Linux i Python w Elektronicznej Sieci – ćwiczenia bis #2: Lab komputerowo - elektroniczny

Projekt „Matematyka dla Ciekawych Świata”,
Robert Ryszard Paciorek
<rrp@opcode.eu.org>

2021-08-03

1 I²C

W ramach kursu omawiana była magistrala I2C zarówno od strony elektronicznej jak i obsługi jej w mikrokontrolerach STM32. Magistrala ta jest powszechnie używana w komputerach stacjonarnych i laptopach, jednak trudno w nich dostać się do odpowiednich połączeń. Z magistrali tej bez problemów możemy korzystać także w komputerach jednopłytkowych typu Pi (gdzie znajduje się na złączu GPIO) obsługiwanych z poziomu Linuxa.

Linux dostarcza narzędzi do obsługi tej magistrali - zarówno z linii poleceń jak i z poziomu kodu źródłowego własnego programu. Poniżej znajdują się przykłady użycia.

1.1 narzędzia linii poleceń

Narzędzia te dostarczane są przez pakiet *i2c-tools*. Podstawowe dwa polecenia służą do odczytu i zapisu wskazanego rejestru urządzenia I2C:

- `i2cget szynai2c adres_ukladu adres_rejestru` – odczytuje rejestr o adresie `adres_rejestru`, z układu I2C o adresie `adres_ukladu` znajdującego się na wskazanej magistrali I2C (`szynai2c`)
- `i2cset szynai2c adres_ukladu adres_rejestru wartosc` – zapisuje podaną wartość do określonego rejestru układu I2C

Pakiet dostarcza także inne polecenia, z których należy wspomnieć o:

- `i2cdetect -l` – listuje magistrale I2C
- `i2cdetect szynai2c` – listuje urządzenia na wskazanej magistrali I2C
- `i2cdump szynai2c adres_ukladu` – listuje rejestry wskazanego układu I2C

Uwaga: Użycie tych poleceń może być niebezpieczne, zwłaszcza gdy nie wiemy jakie układy znajdują się na danej magistrali I2C. Związane jest to z tym iż odczyt rejestru o danym numerze wiąże się z operacją zapisu do układu I2C, a taka (przez układ nie obsługujący adresowania rejestrowego) może być zinterpretowana jako zwykły zapis do układu. Dlatego odradzamy eksperymentowanie z tymi polecenia w laptopach i komputerach stacjonarnych.

1.2 kod C

Przykładowy kod C/C++ demonstrujący użycie komunikacji I2C znajduje się na http://vip.opcode.eu.org/#komunikacja_I2C. Warto zwrócić uwagę na wspomniany wyżej sposób wskazywania adresu rejestru, który będziemy czytać przez dokonanie zapisu do układu. Co może być problematyczne dla układów nie obsługujących takiego adresowania.

Linux na komputerach jednopłytkowych

Typowo instalacja Linuxa dla komputerka jednopłytkowego typu *Pi lub podobnego sprowadza się do nagrania pobranego obrazu systemu (Raspbian dla Raspberry Pi, Armbian dla prawie całej reszty) na kartę SD z użyciem np. polecenia `dd`. Możliwe są także inne metody przygotowania obrazu, np. z użyciem `qemu-debootstrap` pozwalające na dostosowanie tworzonego obrazu[a].

Główną różnicą w stosunku co do systemu na komputerach stacjonarnych i laptopach jest proces startu – nie mamy tutaj doczynienia z BIOSem i GRUBem. Zamiast tego mikrokontroler znajdujący się na płytce posiada wbudowany jakiś bootloader, który typowo potrafi załadować kolejny bootloader z karty SD i przekazać do niego sterowanie.
W przypadku Raspberry ten pierwszy etap bootloadera na karcie SD znajduje się w pliku (identyfikowanym nazwą na partycji typu FAT) `bootcode.bin`, a kolejny to `start.elf` (korzystający z plików `config.txt`, `cmdline.txt` i `kernel.img`).
W przypadku większości innych płytek wykorzystywany jest bootloader *U-Boot* wgrywany na kartę SD (poza systemem plików na niej umieszczonym). Rozwiązanie trochę przypominające to z czym mamy do czynienia w komputerach typu PC. Bootowanie jednak na ogół nie rozpoczyna się od początku karty SD (MBR), a od określonego miejsca na tej karcie (np. 8kiB dla sunxi), gdzie umieszczany jest *U-Boot SPL*. Zajmuje się on m.in. inicjalizacją pamięci i przekazuje sterowanie do właściwego kodu *U-Boot* znajdującego się w dalszej części karty SD (także poza jakąkolwiek partycją – typowo na 40kiB karty SD, pomiędzy MBR a pierwszą partycją). Dopiero ten kod ładuje pliki konfiguracyjne i obrazy jądra z odpowiedniej partycji na karcie SD.[b]

Po załadowaniu jądra wykorzystywany jest typowy systemd[c] znany z „Linuxa" na komputerach PC, który możemy zastąpić nawet zwykłym skryptem sh. Po uruchomieniu otrzymujemy typowy system typu Debian GNU/Linux, konfigurowany przy pomocy standardowych wpisów w `/etc`, `/proc` oraz typowych narzędzi (jak np. `systemctl`, `ip`, `apt`). Różnicą jest dostęp do pinów GPIO mikrokontrolera oraz peryferiów na nich dostępnych.
Za mapowanie poszczególnych peryferiów na piny GPIO (czyli za to czy na danym pinie mamy np. UART-TX, I2C-SDA czy zwykłe GPIO) odpowiada w Linuxie mechanizm *Device Tree*. Systemy takie jak Raspbian czy Armbian dostarczają zestawy plików DTB aktywujących odpowiednie mapowania i celem uzyskania odpowiedniego peryferium należy je aktywować w konfiguracji startowej (`/boot/config.txt` w Raspbianie, `/boot/armbianEnv.txt` w Armbianie). Na przykład:

```
dtparam=i2c_arm=on
dtoverlay=i2c-bcm2708
```

dopisany do `/boot/config.txt` aktywuje interfejs I2C na starszych płytkach *Raspberry Pi*. Natomiast dodanie `i2c0` do linijki `overlays=` w `/boot/armbianEnv.txt` aktywuje I2C na pinach 3 i 5 (PA12, PA11) na *Orange Pi Zero*. Do aktywacji odpowiednich DTB w Armbianie można użyć polecenia `armbian_config`, następnie wejść w `System` i `Hardware`, gdzie można aktywować / dezaktywować takie mapowania. Więcej informacji na temat *Device Tree overlays*: https://docs.armbian.com/User-Guide_Allwinner_overlays/.
Kombinacje niektórych modeli Orange Pi z niektórymi switchami mają problemy z mechanizmem autonegocjacji prędkości łącza ethernetowego. Rozwiązaniem jest wymuszenie prędkości 100Mbit/s po obu stronach lub (jeżeli nie mamy możliwości konfigurowania switcha) ustawienie na OrangePi prędkości 10Mbit/s poleceniem: `ethtool -s eth0 speed 10 duplex full autoneg off`.

---

a. Taką metoda tworzenia lekkiego obrazu dla Raspberry Pi opisałemtrochę szerzej na http://blog.opcode.eu.org/2020/05/15/rpi-debootstrap.html.
b. Więcej na temat bootowania można dowiedzieć się np. z https://linux-sunxi.org/Bootable_SD_card.
c. Więcej informacji na http://www.opcode.eu.org/SystemBoot.xhtml#systemd

Zadanie 1.2.1

1. Zapoznaj się z dokumentacją używanego układu I/O korzystającego z magistrali I2C.
2. Korzystając z płytki prototypowej i kabelków połącz ten układ z magistralą I2C na płytce typu Pi. Pamiętaj o rezystorach podciągających linie zegara i danych – nie są one montowane na płytkach typu *Pi.
3. Przetestuj komunikację z układem i jego działanie korzystając z poleceń `i2cget` / `i2cset`.
4. Zmodyfikuj przedstawiony przykład użycia I2C w kodzie C/C++, tak aby uzyskać program odczytujący wartość z układu I2C i wypisujący informację o wartości poszczególnych wejść układu I2C. Program nie przyjmuje argumentów linii poleceń – adres szyny, układu i numer rejestru powinny być wpisane w kod (np. z użyciem `#define`).

Zadanie 1.2.2

1. Podłącz do magistrali I2C utorzonej w poprzednim zadaniu płytkę z mikrokontrolerem STM32. Układ I/O może pozostać na magistrali.
2. Zmodyfikuj przykładowe kody STM32 związane z obsługą ADC i I2C tak aby mikrokontroler udostępniał kilka rejestrów na szynie I2C w których będą dostępne wartości odczytywane przez przetwornik analogowo-cyfrowy.
3. Zmodyfikuj program w C/C++ tak aby odczytywać na *Pi dane z wejść ADC mikrokontrolera z użyciem I2C.

Zadanie 1.2.3

Zmodyfikuj przykładowe kody STM32 związane z obsługą I2C tak aby mikrokontroler odczytywał dane z układu I/O podłączonego do magistrali I2C. Odczytana wartość powinna być wypisywana w postaci tekstowej na porcie szeregowym.

2 UART

2.1 I^{2}C vs UART

I^{2}C jest bardzo popularną magistralą "lokalną" używaną do połączenia CPU z układami peryferyjnymi (którymi mogą być np. także inne mikrokontrolery). Ma ona ograniczony zasięg – typowo kilkanaście / kilkadziesiąt cm, raczej nie spotyka się kilkumetrowych rozwiązań. Na większych odległościach tego typu funkcje pełni UART[1], najczęściej w wariancie RS485, pozwalającego na łączenie do 32 urządzeń na jednej linii. W odróżnieniu od I2C sam UART nie określa żadnego sposobu adresacji i wykorzystywane są do tego inne protokoły, których powyżej UART/RS485 jest naprawdę bardzo dużo. Chyba najpopularniejszym jest Modbus RTU.

2.2 Modbus – adresacja i identyfikacja ramki

Modbus jest otwartym, prostym protokołem komunikacyjnym występującym w kilku odmianach – RTU, ASCII, TCP. Dwie pierwsze wykorzystują łącze szeregowe typu UART, trzecia używa pakietów IP/TCP. Protokół Modbus:

1. jest protokołem master-slave, czyli jest wyraźnie określone, które urządzenie inicjalizuje transmisję, a które jedynie odpowiada na otrzymane żądania

1. którego używaliśmy już m.in. do programowania mikrokontrolera STM32 i do wypisywania wiadomości "diagnostycznych" przez STM32

2. zapewnia adresację urządzeń (8 bitowy adres, zakres 1–247)
3. określa sposób dostępu do danych w urządzeniu (określany 8 bitowym kodem funkcji) i adresację tych danych (16 bitowy adres rejestru / wejścia binarnego / ...)
4. określa sposób identyfikacji początku i końca ramki

Ten ostatni punkt jest szczególnie istotny przy przesyłaniu danych z użyciem łącza typu UART, gdzie identyfikowany jest tylko początek/koniec bajtu, ale nie ma określonego sposobu identyfikacji początku/końca grupy bajtów którą jest np. ramka jakiegoś protokołu.

Modbus ASCII liczby reprezentujące adresy, kody funkcji i dane zapisuje w postaci tekstowej (jako liczby szesnastkowe), więc do identyfikacji początku i końca ramki mogą posłużyć inne niż (0-9A-F) znaki ASCII – początek ramki oznacza dwukropek (`:`), a koniec ciąg `\r\n`.
Modbus RTU[2] przesyła te wartości liczbowe binarnie (czyli po prostu w postaci danej liczby), zatem nie ma tutaj żadnej wolnej wartości, którą można by użyć jako znacznik początku / końca ramki. W tym przypadku służą do tego zależności czasowe:

- odstęp pomiędzy bajtami w ramce nie może przekroczyć 1.5 czasu trwania transmisji jednego bajtu
- odstęp pomiędzy ramkami musi wynosić przynajmniej 3.5 czasu trwania transmisji jednego bajtu

Pełną specyfikację protokołu można znaleźć na https://modbus.org/specs.php.

Zadanie 2.2.1

Korzystając z zaprezentowanego na zajęciach przykładowego kodu obsługi Modbus RTU dla STM32 (https://github.com/opcode-eu-org-libs/STM32-ModbusRTU), zmodyfikuj rozwiązanie zadania 2.2.1 tak aby na łączu szeregowym między *Pi a mikrokontrolerem stosowany był Modbus RTU.
Wskazówka: Obsługę portokołu Modbust RTU po stronie linuxa może zapewnić program mbpoll lub własny kod oparty o bibliotekę libmodbus.

2. którego obsługa jest wymagana przez standard dla urządzeń zgodnych z Modbus i jest on dużo popularniejszy niż wariant ASCII