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 gruppemedlemerne, og løst i fællesskab. Rapporten er blevet udarbejdet og gennemlæst i kollektiv.

Tecnical University of Denmark DTU
National Space Institute

30010 - Programming Project
25.06.2015

Abstract

This report is a mandatory part of the B.Sc. EE course 30010, Programming Project.
The report documents the entirety of this course, including the journals and the final product, Reflexball, a program written in C.
The first four days the authors learned how to use the ZDS II - Z8Encore! 4.9.3 tools, how to access the timers, the LED's, the buttons and how to use the terminal for displaying graphic. The code from these exercises were used to implement the HAL and some of the API for the project. The program was designed using a flow chart for the main function and a block diagram for the different program layers. The project was successfully implemented on a Z8 Encore Evaluation Board.

Indhold

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. Binære tal er meget lette at implementere i digital logik, og er derfor 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].

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

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 kommat 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

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

2.4 Repræsentation af negative tal

Hvis vi ønsker at repræsentere negative tal, gøres det oftest på 3 forskellige måder: Signed magnitude, 1's kompliment eller 2's kompliment. 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:

$$-37_{10} = 1100101_2 \tag{3}$$

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 kompliment 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.

$$5 - 3 = 5 + (-3) \tag{4}$$

Ydermere har 2's komplement en cyklisk natur, der gør den smart at bruge sammen med triogonmetriske funktioner, såsom cos og sin. Den cykliske natur gør os istand til at gå begge veje rundt i enhedscirklen Dette brugte vi i øvelse 4, hvor vi sørgede for kun at se på bit 0-8.

2.5 Fixed point vs floating

I dette projekt brugte 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 være for langsomt til vores behov[4][**?**]. Vi brugte derfor kun integers, og omdannede dem til fixed point.

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 modul 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 skal 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. Der skal være mulighed for at vælge sværhedsgrad, hvilket afgør hvor mange liv spilleren har, og hvor hurtigt bolden bevæger sig.

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. Hvis spilleren vinder spillet vises et victory-screen og efter et par sekunder går spillet automatisk tilbage til menuen.

5. 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.

6. Spillerens liv og power skal skrives på LED-skærmen når spillet er igang

7. Spilleren samler power hver gang han ødelægger en kasse. Hvis brugeren trykker på venstre og højre-tasten på en gang bruger han sit power og akitverer high power. Når high power er aktiveret ødelægges kasser når de rammes, uafhængigt af deres liv, og bolden reflekteres ikke, men fortsætter gennem kassen. Power fratrækkes 1 hver gang den ødelægger en kasse.

8. Når spilleren bruger high power, vinder en bane, vinder spillet eller dør skal der rulles en tekst over LED-skærmene. Alt afhænigt af situationen, skal livene og tiden igen vises på skærmen efter teksten er rullet over.

9. Hver gang spilleren går videre til næste level, får spilleren fuldt liv og spillerens power-niveau sættes til 0.

10. Hver gang spilleren går videre til næste level, får spilleren fuldt liv og spillerens power-niveau sættes til 0.

11. Spillere kan pause spillet, ved at trykke på en af knapperne.

12. Ved at trykke på alle knapper samtidigt, aktiverer man chef-mode, som giver en blank skærm.

13. Der er ikke noget point-system i spillet, da vi vælger at lægge fokus andre steder.

3.2.2 Krav til strikeren

1. Strikeren skal maskimalt fylde 10% af skærmen på x-aksen.

2. Strikeren skal være delt ind i 3 forskellige områder. Disse 3 områder skal reflektere bolden på forskellig vis afhængig af indgangsvinklen og hvilken del af strikeren den rammer. Reflektionen skal findes igennem trial and error, og vurderes hvad der virker mest naturligt. I oplægget var der lagt op til at strikeren skulle have 5 områder, men vi syntes ikke det fungerede særligt godt, da det var vanskeligt for brugeren at forholde sig til. Vi har derfor valgt 3 områder.

3. Brugeren skal kunne styre strikeren, vha. knapperne på boardet.

3.2.3 Krav til bolden

1. Bolden skal have 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 Evauluation 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. Disse 2 timere er hhv. 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 continous 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.

$$Reloadvalue = 0.001\text{s} \cdot 18.432.000\text{s}^{-1} = 4800_{16} \tag{5}$$

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 continous 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.

$$Reloadvalue = 0.005\text{s} \cdot 18.432.000\text{s}^{-1} = 2400_{16} \tag{6}$$

4 Plan

4.1 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.

4.1.1 Problemer med realloc

I designfasen 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 **(box).capacity == *(box).size* , 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.

4.1.2 Problemer med knapperne

Vi havde problemer med knapperne på boardet: Vi var i tvivl om vores kode var dårlig, eller om det var fordi knapperne var slidte og ødelagte. Vi lavede

en debouncer, og det hjalp lidt på nogle af knapperne, men vi havde stadig problemer, og nåede aldrig at komme til bunds i problemet. Vi er dog ret overbevist om at problemerne stammer fra de slidte knapper.

5 Implementation

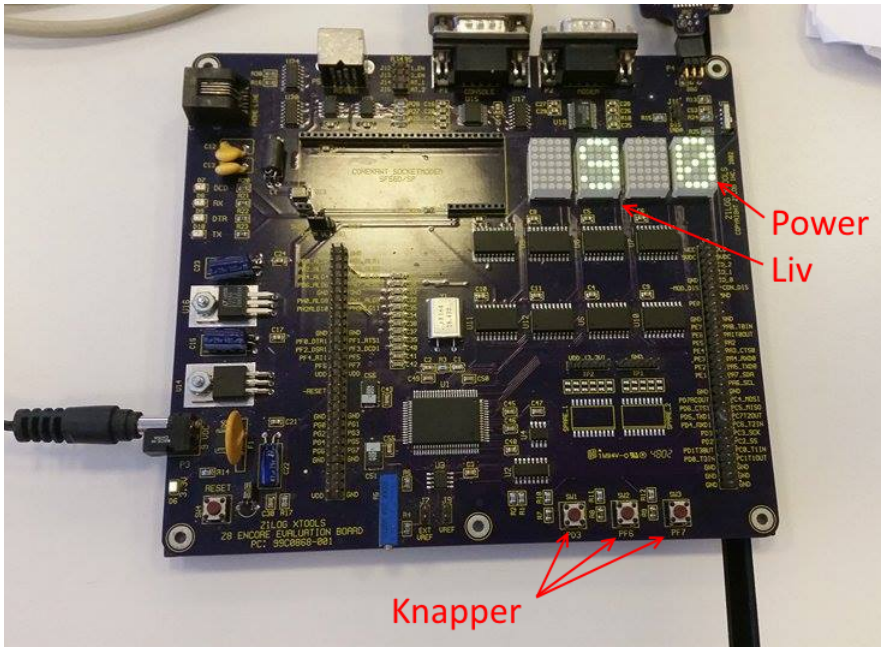
Vi vil i denne sektion gennemgå implementationen af spillet. Vi har valgt blot at gennemgå main-metoden, og flowet af denne. Info om de andre moduler kan findes i dokumentationen.

6 Brugervejledning til ReflexBall

Spillet Reflexball er et arkanoid spil, som handler om at ramme kasser med en bold. Bolden skal holdes i live med en striker, som reflekterer bolden op mod kasserne. Hvis bolden ikke rammer strikeren, og dermed ryger ud af banen, mister man et liv. Kasserne har forskellig styrke alt efter hvilken farve de er. Styrken svarer til det antal gange kassen skal rammes, før den bliver ødelagt. Her er en liste, hvor styrken står til venstre og farven til højre.

1. Lyseblå
2. Lilla
3. Blå
4. Pink
5. Grøn

Når brugeren starter spillet vises en menu. Her kan brugeren styre markøren med knapperne til venstre og i midten. Brugeren vælger den mulighed som markøren står foran ved brug af den højre knap. I menuen kan brugeren se instruktioner, og vælge sværhedsgrad. Når brugeren vælger at starte spillet, vil spillet loade, og bolden sættes over strikeren. Brugeren flytter strikeren ved brug af den venstre og midterste knap. Når brugeren ønsker at sætte bolden i gang trykkes på knappen til højre. Når spillet er i gang kan spillet sættes på pause ved at trykke på den højre knap, og spillet startes da igen ved et tryk på højre knap. Hvis brugeren mister alle sine liv, afsluttes spillet og vender tilbage til menuen. Hvis spilleren derimod får ødelagt alle kasserne vil næste level loade. Ved det sidste level vil en victory screen loade, og vende tilbage til menuen.



Spillets sværhedsgrad har indflydelse på, hvor hurtigt bolden bevæger sig, og hvor mange liv man har. Når man spillet på easy har brugeren 9 liv, og bolden bevæger sig forholdsvis langsomt. På medium er antallet af liv 5, og farten er sat lidt op. På hard har brugeren 3 liv og farten er høj.
Bolden har også egenskaben High Power. Egenskaben aktiveres ved at trykke på den venstre og den midterste knap samtidigt. For hver kasse som brugeren ødelægger, bliver power 1 større. Power skal være 5, før brugeren kan aktivere High Power. Når High Power er aktiveret ødelægger bolden kasserne uanset hvilken holdbarhed de har. Dog mister man 1 power for hver kasse man ødelægger. High Power bliver deaktiveret igen når power er 0. Power bliver nulstillet hvis man mister et liv. Man kan maksimalt have et Power niveau på 9, derefter tæller den ikke yderligere op.

Der findes en hemmelig feature: En boss key er implementeret, som sletter alt hvad der er på skærmen. Denne tilstand aktiveres ved at trykke på alle tre knapper på samme tid. Tilstanden er dog permanent, og spillet skal genstartes, hvis man fortsat vil spille, når chefen er gået.

7 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.

7.1 Application layer

I dette lag findes refball modulet, menu modulet og main modulet.

7.1.1 refball.h

refball.h er et modul der indeholder grundlæggende regler om spillet: kollision, hvordan bolden skal bevæge sig og hvorledes strikeren skal opføre sig. Desuden indeholder den strukturerne *Ball* og *Box*. Ydermere indeholder dette modul også særligt mange konstanter.

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 high power er aktiveret, er implementeret som unsigned char's. I *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 boksenes styrke. Desuden er der to variable der fortæller antallet af bokse der er fyldt i stacket og hvor mange der ikke er ødelagte. I designfasen forestillede vi os at den skulle implementeres som et stack, således arrayernes størrelse var variable, men hvorfor vi ikke gjorde det står i afsnit 5.1.1.

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)

checkBall() er en funktion som kontrollerer og bestemmer boldens bevægelse. checkBall tager bolden, kasserne og strikerens position som argumenter. I funktionen gennemgåes de forskellige scenarier, hvor bolden kan ramme. Først kontrolleres om strikeren er ramt, derefter om kanterne er ramt og til sidst gennemgås alle kasserne og kontrolleres for om de er ramt. Hvis kassen bliver ramt ændres der i boldens retningsvektor, alt afhængigt af hvordan kassen rammes. Endeligt returnerer funktionen et tegn, svarerende til det, bolden har ramt. Hvis bolden intet har ramt foretages ingen ændring på retningsvektorerne, og et blankt tegn sendes tilbage.

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 sætter bolden over strikeren. 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. Afhængigt af levels værdi, bliver Box-stacket fyldt på en speciel måde, således hvert level er unikt.

7.1.2 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 har til formål at printe sværhedsgraden når brugeren vælger:

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

7.2 Application Interface Layer

7.2.1 graphics.h

Dette modul indeholder grafiske elementer til brug i terminalen. Nogle af funktionerne er særligt udviklet til dette spil, men det er muligt de også ville

kunne bruges i andre sammenhæng. Det kan derfor diskuteres om funktionen strengt taget liger 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 givet ved argumentet 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.

7.2.2 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.

7.2.3 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 7.2.2. 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 7.2.2. 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.

7.3 Hardware Abstraction Layer

7.3.1 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. readKey() og checker derefter om værdien er det samme efter 10 ms og returner dette.

7.3.2 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 globable 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, continous 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

[5] Zilog, *Technical Note Floating Point Multiplication*,
http://goo.gl/vUdPW3,
link sidst checket 24-06.