# Коришћење I²C магистрале

## ЦИЉ

Оспособити функционалност I²C магистрале и искористити је за комуникацију са Nunchuk уређајем.

## ИСХОД

Након ове вежбе ћете моћи да:

- Дефинишете подешавања pinmux-а
- Приступите регистрима I²C уређаја кроз магистралу.

## ПОСТАВКА

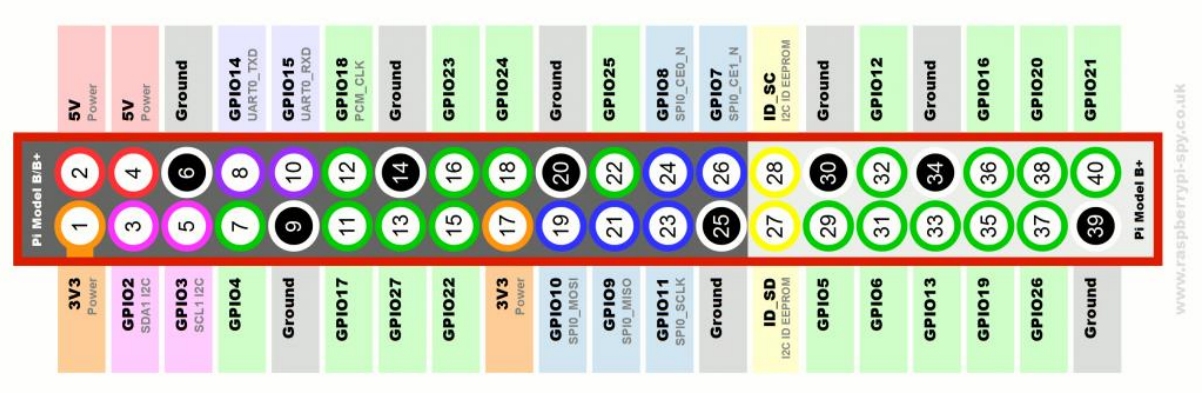
Позиционирајте први терминал на путању `~/linux-kernel-labs/src/linux` ради превођења језгра и DTB (останите на тренутном (`lab10`) branch-у). Позиционирајте други терминал на путању `~/linux-kernel-labs/modules/nfsroot/nunchuk` ради превођења модула. Не заборавите да у оба терминала поставите променљиве `ARCH` и `CROSS_COMPILE` на одговарајуће вредности.

## ПРОНАЛАЖЕЊЕ ИНФОРМАЦИЈА О КОНФИГУРАЦИЈИ МУЛТИПЛЕКСИРАЊА ПИНОВА ЗА I2C1

Као што смо видели у претходној вежби, добили смо nunchuk уређај пријављен на i2c1 магистралу.

Међутим, да бисмо приступили сигналима за податке и такт, морамо подесити и мултиплексирање пинова на SoC-у.

Ако погледате слику испод, видећете да су пинови 3 и 5 (`SDA1` и `SCL1`) конектора `J8` заправо пинови `GPIO2` и `GPIO3`, односно `BCM2` и `BCM3` (http://pinout.xyz/) на BCM2836 (или BCM2709) SoC-у. У документу BCM2835 ARM Peripherals у поглављу 6 General Purpose I/O (GPIO) ћете за пинове `GPIO2` и `GPIO3` видети да морају имати одабрану алтернативну функцију `ALT0` да би имали улогу пинова `SDA1` и `SCL1` периферије `BSC1` периферије. То је довољно информација неопходних да урадите подешавање pinmux-а закоришћење `BSC1` периферије (контролера), у нашем случају `i2c1` чвора.




*Слика 1 Распоред пинова на конектору 40-пинском конектору J8 на Raspberry Pi 2 плочи*

У истом документу можете видети да GPIO пинови омогућавају укључивање интерних pull-up и pull-down отпорника, као и предвиђено подешавање за сваки у колони PULL (предвиђено је High односно pull-up за оба пина, мада се могу користити и без интерног pull-up отпорника јер за поменуте пинове већ постоје екстерни, на плочи).

Такође, у истом поглављу можете видети и да се GPIO периферија (која је заправо комбинација pinmux и interrupt контролера) налази на адреси `0x7E200000`, као и информацију о сваком контролном регистру ове периферије (одстојање од базне адресе и улогу коју има са детаљном дефиницијом сваког од 32 бита).

## ДОДАВАЊЕ PINMUX ПОДЕШАВАЊА У DEVICE TREE

Када знамо позиције регистира, покушајмо да схватимо како се оне користе у постојећем коду. Отворите Device Tree за SoC-ове засноване на BCM2708 (arch/arm/boot/dts/bcm283x.dtsi), који посредством bcm2709.dtsi датотеке укључује

и већ поменута bcm2709-rpi-2-b.dts DTS датотека. У датотеци bcm283x.dtsi ћете осим дефиниције `i2c1` чвора као потомка SoC чвора пронаћи и `gpio` чвор (pinmux контролер) са основним подешавањима, укључујући базну адресу ове периферије (`0x7e200000`).

У датотеци bcm2709-rpi-2-b.dts се налазе додатна подешавања ове периферије која су ту измештена јер одговарају само bcm2709 SoC-у, односно Raspberry Pi 2 плочи. За сваку од периферија којој је потребно дефинисати пинове које користи се на овом месту дефинише чвор потомак чвора `gpio`, па тако треба урадити и за `i2c1` периферију (у оригиналној верзији овог документа је то и било дефинисано па можете искористити да одатле копирате или проверите оно што сте урадили).

Касније се референца на овај чвор користи у пољу `pinctrl-0` одговарајуће периферије (`i2c1` у нашем случају).

Могућа подешавања за pinmux контролер односно `gpio` чвор (а која користи сам gpio руковалац) су:

Све опције које се могу користити можете пронаћи у документацији BCM2835 GPIO модула (Documentation/devicetree/bindings/pinctrl/brcm,bcm2835-gpio.txt). За ову вежбу је у поменутом чвору потомку чвора `gpio` довољно подесити опције у наставку.

Подешавања чвора потомка:

- **brcm,pins:** Низ ћелија. Свака ћелија садржи ID пина. Валидни ID су целобројни делови GPIO ID; 0==GPIO0, 1==GPIO1, ... 53==GPIO53.

- **brcm,function:** Цео број који представља функцију мултиплексирања пина/пинова:

  0: GPIO in

  1: GPIO out

  2: alt5

  3: alt4

  **4: alt0**

```
        5: alt1

        6: alt2

        7: alt3
```

- brcm,pull: Цео број који представља примену pull-down/up на pin/pinove:

```
        0: none

        1: down

        2: up
```

На основу горенаведеног, чвор потомак чвора `gpio` који треба додати у датотеку bcm2709-rpi-2-b-custom.dts треба да изгледа овако:

```
&gpio {
        i2c1_pins: i2c1 {
                brcm,pins = <2 3>; /* GPIO2 i GPIO3 */
                brcm,function = <4>; /* alt0 */
        };
};
```

Такође, подешавања пинова сада треба увезати и у чвор `i2c1` који предсатавља нашу периферију, односно дефинисати да `i2c1` користи управо те пинове:

```
        pinctrl-names = "default";
        pinctrl-0 = <&i2c1_pins>;
```

Приликом додавања подешавања у DTS датотеку, не заборавите да подешавања која већ постоје у DTSI датотекама не морате понављати и у DTS (нпр. основна подешавања за gpio чвор смо видели да постоје, додајемо само

подешавања везана за пинове `i2c1` периферије тако што се референцирамо на постојећи чвор `gpio` као `&gpio`). Слично смо радили и за чвор `i2c1`, с обзиром да је и његова основна дефиниција у DTSI датотеци. Дакле додајемо само нове декларације, односно подешавања која треба да замене стара, заједничка за фамилију процесора.

Поново преведите све DTB датотеке, прекопирајте и на крају поново покрените плочу.

## ТЕСТИРАЊЕ I²C МАГИСТРАЛЕ

За тестирање магистрале ћемо користити `i2cdetect` команду да бисмо се уверили да све ради како је очекивано за `i2c1`:

```
# i2cdetect -l
i2c-1     i2c       3f804000.i2c                I2C adapter
```

Уколико не видите поменути адаптер, вероватно модул `i2c_dev` није укључен (можете проверити командом `lsmod`). Модул можете додати командом

```
modprobe i2c_dev
```

Да не бисте сваки пут морали да додајете овај модул, једноставније је да га укључите у кернел:

```
make xconfig
```

Претражите опцију `I2C device interface` (`I2C_CHARDEV`) и укључите је. Поново преведите кернел, прекопирајте `zImage` и поново покрените плочу. Након тога поновите претходни корак како бисте се уверили да је детектован `i2c-1` адаптер.

Ако се и даље не види адаптер, могуће је да сте направили грешку у DTS датотеци.

```
# i2cdetect -F 1
Functionalities implemented by bus #1
I2C                              yes
```

```
SMBus quick command             yes
SMBus send byte                 yes
SMBus receive byte              yes
SMBus write byte                yes
SMBus read byte                 yes
SMBus write word                yes
SMBus read word                 yes
SMBus process call              yes
SMBus block write               yes
SMBus block read                no
SMBus block process call        no
SMBus PEC                       yes
I2C block write                 yes
I2C block read                  yes
```

Уколико приметите да SMBus Quick Commands нису доступне, а обзиром да их `i2cdetect` користи на I²C магистрали, можете помоћу `i2cdetect -r` користити уобичајени скуп I²C команди, и тиме детектовати све уређаје на магистрали.

Уверите се да све ради како је очекивано, покрените `i2cdetect 1`. Ово би требало да скенира све уређје на `i2c1` магистрали. Требало би да видите уређај на адреси `0x52`. То је ваш nunchuk.

## ИНИЦИЈАЛИЗАЦИЈА УРЕЂАЈА

Следећи корак је читање стања nunchuk регистара, да бисмо нпр. проверили да ли је неки тастер притиснут или не.

Пре него што будемо у стању да читамо nunchuk регистре, прва ствар коју морамо урадити је да му пошаљемо иницијализационе команде. То је уједно и згодан начин да се уверимо да I²C комуникација исправно ради.

У функцији `probe()` (покреће се сваки пут када је одговарајуи уређај детектован):

1. Користећи I2C raw API (на слајдовима са предавања), пошаљите уређају 2 бајта: `0xf0` и `0x55`[1]. Уверите се да проверавате повратну вредност функција које користите. Оне би могле да открију евентуалне проблеме у комуникацији. У документацији на Интеренту потражите примере како исправно руковати грешкама користећи исту функцију.
2. Омогућите процесору да сачека 1 ms корстећи функцију `udelay()`. Проверите која заглавља је потребно укључити за ову функцију.
3. На исти начин, пошаљите сада још и бајтове `0xfb` и `0x0`. Овим је завршена иницијализација nunchuk-a.

Поново преведите и учитајте модул. Уверите се да не постоје комуникационе грешке, односно поруке које би биле исписане у случају грешке.

[1] I²C poruke za komunikaciju sa nunchuk-om su u sledećem formatu: <i2c_address> <register> za čitanje, odnosno <i2c_address> <register> <value> za pisanje. Adresa 0x52 se šalje od strane okruženja, tako da je jedino što morate da uradite upis ostalih bajtova, adrese registra i vrednosti koju upisujete (ukoliko radite upis). Postoje dva načina za uspostavu komunikacije. Prvi poznat način je sa enkripcijom upisom 0x00 u registar 0x40 nunchuk-a. Na ovaj način morate dekriptovati svaki bajt koji pročitate iz nunchuk-a (nije komplikovano, ali jeste posao koji zhteva vreme). Na žalost, takva enkripcija ne radi na neoriginalnim nunchuk uređajima, pa morate umesto toga koristiti komunikaciju bez enkripcije, koju postižete upisom 0x55 u registar na adresi 0xf0. Ovo radi sa svim brendovima nunchuka (uključujući originalne Nintendo).

## ЧИТАЊЕ NUNCHUK РЕГИСТАРА

Nunchuk показује једно доста чудно понашање: делује као да се стање интерних регистара освежава само када се догоди читање стања регистара.

Као последица тога, неопходно је да два пута прочитамо регистре којима желимо да проверимо стање.

1. Да би код задржали једноставним и читљивим, направимо функцију `nunchuk_read_registers` која ће једном прочитати регистре. У овој функцији:
2. Почните са паузом од 10 ms позивом функције `mdelay()`. То је трајање неопходне паузе између две узастопне $I^2C$ операције.
3. Упишите `0x00` на магистралу. То ће омогућити читање стања регистара.
4. Додајте још једну паузу од 10 ms.
5. Прочитајте 6 бајтова са уређаја, и даље користећи I2C raw API. Проверите повратне вредности као и до сада.

## ЧИТАЊЕ СТАЊА NUNCHUK ТАСТЕРА

Вратимо се на функцију `probe()`, где вашу нову функцију треба да позовете два пута.

Након другог позива, одредите стање Z и C тастера, које се може пронаћи у шестом бајту који сте прочитали.

Као што је објашњено у документу nunchuck-datasheet.pdf:

- бит 0 == 0 означава да је тастер Z притиснут
- бит 0 == 1 означава да тастер Z НИЈЕ притиснут
- бит 1 == 0 означава да је тастер C притиснут
- бит 1 == 1 означава да тастер C НИЈЕ притиснут.

Користећи се логичким операторима, напишите код који иницијализује `zpressed` целобројну променљиву, чија вредност треба да буде 1 када је тастер Z притиснут, а 0 у супротном. Слично урадите и за променљиву `cpressed` и тастер C.

Исписати одговарајуће поруке на основу стања тастера.

## ТЕСТИРАЊЕ

Преведите ваш модул и учитајте га поново. Требало би да је детектовано да ни један тастер није притиснут. Уклоните модул.

Сада држите притиснут тастер Z и поново учитајте и уклоните модул:

```
insmod /root/nunchuk/nunchuk.ko; rmmod nunchuk
```

Сада би требало да видите поруке које потврђују да је руковалац открио да је тастер Z био притиснут.

Урадите исто више пута са различитим стањем тастера.

У овој фази смо се само уверили да можемо да читамо стање регистара уређаја кроз $I^2C$ магистралу. Наравно, учитавање и уклањање модула сваки пут није згодан начин за приступ таквим подацима.

## САЧУВАЈТЕ СВЕ ИЗМЕНЕ

Да бисте потврдили и сачували све измене, најбоље је да их додате, а потом и локално комитујете на GIT, док сте позиционирани у неки од директоријума репозиторијума који је мењан, нпр. `~/linux-kernel-labs` и исто за `~/linux-kernel-labs/src/linux`:

```
git add -A

git commit -as -m "lab11.1 done"
```

Да би измене постале видљиве и у репозиторијуму на серверу, потребно би још било урадити нпр. `git push`, али то у овом случају није неопходно нити имамо неопходна права за то.