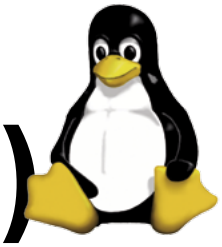
Wprowadzenie do Linuksa embedded (3)



Obsługa portów GPIO

Porty GPIO są jednym z najprostszych układów peryferyjnych, od obsługi których zwykle rozpoczynamy tworzenie programów zapoznając się z danym typem mikrokontrolera. W wypadku aplikacji pracującej pod RTOS'em lub bez systemu operacyjnego, obsługa portów GPIO sprowadza się do wpisania do rejestrów portu kilku wartości określających jego konfigurację, czyli kierunku wejście lub wyjście, załączenie lub wyłączenie rezystorów podciągających itp. W wypadku systemu Linux jest inaczej, ponieważ programista ma do czynienia z obsługą urządzeń, a porty GPIO umożliwiając ich sterowanie nie są urządzeniem w ogólnym rozumieniu tego słowa.

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

Po skonfigurowaniu portu, najczęściej wpisując lub odczytując dane z pojedynczego rejestru, mamy możliwość zmiany stanu linii wyjściowej lub odczytu stanu linii wejściowej. W podobny sposób są zbudowane porty mikrokontrolerów rodziny AT91, do której należy m.in.AT91RM9200. Mamy w nim możliwość sterowania linia GPIO za pomocą kilkunastu rejestrów. W wypadku Linuksa sytuacja komplikuje się, ponieważ bezpośredni dostęp do portów wejścia wyjścia z poziomu programu użytkownika nie jest możliwy, a komunikacja pomiędzy urządzeniami a oprogramowaniem musi odbywać się za pomocą sterowników. Istnieje możliwość obejścia tego za pomocą sterownika /dev/mem, ale nie jest to rozwiązanie zalecane. Brak bezpośredniego dostępu do rejestrów oraz całej fizycznej przestrzeni adresowej jest podyktowany względami bezpieczeństwa. Nietrudno sobie wyobrazić aplikację, która w wyniku zapisania błędnych danych powoduje awarię całego systemu. W Linuksie nie istnieje jeden generyczny mechanizm dostępu do portów GPIO z przestrzeni użytkownika, o tym dlaczego tak jest napiszemy w dalszej części tekstu. W części praktycznej natomiast pokażemy w miarę bezpieczny i uniwersalny sposób na dostęp do portów GPIO w Linuksie z programu użytkownika, na przykładzie płyty prototypowej BF210 za pomocą jednego z mechanizmów – SYSFS GPIO, który nie jest jednak całkowicie zgodny z filozofią sterowników urządzeń.

Sposoby dostępu do portów GPIO w systemie Linux

W Linuksie długo nie istniał standardowy, bezpośredni sterownik linii GPIO, co ma swoje uzasadnienie. Do portów GPIO najczęściej dołączamy jakieś proste układy, na przykład diody LED, przyciski itp. Zgodnie z filozofią systemu Linux właśnie te dołączone urządzenia, czyli wspomniane diody LED i przyciski, powinny mieć swoje sterowniki udostępniające je aplikacjom użytkownika. Porty GPIO stanowią tylko mechanizm dołączania urządzeń, a nie urządzenie samo w sobie. To do jakiego wejścia lub wyjścia dane urządzenie jest dołączone oraz sposób jego sterowania nie powinien mieć znaczenia dla aplikacji. I to jest właśnie dodatkowe zadanie sterowników urządzeń, które stanowią dodatkową warstwę abstrakcji pomiędzy sprzętem a aplikacją użytkownika. Na przykład dioda LED fizycznie może być dołączona bezpośrednio do linii GPIO, ale równie dobrze może być do układu ekspandera I²C. Jednak to gdzie jest ona dołączona z punktu widzenia funkcjonalności aplikacji nie ma żadnego znaczenia.

Na początek zajmiemy się najbardziej eleganckim sposobem dostępu do linii GPIO, czyli sterownikami funkcjonalnymi, które używają pojedynczo linii GPIO. Są to mianowicie sterownik diod LED (LED class driver) oraz sterownik podsystemu Linux Input System dla klawiatury dołączonej do linii GPIO, czyli „GPIO Keys".

Sterownik diod LED (Led Class Driver)

Sterownik LED Class Driver, umożliwia kontrolę diod LED z aplikacji pracującej w przestrzeni użytkownika. Zapewnia on zaawansowany interfejs dla LED'ów umożliwiający ich: włączanie, wyłączanie, włączanie na określony czas bez interwencji aplikacji i sterowanie jasnością świecenia. Funkcja sterowania jasnością zależy od możliwości sprzętu i nie zawsze jest dostępna.

Aby używać sterownika LED Class Driver należy na etapie konfiguracji jądra (Device Drivers –> LED Devices) włączyć następujące opcje:

- LED Class support,
- LED Support for GPIO connected LEDS.

Pierwsza opcja włącza w kernelu obsługę LED Class Drivers, natomiast druga umożliwia używanie przez nią linii GPIO. Z uwagi na to, że w tego rodzaju urządzeniach nie istnieje system powiadamiania o dostępnych zasobach sprzętowych, takich jak BIOS, należy poinformować kernel, do której linii jest dołączona dana dioda oraz jaka jest jej nazwa. Przypisanie poszczególnych urządzeń jest realizowane w kodzie jądra Linuksa, odpowiedzialnym za inicjalizację maszyny. Na przykład, aby diodom o nazwach led0 oraz led1 przypisać sterowanie za pomocą portów PC2 oraz PC3, należy zadeklarować strukturę jak na listingu 8 oraz zarejestrować ją w systemie wywołując funkcję:

```
/* LEDs */
at91_gpio_leds(my_leds, ARRAY_
SIZE(my_leds));
```

Dopisanie diod LED w pliku konfiguracyjny maszyny będzie skutkowało tym, że w katalogu /sys/class/led0 oraz /sys/class/led1 pojawią się następujące pliki umożliwiające kontrolowanie dołączonych diod LED:

- brightness: umożliwia sterowanie jasnością diody LED. W wypadku implementacji za pomocą linii GPIO jak powyżej, wpisanie do tego pliku wartości 0 powoduje wyłączenie diody, natomiast wpisanie wartości różnej od 0 spowoduje jej włączenie.
- trigger: umożliwia ustawienie zaawansowanych wyzwalaczy np. mruganie

Listing 8. Struktura przypisująca diodom LED0 i LED1 sterowanie za pomocą wyprowadzeń PC2 i PC3
```
static struct at91_gpio_led my_leds[] = {
  {
    .name           = "led0",
    .gpio           = AT91_PIN_PC2,
    .trigger        = "timer",
  },
  {
    .name           = "led1",
    .gpio           = AT91_PIN_PC3,
    .trigger        = "timer",
  }
};
```

Listing 9. Deklaracja struktury, która spowoduje generowanie zdarzeń na skutek zbocza opadającego na liniach PC2 i PC3
```
static struct gpio_keys_button my_buttons[] = {
  { .desc = „Key0",
    .gpio = AT91_PIN_PC2,
    .active_low = 1,
    .code = KEY_C,
    .type = EV_KEY,
  },
  { .desc = „Key1",
    .gpio = AT91_PIN_PC2,
    .active_low = 1,
    .code = KEY_D,
    .type = EV_KEY,
  },
};

static struct gpio_keys_platform_data my_button_data = {
  .buttons = my_buttons,
  .nbuttons = ARRAY_SIZE(my_buttons),
};

static struct platform_device gpio_keys_dev = {
  .name = "gpio-keys",
  .id = 0,
  .dev =
  {
    .platform_data = &my_button_data,
  }
};
```

Lisiting 10. Rejestracja zdarzeń z listingu 1 w kodzie maszyny
```
for(i=0; i < ARRAY_SIZE(my_buttons); i++)
   at91_select_gpio(my_buttons[i].gpio, 0);
   platform_device_register(&gpio_keys_dev);
```

podczas aktywności interfejsu USB lub karty sieciowej bez ingerencji programu zewnętrznego np. wpisanie do tego pliku wartości eth0 spowoduje mruganie diody LED w momencie aktywności interfejsu sieciowego. Wpisanie wartości timer umożliwia sterowanie mruganiem diod LED.

– delay_on, delay_off: wpisanie do tych plików wartości włączonej w ms określa stosunek czasu włączenia do wyłączenia diody, co umożliwia jej miganie z wybranym wypełnieniem, gdy jako trigger wybrany został timer.

Na przykład, aby włączyć diodę led0 wystarczy z wiersza polecenia, wpisać:

echo 1 > /sys/class/led/led1/brightness

Sterownik klawiatury z użyciem linii GPIO

W Linuksie obsługa klawiatury jest realizowana za pośrednictwem podsystemu Linux Input System zapewniającego obsługę urządzeń wejściowych, takich jak myszki i klawiatury za pomocą jednolitego interfejsu. Istnieje możliwość uruchomienia sterownika GPIO Keys, który umożliwia współpracę prostych klawiszy dołączonych do linii GPIO z podsystemem Linux Input System. Dzięki uruchomieniu tego sterownika zyskujemy możliwość generowania zdarzeń w wyniku zmiany stanu linii GPIO. Jest to istotne, ponieważ aplikacja korzystająca z podsystemu Linux Input System może zostać uśpiona do czasu wystąpienia zdarzenia od wciśnięcia klawisza. Należy pamiętać, że pracujemy w systemie wielozadaniowym i należy unikać aktywnego odpytywania o stan linii, tak jak to robi się w przypadku aplikacji dla mikrokontrolerów. Aby umożliwić działanie klawiatury dołączonej do linii GPIO, należy w kernelu uaktywnić następujące opcje:

– Event Interface,
– Keyboyard,
– GPIO buttons.

Driver ten również wymaga zdefiniowania struktur w pliku inicjalizacyjnym maszyny zawierających informacje o rodzaju generowanych zdarzeń oraz o przypisanych do nich liniach GPIO. Na przykład, jeżeli chcemy, aby zbocze opadające na liniach PC2 i PC3 wygenerowało zdarzenie od klawiszy „C" i „D", należy zdefiniować struktury, które umieszczono na listingu 9, a następnie zarejestrować w kodzie inicjalizacyjnym maszyny (listing 10).

Od strony aplikacji przestrzeni użytkownika dostęp do wciśniętych klawiszy jest realizowany za pomocą urządzenia /dev/input/event0 udostępniającego ich kody. Za pomocą funkcji blokującej read można uśpić bieżący proces do momentu zaistnienia zdarzenia wciśnięcia klawisza, co jest bardzo ważne w przypadku środowiska, w którym może być uruchomionych wiele procesów.

Metoda brute-force, czyli /dev/mem

Przy omawiania sposobu dostępu do portów GPIO z powodów czysto porządkowych omówimy metodę, której nie powinno stosować się pod żadnym pozorem, a która bardzo często jest używana przez początkujących użytkowników. W Linuksie istnieje sterownik reprezentowany przez plik /dev/mem, którego zadaniem jest udostępnianie całej pamięci fizycznej. Dostęp do tego pliku ma standardowo tylko administrator (root). Użycie tego pliku umożliwia dostęp do fizycznej przestrzeni adresowej, a zatem również do portów GPIO. W Linuksie istnieje również możliwość użycia funkcji mmap, która umożliwia mapowanie pliku w przestrzeni adresowej procesu. Zatem stosując sterownik /dev/mem oraz funkcje mmap możemy uzyskać dostęp do dowolnych rejestrów mikrokontrolera. Przykład użycia /dev/mem wraz z mmap w celu uzyskania dostępu do pamięci fizycznej o adresie base i rozmiarze 1 strony przedstawiono na listingu 11.

Działanie funkcji sprowadza się do otwarcia pliku /dev/mem, a następnie mapowanie tego pliku w przestrzeni adresowej

Listing 11. Przykład użycia /dev/mem wraz z mmap w celu uzyskania dostępu do pamięci fizycznej o adresie base i rozmiarze 1 strony
```
uint32_t *mmap_base_address(unsigned base)
{
  int hwnd = open(„/dev/mem",O_RDWR|O_SYNC);
  if(hwnd<0)
  {
    //error handling
  }
  //Map to process space and calculate address
  unsigned long map_addr = base & ~(getpagesize()-1);
  void *mem = mmap(0,getpagesize(),PROT_READ|PROT_WRITE,MAP_SHARED,hwnd,map_
addr);
  if(!mem)
  {
    //Error handling
  }
  uint32_t *regs = (uint32_t*) mem + (base-map_addr)/4;
  return regs;
}
```

procesu. W wyniku jej działania otrzymujemy wskaźnik do dowolnego obszaru pamięci, a więc również do obszaru rejestrów GPIO, których możemy używać, tak jak w przypadku aplikacji pisanej dla mikrokontrolera. Rozwiązanie to jest bardzo niebezpiecznie i nigdy w praktyce nie powinno być używane. Bezpośredni dostęp do pamięci fizycznej powoduje że nawet niewielki błąd w programie może spowodować zawieszenie się całego systemu. Można powiedzieć, że poprzez użycie tego mechanizmu tracimy wszelkie zalety jakim jest użycie systemu operacyjnego.

Listing 12. Przekształcenie wartości liczbowej przekazanej jako gpio na liczbę

```
//Write int to the selected item
int sysfsgpio_write_int(unsigned gpio, const char *item)
{
  char buf[16];
  snprintf( buf, sizeof(buf),"%d", gpio);
  return sysfsgpio_write(item,buf);
}
```

Listing 13.

```
enum gpio_dir
{
  gpio_dir_input,
  gpio_dir_output
};

int gpio_direction(unsigned gpio, enum gpio_dir dir)
{
  char buf[32];
  snprintf( buf, sizeof(buf),"gpio%d/direction",gpio );
  return sysfsgpio_write(buf, dir==gpio_dir_output?"out":"in");
}
```

Dostęp do linii GPIO za pomocą sysfs gpio interface

Dwie opisane metody, które pośrednio odnoszą się do portów GPIO są bardzo wygodne w wypadku, gdy mamy do czynienia z gotowym urządzeniem i mamy ściśle określone przeznaczenie poszczególnych linii. W przypadku płytek uniwersalnych, takich jak BF210, jest ciężko określić końcowe przeznaczenie linii GPIO i zdefiniować odpowiednie struktury w pliku inicjalizacyjnym maszyny, natomiast użycie /dev/mem jest zbyt niebezpieczne. Dawniej ten problem rozwiązywało się za pomocą sterowników pisanych specjalnie do obsługi GPIO, natomiast obecnie w jądrze istnieje możliwość dostępu do portów GPIO z przestrzeni użytkownika w bezpieczny sposób za pomocą sysfs gpio interface. Ten interfejs jest dostępny za pomocą systemu plików sysfs w katalogu /sys/class/gpio. Należy tutaj podkreślić, że w przypadku rozwiązań produkcyjnych raczej należy używać mechanizmów typu LED class driver czy GPIO keyboyard zamiast interfejsu GPIO.

Poszczególne porty GPIO w interfejsie GPIO są identyfikowane za pomocą liczby porządkowej. W przypadku platformy AT91 linie portu PA0...PA31 mają identyfikatory 32...63, linie portu PB identyfikatory 64...95, natomiast linie portu PC mają identyfikatory 96...127. Aby daną linię portu udostępnić w przestrzeni użytkownika, należy do pliku /sys/class/gpio/export wpisać numer porządkowy linii, którą chcemy udostępnić. Na przykład, aby udostępnić linię portu PB0 w wierszu polecenia należy wydać polecenie: echo 64 > /sys/class/gpio/export.

Zwrócenie danej linii portu do przestrzeni jądra następuje poprzez wpisanie numeru porządkowego linii do pliku /sys/class/gpio/unexport. Mechanizm udostępniania jest stosunkowo bezpieczny, ponieważ w przypadku, gdy chcemy wyeksportować do przestrzeni użytkownika linię portu używaną przez inny driver, to nie zostanie ona udostępniona, a funkcja read() pisząca do pliku export nie powiedzie się zwracając błąd. Dzięki takiemu podejściu mamy pewność, że zmienimy konfiguracji linii używanej w przestrzeni jądra przez inny sterownik. Wyeksportowanie danej linii GPIO do przestrzeni użytkownika spowoduje pojawienie się w katalogu /sys/class/gpio katalogu gpioXXX, gdzie XXX określa numer porządkowy linii GPIO. Dla wspomnianej linii PB0 po wykonaniu polecenia echo 64 > /sys/class/gpio/export, gdy linia nie jest używana przez jądro pojawi się katalog /sys/class/gpio/gpio64, w którym znajdują się pliki umożliwiające sterowanie linią GPIO:

- direction: wpisanie do niego wartości in spowoduje ustawienie portu w kierunku wejścia, wpisanie wartości out powoduje ustawienie wybranego portu w kierunku wyjścia.
- value: w przypadku odczytu umożliwia odczytanie stanu linii portu. W przypadku zapisu wpisanie wartości 1 ustawia wybrany port w stan wysoki, wpisanie wartości 0 zeruje wybrany port.

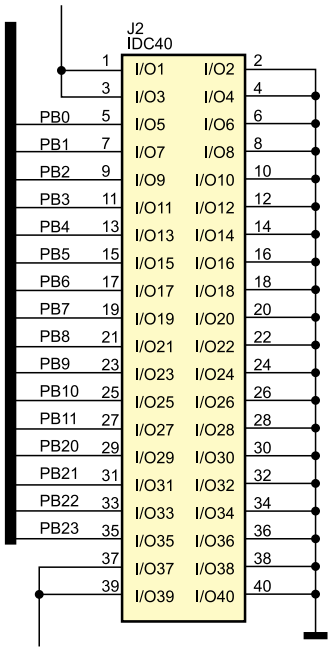
Na przykład aby skonfigurować wyeksportowaną linie PB0 w kierunku wyjścia i ustawić na wyjściu stan niski należy w wierszu poleceń wpisać następującą sekwencję:

```
echo out > /sys/class/gpio/gpio64/direction
echo 0 > /sys/class/gpio/gpio64/value
```

Naturalnie, w praktyce zapis oraz odczyt wspomnianych plików realizowany bezpośrednio w programie na przykład za pomocą wywołań systemowych read(), write(). Przedstawiony sposób nadaje się do sterowania pojedynczymi liniami, do których są podłączone takie peryferia, jak dioda LED, przekaźnik itd. W przypadku realizacji bardziej skomplikowanych sekwencji będziemy zmuszeni napisać sterownik jądra bezpośrednio dla danego urządzenia, ponieważ dostęp za pomocą sysfs-gpio z programu użytkownika jest stosunkowo powolny.

Przykład sterowania liniami GPIO za pomocą klasy GPIO

Po zapoznaniu się z teoretycznymi aspektami użycia GPIO, pokażemy na bazie prostego przykładu dla ARMputera BF210, jak za pomocą klasy GPIO we własnym programie odczytywać, i zmieniać stan wybranych linii portów.



KAMODLED8: (+5V – J2PIN1, GND – J2PIN2, D0 - J2PIN5, D1 – J2PIN7, D2 – J2PIN17, D3 – J2PIN19, D4-J2PIN4, D5-J2PIN6, D6-J2PIN8, D7-J2PIN10)
KAMODKB4x4: (COL1 – J2PIN12, ROW1-J2PIN9)

Rysunek 6. Sposób połączenia modułów

Do uruchomienia programu z przykładu jest potrzebny moduł komputera BF210 oraz moduły KamodLed8 i KamodKB4x4. Na rysunku 6 przedstawiono sposób połączenia modułów.

Sterowanie portami GPIO, pokażemy na bardzo prostym przykładzie, w którym każde wciśnięcie klawisza „1” na klawiaturze KAMODKB4x4 będzie powodować zwiększenie licznika binarnego, którego stan zostanie odzwierciedlony za pomocą diod D0...D3 modułu KAMODLED8.

Przykładowy program został napisany w pojedynczym pliku sysgpio.c. Wykorzystując mechanizm dostępowy klasy GPIO, napisano funkcje, których zadaniem jest wpisywanie (podobnie jak polecenie echo) wartości oraz odczyt (jak polecenie cat) pli-

Listing 14. Licznik binarny

```
//SET signal termination
  struct sigaction sa;
  sigemptyset(&sa.sa_mask);
  sigaddset(&sa.sa_mask,SIGINT);
  sigaddset(&sa.sa_mask,SIGTERM);
  sa.sa_handler = terminate;
  sigaction(SIGINT,&sa, 0);
  sigaction(SIGTERM,&sa, 0);
  //Initialize gpio
  gpio_export(LED1);
  gpio_export(LED2);
  gpio_export(LED3);
  gpio_export(LED4);
  gpio_export(KEY);
  //Set direction
  gpio_direction(LED1,gpio_dir_output);
  gpio_direction(LED2,gpio_dir_output);
  gpio_direction(LED3,gpio_dir_output);
  gpio_direction(LED4,gpio_dir_output);
  gpio_direction(KEY,gpio_dir_input);
  //Main loop
  for(unsigned cnt=0;;)
  {
    //Read value and sleep
    int val1 = gpio_get_value(KEY);
    usleep(SLEEP_VALUE);
    int val2 = gpio_get_value(KEY);
    //Error handling
    if(val1<0 || val2<0) { perror("getval"); exit(-1); }
    //Edge detection
    if(val1==0 && val2==1)
    {
      cnt++;
      if(gpio_set_value(LED1, cnt & 1)<0) { perror("setval1"); cleanup();
exit(-1); }
      if(gpio_set_value(LED2, cnt & 2)<0) { perror("setval2"); cleanup();
exit(-1); }
      if(gpio_set_value(LED3, cnt & 4),0) { perror("setval3"); cleanup();
exit(-1); }
      if(gpio_set_value(LED4, cnt & 8)<0) { perror("setval4"); cleanup();
exit(-1); }
    }
  }
  cleanup();
}
```

ków reprezentujących wybraną linię GPIO. Zapis do plików jest realizowany za pomocą funkcji biblioteki standardowej stdio fputs(), natomiast odczyt jest realizowany za pomocą funkcji fgets(), co realizują funkcje int sysfsgpio_write(const char *item, const char *value) oraz int sysfsgpio_read(const char *item, char *value, size_t value_len). Ich zadaniem jest odpowiednio, zapis oraz odczyt pliku o nazwie przekazanej jako item w katalogu /sys/class/gpio/ wartości tekstowej przekazanej jako value. W przypadku powodzenia funkcja zwraca 0, natomiast w przypadku niepowodzenia jest zwracana wartość ujemna. Funkcja pomocnicza umieszczona na listingu 12 przekształca wartość liczbową przekazaną jako gpio na liczbę i zapisuje ją do pliku określonego jako item za pomocą funkcji sysfsgpio_write. Zadaniem funkcji gpio_export jest udostępnienie portu GPIO dla przestrzeni użytkownika poprzez wpisanie numeru porządkowego portu do pliku /sys/class/gpio/export:

```
//Export selected gpio
inline int gpio_export(unsigned
gpio)
{
        return sysfsgpio_write_
int(gpio,"export");
}
```

Podobną rolę pełni funkcja unexport, której zadaniem jest zwrócenie linii o numerze porządkowym przekazanym jako gpio do przestrzeni jądra:

```
//Unexport selected gpio
inline int gpio_unexport(unsigned
gpio)
{
        return sysfsgpio_write_
int(gpio,"unexport");
}
```

Funkcja gpio_direction umożliwia skonfigurowanie wybranej linii GPIO w kierunku wejścia lub wyjścia (listing 13). Jej działanie sprowadza się do wpisania do pliku direction wybranej linii gpio przekazanej jako argument gpio, wartości in lub out, co spowoduje ustawienie tej linii jako wejściowej lub wyjściowej. Ustawienie wybranej linii portu na poziomie wysokim lub niskim jest realizowane za pomocą funkcji:

```
//Set gpio value
int gpio_set_value(unsigned gpio,
bool value)
{
  char buf[32];
 snprintf(buf,sizeof(buf),"gpio%d/
value",gpio );
  return sysfsgpio_write(buf,
value?"1":"0");
}
```

Działanie tej funkcji sprowadza się do wpisania „0" lub „1" do pliku value wartości wybranej jako argument gpio linii GPIO. Odczyt linii portu jest realizowany za pomocą funkcji:

```
int gpio_get_value(unsigned gpio)
{
  char buf[32];
  snprintf( buf,
sizeof(buf),"gpio%d/value",gpio
);
  int res = sysfsgpio_read( buf,
buf,sizeof(buf) );
  if(res < 0) return res;
  return atoi(buf);
}
```

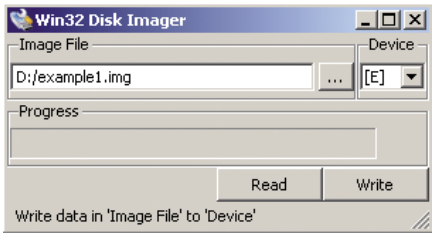
Funkcja ta odczytuje z pliku value stan wybranej linii gpio w postaci tekstowej 0 lub 1, a następnie przekształca ją na wartość typu int. Przykładowy kod programu licznika binarnego, wykorzystujący powyższe funkcje przedstawiono na listingu 14.

Na początku jest instalowany jest handler dla sygnałów SIGINT oraz SIGTERM, którego zadaniem jest oddanie przed zakończeniem programu linii GPIO za pomocą funkcji gpio_unexport() do przestrzeni jądra. Następnie jest wywoływana funkcja gpio_export(), która powoduje wyeksportowanie wybranych linii GPIO do przestrzeni użytkownika. Linie diod LED1...LED4 są konfigurowane jako wyjście, natomiast linia klawisza KEY, jest konfigurowana jako wejście. Teraz program wchodzi do pętli nieskończonej, w której jest odczytywany stan linii KEY, po czym jest wywoływana funkcja usleep() usypiająca proces na 20 ms, a następnie ponownie jest odczytywany stan linii KEY. W przypadku zmiany poziomu z „1" na „0" (zbocze opadające) jest zwiększana wartość zmiennej cnt, a następnie ustawiane są poszczególne linie portów LED1...LED4 odzwierciedlające stan bitów portu cnt. Zatrzymanie wykonania programu, a więc zatrzymanie pętli głównej odbywa się w wyniku utrzymania sygnału SIGINT (wciśnięcie na konsoli CTRL+C), lub SIGTERM (sygnał terminacji procesu), w wyniku czego jest wywoływany umieszczono na listingu 15 handler terminate(), w którym na konsolę jest wysyłany komunikat o zamknięciu programu, zostaje przywrócony stary handler obsługi sygnałów, jest wywoływana funkcja cleanup(), której zadaniem jest zwrócenie linii GPIO do przestrzeni jądra poprzez wywołanie gpio_unexport(), a następnie proces ulega zakończeniu dzięki wywołaniu funkcji exit().

Opisany powyżej program ma wadę polegającą na tym, że stan linii klawisza jest sprawdzany w aktywnej pętli, co nie jest do-

Listing 15. Wywołanie handlera terminate

```
//Termination handler
void terminate(int sig)
{
  printf("Led program
terminated\n");
  cleanup();
  //Restore default sig
  struct sigaction sa;
  sigemptyset(&sa.sa_mask);
  sigaddset(&sa.sa_mask,SIGINT);
  sigaddset(&sa.sa_mask,SIGTERM);
  sa.sa_handler = SIG_DFL;
  sigaction(SIGINT,&sa, 0);
  exit(-1);
}
```



Rysunek 7. Okno wyboru identyfikatora dysku

brym rozwiązaniem w przypadku środowisk wielozadaniowych. Wówczas dużo lepszym rozwiązaniem byłoby użycie sterownika GPIO KEYS i podsystemu Input. Takie rozwiązanie zapewniałoby uśpienie procesu do momentu wciśnięcia klawisza bez konieczności cyklicznego sprawdzania stanu wejścia KEY.

Uruchomienie przykładu, na BF210

Skompilowany przykład wraz z podstawowym system Linux jest dostarczany w postaci obrazu karty SD, w postaci pojedynczego pliku example1.img W systemie Windows obraz z przykładem można nagrać na kartę SD za pomocą programu Win32diskImager, który można pobrać ze strony *https://wiki.kubuntu.org/Win32DiskImager*. Do czytnika kart SD należy włożyć kartę, a następnie podłączyć czytnik do komputera. Teraz uruchomić program i jako opcje image file wybrać plik example1.img, natomiast jako device należy wybrać identyfikator dysku, którym jest czytnik kart SD (rysunek 7). Kliknięcie na guzik Write powoduje rozpoczęcie zapisywania obrazu karty. Z poziomu systemu Linux, obraz można nagrać na kartę za pomocą standardowego polecenia dd (dd if=example.img of=/dev/identyfikator_urz_czytnika).

Po nagraniu obrazu karty należy ją umieścić w gnieździe karty komputera BF210, do portu DBGU należy dołączyć komputer PC, na którym powinien być uruchomiony program terminalowy (Hyperterminal, minicom itp.) skonfigurowany z następującymi parametrami transmisji 115200, n, 8, 1. Po włączeniu zasilania powinien być widoczny proces bootowania Linuksa, który powinien zakończyć się wyświetleniem zachęty do logowania bf210-at91 login:. Jako nazwę użytkownika należy wpisać root, a na pytanie o hasło wcisnąć klawisz Enter. Po zalogowaniu się należy wpisać polecenie sysgpio, co spowoduje uruchomienie opisanego wcześniej przykładu. Wciskając klawisz „1" możemy obserwować jak zmienia się stan diod LED1...LED4 reprezentujący licznik binarny. Zakończenie programu jest możliwe poprzez wciśnięcie kombinacji klawiszy CTRL+C, co spowoduje wypisanie komunikatu pożegnalnego oraz zakończenie pracy programu.

Kompilowanie programu ze źródeł

System Linux oraz wszystkie przykłady zostały przygotowane z wykorzystaniem pakietu OpenEmbedded. Sposób skompilowania powyższego przykładu oraz systemu wykracza poza ramy niniejszego artykułu. Dodatkowe pakiety dla OpenEmbedded oraz konfiguracja dla BF210 są dostarczane w postaci pliku oe.tar.gz. Aby skompilować powyższy przykład, należy skonfigurować OpenEmbeded zgodnie z dokumentacją dostępną na stronie projektu i dołączyć pakiety lokalne. Aby zbudować kompletny obraz dla przykładu, należy wydać polecenie bitbake ep-examples-image. Aby zbudować tylko pakiet z powyższym przykładem, należy wydać polecenie bitbake ep-example-sysfsgpio

Lucjan Bryndza, EP