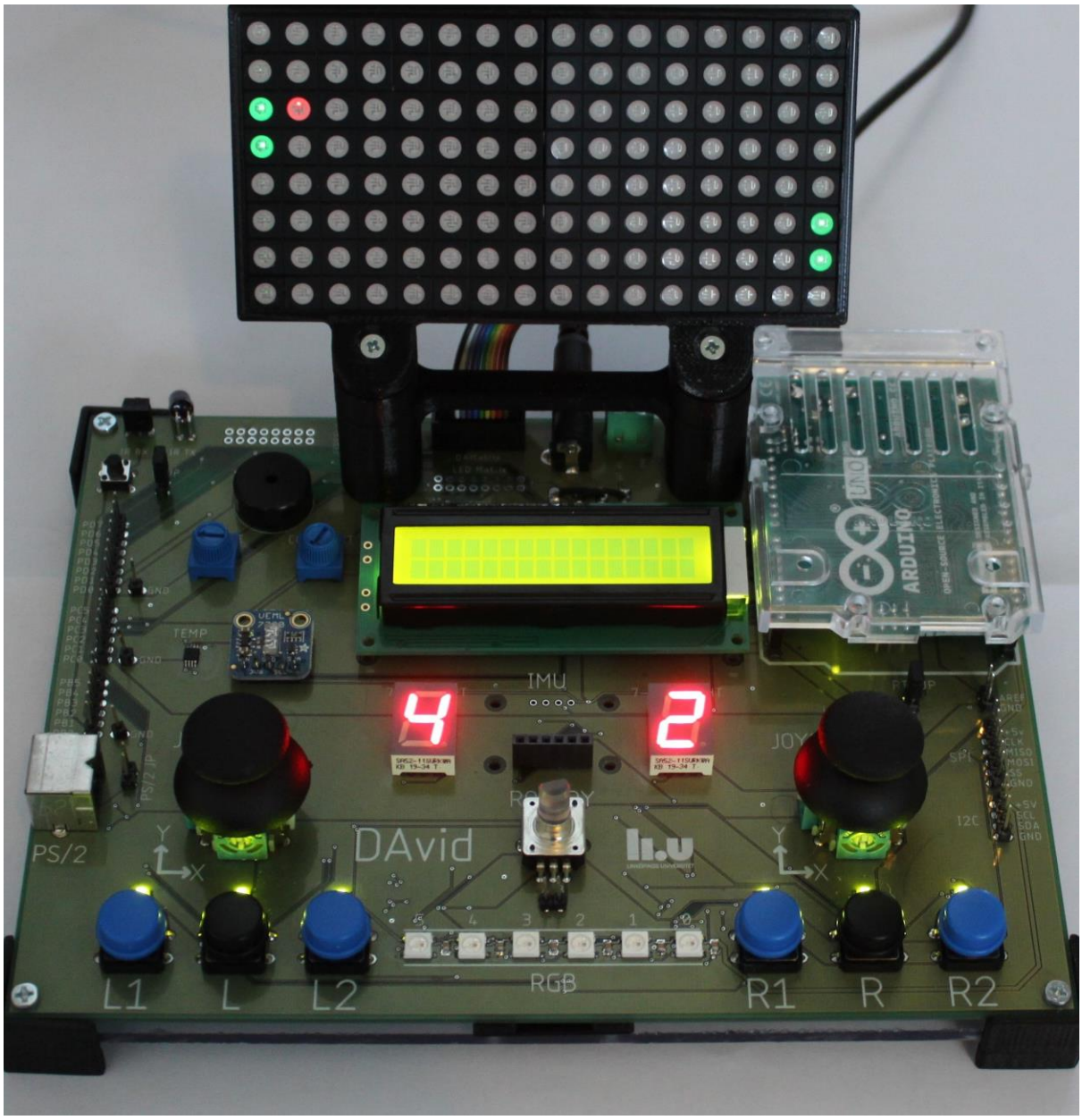
PONG-SPEL

Rapport för Mikrodatorprojekt utfört av:

Johan Klasén, Oskar Lund

Version 2, 2021-05-17

Kontaktperson: Johan, johkl473@student.liu.se

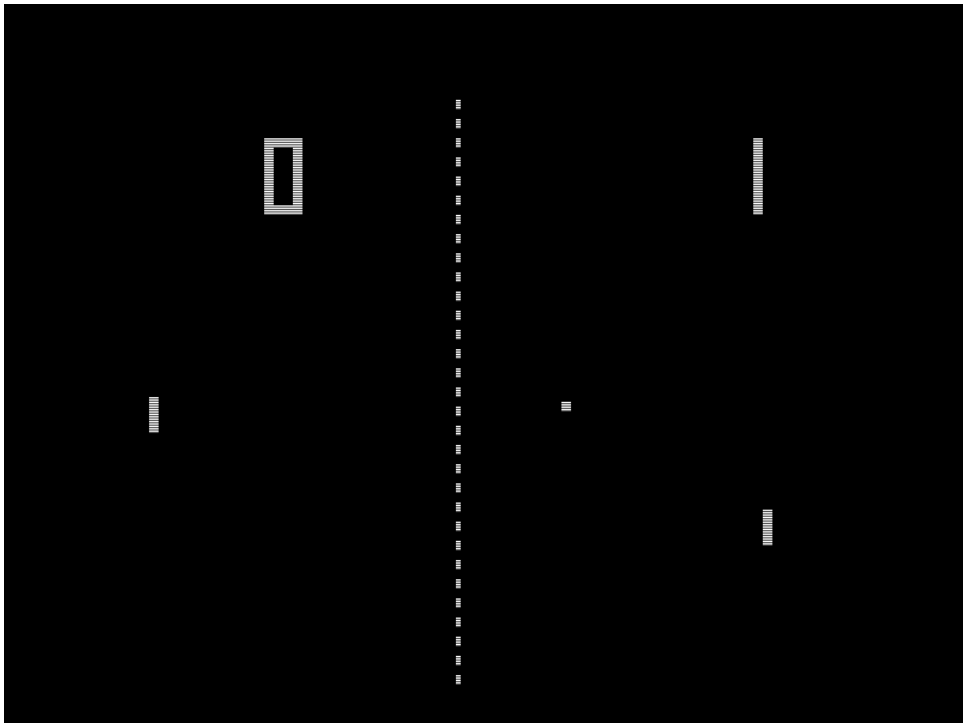


Innehållsförteckning

Figurförteckning

1 Inledning

I denna rapport beskrivs hur spelet Pong programmerades med hjälp av hårdvarukortet DAvid[1] och mikrokontrollern Atmega328p[2]. Projektet var en del av kursen TSIU51 och har syftet att med given hårdvara konstruera valfritt program som använder sig av modulerna på DAvid och med all kod skriven i AVR-assembler.



Figur 1: *Det klassiska spelet Pong, utgivet av företaget Atari för första gången 1972. Spelplanen är uppdelad i mitten med en vertikal streckad linje, och man ser bollen till höger om den. På varje ände av spelplanen ser man två plank som styrs av spelarna och ovanför finner man varsin poängräknare.*

I projektet beslutades att spelet Pong skulle programmeras. Pong bygger på en rektangulär spelplan som kan ses i **Figur 1**, där det på två motstående sidor finns ett spelarstyrt plank. En boll ”servas” från mitten och kan studsa på dem två sidorna som inte har planket. Nuddar bollen en av planksidorna får motstående spelare ett poäng. Spelarna kan flytta sitt plank längs med sin sida och målet är att förhindra att bollen kommer igenom, samtidigt som man styr in den på motståndarens sida. Vi ville försöka göra detta spel med hjälp av hårdvaran på ett DAvid-kort och någon form av display.

1.1 Kravspecifikation

För projektet skrevs en kravspecifikation med de skall-krav och utökade delar som skulle kunna implementeras. Nedan listas skall-kraven och de utökade kraven för spelet Pong.

[1] DAvid är ett hårdvarukort utvecklat av ISY, beskrivs utförligare senare i rapporten.

[2] Atmega328p är en mikrokontroller av Atmel.

1.1.1 Skall-krav:

- Spelet skall visas på en display (2 länkade DA-matrix).
- Skall styras av båda joysticksen av två spelare.
- Spelet skall starta med en knapptryckning.
- Båda 7-segmentsdisplayerna skall visa respektive spelares poäng.
- Varje spel skall gå i set (typ bäst av/först till 3 eller 5).
- Poäng och vinster skall visas som meddelande på display (Spelare 1 och 2 eller vänster/höger).
- Spelet skall ha ljud.

1.1.2 Utökade krav:

- Man skall välja namn på båda spelare (Rotary + knappar).
- Spelet skall ha ett enkelt menysystem för 1–2 spelare (antal).
- Vi skall bygga en ”AI” för enspelarläge (ev. så enkelt att det ena racket rör sig slumpmässigt eller alltid kontrar bollen).

1.1.3 Komponenter:

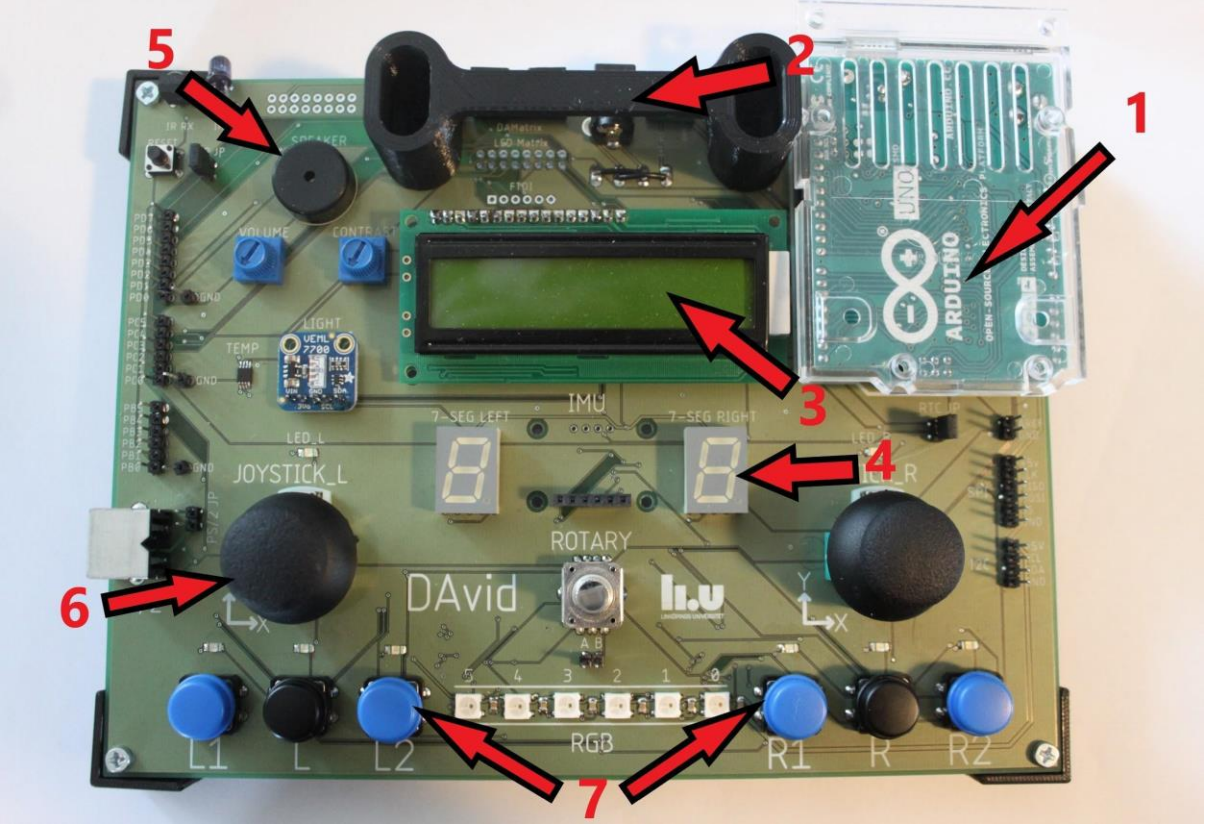
- 1x DAvid-kort
- 2x 8x8 RGB-LED-matriser, så kallade DA-matrix

2 Översikt

I projektet användes olika hårdvaror som programmerades som gränssnitt för användaren. I det här avsnittet kommer en kortfattad beskrivning av den hårdvara som användes och vad den användes till.

2.1 DAvid-kortet

DAvid är ett hårdvarukort utvecklat av ISY. Kortet är utrustat med ett stort antal hårdvarumoduler som LCD-skärm, knappar, joysticks, LED:s med mera [1]. Kortet har också möjlighet att koppla andra komponenter till kortet, till exempel en LED-matris, tangentbord o.s.v. Kortet styrs av en mikrokontroller i Arduino Uno R3.Programmeringen sker till Arduinons mikrokontroller som styr DAvid-kortet genom sina *I/O-pins* (*Input/Output*) eller *I/O*-stift.



Figur 2: *En översikt av David-kortet och de hårdvarumoduler som användes i projektet.*

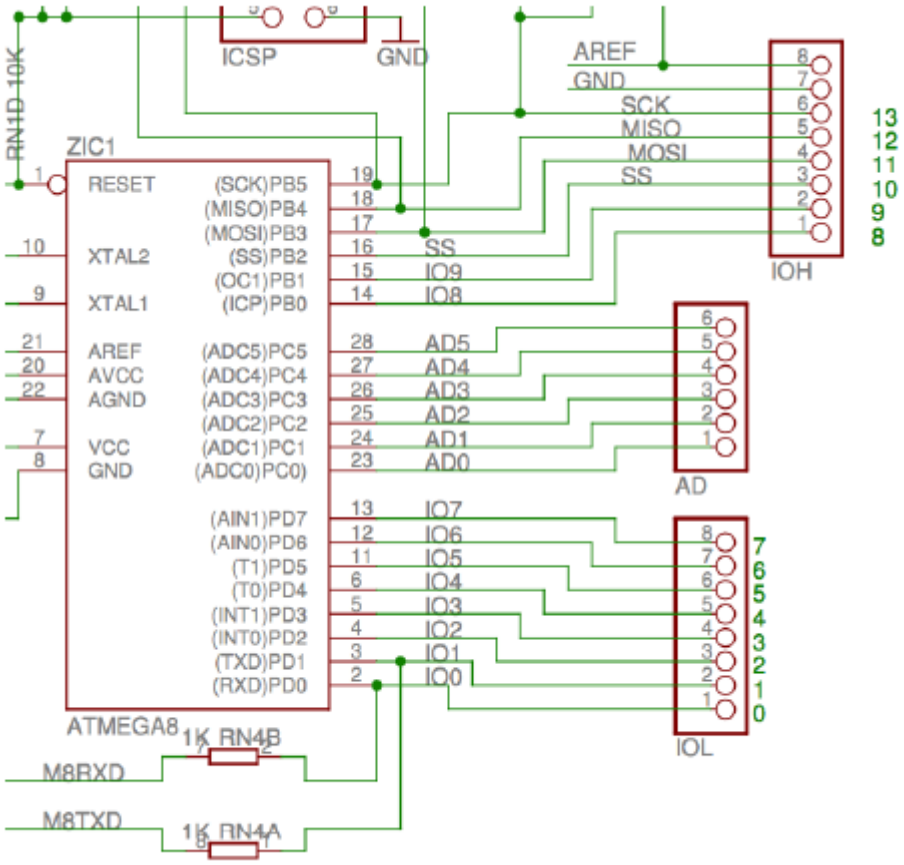
Nedan är en beskrivning av den hårdvara (1–7) som visas i **Figur 2** [1].

1. Arduino Uno med mikrokontrollern Atmega328p som programmerades i AVR-assembler.
2. Fäste för DA-matrixen som användes som spelplan för Pong.
3. En LCD-display för skriftliga meddelanden till spelarna. Till exempel användes den för att skriva välkomstmeddelande och vinstmeddelande.
4. 7-segmentsdisplayerna som visar spelarnas poäng. Ett decimalt värde från 0–9 eller ett hexadecimalt värde från 0–F kan visas på displayen.
5. En högtalare som ger enkla ljudeffekter till Pong.
6. Joysticks som används som kontroll för spelarnas plank och används endast vertikalt.
7. Knapparna som används för att starta spelet. LCD:n ger ett val att hålla ner knappen för att starta.

3 Projektets delar

3.1 Atmega328p

I projektet användes mikrokontrollern Atmega328p som styr själva DAvid-kortet. Atmega328p är en 8 bitars mikrokontroller som har kringutrustning som *Timers*, *Counters* och USART [2]. I detta avsnitt kommer först stift och portar beskrivas och därefter de *Timers* och *Counters* som användes i spelet.



Figur 3: *På bilden syns en schematisk bild på processorn ATmega328p I/O-pins. Stiften är uppdelade i 3 grupper, PB, PC och PD. PC0–PC5 är kopplade till så kallade Analog-till-Digital-omvandlare och kan därför ta emot analoga signaler, medan PB och PD endast kan hantera digitala signaler.*

3.1.1 Portar/Stift

Atmega328p har ett antal stift som används för att kommunicera med omvärlden [2]. Dessa pins är indelade som Port B, Port C och Port D och ett schema för det hittas i **Figur 3**. Alla dessa portar används för kommunikation med DAvid-kortet. De pins som används i projektet är PB5–PB0, PC5–PC0 och PD7–PD0. Mer om vad dessa pins används till förklaras i avsnitten Kommunikationsprotokoll och Moduler.

3.1.2 Timers/Counters

Atmega328p har två 8-bitars *Timer/Counters, eller räknare,* och en 16-bitars räknare [2]. Dessa används bland annat för att skapa *interrupts* med jämna mellanrum eller för att skapa en kontinuerlig räknare.

I spelet Pong används två räknare. En av dem bestämmer när spelets Component Table ska uppdateras, så att till exempel bollens eller plankens position kan förändras, och körs under hela spelets gång. Den andra aktiveras enbart när ett ljud ska spelas och hjälper till att skapa ljudsignaler med rätt periodicitet.

Båda räknarna körs i vad som i processordatabladet kallas *CTC-mode* (*Clear on Timer Compare*) vilket innebär att de laddas med ett värde som räknaren ska jämföras mot [2]. Räknarna tickar upp ett register med en bestämd takt som styrs av processorns hastighet, men kan saktas ner genom att sätta ett så kallat *prescaler-värde*. Registret jämförs ständigt mot det laddade värdet och när de matchar nollställs timerns register och ett avbrott kan genereras om så önskas.

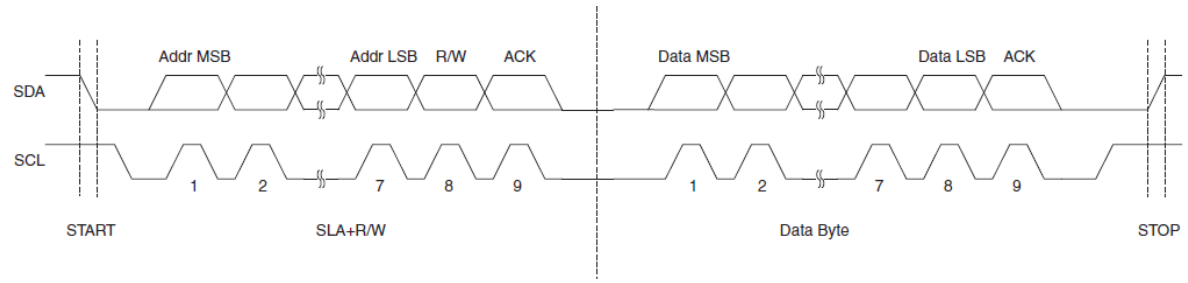
För 16-bitars-timern som styr spelavbrotten används ett prescalervärde på 64, vilket ger timern en takt på 250 kHz med en processorhastighet på 16 MHz [2]. Jämförelsevärdet laddas till 31 250 vilket gör att avbrottet kommer med en frekvens av 8 Hz, alltså åtta gånger i sekunden.

3.2 Kommunikationsprotokoll

DAvid-kortet styrs som tidigare nämnts av en mikrokontroller med ett begränsat antal *I/O*-stift. Dessa är ofta inte tillräckliga för att kunna styra ett stort antal hårdvarumoduler. För att mikrokontrollern ska kunna kommunicera med alla hårdvarumoduler på kortet måste ett annat kommunikationsprotokoll användas som kan hantera ett större antal kanaler. På DAvid-kortet används TWI (*Two-Wire serial Interface*) och SPI (*Serial Peripheral Interface*) för att göra just detta [1].

3.2.1 TWI

DAvid har en TWI buss som majoriteten av hårdvarumodulerna är knutna till. TWI-bussen är uppbyggd av två fysiska *ledningar,* SDA (*Serial Data-Line*) på *pin* PC4, SCL (*Serial Clock-Line*) på *pin* PC5 [1–2]. Modulerna kommunicerar med varandra genom att en *master* styr en eller flera *slaves* eller slavar. Mastern är den modul som bestämmer när kommunikationen mellan mastern och slaven ska börja och sluta. På DAvid-kortet är Atmega328p mastern och modulerna är slavar.

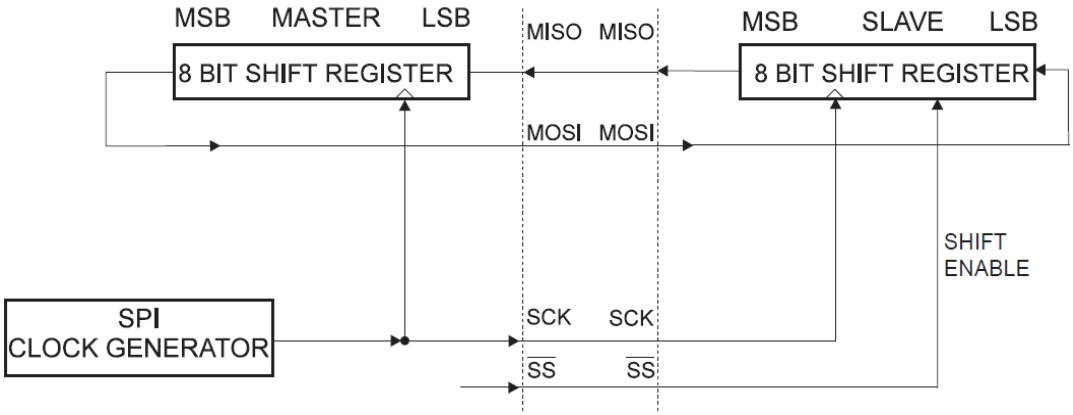


Figur 4: *Typisk överföringssekvens för TWI med datalinjen SDA och klockan SCL [2].*

När mastern vill kommunicera med en modul börjar den med att först dra SDA och sedan SCL låg och därefter skicka en 7-bitars adress och en R/W-bit (*Read/Write*). Slaven svarar sedan med ett *acknowledge* eller *ack* genom att släppa SDA, så att den går hög. Om mastern ska skriva skickar den därefter data i paket om en byte och slaven svarar med ett ack. När överföringen är klar markeras ett stopp genom att mastern släpper först SCL sedan SDA hög.

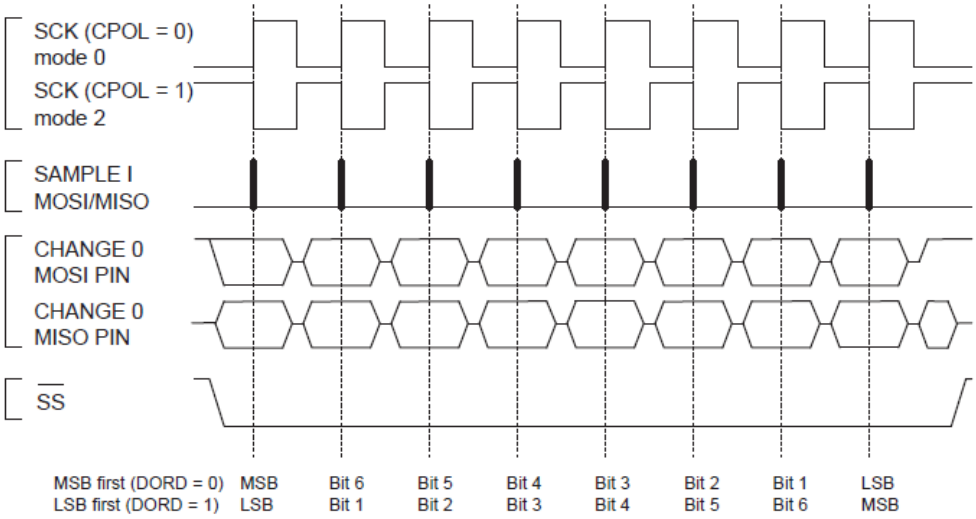
När slaven ska skriva till mastern sker detta genom att mastern talar om för slaven att den ska börja skriva genom att skriva adressen följt av en *read-bit.* Slaven kan sedan börja skicka bytes och mastern svarar med ett *ack*.

3.2.2 SPI



Figur 5: *Bilden visar en schematisk överföring via SPI. Mastern och slaven har varsina skiftregister som data överförs mellan, linjerna där mellan benämns MOSI och MISO. Mastern är ansvarig för att generera klockpulser via SCK som styr överföringshastigheten. Mastern hanterar även SS, Slave Select, som fungerar som en enable för slavens skiftregister och är masterns sätt att bestämma vilken slav den kommunicerar med.*

Förutom TWI använder DAvid också SPI för kommunikation med moduler som kan kopplas till ett separat kontaktdon. I spelet Pong används SPI för att skriva till en DA-matrix. SPI fungerar på liknande sätt som TWI men har några stora skillnader. Data-linjen för läsning och skrivning till slaven sker på olika portar. Dessa kallas för MOSI (*Master Out Slave In*) och MISO (*Master In Slave Out*), och ligger på respektive *pin* PB3 och PB4 [1]. Som TWI har också SPI en klocka som kallas för SCLK och ligger på *pin* PB5. Utöver dessa har även SPI en linje för SS (*Slave Select*). Dessa data-linjer visas och beskrivs i **Figur 5**. För DA-matrix är SS på *pin* PB2. SS har en egen linje till varje enskild slav men gemensamma linjer för MISO/MOSI vilket göra att SPI använder mer fysiska ledningar än TWI. En fördel med SPI är att datakommunikationen är snabbare.



Figur 6: *Bilden visar en typisk överföringssekvens över tid för SPI med klockan SCK och datan på MOSI. I överföringen till DA-matrix används CPOL = 0 och DORD = 0 [2].*

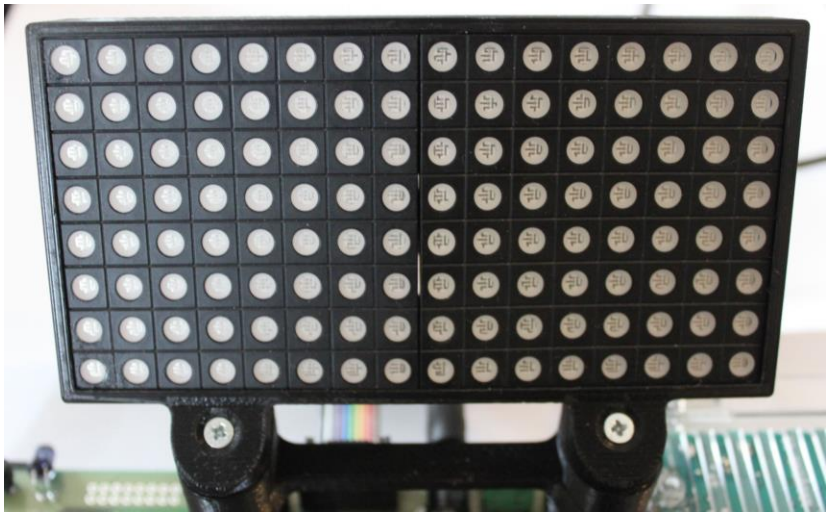
Kommunikationen börjar med att mastern drar SS låg och lägger ut den mest signifikanta biten på MOSI. Klockan pulsas genom att dras hög och sedan låg igen under en satt period styrt av överföringshastigheten. Detta är signalen för slaven att läsa av den skrivna datan, och när klockpulsen går låg fortsätter master att skifta ut nästa bit av datan på MOSI och pulsa klockan igen, vilket upprepas till hela datamängden har överförts. Avslutningsvis tydliggörs ett stopp genom att master sätter tillbaka SS till sitt höga ursprungsläge.

3.3 Moduler

I spelet Pong används ett antal olika hårdvarumoduler som joystick, knappar, LED:s och högtalare. Hur dessa används och hur de fungerar kommer beskrivas i denna del.

3.3.1 Led-matrisen DA-matrix

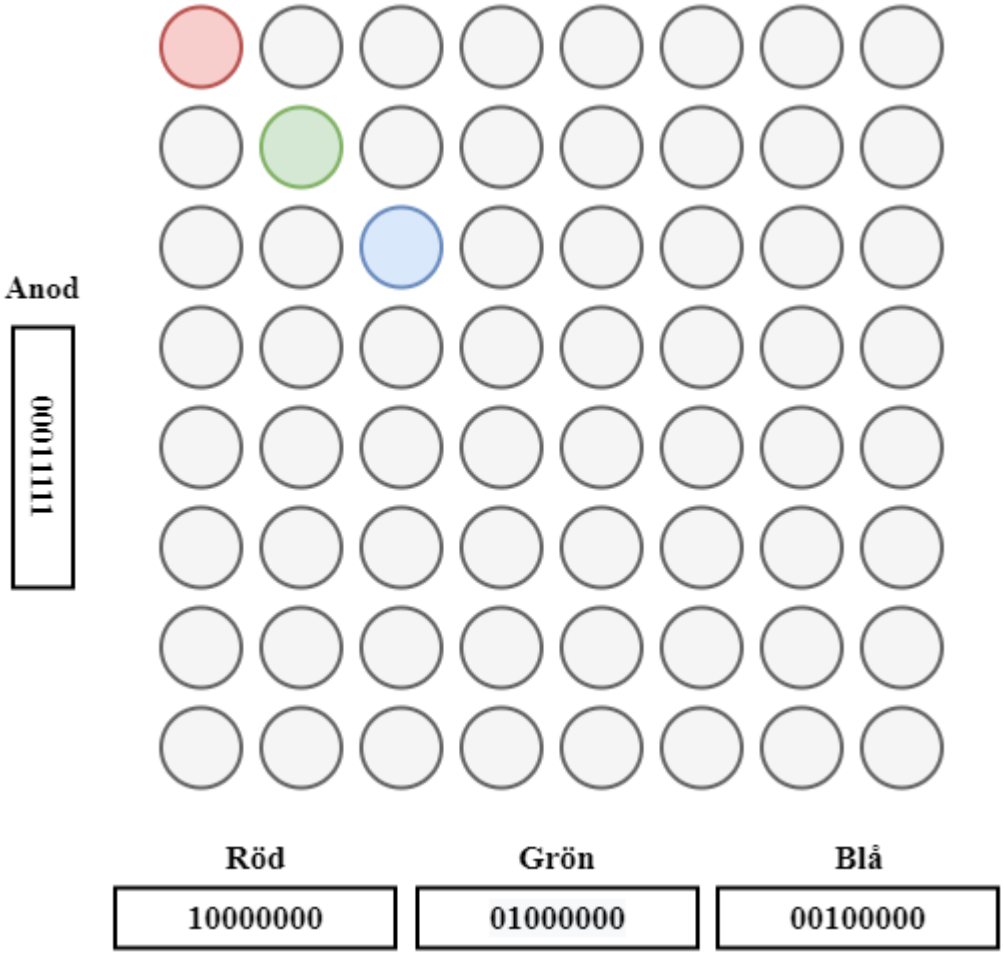
DA-matrix är en 8x8 RGB-LED matris som använder kommunikationsprotokollet SPI [1, 3]. I spelet Pong används DA-matrix som display där dioderna fungerar som pixlar för att visa plankens och bollens position.



Figur 7: *Bild på två länkade DA-matriser med dimensionen 16x8 pixlar. Varje ”pixel” eller ruta består av tre separata dioder i färgerna röd, grön respektive blå (RGB), som kan tändas individuellt. Displayerna är kopplade till DAvid-kortet via SPI, vilket beskrivs mer utförligt senare i rapporten.*

För att tända en diod behöver raden man ska tända påföras en spänning och den kolumn som dioden ligger på jordas vilket ger en ström genom dioden [3]. Då en DA-matrix är en RGB-matris är det lite krångligare att få den att gör som man vill och man använder speciella drivkretsar för att få rätt diod att lysa.

För att kunna styra matrisen seriellt med SPI behöver matrisen spara de värden som skrivs till den genom SPI. Detta görs genom att skifta in bitarna till 4 skiftregister [1, 3]. En för raderna och tre för färgerna röd, grön och blå. Raderna som ska lysa påförs en spänning då en 0 har skiftats och en 0 i RGB registren jordar kolumnen.

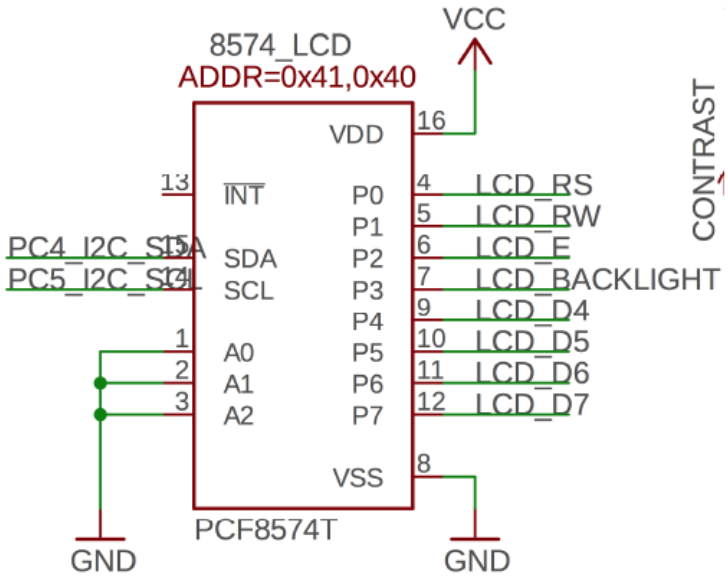


Figur 8: *I figuren är de svarta rektanglarna skiftregister med data för vilken diod som ska lysa i vilken färg. I detta fall kommer dioderna i diagonalen lysa rött, grönt och blått.*

Med SPI kan man då skriva 32 bitar till matrisen för att tända valfria dioder. I fallet med spelet Pong används två matriser vilket gör att man skriver först till den ena och sedan till den andra. Den första matrisen skiftar då vidare de första 4 byten till den andra matrisen och efter ytterligare 4 bytes är skrivningen till hela 16x8 matrisen klar.

3.3.2 I/O-expander - PCF8574

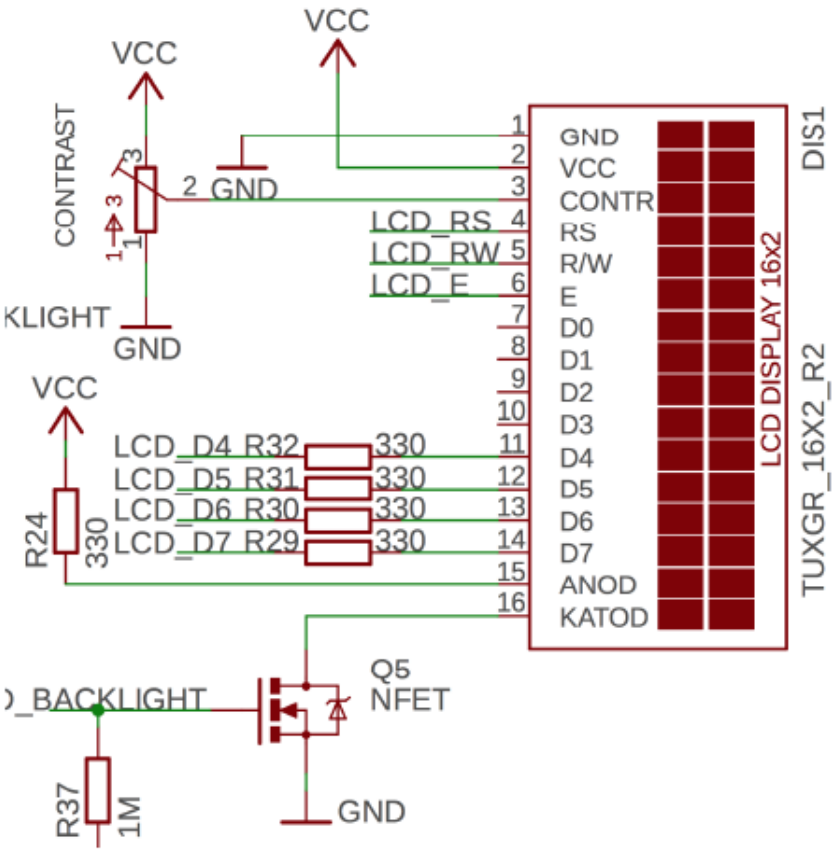
Många av modulerna på DAvid styrs av TWI. Men dessa moduler har inget sätt att själva översätta det seriella datat. I stället används en så kallad I/O-expander (PCF8574) [1, 4] som mastern kommunicera med och I/O-expandern kommunicerar med modulen parallellt.



Figur 9: *Schematisk bild över I/O expandern PCF8574. Just denna visar hur man på DAvid-kortet kan skriva data till 8 parallella portar (P0–P7) på en LCD display genom två datalinjer (SDA och SDL) från Arduinon och m.h.a. TWI-kommunikation. Signalera A0–A2 används för att bestämma adressen till I/O-expander och man kan på så sätt ha flera olika på samma TWI-buss.*

3.3.3 LCD-display - LCD HD44780

I Pong används en LCD skärm för att visa meddelanden i form av text till spelarna, till exempel ett välkomstmeddelande eller ett vinstmeddelande. Skärmen styrs genom en LCD kontrollen HD44780 [1, 5]. För att processorn ska kunna kommunicera med LCD:n gör den det genom TWI. För att det ska gå behövs också en I/O-expander som översätter seriellt data till parallell kommunikation med HD44780.



Figur 10: *På bilden visas hur I/O-expandern är kopplade till drivchippet HD4470. Datalinjerna D7 - D4 och styrsignalerna RS, RW, E och BACKLIGHT kommer alla från I/O-expandern som i sin tur tar emot datan via TWI från DAvid-kortets mikroprocessor.*

Skrivning till displayen görs via fyra datalinjer, *D7 – D4*, i vad tillverkarna kallar *4 bit mode* [1, 5], men samtliga displaykommandon och tecken omfattar en byte. Detta leder till att varje skrivning av en byte behöver delas upp i två 4-bits delar som skickas i sekvens, med höga *nibblen* (ena halvan av en *byte*) först. Signalen *BACKLIGHT* reglerar strömtillförseln till displayens bakgrundsljus. *R/W* avgör om man vill skriva till eller läsa från displayen. Båda dessa ett-ställs vid uppstart av kortet och hålls konstanta. Styrsignalen *RS* avgör om det är ett kommando, eller ett tecken för utskrift som skickas till displayen. Styrsignalen *E* behöver sedan pulsas för att ange för displayen att datalinjerna har ändrats och kan läsas av. En skrivning av ett tecken på en byte från processorn till displayen blir således 4 skrivningar till I/O-expandern via TWI. En första skrivning med den höga *nibblen* av datan lagt på D7–D4, med E satt hög och RS i önskat läge, en andra identisk skrivning fast med E satt låg, följt av en tredje skrivning med låga *nibblen* av datan på D7 – D4 och E hög, och en sista med E sedan satt låg.

3.3.4 Högtalare

För att få ljud till spelet användes DAvid-kortets piezoelektriska högtalare. Högtalaren försörjs passivt av kortets strömförsörjning, och för att få den att generera ljud behöver man enbart skicka ut fyrkantsvågor med rätt periodicitet till den via PB1 på Arduinon [1].

Enligt hårdvarubeskrivningen beter sig högtalaren bäst vid frekvenser omkring 3000–4000 Hz, och av den anledningen valdes tre stamtoner ut (G, A och B) [6] eftersom de råkade ligga lämpligt fördelade inom intervallet.

För att kunna generera fyrkantsvågorna med bra precision parallellt med spelets huvudprogram användes timerbaserade avbrottsrutiner. Tonen A har en frekvens på 3520 Hz och för att få en fyrkantsvåg krävs det att PB1 sätts på och sedan stängs av en gång inom den perioden. Det behövs alltså en *toggle*-funktion som anropas med dubbla frekvensen av tonen, i detta fall 7040 Hz.

Formeln nedan användes för att för varje ton ta fram ett jämförelsevärde för timern att avgöra när avbrottsrutinen skulle köras. Då en av processorns 8-bitars timers användes bör detta jämförelsevärde ligga mellan 0–255, och gärna nära 255 för att minimera avrundningsfel. Med det i åtanke visade det sig att ett prescaler-värde på 32 var lämpligt.

$$Räknarvärde = \frac{\frac{F_{CPU}}{Prescaler}}{\frac{F_{Ton}}{2}}$$

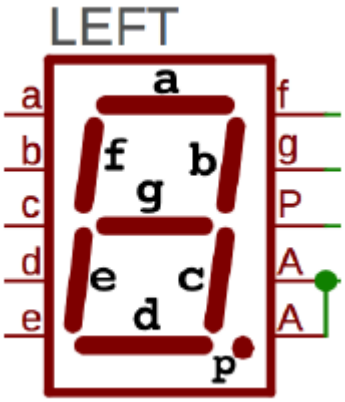
Ton	Frekvens (Hz)	Flippfrekvens (Hz)	Räknarvärde	Antal flipp under 0.05 sekunder
b''''	3951.07	7902.13	63.27	395.1
a''''	3520.00	7040.00	71.02	352.0
g''''	3135.96	6271.93	79.72	313.6

Figur 11: *Tabellen visar frekvenserna för de toner som användes, det uträknade räknarvärdet samt hur många gånger signalen måste flippas av och på får att tonen ska vara i 0.05 sekunder. Detta gäller för ett prescalervärde på 32 vilket ger räknarvärden lagom i intervallet 0–255. Nästa möjliga prescalervärde gav räknarvärden över 255 och mindre prescalervärde skulle ge sämre upplösning.*

För att kunna hålla tonen en längre tid konstruerades avbrottrutinen för att den vid anrop inte bara ska göra en toggle på signalen till högtalaren, utan den räknar även ner en inre 16-bitars timer vars värden lagras i SRAM. Utifrån detta var det bara att för varje ton skapa subrutiner som kunde anropas av spelprogrammet när man ville ha en specifik ton. Dessa subrutiner stod för att först ladda in konstanter för tonens frekvens och längd till timern och SRAM för att sedan sätta i gång räknaren och generera avbrott från den. Avbrottsrutinen hade som eget ansvar att inaktivera sig själv när längden för tonen hade nåtts.

3.3.5 7-segmentdisplay

För att visa spelarnas poäng i spelet används två 7-segmentsdisplayer. Dessa har varsin PCF8574 med adresserna $24 och $25 [1]. För att skriva en siffra till 7-segmentet görs en *lookup* på värdet mot en tabell (se **Figur 13**) med hexadecimala koder. Varje bit i denna hexadecimala kod motsvarar om ett segment på displayen ska vara tänd eller släckt och matchar kodningen som kan ses i **Figur 12** där bit 0–6 motsvarar segment *a–g* och bit 7 motsvarar punkten *p [1]*. Dessa koder skrivs ut via TWI



Figur 13: *Modell över 7-segmentdisplayen.*

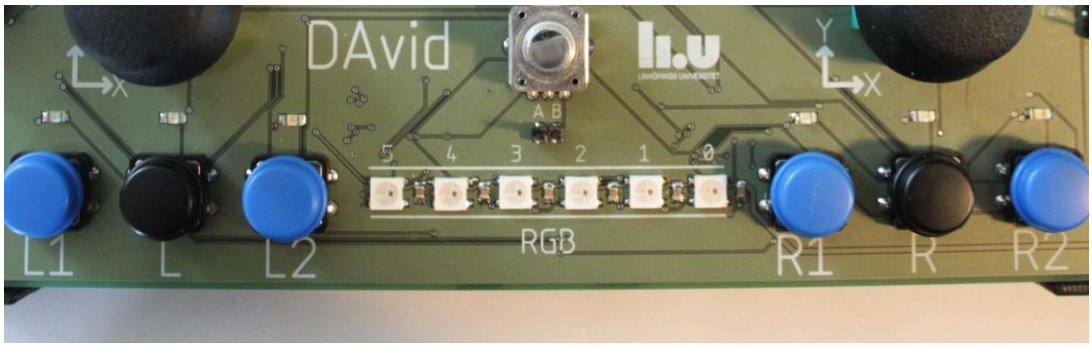
till I/O-expandern och ligger kvar där tills de skrivs över, vilket gör att det bara behövs göra en skrivning varje gång en spelare har fått ett poäng eller vid eventuell nollställning.

```
TAB_7SEG:
    .db   $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F,
$6F, $77, $7C, $39, $5E, $79, $71
```

Figur 12: *En tabell över de koder som representerar hexadecimal siffran 0–F. $3F = '0', $06 = '1' o.s.v. till 'F', där värdet i tabellen är ett åtta binärt värdet där en etta representerar att ett visst segment ska vara tänt.*

3.3.6 Knappar och Joysticks

DAvid-kortet har ett antal tryckknappar och två joysticks. Tryckknapparna som används i programmet heter L och R och visas på bilden i **Figur 14** [1]. Dessa är direkt anslutna till PD1 och PD0 och läses direkt från dessa pins. Vid knapptryck ger dessa pins en låg signal. Programmet använder L och R för att kolla om spelarna är redo. Vid knapptryck startar spelet.



Figur 14: *På bilden syns de tryckknappar som finns tillgängliga på DAvid-kortet. Knapparna L och R är det som har används till spelet och dessa är direkt kopplade till stift på Arduinon. Knapparna L1, L2, R1 och R2 fyller samma funktion med är kopplade till en I/O-expander som kommunicerar med processorn via TWI.*

Joysticksen avläses i stället genom en AD-omvandlare där en horisontal och ett vertikalt värde avläses [1]. Vänstra joysticken har kanalerna PC2 och PC3 för horisontella och vertikala värden och högra joysticken har kanalerna PC0 och PC1. I spelet avläses signalen och jämför den med konstanta värden för den vertikala riktningen.

```
call   READ_JOY_L_V
cpi    r16, $40        ; Gå ner
brcs   INC_PADDLE1
cpi    r16, $BF        ; Gå upp
brcc   DEC_PADDLE1
```

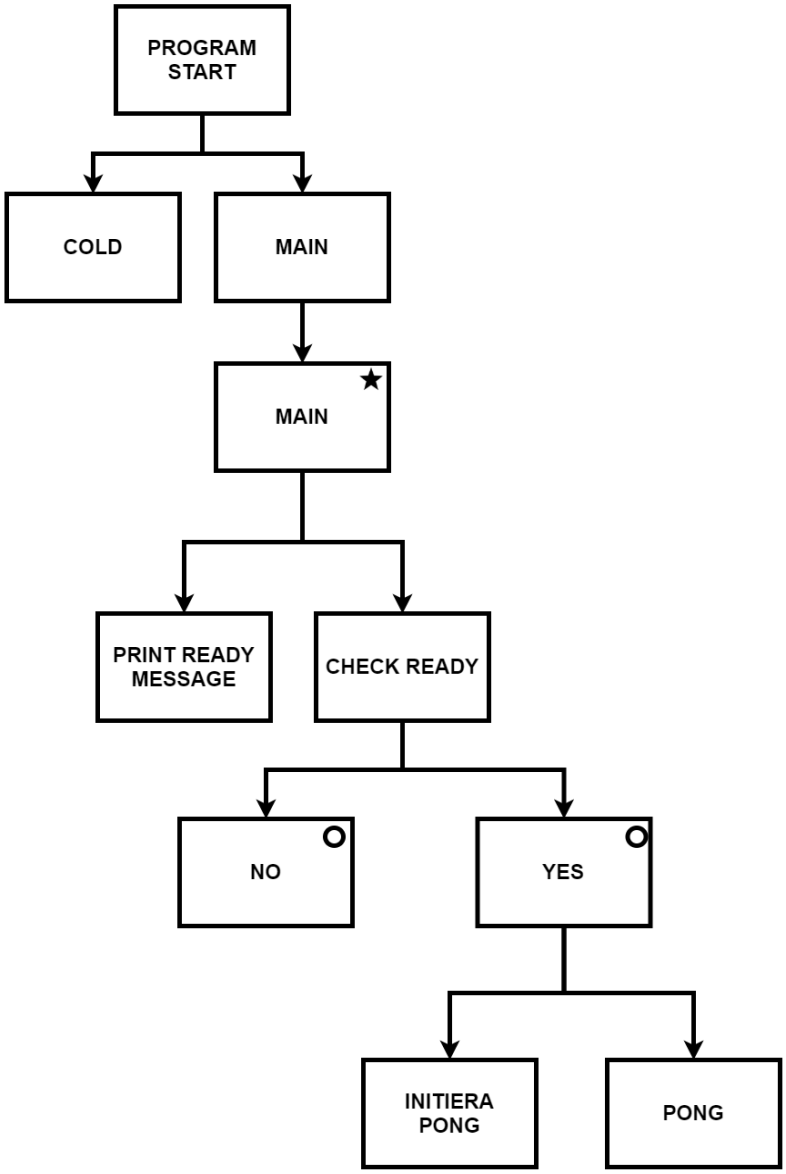
Figur 15: *Urklipp av kod för hantering av joystickens AD-signaler.*

I spelet jämförs värdena med det hexadecimala värdet $40 för att avgöra om planket ska flytta sig neråt då avbrottsrutinen kör. Plankets y-koordinat ökas då värdet från AD-omvandlaren är mindre än

$40. På samma sätt jämförs värdet $BF för att se om plankets y-koordinat ska minskas ett steg om värdet är större än $BF.

4 Programstruktur

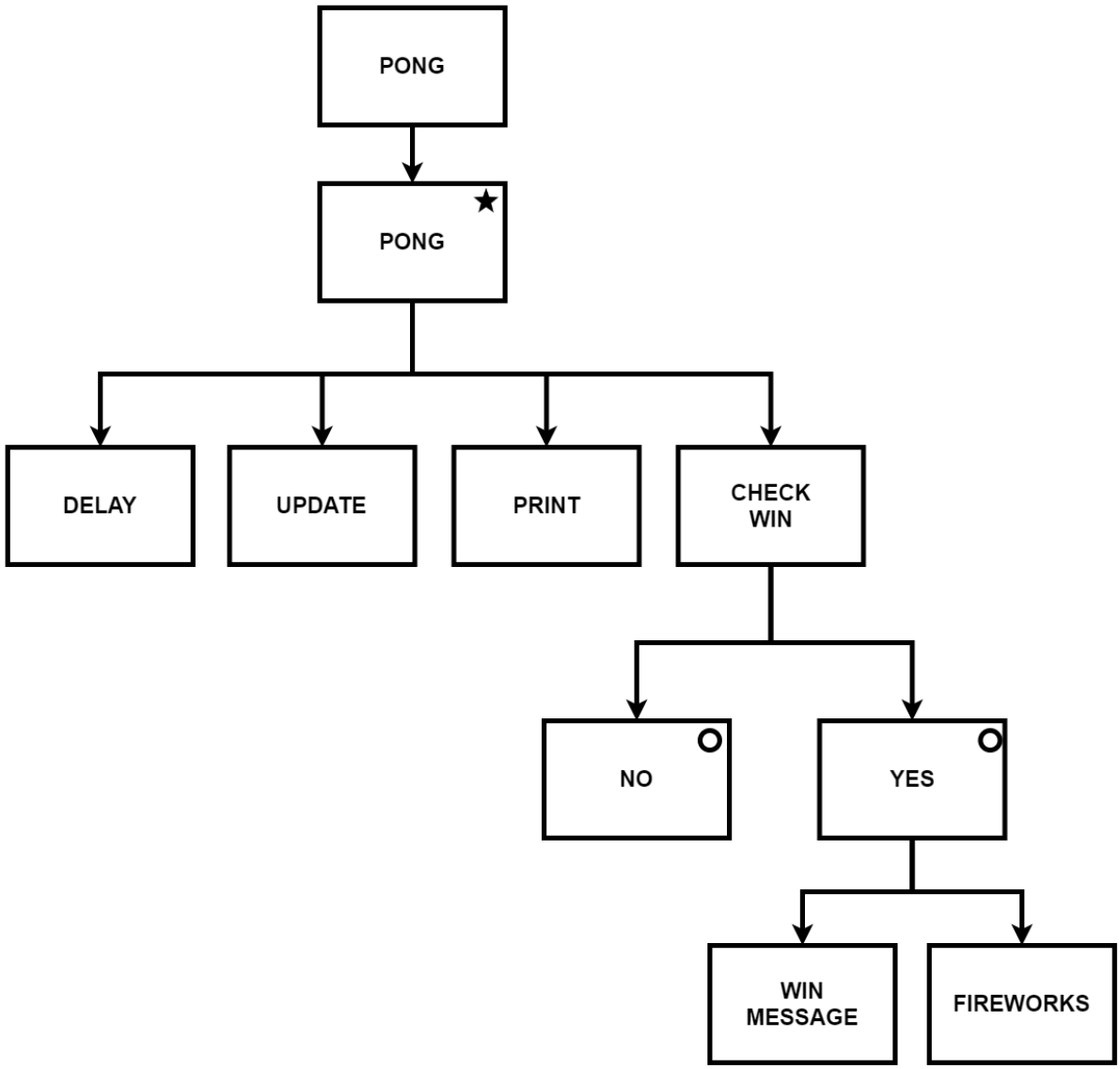
I denna del beskrivs en överblick över spelet Pongs struktur och minneshantering. Först kommer ett JSP-diagram över det generella programmet och sedan förklaras hur minnet är uppdelat som ett Spelbräde och Videominne och hur de används i programmet. Till slut kommer en mer detaljerad beskrivning av de största funktionerna i spelet Pong.



Figur 16: *JSP-diagram över en övergripande del av programmet. Programmet börjar exekvera med initieringar och står sedan och itererar runt* `PONG`.

I **Figur 16** är ett JSP-diagram över den stora delen av programmet. Exekveringen börjar med en `COLD-START` där hårdvara och till viss del minnet initieras för att spelet ska kunna startas korrekt. Efter hårdvaruinitieringen får spelaren ett välkomstmeddelande och ett val att starta spelet genom att trycka på en knapp.

Därefter påbörjas körningen av spelet Pong och spelets objekt sätts till ett lämpligt utgångsläge. Här startas också en timerbaserad avbrottsrutin som löper parallellt med spelets program. Avbrottsrutinens ansvar är att uppdatera positioner och data för boll och plank i spelets minne.



Figur 17: *JSP-diagrammet över* `PONG` *visar flödet för själva spelet.* `PONG` *är en loop som fortsätter iterera tills att en spelare vunnit. PONG börjar med att vänta i 30 ms innan den kör funktionen* `UPDATE` *som laddar koordinater över spelobjekten till videominnet. Sedan skrivs informationen till DA-matrix genom* `PRINT`*. Till slut kontrollerar* `PONG` *om en spelare vunnit och skriver ut ett vinstmeddelande eller fortsätter från* `PONG` *igen.*

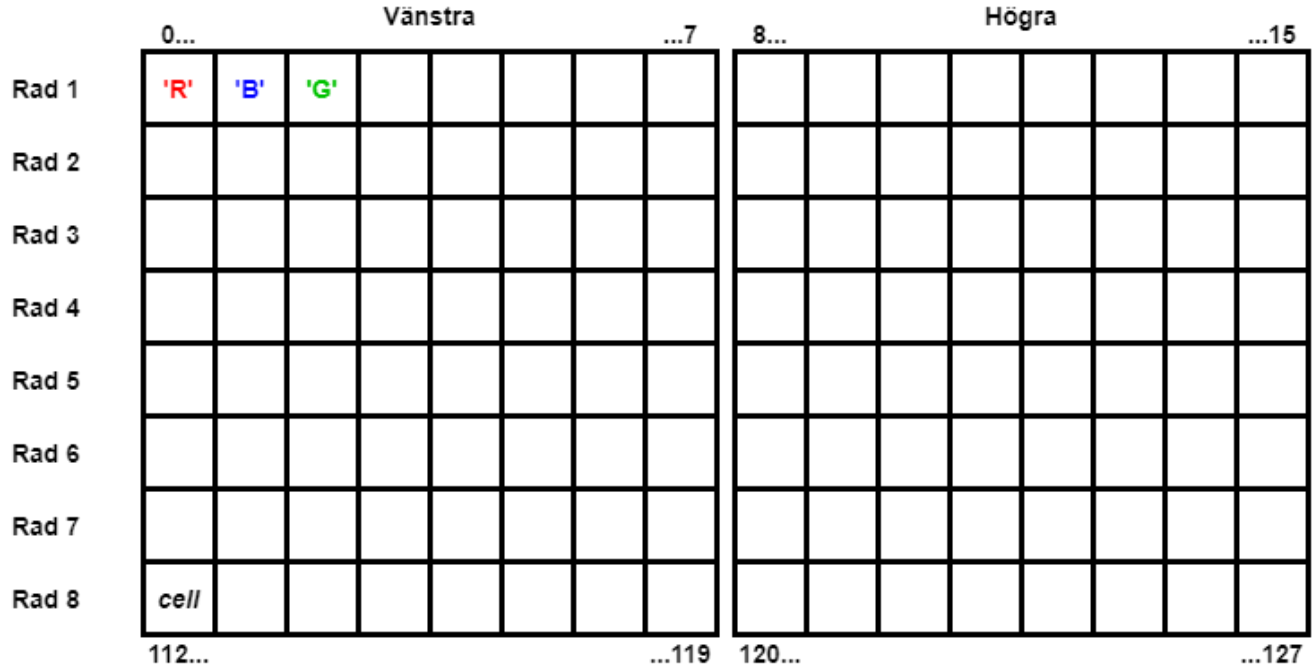
Spelloopen `PONG` som syns i **Figur 17** börjar med `DELAY`, en vänte-rutin som pausar programmet i 30 ms innan den fortsätter. Sedan uppdaterar programmet själva spelplanen med alla ändringar från avbrotten. När uppdateringen är klar skrivs detta ut till en DA-matrix (LED-matris). Slutligen kontrollerar programmet om någon av spelarna har vunnit och skriver ut ett vinstmeddelande eller fortsätter till `PONG`. Då en spelare vunnit hoppar programmet till `MAIN` och ett nytt spel kan påbörjas med ett knapptryck.

4.1.1 Minnesstruktur

I föregående del beskrivs strukturen av spelet. I denna del presenteras minnestrukturen. Minnet är uppdelat i ett Spelbräde och ett Videominne där Spelbrädet håller data för boll och plank. Spelbrädet översätts sedan till videominnet som skrivs ut till DA-matrix.

4.1.1.1 Spelbräde

Spelbrädet är det minne som används för att manipulera vad som ska visas på DA-matrix. Då ett plank eller boll ska byta position skrivs först förändringarna till Spelbrädet som sedan översätts till Videominnet. Spelbrädet är strukturerat som ett koordinatsystem där byte 0 är LED:en högst upp till vänster på DA-matrix.



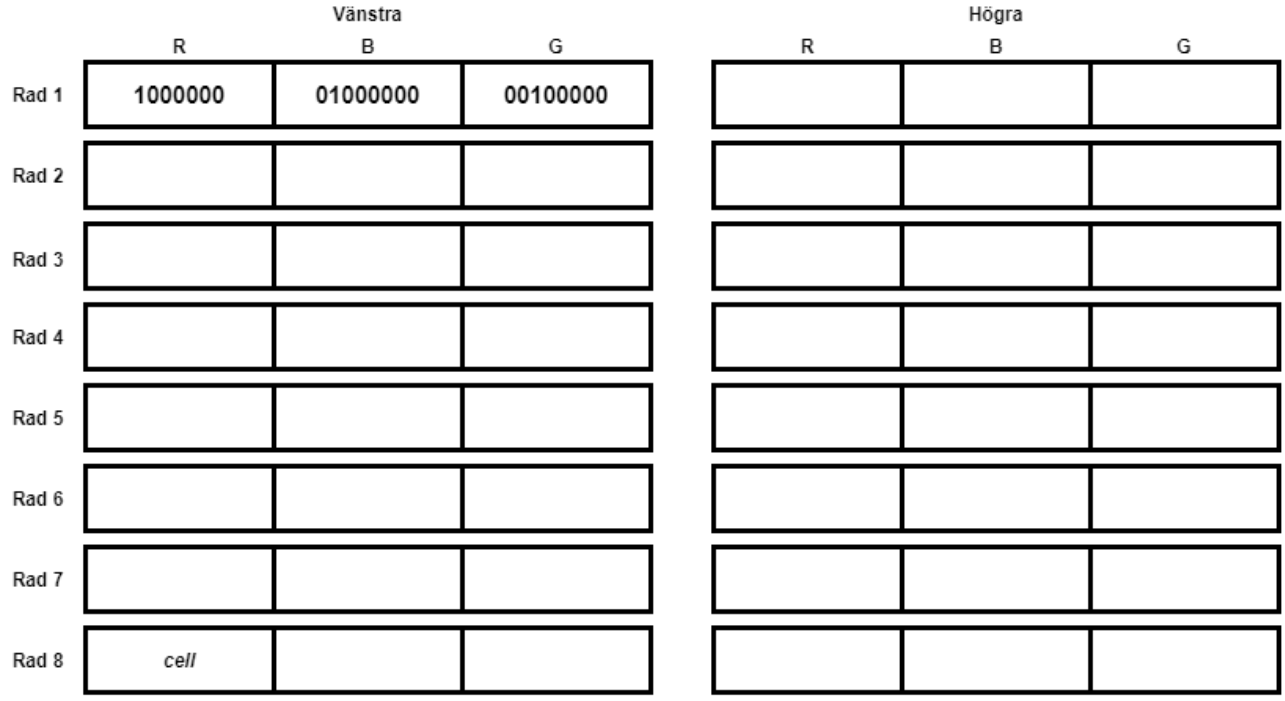
Figur 18: *Modell över Spelbrädet för Pong. En cell är en byte. Brädet går från byte 0 till byte 127. Varje cell innehåller data för färgen på LED:en i form av ett ASCII-tecken.*

I **Figur 18** ovan ser man en förenklad modell över hur Spelbrädet är strukturerat i minnet. Jämför man den med **Figur 7**, bilden på DA-matriserna, syns en tydlig likhet och det blir då smidigt att tänka att videominnet representerar ett koordinatsystem för displayerna. Raderna går från rad 1 till rad 8 där varje rad består av 16 bytes. Hela Spelbrädet har en storlek på 128 bytes som listade från 0–127. Varje rad är uppdelad i två halvor, Vänster och Höger. Dessa representerar vänster DA-matrix och höger DA-matrix. Varje cell innehåller ett tecken som motsvarar den färg som LED:en på motsvarande koordinat ska lysa i. I fallet **Figur 18** kommer de tre första LED-dioderna på rad ett att lysa rött, blått, och grönt i den ordningen.

4.1.1.2 Videominne

Videominnet är det minne som innehåller data som ska skrivas till DA-matrix. Till skillnad från Spelbrädet är det inte strukturerat som ett koordinatsystem då skrivningen till DA-matrix kräver ett annat format. Videominnet har 8 rader och varje rad är uppdelad som två delar för vänster och

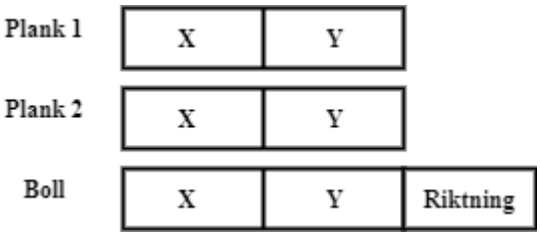
höger matris. Varje del är sedan uppdelad i tre bytes för färgerna röd, blå och grön. Byten håller sedan tillståndet för varje diod på raden i matrisen i varje bit. En etta i "R" byten kommer göra dioden på samma position som biten färgen röd. En kombination i R, G, B kan få dioden lysa i olika färger.



Figur 19: *Modell över Videominnet för Pong. En cell är en byte. Minnet går från byte 0 till byte 47.*

I exemplet i **Figur 19** kommer dioderna längst upp till vänster på rad 1 lysa rött, blått och grönt i samma ordning.

4.1.1.3 Component table



Figur 20: *En modell över hur spelets state är sparat i minnet. Varje cell i bilden är en byte.*

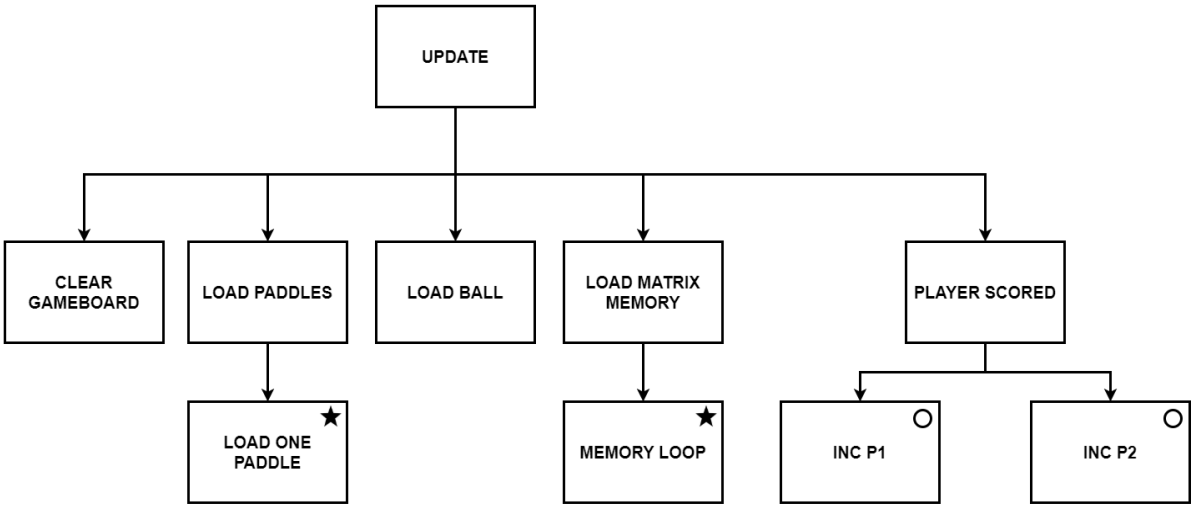
För att kunna ändra positionen för boll och plank körs en avbrottsrutin med jämna mellanrum för att uppdatera boll och plankens position. När avbrottsrutinen kör gör den en kontroll om spelarna har flyttat planken med joysticken och uppdaterar plankets position till ett minne som kallas för *Component Table*. På liknande sätt kommer bollens position att uppdateras varje gång

avbrottsrutinen kör. Bollens nuvarande position kan laddas från minnet och utifrån en riktningsvariabel beräknas den nya positionen.

4.1.2 Funktionsbeskrivning

Spelet har ett antal större funktioner som inte visas tydligt i det stora JSP-diagrammet. `UPDATE`, `PRINT` och avbrottsrutinerna är där största delen av programmet körs. Därför kommer dessa funktioner beskrivas lite noggrannare.

4.1.2.1 Update



Figur 21: *JSP över funktionen `UPDATE`. Slutsatsen av denna funktion är att `UPDATE` laddar ny information från spelets objekt till ett spelbräde och till slut laddar denna information till ett videominne.*

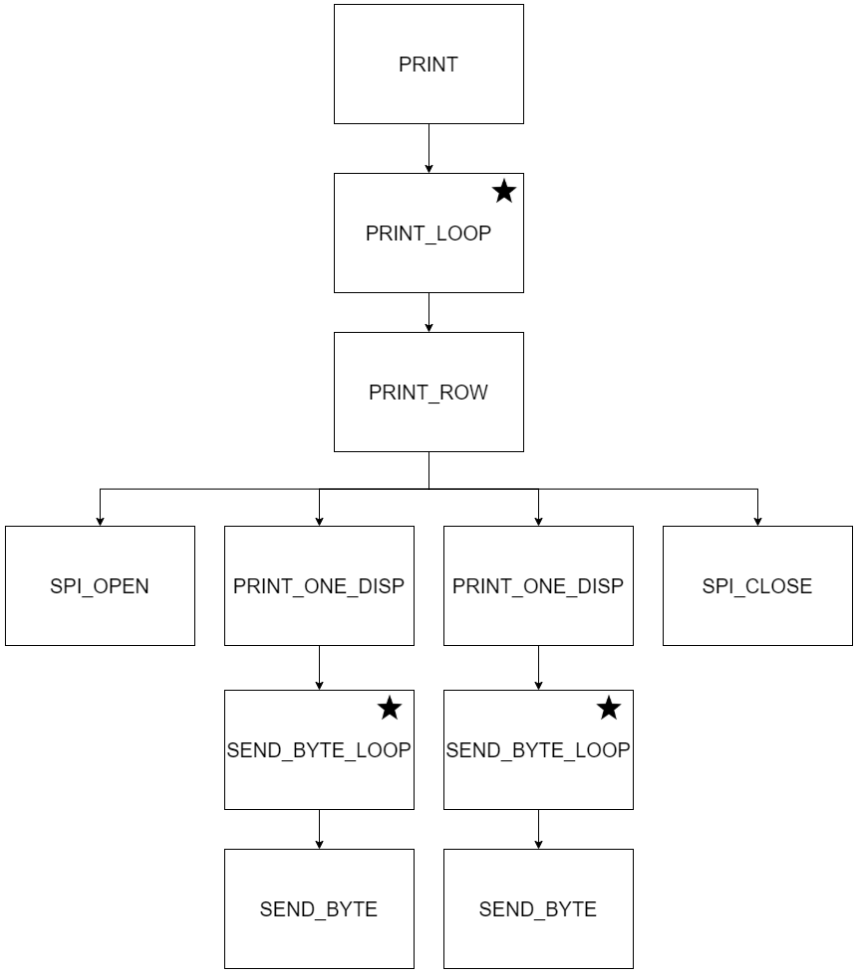
`UPDATE` är en funktion som körs efter delayen på 30 ms efter att Pong har initierats. `UPDATE` börjar med att skriva över alla bytes i spelbrädet med nollor. Detta görs för att sedan kunna skriva ut de nya koordinaterna till objekten boll och plank.

Därefter körs två *load*-funktioner som laddar boll och plankens dåvarande positioner från *Component Table* till spelbrädet. Dessa funktioner är uppdelade som `LOAD PADDLES` och `LOAD BALL`. Efter *load*-funktionerna körs funktionen `LOAD MATRIX MEMORY`, denna funktion laddar innehållet från spelbrädet som precis blivit uppdaterat till LED-matrisens videominne.

När skrivningen till videominnet är klar körs en avslutande funktion `PLAYER SCORED`. Den kontrollerar ifall en spelare fått poäng genom att kontrollera flaggor i *Component Table*. Om en spelare har gjort ett mål ökas dennes poäng och en kontroll över om spelaren har vunnit på maxpoäng. Då spelaren har vunnit sätts en *win*-flagga och bollens position återställs till ursprungspositionen. I fallet då en spelare inte vunnit sätts ingen *win*-flagga utan återställer endast bollens position.

4.1.2.2 Print

Funktionen som körs efter *PONG* är *DA PRINT MEM*. Denna funktion skriver alla bytes i videominnet till DA-matrix. Fördelen med videominnet är att man på ett enklare sätt kan gå genom minnet och skriva innehållet som det ligger utan att behöva göra hopp i data.



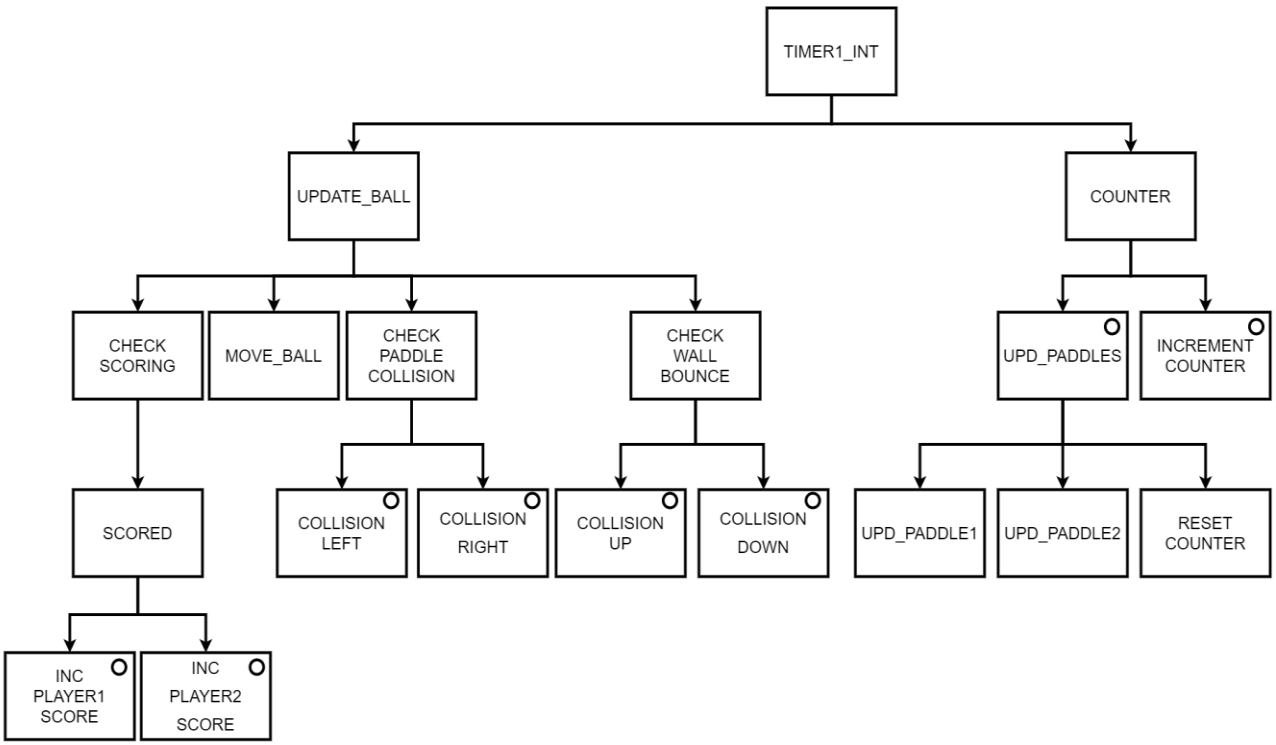
Figur 22: *JSP-diagram över hur skrivningen till båda DA-matrix ser ut. Först printas en rad över två DA-matrix där* *PRINT_ONE_DISP* *skriver ute varje bit till matrisen. Detta görs sedan på alla rader.*

I **Figur 21** visas ett JSP diagram över hur skrivningen till matriserna går till. Först körs en loop som går genom alla rader i matriserna. För varje rad öppnas SPI innan skrivningen sker till båda matrisernas rader. Sedan körs en loop som skriver varje byte till raden. Detta görs två gånger då datan skiftas in som beskrevs tidigare. Till slut stängs SPI och skrivningen körs igen på nästa rad o.s.v. tills alla rader är utskrivna.

Först ligger en loop där funktionen *PRINT ROW* körs. *PRINT ROW* i sin tur har ett antal funktioner som genom protokollet SPI skriver till DA-matrix. Eftersom spelet utnyttjar två matriser skrivs datat som om man skriver till en display men två gånger. Efter att kommunikationen till matrisen har öppnats kör funktionen *PRINT ONE DISPLAY* två gånger vilket skriver 32x2 bytes till matrisen. Sedan stängs SPI och *PRINT ROW* körs igen på nästa rad till att alla rader är skrivna.

4.1.2.3 Timer-avbrott

Parallellt med `PONG` körs även en *Timer, eller räknare,* som vid jämna mellanrum ger ett *interrupt* eller ett avbrott till mikrokontrollern. När avbrottet sker körs en funktion som uppdaterar bollens och plankens tillstånd beroende på om spelarna har rört joysticken.



Figur 23: *Avbrottsrutinen i diagrammet visar hur bollen och planken uppdateras.* `TIMER1_INT` *börjar med att kontrollera om en spelare gjort poäng och ökar spelarpoängen. Sedan flyttas bollen och kontrollerar om bollen har kollision med ett plank. Därefter kontrolleras om bollen har kollision mot en vägg. Då bollen rör sig i en annan hastighet än planken räknas en räknare upp för att vid jämna intervall uppdatera planken. Detta görs i* `COUNTER` *där en selektion sker om räknaren ska ökas eller om planken ska uppdatera sin position.*

Avbrottsrutinen, som presenteras som JSP i **Figur 23**, kontrollerar först om bollen har gjort ett mål i en av spelarnas sidor och sätter en poäng-flagga för spelaren som gjort målet. Därefter ändras bollens koordinat i *Component Table* utifrån bollens riktning. Efter att bollen har förflyttat sig kontrolleras om bollen har en kollision med ett spelarplank och ändra riktningen för bollen om en kollision har inträffat. Sedan utförs ett kollisionstest även för botten och toppen av skärmen och beslutar om den nya riktningen. Avbrottsrutinen körs alltså med jämna tidsintervall och förändrar tillstånden i *Component Table*.

5 Diskussion

I sin nuvarande form har projektet resulterat i ett fungerande spel som möter de skall-krav som angavs i kravspecifikationen, men det finns delar som man skulle kunna ha gjort annorlunda eller utvecklat vidare. Detta diskuteras kort i följande stycken.

En stor del av projekttiden lades initialt bara på att få dem grundläggande kommunikationsprotokollen och hårvarurutinerna att fungera. Det gjordes även delvis utan ett tydligt mål för användandet. En del av jobbet blev kanske i onödan och man skulle kunna ha sparat tid där genom att tidigare planera ut spelets generella upplägg. Detta ledde till att det blev mindre tid över till att bygga och polera spelets rutiner.

På grund av att tiden tog längre tid för skall-kraven än beräknat fanns det ingen tid för de inplanerade utökade kraven listade i kravspecifikationen. AI:n för enspelarläge insåg vi under projektets gång skulle ta upp en betydande del av tiden vilket gjorde att den blev utesluten. Det andra utökade kravet att kunna namnge spelarna hade inte varit särskilt svårt men tiden räckte inte till för detta.

Ett misstag som gjordes var att lägga för mycket ansvar i avbrottsrutinen för Timer 1. Detta gjorde att den blev för lång och krånglig. Det blev då svårare att veta om problem uppstod när avbrotten körde. Det vi skulle gjort annorlunda är att flytta ut så mycket kod som möjligt och endast sätta flaggor som bestämmer vad huvudprogrammet ska göra när avbrotten kommer.

Det tydligaste förbättringsmöjligheterna är pseudo-slumpgenereringen av bollens startriktning som introducerade en bugg. Den gör att bollen temporärt försvinner från displayen tills att någon spelare råkar få ett poäng. Det bör inte vara omöjligt att lösa problemet, men det skulle vara svårt att lokalisera felet då felsökningen hade blivit komplex eller skulle kräva mycket *backtracking* för att lösa. Då slumpgenerering av bollens riktning är viktigt för spelupplevelsen ansåg vi inte att det var värt att koden återställs till tidigare version som fungerade men var för förutsägbar för att spelet ska uppfattas som bra.

Vi hade även några buggar i DA-matrix som introducerades tidigt i projektet. En av dessa var skrivningen till DA-matrix som hade oväntade blinkningar på dioder och även att närliggande dioder på samma kolumner lyste svagt fast de inte skulle. Det löstes delvis med att sänka uppdateringsfrekvensen på matrisen. En annan bugg i matrisen var den översta raden som lyste starkare än de andra raderna. Detta blev tydligt då man styrde planken till toppen av skärmen och när bollen studsade i taket. Detta var en bugg som vi inte lyckades hitta problemet till. Vi misstänker att detta kan bero på hur skrivningen sker till matriserna men kan också vara ett problem hos hårdvaran.

Övrig finpolering på spelet skulle var att lägga till flera och mer komplicerade ljud till exempel i början och slutet av spelet. Vi skull också velat ha flera utskrifter, både på LCD:n och på båda DA-matrix. Det påbörjades stöd för animeringar på DA-matrix men det prioriterades ner och färdigställdes inte.

5.1 Slutsats

Spelet fungerar nu enligt kravspecifikationen och vi är nöjda över slutresultatet. Det finns dock mycket utvecklingspotential både när det gäller flera implementationer och lösning av alla buggar. Det har varit svårt att i ett lågnivåspråk som assembler få ihop en större helhet och samtidigt hålla det strukturerat. För att vara första gången vi har gjort ett omfattande projekt med många olika hårdvarumoduler som måste fungera tillsammans tycker vi att det gått relativt bra.

6 Referenslista

1. *Datorteknik DAvid Hårdvarubeskrivning*. Linköping: Institutionen för systemteknik; 2021.
2. ATmega328P, *8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*, Datasheet [Internet]. San Jose: Atmel Corporation; c2015.
3. Michael Josefsson, *Drivning av LED-matrisen DA-matrix*. Linköping: Institutionen för systemteknik; 2021.
4. *PCF8574; PCF8574A Remote 8-bit I/O expander for I2C-bus with interrupt, Product datasheet* [Internet]. NXP Semiconductors; 2013.
5. *HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)* [Internet]. Tokyo: Hitachi Ltd.; c1998.
6. Suits, Bryan. *Physics of Music, Frequencies for equal-tempered scale* [Internet] Houghton, Michigan: Michigan Tech University; 1998. [https://pages.mtu.edu/~suits/notefreqs.html, hämtad senast 25/04/2021]