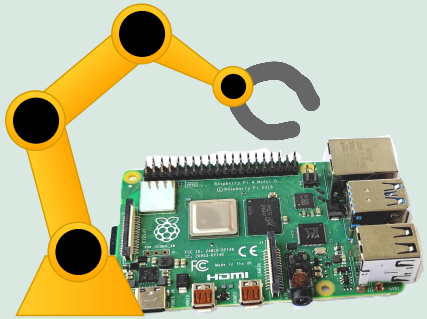
SKPS
Systemy komputerowe w sterowaniu i pomiarach
Część WZ, Wykład 2 - Raspberry Pi w laboratorium



dr hab. inż. Wojciech M. Zabołotny, prof. uczelni

Wydział Elektroniki i Technik Informacyjnych PW, pok. 225,
wojciech.zabolotny@pw.edu.pl (proszę dodać [SKPS] w temacie)

Nasze środowisko laboratoryjne

- Zestaw z komputerem jednopłytkowym Raspberry Pi 4
- Płytka rozszerzeń
- Zestaw czujników i modułów I/O
- Przewody połączeniowe
- Płytki prototypowe

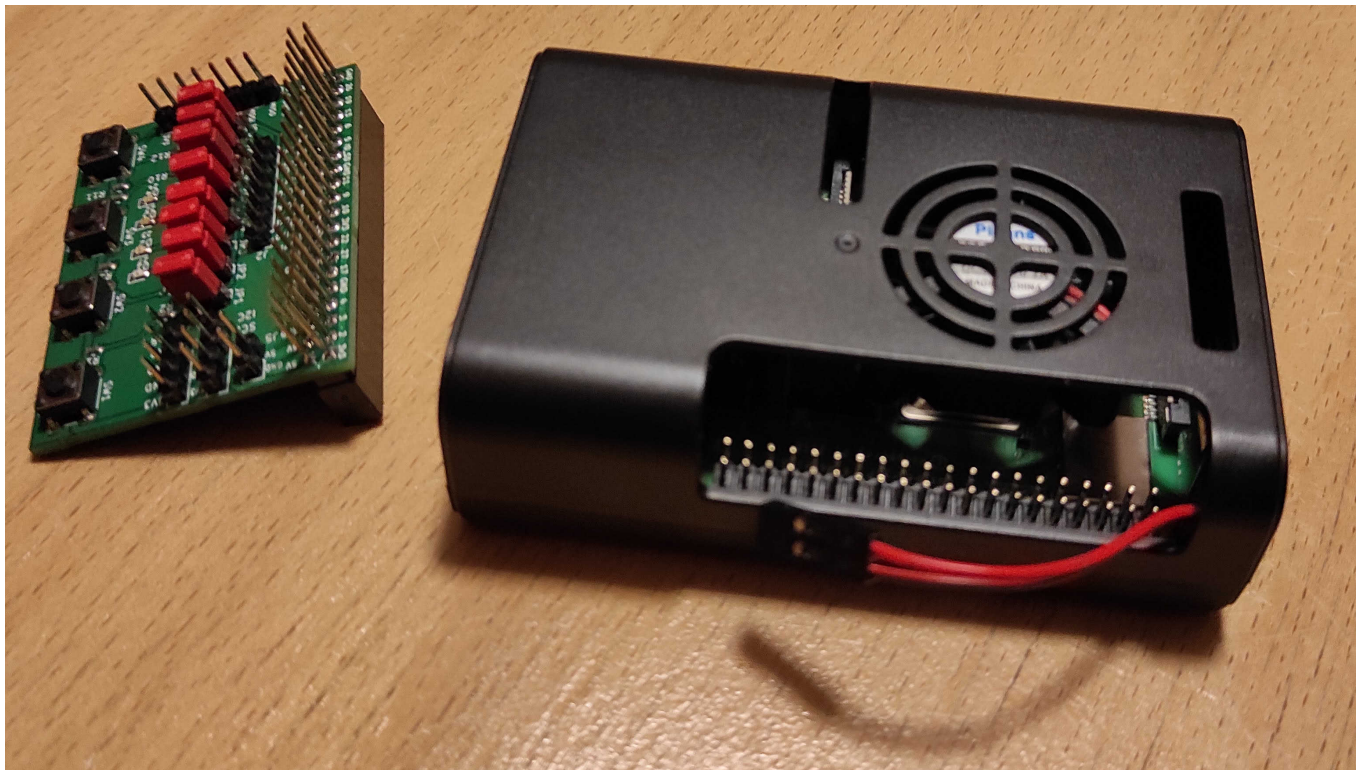
Jak będziemy korzystać z zestawu

- Raspberry Pi jest dość “potężnym” komputerem jednopłytkowym, będziemy chcieli jednak używać go w charakterze systemu wbudowanego.
- Płytki laboratoryjne będą używane przez różne zespoły w różnych terminach.
- Co wynika z powyższych wymagań?
  - Nie używamy standardowej dystrybucji Raspbian lub analogicznej
  - Używamy systemu Linux tworzonego przez nas na nasze potrzeby za pomocą środowiska Buildroot. Ewentualnie możemy skorzystać z OpenWRT z naszymi modyfikacjami.
  - Z systemem komunikujemy się za pomocą interfejsu UART (konsoli szeregowej). Jeśli mamy już działającą sieć, możemy użyć dostępu przez sieć (np. SSH).

Bezpieczeństwo korzystania z zestawu

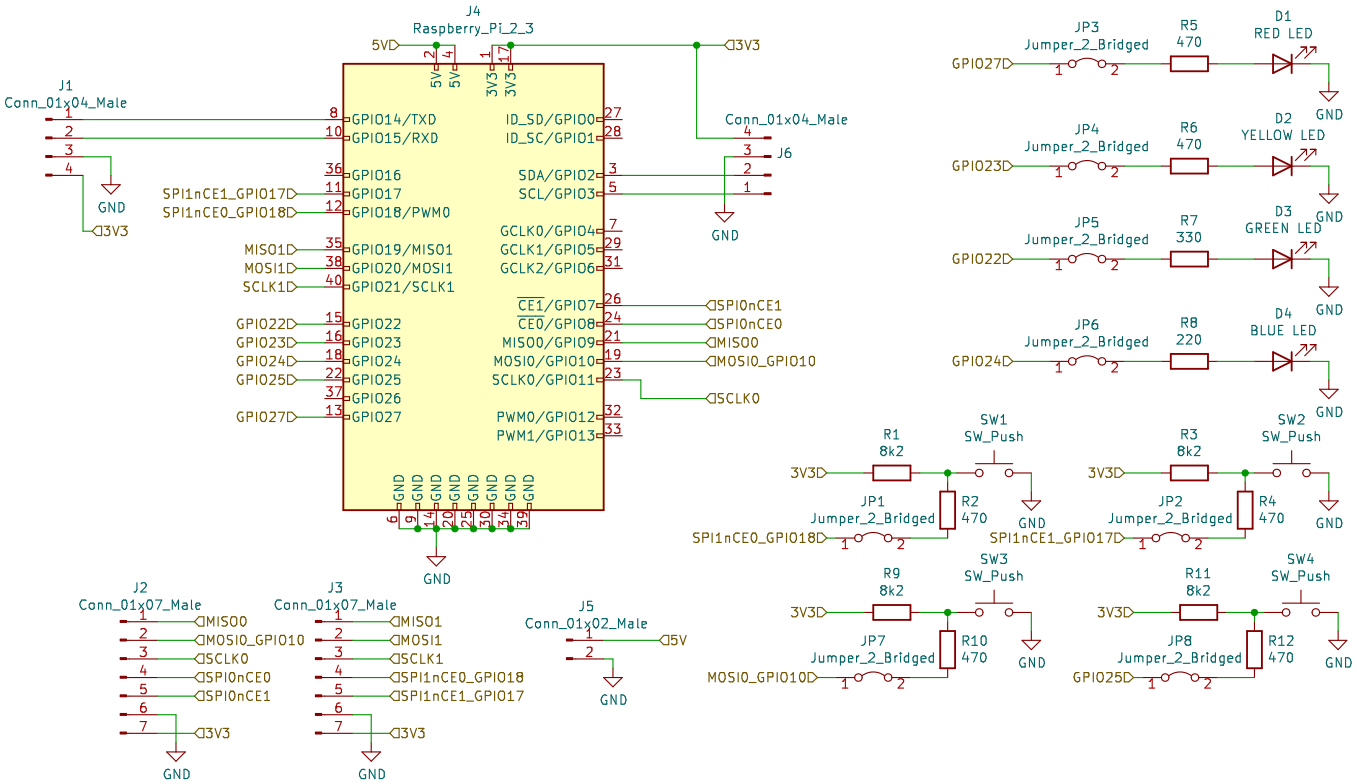
- W laboratorium będziecie Państwo mieć możliwość pracy bezpośrednio ze sprzętem. Daje to duże możliwości, ale stwarza też pewne zagrożenia dla sprzętu i dla Państwa.
  - Wszelkie połączenia powinny być wykonane starannie i przemyślane. Koniecznie należy je zweryfikować przed włączeniem zasilania.
  - Zwracajmy uwagę na poziomy napięć logicznych i napięcia zasilające. Raspberry Pi nie toleruje podawania napięć ujemnych lub powyżej 3,3V na swoje wyprowadzenia (tak naprawdę istnieje pewien zapas bezpieczeństwa, ale lepiej na niego nie liczyć).
  - Upewnijmy się, jak zapewnić, żeby moduły, które chcemy podłączyć do naszej płytki pracowały z poziomami logicznymi od 0 do 3,3V.
  - Zasilacz do którego podłączamy RPi nie ma przewodu ochronnego (GND, masy). Ze względu na obecność kondensatorów przeciwzakłóceniowych w zasilaczu, może występować pewna upływność między przewodem “fazy” sieci energetycznej a masą RPi. Żeby uniknąć uszkodzenia RPi lub komputera, albo nieprzyjemnego wstrząsu elektrycznego, pamiętajmy, żeby podłączyć masę RPi do masy komputera PC (np. przez adapter UART) zanim podłączymy zasilacz RPi do sieci.

Płytka rozszerzeń



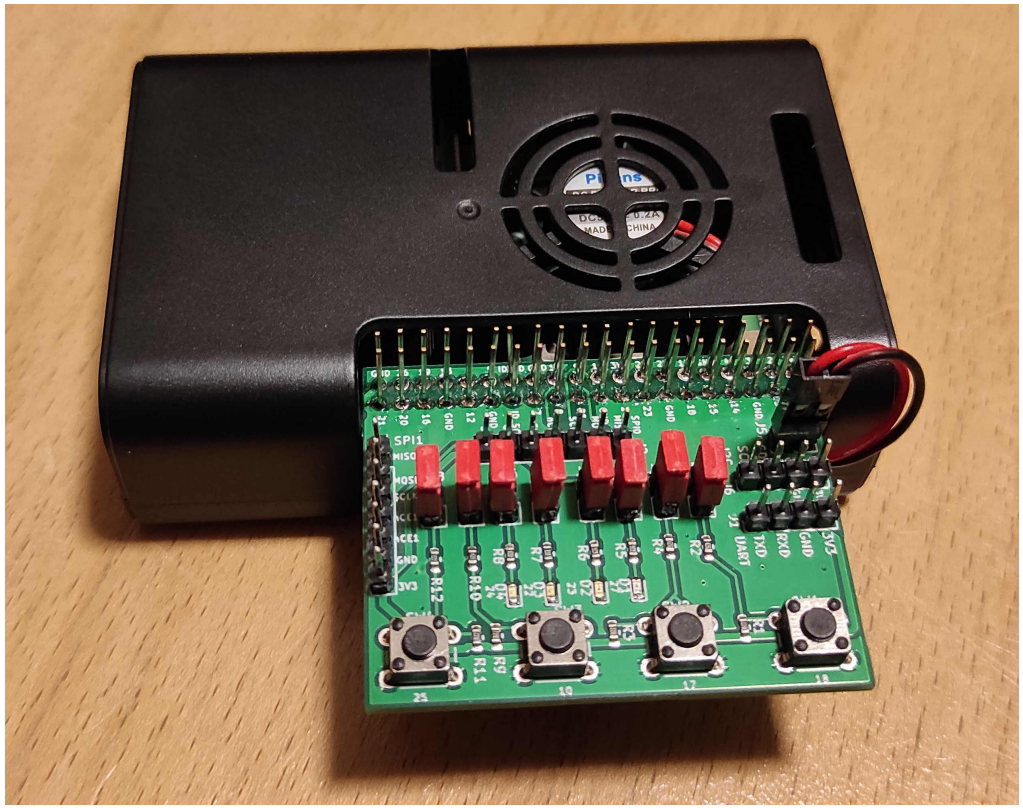
- Oprócz komputerka RPi w obudowie, istotnym elementem zestawu jest płytka rozszerzeń. Jest ona wyposażona w cztery przyciski i cztery diody LED z opornikami.
- Projekt płytki jest dostępny w publicznym repozytorium.

Płytka rozszerzeń - schemat



- Płytka rozszerzeń umożliwia dostęp do pełnego złącza RPi. Oprócz tego dostępne są złącza SPI0, SPI1, UART i I2C. Niektóre złącza używane do przycisków lub diod LED, są też wykorzystywane do innych interfejsów. Zwora pozwala odłączyć przycisk, lub diodę LED.

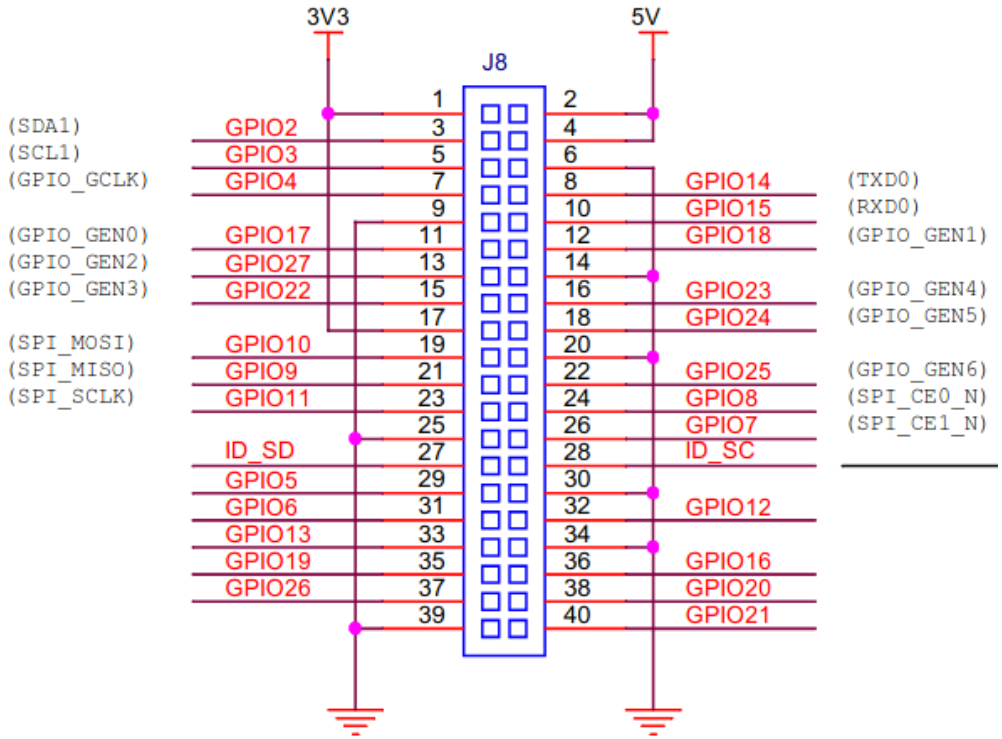
Płytka rozszerzeń podłączona do RPi



- Raspberry Pi z podłączoną płytką rozszerzeń.
- Proszę zwrócić uwagę na podłączenie wentylatorka.

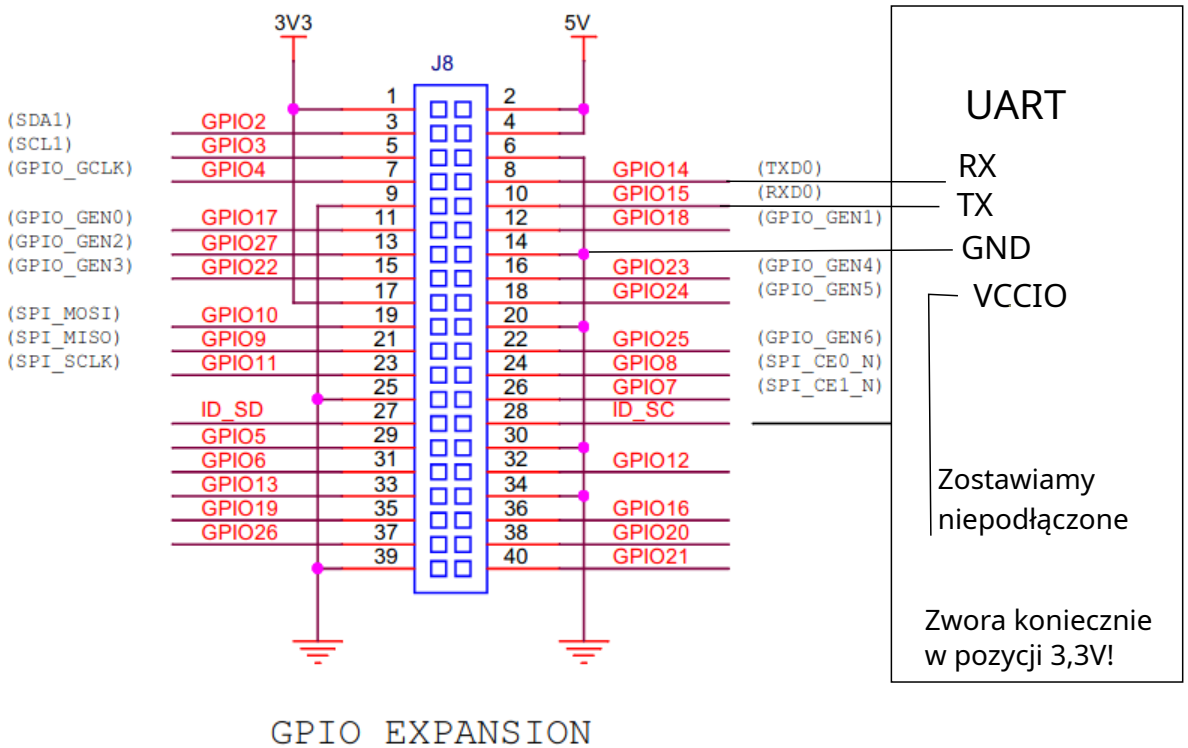
Co musimy podłączyć do zestawu?

- Na pierwszych zajęciach, na pewno będziemy musieli podłączyć
  - Adapter UART-USB
  - Zasilacz
  - Kabel sieci Ethernet
- Do połączeń wykorzystujemy gniazda RPi, złącza na płytce rozszerzeń, albo złącze rozszerzeń. Opis wyprowadzeń złącza (z oficjalnego schematu) jest na rysunku.



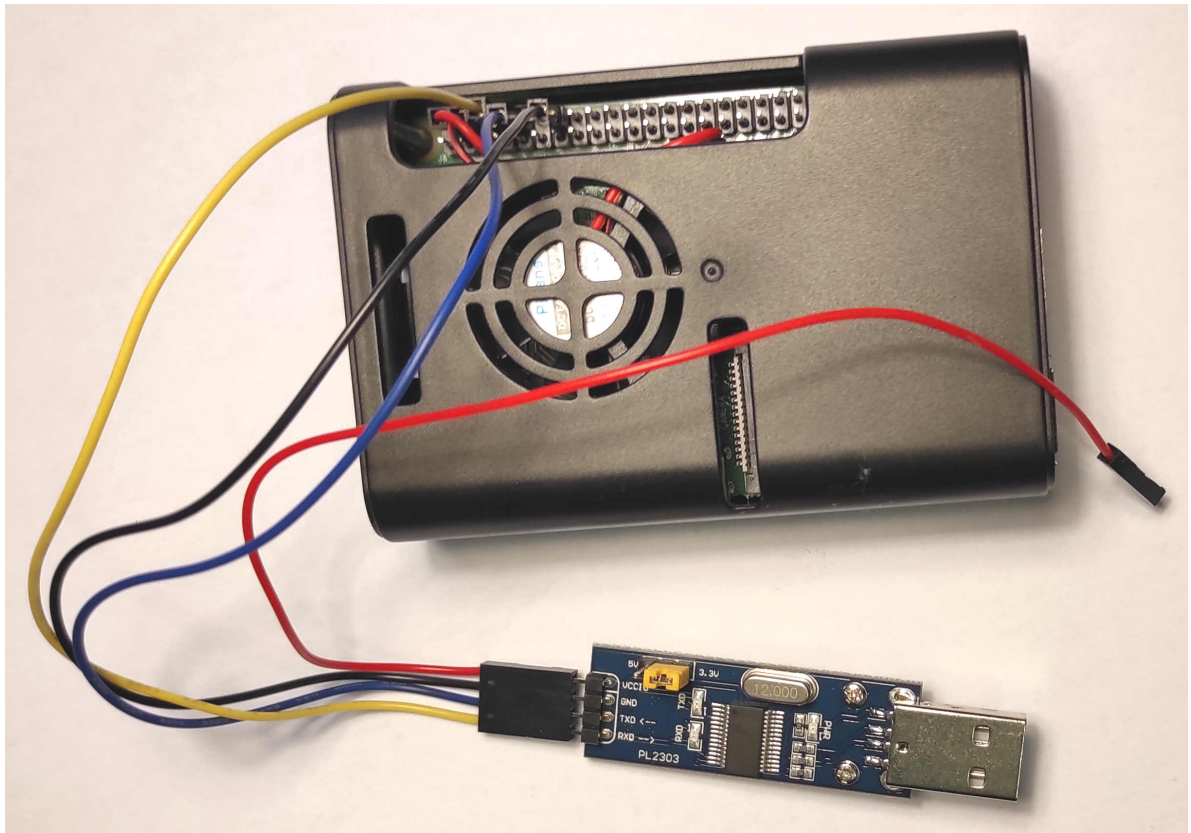
Podłączenie UART

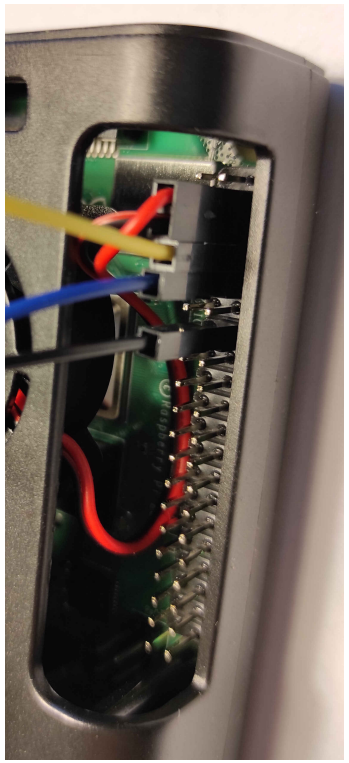
- Podłączenie UART musimy zrealizować samodzielnie. Należy starannie upewnić się, że łączymy właściwe wyprowadzenia. Poniżej połączenia bez korzystania z płytki rozszerzeń.



Podłączenie UART - zdjęcia

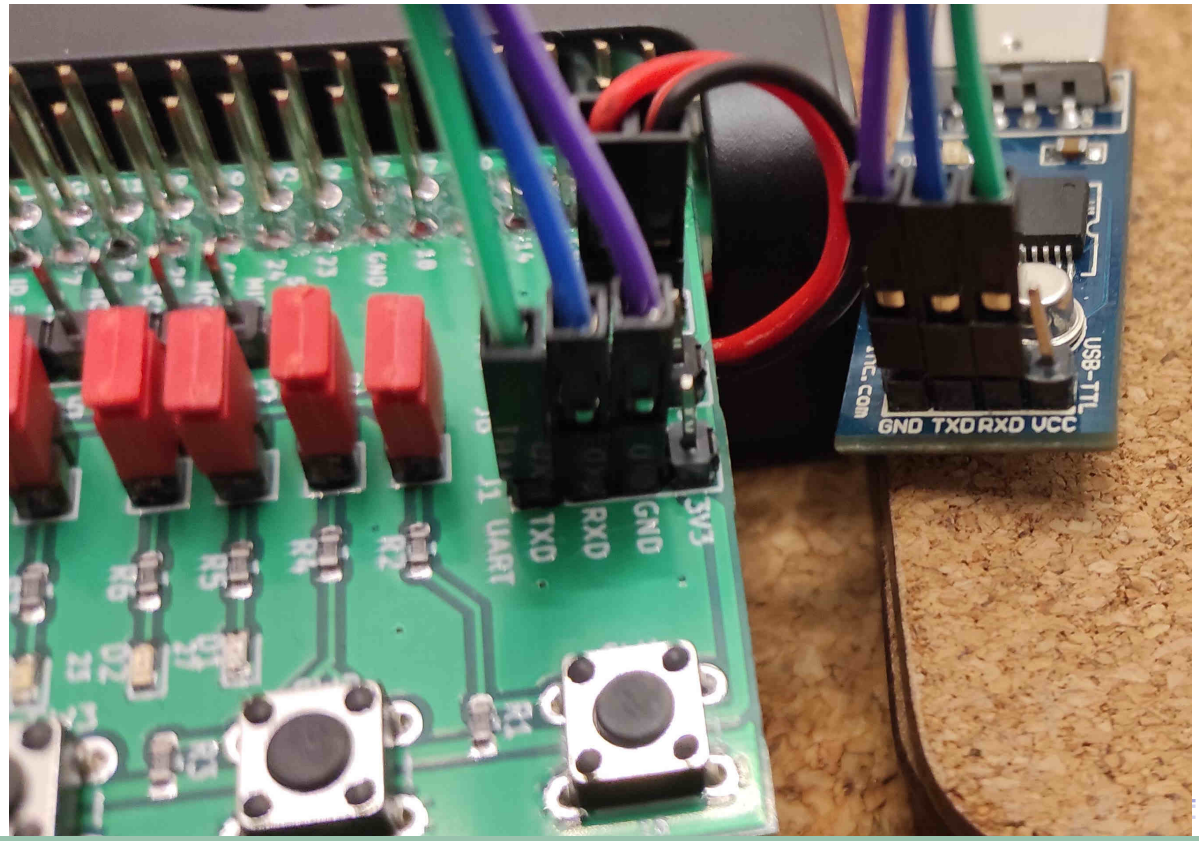
- Poniższe zdjęcia pokazują podłączenie adaptera UART-USB bez płytki rozszerzeń “w naturze”.





Podłączenie UART - z płytką rozszerzeń

- Połączenie adaptera UART-USB z płytką rozszerzeń jest prostsze. Łączymy ze sobą wyprowadzenia "GND"(masę), a wyprowadzenia RXD i TXD łączymy “na krzyż”. Inne wyprowadzenia (5V, 3,3V, Vcc itp.) pozostawiamy niepodłączone.
  Uwaga! Adapter używany w laboratorium może różnić się od tego na zdjęciu



Budowanie Linuxa przy pomocy Buildroota dla RPi 4

- Kompilacja będzie podobna do tego, co na pierwszym wykładzie robiliśmy dla maszyny emulowanej.
- Domyślna konfiguracja:
  `make raspberrypi4_64_defconfig` – 64–bitowa
  `make raspberrypi4_defconfig` – 32–bitowa
- Warto zmienić zestaw narzędzi
- Kompilacja, tak jak poprzednio przez `make`

Co dostajemy, jako wynik kompilacji?

- W `output/images` dostajemy
  - `Image` – jądro naszego systemu,
  - `bcm2711-rpi-4-b.dtb` – "drzewo urządzeń" (DT) naszego systemu,
  - `rootfs.*` – obraz głównego systemu plików (rootfs) naszego systemu w różnych formatach,
  - `sdcard.img` – obraz karty SD naszego systemu.
- Jak możemy wykorzystać wynik kompilacji?

Jak zainstalować stworzony obraz?

- Wygenerowany przez BR obraz karty można bezpośrednio wgrać na kartę. Umieszczamy kartę w czytniku, podłączamy do PC i ustalamy jako jakie urządzenie jest widoczna. Przypuśćmy, że jest to `/dev/sdX`.
- Do przeniesienia obrazu na kartę, możemy użyć polecenia `dd`:
  ```
  dd if=sdcard.img of=/dev/sdX bs=4096
  ```
- Tak wgrany obraz można uruchomić w RPi.
- Możemy też użyć obrazu karty jako wirtualnego urządzenia blokowego:
  ```
  losetup -P -f sdcard.img; losetup -l
  ```
- Następnie możemy obejrzeć partycje i ich zawartość.

Co zawiera karta SD?

- Na karcie stworzone są dwie partycje. Pierwsza sformatowana systemem plików VFAT, zawierająca znane nam już pliki: `bcm2711-rpi-4-b.dtb` i `Image` oraz pliki związane ze standardowym programem ładującym RPi: `config.txt`, `cmdline.txt`, `fixup.dat`, `start.elf` i katalog `overlays`.
- Pliki `config.txt` i `cmdline.txt` możemy modyfikować, wpływając na sposób ładowania systemu.
- Katalog `overlays` zawiera nakładki na drzewo urządzeń. Możemy je wybierać w zbiorze `config.txt`.

Drobne optymalizacje

- Gdy przeanalizujemy tablicę partycji karty SD, to zobaczymy, że nie cała karta jest wykorzystana.
- Możemy zwiększyć rozmiar drugiej partycji tak, aby wypełniła całą kartę. Nie powinniśmy tylko zmienić położenia jej początku.
- Następnie poleceniem `resize2fs` możemy zwiększyć rozmiar systemu plików tak, żeby zajmował całą partycję.

Czy to właściwy sposób użycia karty SD w laboratorium?

- Metoda używająca wygenerowanego obrazu karty dobrze nadaje się do instalacji finalnego, gotowego obrazu systemu.
- Podczas prac rozwojowych, to metoda ma jednak poważną wadę – wymaga ciągłego przekładania karty SD. między RPi, a czytnikiem. Może to spowodować szybkie zużycie gniazd i karty.
- Czy możemy tego uniknąć?

System “ratunkowy”

- Karta SD może pomieścić więcej niż jeden system.
- Moglibyśmy więc mieć stabilną wersję Linuxa zainstalowaną jako system “ratunkowy” i za jej pomocą przeinstalowywać nasz rozwijany system.
- Teoretycznie moglibyśmy przełączać ładowany system modyfikując zbiory `config.txt` i `cmdline.txt`.
- Niestety, takie proste rozwiązanie zawiedzie, gdy wybrany system będzie działać niepoprawnie (możemy nie być w stanie zamontować partycji VFAT i dokonać niezbędnej edycji zbiorów `config.txt` i `cmdline.txt`.
- W przypadku większości systemów wbudowanych dobrym rozwiązaniem jest użycie specjalnego programu ładującego - bootloadera, który pozwoli nam wybrać ładowany system np. za pomocą konsoli szeregowej (lub przełącznika, jeśli jest podłączony do płytki).
- Bootloaderem dobrze nadającym się do tego celu jest U-Boot. Może on być zbudowany przy pomocy Buildroota (Bootloaders->U-Boot).
- Niestety, firmware Raspberry Pi nie zapewnia poprawnej współpracy z loaderem U-Boot. Oryginalny bootloader musi załadować i zmodyfikować drzewo urządzeń, zanim zostanie uruchomiony U-Boot. Nie pozwala to na swobodny wybór drzewa urządzeń z poziomu U-Boota.

Możliwe realizacje systemu “ratunkowego” w.1

- **W przypadku RPi, oba systemy muszą używać tego samego drzewa urządzeń ładowanego przez oryginalny firmware.**
- Na partycji VFAT karty umieszczamy jądro naszego systemu i jądro systemu ratunkowego pod różnymi nazwami.
- Poza partycją VFAT, karta musi zawierać dodatkowe partycje: na główny system plików systemu ratunkowego (jeśli system ratunkowy nie używa ramdysku startowego) i na główny system plików naszego systemu (kolejne partycje będą się nazywać `/dev/mmcblk0p2`, `/dev/mmcblk0p3` ...).
- W zbiorze `config.txt` każemy załadować `u-boot.bin` zamiast jądra systemu.
- U-Boot skonfigurowany jest tak, żeby po kilku sekundach oczekiwania domyślnie ładował nasz system.
- Jeśli podczas oczekiwania, zostanie odebrany jakiś znak z konsoli szeregowej, to U-Boot przechodzi w tryb interaktywny.
- Jeśli w środowisku U-Boota jest dodana definicja komend powodujących uruchomienie systemu ratunkowego (np. `rescue`), to wpisanie komendy `run rescue` spowoduje uruchomienie systemu ratunkowego.

System ratunkowy w.1

- Instalacja, lub reinstalacja naszego systemu jest możliwa po uruchomieniu systemu ratunkowego.
- Możemy wówczas na partycję VFAT skopiować nowe jądro naszego systemu.
- Nowy obraz naszego głównego systemu plików możemy wgrać poleceniem `dd` na właściwą partycję. Następnie, poleceniem `resize2fs` możemy go powiększyć do obszaru całej partycji.
- Polecenie `dd` może zapisywać dane otrzymane przez standardowe wejście. Możemy dzięki temu odbierać dane przez sieć i od razu zapisywać je na partycję (jest to niezbędne, gdy cały obraz systemu plików jest zbyt duży, by zmieścić się w pamięci lub na partycji “ratunkowej”).
- Taka technika pracy nadal wymaga modyfikacji karty SD po wygenerowaniu kolejnej wersji naszego systemu. Może i tego da się uniknąć?

Możliwe realizacje systemu “ratunkowego” w.2

- **W tej wersji systemy “ratunkowy” i “normalny” mogą używać różnych drzew urządzeń, ale do płytki musi być podłączony przełącznik**
- Korzystamy ze specjalnych funkcji os_prefix i GPIO Filter, których możemy użyć w pliku `config.txt` i które są obsługiwane przez oryginalny firmware RPi.
- Wykorzystując te funkcje, możemy umieścić: jądro systemu, drzewo urządzeń i zbiór `cmdline.txt` oddzielnie dla każdego systemu w oddzielnych katalogach partycji VFAT karty SD.

Praktyczna realizacja systemu “ratunkowego” w.2

- Do wyprowadzenia GPIO 25 mamy podłączony przełącznik, zwierający je (przez rezystor 100 Ω) do masy.
- Na końcu zbioru `config.txt` umieszczamy komendy podane w prawej kolumnie.
- Na partycji VFAT karty SD tworzymy dwa katalogi: `rescue` i `user`.
- Jądro, drzewo urządzeń i zbiór `cmdline.txt` dla systemu ratunkowego wgrywamy do katalogu `rescue`, a dla systemu użytkownika do katalogu `user`.
- Możemy musieć utworzyć dodatkowe partycje na główne systemy plików (rootfs) obu tych systemów. Możemy musieć zmodyfikować zbiory `cmdline.txt`, aby jądro dostało poprawną informację o lokalizacji głównego systemu plików.

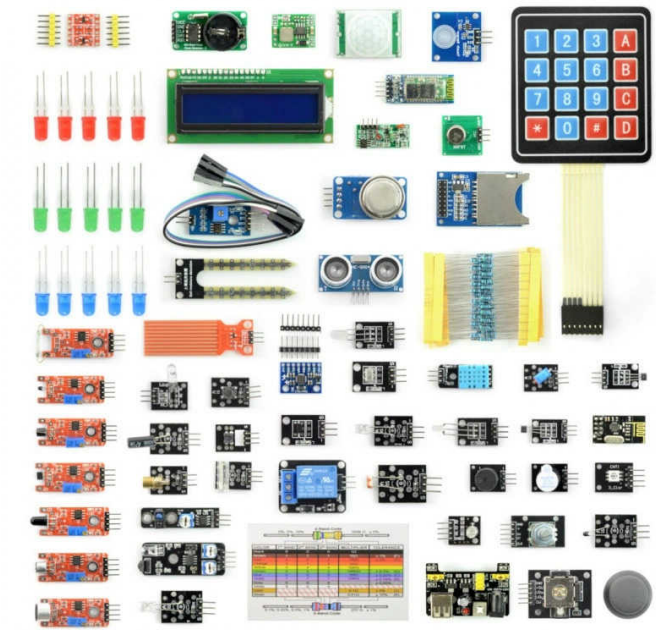
```
gpio=25=ip,pu
[gpio25=0]
os_prefix=rescue/
# We MAY change the name
# of the kernel
# when necessary
kernel=kernel_rescue.img
[gpio25=1]
os_prefix=user/
kernel=kernel_user.img
[all]
```

Praca z systemem uruchamianym przez sieć

- Jeśli nasz obraz systemu nie jest zbyt duży, to możemy używać systemu plików “Initramfs” (omówiony na pierwszym wykładzie). Wtedy możemy pobrać cały obraz systemu (kernel+rootfs) z serwera `tftp` i uruchomić.
- Nawet jeśli system plików jest zbyt duży, by U-Boot był w stanie załadować jądro z systemem plików, możemy uruchamiać system przez sieć. Wówczas jądro ładujemy przez `tftp`, a system plików montujemy z serwera NFS.
- Uwaga! Praca z systemem plików udostępnianym przez NFS może być niebezpieczna dla serwera.
- Inną alternatywą, może być udostępnienie całego “sieciowego dysku” za pomocą protokołu `nbd`. Niestety to rozwiązanie wymaga uruchomienia systemu z “ramdyskiem startowym” który dopiero podłączy główny system plików. Jest ono dość trudne w realizacji.

Dodatkowe wyposażenie w laboratorium

- Do Państwa dyspozycji będzie zestaw dodatkowych urządzeń peryferyjnych, oraz serwomechanizmy i przetworniki A/D i D/A.
- Elementy zestawu mogą być podłączane za pośrednictwem interfejsów takich jak GPIO, SPI lub I2C (**Uwaga na poziomy napięć!**).



Źródło: Botland

Obsługa GPIO

- Co to jest GPIO?
  - General purpose I/O – “wejście wyjście ogólnego przeznaczenia”
    - Wyprowadzenie układu, które możemy skonfigurować jako wejście, lub jako wyjście i stosownie do tego odczytywać, lub modyfikować jego stan.
    - Czasami możemy konfigurować bardziej zaawansowane funkcje – np. zgłaszanie przerwań, włączanie rezystorów podciągających itp.
- Możliwości obsługi z programów użytkownika
  - bezpośredni dostęp do rejestrów, np. przez `/dev/mem`
  - Możliwość obsługi przez sterownik komunikujący się przez sysfs (wymaga włączenia w jądrze opcji: `Device drivers->GPIO Support->sys/class/gpio...`, jest to rozwiązanie przenośne).
  - Dostęp przez `/dev/mem` jest szybszy, ale bardziej ryzykowny (specyficzny dla danej platformy, brak kontroli dostępu przez różne programy, ryzyko zakłócenia pracy systemu w razie zmiany konfiguracji GPIO używanego w innych celach)
  - Nowa biblioteka libgpiod (rozwiązanie przenośne)
  - Czasami możemy użyć specjalnych bibliotek (np. wiringpi lub python-rpi-gpio – są to też rozwiązania specyficzne dla określonej platformy).

Obsługa GPIO przez sysfs (stare, wycofywane, lecz ciągle używane)

- Dokumentacja w źródłach jądra
- Prezentacja o używaniu interfejsów GPIO, SPI, i I2C
- Przejmowanie i zwalnianie wyprowadzenia GPIO:
  - `echo number > /sys/class/gpio/export`
  - `echo number > /sys/class/gpio/unexport`
- Przełączanie kierunku:
  - `echo xx > /sys/class/gpio/gpioNN/direction`
  - `xx =[in | out | low | high]` (low & high włączają wyjście od razu ustawiając poziom, gwarantują brak “szpilek”)
- Zmiana lub odczyt poziomu:
  - `echo [0 | 1] > /sys/class/gpio/gpioNN/value`
  - `cat /sys/class/gpio/gpioNN/value`
- Sterowanie generacją przerwań:
  - `echo [none | rising| falling| both] > /sys/class/gpio/gpioNN/edge`

Obsługa GPIO przez sysfs – niuanse

- Jeśli odczytujemy GPIO z programu, to trzymanie otwartego pliku `.../value`, rodzi pewne konsekwencje.
  - Żeby ponownie odczytać wartość, trzeba pozycję odczytu ustawić na początek pliku.
  - Jeśli ustawimy pozycję odczytu na **początek** pliku, operacja **poll** wykonana na tym pliku, uśpi wątek, do chwili wystąpienia właściwej zmiany poziomu (UWAGA! poll musi być czułe na zdarzenia POLLERR lub POLLPRI).

Przykład kodu w Pythonie czekającego na naciśnięcie przycisku

(UWAGA! Wcześniej należy wyeksportować GPIO i włączyć generację przerwań!)

```
#!/usr/bin/python
import select
f=open("/sys/class/gpio/gpio27/value","r")
e=select.epoll()
e.register(f,select.EPOLLPRI)
while True:
        print f.read()
        e.poll()
        f.seek(0,0)
```

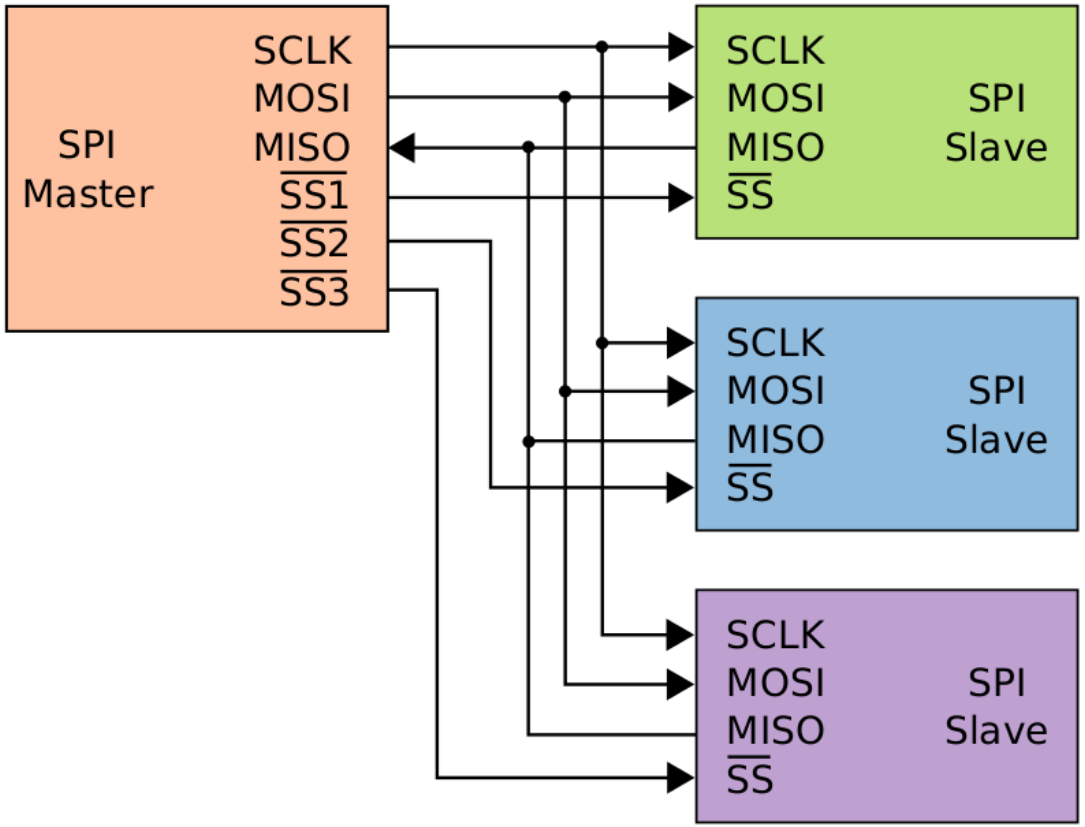
Nowa metoda korzystania z GPIO

- Od kilku lat dostępna jest nowa metoda korzystania z GPIO.
- W nowej metodzie GPIO jest traktowane jako urządzenie znakowe i jest zarządzane przez proces, który je otworzył.
  - Gdy proces zostanie zakończony, przywracany jest poprzedni status GPIO.
- Dostęp jest szybszy, ale ograniczone są możliwości korzystania w skryptach *(są dostępne narzędzia, takie jak gpioget i gpioset, ale stan wyprowadzenia po zakończeniu procesu gpioset w zasadzie jest nieokreślony.)*
- Możliwe jest zdefiniowanie bardziej złożonych zachowań wyprowadzeń.

Gdzie znaleźć informacje o nowej metodzie korzystania z GPIO?

- Prezentacja z konferencji Linux Piter 2018,
- Źródła jądra include/uapi/linux/gpio.h,
- Biblioteka C opakowująca wywołania `ioctl`. *(Uwaga! Wersja używana przez BR może różnić się od tej. Proszę odnaleźć zbiór* `gpiod.h` *w katalogu, w którym BR zbudował* `libgpiod`*, aby wiedzieć jakie dokładnie są prototypy funkcji.)*
- Dla Pythona - oficjalne przykłady ze źródeł ligpiod, lub “help(gpiod)” w Pythonie.

Interfejs SPI



Źródło:GPIO, SPI and I2C from Userspace, the True LinuxWay

Przykłady urządzeń podłączanych przez SPI

- Akcelerometr
- 24–kanałowy kontroler PWM
- Moduł do komunikacji radiowej na dużą odległość

Obsługa SPI

- Z poziomu programów użytkownika mamy dostęp przez sterownik “spidev”.
- Tworzy on pliki specjalne `/dev/spidevX.Y`, gdzie `X` jest numerem magistrali, a `Y` numerem urządzenia.
- Podstawowe operacje: read i write pozwalają na przeprowadzenie zapisu **lub** odczytu bloku danych. Aby przeprowadzić właściwą dla SPI operację równoczesnego zapisu i odczytu, musimy użyć bardziej zaawansowanej funkcji **ioctl**.

Obsługa SPI za pomocą ioctl

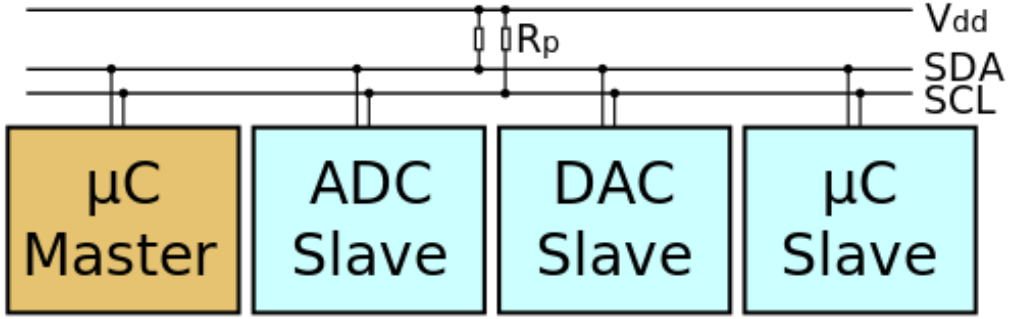
- Program użytkownika wypełnia strukturę spi_ioc_transfer i wywołuje funkcję `ioctl` z komendą SPI_IOC_MESSAGE.
- Przykłady – programy spidev_fdx.c i spidev_test.c w źródłach jądra.

Obsługa of SPI przez GPIO

- Często musimy obsłużyć urządzenie SPI podłączone do zwykłych wyprowadzeń procesora (GPIO).
- Możemy wtedy skorzystać ze sterownika spi-gpio, ale wymaga on, aby nasze urządzenie było zdefiniowane w pliku opisu płyty, lub w DT (przykład).
- Alternatywą może być podejście zrealizowane w module spi-gpio-custom.c (można podobne rozwiązanie zastosować w przypadku innych sterowników zasadniczo przeznaczonych do “platform devices”).

I2C interface

- I2C jest magistralą pozwalającą wiele urządzeń podłączonych za pomocą magistrali o dwóch liniach `SDA` i `SCL` (+ masa)



Source: Wikipedia

- Przykłady urządzeń: Przetwornik A/C, Akcelerometr, Czujnik ciśnienia, temperatury i wilgotności.

Obsługa I2C

- Programy użytkownika mogą obsługiwać urządzenia z interfejsem I2C przez sterownik i2c-dev, opisany w źródłach jądra.
- Sterownik tworzy pliki specjalne: `/dev/i2c-N` , gdzie `N` jest numerem kontrolera.
- Podobnie jak w przypadku SPI, proste transfery mogą być realizowane przez funkcje `write` i `read`, ale bardziej złożone operacje wymagają użycia `ioctl`.
- W szczególności `ioctl` z komendą `I2C_SLAVE` jest niezbędne, aby ustalić addres urządzenia, które chcemy kontrolować!
- Komenda `I2C_RDWR` pozwala zlecić wykonanie złożonej sekwencji zapisów i odczytów (w dowolnej kolejności). Jej argumentem jest adres struktury i2c_rdwr_ioctl_data, zawierającej listę adresów struktur i2c_msg, opisujących kolejne elementarne operacje.
- Przykładowe kody: [1] , [2]
- Uwaga! W pewnych sytuacjach Raspberry Pi błędnie obsługuje transfery I2C.

Obsługa I2C przez GPIO

- Podobnie jak w przypadku SPI, konieczne jest, aby nasze urządzenie było zdefiniowane w pliku opisu płyty, lub w DT (przykład).
- Alternatywnym rozwiązaniem może być użycie modułu i2c-gpio-custom.c pozwalającego dynamicznie zdefiniować linie GPIO, na których ma zostać zrealizowany programowy interfejs I2C.

Biblioteka periphery

- Biblioteka periphery oferuje wygodny dostęp do peryferiów systemu wbudowanego w różnych językach programowania.
  - Język C – c-periphery
  - Język Python – python-periphery
  - Język Lua – lua-periphery

Dziękuję za uwagę!

SKPS
Systemy komputerowe w sterowaniu i pomiarach
Część WZ, Wykład 3 - Rozszerzanie Buildroota i OpenWRT

dr hab. inż. Wojciech M. Zabołotny, prof. uczelni

Wydział Elektroniki i Technik Informacyjnych PW, pok. 225,
wojciech.zabolotny@pw.edu.pl (proszę dodać [SKPS] w temacie)

Jak w przenośny sposób używać PWM?

- Nadal istnieje dostęp do PWM przez sysfs.
- Można też skorzystać z biblioteki python-periphery
  - Daje ona dostęp do wielu funkcji związanych z GPIO i funkcjami alternatywnymi (PWM, I2C, SPI...).
  - Jest zaimplementowana w języku Python, co ułatwia jej przenoszenie.

Przykład korzystania z PWM przez python3-periphery

Poniższy przykład w RPI4 uruchomi PWM na wyprowadzeniach 18 i 19 (w `config.txt` musi być linia `dtoverlay=pwm-2chan`):

```
import periphery as p
s=p.PWM(0,0)
s.period=0.001
s.duty_cycle=0.2
s.enable()
t=p.PWM(0,1)
t.period=0.01
t.duty_cycle=0.3
t.enable()
```

Rozszerzanie Buildroota - skrypty

- Napisaliśmy skrypt powłoki, lub w języku Python
- Jak przenieść go na maszynę wirtualną?
  - Na wykładzie 1 został omówiony mechanizm nakładek na system plików. Wymaga on jednak rekompilacji BR i restartu maszyny. Czy można przenieść nowy skrypt podczas pracy systemu?
  - Jeśli zbudowaliśmy BR z klientem SSH, możemy skopiować skrypt ze stacji roboczej przez `scp`.
  - Możemy uruchomić serwer HTTP na stacji roboczej: `python3 -m http.server` w odpowiednim katalogu, a następnie skopiować skrypt na system docelowy poleceniem `wget` (**Uwaga**, nie nadpisuje zbiorów i kasuje atrybut wykonywalności).
- Uruchamianie skryptów przy starcie systemu?
  - Jeśli używamy standardowego mechanizmu uruchamiania systemu (systemV, BusyBox), możemy dodać odpowiedni skrypt w katalogu `/etc/init.d`. Skrypty te są wykonywane przez skrypt /etc/init.d/rcS przy starcie, a przez skrypt /etc/init.d/rcK przy zatrzymaniu systemu.

Jak skompilować program w języku C?

- Środowisko Buildroot udostępnia cały zestaw narzędzi (kompilator, biblioteki itp.). Czy możemy z nich skorzystać, aby skompilować nasze aplikacje?
- Możemy przygotować odpowiedni makefile, używający "toolchainu" BR i użyć go do kompilacji naszej aplikacji. Możemy też podzielić ustawienia między makefile i skrypt budujący, co pozwoli kompilować aplikację albo w wersji dla stacji roboczej, albo w wersji dla maszyny wirtualnej (katalog demo1BR w przykładach do tego wykładu).
- Można przygotować przenośną wersję "toolchainu" przez użycie komendy `make sdk`.
- Skompilowaną aplikację możemy przenieść na maszynę docelową podobnie jak skrypt.

Rozwiązanie “porządne” - pakiety BR

- Rozwiązanie z poprzedniego slajdu działa, ale nie uwzględnia pełnej konfiguracji BR.
- Zdecydowanie lepszym rozwiązaniem jest stworzenie pakietu BR zawierającego naszą aplikację.
- Niestety, tworzenie pakietów BR jest tematem zbyt złożonym, abyśmy mogli go w pełni omówić na tym wykładzie.
- Polecane źródła informacji:
  - Dodawanie nowych pakietów do BR
  - Dodawanie poprawek do pakietów BR
- Dla łatwego zarządzania własnym projektem wskazane jest przechowywanie modyfikacji poza katalogiem Buildroota.

Inne niuanse BR

- Obsługa urządzeń podłączanych podczas pracy systemu (hot-plug)
  - Konieczne jest dokompilowanie programu “eudev”, co wymaga wcześniej włączenia dynamicznego zarządzania plikami specjalnymi urządzeń:
    - `System_configuration/dev_management` w menu
- Korzystanie z debuggera przy uruchamianiu aplikacji na systemie wbudowanym – dokumentacja “Using gdb in Buildroot”.

Dlaczego OpenWRT?

- Środowisko Buildroot pozwala uzyskać system doskonale zoptymalizowany pod kątem naszych potrzeb.
- Ma to jednak swoją cenę:
  - Niemożność korzystania z instalowalnych pakietów. Byłoby to niezgodne z możliwością ograniczania funkcjonalności jądra lub biblioteki libc.
  - Konieczność pełnej rekompilacji po większości zmian konfiguracji.
    - Czasem wystarczy wymuszenie rekompilacji pewnych pakietów.
- Alternatywą może być środowisko OpenWRT
  - Możemy skorzystać z prekompilowanej binarnej wersji
  - Możemy doinstalowywać i odinstalowywać pakiety bez reinstalacji systemu, a nawet podczas pracy systemu.

Środowisko OpenWRT

- Bardzo popularne środowisko do kompilacji obrazu systemu dla urządzeń sieciowych (nazwa wzięła się od urządzeń WRT firmy Linksys (przejętej w 2003 przez Cisco a w 2013 przez firmę Belkin, ale nadal funkcjonującej jako odrębna marka))
- Oferuje obsługę bardzo wielu urządzeń http://wiki.openwrt.org/toh/start
- Gigantyczna lista pakietów

Detale techniczne OpenWRT

- Używa zapisywalnego systemu plików. Typowo jest to nakładka (overlayfs) systemu jffs2, nakładana na podstawowy obraz systemu squashfs. Dodatkowo podstawowy obraz może służyć jako wersja ratunkowa (oczywiście można wybrać inną konfigurację – np. system plików e4fs na karcie SD).
- System pakietowania opkg
- Duża liczba pakietów obsługujących urządzenia sieciowe, protokoły sieciowe itp.
  Np. urządzenie z wgranym systemem OpenWRT z pakietami olsr lub B.A.T.M.A.N. może pracować jako węzeł bezprzewodowej sieci kratowej (np. Funkfeuer , lub Freifunk).

System pakietowania

- Program “opkg” obsługujący pakiety typu *.ipk.
- Opkg obsługuje zarówno proste polecenia, jak `install` i `remove`, jak i bardziej złożone (niżej kilka bardziej popularnych)
  - `files` – wyświetla listę plików zawartych w pakiecie,
  - `update` – pobranie listy dostępnych pakietów,
  - `list` – wyświetlenie listy pakietów,
  - `list-installed` – wyświetlenie listy zainstalowanych pakietów.

Możliwość skorzystania z wersji prekompilowanej

- Z uwagi na dostępność systemu pakietów, możliwa jest elastyczna konfiguracja systemu bez rekompilacji.
- Dostępne są zarówno pakiety z programami, jak i z modułami jądra (kmod-*), co często pozwala uniknąć rekompilacji ze zmodyfikowanymi ustawieniami jądra.
- Oczywiście wymusza to obecność funkcji i interfejsów w bibliotece libc i jądrze, które być może nie będą wykorzystane (bo używają ich tylko niektóre pakiety, których nigdy nie użyjemy).
- Informacja o zainstalowanych pakietach jest przechowywana w `/usr/lib/opkg` i zwiększa zajętość pamięci FLASH...

Pobieranie wersji prekompilowanej

- Na głównej stronie projektu http://www.openwrt.org jest odnośnik do stabilnych wersji (releases).
- Wybieramy aktualną wersję i właściwą platformę (np. dla 64-bitowej maszyny virt – armvirt/64, albo dla RPi4 – bcm27xx/bcm2711).
- Dla virt64 mamy do dyspozycji obraz systemu zarówno w wersji korzystającej z ramdysku: Image-initramfs jak i wersji korzystającej z głównego systemu plików umieszczonego na nośniku - Image.
- W tym drugim przypadku musimy pobrać jeszcze obraz systemu plików - root.ext4.gz, a następnie rozpakować go.
- Dla RPi4 mamy po prostu obraz do wgrania na kartę SD – rpi-4-ext4-factory.img.gz.

Wygodne narzędzie konfiguracyjne

- Instalując OpenWRT, możemy zainstalować w naszym systemie wygodne narzędzie konfiguracyjne – LuCI.
  - Stanowi ono nakładkę graficzną na system konfiguracji UCI – Unified Configuration System (dodatkowe informacje: [1])
  - Możliwe jest jego rozszerzanie: [1], [2]
- Czy możemy sprawdzić jak działa LuCI, używając maszyny emulowanej w QEMU?
  - Musielibyśmy udostępnić jakoś „gospodarzowi” port, na którym nasłuchuje serwer HTTP LuCI

Uruchamianie QEMU z udostępnieniem portu maszyny wirtualnej

- QEMU udostępnia opcję, którą możemy dołączyć do
  `-netdev user hostfwd=[tcp|udp]:[hostaddr]:hostport - \ [guestaddr]:guestport`
- W naszym przypadku, chcemy udostępnić port 80 maszyny emulowanej na (na przykład) porcie 8888 gospodarza: dopisujemy w parametrach `-netdev user` opcję `hostfwd=tcp::8888-:80`

Konfiguracja sieci w OpenWRT

- OpenWRT zostało stworzone z myślą o sprzęcie sieciowym (rutery, punkty dostępowe). Dlatego domyślna konfiguracja uruchamia typowe dla takiego sprzętu usługi (DNS, DHCP)
- Jeśli nasz system ma jeden interfejs sieciowy, zostanie on potraktowany jako LAN, i zostaną na nim udostępnione te serwisy. Jeśli go podłączymy do sieci, jako klienta, możemy zakłócić jej funkcjonowanie.
- Przy uruchamianiu rzeczywistego sprzętu wskazana jest odpowiednia modyfikacja obsługi sieci w OpenWRT przez podłączeniem do rzeczywistej sieci.
- Możemy to zrobić w LuCI (ale to wymaga połączenia sieciowego)
- Jak to zrobić inaczej?
- Możemy użyć interfejsu UCI w linii poleceń...

Interfejs konfiguracyjny OpenWRT – UCI

- Podstawowy interfejs konfiguracyjny OpenWRT to UCI, opisany tu.
- Zasadniczo wymaga on ręcznej edycji zbiorów konfiguracyjnych, choć możliwe jest też przeprowadzenie jej za pośrednictwem narzędzia “uci” (np. w skryptach).
- Jak możemy zmienić konfigurację sieci?

Ręczna edycja zbioru

Zbiór `/etc/config/network`

```
config interface 'lan'
    option type 'bridge'
    option ifname 'eth0'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'
```

→

```
config interface 'lan'
    option ifname 'eth0'
    option proto 'dhcp'
```

Po skończonej edycji robimy: `/etc/init.d/network reload`

Oprócz tego powinniśmy usunąć zbiór uruchamiający serwer DNS/DHCP dnsmasq: `rm /etc/rc.d/S19dnsmasq`

Obróbka przy pomocy “uci”

- Kasowanie poprzedniej zawartości sekcji:
  `uci delete network.lan`
- Utworzenie nowej, pustej sekcji:
  `uci set network.lan=interface`
- Ustawienie opcji: `uci set network.lan.ifname='eth0'`
  `uci set network.lan.proto='dhcp'`
- Zapis sekcji:
  `uci commit`
- Restart obsługi sieci z nowymi ustawieniami:
  `/etc/init.d/network restart`
- Oprócz tego powinniśmy usunąć zbiór uruchamiający serwer DNS/DHCP dnsmasq:
  `rm /etc/rc.d/S19dnsmasq`

Gdy jednak chcemy skompilować własną wersję OpenWRT

- Całościowy opis systemu budowania OpenWRT jest na tej stronie
- Informacje dotyczące przygotowania systemu są na tej stronie
- Pobranie źródeł OpenWRT:
  - Dostęp do wersji "rozwojowej"
    ```
    git clone https://git.openwrt.org/openwrt/openwrt.git
    ```
  - Dostęp do aktualnej wersji "stabilnej"
    ```
    git checkout openwrt-22.03.3
    ```

Przygotowanie źródeł do kompilacji

- Po ściągnięciu źródeł (opis):
  `git clone https://git.openwrt.org/openwrt/openwrt.git`
- Jeśli źródła były już pobrane, możemy je odświeżyć: `git pull`
- Uaktualniamy listę pakietów (w katalogu openwrt):
  ```
  $ ./scripts/feeds update -a
  ```
  Możemy wyszukiwać interesujące nas pakiety:
  ```
  $ ./scripts/feeds search nano
  Search results in feed 'packages':
  nano  An enhanced clone of the Pico text editor
  ```
- Możemy dodać interesujący nas pakiet do źródeł:
  ```
  $ ./scripts/feeds install nano
  Installing package 'nano'
  ```
  Uwaga! Nie powoduje to automatycznego wybrania danego pakietu do kompilacji!

Konfiguracja systemu

- Podobnie jak w Buildroocie (z którego zresztą OpenWRT się wywodzi) mamy opcję konfiguracji systemu:
  `make menuconfig`
  oraz jądra:
  `make kernel_menuconfig`
- W porównaniu z Buildrootem lepiej zorganizowane jest przechowanie konfiguracji jądra (`make clean` jej nie niszczy...).
- Konfiguracja busybox'a jest obsługiwana przez `make menuconfig`.
- Uwaga! Przy konfiguracji pakietów z oprogramowaniem "M" oznacza, że dane oprogramowanie będzie skompilowane jako pakiet .ipk, podczas gdy "*" oznacza, że będzie dodatkowo wbudowane w obraz systemu plików.

Kompilacja systemu

- Kompilacja obrazu systemu pakietów jest realizowana po prostu przez wywołanie “make”,
- Aby zapewnić pełną diagnostykę błędów, wskazane jest wywoływanie “make” z opcją “V=s”,
- Na komputerze wieloprocesorowym możemy przyspieszyć kompilację przez jej zrównoleglenie, używając opcji “-j”.
  Na przykład: `make V=s -j 9`
  Zaleca się, aby liczba po opcji `-j` była o 1 większa od liczby procesorów.
  Uwaga! Czasami kompilacja z opcją -j kończy się błędem. Należy wówczas sprobować ponowić kompilację bez tej opcji.
- Po kompilacji, gotowe obrazy jądra i systemu plików oraz pakiety zostają umieszczone w katalogu **bin**.

“Czyszczenie” systemu przed kolejną kompilacją

- W porównaniu z Buildrootem wprowadzono opcję bardziej selektywnego wyboru, co chcemy usunąć:
  - `make clean` – czyści tylko katalogi `bin` i `build_dir`
  - `make dirclean` – czyści powyższe katalogi i `staging_dir`, `toolchain` oraz `logs`
  - `make distclean` – czyści wszystko, łącznie z konfiguracjami, załadowanymi źródłami dodatkowymi pakietami (“feeds”).

Dodawanie własnych pakietów

- Podobnie jak w przypadku Buildroota, możliwe jest dodawanie własnych pakietów.
- W wersji podstawowej, wystarczy stworzyć odpowiedni Makefile (przykłady)
- W wersji rozbudowanej, można dodać obsługę pliku Config.in, podobnie jak w Buildroocie.

Określanie zależności między pakietami

- W OpenWRT, możemy określić zależności między pakietami

```
define Package/cups
  SECTION:=net
  CATEGORY:=Network
  DEPENDS:=+zlib +libpthread +libpng +libjpeg +libstdcpp +libusb
  TITLE:=Common UNIX Printing System
  URL:=http://www.cups.org/
endef
```

Określanie położenia źródeł własnych pakietów

- Służą do tego opcje: `PKG_SOURCE` i `PKG_SOURCE_URL` w Makefile'u, opisane razem z innymi zmiennymi definiującymi pakiet.
- Oprócz pobierania źródeł w postaci archiwum , możliwe jest też skorzystanie z repozytoriów używających protokołów takich jak cvs, bzr, hg, git, lub svn.
- Możliwa jest też kompilacja ze źródeł lokalnych, przy czym może to być rozwiązanie tymczasowe, albo możemy na stałe uczynić źródła częścią pakietu, jak to jest np. zrobione w pakiecie `mtd`.
- Przykład własnego pakietu z lokalnymi źródłami

Przykład – źródła dostępne przez CVS

```
PKG_NAME:=transocks
PKG_VERSION:=1.3
PKG_RELEASE:=1

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_SOURCE_URL:=:pserver:anonymous@transocks.cvs.sourceforge.net:/cvsroot/transocks
PKG_SOURCE_SUBDIR:=transocks
PKG_SOURCE_VERSION:=-DNOW
PKG_SOURCE_PROTO:=cvs
Przykład - źródła dostępne przez git
PKG_NAME:=wing
PKG_VERSION:=20120805
PKG_RELEASE:=1
PKG_REV:=d189e36d111788a647cd784536137acbe7a92e17

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.bz2
PKG_SOURCE_URL:=git://github.com/rriggio/click.git
PKG_SOURCE_SUBDIR:=$(PKG_NAME)-$(PKG_VERSION)
PKG_SOURCE_VERSION:=$(PKG_REV)
PKG_SOURCE_PROTO:=git
```

Przykład – źródła dostępne przez hg

```
PKG_NAME:=etherpuppet
PKG_REV:=90decf83d1d4
PKG_VERSION:=0.3_r$(PKG_REV)
PKG_RELEASE:=1

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.bz2
PKG_SOURCE_URL:=http://hg.secdev.org/etherpuppet
PKG_SOURCE_SUBDIR:=$(PKG_NAME)-$(PKG_VERSION)
PKG_SOURCE_VERSION:=$(PKG_REV)
PKG_SOURCE_PROTO:=hg
Przykład - źródła dostępne przez bzr
PKG_NAME:=sftpserver
PKG_REV:=228
PKG_VERSION:=r$(PKG_REV)
PKG_RELEASE:=1

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_SOURCE_URL:=http://www.greenend.org.uk/rjk/bzr/sftpserver.dev
PKG_SOURCE_PROTO:=bzr
PKG_SOURCE_SUBDIR:=$(PKG_NAME)
PKG_SOURCE_VERSION:=$(PKG_REV)
```

Przykład – źródła dostępne przez SVN

```
PKG_NAME:=aprx
PKG_REV:=421
PKG_VERSION:=2.00_r$(PKG_REV)
PKG_RELEASE:=3

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.bz2
PKG_SOURCE_URL:=http://repo.ham.fi/svn/aprx/trunk/
PKG_SOURCE_SUBDIR:=$(PKG_NAME)-$(PKG_VERSION)
PKG_SOURCE_VERSION:=$(PKG_REV)
PKG_SOURCE_PROTO:=svn
```

Możliwość korzystania z SDK

- Tworzenie aplikacji dla OpenWRT nie zawsze wymaga korzystania z pełnego środowiska.
- Możemy przyspieszyć pracę i zmniejszyć zapotrzebowanie na miejsce na dysku, budując SDK (opcja dostępna przez `make menuconfig`).
- SDK może być zainstalowane w dowolnym katalogu i pozwala skompilować pojedyncze pakiety oprogramowania użytkownika, bez rekompilacji całego systemu.
- Bardziej szczegółowe informacje: tutaj
- **Prekompilowane wersje OpenWRT dostarczane są z SDK virt64, RPi4.**

Strona ze szczegółowymi informacjami na temat rozbudowy OpenWRT

- Na stronie https://openwrt.org/docs/guide-developer/start możemy znaleźć bardziej szczegółowe informacje na temat zagadnień związanych z rozwojem OpenWRT.
- Bardzo przystępna seria artykułów o tworzeniu własnych pakietów dla OpenWRT, zaczyna się tu.
- W szczególności, gdy chcemy obsłużyć nową platformę sprzętową za pomocą OpenWRT, warto przeczytać te strony: [1], [2], [3].

Uproszczony opis korzystania z dodatkowego pakietu

- Korzystamy ze szkieletu pakietu
- Dopisujemy na końcu pliku feeds.conf.default linię z pełną ścieżką do katalogu zawierającego katalogi z pakietami:
  `src-link skps /pelna/sciezka/demo1_owrt_pkg`
- Wykonujemy (Uwaga! Może być konieczne wykonanie wcześniej „export LANG=C”):
  `scripts/feeds update -a`
  `scripts/feeds install -p skps -a`
  (powinniśmy dostać informację o zainstalowaniu naszych pakietów)
- Wykonujemy: `make menuconfig` i zaznaczamy nasze pakiety do kompilacji i instalacji.

Uproszczony opis cd.

- Jeśli wywołamy po prostu „make”, to SDK zacznie kompilować wiele komponentów, które nie są nam potrzebne. Dlatego powinniśmy wyraźnie zażądać, że chcemy skompilować tylko nasz pakiet:
  `make package/demo1/compile`
- Jeśli kompilacja się nie uda, zostaniemy poproszeni o jej powtórzenie z opcjami gwarantującymi lepszą diagnostykę:
  `make package/demo1/compile -j1 V=s`
  Musimy wówczas przejrzeć ewentualne komunikaty o błędach i usunąć problem.
- Jeśli wszystko się powiedzie, znajdziemy nasz pakiet (z rozszerzeniem .ipk) w katalogu bin (w jednym z podkatalogów). Musimy go przenieść na naszą maszynę wirtualną – przez wget, lub przez 9P
- Pakiet instalujemy komendą `opkg install pakiet.ipk`

Przyspieszenie korzystania z SDK w laboratorium

- Żeby maksymalnie przyspieszyć pracę z SDK w laboratorium, wskazane jest aby w konfiguracji (po `make menuconfig`) w pozycji `Global Build Settings` wyłączyć opcje:
  - `Select all target specific packages by default`
  - `Select all kernel module packages by default`
  - `Select all userspace packages by default`
  - `Cryptographically sign package lists`
- w pozycji `Advanced configuration options` wskazane jest wyłączenie opcji `Automatic removal of build directories`. Spowoduje to przechowanie wyników kompilacji i przyspieszenie ewentualnej rekompilacji.

Dziękuję za uwagę!