TSEA83 - Projektrapport

Hugo Nilsson, Edvard Wetind, Mikael Lundgren, Christoffer Näs
Grupp 48

2023-05-14

Linköpings Universitet

Contents

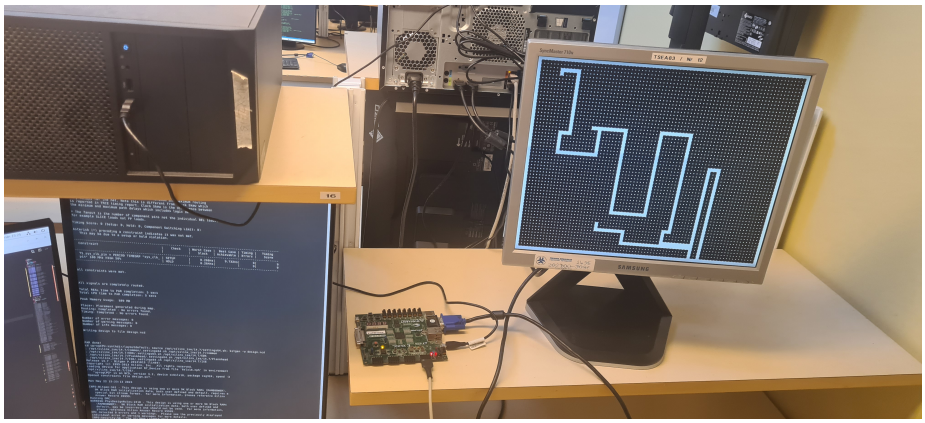
1 Inledning

Studenter som läser Civilingenjörsprogrammet i Datateknik läser under sin studieperiod en rad olika datalogiska kurser. Datorkonstruktion, TSEA83, är en av de mest avancerade kurserna som går inom området och läses av studenter i andra året, under andra terminen. Kursen går över en hel termin, där första perioden ägnas åt projektet förberedande laborationer. Den andra perioden ägnas åt ett projektarbete, som utförs i grupper om upp till 4 personer. Syftet med projektet var att applicera de kunskaper som lärdes ut under första perioden för att bygga en dator från grunden upp och fördjupa förståelsen för hur en dator fungerar och hur den fungerar. Centralt för projektet har varit att på ett effektivt sätt hantera det relativt begränsade minne som Nexys3 besitter. Det slutgiltiga målet var att sedan implementera valfritt spel på den egenbyggda arkitekturen. I detta projekt valdes det klassiska spelet "Snake".

Snake kommer ursprungligen från spelet *Blockade*, ett två-spelare spel från 1976 med liknande dynamik. I *Blockade* var målet istället att undvika den andra spelare och därmed överleva längst. Spelet Snake som det ofta känns till av allmänheten tog sin första form 1978, då det gavs ut till datorn TRS-80 under namnet *Worm*. Snake i modern form kom först 1997 då telefonbolaget Nokia lanserade sin telefon Nokia 6610, på vilken Snake medföljde.[1] Spelet går ut på att spelaren kontrollerar en rektangulär figur - liknande en orm - vars mål är att äta "äpplen" som slumpmässigt placeras ut på spelplanen. För varje äpple som ormen äter blir den längre, och spelet blir svårare. Spelet är över när spelaren kolliderar med någon av de fyra väggarna eller sig själv. Ofta räknas varje äpple som ormen förtär som ett poäng. Implementationen av Snake i detta projekt är av snarlik karaktär, med undantaget att poäng inte räknas.

2 Apparaten

Den fysiska delen av konstruktionen, hårdvaran, består av ett Nexys3 FPGA-kort, en joystick, USB-A till micro-USB, en VGA-kabel och en VGA-kompitabel skärm. Skärmen har en upplösning på 640 x 480 pixlar. Datorn förbreds för användning genom att sammankoppla joysticken och Nexys3-kortet. För att datorn ska kunna avläsa insignalerna från joysticken är det viktigt att den sitter kopplad i porten märkt *JA*. Skärmen skall kopplas med VGA-sladd till FPGA-kortet via VGA-porten som sitter på kortets vänstra sida, bredvid Ethernet-porten. För att förse kortet med ström samt möjliggöra programmering måste kortet vara kopplat till en annan dator med syntetiseringsmjukvara. Detta projekt har använt sig av ModelSim för att simulera. Se *figur 1* nedan för referens över hur en korrekt konfiguration ser ut.



Figur 1. Bild över konstruktionen.

Nexys3 kortet blir aktivt omedelbart vid spänningsmatning. Är kortet programmerat är det nu möjligt att börja använda. Spelet skall automatiskt visas på skärmen. Om det av användaren önskas går det att enkelt omprogrammera efter ändringar i koden genom att följa nedan steg:

1. Öppna programkoden i ModelSim genom att köra kommandot *make lab.sim* i terminalen
2. Skapa bit-fil genom att köra kommandot *make lab.bitgen*
3. Programmera kortet genom att i terminale skriva kommandoraden *make lab.prog*

Ormen kan styras av användaren genom att trycka joystick-handtaget i önskad riktning. Det är möjligt att styra ormen, dock finns ingen kod för att ta bort ormens svans. Målet är att konsumera så många äpplen som möjligt utan att kollidera med sig själv eller med någon av väggarna. För att äta äpplen måste spelaren manövrera ormen så att dess huvud träffar äpplet. Mjukvara för äpplen avsaknas helt i skrivande stund.

3 Teori

Detta avsnitt är ägnat till att beskriva och förklara olika standarder eller teknologier som användes under projektet för att möjliggöra full implementation.

3.1 FPGA

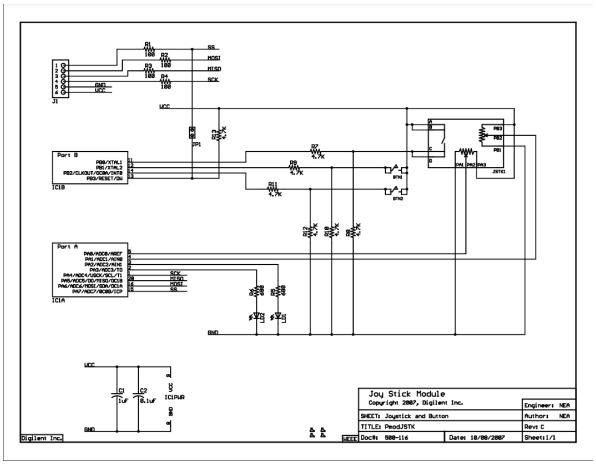
Field Programmable Gate Array är en typ av kretskort som består av configurable logic blocks (CLB). Dessa CLB sitter ihop med programmerbara sammankopplingar. Genom att manipulera dessa länkar med kod kan kortet programmeras till att utföra en särskild uppgift. Det är även möjligt att efter användning till ett särskilt syfte att omprogrammera FPGA:n till ett annat ändamål. [4]

FPGA-kort används inom en stor mängd olika branscher. Några konkreta exempel på industrier där FPGA-kortet är oumbärligt inom luftfart, medicin eller bil-tillverkning.

3.2 PMOD

Peripheral Modelule Interface, eller PMOD, är en rad elektroniska enheter som går att ansluta till ett FPGA-kort eller en mikro-kontroller för att utöka dess funktionalitet. PMOD är ett varumärke som utvecklats av Digilent, vilka också utvecklat Nexys3-kortet som är centralt för det här projektet. [3]

Digilent Pmod Joystick är av särskilt intresse, då det är det primära sättet som användaren kommunicerar med datorn. Denna kommunikation sker över en SPI-kabel som sitter mellan PMOD och FPGA-kortet. *Se avsnitt 3.3.* Joysticken skickar via denna kabel en bit i taget till datorn. *Se figur 1.*



Figur 1. Diagram över Pmod Joystick. *(Digilent, 2023.)*

Modulen skickar utdata på MISO (Master In, Slave out) och tar emot indata på MOSI (Master out, Slave in). SS (Slave Select) måste vara aktivt låg under hela kommunikationsperioden.

3.3 SPI

Serial Peripheral Interface (SPI) är ett kommunikationsprotokoll som använder sig av seriell överföring av bitar mellan två enheter. Enheter som använder SPI sägs vara i ett Master-Slave förhållande, där den kontrollerande enheten är master. Fördelen med SPI-kommunikation är att där inte förekommer stop/start-bitar, vilket innebär kontinuerlig kommunikation utan avbrott. [5]

3.4 Mjukvara

I K1 finns 26 operationer tillgängliga, med hjälp utav dessa kan man koda snake i programminnet. Dem flesta är lika assembler instruktioner, liksom load, store, add osv. Dock finns det några som är skapade för just snake. För hela listan av operationer se tabell i appendix

1. LDRAN: För att hitta en slumpmässigt position på spelbrädet används funktionen LDRAN. Den hämtar ett slumpmässigt värde från RANDOM komponenten och sparar detta i hr för att senare kunna placera ut ett nytt äpple.

2. Bortagning av svansen: I PICT_MEM sparas rotationen på bit av ormen. Varje tick när biten längst bak ska ta bort vandrar en pekare över hela ormen för att sedan ta bort sista biten. Detta i kombination med att lägga till en ny bit längst fram varje tick ger en illusion utav att ormen rör sig.

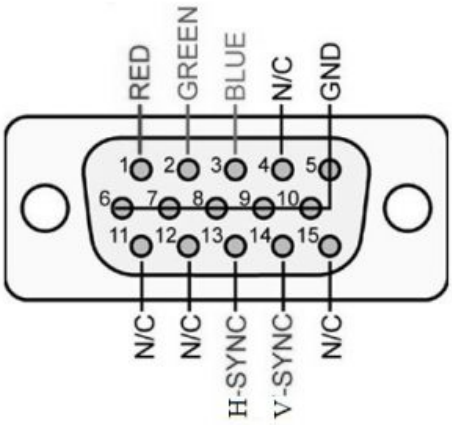
3. VHR: VHR_STORE och VHR_LOAD är de två funktioner som används för att komminucera med PICT_MEM. Där VHR_LOAD hämtar ett specifikt värde från PICT_MEM. VHR_STORE sparar värdet i HR till en specifik position i PICT_MEM. Dessa två funktioner gör det då möjligt för att ta bort svansen samt lägga till ett huvud varje tick.

3.5 VGA

Video Graphics Array, eller VGA, är en teknik för att uppvisa färg-grafik på en skärm med upplösning upp till 640 x 480. VGA introducerades på marknaden av IBM 1987. [6]

VGA använder en kontakt med 15-pin, där varje pin har en egen funktion. *Se figur 2.*

Största skillnaden mellan VGA och moderna gränssnitt är att VGA använder sig av analoga signaler.



Figur 2. Karta över VGA-kontakten. *(Elprocus, 2023.)*

4 Hårdvara

Denna del kommer att behandla Datorns hårdvara. Hårdvaran är kodad i VHDL och programmerad på ett Nexys 3 kort.

Logic Cells:	2278 Slices
Block RAM:	576 Kbit
Cellular RAM:	16 MB

Tabell 1. Nexys3 Specifikationer

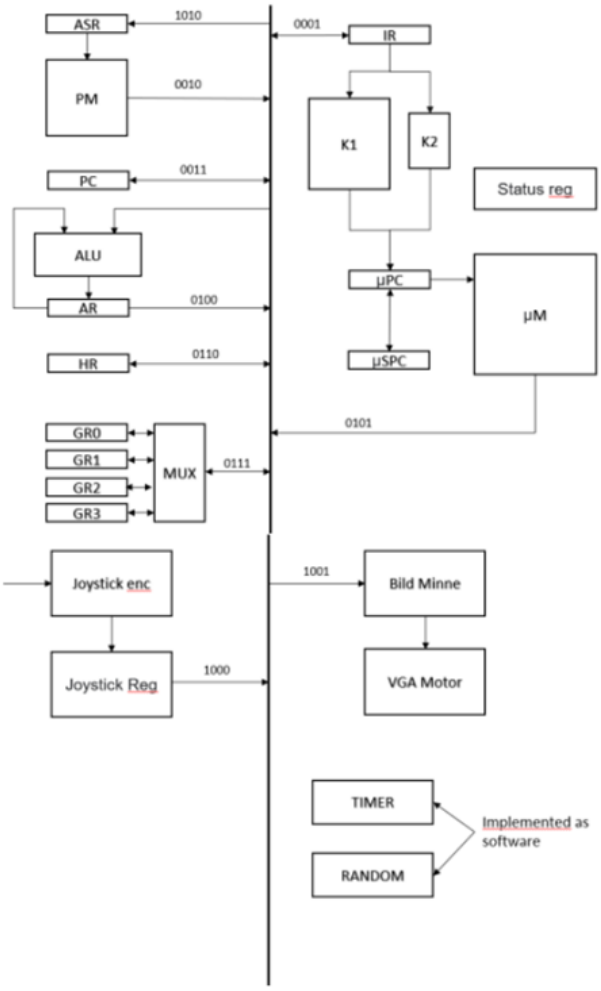
4.1 CPU. In och Utgångar

CPUn, *uprogCPU* tar in en *clk*, en *rst* signal som används för att kunna göra en reset av datorn samt signalen *MISO* som hör ihop med joysticken. Som utgångar har den *SS*, *MOSI* och *SCLK* som hör till joysticken, samt *vgaRed*, *vgaBlue*,*vgaGreen*, *Hsync* och *Vsync* som går till VGA skärmen och realiserar bilden på skärmen. Dessutom finns utgången *LED* som talar om när LED panelen på nexys 3 kortet ska lysa. Se *figur 3*.

Figur 3. Blockschema cpu

4.2 CPU. Detaljerad beskrivning och Databus

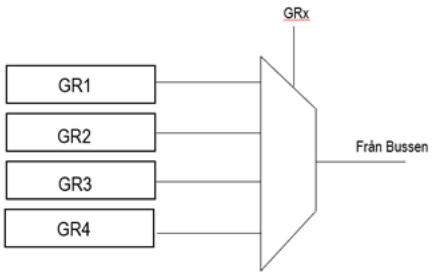
Datorn innehåller 5 minnen, Programminnet, Microminnet, Bildminnet, K1 minnet, K2 minnet. CPU:n är länken mellan alla minnen som får datorn att köra, Se *figur 4. CPU:n läser av Programminnet och köra operationer således. All information skickas via databussen som bestäms utav mikrominnets TB (tillbuss) och FB(frånbuss).* Databuseen är 21 bitar bred vilket tillåter 5 bitar för Operationen, 2 bitar för addressmodul, 2 bitar för vilket generellt register och 12 bitar för address till Programminnet.



Figur 4. Blockschema databus

4.3 GRx Muxen

Till bussen är en mux kopplad med 4 tillhörande register GR1, GR2, GR3 och GR 4. Beroende på 2 bitars signalen *GRx* kommer olika bitar kunna väljas. *GRx* bitarna fås av GR bitarna ur nuvarande intruktion i IR registret. Se *figur 5*.



Figur 5. Blockschema Grx Mux

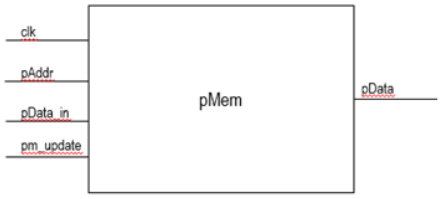
4.4 K1,K2 Minnen

K1 minnet, *K1* tar in OP bitarna ifrån den nuvarande instruktionen i IR register, *Operand* och skickar ut rätt adress i *K1_adress* för rätt mikrointruktion till uPC. K2 minnet, *K2* tar in M bitarna ifrån nuvarande instruktion i IR registret, *Modd* och skickar vidare det till uPC. Se *figur 6*.

Figur 6. Blockschema K1 och K2 Minnen

4.5 Program Memory

Programminnet, *pMem* innehåller alla program instruktioner. Minnet är klockat på klockans signal *clk*. Då signalen *pm_update* är låg skickas bitarna på plats *pAddr* i minnet ut på *pData*. Då textitpm_ update är hög laddas bitarna i *pData_in* in i minnet på plats *pAddr*. Se *figur 7*.



Figur 7. Blockshema Programminne

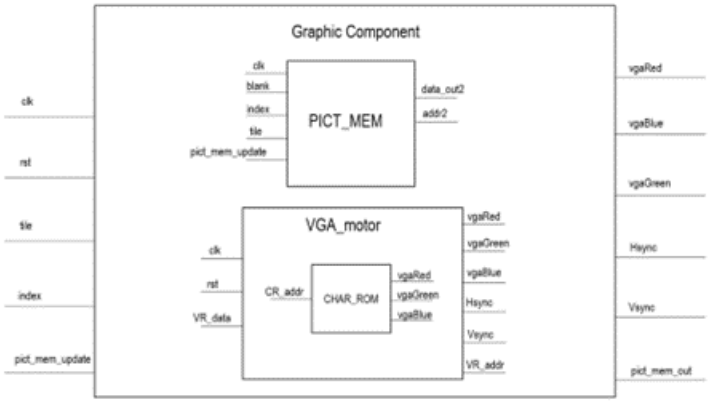
4.6 Mikrominne

Mikrominnet, *uMem* innehåller alla microinstruktioner. Insignalen *uAddr* representerar vilken position i minnet som microinstruktionen ligger på. I signalen *uData* skickas rätt mikrokod ut. *Se figur 8*.

Figur 8. Blockshema Mikrominne

4.7 Graphic Component

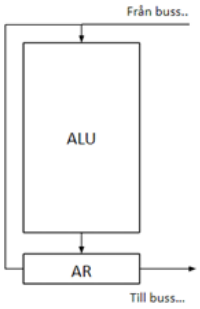
Graphic Component, *graphic_comp* hanterar all grafik och jobbar emot CPUn. Det är den enda grafiska komponenten som kommunicerar med processorn. Den innehåller ett bildminne, *PICT_mem* samt en VGA motor, *VGA_motor*. Bildminnet innehåller den information som ska visas på skärmen, en plats i minnet är en tile på skärmen. Bildminnet är 60x80 stort och därmed är spelplanen 60x80 tiles stor. VGA motorn realiserar vad som finns i bildminnet till signaler som kan visas på skärmen via en VGA anslutning. VGA motorn innehåller ett minne, char_rom som berättar vad varje tile ska realiseras till. VGA motorn skickar vilken address den ska realisera via *addr2_sig* till bildminnet som svarar med vilken typ av tile via *data_out2_sig*. Se *figur 9*.



Figur 9. Blockshema Graphic Component

4.8 ALU

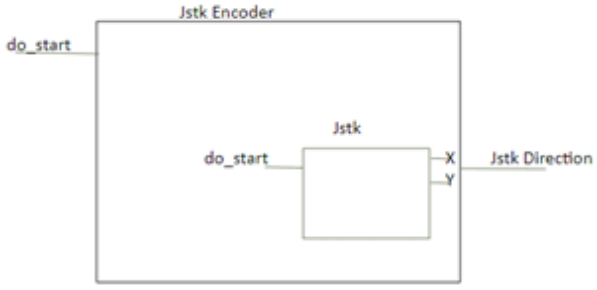
ALUn utför en operation emellan värdet i AR registret och värdet på bussen. Resultatet av uträkningen sparas sedan i AR registret. Medhälp av ALUn kan mikrooperationer i tabel 1 utföras. Se *figur 11.* Utifrån dessa operationer sätts även flaggorna O,Z,N,C. Med hjälp av dessa flaggor kan man veta vad resultatet för ALU operationen blev. Se *tabell 3* för flaggorna



Figur 10. Blockshema ALU

4.9 Joystick avkodning

Joystickavkodaren, Jstk Encoder, tar in en signal *do_start* som är klockad på spelets frekvens. Då do_start är hög börjar man hämta Joysticks signal. En kodbasis som tolkar Joystickens alla in och utsignaler ger ut två utsignaler X och Y som Jstk Encoder sedan avläser och dumpar ner till en fyra bitar lång signal som innehåller ormens riktning, *Jstk Direction.* De fyra bitarna representerar vänster, höger, upp och ned. Se *figur 11.*



Figur 11. Blockshema Joystick Encoder

5 Slutsatser

Vid flertalet tillfällen uppstod problem med hårdvara där en liten kod ändring gjorde att tidigare fungerande funktionaliteter helt försvann. Ett exempel på detta var när det utfördes en hel del ändringar till koden som till synes inte borde ha ställt till det. Då slutade både Joystick avkodningen och VGA motorn att fungera. Detta problem löstes ändas genom att gå tillbaka till äldre VHDL kod och skriva om ändringarna. Ett annat problem var på var att fler operationer än vad som tänks gjordes så de fyra bitar för OP-fältet räckte då inte längre till och behövdes utvidgas till 5 bitar. Det hade varit bra om det fanns en tydligare idé med datorn innan koden började skrivits. Intruktionsbredden var tvungen att ändras till 21 bitar i stället för 20 bitar. Om samtliga operationer hade varit bestämda hade tid sparats. Programinstruktionerna hårdkodades i början för hand vilket både tog lång tid och ökade risken för felskrivning. Fördelaktigt hade en kompilator för programkod skapats för att underlätta arbetet. Projektet har varit väldigt givande för alla medtagande och många lärdomar har dragits. En lärdom som dragits är att en bra planering av vad som ska skapas och hur arbetet ska utföras kan spara väldigt mycket arbete. En annan lärdom som dragits är att om man kör helt fast så kan det vara nödvändigt att gå tillbaka ett steg och börja om, hitta en annan lösning på problemet.

6 Referenser

[1] Angelos, A February 23rd, 2022. *The History of Snake: How the nokia game defined a new era for the mobile game industry* Accessed: May 18th, 2023. https://www.itsnicethat.com/features/taneli-armanto-the-history-of-snake-design-legacies-230221

[2] Pena, E., & Legaspi, M G. December 2020, UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter *Volume 54* Accessed: May 19th, 2023. https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

[3] PMOD Connector Boards. Digilent. Accessed: May 21th, 2023. https://digilent.com/shop/boards-and-components/system-board-expansion-modules/pmods/

[4] Field Programmable Gate Array (FPGA). Xilinx. Accessed: May 21th, 2023. https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[5] SPI. Digilent. Accessed: 21th May, 2023. https://digilent.com/reference/learn/fundamentals/communication-protocols/spi/start

[6] Britannica, T. Editors of Encyclopaedia. (2021, August 16). VGA. Accessed: May 22nd, 2023. Encyclopedia Britannica. https://www.britannica.com/technology/VGA

7 Appendix.

ALU Operation	Kodning
NOP	
BUSS	AR := BUSS
Nollställ AR	AR := 0
ADD	AR := AR + GRx
SUB	AR := AR - Grx
Bitwise or	
Logical shift right	
Logical shift left	

Tabell 2. ALU Operations

Flagga = 1	Betydelse
Z	Ifall operationen = 0
O	Ifall operationen orsakar spill
N	Då mest signifikanta biten = 1
L	Då LC = 0
C	Minnessiffra vid addition/subtraktion

Tabell 3. Flaggor.

Operation	
HALT	Stannar programmet
LOAD	GRx := PM(A)
STORE	PM(A) := GRx
ADD	GRx += PM(A)
AND	GRx %% PM(A)
SUB	GRx -= PM(A)
LSR	Logic Shift Right
BRA	Branch to PM(A)
BNE	BRA if not equal
BSE	BRA if Smaller or Equal
BGE	BRA if Greater or Equal
BEQ	BRA if EQual
BPL	BRA if two complement negative
LDJ	Load Jstk to AR
CMP	GRx == PM(A)
CMPA	AR == PM(A)
STRLC	LC := PM(A)
JMP	PC := PM(A)
LDRAN	HR := Random
STRHR	PM(A) := HR
LDHR	HR := PM(A)
VHR_STR	PICT_MEM(A) := HR
VHR_LOAD	HR := PICT_MEM(A)'
ADDHR	HR += PM(A)
SUBHR	HR -= PM(A)
LC–	LC–
BRL	BRA if LC == 0

Tabell 4. Mikrokod operationer

List of Figures

List of Tables