Øvelse 7 - Linux Device Model and Bus Interface

Hardware

Det device, som vi ønsker at lave en SPI device driver til, er et OLED 16-bit Color 0.96" display. Dette device implementeres med MOSI, da der skal skrives fra Rpi til display, og det er derfor kun write, der implementeres i denne opgave.

Implementering af Device Tree Overlay

Når vi skal implementere .dto filen, skal vi først finde bus nummeret for vores SPI kommunikation. På Raspberry Pi Zero W diagrammet kan vi se, at der er tilknyttet SPI til følgende pins: GPIO7 - GPIO11. Vi kan derefter slå op i BCM2835 Reference Manual side 102 og se, at disse pins alternative funktioner (ALT0) er tilnyttet SPI bus 0.

6.2 Alternative Function Assignments

Every GPIO pin can carry an alternate function. Up to 6 alternate function are available but not every pin has that many alternate functions. The table below gives a quick over view.

	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	<reserved>			
GPIO1	High	SCL0	SA4	<reserved>			
GPIO2	High	SDA1	SA3	<reserved>			
GPIO3	High	SCL1	SA2	<reserved>			
GPIO4	High	GPCLK0	SA1	<reserved>			ARM_TDI
GPIO5	High	GPCLK1	SA0	<reserved>			ARM_TDO
GPIO6	High	GPCLK2	SOE_N / SE	<reserved>			ARM_RTCK
GPIO7	High	SPI0_CE1_N	SWE_N / SRW_N	<reserved>			
GPIO8	High	SPI0_CE0_N	SD0	<reserved>			
GPIO9	Low	SPI0_MISO	SD1	<reserved>			
GPIO10	Low	SPI0_MOSI	SD2	<reserved>			
GPIO11	Low	SPI0_SCLK	SD3	<reserved>			
GPIO12	Low	PWM0	SD4	<reserved>			ARM_TMS
GPIO13	Low	PWM1	SD5	<reserved>			ARM_TCK

I databladet for driveren ssd1331 kan vi se nedenstående information omkring den serielle kommunikation. Her fremgår det at CPOL skal sættes til 1, da SCLK starter high, og CPHA sættes til 1, da der samples fra low til high.

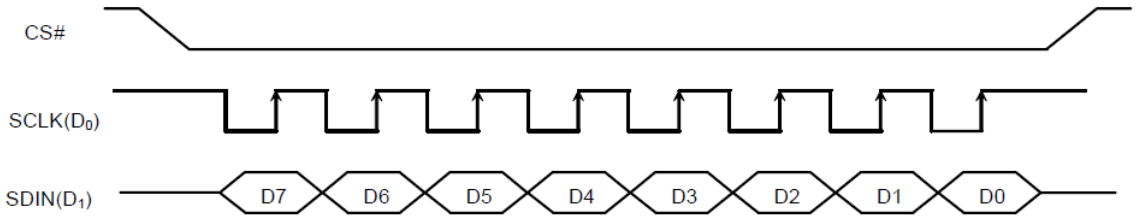


Figure 39 - Serial interface characteristics

Da vi ønsker at indstille kommunikationen til SPI mode 3, sættes CPOL og CPHA flaget ved at angive begge i vores device three overlay fil under spi_drv@0. reg sættes til 0, hvilket refererer til index 0 på en liste af CS (Chip select). Til sidst tilføjes den maksimale frekvens på 6,67 MHz. Frekvensen beregnes ved at aflæse minimum clock cycle time i databladet til 150 ns. Ud fra denne tid kan vi beregne frekvensen til 1/150 ns = 6,67 MHz.

```
/dts-v1/;
/plugin/;
/ {
    compatible = "brcm,bcm2835", "brcm,bcm2708";
    /* disable spi-dev for spi0.0 */
    fragment@0 {
        target = <&spi0>; // SPI Bus 0
        __overlay__ {
            status = "okay";
            spidev@0{     // SPI Chip Select 0
```

```
                status = "disabled";
            };
        };
    };
    fragment@1 {
        target = <&spi0>; // SPI Bus 0
        __overlay__ {
            /* needed to avoid dtc warning */
            #address-cells = <1>;
            #size-cells = <0>;
            spi_drv:spi_drv@0 {
                compatible = "ase, spi_drv";
                reg = <0>; // SPI Chip Select 0
                spi-cpha; /* Comment in to set CPHA */
                spi-cpol; /* Comment in to set CPOL */
                spi-max-frequency = <6600000>;
            };
        };
    };
};
```

Implementering af Device Driver

Probe/remove

I probe opretter vi de devices, der ønskes. Vi skal bruge et SPI device og de to gpio'er, som skal kommunikere med D/C og Reset på vores display. SPI får minornummer 0, D/C får minornummer 1 og Reset får minornummer 2. D/C bruges til at skelne mellem data eller kommando overførsel.
Vi laver fejlhåndtering i slutningen af probe, som vi kender det fra tidligere opgaver.

```
static int spi_drv_probe(struct spi_device *sdev)
{
      int err = 0;
      struct device *spi_drv_device;

      printk(KERN_DEBUG "New SPI device: %s using chip select: %i\n", sdev->modalias, sdev->

      /* Check we are not creating more
     devices than we have space for */
      if (spi_devs_cnt > spi_devs_len) {
        printk(KERN_ERR "Too many SPI devices for driver\n");
        return -ENODEV;
      }

      /* Configure bits_per_word, always 8-bit for RPI!!! */
      sdev->bits_per_word = 8;
      spi_setup(sdev);

      spi_devs.spi = sdev;

      /* We map spi_devs index to minor number here */
      spi_drv_device = device_create(spi_drv_class, NULL,
                                MKDEV(MAJOR(devno), spi_devs_cnt),
                                NULL, "spi_drv%d", spi_devs_cnt);
      if (IS_ERR(spi_drv_device))
    {
        printk(KERN_ALERT "FAILED TO CREATE DEVICE\n");
        goto Error;
    }

      else
        printk(KERN_ALERT "Using spi_devs%i on major:%i, minor:%i\n",
           spi_devs_cnt, MAJOR(devno), spi_devs_cnt);

      /* Update local array of SPI devices */
      spi_devs.channel = 0x00; // channel address

    ++spi_devs_cnt;
```

```
    spi_devs.dc_gpio = 13;
    spi_devs.rst_gpio = 19;

      err = gpio_request(spi_devs.dc_gpio, "gpio_dc");
      if(err) {
        pr_err("Failed gpio request for dc_gpio\n");
        goto Error1;
      }
      gpio_direction_output(spi_devs.dc_gpio, 0);

      spi_drv_device = device_create(spi_drv_class, NULL, MKDEV(MAJOR(devno), spi_devs_cnt),
      if(IS_ERR(spi_drv_device)) {
        pr_err("Failed gpio device request for dc_gpio\n");
        goto Error2;
      }

    ++spi_devs_cnt;

      err = gpio_request(spi_devs.rst_gpio, "gpio_rst");
      if(err) {
        pr_err("Failed gpio request for rst_gpio\n");
        goto Error3;
      }
      gpio_direction_output(spi_devs.rst_gpio, 0); // GPIO direction;

      spi_drv_device = device_create(spi_drv_class, NULL, MKDEV(MAJOR(devno), spi_devs_cnt),
    if(IS_ERR(spi_drv_device)){
          pr_err("Failed gpio device request for rst_gpio\n");
          goto Error4;
    }

      return 0;

      Error4: gpio_free(spi_devs.rst_gpio);
      Error3: device_destroy(spi_drv_class, MKDEV(MAJOR(devno), --spi_devs_cnt));
      Error2: gpio_free(spi_devs.dc_gpio);
    Error1: device_destroy(spi_drv_class, MKDEV(MAJOR(devno), --spi_devs_cnt));
    Error:

    return err;
}
```

I remove rydder vi op efter de devices, der er oprettet ved at køre hhv. device_destroy og gpio_free.

```
static int spi_drv_remove(struct spi_device *sdev)
{
     int its_minor = 0;

      printk (KERN_ALERT "Removing spi device\n");

      /* Destroy devices created in probe() */
      device_destroy(spi_drv_class, MKDEV(MAJOR(devno), its_minor));
    device_destroy(spi_drv_class, MKDEV(MAJOR(devno), ++its_minor));
    device_destroy(spi_drv_class, MKDEV(MAJOR(devno), ++its_minor));
    gpio_free(spi_devs.dc_gpio);
    gpio_free(spi_devs.rst_gpio);

      return 0;
}
```

Write

For at kunne anvende vores spi driver til MOSI skal der skrives en write funktion. Write funktionen virker i hovedtræk som tidligere write funktioner ved at der indlæses en buffer fra user space, som konverteres til en int. Denne int bruges derefter til noget alt efter, hvilket minornummer, der er tale om. Er der tale om minornummer 0, skal der sendes gennem spi, og ellers skal der skrives til en af de to gpio'er.

```
ssize_t spi_drv_write(struct file *filep, const char __user *ubuf,
                      size_t count, loff_t *f_pos)
{
    int minor, len, value;

      minor = iminor(filep->f_inode);

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

    char kbuf[MAX_LEN];
        len = count < MAX_LEN ? count : MAX_LEN;
          if(copy_from_user(kbuf, ubuf, len))
        return -EFAULT;

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

          if(MODULE_DEBUG)
            printk("string from user: %s\n", kbuf);

          /* Convert string to int */
        sscanf(kbuf,"%i", &value);

        if(MODULE_DEBUG)
            printk("value %i\n",value);

    if(minor == 0) {
        spi_write_message(spi_devs.spi, value); // send spi message
    }

    else {
        switch(minor) {
            case 1:
            gpio_set_value(spi_devs.dc_gpio, value); // set gpio-dc
            break;
            case 2:
            gpio_set_value(spi_devs.rst_gpio, value); // set gpio_reset
            break;
            default:
            break;
        }
    }

      /* 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;
}
```

Vi anvender en hjælpefunktion til at håndtere SPI write. Ved denne hjælpefunktion sendes der 8 bit ad gangen med SPI.

```
int spi_write_message(struct spi_device *spi, uint8_t data)
 {
    int err;
    struct spi_transfer t[1]; /* Only one transfer */
    struct spi_message m;

    memset(t, 0, sizeof(t)); /* Init Memory */
    spi_message_init(&m); /* Init Msg */
    m.spi = spi; /* Use current SPI I/F */

    t[0].tx_buf = &data; /* Transmit data */
    t[0].rx_buf = NULL; /* Recieve No data */
```

```
    t[0].len = 1; /* Transfer Size in Bytes */
    spi_message_add_tail(&t[0], &m); /* Add Msg to queue */
    err = spi_sync(m.spi, &m); /* Blocking Transmit */

    if(MODULE_DEBUG)
        printk("string from user sent through spi. Value %d\n", data);

    if(MODULE_DEBUG)
        printk("err value: %d\n", err);

    return err;
}
```

Test

For at sikre at device tree overlay loades automatisk ved boot, så starter vi med at kopiere vores .ko fil til /lib/modules/5.4.83:

```
root@raspberrypi0-wifi:~# cd /lib/modules/5.4.83
root@raspberrypi0-wifi:/lib/modules/5.4.83# ls
kernel                  modules.builtin.modinfo  modules.softdep
modules.alias           modules.dep              modules.symbols
modules.alias.bin       modules.dep.bin          modules.symbols.bin
modules.builtin         modules.devname          spi_drv.ko
modules.builtin.bin     modules.order
```

Modulet registreres med modprobe -a og der laves en entry i /etc/module_load.d:

```
root@raspberrypi0-wifi:~# echo spi_drv > /etc/modules-load.d/spi_drv.conf
root@raspberrypi0-wifi:~# cd /etc/modules-load.d
root@raspberrypi0-wifi:/etc/modules-load.d# ls
i2c_dev.conf  spi_drv.conf
```

Overlay filen kopieres til /boot/overlays/:

```
root@raspberrypi0-wifi:~# cp spi_drv.dtbo /boot/overlays
root@raspberrypi0-wifi:~# cd /boot/overlays
root@raspberrypi0-wifi:/boot/overlays# ls
at86rf233.dtbo          hifiberry-digi.dtbo      pitft35-resistive.dtbo
disable-bt.dtbo         i2c-rtc.dtbo             pps-gpio.dtbo
dwc-otg.dtbo            iqaudio-dac.dtbo         rpi-ft5406.dtbo
dwc2.dtbo               iqaudio-dacplus.dtbo     rpi-poe.dtbo
gpio-ir-tx.dtbo         mcp2515-can0.dtbo        sdio.dtbo
gpio-ir.dtbo            mcp2515-can1.dtbo        spi_drv.dtbo
gpio-key.dtbo           miniuart-bt.dtbo         vc4-fkms-v3d.dtbo
hifiberry-amp.dtbo      pitft22.dtbo             vc4-kms-v3d.dtbo
hifiberry-dac.dtbo      pitft28-capacitive.dtbo  w1-gpio-pullup.dtbo
hifiberry-dacplus.dtbo  pitft28-resistive.dtbo   w1-gpio.dtbo
```

Til sidst tilføjes en entry til config filen, så den loades under boot:

```
## Angiv overlay
dtoverlay=plat_drv
dtoverlay=spi_drv
```

For at teste om spi driveren virker som ønsket, loader vi først modulet:

```
insmod spi_drv.ko
```

Vi kører kommandoen dmesg og kan se, at modulet er loadet som ønsket. Dette bekræftes ved, at vi kan se, at de ønskede noder er oprettet i /dev, nemlig gpio_dc, gpio_rst og spi_drv0.

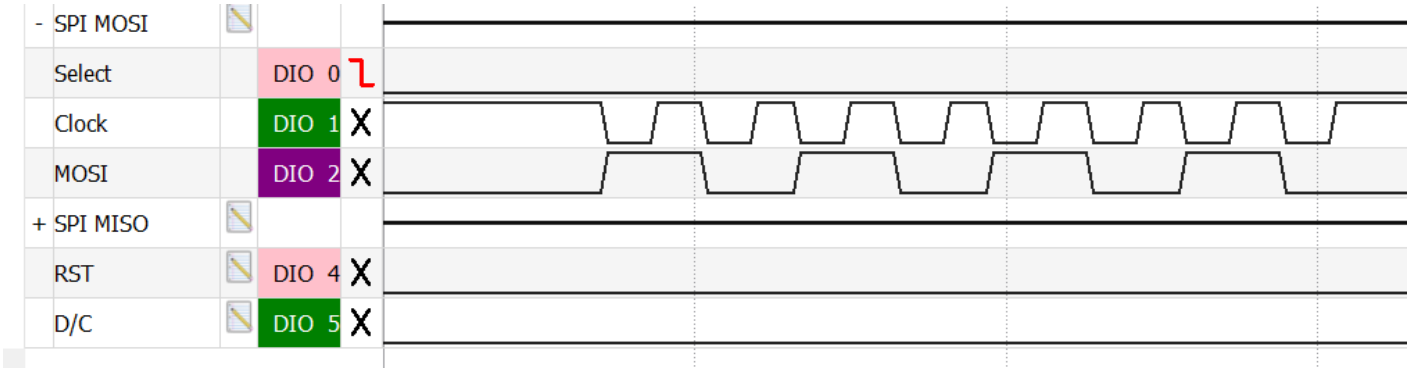
```
 81.527604] spi_drv driver initializing
 81.536469] Assigned major no: 238
 81.551761] New SPI device:  spi_drv using chip select: 0
 81.552145] Using spi_devs0 on major:238, minor:0
```

```
root@raspberrypi0-wifi:~# ls /dev
autofs           fuse             loop1          mqueue         ram2           snd
block            gpio_dc          loop2          net            ram3           spi_drv0
btrfs-control    gpio_rst         loop3          null           ram4           stderr
bus              gpiochip0        loop4          ppp            ram5           stdin
cachefiles       gpiomem          loop5          ptmx           ram6           stdout
char             hwrng            loop6          pts            ram7           tty
console          i2c-1            loop7          ram0           ram8           tty0
cpu_dma_latency  i2c-2            mapper         ram1           ram9           tty1
cuse             initctl          media0         ram10          random         tty10
disk             input            media1         ram11          raw            tty11
dma_heap         kmsg             mem            ram12          rfkill         tty12
dri              log              mmcblk0        ram13          serial0        tty13
fd               loop-control     mmcblk0p1      ram14          serial1        tty14
full             loop0            mmcblk0p2      ram15          shm            tty15
```

Når vi har konstateret at probe() funktionen er implementeret og virker som ønsket, tester vi spi kommunikation ved at måle signalerne med Analog Discovery. Vi kører en tilfældig kommando:

```
echo 0xAA > /dev/spi_drv0
```

Vi kan se på vores måling, at data sendes som ønsket, da det stemmer overens med billedet fra databladet, som vi har set på tidligere.



Som det næste kører vi kommandoen, som tænder skærmen:

```
echo 0xAF > /dev/spi_drv0
```

Vi kan se, at skærmen tænder.