Sprawozdanie z ćwiczenia nr 2

Mateusz Okulus, Mariusz Pakulski

Cel ćwiczenia

Celem laboratorium było wykonanie pięciu zadań, które dotyczyły sterowania układami za pomocą modulacji szerokości impulsów (PWM, Pulse-width Modulation). Należało się również zapoznać z systemem OpenWRT oraz obsługą interfejsów GPIO przez sysfs.

Połączenie RaspberryPI

RaspberryPI łączymy tak jak w poprzednim ćwiczeniu.

Konfiguracja OpenWRT

- Pobieramy obraz systemu
- Dekompresujemy

  ```
  gzip -d openwrt-21.02.1-bcm27xx-bcm2711-rpi-4-ext4-factory.img.gz
  ```

- Ładujemy jako urządzenie loop

  ```
  losetup -P -f openwrt-21.02.1-bcm27xx-bcm2711-rpi-4-ext4-factory.img
  ```

- Urządzenie jest oznaczone jako `/dev/loop0` po wywołaniu `losetup -a`
- Kopiujemy rootfs z obrazu OpenWRT do karty SD

  ```
  dd if=/dev/loop0p2 of=/dev/mmcblk0p2 bs=4096
  ```

- Tworzymy odpowiednie katalogi

  ```
  mkdir /mnt/boot /mnt/owrt
  ```

- Montujemy odpowiednie partycje

  ```
  mount /dev/loop0p1 /mnt/owrt
  mount /dev/mmcblk0p1 /mnt/boot
  ```

- Kopiujemy pliki z obrazu OpenWRT do karty SD

```
cp /mnt/owrt/cmdline.txt /mnt/boot/user/
cp /mnt/owrt/kernel8.img /mnt/boot/user/
cp /mnt/owrt/bcm2711-rpi-4-b.dtb /mnt/boot/user/
```

- Powiększamy system plików OpenWRT tak, żeby wypełnił partycję

  `resize2fs /dev/mmcblk0p2`

- Restartujemy i uruchamiamy OpenWRT przytrzymując przycisk na RPI

Konfiguracja sieci w OpenWRT

Zgodnie z instrukcją w pliku `/etc/network/config` część `device` usuwamy, a interfejs `lan` ustawiamy na używanie `dhcp`. Restartujemy sieć poleceniem `/etc/init.d/network reload`.

Dodatkowa konfiguracja

- Aktualizujemy listę dostępnych pakietów

  `opkg update`

- Instalujemy python i pip

  `opkg install python3 python3-pip`

- Instalujemy bibliotekę gpio4

  `pip3 install gpio4`

Zadanie 1 - wyjście dla LED

```python
import gpio4
import time

PIN = 27

gpio = gpio4.SysfsGPIO(PIN)
gpio.export = True
gpio.direction = 'out'

for _ in range(10):
    gpio.value = 1
    time.sleep(0.5)
    gpio.value = 0
    time.sleep(0.5)

gpio.export = False
```

Ze schematu płytki rozszerzeń odczytujemy, że LED czerwony jest obsługiwany przez pin 27. W programie tworzymy pin (`gpio.export = True`) i ustawiamy go na wyjście (`gpio.direction = 'out'`). Następnie dziesięciokrotnie włączamy pin, czekamy pół sekundy, wyłączamy i czekamy kolejne pół sekundy.

Zadanie 2 - wyjście dla LED z płynną zmianą jasności

```
import gpio4
import time
import math

PIN = 27

def cycle(gpio, period, duty=0.5):
    gpio.value = 1
    time.sleep(period * duty)
    gpio.value = 0
    time.sleep(period * (1 - duty))

gpio = gpio4.SysfsGPIO(PIN)
gpio.export = True
gpio.direction = 'out'

duration = 10
start = time.time()
end = time.time() + duration
t = 0
frequency = 100
period = 1/frequency

while time.time() < end:
    duty = math.sin(math.pi * t/duration)
    print(duty)
    cycle(gpio, period, duty)
    t += period

gpio.export = False
```

Wizualnie zmianę jasności LED możemy uzyskać poprzez migotanie z dużą częstotliwością. Jasność będzie odpowiadała współczynnikowi wypełnienia fali - jeżeli dioda będzie świecić przez jedną trzecią okresu to będzie to odpowiadało jednej trzeciej normalnej jasności diody.

Nadal używamy tego samego pina. Dodajemy funkcję pomocniczą `cycle`, która

na zadanym obiekcie `gpio` wyprowadzi jeden okres o długości `period` sygnału o określonym współczynniku wypełnienia `duty`.

Zakładamy, że wystarczy częstotliwość 100 Hz. Zmienność jasności uzyskujemy przez wybór `duty` za pomocą `sin(pi * t/10)` do przy długości 10 sekund od powiada sinusowi od 0 do pi - sygnał zaczyna się w zerze, wznosi się do jedynki w ciągu pierwszych pięciu sekund, a następnie opada do zera przez kolejne pięć sekund.

Ponieważ jądrem OpenWRT jest Linux, który nie jest systemem czasu rzeczywistego, czekanie przez określony czas za pomocą `time.sleep` zwykle czeka trochę dłużej niż zadano. Ponieważ czekamy 2*100*100=20,000 razy nawet niewielki błąd szybko się kumuluje. W programie objawia się to tym, że wypisywana wartość `duty` nie zeruje się do końca. Na potrzeby laboratorium założyliśmy, że problem ten można pominąć.

Zadanie 3 - wejście

```
import gpio4
import time

PIN = 27
BUTTON_PIN = 18

gpio = gpio4.SysfsGPIO(PIN)
gpio.export = True
gpio.direction = 'out'

button = gpio4.SysfsGPIO(BUTTON_PIN)
button.export = True
button.direction = 'in'

# 1 bo z ukladu przyciskanie robi zwarcie do masy
while button.value == 1:
    time.sleep(1/60)
gpio.value = 1
time.sleep(3)
gpio.value = 0

button.export = False
gpio.export = False
```

Wybraliśmy przycisk 1. Na podstawie schematu płytki rozszerzeń odczytujemy odpowiadający mu pin 18. Na schemacie zauważamy też, że na wejście 18 prąd ciągle płynie, a naciśnięcie przycisku powoduje przekierowanie prądu do masy. Oznacza to, że po wciśnięciu GPIO przechodzi z 1 do 0, a nie z 0 do 1 jak można

by się tego spodziewać.

Pin odpowiadający przyciskowi ustawiamy w tryb `'in'` do odczytu. Aktywnie oczekujemy na pojawienie się 1, sprawdzając wejście 60 razy na sekundę. Po wciśnięciu włączamy LED, czekamy 3 sekundy, a następnie ją wyłączamy.

Zadanie 4 - wyjście PWM, buzzer pasywny

Schemat jest dostępny w pliku `buzzer.fzz`. Buzzer zasilamy napięciem 3.3V. Dodatkowo dodaliśmy rezystor 100 Ohmów.

```
import gpio4
import time
import math

PIN = 17

def cycle(gpio, period, duty=0.5):
    gpio.value = 1
    time.sleep(period * duty)
    gpio.value = 0
    time.sleep(period * (1 - duty))

gpio = gpio4.SysfsGPIO(PIN)
gpio.export = True
gpio.direction = 'out'

base_frequency = 261.63
mult = 2**(1/12)

for i in range(24):
    duration = 1
    start = time.time()
    end = time.time() + duration
    t = 0
    frequency = base_frequency * mult**i
    period = 1/frequency
    while time.time() < end:
        cycle(gpio, period, 0.5)
        t += period
gpio.export = False
```

Dźwięk zależy tylko od częstotliwości, więc `duty` zawsze wynosi `0.5`. Częstotliwości kolejnych nut obliczamy wybierając częstotliwość bazową nuty C i mnożąc ją przez kolejne potęgi pierwiastka dwunastego stopnia z dwóch. Dzięki temu po 12 nutach (oktawa) częstotliwość zwiększy się dwukrotnie, przechodząc do

kolejnej oktawy. Każdą nutę gramy 1 sekundę.

Zadanie 5 - sterowanie serwomotorem

Schemat jest dostępny w pliku `servo.fzz`. W repozytorium znajdują się również zdjęcia podłączenia urządzeń oraz schematu w programie Fritzing.

Do zadania piątego wybraliśmy serwomotor AR-3606HB. Steruje się nim za pomocą podania sygnału o częstotliwości 50Hz (okres 20ms). Współczynnik wypełnienia powinien wynosić od 5% (1ms) do 10% (2ms), co odpowiada pozycji wychylenia maksymalnej i minimalnej.

```
import gpio4
import time
import math

PIN = 17

def cycle(gpio, period, duty=0.5):
    gpio.value = 1
    time.sleep(period * duty)
    gpio.value = 0
    time.sleep(period * (1 - duty))

def position(gpio, value, duration):
    period = 0.020
    start = time.time()
    end = time.time() + duration
    t = 0
    duty = 0.05 + 0.05 * value
    while time.time() < end:
        cycle(gpio, period, duty)
        t += period

gpio = gpio4.SysfsGPIO(PIN)
gpio.export = True
gpio.direction = 'out'

for i in range(12):
    v = (i % 3) / 3
    position(gpio, v, 1)
gpio.export = False
```

Funkcja `position` na podstawie wartości `value` od 0 do 1 ustawi pozycję od minimalnej do maksymalnej. `duration` określa przez ile czasu wystawić sygnał. Program 4 razy wysteruje serwomotor do pozycji 0, 0.5 i 1, po sekundę na każdą.

Podsumowanie

Na laboratorium udało się nam wykonać wszystkie wymagane zadania. Nauczyliśmy się sterować układami poprzez polecenia wykonywane w systemie OpenWRT. Po konfiguracji tego systemu przećwiczyliśmy obsługę urządzeń przez interfejs GPIO za pomocą prostego wyjścia (migotanie LED), prostego wejścia (przycisk), a także za pomocą sygnału o zmiennej częstotliwości (buzzer) i zmiennym współczynniku wypełnienia (płynna zmiana LED, serwomotor).