Wiki »

Exercise 7 - LDD with SPI ¶

Indsættelse af ADS7870 Driver i Linux

Byg koden vha den tilhørende Makefile

Koden bygges vha. den tilhørende Makefil. Resultatet er to kernemoduler: ''hotplug_ads7870_spi_device.ko'' og ''ads7870mod.ko''. Sidst nævnte er bygget ud fra 2 filer: ads7870.c og ads7870-spi.c.

```
stud@GoldenImage:~/HAL/ex7/ads7870$ make
make ARCH=arm CROSS_COMPILE=arm-angstrom-linux-gnueabi- -C ~/sources/linux-3.2.6 M=/home/stud/HAL/ex7/ad
s7870 modules
make[1]: Entering directory `/home/stud/sources/linux-3.2.6'
  Building modules, stage 2.
  MODPOST 2 modules
make[1]: Leaving directory `/home/stud/sources/linux-3.2.6'
stud@GoldenImage:~/HAL/ex7/ads7870$ ls -l
total 568
-rw------- 1 stud stud   6225 Oct 23 07:26 ads7870.c
-rw------- 1 stud stud   2172 Oct 23 07:26 ads7870.h
-rw-rw-r-- 1 stud stud 138539 Oct 23 08:03 ads7870mod.ko
-rw-rw-r-- 1 stud stud   1417 Oct 23 08:03 ads7870mod.mod.c
-rw-rw-r-- 1 stud stud  20980 Oct 23 08:03 ads7870mod.mod.o
-rw-rw-r-- 1 stud stud 118853 Oct 23 08:03 ads7870mod.o
-rw-rw-r-- 1 stud stud  75308 Oct 23 08:03 ads7870.o
-rw------- 1 stud stud   5759 Oct 23 07:25 ads7870-spi.c
-rw------- 1 stud stud    397 Oct 23 07:26 ads7870-spi.h
-rw-rw-r-- 1 stud stud  50572 Oct 23 08:03 ads7870-spi.o
-rw------- 1 stud stud   2281 Oct 23 07:26 hotplug_ads7870_spi_device.c
-rw-rw-r-- 1 stud stud  53191 Oct 23 08:03 hotplug_ads7870_spi_device.ko
-rw-rw-r-- 1 stud stud    856 Oct 23 08:03 hotplug_ads7870_spi_device.mod.c
-rw-rw-r-- 1 stud stud  19940 Oct 23 08:03 hotplug_ads7870_spi_device.mod.o
-rw-rw-r-- 1 stud stud  35896 Oct 23 08:03 hotplug_ads7870_spi_device.o
-rw-rw-r-- 1 stud stud   1052 Oct 23 07:59 Makefile
-rw------- 1 stud stud    242 Oct 23 07:26 mk_ads7870.sh
-rw-rw-r-- 1 stud stud    112 Oct 23 13:11 modules.order
-rw-rw-r-- 1 stud stud      0 Oct 23 08:03 Module.symvers
-rw------- 1 stud stud   1751 Oct 23 07:26 rd.c
stud@GoldenImage:~/HAL/ex7/ads7870$
```

Hvad er formålet med "hotplug_ads7870_spi_device.ko''
SVAR: SPI "cold-plugges" normal under boot. Formålet med "hotplug_ads7870_spi_device.ko" er at kunne hot-plugge et SPI device. Dvs. koble det til når imens systemet kører. "hotplug_ads7870_spi_device.ko" udfører "hot cold-pluggin"

b, c, d) Kopier begge kernemoduler til Devkit8000

Begge kernemoduler kopieres til DevKit8000 og ''ads7870mod.ko'' indsættes. Herved registreres driveren i Linux
Disse beskeder er tilføjet kerneloggen (ses via *dmesg*) Se skærmbillede herunder:

> *ADS7870 driver initializing*

```
[   66.181701] ADS7870 driver initializing
root@beagleboard:~# insmod ads7870mod.ko
```

"hotplug_ads7870_spi_device.ko" indsættes nu i kernen. Herved registreres et SPI device.
Disse beskeder er tilføjet kerneloggen (ses via *dmesg*) Se skærmbillede herunder:

> *Adding SPI Device: ads7880, bus: 1, chip-sel: 0*
> *Ned SPI device: ads7870 using chip select: 0*
> *Probing ADS7870, Revision 1*

```
[  129.338592] Adding SPI Device: ads7870, bus: 1, chip-sel: 0
[  129.372741] New SPI device: ads7870 using chip select: 0
[  129.376556] Probing ADS7870, Revision 1
root@beagleboard:~# insmod hotplug_ads7870_spi_device.ko
```

Hvad er det som er sket?
SVAR: Der tilføjes et SPI device med navnet ads7870, som sidder på bus1 og bruger chip select 0.

Hvilke metoder er blevet kaldt i device driveren (ads7870-spi.c) efter at vi har tilføjet et device?
SVAR:

Der bliver kaldt én metode i ads7870-spi.c driveren: ads7870_spi_probe(struct spi_device *spi)

Opret noder til udlæsning af ADC inputs

Der oprettes 8 noder til udlæsning af ADC inputs vha. et givet script.

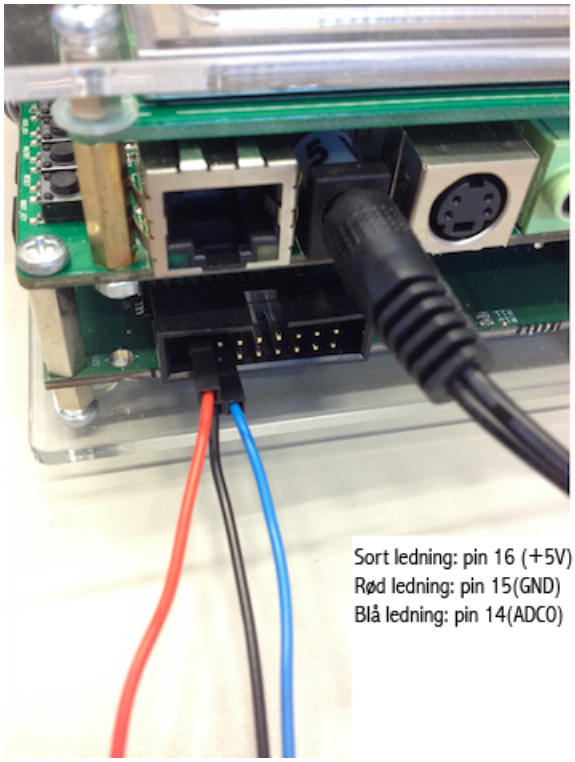
> *chmod +x /home/root/mk_ads7870.sh*
> *./home/root/mk_ads7870.sh*

Test af ADS7870

DevKit8000 Add-On board stik J4 er forbundet til A/D konverterens input. Billedet herunder viser forbindelsen. Tabellen er fundet her:
https://redmine.iha.dk/devs/projects/devkit8000/wiki/AddonInterfaces

Lån et analogt potentiometer på værkstedet om forbind det som vist herunder:

Potentiometerets midterben er forbundet til pin 14 (J4) vha. den blå ledning. Den røde og sorte ledning tilsluttes ben 1 og 3.
Potentiometeret er dog ikke et mellem 10K-1M ohm, men et 4,7K ohm.
Spændingen kan via potentiometeret justeres fra 0V til 5V, ingangen på A/D konverteren kan kun måle op til 2.047V, men tager ikke skade af at få 5V.



ADC / DAC (J4)

Connects to AD7870 + DAC7612
Voltage levels: DAC out: 0-4.096V, ADC in: 0-2.096V, GPIO: 5V
16-Pin Header, 2.54mm, Male

Pin	Name	OMAP IO	Direction
1	AOUT1	DAC_OUT_B	OUT
2	AOUT0	DAC_OUT_A	OUT
3	AGPIO3	ADC_GPIO3	BIDIR
4	AGPIO2	ADC_GPIO2	BIDIR
5	AGPIO1	ADC_GPIO1	BIDIR
6	AGPIO0	ADC_GPIO0	BIDIR
7	AIN7	ADC_AIN7	IN
8	AIN6	ADC_AIN6	IN
9	AIN5	ADC_AIN5	IN
10	AIN4	ADC_AIN4	IN
11	AIN3	ADC_AIN3	IN
12	AIN2	ADC_AIN2	IN
13	AIN1	ADC_AIN1	IN
14	AIN0	ADC_AIN0	IN
15	GND	GND	
16	+5V	+5V	

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:

For at teste anvendes *cat* til den adc kanal som potentiometeret er forbundet til. Vi har brugt ADC0.

> *cat /dev/adc0*

Nedenstående er udskriften i konsollen:

```
1196
1203
1201
1201
1199
1196
1198
1200
1202
1200
1200
1199
1199
1198
1198
1198
1198
1201
1197
1197
1202
1199
1201
1195
1199
1200
1194
1199
1201
1200
1200
1200
1197
1194
```

Konklusionen er at potentiometeret og AD konverteret virker som forventet.

Prøv at fjerne hotplug_ads7870_spi_device.ko modulet og herefter at læse fra /dev/adc0 igen. Vi får nu en fejl, men hvorfor?
SVAR:

Vi får nedenstående fejl:

> *cat: can't open '/dev/adc0'_ No such device*

Vi får fejlen da vores device ikke er registreret da hotplug_ads7870_spi_device.ko modulet er taget ud.

Hvor i koden er det at vi bliver stoppet i at tilgå et device som ikke eksisterer?
SVAR: Det gør vi i "ads7870_cdrv_open(struct inode *inode, struct file *filep" (ads7870.c) kommer vi til en if sætning:

```
1  /* Check if a ads7870 device is registered */
2    if(!(ads7870_spi_device=ads7870_get_device()))
3      return -ENODEV;
```

Vi kommer ind i if sætningen da SPI device og get device er forskellige, herefter stoppes vi, da devicet ikke eksisterer.

Konstruktion af egen SPI device driver

Herunder konstrueres egen SPI device driver. Stepsne heri er angivet med overskrifterne herunder.

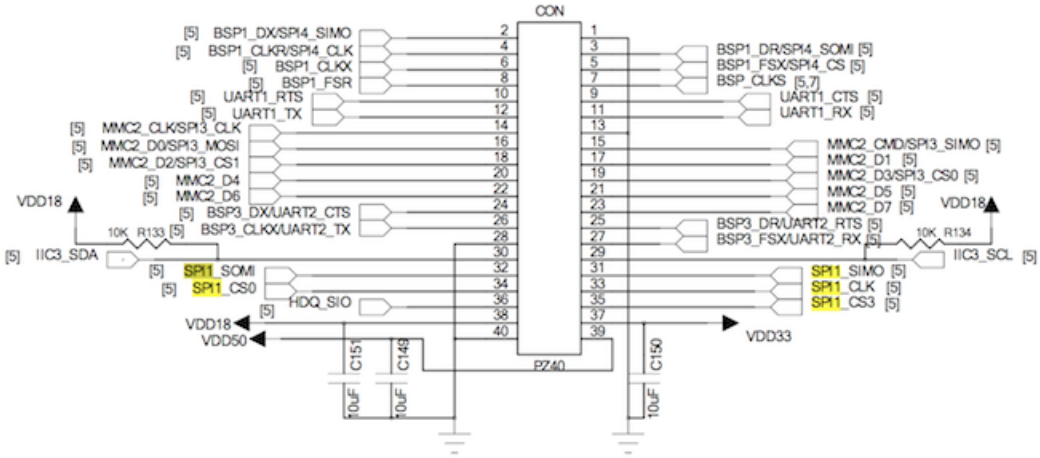
Konstruktion af SPI hot-plug kernemodul

Ud fra 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

Vi ønsker at anvende PSoC4 som SPI device

SPI bus nummer
SPI bus nummeret er fundet vha.
https://redmine.iha.dk/devs/projects/devkit8000/repository/revisions/master/entry/documentation/DevKit8000_schematic.pdf
På side 12 ses en CON port (Connector) Heref ses at SPI1 føres ud af DevKit8000 via CON porten.



Desuden ses på **https://redmine.iha.dk/devs/projects/devkit8000/wiki/AddonInterfaces** at SPI1 er den SPI bus der kører ud af J9 stikket på add-on printet.

Pin	Name	OMAP IO	Direction
1	SPI1_SIMO	SPI1_SIMO	OUT
2	SPI1_SOMI	SPI1_SOMI	OUT
3	SPI1_CS0	SPI1_CS0	OUT
4	SPI1_CS3	SPI1_CS3	OUT
5	SPI1_CLK	SPI1_CLK	OUT
6			
7	GND		

SPI chip-select (ss) nummer

Tabellen ovenover viser at der er mulighed for at vælge chip select 0 eller chip select 3. Chip Select 0 vælges!

SPI max frekvens

Ud fra side 22 i databladet for PSoC4 **http://www.cypress.com/?docID=46322** , ses SPI max frekvens er 8MHz. Da vi ikke ønsker at køre med max frekvensen vælger vi at benytte 1MHz

Table 28. Fixed SPI AC Specifications

(Guaranteed by Characterization)

Spec ID	Parameter	Description	Min	Typ	Max	Units
SID166	F_{SPI}	SPI operating frequency (master; 6X oversampling)	–	–	8	MHz

SPI Clock Mode
Ud fra spi.h (**http://lxr.free-electrons.com/source/include/linux/spi/spi.h**) ses herunder at 4 forskellige modes er muligt. Vi har valg SPI_MODE_3. Herved er det bestemt at CPOL/CPHA = 11.

```
83 #define SPI_CPHA        0x01                    /* clock phase */
84 #define SPI_CPOL        0x02                    /* clock polarity */
85 #define SPI_MODE_0      (0|0)                   /* (original MicroWire) */
86 #define SPI_MODE_1      (0|SPI_CPHA)
87 #define SPI_MODE_2      (SPI_CPOL|0)
88 #define SPI_MODE_3      (SPI_CPOL|SPI_CPHA)
```

Antal bits per transmission

Ud fra databladet (**https://redmine.iha.dk/devs/attachments/download/52/P4_SpiSlave_CapSense_datasheet.pdf**) Ses at read er en 8 bit funktion og write er en 16 bit funktion

Derfor benyttes 8 bit pr. transmission.

Opsumering af information

SPI bus nummer: SPI1

SPI chip-select (ss) nummer: CS0

SPI max frekvens: 1 MHz (Vi oplevede under implementeringen og test at kommunikationen ikke virkede ved 8MHz, så vi har valgt en væsentlig lavere clock frekvens)

SPI Clock Mode: SPI_MODE_3

Antal bits per transmission: 8 bit

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?

hotplug_ads7870_spi_device.c er kopieret og navngivet til *hotplug_psoc4_spi_device.c*. Parametrene er fra ovenstående opgave er opdateret.
Vi har "hotplug" modulet, da det gør det muligt at tilføje et device efter bootup. Formålet er her at tilføje et SPI decive (PSoC4), selvom DevKit8000 er startet op.
Ændringerne ses herunder:

```
 1 /*
 2  * Slave Device Config
 3  */
 4 static struct spi_board_info slave_spi_board_info = {
 5   .modalias        = "psoc4_spi",    // Alias to look for
 6   .bus_num         = 1,        // SPI Bus no.
 7   .chip_select       = 0,        // Chip Select No.
 8   .max_speed_hz       = 1000000,    // Max Clock Speed
 9   .controller_data    = &mcspi_config,//
10   .mode            = SPI_MODE_3,    // Clock polarity
11 };
```

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.

Første erklæres structs og variable som init funktionen benytter:

```
 1 /* Char Driver Globals */
 2 static struct cdev psoc4Dev;
 3 struct file_operations psoc4_Fops;
 4 static int devno;
 5
 6 /* We are only using ONE SPI device so far */
 7 static struct spi_device *psoc4_spi_device = NULL;
 8
 9 /* Method to retrieve SPI device pointer */
10 struct spi_device* psoc4_get_device(void){
11   return psoc4_spi_device;
12 }
```

Herunder ses init implementeret.

```
 1 /*
 2  * PSoC4 SPI Init
 3  * Registers the spi driver with the SPI host
 4  */
 5 int psoc4_cdrv_init(void)
 6 {
 7       int err;
 8
 9       if(MODULE_DEBUG)
10     printk(KERN_DEBUG "TEST: psoc4.c psoc4_spi_init\n");
11
12       /* Registers the spi driver with the SPI host*/
13       err = spi_register_driver(&psoc4_spi_driver);
```

```
14        if(err){
15      printk (KERN_ALERT "Error %d registering the ads7870 SPI driver\n", err);
16      ERRGOTO(error, "Failed SPI Initialization\n");
17        }
18
19        /* Allocate chrdev region */
20        devno = MKDEV(PSOC4_MAJOR, PSOC4_MINOR);
21        err = register_chrdev_region(devno, NBR_PSOC4_CH, "psoc4");
22        if(err){
23      ERRGOTO(err_spi_init, "Failed registering char region (%d,%d) +%d, error %d\n",
24          PSOC4_MAJOR, PSOC4_MINOR, NBR_PSOC4_CH, err);
25        }
26
27        /* Register Char Device */
28        cdev_init(&psoc4Dev, &psoc4_Fops);
29        err = cdev_add(&psoc4Dev, devno, NBR_PSOC4_CH);
30        if (err){
31      ERRGOTO(err_register, "Error %d adding PSOC4 device\n", err);
32
33        }
34
35        return 0;
36
37    err_register:
38    unregister_chrdev_region(devno, NBR_PSOC4_CH);
39
40    err_spi_init:
41    spi_unregister_driver(&psoc4_spi_driver);
42
43    error:
44    return err;
45  }
46
```

SPI driver structen er nødvendig for init funktionen. Herunder ses implementeringen:

```
1  /*
2   * PSOC4 SPI Driver Struct
3   *
4   */
5  static struct spi_driver psoc4_spi_driver = {
6        .driver = {
7      .name   = "psoc4",
8      .bus    = &spi_bus_type,
9      .owner = THIS_MODULE,
10       },
11 };
```

Init() har det formål at registrere SPI driveren med SPI hosten. Der allokeres plads vha. MAJOR og MINOR numre og char devicet registreres. Desuden håndteres eventuelle fejl.

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().

Probe() ses herunder implementeret. Det er heri at bit pr transmission sættes til 8 bit og SPI sættes op.

```
1  /*
2   * PSOC4 Probe
3   * Used by the SPI Master to probe the device
4   * when an SPI device is registered.
5   */
6  static int __devinit psoc4_spi_probe(struct spi_device *spi)
7  {
8        int err = 0;
9
10       printk(KERN_DEBUG "New SPI device: %s using chip select: %i\n",
11         spi->modalias, spi->chip_select);
12
13       spi->bits_per_word = 8;  // Bit per Transfer
14       spi_setup(spi);
15
16       if(MODULE_DEBUG)
17     printk(KERN_DEBUG "TEST: psoc4.c psoc4_spi_probe\n");
18
19       /* In this case we assume just one device */
20       psoc4_spi_device = spi;
21
22       return err;
23 }
```

psoc4_spi_driver structen skal opdateres - det ses herunder

```
1  /*
2   * PSOC4 SPI Driver Struct
3   * Holds function pointers to probe/release
4   * methods and the name under which it is registered
5   *
6   */
7  static struct spi_driver psoc4_spi_driver = {
8        .driver = {
9      .name   = "psoc4",
10     .bus    = &spi_bus_type,
11     .owner = THIS_MODULE,
12       },
13       .probe = psoc4_spi_probe,
14       .remove = __devexit_p(psoc4_remove),
15 };
```

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

```
1  /*
2   * PSoC4 SPI Exit
```

```
 * Exit routine called from character driver.
 * Unregisters the driver from the SPI host
 */
void psoc4_cdrv_exit(void)
{
      printk("psoc4 driver Exit\n");
      cdev_del(&psoc4Dev);

      unregister_chrdev_region(devno, NBR_PSOC4_CH);

      spi_unregister_driver(&psoc4_spi_driver);
}
```

Eftersom vi har lavet alt koden i én psoc.c fil, har vi implementeret *spi_unregister_driver()* i *psoc4_cdrv_exit()* og lagt *spi_unregister_driver()* i fejlhåndteringen i init funktionen, istedet for at kalde en funktion der gør det samme

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

```
/*
 * PSOC4 Remove
 * Called when the SPI device is removed
 */
static int __devexit psoc4_remove(struct spi_device *spi)
{
      struct psoc4 *psoc4dev = dev_get_drvdata(&spi->dev);

      if(MODULE_DEBUG)
    printk(KERN_DEBUG "TEST: psoc4.c psco4_remove\n");

      psoc4_spi_device = 0;

      printk (KERN_ALERT "Removing SPI device %s on chip select %i\n",
          spi->modalias, spi->chip_select);

      kfree(psoc4dev); /* Free kernel memory */

      return 0;
}
```

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?

Makefilen opdateres med de 2 nedenstående linjer:

```
obj-m += psoc4mod.o hotplug_psoc4_spi_device.o
 psoc4mod-objs := psoc4.o
```

Kernemodulerne bygges og kopires til DevKit8000.

Når psoc4mod.ko indsættes, skriver kerneloggen at *psoc4_cdrv_init()* køres. Når hotplug modulet indsættes herefter. Tilføjes et SPI device

Beskederne fra kerneloggen er vist herunder

Når psoc4mod.ko indsættes:

> *[1258.286499] TEST: psoc4.c psoc4_cdrv_init*

Når hotplug indsættes:

> *[1378.471130] Adding SPI Device: psoc4, bus: 1, chip-sel: 0*
> *[1378.491699] New SPI device: psoc4 using chip select: 0*
> *[1378.491851] TEST: psoc4.c psoc4_spi_probe*

Hvad er det som er sket?
SVAR: Der tilføjes et SPI device med navnet psoc4, som sidder på bus1 og bruger chip select 0.

Hvilke metoder er blevet kaldt i device driveren (psoc4mod.c) efter at vi har tilføjet et device?
SVAR: Der bliver kaldt én metode i psoc4mod.c driveren: psoc4_spi_probe(struct spi_device *spi)

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.

Herunder ses open og release metoderne, disse er blot oprettet.

```
*
 * PSoC4 CDRV OPEN
 *
 */
int psoc4_cdrv_open(struct inode *inode, struct file *filep)
{
  /*
      return 0;
}

/*
 * PSoC4 CDRV RELEASE
 *
 */
int psoc4_cdrv_release(struct inode *inode, struct file *filep)
{

      return 0;
}
```

FOPS structen oprettes:

```
/*
 * PSOC4 SPI FOPS Struct
 *
```

```
 */
struct file_operations psoc4_Fops =
{
      .owner    = THIS_MODULE,
      .open     = psoc4_cdrv_open,
      .release = psoc4_cdrv_release,
};
```

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.

```
/*
 * PSoC4 CDRV WRITE
 */
ssize_t psoc4_cdrv_write(struct file *filep, const char __user *ubuf,
                size_t count, loff_t *f_pos)
{
      int psoc4_function = PSOC4_RGBLED;    // PSoC4 function to Write too

      int err, myVar, minor, len;
      char myArray[MAXLEN];

      minor = MINOR(filep->f_dentry->d_inode->i_rdev);

      printk(KERN_ALERT "Writing to PSoC4 [Minor] %i \n", minor);

      /* Limit copy length to MAXLEN allocated and Copy from user */
      len = count < MAXLEN ? count : MAXLEN;
      if(copy_from_user(myArray, ubuf, len))
    return -EFAULT;

      /* Pad null termination to string */
      myArray[len] = '\0';

      // http://www.gnugeneration.com/mirrors/kernel-api/r4343.html
      err = copy_from_user(myArray, ubuf, sizeof(ubuf));

      if(err != 0)
      {
    printk(KERN_ALERT "ERROR in copy_from_user, ERRORCODE: %d\n" , err);
      }

      sscanf(myArray, "%i", &myVar);

      /*
      * Do something
      */
      if(MODULE_DEBUG)
    printk(KERN_ALERT "Writing %i to address: %d\n", myVar, psoc4_function);

      /*
       * Call SPI WRITE
       */

      psoc4_spi_write_reg8(psoc4_spi_device, psoc4_function, (u8)myVar);

      /* Legacy file ptr f_pos. Used to support
      * random access but in char drv we dont!
      * Move it the length actually  written
      * for compability */
      *f_pos += len;

      /* return length actually written */
      return len;
}
```

Write funktionen læser *copy_from_user* og ligger en givet værdi i noden for det oprettede device.
psoc4_spi_write_reg8 modtager en addresse og værdien og sync'er det til PSoC4 via SPI.

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.

Ud fra addr parameteren, oprettes en cmd variabel, som bliver lagt i tx bufferen (Denne indeholder at det er en write metode, samt den ønskede PSoC4 funktions-adresse)
psoc4_spi_write_reg8 modtager en funktion-adresse og en værdi, skriver dem i TX bufferen og syncer med PSoC4.
Vi har tilegent os viden ved at tage udgangspunkt i ads7870 projektet og herigennem bygget egen write metode op. Dette er gjort samtidigt med debugging.

```
/*
 * PSOC4 SPI Write 8-bit Register
 * Writes 8-bit content to register at
 * the provided PSOC4 function address
 */
int psoc4_spi_write_reg8(struct spi_device *spi, u8 addr, u8 data)
{
      struct spi_transfer t[2];    // Number of packages
      struct spi_message m;
      u8 cmd;            // Local var for WRITE bit and ADDRESS

      if(MODULE_DEBUG)
    printk(KERN_DEBUG "TEST: psoc4.c psoc4_spi_write_reg8\n");

      /* Check for valid spi device */
    if(!spi)
    return -ENODEV;

      /* Create start WR CMD byte:
      *
      * | 0 | 1 |     ADDR       |
      *   7   6  5  4  3  2  1  0
      */

```

```
25        cmd =  (0<<7) | (1<<6) | (addr & 0x1F);
26
27        /* Init Message */
28        memset(&t, 0, sizeof(t));
29        spi_message_init(&m);
30        m.spi = spi;
31
32        if(MODULE_DEBUG)
33      printk(KERN_DEBUG "PSOC4: Write Reg8 Addr 0x%x Data 0x%02x\n", addr, data);
34
35        /* Configure tx/rx buffers */
36        t[0].tx_buf = &cmd;
37        t[0].rx_buf = NULL;
38        t[0].len = 1;
39        spi_message_add_tail(&t[0], &m);
40
41        t[1].tx_buf = &data;
42        t[1].rx_buf = NULL;
43        t[1].len = 1;
44        spi_message_add_tail(&t[1], &m);
45
46        /* Transmit SPI Data (blocking) */
47        spi_sync(m.spi, &m);
48
49        return 0;
50  }
51
```

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

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

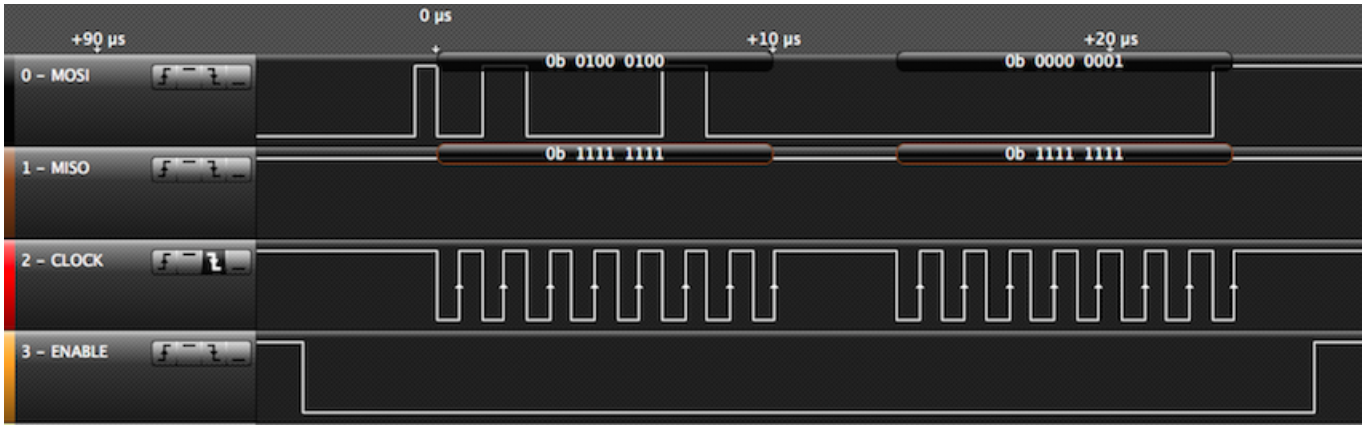
Modulet indsættes og SPI devicet tilføjes. Herefter oprettes noden */dev/psoc4*, som er den vi skriver en værdi til.

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 :-)

Der skrives 1 til noden "psoc4"

```
1  echo 1 > /dev/psoc4
```

Herunder ses scop billedet fra logik analysatoren



Som det ses af både ovenstående scop billede og på led dioden på PSoC4, så skrives værdien '1' til rgb led dioden, herved lyser denne grøn. Scop billede viser at cmd variablen (01000100) og date værdien (00000001)

Konstruktion af char driver Read

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.

Read metoden er implementeret herunder. Der oprettes en variabel result, hvis adresse bliver benyttet i *psoc4_spi_read_reg16* vha en reference. Denne benyttes så vi kan udskrive data.
I linje 22, ses at *ddr = PSOC4_DUMMY;*, Herved forventes at der udskrives 0xBEEF, som er bestemt i PSoC4 demo projektet.

```
1  /*
2   * PSoC4 CDRV READ
3   */
4  ssize_t psoc4_cdrv_read(struct file *filep, char __user *ubuf,
5                          size_t count, loff_t *f_pos)
6  {
7        int minor, len;
8        char resultBuf[MAXLEN];
9        s16 result;
10       u8 addr;
11
12       if(MODULE_DEBUG)
13     printk(KERN_DEBUG "TEST: psoc4.c psoc4_cdrv_read\n");
14
15       minor = MINOR(filep->f_dentry->d_inode->i_rdev);
16
17       if(MODULE_DEBUG)
18     printk(KERN_ALERT "Reading from psoc4 [Minor] %i \n", minor);
19
20       /* Perform SPI read */
21
22       addr = PSOC4_DUMMY;
23
24       psoc4_spi_read_reg16(psoc4_spi_device, addr , &result);
25
26       len = snprintf(resultBuf, count , "%d\n", result);
```

```
27        len++;
28
29        /* Copy data to user space */
30        if(copy_to_user(ubuf, resultBuf, len))
31      return -EFAULT;
32
33        /* Move fileptr */
34        *f_pos += len;
35
36        return len;
37  }
38
```

t) Opret en spi læse funktion. Se f.eks "ads7870_spi_read_reg16" i ads7870-spi.c. og anvend den i din read metode.

Hvorfor kan det at det er nødvendigt at lave en skrivning til spi slave device'et inden man foretager en læsning?
SPI devicet skal have at vide hvad man gerne vil læse, derfor skal der skrivet til devicet inden.
Det er nødvendigt at sætte et delay ind (50us), da der skal bruges tid på at finde data frem.

```
1  /*
2   * PSoC4 SPI Read 16-bit Register
3   * Reads 16-bit content of register at
4   * the provided PSoC4 address
5   */
6  int psoc4_spi_read_reg16(struct spi_device *spi, u8 addr, u16* value)
7  {
8         struct spi_transfer t[4];
9         struct spi_message m;
10        u8 cmd[2];
11        u8 data[2];
12        u16 dataw;
13
14        if(MODULE_DEBUG)
15      printk(KERN_DEBUG "TEST: psoc4.c psoc4_spi_read_reg16\n");
16
17        /* Check for valid spi device */
18        if(!spi)
19      return -ENODEV;
20
21        /* Create Cmd byte:
22        *
23        * | 0 |  0 |       ADDR       |
24        *   7    6   5  4  3  2  1  0
25        */
26        cmd[1] = (0<<7) | (0<<6) | (addr & 0x1f);
27        cmd[0] = 0;
28
29        /* Init Message */
30        memset(t, 0, sizeof(t));
31        spi_message_init(&m);
32        m.spi = spi;
33
34        /* Configure tx/rx buffers */
35        t[0].tx_buf = &cmd[1];
36        t[0].rx_buf = NULL;
37        t[0].len = 1;
38        spi_message_add_tail(&t[0], &m);
39
40        t[1].tx_buf = &cmd[0];
41        t[1].rx_buf = NULL;
42        t[1].len = 1;
43        t[1].delay_usecs = 50;
44        spi_message_add_tail(&t[1], &m);
45
46        t[2].tx_buf = NULL;
47        t[2].rx_buf = &data[1];
48        t[2].len = 1;
49        spi_message_add_tail(&t[2], &m);
50
51        t[3].tx_buf = NULL;
52        t[3].rx_buf = &data[0];
53        t[3].len = 1;
54        spi_message_add_tail(&t[3], &m);
55
56        /* Transmit SPI Data (blocking) */
57        spi_sync(m.spi, &m);
58
59        dataw = data[1] << 8 | data[0];
60
61        if(MODULE_DEBUG)
62      printk(KERN_DEBUG "PSOC4: Read Reg16 Addr: 0x0%d Data: %d\n", addr, dataw);
63
64        *value = dataw;
65        return 0;
66 }
67
```

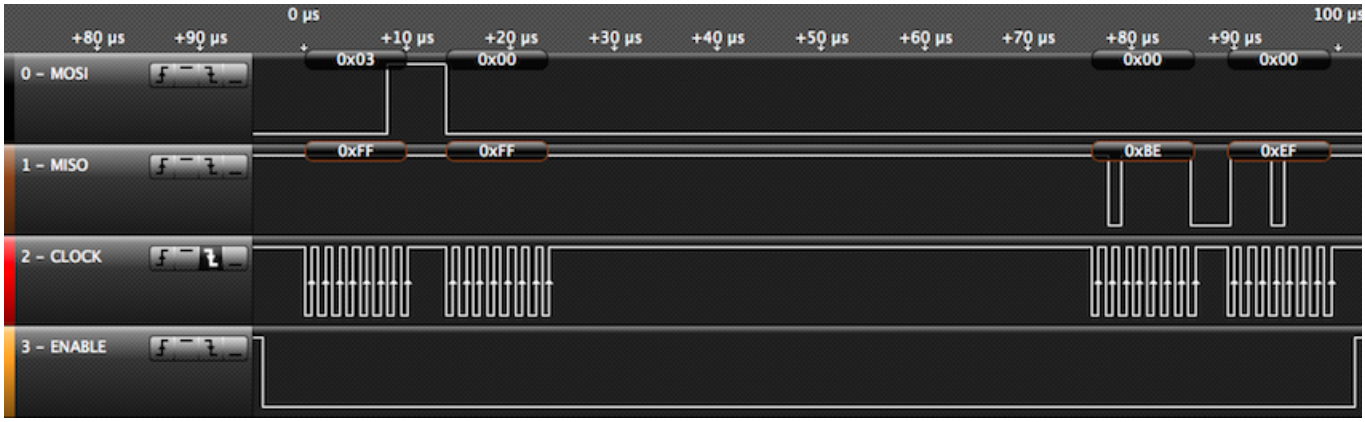
u) Byg din driver og gentag steps (c) - (e). Prøv Herefter at læse værdier fra dit spi device:

Stepsne er gentaget, intet nyt under solen.

Der læses fra devicet via:

> *cat /dev/psoc4*

Herunder ses scop billedet. Der retuneres netop 0xBE og 0xEF, som ønsket.



Appendix

For at gennemføre ovenstående tests skal nedenstående udføres:

Forbind DevKit8000 og PSoC4 som tabellen herunder foreskriver:

Devkit	Signal	PSOC4
J9.1	MOSI	J3.4 (P3 [0])
J9.2	MISO	J3.5 (P3 [1])
J9.3	SS0	J4.3 (P0 [7])
J9.5	SCK	J3.6 (P0 [6])
J9.20	GND	J3.7 (GND)

Programmer PSoC4 med kode fra demoprojektet :
https://redmine.iha.dk/devs/attachments/download/51/P4_SpiSlave_CapSense.cyprj.Archive03.zip

OBS! device.h skal ændres til project.h! Da PSoC creator 3.0 ikke bruger device.h....

Husk også at nedenstående skal skrives til Devkittet inden kernemodulet indsættes. Dette gøres for at route addon boardets SPI ud af J9 stikket.

> *echo 0x3 > /sys/class/cpldrv/cpld/spi_route_reg*
> *echo 0x1 > /sys/class/cpldrv/cpld/ext_serial_if_route_reg*

- Tilføjet af Jakob Schmidt for 28 dage siden

Punkter vi var valgt at lægge vægt på af review

Punkt1: forståelse af cold-plug og hot-plug

Punkt2: Forståelse af udleveret kernemoduler

Punkt3: Implementering af SPI

1)

Besvarelsen af cold-plug er lidt mangelfuld, men dog korrekt. Hvad vil det siges at det plugges under boot? Dette kunne i godt beskrive tydligere.
Hot-plug er korrekt beskrevet, men hvad vil det sige, at I hot cold-plugger "hotplug_ads7870_spi_device.ko"? Uddyb dette nærmere.

2)

I overfører og bygger kernerne korrekt. Beskrivelsen, for hvad der sker under dette, er også fin. Testen med potentiometeret ser rigtig fin ud.
Beskrivelsen for hvad der sker når hotplug_ads7870_spi_device.ko fjernes giver I ikke nogen særlig god forklaring på!

3)

Den første del hvor i finder og beskriver Bus-nummer, chip-select osv. er rigtig flot beskrevet med gode henvisninger til hvor i har fundet informationerne. I får skrevet de fundende info. ind i jeres struct som forventet.
Kommentarer til implementering af den nye c-driver fil er mangelfulde for ikke at sige helt udeladt. Vi kan ikke se om i har forsåelse for koden, eller om det er copy-paste. Dette er et generelt problem for alle jeres kodeudsnit.
Jeres test hvor i skriver til device er rigtig god. Det er lækkert med billeder fra analysatoren, dette giver et fint billede for hvad der sker. Samme gør sig gældende for Read funktionen, dog er kommentarer til koden igen mangelfulde.

Samlet konklusion
I skal oppe jer med kommentarer til jeres kodeudsnit. Det fremgår ikke tydeligt nok om i forstår koden. I er rigtig gode til at henvise til de steder hvor i finder information. Ligeledes er jeres test rigtig godt dokumenteret.
Alt i alt er opgaven **godkendt**, men forklaringerne er for mangelfulde.

- Tilføjet af Peter Høgh Mikkelsen for 15 dage siden

Men thumbs up for logic analyzer billeder
/Peter

dmesg_1.png (16,352 KB) Poul Overgaard Pedersen, 2014-10-23 11:18
dmesg_2.png (30,371 KB) Poul Overgaard Pedersen, 2014-10-23 11:18
Makefile.png (94,069 KB) Poul Overgaard Pedersen, 2014-10-23 11:18
g_cat_udskrift.png (13,773 KB) Poul Overgaard Pedersen, 2014-10-23 12:39
SPI1_bus.png (165,584 KB) Poul Overgaard Pedersen, 2014-10-27 11:54
SPI_freq_max.png (202,302 KB) Poul Overgaard Pedersen, 2014-10-27 11:57

CPHA.png (170,476 KB) Poul Overgaard Pedersen, 2014-10-27 12:07
CPOL.png (219,104 KB) Poul Overgaard Pedersen, 2014-10-27 12:11
rise_fall.png (193,843 KB) Poul Overgaard Pedersen, 2014-10-27 12:13
SPI1.png (142,391 KB) Poul Overgaard Pedersen, 2014-10-28 11:49
J9.png (57,424 KB) Poul Overgaard Pedersen, 2014-10-28 11:53
SPI_PSOC_MAX.png (57,311 KB) Poul Overgaard Pedersen, 2014-10-28 12:32
mode3.png (92,722 KB) Poul Overgaard Pedersen, 2014-10-28 12:32
potentiometer_connection.png (347,174 KB) Poul Overgaard Pedersen, 2014-10-31 11:23
dev_psoc4.png (37,943 KB) Poul Overgaard Pedersen, 2014-11-03 12:29
write_1.png (219,668 KB) Poul Overgaard Pedersen, 2014-11-03 13:13
beef.png (150,865 KB) Poul Overgaard Pedersen, 2014-11-03 13:30
psoc4_spi_src.zip - Kode (5,173 KB) Mick Kirkegaard Nielsen, 2014-11-04 11:25