# Øvelse: Linux Device Driver with SPI

## Formål

Formålet med denne øvelse er at få erfaring med at anvende bus interfaces i en device driver. Herunder at opnå forståelse for anvendelsen af Hot- og Cold-plugging og hvordan dette implementeres i en device driver. Øvelsen tager udgangspunkt i et SPI bus interface og man vil derfor også opnå erfaring med anvendelsen af SPI som kommunikationsform.

## Forberedelse

- Du skal have et funktionelt VM-Ware Image og et Devkit8000.
- Du skal være i stand til at kunne kompilere et kerne modul.
- Skim **Wikipedea's afsnit om SPI**
- Skim SPI dokumentationen i kernel tree ( **/home/stud/sources/linux-3.2.6/Documentation/spi/spi-summary** )
- Skim spi.h igennem ( **/home/stud/sources/linux-3.2.6/include/linux/spi/spi.h**)
- Skim **Add-On Board Schematics**
- Skim **ADS7870 Datablad**
- Skim **DAC7612 Datablad**?
- Skim **PSOC Interface**
- Checkout koden fra **https://svn.nfit.au.dk/svn/iha_phm-phm/exercises/mps/ads7870** eller download filerne vha din browser til en ny folder
- Analogt Potentiometer (10K-1M ohm) og tre prøveledninger (ex dem fra STK500)

## Øvelses Steps

Med udgangspunkt i en eksisterende driver til A/D konverteren ADS7870 skal der implementeres en ny driver til et SPI device. Dette device kan være D/A konverteren DAC7612 som er monteret på **Add-on Printet** . Alternativt kan f.eks. PSOC3/PSOC4 tilkobles eksternt og disse programmeres med de respektive SPI device demo projekter som kan findes. Du kan finde mere information **her**.

## Indsættelse af ADS7870 Driver i Linux

a) Byg koden vha den tilhørende Makefile. Bemærk at resultatet er to kernemoduler: ''hotplug_ads7870_spi_device.ko'' og ''ads7870mod.ko''. Driveren (ads7870mod.ko) er bygget udfra to filer: ads7870.c (Overordnet character driver) og ads7870-spi.c (SPI specifikke ting). Hvad er formålet det andet hotplug kernemodulet? -Check i source koden.

b) Kopier begge kernemoduler til Devkit8000

c) Du skal nu registrere driveren i Linux. Dette gør du som sædvanligt ved indsætte driver (ads7870mod.ko) i kernen. Bemærk under ''spi_init'' i ads7870-spi.c at driveren bliver registrer sig i SPI frameworket. Noter hvilke beskeder som er tilføjet kerneloggen vha dmesg.

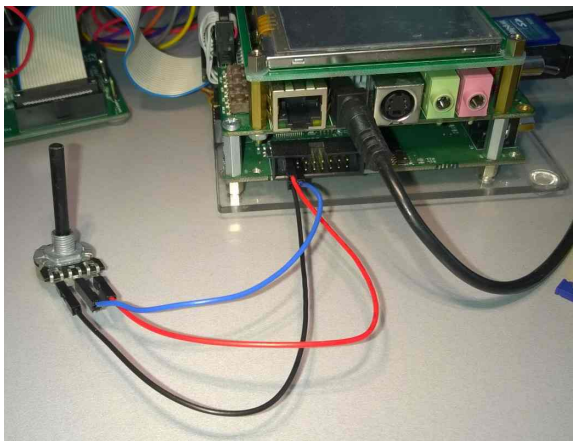
d) Nu skal vi have registreret et SPI device. Indsæt derfor *hotplug_ads7870_spi_device.ko* i kernen og noter igen beskederne i dmesg.
Hvad er det som er sket? Hvilke metoder er blevet kaldt i device driveren (ads7870-spi.c) efter at vi har tilføjet et device?

e) Opret noder til udlæsning af ADC inputs. ADS7870 har 8 kanaler og der oprettes derfor 8 devices (/dev/adc0 ... /dev/adc7). Dette kan med fordel gøres vha scriptet mk_ads7870_nodes.sh (Som selvfølgelig også skal kopieres til Devkit8000).

### Test af ADS7870

A/D konverterens inputs er forbundet til Add-On boardets stik J4. Se detaljer om stikkene **her** . ADS7870 er en 12-bit SAR A/D konverter som har en input spændingsområde på [0..2047mV] og et 2'komplement output. Dvs udlæser ADS7870 115 så svarer det til 115mV på indgangen.

f) Lån et analogt **potentiometer** på værkstedet (Læg det på plads efter brug!), de kan findes i rummet for enden af gangen ved værkstedet. Det skal være i størrelsesordnen 10K-1M ohm. Find tre testledninger (f.eks fra dit gamle STK500 kit) og monter dem som vist her:



Spændingen kan nu justeres fra 0-5 volt. Indgangen kan kun måle op til 2.047 volt, men beskadiges ikke op til de 5 volt.

g) Med kernemoduler indsat, noder oprettet og potentiometret forbundet er det nu tid til at teste. Anvend ''cat'' til at udlæse data fra den kanal som du har forbundet potentiometret til og drej potentiometret for at udlæse forskellige værdier:

```
cat /dev/adc0
```

Prøv at fjerne *hotplug_ads7870_spi_device.ko* modulet og herefter at læse fra /dev/adc0 igen. Vi får nu en fejl, men hvorfor? Hvor i koden er det at vi bliver stoppet i at tilgå et device som ikke eksisterer?

### Konstruktion af egen SPI device driver

Nu hvor du har prøvet at anvende den eksisterende driver, er det på tide at skrive din egen!

### Konstruktion af SPI hot-plug kernemodul

h) Udfra diagrammer og datablade skal du udlede følgende information:

- SPI bus nummer

- SPI chip-select (ss) nummer
- SPI max frekvens
- SPI Clock Mode
- Antal bits per transmission

Værdier og hvordan du har fundet dem, skal selvfølgelig med på wiki besvarelsen.

Se evt denne vejledning: **SPI_LDD_Design_Process.pdf**

i) Kopier *hotplug_ads7870_spi_device.c* til et nyt og signende navn. Opdatér herefter filen med
signende navne og de parametre som er fundet i (h). Hvorfor er det at vi har et "hotplug" modul? Hvad er formålet?

### Konstruktion af char driver Init/Exit

j) Opret en ny driver c-fil og implementer Init præcis som du har gjort det tidligere for en GPIO driver, med undtagelse af brugen af GPIO_request, *direction, _value etc. Hvor du I GPIO driveren requestede en GPIO, requester vi har en SPI ressource. Se funktionen _spi_init()* i ads7870-spi.c hvordan dette gøres. Husk! at du skal have udfyldt en *spi_driver* struct med et *name* magen til det du angav i din hotplug... fil.

k) Opret en *probe()* metode i stil med den du finder i ads7870-spi.c. I probe metoden skal du angive antal *bits_per_word* som fundet i (h) og kalde *spi_setup* for at tilskive parametrene. Lav herefter en printk med en signende tekst, således at du ved når probe bliver kaldt. Vi antager i denne driver at der netop er ét spi device per driver og gemmer derfor device pointeren i en static variabel i driveren. Dette er hvad som sker i linien *ads7870_spi_device = spi;* i *ads7870_spi_probe()*.

l) Implementer *Exit()* som du har gjort det i GPIO driveren, denne gang er det blot SPI resourcen du skal frigive. Se i ads7870.c og ads7870-spi.c hvordan

m) Implementer ''Remove''. Denne skal anvende printk til at give en passende besked.

n) Kopier den eksisterende Makefile og tilret den til de nye filnavne. Byg kernemodulerne og kopier dem til Devkit8000. På Devkit8000 gentager du punkterne (c) og (d). Hvad sker der når driver og device (hotplugxx) indsættes? Bliver *probe()* kaldt og hvad betyder det?

### Konstruktion af char driver Write
### Nu er det på tide at få banket noget funktionalitet på plads i driveren og du skal derfor igang med at implementere en skrive metode.

o) Opret ''open'' og ''release'' metoder som gjort tidligere. Disse skal pt ikke udføre noget arbejde.

p) I character driveren opretter du nu en ''write'' metode, igen præcis som du gjorde det i GPIO driveren. Denne gang vil vi dog gerne bruge minor nummeret til at styre hvilket "sub-device" som tilskrives på vores spi device. Dette kunne være hvilken kanal på en DAC som skal styres, eller en intern adresse på PSOC device'et, præcis som vi tidligere så det ifb med LM75 temperatur sensoren som havde fire interne adresser. Se i write metoden i ads7870.c hvordan man udleder minor nummeret. De data som skal tilskrives vore device skal naturligvis kopieres fra user space vha ''copy_from_user'' og konverteres til binær form. Vi har nu en adresse via minor nummeret og data fra user space og mangler blot at skrive det ud på SPI interfacet.

q) Når data skal kommunikeres ud på SPI interfacet, er det dit ansvar at sammenstykke en data streng som svarer overens med det man kan finde i databladet ( **PSOC4** / **DAC7612** / etc). Se funktionen ''ads7870_spi_write_reg8'' i ads7870-spi.c som tilskriver ADS7870 med 8-bit data. Du kan med fordel lægge SPI beskedopbygning og afsendelse i en funktion, som det er gjort her. Husk at lave godt med debug beskeder! Hvordan skal beskederne opbygges til det device du anvender? Beskriv hvordan og hvorfra du har din viden til at kunne gøre dette.

r) Byg din driver og gentag steps (c) - (e). Prøv Herefter at skrive en værdi til dit spi device:

```
echo 0xff > /dev/<my_spi_device_node>
```

Hvad sker der? Har det den ønskede effekt? Hvis du efter en tid ikke kan finde ud af om der sker noget, var det måske en idé at sætte et oscilloscope på SPI forbindelserne for at checke om kommunikationen sker som specificeret i databladet. Et oscilloscope plot i wikien giver en ekstra stor smiley hos underviser :-)

### Konstruktion af char driver Read

Det er næsten det samme som write, bare omvendt og lidt til...

s) Opret read metoden som sædvanligt, der hvor du fik data fra et GPIO input, skal du nu have noget fra SPI! Gør det samme som i write mht minor numre.

t) Opret en spi læse funktion. Se f.eks ''ads7870_spi_read_reg16'' i ads7870-spi.c. og anvend den i din read metode.

Bemærk!!! Bruger du PSOC boards, er der en forsinkelse mellem afsendelse af kommando (første transfer) og retur data (anden transfer) (PSOC4=50us/PSOC3=120us) da PSOC'en først lige skal have fundet data frem. Dette imødekommer man ved at sætte en parameter på den første transfer: .delay_usecs=150 (PSOC3):

```
t[0].delay_usecs = 150;
```

Hvorfor kan det at det er nødvendigt at lave en skrivning til spi slave device'et inden man foretager en læsning?

u) Byg din driver og gentag steps (c) - (e). Prøv Herefter at læse værdier fra dit spi device:

```
cat /dev/<my_spi_device_node>
```

Modtager du ikke data kan du som i (r) prøve med et oscilloscope. Prøver du at læse fra DAC7612, skal du nærlæse databladet én gang til og kigge efter MISO/DOUT forbindelsen.

### Hints!

ADS7870 driveren har 2 lag:

- ads7870.c : Character driver funktionerne samt hvordan der hentes en sampling fra A/D konverteren
- ads-7870-spi.c : Beskæftiger sig kun med formattering af data til SPI interfacet. Dette lag kan skiftes, hvis der f.eks skal bruges I2C som fysisk medie i stedet.

**SPI_potentiometer.jpg** (68,618 KB) Peter Høgh Mikkelsen, 2014-08-18 11:05