Aleksy Barcz
Grzegorz Rząca

04.07.2010
Warszawa

Sterownik do przetwornika A/C PIC18F4550 podłączanego przez USB

http://www.ise.pw.edu.pl/~wzab/opiclab/

Dokumentacja końcowa

Wydział Elektroniki i Technik Informacyjnych - Politechnika Warszawska
Projekt z przedmiotu Sterowniki urządzeń - podstawy programowania

Spis treści:

1. Cel projektu

Sterownik przetwornika dla systemu Linux umożliwia jednoczesny odczyt danych z siedmiu kanałów urządzenia przez wiele programów użytkownika.
Każdy program użytkownika może zażądać od sterownika danych z dowolnej liczby spośród dostępnych kanałów, z dowolną częstotliwością, mieszczącą się w faktycznych możliwościach urządzenia oraz w pakietach o różnej ilości próbek (każda próbka stanowi zapis danych z wybranych kanałów z pojedynczej chwili czasu).
Sterownik powinien tak dobrać parametry próbkowania dla danego klienta, żeby w miarę możliwości spełnić jego wymagania, nie zakłócając pracy innych programów już pobierających dane od sterownika.
W przypadku, gdy program użytkownika nie nadąża z odbieraniem danych od sterownika, sterownik powinien go ostrzec, a przy braku reakcji na ostrzeżenie - odłączyć od urządzenia.

2. Komunikacja z programem użytkownika (a.k.a. dokumentacja użytkownika)

Do komunikacji ze sterownikiem używać należy komend IOCTL
IOCTL_SET_PARAMS oraz IOCTL_GET_DATA
Próba użycia funkcji read() i write() skończy się zwróceniem błędu.
Przy tworzeniu programu użytkownika trzeba dołączyć plik usb-ioctl.h w którym są odpowiednie prototypy
(oraz sys/ioctl.h w którym są niezbędne funkcje).
Użycie:
IOCTL_SET_PARAMS
Służy do 'przedstawienia się' sterownikowi i ustawienia/zmiany parametrów z jakimi chcemy pobierać próbki.
Do jądra musimy przekazać wskaźnik na tablicę przynajmniej 11 elementów typu int (z tylu będą czytane dane)
W pierwszym polu trzeba umieścić tzw. magic number (zdefiniowany jako stała USB_AD_IOCTL_MAGIC_NUMBER w usb-ioctl.h)
W drugim polu trzeba umieścić PID procesu użytkownika. Za pomocą numeru PID sterownik rozróżnia programy klienckie.

W trzecim polu trzeba umieścić częstotliwość z jaką chcemy próbkować
W czwartym polu trzeba umieścić ile próbek chcemy odbierać naraz
W następnych 7 określa się które kanały nas interesują 1 - będzie próbkowany 0 - nie będzie próbkowany

Wartość zwracana:
-1 w wypadku błędu
Należy używać zmiennej errno (extern int errno;) do sprawdzania jaki to był błąd
EFAULT w wypadku problemu z alokacją pamięci lub gdy sterownik nie będzie w stanie czytać z pamięci do której wskaźnik przekazaliśmy
ENODEV w wypadku braku urządzenia/interfejsu
EIO jeżeli magic number będzie nie poprawny
0 jeśli wszystko wykona się poprawnie

IOCTL_GET_DATA
Służy do pobierania danych z urządzenia.
Do jądra musimy przekazać wskaźnik na tablicę będącą w stanie pomieścić odpowiednią ilość próbek. W pierwszym polu (int) trzeba umieścić tzw. magic number (zdefiniowany jako stała USB_AD_IOCTL_MAGIC_NUMBER w usb-ioctl.h)
W drugim polu (int) trzeba umieścić PID procesu użytkownika. Za pomocą numeru PID sterownik rozróżnia programy klienckie.

Wartość zwracana:
-1 w wypadku błędu
Należy używać zmiennej errno (extern int errno;) do sprawdzania jaki to był błąd
EFAULT w wypadku problemu z alokacją pamięci lub gdy sterownik nie będzie w stanie czytać/pisać do odpowiedniej ilości pamięci do której wskaźnik przekazaliśmy lub gdy nie wykonaliśmy nigdy IOCTL_SET_PARAMS
ENODEV w wypadku braku urządzenia/interfejsu
EIO jeżeli magic number będzie nie poprawny
USB_AD_CLIENT_TOO_SLOW jeśli klient nie nadąża z odbieraniem danych
(UWAGA klient mimo to dostanie dane, jednak nie będą one aktualne [nie odświeżane od czasu zapełnienia pomocniczych buforów dla danego klienta])
0 jeśli wszystko wykona się poprawnie

3. Podstawowy cykl życia sterownika

Dla zapewnienia możliwie jak najszerszej puli jednoczesnych żądań, zaimplementowaliśmy następujące rozwiązanie:

- po wykryciu urządzenia, sterownik tworzy struktury danych potrzebne do komunikacji z programami użytkownika
- po podłączeniu pierwszego użytkownika, sterownik tworzy odpowiednie struktury danych dla programu użytkownika, a następnie rozpoczyna próbkowanie na wszystkich kanałach z maksymalną możliwą prędkością przy pomocy bulk URB zlecanych asynchronicznie - każde wywołanie metody read_callback powoduje zapisanie otrzymanych danych po kolei do buforów przypisanych do podłączonych użytkowników, ewentualne ustawienie flag ostrzeżeń bądź usunięcie zbyt wolnych klientów, a następnie ponowne zlecenie URBa do odczytu
- podłączenie każdego kolejnego użytkownika powoduje utworzenie dla niego odpowiednich struktur danych oraz rozpoczęcie zapisu do tych struktur danych z żądanych kanałów
- zbyt wolny program użytkownika zostaje ostrzeżony podczas kolejnej próby odczytu danych z urządzenia, a w przypadku braku reakcji na ostrzeżenie, bądź zbyt późnego odczytu, zostaje odłączony od sterownika, a przynależne mu struktury danych - usunięte
- odłączenie programu od sterownika jest realizowane automatycznie przez mechanizm usuwania zbyt powolnych programów
- odłączenie ostatniego z programów użytkownika powoduje zatrzymanie próbkowania z urządzenia
- odłączenie urządzenia powoduje usunięcie struktur danych

Ponadto:

- Odłączenie urządzenia w momencie, gdy są podłączone do niego programy użytkownika powoduje natychmiastowe zakończenie wszystkich ich operacji odczytu i, następnie, bezpieczne usunięcie struktur danych.

4. Struktury danych

Główną strukturą danych w sterowniku jest tablica klientów (programów użytkownika korzystających ze sterownika) - gpClients_array.
Każdy wpis w tablicy zawiera podstawowe dane opisujące klienta, takie jak:

- PID
- parametry próbkowania
- zmienne wykorzystywane do ostrzeżenia a następnie usunięcia klienta
- własną kolejkę, na której czeka na dane od sterownika
- dwa bufory danych

Każdy z buforów klienta zawiera:

- bufor danych o rozmiarze odpowiadającym wielkości pobranych danych z jednej chwili czasu wymnożonym przez ilość próbek z różnych chwil, jakie chce naraz otrzymywać klient, wraz ze znacznikiem pozycji

- parametr określający z ilu próbek należy wybierać jedną wartość (możliwe łatwe rozszerzenie funkcjonalności o uśrednianie danych z kilku różnych chwil) - dla zapewnienia pożądanej przez klienta częstotliwości próbkowania
- flagę, pokazującą czy bufor jest pełny

Każdy zapis do bufora (ma miejsce po każdym otrzymanym URBie z danymi od przetwornika) oraz odczyt (skopiowanie do przestrzeni użytkownika całego pełnego bufora danych) realizowane są wyłącznie przez odpowiednie metody bufora, zapewniające właściwy dostęp do danych.

Każdy wybór bajtów z aktualnej próbki otrzymanej od przetwornika do postaci gotowej do zapisania do buforów realizowany jest przez odpowiednią metodę klienta.

Klient budzony jest, gdy jeden z jego buforów jest pełen i gotów do odczytu. Dla zapewnienia, że zebrane dane nie ulegną nadpisaniu aż do momentu ich odczytania przez użytkownika, a kolejne nadchodzące próbki nie zostaną utracona w trakcie oczekiwania na odczyt, zastosowaliśmy podwójne buforowanie - w trakcie, gdy jeden bufor jest pełen i czeka na odczyt użytkownika, sterownik przenosi zapis do drugiego bufora i kontynuuje aż do jego zapełnienia.

5. Przykładowy program użytkownika usb-user

Dołączony przykładowy program (usb-user.c) pokazuje przykładową komunikację ze sterownikiem - pozwala na ustawienie parametrów próbkowania oraz ich zmianę w trakcie trwania komunikacji ze sterownikiem oraz wypisuje odbierane dane na konsolę.