GPU I FPGA

PROJEKTRAPPORT I DIGITALA PROJEKT

AV DANIEL OLSSON OCH FRANCISCO IGLESIAS

2004-05-17

INSTUTITIONEN FÖR INFORMATIONSTEKNOLOGI

INNEHÅLLSFÖRTECKNING

Illustrationsförteckning

INLEDNING

Kärnan kring vårt projekt var att utveckla något med en FPGA. Beskrivningsspråket för att modellera digital hårdvara – VHDL, hade vi kommit i kontakt med i föregående kurser. Detta ville vi omsätta i ett eget system med egen design!

Projektet vi satte igång med innebar att konfigurera FPGAn som en GPU (Graphic Processing Unit). Den skulle klara av att återge en bild på en LCD-display som den hämtade från ett externt minne. Vi planerade även grafiska funktioner för att manipulera minnet, som att fylla ytor och att flytta areor.

En processor skulle via ett enkelt testprogram förse GPUn med all data för att kunna göra det den var avsedd för.

Att det skulle vara en dans på rosor var inget vi hade förväntat oss, men att få en krets att bete sig på exakt det sätt vi ville visa sig vara riktigt svårt. Vi kommer här att presentera vår arbetsgång från liten idé till något realiserbart.

TEORI

Idén var att en processor skriver bildinformation till SRAM i form av bitmappsdata. FPGAn läser bildinformationen från SRAMet och återger detta på en LCD-display. FPGAn skulle även kunna utföra enkla grafiska operationer på bitmappsdatan i SRAMet, såsom fylla och flytta minnesareor[1].

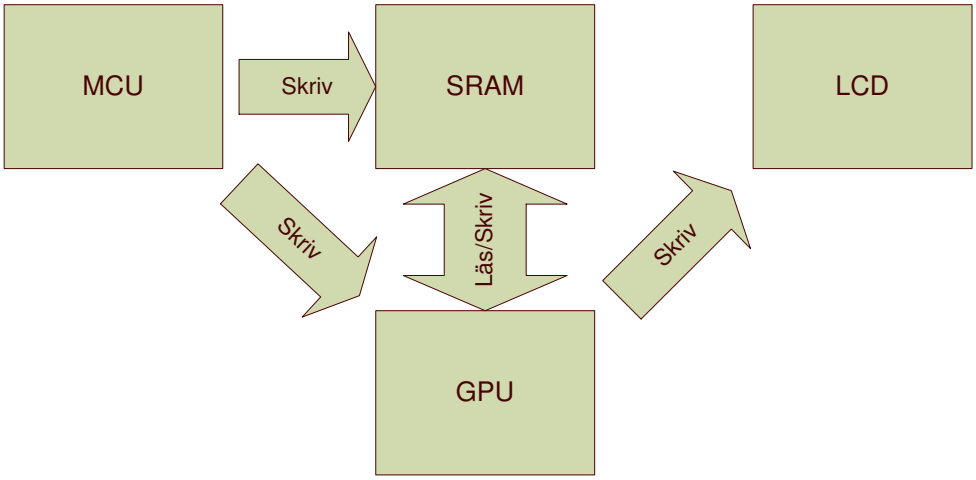


Illustration 1Blockschema över konstruktionen

FPGA-kretsen vi fick till förfogande var en Xilinx 4010E. En kraftfull krets som gott och väl skulle räcka för våra behov. Denna krets är en SRAM-baserad FPGA, vilket innebär att den sparar sin konfiguration i ett SRAM. Detta gör att den tappar all programmering när konstruktionen blir spänningslös, och således måste programmeras vid uppstart.

Efter att ha läst in åtskilliga datablad visade det sig att det fanns 8 olika sätt att programmera Xilinx-kretsen på. Vi valde att under utveckling använda oss av ett *nionde* - det inbyggda JTAG gränssnittet i kretsen. Detta skulle möjliggöra att vi kunde programmera kretsen från utvecklingsmiljön via en ”Parallell – JTAG”-kabel, utan att några stora ingrepp i konstruktionen behöver göras. Samtidigt reserverade vi oss för att eventuellt kunna programmera FPGAn från processorn.

Initialt siktade vi in oss på en mikrokontroller från Motorola – 68HC11. Denna har inbyggt en uppsjö av funktioner som räknare och I/O-portar, som verkade frestande att slippa implementera. Men, eftersom vår applikation till stor del bestod av att flytta data och hantera minnesmängder större än 512 bytes, avrådde vår handledare oss starkt från detta. Rådet var att istället inrikta oss på en släkting Motorola 68008. Detta visade sig senare vara ett mycket klokt beslut.

Ett kretsschema kring en 68008 processor blev vår första karta. Utgångspunkten var att hålla det enkelt. 48-pinnars versionen av 68008 har en 20 bitars bred

1Man kan göra en grov jämförelse för de som är bekanta med hårdvaran i Amiga-datorerna. FPGAn funktion är enklare varianter av de funktioner som sitter i hjälpkretsarna *Blittern* och *Koppern*.

adressbuss. Det innebär att den kan adressera upp till 1Mb. Det är inom denna adressrymd vi måste kunna adressera alla våra kringkomponenter, eftersom 68008 helt saknar I/O-portar och endast kommunicerar via sin 8-bitars asynkrona adress-/databuss.

Alla komponenter skulle dela på styr-, adress- och databuss. Logik för styrbussen skulle också delas för att inte använda mer komponenter än nödvändigt. Då vår FPGA även var tänkt att agera master på bussen krävs det att det inte existerar två konkurrerande system för styrlogiken.

Ett enkelt och lockande sätt (eftersom vår FPGA är en mycket avancerad programmerbar krets) hade varit att dra in alla signaler från processorn och bussen på FPGAn och låta den hantera alla kontrollsignaler. Men, eftersom FPGAn vid denna tidpunkt innehöll många frågetecken förlitade vi oss på en för detta ändamål välkänd krets – PAL22V10.

Tack vare PAL-kretsen skulle elementära funktioner såsom att skriva och läsa till SRAM fungera i konstruktionen utan en konfigurerad FPGA.

Displayenheten bestod av en 128x64 pixlars LCD-display av typen Batron. Denna drivs av 2 st KS0108B drivkretsar. Dessa innehåller displayminne samt logik för att presentera detta på Batrondisplayen. Drivkretsarna har en insignal som anger om indatan är displaydata eller en instruktion.

Data	**Instruktion**
3E	Display av
3F	Display på
40-7F	Y adress (0-63)
B8-BF	X adress (0-7)
C0-FF	Y startlinje (0-63)

Table 1Instruktioner till LCD drivkrets KS0108B

I PAL-kretsen defineras minnesmappningen. Eftersom det är 4 olika enheter som det ska kunna genereras separata styrsignaler till kräver det minst 3 adressbitar från processorn. Vi valde att dra in de 3 översta bitarna (A17-A19). LCD-displayen är uppdelad i 2 identiska delar. Dessa är i sin tur uppdelade i data och instruktions areor. För att spara på utgångar från PALen virades istället A16 till D/I-ingången på displayen som avgör om det är instruktion eller data. Ut från PALen gick då bara en chip-select till var LCD.

PALen översätter också de olika asynkrona signalerna från SRAM till processor. CS, OE och WE till SRAM, och DTACK till processorn.

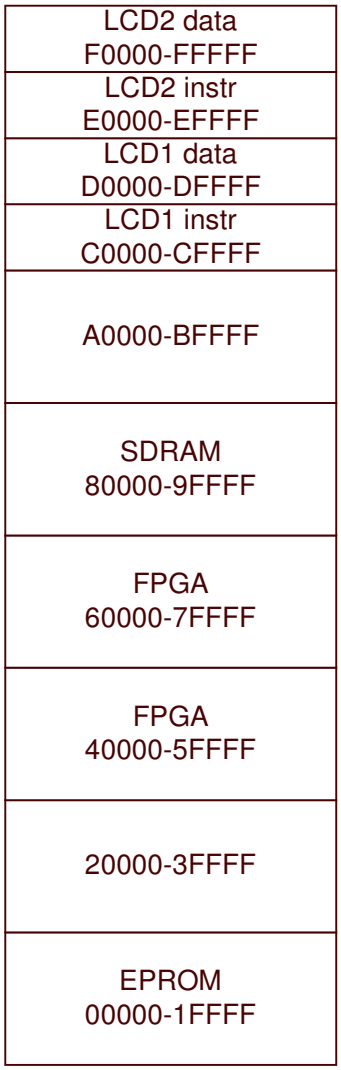


Illustration 2Adressareor

UTFÖRANDE

KOMPONENTPLACERING

Vi började med att placera ut komponenterna på kortet. Processorn spelar en central roll och blir således placerad så att övriga komponenter kan placeras runt denna. FPGAn med sina många ben placerades på en lite större yta för sig själv. SRAM som är gemensamt för både processor och FPGA försöktes placeras mellan dessa. Slutligen sattes kontakten till LCD-displayen nära FPGAn.

Vi byggde en oscillator enligt applikationsexempel. Med hjälp av logikanalysatorn undersökte vi signalen så att den verkligen var 8 Mhz. Det visade sig att oscillatorn oscillerade på fel frekvens, samt att den var väldigt instabil. Detta visade sig bero på en feldimensionerad kondensator.

Härnäst följde virning av adress- och databussar. I alla dragningar av dessa utgick vi från processorn för att hålla ledningar så korta som möjligt. Även klockan och övriga styrsignaler virades, och alla klocksignaler utgick från oscillatorn för att få en så stabil referens som möjligt.

Innan uppkopplingarna kunde testas måste PALen programmeras. Detta gjordes genom att assemblera pld-filen i appendix A, och sedan programmera den med binär-filen. Detta upprepades 2-3 gånger till det att vi kunde verifiera att kretsen uppförde sig som förväntat med logikproben.

När vi nu testade våra komponenter från utvecklingssystemet fungerade SRAMet som det skulle. Däremot verkade LCD-displayen vara helt död. Det visade sig att vi saknade logik för att hantera det synkrona display-gränsnittet. Detta löstes med 2 J-K vippor (74HC73) och en utgång på PALen. Nu fick processorn VPA-signalen (som talar om att periferienheten är synkron) när den skrev eller läste på adresserna till LCD-displayen.

PROGRAMMERING

GPU

Det absolut svåraste momentet var att programmera och konfigurera FPGAn. Den skall både klara av att processorn läser och skriver till den, samt göra dessa operationer själv, utan att komma i konflikt med processorn.

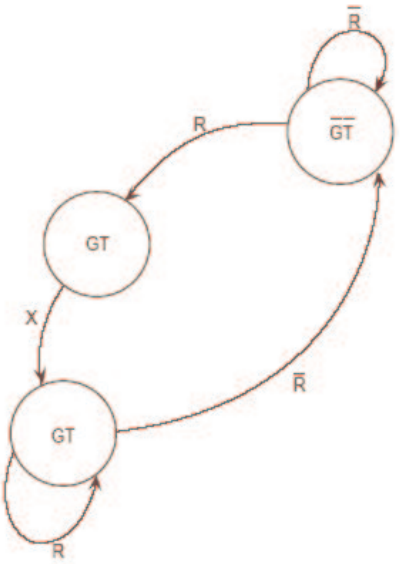


Illustration 3Tillståndsdiagram för busshanteringen på 68008

För att kunna göra detta krävs att man följer 68008 busshanteringsprotokoll, med signalerna BR och BG (se R resp GT i Illustration 3). Bara en enhet kan vara bussmaster åt gången, och driva adress- och databussen. Den andra enheten måste sätta bussen i högimpendivt läge och lyssna/vänta. Ett tillståndsdiagram för busshanteraren i GPUn återfinns i Illustration 4. I 'skriv'- och 'läs'-tillstånden skriver respektive läser GPUn. I de andra agerar den som en vanlig periferienhet.

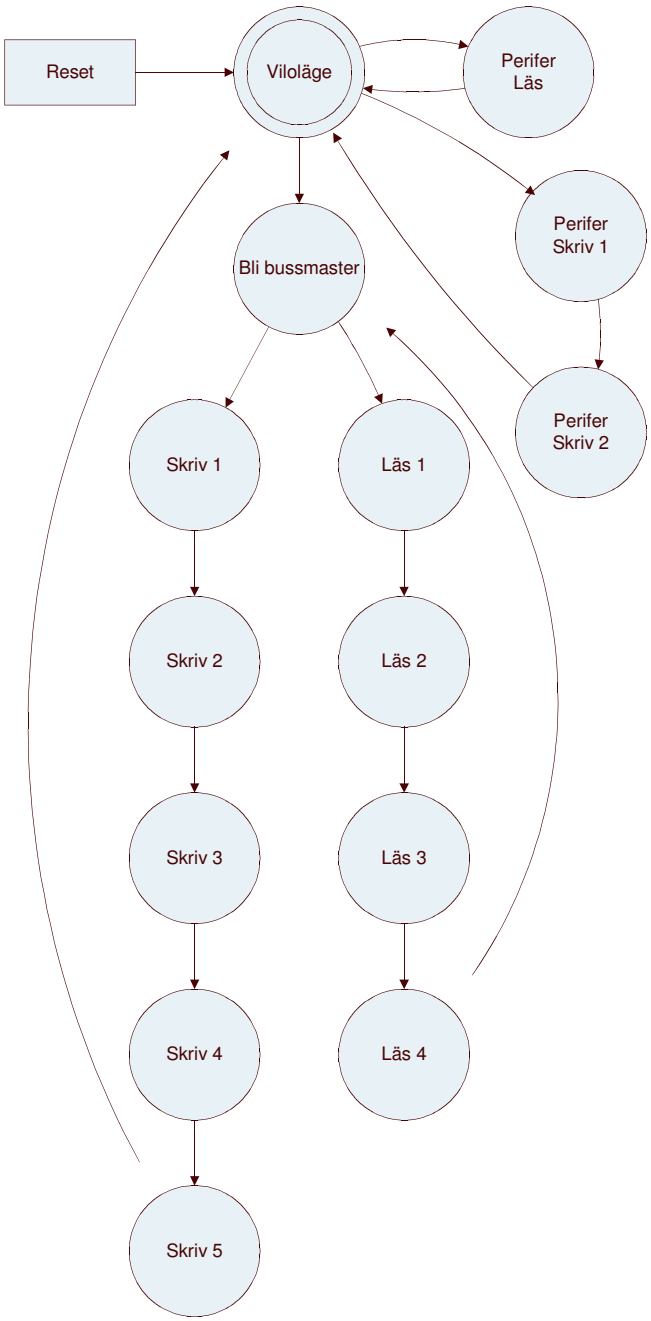


Illustration 4Tillståndsdiagram, busshanteraren GPU

GPUn beskrivs i VHDL-kod som modellerar den i komponenter, bestående av tillståndsmaskiner. VHDL-koden kompileras (Illustration 5) och en nätlista genereras. Nätlistan går igenom några optimerande steg via placeringsalgoritmer specifika för FPGA kretsen i fråga innan den laddas ner i det inbyggda SRAMet på FPGAn.

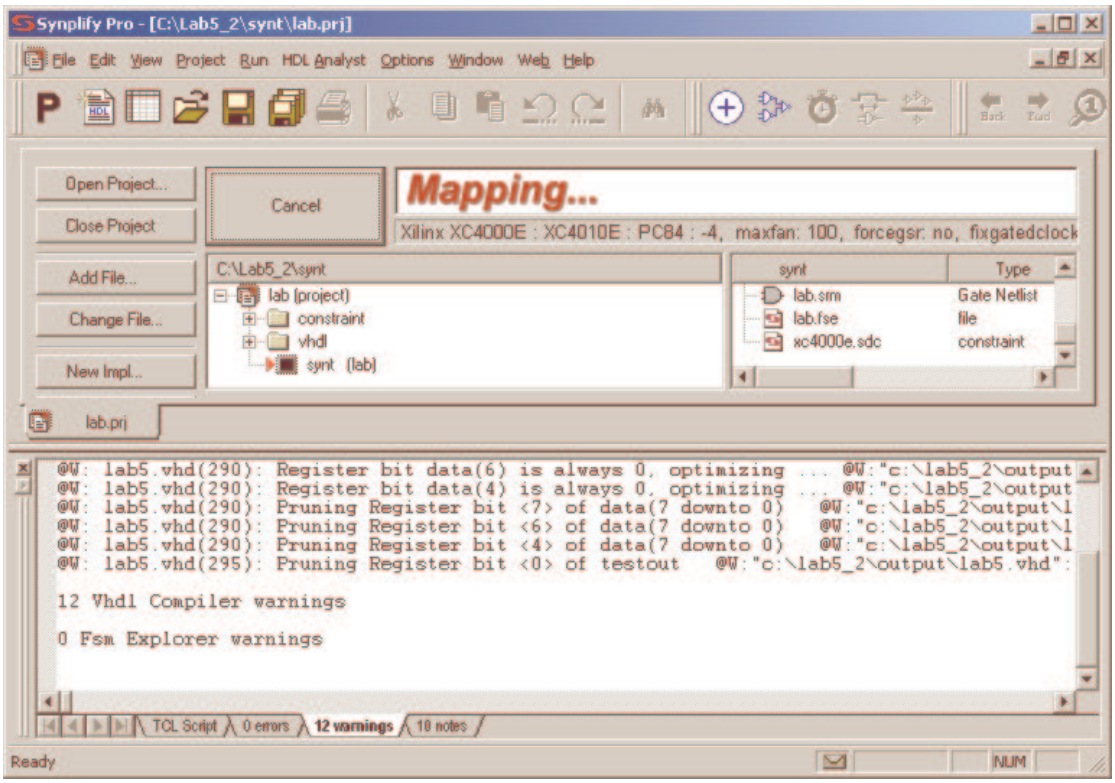


Illustration 5Kompilering och mappning av VHDL-kod

MCU

Programmet för processorn är mycket enkelt och har som ändamål just att visa funktionen hos GPUn. Den ska initalisera GPUn, skriva till SRAM, starta GPUn och iterera detta.

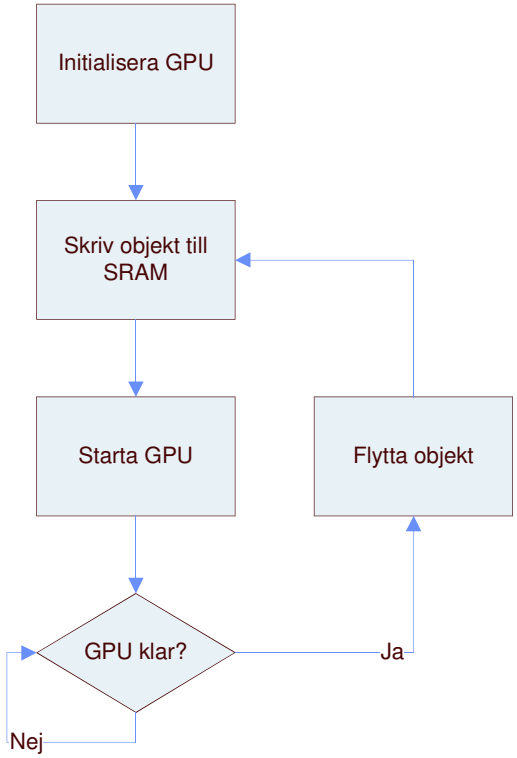


Illustration 6Flödesschema över program i microprocessorn

RESULTAT

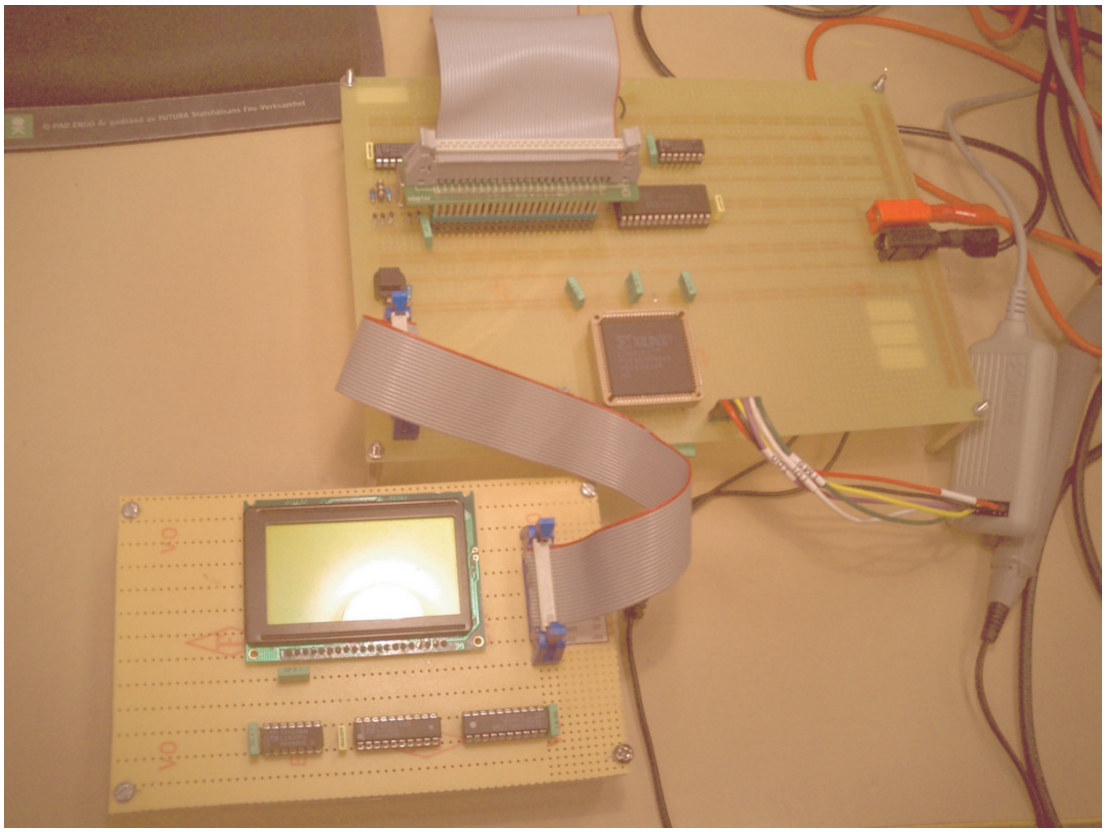


Illustration 7Konstruktionen i sin helhet

Då slutparten av vårt omskrivna testprogram håller på att avslutas kan vare sig prestanda eller funktion på konstruktionen i sin helhet visas.

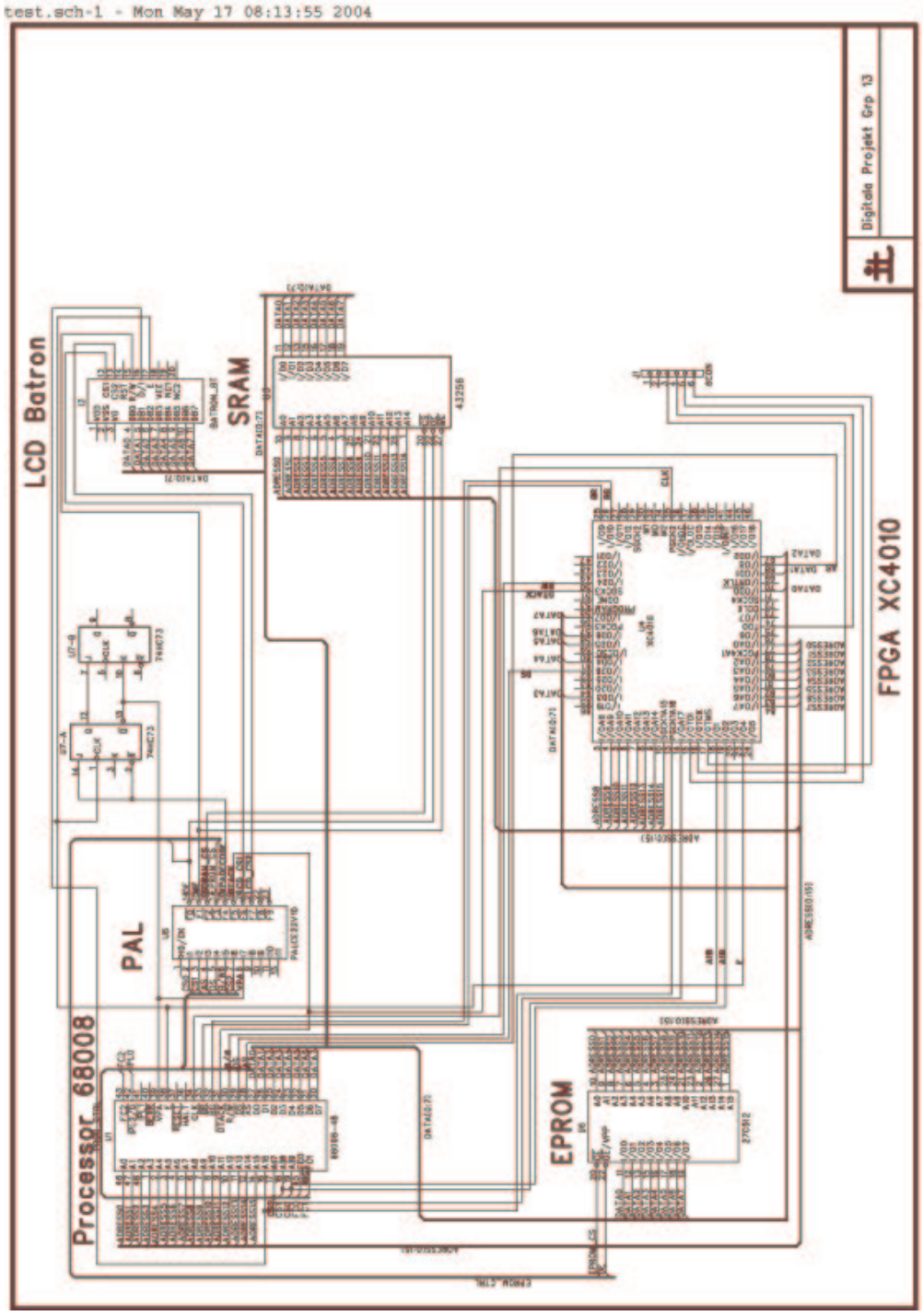
Konstruktion i sin helhet ses dock i Illustration 7. Inte helt förvånande var uppsättning och konfiguration av FPGAn besvärligare än väntat. Men resan var väldigt rolig. Dålig planering av projeketet resulterade i ett ganska stressigt avslut.

Referenslista

IT-68, Utvecklingssystem för 68008
MC68000 User Manual
XC4000E and XC4000X Series Field Programmable Gate Arrays
Xilinx Application Note – Boundary-scan in XC4000, Spartan and XC5200 Series Devices
KS0108B 64ch Segment Driver For Dox Matrix LCD

APPENDIX

A. KRETSSCHEMA



B. PAL CONFIGURATION – PAL22V10_1.PLD

```
Title MemUnit of 68k proj

device 22V10
  'CLK                1
  CS0                 2   'Chip select 0
  CS1                 3   'Chip select 1
  AS                  4   'Address strobe, active low
  DS                  5   'Data strobe, active low
  RW                  6   'Read/Write, write - active low
  CS2                 7   'Chip select 2
  VPA                 8   'VPA from ext logic to 68k
  I8                  9
  I9                  10
  I10                 11
  GND                 12
  I11                 13
  OE                  14  'Output Enable
  WE                  15  'Write Enable, active low
  SDRAMCS             16  'SDRAM Chip Select
  EPROMCS             17  'EPROM Chip Select
  VPADECODE           18  'LCD VPA decode to ext logic
  DTACK               19  'ACK to 68k
  LCDCS1              20  'LCD Chip select 1
  LCDCS2              21  'LCD Chip select 2
  IO8                 22
  IO9                 23
  VCC                 24

' ---------- Memory Map ----------
' CS0 | CS1 | CS2 | Hex | Device |
' ----|-----|-----|-----|--------|
'  0  |  0  |  0  | 0,1 | EPROM  |
'  0  |  0  |  1  | 2,3 |   -    |
'  0  |  1  |  0  | 4,5 | FPGA-D |
'  0  |  1  |  1  | 6,7 | FPGA-C |
'  1  |  0  |  0  | 8,9 | SDRAM  |
'  1  |  0  |  1  | A,B |   -    |
'  1  |  1  |  0  |  C  | LCD1-I |
'  1  |  1  |  0  |  D  | LCD1-D |
'  1  |  1  |  1  |  E  | LCD2-I |
'  1  |  1  |  1  |  F  | LCD2-D |

start

OE /= /DS*RW;

WE /= /RW*/DS;

'32k  SDRAM @ 0x80000-0x87fff
SDRAMCS /= CS0*/CS1*/CS2*/AS;

'64k  EPROM @ 0x00000-0x0ffff
EPROMCS /= /CS0*/CS1*/CS2*/AS;

VPADECODE = CS0*CS1*/AS;

'Disable DTACK when the FPGA is addressed
DTACK.ENA = /CS1*/CS2*/AS;
DTACK /= /SDRAMCS + /EPROMCS + /CS0*CS1*/AS;
```

```
'LCD1 Instr = 0xC0000 and Data = 0xD0000
LCDCS1 /= /VPA*CS0*CS1*/CS2;

'LCD2 Instr = 0xE0000 and Data = 0xF0000
LCDCS2 /= /VPA*CS0*CS1*CS2;

end
```

C. FPGA VHDL EASE-DIAGRAM