Uwagi

iii: English abstract
3: Dwie-trzy linijki zagajenia
3: Skąd wiemy, że jest najpopularniejsza? Trzeba zacytować.
3: Ośmiobitowa
3: cytowanie
3: jak wyżej
3: harwardzkiej
3: bez przecinka
4: –
4: bez przecinka
4: cytowania
4: Po prostu: ma.
4: cytowanie
4: ang. *communications device class*
4: Nigdzie w pracy nie ma odnośnika do rys. 2.1.
6: *general-purpose programming languages*
6: cytowanie
6: bez przecinka
6: –
6: cytowanie
6: cytowanie
6: cytowanie
6: cytowanie
6: Ten rozdział jest zdecydowanie za krótki.
7: przecinek
7: bez przecinka
7: zdanie wprowadzenia
7: .
7: zdanie wprowadzenia
8: Wbrew tytułowi w tym podrozdziale nie mamy opisu istniejących rozwiązań.
8: Dość dokładnie wymagania dla języka. Czym on się będzie różnił od już istniejących rozwiązań?
8: –
9: cytowanie
9: cytowanie
9: Tak.
9: Opis narzędzi. Dlatego te narzędzia? Dlaczego one wybrane?

11: A co z wyjściem? Co generuje kompilator?

12: 2-3 linijki zagajenia

17: Zawsze na końcu.

17: Pierwszy akapit: W ramach pracy dyplomowej powstało ... Jakie cechy? Jakie własności?

Politechnika
Śląska

PROJEKT INŻYNIERSKI

Język programowania dla mikrokontrolerów AVR

Maciej WALERYN
Nr albumu: 296441

Kierunek: Informatyka
Specjalność: Grafika komputerowa

PROWADZĄCY PRACĘ
dr hab. inż. Krzysztof Simiński
KATEDRA Algorytmiki i Oprogramowania
Wydział Automatyki, Elektroniki i Informatyki

OPIEKUN, PROMOTOR POMOCNICZY
⟨stopień naukowy imię i nazwisko⟩

Gliwice 2025

Tytuł pracy

Język programowania dla mikrokontrolerów AVR

Streszczenie

Celem dyplomu jest zaprojektowanie języka programowania oraz napisanie kompilatora dla mikrokontrolera w architekturze AVR. Język ten ma spełniać cechę kompletności Turinga, posiadać znane dla współczesnych języków programowania funkcjonalności, systemy zapobiegające powstawaniu błędów w programie (np. *garbage collector*) i podstawowe algorytmy optymalizacji zmniejszające kod wynikowy oraz czas wykonania programów.

Słowa kluczowe

język programowania, kompilator, mikrokontrolery AVR

Thesis title

Programming language for AVR microcontrollers

Abstract

[English abstract] (Thesis abstract – to be copied into an appropriate field during an electronic submission – in English.)

Key words

programming language, compiler, AVR microcontrollers

Spis treści

Rozdział 1

Wstęp

Wytwarzanie oprogramowania dla systemów wbudowanych wymaga wiedzy specjalistycznej na wysokim poziomie. W przypadku tworzenia projektów amatorskich, przodującymi technologiami są płytki rozwojowe oparte o mikrokontrolery AVR serii ATMega. Ograniczenia tej architektury, tj. mała ilość pamięci operacyjnej i programowej, skłaniają programistów do korzystania z niskopoziomowych języków oraz bibliotek. Te czynniki są dużym utrudnieniem dla nowicjuszy, co w dużej mierze prowadzi do zniechęcenia do rozwoju w kierunku systemów wbudowanych, wydłuża czas realizacji projektów i zwiększa liczbę powstających błędów w wytwarzanym oprogramowaniu.

Celem tej pracy jest opracowanie języka programowania, posiadającego zalety języków wysokopoziomowych, ułatwiającego pracę z mikrokontrolerami AVR. Docelowym układem, dla którego generowany oraz testowany będzie kod wynikowy, jest mikrokontroler ATMega328, znany szeroko z występowania w płytkach rozwojowych Arduino Uno. Ze względu na rozmiar rodziny mikrokontrolerów AVR, kompilator będzie umożliwiał wprowadzenie parametrów konfiguracyjnych, pozwalając tym samym na wsparcie dla większości członków tej rodziny mikrokontrolerów.

Opis rozdziałów będzie gotowy po bliższym przygotowaniu listy rozdziałów :)

- wprowadzenie w problem/zagadnienie
- osadzenie problemu w dziedzinie
- cel pracy
- zakres pracy
- zwięzła charakterystyka rozdziałów
- jednoznaczne określenie wkładu autora, w przypadku prac wieloosobowych – tabela z autorstwem poszczególnych elementów pracy

Rozdział 2

Języki programowania dla mikrokontrolerów AVR

[Dwie-trzy linijki zagajenia]

2.1 Mikrokontrolery AVR

Platforma mikrokontrolerów AVR, utworzona przez firmy Atmel i aktualnie będąca własnością Microchip, jest jedną z najbardziej popularnych platform wykorzystywanych w budowaniu urządzeń wbudowanych [Skąd wiemy, że jest najpopularniejsza? Trzeba zacytować.]. Ośmio-bitowa [Ośmiobitowa] architektura tych mikrokontrolerów powstała w roku 1996, na rynek wprowadzona została w roku 1997. Jej najważniejszymi cechami stały się prostota wytwarzania dla niej oprogramowania, niski pobór mocy oraz przystępna cena układów [cytowanie].

Mikrokontrolery AVR zostały oparte na ośmio-bitowym [jak wyżej] procesorze RISC w zmodyfikowanej architekturze Harwardzkiej [harwardzkiej] z autorskim zestawem instrukcji. W zależności od rodziny, [bez przecinka] rdzeń procesora może pracować z zakresie częstotliwości 1-20 MHz lub 32 MHz dla rodziny XMEGA. W podstawowej wersji architektury, dyspozycji programisty zostały przekazane:

- pamięć Flash, wykorzystywana jako pamięć programu,
- pamieć SRAM, służąca przechowywaniu zmiennych,
- pamięć EEPROM, umożliwiająca przechowywanie dużych wartości statycznych,
- zbiór rejestrów wewnętrznych kontrolujących pracę mikrokontrolera oraz służących do wykonywania instrukcji,
- rejestry portów wejścia/wyjścia,

- w zależności od modelu: liczniki zegarowe, konwertery analogowo-cyfrowe i cyfrowo-analogowe, sprzętowe interfejsy dedykowane dla protokołów tj. TWI, UART, SPI, USB, Ethernet.

Zależnie od wymagań projektowanego systemu systemu wbudowanego, dostępne jest wiele modeli mikrokontrolerów. Ze względu na ich mnogość można wyróżnić trzy najpopularniejsze podrodziny:

- tinyAVR - [–] niska cena, mała ilość pamięci oraz wyprowadzeń (od 8 do 32),
- megaAVR - szeroka gama rozszerzeń i funkcji, większa ilość pamięci i wyprowadzeń (od 28 do 100) względem tinyAVR,
- XMEGA - wyższe taktowanie procesora.

2.2 Popularyzacja platformy AVR za pośrednictwem Arduino

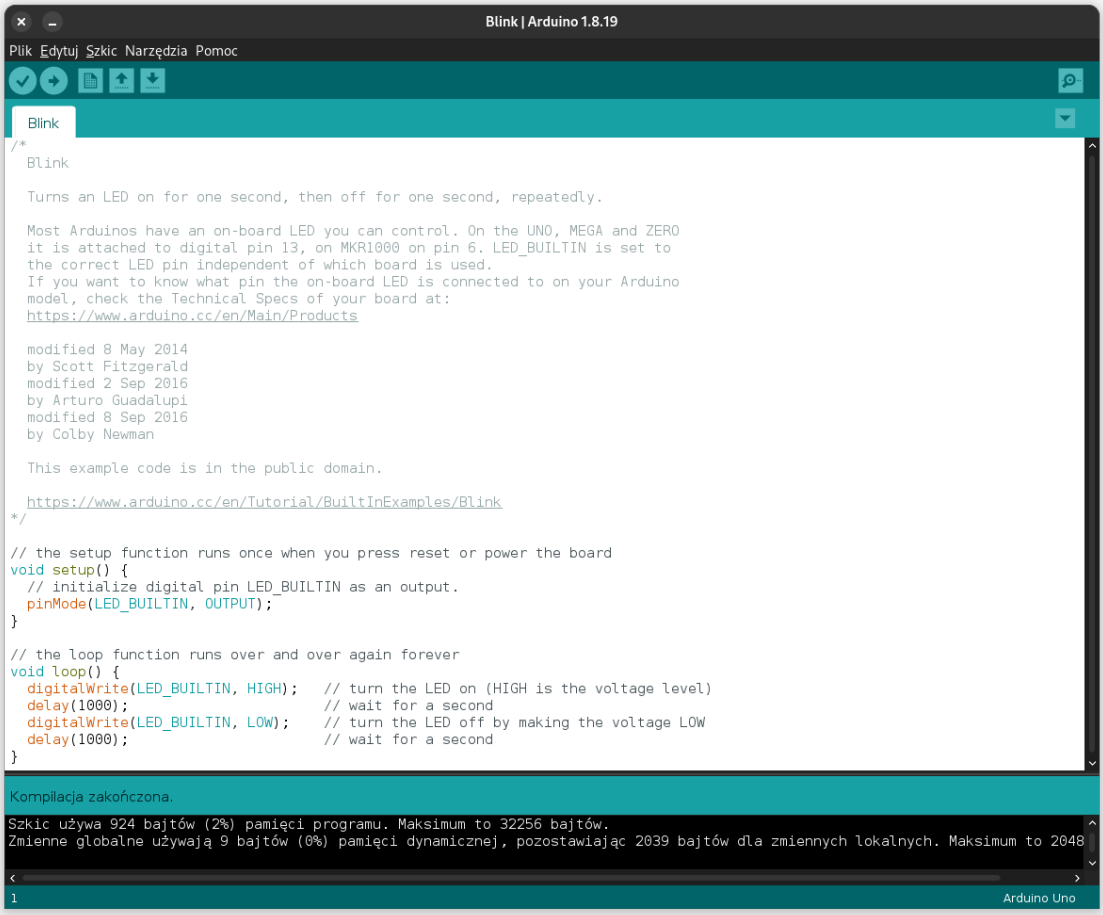
Prawdziwym przełomowym momentem dla platformy AVR stał się rok 2010, kiedy w trakcie wydarzenia Maker Faire w Nowym Yorku, [bez przecinka] zaprezentowano nową płytkę rozwojową firmy Arduino, nazwaną Arduino UNO. Pomimo istnienia poprzedników, płytka ta stała się najpopularniejszą platformą dla początkujących elektroników. [cytowania]

UNO posiada [Po prostu: ma.] dwa mikrokontrolery AVR [cytowanie]:

- ATmega328P będący głównym mikrokontrolerem podłączonym do portów wejścia-/wyjścia na płytce rozwojowej,
- ATmega16U2 wykorzystywany jako programator USB oraz pozwalający na komunikację z głównym mikrokontrolerm poprzez port szeregowy. Po modyfikacji jego programu, możliwe jest także wykorzystanie innych trybów protokołu USB np. HID.

Użycie dwóch układów nie tylko zwiększyło możliwości płytki prototypowej. Miało także wpływ na łatwość korzystania z platformy poprzez brak wymogu instalacji sterowników dzięki protokołowi USB CDC (communications device class [ang. *communications device class*]).

Dla płytki Arduino powstało także środowisko programistyczne i zestaw bibliotek skierowany do początkujących programistów. Wyróżniają się one prostotą i przejrzystością. Edytor pozwala na kompilację kodu i programowanie płytek rozwojowych bez skomplikowanej konfiguracji. Biblioteki dla języka C, dostarczone wraz ze środowiskiem, ukierunkowane są na prostotę i czytelność kodu, ukrywając przed programistą niskopoziomowe cechy programowania systemów wbudowanych. [Nigdzie w pracy nie ma odnośnika do rys. 2.1.]



Rysunek 2.1: Edytor Arduino IDE z przykładowym programem Blink

2.3 Języki programowania

Oprogramowanie wytwarzane dla systemów wbudowanych wymaga niskopoziomowego dostępu do sprzętu. Ze względu na niską ilość zasobów i jednoczesną złożoność programów pracujących na mikrokontrolerach, wykorzystuje się języki ogólnego przeznaczenia (ang. general-purpose programming languages [*general-purpose programming languages*]), które pozwalają na bezpośrednią manipulację pamięcią. Oficjalnie wspieranymi językami przez środowisko programistyczne AVR są C, C++ i Assembly, dostarczanymi za pośrednictwem GNU Compiler Collection [cytowanie].

Wraz ze wzrostem popularności mikrokontrolerów AVR, [bez przecinka] powstały także alternatywne kompilatory i interpretery dla popularnych języków programowania. Implementacje niektórych z języków można znaleźć pod nazwami:

- AVR-Rust - [–] zmodyfikowana wersja kompilatora Rust dla platformy AVR [cytowanie],
- AVR-Ada (AVR-GNAT) - kompilator języka Ada dla AVR [cytowanie],
- PyMite - minimalistyczny interpreter jezyka Python 2.5 [cytowanie],
- NanoVM - wirtualna maszyna dla kodu bajtowego języka Java [cytowanie].

[Ten rozdział jest zdecydowanie za krótki.]

Rozdział 3

Wymagania i narzędzia

3.1 Wymagania projektowe

Język programowania, będący tematem pracy [przecinek] musi spełniać szereg założeń, pozwalających na wytwarzanie oprogramowania dla systemów wbudowanych z ograniczoną ilością zasobów pamięciowych i obliczeniowych. Ponadto, [bez przecinka] język ma być zrozumiały dla początkujących programistów poprzez zastosowanie odpowiedniej semantyki i słów kluczowych.

3.1.1 Wymagania funkcjonalne

[zdanie wprowadzenia]

- kompilowalność do kodu maszynowego, specyficznego dla danej platformy,
- dostęp do sprzętowych zasobów niskiego poziomu np.: adresy bezpośrednie pamięci, wskaźniki, zapis w rejestrach,
- ścisłe typowanie poprzez składnię języka,
- prosta i czytelna semantyka, zrozumiała dla osób początkujących,
- mechanizmy abstrakcyjne ułatwiające organizacje kodu,
- możliwość łączenia wielu plików kodu źródłowego z widoczną separacją pochodzenia zmiennych i funkcji [.]

3.1.2 Wymagania niefunkcjonalne

[zdanie wprowadzenia]

- mechanizmy optymalizacji kodu zmniejszające złożoność pamięciową i obliczeniową programów,

- statyczne kontrola typów danych na poziomie kompilacji kodu,
- ścisła kontrola spójności składni kodu,
- kompletność w sensie Turinga.

3.1.3 Istniejące rozwiązania

[Wbrew tytułowi w tym podrozdziale nie mamy opisu istniejących rozwiązań.] Oficjalnie dostępne narzędzia dla mikrokontrolerów AVR, zapewniane przez firmę Microchip, pozwalają na programowanie w językach C, C++ i Assembly. Język uCricket ma na celu zminimalizowanie popełnianych przez początkujących programistów błędów pojawiających się w trakcie wytwarzania oprogramowania poprzez zapewnienie stosownych mechanizmów prewencji. Jednym z takich mechanizmów jest dużo bardziej ścisła kontrola typów danych. Języki C/C++, w swoim mechanizmie typowania, traktują wartości numeryczne i logiczne jako jeden typ, który można stosować zamiennie, co może prowadzić do niejasnego zachowania programu. Czytelność języka, w porównaniu do oficjalnych rozwiązań, starano się także rozwiązać poprzez dodanie odpowiednich słów kluczowych, definiujących elementy złożone, w formie przedrostków:

- scope `scope` - definicja odseparowanej przestrzeni nazw,
- func - definicja funkcji,
- var - definicja zmiennej,
- ptr - definicja wskaźnika.

Kod języka uCricket wymaga stosowania przestrzeni nazw, z myślą redukcji powtórzeń i konfliktów nazw, w szczególności przy dołączaniu kolejnych plików źródłowych.

[Dość dokładnie wymagania dla języka. Czym on się będzie różnił od już istniejących rozwiązań?]

3.2 Narzędzia

Do zaprojektowania i implementacji języka programowania uCricket wykorzystano następujące narzędzia:

- Java - [–] język programowania cechujący się obiektowością, czytelną składnią i zaawansowaną kontrolę typów, ułatwia on pracę z wieloma typami dziedziczącymi między sobą;

- JFlex - narzędzie generujące kod analizatora leksykalnego w języku Java, pozwala na korzystanie z wyrażeń regularnych, specyfikacja leksykalna definiowana jest w oddzielnym pliku [cytowanie];
- GNU Bison - zaawansowane narzędzie generujące kod analizatora składniowego na podstawie specyfikacji gramatycznej. Powszechnie wykorzystywany jako narzędzie pomocnicze w implmementacji kompilatorów. Posiada wsparcie dla języka Java;
- LLVM - zestaw bibliotek i narzędzi pozwalających na generowanie kodu pośredniego poprzez udostępnione interfejsy programistyczne. Wspiera wiele platform, m.in: x86, ARM, PowerPC, AVR. Dostarcza także mechanizmy optymalizacji kodu wynikowego [cytowanie];
- JavaCPP Presets LLVM - biblioteka-most udostępniająca interfejsy dla biblioteki LLVM języka C poprzez wykorzystanie JNI (Java Native Interface);
- avr-gcc - kompilator języka C dla mikrokontrolerów AVR. Stosowany jest jako generator kodu maszynowego dla kodu generowanego przez LLVM;
- avrdude - program pozwalający na programowanie mikrokontrolerów AVR;
- Cutter - platforma do inżynierii wstecznej pomagająca w analizie wygenerowanego kodu maszynowego. Pracuje w oparciu o Rizin Framework;
- AVR Simulator - symulator mikrokontrolera ATmega328P działający w przeglądarce;
- *Arduino Pro Mini - płytka rozwojowa z mikrokontrolerem ATmega328P;*
- *USBasp - programator mikrokontrolerów AVR.*

Czy brać pod uwagę także płytkę rozwojową i programator? [Tak.]

Zestaw tych narzędzi i bibliotek pozwala na projektowanie skompikowanych kompilatorów generujących niskopoziomowy, a także testowanie i weryfikacji poprawności programów napisanych w tworzonym języku programowania.

[Opis narzędzi. Dlatego te narzędzia? Dlaczego one wybrane?]

Rozdział 4

Specyfikacja zewnętrzna

4.1 Kompilator

4.1.1 Wymagania

Do uruchomienia kompilatora wymagane jest zapewnienie odpowiedniego środowiska uruchomieniowego składającego się z:

- systemu operacyjnego z rodziny GNU/Linux lub środowiska uruchomieniowego zapewniającego warunki pracy dla programów kompilowanych dla systemów GNU/Linux np. maszyna wirtualna, WSL, kontener Docker/Podman/Distrobox;
- maszyna wirtualna Java w wersji 17 lub wyższej;
- zestaw narzędzi avr-gcc wraz z linkerem i assemblerem.

4.1.2 Instalacja

Kompilator dostarczany jest jako archiwum programu Java w formacie .jar. Oprogramowanie LLVM zostało dołączone w archiwum jako biblioteka w formie binarnej dla platformy GNU/Linux.

4.1.3 Użycie kompilatora

Uruchomienie kompilatora odbywa się poprzez przekazanie maszynie wirtualnej programu kompilatora oraz dodaniu nazwy głównego pliku źródłowego bez rozszerzenia.

```
1 java -jar uCricket.jar [glowny_plik_zrodlowy]
```

[A co z wyjściem? Co generuje kompilator?]

4.2 Język programowania

[2-3 linijki zagajenia]

4.2.1 Konwencja nazewniczna identyfikatorów

Nazywanie elementów definiowanych przez użytkownika wymaga ścisłego używania nazw zgodnych z konwencją zapisu identyfikatorów. Identyfikatory przyjmują znaki alfanumeryczne, włącznie z dużymi i małymi literami, nie mogą zaczynać się cyfrą. [Da się je opisać wyrażeniem regularnym?]

4.2.2 Pliki źródłowe

Nazwy plików źródłowych muszą spełniać wymagania zapisu identyfikatorów. Nazwy te są wykorzystywane do identyfikacji plików źródłowych podczas dołączania źródeł między sobą i są niezależne od nazw przestrzeni (scopes). Rozszerzeniem dla plików źródłowych jest *.uchirp*.

- wymagania sprzętowe i programowe
- sposób instalacji
- sposób aktywacji
- sposób obsługi
- administracja systemem
- kwestie bezpieczeństwa
- przykład działania

Rozdział 5

Specyfikacja wewnętrzna

Jeśli „Specyfikacja wewnętrzna”:

- przedstawienie idei
- architektura systemu
- opis struktur danych (i organizacji baz danych)
- komponenty, moduły, biblioteki, przegląd ważniejszych klas (jeśli występują)
- przegląd ważniejszych algorytmów (jeśli występują)
- szczegóły implementacji wybranych fragmentów, zastosowane wzorce projektowe
- diagramy UML

Rozdział 6

Weryfikacja i walidacja

- sposób testowania w ramach pracy (np. odniesienie do modelu V)
- organizacja eksperymentów
- przypadki testowe zakres testowania (pełny/niepełny)
- wykryte i usunięte błędy
- opcjonalnie wyniki badań eksperymentalnych

Rozdział 7

Podsumowanie i wnioski

[Zawsze na końcu.] [Pierwszy akapit: W ramach pracy dyplomowej powstało ... Jakie cechy? Jakie własności?]

- uzyskane wyniki w świetle postawionych celów i zdefiniowanych wyżej wymagań
- kierunki ewentualnych danych prac (rozbudowa funkcjonalna ...)
- problemy napotkane w trakcie pracy

Dodatki

Spis skrótów i symboli

DNA kwas deoksyrybonukleinowy (ang. *deoxyribonucleic acid*)

MVC model – widok – kontroler (ang. *model–view–controller*)

N liczebność zbioru danych

μ stopnień przyleżności do zbioru

$\mathbb{E}$ zbiór krawędzi grafu

$\mathcal{L}$ transformata Laplace'a

Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego, należy je przenieść w to miejsce.

```
1 if (_nClusters < 1)
2     throw std::string ("unknown number of clusters");
3 if (_nIterations < 1 and _epsilon < 0)
4     throw std::string ("You should set a maximal number of
          iteration or minimal difference — epsilon.");
5 if (_nIterations > 0 and _epsilon > 0)
6     throw std::string ("Both number of iterations and minimal
          epsilon set — you should set either number of iterations
          or minimal epsilon.");
```

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- dane testowe,
- film pokazujący działanie opracowanego oprogramowania lub zaprojektowanego i wykonanego urządzenia,
- itp.

Spis rysunków

Spis tabel