Hands-on Linux Academy 2018

Łukasz Skalski <contact@lukasz-skalski.com>

Plan szkolenia

ĆWICZENIE 1: *Przygotowanie karty SD z systemem dla komputera VisionSOM-6ULL*

- Instrukcja przygotowania karty SD w systemach operacyjnych Linux i Windows
- Linux w systemach wbudowanych? Czy to ma sens?
- Uruchomienie płytki *VisionSOM* oraz konfiguracja komputera do pracy w trybie *Access Point*

Plan szkolenia

ĆWICZENIE 1: *Przygotowanie karty SD z systemem dla komputera VisionSOM-6ULL*

- Instrukcja przygotowania karty SD w systemach operacyjnych Linux i Windows
- Linux w systemach wbudowanych? Czy to ma sens?
- Uruchomienie płytki *VisionSOM* oraz konfiguracja komputera do pracy w trybie *Access Point*

ĆWICZENIE 2: *Interfejsy komunikacyjne w przykładach*

- Sterowanie portami GPIO poprzez interfejs */sys/class/gpio*
- Obsługa przycisków z wykorzystaniem podsystemu *GPIO Buttons*
- Obsługa diod LED z wykorzystaniem podsystemu *LED Class Driver*
- Magistrala I2C na przykładzie obsługi modułu żyroskopu
- Magistrala SPI na przykładzie aplikacji typu "*loopback*"
- Obsługa magistrali 1-Wire z wykorzystaniem czujnika temperatury *DS18B20*

Plan szkolenia

ĆWICZENIE 1: *Przygotowanie karty SD z systemem dla komputera VisionSOM-6ULL*

- Instrukcja przygotowania karty SD w systemach operacyjnych Linux i Windows
- Linux w systemach wbudowanych? Czy to ma sens?
- Uruchomienie płytki *VisionSOM* oraz konfiguracja komputera do pracy w trybie *Access Point*

ĆWICZENIE 2: *Interfejsy komunikacyjne w przykładach*

- Sterowanie portami GPIO poprzez interfejs */sys/class/gpio*
- Obsługa przycisków z wykorzystaniem podsystemu *GPIO Buttons*
- Obsługa diod LED z wykorzystaniem podsystemu *LED Class Driver*
- Magistrala I2C na przykładzie obsługi modułu żyroskopu
- Magistrala SPI na przykładzie aplikacji typu "*loopback*"
- Obsługa magistrali 1-Wire z wykorzystaniem czujnika temperatury *DS18B20*

ĆWICZENIE 3: *Time-to-market w systemach wbudowanych, czyli wykorzystanie gotowych komponentów oprogramowania*

- *Node.js* - systemy wbudowane i JavaScript?
- Prosta implementacja serwera WWW z wykorzystaniem *Node.js*
- Serwer WWW z podziałem na funkcje *front-end* oraz *back-end*
- Komunikacja *front-end*<->*back-end* z wykorzystaniem *socket.io*
- Serwer WWW z odczytem danych z modułu żyroskopu
- Rozbudowa interfejsu serwera WWW o elementy grafiki 3D (*Three.js*)
- Sterowanie wyprowadzeniami GPIO z poziomu przeglądarki internetowej

ĆWICZENIE 1:

Przygotowanie karty SD z systemem dla komputera VisionSOM-6ULL

Przygotowanie karty SD w systemie Linux

- Jedną najprostszych metod zapisania obrazu na karcie SD jest wykorzystanie narzędzia `dd`:

```
dd if=<pliku_wejściowy> of=<plik_wyjściowy> <dodatkowe opcje>
```

Przygotowanie karty SD w systemie Linux

- Jedną najprostszych metod zapisania obrazu na karcie SD jest wykorzystanie narzędzia `dd`:

```
dd if=<pliku_wejściowy> of=<plik_wyjściowy> <dodatkowe opcje>
```

- Poprawne określenie pliku wyjściowego reprezentującego podłączoną do czytnika kartę SD, pozwala na ostateczne określenie formy polecenia `dd`:

```
sudo dd if=/path/linux-academy-2018.img of=/dev/sdX bs=4M oflag=dsync
```

Przygotowanie karty SD w systemie Linux

- Jedną najprostszych metod zapisania obrazu na karcie SD jest wykorzystanie narzędzia `dd`:

```
dd if=<pliku_wejściowy> of=<plik_wyjściowy> <dodatkowe opcje>
```

- Poprawne określenie pliku wyjściowego reprezentującego podłączoną do czytnika kartę SD, pozwala na ostateczne określenie formy polecenia `dd`:

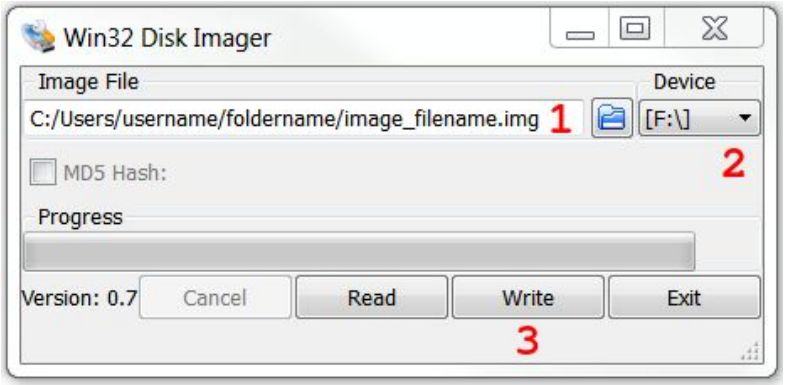
```
sudo dd if=/path/linux-academy-2018.img of=/dev/sdX bs=4M oflag=dsync
```

- Polecenie `dd` nie umożliwia nam monitorowania postępu zapisu danych na kartę SD. Wygodnym rozwiązaniem jest wykorzystanie narzędzia `pv`, który wyświetla informacje o postępie odczytu danych z pliku:

```
pv linux-academy-2018.img | dd of=/dev/sdX bs=4M ofag=dsync
25.5MiB 0:00:02 [5.03MiB/s] [====>                ] 21% ETA 0:00:27
```

Przygotowanie karty SD w systemie Windows

Jedną z najprostszych opcji w przygotowaniu karty SD w systemie operacyjnym Windows jest wykorzystanie darmowego narzędzia `Win32DiskImager`:



Po umieszczeniu czytnika z karta w slocie USB, użytkownik zostanie poproszony o:

- wskazanie ścieżki do pliku **.img* z obrazem systemu (1),
- wskazanie urządzenia docelowego (2),
- przeprowadzenie operacji zapisu poprzez wybranie przycisku *Write* (3).

Linux w systemach wbudowanych?

System operacyjny definiowany jako:

Linux w systemach wbudowanych?

System operacyjny definiowany jako:

- Menedżer zasobów

Linux w systemach wbudowanych?

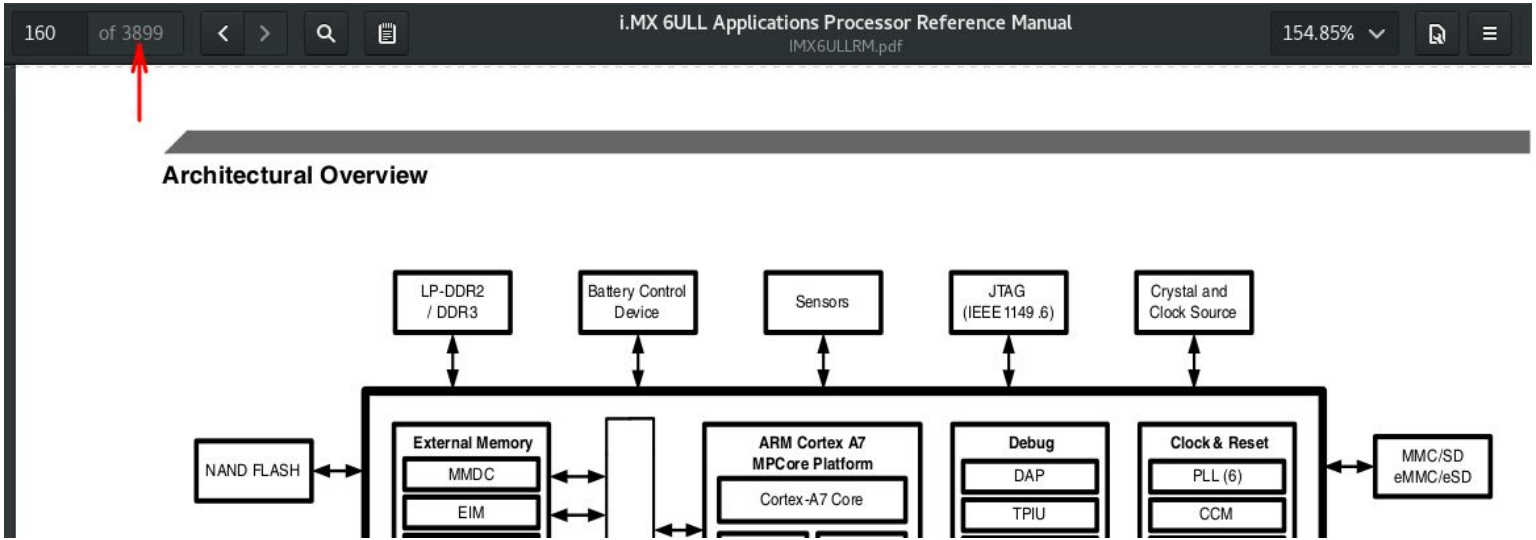
System operacyjny definiowany jako:

- Menedżer zasobów
- Abstrakcja sprzętu

Linux w systemach wbudowanych?

System operacyjny definiowany jako:

- Menedżer zasobów
- Abstrakcja sprzętu



Linux w systemach wbudowanych?

System operacyjny Linux jest odpowiednim rozwiązaniem jeżeli:

Linux w systemach wbudowanych?

System operacyjny Linux jest odpowiednim rozwiązaniem jeżeli:

- od projektowanego urządzenia wymagana jest wysoka niezawodność działania,

Linux w systemach wbudowanych?

System operacyjny Linux jest odpowiednim rozwiązaniem jeżeli:

- od projektowanego urządzenia wymagana jest wysoka niezawodność działania,
- urządzenie ma posiadać rozbudowany graficzny interfejs użytkownika, przetwarzać duże ilości danych, … ,

Linux w systemach wbudowanych?

System operacyjny Linux jest odpowiednim rozwiązaniem jeżeli:

- od projektowanego urządzenia wymagana jest wysoka niezawodność działania,
- urządzenie ma posiadać rozbudowany graficzny interfejs użytkownika, przetwarzać duże ilości danych, … ,
- programowanie w języku C/C++ nie jest naszą najmocniejszą stroną,

Linux w systemach wbudowanych?

System operacyjny Linux jest odpowiednim rozwiązaniem jeżeli:

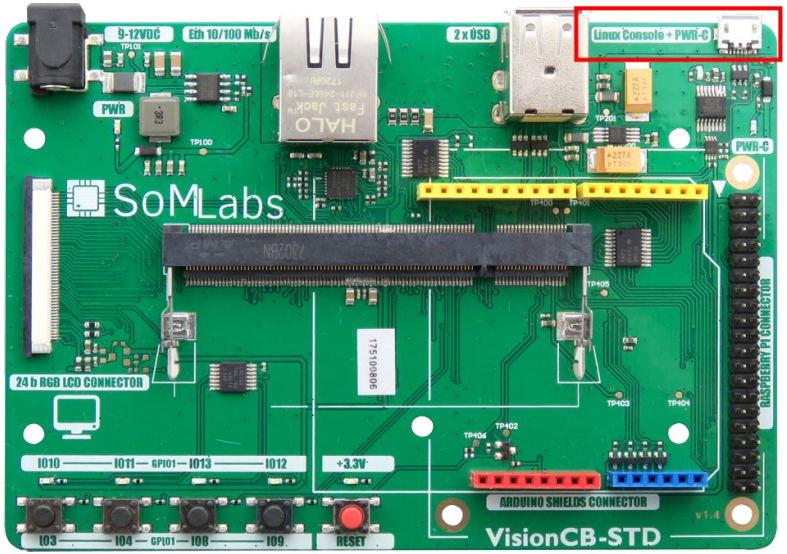
- od projektowanego urządzenia wymagana jest wysoka niezawodność działania,
- urządzenie ma posiadać rozbudowany graficzny interfejs użytkownika, przetwarzać duże ilości danych, … ,
- programowanie w języku C/C++ nie jest naszą najmocniejszą stroną,
- projektowane urządzenia wymaga implementacji stosów USB, Ethernet, Bluetooth,

Linux w systemach wbudowanych?

System operacyjny Linux jest odpowiednim rozwiązaniem jeżeli:

- od projektowanego urządzenia wymagana jest wysoka niezawodność działania,
- urządzenie ma posiadać rozbudowany graficzny interfejs użytkownika, przetwarzać duże ilości danych, … ,
- programowanie w języku C/C++ nie jest naszą najmocniejszą stroną,
- projektowane urządzenia wymaga implementacji stosów USB, Ethernet, Bluetooth,
- zadania stawiane przed naszym urządzeniem, mogą zostać przynajmniej częściowo zrealizowane za pomocą istniejących już pakietów oprogramowania,

Uruchomienie płytki *VisionSOM-6ULL*



Aby uzyskać dostęp do konsoli systemu Linux, należy uruchomić dowolny program emulatora terminalu z ustawieniami transmisji **115200 8N1**:

```
picocom -b 115200 /dev/ttyUSB0
```

Konfiguracja sieci – tryb *Access Point*

Na potrzeby szkolenia oraz w celu łatwej konfiguracji modułu *VisionSOM* do pracy w trybie *Access Point*, w stosunku do domyślnej konfiguracji sieci wprowadzono następujące zmiany:

Konfiguracja sieci – tryb *Access Point*

Na potrzeby szkolenia oraz w celu łatwej konfiguracji modułu *VisionSOM* do pracy w trybie *Access Point*, w stosunku do domyślnej konfiguracji sieci wprowadzono następujące zmiany:

- zainstalowano sterowniki dla wbudowanego modułu *WiFi Murata 1DX*:

```
wget http://ftp.somlabs.com/visionsom-6ull-bcmfirmware.tar.xz
tar -xJf /root/visionsom-6ull-bcmfirmware.tar.xz
```

Konfiguracja sieci – tryb *Access Point*

Na potrzeby szkolenia oraz w celu łatwej konfiguracji modułu *VisionSOM* do pracy w trybie *Access Point*, w stosunku do domyślnej konfiguracji sieci wprowadzono następujące zmiany:

- zainstalowano sterowniki dla wbudowanego modułu *WiFi Murata 1DX*:

```
wget http://ftp.somlabs.com/visionsom-6ull-bcmfirmware.tar.xz
tar -xJf /root/visionsom-6ull-bcmfirmware.tar.xz
```

- z wykorzystaniem narzędzia `apt-get`, w systemie zainstalowano pakiety:

```
apt-get install rfkill
apt-get install hostapd
apt-get install dnsmasq
```

Konfiguracja sieci – tryb *Access Point*

- za pomocą narzędzia `rfkill`, włączono w systemie wszystkie dostępne urządzenia sieci bezprzewodowych:

```
rfkill unblock all
```

Konfiguracja sieci – tryb *Access Point*

- za pomocą narzędzia `rfkill`, włączono w systemie wszystkie dostępne urządzenia sieci bezprzewodowych:

```
rfkill unblock all
```

- poprzez edycję pliku */etc/network/interfaces* do urządzenia przypisano statyczny adres IP dla interfejsu *wlan0*:

```
auto wlan0
iface wlan0 inet static
address 192.168.1.1
netmask 255.255.255.0
```

Konfiguracja sieci – tryb *Access Point*

- dokonano również modyfikacji pliku konfiguracyjnego */etc/dnsmasq.conf* w którym zdefiniowano pulę adresów IP przydzielanych przez serwer DHCP (192.168.1.2-254) oraz czas ich dzierżawy:

```
interface=wlan0
dhcp-range=192.168.1.2,192.168.1.254,255.255.255.0,24h
```

Konfiguracja sieci – tryb *Access Point*

- dokonano również modyfikacji pliku konfiguracyjnego */etc/dnsmasq.conf* w którym zdefiniowano pulę adresów IP przydzielanych przez serwer DHCP (192.168.1.2-254) oraz czas ich dzierżawy:

```
interface=wlan0
dhcp-range=192.168.1.2,192.168.1.254,255.255.255.0,24h
```

- na ostatnim etapie konfiguracji, przełączono moduł WiFi do pracy w trybie **AP** (*Access Point*) – domyślnie moduł pracuje w trybie **STA** (*Station*), umożliwiającym podłączenie się do już istniejących sieci WiFi:

```
echo 2 > /sys/module/bcmdhd/parameters/op_mode
```

Konfiguracja sieci – plik *hostapd.conf*

Konfiguracja sieci – plik *hostapd.conf*

- za pomocą polecenia `touch`, utwórzmy pusty plik `hostapd.conf` w lokalizacji */etc/hostapd*:

```
touch /etc/hostapd/hostapd.conf
```

/etc/hostapd/hostapd.conf

Konfiguracja sieci – plik *hostapd.conf*

- za pomocą polecenia `touch`, utwórzmy pusty plik `hostapd.conf` w lokalizacji */etc/hostapd*:

```
touch /etc/hostapd/hostapd.conf
```

- wykorzystując wybrany edytor tekstu (`vim` lub `nano`), w nowo utworzonym pliku `hostapd.conf` umieśćmy informacje o:

/etc/hostapd/hostapd.conf

Konfiguracja sieci – plik *hostapd.conf*

- za pomocą polecenia `touch`, utwórzmy pusty plik `hostapd.conf` w lokalizacji */etc/hostapd*:

```
touch /etc/hostapd/hostapd.conf
```

- wykorzystując wybrany edytor tekstu (`vim` lub `nano`), w nowo utworzonym pliku `hostapd.conf` umieśćmy informacje o:
    - nazwie interfejsu bezprzewodowego

/etc/hostapd/hostapd.conf

```
interface=wlan0
```

Konfiguracja sieci – plik *hostapd.conf*

- za pomocą polecenia `touch`, utwórzmy pusty plik `hostapd.conf` w lokalizacji */etc/hostapd*:

```
touch /etc/hostapd/hostapd.conf
```

- wykorzystując wybrany edytor tekstu (`vim` lub `nano`), w nowo utworzonym pliku `hostapd.conf` umieśćmy informacje o:
    - nazwie interfejsu bezprzewodowego
    - identyfikatorze sieci i haśle dostępu

/etc/hostapd/hostapd.conf

```
interface=wlan0

ssid=SOMLABS
wpa_passphrase=12345678
```

Konfiguracja sieci – plik *hostapd.conf*

- za pomocą polecenia `touch`, utwórzmy pusty plik `hostapd.conf` w lokalizacji */etc/hostapd*:

```
touch /etc/hostapd/hostapd.conf
```

- wykorzystując wybrany edytor tekstu (`vim` lub `nano`), w nowo utworzonym pliku `hostapd.conf` umieśćmy informacje o:
    - nazwie interfejsu bezprzewodowego
    - identyfikatorze sieci i haśle dostępu
    - standardzie pracy sieci i numerze kanału

/etc/hostapd/hostapd.conf

```
interface=wlan0

ssid=SOMLABS
wpa_passphrase=12345678

hw_mode=g
channel=11
```

Konfiguracja sieci – plik *hostapd.conf*

- za pomocą polecenia `touch`, utwórzmy pusty plik `hostapd.conf` w lokalizacji */etc/hostapd*:

```
touch /etc/hostapd/hostapd.conf
```

- wykorzystując wybrany edytor tekstu (`vim` lub `nano`), w nowo utworzonym pliku `hostapd.conf` umieśćmy informacje o:
    - nazwie interfejsu bezprzewodowego
    - identyfikatorze sieci i haśle dostępu
    - standardzie pracy sieci i numerze kanału
    - opcjach rozgłaszania SSID

/etc/hostapd/hostapd.conf

```
interface=wlan0

ssid=SOMLABS
wpa_passphrase=12345678

hw_mode=g
channel=11

ignore_broadcast_ssid=0
```

Konfiguracja sieci – plik *hostapd.conf*

- za pomocą polecenia `touch`, utwórzmy pusty plik `hostapd.conf` w lokalizacji */etc/hostapd*:

```
touch /etc/hostapd/hostapd.conf
```

- wykorzystując wybrany edytor tekstu (`vim` lub `nano`), w nowo utworzonym pliku `hostapd.conf` umieśćmy informacje o:
  - nazwie interfejsu bezprzewodowego
  - identyfikatorze sieci i haśle dostępu
  - standardzie pracy sieci i numerze kanału
  - opcjach rozgłaszania SSID
  - parametrach szyfrowania

/etc/hostapd/hostapd.conf

```
interface=wlan0

ssid=SOMLABS
wpa_passphrase=12345678

hw_mode=g
channel=11

ignore_broadcast_ssid=0

auth_algs=1
wpa=2
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
```

Konfiguracja sieci – plik *hostapd.conf*

- Po zapisaniu zmian w pliku *hostapd.conf* możemy przystąpić do uruchomienia punktu dostępowego za pomocą polecenia:

```
hostapd -B /etc/hostapd/hostapd.conf
```

Konfiguracja sieci – plik *hostapd.conf*

- Po zapisaniu zmian w pliku *hostapd.conf* możemy przystąpić do uruchomienia punktu dostępowego za pomocą polecenia:

```
hostapd -B /etc/hostapd/hostapd.conf
```

- Poprawne przygotowanie pliku konfiguracyjnego i uruchomienie punktu dostępowego zostanie potwierdzone komunikatem:

```
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
```

ĆWICZENIE 2.1:

Interfejsy komunikacyjne w przykładach - sterowanie GPIO poprzez interfejs /sys/class/gpio

Interfejs /sys/class/gpio - podstawy

- w urządzeniach wbudowanych pracujących pod kontrolą systemu operacyjnego Linux, bezpośredni dostęp do układów peryferyjnych ma wyłącznie jądro systemu,

Interfejs /sys/class/gpio - podstawy

- w urządzeniach wbudowanych pracujących pod kontrolą systemu operacyjnego Linux, bezpośredni dostęp do układów peryferyjnych ma wyłącznie jądro systemu,

- procesy pracujące w przestrzeni użytkownika mogą uzyskać dostęp do sprzętu wyłącznie z wykorzystaniem dedykowanych sterowników sprzętu,

Interfejs /sys/class/gpio - podstawy

- w urządzeniach wbudowanych pracujących pod kontrolą systemu operacyjnego Linux, bezpośredni dostęp do układów peryferyjnych ma wyłącznie jądro systemu,
- procesy pracujące w przestrzeni użytkownika mogą uzyskać dostęp do sprzętu wyłącznie z wykorzystaniem dedykowanych sterowników sprzętu,
- na potrzeby portów GPIO, jądro systemu udostępnia:
    - sterownik kontroli "wyjść" cyfrowych (podsystem ***Led Class Driver***),
    - sterownik kontroli "wejść" cyfrowych (podsystem ***GPIO Buttons***),
    - generyczny sterownik wejść/wyjść (***sysfs gpio interface***),

Interfejs /sys/class/gpio - podstawy

- z poziomu przestrzeni użytkownika, dostęp do informacji udostępnianych przez sterownik jest realizowany poprzez szereg plików dostępnych w katalogu */sys/class/gpio*:

```
root@localhost:~# cd /sys/class/gpio/
root@localhost:/sys/class/gpio# ls -l
total 0
--w------- 1 root root 4096 Oct 1 19:40 export
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip0
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip128
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip32
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip64
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip96
--w------- 1 root root 4096 Oct 1 19:40 unexport
```

Interfejs /sys/class/gpio - podstawy

- z poziomu przestrzeni użytkownika, dostęp do informacji udostępnianych przez sterownik jest realizowany poprzez szereg plików dostępnych w katalogu */sys/class/gpio*:

```
root@localhost:~# cd /sys/class/gpio/
root@localhost:/sys/class/gpio# ls -l
total 0
--w------- 1 root root 4096 Oct 1 19:40 export
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip0
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip128
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip32
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip64
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip96
--w------- 1 root root 4096 Oct 1 19:40 unexport
```

- wyeksportowanie wybranego wyprowadzenia odbywa się poprzez zapis jego numeru porządkowego do pliku *export*, np..:

```
root@localhost:~# echo 10 > /sys/class/gpio/export
```

Interfejs /sys/class/gpio - podstawy

- z poziomu przestrzeni użytkownika, dostęp do informacji udostępnianych przez sterownik jest realizowany poprzez szereg plików dostępnych w katalogu */sys/class/gpio*:

```
root@localhost:~# cd /sys/class/gpio/
root@localhost:/sys/class/gpio# ls -l
total 0
--w------- 1 root root 4096 Oct 1 19:40 export
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip0
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip128
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip32
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip64
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip96
--w------- 1 root root 4096 Oct 1 19:40 unexport
```

- wyeksportowanie wybranego wyprowadzenia odbywa się poprzez zapis jego numeru porządkowego do pliku *export*, np..:

```
root@localhost:~# echo 10 > /sys/class/gpio/export
```

?

Interfejs /sys/class/gpio - podstawy

<GPIO number> = (<GPIO bank> – 1) * 32 + <GPIO bit>

Interfejs /sys/class/gpio - podstawy

<GPIO number> = (<GPIO bank> – 1) * 32 + <GPIO bit>

- **GPIO1_IO17** -> (1-1) * 32 + 17 = 17

Interfejs /sys/class/gpio - podstawy

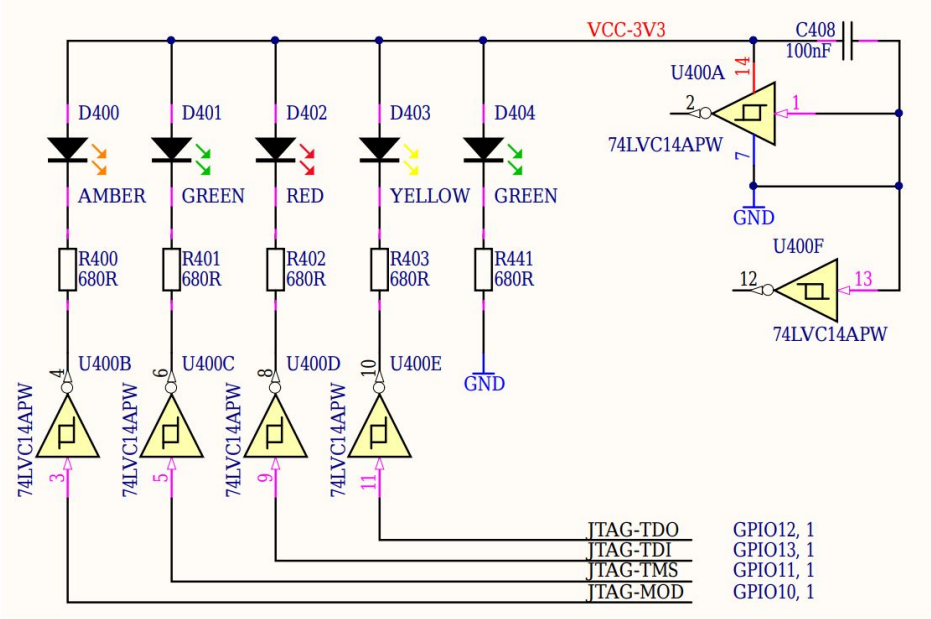
<GPIO number> = (<GPIO bank> – 1) * 32 + <GPIO bit>

- **GPIO1_IO17** -> (1-1) * 32 + 17 = 17
- **GPIO4_IO24** -> (4-1) * 32 + 24 = 120

Interfejs /sys/class/gpio - podstawy

<GPIO number> = (<GPIO bank> – 1) * 32 + <GPIO bit>

- **GPIO1_IO17** -> (1-1) * 32 + 17 = 17
- **GPIO4_IO24** -> (4-1) * 32 + 24 = 120
- **GPIO1_IO10** -> (1-1) * 32 + 17 = 10



Interfejs /sys/class/gpio - podstawy

- każda linia GPIO wyeksportowana do przestrzeni użytkownika reprezentowana jest przez katalog */sys/class/gpio/gpioX*, gdzie X określa numer porządkowy portu:

```
root@localhost:~# cd sys/class/gpio/gpio10
root@localhost:/sys/class/gpio/gpio10# ls -l
total 0
-rw-r--r-- 1 root root 4096 Oct 1 23:04 active_low
lrwxrwxrwx 1 root root 0    Oct 1 23:04 device
-rw-r--r-- 1 root root 4096 Oct 1 23:04 direction
-rw-r--r-- 1 root root 4096 Oct 1 23:04 edge
drwxr-xr-x 2 root root 0    Oct 1 23:04 power
lrwxrwxrwx 1 root root 0    Oct 1 23:04 subsystem
-rw-r--r-- 1 root root 4096 Oct 1 23:04 uevent
-rw-r--r-- 1 root root 4096 Oct 1 23:04 value
```

- sterowanie wybranym portem GPIO realizowane jest poprzez zapis/odczyt plików umieszczonych w katalogu */sys/class/gpio/gpioX*,

Interfejs /sys/class/gpio - podstawy

Plik ***direction*** – umożliwia sterowanie kierunkiem pracy danego wyprowadzenia (wejście/wyjście). Ustawienie kierunku pracy odbywa się poprzez zapis wartości ***out*** (dla wyjścia) lub ***in*** (dla wejścia):

- port *gpioX* pracujący jako wyjście:

```
echo out > /sys/class/gpio/gpioX/direction
```

- port *gpioX* pracujący jako wejście:

```
echo in > /sys/class/gpio/gpioX/direction
```

Interfejs /sys/class/gpio - podstawy

Plik ***value*** – jeżeli wyprowadzenie GPIO pracuje jako wyjście, wówczas zapis wartości 0 ustawia stan niski na wyjściu, natomiast zapis 1 - stan wysoki. W przypadku konfiguracji jako wejście, odczyt zawartości pliku umożliwia odczyt stanu logicznego linii:

- ustawienie stanu niskiego na wyprowadzeniu *gpioX*:

```
echo 0 > /sys/class/gpio/gpioX/value
```

- odczyt stanu linii wejściowej *gpioX*:

```
cat /sys/class/gpio/gpioX/value
```

Interfejs /sys/class/gpio - podstawy

Plik ***edge*** – umożliwia określenie zbocza sygnału jakie wyzwoli zgłoszenie zmiany stanu dla danego wyprowadzenia GPIO, skonfigurowanego jako wejście. Dopuszczalne wartości to: ***none***, ***rising***, ***falling*** oraz ***both***, np.:

- wyzwalanie zboczem opadającym dla wejścia *gpioX*:

```
echo falling > /sys/class/gpio/gpioX/edge
```

- wyzwalanie zboczem narastającym dla wejścia *gpioX*:

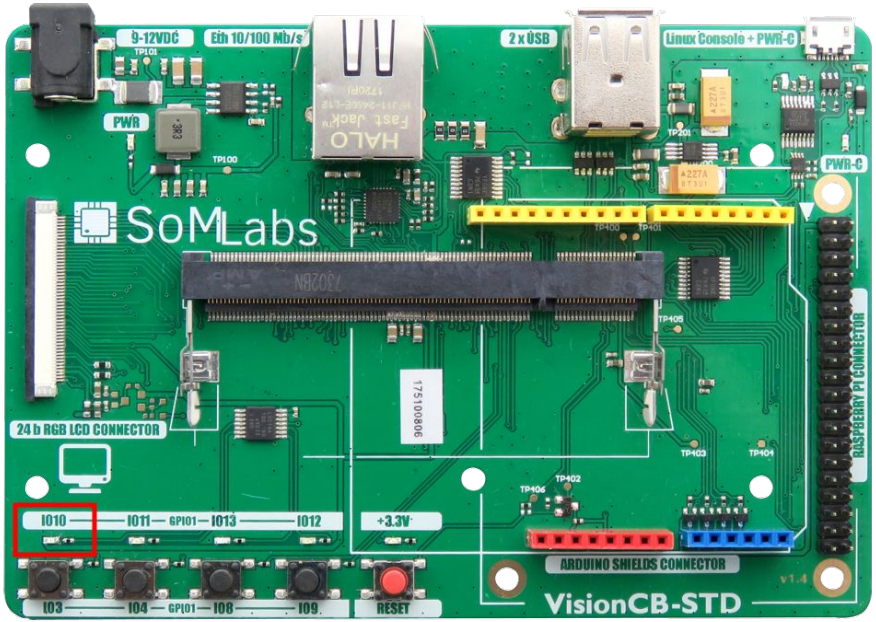
```
echo rising > /sys/class/gpio/gpioX/edge
```

ĆWICZENIE 2.2:

Sterowanie GPIO poprzez interfejs /sys/class/gpio - "Hello World" [skrypt powłoki]

"Hello World" - sterowanie diodą LED [skrypt]

Krótki teoretyczny wstęp zawarty w podpunkcie 2.1, umożliwia nam przygotowanie pierwszego prostego skryptu powłoki, którego zadaniem będzie sekwencyjne miganie diodą LED, podłączoną w płycie bazowej *VisionCB-STD* do wyprowadzenia ***GPIO1_10***.



ĆWICZENIE 2.3:

Sterowanie GPIO poprzez interfejs /sys/class/gpio - "Hello World" [Język C]

"Hello World" - sterowanie diodą LED [Język C]

- `open();`
- `write();`
- `read();`
- `close();`

"Hello World" - sterowanie diodą LED [Język C]

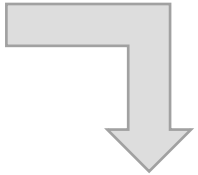
- `open();`
- `write();`
- `read();`
- `close();`

- **Eksport wyprowadzenia GPIO**
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO

```
static int
gpio_export (unsigned int gpio)
{

  int fd, len;
  char buf[BUF_SIZE];

  fd = open (GPIO_DIR "/export", O_WRONLY);
  if (fd < 0)
    {
      perror ("gpio/export");
      return fd;
    }

  len = snprintf (buf, sizeof(buf), "%d", gpio);
  write (fd, buf, len);
  close (fd);

  return 0;
}
```

"Hello World" - sterowanie diodą LED [Język C]

- `open();`
- `write();`
- `read();`
- `close();`

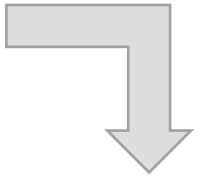


- Eksport wyprowadzenia GPIO
- **Zmiana kierunku pracy GPIO**
- Zmiana stanu na wyjściu GPIO

```
static int
gpio_set_direction (unsigned int gpio,
                    unsigned int direction)
{
  int fd;
  char buf[BUF_SIZE];

  snprintf (buf, sizeof(buf), GPIO_DIR "/gpio%d/direction",gpio);

  fd = open (buf, O_WRONLY);
  if (fd < 0)
    {
      perror ("gpio/direction");
      return fd;
    }

  if (direction)
    write (fd, "out", sizeof("out"));
  else
    write (fd, "in", sizeof("in"));

  close (fd);
  return 0;
}
```

"Hello World" - sterowanie diodą LED [Język C]

- `open();`
- `write();`
- `read();`
- `close();`



- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- **Zmiana stanu na wyjściu GPIO**

```
static int
gpio_set_value (unsigned int gpio,
                unsigned int value)
{
  int fd;
  char buf[BUF_SIZE];

  snprintf (buf, sizeof(buf), GPIO_DIR "/gpio%d/value", gpio);
  fd = open (buf, O_WRONLY);
  if (fd < 0)
    {
      perror ("gpio/set-value");
      return fd;
    }

  if (value)
    write (fd, "1", 2);
  else
    write (fd, "0", 2);

  close (fd);
  return 0;
}
```

"Hello World" - sterowanie diodą LED [Język C]

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO

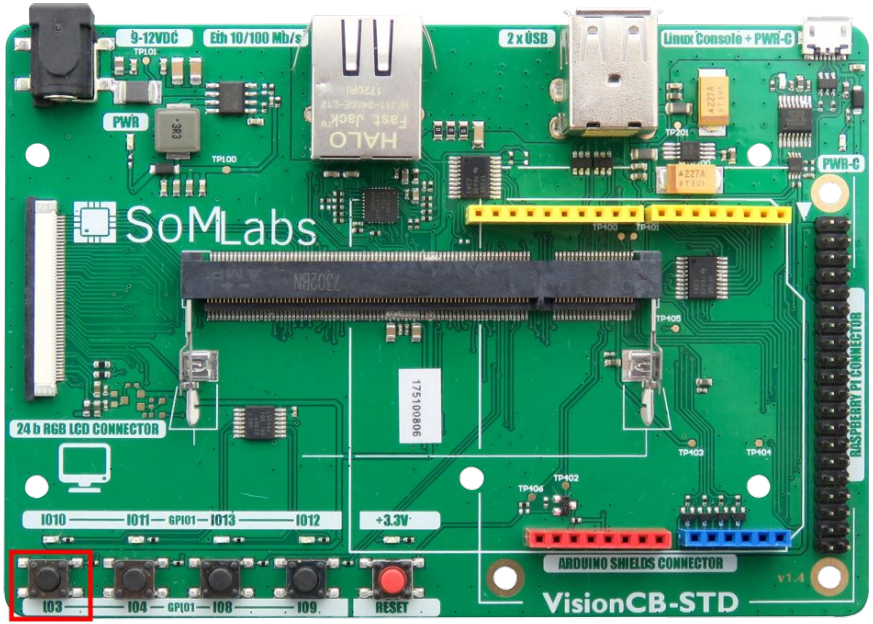
```
#define GPIO_PIN        10
#define GPIO_DIR        "/sys/class/gpio"
#define GPIO_IN         0
#define GPIO_OUT        1

int main (void)
{
  if (gpio_export (GPIO_PIN) < 0)
    exit (EXIT_FAILURE);

  if (gpio_set_direction (GPIO_PIN, GPIO_OUT) < 0)
    exit (EXIT_FAILURE);

  while (1)
    {
      gpio_set_value (GPIO_PIN, 1);
      sleep (1);

      gpio_set_value (GPIO_PIN, 0);
      sleep (1);
    }
  return EXIT_SUCCESS;
}
```

ĆWICZENIE 2.4:

Sterowanie GPIO poprzez interfejs /sys/class/gpio - obsługa przycisku

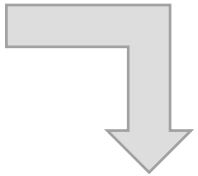
Prosta obsługa przycisku z funkcją poll()

Bazując na kodzie z podpunktu 2.3, zaimplementujemy przerwaniową obsługę przycisku podłączonego do wyprowadzenia ***GPIO1_3***.



Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`



- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- **Konfiguracja zbocza**
- Pobranie deskryptora pliku

```
static int
gpio_set_edge (unsigned int  gpio,
               char         *edge)
{
  int fd;
  char buf[BUF_SIZE];

  snprintf (buf, sizeof(buf), GPIO_DIR "/gpio%d/edge", gpio);

  fd = open (buf, O_WRONLY);
  if (fd < 0)
    {
      perror ("gpio/edge");
      return fd;
    }
  write (fd, edge, strlen(edge) + 1);
  close (fd);

  return 0;
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- **Pobranie deskryptora pliku**

```
static int
gpio_fd_open (unsigned int gpio)
{
  int fd;
  char buf[BUF_SIZE];

  snprintf (buf, sizeof(buf), GPIO_DIR "/gpio%d/value", gpio);

  fd = open (buf, O_RDONLY | O_NONBLOCK );
  if (fd < 0)
    perror ("gpio/fd_open");

  return fd;
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- Pobranie deskryptora pliku

```
int main (void)
{
  struct pollfd fdset[1];
  int nfds = 1, fd, ret;

  if (gpio_export (GPIO_PIN) < 0)
    exit (EXIT_FAILURE);

  if (gpio_set_direction (GPIO_PIN, GPIO_IN) < 0)
    exit (EXIT_FAILURE);


  if (gpio_set_edge (GPIO_PIN, "rising") < 0)
    exit (EXIT_FAILURE);


  fd = gpio_fd_open (GPIO_PIN);
  if (fd < 0)
    exit (EXIT_FAILURE);


  lseek (fd, 0, SEEK_SET);
  read (fd, &buf, BUF_SIZE);


  while (1)
    {
      //TODO
    }
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- Pobranie deskryptora pliku

```
int main (void)
{
  struct pollfd fdset[1];
  int nfds = 1, fd, ret;

  if (gpio_export (GPIO_PIN) < 0)
    exit (EXIT_FAILURE);

  if (gpio_set_direction (GPIO_PIN, GPIO_IN) < 0)
    exit (EXIT_FAILURE);


  if (gpio_set_edge (GPIO_PIN, "rising") < 0)
    exit (EXIT_FAILURE);


  fd = gpio_fd_open (GPIO_PIN);
  if (fd < 0)
    exit (EXIT_FAILURE);


  lseek (fd, 0, SEEK_SET);
  read (fd, &buf, BUF_SIZE);


  while (1)
    {
      //TODO
    }
}
```

```
struct pollfd
{
  int fd;
  short events;
  short revents;
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- Pobranie deskryptora pliku

```
int main (void)
{
  struct pollfd fdset[1];
  int nfds = 1, fd, ret;

  if (gpio_export (GPIO_PIN) < 0)
    exit (EXIT_FAILURE);

  if (gpio_set_direction (GPIO_PIN, GPIO_IN) < 0)
    exit (EXIT_FAILURE);


  if (gpio_set_edge (GPIO_PIN, "rising") < 0)
    exit (EXIT_FAILURE);


  fd = gpio_fd_open (GPIO_PIN);
  if (fd < 0)
    exit (EXIT_FAILURE);


  lseek (fd, 0, SEEK_SET);
  read (fd, &buf, BUF_SIZE);


  while (1)
    {
      //TODO
    }
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- Pobranie deskryptora pliku

```
int main (void)
{
  struct pollfd fdset[1];
  int nfds = 1, fd, ret;

  if (gpio_export (GPIO_PIN) < 0)
    exit (EXIT_FAILURE);

  if (gpio_set_direction (GPIO_PIN, GPIO_IN) < 0)
    exit (EXIT_FAILURE);


  if (gpio_set_edge (GPIO_PIN, "rising") < 0)
    exit (EXIT_FAILURE);


  fd = gpio_fd_open (GPIO_PIN);
  if (fd < 0)
    exit (EXIT_FAILURE);


  lseek (fd, 0, SEEK_SET);
  read (fd, &buf, BUF_SIZE);


  while (1)
    {
      //TODO
    }
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- Pobranie deskryptora pliku

```
int main (void)
{
  struct pollfd fdset[1];
  int nfds = 1, fd, ret;

  if (gpio_export (GPIO_PIN) < 0)
    exit (EXIT_FAILURE);

  if (gpio_set_direction (GPIO_PIN, GPIO_IN) < 0)
    exit (EXIT_FAILURE);


  if (gpio_set_edge (GPIO_PIN, "rising") < 0)
    exit (EXIT_FAILURE);


  fd = gpio_fd_open (GPIO_PIN);
  if (fd < 0)
    exit (EXIT_FAILURE);


  lseek (fd, 0, SEEK_SET);
  read (fd, &buf, BUF_SIZE);


  while (1)
    {
      //TODO
    }
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- Pobranie deskryptora pliku

```
while (1)
  {
    memset (fdset, 0, sizeof(fdset));

    fdset[0].fd = fd;
    fdset[0].events = POLLPRI;

    ret = poll (fdset, nfds, -1);
    if (ret < 0) {
      printf ("poll(): failed!\n");
      goto exit;
    }

    if (fdset[0].revents & POLLPRI) {
      printf ("poll(): GPIO_%d interrupt occurred\n", GPIO_PIN);
      lseek (fdset[0].fd, 0, SEEK_SET);
      read (fdset[0].fd, &buf, BUF_SIZE);
    }

   fflush(stdout);
}
```

Prosta obsługa przycisku z funkcją poll()

- `open();`
- `write();`
- `read();`
- `close();`

- Eksport wyprowadzenia GPIO
- Zmiana kierunku pracy GPIO
- Zmiana stanu na wyjściu GPIO
- Konfiguracja zbocza
- Pobranie deskryptora pliku

```
while (1)
  {
    memset (fdset, 0, sizeof(fdset));

    fdset[0].fd = fd;
    fdset[0].events = POLLPRI;

    ret = poll (fdset, nfds, -1);
    if (ret < 0) {
      printf ("poll(): failed!\n");
      goto exit;
    }

    if (fdset[0].revents & POLLPRI) {
      printf ("poll(): GPIO_%d interrupt occurred\n", GPIO_PIN);
      lseek (fdset[0].fd, 0, SEEK_SET);
      read (fdset[0].fd, &buf, BUF_SIZE);
    }

   fflush(stdout);
}
```

ĆWICZENIE 2.5:

Obsługa przycisku z wykorzystaniem podsystemu GPIO Buttons

Obsługa przycisku – podsystem *GPIO Buttons*

- konfigurację sterownika *GPIO Buttons* rozpoczynamy od jego włączenia w jądrze systemu:

```
Device Drivers --->
  Input device support --->
    [*] Keyboards --->
      <*> GPIO Buttons
```

- niezbędne jest również włączenie tzw. interfejsu zdarzeń (*Event Interface*) z podsystemu *Linux Input System*:

```
Device Drivers --->
  Input device support --->
    <*> Event interface
```

Obsługa przycisku – podsystem *GPIO Buttons*

- konfigurację sterownika *GPIO Buttons* rozpoczynamy od jego włączenia w jądrze systemu:

```
Device Drivers --->
  Input device support --->
    [*] Keyboards --->
      <*> GPIO Buttons
```

- niezbędne jest również włączenie tzw. interfejsu zdarzeń (*Event Interface*) z podsystemu *Linux Input System*:

```
Device Drivers --->
  Input device support --->
    <*> Event interface
```

???

Kompilacja jądra systemu Linux "w pigułce"

- domyślna konfiguracja jądra systemu:

```
cd /root/kernel/
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- visionsom-6ull-linux_defconfig
```

- konfiguracja jądra z wykorzystaniem "graficznego" interfejsu:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig
```

- właściwa kompilacja jądra systemu:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- zImage
```

Opis konfiguracji sprzętowej z *Device Tree*

- *Device Tree* to pliki tekstowe opisujące urządzenie i konfigurację podłączonego sprzętu

Opis konfiguracji sprzętowej z *Device Tree*

- *Device Tree* to pliki tekstowe opisujące urządzenie i konfigurację podłączonego sprzętu
- Dla architektury ARM, pliki *Device Tree* umieszczone są w katalogu *arch/arm/boot/dts*

Opis konfiguracji sprzętowej z *Device Tree*

- *Device Tree* to pliki tekstowe opisujące urządzenie i konfigurację podłączonego sprzętu
- Dla architektury ARM, pliki *Device Tree* umieszczone są w katalogu *arch/arm/boot/dts*
- Pliki Device Tree możemy podzielić na:
  - ***.dtsi** - opisujące konkretny model SoC-a,
  - ***.dts** - plik opisujący konkretną płytkę / urządzenie,
  - ***.dtb** - skompilowany plik *Device Tree*

Opis konfiguracji sprzętowej z *Device Tree*

- *Device Tree* to pliki tekstowe opisujące urządzenie i konfigurację podłączonego sprzętu
- Dla architektury ARM, pliki *Device Tree* umieszczone są w katalogu *arch/arm/boot/dts*
- Pliki Device Tree możemy podzielić na:
  - ***.dtsi** - opisujące konkretny model SoC-a,
  - ***.dts** - plik opisujący konkretną płytkę / urządzenie,
  - ***.dtb** - skompilowany plik *Device Tree*
- Kompilacja *Device Tree*:

```
make ARCH=arm somlabs-visionsom-6ull.dtb
make ARCH=arm dtbs
```

Opis konfiguracji sprzętowej z *Device Tree*

- w plikach *Device Tree* umieszczamy opis w formacie wymaganym przez sterownik danego urządzenia:

Opis konfiguracji sprzętowej z *Device Tree*

- w plikach *Device Tree* umieszczamy opis w formacie wymaganym przez sterownik danego urządzenia:

https://www.kernel.org/doc/Documentation/devicetree/bindings/

Opis konfiguracji sprzętowej z *Device Tree*

- w plikach *Device Tree* umieszczamy opis w formacie wymaganym przez sterownik danego urządzenia:

https://www.kernel.org/doc/Documentation/devicetree/bindings/

```
gpio-keys {

  compatible = "gpio-keys";



  btn3 {
    label = "btn3";
    gpios = <&gpio1 8 GPIO_ACTIVE_HIGH>;
    linux,code = <103>; /* <KEY_UP> */
  };

  btn4 {
    label = "btn4";
    gpios = <&gpio1 9 GPIO_ACTIVE_HIGH>;
    linux,code = <108>; /* <KEY_DOWN> */
  };
};
```

Opis konfiguracji sprzętowej z *Device Tree*

- w plikach *Device Tree* umieszczamy opis w formacie wymaganym przez sterownik danego urządzenia:

https://www.kernel.org/doc/Documentation/devicetree/bindings/

```
gpio-keys {

  compatible = "gpio-keys";
  pinctrl-0 = <&pinctrl_gpio_keys>;
  pinctrl-names = "default";

  btn3 {
    label = "btn3";
    gpios = <&gpio1 8 GPIO_ACTIVE_HIGH>;
    linux,code = <103>; /* <KEY_UP> */
  };

  btn4 {
    label = "btn4";
    gpios = <&gpio1 9 GPIO_ACTIVE_HIGH>;
    linux,code = <108>; /* <KEY_DOWN> */
  };
};
```

Opis konfiguracji sprzętowej z *Device Tree*

- w plikach *Device Tree* umieszczamy opis w formacie wymaganym przez sterownik danego urządzenia:

https://www.kernel.org/doc/Documentation/devicetree/bindings/

```
gpio-keys {

  compatible = "gpio-keys";
  pinctrl-0 = <&pinctrl_gpio_keys>;
  pinctrl-names = "default";

  btn3 {
    label = "btn3";
    gpios = <&gpio1 8 GPIO_ACTIVE_HIGH>;
    linux,code = <103>; /* <KEY_UP> */
  };

  btn4 {
    label = "btn4";
    gpios = <&gpio1 9 GPIO_ACTIVE_HIGH>;
    linux,code = <108>; /* <KEY_DOWN> */
  };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_gpio_keys: gpio-keys {
      fsl,pins = <
        MX6UL_PAD_GPIO1_IO08__GPIO1_IO08 0x1b0b0
        MX6UL_PAD_GPIO1_IO09__GPIO1_IO09 0x1b0b0
      >;
    };
  };
};
```

Opis konfiguracji sprzętowej z *Device Tree*

- w plikach *Device Tree* umieszczamy opis w formacie wymaganym przez sterownik danego urządzenia:

https://www.kernel.org/doc/Documentation/devicetree/bindings/

```
gpio-keys {

  compatible = "gpio-keys";
  pinctrl-0 = <&pinctrl_gpio_keys>;
  pinctrl-names = "default";

  btn3 {
    label = "btn3";
    gpios = <&gpio1 8 GPIO_ACTIVE_HIGH>;
    linux,code = <103>; /* <KEY_UP> */
  };

  btn4 {
    label = "btn4";
    gpios = <&gpio1 9 GPIO_ACTIVE_HIGH>;
    linux,code = <108>; /* <KEY_DOWN> */
  };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_gpio_keys: gpio-keys {
      fsl,pins = <
        MX6UL_PAD_GPIO1_IO08__GPIO1_IO08 0x1b0b0
        MX6UL_PAD_GPIO1_IO09__GPIO1_IO09 0x1b0b0
      >;
    };
  };
};
```

Opis konfiguracji sprzętowej z *Device Tree*

- w plikach *Device Tree* umieszczamy opis w formacie wymaganym przez sterownik danego urządzenia:

https://www.kernel.org/doc/Documentation/devicetree/bindings/

```
gpio-keys {

  compatible = "gpio-keys";
  pinctrl-0 = <&pinctrl_gpio_keys>;
  pinctrl-names = "default";

  btn3 {
    label = "btn3";
    gpios = <&gpio1 8 GPIO_ACTIVE_HIGH>;
    linux,code = <103>; /* <KEY_UP> */
  };

  btn4 {
    label = "btn4";
    gpios = <&gpio1 9 GPIO_ACTIVE_HIGH>;
    linux,code = <108>; /* <KEY_DOWN> */
  };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_gpio_keys: gpio-keys {
      fsl,pins = <
        MX6UL_PAD_GPIO1_IO08__GPIO1_IO08 0x1b0b0
        MX6UL_PAD_GPIO1_IO09__GPIO1_IO09 0x1b0b0
      >;
    };
  };
};
```

Opis konfiguracji sprzętowej z *Device Tree*

```
PAD_CTL_HYS                     (1 << 16)
PAD_CTL_PUS_100K_DOWN           (0 << 14)
PAD_CTL_PUS_47K_UP              (1 << 14)
PAD_CTL_PUS_100K_UP             (2 << 14)
PAD_CTL_PUS_22K_UP              (3 << 14)
PAD_CTL_PUE                     (1 << 13)
PAD_CTL_PKE                     (1 << 12)
PAD_CTL_ODE                     (1 << 11)
PAD_CTL_SPEED_LOW               (1 << 6)
PAD_CTL_SPEED_MED               (2 << 6)
PAD_CTL_SPEED_HIGH              (3 << 6)
PAD_CTL_DSE_DISABLE             (0 << 3)
PAD_CTL_DSE_240ohm              (1 << 3)
PAD_CTL_DSE_120ohm              (2 << 3)
PAD_CTL_DSE_80ohm               (3 << 3)
PAD_CTL_DSE_60ohm               (4 << 3)
PAD_CTL_DSE_48ohm               (5 << 3)
PAD_CTL_DSE_40ohm               (6 << 3)
PAD_CTL_DSE_34ohm               (7 << 3)
PAD_CTL_SRE_FAST                (1 << 0)
PAD_CTL_SRE_SLOW                (0 << 0)
```

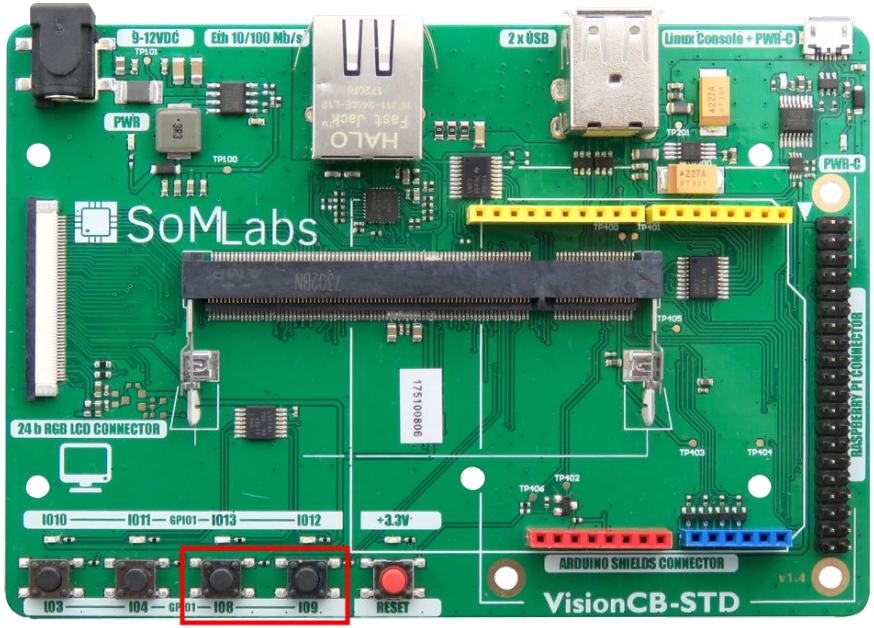
Opis konfiguracji sprzętowej z *Device Tree*

```
PAD_CTL_HYS                    (1 << 16)
PAD_CTL_PUS_100K_DOWN          (0 << 14)
PAD_CTL_PUS_47K_UP             (1 << 14)
PAD_CTL_PUS_100K_UP            (2 << 14)
PAD_CTL_PUS_22K_UP             (3 << 14)
PAD_CTL_PUE                    (1 << 13)
PAD_CTL_PKE                    (1 << 12)
PAD_CTL_ODE                    (1 << 11)
PAD_CTL_SPEED_LOW              (1 << 6)
PAD_CTL_SPEED_MED              (2 << 6)
PAD_CTL_SPEED_HIGH             (3 << 6)
PAD_CTL_DSE_DISABLE            (0 << 3)
PAD_CTL_DSE_240ohm             (1 << 3)
PAD_CTL_DSE_120ohm             (2 << 3)
PAD_CTL_DSE_80ohm              (3 << 3)
PAD_CTL_DSE_60ohm              (4 << 3)
PAD_CTL_DSE_48ohm              (5 << 3)
PAD_CTL_DSE_40ohm              (6 << 3)
PAD_CTL_DSE_34ohm              (7 << 3)
PAD_CTL_SRE_FAST               (1 << 0)
PAD_CTL_SRE_SLOW               (0 << 0)
```

Interactive i.MX Pin Mux Tool

Obsługa przycisku – podsystem *GPIO Buttons*

Na podstawie przedstawionego opisu *Device Tree*, ze sterownikiem ***gpio-keys***, skojarzone zostały przyciski podłączone do wyprowadzeń ***GPIO1_8*** oraz ***GPIO1_9*** .



Obsługa przycisku – podsystem *GPIO Buttons*

- od strony procesu w przestrzeni użytkownika, dostęp do przycisków jest realizowany poprzez plik ***/dev/input/eventX***

Obsługa przycisku – podsystem *GPIO Buttons*

- od strony procesu w przestrzeni użytkownika, dostęp do przycisków jest realizowany poprzez plik ***/dev/input/eventX***

- zdarzenia informujące o wciśnięciu przycisku przekazywane są do przestrzeni użytkownika przez generyczny interfejs zdarzeń (*Event Interface*). Każde z takich zdarzeń opisane jest strukturą ***input_event***:

```
struct input_event {
  struct timeval time;
  unsigned short type;
  unsigned short code;
  unsigned int value;
};
```

Obsługa przycisku – podsystem *GPIO Buttons*

- od strony procesu w przestrzeni użytkownika, dostęp do przycisków jest realizowany poprzez plik ***/dev/input/eventX***

- zdarzenia informujące o wciśnięciu przycisku przekazywane są do przestrzeni użytkownika przez generyczny interfejs zdarzeń (*Event Interface*). Każde z takich zdarzeń opisane jest strukturą ***input_event***:

```
struct input_event {
  struct timeval time;
  unsigned short type;
  unsigned short code;
  unsigned int value;
};
```

- struktura ta, poprzez określone pola dostarcza procesom użytkownika informacje o czasie wystąpienia zdarzenia (pole ***time***), typie zdarzenia (pole ***type***), kodzie użytego przycisku (pole ***code***) oraz jego aktualnym położeniu (pole ***value***).

Obsługa przycisku – podsystem *GPIO Buttons*

Typy zdarzeń dla pola `.code`:

- **EV_REL** - używany do określenia względnego przesunięcia, np.: ruch myszą o 5 jednostek w lewo od aktualnego położenia,
- **EV_ABS** - używany do bezwzględnego określenia położenia, np.: współrzędne punktu w interfejsach dotykowych,
- **EV_SYN** - separator zdarzeń w czasie lub przestrzeni - wykorzystywany np: w ekranach dotykowych z opcją multitouch,
- **EV_KEY** - używany do opisania stanu urządzeń takich jak klawiatury, przyciski, itd.

Obsługa przycisku – podsystem *GPIO Buttons*

```
int
main (void)
{
  struct input_event ev;
  int size = sizeof(ev), fd;

  fd = open ("/dev/input/event2", O_RDONLY);
  if (fd < 0)
    {
      printf ("event2: failed!\n");
      return EXIT_FAILURE;
    }

  while (1)
    {

    } /* while */

exit:
  close (fd);
  return EXIT_FAILURE;
}
```

Obsługa przycisku – podsystem *GPIO Buttons*

```
int
main (void)
{
  struct input_event ev;
  int size = sizeof(ev), fd;

  fd = open ("/dev/input/event2", O_RDONLY);
  if (fd < 0)
    {
      printf ("event2: failed!\n");
      return EXIT_FAILURE;
    }

  while (1)
    {

    } /* while */

exit:
  close (fd);
  return EXIT_FAILURE;
}
```

Obsługa przycisku – podsystem *GPIO Buttons*

```
int
main (void)
{
  struct input_event ev;
  int size = sizeof(ev), fd;

  fd = open ("/dev/input/event2", O_RDONLY);
  if (fd < 0)
    {
      printf ("event2: failed!\n");
      return EXIT_FAILURE;
    }

  while (1)
    {

    } /* while */

exit:
  close (fd);
  return EXIT_FAILURE;
}
```

```
if (read(fd, &ev, size) < size)
  {
    printf ("Reading from /dev/input/event2 failed!\n");
    goto exit;
  }

if (ev.type == EV_KEY)
  {
    if (ev.code == KEY_DOWN)
     ev.value ? printf("KEY_DOWN:0") : printf("KEY_DOWN:1");
    else if (ev.code == KEY_UP)
      ev.value ? printf("KEY_UP:0") : printf("KEY_UP:1");
    else
      puts ("WTF?!");
  }
```

Obsługa przycisku – podsystem *GPIO Buttons*

```
int
main (void)
{
  struct input_event ev;
  int size = sizeof(ev), fd;

  fd = open ("/dev/input/event2", O_RDONLY);
  if (fd < 0)
    {
      printf ("event2: failed!\n");
      return EXIT_FAILURE;
    }

  while (1)
    {

    } /* while */

exit:
  close (fd);
  return EXIT_FAILURE;
}
```

```
if (read(fd, &ev, size) < size)
  {
    printf ("Reading from /dev/input/event2 failed!\n");
    goto exit;
  }

if (ev.type == EV_KEY)
  {
    if (ev.code == KEY_DOWN)
     ev.value ? printf("KEY_DOWN:0") : printf("KEY_DOWN:1");
    else if (ev.code == KEY_UP)
      ev.value ? printf("KEY_UP:0") : printf("KEY_UP:1");
    else
      puts ("WTF?!");
  }
```

ĆWICZENIE 2.6:

Obsługa diod LED z wykorzystaniem podsystemu LED Class Driver

LED Class Driver - konfiguracja sterownika

- konfiguracja sterownika *LED Class Driver* w jądrze systemu:

```
Device Drivers --->
  [*] LED Support --->
    <*> LED Class Support
    <*> LED Support for GPIO connected LEDs
```

- konfiguracja wyzwalaczy dla sterownika *LED Class Driver*:

```
Device Drivers --->
  [*] LED Support --->
    <*> LED Trigger Support
      <*> LED Heartbeat Trigger
      <*> LED CPU Trigger
      <*> LED Default ON Trigger
      / … /
```

LED Class Driver – opis *Device Tree*

- opis w *Device Tree* definiujący podłączone diody LED:

```
leds {

  compatible = "gpio-leds";
  pinctrl-0 = <&pinctrl_gpio_leds>;
  pinctrl-names = "default";

  led3 {
    label = "led3";
    gpios = <&gpio1 13 GPIO_ACTIVE_HIGH>;
    linux,default-trigger = "heartbeat";
  };

  led4 {
    label = "led4";
    gpios = <&gpio1 12 GPIO_ACTIVE_HIGH>;
    linux,default-trigger = "mmc1";
  };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_gpio_leds: gpio-leds {
      fsl,pins = <
        MX6UL_PAD_JTAG_TDI__GPIO1_IO13 0x17099
        MX6UL_PAD_JTAG_TDO__GPIO1_IO12 0x17099
      >;
    };
  };
};
```

https://www.kernel.org/doc/Documentation/devicetree/bindings/leds/leds-gpio.txt

ĆWICZENIE 2.7:

Magistrala I2C

Magistrala I2C – konfiguracja jądra systemu

- włączenie sterownika I2C w jądrze systemu:

```
Device Drivers --->
  [*] I2C support --->
    <*> I2C device interface
```

- włączenie kontrolera magistrali I2C:

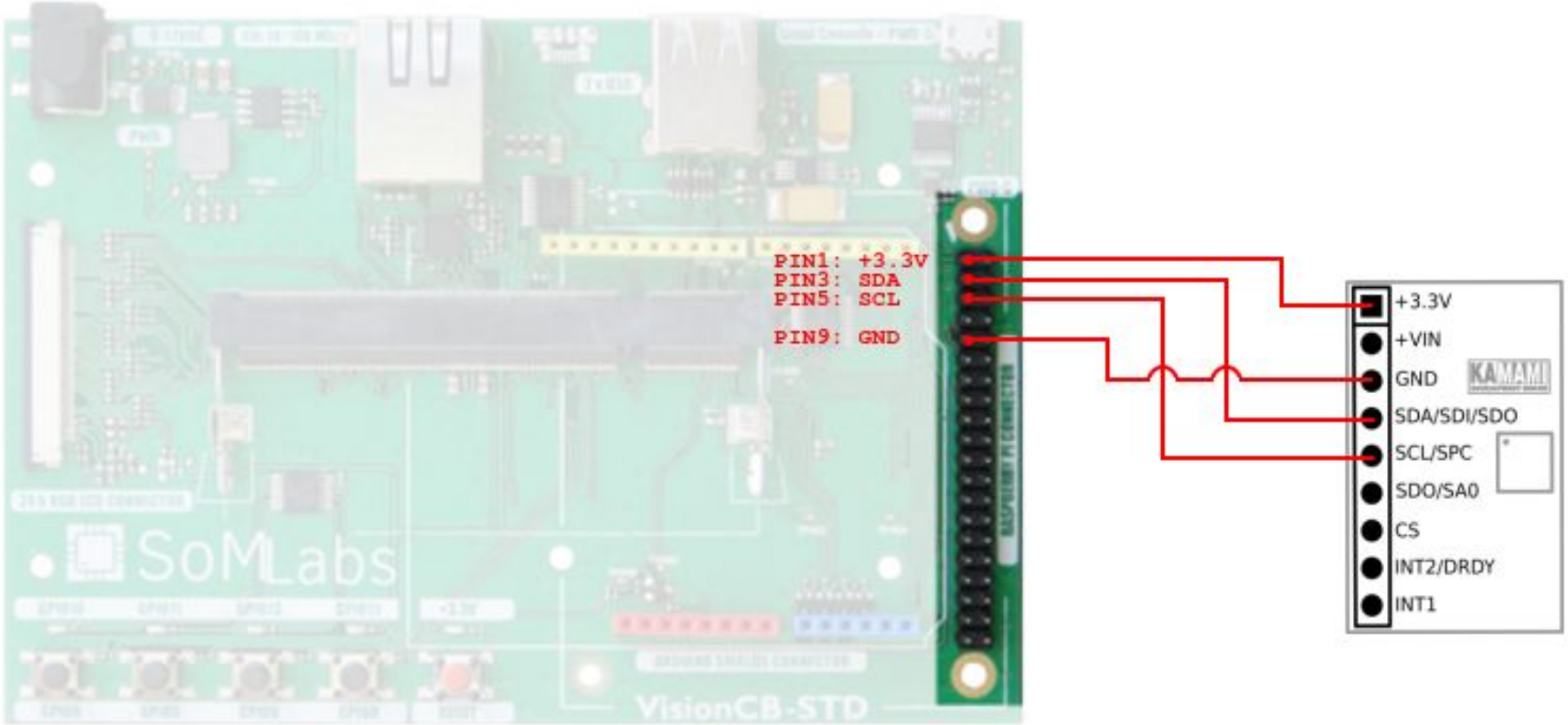
```
Device Drivers --->
  [*] I2C support --->
    [*] I2C Hardware Bus Support --->
      <*> IMX I2C interface
      < > GPIO-based bitbanging I2C
```

Magistrala I2C – opis Device Tree

```
&i2c1 {

  clock-frequency = <100000>;
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_i2c1>;
  //status = "okay";

};

&i2c2 {

  clock_frequency = <100000>;
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_i2c2>;
  status = "okay";

};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_i2c1: i2c1grp {
      fsl,pins = <
        MX6UL_PAD_UART4_TX_DATA__I2C1_SCL 0x4001b8b0
        MX6UL_PAD_UART4_RX_DATA__I2C1_SDA 0x4001b8b0
      >;
    };

    pinctrl_i2c2: i2c2grp {
      fsl,pins = <
        MX6UL_PAD_UART5_TX_DATA__I2C2_SCL 0x4001b8b0
        MX6UL_PAD_UART5_RX_DATA__I2C2_SDA 0x4001b8b0
      >;
    };
  };
};
```

Magistrala I2C - podłączenie żyroskopu



Magistrala I2C – pakiet *i2c-tools*

i2cdetect – umożliwia przeskanowanie wskazanej w parametrze magistrali I2C. Wynikiem działania polecenia jest tablica adresów wraz z listą dostępnych na magistrali urządzeń.

- ogólna postać polecenia:

```
i2cdetect [-y] [-a] [-q|-r] I2CBUS [FIRST LAST]
```

- przeskanowanie magistrali sprzętowej *i2c_1*:

```
i2cdetect -y 1
```

Magistrala I2C – pakiet *i2c-tools*

i2cset – umożliwia zapisanie określonej wartości pod wybrany rejestr układu *Slave*. Pierwszym parametrem polecenia jest numer porządkowy magistrali, następnie adres układu *Slave*, numer rejestru, wartość zapisywanej danej oraz jej rozmiar (domyślnie 1 bajt).

- ogólna postać polecenia:

```
i2cset [-f] [-y] [-m MASK] I2CBUS CHIP-ADDRESS DATA-ADDRESS [VALUE [MODE]]
```

- zapis wartości 0x20 do rejestru 0x10 układu o adresie 0x30 podłączonego do magistrali *i2c_1*:

```
i2cset -y 1 0x30 0x10 0x20
```

Magistrala I2C – pakiet *i2c-tools*

i2cget – pozwala na odczyt danej z wybranego układu *Slave* i określonego rejestru. Pierwszym parametrem polecenia jest numer porządkowy magistrali, następnie adres układu *Slave*, numer rejestru spod którego będziemy wykonywać odczyt oraz rozmiar odczytywanej danej (domyślnie jednobajtowej).

- ogólna postać polecenia:

```
i2cget [-f] [-y] I2CBUS CHIP-ADDRESS [DATA-ADDRESS [MODE]]
```

- odczyt dwubajtowej zawartości rejestru 0x0F układu o adresie 0x6b podłączonego do *i2c_1*:

```
i2cget -y 1 0x6b 0x0F w
```

Magistrala I2C - obsługa programowa

- `open();`
- `write();`
- `read();`
- `close();`

Magistrala I2C - obsługa programowa

- `open();`
- `write();`
- `read();`
- `close();`

- **Otworzenie urządzenia /dev/i2c-X**
- Wybranie adresu urządzenia SLAVE
- Zapis danych do urządzenia SLAVE
- Odczyt danych z urządzenia SLAVE

```
/* open i2c device */
int i2c_fd = open ("/dev/i2c-1", O_RDWR);
if (i2c_fd < 0)
  return EXIT_FAILURE;

/* set slave address */
int ret = ioctl (i2c_fd, I2C_SLAVE, SLAVE_ADDR);
  return EXIT_FAILURE;

/* write data */
char buf[3];
buf[0] = 0x01;
buf[1] = 0x0F;
buf[2] = 0xFF;

if (write (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;

/* read data */
if (read (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;
```

Magistrala I2C - obsługa programowa

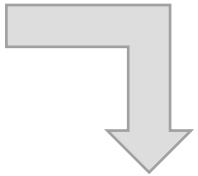
- `open();`
- `write();`
- `read();`
- `close();`

- Otworzenie urządzenia /dev/i2c-X
- **Wybranie adresu urządzenia SLAVE**
- Zapis danych do urządzenia SLAVE
- Odczyt danych z urządzenia SLAVE

```
/* open i2c device */
int i2c_fd = open ("/dev/i2c-1", O_RDWR);
if (i2c_fd < 0)
  return EXIT_FAILURE;

/* set slave address */
int ret = ioctl (i2c_fd, I2C_SLAVE, SLAVE_ADDR);
  return EXIT_FAILURE;

/* write data */
char buf[3];
buf[0] = 0x01;
buf[1] = 0x0F;
buf[2] = 0xFF;

if (write (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;

/* read data */
if (read (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;
```

Magistrala I2C - obsługa programowa

- `open();`
- `write();`
- `read();`
- `close();`

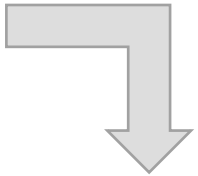


- Otworzenie urządzenia /dev/i2c-X
- Wybranie adresu urządzenia SLAVE
- **Zapis danych do urządzenia SLAVE**
- Odczyt danych z urządzenia SLAVE

```
/* open i2c device */
int i2c_fd = open ("/dev/i2c-1", O_RDWR);
if (i2c_fd < 0)
  return EXIT_FAILURE;

/* set slave address */
int ret = ioctl (i2c_fd, I2C_SLAVE, SLAVE_ADDR);
  return EXIT_FAILURE;

/* write data */
char buf[3];
buf[0] = 0x01;
buf[1] = 0x0F;
buf[2] = 0xFF;

if (write (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;

/* read data */
if (read (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;
```

Magistrala I2C - obsługa programowa

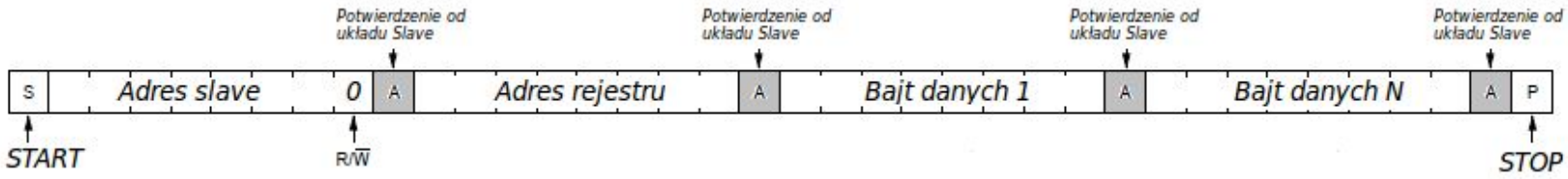
- `open();`
- `write();`
- `read();`
- `close();`



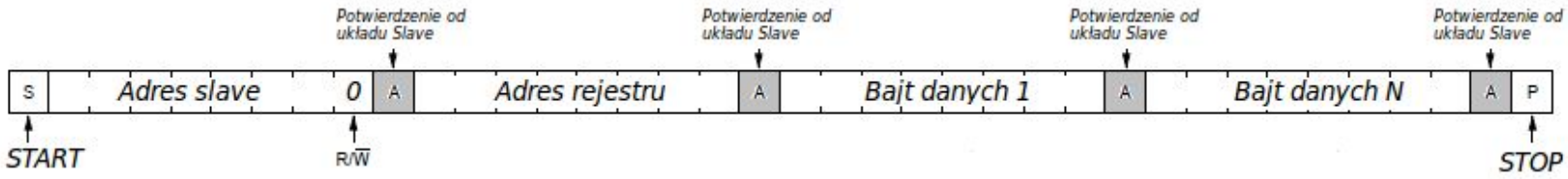
- Otworzenie urządzenia /dev/i2c-X
- Wybranie adresu urządzenia SLAVE
- Zapis danych do urządzenia SLAVE
- **Odczyt danych z urządzenia SLAVE**

```
/* open i2c device */
int i2c_fd = open ("/dev/i2c-1", O_RDWR);
if (i2c_fd < 0)
  return EXIT_FAILURE;

/* set slave address */
int ret = ioctl (i2c_fd, I2C_SLAVE, SLAVE_ADDR);
  return EXIT_FAILURE;

/* write data */
char buf[3];
buf[0] = 0x01;
buf[1] = 0x0F;
buf[2] = 0xFF;

if (write (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;

/* read data */
if (read (i2c_fd, buf, 3) != 3)
  return EXIT_FAILURE;
```

I2C – komunikacja na poziomie bajtowym

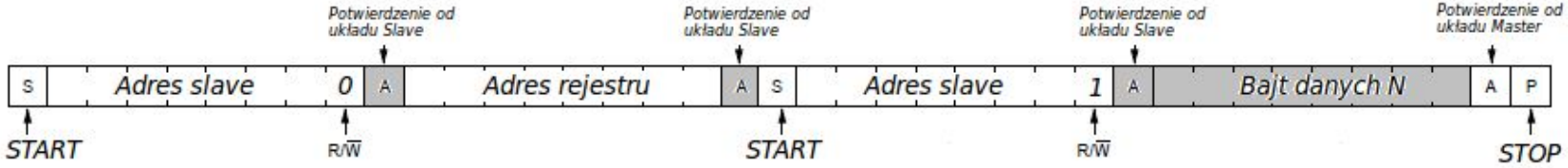


I2C – komunikacja na poziomie bajtowym

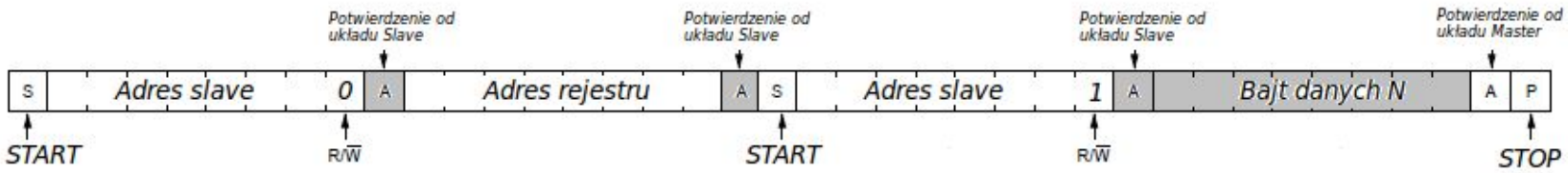



```
unsigned char init_seq[6];

init_seq[0] = (CTRL_REG1 | AUTO_INCREMENT);
init_seq[1] = 0xCF; /* CTRL_REG1: normal mode, xyz enable */
init_seq[2] = 0x01; /* CTRL_REG2: <default value> */
init_seq[3] = 0x00; /* CTRL_REG3: <default value> */
init_seq[4] = 0x80; /* CTRL_REG4: 250dps, Block Data Update */
init_seq[5] = 0x02; /* CTRL_REG5: <default value> */

if (write (i2c_fd, init_seq, 6) != 6)
  return -1;
```

I2C – komunikacja na poziomie bajtowym

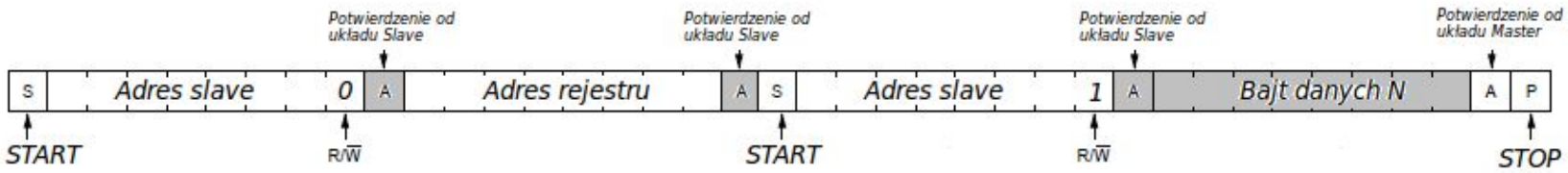


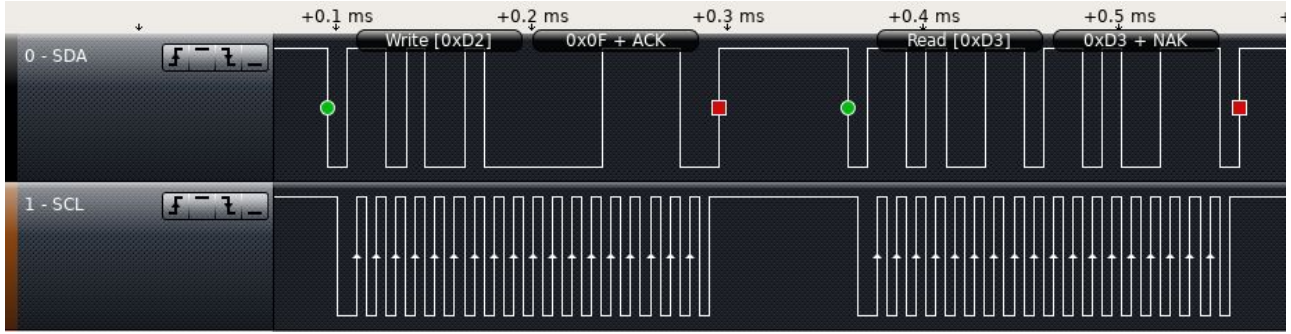
I2C – komunikacja na poziomie bajtowym




```
buf[0] = reg;
write(i2c, buf, 1);
read(i2c, buf, 1);
return buf[0];
```

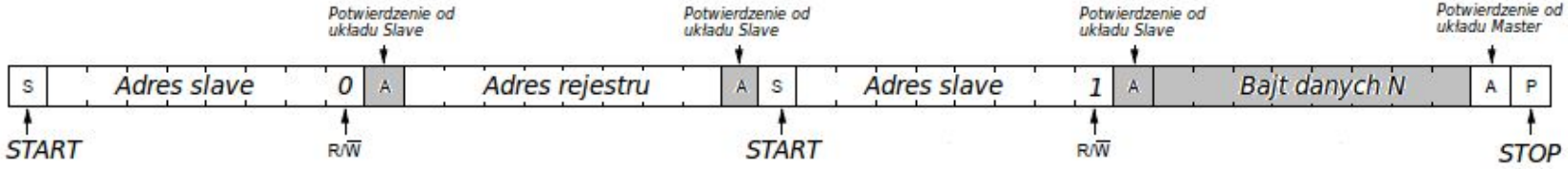
I2C – komunikacja na poziomie bajtowym



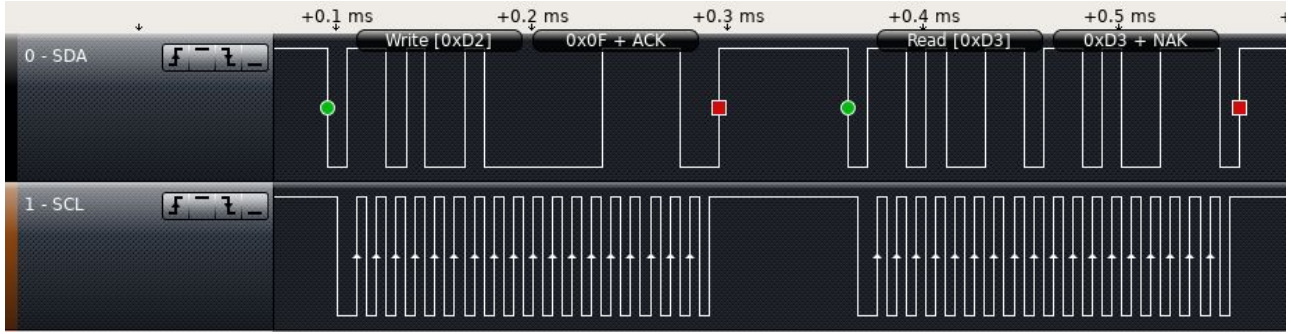


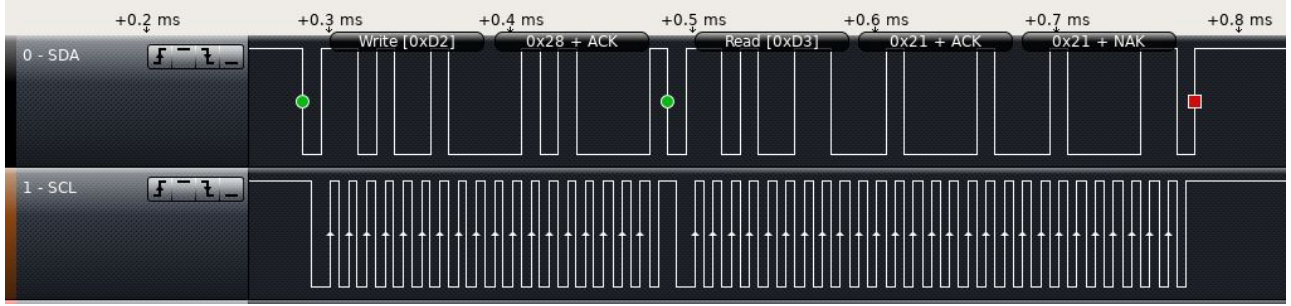

```
buf[0] = reg;
write(i2c, buf, 1);
read(i2c, buf, 1);
return buf[0];
```

I2C – komunikacja na poziomie bajtowym

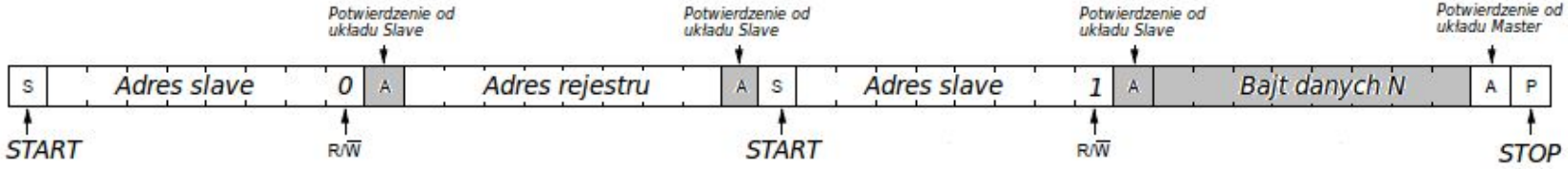



```
buf[0] = reg;
write(i2c, buf, 1);
read(i2c, buf, 1);
return buf[0];
```

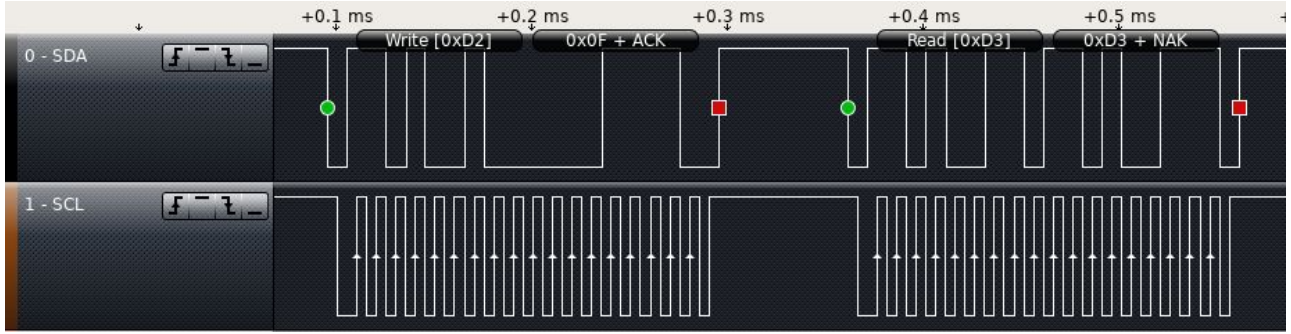




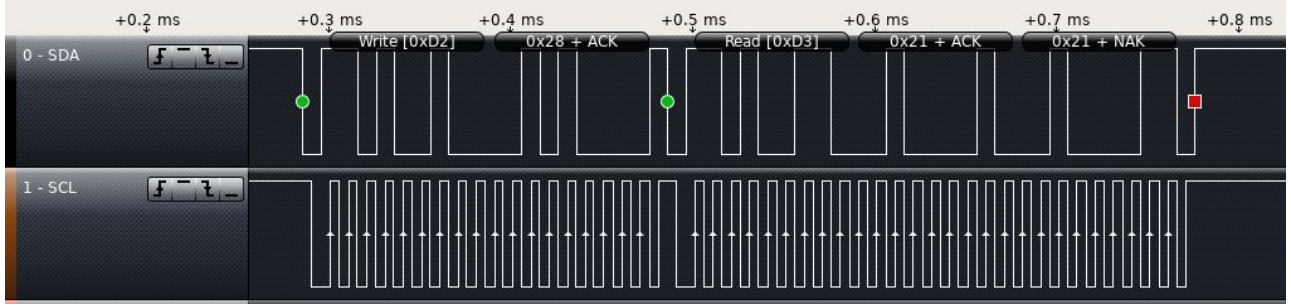
I2C – komunikacja na poziomie bajtowym




```
buf[0] = reg;
write(i2c, buf, 1);
read(i2c, buf, 1);
return buf[0];
```




```
ioctl (fd, I2C_RDWR, …);
```



Magistrala I2C – operacja RD/WR

```
unsigned char reg_addr = STATUS_REG;
unsigned char reg_data[1];
int ret;

struct i2c_msg messages[] =
  {
    {
      GYRO_ADDR,        /* slave address */
      0,                /* flags: 0 */
      sizeof(reg_addr), /* transfer size */
      &reg_addr         /* data */
    },

    {
      GYRO_ADDR,        /* slave address */
      I2C_M_RD,         /* flags: READ */
      sizeof(reg_data), /* transfer size */
      reg_data          /* data */
    }
  };
```

Magistrala I2C – operacja RD/WR

```
unsigned char reg_addr = STATUS_REG;
unsigned char reg_data[1];
int ret;

struct i2c_msg messages[] =
  {
    {
      GYRO_ADDR,          /* slave address */
      0,                  /* flags: 0 */
      sizeof(reg_addr),   /* transfer size */
      &reg_addr           /* data */
    },

    {
      GYRO_ADDR,          /* slave address */
      I2C_M_RD,           /* flags: READ */
      sizeof(reg_data),   /* transfer size */
      reg_data            /* data */
    }
  };
```

```
struct i2c_rdwr_ioctl_data packets =
  {
    messages,
    sizeof(messages) / sizeof(struct i2c_msg)
  };

ret = ioctl (i2c_fd, I2C_RDWR, &packets);
if (ret < 0)
  return ret;
```

Magistrala I2C - moduł żyroskopu

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010100
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output

Magistrala I2C - moduł żyroskopu

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010100
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output

Magistrala I2C - moduł żyroskopu

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010100
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output

Magistrala I2C - moduł żyroskopu

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010100
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output

ĆWICZENIE 2.8:

Magistrala SPI

Magistrala SPI – konfiguracja jądra systemu

- włączenie kontrolera magistrali SPI w jądrze systemu:

```
Device Drivers --->
  [*] SPI support --->
    <*> Freescale i.MX SPI controllers
    < > GPIO-based bitbanging SPI Master
```

- włączenie sterownika ***spidev*** umożliwiającego uzyskanie dostępu do magistrali z przestrzeni użytkownika:

```
Device Drivers --->
  [*] SPI support --->
    <*> User mode SPI device driver support
```

Magistrala SPI – opis Device Tree

```
&ecspi3 {



 status = "okay";

};
```

Magistrala SPI – opis Device Tree

```
&ecspi3 {
 fsl,spi-num-chipselects = <1>;



 status = "okay";

};
```

Magistrala SPI – opis Device Tree

```
&ecspi3 {
 fsl,spi-num-chipselects = <1>;

 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_ecspi3: ecspi3grp {
      fsl,pins = <
        MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO   0x1b0b1
        MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI   0x1b0b1
        MX6UL_PAD_UART2_RX_DATA__ECSPI3_SCLK 0x1b0b1
        MX6UL_PAD_UART2_TX_DATA__ECSPI3_SS0  0x1b0b1
      >;
    };
  };
};
```

Magistrala SPI – opis Device Tree

```
&ecspi3 {
 fsl,spi-num-chipselects = <1>;

 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_ecspi3: ecspi3grp {
      fsl,pins = <
        MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO   0x1b0b1
        MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI   0x1b0b1
        MX6UL_PAD_UART2_RX_DATA__ECSPI3_SCLK 0x1b0b1
        MX6UL_PAD_UART2_TX_DATA__ECSPI3_SS0  0x1b0b1
      >;
    };
  };
};
```

Magistrala SPI – opis Device Tree

```
&ecspi3 {
 fsl,spi-num-chipselects = <1>;

 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_ecspi3: ecspi3grp {
      fsl,pins = <
        MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO   0x1b0b1
        MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI   0x1b0b1
        MX6UL_PAD_UART2_RX_DATA__ECSPI3_SCLK 0x1b0b1
        MX6UL_PAD_UART2_TX_DATA__ECSPI3_SS0  0x1b0b1
      >;
    };
  };
};
```

Magistrala SPI – opis Device Tree

```
&ecspi3 {
 fsl,spi-num-chipselects = <1>;

 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_ecspi3: ecspi3grp {
      fsl,pins = <
        MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO   0x1b0b1
        MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI   0x1b0b1
        MX6UL_PAD_UART2_RX_DATA__ECSPI3_SCLK 0x1b0b1
        MX6UL_PAD_UART2_TX_DATA__ECSPI3_SS0  0x1b0b1
      >;
    };
  };
};
```

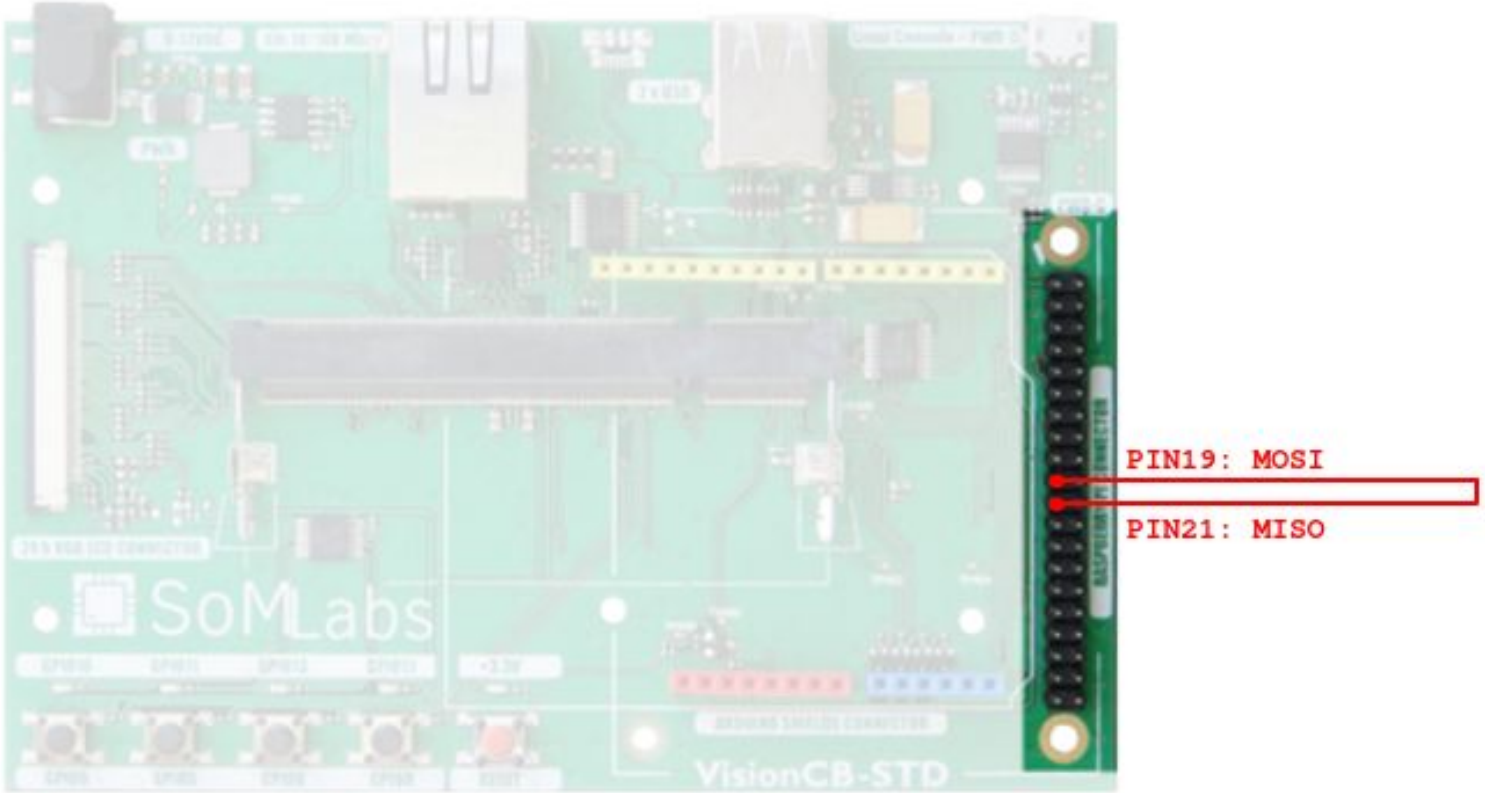
https://community.nxp.com/thread/467443

Magistrala SPI – opis Device Tree

```
&ecspi3 {
 fsl,spi-num-chipselects = <1>;
 cs-gpios = <&gpio1 20 0>;
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
  imx6ul-evk {

    pinctrl_ecspi3: ecspi3grp {
      fsl,pins = <
        MX6UL_PAD_UART2_RTS_B__ECSPI3_MISO   0x1b0b1
        MX6UL_PAD_UART2_CTS_B__ECSPI3_MOSI   0x1b0b1
        MX6UL_PAD_UART2_RX_DATA__ECSPI3_SCLK 0x1b0b1
        ~~MX6UL_PAD_UART2_TX_DATA__ECSPI3_SS0  0x1b0b1~~
        MX6UL_PAD_UART2_TX_DATA__GPIO1_IO20  0x1b0b1
      >;
    };
  };
};
```

Magistrala SPI - połączenie "loopback"



Magistrala SPI - obsługa programowa

```
long param;
ioctl(handle, SPI_IOC_xxx, &param);
```

Argument SPI_IOX_xxx	Opis/parametry
SPI_IOC_RD_MODE SPI_IOC_WR_MODE	Umożliwia ustawienie lub odczytanie trybu pracy magistrali SPI, Dopuszczalne parametry to SPI_MODE_0 do SPI_MODE_3 , które umożliwiają wybór sposobu pracy magistrali.
SPI_IOC_RD_LSB_FIRST SPI_IOC_WR_LSB_FIRST	Umożliwiają odczyt lub zmianę kolejności bitów wysyłanych magistralą. Wartość równa zero oznacza tryb pracy MSB, natomiast wartość różna od zera tryb pracy LSB
SPI_IOC_RD_BITS_PER_WORD SPI_IOC_WR_BITS_PER_WORD	Umożliwia odczytanie lub zapisanie, liczby bitów które będą przesyłane magistralą SPI podczas transmisji pojedynczego słowa
SPI_IOC_RD_MAX_SPEED_HZ SPI_IOC_WR_MAX_SPEED_HZ	Pozwala na zapisanie lub odczytanie maksymalnej dopuszczalnej prędkości transmisji SPI.

Magistrala SPI - obsługa programowa

```
long param;
ioctl(handle, SPI_IOC_xxx, &param);
```

```
ioctl(handle, SPI_IOC_MESSAGE(n), tab)
```

Argument SPI_IOX_xxx	Opis/parametry
SPI_IOC_RD_MODE SPI_IOC_WR_MODE	Umożliwia ustawienie lub odczytanie trybu pracy magistrali SPI, Dopuszczalne parametry to SPI_MODE_0 do SPI_MODE_3 , które umożliwiają wybór sposobu pracy magistrali.
SPI_IOC_RD_LSB_FIRST SPI_IOC_WR_LSB_FIRST	Umożliwiają odczyt lub zmianę kolejności bitów wysyłanych magistralą. Wartość równa zero oznacza tryb pracy MSB, natomiast wartość różna od zera tryb pracy LSB
SPI_IOC_RD_BITS_PER_WORD SPI_IOC_WR_BITS_PER_WORD	Umożliwia odczytanie lub zapisanie, liczby bitów które będą przesyłane magistralą SPI podczas transmisji pojedynczego słowa
SPI_IOC_RD_MAX_SPEED_HZ SPI_IOC_WR_MAX_SPEED_HZ	Pozwala na zapisanie lub odczytanie maksymalnej dopuszczalnej prędkości transmisji SPI.

```
struct spi_ioc_transfer {
    __u64           tx_buf;
    __u64           rx_buf;
    __u32           len;
    __u32           speed_hz;
    __u16           delay_usecs;
    __u8            bits_per_word;
    __u8            cs_change;
    __u32           pad;
};
```

ĆWICZENIE 2.9:

Magistrala 1-Wire

Magistrala 1-Wire – konfiguracja jądra systemu

- włączenie kontrolera magistrali 1-Wire w jądrze systemu:

```
Device Drivers --->
<*> Dallas's 1-wire support --->
  1-wire Bus Masters --->
    < >DS2490 USB <-> W1 transport layer for 1-wire
    < > Maxim DS2482 I2C to 1-Wire bridge
    <*> GPIO 1-wire busmaster
```

- włączenie sterownika dla urządzeń *Slave*:

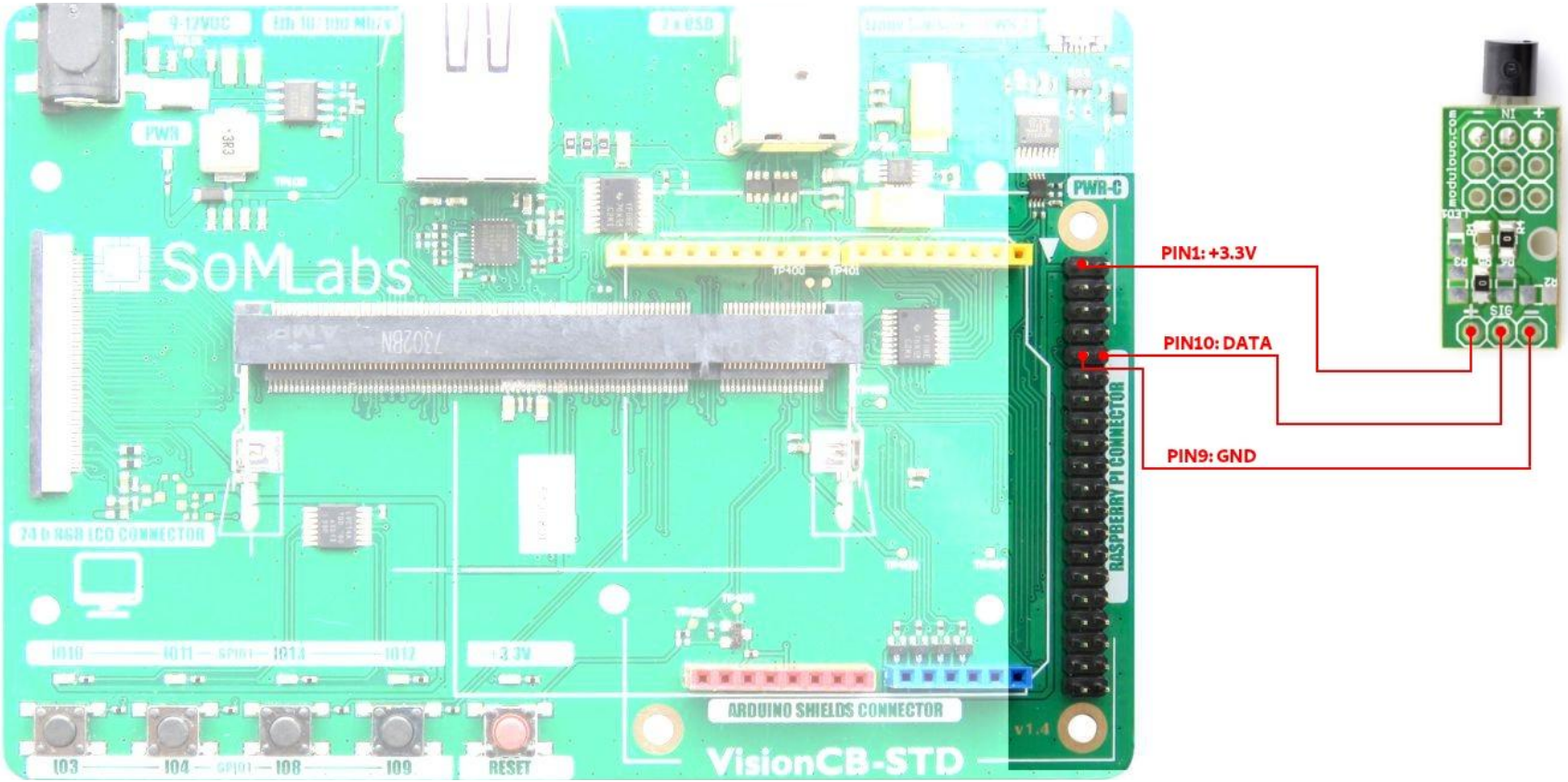
```
Device Drivers --->
<*> Dallas's 1-wire support --->
  1-wire Slaves --->
    <*> Thermal family implementation
    < > 1kb EEPROM family support (DS2431)
```

Magistrala 1-Wire – opis *Device Tree*

```
onewire {
  compatible = "w1-gpio";
  pinctrl-0 = <&pinctrl_w1_gpio>;
  pinctrl-names = "default";
  gpios = <&gpio1 29 0>;
};
```

```
&iomuxc {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_hog_1>;
    imx6ul-evk {

      pinctrl_w1_gpio: onewire {
        fsl,pins = <
          MX6UL_PAD_UART4_RX_DATA__GPIO1_IO29  0x4001b8b1
      >;
    };
  };
};
```

Magistrala 1-Wire - podłączenie czujnika DS18B20



Magistrala 1-Wire - podstawy

- z poziomu przestrzeni użytkownika, dostęp do informacji udostępnianych przez sterownik jest realizowany poprzez szereg plików dostępnych w katalogu */sys/bus/w1/devices*:

```
root@localhost:~# cd /sys/bus/w1/devices
root@localhost:/sys/bus/w1/devices# ls -l
total 0
lrwxrwxrwx 1 root root 0 May 15 15:09 w1_bus_master1
```

Magistrala 1-Wire - podstawy

- z poziomu przestrzeni użytkownika, dostęp do informacji udostępnianych przez sterownik jest realizowany poprzez szereg plików dostępnych w katalogu */sys/bus/w1/devices*:

```
root@localhost:~# cd /sys/bus/w1/devices
root@localhost:/sys/bus/w1/devices# ls -l
total 0
lrwxrwxrwx 1 root root 0 May 15 15:09 w1_bus_master1
lrwxrwxrwx 1 root root 0 May 15 15:09 28-000002f1af7c
lrwxrwxrwx 1 root root 0 May 15 15:09 28-000002f203e3
lrwxrwxrwx 1 root root 0 May 15 15:09 28-000002f218f8
```

- każdy dołączony do magistrali układ *Slave* reprezentowany jest poprzez katalog w lokalizacji */sys/bus/w1/devices* o nazwie będącej połączeniem kodu rodziny układu i jego numeru identyfikacyjnego. Liczba katalogów w lokalizacji */sys/bus/w1/devices* informuje nas więc o liczbie podłączonych sensorów

Magistrala 1-Wire - podstawy

- każdy folder oznaczony numerem identyfikacyjnym sensora zawiera szereg plików, spośród których z punktu widzenia użytkownika najważniejszy jest plik ***w1_slave***. Zawartość pliku jest tworzona dynamicznie przez jądro systemu w momencie odczytu i zawiera informację o wartości temperatury oraz sumie kontrolnej CRC:

```
root@somlabs:/sys/bus/w1/devices# cat 28-000008bedfea/w1_slave
a5 01 4b 46 7f ff 0b 10 f7 : crc=f7 YES
a5 01 4b 46 7f ff 0b 10 f7 t=26312
```

Magistrala 1-Wire - podstawy

- każdy folder oznaczony numerem identyfikacyjnym sensora zawiera szereg plików, spośród których z punktu widzenia użytkownika najważniejszy jest plik ***w1_slave***. Zawartość pliku jest tworzona dynamicznie przez jądro systemu w momencie odczytu i zawiera informację o wartości temperatury oraz sumie kontrolnej CRC:

```
root@somlabs:/sys/bus/w1/devices# cat 28-000008bedfea/w1_slave
a5 01 4b 46 7f ff 0b 10 f7 : crc=f7 YES
a5 01 4b 46 7f ff 0b 10 f7 t=26312
```

Magistrala 1-Wire - podstawy

Sterownik dostarcza również kilku prostych mechanizmów umożliwiających sterowanie pracą układu *Master*. Konfiguracja pracy odbywa się poprzez zapis/odczyt plików umieszczonych w katalogu ***w1_bus_master1***:

Magistrala 1-Wire - podstawy

Sterownik dostarcza również kilku prostych mechanizmów umożliwiających sterowanie pracą układu *Master*. Konfiguracja pracy odbywa się poprzez zapis/odczyt plików umieszczonych w katalogu ***w1_bus_master1***:

- sprawdzenie liczby podłączonych czujników:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slave_count
```

Magistrala 1-Wire - podstawy

Sterownik dostarcza również kilku prostych mechanizmów umożliwiających sterowanie pracą układu *Master*. Konfiguracja pracy odbywa się poprzez zapis/odczyt plików umieszczonych w katalogu ***w1_bus_master1***:

- sprawdzenie liczby podłączonych czujników:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slave_count
```

- odczyt numerów ID podłączonych czujników:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slaves
```

Magistrala 1-Wire - podstawy

Sterownik dostarcza również kilku prostych mechanizmów umożliwiających sterowanie pracą układu *Master*. Konfiguracja pracy odbywa się poprzez zapis/odczyt plików umieszczonych w katalogu ***w1_bus_master1***:

- sprawdzenie liczby podłączonych czujników:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slave_count
```

- odczyt numerów ID podłączonych czujników:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slaves
```

- jednokrotne skanowanie magistrali po podłączeniu/odłączeniu nowych czujników:

```
echo 1 > /sys/bus/w1/devices/w1_bus_master1/w1_master_search
```

Magistrala 1-Wire - podstawy

Sterownik dostarcza również kilku prostych mechanizmów umożliwiających sterowanie pracą układu *Master*. Konfiguracja pracy odbywa się poprzez zapis/odczyt plików umieszczonych w katalogu ***w1_bus_master1***:

- sprawdzenie liczby podłączonych czujników:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slave_count
```

- odczyt numerów ID podłączonych czujników:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slaves
```

- jednokrotne skanowanie magistrali po podłączeniu/odłączeniu nowych czujników:

```
echo 1 > /sys/bus/w1/devices/w1_bus_master1/w1_master_search
```

- wyrejestrowanie z systemu czujnika o numerze ID *28-000002f1af7c*:

```
echo 28-000002f1af7c > /sys/bus/w1/devices/w1_bus_master1/w1_master_remove
```

ĆWICZENIE 3:

Time-to-market w systemach wbudowanych, czyli wykorzystanie gotowych komponentów oprogramowania

Node.js - systemy wbudowane i JavaScript?

"Node.js to wieloplatformowe środowisko uruchomieniowe JavaScript udostępnione na licencji open-source […]"

Node.js - systemy wbudowane i JavaScript?

"Node.js to wieloplatformowe środowisko uruchomieniowe JavaScript udostępnione na licencji open-source […]"

- dostępne w systemach *Linux*, *Mac OS X*, *Windows*, *Solaris*, *FreeBSD*, *OpenBSD*, … ,

Node.js - systemy wbudowane i JavaScript?

"Node.js to wieloplatformowe środowisko uruchomieniowe JavaScript udostępnione na licencji open-source […]"

- dostępne w systemach *Linux*, *Mac OS X*, *Windows*, *Solaris*, *FreeBSD*, *OpenBSD*, … ,
- umożliwia uruchomienie kodu JavaScript poza przeglądarką internetową,

Node.js - systemy wbudowane i JavaScript?

"Node.js to wieloplatformowe środowisko uruchomieniowe JavaScript udostępnione na licencji open-source […]"

- dostępne w systemach *Linux*, *Mac OS X*, *Windows*, *Solaris*, *FreeBSD*, *OpenBSD*, … ,
- umożliwia uruchomienie kodu JavaScript poza przeglądarką internetową,
- *Node.js* nie jest serwerem, umożliwia jednak proste i szybkie utworzenie serwera oraz złożonych aplikacji internetowych,

Node.js - systemy wbudowane i JavaScript?

"Node.js to wieloplatformowe środowisko uruchomieniowe JavaScript udostępnione na licencji open-source […]"

- dostępne w systemach *Linux*, *Mac OS X*, *Windows*, *Solaris*, *FreeBSD*, *OpenBSD*, … ,
- umożliwia uruchomienie kodu JavaScript poza przeglądarką internetową,
- *Node.js* nie jest serwerem, umożliwia jednak proste i szybkie utworzenie serwera oraz złożonych aplikacji internetowych,
- ponieważ kod programu jest uruchamiany poza przeglądarką, programista ma możliwość tworzenia typowych rozwiązań "*server-side*",

Node.js - instalacja środowiska

Dla dystrybucji *Debian*, instalacja pakietu nodejs przebiega w sposób standardowy dla narzędzia *apt-get*:

```
root@localhost:~# apt-get install nodejs
Selecting previously unselected package nodejs.
Preparing to unpack .../nodejs_8.11.4~dfsg-1_armhf.deb ...
Unpacking nodejs (8.11.4~dfsg-1) ...
```

Node.js - instalacja środowiska

Dla dystrybucji *Debian*, instalacja pakietu nodejs przebiega w sposób standardowy dla narzędzia *apt-get*:

```
root@localhost:~# apt-get install nodejs
Selecting previously unselected package nodejs.
Preparing to unpack .../nodejs_8.11.4~dfsg-1_armhf.deb ...
Unpacking nodejs (8.11.4~dfsg-1) ...
```

Aby przetestować poprawność instalacji, wywołajmy komendę *nodejs -v*:

```
root@localhost:~# nodejs -v
v8.11.4
```

ĆWICZENIE 3.2:

Prosta implementacja serwera WWW z wykorzystaniem Node.js

Node.js - prosta implementacja serwera WWW

Plik */root/linux-academy/3-2/main.js*:

```
var http = require ('http');

var PORT = 8080;

var server = http.createServer (function handler (request, response) {
  response.writeHead (200, {'Content-Type': 'text/plain'});
  response.end ('Hello World!');
});

server.listen (PORT);
```

Node.js - prosta implementacja serwera WWW

Plik */root/linux-academy/3-2/main.js*:

```
var http = require ('http');

var PORT = 8080;

var server = http.createServer (function handler (request, response) {
  response.writeHead (200, {'Content-Type': 'text/html'});
  response.end ('<!DOCTYPE html><html><head>
  <script src='/socket.io/socket.io.js'></script> <script> var socket = io();
  socket.on ('time', function (data) {document.getElementById("test").innerHTML
  = data.message; }); </script></head><body><h1>Hello World!</h1>
  <p id="test">JavaScript can change HTML content.</p></body></html>');
});

server.listen (PORT);
```

ĆWICZENIE 3.3:

Serwer WWW z podziałem na funkcje front-end oraz back-end

Serwer WWW – podział *front-end/back-end*

Serwer WWW – podział *front-end/back-end*

Plik */root/linux-academy/3-3/main.js*:

```
var http = require ('http');
var fs = require ('fs');

var index = fs.readFileSync (__dirname + '/index.html');

var PORT = 8080;

var server = http.createServer (function handler (request, response) {
  response.writeHead (200, {'Content-Type': 'text/html'});
  response.end (index);
});

server.listen (PORT);
```

Serwer WWW – podział *front-end/back-end*

Plik */root/linux-academy/3-3/index.html*:

```
<!DOCTYPE html>
<html>

  <head>
  </head>

  <body>
    <h1>Hello World!</h1>
  </body>

</html>
```

ĆWICZENIE 3.4:

Komunikacja front-end<->back-end z wykorzystaniem socket.io

Komunikacja z wykorzystaniem biblioteki *socket.io*

Domyślnym managerem pakietów dla środowiska *Node.js* jest **NPM**. Jest to aplikacja wiersza poleceń, za pomocą której można instalować aplikacje dostępne w repozytorium NPM. Strona domowa aplikacji zawiera wyszukiwarkę pakietów:

https://www.npmjs.com/

Aby zainstalować pakiet wykonujemy polecenie, np.:

```
npm install socket.io
```

W aplikacji importujemy moduł poprzez wywołanie:

```
var io = require ('socket.io');
```

Komunikacja z wykorzystaniem biblioteki *socket.io*

Fragment pliku */root/linux-academy/3-4/main.js*:

```
/* ... */

var io = require ('socket.io').listen(server);

io.on ('connection', function (socket) {
  console.log ('We have new connection!');
});

function send_time() {
  io.emit ('time', {message: new Date().toISOString()});
}

setInterval (send_time, 1000);

/* ... */
```

Komunikacja z wykorzystaniem biblioteki *socket.io*

Plik */root/linux-academy/3-4/index.html*:

```
<!DOCTYPE html>
<html>
  <head>
    <script src='/socket.io/socket.io.js'></script>
    <script>
      var socket = io();
      socket.on ('time', function (data) {
        document.getElementById("test").innerHTML = data.message;
      });
    </script>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p id="test">JavaScript can change HTML content.</p>
  </body>
</html>
```

ĆWICZENIE 3.5:

Serwer WWW z odczytem danych z modułu żyroskopu

Odczyt danych z modułu żyroskopu - opcja 1

Bezpośrednia implementacja obsługi żyroskopu w kodzie serwera – z wykorzystaniem operacji na plikach lub gotowych modułów *Node.js*, instalowanych poprzez menadżer pakietów NPM. Przykładem takiego modułu może być pakiet *i2c*, instalowany poleceniem:

```
npm install i2c
```

który udostępnia proste API do realizacji niskopoziomowych operacji zapisu/odczytu danych na magistrali, np.:

```
var i2c = require('i2c');
var wire = new i2c(address, {device: '/dev/i2c-1'});
wire.writeByte(byte, function(err) {});
wire.writeBytes(command, [byte0, byte1], function(err) {});
wire.readByte(function(err, res) { // result is single byte })
wire.readBytes(command, length, function(err, res) {});
```

Odczyt danych z modułu żyroskopu - opcja 2

Implementacja obsługi procesu potomnego:

```
var spawn = require('child_process').spawn;

var child = spawn ('/tmp/gyro-i2c');

child.stdout.on ('data', function (data) {
  io.emit ('xyz', {message: data.toString().split('\n')[0]});
});

child.stderr.on ('data', function (data) {
  console.log ('stderr: ' + data);
});

child.on ('close', function (code) {
  console.log ('exit: ' + code);
});
```

Odczyt danych z modułu żyroskopu - opcja 2

Implementacja obsługi procesu potomnego:

```
var spawn = require('child_process').spawn;

var child = spawn ('/tmp/gyro-i2c');

child.stdout.on ('data', function (data) {
  io.emit ('xyz', {message: data.toString().split('\n')[0]});
});

child.stderr.on ('data', function (data) {
  console.log ('stderr: ' + data);
});

child.on ('close', function (code) {
  console.log ('exit: ' + code);
});
```

Odczyt danych z modułu żyroskopu - opcja 2

Implementacja obsługi procesu potomnego:

```
var spawn = require('child_process').spawn;

var child = spawn ('/tmp/gyro-i2c');

child.stdout.on ('data', function (data) {
  io.emit ('xyz', {message: data.toString().split('\n')[0]});
});

child.stderr.on ('data', function (data) {
  console.log ('stderr: ' + data);
});

child.on ('close', function (code) {
  console.log ('exit: ' + code);
});
```

Odczyt danych z modułu żyroskopu - opcja 2

Implementacja obsługi procesu potomnego:

```
var spawn = require('child_process').spawn;

var child = spawn ('/tmp/gyro-i2c');

child.stdout.on ('data', function (data) {
  io.emit ('xyz', {message: data.toString().split('\n')[0]});
});

child.stderr.on ('data', function (data) {
  console.log ('stderr: ' + data);
});

child.on ('close', function (code) {
  console.log ('exit: ' + code);
});
```

Odczyt danych z modułu żyroskopu - opcja 2

Implementacja obsługi procesu potomnego:

```
var spawn = require('child_process').spawn;

var child = spawn ('/tmp/gyro-i2c');

child.stdout.on ('data', function (data) {
  io.emit ('xyz', {message: data.toString().split('\n')[0]});
});

child.stderr.on ('data', function (data) {
  console.log ('stderr: ' + data);
});

child.on ('close', function (code) {
  console.log ('exit: ' + code);
});
```

Odczyt danych z modułu żyroskopu - opcja 2

Tabela z sekcji `<body>` - plik */root/linux-academy/3-5/index.html*:

```
<table>
  <tr>
    <th>X [deg]</th>
    <td><p id="x_val">---</p></td>
  </tr>

  <tr>
    <th>Y [deg]</th>
    <td><p id="y_val">---</p></td>
  </tr>

  <tr>
    <th>Z [deg]</th>
    <td><p id="z_val">---</p></td>
  </tr>
</table>
```

Odczyt danych z modułu żyroskopu - opcja 2

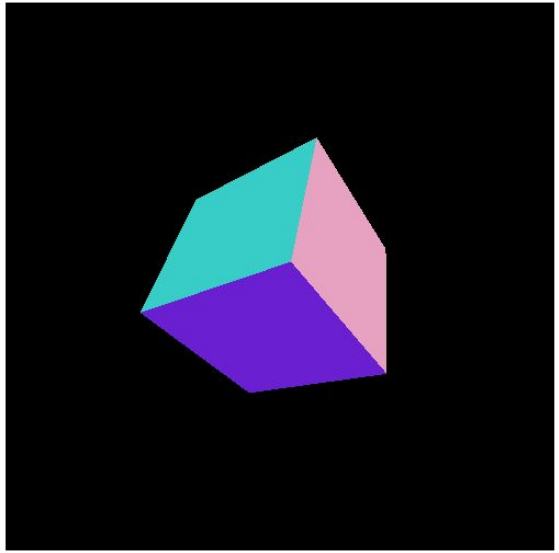
Obsługa wiadomości `xyz` w sekcji `<head>`:

```
<script>

  var socket = io();

  socket.on ('xyz', function (data) {

    var arr = data.message.split(" ");

    document.getElementById("x_val").innerHTML = arr[0];
    document.getElementById("y_val").innerHTML = arr[1];
    document.getElementById("z_val").innerHTML = arr[2];
  });

</script>
```

ĆWICZENIE 3.6:

Rozbudowa interfejsu serwera WWW o elementy grafiki 3D (Three.js)

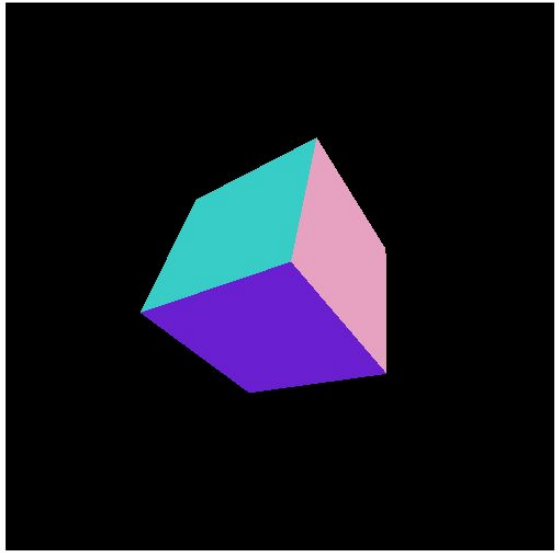
Interfejs użytkownika z elementami grafiki 3D



Gyroscope I2C

X [deg]	153.19
Y [deg]	125.43
Z [deg]	73.18

Interfejs użytkownika z elementami grafiki 3D

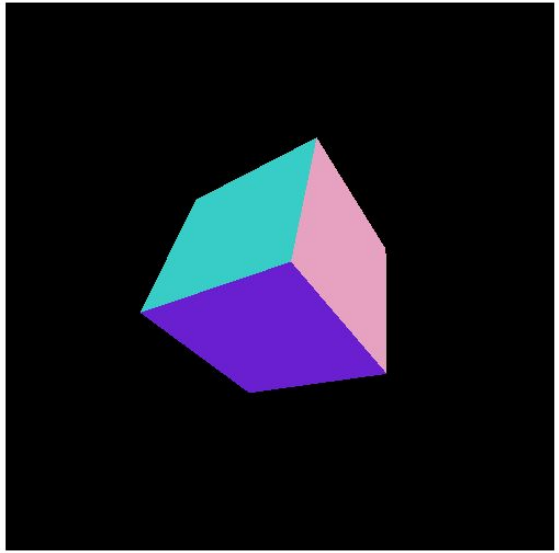


Gyroscope I2C

X [deg]	153.19
Y [deg]	125.43
Z [deg]	73.18

- **API WebGL** - oficjalne rozszerzenie języka HTML o interfejs grafiki 3D

Interfejs użytkownika z elementami grafiki 3D



Gyroscope I2C

X [deg]	153.19
Y [deg]	125.43
Z [deg]	73.18

- **API WebGL** - oficjalne rozszerzenie języka HTML o interfejs grafiki 3D

- **Biblioteka Three.js** - wysokopoziomowe API dla WebGL

```
(wget http://threejs.org/build/three.min.js)
```

Interfejs użytkownika z elementami grafiki 3D

- **Inicjalizacja *WebGL***
- Inicjalizacja buforów danych
- Rysowanie sceny
- *Shader* - inicjalizacja
- *Shader* - funkcje pomocnicze
- *Shader* - kod GLSC

```
<body>
  <button class="runButton" onclick="webGLStart();">Start</button>
  <canvas id="webgl_canvas" width="500" height="500"></canvas>
</body>

//snip

function webGLStart()
{
  var canvas = document.getElementById("webgl_canvas");

  initGL(canvas);
  initShaders();
  initBuffers();

  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.enable(gl.DEPTH_TEST);

  drawScene();
}
```

Interfejs użytkownika z elementami grafiki 3D

- Inicjalizacja *WebGL*
- **Inicjalizacja buforów danych**
- Rysowanie sceny
- *Shader* - inicjalizacja
- *Shader* - funkcje pomocnicze
- *Shader* - kod GLSC

```
var squareVertexPositionBuffer;

function initBuffers() {

  squareVertexPositionBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);

  vertices = [
     1.0,  1.0,  0.0,
    -1.0,  1.0,  0.0,
     1.0, -1.0,  0.0,
    -1.0, -1.0,  0.0,
  ];

  gl.bufferData(gl.ARRAY_BUFFER,
                new Float32Array(vertices),
                gl.STATIC_DRAW);

  squareVertexPositionBuffer.itemSize = 3;
  squareVertexPositionBuffer.numItems = 4;
}
```

Interfejs użytkownika z elementami grafiki 3D

- Inicjalizacja *WebGL*
- Inicjalizacja buforów danych
- **Rysowanie sceny**
- *Shader* - inicjalizacja
- *Shader* - funkcje pomocnicze
- *Shader* - kod GLSC

```
function drawScene()
{
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  mat4.perspective(45, gl.viewportWidth / gl.viewportHeight,
                   0.1, 100.0, pMatrix);

  mat4.identity(mvMatrix);

  mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
  gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
  gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                         squareVertexPositionBuffer.itemSize,
                         gl.FLOAT, false, 0, 0);
  setMatrixUniforms();
  gl.drawArrays(gl.TRIANGLE_STRIP, 0,
                squareVertexPositionBuffer.numItems);
}
```

Interfejs użytkownika z elementami grafiki 3D

- Inicjalizacja *WebGL*
- Inicjalizacja buforów danych
- Rysowanie sceny
- ***Shader* - inicjalizacja**
- *Shader* - funkcje pomocnicze
- *Shader* - kod GLSC

```
function initShaders()
{
  var fragmentShader = getShader(gl, "shader-fs");
  var vertexShader = getShader(gl, "shader-vs");

  shaderProgram = gl.createProgram();
  gl.attachShader(shaderProgram, vertexShader);
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);

  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
    alert("Could not initialise shaders");
  }

  gl.useProgram(shaderProgram);

  shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
                                                                "aVertexPosition");
  gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

  shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
  shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
}
```

Interfejs użytkownika z elementami grafiki 3D

- Inicjalizacja *WebGL*
- Inicjalizacja buforów danych
- Rysowanie sceny
- *Shader* - inicjalizacja
- ***Shader* - funkcje pomocnicze**
- *Shader* - kod GLSC

```
function getShader(gl, id)
{
  var shaderScript = document.getElementById(id);
  if (!shaderScript)
    return null;

  var str = "";
  var k = shaderScript.firstChild;
  while (k) {
    if (k.nodeType == 3)
      str += k.textContent;
    k = k.nextSibling;
  }
  var shader;
  if (shaderScript.type == "x-shader/x-fragment") {
    shader = gl.createShader(gl.FRAGMENT_SHADER);
  } else if (shaderScript.type == "x-shader/x-vertex") {
    shader = gl.createShader(gl.VERTEX_SHADER);
  } else {
    return null;
  }
  gl.shaderSource(shader, str);
  gl.compileShader(shader);
  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    alert(gl.getShaderInfoLog(shader));
    return null;
  }
  return shader;
}
```

Interfejs użytkownika z elementami grafiki 3D

- Inicjalizacja *WebGL*
- Inicjalizacja buforów danych
- Rysowanie sceny
- *Shader* - inicjalizacja
- *Shader* - funkcje pomocnicze
- ***Shader* - kod GLSC**

```
<script id="shader-fs" type="x-shader/x-fragment">

  precision mediump float;

  void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }

</script>

<script id="shader-vs" type="x-shader/x-vertex">

  attribute vec3 aVertexPosition;

  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;

  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
  }

</script>
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- **Definicja "płótna" w sekcji HEAD**
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- **Dołączenie biblioteki Three.js**
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- **Definicja zmiennych**
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- **Utworzenie obiektu "sceny"**
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- **Utworzenie obiektu "kamery"**
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- **Utworzenie obiektu geometrycznego**
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- **Utworzenie obiektu "materiału"**
- Połączenie figury z materiałem
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- **Połączenie figury z materiałem**
- Określenie obszaru renderowania

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

three.js

- Definicja "płótna" w sekcji HEAD
- Dołączenie biblioteki Three.js
- Definicja zmiennych
- Utworzenie obiektu "sceny"
- Utworzenie obiektu "kamery"
- Utworzenie obiektu geometrycznego
- Utworzenie obiektu "materiału"
- Połączenie figury z materiałem
- **Określenie obszaru renderowania**

```
<canvas id="mycanvas" width="500" height="500"></canvas>

<script src='three.min.js'></script>

//snip

var camera, scene, renderer;
var geometry, material, mesh;
var x, y, z;

function init() {

  scene = new THREE.Scene();

  camera = new THREE.PerspectiveCamera (70, 500/500, 0.01, 10);
  camera.position.z = 0.5;

  geometry = new THREE.BoxGeometry (0.2, 0.2, 0.2);
  material = new THREE.MeshNormalMaterial();

  mesh = new THREE.Mesh (geometry, material);
  scene.add (mesh);

  renderer = new THREE.WebGLRenderer ({ canvas: mycanvas});
  renderer.setSize (500, 500);
  document.body.appendChild (renderer.domElement);
}
```

Interfejs użytkownika z elementami grafiki 3D

Utwórzmy również funkcję `animate()`, która dokona obrotu obiektu, zgodnie z kątem obrotu zapisanym w zmiennych `x, y, z`:

```
function animate() {

  requestAnimationFrame (animate);

  mesh.rotation.x = THREE.Math.degToRad(x);
  mesh.rotation.y = THREE.Math.degToRad(y);
  mesh.rotation.z = THREE.Math.degToRad(z);

  renderer.render (scene, camera);
}
```

Interfejs użytkownika z elementami grafiki 3D

Niewielkiej modyfikacji wymaga również sam kod serwera z pliku */root/linux-academy/3-6/main.js*:

```
var url = require('url');
var server = http.createServer (function handler (request, response) {

  var pathname = url.parse(request.url).pathname;
  console.log("Request for " + pathname + " received.");

  response.writeHead (200, {'Content-Type': 'text/html'});
  if(pathname == "/") {
    var index = fs.readFileSync (__dirname + '/index.html');
    response.write (index);
  } else if (pathname == "/three.min.js") {
    var script = fs.readFileSync (__dirname + '/three.min.js');
    response.write (script);
  }
  response.end();
});
```

PRZYKŁAD A (4.1):

Sterowanie wyprowadzeniami GPIO z wykorzystaniem Node.js oraz modułu onoff

Sterowanie GPIO - moduł *onoff*

Plik */root/linux-academy/4-1/main.js*:

```
var gpio = require('onoff').Gpio;

var led = new gpio(10, 'out');
var button = new gpio(3, 'in', 'both');

led.writeSync(button.readSync());

button.watch(function (err, value) {
  led.writeSync(value);
});

function unexport() {
  led.unexport();
  button.unexport();
};

process.on('SIGINT', unexport);
```

Sterowanie GPIO - moduł *onoff*

Plik */root/linux-academy/4-1/main.js*:

```
var gpio = require('onoff').Gpio;

var led = new gpio(10, 'out');
var button = new gpio(3, 'in', 'both');

led.writeSync(button.readSync());

button.watch(function (err, value) {
  led.writeSync(value);
});

function unexport() {
  led.unexport();
  button.unexport();
};

process.on('SIGINT', unexport);
```

Sterowanie GPIO - moduł *onoff*

Plik */root/linux-academy/4-1/main.js*:

```
var gpio = require('onoff').Gpio;

var led = new gpio(10, 'out');
var button = new gpio(3, 'in', 'both');

led.writeSync(button.readSync());

button.watch(function (err, value) {
  led.writeSync(value);
});

function unexport() {
  led.unexport();
  button.unexport();
};

process.on('SIGINT', unexport);
```

Sterowanie GPIO - moduł *onoff*

Plik */root/linux-academy/4-1/main.js*:

```
var gpio = require('onoff').Gpio;

var led = new gpio(10, 'out');
var button = new gpio(3, 'in', 'both');

led.writeSync(button.readSync());

button.watch(function (err, value) {
  led.writeSync(value);
});

function unexport() {
  led.unexport();
  button.unexport();
};

process.on('SIGINT', unexport);
```

Sterowanie GPIO - moduł *onoff*

Plik */root/linux-academy/4-1/main.js*:

```
var gpio = require('onoff').Gpio;

var led = new gpio(10, 'out');
var button = new gpio(3, 'in', 'both');

led.writeSync(button.readSync());

button.watch(function (err, value) {
  led.writeSync(value);
});

function unexport() {
  led.unexport();
  button.unexport();
};

process.on('SIGINT', unexport);
```

Sterowanie GPIO - moduł *onoff*

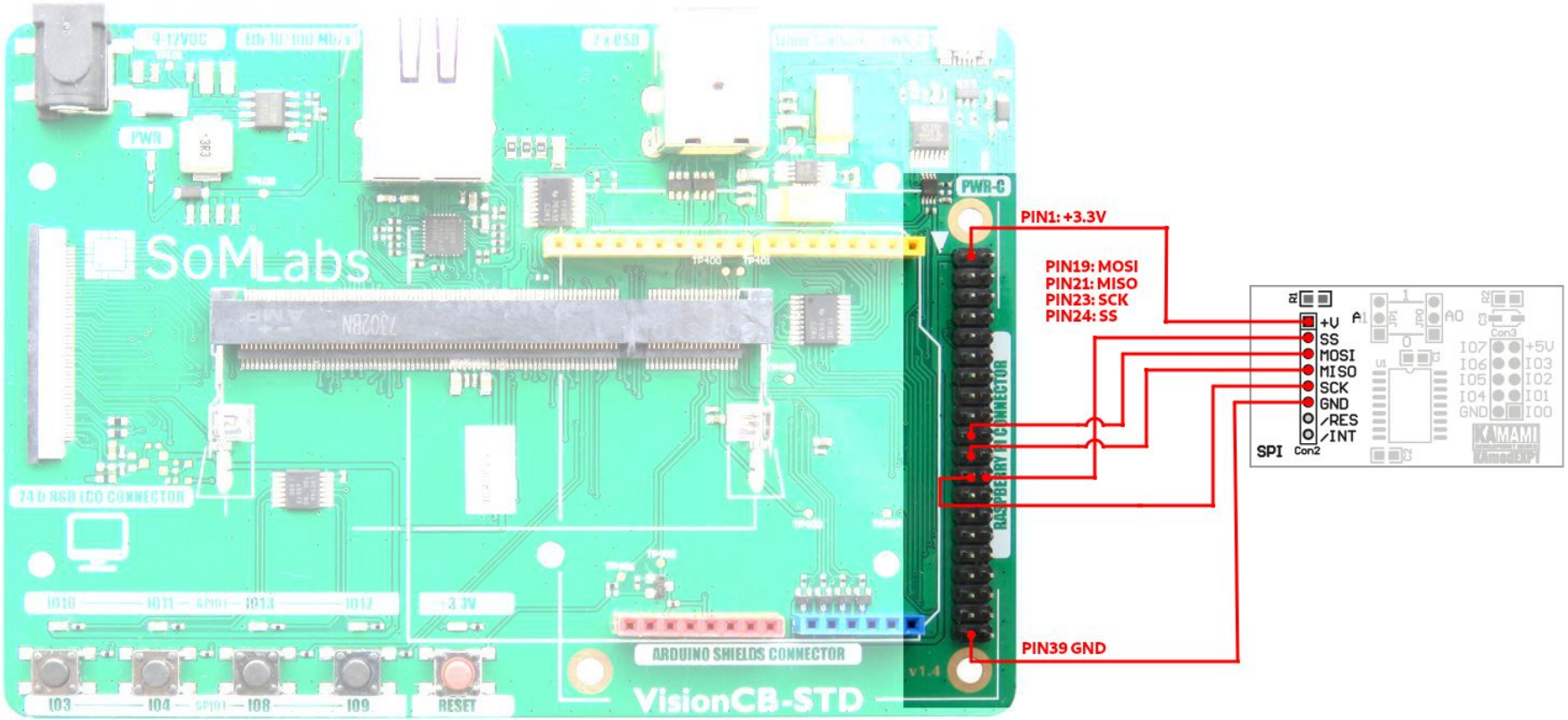
Plik */root/linux-academy/4-1/main.js*:

```
var gpio = require('onoff').Gpio;

var led = new gpio(10, 'out');
var button = new gpio(3, 'in', 'both');

led.writeSync(button.readSync());

button.watch(function (err, value) {
  led.writeSync(value);
});

function unexport() {
  led.unexport();
  button.unexport();
};

process.on('SIGINT', unexport);
```

PRZYKŁAD B (4.2):

Sterowanie wyprowadzeniami GPIO modułu KAmodEXP1 (MCP23S08) z wykorzystaniem Node.js

Podłączenia ekspandera *KAmodEXP1*



Moduł KAmodEXP1 (MCP23S08) – sterownik

- włączenie sterownika układu MCP23S08:

```
Device Drivers --->
  -*- GPIO Support --->
    SPI GPIO expanders --->
      <*> Microchip MCP23xxx I/O expander
```

Moduł KAmodEXP1 (MCP23S08) – sterownik

- włączenie sterownika układu MCP23S08:

```
Device Drivers --->
  -*- GPIO Support --->
    SPI GPIO expanders --->
      <*> Microchip MCP23xxx I/O expander
```

- aktualny opis *Device Tree*:

```
&ecspi3 {
 fsl,spi-num-chipselects = <1>;
 cs-gpios = <&gpio1 20 0>;
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };
};
```

Moduł KAmodEXP1 (MCP23S08) – sterownik

- włączenie sterownika układu MCP23S08:

```
Device Drivers --->
  -*- GPIO Support --->
    SPI GPIO expanders --->
      <*> Microchip MCP23xxx I/O expander
```

- aktualny opis *Device Tree*:

- uzupełniony opis *Device Tree*:

```
&ecspi3 {
 fsl,spi-num-chipselects = <2>;
 cs-gpios = <&gpio1 0 0> <&gpio1 20 0>;
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };

 gpiom1: gpio@1 {
      compatible = "microchip,mcp23s08";
      gpio-controller;
      microchip,spi-present-mask = <0x01>;
      reg = <1>;
      spi-max-frequency = <1000000>;
 };
};
```

Moduł KAmodEXP1 (MCP23S08) – sterownik

- po poprawnej konfiguracji jądra systemu oraz przygotowaniu opisu *Device Tree*, w katalogu */sys/class/gpio* zostanie utworzony nowy wpis reprezentujący nowy kontroler GPIO:

```
root@localhost:~# cd /sys/class/gpio/
root@localhost:/sys/class/gpio# ls -l
total 0
--w------- 1 root root 4096 Oct 1 19:40 export
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip0
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip128
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip32
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip504
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip64
lrwxrwxrwx 1 root root 0    Oct 1 19:40 gpiochip96
--w------- 1 root root 4096 Oct 1 19:40 unexport
```

- wyeksportowanie do przestrzeni użytkownika wyprowadzenia ***D0*** modułu ***KAmodEXP1***:

```
root@localhost:~# echo 10 > /sys/class/gpio/export
```

Moduł KAmodEXP1 (MCP23S08) – serwer WWW

Plik */root/linux-academy/4-2/main.js*:

```
var led0 = new gpio(504, 'out');
...
var btn0 = new gpio(3, 'in', 'both');
...

var index = fs.readFileSync (__dirname + '/index.html');
var PORT = 8080;

var server = http.createServer (function handler (request, response) {
  response.writeHead (200, {'Content-Type': 'text/html'});
  response.end (index);
});
var io = require ('socket.io').listen(server);

io.on ('connection', function (socket) {
  socket.on('led0', function(data) { led0.writeSync(data); });
  ...
});

btn0.watch(function (err, value) {
  io.emit ('btn0', value);
});
...
server.listen (PORT);
```

PRZYKŁAD C (4.3):

Moduł KAmodEXP1 (MCP23S08) - sterownik LED Class Driver

KAmodEXP1 (MCP23S08) – *LED Class Driver*

```
&ecspi3 {
 fsl,spi-num-chipselects = <2>;
 cs-gpios = <&gpio1 0 0> <&gpio1 20 0>;
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };

 gpiom1: gpio@1 {
      compatible = "microchip,mcp23s08";
      gpio-controller;
      microchip,spi-present-mask = <0x01>;
      reg = <1>;
      spi-max-frequency = <1000000>;
 };
};
```

KAmodEXP1 (MCP23S08) – *LED Class Driver*

```
&ecspi3 {
 fsl,spi-num-chipselects = <2>;
 cs-gpios = <&gpio1 0 0> <&gpio1 20 0>;
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };

 gpiom1: gpio@1 {
      compatible = "microchip,mcp23s08";
      gpio-controller;
      microchip,spi-present-mask = <0x01>;
      reg = <1>;
      spi-max-frequency = <1000000>;
 };
};
```

KAmodEXP1 (MCP23S08) – *LED Class Driver*

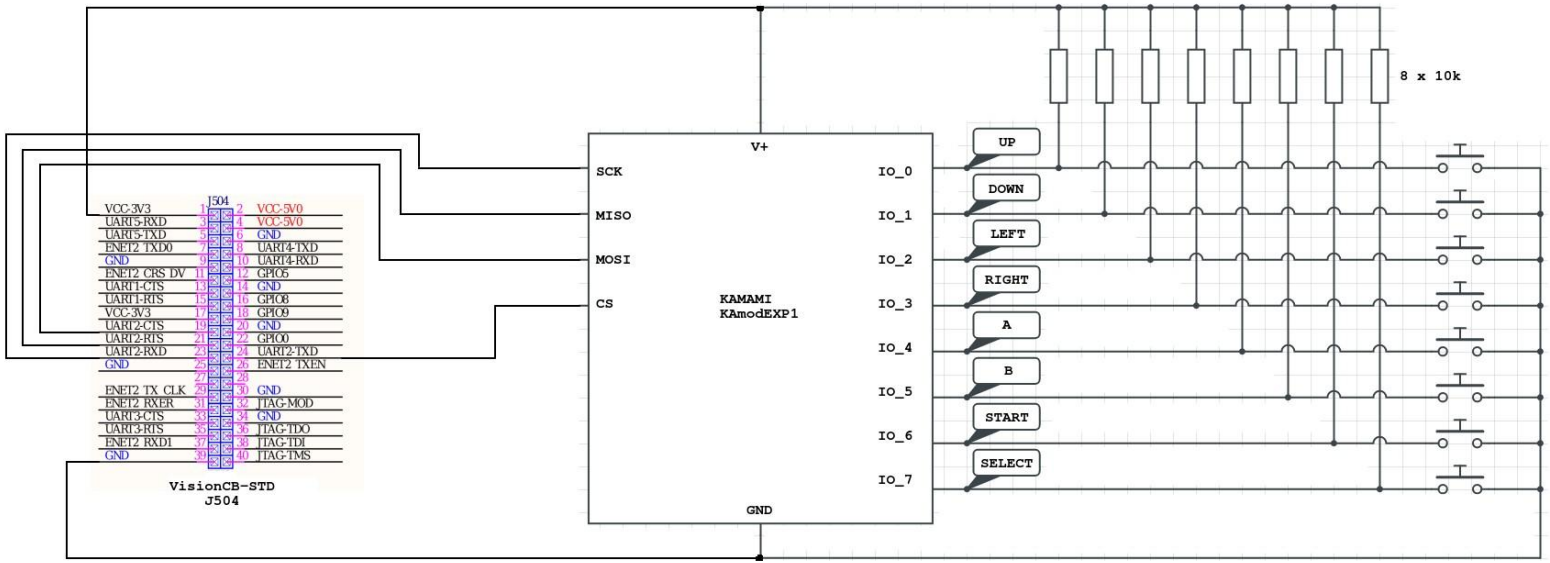
```
&ecspi3 {
 fsl,spi-num-chipselects = <2>;
 cs-gpios = <&gpio1 0 0> <&gpio1 20 0>;
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi3>;
 status = "okay";

 spidev@0 {
      compatible = "spidev";
      spi-max-frequency = <20000000>;
      reg = <0>;
 };

 gpiom1: gpio@1 {
      compatible = "microchip,mcp23s08";
      gpio-controller;
      microchip,spi-present-mask = <0x01>;
      reg = <1>;
      spi-max-frequency = <1000000>;
 };
};
```

```
leds {
      compatible = "gpio-leds";
      pinctrl-0 = <&pinctrl_gpio_leds>;
      pinctrl-names = "default";

      led3 {
                  label = "led3";
                  gpios = <&gpio1 13 GPIO_ACTIVE_HIGH>;
                  linux,default-trigger = "heartbeat";
      };

      /***/

      kamod-led0 {
                  label = "kamod-led0";
                  gpios = <&gpiom1 0 GPIO_ACTIVE_HIGH>;
                  linux,default-trigger = "heartbeat";
      };

      /***/

      kamod-led7 {
                  label = "kamod-led7";
                  gpios = <&gpiom1 7 GPIO_ACTIVE_HIGH>;
                  linux,default-trigger = "heartbeat";
      };
};
```

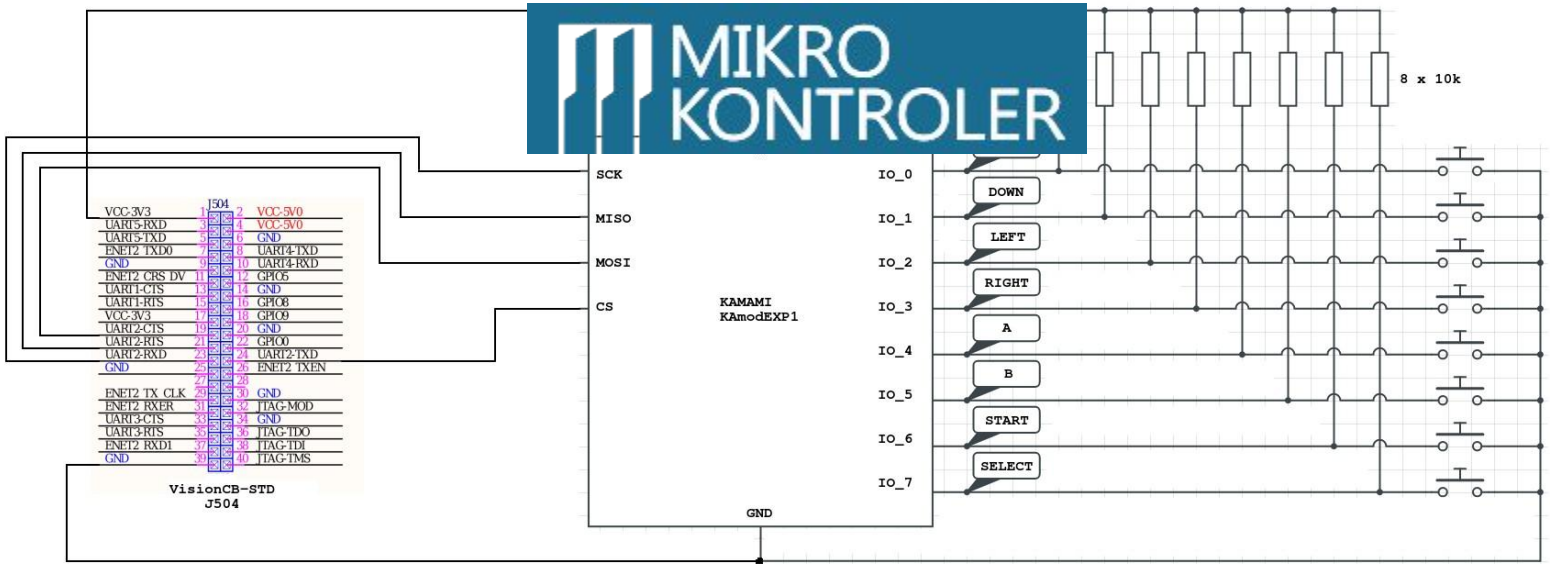
Czy to nareszcie koniec? ;)


MARIO
000000
×00
WORLD
1-1
TIME
SUPER
MARIO BROS.
©1985 NINTENDO
1 PLAYER GAME
2 PLAYER GAME
TOP- 000000
SONY
SELECT
START
VisionCB-STD
SOM labs


MARIO
WORLD
TIME
1-1
SUPER
MARIO BROS.


J504
VCC-3V3 1
UART5-RXD 3
UART5-TXD 5
ENET2 TXD0 7
GND 9
ENET2 CRS DV 11
UART1-CTS 13
UART1-RTS 15
VCC-3V3 17
UART2-CTS 19
UART2-RTS 21
UART2-RXD 23
GND 25
27
ENET2 TX CLK 29
ENET2 RXER 31
UART3-CTS 33
UART3-RTS 35
ENET2 RXD1 37
GND 39
2 VCC-5V0
4 VCC-5V0
6 GND
8 UART4-TXD
10 UART4-RXD
12 GPIO5
14 GND
16 GPIO8
18 GPIO9
20 GND
22 GPIO0
24 UART2-TXD
26 ENET2 TXEN
28
30 GND
32 JTAG-MOD
34 GND
36 JTAG-TDO
38 JTAG-TDI
40 JTAG-TMS
VisionCB-STD
J504
V+
SCK
MISO
MOSI
CS
KAMAMI
KAmodEXP1
IO_0
IO_1
IO_2
IO_3
IO_4
IO_5
IO_6
IO_7
GND
UP
DOWN
LEFT
RIGHT
A
B
START
SELECT
8 x 10k


MARIO
WORLD
TIME
SUPER
MARIO BROS.


MIKRO
KONTROLER
8 x 10k
SCK
MISO
MOSI
CS
KAMAMI
KAmodEXP1
IO_0
IO_1
IO_2
IO_3
IO_4
IO_5
IO_6
IO_7
GND
DOWN
LEFT
RIGHT
A
B
START
SELECT
J504
VCC-3V3
UART5-RXD
UART5-TXD
ENET2 TXD0
GND
ENET2 CRS DV
UART1-CTS
UART1-RTS
VCC-3V3
UART2-CTS
UART2-RTS
UART2-RXD
GND
ENET2 TX CLK
ENET2 RXER
UART3-CTS
UART3-RTS
ENET2 RXD1
GND
VCC-5V0
VCC-5V0
GND
UART4-TXD
UART4-RXD
GPIO5
GND
GPIO8
GPIO9
GND
GPIO0
UART2-TXD
ENET2 TXEN
GND
JTAG-MOD
GND
JTAG-TDO
JTAG-TDI
JTAG-TMS
VisionCB-STD
J504