
30010 - Programmeringsprojekt

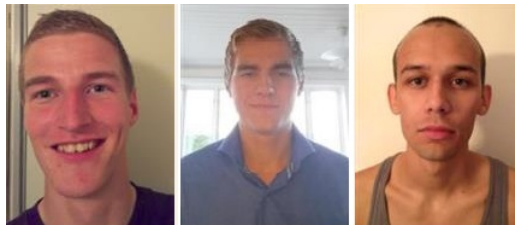
Reflexball

Gruppe 3

Martin Boye Brunsgaard, s144012(1)

Tore Gederaas Kanstad, s144021(2)

Peter Asbjørn Leer Bysted, s144045(3)



1

2

3

Alle medlemmer har været tilstede under øvelserne, og deltaget i udarbejdelse af journalerne. Ydermere har arbejdet været fordelt ligeligt over gruppemedlemmerne, og løst i fællesskab. Rapporten er blevet udarbejdet og gennemlæst i kollektiv.

Tecnical University of Denmark DTU
National Space Institute
30010 - Programming Project
24.06.2015

Abstract

This report covers the reflexbal game, which is a mandatory part of the B.Sc. EE course 30010 Programming Project.

The report documents the entire course of the exercises in which the digital logic behind a simple vending machine was designed using VHDL. The project was split into three sub-assignments: The first assignment was to drive a seven-segment display with hexadecimal numbers, the second was to display two different 2-digit decimal numbers simultaneously and the last was to implement a CPU using a data path controlled by a FSM. The 3 assignments were combined into one circuit and implemented on a Basys2 Spartan FPGA board.

Contents

1	Introduktion	3
2	Teori	3
2.1	Binære tal	3
2.2	Unsigned repræsentation	3
2.3	Fixed point kommatall	4
2.4	Repræsentation af negative tal	4
2.4.1	Signed magnitude	4
2.4.2	2's komplement	5
2.5	Fixed point vs floating	5
3	Design af Reflexball	5
3.1	Tekniske mål	5
3.2	Krav til spillet	6
3.2.1	Overordnede krav til spillet	6
3.2.2	Krav til strikeren	7
3.2.3	Krav til bolden	7
3.2.4	Krav til boksene	7
3.3	Timere	8
3.3.1	Timer0	8
3.3.2	Timer1	8
4	Brugervejledning til ReflexBall	8
5	Plan	8
5.1	Problemer	8
5.1.1	Problemer med realloc	9
6	Dokumentation	9
6.1	Application layer	9
6.1.1	reflexball.h	9
6.1.2	menu	11
6.2	Application Interface Layer	12
6.2.1	graphics.h	12
6.2.2	lut.h	13
6.2.3	math.h	13
6.3	Hardware Abstraction Layer	14
6.3.1	keys.h	14

6.3.2	timer.h	14
-------	-------------------	----

1 Introduktion

Målet med dette projekt er at designe og implementere et program. Programmet skal skrives i C og det skal implementeres på en Zilog Z8 encore microprocessor vha. ZDS II - Z8Encore! 4.9.3 værktøjer. Programmet skal dokumenteres vha. flowcharts, grafer og beskrivelser af de enkelte funktioner.

Programmet skal være et spil, Reflexball. Spilleren styrer en striker, som skal bruges til at reflektere en bold, således den kan bevæge sig rundt på banen. Hvis bolden rammer en af kanterne, skal bolden ligeledes også reflekteres. Hvis spilleren ikke rammer bolden ryger bolden ud af banen, og spilleren fratrækkes et liv. Såfremt spilleren ikke har flere liv tilbage, afsluttes spillet. Desuden indføres der nogle bokse i spillet, som spilleren skal ødelægge. Når spilleren har ødelagt alle disse bokse går spilleren videre til næste bane, eller vinder såfremt han er på sidste bane. Den grafiske flade bliver implementeret ved at skrive til en terminal. Ydermere får brugeren fremvist informationer fra spillet på LED'erne på boardet.

2 Teori

Vi vil i dette afsnit gennemgå den basale teori bag binære tal og slutteligt indføre læseren i de forskellige formater, deriblandt fixed-point format, og hvorfor det er interessant at bruge denne repræsentation i vores projekt.

2.1 Binære tal

Et binært tal er et tal der kan udtrykkes i det binære talsystem/base-2, hvor grundtallet er 2. Da det er meget let at implementere i digital logik, er det et system der bruges internt i computere verden over.

Et binært tal består af bits, som svarer til et ciffer. Et bit kan have en af to tilstande: logisk højt eller logisk lavt. Dette medfører da hvis vi har n bits har vi 2^n forskellige tilstande. Disse forskellige tilstande kan fortolkes på forskellige måder, og vi vil i de næste afsnit gennemgå nogle af de forskellige representationer.

2.2 Unsigned repræsentation

I det binære talsystem er grundtallet vanligvis 2(det kunne potentielt også være -2). Det betyder således at i en n -bit streng, vil bittet yderst til højre

være vægtet med 2^0 , det næste med 2^1 op til 2^n gående mod venstre. Tallet 5(base-10) kan da skrives som i ligning 1. Ydermere tæller vi også fra højre mod venstre, og første bit står således også på 0 plads. Dette bit kaldes oftest LSB(least significant bit), hvorimod det bit der står helt til venstre oftest kaldes MSB(most significant bit) [2, s. 18].

$$5_{10} = 101_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \quad (1)$$

Med de indførte definitioner har vi kun mulighed for at repræsentere positive heltal. Vi ønsker også at kunne skrive kommatal og negative tal.

2.3 Fixed point kommatal

Kommatal kan indføres på en simpel måde, ved blot at vægte i omvendt retning når man går mod højre, således at bittet til højre for kommaet har vægtningen 2^{-1} , bittet 2 til højre for kommaet vægtningen 2^{-2} osv. Hvis man har en n-bit streng med b tal til højre for kommaet, har man da muligheden for at skrive tal mellem 0 og $\frac{2^n-1}{2^b}$ [1, s. 4]

Tallet 13.625 kan f.eks skrives som

$$13.625_{10} = 1101.101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3} \quad (2)$$

2.4 Repræsentation af negative tal

Hvis vi ønsker at repræsentere negative tal, gøres det oftest på 3 forskellige måder: Signed magnitude, 1's komplement eller 2's komplement. Vi vil her gennemgå signed magnitude og 2's komplement.

2.4.1 Signed magnitude

En af måderne at repræsentere fortegnet på bit-strengen, er ved at lade det mest signifikante bit(MSB: længst til venstre) bestemme fortegnet, hvor 0 indikerer et positivt tal og 1 indikerer et negativt tal. F.eks. kan tallet -37 i signed magnitude repræsentation skrives således:

$$-37_{10} = 1100101_2 \quad (3)$$

Signed magnitude repræsentation, har dog den ulempe, at man spilder et bit, f.eks. hvis man har en 4-bit streng gælder der at 1000 = 0000, så istedet for at have 2^4 tilstande har man blot $2^4 - 1$. Desuden er signed magnitude ikke så velegnet til brug i digitale systemer.[2, s. 260]

2.4.2 2's komplement

En anden måde at repræsentere negative tal, kan gøres vha. 2's komplement. 2's komplement findes ved at invertere et unsigned tal og derefter lægge 1 til. 2's komplement har to fordele: Der er kun et 0, og subtraktion kan gøres på samme måde som addition, så hvis vi ønsker at subtrahere 3 fra 5, skal vi blot finde 2's komplement af 3 og lægge det til 5, som i ligning 4. Disse fordele gør 2's komplement en god metode til at repræsentere tal i digitale systemer.

$$5 - 3 = 5 + (-3) \quad (4)$$

2.5 Fixed point vs floating

I dette projekt bruger vi ikke floating-point typer. Dette skyldes at Z8 serien er en 8-bit processor, og disse bruger oftest ALU's (Arithmetic Logic Unit) designet til fixed point aritmetik.[3, s. 5]. Det er muligt at få processorer med indbyggede FPU (floating-point units), der er optimerede til at arbejde med floating-point typer, eller coprocessorer til at supplere CPU'en. Hvis vi insisterede på at arbejde i floating-point format, ville vi være nødt til at bruge Zilog's bibliotek til floating-point, hvilket ville kræve både større ram forbrug og være for langsomt til vores behov[4]. Vi har derfor kun brugt integers, og omdannede dem til fixed point i vores projekt.

3 Design af Reflexball

I udarbejdelsen af dette program havde vi nogle forskellige tekniske krav og mål, som vi ønskede at designe programmet efter.

3.1 Tekniske mål

Vi lavede en liste af krav til programmets design som vi i så høj grad som muligt ønskede at overholde.

1. Vi ønsker en veldefineret struktur. Vi vil derfor undgå globale variable i så høj grad som muligt, derfor skal vi lave funktioner som tager pegere til strukturer eller variable som inputs, frem for at tilgå globale variable. Få undtagelser findes dog til dette, f.eks. i biblioteket der tilgår timeren.

2. Vi ville udvikle nogle moduler der var uafhængige af hinanden, således at vores grafik i mindst mulig grad kommunikerede med vores modul indeholdende spillets back-end(Refball.c). Denne kommunikation skulle såfremt foregå igennem main-metoden, således man let kan få et overblik ved at se på main-metoden.

3.2 Krav til spillet

3.2.1 Overordnede krav til spillet

1. Spillet er et arkanoid spil, bestående af 3 levels. Banerne skal være i stigende sværhedsgrad. Dette gøres ved at boksene gøres stærkere, således de skal rammes flere gange for at ødelægges, og også tilføje flere kasser.
2. Spilleren har 3 liv til at starte med, og får et liv for hver bane han vinder.
3. Hvis spilleren ikke har flere liv tilbage, afsluttes spillet og der vises game over på skærmen. Efter et par sekunder går spillet automatisk tilbage til menuen.
4. Når banen begynder, eller hvis spilleren mister et liv, placeres bolden over strikeren, og spilleren kan frit bevæge strikeren, hvor bolden følger efter. Hvis spilleren trykker på den givne knap, affyres bolden i en opadgående lodret linje.
5. Spillerens liv og tiden skal skrives på LED-skærmen når spillet er igang
6. Spilleren samler power hver gang han rammer en kasse. Hvis brugeren trykker på venstre og højre-tasten på en gang bruger han sit power og aktiverer hiii power. Når hiii power er aktiveret ødelægges kasser når de rammes, uafhængigt af deres liv, og bolden reflekteres ikke, men fortsætter gennem kassen.
7. Når spilleren bruger hiii power, vinder en bane, vinder spillet eller dør skal der rulles en tekst over LED-skærmene. Alt afhængigt af situationen, skal livene og tiden igen vises på skærmen efter teksten er rullet over.

3.2.2 Krav til strikeren

1. Strikeren skal maksimalt fylde 10% af skærmen på x-aksen.
2. Strikeren skal være delt ind i 5 forskellige områder. Disse 5 områder skal reflektere bolden på forskellig vis afhængig af indgangsvinklen og hvilken del af strikeren den rammer.
3. Brugeren skal kunne styre strikeren, vha. knapperne på boardet.

3.2.3 Krav til bolden

1. Bolden skal et x- og y koordinat og en retningsvektor, begge i 18.14 format. Bolden har desuden nogle variable med info om spillerens power, om bolden er ude og om spilleren har aktiveret power.
2. Boldens retningsvektor skal altid have længden 1, da dette gør kollisionstest let.

3.2.4 Krav til boksene

1. Alle bokse skal have de samme dimensioner, vi valgte 2x6 pixels.
2. Boksene skal kunne have forskellig styrke, således at nogle kasser skal rammes flere gange før de går i stykker. Kassens styrke skal således repræsenteres ved en farve, og farven ændrer sig således også når man rammer en kasse uden at ødelægge den.
3. Hvis man rammer boksen på den horizontale side, skal y-elementet af retningsvektoren inverteres.
4. Hvis man rammer boksen på den vertikale side, skal x-elementet af retningsvektoren inverteres.
5. Hvis man rammer et hjørne, skal både x- og y-elementet inverteres.
6. Når en kasse bliver ødelagt slettes den fra banen

3.3 Timere

På Z8 Encore Evaluation Boardet er der 4 forskellige timere, timer0 til timer3. Disse timere kan konfigureres efter brugerens behov. I vores projekt har vi brugt 2 timere, en til at styre spillets tid, og en anden til at styre LED skærmene, vi har blot brugt timer0 og timer1.

3.3.1 Timer0

Timer0 er en timer der sender et tick hvert millisekund. Timeren bliver brugt i main-funktionen og til debouncing af knapperne. Timeren er sat i continuous mode, da vi ønsker at den blot skal fortsætte ubetinget, og der foretages ingen clock division af tælleren. Reload værdien fandtes ved udregningen i ligning 5. Interrupt Prioriteten sættes til høj ved at skrive 0x20 til både IRQ0ENH og IRQ0ENL.

$$Reloadvalue = 0.001s \cdot 18.432.000s^{-1} = 4800_{16} \quad (5)$$

3.3.2 Timer1

Timer1 er en timer der sender et tick hvert 500 μs . Timeren bliver kun brugt i **led.h**. Denne timer er også sat i continuous mode, og der bliver heller ikke her foretaget clock division. Reload værdien fandtes ved udregningen i ligning 6. Interrupt prioriten sættes til lav ved kun at skrive til IRQ0ENL.

$$Reloadvalue = 0.005s \cdot 18.432.000s^{-1} = 2400_{16} \quad (6)$$

4 Brugervejledning til ReflexBall

5 Plan

5.1 Problemer

Under udarbejdelsen af programmet havde vi problemer, som vi ikke havde forudset under design-fasen, vi vil her gennemgå nogle af dem der voldte os mest besvær, at debugge.

5.1.1 Problemer med realloc

I design-fasen havde vi forestillet os at vores boxstruct blot skulle være skrevet som i **newBoxStack()**, hvor vi dog kun allokerede plads til et enkelt element i hvert array. Ydermere skulle der være en variabel kaldet *capacity*, der betegnede hvor stort stacket var. Vi ville derefter i **createBoxes()** undersøge om `size == capacity`, og hvis det var sandt allokere yderlige 10 pladser med **realloc**. Dette fungerede dog ikke, og boksene fik tilfældige lokationer på banen. Vi mistænker at der ikke var plads til at dynamisk allokere plads på boardet og finde sammenhængende plads i rammene, og det derfor gik galt. Hvis vi i stedet startede med at allokere plads, gav det os ikke problemer.

6 Dokumentation

Vi har udviklet alle moduler i fællesskab, og det giver ikke mening at tilskrive nogle personer en særlig del af koden, da meget lidt kode er skrevet af udelukkende en person.

6.1 Application layer

I dette lag findes **reflexball** modulet, **menu** modulet og **main** modulet.

6.1.1 reflexball.h

Reflexball er et modul der indeholder grundlæggende regler om spillet, deriblandt kollision, hvordan bolden skal bevæge sig og hvorledes strikeren skal opføre sig. Desuden indeholder den strukturerne **Ball** og **Box**. Desuden indeholder dette modul også særligt mange konstanter, som er konstanter der definerer spillets grænser.

Strukturen **Ball**

Ball er en stuktur som har variablene opgivet i 3.2.3. **outOfBounds**, der fortæller om spilleren er inde for banen, og **powerActivated**, der fortæller om hiii power er aktiveret, er implementeret som `unsigned char`'s da de kun kan være 0 eller 1. I variablen, **power**, gemmes hvor meget power spilleren har opladet.

Strukturen Box

Box er en struktur der indeholder samtlige bokse i spillet. Den indeholder 3 pointere til char-arrays: et koordinatsæt og et tilhørende array med boksens styrke. Desuden er der to variable der fortæller antallet af bokse der er fyldt i stacket og hvor mange der ikke er ødelagte. Initielt forestillede vi os at den skulle implementeres som et stack, således arrayernes størrelse var variable, men hvorfor vi ikke gjorde det står i afsnit ??.

void moveBall(Ball * ball)

Denne funktion flytter bolden ved at tage en peger til en Ball som argument og lægger retningsvektoren til x og y koordinaterne.

void moveStriker(long * x,char direction)

Denne funktion tager to variable som argumenter, en pointer til strikerens x-lokation, og strikerens retning. Hvis *direction* er 1 bevæger strikeren sig mod positiv x-retning og STRIKER_SPEED lægges til, ellers bevæger strikeren sig mod negativ x-retning og STRIKER_SPEED fratrækkes strikerens x-lokation.

unsigned char checkBall(Ball * ball,Box * box, int x)

SHIT HERE (MARTIN!!!)

long toTerminalCoordinates(long x)

Denne funktion omdanner tal i 2.14 eller 18.14 til heltal man kan bruge i terminalen. Afrunding laves som vanligvis, ved at afrunde til nærmeste heltal.

void setBallOverStriker(Ball * ball, long st)

Denne funktion tager en Ball og en long som argumenter. Funktionen omdanner st til 18.14 format og sætter boldens x-koordinat til dens værdi. Boldens y-koordinat sættes over strikeren, vha. konstanterne STRIKER_Y og OVER_STRIKER, også i 18.14. Boldens retningsvektor sættes derefter til at gå lodret op, og roteres derefter 40 grader mod venstre.

Box * newBoxStack()

Denne funktion bliver brugt til at lave et nyt Box-stack. Der bliver allokeret plads, så der er plads til antallet af bokse givet ved konstanten MAX_BOXES. Antallet af elementer, size, i stacket sættes til 0 og pegeren til Box-stacket.

void createBoxes(Box * box, char level)

Denne funktion tager en peger til Box-stacket og en character der repræsenterer level som argumenter. Alf afhængigt af hvilken level man tager som argument, bliver Box-stacket fyldt på en speciel måde, således hvert level er unikt.

6.1.2 menu

Dette modul indeholder funktioner til at tegne og vise grafik når man bevæger sig rundt i menuen.

initiateMenu()

Denne funktion renser først skærmen og printer derefter menuen. Slutteligt sættes markøren på Start Game.

moveMarker(int selectedOption)

Denne funktion sætter markøren alt afhængigt af inputtet.

void printDifficulty(short diff)

Denne funktion tager en short som argument. Der udskrives noget forskelligt afhængigt af argumentet.

1. Hvis diff er 1, skrives der "Easy"
2. Hvis diff er 2, skrives der "Normal"
3. Hvis diff er 3, skrives der "Hard"

printHelp()

Denne funktion printer hjælpe-teksten. Startstedet for teksternes x-koordinat bestemmes af konstanten LEFT_BORDER

6.2 Application Interface Layer

6.2.1 graphics.h

Dette modul indeholder grafiske elementer til brug i terminalen. Vi har valgt at bruge nogle definerede konstanter fra spillet (STRIKER_Y, BOX_SIZE, osv.) Dette har vi gjort for at undgå at sende for mange variable, når disse blot alligevel er konstante hele spillet. Med meget få ændringer i funktionerne, kunne man dog ændre dette. Det kan derfor diskuteres om funktionen strengt taget ligger i API-laget. Måske den kan siges at ligge i grænsefeltet.

void drawBox(unsigned char x, unsigned char y, unsigned char color)

Denne funktion tegner en boks med bredden BOX_SIZE og højden 2. Koordinaterne til det øverste venstre hjørne gives som argumenter, sammen med kassens farve, hvor farveskemaet i fgcolor bruges.

void drawChar(unsigned char x, unsigned char y, char tegn)

Denne funktion tager et koordinatsæt og et tegn som argumenter. Tegnet bliver skrevet på det givne koordinatsæt.

void moveDrawStriker(unsigned char x, unsigned char direction)

heeej

void drawBounds(int x1, int y1, int x2, int y2, unsigned char color)

Denne funktion tegner banens kanter. Den tager 2 koordinatsæt som input, x1 og y1 svarende til det øverste venstre hjørne og x2 og y2 svarende til det nederste højre hjørne. Variablen color bruges til at bestemme farven på kanterne.

void drawLogo()

Denne funktion tegner spillets logo. Den bruger konstanten `LEFT_BORDER` til at bestemme på hvilket x-koordinat den skal begynde at skrive fra, således det bliver logoets venstre kant.

6.2.2 lut.h

Dette modul indeholder en konstant tabel med sinus værdier for en cirkel delt i 512 stykker. Hvis x er vinklen i radian indsættes da blot $\frac{x \cdot \pi}{256}$ i tabellen.

6.2.3 math.h

Dette modul indeholder nogle generelle matematiske funktioner, heriblandt sin og cosinus, og to makroer til at regne i 2.14 eller 18.14.

Makroer

Modulet indeholder to makroer, en til at multiplicere to tal i .14 format, og en til at dividere to tal i .14 format, hhv. `FIX14_MULT(a, b)` og `FIX14_div(a,b)`

long sin(int x)

Denne funktion tager en int som argument. Vinklen skal ikke være i radian, men skal bruge opdelingen af cirklen beskrevet i afsnit 6.2.2. Der returneres sinus til den givne vinkel.

long cos(int x)

Denne funktion tager en int som argument. Vinklen skal ikke være i radian, men skal bruge opdelingen af cirklen beskrevet i afsnit 6.2.2. Der returneres cosinus til den givne vinkel.

int arcsin(int y)

Denne funktion tager en int som argument, og finder arcsinus til integeren. Resultatet returneres med korrekt fortegn, således en negativ int også returnerer en negativ vinkel.

6.3 Hardware Abstraction Layer

6.3.1 keys.h

Dette modul får inputs fra knapperne, og kan debounce ved hjælp af timer.h

void iniKeys()

Denne funktion initialiserer den korrekte data-direction på de pins der er forbundne til knapperne, således værdierne kan læses, uden at vi forsøger at skrive outputs samtidigt.

char readKey()

Denne funktion læser fra knapperne, og returnerer en bit streng, hvor de tre knapper er på hver deres plads i strengen. Hvis pladsen tilhørende knappen er 1, betyder det at knappen bliver trykket. Denne funktion kan godt detektere hvis brugeren trykker flere knapper ind samtidigt. Pladserne er konfigureret således:

1. Knappen til højre er på LSB(least significant bit)
2. Den midterste knap er på 1. plads i bit-strengen.
3. Knappen til venstre er på 2. plads i bit-strengen.

char getKey

Denne funktion bruges hvis man ønsker debouncing. Den læser vha. read-Key() og checker derefter om værdien er det samme efter 10 ms og returner dette.

6.3.2 timer.h

Dette modul har med vores primære timer at gøre. Den har 2 globale variable: *time* og *timeWait*. Time tæller hvor lang tid timeren har været tændt. Grunden til at vi har globale variable her, er fordi timeren skal være uafhængig af main-metoden og køre så hurtigt som muligt. Main-metoden kan få adgang til variablene ved nogle setter- og getter-funktioner

void setTimer()

Denne funktion sætter vores timer til prescaling 0, continuous mode og høj prioritet for interrupt funktionen. Denne timer er sat til at køre hvert ms.

void resetTimer()

Denne funktion sætter de til modulet tilhørende globale variable, *time* og *timeWait*, til 0.

void timer0int

Dette er interruptfunktionen tilhørende timeren. Den lægger 1 til *time* og trækker 1 fra *timeWait*.

void SetDelay(int input)

Denne funktion sætter *timeWait* til værdien givet i argumentet. Meningen er at bruge *timeWait* som en slags delay, man kan checke værdien på

getDelay

Denne funktion er blot en getter, der returnerer *timeWait*

unsigned long getCentis()

Denne funktion er blot en getter, der returnerer *time*

Kildeliste

- [1] Randy Yates, *Fixed-Point Arithmetic: An Introduction*, Digital Signal Labs 23. August 2007
- [2] Stephen Brown & Zvonko Vranesic *Fundamentals of Digital Logic with VHDL Design*, McGraw Hill International Edition , Third Edition, 2009
- [3] Zilog, *Using the ZiLOG Xtools Z8 Encore! C Compiler*,
<http://goo.gl/kUPqL7> ,
link sidst checket 23-06. Kan ellers findes ved google søgning eller på Zilog's hjemmeside

- [4] Zilog, *Technical Note Floating Point Routines*
<http://goo.gl/eRBEn0> ,
link sidst checket 23-06. Kan ellers findes ved google søgning eller på
Zilog's hjemmeside