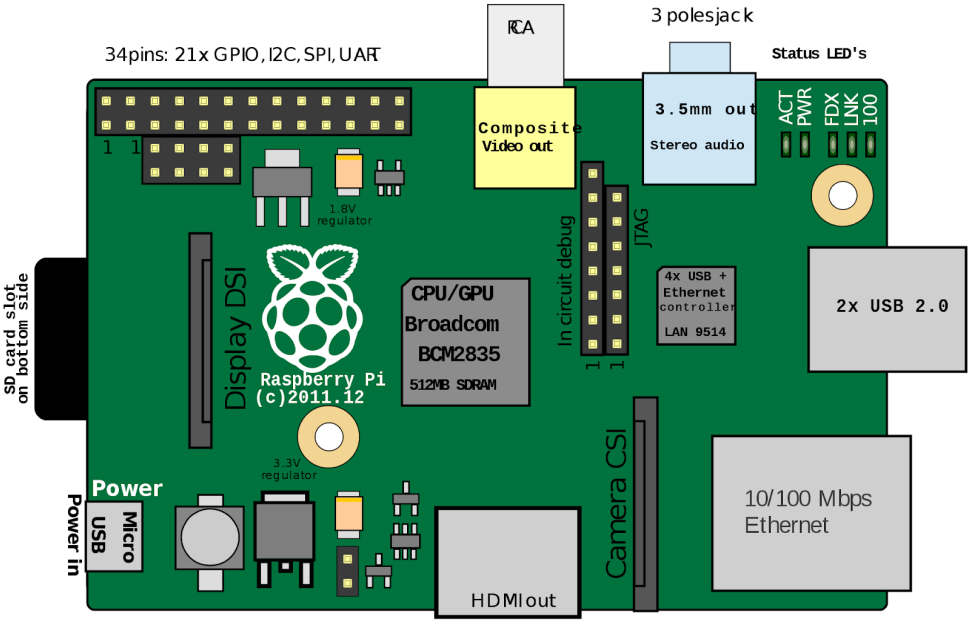
Linux w systemach wbudowanych

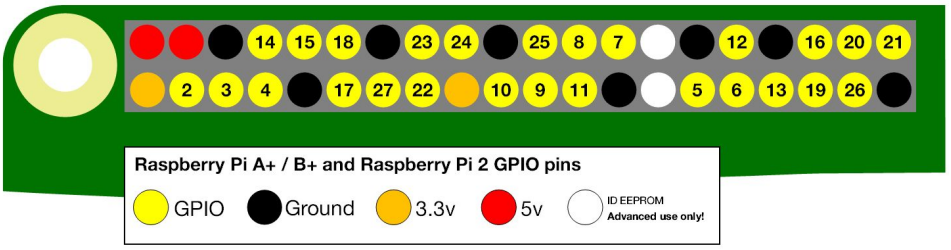
Laboratorium 2
Sprawozdanie
12.04.2018



Bartosz Selwesiuk

1) **GPIO - General Purpose Input/Output**

Raspberry Pi wyposażone jest w złącza wejścia/wyjścia ogólnego przeznaczenia (GPIO). GPIO jest to interfejs komunikacyjny pomiędzy elementami systemu komputerowego, takimi jak mikroprocesor czy urządzenie peryferyjne. Piny mogą pełnić zarówno rolę wejść, jak i wyjść i jest to zazwyczaj właściwość konfigurowalna. Dodatkowo są one często grupowane w porty.



2) **Obsługa pinów GPIO**

Zazwyczaj dostęp do pinów odbywa się bezpośrednie przez moduły jądra, ale również w celu ich konfiguracji i obsługi możemy skorzystać z interfejsu sysfs z poziomu linuxowego user space. Interfejs dostępny jest pod następującą ścieżką: */sys/class/gpio/.*

Podstawową i pierwszą w kolejności operacją związaną z obsługą pinów jest "zarezerwowanie" wybranego GPIO:
$ echo NN > /sys/class/gpio/export
NN - numer pinu

W przypadku, gdy operacja się powiodła, tworzony jest katalog:
/sys/class/gpio/gpioNN/.

W celu wybrania kierunku wykonujemy następującą komendę:
$ echo xx > /sys/class/gpio/gpioNN/direction
xx = [in | out | low | high]
(low & high włączają wyjście od razu ustawiając poziom)

Teraz możemy zarządzać poziomem pinu:
$ echo [0 | 1] > /sys/class/gpio/gpioNN/value
czy sterować generacją przerwań:
$ echo [none | rising| falling| both] > /sys/class/gpio/gpioNN/edge

3) **Biblioteki do obsługi GPIO**

Pinami możemy również oczywiście sterować z poziomu aplikacji. W przypadku C i Pythona możemy skorzystać odpowiednio z bibliotek WiringPi czy raspberry-gpio-python.

WiringPi, oprócz możliwości ustawiania pinów przy pomocy jej bibliotek, udostępnia dodatkowo aplikację konsolową **gpio**, której opcje sprawdzimy tutaj:
$ man gpio

WiringPi używa własnej numeracji pinów, innej niż numeracja Broadcom GPIO. Podstawową numerację możemy zapewnić inicjalizując bibliotekę przy użyciu metody **wiringPiSetupSys**. Wówczas WiringPi nie kontaktuje się z hardwarem bezpośrednio - korzysta z interfejsu sysfs opisanego wcześniej.

W wersji 2 biblioteki funkcje inicjalizujące zwracają zawsze 0 - by temu zapobiec, należy ustawić zmienną środowiskową WIRINGPI_CODES na dowolną wartość.

4) **Aplikacja - diody LED i przyciski**

Obsługa diod nie stanowiła żadnego problemu. W celu wykrywania przycisków konieczne było zaprojektowanie mechanizmu debounce'ingu. Takie podejście pozwala na zignorowanie nadmiernej ilości zmian sygnału i wykrycie tylko oczekiwanych zmian.

5) **Debugowanie aplikacji - gdb**

GDB (The GNU Debugger) to debugger będący częścią projektu GNU obsługujący wiele architektur i dostępny dla wielu systemów operacyjnych.

Debugowanie własnej aplikacji na Raspberry Pi z poziomu stacji roboczej wymaga następujących kroków:
- kompilacja odpowiednich pakietów w konfiguracji Buildroot:
    Target packages – Debugging,... → gdb → gdbserver
    Toolchain → Build cross gdb for the host
    Build options → build packages with debugging symbol
- kompilacja aplikacji z flagą *-g3* (dołączane są wówczas symbole i makra)
- uruchomienie serwera gdb na płytce:
    $ gdbserver host:7654 application
- uruchomienie gdb (na odpowiednią architekturę) na stacji roboczej, wykorzystując ten sam plik binarny
    $ BRPATH/output/host/usr/bin/arm-none-linux-gnueabi-gdb /ścieżka/application
    $ target remote xxx.yyy.zzz.vvv:7654

Na początku gdb zatrzymuje się przed wykonaniem programu. Aby ustawić breakpoint: *break N*, gdzie N - numer linii. W celu kontynuacji wykonujemy polecenie *continue* (lub *c*). Przejście do kolejnej linii programu: *step* (lub *s*). Komendą *list* wyświetlamy kod.

Pomoc na temat gdb:
$ gdb --help

6) **Przekształcenie aplikacji w pakiet Buildroota**

W przypadku aplikacji napisanej w języku skryptowym wystarczy zapewnić odpowiedni interpreter i biblioteki w konfiguracji Buildroota oraz umieścić skrypty w odpowiednim miejscu naszego systemu plików. Aplikacje napisanego w języku kompilowanym wymagają kompilacji.

Stworzenie pakietu:
- Config.in - plik opisujący sposób konfiguracji naszego pakietu, zawierający między innymi zależności od innych bibliotek, opis oraz link do strony projektu
- application.mk - plik definiujący wersję, licencję, lokalizację i sposób pobierania pakietu, a także komendy budujące/instalujące aplikację w naszym systemie plików
- powyższe pliki umieszczamy w *BRPATH/package/application*
- dodanie naszej aplikacji do *BRPATH/package/Config.in*
- po wybraniu paczki w *make menuconfig* i przekompilowaniu systemu aplikacja powinna znajdować się pod ścieżką */usr/bin*

Zbudowanie pakietu:
$ make application rebuild
w głównym katalogu Buildroota.

7) **Rozwiązanie**

W rozwiązaniu laboratorium dostarczone są trzy skrypty:
- *build_br.sh* - pobierający Buildroota i kopiujący pliki .config, Config.in oraz katalog z paczką
- *build_lab2.sh* - budujący aplikację
- *lab2.sh* - uruchamiający aplikację

Uwaga! Należy upewnić się, że ścieżki: do buildroota w build_lab2.sh oraz LAB2_SITE w package/lab2/lab2.mk są poprawne.

Plik *lab2_old.c* to aplikacja z laboratorium, *lab2.c* to wersja poprawiona.

8) **Referencje**

https://buildroot.org/downloads/manual/manual.html
http://www.mechatrobot.pl/rasp2/
https://falsinsoft.blogspot.com/2012/11/access-gpio-from-linux-user-space.html
http://wiringpi.com/reference/
https://stackoverflow.com/