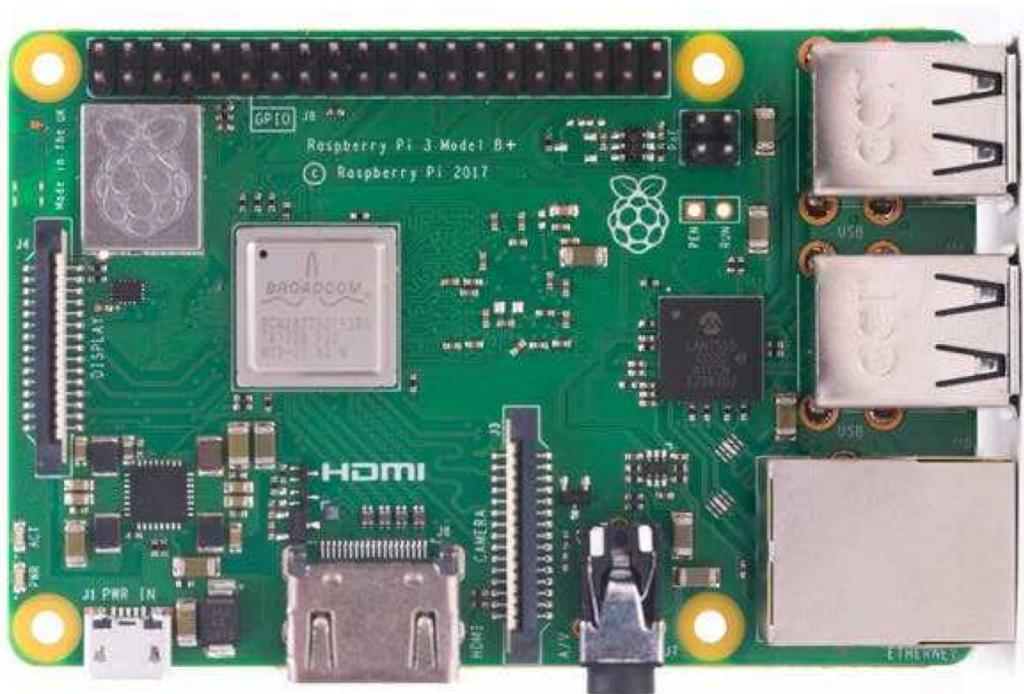


Baremetal i C++

na przykładzie Raspberry Pi i biblioteki Circle

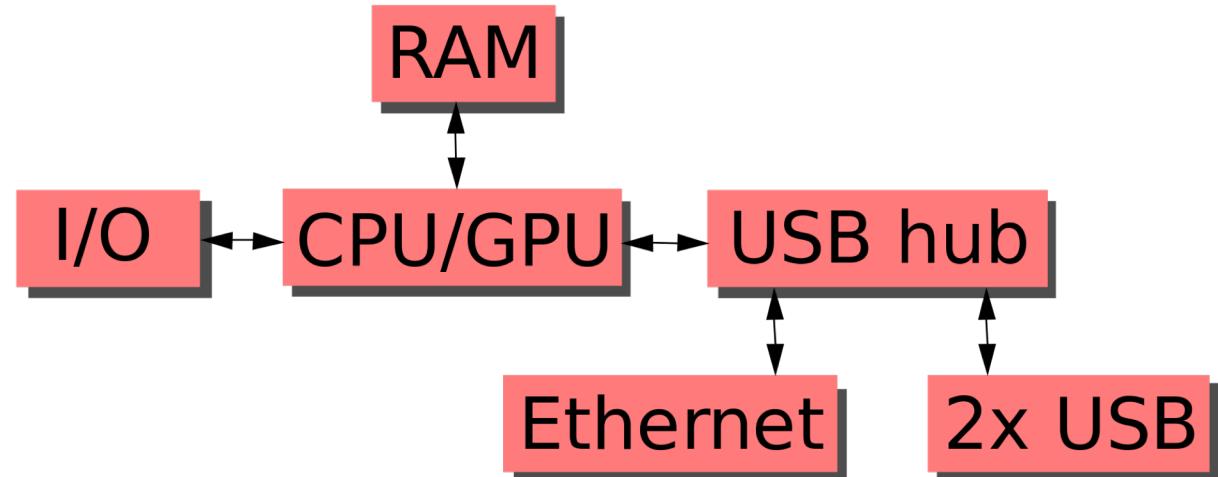
Raspberry Pi (3 B+)



Źródło: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>

Baremetal

- brak systemu operacyjnego,
- bezpośredni dostęp do sprzętu,

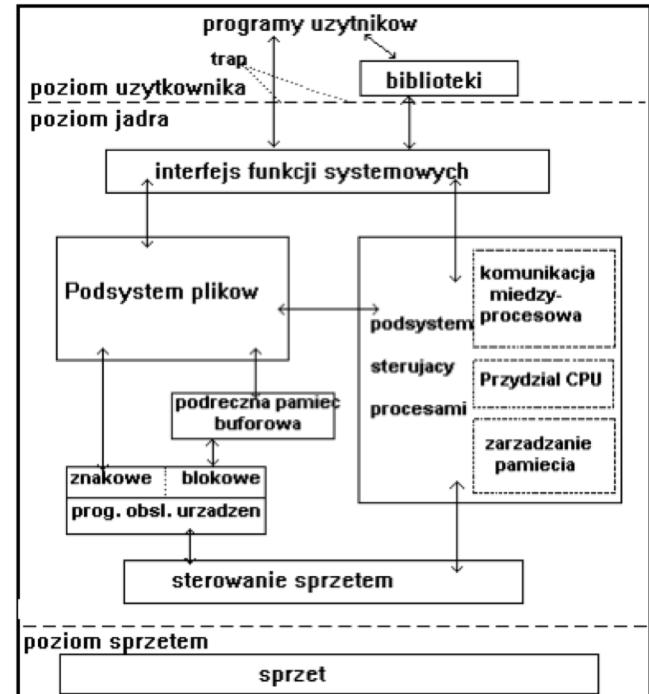


Źródło: https://en.wikipedia.org/wiki/Raspberry_Pi

Co nam daje system operacyjny?

System operacyjny ułatwia zarządzanie sprzętem:

- implementuje sterowniki,
- wystawia wywołania systemowe,
- przejmuje przerwania,
- buforuje dane,
- zarządza pamięcią,
- zarządza procesami.



Źródło: <http://students.mimuw.edu.pl/SO/Linux/Temat01/temat011.html>

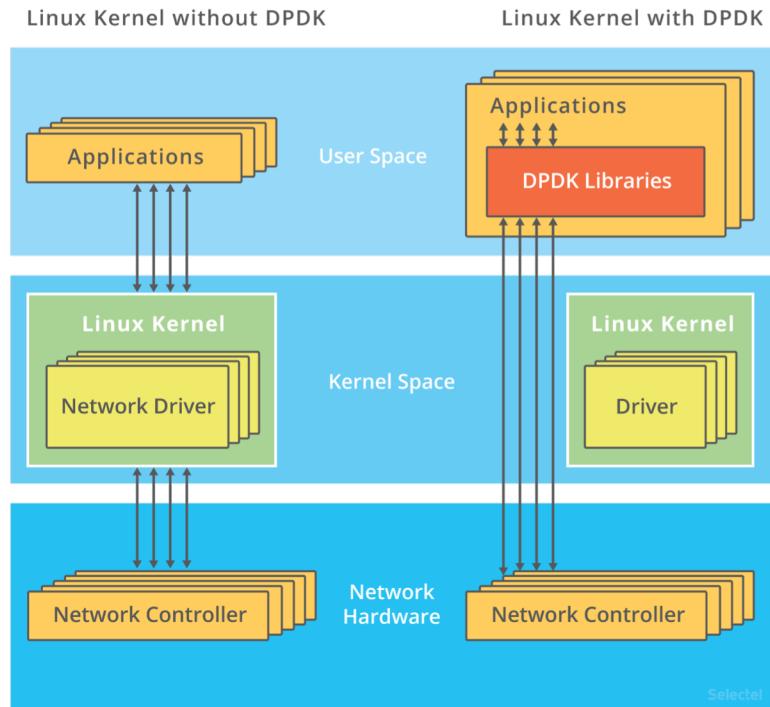
Dlaczego rezygnować z systemu operacyjnego?

- zajmuje zasoby (pamięć, CPU);
- uruchamia niepotrzebne procesy;
- posiada uniwersalne rozwiązania, które w pewnych przypadkach mogą nie być optymalne.

A może jakieś rozwiązanie hybrydowe?

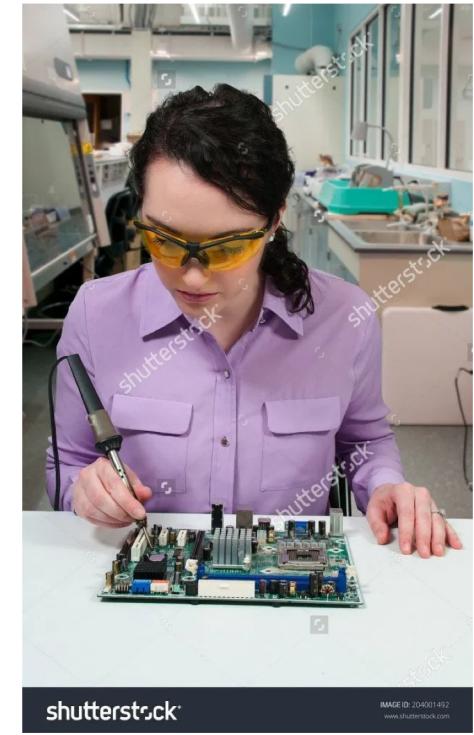
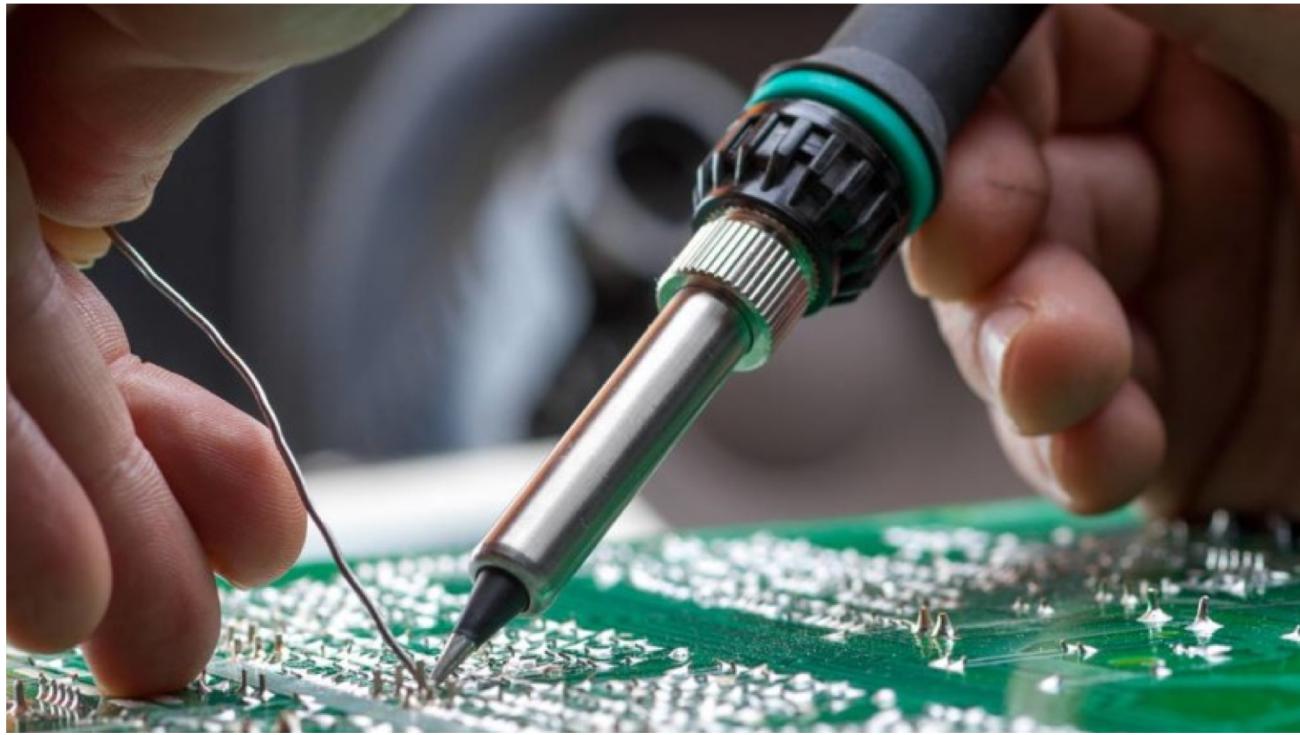
Problem przepustowości sieci?

DPDK



Źródło: <https://blog.selectel.com/introduction-dpdk-architecture-principles/>

Baremetal - brzmi jak coś trudnego...



Źródło: <https://www.stukpuk.pl/blog/lutowanie-przewodu-elektrycznego-o-czym-powinienes-pamietac/>
<https://makezine.com/2016/03/08/beautiful-woman-soldering-stock-photo-wrong/>

shutterstock

IMAGE ID: 204001492
www.shutterstock.com

Dlaczego Raspberry Pi?

- popularne i tanie;
- trudne (niemożliwe?) do zbrickowania;
- słaba dokumentacja, ale mnóstwo nieoficjalnych tutoriali;
- mnogość bibliotek.

Dlaczego Raspberry Pi jest ciężko zbrickować?

Głównym atutem jest tu nietypowy mechanizm bootowania:

- zaczyna GPU;
- ładuje z karty SD (lub MMC w IV) zamknięty firmware;
- konfiguruje rdzeń ARM;
- uruchamia plik jądra, którym może być nasz kod.

Dlaczego Circle?

Obecnie jest to chyba jedyna względnie znana biblioteka, która ma choć częściowo zaimplementowaną bibliotekę standardową.

- obsługa ekranu;
- obsługa sieci (łącznie ze stosem TCP/IP);
- obsługa GPIO;
- obsługa wszystkich 4 rdzeni.

Jak zacząć?

- pobrać bibliotekę:
<https://github.com/rsta2/circle>,
- pobrać kompilator:
<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-a/downloads>
- skonfigurować,
- zbudować.

x86_64 Linux hosted cross compilers

AArch32 bare-metal target (arm-eabi)

- [gcc-arm-8.3-2019.03-x86_64-arm-eabi.tar.xz](#)
- [gcc-arm-8.3-2019.03-x86_64-arm-eabi.tar.xz.asc](#)

AArch64 ELF bare-metal target (aarch64-elf)

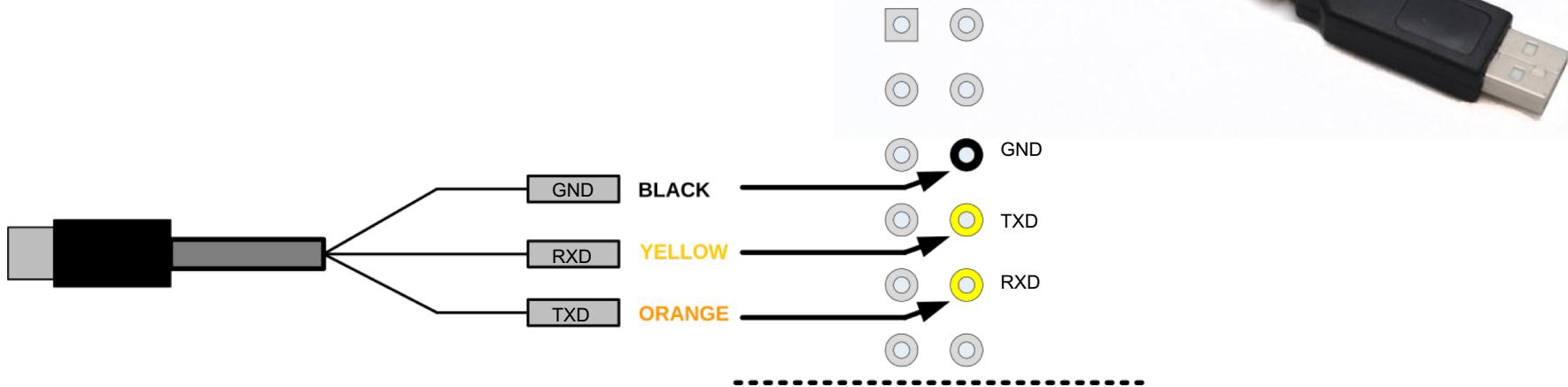
- [gcc-arm-8.3-2019.03-x86_64-aarch64-elf.tar.xz](#)
- [gcc-arm-8.3-2019.03-x86_64-aarch64-elf.tar.xz.asc](#)

GitHub

arm Developer

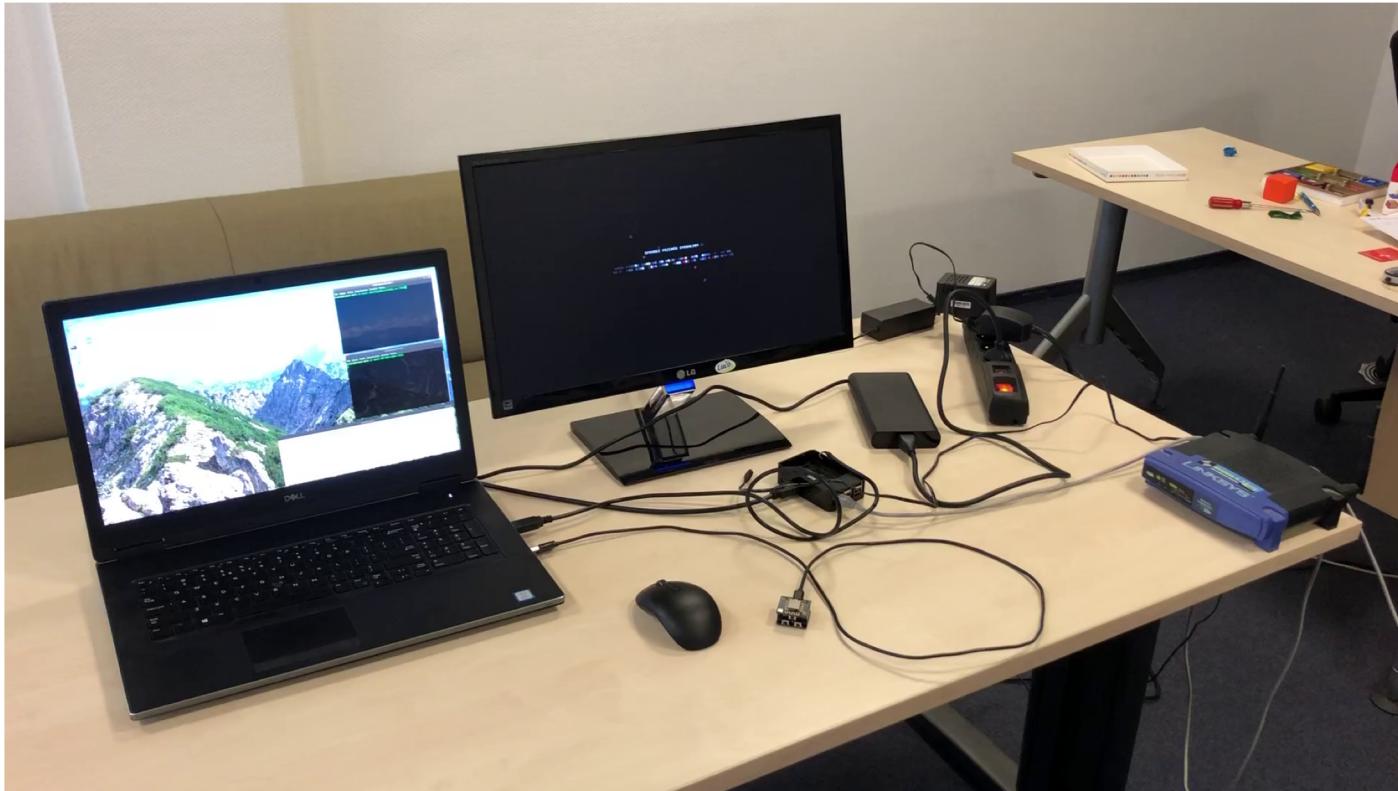
Przydatny sprzęt:

- adapter portu szeregowego na 3.3V,
- monitor obsługujący HDMI,
- opcjonalnie klawiatura.

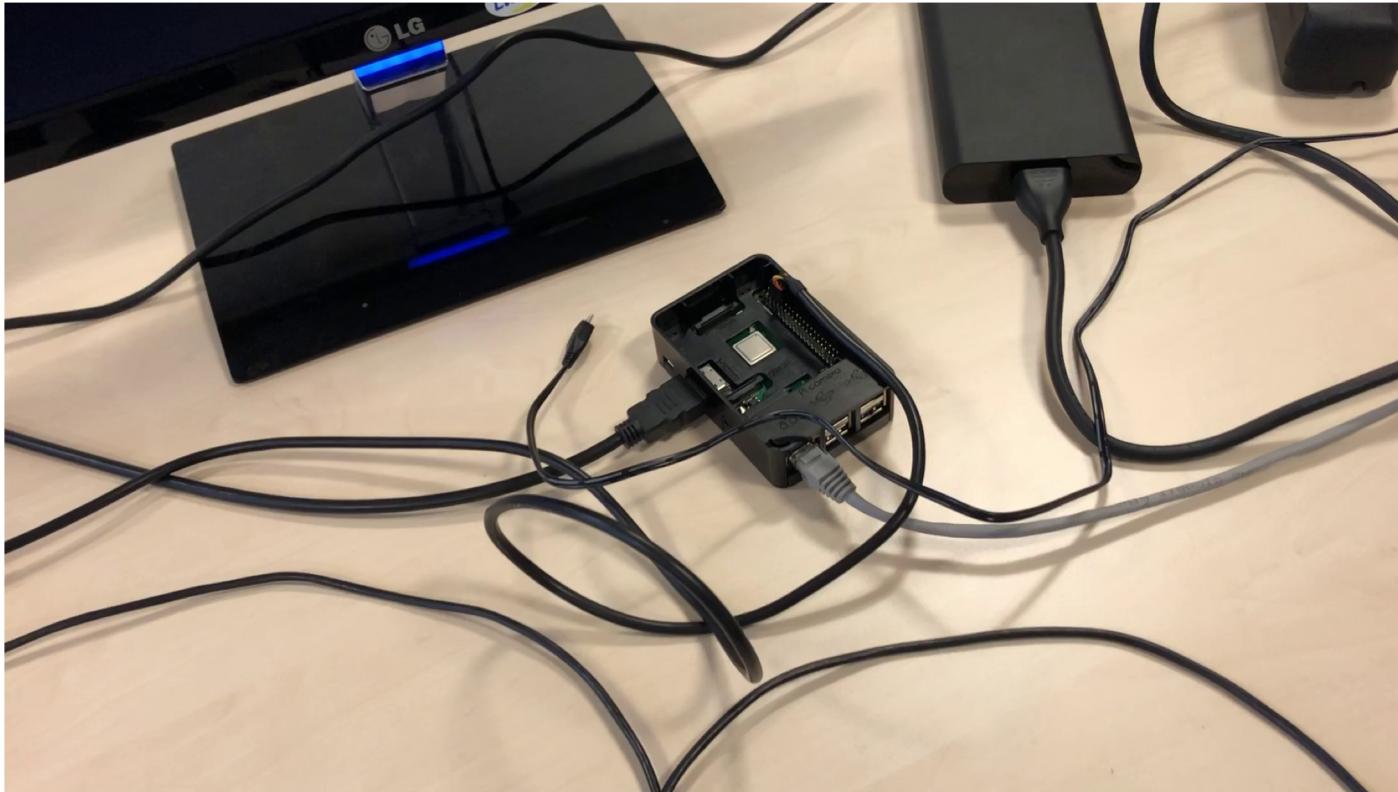


Źródło: https://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_RPi.pdf

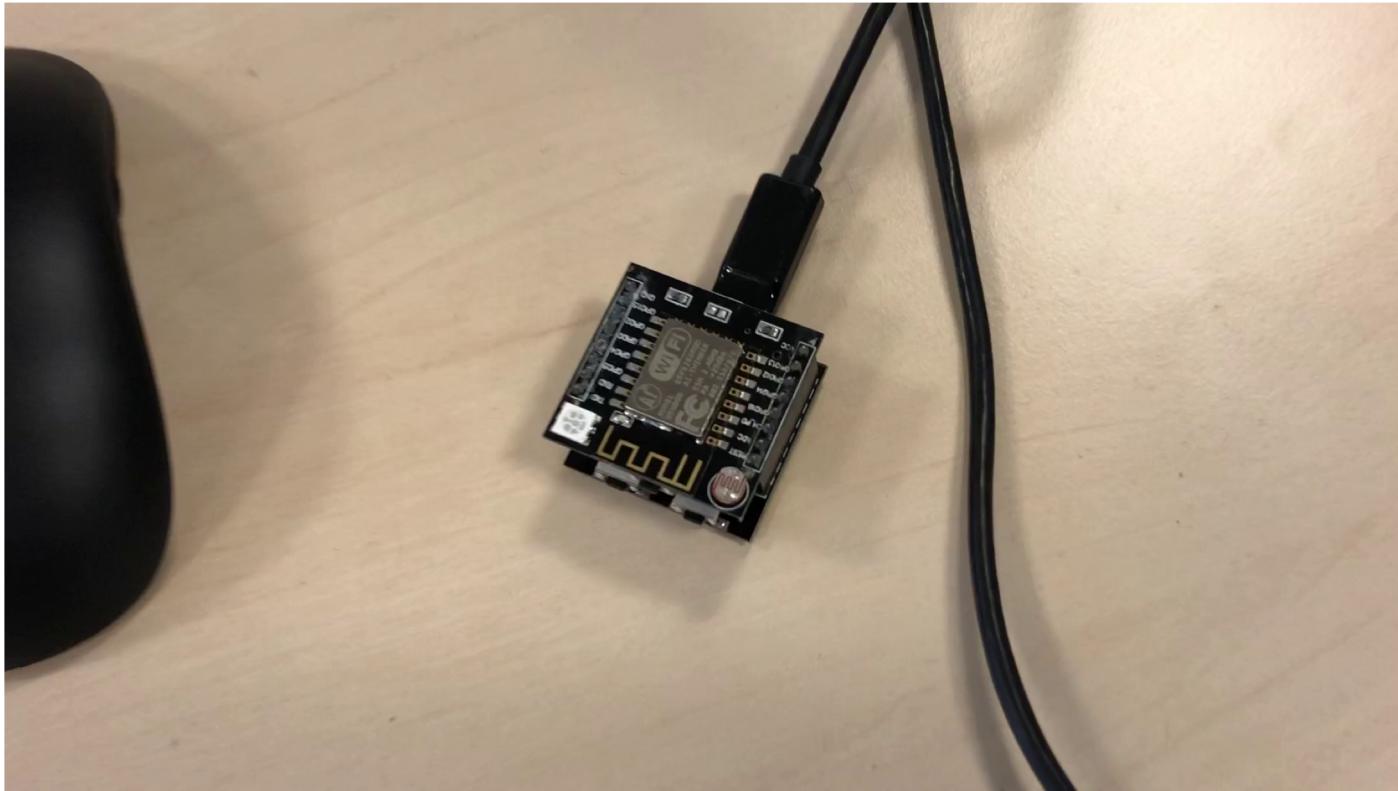
Przykładowy setup:



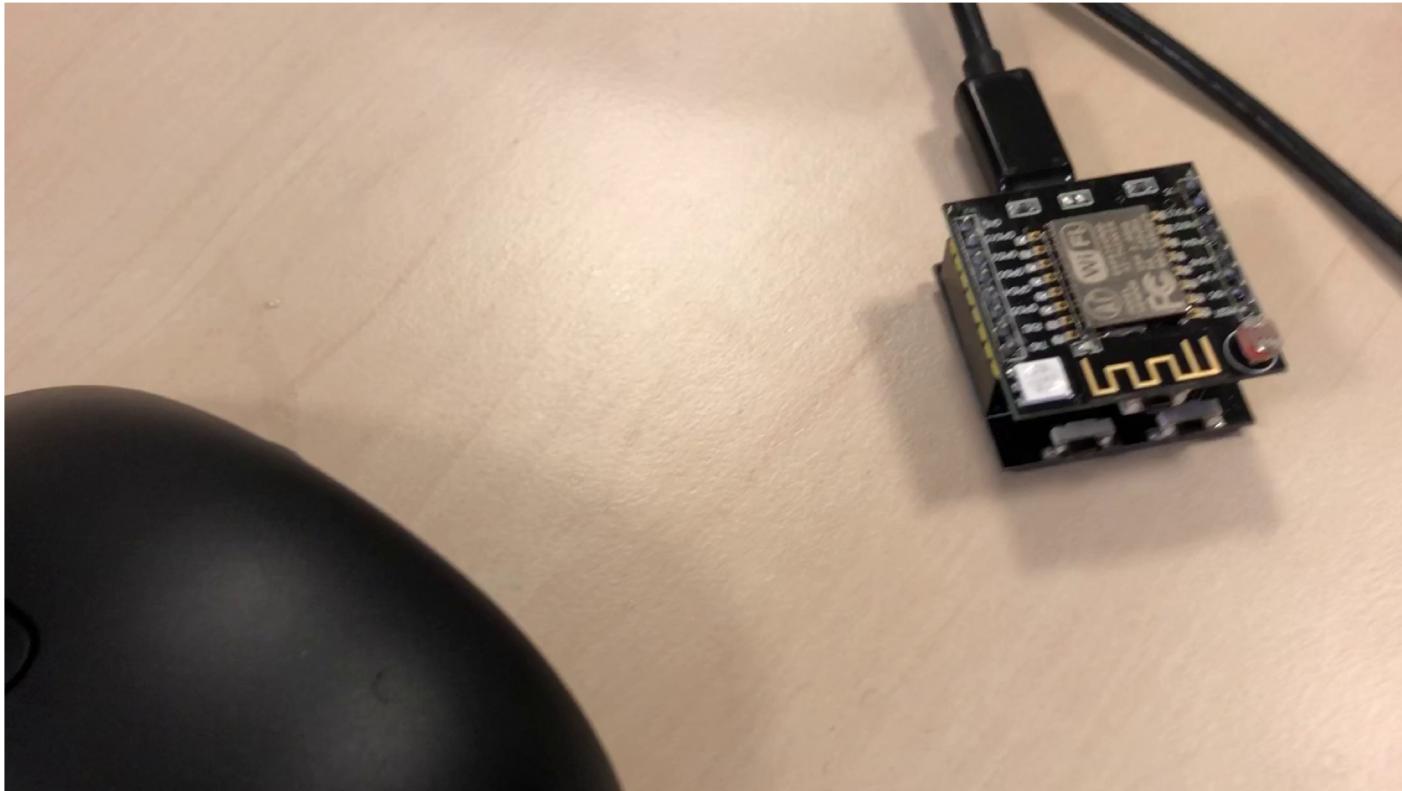
Przykładowy setup:



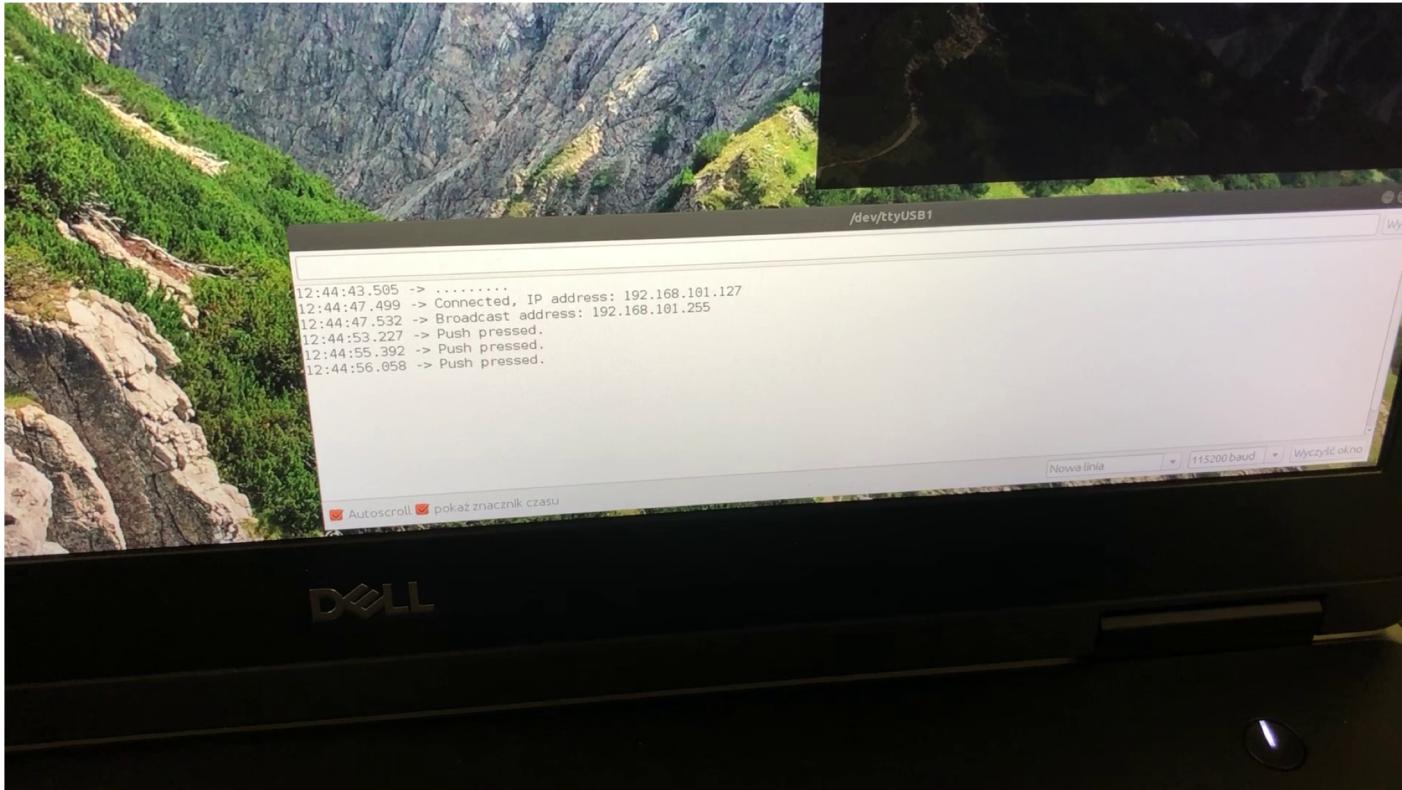
Przykładowy setup:



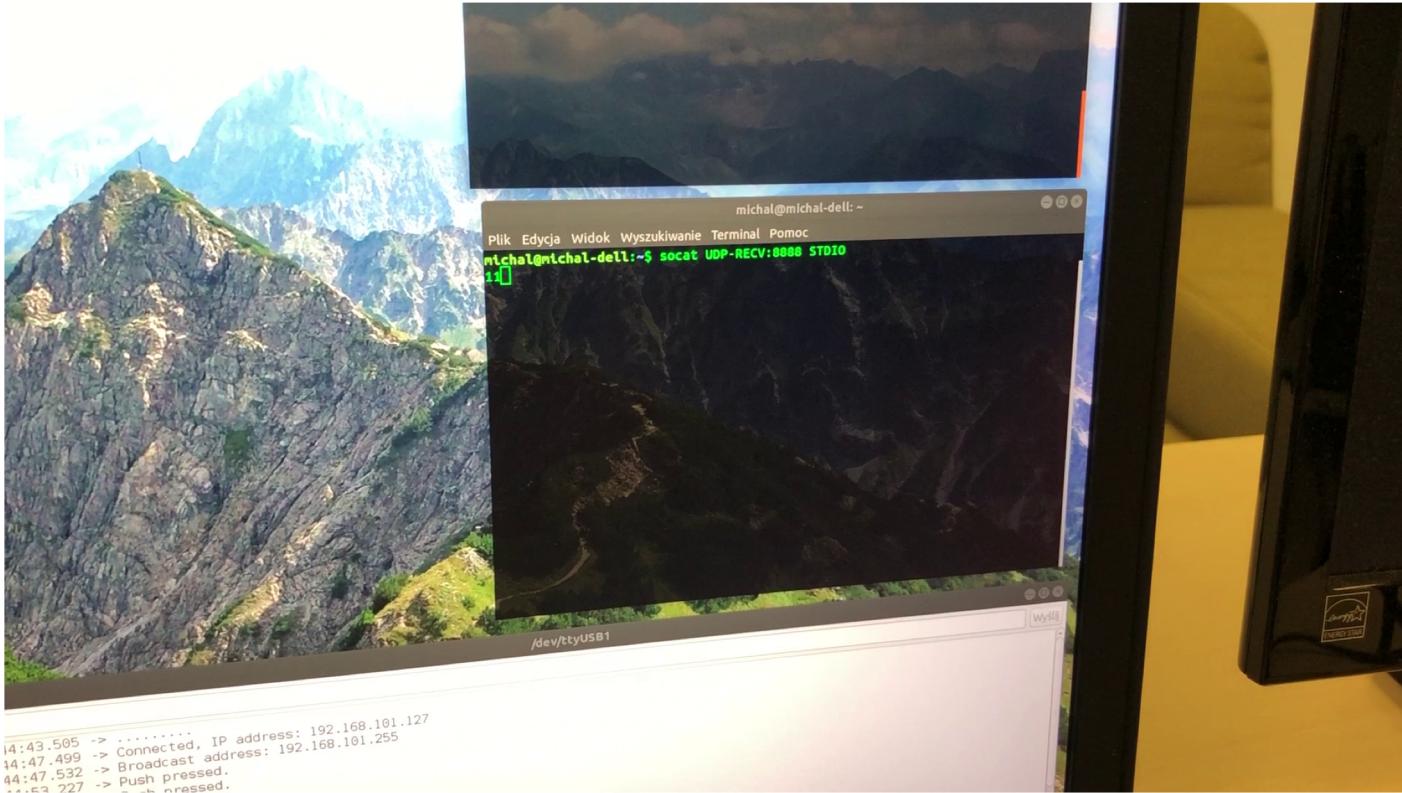
Przykładowy setup:



Przykładowy setup:



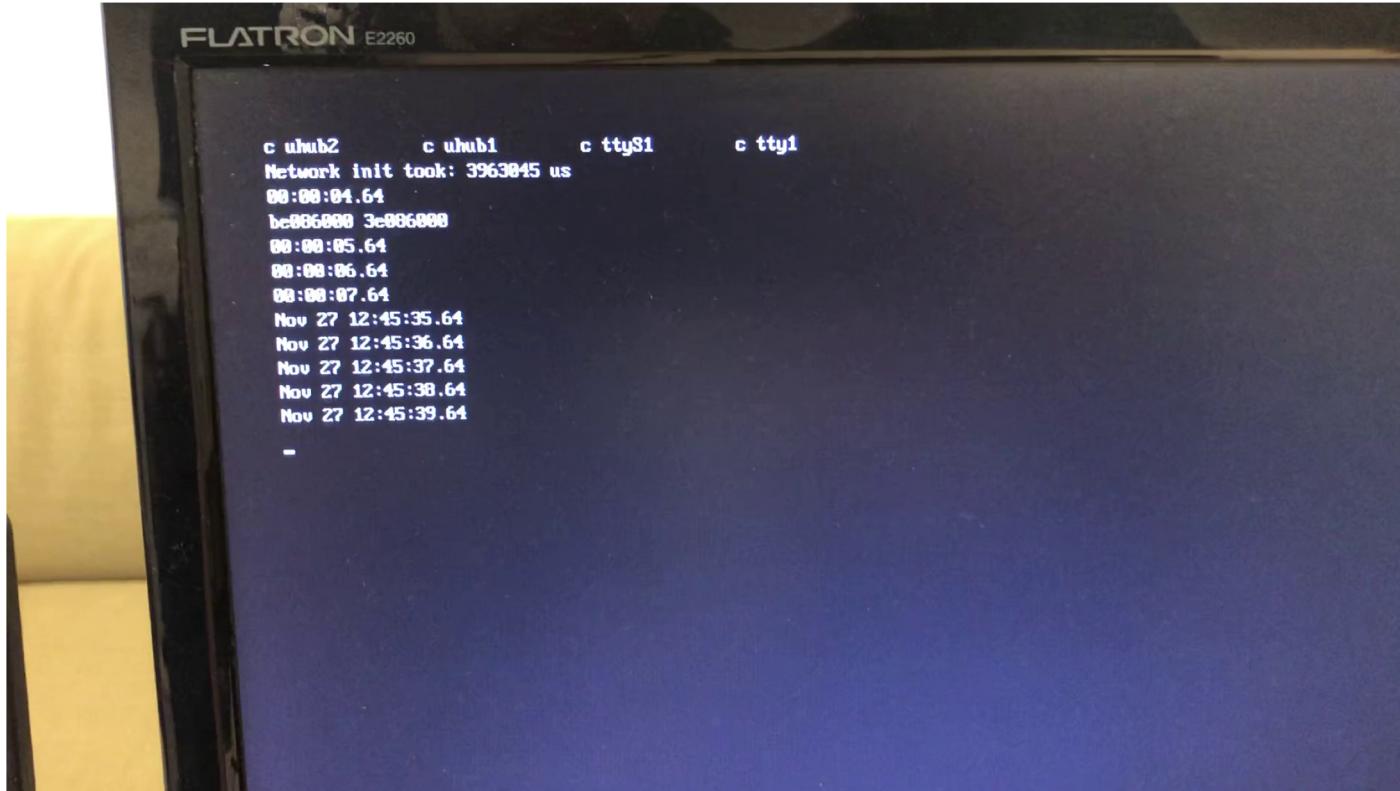
Przykładowy setup:



Przykładowy setup:



Przykładowy setup:



Przykładowy setup:



Konfiguracja

Config.mk:

AARCH ?= 64

RASPI ?= 3

PREFIX ?= arm-eabi-

PREFIX64 ?= aarch64-elf-

Moje modyfikacje:

```
diff --git a/Rules.mk b/Rules.mk
```

```
-CPPFLAGS += -fno-exceptions -fno-rtti -nostdinc++
+CPPFLAGS += -fno-exceptions -fno-rtti #-nostdinc++
```

```
diff --git a/include/circle/sysconfig.h b/include/circle/sysconfig.h
```

```
-/#define ARM_ALLOW_MULTI_CORE
+#define ARM_ALLOW_MULTI_CORE
```

```
-#define DEFAULT_KEYMAP "DE"
+/#define DEFAULT_KEYMAP "DE"
```

```
-/#define DEFAULT_KEYMAP "US"
+#define DEFAULT_KEYMAP "US"
```

Struktura kodu demo

main(): tworzy myKernel i uruchamia w nim metodę Run().

myKernel: pola zinstancjami sterowników:

- obsługa pamięci,
- obsługa przerwań,
- obsługa zegara,
- obsługa listy urządzeń,
- ekran,
- logger,
- scheduler,
- multicore.

myMultiCore: jedna metoda Run(int core)

- uruchamiana równolegle,
- argument informuje o numerze rdzenia,
- klasa przechowuje referencje do sterowników w myKernel.

CScheduler: automatycznie rejestruje obiekty dziedziczące po CTask.

myUDPRRecv: potomek CTask - przetwarza pakiety UDP

CNTPDaemon: potomek CTask - klasa biblioteczna, synchronizuje zegar z serwerem czasu

Główna klasa:

```
class myKernel
{
    public:
        myKernel(void);
        virtual ~myKernel(void) {};

        boolean Initialize(void);
        int Run(void);

    private:
        CMemorySystem mem;
        CActLED led;
        CKernelOptions kopts;
        CDeviceNameService dname;
        CScreenDevice scr;
        CSerialDevice ser;
        CExceptionHandler exh;
        CIInterruptSystem irq;
        CTimer tmr;
        CLogger log;
        CScheduler sched;
        CUSBHCIDevice usb_hci;
        CNetSubSystem net;
        myMultiCore mcore;
        unsigned ticks;
};
```

Główna pętla:

```
int myKernel::Run(void)
{
    CString txt;
    log.WriteLine("Kernel", LogNotice, "Start");
    dname.ListDevices(&scr);
    dname.ListDevices(&ser);
    txt.Format("Network init took: %u us\n", ticks);
    scr.Write(txt, txt.GetLength());
    tmr.SetTimeZone(TimeZone);
    mcore.Run(0);
    return EXIT_HALT;
}
```

```
int main(void)
{
    myKernel k;

    if (!k.Initialize()) {
        return EXIT_HALT;
    }

    return k.Run();
}
```

Inicjalizacja:

```
boolean myKernel::Initialize(void)
{
    if (!scr.Initialize())
        return false;

    if (!ser.Initialize(115200))
        return false;

    if (!log.Initialize(&ser))
        return false;

    if (!irq.Initialize())
        return false;

    if (!tmr.Initialize())
        return false;

    ticks = tmr.GetClockTicks();

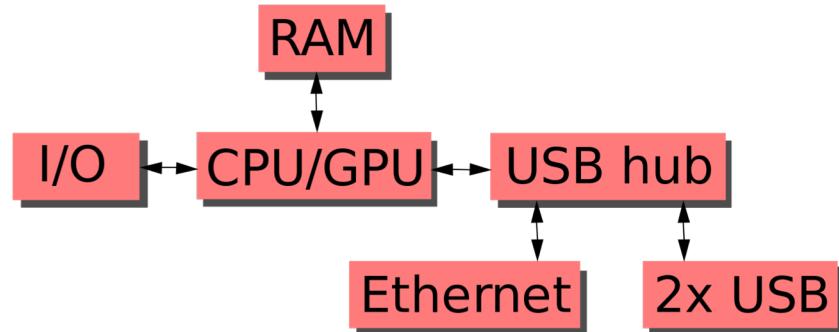
    if (!usb_hci.Initialize())
        return false;

    if (!net.Initialize())
        return false;

    ticks = tmr.GetClockTicks() - ticks;

    if (!mcore.Initialize())
        return false;

    return true;
}
```



Jak to wygląda w środku?

```
.globl _start
_start:           /* normally entered from armstub8 in EL2 after boot */

    mrs x0, CurrentEL      /* check if already in EL1t mode? */
    cmp x0, #4
    beq 1f

    ldr x0, =MEM_EXCEPTION_STACK /* IRQ, FIQ and exception handler run in EL1h */
    msr sp_el1, x0      /* init their stack */

    ldr x0, =VectorTable  /* init exception vector table for EL2 */
    msr vbar_el2, x0

    armv8_switch_to_el1_m x0, x1

1:   ldr x0, =MEM_KERNEL_STACK /* main thread runs in EL1t and uses sp_el0 */
    mov sp, x0      /* init its stack */

    ldr x0, =VectorTable  /* init exception vector table */
    msr vbar_el1, x0

    b  sysinit
```

Multicore - inicializacja:

```
#ifdef ARM_ALLOW_MULTI_CORE

.globl _start_secondary
_start_secondary:    /* normally entered from armstub8 in EL2 after boot */

/* (...) */

b sysinit_secondary

#endif
```

Scheduler!

```
void myMultiCore::EthComm(void)
{
    new CNTPDaemon(NTPServer, &net);
    new myUDPRRecv(net, UdpPort);

    while (1) {
        CString t = *CTimer::Get()->GetTimeString();
        t.Append("\n");
        scr.Write(t, t.GetLength());
        sched.Sleep(1);
    }
}
```

```
void myUDPRRecv::Run(void)
{
    char buf[FRAME_BUFFER_SIZE+1];

    if (sock.Bind(port) < 0) {
        CLogger::Get()->Write("network", LogError, "Cannot bind to port %u", port);
        return;
    }

    sock.SetOptionBroadcast(true);

    while(1) {
        int n = sock.Receive(buf, FRAME_BUFFER_SIZE, 0);

        if (n < 0) {
            CLogger::Get()->Write("network", LogError, "Socket error %d", n);
        }
        else if (n > 0) {
            buf[n] = 0;
            CLogger::Get()->Write("network", LogNotice, "Socket data: %s", buf);
        }
    }

    CScheduler::Get()->Yield();
}
```

Multicore

```
void myMultiCore::Run(unsigned core) {  
    if (core == 0)  
        boot = false;  
    else  
        while(boot);  
  
    switch(core) {  
        case 0:  
            break;  
        case 1:  
            GIComm();  
            break;  
        case 2:  
            EthComm();  
            break;  
        case 3:  
            break;  
        default:  
            break;  
    }  
    running &= ~(1 << core);  
    if (core == 0)  
        while(running);  
}
```

OpenGL - betonowe koło ratunkowe

```
volatile __attribute__((aligned(4))) uint32_t* GL::V3D = reinterpret_cast<uint32_t*>(static_cast<uintptr_t>(ARM_IO_BASE + 0xc00000));  
  
bool GL::Init(void)  
{  
    const uint32_t CLOCK_ID_V3D = 4;  
    const uint32_t TAG_ENABLE_QPU = 0x00030012;  
    uint32_t tags[9] = { PROPTAG_GET_CLOCK_RATE, 8, 8, CLOCK_ID_V3D, 250000000, TAG_ENABLE_QPU, 4, 4, 1 };  
  
    if (mbx.GetTags(tags, sizeof(tags)) == false)  
        return false;  
  
    if (GL::V3D[V3DReg::IDENT0] != 0x02443356) // magic number  
        return false;  
  
    return true;  
}
```

Pytania?

Dziękuję bardzo