Tetris By Tobbe

EDI021 Digitala Projekt

Tobbe Lundberg – dt05tl3@student.lth.se

Handledare: Bertil Lindvall

Lund, Mars 2009

Abstract

This paper discusses how the famous game of Tetris can be implemented using an ATmega16 microcontroller, a 128x64 pixel LCD display, and some other assorted hardware. It begins with specifications, moves on to implementation and ends with a discussion of the finished system. The main problems were understanding how the LCD display worked and trying to fit everything in to the limited memory of the ATmega16. The result is a fully functional implementation of Tetris that can send game scores to a PC for keeping track of high scores.

Innehållsförteckning

Inledning

Om kursen

Kursen EDI021 Digitala Projekt erbjuder studenter att konstruera ett system där huvudkomponenten är en Atmel AVR ATmega16 eller en Motorola 68008. ATmega16 är en komplett minidator med bland annat processor, ram, programminne (flash) och eeprom. 68008 är bara en processor så där får studenten själv koppla in ram med mera.

Då hela kursen är centrerad runt ett stort projekt utan någon schemalagd lärartid krävs mycket enskilt arbete och noggrann tidsplanering. Kursen ger bra övning i att på egen hand producera ett resultat, vilket utbildningen i övrigt har alldeles för lite fokusering på.

För datateknologer innebär kursen en välbehövlig närkontakt med elektronik i allmänhet och lågnivåprogrammering av digital elektronik i synnerhet.

Om projektvalet

Jag valde att göra ett tetris för att jag tänkte det skulle ge en lagom balans mellan hårdvara och mjukvara. Hårdvaran är relativt enkel då den i stora drag bara består av en microkontroller, en lcd-skärm och några knappar.

Den enkla hårdvaran passar mig eftersom jag inte har någon erfarenhet av att jobba med sådant. Mjukvaran visste jag skulle bli mer komplex, men det trodde jag inte skulle innebära några problem då jag är van vid programmering.

Jag valde att basera projektet på en ATmega16 främst för att det är någonting jag skulle kunna se mig själv använda i framtida hobbyprojekt.

Dessutom hade jag aldrig programmerat ett tetris, och det är väl något alla datoringenjörer borde ha gjort minst en gång?

Syfte

Det främsta målet jag hade med kursen var att få en djupare förståelse för hur man programmerar en microkontroller och hur man kan använda den för att styra extern hårdvara. Jag ville även få lite praktisk erfarenhet av att själv designa och konstruera ett system som till stor del består av hårdvara.

Arbetsgång

Kravspecifikation

Innan arbetet på hårdvaran och mjukvaran kunde sättas igång skulle en kravspecifikation produceras. Kravspecifikationen skulle skrivas åt det optimistiska hållet så att det alltid skulle finnas att göra. Författandet av kravspecifikationen tvingar fram tydligt formulerade mål och det hjälper till under utvecklingen av systemet.

Kravlistan nedan är uppdelad i två delar. Den första delen beskriver mål som måste uppfyllas för att projektet ska ses som lyckat. Den sista delen är en lista på ”i mån av tid”-krav.

Högsta prioritet

1.1. När en ny omgång tetris startar skall poäng och nivå nollställas och spelplanen skall inte innehålla några tetrisblock.
1.2. När konstruktionen startar skall en ny omgång tetris startas direkt.
1.3. Tetrisfigurerna skall komma i slumpmässigt vald ordning.
1.4. När en (vågrät) rad på spelplanen fullständigt täcks av tetrisblock skall denna rad elimineras och alla tetrisblock ovanför skall flyttas ner exakt en rad.
1.5. När en rad elimineras skall spelaren få (1+nivå) poäng.
1.6. När 10 rader har eliminerats skall nivån öka med ett steg.
1.7. Det grafiska användargränssnittet skall visa spelplanen, spelarens poäng, spelarens nivå och nästa tetrisfigur.
1.8. Spelplanen skall vara 20 gånger 12 rutor (tetrisblock) stor.
1.9. Varje tetrisfigur skall bestå av fyra tetrisblock.
1.10. När spelaren förlorar skall det ej komma fler tetrisfigurer.
1.11. Om ett tetrisblock placeras så att det täcker någon del av översta raden på spelplanen har spelaren förlorat.
1.12. Tetrisfigurerna startar högst upp på spelplanen och rör sig själva neråt tills de stöter emot ett tetrisblock eller botten av spelplanen. När de gör det stannar figuren. En tetrisfigur som ej har stannat benämns som att vara i rörelse.
1.13. Tetrisfiguren som är i rörelse skall kunna flyttas åt vänster, åt höger och roteras. Var och en av dessa tre funktioner skall skötas av varsin hårdvaruknapp.
1.14. När en tetrisfigur har stannat skall det fortfarande gå att flytta figuren i sidled en kort stund, om det finns utrymme för det på spelplanen.
1.15. Hårdvaruanvändargränssnittet skall bestå av en grafisk skärm som kan visa 64x128 pixlar (bredd gånger höjd) och fyra tryckknappar för att styra tetrisfigurerna.

Lägre prioritet

2.1. Genom att trycka på en av hårdvaruknapparna skall tetrisfiguren som är i rörelse flyttas nedåt snabbare än normalt.
2.2. När användaren startar konstruktionen skall en meny visas på skärmen. Menyn skall innehålla val för att starta spelet eller att stänga av konstruktionen. Detta krav ersätter krav 1.2.
2.3. Tetrisfigur i rörelse rör sig nedåt med en viss hastighet. Denna hastighet skall ökas för varje ny nivå.
2.4. När spelaren förlorar skall spelarens poäng föras över till en dator där poängen registreras på en highscore-lista.
2.5. Kommunikation med datorn skall ske via datorns serieport (COM-port).
2.6. När spelaren förlorar skall en ny omgång direkt startas.
2.7. När spelaren förlorar skall en meny visas på skärmen där det går att välja att starta en ny omgång eller att stänga av konstruktionen. Detta krav ersätter krav 2.6. Det är samma meny som beskrivs i krav 2.2.
2.8. Konstruktionen skall ha en tryckknapp som sätter igång konstruktionen om den är avstängd och stänger av den om den är igång.

Hårdvara, mjukvara, hårdvara och mjukvara igen

I samband med att kravlistan skrevs skulle även ett kopplingsblockschema ritas. Första versionen av schemat har den hårdvara som krävs för att kunna uppfylla de högst prioriterade kraven. Bara vissa av kraven med lägre prioritet kan implementeras. Blockschemat finns i appendix b.

När kraven och kopplingblockschemat var godkänt av handledaren delades de verktyg och komponenter ut som behövdes för att konstruera tetriset.

När jag skulle bygga hårdvaran försökte jag vira så många sladdar som möjligt så att det inte skulle göra så mycket om jag råkade göra fel. Men jag började med att löda fast de komponenter jag inte kunde vira för att inte smälta några virsladdar med lödkolven. När strömförsörjning, microkontroller, knappar och kontakt till skärmen var på plats började jag med mjukvaran.

Jag hade väldigt liten erfarenhet av att skriva kod för ATmegan, så jag fick ta väldigt små steg i början. I princip skrev jag ett litet testprogram för varje enskild sak jag ville kunna göra. När jag förstod hur saken jag testade fungerade lade jag till funktionaliteten i mitt program för tetriset. Till exempel gick det första testprogrammet jag skrev endast ut på att sätta alla pinnar på port A låga förutom en som jag satte hög. Sen skrev jag ett litet testprogram för att lära mig hur externa avbrott fungerade. Så höll jag på tills jag hade större delen av den funktionalitet jag behövde för mitt tetris.

När den grundläggande funktionaliteten var på plats började jag fundera på lite utökningar av hårdvaran för att kunna implementera de sista kraven. Utöver det som står i kraven byggdes det även in stöd för att kunna köra tetriset från ett 9V-batteri så att man inte är beroende av ett labbaggregat för att kunna demonstrera systemet.

Med den nya hårdvaran på plats var det dags för mer mjukvara. Det första som gjorde var att lägga till en meny i spelet som visas när tetriset startas och när spelaren har förlorat. Precis som kraven säger kan man i menyn välja att starta en ny omgång tetris eller att stänga av konstruktionen.

Enligt kraven ska den slutliga poängen för en omgång skickas till datorns serieport. Det görs, men även poänguppdateringar under spelets gång skickas till datorn. På datorn körs ett Windows-konsolprogram skrivet i C++ som tar emot poängen och visar dem på skärmen. När en omgång är slut skrivs den slutgiltiga poängen in i en highscorelista. Highscorelistan sparas i en vanlig textfil på datorn.

Hårdvara

Microkontroller Atmel AVR ATmega16

Kärnan i hela projektet är en microkontroller, eller enchipsdator, från Atmel som heter ATmega16. Den har bland annat en 8 bitars processor, 1kB ram, 16kB programminne och 32st in/ut-pinnar. Pinnarna kan användas som digitala utpinnar där du kan välja att skicka ut en digital etta eller digital nolla. Används de som vanliga digitala inpinnar kan de läsa digitala ettor och nollor. Pinnarna har även en annan specifik funktion som man kan aktivera om man

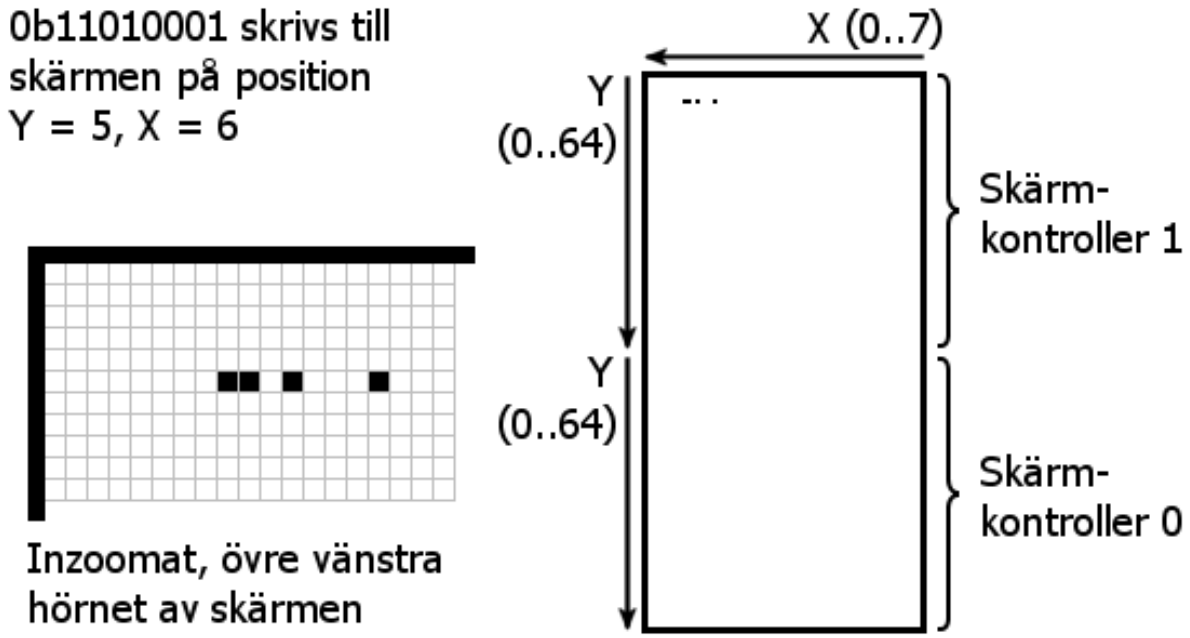
vill. Till exempel har jag valt att aktivera extern avbrott på en av pinnarna och några av pinnarna används i JTAG-läge för att man ska kunna programmera och debugga ATmegan.

LCD-skärm Batron BT128064B

Spelet visas på en svartvit (svartgrön) skärm som är 64 pixlar bred och 128 pixlar hög. Skärmen har en kontrollerkrets som styr de övre 64x64 pixlarna och en annan likadan krets som styr de nedre 64x64 pixlarna. Detta innebar vissa intressanta utmaningar när halva tetrisfiguren skulle ritas med hjälp av den ena kontrollern och den andra halvan skulle ritas med den andra kontrollern.

Skärmbilden uppdateras alltid med åtta pixlar i taget. Varje skärmhalva har åtta kolumner i x-led och 64 rader i y-led. Varje kolumn är åtta pixlar bred, och det är alltså en sådan kolumn som uppdateras i taget. En byte skrivs till databussen på displayen och en binär etta blir en aktiverad (mörk) pixel och en binär nolla blir en ljus pixel.

Kontrollerkretsen heter KS0108B eller HD61202 beroende på tillverkare.



Spänningsregulator LP3855ET

Detta är en spänningsregulator som har väldigt lågt spänningsfall och som ger 5V ut. Det låga spänningsfallet gör valet av matningsspänning lite mer flexibelt än om en vanlig 7805 hade använts. För ett 9V-batteri som jag använder fungerar dock båda två lika bra.

Mjukvaruavstängning

För att ATmegan ska kunna stänga av sig själv används en krets bestående av tre motstånd på 4700 ohm, en BC109C-transistor (NPN) och en BC179-transistor (PNP). En pinne på microkontrollern styr den ena transistorn som i sin tur styr den som i öppet läge förser microkontrollern med spänning. Se appendix b för kopplingsschema.

UART

Normalt behövs en MAX232-krets eller liknande för att konvertera ATmegans 0-5V nivåer till rs232-protokollets ±3 till ±15 volt. Jag har dock valt att bara ha kontaktpinnar på min konstruktion eftersom jag har en sladd med både inbyggd rs232-nivåkonvertering och inbyggd usb till serieport-adapter.

Ett alternativ till MAX232 och motsvarigheter hade varit en modul byggd runt ett FTDI-chip så att jag hade haft en USB-port direkt på konstruktionen, men sådana moduler är tyvärr relativt dyra.

Knappar

För att styra spelet används fyra stycken knappar. De är alla anslutna till en pinne som genererar ett avbrott när en knapp trycks ner. De är även ansluta till var sin vanlig in/ut-pinne så att jag kan läsa av vilken knapp det var som trycktes ner. Det interna pullupmotståndet är aktiverat på alla fyra in/ut-pinnar.

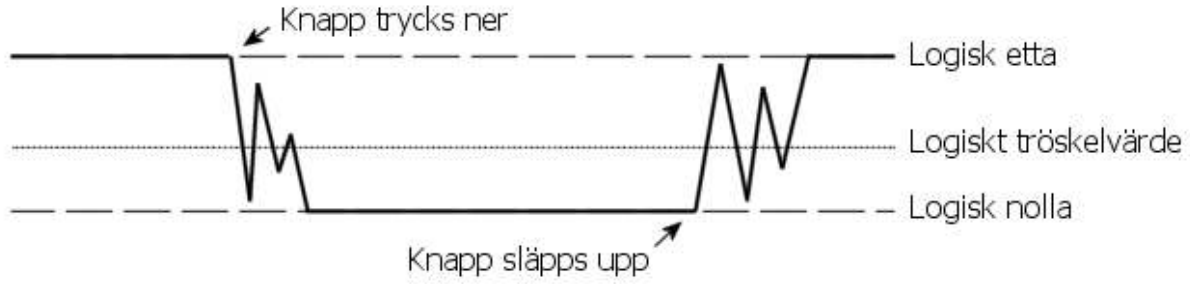
Mjukvara

Knappar

Knapparna var det första jag fick att fungera. Jag har anslutit alla knappar till ett externt avbrott för att slippa polla knapparna hela tiden. Från början växlade jag bara en utpinne mellan låg och hög när jag tryckte på en knapp för att verifiera att det fungerade så långt.

När jag fick igång en timer också kunde jag implementera en mer sofistikerad version av knapphanteringen.

Ett problem med knappar är något som kallas bounce, eller studs på svenska. Vad det innebär är att när knappen trycks kommer mekaniken i knappen att generera vad microkontrollern tycker ser ut som många korta tryckningar på knappen. Figur 1 är ett illustrerande exempel på hur det skulle kunna se ut.



Figur 1. Signal vid knapptryckning

Det finns både hårdvarubaserade och mjukvarubaserade lösningar på problemet med studs. Jag valde en egenpåhittad mjukvarulösning som bygger på att de flesta knappar har en studstid på max 10 millisekunder och att en responstid på mindre än 50 millisekunder inte

märks av människor (JG Ganssle, *A Guide To Debouncing*, http://www.ganssle.com/debouncing.pdf).

När jag får ett knappinterrupt startar jag en timer som kommer att skapa ett timerinterrupt efter ca 30 millisekunder. Jag avaktiverar även knappinterruptet för att inte fler interrupts ska genereras på efterföljande studs. När timerinterruptet kommer borde knappen ha slutat studsa och jag läser av alla pinnar knapparna är anslutna till för att se vilken knapp som blev nertryckt.

Timerinterruptet kommer att fortsätta komma var 30de millisekund tills jag upptäcker att ingen knapp är nedtryckt. Då stänger jag av timern och aktiverar knappinterruptet igen.

Timers

ATmegan har tre timers, två åttabitars och en sextonbitars. Två av dess används.

Sextonbitarstimern används till att hantera knappstudsarna och den ena åttabitarstimern används till att automatiskt flytta tetrisfigurerna neråt på skärmen.

Då det bara är en åttabitarstimer kan den inte räkna så långt (kan max räkna 255 klockcykler). Även med den största prescalern som är 1024 räknar den till 255 för fort för att direkt kunna användas till att uppdatera tetrisfigurens y-position. Därför finns det en vanlig räknarvariabel i avbrottsrutinen för den timern som håller reda på hur många gånger avbrottet har skett. Först efter ett visst antal avbrott uppdateras tetrisfigurens position. Hur många gånger avbrottet måste ske för att positionen ska uppdateras beror på vilken nivå spelaren är på i tetriset. Ju högre nivå desto färre avbrott behöver ske. Detta leder alltså till att tetrisfigurerna kommer att falla snabbare och snabbare allt eftersom spelarens nivå ökar.

Åttabitarstimern används även till att bestämma hur länge spelaren har på sig att flytta tetrisfiguren i sidled när den har kommit så långt ner den kan i sin nuvarande x-position. En liten period för detta behövs för att låta spelaren skjuta in en tetrisfigur under en annan.

Displayen

All kod för att uppdatera displayen ligger i filen display.c. Där finns bland annat en funktion för att initiera de två kontrollerkretsarna för över och undre halvan av displayen, en funktion för att tömma displayen och en funktion för att rita ut tetrisfigurer.

Enligt databladet för displayen måste man vänta minst en millisekund mellan varje instruktion till displayen. Detta gör att det blir plågsamt slött att uppdatera hela displayen. Som tur var kunde jag köra betydligt fortare än så. Nu väntar jag bara några få klockcykler mellan varje instruktion.

Trots att displayen uppdateras fortare än vad databladet säger den ska göras går det inte att hela tiden rita om hela displayen. Därför har stor vikt lagts vid att skriva displayfunktionerna så att de uppdaterar så få pixlar som möjligt.

Huvudloop

Efter att ha ställt in all hårdvara går programmet som körs på ATmegan in i en oändlig loop där både menyn och själva spellogiken hanteras.

Meny

Koden för menyn är väldigt enkel. Det första den gör är att rita upp hela menyn en gång. Sen ritas ingenting förrän användaren trycker på upp- eller nerknappen. Då ritas bara nedre halvan av menyn om, där de olika menyvalen finns, för att göra uppritningen så snabb som möjligt.

Spel

Det första som sker i speldelen av huvudloopen är att den aktiva tetrisfiguren ritas upp på skärmen. Sen kollas det om figuren har kommit så långt ner den kan, antingen för att den har nått botten av spelplanen eller för att den har stannat ovanpå en annan tetrisfigur.

Har figuren stannat kollar koden om en någon rad har fullbordats och där med skall tas bort. Det finns även kollar för om spelaren har förlorat, om poäng skall uppdateras, med mera. Under denhär tiden är avbrott avstängda så att inte användaren kan flytta på tetrisfiguren medans kollarna körs.

Highscorelista

Så fort spelaren får poäng skrivs den nya poängen ut på UARTen. När spelaren förlorar skrivs den slutgiltiga poängen ut.

UARTen kan kopplas till en dators USB-port med hjälp av en speciell kabel jag har. Kabeln skapar en virtuell serieport i datorn som det går att läsa ifrån.

Jag har skrivit ett program för Windows i C++ som kan läsa från en serieport på datorn och hantera poängen ATmegan skriver ut. Det är ett enkelt konsollbaserat program som kontinuerligt skriver ut aktuellt poäng under spelets gång och sen uppdaterar en highscorelista när spelet är slut.

Resultat

Nästan alla krav blev uppfyllda. Det enda som inte klarades av var att det skulle finnas en knapp för att när som helst kunna stänga av konstruktionen. Som det är nu får användaren vänta tills aktuell spelomgång är slut och menyn visas på skärmen. Där kan han välja att stänga av. Alternativt får användaren bryta strömmen till systemet, men då kommer inte highscorelistan att sparas på grund av att programmet som kör på datorn för att ta emot poäng aldrig blir informerat om att spelet håller på att stängas av.

Jag har ett fullt fungerande tetrisspel och jag är mycket nöjd med vad jag har åstadkommit. Mina kompisar som jag har visat det för har blivit imponerade.

Slutsats

Lärdomar

Jag har lärt mig mycket om både hårdvarukonstruktion och lågnivåprogrammering. Jag känner mig nu redo att helt på egen hand kunna lösa problem med hjälp av en ATmega16 och kringelektronik.

Jag har blivit bättre på att läsa och förstå datablad och kan göra informerade val gällande vilka komponenter som är bäst för en given uppgift.

En annan viktig lärdom har varit att inte bara ta kod direkt från Internet utan att förstå hur den fungerar. Det är bättre att försöka först vad som står i databladen och skriva kod efter den informationen.

Problem

Skärmen

Det första större problemet jag stötte på var att få displayen att fungera. Jag kunde först inte förstå vad jag behövde göra för att initiera displayen. Jag förstod inte heller riktigt hur det fungerade att välja vilken halva av skärmen man skulle rita på.

Jag försökte hitta information på Internet, men hittade inte så mycket fakta. Jag hittade däremot flera färdigskrivna bibliotek för att rita ut på skärmen. Jag fick inte något av biblioteken till att fungera och när jag kollade på koden blev jag nästan bara ännu mer förvirrad då alla såg ut att göra olika saker för att uppnå samma resultat.

Tillslut tog jag hjälp av min handledare Bertil Lindvall som kom och visade hur jag skulle tolka databladet. Han visade också hur jag kunde manipulera signalerna till skärmen direkt från datorn utan att behöva skriva kod för att gör det genom att använda debugmöjligheterna i AVR Studio.

Det visade sig att även om jag hade fått något av biblioteken att fungera hade jag aldrig kunnat använda dem. Eftersom de är generella bibliotek som ska kunna användas i många situationer är de för stora och för slöa för mig. Jag behöver specialskrivna funktioner.

Platsbrist

Bland det första jag ville rita ut på skärmen var bakgrunden till spelet. Jag sparar den i en matris av åttabitars heltal som är åtta element bred och 128 element hög. Detta blir totalt 1024 bitar. Redan här fick jag problem. Alla variabler sparas i ramminnet, och eftersom ramminnet bara är 1024 bitar blev det överfullt direkt. Eftersom jag inte behövde ändra i matrisen sparade jag den i programminnet istället.

Platsbrist i både ramminnet och programminnet blev någonting som förföljde mig genom hela projektet.

Alla konstanta datastrukturer fick jag lagra i programminnet, men då även programminnet har en begränsad storlek började det också ta slut efter ett tag. Till slut fick jag ta bort några

ganska stora men snabba och specialskrivna funktioner från displaybiblioteket och ersätta dem med en mer generell funktion som var betydligt mindre, men tyvärr också mindre effektiv. Uppritningen blev tydligt slöare.

Siffrorna för poäng och nivå sparas i programminnet så att jag kan hämta dem där och rita ut dem på skärmen. Först ville jag ha siffran noll på index noll, siffran ett på index ett och så vidare, men för att spara plats blev jag tvungen att ha både noll och ett på index noll, två och tre på index ett och så vidare. Det gjorde det lite krångligare att skriva ut siffrorna än vad det kunde ha varit. Detta är ett exempel på att ett färdigskrivet bibliotek till displayen inte hade fungerat. Där har varje tecken ett specifikt index.

Självavstängningen

Här blev elektroniken för avancerad för att jag skulle lösa det på egen hand. Jag fick ta hjälp av min handledare och en av hans kollegor för att reda ut det.

Tyvärr fungerade det inte helt perfekt även efter deras hjälp. Jag vet fortfarande inte exakt varför eller hur felet uppstår, men jag löste symptomet genom en ändring i mjukvaran. Problemet var att hårdvaran ibland startade om istället för att stänga av sig när jag skickade avstängningssignalen. Jag löste det genom att vänta med att skicka ”håll mig igång”-signalen till efter displayinitieringen. På så sätt startar inte spelet om det inte kommer en tillräckligt lång Vcc-puls.

Förbättringar

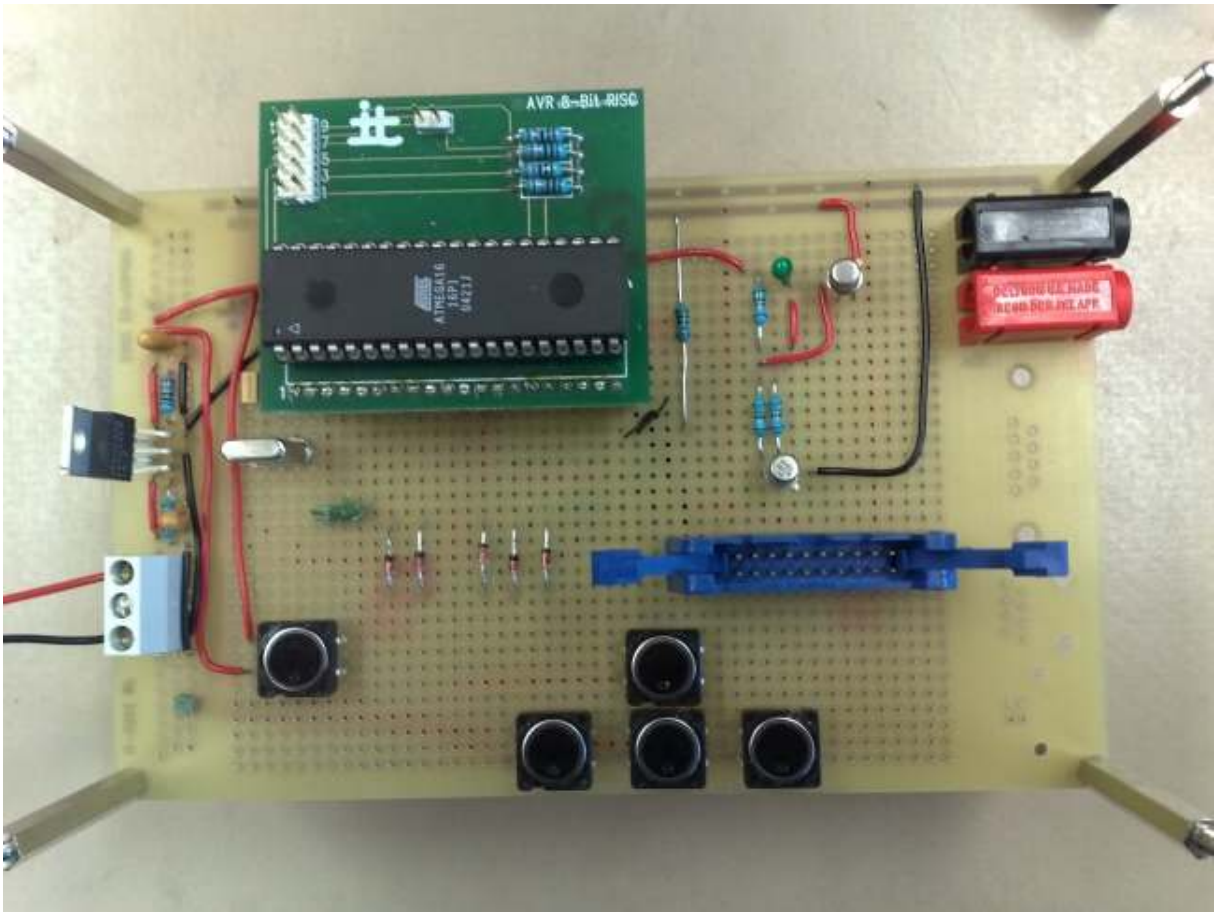
Den första förbättringen som borde göras är att låta användaren hålla inne en knapp för att till exempel snabbt flytta tetrisfiguren i sidled eller nedåt. Kan antagligen lösas genom att återaktivera knappavbrottet efter en viss tid i stället för när ingen knapp är nedtryckt.

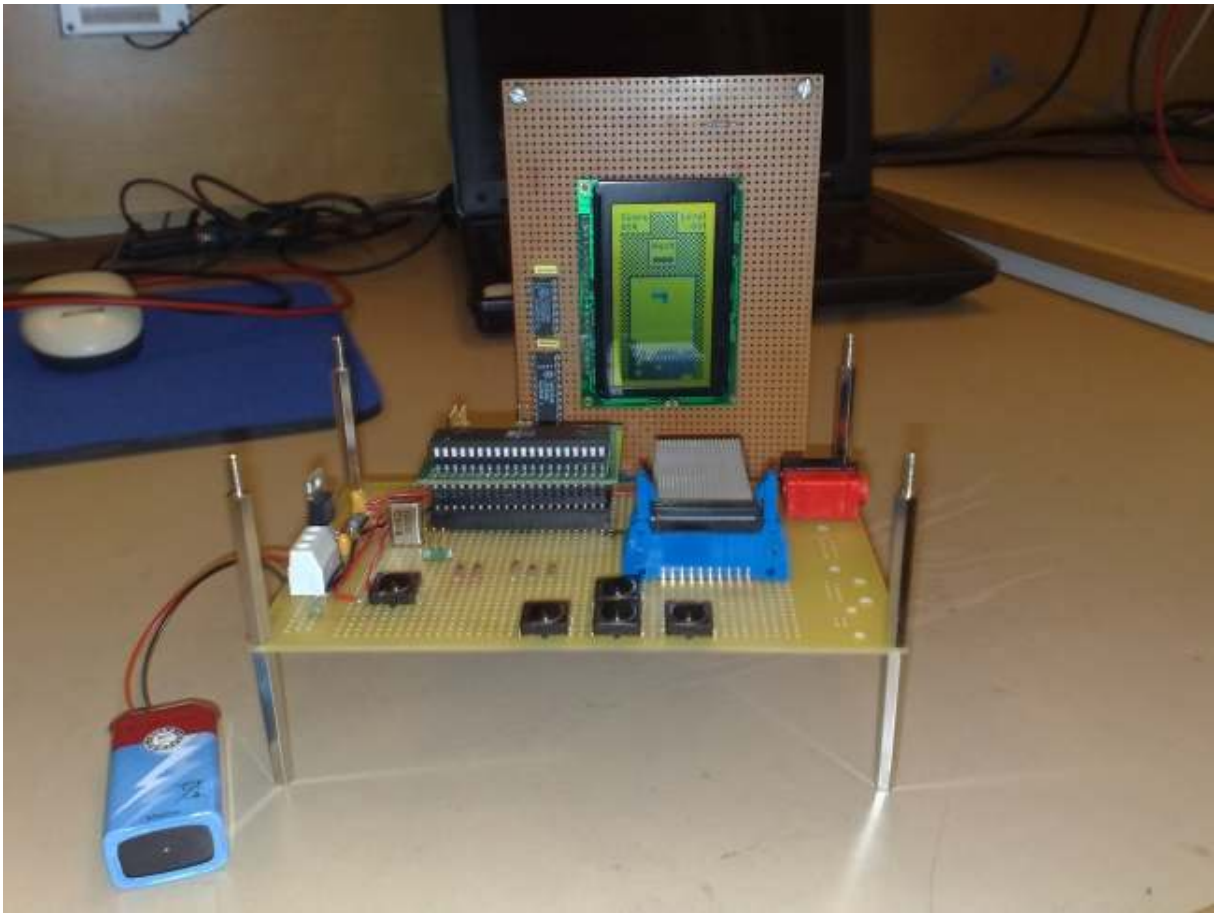
En annan sak som har stört mig lite är att ibland när en tetrisfigur inte kan flytta längre ner blir det en grafikbugg. Om buggen uppträder ritas ett tetrisblock i nederkant av tetrisfiguren till vänster. Jag vet inte vad det beror på, och blocket försvinner snabbt igen eftersom jag ritar om alla rader med tetrisblock när en tetrisfigur har placerats.

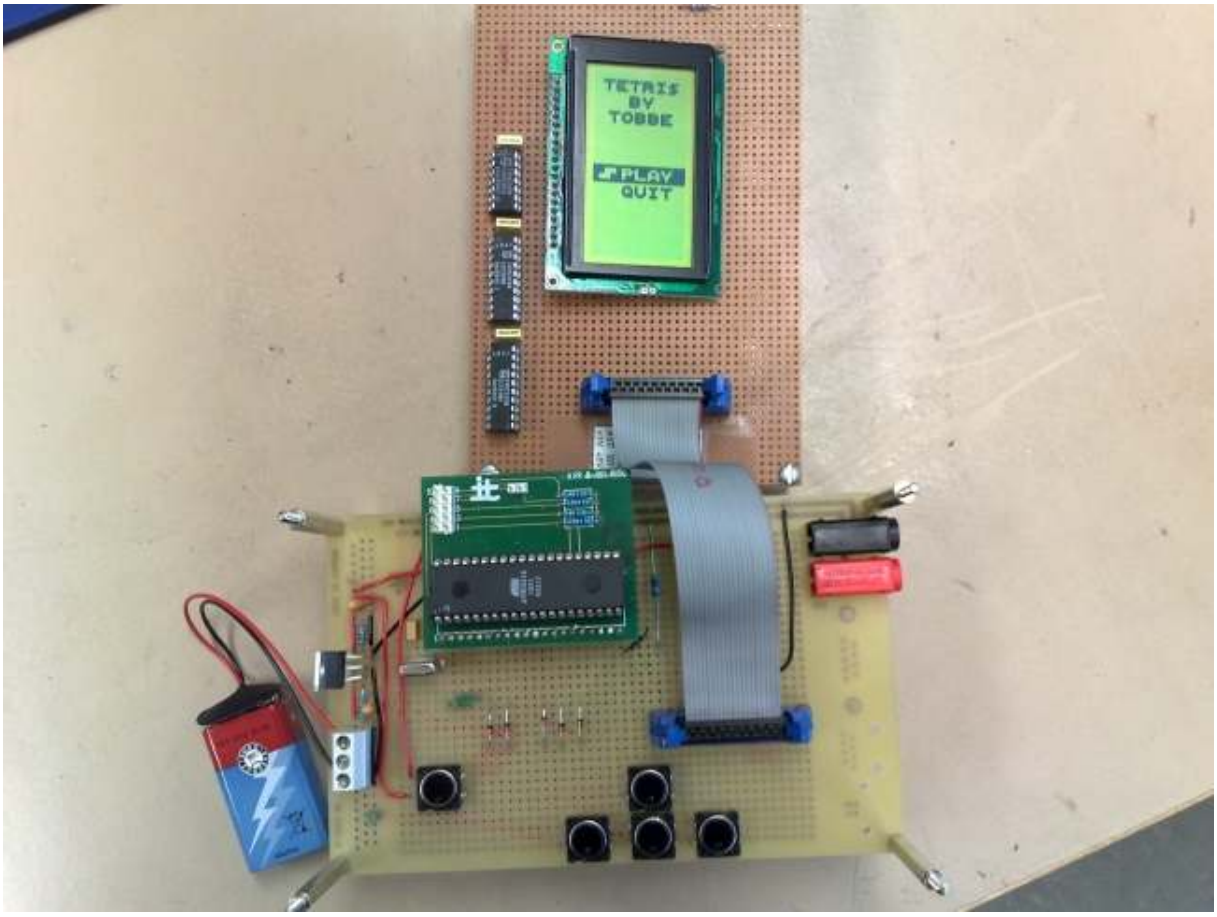
För att göra highscorelistan mer intressant skulle programmet som körs på Windows kunna fråga efter spelarens namn när ett nytt highscore har registrerats och spara det tillsammans med poängen i highscorelistan.

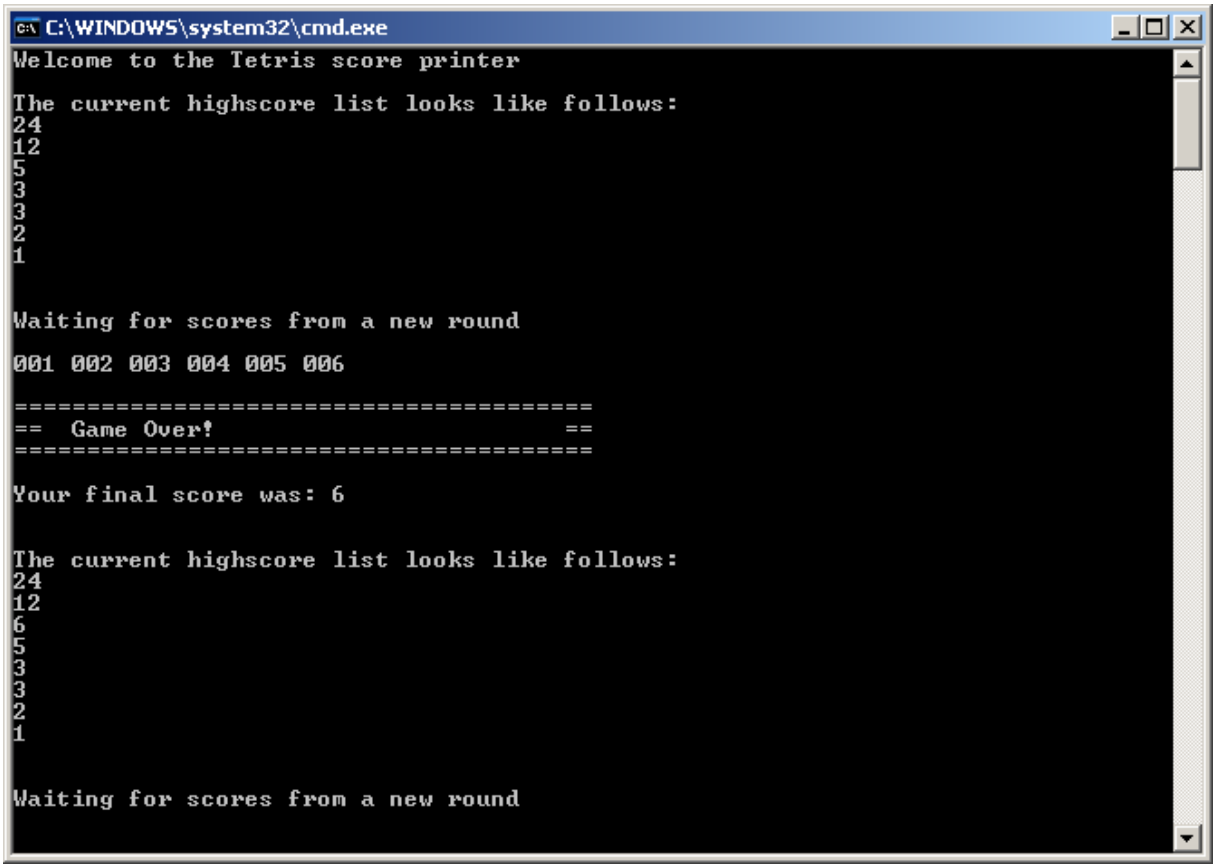
Som det är nu är tetriset helt klart portabelt, men om displayen flyttades upp på samma kort som resten av hårdvaran och allt placerades lite tätare skulle det kunna kallas handhållet.

Appendix A: Bilder

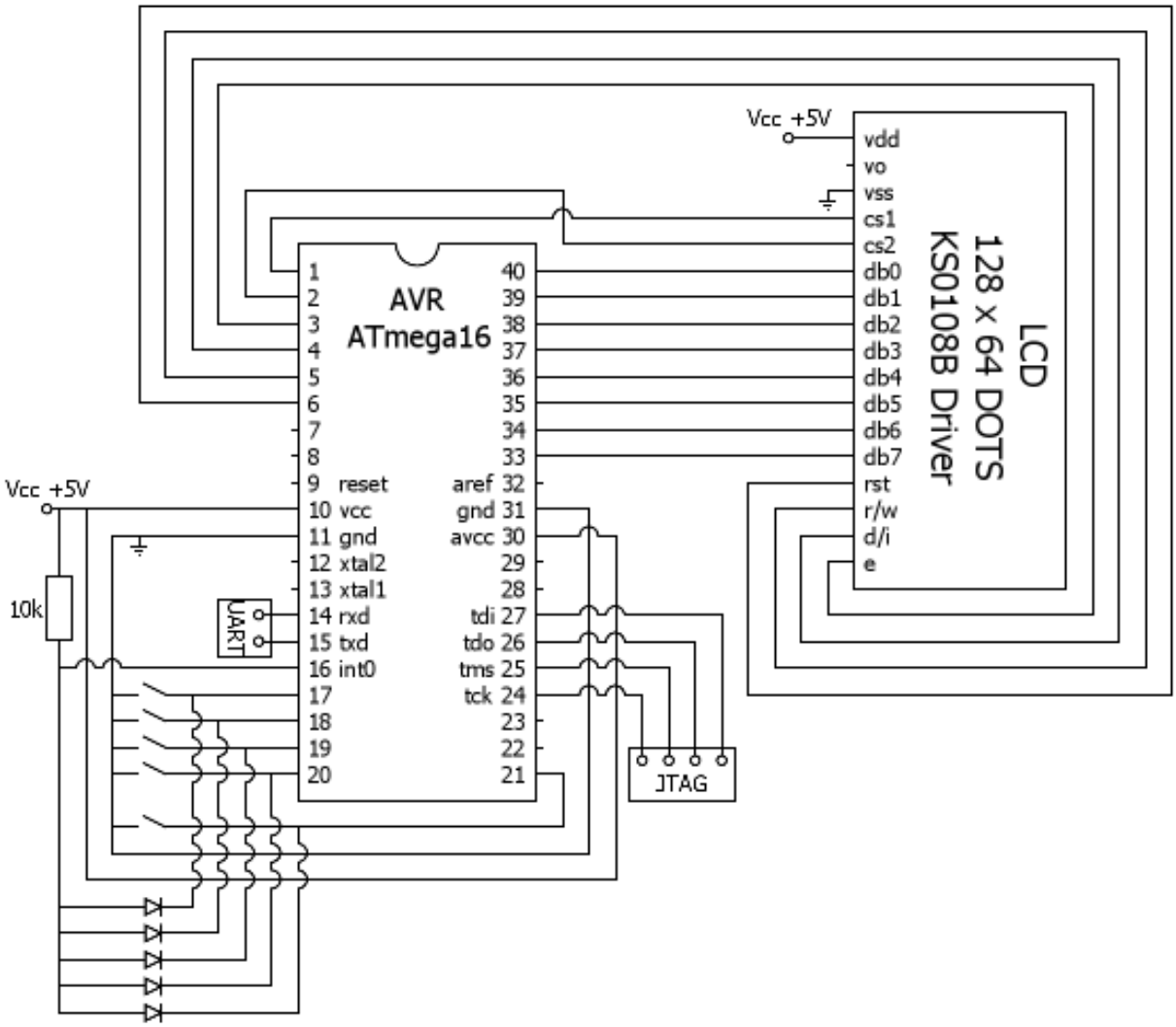




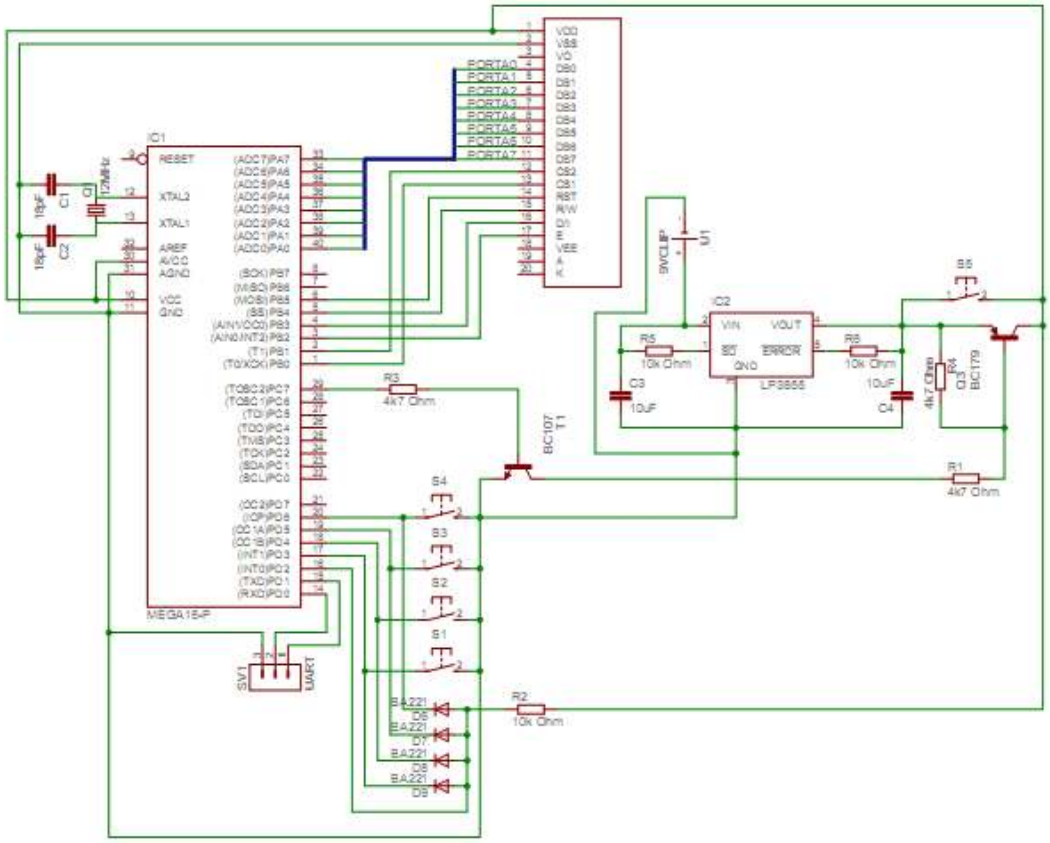

TETRIS
BY
TOBBE
PLAY
QUIT


C:\WINDOWS\system32\cmd.exe
Welcome to the Tetris score printer
The current highscore list looks like follows:
24
12
5
3
3
2
1
Waiting for scores from a new round
001 002 003 004 005 006
== Game Over!   ==
Your final score was: 6
The current highscore list looks like follows:
24
12
6
5
3
3
2
1
Waiting for scores from a new round

Appendix B: Blockscheman



Figur 2. Blockschema version 1



Figur 3. Blockschema slutgiltig version