МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра компьютерных технологий

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Стринадка Кирилла Александровича

Тема работы:

РАЗРАБОТКА IDE ЯЗЫКОВ REFLEX/INDUSTRIALC ДЛЯ ЦЕЛЕЙ ПРОГРАММИРОВАНИЯ ВСТРАИВАЕМЫХ СИСТЕМ НА БАЗЕ МИКРОКОНТРОЛЛЕРОВ ATMEGA

«К защите допущена»
Заведующий кафедрой,
д.т.н., доцент
Зюбин В.Е. /……..
(ФИО) / (подпись)
«…»……20…г.

Руководитель ВКР
д.т.н., доцент,
зав. каф. КТ ФИТ НГУ
Зюбин В.Е ./……...
(ФИО) / (подпись)
«…»……20…г.

Соруководитель ВКР
к.ф-м.н., доцент каф. СИ
ФИТ НГУ
Ануреев И. С. /……...
(ФИО) / (подпись)
«…»……20…г.

Новосибирск, 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий
Кафедра компьютерных технологий
(название кафедры)

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Зюбин В. Е.
(фамилия, И., О.)

…….
(подпись)
«……»……20…г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Стринадка Кириллу Александровичу, группы 20207
(фамилия, имя, отчество, номер группы)

Тема «Разработка IDE языков Reflex/IndustrialC для целей программирования встраиваемых систем на базе микроконтроллеров ATmega»
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе от……№……

Срок сдачи студентом готовой работы……2024 г.

Исходные данные (или цель работы): Совершенствование инструментария программирования для микроконтроллеров ATmega через объединение процесс-ориентированных языков IndustrialC и Reflex и развитие их грамматики с использованием технологии Eclipse Xtext.

Структурные части работы: Анализ предметной области, обзор существующих языков, формулирование требований к коррекции языка, коррекция грамматики языка, реализация интегрированной среды разработки.

Консультанты по разделам ВКР (при необходимости, с указанием разделов):

_____
(раздел, ФИО)

Руководитель ВКР
д.т.н., доцент, зав. каф. компьютерных технологий ФИТ
Зюбин В.Е. /……...
(ФИО) / (подпись)
«…»……20…г.

Задание принял к исполнению
Стринадка К. А. /……...
(ФИО студента) / (подпись)
«…»……20…г.

Соруководитель ВКР
к.ф-м.н., доцент каф. СИ ФИТ НГУ
Ануреев И. С. /……...
(ФИО) / (подпись)
«…»……20…г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время активно развиваются и получают свое повсеместное распространение встраиваемые системы в рамках киберфизических систем и в рамках концепции индустрии 4.0, что приводит не только к автоматизации многих процессов, но и к значительному росту сложности алгоритмов для этих встраиваемых систем.

Вычислительная основа таких встраиваемых систем представлена микроконтроллерами, среди которых самым распространенным является серия микроконтроллеров ATmega.

На данный момент встраиваемые системы в основном программируются на низкоуровневых языках, таких как Assembly, C и C++, приводит к созданию громоздких и ненадежных программ, которые могут содержать логические ошибки. Особенно это касается систем с большим количеством микроконтроллеров, где различные узлы могут программироваться на разных языках, что значительно увеличивает трудоемкость данного процесса. Это все приводит к тому, что значительная часть бюджета в проектах распределенного управления уходит на программную часть.

Таким образом, становится актуальной проблема исследования новых подходов к разработке ПО для встраиваемых систем.

В ИАиЭ СО РАН идет работа в этом направлении. Для решения вышеописанных проблем разрабатываются процесс-ориентированные диалекты Си:IndustrialC [1] и Reflex [2]. Данные языки имеют свои недостатки, в частности, в ограниченных возможностях работы с периферией и прерываниями для Reflex и в высоком пороге входа для новых разработчиков в IndustrialC. В данном контексте возникает актуальная задача разработки инструментария, способного полноценно использовать потенциал микроконтроллеров ATmega.

По результатам практической апробации данных языков появилось понимание, что их нужно объединять на технологии Eclipse и развивать, добавлением в синтаксис недостающих конструкций.

В свете вышеизложенного, целью данной выпускной квалификационной работы является разработка и совершенствование инструментария программирования для микроконтроллеров ATmega. Анализируя существующие процесс-ориентированные языки программирования Industrial-C и Reflex, было решено проводить их "слияние". Мотивация состоит в том, что IndustrialC разработан специально для микроконтроллеров ATmega на основе технологии bison/Antlr, а Reflex разработан с помощью более продвинутой технологии Eclipse Xtext, но в нем отсутствует возможность работы с периферией. Таким образом необходимо объединить две данные технологии с помощью Eclipse Xtext, создав унифицированный язык программирования.

Также существуют практические проекты по смежным темам, по результатам которых были сформированы предложения по модификации синтаксиса. Одним из таких предложенных примеров являются "легковесные состояния".

В рамках работы предстоит анализировать и совершенствовать грамматику языка Reflex, добавляя необходимые конструкции для корректной работы с микроконтроллерами ATmega. Основным результатом проекта станет создание интегрированной среды разработки, ориентированной на микроконтроллеры ATmega [3], с акцентом на сокращение трудоемкости разработки программ для данной платформы.

Цель Создание IDE для языка Reflex с развитием его грамматики для целей программирования встраиваемых систем на базе микроконтроллеров ATmega.

Задачи работы:

- Проанализировать существующие версии процесс-ориентированных языков Reflex и IndustrialC и специфику работы с микроконтроллерами ATmega.
- Сформировать список требований для коррекции объединенного языка.
- Скорректировать грамматику, определить семантику, правила валидации и правила генерации выходных файлов для объединенного языка.
- Реализовать интегрированную среду разработки, включающую синтаксически ориентированный редактор, парсер и кодогенератор для объединенного языка.

Научная новизна :

- Проведено объединение языков Reflex и IndustrialC с последующем развитием грамматики.
- Добавлены недостающие синтаксические конструкции (циклы, массивы), введены функции, процедуры обработка прерываний (ISR), atomic блоки кода, volatile переменные.
- Реализована кодогенерация для новых синтаксических конструкций.
- Создана интегрированная среда разработки ориентированная на микропроцессоры ATmega.

Практическая ценность :

- Сокращение трудоемкости разработки программ для микроконтроллера ATmega.
- Сокращение стоимости владения программным обеспечением для разработки в данной области.

Структура работы :

Работа состоит из введения, 3 глав и заключения.

В первой главе на основе литературных данных описывается предметная область и приводится анализ аналогов среди языков программирования, ориентированных на разработку ПО для встраиваемых систем, выявляются их преимущества и недостатки, а также предъявляются требования к разрабатываемому языку. Во второй главе описывается синтаксис разрабатываемого языка Reflex с учетом введенных корректировок в его грамматику, а также предлагаемый способ обеспечения поддержки микроконтроллеров ATmega. В третьей главе обосновывается выбор стека технологий, описывается практическая реализация ядра web среды разработки Reflex.

Глава 1: Анализ предметной области и формулировка требований к разрабатываемым средствам

1.1 Предметная область

Современные разработки встраиваемых систем [4], которые базируются на микроконтроллерах ATmega [5], становятся все более популярными благодаря их широкому спектру применения – от бытовой электроники до промышленного оборудования [6]. Программирование таких систем требует специализированных инструментов, которые учитывают специфику аппаратных ресурсов этих микроконтроллеров. В этой работе предлагается обзор данной области, в том числе анализ существующих языков программирования, которые чаще всего используются для разработки встраиваемых систем, с упором на Reflex и IndustrialC. Также рассматриваются основные проблемы, возникающие у разработчиков и возможные способы их решения.

1.1.1 Кибер-физические системы и ПЛК

Кибер-физические системы (КФС) — это сложные системы, которые объединяют вычисления с физическими процессами [7]. Они используют встраиваемые системы и сети для мониторинга и управления физическими процессами.

В таких системах обычно используется влияние обратной связи. Обратная связь – явление, при котором физические процессы явно или косвенно влияют на вычисления на микроконтроллерах, а вычисления, в свою очередь, тоже влияют на ход физических процессов. Данное взаимодействие между кибернетикой и физическими компонентами является ключевой особенностью КФС.

Основные компоненты КФС включают в себя объекты управления, датчики, которые реагируют на внешние события, и центральный управляющий объект. Программируемые логические контроллеры (ПЛК) выступают в качестве центрального элемента в таких системах. Они обеспечивают управление физическими процессами, обрабатывают входящие сигналы и принимают решения в зависимости от обратной связи.

Такие решения применяется в медицинской области, для управления потоком транспорта, в авиационной индустрии.

Таким образом, КФС представляют собой интеграцию физических процессов и кибернетики, что открывает широкие возможности для инноваций в различных областях, включая медицину, транспорт и промышленность. В этом контексте ПЛК играют ключевую роль в обеспечении надежного управления и точного взаимодействия между кибернетическими и физическими компонентами.

Рисунок 1 – Компоненты системы и их взаимодействие

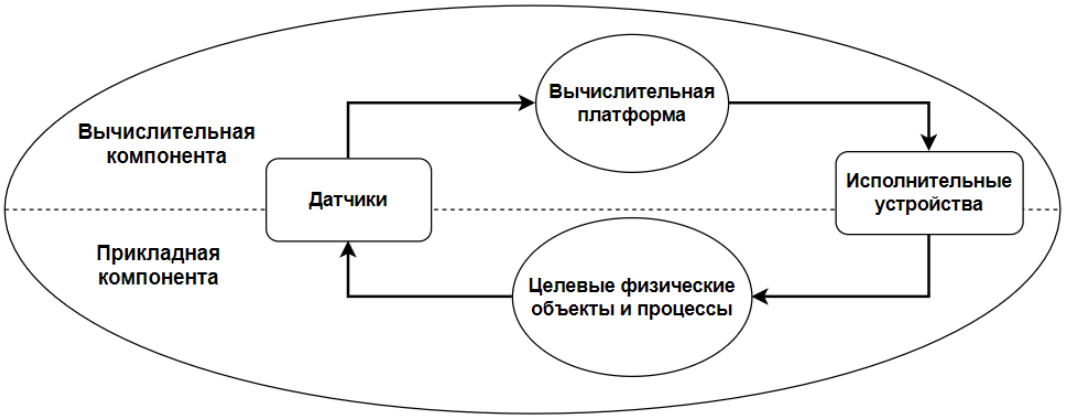

Рисунок 1 – Схема простейшей КФС

Данная схема демонстрирует простейшую КФС, разделенную на вычислительную и прикладную компоненты. Датчики отслеживают изменения и отправляют информацию на прикладной уровень для обработки. В ответ на полученные данные, прикладной уровень отправляет команды на микроконтроллеры, которые воздействуют на физические объекты и процессы. Обратная связь через датчики позволяет КФС реагировать на изменения, создавая цикл взаимодействия между кибернетическими и физическими компонентами.

Вычислительная компонента – компоненты КФС, которые занимаются обработкой данных, логикой и программным управлением. Именно здесь находятся микроконтроллеры, которые управляют физическими процессами системы.

Прикладная компонента – Это область, в которой физические процессы взаимодействуют с вычислительной компонентой через датчики и исполнительные устройства. Это может быть, например, программа автоматического управления или система мониторинга состояния.

Датчики – механизмы и датчики, необходимые для обратной связи, которые обнаруживают изменения внутри КФС и преобразуют эти данные в цифровые сигналы, доступные для последующей обработки на уровне вычислительной компоненты.

Исполнительные устройства – физические устройства, непосредственно выполняющие инструкции от вычислительной платформы. Примерами таких устройств могут быть различные электрические моторы, гидравлические системы.

Целевые физические объекты и процессы – Это конкретные элементы физического мира, над которыми КФС осуществляет контроль или наблюдение. Они могут включать в себя различные машины, транспортные средства или промышленные процессы.

1.1.2 Введение в процесс-ориентированные языки

Процесс-ориентированные языки [8], являются инструментом для разработки и регулирования сложноструктурированных процессов в различных КФС. Они разрабатываются с целью оптимизации программирования встраиваемых систем.

В отличие от языков общего назначения, процесс-ориентированные языки предоставляют инструменты для более точного описания и управления сложными алгоритмами, которые часто используются в современном оборудовании, например, в автомобильной промышленности. Эти языки позволяют разрабатывать программы, которые лучше отражают реальные процессы и предоставляют большую гибкость и контроль за исполнением отдельных процессов в системе [9].

Reflex и IndustrialC являются примерами таких языков и предлагают немного разные подходы для программирования встраиваемых систем. Reflex выделяется своей простотой и интуитивностью, в то время как IndustrialC расширяет спектр возможностей для работы с микроконтроллерами серии ATmega, что делает его более сложным, но также более универсальным инструментом.

1.1.2 Введение в предметно-ориентированные языки

Предметно-ориентированные языки (DSL) нацелены на решение задач в конкретных областях. Они применяются в разных сферах, начиная от настройки программ и заканчивая моделированием процессов [10]. Такой язык является специализированным инструментом, который разрабатывается для удовлетворения конкретных потребностей в конкретных предметных областях. Например, Structured Query Language (SQL) является, пожалуй, самым популярным примером DSL, который ориентирован на предметную область, связанную с управлением данных, в реляционных базах данных.

Проектирование программной компоненты с помощью предметно-ориентированного языка (DSL) для КФС имеет ряд преимуществ перед использованием языков общего назначения, таких как C, C++, Assembly. Эти преимущества заключаются в конкретной специализации и простоте использования по сравнению с языками общего назначения [11]. Специализации DSL под конкретную предметную область делает его более эффективными для решения задач этой области. DSL специально ограничивает разработчика в количестве доступных инструкций в языке, необходимых для решения специфической задачи. По сути, для специалиста заранее должны быть определены правила и последовательности написания программ, что существенно упрощает разработку. Это особенно важно для микропроцессоров ATmega, где ресурсы ограничены, и каждая оптимизация может существенно улучшить производительность.

Для создания DSL важно понимать специфику предметной области, для которой он предназначен. В нашем случае — это разработка программ для микропроцессоров ATmega.

Процесс-ориентированные языки, являются инструментом для разработки и регулирования

1.1.3 Особенности микроконтроллеров ATmega

Микроконтроллеры ATmega являются одними из самых распространенных и широко применяются во встраиваемых системах благодаря своей гибкости, производительности и доступности. В настоящее время такие микроконтроллеры широко используются во многих областях промышленного применения, в которых ранее доминировали другие устройства. Их сильные стороны, такие как: вычислительная мощность, низкая стоимость и малые размеры позволяют им стать заменой промышленных контроллеров ПЛК и аналоговых электронных схем [12].

В качестве примера микроконтроллера данной серии можно рассмотреть ATmega8 [13]. Следует отметить, что устройство представляет собой 8-битный микроконтроллер с RISC-архитектурой, что обеспечивает исполнение мощных инструкций за один тактовый цикл, позволяя достичь скорости выполнения операций до 16 MIPS при тактовой частоте 16 МГц.

Однако существующие инструменты для программирования ATmega имеют ограничения, что делает их неудобными для разработки сложных проектов. Актуальной задачей в данной сфере является разработка универсального инструмента, который будет учитывать особенности и позволит в полной мере реализовать потенциал ATmega микроконтроллеров.

1.2 Анализ Существующих Решений

На текущий момент существует несколько языков программирования, предназначенных для встраиваемых систем, включая C, C++, и Assembly [14]. Однако, применение этих языков не всегда оптимально для разработки приложений, требующих высокой производительности и эффективного использования ограниченных ресурсов микроконтроллеров ATmega. Reflex и IndustrialC были разработаны для решения этих проблем, но существующие версии этих языков ограничены в функциональности и поддержке.

В ходе академического исследования были выявлены и проанализированы слабые стороны существующих процесс-ориентированных языков. Reflex, хоть и легок в освоении, не предлагает ряда критически важных функций, вроде управления периферийными устройствами, структур данных, таких как массивы, конструкций, таких как циклы. IndustrialC, с другой стороны, обеспечивает большую гибкость, но сложен для новичков и требует сложных инструментов.

В процессе анализа литературы были рассмотрены источники, посвященные вопросам программирования встраиваемых систем и языкам программирования, таким как Reflex и IndustrialC. В работах [1] и [2] проведен анализ существующих процесс-ориентированных

языков программирования Reflex и IndustrialC, выявлены их преимущества и недостатки в контексте программирования микроконтроллеров ATmega. Также проанализированы существующие проблемы, связанные с ограничениями сред программирования для ATmega, и выделена необходимость унификации синтаксиса языков программирования Reflex и IndustrialC.

Учитывая это, необходимо пересмотреть существующие подходы к программированию микроконтроллеров ATmega, чтобы найти баланс между простотой и функциональностью. Это также повлечет за собой повышение удобства для пользователя и одновременно обеспечит надежную поддержку различных аппаратных компонентов и системных архитектур.

1.2.1 Особенности языка Reflex

Reflex – это процесс-ориентированный диалект языка C, разработанный в институте автоматики и электрометрии Сибирского отделения Российской академии наук. Одним из его главных преимуществ является простой синтаксис, который позволяет снизить порог входа для новых разработчиков. С помощью данного языка можно описывать сложные процессы, которые будут работать по принципу конечных автоматов. Процессы могут взаимодействовать друг с другому, управлять физическими портами микроконтроллеров. Однако данный язык ограничен в своей функциональности. Например, в нем отсутствует поддержка прерываний.

1.2.2 Особенности языка IndustrialC

IndustrialC – язык, нацеленный на программирование систем управления на базе микроконтроллеров AVR архитектуры. Преимуществом данного языка является более богатый набор возможностей по сравнению с Reflex. Здесь присутствует возможность работы с прерываниями, регистрами и битами микроконтроллера. Тем не менее, у IndustrialC высокий порог входа для новых разработчиков. Его освоение вносит дополнительные сложности в разработку. Также данный язык ограничен возможностью работы лишь с AVR архитектурой.

1.2.2 Особенности архитектуры AVR

Архитектура AVR [15] представляет собой тип микроконтроллеров, базирующуюся на принципах современной интеграции полупроводников и программных возможностей, характерных для 1990-х годов. Эта архитектура включает в себя набор общих рабочих регистров, доступных для выполнения операций в арифметико-логическом блоке (ALU) всего за один тактовый цикл. Это обеспечивает высокую скорость обработки данных при минимальной задержке.

Главной особенностью архитектуры AVR является использование гарвардской архитектуры, для которой характерно отделение пространства программной памяти от пространства данных.

Программная память в архитектуре AVR может вмещать до 8 Мбайт, а объем данных — до 16 Мбайт, что делает эти микроконтроллеры пригодными для широкого спектра приложений.

Кроме того, AVR микроконтроллеры обладают улучшенной системой работы с указателями, которые могут использоваться для косвенного адресования с автоматическим увеличением или уменьшением значения указателя после каждого обращения. Это особенно эффективно при реализации стека или копировании данных между различными областями памяти.

Адресация в AVR позволяет эффективно управлять как статическими, так и динамическими структурами данных. Например, режим косвенной адресации с смещением предоставляет возможность эффективного доступа к элементам структур или массивов, что широко используется при программировании на C.

1.3 Выявление проблем и формулирование гипотезы исследования

1.3.1 Неудовлетворенность существующими инструментами

На данный момент существует ряд проблем, с которыми сталкиваются специалисты при разработке под данную платформу микроконтроллеров. Одной из таких проблем является излишняя сложность языков программирования, которые применяются для разработки. Среди них такие языки, как C, C++, Assembly. Они предоставляют излишнюю свободу в реализации программ и количестве доступных конструкций, что значительно затрудняет разработку и дальнейшее поддержание кодовой базы. Четкая структура программы и ограничение количества возможных конструкций в языке позволило бы значительно упростить разработку.

Также важными для разработки являются инструменты, в которых мы пишем код. Существующие IDE для встраиваемых систем не отвечают потребностям разработчиков [16]. Платформа Arduino хоть и предоставляет базовый функционал среды разработки, но для сложных проектов его явно недостаточно. Хотелось бы иметь среду разработки с богатым функционалом, которая в себя включает подсветку синтаксиса, автодополнение кода, автоматическую кодогенерацию, кроссплатформенность [17]. Поэтому существует необходимость разработки такой интегрированной среды разработки под язык, который планируется создать.

Перечисленные недостатки существующих инструментов делают особо актуальными разработки новых инструментов в данной области.

1.3.2 Гипотеза исследования

Проанализировав сложившиеся подходы к разработке под микроконтроллеры ATmega, можно сделать вывод, что существует необходимость в совершенствовании текущих инструментов, а также, создании нового языка программирования, который объединит в себе все достоинства языков Reflex и IndustrialC.

Гипотеза исследования заключается в том, что создание универсальной среды программирования, на основе усовершенствованного языка Reflex, с учетом требований встраиваемых систем и объединение его с языком IndustrialC должно привести к созданию эффективного инструментария для разработки встраиваемых систем на базе микроконтроллеров ATmega. Такой подход может сократить трудоемкость процесса разработки и сделать его более интуитивным.

1.4 Требования к разрабатываемому инструментарию

Интегрированная Среда Разработки (IDE) представляет собой неотъемлемый инструмент для разработчиков в области встраиваемых систем, обеспечивая единое пространство для написания, отладки и анализа программного кода. В контексте разработки на языках программирования Reflex и IndustrialC для микроконтроллеров ATmega, IDE становится критическим компонентом, обеспечивающим эффективное использование аппаратных ресурсов и удобство разработки.

Создание такой среды разработки представляет собой комплексную задачу, требующую глубокого понимания как аппаратных, так и программных аспектов. Для успешного развития IDE необходимо учитывать особенности языков программирования, специфику аппаратных ресурсов ATmega, а также требования разработчиков в области встраиваемых систем. Разрабатываемый инструментарий должен предоставлять удобные средства для работы с периферией микроконтроллеров, обеспечивать отладку и мониторинг ресурсов в реальном времени.

Также следует рассмотреть вариант создания web-ориентированной IDE, которая предоставит дополнительную гибкость разработчикам, позволяя им работать над проектами удаленно, что ускоряет процесс разработки и снижает стоимость владения таким инструментом [18].

Анализ существующих проблем в языках Reflex и IndustrialC позволяет выделить ключевые **требования** к разрабатываемому инструментарию:

1. Генерируемый код должен быть платформонезависимым.
2. В язык Reflex должна быть введена поддержка: функций, процедур обработки прерываний, критических секций кода, volatile переменных, циклов for, массивов, а

также должна быть введена возможность использования локальных переменных в функциях, процедурах обработки прерываний и состояниях процесса.

3. Выходной файл должен генерироваться в отдельной директории.
4. В качестве целевого языка для кодогенератора должен быть использован язык Си.
5. Обеспечить возможность преобразования кода в язык Си.
6. Должна быть обеспечена возможность переиспользования проекта для создания web IDE.

1.5 Основные выводы главы

В первой главе была проанализирована предметная область, выявлены основные аспекты разработки встраиваемых систем на базе микроконтроллеров ATmega, а также изучены проблемы, с которыми сталкиваются разработчики при использовании языков Reflex и IndustrialC. Этот анализ позволил определить пути решения возникающих проблем и сформулировать требования к новой интегрированной среде разработки (IDE), которая будет объединять процесс-ориентированную парадигму Reflex и гибкость IndustrialC.

Основные проблемы существующих инструментов, такие как отсутствие поддержки периферийных устройств, ограниченный набор конструкций, сложный синтаксис и высокая трудоемкость разработки, были учтены при определении требований к новой IDE. Для устранения этих проблем предлагается дальнейшее развитие грамматики языков Reflex и IndustrialC, а также создание интегрированной среды разработки, обеспечивающей более удобные и эффективные инструменты.

В рамках новой IDE предполагается реализовать поддержку многомодульных проектов, работу с периферией, а также возможность удаленного доступа через web-интерфейс. Для этого будет проведена коррекция синтаксиса Reflex и разработан синтаксически ориентированный редактор, который позволит ускорить процесс программирования.

В данной главе были заложены основы для дальнейшей разработки, анализа и проверки созданной IDE. Объединение преимуществ языков Reflex и IndustrialC в новой IDE позволит создать инструмент, который будет лучше соответствовать потребностям разработчиков встраиваемых систем. В последующих главах будет рассмотрен процесс усовершенствования вышеупомянутых языков, реализации данной интегрированной среды разработки и проведена апробация ее функционала на реальных проектах.

Таким образом, данная глава обеспечивает теоретическую и аналитическую базу для дальнейшего изложения результатов исследования и разработки в последующих главах выпускной квалификационной работы.

Глава 2: Язык Reflex

Для разработки языка программирования важно определиться с областью его применения. Предметно-ориентированные языки (DSL) предназначены для конкретных задач и отличаются от языков общего назначения своей специализацией. Они часто создаются для автоматизации процессов, управления встраиваемыми системами или реализации других узконаправленных функциональностей. Именно для таких целей планируется реализовать предметно-ориентированный язык в рамках данной работы. Важно также понять, какие требования предъявляются к этому языку в контексте встраиваемых систем на базе микроконтроллеров ATmega.

В Reflex основное внимание уделяется процесс-ориентированному подходу, что позволяет легко моделировать сложные системы, которые состоят из множества взаимосвязанных процессов. Данный подход также помогает упростить процесс программирования таких систем, благодаря возможности использования различных процессов для описания и выполнения необходимых задач. Это особенно полезно для встраиваемых систем, где ресурсы ограничены, а эффективность кода имеет решающее значение.

Одной из ключевых задач при разработке Reflex является совершенствование грамматики. Необходимо ввести недостающие синтаксические конструкции, расширить возможности языка и оптимизировать его для работы с периферией микроконтроллеров ATmega. Также важным моментом является унификация синтаксиса Reflex и IndustrialC для обеспечения совместимости с AVR архитектурой. Благодаря этим изменениям, язык в перспективе становится более гибким и удобным инструментом для использования в промышленной автоматизации и других приложениях.

В данной главе рассмотрим синтаксис разрабатываемого языка Reflex. Также здесь будут описаны нововведенные в язык конструкции и правила кодогенерации для них.

Кроме того, в этой главе мы рассмотрим особенности трансляции Reflex в язык Си. Этот процесс включает в себя создание лексера и парсера, работу с токенами, терминальными и нетерминальными символами, а также алгоритмы кодогенерации.

Кроме того, существует необходимость введения элементов контроля версий для данного проекта, что позволит лучше организовать работу над проектом в команде.

Все перечисленные аспекты помогут нам достичь гипотезы исследования, заявленной в первой главе, и станут основой для дальнейшего исследования.

2.1 Базовые элементы синтаксиса. Структура программы

Reflex похож на языки, которые используют C-подобный синтаксис, что делает его интуитивно понятным для большинства программистов. Программа начинается с ключевого

слова **program**, за которым следует название программы. Основные блоки в программе заключены в фигурные скобки. Этот блок содержит такие элементы, как константы, переменные, порты, процессы, состояния и функции.

В общем виде программа на языке Reflex имеет следующий вид:

```
program ИмяПрограммы {
  clock 0t10ms;
  // Определения констант
  const тип ИмяКонстанты = значение;
  // Определение портов
  bool inputSignal as input (read = ReadPort, config = ConfigPort, bit = BitName);
  bool outputSignal as output (read = ReadPort, write = WritePort, config = ConfigPort, bit = BitName);

 // Импорт платформозависимых битов, регистров, векторов
  import PlatformSpecificBitsAndVectors {
    // Пример ввода битов
    register      REGISTER_NAME;
    bit                 BIT_NAME1;
    bit                 BIT_NAME2;
    bit                 BIT_NAME3;
    bit                 BIT_NAME4;
    bit                 BIT_NAME5;
    // Пример векторов прерываний
    vector              INTERRUPT_VECTOR1;
    vector              INTERRUPT_VECTOR2;
    vector              INTERRUPT_VECTOR3;
    vector              INTERRUPT_VECTOR4;
    vector              INTERRUPT_VECTOR5;
  }
// Обработчики прерываний (ISR)
  ISR (INTERRUPT_VECTOR1) {
    // Код обработчика прерывания для INTERRUPT_VECTOR1
    globalVar = functionName(globalVar);
  }
  ISR (INTERRUPT_VECTOR2) {
    // Код обработчика прерывания для INTERRUPT_VECTOR2
    if (inputSignal) {
       outputSignal = !outputSignal;
    }
  }
  // Определения перечислений
  enum ИмяПеречисления {
```

```
    Перечисление1,
    Перечисление2 = значение
};
// Глобальные переменные
int16 globalVar = начальноеЗначение;
// Определение массивов
int16 dataArray[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
bool statusArray[5] = {true, false, true, false, true};
float sensorReadings[4];
// Функции
типФункции имяФункции(тип аргумента) { ... }
// Определение процессов
process ИмяПроцесса [active] {
    state Состояние1 [looped] {
        // Код состояния
        if (условие) {
            set state Состояние2;
        }
        timeout (0t2s) {
            // Код таймаута
            set state Состояние1;
        }
    }
    state Состояние2 {
        // Код состояния
    }
}
}
```

Листинг 1 - Структура программы на языке Reflex

Данный листинг представляет собой базовую структуру программы на языке Reflex и демонстрирует использование основных конструкций языка.

Таблица 1 - ключевые слова для разработки на языке Reflex.

Ключевое слово	**Описание**
program	Начало определения программы.
import	Импорт платформозависимых битов, регистров и векторов.

clock	Установка таймера для всей программы.
const	Определение константы.
enum	Определение перечисления.
input/output	Определение входных и выходных портов.
process	Определение процесса.
state	Определение состояния в процессе.
looped	Указание на то, что состояние является зацикленным.
timeout	Выполнение кода после определённого промежутка времени.
set state	Переключение состояний внутри процесса.
ISR	Определение обработчика прерывания.

Следующий листинг демонстрирует базовые возможности языка. Здесь описан процесс, который контролирует работу вентилятора через выходной порт микроконтроллера с помощью данных, которые поступают на входной порт микроконтроллера от температурного датчика.

```
program FanControl {

        clock 0t10ms;
        bool tempSensor ureSensor as input (read = PIN0, config = DDR0, bit = BIT1);
        bool fanControl as output (read = PIN0, write = PORT0, config = DDR0, bit = BIT2);

        process MonitorTemperature {
                float temperature;
                state CheckTemperature looped {
                        temperature = tempSensor;
                        if (temperature > 30.0) {
                                fanControl = true;
                        } else {
                                fanControl = false;
                        }
                }
        }
}
```

Листинг 2 - Пример программы на языке Reflex

Это типичный пример того, как программы на языке Reflex используют управляющие конструкции в процесс-ориентированной парадигме.

Аналогичная программа, написанная на языке C или C++ была бы в несколько раз больше по объему кода и менее читаема.

2.2 Процессы и состояния

Процессы и состояния — это основные элементы синтаксиса в Reflex, которые позволяют регулировать процессы в КФС.

Процесс, по своей сути, похож на конечный автомат. Управление процессом происходит с помощью событий. Такими событиями могут быть таймауты, изменения переменных, переключение состояний внутри процесса. Процесс реагирует на такие события и может изменять выходные данные, поступающие на микроконтроллер, либо запускать или изменять состояния в других процессах.

Далее в таблице приведены основные ключевые слова языка Reflex для описания процессов.

Процессы в Reflex являются основными единицами выполнения кода. Каждый процесс

может содержать переменные, состояния, а также импортированные переменные из других процессов. Процесс можно сделать активным с помощью ключевого слова **active**, чтобы выполнять его код при запуске программы.

Таблица 2 - ключевые слова для разработки на языке Reflex.

Ключевое слово	Описание
process	Определяет блок кода как процесс, который может содержать переменные, состояния и импортированные переменные из других процессов.
active	Модификатор процесса, указывающий, что процесс должен быть активирован при запуске программы.
state	Определяет состояние внутри процесса, которое указывает этапы выполнения кода.
looped	Опция состояния, которая позволяет зациклить выполнение данного состояния.
shared	Указывает на переменные, которые объявлены в одном процессе и доступны в других.
timeout	Устанавливает временной интервал для состояния, определяющий период его активности до перехода к следующему шагу или состоянию.

Процессы и состояния позволяют управлять ходом исполнения программы в КФС. Например, один процесс может отвечать за управление двигателем, другой — за чтение данных с датчиков, третий — за обработку этих данных. Состояния помогают организовать последовательность действий внутри процесса.

2.3 Управляющие конструкции и циклы

Reflex использует стандартные условия **if-else**, циклы **for**, а также специфические конкретно для этого языка конструкции, такие как состояния (**state**) и процессы (**process**). Состояния в процессе позволяют определить этапы работы программы и переключаться между ними с помощью операторов **set state**, **next state**.

Ключевое слово **looped**, в состоянии в процесса указывает на бесконечное повторение данного состояния, пока не будет вызвана команда перехода в другое состояние.

Программа "ConveyorControl" демонстрирует применение состояний и циклов для управления работой конвейера. Конвейер запускается и останавливается в зависимости от сигнала внешнего датчика.

```
program ConveyorControl {
  clock 0t10ms;
  bool conveyorMotor as input (read = PIN0, config = DDR0, bit = BIT1);
  bool speedSensor as output (read = PIN0, write = PORT0, config = DDR0, bit = BIT2);

  process MonitorSpeed {
    int speed = speedSensor;
    state StartConveyor {
      conveyorMotor = true;
      set state CheckSpeed;
    }
    state CheckSpeed looped {
      if (speed > 100) {
        conveyorMotor = false;
        set state StartConveyor;
      }
    }
  }
}
```

Листинг 5 - Пример программы на языке Reflex. "ConveyorControl".

2.4 Взаимодействие процессов через состояния

Reflex позволяет процессам взаимодействовать между собой через состояния и переменные. Переменные, объявленные в одном процессе, можно разделять с другими процессами с помощью ключевого слово **shared**. Импортированные переменные могут быть использованы для передачи данных из одного процесса в другой, что позволяет организовать более сложные схемы взаимодействия между процессами.

Пример программы, которая использует взаимодействие процессов с помощью “shared” переменной:

```
program SensorControl {
  clock 0t10ms;
  bool pressureSensor as input (read = PIN0, config = DDR0, bit = BIT1);
  bool valveControl as output (read = PIN0, write = PORT0, config = DDR0, bit = BIT2);

  process PressureMonitor {
    state CheckPressure looped {
      float pressure = pressureSensor;
      if (pressure > 2.0) {
        set state ReleasePressure;
      }
    }
    state ReleasePressure {
      int16 y shared; // shared-переменная, определенная в этом процессе
      y = 1; // сигнал, что давление высокое
      valveControl = true;
      timeout (0t2s) {
        valveControl = false;
        set state CheckPressure;
      }
    }
  }
  process AlertSystem {
    shared y from process PressureMonitor; // импортирование переменной y
    state Alert looped {
      float pressure = pressureSensor;
      if (pressure > 2.5 && y == 1) {
        // вызов сигнала тревоги, если давление слишком высокое и переменная y равна 1
      }
    }
  }
}
```

Листинг 6 - Пример программы с разделяемой переменной на языке Reflex

Такой подход позволяет создавать сложные встраиваемые системы, которые могут реагировать на различные ситуации, связанные с целевыми физическими объектами и процессами.

2.5 Введенные в Reflex конструкции

В рамках данной работы язык Reflex был расширен новыми синтаксическими конструкциями, которых раньше в нем не было. Среди таких конструкций можно выделить циклы for, массивы, функции, ISR обработчики прерываний, atomic блоки кода, volatile переменные.

2.5.1 Циклы for

Циклы представляют собой важный элемент в программировании, обеспечивающий возможность многократного выполнения блока кода. В языке Reflex цикл for был введен с использованием следующей грамматики:

```
IterationStatement:
        ForStatement;

ForStatement:
        'for' '('forList=ForList ')' '{'
                statements=StatementSequence
        '}';

ForList:
        initializations += ForInitializationSection
        (',' initializations += ForInitializationSection)*';'
        (loopcondition = LogicalOrExpression ';')
        step += Expression (',' step += Expression)* ;

ForInitializationSection:
        variable=[IdReference] '=' initValue = Expression ;
```

Листинг 3 - Введенная грамматика для циклов **for** в нотации Xtext

Рассмотрим также Xtend код, который отвечает за генерацию кода в язык Си из Reflex:

```
def translateIterationStat(State state, IterationStatement statement) {
        if (statement instanceof ForStatement) {
                return translateForStat(state, statement);
        }
        return "'//here was not supported iteration loop;'"
```

```
}

def translateForStat(State state, ForStatement statement) {
	var ForList forList = statement.forList;
	// Собираю все инициализации в одну строку (Например: i = 0, j = 0)
	var String initializations = forList.initializations.stream
	.map[expressionGenerator
	.translateExpr(it.variable)+'='+expressionGenerator.translateExpr(it.initValue)]
	.toList()
	.join(", ")
	// Собираю все step в одну строку (Например: i++, j++)
	var String stepsExpression = forList.step.stream
		.map[expressionGenerator.translateExpr(it)]
		.toList()
		.join(", ")

	return
	'''
	for(«initializations»;«expressionGenerator.translateExpr(forList.loopcondition)»; «stepsExpression»)
	{
		«FOR stat : statement.statements.statements»
			«translateStatement(state, stat)»
		«ENDFOR»
	}
	'''
}
```

Листинг 4 - Правила кодогенерации для циклов **for** в нотации Xtend

Таким образом в язык Reflex была добавлена поддержка циклов **for**.

2.5.2 Массивы

Еще одним значительным нововведением в языке Reflex стало введение массивов, синтаксис которых имеет сходство с C-подобными массивами. В следующем листинге представлена грамматика для работы с массивами в нотации Xtext:

```
ArrayAssignmentStatement:
	(array=ArrayReference) '=' value=ArrayInitialization;

ArrayElementAssignmentStatement:
	(element=ArrayElementReference) ('=' value=Expression)?;

ArrayReference:
  name=ID;
```

```
ArrayElementReference:
        array=[ArrayReference] '[' index=Expression ']';

ArraySpecificationInit:
        type=Type ref=ArrayReference
        '[' (size=Expression) ']'
        ('=' values=ArrayInitialization)?;

ArrayInitialization :
        '{' elements+=Expression (',' elements+=Expression)* '}';
```

Листинг 5 - Введенная грамматика для массивов в нотации Xtext

Для генерации кода, связанного с массивами, используется язык Xtend. Был реализован класс, который обрабатывает синтаксические конструкции, связанные с массивами:

```
class ArrayGenerationHelper {
    static ReflexIdentifiersHelper identifiersHelper = new ReflexIdentifiersHelper
    static ExpressionGenerationHelper expressionGenerator
        = new ExpressionGenerationHelper(identifiersHelper)

    def static getArrayRepresentation(ArraySpecificationInit array) {
        var String name = array.ref.name
        var String size = expressionGenerator.translateExpr(array.size)
        if (array.values !== null
        && array.values.elements !== null
        && array.values.elements.size > 0) {
            return '''«name»[«size»] = «FOR value : array.values.elements
            BEFORE '{' SEPARATOR ', ' AFTER
            '}'»«getStringAssigmentElement(value)»«ENDFOR»'''
        }
        return '''«name»[«size»]'''
    }

    static def getStringAssigmentElement(Expression element)
    '''«expressionGenerator.translateExpr(element)»'''
}
```

Листинг 6 - Правила кодогенерации для массивов в нотации Xtend

Таким образом в язык Reflex была добавлена поддержка массивов. Введение массивов в язык Reflex значительно расширяет его функциональные возможности.

2.5.3 Функции

Еще одним значительным нововведением в языке Reflex стали функции, которые ранее не поддерживались. В следующем листинге представлена грамматика для функций в нотации Xtext:

```
LocalVariable:
        (PhysicalVariable | ProgramVariable) ";";

Function:
        returnType=Type name=ID
        "(" (parameters+=ProgramVariable ("," parameters+=ProgramVariable)*)? ")"
        (declaration=FunctionDeclaration | definition=CompoundStatement);

FunctionDeclaration:
  ";"; // Отмечает, что это только объявление функции
```

Листинг 7 - Введенная грамматика для массивов в нотации Xtext

Кроме того, был добавлен механизм определения области видимости функций, а также добавлены локальные переменные, которые видны исключительно внутри функции. Это необходимо для корректного отображения переменных в программе. Рассмотрим пример кода, который обрабатывает локальные переменные:

```
private def IScope getIdReferenceScope(EObject ctx) {
        if (func !== null) {
                if (func.parameters !== null) {}
                        candidates.addAll(func.parameters)
                if (func.definition !== null && func.definition.localVariables !== null) {
                        // обрабатываем локальные переменные функции
                func.definition.localVariables.forEach [
                if (it instanceof LocalVariable) {
                candidates.add(it)
}
]
                }
}
        return Scopes.scopeFor(candidates)
}
```

Листинг 8 - Введенная грамматика для массивов в нотации Xtext

Таким образом в язык Reflex была добавлена поддержка функций. Введение функций язык Reflex значительно расширяет его функциональные возможности.

2.5.3 ISR. Процедуры обработки прерываний

Еще одним значительным нововведением в языке Reflex стало введение процедур обработки прерываний (ISR). ISR позволяют регистрировать собственные обработчики

прерываний для векторов микроконтроллера ATmega. Это нововведение значительно расширяет функциональные возможности языка и упрощает работу с прерываниями.

В следующем листинге представлена грамматика для работы с ISR в нотации Xtext:

```
IsrStatement:
        "ISR" "(" (vector=[IdReference])")"
        definition=CompoundStatement;
```

Листинг 9 - Введенная грамматика для ISR процедур обработки прерываний в нотации Xtext

Таким образом, в язык Reflex была добавлена поддержка процедур обработки прерываний (ISR). Введение ISR в язык Reflex значительно расширяет его функциональные возможности, позволяя разработчикам эффективно управлять прерываниями и улучшая взаимодействие с аппаратным обеспечением микроконтроллеров.

2.5.3 Критические секции и volatile переменные

Для повышения надежности и предсказуемости работы встраиваемых систем в язык Reflex были добавлены критические секции и поддержка volatile переменных.

Критические секции позволяют запретить прерывания для команд внутри блока atomic, что гарантирует их выполнение без прерываний. Это особенно важно для операций, которые должны быть выполнены атомарно.

В следующем листинге представлена грамматика для критических секций в нотации Xtext:

```
AtomicBlock:
        "atomic" "{" statements+=Statement* "}";
```

Листинг 10 - Введенная грамматика для критических секций в нотации Xtext

Поддержка volatile переменных была добавлена для обеспечения правильного обращения с переменными, которые могут быть изменены внешними процессами, такими как прерывания. Ключевое слово volatile информирует компилятор о том, что значение переменной может измениться в любой момент, и не должно кэшироваться.

В следующем листинге представлена грамматика для volatile переменных в нотации Xtext:

```
PhysicalVariable:
        (volatile?="volatile")? mappingType=MappingType? type=Type name=ID "as"
        mapping=PortMapping ;

ProgramVariable:
        (volatile?="volatile")? type=Type name=ID;
```

Листинг 11 - Введенная грамматика для volatile переменных в нотации Xtext

Таким образом, в язык Reflex была добавлена поддержка критических секций и volatile переменных, что значительно расширяет его функциональные возможности и повышает надежность программ для встраиваемых систем.

2.6 Выводы по Reflex

Была усовершенствована грамматика данного языка, добавлены недостающие синтаксические конструкции.

Обновление грамматики и расширение возможностей языка Reflex, включая поддержку циклов for, функций, обработчиков прерываний, массивов, критических секций, volatile переменных, значительно повышают его функциональность и удобство использования. Эти изменения способствуют созданию более структурированного и понятного кода, что особенно важно в разработке сложных программных систем

Таким образом, Reflex является перспективным инструментом для разработки встраиваемых систем на базе микроконтроллеров ATmega. С его помощью можно создавать сложные структуры, использовать управляющие конструкции, определять переменные и типы данных, а также работать с портами ввода и вывода у микроконтроллеров. Reflex позволяет создавать сложные программы, оптимизированные для встраиваемых систем, что делает его мощным инструментом для решения широкого круга задач.

Глава 3: Выбор инструментов и создание Reflex IDE

Разработка предметно-ориентированного языка программирования влечёт за собой не только трудоемкий процесс создания синтаксиса и семантики, но и необходимость тщательного подбора технологического стека. Этот выбор становится ключевым аспектом, определяющим гибкость, масштабируемость и удобство использования нового языка в разнообразных проектах.

В данной главе рассмотрим особенности реализации интегрированной среды разработки, для вышерассмотренного предметно-ориентированного языка Reflex. Особое внимание уделим существующим инструментам, их недостаткам и тому, какое улучшения планируется внедрить в эти инструменты.

3.1 Выбор технологий

Выбор подходящих технологий – важный этап для в создании инструментов разработки. Правильно подобранный технологический стек позволяет разработчикам максимально реализовать потенциал языка, а также упрощает его дальнейшее внедрение и адаптацию под специфические задачи пользователей.

3.1.1 Особенности web-IDE по сравнению с обычными IDE

В рамках проектирования и реализации интегрированной среды разработки необходимо определить формат взаимодействия разработчика с этой средой.

Рассмотрим два возможных варианта: использовать традиционную IDE, которая запускается локально на машине разработчика, или web-IDE, взаимодействие с которой осуществляется через браузер. Оба подхода имеют свои преимущества и недостатки, и конечное решение зависит от ряда факторов.

Традиционные IDE

Традиционные IDE обычно устанавливаются на компьютере разработчика и используют ресурсы этой машины для компиляции и запуска кода. Они, как правило, более быстрые и надежные, поскольку не зависят от интернет-соединения. Однако, несмотря на стабильность и быстроту, такие среды имеют несколько серьезных недостатков.

Во-первых, традиционные IDE подвержены риску потери данных, если что-то случится с машиной разработчика, например, поломка или сбой операционной системы. Во-вторых, они ограничены одной рабочей станцией, что делает совместную разработку сложной. Если команда разработчиков работает в разных местах, обмен кодом и совместная работа требуют дополнительных инструментов.

Web-IDE

Web-IDE, напротив, работает через браузер, что дает несколько значимых преимуществ. Во-первых, такие IDE доступны из любого места с доступом в интернет. Это

означает, что разработчики могут работать удаленно, без необходимости инсталляции дополнительных программных пакетов. Во-вторых, web-IDE обычно используют облачные сервисы для хранения данных, что снижает риск потери данных при сбое оборудования.

Одно из ключевых преимуществ web-IDE — это возможность командной работы в режиме реального времени. Возможность одновременного редактирования проекта значительно упрощает совместную работу.

Для реализации данного проекта было решено выбрать способ с web-IDE, из-за неоспоримых преимуществ данного подхода.

Reflex IDE будет включать в себя языковой сервер, который будет взаимодействовать с веб-редактором кода и обрабатывать все необходимые запросы, такие как автодополнение кода, проверка синтаксиса, рефакторинг и другие функции, необходимые для полноценной IDE.

Таким образом, использование web-IDE позволит Reflex IDE быть более кроссплатформенной, гибкой и готовой для совместной работы платформой. Такой подход также будет способствовать более быстрой и эффективной разработке, поскольку разработчики могут работать из любой точки мира, имея доступ к сети.

3.2.1 Language Server Protocol для создания web-IDE

Преимущества LSP для web-IDE

Language Server Protocol (LSP) [19] — это стандарт для взаимодействия между языковым сервером и редакторам кода по HTTP протоколу. Данный протокол обеспечивает высокий уровень гибкости, поскольку не привязан к конкретной интегрированной среде разработки (IDE). Это делает его идеальным инструментом для разработки Reflex IDE. С помощью LSP можно легко интегрировать языковой сервер с различными редакторами кода, что обеспечивает максимальную совместимость и широкие возможности для расширения.

LSP работает по протоколу HTTP, передавая данные в формате JSON, что обеспечивает кроссплатформенность. Основная идея заключается в том, что языковой сервер обрабатывает различные запросы от редактора кода, такие как автодополнение, анализ синтаксиса, рефакторинг и другие. Это позволяет использовать один языковой сервер для работы с несколькими редакторами, что значительно упрощает разработку и обеспечивает широкую поддержку различных IDE.

Принцип работы LSP можно разбить на три ключевых этапа:

Открытие файла: Пользователь открывает файл в редакторе, редактор отправляет запрос на языковой сервер, который содержит информацию о файле и его содержимом.

Изменение файла: Для каждого вносимого изменения в файл, редактор отправляет запрос на языковой сервер. Это позволяет серверу поддерживать актуальное состояние кода и

обрабатывать запросы, связанные с автодополнением, анализом синтаксиса и другими функциями.

Закрытие файла: При закрытии файла, редактор отправляет языковому серверу запрос о завершении обработки. Данный запрос необходим для освобождения ресурсов сервера.

Таким образом, LSP способен обеспечить взаимодействие между языковым сервером и редакторами кода на стандартизированной основе. Это также открывает возможности для интеграции Reflex IDE с популярными средами разработки, такими как Eclipse Theia, Visual Studio Code и другими.

Протокол LSP имеет несколько важных преимуществ, которые делают его идеальным выбором для реализации Reflex IDE. Во-первых, он обеспечивает независимость от конкретной IDE, что позволяет интегрировать Reflex с различными редакторами кода без необходимости разрабатывать отдельные модули для каждой IDE. Во-вторых, LSP упрощает интеграцию Reflex с другими языками программирования, что позволяет расширять функциональность IDE. В-третьих, протокол обеспечивает стабильное взаимодействие и снижает вероятность ошибок, поскольку работает по стандартному протоколу HTTP.

Учитывая эти преимущества,было решено, что LSP — оптимальный выбор для Reflex IDE, который обеспечит необходимую гибкость, стабильность и расширяемость для разрабатываемого инструмента. Использование LSP также упрощает разработку Reflex IDE, поскольку позволяет использовать готовые решения для взаимодействия с редакторами кода. Такой подход гарантирует, что Reflex IDE будет совместим с популярными IDE и сможет легко интегрироваться с другими языками программирования, что делает LSP незаменимым инструментом для реализации Reflex IDE.

3.1.2 Сравнение популярных инструментов для реализации web-IDE

Чтобы выбрать подходящий инструмент для создания web-IDE, было изучено несколько популярных решений. В их числе Eclipse Theia, Visual Studio Code (VS Code) и Atom. Данные среды предоставляют разные возможности и поддерживают различные подходы к созданию IDE. Давайте рассмотрим каждую из них более подробно, чтобы понять, почему мы выбрали Eclipse Theia для Reflex IDE.

Eclipse Theia

Eclipse Theia [20] — это современная web-IDE, разработанная на базе технологий Eclipse. Она поддерживает Language Server Protocol (LSP). Это важный фактор, поскольку Reflex IDE должен взаимодействовать с языковым сервером через LSP. Theia также предоставляет инструменты для расширения и возможности для кастомизации, что также является важным преимуществом.

Одним из ключевых преимуществ Eclipse Theia является её гибкость. Поскольку эта среда работает через браузер, она может быть доступна из любой точки с доступом в сеть, sчто упрощает командную работу и позволяет разработчикам легко обмениваться кодом и работать над проектами в режиме реального времени. Кроме того, Theia предлагает встроенные инструменты для разработки и отладки, что делает её привлекательной для нашего проекта.

Visual Studio Code

Visual Studio Code [21] — это популярная среда разработки с открытым исходным кодом от Microsoft. Она также поддерживает LSP, что позволяет ей взаимодействовать с различными языковыми серверами. Однако у VS Code есть некоторые ограничения в плане интеграции с другими фреймворками. Например, она менее гибкая по сравнению с Eclipse Theia, что может ограничивать возможности кастомизации.

Тем не менее, VS Code обладает большим сообществом разработчиков и множеством доступных расширений, что делает её привлекательной для многих пользователей. Однако, при сравнении с Eclipse Theia, эта среда может быть менее подходящей для нашего проекта из-за ограничений в интеграции с другими фреймворками и меньшей гибкости.

Atom

Atom [22] — это легкий редактор кода, также имеющий открытый исходный код. Его основной акцент — на расширяемости и кастомизации. Atom поддерживает различные плагины и расширения, что позволяет добавлять новые функции и возможности. Однако, по сравнению с Eclipse Theia и VS Code, Atom менее стабильный и надёжный.

Несмотря на его лёгкость и расширяемость, Atom не подходит для нашего проекта, поскольку он не обладает такой же поддержкой LSP, как Eclipse Theia, и может быть подвержен сбоям. Кроме того, Atom не предоставляет такой же уровень инструментов для разработки и отладки, что делает его менее привлекательным выбором для создания Reflex IDE.

Таблица 3 - сравнение ключевых особенностей наиболее популярных редакторов кода.

Ключевые особенности	**Eclipse Theia**	**Atom**	**Visual Studio Code**
Открытый исходный код	Да, под лицензией Eclipse Public License 2.0	Да, под лицензией MIT	Да, под лицензией MIT

Поддержка протокола LSP (Language Server Protocol)	Полная поддержка	Базовая поддержка через плагины	Полная поддержка
Расширяемость	Высокая, поддерживает расширения VS Code	Расширяемый с помощью пакетов и тем	Высокая, большое количество расширений
Среда выполнения	Используется как настольное, так и веб-приложение	Преимущественно настольное приложение	Настольное приложение
Кастомизация	Высокая, с возможностью изменения интерфейса	Высокая, но может быть менее оптимизирована	Высокая, с отзывчивым и настраиваемым интерфейсом
Сообщество и поддержка	Сильное сообщество под эгидой Eclipse Foundation	Большое сообщество, но развитие замедляется	Очень большое сообщество, активно развивается
Производительность	Оптимизирована для использования в облаке и на рабочем столе	Может быть медленнее, особенно с множеством плагинов	Быстрая и отзывчивая
Встроенная отладка	Продвинутые возможности отладки	Базовая отладка через пакеты	Продвинутые возможности отладки
Совместимость с облаком	Разработана с учетом совместимости с облаком	Ограниченная функциональность облака	Хорошая поддержка операций в облаке

Из всех рассмотренных сред выбор пал на Eclipse Theia. Она поддерживает LSP, что позволит с наименьшими трудозатратами интегрировать языковой сервер и, если

впоследствии понадобится, можно будет подключить языковой сервер к другому редактору, который также будет поддерживать LSP. Theia также обеспечивает гибкость и возможности для кастомизации. Также очень важным фактором оказалась возможность использования Eclipse Theia в качестве веб-приложения, чего, например, нельзя сказать про Visual Studio Code.

3.1.4 Выбор инструментов для реализации языкового сервера

При выборе инструмента для создания Reflex IDE мы рассматривали несколько популярных опций. Два основных конкурента Eclipse Xtext — это JetBrains MPS и ANTLR. Оба инструмента широко используются для разработки предметно-ориентированных языков (DSL), каждый со своими особенностями и возможностями. Рассмотрим более подробно данные решения для анализа их ключевых особенностей и определения наиболее подходящего инструмента.

JetBrains MPS

JetBrains MPS [23] — это платформа для создания предметно-ориентированных языков, которая использует подход метапрограммирования. Суть MPS заключается в том, что можно создавать DSL с использованием графических и текстовых элементов. Данный инструмент подходит для визуальных языков и позволяет определять правила, модели данных и поведение объектов. Однако такой подход может быть избыточным для разработки Reflex IDE. MPS ориентирован на сложные языки с визуальными элементами, что не всегда подходит для языков, подобных Reflex. Кроме того, интеграция MPS с другими IDE может быть сложной, так как он больше ориентирован на собственную среду разработки, что ограничивает его совместимость с другими платформами.

ANTLR

ANTLR [24] — это инструмент для создания парсеров и лексеров. Он предоставляет мощные средства для определения грамматик, синтаксиса и правил лексического анализа. ANTLR отлично подходит для создания языков программирования и обеспечивает гибкость в работе с парсерами. Он широко используется в академических и промышленных проектах, где необходимо создавать лексеры и парсеры для различных языков.

Однако ANTLR, несмотря на его мощные возможности для парсинга, не предоставляет таких же комплексных инструментов, как Eclipse Xtext. Это означает, что будет необходимо разрабатывать редакторы кода, средства автодополнения и другие элементы самостоятельно. Кроме того, ANTLR не предоставляет такую же тесную интеграцию с web-IDE, как Eclipse Xtext, что делает его менее подходящим для нашей задачи.

Eclipse Xtext

Eclipse Xtext [25] — это фреймворк для создания предметно-ориентированных языков (DSL), который предоставляет широкий набор инструментов для разработки языков программирования. Он включает средства для анализа, синтаксического разбирательства, генерации кода, рефакторинга и многих других задач, связанных с созданием DSL. Давайте рассмотрим, что делает Xtext уникальным и почему он был выбран для разработки Reflex IDE.

Встроенные модули парсера AST: Одной из ключевых особенностей Eclipse Xtext является автоматическое создание парсера и абстрактного синтаксического дерева (AST) на основе заданной грамматики. Это значительно упрощает процесс разработки DSL, позволяя разработчику сосредоточиться на синтаксисе и правилах языка. Встроенные модули парсера обеспечивают быструю и эффективную обработку исходного кода, что важно для создания Reflex IDE.

Eclipse Xtext также предлагает мощные инструменты для кодогенерации. Он позволяет генерировать код на основе AST, что может быть использовано для компиляции или трансляции в другие языки. Кодогенерация может быть сконфигурирована для работы с различными целевыми платформами, что делает Xtext универсальным инструментом для создания DSL. Это особенно важно для Reflex IDE, поскольку наш язык будет генерировать код, который затем используется для программирования микроконтроллеров ATmega.

Eclipse Xtext предоставляет инструменты для синтаксического анализа, позволяющие разработчикам проверять код на наличие ошибок и предупреждений. Встроенные средства автодополнения упрощают написание кода, предлагая подсказки и автоматические завершения. Эти функции важны для Reflex IDE, поскольку они улучшают опыт разработки и помогают избежать ошибок в коде.

Eclipse Xtext тесно интегрирован с Eclipse IDE, что позволяет легко использовать его в различных проектах. Эта интеграция предоставляет автоматическое создание редакторов кода, автодополнения, рефакторинга и других функций, необходимых для полноценной IDE. Reflex IDE может использовать эти возможности для ускорения разработки и повышения производительности.

Поддержка LSP: Ещё одним преимуществом Eclipse Xtext является поддержка Language Server Protocol (LSP) [26]. Это позволяет интегрировать Reflex IDE с веб-редакторами, такими как Eclipse Theia, что обеспечивает кроссплатформенность и возможность удаленного доступа. LSP позволяет взаимодействовать с различными средами разработки, расширяя возможности Reflex IDE.

Обширное сообщество и поддержка: Eclipse Xtext поддерживается активным сообществом разработчиков и Eclipse Foundation. Это обеспечивает регулярные обновления,

исправление ошибок и обширную документацию. Поддержка сообщества означает, что можно найти множество ресурсов, обучающих материалов и примеров, что облегчает разработку Reflex IDE.

В итоге, Eclipse Xtext был выбран для разработки Reflex IDE из-за его универсальности, встроенных модулей парсера AST, кодогенерации, тесной интеграции с Eclipse IDE, а также поддержки LSP. Это делает Xtext наиболее подходящим фреймворком для создания Reflex IDE, предоставляя все необходимые инструменты для разработки DSL, автодополнения, синтаксического анализа и кодогенерации. При помощи данного инструмента можно эффективно реализовать Reflex IDE, обеспечивая стабильность, производительность и кроссплатформенность.

3.2 Архитектура разрабатываемой системы Reflex IDE

Архитектура Reflex IDE базируется на концепции, предложенной Eclipse Xtext. Этот фреймворк предоставляет множество встроенных модулей, позволяющих автоматизировать процесс создания предметно-ориентированных языков (DSL).

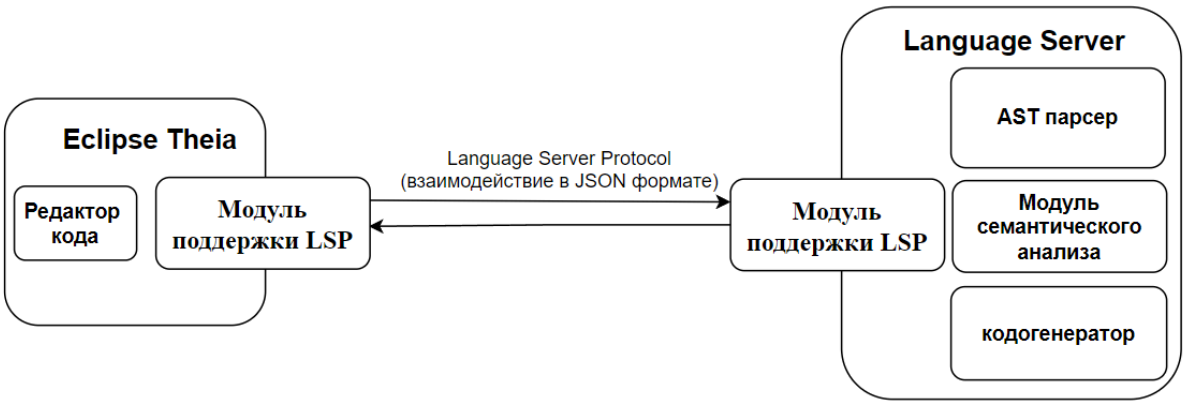

Рисунок 2 – Схема взаимодействия языкового сервера и редактора кода по LSP протоколу.

Среди основных компонентов Reflex IDE можно выделить следующие:

Редактор кода: этот модуль отвечает за взаимодействие с пользователем во время написания кода. Редактор может быть стандартным Eclipse-редактором или веб-редактором, который работает по протоколу LSP. Он предоставляет функционал автодополнения, подсветки синтаксиса, а также поддерживает обработку ошибок в реальном времени.

AST Парсер: парсер обрабатывает исходный код, создавая абстрактное синтаксическое дерево (AST). В случае ошибки парсер возвращает информацию в редактор, чтобы отобразить соответствующие предупреждения. Если ошибки отсутствуют, парсер передаёт AST в другие модули системы.

Модуль семантического анализа: этот компонент отвечает за валидацию кода, используя AST, чтобы убедиться, что программа соответствует правилам языка Reflex. Он проверяет семантику кода, чтобы гарантировать его корректность.

Кодогенератор: этот модуль генерирует код на основе AST. Он может создавать код на различных языках программирования или для разных платформ. В Reflex IDE используются кодогенераторы для языка C и платформы Arduino, чтобы поддерживать встраиваемые системы на базе микроконтроллеров ATmega.

Модуль поддержки LSP: этот модуль реализует поддержку Language Server Protocol, обеспечивая взаимодействие между редактором кода и языковым сервером. Это ключевой компонент, который позволяет Reflex IDE работать с веб-редакторами и обеспечивать удаленный доступ.

3.3 Структура модулей Reflex IDE

Проект Reflex IDE, разработанный с использованием Eclipse Xtext, состоит из нескольких модулей. Это позволяет легко добавлять новые компоненты или расширять существующие. Архитектура проекта включает несколько ключевых модулей, каждый из которых отвечает за определенные аспекты функциональности IDE. Рассмотрим основные модули, реализованной Reflex IDE:

ru.iaie.reflex – главный модуль, который содержит несколько критически важных компонентов для Reflex IDE. Именно здесь реализованы AST парсер, модуль семантического анализа и обобщенный кодогенератор.

ru.iaie.reflex.generators.r2c – модуль, который содержит кодогенераторы для языка C и платформы Arduino. Генераторы получают AST от парсера и преобразуют его в исходный код на языке C. Внутри находятся различные компоненты, отвечающие за генерацию файлов, функций, констант, портов и других элементы, необходимых для работы кода на целевой платформе.

ru.iaie.reflex.ide – модуль, который включает в себя компоненты поддержки Language Server Protocol (LSP). Он обеспечивает взаимодействие между языковым сервером Reflex IDE и веб-редактором кода. Поддержка LSP позволяет Reflex IDE интегрироваться с редакторами, такими как Eclipse Theia.

ru.iaie.reflex.ui – модуль, который предоставляет интерфейс пользователя (UI) для Reflex IDE. Он обеспечивает взаимодействие с пользователем через редактор кода в Eclipse IDE. Также здесь реализуется поддержка LSP, позволяющая Reflex IDE работать с веб-редакторами.

ru.iaie.reflex.scoping – модуль, который отвечает за область видимости в Reflex IDE. Он управляет тем, какие переменные и функции доступны в разных частях кода, обеспечивая корректную работу парсера и семантического анализа.

ru.iaie.reflex.typing – модуль, который отвечающий за управление типами данных в Reflex IDE. Он обеспечивает правильное определение типов переменных и констант, что важно для корректной работы семантического анализа и кодогенерации.

ru.iaie.reflex.utils – модуль, который содержит различные вспомогательные утилиты, используемые другими модулями Reflex IDE. Он может включать инструменты для работы с AST, генерации идентификаторов, обработки выражений и другие утилиты.

ru.iaie.reflex.validation – модуль, который отвечает за валидацию кода в Reflex IDE. Он проводит более глубокую проверку кода на предмет ошибок, которые могли быть пропущены на этапе семантического анализа.

ru.iaie.reflex.tests – модуль, который содержит тесты для Reflex IDE. Он обеспечивает автоматическое тестирование различных компонентов IDE, что важно для поддержания качества кода и предотвращения регрессий.

Таким образом, структура Reflex IDE основана на слабосвязанных модуль Eclipse, что обеспечивает гибкость и расширяемость. Eclipse Xtext автоматически реализует многие компоненты, такие как парсер и AST, в то время как другие компоненты, такие как кодогенератор и семантический анализ, необходимо реализовывать вручную. Такая структура позволяет Reflex IDE быть модульной, что облегчает интеграцию с другими фреймворками и редакторами кода.

3.4 Генерация кода в Reflex IDE

Генерация кода — ключевой этап в разработке на Reflex, предмете-ориентированном языке, применяемом для микроконтроллеров и встраиваемых систем. Процесс включает преобразование исходного кода на Reflex в язык C, который компилируется и загружается на целевую платформу.

3.4.1 Процесс преобразования Reflex в C

Генерация кода начинается с обработки исходного кода на языке Reflex. Здесь используется Xtext для создания абстрактного синтаксического дерева (AST), которое отражает структуру программы: процессы, состояния, переменные и прочие элементы. После этого начинается этап семантического анализа, который проверяет корректность кода, типы данных и логику программы.

Генерация кода на языке C происходит через модули, которые трансформируют AST в компилируемый код. Reflex IDE поддерживает несколько уровней кодогенерации, позволяя

эффективно реализовывать сложные переходы между состояниями, управлять таймерами внутри состояний и другие особенности Reflex.

Ключевые модули Reflex IDE, ответственные за генерацию кода, включают:

Парсер: Создает AST на основе исходного кода Reflex. Этот этап гарантирует, что синтаксические конструкции соответствуют правилам языка.

Семантический анализатор: Проверяет логические ошибки, неправильные типы данных, неинициализированные переменные, а также корректность использования процессов и состояний.

Кодогенератор: Преобразует AST в код на C, учитывая целевую платформу. Reflex IDE может использовать разные кодогенераторы для конкретных платформ.

Модуль кодогенерации: Обрабатывает AST, создавая код на C, который затем компилируется и загружается на целевую платформу. Здесь происходит структурирование сгенерированного кода, создание файлов и обеспечение совместимости с компиляторами

3.4.3 Гибкость модуля кодогенерации

Eclipse Xtext фреймворк поставляется вместе с удобным инструментом для гибкой кодогенерации – Eclipse Xtend.

Использование Xtend для генерации кода в Reflex IDE предоставляет значительные преимущества. Xtend позволяет писать компактный, легко читаемый код, используя шаблоны строк и расширения для эффективного внедрения текста C-кода в Xtend. Это делает кодогенерацию более гибкой и настраиваемой, позволяя при необходимости легко вносить изменения.

Например, с помощью Xtend имеется возможность напрямую работать с элементами AST дерева.

Resource – структура данных, которая полностью хранит все элементы нашей программы в виде AST дерева. в следующем примере наглядно показана работа с данной структурой на языке Xtend. Здесь происходит создание имен и итоговой структуры для генерируемых файлов, а также запускается следующие необходимые кодогенераторы.

```
new(Resource resource, IFileSystemAccess2 fsa) {\
        sourceFileName = resource.getURI().lastSegment.toString.replace(".rcs", "")
        rootDirName = sourceFileName
        program = resource.getProgram()
        this.fsa = fsa
        identifiersHelper = new ReflexIdentifiersHelper
        portGenerationHelper = new PortGenerationHelper(identifiersHelper)
        constGenerationHelper = new ConstantGenerationHelper(identifiersHelper)
        varGenerationHelper = new VariableGenerationHelper(identifiersHelper)
    }
```

Листинг 12 - конструктор для класса R2CFileGenerator.xtend, который отвечает за генерацию структуры и имен файлов на языке C.

3.4.2 Структура сгенерированных файлов на языке C

При разработке программы на языке C на основе исходного кода, написанного на языке Reflex, формируется несколько ключевых файлов. Каждый из них выполняет определенные функции, обеспечивая работу приложения. Ниже представлена структура и назначение основных сгенерированных файлов:

Главный файл программы: Этот файл включает в себя основную логику работы программы. Название файла совпадает с названием исходного файла на языке Reflex. Здесь происходит инициализация всех переменных, процессов и состояний, а также управление основным циклом выполнения. Файл служит связующим звеном для всех процессов и состояний, определенных в Reflex коде, реализуя их в контексте микроконтроллера.

CMakeLists.txt - файл для сборки проекта с использованием CMake, указывающий, как компилировать и связывать файлы программы

Файлы констант:

r_cnst.h: Определяет константы, используемые в программе, например, пороговые значения датчиков, таймауты и другие важные параметры, которые требуются для корректной работы программы.

r_main.h: Содержит глобальные константы и подключения пользовательских библиотек, необходимые для функционирования программы.

Визуально структура каталогов, где хранятся файлы, отображается следующим образом:

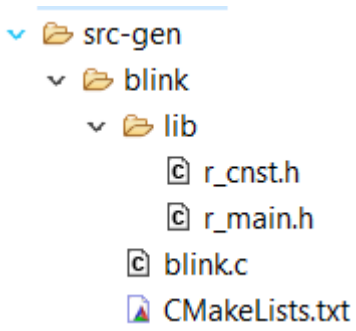

Рисунок 3 – Структура сгенерированных файлов на языке Си.

Такая организация файлов и каталогов способствует упорядоченной и систематизированной разработке, упрощает навигацию по проекту и поддержку кода.

3.5 Основные выводы по главе

В данной главе был обоснован выбор стека технологий для реализации ядра веб среды разработки. Описаны архитектура разрабатываемой системы, структура проекта, процесс генерации кода из Reflex в язык C.

Заключение

В рамках данной выпускной квалификационной работы было проведено объединение языков Reflex и IndustrialC с последующем развитием грамматики. Объединенный язык предназначен для применения во встраиваемых системах на базе микроконтроллеров ATmega. Данная версия языка представляет собой эффективное решение, открывающее новые возможности для расширения функциональности различных микроконтроллерных платформ. Особенностью разработанного инструмента является его гибкость и масштабируемость, что делает его применимым к широкому спектру задач автоматизации и управления.

В ходе работы были достигнуты следующие научные и практические результаты:

1) Проанализированы особенности и принципы процесс-ориентированного программирования на примере языков Reflex и Industrial-C, что позволило выявить сильные и слабые стороны данных языков, найти в них вектор дальнейшего развития.
2) Язык Reflex был объединен с IndustrialC и расширен следующими конструкциями: функции, процедуры обработки прерываний (ISR), критические секции кода, volatile переменные, циклы и массивы.
3) Разработаны и детализированы правила трансляции новых конструкций в язык C, что обеспечивает генерацию оптимизированного кода, адаптированного под специфику микроконтроллерных устройств.
4) Расширена функциональность среды разработки Reflex IDE за счет внедрения усовершенствованного модуля кодогенерации, что позволяет учитывать особенности новой версии языка при создании программного кода.
5) Оптимизирована генерация кода для существующих конструкций с целью повышения производительности системы и упрощения процесса разработки.
6) Внедрен новый модуль для выбора целевой платформы, интегрированный согласно стандартам Language Server Protocol, что расширяет возможности конфигурирования и адаптации разрабатываемых приложений под конкретные условия эксплуатации.

Проделанная работа позволила значительно расширить функциональные возможности языка Reflex и повысить уровень его применимости в реальных проектах микроконтроллерных систем. Результаты работы были представлены Международной научной студенческой конференции 2024, что подтверждает актуальность и значимость выполненного исследования. В дальнейшем планируется разработка поддержки дополнительных микроконтроллерных платформ, что позволит расширить сферу применения разработанного решения.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Стринадка Кирилл Александрович _____

ФИО студента *Подпись студента*

« ____ »_____2024 г.

(заполняется от руки)

Список литературы

1. Розов А. С., Зюбин В. Е. Адаптация процесс-ориентированного подхода к разработке встраиваемых микроконтроллерных систем // Автометрия. 2019. Том 55, № 2, С. 114–122. - 259 с. (DOI: 10.15372/AUT20190212)
2. Зюбин, В. Е. Язык «Рефлекс» – диалект Си для программируемых логических контроллеров // Шестая международная научно-практическая конференция «Средства и системы автоматизации» CSAF. – Т. 6. - 21 с.
3. Kumar R. H., Roopa A. U., Sathiya D. P. Arduino ATMEGA-328 microcontroller //Int. J. Innov. Res. Electr. Electron. Instrum. Control Eng. – 2015. – Т. 3. – №. 4. – С. 27-29. - 49 с.
4. Gamatié A. Designing embedded systems with the Signal programming language: synchronous, reactive specification. – Springer Science & Business Media, 2009. - 1200 с.
5. Масюк В.М., Кодубенко В.И., Симонова Л.С. Обзор и классификация современных микроконтроллеров в области мехатроники и робототехники // Материалы всерос. науч.-техн. конф. «Наукоемкие технологии в приборо- и машиностроении и развитие инновационной деятельности в вузе». – Калуга, 2016. – Т. 5. – С. 40–44. - 106 с.
6. Anureev I. et al. Towards safe cyber-physical systems: the Reflex language and its transformational semantics //2019 International Siberian Conference on Control and Communications (SIBCON). – IEEE, 2019. – С. 1-6. - 84 с.
7. Lee E. A., Seshia S. A. Introduction to embedded systems: A cyber-physical systems approach. – MIT press, 2016. - 278 с.
8. Zyubin V. Using process-oriented programming in LabVIEW //Proceedings of the Second IASTED International Multi-Conference on" Automation, Control, and Information technology ": Control, Diagnostics, and Automation. Novosibirsk. – 2010. – С. 35-41. - 183 с.
9. Villamor M. M. A review on process-oriented approaches for analyzing novice solutions to programming problems //Research and Practice in Technology Enhanced Learning. – 2020. – Т. 15. – №. 1. – С. 8. - 42 с.
10. Golden P., Dedieu H., Jacobsen K. S. (ed.). Fundamentals of DSL technology. – CRC Press, 2005. - 539 с.
11. Mernik M., Heering J., Sloane A. M. When and how to develop domain-specific languages //ACM computing surveys (CSUR). – 2005. – Т. 37. – №. 4. – С. 316-344. - 486 с.
12. Kunikowski W. et al. An overview of ATmega AVR microcontrollers used in scientific research and industrial applications //Pomiary Automatyka Robotyka. – 2015. – Т. 19. – №. 1. – С. 15-19. - 52 с.

13. Ali M. M., Islam R., Rahman A. Low Cost Temperature and Humidity Estimator with Atmega8 Microcontroller //International Journal of Trend in Scientific Research and Development (IJTSRD). – 2021. – Т. 5. – С. 243-250. - 384 с.
14. Cooling J. E. Languages for the programming of real-time embedded systems a survey and comparison //Microprocessors and Microsystems. – 1996. – Т. 20. – №. 2. – С. 67-77. - 275 с.
15. Myklebust G. The AVR microcontroller and C compiler co-design //ATMEL Development Center, Trondheim, Norway. – 1996. - 59 с.
16. Vostrukhin A. et al. Studying digital signal processing on arduino based platform //Proceedings of the 15th International Scientific Conference on Engineering for Rural Development, Jelgava, Latvia. – 2016. – С. 25-27. - 94 с.
17. Alizadehsani Z. et al. Modern integrated development environment (ides) //Sustainable Smart Cities and Territories. – Springer International Publishing, 2022. – С. 274-288. - 489 с.
18. Goldman M., Little G., Miller R. C. Real-time collaborative coding in a web IDE //Proceedings of the 24th annual ACM symposium on User interface software and technology. – 2011. – С. 155-164. - 752 с.
19. Rodriguez-Echeverria R. et al. Towards a language server protocol infrastructure for graphical modeling //Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. – 2018. – С. 370-380. - 583 с.
20. Saini R., Bali S., Mussbacher G. Towards web collaborative modelling for the user requirements notation using eclipse che and theia ide //2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE). – IEEE, 2019. – С. 15-18. - 316 с.
21. Минакова О.В., Акамсина Н.В., Курипта О.В. РАЗРАБОТКА ПРОГРАММНЫХ ИНСТРУМЕНТОВ НА БАЗЕ РАСШИРЯЕМЫХ ПЛАТФОРМ С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ // Вестник ВГТУ. 2022. №4. URL: https://cyberleninka.ru/article/n/razrabotka-programmnyh-instrumentov-na-baze-rasshiryaemyh-platform-s-otkrytym-ishodnym-kodom (дата обращения: 04.05.2024). - 63 с.
22. Sumangali K., Borra L., Mishra A. S. A Comprehensive review on the open source hackable text editor-ATOM //IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2017. – Т. 263. – №. 4. – С. 042061. - 131 с.
23. Bucchiarone A. et al. (ed.). Domain-specific languages in practice: with JetBrains MPS. – Springer Nature, 2021. - 591 с.

24. Parr T. J., Quong R. W. ANTLR: A predicated‐LL (k) parser generator //Software: Practice and Experience. – 1995. – Т. 25. – №. 7. – С. 789-810. - 1200 с.
25. Eysholdt M., Behrens H. Xtext: implement your language faster than the quick and dirty way //Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. – 2010. – С. 307-309. - 628 с.
26. Rask J. K. et al. The specification language server protocol: A proposal for standardised LSP extensions //arXiv preprint arXiv:2108.02961. – 2021. - 338 с.

Приложение А

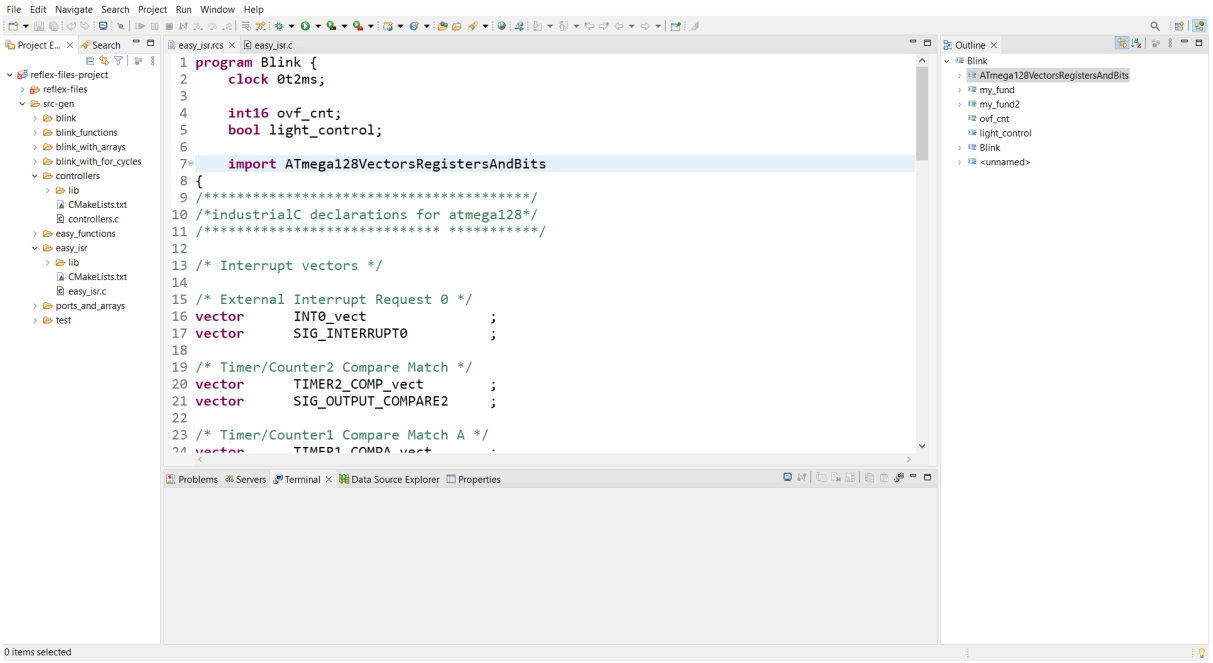

Рисунок 4 – Интерфейс среды разработки для языка Reflex.

Приложение Б

IDE для объединенного языка Reflex

Руководство Оператора

Листов 6

Новосибирск, 2024

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном программном документе приведено руководство оператора по применению и эксплуатации программной системы, представленной в виде web IDE процесс-ориентированного языка Reflex.

В разделе «Назначение программы» указаны сведения о назначении программы и перечислены ее функции. В разделе «Условия выполнения программы» перечислены условия, являющиеся необходимыми для выполнения программы. Раздел «Выполнение программы» содержит последовательность действий оператора, которые необходимы для загрузки, запуска, выполнения и завершения программы. В разделе «Сообщения оператору» приведено описание текстовых сообщений и описаны действия оператора при их возникновении.

Оформление программного документа «Руководство оператора» произведено по требованиям ГОСТ 19.505-79 «ЕСПД. Руководство оператора» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

1. Назначение программы

Программная система предназначена для использования процесс-ориентированного языка программирования poST с помощью web технологий, представленных в виде web IDE.

Функции:

- Отображение файловой системы;
- Создание / удаление файлов; Редактирование файлов;
- Подсветка синтаксиса;
- Выделение ошибок и предупреждений;
- Автодополнение;
- Навигация по коду;
- Поиск по коду;
- Синтаксическая и семантическая проверка;
- Список ошибок и предупреждений, с возможностью перехода в проблемный участок кода;
- Отображение структуры открытого файла;
- Генерация файлов Си кода.

2. Условия выполнения программы

2.1. Минимальный состав аппаратных средств

Для работы программного средства требуется персональный компьютер со следующими техническими требованиями:

- Процессор, совместимый с архитектурой x86, и тактовой частотой 2 ГГц или выше;
- Оперативную память объемом 4 Гб или выше; Жесткий диск объемом 250 Гб или выше;
- Лицензионная операционная система, unix-подобная или windows;

2.2. Минимальный состав программных средств

Для работы программного средства требуется наличие на персональном компьютере установленной виртуальной машины Java, интегрированной среды разработки Eclipse IDE с установленным плагином Xtext.

2.3. Требования к оператору

Конечный оператор (администратор) программы должен обладать практическими навыками работы с серверными терминалами, интегрированными средами разработки, уметь настраивать сетевые соединения. Для понимания работы системы, администратор должен обладать навыками мониторинга процессов в операционной системе.

3. Выполнение программы

3.1. Загрузка и запуск программы

Перед запуском программного средства необходимо скачать исходные коды Reflex IDE и импортировать их в Eclipse IDE.

В интерфейсе Eclipse IDE в пакете «ru.iaie.reflex» нажать правой кнопкой мыши на файл «Reflex.xtext», выбрать пункт «Generate Xtext Artifacts».

В интерфейсе Eclipse IDE нажать на пакет «ru.iaie.reflex» правой кнопкой мыши, выбрать пункт «Run As» далее выбрать Eclipse Application. Этими действиями будет произведен запуск Reflex IDE. Запуск IDE производится с помощью пакетного фреймворка Eclipse Xtext.

3.2. Выполнение программы

В целевой Reflex IDE необходимо создать файл с расширением **.rcs**, в этом файле производится написание программы на языке Reflex.

После сохранения исходного файла с расширением .rcs будут сгенерированы файлы на языке Си в директории проекта в папке «src-gen».

После запуска на экране колонки отобразится главное меню графического интерфейса программы. Можно взаимодействовать с ним напрямую, а также можно дальше использовать голосовые команды.

3.3. Завершение работы программы

Завершить работу приложения можно либо нажав на физическую кнопку завершения Reflex IDE.

4. Сообщения оператору

В ходе работы программной системы оператор получает информационные сообщения работы IDE в терминале Eclipse IDE.

При получения информационного сообщения с ошибкой времени исполнения Reflex IDE, оператор должен сообщить об ошибке разработчикам системы и принять действия для устранения нарушения путем перезапуска системы.

5. Лист регистрации изменений

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа	Подпись	Дата
Номер изм.	измененных	замененных	новых	аннулированных					

Таблица 4 – Лист регистрации изменений в программном документе «Руководство оператора»