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

Разработка устройства определения положения объекта с передачей параметров на ПК

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
ЮУрГУ – 12.03.01.2021.308-407 КР

Руководитель, доцент
_____/ С.В. Колодий
_____2021 г.

Автор работы:
студент группы КЭ-413
_____/ А.А. Спиридон
_____2021 г.

Нормоконтролер, доцент
_____/ А.С. Волосников
_____2021 г.

Челябинск 2021

АННОТАЦИЯ

Спиридон А.А. Разработка устройства определения положения объекта с передачей параметров на ПК. – Челябинск: ЮУрГУ, КЭ-413, 2021, 52 с., 34 ил., библиогр. список – 20 наим.

В ходе выполнения данной работы были выполнены следующие пункты:

1) Произведены измерения температуры с помощью встроенного температурного датчика микроконтроллера STM32F411RE с периодичностью 50 мс;

2) Произведены измерения ускорения с помощью акселерометра ADXL345 с периодичностью 50 мс;

3) Осуществлена передача данных в виде пакетов с микроконтроллера на компьютер с помощью UART с периодичностью 100 мс;

4) Для «параллельного» измерения температуры и передачи данных использовалась обёртка над ОСРВ FreeRTOS [1], написанная на языке программирования C++;

5) Написана программа на ПК с использованием графической подсистемы WPF платформы .NET: эта программа создаёт визуальное изображение платы микроконтроллера, которая вращается при вращении реальной платы, на которой расположен акселерометр.

Верхний уровень проекта написан на языке программирования C# [2], нижний уровень – на языке C++ [3].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Процессор – это цифровая схема, которая выполняет операции с некоторым внешним источником данных, обычно с памятью или каким-либо другим потоком данных. Обычно он представляет собой микропроцессор, который может быть реализован на одном кристалле интегральной схемы металл-оксид-полупроводник [4].

Микропроцессор представляет собой многоцелевую цифровую интегральную схему на основе регистров, управляемую часами, которая принимает двоичные данные в качестве входных данных, обрабатывает их в соответствии с инструкциями, хранящимися в своей памяти, и предоставляет результаты (также в двоичной форме) в качестве выходных. Микропроцессоры содержат как комбинационную логику, так и последовательную цифровую логику. Микропроцессоры работают с числами и символами, представленными в двоичной системе счисления [5].

Использование микропроцессоров в приборах стало общепринятым ввиду того, что микропроцессор может обрабатывать сигналы от нескольких датчиков сразу, сравнивать измеренные значения с номинальным значением и отображать результаты на экране визуального дисплея. Таким образом, микропроцессорные системы позволяют быстро и эффективно решать задачи управления данными и обработки результатов [6].

Проектируемая система должна состоять из 5 составных частей:

1) измерение температуры,

2) измерение ускорения по трём осям,

3) создание пакетов данных и их отправка по UART,

4) получение данных с последовательного порта и далнейшая распаковка данных,

5) отображение полученной информации (как в виде трёхмерного графического изображения объекта, так и в виде меняющихся численных значений измеренной температуры и ускорения).

Измерения с использованием температурного датчика, встроенного в микроконтроллер, имеют главную особенность: такие измерения показывают не температуру окружающей среды, а температуру самого микроконтроллера. Это накладывает определённые ограничения на использование такого датчика для измерения температуры и определяет его область применения.

Например, использовать встроенный датчик температуры можно для индикации перегрева устройства. Соответственно, передавая измеренную температуры на компьютер и визуально отображая её, можно значительно облегчить процесс идентификации перегрева устройства.

С помощью акселерометра можно определять не только относительные ускорения объекта, но и его ориентацию в трёхмерном пространстве. Это может быть применено, например, в СУ транспортными средствами.

Измерительная часть данной системы была спроектирована таким образом, чтобы при необходимости можно было достаточно легко заменить способ измерения температуры или ускорения. К примеру, если будет принято решение измерять температуру не с помощью встроенного температурного датчика, а с помощью температурного датчика, находящегося на другой плате, тогда придётся переписывать не всю систему, а лишь один класс, отвечающий за работу непосредственно с АЦП.

Аналогичным образом, если будет принято решение заменить способ передачи данных (допустим, USART будет замен на USB) или, например, будет принято решение параллельно использовать два разных акселерометра для измерения относительных ускорений объекта, то не придётся переписывать всю систему целиком.

То есть проектируемая система может быть довольно гибкой и расширяемой.

1 АНАЛИЗ ТРЕБОВАНИЙ ТЕХНИЧЕСКОГО ЗАДАНИЯ

1.1. Для разработки должна использоваться отладочная плата XNUCLEO-F411RE

XNUCLEO-F411RE – отладочная плата компании Waveshare. В основе платы ARM Cortex-M4 микроконтроллер STM32F411RET6. Эта отладочная плата представляет собой гибкую платформу, позволяющую разработчикам реализовать собственные идеи и в кратчайшие сроки сделать прототип будущего изделия.

Разъемы ST Morpho платы XNUCLEO-F411RE обеспечивают полный доступ к линиям портов ввода/вывода (I/O) и дальнейшее периферийное расширение.

Поддержка mbed делает возможным быстрое построение прототипа устройства с использованием SDK и online инструментов. Комплексное бесплатное программное обеспечение (HAL библиотека) включает различные примеры софта. Изделие поставляется с отдельным модулем ST-Link/ V2.

Технические характеристики микроконтроллера STM32F411RET6:

- ядро: ARM 32-Бит Cortex-M4;
- рабочая частота: 100МГц;
- рабочее напряжение: 1.7…3.6В;
- память: 512кБ Flash, 128кБ SRAM;
- интерфейсы: 1 x SDIO, 1 x USB 2.0 FS, 5 x SPI or 5 x I2S, 3 x USART, 3 x I2C;
- АЦП/ЦАП: 1 x АЦП (12 Бит, 16 каналов).

Остальные технические характеристики:

- SPX3819M5: регулятор напряжения 3,3 В;
- AMS1117-5.0: регулятор напряжения 5,0 В;
- CP2102: преобразователь USB в UART;
- разъем Arduino: для подключения щитов Arduino;
- интерфейс ICSP: Arduino ICSP;

- USB TO UART: для отладки;
- разъем USB: интерфейс связи USB;
- интерфейс SWD: для программирования и отладки;
- заголовки ST Morpho: доступ к VCC, GND и всем входам / выходам, прост в расширении;
- 6-12 В постоянного тока;
- пользовательская кнопка;
- кнопка сброса;
- индикатор питания;
- пользовательский светодиод;
- 500 мА быстрый самовосстанавливающийся предохранитель;
- индикатор Rx / Tx последовательного порта;
- кристалл 8 МГц;
- кристалл 32,768 кГц.

Комплектация:

- 1 x Отладочная плата (XNUCLEO-F411RE);
- 1 x Программатор (ST-LINK/V2 (mini));
- 1 x Кабель (USB Type A Plug to Micro B Plug Cable);
- 1 x Кабель (USB Type A Plug to Receptacle Cable).

Технические параметры представлены в таблице 1.

Таблица 1 – Технические параметры

Серия оценочной/отладочной платы	XNUCLEO
Серия оценочной/отладочной платы	Cortex-M4
Разрядность шины данных, Бит	32
Наименование базового компонента	STM32F411RET6
Тип разъема для прямого подключения плат расширения	Arduino

Наличие USB интерфейса	Да
Наличие установленного (в комплекте) дисплея	Нет
Наличие макетной области	Нет
Особенности	ST-LINK/V2-mini, Arduino-интерфейс
Вес, г	154

1.2. Программное обеспечение должно измерять положение платы в пространстве X, Y, Z

Под ориентацией (положением) объекта в пространстве понимаются описательные характеристики расположения объекта в пространстве (т.е. его угловое положение или направление). Более конкретно, ориентация объекта относится к воображаемому вращению, которое необходимо для перемещения объекта из исходного положения в его текущее положение [7].

Иначе говоря, программное обеспечение должно измерять углы поворота объекта в трёхмерном пространстве XYZ.

1.3. Устройство должно измерять угол наклона и ускорение измеряемого объекта

Ускорение – это физическая величина, которая равна отношению изменения скорости объекта v к величине времени t, за который произошло это изменение. Является векторной величиной, имеющей направление и амплитуду (или размер/величину), и может выражаться как первая производная скорости v по времени или как вторая производная перемещения объекта x по времени [8].

$$a = \frac{dv}{dt} = \frac{d^2x}{dt^2} \quad (1)$$

Пространственная ориентация направления ускорения объекта определяется направлением равнодействующей силы (т.е. векторной суммой сил, действующих на объект) [8, 9].

Амплитуда (величина) ускорения объекта по второму закону Ньютона является комбинацией двух составляющих: 1) величина ускорения прямо пропорциональна равнодействующей сил, действующих на объект, и 2) величина ускорения обратно пропорциональна массе объекта [8, 10].

Угол наклона – это количественная мера поворота объекта относительно фиксированной точки (обычно центра объекта) [11].

Рисунок иллюстрирует наклон объекта относительно осей OX, OY и OZ.

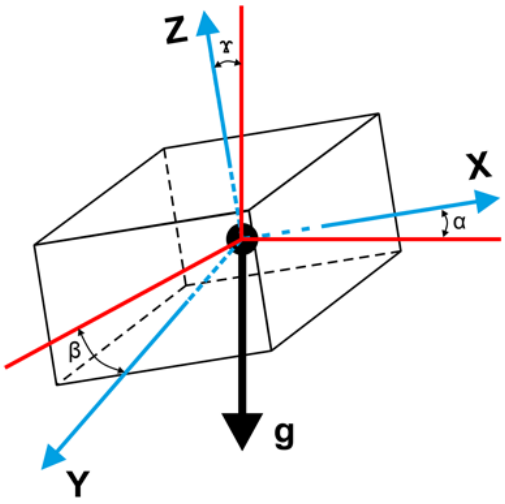


Рисунок 1 – Наклон объекта относительно осей OX, OY и OZ

Измерение наклона объекта можно производить с помощью измерения ускорений по трём осям. Известно, что угол поворота объекта может быть найден по формулам:

$$\alpha = \arctan\left(\frac{A_x}{\sqrt{A_y^2 + A_z^2}}\right) \quad (2)$$

$$\beta = \arctan\left(\frac{A_y}{\sqrt{A_x^2 + A_z^2}}\right) \quad (3)$$

$$\gamma = \arctan\left(\frac{A_z}{\sqrt{A_y^2 + A_z^2}}\right), \quad (4)$$

где α, β и γ – углы отклонения объекта от осей OX, OY и OZ соответственно;

A_x, A_y и A_z – величина ускорения объекта вдоль осей OX, OY и OZ соответственно.

1.4. Период измерения должен быть 50 ms

Необходимо обеспечить с периодичностью раз в 50ms опрос канала АЦП, к которому подключен температурный датчик, и обращение к регистру I2C_DR для получения значений ускорения, измеренных с помощью акселерометра ADXL345.

Это можно реализовать с помощью операционной системы реального времени (ОСРВ).

1.5. К измеренным значениям параметров должен быть применен цифровой фильтр

К измеренным значениям параметров должен быть применен цифровой фильтр вида:

$$T = \int \begin{pmatrix} 1 - e^{-\frac{dt}{RC}} & RC > 0\ sec, \\ 1 & RC \leq 0\ sec \end{pmatrix} \quad (5)$$

$$FilteredValue = OldFiltered + (Value - OldFiltered), \quad (6)$$

где dt – промежуток времени (в данной работе равный 100 мс);

$Value$ – текущее нефильтрованное измеренное значение температуры;

$OldFiltered$ – предыдущее фильтрованное значение.

1.6. Для измерения параметров должен использоваться датчик ADXL345

Измерение ускорения производится каждые 50 мс с помощью трёхосевого акселерометра ADXL345 и I2C.

Акселерометр ADXL345 предназначен для измерения статического ускорения свободного падения, а также динамического ускорения в результате движения или механических вибраций. Измерение производится по трём осям с разрешением 13 бит. Выходные данные устройства могут быть доступны с помощью интерфейсов SPI или I2C.

Схема акселерометра ADXL345 представлена на рисунке 2.

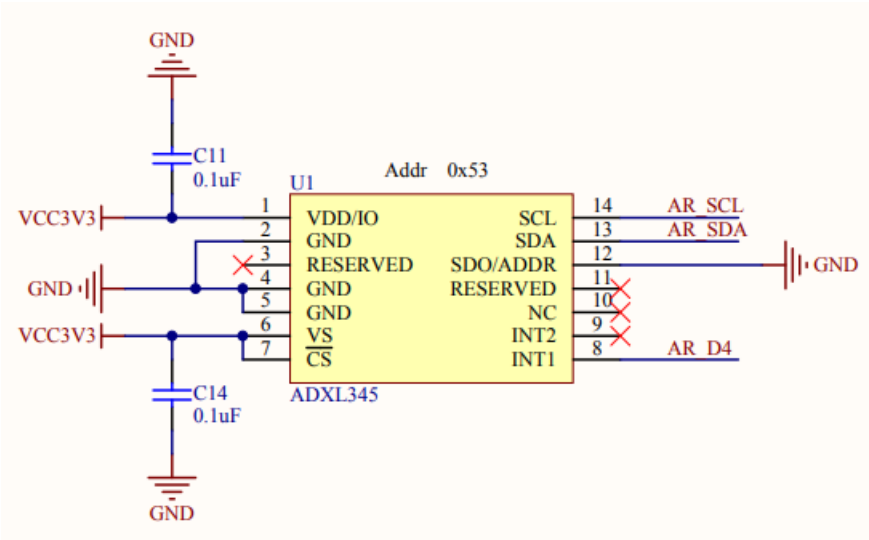


Рисунок 2 – Схема акселерометра ADXL345

На рисунке 3 представлена функциональная блок-схема акселерометра ADXL345.

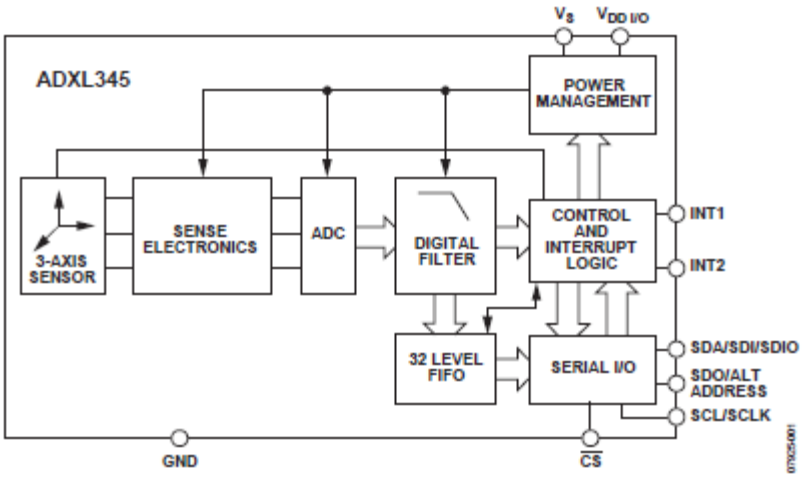


Рисунок 3 – Функциональная блок-схема акселерометра ADXL345

С более подробным описанием акселерометра ADXL345 можно ознакомиться в документации на устройство.

1.7. Получение данных с датчика ADXL345 должно производиться через I2C1 (PB8, PB9) интерфейс

I2C (Inter-Integrated Circuit) – это синхронная последовательная шина для передачи данных, реализующая пакетную коммутацию и модель «multi-master, multi-slave», а также широко использующаяся для присоединения низкоскоростных периферийных интегральных микросхем к процессору или микроконтроллеру на коротких дистанциях [12].

SCL (Serial Clock Line) – это линия, по которой последовательно передается сигнал тактирования.

SDA (Serial Data Line) – это линия, по которой последовательно передаются данные.

Типичная схема I2C представлена на рисунке 4. Согласно этой схеме, микроконтроллер становится Master, а акселерометр и гироскоп – Slave.

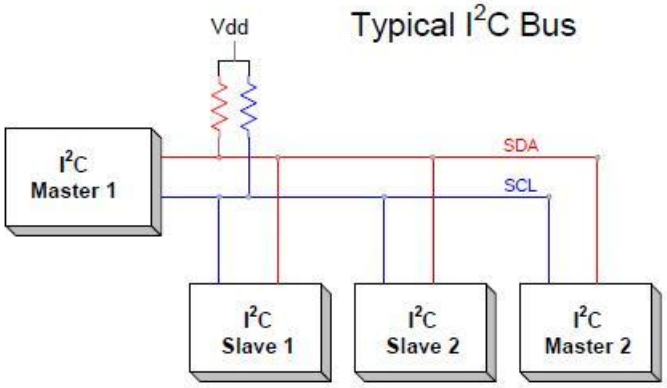


Рисунок 4 – Типичная схема I2C

Причём, в I2C мастером называется устройство, которое генерирует тактирование для других устройств и инициализирует общение с Slave-устройствами, тогда как Slave-устройство принимает сигнал тактирования и отвечает при обращении от мастера [12].

Шина является шиной с несколькими ведущими, что означает, что может присутствовать любое количество Master- устройств. Кроме того, роли ведущего и ведомого могут меняться между сообщениями (после отправки STOP) [12].

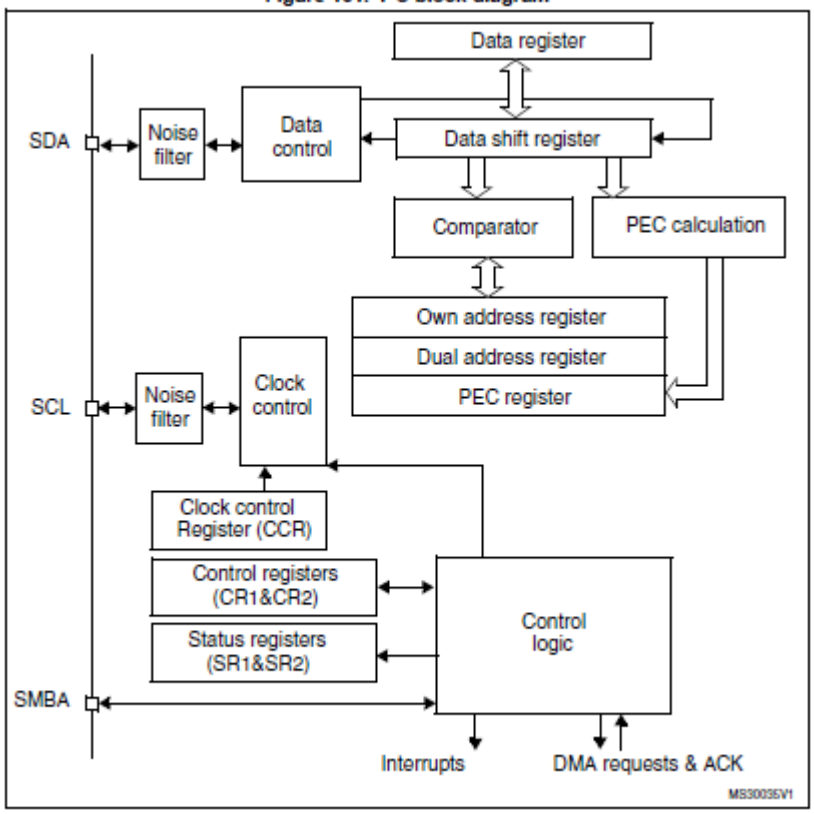
Для данного устройства шины может быть четыре возможных режима работы, хотя большинство устройств используют только одну роль и два ее режима:

- master transmit – ведущий узел отправляет данные Slave-устройству;
- master receive – мастер получает данные от Slave-устройства;
- slave transmit – Slave-устройство отправляет данные мастеру;
- slave receive – Slave-устройство получает данные от мастера [12].

Acknowledgement (ACK) – это сигнал, который передается между взаимодействующими процессами, компьютерами или устройствами для обозначения подтверждения или получения сообщения в рамках протокола связи.

Блок-диаграмма модуля I2C представлена на рисунке 5.

Figure 161. I²C block diagram

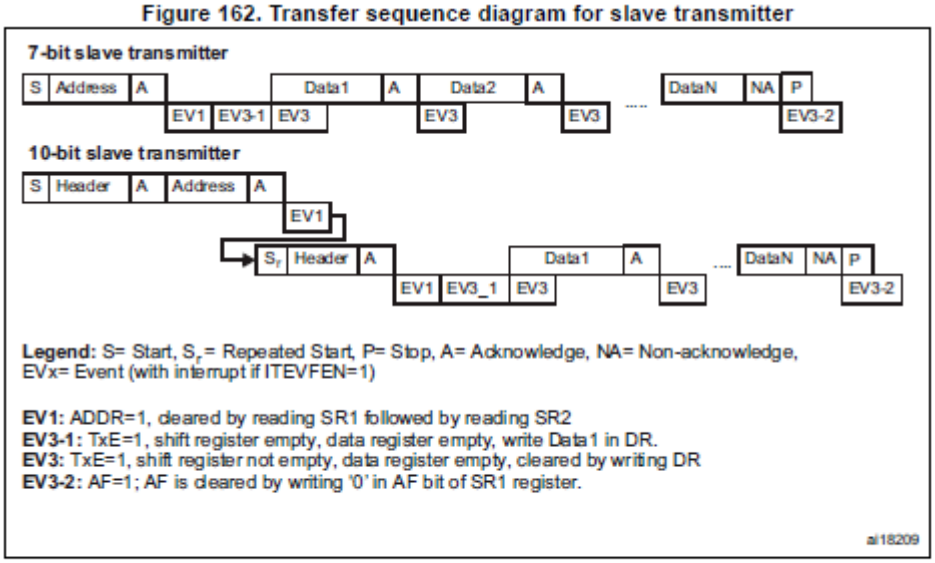


1. SMBA is an optional signal in SMBus mode. This signal is not applicable if SMBus is disabled.

Рисунок 5 – Блок-диаграмма модуля I2C

Диаграмма последовательности передачи данных для slave-устройства, отправляющего данные, представлена на рисунке 6.

Figure 162. Transfer sequence diagram for slave transmitter

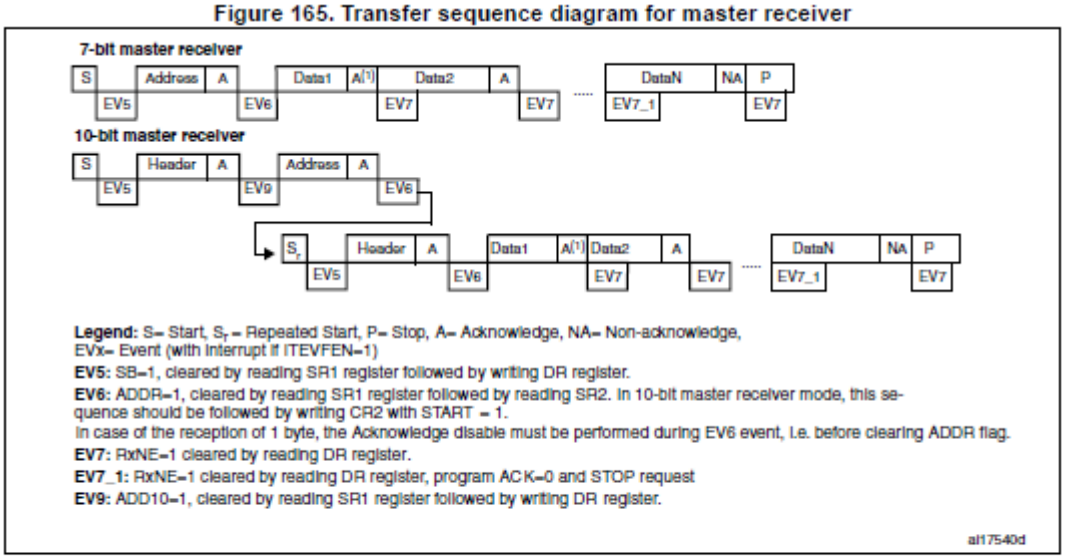


1. The EV1 and EV3_1 events stretch SCL low until the end of the corresponding software sequence.
2. The EV3 event stretches SCL low if the software sequence is not completed before the end of the next byte transmission.

Рисунок 6 – Диаграмма последовательности передачи данных для slave-устройства, отправляющего данные

Диаграмма последовательности приёма данных для master-устройства представлена на рисунке 7.

Figure 165. Transfer sequence diagram for master receiver



1. If a single byte is received, it is NA.
2. The EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
3. The EV7 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
4. The EV7_1 software sequence must be completed before the ACK pulse of the current byte transfer.

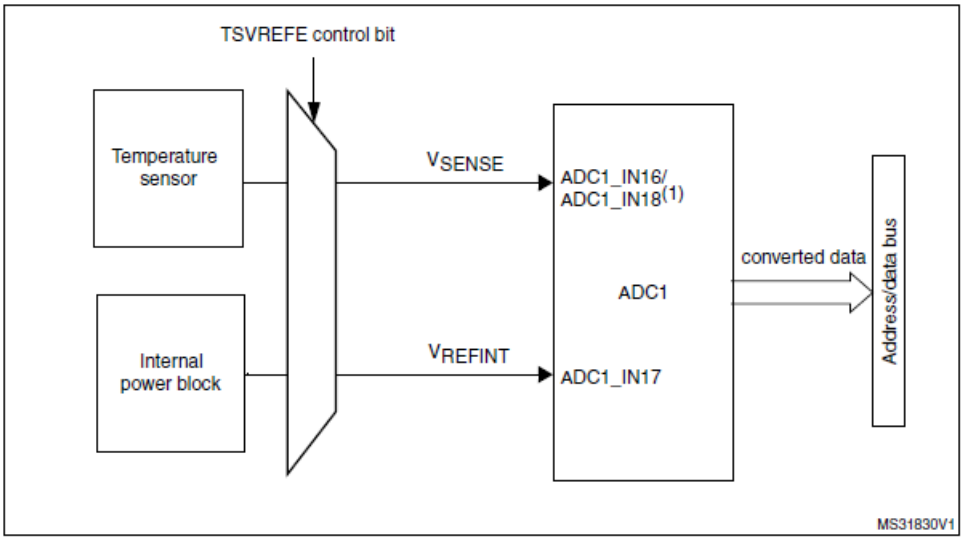
Рисунок 7 – Диаграмма последовательности приёма данных для master-устройства

1.8. Устройство должно измерять температуру со встроенного в микроконтроллер датчика температуры

Устройство должно измерять температуру со встроенного в микроконтроллер датчика температуры.

Встроенный температурный датчик может быть использован для измерения окружающей температуры устройства.

На рисунке 8 представлена блок-схема температурного датчика.



1. V_{SENSE} is input to ADC1_IN18

Рисунок 8 – Блок-схема температурного датчика

Температурный датчик измеряет в диапазоне от -40 до 125 °C с точностью до 1.5°C.

Для расчёта температуры T_C (в градусах Цельсия) используется следующая формула:

$$T_C = \frac{V_{SENSE} - V_{25}}{Avg\_Slope} + 25, \quad (7)$$

где V_{SENSE} – напряжение на выходе температурного датчика;

$V_{25} = 0.76$ В – напряжение при 25°C;

$Avg\_Slope = 0.25$ мВ/°C – средний наклон линии, описывающей зависимость между температурой и напряжением V_{SENSE}.

Для того, чтобы конвертировать напряжение с выхода температурного датчика в двоичные значения, которые можно записать в регистр DR, используется 12-разрядный АЦП.

Поэтому после обращения к данным, хранящимся в регистре DR

$$V_{SENSE} = CV \cdot \frac{VDD}{ADC_{max}}, \quad (8)$$

где CV – конвертированные значения (от англ. Converted Values);

$V_{ref} = 3.3$ В – референтное напряжение на АЦП;

$ADC_{max} = 4096$ – максимальное значение АЦП.

Подключается температурный датчик к следующим каналам АЦП: ADC_IN16 или ADC_IN18.

В микроконтроллере STM32F411RE недоступны ADC2 и ADC3, поэтому для измерения температуры будем подключаться к ADC1.

АЦП имеет до 19 мультиплексированных каналов, что позволяет измерять сигналы от 16 внешних источников, двух внутренних источников и канала V_{BAT}. Аналого-цифровое преобразование каналов может выполняться в одиночном, непрерывном, сканированном или прерывистом режиме (в данной работе реализуется одиночное преобразование).

Датчик температуры внутренне подключен к каналу ADC1_IN18, который используется совместно с V_{BAT}. Необходимо выбирать либо датчик температуры, либо V_{BAT}, с выполнением лишь одного преобразование. Когда на канале ADC1_IN18 выбраны и датчик температуры, и преобразование V_{BAT}, выполняется только преобразование V_{BAT}.

АЦП имеет две схемы синхронизации: тактирование для аналоговой схемы (ADCCLK) и тактирование для цифрового интерфейса (используются для чтения/записи регистров). Для того, чтобы можно было считывать данные из регистров, будем реализовывать второй вариант.

Синхронизация цифрового интерфейса может быть включена или отключена индивидуально для каждого АЦП через регистр включения тактирования периферийных устройств RCC_APB2ENR.

Итак, алгоритм настройки АЦП для измерения температуры следующий:

1) ADC1 подключается к шине APB2 через регистр RCC_APB2ENR,

2) выбирается 18-й канал через регистр ADC_SQR3;

3) в регистре ADC_CR1 в битах RES должно устанавливаться 00 (в данной реализации выставить это значение можно с помощью Bits12);

4) в регистре ADC_CR2 в битах CONT и EOCS выставим значения, соответствующие единичному преобразованию;

5) в регистре ADC_SQR1 в битах L выставляется длина последовательности для регулярных каналов;

6) в регистре ADC_SMPR1, отвечающему за время дискретизации для каналов АЦП от 10-го до 18-го, выберем время сэмплирования для 18-го канала, равное 84 циклам.

В конце активируем АЦП путём выставления бита ADON в регистре ADC_CR2.

В коде эти действия прописываются следующим образом:

```
    RCC::APB2ENR::ADC1EN::Enable::Set();
    ADC1::SQR3::SQ1::Channel18::Set();        // Select ADC1_IN16 or ADC1_IN18
 input channel.
    ADC_Common::CCR::TSVREFE::Enable::Set();  // Set the TSVREFE bit in the AD
C_CCR register to wake up the temperature sensor.
    ADC1::CR1::RES::Bits12::Set();
    ADC1::CR2::CONT::SingleConversion::Set();
    ADC1::CR2::EOCS::SingleConversion::Set();
    ADC1::SQR1::L::Conversions1::Set();
    ADC1::SMPR1::SMP18::Cycles84::Set();
    ADC1::CR2::ADON::Enable::Set();
```

1.9. Вывод значений должен осуществляться через USART2 на скорости 19200 кБит/с

Микроконтроллер STM32F411RE имеет три универсальных синхронных/асинхронных приёмо-передатчиков (USART1, USART2 и USART6).

Эти три интерфейса обеспечивают асинхронную связь, поддержку IrDA SIR ENDEC (от англ. InfraRed Data Association Encoder-Decoder – кодек ИК-порта), многопроцессорный режим связи, однопроводной полудуплексный режим связи и имеют возможность LIN Master/Slave. Интерфейсы USART1 и USART6 могут осуществлять обмен данными со скоростью до 12,5 Мбит/с. Интерфейс USART2 осуществляет обмен данными со скоростью до 6,25 бит/с.

В данной работе наша задача передавать данные с микроконтроллера, поэтому будем использовать USART в режиме передатчика. Передатчик может отправлять слова данных длиной 8 или 9 бит в зависимости от состояния бита M. Когда бит разрешения передачи (TE) установлен, данные из сдвигового регистра передачи выводятся на вывод TX, а соответствующие тактовые импульсы выводятся на вывод CK.

Количество стоповых битов, передаваемых с каждым символом, можно запрограммировать в CR2, биты 13, 12. Мы будем использовать 1 стоповый бит (он стоит по умолчанию).

USART1 тактируется от шины APB2, поэтому в регистре RCC_APB2ENR выставим бит USART1EN в единицу.

Затем настроить порты A2 и A3 как альтернативные и определить для этих портов альтернативные функции для передачи по USART с помощью USART2.

Под скоростью передачи данных в компьютерных технологиях понимают количество бит данных, которые передаются или обрабатываются за единицу времени [13].

Согласно техническому заданию, скорость передачи данных по USART должна быть равна 19200 кБит/с.

Код настройки USART представлен ниже:

```
// Switch on clock on PortA (for USART2).
RCC::AHB1ENR::GPIOAEN::Enable::Set() ;
```

```
RCC::APB1ENRPack<
  RCC::APB1ENR::USART2EN::Enable
  >::Set() ;

GPIOA::MODERPack<
  GPIOA::MODER::MODER2::Alternate,    // Uart2 TX.
  GPIOA::MODER::MODER3::Alternate     // Uart2 RX.
  >::Set() ;

GPIOA::AFRLPack <
  GPIOA::AFRL::AFRL2::Af7,            // Uart2 TX.
  GPIOA::AFRL::AFRL3::Af7             // Uart2 RX.
  >::Set() ;

USART2::BRR::Write(UartSpeed19200) ;
USART2::CR1::UE::Enable::Set() ;

NVIC::ISER1::Write(1<<6);
```

Для UART пакет (т.е. единица передачи цифровых данных) выглядит, как показано на рисунке 9.

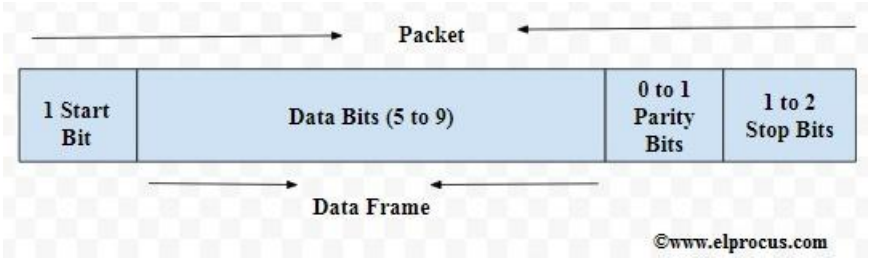


Рисунок 9 – Пакет UART

Измеренные данные кодируются с помощью формата пакета, изображённого на рисунке: первый байт является заголовочным и содержит общую информацию о передаваемых данных (с какого датчика производились измерения, какая величина передаётся, а также контрольная сумма для заголовочного байта). Последующие 4 байта используются для хранения и передачи измеренных значений (подразумевается, что измеренные значений представляют собой значения типа float длиной 4 байта), а последний байт используется в качестве контрольной суммы для всего пакета.

Формат пакета для передачи измеренных данных в данной работе представлен на рисунке 10.

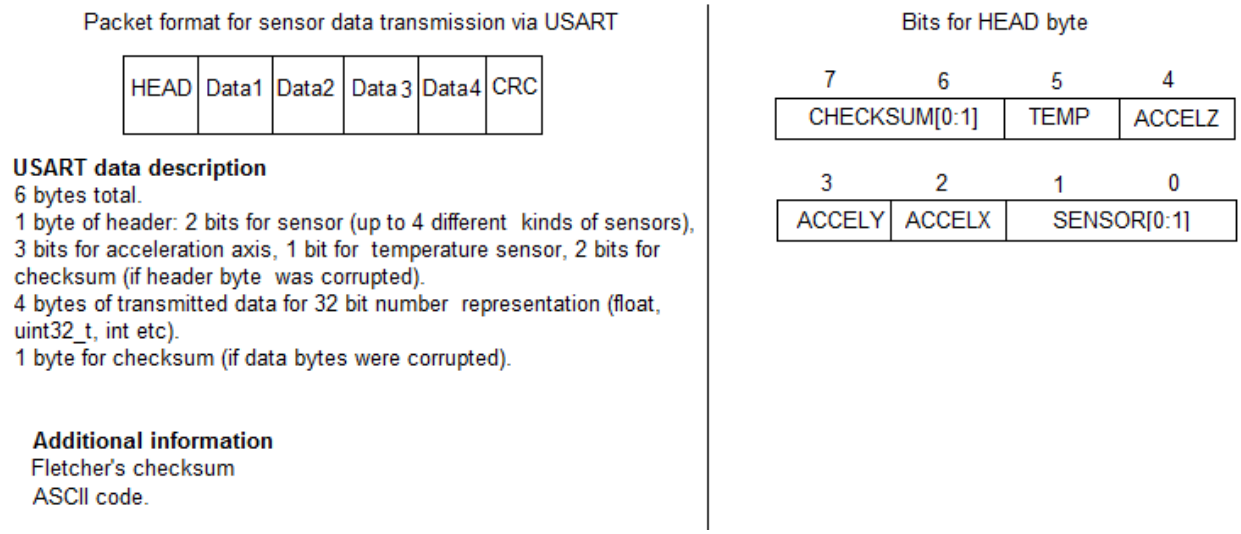


Рисунок 10 – Формат пакета для передачи измеренных данных

Данный формат пакета является настраиваемым, вследствие чего HEAD байт может содержать в себе информацию о том, что передаётся какая-то другая физическая величина (например, данные с трёхосевого акселерометра). Помимо этого может использоваться два HEAD байта, если необходимо значительно увеличить количество измеряемых величин либо, например, уточнить с какого датчика производились измерения (при использовании нескольких датчиков для одной и той же физической величины, соответственно).

Для этого значение, содержащееся в HEAD байте просто проверяется с тем значением, которое может быть выставлено при использовании любого из возможных датчиков. По совпадению этого числа и определяется датчик, с которого были измерены значения.

В HEAD байт можно записывать значения с помощью следующих переменных:

```
#define DT_RESET_BYTE       0x00

#define DT_SENSOR_TEMP      0b00000000
#define DT_SENSOR_ACCEL     0b10000000

#define DT_ACCELX           0b00100000
#define DT_ACCELY           0b00010000
#define DT_ACCELZ           0b00001000
#define DT_TEMP             0b00000100

#define DT_CHECKSUM_0       0b00000000
#define DT_CHECKSUM_1       0b00000001
#define DT_CHECKSUM_2       0b00000010
#define DT_CHECKSUM_3       0b00000011
```

Регистры (переменные в конфигурационном файле dtregisters.h) определены следующим образом:

```
inline char DT_HEAD_TEMP;
inline char DT_DATA0_TEMP;
inline char DT_DATA1_TEMP;
inline char DT_DATA2_TEMP;
inline char DT_DATA3_TEMP;
inline char DT_CRC_TEMP;

inline char DT_HEAD_ACCELX;
inline char DT_DATA0_ACCELX;
inline char DT_DATA1_ACCELX;
inline char DT_DATA2_ACCELX;
inline char DT_DATA3_ACCELX;
inline char DT_CRC_ACCELX;

inline char DT_HEAD_ACCELY;
inline char DT_DATA0_ACCELY;
inline char DT_DATA1_ACCELY;
inline char DT_DATA2_ACCELY;
inline char DT_DATA3_ACCELY;
inline char DT_CRC_ACCELY;

inline char DT_HEAD_ACCELZ;
inline char DT_DATA0_ACCELZ;
inline char DT_DATA1_ACCELZ;
inline char DT_DATA2_ACCELZ;
inline char DT_DATA3_ACCELZ;
inline char DT_CRC_ACCELZ;
```

Распаковку данных в приложении верхнего уровня осуществляет класс ComPort.

Когда пользователь нажимает кнопку Connect на графическом интерфейсе, программа переходит в обработку события. Оттуда потом переходит в метод Config() класса ComPort.

Именно в методе Config() и производится настройка COM-порта (имя COM-порта, скорость передачи данных, проверка чётности и количество стоповых битов):

```
        public void Config(string portName, string baudRate="19200",
            string parity="None", string stopBits="1")
        {
            /* Get a value indicating the open or closed status of the
            SerialPort object.
            Close serial port if it is open at the initial time. */
            if (comPort.IsOpen == true)
            {
                this.Close();
            }

            try
            {
                comPort.PortName = portName;
                comPort.BaudRate = Int32.Parse(baudRate);
                comPort.Parity = (Parity)Enum.Parse(typeof(Parity), parity);
                comPort.StopBits = (StopBits)Enum.Parse(typeof(StopBits), stop
Bits);
                comPort.DataBits = 8;
            }
            catch (System.Exception ex)
            {
                GraphWPF.Exceptions.DisplayException(ex);
            }
        }
```

По мере поступления новых данных на COM-порт вызывает метод обработки данных DataReceived() класса ComPort. В этом методе создаётся массив comBuffer comBuffer (который должен быть фиксированной длины, чтобы не произошло

переполнение памяти) и вызывается метод распаковки данных DecodeMeasuredData():

```
        private void DataReceived(object sender, SerialDataReceivedEventArgs e
)
        {
            try
            {
                /* This piece of code do not allows other threads to use COM-
port
                until you finish to receive data completely. */
                lock (Obj)
                {
                    byte[] comBuffer = new byte[24];
                    comPort.Read(comBuffer, 0, comBuffer.Length);

                    // Unpack receiced message.
                    this.DecodeMeasuredData(comBuffer);
                }
            }
            catch (System.Exception ex)
            {
                GraphWPF.Exceptions.DisplayException(ex);
            }
        }
```

Метод DecodeMeasuredData() имеет битовые значения для каждого предполагаемого датчика, с этими значениями впоследствии сравнивается каждый байт полученных данных. И в зависимости от того, равно ли битовое значение для датчика HEAD байту, определяется, с какого датчика были измерены данные. Затем дальнейшие 4 байта конвертируются в значение типа float, и для температуры вызывается метод изменения текущей температуры моделируемого физического объекта (печатной платы), куда и передаются конвертированное значение.

Код для метода DecodeMeasuredData() представлен ниже:

```
        private void DecodeMeasuredData(byte[] comByte)
        {
            // Bytes for sensor decoding (see dtregisters.h).
            byte TempSensor     = 0b00000000 | 0b00000100;
```

```
            byte AccelerometerX = 0b10000000 | 0b00100000;
            byte AccelerometerY = 0b10000000 | 0b00010000;
            byte AccelerometerZ = 0b10000000 | 0b00001000;

            for (int i = 0; i < comByte.Length; i++)
            {
                /* Size of a packet is 6 bytes in the MeasuringSystem,
                so sensor index is every 6th. */
                if (i % PacketSize == 0)     // Get what sensor sent data.
                {
                    if (comByte[i] == TempSensor)
                    {
                        float value = System.BitConverter.ToSingle(comByte, i+
1);     // Get 4 bytes.
                        _CurcuitBoard.SetTemperature(value);
                    }
                    else if (comByte[i] == AccelerometerX)
                    {
                        float value = System.BitConverter.ToSingle(comByte, i+
1);     // Get 4 bytes.
                    }
                    else if (comByte[i] == AccelerometerY)
                    {
                        float value = System.BitConverter.ToSingle(comByte, i+
1);     // Get 4 bytes.
                    }
                    else if (comByte[i] == AccelerometerZ)
                    {
                        float value = System.BitConverter.ToSingle(comByte, i+
1);     // Get 4 bytes.
                    }
                }
            }
        }
```

1.10. Период вывода информации раз в 100ms

Необходимо обеспечить отправку измеренных значений по USART2 и обновление графических элементов каждые 100 мс.

Отправку измеренных значений по USART2 можно реализовать с помощью операционной системы реального времени (ОСРВ), обновление графических элементов реализуется с помощью таймеров.

1.11. Программа на ПК должна отображать положение объекта в 3D

Графический интерфейс выполнен с использованием графической подсистемы WPF с использованием языка программирования C#.

На рисунке 11 представлен эскиз проектируемого графического интерфейса, созданный с помощью Paint с целью понимания, из каких элементов будет состоять графический интерфейс, и где эти элементы будут располагаться (финальная версия графического интерфейса может отличаться от эскиза).

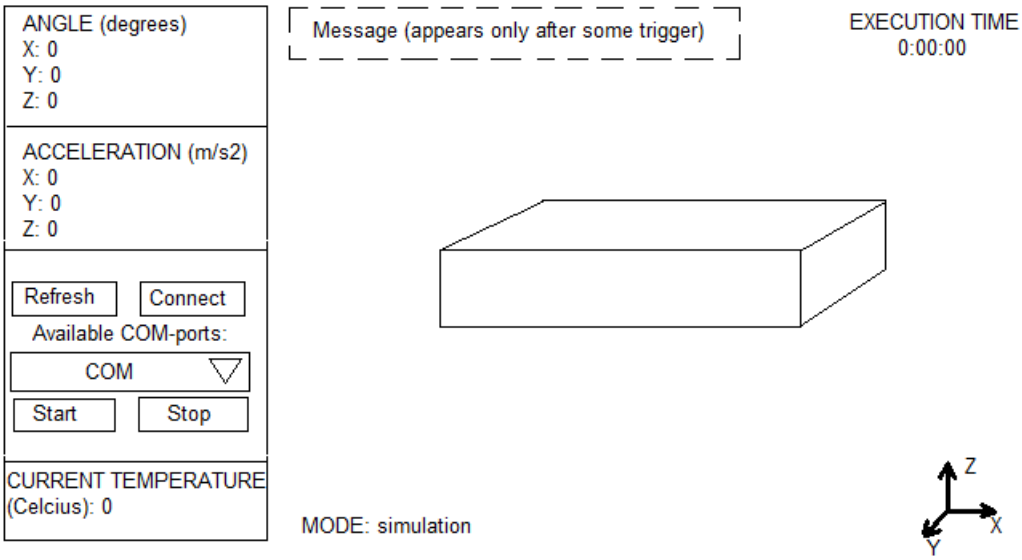


Рисунок 11 – Примерное изображение проектируемого графического интерфейса

Трёхмерная модель выполнена с помощью XAML-разметки.

Для переносов, масштабирования и поворотов 3D-объектов используются матрицы поворота, сдвига и масштабирования. При их умножении на вектор $\bar{A} = (a_x, a_y, a_z)$ происходит его поворот вокруг соответствующей оси, параллельный перенос или изменение длины.

Вращение в трехмерном пространстве представимо в виде последовательности поворотов вектора $\bar{A}$ вокруг трех взаимно перпендикулярных осей, например, вокруг осей декартовой системы координат *OXYZ* . Соответствующие матрицы поворота имеют вид:

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \quad (9)$$

$$M_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \quad (10)$$

$$M_z(\alpha) = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (11)$$

Для правой системы координат положительным направлением вращения считается движение вектора против часовой стрелки. При повороте вектора $\bar{A} = (a_x, a_y, a_z)$ на угол α вокруг оси *OX* применяются формулы:

$$A' = M_x \cdot A \quad (12)$$

или

$$\begin{pmatrix} a_x' \\ a_y' \\ a_z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \quad (13)$$

Тогда:

$$a_x' = a_x$$

$$a_y' = a_y \cdot \cos(\alpha) - a_z \cdot \sin(\alpha)$$

$$a_z' = a_y \cdot \sin(\alpha) + a_z \cdot \cos(\alpha)$$

Повороту вокруг произвольной оси соответствует матрица, равная произведению соответствующих трех матриц поворота. При переносе и повороте изменяется ориентация геометрической модели объекта по отношению к плоскости экрана, сама модель остается неизменной.

Однако с помощью графической подсистемы WPF платформы .NET можно упростить процесс поворота объекта в трёхмерном пространстве. Для этого

достаточно в XAML разметке использовать класс AxisAngleRotation3D, который позволяет осуществлять поворот трёхмерного объекта относительно оси, заданной с помощью свойства Axis. Например, для поворота объекта вокруг оси X, нужно прописать:

```
<AxisAngleRotation3D x:Name="Model3dRotateAngleX" Axis="1 0 0" />
```

Далее вращать 3D-модель из кода можно путём обращения к Model3dRotateAngleX, меняя свойство Angle, позволяющее задавать угол поворота в градусах:

```
namespace Simulation3d
{
    public partial class MainWindow : Window
    {
        ...
        #region Keyboard handling
         private void KeyUp_Handling(object sender, KeyEventArgs e)
         {
             ...
             if (e.Key == Key.E)         // Rotation around X axis.
             {
                 _CurcuitBoard.SetRotation(5, 0, 0);
                 this.angle = _CurcuitBoard.GetRotation();
                 Model3dRotateAngleX.Angle = this.angle.X;
             }
             else if (e.Key == Key.Q)    // Rotation around X axis.
             {
                 _CurcuitBoard.SetRotation(-5, 0, 0);
                 this.angle = _CurcuitBoard.GetRotation();
                 Model3dRotateAngleX.Angle = this.angle.X;
             }
             ...
             myCanvas.Focus();
         }
         #endregion  // Keyboard handling
    }
}
```

Выставление значения ускорения производится путём вызова метода CircuitBoard.SetAcceleration(), в котором устанавливаются значения ускорения

трёхмерной модели по трём осям, и вычисляется угол поворота с помощью формул (2), (3) и (4).

```
namespace Simulation3d
{
    public class CircuitBoard
    {
        public void SetAcceleration(float dx = 0, float dy = 0, float dz = 0)
        {
            // Adjust all accelerations.
            _Accel.X = dx;
            _Accel.Y = dy;
            _Accel.Z = dz;

            // Calculate rotation using acceleration.
            float dxAngle = (float)System.Math.Atan2(_Accel.Y,
                System.Math.Sqrt(System.Math.Pow(_Accel.X, 2) + System.Math.Po
w(_Accel.Z, 2)));
            float dyAngle = (float)System.Math.Atan2(_Accel.X,
                System.Math.Sqrt(System.Math.Pow(_Accel.Y, 2) + System.Math.Po
w(_Accel.Z, 2)));
            float dzAngle = (float)System.Math.Atan2(System.Math.Sqrt(System.M
ath.Pow(_Accel.X, 2) + System.Math.Pow(_Accel.Y, 2)),
                _Accel.Z);

            this.SetRotation(dxAngle, dyAngle, dzAngle);
        }
```

1.12. Архитектура должна быть представлена в виде UML диаграмм в пакете Star UML

UML-диаграмма – это язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур [14].

StarUML – программный инструмент моделирования, который поддерживает UML (Унифицированный язык моделирования). StarUML ориентирован на UML версии 1.4 и поддерживает одиннадцать различных типов диаграмм, принятых в

нотации UML 2.0. Он активно поддерживает подход MDA (Модельно-управляемая архитектура), реализуя концепцию профилей UML.

Model Driven Architecture (MDA) – новый подход к разработке ПО, предложенный консорциумом OMG. Архитектура MDA может предоставить ряд преимуществ по сравнению с существующими методиками: упрощение разработки многоплатформенных систем, простота смены технологической платформы, повышение скорости разработки и качества разрабатываемых программ и многое другое. Однако всё это станет возможным только тогда, когда среды разработки будут поддерживать новую технологию и полностью реализовывать её потенциал.

1.13. Приложение должно быть написано на языке C++ с использование компилятора ARM 8.40.2

Язык программирования C++ является компилируемым строго типизированным языком программирования общего назначения. Поддерживает разные парадигмы программирования: процедурную, обобщённую, функциональную; наибольшее внимание уделено поддержке объектно-ориентированного программирования.

Особенности C++:

- поддержка объектно-ориентированного программирования через классы (предоставляет все четыре возможности ООП – абстракцию, инкапсуляцию, наследование (в том числе и множественное) и полиморфизм);
- поддержка обобщенного программирования через шаблоны функций и классов;
- стандартная библиотека C++ состоит из стандартной библиотеки C (с некоторыми модификациями) и библиотеки шаблонов (Standard Template Library, STL), которая предоставляет обширный набор обобщенных контейнеров и алгоритмов;
- дополнительные типы данных;

- обработка исключений;
- виртуальные функции;
- пространства имён;
- встраиваемые (inline) функции;
- перегрузка (overloading) операторов;
- перегрузка имен функций;
- ссылки и операторы управления свободно распределяемой памятью.

IAR Embedded Workbench — комплексная среда разработки, многочисленные версии которой поддерживают большинство микроконтроллеров от различных производителей. Прежде всего, необходимо пояснить, что IAR EW является не просто программой, а именно комплексной средой разработки, поскольку состоит из целого комплекса программных и аппаратных инструментов. И хотя, как правило, аппаратные отладчики не поставляются в случае покупки только лицензии на пакет, так как в среду включена поддержка J TAG-адаптеров различных фирм, тем не менее, компания IAR рекомендует использовать в комплекте аппаратное обеспечение выпускаемое ею.

IAR Embedded Workbench предлагает обширную поддержку 8, 16 и 32-битных микроконтроллеров (MCU). Благодаря высокой скорости процессов оптимизации IAR Embedded Workbench позволяет быстро генерировать и выполнять необходимый код. Продукт IAR Embedded Workbench создан в сотрудничестве с ведущими мировыми IT-вендорами (Apple, Black&Decker, Cisco Systems, Ember, Ericsson, Hewlett-Packard, Motorola, Panasonic, Siemens и др.), что обеспечивает наиболее обширную поддержку архитектур процессоров на рынке.

По всему миру используется уже свыше 100 тысяч лицензий на ПО IAR Embedded Workbench. Многие успешные производители продуктов со встроенным программным обеспечением используют данное решение для критически важных приложений, в том числе в медицинской, промышленной и коммуникационной сфере. Надежность ПО IAR Embedded Workbench подтверждена многочисленными независимыми и коммерческими тестами.

Собственная технология IAR Systems позволяет в рамках одного инструментария легко перемещаться между 8, 16 и 32-битными микроконтроллерами с любой релевантной архитектурой, включая ядра ARM. Для стандартизации процесса разработки единый инструментарий также позволяет многократно использовать код в разных проектах.

1.14. При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и C++ обертка над ней

При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и C++ обертка над ней.

Ввиду того, что измерительная часть проектируемой системы должна выполнять сразу две задачи параллельно: измерение данных и их передачу, то можно использовать операционную систему реального времени.

Системное программное обеспечение (англ. System Software) – это программное обеспечение, предназначенное для обеспечения платформы для другого программного обеспечения. Примеры системного программного обеспечения включают операционные системы (macOS, Linux, Android и Microsoft Windows), программное обеспечение для научных вычислений, игровые движки и промышленную автоматизацию [15].

Операционная система (англ. Operating System) – это системное программное обеспечение, которое управляет компьютерным оборудованием, ресурсами программного обеспечения и предоставляет общие услуги для компьютерных программ [16].

Операционной системой реального времени (англ. Real-time operating system) называется операционная система, предназначенная для обслуживания приложений реального времени, которые позволяют обрабатывать данные по мере их поступления, обычно без задержек в буфере [17].

Операционная система реального времени FreeRTOS является полностью бесплатной ОСРВ с открытым исходным кодом, причём лицензия не накладывает

никаких ограничений в плане её использования как в коммерческом, так и некоммерческом ПО [18].

На официальном сайте FreeRTOS [18] можно ознакомиться с подробной документацией к данной ОСРВ и скачать исходный код. Также скачать исходный код можно по соответствующей ссылке [19].

По ссылке [1] представлен исходный код обёртки над FreeRTOS, написанной на C++ и используемой в данной работе; там же можно ознакомиться с тем, как пользоваться этой обёрткой.

Например, можно использовать операционную систему реального времени, чтобы регулировать не только измерения и выполнение расчётов, но и отправку данных через UART.

В данной работе используются две задачи: задача измерения и задача передачи данных.

2 ОПИСАНИЕ ФУНКЦИОНАЛЬНОЙ СХЕМЫ УСТРОЙСТВА

Согласно стандарту [20], функциональной схемой (или функциональной цепью) называется совокупность элементов, функциональных групп и устройств (или совокупность функциональных частей) с линиями взаимосвязей, образующих канал или тракт определенного назначения.

Линия взаимосвязи – это отрезок линии, указывающей на наличие связи между функциональными частями изделия [20].

Функциональная схема устройства представлена на рисунке 12.

Рисунок 12 – Функциональная схема проектируемого устройства

Устройство – секундомер состоит из следующих функциональных блоков:

- Встроенный температурный датчик для измерения температуры;
- Акселерометр ADXL345 для измерения ускорения;
- Микроконтроллер, который осуществляет прием измерение данных с помощью встроенного температурного датчика, их фильтрацию и передачу данных на устройство UART;
- Устройство UART для передачи данных с микроконтроллера на компьютер;
- Персональный компьютер, на котором исполняется программа, отображающая измеренные данные температуры и ускорения в числовом виде и имитирующая поворот платы в трёхмерном пространстве.

3 ПРОГРАММНАЯ АРХИТЕКТУРА СИСТЕМЫ

Данный раздел содержит описание составных частей системы и того, как эти части между собой взаимодействуют.

3.1. Общая архитектура

Программа измерительной системы состоит из следующих компонентов:

1) Application (уровень приложения) – здесь задачи обращаются к классу, отвечающему за измерение температуры, и к классу, отвечающему за передачу данных;

2) Measurement (уровень физических величин, которые измеряются) – здесь производится обращение к измерительным устройствам, определённым в программном компоненте Peripherals, и фильтрам.

3) Transmission (уровень создания пакетов данных для передачи) – здесь формируются пакеты для передачи данных.

4) Peripherals (уровень периферийных устройств) – здесь определены измерительные приборы, а также драйвера, управляющие USART, I2C и АЦП.

5) Filters – здесь определён фильтр, с помощью которого осуществляется фильтрация измеренных значений температуры и ускорения.

6) CPU – здесь определены классы, обеспечивающие обращение к регистрам микроконтроллера для реализации функций USART и АЦП.

Диаграмма компонентов измерительной системы представлена на рисунке 13.

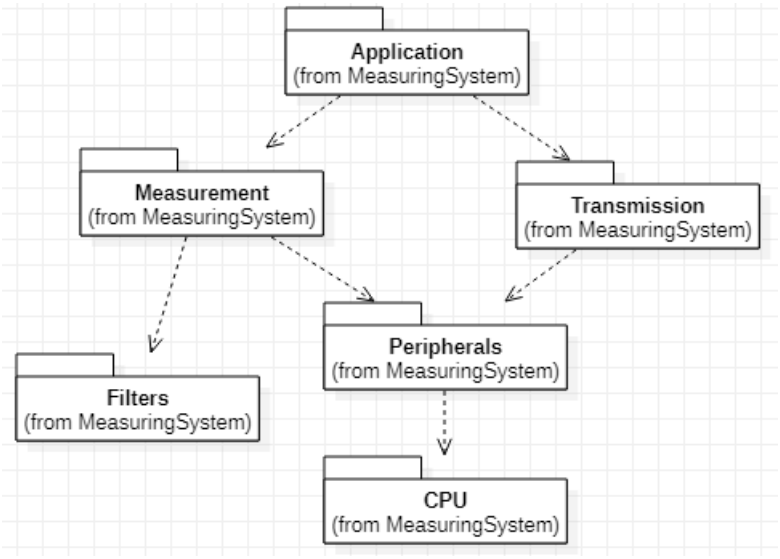


Рисунок 13 – Диаграмма компонентов измерительной системы

Диаграмма классов измерительной системы представлена на рисунке 14.

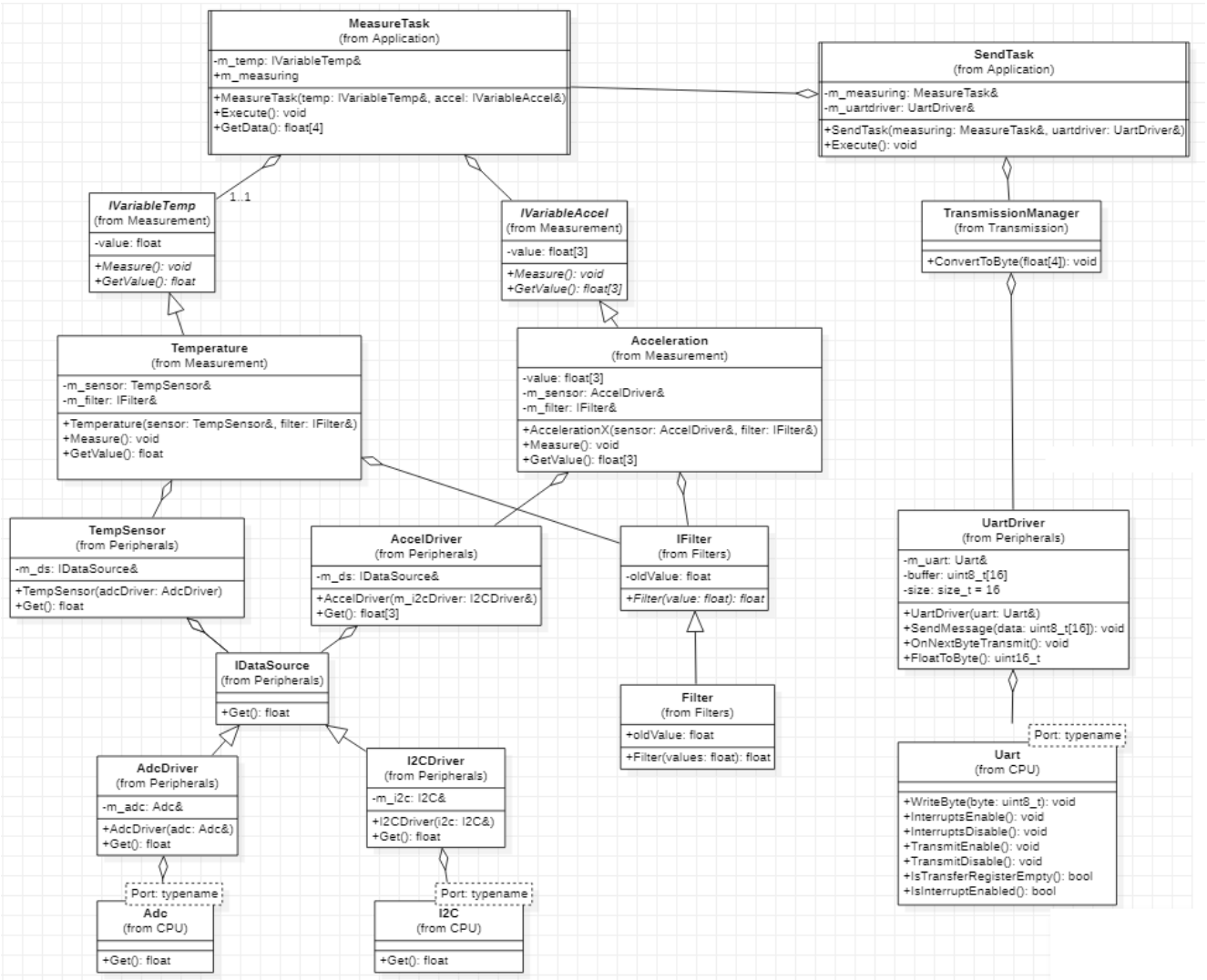


Рисунок 14 – Диаграмма классов измерительной системы

3.2. Детальная архитектура

Детальное описание классов представлено на рисунках 15–28.

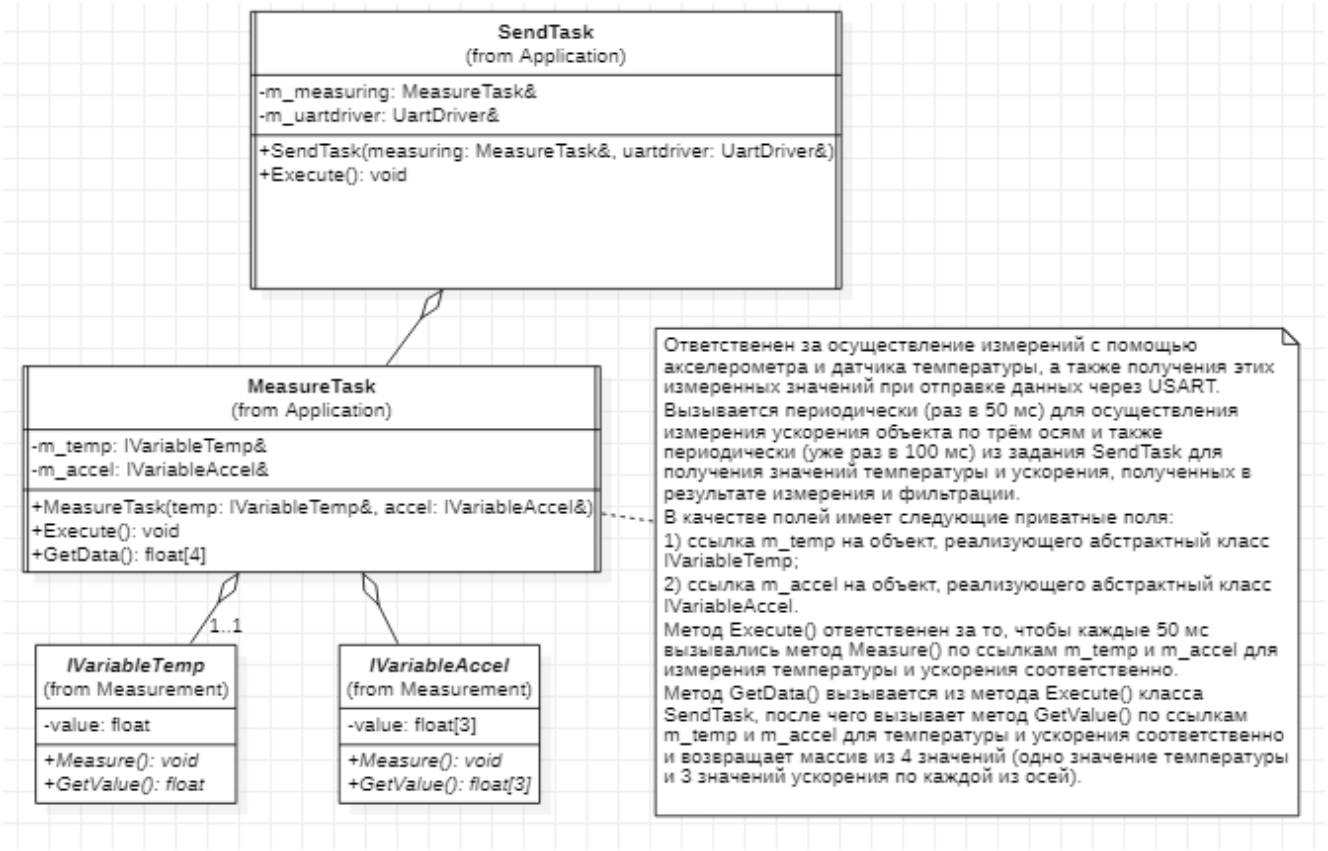


Рисунок 15 – Класс MeasureTask

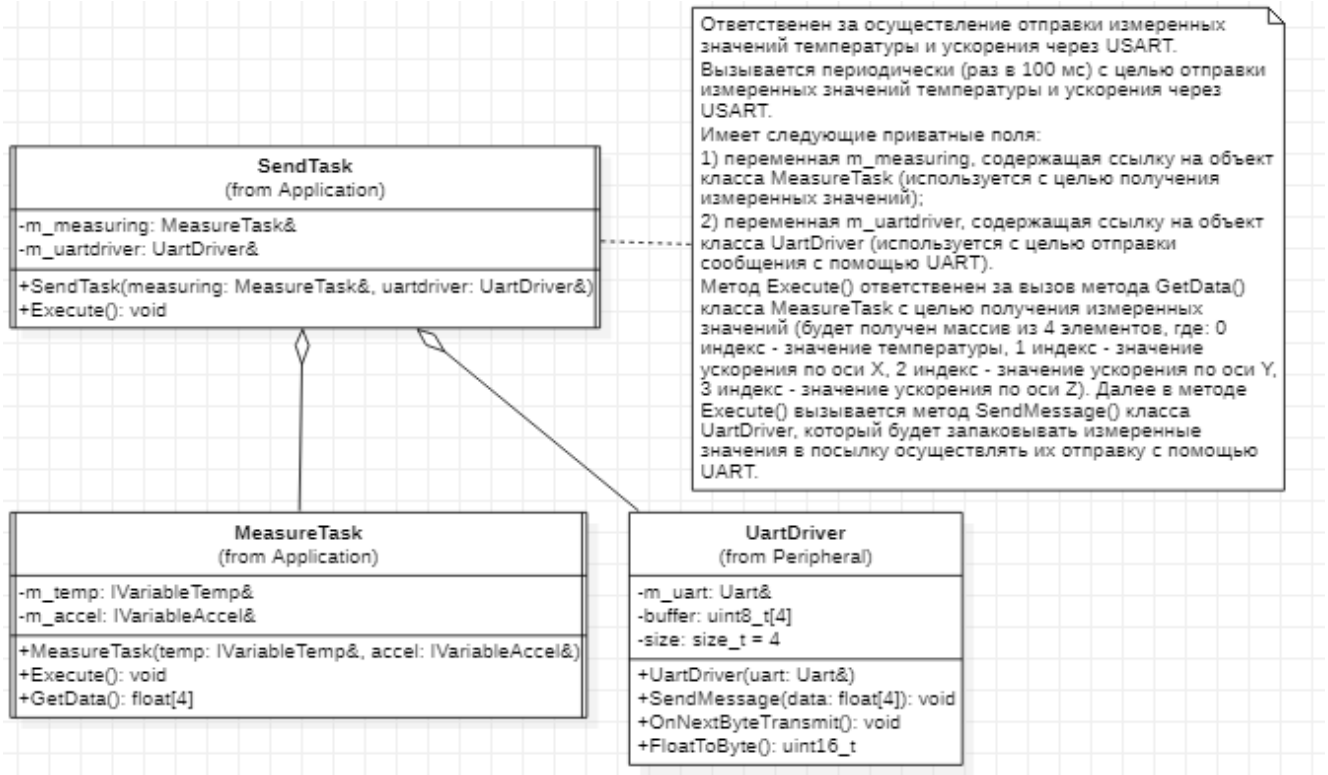


Рисунок 16 – Класс SendTask

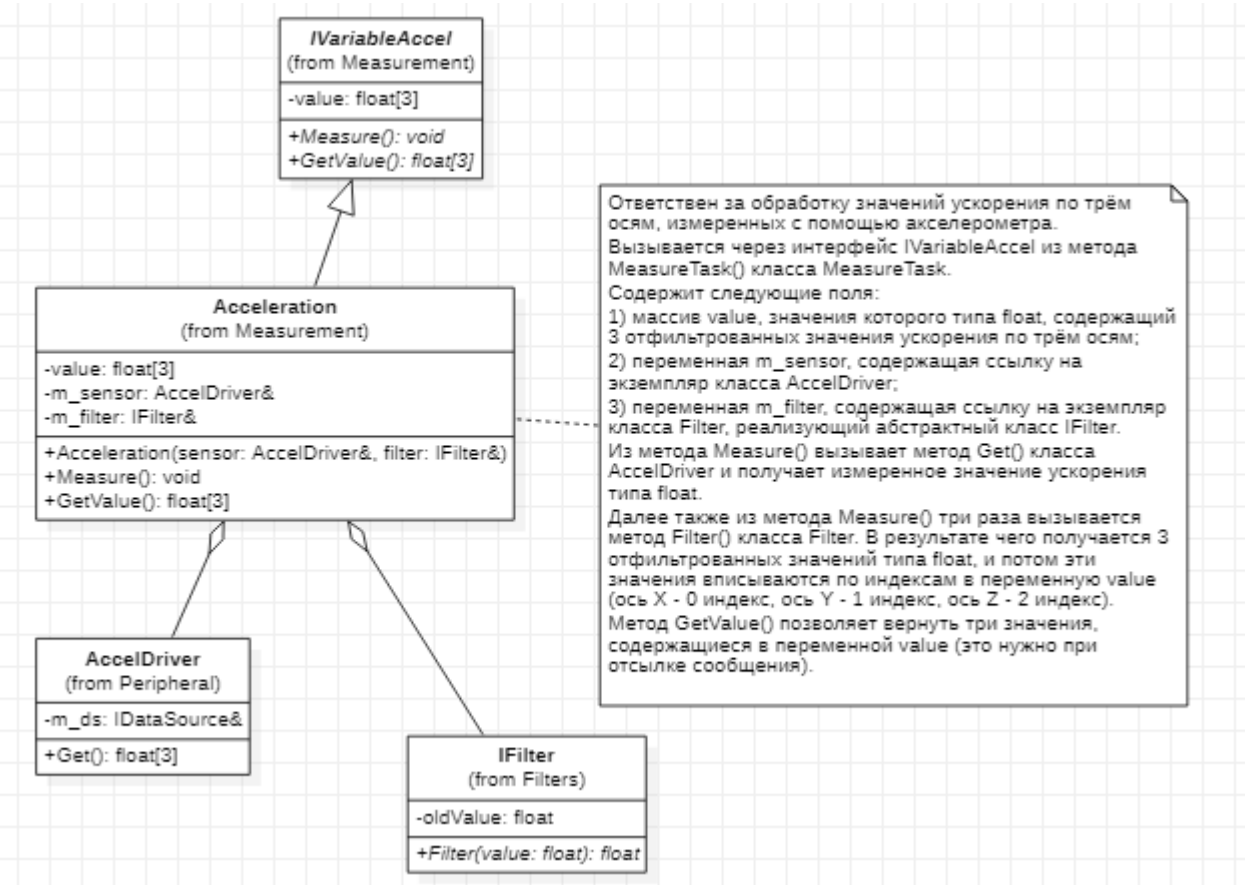


Рисунок 17 – Класс Acceleration

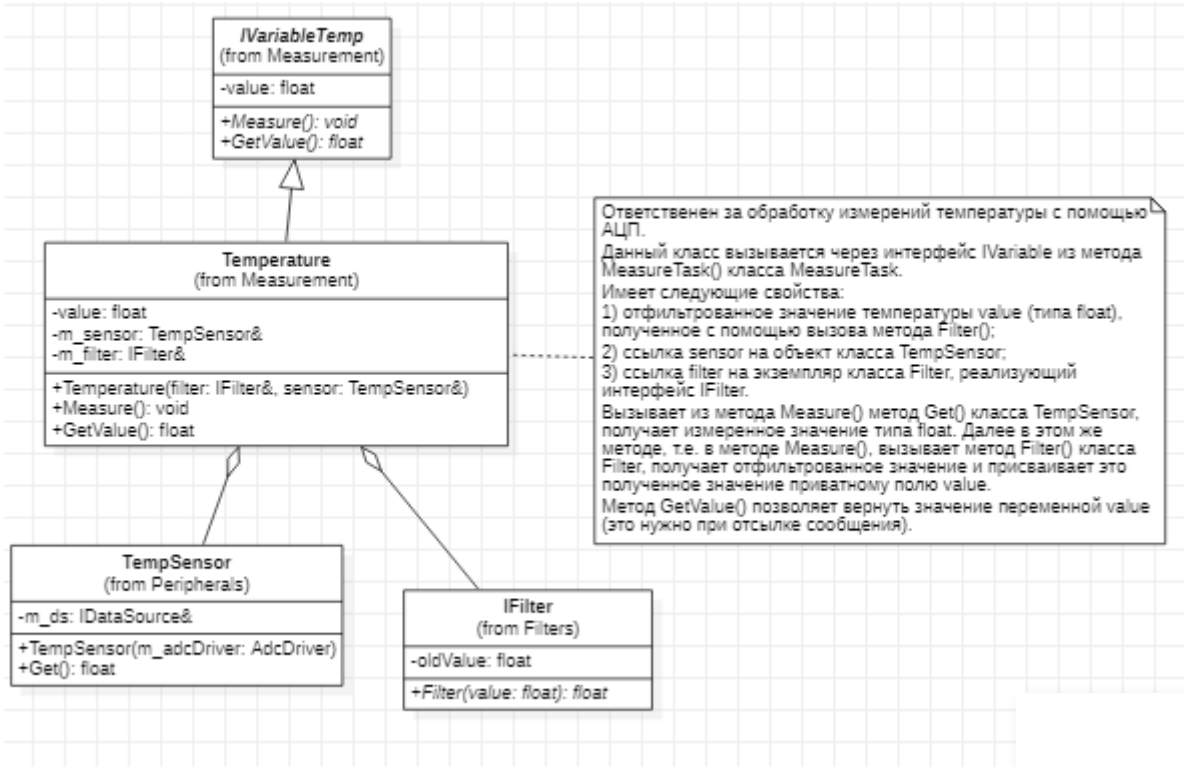


Рисунок 18 – Класс Temperature

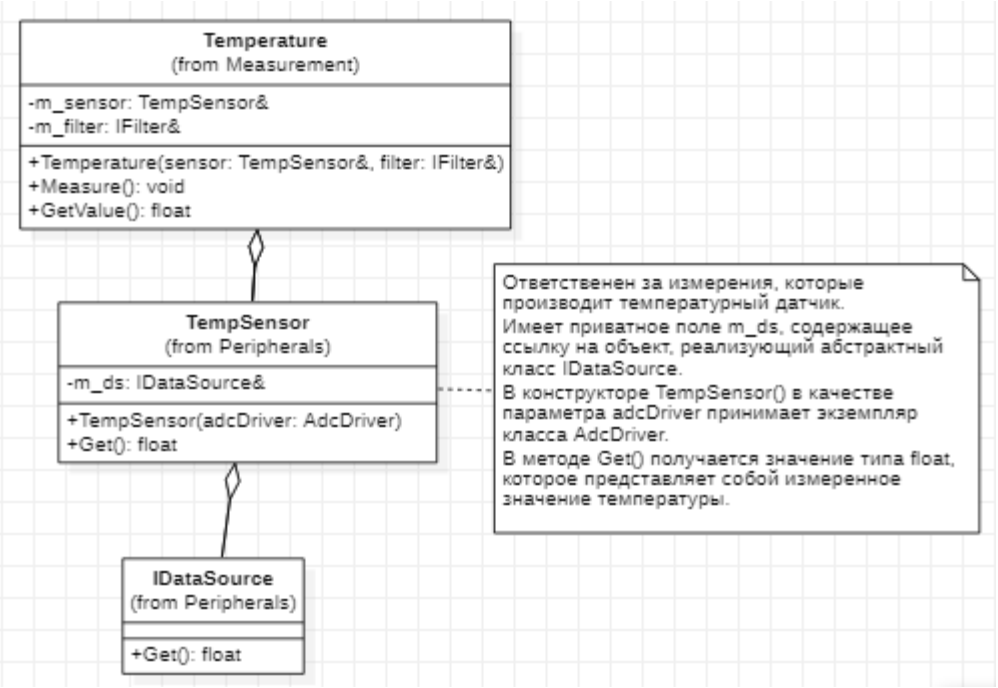


Рисунок 19 – Класс TempSensor

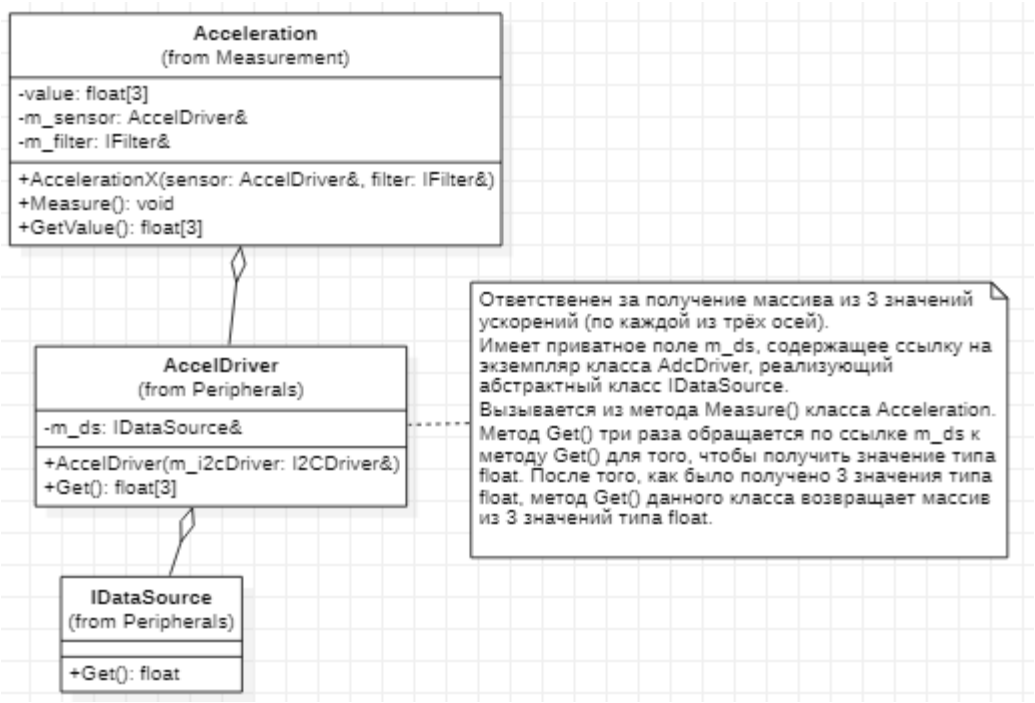


Рисунок 20 – Класс AccelDriver

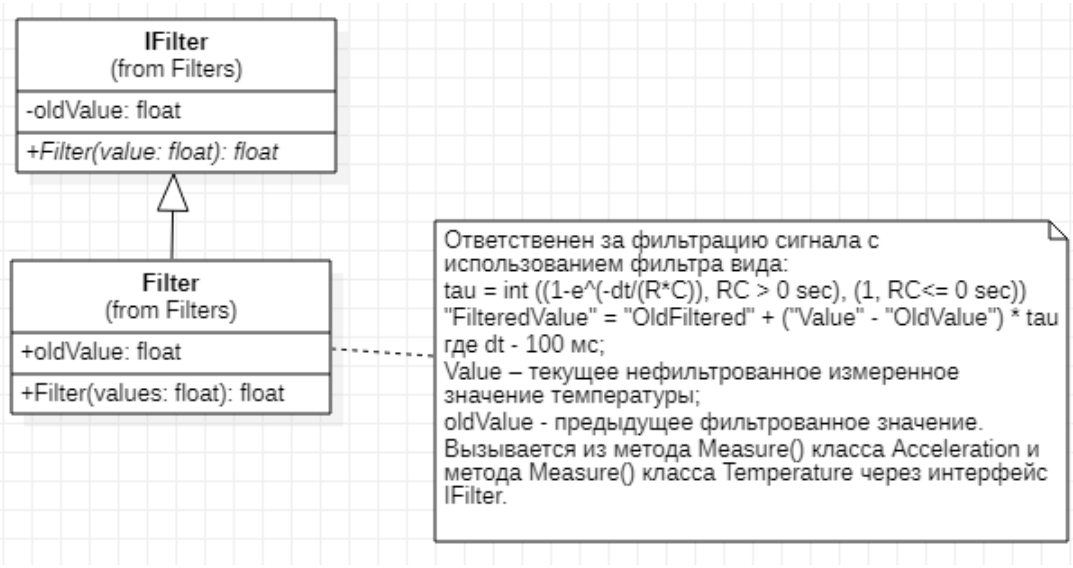


Рисунок 21 – Класс Fileter

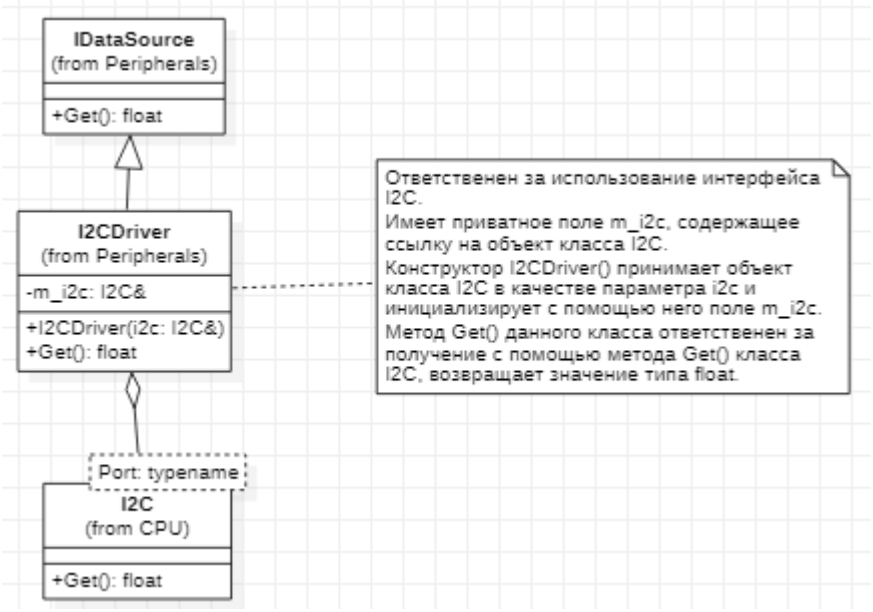


Рисунок 22 – Класс I2CDriver

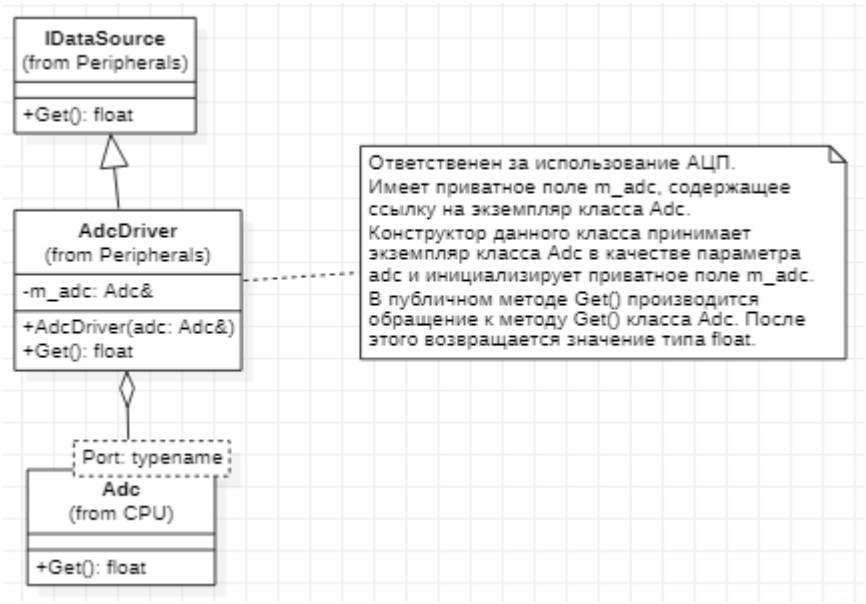


Рисунок 23 – Класс AdcDriver

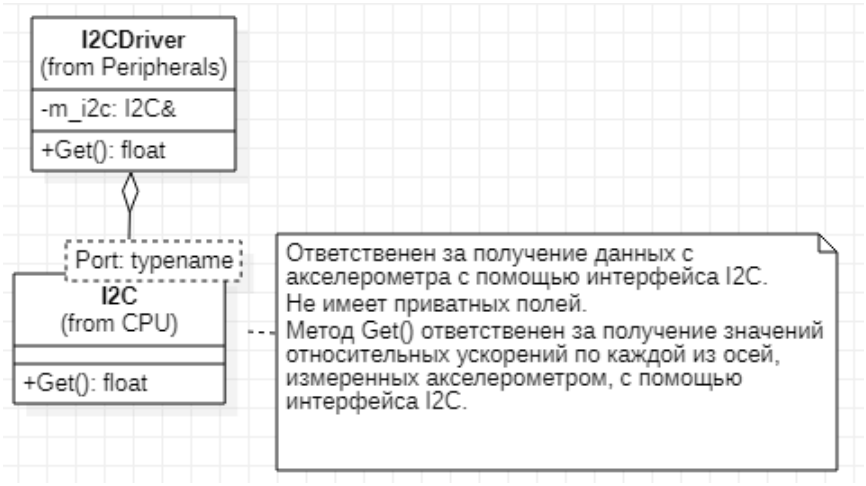


Рисунок 24 – Класс I2C

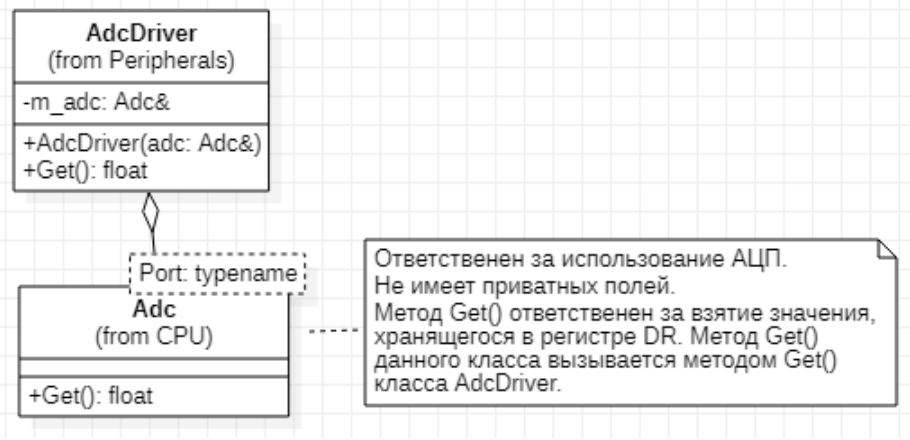


Рисунок 25 – Класс Adc

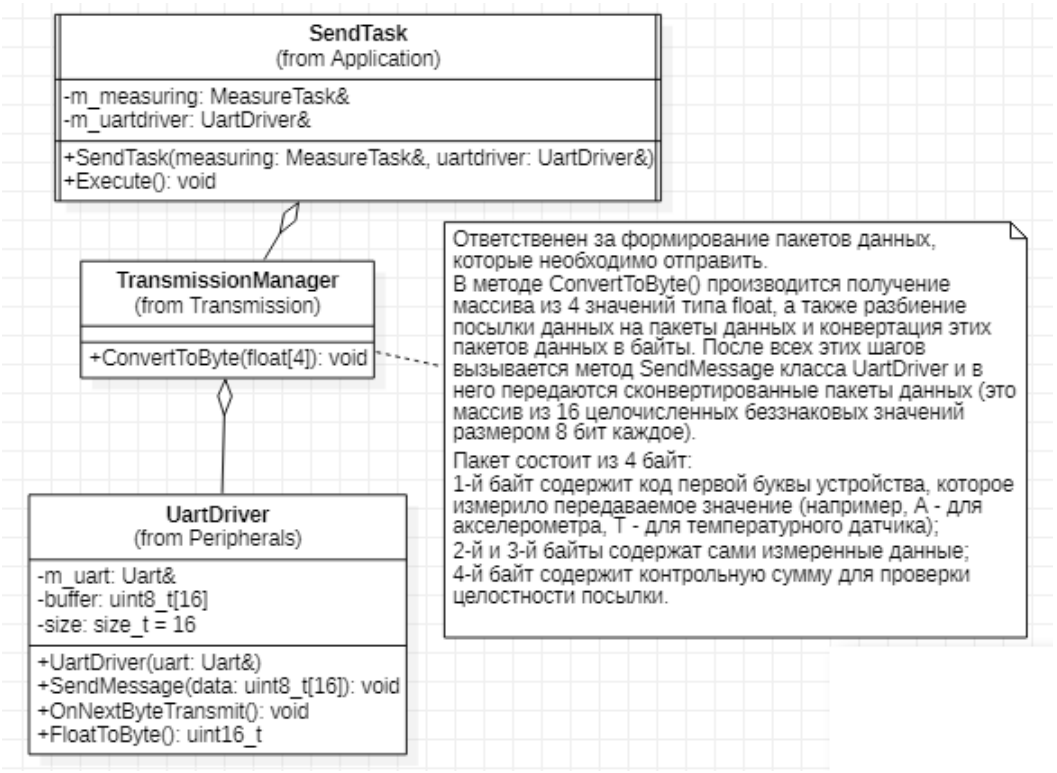


Рисунок 26 – Класс TransmissionManager

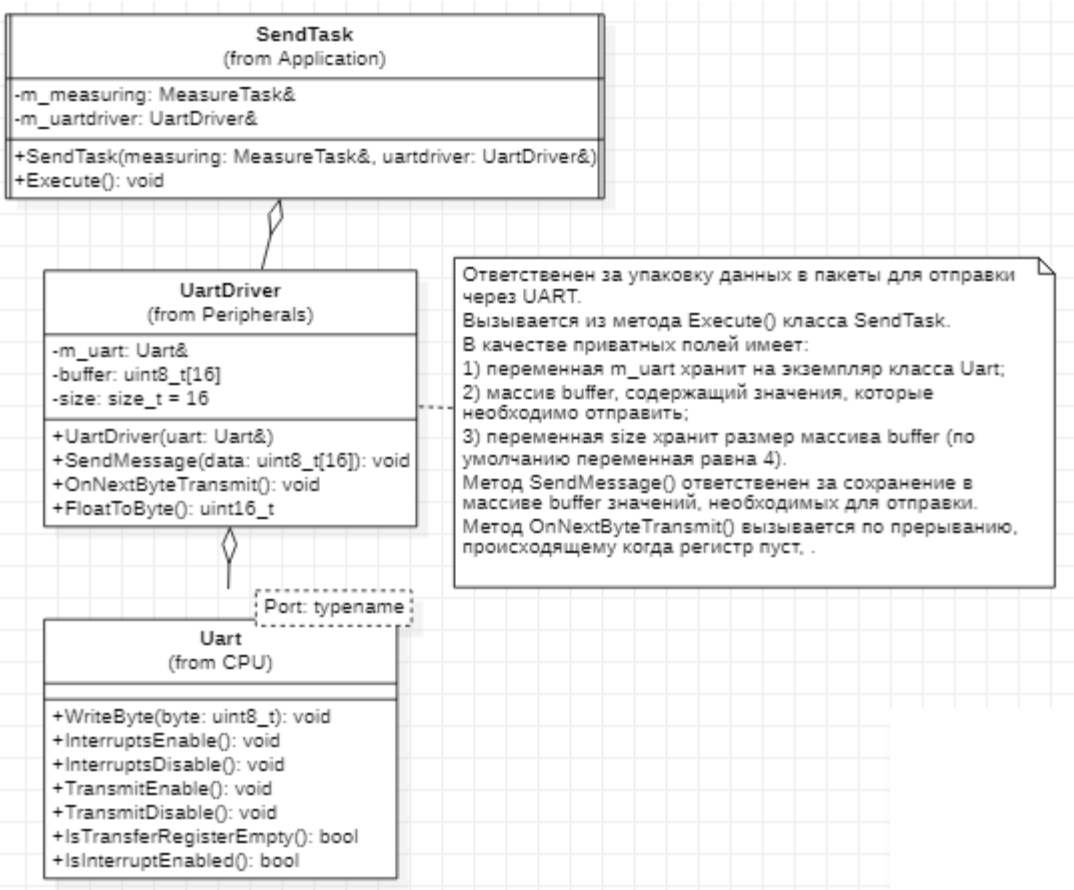


Рисунок 27 – Класс UartDriver

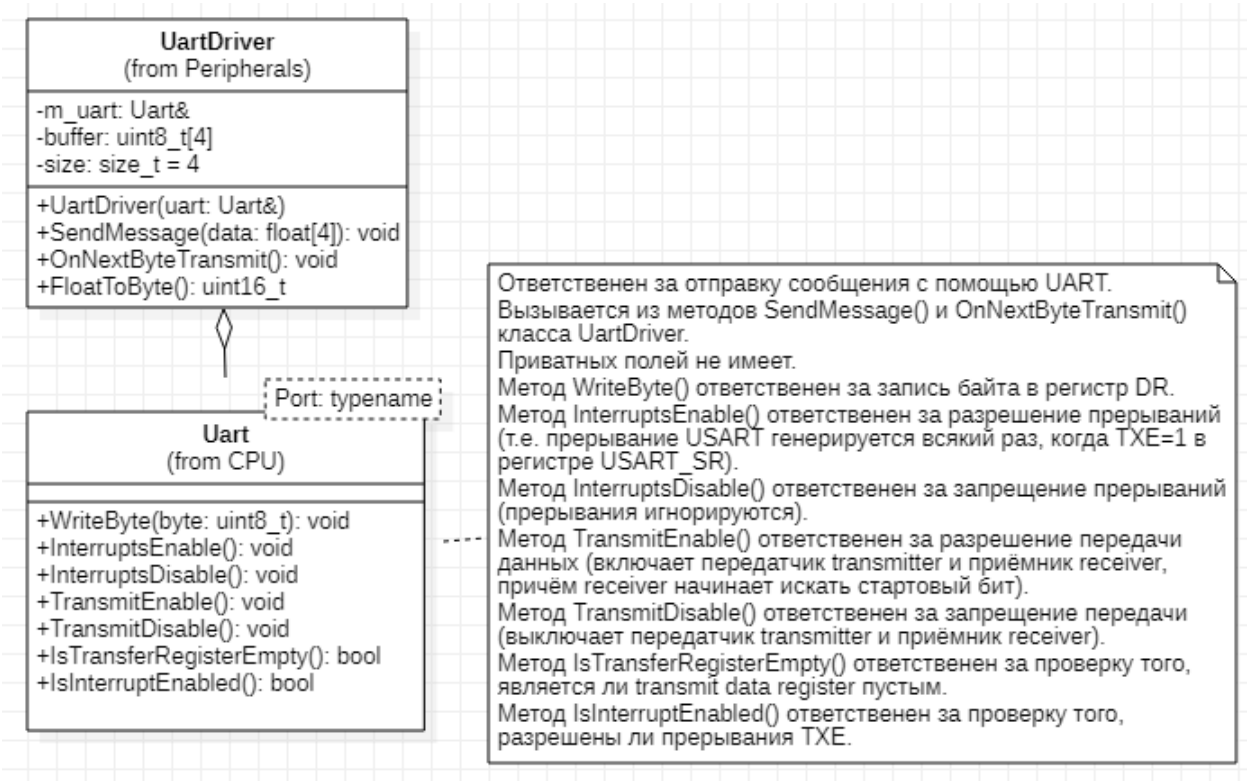


Рисунок 28 – Класс Uart

4 ДЕМОНСТРАЦИЯ РАБОТОСПОСОБНОСТИ СИСТЕМЫ

Начало работы с графическим интерфейсом при запуске программы верхнего уровня представлено на рисунке 29.

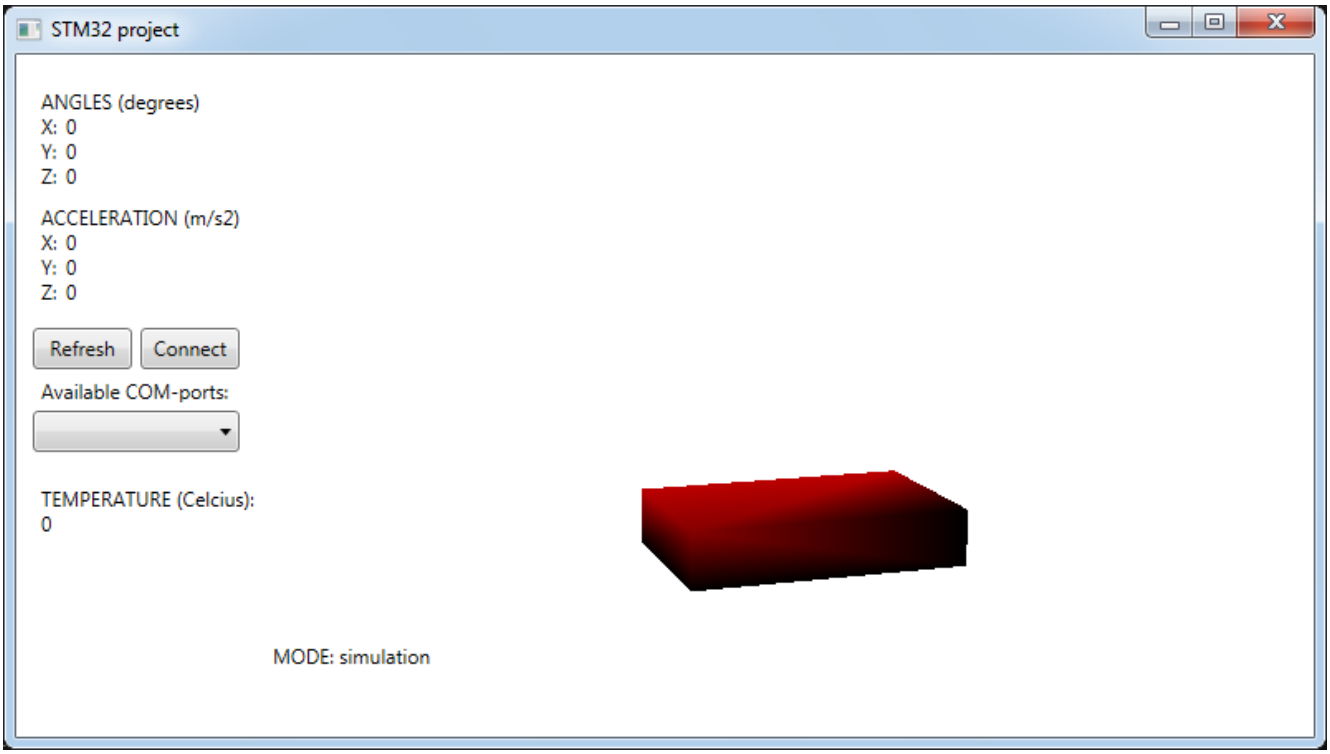


Рисунок 29 – Начало работы с графическим интерфейсом

Ускорение можно выставить с помощью нажатия клавиш A, S, W и D. При единичном нажатии на клавишу значение ускорения увеличивается на 5 единиц, после чего каждые 100 мс уменьшается на единицу.

Вращение вокруг оси X производится путём нажатия на клавиши Q и E, вокруг оси Y – на клавиши R и F, вокруг оси Z – на клавиши Z и X.

На рисунках 30–32 представлен результат вращения 3D-модели в режиме симуляции.

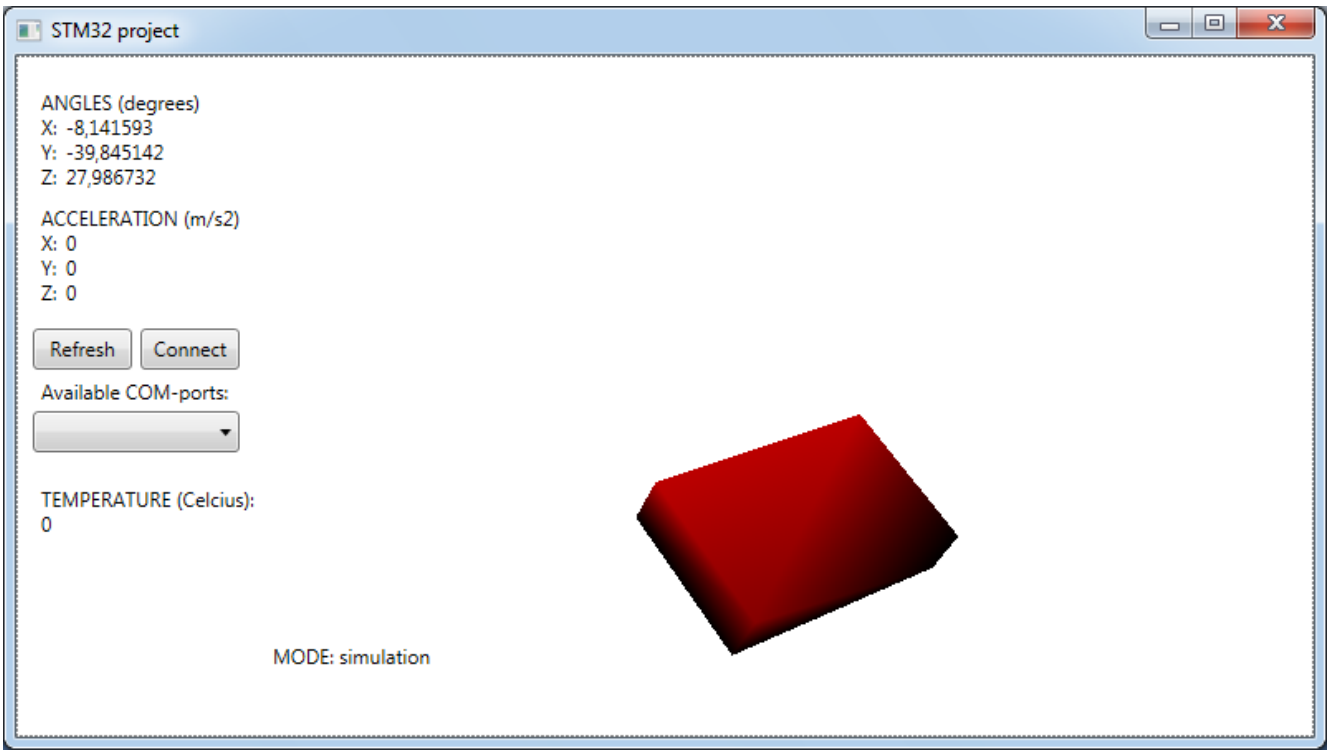


Рисунок 30 – Поворот трёхмерной модели

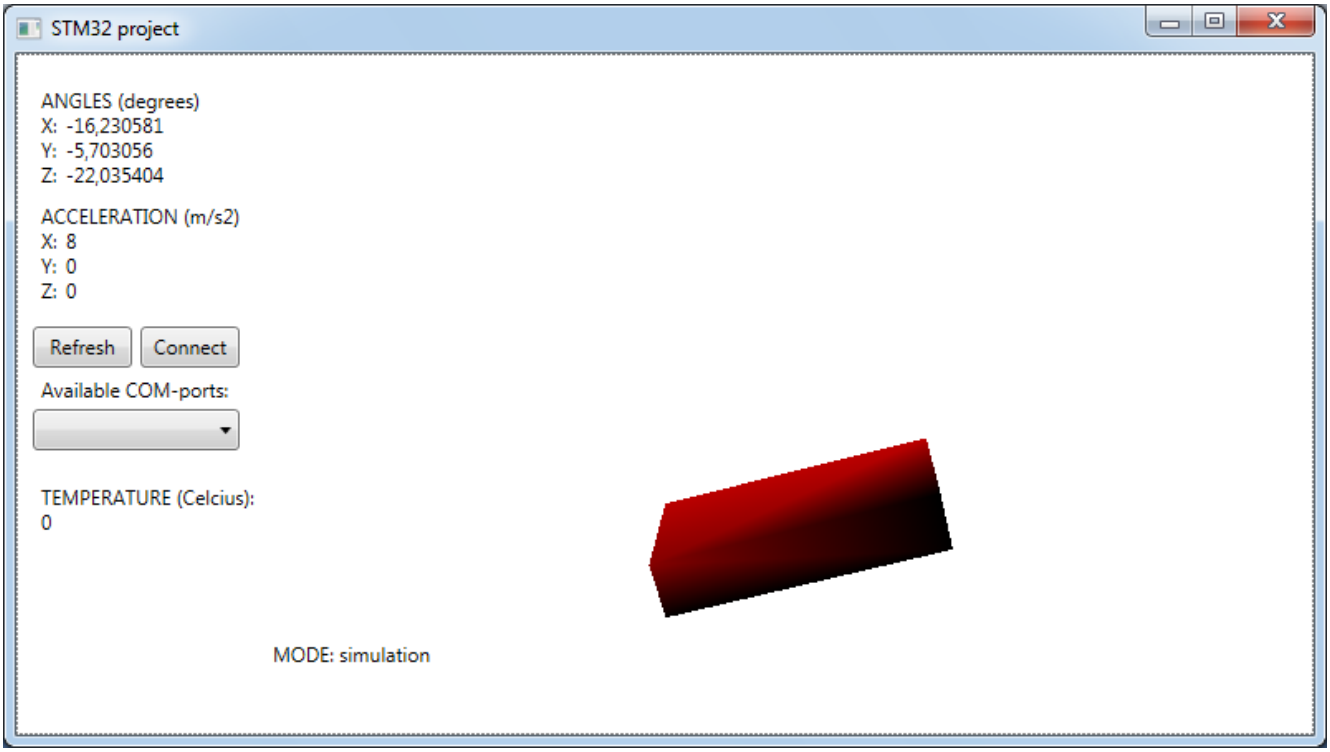


Рисунок 31 – Поворот трёхмерной модели с помощью корректировки ускорения

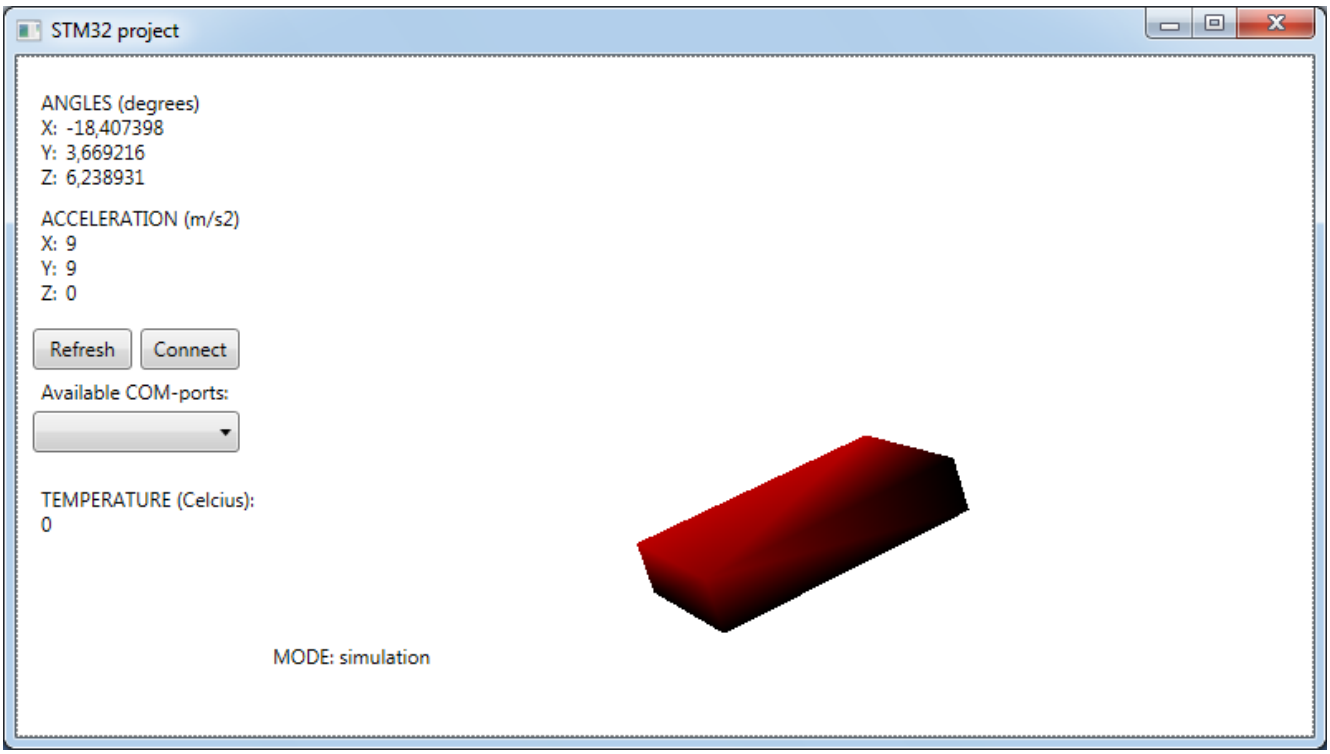


Рисунок 32 – Поворот трёхмерной модели с помощью корректировки ускорения

Процесс получения параметров через COM-порт представлен на рисунке 33.

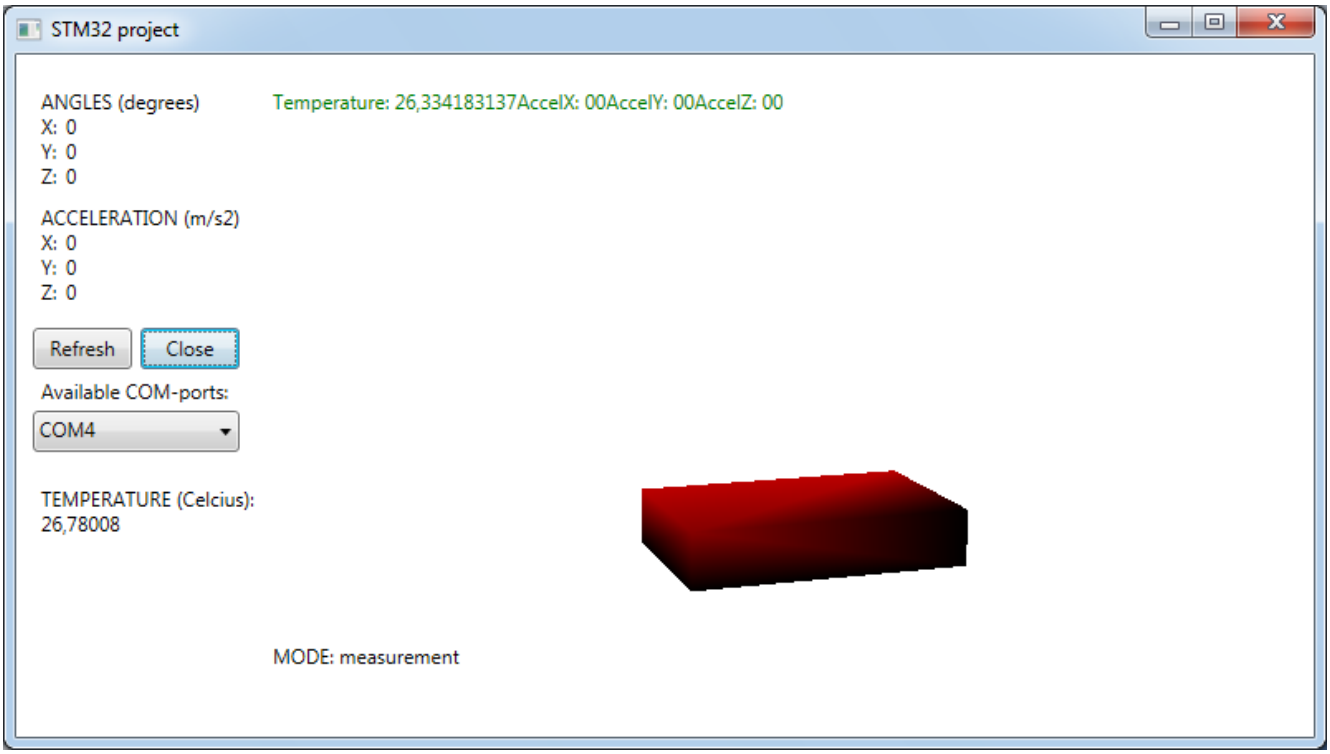


Рисунок 33 – Процесс получения параметров через COM-порт

На рисунке 34 представлено закрытие COM-порта.

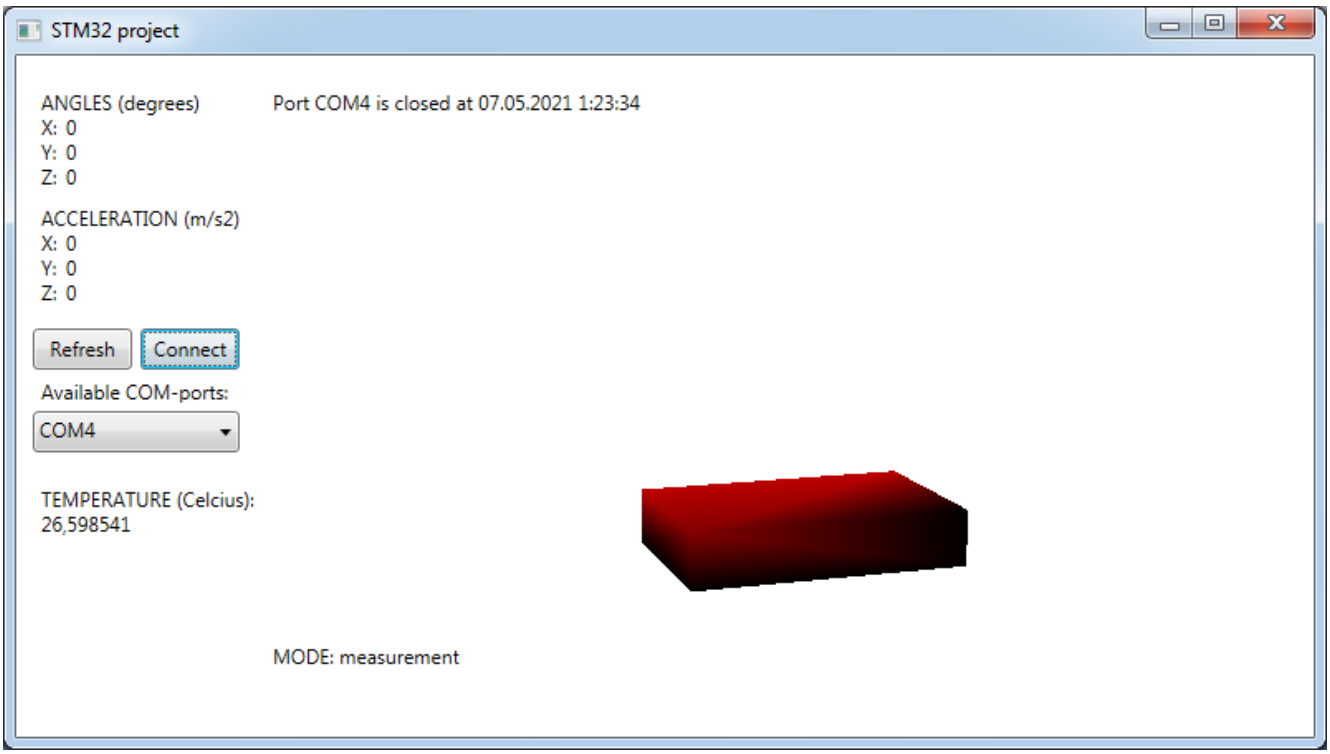


Рисунок 34 – Закрытие последовательного порта

При закрытии последовательного порта слева внизу высвечивается сообщение о том, что COM-порт закрыт.

ЗАКЛЮЧЕНИЕ

В данной работе было осуществлено измерение температуры с помощью встроенного температурного датчика STM32F411RE, измеренные данные были переданы на компьютер по UART, раскодированы и отображены на графическом интерфейсе.

Измерение температуры производилось каждые 50 мс с использованием АЦП и встроенного в микроконтроллер датчика температуры, измерение ускорения производилось каждые 50 мс с использованием I2C и трёхосевого акселерометра ADXL345, а передача данных – каждые 100 мс с использованием UART.

В приложении верхнего уровня распаковка данных производилась по мере поступления новых данных на COM-порт, обновление состояния визуальных объектов в окне графического интерфейса производилось каждые 100 мс.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. FreeRTOS Wrapper – https://github.com/lamer0k/RtosWrapper.

2. Пролижение верхнего уровня (Model3d-SerialPort-WPF) – https://github.com/alexeysp11/Model3d-SerialPort-WPF.

3. Приложение нижнего уровня (STM32F4-Accel-Temperature) – https://github.com/alexeysp11/STM32F4-Accel-Temperature.

4. https://en.wikipedia.org/wiki/Processor_(computing).

5. https://en.wikipedia.org/wiki/Microprocessor.

6. http://what-when-how.com/metrology/microprocessors-in-metrology/.

7. Orientation (geometry) – https://en.wikipedia.org/wiki/Orientation_(geometry).

8. Acceleration – https://en.wikipedia.org/wiki/Acceleration.

9. Net force – https://en.wikipedia.org/wiki/Net_force.

10. Newton's laws of motion – https://en.wikipedia.org/wiki/Newton%27s_laws_of_motion.

11. Angle of rotation – https://en.wikipedia.org/wiki/Angle_of_rotation.

12. https://en.wikipedia.org/wiki/I%C2%B2C.

13. Bit rate – https://en.wikipedia.org/wiki/Bit_rate.

14. UML – https://ru.wikipedia.org/wiki/UML.

15. System software – https://en.wikipedia.org/wiki/System_software.

16. Operating system – https://en.wikipedia.org/wiki/Operating_system.

17. Real-time operating system – https://en.wikipedia.org/wiki/Real-time_operating_system.

18. FreeRTOS. Real-time operating system for microcontrollers – https://www.freertos.org/.

19. FreeRTOS Github repository – https://github.com/FreeRTOS/FreeRTOS.

20. ГОСТ 2.702-2011 Единая система конструкторской документации (ЕСКД). Правила выполнения электрических схем.