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 exercise journals and the final product, Reflexball, a program written in C and implemented on a microprocessor.
The first part of the course was learning 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 PuTTY 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.

Resume

Denne rapport er en obligatorisk del af 30010, Programmeringsprojektet, som er en af de teknologiske linjefag på B.sc, EE.
Denne rapport dokumenterer helheden af dette kursus, inklusive journalerne og det endelige produkt, Reflexball, der er et program skrevet i C og implementeret på en mikroprocessor. I de første fire dage lærte forfatterne at bruge ZDS II - Z8Encore! 4.9.3 værktøjskassen, at konfigurere timerne, at vise strenge på LED'erne, at læse inputs fra knapperne og at bruge PuTTY til at vise grafik. Koden fra øvelserne blev brugt til at lave et HAL og en stor del af programmets API. Programmet blev designet ved hjælp af bla. flow charts og block diagrammer til at repræsentere programmets lag. Programmet blev med success implenteret på et Z8 Encore Evaluation Board.

Indhold

1 Introduktion **4**

2 Teori **5**
2.1 Binære tal 5
2.2 Unsigned repræsentation 5
2.3 Fixed point kommatal 5
2.4 Repræsentation af negative tal 6
2.4.1 Signed magnitude 6
2.4.2 2's komplement 6
2.5 Fixed point vs floating 7

3 Design af Reflexball **7**
3.1 Tekniske mål 7
3.2 Krav til spillet 7
3.2.1 Overordnede krav til spillet 7
3.2.2 Krav til strikeren 9
3.2.3 Krav til bolden 9
3.2.4 Krav til boksene 9
3.3 Timere 10
3.3.1 Timer0 10
3.3.2 Timer1 10

4 Planlægning og test af programmet **10**
4.1 Problemer 11
4.1.1 Problemer med realloc 11
4.1.2 Problemer med knapperne 11
4.2 Test af programmet 12

5 Implementation **12**
5.1 main.c 12

6 Konklusion **16**

7 Kildeliste **17**

8 Brugervejledning til ReflexBall **18**
8.1 Opsætning af PuTTY 19

9 Dokumentation **20**
9.1 Application layer 20
9.1.1 main.c 20
9.1.2 refball.h 21
9.1.3 menu 22
9.2 Application Interface Layer 23
9.2.1 graphics.h 23
9.2.2 lut.h 25
9.2.3 math.h 25
9.3 Hardware Abstraction Layer 26
9.3.1 keys.h 26
9.3.2 ctimer.h 27
9.3.3 LED.h 27

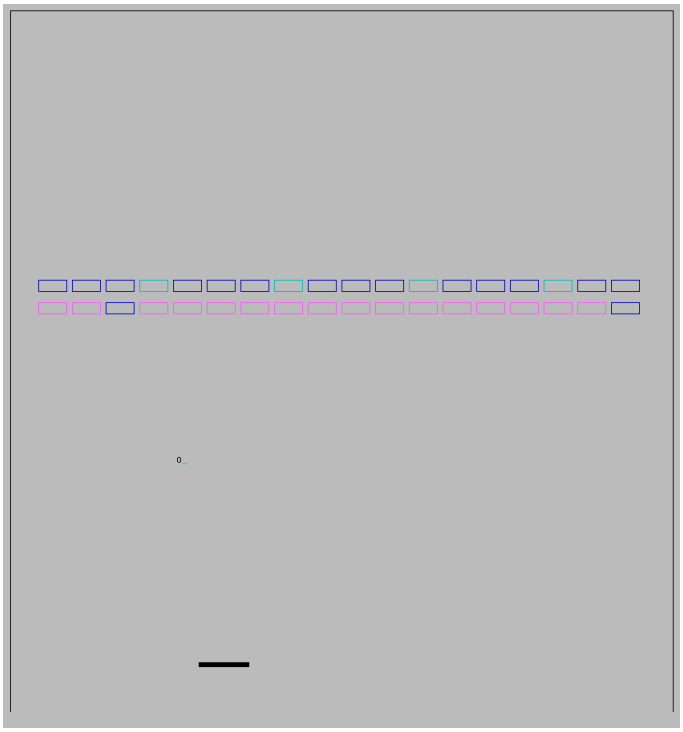
10 Appendix A **29**

11 Kode **29**

12 AppendixB **29**
12.1 Journal 29
12.2 Kode fra øvelserne 34

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.



Figur 1: Reflexball vist i PuTTY.

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

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

2.4 Repræsentation af negative tal

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

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

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 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.

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

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][5]. 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 5 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 sættes til 0.

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

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

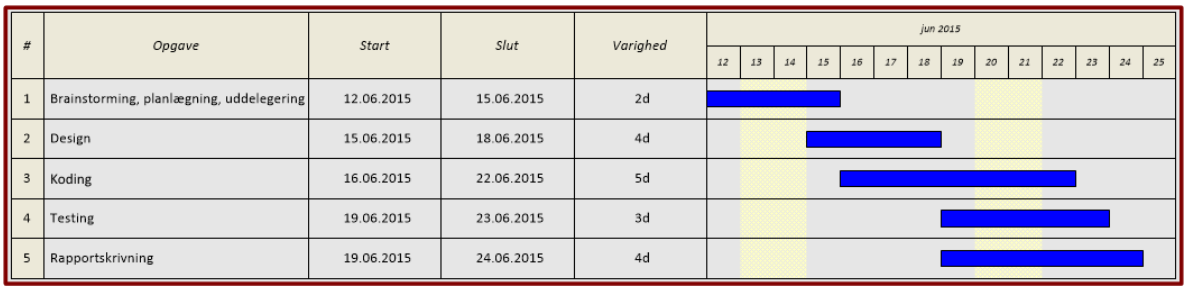
3.3.2 Timer1

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 Planlægning og test af programmet

Planen som blev lagt de første dage i projektet blev egentlig fulgt ganske godt igennem projektet. De første dage blev brugt på brainstorming og planlægning af, hvordan vi ønskede at vores spil skulle se ud, og hvilke features som skulle inkluderes. De tekniske specifikationer er et resultat deraf. I overlappet på design og brainstorm delen blev forskellige dele af projektet delt ud, sådan at hver enkel gruppemedlem kom med et udkast til design af forskellige dele af programmet. Igennem designfasen blev der kigget på flowcharts, samt skrevet pseudokode. Hen mod slutningen af designfasen blev der skrevet mere og mere reel kode. Kodeskrivningen og testing delen følger hinanden. Sidst i kodefasen blev vi nød til at droppe et af de mål vi havde sat os for at



#	Opgave	Start	Slut	Varighed
1	Brainstorming, planlægning, uddelegering	12.06.2015	15.06.2015	2d
2	Design	15.06.2015	18.06.2015	4d
3	Koding	16.06.2015	22.06.2015	5d
4	Testing	19.06.2015	23.06.2015	3d
5	Rapportskrivning	19.06.2015	24.06.2015	4d

Figur 2: Gantt chart af vores plan

overholde planen, som vi havde lagt og blive færdig med projektet. De sidste dage op mod deadline blev der skrevet rapport.

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.

4.2 Test af programmet

Programmet blev testet op mod vores designkrav og de tekniske specifikationer. Først blev grundelementerne kodet og testet, og når det levede op til specifikationerne blev programmet udvidet med nye funktioner. Koden blev altså skrevet og testet med en buttom up metode, hvor det første element var selve banen. Derefter fulgte strikeren, samt det at kunne få strikeren til at flytte sig. Næste skridt var boldens bevægelse, samt den simple refleksion på kanterne og strikeren, hvor indgangsvinkel var lig udgangsvinkel. Så fulgte kasserne med alle deres egenskaber, og boldens refleksion på kasserne. Endeligt blev strikerens udgangsvinkel ændret alt efter indgangsvinkel. En menu blev lavet sideløbende med spillet. Al vores testing blev udført i terminalen, hvor programmet var sat op til at teste, i den forstand, at programmet printede informationer som vi havde brug for. Dette gjorde vi f.eks. i forbindelse med testing af strikeren, hvor vi printede hvor bolden blev detekteret på strikeren, hvad ind- og udgangsvinklen var. Dette gjorde vi for at checke om vores program opførte sig som forventet.

5 Implementation

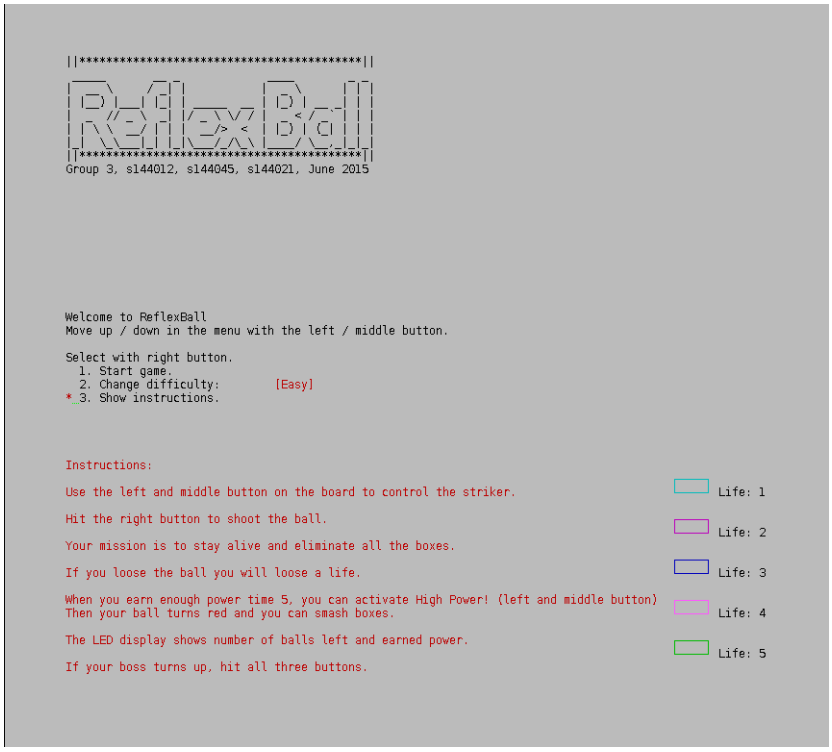
Vi vil i denne sektion gennemgå implementationen af spillet. Vi har valgt blot at gennemgå main-metoden, og flowet af denne. Info om de andre moduler kan findes i dokumentationen.

5.1 main.c

Det her spil styres af en main.c fil med en main funktion. For at forbedre strukturen og øge læsbarheden er selve gameplayet håndteret af en funktion der hedder **Game**. Denne funktion bliver kaldet af main når brugeren vælger start game og returnerer antal liv der er tilbage når spillet afsluttes. På denne måde registeres sejr / tab.
Main funktionen starter med at tegne menuen, sende teksten "Welcome" til LED-displayet og går derefter ind i en uendelig løkke, hvor den venter på at brugeren trykker på en knap. Når brugeren trykker på den venstre og midterste knap bladrer man igennem menuen ved at øge eller formindske

selectedOption, der holder styr på hvor man befinder sig i menuen. Når brugeren trykker på højre knap undersøger programmet *selectedOptions* værdi og foretager en handling baseret på dens værdi. Det kan være at starte et nyt spil, ændre sværhedssgrad eller vise hjælp.

Når et nyt spil begynder, starter funktionen Game med at positionere stri-



Figur 3: Her ses menuen med show instructions aktiveret

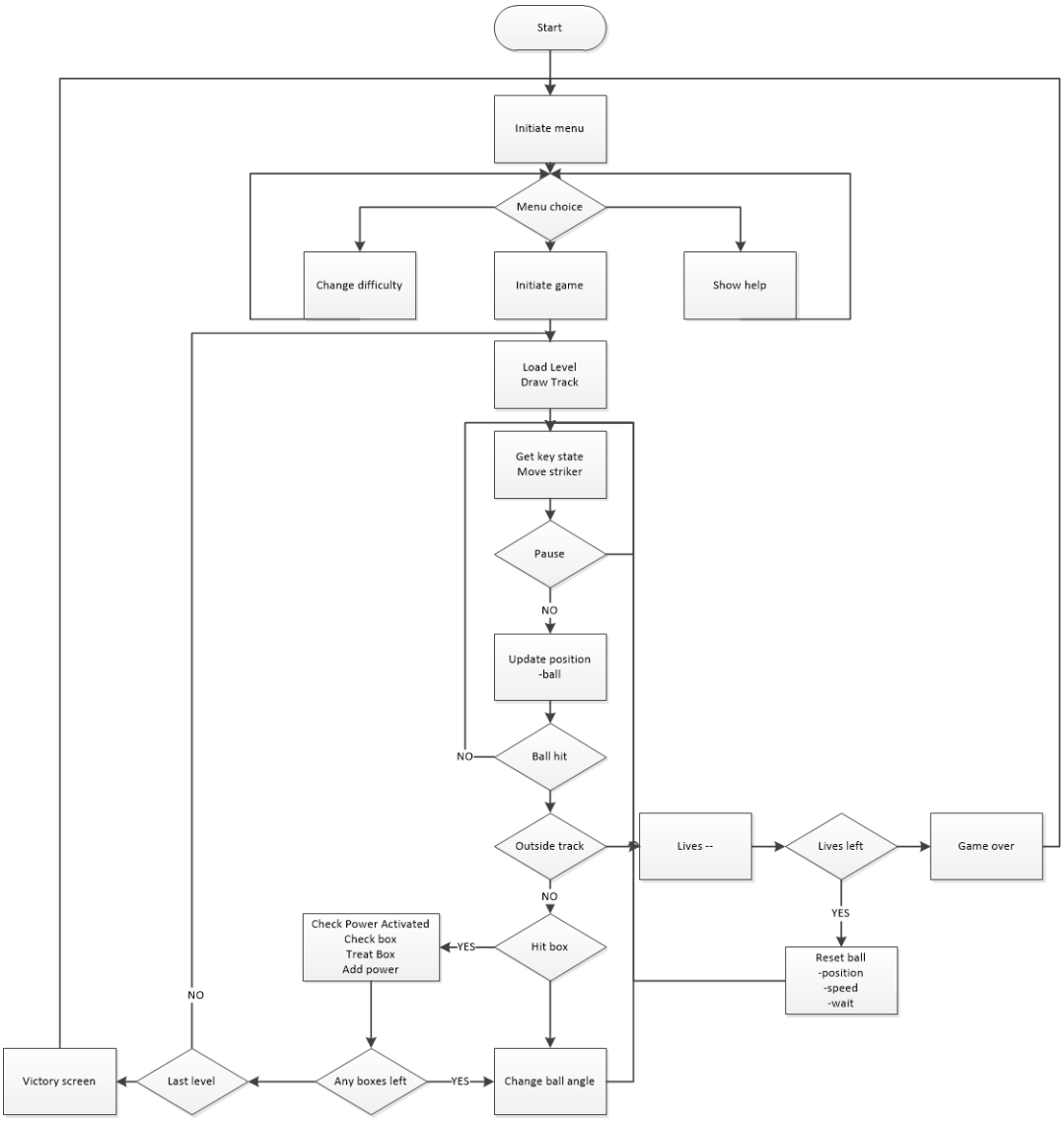
keren, vælge hvor mange bolde brugeren skal have hver level, hvor hurtigt bolden skal køre og tegner tilsidst banen.

Derefter går man ind i en løkke der kører en gang per level helt til max level er nået eller til brugeren ikke har flere liv. Hver gang en ny level starter får brugeren fuldt liv og bolden bliver placeret over strikeren. LED-displayet viser også hvilken level man er nået til. For at skrive tal fra variable på LED-displayet type-castes de til den tilhørende char-værdi og lægges ind i et char array der bliver konkateneret med den resterende streng. Denne streng sendes til LED-displayet med funktionen **LEDSetString**. Funktionen **setLedMode** benyttes for at rigtig visning bliver brugt.

Bolden tegnes med funktionen **drawChar**, hvor det 3. argument er typen character der skal tegnes. Oftest er dette et "o", men hvis bolden rammer

strikeren eller kanterne printes der en char der gør at disse bliver grafisk bevaret. Hvilken character der skal styres af funktionen **checkBall**. Envidere dannes og tegnes bokserne med farver der svarer til styrken.
Når initialiseringen for en level er færdig går man ind i en ny løkke, som kører så længe man har liv og bokse tilbage. Til at begynde med registrerer programmet hvilke knapper der et trykket ind. Når brugeren trykker højre knap skydes bolden. Hvis brugeren trykker på den højre tast igen pauses spillet. når brugeren trykker alle tre knapper bliver skærmen renset (chef-mode).
Hvis spillet ikke er pauset har brugeren mulighed for at flytte strikeren med venstre og midterste knap, og aktivere High Power med begge knapper. Når High Power aktiveres ruller teksten "Power!"over LED-displayet og BEL-characteren printes(siger en lyd, hvis PuTTY er indstillet korrekt)

Det benyttes en tæller til at regulere den frekvens boldens position opdateres ved. Hvis bolden er skudt ud og aktiv, checkes bolden for om den kommer til at ramme en kant, en boks eller strikeren. Hvis den kommer til at gøre det endres boldens retning. Envidere males bolden over.
Der testes for om bolden er ude af banen. Hvis det er tilfældet bliver bolden sat over strikeren igen og brugeren har nu en mindre bold tilbage. LED-displayet opdateres med rigtige antal bolde og mængde *Power*. Til sidst flyttes og tegnes bolden. Bolden tegnes med farven rød hvis High Power er aktiveret.
Når brugeren ikke har flere bolde tilbage eller gennemført spillet kaldes funktionen **drawGameOver** eller **drawVictory**. De funktioner bruger lang tid på at køre, hvilket giver brugeren tid til at se hvad der står.



Figur 4: Flowchart over main

6 Konklusion

I dette kursus har vi fået et kendskab til mikroprocessorere, programarkitektur og planlægning af design og udførelse af et medium størrelse program. Vi har i kursusforløbet med success lavet et program, Reflexball, som levede op til de designmæssige og tekniske krav stillet af Lektor José M.G Merayo, samt vores egne krav. I starten af kurset blev der lavet en plan for projektets udførelse, som i det store hele blev fulgt.

7 Kildeliste

Litteratur

[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

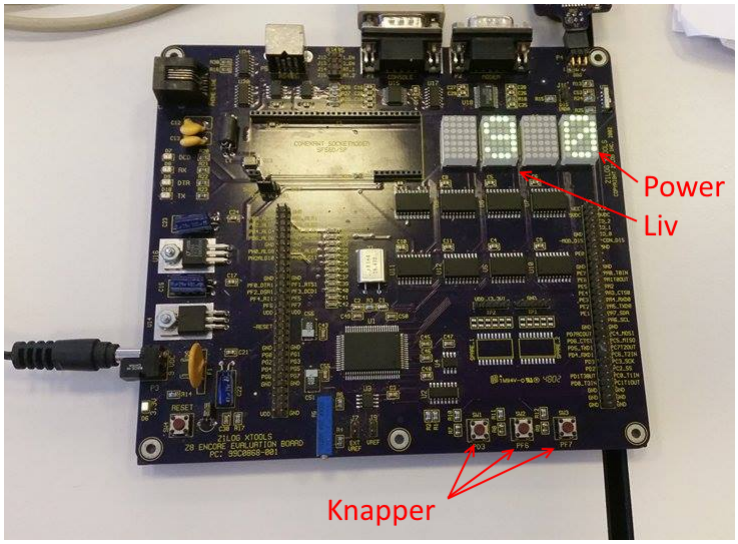
[5] Zilog, *Technical Note Floating Point Multiplication*,
http://goo.gl/vUdPW3,
link sidst checket 24-06.

8 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.

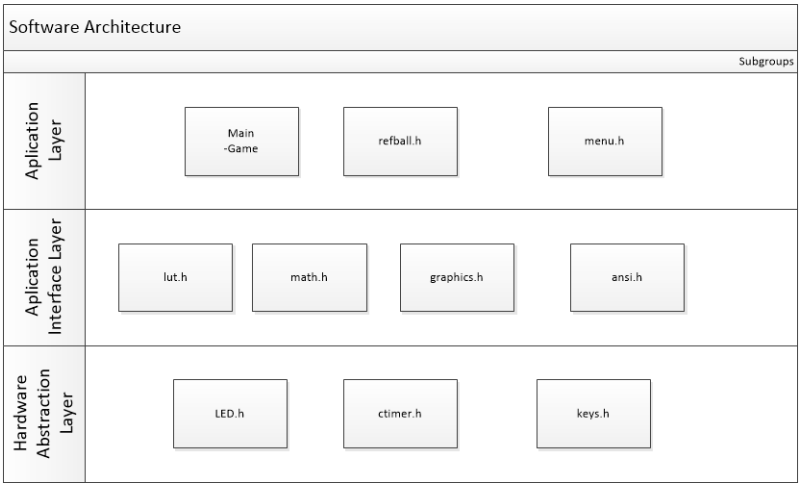
Der findes en hemmelig feature: En boss key er implementeret, som sletter alt hvad der er på skærmen. Denne tilstand aktiveres ved at trykke på alle tre knapper på samme tid. Tilstanden er dog permanent, og spillet skal genstartes, hvis man fortsat vil spille, når chefen er gået.

8.1 Opsætning af PuTTY

Putty-guide: Under *Bell* skal der sættes flueben ved *Play a custom sound file*. Filen *HighPower.wav* skal vælges. Under *Window* skal der være 2000 lines of scrollback, 274 kolonner og 71 rækker, og der skal sættes flueben ved *Change the number of rows and columns* og *Reset scrollback on display activity*. *Cursor appearance* sættes til *Underline*, skriften skal være af typen DejaVu Sans Mono, 7-point, Non-Antialiased. Remote character set er *CP850* og Unicode line drawing code points bruges. Der skal også sættes flueben ved *Allow terminal to specify ANSI colors*. Under *Connection/Serial* skal der vælges en baudrate på 57600, 8 data bits, 1 stop bit og ingen paritet.

9 Dokumentation

I dette afsnitt er projektets filer, funktioner, moduler og makroer dokumenteret og beskrevet. Fokuset er på funktionernes vigtigste egenskaber og virkemåder. Filerne med tilhørende funktioner er delt op i tre forskellige lag: Application, Application Interface og Hardware Abstraction Layer. Se figur 9. I Application Layer ligger de applikationspecifikke filer, disse filer kan bruges på anden hardware uden at modificere dem. I Hardware Abstraction Layer ligger de hardwarespecifikke filer. Disse filer muligør kommunikation mellem hardware og vores software. Disse filer kan benyttes i helt andre programmer såfremt det er samme type hardware der benyttes.



Figur 5: Block diagram over de forskellige lag

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.

9.1 Application layer

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

9.1.1 main.c

ReflexBall styres af /textitmain.c som består af to funktioner. /textbfmain styrer den overordnede kørsel av programmet og menuen, og /textbfGame

kontrollerer de forskellige levels og selve gameplayet. Se nærmere beskrivelse af /textbfmain.c under implementation.

9.1.2 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 4.1.1.

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.

9.1.3 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

9.2 Application Interface Layer

9.2.1 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. drawBall(unsigned char x, unsigned char y, unsigned char color) Denne funktion tager et koordinatsæt og printer et o i farven bestemt af det 3. parameter.

drawStriker(unsigned char x, unsigned char color, char strikerWidth, char strikerY)

Denne funktion tegner blot en striker centrummet for strikeren er x, color er farven, strikerWidth er bredden på hver side, og strikerY er Y-koordinatet.

drawGameOver()

Denne funktion tegner ASCII-kunst af /textitGAME OVER og scroller den ud af skærmen.

drawVictory()

Denne funktion tegner ASCII-kunst af Arnold Schwarzenegger og scroller den ud af skærmen.

scrollText(char y, char delay)

Denne funktion printer mange linjer med mellemrum sådan at det der tidligere er skrevet bliver scrollet væk.

printExampleBoxes(char x, char y, char boxSize)

Denne funktion printer de 5 typer bokser der bruges i spillet sådan at brugeren kan se hvor meget de forskellige boksr tåler.

9.2.2 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.

9.2.3 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 9.2.2. 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 9.2.2. 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.

9.3 Hardware Abstraction Layer

9.3.1 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.

9.3.2 ctimer.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 og køre så hurtigt som muligt. Main funktionen 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*

9.3.3 LED.h

I dette modul bruges der nogle globale variable. Dette gøres for at f.eks. kunne holde styr på hvilken kolonne og LED-enhet der skal lyse. Det hadde været muligt at undgått brugen af globale variable, men da ville man være nødt til

at sende mange flere variable som parametre fra main. Dette ville ikke øge effektiviteten af programmet, men snarere gjort det hele mere kompliceret eftersom dette modul kører på sin egen frekvens. Og siden variablene kun er relevante for modulet er det ikke nogen idé i at de ligger i main.

LEDInit()

Denne funktion indstiller TIMER1 til at give et interrupt hvert 0.5 ms. Envidere initialiseres de globale variabler i LED.C.

timer1int()

Denne funktion kaldes af interrupts fra TIMER1 på boardet. Denne funktion kalder funktionen LEDUpdate().

setLedMode(char valueIn)

Denne funktion er blot en setter der kontrollerer hvilken funktion der kaldes af LEDUpdate.

LEDUpdate()

Denne funktion kalder en af to funktioner, LEDUpdateOnce() eller LEDUpdatePrint().

LEDSetString(char *string)

Denne funktion læser en streng og kopierer den over til den globale variabel /textitstring. Envidere nulstiller den de globale variable.

LEDLoadBuffer()

Denne funktion indlæser bufferen fra et karakterset på den måde, at når funktionen LEDUpdatePrint() kaldes er teksten umiddelbart på displayet. Med andre ord; denne funktion gør at teksten ikke ruller ind og er derfor nyttig når man skal opdatere displayet hurtigt.

LEDUpdatePrint()

Denne funktion bruger tids-multiplexing til at belyse alle kolonnene. For at få dette til at fungere bruges to variable der holder styr på hvilken LED-enhed og kolonne der skal lyse.

LEDUpdateOnce()

Denne funktion ruller en tekststreng over displayet og holder på de sidste fire tegn. Displayet multiplexes på samme måde som i LEDUpdatePrint(), men strengen bliver også rullet. For at kontrollere hastigheden bruges en tæller.

10 Appendix A

11 Kode

12 AppendixB

12.1 Journal

Journal Dag 1

Øvelse 1.1

Den første øvelse startede med at blive fortrolig med udviklingsværktøjet ZDS II. Vi lavede et nyt projekt, hvor indstillingerne blev sat til en CPU af typen Z8F6403 og et registerminde på 4KB. Vi skrevskrevet en simpel C-fil der udskrev "hello world". Pakkene ez8.h og sio.h blev også inkluderet i filen. Til sidst blev C-filen gemt, bygget og simuleret i HyperTerminal.

Øvelse 1.2-1.3

I denne øvelse blev der opgivet en C-fil der indeholdt fejl. Motivationen for dette, var at lære hvordan ZDS II's debugger fungerer. Debuggeren blev brugt til at manuelt køre linje for linje i C-programmet helt til alle fejl var rettet. Debuggerens "Go to Cursor" og "Step Over" blev flittigt brugt sammen med en oversigt over variablene i og r. En af de fejl der blev opdaget var at funktionen power multiplicerte tallet a en gang for meget. Dette medførte bit-overflow, og forkerte udregninger som følge. Denne fejl blev rettet i for løkken, og programmet kørte korrekt.

Øvelse 2

I denne øvelse blev HyperTerminal's output kontrolleret ved hjælp af nogle selvdefinerede funktioner. Vi lavede alle de funktioner der blev bedt om og supplerede også med de frivilligefunktioner(up, down, left, right). Funktionen clrscr() som renser skærmen. funktionen clreol() som renser resten af linjen som markøren står på. Funktionen gotoxy(unsigned char x,unsigned char y) som flyttede markøren til den valgte position. Funktionen underline(char on) som bruges til at vælge eller fravægle om teksten skal være understreget. Funktionen blink(char on) som gør, at den skrevne tekst blinker. Funktionen reverse(char on) som bruges til at bytte om på farvene på forgrund og baggrund.

Til sidst lavede vi en funktion der kunne lave et vindue i terminalvinduet. Funktionen accepterede to sæt med koordinater(for hhv. Nordvestlige og sydøstlige hjørner), stilform for vinduet og titeltekst. Vi besluttede at lave funktionen således den ikke skrev de blanke mellemrum inde i vinduet.
På denne måde kunne man omringe tekst der allerede var skrevet. Funktionen blev udstyret med 2 forskellige stilarter, som gav forskelligt udseende vinduer.

Journal Dag 2

Øvelse 3

I denne øvelse startede vi med at lave en filstruktur vi kunne bruge senere. En header-fil blev defineret med de funktioner der blev skrevet under sidste øvelse. For at systemet skulle køre effektivt blev denne header-fil bliver kun læst ind hvis den ikke var læst ind fra før. Et nyt projekt blev lavet og de gamle funktioner blev gemt i filen ansi.c. De funktioner som beskrev fixed-point aritmetik blev skrevet i filen math.c.

Øvelse 3.1

Denne øvelse, der indeholt spørgsmål omkring bit-manipulation blev ligeledes besvaret.

Øvelse 4.1

Instruktionerne blev blot fulgt og vi fik med succes skabt en LUT vha. excel-programmet, excellut.

Øvelse 4.2

Vi skulle i denne opgave lave en funktion der kunne finde sinus for enhver integer som input – den skulle således kunne finde for negative inputs og positive inputs. Vi kunne i denne sammenhæng blot bruge to complement's repetetive natur sammen med sinus repetetive natur, og derved indse at vi blot kunne fjerne de sidste 7 bit, og kun bruge de første 9. Dette blev gjort ved en simpel bit operation, hvor inputtet blev and'et med 0x01FF, hvilket gjorde, at vores input altid var mellem 0-512.

Cosinus funktionen implementeres også ved blot at bruge egenskaben cos(a)=sin(a+90)
Da vi testede den funktion fandt vi et problem med compileren, da den compilerede -128 som 128.
Følgende workaround blev brugt: -128 = -(int)128

Øvelse 4.3

I denne øvelse skulle vi lave en funktion der kunne rotere en vektor.
Vi lavede første en funktion, der kunne initialisere en vektors x og y værdier. Disse inputs var floating point format, og blev omformet til fixed-point format.
Rotationsfunktionen blev derefter implementeret, hvor multiplikation blev gjort med de opgivne makroer og de før kreerede cos- og sinusfunktioner.

Øvelse 4.4

Denne øvelse, der indeholt spørgsmål omkring bit-manipulation blev ligeledes besvaret.

Journal Dag 3

Øvelse 5.1

I denne øvelse skulle vi lave en funktion, readKey() der kunne læse brugerinput på de tre knapper. Vi skulle derefter teste funktionen med en counter. Vores funktion virkede ikke i første omgang, da vi havde læst kredsløbs-diagrammet forkert og troet at schmitt-triggeren på inputtet var af inverterende type. Vi regner derfor med at inputværdierne var høj når de blev trykket – det var selvfølgelig ikke tilfældet og det gav os problemer – det virkede dog da vi fik det rettet. Vi havde også nogle problemer med debouncing og dårlige knapper. Alt i alt blev resultatet acceptabelt.

Øvelse 5.2

I denne øvelse skulle vi lave et program der ville outputte værdien for vores counter fra øvelse 5.1 og vise dem på LED'erne på boardet. Dette blev gjort ved at sætte værdien af counteren inputtet til LED'erne. Dette program fungerede som forventet.

Øvelse 6

I denne øvelse skulle vi lære at skrive til en timer. Vi skulle i høj grad blot følge instruktionerne. Vi valgte en prescale værdi på 2^7, altså den største mulige for timerne. Reloadværdien blev da udregnet udfra dette. Vores nederste værdi blev bare sat til 1. Interrupt prioriteten blev sat til normal. Vi havde nu problemer med at forbinde til vores board, men simuleringen i programmet fungerede fint. Dagen efter fik vi vores board til at fungere igen, og så at vores program fungerede.

Øvelse 6.1

Vi brugte vores timer fra øvelse 6 til at lave et stopur, og det fungerede også som forventet. Vi brugte her knapperne på boardet til at styre vores stopur. Det var her vigtigt at gøre så lidt som muligt i vores interrupt-funktion, da vi ønsker at den skal være hurtig. Vores samlede stopurs løsning brugte funktionen windows(unsigned char x, unsigned char y, unsigned char x1, unsigned char y2) til at danne rammen. Derefter blev der lavet en funktion som returnerer tiden fra vores timer. Dette blev brugt til at skrive splittime 1 og 2. Derefter blev tiden skrevet sådan, at den ville opdaterer hvert sekund. Programmet blev lavet sådan, at det også var muligt at resette timeren.

Journal Dag 4

Øvelse 7

I denne øvelse skulle vi skrive en streng indeholdende 4 karakterer på vores LED-skærme. Dette gjordes ved at multiplexe signalerne ud. Først skulle vi have en ny clock med en periodetid på 0.5 ms. Vores design fungerede ikke helt i første implementation, da bogstaverne var vendt forkert. Dette skyldes at søjle 0 på LED'en er søjlen helt til højre(og 4 til venstre), og vi havde således vendt bogstaverne om. Dette blev fikset og derefter fungerede vores design efter hensigten.

Øvelse 8

I denne øvelse skulle vi have en streng til at rulle henover skærmen. Dette blev løst med en en buffer som indeholdte 5 bogstaver. Fire bogstaver som vises, og det femte som er klar til at blive hentet ind ligger i bufferen. Den måde rulle effekten blev implementeret var ved at skifte søjlerne en gang til venstre, og dermed give effekten af, at der rulles mod højre. Når der er blevet skiftet 6 gange, svarende til et helt tegn indlæses et nyt tegn i videobufferen, og det som tidligere stod først smides ud. Sådan fortsætter koden indtil den løber ind i enden på strengen. Denne øvelse blev også løst, og forskellige funktioner, som udbyggede funktionaliteten blev lavet, som blev brugt senere i projektet.

12.2 Kode fra øvelserne