Usługi i aplikacje Internetu rzeczy (PBL5)

Testowanie systemu Internetu Rzeczy

Aleksander Pruszkowski

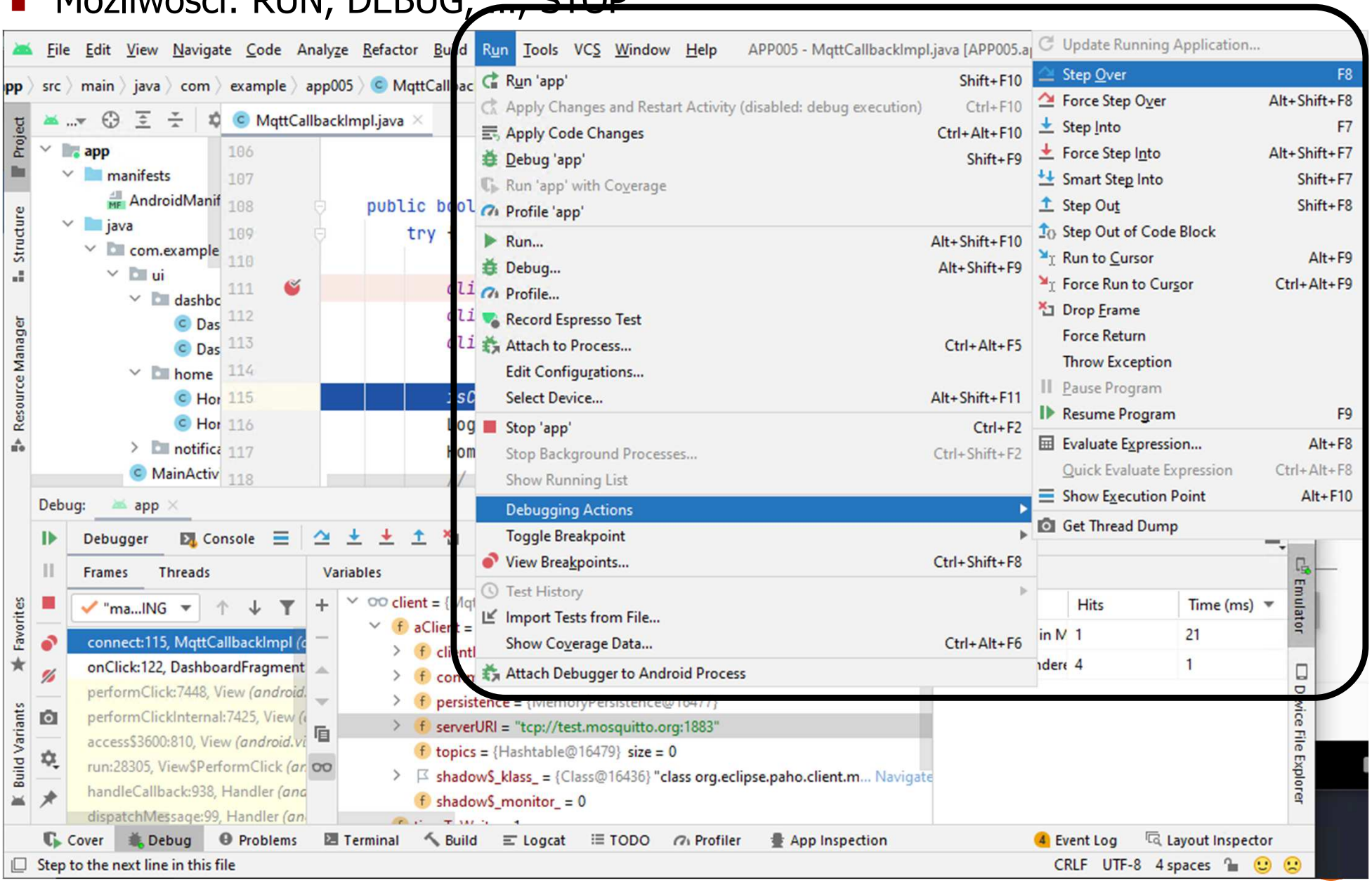
Instytut Telekomunikacji Politechniki Warszawskiej

Testowanie systemu - cześć I

- Android – Uruchamianie aplikacji
  - Po utworzeniu kodu programista powinien go uruchomić
    - Nie ma tu mowy o poleceniu „RUN” ale o zaawansowanym przejściu przez ewentualne błędy pozostawione w kodzie i ich usunięciu
  - Aplikacje dla systemu Android można uruchamiać w tzw. „pracy korkowej” z wykorzystaniem narzędzia „DEBUG” oraz za pomocą klasy „android.util.Log” i infrastruktury ADB

Testowanie systemu - cześć I

- Android – Uruchamianie aplikacji - DEBUG
  - Możliwości: RUN, DEBUG, ..., STOP



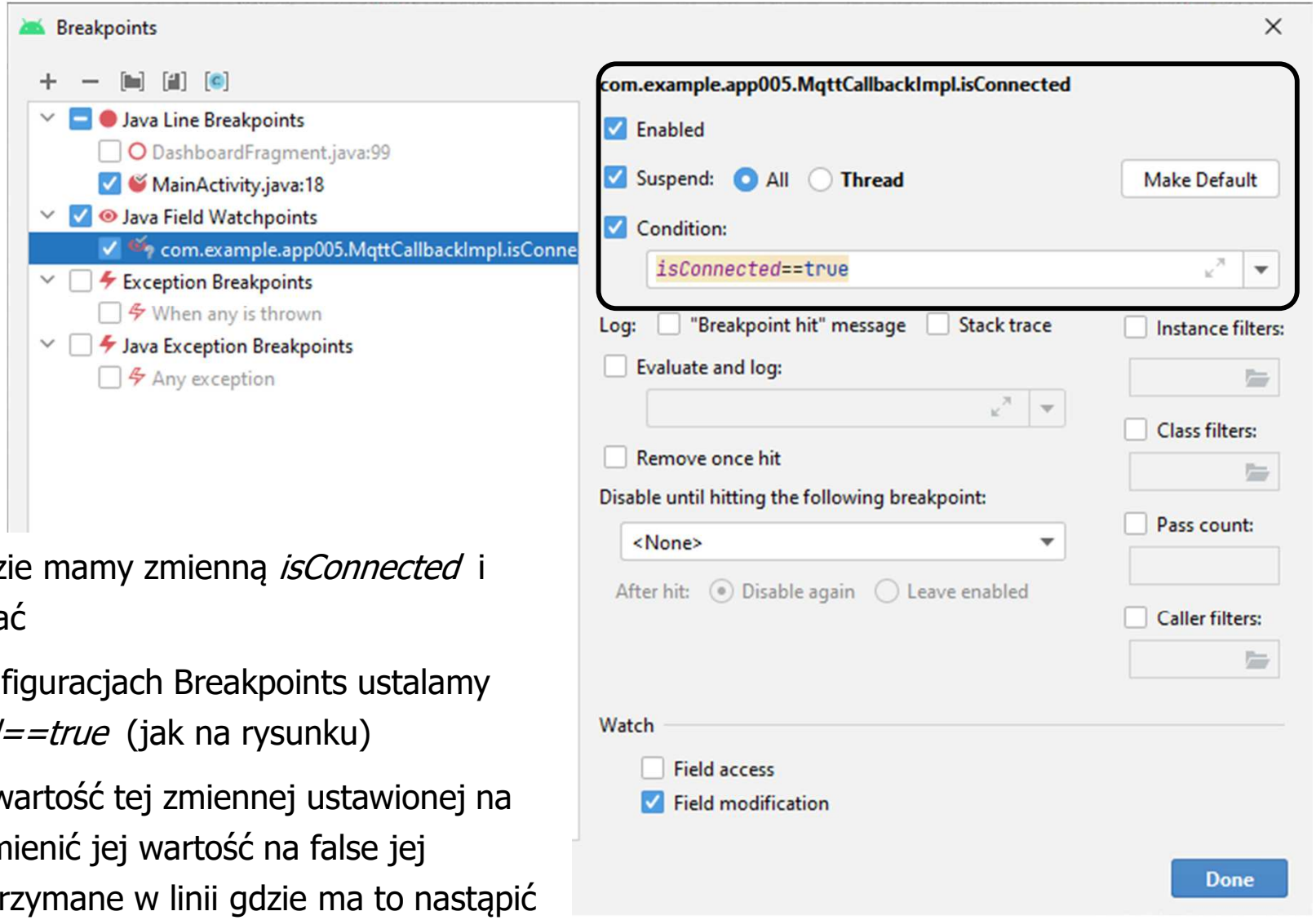
Testowanie systemu - cześć I

- Android – Uruchamianie aplikacji - DEBUG
  - Możemy dokonać pracy krokowej
    - Przechodząc linia po linii – mało praktyczne
    - Poprzez ustawienie tzw. pułapek w kodzie (Breakpoint) – ustawiany w miejscu kodu na którym ma zatrzymać się system
    - Poprzez ustawienie tzw. pułapek na zmiennych (Watchpoint) – ustawiamy na zmiennej której zmiana podczas działania aplikacji zatrzyma system na miejscu gdzie nastąpiła jej zmiana
      - Uwaga emulator potrafi znacząco zwolnić
    - TIPS: Warto ustawić w kodzie przynajmniej jeden Breakpoint a Watchpoint'ów tyle ile potrzeba

Testowanie systemu - cześć I

- Android – Uruchamianie aplikacji - DEBUG
  - Jak uruchomić aplikacje dla Pracy krokowej - wybieramy „SHIFT-F9” potem
    - F9 – „Resume Program” czyli wznowienie wykonywania programu od miejsca jego zatrzymania
    - F8 (step over) – przejście „ponad” metodami, czyli gdy nie sprawdzamy implementacji danej metody a tylko ja wykonujemy w jednym kroku, przeciwność do F7 (step into) gdzie system przejdzie przez implementację aktualnie wykonywanej metody
      - Uwaga - używając wyłącznie F7 będziemy wchodzili w każdą metodę łącznie z tymi systemowymi
    - CTRL+F2 – zatrzymanie pracy krokowej – niezbędne gdy chcemy dokonać poprawkę w kodzie
    - CTRL+SHIFT+F8 – podgląd Breakpoint/Watchpoint i ich ewentualna konfiguracja

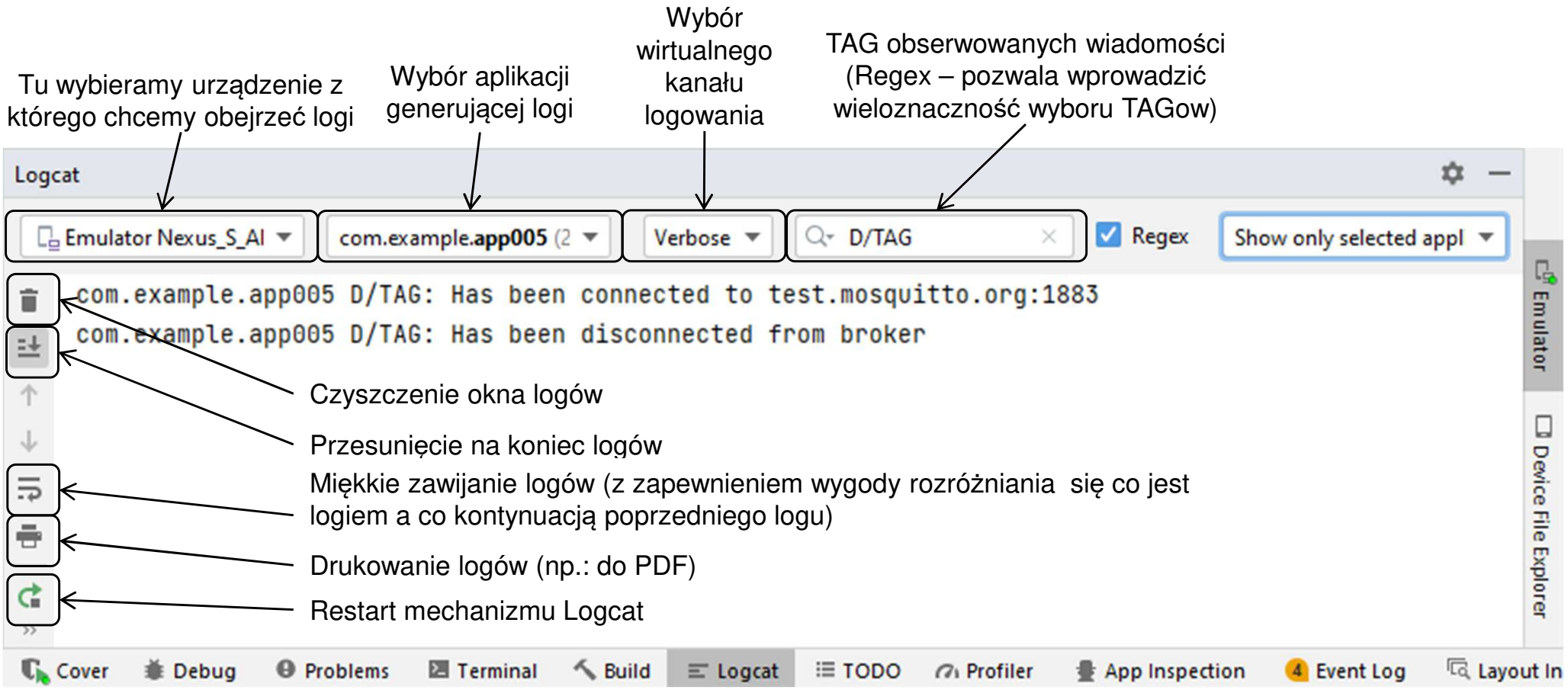
- Android – Uruchamianie aplikacji – DEBUG
  - Konfiguracja Breakpoints/Watchpoints (CTRL+SHIFT+F8)



- Zakładamy, że w kodzie mamy zmienną *isConnected* i chcemy ją obserwować
- Wybieramy ją i w konfiguracjach Breakpoints ustalamy warunek *isConnected==true* (jak na rysunku)
- Gdy aplikacja mając wartość tej zmiennej ustawionej na true będzie chciała zmienić jej wartość na false jej działanie zostanie zatrzymane w linii gdzie ma to nastąpić
  - Pozornie może to wyglądać na działanie odwrotne!

Testowanie systemu - cześć I

- Android – Uruchamianie aplikacji – Mechanizm Logcat
  - Dostęp do logów - przez wybranie „**View” > „Tool Windows” > „Logcat”**



Testowanie systemu - cześć I

- Android – Uruchamianie aplikacji – Logcat oprogramowanie
  - Logi maja kategoryzacje (wybieralną przez - patrz poprzedni slajd: „Wybór wirtualnego kanału logowania„):
    - Błędy (Errors)

```
Log.e(String, String)
```

    - Ostrzeżenia (Warrnigs)

```
Log.w(String, String)
```

    - Informacje (Info)

```
Log.i(String, String)
```

    - Diagnostyka (Diagnostics)

```
Log.d(String, String)
```

    - Komunikaty (Verbose)

```
Log.v(String, String)
```

- Android – Uruchamianie aplikacji – Logcat oprogramowanie, cd.
  - Przykład użycia w kodzie

```
package com.example.app008;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {
   private static final String TAG = "TAGmain" ;
   @Override protected void onCreate(Bundle savedInstanceState) {
       Log.d(TAG, "onCreate was invoked");
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
    }
}
```

  - Lub bardziej zaawansowane wywołanie

```
private static final String TAG = "TAGmqtt";
...
Log.d(TAG2, "Has been connected to "+domain+":"+port);
```

Tagi dają możliwość rozróżniania wiadomości należących do jednego mechanizmu w ramach tej samej aplikacji (wiele klas może wspierać ten jeden mechanizm)
Tu także można wykorzystać wieloznaczność TAGów – aby wspólnie obserwować podobne logi

- Android – Uruchamianie aplikacji – Logcat i ADB
  - Narzędzie ADB pozwala na wiele, np.:
    - Podgląd „osiągalnych” urządzeń:

```
>adb.exe devices -l
List of devices attached
emulator-5554  device product:sdk_phone_x86 ...
```

    - Połączenie z shell’em emulatora (opcja „-e” pozwala wybrać emulator)

```
>adb.exe -e shell
generic_x86:/ $ uname -a
Linux localhost 5.4.61-android11-2-00094-gd61ede24cbb2-ab7064445 #1
SMP PREEMPT Wed Jan 6 00:39:01 UTC 2
021 i686
generic_x86:/ $ exit
```

Tu można wydać polecenie *reboot –p* przydatne aby wykonać czyszczenie emulatora poprzez „Wipe data”

    - Połączenie z Logcat’em (w kodzie logujemy stosując TAG: `TAGmain)` – przydatne gdy nie chcemy uruchamiać całego Android Studio

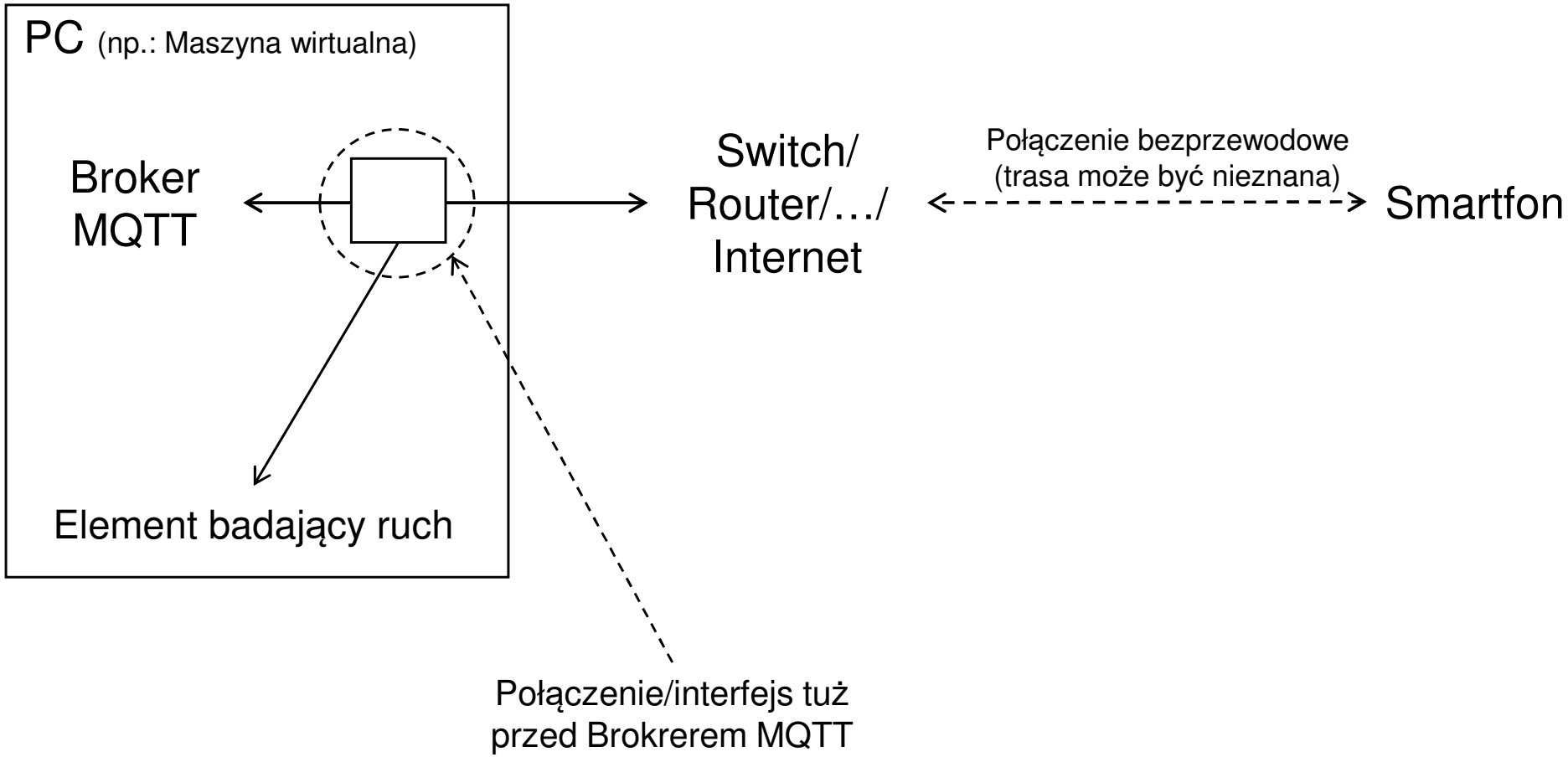
```
>adb.exe -e logcat TAGmain:* ActivityManager:E *:S
12-12 14:14:59.290 24025 24025 D TAGmain : onCreate was invoked
```

    - Więcej na: https://developer.android.com/studio/command-line/logcat

Testowanie systemu - cześć I

Zadanie:
Dodać do APP007/ APP006 w przemyślanych miejscach komunikaty diagnostyczne, t.j. przynajmniej dla zdarzeń: a)zestawienie połączenia z brokrem MQTT, b)zerwanie połączenia z brokerem (emulacje zerwania połączenia najwygodniej wykonać poprzez wyłączenie brokera albo przez wyłączenie połączenia sieciowego), c)pomyślne/niepomyślne zasubskrybowanie wiadomości w zadanym temacie, d)pomyślne/niepomyślne opublikowanie widomości). W raporcie pokazać „zrzut” działania mechanizmu logowania z zaznaczeniem kiedy dany komunikat się pojawił.

- Badanie ruchu sieciowego
  - Gdy chcemy badać ruch sieciowy możemy użyć narzędzia do łapania tego ruchu



Testowanie systemu - cześć I

- Badanie ruchu sieciowego, cd.
  - Tcpdump - gdy na maszynie badającej zasoby mamy skromne zasoby
    - Program „łapie” pakiety a ich wizualizacja jest bardzo skromna

```
su –
```

`/usr/sbin/tcpdump –n –i enp0s8 "not tcp port 22" –vvv`

- Efekt na konsoli (uwaga program uruchomiono bez zapisywania czegokolwiek do pliku):

```
tcpdump: listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
14:15:35.276302 IP (tos 0x0, ttl 64, id 7211, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.89.3.34244 > 192.168.89.5.12345: Flags [S], cksum 0x3388 (incorrect -> 0x657d),
    seq 3692844898, win 64240,options [mss 1460,sackOK,TS val 2546938604 ecr 0,nop,wscale 7],
    length 0
14:15:35.276770 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.89.5.12345>192.168.89.3.34244: Flags[S.], cksum 0xd9ad (correct), seq 3514462896,
    ack 3692844899, win 65160, options [mss 1460,sackOK,TS val 2078263323 ecr 2546938604,nop,
    wscale 7], length 0
14:15:35.276787 IP (tos 0x0, ttl 64, id 7212, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.89.3.34244 > 192.168.89.5.12345: Flags [.], cksum 0x3380 (incorrect -> 0x050c),
    seq 1, ack 1, win 502, options [nop,nop,TS val 2546938605 ecr 2078263323], length 0
...
```

Testowanie systemu - cześć I

- Badanie ruchu sieciowego, cd.
  - Tshark - gdy chcemy badać ruch sieciowy ale posiadamy GUI
    - Wireshark jest dostępny dla platform z interfejsem graficznym, czasami mamy dostęp do maszyn wyłącznie poprzez protokół SSH lub telnet
  - W grę wchodzi wtedy wyłącznie tryb tekstowy i polecenia wydawane w tym trybie

```
su –
```

Przełączamy się na użytkownika root (prawa dostępu do interfejsów sieciowych)

Jak zdobyć listę obsługiwanych przez narzędzie interfejsów: sudo tshark –D

```
/usr/bin/tshark –n –i enp0s8 –w pakiety.pcap "not tcp port 22"
```

```
chown student.student pakiety.pcap
mv pakiety.pcap /home/student
```

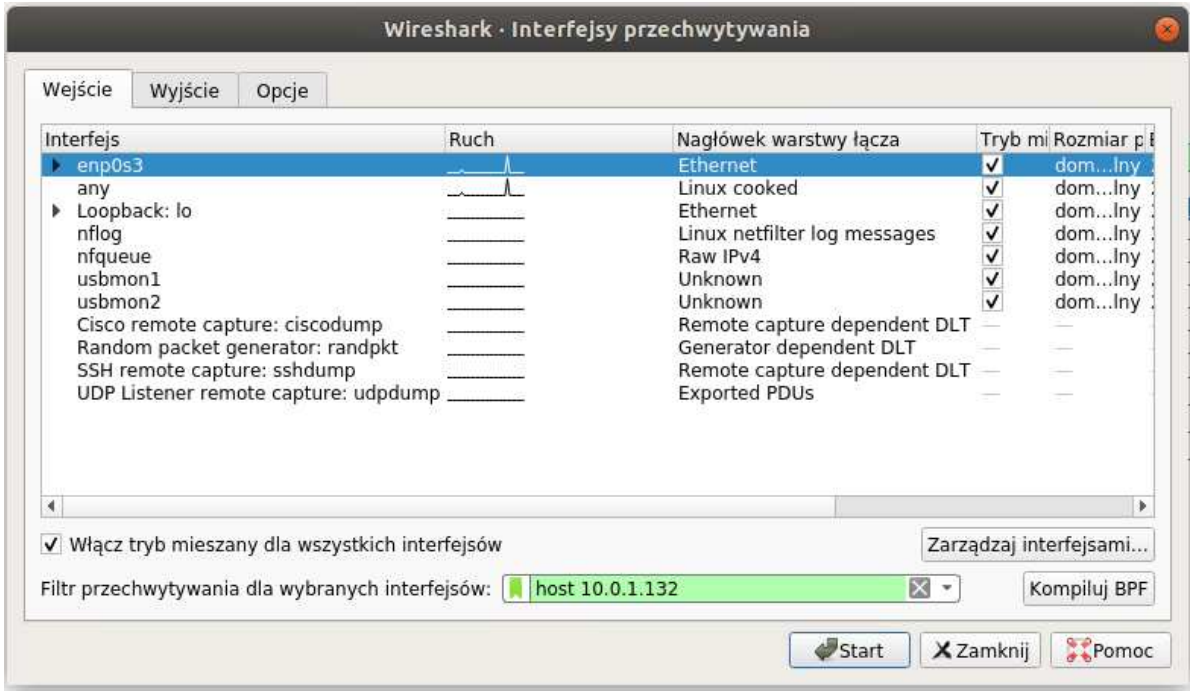
Zmiana praw własności i lokalizacji pliku PCAP (z root na student oraz przeniesienie pliku do katalogu domowego użytkownika student)

```
exit
```

- Tutaj podajemy w opcjach
  - -n – aby tshark nie próbował używać DNS podczas swojej pracy (użyteczne)
  - -i enp0s8 – na jakim interfejsie ruch ma być „łapany”
  - -w pakiety.pcap – nazwa pliku gdzie pakiety mają być umieszczane (uwaga na wolną przestrzeń)
  - `"not tcp port 22"` - jakiego ruchu mamy nie łapać – tu wybraliśmy „nie łapanie” ruchu na porcie 22 gdyż SSH połączyliśmy się z maszyną gdzie „łapiemy” pakiety
- Proces „łapania” przerywamy kiedykolwiek poprzez CTRL+C

Testowanie systemu - cześć I

- Badanie ruchu sieciowego, cd.
  - Wireshark to narzędzie które pozwala łapać i analizować poszczególne pakiety
    - Łapać można ruch na określonym interfejsie jak i z jakich/do jakich urządzeń ma być kierowany (tu interfejs enp0s3 i urządzenie o IP: 10.0.1.132)



  - Reguły filtru przechwytywania mogą być zaawansowane, np..:

```
(host 10.0.1.132 ) || (host 10.0.1.133)
```

Chcemy obserwować ruch związany z urządzeniami o IP: 10.0.1.132 i 10.0.1.133

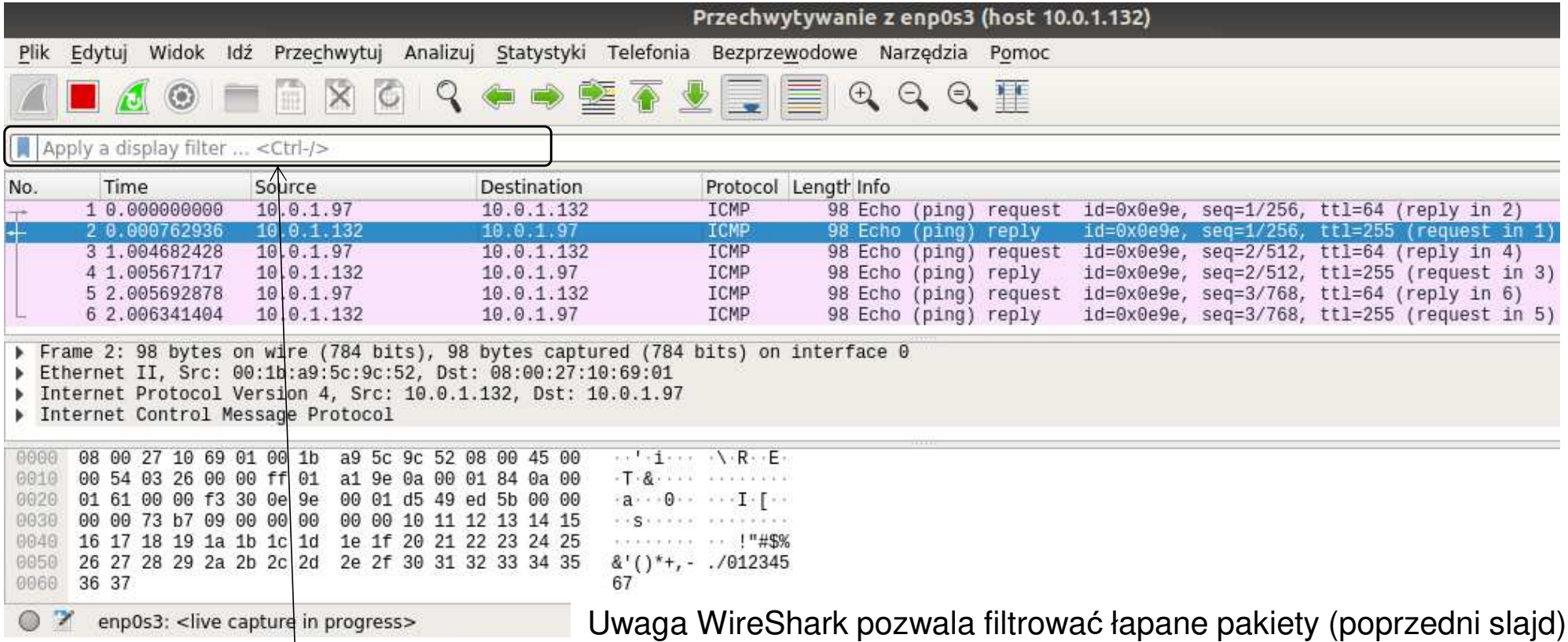
lub

```
tcp port (1883 || 9001)
```

Chcemy obserwować ruch związany z połączeniami na portach 1883 i 9001 protokołu TCP

Testowanie systemu - cześć I

- Badanie ruchu sieciowego, cd.
  - Wireshark - gdy chcemy badać ruch sieciowy możemy użyć narzędzia do łapania tego ruchu
    - Można łapać także bardziej selektywnie – określony protokół (np.: TCP na porcie 1883)



W tym polu wprowadzamy reguły filtrowania podczas wyświetlania

Uwaga WireShark pozwala filtrować łapane pakiety (poprzedni slajd) jak i filtrować wyświetlane pakiety!
Dla efektywnej archiwizacji złapanych pakietów warto dobrze dobrać pierwszy z tych filtrów – drugi jest przydatny do analizy ruchu
Po zarchiwizowaniu złapanych pakietów można je dalej analizować!

- Badanie ruchu sieciowego, cd.
  - Gdzie łapać pakiety?
    - Na maszynie gdzie uruchomiony jest broker
      - Czasami nie mamy dostępu do konsoli takiej maszyny (np.: test.mosquitto.org)
    - Na maszynie klienta
      - Tu też możemy mieć problem z uruchomieniem wcześniej opisanych narzędzi (np.: gdy testujemy w taki sposób aplikacje Android na telefonie)
    - Na tzw. routerze brzegowym
      - Tu z reguły jest możliwość „łapania” ruchu między określonymi interfejsami i możliwości te może ograniczać dostępna przestrzeń dysków lokalnych tego urządzenia
    - Na przełączniku sieciowym (switch)
      - O ile przełącznik ten jest zarządzany to dostarcza możliwość zestawienia określonego portów tak aby był lustrem innego portu – wtedy ruch z określonego portu przekazywany jest też na inny wybrany przez nas port
  - Uwaga! powyższe metody łapania pakietów są mało skuteczne gdy tworzymy aplikacje korzystające z X.509 (TLS, MQTTS)
    - W tym połączeniu ruch jest szyfrowany, zobaczymy zatem tylko fazę zestawiania połączenia szyfrowanego a potem pakiety z danymi których treść będzie nie czytelna

Testowanie systemu - cześć I

Zadanie:
Część I) Na maszynach wirtualnych z brokerem skonfiguruj broker MQTT i złap jak najoszczędniej ruch klientów Anroidowych wymieniających się wiadomościami z Brokerem (maszyna wirtualna)
Część II) Złap ruch MQTT między aplikacją JavaScript korzystającą z WebSocket a jednym z publicznych brokerów
Zadbaj aby złapane pliki zawierały wszystko co ważne i nic zbędnego – czyli by pliki z pakietami miały jak najmniejszą wielkość a zarazem by w ich treści zawarty był cały ruch sieciowy między klientem a brokerem

Testowanie systemu - cześć I

- GDB – uruchamianie aplikacji tworzonej w C
  - Kompilacja

```
gcc -g main.c -o obraz_aplikacji
```

  - Uruchomienie aplikacji

```
gdb ./obraz_aplikacji
```

  - Zobaczymy

```
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
...
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./obraz_aplikacji...done.
```

- GDB – uruchamianie aplikacji tworzonej w C
  - Używanie narzędzia
    - Listing kodu – wprowadzamy „l” lub „l 90” – gdy interesują nas okolice linii 90

```
85 unsigned char my_str[MAX_BUF];
86 unsigned char my_str2[MAX_BUF];
87 struct sockaddr_in cliAddr;
88 int cliLen=sizeof(cliAddr);
90 int pos=recvfrom(s, my_str, MAX_BUF,0,(struct sockaddr *)&cliAddr, &cliLen);
91 if(pos==0){
92   printf("Peer was disconeted\n");break;
93 }else if(pos<0){
94   printf("ERROR: %s (%s-%d)\n", strerror(errno), __FILE__, __LINE__);
```

    - Praca krokowa
      - s - krok (jedna instrukcja, jedna linia z instrukcjami) w tym także w głąb funkcji
      - n – krok do następnej funkcji na tym poziomie kodu
      - c – kontynuuj wykonywanie kodu do następnej pułapki lub końca działania aplikacji
      - run – uruchomienie programu
    - Ustawianie breakpoints
      - wprowadzamy „b 91” gdy chcemy ustawić pułapkę na linii 91 (motywacja: po wywołaniu `recvfrom()` chcemy zobaczyć co jest zwracane w zmiennej pos)
      - sprawdzenie gdzie pułapki ustawiono: i b

- GDB – uruchamianie aplikacji tworzonej w C
  - Używanie narzędzia, cd.
    - Pułapki na zmiennych, zatrzymaj kod na linii numer 91 gdy zmienna pos będzie miała określoną wartość (gdy ma inną wartość GDB nie zatrzyma się na wskazanej linii)

```
br 91 if pos==10
```

    - A gdy mamy zmienne typu „char *” np.: gdy my_str będzie miała określoną treść

```
br 91 if strcmp(my_str, "tekst")==0
```

Tę pułapkę można ustawić dopiero gdy kod zacznie działać(!) np.: zatrzymał się na innej pułapce

    - Inspekcja stanu zmiennych:
      - p pos – gdy GDB ma wypisać wartość pos (tu treść będzie w trybie DEC)
      - p `my_str` - gdy chcemy wypisać zawartość zmiennej typu „char *” - tu otrzymamy coś takiego:

```
$3 = "test\n\000\230\000"
```

      - `p my_str/x` – jak wyżej ale gdy chcielibyśmy zobaczyć wartości w HEX:

```
$4 = {0x74, 0x65, 0x73, 0x74, 0xa, 0x0, 0x98, 0x0}
```

  - Więcej na temat:
    - https://www.sourceware.org/gdb/

- GDB – uruchamianie aplikacji tworzonej w Python
  - Tu podobnie można użyć GDB poprzez uruchomienie:

```
python –m pdb moja_aplikacja.py
```

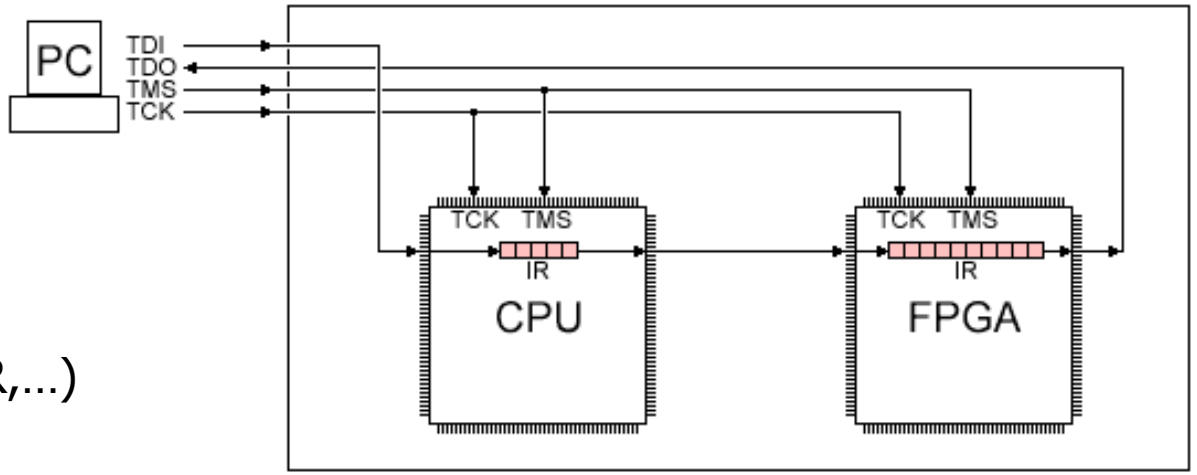
  - Obsługa podobnie jak w GDB dla C/C++ (choć nieco ograniczone)
    - s – krok w tym także w głąb funkcji
    - n – krok do następnej funkcji w tym poziomie kodu
    - c – kontynuuj wykonywanie kodu do następnej pułapki lub końca działania aplikacji
    - l – listuj kod
    - b XX – ustawienie pułapki w linii XX
    - p zmienna – wydruk zmiennej
    - ...
    - oraz warunkowe pułapki

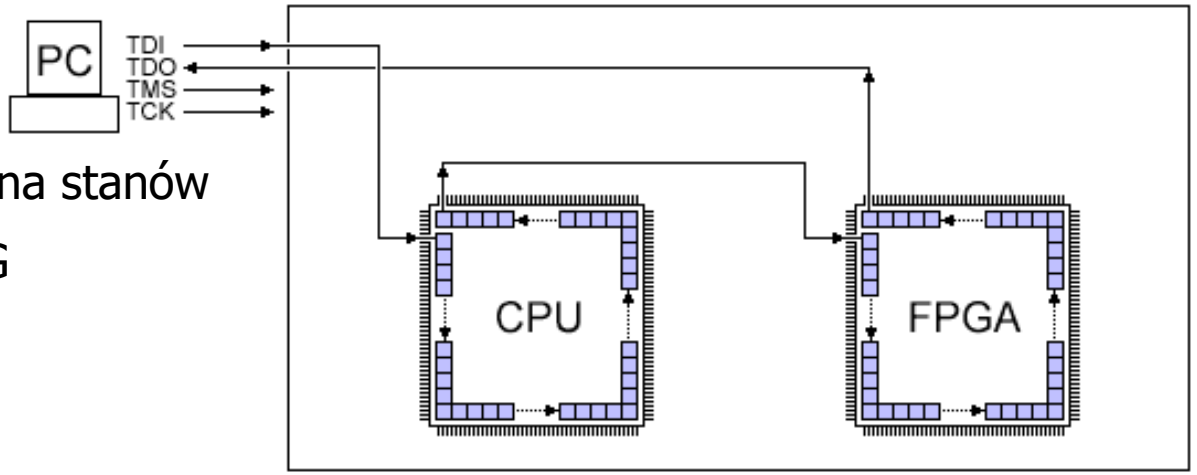
```
b 48                                    (numer linii w kodzie gdzie będzie pułapka)
b                          (system przedstawi listę pułapek – tu interesuje na ta z numerem 1)
condition 1 msg.topic=="test/1"   (teraz otrzymawszy wiadomość w temacie test/1
                                        program zatrzyma się w pułapce numer 1)
```

- Więcej na temat:
  - https://pymotw.com/2/pdb/

Zadanie:
Dla kodu subskrybenta (w C lub Python) wiadomości MQTT współpracującego z publicznym brokerem, ustaw pułapki w kodzie: a)gdy po subskrypcji aplikacja otrzyma wiadomość o treści „catch me”, b)gdy odebrana wiadomość będzie miała temat dłuższy niż 10 znaków, c)gdy wiadomość będzie miała na drugim znaki literę ‚A’.

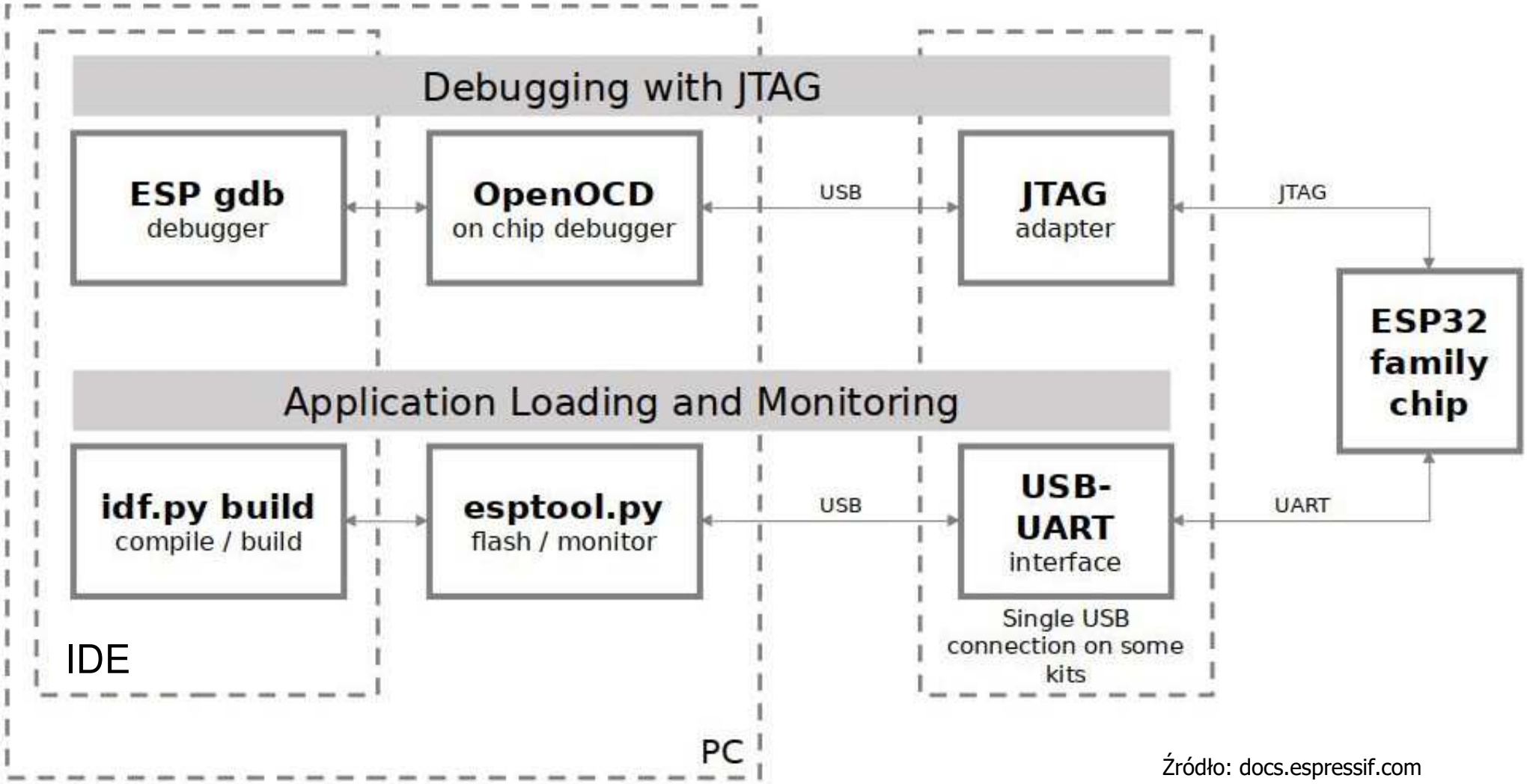
- JTAG – szeregowa specjalizowana magistrala testowania układów
  - Sygnały
    - TCK – sygnał zegary
    - TDI - wejście testowe
    - TDO - wyjście testowe
    - TMS - tryb testowania (DR/IR,…)
    - TRST# - inicjowanie testu
  - Kontroler TAP (Test Access Port)
    - Wbudowany w testowane elementy
    - Implementowany jako maszyna stanów
    - Obowiązkowe instrukcje JTAG
      - BYPASS (np.: dla CPU 11111)
      - EXTEST
      - SAMPLE/PRELOAD
      - IDCODE
        - Opcjonalna, ale zawsze występuje i zwraca 32bitowy identyfikator producenta i typu danej części utrzymywane w JEDEC Standard Manufacturer's Identification Code standard





Źródło: www.fpga4fun.com

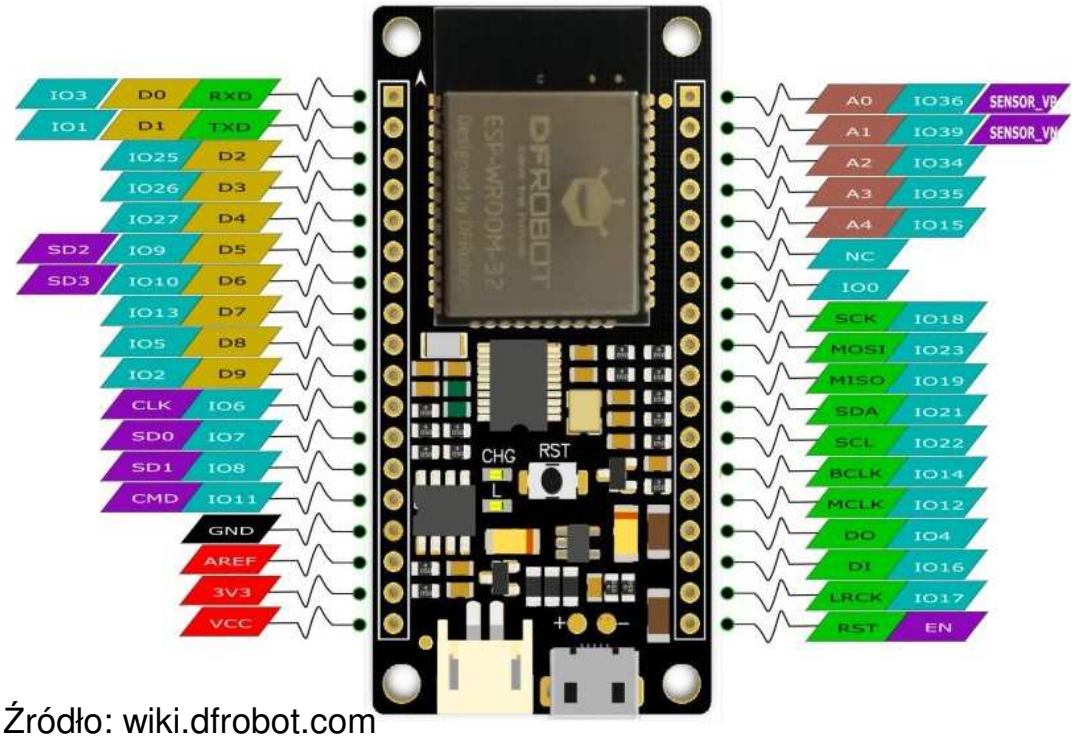
- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - Architektura kompletów
    - UART – komunikaty portem szeregowym i ładowanie programu (typowo), JTAG – ładowanie kodu (opcja) i uruchamianie kodu (stan zmiennych, przebieg działania, …)



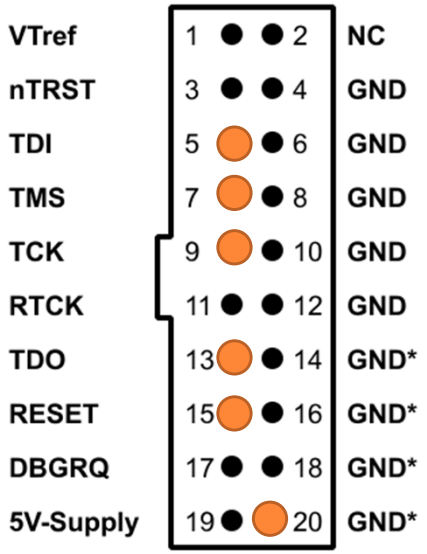
Źródło: docs.espressif.com

- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - Łączenie z ESP32 (FireBeetle ESP32 IOT) z programatorem SEGGER J-Link

ESP32 pint	**Jlink-connector**	**Signal Name**
GND	GND	Ground
IO14	25	TMS
IO12	10	TDI
IO15	9	TDO
IO13	11	TCK
3.3V	1	VTref (opcja)
RST	15	RESET (opcja)



Źródło: wiki.dfrobot.com



Źródło: www.segger.com

Testowanie systemu - cześć I

- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - OpenOCD
    - Narzędzie komunikacji z interfejsem JTAG, wspierające wiele platform (CPU, płytek i interfejsów JTAG-PC)
      - Dla ESP32 wymagana jest specjalizowana wersja OpenESP, binarna wersja do pobrania z:
        - https://github.com/espressif/openocd-esp32/releases
    - Wywołanie wspierane przez polecenia przesyłane protokołem telnet (np.: telnet 127.0.0.1 4444)

```
openocd.exe –f interface/jlink.cfg –f target/esp32.cfg
```

    - lub wywołanie z wbudowanymi poleceniami:

```
openocd.exe –f interface/jlink.cfg –f target/esp32.cfg \
–c "init ; reset halt ; \
program_esp firmware.bin 0x10000 reset verify ; shutdown ; exit"
```

    - Polecenie wykona: inicjacje, zatrzymanie CPU (niezbędne do zmiany firmware), programowanie (program_esp – dedykowane dla ESP), wyłącznie serwera (shutdown) i zakończenie pracy (exit)

Testowanie systemu - cześć I

- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - Platformio Core przypomnienie
    - Instalacja Platformio Core (Linux-Debian)

```
sudo apt-get update ; sudo apt-get install python3 python3-pip
wget https://raw.githubusercontent.com/platformio/platformio-core-installer/master/get-platformio.py
python3 get-platformio.py
```

    - Instalacja Platformio Core (Windows)

```
wget https://www.python.org/ftp/python/3.11.1/python-3.11.1-amd64.exe
python-3.11.1-amd64.exe
wget https://raw.githubusercontent.com/platformio/platformio-core-installer/master/get-platformio.py
python.exe get-platformio.py
```

    - Inicjacja projektu

```
mkdir Blink && cd Blink
pio project init --board firebeetle32
```

Testowanie systemu - cześć I

- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - Uruchamianie kodu z wykorzystaniem Platformio Core (projekt: Blink 10/1)

src/main.cpp:

```
#include "Arduino.h"
#define LED_BUILTIN 2

int state=0;
int my_delay=1000;

void setup(){
  pinMode(2, OUTPUT);
}
void loop() {
  if(state==0){
     digitalWrite(2, HIGH);
     my_delay=1000;
     state=1;
  }else{
     digitalWrite(2, LOW);
     my_delay=100;
     state=0;
  }
  delay(my_delay);
}
```

platformio.ini:

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:firebeetle32]
platform = espressif32
board = firebeetle32
framework = arduino
upload_port = com29  ;lub w Linux np.: /dev/ttyUSB0
```

Testowanie systemu - cześć I

- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - Instalowanie firmware
    - Przed używaniem JTAG w niektórych platformach konieczne jest skasowanie pamięci lub zainstalowanie aplikacji – pod systemem Linux

```
export PATH=/home/pi/.platformio/penv/bin:$PATH
platformio run -e firebeetle32
platformio run -e firebeetle32 --target upload
```

Przydatne pod systemem Linux, dla systemu Windows konieczne jest zainstalowanie PlatformIO z zezwoleniem na modyfikację PATH

    - Lub proste skasowanie pamięci - pod systemem Windows

```
esptool.exe --chip esp32 -p COM29 erase_flash
```

      - pod systemem Linux (z zainstalowanym ESP-IDF)

```
sudo chmod 777 /dev/ttyUSB0
/home/student/esp/esp-idf/components/esptool --chip esp32 -p /dev/ttyUSB0 erase_flash
```

    - Ładowanie firmware za pomocą JTAG - pod Linux

```
sudo openocd -f interface/jlink.cfg -f target/esp32.cfg \
-c "init ; reset halt ; program_esp firmware.bin 0x10000 reset verify ; shutdown ; exit"
```

Testowanie systemu - cześć I

- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - Uruchamianie kodu z wykorzystaniem Platformio Core (plik .pioinit)

```
define pio_reset_halt_target
            monitor reset halt
            flushregs
end
define pio_reset_run_target
            monitor reset
end
define sy
            watch state
end
define my
            c
            p state
end
target extended-remote | "___tu_samemu_wypelnij___/.platformio/packages/tool-openocd-esp32/bin/openocd.exe"
                                          -s "___tu_samemu_wypelnij___/.platformio/packages/tool-openocd-esp32"
                                                -c "gdb_port pipe; tcl_port disabled; telnet_port disabled"
                        -s "___tu_samemu_wypelnij___/.platformio/packages/tool-openocd-esp32/share/openocd/scripts"
                                             -f "interface/jlink.cfg" -f "board/esp-wroom-32.cfg" -c "adapter_khz 500"
monitor program_esp "{___tu_samemu_wypelnij___/.platformio/packages/framework-arduinoespressif32/tools/sdk/
                                                              esp32/bin/bootloader_dio_40m.bin}" 0x1000 verify
monitor program_esp "{___tu_samemu_wypelnij___/.pio/build/firebeetle32/partitions.bin}" 0x8000 verify
monitor program_esp "{___tu_samemu_wypelnij___/.platformio/packages/framework-arduinoespressif32/tools/
                                                                    partitions/boot_app0.bin}" 0xe000 verify
monitor program_esp "{___tu_samemu_wypelnij___/.pio/build/firebeetle32/firmware.bin}" 0x10000 verify
pio_reset_halt_target
tbreak setup
define pio_restart_target
   pio_reset_halt_target
   tbreak setup
end
```

Ustawienie pułapki na zmianę stanu zmiennej state

Kontynuuj aż do zatrzymania a potem wypisz wartość zmiennej state

- JTAG – szeregowa specjalizowana magistrala testowania układów, cd.
  - Uruchamianie procesu debuger'a (bez konieczności uruchamiania osobno openocd)

```
>platformio run
>platformio debug --interface=gdb  -x .pioinit
GNU gdb (crosstool-NG esp-2021r2-patch3) 9.2.90.20200913-git
...
Reading symbols from .pio/build/firebeetle32/firmware.elf...
Temporary breakpoint 1 at 0x400d0ee7: file src/main.cpp, line 8.
...
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     del  y   0x400d0ee7 in setup() at src/main.cpp:8
(gdb) sy
...
(gdb) my
...
Thread 1 "loopTask" hit Temporary breakpoint 1, setup () at src/main.cpp:8
8           pinMode(2, OUTPUT);
$1 = 0
$2 = 1000
```

Wywołanie funkcji *sy* zdefiniowanej w .pioinit (ustawia watchpoints na zmiennej *state*)

Wywołanie własnej funkcji *my* zdefiniowanej w .pioinit

Polecenie *my* uruchamia kod (c) oraz po zmianie zmiennej *state* wypisuje jej stan

Zadanie:
W kodzie subskrybenta protokołu MQTT napisanego w języku C++ dla ESP32 (bazującego na modelu Arduino) ustaw pułapkę w kodzie klienta która przerwie wykonywanie kodu gdy po subskrypcji aplikacja otrzyma wiadomość o treści „esp 32 caught me”

- Dodatki
  - Kod MQTT klienta dla Platformio

src/main.cpp

```
#include <WiFi.h>
#include <PubSubClient.h>
const char* ssid = "xxxxxx";
const char* password = "xxxxx";
#define MQTT_TOPIC_IN  "PBL5/2022/test"
WiFiClient espClient;
PubSubClient client(espClient);
void setup_wifi() {
  WiFi.begin((char*)ssid, (char*)password);
  while (WiFi.status() != WL_CONNECTED)
    delay(500);
  Serial.println(WiFi.localIP());
}
void reconnect() {
  while (!client.connected()) {
    if (client.connect("test01")){
      client.subscribe(MQTT_TOPIC_IN);
    } else {
      delay(5000);
    }
} }
```

```
void callback(char* t, byte *p, unsigned int l){
  for(int i = 0; i<l; i++)
    Serial.print((char)p[i]);
  Serial.println("");
}
void setup(){
  setup_wifi();
  client.setServer("test.mosquitto.org", 1883);
  client.setCallback(callback);
}
void loop(){
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

platformio.ini

```
[env:firebeetle32]
lib_deps = PubSubClient@2.7
platform = espressif32
board = firebeetle32
framework = arduino
upload_port = com11
```

Dziękuje za uwagę