МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2024 г.

Разработка драйвера для высокоточных весов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–090301.2024.405 ПЗ ВКР

Консультант,
преподаватель каф. ЭВМ
_____ Н.Д. Топольский
«___»_____2024 г.

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ Д.В. Топольский
«___»_____ 2024 г.

Автор работы
студент группы КЭ-405
_____ Д.В. Романов
«___»_____ 2024 г.

Нормоконтролёр,
ст. преподаватель каф. ЭВМ
_____ С.В. Сяськов
«___»_____2024 г.

Челябинск-2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2023 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Романову Дмитрию Васильевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка драйвера для высокоточных весов».

2. **Срок сдачи студентом законченной работы:** 01 июня 2024 г.

3. **Исходные данные к работе:**

- весы компании T-Scale серии «ROW»;
- весы компании Santint модели ES-7000;
- интегрированная среда разработки «Qt Creator»;
- язык программирования C++;
- декларативный язык программирования QML;
- 1-портовый преобразователь RS-232 в USB MOXA серии UPort 1110.

4. Перечень вопросов, подлежащих разработке:

1. Анализ предметной области и аналитический обзор литературы.
2. Разработка драйвера для высокоточных весов.
3. Разработка графического интерфейса для проверки работы драйвера.
4. Тестирование драйвера.

5. Дата выдачи задания: 01 декабря 2023 г.

Руководитель работы _____ /Д.В. Топольский/

Студент _____ /Д.В. Романов/

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов выпускной квалификационной работы	Срок выполнения этапов работы	Подпись руководителя
Написание введения и обзора литературы, анализ предметной области	29.02.2024	
Написание программного кода для вывода данных с высокоточных весов	01.04.2024	
Разработка графического интерфейса для проверки работоспособности кода при взаимодействии с весами	01.05.2024	
Тестирование работоспособности	15.05.2024	
Компоновка текста работы и сдача на нормоконтроль	23.05.2024	
Подготовка презентации и доклада	30.05.2024	

Заведующий кафедрой _____ /Д.В. Топольский/

Студент _____ /Д.В. Романов/

АННОТАЦИЯ

Д.В. Романов. Разработка драйвера для высокоточных весов. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2024, 85 с., библиографический список – 17 наим.

Данная выпускная квалификационная работа посвящена разработке драйвера для высокоточных весов для его внедрения в программу «StartColor» компании «Два Стахановца».

В первой главе произведён анализ предметной области и обзор литературы. Также в данной главе определён класс точности весов, описан интерфейс RS-232, описана установка: весы, преобразователь и кабель. В ходе выполненной работы мы подключили установку. Также в данной главе описаны используемые средства разработки: язык программирования C++, интегрированная среда разработки Qt Creator с фреймворком Qt Framework и модулем QtSerialPort, а также декларативный язык программирования QML.

Во второй главе описана разработка драйвера для высокоточных весов. Сделано описание и анализ задачи, выбор алгоритмов и путей решения, описана структура будущей программы и выполнено кодирование драйвера.

В третьей главе описан процесс проектировки графического интерфейса, который стал фундаментом написания приложения для весов для первичного тестирования драйвера и весов колористом.

В четвёртой главе описано выполнение тестирования драйвера и корректировка алгоритма. Весы и драйвер были отданы на тест колористу для первичной проверки, которую они прошли.

Успешно разработанный драйвер был внедрён в приложение «StartColor» и теперь компания «Два Стахановца» имеет 3 модели поддерживаемых весов, включая ту, драйвер которой был разработан в рамках данной ВКР.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компания «Два Стахановца» уже 6 лет занимается продажей краски на заказ. Компания делает заказы для заводов, цехов и промышленных комплексов, однако спрос на данную продукцию есть не только у промышленности, но и у специалистов в области колористики, таких как маляры, занимающиеся кузовным ремонтом, так как при починке элемента кузова после ДТП нужно подобрать цвет близкий к тому, что был в момент покраски на производстве. Для выполнения данной задачи используют специальные приборы, способные помочь подбору пигментов, такие как, к примеру, спектрофотометры, колориметры и прочие приборы, способные вычислить спектральные данные поверхности.

Каждый оттенок в цветовой гамме по-своему индивидуален из-за разного цвета, насыщенности и яркости, из-за чего их разнообразие достаточно объёмно. Для получения определённого оттенка, в колористике смешиваются пигменты разных цветов в определённых пропорциях – данная операция называется *подбором*. Пропорции считаются в процентах, а затем конвертируются в единицы измерения массы *компонента* нужного цвета. Чтобы проверить полученный компонент, колорист краскопультом наносит полученную жидкость на пластину, так получается *шаг* «выкраса». Если полученный цвет не удовлетворяет заказчика или колориста, то добавляется ещё один или несколько пигментов (по усмотрению колориста) и полученный новый компонент опять наносится на пластину. При получении нужного цвета на пластине, происходит долив жидкости, согласно указанному объёму при подборе, всё также соблюдая пропорции – шаг при этом становится финальным.

Для точного налива всех пигментов нельзя обойтись без высокоточных весов. Весы, помимо точного налива пигментов, также помогают в учёте потраченной краски при наливе и «выкрасе» пластин.

Помимо всего вышесказанного, компания также занимается установкой рабочих мест, внедрением своего приложения для подбора краски и размещением

сервера на местах заказчиков. Приложение поддерживает работу нескольких весов, однако в связи с текущей ситуацией в стране, рынок измерительных приборов стал более ограниченным, чем ранее, и возникает необходимость находить похожие, либо более выгодные модели, но не уступающие по точности измерений имеющейся аппаратуры для поддержания конкурентоспособности компании. В связи с этим, есть потребность в увеличении числа поддерживаемых измерительных приборов, в том числе и весов.

Написание драйвера для весов будет осуществляться с помощью языка C++ в интегрированной среде разработки «Qt Creator» с использованием встроенного фреймворка «Qt Framework». Работу драйвера необходимо проверить в графическом интерфейсе, он должен быть реализован с помощью декларативного языка программирования QML, являющийся частью Qt, для наглядности передачи данных с весов на компьютер, а также проверки функционала драйвера.

Для подключения весов к компьютеру будет использоваться 1-портовый преобразователь с USB на RS-232 от компании «MOXA» модели UPort 1110.

В данной работе будут описаны все этапы разработки и возникшие проблемы в ходе выполнения данной ВКР.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЗОР ЛИТЕРАТУРЫ

1.1. Общие сведения

В настоящее время высокоточные весы играют важную роль во многих сферах промышленности, научных исследованиях и в технических приложениях. Они используются для точного измерения массы различных материалов и продуктов, что является ключевым элементом контроля качества производства, проведения точных экспериментов и выполнения различных измерений. Для эффективного функционирования электронных высокоточных весов необходимы не только сами весы, но и специализированные программные решения для них.

Существует несколько видов электронных весов, различающиеся по назначению, размерам и диапазону измеряемых величин [1, с. 187-188], среди них есть:

– платформенные;

– напольные;

– медицинские;

– бытовые;

– подвесные;

– крановые;

– автомобильные;

– ювелирные.

Весы могут иметь разный тип преобразователей:

– струнный;

– пьезокварцевой;

– тензометрический.

Каждые весы имеют свой класс точности.

1.2. Определение класса точности весов

Для определения класса точности весов можно использовать документацию OIML – международной организации законодательной метрологии (МОЗМ). У этой организации есть международная рекомендация OIML R 76-1 «Неавтоматические инструменты взвешивания», где описывается стандарт для определения класса точности весов [2]. Данный документ имеет официальное русское издание [3].

Согласно данным документам, есть 4 вида класса точности, каждый из них имеет своё название и обозначение, заключённой в овал. В таблице 1.1 показаны наименования классов точности и их обозначения на весах.

Таблица 1.1 – Классы точности весов

Класс точности	Обозначение, наносимое на весы
Специальный	(I)
Высокий	(II)
Средний	(III)
Обычный	(IIII)

Класс точности весов обычно указывается в документации к этим весам, либо наносится маркировка на этикетке весов.

В документации весов [4] нет информации о классе точности. Рассмотрим информационные этикетки на весах T-Scale (рисунок 1.1) и на весах Santint (рисунок 1.2).

Рисунок 1.1 – Информационная этикетка весов T-Scale

Рисунок 1.2 – Информационная этикетка весов Santint

Как мы видим на рисунке 1.1, на весах T-Scale нет обозначения класса точности, в то время как у весов Santint на рисунке 1.2 обозначение есть. Оно находится справа сверху и, согласно таблице 1.1, имеет следующую классификацию: высокий II.

Также, на обоих рисунках обратим внимание на максимальный допустимый вес (Max 7 kg), минимальный вес (Min 2 g), действительную цену деления (d = 0.1 g), а также на производителя весов: «TSCALE ELECTRONICS MFG (KUNSHAN) Co., Ltd.» – как мы видим, у них одинаковые весовые параметры, а также один и тот же производитель. Исходя из этого, можем предположить, что весы T-Scale имеют тот же класс точности, что и весы Santint.

Таким образом, делаем вывод, что мы имеем высокоточные электронные весы.

1.3. Интерфейс последовательного подключения RS-232

RS-232 представляет из себя 9 контактов (DTE) или 9 отверстий (DCE) и называется DB-9. Существует аналог с 25 контактами DB-25.

Контактный разъём имеет преобразователь, отверстия используются на весах, а кабель имеет оба типа. контакты пронумерованы так, как показано на рисунках 1.3 (а) и 1.3 (б).

а б

Рисунок 1.3 – Разъём DB9: штыревой (DTE) (а), с отверстиями (DCE) (б)

Назначение каждого из разъёмов показано в таблице 1.2.

Таблица 1.2 – Назначения контактов интерфейса RS-232

№	Наименование	Направление	Объяснение
1	**DCD** (Data Carrier Detect)	←	Сигнал используется для указания, что удалённое устройство готово к обмену данными.
2	**RxD** (Receive Data)	←	Прием данных от устройства.
3	**TxD** (Transmit Data)	→	Передача данных на устройство.
4	**DTR** (Data Terminal Ready)	→	Сигнал, указывающий, что локальное устройство готово к передаче данных.
5	**GND** (Signal Ground)	↔	Земля. Каждый сигнал проходит по своему каналу и через землю.
6	**DSR** (Data Set Ready)	←	Сигнал, указывающий, что удаленное устройство готово к передаче данных.
7	**RTS** (Request To Send)	→	Сигнал, указывающий, что локальное устройство готово к передаче данных.
8	**CTS** (Clear To Send)	←	Сигнал, указывающий, что удаленное устройство готово принимать данные.
9	**RI** (Ring Indicator)	←	Сигнал, использующийся для индикации входящего вызова.

1.4. Описание установки

Перед началом работы разберём нашу установку. Установка содержит в себе:

– высокоточные электронные весы;

– преобразователь с RS-232 на USB;

– кабель RS-232 (вилка-розетка).

Опишем все вышеперечисленные компоненты.

1.4.1. Описание внешнего вида высокоточных электронных весов

Для определения типа нашей установки, можем воспользоваться техническим руководством весов T-Scale [4], данное руководство подходит и для весов Santint.

Как видно на рисунке А.1 приложения А, весы являются платформенными, это видно в «Overall View» (с англ. – Общий вид), где показан поддон (на англ. – Pan) для весов, что является ничем иным, как платформой. Также, на рисунке указан порт подключения к весам «RS-232». В категории «Dimensions» (с англ. – Размеры) указаны размеры весов: радиус платформы, высота и длина весов, угол наклона клавиатуры относительно стойки и высота от ножек весов до платформы.

По типу преобразователя весы являются тензометрическими, убедиться в этом поможет 24 элемент на рисунке А.2 приложения А, подписанный в на рисунке А.3 приложения А как «Load Cell» (с англ. – Тензодатчик). Тензодатчик деформируется усилием взвешиваемой массы из-за упругого металлического корпуса и цепей тензорезисторов, таким образом можно узнать вес взвешиваемого предмета на платформе [5, с. 166].

Наверху у весов есть дисплей, который показан на рисунке 1.4.

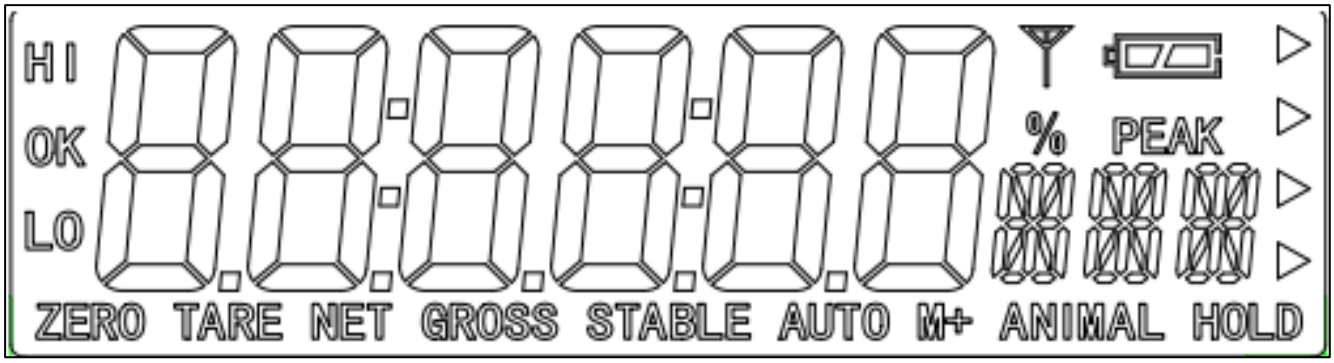

Рисунок 1.4 – Дисплей на весах T-Scale

Каждое слово на рисунке 1.4 – это индикация. Индикации описаны в техническом руководстве [4, с. 8] в виде таблицы на английском языке, перевод значений индикации представлен в таблице А.1 приложения А.

Под дисплеем есть шесть кнопок. Расположение и название кнопок показаны на рисунке 1.5.

Рисунок 1.5 – Расположение кнопок на весах T-Scale «ROW Series»

Каждая кнопка имеет свой собственный функционал, значение которого приведено в таблице А.2 приложения А.

1.4.2. Настройки вывода весов

Чтобы войти в настройки, необходимо во время взвешивания одновременно нажать на кнопки UNIT и M+. Из всех настроек нас будет интересовать только одна – передача данных. Найти эту настройку можно, перейдя в пункт меню «**F4 Prt**», где будет несколько данных настроек. Некоторые настройки описаны в таблице 1.3.

Таблица 1.3 – Режимы передачи данных на весах производства T-Scale

Режим	Назначение	Подходит	Объяснение
P Prt	Вывод значения при нажатии кнопки M+	×	Команда выводит значение только по при нажатии кнопки на весах.
P Cont	Беспрерывная передача данных	✓	Весы будут бесконечно транслировать набор байт.
ASK	Передача в виде «запрос-ответ»	✓	Передача происходит исключительно по командам.
P Stab	Передача только при стабильном весе	×	Передача веса происходит только если горит индикация «STABLE».
P Auto	Автоматический режим	×	Автоматически передаёт значение, если вес стабилен и отличен от предыдущего.

Исходя из этих настроек, можно сделать вывод, что из всех нам понадобился либо беспрерывная передача данных, либо передача данных в виде «запрос-ответ». Это связано с тем, что при запуске программы весы должны сразу выдавать вес, а любые действия будут организованы с помощью кнопок в графическом интерфейсе. Остальные три вида передачи нам не подходят, так как данные с весов не приходят в режиме реального времени.

1.4.3. Массив байтов, выводимый весами

Массив байт имеет следующий вид, показанный на рисунке 1.6.

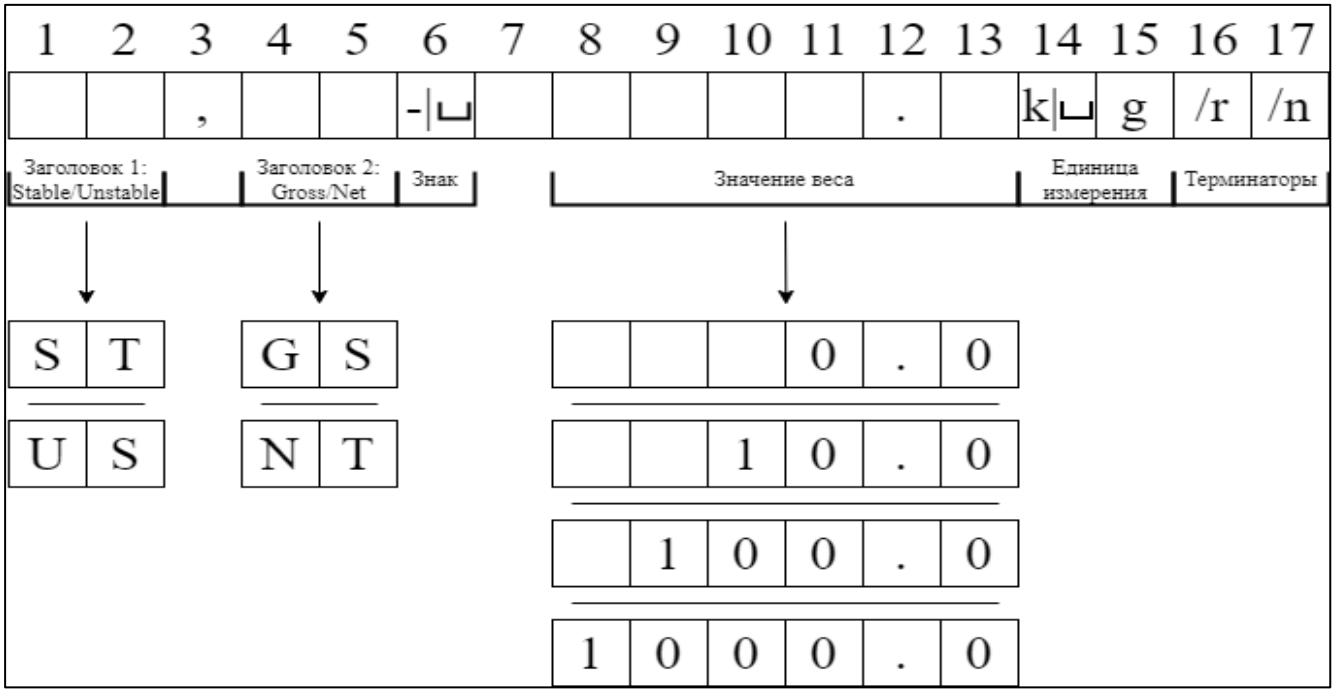

Рисунок 1.6 – Вывод массива байтов с весов

В таблице 1.4 объясняется содержимое рисунка 1.6.

Таблица 1.4 – Содержимое рисунка 1.6

Индекс	Название	Пояснение
1-2	Заголовок 1	Содержит 2 байта: **ST** или **US**, которые означают, что вес стабилен или нестабилен соответственно.
3	Запятая	Разделитель между двумя заголовками.
4-5	Заголовок 2	Содержит 2 байта: **GS** или **NT**, которые означают, что сейчас взвешивание происходит либо с тарой, либо без неё соответственно.
6	Знак	Вывод знака (если знак положительный, то байт остаётся пустым), иначе выводится минус.
7-13	Значение веса	7 байт всегда пуст из-за максимального значения веса в 7000 грамм. Байты с 8 по 11 являются целой частью, байт под номером 12 – разделитель (точка), а байт номер 13 содержит дробную часть веса.
14-15	Единица измерения	Вес может иметь единицу измерения грамм (**g**) или килограмм (**kg**). Если вес в граммах, то вместо «k» будет пустой байт.
16-17	Терминаторы	Байты, указывающие на конец массива.

1.4.4. Команды, принимаемые весами

Весы способны принимать следующие команды:

1. **R (read)** – чтение данных с весов;

2. **Z (zero)** – обнуление;

3. **T (tare)** – смена веса с брутто на нетто.

Данные команды работают в обоих выбранных нами режимах. Команды «**Z**» и «**T**» могут быть отправлены, только если вес стабилен.

1.4.5. Описание преобразователя с COM-порта на USB

Преобразователь позволяет получать данные из RS-232 интерфейса в универсальный USB интерфейс.

Перед использованием нужно установить драйвер, который можно скачать с сайта производителя [6]. Как заверяет сама компания MOXA, их устройство никак не меняет данные, отправляет байты в их исходном виде.

На адаптере есть три индикации, которые подписаны как «**Active**», «**TxD**» и «**RxD**» (рисунок 1.7).

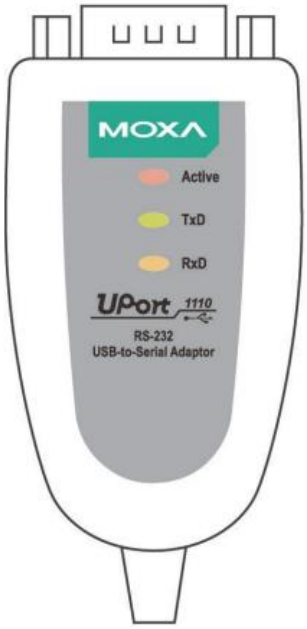

Рисунок 1.7 – Внешний вид адаптера MOXA UPort1110

В руководстве пользователя адаптера [7, с. 5] описаны значения индикаций на английском языке. В приложении Б представлено описание индикаций.

1.4.6. Описание кабеля подключения RS-232

Кабели бывают двух типов, каждый из которых имеет свой тип соединения. Соединения бывают *прямыми* и *перекрёстными*. Кабель с перекрёстным соединением называется *нуль-модемным* кабелем, в то время как кабель с прямым соединением называется *прямым последовательным* кабелем.

Прямое соединение предусматривает наличие модема, который будет менять направление соединителей, либо, на одном из устройств должны быть заранее инвертированы контакты RxD с TxD, DTR с DSR и RTS с CTS, как показано на рисунке 1.8. Иначе, чтение и запись производиться не будет, так как, к примеру, контакт RxD на исходном устройстве будет подключен к аналогичному контакту RxD на подсоединяемом устройстве, в таком случае возникнет ситуация, что контакт на чтение будет подключён к чтению другого устройства и никакая запись производиться не будет, собственно, как и чтение.

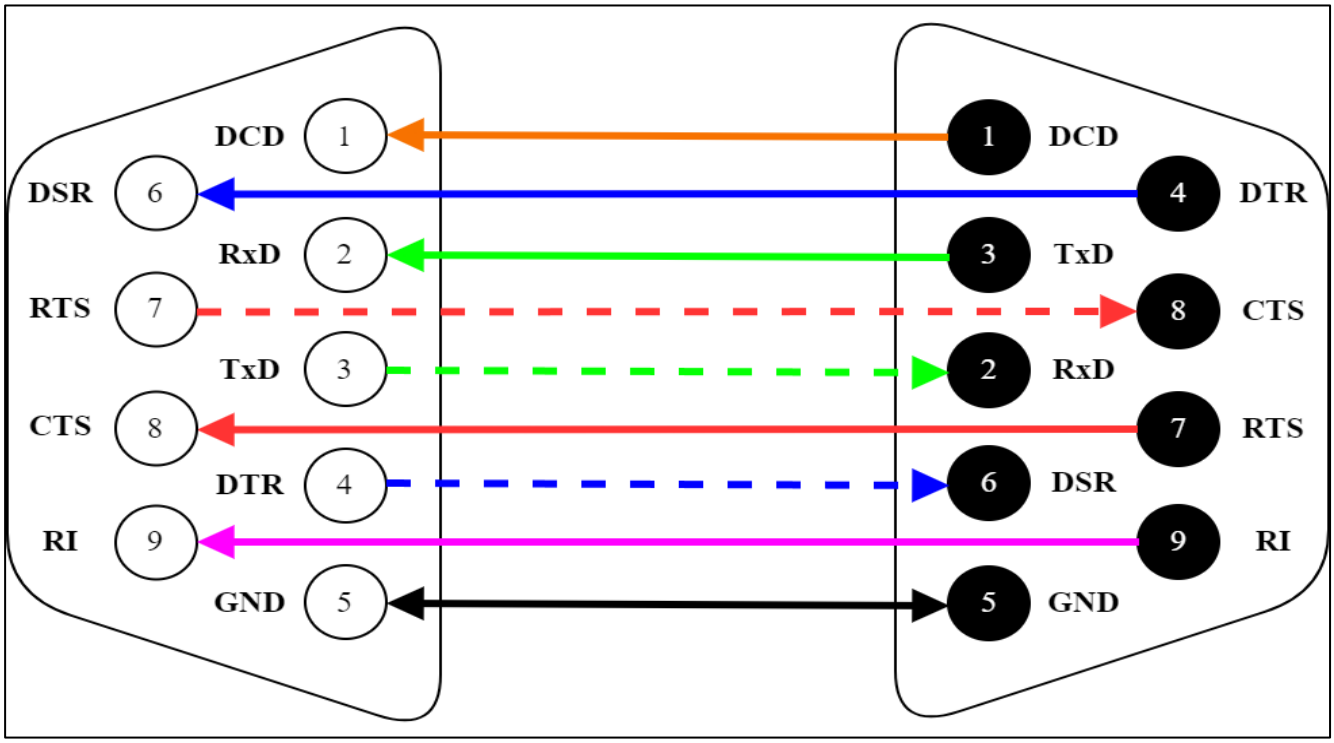

Рисунок 1.8 – Соединение с прямым последовательным кабелем

Нуль-модемный кабель предусматривает работу без модема, так как в нём контакты заранее инвертированы (подключены перекрёстно). Данный тип показан на рисунке 1.9.

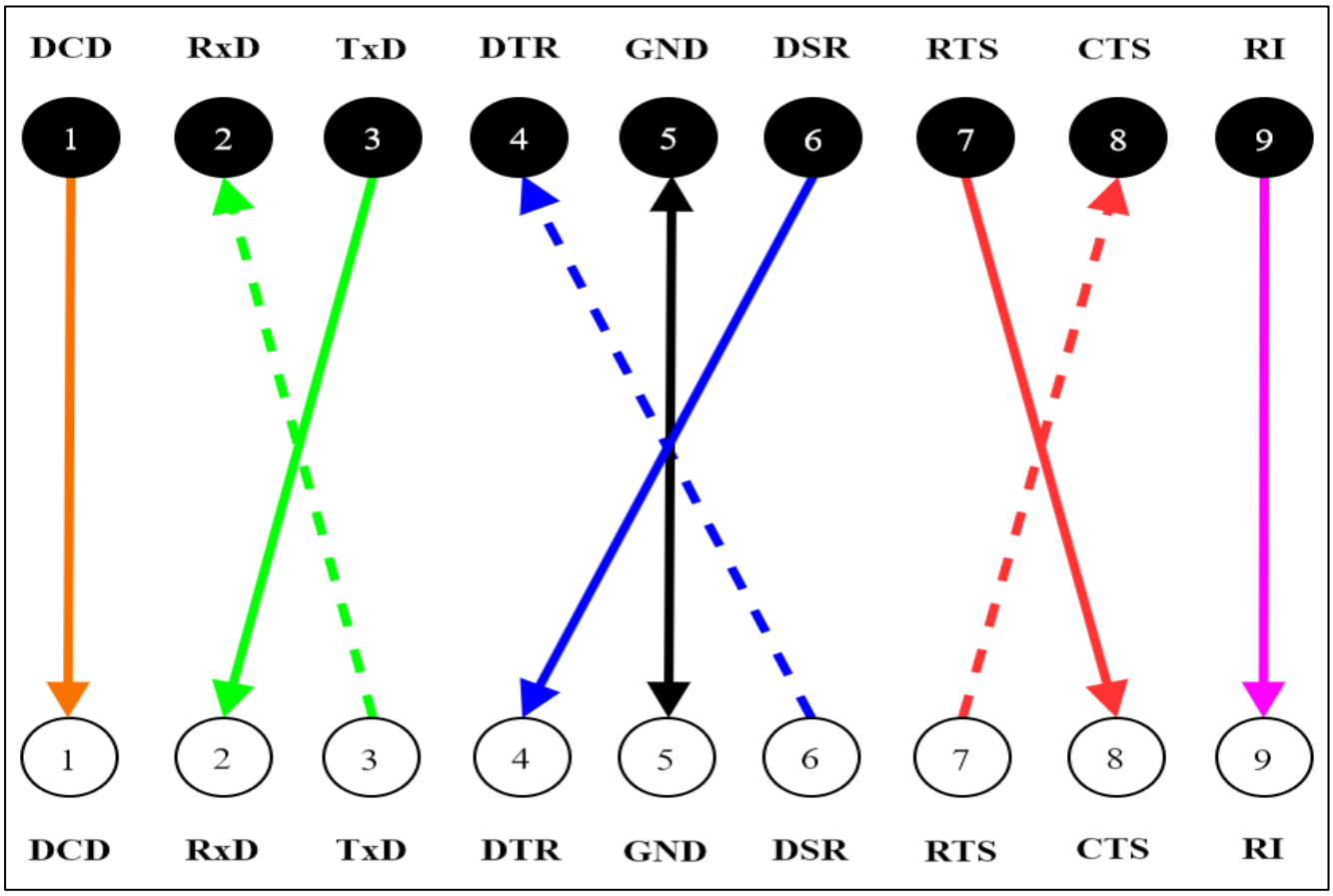

Рисунок 1.9 – Соединение с перекрёстным подключением

Выявить наш тип кабеля не предоставляется возможным, проверим это во время подключения всей установки.

1.5. Параметры последовательного интерфейса

Последовательное соединение имеет 5 параметров:

1. **Baud rate** – скорость передачи данных в бодах (бит в секунду) Обычные значения: 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200.

2. **Parity** – чётность (проверка чётности). Используется для обнаружения ошибок в переданных данных. Всего есть 5 типов проверки чётности:

– **No parity** (нет проверки) – не используется контроль чётности;

– **Even Parity** (чётная чётность) – добавляется бит, чтобы общее количество единиц в передаваемых данных было четным;

– **Odd Parity** (нечётная чётность) – добавляется бит, чтобы общее количество единиц в передаваемых данных было нечетным;

– **Mark Parity** (помечаемая чётность) – бит чётности выставляется в 1;

– **Space Parity** (пустая чётность) – бит чётности выставляется в 0.

3. **Data Bits** (биты данных) – указывает количество битов данных в каждом передаваемом символе. Обычные значения: 5, 6, 7, 8.

4. **Stop Bits** (стоп-биты) – указывает количество битов, используемых для обозначения конца символа. Обычные значения: 1, 1.5, 2.

5. **Flow Control** (управление потоком) – определяет методы контроля потока данных между устройствами для предотвращения переполнения буфера. Основные типы:

– **None** (без управления потоком данных) – при такой настройке не используется управление потоком данных;

– **Hardware** (аппаратное управление потоком данных) – использует дополнительные линии связи (RTS/CTS или DTR/DSR);

– **Software** (программное управление потоком данных) – использует специальные управляющие символы (XON/XOFF).

При подключении двух устройств обязательна идентичность данных настроек.

1.6. Подключение установки

Для подключения установки используем все вышеописанные компоненты воедино. Подключаем преобразователь к компьютеру и устанавливаем драйвер. После установки индикация «**Active**» будет гореть красным цветом.

Включаем весы и ставим их в режим беспрерывной передачи данных.

Параметры для передачи данных с весов указаны в их техническом руководстве [4, с. 23]. На весах есть только одно изменяемое значение – это скорость передачи данных, остальные настройки даны по умолчанию и никак не могут быть изменены. Скорость передачи данных с весов устанавливается сразу

после выбора режима передачи. На выбор даётся 1200, 2400, 4800 и 9600 бод. Выберем максимальную скорость в 9600 бод.

Параметры на весах и преобразователе должны быть идентичными. Значения параметров приведены в таблице 1.5.

Таблица 1.5 – Параметры последовательного порта

Параметр	Значение
Baud rate	9600
Parity	No Parity
Data Bits	8
Stop Bits	1
Flow Control	None

Зайдём в настройки порта преобразователя MOXA и выставим все значения, представленные ранее в таблице 1.5. Результат на рисунке 1.10.

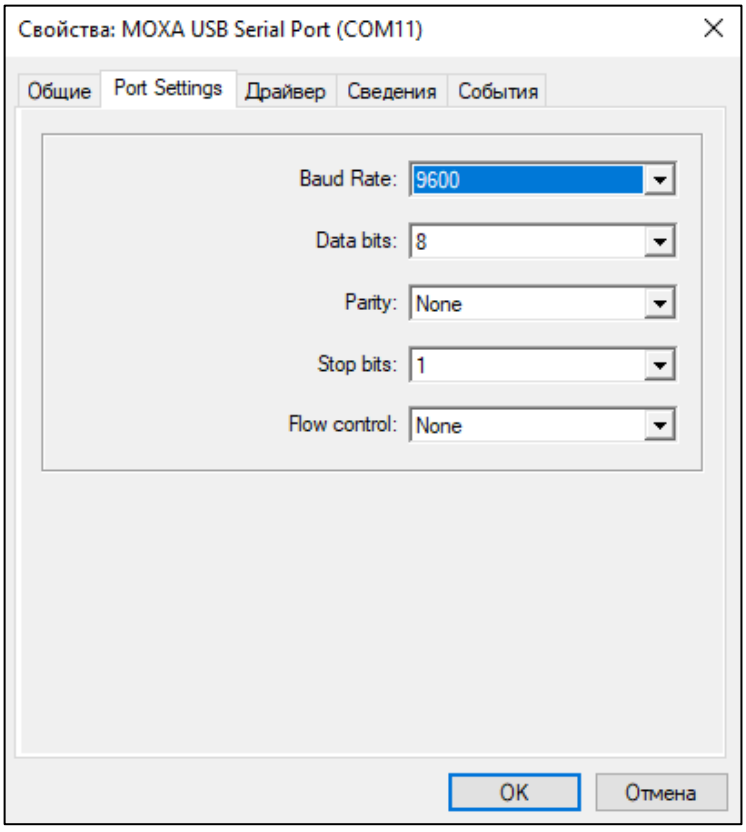

Рисунок 1.10 – Параметры преобразователя в свойствах Windows

Подключим кабелем преобразователь и весы.

После подключения можно воспользоваться утилитой PComm Lite, а именно программой PComm Terminal Emulator от компании MOXA, скачав её на официальном сайте. В этой программе можно проверить работоспособность последовательного порта.

Перед началом работы необходимо выставить значения из таблицы 1.5. Результат на рисунке 1.11.

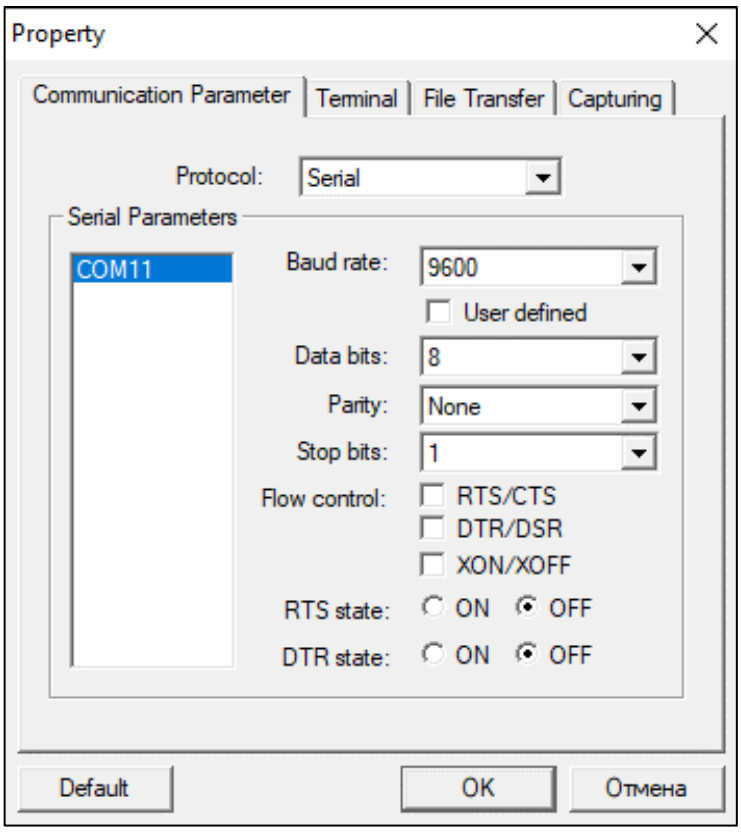

Рисунок 1.11 – Ввод параметров в программе PComm Terminal Emulator

После запуска приложение должно выводить значения с весов, однако, при первом запуске такого не происходит. Более того, на преобразователе не загорается индикация чтения, следовательно, можем предположить, что проблема может быть либо в том, что есть внутренняя неисправность внутри весов или преобразователя, либо проблема в проводе (неправильное расположение контактов передачи).

Для проверки весов на неисправность поступим достаточно просто – возьмём другие весы. Если подключить другие весы, то проблема не исчезает, следовательно, можем проверить другие компоненты установки.

Чтобы проверить преобразователь, достаточно замкнуть RxD и TxD (2 и 3 контакты на чтение и запись) – тогда при отправке данных эти же данные и будут возвращаться. Как показал данный эксперимент, отправленные данные возвращаются обратно в исходном виде, а также загорается индикация чтения, следовательно, можем предположить, что проблема в неправильном размещении контактов – их нужно инвертировать.

Для работы с данными весами достаточно контактов под номерами: 2, 3 и 5 (согласно техническому руководству). Контакт под номером 5 – это земля, он нужен, так как сигнал идёт по двум проводам. Нужно инвертировать 2 и 3 контакты – сделаем это с помощью проводов от Arduino, результат на рисунке 1.12.

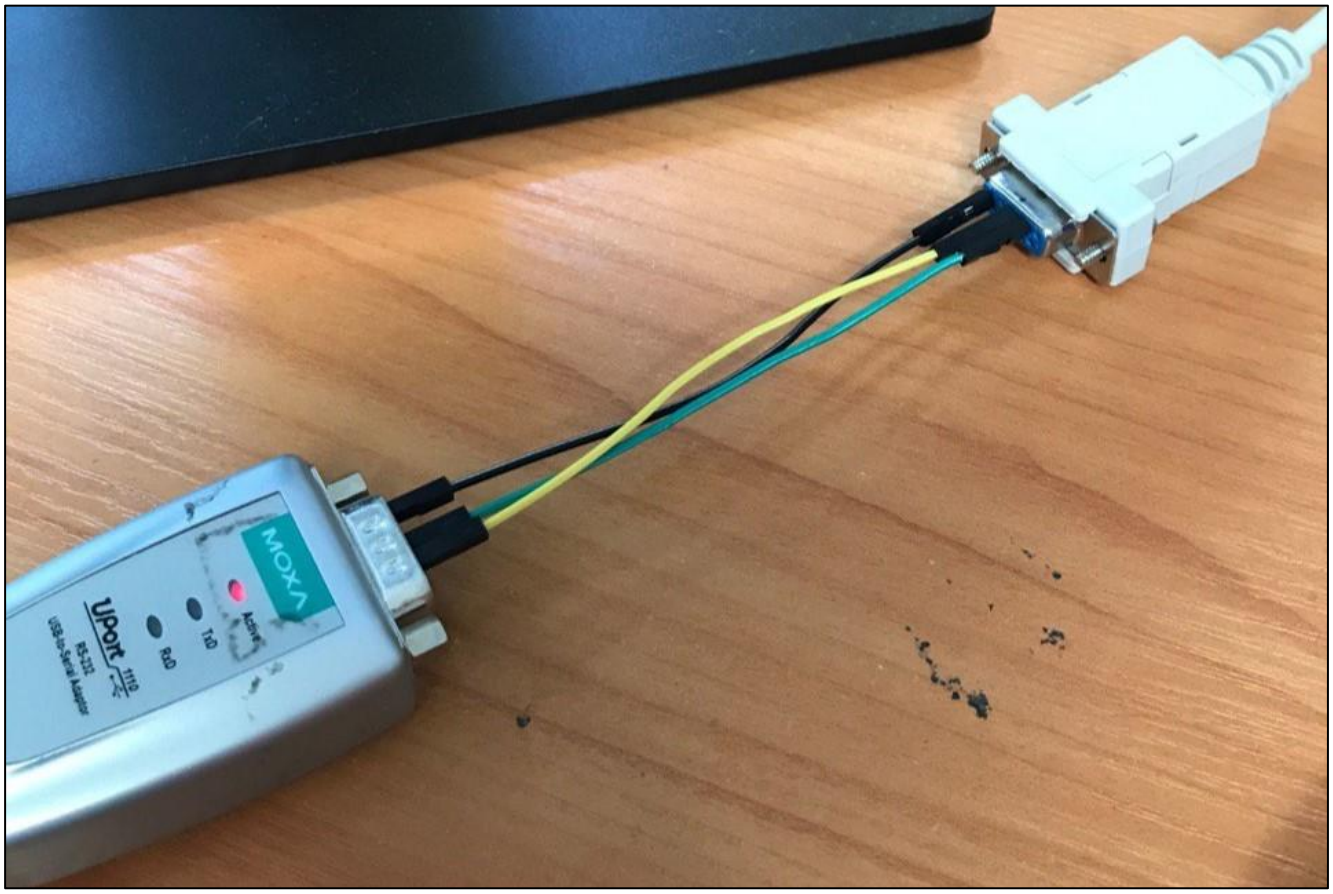

Рисунок 1.12 – Подключение преобразователя к кабелю перекрёстным подключением

После подключения запускаем программу для проверки работоспособности установки. Результат на рисунке 1.13.

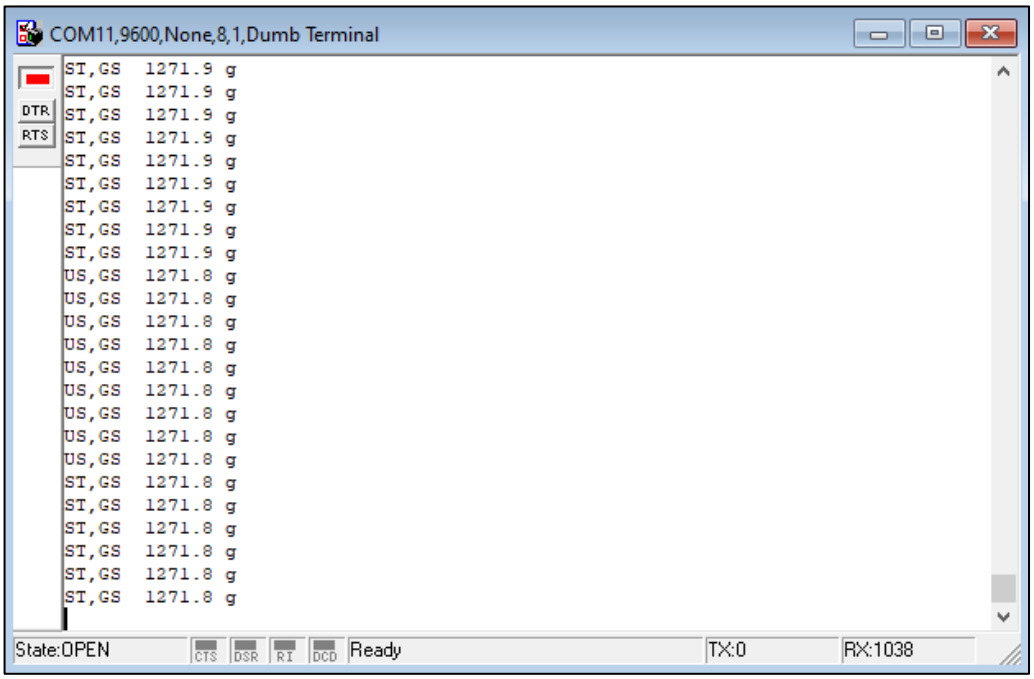

Рисунок 1.13 – Вывод из приложения PComm Terminal Emulator

Итак, проблема крылась в неправильном подключении разъёмов. Для того, чтобы значения выводились корректно – контакты на кабеле нужно перепаять.

1.7. Средства разработки

Для написания драйвера будут использоваться следующие средства разработки, согласно заданию:

– язык программирования C++;

– интегрированная среда разработки Qt Creator, в которой содержится встроенный фреймворк Qt Framework и модуль QtSerialPort;

– Декларативный язык программирования QML.

Опишем все вышесказанные средства разработки.

1.7.1. Язык программирования C++

Написание драйверов, обеспечивающих взаимодействие с аппаратурой, обычно осуществляется на компилируемых языках, благодаря их высокой производительности и близкому контролю над ресурсами. В компилируемых языках обеспечивается эффективное управление памятью, низкоуровневый доступ к аппаратуре, статическая типизация для более раннего обнаружения ошибок и

компиляция в машинный код для повышения производительности. Интерпретируемые языки редко используются для этой цели из-за потенциальных ограничений производительности и уровня абстракции. Однако, некоторые проекты могут быть реализованы с использованием интерпретируемых языков для написания драйверов, так как с их помощью можно написать код быстрее, хотя это является исключением и требует особых обоснований в контексте конкретных требований системы.

Есть ряд компилируемых и статически типизированных языков программирования, использующих разные вариации работы. Например, язык C поддерживает процедурное программирование и в нём нет ООП, язык C# поддерживает объектно-ориентированное программирование (то есть обязательное наличие классов), а язык C++ поддерживает и ту и иную концепцию, позволяет работать с передачей данных измерительных приборов в обе стороны быстро и эффективно, поэтому используется этот язык программирования, так как он подходит для компании больше всего. C++ также имеет множество удобных фреймворков для работы с ним.

Фреймворк – это набор инструментов, библиотек, структур данных и готовых решений, предназначенные удобного процесса разработки программного обеспечения. Они представляют собой своеобразный «скелет» или основу, на которой разрабатывается приложение или программное решение.

1.7.2. Qt Creator, Qt Framework и модуль QtSerialPort

Qt Creator [8] – это интегрированная среда разработки (IDE) для создания приложений на C++ с использованием Qt Framework, она поддерживает кроссплатформенную разработку, предоставляет инструменты для отладки, автодополнения кода и визуального проектирования интерфейсов.

Qt Framework [9] – это популярный кроссплатформенный фреймворк для разработки программного обеспечения. Он предоставляет инструменты и библиотеки для создания графических пользовательских интерфейсов (GUI), а

также для разработки приложений, работающих на различных операционных системах. Qt фреймворк также используется и другими языками программирования, например, Python. На сайте разработчика есть обширная подробная документация по методам, классам и модулях, используемых в фреймворке.

Так как Qt Framework достаточно популярен, на форумах для разработчиков C++, использующих данный фреймворк, есть много статей об использовании и о проблемах, которые могут возникнуть. Есть также учебные пособия и книги по данному фреймворку [10].

Так как драйвер будет разрабатываться в интегрированной среде разработки Qt Creator с использованием Qt Framework, то воспользуемся модулем QtSerialPort [11], которая позволяет работать с последовательными портами в среде Qt.

Данный модуль имеет два класса: QSerialPort [12] и QSerialPortInfo [13]. В таблице 1.6 представлен функционал каждого класса модуля QtSerialPort.

Таблица 1.6 – Функционал классов модуля QtSerialPort

Класс QSerialPort	**Класс QSerialPortInfo**
1. Выставление параметров последовательного интерфейса. 2. Открытие/закрытие порта. 3. Чтение из порта и запись в порт. 4. Предоставление информации о состоянии порта.	1. Предоставление информации о всех доступных последовательных портах. 2. Выдаёт информацию о конкретном порте (имя, описание и т.д.).

Основная работа будет протекать с классом QSerialPort, так как в нём есть необходимые методы для всех необходимых манипуляций над последовательным интерфейсом, в то время как класс QSerialPortInfo используется для сводной информации – например, какие на данный момент подключены COM-порты, их названия в системе, информация об их имени (наименования устройств), их производителе и так далее.

1.7.3. Декларативный язык программирования QML

QML (Qt Modeling Language) – язык программирования, предназначенный для создания современных и интерактивных пользовательских интерфейсов в Qt. Его декларативный подход упрощает описание и настройку элементов интерфейса, делая код более интуитивно понятным. Благодаря поддержке JavaScript, QML позволяет легко реализовать логику приложения и обработку событий. Кроме того, QML упрощает создание анимаций и переходов между состояниями интерфейса, а также автоматически обновляет интерфейс при изменении данных. Этот язык поддерживает разработку приложений для различных платформ, таких как Windows, macOS, Linux, Android и iOS.

Преимущество декларативных языков программирования заключается в том, что декларативные языки программирования фокусируются на описании желаемого результата, а не на том, как этот результат достичь. В отличие от императивных языков, где каждый шаг детально прописывается, в декларативных языках программист указывает конечную цель, а система сама определяет последовательность действий для её достижения. Этот подход делает код более абстрактным, читаемым и упрощает разработку, так как программа скрывает сложные детали выполнения от разработчика.

Вывод по разделу 1

В первом разделе мы проанализировали предметную область, был выявлен класс точности весов, было произведено описание весов, преобразователя и кабеля, с помощью которого мы соединяем весы и преобразователь вместе. В конечном итоге, установка была подключена, начался вывод с весов и, таким образом, мы можем начинать нашу дальнейшую работу.

Также в данном разделе описаны средства разработки: язык C++, среда разработки Qt Creator со своим встроенным фреймворком и модулем, которым мы будем пользоваться, а также декларативный язык программирования QML, который нужен для создания графического интерфейса.

2. РАЗРАБОТКА ДРАЙВЕРА ДЛЯ ВЫСОКОТОЧНЫХ ВЕСОВ

При написании драйвера нам следует придерживаться определённых этапов, или, можно сказать, общепринятые подходы к программированию, которые помогут нам при выполнении задания [14, с. 5-6]. Среди этапов можно выделить:

1. Постановку задачи.
2. Анализ и описание задачи.
3. Выбор алгоритмов и путей решения.
4. Описание структуры программы.
5. Кодирование.
6. Отладка и тестирование.
7. Корректировка.
8. Выход программы.
9. Поддержка кода.

Данные этапы можно объединить в разделы и, таким образом, описать все этапы программирования, чем мы и займёмся при дальнейшей работе.

2.1. Постановка задачи

По техническому заданию требуется написать драйвер, который можно будет внедрить в приложение. Перед внедрением необходимо написать алгоритм для драйвера и сделать графический интерфейс, чтобы колорист смог первично протестировать весы вместе с драйвером и, если весы пройдут тестирование, то можно будет внедрять драйвер непосредственно в приложение.

При проектировании можно отметить функциональные и нефункциональные требования к системе. При изучении технического руководства к весам было выявлено, что они имеют смены типа тары, имеют индикацию «стабильного» веса, а их максимальная масса взвешивания составляет 7 кг. Исходя из этой информации были выдвинуты составить функциональные и нефункциональные требования к приложению.

Функциональные требования:

1. Вывод значений с весов должен происходить в режиме реального времени.

2. Добавить возможность фиксации значения в определённый момент времени по кнопке.

3. Хранение зафиксированных значений.

4. Очистка зафиксированных значений.

5. Добавить функцию обнуления.

6. Обнулять значение веса при фиксации.

7. Показывать индикацию при типе тары (брутто/нетто).

8. Показывать индикацию при стабильном/нестабильном весе.

9. Выдавать индикацию, если был зафиксирован перевес.

10. Добавить к снятым значениям параметры, при которых они были сняты (при стабильном/нестабильном весе и является ли отснятое значение массой брутто/нетто).

Нефункциональные требования:

1. Кнопки должны быть такого размера, чтобы удобно было нажимать их на сенсорном терминале.

2. Функции обнуления и замера брутто не должны тормозить общую работу приложения.

3. Компоненты интерфейса должны меняться пропорционально разрешению главной формы приложения.

4. Минимальное разрешение – 1200x700 пикселей.

5. Программа должна самостоятельно определить, к какому COM-порту подключены весы.

6. Сборку приложения реализовать с использованием CMake.

2.2. Анализ и описание задачи

Программа пишется для колористов, которые работают на сенсорных терминалах (сенсорных компьютерах), им необходим соответствующий

интерфейс, позволяющий удобно обращаться с функционалом программы. Именно поэтому нужно использовать большой экран для вывода и кнопки большого размера. Часто используемая кнопка (или кнопки) должны располагаться в месте, где удобнее всего на них нажимать. Однако точно узнать, удобно ли людям работать при таком интерфейсе можно только при тестировании приложения и при наличии жалоб поменять местоположение компонентов интерфейса.

Так как колористы обычно используют приложение во весь экран терминала, то вполне логично менять размеры компонентов интерфейса пропорционально разрешению главной формы. Об этих аспектах позаботимся при написании графической части работы.

Программа по фиксированию веса может работать в двух вариантах: либо фиксирование веса происходит по кнопке, отправляя команду на весы, либо весы передают своё значение в режиме реального времени и, как только мы получим стабильное значение – фиксируем его. Наиболее удобно будет работать при втором варианте, тем более если это будет сопровождаться индикацией характера взвешивания. Именно поэтому выбран этот вариант использования, пусть даже это будет более ресурсозатратно (очевидно), из-за постоянного считывания значения с весов и передача его в графический интерфейс. С другой стороны, вся наша работа связана с использованием оборудования вроде весов, наша обязанность сделать эту работу максимально комфортной – поэтому необходима реализация именно этого пункта технического задания. По этой же причине необходима реализация функции определения, к какому COM-порту подключены весы.

Функция обнуления играет важную роль при использовании весов. Дело в том, что при работе на производстве, да ещё и с большим количеством людей происходит масса вибраций. Весы могут чувствительно к этому относиться и из-за этого, после снятия значения, при возврате к исходному (нулевому) весу, весы могут показывать ненулевое значение. Это недопустимо, так как на производстве весы используются как главный контролёр учёта слитых пигментов, а также

именно по ним ведётся налив пропорций по рецепту. Функция обнуления упрощает работу с весами, которые «отклонились от курса». Обнуление может происходить неявно после того, как фиксированное значение будет записано в буфер, а может быть реализована и явно, в виде кнопки, чтобы человек сам контролировал процесс.

Также обнуление должно происходить после каждого фиксирования значения. Это сделано для того, чтобы колорист самостоятельно не нажимал на кнопку обнуления, когда наливает новый компонент в ту же тару.

Работая в приложении компании, заходя во вкладку с настройкой весов есть кнопка «обнулить» для тестирования работоспособности. Однако при тестировании этой функции возникла проблема – при многократном нажатии приложение зависало, либо вылетало. Это вызвано тем, что при нажатии кнопки многократно вызывается рекурсия, которая сильно тормозит главный поток приложения. Читая разные форумы по программированию, можно наткнуться на высказывания, связанные с работой приложения с графическим интерфейсом. Пользователи форумов отмечают, что ни в коем случае нельзя использовать большие вычисления в главном потоке (так как у него не будет времени отрисовывать графическую составляющую). Рекомендуется такие вычисления выводить в отдельный поток, так как графический интерфейс всегда строится в главном потоке и перенести его в другой поток можно, но эта процедура крайне нежелательна. Нам же предстоит разобраться, как подать команду на весы, не замедляя при этом главный поток даже при многократном нажатии.

Для написания приложения нам нужно использовать утилиту «CMake» [15], которая также способна автоматизировать процесс установки и сборки пакетов. Дело в том, что приложение StartColor делается с помощью этого «сборщика».

Важно заметить, что сам «CMake» сборкой не занимается, лишь генерируя файлы используя обычный файл с расширением *.txt*. «CMake» поддерживает большинство компиляторов. Он является своего рода стандартом для систем сборок, таких как make и Ninja.

2.3. Выбор алгоритмов и путей решения

Согласно тому, что мы описали ранее, нам нужна программа, способная выводить значения с весов в режиме реального времени. Первое, что приходит на ум, это установить весы в режим «чтения-записи» и в цикле бесконечно передавать команду на чтение с весов. Пока мы не пользуемся графическим интерфейсом, этот вариант имеет место быть, чтобы узнать, что будут выдавать весы в консоль. Однако при дальнейшей работе главный поток приложения будет занят приёмом/отправкой данных на весы и из-за этого у нас не будет работать графический интерфейс (как мы говорили ранее), следовательно, нужно искать другой вариант.

Другой вариант заключается в использовании «сигналов и слотов», которые доступны в библиотеке Qt. В разных наборах инструментов разработки используют callback-функции (функции обратного вызова). Callback– это указатель на функцию, если вы хотите, чтобы функция обработки уведомляла о каком-либо событии – нужно передать указатель на другую функцию в функцию обработки [16].

Сигналы и слоты используются как альтернатива callback-функциям. Сигнал излучается при достижении определённого события при помощи макроса **emit** [функция_сигнала()]. Слот – это функция, которая вызывается в ответ при получении сигнала.

В качестве сигнала будет использоваться readyRead() из класса QSerialPort. В качестве слота подключим нашу функцию, которая будет брать значение с весов.

При данной реализации алгоритма графический интерфейс не будет виснуть, так как вызовы происходят асинхронно.

В общем виде вся работа кода достаточно проста и описана с помощью схемы на рисунке 2.1, где есть блок с получением значения с весов.

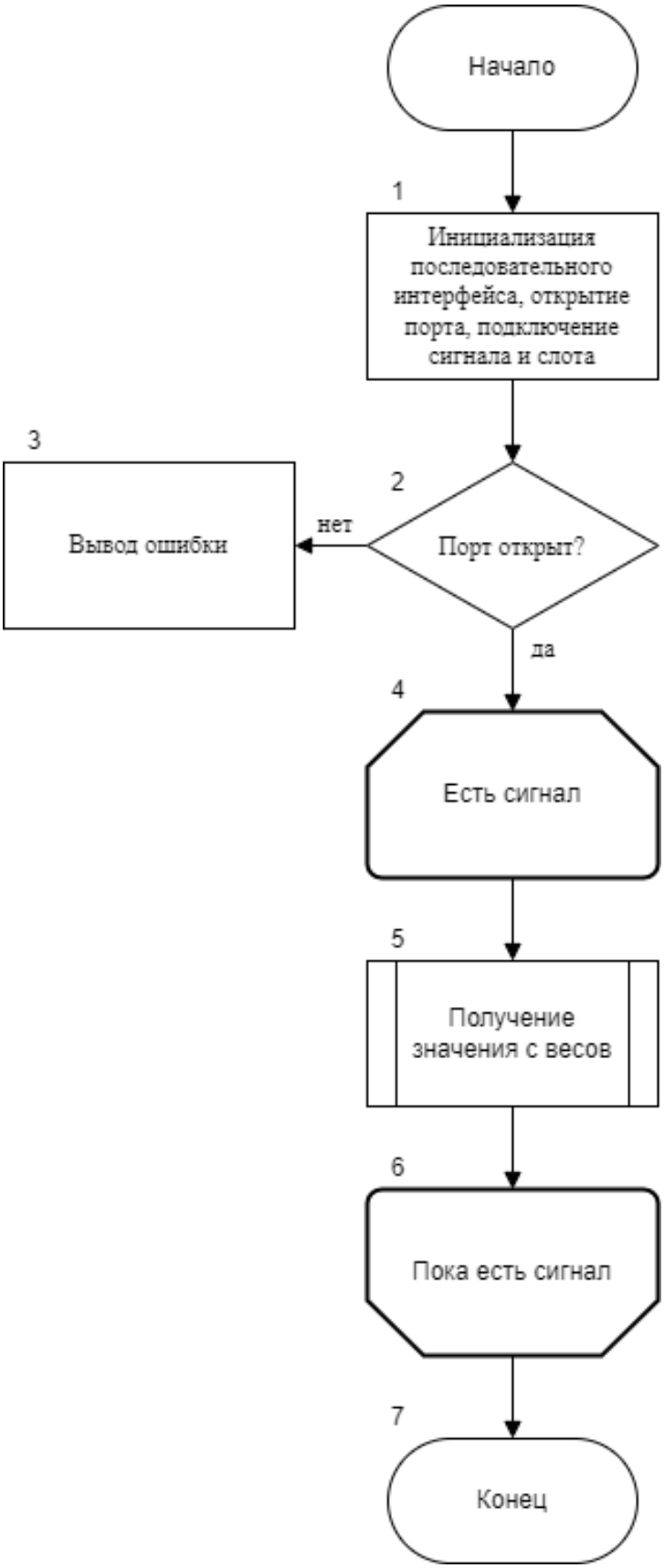

Рисунок 2.1 – Работа кода в общем виде

Схема алгоритма получения значения с весов показано на рисунке 2.2.

Рисунок 2.2 – Алгоритм получения значения с весов

Получать данные с весов будем с помощью алгоритма, показанном на рисунке 2.3

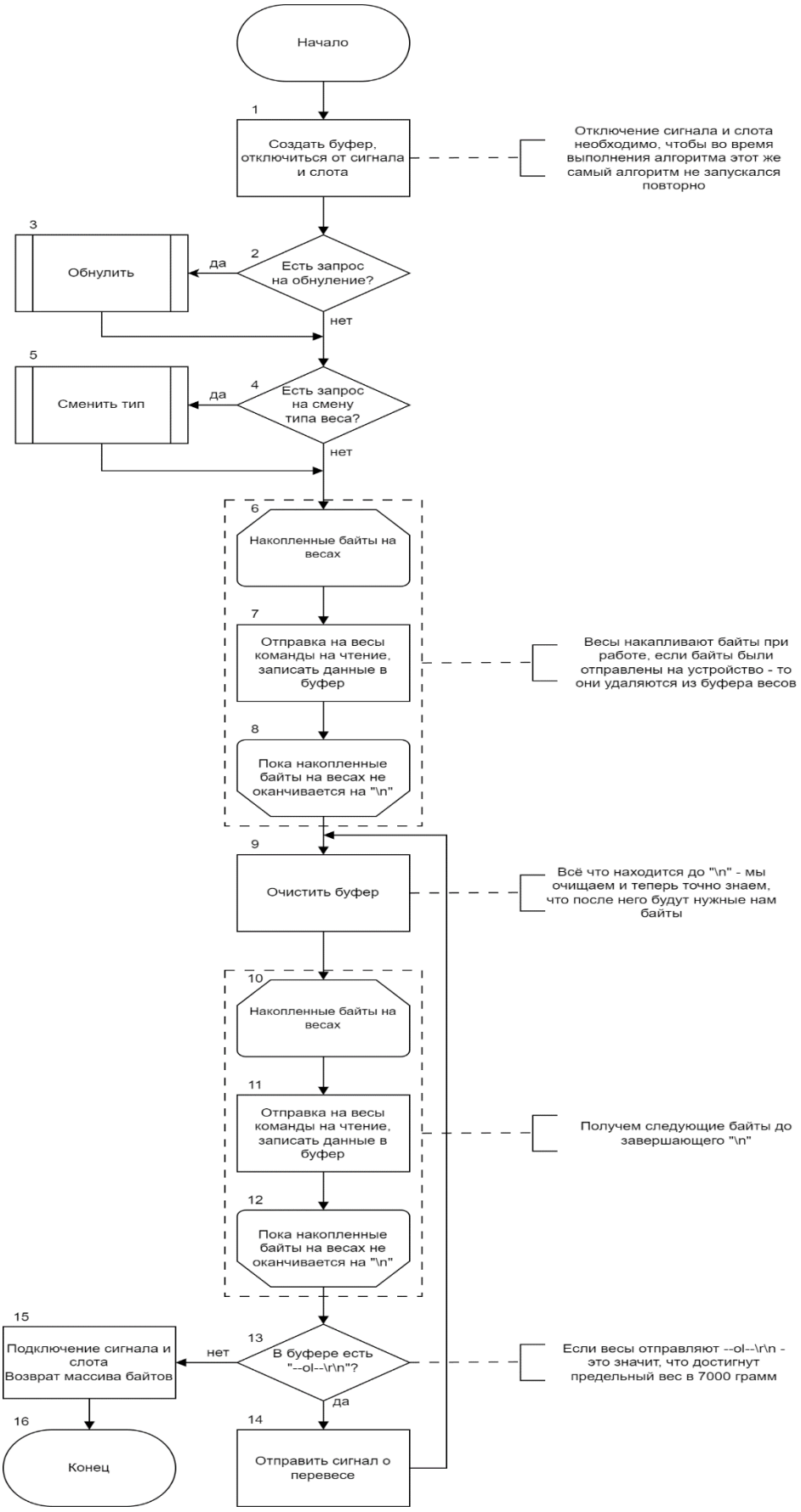

Рисунок 2.3 – Алгоритм преобразования массива байтов с весов

Помимо этих двух алгоритмов также составлены алгоритм обнуления значения весов (рисунок 2.4) и смены веса (рисунок 2.5).

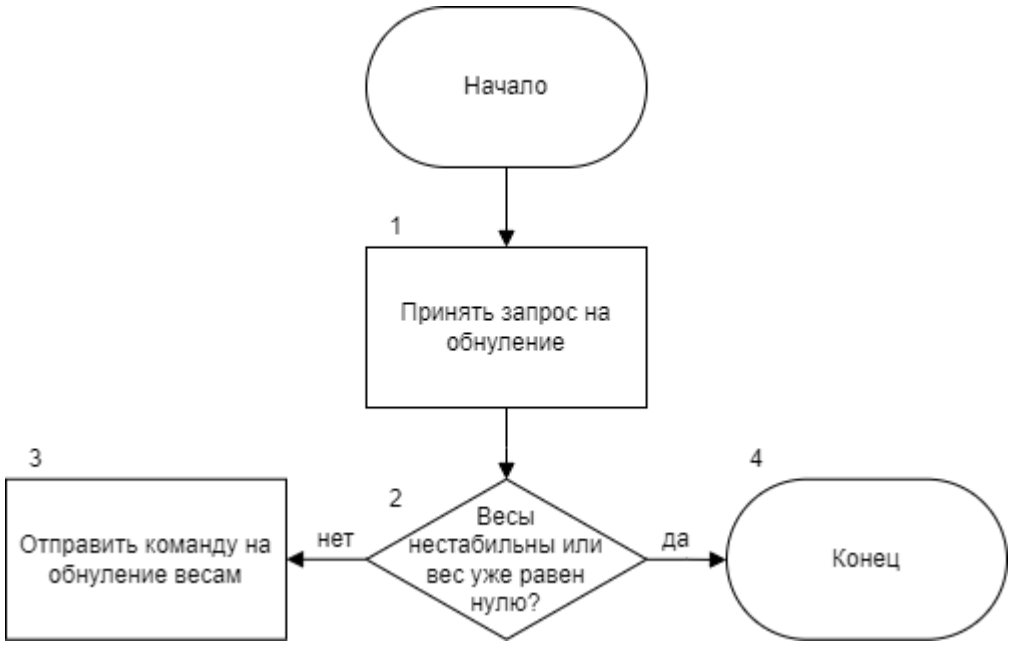

Рисунок 2.4 – Алгоритм отправки команды на обнуление весам

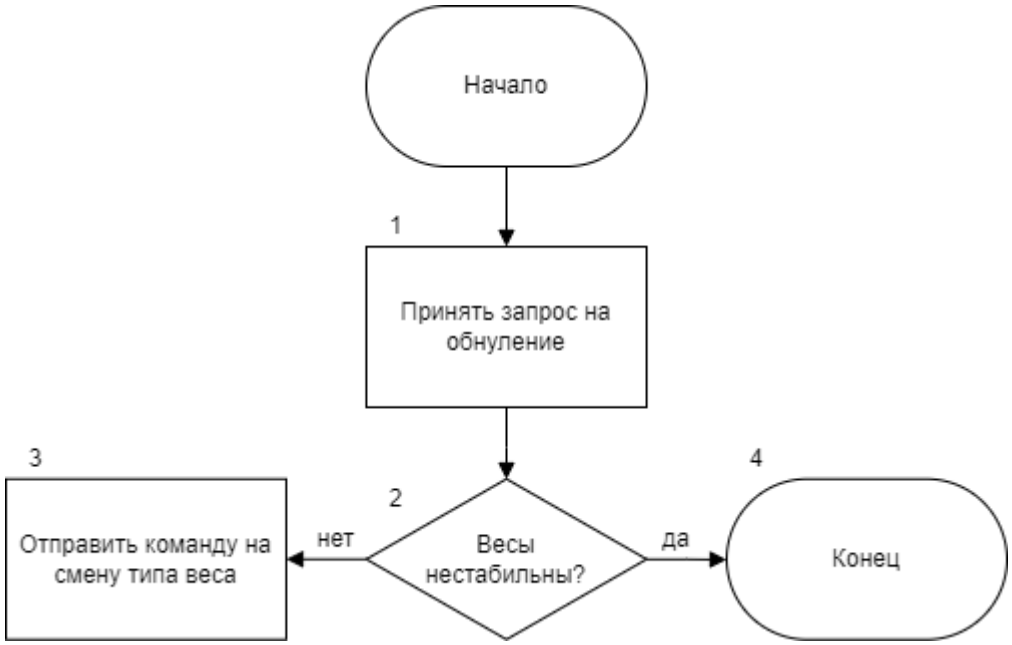

Рисунок 2.5 – Алгоритм отправки команды на смену типа веса весам

Алгоритм получения значения с весов будет содержать в себе алгоритм преобразования массива байтов, который, в свою очередь, будет вызывать алгоритм обнуления или смены типа веса, если был запрос либо на то, либо на другое.

Так как вес будет приходить циклично с использованием сигнала и слота, то в таком случае запросы будут проверяться на *каждом круге* получения веса, таким образом отправка команды на обнуление или смену типа никак не будет мешать работе основного алгоритма по получению веса.

Запрос на обнуление и смену типа планируется сделать по кнопке в графическом интерфейсе.

2.4. Описание структуры программы

Как уже было оговорено ранее, в основном мы будем использовать модуль QtSerialPort с его классами QSerialPort и QSerialPortInfo, а для сборки будет использоваться CMake.

Для начала, нужно создать класс для дальнейшей работы с весами – назовём его TScale. В нём будут методы, описанные в таблице 2.1.

Таблица 2.1 – Методы, которые будут реализованы в драйвере

Метод	Описание
Init	Метод, в котором происходит инициализация весов (устанавливается скорость передачи, биты данных, чётность, стоп-биты, контроль потока и порт, к которому подключены весы)
openPort **closePort**	Методы, предназначенные для открытия и закрытия порта
recieveData	Получает массив байтов с порта, а затем в нём же происходит преобразование к нужному 17-байтовому массиву
getValue	Метод, в котором извлекаются 8-13 байты, а также этот метод является и слотом для сигнала readyRead() класса QSerialPort
dataZero	Метод, в котором отправляется команда на обнуление
dataTare	Метод, в котором отправляется команда на смену типа веса

Метод инициализации и открытия порта планируется сделать в конструкторе класса, согласно первому блоку на рисунке 2.1. Метод reciveData() будет закодирован согласно алгоритму, описанном на рисунке 2.3, getValue() согласно рисунку 2.2, dataZero() и dataTare() согласно рисункам 2.4 и 2.5 соответственно.

Для вывода данных в консоль воспользуемся библиотекой QDebug. Консоль поможет нам в регулировании процесса при запуске кода.

Подключим все библиотеки, воспользовавшись директивой #include.

Экземпляр класса из C++ будет передан в QML для работы с ним.

Графическая часть программы будет состоять из трёх частей: левой, центральной и правой. С левой стороны будут располагаться результаты расчёта, а также предусмотрен функционал очистки этих результатов. В центральной части будут располагаться индикации и вывод веса, а также будет расположен функционал обнуления весов. Справа будет располагаться функционал фиксации значения и смена типа взвешивания, а также будет поле, которое будет хранить значение массы тары. Исходя из вышеперечисленного функционала, сделаем диаграмму вариантов использования интерфейса (рисунок 2.6).

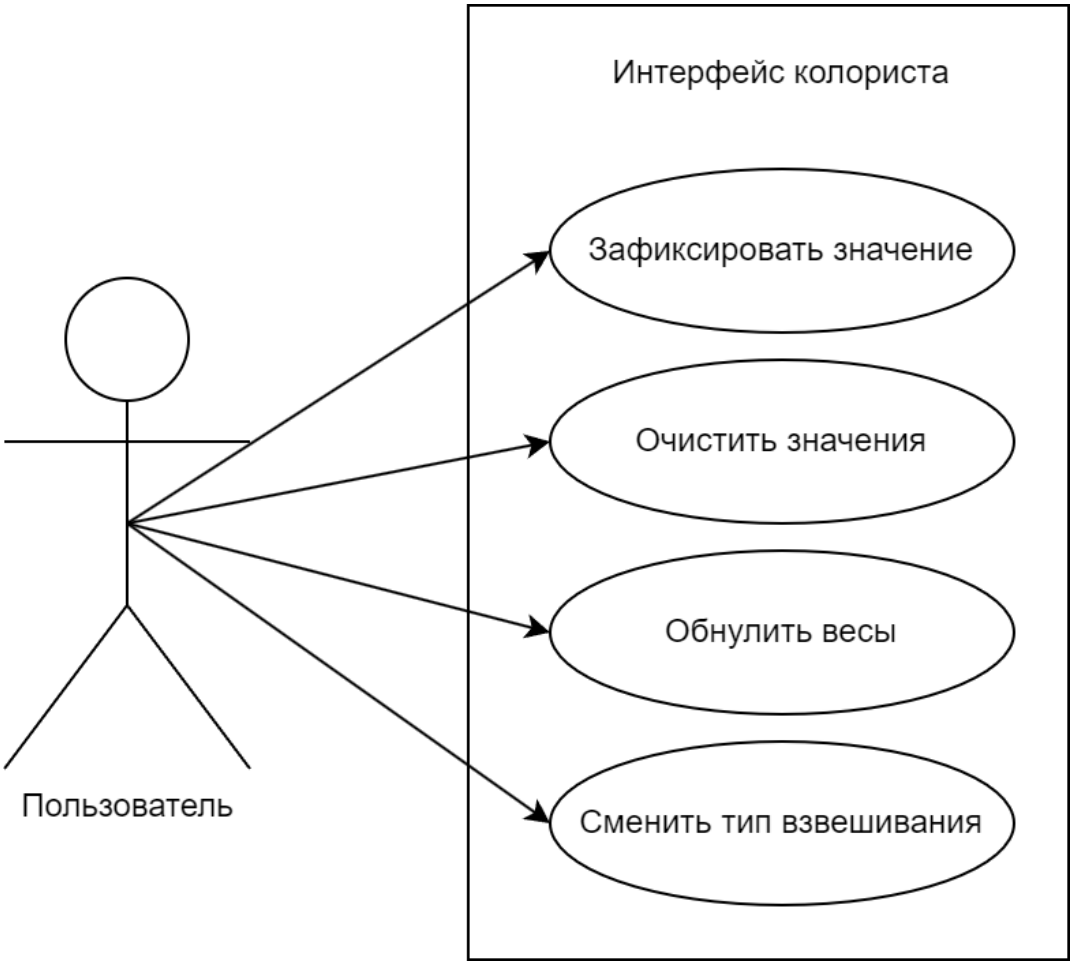

Рисунок 2.6 – Диаграмма вариантов использования интерфейса

Графическая составляющая будет описана после написания основного кода (драйвера).

2.5. Кодирование

Создадим класс TScale. Распишем все необходимые методы в файле TScale.h, как представлено в листинге В.1 приложения В.

При создании класса TScale, создадим для него конструктор, в котором создаётся динамически класс QSerialPort, затем, будем вызывать методы инициализации и открытия порта, а также подключим сигнал и слот (листинг 2.1).

Листинг 2.1 – Конструктор класса TScale

```
TScale::TScale(QObject *parent) : QObject{parent} {
    serial = new QSerialPort(); // Выделяем память под класс
    Init(); // Метод инициализации
openPort(); // Открыть порт
// Подключить сигнал и слот
    _connectionToValue  = connect(serial, &QIODevice::readyRead,
                                  this, &TScale::getValue);
};
```

Как мы видим на листинге 2.1, мы воспользовались оператором *new* для выделения динамической памяти для экземпляра класса QSerialPort – не забудем воспользоваться оператором *detele* чтобы не допустить утечку памяти. Запишем это в деструктор класса TScale. Результат на листинге 2.2.

Листинг 2.2 – Деструктор класса TScale

```
TScale::~TScale() {
    delete serial; // Удаляем экземпляр класса в деструкторе
}
```

Установим настройки передачи данных через экземпляр класса QSerialPort, воспользуемся для этого отдельным методом Init, показанным на листинге 2.3. В этом же коде можем использовать класс QSerialPortInfo для выявления нашего устройства через его имя: если устройство с таким именем есть, то только с ним мы и будем работать, не затрагивая остальные COM-порты.

Листинг 2.3 – Метод Init, в котором происходит инициализация нашего устройства

```
// Метод инициализации, вызывается в конструкторе класса.
// Выставляет параметры последовательного порта и открывает его.
void TScale::Init() {
    QString name;
    foreach (auto info, QSerialPortInfo::availablePorts())
    {
        qDebug() << info.portName() + " " +
                    info.description() + " " +
                    info.manufacturer();
        if (info.manufacturer().contains("Moxa"))
            name = info.portName();
    }
    serial->setPortName(name); // порт
    serial->setBaudRate(QSerialPort::BaudRate::Baud9600); // скорость
    serial->setDataBits(QSerialPort::DataBits::Data8); // биты данных
    serial->setParity(QSerialPort::Parity::NoParity); // парность
    serial->setStopBits(QSerialPort::StopBits::OneStop); // стоп-биты
    serial->setFlowControl(QSerialPort::FlowControl::NoFlowControl);
// регулирование потока
    inited = true;
}
```

Код открытия порта описано в листинге 2.4, а код закрытия в листинге 2.5.

Листинг 2.4 – Открытие последовательного порта

```
// Открывает порт (если открыт – посылает true, иначе false)
bool TScale::openPort() {
    serial->open(QIODevice::ReadWrite);
    if (serial->isOpen() && serial->isReadable()) {
        qDebug() << "PORT OPENED AND READABLE";
        return true;
    } else
        qDebug() << "PORT NOT OPENED AND UNREADABLE";
    return false;
}
```

Листинг 2.5 – Закрытие последовательного порта

```
// Закрывает порт (если закрыт - посылает true, иначе false)
bool TScale::closePort() {
    serial->close();
    if (!serial->isOpen() && !serial->isReadable())
    {
        qDebug() << "PORT CLOSED AND UNREADABLE";
        return true;
    }
    else

        qDebug() << "PORT NOT CLOSED AND READABLE";
    return false;
}
```

В данных функциях тип возвращаемого значения – булевая переменная. Это сделано для дальнейшей работы с операторами if.

Напишем обработчик массива байтов, как показано в листинге 2.6.

Листинг 2.6 – Обработчик массива байтов

```
// Метод получения данных с весов, вызывается в getValue()
QByteArray TScale::dataRecieve() {
    QByteArray ba; // создаём буфер
    disconnect(_connectionToValue); // Отключаемся от сигнала и слота
    if (getRequestZero()) {      // Если есть запрос на обнуление
        dataZero();              // Обнуляем
    }
    if (getRequestTare()) {      // Если есть запрос на смену типа тары
        dataTare();              // Меняем тип
    }
    // Далее идёт цикл, который собирает весь накопленный буфер с весов
    while (!ba.endsWith('\n')) {
        serial->write("R");
        serial->waitForBytesWritten();
        ba += serial->read(1);
    }
    ba.clear(); // Очищаем буфер
```

Продолжение листинга 2.6

```
    // Получаем новую строку
    while (!ba.endsWith('\n')) {
        serial->write("R");
        serial->waitForBytesWritten();
        ba += serial->read(1);
        // Если весы обнаружили перевес
        if (ba == "--ol--\r\n") {
            ba.clear(); // Очищаем буфер
            qDebug() << "OVERWEIGHT"; // Пишем в консоль о перевесе
            _value = 10000; // Значение меняем
            emit valueChanged(); // Даём сигнал о смене значения
            continue; // Начинаем заново собирать строку
        }
        if (ba == "\r\n") { // это для весов SantInt
            ba.clear();
            serial->close();
            serial->open(QSerialPort::ReadWrite);
        }
    }
    // Подключаем сигнал и слот обратно
    _connectionToValue = connect(serial, &QIODevice::readyRead,
                                 this, &TScale::getValue);
    return ba; // Возвращаем массив байтов (буфер)
}
```

Теперь, напишем метод, получающий число из массива байтов. Результат на листинге 2.7.

Листинг 2.7 – Метод, получающий число из массива байтов

```
// Основной метод драйвера – это наш слот, в котором мы забираем
// число из массива байт. Возвращает double.
double TScale::getValue() {
    double number; // Создаём переменную типа double
    QByteArray ba = dataRecieve(); // Получаем массив байт
    QByteArray numberQBA; // Ещё один буферный массив
```

Продолжение листинга 2.7

```
// Далее идёт цикл, в котором очищаются пустые байты
    for (int i = 7; i < 13; i++) {
        if (ba[i] == ' ') // Если найден пустой байт в буфере
            continue;     // В другой массив ничего не вписываем
        numberQBA.append(ba[i]); // Иначе записываем
    }
    number = QString(numberQBA).toDouble(); // Конвертируем из
                                            // массива байтов число
    if (qFuzzyIsNull(number)) // Если число нулевое
        zero = true;
    else
        zero = false;
    if (ba.contains('-')) // Если в массиве есть знак «-»…
        number = -number; // То меняем знак числа
    _value = number;
    emit valueChanged(); // Отправка сигнала
    qDebug() << _value;  // Показываем число в консоли
    return number; // Возврат числа
}
```

В листингах 2.8 и 2.9 содержатся методы, один из которых обнуляет, а второй меняет значение тары с брутто на нетто.

Листинг 2.8 – Метод обнуления весов

```
// Метод используется в recieveData() и будет вызван,
// если был запрос на обнуление. Возвращает число:
// -1 – Весы не стабильны; 0 – Весы уже обнулены; 1 – Успешное обнуление
int TScale::dataZero() {
    setRequestZero(); // Метод, в котором меняем запрос с true на false
    if (!isStable()) { // Если весы не стабильны
        qDebug() << "Scales are not STABLE!"; // Сообщаем в консоль
        return -1; // Возвращаем ответ что весы не стабильны
    }
```

Продолжение листинга 2.8

```
    if (zero) { // Если весы уже нулевые
        qDebug() << "Value already 0"; // Сообщаем в консоль
        return 0; // Возвращаем ответ что весы уже обнулены
    }
    serial->close(); // Закрываем порт
    serial->open(QSerialPort::WriteOnly); // Включаем порт в чтение
    serial->write("Z"); // Посылаем команду на обнуление
    serial->waitForBytesWritten(); // Ждём обработки команды
    serial->close(); // Закрываем порт
    serial->open(QSerialPort::ReadWrite); // Открываем в режиме
                                          // чтения-записи
    return 1; // Возвращаем ответ, что обнуление успешно
}
```

Листинг 2.9 – Метод смены тары с брутто на нетто

```
// Метод используется в recieveData() и будет вызван,
// если был запрос на смену типа тары. Возвращает число:
// -1 – Весы не стабильны; 0 – Режим GROSS; 1 – Режим NET
int TScale::dataTare() {
    setRequestTare(); // Метод, в котором меняем запрос с true на false
    if (!isStable()) { // Если весы не стабильны
        qDebug() << "Scales are not STABLE!"; // Сообщаем в консоль
        //return -1; // Возвращаем ответ что весы не стабильны
    }
    serial->close(); // Закрываем порт
    serial->open(QSerialPort::WriteOnly); // Включаем порт в чтение
    serial->write("T"); // Посылаем команду на смену типа
    serial->waitForBytesWritten(); // Ждём обработки команды
    serial->close();// Закрываем порт
    serial->open(QSerialPort::ReadWrite); // Открываем в режиме
                                          // чтения-записи
    if      (grossMode()) return  0; // Если GS – то возврат 0
    else if   (netMode()) return  1; // Если NT – то возврат 1
}
```

В функции main создадим экземпляр класса TScale, чтобы вызвать конструктор и, таким образом, положить начало работе программы (листинг 2.10).

Листинг 2.10 – Содержимое файла main.cpp

```
// Библиотеки
// #include * * *
int main(int argc, char *argv[])
{
    /*
     * Код, автоматически сгенерированный Qt
    */
    TScale scales; // Создаём экземпляр класса TScale
    engine.load(url);
    app.exec();
    return 0;
}
```

Запустим приложение, зайдём во вкладку «Вывод приложения» и увидим результат на рисунке 2.7.

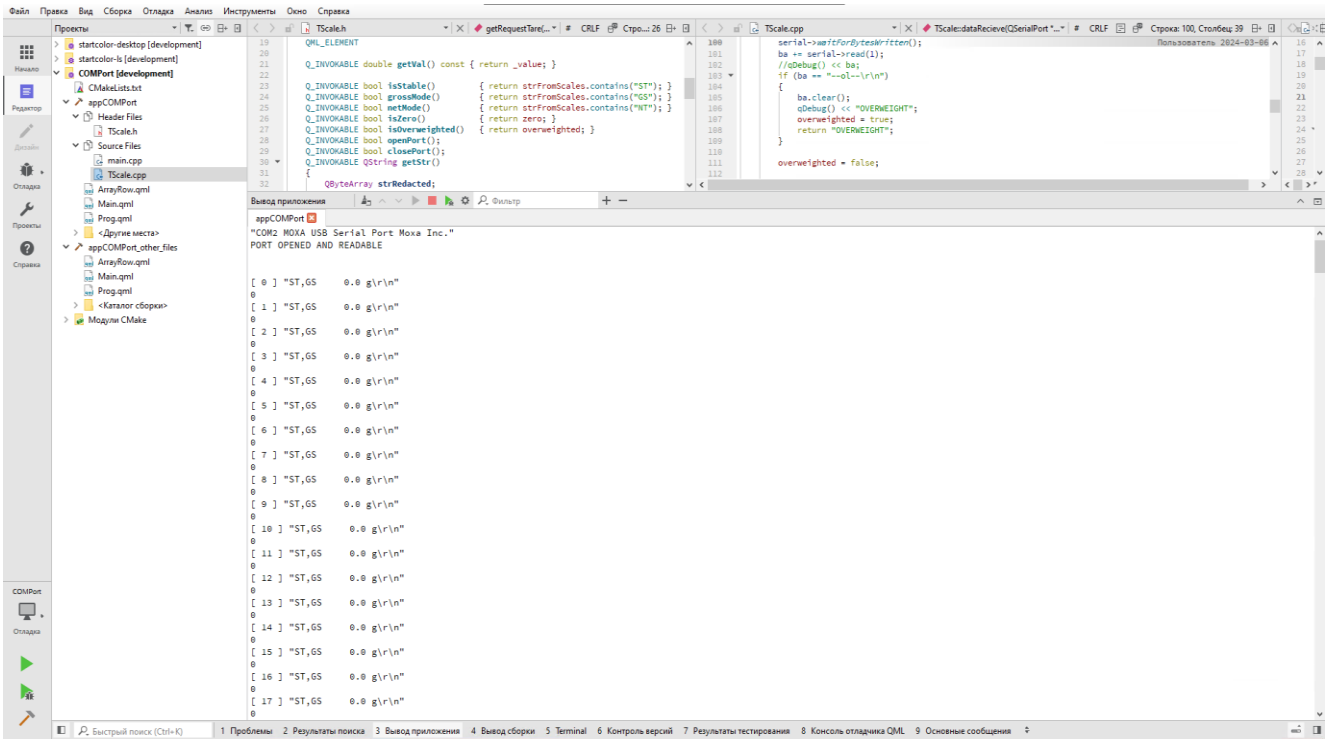

Рисунок 2.7 – Запуск приложения

Как мы видим на рисунке 2.7 приложение успешно запустилось, и мы видим консоль, в которой на первой строчке написано «COM2 MOXA USB Serial Port Moxa Inc.» – это единственный порт, который сейчас подключен к системе. Из-за логики метода Init() на листинге 2.3, в данной строчке вывелось:

1. Название COM-порта.
2. Описание.
3. Производитель.

Мы подключились к данному порту и, как гласит вторая строчка, порт успешно открылся и готов к чтению/записи. Данная строчка появилась из-за работы метода openPort() на листинге 2.4.

Далее, идёт трансляция массива байтов, который мы получаем из-за работы метода recieveData() на листинге 2.6. Для наглядности был добавлен счётчик, который показывает текущую итерацию вывода преобразованного массива с порта.

И, наконец, выводится текущий вес типом *double* – это результат работы метода getValue() на листинге 2.7.

Функции обнуления и смены типа веса проверить проблематично на данном этапе, однако мы сможем проверить данные функции, когда реализуем графический интерфейс – данные функции будут вызываться по кнопкам.

Вывод по разделу 2

В данном разделе занялись постановкой задачи: составили функциональные и нефункциональные требования, а также провели анализ и описание задачи, где объяснили функциональные и нефункциональные требования. Затем выбрали алгоритмы для будущего драйвера и пути решения, составили графические схемы алгоритмов, описали структуру будущего драйвера и приложения для него. Выделили методы, которые будут использоваться в приложении.

По итогу проделанной работы был написан код для драйвера, который успешно заработал. Следующим этапом будет разработка графического интерфейса для проверки работоспособности драйвера.

3. РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ДЛЯ ПРОВЕРКИ РАБОТЫ ДРАЙВЕРА

В данном разделе будет реализован графический интерфейс для проверки работы драйвера. Разработка графического интерфейса нужна для:

а) проверки работоспособности драйвера совместно с графическим интерфейсом;

б) первичного тестирования колористом;

в) получения опыта в разработке приложений с использованием этого языка.

3.1. Создание интерфейса для проверки работоспособности драйвера

Для начала, передадим экземпляр класса TScale в графическую среду QML. Результат на листинге 3.1.

```
// Библиотеки
// #include * * *
int main(int argc, char *argv[])
{
    /*
     * Код, автоматически сгенерированный Qt
    */

    TScale scales;
    engine.rootContext()->setContextProperty("scales", &scales);

    engine.load(url);
    app.exec();
    return 0;
}
```

Перед началом проектирования приложения напишем простой код для проверки работоспособности программы – в нём будет всего 2 кнопки: «Обнулить» и «Тара», а также текстовое поле для транслирования веса. Все эти поля будут расположены в ряд. В листинге 3.2 показан код такого графического интерфейса.

Листинг 3.2 – Реализация простого графического интерфейса

```
// Импорт библиотек
import tscale
import QtQuick
import QtQuick.Controls
// Код приложения
Window { // Основное окно приложения
    id: window width: 800 height: 600
    title: qsTr("Работа с весами производства T-Scale")
    Connections { // Подключения
        target: scales // Берём экземпляр класса «T-Scale»
        onValueChanged: {
            valueText.text = scales.value
        } // Если значение меняется, то меняется и текст
    }
    Row { // Все элементы внутри фигурных скобок будут в ряд
        Button{ // Кнопка обнуления
            text: "ОБНУЛИТЬ"
            onClicked: { // По нажатию делаем запрос на обнуление
                scales.setRequestZero()
            }
        }
        Text { // Вывод веса
            id: valueText
            objectName: "scalesValue"
            text: " "
        }
        Button { // Кнопка смены типа веса
            text: "ТАРА"
            onClicked: { // По нажатию делаем запрос на смену
                scales.setRequestTare()
            }
        } } }
```

В листинге 3.2 у нас есть класс «Window» – основное окно приложения, в самом окне содержатся элементы:

– «Connections» (подключения) – в данном элементе можно определить экземпляр класса, который передаётся из C++ в QML. В нём будут доступны все методы, сигналы и свойства, которые мы прописали ранее для этого класса, и их можно будет вызывать непосредственно из QML;

– «Row» (ряд) – данный элемент располагает дочерние объекты в ряд;

– «Button» (кнопка) – это элемент, на который можно нажать, чтобы выполнилась команда;

– «Text» (текст) – представляет из себя видоизменяемый текст.

Запустим код из листинга 3.2 и увидим результат на рисунке 3.1.

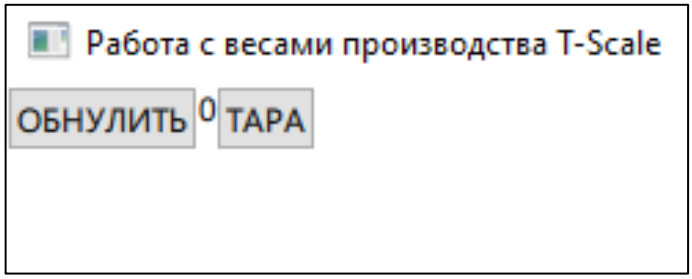

Рисунок 3.1 – Проверочный графический интерфейс

Трансляция в текстовое поле происходит в режиме реального времени, при изменении веса оно динамически меняется. Например, при весе 288.8 грамм программа покажет результат, как показано на рисунке 3.2.

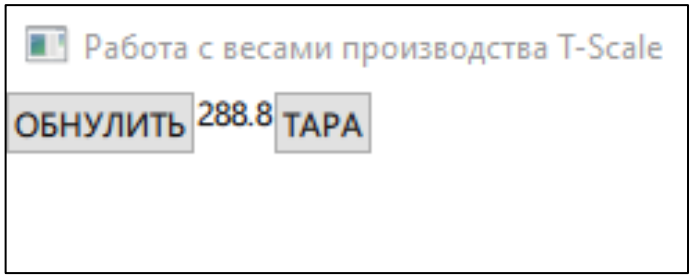

Рисунок 3.2 – Пример работы программы

Можно свериться с консолью, как показано на рисунке 3.3.

Рисунок 3.3 – Сверка работы графического интерфейса с консолью

При нажатии на кнопку «Обнулить» – отправляется команда на обнуление и вес становится равным нулю. При нажатии на кнопку «Тара» – отправляется команда на смену типа веса – весы автоматически обнуляются, а в массиве 4 и 5 байты выводят «NT» вместо «GS». Команды могут исполняться только когда вес стабильный.

3.2. Проектирование интерфейса

Перед началом написания кода для более привлекательного интерфейса спроектируем модель нашего интерфейса, согласно нефункциональным требованиям, а также согласно структуре нашей программы. Модель представлена на рисунке 3.4.

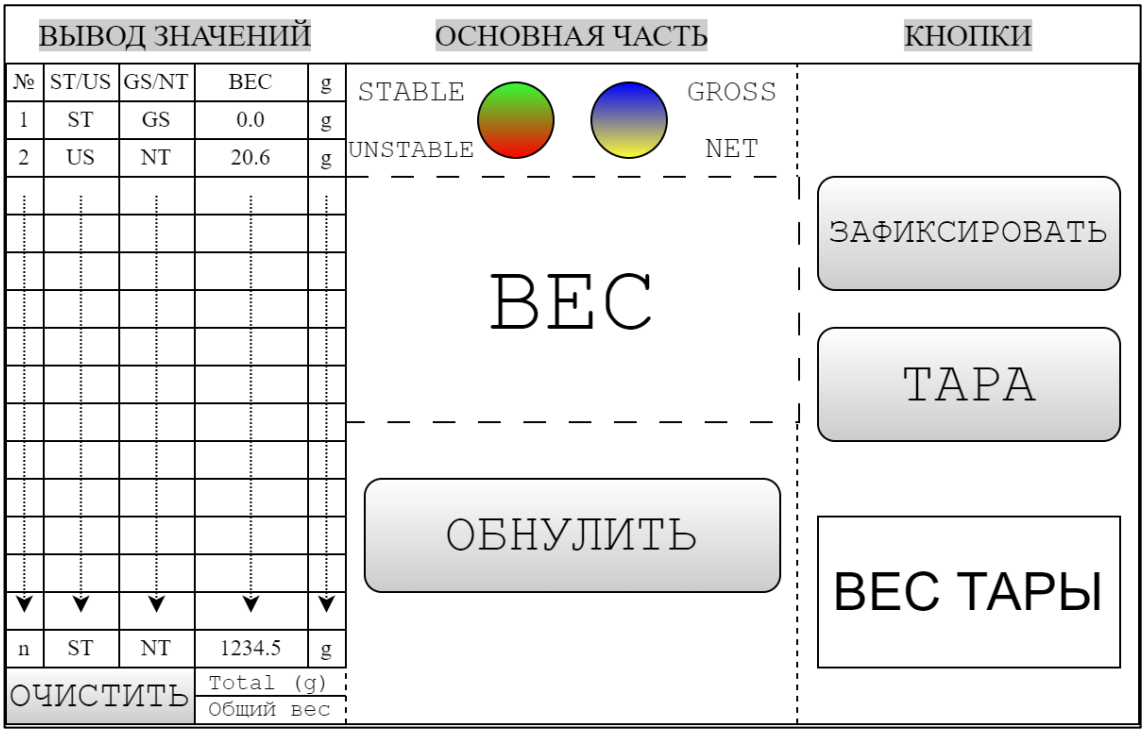

Рисунок 3.4 – проектировка графического интерфейса

3.3. Итоговый вид приложения

На листинге 3.2 продемонстрирован файл Main.qml – основной файл приложения, с помощью которого приложение начинает построение всего графического интерфейса. Внутри этого файла есть элемент Prog – это файл Prog.qml, в котором описана вся логика приложения, в свою очередь Main.qml является некой главной «оболочкой» графического интерфейса, где мы устанавливаем минимальные размеры окна, устанавливаем название приложения и делаем его видимым.

Листинг 3.3 – Содержимое файла Main.qml

```
import tscale
import QtQuick
import QtQuick.Controls
Window {
    id: window
    visible: true
    title: qsTr("Работа с весами производства T-Scale")
    minimumWidth: 1200 // Задали минимальную ширину
    minimumHeight: 700 // Задали минимальную высоту
    width: minimumWidth height: minimumHeight
    // Элемент, в котором происходит вся логика работы приложения
    Prog { }
}
```

Файл Prog.qml представлен в приложении Г. Как видно по листингу Г.1 приложения Г, помимо типов «Connection», «Row», «Button» и «Text», в файле содержатся такие элементы как:

– «Layout» (Макет) – выстраивает элементы в определённой последовательности (в колонку, в ряд, в виде таблицы и т.д.). В коде используются «ColumnLayout» и «RowLayout»;

– «Gradient» (градиент) – создаёт градиент, с помощью «GradientStop» указывается положение первого и второго цвета градиента;

– «Item» (предмет) – является базовым типом и простейшим контейнером;

– «Rectangle» (прямоугольник) – контейнер, у которого можно выставить задний фон, обводку или скругление. Унаследован от «Item»;

– «ListView» (вид «списком») – простой список. Есть свойство «delegate», в котором указывается как именно должен выводится список;

– «MenuSeparator» (разделитель) – обычный разделитель (по сути, это закрашенный прямоугольник серого цвета);

– Label (надпись) – более простая версия типа «Text», не имеет такой гибкой настройки как его более сложная версия.

Более подробная документация по QML типам есть в документации Qt [17], в ней описаны свойства, методы и прочая справочная информация о каждом QML типе.

Запустим проект с кодом из листинга 3.3 совместно с листингом Г.1 приложения Г и смотрим на результат на рисунке 3.5.

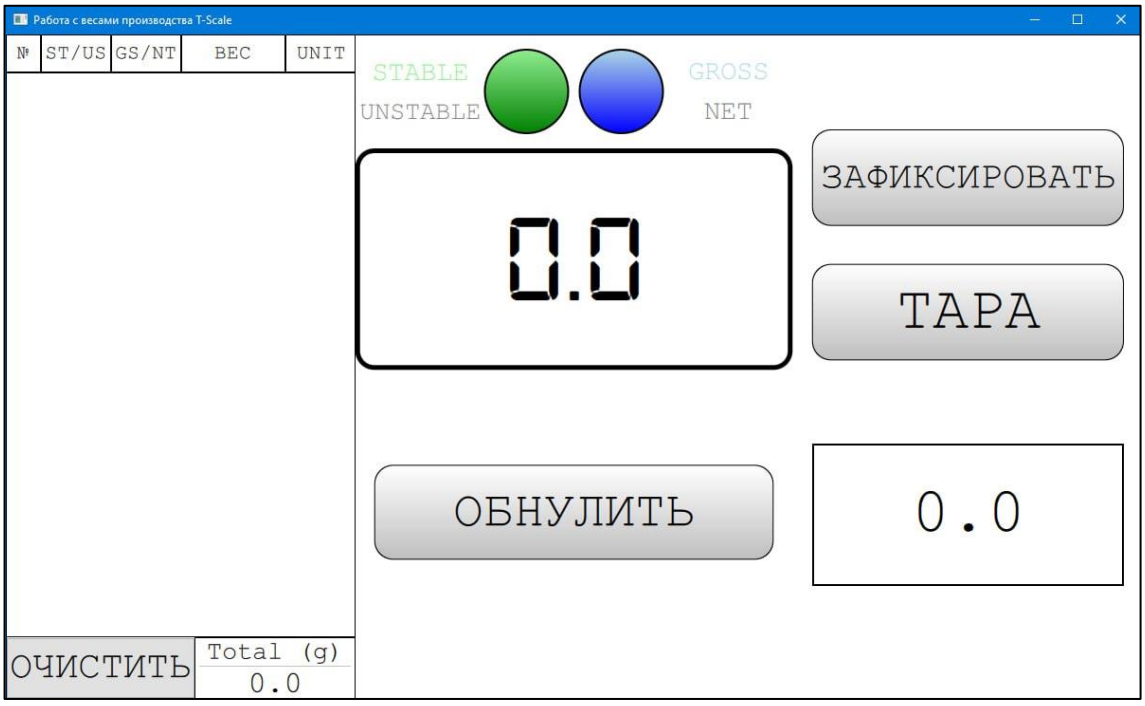

Рисунок 3.5 – Итоговый вид приложения

Как мы видим, благодаря гибкой настройке типов в QML, у нас получилось воспроизвести макет интерфейса, представленный на рисунке 3.4.

Теперь, проверим работоспособность вывода веса в центральной части приложения. Для примера положим на весы предмет произвольного веса. Результат на рисунке 3.6.

Рисунок 3.6 – Вывод произвольного веса с весов в интерфейс

Как мы видим на рисунке 3.6 – вес действительно передаётся. Зафиксируем данный вес нажав на кнопку «ТАРА» (рисунок 3.7).

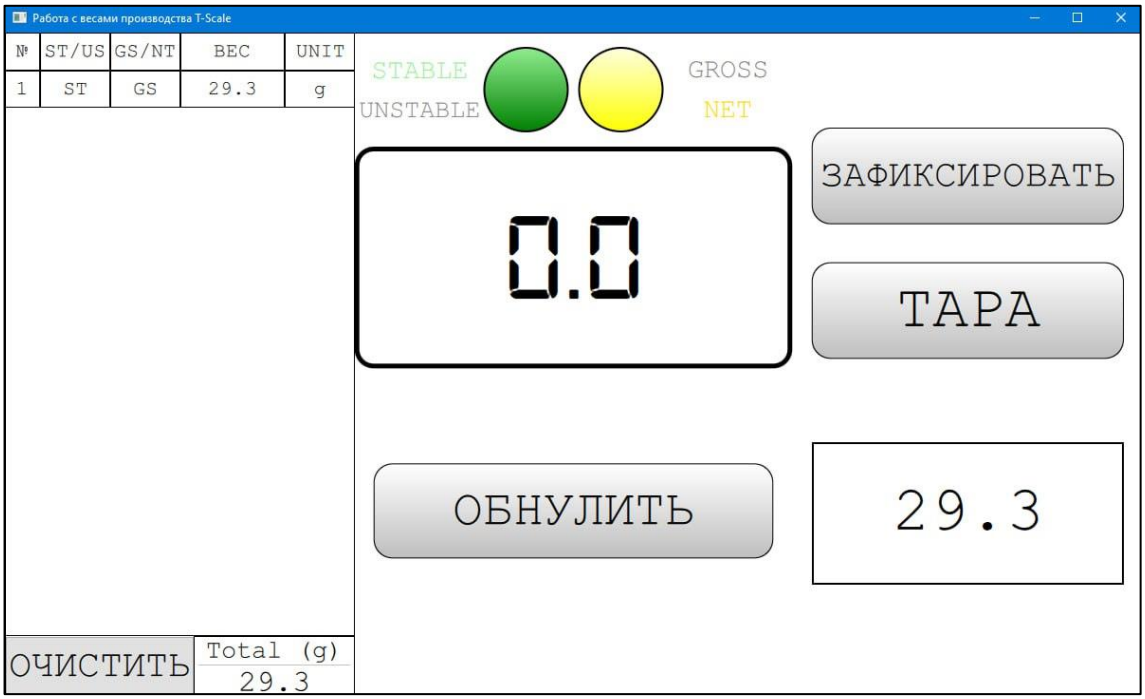

Рисунок 3.7 – Фиксация веса тары

Как видно на рисунке 3.7, при нажатии на кнопку «ТАРА» вес сменился с GS на NT, тем самым вызвав смену индикации с GROSS на NET, цвет при этом сменился с синего на золотистый. В отдельное поле вынесено значение тары.

Попробуем сымитировать работу колориста: допустим, начали проводить инвентаризацию всех пигментов одной окрасочной линии, тогда, «финальная» работа будет выглядеть таким образом, как представлено на рисунке 3.8.

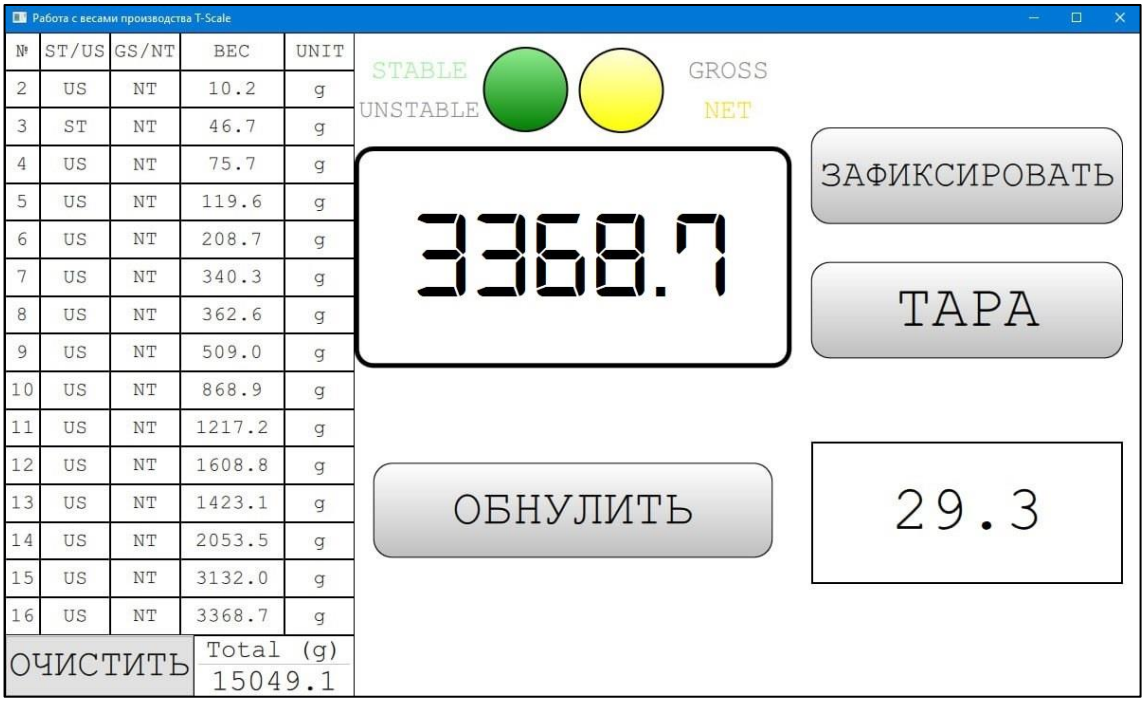

Рисунок 3.8 – Имитация работы колориста

Как мы видим на рисунке 3.8, слева есть столбцы с измеренным весом, когда количество превышает 15, то первый элемент выходит за пределы контейнера, то есть первый видимый элемент теперь с номером 2, а последний с номером 16. Внизу, под строчками, выводится общий вес в граммах.

Такие большие числа в примере взяты для демонстрирования того, что они влезают в прямоугольник с выводом веса, в реальности, при работе, весы редко будут переходить порог в 1 килограмм. При фиксации значения весы *автоматически обнуляются*, именно поэтому общий вес вышел в примерно 15 кг.

Также, стоит упомянуть, что вес нестабилен из-за того, что вес снимался под динамически увеличивающимся весом (для примера). Снимок экрана производился, когда вес уже был стабильным.

Очистим значения и продемонстрируем индикацию нестабильного веса (рисунок 3.9).

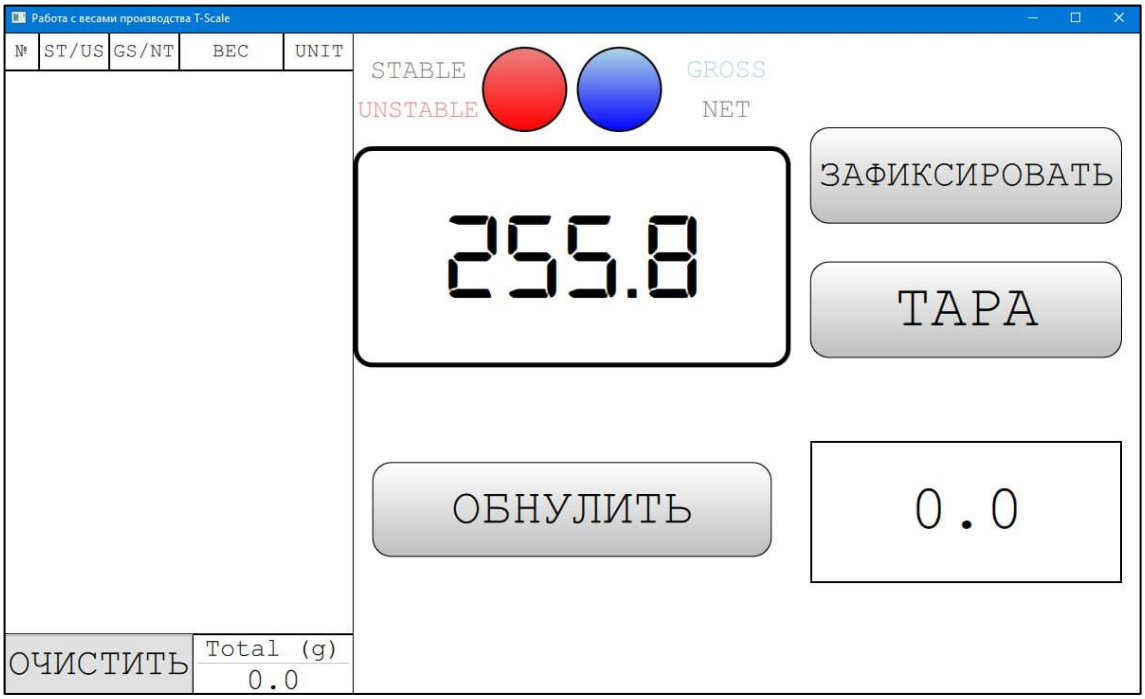

Рисунок 3.9 – Индикация нестабильного веса

Как мы видим на рисунке 3.9, после очистки все данные обнулились, а индикация нестабильного веса работает.

Теперь, проверим индикацию, когда что-то выходит за пределы допустимого: например, когда весы передают не то значение, которое ожидается (не тот массив байтов), как показано на рисунке 3.10.

Ровно таким же способом работает и индикация при перевесе (когда весы передают информацию о том, что обнаружен перевес). Такую ситуацию можно воспроизвести, однако это крайне не рекомендуется, так как если на тензодатчик ставить массу сверх его значений – то может произойти деформация тензодатчика, после чего он может перестать работать корректно.

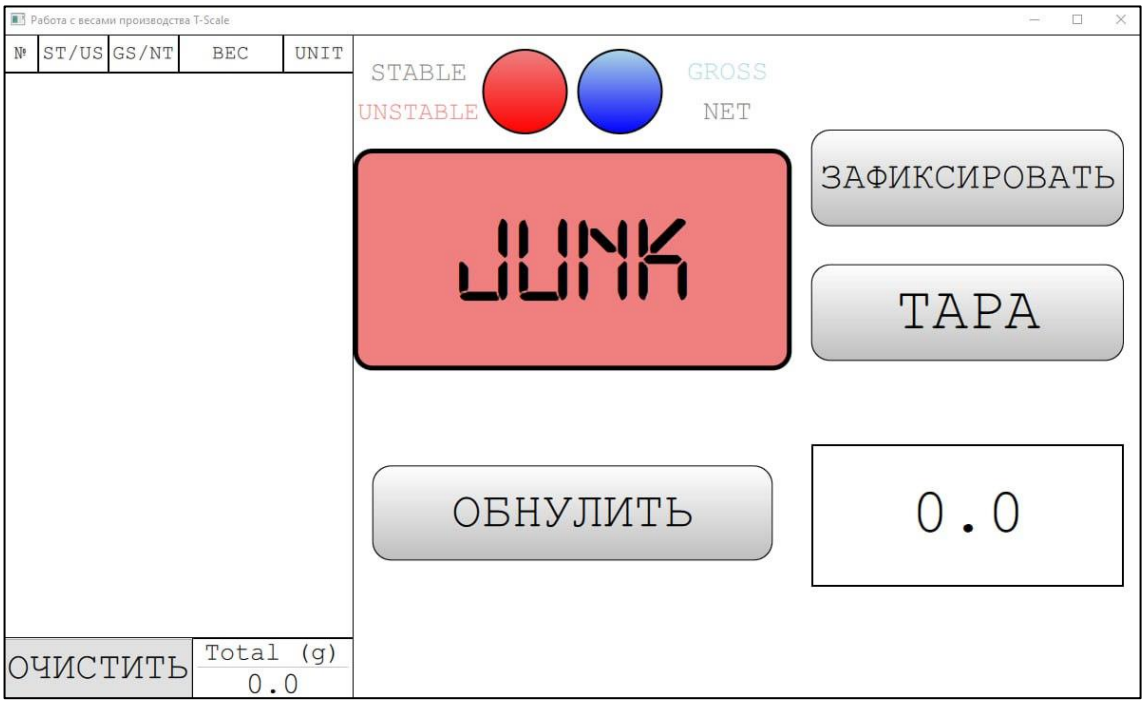

Рисунок 3.10 – Индикация недопустимого результата

Функция обнуления отрабатывает как надо – посылает команду весам на обнуление и, если весы стабильны – то вес обнуляется.

Все данные индикации нужны, чтобы колорист понимал, в каком сейчас состоянии весы (стабильны/нестабильны), знал какой вес сейчас снимается (вес тары или масса нетто) и был предупреждён о недопустимых значениях (вывод недопустимого значения/перевес).

Вывод по разделу 3

В ходе выполнения данного раздела мы создали простое приложение для проверки работоспособности драйвера и спроектировали приложение. Согласно макету спроектированного приложения была написана программа, в которой будет работать колорист для первичной проверки драйвера и весов перед внедрением драйвера в приложение «StartColor».

Также были проверены все индикации, которые нужны колористу для того, чтобы он понимал, какое состояние сейчас имеют весы.

4. ТЕСТИРОВАНИЕ ДРАЙВЕРА

В ходе тестирования мы убедились, что код работает стабильно все функциональные и нефункциональные требования выполнены. Однако, во время проверки возникла ситуация, когда вместо нужного нам набора байтов в консоль изредка выводилось не то, что ожидалось. Экспериментальным путём было выяснено, что при коротком замыкании контактов выводится не тот массив байт, который должен быть.

После устранения данной неполадки весы с программой будут отданы колористу для первичной проверки весов и драйвера и только после успешного тестирования мы можем внедрять драйвер в приложение «StartColor».

4.1. Устранение неполадок, корректировка

Для предостережения пользователя необходимо написать код, который фиксирует, что произошло замыкание контактов. При возникновении данной ситуации будет выводится сообщение на экран: «JUNK», говоря о том, что выводится не ожидаемый набор байт.

Дополним метод recieveData() кодом, представленным на листинге 4.1.

Листинг 4.1 – Добавление условия в метод reciveData() для обработки короткого замыкания

```
// Делаем проверку на непредвиденные байты
if (ba.length() > 17 || (ba.endsWith('\n') && ba.length() < 17))
{
    ba.clear(); // Очищаем буфер
    qDebug() << "JUNK DETECTED"; // Сообщаем в консоль
    _value = -10000.0;
    emit valueChanged(); // Испускаем сигнал
}
```

Как видно на листинге 4.1, мы будем передавать значение –10000 в графический интерфейс, который сигнализирует о том, что зафиксирован не ожидаемый набор байт.

Для проверки мы сымитировали короткое замыкание и код отработал, как и требовалось.

4.2. Фактическое (первичное) тестирование

После окончания работы с драйвером для весов, они были переданы колористу для практической проверки работоспособности.

В ходе тестирования было выявлено, что весы обладают большей чувствительностью, чем аналоги и работать в помещении, где происходит большое количество внешних колебаний, проблематично.

Поправить эту проблему программно не получится, так как данная проблема связана с чувствительностью тензодатчика. Исходя из этого, весы должны использоваться в месте, где нет большого количество колебаний внешней среды.

В целом, если не ставить весы в месте с большим количеством вибраций, то они прошли первичное тестирование и драйвер может быть внедрён в приложение «StartColor».

4.3. Внедрение драйвера

Для начала драйвер был протестирован на тестовой версии приложения, после чего был выпуск версии для пользователей.

В приложении «StartColor» есть настройки с пунктом «Весы», где можно выбрать модель весов (наша модель в данном приложении называется Santint), выбрать порт, откуда передаются данные и настроить параметры последовательного порта. После настройки параметров для весов их вывод можно проверить в специальном поле, а также можно проверить функцию обнуления весов.

Настроем параметры под наши весы. После ввода параметров выведется сообщение «Весы подключены!» в вверху приложения и снизу, в поле правее надписи «Текущий вес» начнётся трансляция веса (рисунок 4.1).

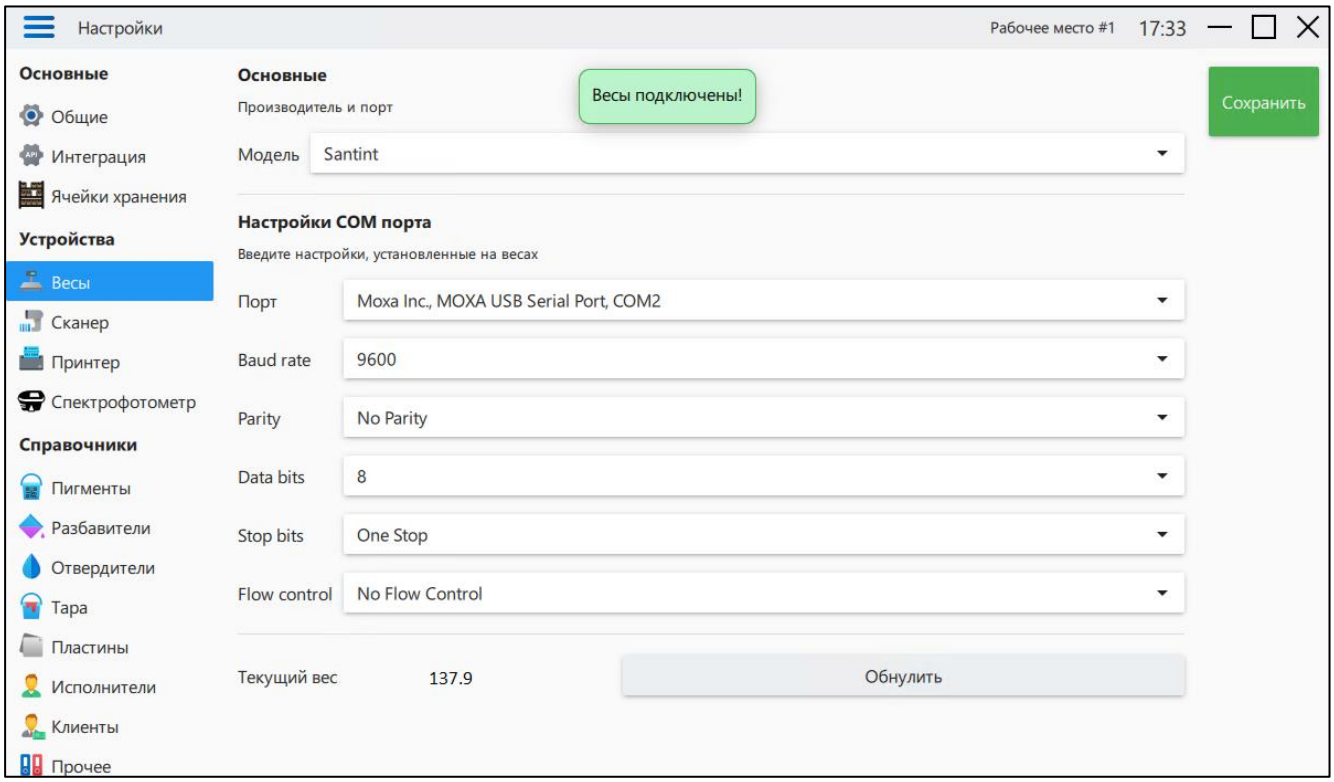

Рисунок 4.1 – Вывод значения весов в программе «StartColor»

Как видно на рисунке 4.1, установлена модель «Santint», имя порта называется «Moxa Inc., MOXA USB Serial Port, COM2», который означает, что передача происходит с помощью преобразователя, введены параметры, согласно таблице 1.5, сверху появилось сообщение о том, что весы подключены и происходит вывод веса в поле правее надписи «Текущий вес».

Таким образом, теперь приложение «StartColor» поддерживает работу весов T-Scale и Santint.

Вывод по разделу 4

В данном разделе мы протестировали драйвер и обнаружили неполадку, которую устранили. Драйвер вместе с весами прошли первичное тестирование, после чего сам драйвер был внедрён в приложение «StartColor». Теперь приложение поддерживает работу весов T-Scale и Santint.

ЗАКЛЮЧЕНИЕ

При анализе предметной области был выявлен тип весов, их класс точности. Был разобран интерфейс RS-232, который есть на весах, преобразователе и проводе. Была описана вся установка: описан внешний вид весов, настройки весов, массив байтов, который выводится при подключении к компьютеру, команды, которые принимают весы, а также, помимо весов, описаны ещё преобразователь с RS-232 на USB и кабель, соединяющий весы и преобразователь. Были выявлены параметры последовательного подключения, чтобы подключить всю установку воедино.

Далее были описаны используемые средства разработки: язык C++, интегрированная среда разработки Qt Creator совместно с фреймворком Qt Framework и модулем QtSerialPort, а также описан декларативный язык программирования QML. Все данные средства разработки стали фундаментом выполнения работы.

В ходе разработки были выявлены функциональные и нефункциональные требования к приложению, проведён анализ поставленной задачи и выбраны алгоритмы и пути решения задачи. Была описана структура программы, которая стала фундаментом написания кода для драйвера высокоточных весов.

Затем был разработан простейший графический интерфейс для проверки работоспособности драйвера совместно с ним. Была спроектирована модель приложения, которая стала основой для написания приложения для драйвера, чтобы затем колорист смог проверить драйвер совместно с весами.

После всех вышеперечисленных действий мы произвели тестирование драйвера, сделали необходимые корректировки. Приложение для весов было передано колористу для первичного тестирования драйвера совместно с весами.

Драйвер и весы в ходе первичного тестирования показали себя хорошо, однако был обнаружен недостаток в виде чувствительного тензодатчика. Исправить данный недостаток программно не представляется возможным. Для

работы с такими весами нужно выбирать среду, где на них меньше всего воздействуют внешние вибрации.

Драйвер был успешно внедрён в приложение «StartColor» и теперь, на данный момент, приложение поддерживает работу трёх моделей весов, драйвер одной из этих моделей был сделан в ходе данной работы.

Сейчас весы компании Santint модели ES7000 находятся у партнёра компании на практическом пользовании, совместно с другим оборудованием для рабочего места колориста.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Простов, С. М. Основы и методология научных исследований: учебное пособие / С. М. Простов. — Кемерово: КузГТУ имени Т.Ф. Горбачева, 2022. — ISBN 978-5-00137-299-8. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/257579 (дата обращения: 26.02.2024). — Режим доступа: для авториз. пользователей.

2. Международная рекомендация OIML R76-1 «Part 1: Non-automatic weighing instruments». URL: https://www.oiml.org/en/files/pdf_r/r076-1-e06.pdf (дата обращения: 01.03.2024).

3. ГОСТ OIML R 76-1 – 2011 «Весы неавтоматического действия. Часть 1». URL: https://mas.center/upload/medialibrary/174/GOST-OIML-R-76_1_2011.pdf (дата обращения: 01.03.2024).

4. Техническое руководство весов «T-Scale ROW Precision Balance Series». URL: https://guidessimo.com/document/2319031/t-scale-row-precision-balance-series-technical-manual-30.html (дата обращения 16.01.2024).

5. Глуханов, А. А. Методы и средства измерений, испытаний и контроля : учебное пособие / А. А. Глуханов. — Архангельск : САФУ, 2020. — 188 с. — ISBN 978-5-261-01462-1. — Текст: электронный// Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/226832 (дата обращения: 28.02.2024). — Режим доступа: для авториз. пользователей.

6. Страница на сайте MOXA с UPort1100: офиц. сайт. URL: https://www.moxa.com/en/products/industrial-edge-connectivity/usb-to-serial-converters-usb-hubs/usb-to-serial-converters/uport-1100-series#resources (дата обращения 17.01.2024).

7. Руководство пользователя адаптера MOXA серии UPort1100. URL: https://www.moxa.com/getmedia/a2924269-6076-4c8f-9c1e-7268e235dde1/moxa-uport-1100-series-manual-v9.0.pdf (дата обращения: 17.01.2024).

8. Qt Creator: офиц. сайт. URL: https://www.qt.io/product/development-tools (дата обращения: 27.12.2023).

9. Qt: фреймворк для C++: офиц. сайт. URL: https://www.qt.io/product/framework (дата обращения: 27.12.2023).

10. Разработка графического интерфейса пользователя информационной системы с использованием библиотеки Qt : учебное пособие / Ю. В. Минин, А. И. Елисеев, В. В. Алексеев, Ю. А. Губсков. — Тамбов : ТГТУ, 2021. — 84 с. — ISBN 978-5-8265-2397-1.— Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/320516 (дата обращения: 27.12.2023). — Режим доступа: для авториз. пользователей.

11. Модуль для работы с последовательными портами с использованием Qt фреймворка, взятой с официального сайта с документацией по Qt: офиц. сайт. URL: https://wiki.qt.io/Qt_Serial_Port (дата обращения: 20.01.2024).

12. Документация по классу QSerialPort: офиц. сайт. URL: https://doc.qt.io/qt-5/qserialport.html (дата обращения: 20.01.2024).

13. Документация по классу QSerialPortInfo: офиц. сайт. URL: https://doc.qt.io/qt-5/qserialportinfo.html (20.01.2024).

14. Топольский, Д. В. Программирование на Ассемблере: учеб. пособие по направлению «Програм. инженерия» / Д. В. Топольский, И. Г. Топольская; Юж.-Урал. гос. ун-т, Нижневартов. фил., Каф. Общепроф. и спец. дисциплины по юриспруденции; ЮУрГУ. – Екатеринбург: ФОРТ ДИАЛОГ-Исеть, 2016. – 63 с.: схем (дата обращения: 07.03.2024).

15. Сайт Cmake: офиц. сайт. URL: https://cmake.org/ (дата обращения: 15.01.2024).

16. Документация о сигналах и слотах в библиотеке Qt. Офиц.сайт. URL: https://doc.qt.io/qt-6/signalsandslots.html (дата обращения 25.01.2024).

17. Документация по всем типам QML. Офиц. сайт. URL: https://doc.qt.io/qt-6/qmltypes.html (дата обращения: 01.03.2024).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ВЕСОВ

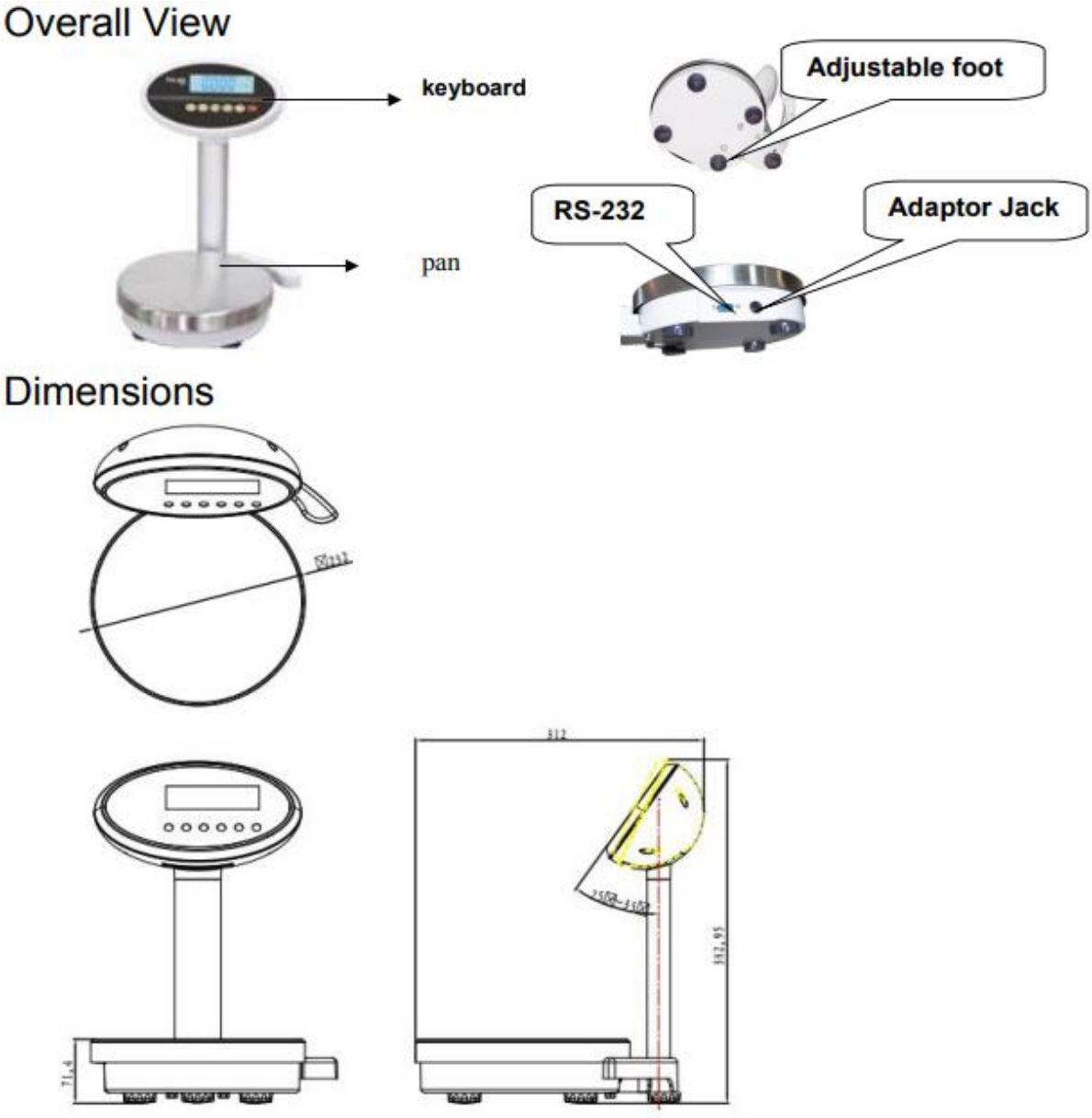

Рисунок А.1 – Внешний вид весов

Изображение взято из технического руководства весов T-Scale [4, с. 3].

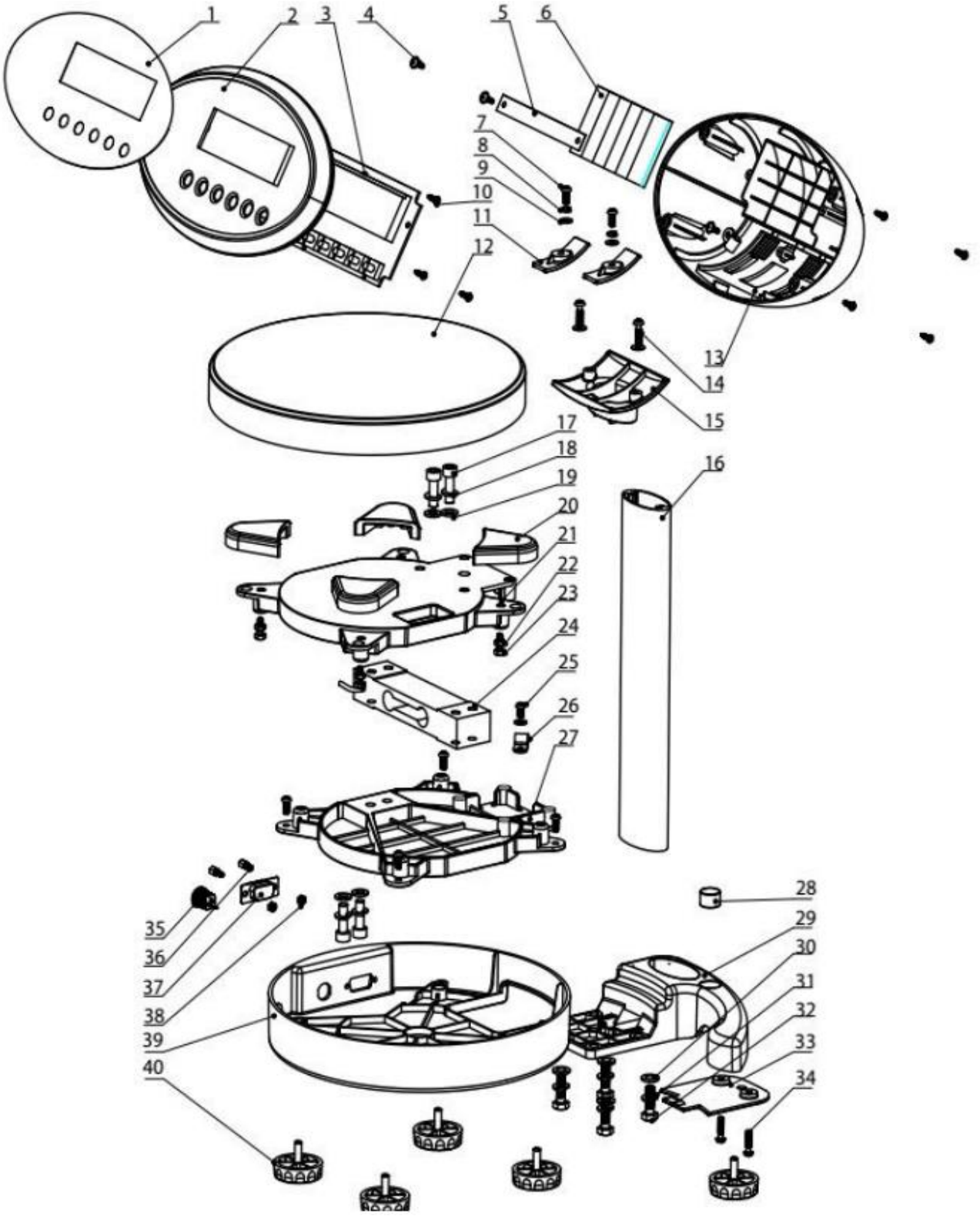

Рисунок А.2 – Внутренняя компоновка весов

Изображение взято из технического руководства весов T-Scale [4, с. 26].

S.No	Name	Pcs	Materials	Remarks
40	NB Foot	5	PP+ Carbon Steel	Gray
39	ROW-Frame housing	1	ABS	
38	D Type nut M2.5	2		
37	DB9 Socket -9S	1		
36	D Type Screw M2.5	2		
35	Adaptor Jack - Round	1		
34	M4x16 Round Head Screw	2	SS	
33	Rod Holder Cover	1	ABS	
32	M6*20 Hex Screw	4	SS	
31	M6 Spring Washer	4	SS	
30	Flat Washer φ6	4		
29	Rod-Holder	1	AL	
28	Level Bubble φ14.7	1		
27	Pan Recepticle Bottom	1	AL	
26	Cable tie	2	PVC	
25	M4x8 Round Head Screw	1		F
24	Load Cell	1	AL	X6-10KG-0.85m
23	M4*12 Hex Screw	4		Over load Protection
22	M4 Nut	4		Over load Protection
21	Pan Receipticle	1	AL	
20	Bush -Pan	4	NBR	Pan Support
19	φ6 Flat Washer	4		
18	M6 Spring Washer	4	65Mn	Load Cell Secure
17	M6x20 Allen Screw	4		
16	Pole- Aluminum	1	AL	
15	ROW- Head Folder	1	ABS	
14	M4x16 Star Screw	2		
13	Indicator Rear Cover	1	ABS	
12	ROW- Pan	1	SUS304	
11	ROW-Transfer Layer	2		
10	M3x8 Star- Screw	8		Self Thread
9	φ4 Flat Gasket	5		
8	M4 Spring Washer	2	65Mn	
7	M4x10 Head Screws	6		
6	Battery	1	Ni-MH	7.2V/2000mAH
5	Battery Clamp	1	PVC	
4	M3x8 Self Thread Screw	3		
3	PCB	1		
2	Indicator Front Cover	1	ABS	
1	Overlay	1		

Рисунок А.3 – Обозначения элементов весов

Изображение взято из технического руководства весов T-Scale [4, с. 27].

Таблица А.1 – Обозначение индикаций на весах T-Scale

Индикация	**Значение индикации**
HI **OK** **LOW**	Проверка веса (индикация работает только при настройке минимального и максимального значения в настройках)
ZERO	Загорается, если значение веса равно нулю
TARE	Индикация тары
GROSS	Загорается, если взвешивается масса брутто
NET	Загорается, если взвешивается масса нетто
STABLE	Загорается, если вес стабилен (должно пройти небольшое количество времени, при котором весы не поддавались колебаниям и вес оставался идентичным)
AUTO	Загорается, когда в настройках включен режим P Auto (подробнее в таблице 3)
M+	Загорается, если в аккумуляторе есть значение
ANIMAL	Загорается, когда включён режим «ANIMAL mode» в настройках
HOLD	Загорается, когда включён режим «HOLD mode» в настройках
	Показание состояния аккумулятора (разряжен/требуется зарядка/заряжен)

Таблица взята из технического руководства весов T-Scale [4, с. 8].

Таблица А.2 – Функции кнопок на весах T-Scale

Кнопка	**При взвешивании**	**В режиме настройки**
ON/OFF	Включение/выключение весов	
ZERO	Сброс в ноль	Подтверждение действия
TARE	Выбор массы нетто/брутто	Листать настройки, либо увеличивать значение
G/N	Переключение между массой нетто/брутто	Перемещает относительно активной цифры вправо
M+	Записать в аккумулятор, если функция аккумулирования не автоматическая. При нажатии также передаёт значение на компьютер или принтер через интерфейс RS-232	Очищает активную цифру
UNIT	Переключает единицы измерения веса	Используется в качестве кнопки «назад», либо перемещает относительно активной цифры влево

Таблица взята из технического руководства весов T-Scale [4, с. 9].

ПРИЛОЖЕНИЕ Б

ЗНАЧЕНИЕ ИНДИКАЦИЙ НА ПРЕОБРАЗОВАТЕЛЕ

Таблица Б.1 – Значение индикаций на преобразователе MOXA модели UPort 1100

Название	Цвет	Описание
Active	Красный	Индикация показывает, работает ли преобразователь. Постоянный: порт работает Выключен: есть неисправность (проблема с преобразователем, установкой драйвера или конфигурацией ПК) или выключен.
TxD	Зелёный	Мерцающий: идёт отправка данных в последовательно устройство.
RxD	Жёлтый	Мерцающий: идёт чтение данных из последовательного устройства.

Таблица взята из руководства пользователя адаптера MOXA серии UPort1100 [7, с. 8].

ПРИЛОЖЕНИЕ В

КОД ДРАЙВЕРА ВЕСОВ

Листинг В.1 – Содержимое файла T-Scale.h

```
#ifndef TSCALE_H
#define TSCALE_H

#include "qqmlintegration.h"
#include <QObject>
#include <QtSerialPort/QSerialPort>
#include <QtSerialPort/QSerialPortInfo>
#include <QDebug>
#include <QString>

class TScale : public QObject
{
    Q_OBJECT
public:
    explicit TScale(QObject *parent = nullptr);
    ~TScale();
    QML_ELEMENT
    // Получение private переменной _value
    Q_INVOKABLE double getVal() const { return _value; }
    // Метод, который определяет, стабильны ли весы
    Q_INVOKABLE bool isStable()    { return strFromScales.contains("ST"); }
    // Метод, который определяет, весы в типе веса тары или нет
    Q_INVOKABLE bool grossMode()   { return strFromScales.contains("GS"); }
    // Метод, который определяет, весы в типе веса нетто или нет
    Q_INVOKABLE bool netMode()     { return strFromScales.contains("NT"); }
    // Метод, который определяет, весы обнулены или нет
    Q_INVOKABLE bool isZero()      { return zero; }

    Q_INVOKABLE bool openPort(); // Метод, открывающий порт
    Q_INVOKABLE bool closePort(); // Метод, закрывающий порт

    // Метод, меняющий запрос на обнуление
    Q_INVOKABLE void setRequestZero() { requestToZero = !requestToZero; }
    // Метод, меняющий запрос на смену типа тары
    Q_INVOKABLE void setRequestTare() { requestToTare = !requestToTare; }
    // Свойство "value", которое мы передаём в QML, меняющаяся по сигналу

    // "valueChanged()" (вызывается метод getVal() на чтение переменной)
    Q_PROPERTY(double value READ getVal NOTIFY valueChanged);

    // Геттер для requestToZero
    bool getRequestZero() { return requestToZero; }
    // Геттер для requestToZero
    bool getRequestTare() { return requestToTare; }

signals:
    void valueChanged(); // Сигнал, который вызываем при смене значения
```

```
private:
    // Переменная для метода disconnect
    QMetaObject::Connection _connectionToValue;

    // ПЕРЕМЕННЫЕ
    QSerialPort *serial;
    QByteArray strFromScales;
    double _value;
    bool overweighted = false;
    bool inited = false;
    bool zero = true;
    bool requestToZero = false;
    bool requestToTare = false;

    void Init(); // Метод инициализации
    // Метод, проверяющий, открыт ли порт
    bool portIsOpen() { return serial->isOpen(); }

    QByteArray dataRecieve(); // Метод преобразования строки байтов
    int dataZero(); // Метод, обнуляющий весы
    int dataTare(); // Метод, меняющий тип веса

public slots:
    // Слот, берущий значение с весов по сигналу QSerialPort::readyRead()
    // Q_INVOCABLE означает, что этот метод можно вызвать в QML
    Q_INVOKABLE double getValue();
};
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ГРАФИЧЕСКОГО ИНТЕРФЕЙСА В QML

Листинг Г.1 – файл Prog.qml

```
// Библиотеки
import tscale
import QtQuick
import QtQuick.Controls
import QtQuick.Layouts

Item {
  property int id: 0
  property double totalWeight: 0.0
  property int coefficientHeight: (window.height - 700) * 0.015
  property int coefficientWidth: (window.width - 1200) * 0.025

  // Подключения

  Connections {
    target: scales
    onValueChanged:
    {
      // Проверки
      if (scales.value < -7001) {
        outputValue.text = "JUNK"
        output.color = "lightcoral"
      } else if (scales.isOverweighted()) {
        outputValue.text = "OVERWEIGHT"
        output.color = "lightcoral"
      } else {
        output.color = "white"
        scales.value % 1 == 0 ? outputValue.text = scales.value + ".0"
                    : outputValue.text = scales.value
      }

      if (scales.isStable()) {
        colorIndicatorStable.gradient = gradGreen
        indicatorStableText.color = "lightgreen"
        indicatorUnstableText.color = "gray"
      } else {
        colorIndicatorStable.gradient = gradRed
        indicatorUnstableText.color = "lightcoral"
        indicatorStableText.color = "gray"
      }

      if (scales.grossMode()) {
        colorIndicatorGrossNet.gradient = gradBlue
        indicatorGrossText.color = "lightblue"
        indicatorNetText.color = "grey"
      } else {
        colorIndicatorGrossNet.gradient = gradYellow
        indicatorNetText.color = "gold"
```

```
      indicatorGrossText.color = "grey"
    }
  }
}

// Градиенты для индикаций

Gradient {
  id: gradRed
  GradientStop { position: 0.0; color: "lightcoral" }
  GradientStop { position: 1.0; color: "red" }
}

Gradient {
  id: gradGreen
  GradientStop { position: 0.0; color: "lightgreen" }
  GradientStop { position: 1.0; color: "green" }
}

Gradient {
  id: gradYellow
  GradientStop { position: 0.0; color: "lightyellow" }
  GradientStop { position: 1.0; color: "yellow" }
}

Gradient {
  id: gradBlue
  GradientStop { position: 0.0; color: "lightblue" }
  GradientStop { position: 1.0; color: "blue" }
}

Rectangle {
  id: root
  width: window.width
  height: window.height
  border.color: "black"

  // Левая часть | ARRAY/CLEAR

  Item {
    id: score
    width: root.width / 13 * 4
    height: root.height
    anchors.left: root.left

    ListModel {
      id: arrayModel

      onCountChanged:
      {
        if(count === 16)
```

```
    {
      arrayModel.remove(0)
    }
  }
}

Row {
  width: parent.width
  height: arrayField.height / 15
  Rectangle {
    border.color: "black"
    width: parent.width / 10
    height: parent.height
    Text {
      anchors.centerIn: parent
      text: "№"
      font.pointSize: 16
      font.family: "Courier New"
    }
  }

  Rectangle {
    border.color: "black"
    width: parent.width / 5
    height: parent.height
    Text {
      anchors.centerIn: parent
      text: "ST/US"
      font.pointSize: 16
      font.family: "Courier New"
    }
  }

  Rectangle {
    border.color: "black"
    width: parent.width / 5
    height: parent.height
    Text {
      anchors.centerIn: parent
      text: "GS/NT"
      font.pointSize: 16
      font.family: "Courier New"
    }
  }

  Rectangle {
    border.color: "black"
    width: parent.width / 10 * 3
    height: parent.height
    Text {
      anchors.centerIn: parent
      text: "BEC"
```

```
      font.pointSize: 16
      font.family: "Courier New"
    }
  }

  Rectangle {
    border.color: "black"
    width: parent.width / 5
    height: parent.height
    Text {
      anchors.centerIn: parent
      text: "UNIT"
      font.pointSize: 16
      font.family: "Courier New"
    }
  }
}

Rectangle {
  id: arrayField
  width: parent.width
  height: parent.height * 0.85
  border.color: "black"
  border.width: 1
  anchors.top: score.top
  anchors.topMargin: arrayField.height / 15
  ListView {
    id: listView
    width: parent.width
    height: parent.height
    anchors.fill: parent
    model: arrayModel
    delegate: Row {
      Rectangle {
        id: idColumn
        border.color: "black"
        width: arrayField.width / 10
        height: arrayField.height / 15
        Text {
          id: idText
          anchors.centerIn: parent
          text: id
          font.pointSize: 16
          font.family: "Courier New"
        }
      }
      Rectangle {
        id: statusColumn
        border.color: "black"
        width: arrayField.width / 5
        height: arrayField.height / 15
        Text {
```

```
            id: statusText
            anchors.centerIn: parent
            text: status
            font.pointSize: 16
            font.family: "Courier New"
          }
        }
        Rectangle {
          id: tareTypeColumn
          border.color: "black"
          width: arrayField.width / 5
          height: arrayField.height / 15
          Text {
            id: tareTypeText
            anchors.centerIn: parent
            text: type
            font.pointSize: 16
            font.family: "Courier New"
          }
        }
        Rectangle {
          id: weightColumn
          border.color: "black"
          width: arrayField.width / 10 * 3
          height: arrayField.height / 15
          Text {
            id: weightText
            anchors.centerIn: parent
            text: weight % 1 === 0 ? weight + ".0" : weight
            font.pointSize: 16
            font.family: "Courier New"
          }
        }
        Rectangle {
          id: unitColumn
          border.color: "black"
          width: arrayField.width / 5
          height: arrayField.height / 15
          Text {
            id: unitText
            anchors.centerIn: parent
            text: unit
            font.pointSize: 16
            font.family: "Courier New"
          }
        }
      }
    }
  }

  RowLayout {
    anchors.top: arrayField.bottom
```

```
    anchors.left: parent.left
    anchors.right: parent.right
    anchors.bottom: parent.bottom

    spacing: -1

    Button {
      Layout.fillHeight: true
      width: idColumn.width + statusColumn.width + tareTypeColumn.width

      text: qsTr("ОЧИСТИТЬ")
      font.pointSize: 30
      font.family: "Courier New"
      onClicked: {
        arrayModel.clear()
        id = 0
      }
    }

    Rectangle {
      id: totalAndClearArea
      Layout.fillWidth: true
      Layout.fillHeight: true
      border.color: "black"
      ColumnLayout {
        anchors.fill: parent
        spacing: -1

        Label {
          Layout.fillWidth: true
          text: "Total (g)"
          font.family: "Courier New"
          font.pointSize: 20
          horizontalAlignment: Text.AlignHCenter
        }

        MenuSeparator {
          Layout.fillWidth: true
        }

        Text {
          id: totalWeightField
          Layout.fillWidth: true
          text: "0.0"
          font.family: "Courier New"
          font.pointSize: 24
          horizontalAlignment: Text.AlignHCenter
        }
      }
    }
  }
}
```

```
// Середина | OUTPUT/ZERO

Item {
  id: main
  width: root.width / 13 * 5
  height: root.height
  anchors.centerIn: root

  //1 Индикации

  Item {
    id: indicators
    anchors.top: main.top
    anchors.bottom: output.top
    height: main.height / 7 + 20
    width: main.width

    //1.1 Индикации STABLE/UNSTABLE

    Item {
      id: indicatorStable

      width: parent.width / 2 - 5
      height: parent.height

      anchors.bottom: parent.bottom
      anchors.left: parent.left

      // 1.1.1 Круглая индикация STABLE/UNSTABLE

      Rectangle {
        id: colorIndicatorStable
        border.color: "black"
        border.width: 2
        width: height
        height: parent.height * 0.75
        anchors.right: parent.right
        anchors.verticalCenter:  parent.verticalCenter
        radius: 90
        PropertyAnimation {
          target: colorIndicatorStable
          property: "gradient"
          duration: 1000
          easing.type: Easing.InOutQuad
        }
      }

      //1.1.2 Надписи STABLE/UNSTABLE

      Item {
        id: indicatorStableField
```

```
    width: parent.width - colorIndicatorStable.width
    height: parent.height

    anchors.top: parent.top
    anchors.right: colorIndicatorStable.left

    ColumnLayout {
      spacing: 10

      anchors.centerIn: parent

      // 1.1.2.1 Надпись STABLE

      Text {
        id: indicatorStableText
        text: qsTr("STABLE")
        Layout.alignment: Qt.AlignHCenter
        Layout.maximumWidth: indicatorStableField.width
        font
        {
          family: "Courier New"
          pointSize: (window.width < 1700) ?
                   22 -coefficientHeight + coefficientWidth
                 : 30
        }
        color: "gray"
      }

      // 1.1.2.2 Надпись UNSTABLE

      Text {
        id: indicatorUnstableText
        text: qsTr("UNSTABLE")
        Layout.alignment: Qt.AlignHCenter
        Layout.maximumWidth: indicatorStableField.width
        font
        {
          family: "Courier New"
          pointSize: (window.width < 1700) ?
                   20 - coefficientHeight + coefficientWidth
                 : 28
        }
        color: "gray"
      }
    }
  }
}

//1.2 Индикации GROSS/NET

Item {
  id: indicatorGrossNet
```

```
width: parent.width / 2 - 5
height: parent.height
anchors.top: parent.top
anchors.bottom: parent.bottom
anchors.right: parent.right
// 1.2.1 Круглая индикация GROSS/NET

Rectangle {
  id: colorIndicatorGrossNet
  border.color: "black"
  border.width: 2
  width: height
  height: parent.height * 0.75
  anchors.left: parent.left
  anchors.verticalCenter:  parent.verticalCenter

  radius: 90
}

// 1.2.2 Надписи GROSS/NET

Item {
  id: indicatorGrossNetField
  width: parent.width - colorIndicatorGrossNet.width
  height: parent.height
  anchors.top: parent.top
  anchors.left: colorIndicatorGrossNet.right

  ColumnLayout {
    spacing: 10
    anchors.centerIn: parent

    // 1.2.2.1 Надпись GROSS

    Text {
      id: indicatorGrossText
      text: qsTr("GROSS")
      Layout.alignment: Qt.AlignHCenter
      Layout.maximumWidth: indicatorGrossNetField.width
      font {
        family: "Courier New"
        pointSize: (window.width < 1700) ?
                  22 - coefficientWidth + coefficientHeight
                : 30
      }
      color: "gray"
    }

    // 1.2.2.2 Надпись NET

    Text {
      id: indicatorNetText
```

```
          text: qsTr("NET")
          Layout.alignment: Qt.AlignHCenter
          font {
            family: "Courier New"
            pointSize: (window.width < 1700) ?
                     22 - coefficientWidth + coefficientHeight
                   : 30
          }
          color: "gray"
        }
      }
    }
  }
}

//Вывод с весов

Rectangle {
  id: output
  width: main.width
  height: main.height / 3
  anchors.top: indicators.bottom
  border.width: 5
  border.color: "black"
  radius: 20

  Text {
    id: outputValue
    objectName: "scalesValue"
    anchors.centerIn: parent
    text: scales?.isOpen() ? " " : "Port is not open!"
    font {
      family: "Digital-7"
      pointSize: 100
    }
    color: "black"
  }
}

// Обнуление ZERO

Button {
  id: buttonZero
  width: main.width - 40
  height: main.height / 7
  anchors.top: output.bottom
  anchors.topMargin: main.height / 7
  anchors.horizontalCenter: parent.horizontalCenter
  hoverEnabled: false
  onClicked: {
    scales.setRequestZero()
  }
```

```
    background: Rectangle {
      id: rectZero
      anchors.fill: parent
      border.width: 1
      border.color: "black"
      radius: 20
      Gradient {
        id: gradZero1
        GradientStop { position: 0.0; color: "white" }
        GradientStop { position: 1.0; color: "#80808080"}
      }
      Gradient {
        id: gradZero2
        GradientStop { position: 1.0; color: "white" }
        GradientStop { position: 0.0; color: "#80808080"}
      }
      gradient: buttonZero.down ? gradZero2 : gradZero1
      Text {
        id: textZero
        text: qsTr("ОБНУЛИТЬ")
        anchors.centerIn: rectZero
        font
        {
          family: "Courier New"
          pointSize: 40
        }
        color: "black"
      }
    }
  }
}

// Правая часть | Операции FIX/TARE

Item {
  id: oper
  width: root.width / 13 * 4
  height: root.height
  anchors.right: root.right

  Item {
    id: operButtons
    width: parent.width
    height: parent.height / 3 + 10
    anchors.top: parent.top
    anchors.topMargin: parent.height / 7

    Button {
      id: buttonFix
      width: parent.width - 40
      height: (parent.height / 2) - 20
      anchors.top: parent.top
```

```
    anchors.horizontalCenter: operButtons.horizontalCenter
    hoverEnabled: false
    onClicked: {
      id++
      arrayModel.append({id: id,
                status: scales.isStable() ? "ST" : "US",
                type: scales.grossMode() ? "GS" : "NT",
                weight: scales.getValue(),
                unit: "g"}
                )
      totalWeight += scales.getValue()
      totalWeightField.text = totalWeight.toFixed(1) % 1 === 0 ?
                              totalWeight.toFixed(1) + ".0"
                             : totalWeight.toFixed(1)
      scales.setRequestZero()
    }
    background: Rectangle {
      id: rectFix
      anchors.fill: parent
      border.width: 1
      border.color: "black"
      radius: 20
      Gradient  {
        id: gradFix1
        GradientStop { position: 0.0; color: "white" }
        GradientStop { position: 1.0; color: "#80808080"}
      }
      Gradient {
        id: gradFix2
        GradientStop { position: 1.0; color: "white" }
        GradientStop { position: 0.0; color: "#80808080"}
      }
      gradient: buttonFix.down ? gradFix2 : gradFix1
      Text {
        id: textFix
        text: qsTr("ЗАФИКСИРОВАТЬ")
        anchors.centerIn: rectFix
        font {
          family: "Courier New"
          pointSize: 30
        }
        color: "black"
      }
    }
  }

  Button {
    id: buttonTare
    width: parent.width - 40
    height: (parent.height / 2) - 20
    anchors.bottom: parent.bottom
    anchors.horizontalCenter: parent.horizontalCenter
```

```
    hoverEnabled: false
    onClicked: {
      id++
      arrayModel.append({id: id,
                status: scales.isStable() ? "ST" : "US",
                type: scales.grossMode() ? "GS" : "NT",
                weight: scales.getValue(),
                unit: "g"}
                )
      totalWeight += scales.getValue()
      totalWeightField.text = totalWeight % 1 === 0 ?
                              totalWeight + ".0"
                            : totalWeight
      scales.setRequestTare()
      tareWeightText.text = outputValue.text
    }
    background: Rectangle {
      id: rectTare
      anchors.fill: parent
      border.width: 1
      border.color: "black"
      radius: 20
      Gradient {
        id: gradTare1
        GradientStop { position: 0.0; color: "white" }
        GradientStop { position: 1.0; color: "#80808080"}
      }
      Gradient {
        id: gradTare2
        GradientStop { position: 1.0; color: "white" }
        GradientStop { position: 0.0; color: "#80808080"}
      }
      gradient: buttonTare.down ? gradTare2 : gradTare1
      Text {
        id: textTare
        text: qsTr("TAPA")
        anchors.centerIn: rectTare
        font {
          family: "Courier New"
          pointSize: 48
        }
        color: "black"
      }
    }
  }
}

Rectangle {
  id: tareWeight
  width: parent.width - 40
  height: parent.height / 7 * 1.5
```

```
        anchors.top: operButtons.bottom
        anchors.topMargin: parent.height / 8
        anchors.left: oper.left
        anchors.leftMargin: 20
        anchors.right: oper.right
        anchors.rightMargin: 20

        border.color: "black"
        border.width: 2

        Text {
          id: tareWeightText
          text: "0.0"
          font.family: "Courier New"
          font.pointSize: 50
          anchors.centerIn: parent
        }
      }
    }
  }
}
```