Linux i Python w Elektronicznej Sieci – ćwiczenia #14: Lab komputerowo - elektroniczny

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

2023-06-25

W ramach kursu omawiana była magistrala I2C zarówno od strony elektronicznej jak i obsługi jej w mikrokontrolerach STM32. Tym razem będziemy wykorzystywać ją do komunikacji peryferiów (rejestru GPIO oraz samodzielnie programowanego mikrokontrolera) z komputerem jednopłytkowym (typu *Pi).

1 I²C w komputerach

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 magistrali I²C - zarówno z linii poleceń jak i z poziomu kodu źródłowego własnego programu.

1.1 narzędzia linii poleceń

Narzędzia linii poleceń 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

Oczywiście możliwa jest też obsługa tej magistrali z poziomu własnego kodu języka C lub C++ przy użyciu odpowiednich wywołań funkcji bibliotecznych. Przykładowy kod C/C++ demonstrujący użycie komunikacji I2C znajduje się na http://vip.opcode.eu.org/#komunikacja_I2C.

1.3 Zadania

UWAGA: Poniższe zadania stanowią ciąg logiczny – **nie demontuj zbudowanego układu po skończeniu zadania**, będzie potrzebny w następnym.

Zadanie 1.3.1

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 odczyt danych z tego układu z użyciem poleceń `i2cget` / `i2cset`. W tym celu do pinów wejściowych układu I/O podłącz jakiś sygnał (wybrane piny podłącz do 3.3V).

*Wskazówka 1: Numer szyny I^2C zależny jest od sposobu podłączenia do płytki *Pi. Listę wszystkich dostępnych magistral I^2C można uzyskać przy pomocy polecenia* `i2cdetec`.
Wskazówka 2: Adres urządzenia I^2C podany jest w jego dokumentacji. Przy odrobinie szczęścia można zobaczyć wszystkie adresy dostępne na szynie I^2C przy pomocy polecenia `i2cdetec`.

Zadanie 1.3.2

1. Przeanalizuj przykładowy program w C dla systemu Linux realizujący obsługę I2C (bezpośredni link do pliku: http://ln.opcode.eu.org/komunikacja_I2C.c).
2. Pobierz/skopiuj ten kod na *Pi i skompiluj zgodnie z instrukcją w komentarzu.
3. Użyj uzyskany program do komunikacji z układem skonstruowanym w zadaniu 1.3.1 (odczytu stanu jego wejść)

Zadanie 1.3.3

1. Podłącz do magistrali I2C utworzonej w poprzednim zadaniu płytkę z mikrokontrolerem STM32. Układ I/O powinien pozostać na tej magistrali.
2. Zmodyfikuj przykładowy kody STM32 związane z obsługą I2C tak aby mikrokontroler pełnił wyłącznie funkcję układu slave (mnożącego przez dwa).
3. Przetestuj zapis i odczyt wartości oraz działanie układu z użyciem `i2cset` / `i2cget`.

Zadanie 1.3.4

Korzystając z przykładowego kodu związanego z obsługą ADC dla STM32, zmodyfikuj program z zadania 1.3.3, tak aby układ zwracał wartość związaną z napięciem podanym na wejście ADC. *Wskazówka: Dla ułatwienia nie wymagamy aby była to wartość napięcia ani surowa wartość odczytywana z ADC - może to być coś proporcjonalnego do niej.*

Zadanie 1.3.5

Zmodyfikuj przykładowe kody STM32 związane z obsługą I2C tak aby to mikrokontroler (a nie płytka *Pi) 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 I²C vs UART

I²C 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
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.

2.3 Modbus w Linuxie

Obsługę protokołu Modbus RTU po stronie Linuxa może zapewnić program `mbpoll` pozwalający zarówno na odczyt (modbusowe funkcje 1, 2, 3 i 4) jak i zapis (modbusowe funkcje 5 i 6) danych z/do urządzenia modbus TCP lub RTU.

Jeżeli ten zestaw funkcjonalności jest dla nas niewystarczający lub po prostu potrzebujemy zintegrować obsługę *Modbus* z naszym kodem programu możemy użyć bezpośrednio biblioteki *libmodbus* (na której bazuje *mbpoll*).

1. którego używaliśmy już m.in. do programowania mikrokontrolera STM32 i do wypisywania wiadomości "diagnostycznych" przez STM32
2. którego obsługa jest wymagana przez standard dla urządzeń zgodnych z Modbus i jest on dużo popularniejszy niż wariant ASCII

Zadanie 2.3.1

Korzystając z zaprezentowanego na zajęciach przykładowego kodu obsługi Modbus RTU dla STM32 (https://github.com/opcode-eu-org-libs/STM32-ModbusRTU), zmodyfikuj rozwiązanie zadania 1.3.4 tak aby na łączu szeregowym między komputerem PC a mikrokontrolerem stosowany był Modbus RTU. **To zadanie nie wymaga zmian połączeń w zbudowanym układzie!**

3 Zadania dodatkowe

Zadanie 3.0.1

Bazując na rozwiązaniu zadania 1.3.2 dokonaj takiej modyfikacji kodu C aby uzyskać program odczytujący wartość z tego układu I2C i wypisujący informację o wartości poszczególnych jego wejść bez konieczności podawania argumentów linii poleceń – adres szyny, układu i numer rejestru powinny być wpisane w kod (np. z użyciem `#define`).

Zadanie 3.0.2

Zmodyfikuj rozwiązanie 1.3.4 tak aby *Pi mógł odczytać z użyciem I2C pełną (12to bitową) surową wartość z ADC. *Wskazówka: I2C przesyła dane jedno bajtowe. Możesz użyć wartości zapisywanej po I2C jako "adresu rejestru" do identyfikacji którą część liczby 12to bitowej chcemy odczytać.*