Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Автоматика и управление»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
д.т.н., профессор
_____/ Л.С. Казаринов
«____» _____2020 г.

Программно-аппаратный имитатор режимов работы термоконтроллера с цифровым интерфейсом связи

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ – 270304.2020.127. ПЗ ВКР

Руководитель проекта
старший преподаватель каф. АиУ
_____/ Канашев Е.А.
«____» _____ 2020 г.

Автор проекта
студент группы КЭ-417
_____/ Быкова И.Д.
«____» _____ 2020 г.

Нормоконтролер,
к.т.н., доцент каф. АиУ
_____/ Барбасова Т.А.
«____» _____ 2020 г.

Челябинск 2020

АННОТАЦИЯ

Быкова И.Д. Программно – аппаратный имитатор режимов работы термоконтроллера с цифровым интерфейсом связи. – Челябинск: ЮУрГУ, ВШ ЭКН; 2020, 98 с., 62 ил., 10 табл., библиографический список – 70 наим., 3 прил.

В выпускной квалификационной работе разрабатывается программно–аппаратный имитатор режимов работы термоконтроллера с цифровым интерфейсом связи. Рассматривается работа протоколов Modbus ASCII и Modbus RTU. Сначала производится разработка аппаратной части. Составляются структурная и принципиальная схемы, производится выбор элементов.

После разработки аппаратной части системы производится написание программного обеспечения, которое удовлетворяет техническому заданию. Для наглядности эксперимента данные выводятся на LCD дисплей. При написании выпускной квалификационной работы использовались следующие пакеты прикладных программ: CubeMX, TRUEStudio, Modbus Poll, Altium designer, Notepad++, Splan.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Микропроцессорные системы применяются практически во всех электронных устройствах. Устройства на основе микропроцессорной техники используются в таких отраслях как: промышленность, электроэнергетика, авиаракетостроение и т.д.

Микропроцессорная система предназначена для управления и автоматизации различных процессов и производства. Неотъемлемой частью системы является микроконтроллер, который выполняет функции сбора, обработки и управления. Любая микропроцессорная система включает в себя программное обеспечение (software) и аппаратное обеспечение (hardware). Микропроцессорная система работает с цифровой информацией, которая представлена в виде цифровых кодов и способна воспринимать числа только в двоичной системе счисления 0 или 1.

Основу микропроцессорной системы составляет микропроцессор (процессор), который выполняет функции приема, передачи, хранения и обработки информации. Остальные устройства, входящие в состав микропроцессорной системы, обслуживают процессор, помогая ему в работе.

Обязательными устройствами для создания микропроцессорной системы являются порты ввода/вывода и отчасти память. Порты ввода/вывода связывают процессор с внешним миром, обеспечивая ввод информации для обработки и вывод результатов обработки, либо управляющих воздействий. К портам ввода подключают кнопки (клавиатуру), различные датчики; к портам вывода – устройства, которые допускают электрическое управление: индикаторы, дисплеи, контакторы, электроклапаны, электродвигатели и т.д.

Память нужна в первую очередь для хранения программы (либо набора программ), необходимой для работы процессора. Программа — это последовательность команд, понятных процессору, написанная человеком. Для работы любой микропроцессорной системы необходимо разработать как программную, так и аппаратную часть. Именно разработка этих частей рассматривается в выпускной квалификационной работе.

1 ПОСТАНОВКА ЗАДАЧИ

При выполнении выпускной квалификационной работы ставились следующие задачи:

– разработка системы управления термоконтроллера, отвечающей техническому заданию;

– разработка структурной и принципиальной схем для заданного способа управления;

– выбор необходимых элементов для возможности реализации разработанных схем;

– разработка программного обеспечения для реализации микропроцессорной системы управления термоконтроллера;

– создание визуального отображения измеряемых параметров.

2 ОСНОВНЫЕ ПРИНЦИПЫ РАБОТЫ МАКЕТИРУЕМОГО ТЕРМОКОНТРОЛЛЕРА

Согласно техническому заданию необходимо разработать программу обеспечивающую имитацию работы термоконтроллера. Связь осуществляется по протоколам Modbus ASCII и Modbus RTU. Разрабатываемый термоконтроллер будет иметь LCD дисплей на котором отображается меню конфигурации и меню измерений.

Навигация по меню осуществляется с помощью кнопок “ВНИЗ” и “ВВЕРХ”.

Кнопки “ВЛЕВО” – “ВПРАВО” служат для выбора необходимого параметра в меню конфигурации.

Кнопка “ОК” служит для перехода в выбранное меню.

Кнопка “ОТМЕНА” обеспечивает выход в предыдущее меню. Например, из меню конфигурации в главное меню.

Кнопка “ПУСК” запускает систему в работу.

Кнопка “СТОП” останавливает работу системы.

При нажатии на кнопку “ПУСК” моделируется работа нагревателя. Нагреватель включается в работу. Соответственно с ПИД регулятора идет выходной сигнал, который зависит от заданных коэффициентов пропорционального звена, интегрального звена и дифференциального звена.

Когда нагреватель в работе, то температура повышается, пока не дойдет до заданной. Когда пользователь решает остановить работу системы, то нажимает на кнопку “СТОП”. Система останавливается. С регулятора перестает поступать сигнал на нагреватель. Температура нагревателя начинает снижаться.

Система передает на компьютер параметры своей работы: заданную температуру, измеренную температуру, запуск или остановку системы. Передача данных осуществляется по интерфейсам Modbus RTU, Modbus ASCII. Выбор интерфейса осуществляется в меню конфигурации. Также имеется возможность задать номер ведомого устройства.

3 РАЗРАБОТКА СТРУКТУРНОЙ И ПРИНЦИПИАЛЬНОЙ СХЕМ УПРАВЛЕНИЯ

Структурная и принципиальная схемы являются основной технической документацией, обозначают взаимодействие элементов системы между собой и необходимы для создания любого электронного устройства.

3.1 Разработка структурной схемы

Перед разработкой системы управления необходимо определиться с основными элементами электрической схемы, произвести их расчет и на основе полученных данных произвести выбор. Также составляются структурная и принципиальная схемы. Структурная схема необходима для разъяснения процессов, происходящих в отдельных функциональных цепях или в самом изделии. Она представляет собой связанные между собой условно–графические изображения. Основные элементы системы выбираются согласно техническому заданию или исходя из расчётов. В выпускной квалификационной работе будут присутствовать следующие элементы системы:

1. Микроконтроллер от компании STMicroelectronics. Микроконтроллеры STM делятся на два семейства 8ми и 32ух разрядные. Перед выбором микроконтроллера необходимо определиться с используемой периферией. Для этого составляется таблица необходимых периферийных устройств, рассчитывается необходимая частота тактирования процессора, рассчитывается количество используемых портов ввода – вывода. После чего анализируется рынок доступных микроконтроллеров и выбирается конкретная модель.

2. Преобразователь интерфейса RS – 485. Почти во все модели микроконтроллеров встроен универсальный асинхронный приемопередатчик

(UART). Для конвертации сигнала с UART в RS – 485 используются специализированные микросхемы.

3. LCD дисплей. В простых микропроцессорных системах для отображения информации используются светодиоды или семисегментные индикаторы. В настоящее время наблюдается тенденция перехода к LCD дисплеям. Это обусловлено лучшей наглядностью и возможностью отобразить сразу несколько параметров. Наиболее простыми и популярными дисплеями являются: дисплей WG12864 на базе микроконтроллера KS0107 и дисплей HD44780. Их главное отличие в представлении данных. Дисплей HD44780 представляет собой графический дисплей с разрешением 16*2 символа. LCD WG12864 имеет разрешение 128*64 пикселя и является графическим дисплеем. HD44780 проще в программировании, но способен отразить меньшее количество параметров.

4. Блок питания цепей управления. В микропроцессорных системах почти не используются трансформаторные блоки питания. Это связано в первую очередь с их большими габаритами, тяжестью, дороговизной. Им на смену пришли компактные импульсные блоки управления. Принцип их работы, следующий: сначала входное напряжение, сглаживается конденсаторами емкостного фильтра, затем происходит выпрямление напряжения с помощью диодного моста. Далее происходит преобразование выпрямленного напряжения в импульсы высокой частоты, после чего высокочастотные импульсы подаются на высоко частотный трансформатор.

5. Элементы коммутации и регулирования (тумблеры, кнопки, потенциометры).

Включают: кнопки запуска системы в работу “ПУСК” и останова системы “СТОП”, а также кнопки управления “ВВЕРХ”, “ВНИЗ”, “ВПРАВО”, “ВЛЕВО”, “ОК”. С помощью данных кнопок осуществляется навигация по меню LCD дисплея и задание необходимых параметров работы. Потенциометр используется для регулирования контрастности дисплея.

6. Кварцевый резонатор. Используется для задания тактовой частоты микроконтроллера.

После описания основных элементов системы составляем схему электрическую функциональную, для системы терморегулятора. Схема представлена на рисунке 3.1.

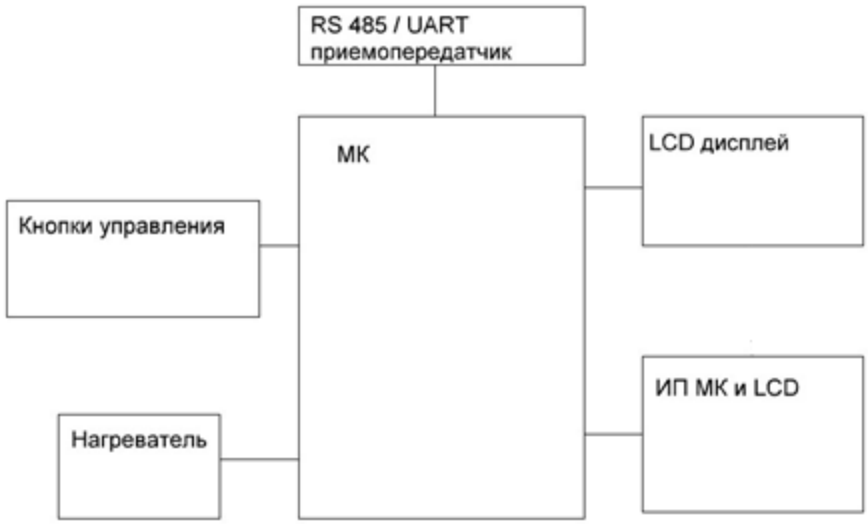


Рисунок 3.1 – Структурная схема микропроцессорной системы

На схеме: МК – микроконтроллер. ИП МК и LCD – источник питания микроконтроллера и LCD дисплея. LCD дисплей – дисплей для ввода / вывода данных. Кнопки управления – кнопки для задания параметров и управления блоком. Нагреватель – нагревательный элемент. RS – 485 / UART – конвертор интерфейсов.

3.2 Разработка принципиальной схемы

3.2.1 Выбор элементов индикации

В выпускной квалификационной работе для вывода измеряемых параметров будет использоваться LCD–дисплей WG12864. Данный дисплей обладает хорошим разрешением 128*64, также является достаточно простым и

доступным и удовлетворяет разрабатываемой системе. Внешний вид LCD–дисплея представлен на рисунке 3.2 [1].

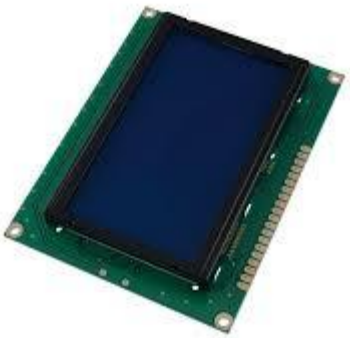


Рисунок 3.2 – Внешний вид LCD–дисплея [1]

WG12864 подключается по схеме, представленной на рисунке 3.3.

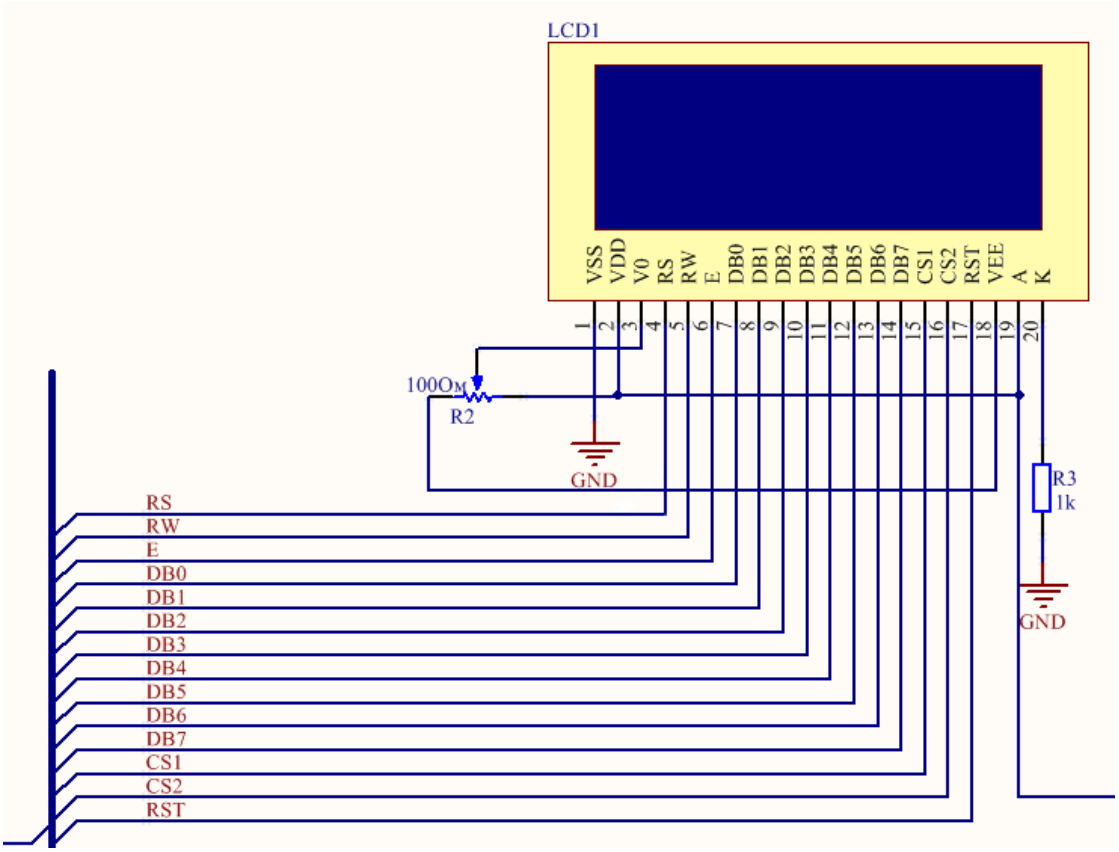


Рисунок 3.3 – Схема электрическая принципиальная по подключению LCD–дисплея.

WG12864 имеет 20 контактов ввода/вывода. В таблице 3.1 указаны назначение контактов ввода/вывода у LCD–дисплея [2].

Таблица 3.1 – Назначение контактов ввода/вывода [2]

Контакт	Назначение
VSS	Земля
VDD	Питание дисплея (5В)
V0	Напряжение регулировки контраста
RS	Сигнал сброса
RW	Чтение/запись. Тип действия определяется по уровню на контакте. 1 – чтение, 0 –запись
E	Строб, синхронизирующий импульс
CS1, CS2	Выбор контроллера
A, K	Питание светодиодной подсветки. Напрямую 5В подавать нельзя. Поэтому питание осуществляется через ограничительный резистор

Управление LCD–дисплеем осуществляется путем подачи управляющих сигналов на входы дисплея. В зависимости от состояния контактов ввода/вывода дисплей определяет команду, которую необходимо выполнить. Список команд представлен в приложении А.

3.2.2 Выбор преобразователя UART / RS – 485

UART – полнодуплексный интерфейс, то есть приемник и передатчик могут работать одновременно, независимо друг от друга. За каждым из них закреплен порт – одна ножка контроллера. Порт приемника обозначают RX, передатчика – TX. Последовательной установкой уровней на этих портах относительно общего провода ("земли") и передается информация. По умолчанию передатчик устанавливает на линии единичный уровень. Передача начинается посылкой бита с нулевым уровнем (старт–бита), затем идут биты

данных младшим битом вперед (низкий уровень – "0", высокий уровень – "1"), завершается посылка передачей одного или двух битов с единичным уровнем (стоп–битов).

Электрический сигнал кадра посылки представлен на рисунке 3.4 [3].

Рисунок 3.4 – Формат кадра UART [3]

Сеть, построенная на интерфейсе RS – 485 представляет собой приемопередатчики, соединенные между собой витой парой – двумя скрученными проводами. Данные передаются сразу по двум проводам. По первому проводу идет оригинальный сигнал, а по второму проводу его инверсная копия. Т.е. если на одном проводе 0, то на другом логическая 1. Данный тип связи очень устойчив к синфазным помехам. Т.к. если на одну линию наведется помеха, то и на вторую линию она тоже наведется, но разность потенциалов при этом не изменится и данные дойдут без повреждения информации.

Изменение сигнала в линии RS485 показано на рисунке 3.5 [4].

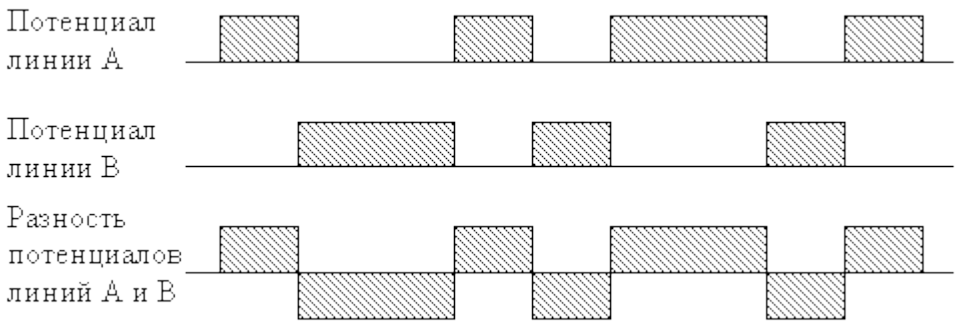


Рисунок 3.5 – Временные диаграммы RS485 [4]

Для преобразования интерфейса UART в RS – 485 будем использовать микросхему ADM 2483. Схема данной микросхемы представлена на рисунке 3.6 [5].

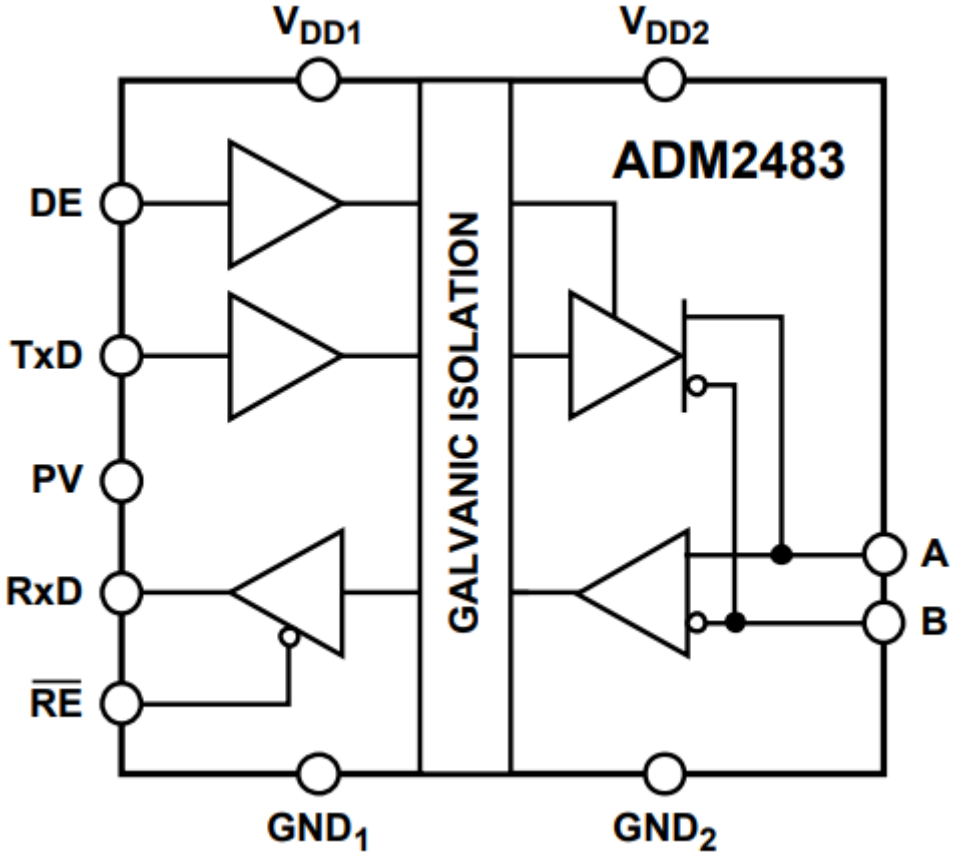


Рисунок 3.6 – Схема ADM2483 [5]

На данной схеме видим, что сигналы RxD, TxD преобразовываются в сигналы A и B интерфейса RS485. Данная схема в корпусе имеет 16 контактов. Расположение контактов микросхемы показано на рисунке 3.7 [6].

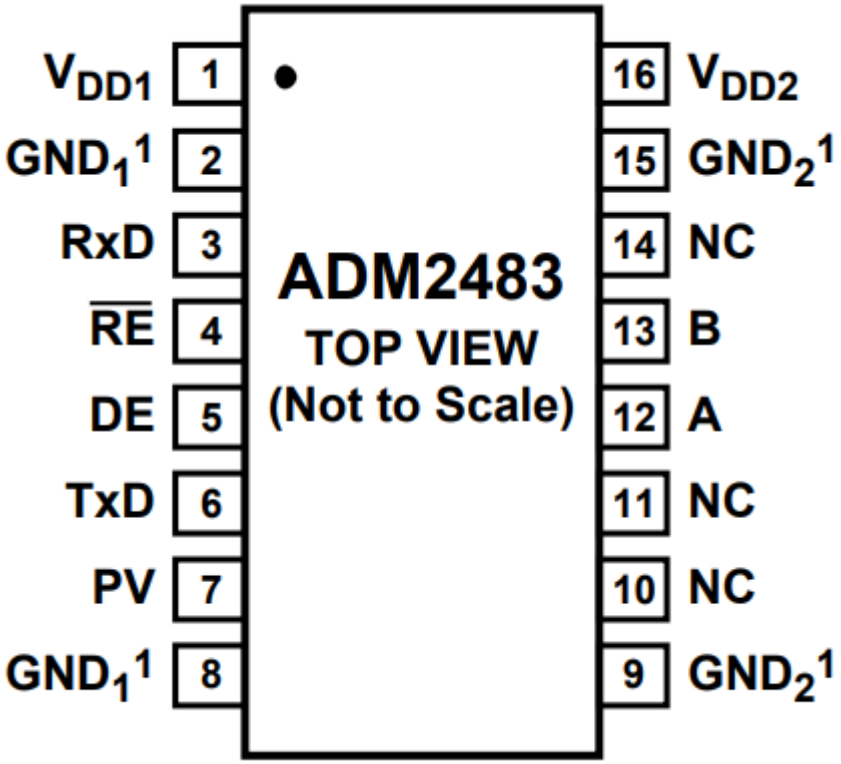


Рисунок 3.7 – Контакты микросхемы ADM2483 [6]

Все входы – выходы микросхемы представлены в таблице 3.2 [7].

Таблица 3.2 – Входы – выходы микросхемы ADM2483 [7]

VDD1	Напряжение питания (3.3 В)
GND1	Земля
RxD	Линия RxD интерфейса UART
RE	Разрешение на прием данных (активно, когда на контакте логический ноль)
DE	Разрешение на передачу данных
TxD	Линия TxD интерфейса UART

Окончание таблицы 3.2

PV	Контакт, используемый во время включения и выключения
VDD2	Напряжение питания (5 В)
GND2	Земля
NC	Неиспользуемый контакт
B	Линия B интерфейса RS485
A	Линия A интерфейса RS485

Как видно из описания контактов входа – выхода микросхема обладает полудуплексным типом связи. Контакты RE и DE можно соединить перемычкой, так как они управляются противоположными логическими сигналами и вывести на отдельный контакт микроконтроллера, который будет отвечать за прием или передачу информации.

3.2.3 Выбор потенциометров

Для регулировки контраста дисплея используем потенциометр на 10 КОм. Технические данные потенциометра представлены в таблице 3.3 [8]. Выбираем потенциометр ППБ–16Б.

Таблица 3.3 – Параметры потенциометра ППБ–16Б [8]

Тип	Переменный
Модель	ППБ
Номинальное сопротивление	10 КОм
Точность %	10
Номинальная мощность	16Вт
Максимальное рабочее напряжение	500В
Количество оборотов	1
Рабочая температура	–65...155

Потенциометр соединен с контактом V0 дисплея и осуществляет регулировку контрастности.

3.2.4 Выбор управляющего устройства. STM32F429ZI

В настоящее время на рынке представлено множество моделей микроконтроллеров. Основные семейства – это AVR, ARM, 8051 и PIC. Каждый из этих контроллеров обладает своими достоинствами и недостатками. Для определения семейства контроллеров для курсового проекта сравним их по ряду признаков.

В таблице 3.4 представлены различия между контроллерами AVR, ARM, 8051 и PIC [9,10,11, 12, 13, 14].

Таблица 3.4 – Сравнение основных семейств микроконтроллеров [9,10,11, 12, 13, 14]

	8051	PIC	AVR	ARM
Разрядность	8 бит	8/16/32 бит	8/32 бит	8/32/64 бит
Интерфейсы	UART, USART, SPI, I2C	UART, USART, LIN, CANN, SPI, I2C	UART, USART, SPI, I2C, USB	UART, USART, LIN, CAN, I2C, I2S, DSP, SAI
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	RISC	RISC	RISC

Окончание таблицы 3.4

	8051	PIC	AVR	ARM
Архитектура памяти	Фон–Неймановская	Гарвардская	Гарвардская	Гарвардская
Энергопотребление	Среднее	Низкое	Низкое	Низкое
Семейства	Вариации 8051	PIC16, PIC17, PIC18, PIC24, PIC32	Tiny, Mega, XMega	ARMv4,5,6,7
Стоимость	Низкая	Средняя	Средняя	Низкая
Популярные микроконтроллеры	AT89C51, P89v51	PIC18fxx8, PIC32Mxx	ATmega8,16, 32, вариации для Arduino	STM8, STM32

Все представленные семейства микроконтроллеров удовлетворяют требованиям выпускной квалификационной работы. Делаем выбор исходя из цены. Низкой ценой обладают два семейства: 8051 и ARM. Но у семейства ARM скорость работы намного выше. Для снятия графиков и написания программы будет использоваться популярный микроконтроллер STM32F429ZI. Для выполнения выпускной квалификационной работы подойдет более дешевый контроллер, но в целях удобства создания макетной установки будет использоваться STM32F429ZI. Также будет использоваться отладочная плата STM32F429 discovery board. Внешний вид платы представлен на рисунке 3.8 [15].

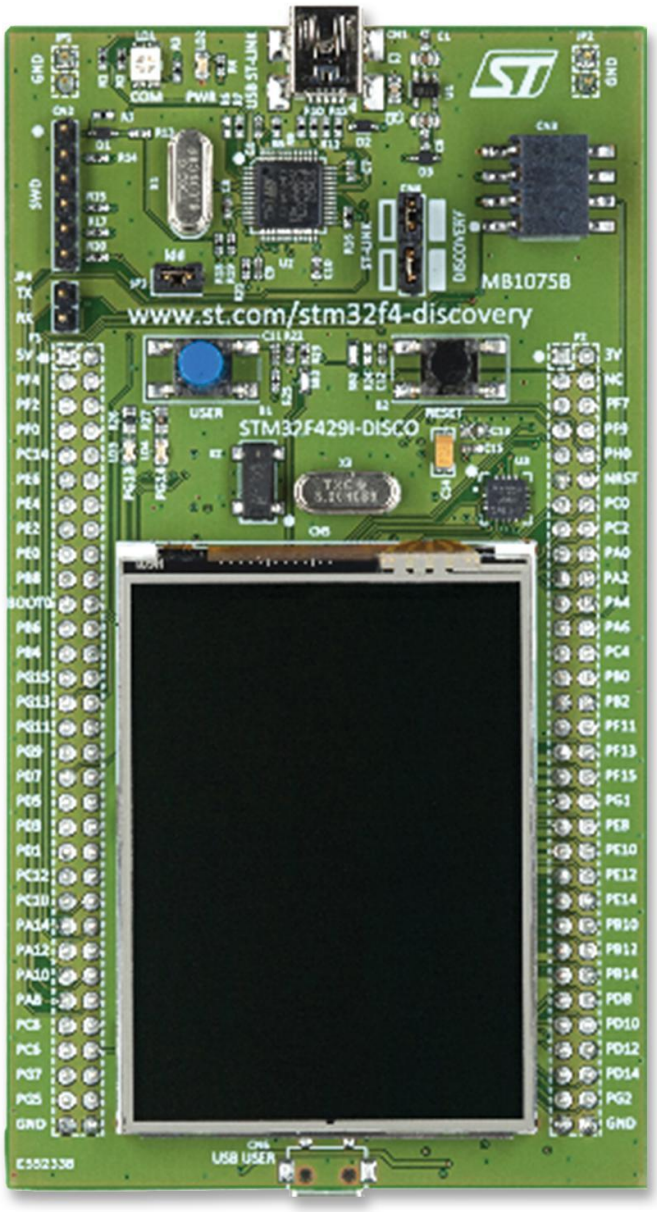


Рисунок 3.8 – STM32F429 discovery board [15]

Данная плата обеспечивает возможность подключения осциллографа и логического анализатора к каждому контакту ввода/вывода микроконтроллера. Также в STM32F429 discovery board встроен программатор ST–LINK. Программатор подключается к микроконтроллеру с помощью специализированных контактов. Функции контактов программатора представлена в таблице 3.5 [16].

Таблица 3.5 – Функции контактов ST_LINK [16]

Контакт	Название	Описание
1	VDD_Target	Напряжение питания
2	SWCLK	Тактирование интерфейса SWD
3	GND	Земля
4	SWDIO	SWD ввод/вывод
5	NRST	Сброс микроконтроллера
6	SWO	Зарезервировано

На отладочной плате имеется возможность прошивки сторонних микроконтроллеров STM. Необходимо изменить конфигурацию перемычек, отвечающих за программирование по интерфейсу SWD и к выводам подключить нужный микроконтроллер.

3.2.5 Выбор источника питания микроконтроллера

Для микроконтроллера выбираем источник напряжения фирмы «Mean Well» PS–05–5, данные которого приведены в таблице 3.6 [17].

Таблица 3.6 – Технические параметры источника питания PS–05–5 [17]

Параметр	Значение
Входное напряжение	85–264 В AC / 120–370 В DC
Выходное напряжение	5 В
Выходной ток	1 А
Мощность	5 Вт

3.2.6 Выбор керамических конденсаторов

При внешнем источнике тактирования конденсаторы устанавливаются рядом с кварцевым резонатором согласно схеме, изображенной на рисунке 3.9 [18].

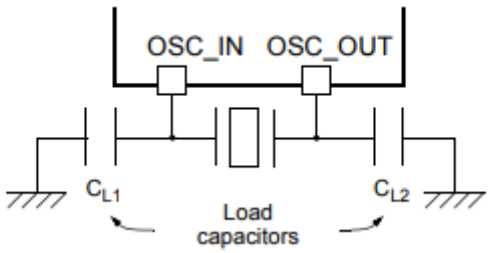


Рисунок 3.9 – Кварцевые резонаторы при внешнем источнике тактирования [18]

Согласно официальной документации на микроконтроллер STM32F429ZI данные конденсаторы должны быть емкостью 0,22 мкФ [19].

В электрических схемах необходимо устанавливать конденсатор между питанием и землей. Это делается для повышения помехоустойчивости системы. Выберем керамический конденсатор на выход источника питания 5 В. Технические данные приведены в таблице 3.7 [20,21].

Таблица 3.7 – Технические данные керамического конденсатора [20,21]

Параметр	Значение
Рабочее напряжение, В	16
Ёмкость, мкФ	0,22
Диапазон $T_{РАБ}$, ºС	–40…+85

В выпускной квалификационной работе составляется система малой мощности и будут использоваться резисторы мощностью 0,5 Вт.

3.2.7 Выбор кварцевого резонатора

Выберем кварцевый резонатор типа HC–49S, частота которого равна 8 МГц. Технические параметры приведены в таблице 3.8 [22].

Таблица 3.8 – Параметры кварцевого резонатора HC–49S [22]

Параметр	Значение
Резонансная частота, МГц	8 МГц
Точность настройки dF/Fx10–6	50
Корпус	Hs–49s
Паразитная ёмкость кварцедержателя в испытательной схеме, С	32 пФ
Рабочая температура, С	–20…70

Кварцевый резонатор необходим для тактирования микроконтроллера.

3.2.8 Выбор подтягивающих резисторов

Имеется несколько вариантов соединения кнопки с управляющим контактом микроконтроллера. Наиболее предпочтительно использовать подтягивающие резисторы. Без использования подтягивающих резисторов на управляющем контакте может установиться неустойчивое состояние. Т.е. состояние, когда неизвестно какой потенциал прикладывается к контакту. Из–за неустойчивого состояния возможны ложные срабатывания кнопок. Для устранения данного эффекта используются подтягивающие резисторы, которые обеспечивают стабильное состояние контакта в момент, когда кнопка разомкнута. На рисунке 3.10 показано подключение, используемое для подтяжки кнопки к логической 1 или логическому 0 [23,24].

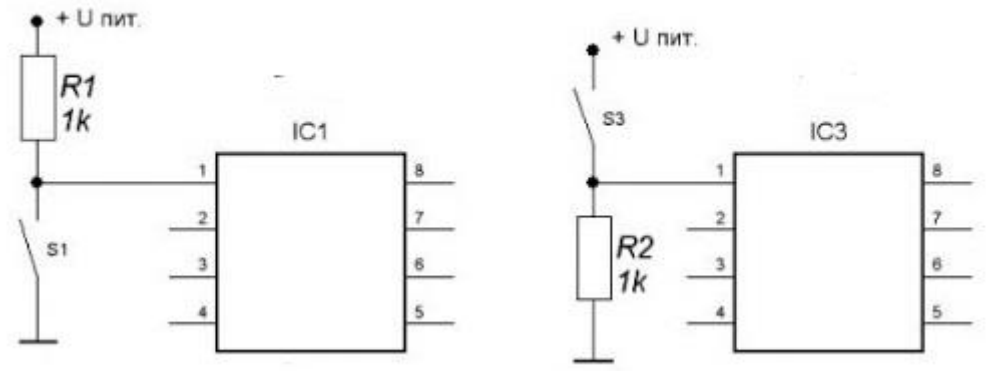


Рисунок 3.10 – Виды подключения кнопок с подтягивающими резисторами [23,24]

3.2.9 Выбор кнопок управления термоконтроллером

Задание управляющих сигналов: “ПУСК”, “СТОП”, “ОК”, “ВЛЕВО”, “ВПРАВО”, “ВНИЗ”, “ВВЕРХ”, “ОТМЕНА”, осуществляется с помощью кнопок или двухпозиционных тумблеров. При коммутации сигнал подается на контакт контроллера, отвечающий за работу системы при данном нажатии. Необходимо учитывать, что при коммутации контактов кнопки возникает дребезг контактов. Есть два способа убрать дребезг:

1) выбрать кнопку с защитой от дребезга. Работа таких кнопок, как правило, основывается на эффекте Холла.

2) программное устранение дребезга.

Т.к. бездребезговые кнопки достаточно дорогие, будем устранять дребезг контактов программно.

Выбираем кнопку PBC10C–2 red [25]. Кнопка предназначена для коммутации слаботочных цепей. Технические характеристики кнопки приведены в таблице 3.9.

Таблица 3.9 – Технические характеристики кнопки РВС10С–2 [25].

Параметр	Значение
Цвет кнопки	Красный
Фиксация	Нет
Функциональное назначение	Кнопочный выключатель
Количество контактных групп	1
Алгоритм работы	On–off
Рабочее напряжение, В	250
Рабочий ток, А	1
Способ монтажа	На панель

3.3 Описание принципиальной схемы системы управления

Нагреватель является объектом управления. Его выход подключен к выходу PD12. На данном выходе можно сформировать ШИМ сигнал для плавной работы нагревателя. Микросхема ADM2483 подключена к выходам PB6, PB7, PA4. Выход PB6 является выходом UART1_Tx микроконтроллера STM32F429ZI. Выход PB7 является выходом UART1_Rx. Через PB6 и PB7 обеспечивается обмен данными между микроконтроллером и внешним миром. Контакт PA4 отвечает за режим передачи: прием, передача. LCD дисплей соединен с контактами микроконтроллера PD0 – PD7. Данные контакты отвечают за прием/передачу данных. Управляющие сигналы от LCD поступают на входы PG8 – 12 и PG15. Кнопки подтянуты к земле. Соединены с контактами PE0 – PE7. Кварцевый резонатор на 8 КГц соединен с контактами OSC32_IN OSC32_OUT. Входы boot1, boot0, T_NRST, SWDIO, SWCLK необходимы для прошивки микроконтроллера.

На рисунке 3.11 представлена принципиальная схема для систем управления.

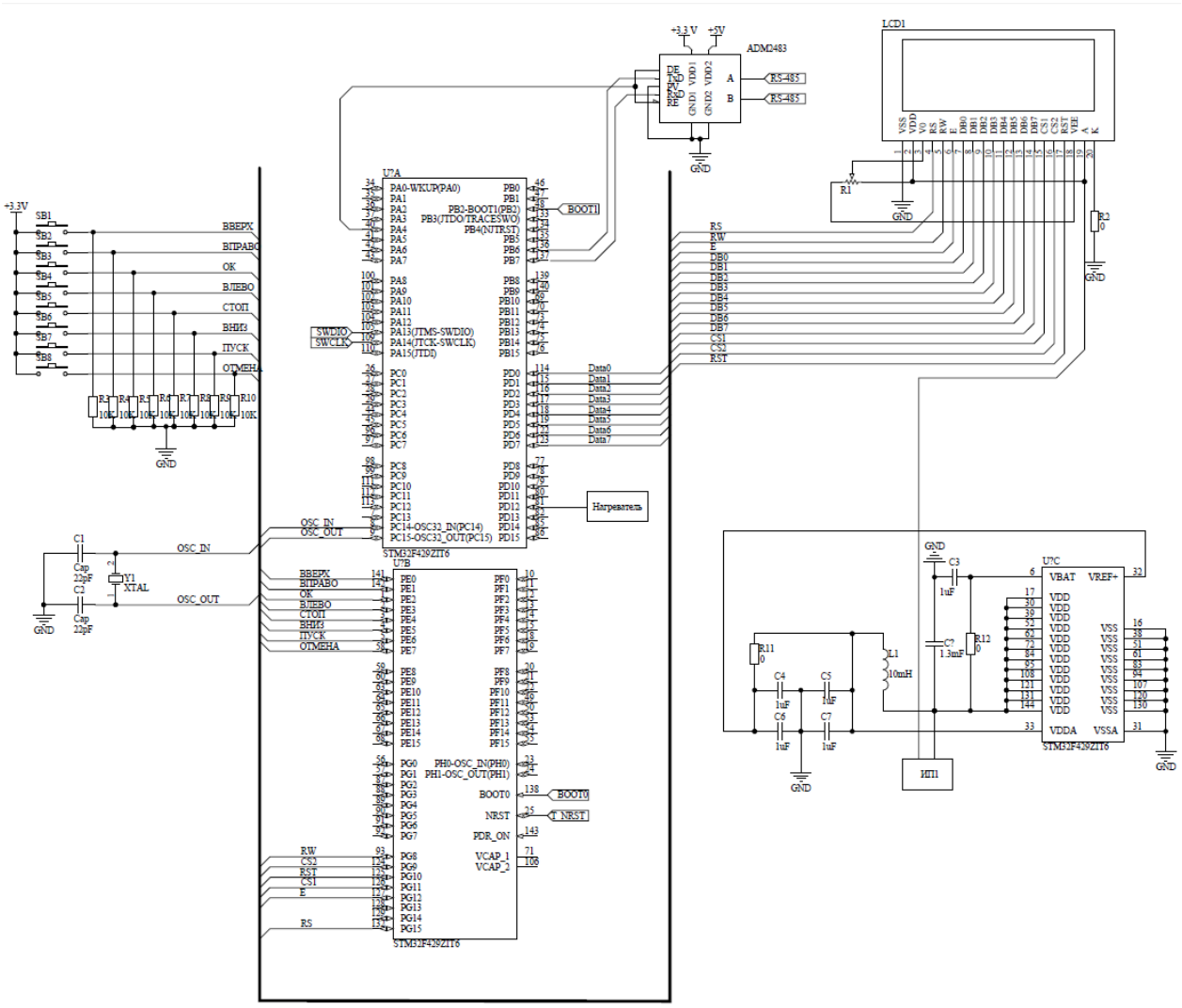


Рисунок 3.11 – Принципиальная схема

Составим таблицу входов – выходов разрабатываемого устройства (см. таблицу 3.10).

Таблица 3.10 – Используемые порты ввода вывода микроконтроллера

Название выхода	Функция
PE0	Кнопка “ВВЕРХ”
PE1	Кнопка “ВПРАВО”
PE2	Кнопка “ОК”
PE3	Кнопка “ВЛЕВО”
PE4	Кнопка “СТОП”
PE5	Кнопка “ВНИЗ”

Окончание таблицы 3.10

Название выхода	Функция
PE6	Кнопка “ПУСК”
PE7	Кнопка “ОТМЕНА”
PG8	RW
PG9	CS2
PG10	RST
PG11	CS1
PG12	E
PG15	SR
PD0	Данные дисплея, вход 0
PD1	Данные дисплея, вход 1
PD2	Данные дисплея, вход 2
PD3	Данные дисплея, вход 3
PD4	Данные дисплея, вход 4
PD5	Данные дисплея, вход 5
PD6	Данные дисплея, вход 6
PD7	Данные дисплея, вход 7
PD12	ШИМ выход на нагреватель
PB6	USART1_Tx
PB7	USART1_Rx
PA14	Разрешение приема/ передачи
SWCLK	Интерфейс SWD
SWDIO	Интерфейс SWD
GND	Земля
N_RSET	RESET MCU
OSC_IN, OSC_OUT	Кварцевый резонатор
VDD	Питание
BOOT0, BOOT1	Контакты прошивки процессора

4 ИСПОЛЬЗУЕМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

4.1 Используемая среда разработки

В выпускной квалификационной работе использовалась бесплатная среда TRUEStudio. TRUEStudio – это интегрированная среда разработки программного обеспечения микроконтроллеров STM, находится в свободном доступе [26], а также регулярно обновляется и поддерживается официальным производителем микроконтроллеров – фирмой STMicroelectronics внешний вид данной среды представлен на рисунке 4.1.

Поддерживаемые микроконтроллеры. TrueStudio работает только с STM32 и поддерживает все микроконтроллеры семейства. Кроме того, в TrueStudio есть поддержка большинства плат от STMicroelectronics. Пользователь может открыть готовые демонстрационные проекты без скачивания каких–либо дополнительных файлов.

Работа с файлами. TrueStudio предлагает к услугам пользователей стандартный набор инструментов для работы с C/C++ файлами: поиск, интерактивный поиск, контекстную подсветку, шаблоны, дерево функций и т.д.

Компиляция и построение проекта. Как было сказано выше, TrueStudio использует GCC для компиляции проекта. При этом возможна оптимизация кода в процессе компиляции.

Поддержка систем контроля версий. TrueStudio обеспечивает одновременную работу нескольких пользователей над проектом за счет поддержки систем контроля версий: CVS, SVN, Git.

Отладка. TrueStudio поддерживает работу с использованием всех популярных отладчиков, в том числе, ST–Link, SEGGER, P&E micro и др.

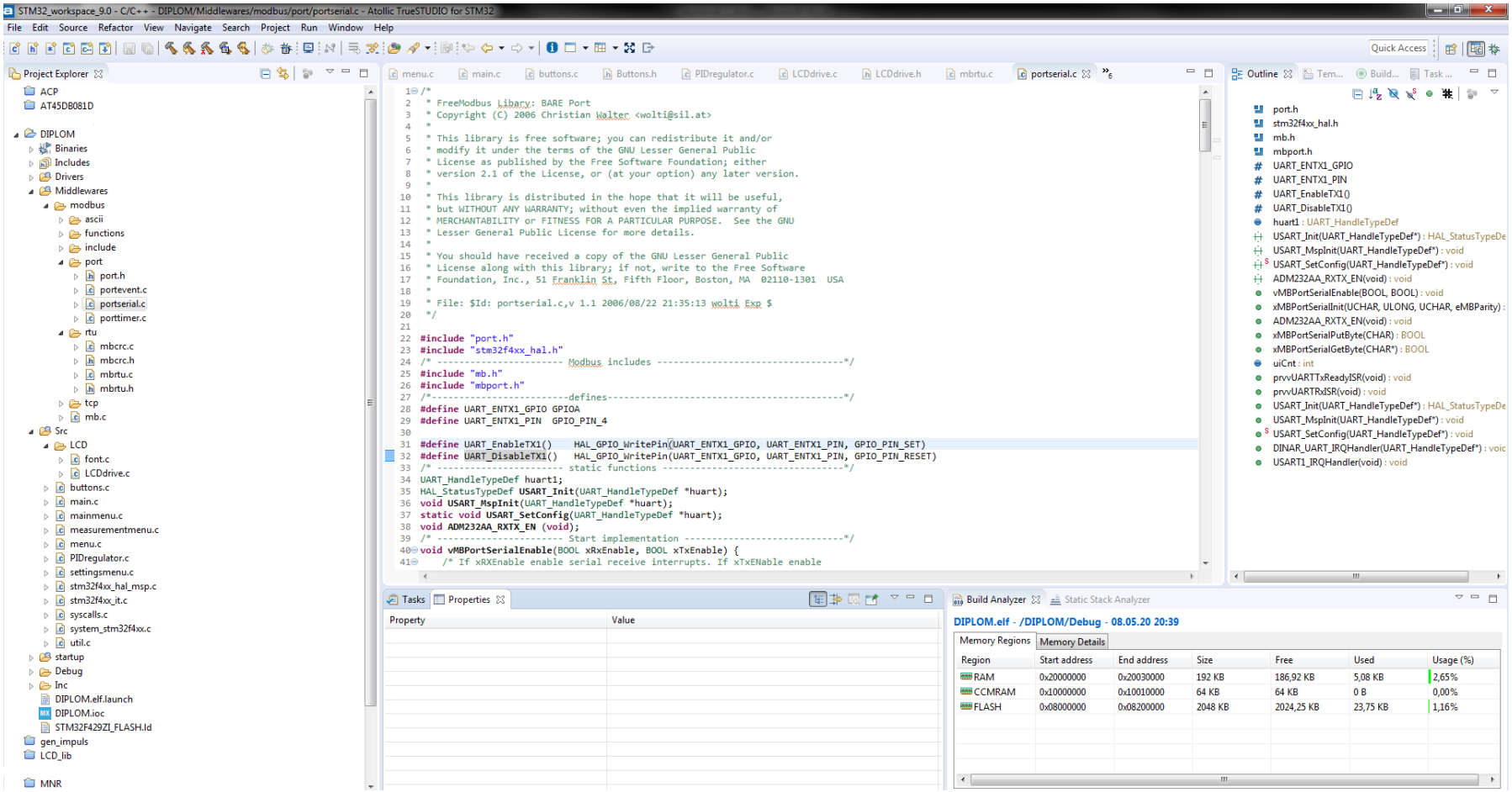


Рисунок 4.1 – Среда разработки TRUEStudio

Как видно из рисунка 4.1. Слева представлено дерево проекта. В центре находится панель редактирования файлов. Справа снизу показана свободная область памяти микроконтроллера.

4.2 Используемые вспомогательные программы

Помимо среды разработки в выпускной квалификационной работе использовались следующие программы:

1) CubeMX – Среда для упрощенной инициализации периферии. Имеется возможность настройки любого периферийного узла и автоматической генерации кода в различные среды разработки. Также имеется возможность настройки системы тактирования микроконтроллера. Рассмотрим настройку периферии в CubeMX. Схема тактирования для термоконтроллера показана на рисунке 4.2.

Как видно из рисунка используется тактирование от внешнего кварцевого резонатора (HSE). Внешний кварц имеет рабочую частоту 8 МГц. Далее тактовый сигнал идет на делитель, а затем на узел PLL задача которого гибко настроить необходимую частоту тактирования. После узла PLL тактовый сигнал идет на шины APB1, APB2 [28]. На данных шинах находится необходимая периферия. Соответственно тактовая частота таймеров будет равна 60 МГц. Внутренний тактовый генератор не используем т.к. у него достаточна большая ошибка тактирования и он не обладает достаточной стабильностью.

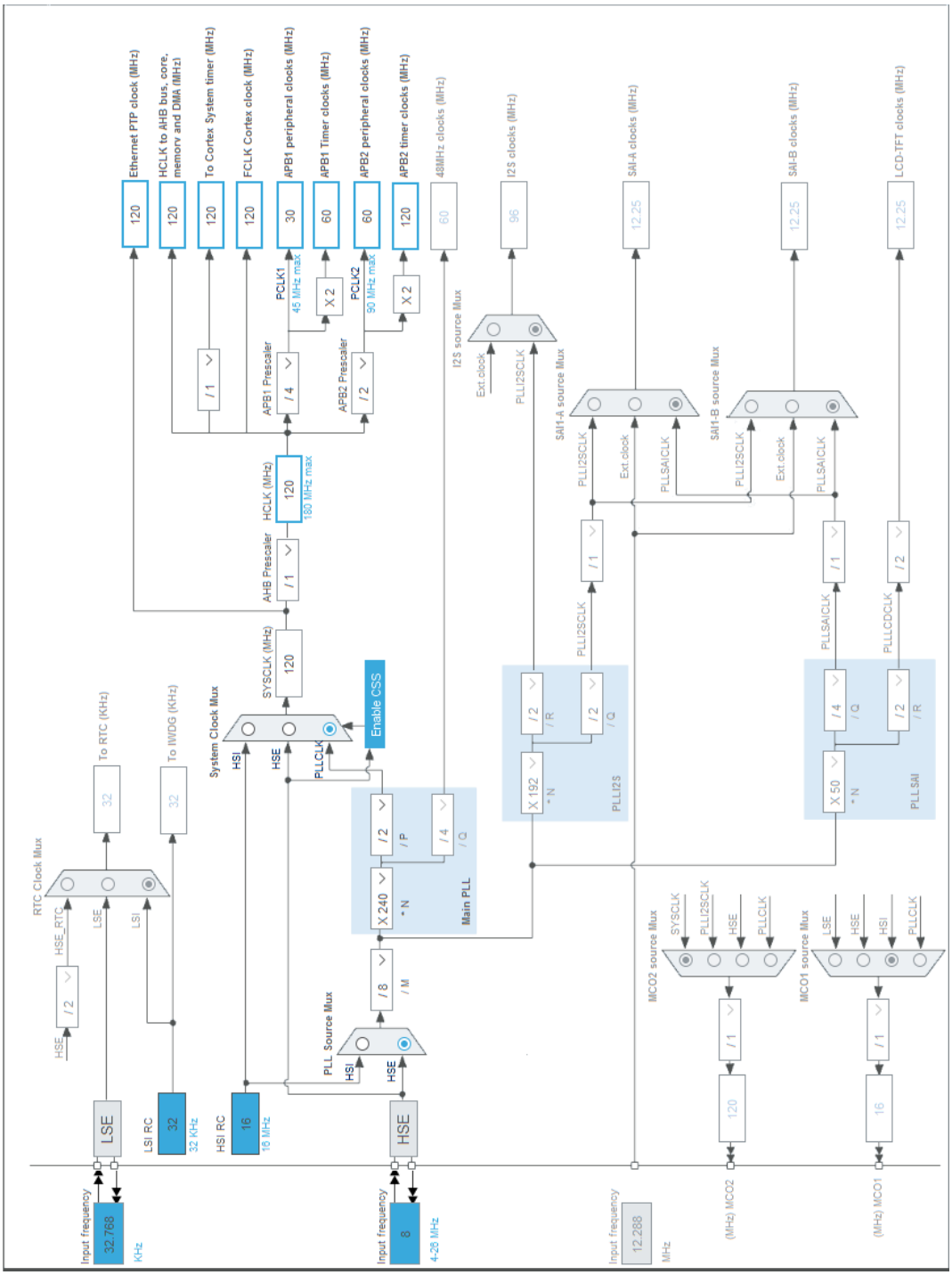


Рисунок 4.2 – Схема тактирования микроконтроллера

Рассмотрим окно настройки работы контактов ввода вывода в том или ином режиме. На рисунке 4.3 показана возможность настройки контакта PB6 на работу в том или ином режиме.

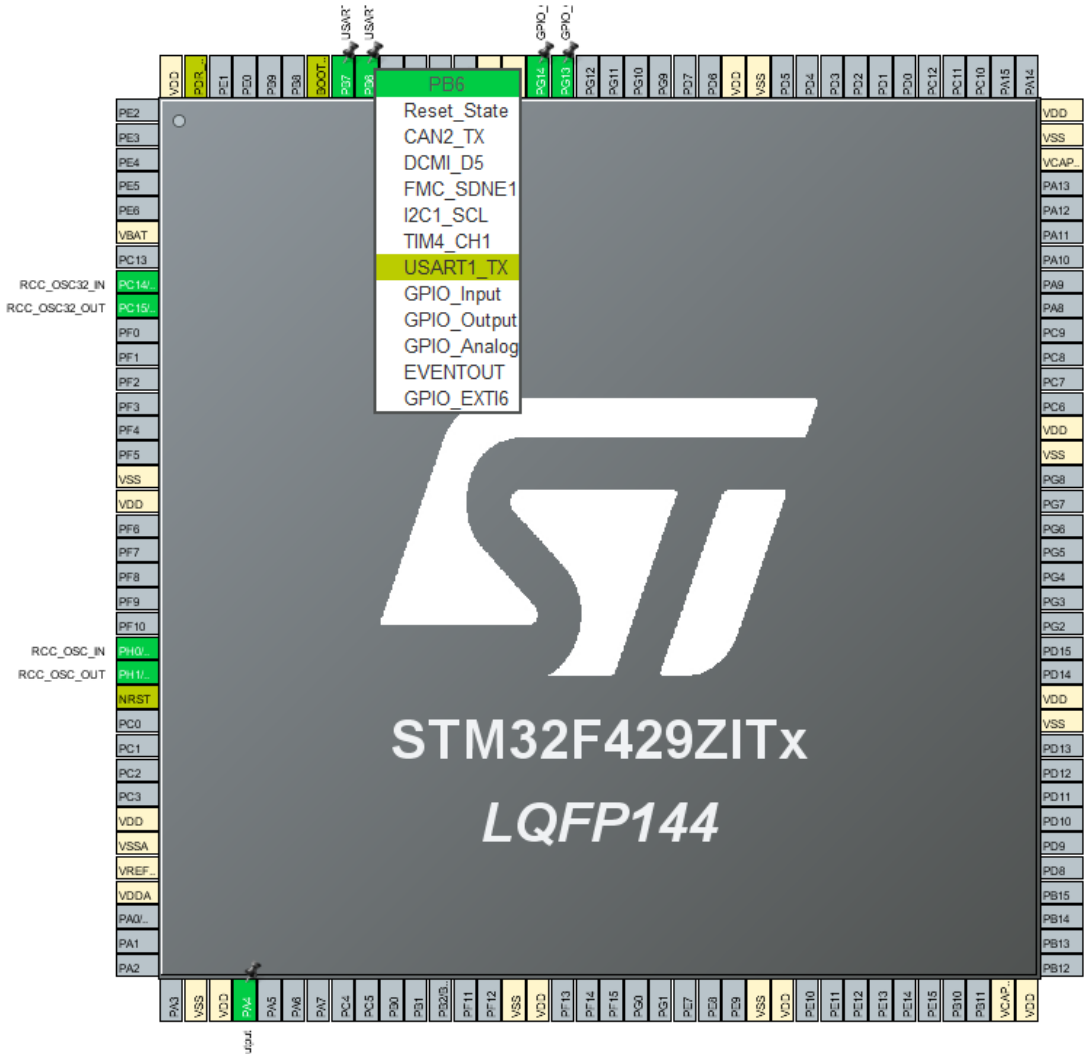


Рисунок 4.3 – Настройка необходимых контактов ввода / вывода.

Как видно из рисунка каждый контакт микроконтроллера может работать в различных режимах. Т.е. контакт PB6 можно настроить как контакт ввода, вывода, как тактовую линию интерфейса I2C, как внешнее прерывание и как вывод USART1_Tx [29].

Рассмотрим окно настройки периферии на примере интерфейса USART1. Данное окно представлено на рисунке 4.4.

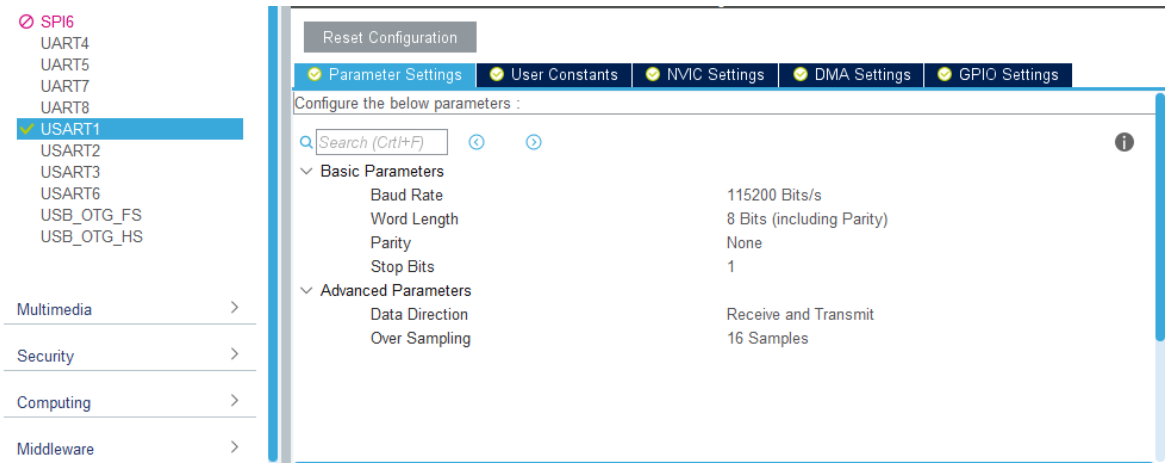


Рисунок 4.4 – Настройка интерфейса USART1

Интерфейс настроен на скорость передачи данных 115200 бод. Формат посылки 8 бит. Проверка четности отсутствует. Количество стоп битов – 1.

После настройки всей необходимой периферии имеется возможность сгенерировать код инициализации сразу в выбранную IDE. При этом надо указать имя создаваемого проекта, глубину стека, IDE в которой будет создан проект.

2) Altium Designer — в выпускной квалификационной работе используется для составления принципиальной схемы разрабатываемого устройства.

3) Notepad++ – текстовый редактор. Используется для редактирования листинга программы.

4) Modbus Poll – программа эмулятор интерфейса Modbus. Имеет возможность настройки запросов от ведущего устройства (master) к ведомому (slave). В роле ведущего устройства выступает персональный компьютер пользователя. В соответствии с заданными настройками программа генерирует запросы от мастера к ведомому по протоколам Modbus ASCII, Modbus RTU. Окно настройки опросов ведомого устройства показано на рисунке 4.5. Используется для отладки связи между ведущим и ведомым устройствами.

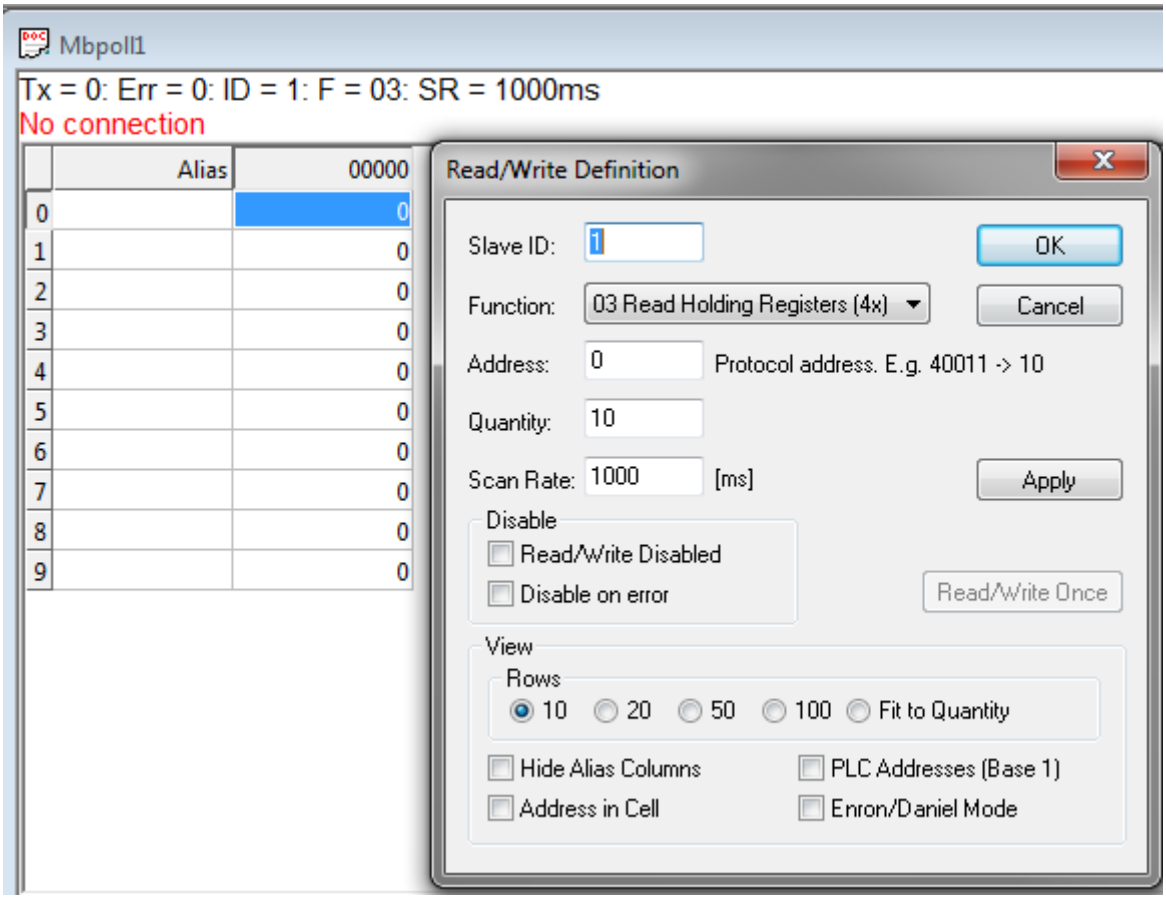


Рисунок 4.5 – Настройка опросов ведомого устройства

Если соединение с ведомым отсутствует, то появляется ошибка – “no connection”.

5) Terminal. Данная программа выводит на com порт компьютера заданное число с необходимой скоростью. Также у нее имеется возможность принимать данные со стороннего устройства. На рисунке 4.6 представлен интерфейс программы terminal.

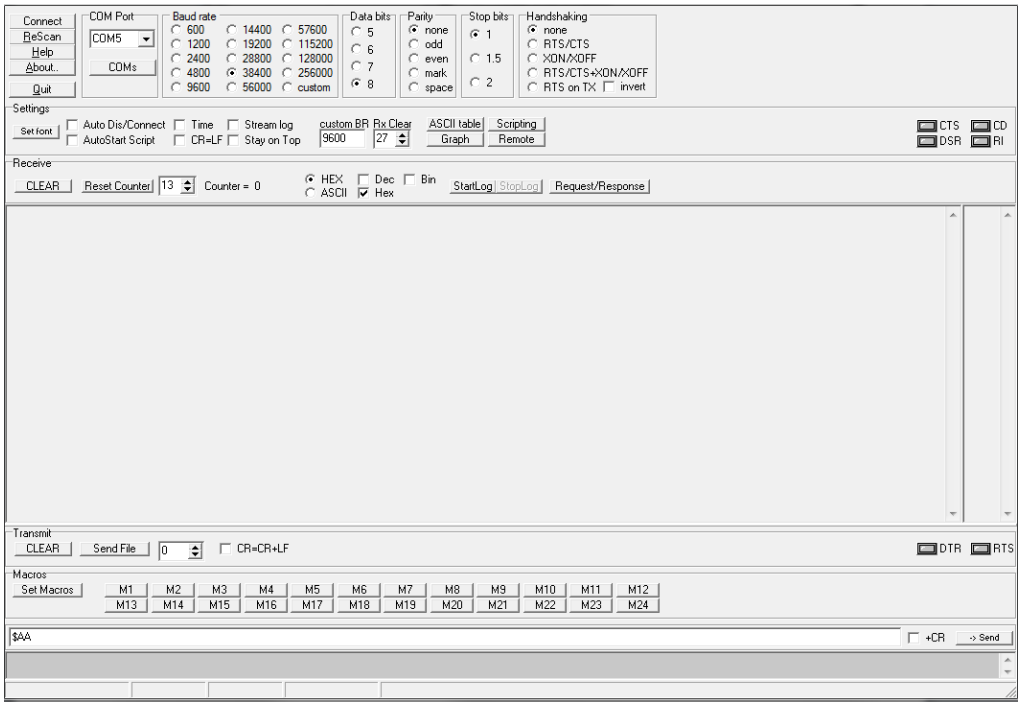


Рисунок 4.6 – Интерфейс программы terminal

Как видно из рисунка имеется выбор com порта компьютера, также настраивается скорость передачи данных. Данные скорости являются типовыми и соответствуют стандарту Modbus. Также настраивается бит четности и количество стоп–битов. Программа необходима для отладки при портировании библиотеки freemodbus.

5 РАЗРАБОТКА И ИССЛЕДОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМЫ УПРАВЛЕНИЯ ТЕРМОКОНТРОЛЛЕРОМ

5.1 Портирование библиотеки freemodbus для микроконтроллера STM32

Библиотека freemodbus является свободно распространяемым программным обеспечением и поддерживает многие типы микроконтроллеров. К примеру: AVR, STR71X, MCF5235 и другие [31]. Готового примера для микроконтроллеров фирмы STMicroelectronics в данной библиотеки нет.

Портирование библиотеки заключается в следующем:

1) необходимо изменить файл port.h [32].

2) изменить файл portserial.c [33]. Данный файл отвечает за интерфейс UART, который использует микроконтроллер. У микроконтроллеров разных фирм различная архитектура и регистры, отвечающие за управление приёмо–передатчиком, различаются. В данном файле будем прописывать инициализацию интерфейса USART1 для микроконтроллера STM32F429ZI.

3) изменить файл porttimer.h [34]. Данный файл отвечает за таймер, который отсчитывает временные интервалы при работе Modbus RTU или Modbus ASCII.

Сначала необходимо загрузить библиотеку в файлы проекта и прописать путь к папкам в директории проекта. Подключение путей include файлов показано на рисунке 5.1.

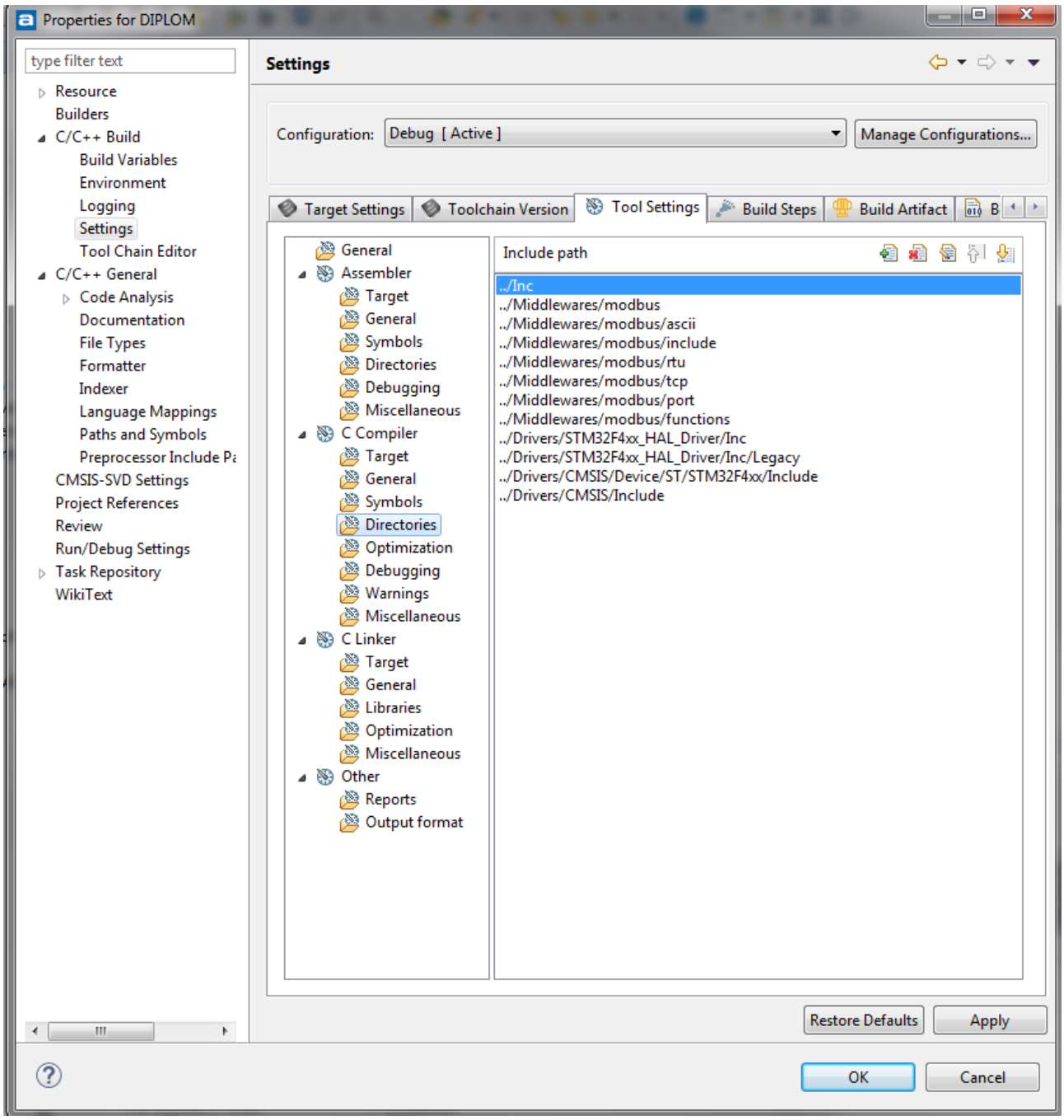


Рисунок 5.1 – Подключение include path

В файле port.h объявляем функции, обеспечивающие атомарность операций.

Листинг 1 – Объявление критических секций

```
#define ENTER_CRITICAL_SECTION( )    ( __critical_enter( ) )
#define EXIT_CRITICAL_SECTION( )     ( __critical_exit( ) )
```

Сами функции прописываем в файле main.c

Листинг 2 – Функции критических секций

```
void __critical_enter(void)
{
    __disable_irq();
    ++lock_nesting_count;
}
void __critical_exit(void)
{
    /* Unlock interrupts only when we are exiting the outermost
nested call. */
    --lock_nesting_count;
    if (lock_nesting_count == 0) {
        __enable_irq();
    }
}
```

Данные функции используются в местах, где недопустимо появление прерываний.

Согласно документации на библиотеку freemodbus для стека протоколов Modbus требуется таймер, чтобы определить конец кадра. Функцию инициализации таймера необходимо настроить на генерацию прерываний с интервалом 50 мкс.

Функция инициализации таймера выглядит следующим образом:

Листинг 3 – Функция инициализации таймера

```
BOOL xMBPortTimersInit(USHORT usTim1Timerout50us) {
 timeout = usTim1Timerout50us;
 __HAL_RCC_TIM5_CLK_ENABLE();
 htim5.Instance = TIM5;
 htim5.Init.Period = 49; // *usTim1Timerout50us;
 htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim5.Init.Prescaler = (HAL_RCC_GetHCLKFreq() / 1000000 /2 ) -
1; // 1мкс
 htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 HAL_NVIC_SetPriority(TIM5_IRQn, 8, 0);
 HAL_NVIC_EnableIRQ(TIM5_IRQn);
 return (HAL_OK == HAL_TIM_Base_Init(&htim5));
}
```

Выбран 5ый таймер. Данный таймер после включения будет генерировать прерывание с частотой 20 кГц.

Проверим работу таймера.

В main.c прописываем следующий код:

Листинг 4 – Проверочные функции

```
xMBPortTimersInit(1);
vMBPortTimersEnable();
```

данные функции инициализируют таймер на заданное время работы и работу таймера.

Каждое прерывание инвертирует контакт 14 порта GPIOG. Снимем осциллограмму, иллюстрирующую работу таймера. Данная осциллограмма представлена на рисунке 5.2.

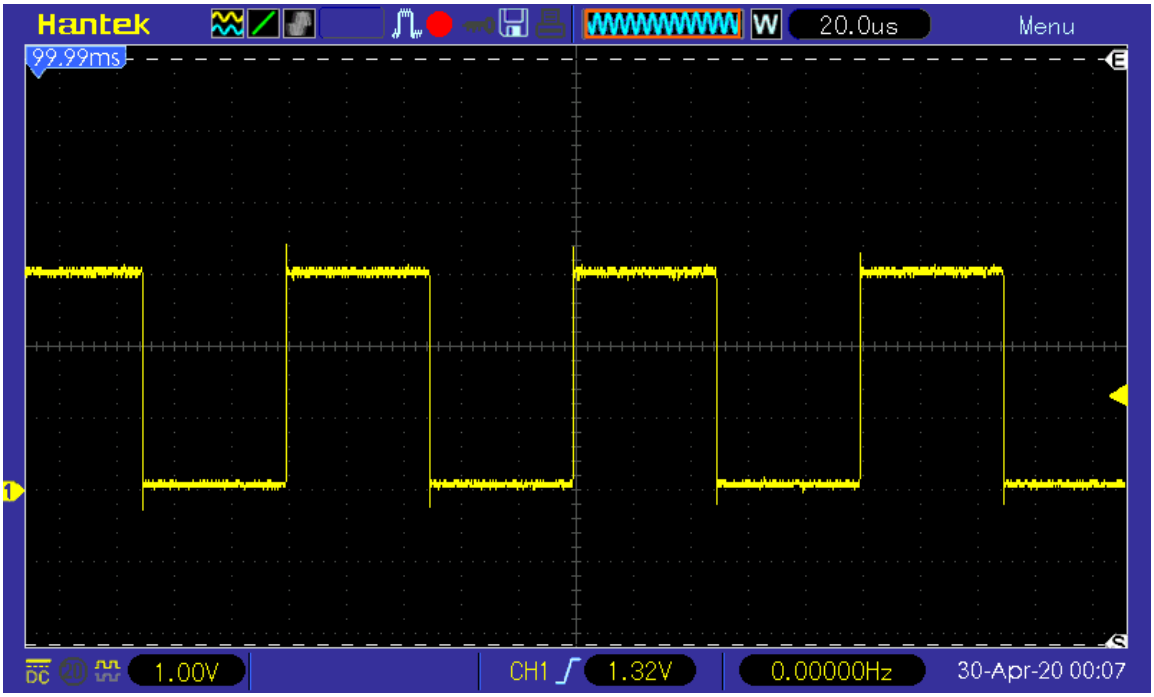


Рисунок 5.2 – Осциллограмма, иллюстрирующая работу таймера

Как видим, из осциллограммы прерывание происходит с заданной частотой. Таймер настроен.

Далее настраивается интерфейс USART1. Необходимо учитывать, что используется полудуплексная микросхема конвертации интерфейсов

ADM2483. Соответственно объявим функции, отвечающие за прием и передачу данных.

Листинг 5 – Объявление функций разрешения передачи данных

```
  #define UART_EnableTX1()  HAL_GPIO_WritePin(UART_ENTX1_GPIO,
UART_ENTX1_PIN, GPIO_PIN_SET)
  #define UART_DisableTX1() HAL_GPIO_WritePin(UART_ENTX1_GPIO,
UART_ENTX1_PIN, GPIO_PIN_RESET)
```

Соответственно функция разрешения передачи и приема приобретает следующий вид:

Листинг 6 – Функции передачи данных

```
  void vMBPortSerialEnable(BOOL xRxEnable, BOOL xTxEnable) {
      /* If xRXEnable enable serial receive interrupts. If
xTxENable enable
       * transmitter empty interrupts.
       */
      if (xRxEnable) {
        UART_DisableTX1(); // вкл ADM2483 на прием
          __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
      } else {
          __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
      }

      if (xTxEnable) {
        UART_EnableTX1(); // вкл ADM2483 на передачу
          __HAL_UART_ENABLE_IT(&huart1, UART_IT_TXE);
      } else {
          __HAL_UART_DISABLE_IT(&huart1, UART_IT_TXE);
      }
  }
```

Как видно из кода при разрешении приема отключается передача и микросхема настраивается на прием информации.

При включении передачи отключается прием данных, и микросхема настраивается на передачу данных.

Функция инициализации интерфейса USART1 выглядит следующим образом:

Листинг 7 – Функция инициализации UART

```
   BOOL xMBPortSerialInit(UCHAR ucPORT, ULONG ulBaudRate, UCHAR
ucDataBits, eMBParity eParity) {
    ADM232AA_RXTX_EN();
        switch (ucPORT) {
            case 0:
             huart1.Instance = USART1;
                break;
            case 1:
             huart1.Instance = USART2;
                break;
            case 2:
             huart1.Instance = USART3;
                break;
            default:
                return FALSE;
        }
        huart1.Init.BaudRate = ulBaudRate;
        switch (ucDataBits) {
            case 8:
             huart1.Init.WordLength = UART_WORDLENGTH_8B;
                break;
            default:
                return FALSE;
        }
        switch (eParity) {
            case MB_PAR_NONE:
             huart1.Init.Parity = UART_PARITY_NONE;
                break;
            case MB_PAR_EVEN:
             huart1.Init.Parity = UART_PARITY_EVEN;
                break;
            case MB_PAR_ODD:
             huart1.Init.Parity = UART_PARITY_ODD;
                break;
            default:
                return FALSE;
        }
        huart1.Init.StopBits = UART_STOPBITS_1;
        huart1.Init.Mode = UART_MODE_TX_RX;
        huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart1.Init.OverSampling = UART_OVERSAMPLING_16;
        return (HAL_OK == USART_Init(&huart1));
   }
```

После настройки интерфейса необходимо проверить его работоспособность. Для этого воспользуемся бесплатно предоставляемой программой terminal. Установим брейкпоинт на функции по приему байта. Если интерфейс работает, то при приеме байта устройство должно среагировать и остановить программу в месте, отмеченном брейкпоинтом.

На рисунке 5.3 показан останов программы при приеме байта 0xBB.

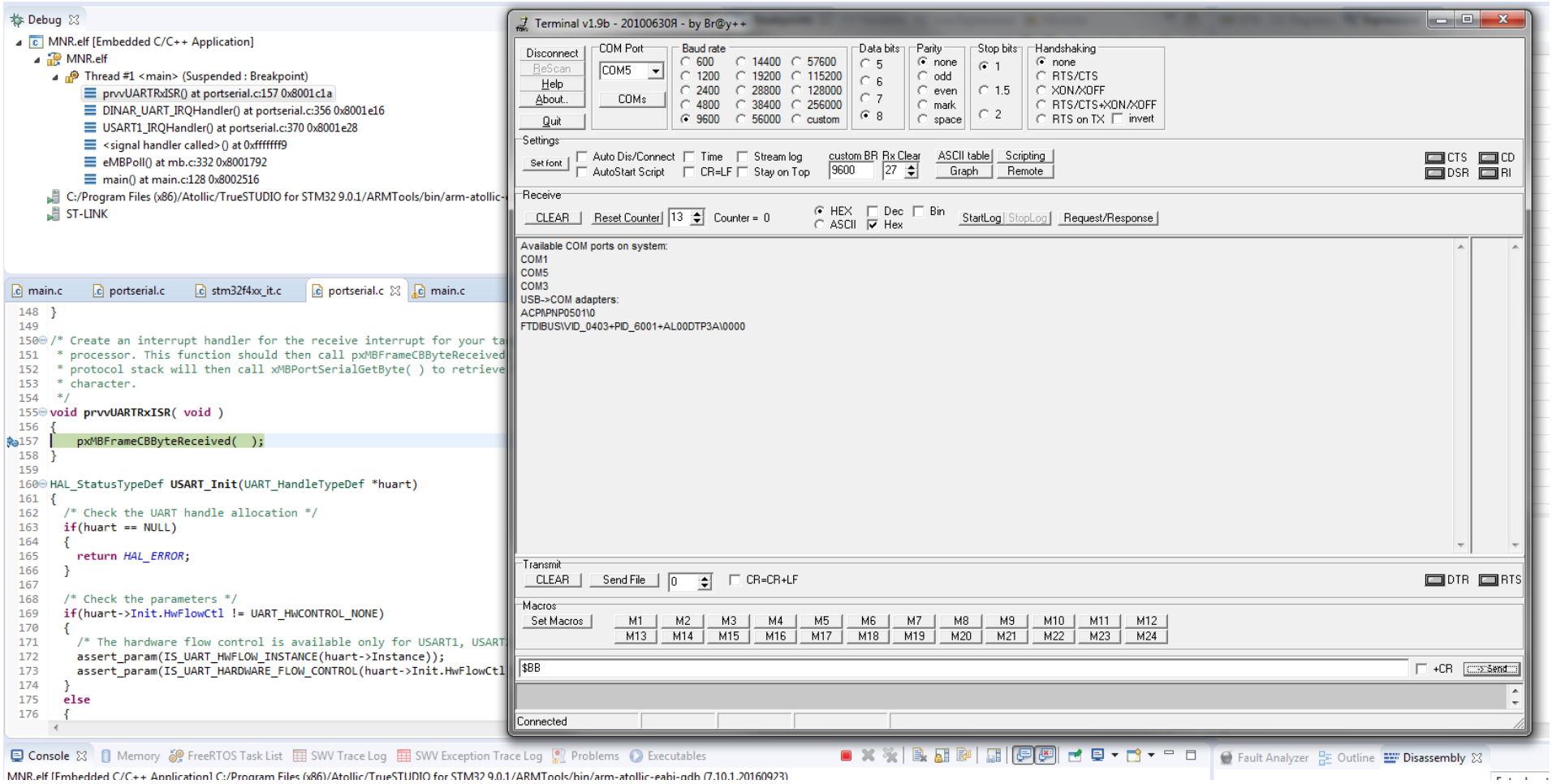
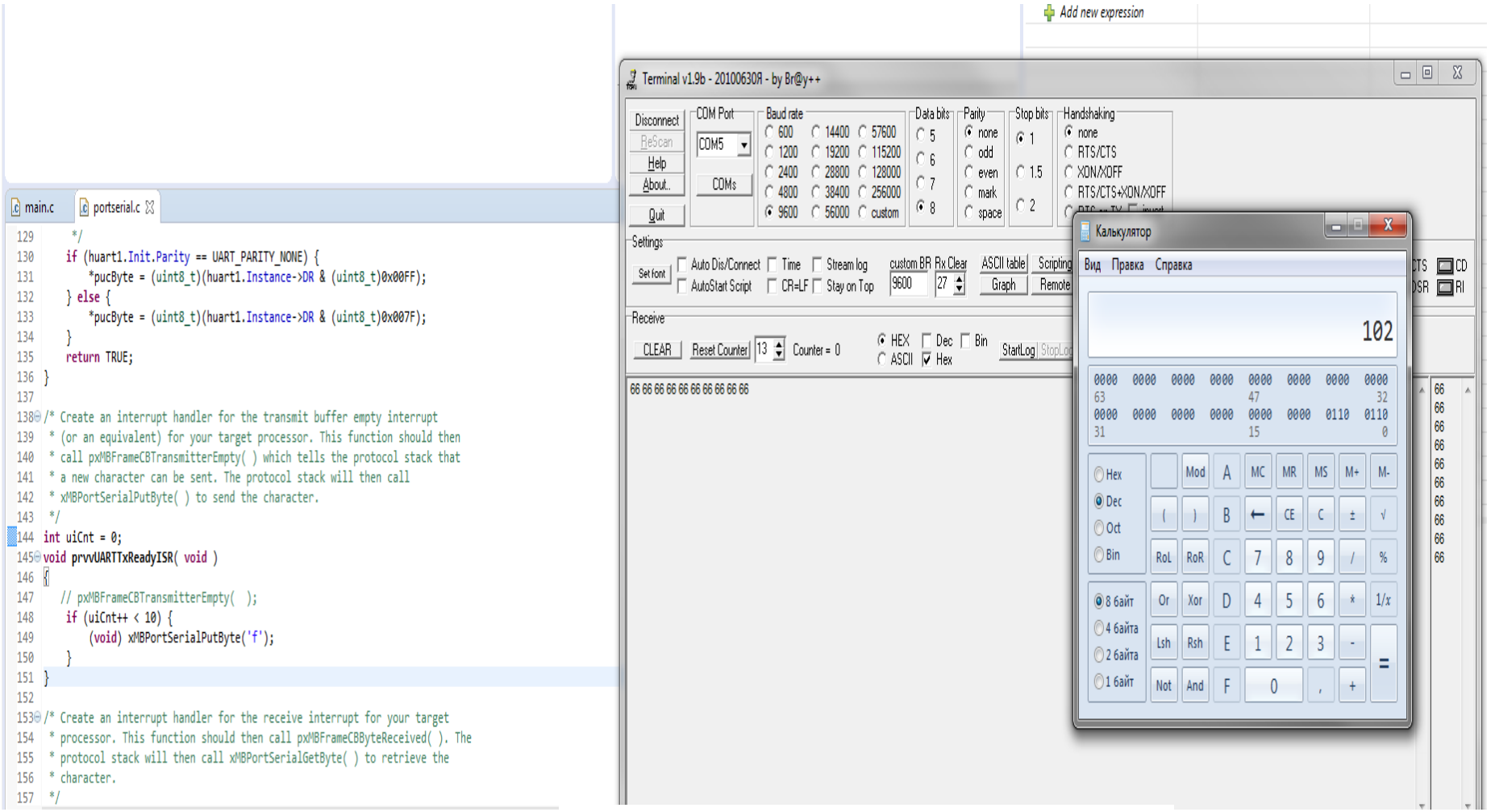


Рисунок 5.3 – Останов программы при приеме байта

Помимо приема система должна обеспечивать передачу данных.

Изменим функцию, отвечающую за передачу данных. Теперь функция передает 10 букв “f”. Код буквы f в таблице ASCII 102. Рассмотрим, что принимает программа terminal. Она принимает 10 значений 66 в шестнадцатеричной системе счисления. С помощью калькулятора переведем шестнадцатеричный формат в десятичный и получим число 102. Соответственно устройство передает и принимает данные. На рисунке 5.4 – показана измененная функция передачи данных, принятые данные и перевод данных в десятичную систему счисления.




```
    */
    if (huart1.Init.Parity == UART_PARITY_NONE) {
        *pucByte = (uint8_t)(huart1.Instance->DR & (uint8_t)0x00FF);
    } else {
        *pucByte = (uint8_t)(huart1.Instance->DR & (uint8_t)0x007F);
    }
    return TRUE;
}

/* Create an interrupt handler for the transmit buffer empty interrupt
 * (or an equivalent) for your target processor. This function should then
 * call pxMBFrameCBTransmitterEmpty( ) which tells the protocol stack that
 * a new character can be sent. The protocol stack will then call
 * xMBPortSerialPutByte( ) to send the character.
 */
int uiCnt = 0;
void prvvUARTTxReadyISR( void )
{
   // pxMBFrameCBTransmitterEmpty(  );
    if (uiCnt++ < 10) {
        (void) xMBPortSerialPutByte('f');
    }
}

/* Create an interrupt handler for the receive interrupt for your target
 * processor. This function should then call pxMBFrameCBByteReceived( ). The
 * protocol stack will then call xMBPortSerialGetByte( ) to retrieve the
 * character.
 */
```

Рисунок 5.4 – Принятые данные

Соберем всю библиотеку полностью. Удалим отладочные изменения. Загрузим в регистры числа 0, 1, 2, 3.

Листинг 8 –Массив данных

```
usRegInputBuf[0] = 0;
usRegInputBuf[1] = 1;
usRegInputBuf[2] = 2;
usRegInputBuf[3] = 3;
```

Установим адрес первого регистра = 10 и количество регистров = 4.

Листинг 9 – Объявление начального адреса и количество регистров

```
#define REG_INPUT_START 10
#define REG_INPUT_NREGS 4
```

Инициализируем интерфейс Modbus RTU.

Листинг 10 – Инициализация modbus

```
    if (eMBInit( MB_RTU, 0x11, 0, 38400, MB_PAR_NONE) !=
MB_ENOERR)
    {
      Error_Handler();
    }

    /* Enable the Modbus Protocol Stack. */
    if (eMBEnable(  ) != MB_ENOERR) {
     Error_Handler();
    }
```

В файле mbconfig.h разрешим работы нужных нам интерфейсов.

Листинг 11 – Разрешение работы

```
#define MB_ASCII_ENABLED                    (  1 )

/*! \brief If Modbus RTU support is enabled. */
#define MB_RTU_ENABLED                      (  1 )
В цикле while (1) { } вызываем функцию
( void )eMBPoll(  );
```

Проверим работу Modbus RTU с помощью утилиты Modbus poll. На рисунке 5.5 показана настройка программы Modbus poll. Согласно программе выбран интерфейс Modbus RTU. Адрес ведомого устройства равен 17. Проверка контроля четности отсутствует. На рисунках 5.6 и 5.7 изображена передача данных по интерфейсу Modbus RTU.

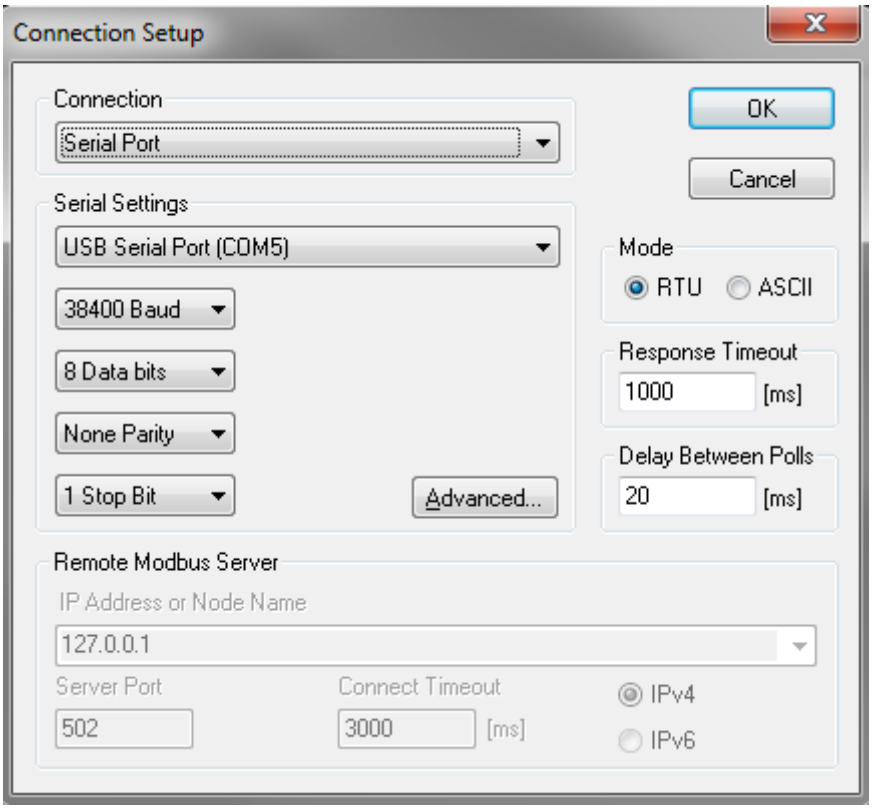


Рисунок 5.5 – Настройка программы Modbus poll

	Alias	00000	Alias	00010
0				1
1				2
2				3
3				
4				
5				
6				
7				
8				
9		0		

Рисунок 5.6 – Полученные данные

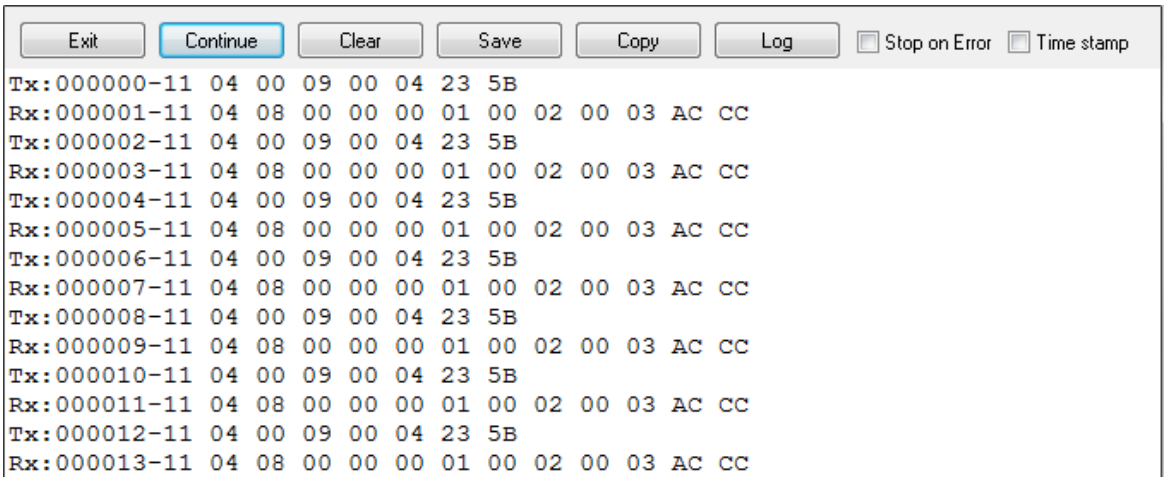


Рисунок 5.7 – Обмен данными по протоколу Modbus RTU

Как видно из рисунков 5.6 и 5.7 данные получены правильно, идет обмен по протоколу Modbus RTU.

Анализируем запрос:

0x11 – Адрес ведомого устройства

0x04 – Код функции на считывание регистров

0x00 – старший байт адреса регистров

0x09 – младший байт адреса регистров

0x00 – старший байт количества считываемых регистров

0x04 – младший байт количества считываемых регистров

0x23 – старший байт контрольной суммы

0x5b – младший байт контрольной суммы

Анализируем ответ:

0x11 – Адрес ведомого устройства

0x04 – Код функции на считывание регистров

0x08 – количество байт данных, которые будут переданы

0x00 – старший байт первого регистра данных

0x00 – младший байте первого регистра данных

0x00 – старший байт второго регистра данных

0x01 – младший байте второго регистра данных

0x00 – старший байт третьего регистра данных

0x02 – младший байте третьего регистра данных

0x00 – старший байт четвертого регистра данных

0x03 – младший байте четвертого регистра данных

0xAC – старший байт контрольной суммы

0xCC – младший байт контрольной суммы

Настроим Modbus ASCII. Для этого изменим функцию инициализации интерфейса на следующую функцию:

Листинг 12 – Инициализация modbus

```
    if (eMBInit( MB_ASCII, 0x11, 0, 38400, MB_PAR_NONE) != MB_ENOERR)
    {
      Error_Handler();
    }
```

Также в программе Modbus poll необходимо переключить интерфейс на Modbus ASCII. Проверим полученные данные и сигнал обмена. На рисунке 5.8 показаны полученные данные и обмен данными по протоколу Modbus ASCII.

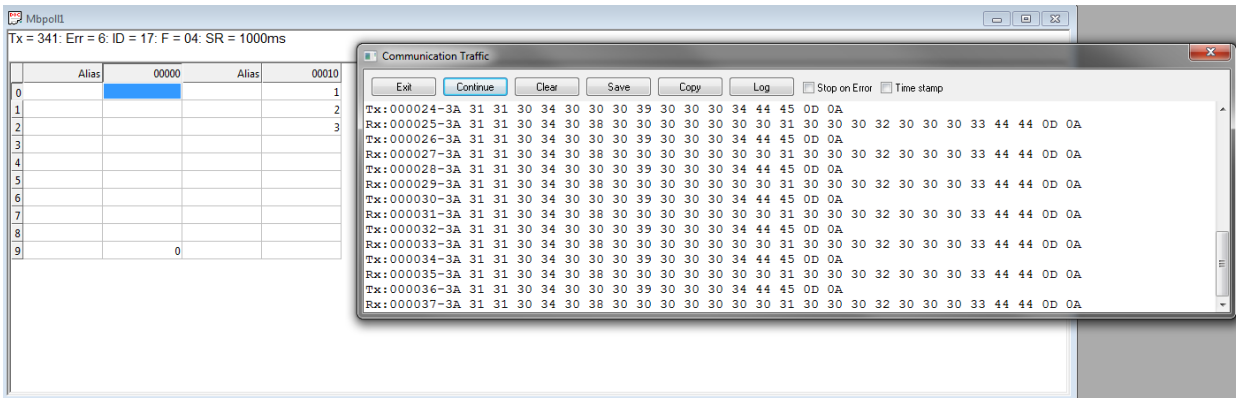


Рисунок 5.8 – Обмен данными по протоколу Modbus ASCII

Видим, что библиотека настроена, работает правильно и готова к интеграции в реальный проект.

Анализируем запрос:

0x3A – стартовый символ передачи “:”

0x31 0x31 – Адрес ведомого устройства. В таблице ASCII символу под номером 0x31 соответствует цифра 1.

0x30 0x34 – Код функции на считывание регистров

0x30 0x30 0x30 0x39 – адрес регистров

0x30 0x30 0x30 0x34 – количество считываемых регистров

0x44 0x45 – LRC

0x0D 0x0A – CRC

Анализируем ответ:

0x3A – стартовый символ передачи ":"

0x31 0x31 – Адрес ведомого устройства.

0x30 0x34 – Код функции на считывание регистров

0x30 0x38 – количество байт данных, которые будут переданы

0x30 0x30 0x30 0x30 – первый байт информации

0x30 0x30 0x30 0x31 – второй байт данных

0x30 0x30 0x30 0x32 – третий байт данных

0x30 0x30 0x30 0x33 – четвертый байт данных

0x44 0x44 – LRC

0x0D 0x0A – CRC

5.2 Разработка библиотеки для LCD дисплея

В LCD WG12864 нет встроенных шрифтов. Поэтому необходимо создать алфавит. Алфавит представляет собой двумерный массив. Пример части массива приведен в приложении Б.

Каждая буква состоит из шести вертикальных фрагментов, по 8 пикселей на фрагмент. Все вместе они образуют букву.

Функции библиотеки:

Листинг 13 – Функции LCD библиотеки

```
   void set_addr(uint8_t line, uint8_t column);
   void write_line(uint8_t line, uint8_t column, const char* text);
   void write_invers_line (uint8_t line, uint8_t column,const char*
text);
   void write_underscore_line(uint8_t line, uint8_t column,const
char* text);
   void write_pix(uint8_t str, uint8_t column, uint32_t pix);
   void fill_LCD(void);
   void LCD_WR_DATE(int date) ;
   void LCD_WR_COM(int command);
   void initLCD(void);
   void invert_letter(int line,int num_bukva);
   int LCD_RD_DATA(int str, int column) ;
   void Init_Gpio_Lcd (void);
```

Программный код основных функций библиотеки приведен в приложении Б. Код функций, которых нет в приложении Б основан на тех же принципах, что и приведенный код.

1) void set_addr(uint8_t line, uint8_t column);

Функция отвечает за установку адреса для дисплея. Для записи каких – либо данных на дисплей необходимо установить адрес. С какой строки и с какого столбца начинаем отрисовку текста или рисунка на дисплее.

2) void write_line(uint8_t line, uint8_t column, const char* text) ;

Функция отвечает за вывод текста на дисплей. Текст выводится на заданную линию и на заданный пиксель. Пример вызова данной функции:

write_line(0, 0, “Быкова И.Д”);

Результат выполнения данной функции представлен на рисунке 5.9.

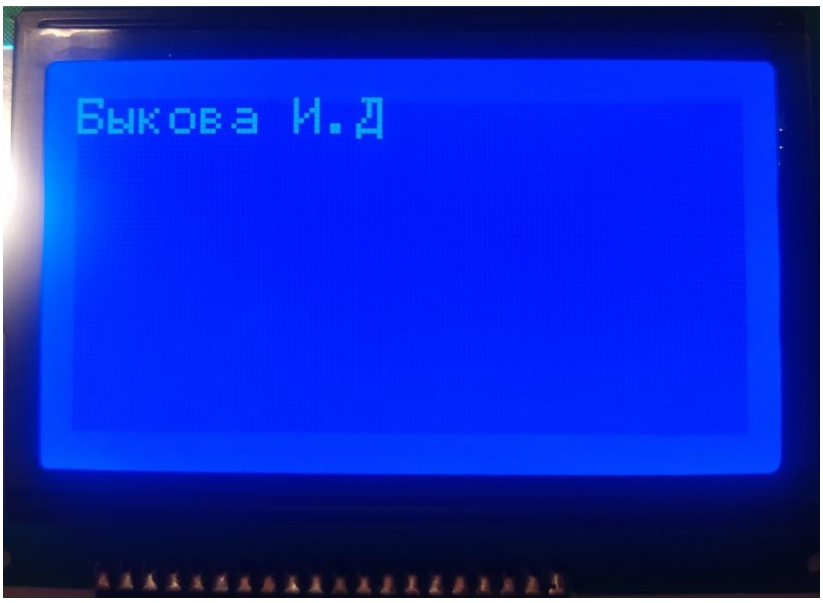


Рисунок 5.9 – Результат выполнения функции write_line

Как видно из рисунка текст начался с нулевого пикселя с нулевой линии. Всего на дисплее 8 строк по 8 пикселей. Текст получился не подчеркнутый, не выделенный.

3) void write_invers_line (uint8_t line, uint8_t column,const char* text);

Функция отвечает за вывод текста на дисплей. Текст выводится на заданную линию и на заданный пиксель. Текст инвертированный. Пример вызова данной функции:

write_invers_line (4, 0, “Быкова И.Д”);

Результат выполнения данной функции представлен на рисунке 5.10.

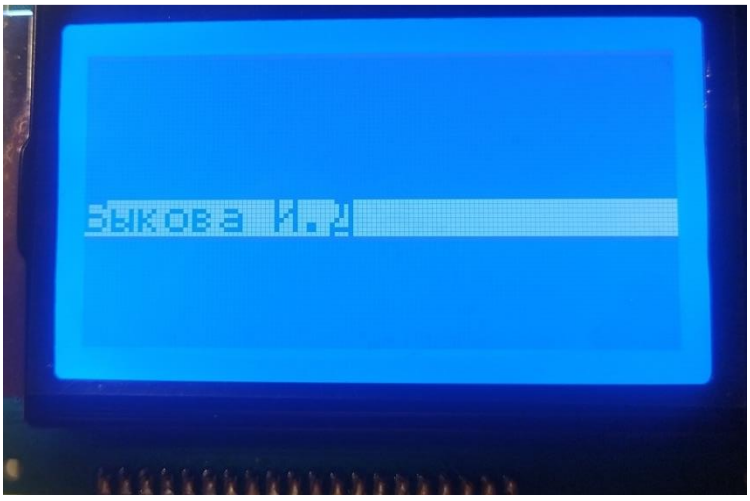


Рисунок 5.10 – Результат выполнения функции write_invers_line

Видим, что текст инвертированный. Начало текста на четвертой линии. Текст начинается с нулевого пикселя.

4) void write_underscore_line(uint8_t line, uint8_t column,const char* text);

Функция отвечает за вывод текста на дисплей. Текст выводится на заданную линию и на заданный пиксель. Текст подчеркнутый. Пример вызова данной функции:

write_underscore_line (0, 0, “Быкова И.Д”);

Результат выполнения данной функции представлен на рисунке 5.11.

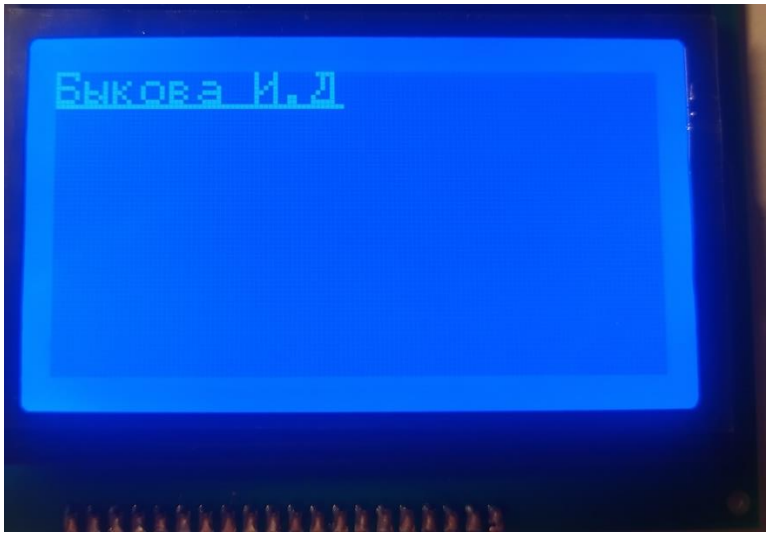


Рисунок 5.11 – Результат выполнения функции write_underscore_line

Результат – подчеркнутый текст, начинающийся с нулевой линии нулевого пикселя.

5) void write_pix(uint8_t str, uint8_t column, uint32_t pix);

Функция отвечает за вывод на дисплей 8ми пикселей там, где это необходимо. Пример вызова данной функции:

write_pix(3, 25, 0xAA);

Пример выполнения данной функции показан на рисунке 5.12

Рисунок 5.12 – Пример выполнения функции write_pix

Видим, что набор пикселей вывелся на третью линию, двадцать пятый столбец и равен 10101010. С помощью данной функции можно создавать на экране рисунки.

6) void fill_LCD(void);

Функция заливки экрана. Используется при обновлении экрана и при его инициализации.

7) void LCD_WR_DATE(int date) ;

Функция записи данных на экран.

Данные на LCD дисплей передаются по алгоритму, представленному на рисунке 5.13.

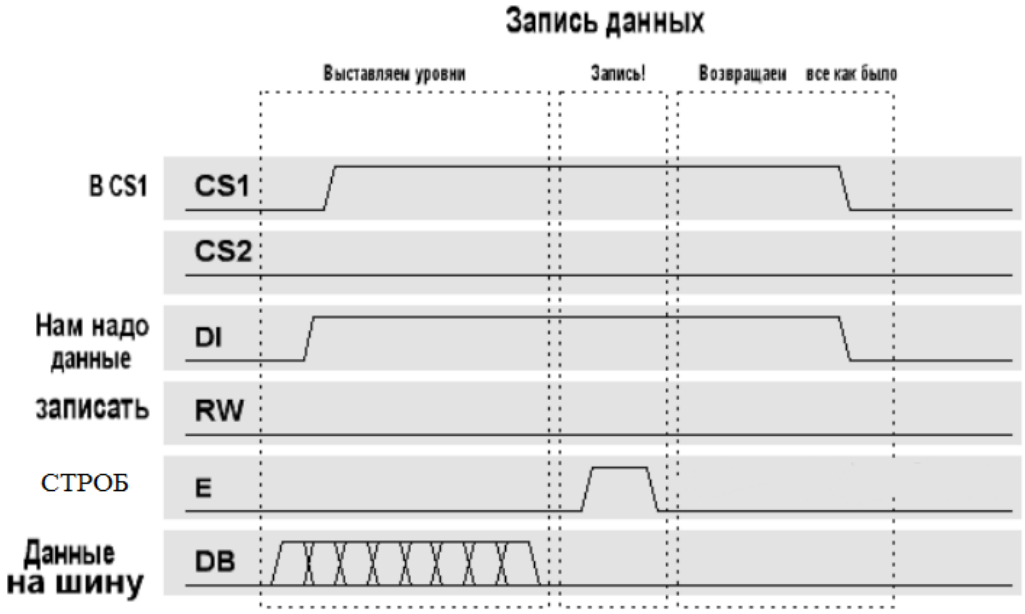


Рисунок 5.13 – Алгоритм записи данных на LCD дисплей

Как видно из программы сначала мы выставляем контакт DI, затем записываем данные и выдаем одиночный импульс.

8) void LCD_WR_COM(int command);

Функция записи на дисплей какой– либо команды. Список команд представлен в приложении А.

Данная функция работает по алгоритму, представленному на рисунке 5.14.

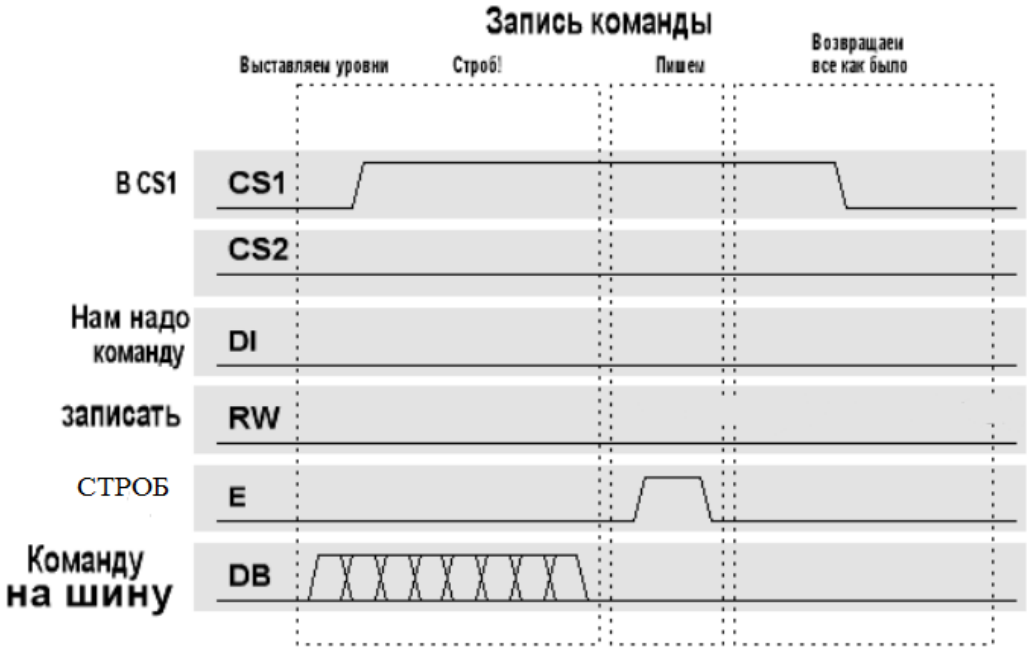


Рисунок 5.14 – Алгоритм для записи команды на LCD дисплей

9) void invert_letter(int line,int num_bukva);

Функция инверсии какой–либо буквы в слове. Результат выполнения данной функции invert_letter(0, 1); представлен на рисунке 5.15

Рисунок 5.15 – Результат выполнения функции invert_letter

Как видно из рисунка инвертировалась нужная буква в нужной строке.

10) int LCD_RD_DATA(int str, int column);

Данная функция считывает данные с указанной строки и колонки. Используется при инверсии выбранной буквы. Сначала мы считываем данные с дисплея, потом инвертируем их и записываем в дисплей.

11) void Init_Gpio_Lcd (void);

Функция инициализации портов ввода вывода для дисплея.

Код функции приведен в приложении Б.

Как видно из кода мы задаем тактирование контактов ввода вывода, а потом настраиваем нужные нам контакты в режим вывода с подтяжкой к земле.

5.3 Разработка ПИД регулятора

В выпускной квалификационной работе ПИД регулятор управляет ШИМ сигналом, выходящим с контакта PD12. Рассмотрим функцию инициализации ШИМ сигнала.

```
void init_PWM(void)
```

Код данной функции приведен в приложении Б.3.

ШИМ сигнал реализован на таймере TIM4. Таймер 4 находится на шине APB1, соответственно частота тактирования таймера будет равна

60МГц. Настраиваем ШИМ сигнал на частоту 3 КГц. Для этого 60000000 / Период = 3000Гц. Соответственно период должен быть равным 20000. Т.к. период равняется (заданное значение + 1), то

```
TMR4.Init.Period = 19999;
```

Коэффициент заполнения отвечает за заполнение ШИМ сигнала. Т.е. если коэффициент заполнения равен периоду, то ШИМ равна 100%.

Снимем диаграмму ШИМ без использования ПИД регулятора. Настроим ШИМ на 50%, задав

```
PWM4.Pulse = 9999;// к-т заполнения
```

Диаграмма ШИМ сигнала без использования ПИД регулятора представлена на рисунке 5.16.

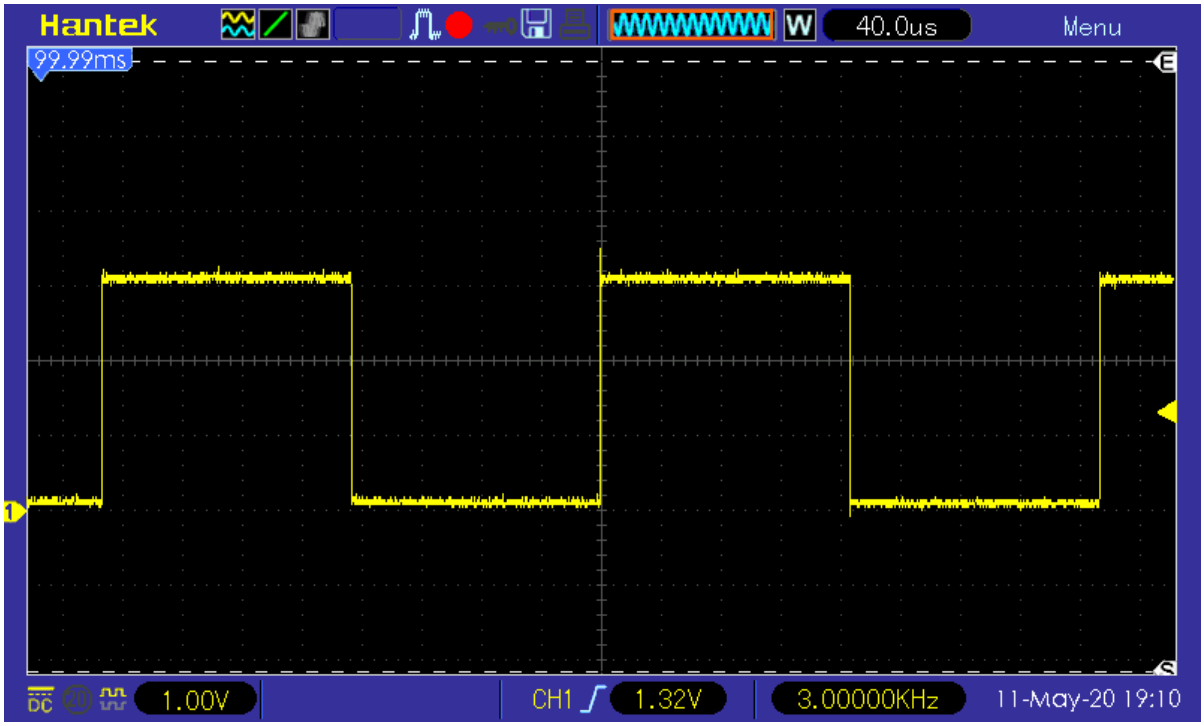


Рисунок 5.16– Диаграмма ШИМ сигнала без использования ПИД регулятора.

Как видно из диаграммы коэффициент заполнения ШИМ равен 50 %. ШИМ сигнал идет с частотой 3 КГц.

ПИД регулятор – это устройство, используемое в системах управления для формирования качественного управляющего сигнала. ПИД регулятор

состоит из трех составляющих. Пропорционального звена, интегрального и дифференцирующего. На рисунке 5.17 показана структурная схема ПИД регулятора.

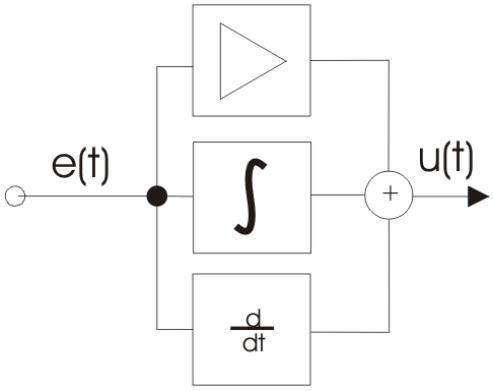


Рисунок 5.17 – Структурная схема ПИД регулятора

5.3.1 Пропорциональная составляющая

Пропорциональная составляющая вырабатывает выходной сигнал, противодействующий отклонению регулируемой величины от заданного значения, наблюдаемому в данный момент времени. Он тем больше, чем больше это отклонение. Если входной сигнал равен заданному значению, то выходной равен нулю.

Однако при использовании только пропорционального регулятора значение регулируемой величины никогда не стабилизируется на заданном значении. Существует так называемая статическая ошибка, которая равна такому отклонению регулируемой величины, которое обеспечивает выходной сигнал, стабилизирующий выходную величину именно на этом значении.

Например, в регуляторе температуры выходной сигнал (мощность нагревателя) постепенно уменьшается при приближении температуры к заданной, и система стабилизируется при мощности, равной тепловым потерям.

5.3.2 Интегральная составляющая

Рассмотрим структурную схему интегрирующего звена, изображенную на рисунке 5.18.

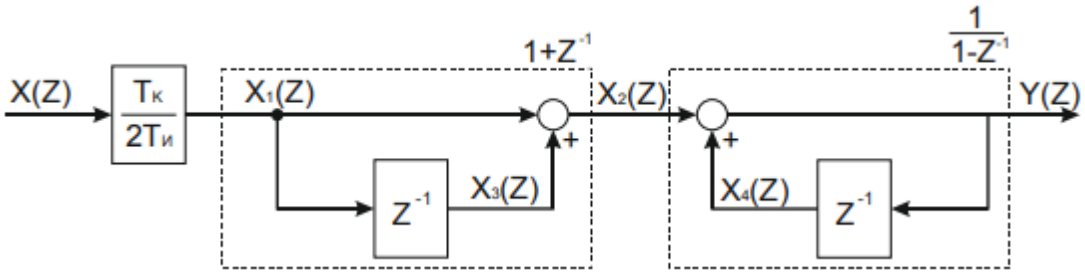


Рисунок 5.18 – структурная схема интегрирующего звена.

Исходя из структурной схемы составим уравнения для поиска интегрирующей составляющей:

Листинг 14 – Функции вычисления интегральной составляющей

```
float Iadd = pid->KI * (pid->EPrev + e) / 2;
float I = pid->IPrev + Iadd;
```

Интегрирующая составляющая пропорциональна интегралу по времени от отклонения регулируемой величины. Её используют для устранения статической ошибки. Она позволяет регулятору со временем учесть статическую ошибку. Интегральная оставляющая имеет свойство накапливания с течением времени.

5.3.3 Дифференциальная составляющая

Рассмотрим структурную схему дифференциальной составляющей, изображенную на рисунке 5.19.

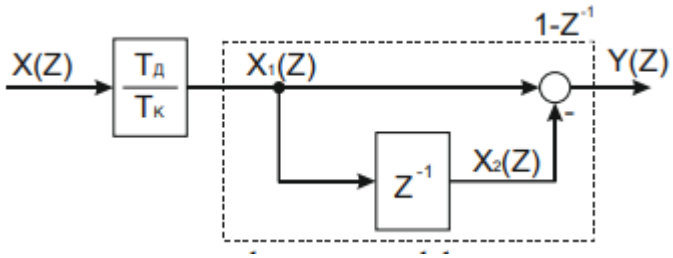


Рисунок 5.19 – Структурная схема дифференциальной составляющей

Составим уравнение для нахождения дифференциальной составляющей:

Листинг 15 – Функция вычисления дифференциальной составляющей

```
float D = pid->KD * (e – pid->EPrev);
```

Дифференцирующая составляющая пропорциональна темпу изменения отклонения регулируемой величины и предназначена для противодействия отклонениям от целевого значения, которые прогнозируются в будущем. Отклонения могут быть вызваны внешними возмущениями или запаздыванием воздействия регулятора на систему.

5.3.4 Работа ПИД регулятора

Регулирование ПИД регулятором проводим с заданным шагом во времени. Это делается для точности регулирования.

Рассмотрим работу ПИД регулятора. Сначала оставим только пропорциональную составляющую. Сделаем коэффициент пропорциональности = 0. Осциллограмма при коэффициенте пропорциональности = 0 изображена на рисунке 5.20.

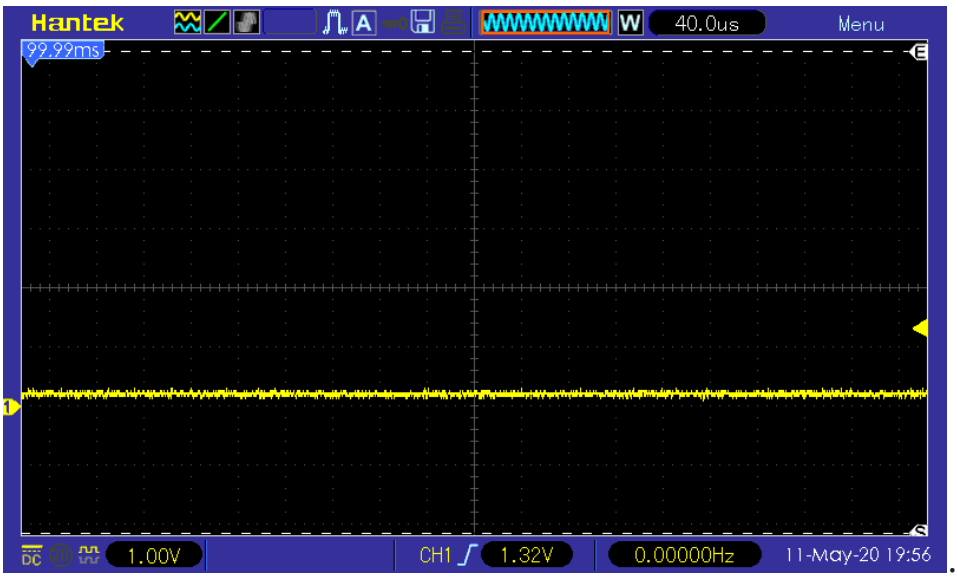


Рисунок 5.20 – Осциллограмма при коэффициенте пропорциональности равном 0

Как видно из рисунка при отсутствии всех трех коэффициентов выходного сигнала нет. Регулятор не работает. Добавим пропорциональную составляющую. Сделаем коэффициент пропорциональности = 10. Диаграмма при коэффициенте пропорциональности равном 10 представлена на рисунке 5.21.

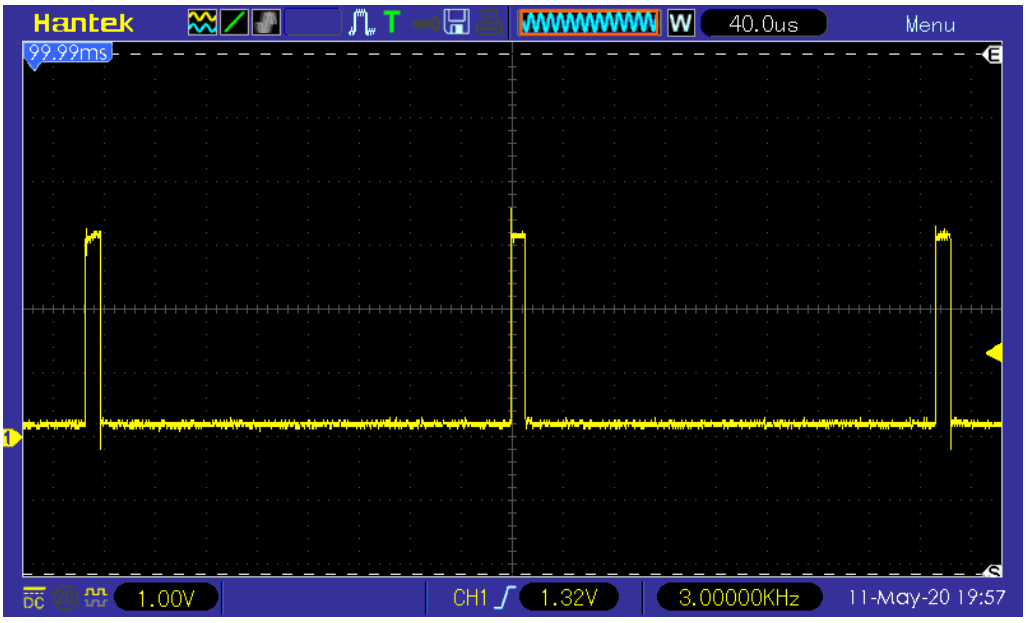


Рисунок 5.21 – Диаграмма при коэффициенте пропорциональности = 10

Добавим коэффициент пропорциональности. Сделаем его равным 100. Диаграмма при коэффициенте пропорциональности равном 100 представлена на рисунке 5.22.

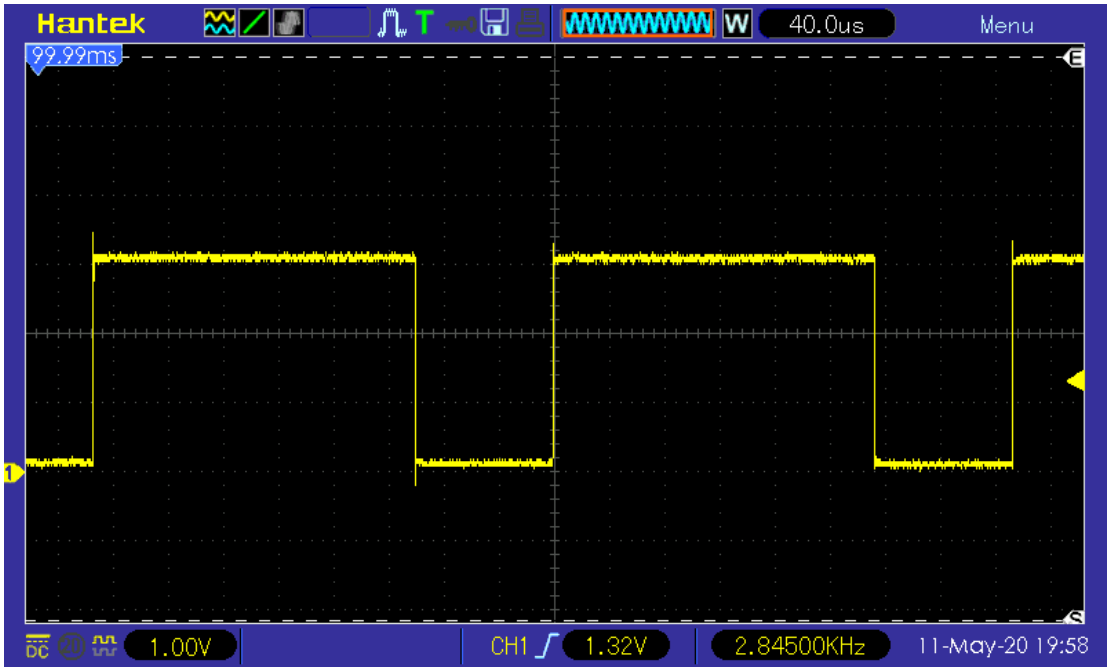


Рисунок 5.22 – Диаграмма при коэффициенте пропорциональности равном 100

Как видно из диаграмм пропорциональное звено отвечает за противодействие отклонению регулируемой величины. Чем больше коэффициент пропорциональности, тем больше противодействие.

Добавим интегральную составляющую.

Должно произойти накапливание интегральной составляющей. Рассмотрим выходной сигнал при отключенной пропорциональной составляющей. Выходной сигнал показан на рисунке 5.23.

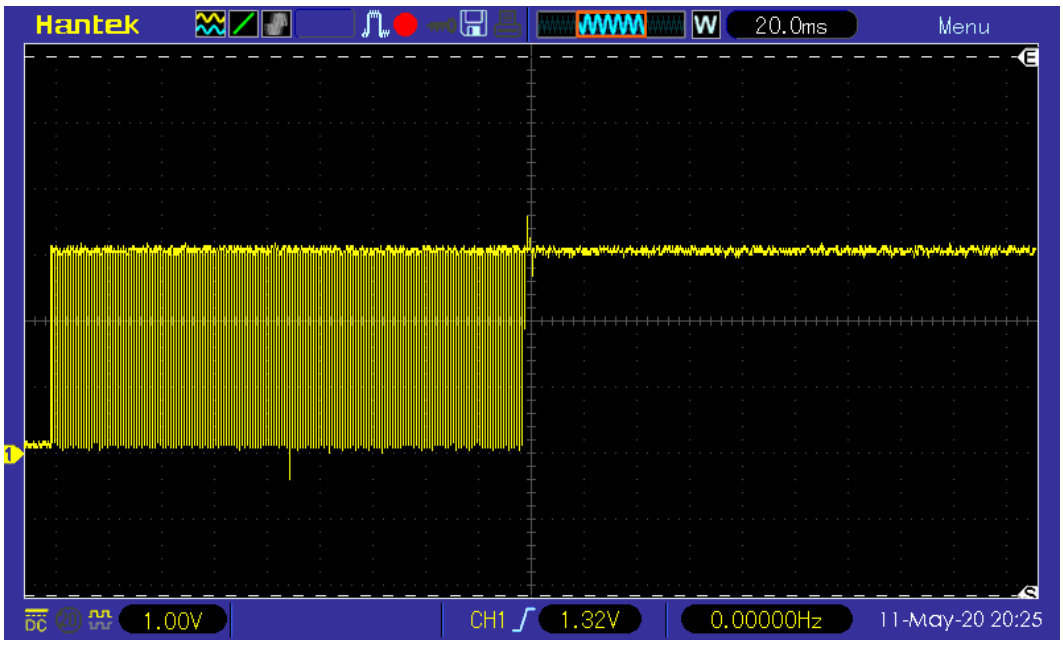


Рисунок 5.23 – Выходной сигнал И регулятора

Как видно из рисунка в начальный момент времени выходной сигнал отсутствовал. Потом начало происходить накопление интегральной составляющей. В конце регулятор оказывает максимальное воздействие из–за учета интегральной составляющей прошлых ошибок.

Рассмотрим ПИ регулятор. Выходной сигнал ПИ регулятора показан на рисунке 5.24.

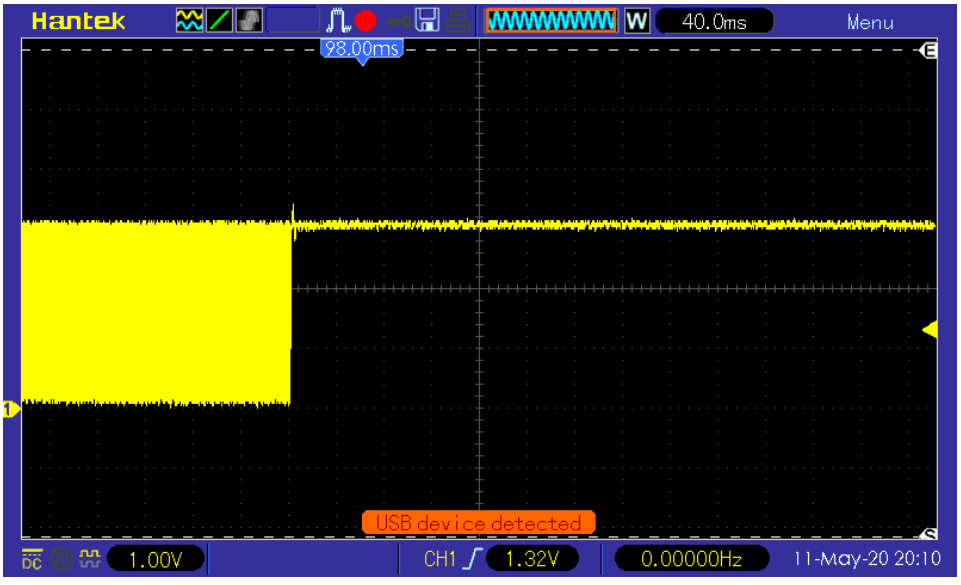


Рисунок 5.24 – Выходной сигнал ПИ регулятора.

Видим, что на начальный момент времени было воздействие пропорциональной составляющей. Потом из–за накопления интегральной составляющей ШИМ сигнал достиг максимального значения.

Рассмотрим дифференциальную составляющую. Дифференциальная составляющая реагирует на темп изменения регулируемой величины. Влияние дифференциальной Составляющей можно посмотреть, поставив точку останова на первом измерении. Так как первом измерении присутствует разница между измеряемой величиной и предыдущим измерением т.е. присутствует темп изменения регулируемой величины, то дифференциальная составляющая срабатывает. На рисунке 5.25 видно, что регулятор реагирует на темп роста измеряемой величины и вносит свою составляющую в формирование выходного ШИМ сигнала.

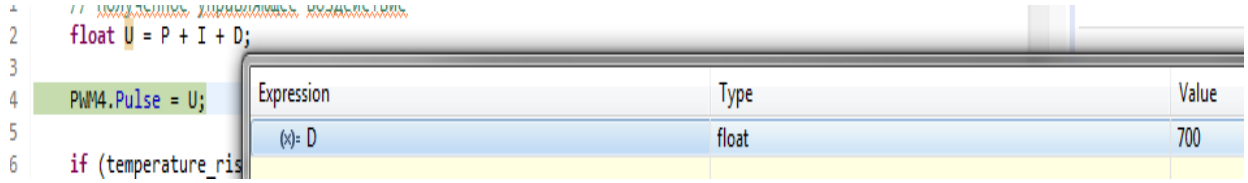


Рисунок 5.25 – Влияние дифференциальной составляющей на ШИМ сигнал

5.4 Исследование системы в работе

Исследуемая система имеет 3 меню.

1) Главное меню. Главное меню показано на рисунке 5.26.

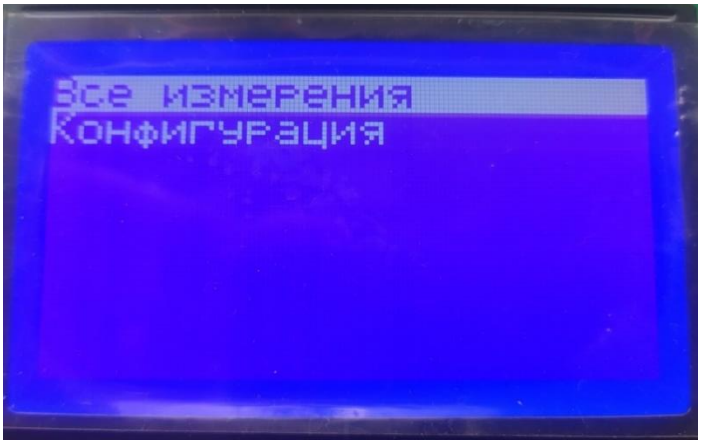


Рисунок 5.26 – Отображение главного меню

Из главного меню, используя кнопки “ВВЕРХ”, “ВНИЗ”, “ОК” осуществляется переход в меню всех измерений и в меню конфигурации.

2) Меню конфигурации.

Меню конфигурации представлено на рисунке 1. Используя кнопки “ВВЕРХ”, “ВНИЗ” осуществляется выбор параметра, который необходимо изменить. Клавишами “ВЛЕВО”, “ВПРАВО” осуществляется изменение параметра. На рисунке 5.27 показано меню конфигурации.

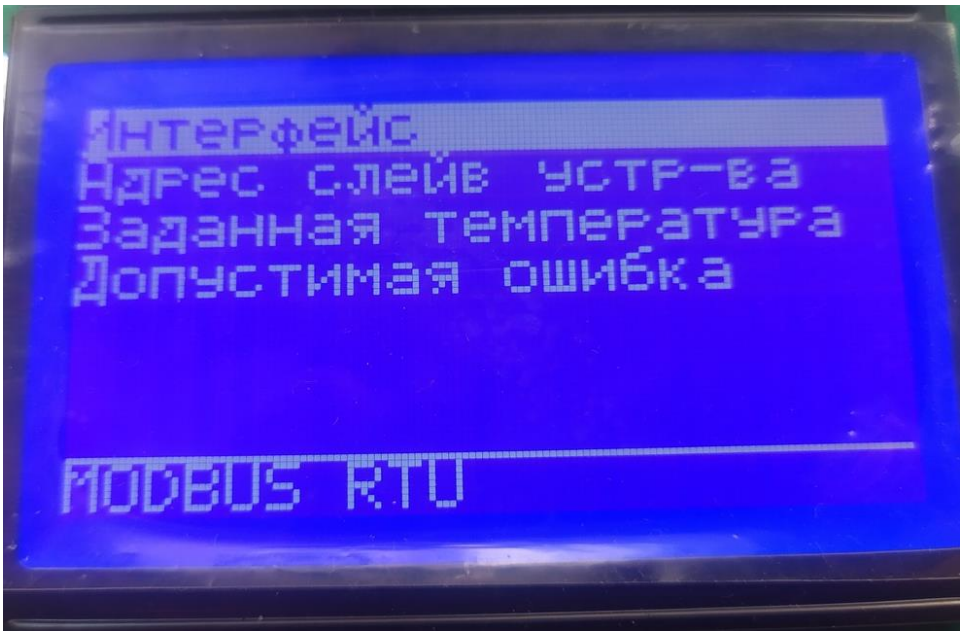


Рисунок 5.27 – Меню конфигурации

При нажатии на кнопку “ВПРАВО” интерфейс Modbus RTU переключается на интерфейс Modbus ASCII. На рисунке 5.28 показан изменившийся параметр, после нажатия на кнопку “ВПРАВО”.

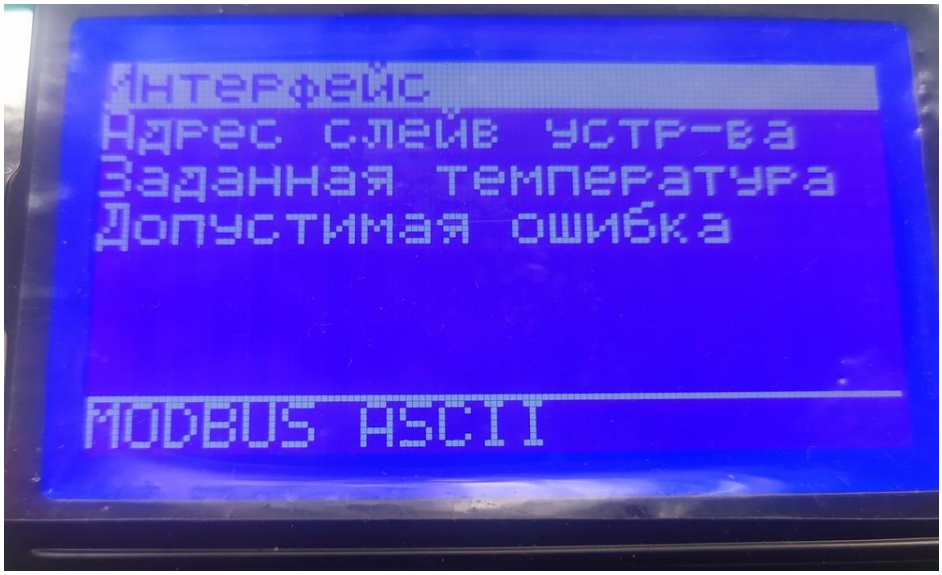


Рисунок 5.28 – Изменение параметра.

3) Меню измерений

Меню измерений показано на рисунке 5.29

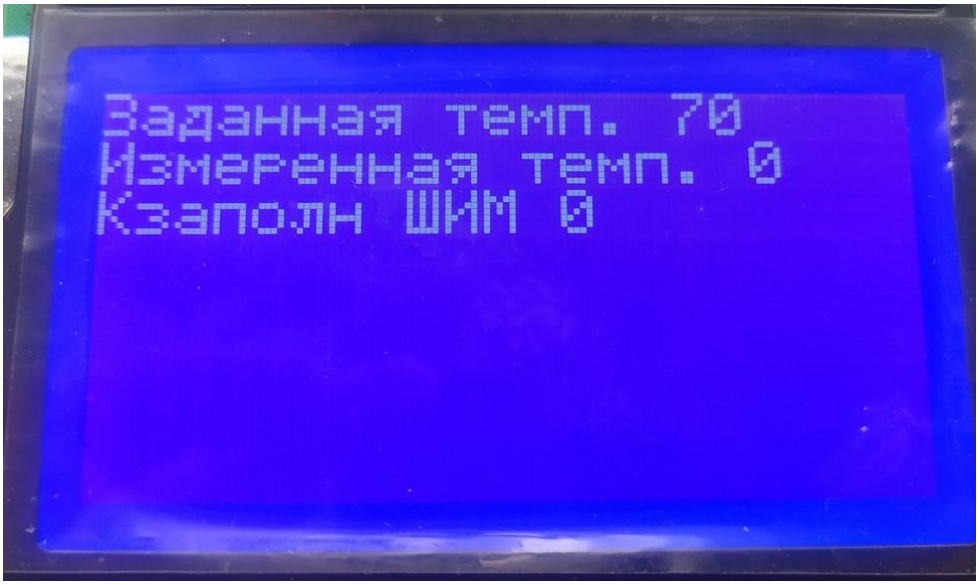


Рисунок 5.29– Меню измерений

Все параметры, отображаемые в меню измерений, передаются по протоколам Modbus ASCII/RTU в зависимости от выбранной конфигурации.

После настройки всех необходимых параметров необходимо нажать на кнопку “ПУСК”. Измеренная температура начнет повышаться, имитируя работу термоконтроллера. ШИМ при этом регулируется ПИД регулятором. Все данные отсылаются по заданному интерфейсу. Система выходит в работу, когда измеренная температура достигает заданной. Пользователь может в любой момент остановить работу системы, нажав на кнопку “СТОП”. При этом температура начнет снижаться, пока не достигнет минимального значения. Минимальное и максимальное значение, до которых доходит температура задаются программистом в коде. Пользователь имеет право выбрать заданную температуру в пределах от минимального до максимального значения. Согласно техническому заданию максимальное значение температуры равно 100 градусов. Минимальное значение равно 0 градусов. Данные пределы могут варьироваться программистом в соответствии с требованием заказчика.

Рассмотрим передачу данных системой в момент времени, когда измеренная температура стала равной 10 градусам.

Передача данных осуществляется по протоколу Modbus RTU. В регистрах лежат следующие значения:

Листинг 16 – Значения в регистрах

```
void GetRegModbus (void) {
   usRegInputBuf[0] = ZadTemp;
   usRegInputBuf[1] = IzmTemp;
   usRegInputBuf[2] = Kzapoln;
   if  (temperature_rise) {
      usRegInputBuf[3] = 1;
   } else {
      usRegInputBuf[3] = 0;
   }
}
```

Соответственно в регистре usRegInputBuf[0] находится заданная температура. Пользователь в меню конфигурации может установить

необходимую заданную температуру. В регистре usRegInputBuf[1] находится измеренная температура в регистре usRegInputBuf[2] находится коэффициент заполнения ШИМ. В регистре usRegInputBuf[3] находится статус системы. Если система запущена в работу, то будет логическая единица. Если система остановлена, то логический ноль.

Рассмотрим параметры системы в ожидании пуска. Параметры представлены на рисунке 5.30.

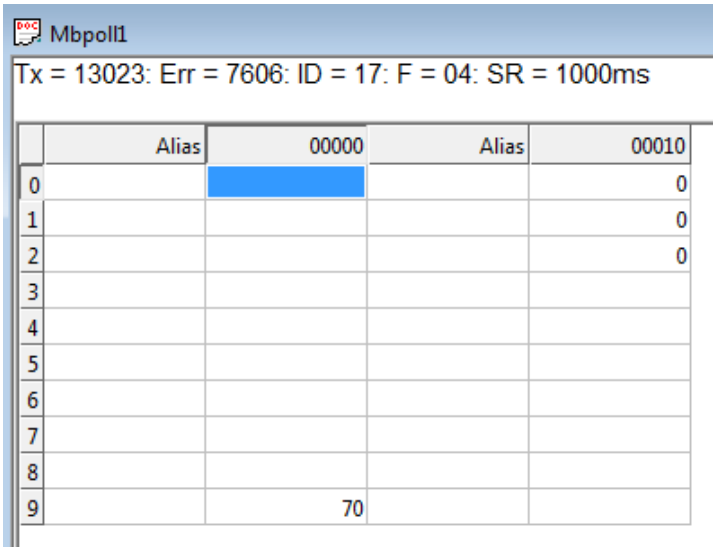


Рисунок 5.30 – Параметры системы перед запуском

Нажмем на кнопку “ПУСК” и дождемся температуры 10 градусов. Параметры, принятые по интерфейсу Modbus RTU представлены на рисунке 5.31.

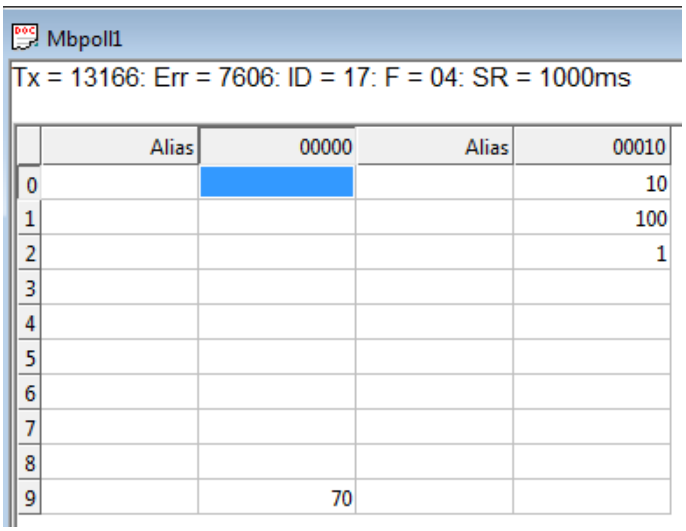


Рисунок 5.31 – Параметры системы в работе

Как видно все данные отображаются корректно.

Соответственно система, имитирующая термоконтроллер работает.

Если пользователь желает изменить настройки. Например регулирование температуры, то необходимо в массиве static ParameterMenuItem* items[] изменить максимальное и минимальное значение необходимого параметра. Массив указателей приведен в листинге в приложении Б.5.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе был разработан программно–аппаратный имитатор режимов работы термоконтроллера с цифровым интерфейсом связи. Была разработана аппаратная часть. Т.е. составлена функциональная схема, на которой изображены основные узлы системы. Данная схема необходима для понимания работы устройства в целом.

На основе функциональной схемы была составлена принципиальная схема. Данная схема необходима для понимания работы отдельных узлов, в частности.

При составлении функциональной схемы был произведен выбор элементов системы и сборка макета системы на базе отладочной платы STM32f4 – discovery.

Была написана библиотека для LCD дисплея. Данная библиотека реализует функции вывода строки на дисплей, функции заливки дисплея, вывода необходимого пикселя и другие.

Был создан ПИД регулятор, для управления нагревателем. С выхода ПИД регулятора были получены сняты осциллограммы: работы П канала, работы И канала, работы ПИ регулятора, работы Д канала.

Была портирована свободно распространяемая библиотека freemodbus под процессор STM32. Также была изучена работа интерфейсов Modbus ASCII и Modbus RTU. В программе Modbuss poll были получены пакеты данных, иллюстрирующие работу данных интерфейсов и обмен данными между устройствами, где в роли мастера выступает персональный компьютер, а в роли ведомого устройства разрабатываемая микропроцессорная система.

После разработки и проверки работоспособности основных узлов было написано основное программное обеспечение, которое соединило в себе все библиотеки и было проверено и протестировано на отладочной плате.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аандрэ, Ф. Микроконтроллеры семейства SX фирмы Ubicom / Ф. Аандрэ. – М.: ДМК, 2016. – 272 с.

2. Аванесян, Г.Р. Интегральные микросхемы ТТЛ, ТТЛШ: справочник / Г.Р. Аванесян, В.В. Левшин. – М.: Машиностроение, 1993. – 256 с.

3. Аванесян, Г.Р. Униполярные интегральные микросхемы. Справочное пособие. (МРБ 1264) / Г.Р. Аванесян, А.А. Беспалов. – М.: ГЛТ, 2003. – 220 с.

4. Аванесян, Г.Р. Униполярные интегральные микросхемы Справочное пособие / Г.Р. Аванесян. – М.: ГЛТ, 2003. – 220 с.

5. Александров К.К. Электротехнические чертежи и схемы/ К.К. Алексанров, Е.Г. Кузьмина. – 3–е изд., стереот. – М.: Издательский дом МЭИ, 2007. – 300 с.

6. Алехин, В.А. Микроконтроллеры PIC: основы программирования и моделирования в интерактивных средах MPLAB IDE, mikroC, TINA, Proteus. Практикум / В.А. Алехин. – М.: ГЛТ, 2016. – 248 с.

7. Белов, А.В. Программирование микроконтроллеров для начинающих и не только. Книга + виртуальный диск / А.В. Белов. – СПб.: Наука и техника, 2016. – 352 с.

8. Белов, А.В. Микроконтроллеры AVR: от азов программирования до создания практических устройств / А.В. Белов. – СПб.: Наука и техника, 2016. – 544 с.

9. Белов, А.В. Разработка устройств на микроконтроллерах AVR: шагаем от "чайника" до профи: Книга / А.В. Белов. – СПб.: Наука и техника, 2013. – 528 с.

10. Бич, М. Микроконтроллеры семейства XC166. Вводный курс разработчика / М. Бич. – М.: ДМК, 2016. – 200 с.

11. Боровский, А.Н. Qt4.7+. Практическое программирование на C++. / А.Н. Боровский. – СПб.: BHV, 2012. – 496 с.

12. Брей, Б. Применение микроконтроллеров PIC 18. Архитектура, программирование и построение интерфейсов с применением C и ассемблера / Б. Брей. – СПб.: КОРОНА–Век, 2014. – 576 с.

13. Вагнер, Б. C#. Эффективное программирование. 50 рекомендаций по усовершенствованию программирования на C# / Б. Вагнер; Пер. с англ. М. Горелик. – М.:Лори, 2013. – 256 с.

14. Васильев, А.Е. Микроконтроллеры. Разработка встраиваемых приложений / А.Е. Васильев. – СПб.: BHV, 2012. – 304 с.

15. Гадре, Д. Занимательные проекты на базе микроконтроллеров tinyAVR / Д. Гадре. – СПб.: BHV, 2012. – 352 с.

16. Гриффитс, Д. Изучаем программирование на C / Д. Гриффитс, Д. Гриффитс. –М.: Эксмо, 2015. –624 с.

17. Гукин, Д. Для "чайников". Программирование на C / Д. Гукин. – М.: Вильямс, 2016. – 384 с.

18. Евстифеев, А.В. Микроконтроллеры AVR семейства Classic фирмы ATMEL / А.В. Евстифеев. – М.:ДМК, 2015. – 286 с.

19. Евстифеев, А.В. Микроконтроллеры AVR семейства Tiny фирмы ATMEL. Руководство пользователя / А.В. Евстифеев. – М.: ДМК, 2015. – 426 с.

20. Евстифеев, А.В. Микроконтроллеры AVR семейств Mega. Руководство пользователя / А.В. Евстифеев. – М.: ДМК, 2015. – 588 с.

21. Заец, Н.И. Радиолюбительские конструкции на PIC–микроконтроллерах. Кн. 1 / Н.И. Заец. – СПб.: КОРОНА–Век, 2015. – 304 с.

22. Иванов, В.Б. Программирование микроконтроллеров для начинающих. Визуальное проектирование, язык C, ассемблер / В.Б. Иванов. – СПб.: КОРОНА–Век, 2015. – 176 с.

23. Каспер, Э. Программирование на языке Ассемблера для микроконтроллеров семейства i8051 / Э. Каспер. –М.: ГЛТ, 2012. –192 с.

24. Кениг, Э. Эффективное программирование на C++. Практическое программирование на примерах. Т. 2 / Э. Кениг, Б.Э. Му. –М.:

Вильямс, 2016. — 368 с.

25. Кравченко, А.В. 10 практических устройств на AVR–микроконтроллерах. Кн. 4 / А.В. Кравченко. – СПб.: КОРОНА–Век, 2013.–320 с.

26. Кравченко, А.В. 10 практических устройств на AVR–микроконтроллерах. Кн.2 / А.В. Кравченко. – СПб.: КОРОНА–Век, 2014. –320 с.

27. Магда, Ю.С. Программирование и отладка C/C++ приложений для микроконтроллеров ARM / Ю.С. Магда. –М.: ДМК, 2014. – 168 с.

28. Магда, Ю.С. Современные микроконтроллеры. Архитектура, программирование, разработка устройств / Ю.С. Магда. –М.: ДМК, 2013. – 228 с.

29. Магда, Ю.С. Микроконтроллеры PIC24. Архитектура и программирование / Ю.С. Магда. – М.: ДМК, 2016. – 240 с.

30. Магда, Ю.С. Программирование и отладка C/C++ приложений микроконтроллеров ARM / Ю.С. Магда. –М.: ДМК Пресс, 2012. – 168 с.

31. МакГрат, М. Программирование на C для начинающих / М. МакГрат. – М.: Эксмо, 2015. –192 с.

32. Мальцев, П.П. Цифровые интегральные микросхемы. Справочник. / П.П. Мальцев и др. – М.: Радио и связь, 1994. – 240 с.

33. Мальцев, П.П. Цифровые интегральные микросхемы: справочник / П.П. Мальцев, Н.С. Долидзе, М.И. Критенко. – М.: Радио и связь, 1994. – 240с.

34. Мартынов, Н.Н. Программирование для Windows на C / Н.Н. Мартынов. – М.:БИНОМ, 2013. – 528 с.

35. Матюшов, Н.В. Начало работы с микроконтроллерами STM8 / Н.В. Матюшов. – М.: Солон–пресс, 2016. – 208 с.

36. Мортон, Д. Микроконтроллеры AVR. Вводный курс / Д. Мортон. – М.: ДМК, 2015. – 272 с.

37. Нефедов, А. Интегральные микросхемы и их зарубежные аналоги т.8.Сер.К 1044–1142 / А. Нефедов. – М.: РадиоСофт, 2001. – 512 с.

38. Нефедов, А. Интегральные микросхемы и их зарубежные аналоги т.11.Сер.К 1564–1814 / А. Нефедов. – М.: РадиоСофт, 2001. – 512 с.

39. Нефедов, А. Интегральные микросхемы и их зарубежные аналоги т.9 Сер.К 1144–1500 / А. Нефедов. – М.: РадиоСофт, 2001. – 512 с.

40. Нефедов, А. Интегральные микросхемы и их зарубежные аналоги т.2. Сер.К143174 / А. Нефедов. – М.: РадиоСофт, 2008. – 640 с.

41. Нефедов, А.В. Интегральные микросхемы и их зарубежные аналоги. т.3 Сер. К175–К505: Справочник / А.В. Нефедов. – М.: РадиоСофт, 2008. – 576с.

42. Перри, Г. Программирование на С для начинающих / Г. Перри, Д. Миллер. – М.: Эксмо, 2015. – 368 с.

43. Понамарев, В. Программирование на С++/С# в Visual Studio.NET 2003. / В. Понамарев. – СПб.: BHV, 2004. – 352 с.

44. Прокопенко, В.С. Программирование микроконтроллеров ATMEL на языке С / В.С. Прокопенко. – СПб.: Корона–Век, 2013. – 320 с.

45. Раскин, Д. Интегральные микросхемы серии STK фирмы SANYO и STR, SAI, SI фирмы Sanken / Д. Раскин. – М.: Горячая линия –Телеком, 2001. – 144 с.

46. Ревич, Ю.В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера / Ю.В. Ревич. – СПб.: BHV, 2012. – 352 с.

47. Редькин, П. 32–битные микроконтроллеры NXP с ядром Cortex–M3 семейства LPC17xx. Полное руководство / П. Редькин. – М.: ДМК, 2015. – 766с.

48. Рюмик, С.М. 1000 и одна микроконтроллерная схема. Книга 1 / С.М. Рюмик. – М.: Додэка–XXI,2012. – 356 с.

49. Рюмик, С.М. 1000 и одна микроконтроллерная схема / С.М. Рюмик. – М.: Додэка XXI, 2012. – 356 с.

50. Саммерфилд, М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++ / М. Саммерфилд. – М.: Символ– Плюс, 2011. – 560 с

51. Секунов, Н. Программирование на C++ в Linux / Н. Секунов. – СПб.: BHV, 2004. – 368 с.

52. Семенов, Б.Ю. Микроконтроллеры MSP430: первое знакомство / Б.Ю. Семенов. – М.: Солон–пресс, 2012. – 128 с.

53. Смирнов, Ю.А. Основы микроэлектроники и микропроцессорной техники: Учебное пособие / Ю.А. Смирнов. – СПб.: Лань, 2013. – 496 с.

54. Смирнов, Ю.А. Электронные и микропроцессорные системы управления автомобилей: Учебное пособие / Ю.А. Смирнов, А.В. Муханов. – СПб.: Лань, 2012. – 624 с.

55. Смирнов, Ю.А. Электронные и микропроцессорные системы управления автомобилей / Ю.А. Смирнов, А.В. Муханов. – СПб.: Лань, 2012. – 624 с.

56. Смирнов, Ю.А. Электронные и микропроцессорные системы управления автомобилей: Учебное пособие / Ю.А. Смирнов, А.В. Муханов, М.А. Пипенко. – СПб.: Лань, 2012. – 624 с.

57. Смирнов, Ю.А. Основы микроэлектроники и микропроцессорной техники: Учебное пособие / Ю.А. Смирнов, С.В. Соколов, Е.В. Титов. – СПб.: Лань, 2013. – 496 с.

58. Соммер, У. Программирование микроконтроллерных плат Arduino/Freeduino / У. Соммер. – СПб.: BHV, 2016. – 256 с.

59. Соммер, У. Программирование микроконтроллерных плат Arduino/Freeduino. / У. Соммер. – СПб.: BHV, 2013. – 256 с.

60. Фленов, М.Е. Программирование на C++ глазами хакера. / М.Е. Фленов. – СПб.: BHV, 2012. – 352 с.

61. Хартов, В.Я. Микроконтроллеры AVR. Практикум для начинающих: Учебное пособие / В.Я. Хартов. – М.: МГТУ им. Баумана, 2012. – 280 с.

62. Хенкеманс, Д. Программирование на C++ / Д. Хенкеманс, М. Ли. – СПб.: Символ–плюс, 2015. – 416 с.

63. Хофманн, М. Микроконтроллеры для начинающих / М. Хофманн. – СПб.: BHV, 2013. – 304 с.

64.Хусаинов Р.З., Качалов А.В. Микропроцессорные системы управления электроприводов. Учебное пособие к курсовому проектированию. Челябинск, Издательство ЮУрГУ, 2015, – 44с.

65. Чиртик, А.А. Программирование на C++: Трюки и эффекты / А.А. Чиртик. – СПб.: Питер, 2010. – 352 с.

66. Шагурин, И.И. Микропроцессоры и микроконтроллеры фирмы Motorola: справочное пособие / И.И. Шагурин. – М.: Радио и связь, 1998. – 560 с.

67. Шагурин, И.И. Современные микроконтроллеры и микропроцессоры фирмы Motorola: Справочник. / И.И. Шагурин. – М.: Горячая линия –Телеком , 2004. – 952 с.

68. Шагурин, И.И. Современные микроконтроллеры и микропроцессоры / И.И. Шагурин. – М.: Горячая линия – Телеком, 2004. – 952 с.

69. Шлее, М. Qt 5.3. Профессиональное программирование на C++. В подлиннике / М. Шлее. – СПб.: BHV, 2016. – 928 с.

70. Шлее, М. Qt4.8. Профессиональное программирование на C++ / М. Шлее. – СПб.: BHV, 2012. – 912 с.

71. Шонфелдер, Г. Измерительные устройства на базе микропроцессора ATmega / Г. Шонфелдер. – СПб.: BHV, 2012. – 288 с.

72. Шонфелдер, Г. Измерительные устройства на базе микропроцессора A Tmega / Г. Шонфелдер, К. Шнайдер. – СПб.: БХВ–Петербург, 2012. – 288 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Программный код расчета скоростей электродвигателей и ПИД–регулирования

Таблица А.1– Программный код расчета скоростей

Команда	D/I	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Назначение
Отображение ВКЛ/ВЫКЛ	L	L	L	L	H	H	H	H	H	L/H	Управляет вкл/выкл отображения. Не влияет на внутреннее состояние и данные ОЗУ изображения. L: ВЫКЛ H: ВКЛ
Установить Адрес	L	L	L	H	Адрес Y (0 ~ 63)						Заносит адрес Y в счётчик адреса Y
Установить Страницу (адрес X)	L	L	H	L	H	H	H	Страница (0 ~ 7)			Заносит адрес X в регистр адреса X
Начальная Строка Отображения	L	L	H	H	Начальная строка отображения (0 ~ 63)						Скроллинг вверх. На сколько пикселей сдвинуть адресное пространство. При этом уехавшее вверх, за экран, вылезет снизу, словно мы провернули экранную область как барабан

Окончание таблицы А.1

Команда	D/I	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Назначение
Чтение Состояния	L	H	BUSY	L	ON/OFF	RESET	L	L	L	L	Чтение состояния. BUSY L: Готовность H: выполняется команда ON/OFF L: Отображение ВКЛ H: Отображение ВЫКЛ RESET L: Нормальный режим H: Сброс
Запись Данных Изображения	H	L	Данные для записи								Записывает данные (DB0:7) в ОЗУ данных изображения. После записи инструкции, адрес Y увеличивается на 1 автоматически.
Чтение Данных Изображения	H	H	Данные для чтения								Читает данные (DB0:7) из ОЗУ данных изображения на шину данных. После чтения адрес Y сам увеличивается на 1 автоматически

ПРИЛОЖЕНИЕ Б

Листинг разработанного программного обеспечения

Листинг Б.1 – Основные программы управления LCD дисплеем

```
   LCDdrive.c
   void write_line(uint8_t line, uint8_t column, const char* text)
{ // страница x, колонка y, текст, который выводим
      int cnt_letters = 0;
      int stolb = 0;  // колонка, счетчик – номер буквы, столбец,
буква – номер в таблице

      while (text[cnt_letters] != '\0') {
         set_addr(line, column);
         if (stolb == 6) { // между буквами интервал
            cnt_letters++;
            stolb = 0;
         }
         int bukva = text[cnt_letters];
         LCD_WR_DATE(array[bukva][stolb]); // без инверсии
         stolb++;
         column++;

      }
      while (column < 128) { // очищаем оставшуюся область дисплея
         set_addr(line, column);
         LCD_WR_DATE(0x00);
         column++;
      }
   }

   void LCD_WR_DATE(int date) {
      SendDate();
      LCD_DATA_PORT->ODR = date;
      delay_us(5);
      Strob_ON();
      delay_us(1);
      Strob_OFF();
   }

   void LCD_WR_COM(int command) {
      SendCommand();
      LCD_DATA_PORT->ODR = command;
      delay_us(5);
      Strob_ON();
      delay_us(1);
      Strob_OFF();
   }

   LCDdrive.h
   #ifndef LCDDRIVE_H_
   #define LCDDRIVE_H_
```

Продолжение листинга Б.1

```
   #define RS          (GPIO_PIN_15)
   #define RW          (GPIO_PIN_8)
   #define E           (GPIO_PIN_12)
   #define DATA0       (GPIO_PIN_0)
   #define DATA1       (GPIO_PIN_1)
   #define DATA2       (GPIO_PIN_2)
   #define DATA3       (GPIO_PIN_3)
   #define DATA4       (GPIO_PIN_4)
   #define DATA5       (GPIO_PIN_5)
   #define DATA6       (GPIO_PIN_6)
   #define DATA7       (GPIO_PIN_7)
   #define CS1         (GPIO_PIN_11)
   #define CS2         (GPIO_PIN_9)
   #define RST         (GPIO_PIN_10)

   #define LCD_DATA_PORT    (GPIOD)
   #define LCD_CTRL_PORT    (GPIOG)

   #define SendDate()    HAL_GPIO_WritePin(LCD_CTRL_PORT, RS,
GPIO_PIN_SET)
   #define SendCommand() HAL_GPIO_WritePin(LCD_CTRL_PORT, RS,
GPIO_PIN_RESET)
   #define Strob_ON()    HAL_GPIO_WritePin(LCD_CTRL_PORT, E,
GPIO_PIN_SET)
   #define Strob_OFF()   HAL_GPIO_WritePin(LCD_CTRL_PORT, E,
GPIO_PIN_RESET)
   #define CS1_ON()    HAL_GPIO_WritePin(LCD_CTRL_PORT, CS1,
GPIO_PIN_SET)
   #define CS1_OFF()     HAL_GPIO_WritePin(LCD_CTRL_PORT, CS1,
GPIO_PIN_RESET)
   #define CS2_ON()    HAL_GPIO_WritePin(LCD_CTRL_PORT, CS2,
GPIO_PIN_SET)
   #define CS2_OFF()     HAL_GPIO_WritePin(LCD_CTRL_PORT, CS2,
GPIO_PIN_RESET)
   #define RW_ON()       HAL_GPIO_WritePin(LCD_CTRL_PORT, RW,
GPIO_PIN_SET)
   #define RW_OFF()    HAL_GPIO_WritePin(LCD_CTRL_PORT, RW,
GPIO_PIN_RESET)
   #define RST_ON()    HAL_GPIO_WritePin(LCD_CTRL_PORT, RST,
GPIO_PIN_SET)
   #define RST_OFF()     HAL_GPIO_WritePin(LCD_CTRL_PORT, RST,
GPIO_PIN_RESET)

   #define MAX_COLUMN 64
   #define MAX_STR 8
   #define FIRST_LINE 0x00B8
   #define LCD_ON  0x003F
   #define LCD_OFF 0x003E
   #define FIRST_LINE_SHIFT  0x00C0
```

Окончание листинга Б.1

```
#define FIRST_COLUMN  0x0040
void set_addr(uint8_t line, uint8_t column);
void write_line(uint8_t line, uint8_t column, const char* text)
;
void write_invers_line (uint8_t line, uint8_t column,const
char* text);
void write_underscore_line(uint8_t line, uint8_t column,const
char* text);
void write_pix(uint8_t str, uint8_t column, uint32_t pix);
void fill_LCD(void);
void LCD_WR_DATE(int date) ;
void LCD_WR_COM(int command);
void initLCD(void);
void invert_letter(int line,int num_bukva);
int LCD_RD_DATA(int str, int column) ;
void write_buf_izm (void* buf);
void write_buf_config (void* buf);
void Init_Gpio_Lcd (void);

#endif /* UTIL_H_ */
```

Листинг Б.2 – Листинг массива для отрисовки текста на LCD дисплее

```
Font.c
char array [356][6] = {

      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
```

Окончание листинга Б.2

```
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0x00, 0x00, 0x00, 0x00, 0x00, 0x00},// (space)
      {0x00, 0x00, 0x5F, 0x00, 0x00, 0x00},// !
      {0x00, 0x07, 0x00, 0x07, 0x00, 0x00},// "
      {0x14, 0x7F, 0x14, 0x7F, 0x14, 0x00},// #
      {0x24, 0x2A, 0x7F, 0x2A, 0x12, 0x00},// $
      {0x23, 0x13, 0x08, 0x64, 0x62, 0x00},// %
      {0x36, 0x49, 0x55, 0x22, 0x50, 0x00},// &
      {0x00, 0x05, 0x03, 0x00, 0x00, 0x00},// '
      {0x00, 0x1C, 0x22, 0x41, 0x00, 0x00},// (
Далее массив продолжается.
   Font.h
#ifndef FONT_H_
#define FONT_H_

extern char array [256][6];

#endif /* FONT_H_ */
```

Листинг Б.3 – Листинг ПИД регулятора

```
PIDregulator.c
   #include <PIDregulator.h>
#include <math.h>
#include <STM32f4xx_hal.h>
#include "main.h"
#include "menu.h"
#include "buttons.h"
#include "settingsmenu.h"
#include "measurementmenu.h"

void PID_Init_Reg(PIDRegulatorData* pid) {

    pid->KP =100;
    pid->KI = 10;
    pid->KD = 10;
    pid->Sigma = 0;

    pid->MinPWM = 0;
    pid->MaxPWM = 20000;
    pid->Setpoint = 70;
    pid->Measurement = 0;
    pid->lastTime = 0;
    pid->IPrev = 0;
```

Продолжение листинга Б.3

```
      pid->EPrev = 0;
       pid->P = 0;
      pid->I = 0;
      pid->D = 0;
   }

   void PID_RegulationStep(PIDRegulatorData* pid) {
      ParameterMenuItem* item2 = (ParameterMenuItem*
)Menu_Settings.items[3];
       pid->Setpoint = ZadTemp;
       pid->Measurement = IzmTemp;
       pid->Sigma = item2->currentValue;

       // Вычислить отклонение
       float e = pid->Setpoint - pid->Measurement;

       // Если отклонение меньше заданного - выход
       if (fabsf(e) < pid->Sigma) {
           pid->EPrev = 0;
           return;
       }

   // Иначе - регулирование

       // Вычислить P
       float P = pid->KP * e;

       // Вычислить добавку I и саму I (метод трапеций)
       float Iadd = pid->KI * (pid->EPrev + e) / 2;
       float I = pid->IPrev + Iadd;

       // Вычислить D
       float D = pid->KD * (e - pid->EPrev);

       // Полученное управляющее воздействие
       float U = P + I + D;

       PWM4.Pulse = U;

       if (temperature_rise) {
         HAL_TIM_PWM_ConfigChannel(&TMR4, &PWM4, TIM_CHANNEL_1);
// настроили шим и канал шим
         HAL_TIM_PWM_Start(&TMR4, TIM_CHANNEL_1);
       } else {
         PWM4.Pulse = 0;
         HAL_TIM_PWM_ConfigChannel(&TMR4, &PWM4, TIM_CHANNEL_1);
// настроили шим и канал шим
         HAL_TIM_PWM_Stop(&TMR4, TIM_CHANNEL_1);
       }
```

Продолжение листинга Б.3

```
        // Ограничение управляющего воздействия
        // Если регулирование в сторону границы, инт. составляющая
не должна накапливаться.
        if (U < pid->MinPWM) {
            U = pid->MinPWM;
            if (e < 0)
                Iadd = 0;
        }
        if (U > pid->MaxPWM) {
            U = pid->MaxPWM;
            if (e > 0)
                Iadd = 0;
        }

        // Сохранение состояния
        pid->EPrev = e;
        pid->IPrev = pid->IPrev + Iadd;

        // Для отображения
        pid->P = P;
        pid->I = I;

    pid->D = D;
    }
    #ifndef PIDREGULATOR_H_
    #define PIDREGULATOR_H_

    #include <stdint.h>
    #define PID_PERIOD_MS 1000

    PIDregulator.h
    typedef struct {

        // Управляющие параметры
        float KP;
        float KI;
        float KD;
        float Sigma;     /// Допустимая ошибка
        float MinPWM;
        float MaxPWM;
        float Setpoint;          /// Уставка
        float Measurement;          /// Измерение

        // Предыдущее состояние
        float IPrev;     /// Накопленная интегральная составляющая
        float EPrev;     /// Отклонение на предыдущем шаге
        uint32_t lastTime;  /// Время прошлого регулирования

        // Выходные переменные
```

Окончание листинга Б.3

```
    float P;
    float I;
    float D;

} PIDRegulatorData;

void PID_Init_Reg(PIDRegulatorData* pid);
void PID_RegulationStep(PIDRegulatorData* pid);

#endif /* PIDREGULATOR_H_ */
Main.c
/* USER CODE BEGIN Header */
/**
```

Листинг Б.4 – Листинг основной программы

```
  ******************************************************************
**************
    * @file           : main.c
    * @brief          : Main program body

******************************************************************
************
    * @attention
    *
    * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
    * All rights reserved.</center></h2>
    *
    * This software component is licensed by ST under BSD 3-
Clause license,
    * the "License"; You may not use this file except in
compliance with the
    * License. You may obtain a copy of the License at:
    *                        opensource.org/licenses/BSD-3-Clause

******************************************************************
************
    */
/* USER CODE END Header */

/* Includes ------------------------------------------------
---------------*/
#include "main.h"
#include "port.h"

/* Private includes ----------------------------------------
---------------*/
/* USER CODE BEGIN Includes */
#include "STM32f4xx_hal.h"
```

Продолжение листинга Б.4

```
  #include "mb.h"
  #include "mbport.h"

  #include <LCD/LCDdrive.h>
  #include <util.h>

  #include "PIDregulator.h"

  #include "menu.h"
  #include "buttons.h"
  #include "settingsmenu.h"
  #include "measurementmenu.h"
  /* USER CODE END Includes */

  /* Private typedef -----------------------------------------
---------------*/

  /* USER CODE BEGIN PTD */

  /* USER CODE END PTD */

  /* Private define ------------------------------------------
---------------*/
  /* USER CODE BEGIN PD */
  #define REG_INPUT_START 10
  #define REG_INPUT_NREGS 4
  /* USER CODE END PD */

  /* Private macro -------------------------------------------
---------------*/
  /* USER CODE BEGIN PM */

  /* USER CODE END PM */

  /* Private variables ---------------------------------------
---------------*/
  static uint32_t lock_nesting_count = 0;
  static USHORT   usRegInputStart = REG_INPUT_START;
  static USHORT   usRegInputBuf[REG_INPUT_NREGS];

  TIM_HandleTypeDef TMR4;
  TIM_OC_InitTypeDef PWM4;
  TIM_HandleTypeDef htim7;
  PIDRegulatorData myPID;
  /* USER CODE BEGIN PV */

  /* USER CODE END PV */

  /* Private function prototypes -----------------------------
---------------*/
```

Продолжение листинга Б.4

```
  void SystemClock_Config(void);
  static void MX_GPIO_Init(void);
  void init_PWM(void);
  void init_work_tim (void);
  void GetMeasurement (void);
  void GetRegModbus(void);
  /* USER CODE BEGIN PFP */

  /* USER CODE END PFP */

  /* Private user code ---------------------------------------
---------------*/
  /* USER CODE BEGIN 0 */

  /* USER CODE END 0 */

  /**

   * @brief The application entry point.
    * @retval int
    */
  int main(void)
  {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration---------------------------------------
---------------*/
    /* Reset of all peripherals, Initializes the Flash interface
and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */

    if (eMBInit( MB_RTU, 0x11, 0, 38400, MB_PAR_NONE) !=
MB_ENOERR)
```

Продолжение листинга Б.4

```
{
    Error_Handler();

 }

  /* Enable the Modbus Protocol Stack. */
  if (eMBEnable(  ) != MB_ENOERR) {
     Error_Handler();
  }
   initTMR_for_pause(); // TIM6 для паузы в us
   initLCD();
   fill_LCD();

   init_PWM();
   PID_Init_Reg (&myPID);
   init_work_tim();

   ScreenManager_Init();

/* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
     ( void )eMBPoll(  );
     ScreenManager();
     GetMeasurement();
     GetRegModbus();

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
  /** Initializes the CPU, AHB and APB busses clocks
```

Продолжение листинга Б.4

```
    */
     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
     RCC_OscInitStruct.PLL.PLLM = 8;
     RCC_OscInitStruct.PLL.PLLN = 240;
     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
     RCC_OscInitStruct.PLL.PLLQ = 4;
     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
     {
       Error_Handler();
     }
     /** Initializes the CPU, AHB and APB busses clocks
     */
     RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3)
!= HAL_OK)
     {
       Error_Handler();
     }
   }

   /**
     * @brief USART1 Initialization Function
     * @param None
     * @retval None
     */

   /**
     * @brief GPIO Initialization Function
     * @param None
     * @retval None
     */
   static void MX_GPIO_Init(void)
   {
     GPIO_InitTypeDef GPIO_InitStruct = {0};

     /* GPIO Ports Clock Enable */
     __HAL_RCC_GPIOC_CLK_ENABLE();
     __HAL_RCC_GPIOH_CLK_ENABLE();
       HAL_RCC_GPIOA_CLK_ENABLE();
```

Продолжение листинга Б.4

```
    __HAL_RCC_GPIOG_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOE_CLK_ENABLE(); // кнопки

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13|GPIO_PIN_14,
GPIO_PIN_RESET);

    /*Configure GPIO pin : PA4 */
    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    /*Configure GPIO pins : PG13 PG14 */
    GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_12;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF2_TIM4;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1 |GPIO_PIN_2
|GPIO_PIN_3 |GPIO_PIN_4 |
    GPIO_PIN_5 |GPIO_PIN_6 |GPIO_PIN_7 ;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
  }

  /* USER CODE BEGIN 4 */

  void init_PWM(void)
  {
    // частота ШИМ = 60МГц / Period = 3КГц
    __HAL_RCC_TIM4_CLK_ENABLE(); // периферия
    TMR4.Instance = TIM4; // таймер
    TMR4.Init.Period = 19999; // число, до которого считает
счетчик
    TMR4.Init.CounterMode = TIM_COUNTERMODE_UP; //счет вверх
```

Продолжение листинга Б.4

```
      TMR4.Init.Prescaler = 0; // предделитель

      TMR4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1; //частоту
не делим
      HAL_TIM_PWM_Init (&TMR4);  //

      PWM4.OCMode  = TIM_OCMODE_PWM1; // 2pwm - с выравниванием по
центру
      PWM4.Pulse = 0;// к-т заполнения
      PWM4.OCPolarity = TIM_OCPOLARITY_HIGH; // прямой ШИМ, Low -
инвертированный
      PWM4.OCFaSTMode = TIM_OCFAST_ENABLE; // бысрее, не за 5
циклов, а за 3
      HAL_TIM_PWM_ConfigChannel(&TMR4, &PWM4, TIM_CHANNEL_1); //
настроили шим и канал шим

      //HAL_TIM_PWM_Start(&TMR4, TIM_CHANNEL_1); // запустили ШИМ
с PD12
   }
   void init_work_tim (void) { // настройка на 1секунду
      __HAL_RCC_TIM7_CLK_ENABLE();
      htim7.Instance = TIM7;
      htim7.Init.Period = 299; // 100мс 10Гц
      htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
      htim7.Init.Prescaler = 19999;
      htim7.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
      HAL_NVIC_SetPriority(TIM7_IRQn, 9, 0);
      HAL_NVIC_EnableIRQ(TIM7_IRQn);
      HAL_TIM_Base_Init(&htim7);
   }
   void GetMeasurement (void) {
      ParameterMenuItem* item1 = (ParameterMenuItem*
)Menu_Settings.items[2];
      ZadTemp  = item1->currentValue;  // получили температуру
заданную

      Kzapoln = PWM4.Pulse * 100 / TMR4.Init.Period ;
      if (Kzapoln > 100) {
         Kzapoln = 100;
      }
   }

   void GetRegModbus (void) {
        usRegInputBuf[0] = ZadTemp;
        usRegInputBuf[1] = IzmTemp;
        usRegInputBuf[2] = Kzapoln;
        if  (temperature_rise) {
           usRegInputBuf[3] = 1;
        } else {
           usRegInputBuf[3] = 0;
```

Продолжение листинга Б.4

```
        }
    }
    void __critical_enter(void)
    {
        __disable_irq();
        ++lock_nesting_count;
    }
    void __critical_exit(void)
    {
        /* Unlock interrupts only when we are exiting the outermost
nested call. */
        --lock_nesting_count;
        if (lock_nesting_count == 0) {
            __enable_irq();
        }
    }

    eMBErrorCode
    eMBRegInputCB( UCHAR * pucRegBuffer, USHORT usAddress, USHORT
usNRegs )
    {
        eMBErrorCode    eStatus = MB_ENOERR;
        int             iRegIndex;

        if( ( usAddress >= REG_INPUT_START )
            && ( usAddress + usNRegs <= REG_INPUT_START +
REG_INPUT_NREGS ) )
        {
            iRegIndex = ( int )( usAddress - usRegInputStart );
            while( usNRegs > 0 )
            {
                *pucRegBuffer++ =
                    ( unsigned char )( usRegInputBuf[iRegIndex] >>
8 );
                *pucRegBuffer++ =
                    ( unsigned char )( usRegInputBuf[iRegIndex] &
0xFF );
                iRegIndex++;
                usNRegs--;
            }
        }
        else
        {
            eStatus = MB_ENOREG;
        }

        return eStatus;
    }

    eMBErrorCode
```

Продолжение листинга Б.4

```
  eMBRegHoldingCB( UCHAR * pucRegBuffer, USHORT usAddress, USHORT
usNRegs,
                   eMBRegisterMode eMode )
  {
      return MB_ENOREG;
  }

  eMBErrorCode
  eMBRegCoilsCB( UCHAR * pucRegBuffer, USHORT usAddress, USHORT
usNCoils,
                 eMBRegisterMode eMode )
  {
      return MB_ENOREG;
  }
  eMBErrorCode

  eMBRegDiscreteCB( UCHAR * pucRegBuffer, USHORT usAddress,
USHORT usNDiscrete )
  {
      return MB_ENOREG;
  }
  /* USER CODE END 4 */

  /**
    * @brief  This function is executed in case of error
occurrence.
    * @retval None
    */
  void Error_Handler(void)
  {
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL
error return state */

    /* USER CODE END Error_Handler_Debug */
  }
  #ifdef  USE_FULL_ASSERT
  /**
    * @brief  Reports the name of the source file and the source
line number
    *         where the assert_param error has occurred.
    * @param  file: pointer to the source file name
    * @param  line: assert_param error line source number
    * @retval None
    */
  void assert_failed(uint8_t *file, uint32_t line)
  {
    /* USER CODE BEGIN 6 */
```

Окончание листинга Б.4

```
   /* User can add his own implementation to report the file name
and line number,
        tex: printf("Wrong parameters value: file %s on line
%d\r\n", file, line) */
     /* USER CODE END 6 */
   }
   #endif /* USE_FULL_ASSERT */
```

Листинг Б. 5 – Листинг меню настроек

```
   /************************ (C) COPYRIGHT STMicroelectronics
*****END OF FILE****/
   Settingsmenu.c

   #include "util.h"
   #include "STM32f4xx_hal.h"
   #include "menu.h"
   #include <stddef.h>
   #include <LCD/LCDdrive.h>
   #include <Buttons.h>

   #include <mainmenu.h>

   static void OnEnter(void);
   static void OnProcessKeys(void);
   static void OnDraw(void);
   static void BackToPreviousMenu(void);

   static const char* MODBUS_TYPES[] = {
       "MODBUS RTU",
       "MODBUS ASCII",
   };
   static ParameterMenuItem* items[] = {
         &(ParameterMenuItem){{"Интерфейс", true}, 0, 0, 0, 1,
NULL, MODBUS_TYPES, 0, 0},
         &(ParameterMenuItem){{"Адрес слейв устр-ва", true}, 1, 0,
0, 20, NULL, NULL, 17, 17},
         &(ParameterMenuItem){{"Заданная температура", true}, 1,
0, 0, 100, NULL, NULL, 70, 70},
         &(ParameterMenuItem){{"Допустимая ошибка", true}, 1, 0,
0, 5, NULL, NULL, 0, 0}
   };

Settingsmenu.h

   typedef struct {
      MenuItem menuItem;
```

Окончание листинга Б. 5

```
      int paramType;           /// Тип параметра: 1 – со вводом
значения; 0 – с выбором из списка
      int dotPosition;       /// Количество символов после запятой
      int minValue;          /// Минимально допустимое значение
параметра
      int maxValue;          /// Максимально допустимое значение
параметра
      const char* units;       /// Указатель на строку с единицей
измерения (NULL если нет)
      const char** paramNames; /// Указатель на массив строк с
названиями значений. Используется при типе 0.
      int defaultValue;        /// Значение по умолчанию
      int currentValue;        /// Текущее значение. При

инициализации структуры не важно.

   } ParameterMenuItem;
```