Министерство науки и высшего образования Российской Федерации
Филиал федерального государственного автономного образовательного учреждения высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)» в г. Миассе
Факультет «Электротехнический»
Кафедра «Автоматика»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
_____С.С. Голощапов
_____2018 г.

Модернизация пульта настройки плат датчиков давления

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ – 270304.2018.022.00 ПЗ ВКР

Консультант
доцент кафедры АиУ
_____Е.В.Вставская
_____2018г.

Руководитель проекта
Начальник технол отдела
_____Н.Ф.Кабилов
_____2018г.

Автор проекта
студент группы МиЭт-482
_____ М.В.Ильиных
_____2018 г.

Нормоконтролер
кафедры АиУ
_____Т.А. Барбасова
_____2018г.

Миасс 2018

АННОТАЦИЯ

Ильиных М.В. Модернизация пульта настройки плат датчиков давления. – Миасс: филиал «ФГАО ВО ЮУрГУ (НИУ)», Эт; 2018, 86 с. 23 ил., библиогр. список – 57 наим., 2 прил., 1 лист схем ф. А3, 1 листов схемы ф. А4

В данной выпускной квалификационной работе проведена автоматизация процесса калибровки плат датчиков давления Метран-55. В ходе работы рассматриваются основные виды датчиков давления. Подробно рассматривается тензорезистивный метод измерения давления. Даётся подробное описание датчика Метран-55. Проводится описание работы стенда по настройки плат. Разрабатывается способ автоматизации процесса калибровки плат датчиков давления Метран-55. Даётся описание электрической схемы для автоматического переключения режимов на пульте. Приведен протокол удаленного доступа калибратора ПКМ-510. Разрабатывается программное обеспечение для микроконтроллера и ПК.

Данная тема является актуальной, так как востребована в реально существующем производстве. Автоматизация процесса калибровки плат датчиков давления Метран-55 позволяет уменьшить время настройки продукции. Сведение к минимуму участие человека в процессе калибровки позволяет уменьшить вероятность ошибок, вызванных человеческим фактором.

					270304.2018.332.00 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Ильиных М.В.			Модернизация пульта настройки плат датчиков давления	Лит.	Лист 6 / Листов 86
Провер.		Кабилов Н.Ф.				Д	
						ЮУрГУ (НИУ) Кафедра «Автоматика»	
Н. Контр.		Барбасова Т.А.					
Утв.		Голощапов С.С.					

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В конце прошлого века на базе лаборатории приборостроительного факультета Южно-Уральского Государственного Университета была основана компания «ЭлМетро». За время своей работы на предприятии было разработано и запущено в производство более двадцати типов измерительных приборов. Данные приборы активно используются в различных сферах промышленной автоматизации. Среди первых разработок компании можно отметить калибраторы давления и электрических сигналов, прецизионные мультиметры, интеллектуальные датчики температуры и преобразователи для них.

Сегодня продукция «ЭлМетро» широко известна на российском и зарубежном рынке. Продукция компании используется в различных областях промышленности. Среди продукции можно выделить многопараметрические кориолисовые расходомеры ЭлМетро-Фломак, газовые ультразвуковые расходомеры ЭлМетро-ДРУ, семейство многоканальных видеографических регистраторов ЭлМетро-ВиЭР, общепромышленные и искробезопасные модули ввода вывода ЭлМетро-МВВ, ПИД-регуляторы ЭлМетро-ТеИР, импульсные источники питания ЭлМетро-ИПТ, преобразователи из RS-485 в USB, HART модемы, калибраторы-контроллеры давления ЭлМетро-Паскаль, прецизионные мультиметры ЭлМетро-Кельвин, портативные калибраторы ЭлМетро-Вольта. Все приборы имеют фирменное программное обеспечение для удаленного конфигурирования с персонального компьютера. Так же одним из направлений деятельности компании является инжиниринговые услуги («метрологический инжиниринг» и «проектный инжиниринг»)[44].

На предприятии осуществляется массовое производство плат датчиков давления Метран-55 (СПГК.5175.151) исполнения 4-20мА. Данный датчик давления, это современный микропроцессорный прибор, который предназначен для работы в системах автоматического контроля, регулирования и управления технологическими процессами и обеспечивает непрерывное преобразование измеряемой величины. Датчик Метран-55 предназначены для преобразования

давления рабочих сред: жидкости, пара, газа в унифицированный токовый выходной сигнал. Принцип работы датчика основан на тензорезистивном эффекте [41].

Одним из этапов производства плат датчика давления Метран-55 являться калибровка. Калибровку плат датчиков осуществляется для дальнейшей корректной работы датчика, а так же для проверки основных электрических параметров собранных плат, что позволяет выявлять бракованные изделия.

Данный процесс являться полуавтоматическим и требует следующего оборудования: источник питания, обеспечивающий напряжение питания 50В; компьютер с установленным программным обеспечением; стенд настройки М-55 (ТО.5175.151.02); калибратор в режиме измерения тока (ПКМ-510); спутник для настройки (ТО.5175.151.03).

Схему для калибровки плат датчика можно видеть на рисунке 1.

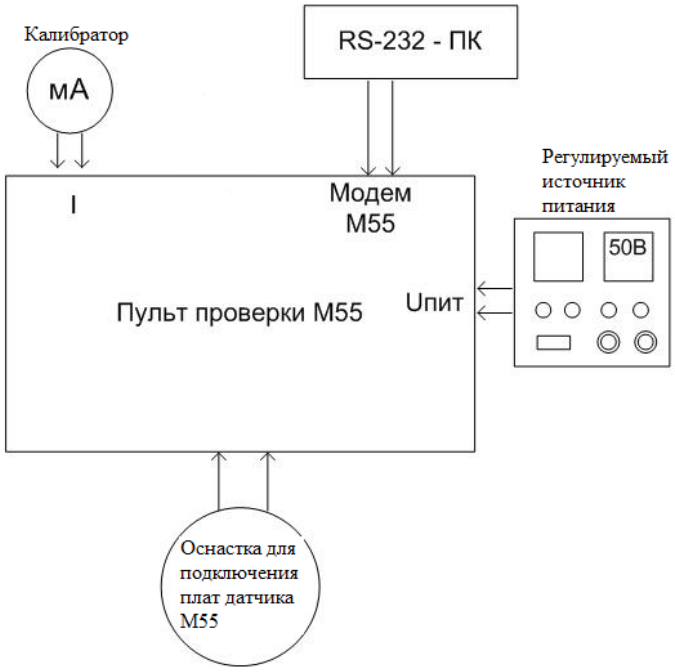

Рисунок 1 – Исходная схема для калибровки плат

Стенд для настройки датчиков Метран-55 предназначен для контроля основных электрических параметров собранных плат. Стенд имитирует тензомост

преобразователя давления. Плата датчика соединяется с проверочным стендом через спутник (рисунок 2). Калибратор так же подключается к проверочному стенду для снятия токовых сигналов с платы датчика.

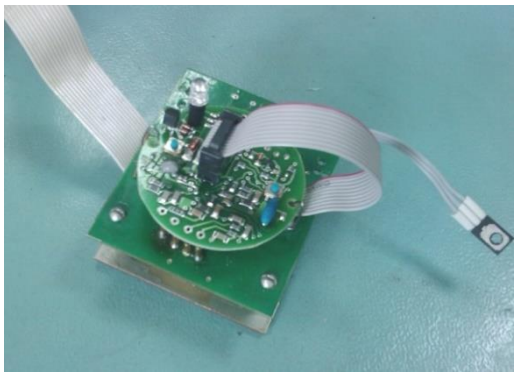

Рисунок 2 – Плата датчика, установленная на оснастку

Полуавтоматическое управление процессом калибровки происходит с компьютера. Программа проверяет связь со спутником, считывает и проверяет коды АЦП датчика. Далее оператору необходимо вписать значения тока с калибратора во всплывающие окна программы (рисунок 3). Используя полученные данные, программа записывает в память датчика исправленные значения.

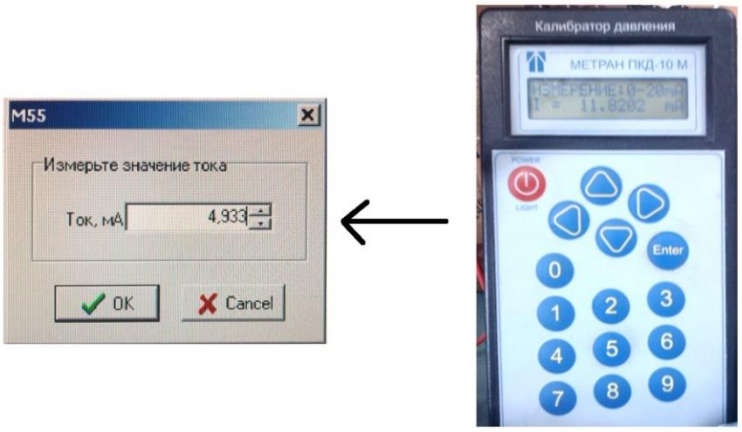

Рисунок 3 – Считывание показаний с калибратора

Далее следует ряд тестов для того, чтобы выявить нарушения в работе датчика. Оператор проверяет показания на калибраторе при 4 мА, 20 мА, максимальный ток платы, ток ошибки, минимальный ток платы, 50% ток платы. Проверяет работоспособность датчика при кратковременных (15 мс) отключениях питания, исправность кнопок и светодиода на датчике. По итогу пройденного теста оператор может судить о характере неисправности в плате. На выходе получается настроенная плата.

Недостатком данного способа являются достаточно высокое время калибровки единицы продукции (около трех минут). Так же данный процесс являться монотонным, что приводит к быстрому утомлению сотрудника.

Поэтому целью дипломного проекта является автоматизация процесса калибровки плат датчиков давления Метран-55.

Для достижения данной цели необходимо решить следующие задачи:

1) исследовать возможности данной установки;

2) разработать способ автоматизации процесса калибровки;

3) разработать электрическую схему, автоматического переключения режимов на стенде;

4) подобрать необходимое оборудование для разработанной схемы;

5) разработать программное обеспечение для микроконтроллера;

6) разработать программное обеспечение для ПК;

7) разработать удобный интерфейс для работы оператора;

8) проверить разработанную систему на соответствие техническому заданию;

9) внедрить систему на предприятие и обучить операторов работе с программным продуктом.

Все указанные вопросы рассмотрены в настоящей пояснительной записке.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Датчики давления

Датчиками давления являются измерительные приборы, которые выдают на выходе сигнал, показывающий величину давления измеряемой среды. Сегодня датчики давления используются практически во всех автоматизированных системах. Ни одна отрасль современного производства не обходиться без данного вида датчика.

Давление необходимо учитывать при проектировании многих химических процессов. Давление определяется как сила, действующая на единицу площади.

Существуют три типа измеряемого давления:

1) Абсолютное давление - атмосферное давление плюс избыточное давление;

2) Избыточное давление - абсолютное давление минус атмосферное давление;

3) Дифференциальное давление - разность давлений между двумя точками.

Датчик давления можно разделить на три основные составляющие части.

1. Первичный преобразователь давления, в состав которого входит чувствительный элемент, выступающий в роле приемника давления.

2. Схема вторичной обработки сигнала, преобразующая электрический сигнал с первичного преобразователя в информационный сигнал.

3. Устройство вывода информационного сигнала.

Существует множество различных видов датчиков давления. Основными отличиями одних приборов от других являются пределы измерений, динамические и частотные диапазоны, точность регистрации давления, допустимые условия эксплуатации, массогабаритные характеристики, принцип преобразования давления в электрических сигнал. Рассмотрим основные виды датчиков по методу измерения давления.

При деформации тензорезисторов происходит изменение сопротивления. На этом явлении основан тензометрический метод измерения давления. Под действием давления деформируется упругий элемент датчика, к которому

прикреплены тензорезисторы. В дальнейшем данный метод будет рассмотрен подробнее.

Пьезорезистивный метод основан на интегральных чувствительных элементах из монокристаллического кремния. Кремниевые преобразователи имеют высокую чувствительность благодаря изменению удельного объемного сопротивления полупроводника при деформировании давлением.

Ёмкостный метод основан на зависимости изменения электрической ёмкости между обкладками конденсатора и измерительной мембраны от подаваемого давления. Основным преимуществом ёмкостного метода является защита от перегрузок.

В основе резонансного метода лежит изменение резонансной частоты колеблющегося упругого элемента при деформировании его силой или давлением. Это и объясняет высокую стабильность датчиков и высокие выходные характеристики прибора. К недостаткам можно отнести индивидуальную характеристику преобразования давления, значительное время отклика, невозможность проводить измерения в агрессивных средах без потери точности показаний прибора.

Индуктивный метод основан на регистрации вихревых токов (токов Фуко). Чувствительный элемент состоит из двух катушек, изолированных между собой металлическим экраном. Преобразователь измеряет смещение мембраны при отсутствии механического контакта. В катушках генерируется электрический сигнал переменного тока таким образом, что заряд и разряд катушек происходит через одинаковые промежутки времени. При отклонении мембраны создается ток в фиксированной основной катушке, что приводит к изменению индуктивности системы. Смещение характеристик основной катушки дает возможность преобразовать давление в стандартизованный сигнал, по своим параметрам прямо пропорциональный приложенному давлению.

В основе ионизационного метода лежит принцип регистрации потока ионизированных частиц. Аналогом являются ламповые диоды. Лампа оснащена двумя электродами: катодом и анодом, — а также нагревателем. В некоторых

лампах последний отсутствует, что связано с использованием более совершенных материалов для электродов. Преимуществом таких ламп является возможность регистрировать низкое давление — вплоть до глубокого вакуума с высокой точностью. Однако следует строго учитывать, что подобные приборы нельзя эксплуатировать, если давление в камере близко к атмосферному. Поэтому подобные преобразователи необходимо сочетать с другими датчиками давления, например, емкостными. Зависимость сигнала от давления является логарифмической.

Одним их самых простых видов датчиков являются ртутные. Работает по принципу сообщающихся сосудов. На один из этих сосудов давить измеряемое давление. Давление определяется по величине ртутного столба.

В основе пьезоэлектрических датчиков лежит прямой пьезоэлектрический эффект, при котором пьезоэлемент генерирует электрический сигнал, пропорциональный действующей на него силе или давлению. Пьезоэлектрические датчики используются для измерения быстроменяющихся акустических и импульсных давлений, обладают широкими динамическими и частотными диапазонами, имеют малую массу и габариты, высокую надежность и могут использоваться в жестких условиях эксплуатации.

Рассмотрим подробнее основные характеристики датчиков.

Насыщение – диапазон выходных сигналов, за верхней границей рабочего диапазона обусловленный нелинейностью преобразования входного воздействия (обычно из-за физических или энергетических ограничений собственно функции преобразования). Каждый датчик имеет свои пределы рабочих характеристик. Даже если он считается линейным, при определенном уровне внешнего воздействия его выходной сигнал перестанет отвечать приведенной линейной зависимости: датчик вошел в зону нелинейности или в зону насыщения.

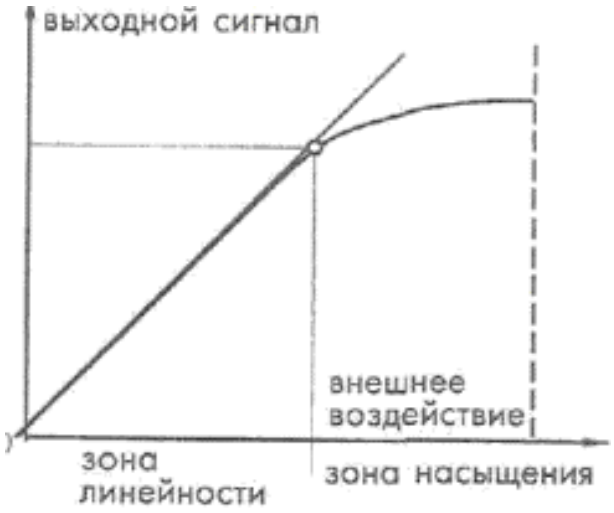

Рисунок 1.1 – Зона линейности и насыщения датчика

Воспроизводимость – способность датчика при соблюдении одинаковых условий выдавать идентичные результаты. Определяется по максимальной разности выходных значений датчика, полученных в двух циклах калибровки. Обычно выражается в процентах от максимального значения входного сигнала:

$$\delta_r = \frac{\Delta}{FS} \cdot 100\%, \quad (1.1)$$

где FS – максимальное значение входного сигнала;

Δ – максимальная разность выходных значений датчика, полученных в двух циклах калибровки.

Причинами плохой воспроизводимости результатов часто являются: тепловой шум, поверхностные заряды, пластичность материалов и т.д.

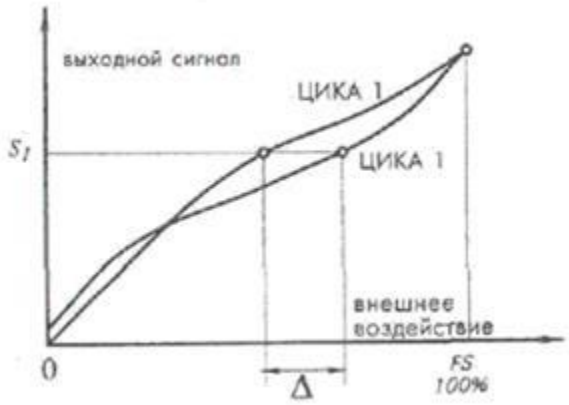

Рисунок 1.2 – Определение воспроизводимости датчика

Мертвая зона – нечувствительность датчика в определенном диапазоне входных сигналов. В пределах этой зоны выходной сигнал остается почти постоянным (часто равным нулю).

Рисунок 1.3 – Мертвая зона датчика

1.2 Тензорезистивные датчики

Тензорезистивные датчики являются наиболее надежные, точные, универсальные и распространенные в промышленной среде.

Принцип действия тензорезистора заключается в изменении длины и поперечного сечения проводников или полупроводников, и, как следствие, изменении величины сопротивления, что показано в формуле 1.2.

$$\frac{\Delta R}{R}=\frac{\Delta \rho}{\rho}+\frac{\Delta S}{S}+\frac{\Delta L}{L}, \quad (1.2)$$

где R , ΔR – начальное значение и приращение сопротивления;

ρ , $\Delta \rho$ – начальное значение и приращение удельного сопротивления;

S , ΔS – начальное значение и приращение поперечного сечения;

L , ΔL – начальное значение и приращение длины.

Выразив площадь через диаметр сечения, а также их отношение и отношение их приращений:

$$S = \pi d^2/4, \quad (1.3)$$

$$\frac{\Delta S}{\Delta d}=2\pi/4\,, \quad (1.4)$$

$$\frac{S}{d}=\pi/4\,, \quad (1.5)$$

получаем:

$$\frac{\Delta S}{S}=2\frac{\Delta d}{d}.$$

Тогда:

$$\frac{\Delta R}{R}=\frac{\Delta \rho}{\rho}+2\frac{\Delta d}{d}+\frac{\Delta L}{L}=\frac{\Delta \rho}{\rho}+(2\nu+1)\frac{\Delta L}{L}, \quad (1.6)$$

где $\nu=\frac{\Delta d}{d}/\frac{\Delta L}{L}$ – коэффициент Пуассона, безразмерный коэффициент отношения поперечного сжатия к продольному растяжению, зависящий только от природы

материала [51, с. 414]. Может измеряться в миллиметрах на миллиметр (мм/мм), миллиметрах на метр (мм/м) или микрострейнах.

Для металлов удельное сопротивление при деформации почти не меняется, поэтому:

$$\frac{\Delta R}{R} = K_f \frac{\Delta L}{L}, \qquad (1.7)$$

где K_f – относительный коэффициент деформации или коэффициент тензочувствительности. Для большинства металлических датчиков $K_f \approx 2$. Полупроводниковые тензоэлементы обладают гораздо большей чувствительностью $K_f \approx 50…200$ [44, с. 290].

Тензорезистор конструктивно представляет собой змейку тонких проводящих полосок, сделанных из тонкой проволоки, фольги или напыления, на гибкой подложке из ткани, бумаги или полимерной пленки. Длина пластины резистора может быть от 0,4 до 150 мм. Номинал тензорезистора варьируется, применяются и 350, и 600 Ом, но наиболее распространенным и оптимальным является номинал 120 Ом [50, с. 50].

Схема измерительных преобразований при использовании тензорезистивного датчика представлена на рисунке 1.4.

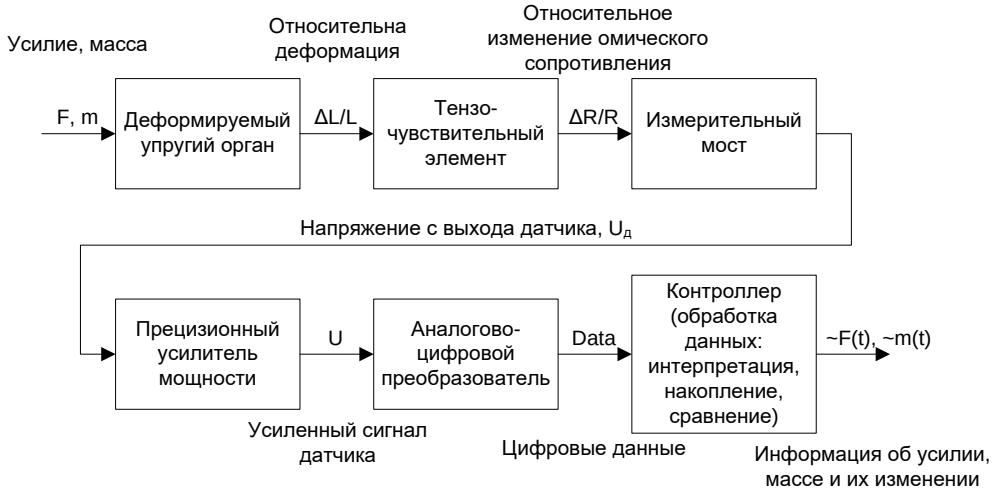

Рисунок 1.4 - Цепочка измерительных преобразований

Измерительный мост использует схему Уитстона, которая представляет собой четыре соединенных резистора: сопротивление одного требуется определить, номинал двух других известен, и один с регулируемым сопротивлением для балансировки моста (рисунок 1.5). К двум узлам подводится питание, с двух других снимается выходное напряжение, которое определяется как разность потенциалов в узлах.

$$U_д = \varphi_{12} - \varphi_{43} = I_{12} \cdot R_1 - I_{43} \cdot R_4 = \left(\frac{R_1}{R_1+R_2} - \frac{R_4}{R_4+R_3}\right) \cdot U_{пит}, \qquad (1.8)$$

где R_1 , R_2, R_3, R_4 – сопротивления резисторов (рисунок 1.5);

φ_{12}, φ_{43} – потенциалы между соответствующими сопротивлениями;

I_{12}, I_{43} – токи, протекающие через соответствующие пары резисторов;

$U_д$, $U_{пит}$ – напряжение на выходе датчика и напряжение питания.

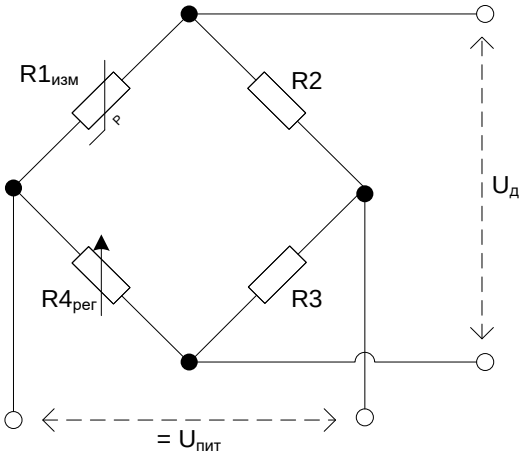

Рисунок 1.5 - Схема измерительного моста

При установке тензодатчика, обычно проводят балансировку моста – устанавливают такое значение регулировочного резистора, что напряжение на выходе датчика равно нулю, при отсутствии усилия на измеряемом органе. Тем самым производится аппаратная регулировка нуля шкалы прибора. Регулировку можно выполнить программно, в контроллере, и не прибегать к балансировке

моста, однако, в таком случае на выходе датчика всегда будет присутствовать некоторое напряжение смещения и ограничивать работу прецизионного усилителя мощности.

На работу тензорезистивного элемента оказывает значительное влияние температура окружающей среды, так как при измерении деформации сопротивление меняется в пределах десятых долей процента, поэтому тензорезисторы изготавливают из материалов с низким температурным коэффициентом, например из константана – термостабильного сплава меди, никеля и марганца. Также на схеме измерительного моста Уитстона в смежное с измеряющим тензорезистором плечо на место R_2 (рисунок 1.5) подключают идентичный компенсирующий резистор, который не подвергается действию деформации, но находится на общей подложке перпендикулярно измеряющему. Таким образом, тепловой эффект действует одинаково на два резистора из противоположенных веток моста и согласно формуле 1.8 компенсируется, а деформации остается подвержен только один.

Для того чтобы учесть падение напряжения на питающих проводах и устранить влияние их длины на результат измерения, используют шестипроводную схему подключения датчика. Два провода включаются в узлы вместе с питанием и служат для контроля и учета реальной величины питающего напряжения в расчетах по формуле 1.8.

При использовании тензорезисторов большинство источников погрешностей аналогичны тем, что возникают при использовании терморезисторов. Основные компоненты погрешностей:

1) случайная погрешность, вызванная технологическим разбросом сопротивлений тензорезисторов;

2) систематическая погрешность, вызванная термоэлектрическим эффектом;

3) тепловой и фликкер-шум измеряемого сопротивления;

4) температурная погрешность, вызванная разогревом датчика, протекающим током;

5) погрешность, связанная с разностью температурных коэффициентов расширения тензорезистора и материала объекта, на который наклеен тензорезистор;

6) погрешность метода (схемы измерения) сопротивления, зависящая от длины проводов и точности измерения их сопротивления;

7) внешние наводки;

8) сопротивление контактов;

9) "ползучесть" сопротивления длительно нагруженного тензорезистора;

10) погрешность измерительного модуля ввода.

1.3 Калибровка датчиков

Калибровка датчика – определение его индивидуальных коэффициентов общей формы передаточной функции. Калибровка производится, если производственные допуски на датчик превышают требуемую точность измерения.

Математическое описание передаточной функции необходимо знать до начала проведения калибровки.

Например, для линейной передаточной функции необходимо провести калибровку минимум в двух точка. Поскольку для определения коэффициентов, описывающих прямую линию, необходимо иметь два уравнения.

Для нелинейных функций калибровку требуется проводить более чем в двух точках. Количество необходимых калибровок диктуется видом математического выражения. Если передаточная функция моделируется полиноминальной зависимостью, число калибровочных точек выбирается в зависимости от требуемой точности. Поскольку, как правило, процесс калибровки занимает довольно много времени, для снижения стоимости изготовления датчиков на производстве количество калибровочных точек задается минимальным.

Другой подход к калибровке нелинейных датчиков – применение кусочно-линейной аппроксимации: любую кривую в пределах достаточно небольшого интервала всегда можно заменить линейной функцией. Поэтому нелинейную

передаточную функцию можно представить в виде комбинации линейных отрезков, каждый из которых обладает своими собственными коэффициентами.

Для проведения калибровки датчиков важно иметь точные физические эталоны, позволяющие моделировать соответствующие внешние воздействия. Например, при калибровке контактного датчика температуры его необходимо помещать среду, где есть возможность точно регулировать температуру. При калибровке инфракрасных датчиков требуется наличие черного тела, а для калибровки гигрометров - набор насыщенных растворов солей, используемых для поддержания постоянной относительной влажности в закрытом контейнере и т.д.

Следовательно, точность последующих измерений напрямую связана с точностью проведения калибровки и ошибка калибровочных эталонов должна включаться в полную ошибку измерений.

1.4 Описание датчиков давления Метран-55

Малогабаритные датчики Метран-55 предназначены для работы в различных отраслях промышленности, системах автоматического контроля, регулирования и управления технологическими процессами и обеспечивают непрерывное преобразование измеряемых величин избыточного, абсолютного давления, разрежения, давления-разрежения нейтральных и агрессивных сред в унифицированный токовый выходной сигнал.

Рисунок 1.6 – Датчик давления Метран-55

Простота конструкции, надежность, малые габариты, невысокая стоимость обеспечивают повышенный спрос потребителей. Преимущества датчиков исполнения МП: погрешность измерений ±0,15%; непрерывная самодиагностика; встроенный фильтр радиопомех; микропроцессорная электроника; возможность простой и удобной настройки параметров двумя кнопками.

Дополнительно датчики Метран-55-ДМП 331П, Метран-55-ДМК 331П, Метран55-ДС 200П, Метран-55-ЛМП 331и предназначены для работы в пищевой промышленности при контакте с пищевыми продуктами. Датчики Метран-55-ДС 200П, Метран-55- ЛМП 331, Метран-55-ЛМК 351, Метран-55-ЛМК 858 предназначены для работы в фармацевтической промышленности, датчики Метран-55-ДМК 331, Метран-55-ДМП 331и, Метран-55-ДМП 333и, Метран-55-ДМП 343 предназначены для работы в медицинской промышленности. Датчик Метран-55-ДМП 343 можно применять в биомедицинском оборудовании - переливание крови, насосы, респираторное оборудование. Датчик Метран-55-ЛМК 457 предназначен для работы в морской воде. Датчики Метран-55-ДМК 331, Метран-55-ЛМК 351 могут иметь кислородное исполнение[41].

Схема датчика Метран-55 представлено на рисунке 1.4.2. Датчик состоит из корпуса 1, мембранного тензопреобразователя 2 и электронного преобразователя 3.

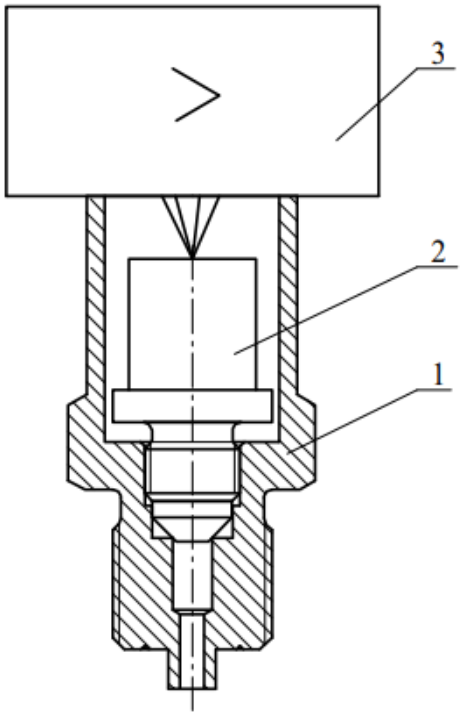

Рисунок 1.4.2 - Схема датчика Метран-55

Работа датчика (кроме датчиков М-55-ДМК 331, М-55-ДМК 331П, М-55-ДМП 330 Л, М-55-ЛМК 331, М-55-ЛМК 351, М-55-ЛМК 358, М-55-ЛМК 457 и М-55- ЛМК 858) основана на использовании тензометрического эффекта в полупроводниках. Датчик генерирует электрический выходной сигнал пропорционально уровню давления в системе. Основным элементом датчика давления является сенсор. Сенсор представляет собой кремниевый чувствительный элемент, размещенный на керамической подложке. Измеряемое давление подводится в рабочую полость и воздействует непосредственно на измерительную мембрану тензопреобразователя, вызывая ее прогиб. Тензорезисторы соединены в мостовую схему. Деформация измерительной мембраны вызывает изменение сопротивления тензорезистора и разбаланс

мостовой схемы. Электрический сигнал, образующийся при разбалансе мостовой схемы, подается в электронный преобразователь. Электронный преобразователь преобразует электрический сигнал от тензопреобразователя в стандартный токовый сигнал.

Работа датчиков М-55-ДМК 331, М-55-ДМК 331П, М-55- ЛМК 331, М-55-ЛМК 351, М-55-ЛМК 358, М-55-ЛМК 457 и М-55-ЛМК 858 основана на использовании емкостного принципа измерения. Сенсорная мембрана действует как растянутая пружина, отклоняясь в ответ на приложенное к ней давление. Смещение сенсорной мембраны пропорционально давлению. При изменении положения мембраны изменяется емкость между сенсорной мембраной и пластиной конденсатора. Вариация емкости вызывает изменение частоты генератора. Изменение частот преобразуется в соответствующий выходной ток, напряжение[41].

2 ПРОЕКТИРОВАНИЕ АТОМАТИЗИРОВАННОЙ СИСТЕМЫ

2.1 Описание работы стенда

Пульт проверки датчиков Метран-55 предназначен для контроля основных электрических параметров собранных плат М55 в исполнении 4/20 мА и 0/5 мА. Малые габаритные размеры устройства позволяют его использование на рабочем столе оператора. Полуавтоматическое управление работой пульта происходит с компьютера. По итогу пройденного теста оператор может судить о характере неисправности в плате. На выходе с пульта получается настроенная плата датчика, переведённая в технологический режим для дальнейшей термотоковой тренировки. Основной принцип работы основан на имитации тензомоста преобразователя давления. Пульт обеспечивает измерение напряжения внутреннего стабилизатора платы, определение дополнительной погрешности платы по нагрузочному сопротивлению и напряжению источника питания.

Пульт проверки плат был разработан для внутреннего использования в ООО НПФ «Специальная Автоматика».

Основные характеристики пульта:

1) напряжение питания 50В±1В;

2) потребляемый ток не более 120 мА;

3) масса не более 1,5 кг;

4) эксплуатация при температуре от плюс 10 до минус 50 градусов Цельсия;

5) по степени защиты от воздействия пыли и воды Пульт соответствует группе IP20 по ГОСТ 14254;

6) пульт обеспечивает питание платы следующими напряжениями: 12 В, 24 В, 36 В, 42 В;

7) нагрузочные сопротивления составляют: 50 и 1050 Ом для плат 4/20 мА; 250 и 2500 Ом для плат 0/20 мА;

В состав установки так же входит:

1) источник питания, обеспечивающий напряжение питания 50В;

2) компьютер с установленным программным обеспечением;

3) калибратор в режиме измерения тока (ПКМ-510);

4) спутник для настройки (ТО.5175.151.03).

2.2 Разработка способа автоматизации процесса калибровки

Основной задачей автоматизации любого процесса является экономия трудовых ресурсов за счет замены труда человека трудом машины. В процессе калибровки сотруднику приходиться делать достаточно много ручной работы, что приводит к увеличению продолжительности процесса. Уменьшив количество ручного труда в данном процессе, мы сможем добиться уменьшения времени калибровки и соответственно получить некоторый экономический эффект.

Поэтому в автоматизированной системе все необходимые токовые показания с ПКМ-510 буду считываться автоматически. Для этого калибратор был соединён с ПК по интерфейсу RS-232. Для программной реализации связи был использован протокол удаленного доступа ПКМ-510.

Так же в разрабатываемой системе предусмотрено автоматическое переключение режимов на стенде. Реализовано это с помощью электрической схемы с микроконтроллером, который подключен по интерфейсу RS-232 к ПК.

Схему для калибровки плат датчика, после автоматизации, можно видеть на рисунке 2.1.

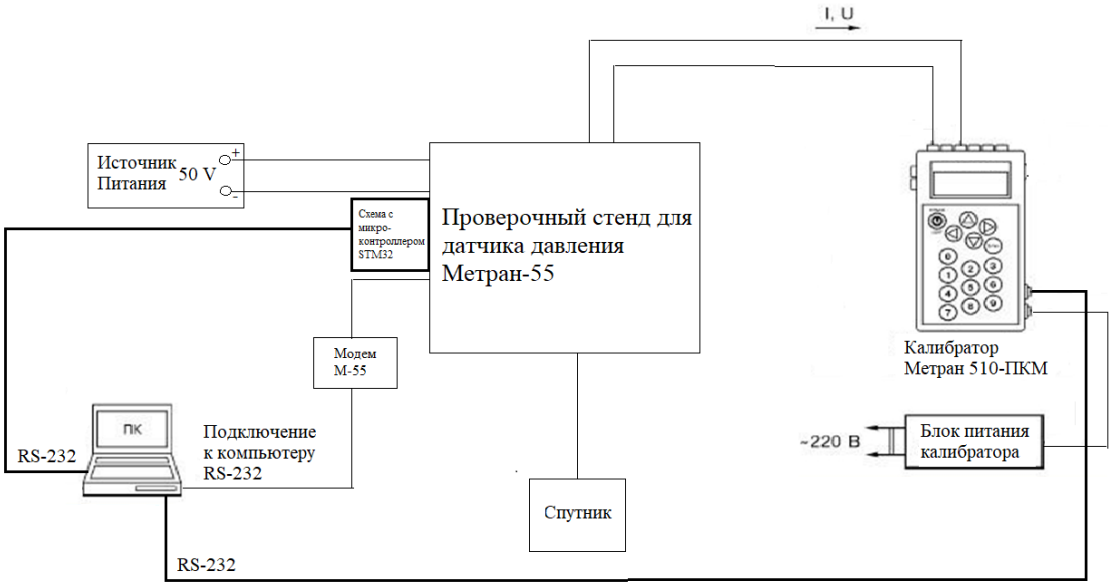

Рисунок 2.1 – Автоматизированная схема калибровки плат датчиков давления Метран-55

2.3 Подбор оборудования для электрической схемы

2.3.1 Выбор микроконтроллера

Для реализации устройства необходимо определиться с аппаратной частью. Возьмем микроконтроллер серии STM32 – STM32F103RXB.

Микроконтроллеры STM32 выпускаются компанией STMicroelectronics и построены на ядре ARM Cortex, которое на текущий момент занимает лидирующие позиции на рынке микроконтроллеров.

Помимо ядра микроконтроллер, содержит внутри себя систему тактирования, систему памяти, включающую Flash память программ и оперативную память данных, и набор периферийных модулей, которые предназначены для получения информации и формирования сигналов управления. К периферийным модулям относятся, например, таймеры, аналого-цифровые и цифро-аналоговые преобразователи, контроллеры прямого доступа к памяти, компараторы, интерфейсы связи.

Данный микроконтроллер будет подходить для реализации нашего устройства. Основные технические характеристики микроконтроллера STM32F103RXB представлены в таблице 1.

Таблица 1 – Основные технические характеристики микроконтроллера

Ядро	ARM Cortex-M3
Объем Flash-памяти программ, кБ	64
Объём оперативной памяти, кБ	20
Максимальная частота ядра, МГц	72
Количество входов/выходов	51
Напряжение питания, В	от 2 до 3.6
Рабочая температура, ˚С	от минус 40 до плюс 85
Обозначение на схеме	DD1
Количество, шт.	1

2.3.2 Выбор реле

Для переключения режимов на пульте по сигналу с микроконтроллера воспользуемся твердотельным реле PS7241.

Таблица 2 – Основные технические характеристики микроконтроллера

Название	PS7241
Напряжение на входе, В	1.2
Объём оперативной памяти, кБ	20
Сопротивление открытого канала, Ом	20
Ток коммутации, мА	120
Рабочая температура, ˚С	от минус 40 до плюс 85
Обозначение на схеме	VU1-VU6
Количество, шт.	6

Между включением реле и входом ножкой микроконтроллера установим резистор величиной 1 кОм.

Таблица 3 – Характеристики резисторов МЛТ-0.125 1 кОм

Название	МЛТ-0.125 ОЖ0.467.180ТУ
Номинал сопротивления, кОм	1
Рассеиваемая мощность, Вт	125
Обозначение на схеме	R1
Количество, шт.	6

2.3.3 Схема питания

Для включения микроконтроллера необходимо организовать его питание напряжением 2…3.6 В. Входное напряжение равно 5 В. Для установления необходимого напряжения питания микроконтроллера использовали схему из одного стабилизатора LM1117, двух керамических конденсаторов по 470 микрофарад и одного электролитического конденсатора на 100 микрофарад, чтобы сгладить возможные помехи напряжения. Технические параметры конденсаторов представлены в таблицах 4,5.

Таблица 4 – Характеристика керамических конденсаторов К10-7В

Название	К10-7В
Номинальная ёмкость, мкФ	0.1
Рабочее напряжение, В	25
Диапазон температур, ˚С	от минус 25 до плюс 85
Обозначение на схеме	С1, С2
Количество, шт.	2

Таблица 5 – Характеристика электролитических конденсаторов К50-35

Название	К50-35
Номинальная ёмкость, мкФ	100
Рабочее напряжение, В	25
Диапазон температур, ˚С	от минус 40 до плюс 85
Обозначение на схеме	С3
Количество, шт.	2

Стабилизатор 78L05 имеет входное напряжение 7…20 В, выходное 4.5…5.5 В, выходной ток 100 мА, что полностью удовлетворяет требованиям к питанию микроконтроллера. Основные технические параметры стабилизатора представлены в таблице 6.

Таблица 6 – Характеристики стабилизатора LM1117

Название	Стабилизатор положительного напряжения 78L05
Входное напряжение, В	4…15
Выходное напряжение, В	3.3
Выходной ток, А	до 1
Диапазон температур, ˚С	от минус 20 до плюс 125
Обозначение на схеме	DA1
Количество, шт.	1

2.3.4 Интерфейс USART

USART – универсальный синхронно-асинхронный приёмопередатчик, поддерживает режим синхронной передачи данных – с использованием дополнительной линии тактового сигнала.

Для того чтобы осуществить подключение USART использовали двухконтактный клеммник DG305-5.0-02P-12.

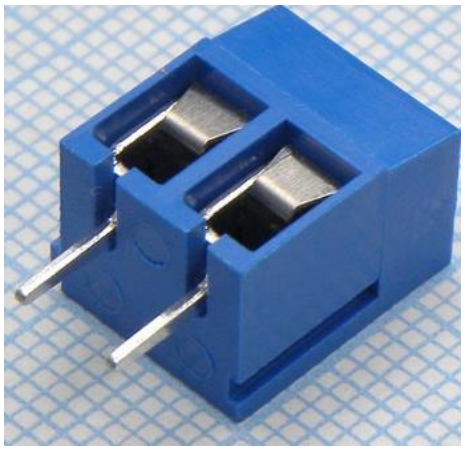

Рисунок 2.2– Внешний вид клеммника

Основные технические параметры клеммника представлены в таблице 7.

Таблица 7 – Клеммник DG305-5.0-02P-12

Тип исполнения	Прямой
Функциональное назначение	Клеммник винтовой
Рабочий ток, А	15
Рабочее напряжение, В	300
Обозначение на схеме	X2
Количество, шт.	1

2.4 Разработка электрической части

Необходимо, чтобы спроектированная электрическая схема по сигналу с компьютера автоматически переключала режимы на пульте. В основе схемы находиться микроконтроллер STM32 с подключённым интерфейсом USART. Получая сигналы по данному интерфейсу, микроконтроллер их обрабатывает и

подает сигнал на нужные реле, которые в свою очередь имитируют процесс нажатия кнопок на пульте, тем самым переключая режимы.

Проверочный стенд использует два внутренних источника питания на 5 В и на 12 В. Чтобы в нашей схеме не использовать дополнительный источник питания, воспользуемся уже существующим на 5 В. От него мы и будем питать микроконтроллер и остальные устройства на схемы.

Для включения микроконтроллера необходимо организовать его питание напряжением 2…3.6 В. Входное напряжение равно 5 В. Для установления необходимого напряжения питания микроконтроллера использовали схему из одного стабилизатора.

Так же была предусмотрена возможность внутрисхемного программирования для возможности внесения изменений в дальнейшем.

Данная схема была установлена в корпус пульта. Таким образом добавление электрической части не увеличило пространства занимаемое всей установкой в целом.

3 ОПИСАНИЕ ПРОТОКОЛА И ПРОГРАММ

3.1 Описание протокола удаленного доступа калибратора Метран 510-ПКМ

Протокол удаленного доступа Метран 510-ПКМ предназначен для реализации удаленного управления калибратором в части измерения и воспроизведения сигналов.

Для подключения калибратора к ПК требуется чтобы:

1. Калибратор Метран 510-ПКМ был приобретен с опцией «RS232», которая включает переходник RS-232 (или с опцией «USB» с переходником в интерфейс USB) и компакт-диск с серийным номером.

2. ПК с наличием интерфейса RS-232 (или USB).

Для подключения к калибратору со стороны ПК используется физический порт RS-232 (или виртуальный в случае использования переходника USB). Параметры связи: скорость связи – 9600 бод; контроль четности – нет; количество стоп-бит - 1. При использовании переходника RS-232 состояние линий RTS и DTR порта должно быть установлено: RTS – установлен; DTR – сброшен.

Управление калибратором осуществляется в режиме «запрос-ответ». В качестве ведущего устройства всегда выступает ПК, в качестве ведомого – калибратор. Информация передается в двоичном виде. Каждая команда представляет собой фиксированный набор транзакций вида «запрос-ответ», где запрос представляет собой посылку компьютером одного байта команды/данных в калибратор и прием ответного байта транзакции.

Выполнение команды калибратором начинается после приема первого запроса от ПК. После отправки калибратором ответа последней транзакции калибратор переходит в режим ожидания следующей команды.

Далее принята следующая схема описания команд протокола в виде таблицы:

Таблица 8 – Описание команд протокола

№	ПК	Калибратор
1	A	B
2	C	D
3	E	F

Первый столбец - это номер транзакции. Второй столбец содержит байты, которые посылает ПК в качестве запроса. Для чисел используется десятичная форма записи, если не указано иное. Третий столбец содержит байт данных, который присылает калибратор в ответ на запрос ПК в данной транзакции. Для чисел используется десятичная форма записи, если не указано иное.

Соответственно вышеприведенной таблице последовательность действий будет такова:

1. ПК отсылает калибратору один байт с содержимым, указанным в поле A и ожидает ответа от калибратора.

2. Калибратор, получив этот байт (A) отсылает в ответ один байт (B).

3. ПК получает ответ (B) от калибратора. Если в поле B таблицы указано константное значение (число), то ПК сверяет полученное значение со значением, указанным в поле B. При несовпадении значений команда считается не выполненной (помеха на линии связи). Если в поле B таблицы указано не константное значение, то ПК запоминает это значение.

4. ПК выполняет следующую транзакцию согласно пунктам 1 – 3 и так до тех пор, пока не будут выполнены все транзакции.

Прерывать выполнение команды со стороны ПК не допускается. В случае, если ПК не получил ответ от калибратора в течение 3-х секунд, то считается, что связь с калибратором отсутствует.

В случае, если калибратор не получил запрос следующей транзакции (текущей выполняемой команды) от ПК в течение 3-х секунд, транзакция отменяется и вся команда считается не выполненной. Калибратор при этом выключается.

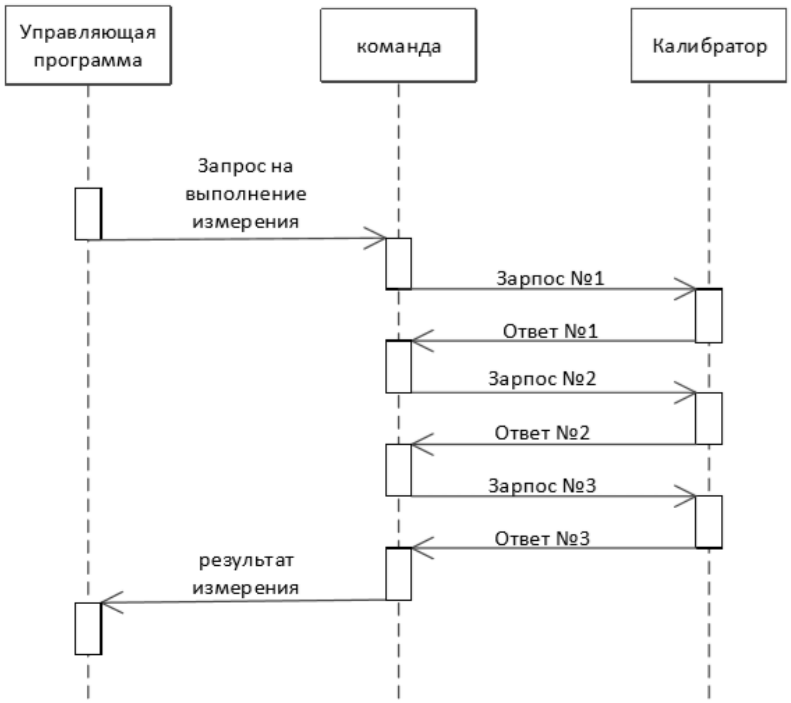

Рисунок 3.1 – Процесс управления калибратором с ПК

В некоторых командах для проверки корректности принятых данных используется контрольная сумма (CheckSumm).

Для передачи/приема числовых значений используется формат IEE754 32bit (float). Значение занимает 4 последовательно расположенных байта.

Таблица 9 – Представление числовых значений

Байт №	Обозначение	Содержимое
1	Value/1	младший байт мантиссы (биты 0-7)
2	Value/2	средний байт мантиссы (биты 8-15)
3	Value/3	старший байт мантиссы (биты 16-22)
4	Value/4	знак и порядок числа (биты 24-31)

Для уменьшения зависимости точности измерений от температуры в калибраторе применяется индивидуальная компенсация температурных уходов

(самокалибровка) некоторых элементов калибратора. Однако, в том случае, если калибратор работает под управлением компьютера, такая компенсация не выполняется в реальном времени. В том случае, если во время измерений температура окружающего воздуха не меняется, допустимо не проводить самокалибровку в течение длительного промежутка времени (возможно, в течение всего цикла измерений). Однако если температура окружающего воздуха значительно меняется во время измерений или необходим длительный по времени цикл непрерывных измерений, рекомендуется периодически давать команду калибратору (посредством указания т.н. «флага самокалибровки») на принудительное проведение самокалибровки. Так как самокалибровка занимает некоторое время, в течение которого калибратор не выдает измерения, то окончательное решение о проведении самокалибровки принимает программное обеспечение пользователя.

Для повышения точности измерений диапазон каждого типа измерения (сила постоянного тока, напряжение постоянного тока, сопротивление постоянному току) разбит на несколько поддиапазонов, каждый из которых имеет заданную точность. Главным критерием выбора поддиапазона измерения является диапазон изменения измеряемой величины.

Для воспроизведения силы постоянного тока, напряжения постоянного тока и сопротивления постоянному току в калибраторе применяется метод последовательного приближения, для которого необходимо выполнить некоторое количество уточняющих итераций (контрольных измерений). Процесс уточнения выполняется до тех пор, пока значение воспроизводимой величины находится вне допустимых пределов.

Алгоритм воспроизведения в общем виде показан на рисунке 3.2.

Рисунок 3.2 – Алгоритм воспроизведения

Алгоритм имеет 2 этапа:

1. Установка целевого значения для воспроизведения.

2. Итерационный процесс уточнения установленного значения (последовательное приближение). При этом соответствующая команда возвращает текущее установленное значение на выходе калибратора на данной итерации. Итерационный процесс можно прекратить после того, как разница между этим значением и целевым станет меньше допустимой погрешности.

В том случае, если требуется воспроизведение не константных (не постоянных во времени) сигналов имеется команда коррекции целевого значения во время итерационного процесса. В отличие от основной команды установки целевого значения она выполняется существенно быстрее. После выполнения команды коррекции целевого значения итерационный процесс последовательного приближения следует начать заново уже с новым целевым значением.

В отличие от команд измерения команды воспроизведения не имеют разбивку по поддиапазонам. Калибратор сам выбирает нужный поддиапазон воспроизведения на основе целевого значения.

Для решения поставленной задачи будет необходимо использовать команду активации удаленного управления (команда №165) и измерение силы постоянного тока в диапазоне 0-22 мА (команда №40).

Команда №165 предназначена для включения интерфейса удаленного управления калибратором по RS-232/USB. Команду следует выполнять один раз перед началом работы с калибратором. После включения питания калибратор реагирует только на данную команду, остальные команды не работают.

Таблица 10 – Формат команды №165

№	ПК	Калибратор
1	165	165
2	Byte1	Byte1
3	Byte2	Byte2
…	…	…
26	Byte25	Byte25
27	165	CheckSumm
28	165	Result

Byte1, Byte2, … Byte25 – 25-ти значный серийный номер программного обеспечения калибратора (с компакт-диска). Серийный номер передается без разделительных тире символами в кодировке ACII в верхнем регистре. Так же существует универсальный набор символов для любого калибратора данной серии, что позволит в дальнейшем не менять данные в программе при смене калибратора. CheckSumm – контрольная сумма Byte1 - Byte25. Вычисляется калибратором и передается в ответ ПК. В том случае, если ответ калибратора не совпадает с контрольной суммой, вычисленной ПК, то команду следует повторить. Result – подтверждение разблокировки интерфейса: значение 1 (0x01) – интерфейс разблокирован, значение 0 (0x00) – интерфейс не разблокирован (серийный номер не соответствует данному калибратору).

Получая команда №40, калибратор выполняет измерение значения силы постоянного тока в диапазоне 0-22 мА и возвращает измеренное значение.

Таблица 11 – Формат команды №165

№	ПК	Калибратор
1	40	40
2	40	SelfCal
3	40	Value/1
4	40	Value/1
5	40	Value/1
6	40	Value/1
7	40	CheckSumm

SelfCal – запрос на выполнение самокалибровки. Допустимые значения: 1 (0x01) – выполнить самокалибровку перед измерением, 0 (0x00) – не выполнять самокалибровку. Value – измеренное значение в формате IEE754 32bit (float), единицы измерения - мА; CheckSumm - контрольная сумма байтов Value/1, …, Value/4.

Порядок работы с калибратором в режиме удаленного доступа будет следующим. После включения питания калибратора следует однократно выполнить команду №165 для активации режима удаленного управления калибратором по интерфейсу RS-232/USB. При этом калибратор должен находиться в главном меню (по умолчанию после включения). Активация режима сохраняется до выключения питания калибратора. После выполнения этой команды становятся доступны остальные команды протокола и появляться возможность считывать токовые сигналы с помощью команды №40.

3.2 Разработка программного обеспечения для микроконтроллера

Задача микроконтроллера отслеживать по интерфейсу USART сигнал с компьютера и переключать соответствующие режимы на пульте. Переключение режимов осуществляется с помощью твердотельного реле. Порты микроконтроллера PA9 (в режиме работы USART1_TX) и PA10 (в режиме работы

USART2_RX) используются для связи с приложением на компьютере. Порты PC0 – PC5 используются как дискретные выходы для подачи сигнала на реле.

Рабочая программа микроконтроллера состоит из двух частей: инициализации и бесконечного цикла. В инициализации задаются состояния портов, конфигурации интерфейса USART и др., начальные значения портов и таймеров. Также задаются команды, которые будет выполнять обработчик прерываний.

В основной функции находится бесконечный цикл, который представляет собой чаще всего цикл while(…) {…} с единицей в аргументе: while(1) {…}.

Параметры связи: скорость связи – 38400 бод; контроль четности – нет; количество стоп-бит – 1; количество битов в байте – 8. Заметим, что такие же свойства должны быть прописаны для программы на компьютере, с которой будет происходить обмен данными по интерфейсу USART. В противном случае, связи между двумя микроконтроллерами не будет, и передача данных будет невозможна.

При первом запуске микроконтроллер самостоятельно включает необходимый режим на пульте: ТС, сенсор 120%, нагрузка 50/200, питание 36В. Данный режим необходим для начала калибровки плат датчиков. Далее программа на микроконтроллере входит в бесконечный цикл и начинает ожидать команду от приложения на компьютере.

Для переключения режимов используется следующие команды: 0x62 – включение максимального тока платы (120%), 0x63 – 50% ток платы, 0x64 – 0% ток платы, 0x65 – минимальный ток платы (-5%), 0x66 – нагрузка 50/200, 0x67 – нагрузка 1050/2500, 0x68 – питание платы 24 В, 0x68 – питание платы 42 В, 0x6a – отключение питания датчика, 0x6b – кратковременное (15мс) отключение питания датчика. Код программы приведен в приложении в разделе Листинг А.2.

3.3 Разработка программного обеспечения для ПК

Приложение для автоматизации процесса калибровки плат датчика давления Метран-55 было реализовано на языке программирования C# в среде разработке Visual Studio. Так же был использован интерфейс программирования приложений

Windows Forms, который отвечает за графический интерфейс пользователя и является частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде.

Приложение обменивается данными с калибратором, микроконтроллером и спутником через интерфейс RS-232. Обмен осуществляется с помощью следующего набора функций.

PortSputnikOpen и PortSputnikClose – для открытия и закрытия последовательного порта спутника. Параметры связи: скорость связи – 1200 бод; контроль четности – нет; количество стоп-бит – 1; количество битов в байте – 8. SputnikId – функция для считывания номера спутника. SputnikPressureCode, SputnikBoardTermoCode, SputnikSensorTermoCode – для считывания кодов АЦП датчика. SputnikRead, SputnikWrite – функции для считывания и записи данных в память датчика. SputnikSensorOn SputnikSensorOff – функции отправляют команды для включения и выключения сенсора датчика. SputnikReset – сброс датчика. SputnikSleep – отключение спутника.

Для открытия и закрытия последовательного порта калибратора используются следующие функции: PortCalibratorOpen, PortCalibratorClose. Параметры связи: скорость связи – 9600 бод; контроль четности – нет; количество стоп-бит – 1; количество битов в байте – 8; RTS – установлен; DTR – сброшен. С помощью функции UnlockCalibrator осуществляется активация режима удаленного управления калибратором. MeasureCurrentCalibrator – для считывания показаний с калибратора.

Для открытия и закрытия последовательного порта микроконтроллера используются функции PortMicroOpen и PortMicroClose. Параметры связи: скорость связи – 38400 бод; контроль четности – нет; количество стоп-бит – 1; количество битов в байте – 8. Для переключения режимов на пульте используется функция Micro.

Так же были написаны функции OldByteToFloat и OldFloatToByte для перевода байтов в число типа float и наоборот. Для поиска портов подключенных

к ПК используется функция findPorts. Функция Clear очищает содержимое интерфейса при очередном запуске калибровки.

Основные параметры и диапазоны значений, считываются с текстового файла с помощью функции ReadingParameters. Таким образом, оператор в дальнейшем может сам менять данные значения, не меняя исходный код программы.

Код программы приведен в приложении в разделе Листинг А.1.

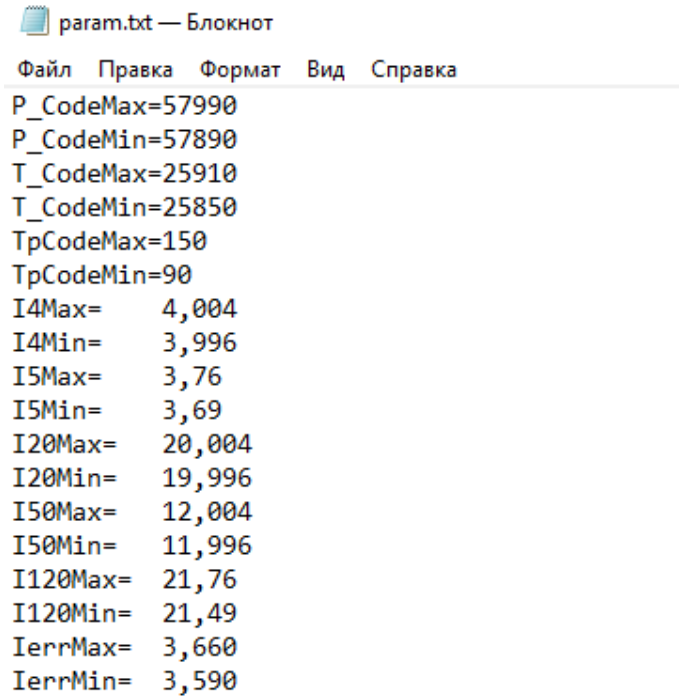

Рисунок 3.2 – Файл с параметрами

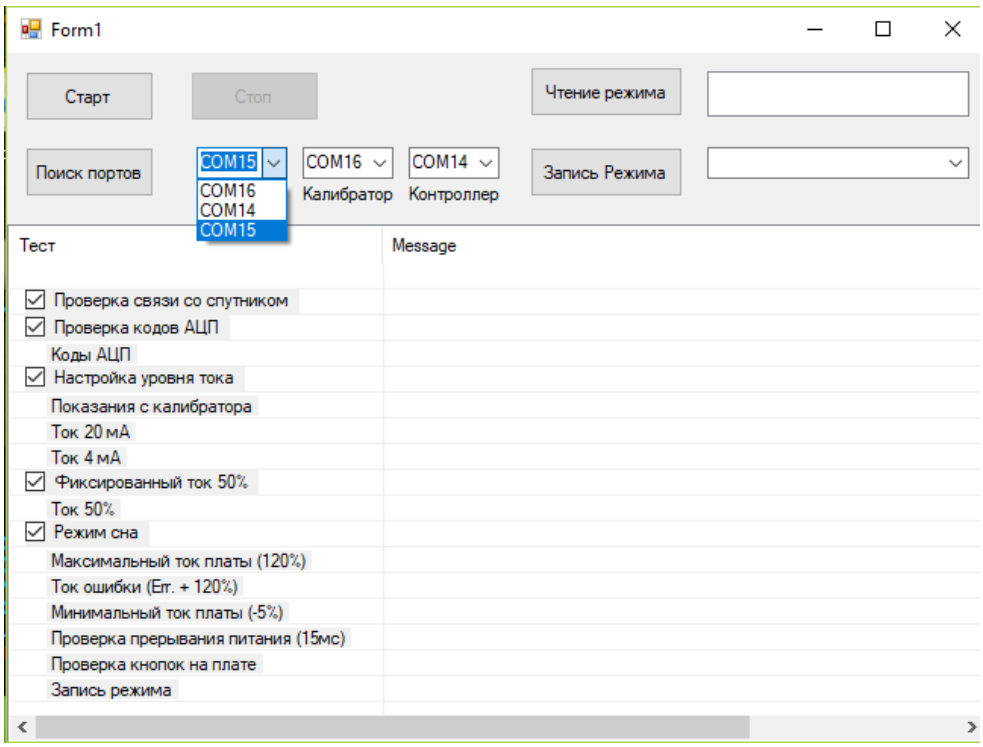

Рисунок 3.3– Выбор портов

При нажатии на кнопку «Старт» запускается процесс калибровки и проверки основных параметров платы. Осуществляется проверка связи со спутником путем считывания ID-спутника. Затем проверяются коды АЦП датчика. Считывая показания с калибратора, происходит автоматическая настройка токов 4 и 20 мА. Далее настроенная плата проходит ряд тестов, чтобы убедиться в правильности настройки токов и исправности платы. Происходит считывания показания тока с калибратора при 4 мА, 20 мА, максимальный ток платы, минимальный ток платы, ток ошибки и 50% ток. Данные показания должны лежать в допустимых пределах. Осуществляется автоматическое прерывание питания платы на 15 мс, при этом исправная плата должна продолжать выдавать корректные значения токов. Затем программа просит оператора проверить исправность кнопок на плате датчика путем одновременного их нажатия. При этом светодиод на плате должен начать мигать чаще одного раза в секунду.

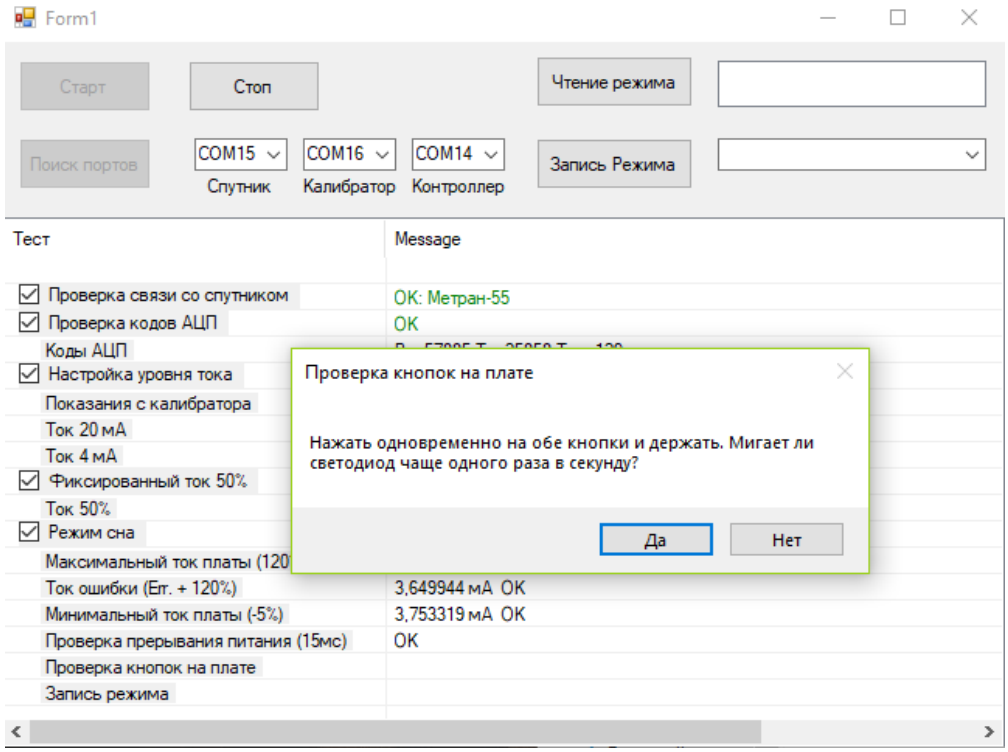

Рисунок 3.4 – Проверка исправности кнопок на плате

Успешно пройдя все тесты, программа записывает датчику необходимый режим и уведомляет оператора об успешном завершении процесса калибровки.

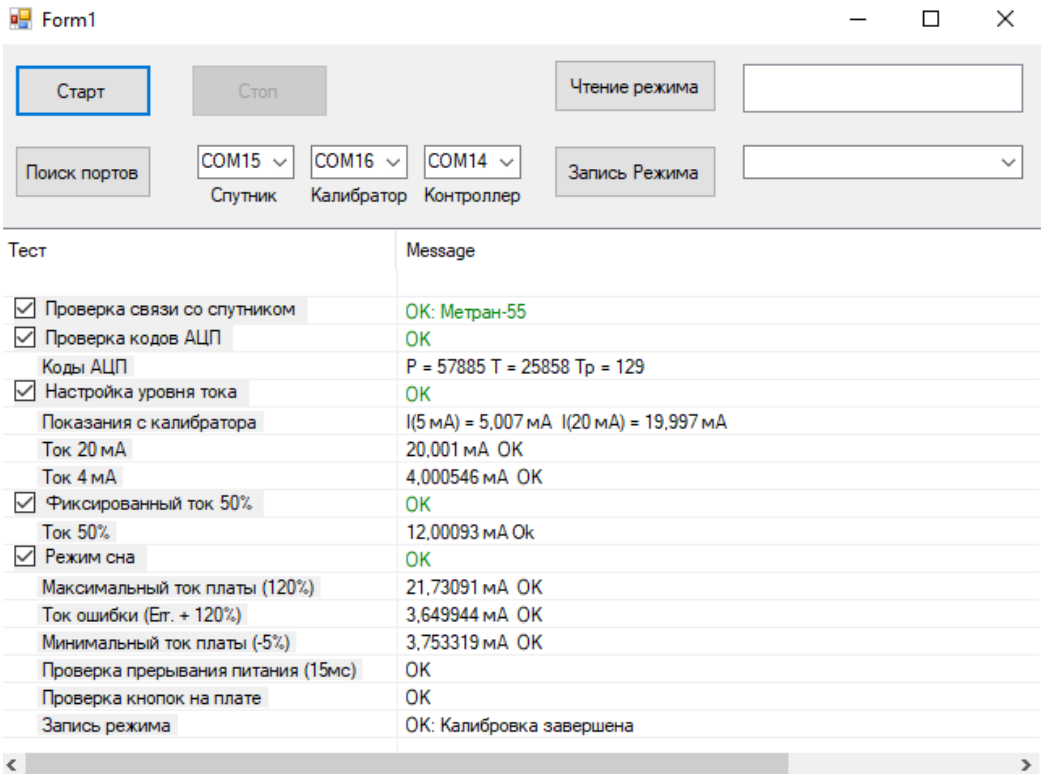

Рисунок 3.5 – Завершение калибровки

В случае обнаружения неисправности датчика или выявления нарушения процесса калибровки, программа останавливает свою работу, выделив красным цветом тот участок проверки, на котором была обнаружена ошибка. Программа так же останавливает процесс калибровки и оповещает оператора о неисправности с помощью всплывающего окна, если связь с одним из внешних устройств была потеряна.

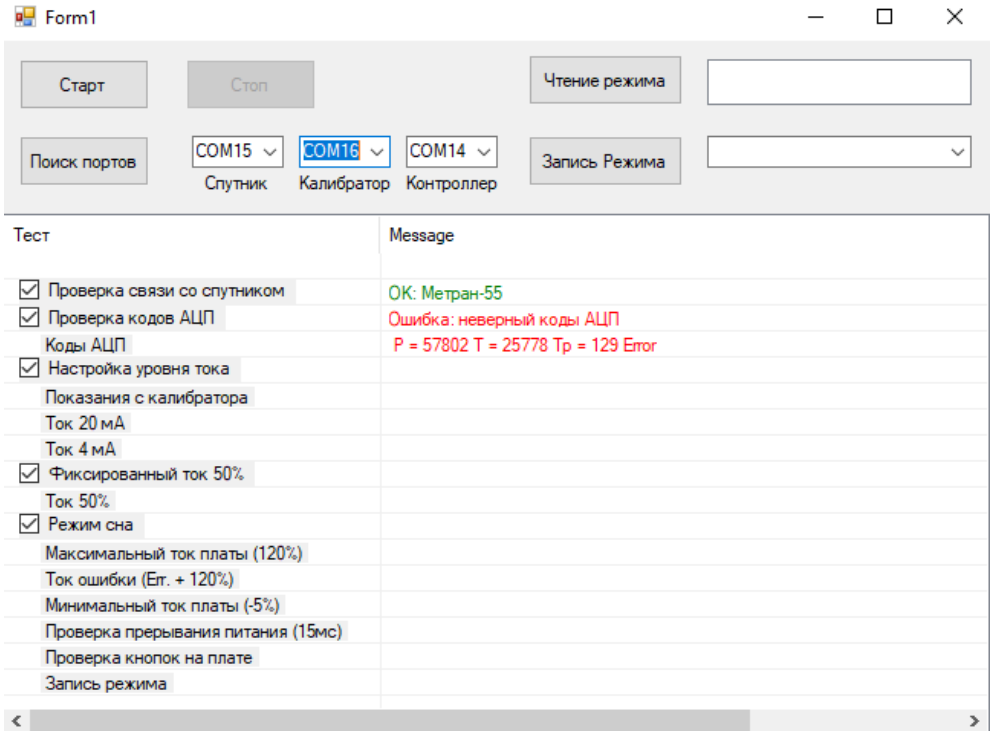

Рисунок 3.6 – Выявление неисправности программой

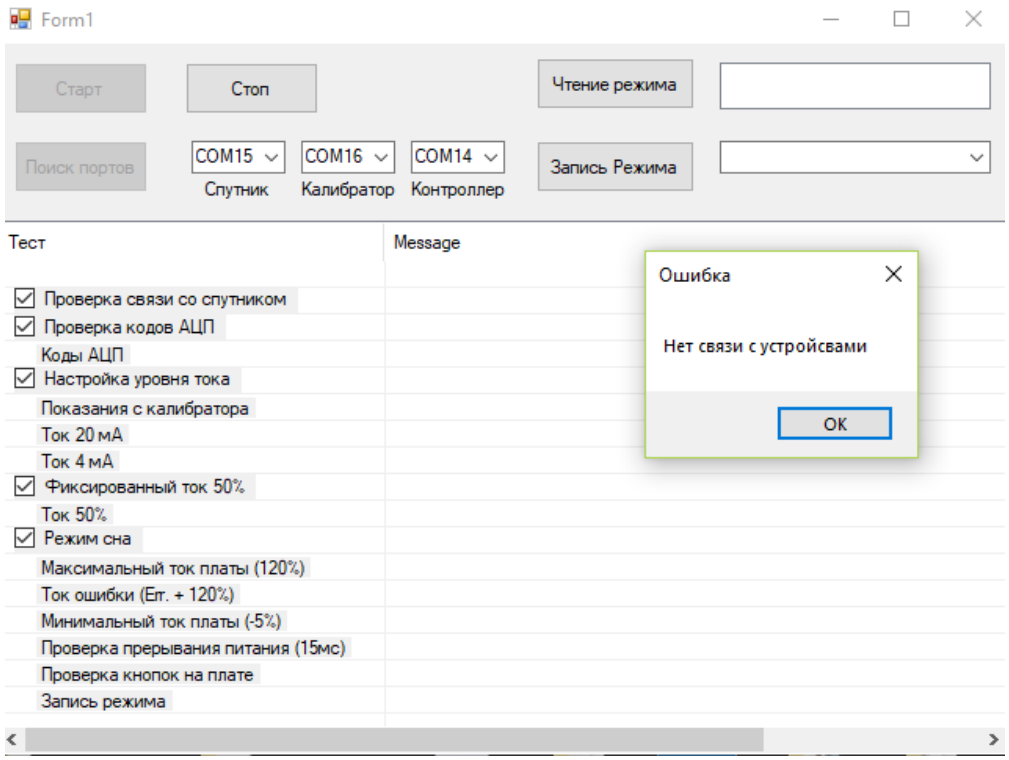

Рисунок 3.7 – Потеря связи с одним из внешних устройств

Процесс калибровки и проверки датчика так же может быть принудительно остановлен оператором с помощью кнопки «Отмена».

Программное обеспечение обладает рядом дополнительных функций: возможность прочитать и записать необходимый режим, в котором работает датчик; возможность производить не всю проверку параметров платы датчика, а только некоторые ее части. Предусмотрена кнопка автоматического поиска портов.

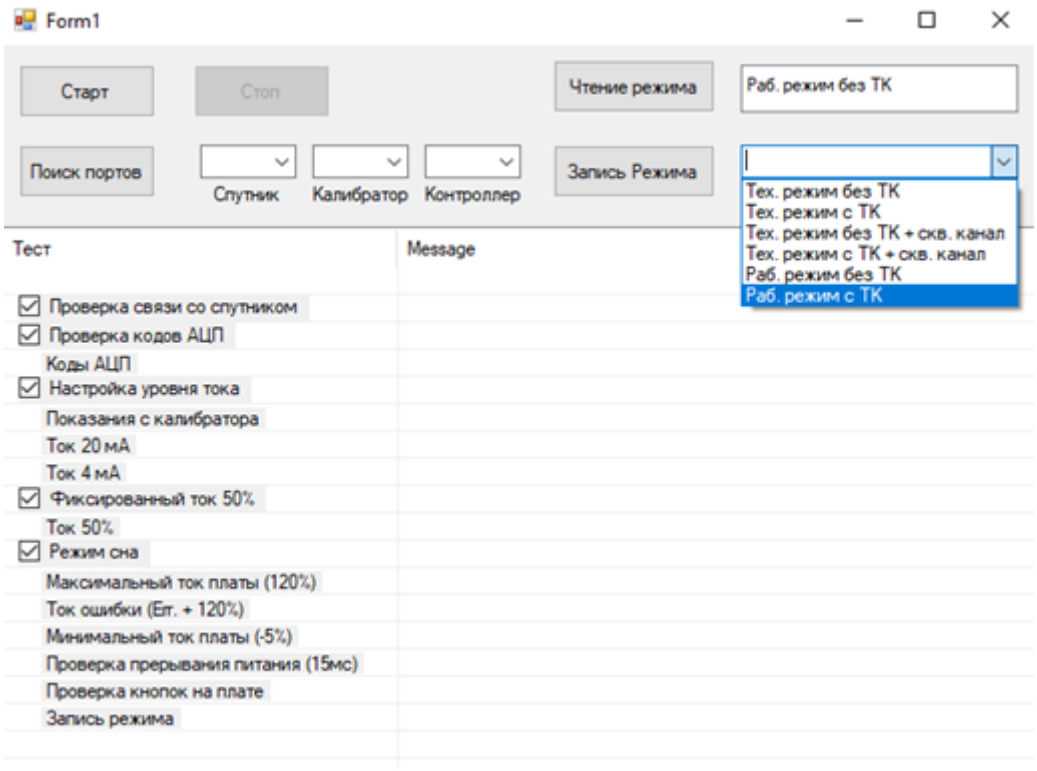

Рисунок 3.8 – Чтение и запись режима датчика

4 ПРАКТИЧЕСКАЯ НАЧИМОСТЬ РАБОТЫ

4.1 Внедрение автоматизированной системы на предприятие

После внедрения автоматизированной системы была проведена калибровка партии плат датчиков давления Метран-55. После калибровки были произведены замеры токовых сигналов на 4 и 20 мА. Так же была подсчитана абсолютная погрешность измерений по формуле:

$$\Delta I = I_{из} - I_{эт}, \quad (4.1)$$

где $I_{из}$ – измеренное значение величины тока в мА;

$I_{эт}$ – эталонное значение величины тока в мА.

Результаты измерений и величину абсолютной погрешности можно видеть на рисунке 4.1.

№	4мА	Абсолютная погрешнось (4 мА)	20 мА	Абсолютная погрешнось (20 мА)
1	4,0023	0,0023	20,0010	0,0010
2	4,0028	0,0028	19,9980	-0,0020
3	Неисправность			
4	4,0003	0,0003	20,0022	0,0022
5	4,0002	0,0002	20,0028	0,0028
6	4,0032	0,0032	20,0027	0,0027
7	4,0003	0,0003	20,0001	0,0001
8	3,9999	-0,0001	20,0012	0,0012
9	4,0023	0,0023	20,0004	0,0004
10	4,0021	0,0021	20,0001	0,0001
11	4,0012	0,0012	19,9990	-0,0010
12	3,9987	-0,0013	19,9999	-0,0001
13	4,0021	0,0021	19,9970	-0,0030
14	4,0004	0,0004	20,0002	0,0002
15	4,0026	0,0026	20,0004	0,0004
16	3,9981	-0,0019	20,0013	0,0013
17	4,0017	0,0017	20,0002	0,0002
18	3,9979	-0,0021	20,0028	0,0028
19	3,9987	-0,0013	19,9982	-0,0018
20	4,0019	0,0019	20,0002	0,0002

Рисунок 4.1 – Результаты измерений откалиброванных датчиков

Для исследования корректности работы автоматизированной системы была использована партии из 20 датчиков. По условию технического задания необходимо обеспечить абсолютную погрешность не более ±0.003 мА. Как видно из рисунка 4.1 только у одного датчика данный параметр выходи за границы. Для этого датчика была проведена повторная настройка и абсолютная погрешность улучшилась.

У датчика №3 была выявлена неисправность в виде неверных кодов АЦП. Программное обеспечение, обнаружив данную ошибку, оповестила оператора, и дальнейшая калибровка была остановлена.

Внедрение автоматизированной системы позволило уменьшить время калибровки единицы продукции с трех до одной минуты. Так же удалось свести участи человека в данном процессе к минимуму. Была написана инструкция по пользованию данной системой для сотрудников.

4.2 Расчёт экономического эффекта

В год производиться около 4000 плат датчиков давления Метран-55. Соответственно внедряемая система позволит сократить 134 часа на калибровку в год. Критерием эффективности создания и внедрения новых средств автоматизации является ожидаемый экономический эффект. Он определяется по формуле:

$$Э = Э_р - Е_н \cdot К_п, \quad (4.1)$$

где $Э_р$ – годовая экономия;

$Е_н$ – нормативный коэффициент (равен 0.24 для приборостроительной отрасли);

$К_п$ = 3 000 руб. – капитальные затраты на проектирование и внедрение.

Годовая экономия $Э_р$ складывается из экономии эксплуатационных расходов и экономии в связи с повышением производительности труда. Таким образом, получаем:

$$Э_р = (Р_1 - Р_2) + \Delta Р_п, \quad (4.2)$$

где Р1 и Р2 – соответственно эксплуатационные расходы до и после внедрения системы;

$\Delta Р_п$ – экономия от повышения производительности труда.

Эксплуатационные расходы системы калибровки плат датчиков давления Метран-55 до и после автоматизации остаются неизменными. Рассчитаем экономию от повышения производительности труда по формуле:

$$\Delta Р_п = Z_п \cdot \left(\frac{\Delta Т}{Т - \Delta Т}\right), \quad (4.3)$$

где $Z_п$ = 420 000 (руб.) – заработная плата сотрудника за год;

$\Delta Т$ = 134 часов – экономия времени в год;

Т = 1970 часов – время работы сотрудника год.

$$\Delta Р_п = 420000 \cdot \left(\frac{134}{1970 - 134}\right) \approx 30\,700 \text{ (руб.)}$$

Таким образом экономический эффект за один год составляет:

$$Э = 30\,700 - 0{,}24 \cdot 3\,000 \approx 30\,000 \text{ (руб.)}.$$

4.3 Перспективы развития автоматизированной системы калибровки

Стенд настройки является универсальным для различных типов датчиков. Проделанная работа по модернизации данного пульта позволяет в дальнейшем, изменив программное обеспечение на ПК, так же автоматизировать процесс калибровки датчиков в исполнении 0-5 мА (СПГК.5175.152), что даст дополнительный экономический эффект.

В дальнейшем так же планируется добавить в приложение функцию автоматического поиска портов.

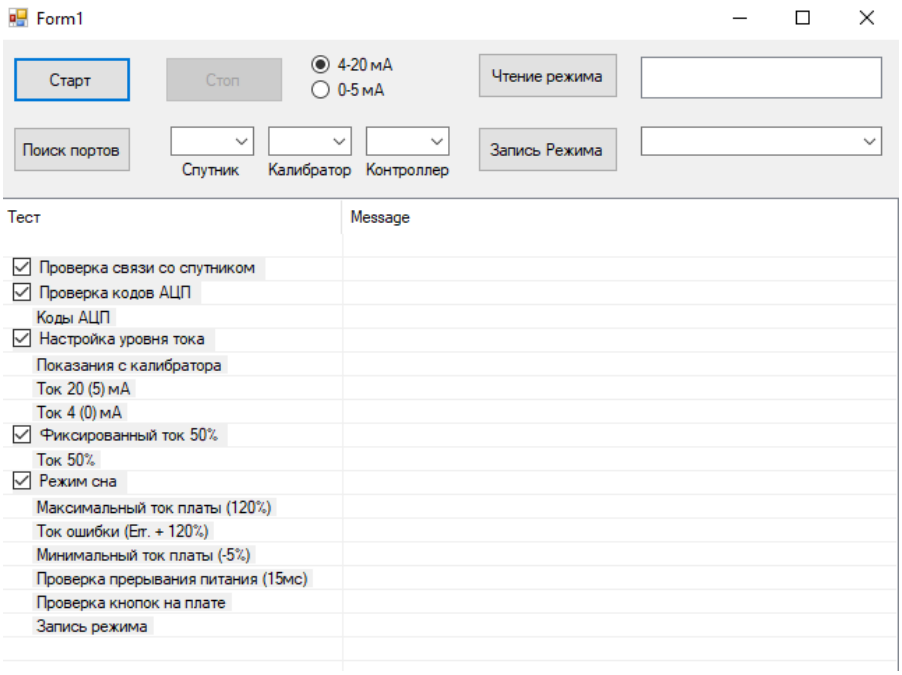

Рисунок 4.2 – Возможность выбирать тип исполнения калибруемого датчика

ЗАКЛЮЧЕНИЕ

В данной дипломной работе был автоматизирован процесс калибровки плат датчиков давления Метран-55. Для реализации данной задачи были исследованы возможности данной установки, разработан способ автоматизации процесса, разработана электрическая схема, разработано программное обеспечение для микроконтроллера и ПК. Написана инструкция для использования данной системы.

После внедрения автоматизированной системы на производство и её тестирование на партии датчиков, можно сказать, что система полностью удовлетворяет требования технического задания. Абсолютная погрешность токовых сигналов откалиброванных датчиков не превышает ±0.003 мА. Максимальный, минимальный и 50% ток настроенных датчиков не выходят за границы допустимых диапазонов. Так же удалось сократить время калибровки одной платы с трех минут до одной, что дало производству дополнительный экономический эффект. Участие человека в процессе настройки датчика сведено к минимуму.

Проделанная работа позволяет в дальнейшем автоматизировать процесс настройки плат датчиков в исполнении 0/5 мА.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Аристова Н.И., Корнеева А.И. Промышленные программно-аппаратные средства на отечественном рынке АСУ ТП. – М.: ООО «Изд-во Научтехлитиздат», 2001. – 402 с.

2 Белов, А.В. Программирование микроконтроллер для начинающих и не только: учебник / А.В. Белов. – Москва: Изд-во: НиТ, 2016. – 352 с.

3 Бесекерский, В. А. Теория систем автоматического управления / В. А. Бесекерский, Е. П. Попов. – М.: Наука 1975. – 768 с.

4 Бессонов Л.А. Теоретические основы электротехники. Электрические цепи / Л.А. Бессонов. – М.: Гардарики, 2002. – 638 с.

5 Биллиг, В.А. Основы программирования на C# / В.А Биллиг. – М.: Бином. Лаборатория знаний, 2006. – 483 с.

6 Биллиг, В.А. Основы программирования на C#: Учебное пособие / В.А. Биллиг. – М.: Бином, 2012. – 483 с.

7 Бродин В. Б., Шагурин И. И. Микроконтроллеры. Архитектура, программирование, интерфейс. – М.: Издательство ЭКОМ, 1999. – 400с.: ил.

8 Волков Н.И. Электромашинные устройства автоматики / Н.И. Волков, В.П. Миловзоров – М: Изд-во «Высшая школа», 1986. – 327 с.

9 ГОСТ 19.104-78. Пояснительная записка. Требования к содержанию и оформлению. – М.: Стандартинформ, 2010. – 40 с.

10 ГОСТ 19.404-79. Основные надписи. – М.: Стандартинформ, 2010. – 4 с.

11 ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. – М.: Изд-во стандартов, 2003. – 24 с.

12 ГОСТ 2.104-2006. Основные надписи. – М.: Стандартинформ, 2006. – 15 с.

13 ГОСТ 2.701-2008. Схемы. Виды и типы. Общие требования к выполнению. – М.: Стандартинформ, 2009. – 15 с.

14 ГОСТ 21.404-85. Обозначения условные приборов и средств автоматизации в схемах. – М.: Стандартинформ, 2009. – 9 с.

15 ГОСТ 21.408-2013. Правила выполнения рабочей документации автоматизации технологических процессов. – М.: Стандартинформ, 2014. – 41 с.

16 ГОСТ 24.302-80. Система технической документации на АСУ. Общие требования к выполнению схем. – М.: Стандартинформ, 2009 – 4 с.

17 ГОСТ 34.201-89. Виды, комплектность и обозначение документов при создании автоматизированных систем. – М.: Изд-во, 2002. – 11 с.

18 ГОСТ 7.9-95. Реферат и аннотация. Общие требования. – М.: Стандартинформ, 2009. – 8 с.

19 ГОСТ 8.417-2002. Единицы величин. – М.: Стандартинформ, 2009. – 24 с.

20 ГОСТ 19.002-80. ЕСПД. Схемы алгоритмов и программ. Правила выполнения. – М.: Изд-во стандартов, 1981. – 9 с.

21 ГОСТ 19.003-80. ЕСПД. Схемы алгоритмов и программ. Обозначение условные графические. – М.: Изд-во стандартов, 1981. – 9 с.

22 ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. – М.: Изд-во стандартов, 1992. – 22 с.

23 Денисенко, В.В. Компьютерное управление технологическим процессом, экспериментом, оборудованием. – М.: Горячая линия-Телеком, 2009. – 608 с.

24 Зевеке Г.В. Основы теории цепей / Г.В. Зевеке, П.А. Ионкин, А.В. Нетушил, – М.: Энергия, 1975. – 752 с.

25 Зельдович Я.Б. Высшая математика для начинающих / Я.Б. Зельдович. – М.: Физмалит, 1963.– 784 с.

26 Зиборов, В.В. Visual C# 2012 на примерах / В.В. Зиборов. – М.: БХВ-Петербург, 2013. – 480 с.

27 Казаринов, Л.С. Автоматизированные информационно-управляющие системы: учебное пособие / Л.С. Казаринов, Д.А. Шнайдер, Т.А. Барбасова. – Челябинск: ЮУрГУ, 2008. – 296 с.

28 Касаткин А.С., Немцов М.В. Электротехника: Учеб. для вузов. 6-е изд., перераб. – М.: Высш. шк., 2000. – 542 с.

29 Класс SerialPort [Электронный ресурс]. Режим доступа: https://msdn.microsoft.com/ru-ru/library/system.io.ports.serialport(v=vs.110).aspx. – (дата обращения 08.04.2018).

30 Константинов В.И.Электроника. Часть 1: Полупроводниковые приборы: конспектлекций / В.И. Константинов, О.В. Константинова, Е.В. Вставская. – Челябинск: Издательский центр ЮУрГУ, 2010. – 79 с.

31 Копылов И.П. Справочник по электрическим машинам: в 2 т. / И.П. Копылов, Б.К. Клоков – М.: Энергоатомиздат, 1988. – Т.1 – 456 с.

32 Копылов И.П. Справочник по электрическим машинам: в 2 т. / И.П. Копылов, Б.К. Клоков – М.: Энергоатомиздат, 1988. – Т.2 – 688 с.

33 Краснов М.Л. Вся высшая математика. / М.Л. Краснов, А.И. Киселев. – М.: Едиториал УРСС, 2003. – Т.1 – 336 с.

34 Краснов М.Л. Вся высшая математика. / М.Л. Краснов, А.И. Киселев. – М.: Едиториал УРСС, 2003. – Т.2 – 294 с.

35 Краснов М.Л. Вся высшая математика. / М.Л. Краснов, А.И. Киселев. – М.: Едиториал УРСС, 2003. – Т.3 – 482 с.

36 Кругляк К. Промышленные сети: Цели и средства // Современные технологии автоматизации. – 2002. №4. с. 6-17.

37 Курсовое и дипломное проектирование. Общие требования к содержанию и оформлению / Т.И. Парубочая, Н.В. Сырейщикова, В.И. Гузеев, Л.В. Винокурова. – Челябинск: Изд. ЮУрГУ, 2008. – 56 с

38 Лабораторный практикум по изучению микроконтроллеров STM32 на базе отладочного модуля STM32F3 Discovery / Бугаев В.И., Мусиенко М.П., Крайнык Я.М. – Москва-Николаев: МФТИ-ЧГУ, 2014. – 33 с.

39 Макаров, И.М. Линейные автоматические системы / И.М. Макаров, Б.М. Менский – М.: Машиностроение 1982. – 505 с.

40 Метран [Электронный ресурс]. Режим доступа: http://www2.emersonprocess.com/. – (дата обращения 09.04.2018).

41 Мирошник И.В. Теория автоматического управления: линейные системы / И.В. Мирошник. – СПб.: Питер, 2005. – 337 с.

42 Новиков, Ю.В. Основы микропроцессорной техники: учебное пособие/ Ю.В. Новиков, П.К. Скоробогатов. – 4-е изд., испр. – М.: Интернет-Университет Информационных Технологий; Бином. Лаборатория знаний, 2009. – 357 с.

43 Никитин В.А. Методы и средства измерений, испытаний и контроля / В.А. Никитин, С.В. Бойко – Оренбург: Изд-во ГОУ ОГУ, 2004. – 474 с

44 ООО «ЭлМетро»: О компании [Электронный ресурс]. Режим доступа: http://www.elmetro.ru/. – (дата обращения 23.05.2018).

45 Пантелеев В.Н., Прошин В.М.. Основы автоматизации производства. - М.: Издательский центр «Академия», 2008. – 192 с.

46 Петраков, Ю.В. Теория автоматического управления технологическими системами: учебное пособие для студентов вузов / Ю.В. Петраков, О.И. Драчев – М.: Машиностроение 2008. – 336 с.

47 Приемопередатчик RS-232. [Электронный ресурс]. Режим доступа: https://ic.milandr.ru/products/interfeysnye_mikroskhemy/rs232/5559in4u/. – (дата обращения 12.05.2018).

48 Проектирование микропроцессорных систем: учебно-методическое пособие к выполнению лабораторных работ / сост. А.С. Карпенков; М-во образования и науки Российской Федерации, Федеральное гос. бюджетное образовательное учреждение высш. образования «Ковровская гос. технол. акад. им. В. А. Дегтярева». – Ковров: Ковровская гос. технологическая акад. им. В. А. Дегтярева, 2016. - 47 с.

49 Профос, П. Измерения в промышленности. Справочник. Кн. 2. / П. Профос. – М.: Изд-во Металлургия, 1990. – 492 с.

50 Сивухин Д.В. Общий курс физики. Т. 1. Механика / Д.В. Сивухин – М.: Изд-во Физматлит, 2005. – 560 с.

51 Смирнов, В.Г. Стандартизация и качество продукции : [учеб. пособие для сред.-спец. образования] / В.Г. Смирнов, М.С. Капица, И.Э. Чиркун. – Минск : Республиканский институт профессионального образования,2013.302 с.

52 Солодовников, В.В. Основы автоматического регулирования теория / В.В. Солодовников, М.А. Айзерман и др. – М.: МАШГИЗ Государственное научно-техническое издательство машиностроительной литературы 1954. – 1144 с

53 Тензо-М: тензодатчики [Электронный ресурс]. Режим доступа: https://www.tenso-m.ru/f/catalog/pdf/pdf_178.pdf – (дата обращения 30.05.2018)

54 Фудзисава, Юкихо 32-битные микропроцессоры и микроконтроллеры SuperH / Юкихо Фудзисава. - М.: Додэка XXI, 2009. - 360 с.

55 Ю Дж. Ядро Cortex-M3 компании ARM. Полное руководство /Дж. Ю. М. : Додэка-XXI, 2012. 552 с.

ПРИЛОЖЕНИЕ А

Листинг А.1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;
using System.IO;

namespace WindowsFormsApp4
{
    public partial class Form1 : Form
    {
        public static float P_CodeMin, P_CodeMax;
        public static float T_CodeMin, T_CodeMax;
        public static float TpCodeMin, TpCodeMax;
        public static float I4Max, I4Min;
        public static float I5Max, I5Min;
        public static float I20Max, I20Min;
        public static float I50Max, I50Min;
        public static float I120Max, I120Min;
        public static float IerrMax, IerrMin;

        ListViewItem Row1 = new ListViewItem();
        ListViewItem Row2 = new ListViewItem();
        ListViewItem Row3 = new ListViewItem();
        ListViewItem Row4 = new ListViewItem();
        ListViewItem Row5 = new ListViewItem();
        ListViewItem Row6 = new ListViewItem();
        ListViewItem Row7 = new ListViewItem();
        ListViewItem Row8 = new ListViewItem();
        ListViewItem Row9 = new ListViewItem();
        ListViewItem Row10 = new ListViewItem();
        ListViewItem Row11 = new ListViewItem();
        ListViewItem Row12 = new ListViewItem();
        ListViewItem Row13 = new ListViewItem();
        ListViewItem Row14 = new ListViewItem();
        ListViewItem Row15 = new ListViewItem();
        ListViewItem Row16 = new ListViewItem();
        ListViewItem Row17 = new ListViewItem();

        /* Функция для считывания необходимых параметров с текстового документа */

        public static void ReadingParameters()
        {
            string[] param;
            param = new string[18];
            string file = @"C:/Users/Михаил/Desktop/диплом/Готовая версия без 0-
5/param.txt";
            using (StreamReader fstream = new StreamReader(file,
System.Text.Encoding.Default))
            {
                string line;
                for (int i = 0; i < 18; i++)
                {
                    line = fstream.ReadLine();
                    param[i] = line.Substring(10);
                }
            }
            P_CodeMax = Convert.ToSingle(param[0]);
            P_CodeMin = Convert.ToSingle(param[1]);
```

```
            T_CodeMax = Convert.ToSingle(param[2]);
            T_CodeMin = Convert.ToSingle(param[3]);
            TpCodeMax = Convert.ToSingle(param[4]);
            TpCodeMin = Convert.ToSingle(param[5]);
            I4Max = Convert.ToSingle(param[6]);
            I4Min = Convert.ToSingle(param[7]);
            I5Max = Convert.ToSingle(param[8]);
            I5Min = Convert.ToSingle(param[9]);
            I20Max = Convert.ToSingle(param[10]);
            I20Min = Convert.ToSingle(param[11]);
            I50Max = Convert.ToSingle(param[12]);
            I50Min = Convert.ToSingle(param[13]);
            I120Max = Convert.ToSingle(param[14]);
            I120Min = Convert.ToSingle(param[15]);
            IerrMax = Convert.ToSingle(param[16]);
            IerrMin = Convert.ToSingle(param[17]);

        }

        /* Инициализация компонентов, поиск портов */

        public Form1()
        {
            InitializeComponent();
            findPorts();
            Stop.Enabled = false;
            ReadingParameters();
            Start.Focus();
        }

        /* Функция очистки экрана */

        public void Clear()
        {
            Row1.Remove();
            Row2.Remove();
            Row3.Remove();
            Row4.Remove();
            Row5.Remove();
            Row6.Remove();
            Row7.Remove();
            Row8.Remove();
            Row9.Remove();
            Row10.Remove();
            Row11.Remove();
            Row12.Remove();
            Row13.Remove();
            Row14.Remove();
            Row15.Remove();
            Row16.Remove();
            Row17.Remove();
            listView1.Items.Clear();
        }

        /* функции для работы со спутником */

        public static void PortSputnikOpen(string SputnikPortName)
        {

            PortSputnik.PortName = SputnikPortName;
            PortSputnik.BaudRate = 1200;
            PortSputnik.Parity = Parity.Odd;
            PortSputnik.StopBits = StopBits.One;
            PortSputnik.DataBits = 8;
            PortSputnik.RtsEnable = true;
            PortSputnik.DtrEnable = false;
            PortSputnik.ReadTimeout = 5000;
            PortSputnik.Open();
        }
        public static void PortSputnikClose()
```

```
{
    PortSputnik.Close();
}
public static byte SputnikId()
{
    byte[] data;
    data = new byte[1];
    byte[] SignalId = { 0x0F };
    try
    {
        for (int i = 0; i < 3; i++)
        {
            PortSputnik.Write(SignalId, 0, SignalId.Length);
            PortSputnik.Read(data, 0, data.Length);
        }
    }
    catch (TimeoutException)
    {
        byte error = 0x00;
        return error;
    }
    return data[0];
}
public static float SputnikPressureCode()
{
    byte[] PressureCode;
    byte[] data;
    PressureCode = new byte[2];
    data = new byte[1];
    byte[] SignalPressureCode = { 0x0B };
    for (int i = 0; i < 4; i++)
    {
        PortSputnik.Write(SignalPressureCode, 0, SignalPressureCode.Length);
        PortSputnik.Read(data, 0, data.Length);
        if (i == 1)
        {
            PressureCode[1] = data[0];
        }
        if (i == 2)
        {
            PressureCode[0] = data[0];
        }
    }
    float PressureCodeFloat;
    PressureCodeFloat = BitConverter.ToChar(PressureCode, 0);
    return PressureCodeFloat;
}
public static float SputnikBoardTermoCode()
{
    byte[] BoardTermoCode;
    byte[] data;
    BoardTermoCode = new byte[2];
    data = new byte[1];
    byte[] SignalBoardTermoCode = { 0x0C };
    for (int i = 0; i < 4; i++)
    {
        PortSputnik.Write(SignalBoardTermoCode, 0, SignalBoardTermoCode.Length);
        PortSputnik.Read(data, 0, data.Length);
        if (i == 1)
        {
            BoardTermoCode[1] = data[0];
        }
        if (i == 2)
        {
            BoardTermoCode[0] = data[0];
        }
    }
    float FloatBoardTermoCode;
    FloatBoardTermoCode = BitConverter.ToChar(BoardTermoCode, 0);
    return FloatBoardTermoCode;
```

```
        }
        public static float SputnikSensorTermoCode()
        {
            byte[] SensorTermoCode;
            byte[] data;
            SensorTermoCode = new byte[2];
            data = new byte[1];
            byte[] SignalSensorTermoCode = { 0x0A };
            for (int i = 0; i < 4; i++)
            {
                PortSputnik.Write(SignalSensorTermoCode, 0, SignalSensorTermoCode.Length);
                PortSputnik.Read(data, 0, data.Length);
                if (i == 1)
                {
                    SensorTermoCode[1] = data[0];
                }
                if (i == 2)
                {
                    SensorTermoCode[0] = data[0];
                }
            }
            float FloatSensorTermoCode;
            FloatSensorTermoCode = BitConverter.ToChar(SensorTermoCode, 0);
            return FloatSensorTermoCode;
        }
        public static byte[] SputnikRead(byte adres, byte bank, byte size, int n) //n -
количесво эементов для считывания
        {
            byte[] signal;
            signal = new byte[n + 5];
            signal[0] = 0x01;
            signal[1] = bank;
            signal[2] = adres;
            signal[3] = size;
            for (int i = 4; i < n + 5; i++)
            {
                signal[i] = 0x01;
            }
            byte[] tape;
            tape = new byte[n + 5];
            for (int i = 0; i < n + 5; i++)
            {
                PortSputnik.Write(signal, i, 1);
                PortSputnik.Read(tape, i, 1);
            }
            byte[] Rom;
            Rom = new byte[n];
            for (int i = 0; i < n; i++)
            {
                Rom[i] = tape[i + 4];
            }
            return Rom;
        }
        public static byte[] SputnikWrite(byte adres, byte bank, byte[] date, byte size,
int n)
        {
            byte[] signal;
            signal = new byte[n + 5];
            signal[0] = 0x02;
            signal[1] = bank;
            signal[2] = adres;
            signal[3] = size;
            for (int i = 4; i < n + 4; i++)
            {
                signal[i] = date[i - 4];
            }
            signal[n + 4] = 0x02;
            byte[] tape;
            tape = new byte[n + 5];
            for (int i = 0; i < n + 5; i++)
```

```
        {
            PortSputnik.Write(signal, i, 1);
            PortSputnik.Read(tape, i, 1);
        }
        return tape;

    }
    public static void SputnikSensorOff()
    {
        byte[] SignalSensorOff = { 0x16 };
        byte[] data;
        data = new byte[1];
        for (int i = 0; i < 2; i++)
        {
            PortSputnik.Write(SignalSensorOff, 0, SignalSensorOff.Length);
            PortSputnik.Read(data, 0, data.Length);
        }
    }
    public static void SputnikSensorOn()
    {
        byte[] SignalSensorOn = { 0x15 };
        byte[] data;
        data = new byte[1];
        for (int i = 0; i < 2; i++)
        {
            PortSputnik.Write(SignalSensorOn, 0, SignalSensorOn.Length);
            PortSputnik.Read(data, 0, data.Length);
        }
    }
    public static void SputnikSleep()
    {
        byte[] SignalSputnikSleep = { 0x17 };
        byte[] data;
        data = new byte[1];
        for (int i = 0; i < 2; i++)
        {
            PortSputnik.Write(SignalSputnikSleep, 0, SignalSputnikSleep.Length);

            PortSputnik.Read(data, 0, data.Length);
        }
    }
    public static byte SputnikCheckPower()
    {
        byte[] SignalCheckPower = { 0x0E };
        byte[] Power;
        byte[] data;
        Power = new byte[2];
        data = new byte[1];
        for (int i = 0; i < 3; i++)
        {
            PortSputnik.Write(SignalCheckPower, 0, SignalCheckPower.Length);
            PortSputnik.Read(data, 0, data.Length);
            if (i == 1 || i == 2)
            {
                Power[i - 1] = data[0];
            }
        }
        return Power[1];
    }

    /* функции для работы с калибратором */

    public static void PortCalibratorOpen(string CalibratorPortName)
    {
        PortCalibrator.PortName = CalibratorPortName;
        PortCalibrator.BaudRate = 9600;
        PortCalibrator.Parity = Parity.None;
        PortCalibrator.StopBits = StopBits.One;
        PortCalibrator.DataBits = 8;
        PortCalibrator.DtrEnable = false;
```

```
            PortCalibrator.RtsEnable = true;
            PortCalibrator.ReadTimeout = 5000;
            PortCalibrator.Open();
        }
        public static void PortCalibratorClose()
        {
            PortCalibrator.Close();
        }
        public static byte UnlockCalibrator()
        {
            byte[] signal;
            signal = new byte[1];
            byte[] data;
            data = new byte[1];
            byte[] SignalUnlockCalibrator = { 0x5A, 0xA8, 0x56, 0x48, 0xC1, 0x16, 0xAD,
0xF2 };
            for (int i = 0; i < 8; i++)
            {
                signal[0] = SignalUnlockCalibrator[i];
                PortCalibrator.Write(signal, 0, signal.Length);
            }
            PortCalibrator.Read(data, 0, data.Length);
            return data[0];
        }
        public static float MeasureCurrentCalibrator()
        {
            byte[] temp1;
            temp1 = new byte[7];
            byte[] data;
            data = new byte[1];
            byte[] SignalMeasureCurrentCalibrator = { 0x28 };
            for (int i = 0; i < 7; i++)
            {
                PortCalibrator.Write(SignalMeasureCurrentCalibrator, 0,
SignalMeasureCurrentCalibrator.Length);
                PortCalibrator.Read(data, 0, data.Length);
                temp1[i] = data[0];
            }
            float FloatMeasureCurrentCalibrator;
            byte[] temp2 = { temp1[2], temp1[3], temp1[4], temp1[5] };
            FloatMeasureCurrentCalibrator = BitConverter.ToSingle(temp2, 0);
            FloatMeasureCurrentCalibrator = Math.Abs(FloatMeasureCurrentCalibrator);
            return FloatMeasureCurrentCalibrator;
        }

        /* функции для работы с микроконтроллером */

        public static void PortMicroOpen(string MicroPortName)
        {
            PortMicro.PortName = MicroPortName;
            PortMicro.BaudRate = 38400;
            PortMicro.Parity = Parity.None;
            PortMicro.StopBits = StopBits.One;
            PortMicro.DataBits = 8;
            PortMicro.DtrEnable = false;
            PortMicro.RtsEnable = true;
            PortMicro.Open();
        }
        public static void PortMicroClose()
        {
            PortMicro.Close();
        }
        public static void Micro(byte a)
        {
            byte[] Signal;
            Signal = new byte[1];
            byte[] SignalMicro = { 0x30, a, 0x0A, 0x0A };
            for (int i = 0; i < 4; i++)
            {
                Signal[0] = SignalMicro[i];
```

```
            PortMicro.Write(Signal, 0, Signal.Length);
        }
    }

    /* функция для перевода бат в float */

    public static float OldByteToFloat(byte b0, byte b1, byte b2, byte b3)
    {
        float f;
        byte[] b;
        b = new byte[4];
        byte a, c;
        b[0] = b3;
        b[1] = b2;
        b[2] = b1;
        b[3] = b0;
        if ((b[3] == 0) || (b[3] == 1)) ;
        else
            b[3] -= 2;
        a = (byte)(b[2] & 0x80);
        c = (byte)(b[3] & 0x1);
        b[3] = (byte)(b[3] >> 1);
        b[3] = (byte)(b[3] + a);
        b[2] = (byte)((b[2] & 0x7f) + (c << 7));
        f = BitConverter.ToSingle(b, 0);
        return f;
    }

    /* функция для перевода float в байт */

    public static byte[] OldFloatToByte(float f)
    {
        byte[] b;
        b = new byte[4];
        byte a, c;
        byte[] FMCFloat;
        FMCFloat = new byte[4];
        b = BitConverter.GetBytes(f);
        a = (byte)((b[2] & 0x80) >> 7);
        c = (byte)(b[3] & 0x80);
        b[3] = (byte)((b[3] << 1) + a + 0);

        if ((b[3] == 0) || (b[3] == 0xfe) || (b[3] == 0xff)) ;
        else
            b[3] += 2;
        b[2] = (byte)(c + (b[2] & 0x7f));
        FMCFloat[0] = b[3];
        FMCFloat[1] = b[2];
        FMCFloat[2] = b[1];
        FMCFloat[3] = b[0];
        return FMCFloat;
    }

    /* порт калибратора, спутника, микроконтроллера  */

    static SerialPort PortCalibrator = new SerialPort();
    static SerialPort PortSputnik = new SerialPort();
    static SerialPort PortMicro = new SerialPort();

    /* функция для поиска портов для comboBox */

    private void findPorts()
    {
        string[] ports = SerialPort.GetPortNames();
        comboBox1.Items.AddRange(ports);
        comboBox2.Items.AddRange(ports);
        comboBox3.Items.AddRange(ports);
    }

    /* Обработка нажатия кнопки, для чтения режима */
```

```
        private void Read_Click(object sender, EventArgs e)
        {
            /*
            Тех. режим без ТК                0x20
            Тех. режим с ТК                  0x22
            Тех. режим без ТК + скв. канал   0x08
            Тех. режим с ТК + скв. канал     0x0A
            Раб. режим без ТК                0x00
            Раб. режим с ТК                  0x02
            */
            if (comboBox2.Text != "")
            {
                if (!PortSputnik.IsOpen)
                {
                    PortSputnikOpen(comboBox2.Text);
                    Thread.Sleep(1000);
                }
                try
                {
                    listBox2.Items.Clear();
                    byte[] FTempModeR = SputnikRead(0x15, 0x00, 1, 1);
                    switch (FTempModeR[0])
                    {
                        case 0x20: listBox2.Items.Add("Тех. режим без ТК"); break;
                        case 0x22: listBox2.Items.Add("Тех. режим с ТК"); break;
                        case 0x08: listBox2.Items.Add("Тех. режим без ТК + скв. канал");
break;
                        case 0x0A: listBox2.Items.Add("Тех. режим с ТК + скв. канал");
break;
                        case 0x00: listBox2.Items.Add("Раб. режим без ТК"); break;
                        case 0x02: listBox2.Items.Add("Раб. режим с ТК"); break;
                    }
                    PortSputnikClose();
                }
                catch (TimeoutException)
                {
                    listBox2.Items.Add("Ошибка: нет связи со спутником");
                }
            }
            else
            {
                listBox2.Items.Add("Ошибка: Порт не выбран");
            }
        }

        /* Обработка нажатия кнопки, для записи режима */

        private void Write_Click(object sender, EventArgs e)
        {
            byte[] FTempModeW;
            FTempModeW = new byte[1];
            if (comboBox4.Text != "")
            {
                if (comboBox2.Text != "")
                {
                    if (!PortSputnik.IsOpen)
                    {
                        PortSputnikOpen(comboBox2.Text);
                        Thread.Sleep(1000);
                    }
                    try
                    {
                        switch (comboBox4.Text)
                        {
                            case "Тех. режим без ТК": FTempModeW[0] = 0x20; break;
                            case "Тех. режим с ТК": FTempModeW[0] = 0x22; break;
                            case "Тех. режим без ТК + скв. канал": FTempModeW[0] = 0x08;
break;
```

```
                            case "Тех. режим с ТК + скв. канал": FTempModeW[0] = 0x0A;
break;
                            case "Раб. режим без ТК": FTempModeW[0] = 0x00; break;
                            case "Раб. режим с ТК": FTempModeW[0] = 0x02; break;
                        }
                        SputnikWrite(0x0015, 0x00, FTempModeW, 1, 1);
                        Thread.Sleep(1000);
                        byte[] FTempModeR = SputnikRead(0x15, 0x00, 1, 1);
                        if (FTempModeW[0] != FTempModeR[0])
                        {
                            comboBox4.Text = "Ошибка: не удалось записать режим";
                        }
                        else
                        {
                            comboBox4.Text = "Режим установлен";
                        }
                        PortSputnikClose();
                    }
                    catch (TimeoutException)
                    {
                        comboBox4.Text = "Ошибка: нет связи со спутником";
                    }
                }
                else
                {
                    comboBox4.Text = "Ошибка: Порт не выбран";
                }
            }
        }

        /* Автоматический поиск портов (не работает!) */

        private void button1_Click(object sender, EventArgs e)
        {
            /*
            comboBox1.Items.Clear();
            comboBox2.Items.Clear();
            comboBox3.Items.Clear();

            string[] ports = SerialPort.GetPortNames();

            PortSputnikOpen(ports[0]);
            if (SputnikId() == 55)  //если ports[0] это спутник
            {
                comboBox2.Items.Add(ports[0]);
                PortSputnikClose();
                PortMicroOpen(ports[1]);
                if(TestMicro() == 0x00)
                {
                    PortMicroClose();
                    comboBox3.Items.Add(ports[1]);
                    comboBox1.Items.Add(ports[2]);
                }
                else
                {
                    PortMicroClose();
                    PortMicroOpen(ports[2]);
                    if(TestMicro() == 0x00)
                    {
                       PortMicroClose();
                       comboBox3.Items.Add(ports[2]);
                       comboBox2.Items.Add(ports[1]);
                    }
                }
            }
            else
            {
                PortSputnikClose();
                PortSputnikOpen(ports[1]);
                if(SputnikId() == 55) //если ports[1] это спутник
```

```
            {
                comboBox2.Items.Add(ports[1]);
                PortSputnikClose();
                PortMicroOpen(ports[0]);
                if(TestMicro() == 0x00)
                {
                    PortMicroClose();
                    comboBox3.Items.Add(ports[0]);
                    comboBox1.Items.Add(ports[2]);
                }
                else
                {
                    PortMicroClose();
                    PortMicroOpen(ports[2]);
                    if (TestMicro() == 0x00)
                    {
                        PortMicroClose();
                        comboBox3.Items.Add(ports[2]);
                        comboBox2.Items.Add(ports[0]);
                    }
                }
            }
            else
            {
                PortSputnikClose();
                PortSputnikOpen(ports[2]);
                if(SputnikId() == 55) //если ports[2] это спутник
                {
                    comboBox2.Items.Add(ports[2]);
                    PortSputnikClose();
                    PortMicroOpen(ports[0]);
                    if (TestMicro() == 0x00)
                    {
                        PortMicroClose();
                        comboBox3.Items.Add(ports[0]);
                        comboBox1.Items.Add(ports[1]);
                    }
                    else
                    {
                        PortMicroClose();
                        PortMicroOpen(ports[1]);
                        if (TestMicro() == 0x00)
                        {
                            PortMicroClose();
                            comboBox3.Items.Add(ports[1]);
                            comboBox2.Items.Add(ports[0]);
                        }
                    }

                }
            }

        } */
    }

    /* Старт программы */

    private System.Threading.CancellationTokenSource _tokenSource;

    private void StartCalibration(System.Threading.CancellationToken cancelToken,
        IProgress<string> TRow2, IProgress<string> TRow3,
        IProgress<string> TRow4, IProgress<string> TRow5,
        IProgress<string> TRow6, IProgress<string> TRow7,
        IProgress<string> TRow8, IProgress<string> TRow9,
        IProgress<string> TRow10, IProgress<string> TRow11,
        IProgress<string> TRow12, IProgress<string> TRow13,
        IProgress<string> TRow14, IProgress<string> TRow15,
        IProgress<string> TRow16, IProgress<string> TRow17,
        bool CheckConnection, bool CheckADCCode, bool SetupCurrent,
        bool CheckFixedCurrent, bool CheckSleepMode)
```

```
{

    float corr_dac;
    float code_i0;
    float corr_i0;
    float corr_corr;
    float FM55CorrDAC;
    byte[] temp1;
    byte[] c1;
    byte[] c2;
    byte[] FTempModeW;
    c1 = new byte[1];
    c2 = new byte[1];
    FTempModeW = new byte[1];
    try
    {
        if (CheckConnection)
        {
            Micro(0x62);
            Thread.Sleep(1000);

            /* Проверка связи со спутником, считываем id спутника */
            for (int i = 0; i < 3; i++)
            {
                if (SputnikId() == 55)
                {
                    Row2.ForeColor = Color.Green;
                    TRow2.Report("OK: Метран-55 ");
                    break;
                }
             /*   else
                {
                    if (i == 2)
                    {
                        Row2.ForeColor = Color.Red;
                        TRow2.Report("Ошибка. Нет связи со спутником");
                        _tokenSource.Cancel();
                    }
                    else
                    {
                        Micro(0x62);
                        Thread.Sleep(1000);
                        continue;
                    }
                } */
                cancelToken.ThrowIfCancellationRequested();
            }
        }

        if (CheckADCCode)
        {

            /* FTempModeW write */

            FTempModeW[0] = 0x00;
            SputnikWrite(0x54, 0x0C, FTempModeW, 1, 1);
            cancelToken.ThrowIfCancellationRequested();

            /* Проверка кодов АЦП датчика */

            for (int i = 0; i < 2; i++)
            {
                float P_Code, T_Code, TpCode;
                bool P, T, Tp;
                P_Code = SputnikPressureCode();
                T_Code = SputnikBoardTermoCode();
                TpCode = SputnikSensorTermoCode();
                P = P_Code >= P_CodeMin && P_Code <= P_CodeMax;
                T = T_Code >= T_CodeMin && T_Code <= T_CodeMax;
                Tp = TpCode >= TpCodeMin && TpCode <= TpCodeMax;
```

```
                        if (P && T && Tp)
                        {
                            Row3.ForeColor = Color.Green;
                            TRow3.Report("OK");
                            TRow4.Report("P = " + P_Code + " T = " + T_Code + " Tp = " +
TpCode);
                            break;
                        }
                        else
                        {
                            if (i == 1)
                            {
                                Row3.ForeColor = Color.Red;
                                Row4.ForeColor = Color.Red;
                                TRow3.Report("Ошибка: неверные коды АЦП");
                                TRow4.Report(" P = " + P_Code + " T = " + T_Code + " Tp = "
+ TpCode + " Error");
                                _tokenSource.Cancel();
                            }
                            else
                            {
                                Thread.Sleep(500);
                                continue;
                            }
                        }
                        cancelToken.ThrowIfCancellationRequested();
                    }
                    /* read corr_dac, code_i0, corr_i0, corr_corr */
                }

                if (SetupCurrent)
                {
                    temp1 = SputnikRead(0x43, 0x0C, 8, 8);
                    corr_dac = OldByteToFloat(temp1[0], temp1[1], temp1[2], temp1[3]);
                    code_i0 = OldByteToFloat(temp1[4], temp1[5], temp1[6], temp1[7]);
                    temp1 = SputnikRead(0x20, 0x0E, 8, 8);
                    corr_i0 = OldByteToFloat(temp1[0], temp1[1], temp1[2], temp1[3]);
                    corr_corr = OldByteToFloat(temp1[4], temp1[5], temp1[6], temp1[7]);
                    cancelToken.ThrowIfCancellationRequested();

                    /* write  FM55CorrDac, c1, c2 */

                    c1[0] = 0x06;
                    c2[0] = 0x20;
                    float FCurrentSetup0 = 1.0f;
                    float FCurrent0 = 4.0f;
                    float FCurrent100 = 20.0f;
                    FM55CorrDAC = code_i0 + corr_i0 + corr_dac * corr_corr * (float)3408.0
                    * ((FCurrent0 + FCurrentSetup0) - FCurrent0) / (FCurrent100 -
FCurrent0);
                    SputnikWrite(0xD7, 0x0C, OldFloatToByte(FM55CorrDAC), 4, 4);
                    cancelToken.ThrowIfCancellationRequested();
                    SputnikWrite(0x4B, 0x0C, c1, 1, 1);
                    cancelToken.ThrowIfCancellationRequested();
                    SputnikWrite(0x54, 0x0C, c2, 1, 1);
                    cancelToken.ThrowIfCancellationRequested();

                    /* Считывание показаний с калибратора (4мА) */

                    UnlockCalibrator();
                    double Itemp4;
                    double Itemp20;
                    Itemp4 = Convert.ToDouble(MeasureCurrentCalibrator());
                    Itemp4 = Math.Round(Itemp4, 3);
                    cancelToken.ThrowIfCancellationRequested();
                    if (Itemp4 == 0)
                    {
                        Row6.ForeColor = Color.Red;
                        TRow6.Report("Ошибка: проверте калибратор или питание платы
датчика");
```

```
                    _tokenSource.Cancel();
                }

                cancelToken.ThrowIfCancellationRequested();

                /* write new_cod_i0 */

                float new_code_i0;
                new_code_i0 = code_i0 - ((float)Itemp4 - (FCurrent0 + FCurrentSetup0))
                * (float)3408.0 * corr_dac / (FCurrent100 - FCurrent0);
                code_i0 = new_code_i0;
                SputnikWrite(0x47, 0x0C, OldFloatToByte(new_code_i0), 4, 4);
                cancelToken.ThrowIfCancellationRequested();

                /* write FM55CorrDAC */

                FM55CorrDAC = code_i0 + corr_i0 + corr_dac * corr_corr * (float)3408.0
                * (FCurrent100 - FCurrent0) / (FCurrent100 - FCurrent0);
                SputnikWrite(0xD7, 0x0C, OldFloatToByte(FM55CorrDAC), 4, 4);
                cancelToken.ThrowIfCancellationRequested();

                /* Считывание показаний с калибратора (20 мА) */

                Itemp20 = Convert.ToDouble(MeasureCurrentCalibrator());
                Itemp20 = Math.Round(Itemp20, 3);
                cancelToken.ThrowIfCancellationRequested();
                if (Itemp20 == 0)
                {
                    Row6.ForeColor = Color.Red;
                    TRow6.Report("Ошибка: проверте калибратор или питание платы
датчика");
                    _tokenSource.Cancel();
                }
                else
                {
                    TRow6.Report("I(5 мА) = " + Itemp4 + " мА " + " I(20 мА) = " +
Itemp20 + " мА ");
                }
                cancelToken.ThrowIfCancellationRequested();

                /* write new_corr_dac */

                float new_corr_dac = (corr_dac * (FCurrent100 - FCurrent0)
                  / ((float)Itemp20 - FCurrent0));
                corr_dac = new_corr_dac;
                SputnikWrite(0x43, 0x0C, OldFloatToByte(new_corr_dac), 4, 4);
                new_code_i0 = code_i0;
                cancelToken.ThrowIfCancellationRequested();

                /* write to ROM */

                SputnikWrite(0x08, 0x00, OldFloatToByte(new_code_i0), 4, 4);
                cancelToken.ThrowIfCancellationRequested();
                SputnikWrite(0x04, 0x00, OldFloatToByte(new_corr_dac), 4, 4);
                cancelToken.ThrowIfCancellationRequested();

                /* restore */

                c1[0] = 0x00;
                SputnikWrite(0x4B, 0x0C, c1, 1, 1);
                cancelToken.ThrowIfCancellationRequested();

                /* проверка уровня тока на калибраторе */

                bool temp20 = false;
                for (int i = 0; i < 5; i++)
                {
                    float I4 = MeasureCurrentCalibrator();
                    if (I4 >= I4Min && I4 <= I4Max)
                    {
```

```
                        TRow8.Report(I4 + " мА" + "  OK");
                        break;
                    }
                    else if (I4 >= I20Min && I4 <= I20Max)
                    {
                        TRow7.Report(I4 + " мА" + "  OK");
                        temp20 = true;
                        break;
                    }
                    else
                    {
                        if (i == 4)
                        {
                            Row5.ForeColor = Color.Red;
                            TRow5.Report("Ошибка: уровни тока вне допустимых
пределов");
                            Row7.ForeColor = Color.Red;
                            TRow7.Report("I = " + I4);
                            _tokenSource.Cancel();

                        }
                        else
                        {
                            Thread.Sleep(1000);
                        }
                    }
                    cancelToken.ThrowIfCancellationRequested();
                }
                for (int i = 0; i < 5; i++)
                {
                    float I20 = MeasureCurrentCalibrator();
                    if (temp20)
                    {
                        if (I20 >= I4Min && I20 <= I4Max)
                        {

                            TRow8.Report(I20 + " мА" + "  OK");
                            Row5.ForeColor = Color.Green;
                            TRow5.Report("OK");
                            break;
                        }
                        else
                        {
                            if (i == 4)
                            {
                                Row5.ForeColor = Color.Red;
                                TRow5.Report("Ошибка: уровни тока вне допустимых
пределов");
                                Row7.ForeColor = Color.Red;
                                TRow7.Report("I = " + I20);
                                _tokenSource.Cancel();
                            }
                            else
                            {
                                Thread.Sleep(1000);
                            }
                        }
                    }
                    else
                    {
                        if (I20 >= I20Min && I20 <= I20Max)
                        {
                            TRow7.Report(I20 + " мА" + "  OK");
                            Row5.ForeColor = Color.Green;
                            TRow5.Report("OK");
                            break;
                        }
                        else
                        {
                            if (i == 4)
```

```
                                {
                                    Row5.ForeColor = Color.Red;
                                    TRow5.Report("Ошибка: уровни тока вне допустимых
пределов");
                                    Row7.ForeColor = Color.Red;
                                    TRow7.Report("I = " + I20);
                                    _tokenSource.Cancel();
                                }
                                else
                                {
                                    Thread.Sleep(1000);
                                }
                            }
                        }
                        cancelToken.ThrowIfCancellationRequested();
                    }
                }

                if (CheckFixedCurrent)
                {

                    /* read corr_dac, code_i0, corr_i0, corr_corr */

                    temp1 = SputnikRead(0x43, 0x0C, 8, 8);
                    cancelToken.ThrowIfCancellationRequested();
                    corr_dac = OldByteToFloat(temp1[0], temp1[1], temp1[2], temp1[3]);
                    code_i0 = OldByteToFloat(temp1[4], temp1[5], temp1[6], temp1[7]);
                    temp1 = SputnikRead(0x20, 0x0E, 8, 8);
                    cancelToken.ThrowIfCancellationRequested();
                    corr_i0 = OldByteToFloat(temp1[0], temp1[1], temp1[2], temp1[3]);
                    corr_corr = OldByteToFloat(temp1[4], temp1[5], temp1[6], temp1[7]);

                    /* write  FM55CorrDac, c1, c2 */

                    float FACFixedCurrentTestValue = 50.0f;
                    c1[0] = 0x06;
                    c2[0] = 0x20;
                    FM55CorrDAC = code_i0 + corr_i0 + corr_dac * corr_corr * (float)3408.0
                    * FACFixedCurrentTestValue / (float)100.0;
                    SputnikWrite(0xD7, 0x0C, OldFloatToByte(FM55CorrDAC), 4, 4);
                    cancelToken.ThrowIfCancellationRequested();
                    SputnikWrite(0x4B, 0x0C, c1, 1, 1);
                    cancelToken.ThrowIfCancellationRequested();
                    SputnikWrite(0x54, 0x0C, c2, 1, 1);
                    cancelToken.ThrowIfCancellationRequested();

                    /* Проверка 50% тока */

                    for (int i = 0; i < 3; i++)
                    {
                        float I12 = MeasureCurrentCalibrator();
                        if (I12 >= I50Min && I12 <= I50Max)
                        {
                            TRow10.Report(I12 + " мА" + " Ok");
                            Row9.ForeColor = Color.Green;
                            TRow9.Report("OK");
                            break;
                        }
                        else
                        {
                            if (i == 2)
                            {
                                Row9.ForeColor = Color.Red;
                                Row10.ForeColor = Color.Red;
                                TRow9.Report("Ошибка: ток вне допустимых пределов");
                                TRow10.Report("I = " + I12);
                                _tokenSource.Cancel();
                            }
                            else
                            {
```

```
                            Thread.Sleep(500);
                            continue;
                        }
                    }
                    cancelToken.ThrowIfCancellationRequested();
                }

                FTempModeW[0] = 0x20;
                SputnikWrite(0x54, 0x0C, FTempModeW, 1, 1);
                cancelToken.ThrowIfCancellationRequested();
                SputnikSensorOff();
                Thread.Sleep(100);
                SputnikSensorOn();
            }

            if (CheckSleepMode)
            {

                /* Режим сна */

                SputnikSleep();
                cancelToken.ThrowIfCancellationRequested();

                /* Проверка входного тока при сенсоре 120 %, -5 %, Err. */

                for (int i = 0; i < 3; i++)
                {
                    float I120 = MeasureCurrentCalibrator();
                    if (I120 >= I120Min && I120 <= I120Max)
                    {
                        TRow12.Report(I120 + " мА" + "  OK");
                        break;
                    }
                    else
                    {
                        if (i == 2)
                        {
                            Row11.ForeColor = Color.Red;
                            TRow11.Report("Ошибка: максимальный ток вне допустимых
пределов");

                            Row12.ForeColor = Color.Red;
                            TRow12.Report("I = " + I120);
                            _tokenSource.Cancel();
                        }
                        else
                        {
                            Thread.Sleep(500);
                            continue;
                        }
                    }
                    cancelToken.ThrowIfCancellationRequested();
                }

                Micro(0x61);
                Thread.Sleep(1000);

                for (int i = 0; i < 3; i++)
                {
                    float I120Err = MeasureCurrentCalibrator();
                    if (I120Err >= IerrMin && I120Err <= IerrMax)
                    {
                        TRow13.Report(I120Err + " мА" + "  OK");
                        break;
                    }
                    else
                    {
                        if (i == 2)
                        {
                            Row11.ForeColor = Color.Red;
```

```
                                TRow11.Report("Ошибка: ток ошибки вне допустимых
пределов");
                                Row13.ForeColor = Color.Red;
                                TRow13.Report("I = " + I120Err);
                                _tokenSource.Cancel();
                            }
                            else
                            {
                                Thread.Sleep(500);
                                continue;
                            }
                        }
                        cancelToken.ThrowIfCancellationRequested();
                    }

                    Micro(0x65);
                    Thread.Sleep(1000);

                    for (int i = 0; i < 3; i++)
                    {
                        float I5 = MeasureCurrentCalibrator();
                        if (I5 >= I5Min && I5 <= I5Max)
                        {
                            TRow14.Report(I5 + " мА" + "  OK");
                            break;
                        }
                        else
                        {
                            if (i == 2)
                            {
                                Row11.ForeColor = Color.Red;
                                TRow11.Report("Ошибка: минимальный ток вне допустимых
пределов");
                                Row14.ForeColor = Color.Red;
                                TRow14.Report("I = " + I5);
                                _tokenSource.Cancel();
                            }
                            else
                            {
                                Thread.Sleep(500);
                                continue;
                            }
                        }
                        cancelToken.ThrowIfCancellationRequested();
                    }

                    Micro(0x62);
                    Thread.Sleep(2000);

                    /* Проверка выходного тока при изменении напряжения питания */

                    Micro(0x6A);
                    Thread.Sleep(1000);
                    FTempModeW[0] = 0x20;
                    SputnikWrite(0x54, 0x0C, FTempModeW, 1, 1);
                    cancelToken.ThrowIfCancellationRequested();
                    SputnikSleep();
                    cancelToken.ThrowIfCancellationRequested();
                    float I15Mc = MeasureCurrentCalibrator();
                    float I15Mc2;
                    float inaccuracy = 0.5f;
                    bool test = true;

                    for (int i = 0; i < 2; i++)
                    {
                        Micro(0x6B);
                        I15Mc2 = MeasureCurrentCalibrator();
                        if (I15Mc2 >= I15Mc - inaccuracy && I15Mc2 <= I15Mc + inaccuracy)
                        {
                            Thread.Sleep(100);
```

```
                    }
                    else
                    {
                        test = false;
                    }
                    cancelToken.ThrowIfCancellationRequested();
                }
                if (test)
                {
                    TRow15.Report("OK");
                }
                else
                {
                    Row11.ForeColor = Color.Red;
                    TRow11.Report("Ошибка: ток падает при прерывании питиния");
                    _tokenSource.Cancel();
                }
                cancelToken.ThrowIfCancellationRequested();
                Micro(0x6A);
                Thread.Sleep(1000);

                /* проверка кнопок на плате */

                string message = "Нажать одновременно на обе кнопки и держать. Мигает
ли светодиод чаще одного раза в секунду?";
                string caption = "Проверка кнопок на плате";
                MessageBoxButtons buttons = MessageBoxButtons.YesNo;
                DialogResult result = MessageBox.Show(message, caption, buttons);

                if (result == System.Windows.Forms.DialogResult.Yes)
                {
                    TRow16.Report("OK");
                }
                else
                {
                    Row11.ForeColor = Color.Red;
                    TRow11.Report("Ошибка: кнопки неисправны");
                    _tokenSource.Cancel();
                }

                /* Сброс датчика, установка тех режима без ТК, завершение калибровки */

                Micro(0x6A);
                Thread.Sleep(1000);
                FTempModeW[0] = 0x20; // тех режим без ТК
                SputnikWrite(0x0015, 0x00, FTempModeW, 1, 1);
                byte[] FTempModeR = SputnikRead(0x15, 0x00, 1, 1);
                if (FTempModeW[0] == FTempModeR[0])
                {
                    Row11.ForeColor = Color.Green;
                    TRow11.Report("OK");
                    TRow17.Report("OK: Калибровка завершена");
                }
                else
                {
                    Row11.ForeColor = Color.Red;
                    TRow11.Report("Ошибка при записи режима");
                    _tokenSource.Cancel();
                    cancelToken.ThrowIfCancellationRequested();
                }
            }
        }
        catch (TimeoutException)
        {
            string message = "Нет связи с устройсвами";
            string caption = "Ошибка";
            MessageBoxButtons buttons = MessageBoxButtons.OK;
            DialogResult result = MessageBox.Show(message, caption, buttons);
            _tokenSource.Cancel();
            cancelToken.ThrowIfCancellationRequested();
```

```
            }
        }


        private async void button2_Click(object sender, EventArgs e)
        {

            Clear();

            ListViewItem Row1 = new ListViewItem();
            ListViewItem Row2 = new ListViewItem();
            ListViewItem Row3 = new ListViewItem();
            ListViewItem Row4 = new ListViewItem();
            ListViewItem Row5 = new ListViewItem();
            ListViewItem Row6 = new ListViewItem();
            ListViewItem Row7 = new ListViewItem();
            ListViewItem Row8 = new ListViewItem();
            ListViewItem Row9 = new ListViewItem();
            ListViewItem Row10 = new ListViewItem();
            ListViewItem Row11 = new ListViewItem();
            ListViewItem Row12 = new ListViewItem();
            ListViewItem Row13 = new ListViewItem();
            ListViewItem Row14 = new ListViewItem();
            ListViewItem Row15 = new ListViewItem();
            ListViewItem Row16 = new ListViewItem();
            ListViewItem Row17 = new ListViewItem();

            listView1.Items.Add(Row1);
            listView1.Items.Add(Row2);
            listView1.Items.Add(Row3);
            listView1.Items.Add(Row4);
            listView1.Items.Add(Row5);
            listView1.Items.Add(Row6);
            listView1.Items.Add(Row7);
            listView1.Items.Add(Row8);
            listView1.Items.Add(Row9);
            listView1.Items.Add(Row10);
            listView1.Items.Add(Row11);
            listView1.Items.Add(Row12);
            listView1.Items.Add(Row13);
            listView1.Items.Add(Row14);
            listView1.Items.Add(Row15);
            listView1.Items.Add(Row16);
            listView1.Items.Add(Row17);

            Progress<string> TRow2 = new Progress<string>(text => Row2.SubItems.Add(text));
            Progress<string> TRow3 = new Progress<string>(text => Row3.SubItems.Add(text));
            Progress<string> TRow4 = new Progress<string>(text => Row4.SubItems.Add(text));
            Progress<string> TRow5 = new Progress<string>(text => Row5.SubItems.Add(text));
            Progress<string> TRow6 = new Progress<string>(text => Row6.SubItems.Add(text));
            Progress<string> TRow7 = new Progress<string>(text => Row7.SubItems.Add(text));
            Progress<string> TRow8 = new Progress<string>(text => Row8.SubItems.Add(text));
            Progress<string> TRow9 = new Progress<string>(text => Row9.SubItems.Add(text));
            Progress<string> TRow10 = new Progress<string>(text =>
Row10.SubItems.Add(text));
            Progress<string> TRow11 = new Progress<string>(text =>
Row11.SubItems.Add(text));
            Progress<string> TRow12 = new Progress<string>(text =>
Row12.SubItems.Add(text));
            Progress<string> TRow13 = new Progress<string>(text =>
Row13.SubItems.Add(text));
            Progress<string> TRow14 = new Progress<string>(text =>
Row14.SubItems.Add(text));
            Progress<string> TRow15 = new Progress<string>(text =>
Row15.SubItems.Add(text));
            Progress<string> TRow16 = new Progress<string>(text =>
Row16.SubItems.Add(text));
            Progress<string> TRow17 = new Progress<string>(text =>
Row17.SubItems.Add(text));
```

```
        _tokenSource = new System.Threading.CancellationTokenSource();
        System.Threading.CancellationToken cancelToken = _tokenSource.Token;

        /* Открытие портов */

        if (comboBox2.Text == "" || comboBox1.Text == "" || comboBox3.Text == "")
        {
            string message = "Не выбраны порты";
            string caption = "Ошибка";
            MessageBoxButtons buttons = MessageBoxButtons.OK;
            DialogResult result = MessageBox.Show(message, caption, buttons);
        }
        else
        {
            Port.Enabled = false;
            Start.Enabled = false;
            Stop.Enabled = true;

            PortSputnikOpen(comboBox2.Text);
            PortCalibratorOpen(comboBox1.Text);
            PortMicroOpen(comboBox3.Text);

            try
            {
                await Task.Run(() => StartCalibration(cancelToken,
                    TRow2, TRow3, TRow4, TRow5, TRow6, TRow7, TRow8, TRow9,
                    TRow10, TRow11, TRow12, TRow13, TRow14, TRow15, TRow16, TRow17,
                    checkBox1.Checked, checkBox2.Checked, checkBox3.Checked,
                    checkBox4.Checked, checkBox5.Checked), cancelToken);
            }

            catch (System.OperationCanceledException)
            {
                PortCalibratorClose();
                PortMicroClose();
                PortSputnikClose();
                _tokenSource.Dispose();
                //случай отмены
            }
            catch (Exception)
            {
                //случай если возникнет какая-то ошибка
            }
            finally
            {
                Port.Enabled = true;
                Start.Enabled = true;
                Stop.Enabled = false;
                Start.Focus();
                PortCalibratorClose();
                PortMicroClose();
                PortSputnikClose();
                _tokenSource.Dispose(); // удаляем источник токена отмены
            }
        }

    }

    /* Нажатие на кнопку отмены */

    private void button1_Click_1(object sender, EventArgs e)
    {
        _tokenSource.Cancel();
    }
```

Листинг А.2

```
#include "main.h"
#include "m55_auto.h"
#define DEBUGOFF

static __IO ErrorStatus HSEStartUpStatus = SUCCESS;
static __IO  uint32_t TimingDelay = 0;
volatile unsigned char tmmode,sstatus;
volatile unsigned int fl;        //flags
volatile unsigned char pwrmode,resmode,loadmode,voltmode;

// *****    MAIN      ******
int main(void)
{ //init state
  unsigned char val;  //read button
       fl=0;
  //
  initHardware();
  usartMsg("clear ports",1);
  GPIO_Write(LEDS_PORT,0x00); //A
  GPIO_Write(BTNS_PORT,0x00);   //B
  GPIO_Write(OPTO_PORT,0x00);     //C
  usartMsg("STM32F103RBT6 m55_auto",1);
  usartMsg("M55 test v021",1);
  Delay_ms(500);
  usartMsg("after delay 500ms",1);
  val=5;
       while(val--){
    GPIO_SetBits(LEDS_PORT,LEDPIN);
    Delay_ms(500);
    GPIO_ResetBits(LEDS_PORT,LEDPIN);
    Delay_ms(500);
  }
  usartMsg("led blink finish",1);
       fl|=RXF;     //set first time flag for RXF

  //start
  usartMsg("Set TC mode",1);
  setTC();    //set pult to init state
  pwrmode=2;
  resmode=0;
  loadmode=0;
  voltmode=0; //not! we switch load and automatically switched voltmode
  usartMsg("Listen..",1);
       while (1){
    val=GPIO_ReadInputDataBit(BTNS_PORT,BTN0);
    if(val==0){
      usartMsg("Button 0",1);
    }
       if(fl&COMRX){   //we received something
      fl&=~COMRX;   //clear com data from user flag
      tmp=(rxdata[0]-0x30)*100+(rxdata[1]-0x30)*10+(rxdata[2]-0x30);  //3 digit
      uartcmd[0]=rxdata[0]; //first enetered byte
      uartcmd[1]=rxdata[1];  //second
      uartcmd[2]=rxdata[2];


      switch(uartcmd[1]){      //second char is command
      case 0x61:    //a switch to err
        usartMsg("setting resistance load mode to err",1);  //
        set_Rmode(ERR);
        showValues();
        usartMsg("set to err",1);  //
        break;
      case 0x62:    //b 120%
        usartMsg("setting resistance load mode to 120%",1);  //
        set_Rmode(R120);
        showValues();
```

```
        usartMsg("Set 120%",1);  //
        break;
      case 0x63:      //c 50%
        usartMsg("setting resistance load mode to 50%",1);  //
        set_Rmode(R50);
        showValues();
        usartMsg("Set 50%",1);  //
        break;
      case 0x64:      //d 0%
        usartMsg("setting resistance load mode to 0%",1);  //
        set_Rmode(R0);
        showValues();
        usartMsg("Set 0%",1);  //
        break;
      case 0x65:      //e -5%
        usartMsg("setting resistance load mode to -5%",1);  //
        set_Rmode(RM5);
        showValues();
        usartMsg("Set -5%",1);  //
        break;
      case 0x66:      //f load 50/200
        usartMsg("setting load mode to 50/200",1);  //
        set_Lmode(L50);
        showValues();
        usartMsg("Set 50/200",1);  //
        break;
      case 0x67:      //g load 1050/2500
        usartMsg("setting load mode to 1050/2500",1);  //
        set_Lmode(L1050);
        showValues();
        usartMsg("Set 1050/2500",1);  //
        break;
      case 0x68:      //h volt 24V
        usartMsg("setting volt mode to 24V",1);  //
        set_Vmode(V24);
        showValues();
        usartMsg("Set 24V",1);  //
        break;
      case 0x69:      //i volt 42V
        usartMsg("setting volt mode to 42V",1);  //
        set_Vmode(V42);
        showValues();
        usartMsg("Set 42V",1);  //
        break;
      case 0x6a:    //j Power off
        click(POFF);
        showValues();
        usartMsg("click PWR off btn",1);  //
        break;
      case 0x6b:    //k
        click(MS15);
        showValues();
        usartMsg("click 15 ms btn",1);  //
        break;
      }
      unsigned char i,*ptr;    //clear array
      for(i=0,ptr=uartcmd;i<3;i++){
        *ptr=0;
        ptr++;
      }
    } //end of if,we received something
     }
}
/*****************************************************************************
* Function Name  : initHardware
* Description    : Configures the hardware including clock,ports,Timers.
* Input          : None
* Output         : None
* Return         : None
*****************************************************************************/
```

```
void initHardware(void){
  Clock_Config();
  GPIO_Config();
  USART_Config();
  SysTick_Configuration();
}
/*******************************************************************************
* Function Name  : Clock_Config
* Description    : Configures the system clock.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void Clock_Config(void){
       ErrorStatus HSEStartUpStatus;
  //SYSCLK, HCLK, PCLK2 and PCLK1 configuration ----------------------------
  //RCC system reset(for debug purpose)
  RCC_DeInit();

  //Enable HSE
  RCC_HSEConfig(RCC_HSE_ON);

  //Wait till HSE is ready
  HSEStartUpStatus = RCC_WaitForHSEStartUp();

  if(HSEStartUpStatus == SUCCESS)
  {
    // Enable Prefetch Buffer
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);

    //Flash 2 wait state
    FLASH_SetLatency(FLASH_Latency_2);

    //HCLK = SYSCLK
    RCC_HCLKConfig(RCC_SYSCLK_Div1);

    //PCLK2 = HCLK APB2 clock
    RCC_PCLK2Config(RCC_HCLK_Div1);

    //PCLK1 = HCLK/2  APB1
    RCC_PCLK1Config(RCC_HCLK_Div2);

    //my m55_auto PLLCLK = 12MHz/2 * 12 = 72 MHz
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_6);

    //Enable PLL
    RCC_PLLCmd(ENABLE);

    //Wait till PLL is ready
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
    {
    }
    //Select PLL as system clock source
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

    //Wait till PLL is used as system clock source
    while(RCC_GetSYSCLKSource() != 0x08)
    {
    }
  }
}
/*******************************************************************************
* Function Name  : GPIO_Config
* Description    : Configures the different GPIO ports pins.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void GPIO_Config(void)
{
```

```
       EXTI_InitTypeDef   EXTI_InitStructure;
       GPIO_InitTypeDef   GPIO_InitStructure;
       NVIC_InitTypeDef   NVIC_InitStructure;
    /* Enable GPIOA, GPIOB, GPIOC, GPIOD, GPIOE and AFIO clocks */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |RCC_APB2Periph_GPIOC
         | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO, ENABLE);

  GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable,ENABLE);// JTAG-DP Disabled and SW-DP
Enabled

       /* Configure LEDs as output push-pull */
  GPIO_InitStructure.GPIO_Pin = LEDPIN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(LEDS_PORT, &GPIO_InitStructure);

  /* Configure Buttons as input floating */
  GPIO_InitStructure.GPIO_Pin = BTN0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(BTNS_PORT, &GPIO_InitStructure);

  /*Configure OPTO control pin*/
  GPIO_InitStructure.GPIO_Pin =
OPTOPIN1|OPTOPIN2|OPTOPIN3|OPTOPIN4|OPTOPIN5|OPTOPIN6|RELPIN1|RELPIN2;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(OPTO_PORT, &GPIO_InitStructure);

       /*Configure pult feedback signal*/
  GPIO_InitStructure.GPIO_Pin = FEED1|FEED2;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(FEED_PORT, &GPIO_InitStructure);

  /*doesnt work
  //GPIO interrupt sources init
       //BTN0 Button PB4
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource0);
       //Enable the EXTI Line0 for PB0 Interrupt
       EXTI_InitStructure.EXTI_Line = EXTI_Line4;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_Init(&EXTI_InitStructure);

  //Enable and set Button EXTI4 Interrupt to the lowest priority
  NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0f;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0f;
  NVIC_Init(&NVIC_InitStructure);
  */

  /* Enable the USART2 Pins Software Remapping
  GPIO_PinRemapConfig(GPIO_Remap_USART2, ENABLE);
  RCC_APB1PeriphClockCmd(COM_USART_CLK[COM], ENABLE);*/

       //Configure USART1 Tx as alternate function push-pull
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Pin = U_TX_PIN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
  //Configure USART Rx as input floating
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_InitStructure.GPIO_Pin = U_RX_PIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}
/*******************************************************************************
* Function Name  : SysTick_Configuration
* Description    : Configure a SysTick Base time to 10 ms.
* Input          : None
```

```
* Output         : None
* Return         : None
*******************************************************************************/
void SysTick_Configuration(void)
{
  /* Setup SysTick Timer for 10 msec interrupts  */
  if (SysTick_Config(SystemCoreClock / 1000))        //SystemCoreClock is for new periph lib
  {
    /* Capture error */
    while (1){
                    //GPIO_SetBits(LEDS_PORT,LED1);
            }
  }

 /* Configure the SysTick handler priority */
  NVIC_SetPriority(SysTick_IRQn, 0x0);
}
#ifdef DEBUGOFF
//Ôóíêöèÿ âðåìåíîé çàäåðæêè, works,but
void Delay_ms(__IO uint32_t nTime)
{
  TimingDelay = nTime;
  while(TimingDelay != 0);
}
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }
}
//RS-232 connection
void USART_Config(){
       USART_InitTypeDef USART_InitStructure;
  /* Enable GPIO clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
       //USART init struct
  //USART_InitStructure.USART_BaudRate = 19200;
  USART_InitStructure.USART_BaudRate = 38400;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  /* USART configuration */
  USART_Init(USART1, &USART_InitStructure);
  /* Enable the USARTx Interrupt */
       NVIC_InitTypeDef NVIC_InitStructure;
  NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
       /* Enable the EVAL_COM1 Receive interrupt: this interrupt is generated when the
     EVAL_COM1 receive data register is not empty */
  USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
  /* Enable USART */
  USART_Cmd(USART1, ENABLE);
}
//message send:if enter=1 adds CR LF,else no CRLF
void usartMsg(unsigned char *msg,unsigned char enter){
  unsigned char * tmp=msg;
  while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
  while(*tmp){
    USART_SendData(USART1,(unsigned char)*tmp);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
    tmp++;
  }
       if(enter){
         USART_SendData(USART1,0xd);    //CR
```

```
        while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
        USART_SendData(USART1,0xa);    //LF
        while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
      }
}
void usartTx(unsigned char *sendbyte){
  while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
  USART_SendData(USART1,(unsigned char) *sendbyte);
      while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
}
//usart byte to symbols
//convert byte to digit sequence,CR LF included
void usartShowByte(unsigned char byte){
      unsigned char digit[3],cnt;
      digit[0]=(byte/100)+0x30;              //msb
      digit[1]=(byte/10%10)+0x30;
      digit[2]=(byte%10)+0x30;               //lsb
      unsigned char *digitptr;
      digitptr=&digit[0]; //takes addressof first element array
      for(cnt=0;cnt<3;cnt++){
            usartTx(digitptr++);      //increase address
      }
  USART_SendData(USART1,0x20);
  while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
}
//convert int to digit sequence,CR LF included
void usartShowInt(unsigned int tbyte){
      unsigned char digit[5],cnt;
      digit[0]=(tbyte/10000)+0x30;                  //msb
      digit[1]=(tbyte/1000%10)+0x30;
      digit[2]=(tbyte/100%10)+0x30;
      digit[3]=(tbyte/10%10)+0x30;
      digit[4]=(tbyte%10)+0x30;              //lsb
      unsigned char *digitptr;
      digitptr=&digit[0]; //takes addressof first element array
      for(cnt=0;cnt<5;cnt++){
            usartTx(digitptr++);      //increase address
      }
  USART_SendData(USART1,0x20);
  while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
}
//us delay func for 16Mhz quarz
void Delay_10us(unsigned char us){
  uint8_t tmp;
  while(us--){
    for(tmp=0;tmp<80;tmp++){
      asm("nop");
    }
  }
}
#else
//RS-232 connection
void USART_Config(){
  asm("nop");}
void usartMsg(unsigned char *msg,unsigned char enter){
  asm("nop");}
void usartTx(unsigned char *sendbyte){
  asm("nop");
}
void usartShowByte(unsigned char byte){
  asm("nop");}
void usartShowInt(unsigned int tbyte){
  asm("nop");}
void Delay_10us(unsigned char us){
  asm("nop");
}
void Delay_ms(__IO uint32_t nTime){
  asm("nop");
}
void TimingDelay_Decrement(void){
```

```
  asm("nop");
}

#endif
//default mode
void RX_mode(void){ //receiver mode
       if(fl&RXF){                //first time rx_mode run
              fl&=~RXF;    //clear flag
              //usartMsg("RXF");
       }
       //if we have data from uart port
       if(fl&COMRX){
              fl&=~COMRX;  //clear com data from user flag
              fl|=RXF;     //clear flag to reinitialize rx mode
              //usartMsg("NRF TX send");
       }
}
//click button on pult M55
void click(unsigned char SAkey){
  switch(SAkey){
  case SA4KEY:
    GPIO_SetBits(OPTO_PORT,SA4);
    Delay_ms(300);
    GPIO_ResetBits(OPTO_PORT,SA4);
    Delay_ms(300);
    break;
  case SA3KEY:
    GPIO_SetBits(OPTO_PORT,SA3);
    Delay_ms(300);
    GPIO_ResetBits(OPTO_PORT,SA3);
    Delay_ms(300);
    break;
  case SA2KEY:
    GPIO_SetBits(OPTO_PORT,SA2);
    Delay_ms(300);
    GPIO_ResetBits(OPTO_PORT,SA2);
    Delay_ms(300);
    break;
  case SA1KEY:
    GPIO_SetBits(OPTO_PORT,SA1);
    Delay_ms(300);
    GPIO_ResetBits(OPTO_PORT,SA1);
    Delay_ms(300);
    break;
  case MS15:
    GPIO_SetBits(OPTO_PORT,SA16);
    Delay_ms(300);
    GPIO_ResetBits(OPTO_PORT,SA16);
    Delay_ms(300);
    break;
  case POFF:
    GPIO_SetBits(OPTO_PORT,SA15);
    Delay_ms(500);
    GPIO_ResetBits(OPTO_PORT,SA15);
    Delay_ms(300);
    break;
  }
}
//click to M55 pult mode
void setTC(void){
  unsigned char readval;  //read LEDIN
  readval=GPIO_ReadInputDataBit(FEED_PORT,FEED1); //if readval=1 ->led on
  if(readval==1){   //if we in TC mode, click to get out
    click(SA4KEY);
  }
  readval=GPIO_ReadInputDataBit(FEED_PORT,FEED1); //
  while(readval==0){
    click(SA4KEY);
    readval=GPIO_ReadInputDataBit(FEED_PORT,FEED1);
  }
```

```
  //click(SA2KEY);
  //click(SA2KEY);  //to set voltmode to 24V
  showValues();
  usartMsg("Set to TC mode",1);
  return;
}
//set modes
void set_Rmode(unsigned char setmode){
  while(resmode!=setmode){
    click(SA3KEY);
    resmode++;
    if(resmode>4)
      resmode=0;
    usartMsg("choosing Rmode",1);
    //any change of resmode occur switching to loadmode=0,voltmode=2
    loadmode=0;
    voltmode=0;
  }
  usartMsg("Rmode set",1);
  return;
}
void set_Lmode(unsigned char setmode){
  while(loadmode!=setmode){
    click(SA2KEY);
    loadmode++;
    if(loadmode>1)
      loadmode=0;
    usartMsg("choosing Lmode",1);
    resmode=0;    //any switch of loadmode switches resmode to 120%
  }
  if((voltmode==1)&&(loadmode==1)){
    voltmode=2;   //switch to 1050/2500 forces to switch 42V if not 42V set
  }
  usartMsg("Lmode set",1);
  return;
}
void set_Vmode(unsigned char setmode){
  while(voltmode!=setmode){
    click(SA1KEY);
    voltmode++;
    if(voltmode>2)
      voltmode=1;
    usartMsg("choosing Vmode",1);
    resmode=0;    //any switch of voltmode switches resmode to 120%
  }
  if((loadmode==1)&&(voltmode==1)){
    loadmode=0;
  }
  usartMsg("Vmode set",1);
  return;
}
//debug func
void showValues(void){
  usartMsg(" resmode, loadmode, voltmode:",0);
  usartShowByte(resmode);
  usartShowByte(loadmode);
  usartShowByte(voltmode);
  usartMsg("-",1);
}

void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  /* Infinite loop */
  while (1)
  {
  }
}
```

ПРИЛОЖЕНИЕ Б. Схемы и чертежи