Użytkowanie Odroid-C1+ (2)

Obsługa systemu plików oraz GPIO

W poprzednim artykule zajmowaliśmy się przygotowaniem ODROIDa do pracy. Zainstalowaliśmy Ubuntu 14.04, nawiązaliśmy z nim połączenie za pomocą portu szeregowego i sieci lokalnej, a także skompilowaliśmy i uruchomiliśmy pierwszy program napisany w języku C. Tym razem zajmiemy się obsługą GPIO i pracą z plikami w systemie Linux. Na koniec zapoznamy się z narzędziami, które potrafią ułatwić pracę każdemu programiście: systemem kontroli wersji git oraz edytorem tekstu vim.

Obsługa GPIO jest jednym z podstawowych zadań systemów wbudowanych, więc nic dziwnego, że jądro Linuksa zawiera odpowiedni sterownik. Podobnie jak LEDy, linie GPIO mogą być konfigurowane za pomocą plików znajdujących się w katalogu */sys/class/gpio*.

Linie GPIO można skonfigurować za pomocą pliku *export* znajdującego się we wspomnianym katalogu: */sys/class/gpio*. W pierwszej kolejności należy do pliku *export* wpisać numer pinu. Najłatwiej można to zrobić jako root za pomocą linii poleceń, np. *echo 88 > /sys/class/gpio/export*. To polecenie spowoduje skonfigurowanie GPIO 88 (wyprowadzenie 11 na konektorze J2 – **rysunek 1**) jako wejście i utworzenie nowego katalogu: */sys/class/gpio/gpio88*. Spośród plików znajdujących się w tym katalogu potrzebne nam będą następujące:

- *active_low* – pin aktywny, gdy ma poziom niski (wartość *1*) lub wysoki (wartość *0*).
- *direction* – kierunek pinu, może być wejściem (wartość *in*) lub wyjściem (wartość *out*).
- *value* – dla wejścia z pliku można odczytać poziom pinu, dla wyjścia wpisując do pliku wartości można zmienić występujący na nim poziom (wartości *0* lub *1*).

Deinicjalizacja pinu odbywa się przez plik */sys/class/gpio/unexport*. Należy do niego wpisać pin, którego konfigurację chcemy usunąć *echo 88 > /sys/class/gpio/unexport*. Jak widać sterowanie liniami wejścia/wyjścia przy wykorzystaniu sterownika GPIO można sprowadzić do operacji plikowych. Poniższe przykłady pokazują, w jaki sposób to zrobić.

ODROID-C1 40pin Layout

Power Pin
Special Function
GPIO/Special Function

WiringPi GPIO#	Export GPIO#	ODROID-C PIN		Label	HEADER		Label	ODROID-C PIN		Export GPIO#	WiringPi GPIO#
				3V3	1	2	5V0				
			I2CA_SDA	SDA1	3	4	5V0				
			I2CA_SCL	SCL1	5	6	GND				
7	83		GPIOY.BIT3	#83	7	8	TXD1	TXD_B		113	
				GND	9	10	RXD1	RXD_B		114	
0	88		GPIOY.BIT8	#88	11	12	#87	GPIOY.BIT7		87	1
2	116		GPIOX.BIT19	#116	13	14	GND				
3	115		GPIOX.BIT18	#115	15	16	#104	GPIOX.BIT7		104	4
				3V3	17	18	#102	GPIOX.BIT5		102	5
12	107	MOSI	GPIOX.BIT10	MOSI	19	20	GND				
13	106	MISO	GPIOX.BIT9	MISO	21	22	#103	GPIOX.BIT6		103	6
14	105	SCLK	GPIOX.BIT8	SCLK	23	24	CE0	GPIOX.BIT20	CE0	117	10
				GND	25	26	#118	GPIOX.BIT21		118	11
			I2CB_SDA	SDA2	27	28	SCL2	I2CB_SCL			
21	101		GPIOX.BIT4	#101	29	30	GND				
22	100		GPIOX.BIT3	#100	31	32	#99	GPIOX.BIT2		99	26
23	108		GPIOX.BIT11	#108	33	34	GND				
24	97		GPIOX.BIT0	#97	35	36	#98	GPIOX.BIT1		98	27
			ADC.AIN1	AIN1	37	38	1V8	1V8			
				GND	39	40	AIN0	ADC.AIN0			

Rysunek 1. Złącze J2 płytki ODROID (źródło: *https://goo.gl/VTckag*)

Sterowanie z linii poleceń

Najłatwiejszym sposobem na kontrolę GPIO jest wiersz poleceń. ODROID umożliwia wybranie 19 pinów (rys. 1). W konfiguracji domyślnej, dostęp do nich możliwy jest wyłącznie z uprawnieniami roota, po wpisaniu *sudo* przed każdym poleceniem.

Na początku należy wyeksportować wybrany pin wpisując do pliku */sys/class/gpio/export* jego numer. Dla pinu 87 (12 na JP2) można to zrobić poleceniem *echo 87 > /sys/class/gpio/export*. Następnie, za pomocą polecenia *echo out > /sys/class/gpio/gpio87/direction* można wybrać kierunek pinu (w przykładzie będzie on wyjściem). Ostatnią czynnością jest ustawienie na porcie wybranego poziomu logicznego – *echo 1 > /sys/class/gpio/gpio87/value*. Jeżeli do wybranego pinu mamy przyłączoną diodę, jak w poprzednim artykule, powinna się ona zaświecić. Dodatkowo,

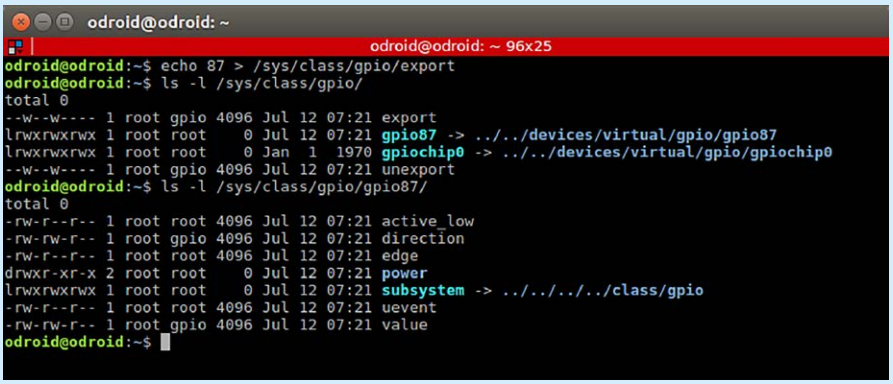
```
odroid@odroid: ~
odroid@odroid: ~ 104x11
odroid@odroid:~$ ls -l /sys/class/gpio/
total 0
--w------- 1 root root 4096 Jul  4 04:06 export
lrwxrwxrwx 1 root root    0 Jul  4 04:06 gpio87 -> ../../devices/virtual/gpio/gpio87
lrwxrwxrwx 1 root root    0 Jul  4 03:56 gpio88 -> ../../devices/virtual/gpio/gpio88
lrwxrwxrwx 1 root root    0 Jan  1  1970 gpiochip0 -> ../../devices/virtual/gpio/gpiochip0
--w------- 1 root root 4096 Jan  1  1970 unexport
odroid@odroid:~$
```

Rysunek 2. Struktura plików sterownika GPIO

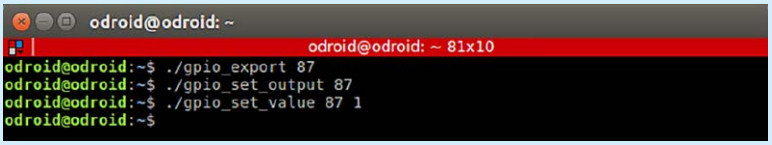
jeśli do pliku *active_low* wpiszemy wartość *1*, poziom wysoki na wyjściu wystąpi po wpisaniu do pliku *value* wartości *0:*

```
echo 1 > /sys/class/gpio/
gpio87/active_low
echo 0 > /sys/class/gpio/
gpio87/value.
```

Wyprowadzenie można wyzerować zmieniając wartość wpisaną w pliku *value* – *echo 0 > /sys/class/gpio/gpio87/value* lub *echo 1 > /sys/class/gpio/gpio87/value* (jeżeli wcześniej wpisaliśmy *1* do pliku *active_low*).



Rysunek 3. Grupy plików GPIO po dodaniu reguły udev



Rysunek 4. Obsługa GPIO za pomocą przykładu programu w języku C

Krótkiego komentarza wymaga jedynie odwołanie do powyższych plików z uprawnieniami administratora *(sudo* przed poleceniem*)*. Na początku wywołajmy polecenie *ls -l /sys/class/gpio/*. Na liście plików i katalogów (**rysunek 2**) widać, kto jest ich właścicielem, do jakiej grupy należą oraz jakie uprawnienia mają właściciel oraz użytkownicy należący do wskazanej grupy. Dla przykładu, plik *export*, należy do użytkownika *root* oraz grupy *root*. Oznacza to, że tylko administrator systemu ma możliwość dodawania nowych wyprowadzeń GPIO. Aby to zmienić, można wykorzystać system rejestracji urządzeń **udev**. Pozwala on na dodanie reguł uruchamianych za każdym razem, kiedy jest rejestrowane określone urządzenie. System ten będzie szerzej opisany w kolejnym artykule, ale już teraz możemy utworzyć proste reguły w katalogu /etc/udev/rules.d/, które pozwolą nam na dostęp do podstawowych możliwości GPIO z poziomu zwykłego użytkownika, więc utwórzmy plik **/etc/udev/rules.d/90-gpio.rules** i wpiszmy do niego dwie reguły:

```
SUBSYSTEM==”gpio”,KERNEL==”gpiochip*”,RUN+=”/
bin/sh -c ,/bin/chgrp gpio /sys/class/gpio/
export /sys/class/gpio/unexport’”,RUN+=”/bin/
sh -c ,/bin/chmod g+w /sys/class/gpio/export
/sys/class/gpio/unexport’”
SUBSYSTEM==”gpio”,KERNEL==”gpio*”,RUN+=”/bin/
sh -c ,/bin/chgrp gpio /sys%p/value  /sys%p/
direction’”,RUN+=”/bin/sh -c ,/bin/chmod g+w
/sys%p/value  /sys%p/direction’”
```

Obie reguły, zaczynające się od słowa SUBSYSTEM, powinny być umieszczone w pojedynczych liniach. Pierwsza reguła zostanie wywołana podczas dodawania urządzenia gpiochip0 i zmieni grupę, do której należą pliki export i unexport na gpio. Druga reguła dotyczy eksportowanych pinów, zmieniając grupę plików value oraz direction. W podobny sposób można modyfikować dostęp do innych plików GPIO. Konieczne jest jeszcze dodanie wspomnianej grupy (*sudo groupadd gpio*) oraz dodanie do niej użytkownika odroid (*sudo usermod –a –G gpio odroid*).

Po ponownym uruchomieniu systemu wszystkie zmiany powinny być widoczne, a dostęp do plików GPIO powinien mieć każdy użytkownik należący do grupy gpio. Efekt pokazano na **rysunku 3**. Dodatkowe informacje na temat sterownika GPIO oraz innych, dostarczanych przez jądro możemy znaleźć bezpośrednio w dokumentacji *Documentation/gpio.txt*.

Przykład w C

Często słyszy się, ze w Linuksie „wszystko jest plikiem”. Zobaczyliśmy już, że jest to prawdą także w wypadku sterowników GPIO i LED. Oznacza to, że z łatwością możemy użyć tych sterowników w dowolnej aplikacji, a jedyne, co musimy zrobić, to napisanie obsługi odczytu i zapisu do pliku.

W przytoczonym przykładzie użyjemy trzech programów napisanych w C: do eksportu linii GPIO, skonfigurowania jej jako wyjścia i zmiany poziomu. Kod źródłowy został umieszczony na githubie. Repozytorium można pobrać wywołując *git clone* ***https://goo.gl/IgGMY1***.

Po ściągnięciu repozytorium znajdziemy w nim trzy programy. Są one do siebie bardzo podobne. Ich uruchomienie powoduje pobranie z wiersza poleceń numeru pinu (oraz wartości wyjścia w wypadku gpio_set_value), utworzenie ścieżki do odpowiedniego pliku i wpisanie do niego wartości. Należy wywołać je w następującej kolejności:

```
gpio_export N
gpio_set_output N
gpio_set_value N V.
```

W miejsce *N* należy wpisać numer konfigurowanego pinu, natomiast w miejsce *V* wartość wyjścia: 0 albo 1 (**rysunek 4**).

Jako ćwiczenie, zachęcam do rozbudowy przykładu, lub napisanie kolejnych aplikacji, np. do odczytu stanu na pinie skonfigurowanym jako wejście.

Przykład w C++

Kolejny przykład napisany został w C++. Umożliwia, zarządzanie liniami GPIO z konsoli za pomocą interaktywnego menu. Można go pobrać z repozytorium git wpisując *git clone* ***https://goo.gl/e3jwHv***.

Pod względem obsługi GPIO program nie różni się niczym od przedstawionego w poprzednim przykładzie. Oba korzystają ze sterownika GPIO dostarczanego przez jądro, wykonując zapis i odczyt plików. Jednak w odróżnieniu

```
odroid@odroid: ~
odroid@odroid: ~ 74x41
[7] Reset
[8] Print
[0] Exit
5
Enter pin index: 0
[1] Add Pin
[2] Export
[3] Unexport
[4] Set Input
[5] Set Output
[6] Set
[7] Reset
[8] Print
[0] Exit
8
[0] GPIO:
        pin:    87
        dir:    /sys/class/gpio/gpio87
        exists: 1
        value:  0
[1] Add Pin
[2] Export
[3] Unexport
[4] Set Input
[5] Set Output
[6] Set
[7] Reset
[8] Print
[0] Exit
6
Entetr pin index: 0
[1] Add Pin
[2] Export
[3] Unexport
[4] Set Input
[5] Set Output
[6] Set
[7] Reset
[8] Print
[0] Exit
```

Rysunek 5. Przykład aplikacji w C++ do konfiguracji i obsługi GPIO

od przykładu w C, ten umożliwia dynamiczne konfigurowanie i sterowanie GPIO w obrębie jednej aplikacji.

Do kontroli GPIO jest używana klasa znajdująca się w plikach *gpio.hpp* oraz *gpio.cpp*. Dostarcza ona metod do sterowania i wyświetlania stanu. W pliku *main.cpp* znajduje się wyświetlanie menu oraz obsługa opcji wpisywanych przez użytkownika (**rysunek 5**).

Narzędzia

Nieodłączną częścią pracy programisty są narzędzia pozwalające na tworzenie i rozwój aplikacji. W tej i kolejnych częściach cyklu, przy okazji poznawania jądra Linuksa, zapoznamy się z programami, dzięki którym praca programisty może stać się bardziej efektywna.

Vim

Podstawowym narzędziem pracy każdego programisty jest edytor tekstu umożliwiający pisanie kodu programu. Wybór jest oczywiście bardzo duży. Od edytorów konsolowych (np. Vim, Emacs, Nano), przez edytory graficzne (np. Gedit, Kate, Geany), aż po rozbudowane środowiska (np. Eclipse, CodeBlocks). Mimo, że edytory okienkowe mogą okazać się na pierwszy rzut oka wygodniejsze i prostsze w obsłudze, warto zwrócić uwagę na powstałego w 1991 r. edytora Vim (**rysunek 6**) i to z kilku powodów.

Pierwszym, być może najważniejszym powodem, jest praca zdalna. Bardzo często mamy możliwość połączenia się z systemem jedynie na odległość, np. za pośrednictwem sieci Ethernet. W takim przypadku warto mieć możliwość edycji plików w systemie docelowym. Taką możliwość dają edytory tekstu uruchamiane w konsoli, m.in. Vim. Dlatego warto poznać jego możliwości już podczas pracy nad aplikacją w systemie hosta.

Kolejnym powodem przemawiającym za użyciem Vim'a, są jego możliwości. Mimo skromnego wyglądu, jest to potężne narzędzie z dużą ilością rozszerzeń. Jego odpowiednie użycie może przyspieszyć pracę nad kodem programu. Do nauki obsługi Vim'a posłużyć nam mogą zarówno liczne materiały dostępne w sieci, jak i samouczek dostarczany z samym programem.

Na koniec warto wspomnieć, że Vim jest jednym z najbardziej popularnych edytorów tekstu wybieranych przez użytkowników Linuksa. Czyni go to jedną z ważniejszych aplikacji w świecie Open Source, którą po prostu warto znać.

Git

Nie ma nic równie frustrującego jak utrata wyników kilkugodzinnej pracy nad kodem. Dlatego też, zanim zaczniemy pisać swoją aplikację warto pomyśleć o sposobie na zabezpieczenie wyników własnej pracy. Warto tutaj rozważyć jeden z systemów kontroli wersji. Pozwolą nam nie tylko na bezpieczne przechowywanie wyników własnej pracy, ale także na stworzenie historii wszystkich zmian w naszym projekcie.

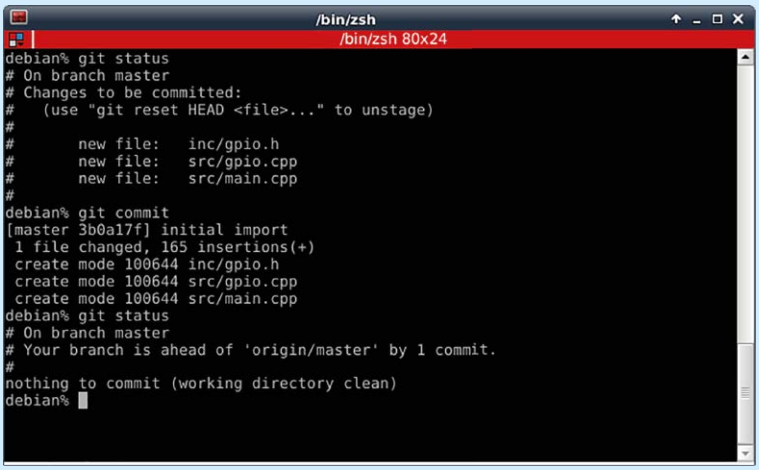
Jednym z najbardziej popularnych i zapewniających największe możliwości systemów kontroli wersji jest powstały w 2005 r. Git (**rysunek 6**). Sprawdza się on doskonale zarówno w projektach małych, nawet jednoosobowych, jak i w dużych, angażujących wielu programistów. Przykładem tych ostatnich jest jądro Linuksa, na którego potrzeby został wykonany Git.

Aby móc bezpiecznie przechowywać projekty, warto skorzystać z serwera Git. Można go przygotować samemu,

```
/bin/zsh
/bin/zsh 88x42
#include <cstdio>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <dirent.h>
#include <unistd.h>

const char *gpio_basepath =  "/sys/class/gpio/";

using namespace std;

void print_main_menu() {
    cout << "[1] Export" << endl;
    cout << "[2] Unexport" << endl;
    cout << "[3] Set" << endl;
    cout << "[4] Reset" << endl;
    cout << "[0] Exit" << endl;
}

int get_option() {
    int option;
    do {
        cin >> option;
    } while(!cin.good());
    return option;
}

int export_gpio(int gpio) {
    ostringstream oss;
    oss << gpio;

    string gpio_path(gpio_basepath);

    gpio_path += "gpio";
    gpio_path += oss.str();

    //check if gpio is already exported
    DIR * gpio_dir = opendir(gpio_path.c_str());
    if(gpio_dir != NULL)
        return -1;
                                                    1,1           Top
```

Rysunek 6. Edycja pliku w Vim

lub skorzystać z istniejących hostingów, jak np. Github i Sourceforge, pozwalających na darmowe przechowywanie projektów o otwartych źródłach. Aby rozpocząć pracę z systemem Git warto zapoznać się z licznymi materiałami w sieci. Dodatkowo serwis Github zawiera interaktywny samouczek pod adresem ***https://goo.gl/DxpdBP***.



Rysunek 7. Sprawdzanie stanu i wprowadzanie zmian do repozytorium

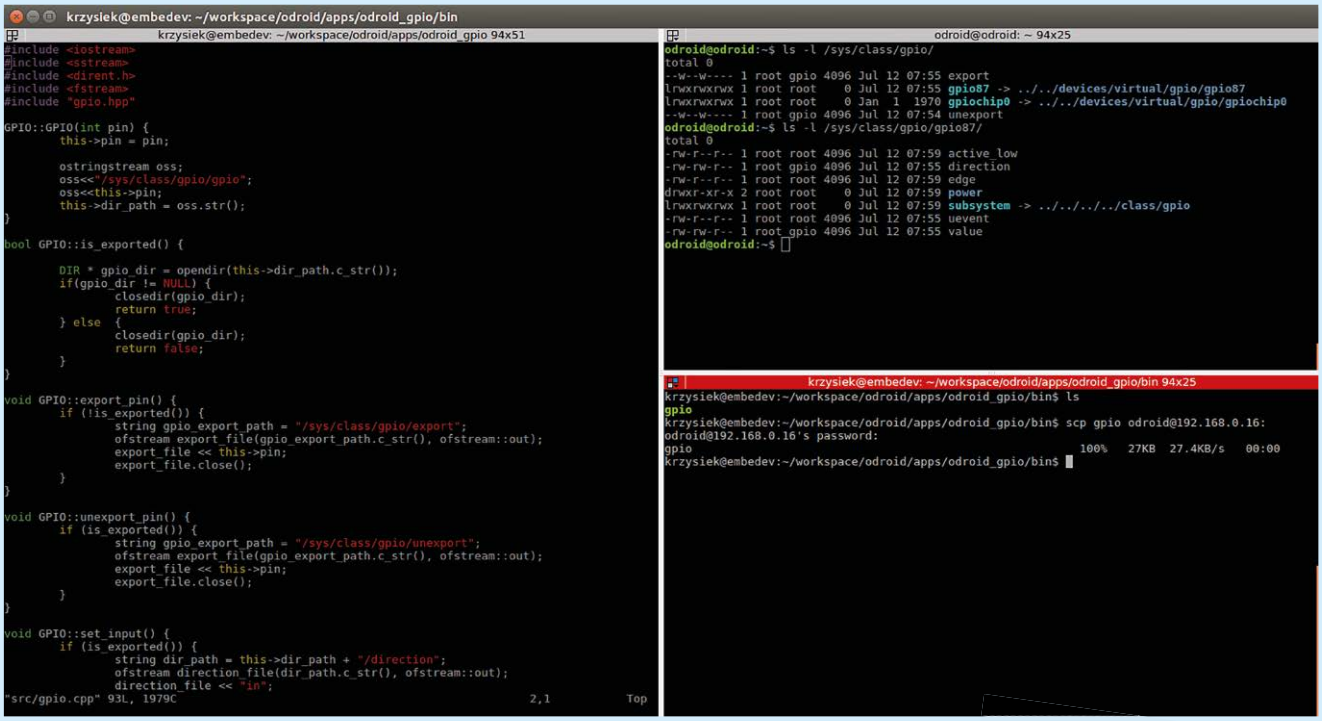
Terminator

Pracując z systemami wbudowanymi, bardzo często naszym podstawowym narzędziem jest terminal. Możemy za jego pomocą m.in. łączyć się zdalnie z urządzeniami, konfigurować je i kompilować programy, więc warto zastanowić się nad zmianą domyślnego programu terminala na coś dającego więcej możliwości podczas pracy z systemem hosta. Jedną z alternatyw jest Terminator (**rysunek 7**). Pozwala na podział okna głównego na kilka niezależnych sesji, dzięki czemu nie musimy się przełączać między wieloma otwartymi oknami.

KRZYSZTOF CHOJNOWSKI

Bibliografia:

https://goo.gl/s2GhLj
https://goo.gl/P8KYwR
https://goo.gl/Wwr9KF
https://goo.gl/Nr9sYP
https://goo.gl/Bzv4ie
https://goo.gl/jGnZKn



Rysunek 8. Podzielone okno Terminatora