Użytkowanie Odroid-C1+ (1)

Środowisko programistyczne, instalowanie systemu Linux

Raspberry Pi wywołał niemałą rewolucję w świecie komputerów jednopłytkowych. Jego sukces spowodował, że zaczęto opracowywać liczne projekty zbliżone do oryginalnego komputera Raspberry, a nawet często przewyższające go wydajnością lub oferowane po niższych cenach. Jako przykłady można tu wymienić: Orange Pi, Banana Pi, lub Odroid. Właśnie temu ostatniemu, a dokładniej modelowi Odroid-C1+ będą poświęcone ten i kolejne artykuły.

Zaczniemy od przygotowania środowiska, uruchomienia Odroida i nawiązywanie połączeń z użyciem kilku dostępnych protokołów. Na koniec uruchomimy kilka praktycznych przykładów, wykorzystując przy tym gotowe sterowniki dostarczane przez jądro. A zatem, zaczynamy.

Środowisko pracy

Wybór środowiska jest dosyć istotną kwestią, ponieważ od tego w dużej mierze zależy komfort pracy. Mimo że w każdej chwili można je zmienić, wybór warto przemyśleć, aby nie przechodzić ponownie przez etapy instalowania i konfigurowania.

Środowisko programistyczne dla systemów wbudowanych składa się z dwóch części. Pierwszą z nich jest stacja robocza, na której powstaje kod i odbywa się jego kompilacja – tzw. host. Drugą stanowi maszyna docelowa, którą w tym wypadku będzie Odroid. Oczywiście, wszystko mogłoby się odbywać na maszynie docelowej, jednak wymagałoby to np. o wiele więcej czasu na skompilowanie programu oraz konieczność zainstalowania narzędzi na urządzeniu często pozbawionym interfejsu graficznego. Z uwagi na to, że naszym systemem docelowym jest Linux, host również powinien być w niego wyposażony. Wybór dystrybucji jest sprawą indywidualną. Mój wybór padł na Ubuntu 14.04 ze względu na jego „dojrzałość" i ogromną społeczność, dzięki której z łatwością można znaleźć odpowiedź na większość pojawiających się problemów. Dodatkową zaletą jest fakt, że na komputerze Odroid używam tego samego systemu. Dzięki takiemu podejściu wiedzę i umiejętności zdobyte podczas pracy z hostem można często zastosować w docelowym systemie. Na temat instalacji Ubuntu dostępnych jest wiele materiałów w sieci, zarówno w języku angielskim, jak i polskim, niezależnie od tego czy będzie to jedyny system operacyjny na komputerze, jeden z kilku, czy też instalacja odbędzie się na maszynie wirtualnej.

W wypadku Odroida, oficjalnie wspierane są dwa systemy: Ubuntu oraz Android, jednak dostępne są również obrazy innych systemów przygotowywanych przez społeczność. Pełną listę można znaleźć na stronie Wiki projektu: ***https://goo.gl/Bq5wPd***. Wszystkie przykłady będą uruchamiane na Ubuntu 14.04, którego obrazy dostępne są pod adresem ***https://goo.gl/6OlocD***.

Pierwsze uruchomienie

Po wybraniu i instalacji środowiska dla hosta możemy przystąpić do uruchamiania naszego ODROIDa. Na początek należy pobrać i rozpakować (potrzebny jest program *unxz*) obraz systemu z oficjalnej strony projektu:

```
wget http://de.eu.odroid.in/ubuntu_14.04lts/
ubuntu-14.04.3lts-lubuntu-odroid-c1-20151020.
img.xz
unxz ubuntu-14.04.3lts-lubuntu-
odroid-c1-20151020.img.xz
```

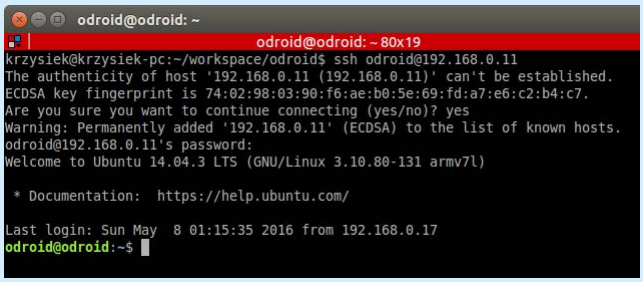
Po rozpakowaniu archiwum otrzymujemy pojedynczy plik o rozszerzeniu img zajmujący ok. 5,2 GB. Jest to obraz systemu, który musimy skopiować bezpośrednio na kartę pamięci MicroSD o dostatecznie dużej pojemności. Najlepiej zrobić to umieszczając kartę bezpośrednio w czytniku, bez użycia adaptera MicroSD – SD, ponieważ zgodnie z informacją na stronie projektu, może to spowodować błędy podczas kopiowania.

Po włożeniu karty do czytnika hosta, musimy znaleźć nadaną jej nazwę urządzenia w postaci */dev/sdx*, gdzie *x* trzeba zastąpić właściwym oznaczeniem. Nazwę można znaleźć np. poleceniem lsblk wyświetlajacym informacje o wszystkich podłączonych dyskach. Po znalezieniu nazwy karty MicroSD wywołujemy polecenia:

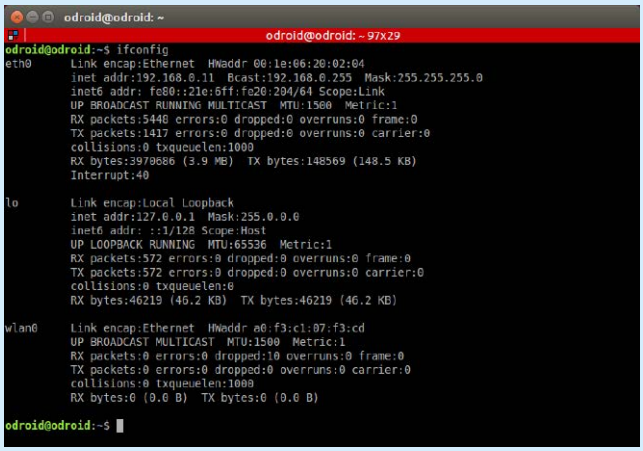
```
sudo dd if=ubuntu-14.04.3lts-lubuntu-
odroid-c1-20151020.img of=/dev/sdx bs=1M
sudo sync
```

„if" jest nazwą obrazu instalowanego systemu, *„of"* jest nazwą urządzenia, natomiast *„bs"* definiuje rozmiar bufora kopiowania. Przy wywoływaniu tego polecenia należy zwrócić szczególną uwagę na nazwę urządzenia */dev/sdx*, ponieważ podając niewłaściwą nazwę można spowodować nadpisanie systemu plików hosta. Drugie z poleceń wymusza opróżnienie wszystkich buforów systemowych, w których mogą znajdować się dane nieskopiowane jeszcze na kartę. Więcej informacji na temat przygotowania karty pamięci można znaleźć na stronie ***https://goo.gl/rnPGNp***.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **66838**, pass: **643ttoo**



Rysunek 1. Połączenie hosta z Odroidem przez ssh



Rysunek 2. Lista dostępnych interfejsów sieciowych

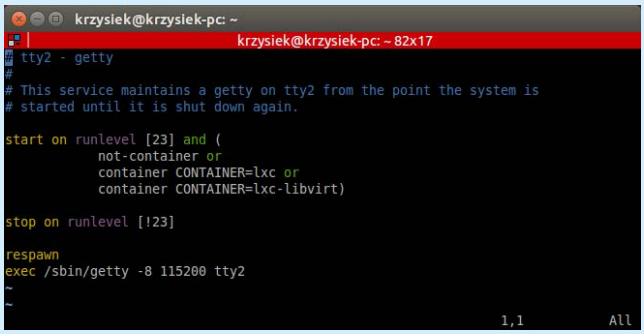
Po przygotowaniu karty można ją umieścić w czytniku Odroida i włączyć zasilanie. System powinien uruchomić się automatycznie, o czym można się przekonać podłączając monitor do gniazda HDMI urządzenia.

Połączenie z hostem

Najprostszym sposobem na rozpoczęcie pracy jest bezpośrednie podłączenie monitora, klawiatury i myszy do Odroida. Nie jest to jednak najlepsze rozwiązanie, gdy środowisko programistyczne znajduje się na hoście. W takiej sytuacji najwygodniej uzyskać połączenie z docelowym urządzeniem za pomocą sieci lokalnej (przez ssh), lub kabla szeregowego.

Połączenie ssh (**rysunek 1**) wymaga przyłączenia do sieci za pomocą kabla ethernetowego lub modułu Wi-Fi. Zainstalowana dystrybucja Ubuntu ma domyślnie włączony serwer ssh, więc po pierwszym uruchomieniu można od razu nawiązać z nim połączenie za pomocą polecenia *ssh odroid@192.168.0.11*. Adres 192.168.0.11 należy zamienić na adres nadany przez router. Domyślne hasło: *odroid*.

Nawiązanie połączenia za pośrednictwem sieci bezprzewodowej wymaga podłączenia zewnętrznej karty sieciowej. W przykładzie użyto karty USB TP-LINK TL-WN725N v2.0. Od razu po przyłączeniu jest ona wykrywana przez system operacyjny, o czym można się przekonać wywołując polecenie *ifconfig* wyświetlające listę dostępnych interfejsów sieciowych (**rysunek 2**). Pozostaje już tylko skonfigurować połączenie poleceniem `sudo nmcli d wifi connect <SSID> password <PASSWORD> iface wlan0`. W miejscach <SSID> i <PASSWORD> należy wpisać nazwę i hasło sieci Wi-Fi. Powyższe polecenia można wywołać po zalogowaniu się przez ssh, po wcześniejszym przyłączeniu kabla sieciowego. W przypadku braku takiej możliwości trzeba



Rysunek 3. Konfiguracja połączenia szeregowego w pliku */etc/init/tty2.conf*

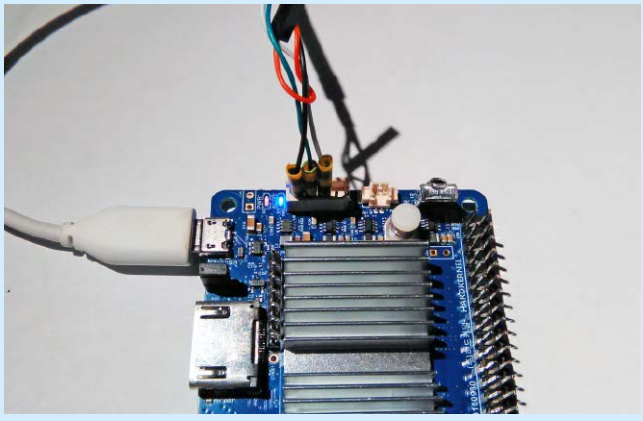
Tabela 1. Połączenia kabla szeregowego Adafruit z gniazdem CON5 Odroida

Adafruit UART-USB	Odroid CON5
+5 V (czerwony)	Niepołączony
RXD (biały)	TXD (2)
TXD (zielony)	RXD (3)
GND (czarny)	GND (4)

skorzystać z klawiatury i monitora, lub opisanego poniżej kabla szeregowego.

Połączenie przez kabel szeregowy wymaga odpowiedniej konfiguracji terminala. W tym celu należy zmodyfikować plik */etc/init/tty2.conf*. Można to zrobić logując się przez ssh lub podłączając kartę pamięci do hosta. W ostatniej linii pliku należy podać wartości odpowiadające połączeniu szeregowemu. Dla użytego w przykładzie konwertera UART-USB Adafruit (***https://goo.gl/0YMCwf***) jest wymagana prędkość 115200 oraz 8 bitów danych (**rysunek 3**). Pełną listę opcji konfiguracji można uzyskać wywołując polecenie `getty --help`. Kabel szeregowy należy przyłączyć do gniazda CON5 ODROIDa zgodnie z **tabelą 1** i **fotografią 4**.

Po dołączeniu kabla do portu USB hosta można skorzystać z dowolnego programu umożliwiającego komunikację szeregową. Jako przykład posłuży program *screen* uruchamiany poleceniem `sudo screen /dev/ttyUSB0 115200`. Logowanie powinno nastąpić automatycznie po włączeniu zasilania. Dodatkową zaletą tego sposobu połączenia jest możliwość obserwacji komunikatów w czasie rozruchu, co pozwala na diagnozowanie ewentualnych błędów, przez



Fotografia 4. Podłączenie kabla szeregowego Adafruit do gniazda CON5 Odroida

które niemożliwy jest poprawny start systemu, a co za tym idzie start serwera ssh.

Hello World

W pierwszym przykładzie napiszemy, skompilujemy, a następnie uruchomimy w systemie docelowym program w języku C. Pozwoli to zapoznać się ze sposobem kompilowania aplikacji w środowisku hosta.

Na początku napiszmy najprostszy program w C. W większości kursów programowania jest to program wyświetlający w konsoli „Hello World". Nie inaczej będzie w naszym przypadku. Kod programu pokazano na **listingu 1**.

Do skompilowania programu potrzebny jest odpowiedni kompilator skrośny (cross-compiler) uruchamiany w systemie hosta, a generujący kod na docelową architekturę. Kompilator taki, wraz z zestawem potrzebnych narzędzi (m. in. linker, debugger), nazywany jest toolchainem. Należy go pobrać z oficjalnej strony projektu i rozpakować poleceniami:

```
wget http://dn.odroid.com/
toolchains/gcc-linaro-arm-linux-
gnueabihf-4.7-2013.04-20130415_linux.tar.bz2
tar -xjf gcc-linaro-arm-linux-
gnueabihf-4.7-2013.04-20130415_linux.tar.bz2
```

Aby można było uruchomić kompilator bez podawania pełnej ścieżki, warto wyeksportować ją do zmiennej środowiskowej PATH zmieniając ścieżkę na właściwą dla ściągniętego i rozpakowanego archiwum:

```
export PATH="$PATH:/home/krzysiek/
workspace/odroid/gcc-linaro-arm-linux-
gnueabihf-4.7-2013.04-20130415_linux/
bin"
```

Aby nie trzeba było eksportować ścieżki po każdym uruchomieniu terminala, warto dodać powyższe polecenie na koniec pliku .bashrc znajdującego się w katalogu domowym użytkownika. Jeżeli zmienna PATH została poprawnie ustawiona można skompilować kod poleceniem *arm-linux-gnueabihf-gcc -o hello hello.c*. Opcja „-o" pozwala na wybranie nazwy pliku docelowego – domyślnie jest to a.out. Próba uruchomienia programu na hoście zakończy się niepowodzeniem, ponieważ plik binarny został wykonany dla innej architektury. Przekonamy się o tym wyświetlając informację o pliku poleceniem file hello (**rysunek 5**). Skopiujmy więc plik hello poleceniem scp na urządzenie docelowe, czyli Odroida za pomocą polecenia scp hello odroid@192.168.0.11 i spróbujmy uruchomić ./hello po zalogowaniu. Powinniśmy zobaczyć napis tak jak na **rysunku 6**.

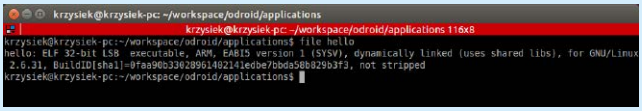
Jeżeli program hello wykonał się poprawnie wyświetlając komunikat jak na **rysunku 7**, to toolchain jest gotowy do kompilowania jądra systemu i innych aplikacji.

Sprzętowe Hello World

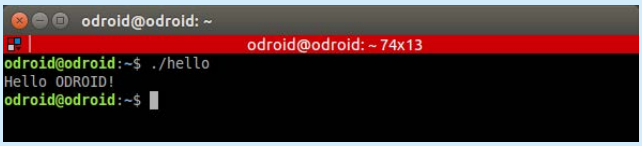
Odpowiednikiem poprzedniego przykładu w świecie elektroniki jest oczywiście mruganie diodą. Z uwagi na to, że funkcjonalność ta jest wykorzystywana praktycznie w każdym urządzeniu do sygnalizacji, w jądrze Linuksa znajdziemy sterownik napisany właśnie do obsługi LEDów. Sterownik ten jest aktywny w dystrybucji zainstalowanej dystrybucji

Listing 1. Pierwszy program dla Odroida

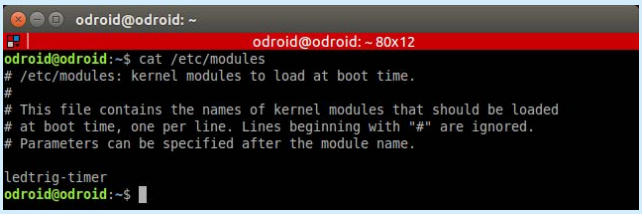
```
#include <stdio.h>
int main(void) {
    printf("Hello ODROID!\n");
    return 0;
}
```



Rysunek 5. Informacje o pliku *hello*



Rysunek 6. Efekt wykonania programu *hello*



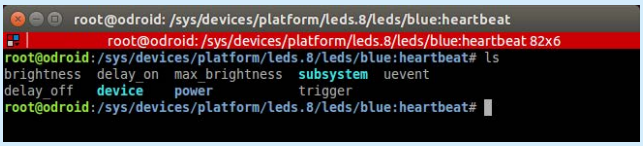
Rysunek 7. Dodanie modułu *ledtrig-timer* do pliku */etc/modules*

Ubuntu, ponieważ korzysta z niego niebieska dioda ALIVE (D1) znajdująca się na płytce. Można się o tym przekonać przeglądając zawartość katalogu */sys/class/leds/* poleceniem `ls /sys/class/leds/`. Znajduje się tam katalog *blue:heartbeat* zawierający ustawienia diody. Na uwagę zasługują znajdujące się w nim pliki *brightness* oraz *trigger*. Pierwszy pozwala na odczyt i ustawienie jasności diody. W wypadku sterowania diodą ze zwykłego portu GPIO, każda wartość od 1 do 255 powoduje świecenie, natomiast wartość 0 zgaszenie diody. Aby zapisać wartość do tego pliku potrzebne są uprawnienia roota. Pozostali użytkownicy mogą go jedynie odczytać. W pliku trigger możliwe jest sprawdzenie i zmiana sposobu wyzwalania diody. Po wyświetleniu zawartości pliku poleceniem *cat trigger* uzyskamy listę możliwych opcji wyzwalania z zaznaczonym nawiasami kwadratowymi aktualnym ustawieniem. Możemy zmienić tą wartość wpisując nową (jako root) bezpośrednio do pliku trigger poleceniami:

```
sudo su
echo cpu0 > trigger
```

Powyższa konfiguracja zmieni ustawienia diody z domyślnego trybu heartbeat, który charakteryzuje się powtarzanym, dwukrotnym mrugnięciem diody, w tryb wskazujący użycie pierwszego rdzenia procesora.

Możliwości wyzwalania jest więcej niż wskazuje na to zawartość pliku trigger, jednak nie są one domyślnie dostępne. Część z trybów została skompilowana jako moduły jądra i należy je najpierw włączyć. Listę dostępnych modułów wyzwalania można sprawdzić poleceniem *ls /lib/modules/3.10.80-131/kernel/drivers/leds/trigger/*. Aby z nich skorzystać trzeba je najpierw załadować. W wypadku wyzwalania typu timer, można wywołać (ładowanie modułów wymaga uprawnień roota) za pomocą



Rysunek 8. Zawartość katalogu */sys/class/leds/blue:heartbeat* po ustawieniu wyzwalania diody na trigger

Listing 2. Sekcja odpowiedzialna za konfigurację LEDów

```
leds {
    compatible = „gpio-leds”;
    /* Blue LED */
    hearbeat {
        label = „blue:heartbeat”;
        gpios = „GPIOAO_13”;
        default-state = „off”;
        linux,default-trigger = „heartbeat”;
    };
};
```

Listing 3. Konfiguracja diody LED

```
leds {
    compatible = „gpio-leds”;
    /* Blue LED */
    hearbeat {
        label = „blue:heartbeat”;
        gpios = „GPIOAO_13”;
        default-state = „off”;
        linux,default-trigger = „heartbeat”;
    };
    my_led {
        gpios = „GPIOAO_83”;
        default-state = „off”;
        linux,default-trigger = „timer”;
    };
};
```

`sudo modprobe ledtrig-timer`. Od tej pory w pliku */sys/class/leds/blue\:heartbeat/trigger* będzie dostępna dodatkowa opcja timer. Moduł załadowany w opisany sposób, będzie dostępny w jądrze do czasu jego jawnego usunięcia, lub do czasu ponownego uruchomienia systemu. Aby moduł był domyślnie ładowany za każdym razem po uruchomieniu systemu, można wpisać jego nazwę do pliku */etc/modules* (rys. 7). Można to zrobić na uruchomionym Odroidzie używając jednego z dostępnych edytorów tekstu, np. nano – `sudo nano /etc/modules`. Zmieńmy ponownie wyzwalanie diody, tym razem na timer. Powinniśmy zaobserwować stałe, równomierne mruganie diody z okresem 1 sekundy. Po ponownym wyświetleniu zawartości katalogu */sys/class/leds/led0* znajdziemy dwa dodatkowe pliki: *delay_on* i *delay_off* (**rysunek 8**). Znajduje się w nich odpowiednio czas włączenia i wyłączenia diody w milisekundach. Wpisując do nich różne wartości możemy zmieniać czas świecenia i okres mrugania.

Podłączenie dodatkowych LEDów do wyprowadzeń GPIO wymaga modyfikacji konfiguracji sprzętowej Odroida, odczytywanej przez jądro systemu. Jest to konieczne w przypadku urządzeń, które nie mogą być automatycznie wykryte przez system operacyjny (w przeciwieństwie do urządzeń typu plug&play). W używanej w przykładzie wersji jądra 3.10.80 wspomniana konfiguracja sprzętowa nie jest kompilowana razem z kodem jądra, lecz dostarczana osobno w pliku *dts* (Device Tree Source). Do jego modyfikacji potrzeba źródeł jądra, dostępnych w oficjalnym repozytorium projektu i możliwych do pobrania za pośrednictwem gita:

```
git clone --depth 1 https://github.com/
hardkernel/linux.git -b odroidc-3.10.y
```

Po sklonowaniu repozytorium na hosta można dokonać modyfikacji w oryginalnym drzewie urządzeń znajdującym się w pliku *linux/arch/arm/boot/dts/meson8b_odroidc.dts*. Po pierwsze należy znaleźć sekcję odpowiedzialną za konfigurację LEDów – **listing 2**. Sekcja ta zawiera odnośnik do użytego sterownika (*gpio-leds*), nazwę (*hearbeat*), etykietę (*blue:heartbeat*), linię GPIO (*GPIOAO_13*), domyślny stan (off) oraz domyślne wyzwalanie (*heartbeat*). Więcej informacji na temat wymienionych pól można znaleźć w dokumentacji jądra znajdującej się w pobranym repozytorium:

```
linux/Documentation/devicetree/bindings/leds/
common.txt
linux/Documentation/devicetree/bindings/leds/
leds-gpio.txt
```

W przykładzie dodatkowa dioda zostanie podłączona przez rezystor do wyprowadzenia 7 gniazda J2 (GPIOAO_83, opis wyprowadzeń na płytce można znaleźć pod adresem ***https://goo.gl/1XyKNC***). Jej konfigurację w pliku dts pokazano na **listingu 3**. Teraz można skompilować zmodyfikowany plik dts za pomocą komend wywoływanych z głównego katalogu sklonowanego repozytorium jądra – linux:

```
cp arch/arm/configs/odroidc_defconfig .config
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
make oldconfig
make dtbs
```

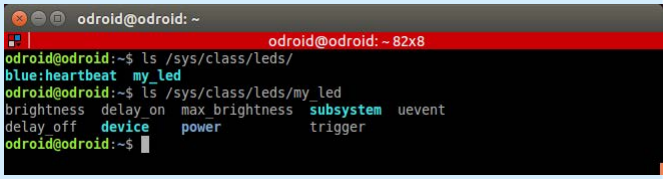
Pierwsza z nich kopiuje domyślną konfigurację jądra do pliku .config wymaganego podczas kompilacji. Jest to plik przygotowany specjalnie dla Odroida i dostarczany razem ze źródłami jądra. Kolejne dwa polecenia przygotowują zmienne środowiskowe używane podczas kompilacji: architekturę procesora oraz nazwę toolchaina. Zmienna CROSS_COMPILE będzie poprawna, jeżeli wcześniej została wyeksportowana zmienna PATH zawierająca odpowiednią ścieżkę do kompilatora. Następne polecenie ma na celu sprawdzenie, czy wszystkie parametry jądra zostały ustawione. Jeżeli nie, będzie je można uzupełnić. Ostatnia komenda uruchamia kompilację zaktualizowanego drzewa urządzeń. W wyniku jej wykonania powstaje nowy plik `arch/arm/boot/dts/meson8b_odroidc.dtb`. Należy go skopiować na partycję BOOT karty pamięci po podłączeniu jej do hosta. Dobrym pomysłem jest wykonaniu kopii zapasowej oryginalnego pliku dtb, znajdującego się na karcie, przed jego nadpisaniem.

Po ponownym włączeniu Odroida, w katalogu */sys/class/leds* będą widoczne dwa katalogi: znany już *blue:heartbeat* oraz nowoutworzony *my_led* (**rysunek 9**), natomiast dioda przyłączona do wyprowadzenia GPIO zacznie mrugać zgodnie z domyślnym ustawieniem wyzwalania timer.

KRZYSZTOF CHOJNOWSKI

Bibliografia:

https://goo.gl/BWL8Ie
https://goo.gl/yiox1X
https://goo.gl/IzwKMF



Rysunek 9. Nowa dioda widoczna po modyfikacji pliku *dts*