Konstruktion av en enkel MP3-spelare
Digitala Projekt
Institutionen för Informationsteknologi

Nils Maltesson, d99nm@efd.lth.se
Mikael Ohlson, d99mo@efd.lth.se

2003-05-20

Abstract

This report describes the work and conditions when constructing a simple MP3 music file player prototype. The aim of the project described herein was to try to construct an MP3 player that could perform simple tasks such as play, stop, go to previous song and jump to next song. The player consists of hardware such as a 68008 microprocessor, logic circuits and memory, controlling a VS1001k MP3 decoder circuit. Programs are written in C code and Motorola 68000 assembler. A thorough description of how the construction was made and a following discussion of the device and our conclusions are the main parts of the report.

Innehåll

1 Inledning

Denna rapport behandlar tillvägagångssättet och det teoretiska arbetet kring konstruktionen av en MP3-spelarprototyp i kursen "Digitala Projekt". Det som främst beskrivs är själva genomförandet av konstruktionen. Därtill diskuterar vi de resultat som uppnåddes och de slutsatser vi kunnat dra av projektet.

1.1 Bakgrund

En MP3-spelare är en kretskonstruktion som kan spela upp audiofiler i filformatet MP3. Detta filformat innehåller informationen om ljudet och är komprimerad för att spara utrymme. Detta gör att en normallång MP3-låt upptar endast 3-5 MB, mot en okomprimerad WAV-fil som tar runt 40-50 MB. På grund av kompressionen går det dock inte att spela upp ljudet hur som helst, utan informationen i filen måste avkodas och packas upp för att låta korrekt. Det är främst denna aspekt som gör MP3-spelaren intressant och som skiljer den från andra typer av musikspelare. I filen ingår också oftast information om låtnamnet, artisten, styckets längd etc, vilket kan läsas ut och visas på en display. En annan fördel med MP3-filformatet är att filen kan kapas utan att det blir problem när den spelas upp. Det medför t ex att man kan spela upp delar av låtar, något som vi kommer att utnyttja i denna spelarprototyp då vår lagringskapacitet är ganska liten.

Anledningen till att vi ville konstruera en MP3-spelare är många, men främst ville vi göra något som skulle vara intressant och utmanande ur ett konstruktionstekniskt perspektiv. Dessutom är en MP3-spelare något som vi skulle kunna ha glädje av även efter kursens genomförande. Det var också en intressant — och svår — utmaning att anpassa kommunikationen till den färdiga MP3-avkodarkretsen på ett effektivt sätt. Alternativet skulle vara att själva implementera avkodningen av informationen i någon typ av logik, men vi insåg snabbt att det inte skulle gå med den begränsade tid vi hade till förfogande. Dessutom skulle det också troligtvis ha varit svårt att uppnå en tillfredsställande avkodningshastighet.

1.2 Kravspecifikation

För att till en början strukturera upp vad vi ville att vår färdiga produkt skulle klara av utformade vi en kravspecifikation som, enkelt uttryckt, anger vilka krav vi vill kunna ställa på den färdiga spelarprototypen. Dessa krav lyckades vi uppfylla till en del, men inte fullständigt. Utförligare beskrivning av vad spelaren i nuläget klarar av kommer senare. Kraven är uppdelade i sådant vi ville att spelaren skulle kunna klara, och sådant som inte är lika nödvändigt och därför tänkt att läggas till i mån av tid. På grund av tidsbristen vi haft under projektet har vi i nuläget dock inte kunnat realisera någon av utbyggnadsmöjligheterna.

1.2.1 Krav

Kraven vi ställt på spelaren är följande:

1. Spelaren skall vara fristående från PC

2. Spelaren skall kunna hantera multimediakort (MMC), dvs någon typ av löstagbart minnesmedia
3. MMC-kortet skall kunna laddas med MP3-filer från en PC med någon typ av fristående program
4. Spelaren skall ha enkla funktioner som "play", "stop", "nästa låt" och "föregående låt"
5. Inmatning av ovanstående val skall göras från en enkel knappsats
6. Låtarna skall kunna spelas upp i minst 64 kbit/s
7. Hörlurar skall kunna anslutas till spelaren
8. Spelaren skall klara av att spela upp iallafall kortare musikklipp, ej nödvändigtvis hela låtar

1.2.2 Utbyggnadsmöjligheter

Om tillfälle infinner sig skulle det vara intressant att implementera följande tillägg till ovanstående kravspecifikation:

1. Spelaren skall vara portabel (vilket i praktiken innebär batteridrift)
2. Möjlighet att ändra volym, diskant och bas skall finnas
3. Anslutning av display för visning av information som hämtas från MP3-filen, t ex låtnummer, titel och speltid
4. Sökning inom låt (snabbspolning)

2 Hårdvaran

I denna del beskrivs de olika delarna av vår MP3-spelare och hur det är tänkt att de ska samverka.

2.1 Allmänt om val av komponenter

Kärnan i MP3-spelaren är den krets som avkodar den digitala informationen och omvandlar den till en ljudsignal som kan förstärkas och skickas ut i t ex ett par hörlurar. För att mata avkodningskretsen med information och för att styra själva uppspelningen av musik behövs en mängd andra komponenter såsom en processor som styr hela förloppet, minnen i vilka processorns program ligger lagrat samt förstås någon typ av lagringsmedia för själva MP3-filerna. För att knyta samman dessa olika delar behövs slutligen en del styrlogik som ser till att processorns instruktioner tolkas på rätt sätt och når ut till de olika kretsarna.

När vi började designa hårdvaran visste vi inte till en början vilken typ av MP3-avkodningskrets som fanns tillgänglig och detta ledde till att vi ville vara så flexibla som möjligt i vår konstruktion av den övriga hårdvaran. Därför beslöt vi oss för att använda en 68008-processor från Motorola och att koppla externa minnesmoduler till den eftersom den inte innehåller något eget minne

överhuvud taget. Vi fick också rådet att låta data från det minnesmedia vi läser av gå via processorn. Detta kom att leda till en del begränsningar i den övriga hårdvaran. Eftersom processorn endast kan adressera totalt 1 MB då den har ett begränsat antal adresspinnar leder det till att vi maximalt kan adressera knappt 1MB MP3-data. Detta gjorde att ett minnesmedia av MMC-typ (Multi Media Card) var onödigt stort och vi använde oss istället av ett (mycket) mindre 128 kB EEPROM. En annan bidragande faktor till att vi använde EEPROM var att MMC-media ej med enkelhet kunde anslutas på prototypbrädet eller till resten av elektroniken. I praktiken innebär detta att vi bara kan spela upp mycket korta musiksnuttar. Hursomhelst är det fullt tillräckligt för att kontrollera om vår spelarprototyp fungerar som vi tänkt. Storleken på programminnet och internminnet som ska lagra stacken och interna variabler åt processorn satte vi till 32 kB eftersom vi ansåg att det var lagom. Processor, minnen och logik kopplas samman med två bussar, dels en adressbuss i vilken information om vilka adresser processorn läser från eller skriver till överförs och dels en databuss som innehåller den transporterade datainformationen.

För att driva kretsarna i konstruktionen var vi tvungna att använda oss av tre klockor från separata klockkristaller. De går på en frekvens av 10 MHz, 6 MHz och 24,576 MHz. Den första klockan på 10 MHz används för att driva processorn och en del kringlogik som t ex adresslogiken. Denna hastighet är den högsta som processorn klarar av och vi vill driva den så snabbt som möjligt för att kunna spela upp musik med så hög bitrate som möjligt. Ju högre bitrate en låt har, desto högre hastighet krävs på systemet för att man inte ska få hack i ljudströmmen. Klockan på 6 MHz används till klockingångarna på MP3-avkodaren då kommandon och MP3-data skickas till den. Dessutom drivs också en del av kringlogiken kopplad till MP3-kretsen med denna klocka. 24 MHz klockan används internt i själva MP3-kretsen för att den ska kunna avkoda musiken i korrekt takt. Denna klocka avgör hur hög bitrate man maximalt kan ha på den bitström som skickas till avkodaren. I fallet med 24 MHz ger det en maximal avkodningshastighet på 192 kbit/s, oftast kan strömmar upp till 256 kbit/s spelas upp utan att det uppstår problem men det är inte garanterat av specifikationen till avkodningskretsen.

2.2 MP3-avkodaren

Den MP3-avkodare vi till slut fick tag på heter VS1001k och kommer från det finska företaget VLSI Solutions Oy. Det är en riktigt praktisk liten krets som klarar av allt vi behöver och en mängd annat. Dels avkodar den förstås MP3-informationen som skickas till den men innehåller också en integrerad D/A-omvandlare och förstärkare vilket innebär att hörbart ljud direkt kan plockas ut från kretsen. Man behöver alltså ingen logik för att omvandla den digitala informationen och förstärka den. Ett kommandointerface finns så att man kan skicka instruktioner om att t ex höja eller sänka volym, bas och diskant men också läsa ut låtnamn och annan information om det ingår i MP3-filhuvudet. Kretsen innehåller dessutom en liten mikroprocessor och lite programminne, vilket gör att man kan lägga in program i MP3-kretsen som kan göra utföra saker. Detta sistnämnda är dock inget som vi utnyttjar i vår något mer begränsade användning.

Emellertid har kretsen en del egenheter som gjorde det rejält besvärligt att ansluta den till vårt system. Till att börja med drivs den med 3.3V medan resten av kretsen kräver 5V. Detta innebär t ex att separat strömförsörjning måste göras och dessutom att alla signaler som kommer från omgivande logik måste sänkas från 5V till 3.3V för att kretsen skall fungera. Problemet med strömförsörjningen löste vi till en början med spänningsdelning över ett variabelt motstånd, men senare visade det sig att det inte gick att bibehålla en stabil spänningsnivå på detta sätt. Detta gjorde att vi var tvungna att använda oss av en spänningsregleringskrets istället. Sänkningen av signalerna till MP3-kretsen löste vi på ett smidigt sätt genom att låta dem gå genom en öppen D-latch som drevs med 3.3V. I praktiken innebar det att vi erhöll en utsignal på 3.3V som var mycket lite fördröjd. Detta fungerade eftersom latchen tålde 5V på ingångarna, men gav 3.3V ut tack vare den låga drivspänningen. Då MP3-kretsen bearbetar en ström av data skickas en signal ut från avkodaren som anger när mer data behöver skickas för att det ska gå att hålla en jämn uppspelningstakt. Det är den enda utsignalen från avkodaren som vi tar hänsyn till och problemet med denna är att den ligger på nivån 3.3V. Emellertid hade vi tur eftersom denna spänning räcker för att 5V-logiken ska reagera på den. Alltså krävdes ingen extra kringlogik för att höja nivån på denna signal. Ett betydligt omständigare och utmanande problem var det protokoll som skulle följas då data och kommandon skickas till kretsen. Dels krävdes att informationen skickades seriellt till avkodaren emedan den data vi hade på databussen från processorn var parallell. Alltså blev vi tvungna att göra en omvandling av signalen från parallell till seriell överföring. För detta krävdes ett åttabitars skiftregister och en räknare som höll reda på vilken bit i strömmen som skickades till avkodaren.

2.2.1 Signaler till MP3-avkodaren

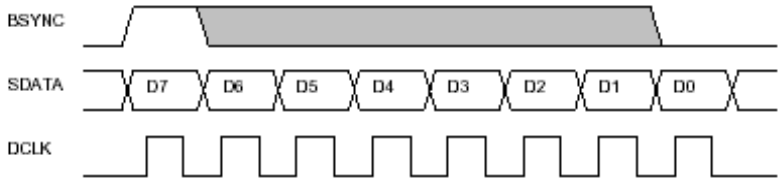
Det finns två lägen på MP3-kretsen som är av intresse för oss. Dels Serial Data Interface-läget, då vi skickar MP3-data till kretsen, och dels Serial Command Interface-läget, då vi skickar kommandon till kretsen.

MP3-data skickas till avkodaren då den begär det genom en signal ut från densamma, DREQ[1]. Då detta uppfattas av den omkringliggande logiken skall en byte MP3-data i taget skickas till kretsen. Detta görs genom att en styrsignal, XCS, hålls hög samtidigt som signalen DCLK klockas åtta gånger. Vid varje klockning av DCLK ska en bit data skiftas in till SDATA från skiftregistret. I början av varje byte som skickas ska dessutom en signal BSYNC hållas hög för att markera att en ny byte börjar. Se tidsdiagrammet i figur 1.

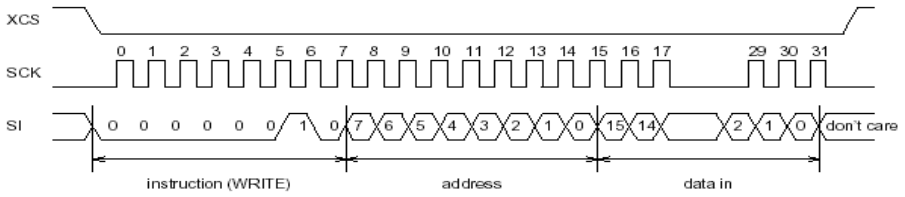
Alla kommandon till kretsen består av fyra bytes. Då ett kommando skickas till MP3-kretsen med signalen SI skall styrsignalen XCS hållas låg under alla fyra byte och får inte gå upp emellan dessa. På liknande sätt som tidigare används åtta pulser från en klocksignal, SCLK, för att skriva in varje byte i det 4 byte långa kommandot. Ingen synkning behövs dock. Se tidsdiagrammet i figur 2.

[1] Värt att notera är att DREQ endast signalerar att det finns utrymme för 32 byte mer data i kretsens interna minne.

Figur 1: Dataskrivcykel av en byte MP3-data



Figur 2: Dataskrivcykel av 4 byte kommando



2.3 Kommandoinmatning till processorn

All kommandoinmatning från användaren är tänkt att ske med 5 vanliga tryckknappar. Dessa är "Play", "Stop", "Next" och "Previous" samt "Reset". Den sistnämnda signalen är enkelt kopplat till processorn, MP3-kretsen och den programmerbara logiken på dessas resetingångar, vilket leder till att systemet startas om vid ett tryck på denna knapp. De fyra första använder sig däremot av avbrott vilket innebär att processorn avbryter det den håller på med och hoppar till speciella avbrottsrutiner, en för varje knapp. Hur dessa rutiner fungerar beskrivs senare under avsnitt 3 på sidan 8. Anledningen till att vi använder oss av avbrott är att en knapptryckning kan komma när som helst, men mycket sällan i jämförelse med allt annat arbete som processorn ska utföra. Därför är det ingen bra idé att använda sig av polling eller liknande. Dessutom stöds avbrott till fullo av processorn. Processorn har olika typer av avbrott (autovektor och normala vektoriserade) och även olika nivåer (1-7) som anger hur kritiska avbrotten är. Alla avbrott som har med knappar att göra är av typen vektoriserade avbrott, och ligger på avbrottsnivå 5. Vektoriserade avbrott fungerar så att processorextern logik anger den vektor som processorn skall utföra tillhörande avbrottshanterare för. Anledningen till att vi använder oss av vektoriserade avbrott är att vi enkelt ska kunna skilja på vilket avbrott som uppstått eftersom alla fyra knappavbrotten ligger på samma avbrottsnivå och annars inte hade gått att skilja åt. Detta görs genom att varje knapp får en egen avbrottsvektor.

Det finns ytterligare ett avbrottskommando utifrån som kan komma in till processorn och det är då MP3-kretsen vill ha mer data att avkoda. Även detta är ett vektoriserat avbrott och ligger på nivå 2, vilket innebär att det är lägre prioriterat än knapptryckningarna. Om en knapptryckning skulle komma sam-

tidigt som detta avbrott prioriteras alltså knappen. Vi har löst det på detta sätt eftersom det inte gör något om dataströmmen till MP3-kretsen avbryts då man trycker på en knapp eftersom alla knapptryckningar på något sätt indikerar att man vill starta eller pausa låten (Play), stanna den (Stop), gå tillbaka (Previous) eller hoppa till nästa (Next).

En sista styrsignal som leds in till processorn är DTACK-signalen, vilket står för "Data Acknowledge". Denna signal måste alltid skickas tillbaka till processorn efter att data har lagts ut på databussen, så att processorn kan fortsätta sin exekvering. Om ingen DTACK-signal skickas tillbaka kommer processorn att stanna upp. Signalen genereras i den programmerbara logiken vilken beskrivs nedan.

2.4 Programmerbar logik

Vi har 5 stycken, programmerbara logikkretsar i vår konstruktion. Främst beror mängden på det protokoll som avkodaren kräver att man följer vid skickande av MP3-data och kommandon till kretsen. Logikkretsarna är alla relativt avgränsade och sköter olika uppgifter, som beskrivs nedan.

2.4.1 Adresslogik

Denna krets sitter i direkt kontakt med processorn och har till uppgift att ta reda på vilken krets som processorn försöker kommunicera med. Då processorn läser från eller skriver till olika adresser läggs adressinformationen ut på adressbussen och analyseras av adresslogiken. Denna ser till att aktivera rätt krets - t ex programminne eller MP3-kretsen, beroende på vilken adress som processorn vill kommunicera vid. Adresslogiken tar också hand om svaret från dessa kretsar och ser till att skicka tillbaka DTACK-signalen till processorn då data utlagd på databussen har uppfattats och mottagits eller perifera enheter har lagt data på databussen och väntar på att processorn skall läsa den.

2.4.2 Avbrottslogik

Avbrottslogiken behandlar avbrotten från knapparna och MP3-kretsen och ser till att rätt avbrott genereras och skickas till processorn, dels vad gäller avbrottsnivå och dels typ. Då någon av knapparna ska generera ett avbrott ser avbrottslogiken till att hänvisning till rätt avbrottsvektor läggs ut på databussen så att det kan läsas in av processorn. På detta sätt får varje knapp en egen avbrottsadress som gör att processorn kan utföra olika saker beroende på vilken knapp som trycks ned.

2.4.3 MP3-styrlogik

Styrlogiken till MP3-kretsen var en av de största utmaningarna i konstruktionen och krävde en hel del tankeverksamhet innan den blev bra. Vi har konstruerat den som en tillståndsmaskin med 4 tillstånd som har 9 insignaler och 8 utsignaler. Mängden styrsignaler till MP3-kretsen och dess olika data- och kommandolägen gjorde att denna logik blev mycket invecklad. Maskinens fyra tillstånd klarar av att skicka data eller kommandon oberoende av varandra och kan kommunicera med de andra programmerbara logikenheterna såväl som skiftregistret

och räknaren som behövs för att skicka data till MP3-avkodaren. Eftersom det är så många signaler och övergångar mellan olika tillstånd hänvisas till bild 4 på sidan 16 för den som är intresserad.

2.4.4 Tillståndslogik

Tillståndslogiken[2] är egentligen en del av MP3-styrlogiken ovan som inte fick plats i den kretsen. Det är en tillståndsmaskin med 8 tillstånd som ser till att hålla styrsignalen XCS till MP3-kretsen i lågt läge under översändningen av 4 byte i ett kommando.

2.4.5 Misc-logik

Denna logik består av ett fåtal grindar. Den ser till att samla ihop ett antal signaler från processorn för att skapa en signal som kan indikera att processorn tagit emot ett avbrott. Denna signal, INTACK, kopplas sedan vidare till adresslogiken och avbrottslogiken för att användas i dess logik.

2.5 Komponenter

Komponentlistan, som till slut blev ganska diger, följer nedan:

- Processor: Motorola 68008
- Minnen: 32 kB SRAM, 32 kB EPROM, 128 kB E^{2}PROM
- Styrlogik: 4 st PALCE22V10H/4, 1 st GAL22V10D
- MP3-avkodare: VLSI Solutions Oy VS1001K
- Övrig logik: 2 st 74HC04N (NOT-grind krets), 74LS165 (8 bit skiftregister), 74LS192N (16-räknare), 74HC573A (8 ingångars D-latch), MAX604 (3,3V spänningsreglerare)
- Klockkristaller: 6 MHz, 10 MHz, 24,576 MHz
- Tryckknappar: 5 st
- Övriga komponenter: Diverse motstånd och kondensatorer

Hur uppkopplingen slutligen blev finns i kopplingsschemat i appendix.

3 Mjukvaran

Mjukvaran i vår konstruktion är dels den "logikkod" som finns i de programmerbara logikkretsarna och dels den programkod som processorn ska exekvera.

[2] På kretsen benämnd "state logic"

3.1 Processorprogram

Processorprogrammet är det som styr strömmen av MP3-data till avkodningen, sköter läsning och skrivning från respektive till minnena och tar hand om de avbrott som sker. Avbrotten är som tidigare nämnts dels knapptryckningar från användaren och dels förfrågan om mer data att avkoda från MP3-logiken. Programmet är uppbyggt i dels Motorola 68000-assembler (gäller anrop till avbrottsrutiner och initiering av processorn) samt i C (resten av programmet samt avbrottsrutiner). Programmet är mycket enkelt till sin uppbyggnad och huvuddelen, dvs den del som behandlar vad som händer vid de olika avbrotten, beskrivs nedan.

Huvudprogrammet ska först initiera pekare till de olika delarna av minnet som man kan skriva till och läsa från. Det innebär i praktiken EEPROM:et samt de adresser som används vid skrivning av kommando respektive MP3-data till MP3-avkodaren. Sedan läses de första byten från EEPROM:et för att få reda på var de olika låtarna börjar respektive slutar. Efter detta resettas och initieras MP3-avkodaren. Det sista som görs i initieringsfasen är att sätta aktuell låt till att vara den första i EEPROM:et och slutligen att aktivera avbrotten. Då avbrotten initierats kan processorn ta hand av de avbrott som genereras av antingen användaren eller avkodningskretsen. Därefter lägger sig programmet och väntar i en evig loop i mainprogrammet. All aktivitet som kan förekomma i processorn kommer att startas av de olika avbrotten.

Det som är tänkt ska hända vid de olika knapptryckningarna är följande. Då användaren trycker på "Play" ska aktuell låt börja spelas upp, och om spelning pågår ska uppspelningen pausa. Tanken är att spelaren ska fortsätta spela på samma ställe om man trycker på "Play" igen. Då användaren trycker på "Stop" ska uppspelningen sluta och nästa gång "Play" trycks ner ska uppspelningen av musiken börja om från början. Om spelaren står i pausläge ska uppspelningen börja om från början på samma sätt som i playläget. Trycker användaren på "Previous" eller "Next" ska uppspelningen hoppa till föregående respektive nästkommande låt och börja spela där om man befinner sig i playläge, annars vänta tills användaren trycker på "Play".

Det som händer vid de olika avbrotten och hur processorn behandlar det hela kommer nedan att beskrivas i pseudokod. För utförligare beskrivning hänvisas till koden i appendix.

```
if (dataRequestInterrupt){
   if (status == PLAYING){
       För över 32 byte MP3-data från EEPROM:et till MP3-avkodaren om
       tillräckligt med data finns tillgängligt. Annars för över så
       mycket som möjligt upp till 32 byte.
       if (slutet på sången nått){
          previousStatus = PLAYING
          status = CHANGE_SONG
          currentSong = currentSong+1
       }
    }
```

```
    else if (status == STOPPED || PAUSED) {
       Gör ingenting alls...
    }
    else if (status == CHANGE_SONG) {
       if (zeroesCounter < 2048){
          Skriv 32 byte nollor som MP3-data
       }
       else {
          Gör en Software Reset av MP3-avkodaren
          Vänta i minst 250 mikrosekunder
          Skicka en 0:a som MP3-data
          status = previousStatus
          Nollställ zeroesCounter
       }
    }

if (playButtonInterrupt) {
   if (status = PLAYING) {
      status = PAUSED
   }
   else if (status = STOPPED || PAUSED) {
      status = PLAYING
   }
}

if (stopButtonInterrupt) {
   status = STOPPED
   Gå tillbaka till början av sången
   Gör en Software Reset av MP3-avkodaren
   Vänta i minst 250 mikrosekunder
   Skicka en 0:a som MP3-data
}

if (previousButtonInterrupt) {
   previousStatus = status
   status = CHANGE_SONG
   currentSong = currentSong-1
}

if (nextButtonInterrupt) {
   previousStatus = status
   status = CHANGE_SONG
   currentSong = currentSong+1
}
```

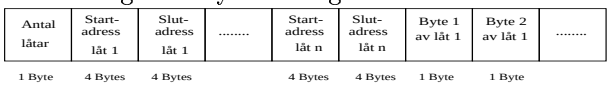
3.2 Logikkoden

Varje programmerbar krets har sin egen logikkod som "bränns in" i kretsen med hjälp av speciell skrivutrustning. Programmen skrivs i språket PALASM[3] och programkoden består bara av tilldelningarna av enkla logiska uttryck som kombinerar insignalerna på angivet sätt så att utsignalerna från kretsen får olika värden beroende på insignalerna. De logiska uttrycken som används är endast "not", "and" och "or". För en närmare beskrivning av denna programkod, se appendix.

3.3 MP3-data i EEPROM:et

MP3-data som lagras i EEPROM-minnet ligger inlagd enligt ett speciellt mönster för att det ska bli lätt för processorprogrammet att läsa den. Den allra första byten i EEPROM:et anger hur många MP3-låtar som ligger inlagda i minnet, n. De därpå $8n$ följande byten anger start- och slutadress i EEPROM-minnet för varje låt. En viktig begränsning som följer av detta är att vi maximalt kan lagra 256 låtar på varje EEPROM. Varje adressangivelse är 4 byte och adresserna anges i ordningen start för låt x, slut för låt x, start för låt x+1, slut för låt x+1 etc. Se bild 3:

Figur 3: Byteordning i EEPROM:et



4 Genomförande av konstruktion

Själva konstruktionen av spelarprototypen gjorde vi genom att först gå igenom den principiella sammankopplingen mellan processorn, minnen och MP3-avkodaren. Vi visste också att vi till en början ville hantera knapptryckningarna med avbrott. Efter att ha sett över datablad och bestämt oss för de komponenter som angavs i hårdvaruavsnittet började vi designa själva kopplingsschemat i programmet PowerLogic. Då det var såpass färdigt att vi kunde börja testa det var det dags att koppla upp hårdvaran. Kopplingsschemat höll vi uppdaterat under hela konstruktionsarbetet och detta har också hjälpt till att fungera som ett slags versionshanteringssystem då ett antal kopplingsscheman och förändringar av konstruktionen gjorts under projektets gång. Detta har även medfört att vi lätt kunnat gå tillbaka då vi insett att ett nytt konstruktionsspår varit en återvändsgränd.

4.1 Utvecklingssystem it-68

Istället för en riktig processor har vi under arbetet använt oss av en "processoremulator", utvecklingssystem it-68. Detta är en testhårdvara ansluten till hårdvaran som emulerar processorn och de minnesplatser som processorn kan

[3]MP3-logiken ligger i en GAL-krets vars kod är skriven i Abel, eftersom uppgiften som kretsen skall utföra är av mycket högre komplexitetsgrad än de övriga logikkretsarnas

adressera. Emulatorn är kopplad till en dator och kan kommunicera med denna. Genom att i datorn ange vilka minnesadresser som ska adresseras i emulatorn och vilka som ska adresseras i riktig hårdvara på det kopplingskort man ansluter emulatorn till, kan man på ett effektivt sätt testa kommunikationen mellan processor och minne. Dessutom kan man ladda ner program till processoremulatorn eller exekvera enstaka kommandon från datorn direkt. Detta har varit ett mycket värdefullt hjälpmedel vid felsökning av hårdvaran genom att t ex se om alla signaler tar sig genom styrlogiken på rätt sätt.

Ett annat viktigt hjälpmedel vi använt oss av har varit det digitala oscilloskopet med 4 ingångar. Detta har gjort det möjligt för oss att samtidigt studera flera signaler och på så sätt kunnat analysera bl a signalernas timing, frekvens, spänningsnivå och mycket annat. Möjligheten att trigga på olika signalers flanker har varit till stor nytta.

4.2 Uppkoppling av hårdvara

Hårdvaran kopplade vi ihop på ett virkort, och det första vi satte samman var processoremulatorn med de minnesmoduler vi valt, RAM:et, EPROM:et och EEPROM:et samt adresslogiken. Då styrsignaler och data- samt adressbussarna var på plats kunde vi testa kommunikationen och kom fram till att den fungerade fint, redan efter första försöket. Det som kom härnäst var att ansluta resten av styrlogiken för MP3-avkodaren, bitskiftregister, räknare, latch med mera. Det enda vi inte anslöt i detta andra steg var avbrottslogiken, knappar och själva MP3-kretsen. Det var ganska svårt att i detta läge säga om signalerna kom fram som de skulle, men de verkade iallafall var för sig vara mer eller mindre korrekta. En hel del omprogrammering av styrlogiken fick göras, främst av MP3-styrlogiken som vid detta inledande skede inte var gjord som en tillståndsmaskin.

Då vi trodde oss ha korrekt omgivande logik kopplade vi även in MP3-kretsen och detta gav upphov till en del problem. Till att börja med märkte vi att den spänningsdelning vi tidigare haft för att förse en del av kretsen med 3,3V inte längre räckte till då även MP3-avkodaren började dra ström. Det gick inte längre att ordna med spänningsdelning utan vi blev tvungna att ansluta en spänningsreglerare som såg till att nivån blev korrekt. Detta tog ett tag att upptäcka eftersom avkodaren drog olika mycket ström i olika lägen. Efter anslutningen av spänningsdelaren uppstod inte längre problemen. Ett mer allvarligt problem vi upptäckte efter anslutning av MP3-kretsen var att timingen mellan de olika styrsignalerna inte var helt bra. Sändning av data kunde börja mitt i en klockcykel. Dessutom krävde avkodningskretsen att styrsignalerna låg höga längre än vad vi kunde åstadkomma med den typ av MP3-styrlogik vi använde. Därför blev vi så småningom tvungna att skriva om denna logik helt och gick då över till att göra den till en tillståndsmaskin. Detta gav ett mycket bättre resultat. Efter en del finjustering av indatasignalen SDATA respektive SI från skiftregistret genom fördröjning genom ett antal NOT-grindar fungerade timingen signalerna emellan fint. I detta läge kunde vi skicka enstaka testsignaler från emulatorn till MP3-kretsen och få den att generera hörbara sinussignaler och variera volymen bland annat.

Ännu ett problem uppstod tyvärr då vi skulle koppla in den sista delen av vår krets, nämligen avbrotten. Eftersom användaren styr val av funktion genom knapptryckning kunde vi inte testa MP3-avkodaren ordentligt innan avbrotten kopplats in. Framförallt kunde vi inte skicka MP3-data till avkodaren, eftersom den signalerar när den vill ha mer data med hjälp av ett avbrott. Efter inkoppling av knappar och avbrottslogik märkte vi att en del signaler från processorn (FC0, FC1 och FC2) som endast skulle vara aktiva vid avbrotthantering fluktuerade mycket i icke aktivt tillstånd. [4] Det innebar att styrlogiken uppfattade det som att de tre signalerna var aktiva samtidigt och att ett avbrott skulle bekräftas. Bekräftelse av avbrott innebar i vårt fall, eftersom vi använde vektoriserade avbrott, att vi skulle aktivera signalen DTACK in till processorn. Detta fick som följd att oönskade DTACK-signaler kunde komma vid t ex minnesåtkomst, varvid detta orsakade databussfel.

5 Resultat och utvärderande diskussion

I denna del diskuteras de problem vi har med konstruktionen, eventuella lösningar och även hur vi skulle kunna ha undvikit de problem vi stött på med den kunskap vi nu har.

5.1 Nuvarande problem med konstruktionen

I skrivande stund har vi inte en fullt fungerande prototyp på grund av de problem vi haft med fluktuerande funktionskodssignaler från processorn. Eftersom det inte finns något annat sätt att avgöra om ett avbrott faktiskt tas omhand av processorn än att se på funktionskoderna är detta ett problem som gjort att vi inte kunnat testa om MP3-avkodaren faktiskt kan avkoda den data vi vill sända till den. För att bli av med de störande DTACK-signalerna som genereras måste avbrotten stängas av och utan dem kan inte MP3-kretsen begära mer data. Detta är alltså ett problem som måste kommas runt för att MP3-avkodningen ska fungera. Ett sätt att lösa problemet skulle vara att undersöka hur länge dessa "falska avbrott" pågår. Det rör sig enligt våra undersökningar endast om tillfälliga signalspikar som är mycket kortare än ett korrekt avbrott. Det borde gå att i logik konstruera en funktion som samplar signalen vid två påföljande klockcykler och från denna information ge en korrekt signal som filtrerar bort de felaktiga, korta avbrottssignalerna.

Eftersom inte avbrotten fungerar vet vi inte heller med säkerhet om vår programkod till processorn fungerar, men om den inte skulle visa sig vara korrekt är det en jämförelsevis mycket enkel sak att ändra. Tack vare styrprogrammets rättframma upplägg är det lätt att felsöka.

Ytterligare en sak som inte fungerar helt tillfredsställande är vår "Reset"-signal som inte verkar resetta processorn utan endast sätta den i HALT-läge. Detta har vi emellertid inte ansett vara ett allvarligare problem och på grund av tidsbrist har vi inte närmare undersökt det, men troligtvis är det ganska enkelt att åtgärda.

[4] Avvikelser på upp till 2.7V som mest.

5.2 Det som i skrivande stund fungerar

Det vi i nuläget vet fungerar som avsett är all kringlogik, exklusive avbrottslogiken. Vi vet att processorn kan adressera allt minne, vi kan läsa från och skriva till RAM:et samt läsa från EEPROM:et. Vi kan också visa att processorn har kontakt med MP3-kretsen och att vi kan exekvera inbyggda tester på denna. Att sända data och kommandon till MP3-avkodaren fungerar även detta fint och vi kan bl a utföra sinusvågstest och reglera volymen med mera.

Om man ser till kravspecifikationen har vi således tyvärr inte kunnat uppfylla alla kraven i specifikationen. Vi får anse oss ha lyckats med punkterna 2-7. Mer tveksamt är det dock om man kan säga att spelaren är fristående från PC eftersom vi fortfarande använder oss av emuleringsverktyget och inte en "riktig" processor. Tyvärr har vi inte heller kunnat bekräfta att spelaren faktiskt kan spela upp några låtar. Däremot kan vi (iallafall via emulatorverktyget om än inte via riktiga knappar i hårdvaran) variera volym, bas och diskant så vi har faktiskt även lyckats uppfylla punkt 2 i tilläggen till kraven.

Själva anser vi dock att huvuddelen av vår konstruktion fungerar även om vi inte lyckats få något faktiskt uppspelat MP3-ljud ur den utan endast en testsinussignal. Hade det inte varit för processorns varierande funktionskodssignaler hade vi haft en korrekt avbrottshantering och därigenom också kunnat se om uppspelning fungerat.

5.3 Tänk om

Såhär i slutet av projektet är det också intressant att reflektera över hur vi skulle ha gjort konstruktionen med den kunskap vi nu besitter om all hårdvara. Vi skulle antagligen ha ändrat på en hel del detaljer. Till att börja med kan man konstatera att det hade varit klokt av oss att börja göra MP3-styrlogiken (och även en del annan logik) i tillståndsmaskiner direkt. Det hade sparat oss en hel del problem och framförallt tid. Att försöka få rätt på timingproblemen vi fick innan vi gick över till en tillståndsmaskin visade sig bli ett övermäktigt problem.

Dessutom kan man fundera på om det egentligen är så lämpligt att leda MP3-datasignalen genom processorn. Egentligen inte, eftersom processorn endast kan adressera 1MB vilket innebär att man i praktiken inte kan ha större minne, utan en del problem. Bättre hade varit om processorn endast styrt den kringliggande logiken och om EEPROM:et (eller någon annan typ av minnesmedia, förslagsvis ett Smart Media-kort av större format) varit förbundet med MP3-kretsen på en separat databuss. Ett annat övervägande vi borde lagt ner mer tid på är huruvida 68008-processorn är optimal för uppgiften. Problemet med denna processor är att den inte själv har någon typ av direkt kommunikationsmöjligheter med "omvärlden". Detta har bevisligen i vårt fall lett till en omfattande kringlogik som man annars kunnat undvika till stor del om man t ex använt sig av en HC11. Den komplexa styrlogiken till MP3-kretsen hade antagligen till stor del kunnat undvikas.

Vi skulle antagligen ha haft det lättare om vi använt oss av autovektoravbrott istället för vektoriserade avbrott eftersom vi då inte hade behövt generera en DTACK vid avbrott. Således hade vi troligtvis inte fått de bussproblem som uppstår i nuläget.

5.4 Slutligen

Det här projektet har visat att det borde vara fullt möjligt att med ganska begränsade resurser, både vad gäller material och tid, konstruera en duglig MP3-spelare som klarar av samtliga av de krav och utbyggnadsmöjligheter vi ställt upp i vår kravspecifikation. Vi kan konstatera att MP3-avkodaren som vi använt är en bra grundstomme att bygga upp en MP3-spelare kring. Den är kraftfull och flexibel, men har ändå enligt vad vi erfar, en del nackdelar. Främst är att den inte har någon möjlighet att ta emot parallell data vilket dels skulle snabba upp utvecklingsarbetet avsevärt och dels göra det mycket lättare att överföra data till kretsen. Som en bonus skulle hela konstruktionen kunna bli snabbare.

Som en följd av att MP3-spelare är relativt enkla att konstruera ser vi att marknadsaktuella MP3-spelare idag har fler och fler finesser och stödjer flera olika filformat. Många moderna spelare är nu även USB-mass storage-enheter och klarar av att visa film och anslutas till TV och ljudanläggningar direkt.

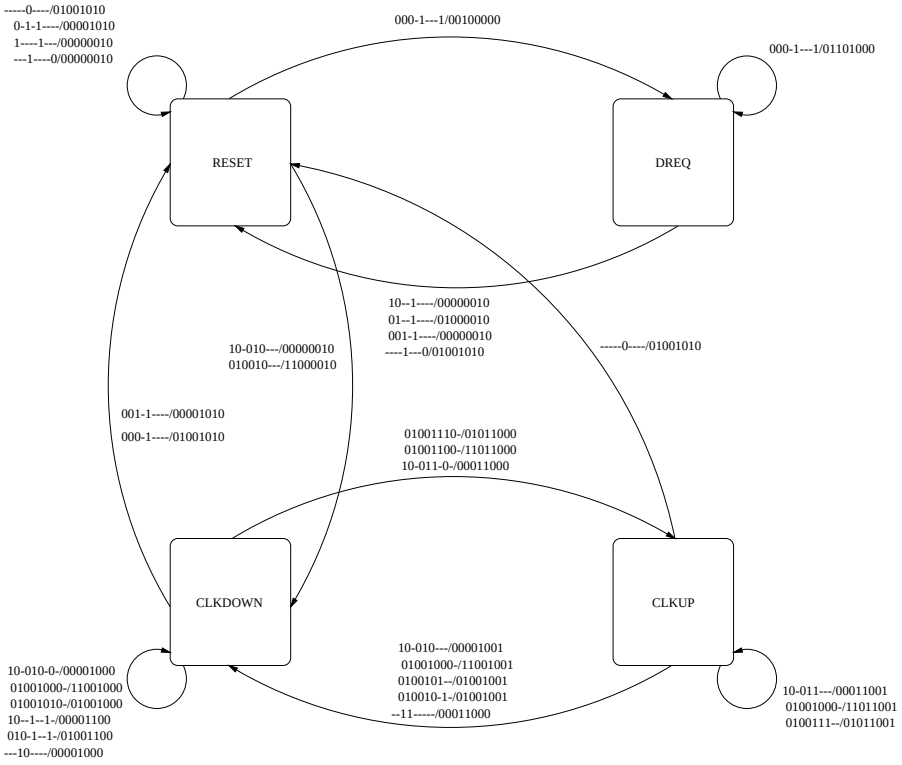
Det man väl får ta och se som den största vinsten med projektet är, fungerande konstruktion eller ej, att vi lärt oss väldigt mycket som vi inte kunde tidigare vad gäller konstruktion av hårdvara och hårdvarurelaterad mjukvara. Det gäller allt från planeringen till själva genomförandet och testning. Mycket av det som vi lärt oss i tidigare kurser i bl a digitalteknik och kretsteori har kommit till användning här, och det är först i detta projekt som vi till fullo kommit att förstå hur det hela hänger samman.

6 Referenslistor

- Föreläsningsanteckningar i Digitalteknik,
  http://www.it.lth.se/it/courses/digitalteknik/Pdfs/OH_All.pdf
- Dokument om "Utvecklingssystem it-68",
  http://www.it.lth.se/it/digproj/PDF_files/it68/it86.pdf
- Dokument om Pal assembler,
  http://www.it.lth.se/it/digproj/PDF_files/pldasm.pdf
- Dokument om Lattice Abel Design,
  http://www.it.lth.se/it/digproj/PDF_files/lattice/abel_design_manual.pdf
  och http://www.it.lth.se/it/digproj/PDF_files/lattice/abel_reference.pdf
- The C Programming Language, Second Edition
  Brian W. Kernighan & Dennis M. Ritchie
- Dokument om hur man skriver en teknisk rapport,
  http://www.it.lth.se/it/courses/Digp/PDF_files/råd.pdf

A Tillståndsdiagram

Figur 4: Tillståndsmaskin för MP3-logik



Insignal 1	CSSCI	Cable select SCI
Insignal 2	CSSDI	Cable select SDI
Insignal 3	HOLD	Håll XCS hög
Insignal 4	$\overline{\text{DS}}$	Data strobe
Insignal 5	$\overline{\text{RESET}}$	Systemreset
Insignal 6	CLK6	6MHz-klocka
Insignal 7	FOURC	Räknaren är mellan fyra och åtta
Insignal 8	EIGHTC	Räknaren är på åtta.
Insignal 9	DREQ	Data Request från MP3-chip
Utsignal 1	BSYNC	Byte sync.
Utsignal 2	$\overline{\text{XCS}}$	Data eller kommando
Utsignal 3	MP3INT	Begär en interrupt för att få mer data
Utsignal 4	CLK	Seriell klocka
Utsignal 5	$\overline{\text{LOAD}}$	Ladda shift-register
Utsignal 6	MP3DTACK	Bekräfta mottagande av data
Utsignal 7	CLEAR	Nollställ räknare
Utsignal 8	UP	Klocka upp räknaren ett steg

B Kretsscheman

C Källkod