Lunds tekniska högskola
Institutionen för informationsteknologi

Rapport i
digitala projekt, lp 2 ht 2003

DMX-styrning

Andreas Källman, d00aka
Per-Henrik Persson, d00pp

2003-12-09

Abstract

In the course Digitala projekt at LTH we designed a device to control intelligent lightning effects over the DMX interface. The goal was to build a good working prototype as functional as a commercial DMX scanner controller. It should be possible to program different sequences on the controller via an easy to understand user interface, and then run these sequences on command. The sequences are stored in a non-volatile memory. The design includes AD-converters, a LCD display and an FPGA. The FPGA is supposed to take care of all the time critical tasks and be controlled by the CPU via its own instruction set.

Innehållsförteckning

1 INLEDNING 4

2 TEORI 4

2.1 DMX-GRÄNSSNITTET 4
2.2 UTVECKLINGSSYSTEMET IT-68 5

3 KONSTRUKTION 5

3.1 HÅRDVARA 5
3.1.1 Processor – MC68008 6
3.1.2 Programmerbar logik - PALCE22V10 6
3.1.3 A/D omvandlare – ADC0809 6
3.1.4 EPROM minne – 27C64 6
3.1.5 RAM-minne – 6264 7
3.1.6 D-vippor – 74HC374 7
3.1.7 Display 7
3.1.8 Programmerbar logik – Xilinx XC4010E 7
3.1.9 EEPROM –29EE010 7
3.2 MONTERING 8
3.3 TESTNING 8

4 MJUKVARA 8

4.1 PROGRAMKOD 8
4.1.1 LCD.h 8
4.1.2 AD.h 9
4.1.3 Test.c 9
4.1.4 DMX.h 10
4.2 FPGA 10

5 RESULTAT 11

6 SLUTSATS 11

7 REFERENSER 11

APPENDIX 1 13

APPENDIX 2 14

APPENDIX 3 22

APPENDIX 4 36

APPENDIX 5 37

APPENDIX 6 40

APPENDIX 7 41

1 Inledning
På allt från stora teaterscener till små trånga dansgolv förekommer olika typer av styrbara ljuseffekter och lampor. Gemensamt för de flesta av dessa ljuskällor är att de går att styra via en seriell databuss (DMX-buss). Man kan via denna buss ställa saker såsom ljusstyrka, färger, position etc. På dansgolv används ofta sk. scannrar. På en scanner går det oftast att ställa färg och mönster på ljusstrålen genom att rotera på olika färghjul. Dessutom kan ljusstrålen flyttas runt i rummet med en spegel.

Syftet med detta projektarbetet var att bygga en fullt fungerande och programmerbar styrning för scannrar. Kostnaden för en färdig sådan ligger på några tusenlappar och uppåt. Funktionen är sådan att man kan programmera in sekvenser som sedan går att spela upp. En sekvens består av ett antal steg i vilka de olika scannrarnas datavärden sparas. Tiden mellan de olika stegen är givetvis ställbar.

För att underlätta programmering brukar styrningarna ha ett antal knappar för att välja vilka scannrar som ska innefattas i ett visst steg i en sekvens, ett antal analoga skjutpotentiometrar för att ställa färg, ljusstyrka etc, samt en joystick för x/y-position av spegeln.

I den första delen av rapporten behandlas lite bakomliggande teori, därefter följer kravspecifikation och val av komponenter. Efter det följer en beskrivning av programkoden till projektet, resultat samt slutsatser.

2 Teori
Det finns mycket bakomliggande teori för ett sådant här projekt. En stor del av informationen om digital konstruktion runt en processor har vi hämtat ur den bok institutionen tillhandahåller till kursen, ”it-68 Utvecklingssystem för MC68008”. Boken går igenom konstruktion med MC68008-processorn samt hur man använder utvecklingssystemet it-68. Mer om detta i avsnitt 2.2. Utöver detta kan det vara på sin plats med en enkel beskrivning av DMX-gränssnittet.

2.1 DMX-gränssnittet
DMX-gränssnittet är en standard för att styra ljus över en seriell buss. Standarden gäller både den fysiska sammankopplingen och protokollet som pratas över bussen. Standarden som den ser ut idag antogs 1990. Tanken är att man på ett enkelt sätt ska kunna styra alla typer av lampor och ljuseffekter. Protokollet har stöd för 512 kanaler, vilket betyder att 512 olika parametrar går att styra. Varje kanal kan anta 256 värden. Exempelvis kan detta användas för att styra ljusstyrkan på i 256 steg på 512 olika lampor på en teaterscen.

Eftersom det är ett seriellt gränssnitt kan ett antal mottagare kopplas samman i serie. På en DMX-buss finns det endast en sändare. I vårt fall scannerstyrningen vi ska bygga. Varje mottagare har en ställbar adress (kanal) som den lyssnar på. Oftast behöver en mottagare flera kanaler för att styra flera olika funktioner. Adressen som då ställs in är grundadressen, de andra kanalerna ligger på adresserna efter denna.

Elektriskt sett är DMX-gränssnittet samma som seriegränssnittet RS485. Signalerna är balanserade för att undvika störningar. Kablaget är partvinnad kabel, kontaktdonet

vanligtvis 3-polig XLR även om standarden säger 5-polig XLR. Hastigheten är 250kbit/s.

Principen för protokollet är att man skickar ut lite startinformation för att få alla enheter på bussen att lyssna, sedan skickas värdena för var och en av de 512 kanalerna ut efter varandra. Man skickar aldrig ut vilken kanal ett visst värde gäller för, i stället lyssnar en mottagare efter startinformationen, och räknar sedan byte efter byte till just ”sin” adress. Startinformationen och datan (tillsammans ett paket) skickas kontinuerligt ut på bussen.

Eftersom överföringshastigheten är 250kbit/s blir varje bit 4µs lång. Ett paket ser ut som följer:

Först skickas 22 låga bitar, därefter två höga bitar. Efter detta följer ett antal dataramar med samma utseende. Varje ram består av en låg startbit, åtta databitar och sist två höga stoppbitar.
Den första ramen som skickas har alltid datavärdet 0, alltså åtta låga bitar. Därefter följer 512 ramar med värdena för de 512 kanalerna. Enligt standarden behöver man inte skicka värdet för alla 512 kanaler, utan man kan när som helst avbryta och börja om med ett nytt paket.

Utöver detta kan man även lägga in paus-bitar på vissa givna ställen. Det använder vi oss dock inte av i vårt projekt.

2.2 Utvecklingssystemet it-68

Till hjälp vid utveckling med Motorola 68008-processorn, finns IT-institutionens utvecklingssystem ”it-68”. Systemet emulerar en 68008-processor och styrs med en terminal över seriegränssnitt. Systemet innehåller det mesta för att kunna debugga hårdvara såväl som mjukvara. Dessutom kan programkod laddas upp till emulatorn och köras därifrån i stället för att brännas till EPROM.

3 Konstruktion

Innan vi satte igång med konstruktionsarbetet skrev vi en enkel kravspecifikation. Denna hittar du i appendix 1.

3.1 Hårdvara

Om man utgår från hur en kommersiell scanner-styrning fungerar är det tämligen lätt att resonera fram vad man behöver i form av hårdvara.

För att kunna handha styrningen behövs in-enheter. Dels ett antal knappar, dels ett antal analoga skjutpotentiometrar. Till dessa behövs givetvis AD-omvandlare. För användaren behövs det någon form av visuell ut-enhet, i detta fallet en LCD-display.

Förutom detta behövs elektronik för att skicka ut seriell data till DMX-bussen, samt någon form av hårdvara för att ta hand om de tidskritiska bitarna när sekvenser spelas upp. Inprogrammerade sekvenser måste kunna sparas så att de finns kvar även om strömmen till styrningen försvinner.

3.1.1 Processor – MC68008
De två processorer som tillhandahålls är enchipsdatorn M68HC11 och MC68008. Vi valde Motorola 68008 därför att en riktig processor är mer flexibel än en enchipsdator.

MC68008 är med sin 8-bitars databuss en enklare medlem i Motorolas 68000-familj. Adressbussen är 20 bitar vilket gör att 1Mb minne kan adresseras.

Processorn kan hantera två typer av avbrott. Den ena typen fungerar så att en enhet genererar ett avbrott, sedan frågar processorn på databussen vilken enhet som genererat avbrottet och tar hand om det som ska ske beroende på vilken enhet det är. Den andra typen av avbrott är autovector-avbrott som, när det sker en avbrottsförfrågan, hoppar till en förinställd adress i programminnet. Den första typen av avbrott kräver kringenheter med en hel del intelligens, den andra typen betydligt mycket enklare. Vi tänkte använda autovector-avbrott i vår konstruktion.

3.1.2 Programmerbar logik - PALCE22V10
För att binda samman de olika kringkomponenterna med varandra och processorn valde vi att använda två Lattice PALCE22V10. Den programmerade logiken används för att generera de signaler som behövs för att processorn och kringkomponenterna ska kunna kommunicera med varandra. Den ena PAL:en används främst för att generera read/write-signaler samt chip select-signaler utifrån processorns adressbuss. Minneskarta finns i appendix 6. Eftersom processorn arbetar med asynkron i/o måste acknowledge-signaler till processorn genereras varje gång den försöker läsa eller skriva till någon enhet. Denna signal generas av PAL:en.

Den andra PAL:en används för avbrottshanteringen. När ett avbrott genereras av någon kringkomponent förväntar sig processorn att få vissa insignaler. Dessa står den andra PAL:en för. Programkod för programmering av PAL:arna finns i appendix 4.

3.1.3 A/D omvandlare – ADC0809
För att slippa flera AD-omvandlare i separata kapslar valde vi ADC0809, som är en 8 bitars A/D omvandlare med 8 stycken kanaler. Då DMX-gränssnittet hanterar värden mellan 0 och 255 passar en 8-bitars AD-omvandlare perfekt. En AD-omvandling går till så att omvandlaren får in en adress (vilken av de åtta kanalerna som ska omvandlas) samt en startsignal. En kort tid senare genererar AD-omvandlaren en klarsignal och det omvandlade värdet kan hämtas på databussen. Tanken är att använda klarsignalen för att generera ett avbrott till processorn. Detta är egentligen lite onödigt eftersom processorn inte kommer vara mer upptagen än att man kan göra AD-omvandligen när man har tid över.

Den lilla mängd logik som behövs för att omvandla processorns läs- och skrivsignaler till start och lässignaler som AD-omvandlaren förstår används den första PAL:en.

3.1.4 EPROM minne – 27C64
För att lagra programkod till processorn behövs någon form av statiskt minne. Vi valde ett EPROM på 8kb. Dessa 8kb är mer än väl för vår konstruktion. Eftersom EPROM:et endast är raderbart med UV-ljus, använder man under utvecklingsfasen det i utvecklingssystemet inbyggda programminnet.

3.1.5 RAM-minne – 6264
Ingen processor utan RAM-minne. I RAM-minnet lagras programvariabler samt programstack. Till detta räcker 8kb minne mer än väl. RAM-minnet såväl som EPROM:et är vanliga minnen med styrsignalerna chip select, read och write vilket gör att de bara är att koppla på PAL:en samt adress- och databuss.

3.1.6 D-vippor – 74HC374
För tangentbordsavkodning används ofta en separat tangentbordsavkodare som kodar av en knappsats i matrisform. Vanligtvis använder man sig också av avbrott för att signalera till processorn att en knapp tryckts ner. Eftersom processorn i vår konstruktion inte kommer göra något direkt tidskritiskt tyckte vi att det räckte med att polla knapparna med jämna mellanrum. För detta ändamålet valde vi att använda oss en uppsättning vanliga D-vippor för att klocka in tangentbordstryckningar på processorns databuss. I 74-serien hittade vi 74HC374, en kapsel med 8 separata D-vippor.

3.1.7 Display
För att komma enkelt undan och ändå tydligt kunna visa status för olika operationer på styrningen valde vi en alfanumerisk LCD-display med 4x20-tecken. Visst hade det varit tuffare med en grafisk display men det hade resulterat i betydligt mycket mer programkod. Displayen kommunicerar via ett synkront gränssnitt, men med lite tricks och logik i PAL:en går den att skriva till precis som vilken annan enhet som helst på databussen. Eftersom displayen är förhållandevis långsam på att skriva ut tecken kan man efter att ha skrivit till den läsa displayens status för att få reda på när den är klar och redo att ta emot ny data.

3.1.8 Programmerbar logik – Xilinx XC4010E
För att generera den seriella DMX-signalen samt lösa de tidskritiska bitarna vid uppspelning av inprogrammerade sekvenser tänkte vi använda en FPGA. Att vi inte valde en färdig UART samt en eventuell realtidsklocka beror mest på att vi tyckte det kunde vara roligt att programmera en FPGA. Tanken är att FPGA:n ska ha en egen instruktionsuppsättning som processorn använder. FPGA:n ska sedan själv hantera inläsning av sekvenser för uppspelning från ett EEPROM och generera den seriella DMX-signalen.

Vi valde Xilinx-kretsen eftersom den har gott om grindekvivalenter, många i/o-pinnar samt 1600 byte inbyggt RAM-minne. Enda nackdelen med kretsen är att den är kapslad i PLCC84-kapsel i stället för en vanlig DIL-kapsel. Detta gör monteringen svårare. För att slippa ta ur kretsen ur sin sockel varje gång den ska programmeras använder vi oss av en speciell programmerare som kopplas in till några av pinnarna på kresen.

3.1.9 EEPROM –29EE010
Sist men inte minst behövde vi någon form av minne att spara inprogrammerade programsekvenser i. Eftersom dessa måste finnas kvar även när strömmen bryts valde vi ett EEPROM på 512kb. Detta minne fungerar i övrigt som precis som RAM- och programminnet. Eftersom processorn inte ska kunna kommunicera direkt med detta minnet, ansluts det inte till den gemensamma adress- och databussen, utan i stället direkt till i/o-pinnar på FPGA:n.

Sist men inte minst behövs det lite kringkomponenter för att få alla saker att fungera tillsammans. Som oscillator valde vi en 8MHz kristall.

LCD-displayen behöver en negativ spänning för att öka kontrasten. Detta löste vi med en enkel spänningsomvandlare (ICL7660).

För att kunna få ut seriella signaler som rent elektriskt följer DMX-standarden behövs en enkel seriell drivkrets som kan skapa balanserade signaler. Valet föll på SN75176, en enkel sändare/mottagare för RS485.

3.2 Montering

All hårdvara monterades på ett smidigt färdig experimentkort. För enkelhetens skull byggdes prototypen med hjälp av virning, endast det absolut nödvändiga löddes.

3.3 Testning

När alla komponenter var monterade var det dags för testning. Det enklaste var att addera komponent för komponent för att lätt kunna felsöka och testa dem var för sig. Det första som måste fungera är naturligtvis processorn. Därför monterades sockel för it-68-systemet samt RAM-minnet. För att processorn ska kunna kommunicera med minna behövs några av de signaler som den ena PAL-kretsen genererar. Därför programmerades och monterades även denna.

När detta fungerade kopplades den interna oscillatorn på it-68-kortet bort till förmån för den egna klockan på vårt experimentkort. Därefter kopplades displayen, knapparna och AD-omvandlaren in en efter en. Små justeringar var nödvändiga, bl a för att vi först hade bestämt oss för att endast kunna skriva till displayen, inte läsa från den. En felprogrammerad PAL ställde också till det.

4 Mjukvara

4.1 Programkod

Eftersom all hårdvara fungerade utan några direkta problem kunde vi tidigt börja skriva lite testprogram i C. All programkod finns i appendix 2.

4.1.1 LCD.h

Detta bibliotek innehåller funktioner för att styra lcd displayen. Det finns funktioner för att initialisera displayen, förflytta pekaren till en viss position på skärmen där positionen till vilken man vill flytta sig skrivs som (x,y) och är absolut, rensa displayen, ge kommandon till displayen, fördröja programmet med en viss tidsenhet skriva ut tecken och skriva ut strängar. De flesta av dessa funktioner är enkla och intuitiva.

Tyvärr fick vi aldrig funktionen för att skriva ut strängar att fungera ordentligt. Den fungerade ibland, medan vid andra tillfällen medförde den att programmet låste sig så fort det kördes igång och alarm lampan tändes. Efter många tappra försök att åtgärda situationen gav vi till slut upp och fann oss i att putstring var opålitlig. Vi blev därmed tvungna att använda oss av putchar istället vilket innebar att koden blev svåröverskådlig.

4.1.2 AD.h

Som namnet antyder innehåller denna modul funktioner som är kopplade till A/D omvandlaren. Det finns tre funktioner i denna klass. AD_start, AD_read och AD_getvalues.

Start funktionen tar som inparameter ett heltal som representerar vilken av de 8 kanalerna som användaren vill omvandla till ett digitalt värde. Omvandlingen utav värdet påbörjas sedan, funktionen tar ingen hänsyn till om omvandlingen hinner slutföras eller inte innan värdet läses av från kanalen.

Read funktionen tar, precis som start funktionen ett heltal som inparameter vilket representerar vilken av de 8 kanalerna som värdet skall läsas från. Värdet från omvandligen hämtas från den specificerade kanalen.

Getvalues är den, av dessa tre funktioner, som är menad att användas från andra moduler. Getvalues kombinerar funktionaliteten av start funktionen och read funktionen. Funktionen loopar igenom alla A/D kanaler, för varje kanal startar den en A/D omvandling, väntar en viss tidsenhet och läser sedan av värdet från kanalen. De värden som läses av jämförs med föregående värde som funnits på kanalen. Om värdet har ändrats mer än tre steg i positiv eller negativ riktning sätts en kanalspecifik bit i första elementet i den vektor där värdena för A/D kanalerna finns lagrade. Anledningen till att vi infört restriktioner på vad som anses som en tillräcklig förändring av reglarna är att de ofta står och varierar mellan 2 stycken värden. Genom att införa denna restriktion undviker vi därmed onödiga uppdateringar utav displayen.

4.1.3 Test.c

Modulen test innehåller vårt huvudprogram. Dess huvudsakliga uppgift är att agera som ett gränssnitt ut mot användaren.

Programmet inleds genom att LCD displayen initialiseras och grundmenyn skrivs ut. Sedan ligger programmet och gör A/D omvandlingar med jämna mellanrum för att se om något av reglagen har ändrat position. Första elementet i vektorn för AD kanalerna gås igenom bitvis för att se vilken av kanalerna som förändrats. Om något av värdena har förändrats sker en uppdatering av skärmen ifall den kanalen för tillfället är synlig. Programmet väntar även på att någon av de 8 knapparna skall tryckas ned. Då detta sker inträffar olika saker beroende på vilken av knapparna som tryckts ned och i vilket tillstånd menyn för tillfället är. Menysystemet är uppbyggt av en mängd switch och if/else satser.

För att motverka kontaktstuds från knapparna måste användaren släppa den knapp han för tillfället tryckt ned innan han kan trycka in en ny knapp. Gränssnittet gör bara en förändring per knapp tryck.

I gränssnittet kan man styra vilka scannrar man vill skicka ut signaler på samt vilket värde som skall skickas ut på dessa scannrar, för tillfället antar vi att varje scanner har 8 kanaler. På grund av skärmens storlek kan endast 4 av dessa kanalers värden visas åt gången. Kanalernas värden kan stegas upp/ner med hjälp utav knappar, om man vill göra stora förändringar i dessa värden kan reglagen användas. Reglagen är direkt knutna till respektive kanalvärde och kan därmed ändras oavsett var i menyn man

befinner sig.

I menyn finns en möjlighet att gå in i ett läge där man skall kunna modifiera sekvenser. Sekvenshantering är dock på grund utav tidsbrist inte implementerad. Gränssnittet har även en knapp som är dedikerad till att sända ut dmx signaler.

4.1.4 DMX.h

Denna klass innehåller en funktion som namngivits till DMX_send. Send funktionen maskar ut till vilka scannrar den skall skicka ut nya DMX signaler och skickar sedan värdena för var och en av de 8 kanalerna för varje scanner till vår FPGA.

4.2 FPGA

Vår första konstruktion såg inte ut som den som beskrivits tidigare i rapporten. I stället för en Xilinx-krets valde vi då en FPGA från Lattice. Denna skulle i sin tur inte bara kunna kommunicera med ett EEPROM utan även också ett RAM-minne i vilket de 512 DMX-kanalernas värden hela tiden skulle finnas lagrade.

Efter att ha programmerat funktioner för att generera den seriella bitströmmen samt initialisera RAM-minnet stötte vi på problem. Logiken fick inte plats i Lattice-kretsen och denna fick därför bytas ut mot den större kretsen från Xilinx. I och med detta kunde vi skippa RAM-minnet eftersom Xilinx-kretsen innehåller tillräckligt med internt minne.

I EEPROM:et ska programsekvenser lagras i något lämpligt format. Exakt hur detta format ska se ut är inte helt klart eftersom sekvensprogrammering inte är färdigimplementerat. Tanken är att på något smart sätt nyttja minnet så bra som möjligt. Givetvis ska bara förändringar av kanalvärdena sparas. Ett smart sätt är att i början av minnet på vissa fasta adresser spara undan referenser till var i minnet första sekvenssteget finns undansparat. Om man sedan för varje steg sparar information om vilka kanaler som ändrat värde samt det nya värdet. Dessutom måste tiden till när nästa steg ska skickas ut på DMX-bussen sparas undan.

Gör man detta på ett smart sätt kan man även tänka sig att en redan sparad programsekvens längd är dynamisk genom att antingen flytta efterföljande sekvenser eller använda sig av någon form av adresspekare så att en sekvens kan fragmenteras. Processorn i systemet är tämligen kraftfull i sammanhanget. Detta i kombination med genomtänkta minnesinstruktioner i FPGA:n möjlighör ett ganska avancerat ”filsystem” för programminnet.

Vid sekvensprogrammering ska processorn generera sekvenskod enligt ovanstående beskrivning. Denna kod ska sedan skrivas till EEPROM:et via instruktioner till FPGA:n. För att köra en sekvens, skickar man en instruktion till FPGA:n som sedan börjar läsa sekvensen och skicka ut datan på DMX-bussen. FPGA:n ska alltså ha hand om hela sekvensuppspelningen, vilket gör att inget tidskritiskt sköts av processorn.

Programmering av FPGA:n sker i utvecklingsverktygen Ease och Eale och programmeras i VHDL. Till hjälp under utvecklingen används simuleringsverktyget Modelsim.

Som tidigare nämnt ska FPGA:n styras av processorn via en egen instruktonsuppsättning. För att snabba upp läsning och skrivning till EEPROM:et samt DMX-bussen ska det finnas instruktoner för att kunna skriva stora block i taget. Detta blir mest effektivt om längden på instruktionerna som skickas till FPGA:n är dynamisk, d v s instruktioner varierar i längd från två till fyra byte. Tabell över instruktionsuppsättningen finns i appendix 5. VHDL-koden finns i appendix 3.

5 Resultat

Tyvärr har vi inte hunnit i närheten så långt med projektet som vi hade velat. Vägen till att uppfylla kravspecifikationen är mycket lång och de program vi skrivit fungerar endast som ett enkelt test av hårdvaran. Dock bevisar den att våra tankar och vår lösning fungerar rent praktiskt! Anledningen till att vi inte blivit klara är tidsbrist. Bytet av FPGA-modell tagit mycket extra tid. Även om mycket tid har lagts på projektet så räckte detta inte.

Programvaran, skriven i C, är mycket enkel. Det finns ännu inget stöd för sekvenser. En stor del utav koden är ägnad till att ge användaren ett någorlunda lätt hanterligt gränssnitt att jobba mot. Det finns stöd för att skicka ut dmx signaler och användaren kan indirekt styra vad för slags signaler som skall skickas iväg.

Funktionerna i FPGA:n är mycket begränsade. Den enda av den tänkta funktionaliteten som är implementerad är möjligheten att skicka ut värden på DMX-bussen. Med andra ord finns det inget stöd för att läsa eller skriva till EEPROM:et eller programmera och köra sekvenser därifrån. Detta betyder att det inte finns något färdigt format för lagring av programsekvenser.

Vi använder oss inte heller av avbrott. Rent hårdvarumässigt är det implementerat, men testprogramvaran vi skrivit utnyttjar inte funktionaliteten.

6 Slutsats

Projektet har varit mycket lärorikt, även om vi inte lyckades nå upp till vår kravspecifikation. Kursen innehåller inslag av det mesta, allt från att läsa datablad till att programmera C. Eftersom projektet är styrt väldigt lite, finns det oändliga möjligheter att hitta på egna lösningar och genomföra sina egna önskemål. Även handledningen har varit mycket bra.

Hade mer tid funnits skulle vi jobbat mot att uppfylla kravspecifikationerna. Några veckor till och vi skulle förmodligen ha gjort det.

Tyvärr sprang priset på vårt projekt iväg rätt snabbt, hade komponentpriserna legat på runt en tusenlapp hade det varit intressant att bygga vidare på projektet för att få en fullt fungerande styrning.

7 Referenser

DMX booster - http://users.skynet.be/kristofnys/booster.htm

DMX512 PAGE FROM Ujjal…details of DMX512 - http://www.dmx512-online.com/

Appendix 1

Kravspecifikation:

Systemet ska kunna:

- Generera giltiga DMX-signaler
- Användargränssnitt med analoga reglar och display
- Ändra värden på DMX-kanaler
- Enkelt och genomtänkt användargränssnitt
- Programmera sekvenser indelade i steg med ställbar stegtid
- Spara sekvenser i ett icke-flyktigt minne
- Köra sparade sekvenser

Appendix 2

C-kod:

```
--------------------------------Test.c--------------------------------
#pragma separate lcd
short int lcd;
#pragma separate lcddata
short int lcddata;
#pragma separate dmx
short int dmx;
#pragma separate eeprom
short int eeprom;
#pragma separate buttons
short int buttons;
#pragma separate memif
short int memif;
#pragma separate ad1
short int ad1;
#pragma separate ad2
short int ad2;
#include <string.h>
#include <stdio.h>
#include "const.h"
unsigned short int ad[AD_CHANNELS+1];
unsigned short int value[8];
#include "lcd.h"
#include "ad.h"
#include "dmx.h"
void initiateLCD();
void updatevalue(unsigned short int nr);

main() {
  unsigned short int status, temp, scanner,nrvalue;
  unsigned short int *valuep;
  signed short int cursor;
  int i;
  for(i=0;i<8;i++)
    value[i]=255;
  status = scanner = cursor =nrvalue = 0;
  valuep=&value[0];
  initiateLCD();
  while(1) {
    AD_getvalues();
    if (ad[0] != 0) {
      int j;
      for(j=0;j<8;j++){
      if(ad[0]&1)
        {
          value[j]=ad[j+1];
          if((j-nrvalue*4)>=0){
            LCD_gotoxy(5+(j-nrvalue*4)*4, 2);
            updatevalue(j);
            LCD_gotoxy(5+(j-nrvalue*4)*4, 2);
          }
        }
      ad[0] = ad[0]>>1;
      }
    }
    temp = buttons;
```

```
if (temp != 0) {
  if(status == 0) {
  switch (temp) {
  case 2:
    status = 1;
    LCD_gotoxy(5, 3);
    LCD_putchar('S');
    LCD_putchar('c');
    LCD_putchar('a');
    LCD_gotoxy(5, 1);
    cursor=0;
    break;
  case 4:
    LCD_gotoxy(9, 3);
          LCD_putchar('S');
    LCD_putchar('c');
    LCD_putchar('a');
    LCD_gotoxy(3, 2);
    status = 2;
    cursor=0;
    break;
  case 8:
    DMX_send(valuep,scanner);
    break;
  case 0x10:
    cursor=(cursor +1)%8;
    LCD_gotoxy(5+cursor*2,0);
    break;
  case 0x20:
    cursor=(cursor - 1);
    if(cursor == -1)
      cursor =7;
    LCD_gotoxy(5+cursor*2,0);
    break;
  case 0x40:
    LCD_gotoxy(5+cursor*2,0);
    scanner=scanner | (1<<cursor);
    LCD_putchar((0x61+cursor));
    LCD_gotoxy(5+cursor*2,0);
    break;
  case 0x80:
    LCD_gotoxy(5+cursor*2,0);
    scanner=scanner & (~(1<<cursor));
    LCD_putchar((0x41+cursor));
    LCD_gotoxy(5+cursor*2,0);
    break;
  default:
    break;
  }
  }else if(status ==1)
  {
    switch (temp) {
    case 2:
      status = 0;
      LCD_gotoxy(5, 3);
      LCD_putchar('S');
      LCD_putchar('e');
      LCD_putchar('q');
      LCD_gotoxy(5, 0);
      break;
    case 4:
```

```
  status = 2;
  LCD_gotoxy(5, 3);
  LCD_putchar('S');
  LCD_putchar('e');
  LCD_putchar('q');
  LCD_gotoxy(9, 3);
  LCD_putchar('S');
  LCD_putchar('c');
  LCD_putchar('a');
  LCD_gotoxy(3, 2);
        break;
case 8:
  DMX_send(valuep,scanner);
  break;
}
    }else if (status == 2) {
switch (temp) {
case 4:
  status = 0;
  cursor=0;
  LCD_gotoxy(9, 3);
  LCD_putchar('V');
  LCD_putchar('a');
  LCD_putchar('l');
  LCD_gotoxy(5, 0);
  break;
case 2:
  status = 1;
  cursor=0;
  LCD_gotoxy(5, 3);
  LCD_putchar('S');
  LCD_putchar('c');
  LCD_putchar('a');
  LCD_gotoxy(9, 3);
  LCD_putchar('V');
  LCD_putchar('a');
  LCD_putchar('l');
  LCD_gotoxy(5, 1);
  break;
case 8:
  DMX_send(valuep,scanner);
  break;
case 0x10:
  cursor=(cursor +1)%5;
  if(cursor==0)
      LCD_gotoxy(3,2);
  else
    LCD_gotoxy(5+(cursor-1)*4,2);
  break;
case 0x20:
  cursor=(cursor - 1);
  if(cursor == -1)
    cursor =4;
  if(cursor==0)
      LCD_gotoxy(3,2);
  else
    LCD_gotoxy(5+(cursor-1)*4,2);
  break;
case 0x80:
  if(cursor==0)
    {
```

```
            nrvalue=1;
            LCD_gotoxy(3,2);
            LCD_putchar(0x31);
            LCD_gotoxy(5,2);
            updatevalue(4);
            LCD_gotoxy(9,2);
            updatevalue(5);
            LCD_gotoxy(13,2);
            updatevalue(6);
            LCD_gotoxy(17,2);
            updatevalue(7);
            LCD_gotoxy(3,2);
            }
          else{
            value[(nrvalue*4)+cursor-1]=(value[(nrvalue*4)+cursor-
1]+1)%256;
            LCD_gotoxy(5+(cursor-1)*4,2);
            updatevalue((nrvalue*4)+cursor-1);
            LCD_gotoxy(5+(cursor-1)*4,2);
          }
          break;
        case 0x40:
          if(cursor==0)
            {
            nrvalue=0;
            LCD_gotoxy(3,2);
            LCD_putchar(0x30);
            LCD_gotoxy(5,2);
            updatevalue(0);
            LCD_gotoxy(9,2);
            updatevalue(1);
            LCD_gotoxy(13,2);
            updatevalue(2);
            LCD_gotoxy(17,2);
            updatevalue(3);
            LCD_gotoxy(3,2);
            }
          else{
                  if(value[(nrvalue*4)+cursor-1]==0)
            value[(nrvalue*4)+cursor-1]=255;
            else
            value[(nrvalue*4)+cursor-1]=(value[(nrvalue*4)+cursor-1]-
1);
            LCD_gotoxy(5+(cursor-1)*4,2);
            updatevalue((nrvalue*4)+cursor-1);
            LCD_gotoxy(5+(cursor-1)*4,2);
          }
          break;
        }
      }
      while(temp!=0)
      {
        temp = buttons;
      }
    }
  }
}

void updatevalue(unsigned short int nr)
{
   LCD_putchar((0x30+value[nr]/100));
```

```
   LCD_putchar((0x30+value[nr]/10-(value[nr]/100)*10));
   LCD_putchar((0x30+value[nr]-(value[nr]/10)*10));
}

void initiateLCD()
{
LCD_init();
LCD_clear();
LCD_gotoxy(0, 0);
LCD_putstring("scan:a b c d e f g h");
LCD_putchar('v');
LCD_putchar('a');
LCD_putchar('l');
LCD_putchar('0');
LCD_putchar(':');
LCD_putchar('2');
LCD_putchar('5');
LCD_putchar('5');
LCD_putchar(' ');
LCD_putchar('2');
LCD_putchar('5');
LCD_putchar('5');
LCD_putchar(' ');
LCD_putchar('2');
LCD_putchar('5');
LCD_putchar('5');
LCD_putchar(' ');
LCD_putchar('2');
LCD_putchar('5');
LCD_putchar('5');
LCD_putchar('S');
LCD_putchar('e');
LCD_putchar('q');
LCD_putchar(':');
LCD_putchar(' ');
LCD_putchar('1');
LCD_gotoxy(0,3);
LCD_putchar('B');
LCD_putchar('a');
LCD_putchar('c');
LCD_putchar('k');
LCD_putchar(' ');
LCD_putchar('S');
LCD_putchar('e');
LCD_putchar('q');
LCD_putchar(' ');
LCD_putchar('V');
LCD_putchar('a');
LCD_putchar('l');
LCD_putchar(' ');
LCD_putchar('S');
LCD_putchar('e');
LCD_putchar('n');
LCD_putchar('d');
LCD_gotoxy(5,0);
}
```

--------------------------------Test.c--------------------------------

--------------------------------AD.h--------------------------------

```
/*
 ad converter lib
 */
#include "const.h"

void AD_start(short int nr);
unsigned short int AD_read(short int nr);

void AD_start(short int nr) {
    unsigned short int *temp;
    temp = (unsigned short int *) 0x13000 + ((nr - 1) << 2);
    *temp = 0x00;
}

unsigned short int AD_read(short int nr) {
    unsigned short int *temp;
    temp = (unsigned short int *) 0x13000 + ((nr - 1) << 2);
    return *temp;
}

void AD_getvalues() {
    short int i;
    int j;
    ad[0] = 0x00;
    for (i = 1; i <= AD_CHANNELS; i++) {
        unsigned short int temp = ad[i];
        AD_start(i);
        for (j = 0; j < 1000; j++) {}
        ad[i] = AD_read(i);
      if ((temp+2) < ad[i] || (temp-2) > ad[i]){
            ad[0] = (1 << (i - 1)) | ad[0];
        }
    }
}
```

----------AD.h----------

----------Lcd.h----------

```
/* LCD-lib

*/

#define LCDCOMMAND 0x10000
#define LCDDATA 0x10004
void LCD_init();
void LCD_clear();
void LCD_command(char cmd);
void LCD_putchar(char ch);
void LCD_putstring(char *ch);
void LCD_gotoxy(short int x, short int y);
void LCD_wait(int i);

void LCD_init() {
    lcd = 0x30;
    LCD_wait(5000);
    lcd = 0x30;
    LCD_wait(100);
    lcd = 0x30;
    LCD_wait(100);
    LCD_command(0x38);
```

```
    LCD_command(0x0e);
    LCD_command(0x14);
    LCD_command(0x06);
    LCD_clear();
}

void LCD_gotoxy(short int x, short int y) {
    short int temp = 0;
    switch (y) {
        case 0: temp = 0;
                break;
        case 1: temp = 0x28;
                break;
        case 2: temp = 0x14;
                break;
        case 3: temp = 0x54;
                break;
        default: temp = 0;
                 break;
    }
    LCD_command(0x80 | (temp + x));
}

void LCD_command(char cmd) {
    lcd = cmd;
    while (lcd & 0x80) {}
}

void LCD_clear() {
    LCD_command(0x01);
}

void LCD_putchar(char ch) {
    lcddata = ch;
    while (lcd & 0x80) {}
}

void LCD_putstring(char *ch) {
  while (*ch != 0) {
    LCD_putchar(*ch);
    ch++;
  }

}

void LCD_wait(int i) {
    while (i > 0) {
        i--;
    }
}
```

--------------------------------Lcd.h--------------------------------

--------------------------------Dmx.h--------------------------------

```
#define DMXCOMMAND 0x12000
void DMX_send(char *value, unsigned short int scanner);

void DMX_send(char *value,unsigned short int scanner){
  int j;
  int i;
  for(j=0;j<8;j++){
```

```
    if(scanner&1)
      {
      for(i=0;i<8;i++){
        dmx = *(value+i)|(j<<11)|(i<<8);
      }
      }
    scanner = scanner>>1;
  }
}
```

----------Dmx.h----------

----------Const.h----------

```
#define AD_CHANNELS 2
#define TRUE 1
#define FALSE 0
```

----------Const.h----------

Appendix 3

VHDL-kod:

```
--------------------------------------------------------------------------------
-- This Vhdl file is generated by EASE/HDL from TRANSLOGIC BV,
-- the 'Graphical Systems Design Tool' tool.
--
-- Ease Version 4.1 (Revision 9).
-- Time stamp  : Sun Dec 07 17:01:27 2003.
--
-- Designed by : .
-- Company     : LTH.
-- Design info : .
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Entity declaration of 'inst_ram'.
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity inst_ram is
  port(
    we       : in     std_logic_vector(0 downto 0) ;
    data     : in     std_logic_vector(32 downto 0) ;
    clk      : in     std_logic ;
    data_out : out    std_logic_vector(32 downto 0) ) ;
end  inst_ram ;

--------------------------------------------------------------------------------
-- Architecture 'a0' of 'inst_ram'
--------------------------------------------------------------------------------

architecture a0 of inst_ram is
       signal mem : std_logic_vector(32 downto 0);
begin
       data_out <= mem;
       process (clk, we) begin
              if (rising_edge(clk)) then
                     if (we = "1") then
                            mem <= data;
                     end if;
              end if;
       end process;
end  a0 ; -- of inst_ram

--------------------------------------------------------------------------------
-- Entity declaration of 'dmx_ram'.
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity dmx_ram is
  port(
    we       : in     std_logic_vector(0 downto 0) ;
    data     : in     std_logic_vector(23 downto 0) ;
    clk      : in     std_logic ;
    data_out : out    std_logic_vector(23 downto 0) ) ;
end  dmx_ram ;

--------------------------------------------------------------------------------
-- Architecture 'a0' of 'dmx_ram'
--------------------------------------------------------------------------------

architecture a0 of dmx_ram is
       signal mem : std_logic_vector(23 downto 0);
begin
       data_out <= mem;
       process (clk, we) begin
```

```
		if (rising_edge(clk)) then
			if (we = "1") then
				mem <= data;
			end if;
		end if;
	end process;
end  a0 ; -- of dmx_ram

-------------------------------------------------------------------------------
-- Entity declaration of 'inst_dec'.
-------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity inst_dec is
  port(
    rd            : in     std_logic_vector(0 downto 0) ;
    wr            : in     std_logic_vector(0 downto 0) ;
    ce            : in     std_logic_vector(0 downto 0) ;
    a2            : in     std_logic_vector(0 downto 0) ;
    data          : inout  std_logic_vector(7 downto 0) ;
    next_state    : out    std_logic_vector(6 downto 0) ;
    current_state : in     std_logic_vector(6 downto 0) ;
    dmx_data      : inout  std_logic_vector(7 downto 0) ;
    dmx_ctrl      : out    std_logic_vector(1 downto 0) ;
    reset         : in     std_logic_vector(0 downto 0) ;
    dmx_ram       : out    std_logic_vector(23 downto 0) ;
    dmx_ram_wr    : out    std_logic_vector(0 downto 0) ;
    inst_ram_wr   : out    std_logic_vector(0 downto 0) ;
    inst_ram      : out    std_logic_vector(32 downto 0) ;
    inst_ram_in   : in     std_logic_vector(32 downto 0) ) ;
end  inst_dec ;

-------------------------------------------------------------------------------
-- Architecture 'a0' of 'inst_dec'
-------------------------------------------------------------------------------

architecture a0 of inst_dec is

begin
	process(reset, rd, wr, ce, a2, data, current_state, dmx_data, inst_ram_in)
		variable state : std_logic_vector(4 downto 0);
		variable mode : std_logic_vector(1 downto 0);
	begin
		state := current_state(6 downto 2);
		mode := current_state(1 downto 0);
		dmx_ctrl <= "00";
		dmx_data <= (others => 'Z');
		dmx_ram <= (others => 'Z');
		dmx_ram_wr <= "0";
		data <= (others => 'Z');
		inst_ram <= (others => 'Z');
		inst_ram_wr <= "0";
		if (reset = "0") then
			state := "00000";
			mode := "00";
		else
			case mode is
				when "00" =>
					if (rd = "0" and ce = "0") then
						mode := "10";
					elsif (wr = "0" and ce = "0") then
						mode := "11";
					end if;
				when "10" => -- read
					if (ce = "1") then
						mode := "00";
					end if;
				when "11" => -- write
					if (ce = "1") then
						mode := "00";
					end if;
				when others =>
					mode := "00";
```

```
                        end case;
                        case state is
                                when "00000" => -- first byte
                                        if (mode = "11") then
                                                state := "00001";
                                        end if;
                                when "00001" =>
                                        inst_ram <= inst_ram_in(32) & data &
inst_ram_in(23 downto 0);
                                        inst_ram_wr <= "1";
                                        state := "10001";
                                when "10001" =>
                                        if (mode = "00") then -- instruction detection
here...
                                                case inst_ram_in(31 downto 29) is
                                                        when "000" =>
                                                                state := "00100";
                                                        when "001" =>
                                                                state := "00110";
                                                        when others =>
                                                                state := "00000";
                                                end case;
                                        end if;
                                when "00010" => -- second byte
                                        if (mode = "11") then
                                                state := "00011";
                                        end if;
                                when "00011" =>
                                        inst_ram <= inst_ram_in(32 downto 24) & data &
inst_ram_in(15 downto 0);
                                        inst_ram_wr <= "1";
                                        state := "10011";
                                when "10011" =>
                                        if (mode = "00") then
                                                state := "00100";
                                        end if;
                                when "00100" => -- third byte
                                        if (mode = "11") then
                                                state := "00101";
                                        end if;
                                when "00101" =>
                                        inst_ram <= inst_ram_in(32 downto 16) & data &
inst_ram_in(7 downto 0);
                                        inst_ram_wr <= "1";
                                        state := "10101";
                                when "10101" =>
                                        if (mode = "00") then
                                                state := "00110";
                                        end if;
                                when "00110" => -- forth byte
                                        if (mode = "11") then
                                                state := "00111";
                                        end if;
                                when "00111" =>
                                        inst_ram <= a2 &inst_ram_in(31 downto 8) & data;
                                        inst_ram_wr <= "1";
                                        state := "10111";
                                when "10111" =>
                                        if (mode = "00") then
                                                state := "00000";
                                                -- here it happens
                                                case inst_ram_in(32 downto 28) is
                                                        when "00000" =>
                                                                dmx_ram <= inst_ram_in(31
downto 24) & inst_ram_in(15 downto 0);
                                                                dmx_ram_wr <= "1";
                                                                dmx_ctrl <= "10";
                                                        when "00001" =>
                                                                dmx_ram <= inst_ram_in(31
downto 24) & inst_ram_in(15 downto 0);
                                                                dmx_ram_wr <= "1";
                                                                dmx_ctrl <= "10";
                                                        when "00010" =>
                                                                dmx_ram <= inst_ram_in(31
downto 25) & (inst_ram_in(24) & inst_ram_in(15 downto 8))+1 & inst_ram_in(7 downto 0);
                                                                dmx_ram_wr <= "1";
                                                                dmx_ctrl <= "11";
```

```
                                                        when "00011" =>
                                                               dmx_ram <= inst_ram_in(31
downto 24) & inst_ram_in(15 downto 0);
                                                               dmx_ram_wr <= "1";
                                                               dmx_ctrl <= "11";
                                                        when others =>
                                                end case;
                                        end if;
                                when others =>
                                        state := "00000";
                        end case;
                end if;
                next_state <= state & mode;
        end process;
end  a0 ; -- of inst_dec

--------------------------------------------------------------------------------
-- Entity declaration of 'flipflop'.
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity flipflop is
  generic(
    WIDTH : Natural := 1 ) ;
  port(
    din  : in     std_logic_vector(WIDTH-1 downto 0) ;
    dout : out    std_logic_vector(WIDTH-1 downto 0) ;
    clk  : in     std_logic ) ; -- Clock
end  flipflop ;

--------------------------------------------------------------------------------
-- Architecture 'a0' of 'flipflop'
--------------------------------------------------------------------------------

architecture a0 of flipflop is

begin
        process(clk) begin
                if (clk='1' and clk'last_value='0') then
             dout <= din after 1 ns;
            end if;
        end process;
end  a0 ; -- of flipflop

--------------------------------------------------------------------------------
-- Entity declaration of 'flashif'.
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity flashif is
  port(
    reset   : in     std_logic_vector(0 downto 0) ;
    we      : out    std_logic_vector(0 downto 0) ;
    oe      : out    std_logic_vector(0 downto 0) ;
    address : out    std_logic_vector(18 downto 0) ;
    data    : inout  std_logic_vector(7 downto 0) := (others => 'Z') ) ;
end  flashif ;

--------------------------------------------------------------------------------
-- Architecture 'a0' of 'flashif'
--------------------------------------------------------------------------------

architecture a0 of flashif is

begin
        process(reset) begin
                we <= "1";
                oe <= "1";
                address <= (others => 'Z');
```

```
                data <= (others => 'Z');
                if (reset = "0") then
                end if;
        end process;

end  a0 ; -- of flashif

--------------------------------------------------------------------------------
-- Entity declaration of 'flashctrl'.
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity flashctrl is
  port(
    reset : in     std_logic_vector(0 downto 0) ) ;
end  flashctrl ;

--------------------------------------------------------------------------------
-- Architecture 'a0' of 'flashctrl'
--------------------------------------------------------------------------------

architecture a0 of flashctrl is

begin
        process(reset) begin
                if (reset = "0") then

                end if;
        end process;
end  a0 ; -- of flashctrl

--------------------------------------------------------------------------------
-- Entity declaration of 'ramif'.
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity ramif is
  port(
    data          : inout  std_logic_vector(7 downto 0) ;
    address       : out    std_logic_vector(8 downto 0) ;
    oe            : out    std_logic_vector(0 downto 0) ;
    we            : out    std_logic_vector(0 downto 0) ;
    address_in    : in     std_logic_vector(8 downto 0) ;
    data_in       : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
    wr            : in     std_logic_vector(0 downto 0) ;
    rd1           : in     std_logic_vector(0 downto 0) ;
    ack           : out    std_logic_vector(0 downto 0) ;
    next_state    : out    std_logic_vector(3 downto 0) ;
    current_state : in     std_logic_vector(3 downto 0) ;
    reset         : in     std_logic_vector(0 downto 0) ;
    free          : out    std_logic_vector(0 downto 0) ;
    rd0           : in     std_logic_vector(0 downto 0) ) ;
end  ramif ;

--------------------------------------------------------------------------------
-- Architecture 'a0' of 'ramif'
--------------------------------------------------------------------------------

architecture a0 of ramif is

begin
        process(address_in, wr, rd0, rd1, current_state, reset)
                variable state : std_logic_vector(3 downto 0);
        begin
                state := current_state(3 downto 0);
                data <= (others => 'Z');
                data_in <= (others => 'Z');
                oe <= "1";
                we <= "1";
```

```
                ack <= "0";
                free <= "0";
                address <= (others => 'Z');
                if (reset = "0") then
                        state := "0000";
                else
                        case state is
                                when "0000" =>
                                        if (rd0 = "0" or rd1 = "0") then
                                                --address <= address_in;
                                                state := "0001";
                                        elsif (wr = "0") then
                                                --address <= address_in;
                                                state := "1001";
                                        else
                                                free <= "1";
                                                --state := current_state;
                                        end if;
                                when "0001" => -- kanske helt överflödigt?
                                        address <= address_in;
                                        oe <= "0";
                                        state := "0010";
                                when "0010" =>
                                        address <= address_in;
                                        oe <= "0";
                                        ack <= "1";
                                        state := "0011";
                                when "0011" =>
                                        address <= address_in;
                                        data_in <= data;
                                        state := "0000";


                                when "1001" => -- kanske ett state för mycket?
                                        address <= address_in;
                                        oe <= "1";
                                        data <= data_in;
                                        state := "1010";
                                when "1010" =>
                                        address <= address_in;
                                        oe <= "1";
                                        we <= "0";
                                        ack <= "1";
                                        data <= data_in;
                                        state := "1011";
                                when "1011" =>
                                        address <= address_in;
                                        data <= data_in;
                                        state := "0000";
                                when others =>
                                        state := "0000";
                        end case;
                end if;
                next_state <= state;
        end process;
end  a0 ; -- of ramif

--------------------------------------------------------------------------------
-- Entity declaration of 'dmxgen'.
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity dmxgen is
  port(
    current_state : in     std_logic_vector(42 downto 0) ;
    next_state    : out    std_logic_vector(42 downto 0) ;
    reset         : in     std_logic_vector(0 downto 0) ;
    free          : in     std_logic_vector(0 downto 0) ;
    ack           : in     std_logic_vector(0 downto 0) ;
    data          : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
    rd            : out    std_logic_vector(0 downto 0) := "Z" ;
    address       : out    std_logic_vector(8 downto 0) := (others => 'Z') ;
    dmx           : out    std_logic_vector(0 downto 0) ) ;
```

```
end  dmxgen ;

-------------------------------------------------------------------------
-- Architecture 'a0' of 'dmxgen'
-------------------------------------------------------------------------

architecture a0 of dmxgen is

begin
	process(current_state, reset, free, ack, data)
		variable cur_addr : std_logic_vector(8 downto 0);
		variable rr : std_logic_vector(1 downto 0);
		variable next_byte : std_logic_vector(7 downto 0);
		variable current_byte : std_logic_vector(10 downto 0);
		variable clk_counter : std_logic_vector(4 downto 0);
		variable state : std_logic_vector(3 downto 0);
		variable shift_counter : std_logic_vector(3 downto 0);
	begin
		cur_addr := current_state(42 downto 34);
		rr := current_state(33 downto 32);
		next_byte := current_state(31 downto 24);
		current_byte := current_state(23 downto 13);
		clk_counter := current_state(12 downto 8);
		state := current_state(7 downto 4);
		shift_counter := current_state(3 downto 0);
		rd <= "1";
		address <= (others => 'Z');
		dmx <= "1"; -- dmx is high when idle
		if (reset = "0") then
			cur_addr := "000000000";
			rr := "00";
			next_byte := "00000000";
			current_byte := "00000000000"; --start-byte!
			clk_counter := "00000";
			state := "0000";
			shift_counter := "0000";
		else
			case state is
				when "0000" =>
					current_byte := "00000000000"; -- break
					shift_counter := "1011";
					clk_counter := "00000";
					state := "0001";
					rr := "01";
				when "0001" =>
					if (shift_counter = "0000") then
						state := "0010";
					end if;
				when "0010" =>
					current_byte := "00000000000"; -- break
					shift_counter := "1011";
					clk_counter := "00000";
					state := "0011";
				when "0011" =>
					if (shift_counter = "0000") then
						state := "0100";
					end if;
				when "0100" =>
					current_byte := "00000001111"; -- mark after
break
					shift_counter := "0011";
					clk_counter := "00000";
					state := "0101";
				when "0101" =>
					if (shift_counter = "0000") then
						state := "0110";
					end if;
				when "0110" =>
					current_byte := "11000000000"; -- start code
					shift_counter := "1011";
					clk_counter := "00000";
					state := "0111";
				when "0111" =>
					if (shift_counter = "0000") then
						state := "1000";
					end if;
				when "1000" =>
```

```
                                        current_byte := "11" & next_byte & "0"; -- dmx-
data
                                        shift_counter := "1011";
                                        clk_counter := "00001";
                                        rr := "01";
                                        cur_addr := cur_addr + 1;
                                        state := "1001";
                                when "1001" =>
                                        if (shift_counter = "0000") then
                                                if (cur_addr = "000000000") then
                                                        state := "0000";
                                                else
                                                        state := "1000";
                                                end if;
                                        end if;
                            when others =>
                                state := "0000";
                        end case;
                        if (rr = "01" and free = "1") then
                                address <= cur_addr;
                                rd <= "0";
                                rr := "11";
                        elsif (rr = "11") then
                                if (ack = "1") then
                                        rr := "00";
                                        next_byte := data;
                                else
                                        address <= cur_addr;
                                        rd <= "0";
                                end if;
                        end if;

                        clk_counter := clk_counter + 1;
                        if (clk_counter = "00000") then
                                current_byte := "0" & current_byte(10 downto 1);
                                shift_counter := shift_counter - 1;
                        end if;
                        dmx <= current_byte(0 downto 0);
                end if;
                next_state <= cur_addr & rr & next_byte & current_byte & clk_counter &
state & shift_counter;
        end process;

end  a0 ; -- of dmxgen

-------------------------------------------------------------------------------
-- Entity declaration of 'dmxctrl'.
-------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity dmxctrl is
  port(
    init_done     : out    std_logic_vector(0 downto 0) ;
    reset         : in     std_logic_vector(0 downto 0) ;
    wr            : out    std_logic_vector(0 downto 0) ;
    rd            : out    std_logic_vector(0 downto 0) := "Z" ;
    address       : out    std_logic_vector(8 downto 0) := (others => 'Z') ;
    data          : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
    next_state    : out    std_logic_vector(15 downto 0) ;
    current_state : in     std_logic_vector(15 downto 0) ;
    data_in       : inout  std_logic_vector(7 downto 0) ;
    ctrl_in       : in     std_logic_vector(1 downto 0) ;
    dmx_ram       : in     std_logic_vector(23 downto 0) ;
    free          : in     std_logic_vector(0 downto 0) ;
    ack           : in     std_logic_vector(0 downto 0) ;
    p0            : out    std_logic_vector(0 downto 0) ;
    p1            : out    std_logic_vector(0 downto 0) ) ;
end  dmxctrl ;

-------------------------------------------------------------------------------
-- Architecture 'a0' of 'dmxctrl'
-------------------------------------------------------------------------------
```

```
architecture a0 of dmxctrl is

begin
	process(reset, current_state, data_in, ctrl_in, ack, free, dmx_ram)
		variable state : std_logic_vector(4 downto 0);
		variable curr_addr : std_logic_vector(8 downto 0);
		variable ww : std_logic_vector(1 downto 0);
	begin
		curr_addr := current_state(15 downto 7);
		ww := current_state(6 downto 5);
		state := current_state(4 downto 0);

		init_done <= "1";
		wr <= "1";
		rd <= "1";
		data <= (others => 'Z');
		address <= (others => 'Z');
		data_in <= (others => 'Z');
		p0 <= "0";
		p1 <= "0";
		if (reset = "0") then
			state := "00000";
			curr_addr := "000000000";
			ww := "00";
			p0 <= "1";
		else
			case state is
				when "00000" => -- överflödigt state?
				    address <= curr_addr;
				    state := "00001";
				    wr <= "0";
				when "00001" =>
					address <= curr_addr;
					state := "00010";
					data <= "00000000";
					wr <= "0";
				when "00010" =>
					address <= curr_addr;
					data <= "00000000";
					state := "00011";
				when "00011" =>
					curr_addr := curr_addr + 1;
					if (curr_addr = "000000000") then
						state := "00100";
						init_done <= "0";
						p1 <= "1";
					else
						state := "00000";
					end if;
				when "00100" => -- wait-state

					if (ctrl_in = "10") then
						state := "00101";
					elsif (ctrl_in = "11") then
						state := "00100";
					end if;
				when "00101" =>
					case dmx_ram(23 downto 20) is
						when "0000" =>
							ww := "01";
							state := "00100";
						when others =>
							state := "00100";
					end case;
				when others =>
					state := "00100";
			end case;
			if (ww = "01" and free = "1") then
				ww := "11";
				wr <= "0";
				address <= dmx_ram(16 downto 8);
				data <= dmx_ram(7 downto 0);
			elsif (ww = "11") then
				if (ack = "1") then
					ww := "00";
				else
					address <= dmx_ram(16 downto 8);
```

```
                                    data <= dmx_ram(7 downto 0);
                                    wr <= "0";
                                end if;
                            end if;
                    end if;
                    next_state <= curr_addr & ww & state;
        end process;
end  a0 ; -- of dmxctrl

-------------------------------------------------------------------------------
-- Entity declaration of 'memif'.
-------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

entity memif is
  port(
    rd            : in     std_logic_vector(0 downto 0) ;
    wr            : in     std_logic_vector(0 downto 0) ;
    ce            : in     std_logic_vector(0 downto 0) ;
    reset         : in     STD_LOGIC_VECTOR(0 downto 0) ;
    clk           : in     std_logic ;
    a2            : in     std_logic_vector(0 downto 0) ;
    data          : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
    dmx           : out    std_logic_vector(0 downto 0) ;
    address_flash : out    std_logic_vector(18 downto 0) := (others => 'Z') ;
    data_flash    : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
    we_flash      : out    std_logic_vector(0 downto 0) := "1" ;
    oe_flash      : out    std_logic_vector(0 downto 0) := "1" ;
    oe_ram        : out    std_logic_vector(0 downto 0) ;
    we_ram        : out    std_logic_vector(0 downto 0) ) ;
end  memif ;

-------------------------------------------------------------------------------
-- Architecture 'a0' of 'memif'
-------------------------------------------------------------------------------

architecture a0 of memif is


 component dmxctrl
  port(
    init_done     : out    std_logic_vector(0 downto 0) ;
    reset         : in     std_logic_vector(0 downto 0) ;
    wr            : out    std_logic_vector(0 downto 0) ;
    rd            : out    std_logic_vector(0 downto 0) := "Z" ;
    address       : out    std_logic_vector(8 downto 0) := (others => 'Z') ;
    data          : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
    next_state    : out    std_logic_vector(15 downto 0) ;
    current_state : in     std_logic_vector(15 downto 0) ;
    data_in       : inout  std_logic_vector(7 downto 0) ;
    ctrl_in       : in     std_logic_vector(1 downto 0) ;
    dmx_ram       : in     std_logic_vector(23 downto 0) ;
    free          : in     std_logic_vector(0 downto 0) ;
    ack           : in     std_logic_vector(0 downto 0) ;
    p0            : out    std_logic_vector(0 downto 0) ;
    p1            : out    std_logic_vector(0 downto 0) ) ;
 end component ;
 component dmxgen
  port(
    current_state : in     std_logic_vector(42 downto 0) ;
    next_state    : out    std_logic_vector(42 downto 0) ;
    reset         : in     std_logic_vector(0 downto 0) ;
    free          : in     std_logic_vector(0 downto 0) ;
    ack           : in     std_logic_vector(0 downto 0) ;
    data          : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
    rd            : out    std_logic_vector(0 downto 0) := "Z" ;
    address       : out    std_logic_vector(8 downto 0) := (others => 'Z') ;
    dmx           : out    std_logic_vector(0 downto 0) ) ;
 end component ;
 component ramif
  port(
    data          : inout  std_logic_vector(7 downto 0) ;
    address       : out    std_logic_vector(8 downto 0) ;
```

```
   oe            : out    std_logic_vector(0 downto 0) ;
   we            : out    std_logic_vector(0 downto 0) ;
   address_in    : in     std_logic_vector(8 downto 0) ;
   data_in       : inout  std_logic_vector(7 downto 0) := (others => 'Z') ;
   wr            : in     std_logic_vector(0 downto 0) ;
   rd1           : in     std_logic_vector(0 downto 0) ;
   ack           : out    std_logic_vector(0 downto 0) ;
   next_state    : out    std_logic_vector(3 downto 0) ;
   current_state : in     std_logic_vector(3 downto 0) ;
   reset         : in     std_logic_vector(0 downto 0) ;
   free          : out    std_logic_vector(0 downto 0) ;
   rd0           : in     std_logic_vector(0 downto 0) ) ;
end component ;
component flashctrl
 port(
   reset : in     std_logic_vector(0 downto 0) ) ;
end component ;
component flashif
 port(
   reset   : in     std_logic_vector(0 downto 0) ;
   we      : out    std_logic_vector(0 downto 0) ;
   oe      : out    std_logic_vector(0 downto 0) ;
   address : out    std_logic_vector(18 downto 0) ;
   data    : inout  std_logic_vector(7 downto 0) := (others => 'Z') ) ;
end component ;
component flipflop
 generic(
   WIDTH : Natural := 1 ) ;
 port(
   din  : in     std_logic_vector(WIDTH-1 downto 0) ;
   dout : out    std_logic_vector(WIDTH-1 downto 0) ;
   clk  : in     std_logic ) ; -- Clock
end component ;
component inst_dec
 port(
   rd            : in     std_logic_vector(0 downto 0) ;
   wr            : in     std_logic_vector(0 downto 0) ;
   ce            : in     std_logic_vector(0 downto 0) ;
   a2            : in     std_logic_vector(0 downto 0) ;
   data          : inout  std_logic_vector(7 downto 0) ;
   next_state    : out    std_logic_vector(6 downto 0) ;
   current_state : in     std_logic_vector(6 downto 0) ;
   dmx_data      : inout  std_logic_vector(7 downto 0) ;
   dmx_ctrl      : out    std_logic_vector(1 downto 0) ;
   reset         : in     std_logic_vector(0 downto 0) ;
   dmx_ram       : out    std_logic_vector(23 downto 0) ;
   dmx_ram_wr    : out    std_logic_vector(0 downto 0) ;
   inst_ram_wr   : out    std_logic_vector(0 downto 0) ;
   inst_ram      : out    std_logic_vector(32 downto 0) ;
   inst_ram_in   : in     std_logic_vector(32 downto 0) ) ;
end component ;
component dmx_ram
 port(
   we       : in     std_logic_vector(0 downto 0) ;
   data     : in     std_logic_vector(23 downto 0) ;
   clk      : in     std_logic ;
   data_out : out    std_logic_vector(23 downto 0) ) ;
end component ;
component inst_ram
 port(
   we       : in     std_logic_vector(0 downto 0) ;
   data     : in     std_logic_vector(32 downto 0) ;
   clk      : in     std_logic ;
   data_out : out    std_logic_vector(32 downto 0) ) ;
end component ;
signal Net_0  : std_logic_vector(0 downto 0);
signal Net_1  : std_logic_vector(0 downto 0);
signal Net_2  : std_logic_vector(8 downto 0);
signal Net_3  : std_logic_vector(3 downto 0);
signal Net_4  : std_logic_vector(3 downto 0);
signal Net_5  : std_logic_vector(8 downto 0);
signal Net_6  : std_logic_vector(7 downto 0);
signal Net_7  : std_logic_vector(0 downto 0);
signal Net_8  : std_logic_vector(0 downto 0);
signal Net_9  : std_logic_vector(0 downto 0);
signal Net_11 : std_logic_vector(0 downto 0);
signal Net_16 : std_logic_vector(15 downto 0);
```

```
 signal Net_17 : std_logic_vector(15 downto 0);
 signal Net_18 : std_logic_vector(0 downto 0);
 signal Net_19 : std_logic_vector(0 downto 0);
 signal Net_20 : std_logic_vector(0 downto 0) := "1";
 signal Net_21 : std_logic_vector(0 downto 0);
 signal Net_22 : std_logic_vector(0 downto 0);
 signal Net_23 : std_logic_vector(0 downto 0) := "1";
 signal Net_24 : std_logic_vector(42 downto 0);
 signal Net_25 : std_logic_vector(42 downto 0);
 signal Net_26 : std_logic_vector(0 downto 0);
 signal Net_32 : std_logic_vector(6 downto 0);
 signal Net_33 : std_logic_vector(6 downto 0);
 signal Net_34 : std_logic_vector(1 downto 0);
 signal Net_35 : std_logic_vector(7 downto 0);
 signal Net_36 : std_logic_vector(1 downto 0);
 signal Net_37 : std_logic_vector(23 downto 0);
 signal Net_38 : std_logic_vector(0 downto 0);
 signal Net_39 : std_logic_vector(23 downto 0);
 signal Net_40 : std_logic_vector(0 downto 0);
 signal Net_41 : std_logic_vector(32 downto 0);
 signal Net_42 : std_logic_vector(32 downto 0);

begin

 u1: dmxctrl
    port map(
      init_done => Net_8,
      reset => reset,
      wr => Net_11,
      rd => Net_21,
      address => Net_5,
      data => Net_6,
      next_state => Net_16,
      current_state => Net_17,
      data_in => Net_35,
      ctrl_in => Net_34,
      dmx_ram => Net_37,
      free => Net_7,
      ack => Net_19,
      p0 => we_ram,
      p1 => oe_ram ) ;

 u2: dmxgen
    port map(
      current_state => Net_25,
      next_state => Net_24,
      reset => Net_8,
      free => Net_7,
      ack => Net_19,
      data => Net_6,
      rd => Net_20,
      address => Net_5,
      dmx => Net_9 ) ;

 u3: ramif
    port map(
      data => open,
      address => Net_2,
      oe => Net_1,
      we => Net_0,
      address_in => Net_5,
      data_in => Net_6,
      wr => Net_26,
      rd1 => Net_23,
      ack => Net_18,
      next_state => Net_3,
      current_state => Net_4,
      reset => reset,
      free => Net_7,
      rd0 => Net_22 ) ;

 u4: flashctrl
    port map(
      reset => reset ) ;

 u5: flashif
    port map(
```

```
     reset => reset,
     we => we_flash,
     oe => oe_flash,
     address => address_flash,
     data => data_flash ) ;

u8: flipflop
   generic map(
     WIDTH => 1 )
   port map(
     din => Net_0,
     dout => open,
     clk => clk ) ;

u9: flipflop
   generic map(
     WIDTH => 1 )
   port map(
     din => Net_1,
     dout => open,
     clk => clk ) ;

u10: flipflop
   generic map(
     WIDTH => 9 )
   port map(
     din => Net_2,
     dout => open,
     clk => clk ) ;

u11: flipflop
   generic map(
     WIDTH => 4 )
   port map(
     din => Net_3,
     dout => Net_4,
     clk => clk ) ;

u12: flipflop
   generic map(
     WIDTH => 43 )
   port map(
     din => Net_24,
     dout => Net_25,
     clk => clk ) ;

u13: flipflop
   generic map(
     WIDTH => 1 )
   port map(
     din => Net_9,
     dout => dmx,
     clk => clk ) ;

u14: flipflop
   generic map(
     WIDTH => 16 )
   port map(
     din => Net_16,
     dout => Net_17,
     clk => clk ) ;

u15: flipflop
   generic map(
     WIDTH => 1 )
   port map(
     din => Net_18,
     dout => Net_19,
     clk => clk ) ;

u16: flipflop
   generic map(
     WIDTH => 1 )
   port map(
     din => Net_21,
     dout => Net_22,
     clk => clk ) ;
```

```
 u17: flipflop
    generic map(
      WIDTH => 1 )
    port map(
      din => Net_20,
      dout => Net_23,
      clk => clk ) ;

 u18: flipflop
    generic map(
      WIDTH => 1 )
    port map(
      din => Net_11,
      dout => Net_26,
      clk => clk ) ;

 u0: inst_dec
    port map(
      rd => rd,
      wr => wr,
      ce => ce,
      a2 => a2,
      data => data,
      next_state => Net_32,
      current_state => Net_33,
      dmx_data => Net_35,
      dmx_ctrl => Net_36,
      reset => reset,
      dmx_ram => Net_39,
      dmx_ram_wr => Net_38,
      inst_ram_wr => Net_40,
      inst_ram => Net_41,
      inst_ram_in => Net_42 ) ;

 u6: flipflop
    generic map(
      WIDTH => 7 )
    port map(
      din => Net_32,
      dout => Net_33,
      clk => clk ) ;

 u7: flipflop
    generic map(
      WIDTH => 2 )
    port map(
      din => Net_36,
      dout => Net_34,
      clk => clk ) ;

 u19: dmx_ram
    port map(
      we => Net_38,
      data => Net_39,
      clk => clk,
      data_out => Net_37 ) ;

 u20: inst_ram
    port map(
      we => Net_40,
      data => Net_41,
      clk => clk,
      data_out => Net_42 ) ;
end  a0 ; -- of memif
```

Appendix 4

Programkod, PAL för i/o:

```
device 22V10
CLK         1
A12         2
A13         3
A14         4
A15         5
A16         6
DS          7     'Data strobe
AS          8     'Address strobe
RW          9
GND         12
DTACK       14    'Data Ack
WR          15    'Write
RD          16    'Read
CSPROM            17    'CS EPROM
CSRAM       18    'CS RAM
ADSTART     19    'AD Converter start
CSKEYB            20    'CS keyboard flipflops
DISPE       21    'Display enable
ADOE        22    'AD Converter output enable
CSMEMIF     23    'CS MemIF
VCC         24
start
CSPROM /= /AS * /A16 * /A15 * /A14 * /A13;
CSRAM /= /AS * /A16 * /A15 * A14 * A13;
CSKEYB /= /AS * A16 * /A15 * /A14 * /A13 * A12;
RD /= /DS * RW;
WR /= /DS * /RW;
DISPE = /DS * /AS * A16 * /A15 * /A14 * /A13 * /A12;
CSMEMIF /= /AS * A16 * /A15 * /A14 * A13 * /A12;
ADSTART = /AS * A16 * /A15 * /A14 * A13 * A12 * /RW;
ADOE = /AS * A16 * /A15 * /A14 * A13 * A12 * RW;
DTACK /= /CSPROM + /CSRAM + /CSKEYB + ADSTART + ADOE + /CSMEMIF + /AS
* A16 * /A15 * /A14 * /A13 * /A12;
end
```

Programkod, PAL för avbrottshantering:

```
device 22V10
A1    1
A2    2
A3    3
FC0   4
FC1   5
FC2   6
AS    7     'Address strobe
INTAD 8     'Interrupt AD Converter
VPA   14    'Vector interrupt
IPL02 15    'Interrupt 0/2
IPL1  16    'Interrupt 1
start
VPA /= FC0 * FC1 * FC2 * /AS * A1 * /A2 * A3;
IPL02 /= INTAD;
IPL1 /= 1;
end
```

Appendix 5

Instruktionsuppsättning för FPGA, DMX-instruktioner:

0000 ---A	AAAAAAAA	DDDDDDDD		Sätt DMX-värde på adress
0001 ---A	AAAAAAAA	DDDDDDDD		Skriv värde på adress, burst
0010 ----	DDDDDDDD			Skriv värde burst
0011 ----	DDDDDDDD			Skriv värde sista byten burst

Instruktionsuppsättning för FPGA, EEPROM-instruktioner:

0000 --AA	AAAAAAAA	AAAAAAAA	DDDDDDDD	Skriv data på adress
0001 --AA	AAAAAAAA	AAAAAAAA	DDDDDDDD	Skriv data på adress, burst
0010 ----	DDDDDDDD			Skriv data burst
0011 ----	DDDDDDDD			Skriv data sista byten burst
0100 --AA	AAAAAAAA	AAAAAAAA		Läs data från adress

Förklaring:

A – Adressbitar

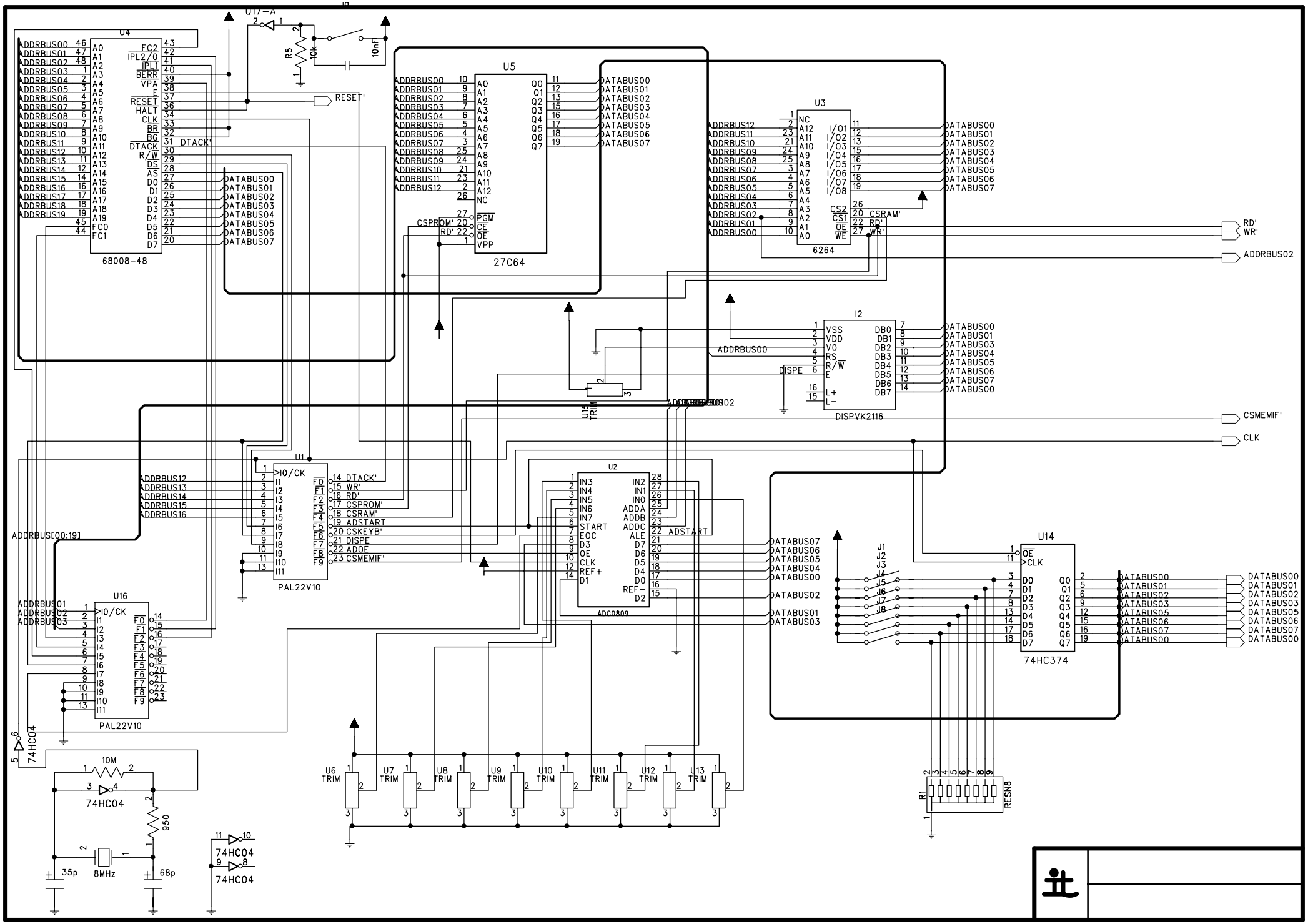
D – Databitar

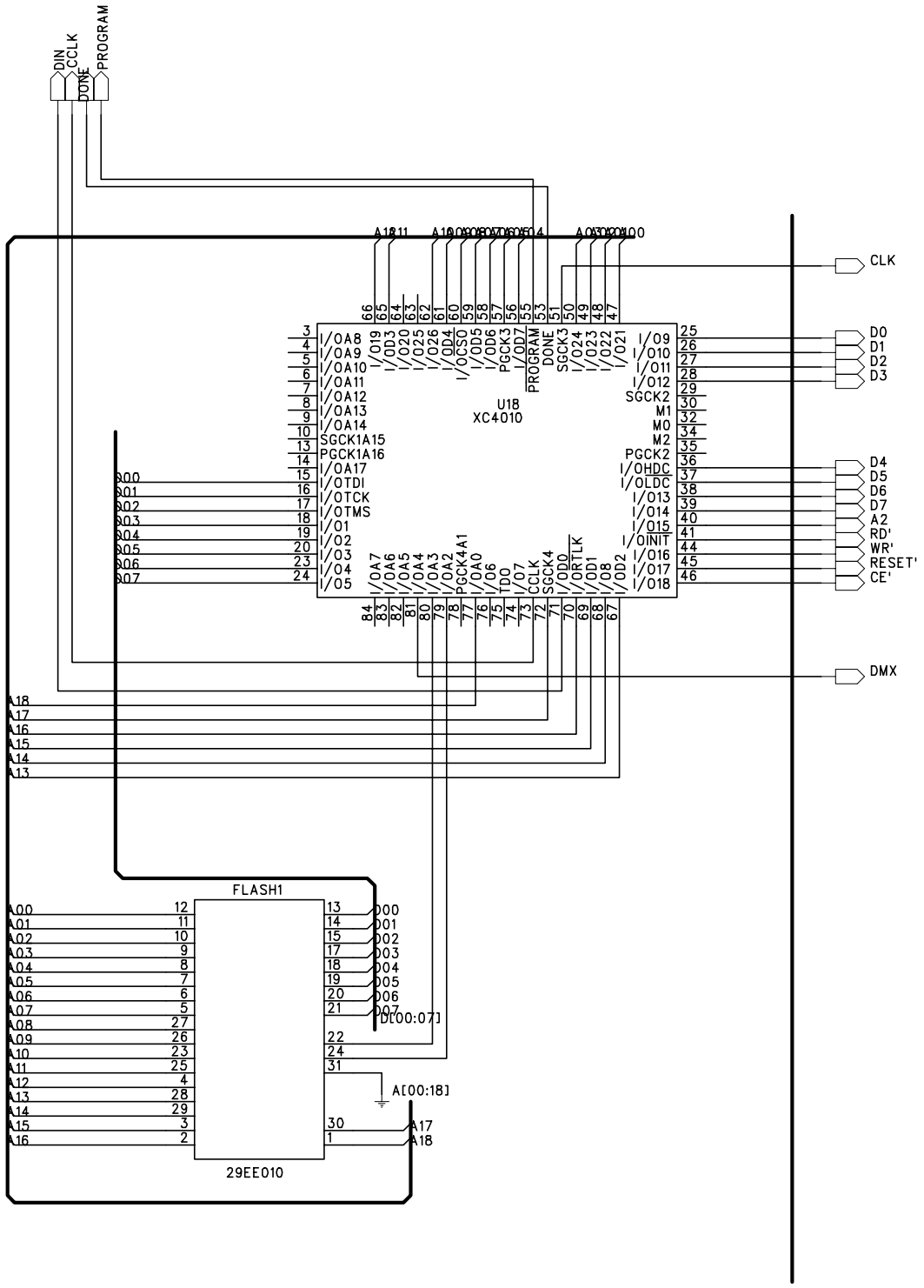
- – Don’t care

Appendix 6

Minneskarta

Minnesadress	Enhet
0x00000- 0x01FFF	EPROM (Programminne)
0x06000- 0x07FFF	RAM (RAM-minne)
0x10000- 0x10001	LCD-display
0x11000	Tangentbord
0x12000	FPGA DMX
0x12004	FPGA EEPROM
0x13000- 0x13007	AD


U4
68008-48
U5
27C64
U3
6264
I2
DISPVK2116
U1
PAL22V10
U2
ADC0809
U14
74HC374
U16
PAL22V10
U17=A
74HC04
R5
10k
10nF
RESET'
DTACK'
CSPROM'
CSRAM'
ADSTART
CSKEYB'
DISPE
ADOE
CSMEMIF'
WR'
RD'
CLK
ADDRBUS02
ADDRBUS00
ADDRBUS[00:19]
10M
950
35p
8MHz
68p
U6 TRIM
U7 TRIM
U8 TRIM
U9 TRIM
U10 TRIM
U11 TRIM
U12 TRIM
U13 TRIM
R1
RESNB


DIN
CCLK
DONE
PROGRAM
U18
XC4010
I/OA8
I/OA9
I/OA10
I/OA11
I/OA12
I/OA13
I/OA14
SGCK1A15
PGCK1A16
I/OA17
I/OTDI
I/OTCK
I/OTMS
I/O1
I/O2
I/O3
I/O4
I/O5
I/O9
I/O10
I/O11
I/O12
SGCK2
M1
M0
M2
PGCK2
I/OHDC
I/OLDC
I/O13
I/O14
I/O15
I/OINIT
I/O16
I/O17
I/O18
CLK
D0
D1
D2
D3
D4
D5
D6
D7
A2
RD'
WR'
RESET'
CE'
DMX
FLASH1
29EE010
D[00:07]
A[00:18]