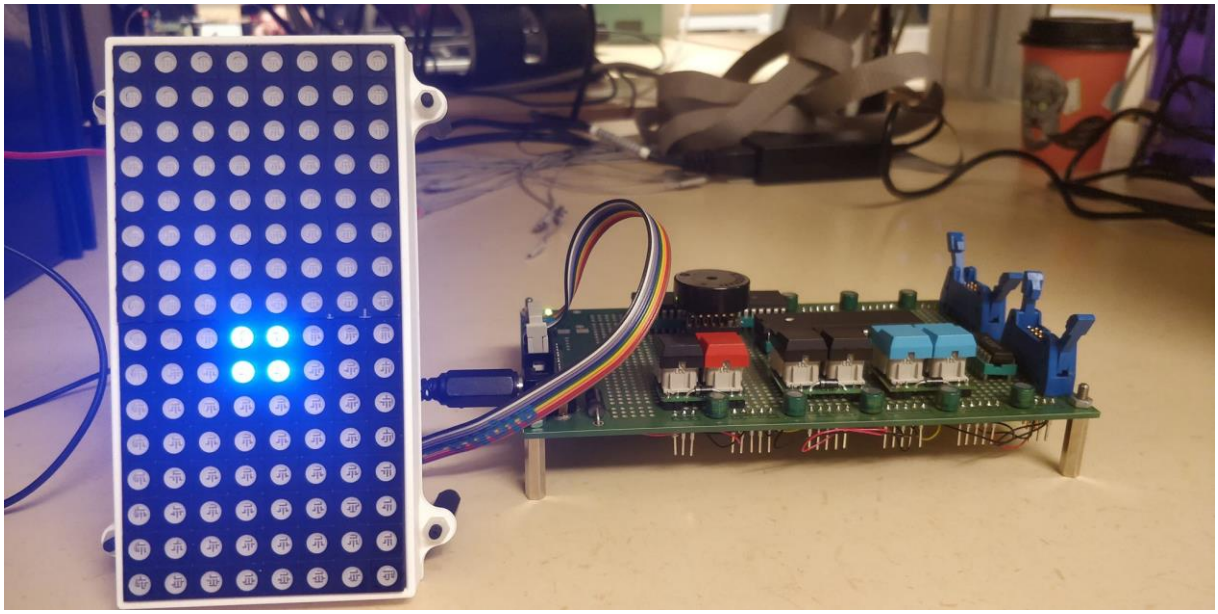
TETRIS



Mikrodatorprojekt Grupp 2

Samuel Nyman, Jesper Hilleberg, Svante Strid, Johanna Jönsson

Innehåll

1. Inledning

Följande rapport redovisar det projektarbete som grupp 2 i kursen, TSIU51 – Mikrodatorprojekt, har utfört för att konstruera ett Tetris med RGB-displayer, knappar och ATmega16A processorer.

Processorerna kallas i följande text även för P1 och P2, för processor 1 och processor 2.

2. Mål

Målet med kursen är att förläna kunskapen som behövs för planering samt konstruktion av en apparat i mindre digitala system. Apparaten ska använda sig av analoga komponenter som samverkar med den digitala miljön.

Kursen har för avsikt att ge kunskap och utveckla studentens förmåga att felsöka mikrokontrollerbaserade system, att arbeta i grupp samt att handskas med skriftlig respektive muntlig kommunikation.

3. Beskrivning av Tetris

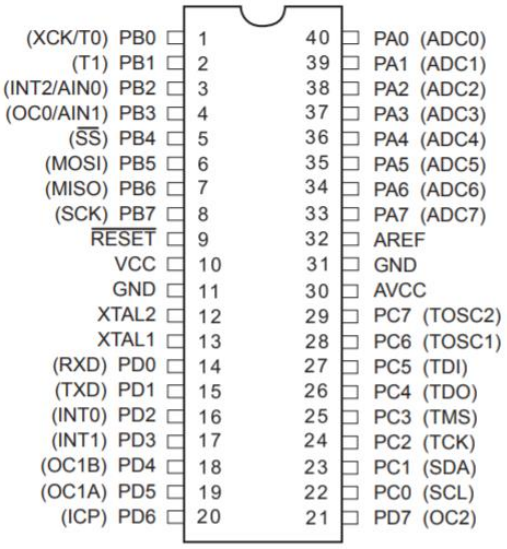
Tetris är ett datorspel som har funnits sedan 1985. Det är ett spel där slumpmässiga block faller ned till botten av en spelplan. Blocken är uppbyggda av 4 rutor i 7 olika former och spelet går ut på att överleva så länge som möjligt. När blocken fallit ned i den vanligen 10 gånger 20 rutor stora spelplanen så gäller det att skapa horisontella fulla rader genom att styra fallande block höger respektive vänster eller snurra det aktiva blocket för att passa in det så bra som möjligt. En full rad rensas och ovanstående fasta block flyttas ned ett steg. I takt med att spelet fortskrider ökar hastigheten i vilken blocken faller ned och gör spelet svårare och till sist omöjligt.

4. Hårdvara

Till vårt projekt har det använts flertalet komponenter. Ett av de krav som fanns i kursen var att använda sig av två processorer, som på något sätt ska kommunicera med varandra. Spelplanen är uppbyggd av två stycken på höjden sammankopplade RGB-displayer. För att styra blocken och nollställa används avstudsade knappar, för dess funktion tillkommer även en OR-grind. Musik och spelljud matas ut ur en högtalare.

4.1 Atmega16A

I projektet ingår två stycken ATmega16A processorer (se figur 1). Dessa har fyra huvudsakliga portar, A, B, C och D. Varje port har åtta pinnar, 0 – 7, som kan användas som in- eller utgångar individuellt. Port C är på en sådan processor reserverad för JTAG-uppkoppling, där PC2 - PC5 används. JTAG-en används sedan för att programmera processorn via programmet Atmel Studio 7.



Figur 1. Processorn ATmega16A. Källa: http://www.isy.liu.se/edu/kurs/TSIU02/5_Atmega16.pdf

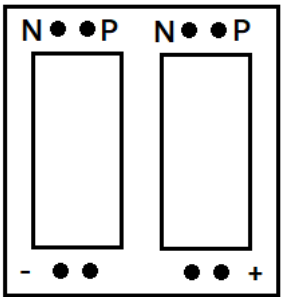
På samma sätt ligger funktionerna för SPI på port B, det som rör avbrott på port D och port A kan ta emot analoga data. I projektet har port A använts som en universal port för bland annat kommunikation mellan processorerna och som latch till RGB-displayen.

4.2 RGB-display

En RGB-display har måtten 8x8(bredd x höjd), alltså åtta lysdioder i varje riktning. I projektet har två stycken displayer använts för att skapa en RGB-display med måttet 8x16. Displayens funktion har varit att kommunicera med videominnet i P1 genom SPI och agerat som spelskärm till spelet, alltså genom att visuellt visa alla block som kodats.

4.3 Avstudsad Tryckknappmodul

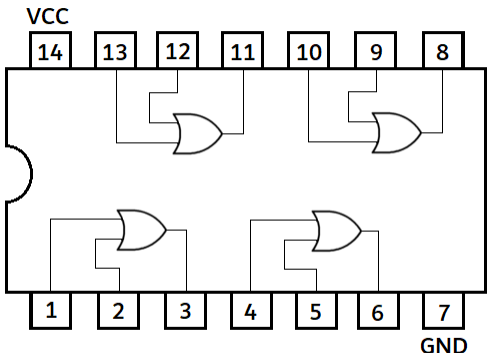
Tryckknappsmodulerna, vardera med två knappar på, används det tre stycken av. Totalt blev det sex stycken tryckknappar. Varje separat knapp har en positiv och en negativ flank. I projektet har endast den positiva flanken använts. Användningsområdet för knappar har varit: återställningsfunktion, styrning åt höger och vänster, samt rotation och sänkning.



Figur 2. Avstudsad tryckknappsmodul.

4.4 OR-grind

OR-grinden som nyttjas är 74LS32 och funktionen den haft i projektet är att notera när en knappnedtryckning har skett och då skapa ett avbrott. Fyra kablar har kopplats till grinden och sedan två inom grinden för att sedan ha en utgångskabel tillbaka till processorn, se Konstruktion för vidare information.



Figur 3. 74LS32, OR-grind.

4.5 Högtalare

I projektet hade högtalaren två funktioner, att ge ljudeffekter vid rörelse och att konstant spela musik. Ljudgenerering är konstruerad genom att ge högtalaren en sviktande hög och låg signal, vilket gör att membranet i högtalaren vibrerar och skapar ett ljud. Ljudet ändras när längden som högtalaren får en hög respektive låg signal ändras.

4.6 SPI

SPI, också kallat Serial Peripheral Interface har använts som kommunikation mellan Processor 1 och RGB-displayerna. SPI innehåller två stycken shift register som kallas MASTER och SLAVE. I projektet har P1 blivit given registret MASTER och RGB-displayerna registret SLAVE. För att överföra information till en RGB-display behöver fyra bytes, 32 bitar, överföras. Men eftersom arbetet sker mellan två stycken

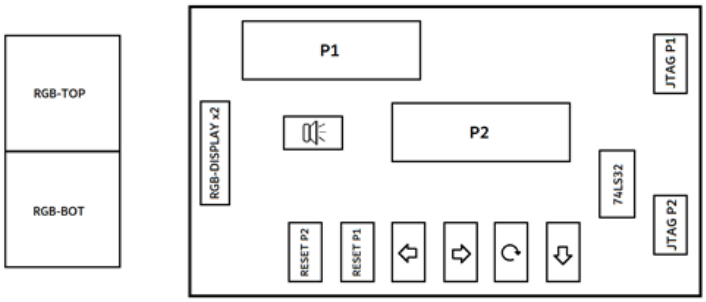
displayer så skickas åtta bytes, 64 bitar. Informationen skickar genom SPI är vilken anod som ska tändas och hur starkt det röda/gröna/blåa ljuset ska lysa i anoden.

Figur 4. Övre rad: Hur RGB-displayen tar emot värden från vänster till höger. Nedre raden: Hur värden seriellt skickas till RGB-displayen.

Eftersom inmatningen sker seriellt så skickas denna information in i fel ordning, alltså istället för att skicka in informationen i ordningen anod, röd, grön, blå så skickas den in i ordningen blå, grön, röd, anod (se figur 4).

5. Konstruktion

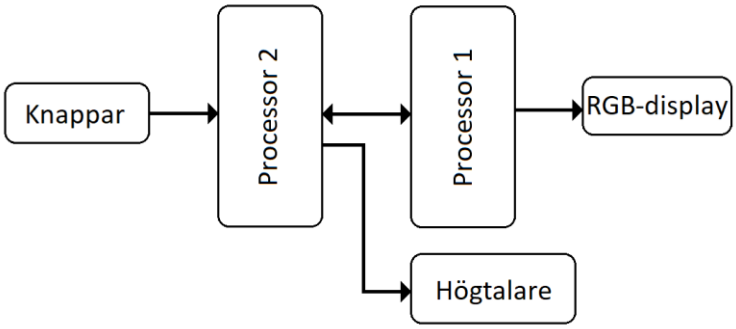
Hårdvaran är placerad enligt figur 5.



Figur 5. Placering av komponenter.

RGB-TOP och RGB-BOT är vardera en RGB-matris med storlek 8x8. RGB-DISPLAY är porten som kopplade kortet till displayen. P1 och P2 står för de två processorerna och knapparnas funktion i speglas i texten/symbolen på respektive knapp. JTAG:arna är placerade åt sidan för enkel åtkomst vid programmering.

Kort sagt är knapparna och högtalaren kopplade till processor 2 och RGB-displayen till Processor 1 (se figur 6). Pilarna indikerar åt vilket håll signalerna går från respektive komponent.

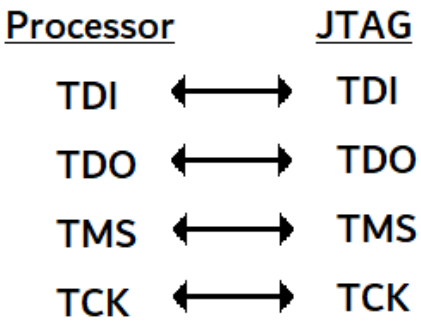


Figur 6. Översikt av uppkopplingen.

Konstruktionen är uppkopplad enligt kretsschema (se bilaga 12.1), där processorerna har varsin resetknapp och JTAG-koppling. Detta för att kunna starta om processorerna närsomhelst och för att kunna programmera dem separat, utan att behöva dra om kablar för att byta processor. Alla komponenter är jordade och kopplade till VCC.

Varje processor är sedan kopplad till de komponenter de ansvarar för. P1 är därmed kopplad till RGB-displayen via B-portens MOSI (Master Out, Slave In), PB5. Genom denna port skickas alla signaler från P1 till displayen. PB7, SCK (Serial Clock), är satt som utgång till displayen. Latchen till RGB-displayen krävde ingen specifik placering och för projektet valdes PA1 till detta för att hålla den ur vägen.

Gemensamt för de båda processorerna är reset-knapp, pinne 9, och JTAG-kopplingen. PC2 - PC5 används för JTAG:en, där Test Data In (TDI) på JTAG:en går till TDI på processorn och detsamma gäller för Test Data Out (TDO), Test Mode Select (TMS) och Test Clock (TCK).



Figur 7. Koppling mellan processor och JTAG.

Kommunikationen mellan processorerna använder portarna PD0 - PD4 samt PD6 på processor 1 och PA0 - PA5 på processor 2. Se kommunikation nedan för vidare information.

P2 är i sin tur kopplad till högtalaren, tryckknapparna och en OR-grind, som även den är kopplad till knapparna. Signalen till högtalaren utgår från PA6. Knapparna är kopplade från sin positiva flank till PB0 - PB3.

OR-grinden som triggar ett avbrott på processor 2 är också kopplad till knapparnas positiva flank. Grinden avgör om någon av knapparna är nedtryckt genom att hela tiden

Ingen knapp nedtryckt: ((0 + 0) + (0 + 0)) = 0

Knapp nedtryckt: ((1 + 0) + (0 + 0)) = 1

Figur 8. OR-grindens funktion.

utföra logiskt ELLER på signalerna från knapparna. Dessa kommer vara 0 fram tills dess att en knapp trycks ner. OR-grinden skickar vidare resultatet till PD2, INT0 (External Interrupt 0 Input).

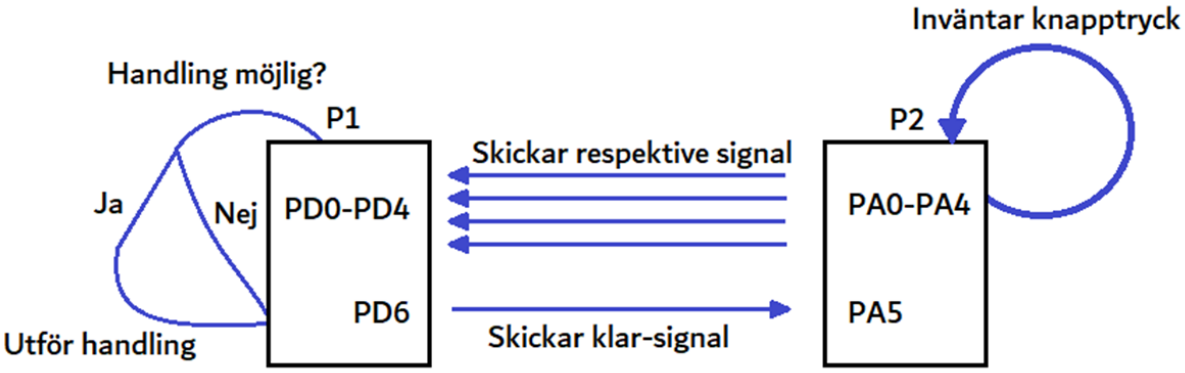
6. Kommunikation

6.1 Uppkoppling

Kommunikationen mellan de två processorerna har behandlats genom att direkt koppla sex kablar mellan dem. P2 har fem kablar med syftet att skicka ut höga och låga signaler, sedan också en kabel som ansvarar för att ta in information från den andra processorn. De portar som använts för detta är PA0 – PA5 där PA0 – PA4 är utgångar och PA5 är en ingång. P1 har då alltså fem ingångs kablar och en utgångskabel, PD0 – PD4 är ingångar och PD6 är den enda utgången. En signal skickas genom att ändra en pinnes värde till ett respektive noll.

6.2 Funktion

Informationen som P2 skickar till P1 handlar om rörelse. Det är P2 som hanterar signalera för vilken knapp som blivit nedtryckt då detta behöver ske i ett avbrott och eftersom P1 redan har två avbrott så blir det riskfyllt att ha ett till.

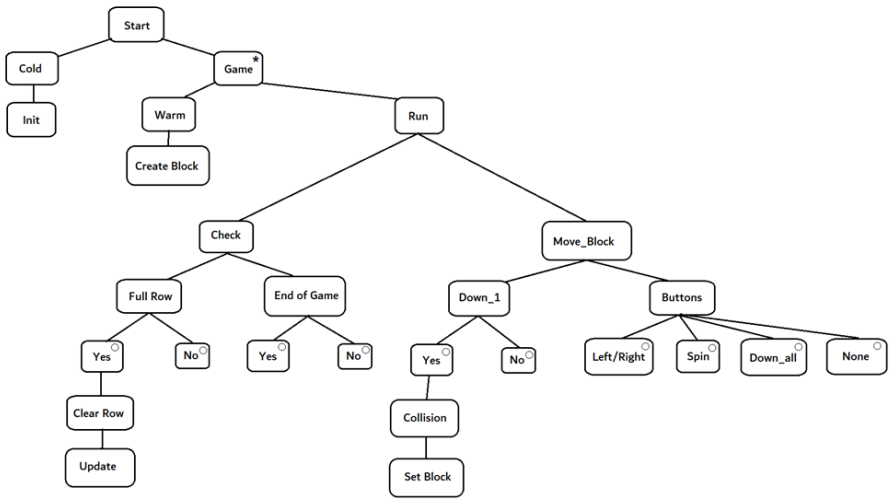


Figur 9. Illustration av kommunikationen mellan processorerna.

När P1 sedan fått informationen av P2 så skickar P1 sin signal till P2 för att berätta att P1 jobbar. Under denna tid väntar P2 på ytterligare en signal från P1 som berättar att jobbet är klart. När P1 är klar så nollställer P2 styrningssignalen och inväntar istället en ny så denna process kan repetera (se figur 9).

7. Processor 1

Processor 1 ansvarade för spelet och dess interna funktioner. Figur 10 beskriver vad som händer i programmet.



*Figur 10. Tankeprocess för P1. Ruta med * betyder att programmet upprepar segmentet. Ruta med o innebär att det sker ett val.*

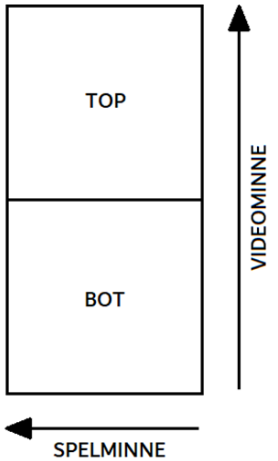
Programmet börjar alltså med att initiera all hårdvara, avbrott och klockor för dessa. Därefter skapas ett block, som det hela tiden sker kontroller på, innan det avgörs om blocket ska byta position. I figur 10 står Down_1 för att blocket ska falla ett steg. Sist flyttas blocket om processorn fått signal från P2. När ett block har fastnat börjar programmet om i Game.

7.1 Videominnet

Videominnet är två 24 byte sparade minnen som innehåller värden för var färgerna ska lysa, ett minne för varje RGB display. Varje byte har ansvar för en rad i lodrät led för en färg. Detta minne används för att skriva ut till displayen via muxen.

7.2 Spelminnet

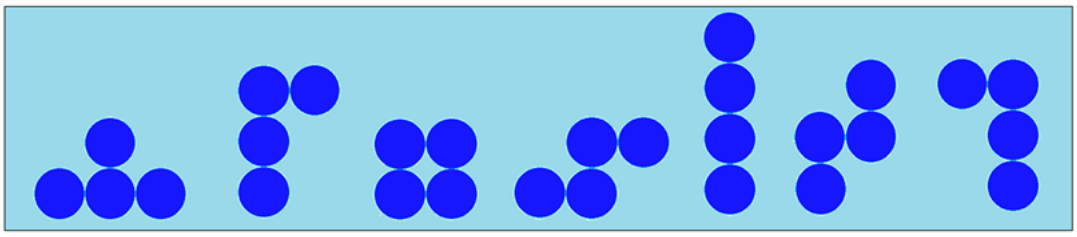
Spelminnet är två 8 byte sparade minnen där en byte håller för en rad i vågrät led. En etta i dessa byte betyder att där är ett block vid den positionen. Detta minne används för att, inne i processorn, kunna testa om olika funktioner är möjliga.



Figur 11. Spelminnet och Videominnets riktning på RGB-displayen.

7.3 Blocken

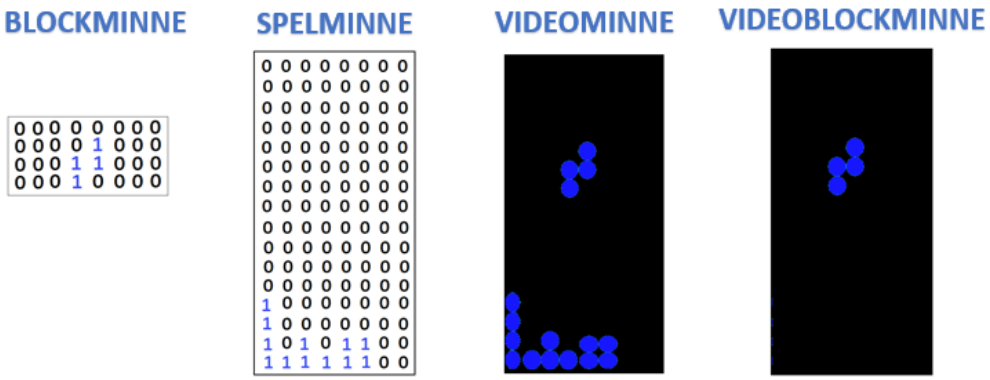
Vi använder de klassiska sju blocken i vårt tetris, som syns i bilden ovan, med färgen som enda skillnad. Alla block är blå och består av fyra lampor som arrangeras på olika sätt för att skapa de olika blockens former.



Figur 12. De sju block som ingår i projektet.

7.3.1 Lagring Av Block

Blocken är lagrade i både ett 4 bytes blockminne och ett helt 24 bytes videoblockminne. Detta för att blockminnet ska kunna kolla av med spelminnet om olika funktioner kan utföras, medan videoblockminnet ska veta vilken del av videominnet som blocket är i och kunna ändra position på det.



Figur 13. Minnena och deras innehåll.

7.3.2 Skapa Block

Blocken skapas från en tabell, `BLOCK_REGISTER`, som innehåller de olika blockens värden. Dessa värden sätts sedan in i blockminnet och lägger dem tre steg åt vänster för att de ska börja i mitten (se figur 13, blockminne). `DOWN_INDEX` sätts till $00 eftersom toppen har nåtts och `SIDE_INDEX` sätts till $03 eftersom tre steg åt vänster har skett.

Efter att blockminnet är klart måste blocket i videoblockminnet skapas. Detta görs när blocket ska åka ner genom att vänster shifta blockminnet. Om carryn är satt för något

av registren som ingår i blockminnet så läggs det till en etta i videoblockminnet på den raden som funktionen befinner sig är på och går vidare till nästa. När blocket gått ner fyra steg så kallas denna funktion inte längre förrän ett nytt block ska skapas.

```
BLOCK_REGISTER:
    .db /*Block 1, $00*/$03, $03, $00,/*Block 2, $03*/ $03, $06, $00, $02, $03, $01, $03, $06, $00, $02, $03, $01,
    ./*Block 3, $0F*/ $06, $03, $00, $01, $03, $02, $06, $03, $00, $01, $03, $02,
   /*Block 4, $1B*/ $07, $01, $00, $01, $01, $03, $04, $07, $00, $03, $02, $02, /*Block 5, $1B*/ $07, $04, $00, $02, $02, $03, $01, $07, $00, $03, $01, $01,
   /*Block 6, $26*/ $02, $03, $02, $02, $07, $00, $02, $06, $02, $07, $02, $00, /*Block 7, $32*/ $0F, $00, $00, $00, $04, $04, $04, $04
```

Figur 14. BLOCK_REGISTER. Lagring av blocken. Återfinns i koden från P1, se bilaga 12.2.

7.4 Muxen

Muxen är en funktion som ligger i ett avbrott som händer så snabbt det bara går. Muxen tar in värdena från videominnet och skickar in dem via SPI. I varje avbrott tar den ett videominnes värde eller ett anodvärde i ordningen blå, grön, röd, anod. Detta gör den åtta gånger, fyra per LED platta. Efter åtta gånger så skickas en etta till LED plattans latch som skickar ut värdena på skärmen och nästa avbrott börjar då på nästa rad.

7.5 Funktioner

Spelets funktioner styrs av interna avbrott, knapparna samt blockens position. Funktionerna som triggas utav knappar ligger i GAME_ON_BUTTONS. Här tar P1 emot signalerna från P2 och går sedan vidare till respektive funktion.

7.5.1 Fall

Funktionen för fallet ligger inte i ett avbrott men det finns ett avbrott som lägger ett värde i ett index för att fall-funktionen ska veta när den ska aktiveras. Detta gjordes för att fall-funktionen inte skulle gå igång mitt i en annan funktion. I fall-funktionen kollar man först om något utav blocken kommer kollidera om ännu en nedstigning sker eller om botten är nådd, i så fall låser sig blocket i positionen där den redan är. Annars så sätts block down index till ett mer och videoblockminnet plockar ut blocket från videominnet och stegar ner det ett steg och sen sätter tillbaka det igen. Det finns även en knapp som kallar denna funktion tills dess att blocket fastnar.

7.5.2 Flytt Höger / Vänster

Flytt åt höger och vänster fungerar i princip på samma sätt, men åt olika håll. Det aktiva blockets värden läses in i ett par register och det första som testas är om det är möjligt

att flytta blocket. Åt höger innebär detta att carryn kollas för att blocket ska bli kvar på spelplanen. Åt vänster kontrolleras istället om bit 7 är satt, vilket innebär att blocket redan ligger mot väggen. I båda fall kontrolleras om det redan ligger något på platsen blocket vill flyttas till i spelminnet.

7.5.3 Radrensning

Varje gång efter att ett block har lagts ner så kollas varje rad i spelminnet om det är fullt, alltså att ett minne är $FF. Om detta har skett så kallas en shift funktion som plockar ut den valda raden, sätter alla rader att hamna ett nedanför och göra det översta tomt. Detta görs även i videominnet. Efter att den gjort detta kollar den igenom allt igen för att se så ingen mer rad är tom.

7.5.4 Förlust

Om ett block fastnar längst upp så kan man inte längre köra ner blocken och måste köra reset för då har man helt enkelt förlorat.

8. Processor 2

Processor två hade i uppgift att konstant spela upp musik, samt att läsa in tryckknapparna och sända den informationen vidare till processor ett. Själva huvudprogrammet består av musikuppspelningen medan knappinläsningen sker i ett avbrott.

8.1 Musik

8.1.1 Toner

Projektet använder sig utav en fast amplitud, då signalen till högtalaren enbart sätts hög eller låg, det vill säga ett eller noll. Därefter används tonernas specifika våglängder och frekvenser för att hitta ett acceptabelt förhållande mellan tonens faktiska ljud och det projektet åstadkommer.

Toner har en specifik våglängd och en frekvens som tillsammans bestämmer hur ljudet som spelas upp beter sig. I projektet blir detta samband att signalen till högtalaren sätts hög en halv våglängd, sedan låg en halv våglängd. Tillsammans bildar detta tonens hela våglängd och detta upprepas sedan frekvensens antal gånger.

8.1.2 Lagring av Noterna

Tetrismelodin som spelas medan spelet är igång består i projektet av ett antal noter som repeteras om och om igen. För att spela upp en not behövs dess frekvens och våglängd. Dessa lagras i två separata tabeller, `FREQUENCY` och `HALF_WAVE` med åtkomst via ASCII-tecken. Observera att våglängden är halverad endast för att underlätta skifte av hög och låg signal.

Frekvensen är i enheten Hertz (Hz) och sedan översatt till hexadecimala tal. Våglängden är i enheten meter(m) och halverad innan, även den översatts till hexadecimala tal. Exempelvis har nu noten A2 frekvensen $6E Hz och den halva våglängden $0F m (se figur 15). Tecknet som representerar A2 är “A”. Noter representeras av tecknen “=” till “R” (se bilaga 12.3, sida 52).

Not	A_2		B_2	
Frekvens	110.0 Hz	$6E	123.0 Hz	$7B
Halv våglängd	15.0 m	$0F	16.0	$0E

Figur 15. Exempel på hur toners frekvens och våglängd översatts.

```
MELODY:
    .db "A<>:?:@<?:>:=<=:?:A<@:?:><>:?:@<A<?<=<=<<:@@:<B:D<C:B:A<:?:A<@:?:><>:?:@<A<?<=<=:::" ,$00
```

Figur 16. Noterna för melodin, skriven i ASCII-tecken.

Själva melodin skrivs sedan helt i ASCII-tecken efter vilka noter som ska spelas upp (se figur 16).

Tecknen “9”, “:”, “;” och “<” är inte noter utan tysta mellanrum för att göra uppspelningen tydligare. Dessa förekommer i båda tabellerna för att pekaren ska hamna rätt, men har bara en funktion i frekvenstabellen, där de utmärker olika långa mellanrum. Dessa har lagts in på gehör och förekommer ibland flera på rad.

8.1.3 Spela Upp Melodin

Ett antal kortare subrutiner hanterar sedan hela ljuduppspelningen. Följande text beskriver hur programmet hanterar den första noten, A2.

Z-pekaren laddas med melodins tabell första gången i `INIT` och pekar därmed på det första ASCII-tecknet, som i detta fall är “A”. Från `MAIN` går funktionen vidare till `GET_NOTE` som hämtar värdet av ASCII-tecknet. För “A” blir detta då $41 och värdet läggs i registren r16 och r17. Pekaren flyttas också till nästa tecken inför nästa not. Därefter jämförs värdet med $00 för att veta om melodin är slut eller inte. Skulle detta stämma väntar programmet kort innan Z-pekaren på nytt laddas med melodin, varpå programmet fortsätter.

I `F_LOOKUP` lokaliseras värdet av r16, tecknet “A”, i tabellen för frekvens och dess motsvarighet, $6E, tar dess plats i r16. På samma sätt ersätter `W_LOOKUP` värdet av “A” i r17 med den halva våglängden, $0F.

Registren r16 och r17 innehar nu de värden som behövs för att spela upp noten. Funktionen `SOUND` börjar med att avgöra om ljud ska spelas upp, eller om tecknet har en frekvens lägre än $49, vilket i tabellen innebär att tecknet i fråga är något av de tecken som representerar ett mellanrum. Eftersom tecknet var “A” hoppar funktionen vidare till `TONE` som kallar på funktionen `BEEP`.

Det första som görs i `BEEP` är att den halva våglängden i r17 kopieras till r18, då den ska användas flera gånger. Därefter kallas funktionen `HIGH_SIG` som sätter pinne 34, som går till högtalaren, hög och därefter kallar på en delay som räknar ner den halva våglängden i registret r18. Tillbaka i `BEEP` kopieras den halva våglängden återigen från r17 till r18 för att användas igen och `LOW_SIG` kallas. `LOW_SIG` gör precis likadant som `HIGH_SIG` med den skillnad att signalen ut är låg istället för hög. I `BEEP` minskas sedan frekvensen i r16 med $01 och funktionen repeteras tills r16 når $00.

Därefter är funktionen `SOUND` klar, noten är uppspelad och funktionen returnerar tillbaka till `MAIN`. Det sista som händer i `MAIN` är ett hopp tillbaka till `MAIN`. Nästa gång programmet kommer in i `GET_NOTE` står pekaren alltså på nästa tecken, första efter "A" är "<". Detta tecken är då inte en not, vilket framkommer när r16 fått dess värde ur frekvenstabellen, som då inte är en riktig frekvens utan bara en önskad längd av mellanrummet.

Mellanrummen hanteras på ett liknande sätt till noterna, men den halva våglängden ignoreras och är i tabellen satt till $00. Ett mellanrum påverkar koden först i `SOUND`. Där är nu "frekvensen" lägre än $49 och istället för att hoppa till `TONE` går nu programmet in i `PAUSE`, vars enda funktion är att vara tyst via `NOBEEP`, som helt enkelt laddar r18 med konstanten `TYST`, som är decimalt 165, varpå `LOW_SIG` kallas. Detta görs två gånger för att åstadkomma en hel våglängd och hela `NOBEEP` upprepas tills r16, med längden på mellanrummet, når $00.

8.2 Knappinläsning

Knappnedtryck triggar ett avbrott eftersom alla tryckknapparna är kopplade till varsin pinne på en OR-grind, sedan är grindens utsignal kopplad till INT0 på P2. När ett avbrott har triggats tillåts inga andra avbrott, alltså inget mer knappnedtryck, förrän funktionen inom avbrottet är åtgärdad. Detta avbrott behövs eftersom om knappnedtryckskollen finns i huvudprogrammet pausas hela programmet medan en knappnedtryckning sker och eftersom P2 också spelar musik så blir avbrottet kritiskt att ha.

8.2.1 Inläsning

Tolkningen av knapparna sker i avbrottet MOVEMENT. Alla berörda register pushas och SREG sparas undan. Då endast pinne 1-4 är kopplade till knappar utförs logiskt AND med $0F för att endast behålla värdena av knapparna. Redan här kontrolleras om en knapp faktiskt är nedtryckt genom jämförelse med $00. Skulle fallet inte vara så hoppar programmet till slutet av MOVEMENT, DONE_DONE för att lägga tillbaka SREG och poppa alla pushade register.

Är en knapp nedtryckt tar MOVEMENT reda på detta genom att jämföra med värden för respektive pinne (se figur 17). När rätt knapp identifierats hoppar programmet till dess funktion, LEFT, RIGHT, ROTATE och DOWN. Skulle av någon anledning inget av värdena stämma, exempelvis att mer än en knapp är nedtryckt, hoppar programmet till DONE_DONE.

```
cpi r16, $01                ;B0
breq LEFT
cpi r16, $02                ;B1
breq RIGHT
cpi r16, $04                ;B2
breq ROTATE
cpi r16, $08                ;B3
breq DOWN
rjmp DONE_DONE
```

Figur 17. Knappavläsning

I funktionerna LEFT, RIGHT, ROTATE och DOWN sätts sedan respektive pinne kopplad till P1 hög och ljudeffekterna kallas. Det allra sista som händer är att CHECK_PINA kallas. Denna funktions syfte är att kontrollera om den andra processorn tagit emot signalen och programmet stannar här tills klarsignalen från P1 är mottagen. Därefter rensas den pinnen och avbrottet avslutas med DONE_DONE.

8.2.2 Ljudeffekt

Varje knapp triggar även en ljudeffekt, som spelas upp även om det inte går att utföra rörelsen. De fungerar då som en bekräftelse på att en knapp blivit nedtryckt, så om ett block står längst till vänster och spelaren vill flytta blocket vidare åt vänster registreras fortfarande knapptrycket, även om blocket inte flyttas.

I detta tetris används två olika ljudeffekter. En för flytt i sidled, alltså vänster och höger, samt en för rotation och att slå ner block. Funktionerna MOVE_SOUNDEFFECT och DROPSPIN_SOUNDEFFECT hanterar detta. Dessa sker i avbrottet nämnt ovan och

laddar helt enkelt r16 och r17 med önskade värden beroende på vilket ljud som önskas, varpå `SOUND` sedan står för själva uppspelningen.

9. Slutsats

De sex veckor som vi arbetade på projektet resulterade i ett fungerande spel. Block skapades slumpmässigt enligt de specifikationer vi angett och funktioner som radrensning, rotation samt flytt av block implementeras med. Komplett till alla funktioner med tillhörande musik. Med hjälp av avbrott har vi även musik samt ett spel som blir svårare allt efter som att tiden går.

Koden är även uppbyggd på ett sådant sätt att det går att tillämpa färger utan större ändringar. I detta skede uppstår små buggar i skapandet av olikfärgade blocket som för med att andra funktioner också lider. Rotationen är en sådan funktion. Detta var sekundärt i vår kravspecifikation och prioriterades bort i mån av tid.

10. Diskussion

Det har varit otroligt lärorikt att som student skapa en apparat från grunden. De mål som är satta för kursen har en tydlig koppling till hur arbetet som en ingenjör kan komma att se ut. Genom både framgångar som motgångar i projektet så har vi utvecklats inom flera områden som problemsökning, kodande av processorer, RGB-displayer och samarbete för att nämna några.

Diskussionen kommer innehålla tankar kring projektarbetet i helhet, hjälpmedel som vi använt oss utav under arbetets gång, motgångar vi påträffat samt lärdomar som vi kan ta med oss i framtida arbeten.

10.1 Verktyg

Det var vid flertalet tillfällen vi fann varandra vid whiteboarden för att förklara vad vi gjort eller för att komma på ett nytt, bättre tillvägagångssätt och det var till en stor fördel när vi jobbade i olika områden av projektet. Visuell framställning och kommunikation var två viktiga verktyg och whiteboarden ett viktigt media. Ett annat viktigt verktyg var logikanalysatorn som fanns på plats. Tillsammans med en multimeter var de vitala instrument i fysiska konstruktionen och gjorde det lättare att felsöka.

Vi gjorde ett smart val att från början bygga upp spelets grund totalt innan vi gick vidare. Detta gjorde att vi sedan kunde lösa alla funktioner och buggar förhållandevis enkelt. Det hindrade en massa onödiga buggar så vi slapp skriva om delar av tidigare kod som inte annars skulle fungera med ny.

10.2 Motgångar

Även om grunden att ha ett video- och spelminne var en otroligt skönt att hantera i koden så blev den mycket svårare, eftersom vi med SPI var tvungna att göra videominnet i lodräta rader men i spelminnet behövde vi vågräta rader. Detta var något som gick att hantera men krävde mycket mer komplexa och långa lösningar.

Uppdelningen för arbetet av projektet var inte jämnt fördelat och procentuellt så blev mer arbete nedlagt av vissa än av andra. Detta var dock inget kritiskt eftersom alla ändå arbetade, men kanske inte lika mycket.

JTAG:en fungerade inte när vi först fick den vilket försvårade arbetet eftersom vi precis startat och redan var överväldigade av det som låg framför oss. Det tog en stund innan vi förstod att den var trasig och spenderade för mycket tid på det.

SPI var något som tog lång tid för oss att förstå då det var relativt nytt för oss att lära från datablad och den informationen som fanns i databladet ansåg vi ej tillräcklig så efter hjälp av handledare fick vi den att fungera korrekt.

Tidspress var något vi led utav då vi i slutet fick sitta mer timmar än vi ville, detta var heller inget enormt men det var en liten nackdel.

10.3 Lärdomar

Efter projektet lärde vi oss att planering är något av det viktigaste i början av ett projekt, även att vi planerade lite i början av vårt projekt så var det långt ifrån tillräckligt. Detta ledde till en ojämn fördelning av arbetet samt tider då vi var osäkra på vad som vi behövde göra.

Vi lärde oss mycket gällande konstruktionen av arbetet vilket inkluderar: uppkoppling, uppbyggnad och vårdnad. Kopplingen av kablarna var inte svårt men i efterhand så kunde vi ha planerat längden på kablarna bättre för att göra det mindre stökigt. Uppbyggnaden av projektet var nytt för oss alla men vi lärde oss snabbt att handskas med det. Vårdnaden av hårdvaran skötte vi bra men det fanns få incidenter då det var ömtåligt och vi var tvungna att anpassa oss till det genom att byta ut eller kontrollera så allt verkligen fungerade som det skulle.

Våra programmeringskunskaper ökade tillsammans med problemlösningen, vilket skedde naturligt under arbetets gång då projektet översteg 1000 rader kod.

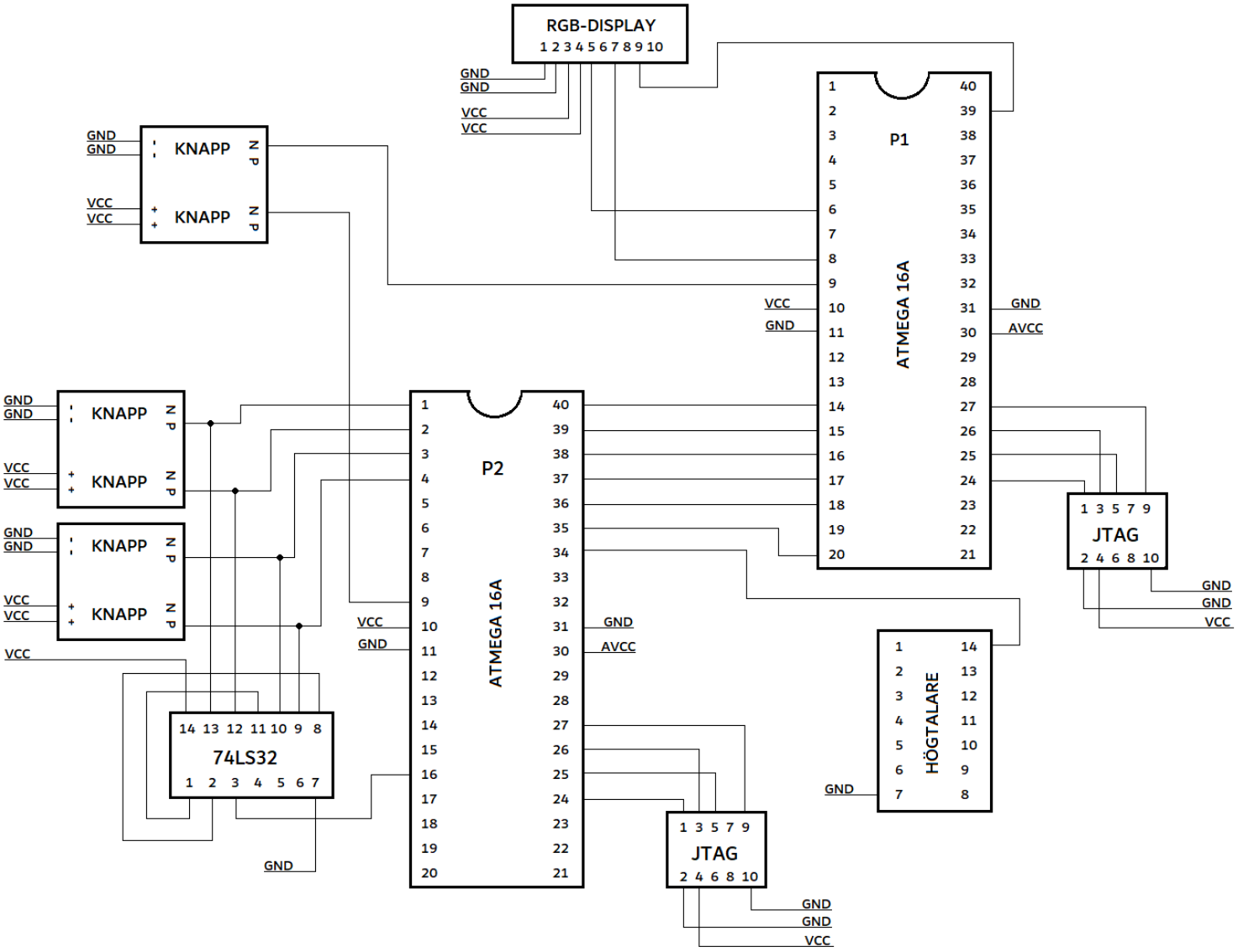
11. Referenslista

Atmel(2002). Atmega16(L) Preliminary. [2019-04-10].
http://www.isy.liu.se/edu/kurs/TSIU02/5_Atmega16.pdf

B. H. Suits, Physics Department, Michigan Technological University(1998). *Physics of Music.* [2019-04-10].
https://pages.mtu.edu/~suits/notefreqs.html

12. Bilagor

12.1 Kretsschema



12.2 Kod Processor 1

```
.org       $00
rjmp       COLD
.org       OVF0addr
rjmp       LED_DISPLAY_MUX

.org       OC1Aaddr
rjmp       CLOCK

.org $02A
ANOD_TB:
           .db        $FE, $FD, $FB, $F7, $EF, $DF, $BF, $7F
ANOD_PEKARE:
           .db $00, $03, $06, $09, $0C, $0F, $12, $15
SHIFT_REGISTER:
           .db         $FF, $FF, $7F, $3F, $1F, $0F, $07, $03, $01, $00
BLOCK_REGISTER:
           .db /*Block 1, $00*/$03, $03, $00,/*Block 2, $03*/ $03, $06,
$00, $02, $03, $01, $03, $06, $00, $02, $03, $01,/*Block 3, $0F*/ $06, $03,
$00, $01, $03, $02, $06, $03, $00, $01, $03, $02,/*Block 4, $1B*/ $07, $01,
$00, $01, $01, $03, $04, $07, $00, $03, $02, $02, /*Block 5, $1B*/ $07,
$04, $00, $02, $02, $03, $01, $07, $00, $03, $01, $01,  /*Block 6, $26*/
$02, $03, $02, $02, $07, $00, $02, $06, $02, $07, $02, $00, /*Block 7,
$32*/ $0F, $00, $00, $00, $04, $04, $04, $04
BLOCK_PEKARE:
           .db $00, $03, $0F, $1B, $27, $33, $3F

           .dseg
ANOD_INDEX:
           .byte 1
LOOP_INDEX:
           .byte 1
BLOCK_VALUE_INDEX:
           .byte 1
BLOCK_SPIN_INDEX:
           .byte 1
CURRENT_BLOCK_VALUES:
           .byte 4
CURRENT_BLOCK_VALUES_TEMP:
           .byte 4
BLOCK_DOWN_INDEX:
           .byte 1
BLOCK_DOWN_INDEX_SAVE:
           .byte 1
BLOCK_SIDE_INDEX:
           .byte 1
VMEM_TOP:
           .byte 24
VMEM_BOT:
           .byte 24
VMEM_BLOCK_TOP:
           .byte 24
VMEM_BLOCK_BOT:
           .byte 24
GMEM:
           .byte 16
TIME_COUNTER:
           .byte 1
TIME1:
```

```
            .byte 1
TIME2:
            .byte 1
TIMER_VALUEH:
            .byte 1
TIMER_VALUEL:
            .byte 1
RANDOM_VALUE:
            .byte 1
CALL_DOWN:
            .byte 1
            .cseg


LED_DISPLAY_MUX:
            push ZH
            push ZL
            push XH
            push XL
            push YH
            push YL
            push r16
            push r17
            push r18
            push r19
            push r20
            push r21
            push r22
            push r23
            in r16,SREG
            push r16

MUXEN:
            lds r18, ANOD_INDEX
            lds r19, LOOP_INDEX
            cpi r19, $04
            breq ANOD
            cpi r19, $08
            breq ANOD
            rjmp COLOUR
ANOD:
            ldi ZL, LOW(ANOD_TB*2)
            ldi ZH, HIGH(ANOD_TB*2)
            add ZL, r18
            lpm r16, Z
            call SPI_MasterTransmit
            rjmp FIX
COLOUR:
            ldi ZL, LOW(ANOD_PEKARE*2)
            ldi ZH, HIGH(ANOD_PEKARE*2)
            add ZL, r18
            lpm r18, Z
            cpi r19, $05
            brge COLOUR2
COLOUR1:
            ldi XL, LOW(VMEM_TOP)
            ldi XH, HIGH(VMEM_TOP)
            rjmp SEND
COLOUR2:
            ldi XL, LOW(VMEM_BOT)
            ldi XH, HIGH(VMEM_BOT)
```

```
            SEND:
            cpi r19, $01
            breq B
            cpi r19, $05
            breq B
            inc r18
            cpi r19, $02
            breq B
            cpi r19, $06
            breq B
            inc r18
B:
            add XL, r18
            ld r16, X
            call SPI_MasterTransmit
FIX:
            inc r19
            cpi r19, $09
            brlo A
            ldi r19, $01
            sbi PORTA, 1
            ;call DELAY
            cbi PORTA, 1
            lds r18, ANOD_INDEX
            inc r18
            cpi r18, $08
            brne R
            clr r18
R:
            sts ANOD_INDEX, r18
A:
            sts LOOP_INDEX, r19

RANDOM_BLOCK:
            lds r16, RANDOM_VALUE
            inc r16
            cpi r16, $07
            brne DONE_NEW_BLOCK
            clr r16
DONE_NEW_BLOCK:
            sts RANDOM_VALUE, r16

DONE_MUX:
            pop r16
            out SREG, r16
            pop r23
            pop r22
            pop r21
            pop r20
            pop r19
            pop r18
            pop r17
            pop r16
            pop YL
            pop YH
            pop XL
            pop XH
            pop ZL
            pop ZH

            reti
```

```
CLOCK:
            push ZH
            push ZL
            push XH
            push XL
            push YH
            push YL
            push r16
            push r17
            push r18
            push r19
            push r20
            push r21
            push r22
            push r23
            push r25
            in r16, SREG
            push r16

            lds r16, TIME1
            lds r17, TIME2
            inc r16
            cp r16, r17
            brne DONE_TIMER_CONTER
            dec r17
            ldi r20, $FF
            sts CALL_DOWN, r20
            sts TIME2, r17
            clr r16

DONE_TIMER_CONTER:
            sts TIME1, r16

            pop r16
            out SREG, r16
            pop r25
            pop r23
            pop r22
            pop r21
            pop r20
            pop r19
            pop r18
            pop r17
            pop r16
            pop YL
            pop YH
            pop XL
            pop XH
            pop ZL
            pop ZH

            reti

COLD:
            ldi r16, HIGH(RAMEND)
            out SPH, r16
            ldi r16, LOW(RAMEND)
            out SPL, r16

INIT_IT:
```

```
            ldi r16, $FF
            out DDRA, r16
            ldi r16, $E0

            out DDRD, r16
            call TEST_INIT
            call SPI_MasterInit
            call INTERNALCLOCKINIT

WARM:
            call ROLL_NEW_BLOCK
            call DELAY
            call CREATE_BLOCK
            call DELAY
            ;call BLOCK_INTO_GAME
            call NEW_BLOCK_INTO
LOOP:
            rjmp WARM

SPI_MasterTransmit:
            sbi PORTB, 4
            out SPDR, r16
Wait_Transmit:
            sbis SPSR, SPIF
            rjmp Wait_Transmit
            cbi PORTB, 4
Wait_End:
            ret


SPI_MasterInit:
            ldi r16, (1<<DDB5) | (1<<DDB7) | (1<<DDB4)
            out DDRB, r16

            ldi r16, (1<<SPE) | (1<<MSTR) | (0<<SPR0)
            out SPCR, r16
            ldi         r16,1<<SPI2X


            out         SPSR,r16
            ret


TEST_INIT:
            ldi ZL, LOW(VMEM_TOP)
            ldi ZH, HIGH(VMEM_TOP)
            ldi r16, $00
            ldi r17, $00
LOOP_TEST:
            st Z+, r16
            inc r17
            cpi r17, $18
            brne LOOP_TEST
            ldi ZL, LOW(VMEM_TOP)
            ldi ZH, HIGH(VMEM_TOP)
            ldi r16, $00
            std Z+0, r16
            std Z+1, r16
            std Z+2, r16
```

```
TEST_INIT2:
           ldi ZL, LOW(VMEM_BOT)
           ldi ZH, HIGH(VMEM_BOT)
           ldi r16, $00
           ldi r17, $00
LOOP_TEST2:
           st Z+, r16
           inc r17
           cpi r17, $18
           brne LOOP_TEST2
           ldi ZL, LOW(VMEM_BOT)
           ldi ZH, HIGH(VMEM_BOT)
           ldi r16, $00
           std Z+0, r16
           std Z+1, r16
           std Z+2, r16
           sts ANOD_INDEX, r16
           sts LOOP_INDEX, r16
           ldi r16, $00

           sts BLOCK_VALUE_INDEX, r16
TEST_INIT3:
           ldi ZL, LOW(VMEM_BLOCK_TOP)
           ldi ZH, HIGH(VMEM_BLOCK_TOP)
           ldi r16, $00
           ldi r17, $00
LOOP_TEST3:
           st Z+, r16
           inc r17
           cpi r17, $18
           brne LOOP_TEST3
TEST_INIT4:
           ldi ZL, LOW(VMEM_BLOCK_BOT)
           ldi ZH, HIGH(VMEM_BLOCK_BOT)
           ldi r16, $00
           ldi r17, $00
LOOP_TEST4:
           st Z+, r16
           inc r17
           cpi r17, $18
           brne LOOP_TEST4
TEST_INIT5:
           ldi ZL, LOW(GMEM)
           ldi ZH, HIGH(GMEM)
           ldi r16, $00
           ldi r17, $00
LOOP_TEST5:
           st Z+, r16
           inc r17
           cpi r17, $16
           brne LOOP_TEST5
           sbi PORTA, 1
           call DELAY
           cbi PORTA, 1
           ret


INTERNALCLOCKINIT:
           ldi r16, (1<<CS01)
           out TCCR0, r16
           ldi r16, (1<<TOIE0)
```

```
          out TIMSK, r16
INIT_TIMER:
          ;ldi                r16, (1<<CS01)
;         out                 TCCR0, r16
          ldi                 r16, (1<<COM1A1)
          out                 TCCR1A, r16
          ldi                 r16, (1<<WGM12)|(1<<CS11)|(1<<CS10)
          out                 TCCR1B, r16


          ldi r16, $FF
          sts TIME2, r16
          ldi r16, $00
          ldi r17, $F0
          sts TIMER_VALUEH, r16
          sts TIMER_VALUEL, r17

          out                 OCR1AH, r16
          out                 OCR1AL,    r17
          ldi                 r16, (1<<TOIE0)|(1<<OCIE1A)
          out                 TIMSK, r16

          sei
          ret

DELAY:
push r22
push r23
          ldi                 r22, $08
delayYttreLoop:
          ldi                 r23,$1F
delayInreLoop:
          dec                 r23
          brne       delayInreLoop
          dec                 r22
          brne       delayYttreLoop
pop r23
pop r22
          ret


DELAY2:
push r22
push r23
          ldi                 r22, $FF
delayYttreLoop2:
          ldi                 r23,$FF
delayInreLoop2:
          dec                 r23
          brne       delayInreLoop2
          dec                 r22
          brne       delayYttreLoop2
pop r23
pop r22
          ret


SHIFT:
          ldi ZL, LOW(SHIFT_REGISTER*2)
          ldi ZH, HIGH(SHIFT_REGISTER*2)
          ldi XL, LOW(VMEM_TOP)
```

```
            ldi XH, HIGH(VMEM_TOP)
            cpi r20, $0A
            brge SHIFTLOOP_TOP
            cpi r20, $09
            brne PC+2
            ldi r25, $FF
            ldi YL, LOW(VMEM_BOT)
            ldi YH, HIGH(VMEM_BOT)
            add ZL, r20
            lpm r21, Z
            mov r20, r21
            lsl r20
            com r20
            ldi r18, $00
SHIFTLOOP:
            ld r16, X
            ld r17, Y
            mov r22, r17
            and r17, r21
            and r22, r20
            lsl r17
            cpi r25, $FF
            brne PC+2
            andi r22, $FE
            or r17, r22
            lsl r16
            brcs IS_1
            rjmp CHECKDONE
IS_1:
            ori r17, $01
CHECKDONE:
            st X, r16
            st Y, r17
            ld r16, X+
            ld r17, Y+
            inc r18
            cpi r18, $18
            brne SHIFTLOOP
            rjmp DONE
SHIFTLOOP_TOP:
            subi r20, $08
            add ZL, r20
            lpm r17, Z
            mov r18, r17
            com r18
            lsl r18
            ldi r19, $00
SHIFTLOOPLOOP_TOP:
            ld r16, X
            mov r20, r16
            and r16, r17
            and r20, r18
            lsl r16
            or r16, r20
TOPCHECKDONE:
            st X, r16
            ld r16, X+
            inc r19
            cpi r19, $18
            brne SHIFTLOOPLOOP_TOP
DONE:
```

```
            ret


ROLL_NEW_BLOCK:
            lds r17, RANDOM_VALUE
            sts BLOCK_VALUE_INDEX, r17
            ret


CREATE_BLOCK:
            ldi ZL, LOW(BLOCK_PEKARE*2)
            ldi ZH, HIGH(BLOCK_PEKARE*2)
            lds r19, BLOCK_VALUE_INDEX
            add ZL, r19
            lpm r16, Z
            ldi ZL, LOW(BLOCK_REGISTER*2)
            ldi ZH, HIGH(BLOCK_REGISTER*2)
            add ZL, r16
            lpm r16, Z+
            lpm r17, Z+
            lpm r18, Z+
            cpi r19, $06
            brne CREATE_BLOCK_3
            lpm r19, Z
            rjmp TIME_TO_CREATE
CREATE_BLOCK_3:
            clr r19
TIME_TO_CREATE:
            ldi XL, LOW(CURRENT_BLOCK_VALUES)
            ldi XH, HIGH(CURRENT_BLOCK_VALUES)
            swap r16
            lsr r16
            swap r17
            lsr r17
            swap r18
            lsr r18
            swap r19
            lsr r19
            st X+, r16
            st X+, r17
            st X+, r18
            st x, r19
            clr r16
            sts BLOCK_DOWN_INDEX, r16
            ldi r16, $01
            sts BLOCK_SPIN_INDEX, r16
            ldi r16, $03
            sts BLOCK_SIDE_INDEX, r16
            ret


NEW_BLOCK_INTO:
            call SHIFT_VIDEOBLOCK
;           call DELAY
;           call DELAY
;           call DELAY
            call ADD_VIDEOBLOCK
            call DELAY
GAME_ON:
            lds r16, CALL_DOWN
            cpi r16, $FF
```

```
            brne GAME_ON_BUTTONS
            call BLOCK_DOWN
            ldi r16, $00
            sts CALL_DOWN, r16
GAME_ON_BUTTONS:
            in r16, PIND
            andi r16, $1F

            cpi r16, $00
            breq GAME_ON
            cpi r16, $02
            breq LEFT
            cpi r16, $04
            breq RIGHT
            cpi r16, $08
            breq SPIN
            cpi r16, $10
            breq DOWN_ALL
READ_FAULT:
            rjmp GAME_ON


LEFT:
            call BLOCK_LEFT
            call CHECK_DONE
            rjmp GAME_ON


RIGHT:
            call BLOCK_RIGHT
            call CHECK_DONE
            rjmp GAME_ON


SPIN:
            call BLOCK_SPIN
            call CHECK_DONE
            rjmp GAME_ON


DOWN_ALL:
            call BLOCK_DOWN
            cpi r25, $FF

            breq DOWN_DOWN

            rjmp DOWN_ALL

DOWN_DOWN:

            call CHECK_DONE
            rjmp GAME_ON


DOWN_1:
            cpi r20, $01
            call SHIFT
            call CHECK_DONE
            rjmp GAME_ON
```

```
CHECK_DONE:
           clr r16
           sbi PORTD, 6

LOOP_DONE:
           in r16, PIND
           andi r16, $1F

           cpi r16, $00
           brne LOOP_DONE
           cbi PORTD, 6

           ret


BLOCK_SPIN:
           ldi ZL, LOW(BLOCK_PEKARE*2)
           ldi ZH, HIGH(BLOCK_PEKARE*2)
           lds r19, BLOCK_VALUE_INDEX
           lds r20, BLOCK_SPIN_INDEX
           cpi r19, $00
           brne CAN_DO
           jmp DONE_BLOCK_SPIN
CAN_DO:
           add ZL, r19
           lpm r16, Z
           cpi r19, $06
           brne NORMAL_SPIN_PEKARE
           ldi r17, $04
           rjmp INC_PEKARE
NORMAL_SPIN_PEKARE:
           ldi r17, $03
INC_PEKARE:
           cpi r20, $00
           breq DONE_INC_PEKARE
           add r16, r17
           dec r20
           rjmp INC_PEKARE
DONE_INC_PEKARE:
           ldi ZL, LOW(BLOCK_REGISTER*2)
           ldi ZH, HIGH(BLOCK_REGISTER*2)
           add ZL, r16
           ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

           ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)
           lpm r16, Z+
           st         Y+, r16
           lpm r16, Z+
           st         Y+, r16
           lpm r16, Z+
           st         Y+, r16
           cpi r19, $06
           brne ADD_BLOCK_SPIN_BEFORE
           lpm r16, Z
           st         Y+, r16
           rjmp ADD_BLOCK_SPIN
ADD_BLOCK_SPIN_BEFORE:
           clr r16
           st Y+, r16
ADD_BLOCK_SPIN:
           ldi ZL, LOW(GMEM)
```

```
            ldi ZH, HIGH(GMEM)
            lds r20, BLOCK_DOWN_INDEX
            add ZL, r20
            ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

            ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)

            ldi r18, $03
LOOP_BLOCK_SPIN:
            ld r16, Y+
            ld r17, Z
            and r17, r16
            cpi r17, $00
            brne DONE_BLOCK_SPIN
            ld r17, -Z
            cpi r18, $00
            breq DONE_LOOP_BLOCK_SPIN
            dec r18
            rjmp LOOP_BLOCK_SPIN
DONE_LOOP_BLOCK_SPIN:
            ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

            ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)

            ldi XL, LOW(CURRENT_BLOCK_VALUES)

            ldi XH, HIGH(CURRENT_BLOCK_VALUES)
            lds r20, BLOCK_SIDE_INDEX
            ldi r18, $04
LOOP_DONE_LOOP_BLOCK_SPIN:
            ld r17, Y+
            clr r19
GO_TO_SIDE:
            cp r20, r19
            breq DONE_SIDE_STEP
            inc r19
            lsl r17
            brcc GO_ON
            call ONE_LESS_SHIFT
            rjmp SAVE_IT
GO_ON:
            rjmp GO_TO_SIDE
DONE_SIDE_STEP:
            st X+, r17
            dec r18
            cpi r18, $00
            brne LOOP_DONE_LOOP_BLOCK_SPIN
SAVE_IT:
            call VIDEO_BLOCK_SPIN
            ;lds r16, BLOCK_SIDE_INDEX
            ;dec r16
            ;sts BLOCK_SIDE_INDEX, r16
            lds r20, BLOCK_SPIN_INDEX
            inc r20
            lds r19, BLOCK_VALUE_INDEX
            cpi r19, $06
            breq LONG_CHECK
            cpi r20, $04
            brne STORE_SPIN_INDEX
            clr r20
            rjmp STORE_SPIN_INDEX
```

```
LONG_CHECK:
           cpi r20, $02
           brne STORE_SPIN_INDEX
           clr r20
STORE_SPIN_INDEX:
           sts BLOCK_SPIN_INDEX, r20
DONE_BLOCK_SPIN:
           ret


ONE_LESS_SHIFT:
           ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

           ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)

           ldi XL, LOW(CURRENT_BLOCK_VALUES)

           ldi XH, HIGH(CURRENT_BLOCK_VALUES)
           lds r20, BLOCK_SIDE_INDEX
           dec r20
           sts BLOCK_SIDE_INDEX, r20
           ldi r18, $04
LOOP_DONE_LOOP_BLOCK_SPIN_LESS:
           ld r17, Y+
           clr r19
           GO_TO_SIDE_LESS:
           cp r20, r19
           breq DONE_SIDE_STEP_LESS
           inc r19
           lsl r17
           rjmp GO_TO_SIDE_LESS
           DONE_SIDE_STEP_LESS:
           st X+, r17
           dec r18
           cpi r18, $00
           brne LOOP_DONE_LOOP_BLOCK_SPIN_LESS
           ret


BLOCK_DOWN:
           ldi r25, $00
                                                                  ; STYR
BLOCK_DOWN_ALL
           ldi ZL, LOW(GMEM)
           ldi ZH, HIGH(GMEM)
           lds r20, BLOCK_DOWN_INDEX
           cpi r20, $10
           breq NOT_POSSIBLE_DOWN
           inc r20
           add ZL, r20
           ldi XL, LOW(CURRENT_BLOCK_VALUES)

           ldi XH, HIGH(CURRENT_BLOCK_VALUES)
           ldi r18, $03
LOOP_BLOCK_DOWN:
           ld r16, X+
```

```
            ld r17, Z


            and r17, r16
            cpi r17, $00
            brne NOT_POSSIBLE_DOWN
            ld  r17, -Z

            cpi r18, $00
            breq DONE_LOOP_BLOCK_DOWN
            dec r18
            rjmp LOOP_BLOCK_DOWN
DONE_LOOP_BLOCK_DOWN:
            call VIDEO_BLOCK_DOWN
            sts BLOCK_DOWN_INDEX, r20
            ret
NOT_POSSIBLE_DOWN:
            ldi r25, $FF
                                                BLOCK_DOWN_ALL
            ldi ZL, LOW(GMEM)
            ldi ZH, HIGH(GMEM)
            lds r20, BLOCK_DOWN_INDEX
            inc r20
            add ZL, r20
            ldi XL, LOW(CURRENT_BLOCK_VALUES)

            ldi XH, HIGH(CURRENT_BLOCK_VALUES)
            ldi r18, $03
LOOP_BLOCK_FASTA:
            ld r16, X+
            ld r17, -Z
            or r17, r16
            st Z, r17
            cpi r18, $00
            breq LOOP_DONE_FASTA
            dec r18
            rjmp LOOP_BLOCK_FASTA
LOOP_DONE_FASTA:
            ldi XL, LOW(VMEM_BLOCK_BOT)
            ldi XH, HIGH(VMEM_BLOCK_BOT)
            ldi YL, LOW(VMEM_BLOCK_TOP)
            ldi YH, HIGH(VMEM_BLOCK_TOP)
            ldi r18, $16
            clr r16
LOOP_DELETE_VMEM:
            st X+, r16
            st Y+, r16
            dec r18
            cpi r18, $00
            brne LOOP_DELETE_VMEM
CHECK_IF_ROW_CLEAR:
            ldi ZL, LOW(GMEM)
            ldi ZH, HIGH(GMEM)
            ldi r16, $12
            add ZL, r16
            ldi r18, $16
            ldi r20, $00
            clr r16
LOOP_CLEAR_ROW:
```

```
            inc r20
            ld r17, -Z
            cpi r17, $FF
            brne NOT_FULL
CLEAR_IT:
            ld r17, -Z
            std Z+1, r17
            dec r18
            cpi r18, $00
            brne CLEAR_IT
            call SHIFT
            rjmp CHECK_IF_ROW_CLEAR
NOT_FULL:
            dec r18
            cpi r18, $00
            brne LOOP_CLEAR_ROW
CREATE_NEW_BLOCK_INGAME:
            call ROLL_NEW_BLOCK
            call CREATE_BLOCK
            call SHIFT_VIDEOBLOCK
            call ADD_VIDEOBLOCK
            ret


BLOCK_RIGHT:
            ldi ZL, LOW(GMEM)
            ldi ZH, HIGH(GMEM)
            lds r17, BLOCK_DOWN_INDEX
            add ZL, r17
            ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

            ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)

            ldi XL, LOW(CURRENT_BLOCK_VALUES)

            ldi XH, HIGH(CURRENT_BLOCK_VALUES)
            ldi r18, $03
LOOP_BLOCK_RIGHT:
            ld r16, X+


            sbrc r16, 0


            rjmp NOT_POSSIBLE_RIGHT

            lsr r16


            ld r17, Z


            and r17, r16
            cpi r17, $00
            brne NOT_POSSIBLE_RIGHT
            st Y+, r16
            ld  r17, -Z


            cpi r18, $00
            breq DONE_LOOP_BLOCK_RIGHT
```

```
            dec r18
            rjmp LOOP_BLOCK_RIGHT
DONE_LOOP_BLOCK_RIGHT:
            ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

            ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)

            ldi XL, LOW(CURRENT_BLOCK_VALUES)

            ldi XH, HIGH(CURRENT_BLOCK_VALUES)
            ldi r18, $04
LOOP_DONE_LOOP_BLOCK_RIGHT:
            ld r17, Y+
            st X+, r17
            dec r18
            cpi r18, $00
            brne LOOP_DONE_LOOP_BLOCK_RIGHT
            call VIDEO_BLOCK_RIGHT
            lds r16, BLOCK_SIDE_INDEX
            dec r16
            sts BLOCK_SIDE_INDEX, r16
NOT_POSSIBLE_RIGHT:
            ret

BLOCK_LEFT:
            ldi ZL, LOW(GMEM)
            ldi ZH, HIGH(GMEM)
            lds r17, BLOCK_DOWN_INDEX
            add ZL, r17
            ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

            ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)

            ldi XL, LOW(CURRENT_BLOCK_VALUES)

            ldi XH, HIGH(CURRENT_BLOCK_VALUES)
            ldi r18, $03
LOOP_BLOCK_LEFT:
            ld r16, X+


            sbrc r16, 7


            rjmp NOT_POSSIBLE_LEFT

            lsl r16


            ld r17, Z


            and r17, r16
            cpi r17, $00
            brne NOT_POSSIBLE_LEFT
            st Y+, r16
            ld  r17, -Z

            cpi r18, $00
            breq DONE_LOOP_BLOCK_LEFT
            dec r18
```

```
            rjmp LOOP_BLOCK_LEFT
DONE_LOOP_BLOCK_LEFT:
            ldi YL, LOW(CURRENT_BLOCK_VALUES_TEMP)

            ldi YH, HIGH(CURRENT_BLOCK_VALUES_TEMP)

            ldi XL, LOW(CURRENT_BLOCK_VALUES)

            ldi XH, HIGH(CURRENT_BLOCK_VALUES)
            ldi r18, $04
LOOP_DONE_LOOP_BLOCK_LEFT:
            ld r17, Y+
            st X+, r17
            dec r18
            cpi r18, $00
            brne LOOP_DONE_LOOP_BLOCK_LEFT
            call VIDEO_BLOCK_LEFT
            lds r16, BLOCK_SIDE_INDEX
            inc r16
            sts BLOCK_SIDE_INDEX, r16
NOT_POSSIBLE_LEFT:
            ret

VIDEO_BLOCK_DOWN:
            call REMOVE_VIDEOBLOCK
            call SHIFT_VIDEOBLOCK
            call ADD_VIDEOBLOCK
            ret

VIDEO_BLOCK_RIGHT:
            call REMOVE_VIDEOBLOCK
            call SHIFT_VIDEOBLOCK_RIGHT
            call ADD_VIDEOBLOCK
VIDEO_BLOCK_RIGHT_DONE:
            ret


VIDEO_BLOCK_LEFT:
            call REMOVE_VIDEOBLOCK
            call SHIFT_VIDEOBLOCK_LEFT
            call ADD_VIDEOBLOCK
VIDEO_BLOCK_LEFT_DONE:
            ret

VIDEO_BLOCK_SPIN:
            call REMOVE_VIDEOBLOCK
            call SHIFT_VIDEOBLOCK_SPIN
            call ADD_VIDEOBLOCK
VIDEO_BLOCK_SPIN_DONE:
            ret


ADD_VIDEOBLOCK:
            ldi XL, LOW(VMEM_TOP)
            ldi XH, HIGH(VMEM_TOP)
            ldi YL, LOW(VMEM_BLOCK_TOP)
            ldi YH, HIGH(VMEM_BLOCK_TOP)
            ldi r18, $00
LOOP_TOP_ADD:
            ld r16, X
            ld r17, Y
```

```
            or r16, r17
            st X, r16
            ld r16, X+
            ld r17, Y+
            cpi r18, $18
            breq ADD_BOT
            inc r18
            rjmp LOOP_TOP_ADD
ADD_BOT:
            ldi XL, LOW(VMEM_BOT)
            ldi XH, HIGH(VMEM_BOT)
            ldi YL, LOW(VMEM_BLOCK_BOT)
            ldi YH, HIGH(VMEM_BLOCK_BOT)
            clr r18
LOOP_BOT_ADD:
            ld r16, X
            ld r17, Y
            or r16, r17
            st X, r16
            ld r16, X+
            ld r17, Y+
            cpi r18, $18
            breq DONE_ADD
            inc r18
            rjmp LOOP_BOT_ADD
DONE_ADD:
            ret


REMOVE_VIDEOBLOCK:
            ldi XL, LOW(VMEM_TOP)
            ldi XH, HIGH(VMEM_TOP)
            ldi YL, LOW(VMEM_BLOCK_TOP)
            ldi YH, HIGH(VMEM_BLOCK_TOP)
            ldi r18, $00
LOOP_TOP_REMOVE:
            ld r16, X
            ld r17, Y
            com r17
            and r16, r17
            st X, r16
            ld r16, X+
            ld r17, Y+
            cpi r18, $18
            breq REMOVE_BOT
            inc r18
            rjmp LOOP_TOP_REMOVE
REMOVE_BOT:
            ldi XL, LOW(VMEM_BOT)
            ldi XH, HIGH(VMEM_BOT)
            ldi YL, LOW(VMEM_BLOCK_BOT)
            ldi YH, HIGH(VMEM_BLOCK_BOT)
            clr r18
LOOP_BOT_REMOVE:
            ld r16, X
            ld r17, Y
            com r17
            and r16, r17
            st X, r16
            ld r16, X+
            ld r17, Y+
```

```
            cpi r18, $18
            breq DONE_REMOVE
            inc r18
            rjmp LOOP_BOT_REMOVE
DONE_REMOVE:
            ret



SHIFT_VIDEOBLOCK_LEFT:
            ldi ZL, LOW(VMEM_BLOCK_TOP)
            ldi ZH, HIGH(VMEM_BLOCK_TOP)
            ldi YL, LOW(VMEM_BLOCK_BOT)
            ldi YH, HIGH(VMEM_BLOCK_BOT)
            ldi r19, $00
SHIFT_VIDEOBLOCK_LEFT_LOOP:
            ldd r16, Z+3
            ldd r17, Y+3
            st Z, r16
            st Y, r17
            ld r16, Z+
            ld r17, Y+
            ;ld r16, X+
            ;ld r17, Y+
            inc r19
            cpi r19, $15
            brne SHIFT_VIDEOBLOCK_LEFT_LOOP
            clr r16
            st Z, r16
            st Y, r16
            ret

SHIFT_VIDEOBLOCK_RIGHT:
            ldi ZL, LOW(VMEM_BLOCK_TOP)
            ldi ZH, HIGH(VMEM_BLOCK_TOP)
            ldi YL, LOW(VMEM_BLOCK_BOT)
            ldi YH, HIGH(VMEM_BLOCK_BOT)
            ldi r16, $15
            add ZL, r16
            add YL, r16
            ldi r19, $00
SHIFT_VIDEOBLOCK_RIGHT_LOOP:
            ld r16, -Z
            ld r17, -Y
            std Z+3, r16
            std Y+3, r17
            inc r19
            cpi r19, $17
            brne SHIFT_VIDEOBLOCK_RIGHT_LOOP
            ldi ZL, LOW(VMEM_BLOCK_TOP)
            ldi ZH, HIGH(VMEM_BLOCK_TOP)
            ldi YL, LOW(VMEM_BLOCK_BOT)
            ldi YH, HIGH(VMEM_BLOCK_BOT)
            clr r16
            st Z, r16
            st Y, r16
            ret

SHIFT_VIDEOBLOCK:
            ldi XL, LOW(VMEM_BLOCK_TOP)
            ldi XH, HIGH(VMEM_BLOCK_TOP)
```

```
           ldi YL, LOW(VMEM_BLOCK_BOT)
           ldi YH, HIGH(VMEM_BLOCK_BOT)
           ldi r18, $00
SHIFTLOOP_VIDEOBLOCK:
           ld r16, X
           ld r17, Y
           lsl r17
           lsl r16
           brcs IS_1_VIDEOBLOCK
           rjmp CHECKDONE_VIDEOBLOCK
IS_1_VIDEOBLOCK:
           ori r17, $01
CHECKDONE_VIDEOBLOCK:
           st X, r16
           st Y, r17
           ld r16, X+
           ld r17, Y+
           inc r18
           cpi r18, $18
           brne SHIFTLOOP_VIDEOBLOCK
           lds r16, BLOCK_DOWN_INDEX
           cpi r16, $04
           brge DONE_DONESHIFT
           call ADD_FROM_BLOCKMEM
           ;lds r16, BLOCK_DOWN_INDEX
           ;inc r16
           ;sts BLOCK_DOWN_INDEX, r16
DONE_DONESHIFT:
           lds r16, BLOCK_DOWN_INDEX
           inc r16
           sts BLOCK_DOWN_INDEX, r16
           rjmp DONE

ADD_FROM_BLOCKMEM:
           ldi YL, LOW(VMEM_BLOCK_TOP)
           ldi YH, HIGH(VMEM_BLOCK_TOP)
           ldi XL, LOW(CURRENT_BLOCK_VALUES)
           ldi XH, HIGH(CURRENT_BLOCK_VALUES)
           lds r16, BLOCK_DOWN_INDEX
           add XL, r16
           ld r16, X
           clr r18
LOOP_ADD_FROM_BLOCKMEM:
           ld r17, Y
           lsl r16
           brcc NEXT_LOOP
           ori r17, $01
NEXT_LOOP:
           st Y, r17
           ld r17, Y+
           ld r17, Y+
           ld r17, Y+
           inc r18
           cpi r18, $08
           brne LOOP_ADD_FROM_BLOCKMEM
           ret



SHIFT_VIDEOBLOCK_SPIN:
           ldi XL, LOW(VMEM_BLOCK_BOT)
```

```
            ldi XH, HIGH(VMEM_BLOCK_BOT)
            ldi YL, LOW(VMEM_BLOCK_TOP)
            ldi YH, HIGH(VMEM_BLOCK_TOP)
            clr r16
            clr r17
LOOP_DELETE_MEM:
            st X+, r16
            st Y+, r16
            inc r17
            cpi r17, $18
            brne LOOP_DELETE_MEM
ADD_THE_NEW_BLOCK:
            clr r17
            lds r16, BLOCK_DOWN_INDEX
            sts BLOCK_DOWN_INDEX_SAVE, r16
            sts BLOCK_DOWN_INDEX, r17
LOOP_ADD_IN_SPINED_BLOCK:
            call SHIFT_VIDEOBLOCK
            lds r17, BLOCK_DOWN_INDEX
            lds r16, BLOCK_DOWN_INDEX_SAVE
            cp r17, r16
            brne LOOP_ADD_IN_SPINED_BLOCK
DONE_SPIN:
            ret
```

12.2 Kod Processor 2

```
            .equ TYST = 165

            .org $00
            rjmp START
            .org INT0addr
            rjmp MOVEMENT

START:

            ldi r16, HIGH(RAMEND)
            out SPH, r16
            ldi r16, LOW(RAMEND)
            out SPL, r16

            call INIT

            sei

MAIN:

            //MUSIKEN
            call GET_NOTE
            call F_LOOKUP
            call W_LOOKUP
            call SOUND
            rjmp MAIN
            END:
            rjmp END



CHECK_PINA:
            in r16, PINA
            andi r16, $20
            cpi r16, $20
            ; P1 KLAR
            brne CHECK_PINA
            clr r16
            out PORTA, r16
CLEAR_PINA:
            in r16, PINA
            andi r16, $20
            cpi r16, $00
            brne CLEAR_PINA
DONE:
            ret


;------------------------// SOUNDEFFECT

MOVE_SOUNDEFFECT:                                           ; Höger/
Vänster
            ldi r16, $93
            ldi r17, $07
            rjmp SOUNDEFFECT_DONE
DROPSPIN_SOUNDEFFECT:                                       ; Slå ner
block
```

```
            ldi r16, $52
            ldi r17, $08
            rjmp SOUNDEFFECT_DONE
SOUNDEFFECT_DONE:
            call SOUND
            ret

;------------------------// MUSIKEN

GET_NOTE:
            lpm r16, Z+
            mov r17, r16
            cpi r16, $00
            brne GET_NOTE_DONE
            ldi r18, $FF
            call DELAY
            ldi ZH, HIGH(MELODY*2)
            ldi ZL, LOW(MELODY*2)
GET_NOTE_DONE:
            ret

F_LOOKUP:
            push ZH
            push ZL
            ldi ZH, HIGH(FREQUENCY*2)
            ldi ZL, LOW(FREQUENCY*2)
            subi r16, $39
            add ZL, r16
            lpm r16, Z
            pop ZL
            pop ZH
            ret

W_LOOKUP:
            push ZH
            push ZL
            ldi ZH, HIGH(HALF_WAVE*2)
            ldi ZL, LOW(HALF_WAVE*2)
            subi r17, $39
            add ZL, r17
            lpm r17, Z
            pop ZL
            pop ZH
            ret

SOUND:
            cpi r16, $49
            brsh TONE
PAUSE:
            call NOBEEP
            dec r16
            brne PAUSE
            rjmp SOUND_DONE
TONE:
            call BEEP
SOUND_DONE:
            ret

BEEP:
            mov r18, r17
            call HIGH_SIG
```

```
            mov r18, r17
            call LOW_SIG
            dec r16
            breq D_BEEP
            rjmp BEEP
D_BEEP:
            ret

NOBEEP:
            ldi r18, TYST
            call LOW_SIG
            ldi r18, TYST
            call LOW_SIG
            ret

HIGH_SIG:
            sbi PORTA,6
            call DELAY
            ret

LOW_SIG:
            cbi PORTA, 6
            call DELAY
            ret

DELAY:
delayYttreLoop:
            ldi r19,0x04                                        ;
ljus/mörk snabb/långsam ton
delayInreLoop:
            dec r19
            brne delayInreLoop
            dec r18
            brne delayYttreLoop
            ret

MOVEMENT:

            push r16
            push r17
            push r18
            push r19
            in r16,SREG
            push r16

            in r16, PINB
            andi r16, $0F
            cpi r16, $00
            breq DONE_DONE

            cpi r16, $01                                        ;B0
            breq LEFT
            cpi r16, $02                                        ;B1
            breq RIGHT
            cpi r16, $04                                        ;B2
            breq ROTATE
            cpi r16, $08                                        ;B3
            breq DOWN
            rjmp DONE_DONE
LEFT:
            sbi PORTA, 1
```

```
            call MOVE_SOUNDEFFECT
            rjmp CHECK_DONE
RIGHT:
            sbi PORTA, 2
            call MOVE_SOUNDEFFECT
            rjmp CHECK_DONE
ROTATE:
            sbi PORTA, 3
            call DROPSPIN_SOUNDEFFECT
            rjmp CHECK_DONE
DOWN:
            sbi PORTA, 4
            call DROPSPIN_SOUNDEFFECT
            rjmp CHECK_DONE
CHECK_DONE:
            call CHECK_PINA
DONE_DONE:
            pop r16
            out SREG,r16
            pop r19
            pop r18
            pop r17
            pop r16

            reti


INIT:

            ldi r16, $5F
            out DDRA, r16                   ;Initierar A portarna

            clr r16
            out DDRB, r16                   ;Initierar B portarna

            ldi r16, (1<<ISC01) | (1<<ISC00)
            out MCUCR, r16
            ldi r16, (1<<INT0)
            out GICR, r16

            ldi ZH, HIGH(MELODY*2)
            ldi ZL, LOW(MELODY*2)

            ldi r16,$00

            ret


MELODY:     ;R1
                                                        R4

                        R7
                                    R8
              TEST
            .db
            "A<>:?:@<?:>:=<=:?:A<@:?:><>:?:@<A<?<=<=<<:@@:<B:D<C:B:A<:?:A<@:
?:><>:?:@<A<?<=<=:::" ,$00;<<AAA<???<@@@<>>><???<===<III<<AAA<???<@@@, $00
;<>>><==<??",$00
            ;           A<>:?:@<?:>:=<=:?:A<@:?:><>:?:@<A<?<=<=<<
JONSSONLIGAN:
```

```
            .db
"C<D<C<G<:G<C<D<C<G:C:G<:@:D<D<A<D<<C<D<C<F<:F<C<D<C<F:C:F:E:C<C<@<C<",$00

FREQUENCY:
            ;           9           :           ;           <           =
             >          ?           @           A           B           C
             D          E           F           G           H           I
             J          K           L           M           N           O
             P          Q           R           S
            .db
$10,$20,$30,$40,$49,$52,$57,$62,$6E,$7B,$83,$93,$A5,$AF,$C4,$DC,$45,$4D,$5C
,$67,$74,$8A,$9B,$B9,$CF,$E9,$00
            ;           1           2           3           4           D2
             E2         F2          G2          A2          B2          C3
             D3         E3          F3          G3          A3          Db2 Eb2
Gb2 Ab2 Bb2 Db3 Eb3 Gb3 Ab3 Bb3

HALF_WAVE: ;GLÖM EJ HALVERA OCH GÖRA OM TILL HEX
            ;           9           :           ;           <           =
             >          ?           @           A           B           C
             D          E           F           G           H           I
             J          K           L           M           N           O
             P          Q           R           S
            .db         $0, $0, $0, $0,
$17,$15,$14,$11,$0F,$0E,$0D,$0C,$0B,$0A,$09,$08,$19,$16,$12,$10,$14, $C,
$B,$5D ,$8, $7, $00
            ;           1           2           3           4           D2
             E2         F2          G2          A2          B2          C3
             D3         E3          F3          G3          A3          Db2 Eb2
Gb2 Ab2 Bb2 Db3 Eb3 Gb3 Ab3 Bb3
```