Razširitev in podpora serijskih komunikacijskih vmesnikov na Raspberry Pi platformi

Uroš Sadek[1,2], Amor Chowdhury[1,2]
[1]Margento R&D, Gosposvetska cesta 84, 2000 Maribor
[2]Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2000 Maribor, Slovenija
uros.sadek@margento.com, amor.chowdhury@margento.com

Extension and support of serial communication interfaces on Raspberry Pi platform

Raspberry Pi is a series of single-board credit card-sized computers, with first intention of learning of computer science basics. But since Raspberry Pi *compute module* was released, it also finds the way in industrial applications more frequently. Raspberry Pi runs some Linux distributions, which offers support for most of basic peripheral units available on non-industrial models. Industrial model of Raspberry Pi offers some more peripherals, of which software support isn't available yet, in official Linux distributions. The paper presents extension of software support of existing UART and SPI peripherals and also additional support for external UART communication buses, which runs over existing I2C communication bus.

Kratek pregled prispevka

Raspberry Pi je serija mikroračunalnikov v velikosti kreditne kartice, prvotno namenjenih kot učna okolja za učenje osnov računalništva. Z izdajo modela *compute module,* pa se je uporaba Raspberry Pi preselila tudi v industrijska okolja. Omenjeni mikroračunalnik poganja distribucije Linux okolja, ki v namene učnega okolja nudi podporo osnovnih periferij, dostopnih na učnih modelih Raspberry Pi. Za koriščenje polne funkcionalnosti industrijskega modela, je potrebna izdelava dodatne podpore v Linux okolju. V prispevku, je predstavljena razširitev podpore industrijskega *(compute module)* modela mikroračunalnika, za obstoječa UART in SPI komunikacijska vodila, ter razširitev dodatnih zunanjih UART komunikacijskih vodil preko obstoječega I2C komunikacijskega vodila.

1 Uvod

Raspberry Pi (RPi) *compute module*, se pojavlja v čedalje več industrijskih ter tudi komercialnih aplikacijah. Prvi pomemben razlog za prodor RPi v industrijske in komercialne aplikacije je izdaja RPi *compute* modula, drugi pomemben razlog je Linux operacijski sistem, ki ga poganja RPi. Linux operacijski sistem je prav tako prodornejši v industrijsko ter komercialno rabo za to, ker je odprtokodni sistem (open source) in v prvi vrsti prihrani nakup licence. Razen tega, da se ognemo nakupu potrebne licence, prinaša Linux še nekatere druge prednosti kot so varnost, zmogljivost in prilagodljivost sistema. Do izvorne kode sistema ima dostop vsakdo, ter lahko sistem nadgradi ali dogradi za svoje potrebe. Zaradi naštetih razlogov, ki prinašajo prednost Linux sistema, smo se pri razvoju Margento CarPC (MCPC) naprave odločili za uporabo RPi *compute* modula. MCPC zahteva uporabo vseh SPI in UART komunikacijskih vodil, ki so na voljo, ter tudi razširitev dodatnih zunanjih UART komunikacijskih vodil. Zaradi prvotnega namena RPi, kot učnega pripomočka, je programska podpora za uporabo zahtevanih perifernih enot precej omejena. Zato, smo na osnovi odprtokodnega Linux sistema le tega prilagodili, ter dogradili potrebam MCPC naprave.

V članku najprej podrobneje predstavimo problem posamezne periferne enote. V nadaljevanju podrobneje podamo okvirno strukturo GNU/Linux operacijskega sistema, na podlagi česar pojasnimo način dodajanja dodatnih funkcionalnosti v Linux sistem. Nato se osredotočimo na rešitve posamezne periferne enote, kjer pojasnimo pot do delujočih rešitev.

2 SPI in UART vodila na Raspberry Pi

RPi ima dva tipa komunikacijskih vodil [1]. Prvi tip vodil sta splošno UART0 in SPI0 vodilo, drugi tip so pomožna (auxiliary) komunikacijska vodila mini UART1, SPI1 in SPI2. Prvi tip komunikacijskih vodil, je fizično dostopen na vseh različnih modelih RPi in ima tudi programsko podporo na distribuciji operacijskega sistema Linux (Raspbian), ki je namenjen uporabi na RPi. Za razliko od prvega tipa, drugi tip vodil ni prisoten na vseh modelih RPi, razen na *compute* modulu. Ta vodila se tudi razlikujejo od UART0 in SPI0 na nivoju strojne opreme, ter nimajo programske podpore (gonilnikov) za njihovo delovanje. Da smo lahko uporabili vsa prisotna komunikacijska vodila, smo dogradili programsko podporo za nepodprta vodila, ter dodali dodatne funkcionalnosti za potrebe MCPC naprave.

2.1 UART standard

Universal asynchronous receiver/transmitter (UART) se običajno uporablja v kombinaciji s standardi EIA, RS-232, RS-422 in RS-485 [2]. Tekom razvoja UART komunikacijskega vodila so se, na podlagi raznovrstnih izboljšav fizičnega vmesnika, ter prilagajanju posameznim aplikacijam, uveljavili različni modeli UART komunikacijskega vodila.

Za UART0 komunikacijsko vodilo na RPi, je značilen ARM (PL011) model implementacije. Ta model se rahlo razlikuje od splošnega modela 16650. Variacije med omenjenima modeloma, so rahla prilagajanja modela za implementacijo na ARM arhitekturi mikroprocesorja. Za razliko od UART0, pomožno UART1 vodilo znano tudi kot mini UART, ni kompatibilno z modelom 16650. Struktura nastavitvenih registrov je podobna kot pri modelu 16550, vendar ne podpira polne funkcionalnosti 16550 modela. Okrnjena različica 16550 modela, podpira nekaj osnovnih funkcionalnosti, ki se najpogosteje uporabljajo v kombinaciji z RS-232 standardom. Za razliko od UART0, mini UART ne podpira naslednjih funkcionalnosti:

- Detekcije prekinitve komunikacije (Break detection).
- Detekcije napake komunikacijskega okna (Framing error detection).
- Paritetnega bita (Parity bit).
- DCD, DSR, DTR in RI signalnih linij.

Mini UART prav tako ne podpira uporabe DMA (Direct Memory Access) enote. DMA enota, razbremeni CPU s prenašanjem podatkov direktno med dvema pomnilniškima mestoma, med pomnilnikom in periferno enoto, ali med dvema perifernima enotama.

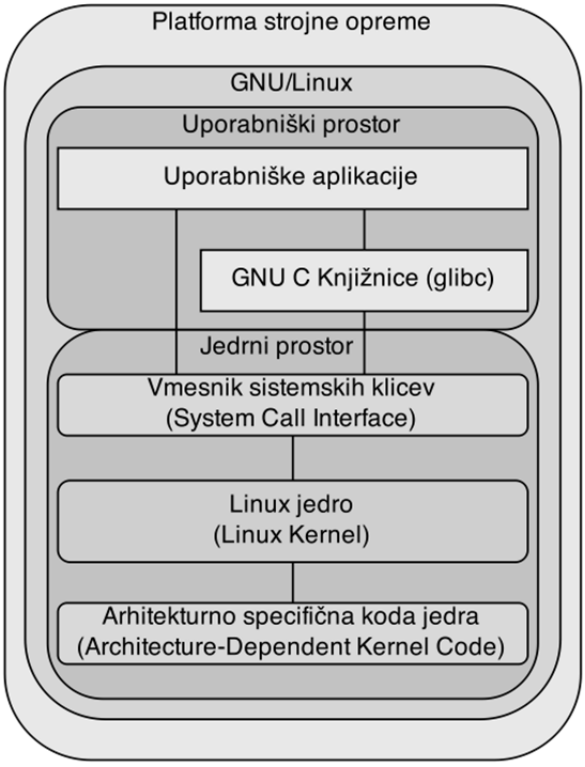
2.2 SPI standard

Serial periferal interface (SPI) komunikacijsko vodilo, se uporablja za povezovanje zunanjih perifernih enot z mikroprocesorjem. Prenos podatkov najpogosteje poteka dvosmerno istočasno, (full-duplex) z uporabo *master-slave* strukture. SPI vodilo nima formalnega standarda, (prilagojeni standardi – *de facto* standard) in je prišel v uporabo predvsem zaradi uporabe v izdelkih na tržišču. Posledica tega, je tudi precej različic strukture SPI vodila, ki so se razvile zaradi prilagajanja aplikacijam v katerih se je SPI vodilo uporabljalo [3].

SPI vodila na RPi imajo *master* strukturo in lahko delujejo le v tem načinu (*slave* struktura SPI vodila podprta s strani BSC – Broadcom Serial Controller). SPI0 vodilo deluje v standardnem načinu, ki se je najbolj uveljavil oziroma se uporablja najpogosteje. RPi ima samo eno SPI vodilo tega tipa (SPI0), ki je tudi podprto s strani programskega vmesnika. SPI1 in SPI2 vodili na RPi prav tako implementirata *SPI master* strukturo, vendar se njuna struktura (registrov) razlikuje od SPI0. Slednja omogoča nastavitev delovanja vodila v širšem območju, kateri zajema poleg najpogosteje uporabljenega načina delovanja tudi druge uporabljane variacije. Zaradi tega sta SPI1 in SPI2 imenovana tudi *Universal SPI Master*. Za razliko od SPI0, SPI1 in SPI2 ne podpirata uporabe DMA enote. Posledica tega je precej večja obremenitev CPU enote, kar povzroči omejitev hitrosti prenosa pri višjih urinih taktih SPI vodila.

3 Struktura operacijskega sistema Linux

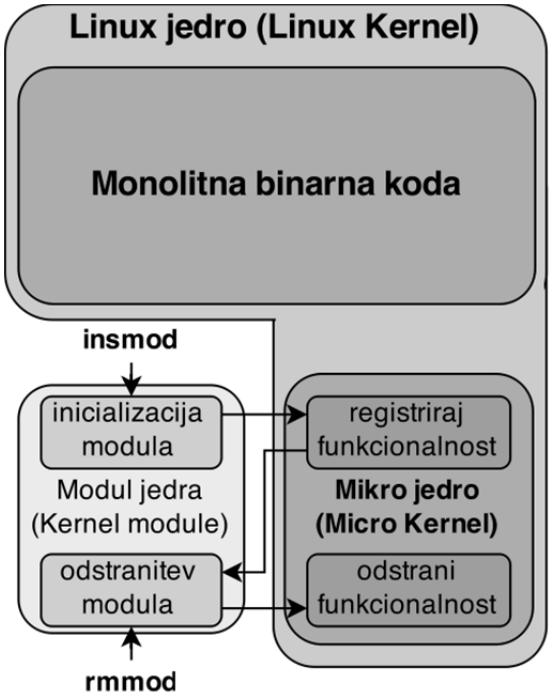
Za lažjo predstavitev prilagajanja in razširjanja funkcionalnosti na operacijskem sistemu Linux, razložimo strukturo sistema, ki je prikazana na sliki 3.1. Celotni Linux sistem delimo na dva večja bloka. Prvi blok vsebuje uporabniške aplikacije, ki s pomočjo standardnih C sistemskih knjižnic dostopajo do jedra operacijskega sistema, kamor lahko dostopajo tudi direktno. Ta blok se tudi imenuje *lupina* (shell) operacijskega sistema Linux. Drugi večji blok, ki opravlja osnovne naloge operacijskega sistema kot je razvrščanje procesov, je jedrni del. Ta vsebuje vmesnik sistemskih klicev, za povezavo višje nivojskih aplikacijskih klicev, z Linux jedrom. Pod Linux jedrom se nahaja koda jedrnega dela, ki je specifična za arhitekturo procesorja, na katerem teče operacijski sistem. Slednja, povezuje jedro operacijskega sistema s strojno opremo mikroprocesorja [4].



Slika 3.1: Struktura operacijskega sistema GNU/Linux.

Jedro Linuxa je zasnovano kot kombinacija monolitne binarne kode in *mikro-jedra,* kot prikazuje slika 3.2. Monolitna binarna koda vsebuje vso kodo za delo z uporabnikom, strojno opremo mikroprocesorja ter dodanimi funkcionalnostmi drugih (zunanjih) periferij. Monolitna koda je nameščena skupaj z jedrom in je pripravljena za hitri dostop. Slabost monolitne kode je potrebno prevajanje celotnega Linux jedra, v primeru dodajanja ali

odstranjevanja posameznih funkcionalnosti. Za razliko od monolitne binarne kode, je *mikro-jedro* zasnovano iz manjših binarnih sklopov (modularna zasnova), ter tako omogoča dodajanje in odstranjevanje posameznih funkcionalnosti (v obliki modulov), tekom delovanja operacijskega sistema. Slabost takšne modularne zasnove se odraža v počasnejšem delovanju posameznih manjših sklopov.



Slika 3.2: Modul Linux jedra.

Slika 3.2 prav tako prikazuje, način dodajanja modula jedra v *mikro-jedro*. Proces registracije modula se začne z *insmod* ukazom, ki mu sledi ime modula s končnico *ko* (kernel object). Od te točke dalje se izvede samo-inicializacija modula, ki vsebuje metode za lokalne nastavitve, ter knjižnice za registracijo funkcionalnosti v *mikro-jedru*. Podobno kot inicializacija modula, poteka tudi odstranitev modula, ki jo vzbudi komanda *rmmod* ali pa neuspešna registracija funkcionalnosti v *mikro-jedru*. Aktivacija inicializacije/odstranitve modula, je možna tudi s strani druge funkcionalnosti (ni prikazano na sliki 3.2), v primeru da je slednja odvisna od tega modula. Več o programiranju modulov je predstavljeno v elektronskem viru [5].

4 Podpora mini UART, SPI1 in SPI2 komunikacijskih vodil

V predhodnem poglavju, smo navedli opis strukture Linux operacijskega sistema, ter zgradbo Linux jedra, v katerega smo dodali potrebno podporo, za nepodprta periferna vodila mini UART, SPI1 in SPI2, na modularni način. S tem, smo se izognili ponovnemu prevajanju celotnega Linux jedra, za vsako nadgrajeno različico jedra. Z izbiro modularnega dodajanja funkcionalnosti, nismo dosegli maksimalne zmogljivosti dodanih funkcionalnosti, so pa dosežene zmogljivosti sprejemljive s strani zmogljivostnih zahtev za potrebe MCPC naprave. Vse dodane funkcionalnosti izvirajo/temeljijo na izvorni kodi Linux jedra *rpi-3.12.y* iz *raspberry* izvorne kode na *GitHub* [6].

4.1 mini UART

Izvorna koda Linux sistema, vsebuje precej več funkcionalnosti, kot so privzeto vključene v monolitno binarno kodo, ali v *mikro-jedro*. Zato je smiselno pregledati izvorno kodo za morebitne obstoječe rešitve. Pri pregledu izvorne kode Linux jedra, je bila ugotovljena obstoječa funkcionalnost za mini UART komunikacijsko vodilo, pod oznako modela 8250 (model UART vodila). Izvorna koda 8250 privzeto ni vključena v monolitno binarno kodo, niti v modularno *mikro-jedro*. Preden smo dosegli delujočo podporo za mini UART vodilo, je bilo potrebnih nekaj popravkov pri nastavitvi takta ure vodila, ki se je sprva nastavil napačno. Delujočo kodo smo prevedli v modul jedra, ki ga lahko nadalje uporabimo tekom delovanja.

4.2 SPI1 in SPI2

Za razliko od mini UART vodila, SPI1 in SPI2 nimata podpore v izvorni kodi Linux sistema. Kot smo podali v poglavju 2.2, lahko rečemo da imata SPI1 in SPI2 univerzalno strukturo SPI vodila (možnost podpore več različic SPI vodil). Posledično obstaja tudi možnost uporabe vodil SPI1 in SPI2, z enakimi funkcionalnostmi kot uporabljamo vodilo SPI0. To smo dosegli tako, da smo v izvorni kodi vodila SPI0, prilagodili povezave med

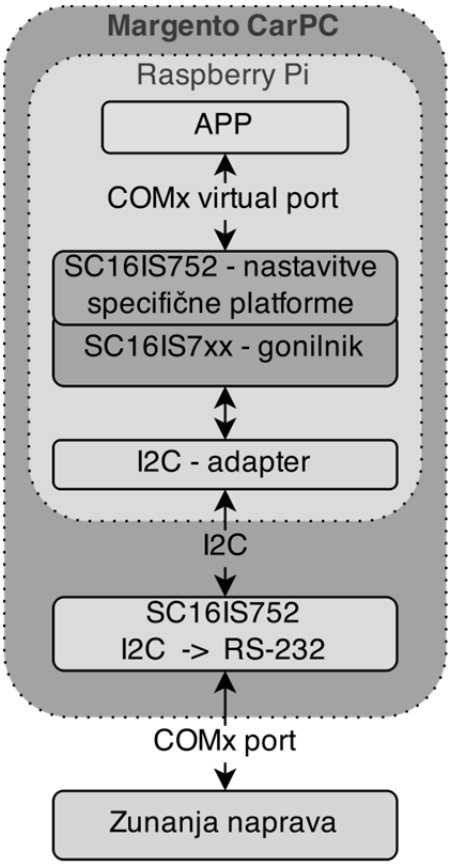
funkcionalnostjo vodila SPI0 in registri periferij SPI1 ter SPI2 vodil. Opomniti velja, da imata vodili SPI1 in SPI2 enako strukturo, posledično podpirata enake funkcionalnosti, in lahko uporabimo isti modul za obe vodili. Prilagojeno kodo vodil smo prevedli v modul, ki ga uporabimo za inicializacijo SPIx vodila v *master* načinu. Podrobneje o inicializaciji gonilnika SPI vodila, dosegljivo na povezavi [7].

5 Razširitev zunanjih UART vodil preko I2C vodila

MCPC naprava uporablja notranji UART vodili (UART0 in mini UART), za komunikacijo z GPRS in GPS napravama. Za komunikacijo MCPC naprave z zunanjimi napravami, se pojavi zahteva po dodatnih UART vodilih. Za implementacijo slednjih, MCPC uporablja integrirana vezja SC16IS752, ki podpirajo I2C/SPI *slave* način delovanja ter UART model 16C450 na drugi strani. Omenjeno integrirano vezje omogoča pretvorbo I2C/SPI protokola, na RS-232/RS-485. Specifična za MCPC je pretvorba I2C na RS-232 protokol. Potek komunikacije med aplikacijo in zunanjo napravo, je prikazan na sliki 5.1, kjer je poudarjen manjkajoči člen, sestavljen iz gonilnika družine integriranih vezij SC16IS7xx, ter inicializacije specifičnega integriranega vezja SC16IS752.

5.1 I2C adapter

Za povezavo komunikacije med gonilnikom SC16IS7xx, ter integriranim vezjem SC16IS752, je uporabljeno I2C vodilo. Gonilnik za dostopanje do I2C vodila, se v jedru Linux sistema prav tako nahaja kot modul. Na sliki 5.1 je I2C gonilnik predstavljen kot I2C-adapter. Dostop do I2C-adapterja, je zagotovljen preko I2C knjižnic, ki jih uporablja gonilnik SC16IS7xx. Modul I2C-adapterja je neodvisen od drugih modulov, in direktno dostopa do I2C periferne enote mikroprocesorja.



Slika 5.1: Komunikacija aplikacije z zunanjo napravo.

5.2 Gonilnik SC16IS7xx

Za razliko od I2C neodvisnega modula, je modul SC16IS7xx gonilnika odvisen od modula SC16IS752 za nastavitev specifičnega integriranega vezja. Vključitev modula SC16IS7xx v Linux jedro, ne bo sprožila samo-inicializacije le tega. Inicializacija modula v tem primeru poteka drugače, saj je inicializacija odvisna od specifičnih podatkov za določeno integrirano vezje oziroma napravo. SC16IS7xx modul vsebuje tabelo naprav, katerih funkcionalnosti podpira. Z vključitvijo modula za inicializacijo specifične naprave (SC16IS752) v Linux jedro, podamo specifične podatke naprave, ki jo želimo nastaviti. Linux jedro preišče sezname naprav vseh obstoječih modulov, in v primeru uspešnega iskanja, poda specifične podatke naprave, ter sproži inicializacijo modula, ki podpira funkcionalnosti iskane naprave.

Izvorna koda uporabljenega Linux jedra, ne zajema funkcionalnosti družine integriranih vezij SC16IS7xx. S preiskavo izvornih kod drugih različic Linux jeder, je bilo ugotovljeno, da različica *3.16.y* in dalje, podpirajo iskane funkcionalnosti. Izvorni kodi različic *3.12.y* ter *3.16.y* niso do potankosti kompatibilni. Za so bile funkcionalnosti integriranih vezij SC16IS7xx, prenešene iz različice *3.16.y* na različico *3.12.y*, z ustreznimi prilagoditvami. Za inicializacijo funkcionalnosti specifične naprave, je bil ustvarjen nastavitveni modul SC16IS752, ki vsebuje podatke o specifični platformi naprave in ob vključitvi sproži samo-inicializacijo SC16IS7xx gonilnika.

6 Zaključek

V prispevku smo predstavili problem omejene podpore zahtevanih funkcionalnosti, ter razširitev zunanjih funkcionalnosti pri razvoju Margento CarPC naprave. Podali smo okvirno strukturo operacijskega sistema GNU/Linux, s čimer smo pojasnili medsebojno povezavo osnovnih gradnikov. Na podlagi strukture Linux sistema, je razložen način dodajanja novih funkcionalnosti na Linux platformi. Na podlagi česar, smo predstavili novo dodano podporo za vodila mini UART, SPI1, SPI2 ter razširitev zunanjih funkcionalnosti.

Pri preizkusu notranjih perifernih vodil, mini UART, SPI1 ter SPI2, nismo ugotovili kakršnihkoli problemov glede zmanjšanih zmogljivosti ali napak v samem delovanju. UART0 in mini UART vodili, sta uporabljeni za komunikacijo z GPRS/GSM ter GPS moduloma, ki ne zahtevata pretirane hitrosti UART vodila. SPI1 vodilo je uporabljeno za komunikacijo z micro-SD pomnilniško kartico, med tem ko SPI2 vodilo za enkrat ni uporabljeno. SD pomnilniška kartica, je namenjena shranjevanju zgodovine podatkov o delovanju aplikacij, ter morebitnih informacijah o napakah. Pasovna širina SPI1 vodila, kljub omejitvam DMA enote, ter modularni obliki gonilnika, preverjeno zadošča potrebam MCPC naprave.

Pri preizkusu razširitve zunanjih UART vodil, smo ugotovili težavo, ki se pojavlja v zvezi z neuporabo signalnih linij nadzora pretoka podatkov (DTR, RTS). V tem primeru se zgodi, da je pasovna širina I2C vodila manjša, od skupne pasovne širine vseh UART vodil, ter prihaja do izgube podatkov. To se dogaja predvsem pri sočasni uporabi več pretvornikov SC16IS7xx preko enega I2C vodila, ter uporabi maksimalnih hitrosti UART vodil. Rešitev problema je uporaba signalnih linij za nadzor pretoka podatkov, ali pa povečanje pasovne širine z uporabo SPI vodila. Kot zunanje naprave, se bodo preko RS-232 protokola povezovali, *validator* terminali, termični tiskalnik ipd. Za namene MCPC naprave, so dosežene zmogljivosti zunanjih UART vodil, preko I2C vodila, preverjeno zadovoljive.

7 Literatura

[1] Broadcom Corporation. BCM2865 ARM Peripherals. URL: http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf (Citirano 23.2.2015).

[2] Universal asynchronous receiver/transmitter. Wikipedia. URL: http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter#Synchronous_transmission (Citirano 23.2.2015).

[3] Serial Peripheral Interface Bus. Wikipedia. URL: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus (Citirano 23.2.2015).

[4] M. Tim Jones, Anatomy of the Linux kernel. 2007. URL: http://www.ibm.com/developerworks/library/l-linux-kernel/ (Citirano 23.2.2015).

[5] S. Peter Jay, The Linux Kernel Module Programming Guide. URL: http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html (Citirano 23.2.2015).

[6] Raspberry Pi Linux kernel source code. GitHub. URL: https://github.com/raspberrypi/linux/tree/rpi-3.12.y (Citirano 23.2.2015).

[7] B. Mark, D. Brownell, R. King, G. Likely, D. Pervushin, S. Street, M. Underwood, A. Victor, L. Walleijm V. Wool, Overview of Linux kernel SPI support. 2012. URL: https://www.kernel.org/doc/Documentation/spi/spi-summary (Citirano 23.2.2015).