Institutionen för Informationsteknologi

Projektrapport i Digitala projekt

Handledare: Bertil Lindvall

HANDDATOR

Projektet utfört av grupp 02:

Mathias Johansson

Marcus Löfgren

Datum: 2001-05-23

Abstract

The goal was to build a small, rather general, computer with similar funktions as a palmpilot of today. This was to be done with the Motorola 68008 processor. The main specified demands was graphical user interface, ability to communicate with a PC, recordning and playback of sound and a small operating system that allows the user to select which program to be run.

A graphical user interface was accomplished with a LCD-display and program menues. PC communication over the serial interface RS-232 was quite easily implemented with an UART-circuit.

To obtain the ability to play and record sound without suffering from the limitations of the processor, some sort of DMA (Direct Memory Access) was to be constructed. This was done with a 512 kB RAM, a 84-pin Lattice and a handfull of counters. Obviously a A/D-converter and a D/A-converter are also needed. A state machine and a byte counter was implemented in the Lattice. The external counters keeps track of the memory address. When the request comes for, say playing a sound, the right state is obtained if the parameters are correct. The data on the adress, supplied by the address counters, are then played as long as the byte counter allows it to be.

The simple operating system is not actually an OS. It is more like a boot menu that allows the user to choose among the stored programs, which one to run for the moment.

Most of the demands of specification was fullfilled but some demands, like the stereo sound and a calendar program, were left out because of lack of time and because their implementation was not necessery for the project. Everything else was constructed according to plan and it is working too!

Innehållsförteckning

1 Inledning

1.1 Presentation av projektet

Målet, för grupp 02, med Digitala projekt var att framställa en fungerande prototyp av en generell handdator. Denna skulle ha flera egenskaper, bl a in- och uppspelning av ljud och kommunikation med PC. Den exakta kravspecifikationen av handdatorn finns att läsa i appendix 1. Där framgår det också att mjukvarusidan inte specificerades lika hårt som hårdvarusidan. Detta beror på att det efter färdigställandet av hårdvaran går att skriva ett antal mer eller mindre vettiga program som alla kan köras på handdatorn. Därför lämnades detta till att genomföras i mån av tid. Vissa program är dock angivna, t ex ett minimalt operativsystem.

En så generell konstruktion som var i åtanke går att utöka i princip hur mycket som helst. Det är bara en fråga om att komma på nya applikationer. Detta har i viss mån begränsats av processortypen, en Motorola 68008. Tiden har varit en annan kraftigt begränsande faktor. Detta märktes tydligast på att det aldrig fanns tid att försöka implementera några av de olika idéer till utökning, som uppstod fortlöpande under projektet. Andra begränsande faktorer har varit brist på passande kretsar och att logiken ökat till oväntade storlekar. Det senare yttrade sig mest som brist på tillgängliga pinnar på logikkretsarna. Detta trots att närmare 160 pinnar fanns tillgängliga från början!

1.2 Rapportens upplägg

Rapporten tar börjar med att i kapitel 2 ta upp själva konstruerandet, d v s komponentval, schemaritning mm. Kapitel 3 ägnas åt funktionen av handdatorns olika delar. I kapitlet tas hårvara upp för sig och mjukvara för sig i två stora block. Det bör sägas att inte varje liten tråd förklaras och inte heller minsta variabeldeklaration i programmen. Det som tas upp är de bitar som författarna ansett vara väsentliga – både ur läsarens synpunkt och ur konstruktionssynpunkt.

I kapitel 4 utvärderas konstruktionen och projektet i sin helhet. Här tas en del problem upp, som uppstått under projektets gång. Avstegen från kravspecifikationen tas också upp.

De sista två delarna av rapporten är litteraturreferenser och bilagor. Den tilltänkta läsaren har förmodligen stött på de flesta av referenserna. Bilagorna bör studeras i samband med läsning för att ge ökad förståelse. En förteckning över dem kan ses i innehållsförteckningen.

2 Genomförande

2.1 Val av konstruktion

Redan innan kursstart så stod det klart att det skulle bli en egen konstruktion och inte ett av de förslag som finns på institutionens hemsida. Detta berodde dels på att inget av institutionens förslag lockade riktigt och dels på att det var mer intressant att göra något som inte var så vanligt, som t ex nummerpresentatör. Diskussionen gick från spelkonsol till sampeloscilloskop och till slut blev valet en relativt generell dator, som skulle ha möjligheter liknande en primitiv palmpilot. Denna skulle ha en grafisk display och bara ett fåtal knappar för att styra applikationerna. Den skulle också sampla ljud, spela upp ljud och kommunicera med en PC. Dessutom skulle den ha någon form av operativsystem som tillåter användaren att välja vilket av, de i handdatorn lagrade, programmen som ska köras.

Fördelen med detta var att med en generell hårdvara så kunde sedan de flesta andra applikationer implementeras m h a mjukvara. Tidsaspekten för kursen satte dock en snäv gräns för antalet applikationer, som kunde tänkas vara färdigställda vid kursens deadline.

Den färdiga kravspecifikationen kan ses i appendix 1.

2.2 Komponentval

Fullständig komponentförteckning kan ses i appendix 6.

Då applikationerna, nämnda i föregående avsnitt, skulle kräva relativt mycket minne fick valet av processor bli Motorola 68008. Det andra alternativet, 68HC11, kunde inte adressera lika mycket minne och var därför uteslutet. Till RAM-minne ett SRAM på 512 kB. Som ROM-minne valdes ett flashROM på 512 kB, för att ge möjlighet åt en funktion som kunde lagra ev data från kommunikationen med en PC. Ett flashROM tillåter segmentvis omflashning av minnet, utan att påverka övrig lagrad data, och kan i princip fungera som en miniatyrhårddisk.

Då M68008 bara kan adressera 1 MB minne och då RAM och flashROM tillsammans tar upp 1 MB minne, betyder detta att några adresser måste tas från minnena för att kunna adressera andra kretsar. Detta minskar inte lagringsförmågan i någon nämnvärd utsträckning och är både acceptabelt och nödvändigt.

Logiken för adressavkodning och skapande av chipselect- och enablesignaler kunde antingen implementeras direkt på grindnivå eller m h a PAL:ar eller m h a Lattice-kretsar. Att bygga logiken på grindnivå var egentligen inget alternativ, med tanke på den mängd logik som fordrades. Därmed stod valet mellan PAL och Lattice. Det stod redan klart att det skulle behövas ett flertal PAL:ar och efter diskussion med handledare så föll valet på Lattice-kretsarna. Dessa är flexibla och kan bl a implementera både räknare och tillståndsmaskiner, förutom den vanliga logiken.

Den grafiska displayen skulle helst vara så stor som möjligt eftersom vi bl a ville kunna spela spel på handdatorn. Institutionen tillhandahöll en LCD-display på 128x64 punkter, som räckte väl för en prototyp.

Det var från början tänkt att endast åtta tangenter skulle finnas på handdatorn. Av dessa skulle fyra vara riktningstangenter för att styra markörer, spelare mm. De resterande fyra tangenterna skulle vara applikationsspecifika och därmed ha olika funktion för olika program. Det visade sig dock att det enklaste sättet att bygga detta på var att använda tangentbord med 16 tangenter och en avkodare. Detta minimerade besvär med att själv behöva skapa egen tangentavkodning och var inget stort avsteg från specifikationen.

En realtidsklocka var också nödvändig, dels för ev kalender- och klockprogram, dels för att kunna ställa antal interrupt per tidsenhet.

Systemklockan konstruerades efter ritningen på institutionens hemsida.

De komponenter som hittills tagits upp utgör stommen i konstruktionen och det går med dessa delar att skapa en liten dator och köra program som styrs med tangenterna och som visar resultat på displayen. Dock saknas kommunikation utåt, såvida man inte använder emulatorn.

Till dataöverföring krävdes en UART för kommunikation till serieporten på en PC. I mån av tid var ett experiment med USB av intresse.

Övriga funktioner som skulle finnas var in- och uppspelning av ljud. Dessutom skulle möjligheten finnas att använda ingången till annat än inspelning av ljud, t ex som simpel oscilloskopingång. För att avlasta processorn och för att få större tillgång till minne, valdes att bygga en simpel DMA-funktion (Direct Memory Access). Så förutom en A/D-omvandlare och en D/A-omvandlare så behövdes nu även ett extra I/O-minne, adressräknare, byteräknare och styrlogik i form av en andra Lattice-krets. En buffert efter D/A-omvandlaren var dessutom nödvändig för att orka driva en högtalare. I/O-minnet valdes till 512 kB, byteräknaren implementerades i Lattice-kretsen och adressräknaren byggdes upp m h a fem 4-bitars räknare. Bufferten efter D/A-omvandlaren består av en OP-förstärkare kopplad till en gångs förstärkning.

2.3 Schemakonstruktion

Schemat ritades i programmet PAD:s Power Logic och kan ses i appendix 2. Vissa komponenter fanns inte inlagda i PAD:s bibliotek och skapades därför på egen hand. Detta gäller Lattice-kretsarna och flashROM:et. Dessa är därför inte tillgängliga för någon som ev skulle vilja använda dem i framtida konstruktioner, utan ligger i den katalog som tilldelades grupp 02.

Det bör också nämnas att pinnumreringen på schemasymbolen för Lattice-kretsarna inte stämmer med verkligheten. Detta beror på att kompilatorn kan ändra om pintilldelningen vid slutkompilering och om inte detta kontrolleras så kommer det förmodligen att hända till någon utsträckning.

2.4 Logik

Logiken implementerades helt i två 84-pinnars Latticekrestsar. Den andra kretsen styrde DMA-delen, d v s I/O-minne, ADC, DAC och räknare, och den första kretsen styrde allt annat.

Vid programmering av Lattice-kretsar används ett tillhörande utvecklingsprogram som i princip består av en grafisk del och en textbaserad del. I den grafiska delen ritas scheman i samma stil som i vilket schemaritningsprogram som helst. I den textbaserade delen skrivs de logiska samband som ska genomföras av kretsen och in- och utgångar till dessa samband skapar en ”låda” som sedan kan användas i den grafiska delen. Det är i den grafiska delen som de fysiska anslutningarna på kretsen kopplas till en ”låda” med egna kommandon.

Den första Lattice-kretsen, Lattice 1, som styr chipselectsignaler mm för kretsarna direkt runt processorn, är helt kombinatorisk. Den andra Lattice-kretsen, Latice 2, innehåller både kombinatorik och en tillståndsmaskin. Tillståndsmaskinen styr vad som ska hända och när, vid en sampling eller en ljuduppspelning.

För kod och grafiska scheman till Lattice-kretsarna, se appendix 4. Funktionen hos DMA-delen förklaras närmare i avsnitt 3.1.3.

2.5 Komponentplacering

Vid placering av komponenterna på kretskortet så sattes komponenterna i så stor utsträckning som möjligt, efter grupptillhörighet. Det betyder att processorn, RAM, flashROM och Lattice 1 sattes intill varandra. Likadant gjordes för resten av kretsarna. Detta för att minimera kabellängd och lättare kunna hålla reda på vilka kretsar som skulle kopplas samman.

Ett bra exempel på vikten av att hålla ordning på kretsarna var DMA-delen. Denna innehöll intern databuss och adressbuss, som båda var skilda från den vanliga data- respektive adressbussen. En avskild komponentplacering gav en bättre överblick och snabbade upp sammankopplingen.

I efterhand upptäcktes att vissa placeringar kunde ha varit smartare – t ex bör det lämnas plats runt kretsar där IC-utdragare kommer att användas.

2.6 Programmering

Diverse program var nödvändiga för konstruktionen. Av de nödvändigaste programmen kan nämnas grafikrutiner och rutiner för att visa text på displayen. Dessa program skrevs först för att sedan kunna utnyttjas i efterföljande program och därmed underlätta fortsatt programmering. Grafik- och textrutiner skrevs i Assembler och större nyttoprogram skrevs i C. Nackdelarna med Assemblerprogrammering upptäcktes när avlusning av programmen skulle göras; de är svårare att felsöka i. C-programmeringen var lättare att överskåda men tillåter ju fortfarande programmeraren att göra lågnivåfel.

3 Resultat och funktion

3.1 Hårdvara

Med små variationer så har i princip hela hårdvaruspecifikationen uppfyllts. Hela konstruktionen bygger, som nämnts tidigare, på en M68008 processor. Denna klockas i 10 MHz av klockan som finns angiven på institutionens hemsida. Som minne finns i princip 512 kB ickeflyktigt minne i form av flashROM och 512 kB SRAM. Den specificerade uppdelningen av minnesarean kan ses i appendix 6.

Handdatorn styrs av ett 16-knappars tangentbord med tangenavkodare och resultaten visas på en 128x64 punkters LCD-display. För att kunna få klockslag och datum finns en realtidsklocka, dock utan backupbatteri i prototypen. Vidare så har den en UART för kommunikation via RS-232 till PC. En 8-bitars lysdiodsvektor med tillhörande latch finns också för att kunna se att t ex tangentnedtryckningar registreras. En viss effektfullhet, som ges av dioderna, kan inte förnekas. Dessa komponenter styrs alla av en 84-pinnars Lattice-krets (denna kommer i fortsättningen refereras till som Lattice1), som bl a avkodar adressbussen för att kunna skapa chipselect-signaler och signaler för läsning och skrivning.

För sampling och uppspelning av ljud finns A/D-omvandlare respektive D/A-omvandlare. Dessa är anslutna i DMA-delen, som klarar av att spela upp eller sampla ljud utan att belasta processorn med annat än initieringen av respektive händelse. DMA-delen är uppbyggd av ett 512 kB SRAM, 5 stycken 4-bitars räknare och en 84-pinnars Lattice-krets (denna kommer i fortsättningen refereras till som Lattice2). En buffert i form av en OP, kopplad i enhetsförstärkning, finns även. Denna sitter efter D/A-omvandlaren för att driva högtalare.

3.1.1 LCD-display

Displayen utgörs av två seperata controllers som vardera sköter 64x64 punkter, placerade brevid varandra. Pixeldata latchas internt, 8 vertikalt intilliggande pixlar per byte och bytedata addresseras med två interna register, för rad och kolumn. För att skriva eller läsa bilddata från displayen, samt kontrollera funktioner som på/av samt dataadress används ett status/kontroll-register och ett dataregister för vardera kontrollern. Processorn ser alltså displayen som totalt fyra byteregister i addressrymden. För närmare beskrivning av LCD-controllern hänvisas till databladen för kretsen (KS0108B) under Display/Drivers-rubriken på kursens hemsida.

3.1.2 Seriekommunikation med UART

För seriekommunikation med PC via standardgränssnittet RS-232 användes en UART-krets av modell National PC16550D. 16550 är i det närmaste industristandard och återfinns bl a i så gott som alla PC-datorer. Kretsen har funktioner som 16 bytes FIFO-köer, stöd för DMA-hantering och givetvis interrupt. Handdatorn var först tänkt att utnyttja DMA-överföring vid kommunikation, enligt samma princip som ljuduppspelningen med en Lattice-krets som DMA-kontroller och det extra I/O-SRAM:et som lagringsutrymme. Detta reviderades dock så att processorn istället kontrollerar UART-kretsen med mjukvara i form av interrupthanterare, då hårdvarulogik skulle blivit alltför komplex och säkerligen krävt ytterligare en extra Lattice-

krets, eller en mer kraftfull FPGA, medan prestandavinsten torde vara relativt försumbar. Processorn kontrollerar kretsen via åtta minnesmappade register, för detaljer hänvisas till databladen på kursens hemsida.

Mellan UART-kretsen och serieporten måste en anpassningskrets placeras. Detta för att omvandla logiska TTL-signalnivåer till RS-232standardens 12 V signalnivåer. En färdig krets från Maxim avsedd för just detta syfte användes.

3.1.3 Lattice 1

Den första Lattice-kretsen innehåller logik för funktioner som att generera chip-select och enable-signaler till minneskretsar och andra I/O-enheter, för att avläsa knapptryckningar och som interruptcontroller, samt innehåller en klockdelare som delar ner systemklockan till en ca 19 kHz klocka och en ca 20 Hz pulssignal. Syftet med dessa nerdelade klocksignaler förklaras nedan. Adresseringslogiken är bara enkel kombinatorik med adressbitar och R/W, DS och AS-signalerna från processorn som insignaler, som implementerar adressrymdsuppdelningen som bifogas i bilaga 6.

Det finns totalt tre interruptnivåer på MC68008, varav konstruktionen använder två, nämligen nivå 2 och 5. På nivå 2 ligger realtidsklockan ensam, och avbrottet är av autovektor-typ. Det innebär att när ett sådant avbrott kommer letar processorn automatiskt på en förutbestämd adress efter pekare till avbrottshanteringsrutinen. På konstruktionen finns en mängd andra enheter som också behöver kunna generera interrupt, men med bara två ytterligare nivåer att använda fick detta lösas med s.k. vektoriserade avbrott. Detta innebär att interruptkontrollern lägger ut ett åttabitars vektornummer på databussen när avbrottet accepterats av CPU:n, varvid nummret används som index i avbrottstabellen. Denna logik finns implementerad i Lattice 1, avbrottskällor som hanteras på nivå 5 är knapptryckning, DMA slutförd, UART-interrupt och USB-interrupt.

Knapptryckning genererar normalt sett ett interrupt när någon knapp trycks ner, därefter inga nya förrän samtliga knappar släppts upp och nästa knapp trycks ner. Senast nedtryckt knapp kan avläsas genom att läsa en byteadress som mappats till ett register i Lattice 1. Knapparna numreras 0-15 och bit 7 i denna byte är 1 (logiskt hög) om denna knapp är nertryckt vid avläsningstillfället.

För att underlätta styrning av spel eller grafisk markör eller liknande finns ett andra knapptryckningsläge. Genom att skriva en statusbyte (värdet 1) till samma adress man normalt avläser knappnummret från, ställs logiken i autorepetitionsläge. Så länge en knapp är nedtryckt genereras då nya knapptryckningsinterrupt ca 20 gånger per sekund. Detta är syftet med den 20 Hz nerdelade pulssignalen nämd ovan. Eftersom nerdelningen av 10 MHz systemklockan även gav mellanledssignalen 19 kHz leddes denna över till Lattice 2 för att styra DMA-timingen.

3.1.4 DMA

DMA står för Direct Memory Access och som namnet antyder är det en lösning för att kunna flytta (ofta stora mängder) data snabbt mellan I/O -enheter och minne, utan processorns inblandning. Detta är idealiskt för tillämpningar såsom ljuduppspelning och ibland i

grafiksammanhang om man exempelvis vill flytta runt stora grafikblock på en skärm. I denna konstruktion valdes DMA som metod att möjliggöra in- och uppspelning av ljud och då vi inte kunde hitta någon lämplig fristående kontrollkrets implementerades funktionen med programmerbar logik. För att undvika bussarbitreringsproblem på processorns databuss samt minimera resursbristen i form av tillgängligt minne används ett extra I/O-SRAM på 512 kb. Logiken, som är programmerad i Lattice 2, består av av en tillståndsmaskin som startas med ett kommando från processorn. En typisk överföring kommer att beskrivas kortfattat:

1. Initiering. Processorn skriver en startadress, var i I/O-minnet data börjar, till ett antal räknare som fungerar som adressregister. Adressen på 19 bitar är uppdelad i tre bytes och skrivs som vanlig data till respektive register. Processorn skriver därefter ett 16-bitars ord som anger antalet bytes som ska överföras. Detta ord lagras i en intern räknare i Lattice 2. Därefter kan överföringen påbörjas genom att processorn skriver till ett statusregister, också internt i Lattice2, med information såsom önskad samplehastighet (finns tre möjliga att välja mellan) och huruvida man önskar spela upp eller spela in ljud.

2. DMA pågår. Kretsen genomlöper nu en tillståndsloop där läs- eller skrivsignaler genereras till I/O-minnet, motsvarande signaler genereras till DAC eller ADC-kretsarna, varvid data överförs. Adressräknarna fås att räkna upp till nästa adress, byteräknaren räknar ner, och om inte byteräknaren räknat ner till noll upprepas proceduren efter att nästa samplepuls inväntats. Samplehastigheten avgörs genom att en nerdelad klocksignal inväntas. Man kan genom tidigare nämda statusregister välja om man vill använda en sampleklocka på ca 19 kHz, ca 10 kHz eller ca 4,5 kHz. 19 kHz-signalen genereras genom nerdelning av systemklockan i Lattice 1, och vid behov delas ner ytterligare en eller två gånger i Lattice 2. 10 kHz anses som lämpligast vad gäller avvägningen mellan ljudkvalité och storlek på sampledata.

3. DMA slutförd. När överföringen är avslutad genereras ett interrupt till processorn och tillståndsmaskinen återgår till starttillståndet i väntan på eventuell initiering av nästa överföring.

Här kan tilläggas att det finns ytterligare två finesser inbyggda i Lattice 2, framför allt avsedda vid inspelning av ljud. Då byteräknaren används för att ange hur många bytes man önskar överföra blir största möjliga operationen på 64 kb data (16 bitars räknare). För att underlätta inspelning kan man i statusregistret ange att man vill att överföringen ska fortsätta tills hela I/O-minnet har addresserats. På så sätt kan man överföra maximalt 512 kb data i samma transaktion.

Det finns, bara vid inspelning, ett ytterligare samplehastighetsval nämligen maximal hastighet. Detta är tänkt som ett alternativ för funktion som ett primitivt digitalt oscilloskop, och överför data med samma hastighet som A/D-omvandlaren kan leverera nya värden, enligt databladet varje 800 ns.

Ett tillståndsschema för Lattice 2 kan ses i appendix 5.

3.1.5 Seperat I/O-minne

Det extra I/O-SRAM:et på 512 kilobyte ligger naturligtvis utanför processorns adresseringsområde, då MC68008 endast kan adressera en megabyte. Nackdelen med denna lösning är att det blir lite svårare för processorn att arbeta mot detta minne, men det är fullt

möjligt. Processorn måste bara skriva önskad I/O-minnesadress på samma adressräknare som används vid DMA enligt kap. 3.1.4 och därefter kan byten på denna adress läsas eller skrivas genom en vanlig 8bitars läs- eller skrivoperation på ett Lattice2-register avsett för detta ändamål. För att undvika att behöva göra upp till tre adressskrivningar för varje byte, förutom själva operationen på bytedata, finns ett autoinkrement-läge. Detta aktiveras genom att skriva en särskild bit i statusregistret på Lattice 2, vilket nämdes i samband med DMA-initiering. Då autoinkrement är aktivt kommer varje läs- eller skrivoperation på en byte i I/O-minnet medföra, att adressräknarna automatiskt räknar upp ett steg till nästföljande byte. På så sätt underlättas överföringar mellan processorns minne och I/O-minnet, t ex för att läsa 256 bytes från I/O-minnet i autoinkrementläget behövs bara en initiering av adressregisterna så de pekar på första byten, därefter görs 256 byteläsningar från SRAM-registret i Lattice 2 och datan kan behandlas som önskat av processorn.

3.2 Mjukvara

Det största målet med mjukvaran i det här projektet är att implementera ett delat system med dels en maskinnära "operativsystem"del och dels mer eller mindre fristående applikationer. Operativsystemet är permanenta rutiner som startar upp datorn och sköter all nödvändig initiering, sköter all interrupthantering på låg maskinnära nivå, tillhandahåller grafiska primitiver och textutmatningsrutiner avsedda för LCD-displayen och andra rutiner som kan vara av nytta för applikationsprogrammeraren, samt ett menyprogram för att hantera visning, val och start av tillgängliga applikationer. Applikationerna skall kunna skrivas och kompileras som fristående program, med tillgång till systemrutinerna genom en enkel #include-sats i programkoden.

Lagring och hantering av applikationsprogramen i handdatorn sker genom uppkoppling mot ett terminalprogram på PC, i vilket programfilerna sammanställs och skickas till handdatorn, som lagrar binärdatan i sitt eget flash-rom.

3.2.1 Initiering och interrupt

Ett litet program som alltid körs efter reset av maskinen och initierar register, avbrottstabell, globala pekare och liknande är nödvändigt. Detta är skrivet i assembler för maximal flexibilitet och inkluderar även små användbara rutiner som att sätta interruptprioritetsnivå i processorn, samt FLASH-minnesrutiner som beskrivs på annan plats i detta kapittel.

Det viktigaste som görs i denna del av systemet är lågnivåinterrupthanteringen. Vid en knapptryckning exempelvis, anropas en liten rutin som från hårdvaran läser in vilken knapp som tryckts, detta fungerar även som acknowledge till Lattice 1-logiken som därmed anser interruptet behandlat. Men assemblerrutinen tittar nu efter i en global pekar-variabel och om denna pekare är skild från noll anropas adressen. Detta gör det enkelt för applikationsprogram att implementera egna högre nivåer av interrupthantering, allt som behövs är att den globala variablen tilldelas en pekare till rutinen man önskar anropad. (Påpekas bör, att det är absolut väsentligt att nollställa dessa pekarvariabler i samband med att applikationen avslutas!)

3.2.2 Grafik- och textrutiner

Då den faktiska uppbyggnaden av LCD-skärmen, med två seperata skärmhalvor kontrollerade av var sin styrkrets, vore tidsödande och jobbig att accessa direkt används ett system där en minnesarea i det vanliga dataminnet används som mellanlagring. Denna area har samma pixel-ordning som LCD-skärmen, åtta vertikala pixels per byte men hela skärmen, 128 bytes per rad i X-led, ligger konsekutivt i minnet. Dessutom används två extra byterader, en ovanför respektive under den egentliga skärmarean, för att minska clipping-problemen i grafikrutinerna. Det gör alltså inget att det i vissa fall ritas upp till 8 punkter ovanför eller under den egentliga skärmkanten. När applikationsprogrammet med hjälp av grafikrutinerna åstadkommit ett önskat bildmönster i detta skärmminne anropas en uppdateringsfunktion som överför pixeldatan för hela bilden till LCD-skärmen på en gång.

Många av grafikrutinerna kräver en parameter som är en pekare till ovan nämda skärmarea. En standardarea tillhandahålls som global variabel av systemet och oftast räcker det att ange denna, men för maximal flexibilitet är det alltså möjligt att låta rutinerna rendera sina resultat i godtyckligt minnesområde.

På samma sätt kräver t.ex textrutinerna att en pekare på en typsnittstruktur skickas med. Systemet tillhandahåller för närvarande tre olika standardtypsnitt, där alla ASCII-tecken mellan 32 och 255 är definierade. Pekarna finns tillgängliga som globala variabler och typsnitten ligger permanent tillgängliga i operativsystemet, men en applikation kan även inkludera sina egna typsnittsstrukturer och alltså enkelt använda dessa med standardrutinerna. Ett enkelt PC-program som omvandlar godtycklig TrueType(tm)-font till typsnittsdata i rätt format har utvecklats.

Slutligen finns det ett par rutiner för att rita s.k sprites, figurer av godtycklig skepnad. Som en av parametrarna till dessa skickas en pekare på en datastruktur som beskriver bilden. För att underlätta för applikationsprogrammeraren har ännu ett enkelt PC-program som omvandlar svartvita bmp-filer till spritedata utvecklats. I systemet finns 18 stycken små standardsprites inkluderade, åtkomliga genom globala pekare. Dessa föreställer generellt användbara saker som ikryssade respektive tomma rutor, Ja och Nej-symboler och liknande.

3.2.3 Filsystem och terminalprogram på PC

Applikationsprogrammen skrivs och kompileras med Intertools programvara, förslagsvis C. Genom att bara inkludera en headerfil som definierar pekare till alla system-rutiner och -variabler kan man indirekt komma åt dessa från applikation utan att behöva länka varje enskilt program med systemrutinerna. Efter kompileringen fås som vanligt en hex-fil med kod i motorolas S-format. En särskild rutin för att avkoda detta format till binär körbar kod används i terminalprogrammet på PC-sidan, med tanken att det ska vara enkelt att välja vilka hex-filer man vill lagra på handdatorn, och dessa kommer laddas in och avkodas automatiskt. Denna finess är dock i skrivande stund helt otestad och som alternativ letades ett freeware-program upp på nätet som genererar motsvarande binärfiler.

Terminalprogrammet på PC-sidan är lite av ett kontrollcenter, varifrån anslutning av handdatorn via serieporten kan ske. Härifrån kan synkronisering mot PC-klockan ske, nya program och operativsystemversioner laddas in för nerladdning till handdatorn. Ursprungligen var det tänkt att kalenderprogram och någon form av adress/telefonbok på handdatorn skulle

kunna synkroniseras med PC-motsvarighet här, samt kanske eventuellt flerspelarläge i något spel skulle kunna implementeras genom att ena spelaren spelade på en PC-versionen av spelet. Detta hann dock inte förverkligas, men möjligheterna är goda.

På handdatorsidan lagras applikationerna i ett primitivt filsystem i flashrom:et. Detta är i princip en länkad lista med en uppsättning binärfiler enligt ovan för varje applikation, tillsammans med ett filnamn för varje element i listan. Möjlighet till komprimering av varje applikations binärdata för att kraftigt minska förbrukningen av tillgängligt flashminne per applikation, finns inlagd i filsystemet. Ambitionen är att anpassa en version av det fritt tillgängliga komprimeningsbiblioteket zlib till handdatorn och inkludera detta i operativsystemet. Detta har i skrivande stund inte implementerats.

3.2.4 Bootmenyn

Bootmenyns funktion är enkel. Detta är programmet som startas av initieringsfunktionen efter reset och är operativsystemets gränssnitt mot omvärlden. Utformningen är enkel i form av en meny där tillgängliga applikationer i flashrom:et listas. Användaren kan välja ett program och köra det. Vid programstart kopieras binärkoden från flashrom till ramminnet, varefter huvudfunktionen anropas och applikationen tar över maskinen. När applikationen avslutas sker återhopp till bootmenyn som återställer menyn och proceduren kan återupprepas.

3.2.5 Program för omflashning

Detta är smårutiner skrivna i assembler som sköter utsuddning och omskrivning av flashminnet, tillsammans med ett enkelt applikationsprogam för att demonstrera erease-funktionen. Flashning sker automatiskt när det via terminalprogrammet på PC-sidan skickas över ny binärdata till filsystemet.

3.2.6 Program för seriekommunikation

Huvuddelen av kommunikationsarbetet sker automatiskt i bakgrunden, via lågnivå-interrupthanteraren för UART:en. All informationsöverföring sker med paket, som definieras av en typ-flagga, ett datalängd-fält, datan själv och eventuellt fyra bytes CRC-summa på slutet. CRC-rutinerna är inte implementerade än men sådana ingår i zlib-biblioteket nämt ovan.

Ett fåtal av de enklaste paktettyperna kan behandlas direkt i lågnivåhanteringen, exempelvis paketet som innehåller klockslag och datum (för synkronisering av realtidsklockan mot PC). De övriga paketen hanteras på lågnivån så flexibelt som möjligt. Inkommande paket tas emot fullständigt (och skulle enligt ambitionen CRC-kontrolleras om så erfodrades) varefter typflaggan och datapekaren skickas vidare till eventuell applikationsspecifik högnivåhanterare. Med tanke på att typflaggan för de generella, applikationsdefinierade pakettyperna är på 32 bitar uppnås en avsevärd flexibilitet och expansionsmöjligheterna är stora.

Ett applikationsprogram som används för att ställa om serieporthastigheten finns implementerat, och pakettyper för att meddela andra sidan av en fungerande kommunikationslänk att linjehastigheten kommer ändras finns också definierade.

Tidigare nämnd överföring av binärdata som ska flashas i filsystemet sker med ett standardpaket. Här är det verkligen motiverat med CRC-kontroll då det är ytterst viktigt att körbar kod inte innehåller bitfel. En lyckad anpassning av zlib-biblioteket hade/kommer att lösa dessa problem, emedan nuvarande system funkar genom att CRC-fältet helt enkelt ignoreras men skickas med i överföringen.

4 Utvärdering

4.1 Hårdvara

Hårdvaran fungerar tillfredsställande och det enda som inte blev genomfört var stereoljud. I efterhand kan det tyckas vara ett onödigt krav. Monoljudet som finns är fullt tillräckligt för de korta ljudstycken som kan spelas upp.

4.2 Mjukvara

I specifikationen står det att både kalenderprogram och röstprogram skulle skrivas. Detta har det inte funnits tid till. Av dessa skulle kalenderprogrammet ha varit av störst nytta medan röstprogrammets funktion mest skulle ha bestått av att demonstrera inspelningsfunktionen hos hårdvaran. Då handdatorn från början byggdes med väldigt generella funktioner, finns det förstås ett otal tänkbara applikationer som kan skapas via mjukvara. Detta var ju också ett av syftena med konstruktionen.

4.3 Logik

Vid valet att använda Lattice-kretsar hade ingen av projektdeltagarna stött på denna krets tidigare. Dock var fördelarna med kretsen så stora att detta överbryggade nackdelar såsom inlärningstid och lite ringrostiga kunskaper om kretsen på institutionen. Dessutom var de nödvändiga för att implementera DMA.

Det tog dock oväntat mycket tid att få kretsarna att fungera som de skulle. Detta berodde till stor del på att utvecklingsverktygen för Lattice är långt ifrån fulländade. Dessutom är manualer och datablad ett antal rejäla dokument, som man inte hittar en lösning snabbt och lätt i.

En incident bör nämnas närmare. Vid konstruktionen av Lattice 2 inträffade oförklarliga fel när koden skulle kompileras. Kompilatorn trodde vid något tillfälle att en utgång kopplades tillbaka till en ingång inuti Lattice-kretsen, vilket är otillåtet om inte kompilatorn lyckas sätta in buffertar emellan. Det lyckades givetvis inte. Efter lång felsökning konstaterades att ingen sådan återkoppling gjordes men inga försök lyckades åtgärda kompileringsfelet. Inte ens kompilering med en nyare version av utvecklingssystemet gav någon förbättring. Den slutliga lösningen fick till bli att göra om hela kretsen på ett annorlunda sätt. Alla tester och genomgångar av kopplingarna gjorde klart att den förklaringen måste vara en bugg i programmet. Sådana fel tar lång tid att konstatera och därför var konstruktionen av Lattice 2 knappast tidseffektiv på något sätt.

Ytterligare irritationsmoment var den dåliga programmeringssockeln i PROM-programmeringsrummet. Efter ett tag var den utsliten och kunde omöjligt hålla de stora Lattice-kretsarna på plats. Istället fick vi sitta och passa in kretsarna för hand och sedan hålla dem där tills programmeringen var klar.

4.4 Utökningsmöjligheter

Från början var det tänkt att även kommunikationen till PC skulle ingå i DMA-delen av konstruktionen. För att underlätta konstruktionen sattes UART:en istället som en av kretsarna som adresseras direkt av processorn. Detta underlättade konstruktionen väsentligt, då inga nya tillstånd behövdes i DMA:ns tillståndsmaskin. Om detta skulle genomföras så skulle kommunikationen kunna snabbas upp lite m h a det extra minnet i DMA-delen. Detta skulle avlasta processorn, som därmed skulle kunna utnyttjas effektivare.

Ett experiment med USB-kommunikation var också planerat från början men tidsbrist satte helt stopp för de planerna. Detta var en besvikelse, eftersom detta hade varit en nyttig erfarenhet från en lite modernare tillämpning och det kunde också ha gett institutionen insikt i USB. Det hade varit bra då ämnet säkerligen kommer att dyka upp igen i något framtida projekt hos institutionen.

En ytterligare kommunikationsmöjlighet, med goda nyttomöjligheter, vore en eller två generella 8-bitars digitala I/O-portar i direkt anslutning till databussen. Detta hade bara stulit en eller två adressplatser från minnesadresseringen och fördelarna med porten hade utan vidare vägt upp för den lilla nackdelen. Man hade t ex kunnat koppla in ytterligare periferienheter, liknande DMA-delen, till handdatorn. Detta hade kunnat utnyttjas till utökning av lagringsutrymme mm. Andra tillämpningar vore inkoppling av externt tangentbord, mus och joystick. Möjligheterna hade varit många.

5 Referenslista

ABEL-HDL Reference Manual (Version 8.0), Lattice Semiconductor Corporation

Design Verification Tools User Manual (Version 8.0), Lattice Semiconductor Corporation

Institutionens för Informationsteknologi hemsida (http://www.it.lth.se/it/courses/Digp/index.html) (2001-01-01 – 2001-05-16)

InterTools Assembly Language Manual (68000 Family), Institutionen för Informationsteknologi

InterTools Toolkit User's Manual, Institutionen för Informationsteknologi

IspDesignExpert User Manual (Version 8.0), Lattice Semiconductor Corporation

it-68 utvecklingssytem för MC68008 (version 4.0), Institutionen för Informationsteknologi

Parr, E. A. (1986), *How to use OP amps*, Bernard Babani (publishing) LTD

Ritchie, Dennis M. – Kernigan, Brian W. (1988), *The C programming language (second edition)*, Prentice Hall

Zlib hemsida (http://www.info-zip.org/pub/infozip/zlib/)

6 Appendix

1 Kravspecifikation av handdatorn

2 Schema

3 Pinkonfiguration för Lattice 1 och Lattice 2

4 Utskrifter av Latticekoden

5 Tillståndsschema för DMA

6 Uppdelning av minnesarean

7 Exempel på programkod

8 Komponentförteckning