Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Odsek za računarsku tehniku i računarske komunikacije

Linuks u ugrađenim sistemima i razvoj rukovalaca

Linuks uređaj i model rukovaoca
Drugi deo

Sadržaj

- Linuks uređaj i model rukovaoca
  - Komunikacija sa I2C uređajem
  - Uvod u pin muxing

Komunikacija sa I2C uređajem

LINUKS UREĐAJ I MODEL RUKOVAOCA

Komunikacija sa I2C uređajem: API

- Najjednostavniji API za komunikaciju sa I2C uređajem obezbeđuje funkcije za slanje i prihvat podataka:
  - int i2c_master_send(struct i2c_client *client, const char *buf, int count);
    Šalje klijentu sadržaj iz buf.
  - int i2c_master_recv(struct i2c_client *client, char *buf, int count);
    Prihvata broj bajtova od klijenta i smešta ih u buf.

Komunikacija sa I2C uređajem: transfer poruka

- API za transfer poruka dozvoljava opis transfera koji se sastoji iz nekoliko poruka, gde je svaka poruka transakcija u jednom pravcu:
  - int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msg, int num);
  - struct i2c_adapter * pokazivač može da se dobavi korišćenjem client->adapter
  - struct i2c_msg struktura definiše dužinu, lokaciju i smer poruke.

I2C: primer transfera poruka

```
struct i2c_msg msg[2];
int error;
u8 start_reg;
u8 buf[10];

msg[0].addr = client->addr;
msg[0].flags = 0;
msg[0].len = 1;
msg[0].buf = &start_reg;
start_reg = 0x10;

msg[1].addr = client->addr;
msg[1].flags = I2C_M_RD;
msg[1].len = sizeof(buf);
msg[1].buf = buf;

error = i2c_transfer(client->adapter, msg, 2);
```

SMBus pozivi

- SMBus je podgrupa I2C protokola.
- Definiše standardni set transakcija, na primer čitanje ili upis registra u uređaj.
- Linuks obezbeđuje SMBus funkcije koje bi trebale biti korišćene umesto postojećeg API-ja, ukoliko I2C uređaj podržava takav standardni tip transakcija. Tada, rukovalac može biti korišćen od strane SMBus-a i I2C adaptera (ne mogu da se koriste I2C komande na SMBus adapterima).
- Primer: i2c_smbus_read_byte_data() funkcija dozvoljava čitanje jednog bajta podataka iz registra uređaja.
  - Radi sledeće operacije: S Addr Wr [A] Comm [A] S Addr Rd [A] [Data] NA P
  - Što znači da prvo piše komandu dužine jednog bajta (Comm), a nakon toga čita jedan bajt podataka ([Data]).
- Documentation/i2c/smbus-protocol sadrži detaljan prikaz navedenog.

Spisak SMBus funkcija

- Čitanje/Upis jednog bajta
  - s32 i2c_smbus_read_byte(const struct i2c_client *client);
  - s32 i2c_smbus_write_byte(const struct i2c_client *client, u8 value);
- Pisanje komandnog bajta i čitanje ili upis jednog bajta
  - s32 i2c_smbus_read_byte_data(const struct i2c_client *client, u8 command);
  - s32 i2c_smbus_write_byte_data(const struct i2c_client *client, u8 command, u8 value);
- Pisanje komandnog bajta i čitanje ili upis jedne reči (word)
  - s32 i2c_smbus_read_word_data(const struct i2c_client *client, u8 command);
  - s32 i2c_smbus_write_word_data(const struct i2c_client *client, u8 command, u16 value);
- Pisanje komandnog bajta i čitanje ili upis bloka podataka (najviše 32 bajta)
  - s32 i2c_smbus_read_block_data(const struct i2c_client *client, u8 command, u8 *values);
  - s32 i2c_smbus_write_block_data(const struct i2c_client *client, u8 command, u8 length, const u8 *values);
- Pisanje komandnog bajta i čitanje ili upis bloka podataka (bez ograničenja)
  - s32 i2c_smbus_read_i2c_block_data(const struct i2c_client *client, u8 command, u8 length, u8 *values);
  - s32 i2c_smbus_write_i2c_block_data(const struct i2c_client *client, u8 command, u8 length, const u8 *values);

I2C funkcionalnost

- Ne podržavaju svi I2C kontroleri sve funkcionalnosti.
- Stoga, I2C rukovaoci kontrolera obaveste I2C jezgro koje funkcionalnosti podržavaju.
- I2C rukovaoci uređajem moraju da provere da li su funkcionalnosti koje su im potrebne obezbeđene od strane I2C kontrolera koji je u upotrebi na sistemu.
- i2c_check_functionality() funkcija dozvoljava takvu proveru.
- Primeri funkcionalnosti: I2C_FUNC_I2C za mogućnost korišćenja osnovnih I2C funkcija, I2C_FUNC_SMBUS_BYTE_DATA za mogućnost korišćenja SMBus komandi za upis komandi i čitanje/upis jednog bajta podataka.
- Pogledajte include/uapi/linux/i2c.h za potpun spisak postojećih funkcionalnosti.

Reference

- http://en.wikipedia.org/wiki/I2C, uopštena prezentacija I2C protokola
- Documentation/i2c/, detalji o Linuks podršci za I2C
  - writing-clients, kako se pišu I2C rukovaoci uređajem
  - instantiating-devices, kako se instanciraju uređaji
  - smbus-protocol, detalji o SMBus funkcijama
  - functionality, kako mehanizam funkcionalnosti radi
  - I još mnogo drugih dokumenata
- http://free-electrons.com/pub/video/2012/elce/elce-2012-anders-board-bringup-i2c.webm, odlično predavanje: You, me and I2C - David Anders na ELCE 2012.

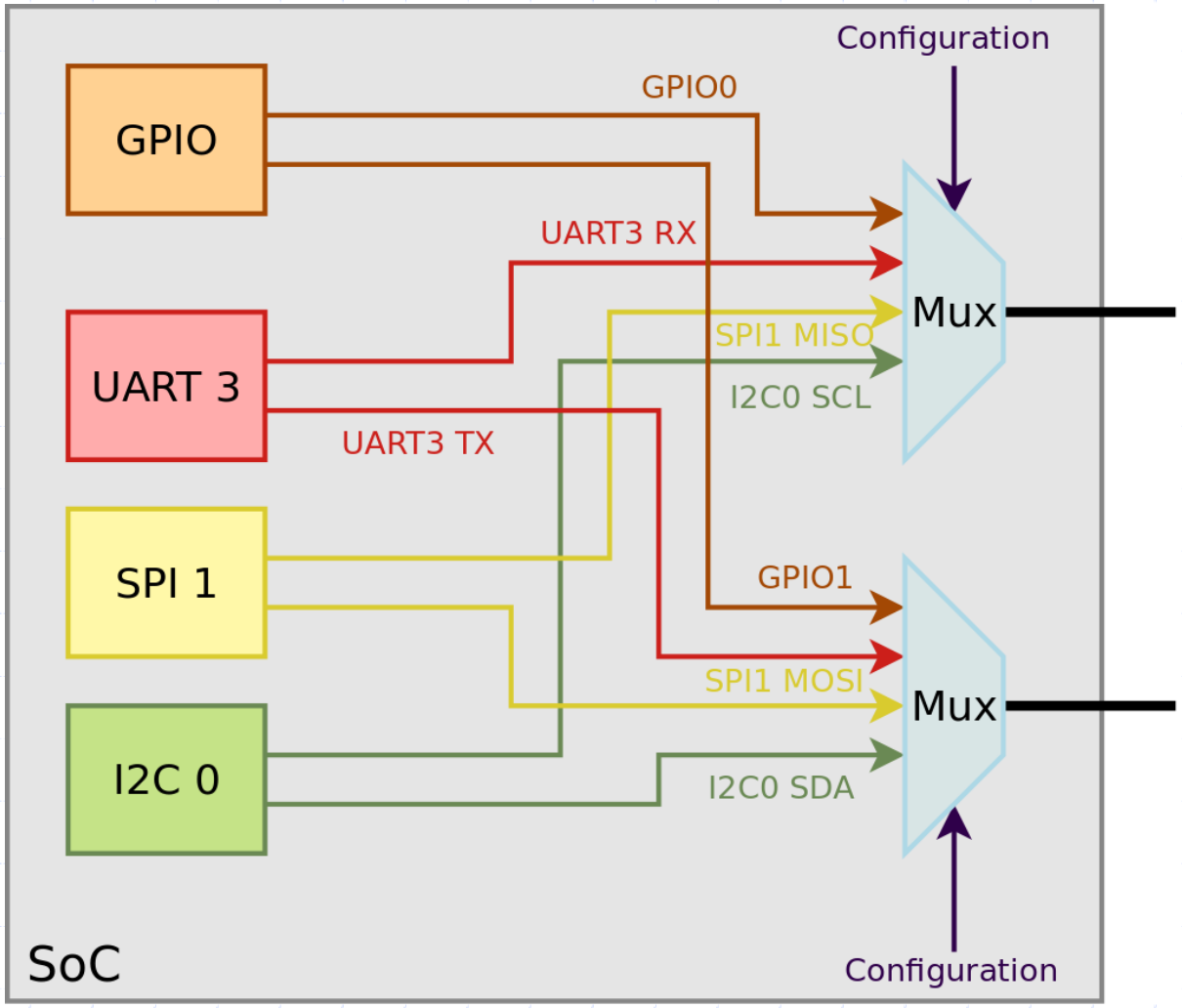
Uvod u pin muxing

LINUKS UREĐAJ I MODEL RUKOVAOCA

Šta je to multipleksiranje pinova?

- Moderni SoC-ovi sadrže sve više i više blokova fizičke arhitekture, od kojih je mnogima potrebna sprega ka spoljnom svetu koju omogućuju pinovi.
- Međutim, fizička veličina ovih čipova ostaje mala, te je broj dostupnih pinova ograničen.
- Zbog ovog razloga nisu sve interne odlike blokova fizičke arhitekture dostupne istovremeno na pinovima.
- Ovi pinovi su multipleksirani: dozvoljavaju ili funkcionalnost bloka fizičke arhitekture A ili funkcionalnost bloka fizičke arhitekture B.
- Multipleksiranje je uglavnom konfigurabilno u programskoj podršci.

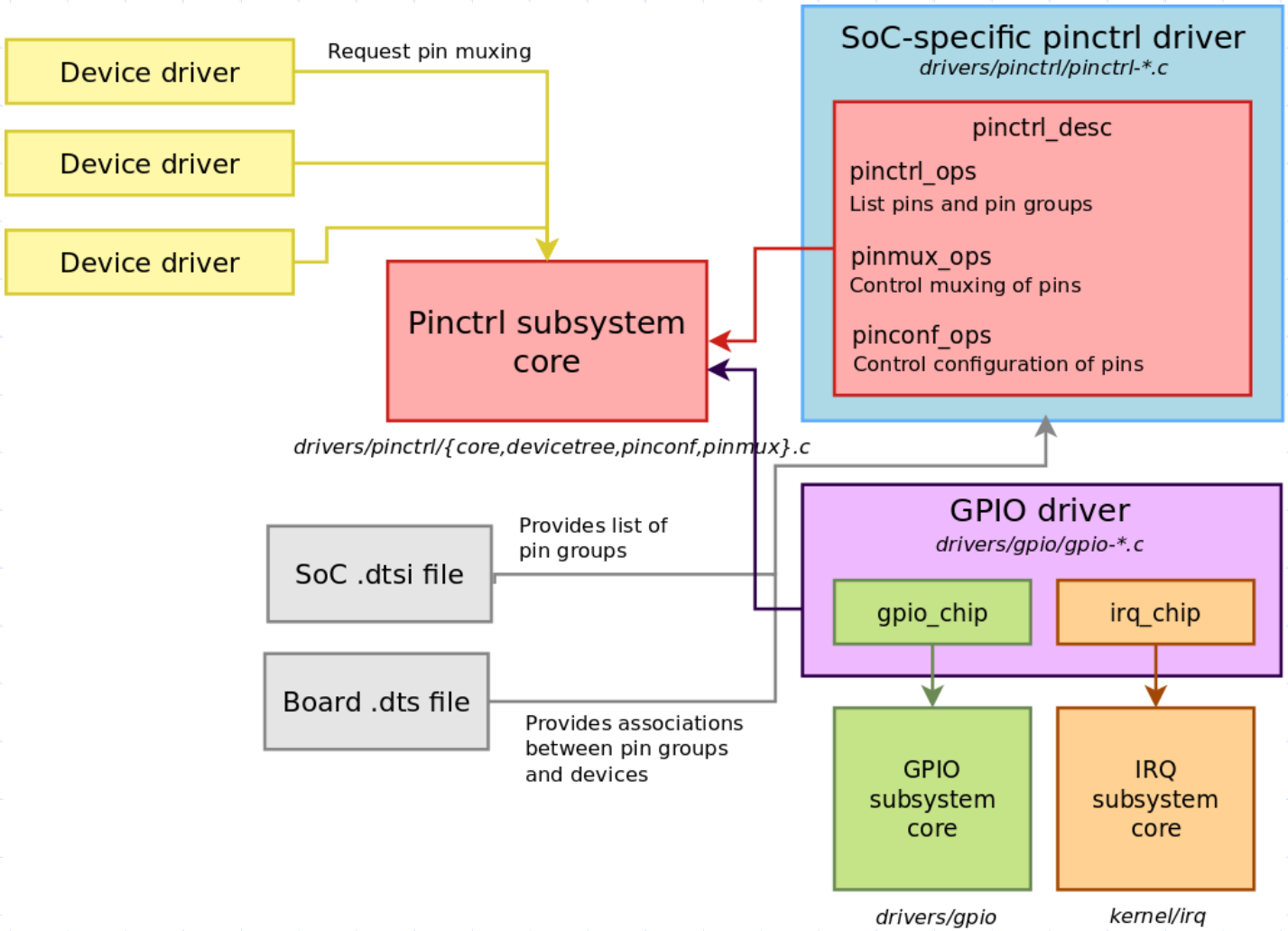
Dijagram multipleksiranja pinova



Multipleksiranje pinova u Linuks kernelu

- U verziji Linuksa 3.2, dodat je pinctrl podsistem.
- Ovaj podsistem, koji se nalazi u drivers/pinctrl obezbeđuje generički podsistem za rukovanje multipleksiranjem pinova. Sadrži:
  - Spregu za multipleksiranje pinova za rukovaoce, za implementaciju u SoC-specifičnim rukovaocima za konfiguraciju multipleksiranja.
  - Spregu za multipleksiranje pinova za potrošače, za rukovaoce uređaja.
- Većina pinctrl rukovalaca obezbeđuje vezu stabla uređaja, gde multipleksiranje pinova mora biti opisano u stablu uređaja.
  - Tačna veza stabla uređaja zavisi od svakog rukovaoca. Svaka zasebna veza je opisana u Documentation/devicetree/bindings/pinctrl.

Dijagram pinctrl podsistema



Veza stabla uređaja za potrošačke uređaje (1/2)

- Uređaji koji zahtevaju da određeni pinovi budu multipleksirani će koristiti odlike stabla uređaja pinctrl-<x> i pinctrl-names.
- Odlike pinctrl-0, pinctrl-1, pinctrl-<x> se vezuju za konfiguraciju pina za dato stanje uređaja.
- Odlika pinctrl-names asocira ime sa pojedinačnim stanjem. Ime default je specijalno i automatski se odabira od strane rukovaoca uređajem, bez potrebe za eksplicitnim pozivom pinctrl funkcije.

Veza stabla uređaja za potrošačke uređaje (2/2)

- U većini slučajeva, dovoljno je sledeće:

```
pmx_twsi0:pinctrl@0x10101010 {
        ...
};
i2c@11000 {
        pinctrl-0 = <&pmx_twsi0>;
        pinctrl-1 = <&pmx_twsi1>;
        pinctrl-names = “default”, “alternativa”;
        ...
};
```

- Pogledajte Documentation/devicetree/bindings/pinctrl/pinctrlbindings.txt za detalje.

Definisanje pinctrl konfiguracija

- Različite pinctrl configurations moraju biti definisane kao čvorovi potomci glavnog pinctrl uređaja (koji kontroliše multipleksiranje pinova).
- Konfiguracije mogu biti definisane na:
  - SoC nivou (.dtsi datoteka), za konfiguraciju pinova koji su često deljeni između više ploča
  - Nivou ploče (.dts datoteka) za konfiguracije koje su specifične za ploču.
- Odlika pinctrl-<x> potrošačkog uređaja pokazuje na konfiguraciju pina koja mu je potrebna kroz phandle stabla uređaja.
- Opis konfiguracija je specifičan za svaki pinctrl rukovalac. Videti u Documentation/devicetree/bindings/pinctrl za više dokumentacije o vezama stabla uređaja.

Primer na OMAP/AM33xx

- Na OMAP/AM33xx, koristi se rukovalac pinctrl-single. Uobičajen je na više SoC-ova i dozvoljava konfigurisanje pinova pisanjem vrednosti u registar.
- U svakoj konfiguraciji pinova, vrednost pinctrl-single,pins daje listu (registar, vrednost) parova potrebnih za konfiguraciju pinova.
- Da bi znali ispravne vrednosti, mora se koristiti dokumentacija SoC-a i ploče (datasheets).

```
am33xx_pinmux: pinmux@44e10800 {
  i2c0_pins: pinmux_i2c0_pins {
    pinctrl-single,pins = <
      /* i2c0_sda.i2c0_sda */
      0x188 (PIN_INPUT_PULLUP | MUX_MODE0)
      /* i2c0_scl.i2c0_scl */
      0x18c (PIN_INPUT_PULLUP | MUX_MODE0)
    >;
  };
};

i2c0: i2c@44e0b000 {
  pinctrl-names = "default";
  pinctrl-0 = <&i2c0_pins>;

  status = "okay";
  clock-frequency = <400000>;

  tps: tps@2d {
    reg = <0x2d>;
  };
};
```

Primer na Allwinner SoC-u

SoC level

arch/arm/boot/dts/sun7i-a20.dtsi

```
/ {
  soc@01c00000 {
    pio: pinctrl@01c20800 {
      compatible = "allwinner,sun7i-a20-pinctrl";
      reg = <0x01c20800 0x400>;
      interrupts = <0 28 1>;

      uart0_pins_a: uart0@0 {
        allwinner,pins = "PB22", "PB23";
        allwinner,function = "uart0";
        allwinner,drive = <0>;
        allwinner,pull = <0>;
      };
      ...
    };
  };
};
```

UART 0 pin mux config

Board level

arch/arm/boot/dts/sun7i-a20-olinuxino-micro.dts

```
/ {
  soc@01c00000 {

    pio: pinctrl@01c20800 {
      led_pins_olinuxino: led_pins@0 {
        allwinner,pins = "PH2";
        allwinner,function = "gpio_out";
        allwinner,drive = <1>;
        allwinner,pull = <0>;
      };
    };

    uart0: serial@01c28000 {
      pinctrl-names = "default";
      pinctrl-0 = <&uart0_pins_a>;
      status = "okay";
    };
  };

  leds {
    compatible = "gpio-leds";
    pinctrl-names = "default";
    pinctrl-0 = <&led_pins_olinuxino>;

    green {
      label = "a20-olinuxino-micro:green:usr";
      gpios = <&pio 7 2 0>;
      default-state = "on";
    };
  };
};
```

LED pin mux config

Enable UART0 and associate pin mux config

Declare LED device and associate pin mux config