Bubble Bobble

Designskiss

Projektmedlemmar:

Natanael Log, natlo809
Matti Lundgren, matlu703
Oscar Johansson, oscjo411

23 mars 2015

1. Inledning

Vi ska bygga Bubble Bobble, ett klassiskt arkadspel där man styr en drake som skjuter små bubblor på spöken. Spelaren styr med hjälp av en joystick och skjuter med en knapp. När ett spöke blir träffat av en bubbla flyter det sakta uppåt på spelplanen tills bubblan spricker om inte spelaren spräcker bubblan själv. Spelaren får poäng för varje fångat spöke den lyckas spricka och förlorar liv varje gång ett spöke nuddar draken. Spelet fortsätter tills draken förlorat alla sina liv.

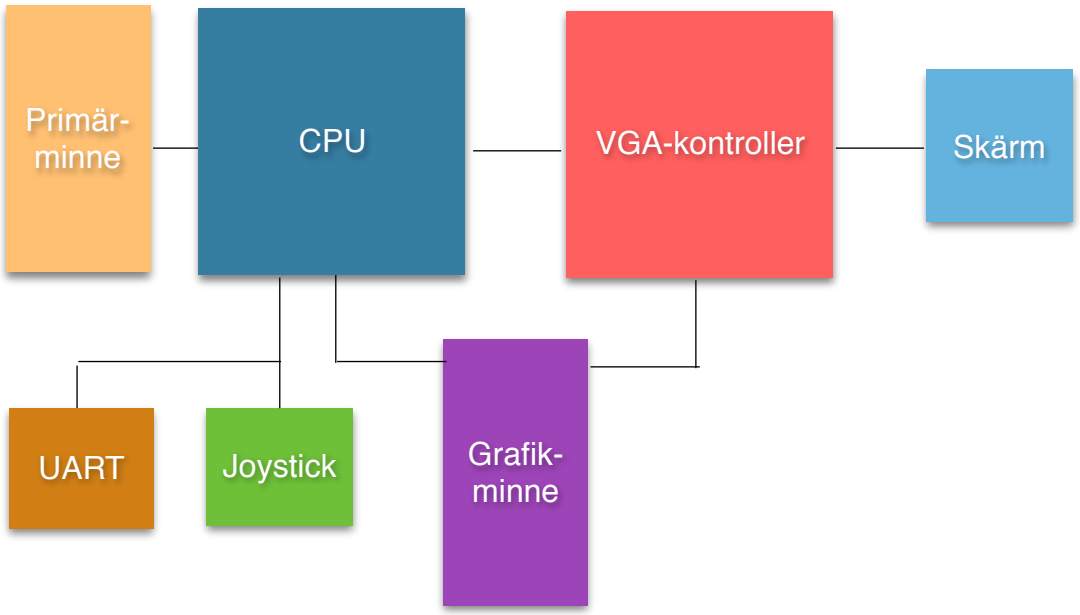
Vår maskin kommer dels bygga på Olle Roos arkitektur för en processor och dels en VGA-kontroller vi själva kommer bygga, inspirerad av föreläsningar från kursen TSEA83. Även en joystick och en VGA-kompatibel skärm kommer användas för att få ut en maximal spelupplevelse!

2. Analys

All spellogik kommer vara programmerad i primärminnet. Utöver vissa instruktioner för att skicka ut data till skärmen kommer endast generella instruktioner användas i systemet, så alla instruktions-rader i minnet kommer bli ganska många. Detta skapar ett behov av att kunna skriva assembly-kod i en textredigerare som sedan kan kompileras till binärkod och laddas upp i primärminnet i maskinen.

3. Konstruktion

Såhär kommer maskinen till stor del se ut:



3.1 CPU

CPU:n kommer vara av typ Olle Roos. Ordbredden kommer bli 32 bitar, främst på grund av att vi behöver ett större minne än 256 rader.

3.1.1 Primärminnet

Maskininstruktionerna har följande format:

Fält	antal bitar	anger
Op	4	vilken instruktion som avses
GRx	4	vilket register som ska användas
PG	1	om instruktionen ska skriva till primär- eller grafikminnet
M	2	adresseringsmod
ADR	16	adressfält

Bitar kvar: 5
Om det skulle visa sig att vi behöver fler operander sparar vi undan några bitar till det.

Assembly-instruktioner den kommer stödja:

- MOVE
- LOAD
- STORE
- ADD
- SUB

- CMP
- BRA
- BNE
- JSR
- RTS
- AND
- OR
- LSR
- LSL
- BRC (branch on collision)

Vi har en outnyttjad plats kvar i K1 som vi avvaktar med att bestämma.

Adresseringsmoder den kommer stödja:
- Direkt adressering
- Omedelbar operand
- Indirekt adressering
- Indexerad adressering

Alla instruktioner och adresseringsmoder är hämtade från Olle Roos-konceptet och följer det förutom BRC som ska underlätta för kollisioner av sprites på spelplanen.

3.1.2 Joystick

Joysticken är kopplad med en SPI-gränssnitt. För att hämta dess data skiftar vi in bytes till MOSI-porten får sen tillbaka bytes med data i MISO-porten. Dessa data lagrar vi i något register.

3.1.3 UART

UART kommer vi använda för att ladda upp vår kompilerade assembly-kod i datorns primärminne. Binärfilen kommer ha en trigger-rad i början som startar igång en assembly-instruktion som börjar läsa in 32 bitar och lägger det i ett register som sedan laddas in i en rad i primärminnet. Detta fortsätter tills en avslutningsinstruktion nås i binärfilen och mikroprogrammet stoppar inläsningen.

3.1.4 Grafikminnet

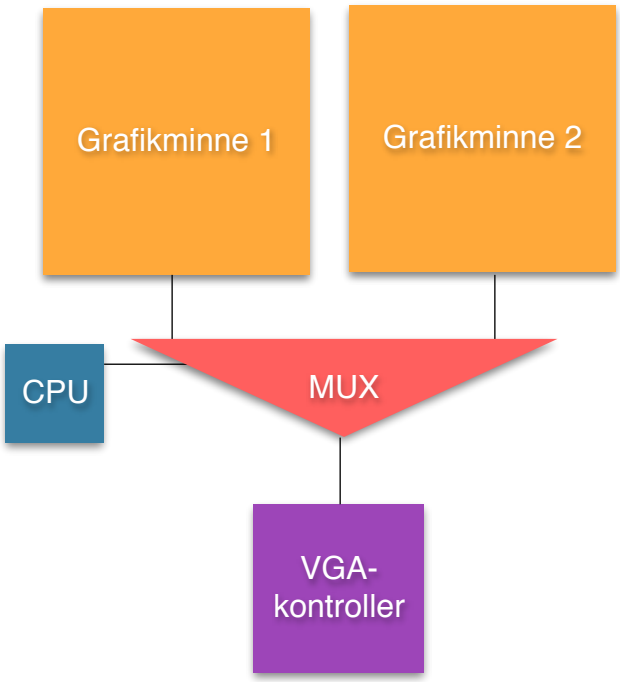
Läs om detta i nästa kapitel.

3.2 VGA-kontroller

VGA-kontrollern ansvarar för att skriva ut det som står i grafikminnet.

3.2.1 Grafikminnet

Grafiken till spelet kommer bygga på ett dubbelbuffrat minne. CPU:n kommer styra vilket grafikminne som ska skrivas ut och vilket som ska modifieras, VGA-kontrollern kommer inte veta vilket minne det skriver ifrån.



Grafikminnet kommer vara uppdelat på detta sätt.

3.2.2 Skärm

Skärmen kopplas med ett VGA-gränssnitt.

- HSynk (1 bit)
- VSynk (1 bit)
- R (3 bitar)
- G (3 bitar)
- B (2 bitar)

4. Milstolpe

Vid halvtidskontroll ska vi kunna rita ut en spelplan på en VGA-kopplad skärm. Det innebär att vi ska ha en färdig assembly-kompilator som ska kunna ladda in vårt program och programmet i sig ska kunna skriva ut en spelplan. Det enda som inte ska vara klart då är själva spellogiken.