Kortrækkende trådløs styreenhed

Anders Dubgaard
anders@mip.sdu.dk

3. oktober, 2005

Ingeniørhøjskolen Odense Teknikum
Vejleder: Anders Stengaard Sørensen

Resumé

Denne rapport beskriver et interface mellem en Sony® PlayStation Controller gamepad og en microcontroller og hvordan informationerne fra gamepad'en kan overføres vha. radiokommunikation til en anden microcontroller. Dette kan bruges til motorstyring og fjernstyring. Desuden beskrives et USB -interface til en microcontroller, og hvordan et program på en almindelig PC med en USB-port kan gøre brug af informationerne fra gamepad'en. Det lykkedes at få hul i gennem så basen fx. kan generere et PWM signal baseret på de analog joysticks og at fjernstyre musikprogrammet XMMS med et perl-script.

Indhold

Figurer

Kapitel 1

Indledning

Motivationen bag for dette projekt var at finde en relativ billig og stabil analog fjernstyring til fx. en robotarm eller en motorstyring. “Hjemmelavede” analoge inputenheder er ofte meget ustabile og industrielle løsninger er typisk dyre. Men en PS controller , som jeg vil kalde for en *gamepad* i denne rapport, har mange fordele, bland andet er det robust, da det er bygget til børn, og da det har været på markedet i flere år, er det også gennemtestet. Udover de 14 knapper, har det fire analoge axer, en x- og en y-akse på hvert af de to små tommelfingerbetjente joysticks. til fx. motorstyring. En ulempe er, at Sony® har valgt ikke at offentligøre informationer omkring gamepad’en og dets protokol, hvorfor denne information skal fremskaffes ad andre veje, fortrinsvis via internettet, og måske med en del trial-and-error udvikling til følge.

At forbinde en microntroller til en PC’er er typisk sket med en RS-232 kommunikation til PC’ens seriel porten. I dag produceres der dog stadig færre PC’ere med seriel port, dette gælder især for laptops. I stedet er de udstyret med indtil flere USB-porte og derfor ville det være interessant at få microcontrolleren til at kommunikere med PC’en på denne måde.

1.1 Problemformulering

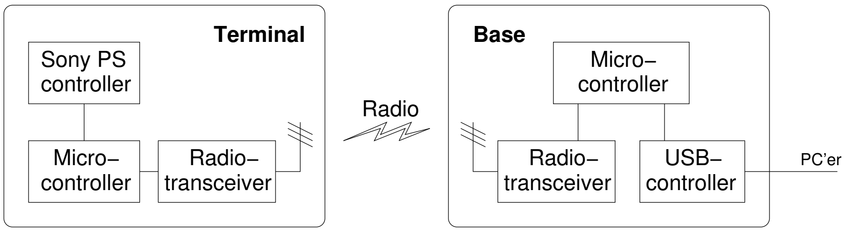
I mange situationer kan det være ønskeligt at have en trådløs styreenhed, fx. til mobil robot. Derudover er er en microcontroller meget alsidig og kan interface mange forskellige typer elektronik så dette projekt kan give grundlag for nye projekter og anvendelser. Blandt andet kan projektet bruges som en fjernstyring til en PC.

Det ønskes undersøgt om et microcontroller-baseret radiolink kan aflæse en gamepad, videresende denne information og bruge det i den anden ende, uden at der opstår mærkbare forsinkelse for brugeren.

1.2 Lidt om valg af komponenter

Fra starten af projektet lå valget af komponenter temmeligt klart;

Atmels AVR microntrollere, Atmega8 og -128, og Chipcons CC1000 var udbredt i studiemiljøet og derfor et naturligt valg. Valget af USB-controlleren,



Figur 1.1: Blokopdeling af systemet

nemlig FTDI's FT232BM, skete på grundlag af Gestur's erfaringer i hans speciale plus at der fandtes drivere til Linux. FTDI'en var lidt dyrere end andre USB-controllere men understøtter bl.a Linux, også i bit-bang-mode.

1.3 Læsevejledning

GND betyder jord og VCC betyder driftspænding, der er 3.3V. I nogle diagrammer forekommer der 5V, som tages tages fra USB-bussen. Dette kan gøres med et almindeligt USB type B stik uden brug af en USB-controller. Det bruger jeg som strømforsyning til terminalen i test-opstillingen (se billedet på forsiden TODO) og det er derfor, der et USB-stik tilkblet terminalen i opstillingen. CC1000 bruges løst om det valgte radiokit.

1.3.1 Kildekode

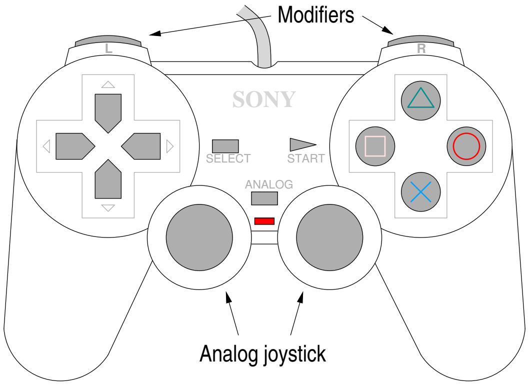
I bilag A har jeg inkluderet størstedelen af den kildekode, der er blevet udviklet under projektet. Denne kode findes også på CD'en bagerst i rapporten, i `/src`-kataloget. Dér ligger også nogle filer med mere generel kode som fx. UART-opsætning, som ikke er inkluderet i rapporten. På CD'en findes datablade og application notes fra litteraturlisten, i `doc`.

Projektets kildekode er frigivet under GNU GPL[1] version 2 eller nyere.

[1]GNU General Public License, `http://www.gnu.org`, i filen `src/COPYING` på CD'en.

1.3.2 Symboler

Playstation



Figur 1.2: Sony PlayStation Controller, aka gamepad.

En PS controller har i alt 15 knapper og 2 små analoge joysticks med hver 2 akser og 1 knap indbygget, se figur 1.2. Resten af knapperne er de fire retningsbestemte til venstre ($\leftarrow$, $\rightarrow$, $\uparrow$ og $\downarrow$), de fire knapper med symbolerne $\triangle$, $\times$, $\circ$ og $\square$ til højre, de fire modifiers foran (R1, R2, L1 og L2), og de to funktionsknapper, Select og Start. Det giver i alt 16 knapper, da Analog-knappens funktion er foruddefineret og vi ønsker at gøre brug af denne funktion.

Kapitel 2

Analyse

2.1 Kravspecifikation

Fra starten er IR-kommunikation valgt fra da vi ønsker at kunne sende data rundt om hjørner og gennem vægge, hvilket IR-dioderne ikke kan, da de skal kunne "se" hinanden.

Udviklingen af systemet vil foregå på Linux-platformen og det har en smule betydning for valg af enkelte komponenter. Blandt ander har jeg ikke taget IOT's egen 186-baserede kit til overvejelse, da det er meget tæt knyttet til DOS-platformen. I tidligere projekter har enkelte studerende udviklet til det under Linux, men ikke uden besvær. Det gælder også for HC11'eren, der har været brugt tidligere i undervisningen på IOT.

- Aflæsning af PS Controller med en AVR, inklusive de analoge akser.
- Trådløs transmission af data fra gamepad'en til en anden AVR.
- Interface fra AVR til en PC'er via USB.

Derudover kan systemet evt. udvides med

- Power-down mode, så terminalen holder længere på et batteri.
- Batteri og opladekredsløb i terminal, så den bliver helt trådløs
- Berøringsfrit "Activation Circuit"

2.2 Interface til Sony(R) PS Controller

Som udgangspunkt vil vi aflæse gamepad'ens tilstand, dvs. hvilke knapper der er aktiveret og værdierne for hver af de analoge akser. Dette skal gøres ofte, så man ikke føler, at man skal vente på et tryk på en knap "kommer igennem" til microcontrolleren og så ofte at opløsning på de analoge sticks ikke bliver så grov så man føler det "hakker", så tit at det føles kontinuert at betjene.

Derudover er det ønskeligt at den tilsluttede microcontroller kan aktivere de analoge joysticks, som ellers sker manuelt med Analog-knappen, således at systemet kan (gen-)aktivere dem, når det er krævet.

Fremover vil jeg skrive "analog mode" når der henvises til "Analog Red Mode".

Værre vil det være på de analoge akser, da et stort delay vil gøre en eventuel motorstyring grov, og ikke så smidig som man ellers kan forestille sig.

Eftersom Sony ikke har offentliggjort nogle specifikationer på gamepad'en har det været nødvendigt at finde informationen på internettet. En god kilde er McCubbin McCubbin [1998], der bl.a. beskriver benforbindelser på tilslutningsstikket og den serielle protokol samt en implementaionen i Motorola assembler til MC68HC11'eren.

2.2.1 Protokol

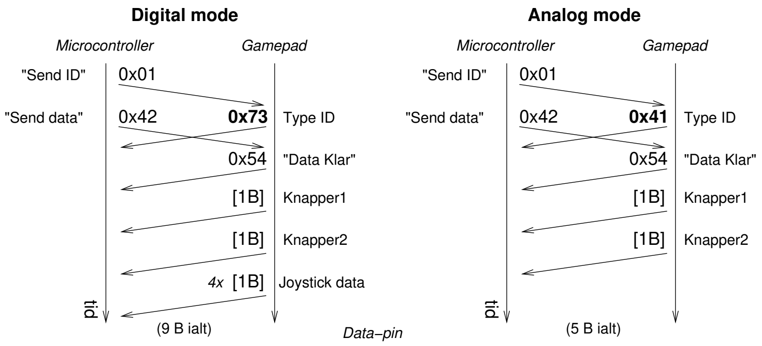
En tovejs protkol giver mulighed for feedback fra basen til terminalen, fx. som ændringer i parametrene for radioforbindelsen pga. af dårligt signal. Der er basalt to måder at lave tovejs kommunikation med CC1000-chippen, nemlig efter master/slave-princippet eller at have to CC1000'ere koblet til mcu'en.

En envejs protokol er mere simpel og dataopsamling udemærket til dataopsamling hvis den kombineres med fejltektion, som fx. CRC16.

Pakkestørrelsen kan være fast eller variabel. En fast pakkestørrelse er mere enkel mens

en variabel pakkelængde er meget fleksibel, som fx. Torvmark [2002b].

Vi vælger da en envejs radioforbindelse med CRC16, med fast pakkestørrelse, da det opfylder systemets krav uden at blive for kompliceret.



Figur 2.1: Definition af Sony Playstation Controllerens protokol

I dette afsnit beskrives hvordan microcontrolleren kan aflæse den serielle kommunikation, gamepad'en benytter sig af.

Kommunikation til og fra gamepad'en sker i byte-størrelser, med den mindst betydende bit først (LSB). En bit skrives på den nedadgående clock-kant og læses af den modtagnde enhed på den opadgående flanke, for at signalet kan nå at falde til ro.

Denne transmission ville for en gamepad i analog mode være forlænget med 4 bytes, nemlig stick'ens udslag i hver retning på de to joysticks. Først kommer højre stick horisontalt (0xFF er fuldt højre udslag), så vertikalt (0xFF er opad). Derefter det samme for venstre stick. Stick'enes knapfunktion transmitteres som bit 1 (højre) og bit 2 (venstre) i byten nr. 4.

Type identifikation

Der findes flere forskellige typer controllere til Sony Playstation, som rat og mus, men i dette projekt ser vi kun på de almindelige controllere, der ligner figur 1.2. De kaldes Analogue Controller in Red Mode og har type-ID = 0x73. Når gamepad'en ikke er i analog mode, giver den samme type-ID som en almindelig, digital gamepad uden de analoge sticks, kaldet Standard Digital Pad, nemlig type ID = 0x41.

ID'et bruges til at tolke den modtagne bytes rigtigt, da knapper og udslag ikke overføres samme steder i bitstrømmen for alle ID'er.

2.3 Radiokommunikation

Dette system kræver kun overførsel af en begrænset mængde data, nemlig de op til 7 bytes fra gamepad'en.

Man kan bruge frequency hopping og spread sprectrum men det er ikke nødvendigt her.

Paritetetscheck fanger kun cirka 50% af teransmissionsfejl, så det er et dårligt valg. Man kunne bruge redundans, men til dette formål er det bedre at bare at smide pakken bort og vente på en ny, da gamepad'en tit aflæses og at de data hurtigt forældes. Derfor vælger jeg CRC, nærmere bestemt CRC16. Der findes en rigtig god implementation af Barr [2002]. CRC16 fanger alle fejl der er mindre end 17bit og 99.998% af fejl på mere end 16 bit, ?. Som det fremgår af Andreassen [2005], leverer et sæt CC1000'ere fuld signal styrke over 30 m med 0 dB forstærkning og over 100 m ved 10 dB i fri luft.

2.4 USB-interface

Dette er undersøgt i Gestur Palsons speciale [Palsson, 2005] med gode resultater, og jeg har valgt at basere min løsning på hans. Jeg har ikke valgt præcis samme løsning, da kravene for dette system kan opfyldes af en RS-232 forbindelse. Problemet ligger som sagt i den fysiske forbindelse, hvorfor en USB-controller, der kan emulere en serielport, er ganske passende.

Jeg har således valgt en DLP-USB232M, der benytter FTDI's FT232BM USB-controller og opfylder disse krav. Da den simulerer en serielport, stiller det ikke yderligere krav til MCU 'en, der blot ser det som en almindelig USART . Andre fordele er at den kan interface 3.3V logik, hvilket er en stor fordel, da CC1000-chippen kræver dette. Vi undgår derfor at have en ekstern level converter i kredsløbet.

Under Linux understøtter driveren til FT232BM-chippen kun "Virtuel Com Port", dvs. at et program ser forbindelsen til MCU'en som en serielport.

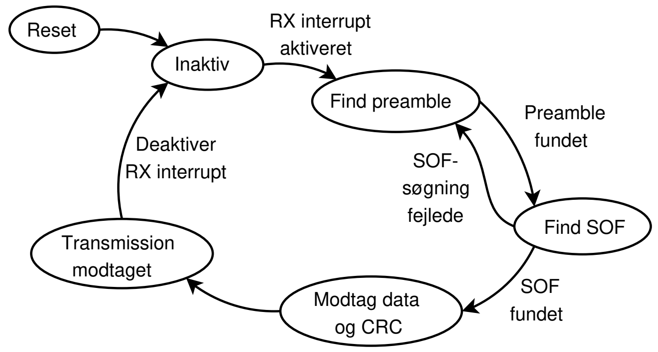
Derudover kan FT232BM også bruges i mode, der fx. kunne erstatte ISP -adapteren. Direkte USB-understøttelse og bit-banging understøttes p.t. kun under Windows.

Den kan bl.a. bruges som en virtuel COM-port, dvs. at den emulerer et USART-interface. Dette er smart da det ikke stiller ekstra krav til MCU'en; har den brugt RS232 før, kan den også bruge USB-controlleren. Det understøtter Linux-driveren dog endnu ikke.

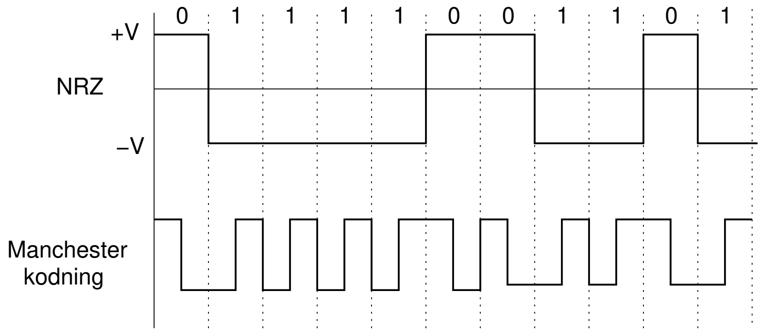
Kapitel 3

Design

3.1 Radiokommunikation



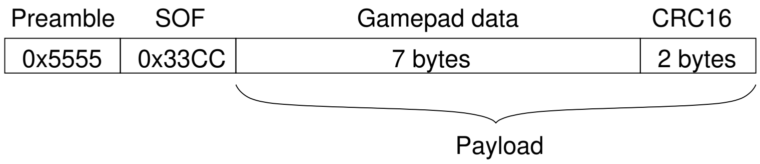
Figur 3.1: Flowdiagram for modtagerrutinen.



Figur 3.2: Manchester kodning har lavere hastighed end NRZ, men synkronisering er garateret pga. mindst et skift ved hver bit.

3.1.1 Protokol

Basal pakke opbygning med CC1000, Torvmark [2002a]. Preamble (0x5555 01010...), Start-of-Frame (0x33CC)

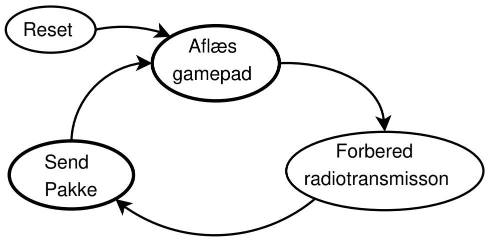


Figur 3.3: Pakkestruktur, CRC16 tilsidst.

3.2 Gamepad

Den rutine, der aflæser gamepad'en, skal være interrupt-styret, således at den kan køre i baggrunden og ikke tage opmærksomheden fra microcontrolerens egentlige opgave. Målet er at kunne aflæse gamepad'ens tilstand ca. 30 gange i sekundet, hvilket vil være ganske brugbart. Aflæsningsfrekvensen må ikke være for lav da man så vil opleve at skulle vente på at en knap aflæses.

Det vil være upraktisk at gemme hver enkelt bit som en byte, da vi har begrænset hukkommelse i vores system. Derfor må vi have en måde at samle 8 bits til en byte, der så gemmes. Dette kan gøres på flere måder men en simpel måde ville være at have buffere til de to transmissionskanaler og så læse eller skrive en bit på hver klokkant. Der kan stadig sendes andre kommandoer til gamepad'en; kommando-byten skal blot sættes ind i Command-bufferen før transmission påbegyndes. Dette forenkler vores algoritme en del da en stor del af programmet vil blive at sætte klokkanten ned, skriv en bit og derefter at sætte klokkanten op, læs en bit. Derudover skal transmissionen dog først påbegyndes og efter niende og sidste byte, afsluttes.



Figur 3.4: Flow for afsender enheden.

3.2.1 Aktivering af "Analog Mode"

Det har været meget sparsomt med oplysninger om hvilke kommandoer, der kan sendes til gamepad'en, fx. for at aktivere joystickene. Jeg fandt dog en sekvens af kommandoer, der skulle sætte gamepad'en i analog mode på `http://www.lynxmotion.com/images/html/build034.htm`.

Metoden er at sætte gamepad'en i "config mode", dernæst sende kommandosekvenser til konfiguration af forskellige indstillinge og til slut at send "exit config mode" til gamepad'en. Således skulle det være muligt at låse gamepad'en i den analoge mode, så de to joysticks altid er aktive.

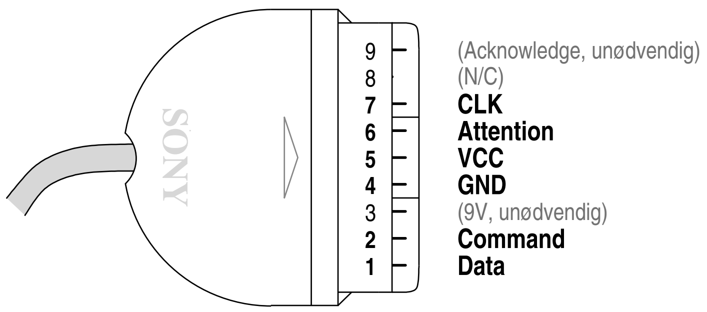
```
0x01 0x43 0x00 0x01 0x00                          ;Config Enable

0x01 0x44 0x00 0x01 0x03 0x00 0x00 0x00 0x00 ;Set Mode and Fix(Analog

0x01 0x43 0x00 0x00 0x00 0x00 0x00 0x00 0x00 ;Config Exit
```

Jeg har prøvet at bruge disse kommandosekvenser, men da det ikke umiddelbart fungerede, valgte jeg ikke at arbejde videre med det, da det ikke er vigtigt for projektets centale problemstilling. PS Controllerens type aflæses allerede, så man kan i fx. en robotcontroller vælge at stoppe en bevægelse hvis gamepad'en ikke er i analog mode, dvs. at typen ikke lig `0x73`.

På samme site (`lynxmotion.com` er der en oversigt [1] over mulige kommandoer (dog uden angivelse af en eventuelt rækkefølge for anvendelse). Disse sekvenser er tilsynelande til PlayStation 2 men da PS Controllere kan bruges både til PS1 og PS2, må man antage at må man antage at protokollen er ens.

3.2.2 Benforbindelser

For at forbinde en microcontroller til en gamepad'en kræves der GND, VCC og 4 I/O-ben; 3 med retning mod gamepad'en og 1 til aflæsning af den serielle kommunikation, se figur 3.5. Ifølge McCubbin [1998] er det nødvendigt med en pull-up modstand på Data-pin'en (pin 1). Desuden kan gamepad'en drives med VCC lig 3.3V eller 5V. Acknowledge-benet (pin 9) viste sig at være overflødigt og pin 3 er spændingen til Dual-Shock motorerne, som ikke bliver benytet, hvorfor ingen af disse pins skal forbindes til microcontrolleren.



Figur 3.5: PS Controllerens benforbindelser

Data Data fra gamepad'en til microntrolleren. For eksempel en bitstrøm, der repræsenterer knapperne tilstand, som svar på kommandoen `0x01`.

Command Data fra microntrolleren til gamepad'en. Byte'en `0x01` betyder ifor eksempel "send alle knappers tilstand".

[1] `www.lynxmotion.com/images/files/ps2cmd01.txt`

Attention Aktiv lav. Sættes lav før kommunikationen med gamepad'en påbegyndes og høj igen ved afslutning.

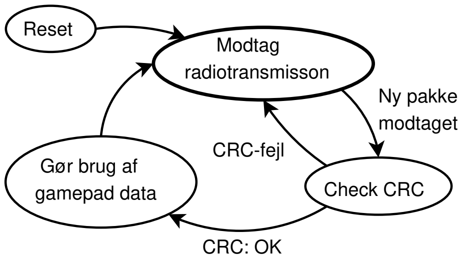
CLK Klokken, der driver den serielle kommunikation.

På microntrolleren kan der vælges 4 vilkårlige ben, såfremt retningen kan sættes som beskrevet.

3.3 USB

Som USB-interface til microcontrolleren har jeg valgt DLP-Designs DLP-USB232M, se også http://www.dlpdesign.com/dnlda for diagrammer og kildekode.

Dette lille demoboard har vist sig at være meget let at bruge, hvis det er det man vil. Man kan ogsa benytte andre drivere for at tilgå USB-bussen, med alt hvad der dertil hører af både godt og ondt, som fx. højere overførsels hastighed, multi-drop bus, op til 500mA strøm men også endnu flere standarder, der skal holdes rede på. Mulige drivere til DLP-USB232M er:



Figur 3.6: Flow for modtageren.

3.4 Virtuel Com-Port

DLP-USB232M kan bruges som en "Virtuel Com-Port", det vil sige at operativsystemets driver får enheden til at ligne en ganske almindelig RS-232 forbindelse fra begge sider. Det er en stor fordel, da den dækker flest operativstystemer og ikke kræver noget særligt kendskab til USB-standarden. Dette kan være en fordel i mindre projekter, hvor tiden eller projektets omfang er begrænset eller at man ønsker at gøre brug at tidligere erfaringer med kommunikation via serielporten. Man har dog ikke muligheden for at kommunikerer med selve USB-controlleren, FT2323BM-chippen (se FTDI Ltd. [2005]), og derfor kan denne metode ikke bruges til fx. at omprogrammere microcontrolleren vha. bit-banging.

3.5 D2XX

FTDI har frigivet en driver til Linux, der giver programmer adgang FT232BM-chippen via USB-porten. Desuden er der startet et projekt, der udvikler en fri D2XX-driver til Linux fra bunden, nemlig *libftdi*.[2] Ulempen er dog at der ikke

[2] http://www.intra2net.com/opensource/ftdi/

findes en D2XX-driver til andre systemer end GNU/Linux og Windows.

Ud fra disse overvejdelser vælges den virtuelle COM-port som interface mellem mcu'en og PC'en og det falder godt i tråd med, at der i dette projekt ligges vægt på enkelthed og bred anvendelighed.

Kapitel 4

Implementation

4.1 Radiokommunikation

Antenne: 14.6 cm ($\frac{1}{4}$ af bølgelængde) pga 433MHz - se Chipcon [2003], side 8. Loop-antenner bør have 50 Ohms impedance. En kvart-bølge ledningsantenne kan bruges direkte ved at fjerne pins 1, 2 og 3 fra Port 2 på CC1000-boardet og så lodde en ledning med den rigtige længde på.

4.1.1 Konfiguration af CC1000

Implemnetation ang. CC1000:

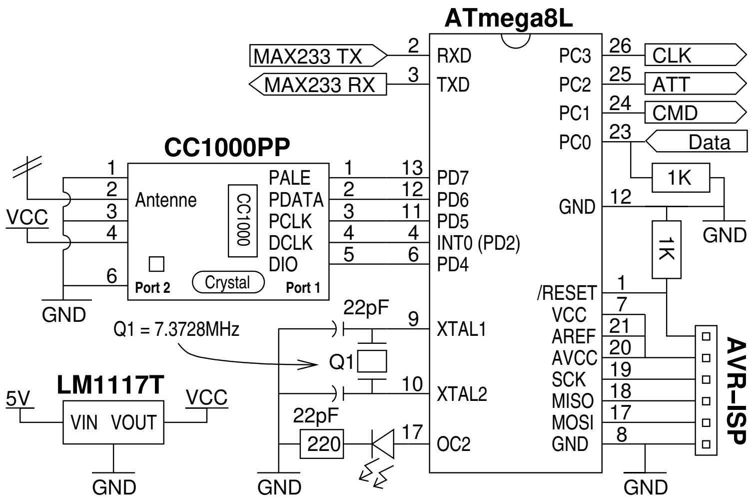
I Torvmark [2002b] leverer Chipcon C-kode, der skullle gøre det let at bruge CC1000'eren med en AVR microcontroller. Men der er ikke taget højde for avr-gcc, så tilpasning af filerne er nødvendig. Dette problem har [Andreassen, 2005] løst.

Chipcons AVR-kode (fra [Torvmark, 2002a]) er tilpasset så det er muligt for en udvikler at definere de nødvnedige register værdier i en tredje fil, som så inkluderes. Andreassen [2005] har desuden skrevet et perl-script, der konverterer en register fil til to header-filer, så det er let at ændre register-værdier, fx. i forbindelse med ændring af baud rate.

Scriptet, `smartrf2header.pl`, (og et tilhørende wrapper script, `chbaudrate.sh`) er inkluderet på CD'en i /src/scripts.

De filer er genereret med `smartrf2headerfile.pl` ud fra "register"-filen fra *SmartRF Studio*. og inkluderes direkte i programmet.

4.2 Gamepad-interface



Figur 4.1: Diagram for terminalen. terminal.c

4.2.1 Benforbindelser

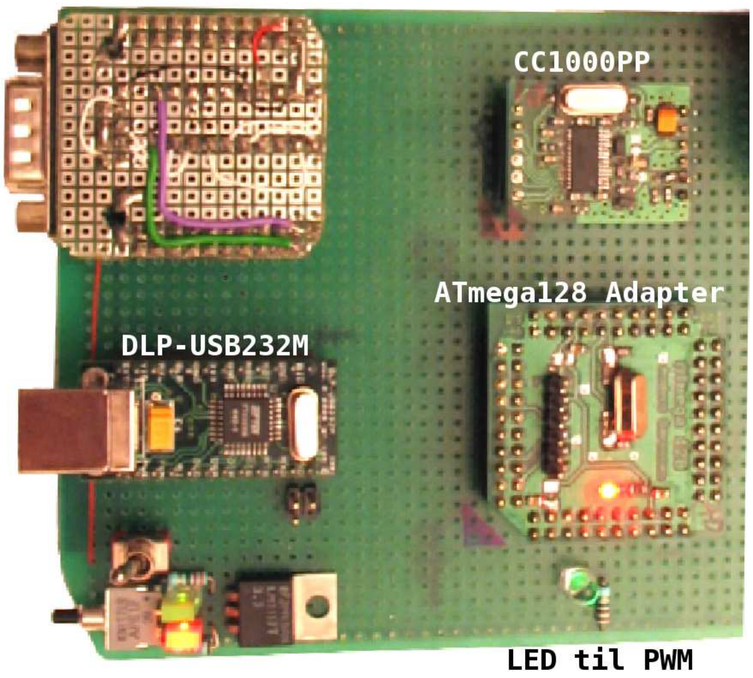
På ATmega8'eren har jeg valgt at føre DATA ind på PC0, CMD på PC1, ATT på PC2 og CLK på PC3, hvor CMD, ATT og CLK skal sættes til outgoing direction:

Port C pin directions

```
#define CMD  PC1  /* code   */
#define ATT  PC2
#define CLK  PC3  /* clock     */
#define ATT  PC0
...
DDRC = (1<<CMD)|(1<<ATT)|(1<<CLK);
```

4.2.2 Interrupt-rutine

Jeg har valgt at ignorere gamepad'ens ID under aflæsning, og dermed altid aflæse 6 date bytes. Skulle gamepad'en være i digital mode, går data-benet højt efter femte byte og dermed aflæses 0xFF for hver af de sidste fire bytes. Gamepad'ens type gemmes dog, så en applikation har muligheden for at tjekke gyldigheden for data i de sidste 4 bytes. Dette gør koden mere simpel og det vil primært være PS Controllere, der understøtter en analog model, der kobles til systemet.



Figur 4.2: Det endelige print, som virker.

4.3 MCU ⇌ PC

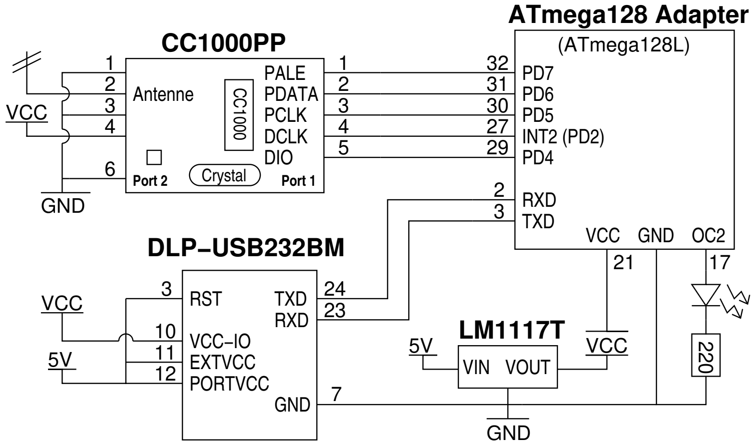
Idet at jeg bruger USB-controlleren som "Virtuel Com-Port", er det næsten kun nødvendigt at tænke på protokollen mellem enderne, da DLP-Design har nogle gode eksempler i deres bruger manual. Min implementation bygger på figur 3, side 7 i DLP Design [2002], således at komponenterne og DLP-USB232M'erens I/O-interface på boardet kører på 3.3V, der bliver reguleret fra de 5V DLP-USB232M'en kan levere fra USB-bussen.

Til protokollen har jeg valgt at lade basen sende data fra terminalen som ASCII-kodede byte-værdier i samme rækkefølge, som de gemmes i structen i microcontrollerne. Det er ikke så effektivt, men man kan let følge med i hvilke data der bliver sendt og hvad de indeholder.

En sådan linje kunne f.eks. se således ud:

Eksempel på data overførsel fra basen til en PC

```
011100111111111111111111011010100111111110000001100011111\^M
```



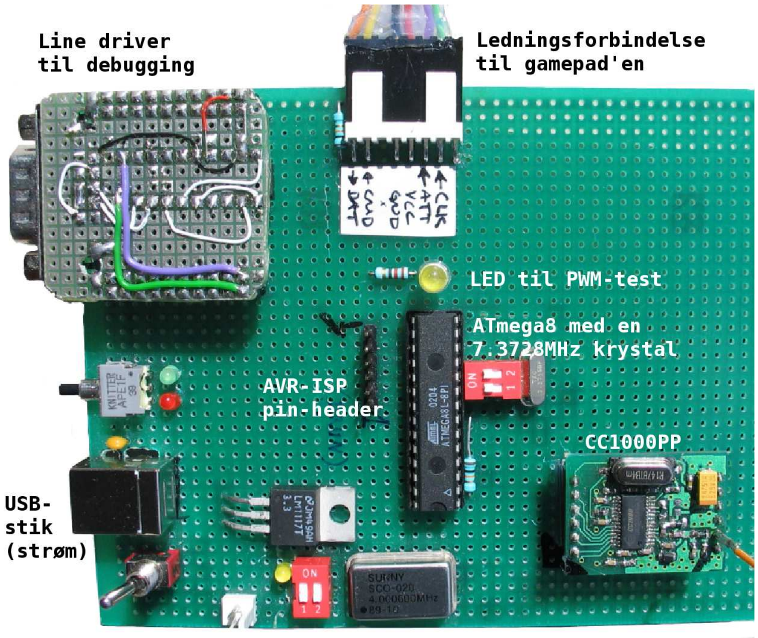
Figur 4.3: Diagram for modtageren, impementeret i base.c.

4.3.1 Fjernbetjening af et program på en PC

På CD'en (og på `http://www.dubgaard.net/projects`www.dubgaard.net/projects) findes også det perl-script, der muliggør at gamepad'en via demoboardene, kan bruges til at fjernbetjene XMMS[1]-programmet med. Det er ment som et proof-of-concept program, så valget af protokollen til PC'en faldt på det mest simple, nemlig at den psx-struct, der overføres via radioforbindelsen, skrives ud i klar teksti og afsluttes med et newline-tegn (`0x10`). PC'en aflæser således hver byte som 8 ASCII-tegn i på den virtuelle COM-port. Disse skal konverteres før de kan bruges som hver talværdier og derfor faldt valget på at bruge perl. Samtidig eksisterer der et perl-mmodul, dvs. bibliotek, så det er let at interface til XMMS.

En bedre måde ville være at lave aflæsningsprogrammet i C og sende data binært med lille protokol. Til protokollen kunne man fx. genbruge idé

[1]Grafisk musikafspiller til X11 på Unix systemer.

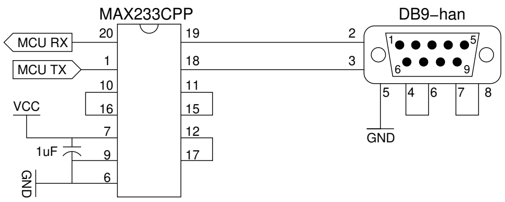


Figur 4.4: Det endelige print, som virker.

4.4 Debugging

På en microcontroller har man ikke umiddelbart samme mulighed for at debugge sit program med udskrifer til skærmen som på en PC, men man kan printe over den indbyggede UART til en PC's serielport (com-port) med en *line driver*. Grunden er, at RS-232 standarden specificerer et logisk 1 som +V og logisk 0 som -V, hvor V kan variere fra 3V op til 24V. Med en enkelt strømforsyning på 3.3V eller 5V har microcontrollere normalt ikke adgang til -5V, så for at kommunikerer med en PC via serielporten skal man bruge en line driver.

På IOT findes der som regel er par forskellige modeller, men da jeg skulle bruge en, var det kun Maxims MAX233CPP-kreds, IOT havde på lager. Det er en fire-kanals line driver, 2 sæt send/receive, som kan køre på 3.3V, se Maxim [2005]. Den er dog er en lidt dyrere end den MAX232, der normalt er tilgængelig, men kræver til gengæld ingen eksterne komponenter og dermed er den god i situationer, hvor der kun er lidt plads frit på et print. Den er koblet op som i



Figur 4.5: Opkobling af MAX233CPP line driver.

et “typical operating circuit”, der er kort beskreet på side 21 i databladet.

Kapitel 5

Test

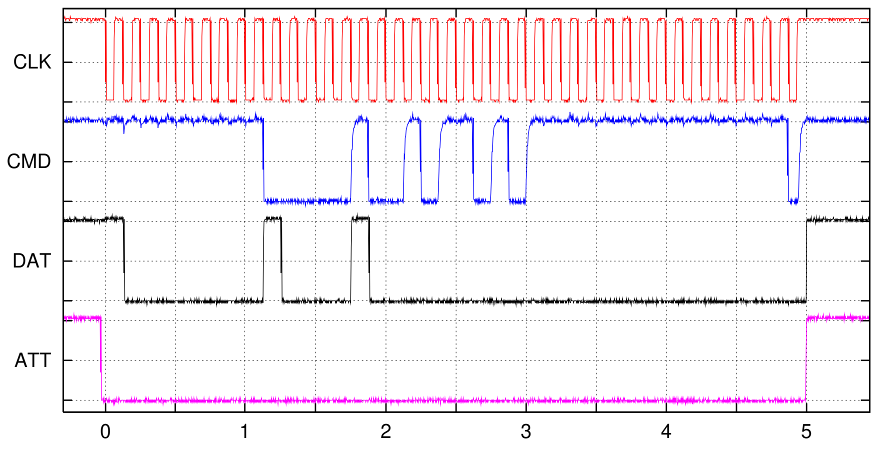
5.1 Gamepad, aflæsninger per sekund throughput

Simpel counter test viste af interrupt rutinen blev kørt 146 gange pr læsning. målet er ca. 30 aflæsninger pr. sekunder, det opnået da rutinen faktisk læser gamepad'en 79 gange i sekundet.

Ved brug af timer0 er både timer1 og timer 2 fri til pwm.

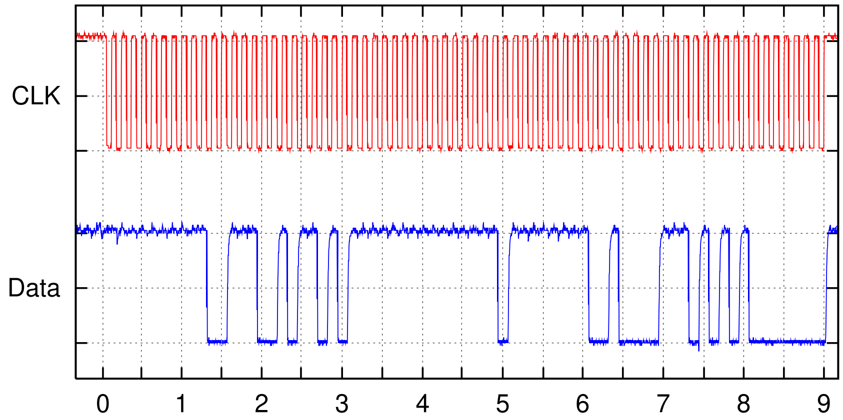
CLK og Data linje fra en gamepad i analog mode LSB sendes først, så bytene skal læses bagfra.

5.1.1 Implementationen følger protokollen



Figur 5.1: PS Controller protokol, digital mode

PS Controller protokollen, vist med en gamepad i digital mode. Gamepad'en modtager (i hex) 01 42 00 00 00 og sender samtidigt FF 41 5A FF 7F (MSB sendes som sidste bit).



Figur 5.2: PS Controller - Data-pin

CLK og Data linjer fra en gamepad i analog mode, Bytene er (i hex): FF 73 5A FF 7F FF 84 AB 00. (MSB sendes som det sidste bit i hver byte)

5.2 Radioprotokol

5.2.1 Output fra debug statements

Ofte bruger jeg en hex-værdi, fx 0x2B i stedet for det tilsvarende decimaltal, da det så er lettere regne ud hvilke bits der er høje eller lave. Dette gør jeg fordi at vi har med et embedded system at gøre og vi derfor er på et lavere abstraktionsniveaui, dvs. tættere det fysiske lag, altså det der faktisk sker med fx. spændingsniveauet på data-linjen til PS Controlleren.

I debug printes i starten af linjen en 8 bits tæller, der inkrementeres for hver linje, således at det er tydeligt om der er tale om en ny linje eller om der slet ikke modtages noget fra microcontrolleren. Endvidere udskrives også CRC-værdien for data i pakken samt det venstre joysticks position mellem midte og helt fremme, dvs. et tal mellem 0x0 og 0xFF (0-255 decimalt).

Basal pakkeoverførsel, bitvis:

```
0101010101010101001100111100110001000001111111111111111111111111111111111111111111111111101
```

vil som debug-output blive udskrevet som:

```
144: CRC: 0x8B50. PWM:  0 01010101 01010101 00110011 11001100 01000001 \
11111111 11111111 11111111 11111111 11111111 11111111 01010000 10001011
```

En pakke med analog-data data kunne i debug-mode se således ud:

```
140: CRC: 0x73AF. PWM: 95 01010101 01010101 00110011 11001100 01110011 \
11111111 11111111 01101000 01111111 00000000 00110101 10101111 01110011
```

5.2.2 Sammenligning af debug output

```
--- Minicom / USB ---
37: CRC: 0xA473. PWM: ff 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
38: CRC: 0xC529. PWM: f9 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
39: CRC: 0xABAE. PWM: d5 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
40: CRC: 0x73AF. PWM: 95 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
41: CRC: 0x4BEF. PWM: 2b 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
42: CRC: 0xCE6E. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
43: CRC: 0xDCAE. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
44: CRC: 0x9E79. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
45: CRC: 0x31CE. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
46: CRC: 0x61CD. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...

-- Minicom / COM ---
38: CRC: 0xC529. PWM: f9 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
39: CRC: 0xABAE. PWM: d5 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
40: CRC: 0x73AF. PWM: 95 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
41: CRC: 0x4BEF. PWM: 2b 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
42: CRC: 0xCE6E. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
43: CRC: 0xDCAE. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
44: CRC: 0x9E79. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
45: CRC: 0x31CE. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
46: CRC: 0x61CD. PWM:  0 01010101 01010101 00110011 11001100 01110011 11111111 11111111 ...
47:
```

Basen udskriver først en pakke når CRC-beregningen er OK, og derfor er den sidste "pæn" i forhold til terminals output, der hele tiden udskriver den afsendte pakke, som jeg har afbrudt midt i udskrivningen.

5.2.3 Througput

Ved timer måling måltes, at basen kan opfange 41 pakker pr. sekund, ved 7.3MHz.

5.2.4 CRC-fejl

CRC-fejl: Hvis debug-output er slået til i `common.h`, printes følgende besked:

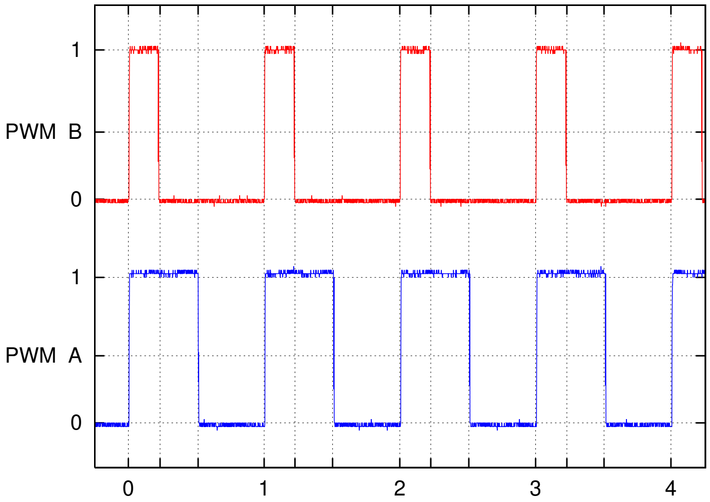
```
62:
CRC_ERROR!! CRC: Package said 0xEEFA, calculation was 0x6FCA
```

Det sker oftest når terminalen er slukket, men basen finder både et preamble og et SOF-match i radiostøjenen. Efter et SOF-match, aflæses der 9 bytes, i dette tilfælde ren støj. En CRC på disse data vil meget sjældent være korrekt, og det er da heller ikke tilfældet her.

5.3 PWM-signal

PWM : Den procentdel af tiden hvor et firkant signal er logisk eet kaldes signalets duty cycle.

PWM, dvs. Pulse Width Modulation, er et firkantsignal hvor man modulerer bredden af pulsen, dvs. længden af den del af signalet der repræsenter logisk 1. Duty cyclen er den procentdel af hver periode, hvor et PWM-signal er trukker højt (for et inverteret PWM-signal er det omvendt.)



Figur 5.3: Eksempel på PWM genereringen.

PWM generering virker tilfredsstillende.

5.4 Strømforbrug

I analog mode: 7.82 mA. Målinger viste at hver knap trækker ca. 8.06 - 7.82 = 0.24 mA. En hurtig test med noget tape viste at 10 aktiveret knapper trak ialt 10.11 mA. Det passer meget godt med 10 * 2.4 mA + 7.82 mA = 10.02 mA, altså kun 0.1 mA mindre end det faktiske.

I digital mode trækker gamepad'en 3.93 mA; 6.34 m. 10 knapper ≈ 2.4 mA + 3.93 = 6.33, altså meget pænt.
Basen blev målt til 68mA og terminalen til 30mA.

5.5 USB - fjernbetjening af XMMS

I bilag B.1 side 51 ses et perl script, som kan fortolke følgende bitstreng og herved sende de tilhørende commandoer til xmms.

Eksempel på data overførsel fra basen til en PC

```
0111001111111111111111111011010100111111110000000110001111\^M
```

Kapitel 6

Perspektivering

I dette projekt er der implementeret en god platform, som kan forbedres og udvides. flere muligheder for forbedringer og udvidelser; her er først et par måder det eksisterende system kan forbedres på:

- Batterier: terminalen er ikke trådløs, men med et par opladelige batterier og noget elektronik til opladningen, kunne den blive helt trådløs og da en ATmega8(L) i SMD-udgave er meget lille, burde terminalen kunne bygges ind i selve gamepad'en. Hvis der ikke findes et display andre steder i systemet, ville det også være en god ide at putte nogle lysdioder i gamepad'en så brugeren ved hvornår batterierne skal lades. Spændingen på batterierne kan fx. aflæses vha. den indbyggede A/D-konverter - er det rigtigt? TODO.

- MCU: power-down og -up af microcontrolleren og radiochippen, så der kan spares strøm, hvilket er ønskeligt hvis basen og/eller terminalen kører på batterier.

- Radio: to-vejs kommunikation mellem terminalen og basen ville være smart, så basen fx. kan bede terminalen om at skrue op for signalstyrken. Samtidig kan protokollen udvides så andre typer data også kan sendes mellem terminalen og basen, fx. ved hjælp af et ID-nummer i starten af pakken.

- aflæs signal styrke på RSSI og derved filtrere noget støj.

- Yderligere søgning efter dokumentation på den protokollen til gamepad'en, fx. så de små joysticks kan aktiveres af microcontrolleren.

- Hvis basen skal tilsluttes en PC, ville det være en god ide at interface til operativsystemet, fx. som en udvidelse til joystick-driveren i Linux-kernen. Så kan gamepad'en bruges af alle de programmer, der i forvejen supporterer joysticks, i stedet for, at jeg som nu, skal tilpasse et mellemlag til hvert enkelt program. Den nuværende måde er ogsa unødvendig resource krævende og det vil ikke være let få fx. SDL[1]-baserede programmer.

- Programmering via USB: Den valgte USB-adapter (FT232BM) understøtter 8 bit parrallel overførsel med D2XX-driveren. Hvis det kunne bruges til at programmere AVR'en med, kan man undvære AVR-ISP-forbindelsen til parrallelporten. Det dog lidt dyrere USB-bussen alligevel bruges til kommunikation med en PC, ville AVR'en let kunne omprogrammeresm, også efter at den var blevet pænt monteret i en kasse.

[1] Et open-source bibliotek til 3D grafik med OpenGL-understøttelse, `www.libsdl.org`.

Udvidelser

Fx. en nærhedsfølsom power-up, så at MCU'en starter lige inden man ville bruge gamepad'en uden at man skal tænde for det først.

- Display: Et LCD-display kunne også være en god ide og det burde være muligt både for kilenten og basen. På terminal burde man dog overveje om en ATmega128 ville være et bedre valg pga. af de mange flere ben.

- Accelerometre, så også måden joystick bevæges på kan bruges som input.

Fremtidigt udvikling

Projektet har været både sjovt og lærerigt og derfor vil jeg i min fritid arbejde videre med projektet og forsøge at implementere nogle af de nævnte forbedringer og udvidelser til projektet. Først og fremmest så jeg gerne at terminalen blev bygget ind i gamepad'en sammen med nogle batterier. Dette vil gøre den brugbar for mange flere, da basen er noget enklere at bygge.

Kapitel 7

Konklusion

Projektet er en succes. Det vigtigste, nemlig at

joysticket kan aflæses og data kan transmitteres over radioforbindelsen med fejl detetektering virker, så de basale krav til projektet er opfyldt. Der er vist en simpel og effektiv måde at tilslutte til en MCU til en USB-bus og dermed at overføre data til en PC.

PWM/motorstyring også OK, 128 spændingsniveauer.

Der ligger ikke noget umiddelbart i vejen for at tilslutte basen til et andet microcontroller-baseret system som dermed kan gøre bruge af joysticket, som fx. CLM's tagvaskerrobot.

Projektet er modulært - både joystick- og radio-delen af projektet kan bruges hver for sig, og da USB-forbindelsen kan agere en virtuel com-port, behøver MCU'en slet ikke at vide noget om USB-bussen. Desuden understøttes der ogaså meget hurtigere baud rate end en almindelig serielport.

Det er vist med et simpelt perl-script at joysticket kan fjernbetjene XMMS-programmet, og dermed at en PC kan fjernstyres.

Selve protokollen til joysticket er ikke afdækket, men der er jo også tale om en lukket protokol.

Alt i alt har det været et meget lærerigt projekt, med et godt tværfagligt udbytte.

Litteratur

Andreassen, T. (2005). *Digital Transmission Through Radio*. Ingeniørhøjskolen Odense Teknikum, bacheloropgave.

Barr, M. (2002). *Slow and Steady Never Lost the Race*. Application note with an implementation of CRC-functions in C with both CRC16 and CRC32, `http://www.embedded.com/2000/0001/0001connect.htm`.

Chipcon (2003). *SmartRF CC1000PP Plug and Play Module User's Manual*. Application note, rev. 1.22, `http://www.chipcon.com`.

DLP Design (2002). *DLP-USB232M User's Manual*. Application note, `http://www.dlpdesign.com`.

FTDI Ltd. (2005). *FT232BM USB USART I.C.*. Datablad, `http://ftdichip.com`.

Maxim (2005). *+5V-Powered, Multichannel RS-232 Drivers/Receivers*. Datablad, 19-4323, Rev 9, 5/00, `http://www.maxim-ic.com/`.

McCubbin, A.J. (1998). *Sony Playstation Controller Information*. `http://www.gamesx.com/controldata/psxcont/psxcont.htm` (Forfatterens website, `http://home.quicknet.com.au/andrewm/electron/psxcont.html` er taget ned og derfor henvises der til et mirror, der har en introduktion på `http://www.gamesx.com/stickadapt/`).

Palsson, G. (2005). *USB as interface between JAVA applications and embedded microcontrollers*. Ingeniørhøjskolen Odense Teknikum, speciale.

Torvmark, K.H. (2002a). *AN009: CC1000/CC1050 Microcontroller interfacing*. Chipcon, rev. 3.0, `http://www.chipcon.com`.

Torvmark, K.H. (2002b). *AN015: RF Modem Reference Design*. Chipcon, application note, rev. 1.22, `http://www.chipcon.com`.

Symbolliste

bit-banging Se dokumentation `/doc/ft232bm-bit-bang-mode.pdf`, side 7

gamepad se PS controller, side 1

ISP In-system Serial Programmer, side 7

MCU Micro Controller Unit, side 7

PS controller Sony®Playstation Controller, side 1

PWM Pulse Width Modulation, side 23

USART Universal Syncronous and Asyncronous serial Reciever and Transmitter, side 7

USB Universal Serial Bus, side I

Bilag A

Programkode

Som beskrevet i afsnit 1.3, er der kun inkluderet den kildekode, der er relevant for projektets problemstilling, dvs. kode, der bruges til joystick- eller radiokommunikationen.

A.1 base.c

```
/* Base unit for my wireless PlayStation Contrller Interface. Target platform
 * is the ATmega128, one of Atmel's AVR microcontrollers.
 *
 * Receives a radio package with joystick data from the terminal.
 * See radio/protocol.[hc].
 *
 * $Id: base.c 193 2005-10-01 00:39:43Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include <avr/interrupt.h>
#include <radio/protocol.h>
#include <radio/radio.h>
#include "common.h"
#include "timer.h"
```

```c
#include "psx.h"

#include "uart.h"

void send_psxdata_in_ascii(volatile psx_data *psxp);

int main(void){
  volatile psx_data *psx = &joydata;
  /*uint8_t *psx_arr = (uint8_t *) psx;
   * Access the psx data via:
   * joydata.Member
   * psx->Member
   * psx_arr[n] */

#ifdef _DEBUG
  uint8_t n = 0;
#endif

  UART_init();
  radio_init(); /* Setup the radio link for receiving */
  start_pwm_timer(); /* Setup the PWM timer */
  package_init(); /* Initialise the package structure for
                     the radio transmission */
  sei();
  while(1){
#ifdef _DEBUG
    printf("%3d: ", n++);
#endif

    do {
      receive_package(); /* Stored in package[] (global var) */
    } while(! check_package_crc(psx)); /* calculate new crc and compare */

    OCR2 = psx_pwm_calc(psx->l_vert); /* Calculate a PWM example */

#ifdef _DEBUG
    printf("PWM: %2x ", OCR2);
    print_package(package);
#else
    send_psxdata_in_ascii(psx);
#endif

  }
  return 0;
}

/* This function sends (prints) the joystick via the USART to a PC
 * that reads the data with the perl script 'psxmms.pl'.
 * On Unix-like system, stty(8) can be used to set the right baud rate for
 * the serial port device.
 */
void send_psxdata_in_ascii(volatile psx_data *psxp){
  int n;
  uint8_t *psx_arr = (uint8_t *) psxp;
```

```
    /* Send data over the UART.
     */
    for(n = 0; n < PSX_DATA_LEN; n++) {
        printbyte_n(psx_arr[n]);
    }
    putchar('\n');
}
```

A.2 terminal.c

```
/* PlayStation Controller Interface Implementation for the ATmega8 with radio
 * communication support for the ATmega8, one of Atmel's AVR
 * microcontrollers. It should be easily portable, especially to other
 * ATmegas, like the '128.
 *
 * Sends the data that has been read from the PS joystick to the "base" unit.
 *
 * $Id: terminal.c 193 2005-10-01 00:39:43Z anders $
 *
 * Copyright (C) 2005 by Anders Dubgaard <anders@dubgaard.net>
 * and Tórur Andreassen <ta@imada.sdu.dk>
 *
 * Credits and references:
 *
 * Andrew J. McCubbin (andrewm@quicknet.com.au),
 * http://www.gamesx.com/controldata/psxcont/psxcont.htm
 * - which is a mirror of the original site,
 * http://home.quicknet.com.au/andrewm/electron/psxcont.html
 *
 * Michael Barr for the CRC implementation,
 * http://www.embedded.com/2000/0001/0001connect.htm
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include <avr/interrupt.h>
#include <radio/protocol.h>
#include <radio/radio.h>
#include "common.h"
```

```c
#include "timer.h"
#include "psx.h"

#ifdef _DEBUG
  #include "uart.h"
#endif

int main(void){
  /* psx is a pointer to the joystick data struct */
  volatile psx_data *psx = &joydata;
  /*uint8_t *psx_arr = (uint8_t *) psx;
   * Access the psx data via:
   * joydata.Member
   * psx->Member
   * psx_arr[n] */

  psx_init(); /* Setup the connection to the PS Joystick*/
  radio_init(); /* Setup the radio connection */
  start_pwm_timer(); /* Setup the PWM timer */
  package_init(); /* Initialise the package structure for
                          the radio transmission */

#ifdef _DEBUG
  UART_init(); /* Enable printing to std. out via. the UART */
  uint8_t n = 0;
#endif

  sei();
  while(1){

#ifdef _DEBUG
    printf("%3d: ", n++);
#endif

    psx = get_psx_data(); /* Get data from the joystick */
    update_package(psx); /* Update the package, new data and CRC16 */
    send_package(); /* Pass the data to the airwaves */

    OCR2 = psx_pwm_calc(psx->l_vert); /* Calculate a PWM example */

#ifdef _DEBUG
    printf("PWM: %2x ", OCR2);
    print_package(package);
#endif

  }
  return 0;
}
```

A.3 psx.h

```c
/*
```

```c
 * $Id: psx.h 218 2005-10-04 00:55:09Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#ifndef _PSX_H
#define _PSX_H 1

#include <avr/signal.h>
#include "common.h"
#include "timer.h"

/* Pin definitions */
#define DATA PC0 /* data, In, uC <-- joy */
#define CMD PC1 /* command, Out, uC --> joy */
#define ATT PC2 /* attention, Out, uC --> joy */
#define CLK PC3 /* clock, Out, uC --> joy */

/* Macros to set pins high or low. */
#define clk_hi() (setbit(PORTC, CLK))
#define clk_lo() (clrbit(PORTC, CLK))
#define att_hi() (setbit(PORTC, ATT))
#define att_lo() (clrbit(PORTC, ATT))
#define cmd_hi() (setbit(PORTC, CMD))
#define cmd_lo() (clrbit(PORTC, CMD))

/* PSX Controller IDs
 * DIGIPAD and ANALOG_RED are the most common types around. */
#define DIGIPAD 0x41
#define ANALOG_RED 0x73
#define NEGCON 0x23
#define ANALOG_GREEN 0x53
#define PSX_MOUSE 0x12

/* The protocol consists of 5 to 9 bytes. The 1st and 3rd byte never changes;
 * Byte 1 = 0xff and byte 3 = 0x54 ( = "here comes the data")
 * - so they go to /dev/null ;-) */
typedef struct {
  uint8_t type; /* 2nd */
  uint8_t buttons; /* 4th */
```

```
  uint8_t buttons2; /* | */
  uint8_t r_horz; /* | */
  uint8_t r_vert; /* | */
  uint8_t l_horz; /* V */
  uint8_t l_vert; /* 9th */
} psx_data;

void psx_init(void);
void start_psx_timer(void);
int check_psx_type(int);
volatile psx_data *get_psx_data(void);
void disable_psx_int(void);
void enable_psx_int(void);
uint8_t psx_pwm_calc(uint8_t axis_pos);

volatile psx_data joydata;
volatile unsigned char *psx_arr;
volatile int psx_arr_len;

#endif // _PSX_H
```

A.4 psx.c

```
/*
 * $Id: psx.c 206 2005-10-02 18:29:26Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include "psx.h"

volatile uint8_t cmdbuf[] = {
  0x01, 0x42, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
volatile uint8_t databuf[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

volatile psx_data *psx = &joydata;
```

```c
volatile int8_t clk;
volatile int8_t bit_cnt = 0;
volatile int8_t byte_cnt = -1; /* -1 is the state prior to reading any data */
volatile uint8_t mask = 1; /* Send LSB first */
volatile uint8_t psx_data_ready = false;

uint8_t middle = 0x80 - 1; /* middle is 127 */
uint8_t nullzone = 5;

/* PS joystick interrupt rutine.
 * Run (here) by soft interrupts from timer0 at 32KHz.
 *
 * States:
 * -1 Gets the attention of the joystick controller
 * 0-8 Read 9 bytes of data, byte 0 is nonthing, byte 2 = 'here's the data'
 * 9 No more data,
 *
 */
SIGNAL(SIG_OVERFLOW0){
  if(byte_cnt < 0){ /* -1 or less... */
    att_lo();
    byte_cnt++;
  }
  else if (byte_cnt >= 0 && byte_cnt < 9){
    if(PINC & 1<<CLK) { /* Write bits on falling edge */
      clk_lo();
      if(cmdbuf[byte_cnt] & mask)
        cmd_hi();
      else
        cmd_lo();
      mask <<= 1;
    }
    else { /* READ bits on rising (leading) edge */
      clk_hi();
      if((PINC & 1<<DATA) == 1)
        setbit(databuf[byte_cnt], 7); /* set MSB in current byte */
      else
        clrbit(databuf[byte_cnt], 7); /* clear MSB in current byte */
      if(bit_cnt < 7)
        databuf[byte_cnt] >>= 1; /* as we receive LSB first */
      bit_cnt++; /* Prepare for next data and cmd bit */
      if(bit_cnt > 7){ /* if this is the 8th bits, prepare for next byte */
        bit_cnt = 0;
        mask = 1;
        byte_cnt++;
      }
    }
  } else { /* byte_cnt = 9 */
    att_hi();
    cmd_hi();
    byte_cnt = -1;
    if(check_psx_type(databuf[1])) { /* Check joystick type */
      /* copy data buffer to psx struct */
      psx->type = databuf[1]; /* 2nd data byte, psx_arr[0] */
```

```
      psx->buttons = databuf[3]; /* 4th data byte, psx_arr[1] */
      psx->buttons2 = databuf[4];/* 5th data byte, psx_arr[2] */
      psx->r_horz = databuf[5]; /* 6th data byte, psx_arr[3] */
      psx->r_vert = databuf[6]; /* 7th data byte, psx_arr[4] */
      psx->l_horz = databuf[7]; /* 8th data byte, psx_arr[5] */
      psx->l_vert = databuf[8]; /* 9th data byte, psx_arr[6] */

      disable_psx_int();
      psx_data_ready = true;
    }
  }
}

/* Initialise the PSX connections; pins and register
 */
void psx_init(void) {
  DDRC = (1<<CMD)|(1<<ATT)|(1<<CLK); // Pin directions, 1 = out, 0 = in

  att_hi(); /* ATT for PSJoy must start high */
  clk_hi(); /* CLK for PSJoy must start high */

  start_psx_timer();
}

/* Enables the joystick data gathering interrupt (which disables itself)
 * and waits for the data to be ready and returns a pointer to it.
 */
volatile psx_data *get_psx_data(void) {
  psx_data_ready = false;
  enable_psx_int();
  while(!psx_data_ready)
    ;
  return psx;
}

void disable_psx_int(void) {
  clrbit(TIMSK, TOIE0);
}

void enable_psx_int(void) {
  setbit(TIMSK, TOIE0);
}

/* Returns 1 if the PS joystick's is OK, otherwise returns 0.
 */
int check_psx_type(int type){
  if(type == DIGIPAD
      || type == ANALOG_RED) {
    return 1;
  }
  else {
    return 0;
  }
}
```

```

/* Wrapper for the PS joystick interface rutine timer start
 */
void start_psx_timer(void){
  start_timer0();
}

/* Simple calculation of an analog joystick's position from
 * the midpoint, 0x80, (when not used) to an end point, 0xFF.
 */
uint8_t psx_pwm_calc(uint8_t axis_pos) {
  uint8_t forward_val = 0;
  /* PWM, from midpoint to end of an axis = 0-0x80
   * - this must be mapped into the output range, 0-0xff
   */
  forward_val = axis_pos;
  if(forward_val > (middle - nullzone))
    forward_val = 0;
  else {
    forward_val = middle - forward_val;
    forward_val = forward_val * 2 + 1;
  }
  return forward_val;
}
```

A.5 radio/protocol.h

```
/*
 * $Id: protocol.h 198 2005-10-01 14:20:12Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#ifndef _PROTOCOL_H
#define _PROTOCOL_H

#include "psx.h"
#include "crc.h"
```

```c

/* The code assumes a 16bit SOF. It was difficult to make it neat
 * without being bothered by the LowByte-first-then-HighByte endian-ness... */
#define PSX_DATA_LEN (sizeof(psx_data)) /* 7B */
#define PREAMBLE 0x5555
#define PREAMBLE_LEN 2
#define SOF 0x33CC
#define SOF_LEN 2
/* HEADER_LEN = 13 if 2B preamble and 2B SOF */
#define HEADER_LEN (PREAMBLE_LEN + SOF_LEN)
#define PACKAGE_LEN (HEADER_LEN + PSX_DATA_LEN + CRC_LEN)
#define PAYLOAD_LEN (PACKAGE_LEN - HEADER_LEN)

volatile uint8_t package[PACKAGE_LEN]; /* package[x] == *(package + x) */
volatile crc pkg_crc;

void package_init(void); /* For use when transmitting */
volatile uint8_t *update_package(volatile psx_data *);
int check_package_crc(volatile psx_data *);

#endif // _PROTOCOL_H
```

A.6 radio/protocol.c

```c
/*
 * $Id: protocol.c 198 2005-10-01 14:20:12Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include "protocol.h"
#include <uart.h>
#include <psx.h>

/* Updates the package from the supplid data struct with new joystick
 * data and calculates the CRC checksum - returns a pointer to the package.
 */
volatile uint8_t *update_package(volatile psx_data *psx_pkg) {
```

```
  int n;
  uint8_t *psx_arr = (uint8_t *) psx_pkg;

  /* Create the package (preamble and SOF is done in package_init()
   *
   * Add data to the package
   */
  for(n = 0; n < PSX_DATA_LEN; n++) {
    package[n + HEADER_LEN] = psx_arr[n];
  }

  /* Calculate CRC16 sum
   * - thanks to Michael Barr in a three part series at
   * http://www.embedded.com/2000/0001/0001connect.htm
   */
  crc crc_val = crcSlow(psx_arr, PSX_DATA_LEN);
  uint8_t *crcp = (uint8_t *) &crc_val;

#ifdef _DEBUG
  printf("CRC: 0x%4X. ", crc_val);
#endif

  /* Add CRC16 sum to the package - low byte lowest.
   */
  for(n = 0; n < CRC_LEN; n++) {
    package[n + HEADER_LEN + PSX_DATA_LEN] = crcp[n];
  }
  return package;
}

/* Store joystick from the radio package in the psx_data array.
 */
void pkg2psxdata(volatile psx_data *psxp){
  uint8_t n;
  uint8_t *psx_arr = (uint8_t *) psxp;

  for(n = 0; n < PSX_DATA_LEN; n++) {
    psx_arr[n] = package[n + HEADER_LEN];
  }
}

/* Updates the supplid data struct with new joystick data and
 * checks the CRC checksum, and returns the result of the CRC calculation.
 */
int check_package_crc(volatile psx_data *psxp){
  crc crc_pkg = 0; /* received CRC value */
  uint8_t *crcp = (uint8_t *) &crc_pkg;
  uint8_t *psx_arr = (uint8_t *) psxp;

  /* Read CRC from package */
  crcp[0] = package[11];
  crcp[1] = package[12];

  /* Read joystick data from package */
```

```
  pkg2psxdata(psxp);

  /* Calculate CRC or the package */
  crc crc_val = crcSlow(psx_arr, PSX_DATA_LEN);

#ifdef _DEBUG
  if(crc_val == crc_pkg) {
    printf("CRC: 0x%4X. ", crc_val);
  }
  else {
    printf("\n\r CRC_ERROR!! ");
    printf("CRC: Package said 0x%X, calculation was 0x%X\n\r",
        crc_pkg, crc_val);
  }
#endif

  return (crc_val == crc_pkg)? 1: 0;
}

/* Insert preamble and Start-of-Frame into the package.
 * It's a quick hack because of the endian-ness.
 */
void package_init(void) {
  package[0] = (uint8_t) (PREAMBLE>>8);
  package[1] = (uint8_t) PREAMBLE;
  package[2] = (uint8_t) (SOF>>8);
  package[3] = (uint8_t) SOF;
  package[4] = 0; /* psx->type */
  package[5] = 0; /* psx->buttons */
  package[6] = 0; /* psx->buttons2 */
  package[7] = 0; /* psx->r_horz */
  package[8] = 0; /* psx->r_vert */
  package[9] = 0; /* psx->l_horz */
  package[10] = 0; /* psx->l_vert */
  package[11] = 0; /* CRC high byte */
  package[12] = 0; /* CRC low byte */
}
```

A.7 radio/radio.h

```
/*
 * $Id: radio.h 176 2005-09-27 17:19:21Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#ifndef _RADIO_H
#define _RADIO_H 1

#include <avr/io.h>
#include <common.h>

void radio_init(void);
void send_package(void);
void receive_package(void);
void disable_radio_int(void);
void enable_radio_int(void);

#endif // _RADIO_H
```

A.8 radio/radio_tx.c

```
/*
 * $Id: radio_tx.c 173 2005-09-27 17:09:59Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include <avr/signal.h>
#include <stdio.h>
#include "radio.h"
#include "protocol.h"
#include "cc1000.h"
#include "cc1000registers.h"
#include <uart.h>

```

```
/* Client / transmitter */
volatile uint8_t package_sent = false;
volatile uint16_t tx_byte_cnt = 0; /* MSB is to be sent first, so shift */
volatile uint8_t tx_mask = 0x80; /* tx_mask to the right. */


/* Sends the joystick data on it's way in a straight forward manner,
 * as the data array has been prepared beforehand.
 *
 */
SIGNAL(SIG_INTERRUPT0) { /* Transmit */
  if(tx_byte_cnt >= 0 && tx_byte_cnt < PACKAGE_LEN) {
    if(package[tx_byte_cnt] & tx_mask) {
      setbit(CC1000_PORT, DIO);
    } else {
      clrbit(CC1000_PORT, DIO);
    }
    tx_mask >>= 1; /* Prepare mask for next bit */
    if(tx_mask < 1) { /* mask=1 => sent one byte */
      tx_mask = 0x80;
      tx_byte_cnt++;
    }
  }
  else { /* Package bytes sent */
    disable_radio_int(); /* Wait to be enabled */
    package_sent = true; /* Tell main code to read new data from joystick */
    tx_byte_cnt = 0;
  }
}

/* Initialise the radio chip.
 * The CC1000 register variables in ConfigureCC1000() have been
 * generated with smartrf2header.pl from the output data of
 * Chipcon's SmartRF(C) programme.
 *
 * The step wise procedure below is described in Chipcon's AN009.
 */
void radio_init(void) {
  CC1000_PDIR |= 1<<PCLK | 1<<PALE | 1<<PDATA;
  ResetCC1000();

  ConfigureCC1000(tx_numregs, tx_regs);
  clrbit(MCUCR, ISC00); // interrupt on falling edge
  setbit(MCUCR, ISC01); // interrupt on falling edge

  CalibrateCC1000();

  setbit(CC1000_PDIR, DIO);
  clrbit(CC1000_PDIR, DCLK);
}

/* Starts the transmitting int. routine.
 *
 */
```

```
void send_package(void) {
  package_sent = false;
  enable_radio_int();
  while(!package_sent)
    ;
}

void disable_radio_int(void) {
  clrbit(GICR, INT0); /* disable interrupt on INT0 */
}

void enable_radio_int(void) {
  setbit(GICR, INT0); /* enable interrupt on INT0 */
}
```

A.9 radio/radio_rx.c

```
/*
 * $Id: radio_rx.c 208 2005-10-02 22:38:34Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include <avr/signal.h>
#include <stdio.h>
#include "radio.h"
#include "protocol.h"
#include "cc1000.h"
#include "cc1000registers.h"
#include <uart.h>

/* Base / receiver */
enum rx_states { FIND_PRE, FIND_SOF, GET_PAYLOAD, PKG_REC };
volatile uint8_t rx_state = FIND_PRE; /* Start with finding the preamble */
volatile uint16_t bitbin = 0; /* 16bit to easier catch Preamble and SOF */
volatile uint8_t rx_byte_cnt = 0;
volatile uint8_t rx_bit_cnt = 0;
volatile uint8_t package_received = false;
```

```c
volatile uint8_t MAX_SPACER_BITS = 24; /* atleast 3 * 8bits */
volatile uint8_t spacer_bits = 0;

/* Receiver-interrupt, with states.
 * Uses a 16 bit value to store the incoming bits which makes it easier
 * to find the preamble and the SOF. Used with a cast to collect
 * single bytes of data.
 * If a 1 byte preamble is detected and no SOF has been found after another
 * 16 bits, the routine starts over.
 */
SIGNAL(SIG_INTERRUPT2) { /* Receive */
  bitbin <<= 1; /* make room for next bit, MSB first */
  bitbin += (CC1000_PINS & 1<<DIO)? 1: 0; /* set bit */

  if (rx_state == FIND_PRE) { /* Search for Preamble */
    if ((uint8_t) bitbin == 0x55 ) { /* preamble is 0x5555, 0x55 is a substr */
      rx_state = FIND_SOF;
      spacer_bits = 0;
    }
  }
  else if (rx_state == FIND_SOF) { /* Search for SOF */
    if (bitbin == SOF ) {
      rx_state = GET_PAYLOAD; /* the following bytes are the data */
    }
    else {
      spacer_bits += 1;
      if(spacer_bits > MAX_SPACER_BITS) {
        rx_state = FIND_PRE;
      }
    }
  }
  else if (rx_state == GET_PAYLOAD) { /* Collect the data */
    if (rx_byte_cnt < PAYLOAD_LEN) { /* - but only a certain amount */
      rx_bit_cnt++;
      if (rx_bit_cnt > 7) { /* Now we have a byte w/ data */
        package[rx_byte_cnt + HEADER_LEN] = (uint8_t) bitbin;
        rx_byte_cnt++;
        rx_bit_cnt = 0;
      }
    }
    else { /* Start over, reset counters */
      rx_state = FIND_PRE;
      bitbin = 0;
      rx_byte_cnt = 0;
      rx_bit_cnt = 0;
      disable_radio_int();
      package_received = true; /* Tell others about the new data */
    }
  }
}

/* Initialise the radio chip.
 * The CC1000 register variables in ConfigureCC1000() have been
 * generated with smartrf2header.pl from the output data of
```

```
 * Chipcon's SmartRF(C) programme.
 *
 * The step wise procedure below is described in Chipcon's AN009.
 */
void radio_init(void) {
  CC1000_PDIR |= 1<<PCLK | 1<<PALE | 1<<PDATA;
  ResetCC1000();

  ConfigureCC1000(rx_numregs, rx_regs);
  setbit(EICRA, ISC20); // interrupt on rising edge
  setbit(EICRA, ISC21); // interrupt on rising edge

  CalibrateCC1000();

  clrbit(CC1000_PDIR, DIO);
  clrbit(CC1000_PDIR, DCLK);
}

/* Starts the receiving int. routine.
 *
 */
void receive_package(void) {
  package_received = false;
  enable_radio_int();
  while(!package_received) {
    ;
  }
}

void disable_radio_int(void) {
  clrbit(EIMSK, INT2); /* disable interrupt on INT2 */
}

void enable_radio_int(void) {
  setbit(EIMSK, INT2); /* enable interrupt on INT2 */
}
```

A.10 radio/cc1000_pins.h

```
/*
 * $Id: cc1000_pins.h 198 2005-10-01 14:20:12Z anders $
 *
 * Copyright (C) 2005 Anders Dubgaard <anders@dubgaard.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```c
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#ifndef _PINSETUP_H
#define _PINSETUP_H

/***********************************************************
 * CC1000 PORT setup
 * ==========
 * use these defines to setup your MCU connection to the CC1000
 ***********************************************************/
#define CC1000_PINS PIND
#define CC1000_PORT PORTD
#define CC1000_PDIR DDRD

#if defined (__AVR_ATmega8__)
  #define PALE PD7
  #define PDATA PD6
  #define PCLK PD5
  #define DCLK PD2 /* INT0 */
  #define DIO PD4
#elif defined (__AVR_ATmega128__)
  #define PALE PD7
  #define PDATA PD6
  #define PCLK PD5
  #define DCLK PD2 /* INT2 */
  #define DIO PD4
#else
# error "You know, I don't know what I'm looking for..."
#endif

#endif /* _PINSETUP_H */
```

Bilag B

Scripts til automation

B.1 psxmms.pl

```perl
#!/usr/bin/perl
# psxmms by Anders Dubgaard <anders@dubgard.net>, 2005.
#
# Reads commands from my custom-built Wireless PlayStation Controller
# Seriel-via-USB interface (on /dev/ttyUSB0 at the moment).
#
# See http://www.dubgaard.net/projects for more information.
#
# Bugs / Missing features:
# - Needs a good rewrite, as it's still a little in-efficient even
# though it's functional.
#
# Based on ctrlxmms by Robin Smidsrød:
#
# ctrlxmms 1.1 (Mon Jun 19 20:27:38 CEST 2000)
# (C) Robin Smidsrød <robin@smidsrod.no> June 2000
#
# This program is licensed according to the GPL.

$version="psxmms.pl by Anders Dubgaard <anders\@dubgaard.net";

use strict;
use warnings;

use Xmms::Remote (); # perl module from CPAN
use POSIX; # ceil and floor for rounding
my $jtime;

my $remote=Xmms::Remote->new;

my $joyinput = "/dev/ttyUSB0";
my $baudrate = 230400;

# set baud rate, assumes a Linux-like system
`stty -F $joyinput $baudrate`;
```

```
open PSJOY, "$joyinput" or die "$!";
#open MOUSE, "| xte";

my $line;
my $mask;

my %psx = (
  "type" => 0,
  "b1" => 0,
  "b2" => 0,
  "r_horz" => 0,
  "r_vert" => 0,
  "l_horz" => 0,
  "l_vert" => 0,
);

# button-press timeout vars
my $timeout = 4; # button press is ignored $timeout-times before reading again
my $timeout2x = $timeout * 2; # for use with the squischy rubber buttons

# X11 vars, using xte.
my $analog_id = 115; # Analog_red_mode = 115 = 0x73
my $x_rel;
my $y_rel;
my $xmove = 0;
my $div = 4;
my $mid;

# buttons1
my $slct = 0;
my $joy_r = 0;
my $joy_l = 0;
my $start = 0;
my $up = 0;
my $right = 0;
my $down = 0;
my $left = 0;

# buttons2
my $l2 = 0;
my $r2 = 0;
my $l1 = 0;
my $r1 = 0;
my $triangle = 0;
my $circle = 0;
my $cross = 0;
my $square = 0;

$ARGV[1] = ""; # $ARGV[1] is hardcoded to be the optional argument

$line = <PSJOY>;
$line = <PSJOY>;
while(1) {
```

```
  read_joystick();
  $ARGV[1] = "";

} #end main loop

sub read_joystick {
$line = <PSJOY>;
#print "B1: " . substr($line, 0, 8);
$psx{'type'} = substr($line, 0, 8);
$psx{'b1'} = substr($line, 8, 8);
$psx{'b2'} = substr($line, 16, 8);
$psx{'r_horz'} = substr($line, 24, 8);
$psx{'r_vert'} = substr($line, 32, 8);
$psx{'l_horz'} = substr($line, 40, 8);
$psx{'l_vert'} = substr($line, 48, 8);

$psx{'type'} = asc2int($psx{'type'});
$psx{'b1'} = asc2int($psx{'b1'});
$psx{'b2'} = asc2int($psx{'b2'});
$psx{'r_horz'} = asc2int($psx{'r_horz'});
$psx{'r_vert'} = asc2int($psx{'r_vert'});
$psx{'l_horz'} = asc2int($psx{'l_horz'});
$psx{'l_vert'} = asc2int($psx{'l_vert'});

#print sprintf("%lx ", $psx{'type'} );
#print sprintf("%lx ", $psx{'b1'} );
#print sprintf("%lx ", $psx{'b2'} );
#print sprintf("%lx ", $psx{'r_horz'});
#print sprintf("%lx ", $psx{'r_vert'});
#print sprintf("%lx ", $psx{'l_horz'});
#print sprintf("%lx ", $psx{'l_vert'});
#print "\n";

# Buttons, active low
# Bit0 Bit1 Bit2 Bit3 Bit4 Bit5 Bit6 Bit7
# b1: SLCT JOYR JOYL STRT UP RGHT DOWN LEFT
# b2: L2 R2 L1 R1 /\ O X |_|

$mask = 1;
if(0 == ($psx{'b1'} & $mask)) {
  if($slct == 0) { $slct = $timeout2x;
  } else { $slct -= 1; }
} # "slct";

$mask <<= 1;
if(0 == ($psx{'b1'} & $mask)) {
  if($joy_l == 0) { $joy_l = $timeout2x;
    `echo mouseclick 1 | xte`;
  } else { $joy_l -= 1; }
} # "joy_l";

$mask <<= 1;
if(0 == ($psx{'b1'} & $mask)) {
  if($joy_r == 0) { $joy_r = $timeout2x;
```

```
    `echo mouseclick 2 | xte`;
  } else { $joy_r -= 1; }
} # "joy_r";

$mask <<= 1;
if(0 == ($psx{'b1'} & $mask)) {
  if($start == 0) { $start = $timeout2x; &do_playpause;
  } else { $start -= 1; }
} # "start";

$mask <<= 1;
if(0 == ($psx{'b1'} & $mask)) {
  if($up == 0) { $up = $timeout; &do_prev;
  } else { $up -= 1; }
} # "up";

$mask <<= 1;
if(0 == ($psx{'b1'} & $mask)) { $ARGV[1]="10";
  if($right == 0) { $right = $timeout; &do_time;
  } else { $right -= 1; }
} # "right";

$mask <<= 1;
if(0 == ($psx{'b1'} & $mask)) {

  if($down == 0) { $down = $timeout; &do_next;
  } else { $down -= 1; }
} # "down";

$mask <<= 1;
if(0 == ($psx{'b1'} & $mask)) { $ARGV[1]="-10";
  if($left == 0) { $left = $timeout; &do_time;
  } else { $left -= 1; }
} # "left";

$mask = 1;
if(0 == ($psx{'b2'} & $mask)) { #$ARGV[1]="20"; # default = 10
  if($l2 == 0) { $l2 = $timeout2x;
    &do_prev; &do_prev; &do_prev; &do_prev; &do_prev;
    &do_prev; &do_prev; &do_prev; &do_prev; &do_prev;
  } else { $l2 -= 1; }
} # "l2"; # jump 10 songs back

$mask <<= 1;
if(0 == ($psx{'b2'} & $mask)) {
  if($r2 == 0) { $r2 = $timeout;
    system("aumix", "-w-5");
  } else { $r2 -= 1; }
} # "r2"; # vol up


$mask <<= 1;
if(0 == ($psx{'b2'} & $mask)) { # $ARGV[1]="20"; # default = 10
  if($l1 == 0) { $l1 = $timeout2x;
```

```
        &do_next; &do_next; &do_next; &do_next; &do_next;
        &do_next; &do_next; &do_next; &do_next; &do_next;
    } else { $l1 = $l1 - 1; }
} # "l1"; # jump 10 songs forward

$mask <<= 1;
if(0 == ($psx{'b2'} & $mask)) {
    if($r1 == 0) { $r1 = $timeout;
        system("aumix", "-w+5");
    } else { $r1 -= 1; }
} # "r1"; # vol up

$mask <<= 1;
if(0 == ($psx{'b2'} & $mask)) {
    if($triangle == 0) { $triangle = $timeout; &do_stop;
    } else { $triangle -= 1; }
} # "triangle";

$mask <<= 1;
if(0 == ($psx{'b2'} & $mask)) {
    if($circle == 0) { $circle = $timeout2x; &do_repeat;
    } else { $circle--; } #= $circle - 1;
} # "circle";

$mask <<= 1;
if(0 == ($psx{'b2'} & $mask)) {
    if($cross == 0) { $cross = $timeout; &do_play;
    } else { $cross -= 1; }
} # "cross";

$mask <<= 1;
if(0 == ($psx{'b2'} & $mask)) {
    if($square == 0) { $square = $timeout2x; &do_shuffle;
    } else { $square -= 1; }
} # "square";

# No buttons, then maybe mouse movement!
# sprintf("%lx", n) converts from hex
#$x_rel = sprintf("%lx", $psx{'l_horz'});
#$y_rel = sprintf("%lx", $psx{'l_vert'});
if($psx{'type'} == $analog_id){
    if($xmove == 0) {
        $xmove = 1; # skip the next mouse move(s)
        if($psx{'l_horz'} > 100 && $psx{'l_horz'} < 150){ $psx{'l_horz'} = 127; }
        if($psx{'l_vert'} > 100 && $psx{'l_vert'} < 150){ $psx{'l_vert'} = 127; }

        $psx{'l_horz'} = ceil($psx{'l_horz'} / 4 - 32);
        $psx{'l_vert'} = ceil($psx{'l_vert'} / 4 - 32);

        #print MOUSE "mousermove $psx{'l_horz'} $psx{'l_vert'}\n";
        `echo mousermove $psx{'l_horz'} $psx{'l_vert'} | xte`;
    } else { $xmove--; }
} else {
    $xmove = 0;
```

```
}

}

# asc2int: convert from str to int: "01110011" to 115 (0x73)
sub asc2int {
  my @parms = @_;
  my $str = $parms[0];
  my $i;
  $i = 0;
  my $sum = 0;
  my $mask = 1;

  for($i=0; $i<8; $i++) {
    if(substr($str, 7 - $i, 1)) { $sum += $mask; }
    $mask <<= 1;
  }
  #print "Total sum: " . sprintf("%lx", $sum) . "\n";
  return $sum;
}

sub do_playpause {
  my ($is_playing,$is_paused)=&songstatus;

  if ($is_playing eq 1) {
    if($is_paused eq 1) {
      &do_pause;
    } else {
      &do_pause;
    }
  } else {
    &do_play;
  }
}

sub do_play { $remote->play; }

sub do_pause { $remote->pause; }

sub do_stop { $remote->stop; }

sub do_next { $remote->playlist_next; }

sub do_prev { $remote->playlist_prev; }

sub do_forward {
  if (defined($ARGV[1])) {
    $jtime=$ARGV[1];
  } else {
    $jtime=5;
  }
  &jtime($jtime);
}
```

```
sub do_backward {
  if (defined($ARGV[1])) {
    $jtime="-" . $ARGV[1];
  } else {
    $jtime=-5;
  }
  &jtime($jtime);
}

sub jtime {
  my $jtime=shift @_;
  my $time=$remote->get_output_time;
  my $factor=1000; # Equals 1 second
  my $newtime=$time+($factor*$jtime);
  $remote->jump_to_time($newtime);

}
sub do_repeat { $remote->toggle_repeat; }

sub do_shuffle { $remote->toggle_shuffle; }

sub songstatus {
  my $playstatus=$remote->is_playing;
  my $pausestatus=$remote->is_paused;
  return ($playstatus,$pausestatus);
}

sub do_time { &jtime($ARGV[1]); }
```

B.2 tds1002.pl

```
#!/usr/bin/perl
# Dumps data from the Tektronix TDS-1002 digital oscilloscope over the RS-232
# interface and makes a nice (gnu-) plot from it.
#
# Copyright (C) 2005 Thor Andreassen <ta@imada.sdu.dk>
# Anders Dubgaard <anders@dubgaard.net>
#
# $Id: tds1002.pl 198 2005-10-01 14:20:12Z anders $
#
# Usage: tds1002.pl <filename>
# - dumps data from oscilloscope to filename and (if $do_gnuplot is set) runs
# gnuplot with output into filename.eps
#
# BUGS / Missing features
# - we currently assume that stty can set up the serial device properly;
# 19200,8N1, no flag.
#
# For details, see the TDS 200-, 1000 and 2000-Series Programmer Manual,
# which is NOT available from Tektronix's homepage, so search for it on
# http://www.google.com (the part number is 071-1075-01).
#
```

```perl
# More Tektronix-capturing information on the net:
# 1. http://secure.netroedge.com/~phil/tekscope/
# 2. http://www.warped.org/tek/
# 3. http://www.febo.com/geekworks/data-capture/
#
# GNUPlot:
# http://www.gnuplot.info/faq/faq.html
# http://www.duke.edu/~hpgavin/gnuplot.html
# http://aquaterm.sourceforge.net/index.shtml?page=s1 ;-)
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

use warnings;
use strict;
use File::Basename;

my $SERIAL_PORT = "/dev/ttyS0";
my $line;
my $do_gnuplot = "ya";

my $data_output;
if(defined $ARGV[0]){
  $data_output = $ARGV[0];
} else {
  $data_output = "/dev/stdout";
  $do_gnuplot = ""; # don't do gnuplotting as we send the coords to stdout
}

my $baudrate = 19200;
# set baud rate, assumes a Linux-like system
`stty -F $SERIAL_PORT $baudrate`;


open IN_DATA, "$SERIAL_PORT" or die "$!";
open OUT_DATA, ">$SERIAL_PORT" or die "$!";

print OUT_DATA "SELECT:CH2 ON\n";
print OUT_DATA "DATA:encdg ASCII\n";
print OUT_DATA "DATA:SOURCE CH1\n";
print OUT_DATA "CURVE?\n";
my $channel1 = <IN_DATA>;
```

```perl
print OUT_DATA "SELECT:CH2 ON\n";
print OUT_DATA "DATA:encdg ASCII\n";
print OUT_DATA "DATA:SOURCE CH2\n";
print OUT_DATA "CURVE?\n";
my $channel2 = <IN_DATA>;

close OUT_DATA;
close IN_DATA;

#$/ = $INPUT_RECORD_SEPARATOR = '\n\ as default
my @ch1 = split ',',$channel1;
my @ch2 = split ',',$channel2;

chop($ch1[@ch1-1]);
chop($ch2[@ch2-1]);

my $curve = "";
for (my $i=0; $i<2500; $i++) {
  $curve .= $i . "\t" . $ch1[$i] . "\t" . $ch2[$i] . "\n";
}
open DATAOUT, ">$data_output" or die "$!";
print DATAOUT $curve;
close DATAOUT;

if($do_gnuplot) {
  my $plotstr = 'set terminal postscript color solid; ' .
    'set output \"' . $data_output . '.eps\"; ' .
    'plot \"' . $data_output . '\" using 1:2 title \"CH1\" with lines lw 1, \"' .
    $data_output . '\" using 1:3 title \"CH2\" with l lt 4 lw 3';
  system("echo \"$plotstr\" | gnuplot");
}

# More TDS1002 commands:
# channel: CH1
# aq: PEAK
# ave: 16
# stat: 0
# ID? fx. ID TEK/TDS 1002,CF:91.1CT TDS2MEM:FV:v6.08
# DATa INIT
# DATa:ENCdg ASCIi
# Data:start 1
# Data:stop 2500
# data:source CH1
# acq:mode PEAK
# acq:numavg 16
# CH1:pos? fx. -1.32E0
# CH1:scale? fx. 1.0E0
# hor:main:scale? fx. 5.0E-5
# hor:main:pos? fx. 0.0E0
# WFMPre:WFId? fx. "Ch1, DC coupling, 1.0E0 V/div,
# 5.0E-5 s/div, 2500 points, Pk Detect mode"
# Data? fx.ASCII;REFA;CH1;1;2500;1
#
```

```
# GnuPlot example:
# plot [xmin:xmax] [ymin:ymax] ”file.dat” using 1:2 title ’CH1’ with lines,\
# ”file.dat” using 1:3 title ’CH2’ with lines 4
#
```