НАУЧНИ ТРУДОВЕ НА
УНИВЕРСИТЕТ ПО ХРАНИТЕЛНИ
ТЕХНОЛОГИИ - ПЛОВДИВ
ТОМ LXII
2015 г.

SCIENTIFIC WORKS OF
UNIVERSITY OF FOOD
TECHNOLOGIES
VOLUME LXII
2015

СОФТУЕРЕН КОНТРОЛ НА GPIO ПРИ SOC ПОД УПРАВЛЕНИЕ НА LINUX

Янислав Картелов[1], Стойчо Христозов[1]
Университет по хранителни технологии - Пловдив[1];

SOFTWARE CONTROLLED GPIO FOR LINUX DRIVEN SOC

Ianislav Kartelov[1], Stoicho Hristozov[1]
UFT – Plovdiv[1];

Abstract

The following work gives an approach of using Linux operating system on System on chip devices and connecting such system to a wide variety of hardware modules. General purpose input/output pins are interface for establishing physical connection with electronic digital signals. In details is covered entire process from theoretical basics to building a working software solution. Build GPIO driver is based on standard for Linux - Linux Character device driver. Set of testing hardware modules as well as testing routines are done over a working system.

Keywords: *GPIO control; SOC; embedded linux; Character device driver*

Въведение

Линукс, като изключително преносима операционна система, може да бъде изпълнявана на голям брой от хардуерни платформи. В списъка, на добре познатите и поддържани от ядрото са ARM, MIPS, x86 (PC), PowerPC и много други. Освен за персонални компютри, сървъри и мейнфрейм компютри, системата е подходяща за управление на малки електронни устройства, познати под общото наименование: вградени системи(embedded). За този клас е характерно, да се изграждат за изпълнение на определени задачи. Обикновено се базират на специфичен микроконтролер, като основен управляващ елемент. Базовия хадуер на компютърната система е изграден в един чип на микроконтролера, т.н. система на чип (SOC) устройство. Може да присъства и допълнителен, обслужван от контролера хардуер за различни цели, към който се прави връзка - например сензори и/или комуникация.

Заедно с предимствата от използване на преносима операционната система за вградените устройства, произтичат и определени трудности. Разработването на необходимите софтуерни драйвери - модули към ядрото, специфични за целевата хардуерна платформа е трудна и специфична задача, предполагаща развиването на набор от правила от ниско ниво - интерфейс към определени специфични хардуерни модули.

В настоящата работа е разгледан подход за развиване на GPIO драйвер на базата на Linux character device драйвер и прилагането му към определена хардуерна платформа.

GPIO драйвер

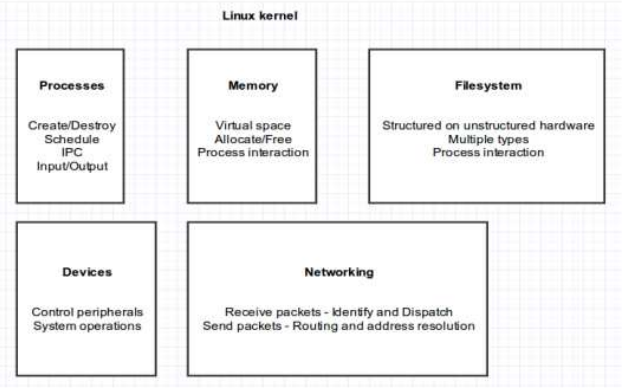
Организация на драйвера. Микроконтролерите обикновено разполагат с определен набор входно/изходни пинове от общо предназначение (GPIO). Към тези пинове на потребителя са предоставени 2, 3, 5 и повече проводни шини като I2C, UART, SPI, PWM и цифрови сигнални. Отделните пинове могат да бъдат конфигурирани в следните варианти:

-във високо или в ниско логическо ниво при изход;

-за прочитане на логическо ниво при вход;

-за прекъсване;

Изграждането на система под управление на Linux ядро, дава на потребителя възможност за абстрактен поглед от високо ниво върху хардуерните детайли. Приложението извиква системни функции, достъпни чрез ядрото, когато желае да се обърне към определен хардуерен модул. От своя страна, ядрото приема заявката и управлява хардуера включен в състава на заявката. Посочения модулен подход дава по-сигурен и стандартизиран интерфейс в сравнение с реализиране на директен хардуерен достъп от приложението.

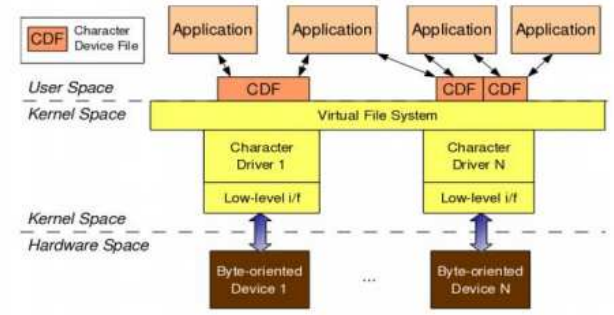


Фиг. 1. Организация на ядрото на Linux

На фигурата (фиг.1.) са посочени основните блокове, изпълнявани от ядрото на Linux. Компонента за управление на файлова система дава налична файлова система (напр. FAT) от неструктуриран хардуерен източник (напр. SD карта).

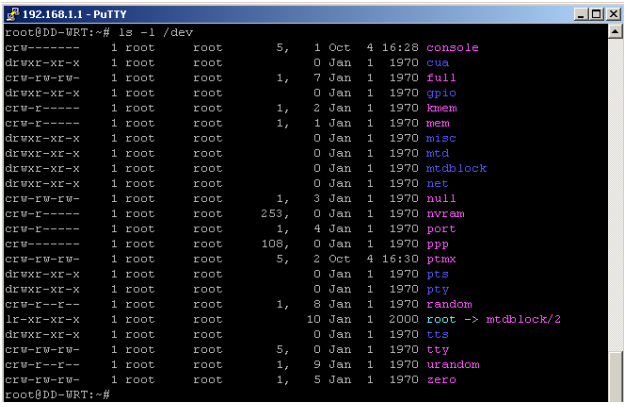
Компонента за управление на устройства е от интерес за настоящата работа. Още наричан драйвер на устройствата, този модул дава възможност за абстрактно управление – интерфейс към специфичния хардуер. Погледнат като размер на софтуера, този компонент е най-обхватен, доколкото в света са налични много голям брой специфични хардуерни устройства.

Linux Character Device е едно от трите стандартни за Linux устройства, заедно с network и block устройствата, съответно използвани за мрежови и файлови функции. В обхвата на символното устройство попадат интерфейсите за вход, серийния, за графика и звук, както и управление на по-специфичен хардуер. На фигурата (фиг.2.) е разгледана организацията на драйвера на устройствата. Той е байтово ориентиран(символно в аспект на езика С) и осигурява комуникация от този тип, каквито са и болшинството от хардуерните устройства. Файл от символен тип е наличен в директорията: /dev и играе роля на комуникационен канал между приложенията от високо ниво и хардуерните модули свързани към системата. Стандартизираните заявки (open, read, write, close) се обработват от драйвера на виртуалната файлова система (VFS) и се превеждат до обръщения от ниско ниво към конкретните свързани хардуерни устройства.



Фиг. 2.Организация на Character device драйвера

Всеки драйвер, регистриран в ядрото на операционната система се свързва с номера, идентифициращи конкретния драйвер, като един и същи драйвер може да има няколко инстанции за различни устройства. Показано на фиг.3, номера от 5-та колона показва драйвера а номера от 6-та колона показва инстанция на един и същи драйвер.



Фиг. 3.Списък на устройствата, поддържани от драйвера и показвани в директория: /dev

GPIO драйвера е базиран на стандартния character device драйвер. Интерфейсната част на този драйвер към приложенията на високо ниво е организирана като файл. За тази цел е необходимо да бъдат дефинирани входните точки на методите, които позволяват работа с файл от приложенията. Описани са като структурата struct file_operations, дадена в <linux/fs.h>. Описание на някой от основните входни точки е дадено на фиг.4.

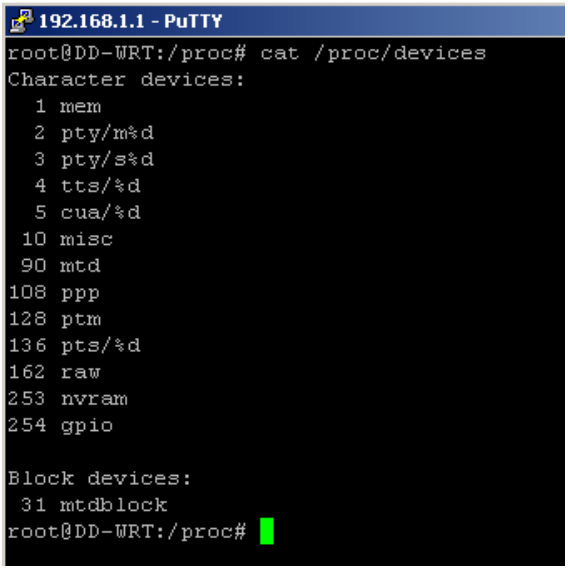
```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *,size_t, loff_t *);
        int (*open) (struct inode *, struct file *);
        int (*release) (struct inode *, struct file *);
};
```

Фиг.4. Описание на входни точки за GPIO драйвера

Освен описването на входните точки и регистрирането им в ядрото е необходимо описване и на следните компоненти, групирани общо по код и по данни. От гледна точка на код, се реализира инициализираща функция, набор от методи за извикване от приложенията, като посочените на фиг.4. и допълнителни служебни функции като обслужване на прекъсвания и др. От гледна точка на данни е необходимо, да се реализират форматите на данни за определяне на състоянието и за пренасяната информация.

Разработване на драйвера. След написване на посочените елементи от софтуера е необходимо компилиране на ядрото. Процеса е многократен поради необходимостта от изчистване на евентуални грешки и достигане до стабилна, работеща версия на софтуера.

Като резултат получаваме работещо изображение на ядро и директория със съответните модулни драйвери. Лист на инсталираните в системата драйвери за устрйства е наличен като файл: /proc/devices (фиг.5.). Различните хардуерни платформи реализират собствен bootup процес, но най-общо се зарежда компресирано изображение на ядрото поставено в пътя на bootloader-a.



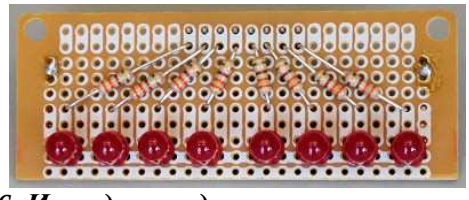
Фиг.5. Списък на инсталираните устройства

След успешно прекомпилиране и зареждане на ядрото може да се премине към реализиране на отделните функции и системата може да бъде тествана. За всеки отделен хардуерен модул, който ще бъде поддържан, се описва структура за модула и методите по инициализиране и спиране. В зависимост от конкретния хардуерен модул тези стъпки включват обмен на данни и установяване на комуникационните портове в дадени състояния. Освен инициализиращ и стопиращ метод се реализират и стандартните за интерфейс като фаил. Сред тях са Open, Release, Read, Write, както и методи за обслужване на прекъсвания.

Тестване на драйвера

Подготовка за тестване. Тестването е последната фаза от разработката на GPIO драйвера. Целта на тестването е да се провери дали драйвера работи коректно според набор от зададени тестови параметри. Използва се метода на тестване ”черна кутия” т.е. отделните функции се тестват с подаване на граничните интерфейси на определен вход и проверка на резултата. Това са трите режима на вход, изход и обработка на прекъсване. За целта на тестването е ралзиран специфичен хардуерен модул (фиг.6.) състоящ се от LED индикатори и обслужващите ги повторители и ограничаващи резистори за тестване на режима на изход и набор от резистори и ключове за тестване на входа(фиг.7, фиг.8). и обслужване на прекъсване. От страна на системата е написана и се извиква команда (приложение) gpio, която изпраща сигнали според желан формат или пък извежда приетите от входа сигнали.

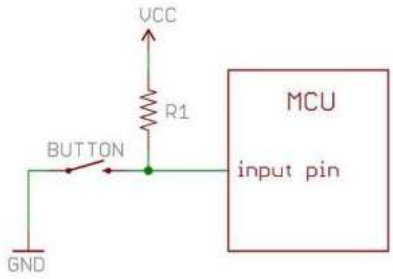
Тестване на изхода. Стартиране на тест за изхода се прави с командата gpio <enable|disable> <pin>, където <enable|disable> установява съответно лог. 1 или лог. 0 на порт зададен с номер на <pin>.



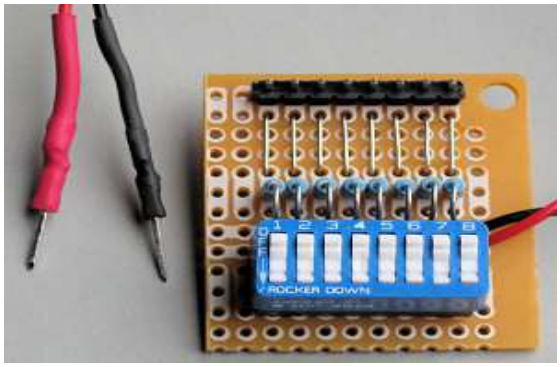
Фиг.6. Изглед на модул за тестване на изхода

Тестване на входа. Тестване на вход се изпълнява по аналогичен начин. Командата изглежда така: gpio poll <pin>. С промяна на логическото ниво на пина се предизвиква

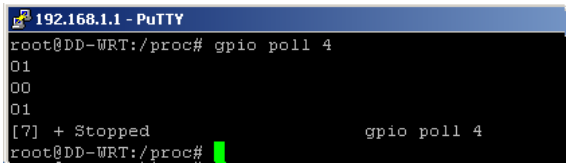
извеждане на съобщение със съответната стойност 00 или 01 (фиг.9).



Фиг.7. Принципна схема тестване на входа



Фиг.8. Изглед на модул за тестване на входа



Фиг.9. gpio за проверка на входа

Резултата от проведеното тестване показва съответствие между подаваните данни и регистрирания отговор. Успешно се установяват както изходен така и входен режим. Софтуерният драйвер работи коректно. Драйвера е реализиран към конкретна версия и компилация на ядрото, и това трябва да се съблюдава. Възможностите за автоматично обновяване, предлагани от операционната система следва да не се използват.

Заключение

Броя, разновидностите, областите на приложение, а с това и значимостта на вградените системи в днешни дни много бързо нараства. Linux в частност, като операционна система, работеща на такива системи, също предизвиква голям интерес. Поради характерните за Linux: отворен код, динамично развитие от работни групи в световен мащаб и популярност, правят системата важна за изучаване и използване. От друга има голямо разнообразие от хардуерни модули, които да бъдат управлявани, и техният брой също нараства непрекъснато.

Предложената работа детайлно навлиза в свързващото звено между трите компонента – вградено устройство, операционна система, хардуерен модул. Преминат е цялостно процеса от теоретична база до построяване на работещ софтуерен код. Не е избран конкретен модел или устройство, по-скоро се разглежда подход, който може да се приложи за гама от устройства и с него да се осигури връзка и управление на произволен хардуерен компонент с познат интерфейс и протокол.

Възможността за достъп до свързаните хардуерни модули през операционната система, тяхното управление и програмиране, значително разширява свободата и възможностите предоставяни от вградените устройства. Това способства изучаването, използването и развиването им.

Литература

[1] Venkateswaran S. Essential Linux Device Drivers. The United States of America:Prentice Hall; 2009

[2]Hallinan C. Embedded Linux Primer, A Practical Real World Approach. The United States of America: Prentice Hall; 2010.

[3]Cooperstein J. Writing Linux Device Drivers, a guide with exercises. CreateSpace Independent Publishing Platform; 2009

[4] Corbet J. Linux Device Drivers, Third Edition. The United States of America:O'Reilly Media, Inc.; 2005

[5]https://learn.sparkfun.com/tutorials/pull-up-resistors/all.

[6] http://www.valvers.com/embedded-linux/raspberry-pi/step01-bare-metal-programming-in-cpt1

[7] https://projects.drogon.net/raspberry-pi/wiringpi/pins/

[8]http://en.wikipedia.org/wiki/General-purpose_input/output.

[9]http://www.opensourceforu.com/2011/02/linux-character-drivers.