Ćwiczenie 14: Przechowywanie danych w pamięci Flash

Instrukcja laboratorium

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

2024-05-04

> No matter which field of work you want to go in, it is of great importance to learn at least one programming language.
>
> — *Ram Ray*

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z:

- wielkością pamięci operacyjnej w mikrokontrolerach,
- dostępem do danych w pamięci programu w architekturze harwardzkiej,
- operacjami na łańcuchach znaków.

Uruchomienie programu wyjściowego

1. Podłącz płytkę *LCD Keypad Shield* do *Arduino Uno*.
2. Program nie buduje się lub działa nieprawidłowo — zależnie od wersji linkera.

Komunikat z linkera, gdy weryfikuje on zajętość pamięci operacyjnej i odmawia wykonania zadania:

```
avr/bin/ld: address 0x800928 of bin/laboratory.elf section `.data' is not within
↪  region `data'
avr/bin/ld: address 0x800929 of bin/laboratory.elf section `.bss' is not within
↪  region `data'
```

Nieprawidłowo działający program, zbudowany przez linker, który nie weryfikuje zajętości pamięci:

Rysunek 1: Przykładowy stan wyświetlacza. Może być też coś zupełnie innego

Kod programu zawiera tablicę `LOCATIONS`, która jest zbiorem struktur typu `Location`, opisujących położenie geograficzne różnych miejscowości.

W realnym urządzeniu taka baza danych może zostać wykorzystana do podawania użytkownikowi położenia na podstawie danych z odbiornika GPS lub sterowania oświetleniem na podstawie czasu z RTC i obliczonej pozycji Słońca nad horyzontem (tzw. sterownik astronomiczny).

Mikrokontrolery AVR zrealizowane są w architekturze harwardzkiej, co oznacza, że pamięć programu i pamięć danych są rozdzielone, a rdzeń procesora zaprojektowany jest do operowania na danych w pamięci RAM. Dane wkompilowane w program z założenia są więc najpierw kopiowane z pamięci programu (Flash) do pamięci danych (SRAM), nawet jeżeli nie będą zmieniane.

Przeciwieństwem architektury harwardzkiej jest architektura von Neumanna, która zakłada przechowywanie instrukcji i danych w tej samej pamięci.

Język C++ jako binarny spadkobierca języka C, zaprojektowanego dla architektury von Neumanna, nie przewiduje pozostawienia danych w innym rodzaju pamięci i odczytywania ich dopiero w trakcie działania programu. Jest to możliwe tylko za pomocą dostarczanych z kompilatorem dedykowanych dyrektyw i funkcji, które muszą być świadomie użyte przez programistę.

Zwróć uwagę, że użycie kwalifikatora `const` przed typem zmiennej nie spełnia pokładanych w nim nadziei. Informuje on tylko kompilator, że nie wolno wprost pozwolić programiście na zmianę tych danych. Niestety, i tak są one kopiowane do pamięci RAM.

Ze względu na bardzo mały rozmiar RAM-u mikrokontrolera, tablica `LOCATIONS` nie mieści się w niej[1], skutkiem czego program nie może działać prawidłowo, a jeżeli linker weryfikuje rozmiar dostępnej pamięci — w ogóle nie jest budowany.

Zadanie podstawowe

Celem zadania podstawowego jest uniknięcie kopiowanie danych do pamięci operacyjnej na starcie programu i odczytywanie ich bezpośrednio z pamięci Flash.

Wymagania funkcjonalne

1. W pierwszej linii wyświetlana jest nazwa miejscowości, a w drugiej — współrzędne geograficzne.
2. Przyciskami *DOWN* i *UP* można przeglądać inne pozycje w bazie lokalizacji.

Rysunek 2: Stan wyświetlacza po poprawieniu programu

[1] W celu zamodelowania tego problemu część nazw jest dopełniona długim ciągiem znaków, co gwarantuje przepełnienie pamięci.

Modyfikacja programu

Zachowanie danych w pamięci programu

W celu pozostawienia danych w pamięci Flash należy w pliku `main.cpp` po nazwie odpowiednich zmiennych dodać dyrektywę `PROGMEM`, np. zapis:

```
1  const char FOO[] = "Longcat is long";
```

skutkuje kopiowaniem danych do pamięci RAM przy uruchomieniu programu[2], a zapis:

```
1  const char FOO[] PROGMEM = "Longcat is long";
```

pozostawi je w pamięci Flash.

W pamięci programu powinny pozostać zarówno wszystkie nazwy miejscowości, jak i sama tablica `LOCATIONS` wraz ze współrzędnymi.

Na tym etapie stan wyświetlacza prawdopodobnie ulegnie zmianie, ale wciąż nie będzie poprawny.

Odczyt współrzędnych geograficznych z pamięci programu

Aby odczytać współrzędne z pamięci Flash, należy najpierw pobrać ich adresy za pomocą operatora &, np.:

```
1  const void* flashAddress = &LOCATIONS[locationIndex].latitude;
```

a następnie odczytać za pomocą funkcji `pgm_read_float()`:

```
1  double latitude = pgm_read_float(flashAddress);
```

Typ `const void*` oznacza wskaźnik (a więc nic innego, jak adres w pamięci) na dane tylko do odczytu.

Rysunek 3: Na tym etapie na wyświetlaczu otrzymujemy już poprawne współrzędne.

[2] I to jeszcze przed uruchomieniem funkcji `main()`!

Odczyt nazwy miejscowości z pamięci programu

Aby odczytać tekst z pamięci Flash należy, podobnie jak dla współrzędnych, odczytać adres danych z bazy danych w pamięci Flash:

```
1  const void* flashAddress = &LOCATIONS[locationIndex].name;
```

Nie jest to jednak jeszcze adres samego tekstu, ale adres tego adresu[3]. Musimy zatem najpierw odczytać z pamięci Flash adres samego tekstu, który to adres jest słowem 16-bitowym:

```
1  uint16_t textAddress = pgm_read_word(flashAddress);
```

Tak odczytany adres możemy przekonwertować na typ `const char*`, aby kompilator rozumiał, że ma do czynienia ze wskaźnikiem na łańcuch znaków:

```
1  const char* name = reinterpret_cast<const char*>(textAddress);
```

Teraz można by odczytać ten tekst bajt po bajcie za pomocą funkcji `pgm_read_byte()`, lub, prościej, za pomocą funkcji `strncpy_P`, która skopiuje fragment tektu do bufora:

```
1  strncpy_P(buf, name, sizeof(buf));
```

Zwróć uwagę że kopiowany jest tylko fragment tekstu, który mieści się na wyświetlaczu, więc zużycie pamięci jest minimalne.

Na wyświetlaczu nad współrzędnymi powinna pojawić się nazwa miejscowości.

Zadanie rozszerzone

Celem zadania rozszerzonego jest umożliwienie przewijania na wyświetlaczu długich nazw miejscowości.

Wymagania funkcjonalne

1. Przyciskami *LEFT* i *RIGHT* można przewijać nazwę miejscowości od początku do końca.
2. Przejście do kolejnej lub poprzedniej lokalizacji za pomocą przycisków *UP* i *DOWN* powinno zawsze wyświetlać początek nazwy miejscowości.

[3] Wynika to ze wspomnianego wcześniej niedostosowania języków C i C++ do architektury harwardzkiej.

Modyfikacja programu

Przesuwanie tekstu

Aby przesunąć tekst należy skorzystać ze zmiennej `textOffset`, która jest obsługiwana za pomocą przycisków *LEFT* i *RIGHT*. Jest to wartość, którą można wprost dodać do adresu odczytywanego tekstu:

```
1 strncpy_P(buf, name + textOffset, sizeof(buf));
```

Ograniczenie przesuwania do długości bieżącego tekstu

Pozostaje do rozwiązania jeden problem — na razie przesuwanie nie uwzględnia długości tekstu i pozwala przesunąć kursor poza koniec tekstu. Niestety, na danych w pamięci programu niemożliwe jest użycie funkcji `strlen()` z biblioteki standardowej. Należy zatem napisać taką funkcję samodzielnie i, odczytując tekst bajt po bajcie, określić jego długość. W tym celu należy uzupełnić funkcję `flashTextLength()` i użyć jej w obsłudze przycisku *RIGHT*.