

POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI

INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ

ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ



SERWER IoT

RASPBERRY PI I APLIKACJE WIERSZA POLECEŃ

APLIKACJE MOBILNE I WBUDOWANE DLA  
INTERNETU RZECZY

MATERIAŁY DO ZAJĘĆ LABORATORYJNYCH

DR INŻ. DOMINIK ŁUCZAK, MGR INŻ. ADRIAN WÓJCIK

DOMINIK.LUCZAK@PUT.POZNAN.PL  
ADRIAN.WOJCIK@PUT.POZNAN.PL

## I. CEL ZAJĘĆ

### WIEDZY

Celem zajęć jest zapoznanie z:

- podstawami struktury systemu operacyjnego Linux (Raspberry Pi OS),
- podstawami składni powłoki systemowej *bash*,
- podstawami składni języka Python3.
- podstawami komunikacji sieciowej w systemie Linux (Raspberry Pi OS).

### UMIEJĘTNOŚCI

Celem zajęć jest nabycie umiejętności w zakresie:

- konfiguracji interfejsów sieciowych w systemie Linux,
- tworzenia aplikacji CLI za pomocą skryptów powłoki systemowej *bash*,
- tworzenia aplikacji CLI za pomocą języków C i C++,
- tworzenia aplikacji CLI za pomocą języka Python,
- obsługi wejść i wyjść cyfrowych komputera jednopłytkowego Raspberry Pi,
- obsługa modułów PWM komputera jednopłytkowego Raspberry Pi,
- konfiguracja serwera WWW w systemie Linux,
- wykorzystanie aplikacji CLI w skryptach PHP / Python po stronie serwera.

### KOMPETENCJE SPOŁECZNYCH

Celem zajęć jest kształtowanie właściwych postaw:

- ugruntowanie rozumienia roli i aplikacji komunikacji sieciowej w systemach informatycznych oraz związanych z nimi zabezpieczeń,
- ugruntowanie rozumienia i świadomości znaczenia pozatechnicznych aspektów i skutków działalności inżyniera i związaną z tym odpowiedzialność za podejmowane decyzje,
- prawidłowej komunikacji technicznej w zakresie tworzenia oprogramowania.
- dobór właściwej technologii i narzędzi programistycznych do zadanego problemu,

## II. POLECENIA KOŃCOWE

Wykonaj [zadania laboratoryjne](#) zgodnie z poleceniami i wskazówkami prowadzącego, pracując samodzielnie lub w dwuosobowym zespole. **Zachowaj zasady bezpieczeństwa podczas pracy!** Wykonaj raport laboratoryjny, dokumentujący prawidłowe wykonanie zadań. Wymogi redakcyjne oraz szablony sprawozdania dostępne są na platformie *eKursy*. Raport oceniany jest w dwóch kategoriach: realizacja zadań i spełnienie wymogów redakcyjnych. Zrealizowane zadania są oceniane zero-jedynkowo. Spełnienie wymogów redakcyjnych oceniane jest procentowo. Sprawozdanie należy przesłać jako rozwiązanie zadania na platformie *eKursy* do niedzieli, 4 kwietnia 2021 do 23:59.

### III. PRZYGOTOWANIE DO ZAJĘĆ

#### A) ZAPOZNANIE Z PRZEPISAMI BHP

Wszystkie informacje dotyczące instrukcji BHP laboratorium są zamieszczone w sali laboratoryjnej oraz na stronie Zakładu [1]. Wszystkie nieścisłości należy wyjaśnić z prowadzącym laboratorium. Wymagane jest zaznajomienie i zastosowanie do regulaminu.

Na zajęcia należy przyjść przygotowanym zgodnie z tematem zajęć. Obowiązuje również materiał ze wszystkich odbytych zajęć.

#### B) WPROWADZENIE DO RASPBERRY PI

Raspberry Pi (RPi) to seria jednopłytkowych komputerów (ang. single-board computer, SBC) czyli urządzeń, które na jednym obwodzie drukowanym zawierają mikroprocesor(y), pamięć oraz urządzenia wejścia/wyjścia. Urządzenie to oparte jest o procesor z rodziny ARM. Raspberry Pi nie ma dysku twardego - w celu załadowania systemu operacyjnego i przechowywania danych oferuje złącze dla kart microSD lub na pamięci USB. Poza tym wyposażony jest w złącze z GPIO oraz popularnymi interfejsami elektronicznymi - UART, I2C i SPI [2]. Jest to bardzo popularna platforma zarówno w dydaktyce informatyki jak i szybkim prototypowniu. Istotną cechą Raspberry Pi jest również rozbudowana i aktywna społeczność użytkowników [3], czego konsekwencją jest szeroki wybór dostępnych aplikacji i bibliotek programistycznych. Rekomendowanym (ale bynajmniej nie jedynym!) system operacyjnym dla RPi jest system Raspberry Pi OS oparty o dystrybucje Linuksa - Debian.

#### C) KONFIGURACJA INTERFEJSÓW SIECIOWYCH

Raspberry Pi umożliwia podłączenie klawiatury i myszy (via USB) oraz monitora (via HDMI). Wyposażone jest jednak również w interfejsy sieciowe: Ethernet (**eth0**) oraz WiFi (**wlan0**). Z tego powodu dużo częściej spotykanym sposobem obsługi RPi jest komunikacja za pośrednictwem protokołu SSH (ang. secure shell), służącego do zdalnego łączenia się z komputerami z wykorzystaniem interfejsów sieciowych. Prawidłowa konfiguracja interfejsów sieciowych jest niezbędna do skutecznego wykorzystania platformy.

Podstawową komendą do konfiguracji interfejsów sieciowych jest polecenie `ifconfig` [4] (nie mylić z analogiczną komendą w systemie Windows - `ipconfig`!).

W celu uzyskania permanentnego (tj. zachowanego po ponownym uruchomieniu urządzenia) *statycznego* adresu IP należy dokonać edycji systemowego pliku `/etc/dhcpd.conf` i dodać konfigurację wg. wzorca zawartego w pliku (listing 1).

*Listing 1. Plik `/etc/dhcpd.conf` - statyczne IP interfejsu **eth0***

```
01. interface eth0
02. static ip_address=192.168.1.15
03. static routers=192.168.1.1
04. static domain_name_servers=8.8.8.8
```

Wykorzystanie WiFi wymaga konfiguracji w postaci dodania identyfikatora sieci SSID (ang. service set identifier) oraz hasła w pliku `/etc/wpa_supplicant/wpa_supplicant.conf` - przykład na listingu 2.

*Listing 2. Plik `/etc/wpa_supplicant/wpa_supplicant.conf` - SSID i hasło sieci bezprzewodowej*

```
01. network={
02.     ssid="M323_01"
03.     psk="labM32301"
04. }
```

**UWAGA!** Należy pamiętać, że nawiązanie połączenia między RPi a innym urządzeniem wymaga aby przynajmniej po jednym interfejsie sieciowym obu urządzeń było w tej samej sieci!

## D) APLIKACJE WIERSZA POLECEŃ

Dwa podstawowe warianty systemu Raspberry Pi OS to [5]:

- wersja Desktopowa - wyposażona w nakładkę graficzną (z możliwością jej wyłączenia),
- wersja Lite (*headless*) - oparta wyłącznie o wiersz poleceń (ang. command line interpreter/interface, CLI).

W ramach kursu system Raspberry Pi OS obsługiwany będzie wyłącznie z poziomu wiersza poleceń. Konieczne będzie więc nabycie umiejętności tworzenia i obsługi aplikacji uruchamianych z poziomu wiersza poleceń. Zostaną do tego wykorzystane trzy różne narzędzia:

- skrypty powłoki systemowej **bash** [6],
- język interpretowany **Python** (wersja 3.5) [7],
- język programowania **C++** [8] z kompilatorem **g++** [9][10].

### 1. BASH

Bash stanowi powłokę systemową (ang. system shell) systemu UNIX. Bash jest domyślnie dostępny w większości dystrybucji systemu Linux oraz w systemie macOS. Na listingu 3. przedstawiono prostą aplikację „Hello World”. Pierwszą linijką skryptu jest ścieżka odpowiedniego interpretera. Na listingu 4. przedstawiono przykład skryptu obsługującego argumenty wejściowe skryptu z wykorzystaniem metody *getopts* [11].

*Listing 3. Bash „Hello World”.*

```
01. #!/bin/bash
02. # Simple 'Hello World' example
03. echo "Hello World!"
```

*Listing 4. Bash - przykład wykorzystania funkcji getopt.*

```
01. #!/bin/bash
02. ***
03. #*****
04. #* @file      /cli_examples/bash/bash_args.sh
05. #* @author    Adrian Wojcik
06. #* @version   V1.0
07. #* @date      14-Mar-2020
08. #* @brief     Raspberry Pi CLI apps example: bash with getopt
09. #*****
10.
11. nonoptionargs=()
12. flagstate=("RESET" "SET")
13. aflag=0
14. bflag=0
15. cvalue=""
16. echo "Bash CLI example"
17. # standard while/case procedure for 'getopt' function
18. while [ $# -gt 0 ]; do
19.     while getopts ":abc:" opt; do
20.         case $opt in
21.             a)
22.                 aflag=1 ;;
```

```
23.      b)
24.          bflag=1 ;;
25.      c)
26.          cvalue=$OPTARG ;;
27.      \?)
28.          echo "option '$_OPTARG' not recognized"
29.          exit 1 ;;
30.      : )
31.          echo "option '$_OPTARG' requires argument"
32.          exit 1 ;;
33.  esac
34.  done
35.  shift $((OPTIND-1))
36.
37.  while [ $# -gt 0 ] && ! [[ "$1" =~ ^- ]]; do
38.      nonoptionargs=("${nonoptionargs[@]}" "$1")
39.      shift
40.  done
41.  done
42.
43.  # Printing results
44.  echo "a(flag)=${flagstate[$aflag]}"
45.  echo "b(flag)=${flagstate[$bflag]}"
46.  echo "c(value)=${cvalue}"
47.  index=1
48.  if [ ${#nonoptionargs[@]} -gt 0 ]; then
49.      for noa in "${nonoptionargs[@]}"
50.      do
51.          echo "Non-option argument #$index: $noa"
52.          index=$((index+1))
53.      done
54.  fi
```

**UWAGA!** W celu umożliwienia wykonania skryptu, należy nadać mu uprawnienia za pomocą komendy `chmod ug+x nazwa_skryptu.sh`.

**UWAGA!** Edytując skrypty systemu Linux z poziomu innego systemu operacyjnego (np. Windows) należy upewnić się, czy kodowanie znaków jest odpowiednie dla systemów UNIX. Edytory kodów źródłowych (np. Notepad++) umożliwiają wybór takiej konfiguracji.

Obliczenia w Bash dokonywane są **tylko** na liczbach całkowitych, nie ma również przeprowadzanej kontroli przepełnienia zmiennych (ang. overflow). Jeżeli dany problem wymaga rozbudowanych operacji na liczbach zmiennoprzecinkowych to Bash nie jest adekwatnym narzędziem do jego rozwiązania. W przypadku prostych obliczeń matematycznych angażujących liczby zmiennoprzecinkowe wykorzystać można aplikację kalkulatora `bc` [12]. Na listingu 5. przedstawiono przykładowe wykorzystanie kalkulatora `bc` do realizacji funkcji liniowej w skrypcie Bash.

*Listing 5. Bash - przykład wykorzystania kalkulatora `bc`.*

```
01.  #!/bin/bash
02.  # temperature in degrees Celsius
03.  tempC="20.5"
04.  # temperature in degrees Fahrenheit
05.  tempF=$(echo "tempf=1.8*$tempC+32.0;tempf" | bc);
06.  # display result
07.  echo "Temperature in degrees Celsius: $tempC *C"
08.  echo "temperature in degrees Fahrenheit: $tempF *F"
```

## 2. PYTHON

Python to popularny język interpretowany ogólnego przeznaczenia, tj. język, który zazwyczaj jest wykonywany (*interpretowany*) jako skrypt przez interpreter, a nie kompilowany do pliku binarnego przez kompilator. Popularność języka Python jest efektem m.in. dostępnością bardzo dużej liczby bibliotek do szerokiej gamy aplikacji. Stosowanie skryptów w języku Python wymaga instalacji interpretera w systemie, jednak w wielu dystrybucjach systemu Linux jest to preinstalowane oprogramowanie. Na listingu 6. przedstawiono prostą aplikację „Hello World”. Pierwszą linią skryptu jest ścieżka odpowiedniego interpretera. Na listingu 7. przedstawiono przykład skryptu obsługującego argumenty wejściowe skryptu z wykorzystaniem metody *getopt* [13].

*Listing 6. Python „Hello World”.*

```
01. #!/usr/bin/python3
02. # Simple 'Hello World' example
03. print("Hello World!")
```

*Listing 7. Python - przykład wykorzystania funkcji getopt.*

```
01. #!/usr/bin/python3
02. ***
03. ****
04. ** @file      /cli_examples/python/python_args.py
05. ** @author    Adrian Wojcik
06. ** @version   V1.0
07. ** @date      14-Mar-2020
08. ** @brief     Raspberry Pi CLI apps example: Python 3 with getopt
09. ****
10.
11. import sys
12. import getopt
13. print("Python3 CLI example")
14.
15. flagstate = ["RESET", "SET"]
16. aflag = 0
17. bflag = 0
18. cvalue = ''
19.
20. sysarg = sys.argv[1:]
21.
22. # standard try/except/for procedure for 'getopt' function
23. try:
24.     opts, args = getopt.getopt(sysarg, 'abc:')
25. except getopt.GetoptError as err:
26.     print(err)
27.     sys.exit(1)
28.
29. for opt, arg in opts:
30.     if opt in '-a':
31.         aflag = 1
32.     elif opt in '-b':
33.         bflag = 1
34.     elif opt in '-c':
35.         cvalue = arg
36.
37. # Printing results
38. print('a(flag)=', flagstate[aflag])
39. print('b(flag)=', flagstate[bflag])
40. if cvalue:
41.     print('c(value)=', cvalue)
42.
43. for arg in args:
44.     sysarg.remove(arg[0])
```

```
45.     if arg[1]:
46.         sysarg.remove(arg[1])
47.
48.     index = 1
49.     for arg in sysarg:
50.         print('Non-option_argument_', index, "_=", arg, sep="")
51.         index = index + 1
```

### 3. C++

C++ to wieloparadygmatowy język programowania ogólnego przeznaczenia. Jego podstawową właściwością jest wysoki poziom niezależności od platformy sprzętowej. Istotną cechą jest również kompatybilność z językiem C - biblioteki języka C++ mogą być pisane w C. Wykorzystanie aplikacji napisanych w C++ wymaga kompilacji kodu źródłowego, a więc instalacji kompilatora w systemie. Na listingu 8. przedstawiono prostą aplikację „Hello World”. Na listingu 9 przedstawiono skrypt służący do kompilacji kodu źródłowego za pomocą g++ [9][10]. Na listingu 10. przedstawiono przykład programu obsługującego argumenty wejściowe skryptu z wykorzystaniem metody *getopt* [14].

*Listing 8. C++ „Hello World”.*

```
01. // Simple 'Hello World' example
02. #include <iostream>
03.
04. int main(int argc, char *argv[])
05. {
06.     std::cout << "Hello_World!" << std::endl;
07.     return 0;
08. }
```

*Listing 9. Kompilacja programu „Hello World” za pomocą g++.*

```
01. #!/bin/bash
02. # C++ 'Hello World' program compilation with g++
03. g++ -Wall -pedantic cpp_HelloWorld.cpp -o cpp_HelloWorld
```

*Listing 10. C++ - przykład wykorzystania funkcji getopt.*

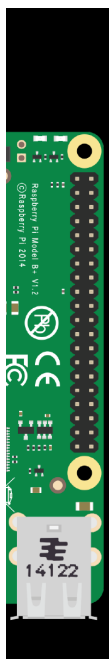
```
01. /**
02.  *****
03.  * @file      /cli_examples/cpp/cpp_args.cpp
04.  * @author    Adrian Wojcik
05.  * @version   V1.0
06.  * @date      14-Mar-2020
07.  * @brief     Raspberry Pi CLI apps example: C++ with getopt
08.  *****
09.  */
10.
11. #include <ctype.h>
12. #include <stdlib.h>
13. #include <unistd.h>
14. #include <iostream>
15.
16. int main(int argc, char *argv[])
17. {
18.     const char *flagstate[2] = { "RESET", "SET" };
19.     int aflag = 0;
20.     int bflag = 0;
21.     char *cvalue = nullptr;
22.     int index;
23.
24.     int arg;
25. }
```

```
26. opterr = 0;
27.
28. std::cout << "C++_CLI_example" << std::endl;
29.
30. /* Standard while/switch procedure for 'getopt' function */
31. while((arg = getopt (argc, argv, "abc:")) != -1)
32. {
33.     switch(arg)
34.     {
35.         case 'a':
36.             aflag = 1;
37.             break;
38.         case 'b':
39.             bflag = 1;
40.             break;
41.         case 'c':
42.             cvalue = optarg;
43.             break;
44.         case '?':
45.             if(optopt == 'c')
46.                 std::cerr << "option_" << static_cast<char>(optopt)
47.                             << "_requires_argument" << std::endl;
48.             else if(isprint(optopt))
49.                 std::cerr << "option_" << static_cast<char>(optopt)
50.                             << "_not_recognized" << std::endl;
51.             else
52.                 std::cerr << "option_character_\\x" << optopt
53.                             << "_not_recognized" << std::endl;
54.             return 1;
55.         default:
56.             abort();
57.     }
58. }
59.
60. /* Printing results */
61. std::cout << "a_(flag)_=" << flagstate[aflag] << std::endl;
62. std::cout << "b_(flag)_=" << flagstate[bflag] << std::endl;
63. if(cvalue != nullptr)
64.     std::cout << "c_(value)_=" << cvalue << std::endl;
65.
66. for (index = optind; index < argc; index++)
67.     std::cout << "Non-option_argument_#" << (index-optind+1)
68.               << ":_ " << argv[index] << std::endl;
69.
70. return 0;
71. }
```



## E) OBSŁUGA WEJŚĆ I WYJŚĆ OGÓLNEGO PRZEZNACZENIA

RPi wyposażone jest w 40-pinowe złącze wyprowadzające wejścia/wyjścia cyfrowe ogólnego przeznaczenia (GPIO), 2 kanały PWM, oraz po jednym z interfejsów: UART, I2C i SPI (rys. 1).

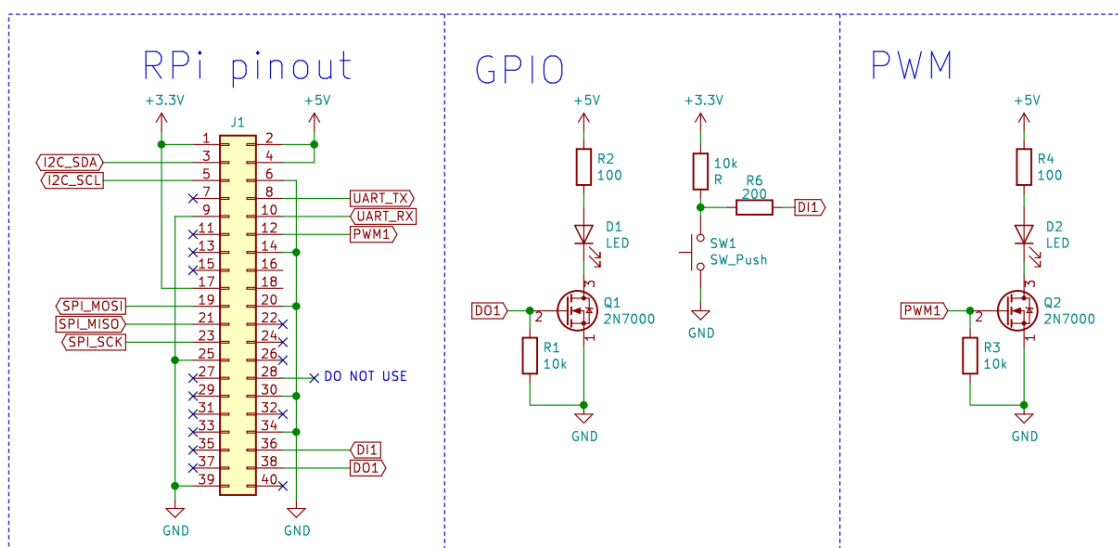


Peripherals	GPIO	Particle	Pin #	Pin #	Particle	GPIO	Peripherals
			1	X	2		
3.3V			3	X	4	5V	
I2C	GPIO2	SDA	5	X	6	5V	
	GPIO3	SCL	7	X	8	GND	
Digital I/O	GPIO4	DO	9	X	10	TX	GPIO14
GND			11	X	12	RX	GPIO15
Digital I/O	GPIO17	D1	13	X	14	D9/A0	GPIO18
Digital I/O	GPIO22	D2	15	X	16	GND	
Digital I/O	GPIO27	D3	17	X	18	D10/A1	GPIO23
3.3V			19	X	20	D11/A2	GPIO24
SPI	GPIO10	MOSI	21	X	22	D12/A3	GPIO25
	GPIO9	MISO	23	X	24	CE0	GPIO8
	GPIO11	SCK	25	X	26	CE1	GPIO7
GND			27	X	28	GND	
DO NOT USE	ID_SD	DO NOT USE	29	X	30	DO NOT USE	ID_SC
Digital I/O	GPIO5	D4	31	X	32	D13/A4	GPIO12
Digital I/O	GPIO6	D5	33	X	34	GND	
PWM 2	GPIO13	D6	35	X	36	D14/A5	GPIO16
PWM 2	GPIO19	D7	37	X	38	D15/A6	GPIO20
Digital I/O	GPIO26	D8	39	X	40	D16/A7	GPIO21
GND							

Rys. 1. Pinout złącza Raspberry Pi.

W systemach Linux dostęp do sprzętu jest analogiczny do odczytu/zapisu do plików reprezentujących ten sprzęt. Można więc obsłużyć wymienione peryferia wykonując odpowiednie komendy `echo` (zapis) oraz `cat` (odczyt). W praktyce jednak dostępne są biblioteki programistyczne zapewniające wygodne API dla podstawowych czynności związanych z danym peryferium, np. WiringPi [15].

Prezentowane poniżej przykłady zrealizowane zostały z wykorzystaniem zewnętrznych elementów jak na schemacie na rys. 2.



Rys. 2. Schemat prostych wyjść cyfrowych (LED) oraz wejścia cyfrowego (przycisk monostabilny).

## 1. GPIO

Poniżej przedstawiono trzy zestawy listingów służących do obsługi GPIO: skrypty powłoki systemowej bash z bezpośrednią obsługą z poziomu systemu plików, skrypty Python z wykorzystaniem biblioteki RPi.GPIO oraz programy w języku C++ z wykorzystaniem biblioteki WiringPi.

Listingi 11.-15. przedstawiają odpowiednio: inicjalizację wyjścia, zapis (1 / 0), deinicjalizację wyjścia cyfrowego oraz przykład aplikacji.

*Listing 11. Obsługa GPIO za pomocą powłoki systemowej - inicjalizacja wyjścia.*

```
01. #!/bin/bash
02. # GPIO output init example
03. # !! run with 'sudo'
04.
05. # Exports pina to userspace
06. echo "20" > /sys/class/gpio/export
07.
08. # Sets pin GPIO20 as an output
09. echo "out" > /sys/class/gpio/gpio20/direction
10.
11. # Sets pin GPIO20 to low
12. echo "0" > /sys/class/gpio/gpio20/value
```

*Listing 12. Obsługa GPIO za pomocą powłoki systemowej - zapis 1.*

```
01. #!/bin/bash
02. # GPIO output set example
03.
04. # Sets pin GPIO20 to high
05. echo "1" > /sys/class/gpio/gpio20/value
```

*Listing 13. Obsługa GPIO za pomocą powłoki systemowej - zapis 0.*

```
01. #!/bin/bash
02. # GPIO output reset example
03.
04. # Sets pin GPIO20 to low
05. echo "0" > /sys/class/gpio/gpio20/value
```

*Listing 14. Obsługa GPIO za pomocą powłoki systemowej - deinicjalizacja.*

```
01. #!/bin/bash
02. # GPIO output deinit example
03.
04. # Unexports pin from userspace
05. echo "20" > /sys/class/gpio/unexport
```

*Listing 15. Obsługa GPIO za pomocą powłoki systemowej - przykład prostej aplikacji: zmiana stanu diody ze statym krokiem.*

```
01. #!/bin/bash
02. ***
03. ****
04. ** @file      /gpio_examples/bash/gpio_output_example.sh
05. ** @author    Adrian Wojcik
06. ** @version   V1.0
07. ** @date      15-Mar-2020
08. ** @brief     Raspberry Pi digital output control: bash script
09. ****
10.
11. bash ./gpio_output_init.sh
12. echo "Press any key to exit."
```

```
13. GPIO_STATE=0
14. while [ true ] ;
15. do
16. read -t .5 -n 1
17. if [ $? = 0 ] ;
18. then
19.     bash ./gpio_output_deinit.sh
20.     exit ;
21. else
22.     if (( GPIO_STATE == 0 )); then
23.         GPIO_STATE=1
24.         bash ./gpio_output_set.sh
25.     else
26.         GPIO_STATE=0
27.         bash ./gpio_output_reset.sh
28.     fi
29. fi
30. done
```

Listingi 16.-18. przedstawiają odpowiednio: inicjalizację wejścia, odczyt oraz przykład aplikacji. Deinicjalizacja jest identyczna niezależnie od kierunku.

*Listing 16. Obsługa GPIO za pomocą powłoki systemowej - inicjalizacja wejścia.*

```
01. #!/bin/bash
02. # GPIO input init example
03. # !! run with 'sudo'
04.
05. # Exports pin to userspace
06. echo "16" > /sys/class/gpio/export
07.
08. # Sets pin GPIO16 as an input
09. echo "in" > /sys/class/gpio/gpio16/direction
```

*Listing 17. Obsługa GPIO za pomocą powłoki systemowej - odczyt.*

```
01. #!/bin/bash
02. # GPIO input read example
03.
04. # Return input value
05. cat /sys/class/gpio/gpio16/value
```

Listing 18. Obsługa GPIO za pomocą powłoki systemowej - przykład prostej aplikacji: detekcja zbocza opadającego.

```
01. #!/bin/bash
02. ***
03. ****
04. ** @file      /gpio_examples/bash/gpio_input_example.sh
05. ** @author    Adrian Wojcik
06. ** @version   V1.0
07. ** @date      15-Mar-2020
08. ** @brief     Raspberry Pi digital input control: bash script
09. ****
10.
11. bash ./gpio_input_init.sh
12. echo "Press any key to exit."
13. GPIO_STATE="1"
14. GPIO_STATE_LAST="1"
15. CNT=0
16. while [ true ] ;
17. do
18.
```

```
19. read -t .1 -n 1
20. if [ $? = 0 ] ;
21. then
22.     bash ./gpio_input_deinit.sh
23.     exit ;
24. else
25.     GPIO_STATE=$(./gpio_input_read.sh)
26.     if [[ $GPIO_STATE -eq 0 ]] && [[ $GPIO_STATE_LAST -eq 1 ]];
27.     then
28.         CNT=$((CNT+1))
29.         echo "Push-button counter: $CNT"
30.     fi
31.     GPIO_STATE_LAST=$((GPIO_STATE))
32. fi
33. done
```

Na listingach 19.-21. przedstawiono przykłady prostej aplikacji dla wyjścia cyfrowego, wejścia cyfrowego oraz sterowanie stanem wyjścia za pomocą wejścia, z wykorzystaniem biblioteki RPi.GPIO.

*Listing 19. Obsługa GPIO z użyciem języka Python - obsługa wyjścia.*

```
01. #!/usr/bin/python3
02. ***
03. ****
04.  * @file      /gpio_examples/python/gpio_output_example.py
05.  * @author    Adrian Wojcik
06.  * @version   V1.0
07.  * @date      15-Mar-2020
08.  * @brief     Raspberry Pi digital output control: Python 3 with RPi.GPIO lib
09.  ****
10.
11. import time
12. import sys
13. import select
14. try:
15.     import RPi.GPIO as GPIO
16. except RuntimeError:
17.     print("Error importing RPi.GPIO! Use 'sudo' to run your script")
18.
19. timeout = 0.1
20. ledState = False
21. i = ''
22.
23. # Pin Definitions:
24. ledPin = 38      #< LED: Physical pin 38, BCM GPIO28
25.
26. # Pin Setup:
27. GPIO.setmode(GPIO.BOARD)
28. GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output
29.
30. print("Press ENTER to exit.")
31.
32. GPIO.output(ledPin, GPIO.LOW)
33. while not i:
34.     # Waiting for I/O completion
35.     i, o, e = select.select( [sys.stdin], [], [], timeout )
36.
37.     if (i):
38.         sys.stdin.readline();
39.         GPIO.cleanup() # cleanup all GPIO
40.         exit()
41.
42.     ledState = not ledState
```

```
43.  
44.     if (ledState):  
45.         GPIO.output(ledPin, GPIO.HIGH)  
46.     else:  
47.         GPIO.output(ledPin, GPIO.LOW)
```

*Listing 20. Obsługa GPIO z użyciem języka Python - obsługa wejścia.*

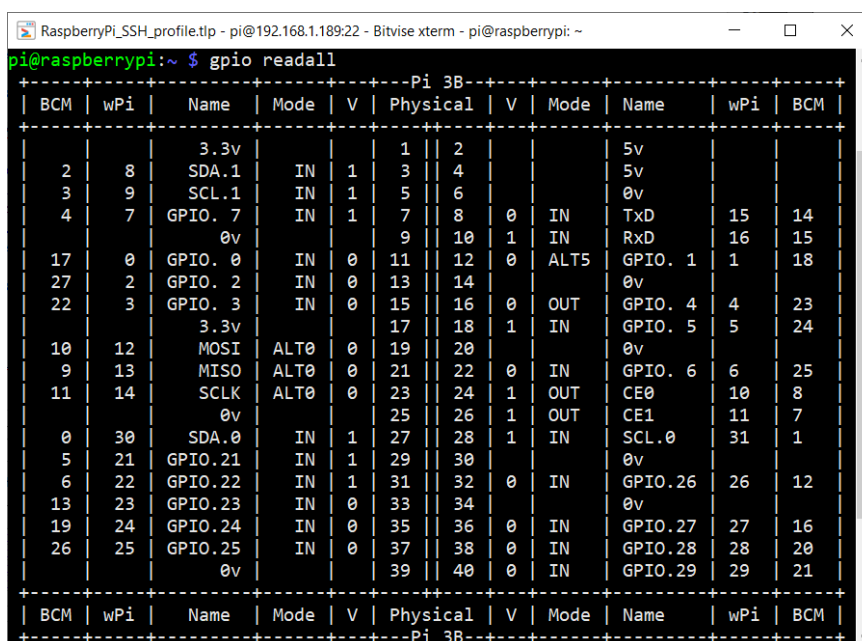
```
01. #!/usr/bin/python3  
02. ***  
03. #*****  
04. #* @file      /gpio_examples/python/gpio_input_example.py  
05. #* @author    Adrian Wojcik  
06. #* @version   V1.0  
07. #* @date      15-Mar-2020  
08. #* @brief     Raspberry Pi digital input control: Python 3 with RPi.GPIO lib  
09. #*****  
10.  
11. import time  
12. import sys  
13. import select  
14. try:  
15.     import RPi.GPIO as GPIO  
16. except RuntimeError:  
17.     print("Error importing RPi.GPIO! Use 'sudo' to run your script")  
18.  
19.     timeout = 0.1  
20.     buttonState = True  
21.     buttonStateLast = True  
22.     cnt = 0  
23.     i = ''  
24.  
25.     # Pin Definitions:  
26.     buttonPin = 36 #< Push-button: Physical pin 36, BCM GPIO16  
27.  
28.     # Pin Setup:  
29.     GPIO.setmode(GPIO.BOARD)  
30.     GPIO.setup(buttonPin, GPIO.IN) # Button pin set as input  
31.  
32.     print("Press ENTER to exit.")  
33.  
34.     while not i:  
35.         # Waiting for I/O completion  
36.         i, o, e = select.select( [sys.stdin], [], [], timeout )  
37.  
38.         if (i):  
39.             sys.stdin.readline();  
40.             GPIO.cleanup() # cleanup all GPIO  
41.             exit()  
42.  
43.         buttonState = (GPIO.input(buttonPin) == GPIO.HIGH)  
44.  
45.         if (buttonState is False) and (buttonStateLast is True):  
46.             cnt = cnt + 1  
47.             print("Push-button counter:", cnt)  
48.  
49.         buttonStateLast = buttonState
```

*Listing 21. Obsługa GPIO z użyciem języka Python - zmiana stanu wyjścia po detekcji zbocza opadającego na wejściu.*

```
01. #!/usr/bin/python3
02. ***
03. #*****
04. #* @file      /gpio_examples/python/gpio_example.py
05. #* @author    Adrian Wojcik
06. #* @version   V1.0
07. #* @date      15-Mar-2020
08. #* @brief     Raspberry Pi GPIO control: Python 3 with RPi.GPIO lib
09. #*****
10.
11. import time
12. import sys
13. import select
14. try:
15.     import RPi.GPIO as GPIO
16. except RuntimeError:
17.     print("Error importing RPi.GPIO! Use 'sudo' to run your script")
18.
19. timeout = 0.1
20. buttonState = True
21. buttonStateLast = True
22. ledState = False
23. ledStateName = ["OFF", "ON"]
24. i = ''
25.
26. # Pin Definitions:
27. ledPin = 38      #< LED: Physical pin 38, BCM GPIO28
28. buttonPin = 36   #< Push-button: Physical pin 36, BCM GPIO16
29.
30. # Pin Setup:
31. GPIO.setmode(GPIO.BOARD)
32. GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output
33. GPIO.setup(buttonPin, GPIO.IN) # Button pin set as input
34.
35. GPIO.output(ledPin, ledState)
36.
37. print("Press ENTER to exit.")
38.
39. while not i:
40.     # Waiting for I/O completion
41.     i, o, e = select.select( [sys.stdin], [], [], timeout )
42.
43.     if (i):
44.         sys.stdin.readline();
45.         GPIO.cleanup() # cleanup all GPIO
46.         exit()
47.
48.     buttonState = (GPIO.input(buttonPin) == GPIO.HIGH)
49.
50.     if (buttonState is False) and (buttonStateLast is True):
51.         ledState = not ledState
52.         GPIO.output(ledPin, ledState)
53.         print("LED state: ", ledStateName[ledState])
54.
55.     buttonStateLast = buttonState
```

W przypadku programów napisanych w języku C++ wykorzystano zewnętrzną bibliotekę WiringPi. Warto zaznaczyć, że jest ona również (*nieoficjalnie*) dostępna dla języka Python. Wraz z instalacją tej biblioteki dostępny jest zestaw dodatkowych narzędzi, w tym aplikacja **gpio**. Umożliwia ona obsługę GPIO bezpośrednio z poziomu terminala w sposób istotnie prostszy niż bezpośredni odczyt/zapis do

plików. Korzystając z narzędzi WiringPi warto zawsze upewnić się, z jakiego systemu numeracji GPIO korzystamy - ułatwia to polecenie `gpio readall` którego rezultat dla RPi model 3B przedstawiono na rys 3.



BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
2	8	3.3v			1	2		5v		
3	9	SDA.1	IN	1	3	4		5v		
4	7	SCL.1	IN	1	5	6		0v		
17	0	GPIO. 7	IN	1	7	8	0	IN	TxD	15
27	2	0v			9	10	1	IN	RxD	16
22	3	GPIO. 0	IN	0	11	12	0	ALT5	GPIO. 1	1
10	12	GPIO. 2	IN	0	13	14		0v		18
9	13	GPIO. 3	IN	0	15	16	0	OUT	GPIO. 4	4
11	14	3.3v			17	18	1	IN	GPIO. 5	5
10	12	MOSI	ALT0	0	19	20		0v		24
9	13	MISO	ALT0	0	21	22	0	IN	GPIO. 6	6
11	14	SCL	ALT0	0	23	24	1	OUT	CE0	10
0	30	0v			25	26	1	OUT	CE1	11
5	21	SDA.0	IN	1	27	28	1	IN	SCL.0	31
6	22	GPIO.21	IN	1	29	30		0v		1
13	23	GPIO.22	IN	1	31	32	0	IN	GPIO.26	26
19	24	GPIO.23	IN	0	33	34		0v		12
26	25	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27
		GPIO.25	IN	0	37	38	0	IN	GPIO.28	28
		0v			39	40	0	IN	GPIO.29	29

Rys. 3. Numeracja złącza wg WiringPi - model 3B.

Na listingach 22.-24. przedstawiono przykłady prostej aplikacji dla wyjścia cyfrowego, wejścia cyfrowego oraz sterowanie stanem wyjścia za pomocą wejścia, z wykorzystaniem biblioteki WiringPi.

Listing 22. Obsługa GPIO z użyciem języka C++ - obsługa wyjścia.

```
01. /**
02.  ****
03.  * @file      /gpio_examples/cpp/gpio_output_example.cpp
04.  * @author    Adrian Wojcik
05.  * @version   V1.0
06.  * @date      15-Mar-2020
07.  * @brief     Raspberry Pi digital output control: C++ with wiringPi lib
08.  ****
09.  */
10.
11. #include <iostream>
12. #include <future>
13. #include <thread>
14. #include <chrono>
15. #include <wiringPi.h>
16.
17. int main()
18. {
19.     const int led = 28; //< Red LED: Physical pin 38, BCM GPIO20, and WiringPi pin
20.         28.
21.
22.     std::chrono::milliseconds timeout(100);
23.     std::future<int> async_getchar = std::async(std::getchar);
24.
25.     wiringPiSetup();
26.
27.     pinMode(led, OUTPUT);
28.
29.     std::cout << "Press ENTER to exit." << std::endl;
```

```
29.
30.     while(1)
31.     {
32.         digitalWrite(led, HIGH);
33.
34.         std::this_thread::sleep_for(timeout);
35.
36.         digitalWrite(led, LOW);
37.
38.         if(async_getchar.wait_for(timeout) == std::future_status::ready)
39.         {
40.             async_getchar.get();
41.             break;
42.         }
43.     }
44.
45.     return 0;
46. }
```

*Listing 23. Obsługa GPIO z użyciem języka C++ - obsługa wejścia.*

```
01. /**
02.  ****
03.  * @file      /gpio_examples/cpp/gpio_input_example.cpp
04.  * @author    Adrian Wojcik
05.  * @version   V1.0
06.  * @date      15-Mar-2020
07.  * @brief     Raspberry Pi digital input control: C++ with wiringPi lib
08.  ****
09.  */
10.
11. #include <iostream>
12. #include <future>
13. #include <thread>
14. #include <chrono>
15. #include <wiringPi.h>
16.
17. int main()
18. {
19.     const int button = 27; //< Push-button: Physical pin 36, BCM GPIO16, and
20.         WiringPi pin 27.
21.
22.     bool gpio_state = true, gpio_state_last = true;
23.     unsigned int cnt = 0;
24.
25.     std::chrono::milliseconds timeout(100);
26.     std::future<int> async_getchar = std::async(std::getchar);
27.
28.     wiringPiSetup();
29.     pinMode(button, INPUT);
30.
31.     std::cout << "Press ENTER to exit." << std::endl;
32.
33.     while(1)
34.     {
35.         gpio_state = (digitalRead(button) == HIGH);
36.
37.         if( !gpio_state && gpio_state_last)
38.         {
39.             cnt++;
40.             std::cout << "Push-button counter: " << cnt << std::endl;
41.         }
42.     }
```



```
43.     gpio_state_last = gpio_state;
44.
45.     if(async_getchar.wait_for(timeout) == std::future_status::ready)
46.     {
47.         async_getchar.get();
48.         break;
49.     }
50. }
51.
52. return 0;
53. }
```

*Listing 24. Obsługa GPIO z użyciem języka C++ - zmiana stanu wyjścia po detekcji zbocza opadającego na wejściu.*

```
01.  /**
02.  ****
03.  * @file      /gpio_examples/cpp/gpio_example.cpp
04.  * @author    Adrian Wojcik
05.  * @version   V1.0
06.  * @date      15-Mar-2020
07.  * @brief     Raspberry Pi GPIO control: C++ with wiringPi lib
08.  ****
09.  */
10.
11. #include <iostream>
12. #include <future>
13. #include <thread>
14. #include <chrono>
15. #include <wiringPi.h>
16.
17. int main()
18. {
19.     const int led = 28;    //< Red LED: Physical pin 38, BCM GPIO20, and WiringPi
20.                           //< pin 28.
21.     const int button = 27; //< Push-button: Physical pin 36, BCM GPIO16, and
22.                           //< WiringPi pin 27.
23.
24.     bool btn_state = true, btn_state_last = true;
25.     bool led_state = false;
26.
27.     std::chrono::milliseconds timeout(100);
28.     std::future<int> async_getchar = std::async(std::getchar);
29.
30.     wiringPiSetup();
31.
32.     pinMode(button, INPUT);
33.     pinMode(led, OUTPUT);
34.
35.     digitalWrite(led, LOW);
36.
37.     std::cout << "Press ENTER to exit." << std::endl;
38.
39.     while(1)
40.     {
41.         btn_state = (digitalRead(button) == HIGH);
42.
43.         if( !btn_state && btn_state_last){
44.             led_state = !led_state;
45.             digitalWrite(led, led_state);
46.             std::cout << "LED state: ";
47.             if(led_state)
48.                 std::cout << "ON" << std::endl;
```

```
47.         else
48.             std::cout << "OFF" << std::endl;
49.     }
50.
51.     btn_state_last = btn_state;
52.
53.     if(async_getchar.wait_for(timeout) == std::future_status::ready)
54.     {
55.         async_getchar.get();
56.         break;
57.     }
58. }
59.
60. return 0;
61. }
```

## 2. PWM

RPi wyposażone jest w 2 kanały PWM, z których każdy może być zmapowany na jeden z 2 różnych pinów. Na listingu 25. przedstawiono skrypt powłoki systemowej wykorzystujący aplikację **gpio** do konfiguracji PWM1 - wyjście pracuje ze stałą częstotliwością 500 Hz oraz wypełnieniem zadany przez użytkownika za pomocą argumentu wejściowego skryptu (pominięto walidacje). Na listingu 26. przykładową aplikację - płynna zmiana wypełnienia.

*Listing 25. Obsługa PWM za pomocą powłoki systemowej i aplikacji **gpio** - ustawienie wypełnienia.*

```
01. #!/bin/bash
02. ***
03. ****
04. ** @file      /pwm_examples/bash/pwm_setDuty_example.sh
05. ** @author    Adrian Wojcik
06. ** @version   V1.0
07. ** @date      15-Mar-2020
08. ** @brief     Raspberry Pi PWM control: bash with gpio app
09. ****
10.
11. gpio -g mode 18 pwm
12. gpio pwm-ms
13. # pwmFrequency in Hz = 19 200 000 Hz / pwmClockDiv / pwmCounter.
14. gpio pwmc 192      # pwmClockDiv
15. gpio pwmr 200      # pwmCounter
16.
17. if [ $# -eq 1 ] ; then
18.     pwm_duty=$(( $1 * 2 ))
19.     gpio -g pwm 18 $pwm_duty
20. else
21.     gpio -g pwm 18 100
22. fi
23.
24. echo "Press any key to continue"
25.
26. while [ true ] ;
27. do
28.     read -t 1 -n 1
29.     if [ $? = 0 ] ; then
30.         break ;
31.     fi
32. done
33.
34. gpio -g mode 18 out
35. gpio unexport 18
```

Listing 26. Obsługa PWM za pomocą powłoki systemowej i aplikacji **gpio** - płynna zmiana wypełnienia.

```
01. #!/bin/bash
02. ***
03. #*****
04. #* @file      /pwm_examples/bash/pwm_example.sh
05. #* @author    Adrian Wojcik
06. #* @version   V1.0
07. #* @date      15-Mar-2020
08. #* @brief     Raspberry Pi PWM control: bash with gpio app
09. #*****
10.
11. gpio -g mode 18 pwm
12. gpio pwm-ms
13. # pwmFrequency in Hz = 19 200 000 Hz / pwmClockDiv / pwmCounter.
14. gpio pwmc 192      # pwmClockDiv
15. gpio pwmr 200      # pwmCounter
16.
17. gpio -g pwm 18 0
18.
19. echo "Press any key to continue"
20.
21. while [ true ] ;
22. do
23.     read -t 1 -n 1
24.     if [ $? = 0 ] ; then
25.         break ;
26.     else
27.
28.         for i in $(seq 0 1 200)
29.         do
30.             gpio -g pwm 18 ${i}
31.         done
32.
33.         for i in $(seq 200 -1 0)
34.         do
35.             gpio -g pwm 18 ${i}
36.         done
37.
38.     fi
39. done
40.
41. gpio -g mode 18 out
42. gpio unexport 18
```

Na listingach 27.-28. przedstawiono analogiczne aplikacje napisane w języku Python z wykorzystaniem biblioteki RPi.GPIO.

Listing 27. Obsługa PWM za pomocą języka Python - ustawienie wypełnienia.

```
01. #!/usr/bin/python3
02. ***
03. #*****
04. #* @file      /pwm_examples/python/pwm_example.py
05. #* @author    Adrian Wojcik
06. #* @version   V1.0
07. #* @date      15-Mar-2020
08. #* @brief     Raspberry Pi PWM control: Python 3 with RPi.GPIO lib
09. #*****
10.
11. import time
12. import sys
13. import select
14. import numpy
```

```
15.
16. try:
17.     import RPi.GPIO as GPIO
18. except RuntimeError:
19.     print("Error importing RPi.GPIO! Use 'sudo' to run your script")
20.
21. timeout = 1
22. i = ''
23. duty = 0 # [%]
24. freq = 500 # [Hz]
25.
26. # Pin Definitions:
27. pwmPin = 12 #< LED: Physical pin 12, BCM GPIO18
28.
29. GPIO.setmode(GPIO.BOARD)
30. GPIO.setup(pwmPin, GPIO.OUT)
31.
32. p = GPIO.PWM(pwmPin, freq)
33. p.start(duty)
34.
35. print("Press ENTER to exit.")
36.
37. while not i:
38.     # Waiting for I/O completion
39.     i, o, e = select.select( [sys.stdin], [], [], timeout )
40.
41.     if (i):
42.         sys.stdin.readline();
43.         p.stop()
44.         GPIO.cleanup() # cleanup all GPIO
45.         exit()
46.
47.     for d in numpy.arange(0, 100, 1):
48.         #print(d)
49.         p.ChangeDutyCycle(d)
50.         time.sleep(0.02)
51.     for d in numpy.arange(100, -1, -1):
52.         #print(d)
53.         p.ChangeDutyCycle(d)
54.         time.sleep(0.02)
```

Listing 28. Obsługa PWM za pomocą języka Python - - płynna zmiana wypełnienia.

```
01. #!/usr/bin/python3
02. ***
03. #*****
04. #* @file      /pwm_examples/python/pwm_example.py
05. #* @author    Adrian Wojcik
06. #* @version   V1.0
07. #* @date      15-Mar-2020
08. #* @brief     Raspberry Pi PWM control: Python 3 with RPi.GPIO lib
09. #*****
10.
11. import time
12. import sys
13. import select
14. import numpy
15.
16. try:
17.     import RPi.GPIO as GPIO
18. except RuntimeError:
19.     print("Error importing RPi.GPIO! Use 'sudo' to run your script")
20.
21. timeout = 1
```

```
22. i = ''
23. duty = 0 # [%]
24. freq = 500 # [Hz]
25.
26. # Pin Definitons:
27. pwmPin = 12 #< LED: Physical pin 12, BCM GPIO18
28.
29. GPIO.setmode(GPIO.BOARD)
30. GPIO.setup(pwmPin, GPIO.OUT)
31.
32. p = GPIO.PWM(pwmPin, freq)
33. p.start(duty)
34.
35. print("Press ENTER to exit.")
36.
37. while not i:
38.     # Waiting for I/O completion
39.     i, o, e = select.select([sys.stdin], [], [], timeout )
40.
41.     if (i):
42.         sys.stdin.readline();
43.         p.stop()
44.         GPIO.cleanup() # cleanup all GPIO
45.         exit()
46.
47.     for d in numpy.arange(0, 100, 0.5):
48.         p.ChangeDutyCycle(d)
49.         time.sleep(0.01)
50.     for d in numpy.arange(100, 0, -0.5):
51.         p.ChangeDutyCycle(d)
52.         time.sleep(0.01)
```

Na listingach 29.-29. przedstawiono analogiczne aplikacje napisane w języku C++ z wykorzystaniem biblioteki WiringPi.

*Listing 29. Obsługa PWM za pomocą języka C++ i biblioteki WiringPi - ustawienie wypełnienia.*

```
01. /**
02.  ****
03.  * @file    /pwm_examples/cpp/pwm_setDuty_example.cpp
04.  * @author  Adrian Wojcik
05.  * @version V1.0
06.  * @date    15-Mar-2020
07.  * @brief   Raspberry Pi PWM control: C++ with wiringPi lib
08.  ****
09.  */
10.
11. #include <iostream>
12. #include <future>
13. #include <thread>
14. #include <chrono>
15. #include <wiringPi.h>
16.
17. int main(int argc, char *argv[])
18. {
19.     const int pwmPin = 1; //< Red LED: Physical pin 12, BCM GPIO18, and WiringPi
        pin 1.
20.     float duty = 0;
21.     const int range = 200;
22.     const int clock = 192;
23.
24.     sscanf(argv[1], "%f", &duty); // C-style
25.     duty *= (float)range;
```

```
26.    duty /= 100.0;
27.
28.    std::chrono::milliseconds timeout(100);
29.    std::future<int> async_getchar = std::async(std::getchar);
30.
31.    wiringPiSetup();
32.
33.    pinMode(pwmPin, PWM_OUTPUT);
34.
35.    pwmSetMode(PWM_MODE_MS);
36.    pwmSetRange(range);
37.    pwmSetClock(clock);
38.
39.    pwmWrite(pwmPin, (int)duty);
40.
41.    std::cout << "Press ENTER to exit." << std::endl;
42.
43.    while(1)
44.    {
45.        if(async_getchar.wait_for(timeout) == std::future_status::ready)
46.        {
47.            async_getchar.get();
48.            break;
49.        }
50.    }
51.
52.    return 0;
53. }
```

Listing 30. Obsługa PWM za pomocą języka C++ i biblioteki WiringP - płynna zmiana wypełnienia.

```
01.  /**
02.   * *****
03.   * @file    /pwm_examples/cpp/pwm_example.cpp
04.   * @author  Adrian Wojcik
05.   * @version V1.0
06.   * @date    15-Mar-2020
07.   * @brief   Raspberry Pi PWM control: C++ with wiringPi lib
08.   * *****
09.   */
10.
11.  #include <iostream>
12.  #include <future>
13.  #include <thread>
14.  #include <chrono>
15.  #include <wiringPi.h>
16.
17.  int main(int argc, char *argv[])
18.  {
19.      const int pwmPin = 1; //< Red LED: Physical pin 12, BCM GPIO18, and WiringPi
20.          pin 1.
21.      const int range = 200;
22.      const int clock = 192;
23.      int step = 1;
24.
25.      std::chrono::milliseconds timeout(1000);
26.      std::chrono::milliseconds delay(10);
27.      std::future<int> async_getchar = std::async(std::getchar);
28.
29.      wiringPiSetup();
30.
31.      pinMode(pwmPin, PWM_OUTPUT);
32.
33.      pwmSetMode(PWM_MODE_MS);
```

```
33.   pwmSetRange(range);
34.   pwmSetClock(clock);
35.
36.   pwmWrite(pwmPin, 0) ;
37.
38.   std::cout << "Press ENTER to exit." << std::endl;
39.
40.   while(1)
41.   {
42.       if(async_getchar.wait_for(timeout) == std::future_status::ready)
43.       {
44.           async_getchar.get();
45.           break;
46.       }
47.
48.       for(int d = 0 ; d <= range ; d+=step)
49.       {
50.           pwmWrite(pwmPin, d) ;
51.           std::this_thread::sleep_for(delay);
52.       }
53.
54.       for(int d = range ; d >= 0 ; d-=step)
55.       {
56.           pwmWrite(pwmPin, d) ;
57.           std::this_thread::sleep_for(delay); ;
58.       }
59.   }
60.
61.   return 0;
62. }
```

## F) KONFIGURACJA SERWERA LIGHTTPD

Wykorzystanie RPi jako serwera Internetu Przedmiotów (*Internetu rzeczy*) wymaga konfiguracji serwera WWW oraz interpretera PHP. W ramach kursu wykorzystana zostanie aplikacja Lighttpd - serwer sieciowy typu open source, dedykowany dla platform o ograniczonych zasobach sprzętowych, zachowując przy tym zgodność ze standardami, bezpieczeństwo i elastyczność [16]. Zestawy laboratoryjne (oraz wersje wirtualne) **nie wymagają dodatkowych instalacji**. Instalacji aplikacji Lighttpd w systemie Raspberry Pi OS możemy dokonać komendą `apt-get install lighttpd`. W celu umożliwienia wykorzystania skryptów PHP na serwerze należy zainstalować niezbędne oprogramowanie komendą `apt-get -y install php7.x-fpm php7.x`, gdzie **x** to numer wersji PHP7 (aktualnie 0-4).

Podstawowa konfiguracja serwera dostępna jest w pliku `/etc/lighttpd/lighttpd.conf` (listing 31.). Modyfikując (nadpisując) pliki konfiguracyjne warto wcześniej wykonać ich kopię bezpieczeństwa:

```
cp nazwa_pliku.conf nazwa_pliku.conf.bak ,  
by później w razie potrzeby przywrócić ustawienia domyślne:  
cp nazwa_pliku.conf.bak nazwa_pliku.conf .
```

Pracując z serwerem warto zmodyfikować ścieżkę roboczą (*server.document-root*) na taką, do której będziemy mieli pełny dostęp z poziomu klienta SSH. Istotną informacją jest również nazwa użytkownika (*server.username*) oraz nazwa grupy (*server.groupname*). Są to kluczowe informacje w kontekście nadania serwerowi odpowiednich uprawnień.

Listing 31. Domyślna zawartość pliku konfiguracyjnego `/etc/lighttpd/lighttpd.conf`

```
01. /var/www/htmlserver.modules = (  
02.     "mod_access",  
03.     "mod_alias",  
04.     "mod_compress",  
05.     "mod_redirect",  
06. )  
07.  
08. server.document-root      = "/var/www/html"  
09. server.upload-dirs        = ( "/var/cache/lighttpd/uploads" )  
10. server.errorlog           = "/var/log/lighttpd/error.log"  
11. server.pid-file          = "/var/run/lighttpd.pid"  
12. server.username          = "www-data"  
13. server.groupname         = "www-data"  
14. server.port               = 80  
15.  
16.  
17. index-file.names          = ( "index.php", "index.html", "index.lighttpd.html"  
18. )  
19. url.access-deny           = ( "~", ".inc" )  
20. static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )  
21.  
22. compress.cache-dir        = "/var/cache/lighttpd/compress/"  
23. compress.filetype         = ( "application/javascript", "text/css", "text/html", "text/  
24. plain" )  
25.  
26. # default listening port for IPv6 falls back to the IPv4 port  
27. include_shell "/usr/share/lighttpd/use-ipv6.pl" + server.port  
28. include_shell "/usr/share/lighttpd/create-mime.assign.pl"  
29. include_shell "/usr/share/lighttpd/include-conf-enabled.pl"
```

Jeżeli chcemy, by serwer miał możliwość wykonywania programów i skryptów dokonujących interakcji z systemem plików oraz sprzętem (GPIO, PWM, UART, I2C, SPI) należy nadać użytkownikowi (tu: *www-data*) odpowiednie uprawnienia. Edycje uprawnień użytkownika zaczynamy od wywołania komendy `sudo visudo`. W celu nadania użytkownikowi możliwości wykonania danego programu należy dodać liniijkę: `www-data ALL = (root)NOPASSWD: /ściezka/do/mojego/programu_lub_skryptu`. Kolejne nazwy plików rozdzielamy przecinkiem.

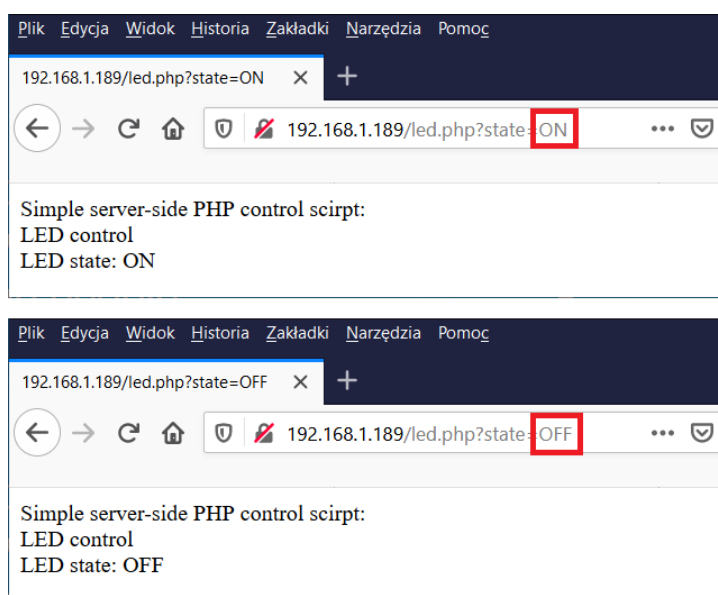


**UWAGA!** Na etapie rozwoju i testowania aplikacji serwerowej wygodną opcją jest **tymczasowe** nadanie serwerowi uprawnień administratora: `www-data ALL=(ALL)NOPASSWD:ALL`. Jest to jednak absolutnie niedopuszczalna sytuacja w finalnym produkcie, ze względów bezpieczeństwa. Nie należy korzystać z tej konfiguracji przy jednoczesnym dostępie do Internetu!

Na listingu 32. przedstawiono przykładowy skrypt PHP wykorzystujący skrypty powłoki systemowej do obsługi GPIO. Na rys. 4 przedstawiono stan wynik egzekucji skryptu w przeglądarce Firefox dla różnych argumentów: *ON* oraz *OFF*. Jeżeli zapewnimy wcześniej odpowiednią konfigurację GPIO jako wyjścia cyfrowego i podłączymy diodę świecącą wg schematu (rys. 2) uzyskujemy możliwość sterowania stanem diody z poziomu każdego urządzenia wyposażonego w przeglądarkę internetową, połączonego z tą samą siecią **lokalną**.

*Listing 32. Przykład skryptu PHP sterującego diodą świecącą (via GPIO) za pośrednictwem skryptu powłoki systemowej.*

```
01. <?php
02. echo "Simple server-side PHP control script:<br>";
03. echo "LED control<br>";
04.
05. $led_state='';
06. if(isset($_GET['state']))
07.     $led_state=$_GET['state'];
08.
09. if(strcmp($led_state, "ON") == 0){
10.     exec('sudo ../gpio_examples/bash/./gpio_output_set.sh');
11.     echo "LED state: ON";
12. }
13. elseif(strcmp($led_state, "OFF") == 0){
14.     exec('sudo ../gpio_examples/bash/./gpio_output_reset.sh');
15.     echo "LED state: OFF";
16. }
17. else {
18.     echo "LED state undefined";
19. }
20. ?>
```



Rys. 4. Odpowiedź serwera - skrypt sterujący stanem diody świecącej (via GPIO)

## IV. SCENARIUSZ DO ZAJĘĆ

### A) ŚRODKI DYDAKTYCZNE

- |            |   |
|------------|---|
| Sprzętowe  | <ul style="list-style-type: none"><li>• zestaw komputerowy,</li><li>• zestaw komputera jednopłytkowego Raspberry Pi,</li><li>• karta microSD,</li><li>• zasilacz microUSB,</li><li>• kabel RJ45,</li><li>• router WiFi,</li><li>• <i>zestaw elementów elektronicznych</i>,</li><li>• <i>multimetr</i>,</li><li>• <i>oscyloskop</i>.</li></ul> |
| Programowe | <ul style="list-style-type: none"><li>• klient SSH (np. Bitwise SSH Client),</li><li>• narzędzie do zapisu binarnych obrazów dysku na zewnętrzne nośniki (np. Win32DiskMaker),</li><li>• edytor tekstu (np. Notepad++, VS Code),</li><li>• program VirtualBox i wirtualna maszyna z systemem Raspberry Pi OS.</li></ul>                       |

### B) ZADANIA DO REALIZACJI

#### Urządzenie fizyczne

1. Uruchom Raspberry Pi: przygotuj kartę pamięci z system operacyjnym i połącz się z urządzeniem za pomocą klienta SSH.
  - (a) Pobierz obraz dysku ze skonfigurowanym systemem Raspberry Pi OS, przekazany przez prowadzącego.
  - (b) Rozpakuj archiwum i zapisz obraz na karcie microSD (min. 16 GB) za pomocą odpowiedniego narzędzia (np. Win32DiskMaker)
  - (c) Zapisz na karcie odpowiedni plik konfiguracyjny umożliwiający dostęp do lokalnej sieci WiFi, wg. instrukcji dostępnej na stronie:  
[www.raspberrypi.org/documentation/configuration/wireless/headless.md](http://www.raspberrypi.org/documentation/configuration/wireless/headless.md)
  - (d) Umieść przygotowaną kartę w slotcie, podłącz zasilanie RPi i odczekaj kilkadziesiąt sekund.
  - (e) Sprawdź adres IP Raspberry Pi w sieci lokalnej.
  - (f) Połącz się z RPi za pomocą klienta SSH (np. Bitwise SSH Client). Wykorzystaj domyślny port 22. Nazwa użytkownika: **pi**, hasło: **raspberrypi** (domyślne wartości).
  - (g) Po uruchomieniu terminala sprawdź konfigurację interfejsów sieciowych.
  - (h) Sprawdź czy RPi ma dostęp do Internetu poleceniem **ping**.
2. Skonfiguruj interfejs sieciowy - ustaw statyczne IP i połącz się z urządzeniem za pomocą klienta SSH.
  - (a) Skonfiguruj interfejs sieciowy **eth0** - wybierz statyczne IP (np. 192.168.0.15/24) i zmodyfikuj odpowiedni plik konfiguracyjny.
  - (b) Ustaw statyczne IP po stronie PC (np. 192.168.0.100/24). Połącz PC i RPi kablem sieciowym.
  - (c) Zresetuj **eth0** i sprawdź konfigurację interfejsów sieciowych.

- (d) Połącz się z RPi za pomocą klienta SSH (np. Bitwise SSH Client). Wykorzystaj domyślny port 22.
- (e) Sprawdź czy RPi ma dostęp do Internetu poleceniem **ping**. Powtórz test po wyłączeniu interfejsu `wlan0`.

## Urządzenie wirtualne

1. Uruchom urządzenie wirtualne - utwórz nową maszynę wirtualną i połącz się z urządzeniem za pomocą klienta SSH.
  - (a) Pobierz obraz wirtualnego dysku z systemem Raspberry Pi OS for Desktop, przekazane przez prowadzącego.
  - (b) Utwórz nową maszynę wirtualną dla systemu Linux (Debian, 32-bit). Wykorzystaj pobrany dysk wirtualny.
  - (c) Skonfiguruj kratę sieciową urządzenia wirtualnego w taki sposób aby zapewniała dostęp do hosta i Internetu (np. Mostkowana karta sieciowa).
  - (d) Bezpośrednio w terminalu urządzenia wirtualnego sprawdź konfigurację interfejsów sieciowych.
  - (e) Połącz się z RPi za pomocą klienta SSH (np. Bitwise SSH Client). Wykorzystaj adres IP interfejsu **eth0**. Wykorzystaj domyślny port 22. Nazwa użytkownika: **pi**, hasło: **raspberry** (domyślne wartości).
  - (f) Po uruchomieniu terminala sprawdź konfigurację interfejsów sieciowych.
  - (g) Sprawdź czy RPi ma dostęp do Internetu poleceniem **ping**.
2. Skonfiguruj interfejs sieciowy - ustaw statyczne IP i połącz się z urządzeniem za pomocą klienta SSH.
  - (a) Dodaj drugą kartę sieciową do urządzenia wirtualnego. Karta powinna zapewniać wyłącznie dostęp do hosta.
  - (b) Skonfiguruj interfejs sieciowy **eth1** - wybierz statyczne IP (np. 192.168.56.15/24) i zmodyfikuj odpowiedni plik konfiguracyjny.
  - (c) Zresetuj **eth1** i sprawdź konfigurację interfejsów sieciowych.
  - (d) Połącz się z RPi za pomocą klienta SSH (np. Bitwise SSH Client). Wykorzystaj domyślny port 22.
  - (e) Sprawdź czy RPi ma dostęp do Internetu poleceniem **ping**. Powtórz test po wyłączeniu interfejsu **eth1**.
3. Napisz aplikację wiersza poleceń z wykorzystaniem powłoki systemowej bash.
  - (a) Stwórz trzy pliki tekstowe: `temperature.dat` `humidity.dat` `pressure.dat`.
  - (b) Umieść w nich odpowiednio następujące treści: `20.5C` `80%` `1023.0hPa`.
  - (c) Aplikacja obsługiwać ma następujące (opcjonalne) argumenty: `-t` `-h` `-p`. Jeżeli dana flaga jest obecna, oznacza to, że aplikacja powinna dokonać odczytu odpowiedniego pliku tekstowego i wyświetlić (w kolejnych liniach) jego treść. Jeżeli więc nie ma argumentów program nie realizuje żadnej akcji.
  - (d) Rozbuduj aplikację o możliwość wyboru jednostki temperatury - po opcji `-t` powinna zostać podana jednostka: `c` (stopnie Celcjusza) lub `F` (stopnie Fahrenheita). Program na podstawie tego wyboru powinien wyświetlić oryginalną lub przeskalowaną wartość zapisaną w pliku.

- 
4. Napisz aplikację wiersza poleceń z wykorzystaniem języka C++ i kompilatora g++.
    - (a) Napisz aplikację spełniającą identyczną specyfikację co skrypt z poprzedniego zadania.
    - (b) Stwórz skrypt do kompilacji programu. Nadaj wynikowemu programowi odpowiednią nazwę.
    - (c) Wykorzystaj pliki z poprzedniego zadania.
  5. Napisz aplikację wiersza poleceń z wykorzystaniem języka Python.
    - (a) Napisz aplikację spełniającą identyczną specyfikację co skrypt i program z poprzednich zadania.
    - (b) Wykorzystaj pliki z poprzedniego zadania.
  6. (\*) Napisz prostą aplikację do obsługi GPIO.
    - (a) Zbuduj układ przedstawiony na rys. 2.
    - (b) Przetestuj przekazane skrypty i programy.
    - (c) Napisz własną aplikację wiersza poleceń: za pomocą przycisku ustawiaj okres zmiany stanu diody.
    - (d) Dodaj obsługę argumentów wejściowych pozwalających na konfigurację aplikacji, tj. jaki jest maksymalny i minimalny okres zmiany stanu diody oraz z jakim krokiem powinien się zmieniać po każdym naciśnięciu przycisku.
  7. (\*) Napisz prostą aplikację do obsługi PWM.
    - (a) Zbuduj układ przedstawiony na rys. 2.
    - (b) Przetestuj przekazane skrypty i programy.
    - (c) Napisz własną aplikację wiersza poleceń - rozbuduj przedstawione interfejsy o możliwość konfiguracji nie tylko wypełnienia ale również częstotliwości.
  8. Napisz skrypt CGI pozwalający na obsługę LED.
    - (a) Sprawdź konfigurację serwera Lighttpd i upewnij się, że pliki z ścieżki `document-root` dostępne są z poziomu przeglądarki internetowej hosta.
    - (b) Stwórz skrypty CGI (PHP lub Python) umożliwiające zapis do pliku wartości „1” lub „0”, przekazanej metodą GET. Możesz wykorzystać w tym celu komendy powłoki systemowej. Jeżeli pracujesz z urządzeniem fizycznym, skrypt powinien sterować wyjściem cyfrowym.
    - (c) Skrypt powinien zwracać odpowiedź zawierającą nazwę pliku (lub numer wyjścia) oraz aktualny stan po zapisie. Informacja powinna być przedstawiona w formacie JSON. Pamiętaj o adekwatnym nagłówku HTTP.

## BIBLIOGRAFIA

1. *Regulaminy porządkowe i instrukcje BHP* [online]. [B.d.] [udostępniono 2019-09-30]. Dostępne z: <http://zsep.cie.put.poznan.pl/materialy-dydaktyczne/MD/Regulaminy-porz%C4%85dkowe-instrukcje-BHP/>.
2. *Raspberry Pi Documentation* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://www.raspberrypi.org/documentation/>.
3. *Raspberry Pi Community - Projects, Blogs, and more* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://www.raspberrypi.org/community/>. Library Catalog: [www.raspberrypi.org](http://www.raspberrypi.org).
4. *ifconfig(8) - Linux manual page* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <http://man7.org/linux/man-pages/man8/ifconfig.8.html>.
5. FOUNDATION, The Raspberry Pi. *Operating system images* [Raspberry Pi] [online] [udostępniono 2021-03-16]. Dostępne z: <https://www.raspberrypi.org/software/operating-systems/>.
6. *Bash documentation — DevDocs* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://devdocs.io/bash/>. Library Catalog: [devdocs.io](http://devdocs.io).
7. *Python 3.5.9 Documentation* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://docs.python.org/3.5/>.
8. *C++11 - cppreference.com* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://en.cppreference.com/w/cpp/11>.
9. *g++(1): GNU project C/C++ compiler - Linux man page* [online] [udostępniono 2021-03-16]. Dostępne z: <https://linux.die.net/man/1/g++>.
10. *GCC online documentation - GNU Project - Free Software Foundation (FSF)* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://gcc.gnu.org/onlinedocs/>.
11. *getopts(1p) - Linux manual page* [online] [udostępniono 2021-03-24]. Dostępne z: <https://man7.org/linux/man-pages/man1/getopts.1p.html>.
12. *bc Command Manual* [online]. [B.d.] [udostępniono 2020-03-22]. Dostępne z: [https://www.gnu.org/software/bc/manual/html\\_mono/bc.html](https://www.gnu.org/software/bc/manual/html_mono/bc.html).
13. *15.6. getopt — C-style parser for command line options — Python 2.7.17 documentation* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://docs.python.org/2/library/getopt.html>.
14. *getopt(1) - Linux manual page* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <http://man7.org/linux/man-pages/man1/getopt.1.html>.
15. *Reference / Wiring Pi* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <http://wiringpi.com/reference/>. Library Catalog: [wiringpi.com](http://wiringpi.com).
16. *Docs - Lighttpd - lighty labs* [online]. [B.d.] [udostępniono 2020-03-18]. Dostępne z: <https://redmine.lighttpd.net/projects/lighttpd/wiki/Docs>.