Digitala projekt rapport

Alexander Westrup, d04aw@student.lth.se
Martin Sandgren, d04ms@student.lth.se

4 december 2007

Innehåll

1 Abstract

Denna rapport är till kursen EDI021 Digitala Projekt om vårt projekt "Bitmasken Bertil". Projektet innefattar konstruktion av både hårdvara och mjukvara. I rapporten går vi igenom alla hårdvarukomponenter som ingår i vår konstruktion och hur vår mjukvara är uppbyggd för att arbeta med hårdvaran. Under kapitlet 5.1 går vi igenom extra intressanta mjukvarulösningar.

2 Inledning

Vårt syfte med detta projekt var att göra något häftigt. Vi snöade snabbt in på något sorts spel och beslöt oss tillslut att göra ett *Snake*-liknande spel. Vi valde att jobba med Motorola 68008 därför att vi ville få en bred kunskap över hela konstruktionsarbetet. Under projektets gång har vi stött på en hel del problem, men med hjälp av andra har vi tagit oss igenom allihop och resultatet blev över förväntan. Vi fick ett roligt spel med många features, t.ex. så har vi stöd för joysticks och animerad grafik.

3 Arbetsgång

Vår första uppgift var att komma på en ide för projektet. Därefter utvecklade vi hårdvaran genom att först designa hur alla komponenter är kopplade och kommunicerar med varandra och sedan sätta fast alla komponenter och vira all tråd mellan dem. Sista steget i hårdvarukonstruktionen var att testa att allt funkade som det var tänkt. Mjukvaran var tänkt att utvecklas parallellt men alla problem med hårdvaran gjorde att vårt fokus var på den. Utvecklingen av mjukvara gick snabbare än väntat så att vi kunde lägga in lite häftiga saker.

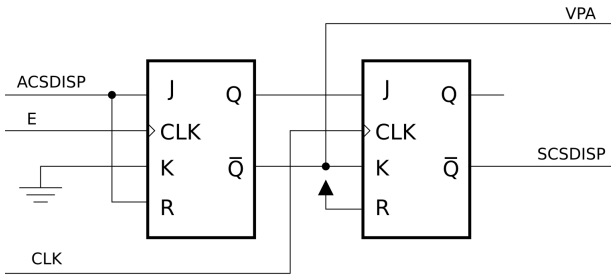
4 Hårdvara

4.1 Processor

Processorn är en Motorola på 8 Mhz, den har en databus på 8 bitar samt en adressbus på 20 bitar. Detta innebär att processorn har 1 Mb adresseringsutryme som kan användas till externa komponenter. Eftersom processorn är en assynkron processor så måste alla assynkrona periferienheter skicka **DTACK** när enheten är klar. Processorn kan ta emot 2 nivåers avbrott, vi använder dessa för vår A/D-omvandlare och räknade.

4.2 Klocka

Klockan vi använder är en exo 3 som har en grundfrekvens på 16 khz. Man kan dock inte få ut detta utan bara fraktioner. Till vår krets använder vi 8 khz. Denna signal går in i processorn samt skärmens synkroniseringsmodul. För att



Figur 1: Synkroniseringsmodul till skärmen

driva de flesta andra periferienheter används signalen **E** från processorn, den är på 800Hz.

4.3 SRAM

Vårt SRAM på 8 kbyte är det minne i vår krets som innehåller stacken och variabler som programmet använder under körning. Minnet nollställs vid frånvarande av ström.

4.4 EPROM

EPROMet är en minnesmodul på 32 kbyte vars uppgift är att lagra all statisk information såsom programkod, statiska variabler och div. dataobjekt t.ex. bilder. Vi valde en EPROM med en kapacitet på 32 kbyte då våra bilder för helskärm har vardera en storlek på 1 kbyte.

4.5 Batron / Skärm

Vår skärm är en liten monokron LCD på 128x64 pixlar. Den har ett inbyggt minne som är 1 kbyte stort där skärmbilden sparas. En väsentlig del för att få skärmen att fungera korrekt är dess synkroniseringsenhet. Från denna enhet använder vi signalen **VPA** för att sätta processorn i synkront läge. **SCSDISP** skickas vidare till vår logikkrets som gör att vi kan skicka synkrona *Chip Select* till skärmen, se figur 1. Eftersom denna enhet alltid körs i synkront läge behöver den inte generera **DTACK** till processorn.

4.6 Logik

I vår krets använder vi två stycken logikkretsar, en ansvarar för *Chip Select* för I/O-enheter och den andra har mest hand om avbrott. Kretsarna programmeras med boolsk logik i särskilda filer som sedan skrivs till kretsen av speciella verktyg.

4.7 Räknare

Vår räknare är en 14-bitars räknare som har processorns **E**-signal som klocka. Ut ur räknaren tar vi signaler till avbrott och som startpuls till vår A/D-omvandlare. Räknaren nollställs via logiken för att få en fördröjning.

4.8 Latchar - Lysdioder

Latchen har till uppgift att hålla värden under körning. Vår latch är kopplad till lysdioderna och bestämmer vilka som ska lysa. Eftersom det är dioder på utgångarna så har vi kopplat på sådant sätt att latchen aldrig behöver använda sin *tristate*-funktionalitet. För att lamporna inte ska överbelastas använder vi ett motståndsnät.

4.9 A/D-omvandlare

Vår A/D-omvandlare har 8 kanaler och omvandlar med 8 bitars upplösning. Utav de åtta kanaler som finns används endast fyra kanaler, två för varje joystick. För att starta en omvandling skriver man till den kanalen man avser. När en omvandling är klar skickas ett avbrott till avbrottslogiken och då kan man läsa av databussen. För att A/D-omvandlaren inte ska ta all processorkraft har vi valt att sänka dess klocka. Detta medför att vi i vårt program kan starta en omvandling och direkt när vi läst av värdet, starta nästa omvandling.

5 Mjukvara

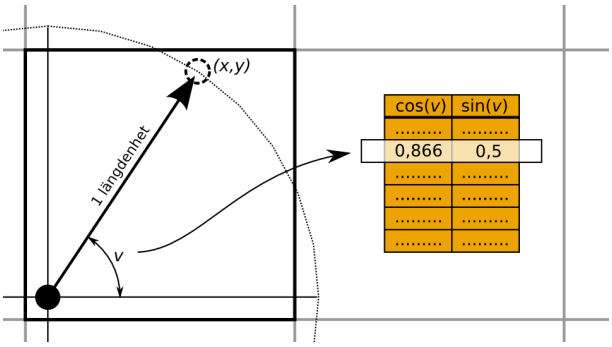
För att kommunicera med hårdvaran så har varje periferienhet en egen adress, denna adress är mappad i logiken. I vårt program har vi ett huvudprogram och två avbrottsrutiner.

5.1 Huvudprogram

För att kunna köra spelet har vi implementerat många funktioner bl.a. funktioner för att rensa skärmen, rita hårdkodade bilder och rita ut enskilda pixlar. För att få en jämn och fin rörelse i spelet har vi valt att implementera *Fixed point*-aritmetik, vilket innebär att masken har koordinater innanför en enskild pixel. För att kunna utnyttja att joystickarna är analoga och ger olika utslag beroende på hur mycket man drar, så har vi implementerat sinus- och cosinustabeller med 64 olika vinklar på 360°. Tack vare detta kan vi kontrollera att masken aldrig åker mer än 1 längdenhet i taget även om masken åker diagonalt, se figur 2.

5.1.1 Bitmasken Bertil

Innan spelet startar så nollställs skärmen och liven till vardera spelare sätt till fyra. Därefter börjar spelet och under en spelcykel händer följande:



Figur 2: Ny position på subkoordinater

1. Programmet väntar på avbrott från klockan.
2. Spelarens nya riktning hämtas från tabellerna utifrån vinkeln på joysticken.
3. Den nya positionen räknas ut och motsvarande pixel fylls i om spelarens position är på en nya pixel.
4. Om pixeln redan är utritad så har spelaren krockat, då dras ett liv från spelaren och en ny omgång startar. Om spelaren inte har fler liv så avslutas spelet.

5.2 Avbrott

Avbrotten körs oberoende från main-funktionen och kallas på av processorn när den får externa vektoriserade avbrott.

5.2.1 A/D-omvandlare

När denna anropas så läses A/D-omvandlaren av och sparar värdet hos korrekt spelare och axel. Därefter startas direkt en ny omvandling på nästa kanal. Detta medför att vi alltid har uppdaterade variabler då A/D-omvandlaren körs flera gånger i sekunden.

5.2.2 Räknare

Detta avbrotts enda uppgift är att kontinuerligt avbryta programmet för att signalera ny uppdatering.

6 Slutsats

Under vårt projekt har en del större och en massa mindre problem dykt upp. Av varenda en av dessa har vi lärt oss en massa. Som datavetare är vår vanliga miljö mjukvaran men under detta projekt har vi fått en god förståelse även för hur hårdvaran fungerar.

Det största problemet vi hade var nog att vi saknade den övergripande förståelsen för hur allt hänger ihop hårdvarumässigt, detta medförde att vi hade svårt att veta vilka signaler som skulle gå vart och detta tog extra lång tid.

Projektet har varit väldigt roligt och inspirerande och vi kommer att rekommendera den till alla som frågar.

Som tack för den utomordentliga hjälp vår handledare (med kollegor) har gett oss under hela projektet så har vi namngett vårt eminenta spel efter honom, ”Bitmasken Bertil”.

A Kopplingschema