МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка ПО для преобразования файла с описанием регистров микроконтроллера в формате pdf в заголовочный файл на языке C++»

Студент И.В. Слезкин _____
(И.О. Фамилия) (личная подпись)

Руководитель С.В. Митин
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент, А.В. Егорова

Тольятти 2023

Аннотация

Тема: «Разработка ПО для преобразования файла с описанием регистров микроконтроллера в формате pdf в заголовочный файл на языке C++».

Целью данной выпускной квалификационной работы является разработка программного обеспечения для преобразования файла с описанием регистров микроконтроллера в формате pdf в заголовочный файл на языке C++.

Объектом исследования являются способы выделения из текста заданных образцов и формирование из них заданной структуры текстового файла.

Предметом исследования являются поиск и выбор программного решения для выбранного способа.

В первом разделе работы был проведен анализ предметной области, представлен краткий обзор микроконтроллеров и их применения в различных областях. В нем была представлена документация, прилагаемая к микроконтроллерам, и приведены примеры из технических паспортов. Также обсуждалась важность заголовочных файлов в программировании и их преимущества.

Во втором разделе были рассмотрены существующие методы преобразования pdf-файлов в формат заголовочных файлов CPP, а также варианты поиска и замены фрагментов текста с помощью регулярных выражений. Были рассмотрены различные методы преобразования, принципы их работы, выделены преимущества и недостатки каждого подхода. В заключение анализа была подчеркнута необходимость создания нового программного обеспечения и изложены требования к нему.

В третьем разделе был разработан алгоритм решения поставленной задачи с использованием языка программирования Python. Кроме того, был создан графический интерфейс пользователя (GUI) для повышения удобства использования конвертера.

Результатом выполнения бакалаврской работы является разработанное программное обеспечение для преобразования файла с описанием регистров микроконтроллера в формате pdf в заголовочный файл на языке C++.

Работа включает 52 страницы с приложениями, 39 рисунков и 26 источников.

Abstract

The title of the bachelor’s thesis is "Development of software to convert a pdf file describing the registers of a microcontroller into a header file in C++".

The aim of this graduate qualification work is to develop software to convert a file describing microcontroller registers in pdf format into a header file in C++.

The object of the research is the methods of selecting given samples from text and forming a given text file structure from them.

The subject of the study is the search and selection of a software solution for the selected method of implementing the task.

In the first section of the paper an analysis of the subject area was made, a brief overview of microcontrollers and their applications in various fields was presented. It presented the documentation provided with microcontrollers and gave examples from datasheets. The importance of header files in programming and their advantages were also discussed.

The second section dealt with existing methods of converting pdf files into the CPP header file format, as well as options for finding and replacing text fragments using regular expressions. Various conversion methods were examined, the principles of their operation, and the advantages and disadvantages of each approach were highlighted. At the end of the analysis, the need for new software was stressed and the requirements for it were outlined.

In the third section an algorithm for solving the task was developed using the Python programming language. In addition, a graphical user interface (GUI) was created to enhance the usability of the converter.

The result of the bachelor's work is the developed software to convert file with the description of registers of microcontroller in pdf format into header file in C++ language.

The bachelor’s thesis consists of an explanatory note on 52 pages, 39 pictures and 26 sources.

Содержание

Введение

С каждым годом растет использование микроконтроллеров в различных приложениях и устройствах. В бытовой технике микроконтроллеры используются для управления работой и для обеспечения дополнительных функций, таких как таймеры, контроль температуры и обнаружение неисправностей. Примерами могут служить стиральные машины, холодильники, кондиционеры и микроволновые печи. В промышленных системах управления микроконтроллеры используются для автоматизации управления машинами и процессами. Они используются для мониторинга датчиков, управления исполнительными механизмами и выполнения алгоритмов для оптимизации работы системы. Примерами могут служить робототехника, производственное оборудование и транспортные системы. Количество микроконтроллеров, используемых в устройстве, определяется сложностью системы, количеством задач, которые должны быть выполнены, и уровнем производительности, необходимым для удовлетворения требований приложения, и может варьироваться от одного до десятка. Основная актуальность их использования заключается в способности обеспечивать точное и аккуратное управление широким спектром устройств и систем.

Документация, предоставляемая производителем микроконтроллера, чаще всего, имеет формат PDF.

PDF (Portable Document Format) – это формат файлов, который стал неотъемлемой частью цифрового мира. Его актуальность заключается в нескольких ключевых особенностях, которые делают его полезным и универсальным форматом для широкого круга приложений. Причинами актуальности формата PDF являются его универсальность, сохранение форматирования, безопасность, интерактивность, малый размер файла простота создания.

Контроллеров и их использование повсеместно, актуальной становиться задача быстро перейти с одного контроллера на другой. Кроме

того, даже в пределах одного семейства могут быть различия в распределении адресного пространства и битов в регистрах микроконтроллера. Вручную описывать эти различия – кропотливая и трудоемкая задача, а точное определение регистров очень важно. Автоматизирование этого процесса позволит снизить риск ошибок из-за человеческого фактора и сэкономить значительное количество времени, которое было бы потрачено на определение регистров вручную.

Новизна данного подхода заключается в использовании PDF-файлов как основы для автоматического создания заголовочных файлов. Существуют разные инструменты и методы создания заголовочных файлов, такие как использование библиотек или шаблонов кода, но они ограничиваются применением лишь при работе с конкретным устройством или его моделью. Также большинство алгоритмов реализуют перевод текста без его структуризации, что является недопустимым в случае работы с микроконтроллерами.

В данной работе исследуются способы поиска и выбора программного решения для выбранного способа реализации задачи. Объектом исследования являются способы выделения из текста заданных образцов и формирование из них заданной структуры текстового файла.

Цель работы – разработка программного обеспечения для преобразования файла с описанием регистров микроконтроллера в формате pdf в заголовочный файл на языке C++.

При этом, необходимо решить следующие задачи:

- изучить структуру документации, сопровождающей микроконтроллеры;
- познакомиться с существующими методами автоматизации процесса преобразования pdf-файла в заголовочный;
- предложить наиболее целесообразного способа достижения поставленной цели;
- выбрать средства разработки;

- разработать эффективный программный код, осуществляющий преобразование файлов;
- разработать пользовательский интерфейс.

На языке программирования Python разработан пользовательский интерфейс и программный модуль, который демонстрирует преобразование файла формата pdf в заголовочный файл на языке C++. Работа программы была протестирована на различных pdf-файлах, содержащих информацию о регистрах микроконтроллеров.

1 Анализ предметной области

1.1 Области применения микроконтроллеров

Микроконтроллер – это термин, используемый для описания системы, которая включает в себя минимум микропроцессор, память программ, память данных и ввод-вывод (I/O). Некоторые микроконтроллерные системы также включают таймеры, счетчики, аналого-цифровые (A/D) преобразователи и так далее. По сути, это одночиповый компьютер, который используется или встраивается в другие устройства или оборудование для выполнения функций управления и также называется встроенным контроллером. Работа микроконтроллера объясняется с помощью блок-схем, чтобы читатель мог полностью понять функции ввода/вывода.

Микроконтроллеры широко используются для полуавтоматического управления, которое не требует такого сложного контроллера, как микропроцессор. Их малые размеры и низкие требования к энергопотреблению делают их идеальными для портативного оборудования, работающего от батарей. Устройства в доме, которые могут управляться микроконтроллером, включают посудомоечные и стиральные машины, электрические обогреватели, мобильные телефоны, пульты дистанционного управления телевизорами, системы безопасности и компьютерные принтеры. Каждый день разрабатываются и поступают на рынок так называемые «умные» версии существующих бытовых приборов. Автомобильная электроника – еще одна быстро развивающаяся область, в которой микроконтроллеры играют важную роль. Несколько микроконтроллеров могут отвечать за отдельные системы, такие как антиблокировочная тормозная система, впрыск топлива, контроль тяги или управление подвеской. Эти микроконтроллеры взаимодействуют друг с другом, чтобы гарантировать плавную и безопасную работу автомобиля. Они отправляют и получают данные через свои периферийные устройства ввода-вывода и

обрабатывают их для выполнения поставленных задач, демонстрируя всю мощь микроконтроллеров в современных технологиях.

Еще один интересный пример растущего использования микроконтроллеров можно найти в системе управления самолетов A320 и A330 Airbus. В этом самолете главный компьютер управления находится на летной палубе под управлением пилота. Но вычисления, связанные с управлением самолетом, не ограничиваются главным компьютером. Каждый из приводов, которые приводят в движение летные поверхности (элероны, щели, спойлеры, элеваторы и рули), имеет свой собственный микроконтроллер. Когда нужно перевести какую-либо поверхность полета из одного положения в другое, сигнал посылается с главного компьютера на соответствующий микроконтроллер в крыльях или хвостовом оперении. Сигнал простой, эквивалентный «переместить поверхность из угла A в угол B». Получив этот сигнал, микроконтроллер управляет током, подаваемым на двигатель привода поверхности полета. Датчики сообщают микроконтроллеру фактическое положение поверхности, и ток, подаваемый на двигатель, регулируется соответствующим образом, мгновение за мгновением. Микроконтроллер время от времени сообщает главному компьютеру о положении поверхности и, наконец, сообщает об успешном завершении действия. Эта система известна как распределенная обработка, в отличие от централизованной обработки одним центральным компьютером. Распределенная обработка уменьшает количество проводов, необходимых в самолете, и снижает нагрузку на главный компьютер, позволяя ему сосредоточиться на других, возможно, более важных задачах.

1.2 Документация сопровождающая микроконтроллеры

Микроконтроллеры являются сложными продуктами для документирования. С одной стороны, поставщик должен описать функциональность вычислительного ядра устройства, функциональность и возможные настройки периферийных устройств, а также электрические и

упаковочные характеристики устройства. Это описание должно быть сосредоточено на исходной функциональности схем устройства и формируемых ими регистрах.

С другой стороны, необработанные подробности о функциональности и настройке сами по себе не могут проинформировать разработчика о том, как использовать данное периферийное устройство, процессор или другую функцию в реальном приложении. Кроме того, этот продукт может иметь стандартизированный программный уровень, с которым он обычно используется, и общую схему, внешнюю по отношению к контроллеру, для реализации требуемой сигнализации.[22]

Еще одна проблема документирования микроконтроллеров связана с их бесконечной гибкостью, поскольку они являются программируемыми продуктами, и тем фактом, что программное обеспечение может быть написано на множестве различных языков программирования с использованием широкого спектра программных инструментов от различных производителей, из которых некоторые, а возможно и ни одного, не поставляются самим производителем.

Наконец, существует проблема программ и аппаратных средств, используемых для создания и отладки программного обеспечения микроконтроллера, а также средств программирования, используемых для малого или массового производства.

В результате, только технические паспорта обычно содержат около сотни страниц, часто до нескольких сотен.

Хотя техническое описание микроконтроллера занимает очень много страниц, в нем хранится далеко не вся информация, которая может потребоваться разработчику. В дополнение также придется обратиться к документации на инструменты, превращающие исходный код в микропрограмму, инструменты для разработки кода и отладки, а также инструменты для программирования.[24]

Для наглядности будет рассмотрен пример документации микроконтроллера: PIC16F18877. Пример первых страниц технического паспорта представлен на рисунке 1.1.

Документация, прилагающееся к простому 8-разрядному микроконтроллеру, такому как Microchip Technology PIC16F18877, включает в себя технический паспорт, список исправлений, некоторые вспомогательные материалы, охватывающие специфические элементы и оценочные платы, спецификации программирования и длинный список примечаний по применению. Также прикладывается исходный код, сопровождающий некоторые заметки по применению, несколько брошюр по продажам и технический документ по аналого-цифровым преобразователям (АЦП).[22]

В техническом паспорте описана подробная информация об одном или нескольких микроконтроллерах (часто из одной кремниевой матрицы создается несколько различных микроконтроллеров с различным количеством выводов и корпусов). Вместо того чтобы создавать и поддерживать документацию для каждого варианта, скорее всего, несколько устройств будут описаны в одном техническом паспорте. В данном случае, рассматриваются два варианта: PIC16F18857 и PIC16F18877.[22]

Для выполнения данной выпускной квалификационной работы необходимо будет обратиться к страницам, на которых приводится описание регистров.

В техническом паспорте раздел информации о регистрах обычно содержит подробное описание каждого из регистров микроконтроллера, включая их назначение, функциональность и то, как они используются в работе микроконтроллера. Эта информация может иметь решающее значение для понимания того, как работает микроконтроллер, и для разработки программного обеспечения, которое взаимодействует с ним.

Раздел регистров технического паспорта может содержать информацию о различных типах регистров, доступных в микроконтроллере,

таких как регистры общего назначения, регистры специальных функций и регистры управления. Он также может включать информацию о том, как осуществляется доступ к регистрам и управление ими, включая инструкции и команды, которые используются для чтения из определенных регистров или записи в них.[11]

Тем самым можно сделать вывод, что всю эту информацию будет необходимо корректно извлечь при преобразовании формата файла в итоговый заголовочный файл.

Существует несколько методов описания регистров микроконтроллера в технических документах. Вот несколько распространенных методов:

Карта регистров – это таблица или диаграмма, которая обеспечивает визуальное представление регистров микроконтроллера, включая их адреса, имена и позиции битов. Этот метод особенно полезен для быстрого определения местоположения и функций конкретных регистров.[22]

Другим распространенным методом описания регистров микроконтроллера является предоставление подробного описания их назначения и функциональности. Этот метод может быть использован для объяснения того, как каждый регистр используется в работе микроконтроллера, и может быть особенно полезен разработчикам, которые пишут программное обеспечение, взаимодействующее с регистрами микроконтроллера.[24]

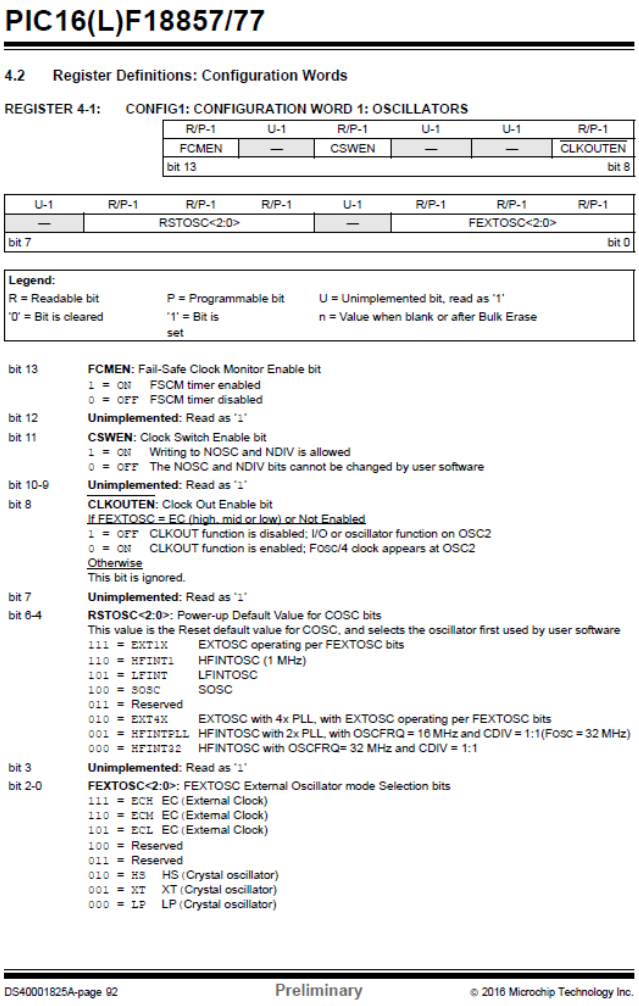
Еще одним важным аспектом регистров микроконтроллера является то, как осуществляется доступ к ним и управление ими. Некоторые технические документы могут содержать подробную информацию об инструкциях или командах, которые используются для чтения из определенных регистров или записи в них, или о последовательности событий, которые должны произойти, чтобы получить доступ к определенному регистру.

В некоторых случаях временные диаграммы могут использоваться для описания того, как осуществляется доступ к регистрам и управление ими с течением времени. Эти диаграммы могут быть особенно полезны для

понимания точных требований к синхронизации конкретного микроконтроллера или системы.

Наконец, некоторые технические документы могут содержать примеры или фрагменты кода, иллюстрирующие, как конкретные регистры используются на практике. Этот метод может быть особенно полезен разработчикам, которые новички в конкретном микроконтроллере или системе, и может помочь прояснить, как регистры используются в реальных приложениях.[11]

На рисунке 1.2 приведен пример страницы с описанием одного из регистров микроконтроллера PIC16F18857/77.



PIC16(L)F18857/77

4.2 Register Definitions: Configuration Words

REGISTER 4-1: CONFIG1: CONFIGURATION WORD 1: OSCILLATORS

R/P-1	U-1	R/P-1	U-1	U-1	R/P-1
FCMEN	—	CSWEN	—	—	CLKOUTEN
bit 13					bit 8

U-1	R/P-1	R/P-1	R/P-1	U-1	R/P-1	R/P-1	R/P-1
—	RSTOSC<2:0>			—	FEXTOSC<2:0>		
bit 7							bit 0

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '1'
'0' = Bit is cleared	'1' = Bit is set	n = Value when blank or after Bulk Erase

bit 13 **FCMEN:** Fail-Safe Clock Monitor Enable bit
1 = ON FSCM timer enabled
0 = OFF FSCM timer disabled

bit 12 **Unimplemented:** Read as '1'

bit 11 **CSWEN:** Clock Switch Enable bit
1 = ON Writing to NOSC and NDIV is allowed
0 = OFF The NOSC and NDIV bits cannot be changed by user software

bit 10-9 **Unimplemented:** Read as '1'

bit 8 **CLKOUTEN:** Clock Out Enable bit
If FEXTOSC = EC (high, mid or low) or Not Enabled
1 = OFF CLKOUT function is disabled; I/O or oscillator function on OSC2
0 = ON CLKOUT function is enabled; FOSC/4 clock appears at OSC2
Otherwise
This bit is ignored.

bit 7 **Unimplemented:** Read as '1'

bit 6-4 **RSTOSC<2:0>:** Power-up Default Value for COSC bits
This value is the Reset default value for COSC, and selects the oscillator first used by user software
111 = EXT1X EXTOSC operating per FEXTOSC bits
110 = HFINT1 HFINTOSC (1 MHz)
101 = LFINT LFINTOSC
100 = SOSC SOSC
011 = Reserved
010 = EXT4X EXTOSC with 4x PLL, with EXTOSC operating per FEXTOSC bits
001 = HFINTPLL HFINTOSC with 2x PLL, with OSCFRQ = 16 MHz and CDIV = 1:1(FOSC = 32 MHz)
000 = HFINT32 HFINTOSC with OSCFRQ= 32 MHz and CDIV = 1:1

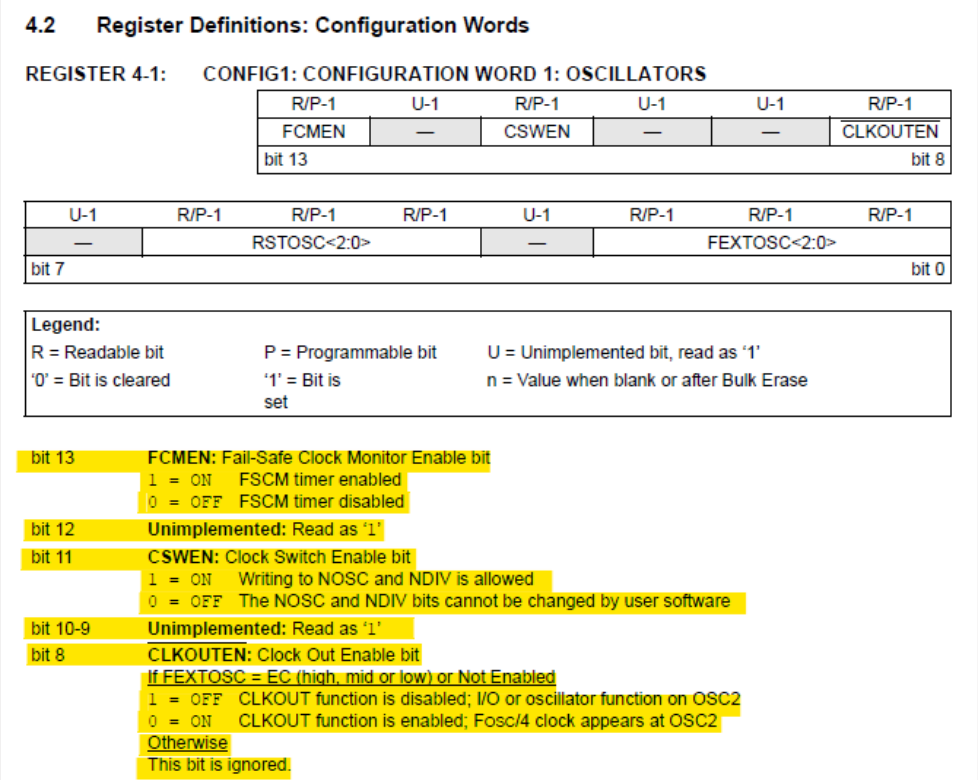
bit 3 **Unimplemented:** Read as '1'

bit 2-0 **FEXTOSC<2:0>:** FEXTOSC External Oscillator mode Selection bits
111 = ECH EC (External Clock)
110 = ECM EC (External Clock)
101 = ECL EC (External Clock)
100 = Reserved
011 = Reserved
010 = HS HS (Crystal oscillator)
001 = XT XT (Crystal oscillator)
000 = LP LP (Crystal oscillator)

DS40001825A-page 92 Preliminary © 2016 Microchip Technology Inc.

Рисунок 1.1 – Пример страницы технического паспорта с описанием регистра

На рисунке 1.2 выделен фрагмент описания регистров, который в дальнейшем необходимо будет анализировать и обрабатывать для составления заголовочного файла.



4.2 Register Definitions: Configuration Words

REGISTER 4-1: CONFIG1: CONFIGURATION WORD 1: OSCILLATORS

R/P-1	U-1	R/P-1	U-1	U-1	R/P-1
FCMEN	—	CSWEN	—	—	CLKOUTEN
bit 13					bit 8

U-1	R/P-1	R/P-1	R/P-1	U-1	R/P-1	R/P-1	R/P-1
—	RSTOSC<2:0>			—	FEXTOSC<2:0>		
bit 7							bit 0

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '1'
'0' = Bit is cleared	'1' = Bit is set	n = Value when blank or after Bulk Erase

bit 13 **FCMEN:** Fail-Safe Clock Monitor Enable bit
1 = ON FSCM timer enabled
0 = OFF FSCM timer disabled

bit 12 **Unimplemented:** Read as '1'

bit 11 **CSWEN:** Clock Switch Enable bit
1 = ON Writing to NOSC and NDIV is allowed
0 = OFF The NOSC and NDIV bits cannot be changed by user software

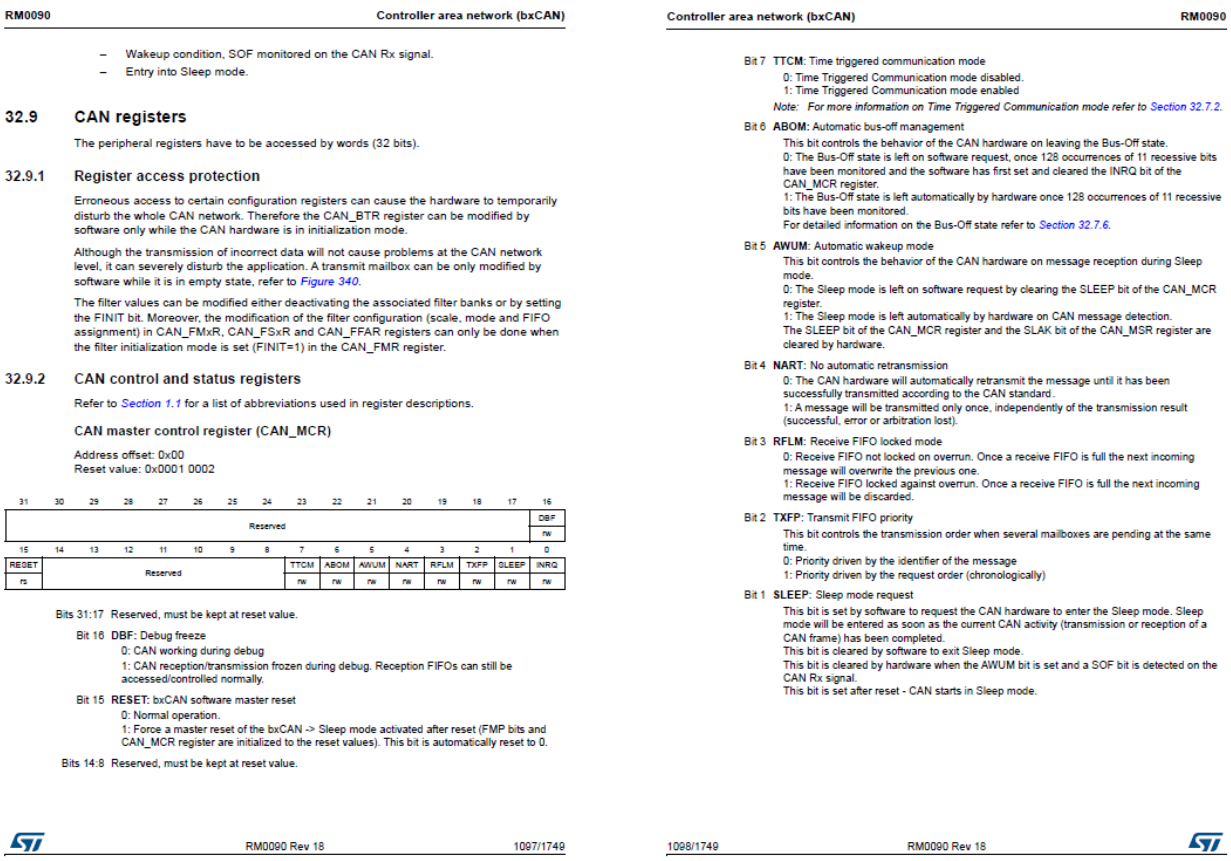
bit 10-9 **Unimplemented:** Read as '1'

bit 8 **CLKOUTEN:** Clock Out Enable bit
If FEXTOSC = EC (high, mid or low) or Not Enabled
1 = OFF CLKOUT function is disabled; I/O or oscillator function on OSC2
0 = ON CLKOUT function is enabled; Fosc/4 clock appears at OSC2
Otherwise
This bit is ignored.

Рисунок 1.2 – Фрагмент описания регистров

Также для сравнения будет приведен пример страниц из технического паспорта семейства микроконтроллеров STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx и STM32F43xxx с различными объемами памяти, пакетами и периферийными устройствами.[22]

Пример страниц представлен на рисунке 1.3, а на рисунке 1.4 также, как и в прошлом примере выделен фрагмент описания регистров.



RM0090 Controller area network (bxCAN)

– Wakeup condition, SOF monitored on the CAN Rx signal.
– Entry into Sleep mode.

32.9 CAN registers

The peripheral registers have to be accessed by words (32 bits).

32.9.1 Register access protection

Erroneous access to certain configuration registers can cause the hardware to temporarily disturb the whole CAN network. Therefore the CAN_BTR register can be modified by software only while the CAN hardware is in initialization mode.

Although the transmission of incorrect data will not cause problems at the CAN network level, it can severely disturb the application. A transmit mailbox can be only modified by software while it is in empty state, refer to *Figure 340*.

The filter values can be modified either deactivating the associated filter banks or by setting the FINIT bit. Moreover, the modification of the filter configuration (scale, mode and FIFO assignment) in CAN_FMxR, CAN_FSxR and CAN_FFAR registers can only be done when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

32.9.2 CAN control and status registers

Refer to *Section 1.1* for a list of abbreviations used in register descriptions.

CAN master control register (CAN_MCR)

Address offset: 0x00
Reset value: 0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															DBF
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	Reserved							TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
rs								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **DBF:** Debug freeze
0: CAN working during debug
1: CAN reception/transmission frozen during debug. Reception FIFOs can still be accessed/controlled normally.

Bit 15 **RESET:** bxCAN software master reset
0: Normal operation.
1: Force a master reset of the bxCAN -> Sleep mode activated after reset (FMP bits and CAN_MCR register are initialized to the reset values). This bit is automatically reset to 0.

Bits 14:8 Reserved, must be kept at reset value.

RM0090 Rev 18 1097/1749

Controller area network (bxCAN) RM0090

Bit 7 **TTCM:** Time triggered communication mode
0: Time Triggered Communication mode disabled.
1: Time Triggered Communication mode enabled
Note: For more information on Time Triggered Communication mode refer to Section 32.7.2.

Bit 6 **ABOM:** Automatic bus-off management
This bit controls the behavior of the CAN hardware on leaving the Bus-Off state.
0: The Bus-Off state is left on software request, once 128 occurrences of 11 recessive bits have been monitored and the software has first set and cleared the INRQ bit of the CAN_MCR register.
1: The Bus-Off state is left automatically by hardware once 128 occurrences of 11 recessive bits have been monitored.
For detailed information on the Bus-Off state refer to *Section 32.7.6*.

Bit 5 **AWUM:** Automatic wakeup mode
This bit controls the behavior of the CAN hardware on message reception during Sleep mode.
0: The Sleep mode is left on software request by clearing the SLEEP bit of the CAN_MCR register.
1: The Sleep mode is left automatically by hardware on CAN message detection.
The SLEEP bit of the CAN_MCR register and the SLAK bit of the CAN_MSR register are cleared by hardware.

Bit 4 **NART:** No automatic retransmission
0: The CAN hardware will automatically retransmit the message until it has been successfully transmitted according to the CAN standard.
1: A message will be transmitted only once, independently of the transmission result (successful, error or arbitration lost).

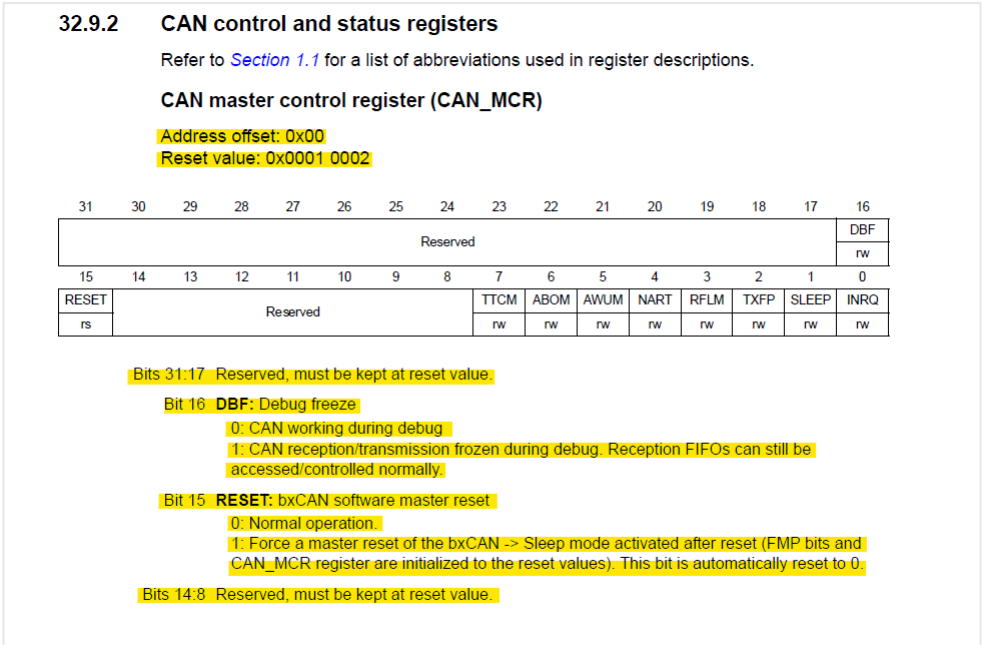
Bit 3 **RFLM:** Receive FIFO locked mode
0: Receive FIFO not locked on overrun. Once a receive FIFO is full the next incoming message will overwrite the previous one.
1: Receive FIFO locked against overrun. Once a receive FIFO is full the next incoming message will be discarded.

Bit 2 **TXFP:** Transmit FIFO priority
This bit controls the transmission order when several mailboxes are pending at the same time.
0: Priority driven by the identifier of the message
1: Priority driven by the request order (chronologically)

Bit 1 **SLEEP:** Sleep mode request
This bit is set by software to request the CAN hardware to enter the Sleep mode. Sleep mode will be entered as soon as the current CAN activity (transmission or reception of a CAN frame) has been completed.
This bit is cleared by software to exit Sleep mode.
This bit is cleared by hardware when the AWUM bit is set and a SOF bit is detected on the CAN Rx signal.
This bit is set after reset - CAN starts in Sleep mode.

1098/1749 RM0090 Rev 18

Рисунок 1.3 – Пример страниц технического паспорта с описанием регистра



32.9.2 CAN control and status registers

Refer to *Section 1.1* for a list of abbreviations used in register descriptions.

CAN master control register (CAN_MCR)

Address offset: 0x00
Reset value: 0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															DBF
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	Reserved							TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
rs								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **DBF:** Debug freeze
0: CAN working during debug
1: CAN reception/transmission frozen during debug. Reception FIFOs can still be accessed/controlled normally.

Bit 15 **RESET:** bxCAN software master reset
0: Normal operation.
1: Force a master reset of the bxCAN -> Sleep mode activated after reset (FMP bits and CAN_MCR register are initialized to the reset values). This bit is automatically reset to 0.

Bits 14:8 Reserved, must be kept at reset value.

Рисунок 1.4 – Фрагмент описания регистров

Хоть и можно увидеть схожесть в двух примерах, однако нельзя не заметить и большую разницу в способе описания, а также служебных словах.

Все это необходимо учитывать при разработке программного обеспечения для преобразования, ведь по итогу должен быть получен заголовочный файл в котором четко описана структура регистра – его биты.

1.3 Использование заголовочных файлов в программировании

Чтобы сократить объем текста в листингах программ и уменьшить количество повторений, создаются и используются серии заголовочных файлов. Заголовочные файлы используются, когда программы используют одну и ту же серию инструкций совершенно одинаковым образом во всех проектах и программах.

Заголовочные файлы – это файлы, в которых объявления функций и определения макросов языка с совместно используются различными исходными файлами. Использование заголовочных файлов позволяет экономить время, так как программа становится короткой и читабельной.[25]

Заголовочные файлы служат двум целям: во-первых, системные заголовочные файлы декларируют интерфейс к частям операционной системы. Они помогают в предоставлении определений и объявлений, необходимых для вызова системных вызовов и библиотек. Во-вторых, заголовочные файлы содержат декларацию интерфейсов между исходными файлами программы.

В заголовочных файлах объявляются различные типы данных (например, типовые определения и прототипы публичных функций) и макросы (например, адреса регистров микроконтроллеров).

Первоначально для программирования микропроцессоров использовался язык ассемблера, но ассемблер трудно понять, и отладка на языке ассемблера также затруднена. Так как архитектура процессора меняется от архитектуры к архитектуре, инструкции ассемблера меняются, поэтому трудно сделать программу переносимой на языке ассемблера.

Для преодоления этих трудностей был использован язык Си. Язык Си прост для понимания и отладки, также коды могут быть сделаны переносимыми с платформы на платформу с помощью языка Си.

Поэтому все заголовочные файлы имеют расширение '.h' предназначенное для использования с языком Си. Это расширение содержит объявление функции и определения макросов. Заголовочные файлы могут быть запрошены с помощью директивы препроцессора #include.

Существует 2 типа заголовочных файлов:

1. Глобальный заголовочный файл Это заголовочный файл, который предоставляется компилятором. Глобальные заголовочные файлы не могут быть изменены пользователем и обеспечивают базовую функциональность. (Например: <stdio.h>, <math.h>, <string.h> и др.)

2. Локальный заголовочный файл. Этот тип заголовочных файлов определяется пользователем. Они создаются с помощью программы. Определяемые пользователем заголовочные файлы подобны встроенным заголовочным файлам. (Например: #include "File_name.h")

Чтобы использовать заголовочные файлы в программе, необходимо использовать директиву #include. Всякий раз, когда появляется потребность включить глобальные заголовочные файлы, пишутся < > (угловые скобки).[25]

На рисунках 1.5 – 1.9 приведен фрагмент кода, демонстрирующий использование заголовочных файлов при программировании.

Первоначально необходимо описать все биты одного из регистров.

```
/*====== ADC control register (ADC_CR)    Address offset: 0x08    Reset value: 0x2000 0000 =====*/
typedef struct{
    uint32_t    ADEN:    1;     /*    Bit 0 ADEN: ADC enable control
                                   This bit is set by software to enable the ADC. The ADC will be
                                   effectively ready to operate once the flag ADRDY has been set. */
    uint32_t    ADDIS:    1;    /*Bit 1 ADDIS: ADC disable command*/

    uint32_t    ADSTART:    1;  /*Bit 2 ADSTART: ADC start of regular conversion    */
    uint32_t    JADSTART:    1; /*Bit 3 JADSTART: ADC start of injected conversion*/

    uint32_t    ADSTP:    1;    /*Bit 4 ADSTP: ADC stop of regular conversion command*/
    uint32_t    JADSTP:    1;   /*Bit 5 JADSTP: ADC stop of injected conversion command */
    uint32_t    nu_1:    22;    /*Bits 27:6 Reserved, must be kept at reset value.    */
    uint32_t    ADVREGEN:    1; /*Bit 28 ADVREGEN: ADC voltage regulator enable    */
    uint32_t    DEEPPWD:    1;  /*Bit 29 DEEPPWD: Deep-power-down enable    */
    uint32_t    ADCALDIF:    1; /*Bit 30 ADCALDIF: Differential mode for calibration*/
    uint32_t    ADCAL:    1;    /* Bit 31 ADCAL: ADC calibration */
}Ts_ADC_CR;
```

Рисунок 1.5 – Описание бит регистра

Архитектурой микроконтроллера определено, что с его памятью можно общаться словами только по 32 бита, поэтому необходимо объединить регистр и 32-битное слово.

```
20  typedef union
21  {    Ts_ADC_CR    reg;
22      uint32_t    u32;
23  }
24  Tu_ADC_CR;
```

Рисунок 1.6 – Объединение регистра и 32 битного слова

Затем описывается макрос, в котором происходит работа с эти регистром.

```
26      #define MAKE_ADC(ADCName, className, ID)
27      class className
28      {
29          public:
30          enum{Id = ID};
31
32          inline static ADC_TypeDef* GetADCName()
33          {
34              return (ADCName);
35          }
36
37          inline static void Set_CR(uint32_t value)
38          {
39              ADCName->CR = value&0xF000003FU;
40          }
41
42      };
```

Рисунок 1.7 – Описание макроса

Такой макрос позволяет создавать одинаковые АЦП на контроллере их может быть несколько.

На рисунке 1.8 приведен пример работы с описанным ранее регистром.

```
44      /* Software procedure to calibrate the ADC */
45      inline static uint32_t calibrate(void)
46      {
47          Tu_ADC_CR CR_val;
48
49          CR_val.u32 = ADC_reg::Get_CR();
50          CR_val.reg.ADCALDIF = 0;
51          CR_val.reg.ADCAL = 1;       //4.Set ADCAL = 1
52          ADC_reg::Set_CR(CR_val.u32);
53          do
54          {                           //5.Wait until ADCAL=0.
55              CR_val.u32 = ADC_reg::Get_CR();
56          }while(CR_val.reg.ADCAL);   //5.Wait until ADCAL = 0.
57
58              return 0;
59      }
```

Рисунок 1.8 – Пример работы с регистром

По функциональному описанию возможно сразу писать программу используя названия бит как указано в PDF документе, что очень удобно и создает легко читаемый код.

Задача сгенерировать описание регистров и создать такие структуры, чтобы потом использовать их в коде. и сразу в коде иметь возможность посмотреть что этот бит означает.

```
61  typedef struct
62  {
63      uint32_t    ADEN:    1;     // Bit 0 ADEN: ADC enable control
64      ....
65      uint32_t    ADCAL:    1;    // Bit 31 ADCAL: ADC calibration
66  }
67  Ts_ADC_CR;
68
69  typedef union
70  {
71      Ts_ADC_CR    reg;
72      uint32_t    u32;
73  }
74  Tu_ADC_CR;
```

Рисунок 1.9 – Пример созданных структур

Процесс создания и использования заголовочных файлов делает написание программ более эффективным. Заголовочные файлы можно сделать доступными для всех проектов пользователя, как глобальные

заголовочные файлы, в отличие от локальных заголовочных файлов. Локальные заголовочные файлы доступны только для проекта, в котором они были созданы. Кроме того, можно разделить проекты на части, чтобы разные программисты могли писать разные разделы программ и сохранять их как заголовочные файлы для использования во всех проектах всеми программистами компании.[24]

Вывод по разделу 1

В первом разделе выпускной квалификационной работы был произведен анализ предметной области.

Первоначально был кратко рассмотрен принцип работы микроконтроллеров, а также приведены несколько примеров их применения в разных областях жизни человека. После чего осуществлено знакомство с составом документации, поставляющимся совместно с микроконтроллерами от производителя, и приведены примеры нескольких страниц из технического паспорта.

В заключении описана цель создания заголовочных файлов в программировании и преимущество их применения при написании программ.

2 Обзор методов преобразования файлов формата pdf в формат заголовочных файлов

Преобразование файлов формата PDF в заголовочные файлы (файлы с расширением .h), используемые в программировании на C++, не является распространенной процедурой, поскольку формат .h не является широко известным форматом файлов. Однако существует несколько методов, которые можно использовать для преобразования файлов формата PDF в формат .h. Ниже будут рассмотрены эти методы.

1. Ручное преобразование. Одним из вариантов преобразования является ручное создание нового заголовочного файла и определение структур, функций и переменных, включенных в PDF-файл. Оно включает в себя копирование и вставку, то есть можно вручную скопировать код или текст из PDF-файла и вставить его в новый заголовочный файл (.h). Этот метод прост, но может занять много времени при работе с большими файлами или сложным кодом. Также стоит учитывать, что технические паспорта микроконтроллеров зачастую защищены от изменения, поэтому скопировать текст из них не так уж просто.

2. Онлайн-инструменты преобразования. Существует несколько онлайн-инструментов, которые могут конвертировать PDF-файлы в текстовый или любой другой формат, который затем можно скопировать и сохранить как заголовочный файл, хотя качество и точность этих инструментов может варьироваться. Среди популярных онлайн-инструментов конвертации можно назвать Zamzar, Online Convert и Convertio.

Далее будет рассмотрено преобразование файла на примере работы онлайн конвертора Zamzar.[26]

В списке возможных конвертируемых форматов, представленных на рисунке 2.1, необходимо найти тип исходного файла (PDF), а затем тип желаемого итогового файла.

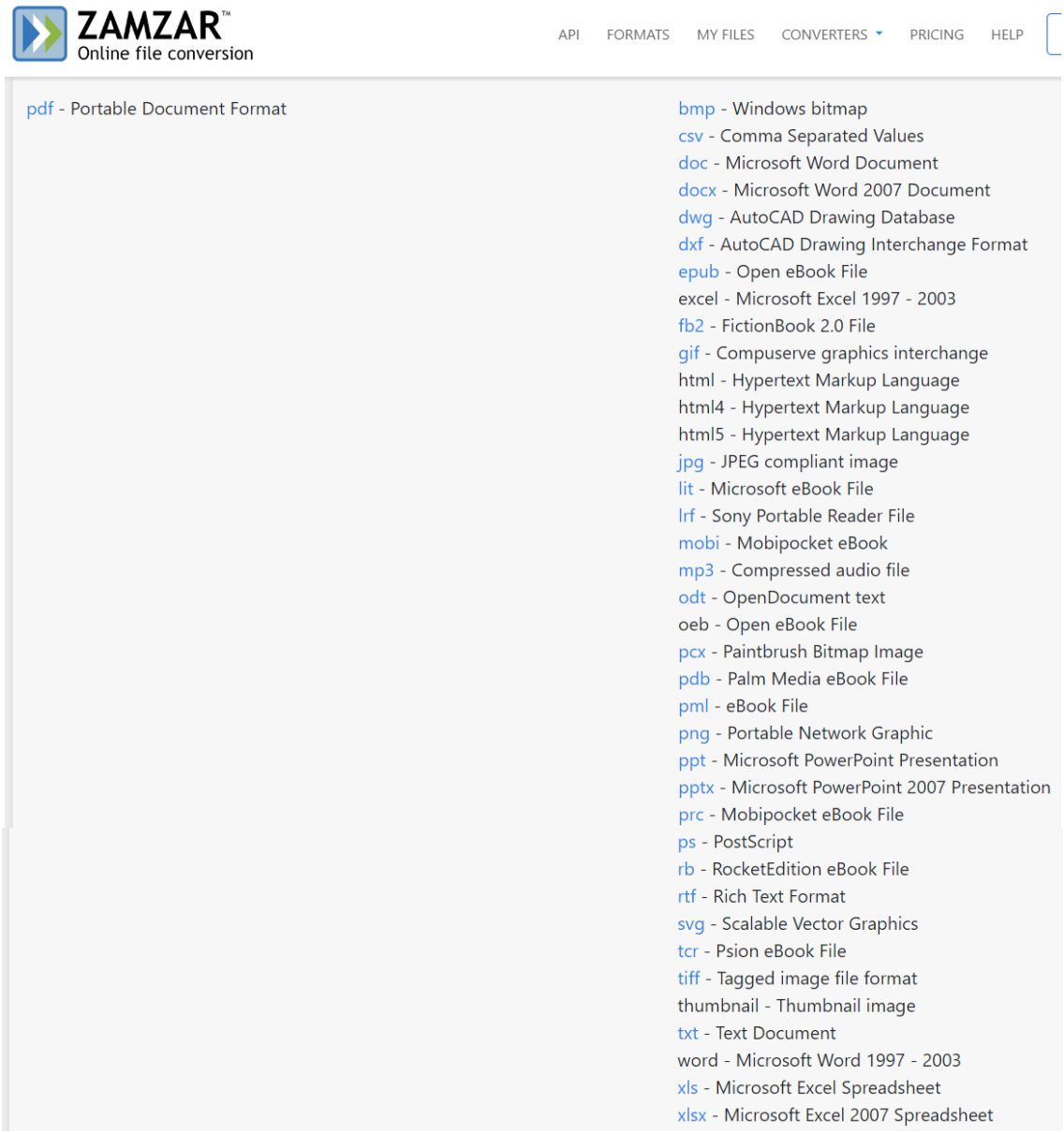


Рисунок 2.1 – Онлайн конвертор Zamzar. Выбор формата

Сразу же можно выделить минус, что отсутствует возможность преобразования файла напрямую в формат .h, поэтому выбор падает на преобразование файла в текстовый формат, который потом в ручную будет необходимо заново сохранить в необходимый.

После перехода по гиперссылке *pdf* открывается страница для загрузки файла, выбора итогового формата и кнопки активирующий процесс преобразования (рисунок 2.2).

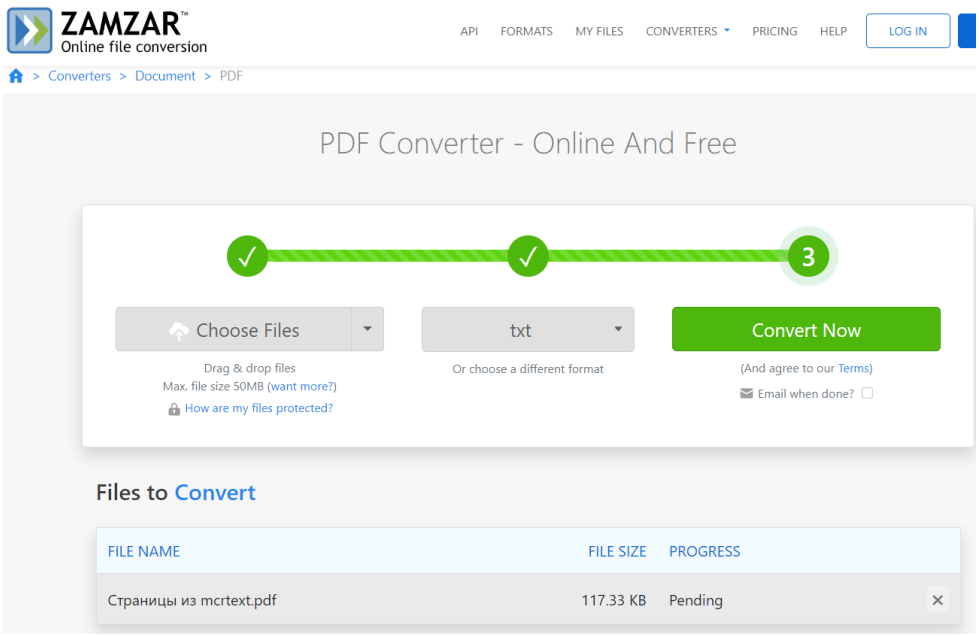


Рисунок 2.2 – Онлайн конвертор Zamzar. Преобразование файла

В нижней части страницы отображается имя и вес выбранного исходного файла, а также прогресс процесса преобразования.

По завершении конвертирования открывается новая страница сайта, представленная на рисунке 2.3, с которой можно скачать итоговый файл.

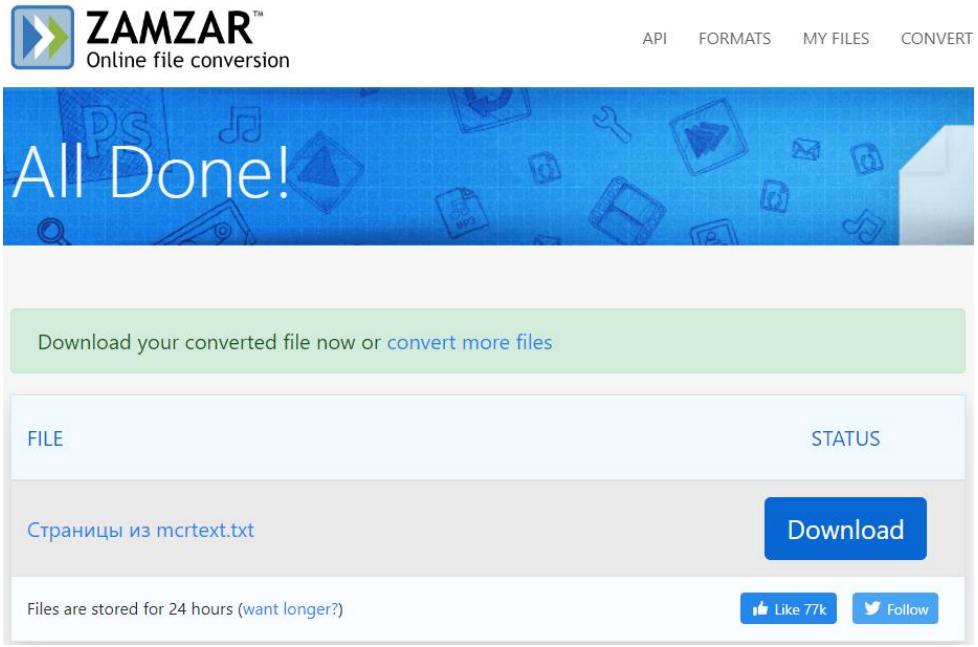


Рисунок 2.3 – Онлайн конвертор Zamzar. Получение итогового файла

При открытии итогового файла, приведенного на рисунке 2.4, можно увидеть, что преобразовался весь возможный текст, который был размещен на страницах исходного PDF-файла. То есть, необходимо будет вносить правки вручную, для выбора необходимых фрагментов. Из плюсов можно отметить, что относительно сохранилась структура файла.

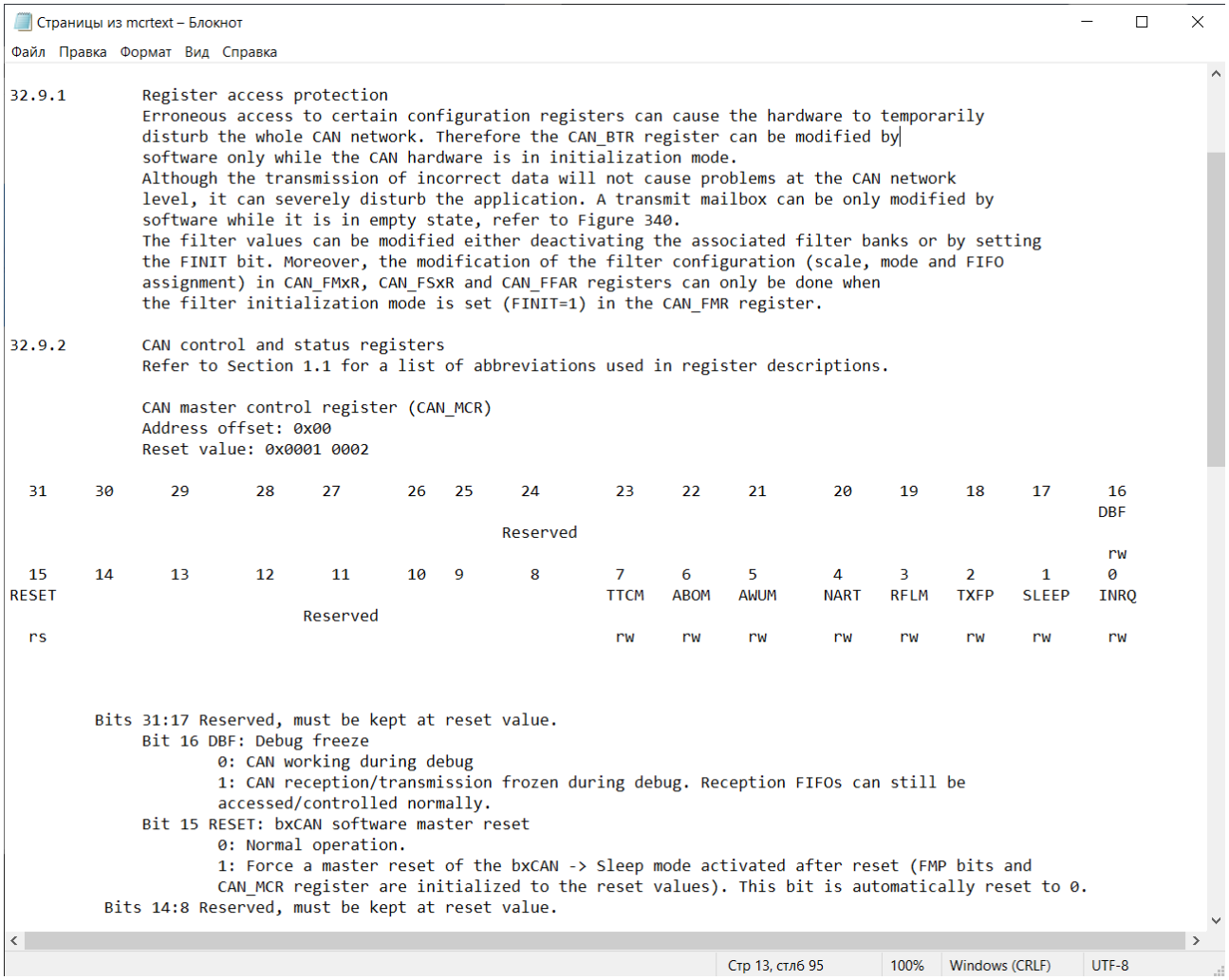


Рисунок 2.4 – Итоговый файл после конвертирования

3. Преобразование файла PDF в текст. Некоторые программы для преобразования PDF-файлов могут конвертировать их в обычный текстовый формат, который затем можно редактировать и сохранять в виде h-файла. Однако, для обеспечения правильного форматирования и синтаксиса может потребоваться ручное редактирование.

Также одним из вариантов взаимодействия с конвертированным текстовым файлом может быть дальнейшее использование программных скриптов, которые содержат в себе регулярные выражения, задача которых заключается в распознавании шаблонов и простого извлечения необходимой информации.

Регулярные выражения используются для определения шаблона поиска в текстовой строке.

С их помощью, помимо прочего, можно выполнять следующие действия:

- Проверка соответствия вводимых данных заданному шаблону, например, можно проверить, является ли значение, введенное в HTML-формуляр, действительным адресом электронной почты.
- Поиск шаблона в фрагменте текста, например, проверка, появляется ли определенное слово в документе.
- Извлечение определенных фрагментов текста, например, извлечение почтовых индексов из таблицы.
- Замена части текста, например, замена любое появление слова другим словом.
- Разделение большого текста на более мелкие части, например, разделение текста при появлении символов точки, запятой или новой строки.[21]

Регулярные выражения распространены повсеместно. Ни один современный язык программирования нельзя назвать полноценным, пока он не поддерживает регулярные выражения.

Несмотря на то, что они широко распространены в языках и фреймворках, регулярные выражения еще не получили широкого распространения в инструментарии современного кодера. Одной из причин, часто используемых для объяснения этого, является трудная кривая обучения. Регулярные выражения трудно освоить и очень сложно читать, если они написаны небрежно.[25]

Ниже будут приведены примеры решения задачи выделения фрагментов текста разными способами реализованными на Python.

1. Поиск определенного слово в строке.

С помощью данного фрагмента, приведенного на рисунке 2.5, происходит проверка, существует ли слово inform в поисковой строке. И если оно есть, то программа выводит результат, говорящий There is inform.[21]

```
1  import re
2
3  if re.search("inform","we need to inform him with the latest information"):
4      print("There is inform")
```

Рисунок 2.5 – Поиск слова в строке

Этот процесс можно улучшить дополнив его методом, который будет выполнять аналогичные действия (рисунок 2.6).

```
1  import re
2
3  allinform = re.findall("inform","We need to inform him with the latest information!")
4
5  for i in allinform:
6      print(i)
```

Рисунок 2.6 – Поиск слова в строке с помощью метода

В данном конкретном случае, inform будет встречаться дважды. Один раз из inform, а другой из information.

2. Сопоставление слов с шаблонами.

Для примера будет взята входная строка «Sat, hat, mat, pat», в которой необходимо найти определенные слова. [21]

У слов в строке можно увидеть сходства, например, что буквы *a* и *t* являются общими для всех слов. Выражение *[shmp]* в коде обозначает начальную букву слов, которые нужно найти. Таким образом, любая подстрока, начинающаяся с букв *s*, *h*, *m* или *p*, будет рассматриваться для

поиска, любая из них, за которой обязательно следует *at* в конце. Фрагмент кода представлен на рисунке 2.7.

```
import re

Str = "Sat, hat, mat, pat"

allStr = re.findall("[shmp]at", Str)

for i in allStr:
    print(i)
```

Рисунок 2.7 – Сопоставление слов с шаблонами

Результатом работы данного фрагмента будет являться выражение:

hat

mat

pat

Стоит обратить внимание, что регулярные выражения обладают удивительной читабельностью, но очень чувствительны к регистру.

3. Сопоставление серии диапазона символов.

Данный подход может быть применен, если необходимо вывести все слова, первая буква которых должна начинаться между *h* и *m* и обязательно сопровождаться окончанием *at*. Полученный фрагмент кода приведен на рисунке 2.8. [21]

```
import re

Str = "sat, hat, mat, pat"

someStr = re.findall("[h-m]at", Str)

for i in someStr:
    print(i)
```

Рисунок 2.8 – Работа с диапазоном символов

Результатом работы данного фрагмента будет являться выражение:

hat

mat

4. Замена строки.

Далее будет рассмотрена еще одна операция с использованием регулярными выражениями, в которой происходит замена элемент строки на что-то другое. Это очень просто и может быть проиллюстрировано фрагментом кода представленном на рисунке 2.9. [21]

```
1  import re
2
3  Food = "hat rat mat pat"
4
5  regex = re.compile("[r]at")
6
7  Food = regex.sub("food", Food)
8
9  print(Food)
```

Рисунок 2.9 – Пример замены строки

В приведенном выше примере слово *rat* заменяется словом *food*. Окончательный результат будет выглядеть следующим образом:

hat food mat pat

В данном случае используется метод замены регулярных выражений, который имеет огромное количество практических применений.

После рассмотрение существующих методов преобразования файла для каждого можно выделить плюсы и минусы его использования.

1. Ручное преобразование.

Плюсы:

– Дает полный контроль над содержанием и структурой результирующего заголовочного файла.
– Подходит для небольших файлов или простых программ.
– Может быть настроен в соответствии с конкретными требованиями.

Минусы:

- Требует много времени и чреват ошибками, особенно для больших или сложных программ.
- Требуется внимание к деталям, поэтому чтобы обеспечить правильное определение структур, функций и переменных, необходимо ручное редактирование.
- Не подходит для преобразования PDF-файлов со сложным форматированием или изображениями.

2. Онлайн-инструменты преобразования.

Плюсы:

- Удобны и просты в использовании, поскольку не требуют установки программного обеспечения.
- Быстро справляются с процессом преобразования.
- Могут быть полезны для извлечения основного текстового содержимого из простых PDF-файлов.

Минусы:

- Качество и точность могут отличаться у разных онлайн-инструментов.
- Может не захватывать сложное форматирование, структуры или изображения.
- Может не подходить для больших или более сложных программ.

3. Преобразование файла PDF в текст.

Плюсы:

- Быстрый и простой способ преобразования PDF-файла в текстовый формат, который затем можно отредактировать и сохранить как заголовочный файл.

– Не требует специализированного программного обеспечения или знаний программирования.
– Может быть эффективным для небольших или более простых программ.

Минусы:

– Может потребовать дополнительного ручного редактирования и форматирования для правильного определения структур, функций и переменных.
– Ограничивается извлечением текста; может не захватывать форматирование или другие специфические элементы из PDF.
– Может не подойти для больших или более сложных программ.

По итогу анализа существующих методов, можно сделать вывод, что ни один подход не преобразует файл напрямую из формата PDF в заголовочный файл. И если все таки обратиться к одному из методов преобразования, то каждый из них не справляется со своей задачей идеально, в большинстве случаев страдает структура файла, либо подход не ориентирован на большое количество информации. Важно отметить, что работа с микроконтроллерами требует точности и использует многостраничные документы, поэтому создание программного обеспечения, которое справлялось бы с задачей преобразования формата, очень актуально на данный момент.

Таким образом, можно составить некоторые требования к создаваемому программному обеспечению, а также к формированию итогового заголовочного файла.

Создаваемая система должна работать на основе скрипта, написанного на языке Python, содержащего регулярные выражения.

Принцип работы его заключается в нахождении определенных фрагментов текста, расположенного между маркерами, которые будут

определяться регулярными выражениями. Сначала происходит считывание в словарь каждой строчки документа, которую необходимо подставить в шаблон, после чего он используется для постановки полученных выражений в правильном порядке. При нахождении первого маркера система начинает копирование текста из файла формата PDF, сохраняя его структуру, после нахождения второго маркера – копирование прекращается. Скрипт должен срабатывать единовременно на обозначенных пользователем страницах исходного документа.

Входными данными, которые пользователь должен предоставить системе будут являться PDF-файл, являющийся техническим паспортом микроконтроллера, а также номера страниц, с которыми системе необходимо будет работать.

Результатом работы является сформированный программой итоговый файл, который должен быть сохранен в формате заголовочного файла языка C++, то есть с расширением .h. Внутри файла должен находиться скопированный структурированный текст.

Вывод по разделу 2

Во втором разделе выпускной квалификационной работы был произведен обзор существующих методов преобразования файлов из формата PDF в формат заголовочных файлов CPP, а также рассмотрены варианты реализации поиска и замены фрагментов текста на основе использования регулярных выражений.

Первоначально были рассмотрены существующие методы преобразования и принцип их работы. После чего проведен анализ каждого подхода и выделены их преимущества и недостатки.

В заключение подведен итог анализа и выделена актуальность создания нового программного обеспечения, а также обозначены требования к нему.

3 Программная реализация

3.1 Средства разработки программного обеспечения

Инструменты разработки программного обеспечения – это приложения, фреймворки, библиотеки или сервисы, которые помогают разработчикам в различных аспектах жизненного цикла разработки программного обеспечения. Эти инструменты направлены на повышение производительности, улучшение совместной работы, оптимизацию рабочих процессов, обеспечение качества кода и упрощение процесса разработки.

Python стал популярным языком программирования, известным своей простотой, универсальностью и обширными библиотеками. В сфере разработки программного обеспечения Python предлагает богатую экосистему инструментов, которые повышают производительность, оптимизируют рабочие процессы и облегчают сотрудничество. Далее будут рассмотрены различные инструменты разработки программного обеспечения на Python.[6]

Интегрированная среда разработки (IDE) – программное приложение, которое предоставляет комплексные инструменты и функции для облегчения разработки программного обеспечения. IDE обычно объединяет редактор кода, средства автоматизации сборки, отладчик и другие утилиты в единый интегрированный интерфейс. Основная цель IDE – повысить производительность разработчика путем предоставления централизованной среды для кодирования, тестирования и отладки.[24]

Примеры существующих IDE для программирования на Python:

- PyCharm. Многофункциональная IDE, предлагающая создание кода, отладку и интеграцию с популярными фреймворками.
- Visual Studio Code: Легкий и расширяемый редактор кода с сильной поддержкой Python.

– Spyder. Это IDE с открытым исходным кодом, специально разработанная для научных вычислений и анализа данных на языке Python.

Редакторы кода – это программные инструменты, предназначенные для написания и редактирования исходного кода. В отличие от интегрированных сред разработки (IDE), редакторы кода обычно сосредоточены на предоставлении легкой и настраиваемой среды для кодирования, без широких возможностей и интеграций, которые есть в IDE.[10]

Примеры существующих редакторов для программирования на Python:

– Sublime Text. Широко используемый редактор кода, известный своей скоростью и простотой, поддерживает несколько языков.
– Atom. Это редактор с открытым исходным кодом, разработанный GitHub. Хорошо настраивается с помощью дополнительных плагинов.
– Notepad++. Это бесплатный редактор исходного кода для Windows. Поддерживает несколько языков программирования, предлагает подсветку синтаксиса, сворачивание кода и удобный интерфейс.

В целом, разработчики могут писать код на любой платформе/программном обеспечении. Это может быть командная строка Windows или даже приложение «Блокнот». Однако такие платформы позволяют только кодировать проект Python без каких-либо расширенных функций, вспомогательных возможностей или опций отладки.

Основным отличительным фактором между редакторами кода Python и IDE является то, что IDE поставляется с графическим интерфейсом, а редакторы кода – нет. Редактор кода может быть частью IDE, которая оптимизирует процесс кодирования.[24]

Также важным ресурсом для разработки программного обеспечения, являются различные библиотеки, предлагающие специализированные инструменты и функциональные возможности, которые упрощают и ускоряют процесс кодирования.

Важность использования библиотек невозможно переоценить. Во-первых, они способствуют повторному использованию кода, позволяя разработчикам использовать существующий, проверенный код, а не начинать с нуля. Это не только экономит время, но и обеспечивает надежность и эффективность. Во-вторых, библиотеки часто предоставляют оптимизированные решения, разработанные для конкретных задач, что приводит к более быстрому и производительному коду.[12]

Кроме того, библиотеки способствуют стандартизации. Придерживаясь установленных соглашений, разработчики могут создавать код, который является последовательным, удобным в обслуживании и более понятным для коллег. Также многие библиотеки имеют активные сообщества, предлагающие поддержку, ресурсы и обсуждения, что способствует обучению, устранению неполадок и сотрудничеству, что еще больше улучшает процесс разработки.

Инструменты разработки программного обеспечения на языке Python играют важную роль в оптимизации процесса разработки. Внедряя их в свои рабочие процессы, можно увеличить уровень производительности и эффективности. Под разрабатываемое приложение будет использована интегрированная среда разработки Visual Studio Code.

3.2 Разработка программного кода

Реализуемый код должен решать задачу по извлечению определенных фрагментов текста из документа. Процесс начинается с чтения каждой строки документа и сохранения ее в словаре для последующей подстановки в шаблон. Затем полученные выражения располагаются в правильной последовательности. Начиная с первого маркера, система копирует текст из PDF-файла, сохраняя его оригинальную структуру. Процесс копирования останавливается при появлении второго маркера.

Результатом работы программы является созданный заголовочный файл C++ (с расширением .h). Этот файл содержит скопированный структурированный текст.

Для наглядности будет построена блок-схема (рисунок 3.1) отражающая пошаговый процесс, которому следует программа для достижения своей цели.

Рисунок 3.1 – Алгоритм работы программы

Для реализации поставленной задачи будут использоваться Python библиотеки, такие как *pdfplumber* и *re*.

pdfplumber – это библиотека, используемая для извлечения текста и данных из файлов PDF.

Она предоставляет высокоуровневый интерфейс для извлечения текста, таблиц и другой информации из PDF-документов. С помощью *pdfplumber* можно получить доступ к отдельным страницам PDF, извлечь текст и выполнить такие операции, как поиск, обрезка и извлечение таблиц.[4]

Программа поддерживает различные форматы PDF и хорошо работает как с отсканированными, так и с созданными в цифровом виде PDF-файлами.

Она широко используется в задачах, связанных с извлечением, анализом и обработкой данных PDF, таких как разбор документов, поиск данных и создание отчетов.

re – это встроенный модуль Python, который обеспечивает поддержку работы с регулярными выражениями.

Используя *re*, можно искать, сопоставлять и манипулировать текстом на основе шаблонов, заданных комбинацией буквенных символов и специальных символов. Регулярные выражения обеспечивают гибкий и эффективный способ работы со сложными строковыми шаблонами, что делает *re* универсальным инструментом для обработки и манипулирования текстом в Python.[4]

Первоначально код импортирует описанные выше библиотеки для дальнейшей работы с ними (рисунок 3.2).

```
1   import pdfplumber
2   import re
```

Рисунок 3.2 – Импорт библиотек

Затем используется функция *extract_text_from_pdf* (рисунок 3.3), которая принимает на вход путь к файлу, номер начальной страницы и номер конечной страницы. Она использует *pdfplumber* для открытия PDF-файла, перебирает указанный диапазон страниц, извлекает текст с каждой страницы и объединяет его в одну строку. Затем функция возвращает извлеченный текст.

```python
# Функция для извлечения текста из PDF-файла
def extract_text_from_pdf(file_path, start_page, end_page):
    with pdfplumber.open(file_path) as pdf:
        text = ""
        for page_number in range(start_page - 1, end_page):
            page = pdf.pages[page_number]
            text += page.extract_text()
        return text
```

Рисунок 3.3 – Функция для извлечения текста из PDF-файла

После этого определяется еще одна функция под названием *save_text_as_h* (рисунок 3.4), которая принимает в качестве параметров извлеченный текст и путь к выходному файлу. Она открывает выходной файл в режиме записи и записывает извлеченный текст в файл.

```python
# Функция для сохранения текста в файл формата .h
def save_text_as_h(text, output_file_path):
    with open(output_file_path, 'w') as file:
        file.write(text)
```

Рисунок 3.4 – Функция для сохранения текста в файл формата .h

Следующая функция *find_register_description* принимает извлеченный текст в качестве входных данных (рисунок 3.5). Она использует регулярные выражения (*re.search*) для поиска определенного шаблона в тексте. Если совпадение найдено, функция присваивает найденное описание переменной *register_description*. Если совпадение не найдено, присваивается пустая строка.

```python
# Функция для поиска описания регистра в тексте
def find_register_description(text):
    register_description_pattern = r"CAN master control register.*?Bit \d+.*?CAN master status register."
    register_description_match = re.search(register_description_pattern, text, re.DOTALL)
    if register_description_match:
        register_description = register_description_match.group(0)
    else:
        register_description = ''
    return register_description
```

Рисунок 3.5 – Функция для поиска описания регистра в тексте

Функция *format_register_description* принимает на вход переменную *register_description* (рисунок 3.6). Она разбивает описание на отдельные строки и обрабатывает каждую строку на основе определенных условий. Она обрабатывает строки, связанные со смещением адреса регистра, значением сброса, битами и их соответствующими описаниями. На основе извлеченной информации генерируется форматированное описание.

```python
def format_register_description(register_description):
    '''
    Переводи текст из ПДФ в список строчек, которые относятся к каждому заголовку с помощью словаря
    '''
    register_lines = register_description.splitlines()
    formatted_description = ''
    flag = False
    bits = {i: [] for i in range(13)}
    counter = 0
    patter_for_first_line = r"CAN master control register \(CAN_MCR\)"

    for line in register_lines:
        line = line.strip()

        # First and second line
        if re.search(patter_for_first_line, line) or line.startswith("Address offset:") or line.startswith("Reset value:"):
            if line.startswith("Address offset:"):
                formatted_description += line + ' '
            elif line.startswith("Reset value:"):
                formatted_description += line + '*/' + f"\ntypedef struct __attribute__((packed)) {{\n"
            else:
                formatted_description += '/*' + line + ' '

        # Bit and info
        elif (line.startswith("Bit ") and line[4] in '01234567' and line[5] == ' ') or \
            (line.startswith("Bit ") and line[4:6] in '1516' and line[6] == ' '):
            if flag:
                formatted_description = formatted_description.rstrip()
            parts = line.split(":") # ['Bit 0 INRQ', 'Initialization request']
            bit_name = parts[0].split()[-1] # ['Bit', '0', 'INRQ']
            line = f"\n\tuint32_t {bit_name}: 1;/* {parts[0]}: {parts[1]} "
            counter += 1
            bits[counter].append(line + '\n')

            flag = True
        # Bits
        elif (line.startswith('Bits')):
            line = line.split()
            line = f'\n\t{line[0]} {line[1]}; /* {" ".join(line[2:])}'

            counter += 1
            bits[counter].append(line + '\n')

        # Discription
        else:
            if line and flag:
                pattern = r"\b\d+/\d+\b.*?bxCAN\b"
                seccond_pattern = r"RM\d+ Rev \d+ \d+/\d+"
                third_pattern = r"For.*Bus-Off state.*"
                another_pattern = r"\d+Controller area network \(bxCAN\) RM\d+"
                if not re.findall(pattern, line) and not re.search(r'^Note:.*', line) and not re.search('CAN master status register',
                    not re.search(seccond_pattern, line) and not re.findall(third_pattern, line) and not re.findall(another_pattern,
                    bits[counter].append(f'\t\t{line}\n')

            else:
                pass

    for i in range(12, 0, -1):
        for l in bits[i]:
            formatted_description += l
        formatted_description = formatted_description.rstrip()
        formatted_description += '*/'

    formatted_description = formatted_description.rstrip()
    formatted_description += '\n}'

    return formatted_description
```

Рисунок 3.6 – Функция для обработки строк с помощью словаря

Затем код вызывает функцию *extract_text_from_pdf* для извлечения текста из файла PDF, заданного путем к файлу и диапазоном страниц. Фрагмент кода приведен на рисунке 3.7.

```
# Извлечение текста из PDF
pdf_text = extract_text_from_pdf(str(bookname_tf.get()), int(firstpg_tf.get()), int(lastpg_tf.get()))
```

Рисунок 3.7 – Функция для извлечения текста из PDF-файла

Извлеченный текст передается в функцию *find_register_description* для поиска описания регистра (рисунок 3.8).

```
# Поиск описания регистра
register_description = find_register_description(pdf_text)
```

Рисунок 3.8 – Функция для поиска описания регистра

Функция *format_register_description*, представленная на рисунке 3.9, вызывается с описанием регистра в качестве входных данных, чтобы отформатировать его в соответствии с определенной структурой.

```
# Форматирование описания регистра
formatted_description = format_register_description(register_description)
```

Рисунок 3.9 – Функция для форматирования описания регистра

Наконец, отформатированное описание сохраняется в .h-файле, указанном в папке *output_file_path*, с помощью функции *save_text_as_h* (рисунок 3.10). Модуль *messagebox.showinfo* отвечает за вывод сообщения о завершении преобразования на экран в будущем пользовательском интерфейсе. Метод *os.system* открывает полученный заголовочный файл через Блокнот, для наглядности работы.

```
# Сохранение описания регистра в файл формата .h
output_file_path = 'output.h'
save_text_as_h(formatted_description, output_file_path)
messagebox.showinfo('Massege', f'Преобразование завершено!')
os.system("notepad output.h")
```

Рисунок 3.10 – Функция для сохранения заголовочного файла

Приведенный код демонстрирует процесс извлечения текста из PDF-файла, нахождения в тексте определенного описания регистра, его форматирования и сохранения в .h-файле.

3.3 Разработка пользовательского интерфейса

Для создания простого пользовательского интерфейса для программного обеспечения будет использоваться библиотека *tkinter* в Python.

tkinter – это стандартный набор инструментов GUI (графического интерфейса пользователя) для создания настольных приложений. Она предоставляет набор модулей и классов, которые позволяют разработчикам создавать различные окна, кнопки, ярлыки, текстовые поля и другие элементы GUI.[24]

С помощью *tkinter* разработчики можно создавать кроссплатформенные приложения, работающие на различных операционных системах, включая Windows, macOS и Linux. Он использует простой и интуитивно понятный синтаксис, что позволяет новичкам сравнительно легко приступать к созданию графических приложений.

Также *tkinter* поддерживает обработку событий для взаимодействия с пользователем, таких как нажатие кнопок и ввод с клавиатуры, управление макетом для расположения и позиционирования элементов GUI в окнах или фреймах, интеграцию графики и изображений в пользовательский интерфейс, настройку внешнего вида с помощью различных опций стилизации.

Далее на рисунках будет приведенный код для создания базового графический интерфейс пользователя.

Первоначально необходимо импортировать необходимые модули из библиотеки *tkinter*, включая класс *Tk* для создания окна приложения и модуль *messagebox* для отображения окон сообщений (рисунок 3.11).[26]

```
from tkinter import *
from tkinter import messagebox
```

Рисунок 3.11 – Импорт модулей библиотеки

Затем с помощью конструктора *Tk()* создается окно приложения, которое представляет собой главное окно GUI-приложения.

Окну присваивается заголовок *«Converter»* и размер 350x150 пикселей с помощью методов *title()* и *geometry()*. Фрагмент кода приведен на рисунке 3.12.

```
window = Tk() #Создаём окно приложения.
window.title("Converter")
window.geometry('350x150')
```

Рисунок 3.12 – Создание окна приложения

После этого создается виджет *Frame* как контейнер для других элементов графического интерфейса. Фрейм конфигурируется с горизонтальной и вертикальной прокладкой с помощью параметров *padx* и *pady*. Затем он упаковывается в окно с помощью метода *pack()*, при этом устанавливается свойство *expand=True*, чтобы фрейм заполнил все окно (рисунок 3.13).

```
frame = Frame(
    window, #Обязательный параметр, который указывает окно для размещения Frame.
    padx=1, #Задаём отступ по горизонтали.
    pady=1 #Задаём отступ по вертикали.
)
frame.pack(expand=True) #Позиционирование виджета окне.
```

Рисунок 3.13 – Создание фрейма

Для отображения текстовых подсказок для пользователя создаются несколько виджетов *Label* (рисунок 3.14).

Каждый виджет *Label* ассоциируется с определенным текстом и размещается во фрейме с помощью метода *grid()*, который задает положение строк и столбцов.

```
book_lb = Label(
    frame,
    text="Введите имя файла  "
)
book_lb.grid(row=1, column=1, sticky=E)

first_lb = Label(
    frame,
    text="Введите начальную страницу  "
)
first_lb.grid(row=3, column=1, sticky=E)

last_lb = Label(
    frame,
    text="Введите конечную страницу  "
)
last_lb.grid(row=5, column=1, sticky=E)
```

Рисунок 3.14 – Создание текстовых виджетов

Каждый виджет соответствует запрашиваемой от пользователя информации, то есть для процесса конвертирования необходимо будет ввести имя исходного файла, а также страницы, на которых расположены необходимые биты.

Затем создаются виджеты ввода, чтобы пользователь мог ввести необходимые данные (рисунок 3.15).

Как и текстовые, виджеты ввода размещаются во фрейме с помощью метода *grid()*, задавая положение строк и столбцов. Они будут размещаться параллельно текстовым для удобства пользователя.

```
bookname_tf = Entry(
    frame,
)
bookname_tf.grid(row=1, column=2)

firstpg_tf = Entry(
    frame,
)
firstpg_tf.grid(row=3, column=2)

lastpg_tf = Entry(
    frame,
)
lastpg_tf.grid(row=5, column=2)
```

Рисунок 3.15 – Создание виджетов ввода

Заключительным шагом является создание кнопки, которая будет отвечать за запуск процесса конвертирования.

Создается виджет кнопки с именем *cal_btn* с текстом «Преобразовать». Параметр *command* устанавливается в функцию под названием *parsing*, которая содержит в себе ранее созданный алгоритм извлечения текста из PDF-файла. Для того, чтобы кнопка выделялась на общем фоне дополнительно ей был задан голубой цвет и шрифт Arial.

Виджет кнопки позиционируется во фрейме с помощью метода *grid()* с указанием позиций строк и столбцов. Фрагмент кода приведен на рисунке 3.16.

```
cal_btn = Button(
    frame, #Заготовка с настроенными отступами.
    text='Преобразовать', #Надпись на кнопке.
    font=("Arial", 10),
    background="#ABCDEF",
    command=parsing
)
cal_btn.grid(row=8, column=2,ipadx=3, sticky=SE)
```

Рисунок 3.16 – Создание кнопки

После запуска проекта получаем на выходе всплывающее окно, представленное на рисунке 3.17.

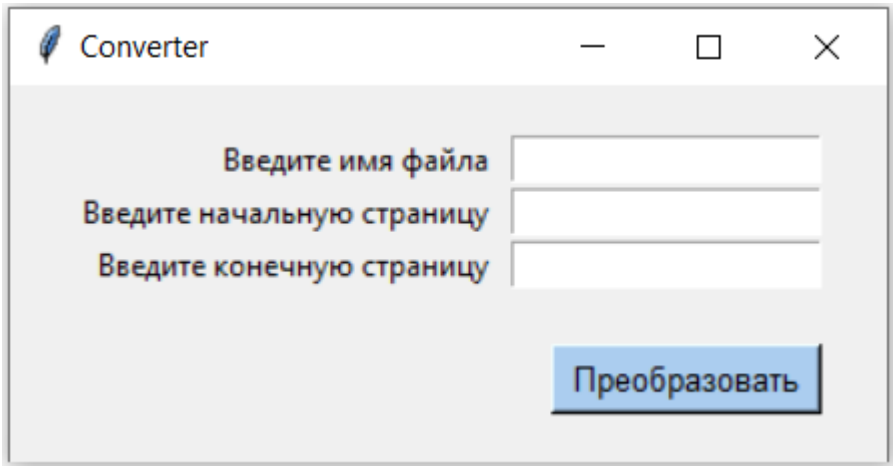


Рисунок 3.17 – Окно созданного приложения

Для проверки его работоспособности введем название исходного документа, размещенного в той же папке, что и файл с кодом проекта, а также страницы, которые необходимо преобразовать (рисунок 3.18).

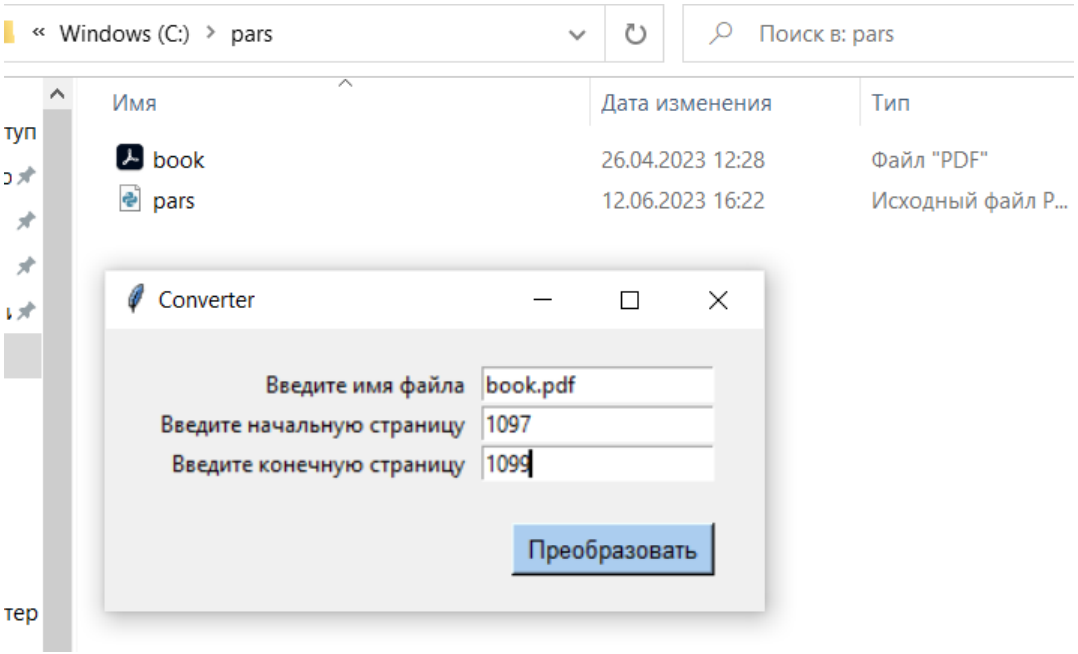


Рисунок 3.18 – Окно созданного приложения с введенными данными

После нажатия кнопки «Преобразовать» запускается процесс конвертирования, по завершению которого на экране появляется сообщение: «Преобразование завершено» (рисунок 3.19).

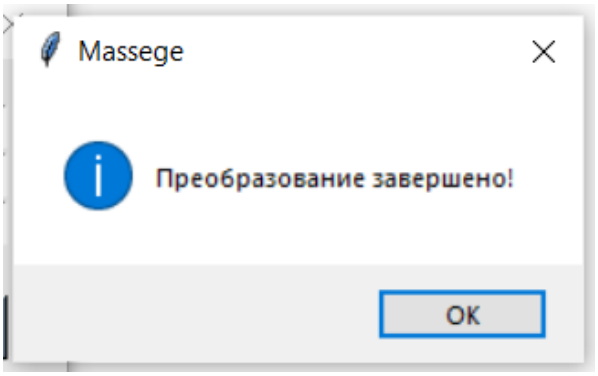


Рисунок 3.19 – Окно «Massege»

Также можно увидеть, что в исходной папке появился итоговый заголовочный файл *output.h* (рисунок 3.20).

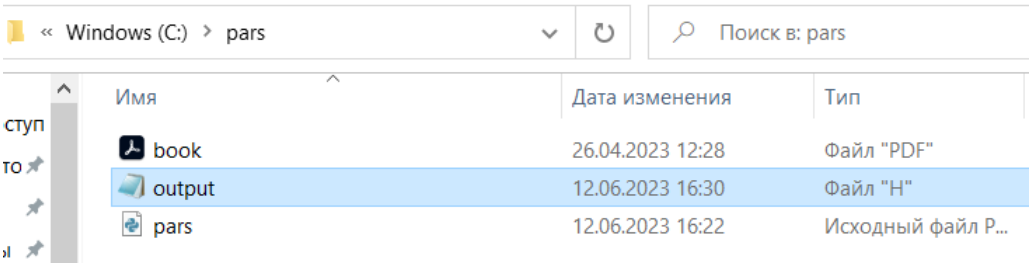


Рисунок 3.20 – Появления файла output.h в исходной папке

Также, для проверки работоспособности программы, при нажатии кнопки «ОК» в окне «Massege» на экране открывается итоговый файл с преобразованным фрагментом. Для удобства трансляции, он открывается в приложении Блокнот (рисунок 3.21).

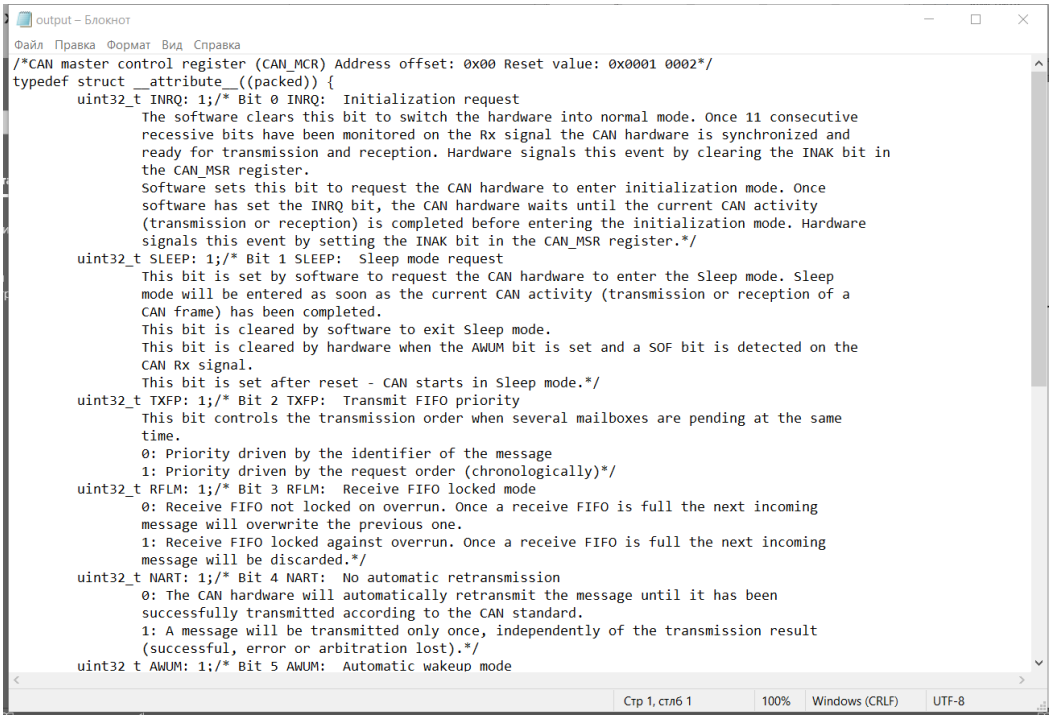


Рисунок 3.21 – Содержание файла output.h

По итогу проведенного тестирование приложения, можно сделать вывод, что оно работает исправно и полностью выполняет свои задачи.

Вывод по разделу 3

Во третьем разделе выпускной квалификационной работы были разработаны на языке программирования Python алгоритм решения поставленной задачи, а также графический интерфейс для удобства работы пользователя с конвертором.

В заключение работа приложения была протестирована на различных PDF-файлах, содержащих информацию о регистрах микроконтроллеров.

Заключение

В ходе выполнения выпускной квалификационной работы была проведена разработка программного обеспечения для преобразования файла, описывающего регистры микроконтроллера в формате pdf, в заголовочный файл на языке программирования C++.

В заключение можно сказать, что каждая из поставленных задач была выполнена:

- В ходе исследования была проанализирована структура документации, прилагаемой к микроконтроллерам, а также принцип их работы на примерах из технических паспортов.
- Были изучены существующие методы автоматизации преобразования pdf-файлов в заголовочные файлы. Рассмотрены различные подходы к преобразованию и оценены их преимущества и недостатки.
- На основе полученной оценки был предложен наиболее подходящий подход для достижения цели работы, легший в основу последующего процесса разработки.
- Были определены подходящие инструменты разработки для проекта. После тщательного рассмотрения было решено, что оптимальным выбором будет язык программирования Python. Универсальность Python, обширные библиотеки и простота использования делают его отличным инструментом для разработки программных решений, включая приложения для преобразования файлов.
- Благодаря применению выбранного подхода и использованию возможностей Python был разработан надежный и прочный код. Программный код эффективно извлекает соответствующую информацию из pdf-файлов, описывающих регистры микроконтроллера, и генерирует соответствующие заголовочные

файлы на языке C++, обеспечивая точность и эффективность процесса преобразования.

– Используя возможности языка Python, был создан графический интерфейс пользователя (GUI) для повышения удобства работы с системой.

Список используемой литературы и используемых источников

1. Букунов С. В. Разработка приложений с графическим пользовательским интерфейсом на языке Python : учебное пособие для СПО / С. В. Букунов, О. В. Букунова. – Санкт-Петербург : Лань, 2023. — 88 с. : ил.

2. Гагарина, Л.Г. Технология разработки программного обеспечения : учеб. пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Сидорова-Виснадул ; под ред. Л.Г. Гагариной. – Москва : ФОРУМ : ИНФРА-М, 2017. – 400 с. – (Высшее образование : Бакалавриат).

3. ГОСТ 7.82–2001. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления : утвержден и введен в действие Постановлением Государственного комитета Российской Федерации по стандартизации и метрологии от 4 сентября 2001 г. № 369-ст : введен впервые : дата введения 2002-07-01. – Москва : Издательство стандартов, 2001. – 26 с

4. Кольцов Д.М., Дубовик Е.В. Справочник PYTHON. Кратко, быстро, под рукой – СПб.: Наука и Техника, 2021. - 288 с., ил.

5. Любанович Билл. Простой Python. Современный стиль программирования. 2-е изд. – СПб.: Питер, 2021. – 592 с.: ил. – (Серия «Бестселлеры O’Reilly»).

6. Основы программирования на языке Python : учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург : Изд-во Урал. ун-та, 2014. – 91с.

7. Очеповский, А.В. Математическое обеспечение и администрирование информационных систем. Бакалаврская работа: учеб.-метод. пособие / А.В. Очеповский, О.М. Гущина, Т.Г. Султанов. – Тольятти : ТГУ, 2018. – 51 с.

8. Очеповский, А.В. Математическое обеспечение и администрирование информационных систем. Выполнение бакалаврской работы : электрон. учеб.-метод. пособие / А.В. Очеповский, О.М. Гущина,

Т.Г. Султанов. – Тольятти : Изд-во ТГУ, 2021. – 1 оптический диск. – ISBN 978-5-8259-1529-6.

9. Программирование на Python в примерах и задачах / Алексей Васильев. — Москва : Эксмо, 2021. — 616 с. — (Российский компьютерный бестселлер).

10. 7 редакторов кода и IDE для Python. URL: https://robotdreams.cc/blog/160-7-redaktorov-koda-i-ide-dlya-python (дата обращения: 3.05.2023).

11. A Guide To PIC Microcontroller Documentation. URL: https://en.wikibooks.org/wiki/A_Guide_To_PIC_Microcontroller_Documentation (дата обращения: 13.03.2023).

12. Advanced Guide to Python 3 Programming / John Hunt – Chippenham, Wiltshire, UK: Midmarsh Technology Ltd., 2019. – 497 p.

13. Data Sheet for PIC16F18857. URL: https://www.microchip.com/en-us/product/PIC16F18857 (дата обращения: 5.03.2023).

14. Intermediate C Programming for the PIC Microcontroller: Simplifying Embedded Programming / Hubert Henry Ward – Leigh, UK: Apress, 2020. – 350 p.

15. Mastering Python Regular Expressions / Félix López, Víctor Romero – Birmingham, UK: Packt Publishing Ltd., 2014. – 97 p.

16. Microcontroller Documentation Explained (Part 1): Datasheet Structure. URL: https://www.elektormagazine.com/articles/microcontroller-documentation-part-1 (дата обращения: 13.03.2023).

17. Microcontroller Documentation Explained (Part 2): Registers and Block Diagrams. URL: https://www.elektormagazine.com/articles/microcontroller-documentation-part-2-registers (дата обращения: 15.03.2023).

18. Modern C++ for Absolute Beginners: A Friendly Introduction to C++ Programming Language and C++11 to C++20 Standards / Slobodan Dmitrović – Belgrade, Serbia: Apress, 2020. – 304 p.

19. Python GUI Programming with Tkinter. Develop responsive and powerful GUI applications with Tkinter / Alan D. Moore – Birmingham, UK: Packt Publishing Ltd., 2018. – 644 p.

20. Python GUI Programming with Tkinter. Second Edition. Design and build functional and user-friendly GUI applications/ Alan D. Moore – Birmingham, UK: Packt Publishing Ltd., 2021. – 860 p.

21. Python RegEx (Regular Expression) — A Comprehensive Guide With Examples. URL: https://medium.com/edureka/python-regex-regular-expression-tutorial-f2d17ffcf17e (дата обращения: 5.05.2023).

22. RM0090 Reference manual. URL: https://www.st.com/resource/en/reference_manual/rm0090-stm32f405415-stm32f407417-stm32f427437-and-stm32f429439-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (дата обращения: 5.03.2023).

23. Software Engineering for Embedded systems. Methods, Practical, Techniques, and Applications. Second edition / Robert Oshana, Mark Kraeling – USA: Elsevier Inc., 2019. – 632 p.

24. Top Python IDEs & Code Editors for 2023. URL: https://aglowiditsolutions.com/blog/best-python-ides-code-editors/ (дата обращения: 2.05.2023).

25. Understanding Regular Expressions in Python. URL: https://medium.com/swlh/understanding-regular-expressions-in-python-21410c64a6f2 (дата обращения: 5.05.2023).

26. ZAMZAR Online file conversion URL: https://www.zamzar.com/ (дата обращения: 5.05.2023).

Приложение А

Листинг итогового программного продукта

```
import pdfplumber
import re
from tkinter import *
from tkinter import messagebox
import os

def parsing():
    # Функция для извлечения текста из PDF-файла
    def extract_text_from_pdf(file_path, start_page, end_page):
        with pdfplumber.open(file_path) as pdf:
            text = ""
            for page_number in range(start_page - 1, end_page):
                page = pdf.pages[page_number]
                text += page.extract_text()
            return text

    # Функция для сохранения текста в файл формата .h
    def save_text_as_h(text, output_file_path):
        with open(output_file_path, 'w') as file:
            file.write(text)

    # Функция для поиска описания регистра в тексте
    def find_register_description(text):
        register_description_pattern = r"CAN master control register.*?Bit \d+.*?CAN master status register."
        register_description_match = re.search(register_description_pattern, text, re.DOTALL)
        if register_description_match:
```

```
            register_description = register_description_match.group(0)
        else:
            register_description = ''
        return register_description


    def format_register_description(register_description):
        '''
        Переводи текст из ПДФ в список строчек, которые относятся к каждому заголовку с помощью словаря
        '''
        register_lines = register_description.splitlines()
        formatted_description = ''
        flag = False
        bits = {i: [] for i in range(13)}
        counter = 0
        patter_for_first_line = r"CAN master control register \(CAN_MCR\)"

        for line in register_lines:
            line = line.strip()

            # First and second line
            if re.search(patter_for_first_line, line) or line.startswith("Address offset:") or line.startswith("Reset value:"):
                if line.startswith("Address offset:"):
                    formatted_description += line + ' '
                elif line.startswith("Reset value:"):
                    formatted_description += line + '*/' + f"\ntypedef struct __attribute__((packed)) {{\n"
```

```
            else:
                formatted_description += '/*' + line + ' '

        # Bit and info
        elif (line.startswith("Bit ") and line[4] in '01234567' and line[5] == '
') or \
            (line.startswith("Bit ") and line[4:6] in '1516' and line[6] == ' '):
            if flag:
            flag = True
        # Bits
        elif (line.startswith('Bits')):
            line = line.split()
            line = f'\n\t{line[0]} {line[1]}; /* {" ".join(line[2:])}'

            counter += 1
            bits[counter].append(line + '\n')

        # Discription
        else:
            if line and flag:
                pattern = r"\b\d+/\d+\b.*?bxCAN\b"
                seccond_pattern = r"RM\d+ Rev \d+ \d+/\d+"
                third_pattern = r"For.*Bus-Off state.*"
                another_pattern = r"\d+Controller area network \(bxCAN\)
RM\d+"
                if not re.findall(pattern, line) and not re.search(r'^Note:.*', line)
and not re.search('CAN master status register', line) and \
                        not re.search(seccond_pattern, line) and not
re.findall(third_pattern, line) and not re.findall(another_pattern, line):
```

```
                    bits[counter].append(f'\t\t{line}\n')
                else:
                    pass

        # Извлечение текста из PDF
        pdf_text = extract_text_from_pdf(str(bookname_tf.get()),
int(firstpg_tf.get()), int(lastpg_tf.get()))

        # Поиск описания регистра
        register_description = find_register_description(pdf_text)

        # Форматирование описания регистра
        formatted_description =
format_register_description(register_description)

        # Сохранение описания регистра в файл формата .h
        output_file_path = 'output.h'
        save_text_as_h(formatted_description, output_file_path)
        messagebox.showinfo('Massege', f'Преобразование завершено!')
        os.system("notepad output.h")


    #Интерфейс
    window = Tk() #Создаём окно приложения.
    window.title("Converter")
    window.geometry('350x150')

    frame = Frame(
```

```
    window, #Обязательный параметр, который указывает окно для размещения Frame.
    padx=1, #Задаём отступ по горизонтали.
    pady=1 #Задаём отступ по вертикали.)
  frame.pack(expand=True) #Позиционирование виджета окне.

  book_lb = Label(
    frame,
    text="Введите имя файла  "
  )
  book_lb.grid(row=1, column=1, sticky=E)

  first_lb = Label(
    frame,
    text="Введите начальную страницу  "
  )
  first_lb.grid(row=3, column=1, sticky=E)

  last_lb = Label(
    frame,
    text="Введите конечную страницу  "
  )
  last_lb.grid(row=5, column=1, sticky=E)

  bookname_tf = Entry(
    frame,
  )
  bookname_tf.grid(row=1, column=2)
```

```
firstpg_tf = Entry(
  frame,
)
firstpg_tf.grid(row=3, column=2)
lastpg_tf = Entry(
  frame,
)
lastpg_tf.grid(row=5, column=2)

prob_lb = Label(
  frame,
)
prob_lb.grid(row=6, column=2)

cal_btn = Button(
  frame, #Заготовка с настроенными отступами.
  text='Преобразовать', #Надпись на кнопке.
  font=("Arial", 10),
  background="#ABCDEF",
  command=parsing
)
cal_btn.grid(row=8, column=2,ipadx=3, sticky=SE)


window.mainloop()
```