Ćwiczenie 17: Przechowywanie danych użytkownika w pamięci EEPROM

Instrukcja laboratorium

Mariusz Chilmon <mariusz.chilmon@ctm.gdynia.pl>

2024-05-27

> Programming is learned by writing programs.
>
> — *Brian Kernighan*

Cel ćwiczenia

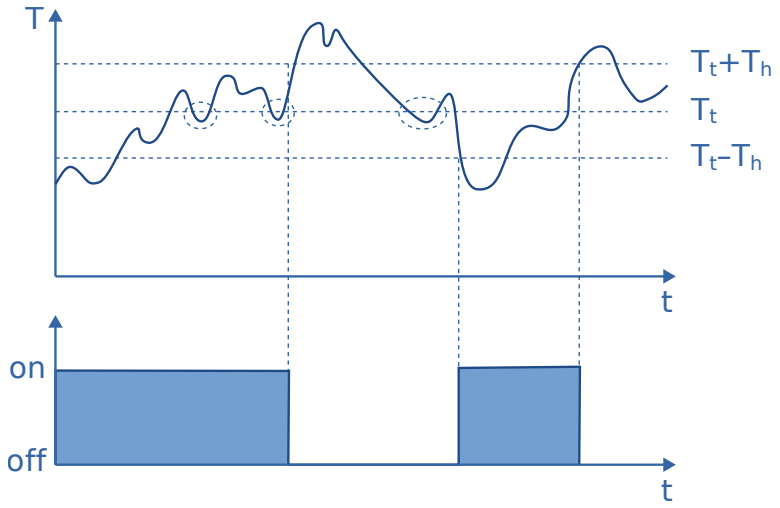
Celem ćwiczenia jest zapoznanie się z:

- obsługą pamięci EEPROM za pomocą biblioteki `avr-libc`,
- obsługą pamięci EEPROM za pomocą bezpośrednich operacji na rejestrach.

Uruchomienie programu wyjściowego

1. Podłącz płytkę *LCD Keypad Shield* do *Arduino Uno*.
2. Podłącz termometr LM35DZ do linii *A5*.
3. W pierwszej linii wyświetlacza widoczne są dwie nastawy: *T* — temperatura zadana T_t i *H* — histereza T_h.
4. Nastawy można zmieniać za pomocą przycisków *UP* i *DOWN*, wybierając je wcześniej przyciskiem *SELECT*.
5. W drugiej linii wyświetlacza widoczna jest bieżąca temperatura i stan grzałki.

Program wyjściowy symuluje termostat, który włącza grzałkę po obniżeniu temperatury poniżej $T_t - T_h$ i wyłącza po prekroczeniu $T_t + T_h$.



Rysunek 1: Histereza w termostacie sterującym grzałką

Zadaniem histerezy jest zmniejszenie częstotliwości przełączania elementu wykonawczego, kosztem zmniejszenia precyzji regulowanego parametru.

W naszym przypadku, gdy mierzona temperatura oscyluje wokół temperatury zadanej T_t np. z powodu ruchu powietrza w pomieszczeniu albo szumu pomiarowego, może dojść do sytuacji, gdy element wykonywaczy byłby przełączany w bardzo krótkich odcinkach czasu. Jest to zjawisko, które może być szkodliwe dla elementu sterującego (np. w przekaźniku może dojść do wypalenia styków), jak i dla elementu wykonawczego (np. kompresor w chłodziarce może ulec szybkiemu zużyciu).

Histereza zazwyczaj jest jednym z parametrów, które są dostępne dla użytkownika jako nastawa, co pozwala mu ustalić kompromis między precyzją sterowania a częstotliwością przełączania.

Rysunek 2: Przykładowy stan wyświetlacza

Zadanie podstawowe

Celem zadania podstawowego jest zapisywanie nastaw urządzenia w pamięci nieulotnej EEPROM.

Wymagania funkcjonalne

1. Po wyjściu z edycji nastaw ustawione wartości zapisywane są w pamięci EEPROM mikrokontrolera.
2. Po zresetowaniu mikrokontrolera wczytywana jest zapamiętana temperatura.
3. Po wyczyszczeniu pamięci przyciskiem *RIGHT* urządzenie wykrywa niezainicjalizowaną pamięć i używa domyślnych nastaw.

Modyfikacja programu

Zapis i odczyt nastaw

Uzupełnij metodę `Thermostat::save()` tak, by zapisywała obie nastawy w pamięci EEPROM pod adresami `EEPROM_ADDRESS_TARGET` i `EEPROM_ADDRESS_HYSTERESIS`.

Uzupełnij metodę `Thermostat::restore()` tak, by odczytywała obie nastawy w pamięci EEPROM pod adresami `EEPROM_ADDRESS_TARGET` i `EEPROM_ADDRESS_HYSTERESIS`. Odczyt odbywa się automatycznie przy inicjalizacji urządzenia.

W pliku nagłówkowym `avr/eeprom.h` zadeklarowane są funkcje obsługujące pamięć EEPROM. Należy zauważyć, że zamiast funkcji z grupy `eeprom_write_XXX()` warto użyć funkcji `eeprom_update_XXX()`, które dokonują zapisu tylko, gdy nowa wartość różni się od poprzedniej.

Dokumentację wspomnianych funkcji można znaleźć na stronie projektu *avr-libc*.

Rozpoznanie niezainicjalizowanej pamięci

Po wciśnięciu przycisku *RIGHT* uruchamiana jest procedura czyszczenia pamięci EEPROM, tj. wypełniania jej wartościami `0xFF`, które prowadzą do interpretacji wartości zadanej temperatury i histerezy jako *nie-liczb*.

Rysunek 3: Odczyt niezainicjalizowanej pamięci

Wyczyszczona pamięć EEPROM nie jest wypełniona bajtami `0x00`, ale `0xFF`. Jest to częsta cecha pamięci nieulotnych.

Liczba zmiennoprzecinkowa, w której wszystkie bity wykładnika są ustawione (co ma miejsce w wyczyszczonej pamięci EEPROM), nie jest poprawną wartością, ale *nie-liczbą* (ang. *NaN — Not a Number*).

Wartość `NaN` można wykryć za pomocą makra `isnan()` zdefiniowanego w bibliotece `math.h`. Po wykryciu takiej wartości w EEPROM należy użyć wartości domyślnych dla nastaw: `TARGET_DEFAULT` i `HYSTERESIS_DEFAULT`.

Zadanie rozszerzone

Celem zadania rozszerzonego jest samodzielna implementacja funkcji odczytującej i zapisującej wartość w pamięci EEPROM.

Wymagania funkcjonalne

1. Funkcjonowanie urządenia nie zmienia się.

Modyfikacja programu

Implementacja odczytu wartości `float` z EEPROM

Napisz własną funkcję odczytującą zmienną typu **`float`** z EEPROM (oczywiście, możesz posłużyć się inną nazwą):

```
1  float eeprom_read_float(const float* address)
```

Przykłady procedur odczytu i zapisu do EEPROM znajdziesz w opisie rejestru EECR (*EEPROM Control Register*) w dokumentacji mikrokontrolera.

Argument funkcji jest typu **`const float`**`*`, gdyż jest to adres w pamięci (wskaźnik). Aby operować na nim jak na liczbie 16-bitowej, w szczególności móc wpisać do pary rejestrów EEAR, można rzutować go na typ `uint16_t` za pomocą operatora rzutowania **`reinterpret_cast`**`<uint16_t>(address)`.

Zwróć uwagę, że należy odczytać liczbę bajtów równą rozmiarowi typu **`float`**. W tym celu możesz zdefiniować tablicę, do której w pętli przepiszesz kolejne bajty z EEPROM:

```
1  uint8_t buffer[sizeof(float)];
```

Bufor można przepisać do zmiennej typu **`float`** za pomocą funkcji `memcpy()` z biblioteki standardowej `string.h`:

```
1  float value;
2  memcpy(&value, buffer, sizeof(float));
```

Implementacja odczytu wartości `float` z EEPROM

Analogicznie zaimplementuj funkcję zapisującą zmienną typu **`float`** do EEPROM:

```
1  void eeprom_write_float(float* address, float value)
```

Zwróć uwagę na zależności czasowe przy ustawianiu bitów `EEMPE` i `EEPE`. Zabezpiecz te operacje blokiem `ATOMIC_BLOCK()`.