Podstawy Systemów Wbudowanych

Wykład 11: Programowanie wejścia/wyjścia w systemie operacyjnym

Angelika Tefelska Dariusz Tefelski

Zakład Fizyki Jądrowej, Wydział Fizyki PW

10 maja 2017

O czym będzie wykład...

1. WiringPi
2. PIGPIO
3. Interfejsy komunikacyjne
   - SPI

WiringPi

WiringPi

WiringPi to podstawowa biblioteka do obsługi GPIO napisana w C in C dla Raspberry Pi. Biblioteka może być używana w programach bazujących na C, C++ i RTB (BASIC) oraz na innych językach po zainstalowaniu odpowiednich wrappers. Głównym celem było stworzenie biblioteki wygodnej w użyciu dla użytkowników przyzwyczajonych do korzystania z Arduino.

Obsługa w bash-u

- Korzystając z WiringPi można obsłużyć GPIO korzystając z bash-a. Podstawowe komendy to:
  - gpio readall - zwraca informacje o wszystkich pinach
  - gpio -g mode numer_pinu out/in - ustawia dany pin jako wejściowy lub wyjściowy
  - gpio -g write numer_pinu wartość - ustawia wartość na danym pinie z zakresu (0,1)
  - gpio -g read numer_pinu - umożliwia odczyt wartości z danego pinu
  - gpio -g mode numer_pinu pwm - ustawienie trybu pwm
  - gpio -g pwm numer_pinu wypenienie - ustawienie wypełnienia z zakresu (0,1023)
  - gpio -g mode numer_pinu down/up - ustawienie rezystorów podciągających

Oznaczenia pinów

P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

WiringPi - bash

Przykład: zapalenie diody (pin 17) po naciśnięciu przycisku (pin 23).

```
#!/bin/bash

gpio -g mode 23 in
gpio -g mode 17 out
gpio -g mode 23 down

while true; do
x='gpio -g read 23'
if [ $x -ge 1 ]
then
gpio -g write 17 1
else
gpio -g write 17 0
sleep 0.2
fi

done
```

WiringPi - C

Korzystanie z biblioteki WiringPi w C

- Bibliotekę WiringPi należy załączyć korzystając z: **#include <wiringPi.h>**
- Do inicjalizacji biblioteki WiringPi służą dwie funkcje:
  - **wiringPiSetup();** - przy wyborze numeracji zaproponowanej przez twórców WiringPi
  - **wiringPiSetupGpio()** - przy wyborze numeracji Broadcom GPIO (numeracja z procesora)
- Ustawienie danego pinu jako wejściowego/wyjściowego można wykonać przy użyciu funkcji: **pinMode(numer pinu, INPUT/OUTPUT)**
- Ustawienie danego pinu jako **PWM: pinMode(numer pinu, PWM_OUTPUT)**
- Ustawienie wartości na danym pinie: **digitalWrite(numer pinu, HIGH/LOW)**
- Ustawienie wypełnienia dla PWM: **pwmWrite(numer pinu, wypełnienie)**, przy czym wypełnienie jest z zakresu: [0,1023].
- Ustawienie rezystorów podciągających: **pullUpDnControl(numer pinu, PUD_UP)**
- Funkcja do opóźnienia: **delay(czas w ms)**.

WiringPi - przykład

Przykład: Jeśli przycisk zostanie włączony to zostanie ustawione wypełnienie ok 7% i dioda zostanie zgaszona. W przeciwnym wypadku wypełnienie będzie ustawione na wartość przeciwną (100%-7%) oraz dioda zacznie mrugać.

```
#include <stdio.h>
#include <wiringPi.h>

const int pwmPin = 18;
const int ledPin = 23;
const int butPin = 17;

const int pwmValue = 75;

int main(void)
{
    wiringPiSetupGpio();

    pinMode(pwmPin, PWM_OUTPUT);
    pinMode(ledPin, OUTPUT);
    pinMode(butPin, INPUT);
    pullUpDnControl(butPin, PUD_UP);
```

WiringPi - przykład

```
    while(1)
    {
        if(digitalRead(butPin))
        {
            pwmWrite(pwmPin, pwmValue);
            digitalWrite(ledPin, LOW);
        }
        else
        {
            pwmWrite(pwmPin, 1024 - pwmValue);
            digitalWrite(ledPin, HIGH);
            delay(75);
            digitalWrite(ledPin, LOW);
            delay(75);
        }
    }

    return 0;
}
```

Kompilacja przykładu: **gcc -o blinker blinker.c -l wiringPi**
Uruchomienie przykładu: **sudo ./blinker**

WiringPi2 - Python

WiringPi2

- WiringPi-Python umożliwia skorzystanie z biblioteki WiringPi pod Pythonem. Jednak ma swoje wady. Przede wszystkim jest wolny i posiada dokumentacje z licznymi lukami. Alternatywą jest skorzystanie z WiringPi2, który całkiem dobrze sprawuje się pod Pythonem.
- Aby skorzystać z WiringPi2 należy go zaimportować: **import wiringpi2 as wiringpi**
- Funkcje do inicjalizacji biblioteki:
  - **wiringpi.wiringPiSetup()** - przy wyborze numeracji zaproponowanej przez twórców WiringPi
  - **wiringpi.wiringPiSetupGpio()** - przy wyborze numeracji Broadcom GPIO
- Konfiguracja portów:
  - **wiringpi.pinMode(numer pinu, 0)** - jako wejście
  - **wiringpi.pinMode(numer pinu, 1)** - jako wyjście
  - **wiringpi.pinMode(numer pinu, 2)** - tryb PWM
- Funkcja do odczytu wartości z pinu: **wiringpi.digitalRead(numer pinu)**
- Ustawienie wartości na danym pinie:
  - **wiringpi.digitalWrite(numer pinu, 0)** - ustawienie 0V
  - **wiringpi.digitalWrite(numer pinu, 1)** - ustawienie 3.3V

WiringPi2 - przykład

Przykład: Zapalenie diody po wciśnięciu przycisku.

```
import wiringpi2 as wiringpi
from time import sleep

wiringpi.wiringPiSetupGpio()
wiringpi.pinMode(24, 1)
wiringpi.digitalWrite(24, 0)

wiringpi.pinMode(25, 0)

while True:
    if wiringpi.digitalRead(25):
        wiringpi.digitalWrite(24, 1)
    else:
        wiringpi.digitalWrite(24, 0)
    sleep(0.05)
```

WiringPi2 - przykład 2

Przykład: Obsługa serwomechanizmu przez PWM:

```
import wiringpi2 as wiringpi

wiringpi.wiringPiSetupGpio()

SERVO_PIN = 25

wiringpi.pinMode(SERVO_PIN, 1)
wiringpi.softPwmCreate(SERVO_PIN, 0, 100)

wiringpi.softPwmWrite(SERVO_PIN, 1)
wiringpi.delay(200)
wiringpi.softPwmWrite(SERVO_PIN, 20)
wiringpi.delay(200)
```

PIGPIO

Obsługa w bash-u

- Bibliotekę PIGPIO można obsłużyć zarówno pod C/C++ jak i Python-em przy włączonym pigpio daemon. Dokumentacja wraz z przykładami jest dostępna na stronie.
- Z biblioteki pigpio można też skorzystać pod bash-em. Podstawowe komendy to:
  - pigs m numer_pinu o/i - ustawienie danego pinu jako wyjściowego (o) lub wejściowego (i)
  - pigs w numer_pinu wartość - ustawienie danej wartości na pinie z zakresu (0,1)
  - pigs r numer_pinu - odczyt wartości z pinu
  - pigs pud numer_pinu u/d - podciągnięcie rezystorów
  - pigs s numer_pinu wypełnienie - funkcja do sterowanie serwomechanizmem, w której wypełnienie przyjmuje wartości z zakresu (500,2500)
  - pigs p numer_pinu wypełnienie - generowanie sygnału PWM o wypełnieniu z zakresu (0,255)

PIGPIO - przykład: mrugająca dioda

```
import time
import pigpio

LED_PIN = 25

#polaczenie z pigpiod daemon
pi = pigpio.pi()
pi.set_mode(LED_PIN, pigpio.OUTPUT)

while True:
  pi.write(LED_PIN, 1)
  pi.write(LED_PIN, 0)
  time.sleep(1)

#clean
pi.set_mode(LED_PIN, pigpio.INPUT)
pi.stop()
```

PIGPIO - przykład: przycisk

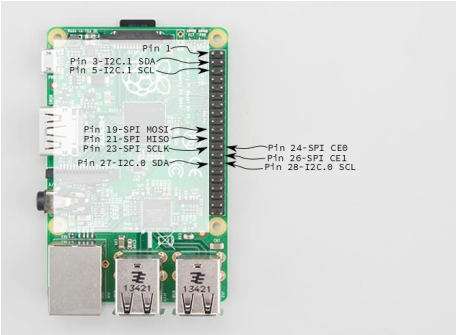
```
import time
import pigpio


LED_PIN = 25

pi = pigpio.pi()

pi.set_mode(LED_PIN, pigpio.INPUT)

while True:
  print pi.read(LED_PIN)
  time.sleep(1)

pi.stop()
```

PIGPIO - przykład: serwomechanizm

```
import time
import pigpio

LED_PIN = 25

pi = pigpio.pi()

pi.set_mode(LED_PIN, pigpio.OUTPUT)

pi.set_servo_pulsewidth(LED_PIN, 500)
time.sleep(1)
pi.set_servo_pulsewidth(LED_PIN, 2500)
time.sleep(1)

pi.set_mode(LED_PIN, pigpio.INPUT)
pi.stop()
```

Interfejsy komunikacyjne

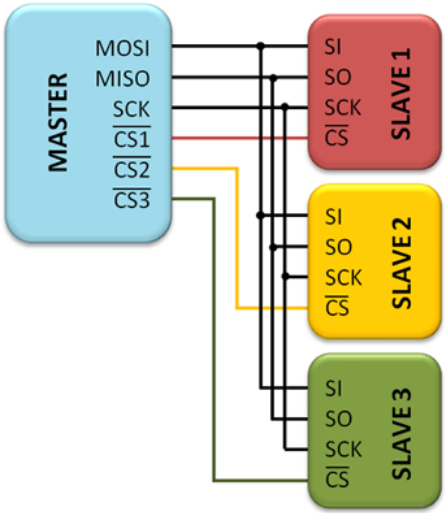
- Raspberry Pi posiada trzy interfejsy komunikacyjne dostępne przez GPIO: I2C, SPI oraz UART.
- SPI umożliwia podłączenie tylko dwóch urządzeń jednocześnie do RaspberryPi.
- Natomiast I2C umożliwia podłączenie wielu urządzeń pod warunkiem niedochodzenia do konfliktu adresowego między urządzeniami.
- Oby dwa interfejsy nie są włączone domyślnie. Zatem wymagają dodatkowej konfiguracji.
- Do obsługi SPI są dostępne następujące biblioteki: WiringPiSPI, BCM2835 oraz SPIDEV.



Serial Peripherial Interface

- SPI jest interfejsem typu full-duplex, co oznacza, że dane mogą być wysyłane i odbierane w tym samym czasie.
- SPI jest interfejsem synchronicznym, co oznacza, że jednym z przewodów przesyła się sygnał zegarowy, synchronizujący wszystkie układy. Generalnie SPI składa się z 4 linii:
  1. MOSI (master out, slave in) – linia łącząca wyjście danych z mastera i wejścia slave'ów.
  2. MISO (master in, slave out) – linia łącząca wyjście danych slave i wejście mastera.
  3. SCK (serial clock) – sygnał zegarowy, synchronizujący układy na magistrali.
  4. CS (chip select) – sygnał informujący slave o rozpoczęciu transmisji. Oznaczany często jako SS.
- Możliwe jest połączenie wielu układów natomiast może być tylko jeden układ typu master.
- Zaletą SPI jest efektywna komunikacja z wybranym urządzeniem peryferyjnym, przy uśpieniu pozostałych dzięki linii SS.
- Wadą SPI jest wykorzystanie aż 4 linii do komunikacji.

Połączenie urządzeń typu slave przez SPI



Rysunek : Schemat podłączenia urządzeń slave przez SPI. [1]

[1] Źródło: mikrokontroler.pl

WiringPiSPI

- Domyślnie SPI jest wyłączone natomiast korzystając z biblioteki WiringPi można łatwo je włączyć: **gpio load spi**
- W celu skorzystania z biblioteki WiringPi należy ją zaimportować: **#include <wiringPiSPI.h>**
- Konfiguracja SPI jest możliwa dzięki funkcji: **wiringPiSPISetup (numer kanału, szybkość)**, gdzie numer kanału to 0 lub 1 a szybkość przyjmuje wartości z zakresu: [500 000, 32 000 000]
- Odczyt i wysyłanie danych odbywa się za pomocą funkcji: **wiringPiSPIDataRW (int channel, unsigned char *data, int len)**

WiringPiSPI - przykład

```
import wiringpi
SPIchannel = 1
SPIspeed = 500000 #Czestotliwosc w Hz

wiringpi.wiringPiSetupGpio()
wiringpi.wiringPiSPISetup(SPIchannel, SPIspeed)

sendData = str(42) #Taki zapis sprawi, ze zostana wyslane dwa
    ↪ bajty: 4 i 2
recvData = wiringpi.wiringPiSPIDataRW(SPIchannel, sendData)
#recvData to lista zawierajaca takie dane jak: [NumOfBytes,
    ↪ recvDataStr] np. [2, '\x04\x02']

#Przyklad alternatywny wyslania calej liczby 42
sendData = chr(42)
recvData = wiringpi.wiringPiSPIDataRW(SPIchannel, sendData)
#recvData = [1, '\x2A']
```

SPI - BCM2835

Funkcje biblioteki BCM2835

- **bcm2835_spi_begin();** - inicjalizacja SPI czyli ustawienie pinów: P1-19 (MOSI), P1-21 (MISO), P1-23 (CLK), P1-24 (CE0) and P1-26 (CE1) jako funkcjonalność podaną w nawiasach.
- **bcm2835_spi_end()** - operacja przeciwna do omówionej w poprzednim punkcie czyli powrót pinów do ich pierwotnych funkcji.
- **bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);** - ustawienie transmisji danych od najbardziej znaczącego bitu.
- **bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);** - wybranie trybu pracy SPI: CPOL = 0, CPHA = 0 (Ustawienie zboczy SCK).
- **bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_128);** - ustawienie dzielnika częstotliwości
- **bcm2835_spi_chipSelect(BCM2835_SPI_CS0);** - ustawienie pinu chip select
- **bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);** - ustawienie aktywacji pinu SS wartością LOW
- **bcm2835_spi_transfer(uint_t value);** - przesłanie jednego bajtu danych
- **bcm2835_spi_transfernb(char *tbuf,char *rbuf,uint32_t len);** - przesłanie dowolnej liczby bajtów danych

SPI - BCM2835 przykład

```
#include <bcm2835.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    bcm2835_init();
    bcm2835_spi_begin();
    bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);
    bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);
    bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_64);
    bcm2835_spi_chipSelect(BCM2835_SPI_CS0);
    bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);

    uint8_t mosi[10] = { 0x60, 0x00 };
    uint8_t miso[10] = { 0 };
    bcm2835_spi_transfernb(mosi, miso, 2);
    printf("Analogue level from SPI: %04x\n", miso[1] + ((miso[0]
        ↪ & 3) << 8));

    bcm2835_spi_end();
    bcm2835_close();
    return 0;
}
```

SPIDEV

Konfiguracja SPI

- Uruchom: sudo raspi-config
- Wybierz: Advanced Options > SPI
- Zrób reboot albo załaduj jeszcze raz jądro linuxa: sudo modprobe spi-bcm2708

Instalacja SPIDEV-a

- sudo apt-get update
- sudo apt-get upgrade
- sudo apt-get install python-dev python3-dev
- cd ∼
- git clone https://github.com/doceme/py-spidev.git
- cd py-spidev
- make
- sudo make install

SPIDEV - przykład

```
#!/usr/bin/python

import spidev
import time

spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 7629

# Split an integer input into a two byte array to send via SPI
def write_pot(input):
    msb = input >> 8
    lsb = input & 0xFF
    spi.xfer([msb, lsb])

# Repeatedly switch a MCP4151 digital pot off then on
while True:
    write_pot(0x1FF)
    time.sleep(0.5)
    write_put(0x00)
    time.sleep(0.5)
```

THAT'S ALL FOR TODAY....
ANY QUESTION??