

Software Implementering

Semesterprojekt 3. Semester

Gruppe 10

Vejleder: Søren Hansen

Gruppemedlemmer:

Navn	Studienummer
Tonni Nybo Follmann	201504573
Stefan Nielsen	201508282
Mikkel Espersen	201507348
Halfdan Vanderbruggen Bjerre	20091153
Ahmad Sabah	201209619
Jacob Munkholm Hansen	201404796

Indhold

Indhold

i

Dokumentation for brugergrænseflade

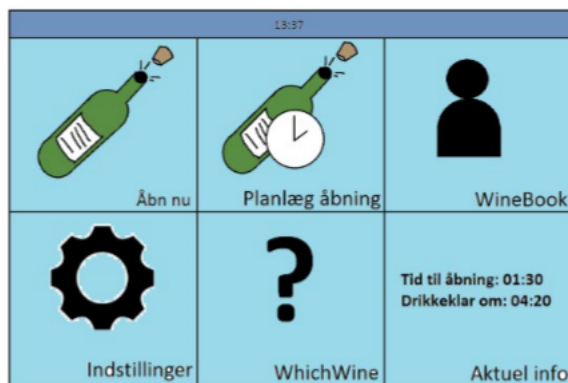
Indledning

For at en bruger kan interagere med WinePrep skal der være en brugergrænseflade. Brugergrænsefladen skal give brugeren mulighed for at kunne benytte WinePreps funktioner. Der skal være mulighed for at brugeren kan benytte sig af muligheden "Åben nu" som åbner vinen straks efter at kommandoen er givet. Yderligere skal der være mulighed for at brugeren kan planlægge åbningen af vinen således at man kan bestemme et tidspunkt på åbning af vinen. På brugergrænsefladen skal det også være muligt at kunne tilgå WineBook, WhichWine, Indstillinger og Aktuel info. Under Aktuel info skal der fremgå hvor lang tid der er tilbage til åbning af vinen. Der er yderligere blevet gjort nogle overvejelser over designet på brugergrænsefladen. Nogle af disse overvejelser vil blive præsenteret her.

Design og implementering

Skitsering af brugergrænseflade

Da touch-mekanismen på Devkit8000 ikke er helt optimal er det blevet besluttet at der skal benyttes store knapper således det ikke bliver svært at vælge de muligheder en bruger beslutter sig for at vælge. Den første skitse for brugergrænsefladen så således ud:



Figur 0.1: Første skitse af brugergrænsefladen

Første udgave af brugergrænseflade

På første udgave af produktet blev det dog besluttet at der ikke skal være nogen grafik på brugergrænsefladen. Yderligere blev det besluttet at det kun er funktionen Åbn nu der skal implementeres da det er denne funktion der danner grundlaget for funktionaliteten af produktet. Derfor blev Real Time Clocken (RTC) heller ikke implementeret, og derfor kunne tiden heller ikke visen, hverken klokkeslættet eller på Aktuel info. Et foreløbigt design på brugergrænsefladen ser således ud:



Figur 0.2: Første skitse af brugergrænsefladen

Her kan det ses at der ikke er blevet sat fokus på design og grafik. Når der trykkes på knappen åben nu, vil programmet sende en kommando til PSOC ved hjælp af SPI om at flasken skal registreres. Dog bruges kun kommandoerne fread og fwrite til at skrive og læse fra en character device driver. Denne kode ses her:

```
FILE *fd;
char buff[2];
fd = fopen(PSOC, "r+");
buff[0] = REGISTER_BOTTLE;
buff[1] = '\0';
fwrite(buff,1,2,fd);
```

Figur 0.3: Eksempel på brug af fwrite funktionen

For at få svar på om hvorvidt en flaske er registreret eller ej skal der læses via character device driveren. Dette gøres med funktionen fread(). Fwrite() ses brugt i følgende eksempel hvor der skal læses om flasken er blevet registreret:

```
buff[0]=0;
while(buff[0]!=VALID_TYPE || buff[0]!=INVALID_TYPE || buff[0]!=NO_BOTTLE)
{
    fread(buff,2,1,fd);
}
```

Figur 0.4: Eksempel på brug af fread funktionen

Først nulstilles bufferen som skal indeholde kommandoen, og derefter læses der i en whilelykke. Når fread() læser VALID_TYPE, INVALID_TYPE eller NO_BOTTLE vil whilelykken brydes og den læste kommando vil indsættes buff's første plads. Når værdien indsættes i buff gennemgås en switch-case hvor der alt efter hvilken værdi der er blevet læst ind i buff reageres på forskellige måder. Et eksempel ses her:

```
switch (buff[0]) {
case VALID_TYPE:
{
    in_progress obj1;
    obj1.setModal(true);
    obj1.exec();
}
break;
```

Figur 0.5: Eksempel på brug af switch-case efter at have læst fra PSOC

Denne kode vil resultere i at der kommer et vindue op hvor der står "Vinen åbnes. Vent venligst...". Man vil få muligheden for at trykke ok for at få vinduet til at forsvinde igen. Efter dette vil vinåbningen enten lykkedes eller fejle. Devkit8000 vil derefter få besked om åbningen er lykkedes eller om den er fejlet. Der vil endnu engang benyttes en switch case til at bestemme hvilken besked brugeren får frem på brugergrænsefladen. Selve klikfunktionen på de forskellige knapper er implementeret ved at højreklikke på de forskellige knapper og trykke på goto. Herefter vil der dukke en ny menu op. Her vælges clicked(). Her vil man få mulighed for at bestemme hvilken funktion knappen skal have når den trykkes. Et eksempel ses her:

```
void MainWindow::on_pushButton_5_clicked()
{
    hideMainMenu();
    showPlanlaegMenu();
}
```

Figur 0.6: Implementering af trykknappen Planlæg åbning

Dette er implementeringen for planlæg åbning knappen. Det der sker når man trykker på knappen er, som det fremgår på kodeeksemplet, at hovedmenuen skjules, og at Planlægningsmenuen vises. Måden hvorpå menuskiftsfunktionen er blevet implementeret er ved brug af show og hide funktionerne i QT. Dette kan ses i følgende eksempel hvor tilbageknappen skjuler alle knapper på den nuværende menu, for derefter at vise alle knapper på hovedmenuen:

```
void MainWindow::on_pushButton_2_clicked()
{
    ui->label->hide();
    ui->pushButton->hide();
    ui->pushButton_3->show();
    ui->pushButton_4->show();
    ui->pushButton_5->show();
    ui->pushButton_6->show();
    ui->pushButton_7->show();
    ui->pushButton_8->show();
    ui->pushButton_2->hide();
    ui->comboBox->hide();
    ui->comboBox_2->hide();
}
|
}
```

Figur 0.7: Skift af menu med tryk på tilbageknappen

Test

Som tidligere beskrevet er der i første udgave kun implementeret Åbn nu funktionen. Derfor er det kun denne funktion der kan testes. Dog kan navigationen på de forskellige menuer på brugergrænsefladen også testes. Menuerne Wine-Book, WhichWine, Indstillinger og Aktuel info er ikke blevet implementeret i denne udgave, og derfor vil test af disse menuer ikke fremgå i denne test.

Read og write funktionerne testes ved at tilkoble en PSoC til Devkit8000 og forsøge at skrive og læse til og fra den. Der er blevet konstrueret nogle meget simple forsøg, hvor funktionerne fread og fwrite bruges. Forsøget er konstrueret således at en led på PSoC'en lyser hver gang der læses fra den. Yderligere vil det der læses fra PSoC'en skrives til en fil på devkitted. Forsøget hvor der læses fra Devkit8000 er vedhæftet som en fil i bilaget, hvor man kan se at led'en på PSoC'en tænder og slukker hver gang der trykkes på Åbn nu på brugergrænsefladen. Koden der er brugt til at teste forbindelsen med er følgende:

```
FILE *fd;
int buff[2];
fd = fopen(PSoC, "r");
fread(buff, 2, 1, fd);

FILE *fd1;

fd1=fopen("/home/root/test2.txt", "w");

fwrite(buff, 1, 2, fd1);
fclose(fd1);
```

Figur 0.8: Kode til test af forbindelse

Denne kode tester blot fread og fwrite funktionerne. PSoC'en er programmeret således at dens LED lyser hver gang der bliver læst fra den. Yderligere gemmer koden det den læser i en tekstfil på devkitted.

Når man trykker på knappen Planlæg åbning dukker følgende menu op:

Figur 0.9: Menu til planlæg åbning med scrolldown menu åben for timer

For minutter ser det således ud:

Figur 0.10: Menu til planlæg åbning med scrolldown menu åben for timer

Resultater

Da hele produktet ikke er fuldt udviklet er det ikke muligt at fremvise nogle resultater. Man kan se på videoen at vi kommer til at tænde og slukke for dioden ved hjælp af brugergrænsefladen på devkittet. Videoen kan ses i bilaget.

Konklusion

Alle test gik som forventet, da dioden på PSoC'en kom til at tænde ved læsning som forventet. Yderligere er designet for brugergrænsefladen, og navigationen i den blevet som forventet. Dog blev designet ikke implementeret, da dette vil gøres i en senere iteration.

Dokumentation for SPI forbindelse imellem Devkit8000 og PSoC

Indledning

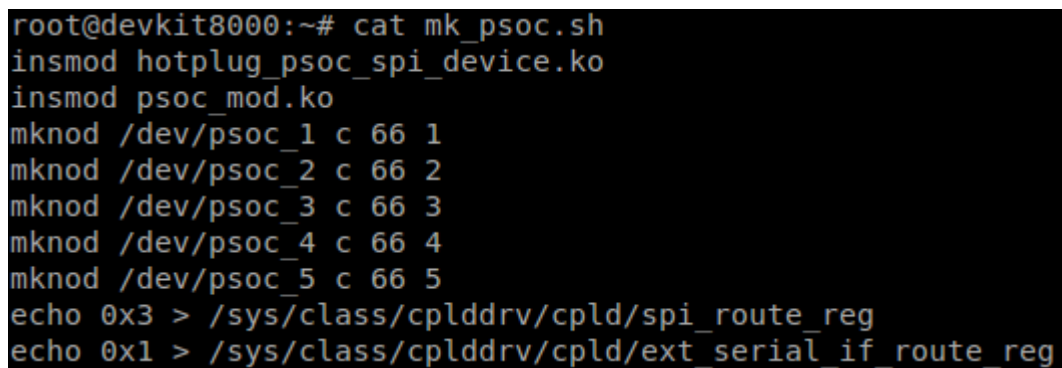
I wineprep projektet skal der bruges en linux platform, som i dette projekt består af et devkit8000 hvorpå der er installeret distributionen Ångström. Denne skal tage imod input fra brugeren af systemet via en grafisk brugergrænseflade og omskrive disse til kommandoer, som sendes til PSoC. Desuden vil PSoC sende status beskeder tilbage til Devkittet. Der vil i projektet indgå en PSoC, som står for kommunikationen med Devkit8000 samt for nogle af systemets motorer og sensorer. Men på grund af et begrænset antal GPIO pinde på PSoC, vil det blive nødvendigt med to ekstra PSoC, som kan tage sig af kommunikationen med de øvrige motorer og sensorer i systemet. Der vil derfor blive lavet en forbindelse mellem de tre PSoC enheder. Der vil i denne dokumentation blive refereret til disse PSoC enheder som PSoC1 (forbindes til Devkit), PSoC2(forbindes til PSoC1) og PSoC3(forbindes til PSoC1). Ovenstående taget i betragtning vil det blive nødvendigt at etableres en tovejsforbindelse imellem devkit8000 og PSoC1, samt en tovejsforbindelse imellem PSoC1 og PSoC2, samt PSoC1 og PSoC3.

Dette kan gøres på flere forskellige måder, f.eks. med en UART protokol som på tidligere semester projekter. Der er i fagene HAL og GFV på 3. semester blevet arbejdet en del med de to serielle dataforbindelses standarder I2C og SPI, og det vil derfor være nærliggende at benytte en af disse to. Det er fra gruppens side, og med opfordring fra vejleder blevet besluttet at SPI vil blive benyttet i dette projekt til forbindelsen imellem Devkit og PSoC. Grunden til dette valg er mest af alt bekvæmmelighed, da der allerede er arbejdet med SPI drivere til Linux og PSoC i tidligere laboratorie øvelser.

Design og implementering

SPI er en synkron dataopførselsmetode, hvor to enheder indgår i et master-slave forhold, og hvor data sendes og modtages mellem de to enheder på samme tid med klokken som "taktstok". For mere information omkring SPI protokollen henvises til bilag(*referance til SPI protokollen). Devkit8000 vil i dette tilfælde udfører rollen som master i forbindelsen til PSoC1, hvilket også betyder at det er denne som står for at starte dataoverførelsen mellem de to enheder. Derudover vil PSoC1 fungere som master i forbindelsen til PSoC2 og PSoC3.

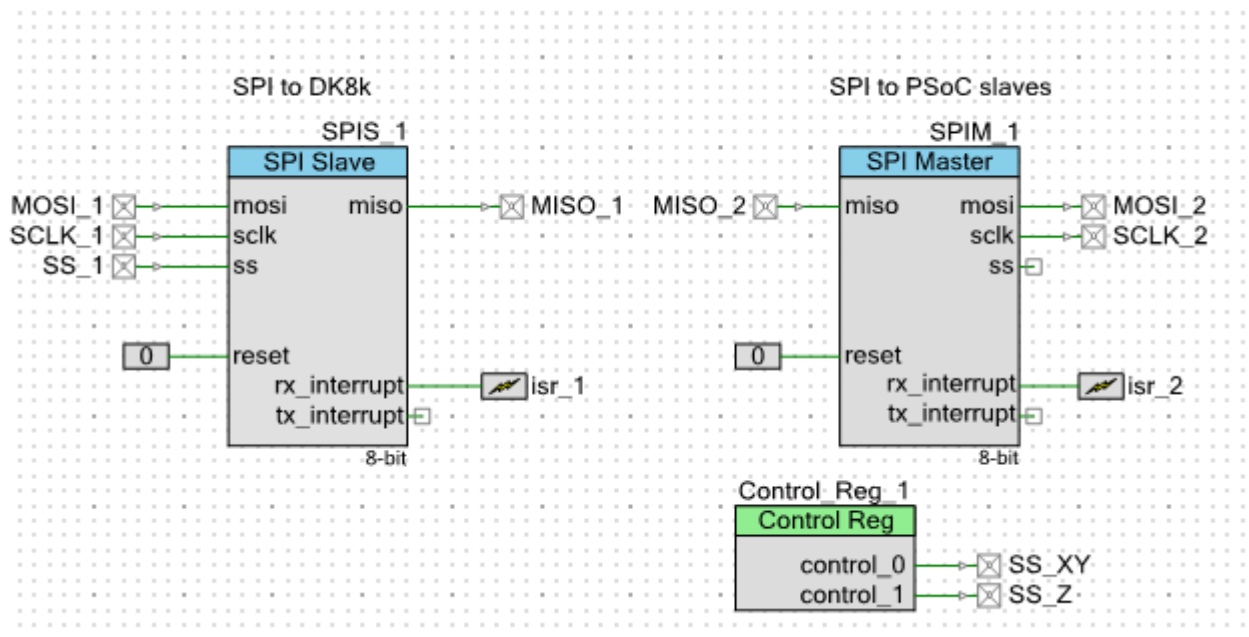
SPI undersøttes naturligvis af Devkit8000, men for at kunne etablere forbindelsen til PSoC1 skal den nødvendige SPI driver skrives til Linux. Denne driver indeholder blandt andet den korrekte opsætningen for forbindelsen, samt implementeringen af SPI metoderne til at sende/modtage data. Der skal også skrives en character device driver, der gør det muligt for et program i userspace at tilgå SPI driveren. Der laves også en hotplug driver der gør det muligt at indsætte PSoC i runtime. For mere information omkring implementeringen af denne driver, samt eksempel på dens anvendelse henvises der til HAL øvelse 6 i bilag. Vi har i dette projekt anvendt den SPI driver som er blevet udleveret på redmine i faget HAL. Så SPI driver modulerne indsættes blot på devkit8000, og der laves de nødvendige device noder. Disse noder er nødvendige for at kunne kommunikere melle user- og kernelspace.

A terminal window with a black background and red text. The prompt is 'root@devkit8000:~#'. The commands entered are: 'cat mk_psoc.sh', 'insmod hotplug_psoc_spi_device.ko', 'insmod psoc_mod.ko', 'mknod /dev/psoc_1 c 66 1', 'mknod /dev/psoc_2 c 66 2', 'mknod /dev/psoc_3 c 66 3', 'mknod /dev/psoc_4 c 66 4', 'mknod /dev/psoc_5 c 66 5', 'echo 0x3 > /sys/class/cplddrv/cpld/spi_route_reg', and 'echo 0x1 > /sys/class/cplddrv/cpld/ext_serial_if_route_reg'.

```
root@devkit8000:~# cat mk_psoc.sh
insmod hotplug_psoc_spi_device.ko
insmod psoc_mod.ko
mknod /dev/psoc_1 c 66 1
mknod /dev/psoc_2 c 66 2
mknod /dev/psoc_3 c 66 3
mknod /dev/psoc_4 c 66 4
mknod /dev/psoc_5 c 66 5
echo 0x3 > /sys/class/cplddrv/cpld/spi_route_reg
echo 0x1 > /sys/class/cplddrv/cpld/ext_serial_if_route_reg
```

Figur 0.12: Indsættelse af moduler i Linux kernen og oprettelse af device noder

De tre PSoC enheder skal ligesom Devkit også have implementeret software, der kan håndtere SPI forbindelsen, og reagere på den ønskede måde når der modtages data. Denne software skrives med værktøjet PSoC-creator, som via et simpelt drag-and-drop interface, gør det nemt at konfigurere SPI. Selve SPI driveren skal derfor ikke skrives, men der skal laves en source fil hvori det er muligt at kalde metoder til styring af SPI, håndtere RX interrupts og anvende den modtagne data til at udføre forskellige opgaver for systemet.



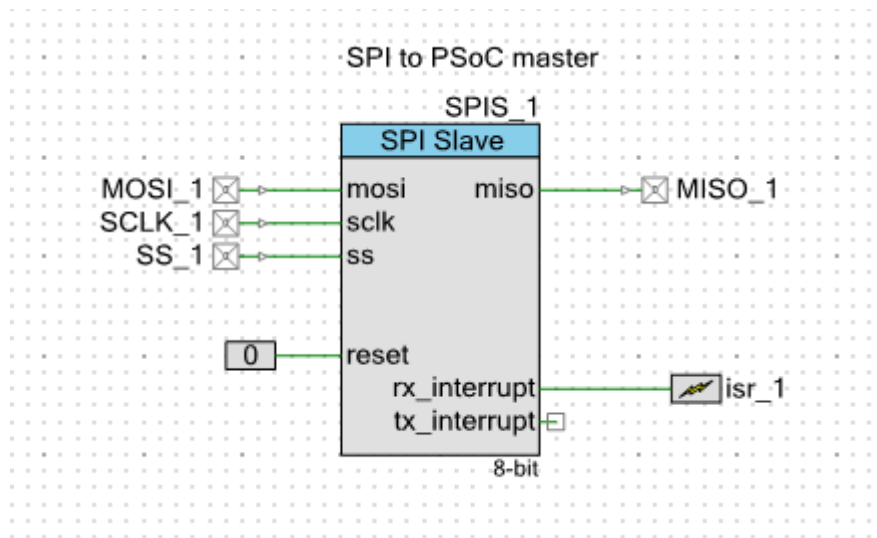
Figur 0.13: Topdesign for PSoC1

PSoC1 skal have både et SPI-master(SPIM) og et SPI-slave(SPIS) modul i PSoC-creator. Der ønskes et interrupt ved RX på SPIS, da slaven skal reagere hver gang der modtages data fra master, hvorefter der skal skrives tilbage, og udføres en opgave. Dette vil blive håndteret i main af en switch implementation, der kaldes hver gang RX interruptet flaget går højt. SPI driveren på Devit8000 er opsat således at der sendes 16 bits til PSoC1. Af disse er de første 8 bit tiltænkt adresse- og kommandobits, og de næste 8 bit er selve data. Der vil i main først blive switched på adresse bits, disse fortæller hvilken overordnet case der benyttes. Kommandobits indikerer om der ønskes data læst tilbage fra PSoC1, eller om det blot er tilstækkeligt at PSoC1 behandler den modtagne data, og sender nogle 0 værdier tilbage. Der vil vi inde i switchen være mulighed for at PSoC1 giver data videre til PSoC2 og PSoC3 i tilfælde, hvor den ikke selv kan udføre den pågældende opgave. Her vil PSoC1 bruge metoder fra SPIM, og fungere som master i forhold til PSoC2 og PSoC3.

```
switch (addr) {  
    case 0x1:  
        switch(cmd) {  
  
            // Write status of bottle to DK8k  
            case GET_STATUS:  
                updateStatus(status);  
  
                if(status != IN_PROCESS || status != VALID_TYPE) {  
                    status = DEFAULT;  
                }  
                break;  
  
            // Registerate bottle  
            case LOCATE_XY:  
                updateStatus(IN_PROCESS);  
  
            // Validate thickness of bottle neck  
            writeToXY(cmd);  
            break;  
  
            // Open bottle  
            case OPEN_BOTTLE:  
                updateStatus(IN_PROCESS);  
        }  
    }  
}
```

Figur 0.14: Eksempel på en switch statement i PSoC koden

Topdesign og koden for PSoC2 og PSoC3 vil blive implementeret næsten på samme måde som PSoC1, hvor der implenteres et RX interrupt og en switch som behandler den modtagne data. De to slave PSoC enheder vil dog kun have et SPIS modul implementeret, og derfor have færre switch cases. Der bruges en 8bit kommando til at sende data til PSoC2 og PSoC3, det er denne kommando, som bliver switched på.



Figur 0.15: Topdesing for PSoC2 og PSoC3

```

CY_ISR(isr_1)
{
    uint8 cmd;

    cmd = SPIS_1_ReadRxData(); // This also clears Rx Status Register

    switch(cmd) {

        // Write status of bottle to master-PSoC
        case GET_STATUS:
            updateStatus(status);
            break;

        // Position x-sensor to just under top of bottle
        case LOCATE_Z_1:
            enabled = POSITION_Z_1;
            break;

        // Position x-sensor to just over top of bottle
        case LOCATE_Z_2:
            enabled = POSITION_Z_2;
            break;
    }
}

```

Figur 0.16: Eksempel på switch cases for PSoC2 og PSoC3

Test

(

Test af SPI forbindelse imellem Devit8000 og PSoC1) Testen for SPI forbindelsen imellem Devkit8000 og PSoC1 foretages med analog discovery, hvor der måles på SS, CLCK, og henholdsvis MISO og MOSI.



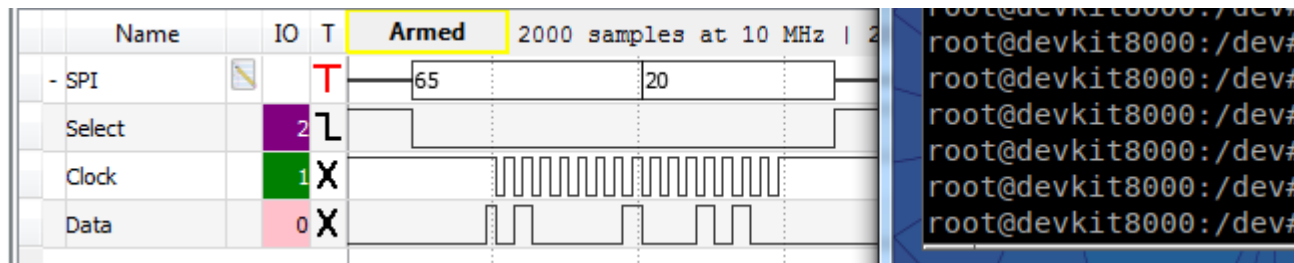
Der udføres to test scenarier. I første test sendes data fra Devkit8000(master) og der foretages en måling på MOSI forbindelsen med logic analyzer funktionen på analog discovery. Denne test skal sikre at vi får sendt de korrekte adresse og data bits over til PSoC1, samt at SS og CLCK opfører sig som ønsket. Det vil sige at SS går lav ved dataoverførsel og at CLCK er stabil. I denne test bruges linux terminalen på devkit8000 til at sende nogle forskellige værdier til PSoC1 med linux kommandoen echo, hvorefter der aflæses bit kombinationer på logic analyzer.

I anden test læses der fortsat på MOSI forbindelsen, og linux kommandoen cat bruges til at læse fra PSoC1. Her aflæses det på terminalen hvad der bliver sendt fra PSoC1. I test programmet på PSoC1 er der implementeret en switch, som gør at når der læses med kommandoen cat PSoC_5 fra devkit terminalen, vil status på knap 2.1 på PSoC1 blive aflæst. Således kan det testes at der sendes den korrekte data fra MISO.

Resultater

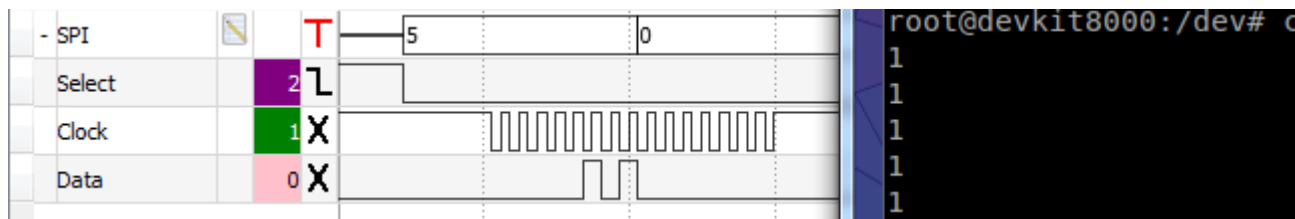
(

Resultater af test for Devkit-PSoC SPI) Ved første test scenarie ses hvordan der sendes værdien 20 fra devkit til PSoC1 med linux kommandoen echo. På logic analyzer ses at der sendes to gange 8bit. først en adresse kode, som er 65, og derefter værdien 20. Det ses også at SS går lav ved dataoverførelse og at clock er stabil. Testen er derfor tilfredsstillende.



Figur 0.18: Data bliver sendt fra Devkit8000 til PSoC1

Ved anden test scenarie bruges linux kommandoen cat til at læse fra PSoC1. Her ses at der sendes en adresse bit 5 og 0 via MOSI, og på terminalen ses status for knappen på PSoC1, som er trykket nede og derfor viser 1.



Figur 0.19: Data bliver læst fra PSoC1

Diskussion

Ud fra ovenstående ses det at SPI kan bruges til at kommunikere imellem devkit8000 og PSoC1, samt at kommunikere imellem PSoC enhederne. Som resultaterne af testen viser er SPI en smart måde hvorpå der kan sendes og modtages data samtidigt. Det skal dog nævnes at der har været mange problemer med denne kommunikation, og der er brugt adskillige timer på at få det til at virke. Så sammenlignet med UART som fungerede problemfrit på 1 og 2 semester, så har SPI været meget mere udfordrende at få til at virke.