

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

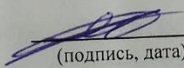
на тему

**КОНВЕРТЕР СПЕЦИАЛИЗИРОВАННЫХ
ПРОТОКОЛОВ ПЕРЕДАЧИ ДАННЫХ ПО
ПОСЛЕДОВАТЕЛЬНОМУ ПОРТУ**

Студент

 31.05.2023 В.К. Трошкин
(подпись, дата)

Руководитель

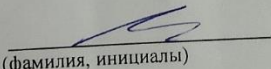
 31.05.2023 М.П. Синёв
(подпись, дата)

Нормоконтролёр

 31.05.2023 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 02.06.23 г. № 14)

Заведующий кафедрой
(подпись)

 М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

20.06.2023 г. № 26)

Секретарь ГЭК


(подпись)

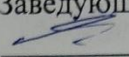
Е.И. Гурин
(фамилия, инициалы)

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« 26 » апреля 20 23 г.

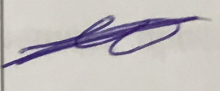
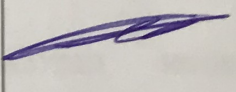
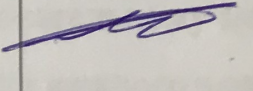
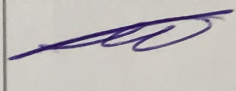
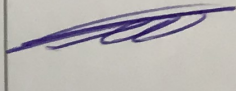
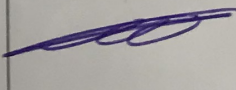
**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Трошкин Владимир Константинович гр 19ВВП1 факультета вычислительной техники.
2. Тема работы Конвертер специализированных протоколов передачи данных по последовательному порту
Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.
3. Руководитель бакалаврской работы к.т.н., доцент, каф. ВТ ПГУ Синёв М.П.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать конвертер специализированных протоколов передачи данных по последовательному порту, который позволит подключать источник данных, работающий по старому протоколу, к устройству обработки данных, работающему по более новому протоколу.

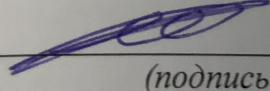
5. Перечень подлежащих разработке вопросов

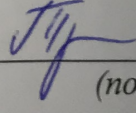
Требуется произвести анализ предметных областей: изучить существующие конверторы протоколов, выделить их достоинства и недостатки. Выбрать язык описания аппаратуры, который подойдет для написания логики конвертера. Определить архитектуру конвертера и вспомогательные средства разработки.

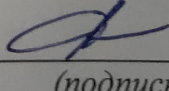
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований	10	01.05.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2023-07.05.2023	
4.	Описание алгоритмов	40	08.05.2023-11.05.2023	
5.	Разработка конвертера специализированных протоколов передачи данных по последовательному порту	150	12.05.2023-29.05.2023	
6.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы  Синёв М.П.
(подпись, дата)

Задание к исполнению принял студент  26.04.23 Трошкин В.К.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)
20.06.2023

Реферат

Бакалаврская работа содержит 87 листов, 30 рисунков, 1 таблицу, 5 источников и 3 приложения

ПРЕОБРАЗОВАНИЕ ДАННЫХ, КОНВЕРТЕР ПРОТОКОЛОВ, ПЛИС, VHDL, FPGA

Цель работы – разработка конвертера специализированных протоколов передачи данных по последовательному порту.

Объектом разработки является конвертер специализированных протоколов передачи данных по последовательному порту.

В результате проделанной работы был разработан конвертер специализированных протоколов передачи данных по последовательному порту в виде модуля на языке VHDL. Проведено тестирование и отладка каждого элемента приложения.

					ПГУ1.090301.23.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб		Трошкин В.К.			Лит.	Лист	Листов
Провер.		Синёв М.П.				2	87
Н. Контр		Бычков А.С.			ФВТ, каф. ВТ гр. 19ВВП1		
Утверд		Митрохин М.А.					

Конвертер специализированных
протоколов передачи данных по
последовательному порту
Пояснительная записка

Содержание

Перечень сокращений и обозначений.....	8
Введение	9
1 Конвертер специализированных протоколов передачи данных по последовательному порту.....	10
1.1 Постановка задачи.....	10
1.1.1 Протокол передачи данных, используемый в старой аппаратуре	10
1.1.2 Протокол передачи данных, используемый в новой аппаратуре	11
1.2 Анализ существующих аналогов.....	12
1.3 Выбор средств реализации.....	12
1.3.1 Программируемые логические интегральные схемы.....	12
1.3.2 Заказные микросхемы.....	13
1.3.3 Микроконтроллеры.....	14
1.4 Выбор языка описания аппаратуры интегральных схем	17
1.4.1 VHDL.....	17
1.4.2 Verilog.....	19
1.5 Выводы	20
1.6 Выбор инструментов разработки	21
1.6.1 Выбор среды разработки	21
1.7 Архитектура проекта	25
1.8 Разработка проекта.....	26
1.8.1 Разработка блока приёма посылок и проверки бита чётности.....	27
1.8.2 Разработка блока проверки инверсной посылки и трёх последних бит	27
1.8.3 Разработка блока передачи посылок по выходному протоколу	27
1.9 Тестирование и отладка.....	29
1.9.1 Симуляция.....	29
1.9.2 Тестирование на отладочной плате.....	37
2 Охрана труда.....	42
2.1 Анализ негативных факторов, воздействующих на оператора ПЭВМ и способы их устранения.....	42

2.1.1 Оборудование рабочего места	42
2.1.2 Освещенность рабочего места	44
2.1.3 Электробезопасность	45
2.1.4 Пожарная безопасность	45
2.1.5 Режим труда и отдыха	46
Заключение	48
Список использованных источников	49
Приложение А – UML – диаграммы	50
Приложение Б – Листинг программы	55
Приложение В – Презентация	77

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

Акроним – разновидность аббревиатуры, которая образуется по первым буквам слов, входящих в зашифрованное понятие

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств

ПК – персональный компьютер

ПЛИС – программируемая логическая интегральная схема

ПИПиТПБ – проверка инверсной посылки и трёх последних бит

ППиПБЧ – приём посылок и проверка бита чётности

ПППВП – передача посылок по выходному протоколу

ПЭВМ – персональная электронно-вычислительная машина

FPGA (field-programmable gate array) – программируемая пользователем вентильная матрица

I²C (Inter-Integrated Circuit) – последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов

IDE (Integrated Development Environment) – интегрированная среда разработки

SPI (Serial Peripheral Interface) – последовательный периферийный интерфейс

Testbench – код для проверки проекта при симуляции

UART (Universal Asynchronous Receiver-Transmitter) – универсальный асинхронный приёмопередатчик

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

VHDL (Very high speed integrated circuits Hardware Description Language) – язык описания аппаратуры высокоскоростных интегральных схем

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

Введение

Цель данной работы — разработка конвертера специализированных протоколов передачи данных по последовательному порту, который позволит подключать источник данных, работающий по старому протоколу, к устройству обработки данных, работающему по более новому протоколу.

Выбранная тема очень важна, так как проблема несовместимости старого и нового оборудования в настоящее время является актуальной и распространенной. С появлением новых технологий и новых протоколов передачи данных, многие предприятия и организации вынуждены обновлять свое оборудование, чтобы оно соответствовало новым требованиям и могло работать в новой среде, ведь когда устройство обработки данных работает с использованием нового протокола, а источник данных работает по старому протоколу, возникает проблема несовместимости. Из-за этой проблемы устройства обработки данных не могут принимать данные от источников, работающих по старому протоколу.

Решить проблему несовместимости оборудования можно путем создания устройства для преобразования протоколов передачи данных. Оно позволит сохранить старое оборудование и использовать его совместно с новым, что повысит эффективность и экономичность использования вычислительных систем, в которых оно задействовано.

В рамках данной темы разработан модуль на языке *VHDL*, реализующий конвертер протоколов, преобразовывающий данные, передаваемые по старому протоколу, в данные, которые могут быть обработаны устройством, работающим по новому протоколу. Этот модуль может послужить основой при разработке устройства для преобразования протоколов передачи данных или может быть использован при разработке новой аппаратуры для обеспечения её совместимости со старой.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

1 Конвертер специализированных протоколов передачи данных по последовательному порту

Практическая значимость этой темы в том, что разработанный в её рамках конвертер позволяет использовать старое оборудование вместе с более современным, что избавляет от финансовых затрат на замену оборудования. Конвертер позволяет использовать уже находящуюся в эксплуатации аппаратуру, которая является проверенной и привычной, а значит, для работы с ней не придётся тратить время и деньги на переподготовку людей, совместно с новым оборудованием, обладающим широкой функциональностью.

1.1 Постановка задачи

Требуется разработать устройство, позволяющее передавать данные между устройствами, поддерживающими различные протоколы передачи данных. В нашем случае аппаратурой как старого, так и нового парка используются специализированные, то есть нестандартные протоколы передачи данных. Старая аппаратура — источник данных — выдаёт их согласно протоколу, описанному ниже.

1.1.1 Протокол передачи данных, используемый в старой аппаратуре

Информация выдаётся в последовательном виде асинхронными старт-стопными посылками. Формат посылки – 1 стартовый бит, 8 информационных, 1 бит контроля на чётность, 2 стоповых бита. Скорость передачи – 9600 бод. Структура старт-стопной посылки изображена на рисунке 1.

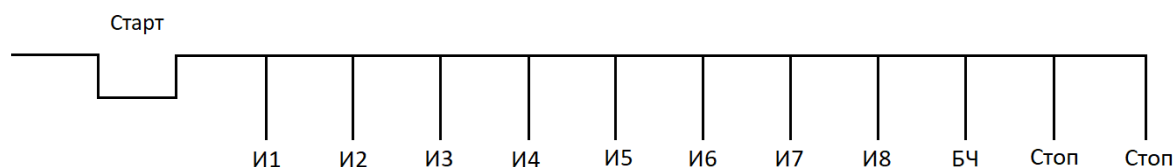


Рисунок 1 – Структура старт-стопной посылки

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

Посылка имеет следующую структуру:

- Старт – стартовый бит;
- И1-И8 – информационные биты;
- БЧ – бит контроля чётности;
- Стоп – стоповые биты.

С целью повышения надёжности каждая посылка передаётся дважды: сначала в прямом, затем в инверсном виде. Инверсии подвергаются информационные биты И1-И8.

Информационная посылка должна иметь следующие структуры:

- прямой код: И1, И2, И3, И4, И5, И6 = 0, И7 = 0, И8 = 0;
- инверсный код: $\neg$ И1, $\neg$ И2, $\neg$ И3, $\neg$ И4, $\neg$ И5, И6 = 1, И7 = 1, И8 = 1.

1.1.2 Протокол передачи данных, используемый в новой аппаратуре

Новая аппаратура, которая должна принять и обработать данные, использует следующий протокол передачи данных: информация выдаётся в последовательном виде асинхронными старт-стопными посылками. Формат посылки – 1 стартовый бит, 8 информационных, 1 бит контроля на чётность, 2 стоповых бита. Скорость передачи – 9600 бод. Структура старт-стопной посылки аналогична изображённой на рисунке 1.

Данные передаются в виде пакета.

Первые два байта – заголовок, первый байт которого равен 55_{16} , второй байт равен 41_{16} .

Третий байт – статус, который может содержать следующие коды ошибок:

- 00 – данные не содержат ошибок;
- 01 – обнаружен некорректный бит чётности или стоповый бит;
- 02 – обнаружено несовпадение данных прямой и инверсной посылки;
- 03 – обнаружены некорректные значения бит И6, И7, И8. То есть в прямой посылке биты И6, И7, И8 не равны 000 или в инверсной посылке биты И6, И7, И8 не равны 111.

Четвёртый байт – количество байт данных в пакете плюс 1.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

Затем идут сами байты данных. Если в процессе обработки данных возникает ошибка, в пакете будут байты до этой ошибки. Сам байт, содержащий ошибку, в пакет включён не будет.

Затем идёт нулевой байт.

Последние два байта – контрольная сумма, которая представляет из себя сумму всех байт данных.

1.2 Анализ существующих аналогов

Найти аналог разрабатываемой системе не удалось по причине использования специализированных (нестандартных) протоколов передачи данных.

1.3 Выбор средств реализации

Существует несколько путей реализации подобных устройств: на ПЛИС, на заказной интегральной схеме или на основе микроконтроллера. Каждый подход имеет свои преимущества и недостатки, и выбор зависит от конкретных требований проекта.

1.3.1 Программируемые логические интегральные схемы

ПЛИС (программируемые логические интегральные схемы) — это реконфигурируемые интегральные схемы с программируемыми логическими блоками, обеспечивающие высокопараллельную обработку [1]. Они позволяют разработчикам создавать собственные цифровые устройства на базе программируемой аппаратуры. Основными особенностями разработки устройств на основе ПЛИС являются:

- программируемость. ПЛИС могут быть программируемыми для реализации любых цифровых функций. Разработчик может программировать ПЛИС, используя специальный язык описания аппаратуры (VHDL, Verilog) или графический интерфейс;

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		12

- гибкость. ПЛИС позволяют создавать устройства с высокой степенью гибкости и конфигурируемости. Разработчик может изменять логику и функциональность устройства в процессе разработки, что позволяет быстро прототипировать и улучшать устройства;

- высокая скорость. ПЛИС могут работать на высоких частотах и способны к параллельной обработке данных, что позволяет создавать устройства с высокой скоростью и производительностью. Это особенно полезно для обработки данных в режиме реального времени;

- экономия места на плате. ПЛИС могут заменить большое количество дискретных компонентов на одном устройстве. Это позволяет сделать устройство более компактным и снижает стоимость производства за счёт уменьшения количества компонентов и производственных операций;

- простота разработки. Разработка устройств на основе ПЛИС происходит в программном окружении и на предоставляемых производителями ПЛИС отладочных платах, что позволяет разработчикам быстро создавать и отлаживать цифровые устройства без необходимости создавать и тестировать сложные печатные платы;

- использование существующих решений. Существуют готовые библиотеки компонентов и блоков, которые можно использовать при разработке устройств на основе ПЛИС. Это позволяет сократить время и затраты на разработку новых устройств.

В целом, разработка устройств на основе ПЛИС предоставляет разработчикам гибкость и возможность быстрого прототипирования, а также позволяет создавать устройства с высокой скоростью работы и производительностью.

1.3.2 Заказные микросхемы

Заказные микросхемы разрабатываются и производятся для конкретных задач. Они не могут быть переконфигурированы или перепрограммированы.

Основными преимуществами заказных микросхем являются:

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		13

- более высокая производительность и энергоэффективность по сравнению с реализацией аналогичного устройства на микросхемах общего назначения, так как они специально разработаны для конкретных задач и требований;

- низкие затраты на обслуживание, так как они не нуждаются в дополнительных компонентах или программном обеспечении;

- высокая надежность и долговечность в экстремальных условиях, таких как высокие температуры, вибрации и воздействия окружающей среды.

Однако, у заказных микросхем также есть свои недостатки:

- высокие затраты на разработку и производство, так как они требуют индивидуального подхода и тщательного тестирования;

- ограниченная гибкость и необходимость значительных изменений в случае изменения требований или условий задачи;

- большие сроки разработки и производства.

Таким образом, заказные микросхемы являются эффективным решением для определенных задач, когда требуются высокая производительность, надежность и энергоэффективность, при этом тираж достаточно велик, чтобы оправдать высокие затраты на разработку.

1.3.3 Микроконтроллеры

Микроконтроллеры – это устройства, которые содержат в себе процессор, память, периферийные устройства ввода/вывода и другие компоненты, необходимые для управления электронными системами.

Особенности разработки устройств на основе микроконтроллеров:

- низкий уровень абстракции. Разработка устройств на микроконтроллерах требует знания аппаратной части и низкоуровневого языка программирования. Программирование микроконтроллера на уровне регистров является обычной практикой, что означает, что разработчик должен знать, как работает аппаратная часть микроконтроллера и как работать с регистрами и периферией;

- ограниченные ресурсы. Микроконтроллеры обычно имеют ограниченные ресурсы, такие как оперативная память, объем памяти программ, число доступных входов и выходов и производительность. При разработке устройства необходимо

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		14

тщательно планировать использование ресурсов, оптимизировать код и избегать излишнего использования памяти;

- питание. Микроконтроллеры обычно требуют низкого напряжения питания, что может ограничить возможности выбора источника питания устройства. Также необходимо учитывать потребление энергии при проектировании устройства, особенно если оно будет работать от батарей;

- средства отладки. Отладка устройств на микроконтроллере может быть более сложной, чем отладка программ на компьютере, так как не всегда есть возможность подключить устройство к монитору и клавиатуре. Для отладки могут использоваться специальные отладочные средства, такие как отладочные платы. Однако, микроконтроллеры более просты в проектировании и программировании, чем ПЛИС;

- отсутствие возможности параллельной обработки данных. В отличие от ПЛИС, микроконтроллеры зачастую не имеют возможности обрабатывать данные параллельно, и поэтому плохо подходят для устройств, которые должны обрабатывать большие объёмы данных с высокой скоростью, например, преобразователей цифрового сигнала;

- интерфейсы ввода и вывода. Микроконтроллеры часто используются для управления различными устройствами, поэтому они обычно имеют множество интерфейсов ввода и вывода, таких как *UART*, *I2C*, *SPI* и другие. Разработчики должны иметь хорошее понимание этих интерфейсов и уметь работать с ними, чтобы подключить микроконтроллер к другим устройствам;

- работа в реальном времени. Микроконтроллеры могут использоваться для реализации систем, которые должны работать в реальном времени, но не требуют высокой производительности, например, систем автоматического управления и мониторинга. Разработчики должны учитывать требования к реальному времени при проектировании устройства и использовать соответствующие алгоритмы и структуры данных;

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		15

- необходимость тестирования. Как и при разработке любого другого устройства, тестирование является важным этапом в разработке устройств на микроконтроллерах. Тестирование может быть более сложным, чем при разработке программ на компьютере, поскольку микроконтроллеры часто работают в реальном времени и могут быть подключены к другим устройствам. При тестировании необходимо учитывать эти особенности и использовать соответствующие методы и инструменты.

Подробнее опишем основные различия и особенности разработки устройств на основе микроконтроллеров по сравнению с ПЛИС.

Архитектура. Основное отличие микроконтроллеров и ПЛИС заключается в их архитектуре. Микроконтроллеры имеют программно-ориентированную архитектуру, в которой программа управляет работой устройства. ПЛИС имеют аппаратно-ориентированную архитектуру, в которой функциональность устройства определяется конфигурационным файлом, а не программой.

Гибкость. ПЛИС более гибкие, чем микроконтроллеры, потому что они могут быть и перепрограммированы, и переконфигурированы для выполнения новых функций. Микроконтроллеры имеют фиксированный набор инструкций и могут быть изменены только путем обновления программного обеспечения.

Сложность. ПЛИС могут быть сложными в проектировании и требуют хорошего знания аппаратного обеспечения и языков описания аппаратуры, чтобы создавать эффективные конфигурационные файлы. Микроконтроллеры более просты в проектировании и программировании.

Ресурсы. ПЛИС имеют больше ресурсов, чем микроконтроллеры, такие как большее количество логических элементов, более быстрые сигнальные цепи и т.д. Это делает их более подходящими для высокопроизводительных и сложных приложений.

Стоимость. Микроконтроллеры обычно стоят дешевле, чем ПЛИС, что делает их более доступными для широкого круга разработчиков и применений.

					<i>ПГУ1.090301.23.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		16

Среда разработки. Для разработки устройств на основе микроконтроллеров используются интегрированные среды разработки (*IDE*), которые предоставляют различные инструменты и библиотеки для разработки приложений. Для ПЛИС используются специализированные программы для создания конфигурационных файлов.

Сроки разработки. Разработка устройств на основе микроконтроллеров обычно занимает меньше времени, чем разработка на ПЛИС, потому что *IDE* содержат много готовых библиотек и инструментов, которые упрощают процесс разработки. Однако, если проект требует высокой производительности или специфической функциональности, то разработка на ПЛИС может занять меньше времени.

Для разработки конвертера специализированных протоколов передачи данных по последовательному порту выберем ПЛИС, так как сравнимые по стоимости микроконтроллеры не обеспечивают необходимой производительности, а затраты на разработку устройства на заказных микросхемах были бы неприемлемы ввиду небольшого тиража разрабатываемого устройства, обусловленного узким распространением протоколов, с которыми оно работает.

1.4 Выбор языка описания аппаратуры интегральных схем

Существует несколько языков описания аппаратуры интегральных схем, наиболее распространенными из которых являются *VHDL (Very High-Speed Integrated Circuit Hardware Description Language)* и *Verilog*.

1.4.1 VHDL

VHDL (Very High-Speed Integrated Circuit Hardware Description Language) — это язык описания аппаратуры, который используется для моделирования и проектирования цифровых электронных систем и интегральных схем. Он представляет собой формальный язык программирования, используемый для описания поведения и структуры цифровых систем [2].

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		17

VHDL был разработан в 1983 году. Он широко используется инженерами в области электроники для разработки и верификации цифровых систем, таких как микропроцессоры, цифровые сигнальные процессоры, программируемые логические интегральные схемы и других ИС.

VHDL позволяет описывать поведение и структуру цифровых систем на разных уровнях абстракции. Он поддерживает описание входов, выходов, внутренних состояний, комбинационных логических схем и последовательностей команд. *VHDL* также обладает возможностями для создания модульных иерархических структур, а также для проверки и верификации разработанных цифровых систем.

Основное применение *VHDL* связано с разработкой и проектированием интегральных схем, но его также можно использовать для разработки программно-аппаратных систем, эмуляции и тестирования цифровых систем, а также для создания формальных спецификаций и документации.

VHDL является стандартом в индустрии электронного проектирования и широко применяется инженерами для разработки сложных цифровых систем и интегральных схем.

Плюсы *VHDL*:

- более строгий и формальный синтаксис, что облегчает верификацию и тестирование;
- поддержка моделирования аппаратуры на разных уровнях абстракции (от низкоуровневых деталей до высокоуровневого поведения);
- мощная типизация и возможность описания сложных данных;
- поддержка иерархического описания схем, что упрощает создание крупных проектов.

Минусы *VHDL*:

- более сложный для изучения и использования в сравнении с *Verilog*;
- больше кода требуется для выполнения простых задач, что может привести к более долгому процессу разработки;

					ПГУ1.090301.23.001 ПЗ	Лист
						18
Изм.	Лист	№ докум.	Подпись	Дата		

- менее популярен в некоторых отраслях, таких как проектирование цифровых систем на кристалле.

1.4.2 Verilog

Verilog - это язык описания аппаратуры, используемый для моделирования, проектирования и верификации цифровых электронных систем и интегральных схем [3].

Verilog был разработан в 1984 компанией *Gateway Design Automation*. В настоящее время существует несколько версий языка, таких как *Verilog-1995*, *Verilog-2001*, *SystemVerilog* и другие.

Язык *Verilog* предоставляет средства для описания поведения и структуры цифровых систем на разных уровнях абстракции. Он позволяет описывать входы, выходы, комбинационные и последовательные логические элементы, модули, иерархические структуры и многое другое. *Verilog* также поддерживает различные уровни абстракции, включая уровни функциональной абстракции, поведенческой абстракции и структурной абстракции.

Основное применение *Verilog* связано с разработкой и верификацией цифровых систем, таких как микропроцессоры, цифровые сигнальные процессоры, программируемые логические устройства и другие интегральные схемы. *Verilog* также может использоваться для создания программно-аппаратных систем, эмуляции и тестирования цифровых систем, а также для создания формальных спецификаций и документации [4].

Плюсы *Verilog*:

- более простой для изучения и использования, особенно для тех, кто уже знаком с языком программирования C;
- широкое распространение в индустрии и активная поддержка от различных поставщиков интегральных схем;
- лаконичный синтаксис, что облегчает написание кода и повышает производительность разработки;
- большое сообщество пользователей, что обеспечивает доступ к обширным библиотекам и ресурсам.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		19

Минусы *Verilog*:

- ограниченная поддержка некоторых продвинутых функций, которые могут быть полезны при проектировании сложных систем;
- более слабая типизация, что может затруднить верификацию и обнаружение ошибок;
- отсутствие стандартизированного спецификационного документа, что может вызывать различия в интерпретации синтаксиса различными инструментами.

1.5 Выводы

Существует несколько причин, почему стоит выбрать *VHDL* для описания аппаратуры интегральных схем:

- формальный и строгий синтаксис. *VHDL* имеет более формальный синтаксис по сравнению с *Verilog*. Это может быть полезно при верификации и тестировании, так как строгий синтаксис облегчает обнаружение и предотвращение ошибок в описании аппаратуры;
- поддержка различных уровней абстракции. *VHDL* позволяет описывать аппаратуру на разных уровнях абстракции, начиная от низкоуровневых деталей до высокоуровневого поведения. Это позволяет более гибко моделировать и проектировать сложные системы;
- мощная типизация и описание данных. *VHDL* обладает богатой системой типов и позволяет описывать сложные структуры данных. Это особенно полезно при работе с большими и сложными проектами, где необходимо обрабатывать различные типы данных;
- иерархическое описание. *VHDL* поддерживает иерархическое описание схем, что упрощает создание и управление крупными проектами. Возможность разбить проект на модули и подмодули облегчает понимание и поддержку кода;
- устойчивость к изменениям. *VHDL* является стандартизированным языком, что означает, что он менее подвержен изменениям и эволюции, чем *Verilog*.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		20

Это может быть важным фактором, особенно для долгосрочных проектов или в отраслях, где стабильность языка имеет значение.

В итоге, выбор *VHDL* для описания аппаратуры интегральных схем обусловлен его формальным синтаксисом, поддержкой различных уровней абстракции, мощной типизацией, иерархическим описанием и устойчивостью к изменениям.

1.6 Выбор инструментов разработки

1.6.1 Выбор среды разработки

Существует несколько популярных сред разработки, моделирования и верификации проектов для программируемых логических интегральных схем (ПЛИС). Приведём сравнение нескольких из них: *Vivado*, *Quartus Prime* и *Libero SoC*.

Vivado (ранее *ISE*) — это интегрированная среда разработки (*IDE*), предоставляемая компанией *Xilinx*, для создания, настройки и развертывания программируемых логических устройств (ПЛИС) и систем на кристалле на базе фирменных чипов *Xilinx*. Она используется для проектирования и верификации цифровых систем, таких как процессоры, программируемые матрицы, периферийные устройства и другие компоненты. *Vivado* предлагает широкий спектр функциональных возможностей, включая создание схемы, синтез, оптимизацию, размещение, трассировку маршрута и генерацию конфигурационных файлов для программирования ПЛИС или систем на кристалле. Она также поддерживает различные языки описания аппаратуры, такие как *VHDL* и *Verilog*, для описания поведения и структуры цифровых систем. *Vivado* является мощным инструментом, который помогает разработчикам создавать и настраивать программируемую логику для широкого спектра приложений.

Плюсы *Vivado*:

- имеет богатый функционал и широкий набор инструментов для разработки *FPGA*, включая синтез, размещение, трассировку, симуляцию и визуализацию;

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		21

- хорошо интегрируется с другими инструментами и IP-библиотеками от *Xilinx*, что упрощает разработку сложных систем;
- поддерживает большой выбор *FPGA*-чипов от *Xilinx*.

Минусы *Vivado*:

- интерфейс *Vivado* может быть сложным для новичков и требует времени для изучения;
- некоторые пользователи отмечают, что *Vivado* может быть медленным и требовательным к ресурсам компьютера.

Quartus Prime - это интегрированная среда разработки (*IDE*), разработанная компанией *Intel* (ранее *Altera*), для программирования и конфигурирования программируемых логических интегральных схем (ПЛИС) и комплексных программируемых вентильных матриц. Она предоставляет инструменты для создания, симуляции, оптимизации и компиляции цифровых схем, которые могут быть реализованы на ПЛИС или *FPGA*.

Quartus Prime поддерживает широкий спектр устройств *Intel FPGA*, включая *Stratix*, *Arria* и *Cyclone*. Эта среда разработки позволяет инженерам проектировать сложные цифровые системы, используя графический интерфейс или язык описания аппаратуры, такой как *VHDL* или *Verilog*.

Основные возможности *Quartus Prime* включают в себя:

- создание схем. *Quartus Prime* предоставляет инструменты для создания схем на уровне блоков, которые можно соединять и конфигурировать для реализации желаемой функциональности;
- симуляция. *IDE* позволяет проводить функциональную и временную симуляцию разработанных цифровых схем, чтобы проверить их работоспособность и удовлетворение требованиям;
- оптимизация и компиляция. *Quartus Prime* предоставляет возможность оптимизировать цифровую схему. Затем среда компилирует схему в конфигурационный файл, который можно загрузить на ПЛИС или *FPGA*;

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

- верификация. *Quartus Prime* поддерживает функции верификации и отладки, позволяя инженерам проверять и исправлять ошибки в цифровых схемах перед их развертыванием на аппаратной платформе;

- управление проектами. С помощью *Quartus Prime* можно управлять проектами, организовывать и структурировать файлы, а также управлять зависимостями между компонентами проекта.

Quartus Prime является мощным инструментом для разработки и конфигурирования цифровых схем на ПЛИС и *FPGA* от *Intel FPGA*, и широко используется в индустрии электроники и разработки программируемых логических устройств.

Плюсы *Quartus Prime*:

- поддерживает *FPGA*-чипы от *Intel* (ранее *Altera*) и позволяет эффективно использовать их функциональность;

- имеет мощные средства синтеза, размещения, трассировки и симуляции, что позволяет создавать сложные цифровые системы;

- интегрируется с другими инструментами *Intel*, такими как сложные функциональные блоки и система анализа *SignalTap*.

Минусы *Quartus Prime*:

- у *Quartus Prime* может быть сложный интерфейс, особенно для новых пользователей;

- некоторые пользователи отмечают, что инструменты *Quartus Prime* могут быть менее стабильными и иметь проблемы с производительностью.

Libero SoC — это программное обеспечение, разработанное компанией *Microsemi* (ныне в составе *Microchip Technology*) для проектирования и разработки систем на кристалле на основе ПЛИС.

Libero SoC предоставляет интегрированную среду разработки, которая позволяет инженерам проектировать, верифицировать и развертывать проекты систем на кристалле с использованием *FPGA* и других компонентов на одном кристалле. Она предоставляет инструменты для проектирования схем, создания

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		23

блоков схемы, написания программного обеспечения и проверки функциональности системы.

Одним из ключевых компонентов *Libero SoC* является инструмент *Libero SoC Design Suite*, который предоставляет широкий набор функций, включая среду разработки, симуляцию, анализ времени выполнения, синтез и размещение элементов, а также программирование и конфигурирование *FPGA*.

Libero SoC обеспечивает разработчикам гибкость и удобство при создании систем на кристалле с использованием *FPGA*, позволяя им оптимизировать производительность, потребляемую мощность и другие параметры для конкретных требований и приложений.

Плюсы *Libero SoC*:

- хорошо интегрируется с *FPGA*-чипами от *Microsemi*, такими как серия *SmartFusion* и *IGLOO*;
- предоставляет широкий выбор инструментов для разработки, включая синтез, размещение, трассировку и симуляцию;
- поддерживает различные протоколы связи и включает функциональность встроенной системы проектирования.

Минусы *Libero SoC*:

- *Libero SoC* имеет более ограниченную поддержку *FPGA*-чипов по сравнению с *Vivado* и *Quartus Prime*;
- Интерфейс *Libero SoC* может быть менее интуитивным и менее продвинутым по сравнению с другими инструментами.

Поскольку доступная отладочная плата имеет на борту кристалл семейства *Spartan 3A* производства *Xilinx*, для разработки будем использовать среду от *Xilinx*. Поскольку *Spartan 3A* не поддерживается современными версиями *Vivado*, использовать будем *ISE 14.7*. Однако для более быстрой симуляции и удобной отладки используем симулятор *Active-HDL*.

					ПГУ1.090301.23.001 ПЗ	Лист
						24
Изм.	Лист	№ докум.	Подпись	Дата		

1.7 Архитектура проекта

Общая архитектура проекта изображена на рисунке 2 в виде *UML* диаграммы активности [5]. Данные, соответствующие старому протоколу, поступают на блок приёма посылок и проверки бита чётности, который переводит данные из последовательного в параллельный вид и проверяет бит чётности. Если бит верный, байт данных передаётся на блок проверки инверсной посылки и трёх последних бит. Иначе на блок передачи посылок по выходному протоколу передаётся сигнал об ошибке. Блок проверки инверсной посылки и трёх последних бит ждёт пару байт и проверяет, соответствует ли инверсия второго байта первому. Также этот блок проверяет последние три бита, которые должны быть нулями в первом байте и единицами во втором байте. Если хотя бы одно из этих условий не выполняется, блок проверки инверсной посылки и трёх последних бит передаёт соответствующий сигнал об ошибке на блок передачи посылок по выходному протоколу. Блок передачи посылок по выходному протоколу формирует пакет с кодом ошибки и теми байтами, которые удалось принять до её возникновения, если получает сигнал об ошибке от предыдущих блоков. Если ошибки не возникает, блок формирует корректный пакет после истечения периода ожидания, составляющего примерно 0,1 секунды с момента приёма последнего байта.

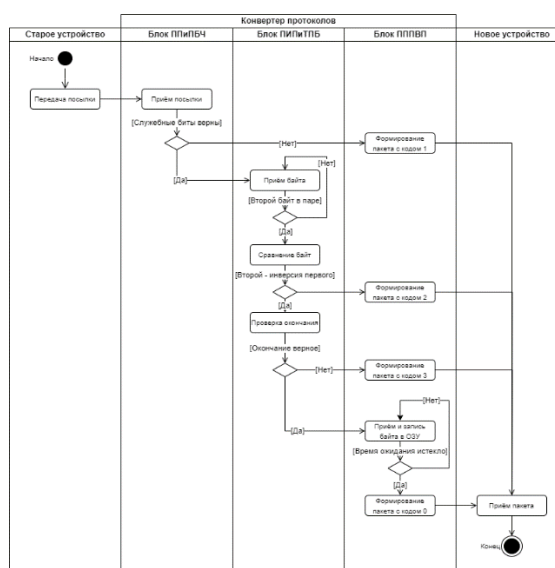


Рисунок 2 – Архитектура проекта

1.8 Разработка проекта

Для наглядного представления функциональных возможностей и взаимодействия между актерами и прецедентами в проекте была составлена диаграмма прецедентов в рамках языка моделирования *UML*, которая представлена на рисунке 3.

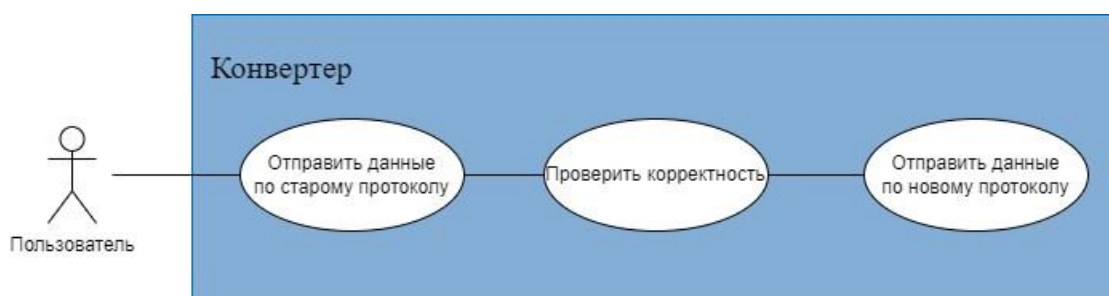


Рисунок 3 – Диаграмма прецедентов

Также для демонстрации последовательности передачи сигналов между блоками была составлена *UML* диаграмма последовательности, которая представлена на рисунке 4.

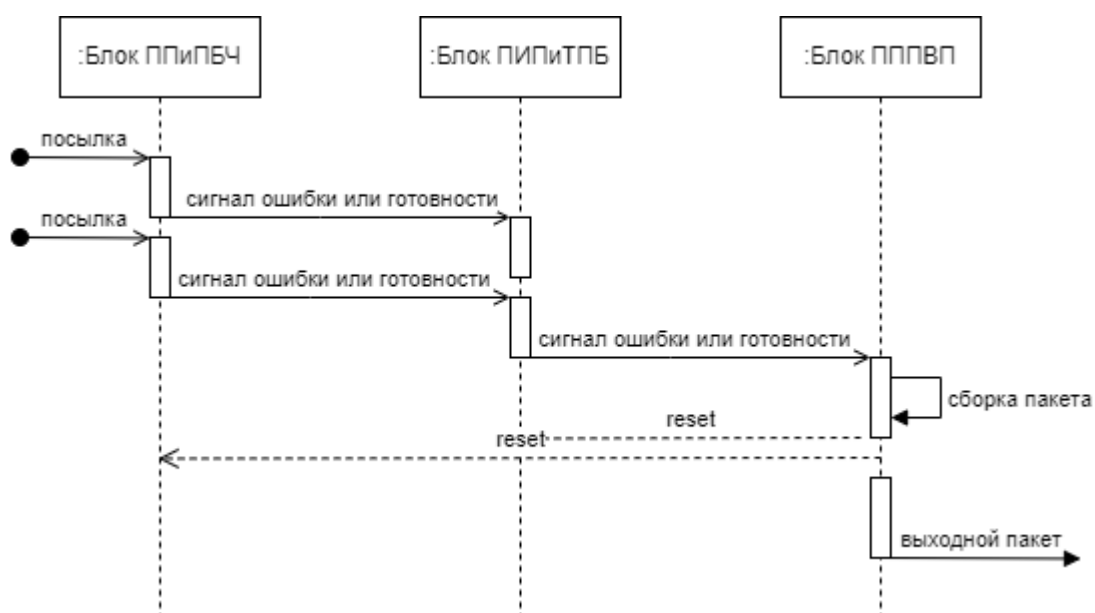


Рисунок 4 – Диаграмма последовательностей

1.8.1 Разработка блока приёма посылок и проверки бита чётности

Блок приёма посылок и проверки бита чётности имеет вспомогательный блок распознавания начала посылки, который выдаёт сигнал, когда единичный уровень на его входе сменяется на нулевой, то есть начинается приём стартового бита посылки. Получив этот сигнал, блок приёма посылок и проверки бита чётности ждёт время, равное половине длительности отправки одного бита информации на скорости 9600 бит/с. По истечении этого времени считывается стартовый бит. Затем блок приёма посылок и проверки бита чётности принимает 8 бит с интервалами, равными полной длительности отправки одного бита информации на скорости 9600 бит/с. Затем с такими же интервалами считываются бит чётности и стоповые биты, проверяется их корректность. Если они корректны, следующему блоку передаётся сигнал о готовности корректного байта и сам байт принятой информации, иначе выдаётся сигнал об ошибке.

1.8.2 Разработка блока проверки инверсной посылки и трёх последних бит

Блок проверки инверсной посылки и трёх последних бит не имеет субблоков. Как только он получает сигнал о готовности корректного байта, он принимает его и ждёт следующего сигнала о готовности. Когда приходит второй сигнал о готовности, блок считывает второй байт. Затем блок сравнивает первый байт с инверсией второго. Если они не совпадают, блок проверки инверсной посылки и трёх последних бит выдаёт сигнал об ошибке инверсии. Если первый байт соответствует инверсии второго, блок проверяет три младших бита первого байта. Если они не равны нулю, блок выдаёт сигнал об ошибке нулей. Младшие три бита второго байта не проверяются, так как предыдущие проверки уже гарантировали, что окончание прямого байта и инверсный байт корректны, а инверсией трёх нулей являются три единицы. Если этих ошибок не возникло, блок выдаёт сигнал о готовности корректного байта и байт, соответствующий первому из принятой пары байт.

1.8.3 Разработка блока передачи посылок по выходному протоколу

Блок передачи посылок по выходному протоколу имеет два вспомогательных блока: блок ОЗУ, в котором формируется пакет,

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		27

соответствующий новому протоколу и блок перевода данных в последовательный вид, который переводит байт в последовательный вид и снабжает его стартовым битом, битом чётности и двумя стоповыми, обеспечивая его передачу по последовательному порту. Блок передачи посылок по выходному протоколу принимает от предыдущего блока байты и записывает их в ОЗУ начиная с адреса 4. С момента приёма последнего байта отсчитывается время ожидания, которое составляет примерно 0,1 секунды. Если время истекло или был получен сигнал об ошибке, блок передачи посылок по выходному протоколу формирует выходной пакет: записывает в ОЗУ заголовок, первый байт которого равен 55_{16} , второй байт равен 41_{16} , начиная с адреса 0.

Затем по адресу 2 записывается статус, который может содержать следующие коды ошибок:

- 00 – данные не содержат ошибок;
- 01 – обнаружен некорректный бит чётности или стоповый бит;
- 02 – обнаружено несовпадение данных прямой и инверсной посылки;
- 03 – обнаружены некорректные окончания байтов. То есть в прямой посылке младшие биты не равны 000 или в инверсной посылке младшие биты не равны 111.

По адресу 3 записывается число, на 1 большее количества байт данных в пакете.

Затем идут сами байты данных. Если в процессе обработки данных возникла ошибка, в пакете будут байты, которые были успешно обработаны до этой ошибки. Сам байт, содержащий ошибку, в пакет включён не будет. По адресу, следующему за последним байтом передаваемых данных, записывается нулевой байт. По следующим двум адресам записываются два байта – контрольная сумма, которая представляет из себя сумму всех байт данных. Затем начинается процесс передачи данных. Блоку перевода данных в последовательный вид поочерёдно передаются байты из ОЗУ, начиная с адреса 0 и до последнего байта посылки. Для передачи каждого байта ему подаётся сигнал, запускающий отправку текущего байта. Затем блок передачи посылок по выходному протоколу ожидает от блока перевода

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		28

данных в последовательный вид сигнал об окончании передачи байта. По этому сигналу блок передачи посылок по выходному протоколу читает из ОЗУ следующий байт данных, который подаётся на вход блока перевода данных в последовательный вид вместе с сигналом, запускающим отправку текущего байта.

1.9 Тестирование и отладка

1.9.1 Симуляция

В процессе разработки возникла необходимость симулировать поведение схемы для оперативной отладки отдельных блоков без задействования отладочной платы. Поскольку *Xilinx ISE* на имеющемся компьютере выполнял построение проекта и симуляцию сравнительно долго, было решено найти инструмент для более быстрой симуляции. В качестве такого инструмента был выбран *Aldec Active-HDL*.

Aldec Active-HDL является одним из популярных инструментов для симуляции и отладки проектов на языке *VHDL*. Вот несколько его плюсов:

- обширные возможности симуляции. *Active-HDL* предоставляет мощные средства симуляции, которые позволяют проверять и проверять правильность работы вашего *VHDL* кода. Он поддерживает различные уровни абстракции, включая поведенческую, структурную и уровень регистровых передач, что обеспечивает гибкость при проведении симуляций;

- интуитивный пользовательский интерфейс. *Active-HDL* имеет интуитивно понятный и легкий в использовании пользовательский интерфейс. Он предлагает графический редактор для создания и модификации схем, а также возможность отладки шаг за шагом, что значительно облегчает процесс разработки и отладки проектов на *VHDL*;

- интеграция с другими инструментами. *Active-HDL* интегрируется с другими популярными инструментами, такими как *Riviera-PRO* (для смешанной симуляции *VHDL/Verilog*), *MATLAB/Simulink* и *ModelSim*. Это обеспечивает большую гибкость и расширенные возможности при работе с проектами;

					ПГУ1.090301.23.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

- поддержка и обновления. *Aldec* предлагает техническую поддержку для пользователей *Active-HDL* и регулярно выпускает обновления программного обеспечения, чтобы улучшить его функциональность и исправить ошибки.

Однако ключевым преимуществом для поставленных задач стала быстрая и удобная симуляция поведения схемы.

Для отладки был разработан испытательный модуль (*testbench*), формирующий на входе разрабатываемого модуля посылки, соответствующие старому протоколу. Для отправки данных он использует блок перевода данных в последовательный вид, также был разработан блок имитации неисправности, позволяющий отправлять посылки с некорректным битом чётности и стоповыми битами. Также имеется возможность отключить отправку инверсных байт.

На рисунке 5 приведено содержание ОЗУ испытательного модуля. Изображены прямые байты корректного тестового пакета в шестнадцатеричном представлении.

	0	1	2	3	4	5	6	7
00	58	60	68	70	78	80	88	90
08	98	A0	A8	B0	B8	C0	C8	D0
10	D8	E0	E8	F0	F8	00	08	10
18	18	20	28	30	38	40	48	50

Рисунок 5 – Корректный тестовый пакет в шестнадцатеричном виде

На рисунке 6 приведено содержание ОЗУ испытательного модуля в двоичном виде, чтобы продемонстрировать, что три младших бита каждого байта равны нулю.

	0	1	2	3	4	5	6	7
00	01011000	01100000	01101000	01110000	01111000	10000000	10001000	10010000
08	10011000	10100000	10101000	10110000	10111000	11000000	11001000	11010000
10	11011000	11100000	11101000	11110000	11111000	00000000	00001000	00010000
18	00011000	00100000	00101000	00110000	00111000	01000000	01001000	01010000

Рисунок 6 – Корректный тестовый пакет в двоичном виде

Выполним симуляцию. На рисунке 7 видно, что за каждым байтом следует инверсный. Сигнал «input» формируется испытательным модулем и подаётся на вход конвертера. Он соответствует старому протоколу. Блок приёма посылок и проверки бита чётности выдаёт сигнал «good» после каждой принятой посылки. Он свидетельствует о корректности посылки и заставляет блок проверки инверсной посылки и трёх последних бит принять и проверить этот байт. После того, как этот блок получит и проверит инверсный байт, он выдаёт сигнал «out_good», свидетельствующий о корректности окончаний и инверсной посылки и заставляющий блок передачи посылок по выходному протоколу записать в ОЗУ очередной байт. На рисунке 8 видно, что сигналов об ошибках во время приёма пакетов не возникло.

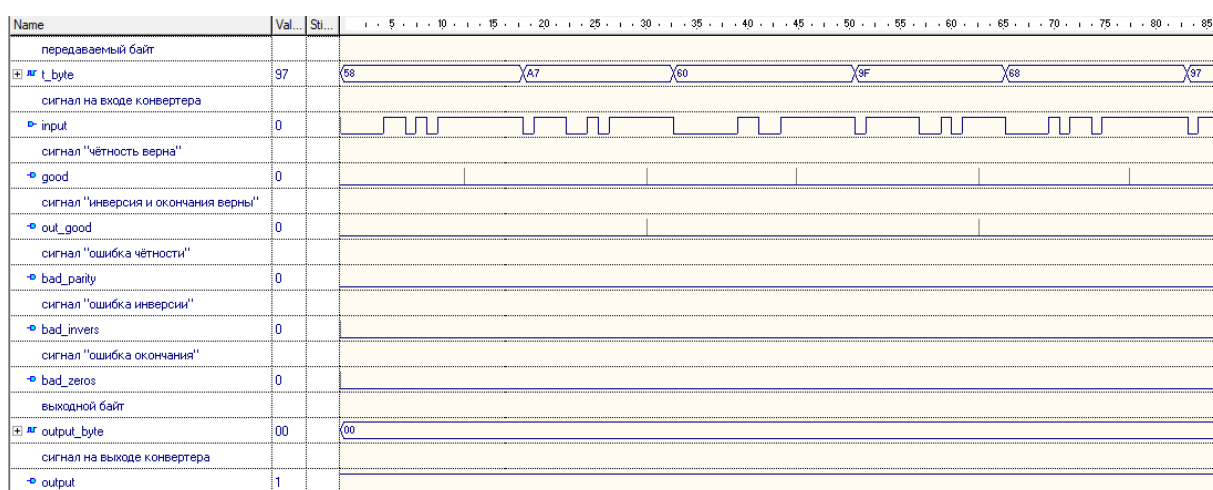


Рисунок 7 – Начало приёма и анализа корректного тестового пакета

После приёма каждого байта блок передачи посылок по выходному протоколу начинает отсчёт времени ожидания «timeout_counter». Если за это время новых данных не приходит, пакет считается принятым и начинается подготовка его к отправке, во время которой пакет в ОЗУ снабжается заголовком, статусом, контрольной суммой и другими данными для соответствия его новому протоколу. Этот момент изображён на рисунке 9.

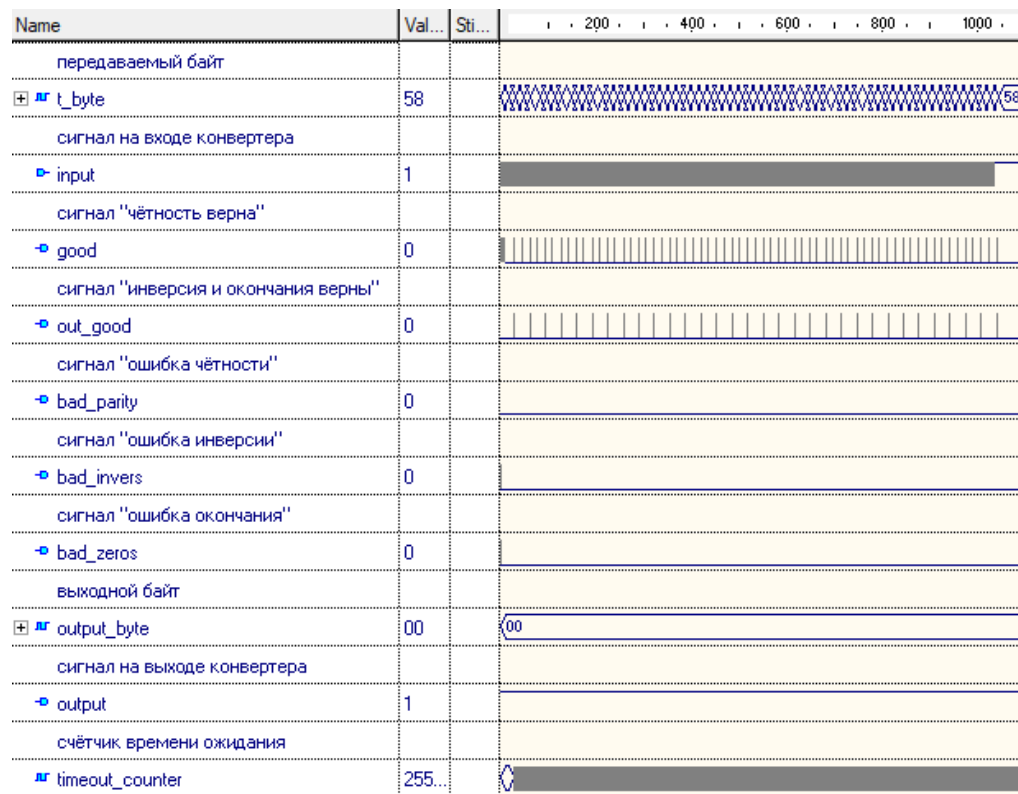


Рисунок 8 – Приём и анализ корректного тестового пакета

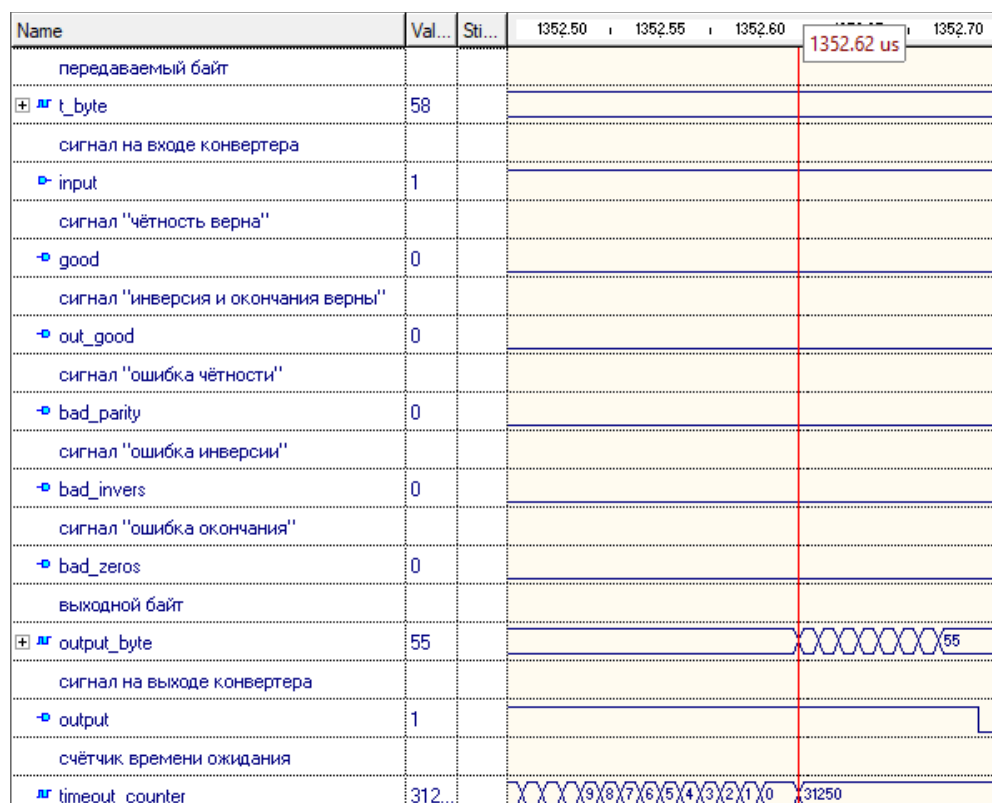


Рисунок 9 – Истечение времени обратного отсчёта

Содержание ОЗУ после приёма всех корректных байт и записи всех необходимых дополнительных данных для соответствия пакета новому протоколу изображено на рисунке 10.

	0	1	2	3	4	5	6	7
00	55	41	00	21	58	60	68	70
08	78	80	88	90	98	A0	A8	B0
10	B8	C0	C8	D0	D8	E0	E8	F0
18	F8	00	08	10	18	20	28	30
20	38	40	48	50	00	80	0F	00
28	00	00	00	00	00	00	00	00

Рисунок 10 – Содержимое ОЗУ

Как видно, все байты переданы корректно, все необходимые данные для нового протокола записаны.

Затем начинается передача данных по новому протоколу. Начало её отражено на рисунке 11. Сигнал «*output*» является выходом конвертера.

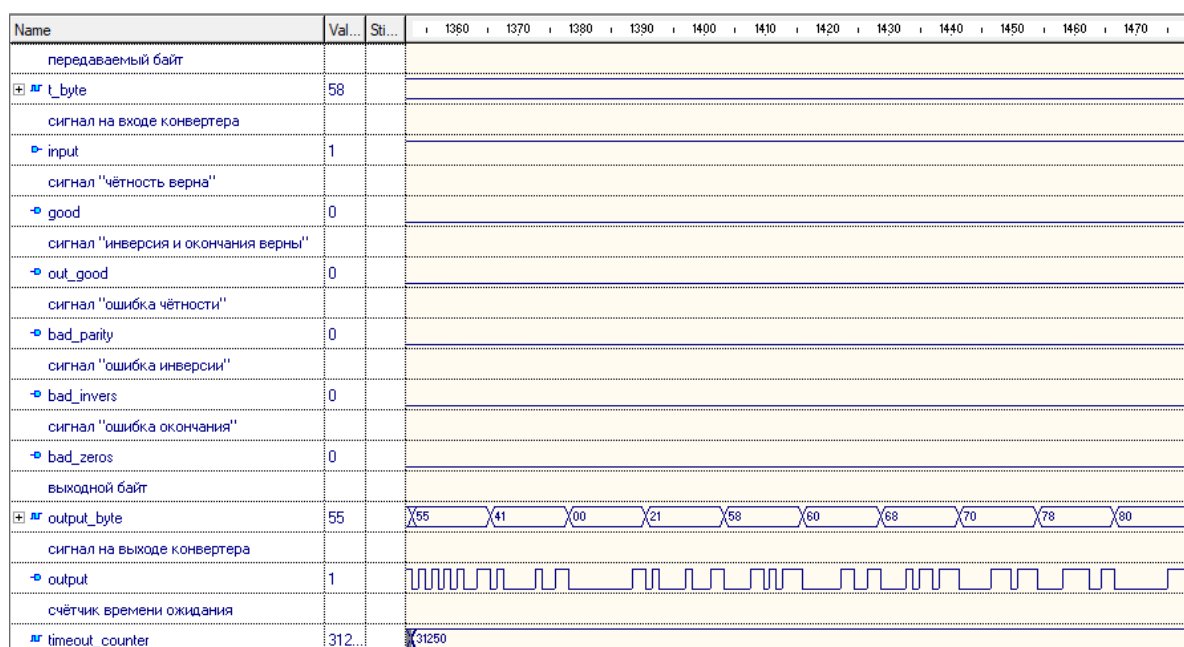


Рисунок 11 – Начало отправки пакета, соответствующего новому протоколу

Отключим инверсию и выполним симуляцию снова. На рисунке 12 видно, что инверсные байты не передаются. Блок приёма посылок и проверки бита чётности выдаёт сигнал «good» после каждой принятой посылки, так как ошибок чётности нет. Блок проверки инверсной посылки после принятия второго байта обнаруживает, что он не является инверсным по отношению к первому. Он выдаёт сигнал об ошибке «bad_invers». Получив этот сигнал, блок передачи посылок по выходному протоколу формирует выходной пакет с кодом ошибки 02. Он изображён на рисунке 13. Затем пакет передаётся по выходному протоколу. Поскольку корректных пар байт не было, пакет данных состоит только из заголовка, кода ошибки инверсии, длины пакета, равной единице, так как она на 1 больше фактической длины, нулевого байта и нулевой контрольной суммы.

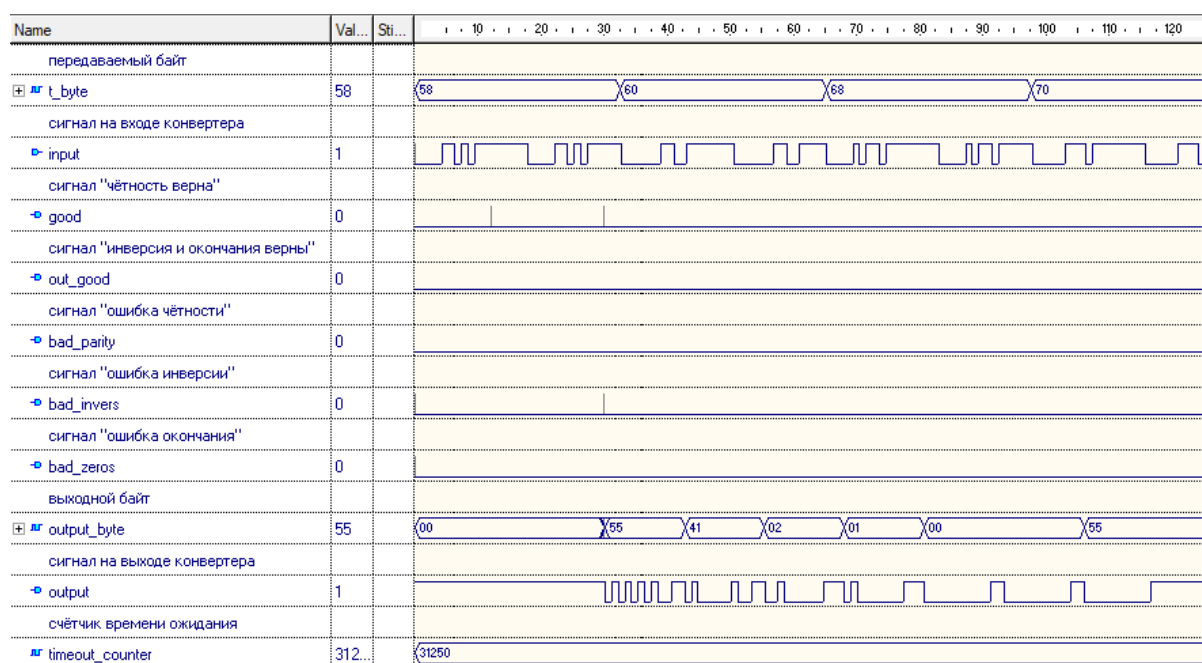


Рисунок 12 – Обнаружение ошибки инверсии

	0	1	2	3	4	5	6	7
00	55	41	02	01	00	00	00	00
08	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00

Рисунок 13 – Содержимое ОЗУ

Включим инверсию и отправим посылку через блок имитации неисправности. На рисунке 14 видно, что блок приёма посылок и проверки бита чётности обнаружил некорректный бит чётности и выдал сигнал об ошибке «*bad_parity*». Получив этот сигнал, блок передачи посылок по выходному протоколу формирует выходной пакет с кодом ошибки 01. Он изображён на рисунке 15. Затем пакет передаётся по выходному протоколу. Поскольку корректных пар байт не было, пакет данных состоит только из заголовка, кода ошибки инверсии, длины пакета, равной единице, так как она на 1 больше фактической длины, нулевого байта и нулевой контрольной суммы.

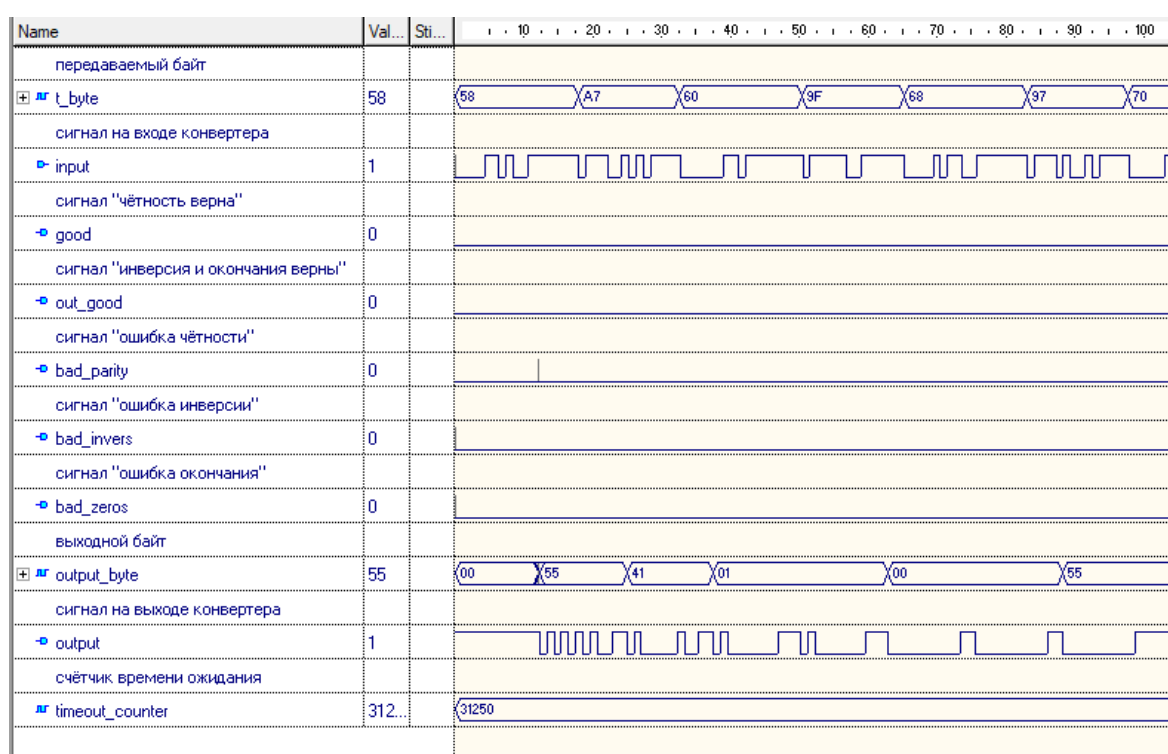


Рисунок 14 – Обнаружение ошибки чётности

	0	1	2	3	4	5	6	7
00	55	41	01	01	00	00	00	00
08	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00

Рисунок 15 – Содержимое ОЗУ

Отключим блок имитации неисправности и внесём в тестовую последовательность некорректное окончание. Пусть байт 5 заканчивается на 110. На рисунке 16 изображён тестовый пакет с внесённой ошибкой.

	0	1	2	3	4	5	6	7
00	01011000	01100000	01101000	01110000	01111000	10000110	10001000	10010000
08	10011000	10100000	10101000	10110000	10111000	11000000	11001000	11010000
10	11011000	11100000	11101000	11110000	11111000	00000000	00001000	00010000
18	00011000	00100000	00101000	00110000	00111000	01000000	01001000	01010000

Рисунок 16 – Тестовый пакет с ошибкой в двоичном виде

Блок проверки инверсной посылки после принятия первого байта из пятой пары обнаруживает, что его окончание некорректно. Он выдаёт сигнал об ошибке «bad_zeros». Это изображено на рисунке 17. Получив этот сигнал, блок передачи посылок по выходному протоколу формирует выходной пакет с кодом ошибки 03. Он изображён на рисунке 18. Затем пакет передаётся по выходному протоколу. В пакете данных видны байты, которые успели передаться корректно.

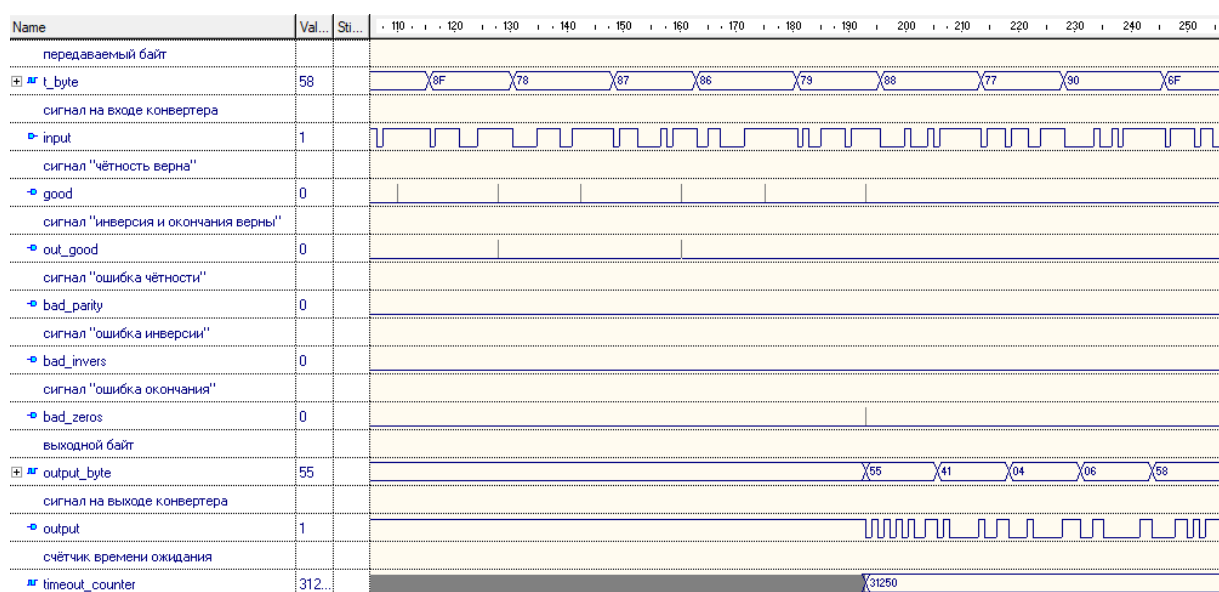


Рисунок 17 – Обнаружение ошибки окончания

	0	1	2	3	4	5	6	7
00	55	41	04	06	58	60	68	70
08	78	00	08	02	00	00	00	00
10	00	00	00	00	00	00	00	00

Рисунок 18 – Содержимое ОЗУ с неверным кодом ошибки

При отладке было обнаружено, что блок передачи посылок по выходному протоколу записывает код ошибки 04 вместо 03. После исправления этой ошибки повторим все вышеперечисленные действия и получим в ОЗУ верный код ошибки, пакет с которым изображён на рисунке 19.

	0	1	2	3	4	5	6	7
00	55	41	03	06	58	60	68	70
08	78	00	08	02	00	00	00	00
10	00	00	00	00	00	00	00	00

Рисунок 19 – Содержимое ОЗУ с верным кодом ошибки

1.9.2 Тестирование на отладочной плате.

На рисунке 20 приведём диаграмму развёртывания отладочного стенда. Отладочная плата подключена к компьютеру с установленными программами для эмуляции приёмника и передатчика, соответствующих обрабатываемым протоколам.

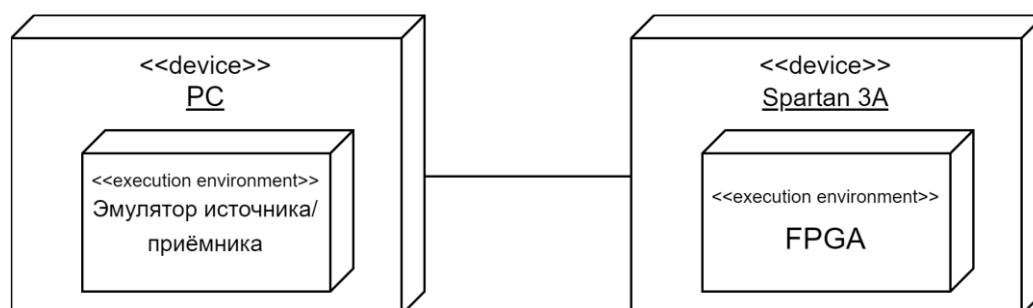


Рисунок 20 – Диаграмма развёртывания отладочного стенда

Перенесём проект в *ISE Xilinx* и сконфигурируем его согласно имеющейся отладочной плате с кристаллом семейства *Spartan 3A*. Конфигурация представлена на рисунке 21.

Property Name	Value
Top-Level Source Type	HDL
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S200A
Package	VQ100
Speed	-4
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

Рисунок 21 – Настройки проекта ISE

Создадим прошивку на основе проекта. Прошивка создавалась без ошибок, что можно увидеть на рисунке 22.

top_lv_sps Project Status (06/16/2023 - 11:09:18)

Property	Value	Errors	Warnings
Project File:	GTP_SPS_int.xise	No Errors	
Module Name:	top_lv_sps	Implementation State:	Programming File Generated
Target Device:	xc3s200a-4vq100	•Errors:	No Errors
Product Version:	ISE 14.7	•Warnings:	706 Warnings (0 new)
Design Goal:	Balanced	•Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	•Timing Constraints:	All Constraints Met
Environment:	System Settings	•Final Timing Score:	0 (Timing Report)

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Total Number Slice Registers	676	3,584	18%	
Number used as Flip Flops	675			
Number used as Latches	1			
Number of 4 input LUTs	885	3,584	24%	
Number of occupied Slices	719	1,792	40%	
Number of Slices containing only related logic	719	719	100%	
Number of Slices containing unrelated logic	0	719	0%	
Total Number of 4 input LUTs	1,013	3,584	28%	
Number used as logic	752			
Number used as a route-thru	128			

Console Log:

```

AM[0]:U_BRAM/ram_rtl_s1_s32_if.ram_rtl_s1_s32_i><RAMB16BWE_RAMB16BWE>.
Process "Generate Programming File" completed successfully
  
```

Рисунок 22 – Прошивка создана успешно

Затем при помощи программы *Xilinx iMPACT*, входящей в состав *ISE*, прошивка была загружена на отладочную плату, что отражено на рисунке 23.

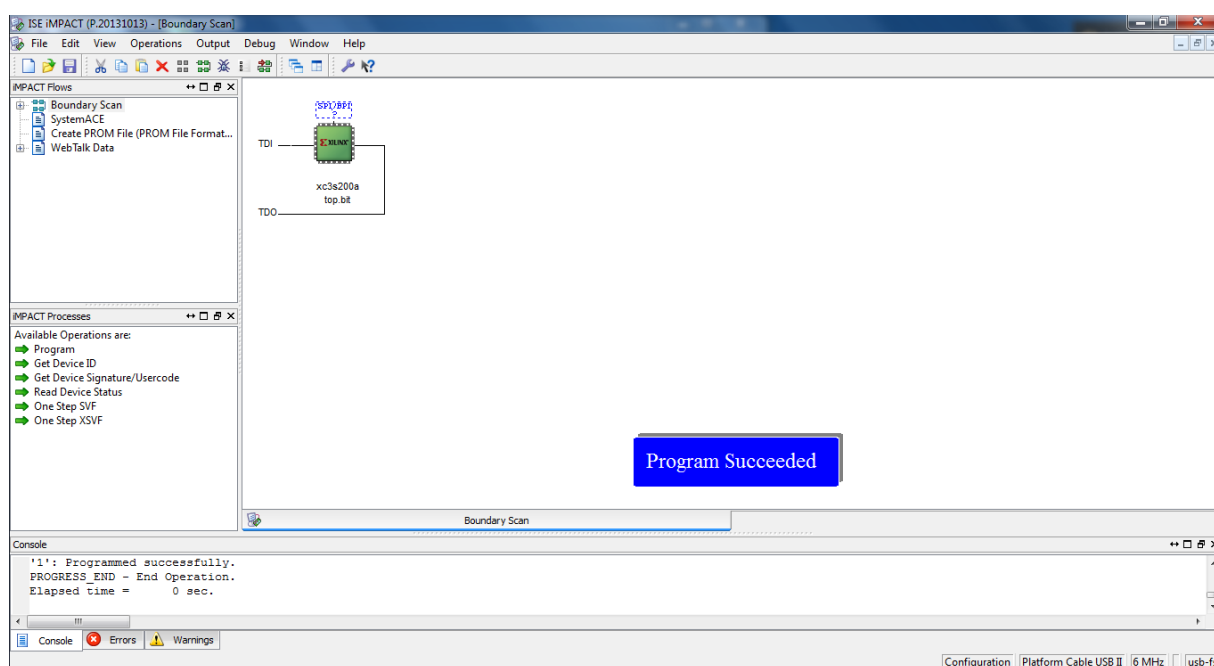


Рисунок 23 – Отладочная плата прошита

По двум последовательным портам отладочная плата была подключена к компьютеру. С помощью двух программ будем эмулировать источник, отдающий данные по старому протоколу, и приёмник, принимающий данные по новому протоколу. Пошлём корректный тестовый пакет, соответствующий старому протоколу, и увидим, что программой-приёмником принимается пакет данных, соответствующий новому протоколу, при этом сами данные переданы верно, инверсные байты отброшены. Это изображено на рисунке 24.

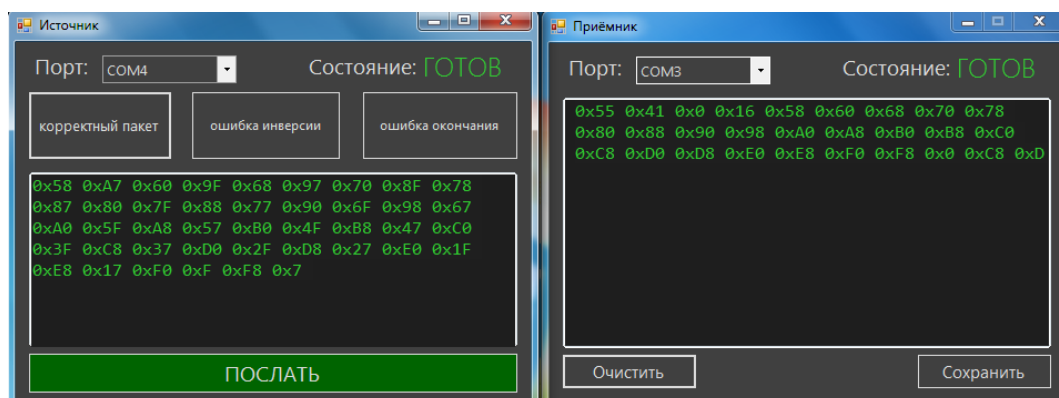


Рисунок 24 – Передача корректного пакета

Теперь передадим пакет с ошибкой в инверсном байте. После 70₁₆ вместо 8F₁₆ внесём 60₁₆. Отправка этого пакета изображена на рисунке 25. Программа – приёмник, обнаружив в пакете код ошибки, распознаёт это и выводит соответствующее сообщение на экран. Затем выводится сам пакет, который можно увидеть на рисунке 26. В нём присутствует код ошибки 2 и байты, успевшие корректно передаться. Для них посчитана контрольная сумма.

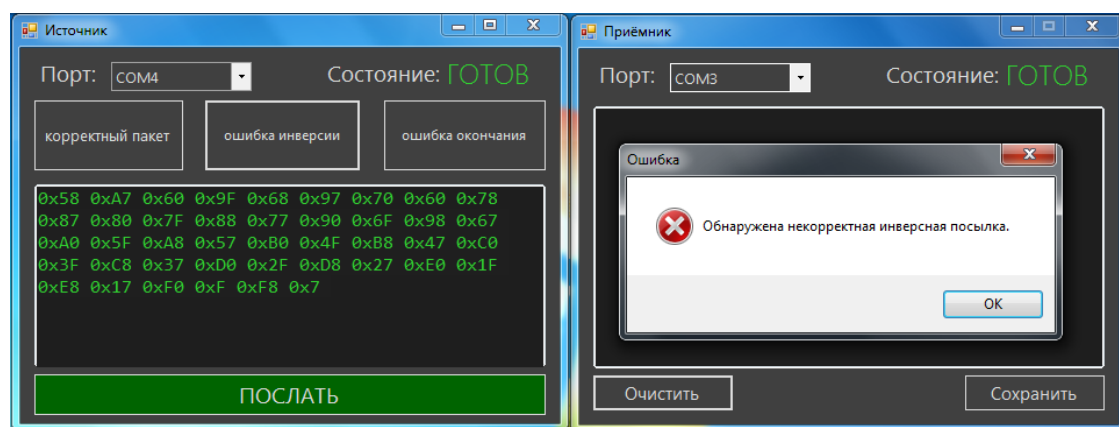


Рисунок 25 – Обнаружение ошибки инверсии

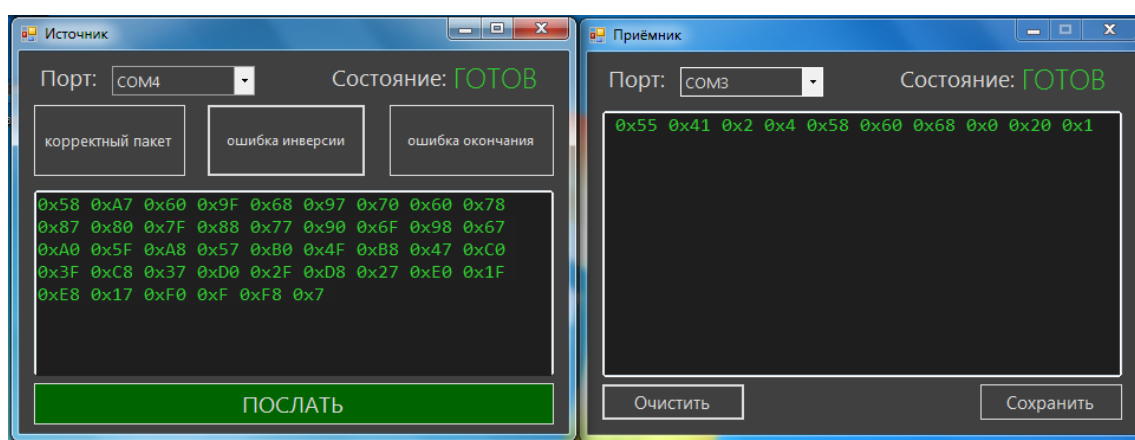


Рисунок 26 – Пакет с ошибкой инверсии

Теперь передадим пакет с ошибкой в окончании. Вместо 70₁₆ внесём 71₁₆. Отправка этого пакета изображена на рисунке 27. Программа – приёмник, обнаружив в пакете код ошибки, распознаёт его и выводит соответствующее сообщение на экран. Затем выводится сам пакет, который можно увидеть на рисунке 28. В нём присутствует код ошибки 3 и байты, успевшие корректно передаться. Для них посчитана контрольная сумма.

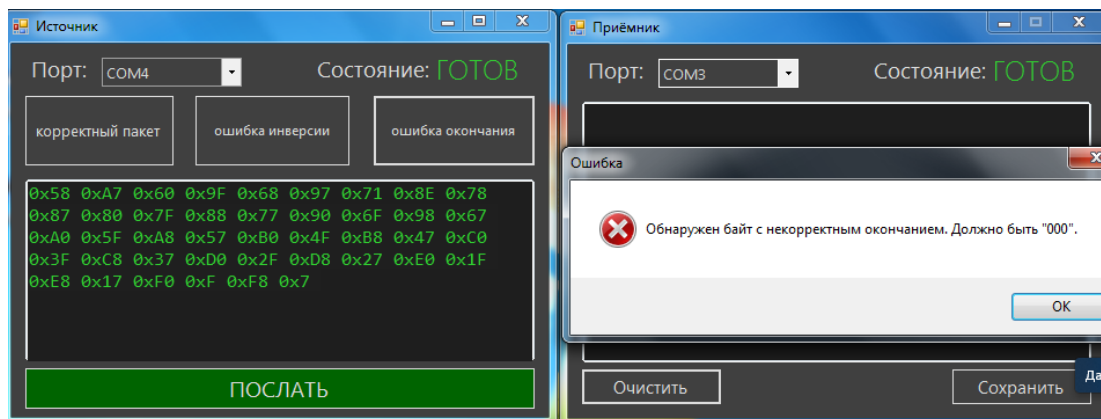


Рисунок 27 – Обнаружение ошибки в окончании

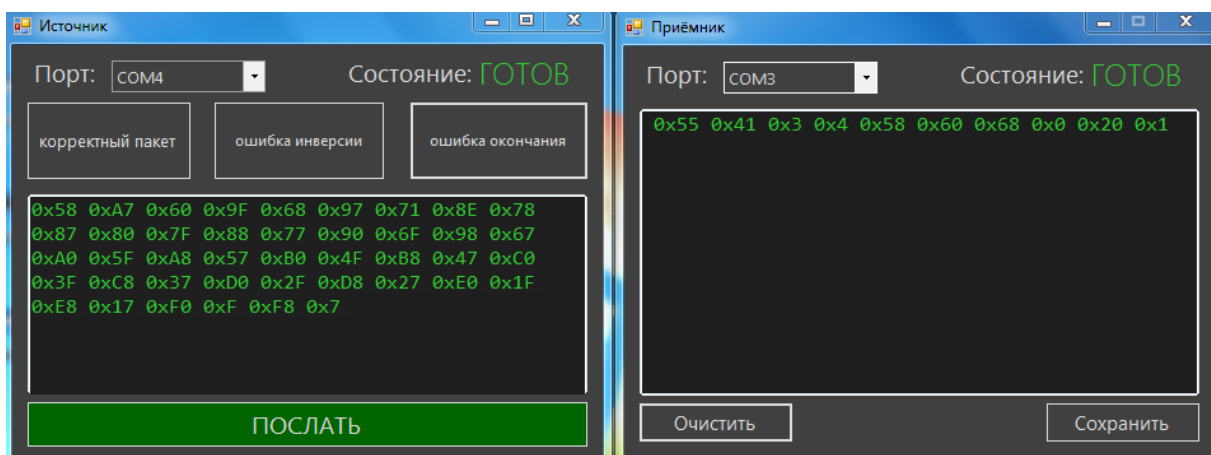


Рисунок 28 – Пакет с ошибкой окончания

Как видим, выходной пакет формируется корректно.

2 Охрана труда

2.1 Анализ негативных факторов, воздействующих на оператора ПЭВМ и способы их устранения

Работа с вычислительной техникой непременно сопровождается ограниченной подвижностью и длительным статическим мышечным напряжением, а также может привести к серьезному перенапряжению зрительных органов. Эти неблагоприятные факторы приводят к переутомлению, снижению работоспособности, психологическому раздражению, появлению физических недомоганий и общей усталости.

Что бы снизить количество вредоносного воздействия средств вычислительной техники на здоровье человека необходимо придерживаться ряда правил использования ПЭВМ. Основными из которых являются:

- правильная организация рабочего места;
- обеспечение достаточной освещенности рабочего места;
- соблюдение правил электробезопасности;
- соблюдение правил пожарной безопасности;
- регуляция режима труда и отдыха.

2.1.1 Оборудование рабочего места

Основные предъявляемые требования к рабочему месту:

- расстояние от глаз до экрана должно быть в пределах 600-700 мм;
- монитор необходимо расположить ниже уровня глаз под углом 10-15 градусов к направлению прямого взгляда;
- высота рабочего стола должна регулироваться в пределах 680-800 мм. В случае отсутствия такой возможности, высота стола должна составлять 725 мм;
- клавиатуру следует размещать на поверхности стола на расстоянии 100-300 мм от края, обращенного к пользователю, или на специальной регулируемой по высоте рабочей поверхности;

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		42

- рабочий стол должен иметь достаточное пространство для ног, включая высоту не менее 600 мм, глубину на уровне колен не менее 450 мм и на уровне вытянутых ног не менее 650 мм.

На рисунке 29 изображена организация рабочего места, которая отвечает указанным требованиям.

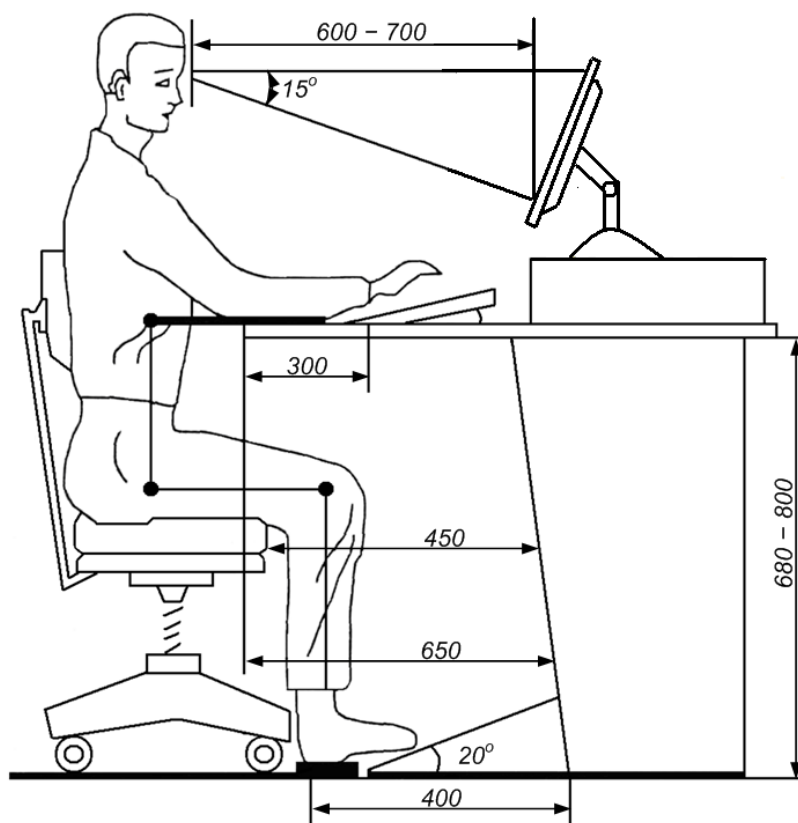


Рисунок 29 – Организация рабочего места

Важно, чтобы рабочее место с дисплеем обеспечивало удобство оператора в положении сидя и не создавало перегрузки костно-мышечной системы.

Организовав правильно рабочее место, необходимо обеспечить правильную посадку пользователя ПК, которая изображена на рисунке 30.

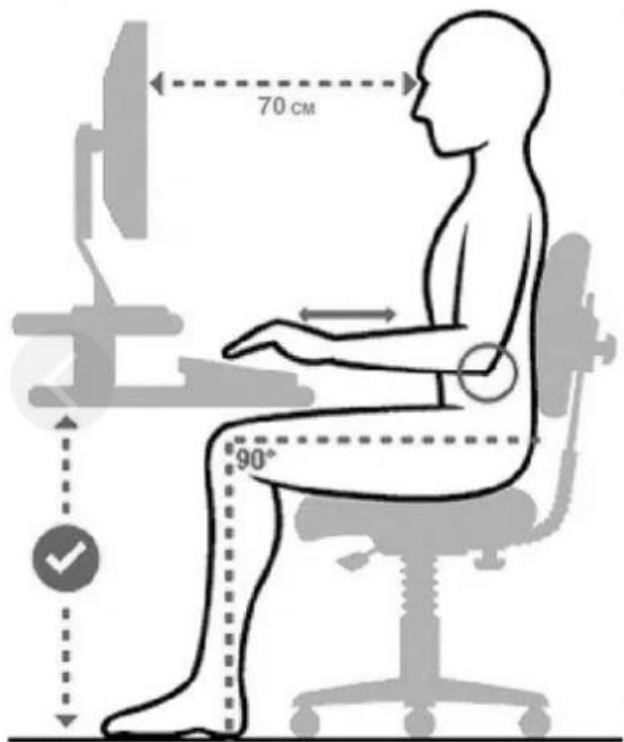


Рисунок 30 – Правильная посадка за ПК

Правильная посадка пользователя ПК обеспечивает выполнение следующих условий:

- запястья должны быть в прямом положении;
- плечи должны быть расслаблены;
- локти должны полностью лежать на столе, перпендикулярно полу (угол в локтевых суставах составляет 90-100 градусов).

2.1.2 Освещенность рабочего места

Стандарты освещенности определяются, учитывая характер работы, место и поверхность. Для пультов ЭВМ и дисплеев рекомендуется комбинированное освещение не менее 400 люкс и общее освещение не менее 200 люкс.

На рабочем месте необходимо обеспечивать равномерную яркость, исключая яркие и блестящие предметы. Для разнообразия в поле зрения рекомендуется использовать отдельные яркие поверхности.

В основном для освещения используются потолочные или встроенные светильники с люминесцентными лампами. Чтобы избежать засветки экранов, светильники общего освещения следует устанавливать сбоку от рабочего места, параллельно линии зрения оператора.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		44

2.1.3 Электробезопасность

Требования к электробезопасности регламентируются ГОСТ 12.1.019-2017 "ЭЛЕКТРОБЕЗОПАСНОСТЬ".

ПК работает от сети переменного тока 50 Гц с напряжением 230 В, что представляет опасность для жизни. При работе на ПК необходимо соблюдать следующие требования:

- все устройства должны быть подключены к одной фазе электрической сети;
- питание устройств ПК должно осуществляться только через розетки типа "Европа" с заземляющими контактами;
- все розетки для подключения устройств ПК должны иметь маркировку по напряжению;
- номинальное напряжение сети должно быть нанесено на стену или щит крупными символами яркой краской (высотой не менее 50 мм) возле розетки.

При использовании вычислительной техники и периферийного оборудования необходимо проявлять осторожность в отношении электропроводки и приборов, помнить, что нарушение правил безопасности может привести к угрозе здоровью и жизни человека.

Необходимо регулярно следить за исправностью электропроводки, выключателей и розеток, используемых для подключения оборудования.

При обнаружении неисправностей немедленно отключите электрооборудование. Продолжение работы возможно только после устранения всех неисправностей.

2.1.4 Пожарная безопасность

Наиболее распространенными причинами пожаров при использовании электроустановок и ПК являются:

- короткое замыкание проводов и электрических устройств;
- воспламенение горючих материалов рядом с электрическими устройствами;
- большие переходные сопротивления в контактных соединениях.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		45

Профилактические меры по пожарной безопасности проводятся в соответствии с федеральным законом от 21.12.1994 №69-ФЗ (ред. от 28.09.2010) "О пожарной безопасности". Для обеспечения пожарной безопасности необходимо принять ряд организационных мероприятий, направленных на обеспечение безопасности людей, предотвращение пожара и его распространения, а также создание условий для эффективного тушения пожаров. Для повышения противопожарной безопасности помещений следует принимать следующие меры:

- проходы и запасные выходы не должны быть загромождены предметами, препятствующими свободному прохождению;
- средства пожаротушения должны быть всегда доступны;
- система вентиляции должна быть оснащена устройствами, автоматически отключающими ее при пожаре, а также огнезадерживающими клапанами;
- автоматическая пожарная сигнализация и средства пожаротушения должны быть в исправном состоянии и соответствовать типовым правилам технического содержания установок пожарной автоматики.

2.1.5 Режим труда и отдыха

Работник, занятый работой с ПК, должен соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности. Различаются следующие виды трудовой деятельности, связанные с использованием персонального компьютера:

- группа А. Работа по чтению информации с экрана видеодисплейного терминала с предварительным запросом;
- группа Б. Работа по вводу информации;
- группа В. Творческая работа в режиме диалога с ПЭВМ.

При выполнении работ, относящихся к различным видам трудовой деятельности, основной вид деятельности с ПК определяется как тот, который занимает не менее 50% времени в течение всей рабочей смены или рабочего дня.

Для каждого вида трудовой деятельности устанавливаются 3 категории тяжести и напряженности работы с ПК:

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		46

- для группы А категория определяется суммарным числом считываемых знаков за рабочую смену (не более 60000 знаков за смену);
- для группы Б категория определяется суммарным числом считываемых или вводимых знаков за рабочую смену (не более 40000 знаков за смену);
- для группы В категория определяется суммарным временем прямой работы с ПК за рабочую смену (не более 6 часов за смену).

Таблица 1 отображает общее время перерывов для каждой категории и вида работы.

Таблица 1 – Время перерывов

Категория работы с ПЭВМ или ВДТ	Уровень нагрузки за рабочую смену при видах работы с ВДТ			Суммарное время регламентированных перерывов, мин	
	группа А, количество знаков	группа Б, количество знаков	группа В час	при 8- часовой смене	при 12- часовой смене
1	До 20000	До 15000	До 2	50	80
2	До 40000	До 30000	До 4	70	110
3	До 60000	До 40000	До 6	90	140

Заключение

В ходе выполнения выпускной квалификационной работы был разработан конвертер специализированных протоколов передачи данных по последовательному порту с использованием языка *VHDL*, как основного языка описания аппаратуры.

Разработанный конвертер позволяет передавать данные между устройствами, поддерживающими различные протоколы передачи данных.

Одним из достоинств разработанного конвертера является быстроедействие, а также корректная работа.

В дальнейшем можно доработать разработанный конвертер, добавив поддержку других протоколов. Добавить интерфейс пользователя для определения необходимых протоколов. Расширить ОЗУ, чтобы увеличить максимальную длину обрабатываемого пакета.

					ПГУ1.090301.23.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Суворова Е.А., Шейнин Ю.Е. Проектирование цифровых систем на *VHDL*. — СПб.: БХВ-Петербург, 2003. — 576 с.: ил.
2. Поляков А.К. Языки *VHDL* и *VERILOG* в проектировании цифровой аппаратуры. — М.: СОЛОН-Пресс, 2003. — 320 с.: ил.
3. Максфилд К. Проектирование на ПЛИС. Курс молодого бойца. — М.: Издательский дом «Додэка-XXI», 2007. — 408 с.: ил.
4. Точки, Рональд, Дж., Уидмер, Нил, С. Цифровые системы. Теория и практика, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2004. — 1024 с.: ил.
5. Дж. Рамбо, М. Блаха *UML 2.0*. Объектно-ориентированное моделирование и разработка, 2-е издание. — СПб.: Питер, 2007 — 544 с.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		49

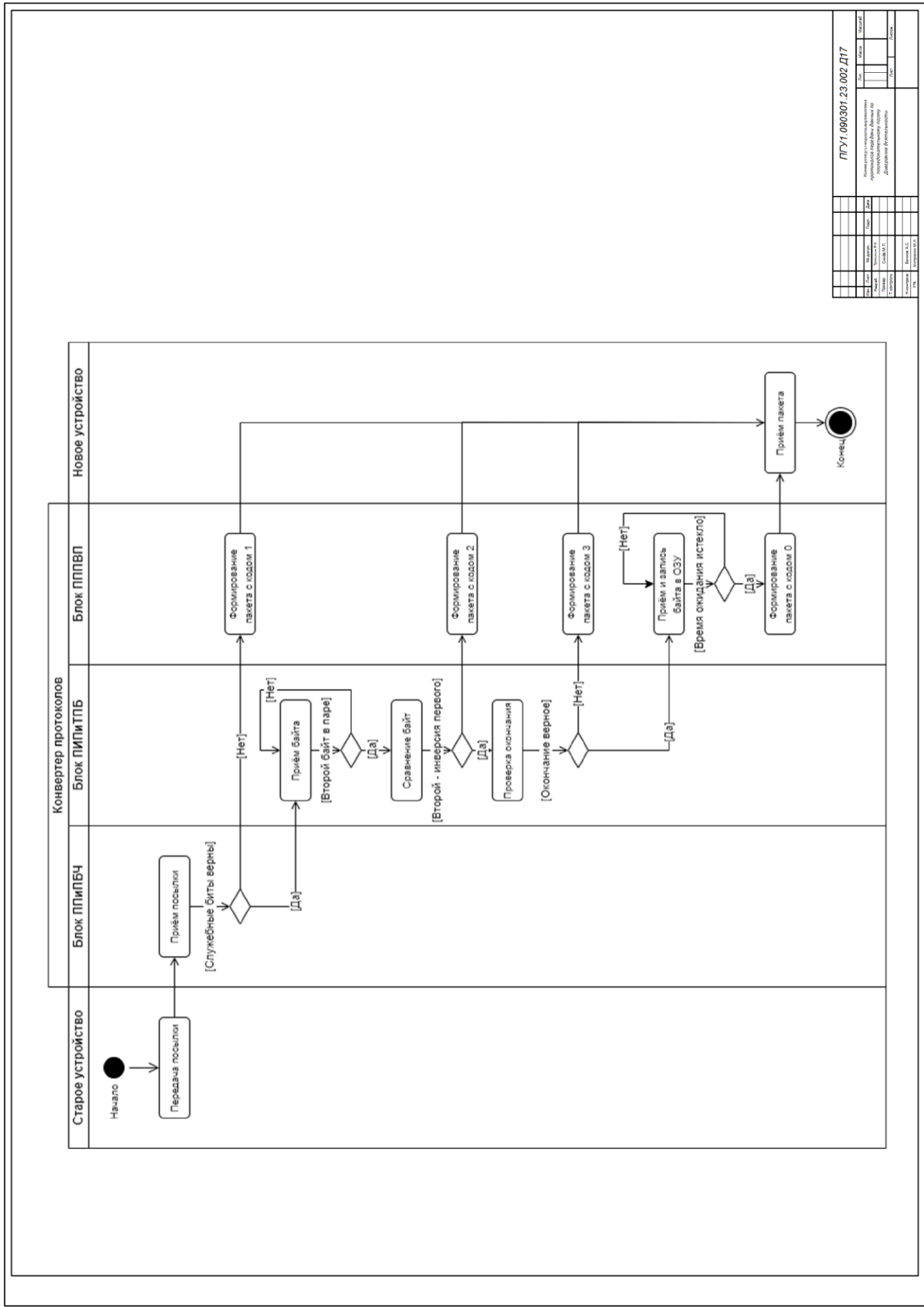
Приложение А

(обязательное)

Конвертер специализированных протоколов передачи данных по
последовательному порту
UML диаграммы

					ПГУ1.090301.23.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А. 1 – UML – диаграмма активности

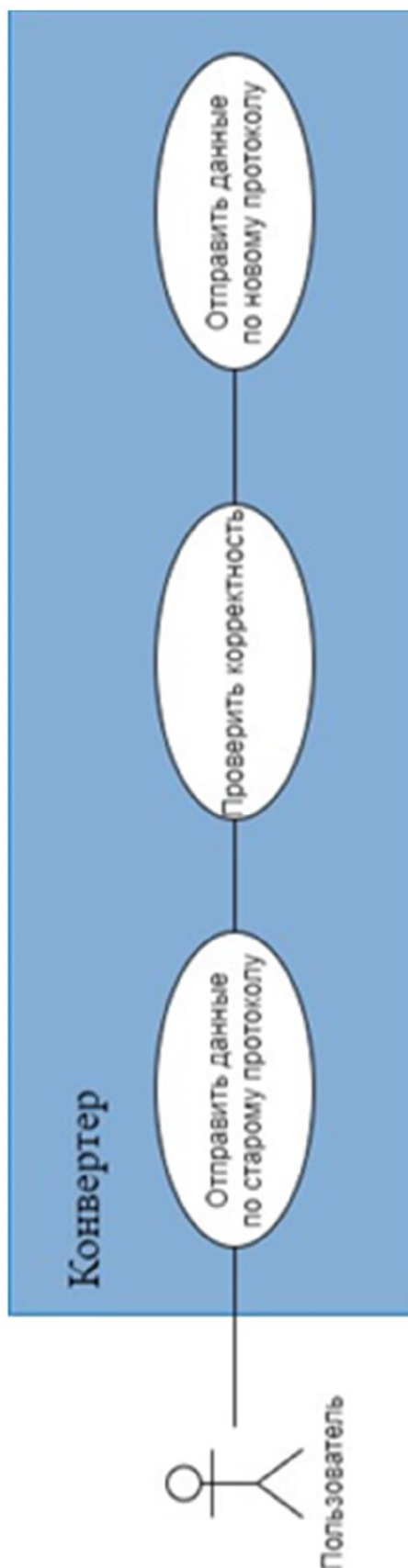


ПГУ1.090301.23.002 Д17									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов
Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов	Степанов

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.23.001 ПЗ

Приложение А.2 – *UML* – диаграмма прецедентов

[illegible]

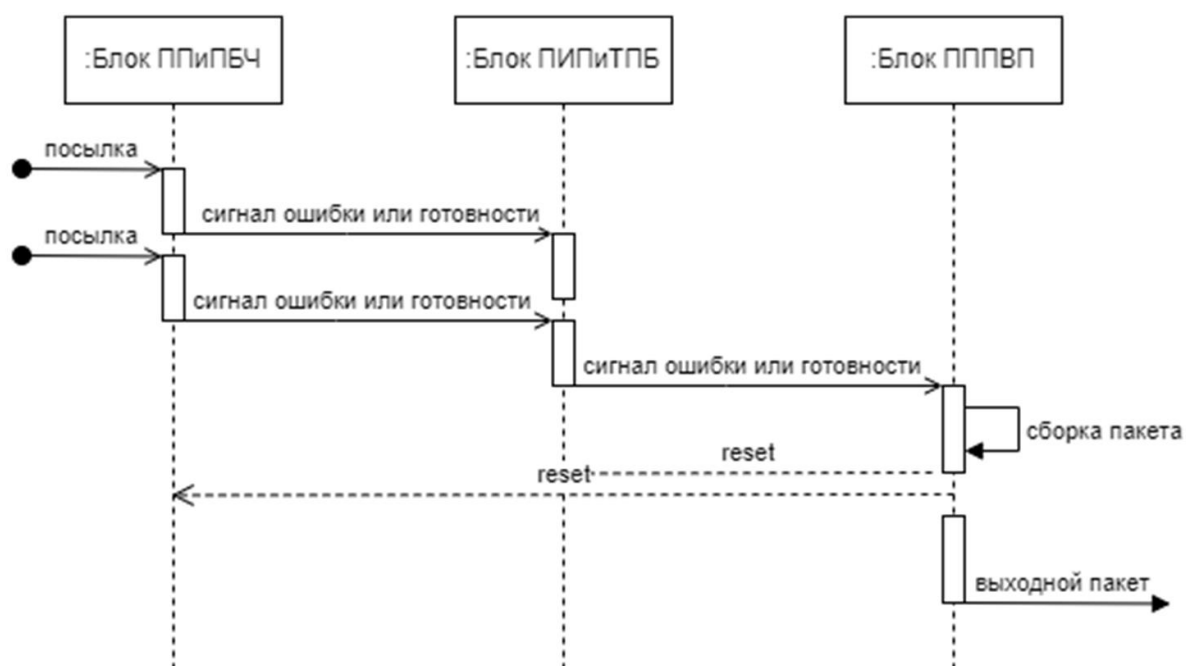
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.23.001 ПЗ

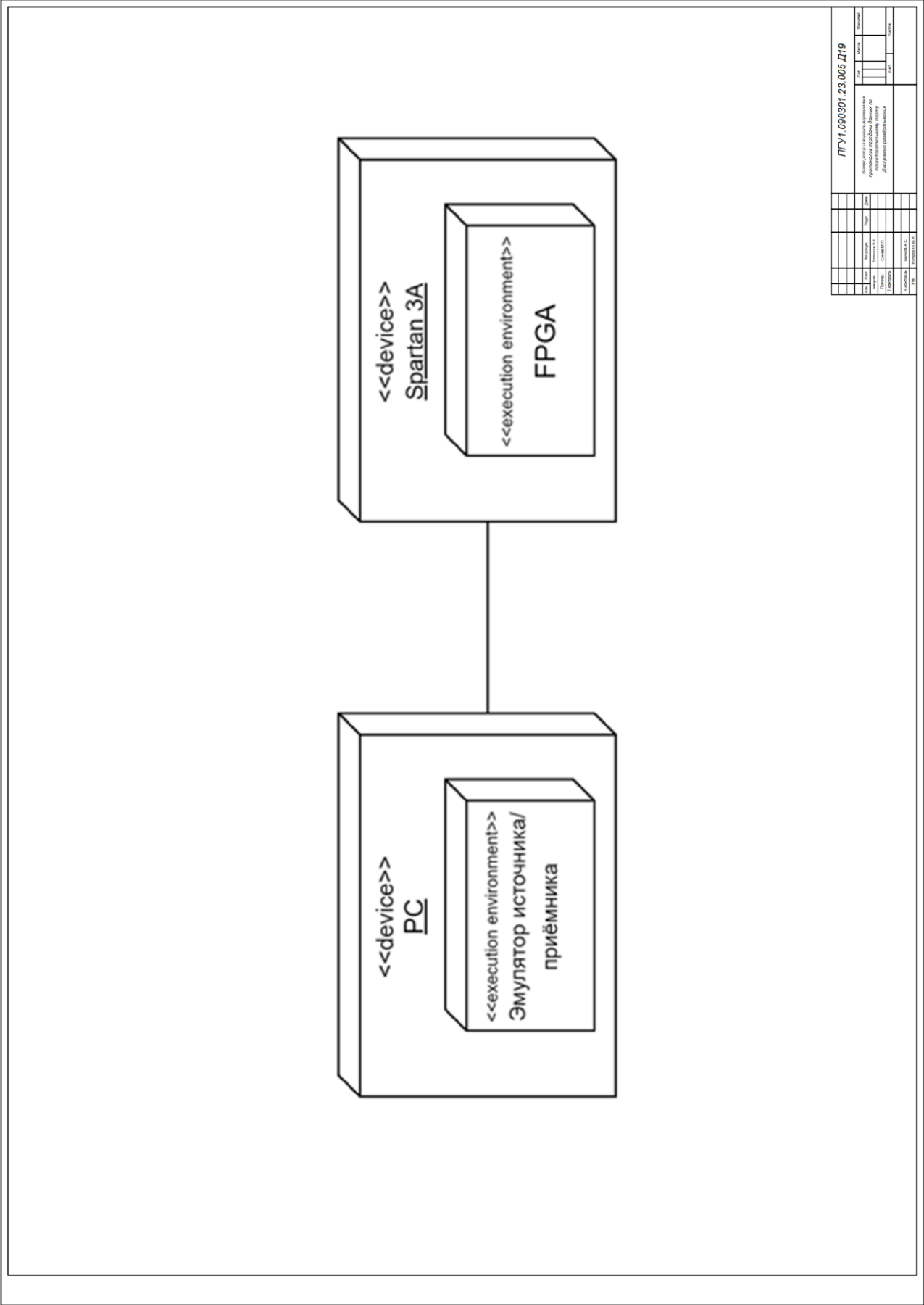
Лист

52

Приложение А.3 – UML – диаграмма последовательности

[illegible]

Приложение А.4 – UML – диаграмма развёртывания



ПГУ1.090301.23.005 Д19									
№	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
1	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
2	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
3	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
4	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
5	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
6	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
7	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
8	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
9	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель
10	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель	Дата	Исполнитель

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.23.001 ПЗ

Приложение Б

(обязательное)

Конвертер специализированных протоколов передачи данных по
последовательному порту
Листинг программы

					ПГУ1.090301.23.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Файл *main.vhd*

```

library ieee;
use ieee.STD_LOGIC_SIGNED.all;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_misc.all;
use IEEE.std_logic_unsigned.all;

library work;

entity fsg_main is
    generic (
        --контроллер входного сигнала
        period : INTEGER := 10417; -- количество тактов на
приём одного бита
        t_period : INTEGER := 10417; --количество тактов на
передачу одного бита
        short_dur: integer := 52083; --пауза между прямой и
инверсной посылками
        long_dur: integer := 208333; --пауза между парами
посылок
        looonger_dur: integer := 10416667; --пауза между
передачами
        DATA_WIDTH : integer := 8; --разрядность шины
данных ОЗУ, в котором собирается посылка
        ADDR_WIDTH : integer := 8; --разрядность шины
адреса ОЗУ, в котором собирается посылка (верни 8)
        TIMEOUT : integer := 3125000; --количество тактов,
в течение которых ожидается байт, если его нет - пакет принят
        BLOCK_AND_RESET_TIMEOUT : integer := 8000000; --
количество тактов, в течение которых блокируется ввод после ошибки
        TRANSMISSION_PERIOD : INTEGER := 10417; --
количество тактов на передачу одного бита
        posylka: std_logic_vector(7 downto 0) := x"6A"    );
    port(
        clk: IN std_logic;
        input: IN std_logic;
        nRESET_IN: IN std_logic;
        output: OUT std_logic
    );
end fsg_main;

architecture MAIN_ARCHITECTURE of fsg_main is
    --обрабатывает сигнал, проверяет его корректность
    component fsg
        generic(
            period : INTEGER);
    port(
        clk : in STD_LOGIC;

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		56

```

        input : in STD_LOGIC;
        reset : in STD_LOGIC;
        tx_empty: IN std_logic;
        output : out STD_LOGIC_VECTOR(7 downto 0);
        good : out STD_LOGIC;
        bad_parity: out std_logic;
        end_strob : out STD_LOGIC );
end component;

--проверяет, совпадают ли прямая и инверсная посылка
component inverse_control is
port (clk: IN std_logic;
      in_good: IN std_logic;
      input: IN std_logic_vector(7 downto 0);
      reset: IN std_logic;
      output: OUT std_logic_vector(7 downto 0);
      bad_invers: OUT std_logic;
      bad_zeros: OUT std_logic;
      out_good: OUT std_logic);
end component;

component packet_transmitter is
--передает данные, полученные по последовательному порту
generic (DATA_WIDTH : integer := 8;
        ADDR_WIDTH : integer := 8;
        TIMEOUT : integer := 3125000;
        BLOCK_AND_RESET_TIMEOUT : integer :=
BLOCK_AND_RESET_TIMEOUT; --количество тактов, в течение которых
блокируется ввод после ошибки
        TRANSMISSION_PERIOD : INTEGER := period);

port (clk: IN std_logic;
      reset: IN std_logic;
      bad_parity : IN std_logic;
      bad_invers : IN std_logic;
      bad_zeros : IN std_logic;
      --ua_status: IN std_logic_vector(7 downto 0);
      WE: IN std_logic;
      input: IN std_logic_vector(DATA_WIDTH - 1 downto
0);

      empty: OUT std_logic;
      block_and_reset: OUT std_logic;
      serial_output: OUT std_logic);
end component;

signal fsg_clk : STD_LOGIC;
signal fsg_reset : STD_LOGIC;
signal data_byte : STD_LOGIC_VECTOR(7 downto 0);
signal inverse_control_output: STD_LOGIC_VECTOR(7 downto 0);
signal good : STD_LOGIC;
signal inverse_control_good : STD_LOGIC;
signal bad_parity : std_logic;
signal bad_invers : std_logic;
signal bad_zeros : std_logic;

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

```

signal status : std_logic_vector(7 downto 0);
signal end_strob : STD_LOGIC;

```

```

signal tx_empty : STD_LOGIC;
signal block_and_reset_main : STD_LOGIC;
signal reset_block: std_logic;

```

```

begin

```

```

    fsg_clk <= clk;

```

```

    fsg_inst : fsg

```

```

    --обрабатывает сигнал, проверяет его корректность

```

```

        generic map (
            period => period
        )

```

```

        port map (
            clk => fsg_clk,
            input => input,
            reset => reset_block,
            tx_empty => tx_empty,
            output => data_byte,
            good => good,
            bad_parity => bad_parity,
            end_strob => end_strob
        );

```

```

    inverse_check_inst: inverse_control

```

```

    --проверяет, совпадают ли прямая и инверсная посылка

```

```

        port map (clk => fsg_clk,
            in_good => good,
            input => data_byte,
            reset => reset_block,
            output => inverse_control_output,
            bad_invers => bad_invers,
            bad_zeros => bad_zeros,
            out_good => inverse_control_good);

```

```

    packet_transmitter_inst: packet_transmitter

```

```

    generic map (DATA_WIDTH => DATA_WIDTH,
        ADDR_WIDTH => ADDR_WIDTH,
        TIMEOUT => TIMEOUT,
        TRANSMISSION_PERIOD => TRANSMISSION_PERIOD)

```

```

    --передает данные, полученные по последовательному порту

```

```

    port map (clk => fsg_clk,
        --ua_status => x"00",
        --ua_status => status,
        reset => fsg_reset,
        bad_parity => bad_parity,
        bad_invers => bad_invers,
        bad_zeros => bad_zeros,

```

					ПГУ1.090301.23.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

```

        WE => inverse_control_good,
        input => inverse_control_output,
        empty => tx_empty,
        block_and_reset => block_and_reset_main,
        serial_output => output);

fsg_reset <= not nRESET_IN;
reset_block <= fsg_reset or block_and_reset_main;

end MAIN_ARCHITECTURE;

```

Файл *fsg.vhd*

```

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_misc.all;

entity fsg is

    generic (period: integer := 10);
    port    (clk: IN std_logic;
             input: IN std_logic;
             reset: IN std_logic;
             tx_empty: IN std_logic;
             output: OUT std_logic_vector(7 downto 0);
             good: OUT std_logic;
             bad_parity: out std_logic;
             end_strob: OUT std_logic
            );

end entity;

architecture beh1 of fsg is

    component start_strob is
    port    (clk: IN std_logic;
             reset: IN std_logic;
             input: IN std_logic;
             output: OUT std_logic);
    end component;

    type state_type is (wait_for_start, count_half_bit,
        read_data_bit, read_parity_bit, read_stop1, read_stop2, error);
    signal state: state_type;

    signal output_start_strob: STD_LOGIC;

    signal next_state_stb: STD_LOGIC;
    signal recieved_byte: std_logic_vector(7 downto 0);

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		59

```

signal parity_bit: std_logic;
signal stop1: std_logic;
signal stop2: std_logic;
signal parity_ok: STD_LOGIC;
--signal packet_good: STD_LOGIC;
signal counter: INTEGER;
signal byte_counter: INTEGER;
signal parity_bit_debug: std_logic;

begin
    start_strob_inst : start_strob
        --выдаёт строб, когда начинается посылка (появляется нулевой
уровень на входе)
    port map (
        clk => clk,
        reset => reset,
        input => input,
        output => output_start_strob
    );

    parity_bit_debug <= XOR_REDUCE(recieved_byte);

    process (clk)
    begin
        if (clk='1' and clk'event) then
            if (reset ='1') then
                state <= wait_for_start; --инициализация сигналов
при сбросе

                byte_counter <= 0;
                output <= "00000000";
                counter <= 0;
                good <= '0';
                parity_ok <= '0';
                bad_parity <= '0';
                parity_bit <= '0';
                stop1 <= '0';
                stop2 <= '0';
                recieved_byte <= x"00";
                next_state_stb <= '0';
            else

                next_state_stb <= '0';

                case state is

                    when wait_for_start =>
                        --состояние ожидания посылки
                        byte_counter <= 0;
                        parity_ok <= '0';
                        parity_bit <= '0';
                        bad_parity <= '0';
                        stop1 <= '0';

                        good <= '0';
                        stop2 <= '0';

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		60

```

recieved_byte <= "00000000";
if (output_start_strob = '1' and tx_empty
= '1') then --если началась новая посылка и старая уже обработана
state <= count_half_bit; --переходим
в состояние отсчёта половины периода приёма одного бита
counter <= period / 2; --
инициализируем счётчик половиной периода
end if;

when count_half_bit =>
--состояние отсчёта половины периода
приёма одного бита
--нужно для того, чтобы не попасть на
фронт сигнала при считывании битов
if (counter = 0) then --по истечении
половины периода проверяем первый бит
if (input /= '0') then --если он не
равен нулю - ошибка
state <= error;
else
state <= read_data_bit; --иначе
переходим в состояние чтения бита данных
next_state_stb <= '1';
counter <= period; --перед
считыванием первого бита данных ждём период
end if;
else
counter <= counter - 1; --обратный
отсчёт половины периода
end if;

when read_data_bit =>
--состояние чтения бита данных
if (counter = 0 and byte_counter /= 7)
then --пока не считали 7 бит
--state <= read_data_bit;
recieved_byte(byte_counter) <=
input; --считываем очередной бит
next_state_stb <= '1';
counter <= period; -- каждый раз
будем ждать период между битами
byte_counter <= byte_counter + 1; --
считаем количество считанных битов
--elsif (counter = 0 and byte_counter =
7) then
elsif (counter = 0) then --если считали 7
бит
recieved_byte(byte_counter) <=
input; --считываем восьмой
state <= read_parity_bit; --и
переходим в состояние чтения бита чётности
next_state_stb <= '1';

counter <= period;
else

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		61

```

counter <= counter - 1; --обратный
отсчёт до приёма следующего бита
end if;

when read_parity_bit =>
    --состояние чтения бита чётности
    --state <= wait_for_start;
    if (counter = 0) then
        parity_bit <= input; --считываем бит
чётности
        state <= read_stop1; --переходим в
состояние считывания первого стопового бита
        next_state_stb <= '1';
        counter <= period;
    else
        counter <= counter - 1;
    end if;

when read_stop1 =>
    --состояние считывания первого стопового
бита
    if (counter = 0) then
        stop1 <= input;
        --
считываем бит чётности
        state <= read_stop2; --переходим
в состояние считывания второго стопового бита
        next_state_stb <= '1';
        counter <= period;
    else
        counter <= counter - 1;
    end if;

when read_stop2 =>
    --состояние считывания второго стопового бита
и проверки четности
    if (counter = 0) then
        stop2 <= input;
        if ((XOR_REDUCE(recieved_byte) =
parity_bit) and (stop1 = '1') and (input = '1')) then --проверка
чётности
            good <= '1';
            output <= recieved_byte; --если
верная, передаём байт дальше
            state <= wait_for_start; --и
переходим в состояние ожидания новой посылки
        else
            output <= "00000000"; --иначе
возвращаем нули и переходим в состояние ошибки
            state <= error;
        end if;
        --parity_ok <= '1' when
(XOR_REDUCE(recieved_byte) = parity_bit) else '0';

        --packet_good <= '1' when (parity_ok
= '1') and (stop1 = '1') and (stop2 = '1') else '0';

```

					ПГУ1.090301.23.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

```

        next_state_stb <= '1';
        counter <= period;
    else
        counter <= counter - 1;
    end if;

    when error =>
        --состояние ошибки
        bad_parity <= '1'; --сигнализируем о
неверной чётности
        state <= wait_for_start;

    when others =>
        state <= wait_for_start;
    end case;
end if;
end if;
end process;

end beh1;

```

Файл *start_strob.vhd*

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity start_strob is
    port (clk: IN std_logic;
          reset: IN std_logic;
          input: IN std_logic;
          output: OUT std_logic);
end start_strob;

--}} End of automatically maintained section

architecture beh of start_strob is
    signal input_d: std_logic := '0';
begin

    process (clk)

    begin

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

```

        if (clk = '1' and clk'event) then
            if (reset = '1') then
                input_d <= '0';
            else
                input_d <= input;
            end if;
        end if;
    end process;

    output <= '1' when ( (input_d = '1') and (input = '0') )
else '0';

end beh;

```

Файл *inverse_control.vhd*

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

--модуль сравнивает, совпадают ли прямой и инверсный байты

entity inverse_control is
    port (clk: IN std_logic;
          in_good: IN std_logic;
          input: IN std_logic_vector(7 downto 0);
          reset: IN std_logic;
          output: OUT std_logic_vector(7 downto 0);
          bad_invers: OUT std_logic;
          bad_zeros: OUT std_logic;
          out_good: OUT std_logic);
end inverse_control;

architecture inverse_control of inverse_control is
    type state_type is (wait_for_input, read_first_packet,
read_second_packet, check_zeros, error);
    signal state: state_type;
    signal first_byte: std_logic_vector(7 downto 0);
    signal second_byte: std_logic_vector(7 downto 0);
    signal byte_ending: std_logic_vector(2 downto 0);
    signal byte_buf: std_logic_vector(7 downto 0) := x"00";
begin

    process (clk)
    begin
        if (clk='1' and clk'event) then

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

```

if (reset = '1') then
    state <= wait_for_input;
    out_good <= '0';
    bad_invers <= '0';
    bad_zeros <= '0';
    first_byte <= "000000000";
    second_byte <= "000000000";
    byte_ending <= "000";
    output <= "000000000";
else

    case state is
    when wait_for_input =>
        out_good <= '0';
        if (in_good = '1') then
            state <= read_first_packet;
            first_byte <= input;
        end if;
    when read_first_packet =>
        if (in_good = '1') then
            state <= read_second_packet;
            second_byte <= input;
        end if;
    when read_second_packet =>
        if (first_byte = not second_byte) then
            --out_good <= '1';
            --output <= first_byte;
            byte_ending <= first_byte(2 downto 0);
            state <= check_zeros;
        else
            bad_invers <= '1';
            state <= error;
            first_byte <= "000000000";
            second_byte <= "000000000";
            output <= "000000000";
            byte_ending <= "000";
        end if;

        when check_zeros =>
            if (byte_ending = "000") then
                out_good <= '1';
                output <= first_byte;
                state <= wait_for_input;
            else
                bad_zeros <= '1';
                state <= error;
                first_byte <= "000000000";
                second_byte <= "000000000";
                output <= "000000000";
                byte_ending <= "000";
            end if;

            when error =>

                bad_invers <= '0';

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		65

```

        bad_zeros <= '0';
        state <= wait_for_input;

        when others =>
            state <= wait_for_input;
        end case;
    end if;
end if;
end process;

end inverse_control;

```

Файл *packet_transmitter.vhd*

```

library ieee;
--use ieee.STD_LOGIC_SIGNED.all;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_misc.all;
use IEEE.std_logic_unsigned.all;

entity packet_transmitter is

    generic (DATA_WIDTH : integer := 8;
            ADDR_WIDTH : integer := 6;
            TIMEOUT : integer := 3000;
            BLOCK_AND_RESET_TIMEOUT : integer := 3000;
            TRANSMISSION_PERIOD : integer := 10);

    port (clk: IN std_logic;
          reset: IN std_logic;
          --ua_status: IN std_logic_vector(7 downto 0);
          bad_parity : IN std_logic;
          bad_invers : IN std_logic;
          bad_zeros : IN std_logic;
          WE: IN std_logic;
          input: IN std_logic_vector(DATA_WIDTH - 1 downto
0);

          empty: OUT std_logic;
          block_and_reset: OUT std_logic;
          serial_output: OUT std_logic);
end packet_transmitter;

--}} End of automatically maintained section

architecture packet_transmitter of packet_transmitter is

    component rams_16 is
        generic (ADDR_WIDTH : integer := 8;
                DATA_WIDTH : integer := 8);

        port(clka : in std_logic;
              clkb : in std_logic;

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		66

```

ena      : in std_logic;
enb      : in std_logic;
wea      : in std_logic;
web      : in std_logic;
addra    : in std_logic_vector(ADDR_WIDTH - 1 downto
0);
addrb    : in std_logic_vector(ADDR_WIDTH - 1 downto
0);
dia      : in std_logic_vector(DATA_WIDTH - 1 downto
0);
dib      : in std_logic_vector(DATA_WIDTH - 1 downto
0);
doa      : out std_logic_vector(DATA_WIDTH - 1 downto
0);
dob      : out std_logic_vector(DATA_WIDTH - 1 downto
0));
end component;

component transmitter_to_pc is

generic (period: integer := TRANSMISSION_PERIOD);

port      (clk: IN std_logic;
            input_byte: IN std_logic_vector(7 downto 0);
            reset: IN std_logic;
            en: IN std_logic;
            tx_empty: OUT std_logic;
            output: OUT std_logic
            );
end component;

signal address_a : STD_LOGIC_VECTOR(ADDR_WIDTH - 1 downto
0);
signal address_b : STD_LOGIC_VECTOR(ADDR_WIDTH - 1 downto
0);
signal timeout_counter : INTEGER RANGE 0 TO TIMEOUT := 0;
signal start_timer : STD_LOGIC := '0';
signal time_is_out : STD_LOGIC := '0';
signal output_byte : std_logic_vector(DATA_WIDTH - 1 downto
0);
signal len_counter : integer := 0;
signal we_b : STD_LOGIC := '0';
signal checksum : std_logic_vector(15 downto 0) := x"0000";
signal input_a_d : std_logic_vector(DATA_WIDTH - 1 downto
0);
signal we_a_d : std_logic;
signal tx_empty : std_logic;
signal tx_en : std_logic;
signal tx_en_d : std_logic;
signal status : std_logic_vector(7 downto 0) := x"00";
signal input_b : std_logic_vector(DATA_WIDTH - 1 downto 0);
type state_type is (idle, recieve, wait_timeout,

form_header2, form_status, form_len, form_zeros,

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

```

form_checksum1, form_checksum2, prep_transmit, reading,
transmit, wait_transmission_end);
    signal state: state_type := idle;
    signal reset_timer : std_logic;

    signal block_and_reset_internal : std_logic := '0';
    signal block_and_reset_timeout_counter : INTEGER RANGE 0 TO
BLOCK_AND_RESET_TIMEOUT := 0;
    signal start_block_and_reset_timer : STD_LOGIC := '0';

begin

    ram_inst : rams_16
    --оперативная память, в которую записывается передача
        generic map (
            ADDR_WIDTH => ADDR_WIDTH,
            DATA_WIDTH => DATA_WIDTH
        )

        port map (
            clka => clk,
            clkb => clk,
            ena => '1',
            enb => '1',
            wea => we, --we_a_d,
            web => we_b,
            addra => address_a,
            addrb => address_b,
            dia => input, --input_a_d,
            dib => input_b,
            doa => open,
            dob => output_byte
        );

    PC_transmitter_inst: transmitter_to_pc
    --осуществляет передачу последовательности в ПЭВМ
    port map (clk => clk,
        input_byte => output_byte,
        reset => reset,
        en => tx_en_d,
        tx_empty => tx_empty,
        output => serial_output
    );

    tx_en_delay: process(clk)
    --задерживает на такт сигнал разрешения передачи
    begin
        if (clk='1' and clk'event) then
            if (reset = '1') then
                tx_en_d <= '0';
            else
                tx_en_d <= tx_en;
            end if;
        end if;
    end process;

```

					ПГУ1.090301.23.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

```

        end if;
    end process;

    timer: process(clk)
        --отсчитывает время, по истечении которого лента считается
        прочитанной и можно передавать в ПЭВМ
    begin
        if (clk='1' and clk'event) then
            if (reset = '1') or (reset_timer = '1') then
                timeout_counter <= TIMEOUT;
            else
                if (start_timer = '1') then
                    timeout_counter <= TIMEOUT;
                elsif (timeout_counter /= 0) then
                    timeout_counter <= timeout_counter - 1;
                end if;
            end if;
        end if;
    end process;

    block_and_reset_timer: process(clk)
    begin
        if (clk='1' and clk'event) then
            if (reset = '1') then
                block_and_reset_timeout_counter <= 0;
                block_and_reset_internal <= '0';
            else
                if (start_block_and_reset_timer = '1') then
                    block_and_reset_timeout_counter <=
BLOCK_AND_RESET_TIMEOUT;

                    elsif (block_and_reset_timeout_counter = 0)
then
                        block_and_reset_internal <= '0';
                    else
                        block_and_reset_internal <= '1';
                        block_and_reset_timeout_counter <=
block_and_reset_timeout_counter - 1;
                    end if;
                end if;
            end if;
        end process;

        block_and_reset <= block_and_reset_internal;

        -- form_status: process (clk)
        -- begin
        -- if (clk='1' and clk'event) then
        --     status(0) <= status(0) or bad_parity;
        --     status(1) <= status(1) or bad_invers;
        --     status(2) <= status(2) or bad_zeros;
        -- end if;
        -- end process;

        packet_processing: process (clk)

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		69

```

begin

    if (clk='1' and clk'event) then

        start_timer <= '0';
        we_b <= '0';
        we_a_d <= '0';
        tx_en <= '0';
        start_block_and_reset_timer <= '0';
        --reset_timer <= '1';

        if (reset ='1') then

            address_a <= "00000100";
            len_counter <= 0;
            status <= x"00";
            address_b <= "00000000";
            state <= idle;

            reset_timer <= '1';

        else

            case state is
                when idle => --ждём начала передачи
                    --address_a <= "00100";
                    if (tx_empty = '1') then --если
последняя передача завершена, выдаём наружу сигнал, что готовы к
приёму новой
                        empty <= '1';
                    end if;
                    len_counter <= 0;
                    checksum <= x"0000";
                    address_b <= "00000000";
                    if (bad_parity = '1') then
                        empty <= '0';
                        status <= x"01";
                        address_b <= "00000";
                        input_b <= x"55";
                        we_b <= '1';
                        reset_timer <= '1';
                        start_block_and_reset_timer <=
'1';

                        state <= form_header2;

                    elsif (bad_invers = '1') then
                        empty <= '0';
                        status <= x"02";
                        address_b <= "00000000";
                        input_b <= x"55";
                        we_b <= '1';
                        reset_timer <= '1';
                        start_block_and_reset_timer <=
'1';

                        state <= form_header2;

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		70

```

elsif (bad_zeros = '1') then
    empty <= '0';
    status <= x"04";
    address_b <= "000000000";
    input_b <= x"55";
    we_b <= '1';
    reset_timer <= '1';
    start_block_and_reset_timer <=
'1';

    state <= form_header2;

elsif (WE = '1') then
    --we_a_d <= '1';
    --input_a_d <= input;
    len_counter <= len_counter + 1;
    address_a <= address_a + 1;
    state <= recieve;
    reset_timer <= '0';
end if;

when recieve =>
    start_timer <= '1';
    --address_a <= address_a + 1;
    checksum <= checksum + input;
    state <= wait_timeout;

when wait_timeout =>
    if (timeout_counter = 0) then
        empty <= '0';
        status <= x"00";
        address_b <= "000000000";
        input_b <= x"55";
        we_b <= '1';
        reset_timer <= '1';
        start_block_and_reset_timer <=
'1';

        state <= form_header2;

    elsif (bad_parity = '1') then
        empty <= '0';
        status <= x"01";
        address_b <= "000000000";
        input_b <= x"55";
        we_b <= '1';
        reset_timer <= '1';
        start_block_and_reset_timer <=
'1';

        state <= form_header2;

    elsif (bad_invers = '1') then
        empty <= '0';
        status <= x"02";
        address_b <= "000000000";
        input_b <= x"55";
        we_b <= '1';

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		71

```

'1';

reset_timer <= '1';
start_block_and_reset_timer <=

state <= form_header2;

elsif (bad_zeros = '1') then
empty <= '0';
status <= x"04";
address_b <= "000000000";
input_b <= x"55";
we_b <= '1';
reset_timer <= '1';
start_block_and_reset_timer <=

'1';

state <= form_header2;

--end if;
elsif (WE = '1') then
--we_a_d <= '1';
--input_a_d <= input;
len_counter <= len_counter + 1;
address_a <= address_a + 1;
state <= recieve;
end if;

when form_header2 =>
empty <= '0';
address_b <= "000000001";
we_b <= '1';
input_b <= x"41";
state <= form_status;

when form_status =>
empty <= '0';
address_b <= "00000010";
input_b <= status;
we_b <= '1';
state <= form_len;

when form_len =>
--status <= x"00";
empty <= '0';
address_b <= "00000011";
input_b <=
conv_std_logic_vector(len_counter + 1, input_b'length);
we_b <= '1';
state <= form_zeros;

when form_zeros =>
empty <= '0';
address_b <=
conv_std_logic_vector(len_counter + 4, address_b'length);

input_b <= x"00";
we_b <= '1';

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

```

state <= form_checksum1;

when form_checksum1 =>
    empty <= '0';
    address_b <=
conv_std_logic_vector(len_counter + 5, address_b'length);
    input_b <= checksum(7 downto 0);
    we_b <= '1';
    state <= form_checksum2;

when form_checksum2 =>
    empty <= '0';
    address_b <=
conv_std_logic_vector(len_counter + 6, address_b'length);
    input_b <= checksum(15 downto 8);
    we_b <= '1';
    state <= prep_transmit;

when prep_transmit =>
    empty <= '0';
    address_b <= "00000000";
    tx_en <= '1';
    state <= reading;

when reading =>
    empty <= '0';
    state <= transmit;

when transmit =>
    empty <= '0';
    --if (address_b =
conv_std_logic_vector(len_counter + 6, address_b'length)) then
        --
        address_a <= "00100";
        reset_timer <= '1';
        state <= idle;
        --
    else
        state <= wait_transmission_end;
    --end if;

when wait_transmission_end =>
    empty <= '0';
    if (address_b =
conv_std_logic_vector(len_counter + 6, address_b'length)) then
        address_a <= "00000100";
        reset_timer <= '1';
        state <= idle;
    elsif (tx_empty = '1') then
        address_b <= address_b + 1;
        tx_en <= '1';
        state <= reading;
    end if;

end case;
end if;

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		73

```

        end if;
    end process;

    -- enter your statements here --

end packet_transmitter;

Файл ram_buf.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity rams_16 is

    generic (ADDR_WIDTH : integer := 8;
            DATA_WIDTH : integer := 8);

    port(clka : in std_logic;
         clkb : in std_logic;
         ena : in std_logic;
         enb : in std_logic;
         wea : in std_logic;
         web : in std_logic;
         addra : in std_logic_vector(ADDR_WIDTH - 1 downto 0);
         addrb : in std_logic_vector(ADDR_WIDTH - 1 downto 0);
         dia : in std_logic_vector(DATA_WIDTH - 1 downto 0);
         dib : in std_logic_vector(DATA_WIDTH - 1 downto 0);
         doa : out std_logic_vector(DATA_WIDTH - 1 downto 0);
         dob : out std_logic_vector(DATA_WIDTH - 1 downto
0));
end rams_16;

architecture syn of rams_16 is
    type ram_type is array (2**ADDR_WIDTH-1 downto 0) of
std_logic_vector(DATA_WIDTH-1 downto 0);
    shared variable RAM : ram_type := (others=>(others=>'0'));
begin

    process (CLKA)
    begin
        if CLKA'event and CLKA = '1' then
            if ENA = '1' then
                if WEA = '1' then
                    RAM(conv_integer(ADDRA)) := DIA;
                end if;
                DOA <= RAM(conv_integer(ADDRA));
            end if;
        end if;
    end process;

    process (CLKB)
    begin

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

```

        if CLKB'event and CLKB = '1' then
            if ENB = '1' then
                if WEB = '1' then
                    RAM(conv_integer(ADDRB)) := DIB;
                end if;
                DOB <= RAM(conv_integer(ADDRB));
            end if;
        end if;
    end process;

end syn;

```

Файл *transmitter_to_pc.vhd*

```

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_misc.all;

entity transmitter_to_pc is

    generic (period: integer := 10);

    port    (clk: IN std_logic;
             input_byte: IN std_logic_vector(7 downto 0);
             reset: IN std_logic;
             en: IN std_logic;
             tx_empty: OUT std_logic;
             output: OUT std_logic
            );
end transmitter_to_pc;

--}} End of automatically maintained section

architecture transmitter_to_pc of transmitter_to_pc is

    signal input_d: std_logic_vector(11 downto 0) := (others =>
'1');
    signal byte_reg: std_logic_vector(7 downto 0) := (others =>
'0');
    signal counter: INTEGER;
    signal packet_counter: INTEGER;
    type state_type is (wait_for_start, read_byte, transmit);
    signal state: state_type;

begin
    process (clk)
    begin
        if (clk='1' and clk'event) then
            if (reset ='1') then

                state <= wait_for_start;
                tx_empty <= '1';

```

					ПГУ1.090301.23.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```

byte_reg <= (others => '0');
counter <= 0;
packet_counter <= 0;
input_d <= x"FFF";
else
    case state is
    when wait_for_start =>
        tx_empty <= '1';
        input_d <= x"FFF";
        counter <= 0;
        packet_counter <= 0;
        if (en = '1') then
            tx_empty <= '0';
            byte_reg <= input_byte;
            state <= read_byte;
        end if;
    when read_byte =>
        counter <= 0;
        packet_counter <= 0;
        input_d <= "11" & (XOR_REDUCE(byte_reg))
& byte_reg & '0';

        counter <= period;
        state <= transmit;

    when transmit =>
        if (counter = 0 and packet_counter /=
11) then
            input_d <= input_d(0) & input_d(11
downto 1) ;

            counter <= period;
            packet_counter <= packet_counter +
1;

        elsif (counter = 0) then
            state <= wait_for_start;
            counter <= period;
        else
            counter <= counter - 1;
        end if;

    when others =>
        counter <= 0;
        packet_counter <= 0;
        state <= wait_for_start;
    end case;
end if;
end if;
end process;

output <= input_d(0);

end transmitter_to_pc;

```

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76

Приложение В

(обязательное)

Конвертер специализированных протоколов передачи данных по
последовательному порту

Презентация

					ПГУ1.090301.23.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

Тема работы: «Конвертер специализированных протоколов передачи данных по последовательному порту»

Автор работы: студент группы 19ВВП1 Трошкин В.К.

Научный руководитель: к.т.н., доцент каф. ВТ ПГУ Синёв М.П.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

Актуальность работы

Практическая значимость этой темы в том, что разработанный в её рамках конвертер позволяет использовать старое оборудование вместе с более современным, что избавляет от финансовых затрат на замену оборудования.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		79

Предмет и объект работы

Предмет работы

- Предметом работы являются технологии проектирования сложных функциональных блоков и устройств при помощи языков описания аппаратуры

Объект работы

- Объектом работы является конвертер специализированных протоколов передачи данных по последовательному порту.

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		80

Цель работы и постановка задачи

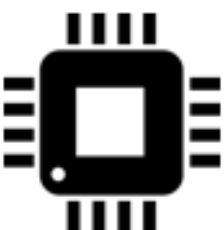
Цель работы

- Целью является разработка конвертера специализированных протоколов передачи данных по последовательному порту.

Постановка задачи

- Требуется разработать устройство, позволяющее передавать данные между устройствами, поддерживающими различные протоколы передачи данных. В нашем случае аппаратурой как старого, так и нового парка используются специализированные, то есть нестандартные протоколы передачи данных.

Стек технологий

• VHDL 

• Xilinx ISE 

• Aldec Active-HDL 

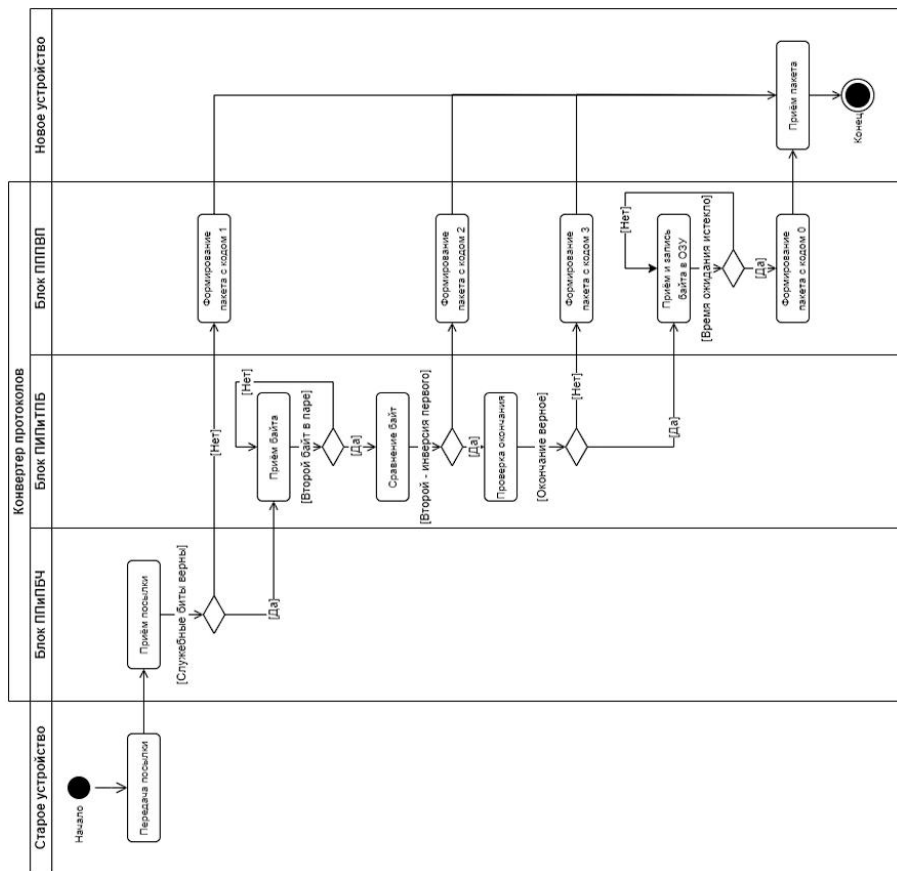
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.23.001 ПЗ

Лист

82

Разработка



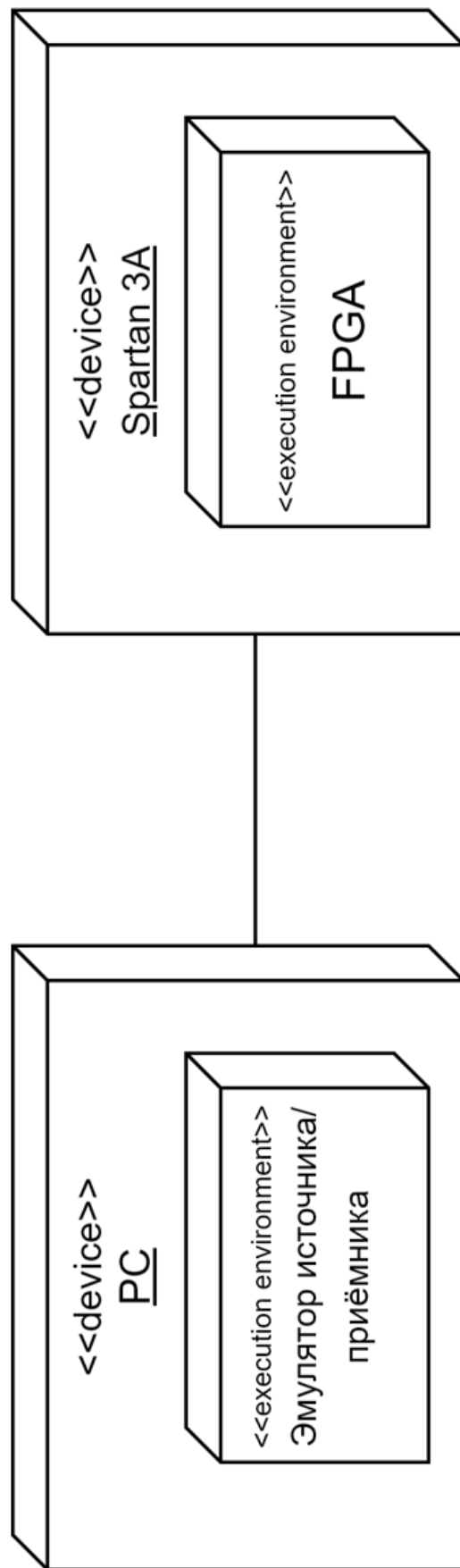
Тестирование и отладка

Name	Val...	Sti...	0	200	400	600	800	1000
передаваемый байт								
input_byte	58							
сигнал на входе конвертера								
input	1							
сигнал "чётность верна"								
good	0							
сигнал "инверсия и окончания верны"								
out_good	0							
сигнал "ошибка чётности"								
bad_parity	0							
сигнал "ошибка инверсии"								
bad_invers	0							
сигнал "ошибка окончания"								
bad_zeros	0							
выходной байт								
output_byte	00							
сигнал на выходе конвертера								
output	1							
счётчик времени ожидания								
timeout_counter	255...							

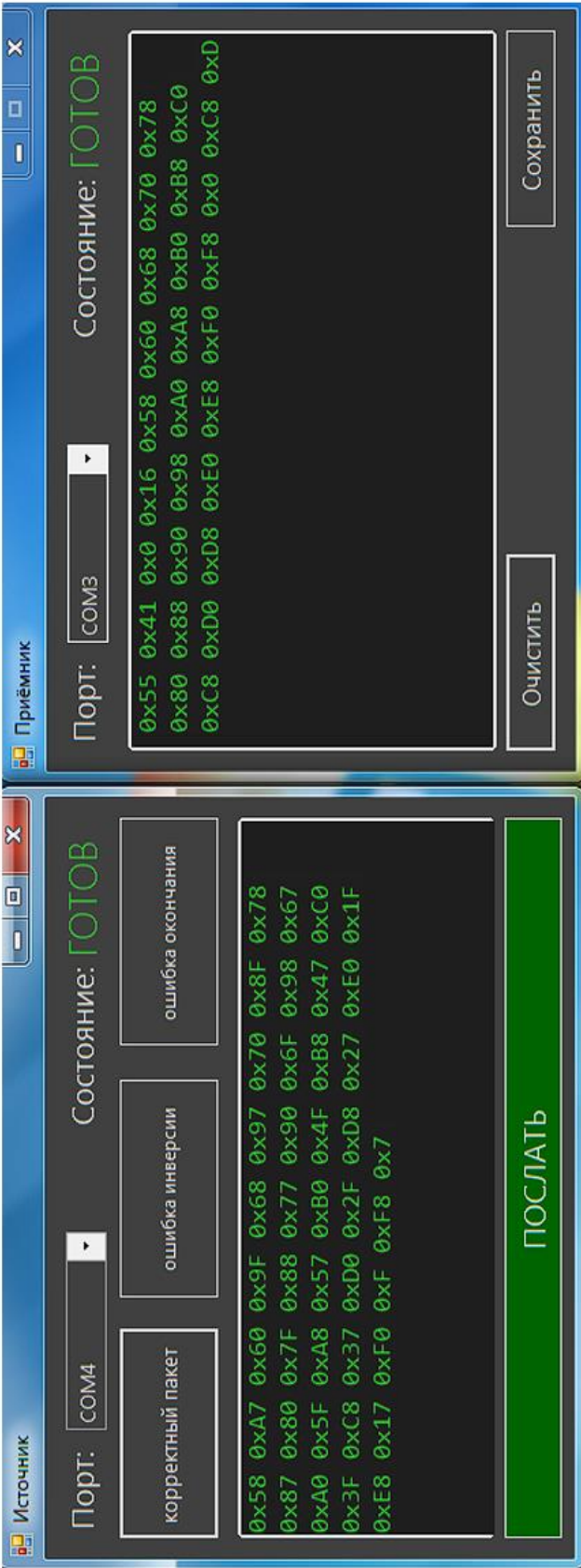
	0	1	2	3	4	5	6	7
00	58	60	68	70	78	80	88	90
08	98	A0	A8	B0	B8	C0	C8	D0
10	D8	E0	E8	F0	F8	00	08	10
18	18	20	28	30	38	40	48	50

	0	1	2	3	4	5	6	7
00	55	41	00	21	58	60	68	70
08	78	80	88	90	98	A0	A8	B0
10	B8	C0	C8	D0	D8	E0	E8	F0
18	F8	00	08	10	18	20	28	30
20	38	40	48	50	00	80	0F	00
28	00	00	00	00	00	00	00	00

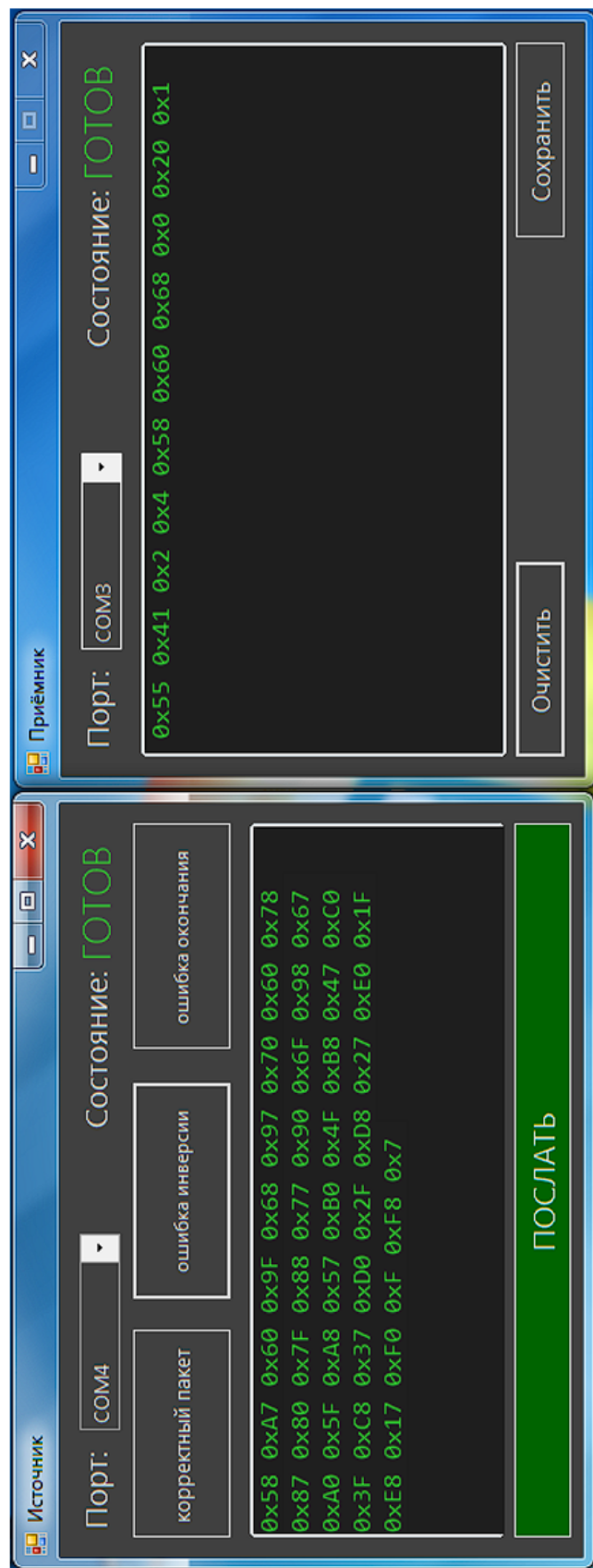
Тестирование и отладка



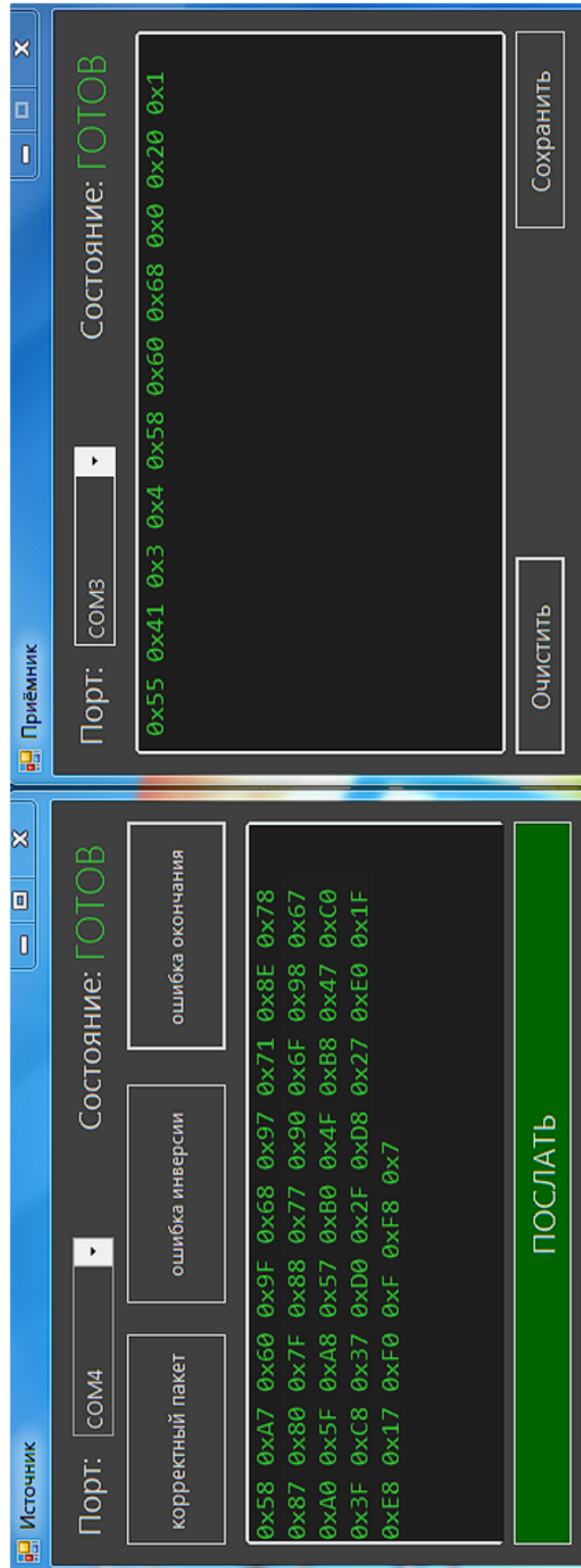
Тестирование и отладка



Тестирование и отладка



Тестирование и отладка



Заключение

- В ходе выполнения выпускной квалификационной работы был разработан конвертер специализированных протоколов передачи данных по последовательному порту с использованием языка VHDL, как основного языка описания аппаратуры.
- Разработанный конвертер позволяет передавать данные между устройствами, поддерживающими различные протоколы передачи данных.
- Одним из достоинств разработанного конвертера является быстроедействие, а также корректная работа.
- В дальнейшем можно доработать разработанный конвертер, добавив поддержку других протоколов. Добавить интерфейс пользователя для определения необходимых протоколов. Расширить ОЗУ, чтобы увеличить максимальную длину обрабатываемого пакета.

Спасибо за внимание!

					ПГУ1.090301.23.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		90

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

_____М.А. Митрохин

« 26 » _____ апреля 20 23 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Трошкин Владимир Константинович гр 19ВВП1 факультета вычислительной техники.
2. Тема работы Конвертер специализированных протоколов передачи данных по последовательному порту
Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.
3. Руководитель бакалаврской работы к.т.н., доцент, каф. ВТ ПГУ Синёв М.П.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать конвертер специализированных протоколов передачи данных по последовательному порту, который позволит подключать источник данных, работающий по старому протоколу, к устройству обработки данных, работающему по более новому протоколу.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие конверторы протоколов, выделить их достоинства и недостатки. Выбрать язык описания аппаратуры, который подойдет для написания логики конвертера. Определить архитектуру конвертера и вспомогательные средства разработки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований	10	01.05.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2023-07.05.2023	
4.	Описание алгоритмов	40	08.05.2023-11.05.2023	
5.	Разработка конвертера специализированных протоколов передачи данных по последовательному порту	150	12.05.2023-29.05.2023	
6.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы _____ Синёв М.П.
(подпись, дата)

Задание к исполнению принял студент _____ Трошкин В.К.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)