Pico Pagers
Urządzenia do powiadamiania klienta w restauracji

Systemy Wbudowane

Jakub Kosmydel
Norbert Morawski

6 czerwca 2023

Spis treści

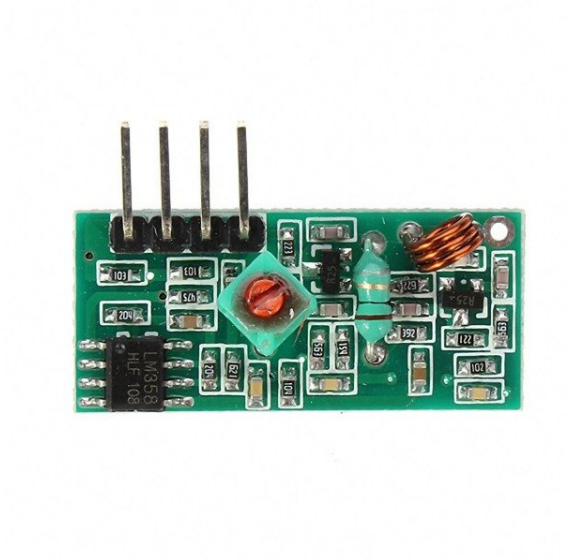
1 Wprowadzenie

Dummy

2 Protokół komunikacyjny

Zastosowaliśmy moduły komunikacyjne 433 MHz. Skłoniły nas do tego niska cena i prostota obsługi oraz brak wymaganej komunikacji zwrotnej przez nasze urządzenia.

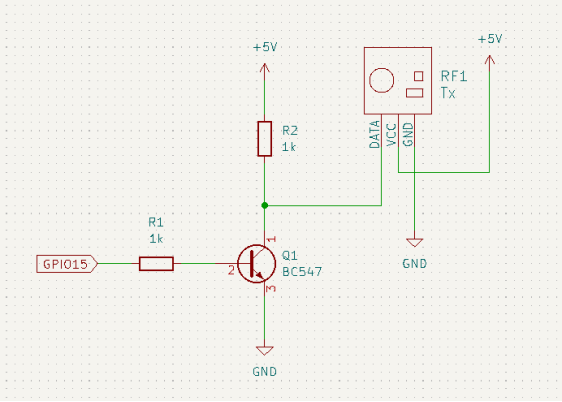
Rysunek 1: Nadajnik



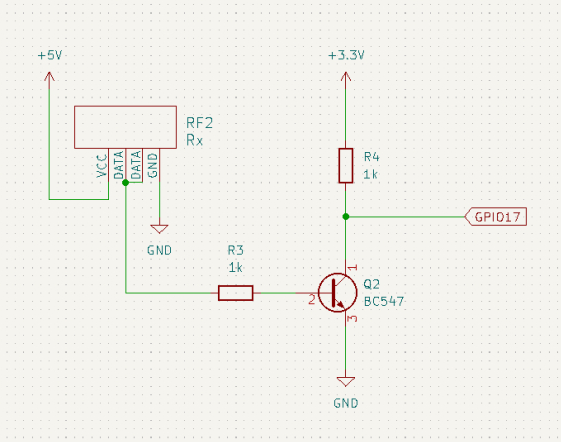
Rysunek 2: Odbiornik

2.1 Warstwa sprzętowa

Obsługa nadajnika/odbiornika opiera się na podłączeniu zasilania i nadawania/odbierania poprzez jeden dostępny przewód danych. Niestety te moduły zasilane są napięciem 5V i taki standard napięć stosują na wyjściu. Pi Pico toleruje tylko 3.3V. Konieczna więc była konwersja poziomów logicznych.



Rysunek 3: Układ nadajnika

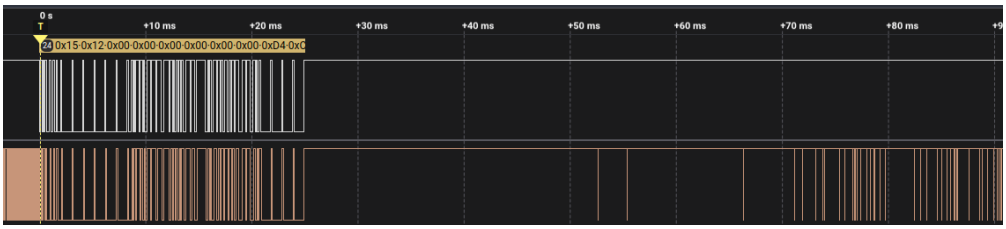


Rysunek 4: Układ odbiornika

Podwójne odwracanie sygnału przez tranzystory niweluje się. Na wejściu odbiornika dostajemy nieodwrócony sygnał z nadajnika.

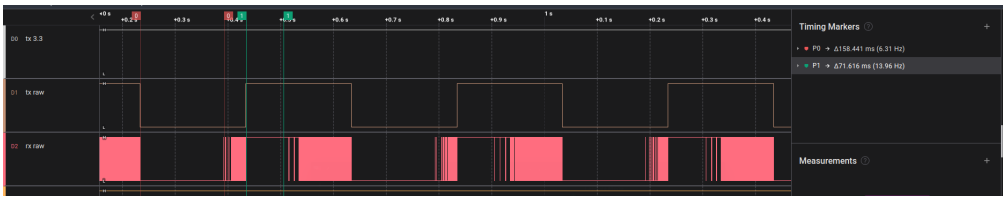
2.2 Warstwa programowa

Pierwszą naszą próbą było wykorzystanie wbudowanej komunikacji UART. Jednak okazało się że moduły te niezbyt dobrze przenoszą niezmieniający się sygnał (co widać poniżej).



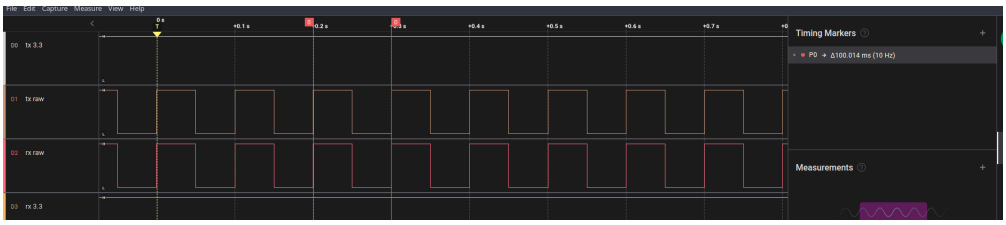
Rysunek 5: Wiadomość po stronie nadajnika/odbiornika

Po ok. 70 ms układ odbiornika zaczyna generować zakłócenia, które mogą być niepoprawnie interpretowane jako sygnały komunikacji.



Rysunek 6: Wyodrębnione zjawisko niestabilności

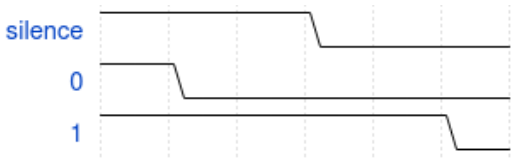
Po dokładnej analizie przebiegów odbiornika wynikło, że stan niski może być utrzymany przez ok. 160 ms, a stan wysoki przez około 72ms. Oznacza to że sygnał musi utrzymywać minimalną częstotliwość 14Hz.



Rysunek 7: Fala przenoszona bez zniekształceń

Przy ciągłych zmianach 10Hz okazuje się wystarczające.

Zaistniała potrzeba implementacji protokołu który utrzymywałby stałą częstotliwość fali nośnej. Przydatny w generacji takiego przebiegu jest PWM. Sterując wypełnieniem impulsu możemy przekazywać informacje binarne.



Rysunek 8: Protokół oparty o PWM

```
// Config
const int SUB_CYCLES = 6;
const int SUB_CYCLES_HIGH_SILENCE = 3;
// transmitter
const int SUB_CYCLES_HIGH_ZERO = 1;
const int SUB_CYCLES_HIGH_ONE = 5;
// receiver allowed
const int SUB_CYCLES_HIGH_ZERO_MAX = 2;
const int SUB_CYCLES_HIGH_ONE_MIN = 4;
```

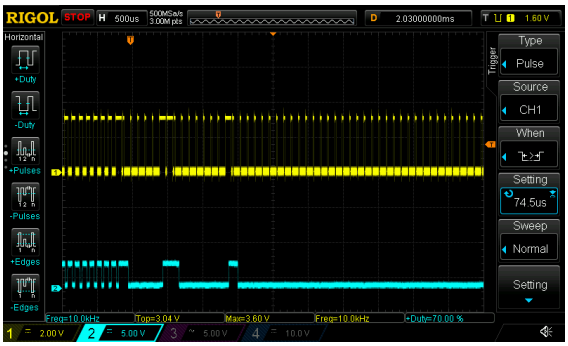
Rysunek 9: Konfiguracja protokołu

0.5 W obecnej wersji (konfigurowalne) zastosowaliśmy podziałkę $\frac{1}{6}$ wypełnienia PWM.

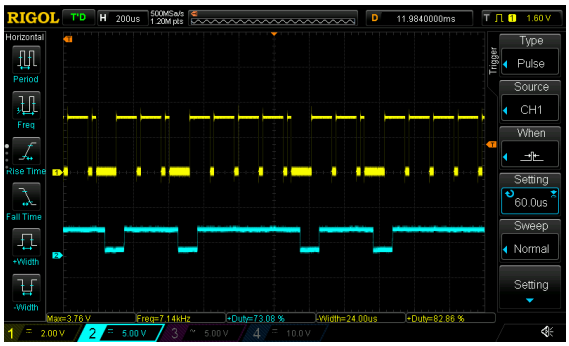
- Cisza to $\frac{3}{6}$ wypełnienia,
- 0 to $\frac{1}{6}$,
- 1 to $\frac{5}{6}$.

Odbiornik akceptuje 0 jako maksymalnie $\frac{2}{6}$ wypełnienia, a 1 jako minimalnie $\frac{4}{6}$ wypełnienia.

Producent określa maksymalną prędkość transmisji na 9600b/s (sugerowałoby to 9600Hz, jeżeli sygnalizowanie jest dwupoziomowe). Jednak generowane krótkie sygnały niekiedy są gubione przez nadajnik.

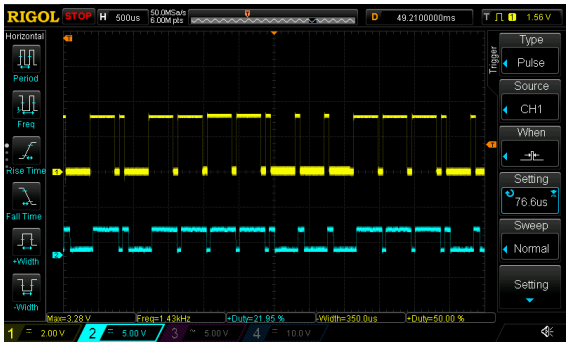


Rysunek 10: Gubienie impulsów 9600Hz



Rysunek 11: Gubienie impulsów 7200Hz

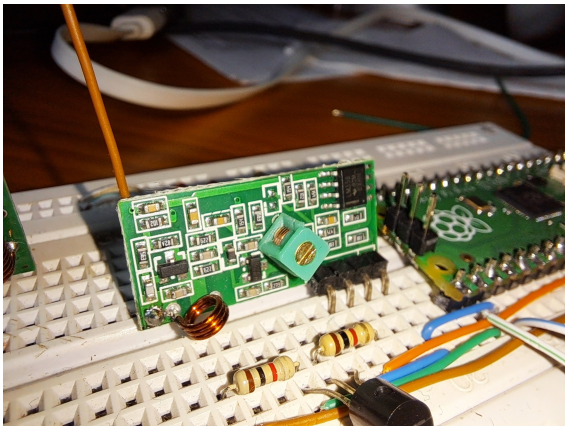
Rysunek 12: Gubienie impulsów 4800Hz



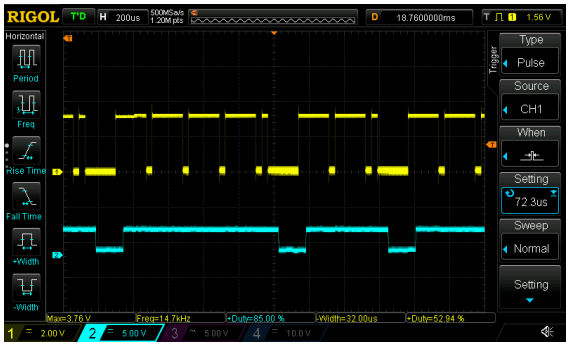
Rysunek 13: 2400Hz

Dopiero przy częstotliwości 2400Hz, wszystkie krótkie impulsy dotarły do odbiornika.

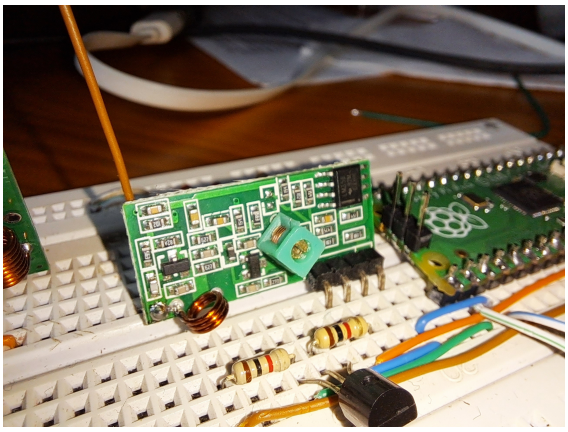
Strojenie Na płytce odbiornika dostępna jest cewka z możliwością dostrajania. Podjęliśmy próby jej nastawienia. Udało się osiągnąć szybkość transmisji 4800 b/s. Dla wyższych częstotliwości dostrajanie nie przyniosło efektów.



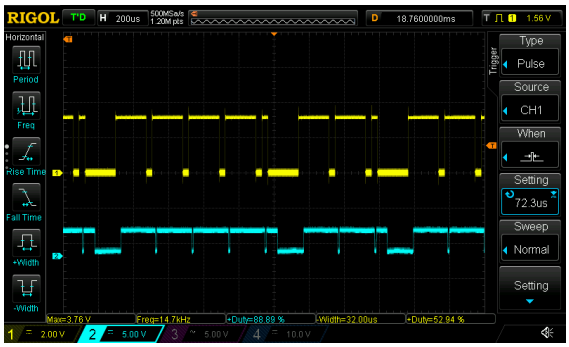
Rysunek 14: Cewka przed dostrajaniem



Rysunek 15: Przebieg 4800Hz przed dostrajaniem



Rysunek 16: Cewka po dostrojeniu



Rysunek 17: Przebieg 4800Hz po dostrojeniu

Zostaliśmy jednak przy transmisji 2400 b/s. Jest bardziej niezawodna, a szybkość nie ma dla nas wielkiego znaczenia. Nasza ramka danych ma rozmiar 16 bajtów. Przy 2400 b/s czas transmisji 1 ramki wynosi 53ms. Jest to bardzo mało w porównaniu do tego jak często będą wysyłane takie ramki.

2.2.1 Implementacja

Wysyłanie zostało zrealizowane z wykorzystaniem sprzętowego PWM i przerwania od jego przepełnienia. Częstotliwość PWM równa jest częstotliwości sygnalizowania w transmisji. Po wywołaniu przerwania przepełnienia, poziom wypełnienia ustawiany jest w zależności od następnego bitu danych. Jeżeli takiego nie ma, nadawana jest cisza.

```
void pwm_wrap_irq() {
    if (pwm_get_irq_status_mask() & (1 << slice_tx)) {
        pwm_clear_irq( slice_num: slice_tx);

        if (!tx_transfer) {
            return;
        }

        if (tx_bit_index == 8) {
            tx_bit_index = 0;
            tx_byte_index++;
            if (tx_byte_index == tx_byte_count) {
                tx_transfer = false;
                pwm_set_gpio_level( gpio: PIN_TX,  level: PWM_DUTY_SILENCE);
                return;
            }
        }

        uint bit = (tx_bytes[tx_byte_index] << tx_bit_index) & 0x80;
        pwm_set_gpio_level( gpio: PIN_TX,
                            level: bit ? PWM_DUTY_ONE : PWM_DUTY_ZERO);
        tx_bit_index++;
    }
}
```

Rysunek 18: Nadawanie PWM

Odbieranie natomiast wykorzystuje funkcję PWM mikrokontrolera RP2040, która umożliwia uruchomienie licznika w zależności od stanu pinu (obsługiwane są tylko piny nieparzyste). Używane jest także przerwanie na tym samym pinie, które wykrywa zbocze opadające (początek bitu). Zeruje ono licznik PWM, i czeka na kolejne zbocze opadające. Przy kolejnym zboczu wartość licznika jest interpretowana.

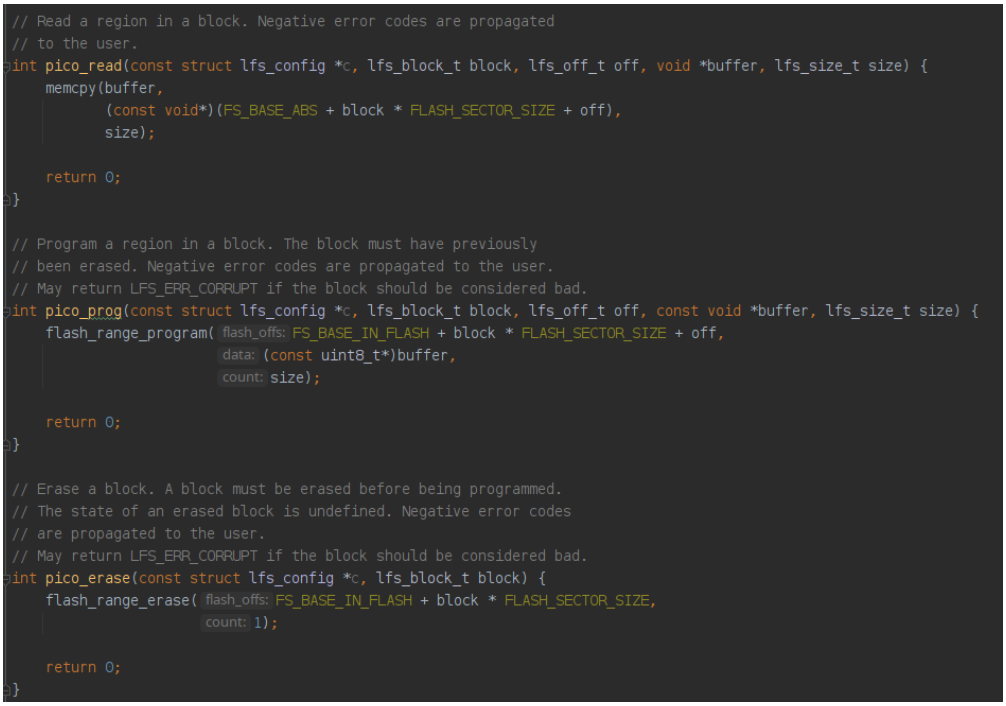
```
void rx_fall_callback(uint gpio, uint32_t events) {
    gpio_acknowledge_irq(gpio, event_mask: events);
    uint cnt = pwm_get_counter( slice_num: slice_rx);
    pwm_set_counter( slice_num: slice_rx, c: 0);

    uint64_t now = time_us_64();

    int bit;
    if (cnt < PWM_DUTY_ZERO_MAX) {
        bit = 0;
    }
    else if (cnt > PWM_DUTY_ONE_MIN) {
        bit = 1;
    }
    else {
        // silence
        bit = -1;
        if (now - last_good_bit > SPACING_ALLOWED_US_MAX) {
            // end of frame
            if (rx_byte_index > 0)
                cb( buf: rx_bytes, bytes: rx_byte_index);

            rx_byte_index = 0;
            rx_bit_index = 0;
        }
    }
```

Rysunek 19: Odbieranie PWM

Koniec ramki jest sygnalizowany przerwą w transmisji (podobnie do protokołu MODBUS). 10 znaków przerwy oznacza koniec ramki, przy czym nadajnik generuje 20 znaków przerwy.

3 System plików

Do implementacji przechowywania plików (głownie statycznych plików strony WWW) został użyty system plików LittleFS. Przy użyciu funkcji dostępu do pamięci Flash, zapisuje on dane w dostępnej pamięci na płytce Pi Pico.



Rysunek 20: Funkcje dostępowe do pamięci Flash, wymagane w konfiguracji LittleFS

4 WiFi

5 Serwer HTTP

Powstała własna implementacja serwera HTTP. Obsługuje on metody GET oraz POST. Interpretuje parametry URL jak i format application/x−www−form−urlencoded używany w formularzach. Używa LittleFS do wysyłania statycznych plików. Nacisk został położony na wygodny interfejs do obsługi serwera.

Rysunek 21: Wygodny interfejs serwera HTTP