Wprowadzenie do Linux'a embedded (2)

Klawiatura matrycowa, podsystem input, zaawansowana obsługa wejścia – wyjścia

W poprzednim odcinku zapoznaliśmy się z obsługą portów GPIO w Linuksie za pomocą aplikacji użytkownika wykorzystując interfejs sysfs-gpio. Sprawdzenia stanu wciśnięcia klawisza dokonywaliśmy poprzez cyklicznie odczytywanie stanu, co powodowało, że proces zupełnie niepotrzebnie był co chwilę aktywowany, w celu wykrycia zbocza na linii wejściowej. W środowiskach wielozadaniowych powinniśmy unikać cyklicznego odpytywania, aby procesor był dostępny dla innych procesów. Poza tym, jak wspomniano poprzednio, porty GPIO nie są same w sobie funkcjonalnym urządzeniem, służą one jedynie do podłączenia innych układów np. diod LED itp. W bieżącym odcinku pokażemy zatem jak w jaki sposób obsłużyć klawiaturę, oraz diody LED zgodnie z filozofią jądra Linuksa.

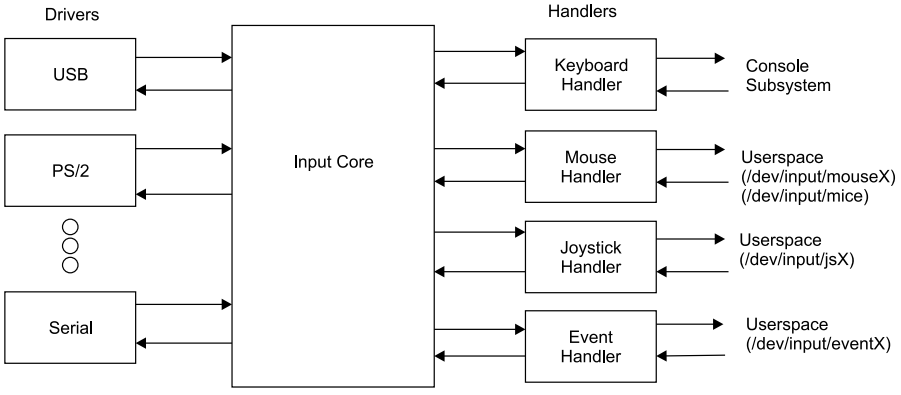
Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 12147, pass: 2e7u6a2a
• pierwsza część kursu

Aby urozmaić przykład zamiast pojedynczego przycisku podłączymy klawiaturę matrycową w roli której wykorzystamy moduł KAMODKBD4x4 firmy Kamami. Klawiaturę tą podłączymy do podsystemu Input jądra, za pomocą odpowiedniego sterownika, natomiast do sterowania diodami LED użyjemy interfejsu leds, który jest przeznaczony do sterowania diodami LED.

Zagadnienia teoretyczne

Podsystem wejścia (*Linux Input Subsystem*). Podsystem wejścia jądra systemu linuks, zapewnia jednolity interfejs dla obsługi wszelkich urządzeń wejściowych, takich jak klawiatury, myszki, joysticki, oraz inne urządzenia. Zapewnia on obsługę urządzeń z wykorzystaniem, zdarzeń. Zapewnia on również sterowanie, np. diodami LED klawiatury, wibracjami joysticka itp. Mechanizm dostarcza warstwę abstrakcji uniezależniającą aplikację użytkownika od sprzętu, magistrali do której zostało urządzenie podłączone. np. USB, PS2, GPIO itp. Schemat blokowy podsystemu wejścia w jądrze Linuksa przedstawiono na **rysunku 4** zaczerpniętym z czasopisma „The Linux Journal" (*http://www.linuxjournal.com*).

Interfejs ten składa się z jądra podsystemu (Input core), które od strony niskopoziomowej komunikuje się ze sterownikami urządzeń, które są odpowiedzialne za bezpośrednią komunikację z urządzeniami. Natomiast od strony aplikacji użytkownika zapewnia jednolity interfejs. Od strony aplikacji dostęp jest możliwy za pomocą kilku interfejsów które stanowią: intefejs klawiatury, Interfejs myszy, Interfejs Joysticka oraz Interfejs zdarzeń. Dostęp do poszczególnych interfejsów aplikacjom zapewniają pliki urządzeń. (Jak mawia stare powiedzenie w linuksie wszystko jest plikiem). Interfejs klawiatury myszy i joysticka jest dedykowany obsłudze wyżej wspomnianych urządzeń i jego zastosowanie nie wymaga dodatkowego komentarza. Interfejs Event służy do podłączania innych rodzajów przycisków czy klawiszy, (np. przycisk zamknięcia obudowy laptopa, przyciski głośności itp.), i umożliwia generowanie zdarzeń, możliwych do odczytania z aplikacji użytkownika. Może się wydawać, że interfejs event, a interfejs klawiatury to w zasadzie to samo, jednak różnica jest taką, że interfejs klawiatury mapuje bezpośrednio klawiaturę do konsoli, Interfejs event z kolei umożliwia odczytywanie stanu pozostałych klawiszy istotnych dla danej platformy.



Rysunek 4. Schemat blokowy podsystemu wejścia w jądrze Linuksa

Lisitng 1. Struktura służąca do obsługi klawiatury
```
struct matrix_keypad_platform_data {
  const struct matrix_keymap_data *keymap_data;

  const unsigned int *row_gpios;
  const unsigned int *col_gpios;
  unsigned int    num_row_gpios;
  unsigned int    num_col_gpios;
  unsigned int    col_scan_delay_us;
  /* key debounce interval in milli-second */
  unsigned int    debounce_ms;
  unsigned int    clustered_irq;
  unsigned int    clustered_irq_flags;
  bool        active_low;
  bool        wakeup;
  bool        no_autorepeat;
};
```

Dane platform device, umożliwiające określenie konfiguracji maszyny. Klasyczne komputery PC mają ściśle określoną architekturę oraz podsystem BIOS umożliwiający, określenie urządzeń zewnętrznych i wewnętrznych będących częścią komputera. Aby dowiedzieć się w jakie urządzenia wyposażony jest komputer, wystarczy odpytać BIOS, lub urządzenia PnP (Plug and Play). W przypadku innych architektur, a szczególnie w przypadku urządzeń wbudowanych, mamy do czynienia z wieloma różnymi konstrukcjami sprzętowymi, zawierającymi różne komponenty i podzespoły. Najczęściej nie istnieje również żaden jednolity interfejs stanowiący funkcjonalny odpowiednik systemu BIOS, który umożliwia zapytanie o listę dostępnego sprzętu. Jedynym zatem sposobem sprawdzenia, listy dostępnych komponentów danej platformy jest zdefiniowanie listy podzespołów bezpośrednio w kodzie. W linuksie kod zdefiniowany jest w postaci struktur danych *platform_data* zawierających listę podzespołów danej maszyny. Lista ta definiowana jest w jądrze w pliku odpowiedzialnym za konfigurację danej maszyny. W przypadku **BF210** plikiem odpowiedzialnym za inicjalizację jest plik w kodzie jądra znajdujący się w następującej lokalizacji: *arch/arm/mach-at91/board-boff210.c* Lista uradzeń dostępnych na danej platformie zorganizowana jest w postaci definicji struktur *struct platform_device*, które następnie są inicjalizowane za pomocą funkcji *platform_device_register()*...

Sterownik klawiatury matrycowej matrix-keypad. Aby urządzenie było widoczne przez podsystem wejścia, musi posiadać sterownik dla podsystemu. Dołączenie odpowiedniego urządzenia wymaga zatem napisania sterownika jądra, co może być zadaniem stosunkowo skomplikowanym dla początkującego użytkownika. W głowach czytelników rodzi się pytanie, czy aby dołączyć tytułową klawiaturę KAMOD4x4 będziemy musieli się zmagać z napisaniem odpowiedniego sterownika dla podsystemu input? Wiadomość jest bardzo optymistyczna i brzmi NIE. Dzięki warstwie abstrakcji obsługi portów GPIO istniejącym w kernelu, oraz danych konfiguracyjnych, dla danej platformy (*platform data*), istnieją gotowe sterowniki dla większości typowych urządzeń dołączanych do portów GPIO. W poprzednim odcinku wspominaliśmy o sterowniku gpio_buttons dla pojedynczych przycisków. W przypadku klawiatury matrycowej odpowiednim sterownikiem będzie *matrix_keypad*, a jedyną rzeczą jaką będziemy musieli wykonać to zdefiniować kilka struktur oraz zarejestrować opis urządzenia w odpowiednich strukturach przechowujących konfigurację, za pomocą funkcji *platform_register_device()*, w pliku inicjalizującym daną płytkę.

Sterowanie klawiatura matrycową odbywa się za pomocą przemiatania poprzez wystawienie odpowiednich stanów na linii wierszy (ROW), oraz sprawdzania stanów na liniach kolumn (COL). Przy czym sprawdzanie stanów może się odbywać za pomocą aktywnego odpytywania, lub za pomocą przerwań. Sterownik **matrix-keypad**, został napisany, z wykorzystaniem tego drugiego sposobu, i do prawidłowej pracy wymaga linii **GPIO,** które potrafią zgłaszać przerwania. Na szczęście porty **GPIO** naszego mikrokontrolera RM9200 posiadają taką możliwość, zatem bez problemu możemy wykorzystać sterownik **matrix_keypad**. Aby sterownik mógł działać poprawnie musimy w strukturach przechowujących konfigurację maszyny, określić rodzaj klawiatury, oraz sposób jej dołączenia co jest realizowane za pomocą struktury z **listingu 1**.

Pole **keymap_data** wskazuje na kolejną strukturę **matrix_keymap_data**, która zawiera wskaźnik do tablicy definiującej przypisania kodów klawiszy do poszczególnych wierszy i kolumn. Pole **row_gpios**, wskazuje na tablicę zawierającą numery porządkowe portów GPIO (poprzedni artykuł), określające fizyczne linie GPIO, które będą użyte w roli linii wierszy, podobnie pole **col_gpios**, określa linie GPIO, które będą użyte w roli linii kolumn. Pola **num_row_gpios** oraz **num_col_gpios**, określają ilość linii kolumn, oraz linii wierszy klawiatury matrycowej. Pole **col_scan_delay_us** określa, czas opóźnienia pomiędzy ustawieniem linii stanu linii wierszy, a odczytywaniem stanu linii kolumn. Pole **debounce_ms** określa czas potrzebny do wyeliminowania drgań zestyków. Pole **active_low** określa, czy stanem aktywnym jest sygnał 1 czy 0. Pole **wakeup**, określa czy klawiatura umożliwia wybudzanie systemu ze stanu uśpienia. Pole **no_autorepeat**, określa że klawiatura nie generuje kodów powtórzeń podczas ciągłego trzymania klawisza, a jedynie eventy wciśnięcia oraz zwolnienia klawisza. Wypełnienie struktury **matrix_keypad_platform**, zapewnia sterownikowi **matrix-keypad**, wszystkie niezbędne dane potrzebne do działania, oraz informuje sterownik o obecności klawiatury matrycowej dołączonej do portów GPIO.

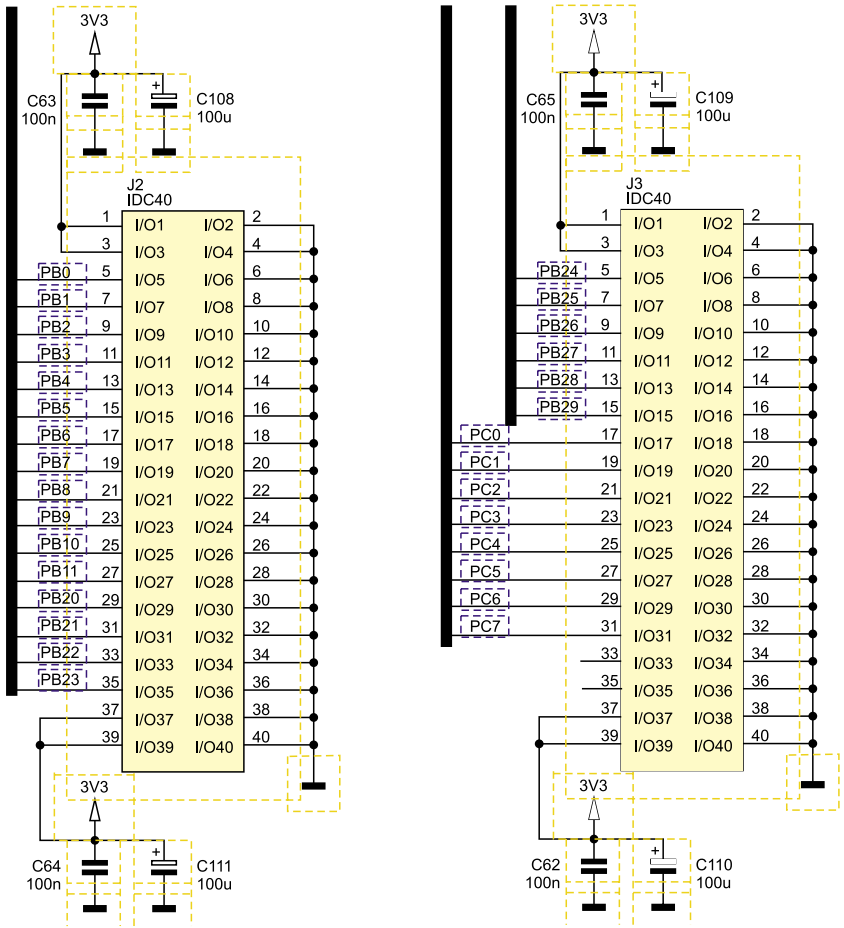
Dostęp do klawiatury z aplikacji użytkownika, z wykorzystaniem interfejsu event. Dostęp do zdarzeń podsystemu input realizowany jest za pomocą pliku /dev/input/eventX (gdzie X określa kolejny numer urządzenia). Po otwarciu urządzenia dane o zdarzeniach od urządzenia, będzie można odczytywać za pomocą wywołania systemowego read(). Ponieważ domyślnie funkcja read(), jest blokująca (usypiająca proces), wywołanie funkcji read spowoduje uśpienie wywołującego procesu do momentu wystąpienia zdarzenia od podsystemu input. Zdarzenia zostały zdefiniowane w postaci struktury umieszczonej na **listingu 2**.

Pole **timeval**, zawiera znacznik czasu wystąpienia zdarzenia, pole **type** określa typ zdarzenia, w przypadku klawiatury, będą to zdarzenia typu **EV_KEY**, natomiast pole **value**, określa wartość dla danego zdarzenia. Dla zdarzenia **EV_KEY**, wciśnięcie klawisza będzie sygnalizowane zwróceniem wartości **1** w polu **value**, zwolnienie klawisza sygnalizowane będzie zwróceniem wartości **0**, natomiast wciśnięcie i przytrzymanie klawisza będzie powodowało generowanie cyklicznych zdarzeń powtórzonych, które będą sygnalizowane wartością **2.** Pole **code** zawiera kod klawisza który został wciśnięty. Zatem odczytując zdarzenia za pomocą funkcji *read()*, spowodujemy uśpienie wywołującego procesu do momentu wystąpienia zdarzenia.

Przykład praktyczny KAMOD4x4 ze sterownikiem matrix-keypad. Diody LED sterowane przez leds-gpio

Wstęp, schemat ideowy. Po zapoznaniu się z częścią teoretyczną przejdziemy do prezentacji, przykładu w którym wykorzystamy klawiaturę matrycowa *KAMODKBD4x4*,oraz minimoduł *KAMODLED8* produkcji *KAMAMI*. Będzie to bardzo prosty przykład, w którym wciśnięcie i trzymanie klawisza, będzie zapalało diody **LED** *D0-D3*, odpowiadające wciśniętemu klawiszowi, natomiast zwolnienie klawisza powodować będzie wyłączenie wszystkich diod. Klawiatura matrycowa będzie dołączona do podsystemu input, z wykorzystaniem sterownika **matrix-keypad**, co umożliwi nam całkowicie pasywne sprawdzanie stanu klawiszy. Diody LED będą dołączone do podsystemu *LED* za pomocą sterownika **leds-gpio**. Aplikacja przestrzeni użytkownika będzie używać interfejsu *evdev*, do odczytywania stanu klawia-

Listing 2. Struktura definiująca zdarzenia
```
struct input_event {
  struct timeval time;
  __u16 type;
  __u16 code;
  __s32 value;
};
```




```
KAMODLED8: (+5V - J2PIN1, GND - J2PIN2,  D0 - J2PIN7, D1 - J2PIN9,
D2 - J2PIN17, D3 - J2PIN19, D4-J2PIN4, D5-J2PIN6, D6-J2PIN8, D7-J2PIN10)
KAMODKB4x4: ( ROW1 - J3PIN9, ROW2 - J3PIN15, ROW3- J3PIN27, ROW4-J3PIN29,
COL1-J3PIN17, COL2-J3PIN19, COL3-J3PIN21, COL4-J3PIN23)
```

Rysunek 5. Schemat ideowy dołączenia modułów do płyty prototypowej BF210

tury. Schemat ideowy dołączenia modułów do płyty prototypowej BF210 przedstawiono na **rysunku 5**.

Linie LED-ów zostały dołączone odpowiednio do linii portów PB1, PB2, PB6, PB7, natomiast linie wierszy i kolumn odpowiednio do linii PB26, PB29, PC5, PC6, PC0, PC1, PC2, PC3. Dzięki możliwości włączenia wbudowanych w linie GPIO rezystorów podciągających, nie będą potrzebne dodatkowe rezystory, a połączenie wszystkich modułów możemy dokonać np. za pomocą przewodów CAB, do nabycia w firmie Kamami.

Konfigurowanie jądra. Jak już wcześniej wspomniano, lista dostępnych urządzeń na danej maszynie determinowana jest za pomocą struktur **platform-device.** BF210 jest płytą prototypową zatem z uwagi na niemożliwość określenia przeznaczenia a zatem przypisania linii GPIO, nie jest możliwe dostarczenie gotowego binarnego jądra, które nadaje się do wykorzystania we wszystkich projektach. W związku z tym w przypadku dołączenia urządzeń które nie są Plug&Play (u nas tylko USB), i wykorzystaniu sterowników jądra, będziemy zmuszeni do edycji kodu źródłowego jądra, a konkretnie pliku odpowiedzialnego za konfigurację maszyny, oraz rekompilację jądra. W przypadku płyty prototypowej BF210, kod linuxa jest zdefiniowany w pliku *arch/arm/mach-at91/board-boff210.c*. Na **listingu 3** przestawiono fragment struktur, odpowiedzialnych za informację, o dołączeniu klawiatury matrycowej, oraz diod LED.

Na początku tworzona jest tablica, zawierająca przypisanie poszczególnych wierszy i kolumn, do odpowiedniego kodu klawiszy. Następnie na podstawie tablicy mapy tworzona jest struktura **matrix_keypad_data**, do której przypisujemy mapę stworzonych klawiszy. Mapa ta określa kod klawisza, zwrócony w wyniku wykrycia wciśnięcia na przecięciu odpowiedniej kolumny i odpowiedniego wiersza. Następnie tworzone są dwie tablice zawierające mapę przypisań linii **GPIO**, do wierszy, oraz kolumn. Następnie na bazie wcześniej zdefiniowanych struktur tworzona jest struktura **matrix_keypad_platform_data** stanowiąca informację dla sterownika **matrix-keypad**. Pole **keymap_data**, zawiera przypisanie wcześniej utworzonej konfiguracji klawiszy. Pola **row_gpios**, **num_row_gpios** zawierają odpowiednio przypisania mapy linii GPIO, które stanowią linie kolumn, oraz ilość linii. Podobnie sytuacja wygląda w przypadku przypisania linii GPIO wierszy. Pole **con_scan_delay_us**, określa opóźnienie pomiędzy wystawieniem stanu na linii wierszy, a odczytaniem stanu kolumn, które zostało zdefiniowane na 10 us. Pole **debounce_time**, zawiera opóźnienie eliminujące drgania zestyków, które zostało ustawione na 10 ms. Pole **active_low** informuje że skanowanie wierszy i kolumn odbywa się za pomocą stanu niskiego, dzięki czemu możliwe jest wykorzystanie wewnętrznych rezystorów podciągających. Pole **wake_up=1**, określa że możliwe jest wybudzanie urządzenia ze stanu uśpienia w wyniku wciśnięcia klawisza. Tak przygotowana struktura informacyjna dla sterownika *matrix_keypad*, jest następnie przypisywana do struktury **platform_device**, która następnie jest rejestrowana w systemie w funkcji inicjalizującej maszyne **bf_board_init()** za pomocą wywołania p**latform_device_register(&keypad_device)**

Tablica struktur **gpio_leds** zdefiniowana jako LEDS zawiera informację o konfiguracji diod LED, i sprowadza się ona do przypisania nazwy poszczególnych diod LED w polu **name** , oraz przypisania odpowiadającej jej linii GPIO. Nazwy te następnie będą widoczne w przestrzeni użytkownika w katalogu **/sys/class/led.**

Zaprezentowany przykład jest dostępny w postaci kompletnego obrazu karty, przygotowanego do testów, jednak w przypadku samodzielnego konfigurowania urządzeń, konieczna będzie modyfikacja kodu jądra Linuksa, chociaż w minimalnie opisanym zakresie, dlatego w dalszej części tekstu opiszemy w jaki sposób dokonywać zmian we własnym zakresie. Mniej dociekliwi czytelnicy mogą pominąć niniejszy fragment.

W świecie Open Source, zmiany w kodzie źródłowym dla istniejącego kodu dołączane są w postaci plików patch. Są to proste pliki tekstowe, które zawierają różnice pomiędzy oryginalnym, a zmodyfikowanym kodem, obsługiwane przez program **patch**. Plik l*eds-keypad.patch*, zawiera patch dla jądra linuxa, zawierający wcześniej opisany fragment kodu, konfigurującego klawiaturę oraz diody. Ponieważ do kompilacji jądra nie są potrzebne żadne biblioteki zewnętrzne możemy w zasadzie wykorzystać dowolną wersję kompilatora **gcc,** która zawiera obsługę **EABI**. Na przykład w Ubuntu 10.10, wystarczy zainstalować pakiet *gcc-4.5-arm-linux-gnueabi* (polecenie *sudo apt-get install gcc-4.5-arm-linux-gnueabi*) Następnie aby skompilować przykład należy ściągnąć źródła jądra 2.6.37 ze strony *http://www.kernel.org/* rozpakować je, a następnie zastosować patch dla maszyny BF210 który jest dostępny tutaj *http://www.boff.pl/content/files/id5/2.6.37-boff-bf210.patch* (p*olecenie patch -p1 -d linux-2.6.37 < 2.6.37-boff-210.patch*). Kolejną czynnością jest zastosowanie patcha, który dodaje opisane wcześniej struktury *platform-data*, za pomocą polecenia: *patch -p1 -d linux-2.6.37 < leds_keypad.patch*. Gdy już mamy odpowiednio skonfigurowane źródła możemy przystąpić konfiguracji jądra. W tym celu do katalogu *linux-2.6.37*, należy skopio-

wać konfigurację jądra **ex2_config** do pliku **.config** w katalogu *linux-2.6.37* (*polecenie cp ex2_config linux-2.6.37/.config*) . Następnie należy zmienić bieżącą ścieżkę aby znaleźć się w katalogu *linux 2.6.37,* a następnie wywołać polecenie *make oldconfig ARCH=arm*, co spowoduje skonfigurowanie jądra według załączonego pliku .config. Po skonfigurowaniu jądra możemy przystąpić do jego kompilacji za pomocą polecenia: *make uImage ARCH=arm*. Po dłużej chwili w katalogu *uImage* pojawi się gotowy plik binarny z jądrem. Tak przygotowane jądro możemy wgrać do katalogu */boot* na karcie docelowej. Kolejną czynnością jest kompilacja modułów jądra, która odbywa się z wykorzystaniem polecenia *make modules ARCH=arm*. Podobnie moduły jądra należy wgrać na kartę docelową np. poprzez zamotowanie karty w katalogu */mnt*, a następnie wydanie polecenia make *modules_install INSTALL_MOD_PATH=/mnt/*

Listing 3. Fragmenty struktur odpowiedzialnych za informację o dołączeniu klawiatury matrycowej oraz diod LED

```
static const uint32_t keymap[] =
{
  KEY(0, 0, KEY_1),
  KEY(0, 1, KEY_4),
  KEY(0, 2, KEY_7),
  KEY(0, 3, KEY_ENTER),
  KEY(1, 0, KEY_2),
  KEY(1, 1, KEY_5),
  KEY(1, 2, KEY_8),
  KEY(1, 3, KEY_0),
  KEY(2, 0, KEY_3),
  KEY(2, 1, KEY_6),
  KEY(2, 2, KEY_9),
  KEY(2, 3, KEY_ESC),
  KEY(3, 0, KEY_A),
  KEY(3, 1, KEY_B),
  KEY(3, 2, KEY_C),
  KEY(3, 3, KEY_D),
};

static const struct matrix_keymap_data keymap_data =
{
  .keymap = keymap,
  .keymap_size = ARRAY_SIZE(keymap)
};

static const uint32_t row_gpios[] =
{
  AT91_PIN_PC0, AT91_PIN_PC1, AT91_PIN_PC2, AT91_PIN_PC3
};

//PB26 PB29 PC5 PC6
static const uint32_t col_gpios[] =
{
  AT91_PIN_PB26, AT91_PIN_PB29, AT91_PIN_PC5, AT91_PIN_PC6
};

static struct matrix_keypad_platform_data keypad_config =
{
  .keymap_data    = &keymap_data,
  .row_gpios = row_gpios,
  .num_row_gpios  = ARRAY_SIZE(row_gpios),
  .col_gpios = col_gpios,
  .num_col_gpios  = ARRAY_SIZE(col_gpios),
  .col_scan_delay_us = 10,
  .debounce_ms = 10,
  .active_low = 1, /* pull up realization */
  .no_autorepeat  = 0,
  .wakeup = 1
};

static struct platform_device keypad_device =
{
  .name = "matrix-keypad",
  .id = -1,
  .dev =
  {
    .platform_data  = &keypad_config,
  },
};

static struct gpio_led leds[] =
{
  {
    .name = "led0",
    .gpio = AT91_PIN_PB1
  },
  {
    .name = "led1",
    .gpio = AT91_PIN_PB2
  },
  {
    .name = "led2",
    .gpio = AT91_PIN_PB6
  },
  {
    .name = "led3",
    .gpio = AT91_PIN_PB7
  }
};
```

Listing 4. Funkcja led_ctl

```
//Led enable disable
static void led_ctl( int no, bool value )
{
  char name_buf[128];
  //Create led by name
  snprintf(name_buf,sizeof(name_buf),"/sys/class/leds/led%i/brightness",no);
  int led_fd = open(name_buf, O_WRONLY);  //Open the led file
  error( led_fd );
  int led_wr = write( led_fd, value?"1":"0",1);  //Write the led value
  error( led_wr );
  close( led_fd );  //Close the led handle
}
```

Aplikacja przestrzeni użytkownika matrixkbd. Aby zademonstrować użycie klawiatury matrycowej, oraz diod LED z wykorzystaniem podsystemów *input*, oraz *led*, napisano bardzo prostą aplikację przestrzeni użytkownika, która w momencie wciśnięcia klawisza, będzie zapalała diody **LED** przypisane do danego klawisza, oraz powodowała wyłączenie diod w momencie puszczenia klawisza. Jednocześnie na standardowym wyjściu (konsoli) wypisywane będą komunikaty informujące o otrzymywanych zdarzeniach klawiatury typu **EV_KEY**. (Kod źródłowy aplikacji znajduje się w pliku *matrixkbd.c*) Klawiatura widoczna będzie jako plik */dev/input/event0*, natomiast diody **D0.. D3**, widoczne będą jako katalogi */sys/class/ledX (gdzie X=0-3)*. Włączanie oraz wyłączanie diod LED realizowane jest za pomocą funkcji *led_ctl* (**listing 4**).

Funkcja ta jako argument *no* przyjmuje numer porządkowy diody (0–3), określająca nr diody, natomiast jako argument *value*, przyjmuje wartość oznaczająca czy ma być ona włączona czy wyłączona. Jej działanie jest bardzo proste i sprowadza się do wpisania za pomocą funkcji write() do pliku *brightness* odpowiedniej diody, wartości 0 lub 1.

Program zaczyna wykonywanie od funkcji *main* w której realizowany jest odczyt stanu kodów klawiszy (**listing 5**).

Na początku za pomocą wywołania systemowego **open()**, jest otwierane urządzenie */dev/input/event0* reprezentujące klawiaturę. Następnie za pomocą funkcji **ioctl()** pobierana jest nazwa klawiatury, co może być użyteczne w przypadku gdy mamy do czynienia z większą ilością urządzeń wejściowych. (W naszym przypadku mamy jedno urządzenie, więc po prostu otwieramy */dev/input/event0*), a następnie wypisujemy nazwę jaka została przypisana urządzeniu klawiatury matrycowej. Następnie program wchodzi do pętli nieskończonej, gdzie zdarzenia od klawiatury odczytywane są za pomocą wywołania systemowego **read()**. Wywołanie to spowoduje uśpienie procesu, do momentu wystąpienia zdarzenia, na interfejsie **event0**. W przypadku wystąpienia zdarzenia tablica struktur events zostanie wypełniona. Następnie w pętli *for()* odczytywane są wszystkie zdarzenia. W przypadku

Listing 5. Odczyt stanu kodów klawiszy

```
//Main loop
int main(void)
{
  //Input event device0
  int input_fd = open("/dev/input/event0",O_RDONLY);
  error(input_fd);
  //Get device name
  {
    char buf[128];
    int ioctl_ret = ioctl(input_fd,EVIOCGNAME(sizeof(buf)),buf);
    error(ioctl_ret);
    printf("Input device name: %s\n",buf);
  }
  //Handle input events
  struct input_event event[MAX_KEY_MSGS];
  for(;;)
  {
    int input_rd = read(input_fd,event,sizeof(event));
    error(input_rd);
    for (size_t i=0; i< input_rd/sizeof (struct input_event);i++)
    if( event[i].type == EV_KEY )  //Interested only with key event
    on_key(&event[i]);
  }
  close( input_fd );
  return 0;
}
```

Listing 6. Funkcja on_key obsługująca zdarzenie naciśnięcia klawisza

```
//Handle input event
static void on_key(const struct input_event *ev)
{
  printf( "Got key scancode %d value %s\n",ev->code, ch_keyval(ev->value) );
  if( ev->value == KEYV_PRESS)          //MAP keyboyard scancodes to key
  switch( ev-> code)
  {
    case KEY_1: led_bitmask(1); break;
    case KEY_2: led_bitmask(2); break;
    case KEY_3: led_bitmask(3); break;
    case KEY_4: led_bitmask(4); break;
    case KEY_5: led_bitmask(5); break;
    case KEY_6: led_bitmask(6); break;
    case KEY_7: led_bitmask(7); break;
    case KEY_8: led_bitmask(8); break;
    case KEY_9: led_bitmask(9); break;
    case KEY_0: led_bitmask(10); break;
    case KEY_A: led_bitmask(11); break;
    case KEY_B:  led_bitmask(12); break;
    case KEY_C:  led_bitmask(13); break;
    case KEY_D: led_bitmask(14); break;
    case KEY_ENTER: led_bitmask(15); break;
    default: led_bitmask(0); break;
  }
  else if( ev->value == KEYV_RELEASE) led_bitmask(0);
}
```

otrzymania zdarzenia typu **EV_KEY**, wywoływana jest funkcja **on_key()**, która jako argument przyjmuje wskaźnik na strukturę **input_event** reprezentującą zdarzenie (**listing 6**).

Funkcja ta na początku wypisuje kod wciśniętego klawisza, oraz jego stan (wciśnięty, zwolniony, autopotarzanie). W przypadku gdy mamy do czynienia z wciśnięciem klawisza, w zależności od numeru klawisza wywoływana jest funkcja **led_bitmask()**, która powoduje włączenie odpowiednich diod LED przypisanych do danego klawisza. W przypadku, wystąpienia zdarzenia od zwolnienia klawisza, wywoływana jest funkcja *led_bitmask(0)*, co powoduje wyłączenie wszystkich diod LED.

Uruchomienie przykładu na BF210. Skompilowany przykład, wraz z kompletnym

Listing 7. Przykładowe komunikaty wyświetlane przez konsolę Linux

```
root@bf210-at91:~# matrixkbd
Input device name: matrix-keypad
Got key scancode 10 value PRESSED
Got key scancode 10 value RELEASED
Got key scancode 10 value PRESSED
Got key scancode 10 value RELEASED
Got key scancode 48 value PRESSED
Got key scancode 48 value RELEASED
Got key scancode 30 value PRESSED
Got key scancode 30 value RELEASED
Got key scancode 4 value PRESSED
Got key scancode 4 value RELEASED
Got key scancode 3 value PRESSED
Got key scancode 3 value RELEASED
Got key scancode 9 value PRESSED
Got key scancode 9 value RELEASED
Got key scancode 48 value PRESSED
Got key scancode 48 value REPEAT
Got key scancode 48 value REPEAT
Got key scancode 48 value REPEAT
Got key scancode 48 value REPEAT
Got key scancode 48 value REPEAT
Got key scancode 48 value RELEASED
```

systemem Linux, oraz jądrem dostosowanym do potrzeb bieżącego odcinka jest dostarczany w postaci pliku obrazu *example2.img*. Przykład można nagrać na czystą kartę SD, w sposób analogiczny jak opisano w poprzednim artykule, a następnie tak nagraną kartę umieścić w złączu BF210 i włączyć napięcie zasilające. Po uruchomieniu linuksa należy zalogować się do konsoli jako root, a następnie uruchomić program wpisując polecenie **matrixkbd**. Teraz wciskając i zwalniając klawisze klawiatury matrycowej, będziemy mogli obserwować zapalanie odpowiednich diod LED w minimodule **KAMODLED8**, oraz obserwować komunikaty w konsoli Linux'owej, na której jest uruchomiony program. Przykładowy fragment komunikatów umieszczono na **listingu 7**.

W komunikatach aplikacji możemy zobaczyć początkową nazwę urządzenia wejściowego, które zostało ustalone w strukturze platform-data, następnie widzimy informację o kolejnych wciśnięciach i zwolnieniach klawisza, oraz w przypadku trzymania klawisza możemy obserwować ciągłe komunikaty, generowane przez mechanizm auto powtarzania.

Lucjan Bryndza, EP
lucjan.bryndza@ep.com.pl