Sebastian Nordqvist
André Reis
Malte Wallander
Oliver Hedetoft
Handledare: Lars-Göran Larsson & Bertil Lindvall

Projektrapport

Digitala System

LUNDS
UNIVERSITET

Sammanfattning

Detta är ett projektarbete i Digitala System som utfördes under vårterminen 2019. Projektet gick ut på att montera en krets med en processor driven grafisk dataskärm samt programmera ett spel. I rapporten kommer den grafiska dataskärmen bara vara benämnd “skärm”. Spelet går ut på en karaktär försöker undvika objekt som kommer mot en. Spelet valdes att kallas för PixelRun, och en stor inspirationskälla för spelet var Google Chromes offline spel T-rex runner.

Projektet inleddes med att rita ett kopplingsschema över kretsen som skulle koppla processorn till skärmen. För att rita kopplingsschemat användes ritprogrammet Eagle. Processorn som användes var en AVR Atmega1284 och skärmen var en grafisk skärm. Datablad användes för att ta reda på vilka pinnar på skärmen som skulle kopplas vart på processorn. Därefter testades samtliga komponenter som sedan skulle vara på kretskortet. Inga fel uppstod och komponenterna kunde därför lödas fast på kretskortet. Ett fel uppstod när skärmen senare skulle tändas, vilket en stor portion av tiden gick åt till att felsöka. Felet var en avbruten sladd vilket gjorde att signalerna igenom den endast kom fram ibland.

Efter att problemet hade lokaliseras påbörjades programmeringen av spelet. Koden skrevs i C och assembly. Spelet fick tre stycken lägen: Start meny, spelet samt en game-over meny. I start menyn står det “Pixel Run” och “start”. När användaren trycker på mittknappen påbörjas en nedräkning, sedan startas spelet. När karaktären nuddar ett hinder avslutas spelet och game-over menyn visas tillsammans med antalet hinder karaktären tog sig förbi.

Projekten utfördes under två månaders tid och resultatet blev en funktionell koppling och ett spelbart spel som använder sig utav tre knappar och en skärm. Spelet har några få buggar men det är ingenting som förstör spelupplevelsen eller påverkar spelaren på något sätt.

Nyckelord

PixelRun - Projektets slutresultat. Ett spel som går ut på att ta sig förbi hinder och samla poäng.
Atmega1284 - En mikrodator

Kopplingsschema - En schematisk bild över en elektrisk krets

T-rex runner - Google Chromes offline spel

Abstract

The idea of this project is to create a game, similar to the T-rex game that Google Chrome gives you when you have no internet access. The project touched on multiple parts, such as programming in C and Assembly, to making your own circuit diagram. A key-component that we had to learn was the graphic display. It involved a lot of problem solving from errors to tackle methods on how to bring the best gaming experience. Everything from a certain pin not working on the hardware side, to solving collisions with character and objects inside the game itself.

A menu, game and an endgame screen with score was implemented. The game also has increased difficulty that speeds up the obstacles depending on your score. There are still things to improve and add, such as character movement and inconsistent buttons but would over all need improvement in hardware and software especially since we only have 8 MHZ clock frequency to play with.

Keywords

PixelRun - The project endresult. A game

Atmega1284 - A microcomputer

Circuit diagram - A schematic picture of an electrical circuit

T-rex runner - Google chrome’s offline game

Innehållsförteckning

Förord

Detta är vårt projektarbete i Digitala System (EITA15) i årskurs 1. Vårt projekt tog många olika former under tidens gång. Vår första tanke var att bygga ett slags pussel, men vi bytte snabbt spår till att koda ett spel och tog vår inspiration från Google Chromes offline spel T-rex runner. Vårt slutgiltliga spel fick namnet PixelRun och går ut på att undvika hinder. Vi

blev väldigt nöjda med slutresultatet och hade väldigt kul under processen. Vi lärde oss även en hel del och i den här rapporten redovisar vi detta.

1 Inledning

1.1 Bakgrund

Detta är ett projektarbete i kursen Digitala System (EITA15) som utfördes under vårterminen 2019 på uppdrag av vår handledare. Projektet är baserat på ett det grundläggande spelkoncept att undvika objekt som kommer flygande mot en karaktär. Vi har valt att kalla spelet PixelRun. En karaktär kommer befinna sig på den vänstra väggen av skärmen och springa längs golvet av skärmen. Samtidigt kommer objekt dyka upp som rör sig mot en. Desto längre in i spelet man når, desto snabbare kommer dessa objektet att röra sig.

1.2 Syfte

Meningen med projektet är att få en djupare förståelse för att koppla kretsars samt programmera en processor, men även det som hör till ett projektarbete. T.ex. planering, val av komponenter, felsökning, problemlösning och färdigställning. Projektet sker i grupp vilket innebär att arbetsfördelning och kommunikation också är viktiga aspekter. En krets ska kopplas med bland annat en skärm och processor varvid denna sen ska programmeras för att skapa ett spel. Spelet ska bestå av en karaktär samt hinder som måste undvikas för att samla poäng. Samt ska det finnas tre olika lägen: Menyn, själva spelet och “Game Over” skärmen.

1.3 Målformulering

Skapa en förståelse för hur den grafiska skärmen fungerar och realisera en krets innefattande en processor och grafisk skärm. Samt programmera ett fungerande spel i C som uppfyller syftet.

1.4 Problemformulering

Hur går man tillväga när man monterar en processor driven skärm?
Hur kopplar man ihop resterande krets med skärmen?
Hur kommunicerar man med skärmen?

Hur programmerar man ett enkelt spel i C?

2 Teknisk bakgrund

Skärmen som används är en GDM12864HLCM (Liquid Crystal Display Module), utvecklad av XIAMEN OCULAR OPTICS. Den består utav en 64x128 pixlar stor skärm som på hårdvarunivå är indelad i två stycken 64x64 pixel chip bredvid varandra. Dessa två chip är oberoende av varandra och måste adresseras var för sig. Skärmen har 20 stycken pins kopplade på sig. Sex av dessa är för ström och jord till skärmen och dess bakgrundsbelysning. Åtta stycken är data pins som kan skicka/sända och ta emot/läsa data som skickas från processorn. De sex sista pinnarna tar emot olika instruktioner direkt till skärmen. Dessa består av: “select chip 1”, “select chip 2”, “reset”, “read/write”, “data/instruction”, och “chip enable”.

“Select chip” väljer vilken av de två chipen man vill kommunicera med. “Reset” återställer signalen. “Read/write” bestämmer vad data pinnarna ska göra, om de ska ta emot eller skicka iväg signaler. “Data/instruction” bestämmer om det som data pinnarna skickar är data, alltså det som ska skickas upp och ritas på skärmen. Eller om det är instruktioner för att aktivera någon av de fem inbyggda funktionerna som finns. Den sistnämnda, “Chip enable” är den som sen skickar signalerna som ligger på pinnarna in till skärmen.

Knappar är kopplade med ett pull-down motstånd. Detta innebär att om knapparna inte är nedtryckta, blir insignalen till mikroprocessorn en logisk nolla. Trycker man ner knappen blir insignalen en logisk etta. Detta är kritiskt att veta om man ska programmera knapparnas funktioner i mikroprocessorn.

Mikroprocessorn är inställd på 8 MHZ för att maximera prestandan. Skärmen består av totalt 8000 pixlar, vilket gör det oerhört svårt att arbeta på 1 MHZ som processorn var inställd på från början.

3 Metod

3.1 Planering

Projektet inleddes med att diskutera hur slutproduktet skulle se ut samt bestämma hur spelet skulle se ut och vilka komponenter som behövdes för detta. Följande komponenter valdes för att realisera kretsen:

4x Knappar
1x Grafisk Display(GDM12864C)
1x Processor (Atmega1284)
1x Potentiometer
2x Kondensatorer (100nF)
4x Resistorer på 10k□
1x Resistor på 100□

Ett kopplingsschema för kretsen ritades i Eagle med hjälp av databladen för processorn och skärmen samt kunskap från laborationerna i kursen. Spelet fick sin inspiration från Google Chromes offlinespel T-rex runner då det verkade vara simpelt nog för att realisera på en enkel processor och skärm. Slutprodukten valdes att kallas för PixelRun, och efterliknar T-rex runner som går ut på att undvika hinder och överleva så länge som möjligt.

3.2 Montering

Monteringen bestod av att koppla samman alla komponenter efter hur de kopplats i kopplingsschemat. Till en början testades allting i en mindre koppling med hjälp av en breadboard för att se till att alla komponenterna fungerade och för att upptäcka eventuella fel i kopplingsschemat. Då inga problem upptäcktes kopplades alla komponenter direkt på ett kretskort.

Processor, skärm, strömkälla, jordkälla, knappar, potentiometer och resistorer löddes fast i kretskortet. Alla dessa delar kopplades sedan samman enligt kopplingsschemat med ledningstråd genom användningen av en “wire-wrap tool” för att skapa den färdiga produkten.

3.3 Felsökning

Skärmen gick inledningsvis inte att starta och en stor portion av arbetet lades ner för att felsöka problemet. Följande fel lokaliserades:

- Alla kopplingar till skärmen var spegelvända och behövdes kopplas om (pin 1 hade sladden som pin 20 skulle ha osv).
- Alla kopplingar dubbelkollades för att se om de var rätt kopplade, samt så att de inte kortslöt varandra.
- Koden ändrades till en väldigt simpel kod som enbart skulle tända och efter en stund släcka skärmen igen.

Tillslut startade skärmen men nya problem uppstod. Kod för att rita ut stora svarta block på skärmen skrevs men det var väldigt slumpmässigt om det faktiskt funkade eller inte, trots kod som i teorin skulle fungera. Därmed måste felet vara något av följande: Störningar i kretsen, problem med hårdvaran eller fel på kopplingen. Följande åtgärder gjordes:

- Kondensator kopplades in för att ta bort störningar och skapa en mer stabil spänning utan lika många transienter.
- Fler och mer lokala jordningar kopplades på.
- Skärmen ersattes mot en ny.
- Alla kopplingar dubbelkollades och vissa byttes ut för att säkerställa att ingen kortslutning skedde.

När detta ej gav ett förbättrat resultat så testades varje pin på Atmega:n och skärmen med en logikpenna för att se till så att alla signalerna lämnade processorn och nådde skärmen. Då framkom det att en av kopplingarna hade en bruten sladd vilket gav en mycket dålig (och oftast ingen) koppling däremellan. Efter ett byte av denna kopplingen och exekvering av koden vi använt tidigare fick vi tillslut skärmen att funka och programmering av själva spelet påbörjades

3.4 Programmering

Inledningsvis fick programmeringen en långsam start då ingen i gruppen hade någon erfarenhet av skärmen. Där fanns många funktioner, diagram och tabeller i databladet som till en början var nästintill obegripliga. Därför började allting med att gå in i debug läget och skicka in signaler enligt databladet för att se vad allt gjorde och vad som behövde göras för att starta skärmen.

Efter en grundläggande förståelse för hur skärmen fungerar, påbörjades programmeringen av funktionerna. Funktioner för att kommunicera med skärmen skrevs för att förenkla arbetsprocessen, den viktigaste funktionen är “LCD_write” som används i olika kombinationer för att rita ut allting på skärmen.

Eftersom att skärmen är grafisk så betyder det att pixlarna måste adresseras var för sig, det går alltså inte att be skärmen att skriva ut ett ord. Istället får många iterationer av funktionen LCD_write användas där varje iteration kan tända eller släcka åtta stycken specifika pixlar. För att tända eller släcka alla pixlar på hela skärmen, som är 64x128 pixlar stor, så krävs det 1024 stycken iterationer. Eftersom att varje iteration tar några hundra klockcykler så kan det lätt börja bli hackigt när mycket måste suddas ut och ritas upp igen snabbt.

När funktionerna för skärmen var helt färdiga blev resten av programmeringen rätt smidig. Därefter följde funktionerna för själva spelet. Spel karaktären skapades och knapparna kopplades till karaktärens rörelse funktioner. Sedan skapades hinder och kod skrevs för att röra hinderna mot karaktären. Karaktärens rörelse var bunden till knapparna som vi sedan la interrupts på, därefter skapades kollision mellan karaktären och hinderna. En sats blev tillagd som flyttar spelet mellan de tre olika lägena: Start meny, Spelet, samt game over. Med hjälp av LCD_write gick det att skapa funktioner som skriver ut olika ord. Detta användes för att bygga upp en start meny där spelets namn “PixelRun” samt ordet “start” stod. En funktion gjordes även för att skriva ut “Game Over” och “Score: xxx” på game over skärmen. En nedräkning skapades även som räknar ner från 3 när spelaren startar spelet. C hade en inbyggd random funktion som användes för att slumpa fram ordningen som hinderna kom i. Det sista som implementerades var att spelaren faktiskt får ett poäng varje gång den passerar ett hinder, samt att objektens hastighet är beroende på denna poängen.

3.5 Färdigställning

Efter detta var spelet färdigt och allt som återstod var att testa den slutliga produkten för buggar. En bugg som upptäcktes var att knapparna inte fungerade som de skulle. Ibland kunde en knapptryckning bli en följd av förflyttningar på karaktären, istället för att flytta den ett steg som det var tänkt. Detta beror på att knapparna är mekaniska och inte är helt stabila, utan "hoppar" lite i sin signal. Lösningen som föreslogs var att prova med en sorts delay efter knapptryckningen, för att bara läsa knappen en gång, och att sätta kondensatorer som ett sorts RC filter vid knappen för att filtrera bort de oönskade hoppen. De båda förslagen förstörde dock resten av spelet.

Delayen gjorde att hela spelet stoppades efter förflyttning av gubben så att hinderna tydligt stannade till. Det löste inte heller problemet utan skapade bara fler. Kondensator lösning fungerade bra under ett test av det, samt när endast en var inkopplad. När sedan de andra kondensatorerna också kopplades in (tre stycken totalt, en till varje knapp) började knapptryckningarna aktivera varandra på något sätt. Därmed kunde ett knapptryck för att flytta karaktären uppåt istället resultera i att spelet startade om från början. Efter några timmars testande skrotades lösningen helt istället, då ingen slutsats kunde dras om varför det hände. Ingen lösning för detta problemet implementerades alltså utan det kvarstår i den slutgiltiga prototypen.

4 Analys

Programmeringen av PixelRun skedde i programmet "Atmel Studio 7.0" och kodades med en blandning av C och assembly. C användes för det mesta då det är ett högnivåspråk som är maskinnära. Detta innebär att C fungerar utmärkt för programmering av mikroprocessorer. Inslag av assembly användes för att programmera väldigt exakta fördröjningar. T.ex så fanns där en funktion till skärmen som krävde en delay på exakt 1µs för att fungera korrekt. Hemsidan "bretmulvey" användes för att skapa kod som svara mot det exakta antalet klockcykler man behövde för att uppnå en specifik delay. En annan fördel med Atmel Studio är att det går att skicka insignaler och läsa utsignaler direkt i programmet i ett "debug" läge. I detta läget går det att stega igenom koden och se den exekveras rad för rad.

Skärmen GDM12864C valdes eftersom det var en grafisk display som tillät friheten att skapa precis vad som helst på skärmen. Spelet som skapades var unikt i sig och friheten underlättade programmering av det hela.

Som tidigare nämnt i metoden tar det några hundra klockcykler att rita eller sudda ut åtta stycken pixlar i rad. Detta medför att när spelet blir snabbare och hinder måste ritas och suddas ut mycket snabbare så blir det tydligt att processorhastigheten sätter sina gränser. Eftersom att processorhastigheten redan är på sitt maximum av 8MHZ så är den enda lösningen till detta problemet att optimera koden så att den tar mindre klockcykler och därmed går snabbare att köra.

En annan sak som även nämns i metoden är det att knapparna inte är helt stabila, utan att de "hoppar" även när de är nedtryckta. Detta är ett vanligt problem som nästintill alla konstruktioner som använder sig utav mekaniska knappar. Ett RC filter samt delay testades, dock löste inte detta problemet då lösningarna enbart medförde fler problem.

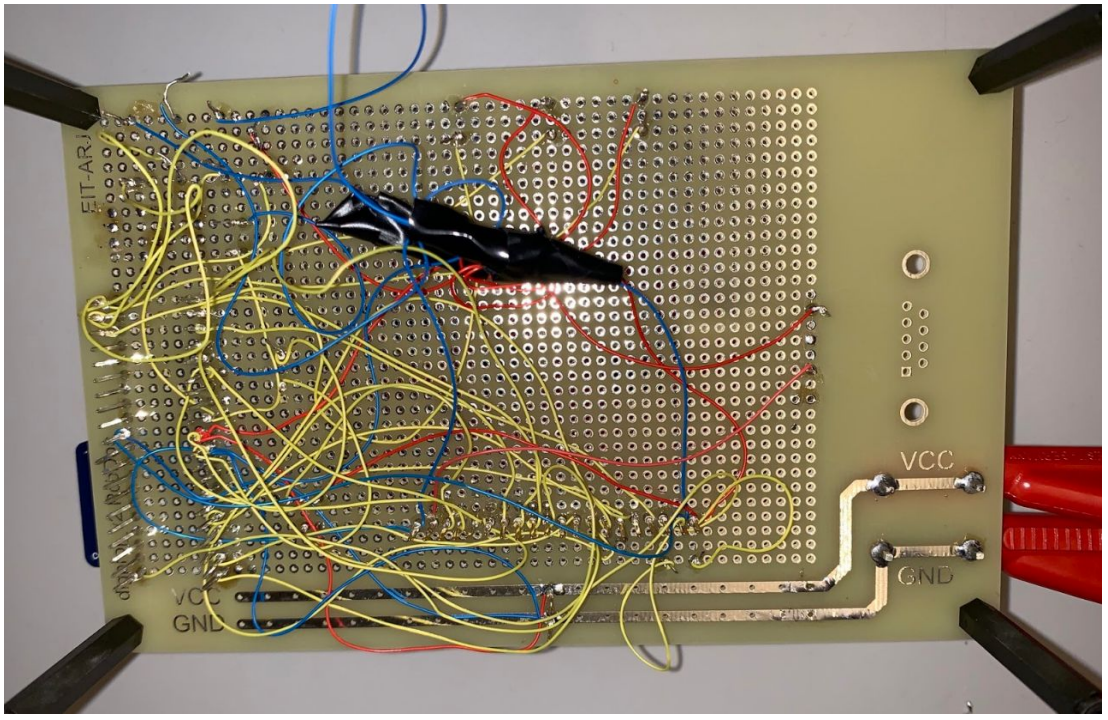
En bugg uppkommer ibland när spelkaraktären förflyttas. När karaktären ritas upp på nytt så ritas ibland även ett åtta pixlar stort sträck ut någonstans på skärmen. Sträcket är dock inte permanent utan suddas ut igen så småningom när ett hinder åker över det. Någon lösning på problemet kunde dock inte hittas eftersom vi inte kunde identifiera ursprunget till problemet.

5 Resultat

Kretskortet kopplades upp med de komponenter som nämns i planerings delen av metoden. Nästa stycke kommer beskriva var allt sitter på kretskortet och utgår ifrån figur 1.

Alla blåa sladdar leder jord, de röda leder spänning och de gula bär signalerna som skickas eller tas emot i processorn. Längst ner till vänster sitter processorn och precis ovanför sitter 10k resistorerna. Kondensatorerna till RC-filtret skulle ha suttit under 10k resistorerna ifall de funkat, vilket togs upp i analysdelen. Längst upp i mitten sitter knapparna som är kopplade till processorn via resistorerna för att skapa en pull-down resistor till knapparna.

Längst ner sitter skärmens 20 pinnar. De alla går direkt till processorn utom de variabla spännings mottagarna. De går via potentiometern och sedan till processorn. Potentiometern sitter mellan processorn och skärmen. En spännings mottagare i skärmen skulle endast ha 4.2V in och är därför kopplad genom resistorn som syns längst till höger på kretskortet. Till höger om skärmen, samt mellan processorn och skärmen sitter de två kondensatorerna. De två kopplings strecken absolut längst ner på kortet är jord och spänning och kopplas via två stycken pinnar in till processorn som sedan leder de vidare till de andra komponenterna.



Figur 1. Undersidan av kretskortet

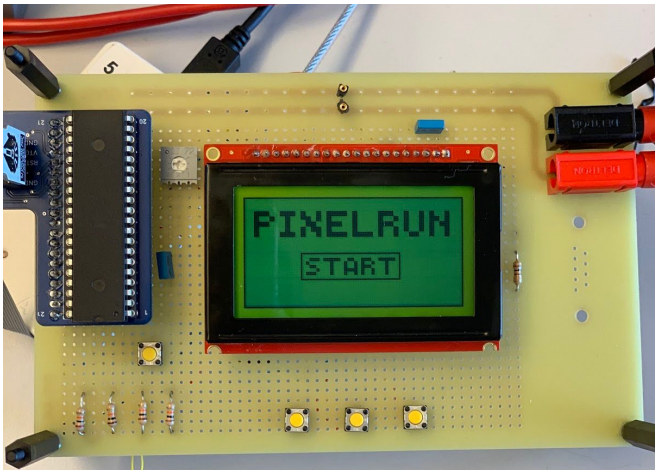
Slutresultatet blev PixelRun ett spel som går ut på att undvika hinder och samla poäng. Tre stycken olika lägen finns i spelet, startskärmen, själva spelet, och en “game over” skärm. Oavsett vilket läge spelaren befinner sig i så finns där en ram på en pixel som sträcker sig runt hela skärmens kant.

När spelet startas kommer spelaren till startskärmen (figur 2) och när spelaren sedan och trycker på mittknappen möts han av en nedräkning innan spelet börjar. I spelläget (figur 3) ritas spelkaraktären ut och hinder börjar röra sig mot karaktären. Spelkaraktären är 12 pixlar bred och 14 lång. Hinderna varierar i utseende men är alla fyra pixlar breda och ordningen

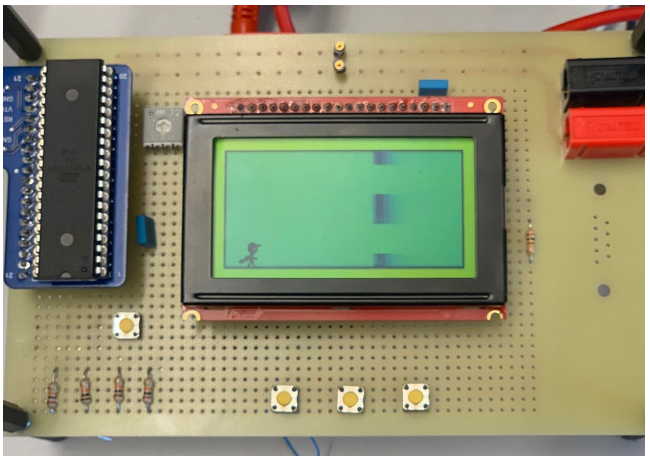
som de renderas i är slumpmässig. För varje hinder som spelaren tar sig förbi får han ett poäng. Dessa poängen används sedan för att räkna ut hur snabbt spelet ska gå. Efter varje poäng ökning finns där en chans att hinderna börjar röra sig snabbare och därmed att spelet blir svårare. Max nivån på hastigheten nås efter 100 poäng är uppnådda och är då lite mer än fem gånger så snabbt som start hastigheten.

När spelaren slutligen kolliderar med ett hinder så möts han av en stor “GAME OVER” text samt får han även sin slutgiltiga poäng utskriven. Detta visas i figur 4. Ännu ett tryck på mittknappen tar tillbaka spelaren till start menyn.

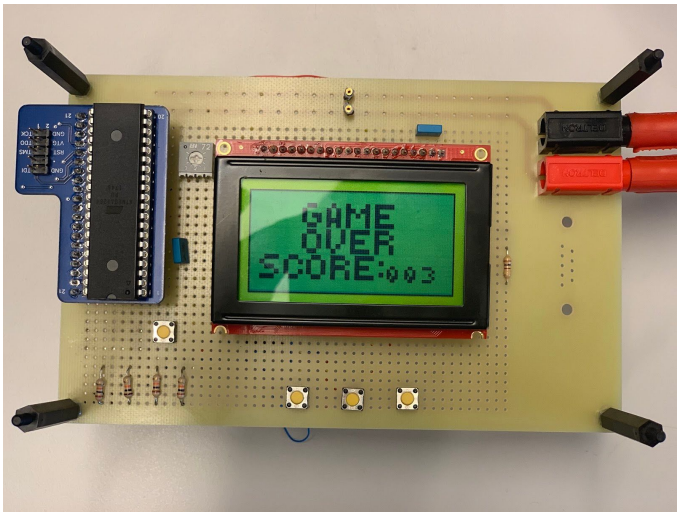
Metoden som undersöker om en kollision har skett mellan spelaren och ett hinder är relativt simpel. Varje gång ett hinder har förflyttats så tittar metoden om hindret och spelaren har samma x och y koordinater och om de har det så “dör” spelaren. Om de däremot inte har samma koordinater, och därmed inte har kolliderat så får spelaren istället ett poäng.



Figur 2. Startskärmen



Figur 3. Spelet



Figur 4. Game over skärmen

Figur 5 är ett exempel på hur koden kunde se ut. Här används funktionen LCD_write för att adressera åtta stycken pixlar åt gången.

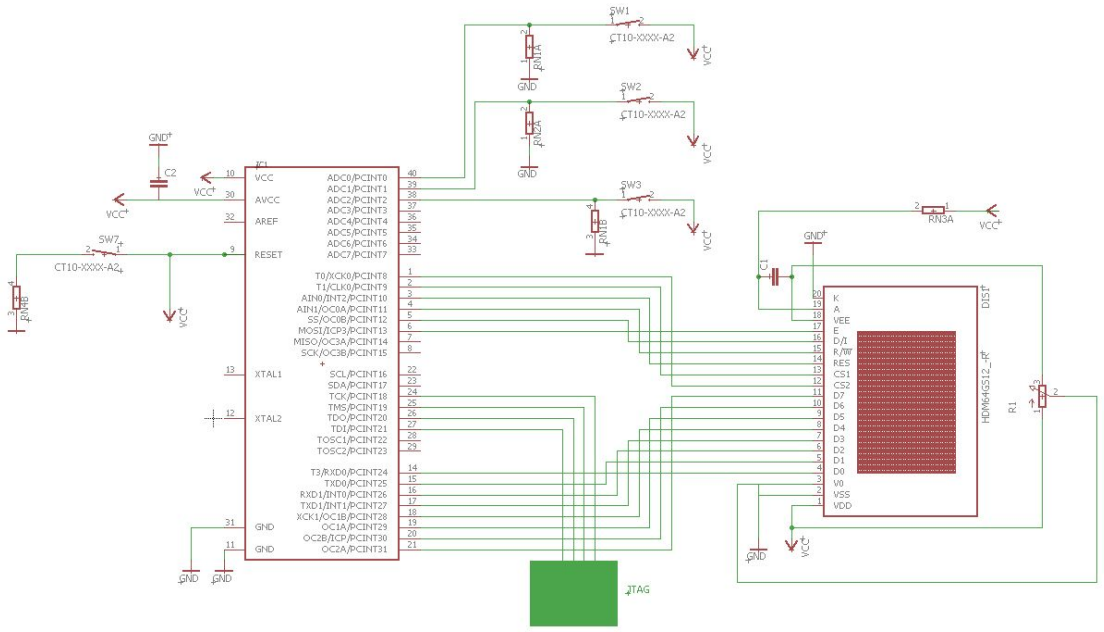
```
void gameOverText(uint8_t x, uint8_t y){

        LCD_write(x, y, 0b11110000);
        LCD_write(x+1, y, 0b00011111);
        LCD_write(x, y-1, 0b11110000);
        LCD_write(x+1, y-1, 0b00011111);
        LCD_write(x, y-2, 0b11110000);
        LCD_write(x+1, y-2, 0b00011111);
        LCD_write(x, y-3, 0b00001110);
        LCD_write(x+1, y-3, 0b11100000);
        LCD_write(x, y-4, 0b00001110);
        LCD_write(x+1, y-4, 0b11100000);
        LCD_write(x, y-5, 0b00001110);
        LCD_write(x+1, y-5, 0b11100000);
        LCD_write(x, y-6, 0b10001110);
        LCD_write(x+1, y-6, 0b11100011);
        LCD_write(x, y-7, 0b10001110);
        LCD_write(x+1, y-7, 0b11100011);
        LCD_write(x, y-8, 0b10001110);
        LCD_write(x+1, y-8, 0b11100011);
        LCD_write(x, y-9, 0b11110000);
        LCD_write(x+1, y-9, 0b00000011);
        LCD_write(x, y-10, 0b11110000);
        LCD_write(x+1, y-10, 0b00000011);
        LCD_write(x, y-11, 0b11110000);
        LCD_write(x+1, y-11, 0b00000011);
```

.

Figur 5. Kodexempel som skriver ut bokstaven “G”.

Kopplingsschemat i figur 6 visar hur skärmen, processorn, knapparna och JTAG:en är kopplade med varandra.



Figur 6. Kopplingsschemat.

6 Slutsats

Kopplingen på kretskortet fungerar och stämmer överens med kopplingsschemat. När det gäller spelet så kom alla önskvärda funktioner med. Spelet består av karaktären och hinderna, samt de tre lägena som nämns i syftet och trots några få buggar fungerar spelet som det är tänkt.

Monteringen av skärmen var knepig till en början då det inte var helt klart vilka pinnar som skulle kopplas till vad. Databladet fick läsas noga för att förstå vad som skulle kopplas vart och efter diskussion med handledaren så gick det tillslut.

Resten av kretsen kopplades snabbt ihop och behövde inte alls gå igenom skärmen som först var misstänkt. Allting som inte hade direkt med skärmen att göra, så som knapparna och de tillhörande resistorerna kopplades istället till processorn som sedan skötte resten.

Kommunikation med skärmen var förvånansvärt enkel när det stod klart vad alla funktioner gjorde och hur de fungerade. Eftersom att metoder skrevs inom klassen för skärmen som tog hand om det viktigaste, gav det oss funktionen LCD_write som används i de andra klasserna. Detta underlättade väldigt mycket då det gick att programmera precis som vanligt, utan att ta hänsyn till vad skärmen höll på med.

Programmeringen av spelet var det som gick smidigast av alla stegen i projektet. Det viktigaste var switch-case satsen som håller koll på var i spelet man befinner sig. Utöver det så användes LCD_write flitigt i olika metoder för att rita ut och uppdatera positionen för karaktären och hinderna när de rörde sig, men även för att rita ut statiska objekt såsom ramen runt om hela spelet och all text.

6.1 Framtida utvecklingsmöjligheter

Eftersom att projektet gick ut på att skapa ett spel så finns där alltid mer funktioner och förbättringar att göra fall tiden hade räckt till.

Något som från början var planerat men som fick överges på grund av tidsbrist är ett highscore system. Poängen som en spelare samlar ihop medan han spelar sparas ingenstans utan när ett nytt spel startas försvinner den helt. Det som var tänkt från början var att spelaren, precis som i det gamla tetris skulle få skriva in en signatur på tre stycken bokstäver som sedan sparas tillsammans med poängen, sorterade efter högst poäng först i en highscore lista. Dock räckte tiden inte till och därför fick denna funktionen skrotas.

Det var även tänkt att spelkaraktären skulle ha animationer på när han "springer" framåt på marken av skärmen mot hinderna, samt en flyg-animation när karaktären befinner sig i luften. Detta är något som var tänkt att fixa som "det sista lilla", men det insågs att hela programmet hade programmerats utifrån karaktärens höjd, och med animationerna så hade stora delar av spelet fått ändras. Detta fanns där ingen tid till och därmed finns inga animationer i spelet.

Bakgrundsmusik till spelet, samt ljudeffekter vid knapptryckningar, ökning av hastighet i spelet och kollision är något som hade haft en positiv påverkan på slut produkten. När den simpla skärmen sätter gränser för hur mycket man kan göra grafiskt, så kan det vara ett bra sätt att ändå öka kvalitén på produkten.

Några andra tillägg att göra skulle vara t.ex optimering samt att fixa knapparna som analysdelen tar upp.

7 Terminologi

MHZ - Megahertz, hertz är ett sätt att mäta periodtid och hastighet för bland annat processorer.
Pull-down resistor - En relativt stor (10k i vårt fall) resistor kopplas in mellan jord och en knapp för att får en logisk nolla när knappen ej är nedtryckt och en logisk etta när den trycks ner.
Logikpenna - Ett verktyg som används för att i realtid mäta om en signal är hög eller låg. Mäts direkt på pinnarna man vill undersöka.
AtmelStudio - Ett användargränssnitt för att programmera och exekvera kod.

RC filter - En kondensator och en resistor koppling som används för att filtrera ut oönskade signaler/frekvenser.

8 Källförteckning

Lunds Tekniska Högskola. https://www.eit.lth.se/index.php?ciuid=1135&coursepage=7251 (Hämtad 2019)

http://www.bretmulvey.com/avrdelay.html (Hämtad 2019)

Hemert, Lars-Hugo. 1992, 2001. Digitala kretsar. 3:10 uppl. Lund: Studenlitteratur AB