UDPong - Pong over UDP

Digital projects (EDI021)
Lund University, Faculty of Engineering.

Thomas Eriksson, Samuel Skånberg

Abstract

The goal with this paper is to demonstrate a construction done with an ATMega16 which is a system-on-a-chip. We decided to do a construction that plays the famous game pong with a PC using UDP packets. This paper includes specification about the hardware being used, discusses problems we encountered and also gives a brief introduction to how the software works.

1 Inledning

I kursen "Digitala project - EDI021" så hade vi som uppgift att göra en konstruktion med hjälp av antingen en mikro-processor (motorola 68008) eller med en enchipsdator. Vi valde enchipsdatorn, närmare bestämt AT-Mega16, då vi fick mycket gratis eftersom vi slapp koppla in RAM-minne, EPROM, etc.

Det var en bra kurs där vi fick friska upp våra kunskaper i elektronik, samt programmera i C vilket är en tacksam paus från Java. Det var bara några få schemalagda timmar vilket innebar mycket personligt ansvar. Det var både för och nackdelar med det.

Vi valde att göra pong [1]. Med hjälp av en skärm, en potentiometer och en nätverksenhet så kan man spela pong med en PC. Spelet är som det klassiska pong där varje spelare har ett racket på sin sida som denne kan flytta upp och ner. Mellan dessa racketar studsar en boll. Spelet går ut på att träffa bollen och skjuta över på motståndaren så att denne missar bollen.

2 Krav

Innan vi började med arbetet skulle vi skriva en kravspecifikation på vad konstruktionen skulle klara av. Först var tanken att vi bara skulle ha en Wake-On-Lan-apparat som slog på eller av en relä så att man t.ex. skulle kunna fjärrstyra en kaffekokare eller något annat som var inkopplat på den relän. Vår handledare tyckte att det var en aning för litet och att vi skulle spinna loss mer. Då kom vi på saker som vi tyckte hade varit roliga

- Konstruktionen ska kunna visa en enkel bitmapsbild skickad över UDP
- Konstruktionen ska kunna spela pong med hjälp av en potentiometer, skärm och nätverksenhet
- Konstruktionen ska kunna sätta på eller slå av en relä med hjälp av Wake-On-Lan-paket

Vi fokuserade på pong. Bitmapsbilden hade varit enkel att implementera men på grund av brist på tid så ville vi först och främst fokusera på pong. Relän blev det inte tid till eftersom vi då hade varit tvungna att sätta oss in i elektroniken kring en relä vilket vi kände att vi inte heller hade tid till.

I slutändan kunde vår konstruktion spela pong över UDP vilket vi är mycket nöjda med. Det var den huvudsakliga uppgiften för den.

3 Hårdvara

3.1 Enchipsdatorn ATMega16

ATMega16 är en enchipsdator av Atmel. Den är 8-bitars och har en massa inbyggda finesser. Till vårt förfogande har vi 16 kb EPROM, 1 kb RAM-minne och en 16 Mhz CPU. De inbyggda finesser vi använde var:

- 2 portar (8 pinnar var) till en skärm
- SPI-pinnarna till en nätverksenhet
- A/D-omvandlingsenhet
- JTAG

3.2 LCD-skärm GDM12864

För att kunna spela spelet behövs naturligtvis en skärm. Vi valde denna skärm eftersom Tobbe Lundberg hade skrivit en bra rapport och bra källkod som vi trodde skulle komma till hjälp.

Skärmen är 128*64 bitar och har två kontroller till den, en för vardera halva av skärmen. Man kan skicka två sorters kommandon till den, instruktions-kommando och data-kommando. Instruktionskommandona var till för att välja var i skärmets minnet vi ville skriva och data-kommandona var till för att skriva en byte (8 pixlar i rad) på vald plats. Alltså, instruktionerna var till för att välja x- och y-koordinater.

3.3 Nätverksenhet ENC28J60-H

Vi ville hitta en nätverksenhet som man enkelt kunde koppla in och prata med hjälp av SPI. Det visade sig att det fanns flera sådana. Vi valde en krets som heter ENC28J60. Men det fanns en annan krets som heter ENC28j60-H som har integrerat en nätverkskontakt (RJ45) på själva kretsen och som bara har gnd, vcc och SPI-pinnar. Så för att slippa allt krångel det skulle innebära med att själv löda på nätverkskontakt, kapacitorer och annan elektronik för att driva nätverkskontakten så valde vi den enkla vägen och tog en ENC28J60-H.

Vi hittade en väldigt bra sida på nätet [2] som hade kopplat en ENC28J60 till en nätverkskontakt och skrivit kod till en ATMega8 för att styra den. Koden gick alldeles utmärkt att porta till ATMega16. Det var bara en fråga om att ändra 5-10 rader för att anpassa den till rätt port-nummer. Det gjorde ingen skillnad att vi hade en ENC28J60-H eftersom de bara använde sig av SPI-pinnarna ändå. Den koden var i sin tur kopierad från AVRLib, ett GPL-bibliotek för AVR-processorer [3]. "Sharing is caring" som man säger.

3.4 Spänningsdelare LP3855

Eftersom ENC28J60-H drivs med 3.3 volt så var vi tvungna att ha en spänningsdelare, i vårt fall valdes LP3855.

3.5 Potentiometer

För att flytta vårt racket upp och ner så behövde vi en potentiometer för en sådan finns i originalpong. För att kunna läsa av värdena så använde vi oss av A/D-omvandlaren som finns på ATMega16. Det räckte med en pinne på port A. Först var vi oroliga över att vi inte skulle kunna använda oss av resten av pinnarna på port A men det visade sig att vi både kunde läsa värden på pin 0 på port A samt använda resten av pinnarna som vanliga in/ut-pinnar.

4 Speluppbyggnad

Eftersom vi bara hade en ATMega16 till vårt förfogande så gjorde vi två klienter, en till vår enchipsdator och en till en PC.

När båda klienter är anslutna börjar spelet. Spelet initieras med ett startpaket från PCns sida. En boll skjuts därefter ut från ATMega16 och den skickar ett paket till PCn med vilka koordinater bollen skickades ut med. ATMega16 beräknar därefter banan och skickar bollens uppdaterade position till PCn, vilken ritar ut bollen och paddeln på skärmen. Bollens position och huruvida den träffar racketena beräknas helt och hållet av ATMega16. I en tidigare design tänkte vi att båda klienterna var och en för sig skulle hålla reda på bollens position. Detta skulle dock ofta leda till att någon av klienterna räknade fel och spelarna fick en felaktig bild av spelet.

För att få förbättrad spelupplevelse så skickar vi också iväg paket när vi flyttat vårt racket så att motståndaren kan se vart man är och då anpassa sina skott efter det.

5 Programvara

5.1 ATMega16

När programmet startas på enchipsdatorn så initierar vi skärmen, nätverksenheten och börjar med A/D-omvandlingen. A/D-omvandlingen genererar avbrott för varje avslutad omvandling. A/D-omvandlaren ställdes in att omvandla på en frekvens av 128 kHz.

Därefter skickar A/D-omvandlaren ett avbrott och ATMega16 hoppar till avbrottsrutinen ISR, där värdet från potentiometern läses av.

Efter att potentiometervärdet erhållits kallas racket-ritningsrutinen `draw_racquet()` där racketet ritas om om det ändrat position sen senaste gången potentiometervärdet beräknats. Vidare beräknas bollens nya position av rutinen `draw_ball()` och bollen ritas ut.

När allt det är klart så sätts en ny omvandling igång och CPUn kan fortsätta där den var innan.

När CPUn inte är i avbrottstrutinen så körs huvudloopen. I huvudloopen anropas kontinuerligt en rutin som lyssnar efter nätverkspaket och när det är gjort skickar den tillbaka ett paket.

5.2 Nätverkspaket

I paketen som vi skickar mellan klienterna finns följande information

- Speltyp. Om värdet är 0 så ställs enheten in på att börja spela en ny omgång pong. Om värdet är 1 ställs enheten in på att visa en bild. I andra fall ignoreras värdet
- Spelarens racket-position i höjdled
- Bollens x - och y-position. Detta skickas bara av ATMega16 eftersom det är den som är ansvarig för beräkning av bollens position.
- Om ATMega16-klienten träffade (värde 1) eller missade bollen (värde 0). Andra värden ignoreras. Samma information finns gällande PC-klienten
- De olika klienternas poäng

Vi var tvungna att göra denna läsning att skicka och läsa paket i par. Innan gjorde vi så att vi bara läste paket i main-loopen och skrev paket i avbrottsrutinen. Men då blir det problem om vi hoppar till avbrottsrutinen när vi håller på att läsa ett paket. Alltså löste vi det på ett fult men enkelt sätt. Det skickas alltså väldigt mycket data som inte används hela tiden.

I nuläget så fungerar bildvisningen delvis men inte helt.

5.3 PC-klienten

PC-klienten är den klient som sätter igång ett spel genom att sätta speltypen till 0 och därmed säga till ATMega16 att pong ska startas. Innan den får ett sånt paket sitter den bara och lyssnar efter paket utan att göra något mer.

PC-klienten är skriven i C och med hjälp av det fria (LGPL) grafikbiblioteket SDL (Simple DirectMedia Layer). Eftersom det är ATMega16 som räknar ut det mesta, som bollens position, poängställningen och liknande, så gör klienten inte så mycket. Den läser från tangentbordet för att kunna flytta sitt racket och den ritar upp sitt racket, bollen, motståndarens racket och poängställningen.

Tanken var från början att vi också skulle kunna skicka bitmapsbilder till ATMega16. Vi utvecklade ett program som gör om 64*128 pixlar stora bmp-bilder till vårt eget filformat där en bit motsvarar en pixel, svart eller vit (bmp_converter). Parallellt med det utvecklade vi också ett program som emulerade vår LCD-skärm, också det i SDL (dead_viewer). Ett program som skickade över filen (med speltyp satt till 1) utvecklades också. Överföringen gick bra men när vi skulle rita upp bilden så gick det inte som vi hade tänkt.

6 Problem under kursens gång

6.1 Skärmen

Vi hade en del problem med skärmen. Vi tog skärmen för att vi hade hittat kod från ett tetrisprojekt av Tobbe Lundberg som såg bra ut. Men det gick ju (som väntat) inte att bara kopiera koden. Så efter ett tag gav vi upp det och läste databladet istället. Vi kollade på timing-diagrammet och försökte sätta våra pinnar i rätt ordning och med rätt timing. När vi väl hade kommit igång med det så kunde Tobbes kod vara till nytta.

Vi märkte att det nog var fel i databladet. Där det stod RS i en tabell tror vi att de menade D/I. När vi tittade på Tobbes kod bekräftades våra misstankar.

Sen fick vi problem med att skicka instruktioner som sätter adressen där vi ska skriva data till skärmen. Om vi bara skrev data så kom det upp en massa linjer och pixlar på skärmen men när vi försökte skriva på y-adress 0 och x-adress 0 så kom ingenting upp när vi skrev data. Men efter att vi tittat på Tobbes lösning så ändrade vi RST-signalen och då gick det bättre.

Vid projektets slutförande kunde skärmen vid flera tillfällen förskjutas en eller flera pixlar, där den sista pixelraden visades på andra sidan skärmen. Vad detta berodde på hittade vi inte.

6.2 AVR Studio

Vi var beredda på att AVR-studio skulle vara lite buggit eftersom vi hade en dålig erfarenhet av Xilinx studio som också är en IDE som man använder för konstruktion av inbyggda system. Våra farhågor blev besannade gång på gång:

I början av projektets gång kraschade programmet när det tappade kontakten med JTAG-dongeln. Det var dock aldrig några filer som blev korrupta eller liknande så krascherna gjorde ingen skada.

Senare i projektet drabbades vi av att AVR Studio kontinuerligt tappade kontakten med JTAG-dongeln och fick startas om.

En stor fördel var dock JTAG-stödet i AVR-studio. Vi använde en del breakpoints och tittade på några variabler vi hade i watchpoint. Det var riktigt trevligt. Men annars använde vi inte debugging-möjligheterna så mycket. Helst skulle vi vilja ha använt Linux och avrdude samt avrice men vi ville komma igång med programmerandet snabbt så vi använde AVR studio. Om vi skulle göra ett till projekt med någon AVR så skulle vi använt oss av avrdude från början. Men vi är glada att vi ändå använde AVR studio eftersom fler på institutionen kan det och kunde hjälpa oss. Men det hade känts bättre om man hade kunnat gå över till avrdude när man hade lite koll på

hur själva ATMega16 fungerade.

7 Literature

References

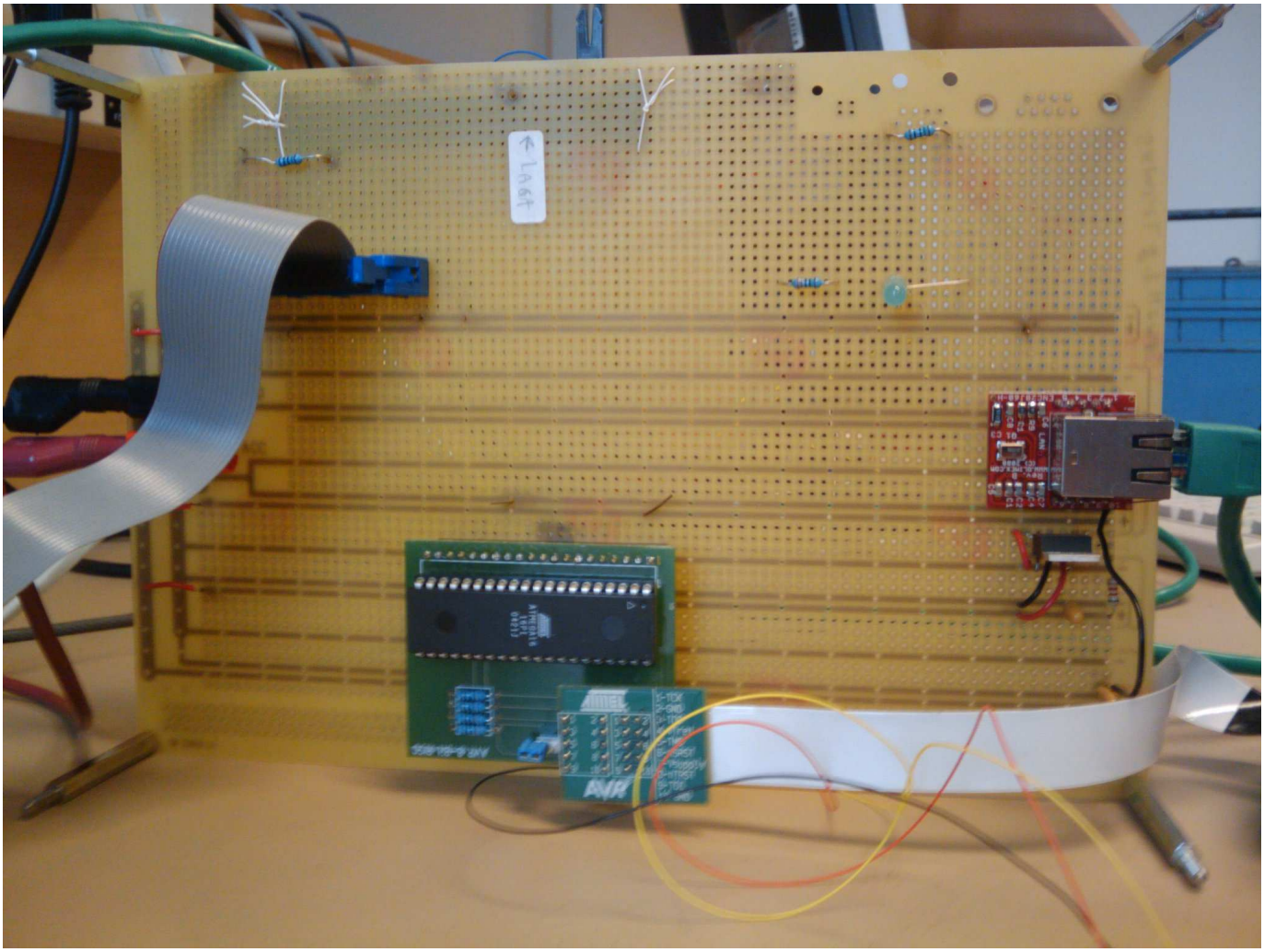
[1] *Pong*
http://en.wikipedia.org/wiki/Pong

[2] *An AVR microcontroller based Ethernet device*
http://www.tuxgraphics.org/electronics/200606/article06061.shtml

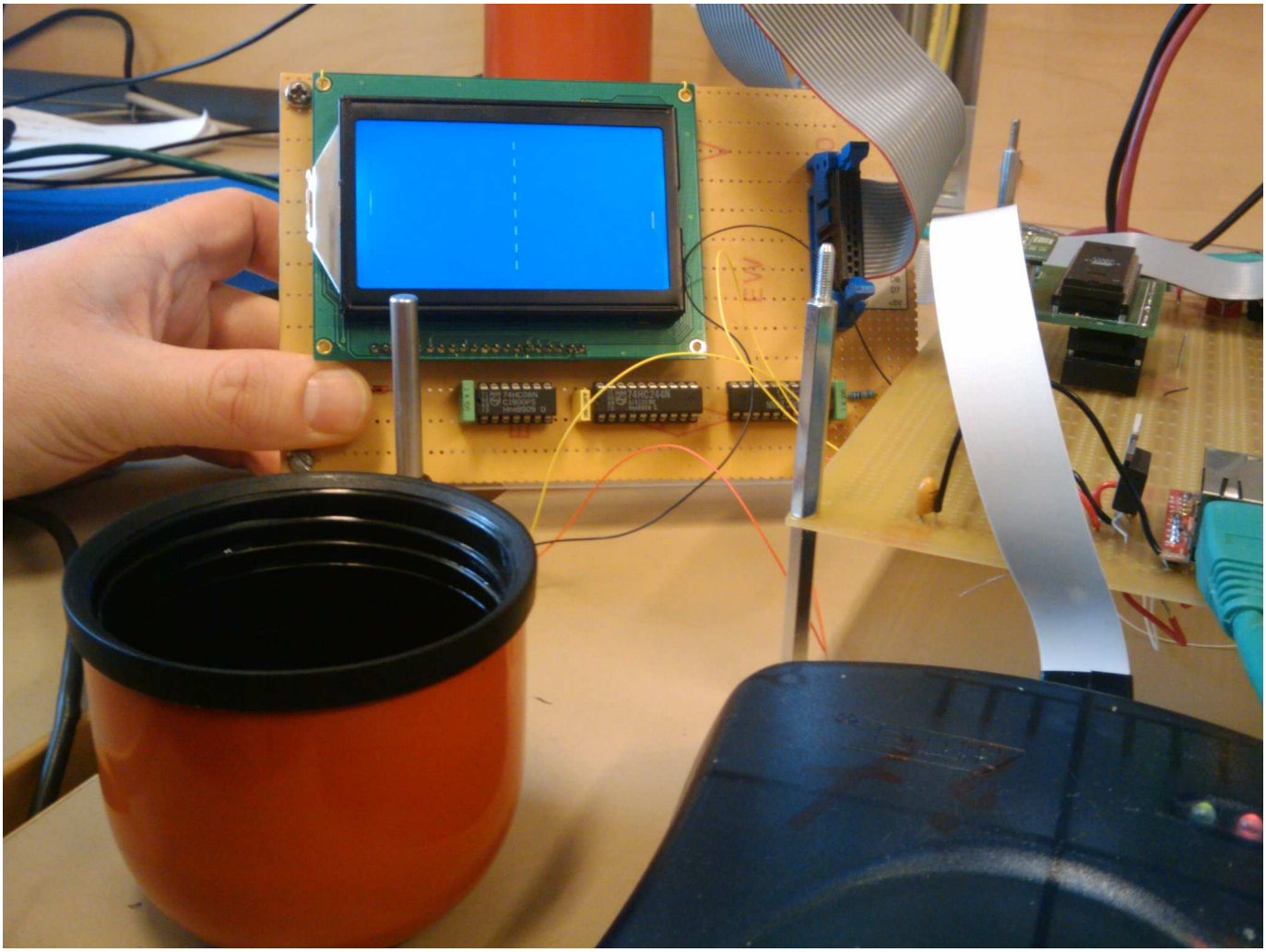
[3] *AVRlib*
http://hubbard.engr.scu.edu/embedded/avr/avrlib/

8 Bilder

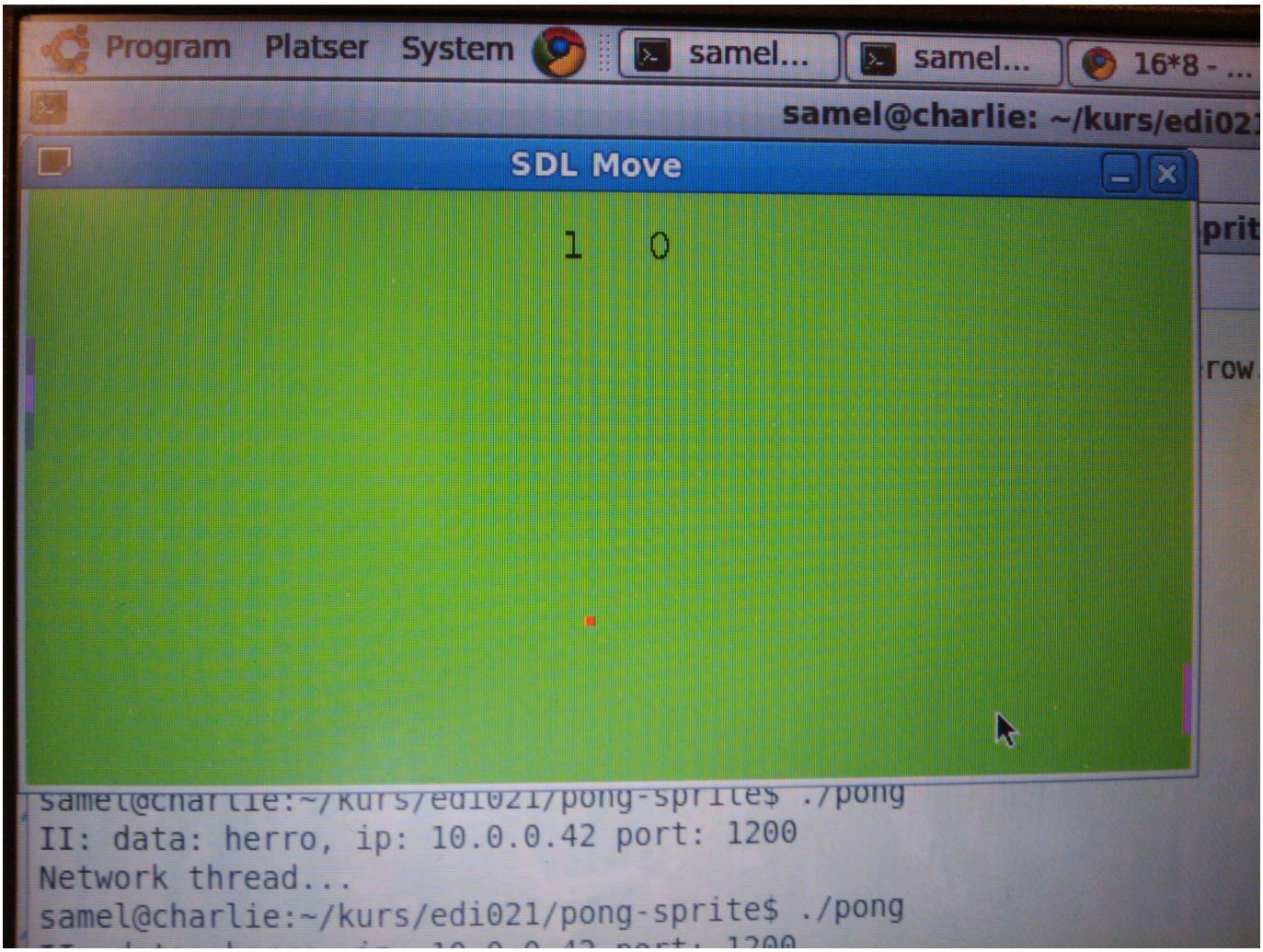
8.1 Konstruktionen in action



8.2 Pong på ATMega16



8.3 PC-klienten



Bilaga: koplingschema