype 'REG_D,BUFFER,KEEP';   "Gul
        q0 pin 80 istype 'REG_D,BUFFER,KEEP';   "Röd

        sreg = [q4,q3,q2,q1,q0];

        we_ram_proc node istype 'com';
        we_ram_state node istype 'com';
        cs_ram_proc node istype 'com';
        cs_ram_state node istype 'com';
        oe_ram_proc node istype 'com';
        oe_ram_state node istype 'com';


"Variables for adress counter

        a0 pin 50 istype 'REG_D,BUFFER,KEEP';
        a1 pin 51 istype 'REG_D,BUFFER,KEEP';
        a2 pin 52 istype 'REG_D,BUFFER,KEEP';
        a3 pin 53 istype 'REG_D,BUFFER,KEEP';
        a4 pin 54 istype 'REG_D,BUFFER,KEEP';
        a5 pin 55 istype 'REG_D,BUFFER,KEEP';
        a6 pin 56 istype 'REG_D,BUFFER,KEEP';
        a7 pin 57 istype 'REG_D,BUFFER,KEEP';
        a8 pin 58 istype 'REG_D,BUFFER,KEEP';
        a9 pin 49 istype 'REG_D,BUFFER,KEEP';
        a10 pin 3 istype 'REG_D,BUFFER,KEEP';
        a11 pin 4 istype 'REG_D,BUFFER,KEEP';
        a12 pin 5 istype 'REG_D,BUFFER,KEEP';
        a13 pin 6 istype 'REG_D,BUFFER,KEEP';
        a14 pin 7 istype 'REG_D,BUFFER,KEEP';
        a15 pin 8 istype 'REG_D,BUFFER,KEEP';
        a16 pin 9 istype 'REG_D,BUFFER,KEEP';
        a17 pin 10 istype 'REG_D,BUFFER,KEEP';
        a18 pin 11 istype 'REG_D,BUFFER,KEEP';
        a19 pin 12 istype 'REG_D,BUFFER,KEEP';

        areg = [a19,a18,a17,a16,a15,a14,a13,a12,a11,a10,a9,a8,a7,a6,a5,a4,a3,a2,a1,a0];

"Values for states
```

```
S0 = 0;
S1 = 1;
S2 = 2;
S3 = 3;
S4 = 4;
S5 = 5;
S6 = 6;
S7 = 7;
S8 = 8;
S9 = 9;
S10 = 10;
S11 = 11;
S12 = 12;
S13 = 13;
S14 = 14;
S15 = 15;
S16 = 16;
S17 = 17;
S18 = 18;


"Equations for outputs

equations

    !cs_prom = (!B19 & !B18 & !B17 & !B16 & !B15 & !B14 & !AS);
    !cs_ram_proc = (B19 & !AS);
    !cs_ram = (!cs_ram_proc # !cs_ram_state);
    le_par = (!B19 & !B18 & !B17 & !B16 & !B15 & !B14 & B13 & B12 & !AS);
    !oe_prom = (!DS & RW);
    !oe_ram_proc = (!DS & RW);
    !oe_ram = (!oe_ram_proc # !oe_ram_state);
    !we_ram_proc = (!DS & !RW);
    !we_ram = (!we_ram_state # !we_ram_proc);
    !DTACK = (!cs_prom # !cs_ram # le_par);
    oe_par = 0;

"Equation for state diagram

    [q4,q3,q2,q1,q0].clk = clock20;

    [a19,a18,a17,a16,a15,a14,a13,a12,a11,a10,a9,a8,a7,a6,a5,a4,a3,a2,a1,a0].clk = clock20;

    [a19,a18,a17,a16,a15,a14,a13,a12,a11,a10,a9,a8,a7,a6,a5,a4,a3,a2,a1,a0].oe = oe_abus;

state_diagram sreg;

    State S0:

        oe_par2 = 1;
        le_par2 = 0;
        BR = 1;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 1;
        oe_addr = 0;
        if (!start) then S11;
        else S0;

    State S1:

        oe_par2 = 1;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 1;
        oe_addr = 0;
        areg = ^h90000;
        goto S2;

    State S2:
```

```
        oe_par2 = 1;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 1;
        oe_addr = 0;
        areg = ^h90000;
        if (!BG & !vsync) then S10;
        else S2;

State S3:
        oe_par2 = 0;
        le_par2 = 1;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 0;
        areg = areg;
        oe_addr = 1;
        if (!vsync) then S16;
        else if (hsync) then S4;
        else S3;

State S4:

        oe_par2 = 0;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 0;
        we_ram_state = 0;
        cs_AD = 0;
        areg = areg;
        oe_addr = 1;
        if (!vsync) then S16;
        else if (!hsync) then S3;
        else S5;

State S5:
        oe_par2 = 0;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 0;
        we_ram_state = 0;
        cs_AD = 0;
        areg = areg;
        oe_addr = 1;
        if (!vsync) then S16;
        else if (!hsync) then S3;
        else S6;


State S6:
        oe_par2 = 0;
        le_par2 = 1;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 0;
        areg = areg + 1;
        oe_addr = 1;
        if (!vsync) then S16;
        else if (!hsync) then S3;
        else S4;

State S7:
        oe_par2 = 0;
        le_par2 = 1;
        BR = 0;
```

```
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 0;
        areg = ^hA0000;
        oe_addr = 1;
        if (vsync) then S12;
        else S7;


State S8:
        oe_par2 = 1;
        le_par2 = 0;
        BR = 1;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 1;
        areg = ^h90000;
        oe_addr = 0;
        goto S9;

State S9:
        oe_par2 = 1;
        le_par2 = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        BR = 1;
        cs_AD = 1;
        areg = ^h90000;
        oe_addr = 0;
        if (!ready) then S11;
        else S9;

State S10:
        oe_par2 = 0;
        le_par2 = 1;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 0;
        areg = areg;
        oe_addr = 1;
        if (vsync) then S4;
        else S10;

State S11:
        oe_par2 = 1;
        le_par2 = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        BR = 1;
        cs_AD = 1;
        areg = ^h90000;
        oe_addr = 0;
        if (start) then S1;
        else S11;

State S12:
        oe_par2 = 0;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 0;
        we_ram_state = 0;
        cs_AD = 0;
        areg = areg;
        oe_addr = 1;
        if (!vsync) then S8;
        else if (!hsync) then S15;
        else S13;
```

```
State S13:
        oe_par2 = 0;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 0;
        we_ram_state = 0;
        cs_AD = 0;
        areg = areg;
        oe_addr = 1;
        if (!vsync) then S8;
        else if (!hsync) then S15;
        else S14;


State S14:
        oe_par2 = 0;
        le_par2 = 1;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 0;
        areg = areg + 1;
        oe_addr = 1;
        if (!vsync) then S8;
        else if (!hsync) then S15;
        else S12;

State S15:
        oe_par2 = 0;
        le_par2 = 1;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 0;
        areg = areg;
        oe_addr = 1;
        if (!vsync) then S8;
        else if (hsync) then S12;
        else S15;

State S16:
        oe_par2 = 1;
        le_par2 = 0;
        BR = 1;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 1;
        areg = areg;
        oe_addr = 0;
        if (ready) then S17;
        else S16;

State S17:
        oe_par2 = 1;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
        we_ram_state = 1;
        cs_AD = 1;
        areg = ^hA0000;
        oe_addr = 0;
        goto S18;

State S18:
        oe_par2 = 1;
        le_par2 = 0;
        BR = 0;
        oe_ram_state = 1;
        cs_ram_state = 1;
```

```
                we_ram_state = 1;
                cs_AD = 1;
                areg = ^hA0000;
                oe_addr = 0;
                if (!BG & !vsync) then S7;
                else S18;

END
```

Appendix C

```
unsigned short int *parallell,*bild_pek1,*bild_pek2;

int left,right,straight = 0;

void slask(){
	return;
}

void drive(int k, int data)
{
	int i,j;
	if (k==0){
		for (j=0; j<150; j=j+1){
			for (i=0; i<500; i=i+1){
				*parallell = data + 0x9;
			}
			for (i=0; i<1000; i=i+1){
				*parallell = data;
			}
		}
	}
	else if (k==-1){
		for (j=0; j<30; j=j+1){
			for (i=0; i<500; i=i+1){
				*parallell = data + 0xA;
			}
			for (i=0; i<1000; i=i+1){
				*parallell = data;
			}
		}
	}
	else if (k==1){
		for (j=0; j<30; j=j+1){
			for (i=0; i<500; i=i+1){
				*parallell = data + 0x5;
			}
			for (i=0; i<1000; i=i+1){
				*parallell = data;
			}
		}
	}
	else if (k==-2){
		for (j=0; j<150; j=j+1){
			for (i=0; i<500; i=i+1){
				*parallell = data + 0x6;
			}
			for (i=0; i<1000; i=i+1){
				*parallell = data;
			}
		}
	}
	return;
}

int analyze(){
	int i,j;
	int pixel1,pixel2;
	left = 0;
	right = 0;
	straight = 0;
	bild_pek1 = (unsigned short int *) 0x090000;
	bild_pek2 = (unsigned short int *) 0x0A0000;
	for (i=0; i<380; i=i+1){
		for (j=0; j<0x42; j=j+1){
			bild_pek1 = 0x90000 + 0xA0*i + j;
			bild_pek2 = 0xA0000 + 0xA0*i + j;
			pixel1=*bild_pek1 & 0xF8;
			pixel2=*bild_pek2 & 0xF8;
			if (pixel1 != pixel2)
				left = left +1;
		}
		for (j=0x42; j<0x5F; j=j+1){
```

```
                        bild_pek1 = 0x90042 + 0xA0*i + j;
                        bild_pek2 = 0xA0042 + 0xA0*i + j;
                        pixel1=*bild_pek1 & 0xF8;
                        pixel2=*bild_pek2 & 0xF8;
                        if (pixel1 != pixel2)
                                straight = straight +1;
                }
                for (j=0x5F; j<0xA3; j=j+1){
                        bild_pek1 = 0x9005F + 0xA0*i + j;
                        bild_pek2 = 0xA005F + 0xA0*i + j;
                        pixel1=*bild_pek1 & 0xF8;
                        pixel2=*bild_pek2 & 0xF8;
                        if (pixel1 != pixel2)
                                right = right +1;
                }
        }

        if (left > right){
                if (left > straight)
                        return -1;
                else return 0;
        }
        else if (right > straight)
                return 1;
        else if (straight > 1)
                return 0;
        else return -2;
}

main()
{
        int par_data;
        int i;
        parallell = (unsigned short int *) 0x003000;
        bild_pek1 = (unsigned short int *) 0x090000;
        bild_pek2 = (unsigned short int *) 0x0A0000;
        *parallell = 0;
        par_data = 0;

        greta:

        *parallell = 0x80;
        par_data = 0x80;
        slask();
        for (i=0; i<10000; i=i+1){
        }
        *parallell = 0xC0;
        par_data = 0xC0;
        slask();

        drive(analyze(), par_data);
        *parallell = 0;
        par_data = 0;
        goto greta;     /* evig loop */
}
```