TSEA83
Monkey computer, teknisk dokumentation

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

Version 1.0

Abstract

This report describes the development of `monkey computer`, a project that was carried out during the second period of the course TSEA83. The project was successfully completed and resulted in a computer that could play a simple "tower defense" game.

Innehåll

1 Inledning

Denna rapport beskriver utvecklingen av `monkey computer`, ett projekt som genomfördes under andra perioden av kursen TSEA83. Målet med projektet var att bygga en fungerande dator med hjälp av ett Basys3 FPGA-kort. Rapporten går främst igenom hårdvaran som konstruerades. Texten tar upp flera av modulerna i slutkonstruktionen som CPU, tangentbordsavkodare och VGA-motor. Störst fokus läggs på bygget och logiken bakom CPU:n.

Under rapporten kommer orden "processor", "CPU" och "dator" att användas synonymt. Vid referering till hela konstruktionen kommer ord som "konstruktionen", "maskinen" eller "`monkey computer`" att användas. Orden "instruktion" och "ord" kommer också att användas synonymt.

2 Apparaten

Under detta avsnitt diskuteras översiktligt vilka anslutningar som `monkey computer` har och hur den används.

2.1 Översikt

Maskinen är byggd på ett Basys3 FPGA-kort. Den "pratar med omvärlden" genom en extern skärm och ett tangentbord som kopplas in i kortet. Detta syns i figur 1.

Figur 1: FPGA-enheten är sammankopplad med en skärm och ett tangentbord

Vi valde att använda ett tangentbord för att hämta indata från användaren. Tangentbordet kopplas med USB in i kortet. För att se spelet på en skärm används VGA-porten.

2.2 Användning

3 Teori

För att förstå `monkey computer` är det bra att känna till de koncept och tekniker som konstruktionen bygger på. Detta avsnitt tar upp de viktigaste.

3.1 Mikroprogrammerad processor

Att en processor är mikroprogrammerad betyder att det för varje assembly-instruktion finns flera mindre instruktioner. Dessa mindre instruktioner kallas mikroinstruktioner, är skrivna i mikrominnet och är väldigt begränsade i komplexitet. Något typiskt som en mikroinstruktion gör, är att säga åt bussen att skicka information från ett register till ALU:ns aritmetiska register.

Varje mikroinstruktion tar exakt en klockcykel att genomföra, och beroende på hur invecklad assembly-instruktion som genomförs kan det ta olika antal mikroinstruktioner/klockcykler att genomföra.

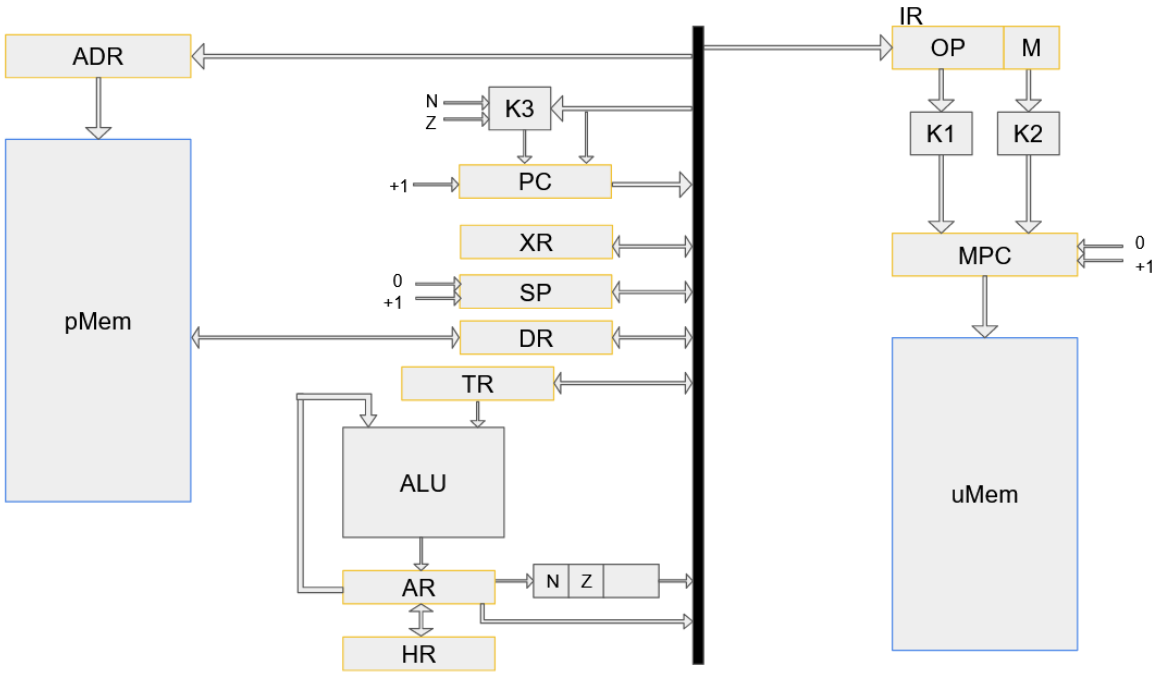
Till skillnad från i en pipelinad dator behöver ingenjören inte tänka på att instruktioner kan köras parallellt, däremot så blir en mikroprogrammerad dator långsammare på grund av dess sekventiella natur.[4].

3.2 Olle-Roos dator

Datorn som byggts är inspirerad av Olle-Roos (OR) datorn (figur 2), med vissa ändringar på register. OR-datorn är en mikroprogrammerad dator som genomgår tre faser vid varje instruktionsutföring: hämtfas, adresseringsfas och exekveringsfas. Hämtfasen hämtar instruktionsordet från programminnet och lagrar den i instruktionsregistret (`IR`). Detta sker på samma sätt oberoende av assembly-instruktion.

Efter hämtfasen går datorn vidare till adresseringsfasen, då `M`-fältet från `IR` läses av och hanteras. FÖRKLARA ADDRESERINGMODER Det som kan skilja sig i denna fas mellan instruktioner är hur `ADDR`-fältet tolkas.

Sista fasen, exekveringsfasen exekverar instruktionen, och skiljer sig mellan alla olika instruktioner. [4]



Figur 2: En Olle-Roos dator

3.3 FPGA

En FPGA-enhet (Field programmable gate array) är ett programmerbart grindnät. FPGAs nät kan enkelt programmeras om genom att skriva kod i hårdvarubeskrivande språk. Den skrivna hårdvarukoden kan sedan syntetiseras, vilket kan liknas vid mjuvarekompileringsprocessen, och laddas in i FPGA-enheten. Denna process gör att FPGA-enheten kan utföra en mängd olika uppgifter beroende på vilken kod som laddas in [1].

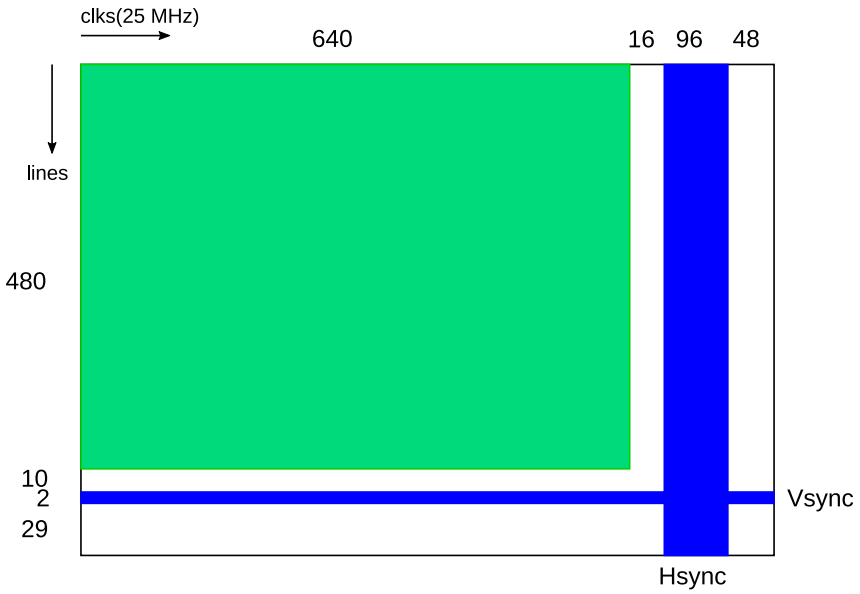
3.4 VHDL

VHDL är ett hårdvarubeskrivande språk vilket betyder att det används för syntetisering av kretsar på exempelvis FPGA kort [2]. Att det är ett hårdvarubeskrivande språk betyder att man inte programmerar i det på det typiska mjukvarusättet genom tilldelning av variabler och sedan använda aritmetik, utan målet med VHDL är att bestämma hur grindnätet på FPGA kortet ska fungera och bete sig genom att beskriva gränssnitt med "entity" och innehållet med "architecture". Utöver det så kan man även göra kombinatorik och skapa muxar med "when-else", det finns även process(clk) vilket använder sig av if satser och klockas samtidigt som alla andra process(clk) [5].

Den största skillnaden i logik mellan VHDL och programmeringsspråk som C är att VHDL är parallellt medan C är sekventiellt. Detta betyder att i VHDL så kommer alla processer att köras samtidigt medan i C så kommer en process att köras efter den andra.

3.5 VGA

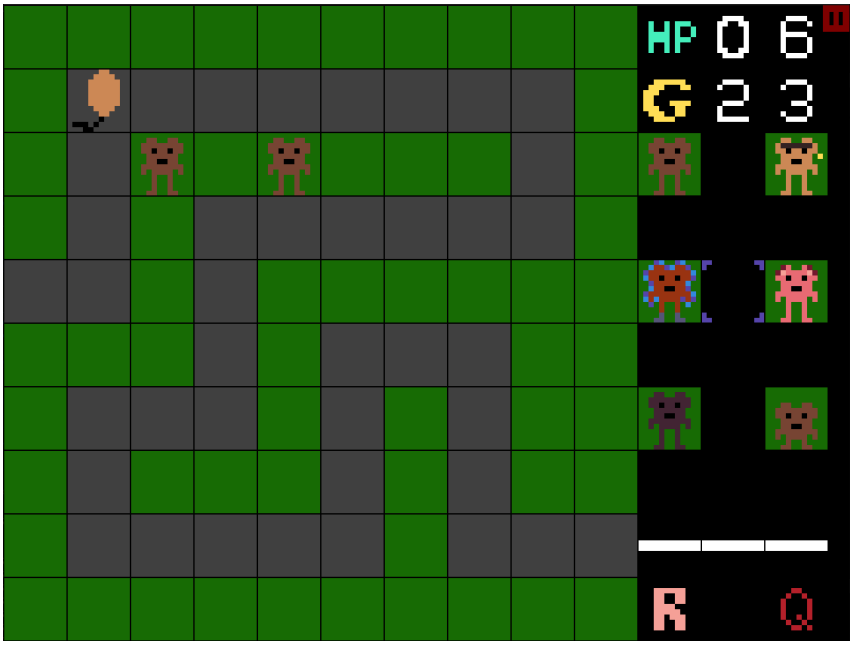
Video Graphics Adapter (VGA) är en display standard utvecklad av IBM. Den levererar analogt en 640 x 480 färgad skärmupplösning med en uppdateringsfrekvens på 60 Hz. En VGA kabel har en 15-pin kontakt där varje pin hanterar en viss funktion[3]. När VGA skärmen ska rita upp en bild så kräver det att man skickar tre analoga signaler: en för röd, en för grön och en för blå. Det krävs också att det skickas med två synkpulser för vertikal(Hsync) och horisontal(Vsync). I figur 3 visas det hur VGA skärmens inmatning sker. Det gröna på bilden är den synliga 640 x 480 skärmen där de analoga signalerna skickas som vanligt. När man kommer utanför skärmen måste man köra "blanking" dvs skicka svart videosignal. De blåa strecken i figuren är när respektive synkpuls måste skickas, de är båda aktivt låga.



Figur 3: VGA timing

3.6 Tilebaserad grafik

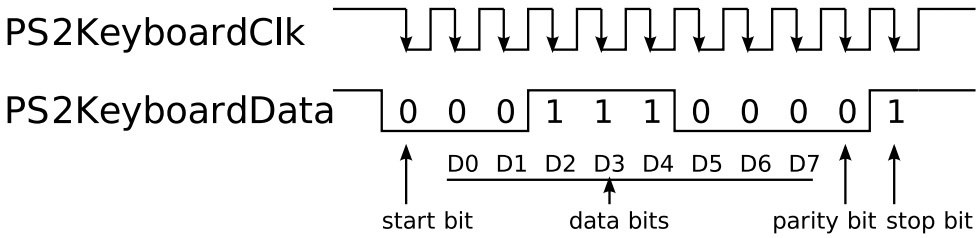
Tilebaserad grafik är en teknik för att rita upp en skärm i spel. Istället för att skapa logik för, och hantera varje pixel på skärmen, delas skärmen upp i mindre delar, så kallade "tiles". Dessa tiles är små bilder som kan placeras i ett bestämt rutnät på skärmen. Genom att använda denna teknik sparas också minne, då samma tile kan användas flera gånger på skärmen.



Figur 4: En skärmdump från spelet, med extra tjocka linjer för att demonstrera dess tile-indelning

3.7 PS/2

PS/2 är en seriell dataöverföringsstandard som används för att koppla in tangentbord och mus till datorer. Standarden använder två ledningar, en för data och en för klocka. Varje dataöverföring ser ut på följande vis: En startbit, 8 databitar, en paritetsbit och slutligen en stoppbit. I figur 5 visualiseras en dataöverföring.



Figur 5: PS/2 dataöverföring

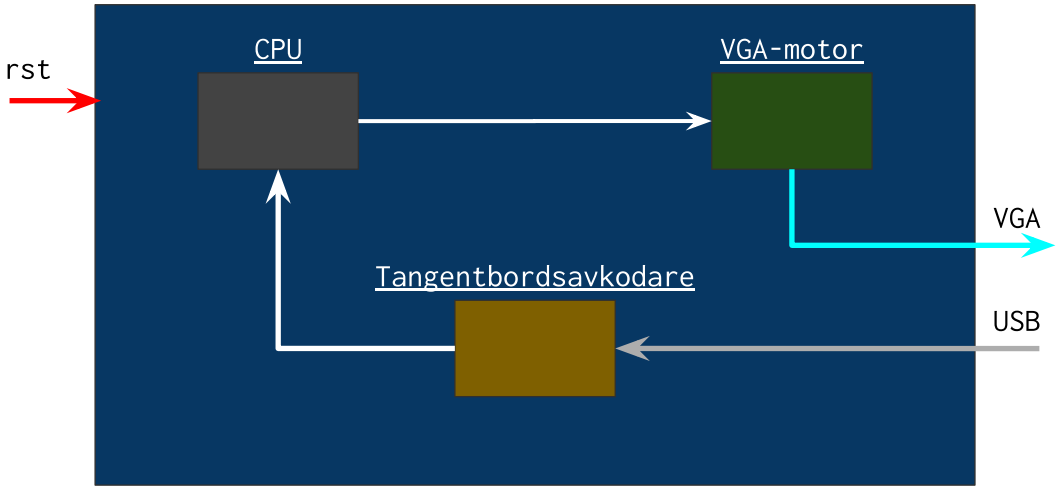
4 Hårdvara

Hårdvaran har skapats på FPGA-kortet som har blivit programmerad med hjälp av VHDL (Hardware descriptive language) för att skapa en processor, en tangentbordsavkodare och en VGA motor.

4.1 Toppmodul

Generella arkitekturen av konstruktionen följer figuren av toppmodulen 6.

Tangentbordsavkodaren får PS/2 data från ett tangentbord genom FPGA:ns USB-port. Signalen avkodas, och skickas till CPU:n (beskrivs i avsnitt 4.3). VGA-motorn läser ur CPU:ns videominne för att skicka ut en VGA 640x480 standard bild (beskrivs i 3.5)



Figur 6: Toppmodul av hårdvaran

4.2 CPU

CPU:n är en mikroprogrammerad processor med en instruktionsbredd på 24 bitar och tar inspiration av Olle-Roos datorns uppbyggnad (se figur 2). Huvudkomponenterna i processorn är programminnet (pMem), ASR, program räknaren (PC), stack pekaren (SP), ALU, 16 generella register (GRx), instruktionsregister (IR), databussen samt mikrominne sektionen som innehåller K1 och K2, mikro PC (uPC) som pekar intill mirkominnet (uMem) sen själva mikrominnet. ASR, PM, PC, ALU/AR, IR, GRx och SP är uppkopplade till databussen.

4.2.1 Programminne

Instruktioner (ord) i programminnet består av 24 bitar och följer figur 7.

Figur 7: Ord inom programminnet

Hela instruktionen består alltså av 5 fält:

- OP, 4 bitar: bestämmer vilken operation som ska utföras. K1 har som uppgift att läsa av OP och peka ut var exekveringsfasen för denna operation finns i mikrominnet.
- GRx, 4 bitar: bestämmer vilket register som ska användas. Kan vara fylld med - (don't care) om operationen inte kräver något register.
- M, 2 bitar: bestämmer vilken adresseringsmetod som ska användas. K2 har som uppgift att läsa av M och peka ut var adresseringsfasen för denna operation finns i mikrominnet.
- KEY, 1 bit: (en obsolet bit som inte används men finns kvar i ordet för att inte förändra ordlängden)
- Adress, 12 bitar: bestämmer vilken adress som ska användas. Kan vara fylld med - (don't care) om operationen inte kräver någon adress.

Om instruktionen behöver nå någon annan adress eller spara något extra värde används de 12 sista bitarna för det. Alla instruktioner är sparad i pMem som är ett två ports block ram (anledningen till detta tas upp i sectionen om VGA motor).

4.2.2 Mikrominne

Alla instruktioner som är programmerade i mikrominnet syns i tabell 1. Mikrominnet syntetiseras som en LUT som innehåller ord på 25 bitar. Bitarna på de här orden speciferar vad som processorna ska göra och hur bussen ska slussa runt data, exakt uppbyggnad av en mikroinstruktion kan ses i figur 8.

Figur 8: Fält i mikroinstruktioner

Bitarna i mikroinstruktionen är indelade i fält som figur 8. TB anger vad för komponent ska skicka sitt nuvarande värde till bussen, FB säger vad för komponent ska ta ett nytt värde från bussen. ALU-fältet säger vad för ALU-operation som ska genomföras. P-biten bestämmer om PC ska inkrementeras eller inte. S-fältet anger om SP ska öka, minska eller inget ska hända. SEQ används för att styra uPC och genomföra olika hopp inom mikrominnet. Om SEQ anger ett hopp är det uADR, det sista fältet, som anger till vilken mikroaddress hoppet ska gå.

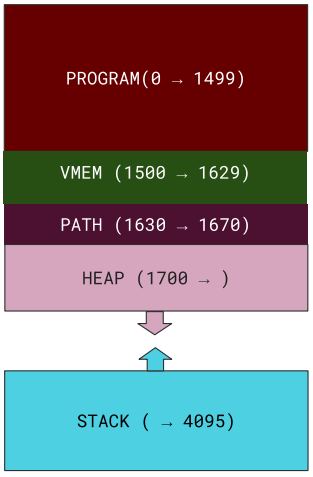
4.2.3 Instruktionsfaser

Varje instruktion börjar med en hämtnings fas där vi först hämtar instruktionen från programminnet och skickar det till IR. Detta sker med hjälp av de första instruktionerna från mikrominnet detta avslutas med att indexera intill K2 med hjälp av M-fältet (se figur 7) och skicka värdet från K2 till uPC, detta händer varje gång en ny instruktion ska köras. Adresserings fasen sker olika beroende vad för värde kom på M-fältet men generellt får den fram vad för operation som ska köras. Efter detta kommer faktiska exekverings fasen som är helt olika beroende på instruktion.

4.2.4 Minnen och register

Huvudminnet sektioneras (enligt figur 9) in i programminne, videominne, "path"-minne, heap och stack. Minnet är ett två-ports-RAM, och är 24x4096 bitar stort. Programminnet innehåller de instruktioner som CPU:n ska utföra. Videominnet innehåller det nuvarande tillståndet över vilka tiles som finns på skärmen. "Path"-minnet innehåller information över vilka tiles som är gångbara för ballonger. Heap används för att lagra "variabler" och stacken används för att lagra returadresser och kontext vid subrutiner.

Som tidigare nämnt, finns bara ett stort huvudminne.



Figur 9: Minnessektioner

4.2.5 ALU

Processorns ALU kan genomföra beräkningar på 24-bitars tal, och är den mest nödvändiga komponent för att tillåta matematiska operationer med instruktioner och för att beräkna hoppaddresser.

ALU:n
är synkron, och genomför också en operation varje klockcykel. Eftersom
ALU:n för det mesta inte behövs behöver den få `NOP`-instruktioner då.

tillåter den genomföra enkla operationer såsom ADD, SUB, MUL, LSR, LSL
och sparar värdet i sitt register (AR), dessa operationer används även för att hoppa inom programminnet. Operationerna sätter även flaggor, de som implementerades var Z flaggan som tittar om värdet är 0, C flaggan som tittar om det är en carry operation, N flaggan för negativa operationer och V flaggan för overflows. ALU:n använder sig av en extra intern signal som är en bit större än faktiska signalen. Detta gjordes för att få enklare beräkningar för flaggorna, till exempel kan V flaggan sättas beroende på sista bitten på sista interna signalen.

Tabell 1: Processorns instruktionsuppsättning

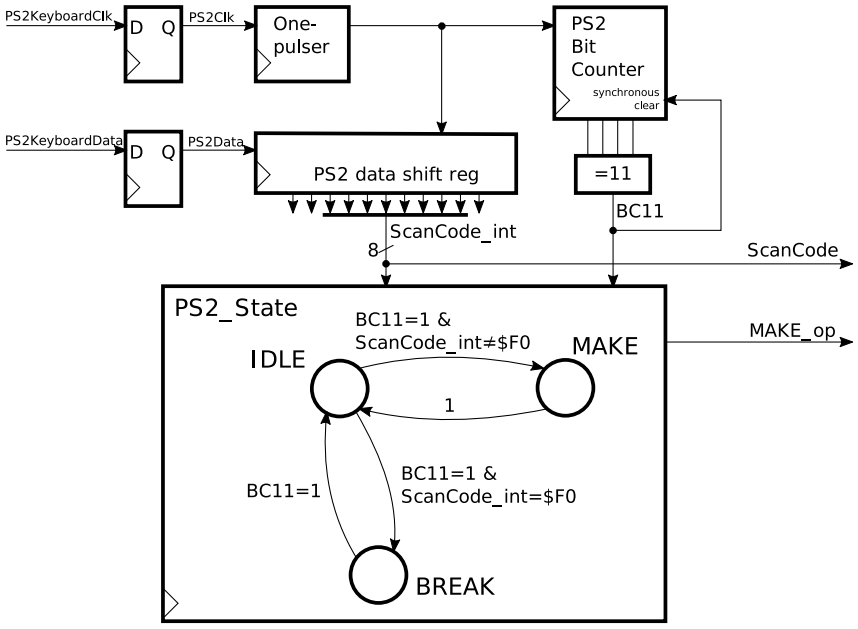
OP	**Instruktion**	**Betydelse**
00000	LD	GRx := ARG
00001	ST	PM(ADDR) := GRx
00010	ADD	GRx += ARG
00011	SUB	GRx -= ARG
00100	CMP	GRx - ARG (statusflaggor sätts)
00101	AND	GRx &= ARG
00110	OR	GRx |= ARG
00111	LSR	GRx »= ARG
01000	LSL	GRx «= ARG
01001	JSR	PUSH(PC), PC := ARG (subrutin)
01010	BRA	PC := ARG
01011	BNE	PC := ARG if Z = 1
01100	BEQ	PC := ARG if Z = 0
01101	PUSH	PM(SP) := GRx, SP–
01110	POP	GRx := PM(SP), SP++
01111	MUL	GRx *= ARG
10000	RET	POP(PC), SP++ (returnera från subrutin)
11111	HALT	Avbryt exekvering, PC := PC

4.3 Tangentbordsavkodare

Tangentbordsavkodaren är till för att avkoda PS/2-data från det inkopplade tangentbordet. Modulen är baserad på Alchasovs och Åkessons lösning på laboration 4 (vga_lab) i denna kurs.

4.3.1 Logik

Hur PS/2-data ser ut förklaras i avsnitt 3.7. Tangentbordsavkodaren är implementerad som en tillståndsmaskin som triggar på PS/2-klocksignalen och skiftar in inkommande bitar i ett skiftregister. Detta visualiseras i figur 10.



Figur 10: Logik för tangentbordsavkodaren

När en komplett dataöverföring har skett, dvs. när 11 bitar har skiftats in, och den resulterande datan inte är lika med `$F0` (break-kod) blir signalen `MAKE_op` hög i klockcykel för att indikera att en tangent på tangentbordet har tryckts ned.

Signalen `ScanCode` uppdateras kontinuerligt, allt eftersom att data skiftas in i skiftregistret, vilket betyder att den inte alltid kommer att vara giltig.

4.3.2 Kommunikation med CPU

CPU:n tar emot signalerna `MAKE_op` och `ScanCode` från tangentbordsavkodaren. När `MAKE_op` är hög, är `ScanCode` en giltig scankod som CPU:n kan använda för att avgöra vilken tangent som har tryckts ned.

Inkomna scankoder från tangentsbordsavkodaren avkodas ytterligare ett steg i CPU:n till mer abstrakta värden. Det nya värdet laddas in i `GR15`, och spellogiken hämtar sina kommandon från detta register. Relationen mellan tangent och spelkommando syns i tabell 2.

Tabell 2: Scankoder och deras motsvarande kommandon

Tangent	Scankod	Spelkommando
`A`	`$1C`	Flytta markör vänster
`D`	`$23`	Flytta markör höger
`W`	`$1D`	Flytta markör uppåt
`S`	`$1B`	Flytta markör nedåt
`Enter`	`$5A`	Avsluta köp
`Space`	`$76`	Plocka upp / sätt ut apa

4.3.3 Begränsningar

Denna tangentsbordsavkodare kan endast hantera tangenter som har en scankod på 8 bitar. Piltangenterna har en scankod på 16 bitar, och därför används inte dessa i spelet.

4.4 VGA-motor

Syftet med VGA-motorn är att jobba parallellt med CPU:n och ansvara för att skicka ut bild till skärmen. Den gör detta genom att läsa CPU:ns videominne och skicka ut en VGA 640x480 standard bild. VGA-motorn är baserad på Alchasovs och Åkessons lösning till laboration 4 (vga_lab) i denna kurs.

4.4.1 Implementation

Utöver den "globala" 100 MHz-klockan har VGA-motorn också en klocka som triggas var 4:e klockcykel, vilket resulterar i en 25 MHz-klocka. Detta görs för att följa standarden för att få en 640x480 skärm i 60Hz. VGA beskrivs i avsnitt 3.5.

Grafiken i spelet är tilebaserad, med 13x10 tiles. Varje tile består av 12x12 "makropixlar" och varje makropixel är 4x4 pixlar. Detta gör att varje tile är 48x48 pixlar stor. En liten marginal på 16 pixlar finns på höger sida av skärmen, eftersom 48*13 = 624, vilket är 16 pixlar kortare än 640.

Utseendet för varje tile-typ sparas i "tileROM"-et, som är en LUT med element på 5 bitar, där varje element står för en viss makropixelfärg i tiletypen. Dessa 5 bitar används för att indexera en annan LUT, "paletteROM", som innehåller färginformation. Denna färginformation skickas sedan vidare till VGA-porten.

Eftersom VGA-motorn måste svepa över alla pixlar i skärmen, finns två räknare `x_subpixel` och `y_subpixel` som håller reda på vilken pixel som ska ritas ut. Dessa räknare räknar upp varje klockcykel, och klockcykeln efter de har nått 799 respektive 520, nollställs de och en ny skärm påbörjas.

Ytterligare signaler finns för att hålla reda på var inom en tile som VGA-motorn befinner sig, och dessa används för att indexera tileROM:et. Det finns `x_within_tile` och `y_within_tile` som håller reda på vilken makropixel som ska ritas ut, och `current_tiletype` som håller reda på vilken tiletyp som ska ritas ut. Dessa kombineras enligt formel 1 för att hitta addressen för att söka i tileROM:et.

$$\texttt{tileROM\_address} = \texttt{x\_macro\_within\_tile} + 12 \cdot \texttt{y\_macro\_within\_tile} + 12^2 \cdot \texttt{current\_tiletype} \quad (1)$$

Hsync och Vsync implementerades helt och hållet enligt standarden som sträck utanför skärmen. Samtidigt används en blank signal som var låg när faktigaste skärmen ritades upp och låg utanför skärmen.

För att hålla koll på vilken rad och kolumn den tile som ska ritas befinner sig, används räknare för rad och kolumn. Dessa räknare räknar upp till 12 respektive 9, på grund av noll-indexering, och räknas upp när `x_subpixel`- respektive `y_subpixel`-signalerna övergår till en ny tile (dvs. =47). Dessa signaler används för att specificera vilken av de 13x10 tiles som ska ritas, och används för att indexera i videominnet. Formel 2 används för att hitta adressen i videominnet.

$$\texttt{vmem\_address} = \texttt{col} + 13 \cdot \texttt{row} \quad (2)$$

4.4.2 Minnen

VGA motorn innehåller två minnen: `tileROM` och `paletteROM`. `tileROM` innehåller utseendet för alla tiles i spelet, och är en LUT med element på 5 bitar. Varje element står för en viss makropixelfärg i tiletypen. Dessa 5 bitar används för att indexera `paletteROM`, som innehåller färginformation. Denna färginformation skickas sedan vidare till VGA-porten för att rita ut pixlarna.

I spelet finns det 67 olika tiletyper, därav innehåller `tileROM` 67*12*12*5 bitar. `paletteROM` innehåller 19 färger, och är en LUT med element på 24 bitar. Dessa 24 bitar står för en färg i RGB-format.

5 Slutsatser

Sista avsnitten spenderas med att reflektera över projektgången, och tar en lite mer informell ton. Frågor som tas upp är: Vad gick bra? Vilka problem förekom? Vad kunde göras bättre? Vad har vi lärt oss?

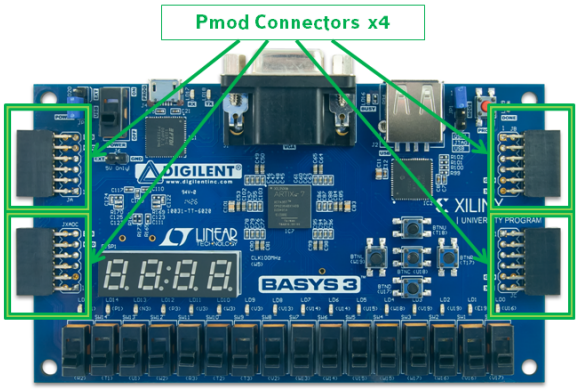
5.1 Sammanfattning

Projektet var en framgång, och vi lyckades skapa ett fungerande (måhända simpelt) spel. Vi lyckades implementera en mikroprogrammerad processor, en tangentbordsavkodare och en VGA-motor.

5.2 Problem och förbättringar

Om vi hade haft mer tid hade vi velat lägga till fler funktioner, en av planerna var till exempel simpla ljudeffekter. Detta hade kunnat göras genom att koppla in en liten piezo-högtalare via en av de fyra Pmod-portarna som syns i figur 11.

Vi märkte att VGA-motorn inte fungerade som den skulle när vi väl började syntetisera, och det berodde på flera olika fel. Vi kunde möjligen ha undvikit dessa fel om vi hade syntetiserat tidigare och itererat på koden oftare.



Figur 11: Basys3 Pmod portar

Själva spelet skrevs i en ganska snabb takt, och det ledde till att assembly-koden blev ganska rörig. Om vi hade haft mer tid hade vi velat strukturera om koden och göra den mer läsbar.

5.3 Lärdomar

Vi spenderade omkring den första tredjedelen av projekttiden helt i simulering, utan att syntetisera koden. Detta ledde till att vi inte testade vår kod på riktigt förrän vi hade en ganska stor kodbas. Det var en stor risk, då det kunde ha visat sig att vår kod inte gick att syntetisera, och vi hade då behövt skriva om en stor del av koden. Som tur var krävdes det endast små ändringar för att få koden att syntetisera.

6 Referenser

[1] Diligent. *Basys 3 Reference Manual*. Accessed: 23 May 2024. URL: https://digilent.com/reference/programmable-logic/basys-3/reference-manual.

[2] The Institute of Electrical och Inc. Electronics Engineers. *IEEE Standard VHDL Language Reference Manual*. Accessed: 22 May 2024. URL: https://edg.uchicago.edu/~tang/VHDLref.pdf.

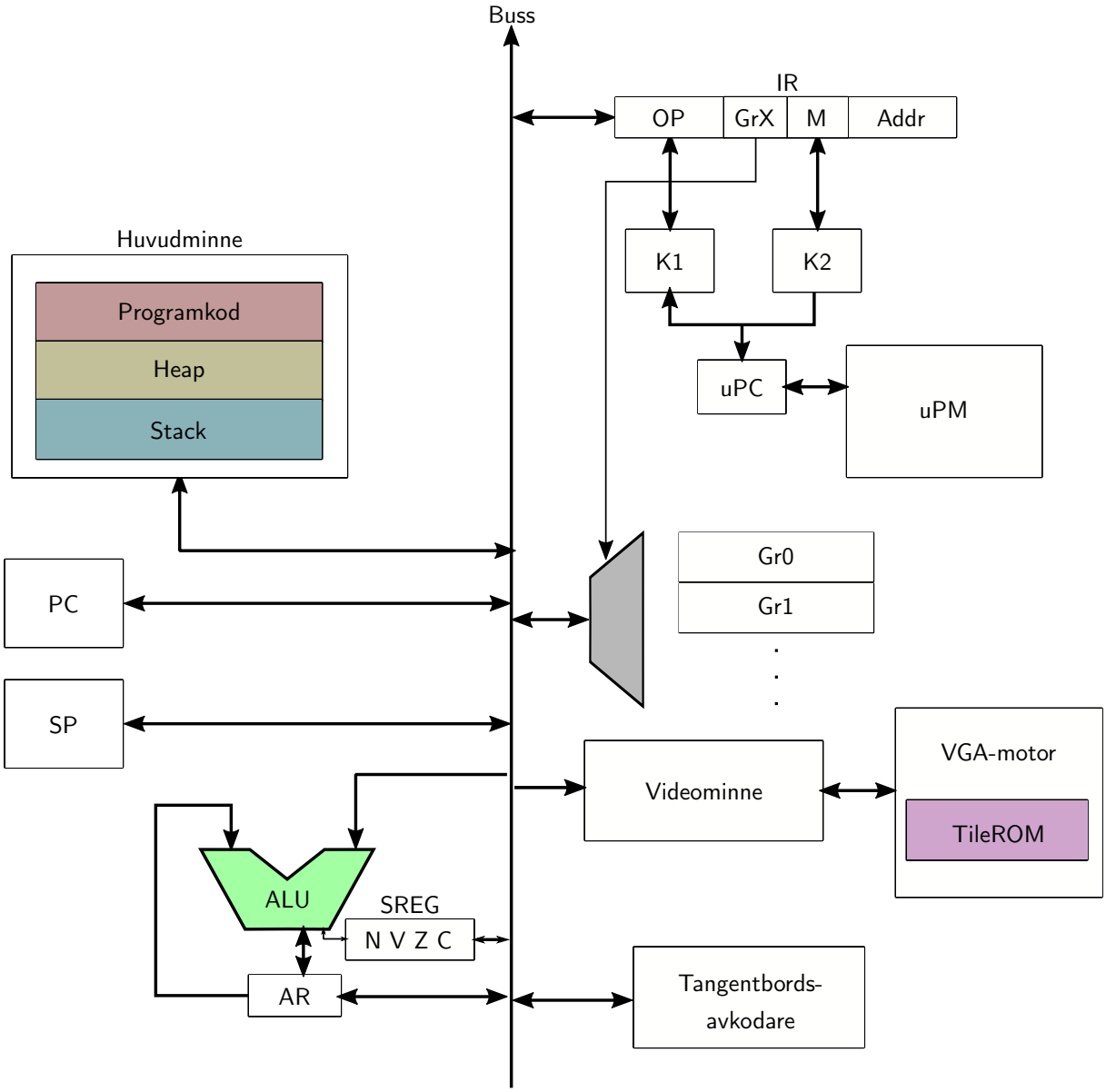
[3] Computer Hope. *VGA*. Accessed: 23 May 2024. URL: https://www.computerhope.com/jargon/v/vga.htm.

[4] Anders Nilsson. *Mikroprogrammering 1*. Accessed: 22 May 2024. URL: https://liuonline.sharepoint.com/sites/Lisam_TSEA83_2024VT_TF/SitePages/F%C3%B6rel%C3%A4sningar.aspx.

[5] Anders Nilsson. *VHDL 1*. Accessed: 23 May 2024. URL: https://liuonline.sharepoint.com/sites/Lisam_TSEA83_2024VT_TF/CourseDocuments/Forms/AllItems.aspx?id=%2Fsites%2FLisam%5FTSEA83%5F2024VT%5FTF%2FCourseDocuments%2FF%C3%B6rel%C3%A4sningar%2FFo8%5FTSEA83%2Epdf&parent=%2Fsites%2FLisam%5FTSEA83%5F2024VT%5FTF%2FCourseDocuments%2FF%C3%B6rel%C3%A4sningar.

7 Appendix. VHDL-kod

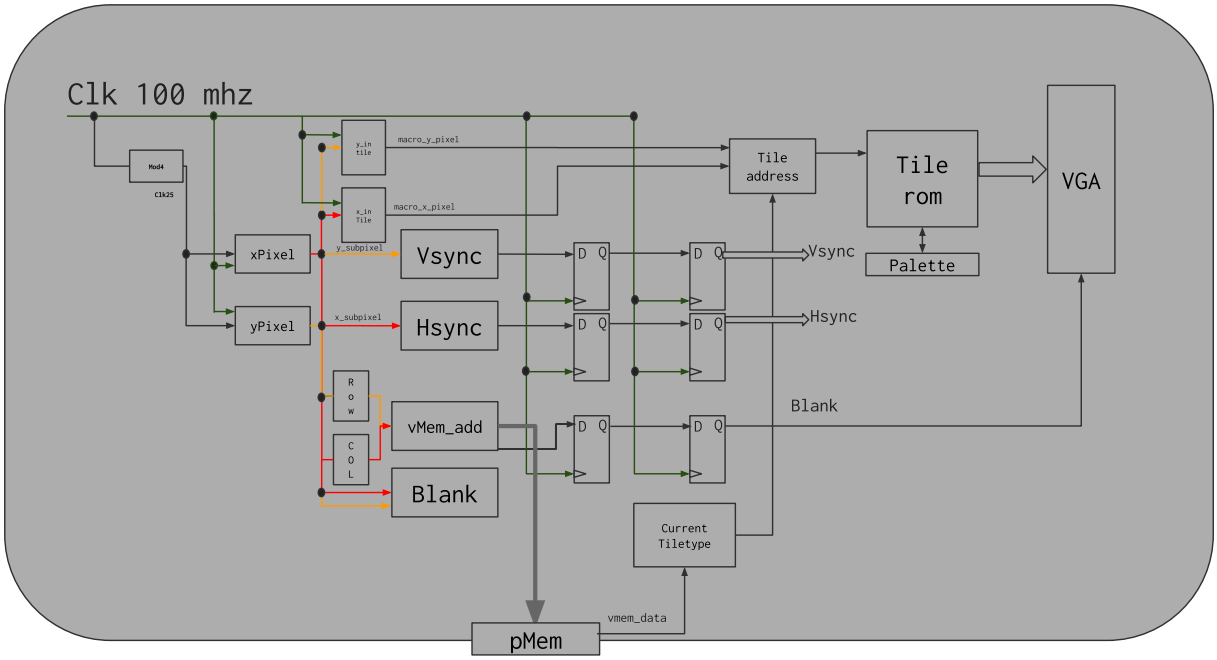
8 Appendix. Övriga listor / bilagor



Figur 12: Översiktligt bild över processorn

```
SIGNAL x_within_tile : unsigned(5 DOWNTO 0); -- value 0-47px
SIGNAL y_within_tile : unsigned(5 DOWNTO 0); -- value 0-47px
ALIAS x_macro_within_tile IS x_within_tile(5 DOWNTO 2); -- divided by 4, value 0-11mpx
ALIAS y_macro_within_tile IS y_within_tile(5 DOWNTO 2); -- divided by 4, value 0-11mpx
```

Figur 13: Utseende för koden av hur makropixlar såg ut



Figur 14: Översiktligt bild över VGA motorn