Podstawy Systemów Wbudowanych

Wykład 12: Programowanie interfejsów komunikacyjnych

Angelika Tefelska Dariusz Tefelski

Zakład Fizyki Jądrowej, Wydział Fizyki PW

18 maja 2017

O czym będzie wykład...

1. Interfejsy
2. Port szeregowy
3. I2C
4. 1-Wire
5. USB
6. Ethernet / Wifi
7. Bluetooth

Interfejsy komunikacyjne

Interfejsy

Interfejs – w prawie telekomunikacyjnym, układ elektryczny, elektroniczny lub optyczny, z oprogramowaniem lub bez oprogramowania, umożliwiający łączenie, współpracę i wymianę sygnałów o określonej postaci pomiędzy urządzeniami połączonymi za jego pośrednictwem zgodnie z odpowiednią specyfikacją techniczną; np. interfejs programu aplikacyjnego, interfejs radiowy oraz interfejs diagnostyczny. [a]

[a] https://pl.wikipedia.org

Port szeregowy

Port szeregowy

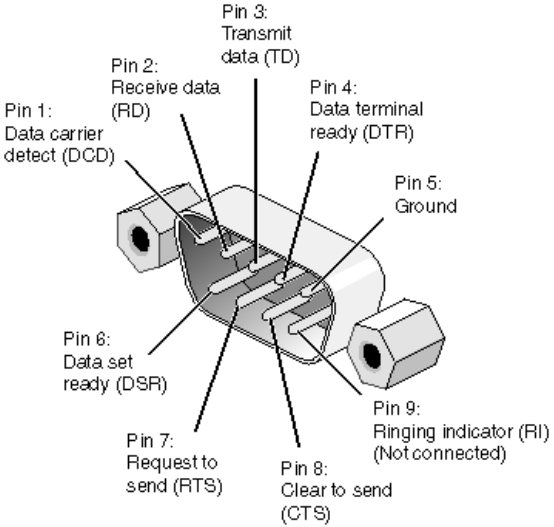
Port szeregowy - jest jednym z najstarszych rozwiązań, pozwalających na transmisję danych na pewną odległość.
Transceiver (urządzenie peryferyjne tranmsitujące dane) nazywany jest UART (Universal Asynchronous Receiver and Transmitter)

- Standard napięć typowo +/- 12V (a nawet do +/- 25V)
- W przypadku mikrokontrolerów w standardzie napięć TTL (1) = +5V, (0) = 0V
- W przypadku mikrokontrolerów w standardzie napięć LVTTL (1) = +3.3V (0) = 0V

Standard RS232

Standard RS232 - opisuje sposób podłączenia urządzenia DTE - *Data Terminal Equipment* z urządzeniem DCE - *Data Circuit-terminating Equipment* albo *Data Communication Equipment*

Rysunek: DB9 typowe złącze dla portu szeregowego ze standardem napięć +/-12V.



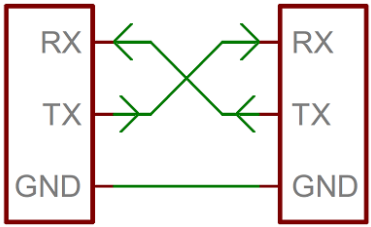
Rysunek: Wyprowadzenia złącza DB9 dla RS232 (pinout)

Zalety:

- Powszechnie znany standard, stosowany przemysłowo
- Tani i łatwo dostępny, dostępne konwertery USB-RS232
- Inne interfejsy szeregowe często wykorzystują UART do komunikacji, np. Bluetooth, GPS
- Wykorzystywany do sterowania różnych urządzeń
- Stosowany jako połączenia terminalowe

Wady:

- Ograniczona długość kabla przesyłowego (ok. 20m) i podatność na zakłócenia wynikająca z napięciowego charakteru interfejsu, stąd wykorzystywane inne standardy np. RS-422, RS-485, w których przesył danych jest prądowy, z wykorzystaniem linii różnicowych.



Rysunek: Minimalne podłączenie interfejsu RS-232

Ramka przesyłowa

- Dane przesyłane przez port szeregowy mogą składać się ze znaków o rozmiarze od 5 do 9 bitów.
- Ramka zaczyna się bitem startu, który ma wartość przeciwną do stanu spoczynkowego. Zazwyczaj w przypadku mikrokontrolerów, stan spoczynkowy jest wysoki, więc, bit startu ma wartość „0”.
- Następnie przesyłane są bity danych (znaku), począwszy od najmniej znaczącego (LSB) do najbardziej znaczącego (MSB).
- Następnie (opcjonalnie) może znajdować się bit parzystości
- Ramkę zakańcza pojedynczy albo podwójny bit stopu, który ma wartość zgodną ze stanem spoczynkowym, czyli stan „1”.
- W przypadku interfejsu RS-232, gdzie sygnały są +/-12V, sytuacja jest odwrócona. Stanem spoczynkowym jest -12V i ten poziom napięcia odpowiada logicznej „1”, natomiast poziom napięcia +12V odpowiada logicznemu „0”. Bit startu, wynosi +12V, więc „0”, a bit stopu -12V, więc „1”.

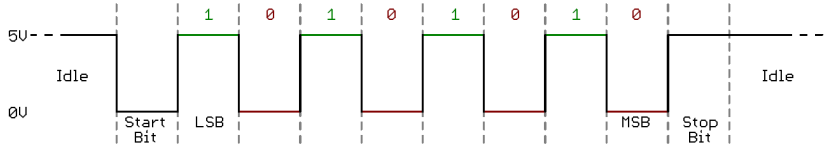
Rysunek: Ramka przesyłowa danych poprzez port szeregowy

Przykład wysyłania ciągu 2 znaków: „O” „K”:

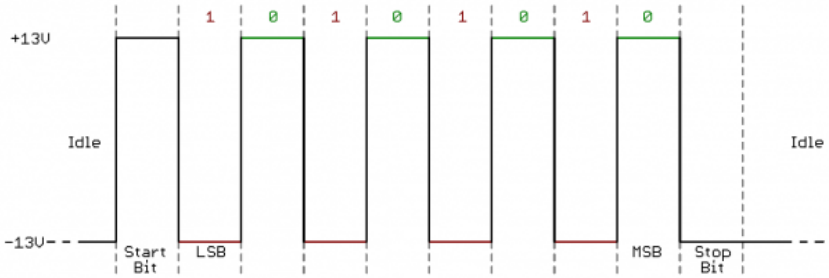
- Znak "O" ma reprezentację bitową: 01001111
- Znak "K" ma reprezentację bitową: 01001011

Rysunek: Znaki „O” i „K” wysyłane przez RS-232

Przykład wysyłania znaków poprzez interfejs mikrokontrolera - standard TTL oraz poprzez typowy interfejs RS-232.



Rysunek: Sygnał na wyjściu TX z mikrokontrolera



Rysunek: Sygnał na wyjściu nadajnika RS-232

Raspberry Pi 3 i port szeregowy

Układ SoC Raspberry PI 3 pozwala użytkownikowi na wykorzystanie 2 sprzętowych interfejsów szeregowych:

- **/dev/ttyS0** - uproszczony interfejs szeregowy (miniuart), który domyślnie jest wyłączony, ale można go włączyć korzystając z narzędzia *raspi-config*, alias **/dev/serial0**
  Po włączeniu dostępny jest na liniach GPIO14 (pin 8) jako **TXD** oraz GPIO15 (pin10) jako **RXD**
- **/dev/ttyAMA0** - pełny interfejs szeregowy, który standardowo jest połączony z interfejsem Bluetooth, alias **/dev/serial1**

Miniuart ma pewne ograniczenia co do możliwości np. ustawienia rozmiaru znaku, a także poważne ograniczenie, które wiąże go z zegarem rdzenia układu SoC. Częstotliwość taktowania rdzenia w tak zaawansowanych układach jest zwykle dynamicznie zmieniana w zależności od obciążenia systemu oraz temperatury układu. Ponieważ w tym przypadku do prawidłowego działania asynchronicznego interfejsu szeregowego potrzebna jest stała częstotliwość rdzenia, ustawiane jest *core_freq=250*. Zmniejsza to troszkę wydajność układu SoC, ale pozwala na stabilne korzystanie z miniuart-a.

Raspberry Pi 3 i port szeregowy

Mapowanie portów szeregowych

Istnieje możliwość zamiany wyprowadzeń portu szeregowego w układzie SoC - poprzez konfigurację programową. Można np. do wyprowadzeń GPIO14 i GPIO15 podłączyć pełny port szeregowy **/dev/ttyAMA0** i wykorzystywać go normalnie interfejs szeregowy, należy wtedy zastanowić się, co z interfejsem Bluetooth. Możliwośći są 2:

- Rezygnacja z wykorzystania Bluetooth, rezygnacja z miniuarta. Plusem będzie dynamiczna konfiguracja częstotliwości rdzenia
- Korzystanie z Bluetooth podłączonego do miniuart-a. W tym momencie Bluetooth będzie podlegał ograniczeniom co do prędkości transmisji szeregowej.

Terminal

Domyślnie port szeregowy po włączeniu w Raspberry Pi 3 jest ustawiony jako **urządzenie terminalowe**. Włączone jest także przekazywanie komunikatów z jądra w czasie uruchamiania systemu na port szeregowy. Parametry połączenia, to: prędkość: 115200 baud, znak: 8 bitów, bit stopu: 1, bit parzystości: brak
Przydatne jest to np. w celach diagnostycznych gdy Raspberry Pi nie jest podłączone do wyświetlacza oraz gdy nie ma dostępu z sieci poprzez SSH.

Konfiguracja portu szeregowego

- Włączenie/wyłączenie portu szeregowego poprzez **raspi-config**, ewentualnie ręczne wpisanie do pliku */boot/config.txt* linijki **enable_uart=1**.
- Wyłączenie interfejsu Bluetooth i przekazanie pełnego portu szeregowego na piny GPIO14 i GPIO15 - wpisanie do pliku */boot/config.txt* linijki **dtoverlay=pi3-disable-bt**
- Zamienienie miejscami portów szeregowych - w pełni sprawny na piny GPIO14 i GPIO15, natomiast miniuart do Bluetooth. Ogranicza to zastosowanie Bluetooth do niskich prędkości - wpisanie do pliku */boot/config.txt* linijki **dtoverlay=pi3-miniuart-bt**

Stosowanie portów w oprogramowaniu

Wykorzystujac porty we własnym oprogramowaniu, zaleca się, aby stosować przygotowane aliasy: /dev/serial0 oraz /dev/serial1. Domyślnie aliasy wskazują na urządzenia w postaci plików: /dev/serial0 -> /dev/ttyS0, /dev/serial1 -> /dev/ttyAMA0

Wyłączenie trybu terminalowego i komunikatów z jądra systemu

Wyłączenie trybu terminalowego

- Należy w pliku **/boot/cmdline.txt** usunąć opcję **console=serial0,115200**
- Wyłączyć **serwis** w **systemd** odpowiedzialny za usługi konsolowe:

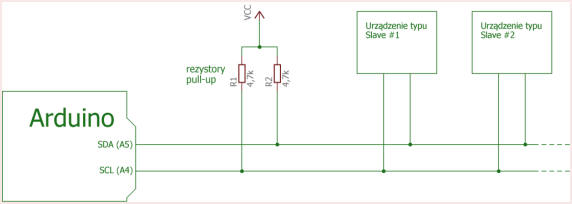
```
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service
```

- Zrestartować Raspberry Pi

```
sudo reboot
```

Magistrala I2C

- I2C może obsługiwać do 127 urządzeń przy 7-bitowym adresowaniu typu slave (urządzenia podrzędne - wysyłające dane np. jakiś czujnik). Do magistrali może być podłączone więcej niż jedno urządzeń typu master (nadrzędne - odbierające dane np. arduino).

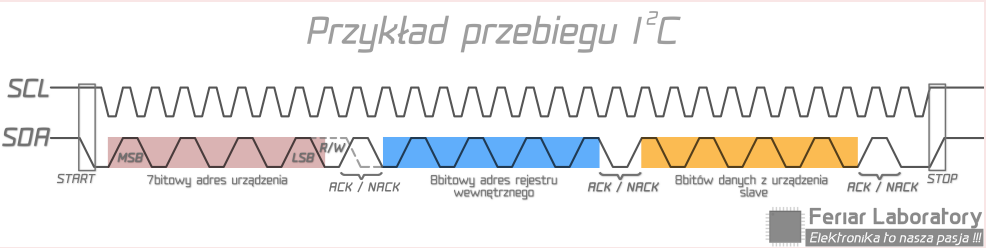


Rysunek: Schemat podłączenia urządzeń typu slave do arduino. [a]

- I2C do transmisji wykorzystuje dwie dwukierunkowe linie: SDA (Serial Data Line - linia danych) i SCL (Serial Clock Line - linia zegara). Obydwie linie są na stałe podciągnięte do źródła zasilania poprzez rezystory podciągające.
- Oporność rezystorów podciągających wpływa na maksymalną osiągalną prędkość transmisji danych. Zakres, w obrębie którego powinna być wybrana wartość rezystora wynosi: (1.8, 47) $k\Omega$. Jednak im mniejsza wartość rezystora tym możliwa szybsza transmisja danych (bardziej strome zbocza sygnału) (np. dla $2k\Omega$ wynosi ok 400kb/s) ale również większy pobór energii. Z tego powodu dobrym kompromisem jest 4.7 $k\Omega$.

Magistrala I2C

Zasada transmisji danych:

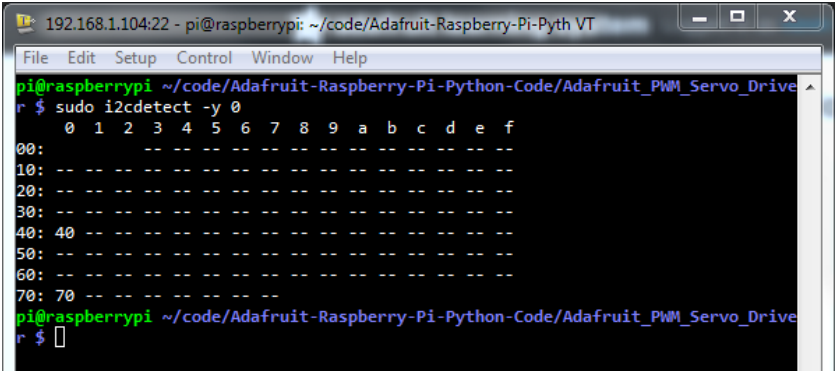


1. Wysłanie bitu startu czyli linia SDA przechodzi ze stanu wysokiego na niski gdy na linii SCL jest stan wysoki.
2. Wysłanie 7 bitów z adresem urządzenia oraz jednego bitu R/W informującego czy urządzenie typu master chce dokonać odczytu czy zapisu (odczyt=1, zapis=0).
3. Urządzenie typu slave zwraca bit ACK (Acknowledge). Jeśli adres został poprawnie wysłany i rozpoznany przez urządzenie typu slave, to poda ono stan niski na linię SDA. W przeciwnym wypadku procedura zostanie przerwania (stan wysoki - NACK).
4. Wysłanie adresu wewnętrznego adresu rejestru, który ma zwracać daną wartość z układu np. gdy chcemy odczytać godzinę to dla zegara RTC będzie to adres rejestru odpowiedzialny za godzinę.
5. Wysłanie bitu ACK jeśli adres został rozpoznany.
6. Wysłanie 8bitów danych.
7. Kończenie procesu wysyłania danych poprzez ustawienie bitu ACK i wysłanie bitu STOP.

Przydatne narzędzia do obsługi I2C na Raspberry Pi

1. Magistrala I2C domyślnie jest wyłączona. Aby ją włączyć należy:
   - Wpisać w terminalu: ***sudo raspi-config***
   - Wybrać opcję ***5 Interfacing options***
   - Następnie wybrać: ***P5 I2C*** i zmienić na ***enable***.
   - W okienku wyskoczy informacja czy zgadzamy się na automatyczne załadowanie jądra. Należy zaznaczyć *yes*
2. Przydatnym narzędziem jest ***i2c-tools***, które umożliwia komunikację z magistralą z poziomu terminala. Aby go zainstalować wystarczy wpisać w terminalu: ***sudo apt-get install -y i2c-tools***. Najważniejsze komendy to:
   - ***i2cdetect -y 1*** - skanuje wszystkie możliwe adresy magistrali i zwraca informacje, czy pod którymś adresem wyryto urządzenie.
   - ***i2cget <bus> <chip> <register>*** - zwraca wartość z wskazanego rejestru urządzenia podłączonego do danej szyny magistrali np. ***i2cget 0 0x2d 0x10*** zwraca wartość z rejestru 0x10 czujnika o adresie 0x2d podłączonego do szyny o adresie 0.
   - ***i2cset <bus> <chip> <register> <value>*** - wysyła wartość do danego rejestru czujnika podłączonego do danej szyny magistrali.
3. I2C można obsłużyć korzystając z trzech bibliotek: ***quick2wire***, ***wiringPiI2C*** oraz ***SMBUS***.

i2cdetect -y 1



Raspberrry Pi 3 posiada 2 magistrale i2c (/dev/i2c-0 i /dev/i2c-1). Magistrala wyprowadzona na złącze 40-pin'owe dla użytkownika (pin nr 3 i nr 5) to /dev/i2c-1, dlatego w narzędziach i2c-tools: i2cdetect -y 1, podawać należy magistralę nr 1. Domyślnie magistrala 0 jest wyłączona. Przeznaczona jest ona do odczytu pamięci eeprom, które mogą znajdować się na dołączonych modułach „HAT", dzięki którym przy starcie systemu zostaną skonfigurowane dla nich sterowniki. Piny przeznaczone do transmisji na magistrali 0, to piny złącza nr 27 oraz 28.

Quick2wire - przykład

```
import quick2wire.i2c as i2c

i2c_bus=i2c.I2CMaster(0) #otwarcie komunikacji

bajt_danych=0b01111000 #dane wysylane do przetwornika
address=0x68 #adres pierwszego przetwornika

#wyslanie danych do przetwornika
i2c_bus.transaction(i2c.writing_bytes(address,bajt_danych)
    ↪ )

#odczytanie 3 bajtow danych p,d - pierwszy i drugi bit
    ↪ danych, c - bajt konfiguracyjny
p,d,c = i2c_bus.transaction(i2c.reading(address,3))[0]
```

WiringPiI2C

Funkcje biblioteki WiringPiI2C:

- **wiringPiI2CSetup(int adres urządzenia)** - funkcja do inicjalizacji połączenia z danym urządzeniem przez magistralę I2C
- **wiringPiI2CRead(int magistrala i2C, int adres rejestru)** - funkcja do odczytu danych z rejestru urządzenia
- **int wiringPiI2CWrite (int magistrala i2C, int adres rejestru, int data)** - funkcja do wysyłania danych do danego rejestru

```
#include <iostream>
#include <errno.h>
#include <wiringPiI2C.h>

using namespace std;

int main()
{
    int fd, result;

    fd = wiringPiI2CSetup(0x60);

    result = wiringPiI2CWriteReg8(fd, 0x40,  0xff);
}
```

SMBUS - przykład

```
import smbus
import time

# define I2C bus number
BUS_NUMBER = 1

# define device address
DEVICE_ADDR = 0x48

bus = smbus.SMBus(BUS_NUMBER)

bus.write_byte(DEVICE_ADDR,0x00)

while True:
    print bus.read_byte(DEVICE_ADDR)
```

1-Wire

- Interfejs 1-Wire został zaprojektowany przez Dallas Semiconductor. Jego zaletą jest minimalizacja użytych linii do minimum (czyli jednej) oraz pasożytnicze zasilanie. Natomiast jest to magistrala wolniejsza od I2C (16kb/s).
- Ograniczenia dotyczące magistrali 1-Wire związane są z parametrami elektrycznymi linii.
- Koncepcja działania magistrali jest zbliżona do I2C. Główna różnica polega do podejściu w definiowaniu logicznych wartości i bitu startu oraz ACK. Te bity są definiowane przez czas trwania sygnału niskiego (zwarcia linii do masy).
- Sygnał reset następuje gdy urządzenie master wyśle stan niski przez 480us(zwarcie linii do masy) a zgłoszenie swojej obecności przez urządzenie slave gdy wyśle stan niski przez 240us.
- Gdy urządzenie typu master chce wysłać wartość 1 to ustawia stan niski o czasie trwania 1-15us a następnie rozwiera linię na 60us. Logiczne zero odpowiada zwarciu linii na czas 60us.
- Urządzenie slave jak chce wysłać wartość 1 to nic nie robi a jak 0 to zwiera linię na 60us.
- Przed odbiorem każdego bitu master wysyła stan niski o długości 1-15us (zwiera linię) a następnie powraca do stanu wysokiego.

Obsługa 1-Wire pod Raspberry Pi

- Do komunikacji z interfejsem 1-wire przydatne są moduły jądra w1-gpio oraz w1-therm. Aby z nich skorzystać należy podłączyć 1-wire do pinu GPIO4.
- W celu załadowania modułów należy skorzystać z komend: **sudo modprobe w1-gpio** oraz **sudo modprobe w1-therm**
- Listę dostępnych czujników można podejrzeć poprzez: **ls /sys/bus/w1/devices** gdzie zostaną nam zwrócone informacje w postacji 28-xxx gdzie xxx to unikatowy identyfikator czujnika.
- **Przykład:** Odczytanie temperatury z czujnika DS18B20 (Linia zaczynająca się od >>> oznacza komendy wprowadzone w terminalu):

```
>>> ls /sys/bus/w1/devices
28-0000054d332a  w1_bus_master1
>>> cat /sys/bus/w1/devices/28-0000054d332a/
    ↪ w1_slave
6d 01 4b 46 7f ff 03 10 70 : crc=70 YES
6d 01 4b 46 7f ff 03 10 70 t=22812
```

Czyli temperatura wynosi $22,812^oC$

Obsługa 1-Wire pod Raspberry Pi

- Przedstawiony przykład odczytu temperatury na poprzednim slajdzie nie jest zbyt wygodny. Najprościej byłoby napisać skrypt w bash-u, który zwracałby temperaturę:

```
#!/bin/sh

sudo modprobe w1-gpio
sudo modprobe w1-therm

TEMP=`cat /sys/bus/w1/devices/28-0000054d332a/
    ↪ w1_slave | grep t= | cut -d "=" -f 2`

TEMP=`echo "scale=1; $TEMP/1000" | bc`
echo "$TEMP C"
```

- Jednym z najbardziej znanych obecnie interfejsów szeregowych jest USB (Universal Serial Bus).
- W Raspberry PI 3 mamy wyprowadzone 4 gniazda USB w standardzie USB 2.0
- Do układu SoC podłączony przez USB jest układ spełniający funkcje HUB-u (stąd 4 wyjścia) a także karta sieciowa Ethernet.

- Raspberry PI 3 posiada gniazdo sieciowe Ethernet w standardzie 100 Mbit, więc nadaje się do podłączenia i transferu danych w standardowych sieciach - interfejs **eth0**
- Raspberry PI 3 posiada także wbudowany interfejs WIFI -150 Mbit - interfejs **wlan0**
- Konfiguracja odbywa się poprzez dostępne w dystrybucji Linux-a narzędzia. Możliwe jest zapisanie np. statycznej konfiguracji w pliku /etc/networking/interfaces. Podłączenie do Access Point-a (AP) w WIFI możliwe jest poprzez konfigurację w pliku /etc/wpa_supplicant/wpa_supplicant.conf
- Możliwe jest także wykorzystanie narzędzi graficznych np. network-manager, albo konsolowych - tekstowych np. wicd

Konfiguracja bluetooth pod Raspberry Pi

W celu podłączenia jakiegoś urządzenia np. klawiatury przez bluetooth trzeba go najpierw skonfigurować. W tym celu należy wykonać następujące kroki:

- **sudo apt-get install bluetooth bluez-utils blueman**
- **hcitool scan** - wyszukujemy urządzenie bluetooth i kopiujemy jego adres np: 60:FB:42:08:1C:80.
- **bluez-simple-agent hci0 60:FB:42:08:1C:80**
- **bluez-test-device trusted 60:FB:42:08:1C:80 yes** - dodajemy nasze urządzenie do zaufanych.
- **bluez-test-input connect 60:FB:42:08:1C:80** - łączymy się.

THAT'S ALL FOR TODAY....
ANY QUESTION??