Poznań University of Technology

Faculty of Control, Robotics and Electrical Engineering

Institute of Robotics and Machine Intelligence

Division of Control and Industrial Electronics

Server IoT
Raspberry Pi and CLIs

Mobile and embedded applications for Internet of Things

Teaching materials for laboratory

Dominik Łuczak, Ph.D.; Adrian Wójcik, M.Sc.

Dominik.Luczak@put.poznan.pl
Adrian.Wojcik@put.poznan.pl

I. Goal

Knowledge

The aim of the course is to familiarize yourself with:

- the basics of the Linux operating system (Raspberry Pi OS),
- the basics of *bash* system shell syntax,
- basics of Python3 language syntax,
- basics of network communication in Linux (Raspberry Pi OS).

Skills

The aim of the course is to acquire skills in:

- configuring network interfaces on Linux,
- creating CLI applications using bash system shell scripts,
- creating CLI applications using C and C++,
- creating a CLI application using Python,
- operating the digital inputs and outputs on a single-board computer (Raspberry Pi),
- operating the PWM modules on a single-board computer (Raspberry Pi),
- web server configuration on Linux,
- use of the CLI applications in CGI scripts.

Social competences

The aim of the course is to develop proper attitudes:

- strengthening the understanding of the role and application of network communication in IT systems and related security issues,
- strengthening understanding and awareness of the importance of non-technical aspects and effects of the engineer's activities, and the related responsibility for the decisions taken,
- proper technical communication in context of software development.
- choosing the right technology and programming tools for the given problem,

II. Laboratory report

Complete laboratory tasks as per the instructor's presentation. Work alone or in a team of two. **Keep safety rules while working!** Prepare laboratory report documenting and proving the proper execution of tasks. Editorial requirements and a report template are available on the *eKursy* platform. The report is graded in two categories: tasks execution and editorial requirements. Tasks are graded as completed (1 point) or uncompleted (0 points). Compliance with the editorial requirements is graded as a percentage. The report should be sent as a *homework* to the *eKursy* platform by Sunday, April 4, 2021 by 23:59.

III. Prepare to course

a) Know the safety rules

All information on the laboratory's safety instructions are provided in the laboratory and on Division website [1]. All inaccuracies and questions should be clarified with the instructor. It is required to be familiar with and apply to the regulations.

Attend the class prepared. Knowledge from all previous topics is mandatory.

b) Introduction to Raspberry Pi

Raspberry Pi (RPi) is a series of single-board computers (SBC), i.e. devices that contain a microprocessor(s), memory and input/output devices on single printed circuit. This device is based on an ARM family processor. Raspberry Pi has no hard disk - to load the operating system and store data, it offers a connector for microSD cards or USB flash drives. In addition, it is equipped with a connector with GPIO and popular low-level electronic interfaces - UART, I2C and SPI [2]. It is a very popular platform both in computer science didactics and rapid prototyping. An important feature of Raspberry Pi is also the extensive and active community of users [3], which results in a wide selection of available applications and programming libraries. The recommended (but by no means the only!) operating system for RPi is the Raspberry Pi OS - Debian-based Linux distributions.

c) Configuration of network interfaces

Raspberry Pi allows you to connect a keyboard and mouse (via USB) and a monitor (via HDMI). However, it is also equipped with network interfaces: Ethernet (**eth0**) and WiFi (**wlan0**). For this reason, a much more common way of using RPi is communication via the SSH (secure shell) protocol, which is used to connect to computers remotely using network interfaces. The correct configuration of network interfaces is therefore necessary for the effective use of the platform.

The basic command for configuring network interfaces is `ifconfig` [4] (not to be confused with the corresponding command in Windows - `ipconfig`!).

In order to obtain a permanent (i.e. saved after restarting the device), *static* IP address, you need to edit the configuration file `/etc/dhcpcd.conf` and add the configuration according to the pattern presented in the file (see: listing 1).

Listing 1. Configuration file /etc/dhcpcd.conf - static IP of ***eth0*** *interface.*

```
interface eth0
static ip_address=192.168.1.15/24
static routers=192.168.1.1
static domain_name_servers=8.8.8.8
```

Using WiFi requires additional configuration: adding wireless network SSID (service set identifier) and password in the configuration file `/etc/wpa_supplicant/wpa_supplicant.conf` - example in the listing 2.

Listing 2. Configuration file /etc/wpa_supplicant/wpa_supplicant.conf - Wireless network SSID and password in room M323.

```
network={
        ssid="M323_01"
        psk="labM32301"
}
```

NOTE! Please note that establishing a connection between RPi and another device requires that at least one network interface of each devices is in the same network!

D) Command line applications (CLIs)

The two basic variants of the Raspberry Pi OS are [5]:

- Desktop version - equipped with a graphic user interface (with the option to disable it),
- Lite version (*headless*) - based only on the command line (command line interpreter/interface, CLI).

As part of this course, the Raspberry Pi OS system will be operated only from the command line. Therefore, you will need to acquire skills to create and use applications launched from the command line. Three different tools will be used for this purpose:

- **bash** system shell scripts [6],
- interpreted language **Python** (version >3.5) [7],
- compiled language **C** ++ [8] with the **g**++ compiler [9].

1. BASH

Bash is the system shell of the UNIX system. It is the default shell in most GNU/Linux distributions and on macOS. Listing 3. shows simple „Hello World" script is presented. The first line of the script is the path of the appropriate interpreter. Listing 4. shows an example of a script that handles the command line input arguments using the *getopts* method [10].

Listing 3. Bash „Hello World".

```bash
#!/bin/bash
# Simple 'Hello World' example
echo "Hello World!"
```

Listing 4. Bash - getopt function example.

```bash
#!/bin/bash
#**
#*****************************************************************************
#* @file    /cli_examples/bash/bash_args.sh
#* @author  Adrian Wojcik
#* @version V1.0
#* @date    14-Mar-2020
#* @brief   Raspberry Pi CLI apps example: bash with getopt
#*****************************************************************************

nonoptionargs=()
flagstate=("RESET" "SET")
aflag=0
bflag=0
cvalue=""
echo "Bash CLI example"
# standard while/case procedure for 'getopt' function
while [ $# -gt 0 ]; do
  while getopts ":abc:" opt; do
    case $opt in
      a)
        aflag=1 ;;
      b)
        bflag=1 ;;
```

```
      c)
        cvalue=$OPTARG ;;
      \?)
        echo "option␣'-$OPTARG␣not␣recognized'"
        exit 1 ;;
      : )
        echo "option␣'-$OPTARG'␣requires␣argument"
        exit 1 ;;
    esac
  done
  shift $((OPTIND-1))

  while [ $# -gt 0 ] && ! [[ "$1" =~ ^- ]]; do
    nonoptionargs=("${nonoptionargs[@]}" "$1")
    shift
  done
done

# Printing results
echo "a␣(flag)␣=␣${flagstate[$aflag]}"
echo "b␣(flag)␣=␣${flagstate[$bflag]}"
echo "c␣(value)␣=␣$cvalue"
index=1
if [ ${#nonoptionargs[@]} -gt 0 ]; then
    for noa in "${nonoptionargs[@]}"
  do
    echo "Non-option␣argument␣#$index:␣$noa"
    index=$((index+1))
  done
fi
```

NOTE! To enable the script to be executed, you must grant it permission with the `chmod +x script_name.sh` command.

NOTE! When editing Linux scripts from another operating system (e.g. Windows), make sure that the character encoding is appropriate for UNIX systems. Source code editors (e.g. Notepad++, VS Code etc.) allow you to choose proper configurations.

Calculations in Bash are done ONLY on integers and there is no overflow control. If a given problem requires extensive operations on floating point numbers, then Bash is not an adequate tool to solve it. For simple mathematical calculations involving floating point numbers, the calculator application `bc` [11] can be used. Listing 5. shows an example of using the `bc` calculator to perform a linear function in the Bash script.

Listing 5. Bash - `bc` calculator app example.

```
#!/bin/bash
# temperature in degrees Celsius
tempC="20.5"
# temperature in degrees Fahrenheit
tempF=$(echo "tempf=1.8*$tempC+32.0;tempf" | bc);
# display result
echo "Temperature␣in␣degrees␣Celsius:␣"$tempC"*C"
echo "temperature␣in␣degrees␣Fahrenheit:␣"$tempF"*F"
```

2. Python

Python is a popular general-purpose interpreted language, i.e. a language that is usually executed (*interpreted*) as a script by the interpreter rather than compiled into a binary by the compiler. The popularity of Python is the result of, among others, the availability of a very large number of libraries for a wide range of applications. Using Python scripts requires installation of interpreter on the system, however, in many Linux distributions it is a pre-installed software. Listing 6. shows a simple „Hello World" application. The first line of the script is the path of the appropriate interpreter. Listing 7. shows an example of a script that handles command line input arguments using the *getopt* method [12].

Listing 6. Python „Hello World".

```python
#!/usr/bin/python3
# Simple 'Hello World' example
print("Hello World!")
```

Listing 7. Python - getopt function example.

```python
#!/usr/bin/python3
#**
#******************************************************************************
#* @file     /cli_examples/python/python_args.py
#* @author   Adrian Wojcik
#* @version  V1.0
#* @date     14-Mar-2020
#* @brief    Raspberry Pi CLI apps example: Python 3 with getopt
#******************************************************************************

import sys
import getopt
print("Python 3 CLI example")

flagstate = ["RESET", "SET"]
aflag = 0
bflag = 0
cvalue = ''

sysarg = sys.argv[1:]

# standard try/exept/for procedure for 'getopt' function
try:
    opts, args = getopt.getopt(sysarg, ':abc:')
except getopt.GetoptError as err:
  print(err)
  sys.exit(1)

for opt, arg in opts:
  if opt in '-a':
    aflag = 1
  elif opt in '-b':
    bflag = 1
  elif opt in '-c':
    cvalue = arg

# Printing results
print('a (flag) =',flagstate[aflag])
print('b (flag) =',flagstate[bflag])
if cvalue:
  print('c (value) =',cvalue)

for arg in opts:
  sysarg.remove(arg[0])
  if arg[1]:
```

```
    sysarg.remove(arg[1])

index = 1
for arg in sysarg:
  print('Non-option␣argument␣#', index, "␣=␣", arg, sep="")
  index = index + 1
```

3. C++

C++ is a multi-paradigm general-purpose programming language. Its basic property is a high level of independence from the hardware platform. An important feature is also compatibility with the C language - C++ libraries can be written in C. Development of applications written in C++ requires compilation of source code, and therefore installation of the compiler on the system. Listing 8. shows simple „Hello World" application. Listing 9. presents a script for compiling source code using g++ compiler [13][9]. Listing 10. shows an example of a program handling command line input arguments using the *getopt* method [14].

Listing 8. C++ „Hello World".

```
// Simple 'Hello World' example
#include <iostream>

int main(int argc, char *argv[])
{
  std::cout << "Hello␣World!" << std::endl;
  return 0;
}
```

Listing 9. Build „Hello World" using g++ compiler.

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

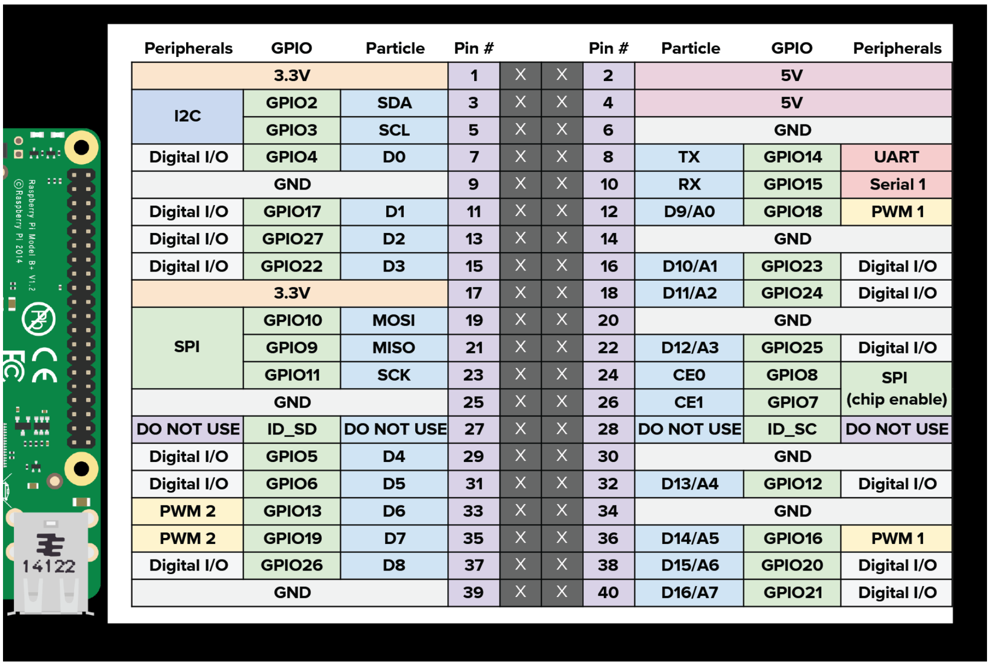
Listing 10. C++ - getopt function example.

```
/**
 ****************************************************************************
 * @file    /cli_examples/cpp/cpp_args.cpp
 * @author  Adrian Wojcik
 * @version V1.0
 * @date    14-Mar-2020
 * @brief   Raspberry Pi CLI apps example: C++ with getopt
 ****************************************************************************
 */

#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream>

int main(int argc, char *argv[])
{
  const char *flagstate[2] = { "RESET", "SET" };
  int aflag = 0;
  int bflag = 0;
  char *cvalue = nullptr;
  int index;

  int arg;

  opterr = 0;
```

```
  std::cout << "C++ CLI example" << std::endl;

  /* Standard while/switch procedure for 'getopt' function */
  while((arg = getopt (argc, argv, "abc:")) != -1)
  {
    switch(arg)
    {
      case 'a':
        aflag = 1;
        break;
      case 'b':
        bflag = 1;
        break;
      case 'c':
        cvalue = optarg;
        break;
      case '?':
        if(optopt == 'c')
          std::cerr << "option -" << static_cast<char>(optopt)
                    << " requires argument" <<std::endl;
        else if(isprint(optopt))
          std::cerr << "option -" << static_cast<char>(optopt)
                    << " not recognized" <<std::endl;
        else
          std::cerr << "option character \\x" << optopt
                    << " not recognized" <<std::endl;
        return 1;
      default:
        abort();
    }
  }

  /* Printing results */
  std::cout << "a (flag) = " << flagstate[aflag] << std::endl;
  std::cout << "b (flag) = " << flagstate[bflag] << std::endl;
  if(cvalue != nullptr)
    std::cout << "c (value) = " << cvalue << std::endl;

  for (index = optind; index < argc; index++)
    std::cout << "Non-option argument #" << (index-optind+1)
              << ": "<<  argv[index] << std::endl;

  return 0;
}
```

E) General purpose input-output connector

RPi is equipped with a 40-pin connector for general purpose digital input/output (GPIO), 2 PWM channels, and interfaces: UART, I2C and SPI (fig. **??**).



Peripherals	GPIO	Particle	Pin #			Pin #	Particle	GPIO	Peripherals
	3.3V		1	X	X	2		5V	
I2C	GPIO2	SDA	3	X	X	4		5V	
	GPIO3	SCL	5	X	X	6		GND	
Digital I/O	GPIO4	D0	7	X	X	8	TX	GPIO14	UART
	GND		9	X	X	10	RX	GPIO15	Serial 1
Digital I/O	GPIO17	D1	11	X	X	12	D9/A0	GPIO18	PWM 1
Digital I/O	GPIO27	D2	13	X	X	14		GND	
Digital I/O	GPIO22	D3	15	X	X	16	D10/A1	GPIO23	Digital I/O
	3.3V		17	X	X	18	D11/A2	GPIO24	Digital I/O
SPI	GPIO10	MOSI	19	X	X	20		GND	
	GPIO9	MISO	21	X	X	22	D12/A3	GPIO25	Digital I/O
	GPIO11	SCK	23	X	X	24	CE0	GPIO8	SPI (chip enable)
	GND		25	X	X	26	CE1	GPIO7	
DO NOT USE	ID_SD	DO NOT USE	27	X	X	28	DO NOT USE	ID_SC	DO NOT USE
Digital I/O	GPIO5	D4	29	X	X	30		GND	
Digital I/O	GPIO6	D5	31	X	X	32	D13/A4	GPIO12	Digital I/O
PWM 2	GPIO13	D6	33	X	X	34		GND	
PWM 2	GPIO19	D7	35	X	X	36	D14/A5	GPIO16	PWM 1
Digital I/O	GPIO26	D8	37	X	X	38	D15/A6	GPIO20	Digital I/O
	GND		39	X	X	40	D16/A7	GPIO21	Digital I/O

Fig. 1. Raspberry Pi low-level socket pinout.

On Linux systems, hardware access is similar to reading from/writing to files - i.e. hardware is often represented as file. It is possible to operate the peripherals by executing the commands `echo` (write) and `cat` (read) respectively. In practice, however, there are available programming libraries that provide a convenient API for basic operations related to a given periphery, e.g. WiringPi [15].

The examples presented below were implemented using external elements as in the diagram in Fig. 2.

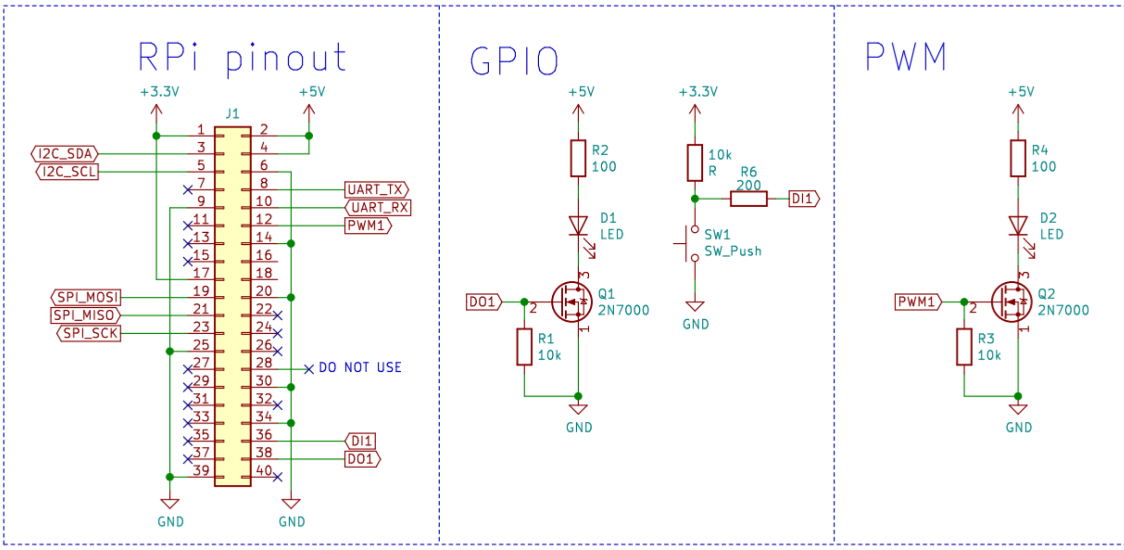


Fig. 2. Electrical diagram of external elements.

1. GPIO

Below are shown three sets of listings for GPIO operations: bash system shell scripts with direct file access, Python scripts using the RPi.GPIO library and C++ programs using the WiringPi library.

Listings 11.-15. shows respectively: output initialization, write operations (1/0), digital output deinitialization and simple application example.

Listing 11. GPIO operations using bash system shell - output initialization.

```bash
#!/bin/bash
# GPIO output init example
# !! run with 'sudo'

# Exports pina to userspace
echo "20" > /sys/class/gpio/export

# Sets pin GPIO20 as an output
echo "out" > /sys/class/gpio/gpio20/direction

# Sets pin GPIO20 to low
echo "0" > /sys/class/gpio/gpio20/value
```

Listing 12. GPIO operations using bash system shell - writing 1 (high).

```bash
#!/bin/bash
# GPIO output set example

# Sets pin GPIO20 to high
echo "1" > /sys/class/gpio/gpio20/value
```

Listing 13. GPIO operations using bash system shell - writing 0 (high).

```bash
#!/bin/bash
# GPIO output reset example

# Sets pin GPIO20 to low
echo "0" > /sys/class/gpio/gpio20/value
```

Listing 14. GPIO operations using bash system shell - output deinitialization.

```bash
#!/bin/bash
# GPIO output deinit example

# Unexports pin from userspace
echo "20" > /sys/class/gpio/unexport
```

Listing 15. OGPIO operations using bash system shell - simple application example: LED toggle.

```bash
#!/bin/bash
#**
#*****************************************************************************
#* @file    /gpio_examples/bash/gpio_output_example.sh
#* @author  Adrian Wojcik
#* @version V1.0
#* @date    15-Mar-2020
#* @brief   Raspberry Pi digital output control: bash script
#*****************************************************************************

bash ./gpio_output_init.sh
echo "Press any key to exit."
GPIO_STATE=0
while [ true ] ;
```

```
do
read -t .5 -n 1
  if [ $? = 0 ] ;
  then
    bash ./gpio_output_deinit.sh
    exit ;
  else
    if (( GPIO_STATE == 0 )); then
      GPIO_STATE=1
      bash ./gpio_output_set.sh
    else
      GPIO_STATE=0
      bash ./gpio_output_reset.sh
    fi
  fi
done
```

Listings 16.-18. shows respectively: input initialization, input readout and application example. Deinitialization is identical regardless of GPIO direction.

Listing 16. GPIO operations using bash system shell - input initialization.

```
#!/bin/bash
# GPIO input init example
# !! run with 'sudo'

# Exports pin to userspace
echo "16" > /sys/class/gpio/export

# Sets pin GPIO16 as an input
echo "in" > /sys/class/gpio/gpio16/direction
```

Listing 17. GPIO operations using bash system shell - input read.

```
#!/bin/bash
# GPIO input read example

# Return input value
cat /sys/class/gpio/gpio16/value
```

Listing 18. GPIO operations using bash system shell - simple application example: falling edge detection.

```
#!/bin/bash
#**
#******************************************************************************
#* @file     /gpio_examples/bash/gpio_input_example.sh
#* @author   Adrian Wojcik
#* @version  V1.0
#* @date     15-Mar-2020
#* @brief    Raspberry Pi digital input control: bash script
#******************************************************************************

bash ./gpio_input_init.sh
echo "Press␣any␣key␣to␣exit."
GPIO_STATE="1"
GPIO_STATE_LAST="1"
CNT=0
while [ true ] ;
do

read -t .1 -n 1
  if [ $? = 0 ] ;
  then
```

```
    bash ./gpio_input_deinit.sh
    exit ;
  else
    GPIO_STATE=$(./gpio_input_read.sh)
    if [[ $GPIO_STATE -eq 0 ]] && [[ $GPIO_STATE_LAST -eq 1 ]];
    then
      CNT=$((CNT+1))
      echo "Push-button counter: $CNT"
    fi
    GPIO_STATE_LAST=$((GPIO_STATE))
  fi
done
```

Listings 19.-21. shows examples of a simple application for a digital output, a digital input, and output control by input edge, using the RPi.GPIO library.

Listing 19. GPIO operations using Python - LED toggle.

```
#!/usr/bin/python3
#**
#******************************************************************************
#* @file     /gpio_examples/python/gpio_output_example.py
#* @author   Adrian Wojcik
#* @version V1.0
#* @date     15-Mar-2020
#* @brief    Raspberry Pi digital output control: Python 3 with RPi.GPIO lib
#******************************************************************************

import time
import sys
import select
try:
  import RPi.GPIO as GPIO
except RuntimeError:
  print("Error importing RPi.GPIO! Use 'sudo' to run your script")

timeout = 0.1
ledState = False
i = ''

# Pin Definitons:
ledPin = 38    #< LED: Physical pin 38, BCM GPIO28

# Pin Setup:
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output

print("Press ENTER to exit.")

GPIO.output(ledPin, GPIO.LOW)
while not i:
  # Waiting for I/O completion
  i, o, e = select.select( [sys.stdin], [], [], timeout )

  if (i):
    sys.stdin.readline();
    GPIO.cleanup() # cleanup all GPIO
    exit()

  ledState = not ledState

  if (ledState):
    GPIO.output(ledPin, GPIO.HIGH)
```

```
  else:
    GPIO.output(ledPin, GPIO.LOW)
```

Listing 20. GPIO operations using Python - falling edge detection.

```
#!/usr/bin/python3
#**
#*******************************************************************************
#* @file    /gpio_examples/python/gpio_input_example.py
#* @author  Adrian Wojcik
#* @version V1.0
#* @date    15-Mar-2020
#* @brief   Raspberry Pi digital input control: Python 3 with RPi.GPIO lib
#*******************************************************************************

import time
import sys
import select
try:
  import RPi.GPIO as GPIO
except RuntimeError:
  print("Error importing RPi.GPIO! Use 'sudo' to run your script")

timeout = 0.1
buttonState = True
buttonStateLast = True
cnt = 0
i = ''

# Pin Definitons:
buttonPin = 36 #< Push-button: Physical pin 36, BCM GPIO16

# Pin Setup:
GPIO.setmode(GPIO.BOARD)
GPIO.setup(buttonPin, GPIO.IN) # Button pin set as input

print("Press ENTER to exit.")

while not i:
  # Waiting for I/O completion
  i, o, e = select.select( [sys.stdin], [], [], timeout )

  if (i):
    sys.stdin.readline();
    GPIO.cleanup() # cleanup all GPIO
    exit()

  buttonState = (GPIO.input(buttonPin) == GPIO.HIGH)

  if (buttonState is False) and (buttonStateLast is True):
    cnt = cnt + 1
    print("Push-button counter:",cnt)

  buttonStateLast = buttonState
```

Listing 21. GPIO operations using Python - LED toggle after edge detection.

```
#!/usr/bin/python3
#**
#*****************************************************************************
#* @file    /gpio_examples/python/gpio_example.py
#* @author  Adrian Wojcik
#* @version V1.0
#* @date    15-Mar-2020
#* @brief   Raspberry Pi GPIO control: Python 3 with RPi.GPIO lib
#*****************************************************************************

import time
import sys
import select
try:
  import RPi.GPIO as GPIO
except RuntimeError:
  print("Error importing RPi.GPIO! Use 'sudo' to run your script")

timeout = 0.1
buttonState = True
buttonStateLast = True
ledState = False
ledStateName = ["OFF", "ON"]
i = ''

# Pin Definitons:
ledPin = 38    #< LED: Physical pin 38, BCM GPIO28
buttonPin = 36 #< Push-button: Physical pin 36, BCM GPIO16

# Pin Setup:
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output
GPIO.setup(buttonPin, GPIO.IN) # Button pin set as input

GPIO.output(ledPin, ledState)

print("Press ENTER to exit.")

while not i:
  # Waiting for I/O completion
  i, o, e = select.select( [sys.stdin], [], [], timeout )

  if (i):
    sys.stdin.readline();
    GPIO.cleanup() # cleanup all GPIO
    exit()

  buttonState = (GPIO.input(buttonPin) == GPIO.HIGH)

  if (buttonState is False) and (buttonStateLast is True):
    ledState = not ledState
    GPIO.output(ledPin, ledState)
    print("LED state:",ledStateName[ledState])

  buttonStateLast = buttonState
```

For programs written in C++, an external WiringPi library was used. It is worth noting that it is also (*unofficially*) available for Python. When you install this library, a set of additional tools is available, including the **gpio** application. It enables GPIO support directly from the terminal level in a way significantly simpler than direct files read/write. When using WiringPi tools, it is always worth making sure which GPIO numbering system we use - this is facilitated by the `gpio readall` command. Result for RPi model 3B is shown in fig 3.

RaspberryPi_SSH_profile.tlp - pi@192.168.1.189:22 - Bitvise xterm - pi@raspberrypi: ~

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

Fig. 3. WiringPi connector numbering - RPi model 3B.

Listings 22.-24. shows examples of a simple application for a digital output, a digital input and and output control by input edge, using C++ with the WiringPi library.

Listing 22. GPIO operations using C++ - LED toggle.

```cpp
/**
 ******************************************************************************
 * @file     /gpio_examples/cpp/gpio_output_example.cpp
 * @author   Adrian Wojcik
 * @version  V1.0
 * @date     15-Mar-2020
 * @brief    Raspberry Pi digital output control: C++ with wiringPi lib
 ******************************************************************************
 */

#include <iostream>
#include <future>
#include <thread>
#include <chrono>
#include <wiringPi.h>

int main()
{
  const int led = 28; //< Red LED: Physical pin 38, BCM GPIO20, and WiringPi pin
      28.

  std::chrono::milliseconds timeout(100);
  std::future<int> async_getchar = std::async(std::getchar);

  wiringPiSetup();

  pinMode(led, OUTPUT);

  std::cout << "Press ENTER to exit." << std::endl;

```

```
  while(1)
  {
    digitalWrite(led, HIGH);

    std::this_thread::sleep_for(timeout);

    digitalWrite(led, LOW);

    if(async_getchar.wait_for(timeout) == std::future_status::ready)
    {
      async_getchar.get();
      break;
    }
  }

  return 0;
}
```

Listing 23. GPIO operations using C++ - falling edge detection.

```
/**
 ******************************************************************************
 * @file     /gpio_examples/cpp/gpio_input_example.cpp
 * @author   Adrian Wojcik
 * @version  V1.0
 * @date     15-Mar-2020
 * @brief    Raspberry Pi digital input control: C++ with wiringPi lib
 ******************************************************************************
 */

#include <iostream>
#include <future>
#include <thread>
#include <chrono>
#include <wiringPi.h>

int main()
{
  const int button = 27; //< Push-button: Physical pin 36, BCM GPIO16, and
      WiringPi pin 27.

  bool gpio_state = true, gpio_state_last = true;
  unsigned int cnt = 0;

  std::chrono::milliseconds timeout(100);
  std::future<int> async_getchar = std::async(std::getchar);

  wiringPiSetup();

  pinMode(button, INPUT);

  std::cout << "Press ENTER to exit." << std::endl;

  while(1)
  {
    gpio_state = (digitalRead(button) == HIGH);

    if( !gpio_state && gpio_state_last)
    {
      cnt++;
      std::cout << "Push-button counter: " << cnt << std::endl;
    }

    gpio_state_last = gpio_state;
```

```

    if(async_getchar.wait_for(timeout) == std::future_status::ready)
    {
      async_getchar.get();
      break;
    }
  }

  return 0;
}
```

Listing 24. GPIO operations using C++ - LED toggle after edge detection.

```
/**
 ******************************************************************************
 * @file    /gpio_examples/cpp/gpio_example.cpp
 * @author  Adrian Wojcik
 * @version V1.0
 * @date    15-Mar-2020
 * @brief   Raspberry Pi GPIO control: C++ with wiringPi lib
 ******************************************************************************
 */

#include <iostream>
#include <future>
#include <thread>
#include <chrono>
#include <wiringPi.h>

int main()
{
  const int led = 28;    //< Red LED: Physical pin 38, BCM GPIO20, and WiringPi
      pin 28.
  const int button = 27; //< Push-button: Physical pin 36, BCM GPIO16, and
      WiringPi pin 27.

  bool btn_state = true, btn_state_last = true;
  bool led_state = false;

  std::chrono::milliseconds timeout(100);
  std::future<int> async_getchar = std::async(std::getchar);

  wiringPiSetup();

  pinMode(button, INPUT);
  pinMode(led, OUTPUT);

  digitalWrite(led, LOW);

  std::cout << "Press ENTER to exit." << std::endl;

  while(1)
  {
    btn_state = (digitalRead(button) == HIGH);

    if( !btn_state && btn_state_last){
      led_state = !led_state;
      digitalWrite(led, led_state);
      std::cout << "LED state: ";
      if(led_state)
        std::cout<< "ON" << std::endl;
      else
        std::cout << "OFF" << std::endl;
    }
```

```cpp

    btn_state_last = btn_state;

    if(async_getchar.wait_for(timeout) == std::future_status::ready)
    {
      async_getchar.get();
      break;
    }
  }

  return 0;
}
```

2. PWM

RPi is equipped with 2 PWM channels, each of can be mapped to one of 2 different pins. L 25. shows a bash system shell script using the **gpio** application to configure PWM1 - the output operates with constant frequency of 500 Hz and duty specified by the user with the script's input argument (input validation is omitted). Listing 26. shows sample application - piecewise-linear periodic change of duty.

*Listing 25. PWM operations using bash system shell and **gpio** app - duty setting.*

```bash
#!/bin/bash
#**
#******************************************************************************
#* @file     /pwm_examples/bash/pwm_setDuty_example.sh
#* @author   Adrian Wojcik
#* @version  V1.0
#* @date     15-Mar-2020
#* @brief    Raspberry Pi PWM control: bash with gpio app
#******************************************************************************

gpio -g mode 18 pwm
gpio pwm-ms
# pwmFrequency in Hz = 19 200 000 Hz / pwmClockDiv / pwmCounter.
gpio pwmc 192    # pwmClockDiv
gpio pwmr 200   # pwmCounter

if [ $# -eq 1 ] ; then
  pwm_duty=$(($1 * 2))
  gpio -g pwm 18 $pwm_duty
else
  gpio -g pwm 18 100
fi

echo "Press any key to continue"

while [ true ] ;
do
  read -t 1 -n 1
  if [ $? = 0 ] ; then
    break ;
  fi
done

gpio -g mode 18 out
gpio unexport 18
```

*Listing 26. PWM operations using bash system shell and **gpio** app - periodic duty change.*

```bash
#!/bin/bash
#**
#***************************************************************************
#* @file     /pwm_examples/bash/pwm_example.sh
#* @author   Adrian Wojcik
#* @version V1.0
#* @date     15-Mar-2020
#* @brief    Raspberry Pi PWM control: bash with gpio app
#***************************************************************************

gpio -g mode 18 pwm
gpio pwm-ms
# pwmFrequency in Hz = 19 200 000 Hz / pwmClockDiv / pwmCounter.
gpio pwmc 192    # pwmClockDiv
gpio pwmr 200   # pwmCounter

gpio -g pwm 18 0

echo "Press any key to continue"

while [ true ] ;
do
  read -t 1 -n 1
  if [ $? = 0 ] ; then
    break ;
  else

    for i in $(seq 0 1 200)
    do
      gpio -g pwm 18 ${i}
    done

    for i in $(seq 200 -1 0)
    do
      gpio -g pwm 18 ${i}
    done

  fi
done

gpio -g mode 18 out
gpio unexport 18
```

Listings 27.-28. shows similar applications in Python using the RPi.GPIO library.

Listing 27. PWM operations using Python - duty setting.

```python
#!/usr/bin/python3
#**
#***************************************************************************
#* @file     /pwm_examples/python/pwm_example.py
#* @author   Adrian Wojcik
#* @version V1.0
#* @date     15-Mar-2020
#* @brief    Raspberry Pi PWM control: Python 3 with RPi.GPIO lib
#***************************************************************************

import time
import sys
import select
import numpy

```

```
try:
  import RPi.GPIO as GPIO
except RuntimeError:
  print("Error importing RPi.GPIO! Use 'sudo' to run your script")

timeout = 1
i = ''
duty = 0 # [%]
freq = 500 # [Hz]

# Pin Definitons:
pwmPin = 12    #< LED: Physical pin 12, BCM GPIO18

GPIO.setmode(GPIO.BOARD)
GPIO.setup(pwmPin, GPIO.OUT)

p = GPIO.PWM(pwmPin, freq)
p.start(duty)

print("Press ENTER to exit.")

while not i:
  # Waiting for I/O completion
  i, o, e = select.select( [sys.stdin], [], [], timeout )

  if (i):
    sys.stdin.readline();
    p.stop()
    GPIO.cleanup() # cleanup all GPIO
    exit()

  for d in numpy.arange(0, 100, 1):
    #print(d)
    p.ChangeDutyCycle(d)
    time.sleep(0.02)
  for d in numpy.arange(100, -1, -1):
    #print(d)
    p.ChangeDutyCycle(d)
    time.sleep(0.02)
```

Listing 28. PWM operations using Python - periodic duty change.

```
#!/usr/bin/python3
#**
#******************************************************************************
#* @file     /pwm_examples/python/pwm_example.py
#* @author   Adrian Wojcik
#* @version V1.0
#* @date     15-Mar-2020
#* @brief    Raspberry Pi PWM control: Python 3 with RPi.GPIO lib
#******************************************************************************

import time
import sys
import select
import numpy

try:
  import RPi.GPIO as GPIO
except RuntimeError:
  print("Error importing RPi.GPIO! Use 'sudo' to run your script")

timeout = 1
i = ''
```

```
duty = 0 # [%]
freq = 500 # [Hz]

# Pin Definitons:
pwmPin = 12     #< LED: Physical pin 12, BCM GPIO18

GPIO.setmode(GPIO.BOARD)
GPIO.setup(pwmPin, GPIO.OUT)

p = GPIO.PWM(pwmPin, freq)
p.start(duty)

print("Press␣ENTER␣to␣exit.")

while not i:
  # Waiting for I/O completion
  i, o, e = select.select( [sys.stdin], [], [], timeout )

  if (i):
    sys.stdin.readline();
    p.stop()
    GPIO.cleanup() # cleanup all GPIO
    exit()

  for d in numpy.arange(0, 100, 0.5):
    p.ChangeDutyCycle(d)
    time.sleep(0.01)
  for d in numpy.arange(100, 0, -0.5):
    p.ChangeDutyCycle(d)
    time.sleep(0.01)
```

Listings 29.-29. shows similar applications in C++ using the WiringPi library.

Listing 29. PWM operations using C++ and WiringPi - duty setting.

```
/**
 ******************************************************************************
 * @file     /pwm_examples/cpp/pwm_setDuty_example.cpp
 * @author   Adrian Wojcik
 * @version V1.0
 * @date     15-Mar-2020
 * @brief    Raspberry Pi PWM control: C++ with wiringPi lib
 ******************************************************************************
 */

#include <iostream>
#include <future>
#include <thread>
#include <chrono>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  const int pwmPin = 1; //< Red LED: Physical pin 12, BCM GPIO18, and WiringPi
      pin 1.
  float duty = 0;
  const int range = 200;
  const int clock = 192;

  sscanf(argv[1],"%f",&duty); // C-style
  duty *= (float)range;
  duty /= 100.0;

```

```
  std::chrono::milliseconds timeout(100);
  std::future<int> async_getchar = std::async(std::getchar);

  wiringPiSetup();

  pinMode(pwmPin, PWM_OUTPUT);

  pwmSetMode(PWM_MODE_MS);
  pwmSetRange(range);
  pwmSetClock(clock);

  pwmWrite(pwmPin, (int)duty) ;

  std::cout << "Press␣ENTER␣to␣exit." << std::endl;

  while(1)
  {
    if(async_getchar.wait_for(timeout) == std::future_status::ready)
    {
      async_getchar.get();
      break;
    }
  }

  return 0;
}
```

Listing 30. PWM operations using C++ and WiringPi - periodic duty change.

```
/**
 ******************************************************************************
 * @file     /pwm_examples/cpp/pwm_example.cpp
 * @author   Adrian Wojcik
 * @version V1.0
 * @date     15-Mar-2020
 * @brief    Raspberry Pi PWM control: C++ with wiringPi lib
 ******************************************************************************
 */

#include <iostream>
#include <future>
#include <thread>
#include <chrono>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  const int pwmPin = 1; //< Red LED: Physical pin 12, BCM GPIO18, and WiringPi
      pin 1.
  const int range = 200;
  const int clock = 192;
  int step = 1;

  std::chrono::milliseconds timeout(1000);
  std::chrono::milliseconds delay(10);
  std::future<int> async_getchar = std::async(std::getchar);

  wiringPiSetup();

  pinMode(pwmPin, PWM_OUTPUT);

  pwmSetMode(PWM_MODE_MS);
  pwmSetRange(range);
  pwmSetClock(clock);
```

```

  pwmWrite(pwmPin, 0) ;

  std::cout << "Press␣ENTER␣to␣exit." << std::endl;

  while(1)
  {
    if(async_getchar.wait_for(timeout) == std::future_status::ready)
    {
      async_getchar.get();
      break;
    }

    for(int d = 0 ; d <= range ; d+=step)
    {
      pwmWrite(pwmPin, d) ;
      std::this_thread::sleep_for(delay);
    }

    for(int d = range ; d >= 0 ; d-=step)
    {
      pwmWrite(pwmPin, d) ;
      std::this_thread::sleep_for(delay); ;
    }
  }

  return 0;
}
```

F) Web server configuration - Lighttpd

Using RPi as the Internet of Things (IoT) server requires configuration of web server and CGI scripts. In this course we will use Lighttpd - an open source web server dedicated to platforms with limited hardware resources, while maintaining compliance with standards, security and flexibility [16]. Laboratory kits (and virtual machines) **do not require additional installation**. You can install the Lighttpd application on the Raspberry Pi OS system with the `apt-get install lighttpd` command. To enable the use of PHP scripts on the server, you must install the necessary software with the command `apt-get -y install php7.x-fpm php7.x`, where **x** is the version number of PHP7 (currently 0-4).

The basic server configuration is available in the file `/etc/lighttpd/lighttpd.conf` (listing 31.). When modifying (overwriting) configuration files, it's a good practice to make a backup copy of them beforehand:
`cp filename.conf filename.conf.bak`,
to restore defaults later if needed:
`cp filename.conf.bak filename.conf`.
Working with the server, it is worth modifying the workspace directory (*server.document-root)* to one that we will have full access to from the SSH client. Important information is also server user name (*server.username*) and group name (*server.groupname*). This is key information in the context of giving the server appropriate permissions.

Listing 31. Default contents of configuration file /etc/lighttpd/lighttpd.conf

```
/var/www/htmlserver.modules = (
        "mod_access",
        "mod_alias",
        "mod_compress",
        "mod_redirect",
)

server.document-root        = "/var/www/html"
server.upload-dirs          = ( "/var/cache/lighttpd/uploads" )
server.errorlog             = "/var/log/lighttpd/error.log"
server.pid-file             = "/var/run/lighttpd.pid"
server.username             = "www-data"
server.groupname            = "www-data"
server.port                 = 80


index-file.names            = ( "index.php", "index.html", "index.lighttpd.html" )
url.access-deny             = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

compress.cache-dir   = "/var/cache/lighttpd/compress/"
compress.filetype    = ( "application/javascript", "text/css", "text/html", "text/plain" )

# default listening port for IPv6 falls back to the IPv4 port
include_shell "/usr/share/lighttpd/use-ipv6.pl " + server.port
include_shell "/usr/share/lighttpd/create-mime.assign.pl"
include_shell "/usr/share/lighttpd/include-conf-enabled.pl"
```

If we want the server to be able to execute programs and scripts interacting with the file system and hardware (GPIO, PWM, UART, I2C, SPI), the user (here: *www-data*) must be given appropriate permissions. We can edit user permissions by running the `sudo visudo` command. In order to give the user the permission to execute a given program, add the following line:
`www-data ALL = (root)NOPASSWD: /path/to/my/program_or_script`.
Separate file names with a comma.

NOTE! At the stage of development and testing of the server application, a convenient option is to **temporary** give the server root privileges: `www-data ALL = (ALL)NOPASSWD: ALL`. However, this is an absolutely unacceptable situation in the final product, for safety reasons. Do not use this configuration while accessing the Internet!

Listing 32. shows an example PHP script using system shell scripts to operate GPIO. Fig 4. shows script execution result in Firefox for various arguments: *ON* and *OFF*. If we provide the appropriate GPIO configuration in advance and connect the LED diode according to the diagram (Fig. 2), we obtain the ability to control the LED state from the level of every device equipped with a web browser connected to the same **local** network.

Listing 32. An example of a PHP script controlling a LED (via GPIO) with a system shell script.

```
<?php
echo "Simple␣server-side␣PHP␣control␣scirpt:<br>";
echo "LED␣control<br>";

$led_state='';
if(isset($_GET['state']))
  $led_state=$_GET['state'];

if(strcmp($led_state, "ON") == 0){
  exec('sudo␣../gpio_examples/bash/./gpio_output_set.sh');
  echo "LED␣state:␣ON";
}
elseif(strcmp($led_state, "OFF") == 0){
  exec('sudo␣../gpio_examples/bash/./gpio_output_reset.sh');
  echo "LED␣state:␣OFF";
}
else {
  echo "LED␣state␣undefiend";
}
?>
```

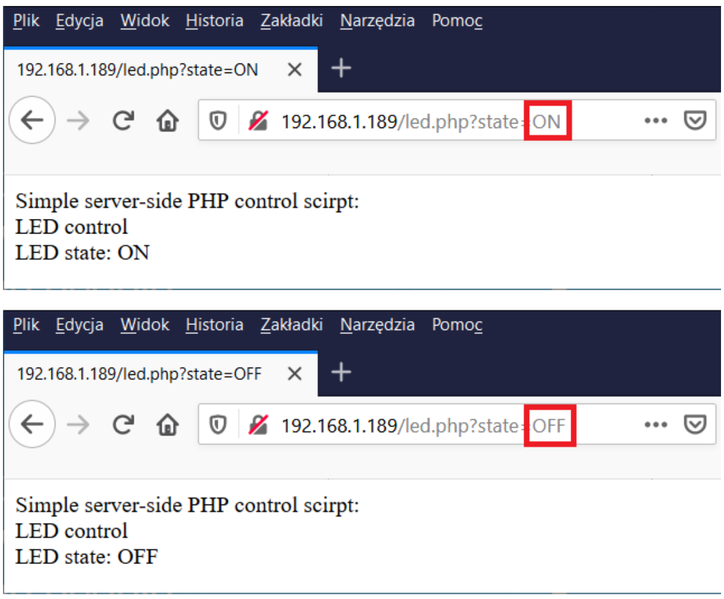


Fig. 4. Server response - script controlling LED state (via GPIO).

IV. Scenario for the class

A) Teaching resources

Hardware
- computer,
- Raspberry Pi single-board computer set,
- miroSD card,
- microUSB 5V power supply,
- RJ45 cable,
- WiFi router,
- *set of electronic components,*
- *multimeter,*
- *oscilloscope,*

Software
- SSH client (e.g. Bitwise SSH Client),
- application for writing binary disk images to external media (e.g. Win32DiskMaker),
- text editor (Notepad++, VS Code),
- VirtualBox application and virtual machine with Raspberry Pi OS for Desktop system.

B) Tasks

Physical device

1. Start the Raspberry Pi: prepare the memory card with the operating system and connect to the device via the SSH client.
   (a) Download the disk image with the configured Raspberry Pi OS system provided by the instructor.
   (b) Unpack the archive and write the image on a microSD card (min. 16 GB) using an appropriate tool (e.g. Win32DiskMaker).
   (c) Save on the card the appropriate configuration file enabling access to the local WiFi network, according to tutorial available at:
   `www.raspberrypi.org/documentation/configuration/wireless/headless.md`
   (d) Place prepared card in the slot, connect the RPi power supply and wait several dozen seconds
   (e) Check the IP address of the Raspberry Pi on the local network.
   (f) Connect to RPi using an SSH client (e.g. Bitvise SSH Client). Use the default port 22. Username: **pi**, password: **raspberry** (default settings).
   (g) After starting the terminal, check the configuration of network interfaces.
   (h) Check if RPi can access the Internet using **ping**.
2. Configure the network interface - set a static IP and connect to the device via the SSH client.
   (a) Configure network interface `eth0` - set a static IP (e.g. 192.168.1.15/24) and modify the appropriate configuration file.
   (b) Set a static IP on a desktop computer (e.g. 192.168.1.100/24). Connect desktop and RPi with a network cable.
   (c) Reset `eth0` and check the configuration of network interfaces.

(d) Connect to RPi using an SSH client (e.g. Bitvise SSH Client). Use the default port 22.

(e) Check if RPi can access the Internet using **ping**. Repeat the test after the `wlan0` interface is disabled.

Virtual device

1. Start virtual Raspberry Pi OS - create a new virtual machine and connect to the device via an SSH client.

   (a) Download the image of a virtual disk with Raspberry Pi OS for Desktop, provided by the instructor.

   (b) Create a new virtual machine for Linux (Debian, 32-bit). Use the downloaded virtual disk.

   (c) Configure the network interface of the virtual machine in such a way that it provides access to the host and the Internet (e.g. bridged network card).

   (d) Check the configuration of network interfaces directly in the virtual machine terminal.

   (e) Connect to virtual RPi using an SSH client (e.g. Bitvise SSH Client). Use the default port 22. Username: **pi**, password: **raspberry** (default settings).

   (f) After starting the terminal, check the configuration of network interfaces.

   (g) Check if RPi can access the Internet using **ping**.

2. Configure the network interface - set a static IP and connect to the device via the SSH client.

   (a) Add a second network adapter to the virtual appliance. The card should only provide access to the host.

   (b) Configure the network interface `eth1` - set a static IP (e.g. 192.168.56.15/24) and modify the appropriate configuration file.

   (c) Reset `eth1` and check the configuration of network interfaces.

   (d) Connect to RPi using an SSH client (e.g. Bitvise SSH Client). Use the default port 22.

   (e) Check if RPi can access the Internet using **ping**. Repeat the test after the `eth1` interface is disabled.

3. Write a command line application using the bash shell.

   (a) Create three text files: `temperature.dat humidity.dat pressure.dat`.

   (b) Put the following content in them: `20.5C 80% 1023.0hPa`.

   (c) The application should support the following (optional) arguments: `-t -h -p`. If a given flag is present, it means that the application should read the appropriate text file and display its content (in the separate lines). So if there are no arguments, the program does not perform any action.

   (d) Extend the application with the option to select the temperature unit - after the `-t` option, the unit should be given: `C` (degrees Celsius) or `F` (degrees Fahrenheit). Based on this selection, the program should display the original or scaled value saved in the file.

4. Write a command line application using C++ and the g++ compiler.

   (a) Write an application that meets the same specification as the script from the previous task.
   (b) Create a script to compile the program. Give the resulting program an appropriate name.
   (c) Use the test files from the previous task.

5. Write a command line application using Python.
   (a) Write an application that meets the same specification as the script and program from the previous tasks.
   (b) Use the test files from the previous tasks.

6. **(*)** Write a simple GPIO application.
   (a) Build the circuit shown in the Fig. 2.
   (b) Test presented in this instruction scripts and programs.
   (c) Write your own command line application: use the button to set the period of the LED state change.
   (d) Add input arguments handling allowing for the application configuration, i.e. setting the maximum and minimum period of the diode blinking and step it should change after each pressing of the button.

7. **(*)** Write a simple PWM application.
   (a) Build the circuit shown in the Fig. 2.
   (b) Test presented in this instruction scripts and programs.
   (c) Write your own command line application: setting both frequency and duty with input arguments.

8. Write a CGI script to support the LED.
   (a) Check the Lighttpd server configuration and make sure the files in `document-root` are accessible from the host's web browser.
   (b) Create CGI scripts (PHP or Python) to write the value "1" or "0" to the file, passed by the GET method. You can use system shell commands for this. If you are working with a physical device, the script should control the digital output.
   (c) The script should respond with JSON containing the file name (or output number) and the current state after executing script. Remember about an adequate HTTP header.

References

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