TSEA83
designspecifikation

Tower-Defense spel

Max Wilhelmsson	maxwi609
Daniel Svensson	dansv959
Daniel Alchasov	danal315
Samuel Åkesson	samak519

Version 0.1

Innehåll

1 Inledning

Detta dokument är en designspecifikation för en implementation av ett ´Tower Defense´-spel på en FPGA-enhet (Basys-3). Spelet går ut på att stoppa ballonger som försöker ta sig igenom banan genom att sätta ut apor som spräcker ballongerna.

Konstruktionen består huvudsakligen av tre komponenter: processorn, VGA-motorn och tangentbordsavkodaren. Processorn ansvarar för uppdatering av spelets tillstånd samt uppdatering av videominnets innehåll. VGA-motorn ansvarar för att rita upp videominnets innehåll på bildskärmen. Tangentbordsavkodaren ansvarar för att läsa av tangenttryckningar på det anslutna tangentbordet och ge enkla, avkodade signaler till processorn.

Ett detaljerat blockschema över hårdvaran syns i figur 1.

2 Analys

2.1 Klocka

VGA-motorn kommer att drivas av fjärdedelspulser från Basys-3:s klocka (som är på 100 MHz), och kommer då klockas i 25 MHz. Resten av spellogiken och uppdatering av videominne borde alltså ha gott om tid att uppdateras innan nästa utritningscykel kommer.

2.2 Processor

Processorn tänks sig att vara mikroprogrammerad med en CISC instruktionsuppsättning. Det utvecklas vidare kring instruktionsuppsättning i avsnitt 3.1. ordbredden tänks vara 24 så närmaste större tvåpotens är 32. Orden tänks just nu vara utdelade på 5 bitar för OP, 3 bitar för generella register då det tänks vara 8 styckna. 2 för addreseringsmetoder och tänker på 12 bitar för ADR.

2.2.1 ALU

Kan bland annnat genomföra addition, subtraktion, multiplikation och andra aritmetiska och logiska operationer. Alla operationer reflekteras i processorns instruktionsuppsättning, som finns i tabell 2.

Beroende på vilken operation som används så uppdaterar vi statusflaggorna (N, V, Z, C) i ett statusregister därefter.

2.2.2 Register

8 generella register tänks användas inom arkitekturen av processoren.
Instruktionsregistret IR måste vara åt det bredare hållet för att all information ska få plats i ett och samma instruktionsord. Vi tänker oss att det går åt minst 22 bitar till IR.

Den generella registerbredden sätts till 16 bitar då datorn behöver 10 bitar bara för att hantera grafikens X- och Y-led.

Stackpekaren lagrar vi i det separata stackregistret SP. Registret kommer behöva ha en tillräckligt stor bredd för att kunna flytta sig uppåt från sista raden i huvudminnet. Om huvudminnet har rader upp till $FFF så behöver bredden på SP vara 12 bitar.

2.3 Minne

Processorns huvudminne kommer att innehålla programkod, heap (data), och stack. På grund av detta måste uppdelning ske baserat på adress.

2.3.1 Programkod

Här lagras all all assembly-kod (och eventuell bootloader-kod). Vi räknar med att det kommer kräva omkring 1000 rader kod, alltså kan det ligga från adress $000 till $3F0 (1008 rader).

2.3.2 Heap

Här lagras alla ´globala´ variabler kring spelet. Några exempel illustreras i tabell 1.

Namn	**Beskrivning**
Ballonger	Lista över alla instantierade ballonger med dess tillhörande variabler: position, liv, färg, animationscykel.
Apor	Lista över instantierade apor med dess tillhörande variabler: position, typ, animationscykel.
Spelare	Hur mycket poäng och liv spelaren har.

Tabell 1: Globala variabler

Vi har inte tänkt på något sätt att förhindra att heap:en och stacken krockar med varandra men eftersom heap börjar på $3F1 och stack börjar på botten i $FFF betyder det att det finns 3086 raders separation, alltså bör inte vara något problem.

2.3.3 Stack

För att kunna använda bland annat subrutiner så implementerar vi en stack. När man kallar på en subrutin så sparas programräknarens (PC) värde vid stackpekaren; därefter räknar stackpekaren ned (-1). Vid återhopp (returns) ökar stackpekaren (+1) och sätter programräknaren till den radens data.

Stacken kan också användas som en ´parameter-stack´ när vi vill slussa data mellan subrutiner. Då används `push` och `pop` för att ladda/lagra värden på stacken.

2.4 Tangentbordavkodning

Tangentbordet avkodas med en speciell hårdvarukomponent. Den hanterar PS/2-signalen genom att sekventiellt slussa in data i ett skiftregister. En tangenttryckning består egentligen av en seriell ström av data, men avkodarens skiftregister kommer endast att ladda det färdiga meddelandet till databussen.

Tangentbordavkodarens uppgift är alltså att gränssnitta mellan PS/2 signaler och processorn. Processorn kommer att polla efter signaler från avkodaren så att tangenttryckningar upplevs hända i realtid.

2.5 Grafik

Det kommer finnas ett videominne som innehåller alla tiles på skärmen och berättar vilket index i tile-ROM som behöver läsas ur för att rita dess utseende.

Videominnet kommer att modifieras av processorn, så VGA-motorns enda uppgift är att läsa utseendet för den specifika tile som den är på genom att söka upp i tile-ROM.

2.6 Ljud / musik

Piezohögtalaren kommer att styras av en digital pin på FPGA-enheten. Den kommer att styras till att pipa i olika frekvenser genom subrutiner i programkoden. Potentiellt kommer det senare att experimenteras med att programmera musik eller till och med sampling.

3 Programmering

3.1 Instruktionsuppsättning

Vårt assembly-språk (som programkoden i huvudminnet är skrivet i) kommer att klara av instruktionerna som anges i tabell 2.

OP	**Instruktion**	**Betydelse**
$0	LOAD: GRx,M,Addr	GRx := PM(A)
$1	STORE: GRx,M,Addr	PM(A) := GRx
$2	ADD: GRx,M,Addr	GRx := GRx + PM(A)
$3	SUB: GRx,M,Addr	GRx := GRx - PM(A)
$4	CMP: PC,M,Z,Addr	GRx - ADR (Sätt endast flaggor)
$5	AND: GRx,M,Addr	GRx := GRx AND PM(A)
$6	OR: GRx,M,Addr	GRx := GRx OR PM(A)
$7	BEQ: PC,Z,Addr	if Z = 0, PC := PC + 1 + ADR else PC := PC + 1
$8	BNE: PC,Z,Addr	if Z = 1, PC := PC + 1 + ADR else PC := PC + 1
$9	BRA: PC,Addr	PC := PC + 1 + ADR
$A	JSR: PC,SP,Addr	PM(SP) := PC, PC := PC + 1 + ADR, SP - 1
$B	RET: PC,SP,Addr	SP + 1, PC := PM(SP)
$C	PUSH: SP,M,Addr	PM(SP) := GRx, SP - 1
$D	POP: SP,M,Addr	GRx := PM(SP), SP + 1
$E	HALT	Avbryt exekvering

Tabell 2: Processorns instruktionsuppsättning

3.2 Assembler

Det kommer att skrivas ett pythonskript tar `.asm`-filer och omvandlar till maskinkod. Denna maskinkod kan antingen skickas genom UART till FPGA-enheten (när vi väl har skapat en bootloader), eller inkluderas i VHDL-koden så att huvudminnet syntetiseras med färdigt innehåll.

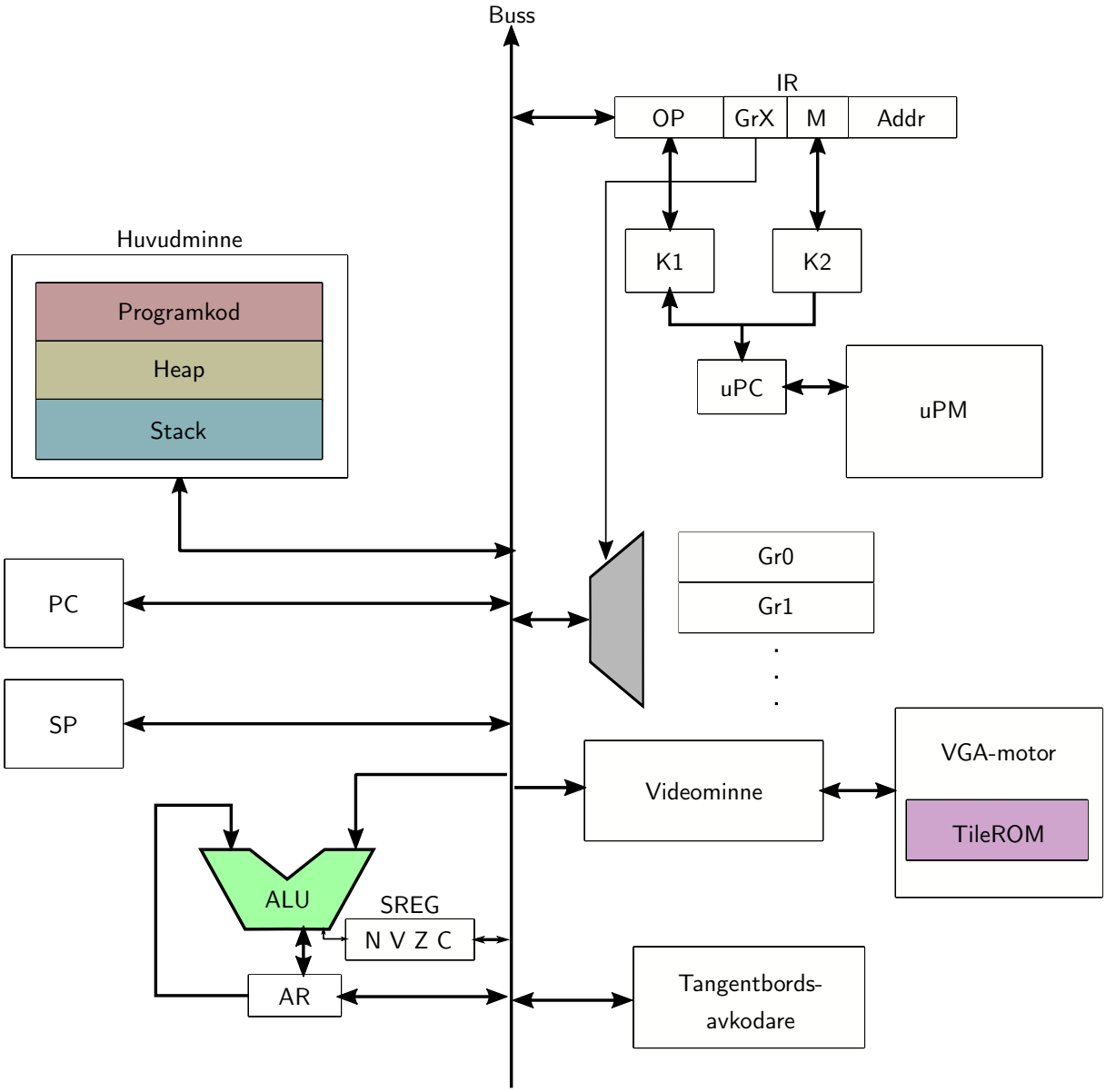
3.3 Bootloader

Senare under projektet kommer vi att skriva en bootloader som gör att vi kan trycka programkod genom UART från en dator, utan att syntetisera om all hårdvara.

4 Milstolpe

Efter halva projekttiden har gått så bör mikrokoden vara helt färdigskriven, och VGA-motorn kommer fungera så att vi kan visa tiles på skärmen.

5 Detaljschema



Figur 1: Detaljschema över alla hårdvarukomponenter i konstruktionen