Labyrintspelet
EDI021
Grupp 5

Kristian Sylwander d04ks@student.lth.se
Emil Wasberger d03ew@student.lth.se
Michael Winberg d04mwi@student.lth.se

16 maj 2008

1 Inledning

Syftet med kursen EDI021 är att bygga ett datorsystem där både hårdvara och mjukvara ingår. En kravspecifikation togs fram för att under kursens gång realisera denna i en fungerande konstruktion.

Konstruktionen denna rapport behandlar bygger på processorn Motorola 68008 som har sitt ursprung i det glada 80-talet. Mycket sladdar och hårdvara är inblandat i uppbyggnaden av konstruktionen. Detta trots att 68008 bara har en 8-bitars databuss.

Labyrintspelet går ut på att navigera genom en labyrint till en utgång. Då en nivå är avklarad genereras en ny svårare labyrint. Nivåerna genereras slumpmässigt[1] och varieras från gång till gång. Labyrinten visualiseras på en LCD och spelaren navigeras med fyra knappar.

2 Kravspecifikation

Nedan följer kravspecifikationen i sitt orginalutförande:

Vi skall tillverka ett spel bestående av en styck motorola-cpu, lcd-skärm, knappar/joystick samt allt som behövs för att koppla ihop allt plus annat såsom minne etc.

Spelet är ett labyrintspel där spelaren navigerar sig i labyrinten.

Krav:

- Grafik på LCD-skärm i form av en labyrint.
- Styrenhet. Antingen joystick eller 4 knappar.

I mån av tid:

- Vi ska exprimentera med hur snabbt man kan rita på LCD-skärmen och om det går att göra en väldigt simpel 3d-labyrint. Annars blir det en tråkig 2d-motsvarighet.
- Egengenererat ljud med D/A-omvandlare och högtalare.

3 Hårdvara

3.1 Komponenter

Processor

Hjärnan i systemet är en `Motorola 68008`. Den kan adressera 1 Mb minne, har en 8 bitars databuss och räknar med 32 bitar internt. Den snurrar på i 10 MHz m.h.a. en extern oscillator.

Under utvecklingsfasen används en `ICE`, *In Circiut Emulation*, vilket är en stor klump hårdvara och mjukvara som kommunicerar med en PC. Denna ersätter processorn och gör det möjligt att debugga systemet.

[1] Egentligen är det pseudoslumpmässigt, om ens det

Minne

Två typer av minne används i konstruktionen. Dels används ett RAM, dels ett ROM.

Ett ROM programmeras varefter det behåller sin data även utan strömförsörjning. Ett ROM går inte att skriva till. Det används för programkod och statisk data som processorn eller programmet behöver för att exekvera.

Ett RAM behåller inte dess data när strömförsörjningen bryts. Det kan både skrivas till och läsas från. RAM-minnet används till stacken, globala variabler och annan icke statisk programdata.

Storleken på dessa valdes i planeringsfasen till 8 Kb ROM och 4 Kb RAM. Dock används 16 Kb ROM och 32 Kb RAM p.g.a. tillgänglighet.

Display

Displayen är en LCD med upplösning 128x64 bildpunkter. Internt består displayen av två stycken 64x64 displayer med kontrollkretsar. Den har därför två stycken chip select.

Till displayen kan det skickas både data och instruktioner. Instruktionerna används bl.a. till att ändra var i bildminnet datan ska skrivas. Sättet som detta görs på är anpassat till att rita rader av tecken. Detta medför att uppritningen av enstaka pixlar kompliceras något.

Displayen är *synkron* och processorn är *asynkron*. En asynkron krets skickar en signal `DTACK` när en dataöverföring är klar och processorn kan gå vidare. Detta görs inte av en synkron krets och därför krävs ett externt nät för att sätta processorn i ett synkront läge.

Knappar

Kontrollerna till spelet utgörs av fyra knappar vilka används för att navigera genom labyrinten. Knapparna är kopplade till en data-latch, `74HC373`, som gör det möjligt att avläsa deras status över databussen.

Styrlogik

Eftersom flera komponenter delar på samma adress- och databuss krävs hårdvara som gör det möjligt att välja vilken komponent som ska läsas från eller skrivas till. Alla komponenter i systemet har en signal `CS`, *chip select*. När signalen är inaktiv är in- och utgångar till och från bussarna i ett högimpedivt läge och lämnar på så sätt dessa fria.

Genom att utnyttja några utav adressbitarna från processorn går det att bestämma vilken `CS` som blir aktiv. Detta utförs av en programmerbar logikkomponent, `PAL22V10`. Eftersom vissa av adressbitarna används till detta ändamål går en del av adressrymden förlorad men det är inget problem i detta fall.

Avbrott

Två av processorns insignaler bestämmer om och vilket avbrott som ska genereras. För att generera avbrott med lagom intervall används en räknare som delar ner oscillatorns klockpuls med en faktor 2^{14}. Det medför att ett avbrott genereras ca 60 gånger per sekund.

Då processorn befinner sig i avbrottshanteringsläge blir signalerna `FC0-2` aktiva. Dessa utnyttjas till att återställa processorn till normalt läge och bekräfta att avbrottet hanterats. Detta kan medföra problem om avbrottsrutinen tar för lång tid (mer än 1/60 sekund). En alternativ lösning är att bekräfta avbrottet via mjukvara genom att utnyttja styrlogik för att generera samma signal.

Processorn har två olika typer av avbrott. Autovektoravbrott som kör en avbrotssrutin på en förprogrammerad adress samt en annan typ som läser adressen från adressbussen. Labyrintspelet använder autovektoravbrott.

4 Mjukvara

4.1 Avbrottshantering

Labyrintspelet har ingen egentlig användning för avbrott. De inkluderades i inlärningssyfte och används för att läsa knapparnas status på jämna intervall. Utan avbrott hade det gått precis lika bra att läsa deras status i början av programmets huvudloop.

Som en eftertanke används avbrottsrutinen även för att göra slumpgeneratorn mer slumpmässig. Slumpgeneratorns seed ändras varje gång avbrotssrutinen körs. Därmed blir labyrinten annorlunda även vid omstart av systemet.

4.2 Labyrintgenerering

Labyrinten genereras med en algoritm kallad *Depth-First Search*[2]. Algoritmen lämnar mycket att önska vad gäller labyrintens svårighetsgrad. Dock är den lätt att implementera. Dessutom gör den begränsade ytan att algoritmens brister blir extra tydliga.

Labyrinten består av rutor i ett rutnät. Dess ursprungsläge är att alla rutor har väggar.

Algoritmen fortsätter sedan med följande steg:

1. Börja på en slumpmässig ruta
2. Leta efter en granne som inte är besökt tidigare
3. Om en obesökt granne hittas, flytta dit och slå ner väggen mellan rutorna. Om ingen hittas så flytta tillbaka till föregående ruta.
4. Repetera steg 2 och 3 tills alla rutor är besökta

Slumptalsgenerering

Standardbiblioteket som medföljde kompilatorn undveks eftersom det enda som saknades var en funktion för att generera slumptal. Att generera slumptal visade sig vara mer komplicerat än vad som först antogs. Efter flera misslyckade försök föll valet på en algoritm som räknar ut CRC. Resultatet för denna uträkning används som indata till nästa uträkning och därmed genereras en till synes slumpmässig talföljd.

[2] http://www.mazeworks.com/mazegen/mazetut/index.htm

5 Sammanfattning

I planeringsfasen blev vi övertalade att använda Motorolan istället för AVR. I efterhand visade det sig att vi lärde oss mycket mer med Motorolan än vad vi gjort om vi använt AVR ATMega.

Våra initiala planer på att bygga ljudkretsar visade sig innebära mer arbete än vad tiden tillät. Vi valde därför på ett tidigt stadie att utesluta ljudkretsen helt.

Om mer tid funnits hade enkel 3D-grafik bestående av linjer absolut varit möjligt. Skärmen var tillräckligt snabb för detta ändamål och processorn likaså.

Kursen har gett oss en större förståelse för hur ett datorsystem och dess individuella komponenter samverkar med varandra. Den största utmaningen för oss låg i att förstå och designa hårdvaran.

6 Kod för styrlogik

```
Title Digital Project
Pattern PAL1
Author digp05
Date 16 April 2008

device 22V10
A14 2 'Används inte
A15 3
A16 4
A17 5
A18 6
VPA 7
AS 8
DS 9
RW 10
VMA 11
INTRESET 13
CSEPROM 14
CSRAM 15
CSDISP1 16
CSDISP2 17
CSBUTTONS 18
RD 19
WR 20
DTACK 21
VPACPU 22
SYNCKRETS 23

start
CSEPROM /= /AS * /A15 * /A16 * /A17;
CSRAM /= /AS * A15 * /A16 * /A17;
CSDISP1 /= /VMA * /A15 * A16 * /A17;
CSDISP2 /= /VMA * A15 * A16 * /A17;
CSBUTTONS /= /AS * /A15 * /A16 * A17;
RD /= /DS * RW;
WR /= /DS * /RW;
DTACK /= /CSEPROM + /CSRAM + /CSBUTTONS;
SYNCKRETS = /AS * A16 * /A17;
VPACPU /= /VPA + /INTRESET;
end
```

7 Kretsschema