Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Вычислительная техника
Кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В.
подпись инициалы,

Непомнящий
фамилия

« ____» _____ 2020 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 *«Информатика и вычислительная техника»*
код и наименование направления

Среда разработки бортовых сетей на интерфейсе CAN
тема

Руководитель	_____ подпись, дата	доцент, канд. техн. наук должность, ученая степень	В. Г. Середкин инициалы,фамилия
Выпускник	_____ подпись, дата		Н. Е. Зайцев инициалы,фамилия
Нормоконтролер	_____ подпись, дата	доцент, канд. техн. наук должность, ученая степень	В. Г. Середкин инициалы,фамилия

Красноярск 2020

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
_____ О.В. Непомнящий
подпись инициалы, фамилия
«_____»_____2020 г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

в форме бакалаврской работы
бакалаврской работы, дипломного проекта, дипломной работы, магистерской диссертации

Студенту Зайцеву Николаю Евгеньевичу

фамилия, имя, отчество

Группа КИ16-08Б Направление (специальность) 09.03.01.01

номер код

Информатика и вычислительная техника

наименование

Тема выпускной квалификационной работы Среда разработки бортовых сетей на интерфейсе CAN

Утверждена приказом по университету № _____ от _____

Руководитель ВКР В. Г. Середкин, доцент, канд. техн. наук

инициалы, фамилия, должность, ученое звание и место работы

Исходные данные для ВКР Существующие аналоги систем подобного класса на отечественном и мировом рынке

Перечень разделов ВКР 1. Анализ задания на ВКР. Изучение архитектурных особенностей интерфейса CAN и процессов взаимодействия микроконтроллеров в структуре CAN-сетей; 2. Анализ существующих программных продуктов разработки ПО для CAN-сетей и формирование требований к среде разработки и ее компонентам; 3. Выбор микроконтроллеров в качестве целевых и описание их программных (регистровых) моделей; 4. Создание графической среды разработки компонентов для CAN-сетей; 5. Отладка и тестирование компонентов для сетей CAN в предлагаемой среде разработки; 6. Заключение.

Перечень графического материала 1. Презентационный материал с этапами выполнения ВКР; 2. Видеоматериал с демонстрацией работоспособности предложенной графической среды разработки бортовых и промышленных CAN-сетей

Руководитель ВКР _____ В. Г. Середкин

подпись инициалы и фамилия

Задание принял к исполнению _____ Н. Е. Зайцев

подпись, инициалы и фамилия студента

«24» июня 2020 г

РЕФЕРАТ

Выпускная квалификационная работа по теме «Среда разработки бортовых сетей на интерфейсе CAN» содержит 54 страницы, 15 иллюстраций, 5 таблиц, 1 приложение и 30 использованных источников.

СРЕДА РАЗРАБОТКИ, IDE, CAN, ГРАФИЧЕСКИЙ ЯЗЫК ПРОГРАММИРОВАНИЯ, STM32

Цель работы: исследование архитектур CAN-сетей и разработка графического интерфейса IDE как инструмента, упрощающего проектирование CAN-сетей бортового и промышленного назначения на микроконтроллерной основе

При выполнении данной работы были изучены архитектурные особенности интерфейса CAN и процессов взаимодействия микроконтроллеров в структуре CAN- сетей, проанализированы существующие программные продукты разработки для CAN-сетей, выбраны микроконтроллеры в качестве целевых.

В результате была разработана IDE, основанная на графическом языке программирования и состоящая из двух основных модулей: графических интерфейс и генератор кода на языке С. Среда разработки была протестирована на нескольких наборах тестовых данных.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В связи с техническим прогрессом и увеличением требований к скорости обработки информации, все чаще применяется параллельная обработка. В частности, становятся популярными распределенные системы управления, где каждый блок отвечает за управление отданной ему части системы. Реализуются такие системы чаще всего на основе интерфейса CAN, который обеспечивает надежный и быстрый обмен данными между независимыми устройствами.

Возможности таких систем достаточно широки, однако по тем или иным причинам производители либо искусственно ограничивают эти возможности, либо даже не предусматривают их. Известные производители электроники заняли нишу производства CAN-сетей, стали практически монополистами, тем самым лишив возможности так или иначе дополнять свои системы тем, что необходимо потребителю.

Целью данной работы является разработка IDE для упрощенного создания CAN-сетей на основе известных программируемых микроконтроллеров, что даст возможность как создавать с нуля бортовые сети, так и дополнять уже существующие.

Данная среда разработки должна позволить специалистам различных областей программировать микроконтроллеры с интерфейсом CAN для дальнейшего их объединения в сеть.

1 Анализ задания на ВКР. Изучение архитектурных особенностей интерфейса CAN и процессов взаимодействия микроконтроллеров в структуре CAN-сетей.

Согласно заданию на ВКР, необходимо исследовать архитектуру CAN-сетей и разработать графическую IDE как инструмент, упрощающей проектирование CAN-сетей бортового и промышленного назначения на микроконтроллерной основе. Этот раздел содержит описание CAN интерфейса, его основные характеристики, возможности и основные области применения.

1.1 Основные характеристики протокола

CAN (Controller Area Network) — это последовательный протокол связи с эффективной поддержкой распределения контроля в реальном времени и очень высоким уровнем безопасности. Основное назначение: организация передачи информации в сложных условиях, таких как среды с высоким уровнем различного рода помех. Этот протокол передачи применяется в автомобильной электронике, машинных устройствах управления, датчиках при передаче информации со скоростями до 1 Мбит/сек.

Протокол CAN разрабатывался для распределенного управления, отсюда вытекают некоторые его особенности, в числе которых:

- приоритет сообщения;
- гарантированные времена ожидания;
- гибкость конфигурации;
- групповой приѐм с временной синхронизацией;
- система непротиворечивости данных;
- multimaster;

- обнаружение и сигнализация ошибок;
- автоматическая ретрансляция разрушенных сообщений;
- различие между временными ошибками и постоянными отказами узлов и автономное отключение дефектных узлов. [1]

1.2 Информационная маршрутизация

В CAN нет никакой информации относительно конфигурации сети (например, адреса узла). Это имеет несколько важных следствий:

- Гибкость системы – узел может быть добавлен в CAN-сеть, без каких-либо изменений в программном или аппаратном обеспечении, какого-либо узла в сети;
- Маршрутизация сообщений – содержание сообщения определяется идентификатором. Идентификатор не указывает адреса сообщения, а описывает значение данных так, чтобы все узлы сети были способны решить фильтрацией сообщений, нужны им эти данные или нет;
- Передача группе – как следует из фильтрации сообщений, любое число узлов может одновременно получать и реагировать на одно и тоже сообщение;
- Непротиворечивость данных – внутри сети CAN гарантировано, что сообщение принято всеми узлами или ни одним узлом;
- Скорость передачи информации в CAN-сети может быть различной для каждой сети. Однако в каждой конкретной сети скорость передачи информации фиксирована;
- Идентификатор и RTR-бит определяют статический приоритет сообщения в течение доступа к шине;

- Посылая кадр удаленного запроса данных, узел может потребовать данные от другого узла. Кадр данных и кадр удаленного запроса данных должны иметь одинаковый идентификатор;
- Когда шина свободна, любой узел может начать передачу сообщения. Если два или больше узла начинают передавать сообщения в одно и тоже время, конфликт при доступе к шине будет решен поразрядным арбитражем используя идентификатор и RTR - бит. Механизм арбитража гарантирует, что ни время, ни информация не будут потеряны. Если кадр данных и кадр удаленного запроса данных начинают передаваться в одно время, то кадр данных имеет более высокий приоритет, чем кадр удаленного запроса данных. В течение арбитража каждый передатчик сравнивает уровень переданного бита с уровнем, считываемым с шины. Если эти уровни одинаковы, узел может продолжать посылать данные дальше. Если был послан уровень лог. '1' (recessive), а с шины считан уровень лог. '0' (dominant), то узел теряет право дальнейшей передачи данных и должен прекратить посылку данных на шину;
- Чтобы достичь высокой безопасности передачи данных, приняты мощные меры нахождения ошибок, сигнализации ошибок и самотестирование в каждом CAN – узле;
- Для обнаружения ошибок приняты такие меры, как текущий контроль (передатчики сравнивают уровни битов, которые переданы, с уровнями на Сигнализация ошибки и время восстановления. Разрушенные сообщения помечаются узлом, обнаружившим ошибку. Такие сообщения прерываются и будут переданы снова. Узлы CAN способны отличить временные ошибки от постоянных отказов. Дефектные узлы будут отключены;
- Все приѐмники проверяют непротиворечивость принимаемого сообщения и подтверждают непротиворечивое сообщение;
- Чтобы уменьшить потребляемую мощность системы, узел CAN может быть переведен в режим "сна". Режим "сна" заканчивается при любом

действии на шине или внутреннем состоянии системы. При пробуждении запускается внутренняя синхронизация, канальный уровень ждѐт стабилизации генератора системы, а затем будет ожидать самосинхронизации к действиям на шине (синхронизация к действиям на шине заканчивается после принятия последовательности 11 битов с лог. "1"). Для пробуждения узла из режима покоя может использоваться некоторое сообщение пробуждения со специальным идентификатором.

1.3 Подключение устройств к CAN-шине

Линия связи по протоколу CAN – это шина, к которой может быть подключён ряд узлов. Количество узлов не имеет никакого теоретического предела. Фактически количество узлов будет ограничено временами задержек и/или электрической нагрузкой на линии шины. Способ, которым выполнена шина, не установлен в данной спецификации. Например, это может быть одиночный провод (+земля), два дифференциальных провода, оптическое стекловолокно.

Шина может принимать одно из дополняющих друг друга значений: "dominant" и "recessive". В случае одновременной подачи "dominant" бита и "recessive" бита, возникающее в результате значение шины будет "dominant". (Прим. переводчика: далее считается что "recessive" = лог. "1", а "dominant" = "0").

1.4 Применение CAN

CAN-сети используются в системах управления, с целью уменьшения совокупной длины проводов, необходимых для подключения датчиков и исполнительных устройств. В частности, с появлением интерфейса CAN в

автомобилестроении, функций у систем управления становится все больше, при том, что сложность укладки проводов – все ниже.

Ввиду различных требований к частоте передаваемых сигналов, объему информации и к резервированию данных, в пределах автомобиля шины CAN подразделяются на три вида:

- Шина CAN силового агрегата (быстрая шина), позволяющая передавать информацию со скоростью) 500 кбит/с. Она служит для связи между блоками управления на линии двигателя и трансмиссии.
- Шина CAN системы "Комфорт" (медленная шина), позволяющая передавать информацию со скоростью 100 кбит/с. Она служит для связи между блоками управления, входящими в систему "Комфорт".
- Шина данных CAN информационно-командной системы (медленная шина), позволяющая передавать данные со скоростью 100 кбит/с. Она служит для связи между различными обслуживающими системами, например, радиосистемой, телефонной и навигационной системами.

Непосредственная связь шин CAN силового агрегата и системы "Комфорт" невозможна ввиду различного уровня напряжений и различных нагрузочных сопротивлений. При этом имеют место еще различные скорости передачи данных по шинам, что исключает обработку разнотипных сигналов в одном устройстве. Поэтому необходимо применение преобразователя (моста) для связи между шинами. Таким преобразователем является так называемый межсетевой интерфейс (Gateway). На автомобиле такой интерфейс может быть встроен в комбинацию приборов или в блок управления бортовой сетью, а также выделен в отдельный прибор межсетевой связи. [3]

1.5 Выводы к разделу

Подводя итог данного раздела, можно сказать, что на сегодняшний день интерфейс CAN применяется повсеместно, хоть и имеет достаточно сложную архитектуру, что увеличивает так называемый порог вхождения в эту область. Имеет смысл в рамках данной работы автоматизировать наиболее сложные процессы, связанные с разработкой ПО для компонентов сети CAN.

2 Анализ существующих программных продуктов разработки ПО для CAN-сетей и формирование требований к среде разработки и ее компонентам.

В данном разделе кратко описаны существующие на рынке технологии для работы с устройствами на CAN-шине, затем, на основании анализа, сформированы требования к IDE.

2.1 Существующие продукты разработки программного обеспечения для CAN-сетей

Поскольку на сегодняшний день интерфейс CAN применяется в каждом современном автомобиле, на рынке все чаще появляются специальные устройства для выявления сбоев в работе системы, а также другой важной информации, касающейся эксплуатации и обслуживания автомобиля. Такие устройства, как правило, запрограммированы на конкретную марку автомобиля, а интерфейс программного обеспечения к ним позволяет выполнить ограниченный список функций. Так как данная работа направлена на разработку универсальных устройств и сетей, рассматривать эти приборы не имеет смысла.

На сегодняшний день в мире существует только один продукт, нацеленный на программирование универсальных can-контроллеров. Этот продукт состоит из среды разработки CannyLab и серии программируемых контроллеров CANNY.

CannyLab — интегрированная среда разработки программного обеспечения для программируемых логических контроллеров CANNY 7, CANNY 7.2 duo, CANNY 5.3 pico, CANNY 5.3 MD 1, CANNY 5.2, CANNY 5 nano, CANNY 5.2 duo, CANNY 3 tiny.

Интегрированная среда разработки CannyLab является инструментом разработки прикладных программ с использованием языка функциональных блоковых диаграмм CFD и предназначена для написания, отладки и записи программ во внутреннюю память программируемых логических контроллеров CANNY 7, CANNY 7.2 duo, CANNY 5.3 pico, CANNY 5.3 MD 1, CANNY 5.2, CANNY 5 nano, CANNY 5.2 duo, CANNY 3 tiny.

Интерфейс пользователя CannyLab состоит из единственного окна - главного окна программы.

Главное окно программы делится на несколько областей:

- заголовок окна;
- панель главного меню;
- панели инструментов редактора;
- панель функциональных блоков;
- рабочая область;
- строка состояния. [2]

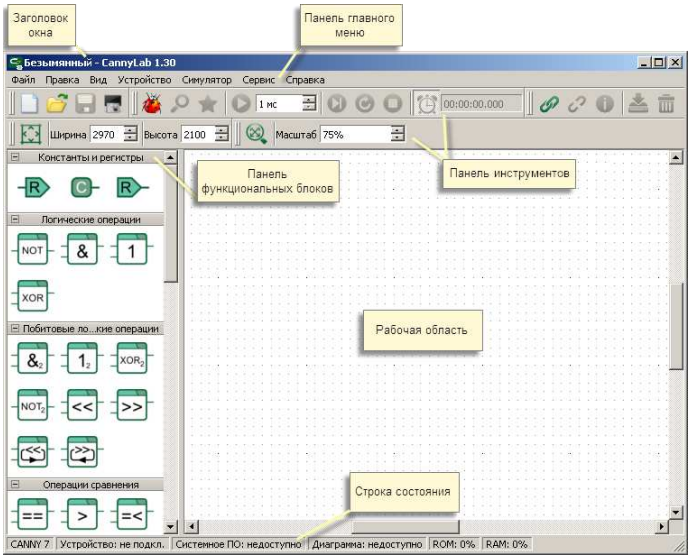


Рисунок 1 – Главное окно CannyLab

Преимуществами CannyLab являются удобный интерфейс, простой графический язык программирования, а также возможность симуляции работы контроллера.

Недостатками данного продукта являются:

- Высокая стоимость;
- Отсутствие гибкости (возможность программировать только контроллеры серии CANNY);
- Отсутствие возможности симуляции и отладки сети из микроконтроллеров.

2.2 Выводы и формирование требований к среде разработки

Интерфейс CAN служит для распределенной обработки информации, в частности, для управления сложными системами посредством обработки сигналов от датчиков и подачи управляющих сигналов на исполнительные устройства. Достаточно редко от блока управления на шине CAN требуются более сложные операции. Исходя из этого, в рамках ВКР достаточно ограничить функционал будущей IDE простыми логическими и

арифметическими операциями, возможностью отправлять сообщения и реагировать на заранее прописанные сообщения.

Опираясь на аналог, принято решение создавать IDE под графический язык программирования, поскольку он более прост, по сравнению с текстовыми, и может быть интуитивно понятен для людей, плохо знакомых с программированием.

3 Выбор микроконтроллеров в качестве целевых и описание их программных (регистровых) моделей.

Данный раздел посвящен формированию перечня микроконтроллеров, на которые нацелена среда разработки, а также описанию их архитектуры.

3.1 Выбор целевых устройств

При выборе целевых устройств для программирования нужно опираться на поддержку интерфейса CAN. Это главный критерий, поскольку цель данной работы – создать универсальную среду разработки. В процессе изучения подходящих устройств выяснилось, что большинство таких устройств в качестве CAN-адаптера используют контроллеры MCP2510 и MCP2515. Это значит, что принципиальной разницы в настройке и программировании интерфейса CAN у разных семейств микроконтроллеров нет. Исходя из этого, имеет смысл выбрать наиболее крупное семейство микроконтроллеров, поддерживающих CAN, с наименьшей стоимостью. Сравним популярные серии микроконтроллеров на количество устройств с CAN адаптером. В следующей таблице [4] сравниваются микроконтроллеры 8081, PIC, AVR и ARM.

Таблица 1 – основное различия между микроконтроллерами AVR, ARM, 8051 и PIC

	8051	PIC	AVR	ARM
Разрядность	8 бит	8/16/32 бит	8/32 бит	32 бит, иногда 64 бит
Интерфейсы	UART, USART,SPI,I2C	PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S	UART, USART, SPI, I2C, иногда CAN, USB, Ethernet	UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI, IrDA
Скорость	12 тактов на инструкцию	4 такта на инструкцию	1 такт на инструкцию	1 такт на инструкцию
Память	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM
Шинная архитектура	CLSC	Частично RISC	RISC	RISC
Архитектура памяти	Фон-неймановская	Гарвардская	Модифицированная	Модифицированная гарвардская
Энергопотребление	Среднее	Низкое	Низкое	Низкое
Семейства	Вариации 8051	PIC16,PIC17, PIC18, PIC24, PIC32	Tiny, Atmega, Xmega, спец. AVR	ARMv4,5,6,7 ...
Производители	NXP, Atmel, Silicon Labs, Dallas, Cyprus, Infineon ...	Microchip	Atmel (Microchip)	Apple, Nvidia, Qualcomm, Samsung Electronics, TI ...
Стоимость	Низкая	Средняя	Средняя	Низкая
Популярные микроконтроллеры	AT89C51, P89v51	PIC18fXX8, PIC16f88X, PIC32MXX	Atmega8, 16, 32; вариации для Arduino	LPC2148, ARM Cortex-M0, ARM Cortex-M3, ARM Cortex-M7

Как видно из таблицы 1, наиболее подходящие микроконтроллеры – это микроконтроллеры с ядром ARM. Разработаны такие микроконтроллеры на базе архитектуры RISC компанией Advanced RISC Machines (ARM). Применяются в потребительских электронных устройствах, таких как смартфоны, планшеты, мультимедийные проигрыватели и другие мобильные устройства. Из-за сокращенного набора команд им требуется меньше транзисторов, что позволяет уменьшить размер матрицы интегральной схемы.

Процессоры ARM с меньшими размерами уменьшают сложность проектирования и сокращают энергопотребление, что делает их пригодными для более миниатюрных устройств.

Наиболее популярные ARM микроконтроллеры среди разработчиков – STM32, разработанные европейской компанией STMicroelectronics. Серия микроконтроллеров STM32, основанная на ядре промышленного стандарта, поставляется с широким выбором инструментов и программного обеспечения для поддержки разработки проектов, что делает это семейство продуктов идеальным как для небольших проектов, так и для сквозных платформ [5]. Исходя из этого, дальнейшие действия по разработке направлены на микроконтроллеры STM32.

3.2 Описание целевых устройств

Для дальнейшего проектирования и разработки ПО, направленного на создание программ для устройств серии STM32, необходимо проанализировать устройства на предмет взаимодействия с памятью, портами ввода-вывода и периферией, а также ознакомиться с языками и способами программирования устройств.

3.2.1 Краткое описание серии микроконтроллеров STM32

К особенностям микроконтроллеров серии STM32 можно отнести объем Flash памяти до 1-го Мб, Dual Bank, а также собственный интерфейс взаимодействия с памятью программ Flash memory interface (FLITF), который защищает память от чтения и записи иными устройствами.

За счет технологии Dual Bank в устройствах STM32 реализована возможность read-while-write, это означает, что пока в одной области памяти выполняется программа записи или чтения, другая область может быть доступна для другой операции, не дожидаясь освобождения первой области.

Помимо этих особенностей, микроконтроллеры STM32 имеют возможность отладки, режимы сна, остановки и ожидания и до 80 портов ввода-вывода.

3.2.2 Периферийные устройства микроконтроллеров серии STM32

Микроконтроллеры STM32 имеют до 9 коммуникационных интерфейсов, среди которых I^2C, SPI, USART, CAN, LIN, USB2.0 и другие. В рамках данной работы достаточно рассмотреть взаимодействие устройств через интерфейс CAN.

Для настройки интерфейса CAN на устройствах серии STM32 необходимо предварительно включить тактирование специализированного контроллера шины CAN, выставить тактовую частоту, инициализировать шину и настроить фильтр входящих сообщений. Если контроллер шины принимает сообщение, соответствующее фильтру, то от контроллера шины на микроконтроллер приходит прерывание, которое обрабатывается в зависимости от заранее прописанных команд в flash-памяти устройства.

Для отправки сообщения по CAN шине необходимо сформировать заголовок сообщения и подготовить несколько байт данных. Затем, с помощью определенной функции этот набор данных отправляется контроллеру шины CAN, который, в свою очередь, выберет подходящий момент и отправит сообщение

3.2.3 Программирование и прошивка микроконтроллеров STM32

Для перезаписи программ в микроконтроллерах серии STM32 предусмотрен специальный контроллер записи и сброса FPEC (Flash program and erase controller). FPEC содержит 12 регистров по 32 бита:

● Регистр контроля доступа Flash (FLASH_ACR), общий для банка 1 и банка 2

● Регистр ключей FPEC (FLASH_KEYR), для банка 1

● Опциональный регистр байтовых ключей (FLASH_OPTKEYR), общий для банка 1 и банка 2

● Регистр управления Flash (FLASH_CR), выделенный для банка 1 и программирования дополнительных байтов

● Регистр состояния Flash (FLASH_SR), предназначенный для банка 1

● Регистр Flash-адреса (FLASH_AR), выделенный для банка 1

● Опциональный байтовый регистр (FLASH_OBR), общий для банков 1 и 2

● Регистр защиты от записи (FLASH_WRPR), общий для банков 1 и 2

● Регистр ключа FPEC2 (FLASH_KEYR2), выделенный для банка 2

● Регистр управления Flash2 (FLASH_CR2), выделенный для банка 2

● Регистр состояния Flash2 (FLASH_SR2), выделенный для банка 2

● Флэш-адрес регистра2 (FLASH_AR2), выделенный банку 2

Перепрограммирование не будет блокировать процессор, пока процессор не даст доступ к Flash памяти.

Память программ состоит из 32-х битных ячеек и может хранить в себе как алгоритм работы, так и некоторые постоянные данные. Модуль памяти программ размещен по особому адресу, который отличается для каждого устройства. После каждого сброса памяти программ блок FPEC защищен и регистр FLASH_CR недоступен. Чтобы разблокировать запись, в регистр FLASH_KEYR должна быть записана битовая последовательность разблокировки. Любая неправильная последовательность блокирует блок FPEC и регистр FLASH_CR до следующего сброса.

3.3 Выводы к разделу

В качестве целевых устройств для разработки IDE были выбраны микроконтроллеры серии STM32, поскольку это самая доступная и популярная линейка устройств с контроллером интерфейса CAN. В рамках ВКР требуется добавить к IDE возможность подключать и взаимодействовать с портами цифрового ввода-вывода, а также с контроллером шины CAN.

4 Создание графической среды разработки компонентов для CAN-сетей

Этот раздел посвящен разработке пользовательского интерфейса, алгоритмов преобразования графических схем в прошивку поддерживаемых устройств, а также описанию нового графического языка, предназначенного для программирования в данной среде

4.1 Описание графического языка для программирования в данной среде разработки

Любой IDE требуется язык программирования, на который она будет направлена. Как правило, набор языков, доступных пользователю, определяется их популярностью и предметной областью. Так, например, у большинства IDE для микроконтроллеров доступны языки Assembler и C. В данном случае, в целях упрощения для конечного пользователя, выбран только один язык – графический. Для этого языка необходимо описать все доступные типы и структуры данных, которые могут понадобиться пользователю при программировании устройств.

4.1.1 Типы данных

Первое, что необходимо любому языку программирования – это константы и переменные. Стандартным набором являются типы «целое число», «дробное число», «символ», «строка», «логический (1 бит)».

Если опираться на существующие графические языки программирования, то константа может быть представлена одним блоком с выходной ножкой справа, а переменная может быть таким же блоком, но в зависимости от доступа к ней, ножка может быть слева или справа. На основе этого, принято решение представлять константы и переменные в виде прямоугольников фиксированного размера с цветом, соответствующим своему типу. Внутри этого блока, если это константа, – значение, соответствующее типу, а в случае с переменной – ее имя. Представление констант и переменных представлены на рисунках 2 и 3 соответственно.

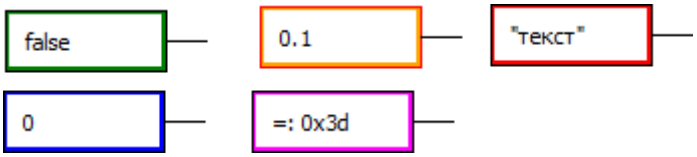


Рисунок 2 – представление констант

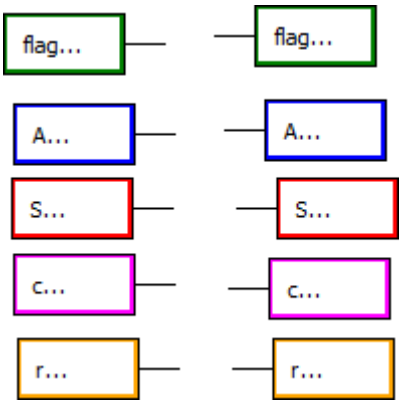


Рисунок 3 – представление переменных

С точки зрения данных, каждый из этих блоков принимает на вход или отдает соответствующий ему тип.

Далее, то, без чего не может существовать язык программирования микроконтроллеров – описание портов ввода-вывода. В данном случае достаточно описать цифровые порты ввода-вывода и блоки приема и отправки сообщений по CAN. Цифровой порт ввода по функционалу напоминает константу, соответственно и выглядеть он должен похожим образом. В центре блока написано имя порта согласно документации на устройство. Выходной порт должен выглядеть похожим образом, но с оговоркой на то, что он должен принимать значение, а не отдавать, соответственно ножка должна быть расположена слева. Блоки приема и отправки сообщения, поскольку функционально то такие же порты, но для другого типа данных, можно представить подобным образом, но обозначив другим цветом для наглядности. Также, целесообразно поместить текст ожидаемого сообщения внутри блока приема. Обозначения портов ввода и вывода, а также блоков приема и отправки сообщений по CAN представлены на рисунке 4.

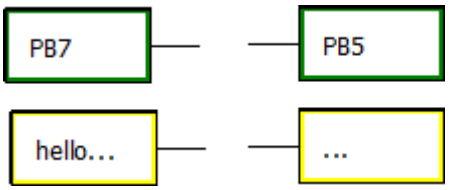


Рисунок 4 – Обозначение портов

Чтобы описать тип данных, отдаваемый или принимаемый этими блоками, нужно задуматься о дальнейшей структуре описания графического кода. На данном этапе принято решение дать цифровым портам логический тип, поскольку их размерности совпадают. Блоку отправки сообщения по CAN – строковый тип, так как далее необходимо сообщение преобразовать в байтовую последовательность, а блоку приема, так как с точки зрения работы алгоритма прием сообщения является прерыванием, разумно присвоить логический тип данных, где правда соответствует успешному приему сообщения.

4.1.2 Операции с данными

При описании текстового языка можно описывать условно бесконечное количество операций над переменными, но в случае с графическим языком это количество ограничивается как минимум размерами рабочего пространства. Также, количество операций ограничивается предметной областью. Конечному пользователю достаточно дать возможность обрабатывать сигналы с портов, для чего ему понадобятся логические операции И, И-НЕ, ИЛИ, ИЛИ-НЕ, XOR и НЕ. Их представление на рисунке 5.

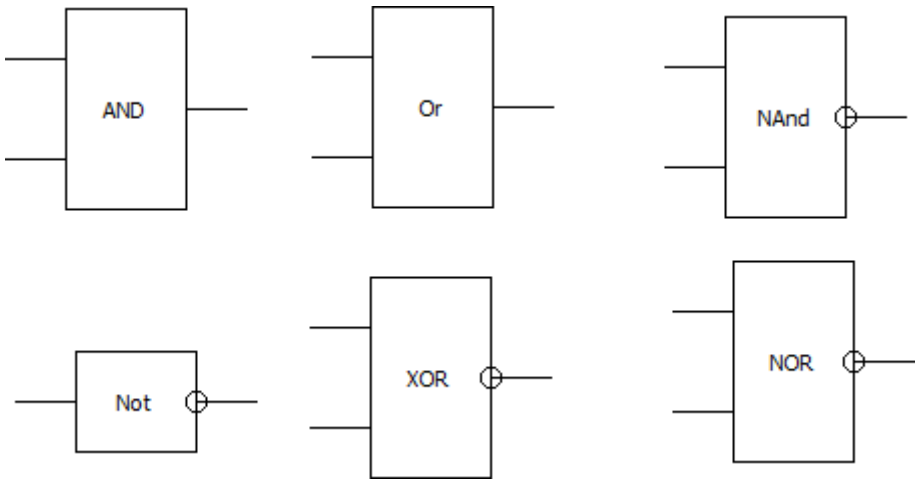


Рисунок 5 – Представление логических операций

Возможны ситуации, когда требуется большее количество входов. Решение такой проблемы на примере блока И представлено на рисунке 6.

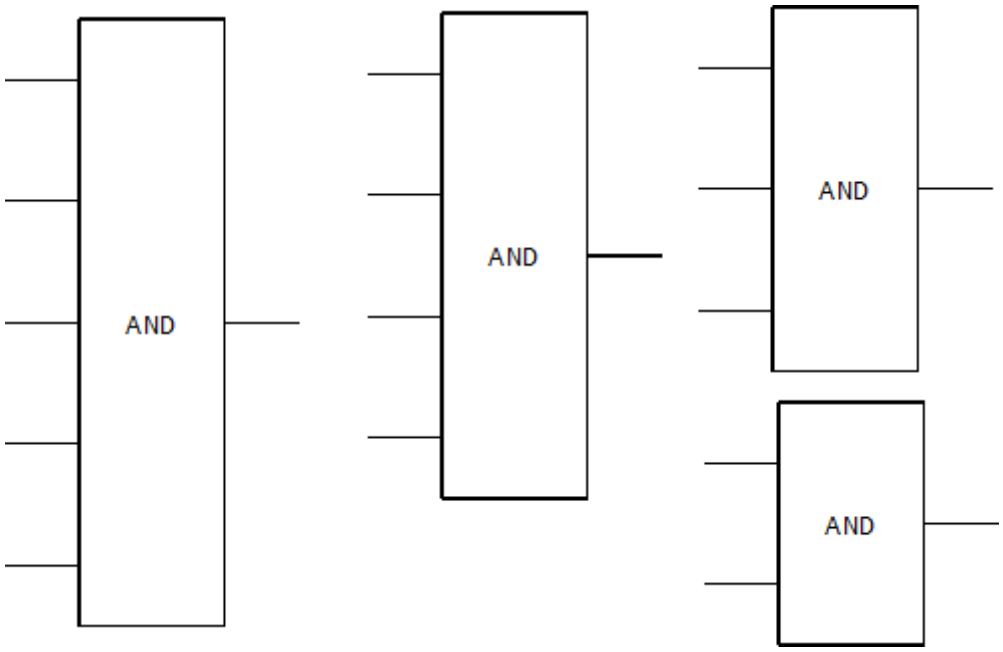


Рисунок 6 – представление операции И с 2, 3, 4 и 5 входами

Помимо логических операций, пользователю нужно дать привычные ему – арифметические операции. Достаточно ограничиться следующим набором:

- Сумма
- Разность
- Произведение
- Частное
- Остаток от деления
- Округление (к ближайшему, к меньшему и к большему)

Операции принимают на вход числовые данные и отдают соответствующие. Представление данных операций н рисунке 7

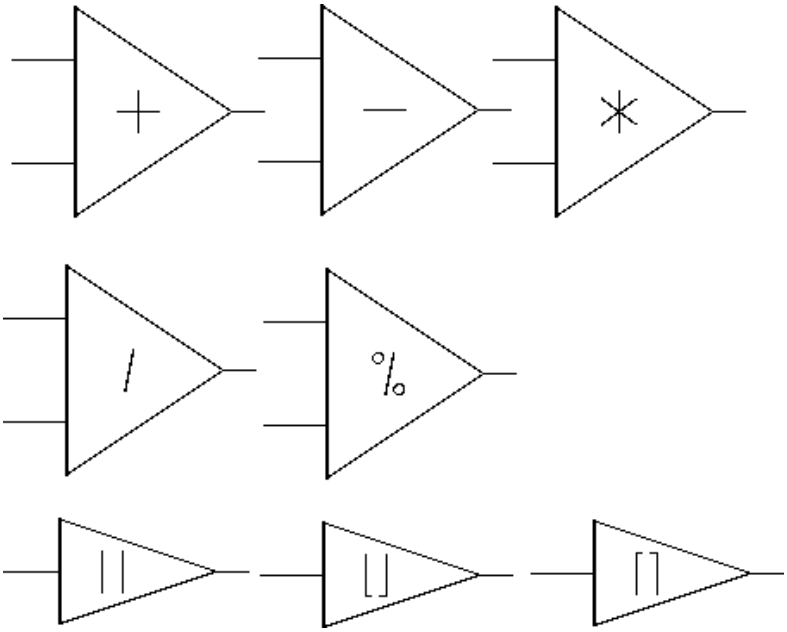


Рисунок 7 – представление арифметических операций

Далее, можно описать также привычные пользователю логические операции над числами: больше, меньше, равно и подобные. Такие операции принимают на вход все числовые типы данных и возвращают логический. Как принято во многих графических языках, первый операнд располагается сверху, второй – снизу. Обозначение представлено на рисунке 8.

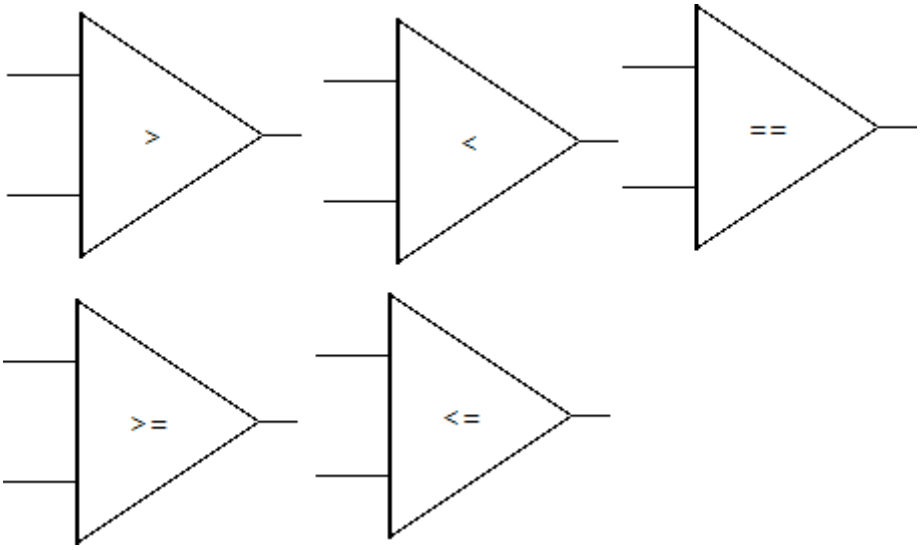
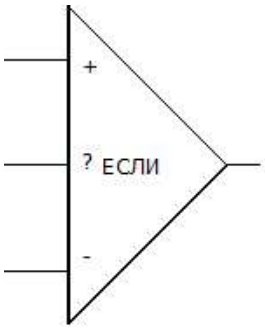
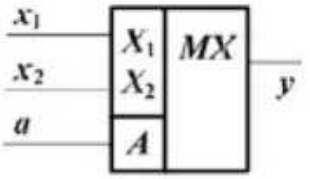


Рисунок 8 – обозначение логических операций над числами

4.1.3 Ветвление

То, без чего язык программирования превращается в обычную комбинационную схему с усложнениями – это ветвление. Наиболее простая для понимания конструкция – ЕСЛИ-ТО-ИНАЧЕ. Эту конструкцию можно представить в виде, привычном для людей, знакомых с электросхемами, – в виде мультиплексора, где в качестве адреса выступает какое-либо условие, возвращающее логический тип, а в качестве данных может быть любой доступный тип данных. Обозначение блока если и его функциональное соответствие мультиплексору представлены на рисунке 9.





Мультиплексор	ЕСЛИ
A	?
X1	-
X2	+
Y	выход

Рисунок 9 – блок если

Привычной для всех языков программирования структуры ЦИКЛ не будет в рамках данной работы, поскольку в рамках предметной области в использовании других циклов нет необходимости.

4.2 Определение схемы взаимодействия с пользователем

На этом этапе необходимо определиться со схемой взаимодействия между приложением и программируемым микроконтроллером. Поскольку микроконтроллеры STM32 имеют достаточно много механизмов защиты от перезаписи памяти программ, то для прошивки целесообразно использовать уже готовые решения. Наиболее популярные программаторы для микроконтроллеров STM32 – это ST-Link, J-Link и Black Magic Probe. В рамках этой работы от программатора не требуется дополнительных возможностей, поэтому достаточно использовать самый дешевый. На сегодняшний день – это ST_Link v2. Причем существует множество различных модификаций данного программатора, в числе которых программаторы от фирмы STM (рис. 10) и от китайских производителей (рис. 11)

Рисунок 10 – Программаторы ST-Link от STMicroelectronics

Рисунок 11 – Программатор ST-Link от китайской фирмы XM

Определившись с программатором, нужно выбрать программное обеспечение для прошивки устройств. Для микроконтроллеров STM32 фирма STMicroelectronics предлагает несколько бесплатных утилит:

- STM32CubeProg
- STLink-Utility

- STVP (STM32)
- Flasher-STM32 – для загрузки через UART
- DfuSe – для загрузки через USB

Для выбора наиболее подходящего, рассмотрим каждую из этих утилит. Первая – STM32CubeProg. Cube Programmer, помимо прошивки позволяет выполнять следующие операции:

- Читать и сохранять Flash-память микроконтроллера, если она не защищена от чтения;
- Очищать как Flash-память, так и внешнюю память микроконтроллера. Причем очищать можно как отдельные блоки, так и все целиком. Под внешней памятью подразумевается память, подключенная посредством SPI, FMC, FSMC, QSPI, OCTOSPI и другими интерфейсами, но работать с ней можно только при помощи внешнего загрузчика (в комплекте уже идет набор из внешних загрузчиков под наиболее популярные платы);
- Устанавливать или снимать флаги запрета чтения Flash иначе чем из программного кода, выполняемого микроконтроллером, снимать или устанавливать флаги запрета записи в конкретные сектора на Flash, и некоторые другие функции;

STM32CubeProg помимо ST-Link может подключаться по UART, USB и OTA (обновление по воздуху). Для начала рассмотрим наиболее интересующий нас вариант подключения посредством ST-Link. При подключении ST-Link V2 к компьютеру и выборе варианта загрузки через ST-Link в окне ST-LINK Configuration отображаются некоторые настройки, которые могут повлиять на способность подключения.

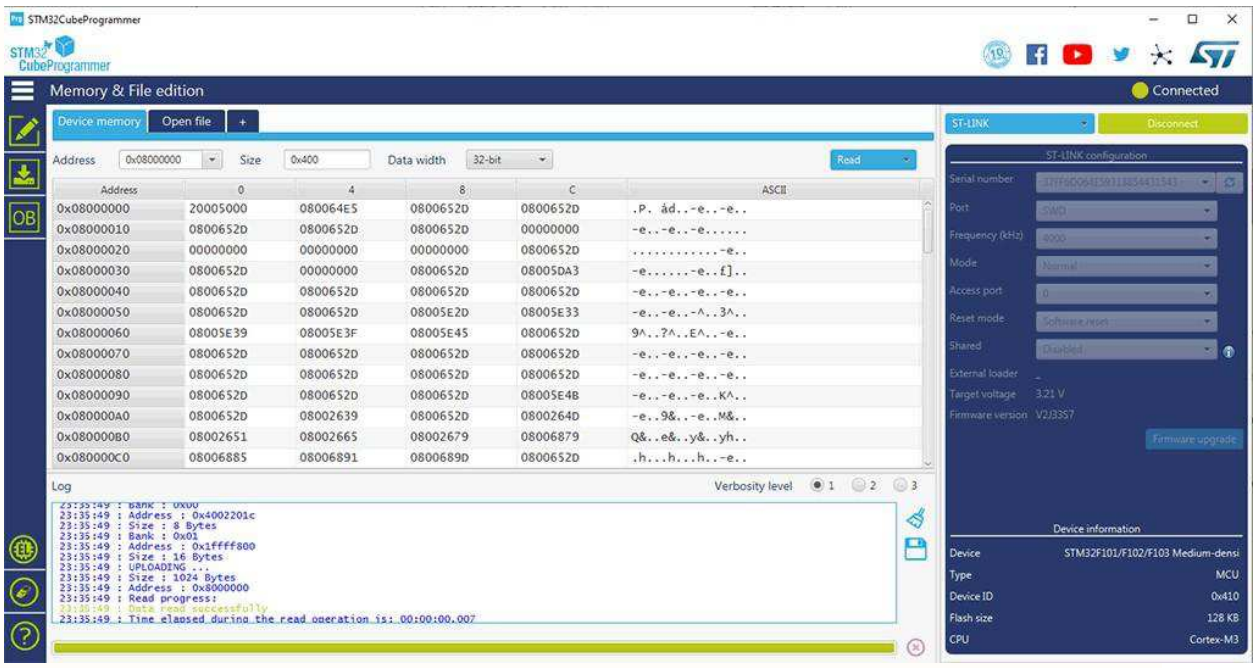


Рисунок 12 – Окно конфигурации STM32CubeProg

Следующая утилита – ST-Link Utility, наиболее популярна среди разработчиков. По набору возможностей и функций ST-Link Utility практически полностью повторяет STM32CubeProg за исключением двух моментов. Utility работает под Windows, Utility может подключаться к платам только посредством SWD или JTAG. Все остальное, начиная от настройки Option Bytes и заканчивая внешними загрузчиками с командной строкой повторяет STM32CubeProg. Для прошивки микроконтроллера с помощью данной утилиты необходимо подключиться к устройству, очистить Flash-память, выбрать файл прошивки firmware.bin и нажать Start. Через незначительный промежуток времени микроконтроллер перейдет к выполнению новой программы.

Несмотря на то, что ST-Link Utility полностью повторяет часть функций STM32CubeProg, многие разработчики используют именно ее, так как считают, что этот продукт более надежен и прост в использовании.

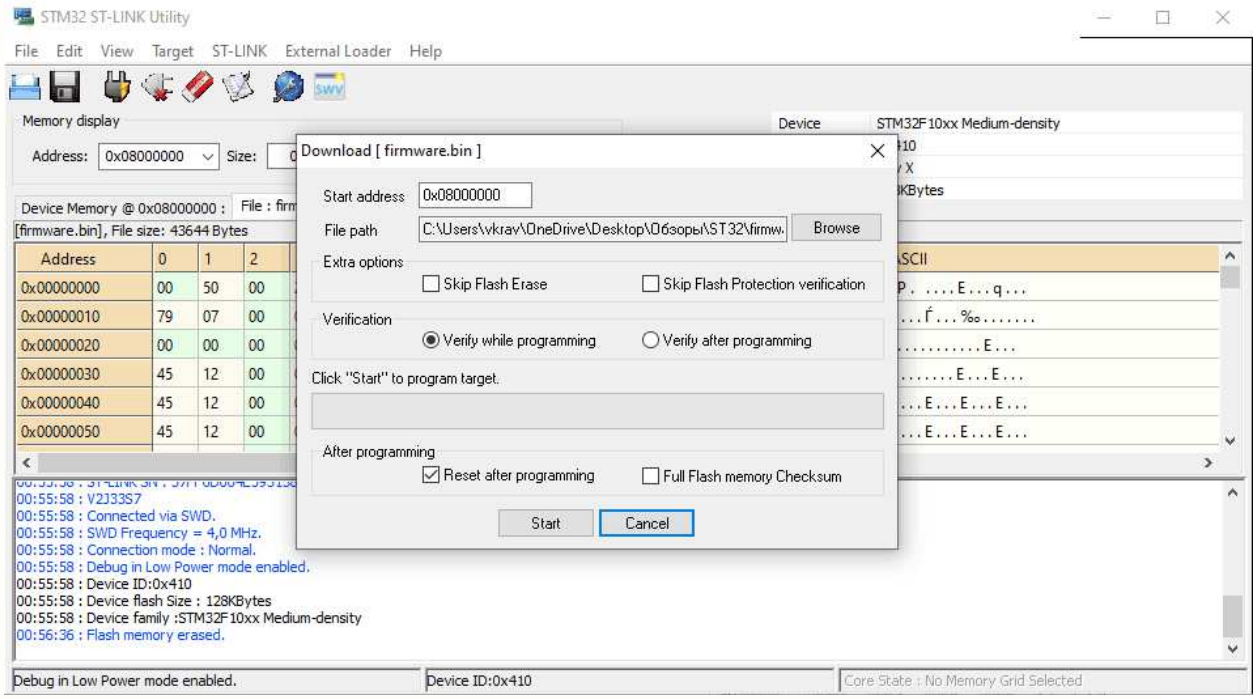


Рисунок 13 – Окно прошивки ST-Link Utility

Утилита STVP (ST Visual Programmer) работает под управлением Windows и поддерживает не только STM32, но и чипы семейства STM8. STVP помимо самой утилиты для прошивки включает еще среду разработки STVD под язык Assembler. Исходя из этого, эта утилита может работать только с файлами hex и s19.

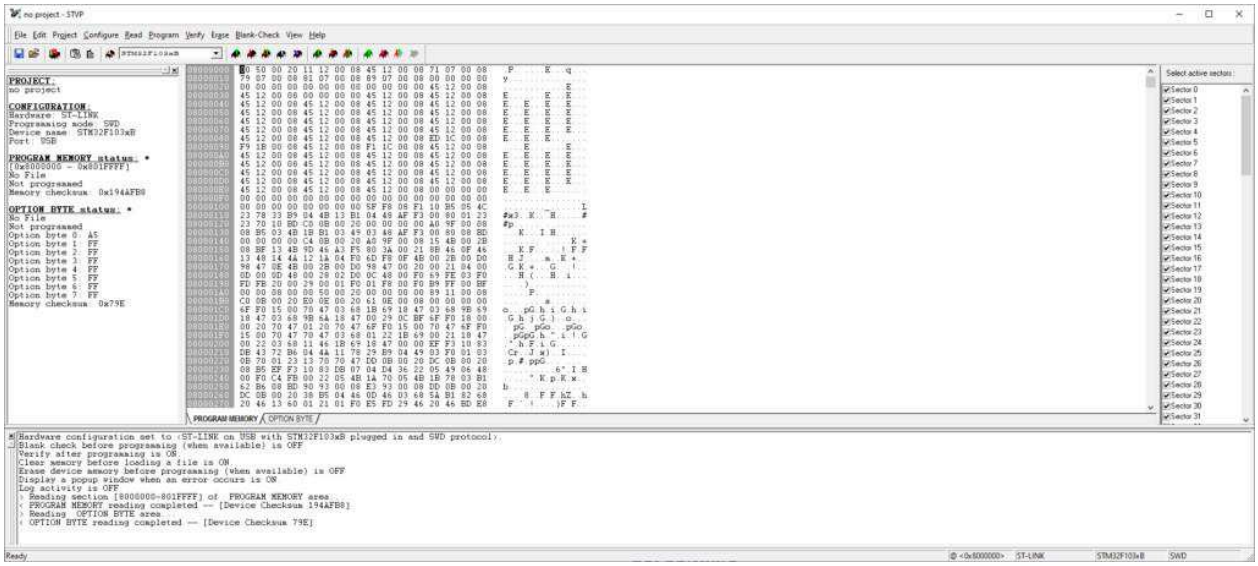


Рисунок 14 – Утилита STVP

Flasher-STM32 (ранее STM32 Flash loader demonstrator) используется для загрузки простых программ. Загружает прошивки только через UART (он же COM-порт) и работает исключительно под Windows. Среди всех утилит данная обладает наиболее простым интерфейсом, поскольку в ней присутствуют только необходимые функции, такие как очистка бита защиты flash-памяти чипа от чтения, управление защитой записи и чтения секторов встроенной flash-памяти, возможность загрузки и выгрузки прошивки и содержимого flash-памяти чипа и редактирование Option Bytes (настроек микроконтроллера).

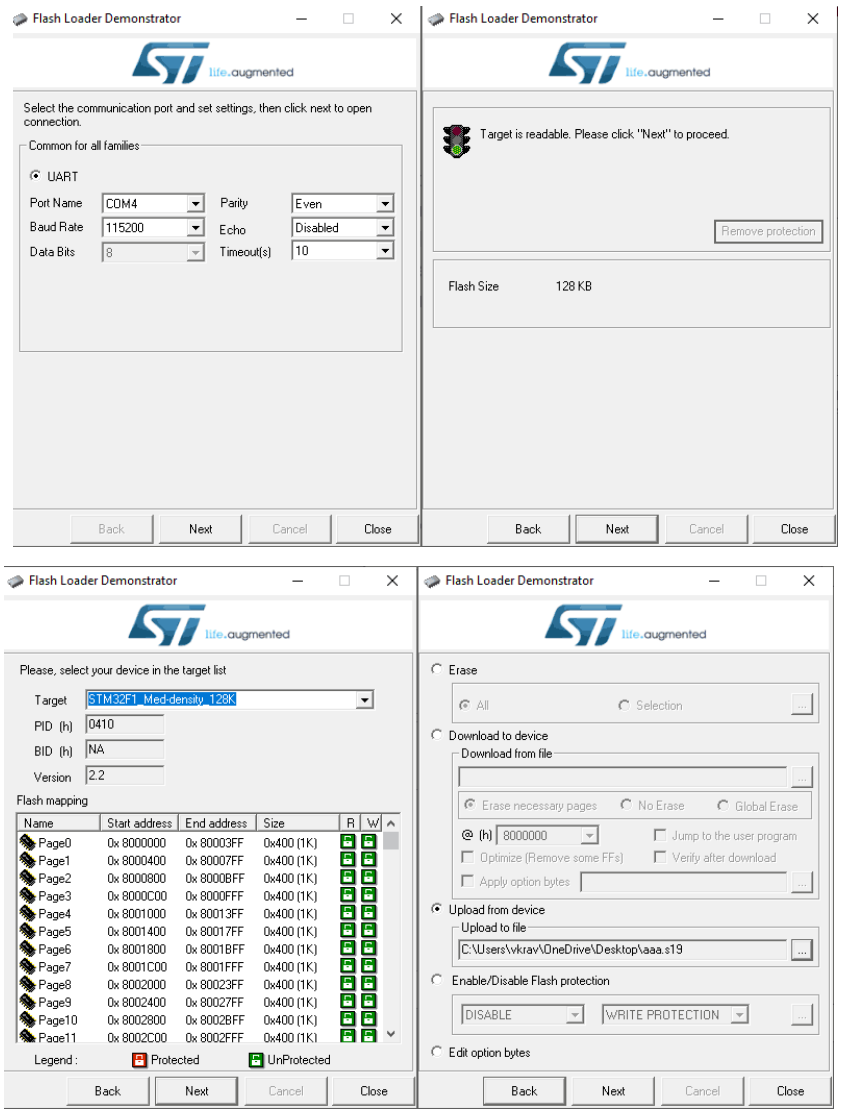


Рисунок 15 – Окно программы Flasher-STM32

Последняя утилита –DfuSe (Device Firmware Upgrade STMicroelectronics Extension). Методика DFU была придумана для еще более простого обновления прошивок. Программа работает только через подключение по USB. Доступны библиотеки для написания собственных приложений по загрузке прошивок в микроконтроллеры. DfuSe принимает на борт прошивки только в формате DFU. Формат содержит не только саму прошивку как таковую, но еще и манифест (информацию о плате). Если подключенная плата не поддерживается, то прошивки не произойдет. Тем самым существенно упрощается вся процедура прошивки и уменьшается риск порчи платы при прошивке, но нужна поддержка режима DFU на самой плате, что несколько сокращает популярность программатора. Файлы DFU создаются при помощи комплектной утилиты DFU File Manager. Она может как запаковать прошивку в файл, так и распаковать.

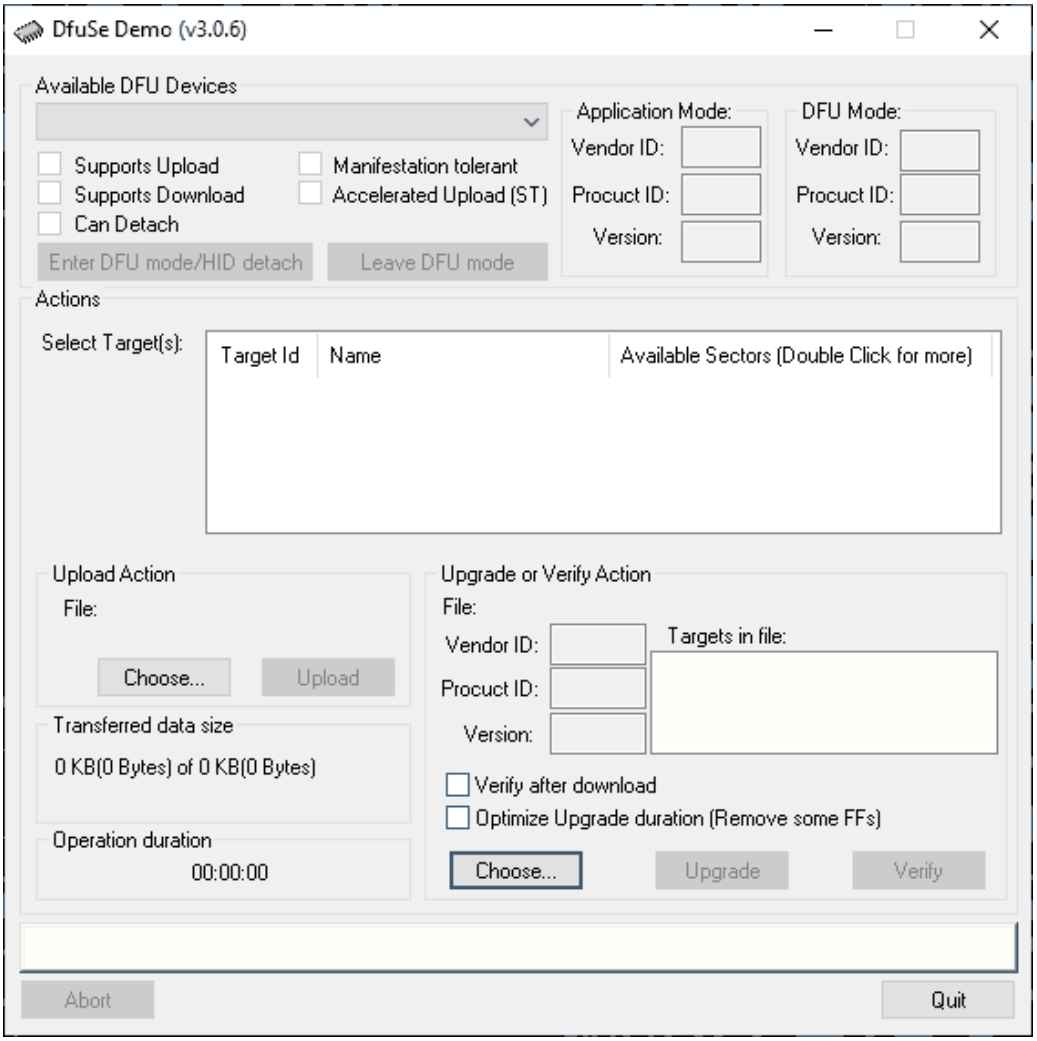


Рисунок 16 – Окно программы DfuSe

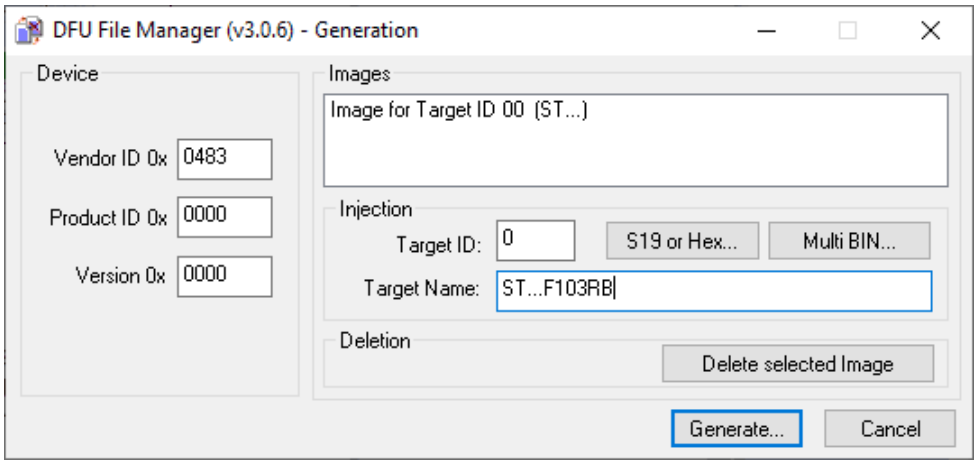


Рисунок 17 – DFU File Manager

Поскольку в данной работе преследуется цель упрощения разработки, то и утилита программирования должна быть упрощенной. Исходя из того, что самый популярный и удобный на сегодняшний день интерфейс взаимодействия – это USB, то и утилита программирования должна работать по USB, а наиболее простая среди таковых – DfuSe.

Уровнем выше должен находиться преобразователь кода, он же компилятор. Этот уровень должен быть разбит на несколько, так как пользователь видит и создает программу на графическом языке программирования, и прежде, чем преобразовывать эту программу в прошивку, будет правильнее сгенерировать ее код на языке C или Assembler. Сразу же встает вопрос выбора промежуточного языка. Существуют готовые компиляторы как для языка C, так и для Assembler, поэтому функционально разницы нет. Однако, практично использовать в качестве промежуточного языка более высокоуровневый язык, поскольку таким образом сокращаются риски различных ошибок. Следовательно, в качестве промежуточного языка нужно использовать C, а компилятор к нему – GNU C.

Исходя из описанного выше, упрощенная структурная схема будущего приложения представлена на рисунке 18.

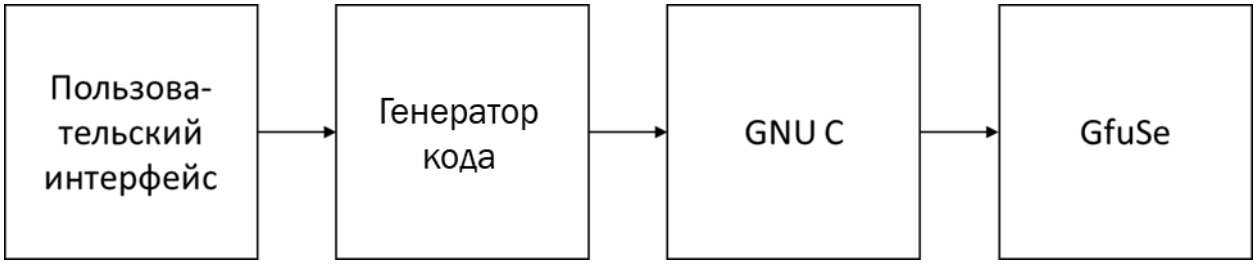


Рисунок 18 – Структурная схема приложения

4.3 Описание алгоритма

С пользовательской точки зрения, создание прошивки начинается с создания нового файла и составления графической схемы взаимодействия портов ввода-вывода, интерфейсов и структур данных. После того, как будет готова схема приложения, пользователь должен запустить сборку. На этом этапе генератор в паре с компилятором подготовят файлы для прошивки и сообщат об этом в главном окне приложения. Далее, пользователь должен будет из главного окна запустить утилиту GfuSe и уже с ее помощью прошить устройство. Блок-схема алгоритма представлена на рисунке 19.

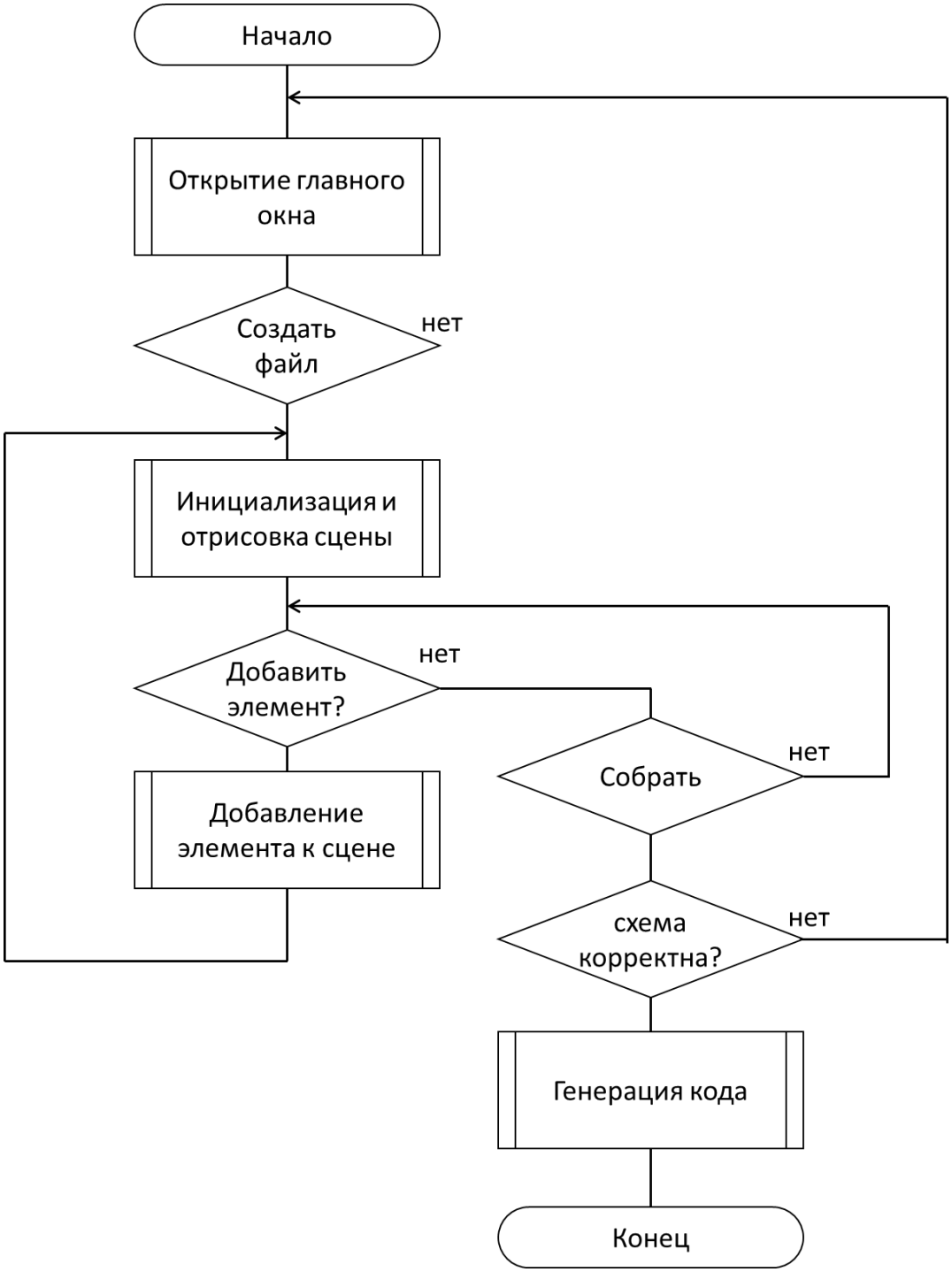


Рисунок 19 – Блок-схема алгоритма работы приложения

4.4 Описание классов

Поскольку за компиляцию и прошивку отвечают уже готовые решения, в рамках данной работы нужно описать 2 модуля: графический интерфейс и

генератор кода на языке С. Графический интерфейс должен предоставлять пользователю возможность размещать графические объекты, описанные в блоке 2.3, на рабочей области экрана в произвольном порядке и создавать взаимосвязи между ними. Генератор, в свою очередь, должен получать от графического интерфейса массив данных, по которым он и будет генерировать код на языке С.

4.4.1 Пользовательский интерфейс

Исходя из выше указанного, для реализации пользовательского интерфейса нужны инструменты для работы с графикой. Наиболее популярные из них – Canvas и QGraphicsScene. Отличаются они количеством надстроек. Canvas – пустая графическая сцена с минимальным набором команд на подобие setPen() и moveTo(). Его плюс заключается в том, что библиотеки для работы с Canvas существуют во всех современных языках программирования и разметки. Минусом является то, что даже для отрисовки одного прямоугольника придется выполнить порядка 6 операций по типу «переместить курсор» и «установить кисть». Таким способом описывать интерфейс для графической среды разработки нецелесообразно. QGraphicScene является наследником QtWidget и не может работать отдельно от библиотеки Qt и QGraphicsView. Это ограничивает список доступных языков программирования, однако наиболее популярные языки поддерживают эти библиотеки. QGraphicsScene имеет ряд преимуществ перед Canvas: в первую очередь, данная библиотека имеет огромное множество методов для отрисовки, добавления, перемещения, удаления различных объектов, а QGraphicsView, созданный для отображения сцены имеет ряд возможностей, без которых сегодня не может существовать ни один

графический редактор (масштабирование, скроллинг, подгонка изображения, фильтр событий и так далее).

Наиболее популярные языки программирования, поддерживающие фреймворк Qt, это C++, Python, Ruby и PHP. PHP по понятным причинам не подходит, а среды остальных, выбор пал на Python, по двум основным причинам: наиболее удобная среда разработки PyCharm от компании JetBrains и высокий уровень программирования с большим количеством библиотек, созданных для ускорения и упрощения процесса разработки.

Определившись с основой, необходимо понять, какие классы будут необходимы в дальнейшем. Поскольку приложение затачивается под работу с графикой, то и большинство классов должны описывать графические объекты. Каждый класс должен быть унаследован от графического объекта и описывать тип или структуру данных, которые он будет визуализировать.

Согласно описанию графического языка программирования, определим классы, описывающие данные объекты.

Таблица 2 – Классы для работы с графическими объектами

Родитель	Класс	Описание	Методы
Qgraphicsitem	Constant	Класс, описывающий отображение графических объектов, соответствующих типу данных "Константы"	– init_(), boundingRect(), select(), unselect()
Constant	INTConstant	целое число	__init_(), doubleClickEvent(), paint(), getWriteInfo()
	FLTConstant	с плавающей точкой	
	BOOLConstant	логический тип	
	STRConstant	строка	
	CHRConstant	символ	
Qgraphicsitem	Item	Класс, описывающий отображение графических объектов, соответствующих стуктурам данных "Операции"	__init_(), boundingRect(), select(), unselect()

Продолжение таблицы 2

Родитель	Класс	Описание	Методы
Item	AndItem	Логическое И	__init_(), paint(), getWriteInfo()
	OrItem	Логическое ИЛИ	
	NOrItem	Логическое ИЛИ-НЕ	
	NAndItem	Логическое И-НЕ	
	NotItem	Логическое НЕ	
	XORItem	Логическое сложение	
	SUMItem	Арфметическое сложение	
	DIFItem	Арифметическая разность	
	MULTItem	Произведение	
	DIVItem	Частное	
	MODItem	Остаток от деления	
	ROUNDItem	Округление к ближайшему	
	TRUNKItem	Округление к меньшему	
	CEILItem	Округление к большему	
	BGRItem	Условие больше	
	LSRItem	Условие меньше	
	EQItem	Условие Равно	
	EQBItem	Условие Больше-равно	
	EQLItem	Условие Меньше-равно	
	IF	Мультиплексор	
Qgraphicsitem	net	Класс, описывающий отображение графических объектов, соответствующих типу данных "Линия связи"	__init_(), paint(), getWriteInfo()
Qgraphicsitem	Port	Класс, описывающий отображение графических объектов, соответствующих типу данных "Порты"	__init_(), paint(), getWriteInfo()
Port	DI_Port	Порт цифрового ввода	
	DO_Port	Порт цифрового вывода	
	CAN_Send	Порт отправки сообщения по CAN	
	CAN_Dump	Порт прерывания приема сообщения от контроллера CAN	

Окончание таблицы 2

Родитель	Класс	Описание	Методы
Qgraphicsitem	Var	Класс, описывающий отображение графических объектов, соответствующих типу данных "Переменные"	– init_(), boundingRect(), select(), unselect(), invert()
Var	StrVat	Строковая переменная	__init_(), paint(), getWriteInfo()
	ChrVar	Символьная переменная	
	IntVar	Целочисленная переменная	
	FltVar	Переменная с плавающей точкой	
	BoolVAr	Логическая переменная	

4.4.2 Генератор кода

Задача данного модуля – принимать от пользовательского интерфейса данные о заполнении графической сцены и в соответствии с ними генерировать код. На вход словарь, описывающий графическую сцену с объектами.

Следует завести методы для генерации нового файла с кодом. Последний также должен хранить код в памяти в текстовом формате отдельными блоками.

Таблица 3 – Методы для преобразования графики в код

Класс	Метод	Описание	Тип возвращаемого объекта	тип принимаемого объекта
Generator	__init___	Инициализация объекта	-	-
	search	Рекурсивный обход и составление скелета формул	Кортеж (строка, конечный объект (словарь))	Строка

Окончание таблицы 3

Класс	Метод	Описание	Тип возвращаемого объекта	тип принимаемого объекта
	findLine	Поиск линии связи на конкретном выходе объекта	Линия (словарь)	Кортеж(объект(словарь)), имя порта(строка)
	findObjByEndOfLine	Поиск объкта на обратном конце линии связи	Объект (словарь)	Линия(словарь)
	getPorts	Поиск объектов класса порт на сцене	Массив объектов	-
	getQuestPorts	Генерация опроса портов ввода	Массив строк	-
	getResult	Основной метод, генерирующий код на языке С из скелетов функций	Кортеж из массивов строк, соответствующих сгенерированному коду для каждого блока	-

4.5 Создание графического интерфейса IDE

Перед тем, как разрабатывать интерфейс, целесообразно описать диаграмму работы приложения. При запуске, приложение ждет от пользователя ввода одного из сигналов:

- Добавление элемента на сцену
- Создание нового проекта
- Сохранение проекта
- Создание глобальной переменной
- Сборка проекта

Обработка всех возможных ситуаций представлена на рисунке 20.

Рисунок 20 – диаграмма работы приложения

Первое, что необходимо сделать для создания приложения на PyQt, –это в среде QtDesigner создать разметку главного окна. Основные необходимые компоненты – QgraphicsView для главной сцены и для предварительного просмотра объекта, TreeWidget для выбора объекта и Label для отображения вспомогательной информации.

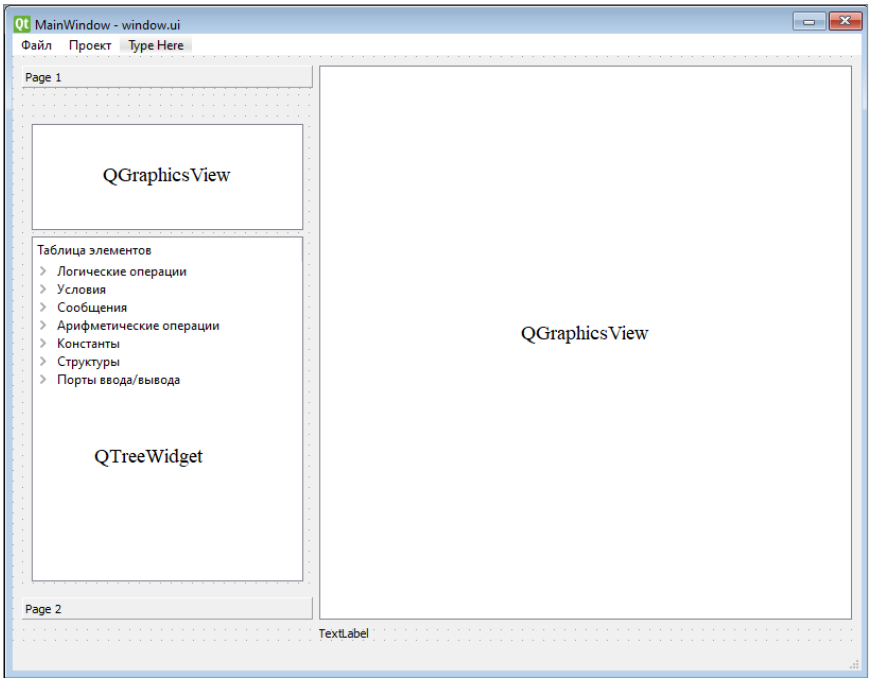


Рисунок 21 – Проектируемый интерфейс

После этого, файл .ui с помощью python библиотеки PyQt преобразуется в файл .py, в котором описывается интерфейс. Этот файл необходимо импортировать и инициализировать в файле app.py. После этого можно запустить приложение с помощью команды python app.py или же из среды PyCharm.

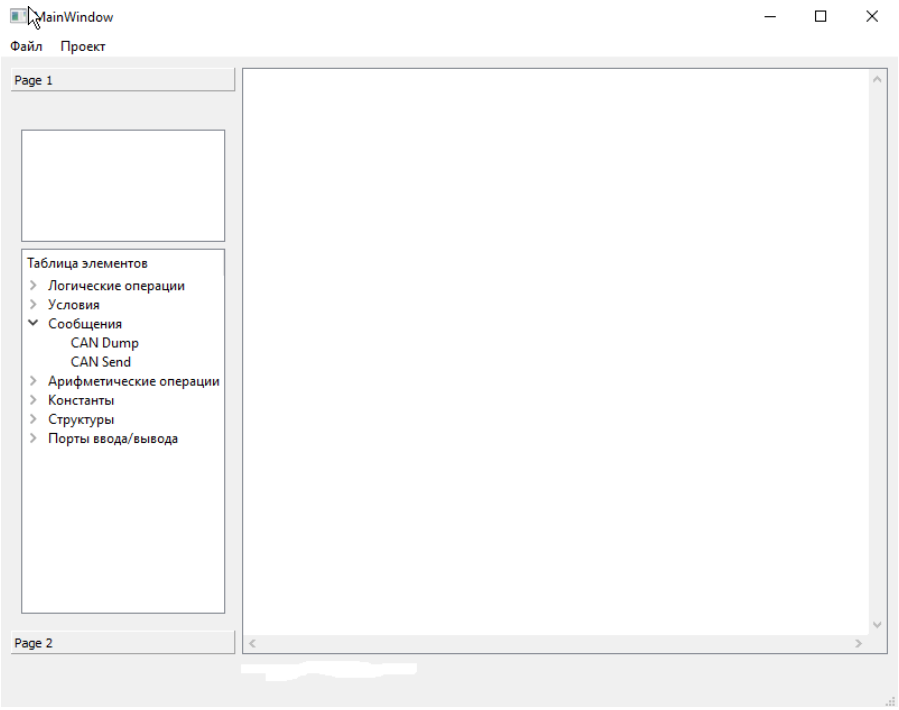


Рисунок 22 – Главное окно приложения

Далее, добавим приложению слушатели или фильтры событий на обработку перемещений мыши или нажатий. Один из таких фильтров будет отображать информацию о позиции курсора на графической сцене.

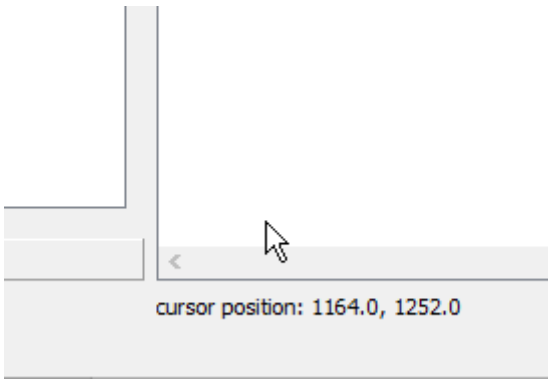


Рисунок 23 – Вывод информации о положении курсора

На следующем этапе необходимо описать в коде классы графических элементов, а также добавить фильтры для взаимодействия с пользователем. Для добавления объектов на сцену ранее был добавлен TreeWidget, в котором описаны соответствующие графическому языка программирования объекты. Добавлен обработчик события click по виджету TreeWidget, в котором определяется выбранный элемент.

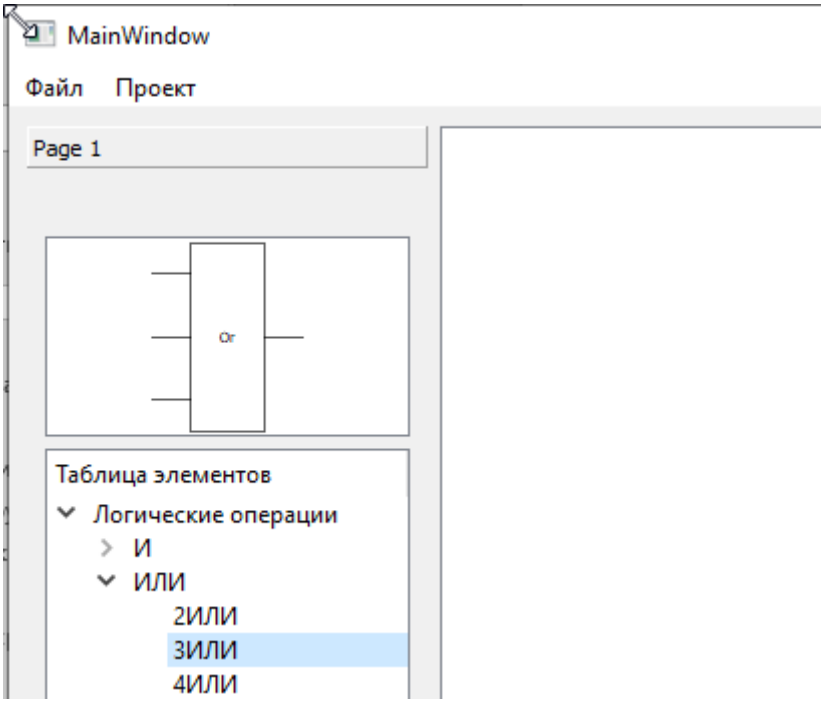


Рисунок 24 – Выбор элемента из главного окна

Далее необходимо добавить возможность добавлять элемент на основную сцену по нажатию на произвольный ее участок. Для этого в фильтре событий мыши определим позицию, тип события и если тип события соответствует клику левой кнопки мыши, добавляется выбранный ранее элемент на нулевую позицию с помощью метода QGraphicssScene.addItem(QGraphicsItem), затем перемещается на заданную позицию с помощью метода QGraphicsItem.setPos(QPointF).

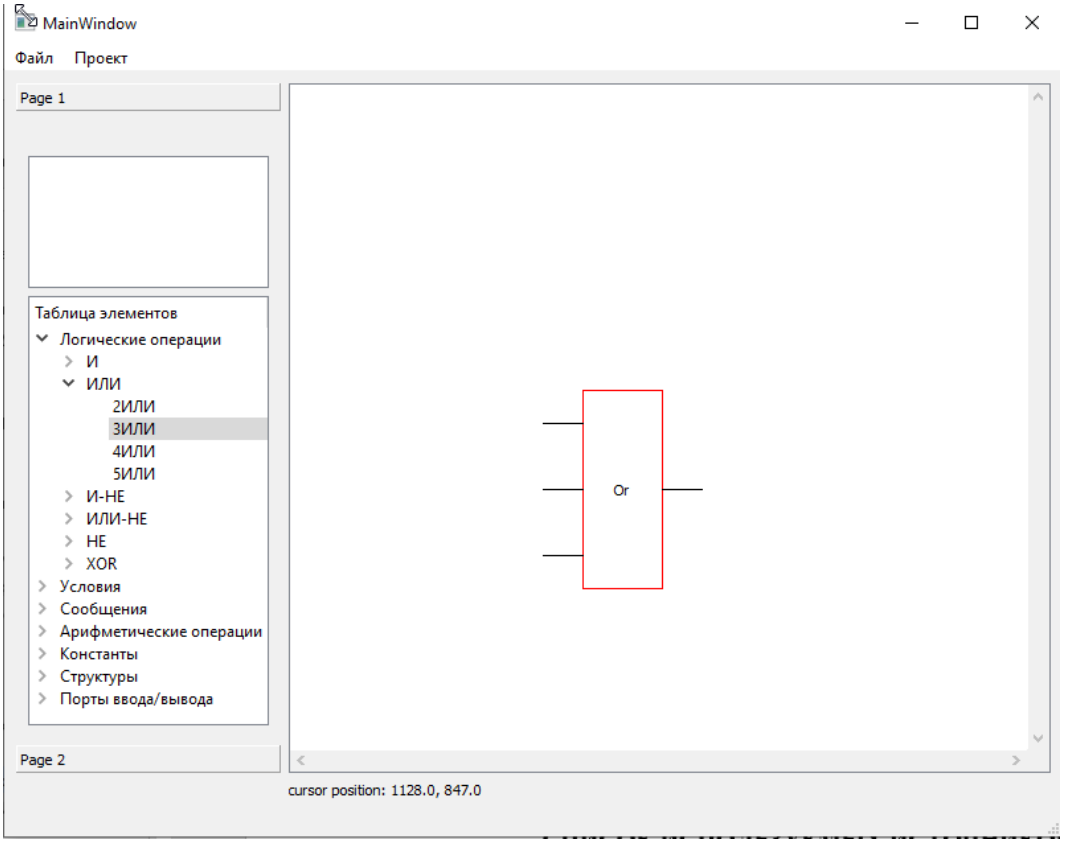


Рисунок 25 – Добавление элемента на главную сцену

Любая графическая IDE должна позволять перемещать объекты по сцене. Чтобы это сделать, необходимо с помощью фильтров событий обработать нажатие клавиши мыши, перемещение курсора и освобождение мыши. Здесь на помощь приходит ранее указанный метод setPos(). Предварительно, определяется наличие объекта в заданной позиции с помощью метода QGraphicsScene.itemAt(QPointF, QTransform).

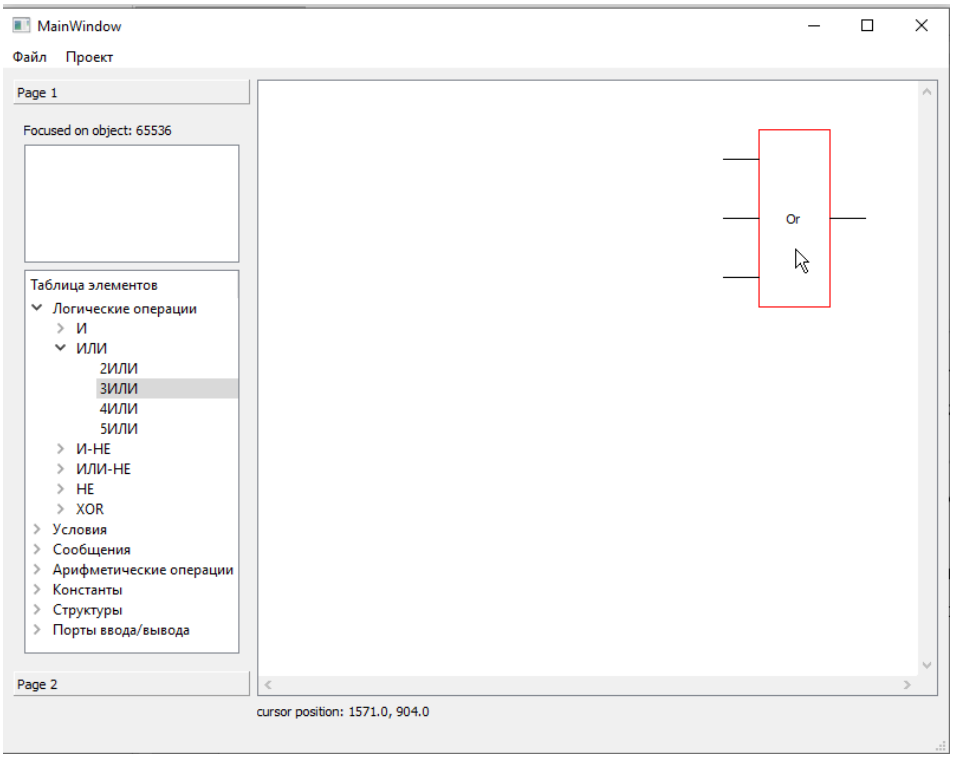


Рисунок 26 – Перемещение объекта

Также немаловажной частью редактора является масштабирование сцены. Виджет QGraphicsView позволяет отображать сцену с заданным коэффициентом масштабирования. Остается только обработать событие колесика мыши с нажатой клавишей ‘Ctrl’ и промасштабировать сцену.

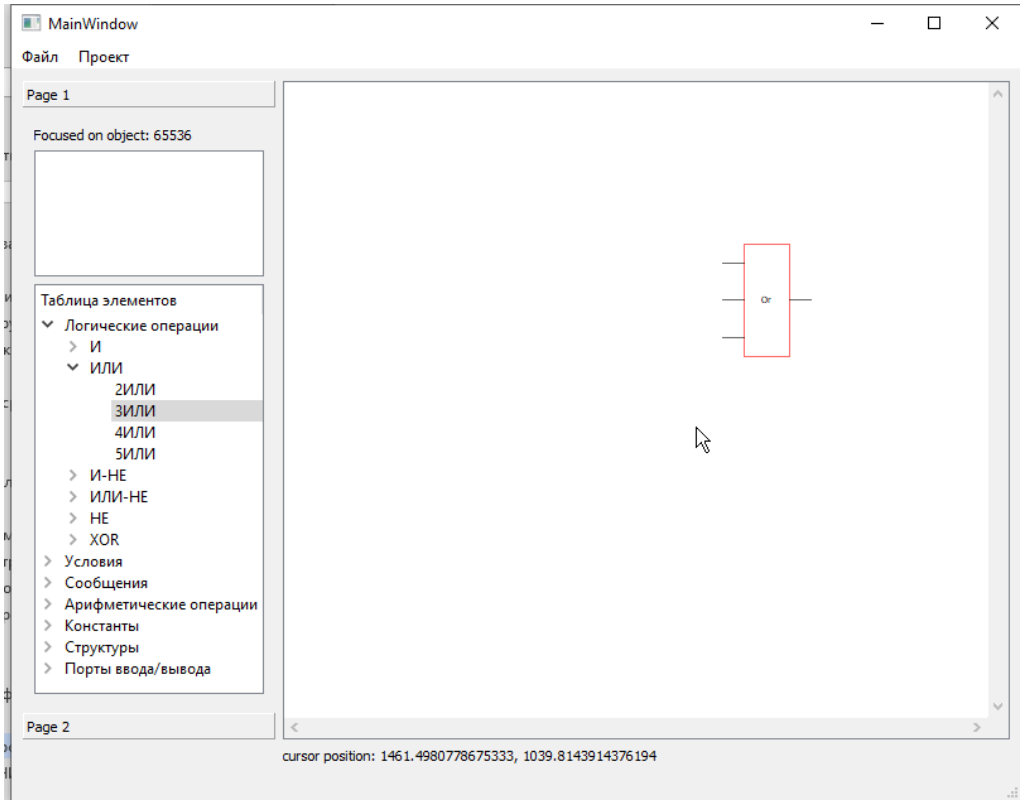


Рисунок 27 – Масштабирование сцены

Для более удобной работы со сценой необходимо обработать событие колесика мыши также и для перемещения по сцене по четырем направлениям: вверх, вниз, влево и вправо. Для перемещения вверх-вниз обработаем событие колесика мыши без дополнительных клавиш, а для перемещения влево-вправо – с клавишей ‘Shift’. Теперь удобно перемещаться по сцене с помощью общепринятых ‘жестов’.

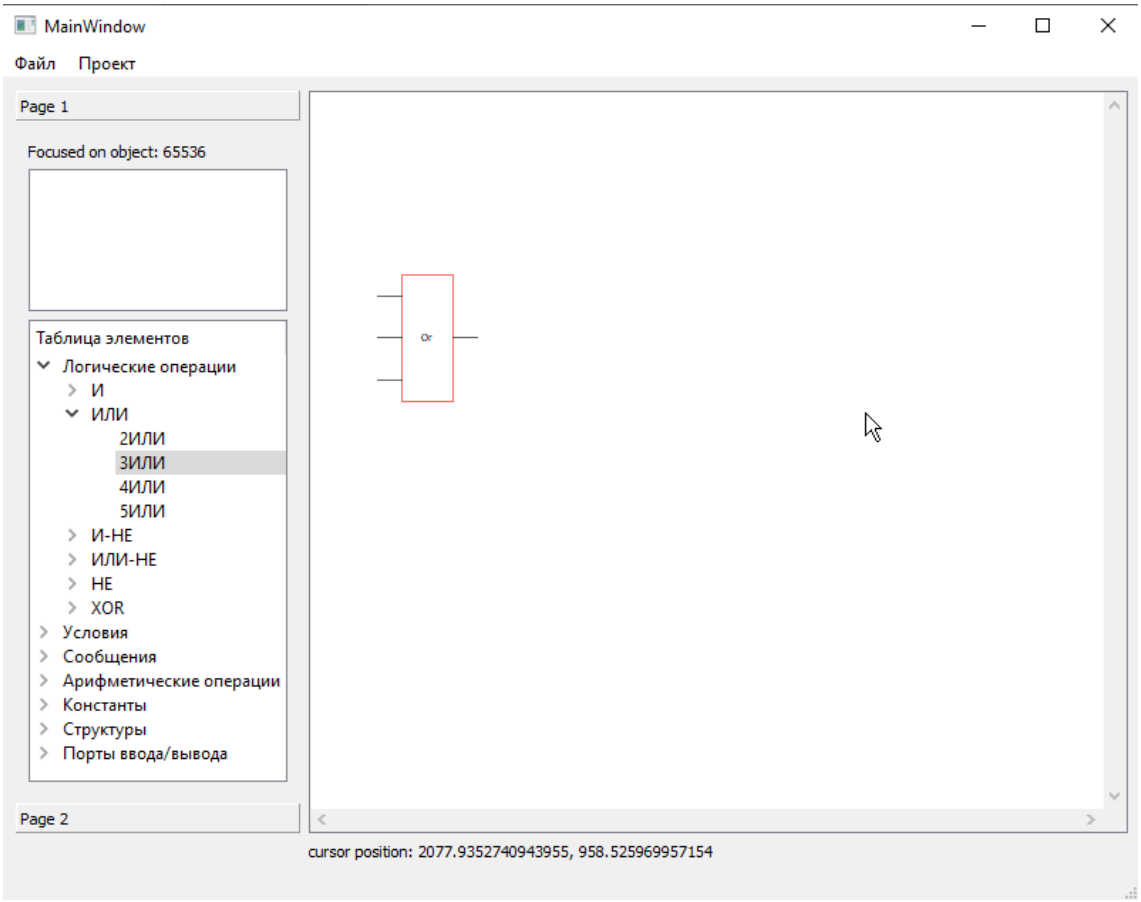


Рисунок 28 – Перемещение по сцене

После того, как разобрались со статическими объектами, необходимо обработать такие объекты как константы, порты ввода-вывода и структуры. Для изменения значения константы целесообразно использовать событие двойного нажатия по нему. Затем, в зависимости от типа константы, должно появиться диалоговое окно, в котором предлагаются допустимые значения. В случае с логической константой, достаточно при двойном нажатии переключать состояние с правды на ложь и наоборот, чтобы уменьшить количество итераций для пользователя.

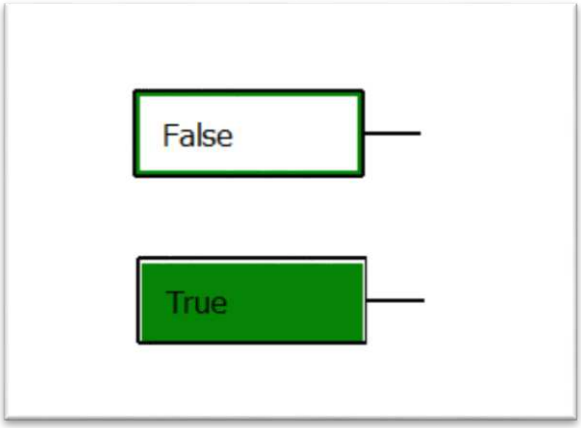


Рисунок 29 – Константы правда/ложь

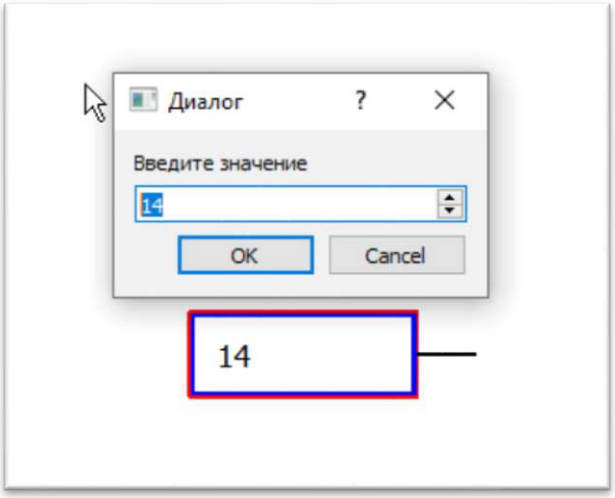


Рисунок 30 – Константа «целое число»

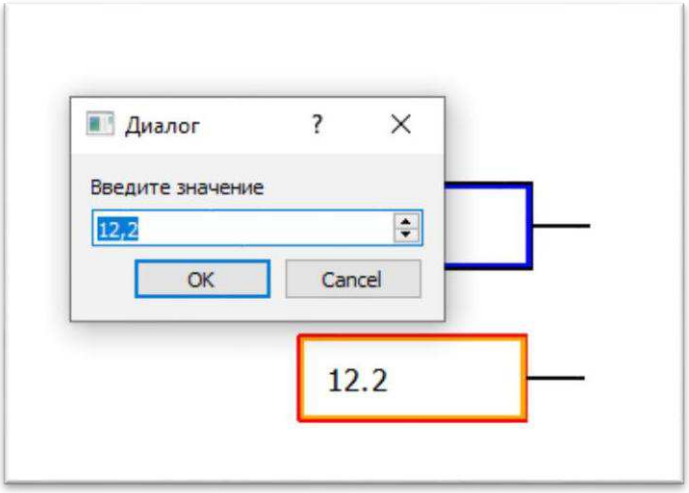


Рисунок 31 – Константа «дробное число»

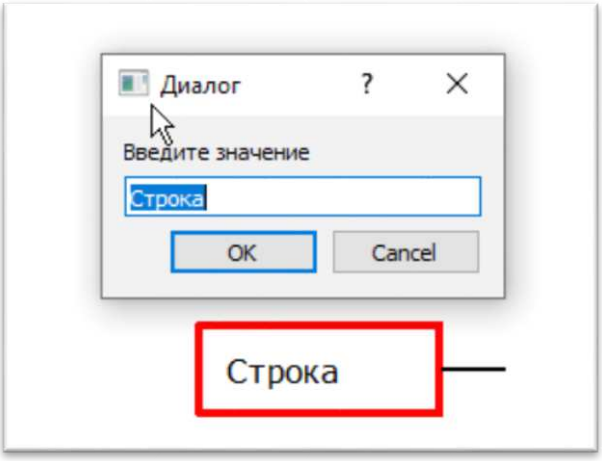


Рисунок 32 – Константа «строка»

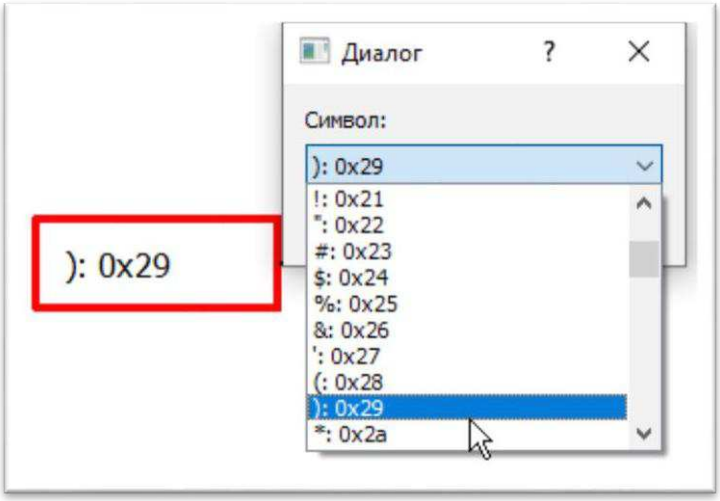


Рисунок 33 – Константа «Символ»

Для работы с портами ввода-вывода задача несколько усложняется. Здесь необходимо предусмотреть тип платы с которой происходит работа. Далее, в зависимости от этого, подгрузим УГО данной платы с доступными портами (количество доступных портов уменьшается с добавлением их на сцену, также оно зависит от типа порта).

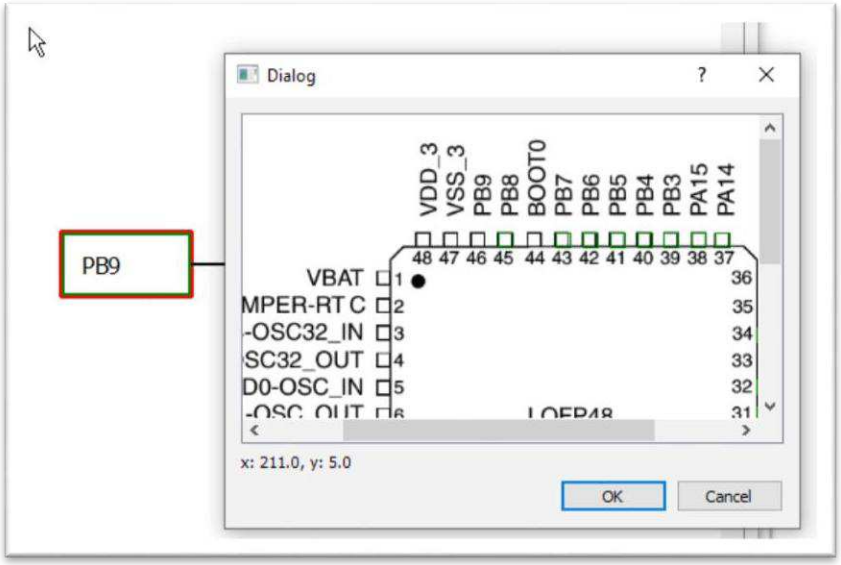


Рисунок 34 – Порты ввода-вывода

С появлением некоторого количества объектов на сцене появилась необходимость выделять несколько элементов сразу, чтобы можно было переместить или удалить группу элементов. На помощь приходят общепринятые стандарты: выделение прямоугольником и поэлементный выбор с помощью клавиши ‘Ctrl’. Также, согласно общепринятым стандартам, добавим обработку нажатия клавиши ‘Del’ для удаления выделенных объектов

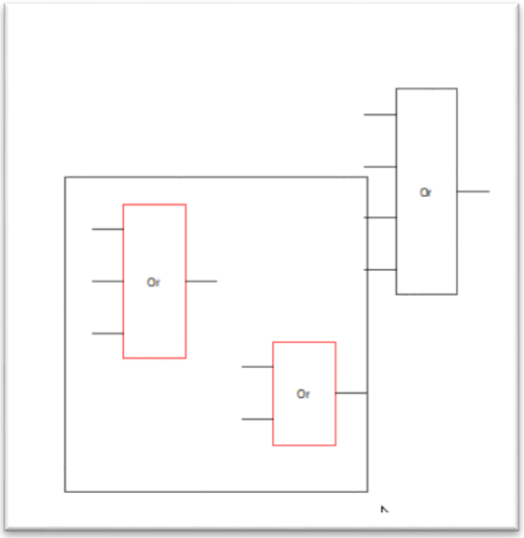


Рисунок 35 – Выделение нескольких объектов

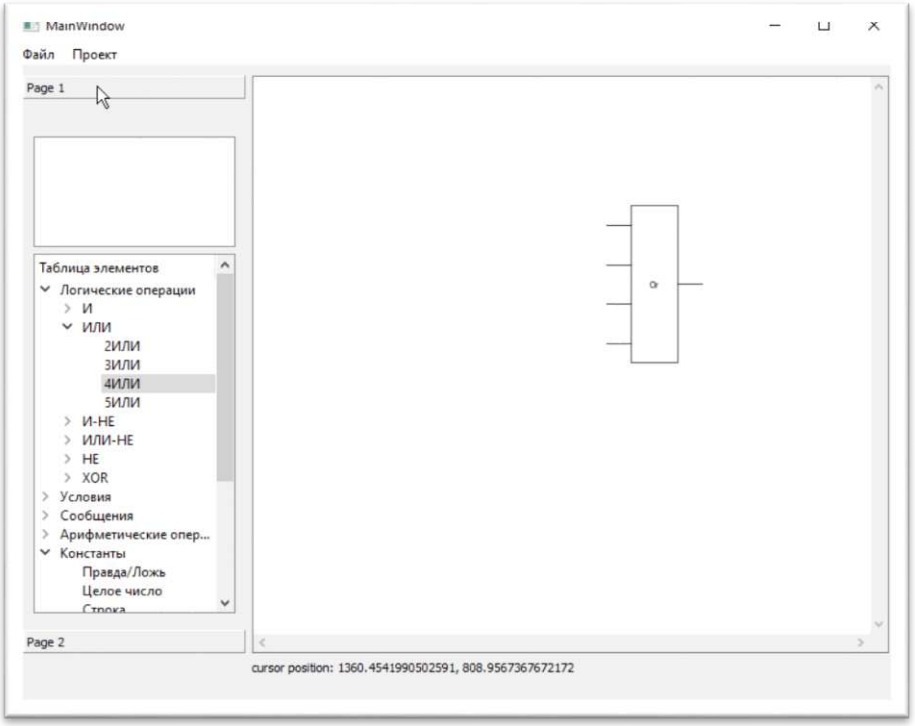


Рисунок 36 – Удаление нескольких объектов

Следующий важный этап – соединение элементов. Каждое соединение должно быть описанием того, что она соединяет. Вместе с массивом элементов сцена будет хранить и массив соединений в формате [{элемент1.ножка, элемент2.ножка, точка начала, точка конца}, …].

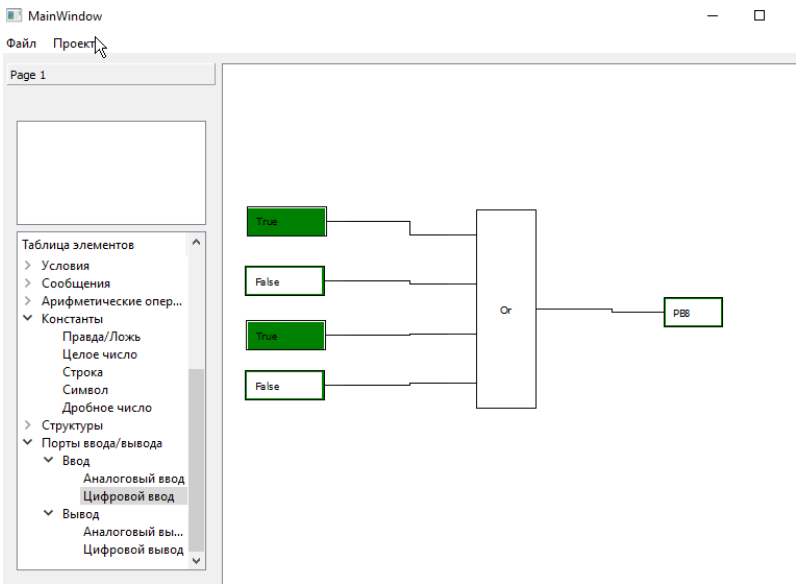


Рисунок 37 – Соединение объектов

Затем добавлена возможность создавать переменные, как глобальные, так и локальные. Создаются переменные из контекстного меню (рис. 38 и 39), созданного кликом правой кнопки мыши по сцене. Так как глобальные переменные не видно на сцене, создано меню обработки глобальных переменных (рис. 40) с возможностью редактировать и удалять записи. Описываются глобальные переменные с помощью специального диалогового окна (рис. 41), где для переменной указываются тип, имя и значение. Для «разворота» переменной в нужную сторону необходимо кликнуть правой клавишей мыши по самой переменной и выбрать «инвертировать r/w» (рис. 42).

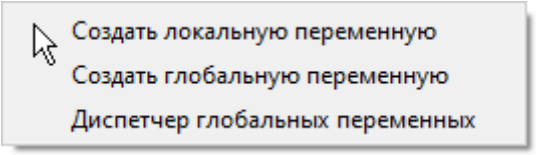


Рисунок 38 – контекстное меню создания переменных

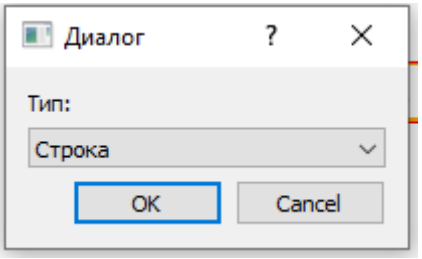


Рисунок 39 – создание локальной переменной

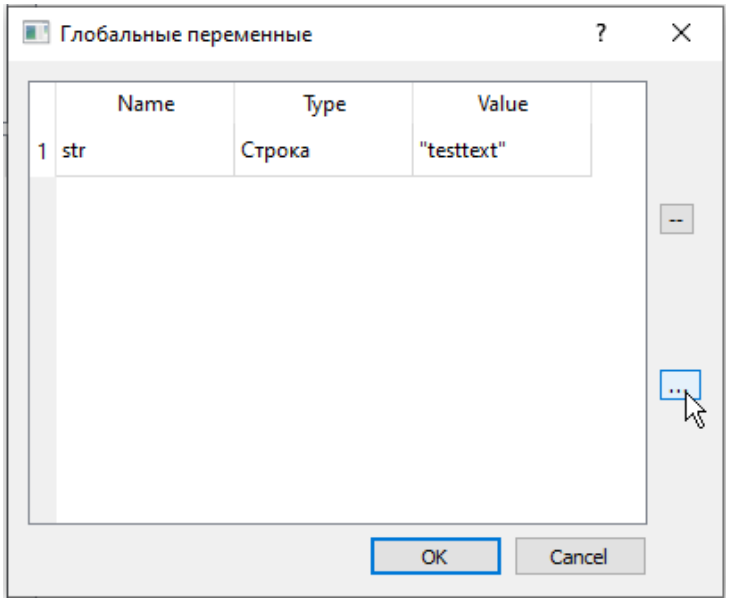


Рисунок 40 – меню редактирования глобальных переменных

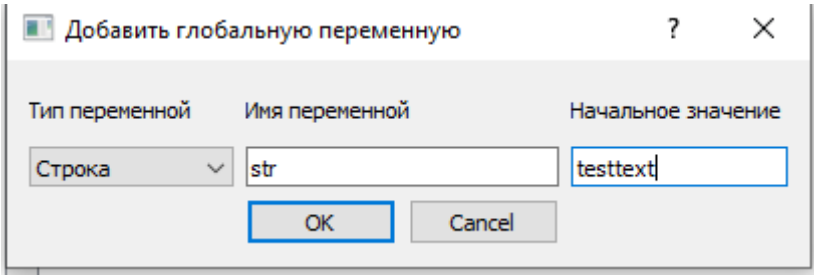


Рисунок 41 – диалоговое окно создания глобальной переменной

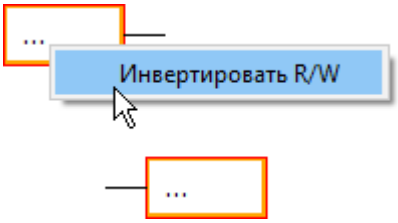


Рисунок 42 – инвертирование блока переменной

Для унификации и удобства пользователя добавлен выбор типа платы. В данной версии пользователю доступны 2 платы: «LQFP48» и «TFBGA64». В зависимости от выбранной платы, при назначении порта, пользователю будет предложено одно из УГО соответствующих плат.

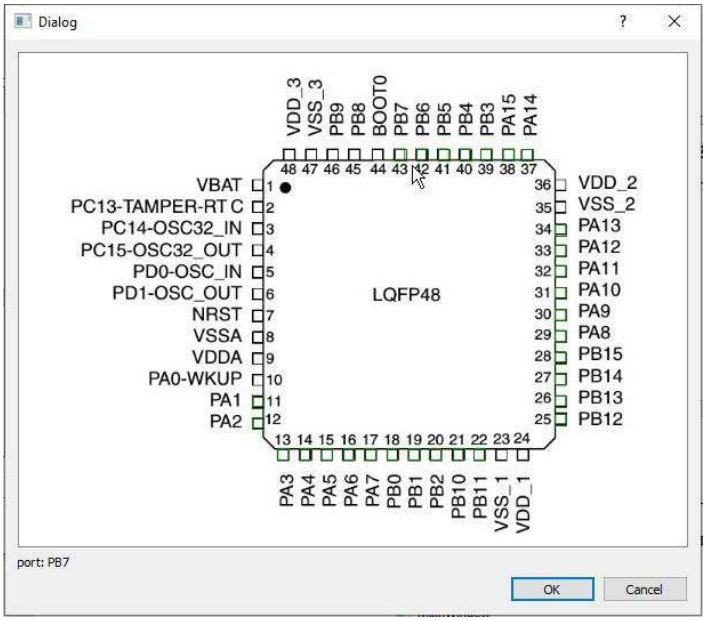


Рисунок 43 – выбор порта «LQFP48»

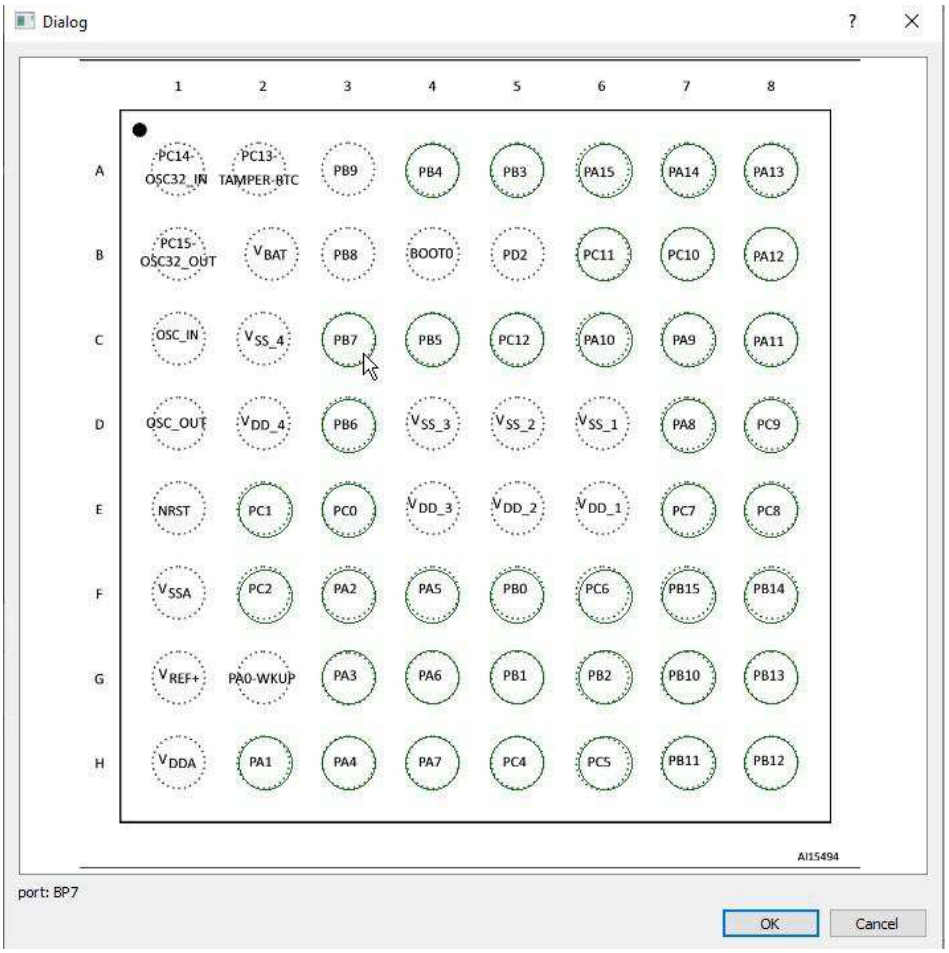


Рисунок 44 – выбор порта «TFBGA64»

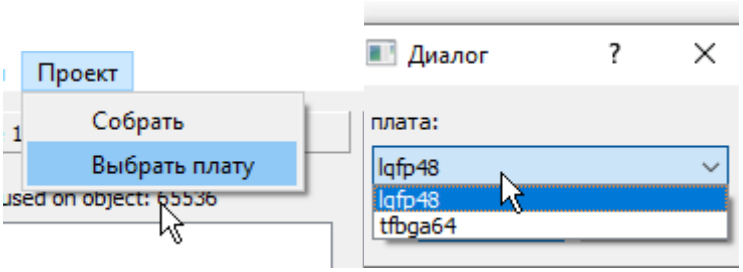


Рисунок 45 – выбор платы

Далее, пользователь может сохранить проект в виде двоичного файла, в котором сохранится массив всех объектов сцены, а может передать данные для дальнейшей сборки.

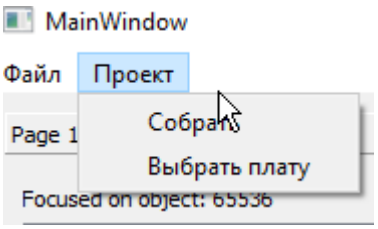


Рисунок 46 – Сборка проекта

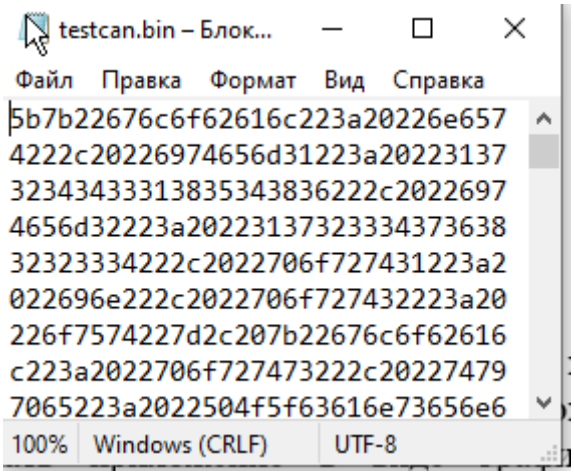


Рисунок 47 – Сохраненный проект

Добавлена обработка ошибок сборки в виде диалогового окна. При разработке генератора кода должны будут обрабатываться ошибки и возвращаться в виде текста графическому интерфейсу.

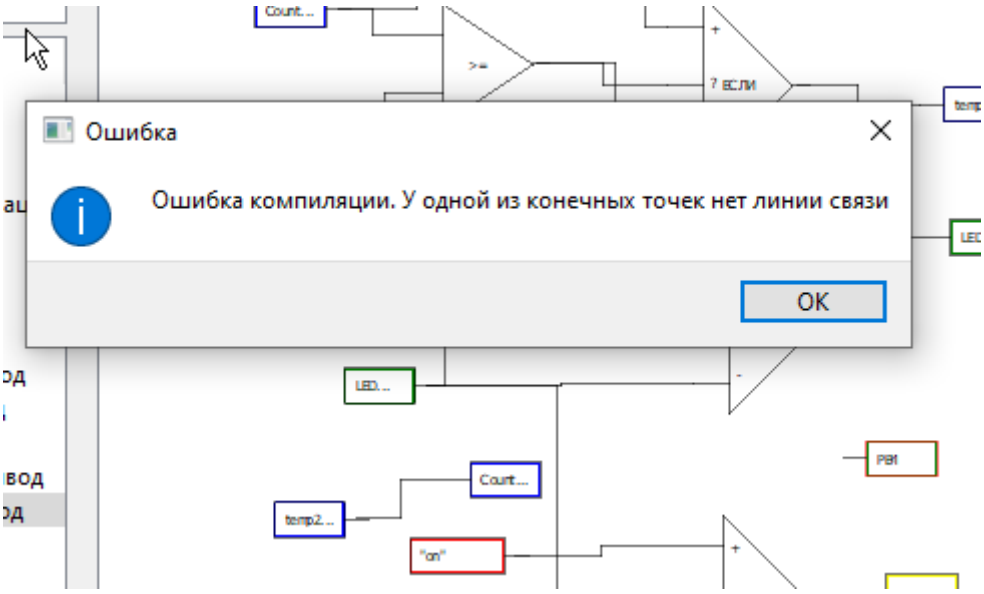


Рисунок 48 – ошибка сборки

4.6 Создание генератора кода

На текущем этапе целесообразно описать способ хранения и передачи данных с графической сцены. Поскольку каждый объект графической сцены имеет сложную структуру, то автоматически преобразовать объект в массив байтов не получится. А так как большинство свойств наследуется у библиотеки Qt, то и всю информацию об объектах хранить и передавать нет необходимости. Достаточно собрать список основных отличительных параметров, которые отображены в таблице 4.

Таблица 4 – Описание отличительных свойств объектов сцены

Ключ поля	Примеры значений	Описание поля
‘global'	‘objects', 'ports', 'net', 'constants', ‘var’, ‘globVar’	Принадлежность объекта к одному из типов (операции, порты и линии связи

Окончание таблицы 4

Ключ поля	Примеры значений	Описание поля
‘type'	‘net: - ; ports: DO_port, DI_port, PO_cansend, PI_candump; objects: OR, AND, XOR, NAND и т.д.; constants: INT, FLT, STR и т.д.	Принадлежность объекта к определенному типу (например, порт ввода или операция "И")
‘val'		Значение объекта. Для портов - имя порта, для констант - значение константы
‘pos'	QPointF()	Позиция объекта относительно начала сцены
‘ports		Описание портов объектов
‘Id’		Уникальный идентификатор объекта
net		
‘item1’		Объект, к которому привязана "левая" сторона линии
‘item2’		Объект, к которому привязана "правая" сторона линии
‘port1’		Описаниие порта "левого" объекта
‘port2’		Описаниие порта "правого" объекта

Вышеуказанный набор данных собирается в список объектов формата словарь (dict), после чего преобразуется в Json с помощью одноименной библиотеки, и затем в байты. Для загрузки объекта на сцену происходит преобразование в обратном порядке, после чего по порядку добавляются сначала все объекты, с помощью функций setVal и setId устанавливаются значения id согласно данным в файле, и только потом добавляются линии связи.

Теперь нужно определиться с порядком генерирования программного кода из данных Json. Поскольку в рамках данной работы сложность конечных программ не должна быть существенной, то разумно вместо полной генерации кода использовать настраиваемый шаблон. Основные аспекты настройки и

инициализации включают в себя инициализацию портов ввода-вывода, настройку CAN интерфейса и описание шаблонных функций, соответствующих функционалу объектов на графической сцене. Листинг шаблона представлен в приложении Б.

Чтобы описать графическую сцену в виде текстовой функции, необходимо обратиться к рекурсии относительно каждой конечной точки. В качестве конечной точки может быть порт вывода или функция CAN Send. Глубина рекурсии определяется самым длинным маршрутом от конечной точки до начальной. Начальной точкой может быть порт ввода, константа или функция CAN Dump. Блок-схема алгоритма представлена на рисунке 49.

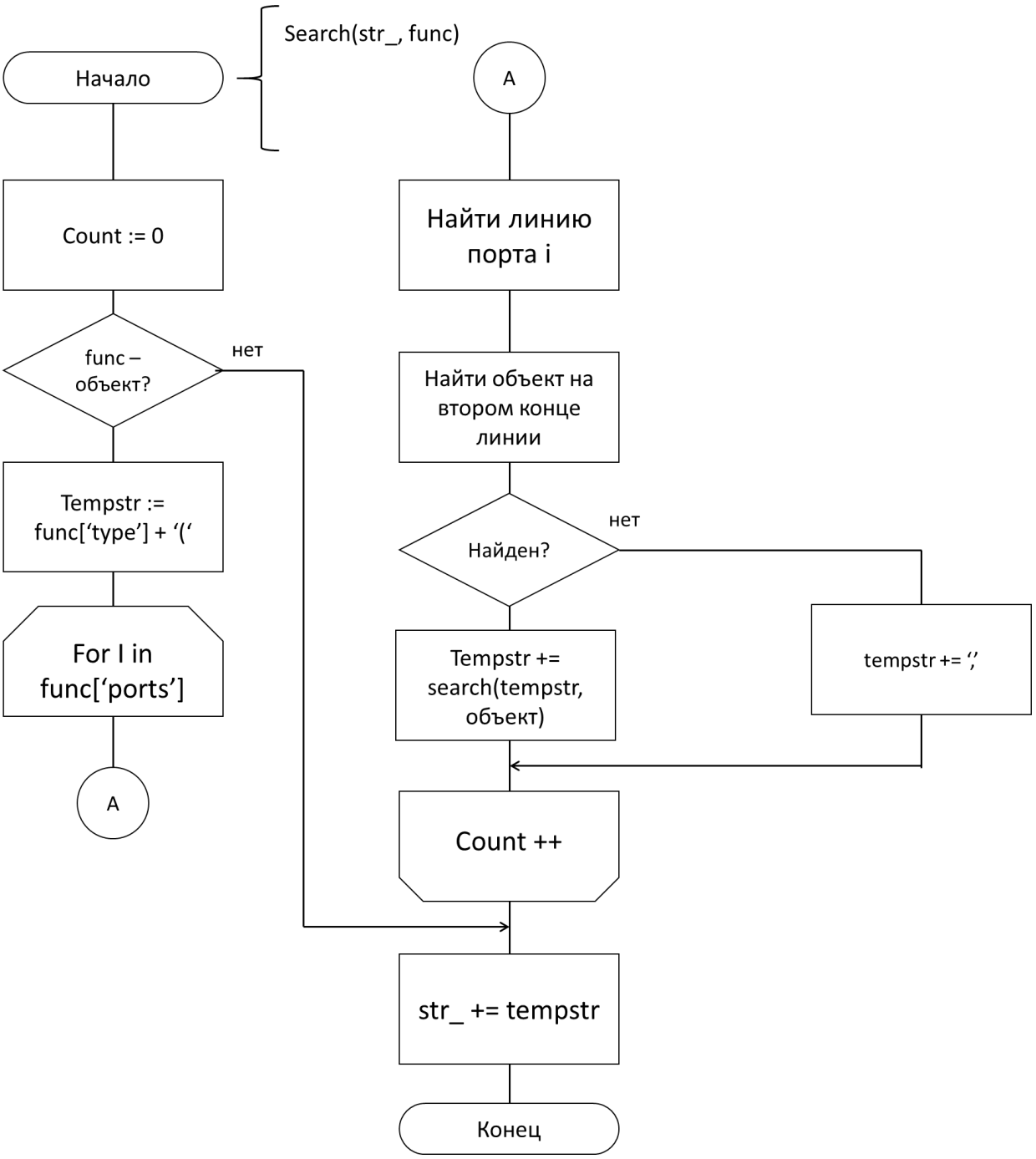


Рисунок 49 – Блок схема рекурсивного алгоритма составления текстовой функции

На основании данного алгоритма необходимо обработать несколько сценариев:

1. Конечная точка – порт, начальная – порт или константа
2. Конечная точка – CAN Send, начальная – порт или константа
3. Конечная точка – порт, начальная – CAN Dump

4. Конечная точка – CAN Send, начальная – CAN Dump

Вышеуказанные сценарии должны описываться по-разному, поскольку порты ввода-вывода и прием-передача сообщений имеют несколько разный механизм. Для обработки данных ситуаций следует добавить несколько блоков к шаблону.

Первый блок /*?candump*/ будет использован для приема сообщений. Здесь будет происходить дополнительная фильтрация по сообщению и вызов какой-либо функции. Следующий – /*?inports*/ отвечает за опрос входных портов и запись их значений в переменные. Для инициализации портов создадим блоки [/*?GPIOA_CRH*/, /*?GPIOA_CRL*/, /*?GPIOB_CRH*/, /*?GPIOB_CRL*/, /*?GPIOC_CRH*/, /*?GPIOC_CRL*/]. В блоке /*?functions*/ будут описаны вызовы функций, сгенерированных рекурсивным методом и присвоение результата их выполнения к вышеуказанной переменной. Последний блок /*?sends*/ будет отвечать за отправку сообщений по шине CAN.

Основой шаблона с точки зрения данной IDE служат шаблонные функции. Их описание представлено в таблице X, а на исходный код шаблона можно посмотреть в приложении Б.

Таблица 5 – соответствие функциональных блоков UI функциям шаблона

Функциональный блок в UI	Соответсвующая функция в шаблоне	Описание функции	Входные параметры	Тип данных на выходе
И	AND	Логическая операция И	(bool a, ...)	bool
ИЛИ	OR	Логическая операция ИЛИ		
ИЛИ-НЕ	NOR	Логическая операция ИЛИ-НЕ		
И-НЕ	NAND	Логическая операция И-НЕ		

Окончание таблицы 5

Функциональный блок в UI	Соответсвующая функция в шаблоне	Описание функции	Входные параметры	Тип данных на выходе
НЕ	NOT	Логическая операция НЕ		
XOR	XOR	Логическая операция XOR		
Сумма	SUM	Арифметическое сложение	(int a, …) (float a, …)	int float
Разность	DIFF	Арифметическая разность		
Произведение	MULT	Арифметическое произведение		
частное	DIV	Арифметическое частное		
Остаток от деления	MOD	Остаток от деления двух чисел		
Округление к ближайшему	ROUND	Округление числа		
Округление к меньшему	FLOOR	Округление числа к меньшему		
Округление к большему	CEIL	Округление числа к большему		
Больше	BIGGER	Сравнение двух чисел	(int a, …) (float a, …) (bool a, …)	bool
Больше-равно	EQBIGGER			
Меньше	LESS			
Меньше-равно	EQLESS			
Равно	EQ			
Can Send	CAN_Send_Test	Отправка сообщения по шине CAN	(string text)	-
Can Dump	USB_LP_CAN1_RX0_IRQHandler	Обработчик прерывания от контроллера CAN. Срабатывает при приеме сообщения, соответствующего фильтру	-	-
Если	IF	Мультиплексор	(<any> a, bool b, <any> c	<any>

Таким образом, при обращении модуля UI к генератору кода, формируется «скелет» из функций, определяющих значение каждого конечного блока, затем в зависимости от сценария, наполняются массивы строк, соответствующие каждому блоку шаблона, массивы строк сортируются согласно парадигме потоков данных и отправляются обратно для дальнейшего формирования файла. На стороне UI принятые массивы строк объединяются в строки и записываются в соответствующие блоки файла шаблона.

4.7 Выводы к разделу

Результатом данного раздела является графическая среда разработки для CAN-сетей на основе микроконтроллеров STM32. Данная среда позволяет создавать программы в виде графических блоков и преобразовывать их в код на языке C для дальнейшей компиляции и прошивки устройств.

5 Отладка и тестирование компонентов для сетей CAN в предлагаемой среде разработки

В данном разделе описана проверка корректности работы программы на некоторых тестовых наборах данных. В результате проверки заключен вывод по проделанной работе.

5.1 Тестирование графического интерфейса

Путем добавления некоторого количества элементов на графическую сцену и связей к ним создан первый тестовый набор. Внешний вид главного окна представлен на рисунке 50.

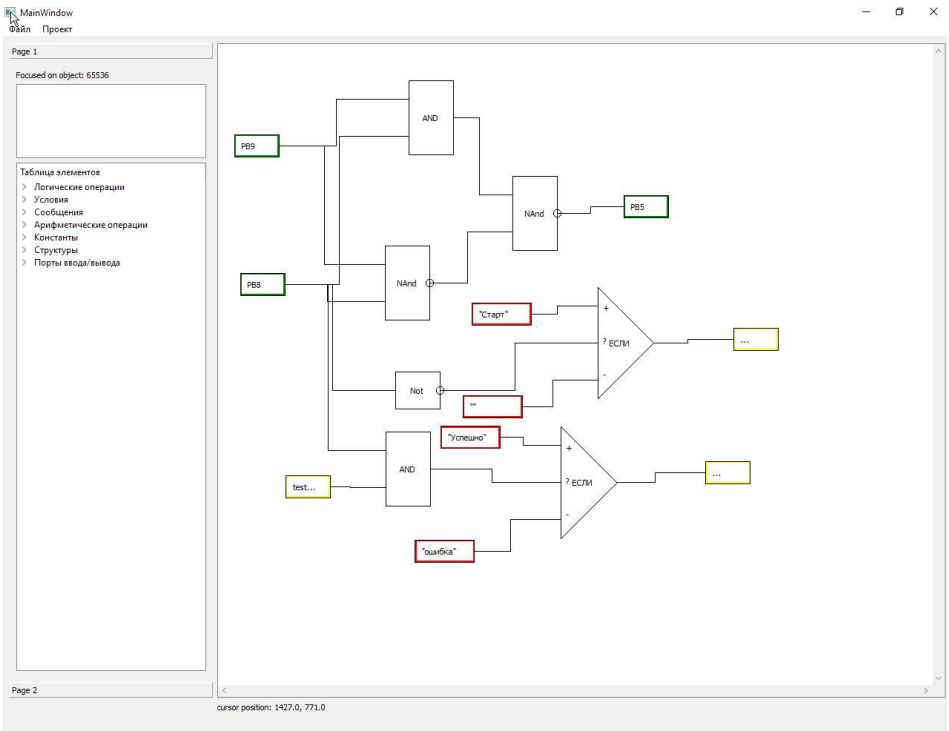


Рисунок 50 – Тестовый набор данных

Произведена проверка прочности взаимосвязей путем перетаскивания объектов в другие области сцены. Ни один из добавленных объектов не потерял связь с другим при перемещении в другое место. Результаты теста на рисунке 51.

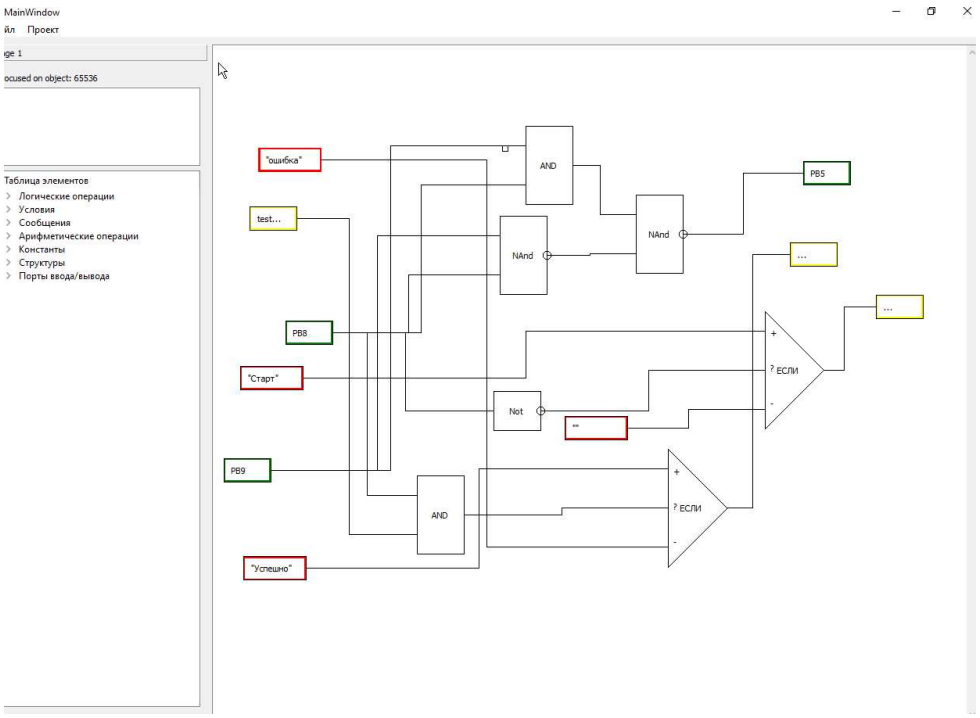


Рисунок 51 – Результаты теста на прочность взаимосвязей

Следующий важный этап – проверка целостности данных при сохранении и загрузке файла. Для проверки проект был сохранен в файле fulltest.bin и загружен обратно. Ни один из объектов их связей не был утерян при сохранении и загрузке.

При проведении тестов графического интерфейса не было замечено ошибок в работе. Все объекты корректно создаются, редактируются, отображаются и перемещаются по сцене.

5.2 Тестирование генератора кода

Для проверки работоспособности генератора кода взят уже готовый набор данных. Исходный код сохранен в файл 123.c. В необходимые блоки был корректно добавлен код, соответствующий исходной схеме. Результаты представлены на рисунках 52 – 54

```
if (RxMessage.StdId == CAN_CMD_Test_Send) {      // Если получили тестовое сообщение
    CAN_Send_Ok();
                    string txt_data = Encoding.ASCII.GetBytes("test");
                uint16 *TxtData
                TxtData = new uint16[8]
                TxtData[0] = txt_data % 256;
                txt_data = txt_data >> 8;
                TxtData[1] = txt_data % 256;
                txt_data = txt_data >> 8;
                TxtData[2] = txt_data % 256;
                txt_data = txt_data >> 8;
                TxtData[3] = txt_data % 256;
                txt_data = txt_data >> 8;
                TxtData[4] = txt_data % 256;
                txt_data = txt_data >> 8;
                TxtData[5] = txt_data % 256;
                txt_data = txt_data >> 8;
                TxtData[6] = txt_data % 256;
                txt_data = txt_data >> 8;
                TxtData[7] = txt_data % 256;
                txt_data = txt_data >> 8;
                if TxtData == RxMessage.Data
                {
                    out_port_17115739615 = IF("Успешно", AND(PB8, true, null), "ошибка", null);
                    CAN_Send_Test(out_port_17115739615);
  }
}
if (RxMessage.StdId == CAN_CMD_Test_Ok) {        // Если получили подтверждение получения
```

Рисунок 52 – Блок /*?candump*/

```
void initPort(void){
    RCC->APB2ENR |= 0x00000010; // Подключение тактового сигнала

    GPIOA->CRH = 0x00000000; // Настройка порта AH
    GPIOA->CRL = 0x00000000; // Настройка порта AL

    GPIOB->CRH = 0x00000022; // Настройка порта BH
    GPIOB->CRL = 0x00300000; // Настройка порта BL

    GPIOC->CRH = 0x00000000; // Настройка порта CH
    GPIOC->CRL = 0x00000000; // Настройка порта CL
}
```

Рисунок 53 – Сгенерированная инициализация портов

```
void main()
{
    initPort();
    while(1){
        // опрос портов
        PB8 = (GPIOB->IDR & GPIO_IDR_IDR8)

        PB9 = (GPIOB->IDR & GPIO_IDR_IDR9)


        // вызов сгенерированных функций
        out_port_PB5 = NAND(AND(PB9, PB8, null), NAND(PB9, PB8, null), null);
        out_port_PB5 & GPIOB->BSRR=GPIO_BSRR_BS5; // установка порта
        !out_port_PB5 & GPIOB->BSRR=GPIO_BSRR_BR5; // сброс порта

        // отправка сообщений
        out_port_172443185486 = IF("Старт", NOT(PB8, null), "", null);
        CAN_Send_Test(out_port_172443185486);
    }
}
```

Рисунок 54 – Сгенерированные блоки /*?functions*/ и /*?cansend*/

Для более справедливой оценки были проведены еще несколько тестов на других наборах данных. Следующий набор данных – несколько маршрутов разной длины от 0 до 3. Скриншоты самого набора данных и результатов генерации кода представлены на рисунках 55 – 58

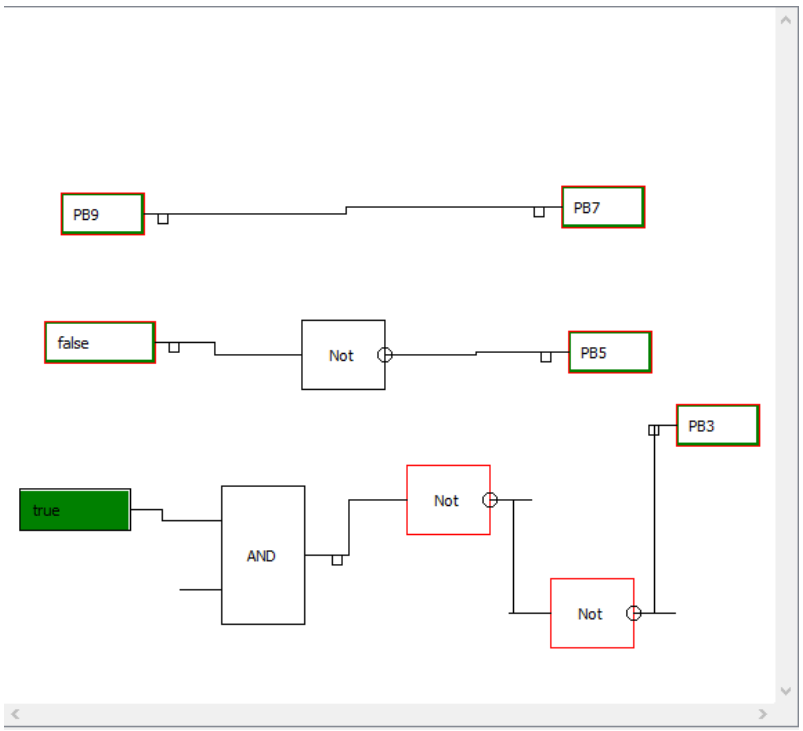


Рисунок 55 – Дополнительный тестовый набор данных

```
if (CAN_GetITStatus(CAN1, CAN_IT_FMP0) != RESET)                    // Проверим почтовый ящик
{
    CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);

    // Проверим тип заголовка сообщения
    if (RxMessage.IDE == CAN_Id_Standard) {
        if (RxMessage.StdId == CAN_CMD_Test_Send) {       // Если получили тестовое сообщение
            CAN_Send_Ok();

        }
        if (RxMessage.StdId == CAN_CMD_Test_Ok) {         // Если получили подтверждение получения

        }
    }
}
```

Рисунок 56 – Блок /*?candump*/

```
void initPort(void){
    RCC->APB2ENR |= 0x00000010; // Подключение тактового сигнала

    GPIOA->CRH = 0x00000000; // Настройка порта AH
    GPIOA->CRL = 0x00000000; // Настройка порта AL

    GPIOB->CRH = 0x00000020; // Настройка порта BH
    GPIOB->CRL = 0x30303000; // Настройка порта BL

    GPIOC->CRH = 0x00000000; // Настройка порта CH
    GPIOC->CRL = 0x00000000; // Настройка порта CL
}
```

Рисунок 57 – Блоки инициализации портов

```
void main()
{
    initPort();
    while(1){
        // опрос портов
        PB9 = (GPIOB->IDR & GPIO_IDR_IDR9)


        // вызов сгенерированных функций
        out_port_PB3 = NOT(NOT(AND(true, null), null), null);
        out_port_PB3 & GPIOB->BSRR=GPIO_BSRR_BS3; // установка порта
        !out_port_PB3 & GPIOB->BSRR=GPIO_BSRR_BR3; // сброс порта
        out_port_PB5 = NOT(false, null);
        out_port_PB5 & GPIOB->BSRR=GPIO_BSRR_BS5; // установка порта
        !out_port_PB5 & GPIOB->BSRR=GPIO_BSRR_BR5; // сброс порта
        out_port_PB7 = PB9;
        out_port_PB7 & GPIOB->BSRR=GPIO_BSRR_BS7; // установка порта
        !out_port_PB7 & GPIOB->BSRR=GPIO_BSRR_BR7; // сброс порта

        // отправка сообщений

    }
}
```

Рисунок 58 – Функциональные блоки

Как видно из рисунка 56, блок реакции на прием сообщения остался пустым, так на исходной схеме нет ни одного блока CAN Dump. Рисунок 57 показывает, что были инициализированы порты PB3, PB5 и PB7 на вывод, а порт PB9 – на вход. На последнем 58-м рисунке отображен опрос порта PB9 и установка/сброс портов PB3, PB5, PB7 в зависимости от результата выполнения соответствующей функции. При этом блок отправки сообщений остался пустым, поскольку ни одного блока CAN Send на исходной схеме не было. Функции сгенерированы корректно.

Следующим этапом следует протестировать правильность генерации кода с точки зрения парадигмы потоков данных для локальных и глобальных переменных. Созданы два тестовых набора. Рассмотрим оба от простого к сложному.

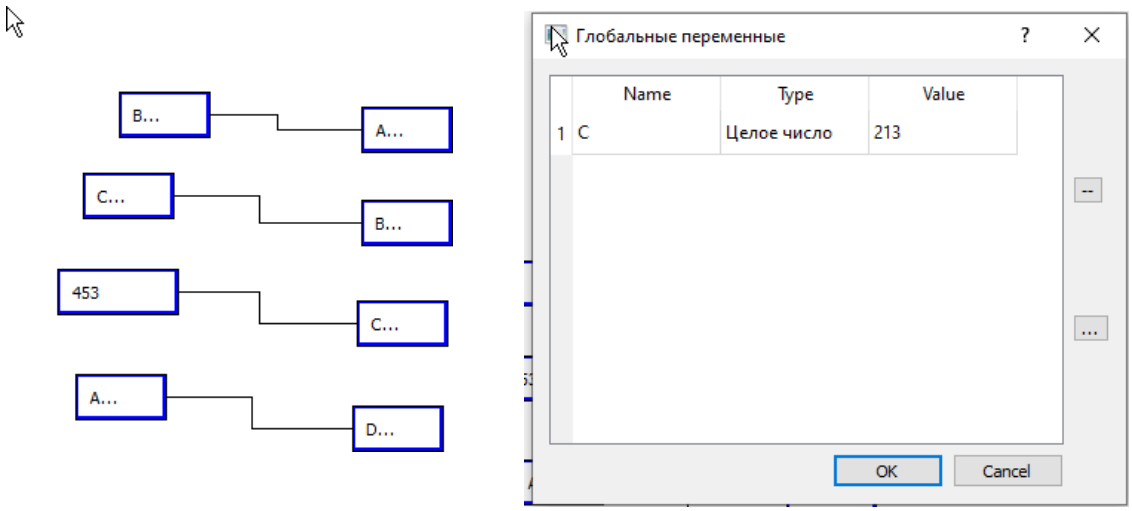


Рисунок 59 – тестовый набор №1 для проверки парадигмы потоков данных

```
int C = 213;

void main()
{
        initPort();

        while(1){
                // опрос портов


                // вызов сгенерированных функций
                C = 453;
                int B = C;
                int A = B;
                int D = A;

                // отправка сообщений

        }
}
```

Рисунок 60 – сгенерированный код для тестового набора

Как видно из рисунка 59, переменной D в итоге должно присвоиться значение 453. На рисунке 60 отображен фрагмент кода, где видно, как была создана глобальная переменная int C, внутри основного цикла обращение происходит уже без указания типа, присвоения были отсортированы в

нужном порядке, чтобы переменной int D было присвоено значение переменной C, равной 453.

Рассмотрим более сложный и комплексный пример. Реализована мигалка с отправкой по CAN-шине текстового сообщения о состоянии лампочки.

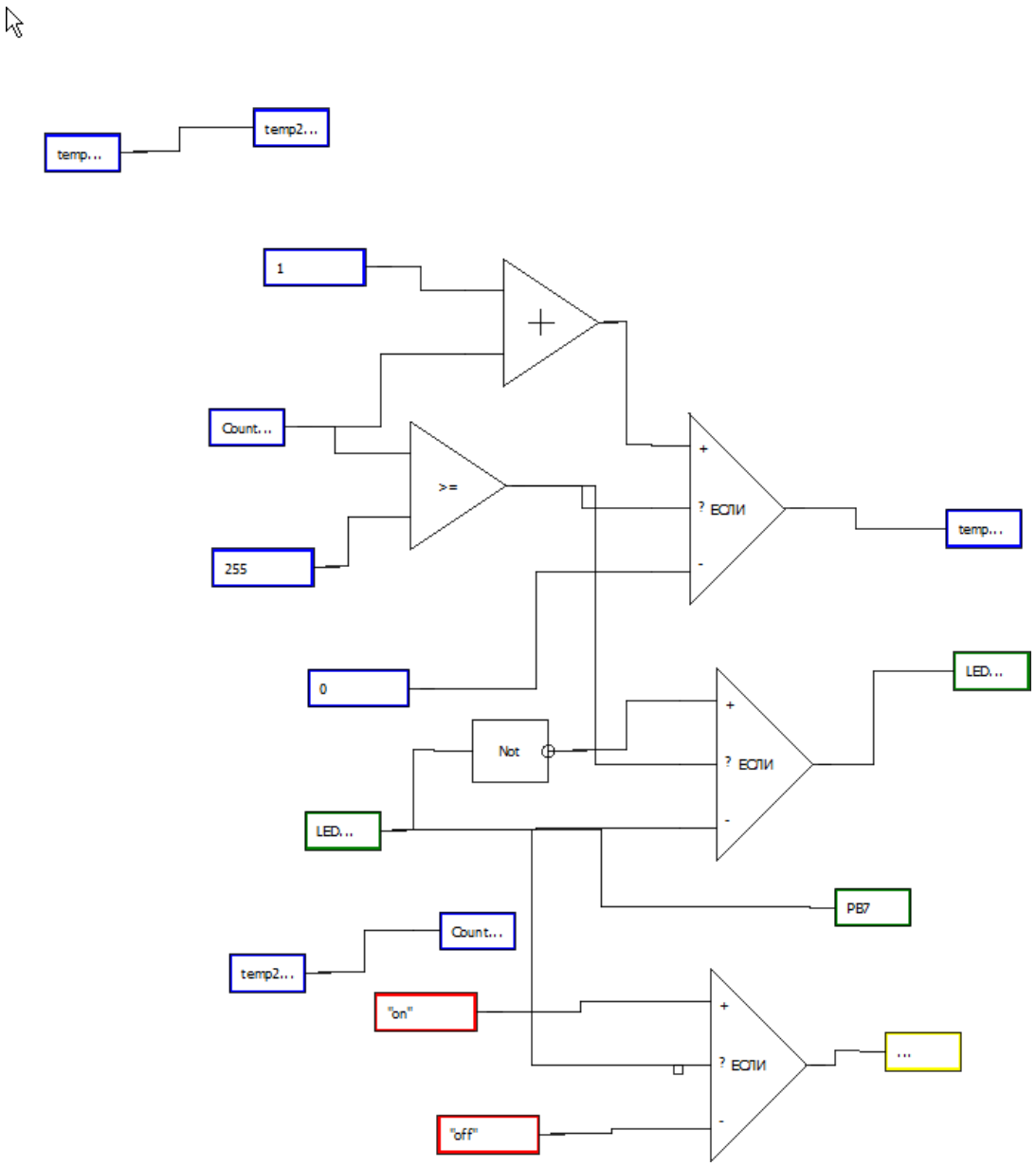


Рисунок 61 – тестовый набор №2

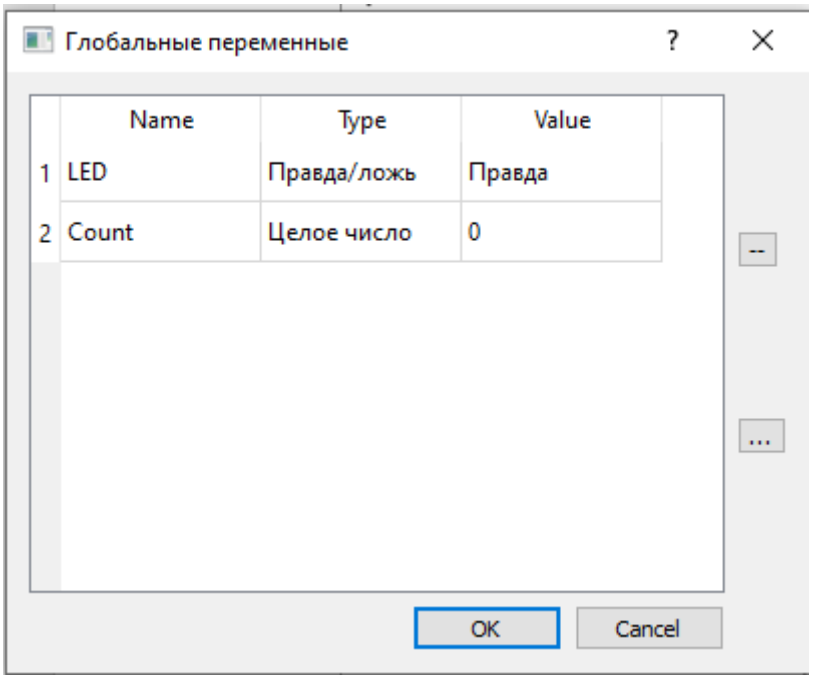


Рисунок 62 – набор глобальных переменных

```
void initPort(void){
        RCC->APB2ENR |= 0x00000010; // Подключение тактового сигнала

        GPIOA->CRH = 0x00000000; // Настройка порта AH
        GPIOA->CRL = 0x00000000; // Настройка порта AL

        GPIOB->CRH = 0x00000000; // Настройка порта BH
        GPIOB->CRL = 0x30000000; // Настройка порта BL

        GPIOC->CRH = 0x00000000; // Настройка порта CH
        GPIOC->CRL = 0x00000000; // Настройка порта CL
}
```

Рисунок 63 – блок настройки портов

```
bool LED = true;
int Count = 0;

void main()
{
        initPort();

        while(1){
                // опрос портов


                // вызов сгенерированных функций
                int temp = IF(SUM(1, Count, null), EQBIGGER(Count, 255, null), 0, null);
                int temp2 = temp;
                Count = temp2;
                LED = IF(NOT(LED, null), EQBIGGER(Count, 255, null), LED, null);
                bool out_port_PB7 = LED;
                       out_port_PB7 & GPIOB->BSRR=GPIO_BSRR_BS7; // установка порта
                       !out_port_PB7 & GPIOB->BSRR=GPIO_BSRR_BR7; // сброс порта


                // отправка сообщений
                out_port_11507155682 = IF("on", LED, "off", null);
                CAN_Send_Test(out_port_11507155682);
        }
}
```

Рисунок 64 – Сгенерированный код

Как видно из рисунка 63, порт PB7 настроен на вывод, согласно исходной схеме программы. Далее, следует пояснить код на рисунке 64. Первые 2 строки – описание глобальных переменных bool LED и int Count, отвечающих за состояние лампочки и количество тактов соответственно. Далее, в основном цикле происходит присваивание переменной temp значения 0 или Count+1 в зависимости от условия Count больше-равно 255. После, полученное значение переходит переменной Count, согласно парадигме потоков данных. Далее, инвертируется значение переменной LED и отправляется на порт PB7. В конце цикла происходит отправка “on” или “off” по CAN-шине, в зависимости от состояния лампочки.

Последним этапом следует проверить обработку простых ошибок в графическом коде. Для этого разорвана одна из связей из предыдущего примера. Результат представлен на рисунке X.

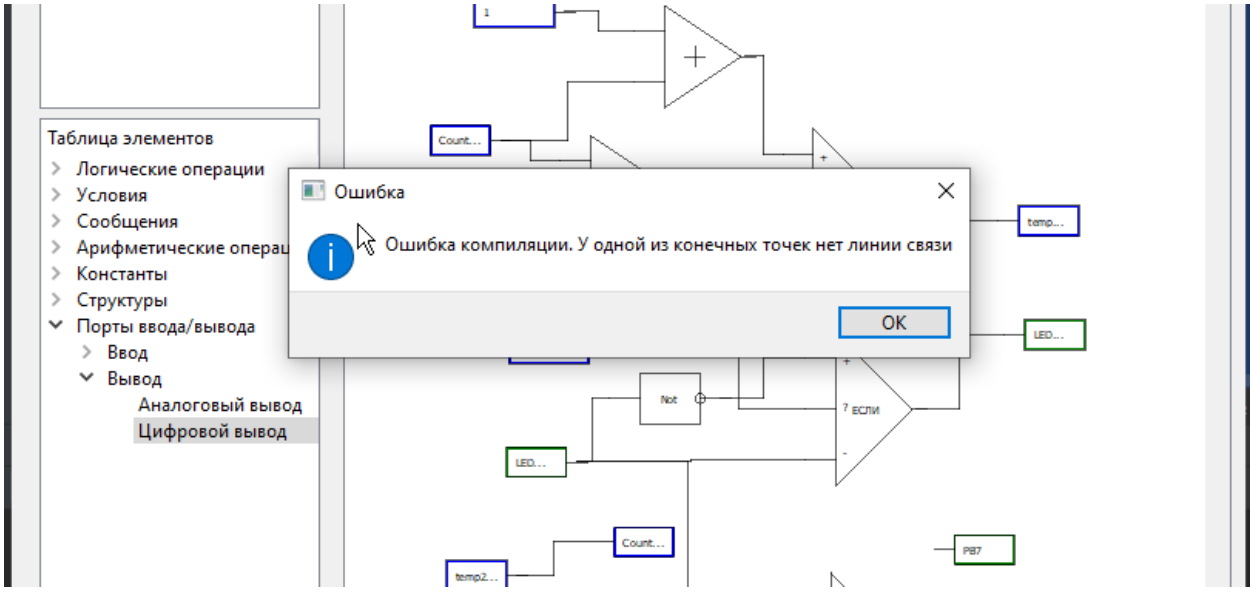


Рисунок 65 – обработка разрыва связи

5.3 Выводы к разделу

Исходя из вышеописанных тестов, разработанная IDE работает корректно на заранее правильно созданных схемах и в случае обнаружения очевидных ошибок в графическом коде выдает соответствующе сообщение. Код, соответствующий графической схеме программы, генерируется корректно.

6 Выводы по проделанной работе

Средствами PyQt был создан модуль графического интерфейса для среды разработки бортовых CAN-сетей. Пользователь с помощью данного модуля может разработать приложение в виде графической схемы. Возможности приложения ограничены простыми арифметическими и логическими преобразованиями таких типов данных, как 1 бит (логический), символ, строка, целое число и число с плавающей точкой. Также

предусмотрена отправка сообщения по шине CAN и реакция на сообщение, отправленное другим устройством. Разработан модуль генерации кода на языке C. Данный модуль способен генерировать корректный программный код для несложных схем и сохранять его отдельным файлом. Сгенерированный код можно передавать на компиляцию сторонним программам для дальнейшей обработки и прошивки устройства.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. CAN интерфейс (Control Area Network) [Электронный ресурс]. – Режим доступа: http://lib.ssga.ru/fulltext/УМК/Метрология/200501_УМК_pdf/8%20-%20Информационно-изм.системы%20(6%20семестр)/модуль%203/лекции/интерфейс%20CAN.pdf. – Дата доступа: 20.12.2019.
2. CannyLab [Электронный ресурс]: CannyLab – Интегрированная среда разработки – режим доступа: https://wiki.canny.ru/index.php?title=CannyLab . – Дата доступа: 10.12.2019.
3. Пособие по программе самообразования № 269: Обмен данными посредством шины CAN II [Электронный ресурс]. – Режим доступа: https://help4auto.com/download/ssp/269_Obmen%20dannymi%20posredstvom%20shiny%20CAN%20II.PDF. – Дата доступа: 20.12.2019.
4. Микроконтроллеры 8051, PIC, AVR и ARM отличия и особенности [Электронный ресурс]. – Режим доступа: http://digitrode.ru/computing-devices/mcu_cpu/1253-mikrokontrollery-8051-pic-avr-i-arm-otlichiya-i-osobennosti.html. – Дата доступа: 10.01.2020.
5. Microcontrollers - STM32 Arm Cortex MCUs - STMicroelectronics [Электронный ресурс]. – Режим доступа: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. – Дата доступа: 20.02.2020.
6. Qt for Python - Qt for Python [Электронный ресурс]. – Режим доступа: https://doc.qt.io/qtforpython/. – Дата доступа: 20.02.2020.
7. СТО 4.2–07–2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. – Введ. 30.12.2013. – 60 с.

ПРИЛОЖЕНИЕ А

Листинг 1 – файл исходного кода генератора codeGenerator.py

```
1.          import copy
2.          import json
3.
4.          class Generator(object):
5.              def_init__(self):
6.                  self.globDict = {
7.                      'and': 'AND',
8.                      'nand': 'NAND',
9.                      'or': 'OR',
10.                     'not': 'NOT',
11.                     'nor': 'NOR',
12.
13.                     'CEIL': 'CEIL',
14.                     'TRUNK': 'TRUNK',
15.                     'ROUND': 'ROUND',
16.                     'SUM': 'SUM',
17.                     'DIF': 'DIF',
18.                     'DIV': 'DIV',
19.                     'MULT': 'MULT',
20.                     'MOD': 'MOD',
21.                     'BIGGER': 'BIGGER',
22.                     'LESS': 'LESS',
23.                     'EQ': 'EQ',
24.                     'EQLESS': 'EQLESS',
25.                     'IF': 'IF',
26.                     'EQBIGGER': 'EQBIGGER'
27.                 }
28.
29.             def search(self, str_, func, found, name):
30.                 found = found
31.                 count = 0
32.                 if func['global'] == 'objects':
33.                     tempstr = self.globDict[func['type']] + '('
34.                     count = 0
35.                     for i in range(1, 6):
36.                         n_line       =       self.findLine(func,
37.         func['ports']['in_' + str(i)]['name'])
38.                         if n_line['line'] != None:
39.                             n_func                              =
40.         self.findObjByEndOfLine(n_line)
41.                             if n_func != None:
42.                                 temp    =    self.search(tempstr,
43.         n_func, found, name)
44.                                 tempstr += temp[1]
45.                                 found, name = temp[2], temp[3]
46.                             else:
47.                                 tempstr += ', '
48.                             count += 1
49.                     tempstr += "null" + '), '
50.                 else:
```

```
51.                     val = func['val']
52.                     if func['type'] == 'chr_const':
53.                         val = "'" + val[0] + "'"
54.                     tempstr = str(val) + ', '
55.                     if func['type'] == 'PI_candump':
56.                         found = True
57.                         name = func['val']
58.                 str_ += tempstr
59.                 return str_, tempstr, found, name
60.
61.             def findLine(self, obj, port):
62.                 for i in self.json_:
63.                     if i['global'] == 'net':
64.                         if   obj['id']   ==   i['item1']   and
65.         obj['ports'][port]['name'] == i['port1']:
66.                             return {'line': i, 'end': 'left'}
67.                         elif   obj['id']   ==   i['item2']   and
68.         obj['ports'][port]['name'] == i['port2']:
69.                             return {'line': i, 'end': 'right'}
70.                 return {'line': None, 'end': None}
71.
72.             def findObjByEndOfLine(self, line_with_end):
73.                 for i in self.json_:
74.                     if i['global'] == 'objects' or i['global'] ==
75.         'ports' or i['global'] == 'constants' \
76.                             or i['global'] == 'var':
77.                         if  line_with_end['end']  ==  'left'  and
78.         line_with_end['line']['item2'] == i['id']:
79.                             return i
80.                         elif line_with_end['end'] == 'right' and
81.         line_with_end['line']['item1'] == i['id']:
82.                             return i
83.                 return None
84.
85.             def getPorts(self):
86.                 ports = {
87.                     'BH': ['0', '0', '0', '0', '0', '0', '0',
88.         '0'],
89.                     'BL': ['0', '0', '0', '0', '0', '0', '0',
90.         '0'],
91.                     'CH': ['0', '0', '0', '0', '0', '0', '0',
92.         '0'],
93.                     'CL': ['0', '0', '0', '0', '0', '0', '0',
94.         '0'],
95.                     'AH': ['0', '0', '0', '0', '0', '0', '0',
96.         '0'],
97.                     'AL': ['0', '0', '0', '0', '0', '0', '0',
98.         '0']
99.                 }
100.                for i in self.json_:
101.                    if i['global'] == 'ports' and i['type'][-4:]
102.        == 'port':
103.                        temp = '0'
104.                        if i['type'] == 'DO_port':
105.                            temp = '3'
106.                        elif i['type'] == 'DI_port':
```

```
107.                            temp = '2'
108.                        let = i['val'][1]
109.                        ind = int(i['val'][2:])
110.                        if ind > 7:
111.                            let += 'H'
112.                        else:
113.                            let += 'L'
114.                        ind = ind % 8
115.                        ports[let][7 - ind] = temp
116.                return ports
117.
118.            def getQuestPorts(self):
119.                ports = []
120.                for i in self.json_:
121.                    try:
122.                        if i['type'][1:] == 'I_port'  and
123.        i['global'] == 'ports':
124.                            ports.append(i['val'] + ' = (GPIO' +
125.        i['val'][1] + '->IDR & GPIO_IDR_IDR' + i['val'][2:] +
126.        ')')
127.                            ports.append('')
128.                    except KeyError:
129.                        pass
130.                return ports
131.
132.            def getResult(self, str_):
133.                self.json_ = json.loads(str_)
134.                out_ports = []
135.                strings = []
136.                sends = []
137.                dumps = []
138.                globalVars = {}
139.                for i in self.json_:
140.                    try:
141.                        if i['global'] == 'globvar':
142.                            globalVars[i['name']] = i['type']
143.                            print(globalVars)
144.                    except KeyError:
145.                        pass
146.                for i in self.json_:
147.                    try:
148.                        if (i['type'][1] == 'O' and  i['global']
149.        == 'ports') \
150.                                or (i['rw'] == 1 and i['global']
151.        == 'var'):
152.                            out_ports.append(i)
153.                    except KeyError:
154.                        pass
155.                if len(out_ports) == 0:
156.                    return  {'mes':  'Ошибка  компиляции.  Нет
157.        конечной точки', 'dat': ''}
158.                for i in out_ports:
159.                    line = self.findLine(i, 'in')
160.                    if line['line'] is None:
161.                        return  {'mes':  'Ошибка  компиляции.  У
162.        одной из конечных точек нет линии связи', 'dat': ''}
```

```
163.                    func = self.findObjByEndOfLine(line)
164.                    if func is None:
165.                        return  {'mes':  'Ошибка  компиляции.  У
166.        одной из промежуточных точек нет линии связи', 'dat': ''}
167.
168.                    string = self.search('?' + ' = ', func, False,
169.        '')
170.
171.                    if (string[2]):
172.                        dumps.append('
173.              string  txt_data  =  Encoding.ASCII.GetBytes("'  +
174.        string[3] + '");')
175.                        dumps.append('
176.              uint16 *TxtData')
177.                        dumps.append('
178.              TxtData = new uint16[8]')
179.                        dumps.append('
180.              TxtData[0] = txt_data % 256;')
181.                        dumps.append('
182.              txt_data = txt_data >> 8;')
183.                        dumps.append('
184.              TxtData[1] = txt_data % 256;')
185.                        dumps.append('
186.              txt_data = txt_data >> 8;')
187.                        dumps.append('
188.              TxtData[2] = txt_data % 256;')
189.                        dumps.append('
190.              txt_data = txt_data >> 8;')
191.                        dumps.append('
192.              TxtData[3] = txt_data % 256;')
193.                        dumps.append('
194.              txt_data = txt_data >> 8;')
195.                        dumps.append('
196.              TxtData[4] = txt_data % 256;')
197.                        dumps.append('
198.              txt_data = txt_data >> 8;')
199.                        dumps.append('
200.              TxtData[5] = txt_data % 256;')
201.                        dumps.append('
202.              txt_data = txt_data >> 8;')
203.                        dumps.append('
204.              TxtData[6] = txt_data % 256;')
205.                        dumps.append('
206.              txt_data = txt_data >> 8;')
207.                        dumps.append('
208.              TxtData[7] = txt_data % 256;')
209.                        dumps.append('
210.              txt_data = txt_data >> 8;')
211.                        dumps.append('
212.              if TxtData == RxMessage.Data')
213.                        dumps.append('
214.              {')
215.                        if i['type'][-4:] == 'port':
216.                            temp         =         string[0][:-
217.        2].replace(string[3], 'true')
218.
```

```
219.                                temp = temp.replace('?', 'out_port_'
220.         + i['id'])
221.                                dumps.append('   ' + temp + ';')
222.                                portname = i['val'][1]
223.                                portnum = i['val'][2:]
224.                                dumps.append('
225.                     out_port_' + i[
226.                                    'val'] + ' & GPIO' + portname +
227.         '->BSRR=GPIO_BSRR_BS' + portnum + '; // установка порта')
228.                                dumps.append('
229.                     !out_port_' + i[
230.                                    'val'] + ' & GPIO' + portname +
231.         '->BSRR=GPIO_BSRR_BR' + portnum + '; // сброс порта')
232.                            elif i['global'] == 'var':
233.                                print('globvar')
234.                                temp          =          string[0][:-
235.         2].replace(string[3], 'true')
236.                                type = i['type'] + ' '
237.                                try:
238.                                    globalVars[i['val']]
239.                                    type = ''
240.                                except KeyError:
241.                                    pass
242.                                temp   =   temp.replace('?',   type   +
243.         i['val'])
244.                                dumps.append('   ' + temp + ';')
245.                            else:
246.                                print('else')
247.                                temp          =          string[0][:-
248.         2].replace(string[3], 'true')
249.                                temp = temp.replace('?', 'out_port_'
250.         + i['id'])
251.                                dumps.append('
252.                     ' + temp + ';')
253.                                dumps.append('
254.                     CAN_Send_Test('  +  'out_port_'  + i['id'] +
255.         ');')
256.                            dumps.append('      }')
257.                        else:
258.                            if i['type'][-4:] == 'port':
259.                                srch = self.search('bool out_port_'
260.         + i['val'] + ' = ', func, False, '')
261.                                string = srch[0]
262.                                portname = i['val'][1]
263.                                portnum = i['val'][2:]
264.
265.                                strings.append({'str': string[:-2] +
266.         ';\n' +
267.                                                        '
268.         out_port_' + i[
269.                                                            'val']  +
270.         ' & GPIO' + portname + '->BSRR=GPIO_BSRR_BS' +
271.                                                        portnum + ';
272.         // установка порта \n' +
273.                                                        '
274.         !out_port_' + i[
```

```
275.                                                        'val']  +
276.         ' & GPIO' + portname + '->BSRR=GPIO_BSRR_BR' +
277.                                                   portnum + ';
278.         // сброс порта \n',
279.                                            'com':  'port  '  +
280.         srch[1]})
281.                         elif i['global'] == 'var':
282.                             type = i['type'] + ' '
283.                             try:
284.                                 globalVars[i['val']]
285.                                 type = ''
286.                             except KeyError:
287.                                 pass
288.                             srch = self.search(type + i['val'] +
289.         ' = ', func, False, '')
290.                             string = srch[0]
291.                             strings.append({'str': string[:-2] +
292.         ';', 'com': i['val'] + ' ' + srch[1]})
293.                         else:
294.                             srch  =  self.search('out_port_'  +
295.         i['id'] + ' = ', func, False, '')
296.                             string = srch[0]
297.                             sends.append(string[:-2] + ';')
298.                             sends.append('CAN_Send_Test('      +
299.         'out_port_' + i['id'] + ');')
300.
301.                 print(strings)
302.                 i = len(strings)
303.                 k = i
304.                 i = i*i
305.
306.                 while i > 0:
307.                     for j in range(i % k,  k):
308.                         l = strings[j]['com'].split(' ')
309.                         r, l = l[1][:-1], l[0]
310.                         if r == strings[i %  k]['com'].split('
311.         ')[0]:
312.                             t = copy.deepcopy(strings[i % k])
313.                             strings[i       %       k]       =
314.         copy.deepcopy(strings[j])
315.                             strings[j] = t
316.                             print('swap:',   strings[i   %   k],
317.         strings[j])
318.                             print('after swap:', strings)
319.                             break
320.                     i-=1
321.                 strings.reverse()
322.
323.
324.                 print(strings)
325.
326.                 for i in range(0, k):
327.                     strings[i] = strings[i]['str']
328.
329.           return strings, sends, dumps
```

ПРИЛОЖЕНИЕ Б

Листинг – файл шаблона для генератора кода

```
1.      #include Math
2.      /*********************************************************
3.  ********************
4.      Определение настройки CAN
5.      *********************************************************
6.  ********************/
7.      #define CAN1_ReMap // Закоментировать, если нет ремапинга
8.  портов
9.
10.     #ifndef CAN1_ReMap
11.           #define CAN1_GPIO_PORT        GPIOA
12.           #define CAN1_RX_SOURCE        GPIO_Pin_11
13. // RX-порт
14.           #define CAN1_TX_SOURCE        GPIO_Pin_12
15. // TX-порт
16.           #define CAN1_Periph        RCC_APB2Periph_GPIOA
17. // Порт перифирии
18.     #else
19.           #define CAN1_GPIO_PORT        GPIOB
20.           #define CAN1_RX_SOURCE        GPIO_Pin_8
21. // RX-порт
22.           #define CAN1_TX_SOURCE        GPIO_Pin_9
23. // TX-порт
24.           #define CAN1_Periph        RCC_APB2Periph_GPIOB
25. // Порт перифирии
26.     #endif
27.
28.
29.
30.     GPIO_InitTypeDef GPIO_InitStructure;
31.     // CAN GPIOs configuration
32.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
33. // включаем тактирование AFIO
34.     RCC_APB2PeriphClockCmd(CAN1_Periph, ENABLE);
35. // включаем тактирование порта
36.
37.     RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN1, ENABLE);
38. // включаем тактирование CAN-шины
39.
40.     // Настраиваем CAN RX pin
41.     GPIO_InitStructure.GPIO_Pin   = CAN1_RX_SOURCE;
42.     GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IPU;
43.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
44.     GPIO_Init(CAN1_GPIO_PORT, &GPIO_InitStructure);
45.
46.     // Настраиваем CAN TX pin
47.     GPIO_InitStructure.GPIO_Pin   = CAN1_TX_SOURCE;
48.     GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
49.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
50.         GPIO_Init(CAN1_GPIO_PORT, &GPIO_InitStructure);
51.
52.         #ifdef CAN1_ReMap
53.               GPIO_PinRemapConfig(GPIO_Remap1_CAN1, ENABLE);
54.     // Переносим Can1 на PB8, PB9
55.         #endif
56.
57.         // Инициализация шины
58.         CAN_InitTypeDef CAN_InitStructure;
59.
60.         CAN_DeInit(CAN1);
61.         CAN_StructInit(&CAN_InitStructure);
62.
63.         // CAN cell init
64.         CAN_InitStructure.CAN_TTCM = DISABLE;
65.         CAN_InitStructure.CAN_ABOM = ENABLE;
66.         CAN_InitStructure.CAN_AWUM = ENABLE;
67.         CAN_InitStructure.CAN_NART = DISABLE;
68.         CAN_InitStructure.CAN_RFLM = DISABLE;
69.         CAN_InitStructure.CAN_TXFP = ENABLE;
70.         CAN_InitStructure.CAN_Mode = CAN_Mode_Normal;
71.         CAN_InitStructure.CAN_SJW = CAN_SJW_1tq;
72.         CAN_InitStructure.CAN_BS1 = CAN_BS1_13tq;
73.         CAN_InitStructure.CAN_BS2 = CAN_BS2_2tq;
74.         CAN_InitStructure.CAN_Prescaler = 50;
75.         CAN_Init(CAN1, &CAN_InitStructure);
76.
77.         // CAN filter init
78.         CAN_FilterInitTypeDef CAN_FilterInitStructure;
79.         CAN_FilterInitStructure.CAN_FilterNumber = 1;
80.         CAN_FilterInitStructure.CAN_FilterMode =
81.     CAN_FilterMode_IdMask;
82.         CAN_FilterInitStructure.CAN_FilterScale =
83.     CAN_FilterScale_32bit;
84.         CAN_FilterInitStructure.CAN_FilterIdHigh = 0x0000;
85.         CAN_FilterInitStructure.CAN_FilterIdLow = 0x0000;
86.         CAN_FilterInitStructure.CAN_FilterMaskIdHigh = 0x0000;
87.         CAN_FilterInitStructure.CAN_FilterMaskIdLow = 0x0000;
88.         CAN_FilterInitStructure.CAN_FilterFIFOAssignment =
89.     CAN_FIFO0;
90.         CAN_FilterInitStructure.CAN_FilterActivation = ENABLE;
91.         CAN_FilterInit(&CAN_FilterInitStructure);
92.
93.         // CAN FIFO0 message pending interrupt enable
94.         CAN_ITConfig(CAN1, CAN_IT_FMP0, ENABLE);
95.
96.         // NVIC Configuration
97.         // Enable CAN1 RX0 interrupt IRQ channel
98.         NVIC_InitTypeDef NVIC_InitStructure;
99.         NVIC_InitStructure.NVIC_IRQChannel = USB_LP_CAN1_RX0_IRQn;
100.        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
101.        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
102.        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
103.        NVIC_Init(&NVIC_InitStructure);
104.
105.        void USB_LP_CAN1_RX0_IRQHandler(void)
```

```
106.        {
107.              CanRxMsg RxMessage;
108.
109.              // Обнулим данные пакета
110.              RxMessage.DLC =     0x00;
111.              RxMessage.ExtId =     0x00;
112.              RxMessage.FMI =     0x00;
113.              RxMessage.IDE =     0x00;
114.              RxMessage.RTR =     0x00;
115.              RxMessage.StdId =     0x00;
116.              RxMessage.Data [0] = 0x00;
117.              RxMessage.Data [1] = 0x00;
118.              RxMessage.Data [2] = 0x00;
119.              RxMessage.Data [3] = 0x00;
120.              RxMessage.Data [4] = 0x00;
121.              RxMessage.Data [5] = 0x00;
122.              RxMessage.Data [6] = 0x00;
123.              RxMessage.Data [7] = 0x00;
124.
125.              if (CAN_GetITStatus(CAN1, CAN_IT_FMP0) != RESET)
126.    // Проверим почтовый ящик
127.              {
128.                    CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
129.
130.                    // Проверим тип заголовка сообщения
131.                    if (RxMessage.IDE == CAN_Id_Standard) {
132.                          if (RxMessage.StdId == CAN_CMD_Test_Send)
133.    {      // Если получили тестовое сообщение
134.                                CAN_Send_Ok();
135.                                /*?candump*/
136.                          }
137.                          if (RxMessage.StdId == CAN_CMD_Test_Ok) {
138.    // Если получили подтверждение получения
139.
140.                          }
141.                    }
142.              }
143.        }
144.
145.        void initPort(void){
146.              RCC->APB2ENR |= 0x00000010; // Подключение тактового
147.    сигнала
148.
149.              GPIOA->CRH = 0x/*?GPIOA_CRH*/; // Настройка порта AH
150.              GPIOA->CRL = 0x/*?GPIOA_CRL*/; // Настройка порта AL
151.
152.              GPIOB->CRH = 0x/*?GPIOB_CRH*/; // Настройка порта BH
153.              GPIOB->CRL = 0x/*?GPIOB_CRL*/; // Настройка порта BL
154.
155.              GPIOC->CRH = 0x/*?GPIOC_CRH*/; // Настройка порта CH
156.              GPIOC->CRL = 0x/*?GPIOC_CRL*/; // Настройка порта CL
157.        }
158.
159.        void CAN_Send_Test(string text)
160.        {
161.              if (text.length() > 0)
```

```
162.            {
163.                  CanTxMsg TxMessage;
164.                  TxMessage.StdId = CAN_CMD_Test_Send;
165.    // Команда шины
166.
167.                  TxMessage.ExtId = 0x00;
168.    // Расширенную команду указывать нет смысла
169.
170.                  TxMessage.IDE = CAN_Id_Standard;
171.    // Формат кадра
172.                  TxMessage.RTR = CAN_RTR_DATA;
173.    // Тип сообщения
174.                  TxMessage.DLC = 3;
175.    // Длина блока данных 3 - передадим три байта
176.                  txt_data = Encoding.ASCII.GetBytes(text);
177.                  TxMessage.Data[0] = txt_data % 256;
178.          // Байт данных №1
179.                  txt_data = txt_data >> 8;
180.                  TxMessage.Data[1] = txt_data % 256;
181.    // Байт данных №2
182.                  txt_data = txt_data >> 8;
183.                  TxMessage.Data[2] = txt_data % 256;
184.          // Байт данных №3
185.
186.                  CAN_Transmit(CAN1, &TxMessage);
187.            }
188.      }
189.
190.      bool AND(bool a, ...)
191.      {
192.            for (bool *ptr = &a; *ptr!=null; ptr++){
193.                  if *ptr == false
194.                        return false;
195.            }
196.            return true;
197.      }
198.
199.      bool OR(bool a, ...)
200.      {
201.            for (bool *ptr = &a; *ptr!=null; ptr++){
202.                  if *ptr == true
203.                        return true;
204.            }
205.            return false;
206.      }
207.
208.
209.      bool NAND(bool a, ...)
210.      {
211.            for (bool *ptr = &a; *ptr!=null; ptr++){
212.                  if *ptr == true
213.                        return false;
214.            }
215.            return true;
216.      }
217.
```

```
218.        bool NOR(bool a, ...)
219.        {
220.              for (bool *ptr = &a; *ptr!=null; ptr++){
221.                    if *ptr == false
222.                          return true;
223.              }
224.              return false;
225.        }
226.
227.        bool XOR(bool a, ...)
228.        {
229.              bool result = false;
230.              for (bool *ptr = &a; *ptr!=null; ptr++){
231.                    if *ptr == true
232.                          result = !result;
233.              }
234.              return result;
235.        }
236.
237.        bool NOT(bool a, ...)
238.        {
239.              int *ptr = &a;
240.              return !(*ptr);
241.        }
242.
243.        float SUM(float a, ...)
244.        {
245.              float result = 0;
246.              for (float *ptr = &a; *ptr!=null; ptr++){
247.                    result += *ptr;
248.              }
249.              return result;
250.        }
251.
252.        float DIFF(float a, ...)
253.        {
254.              float result = a;
255.              float *ptr = &a;
256.              ptr++;
257.              result -= *ptr;
258.              return result;
259.        }
260.
261.        float DIV(float a, ...)
262.        {
263.              float result = a;
264.              float *ptr = &a;
265.              ptr++;
266.              result /= *ptr;
267.              return result;
268.        }
269.
270.        float MULT(float a, ...)
271.        {
272.              float result = a;
273.              float *ptr = &a;
```

```
274.            ptr++;
275.            result *= *ptr;
276.            return result;
277.      }
278.
279.      int MOD(float a, ...)
280.      {
281.            float result = a;
282.            float *ptr = &a;
283.            ptr++;
284.            result = result % *ptr;
285.            return result;
286.      }
287.
288.      bool BIGGER(float a, ...)
289.      {
290.            float result = a;
291.            float *ptr = &a;
292.            ptr++;
293.            return result > *prt;
294.      }
295.
296.      bool BIGGER(bool a, ...)
297.      {
298.            bool result = a;
299.            bool *ptr = &a;
300.            ptr++;
301.            return (result & !*ptr);
302.      }
303.
304.      bool LESS(float a, ...)
305.      {
306.            float result = a;
307.            float *ptr = &a;
308.            ptr++;
309.            return result < *prt;
310.      }
311.
312.      bool LESS(bool a, ...)
313.      {
314.            bool result = a;
315.            bool *ptr = &a;
316.            ptr++;
317.            return (!result & *ptr);
318.      }
319.
320.      bool EQLESS(float a, ...)
321.      {
322.            float result = a;
323.            float *ptr = &a;
324.            ptr++;
325.            return result <= *prt;
326.      }
327.
328.      bool EQLESS(bool a, ...)
329.      {
```

```
330.            bool result = a;
331.            bool *ptr = &a;
332.            ptr++;
333.            return !result;
334.      }
335.
336.      bool EQ(float a, ...)
337.      {
338.            float result = a;
339.            float *ptr = &a;
340.            ptr++;
341.            return result == *prt;
342.      }
343.
344.      bool EQ(bool a, ...)
345.      {
346.            bool result = a;
347.            bool *ptr = &a;
348.            ptr++;
349.            return (result & *ptr);
350.      }
351.
352.      bool EQBIGGER(float a, ...)
353.      {
354.            float result = a;
355.            float *ptr = &a;
356.            ptr++;
357.            return result >= *prt;
358.      }
359.
360.      bool EQBIGGER(bool a, ...)
361.      {
362.            bool result = a;
363.            bool *ptr = &a;
364.            ptr++;
365.            return result;
366.      }
367.
368.      int ROUND(float a, ...)
369.      {
370.            float result = a;
371.            return int(round(result));
372.      }
373.      int CEIL(float a, ...)
374.      {
375.            float result = a;
376.            return int(ceil(result));
377.      }
378.      int TRUNC(float a, ...)
379.      {
380.            float result = a;
381.            return int(floor(result));
382.      }
383.
384.      float IF(float a, bool b, float c, int o)
385.      {
```

```
386.            if (b)
387.                  return a;
388.            else
389.                  return c;
390.      }
391.
392.      int IF(int a, bool b, int c, int o)
393.      {
394.            if (b)
395.                  return a;
396.            else
397.                  return c;
398.      }
399.
400.      string IF(string a, bool b, string c, int o)
401.      {
402.            if (b)
403.                  return a;
404.            else
405.                  return c;
406.      }
407.
408.      char IF(char a, bool b, char c, int o)
409.      {
410.            if (b)
411.                  return a;
412.            else
413.                  return c;
414.      }
415.
416.      bool IF(bool a, bool b, bool c, int o)
417.      {
418.            if (b)
419.                  return a;
420.            else
421.                  return b;
422.      }
423.
424.      /*?global*/
425.
426.      void main()
427.      {
428.            initPort();
429.
430.            while(1){
431.                  // опрос портов
432.                  /*?inports*/
433.
434.                  // вызов сгенерированных функций
435.                  /*?functions*/
436.
437.                  // отправка сообщений
438.                  /*?sends*/
439.            }
440.      }
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
_____ О.В. Непомнящий
подпись инициалы, фамилия
«_____» _____2020 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
код и наименование направления

Среда разработки бортовых сетей на интерфейсе CAN
тема

Руководитель	24.06.20	доцент, канд. техн. наук	В. Г. Середкин
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	24.06.20		Н. Е. Зайцев
	подпись, дата		инициалы, фамилия
Нормоконтролер	24.06.20	доцент, канд. техн. наук	В. Г. Середкин
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2020