Wprowadzenie do Linuksa embedded (4)

Obsługa interfejsu I²C

W systemach z mikrokontrolerem często zachodzi potrzeba transmisji danych pomiędzy układami zewnętrznymi. Popularnym standardem jest interfejs I²C. W artykule zaprezentowano, jak do jego obsługi zaprząc system operacyjny Linux embedded.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 10925, pass: 87thc181
- wszystkie poprzednie części kursu

W systemach mikroprocesorowych komunikacja z urządzeniami peryferyjnymi odbywa się za pomocą magistral szeregowych lub równoległych. W wypadku magistral równoległych najczęściej mamy do czynienia z magistralą systemową. Urządzenie takie jest wówczas dostępne jako fragment pamięci (np. architektura ARM) lub może być dostępne w specjalnej przestrzeni IO (np. architektura x86). Naturalnie w przypadku systemów wykorzystujących MMU obszar ten jest dostępny tylko dla jądra systemu operacyjnego. Wykorzystanie magistral równoległych wymaga wielu linii, skomplikowanych połączeń i generalnie jest zbyt niewygodne. Dlatego najczęściej w ten sposób podłączone są tylko wewnętrzne układy peryferyjne, wbudowane w mikrokontroler. Zewnętrzne układy najczęściej dołączane są za pomocą interfejsów szeregowych. Najpopularniejsze interfejsy szeregowe współczesnych systemów mikroprocesorowych to: SPI, I²C, USB, RS232/RS485, ETHERNET.

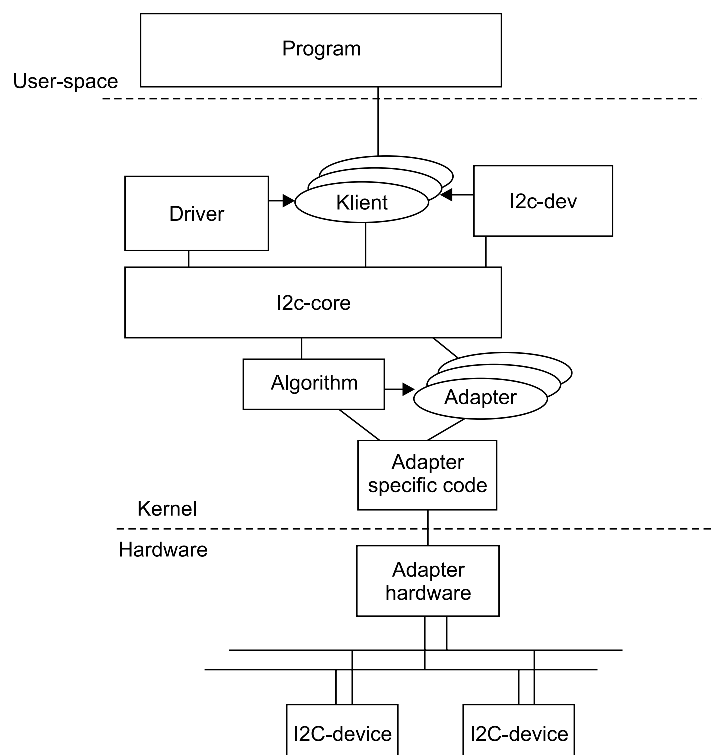
I²C oraz SPI są używane najczęściej do dołączania układów peryferyjnych, takich jak: zegary RTC, pamięci EEPROM itp., znajdujących się w obrębie płytki drukowanej z mikrokontrolerem. Interfejs USB jest stosowany do dołączania urządzeń zewnętrznych. Natomiast interfejsy RS232 i RS485 są używane zwykle do komunikacji z urządzeniami zdalnymi.

W przypadku mikrokontrolerów jednoukładowych oraz rozwiązań bez systemu operacyjnego stosunkowo proste do opanowania są interfejsy SPI, I²C i RS232. Dużo bardziej skomplikowana jest obsługa USB, przez co producenci mikrokontrolerów, aby ułatwić zadanie użytkownikom, dostarczają gotowe biblioteki programowe. Po zapoznaniu się z obsługą portów GPIO w Linuksie skupimy się ma obsłudze urządzeń peryferyjnych dołączanych za pomocą interfejsu I²C. Większość współczesnych mikrokontrolerów ma wbudowany sprzętowy kontroler I²C, którego programowanie jest stosunkowo łatwe, nawet bez pomocy gotowych bibliotek, natomiast w mikrokontrolerach niemających sprzętowego kontrolera I²C można bez większych trudności wykonać programowy interfejs pracujący w trybie *master*, korzystając z funkcji obsługi GPIO. Niestety, podczas tworzenia programów pracujących pod kontrolą systemu Linux metody, których używamy do pisania oprogramowania dla mikrokontrolerów jednoukładowych, nie są odpowiednie, z uwagi na ochronę pamięci oraz wielozadaniowość. Zatem należy skorzystać z pośrednictwa jądra systemu, gdzie istnieje gotowa architektura programowa przeznaczona do obsługi interfejsu I²C. W przykładzie pokażemy sposób obsługi za pomocą sterowników dostępnych w jądrze interfejsu I²C sprzęgającego mikrokontroler z układem zegarka PCF-8563 dostępnego na płycie BF210. Pokażemy również, w jaki sposób uzyskać dostęp do magistrali I²C z aplikacji użytkownika, na przykładzie czujnika temperatury **MCP9803** (moduł KAMOD-TEM), bez konieczności pisania sterowników jądra.

Podsystem I²C

W Linuksie implementacja wszelkich podsystemów realizowana jest w sposób warstwowy, zapewniając maksymalną uniwersalność oraz przenośność na wszystkie platformy wspierane przez jądro. Podobnie jest w przypadku podsystemu I²C. Podsystem ten umożliwia obsługę magistral I²C/SMBUS oraz wielu urządzeń do nich dołączonych. Na **rysunku 6** przedstawiono model programowy podsystemu I²C/SMBUS.

Sercem podsystemu jest moduł jądra *i2c-core*, który komunikuje się z modułem sterownika fizycznego kontrolera magistrali oraz zapewnia



Rysunek 6. Model programowy podsystemu I²C/SMBUS na podstawie *i2c.wiki.kernel.org*

Uwaga! W wypadku typowego adresowania 7-bitowego magistrali I²C, najmłodszy bit określa kierunek transmisji. Zwyczajowo adres I²C danego układu przedstawiamy w postaci liczby 8-bitowej z wyzerowanym najmłodszym bitem R/W=0 (rys. n)

A6	A5	A4	A3	A2	A1	A0	R/W
7	6	5	4	3	2	1	0

W podsystemie I²C Linuksa adres sprzętowy I²C przedstawiany inaczej, mianowicie jest to liczba pozbawiona bitu R/W przez przesunięcie bitowe o 1 w prawo (rys. n)

0	A6	A5	A4	A3	A2	A1	A0
7	6	5	4	3	2	1	0

Na przykład, jeżeli urządzenie ma adres 0x90, w Linuksie należy użyć adresu 0x48

jednolite API, niezależne od sprzętu, umożliwiające innym modułom korzystanie z I²C. Moduł sterownika kontrolera magistrali (Adapter) odpowiada za fizyczną komunikację z kontrolerem I²C, który jest zależny od platformy. Moduł ten może wykorzystywać kod pomocniczy z modułów Alghoritm. Moduł Alghoritm definiuje sposób, w kontroler odwołuje się do interfejsu I²C, ale bez wykonywania fizycznego dostępu do rejestrów kontrolera sprzętowego. Na przykład ten sam kontroler sprzętowy I²C może być w wypadku jednej architektury dołączony do magistrali PCI, natomiast w innym może być dołączony do magistrali systemowej. Z tego powodu różny będzie sposób dostępu do rejestrów tego kontrolera. Innym przykładem jest moduł *i2c-algo-bit*, który zawiera opis sterowania magistralą I²C za pomocą operacji bitowych, ale nie zawiera kodu bezpośrednio sterującego liniami GPIO. Za fizyczne sterowanie liniami jest odpowiedzialny moduł *i2c-gpio*.

Bardzo często moduły sterowników mają zdefiniowany własny algorytm dostępu do magistrali i nie używają dodatkowych modułów *Alghoritm*. Podystem *i2c-core* zapewnia jednolite API, dzięki którym użytkownik może tworzyć oprogramowanie w oderwaniu od obsługi samego interfejsu I²C. Urządzenia są mapowane przez system operacyjny i dostępne jako zasoby standardowe. Na przykład zegar RTC jest obsługiwany przez sterownik jądra i udostępnia urządzenie /dev/rtc0, a aplikacja korzystająca z RTC „nie wie", do jakiego rodzaju interfejsu jest on dołączony.

Istnieje również wydajny sposób dostępu do magistrali I²C z aplikacji użytkownika, realizowany przez moduł *I2c-dev*, który umożliwia dostęp do I²C za pomocą pliku urządzenia */dev/i2c-x* (gdzie x to kolejny numer interfejsu). Dzięki dostępowi do I²C z poziomu aplikacji użytkownika istnieje możliwość bezpośredniej obsługi urządzenia z pominięciem sterowników przestrzeni jądra. Jest to bardzo wygodne rozwiązanie, które może być wykorzystane, gdy nie ma potrzeby użycia funkcjonalnych sterowników, na przykład w wypadku, gdy układ I²C będzie używany jedynie przez pojedynczą aplikację. Może to być bardzo wygodne, ponieważ zwalnia użytkownika z konieczności tworzenia stosunkowo skomplikowanych i trudnych w uruchamianiu sterowników jądra czy konieczności jego rekompilacji oraz tworzenia wpisów w strukturach *platform-device*.

Implementacja kontrolera I²C na platformie BF210. Konfigurowanie *platform-device*

Mikrokontroler AT91RM9200 stanowiący serce Armputera BF210 ma zintegrowany kontroler magistrali I²C, który jest nazwany TWI. Z uwagi na liczne błędy sprzętowe kontrolera, a w szczególności brak możliwości generowania ponownej sekwencji *Start* przed wysłaniem sekwencji *Stop*, jest on w zasadzie bezużyteczny i jego obsługa nie jest dostępna w standardowym kodzie jądra. Implementacja protokołu I²C przy pojedynczym układzie *master* jest łatwa do wykonania z użyciem operacji na liniach GPIO. W systemie Linux istnieje implementacja obsługi I²C realizowana przez moduł *i2c-gpio* przy współpracy z modułem algorytmu *i2c-algo-bit*. Użycie sterownika GPIO jest okupione nieco większym obciążeniem CPU. Dzięki zastosowaniu tego sterownika istnieje możliwość implementacji w zasadzie dowolnej liczby niezależnych interfejsów I²C ograniczonych jedynie liczbą dostępnych linii GPIO. Informowanie o tym, które linie GPIO będą magistralą I²C, jest realizowane przez odpowiednie wpisy w strukturach *platform-device* (**listing 8**) w kodzie jądra odpowiedzialnym za inicjalizację maszyny (*arch/arm/mach-at91/board-boff210.c*).

Struktura *i2c_gpio_platform* zawiera informację o tym, które linie będą pełnić funkcję linii interfejsowych SDA (PA25) oraz SCL (PA26), informację o tym, czy linie GPIO są typu *Open-Drain* i czas trwania sygnału SCK, który ustawiono na 2 μs, co odpowiada trybowi *standard* (*100 kHz*). Tak przygotowana struktura następnie jest wpisywana do struktury *platform_device*, która jest rejestrowana w systemie za pomocą funkcji *platform_device_register()* – **listing 9**.

Przed rejestracją struktury linie PA25 i PA26 są konfigurowane jako wyjściowe z otwartym drenem. Tak zarejestrowana struktura umożliwia sterownikowi *i2c-gpio* odczytanie struktury poszczególnych magistral I²C oraz przyporządkowanych im linii GPIO.

Użycie sterowników funkcjonalnych wbudowanych w jądro

W jądrze istnieje wiele gotowych sterowników dla układów peryferyjnych dołączanych za pomocą I²C. Komunikują się one z magistralą poprzez wewnętrzne API podsystemu *i2c-core*, natomiast aplikacji udostępniają interfejs funkcjonalny reprezentujący funkcję pełnioną przez dany układ. Takim przykładem w BF210 jest układ zegara PCF8563, który jest obsługiwany przez moduł jądra, a w przestrzeni użytkownika jest widoczny w postaci urządzenia */dev/rtc*. To, że jest on dołączony akurat do I²C, jest nieistotne dla użytkownika, ważna jest jego funkcjonalność. Użycie sterowników dostępnych w jądrze jest zgodne z filozofią Linuksa, dlatego przed podjęciem decyzji o zastosowaniu API bezpośredniego dostępu do magistrali I²C należy upewnić się, że nie istnieje gotowy sterownik jądra dla danego układu.

W przeciwieństwie do SMBUS, interfejs I²C nie ma zdefiniowanego mechanizmu ska-

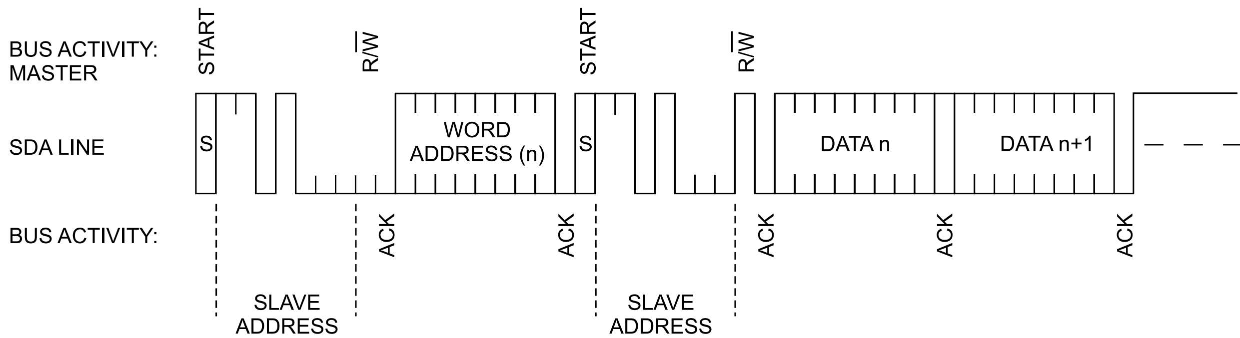
Listing 8. Wpisy w strukturze *platform-device* zawierające konfigurację linii interfejsowych I²C/GPIO

```
//I2c In gpio Mode
static struct i2c_gpio_platform_data i2c_gpio_data =
{
  .sda_pin        = AT91_PIN_PA25,
  .scl_pin        = AT91_PIN_PA26,
  .sda_is_open_drain    = 1,
  .scl_is_open_drain    = 1,
  .udelay            = 2,        /* ~100 kHz */
};

static struct platform_device i2c_gpio_device =
{
  .name           = "i2c-gpio",
  .id             = 0,
  .dev            =
  {
    .platform_data  = &i2c_gpio_data,
  }
};
```

Listing 9. Przygotowanie struktury z konfiguracją I²C

```
 //I2C code
 at91_set_GPIO_periph(AT91_PIN_PA25, 1);          /* TWD (SDA) */
 at91_set_multi_drive(AT91_PIN_PA25, 1);
 at91_set_GPIO_periph(AT91_PIN_PA26, 1);          /* TWCK (SCL) */
 at91_set_multi_drive(AT91_PIN_PA26, 1);
 i2c_register_board_info(0, bf210_i2c_devices, ARRAY_SIZE(bf210_i2c_devices));
 platform_device_register(&i2c_gpio_device);
```



Rysunek 7. Sekwencje odczyt/zapis bez bitu stopu

nowania dołączonych urządzeń, dlatego jedynym sposobem poinformowania o obecności danego układu na magistrali jest wykorzystanie mechanizmu *Platform Device*. Na przykład, aby poinformować o dołączeniu układu PCF8563 do wcześniej utworzonej magistrali *I2C0,* należy zdefiniować następującą strukturę w kodzie jądra odpowiedzialnym za inicjalizację maszyny:

```
//I2C device map
static struct i2c_board_info __
initdata bf210_i2c_devices[] =
{
  {
    I2C_BOARD_INFO("pcf8563", 0x51),
  }
};
```

W strukturze *i2c_board_info* trzeba zdefiniować typ układu dołączonego do I²C oraz poinformować o jego adresie sprzętowym. Następnie tak przygotowaną strukturę należy zarejestrować za pomocą wywołania:

```
i2c_register_board_info(0, bf210_
i2cdevices, ARRAY_SIZE(bf210_i2c_
devices));
```

Wywołanie to jako pierwszy argument przyjmuje numer porządkowy magistrali, do której zostało dołączone urządzenie, natomiast jako kolejne argumenty przyjmuje wskaźnik do opisanej wcześniej struktury oraz liczbę wpisów w strukturze. Tak zarejestrowana struktura pozwoli kernelowi na automatycznie załadowanie i inicjalizację odpowiedniego sterownika dla danego układu.

Obsługa urządzeń I²C z poziomu aplikacji z użyciem API bezpośredniego dostępu do magistrali

Brak zdefiniowanego mechanizmu autodetekcji układów dołączonych do magistrali I²C skutkuje koniecznością zarejestrowania układu w podsystemie *platform-device*, co jest związane z koniecznością modyfikacji kodu jądra. Modyfikacja i ponowna rekompilacja jądra może być kłopotliwa dla początkujących użytkowników. Brak dołączanego układu na liście dostępnych sterowników skutkuje koniecznością napisania sterownika jądra, co może być zadaniem jeszcze bardziej skomplikowanym. W wypadku, gdy nie potrzebujemy sterownika funkcjonalnego dostępnego z poziomu systemu, a dany układ będzie używany w obrębie jednej aplikacji lub projektu, zamiast pisania sterownika w przestrzeni jądra warto rozważyć możliwość użycia bezpośredniego dostępu do magistrali I²C oraz bezpośredniej obsługi układu z poziomu aplikacji. Funkcjonalność taką zapewnia moduł *i2c-dev*, którego zadaniem jest udostępnianie interfejsów I²C za pomocą plików urządzeń (*/dev/i2c-x*, gdzie x oznacza kolejny numer porządkowy). Aby uzyskać dostęp do API I²C w przestrzeni użytkownika, należy załadować moduł *i2c-dev (modprobe i2c-dev)*, a w programie dołączyć pliki nagłówkowe *<linux/i2c-dev.h>* oraz *<linux/i2c.h>*. Przed skorzystaniem z funkcjonalności dostępu do I²C trzeba otworzyć plik reprezentujący daną magistralę (dla 0 magistrali /dev/i2c-0), za pomocą wywołania open (...,O_RDWR). Po otwarciu pliku należy poinformować sterownik o sprzętowym adresie I²C, z którym chcemy się komunikować za pomocą wywołania systemowego ioctl: *ioctl(file, I2C_SLAVE, i2c_addr)*, w którym jako *i2c_addr* należy podać sprzętowy adres I²C układu. Po wykonaniu tej czynności możemy odczytywać dane z urządzenia o adresie sprzętowym *i2c_addr* za pomocą wywołania systemowego *read()* oraz zapisywać dane za pomocą wywołania systemowego *write()*.

Każda operacja odczytu lub zapisu spowoduje wygenerowanie bitu startu, po którym nastąpi wysłanie lub odczytanie danych, a na zakończenie zostanie wysłany bit stopu. Jeżeli chcemy komunikować się z różnymi układami dołączonymi do magistrali, należy za każdym razem przed wywołaniem funkcji *read/write* zmienić adres docelowy poprzez ponownie wywołanie funkcji *ioctl(file,I2C_SLAVE,new_i2c_addr)*, w którym *new_i2c_addr* to adres sprzętowy nowego układu.

Taki prosty sposób sterowania za pomocą wywołań *read/write*, choć dla wielu urządzeń może być wystarczający, nie odzwierciedla specyfiki magistrali I²C. Bardzo często układy I²C mają dodatkowy adres wewnętrzny, określający wewnętrzny rejestr układu i w przypadku żądania odczytu wymagają sekwencji odczyt/zapis bez generowania bitu stopu (**rysunek 7**)

W takim przypadku sekwencja wywołań systemowych *write()/read()* będzie nieodpowiednia, ponieważ pomiędzy wysłaniem adresu rejestru a odczytem danych będzie wygenerowany dodatkowy bit stopu. Aby uzyskać pożądaną sekwencję, należy zastosować wywołanie systemowe *ioctl(file, I2C_RDWR, struct i2c_rdwr_ioctl_data *msgset)*, które umożliwia wykonanie dowolnych transakcji na magistrali I²C jako pojedynczej sekwencji bez generowania bitu stopu. Wywołanie to jako trzeci argument przyjmuje wskaźnik do struktury *i2c_rdwr_ioctl_data*, która zdefiniowana jest następująco:

```
struct i2c_rdwr_ioctl_data {
  struct i2c_msg *msgs;  /* pointers
to i2c_msgs */
  __u32 nmsgs;           /* number
of i2c_msgs */
};
```

Pole ***msgs*** zawiera wskaźnik do tablicy struktur opisujących sekwencję transakcji I²C, natomiast pole ***nmsgs*** zawiera liczbę transakcji. Strukturę transakcji I²C (wiadomości) zdefiniowano jak na **listingu 10**.

Pole ***addr*** zawiera adres sprzętowy I²C, dla którego jest przeznaczona transakcja. Pole ***flags*** zawiera flagi transakcji umożliwiające sterowanie daną transakcją zgodnie z definicjami (np. flaga ***I2C_M_RD*** informuje, że jest to sekwencja odczytu; flaga ***I2C_M_RD*** informuje, że adres jest 10-bitowy). Pole ***len*** określa długość bufora z danymi do wysłania/odebrania, natomiast pole ****buf*** zawiera wskaźnik do bufora danych. Budując odpowiednią tablice składającą się z transakcji zapisu jednego bajtu, a następnie odczytu zadanej liczby bajtów, mamy możliwość wygenerowania prawidłowej sekwencji odczytu.

Dostęp do magistrali I²C z wiersza polecenia

Pakiet ***i2c-tools*** umożliwia dostęp do magistrali I²C z poziomu shella (wiersza polecenia). Wykorzystuje on opisany wcześniej moduł dostępu do magistrali I²C z aplikacji użytkownika ***i2c-dev***. Pakiet składa się z szeregu poleceń, z których najistotniejsze to:

i2cdetect [-y] [-a] [-q|-r] I2CBUS [FIRST LAST] – polecenie umożliwiające przeskanowanie dostępnych urządzeń dołączonych do magistrali I²C. Na przykład wywołanie polecenia *i2cdetect -y 0* w BF210 powoduje wyświetlenie tablicy wszystkich adresów wraz z listą dostępnych urządzeń na danej magistrali (**listing 11**).

Możemy zauważyć, że widoczne są dwa urządzenia: pod adresem 0x51 (zegar PCF8563) oraz pod adresem 0x48 (moduł KAMOD-TEM). Literki UU dla adresu 0x48 oznaczają, że dane urządzenie jest dołączone, ale jest niedostępne

Listing 10. Struktura transakcji I^{2}C

```
struct i2c_msg {
  __u16 addr;     /* slave address */
  __u16 flags;
#define I2C_M_TEN           0x0010 /* this is a 10-bit chip address */
#define I2C_M_RD            0x0001  /* read data, from slave to master */
#define I2C_M_NOSTART       0x4000  /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_REV_DIR_ADDR  0x2000  /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_IGNORE_NAK    0x1000  /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_NO_RD_ACK     0x0800  /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_RECV_LEN      0x0400  /* length will be first received byte */
  __u16 len;     /* msg length */
  __u8 *buf;     /* pointer to msg data */
};
```

Listing 11. Tablica wszystkich adresów wraz z listą dostępnych urządzeń na danej magistrali

```
root@bf210-at91:~# i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- UU -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Listing 12. Składnia polecenia *i2cget*

```
root@bf210-at91:~# i2cget
Usage: i2cget [-f] [-y] I2CBUS CHIP-ADDRESS [DATA-ADDRESS [MODE]]
  I2CBUS is an integer or an I2C bus name
  ADDRESS is an integer (0x03 - 0x77)
  MODE is one of:
    b (read byte data, default)
    w (read word data)
    c (write byte/read byte)
    Append p for SMBus PEC
```

Listing 13. Składnia polecenia *i2cset*

```
i2cset [-f] [-y] [-m MASK] I2CBUS CHIP-ADDRESS DATA-ADDRESS [VALUE] [MODE]
  I2CBUS is an integer or an I2C bus name
  ADDRESS is an integer (0x03 - 0x77)
  MODE is one of:
    c (byte, no value)
    b (byte data, default)
    w (word data)
    Append p for SMBus PEC
```

z poziomu aplikacji użytkownika, ponieważ jest kontrolowane przez sterownik jądra systemu operacyjnego. Natomiast urządzenie pod adresem 0x48 (KAMOD-TEM) jest dostępne, co w odpowiednim polu oznaczono cyfrą 48.

Polecenie *i2cget* umożliwia odczytanie danych z urządzenia dołączonego do magistrali I^{2}C. Składnię tego polecenia umieszczono na **listingu 12**. Jako argumenty przyjmuje ono odpowiednio numer interfejsu I^{2}C, numer sprzętowy urządzenia, adres rejestru do odczytania oraz długość słowa danych (8 lub 16 bitów).

Polecenie *i2cset* umożliwia wpisanie danych do układu podłączonego do magistrali I^{2}C. Jego składnię umieszczono na **listingu 13**. Jako pierwszy argument polecenie przyjmuje, odpowiednio: numer interfejsu I^{2}C, adres układu, adres rejestru, wartość do zapisania oraz długość danych do zapisania.

Przykład praktyczny

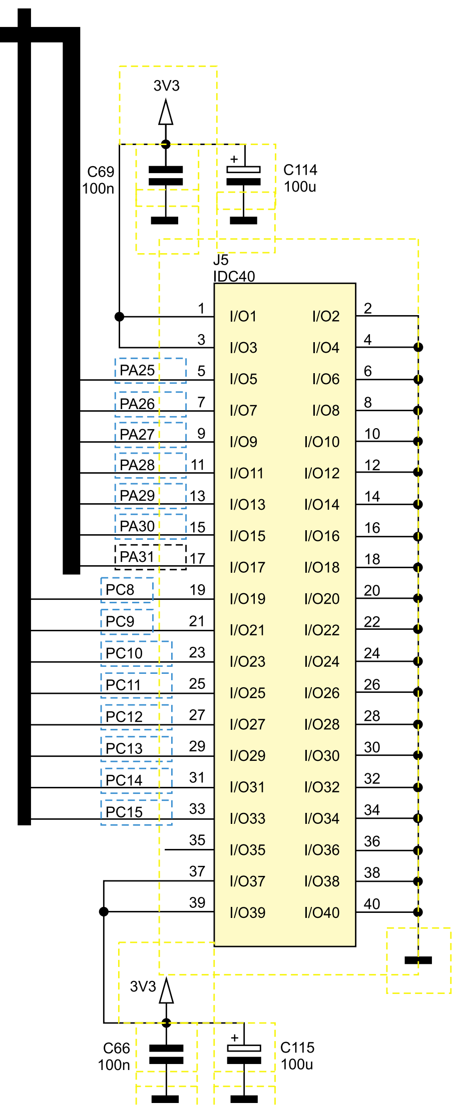
Jako przykład demonstrujący użycie układu dołączonego do magistrali I^{2}C z wykorzystaniem dostępu za pomocą **i2c-dev** pokazano aplikację, która będzie odczytywać temperaturę z modułu KAMOD-TEM, a następnie prezentować aktualny wynik z odświeżaniem co 1 s na standardowym wyjściu (konsoli). (*kod źródłowy aplikacji znajduje się w pliku i2ctemp.c umieszczonym na płycie CD i serwerze FTP*). Dostęp do magistrali I^{2}C będzie realizowany z użyciem opisanego wcześniej API. Do uruchomienia przykładu jest potrzebny moduł KAMOD-TEM. Sposób podłączenia modułu do płytki prototypowej BF210 przedstawiono na **rysunku 7**.

Moduł KAMOD-TEM zawiera czujnik temperatury **MCP2301**, który ma następujące parametry:

- jest wyposażony w interfejsy I^{2}C/SMBUS pracujące z maksymalną częstotliwością taktowania 400 kHz,
- napięcie zasilania 2,7...5,5 V,
- rozdzielczość pomiaru temperatury ustalaną od 9 do 12 bitów,
- zakres pomiary temperatury –55...+125°C z dokładnością ±1°C.

Adres bazowy układu to (1001xxx)b. Jego najmłodsze bity możemy ustalić za pomocą zwór konfiguracyjnych. Układ może generować sygnał alarmujący o przekroczeniu zadanej temperatury.

Po dołączeniu modułu należy upewnić się, że zwory **A0...A2** konfiguracji adresu zostały ustawione w pozycji „0" (adres sprzętowy czujnika 0x90h). Czujnik zawiera kilkanaście rejestrów kontrolnych i konfigurujących umożliwiających między innymi: ustalenie rozdzielczości pomiaru, ustawianie alarmów itp. W przykładzie nie będziemy wykorzystywać alarmów, więc konieczna będzie znajomość tylko dwóch rejestrów: ***ATR*** umożliwiającego odczyt aktualnej temperatury (0x00h) oraz ***CONFIG*** (0x01), który umożliwia ustalenie rozdzielczości pomiaru temperatury. Na **rysunku 8** zamieszczono strukturę rejestru ***CONFIG*** (0x01h), natomiast na **rysunku 9** 16-bitowego rejestru ATR (0x00h) zawierającego wynik pomiaru temperatury.



Rysunek 7. Schemat dołączenia modułu KAMOD-TEM do płytki prototypowej

Starsza połowa rejestru *ATR* zawiera wartość całkowitą temperatury wyrażoną w stopniach Celsjusza, a młodsza ułamkową. Po włączeniu czujnik jest skonfigurowany w trybie 9-bitowym. Aby uzyskać większą rozdzielczość, należy odpowiednio ustawić bity ***resolution*** w rejestrze ***CONFIG***.

Opis aplikacji

Aby zademonstrować przykład użycia interfejsu I^{2}C, napisano program *i2ctemp*, którego zadaniem jest cykliczne odczytywanie aktualnej temperatury z modułu KAMOD-TEM oraz prezentacja wyniku na standardowym wyjściu (terminal). Program rozpoczyna się od pokazanej na **listingu 14** funkcji *main()*. Na jej początku jest

wywoływana funkcja *i2c_bus_open()*, która jako argument przyjmuje numer porządkowy interfejsu I²C, a zwraca uchwyt do pliku reprezentującego daną magistralę**.** Kod funkcji zamieszczono na **listingu 15**.

Działanie tej funkcji sprowadza się do utworzenia w buforze nazwy pliku odpowiadającego interfejsowi (w naszym przypadku */dev/i2c-0*), a następnie otwarcia pliku reprezentującego konkretny interfejs I²C do odczytu i zapisu za pomocą wywołania systemowego *open()*. Następnie jest wywoływana funkcja **[2]**, której zadaniem jest ustalenie adresu sprzętowego, którego będą dotyczyć wywołania systemowe *read(), write()*. W **[3]** za pomocą wywołania systemowego *write()*, do rejestru ***CONFIG*** układu **MCP2301** jest wpisywana wartość *#define MCP9800_RES_12BIT (3<<5)*, co powoduje ustawienie 10-bitowej rozdzielczości pomiaru temperatury. Po zakończeniu inicjalizacji program wchodzi do pętli nieskończonej, w której dzięki wywołaniu funkcji ***tempsensor_get()*** [4] jest odczytywana temperatura z czujnika **MCP2301**, a następnie w **[5]** jest ona wyświetlana na standardowym wyjściu (konsola).

Na **listingu 16** zamieszczono funkcję odpowiedzialną za odczyt temperatury z czujnika. Aby odczytać temperaturę, należy odczytać liczbę 16-bitową z rejestru *ATR*, co jest wykonywane za pomocą sekwencji *zapis/odczyt* bez generowania bitu stopu. Aby wygenerować taką sekwencję, należy użyć wywołania systemowego *ioctl(handle, I2C_RDWR)* oraz zdefiniować strukturę, która w pojedynczej transakcji spowoduje wysłanie bajtu danych z adresem rejestru ***ATR***, a następnie, po wygenerowaniu ponownego bitu startu, odczyta 16 bitów danych będących zawartością rejestru ATR. Jest to realizowane przez utworzenie tablicy składającej się z dwóch struktur ***i2c_msgs*** reprezentujących transakcje zapisu pojedynczego bajtu oraz odczytu dwóch bajtów. Następnie adres tablicy *temp_msgs* oraz liczba jej elementów przekazywana jest do struktury *i2c_rdwr_ioctl_data*, która z kolei jest przekazywana jako argument wywołania *ioctl(handle,I2C_RDWR, msg);* Wywołanie tej funkcji spowoduje wysłanie odpowiedniej sekwencji przez magistralę I²C bez generowania bitu stopu pomiędzy zapisem a odczytem.

Uruchomienie przykładu na BF210

Skompilowany przykład wraz z kompletnym systemem Linux jest dostępny w postaci obrazu *example3.img*. Przykład należy nagrać na kartę SD, tak jak to zostało opisane w pierwszym odcinku. Po nagraniu karty, włożeniu jej do **BF210** oraz uruchomieniu systemu, należy zalogować się do konsoli jako *root,* a następnie uruchomić przykład, wpisując polecenie ***i2ctemp***, co spowoduje uruchomienie programu. Jeżeli moduł KAMOD-TEM jest dołączony, to na terminalu (konsoli) powinna być widoczna aktualna temperatura aktualizowana co 1 s. Zakończenie działania programu jest możliwe przez wciśnięcie kombinacji klawiszy CTRL+C.

Lucjan Bryndza, EP

Bit 7							Bit 0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
One-Shot	Resolution		Fault Queue		ALERT Polarity	COMP/INT	Shutdown

bit 7: 1 – tryb jednokrotnego pomiaru, 0 – tryb ciągłego pomiaru (domyślny)
bity 6–5: rozdzielczość pomiaru: 00b – 9 bit (0,5°C) 01b – 10 bit (0,25°C) 10b – 11 bit (0,125°C) 11b – 12 bit (°C)
bity 4–3: Określa, ile przekroczeń pomiarów wyzwala alarm (dla nas nieistotne)
bit 2: Określa polaryzację linii alarmu (dla nas nieistotne)
bit 1: Określa tryb pracy linii wyjściowej alarmu (dla nas nieistotne)
bit 0: 1 – układ uśpiony, 0 – układ aktywny

Rysunek 8. Opis bitów rejestru *CONFIG* (0x1h)

Bit 15							Bit 8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
Sign	2^6 [°C] 2^5 [°C]		2^4 [°C] 2^3 [°C]		2^2 [°C]	2^1 [°C]	2^0 [°C]

Bit 7							Bit 0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
2^{-1} [°C/bit]	2^{-2} [°C]	2^{-3} [°C]	2^{-4} [°C]	0	0	0	0

Rysunek 9. Opis bitów rejestru *ATR* (0x0h)

Listing 14. Program główny

```
int main(void)
{
  //Initialize i2c bus
  int handle = i2c_bus_open( 0 ); [1]
  error(handle<0);
  //Set destination device address
  int err = ioctl(handle, I2C_SLAVE, MCP_SLAVE_ADDR); [2]
  error(err<0);
  //Write init string to temp sensor
  static const char mcp_init[] = { MCP9800_CONFIG_REG , MCP9800_RES_12BIT };
  err = write( handle, mcp_init, sizeof(mcp_init) ); [3]
  error(err!=sizeof(mcp_init));
  //Main loop for read sensor temp
  for(;;)
  {
      //Read the temp
      float temp = tempsensor_get(handle); [4]
      //Print temp
      printf("%.1f     \r",temp);  [5]
      fflush(stdout);
      //Sleep one sec
      sleep(1);           [6]
  }
  //Close handle
  close(handle);
  return 0;
}
```

Listing 15. Funkcja i2c_bus_open

```
static int i2c_bus_open(int i2cbus)
{
   char filename[65];
   snprintf(filename, sizeof(filename), "/dev/i2c-%d", i2cbus);
   return open(filename, O_RDWR);
}
```

Listing 16. Odczyt temperatury z czujnika

```
float tempsensor_get(int handle)
{
  unsigned char tempreg_addr = MCP9800_TEMP_REG;  //Temp sensor rego
  unsigned char temp_regs[2];
  struct i2c_msg temp_msgs[] =  //I2c msgs
  {
    {
      MCP_SLAVE_ADDR,            //Slave addres
      0,                         //Flags
      sizeof(tempreg_addr),      //transfer sizze
      &tempreg_addr              //Reg address
    },
    {
      MCP_SLAVE_ADDR,            //Slave adddres
      I2C_M_RD,                  //Read flag
      sizeof(temp_regs),         //transfer size
      temp_regs                  //Reg address
    }
  };
  struct i2c_rdwr_ioctl_data msg =
  {
    temp_msgs,                   //Address of messages
    sizeof(temp_msgs)/sizeof(struct i2c_msg)  //Number of messages
  };
  int err =  ioctl(handle, I2C_RDWR, &msg);   //WRWR ioctl
  error(err<0);   //Check error
  return (float)((signed char)temp_regs[0])+ (temp_regs[1]>>4)/16.0f;
}
```