

Pico Pagers

Urządzenia do powiadamiania klienta w restauracji Systemy Wbudowane

Jakub Kosmydel
Norbert Morawski

18 czerwca 2023

Spis treści

1	Wprowadzenie	2
2	Uruchomienie	2
2.1	Konfiguracja sieci WiFi	2
2.1.1	Opis	2
2.1.2	Graf	3
2.2	Parowanie klienta	4
2.2.1	Opis	4
2.2.2	Graf	5
2.3	Zarządzanie klientem	6
2.3.1	Opis	6
2.3.2	Graf	7
3	Protokół komunikacyjny	8
3.1	Warstwa sprzętowa	8
3.2	Warstwa programowa	9
3.2.1	Implementacja	12
4	System plików	13
5	WiFi	14
6	Serwer HTTP	14
7	Kryptografia	15
8	Uruchomienie aplikacji	15

1 Wprowadzenie

W ramach projektu został przygotowany system Pico Pagers, który został zaprojektowany z myślą o automatyzacji procesu obsługi klienta w restauracjach. System składa się z dwóch urządzeń: *Pagers Server*, który jest odpowiedzialny za zarządzanie klientami oraz *Pagers Client*, który informuje gości o statusie ich zamówienia.

2 Uruchomienie

W celu skorzystania z systemu należy go początkowo skonfigurować. Do uruchomienia potrzebne są:

- sieć WiFi (2.4 GHz),
- urządzenie (np. telefon, komputer) połączone z tą siecią WiFi.

2.1 Konfiguracja sieci WiFi

2.1.1 Opis

Konfiguracja urządzenia przebiega w następujący sposób:

1. Podłączamy *Pagers Server* do zasilania.
2. Na dowolnym urządzeniu elektronicznym wyszukujemy sieć WiFi o nazwie *pgers-server*, łączymy się z nią podając hasło *password*.
3. Używając przeglądarki internetowej wchodzimy na adres <http://192.168.4.1>.
4. Przechodzimy do sekcji *Setup the WiFi connection*
5. Wyszukujemy dostępne sieci WiFi używając przycisku *Initiate Scan*.

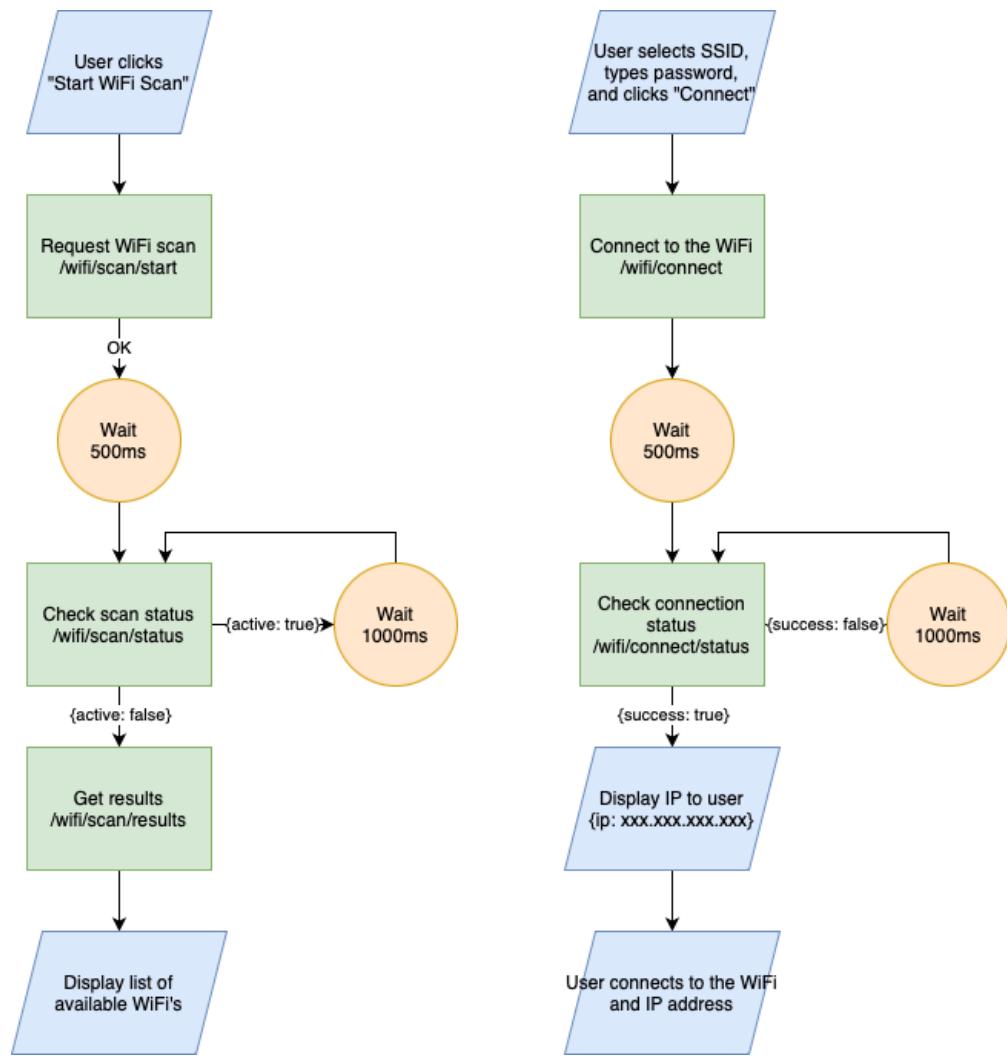
Setup the WiFi connection

Initiate Scan

Rysunek 1: Skanowanie sieci WiFi

6. Wybieramy naszą sieć WiFi oraz podajemy do niej hasło.
7. Klikamy *Connect* w celu połączenia się z wybraną siecią.
8. Kopiujemy podany adres urządzenia.
9. Łączymy swoje urządzenie z siecią WiFi.
10. Wchodzimy pod skopiowany adres urządzenia, gdzie znajdziemy panel zarządzania klientami.

2.1.2 Graf



Rysunek 2: Proces konfiguracji WiFi

2.2 Parowanie klienta

2.2.1 Opis

Dodawanie nowego *Pagers Client* przebiega w następujący sposób:

1. W panelu zarządzania przechodzimy do sekcji *Pair pager*

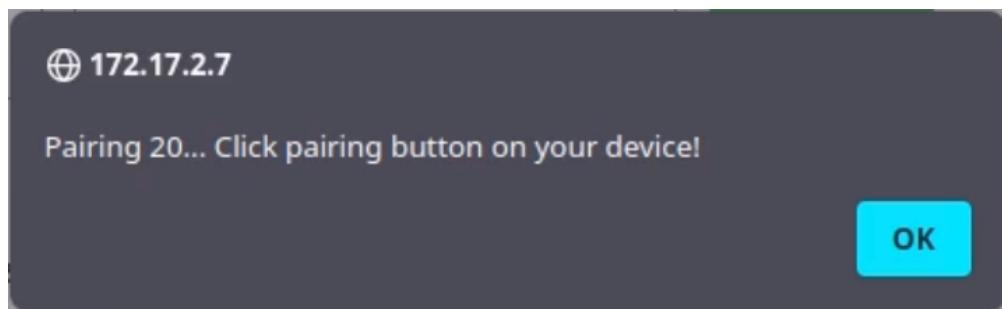
Pair pager

To pair a new pager you need to press pairing button on the pager, and then click Pair button below.

id	▼	Pair
----	---	------

Rysunek 3: Parowanie nowego klienta

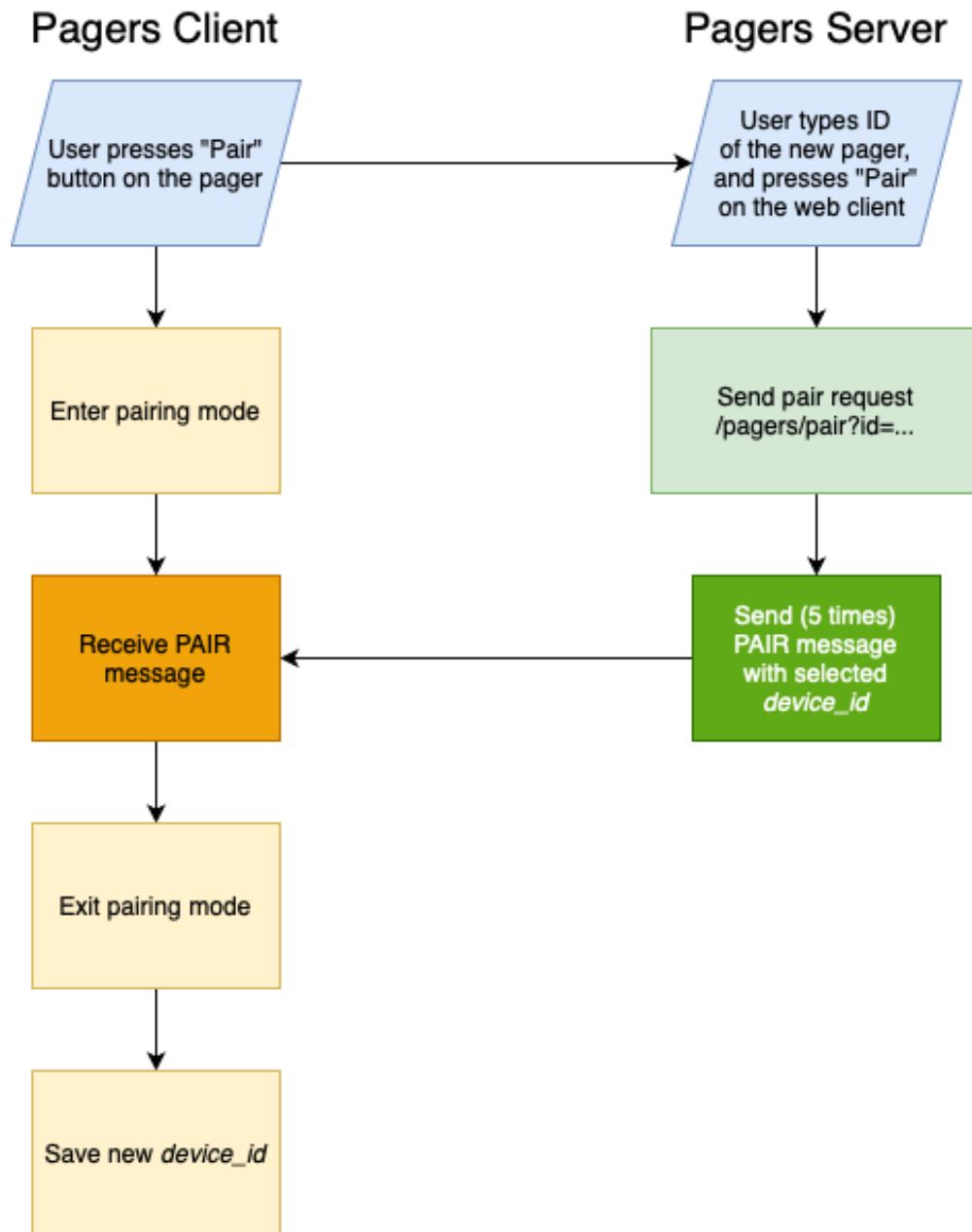
2. Wpisujemy *id* nowego klienta.
3. Klikamy przycisk *Pair*. Pozostawiamy wyświetlony alert.



Rysunek 4: Alert potwierdzający

4. Na urządzeniu *Pagers Client* wciskamy przycisk służący do parowania. Urządzenie potwierdzi, że jest w trybie parowania, migając żółtą diodą.
5. Klikamy przycisk *OK* na alercie, by ukończyć parowanie.

2.2.2 Graf

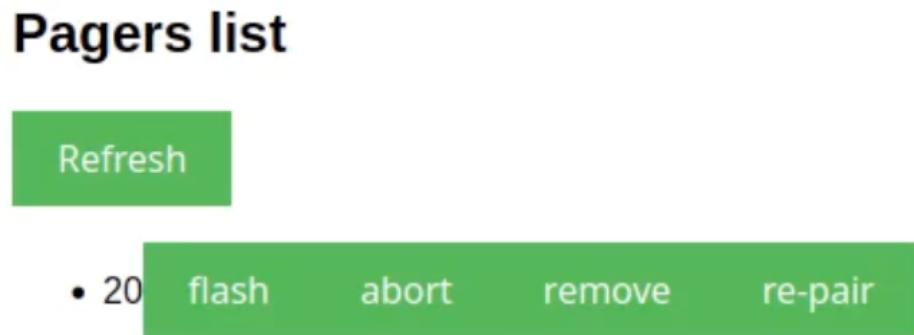


Rysunek 5: Proces parowania klienta

2.3 Zarządzanie klientem

2.3.1 Opis

Panel zarządzania wyświetla listę klientów oraz umożliwia nimi sterowanie.

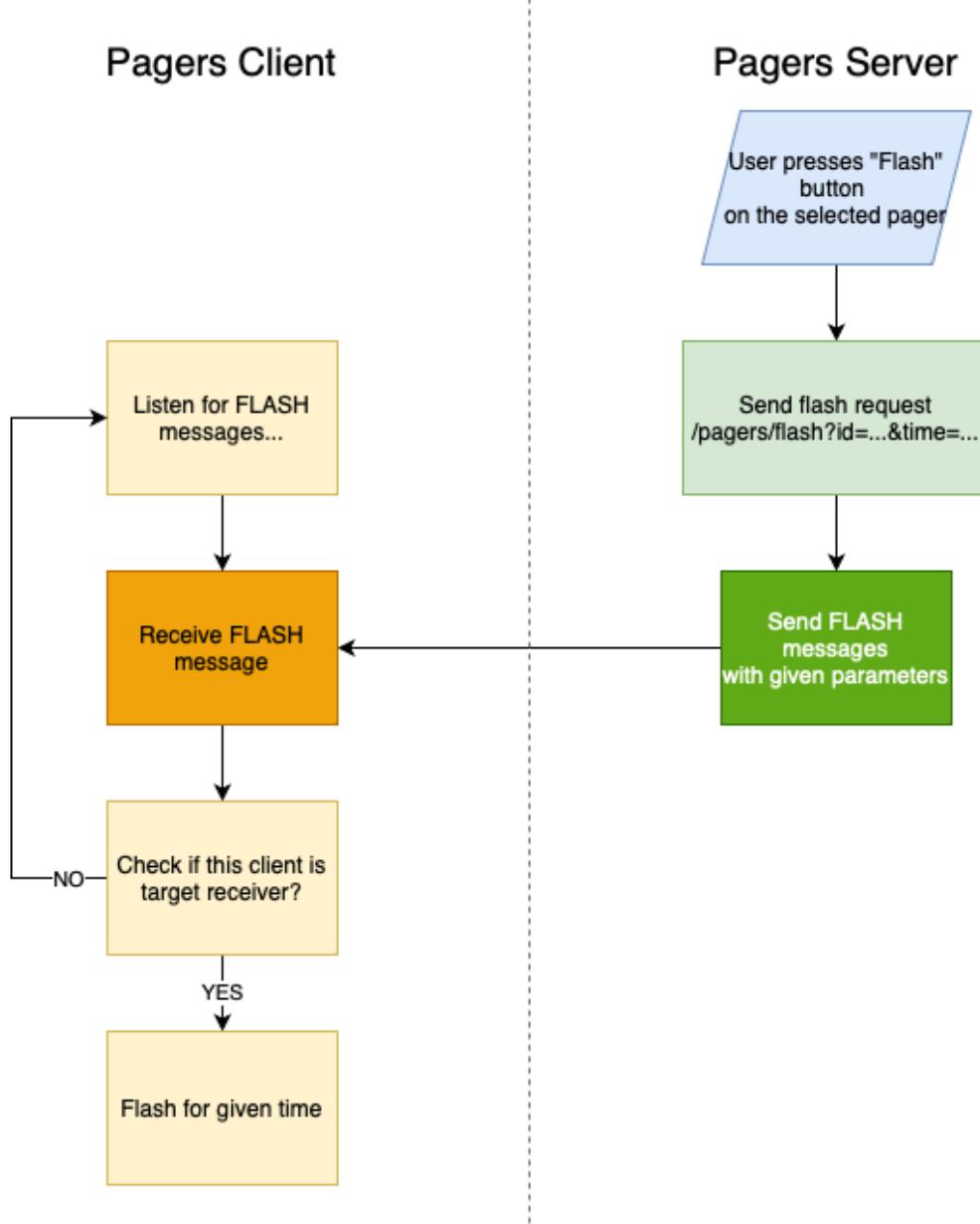


Rysunek 6: Lista sparowanych urządzeń

Dostępne akcje:

- *flash* - wywołanie klienta,
- *abort* - przerwanie wywoływania,
- *remove* - usunięcie urządzenia,
- *re-pair* - ponowne parowanie.

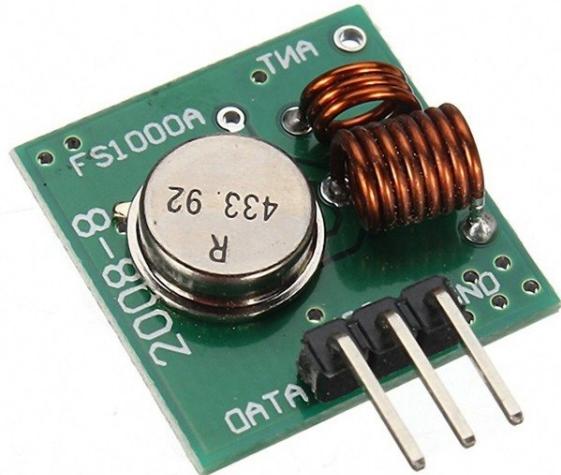
2.3.2 Graf



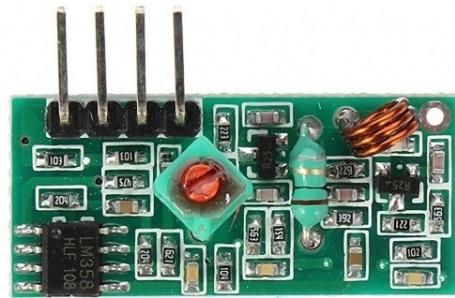
Rysunek 7: Proces wywołwania klienta

3 Protokół komunikacyjny

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



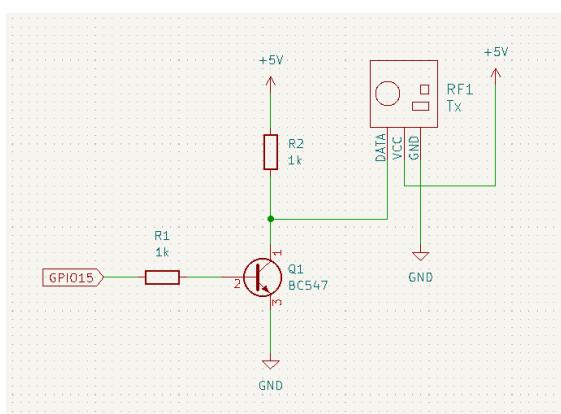
Rysunek 8: Nadajnik



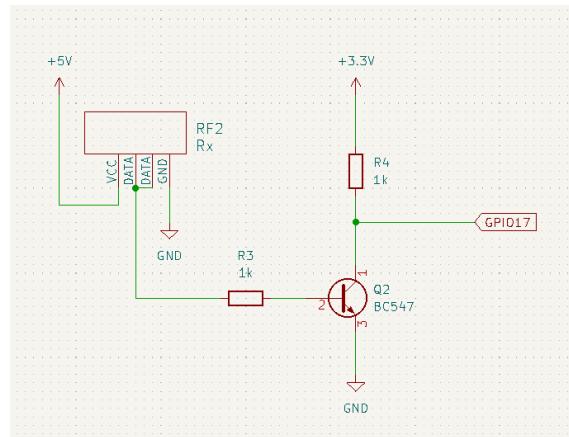
Rysunek 9: Odbiornik

3.1 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 10: Układ nadajnika



Rysunek 11: Układ odbiornika

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

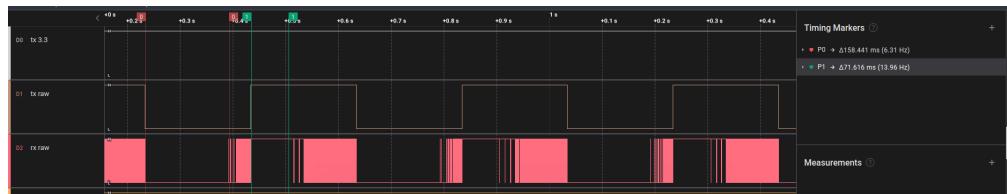
3.2 Warstwa programowa

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



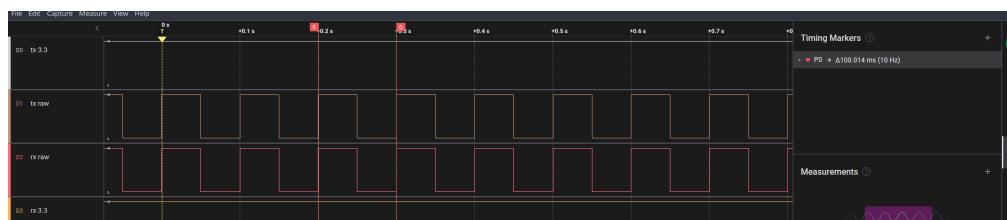
Rysunek 12: Wiadomość po stronie nadajnika/odbiornika

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



Rysunek 13: Wyodrębnione zjawisko niestabilności

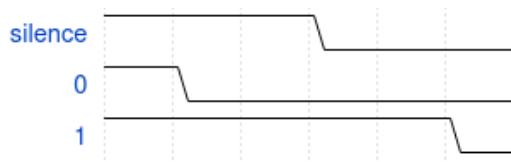
Po dokładnej analizie przebiegów odbiornika wynikło, że stan niski może być utrzymywany przez ok. 160 ms, a stan wysoki przez około 72ms. Oznacza to że sygnał musi utrzymywać minimalną częstotliwość 14Hz.



Rysunek 14: 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 15: 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;
```

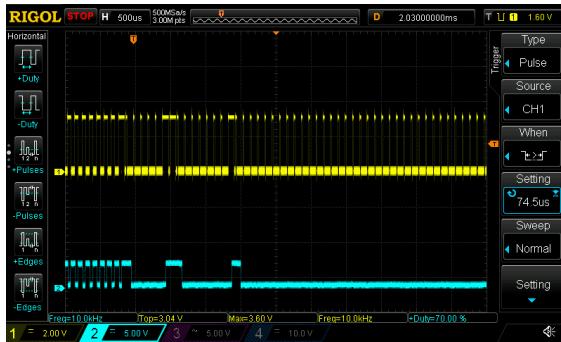
Rysunek 16: Konfiguracja protokołu

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 17: Gubienie impulsów 9600Hz



Rysunek 18: Gubienie impulsów 7200Hz



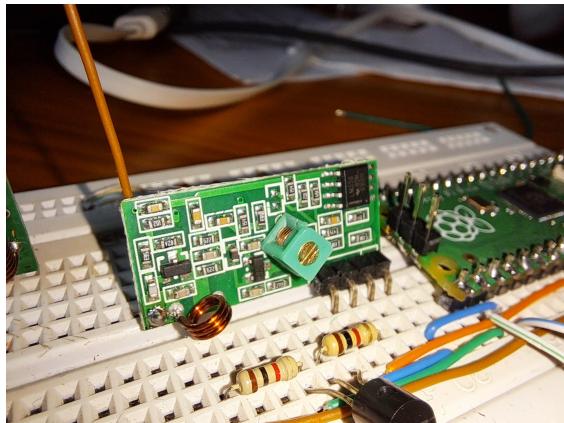
Rysunek 19: Gubienie impulsów 4800Hz



Rysunek 20: 2400Hz

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

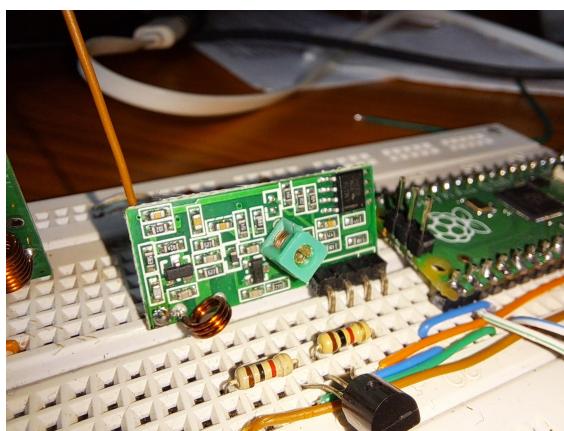
Strojenie Na płytce odbiornika dostępna jest cewka z możliwością dostrajania. Podjęliśmy próby jej nastawienia. Udało się osiągnąć szybkość transmisji 4800 b/s. Dla wyższych częstotliwości dostrajanie nie przyniosło efektów.



Rysunek 21: Cewka przed dostrajaniem



Rysunek 22: Przebieg 4800Hz przed dostrajaniem



Rysunek 23: Cewka po dostrojeniu



Rysunek 24: 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.

3.2.1 Implementacja

Wysyłanie zostało zrealizowane z wykorzystaniem sprzętowego PWM i przerwania od jego przepelenienia. Częstotliwość PWM równa jest częstotliwości sygnalizowania w transmisji. Po wywołaniu przerwania przepelenienia, 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 25: 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 26: Odbieranie PWM

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

4 System plików

Do implementacji przechowywania plików (głównie 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.

```

// Read a region in a block. Negative error codes are propagated
// to the user.
int pico_read(const struct lfs_config *c, lfs_block_t block, lfs_off_t off, void *buffer, lfs_size_t size) {
    memcpy(buffer,
           (const void*)(FS_BASE_ABS + block * FLASH_SECTOR_SIZE + off),
           size);

    return 0;
}

// Program a region in a block. The block must have previously
// been erased. Negative error codes are propagated to the user.
// May return LFS_ERR_CORRUPT if the block should be considered bad.
int pico_prog(const struct lfs_config *c, lfs_block_t block, lfs_off_t off, const void *buffer, lfs_size_t size) {
    flash_range_program( flash_offs: FS_BASE_IN_FLASH + block * FLASH_SECTOR_SIZE + off,
                         data: (const uint8_t*)buffer,
                         count: size);

    return 0;
}

// Erase a block. A block must be erased before being programmed.
// The state of an erased block is undefined. Negative error codes
// are propagated to the user.
// May return LFS_ERR_CORRUPT if the block should be considered bad.
int pico_erase(const struct lfs_config *c, lfs_block_t block) {
    flash_range_erase( flash_offs: FS_BASE_IN_FLASH + block * FLASH_SECTOR_SIZE,
                       count: 1);

    return 0;
}

```

Rysunek 27: Funkcje dostępne do pamięci Flash, wymagane w konfiguracji LittleFS

5 WiFi

Sterowanie systemem odbywa się za pomocą przeglądarki internetowej. W celu skorzystania z urządzenia należy skonfigurować połączenie z siecią WiFi.

6 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.

```

HttpServer server;
server.set_cb_arg( arg: nullptr);
server.start( port: 80);
server.static_content( lfs: &lfs, fs_path: "/static");
server.on( method: Method::GET, path: "/root", callback: root);
server.on( method: Method::GET, path: "/json", callback: json_test_page);
server.on( method: Method::GET, path: "/form", callback: form_test_page);
server.on( method: Method::POST, path: "/form", callback: form_test_page);

```

Rysunek 28: Wygodny interfejs serwera HTTP

7 Kryptografia

Aby oferować podstawowy poziom bezpieczeństwa, nasze rozwiązanie szyfruje wiadomości szyfrem RSA używając 32-bitowego klucza. Przesyłana jest także zaszyfrowana suma kontrolna do sprawdzania poprawności zdekodowanych danych. Funkcjonuje to analogicznie do podpisu cyfrowego

Algorytm 1 Szyfrowanie

```
1: (n, d) – klucz prywatny
2: function SZYFRUJ(dlugosc, data, enc)
3:   for i  $\leftarrow$  0 to dlugosc – 1 do
4:     enc[i]  $\equiv$  data[i]d (mod n)
5:   end for
6: end function
```

Algorytm 2 Deszyfrowanie

```
1: (n, e) – klucz publiczny
2: function ODSZYFRUJ(dlugosc, data, enc)
3:   for i  $\leftarrow$  0 to dlugosc – 1 do
4:     enc[i]  $\equiv$  data[i]e (mod n)
5:   end for
6: end function
```

Algorytm dzieli wejściowe dane na 16-bitowe bloki i szyfruje je do bloków 32-bitowych. Składowe kluczy mają po 32 bity. Wiadomość wydłuża się dwukrotnie. Teraz jej długość to 32 bajty.

8 Uruchomienie aplikacji

Wraz z programami dostarczony jest plik README.md w którym opisane jest jak skompilować i wgrać aplikacje do systemu mikroprocesorowego.