

Zamek otwierany pastylkami iButton

Zofia Semczyszyn Dawid Hebda

17.01.2025

Spis treści

1 Opis projektu	2
2 Przygotowanie do użycia	2
2.1 Użyte elementy elektroniczne	2
2.2 Połączenie modułów	3
2.3 Ustawienia UART	4
3 Działanie aplikacji	4
3.1 Konfiguracja początkowa	4
3.2 Główne działanie - WORK_MODE	4
3.3 Wprowadzanie pastylek - tryb CONFIG_MODE	4
3.4 Wyświetlanie historii dostępu	5
3.5 Czyszczenie pamięci Flash	5
4 Algorytm aplikacji	5
4.1 Działanie sprawdzania trybu pracy aplikacji	7
4.1.1 Inicjalizacja trybu aplikacji	7
4.1.2 Sprawdzenie trybu aplikacji	7
4.2 Delay	7
4.2.1 Inicjalizacja SysTick	7
4.2.2 Oczekanie konkretnej ilości mikrosekund	7
4.3 Komunikacja 1-Wire	7
4.3.1 Sprawdzenie czy pastylka jest przyłożona	7
4.3.2 Wysłanie bitu	7
4.3.3 Odczytanie bitu	8
4.3.4 Odczytanie numeru seryjnego	8
4.4 Otwarcie zamka	8
4.5 RTC	8
4.5.1 Inicjalizacja RTC	8
4.5.2 Pobranie obecnej godziny	8
4.6 UART	8
4.7 Flash	8
4.7.1 Inicjalizacja pamięci Flash	9
4.7.2 Dodanie iButtona	9
4.7.3 Sprawdzenie rejestracji	9
4.7.4 Dodanie wpisu do historii	9
4.7.5 Wyświetlenie historii	9
4.7.6 Reset pamięci	9
4.7.7 Odczyt i zapis do pamięci	9
4.7.8 Funkcje pomocnicze	9
4.7.9 Dodatkowe informacje o pamięci Flash	10

1 Opis projektu

Celem projektu było stworzenie zamka otwieranego pastylkami iButton. Zamek rejestruje dostępy (data i godzina) do pamięci Flash, które można przejrzeć na żądanie przez UART. Aplikacja w pewnej konfiguracji pozwala na zarejestrowanie do pamięci Flash pastylek DS1996, które mogą otwierać zamek.

2 Przygotowanie do użycia

2.1 Użyte elementy elektroniczne

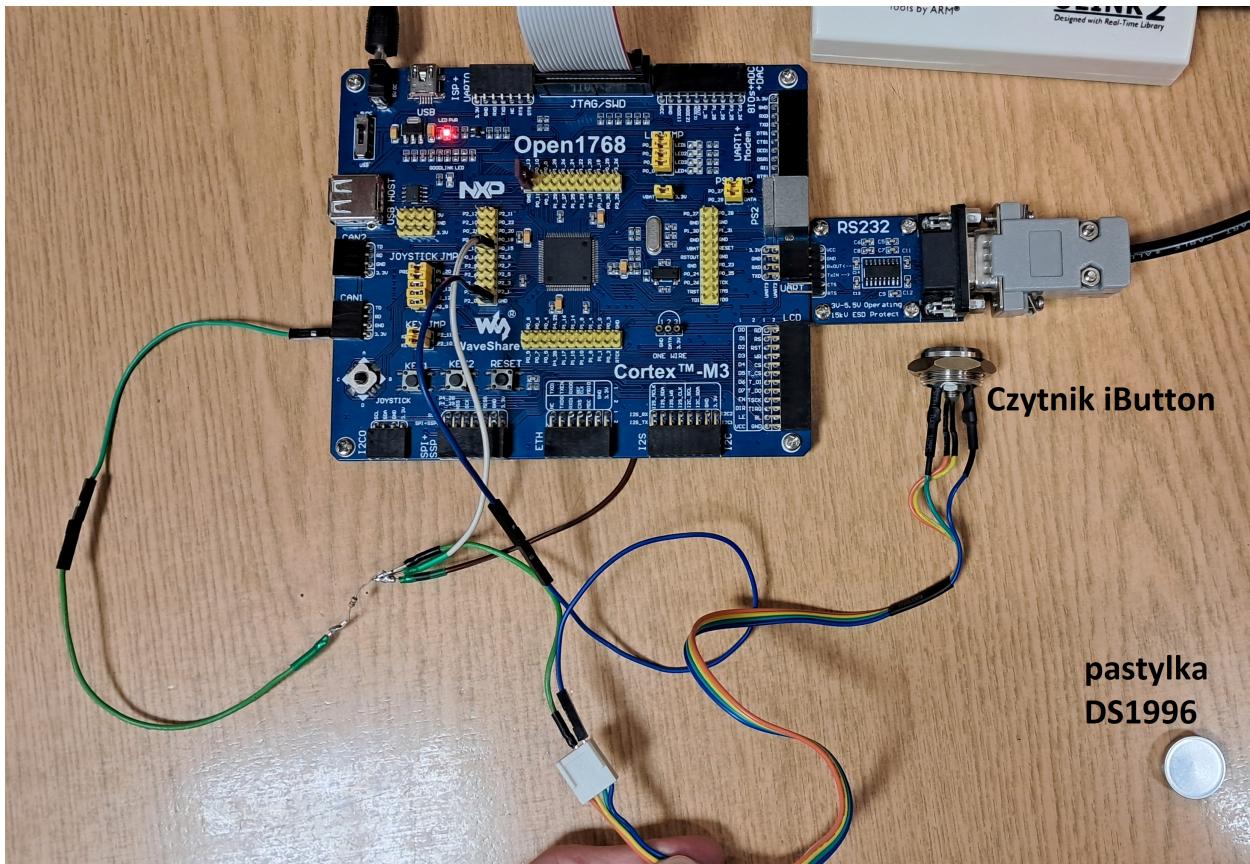
Przy tworzeniu aplikacji zamka otwieranego pastylkami iButton skorzystano z:

- **czytnik iButton** - do odczytania numeru seryjnego pastylek poprzez protokół 1-Wier,
- **pastylki DS1996** - do otwierania zamka,
- **modułu UART RS232** - do komunikacji UART między mikrokontrolerem, a komputerem,
- mikrokontrolera LPC 1786, rezystor około $5\text{ k}\Omega$, zworka.

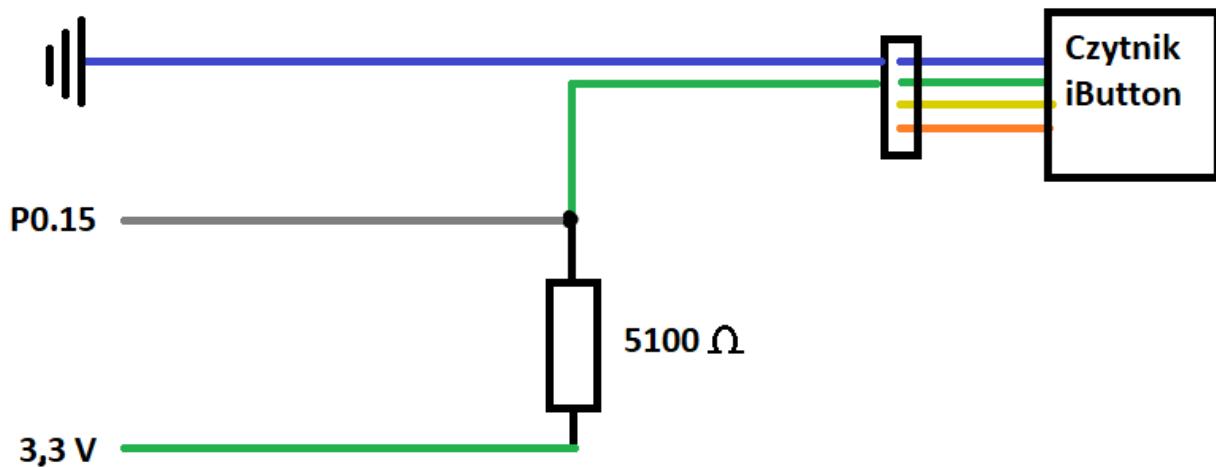
W obrębie mikrokontrolera wykorzystano peryferia:

- **LED** - migają wtedy, gdy zamek miałby się otworzyć,
- **UART2** - do odczytania historii dostępów oraz jako wyjście debugujące, żeby wiedzieć co się dzieje z zamkiem,
- **RTC** - do pobrania aktualnej godziny do zapisu historii dostępów,
- **SysTick_timer** - do odmierzania czasu w komunikacji 1-Wier,
- **pamięć Flash** - do zapisu danych o pastylkach, które mają dostęp oraz historii dostępów do zamka.

2.2 Połączenie modułów



Rysunek 1: Zdjęcie podłączenia modułów.



Rysunek 2: Uproszczony schemat podłączenia czytnika iButton.

Na rysunku 1 przedstawiono jak podłączyć wymienione elementy:

- Czytnik iButton: Do niebieskiego kabla należy podłączyć masę, a do zielonego linię danych (port P0.15) z rezystorem pull-up o wartości około $5\text{ k}\Omega$ (w tym przypadku użyto $5,1\text{ k}\Omega$) podłączonym do zasilania, jak pokazano na [rysunku 2](#). Kabel żółty i pomarańczowy odpowiadają za diodę, która nie jest wykorzystywana.
- Pastylki DS1996: W trakcie korzystania z aplikacji, pastylki będą przykładowane do czytnika.
- Moduł UART RS232: Należy wpiąć do portów przeznaczonych na UART2, pozostawiając niewpięte piny CTS i RTS.
- Tryb dodawania nowych pastylek: Aby przejść w tryb dodawania nowych pastylek, które mogą otworzyć zamek, należy wpiąć zworkę, która połączy port P2.13 z masą.

2.3 Ustawienia UART

Aby odbierać komunikację po UART, należy otworzyć program Tera Term i ustawić prędkość transmisji na 9600 Bd.

3 Działanie aplikacji

3.1 Konfiguracja początkowa

Po włączeniu aplikacja zapyta po UART użytkownika o aktualną datę:

`Enter the day in YYMMDD format:`

Po wpisaniu 6 cyfr zapyta się o aktualną godzinę:

`Enter the time in HHMMSS format:`

Po wpisaniu kolejnych 6 cyfr aplikacja przejdzie do głównego działania.

3.2 Główne działanie - WORK_MODE

Aplikacja co 1 s próbuje odczytać numer przyłożonej pastylki. Użytkownik musi przyłożyć do czytnika iButton pastylkę i odczekać chwilę, aby aplikacja miała szansę ją wykryć. Po poprawnym odczytaniu numeru pastylki, sprawdzone zostaje czy ta pastylka jest wprowadzona jako mająca uprawnienia do otwarcia zamka. Jeśli ma uprawnienia, to w ramach imitacji otwierania zamka zamigają diody na płytce, a dostęp zostanie zapisany w pamięci z obecną datą. Jeśli nie ma uprawnień zostanie wysłany komunikat o tym po UART.

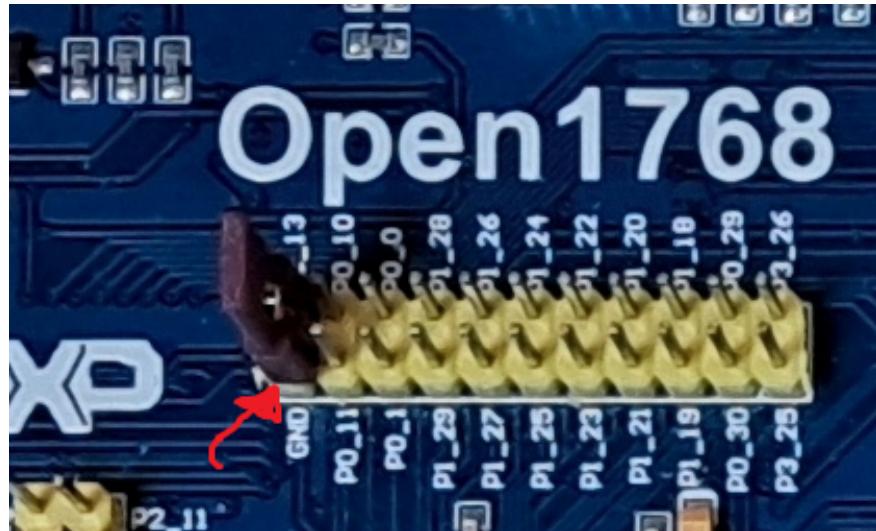
W przypadku wystąpienia sytuacji takich jak:

- brak przyłożonej pastylki,
- niezgadzająca się suma kontrolna odczytanego numeru pastylki,

użytkownik zostanie poinformowany o tym przez komunikat wysłany po UART, należy wtedy ponownie przyłożyć pastylkę.

3.3 Wprowadzanie pastylek - tryb CONFIG_MODE

W celu wprowadzenia pastylki, do bazy pastylek mających uprawnienia do otwierania zamka, należy wpiąć zworkę łączącą port P2.13 z masą, jak pokazano na [rysunku 3](#).



Rysunek 3: Przybliżenie na miejsce na płytce, gdzie należy wpiąć zworkę, żeby móc dodawać nowe pastylki.

Następnie należy przyłożyć pastylkę do czytnika iButton i odczekać chwilę, aby aplikacja miała szanse ją wykryć. Po poprawnym odczytaniu numeru pastylki, sprawdzone zostaje czy ta pastylka jest już wprowadzona jako mająca uprawnienia do otworzenia zamka. Jeśli miała już wcześniej uprawnienia, to komunikat o tym zostanie wysłany po UART. Jeśli nie miała uprawnień zostanie dodana do bazy, komunikat o pomyślnym dodaniu zostanie wysłany po UART.

W tej konfiguracji mogą wystąpić takie same komunikaty o niepowodzeniu, jak w [trybie pracy zamka 3.2](#). Dodatkowo użytkownik może otrzymać komunikat wysłany po UART, że przekroczony został limit możliwych do zarejestrowania pastylek. Jest on ustawiony na 32 pastylki, co wydaje się być rozsądne, ponieważ w pracowni nie ma więcej pastylek. Jeśli jednak byłaby kiedyś potrzeba zarejestrowania większej ilości pastylek, można to przeprogramować.

3.4 Wyświetlanie historii dostępu

Po przyciśnięciu przycisku **KEY1** po UART zostanie wysłana historia jakie pastylki kiedy otwierały zamek.

3.5 Czyszczenie pamięci Flash

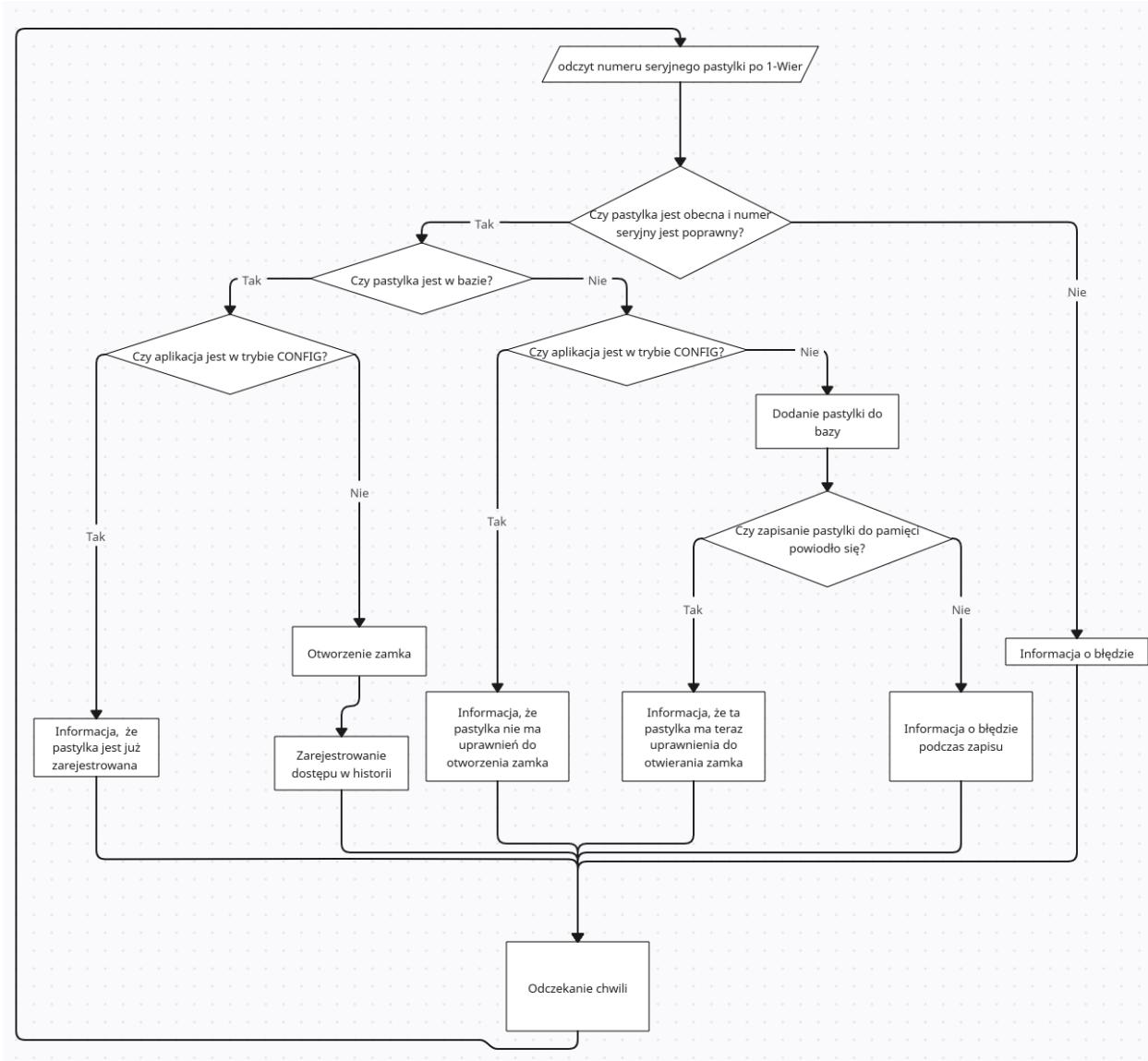
Aby wyczyścić pamięć Flash, należy umieścić **zworkę** łączącą port P2.13 z masą, zgodnie z instrukcją przedstawioną na [rysunku 3](#), a następnie nacisnąć przycisk **KEY2**. Po wykonaniu tej czynności pamięć zostanie wyczyszczona, co spowoduje usunięcie wszystkich zapisanych pastylek z bazy oraz wyzerowanie historii.

4 Algorytm aplikacji

Plik main podzielony jest na trzy części: uruchamianą przy starcie, działającą ciągle i uruchamianą gdy przychodzi przerwanie, od któregoś z przycisków.

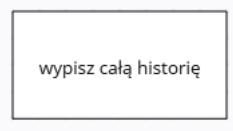
Przy starcie wywoływane są inicjalizacje: SystemInit, UART, SysTick, trybu pracy, zamka, ustawiane są przerwania od przycisków: KEY1 i KEY2, pobierany jest od użytkownika obecny czas i inicjalizowany jest RTC, a także sprawdzane, czy w pamięci Flash znajdują się zapisane dane.

Część działająca ciągle steruje główną logiką aplikacji: sprawdza czy pastylka została przyłożona, odczytana poprawnie, czy jest w bazie i czy aplikacja jest w [trybie zwykłej pracy](#), czy w [trybie wprowadzania pastylek](#). Na podstawie tych danych decyduje, którą funkcję uruchomić i jakie komunikaty wysłać po UART. Schemat działania algorytmu został przedstawiony na [rysunku 4](#).



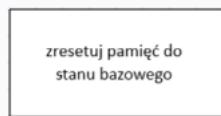
Rysunek 4: Schemat logiki aplikacji

Przerwanie pochodzące od przycisku KEY1 uruchamia funkcję wypisującą historię otworzeń zamka. Schemat działania algorytmu został przedstawiony na [rysunku 5](#).



Rysunek 5: Schemat działania przewiania od przycisku KEY1.

Przerwanie pochodzące od przycisku KEY2 w [trybie zwykłej pracy](#) nic nie robi, a w [trybie wprowadzania pastylek](#) resetuje stan pamięci do stanu bazowego (brak danych o pastylkach). Schemat działania algorytmu został przedstawiony na [rysunku 6](#).



Rysunek 6: Schemat działania przewiania od przycisku KEY2.

4.1 Działanie sprawdzania trybu pracy aplikacji

4.1.1 Inicjalizacja trybu aplikacji

Podczas inicjalizacji trybu aplikacji port 2.13 ustawiany jest na GPIO z trybem pullup.

4.1.2 Sprawdzenie trybu aplikacji

Sprawdzane jest czy port 2.13 jest w stanie wysokim, co oznacza, że zworka nie jest wpięta i jest to tryb pracy. Jeśli port 2.13 jest w stanie niskim, to oznacza, że zworka jest wpięta i jest to tryb wprowadzania pastylek.

4.2 Delay

4.2.1 Inicjalizacja SysTick

Ustawiane jest przerwanie od SysTick jedno na mikrosekundę. W SysTick_Handler zliczana jest ilość przerwań - tików od SysTick.

4.2.2 Oczekanie konkretnej ilości mikrosekund

Oczekanie konkretnej ilości mikrosekund polega na zapisaniu obecnej ilości tików i ilości tików jakie będą po upłynięciu podanej ilości mikrosekund. Następnie sprawdzany jest warunek czy upłynął już podany czas. Warunek jest tak skonstruowany, że jest odporny na przepełnienie zmiennej przechowującej ilość tików, ponieważ występuje ono raz na 71 minut i 35 sekund.

4.3 Komunikacja 1-Wire

Protokół 1-Wire umożliwia komunikację między urządzeniami za pomocą tylko jednego przewodu danych (plus przewód masy). Linia danych może być w stanie wysokim dzięki rezystorowi typu pullup albo ustawiona na stan niski przez mikrokontroler lub DS1996. Każda komunikacja opiera się na ustaleniu pinu mikrokontrolera jako wyjścia, ustaleniu pinu na stan niski i oczekaniu określonej ilości czasu. Następnie pin zostaje przełączony jako wejście, oczekuje się konkretny czas i czyta stan pinu. Na koniec oczekiwany jest czas potrzebny na wrócenie linii do stanu gotowego na następną instrukcję.

4.3.1 Sprawdzenie czy pastylka jest przyłożona

W celu sprawdzenie czy pastylka jest przyłożona wysyłany jest sygnał reset, czyli mikrokontroler ściąga linie danych do stanu niskiego przez $480 \mu\text{s}$, przychodzi na input oczekuje $70 \mu\text{s}$ i sprawdza czy DS1996 ustało stan niski - sygnał obecności. Oczekiwany jest $410 \mu\text{s}$ i zwracany jest czy wystąpił sygnał obecności.

4.3.2 Wysłanie bitu

Wysłanie "1": mikrokontroler ściąga linie danych do stanu niskiego przez $10 \mu\text{s}$, przechodzi na input i oczekuje $73 \mu\text{s}$.

Wysłanie "0": mikrokontroler ściąga linie danych do stanu niskiego przez $65 \mu\text{s}$, przechodzi na input i oczekuje $18 \mu\text{s}$.

4.3.3 Odczytanie bitu

Odczytanie bitu: mikrokontroler ściąga linie danych do stanu niskiego przez $3 \mu\text{s}$, przychodzi na input oczekuje $10 \mu\text{s}$ i sprawdza czy DS1996 ustawiło stan niski czy pozostało w wysokim. Na koniec oczekuje $65 \mu\text{s}$.

4.3.4 Odczytanie numeru seryjnego

Odczytanie numeru seryjnego polega na sprawdzeniu czy DS1996 jest przyłożony, jeśli tak wysyłane jest polecenie w formie jednego bajtu "0x33"(osiem wysłań bitu), które oznacza odczytaj ROM. Następnie czytane jest osiem bajtów numeru seryjnego. Na podstawie pierwszych siedmiu odczytanych bajtów obliczana jest suma kontrolna CRC i porównywana jest z ósmym bajtem.

4.4 Otwarcie zamka

W obecnej formie otwarcie zamka polega na miganiu diodami w pewnej sekwencji przez 4 sekundy. W przyszłości można rozbudować aplikacje, aby otwarcie zamka robił coś innego na przykład sterował servo lub za pomocą przekaźnika zwalniać zamek elektromagnetyczny. Aby zmienić działanie zamka wystarczy podmienić ciało funkcji inicjalizującej zamek i otWireającej zamek.

4.5 RTC

4.5.1 Inicjalizacja RTC

Włączane jest zasilanie RTC i sam zegar. Podczas uruchomienia programu użytkownik podaje obecną datę i godzinę, które zostają ustawione na RTC. Włączana jest aktualizacja czasu w RTC.

4.5.2 Pobranie obecnej godziny

Z RTC pobierana jest obecna data i zwracana jako wynik funkcji.

4.6 UART

Podczas inicjalizacji UART: włączany jest UART2, ustawiane są piny na których będzie nadawał i odbierał, prędkość nadawania 9600 Bd oraz parametry ramki: 8-bitów danych, 1-bit stopu, bark bitu parzystości.

4.7 Flash

Pamięć Flash to rodzaj nieulotnej pamięci półprzewodnikowej, która umożliwia przechowywanie danych nawet po odłączeniu zasilania. Pamięć Flash dzieli się na sektory, które mogą być niezależnie kasowane, zapisywane lub odczytywane.

Operacje na pamięci Flash opierają się na dwóch kluczowych etapach:

- **Przygotowanie sektora** – sektor musi zostać odpowiednio przygotowany przed zapisem.
- **Czyszczenie sektora** – przed zapisaniem nowych danych sektor musi być całkowicie wyczyszczony.

Dane do pamięci Flash zapisywane są w blokach o rozmiarze, równym wielokrotności 256 bajtów.

W ramach tego projektu pamięć Flash jest wykorzystywana do:

- Przechowywania listy zarejestrowanych iButtonów.
- Rejestrowania historii dostępu.
- Identyfikowania, za pomocą znacznika, czy przy starcie programu w pamięci są prawidłowe dane, czyli zapisane przez aplikacje zamka.
- Przechowywanie informacji o liczbie zarejestrowanych iButtonów oraz liczbie wpisów w historii.

4.7.1 Inicjalizacja pamięci Flash

Funkcja `initialize_flash()` sprawdza, czy pamięć Flash została już wcześniej zainicjalizowana, porównując znacznik zapisany w konkretnym miejscu w pamięci z wartością, jakiej oczekujemy.

- W przypadku braku zgadzającego się znacznika, pamięć jest przygotowywana do użycia. Wykorzystywane sektory są czyszczone. Następnie zapisywane są domyślne wartości, takie jak znacznik, liczba zarejestrowanych iButtonów oraz liczba wpisów w historii (ustawione na 0).
- Jeśli znacznik ma prawidłową wartość, pamięć jest gotowa do użycia.

4.7.2 Dodanie iButtona

Funkcja `add_iButton(uint8_t serial_number[])` umożliwia dodanie nowego iButtona do bazy danych. Numer seryjny musi mieć 8 bajtów. Funkcja najpierw odczytuje aktualną liczbę zarejestrowanych iButtonów. Następnie zapisuje nowy iButton do bazy danych oraz aktualizuje liczbę zarejestrowanych pastylek, o ile limit (32 urządzenia) nie został przekroczony.

4.7.3 Sprawdzenie rejestracji

Funkcja `is_registered(uint8_t serial_number[])` sprawdza, czy dany numer seryjny znajduje się w bazie zarejestrowanych iButtonów.

4.7.4 Dodanie wpisu do historii

Funkcja `add_history(uint8_t serial_number[], uint8_t date[])` dodaje wpis do historii dostępu. Każdy wpis zawiera numer seryjny iButtona (8 bajtów) oraz datę dostępu (6 bajtów: rok, miesiąc, dzień, godzina, minuta, sekunda). Jeśli historia jest pełna (256 wpisów), najstarszy wpis zostaje nadpisany.

4.7.5 Wyświetlenie historii

Funkcja `print_history()` wypisuje wszystkie wpisy historii dostępu przez UART. Każdy wpis jest wyświetlane w formacie:

ID: <numer seryjny> Data: <rok>/<miesiąc>/<dzień> <godzina>:<minuta>:<sekunda>

4.7.6 Reset pamięci

Funkcja `reset_memory()` resetuje stan pamięci, zapisując nową wartość znacznika, kontrolującego czy pamięć należy przygotować.

4.7.7 Odczyt i zapis do pamięci

- `read_from_flash()`: Odczytuje dane z określonego sektora pamięci Flash.
- `write_to_flash_sector()`: Zapisuje dane do sektora Flash. Funkcja wymaga, aby dane miały rozmiar będący wielokrotnością 256 bajtów, a maksymalny rozmiar wynosił 4096 bajtów (ponieważ korzystamy z sektorów o rozmiarze 4 kB).

4.7.8 Funkcje pomocnicze

- `prepare_sector(uint32_t sector_number)`: Przygotowuje sektor Flash do operacji zapisu.
- `erase_sector(uint32_t sector_number)`: Czyści wskazany sektor Flash.
- `get_flash_sector_address(uint32_t sector_number)`: Zwraca adres pamięci dla podanego numeru sektora.
- `send_error_uart(unsigned int n)`: Wysyła komunikat błędu IAP przez UART.

4.7.9 Dodatkowe informacje o pamięci Flash

W projekcie korzystamy z trzech sektorów pamięci Flash o rozmiarach 4 kB każdy, które są przeznaczone do przechowywania różnych danych:

- **Sektor 11 (Rejestracja iButtonów):** Przechowuje identyfikatory (ID) zarejestrowanych iButtonów. Każdy ID zajmuje 8 bajtów.
- **Sektor 12 (Historia dostępu):** Przechowuje historię otwierania sejfu. Każdy wpis w historii zajmuje 14 bajtów (8 bajtów numeru seryjnego i 6 bajtów daty).
- **Sektor 10 (Dane pomocnicze):** W sektorze tym przechowywane są następujące informacje:
 - **Znacznik** (bajty 0–7): Służy do weryfikacji, czy trzeba inicjalizować pamięci.
 - **Liczba zarejestrowanych iButtonów** (bajt 8): Przechowuje informację o liczbie zarejestrowanych iButtonów.
 - **Liczba wpisów w historii** (bajty 16–17): Przechowuje liczbę zapisanych wpisów w historii.

Przerwania podczas operacji na pamięci Flash

Podczas wykonywania operacji na pamięci Flash, takich jak zapis, odczyt czy czyszczenie, konieczne jest tymczasowe wyłączenie przerwań (*interrupts*). Dzieje się tak dlatego, że w trakcie korzystania z interfejsu IAP (In-Application Programming) przerwania mogą zakłócić proces i spowodować jego niepowodzenie. W celu zapewnienia stabilności operacji należy:

- Wyłączyć przerwania przed rozpoczęciem operacji, korzystając z funkcji `__disable_irq()`.
- Wykonać wszystkie niezbędne operacje na pamięci Flash.
- Ponownie włączyć przerwania po zakończeniu operacji za pomocą funkcji `__enable_irq()`.

Zarządzanie pamięcią RAM podczas operacji na Flash

Ze względu na specyfikę pamięci Flash każda operacja zapisu wymaga wcześniejszego wyczyszczenia całego sektora. Aby zapobiec utracie danych podczas tej operacji, należy:

- Odczytać zawartość całego sektora do pamięci RAM przed jego wyczyszczeniem.
- Po modyfikacji danych zapisać zaktualizowaną zawartość z powrotem do sektora.

W związku z tym ważne jest, aby rozmiar stosu (*stack*) lub sterty (*heap*) w systemie został zwiększony do wartości odpowiadającej co najmniej wielkości sektora, czyli 4 kB. Pozwala to na przechowanie tymczasowej kopii danych w pamięci RAM.