**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Компьютерные системы и сети (ИУ6)

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***Система цифрового обмена блока чувствительных элементов гидролокационного комплекса***

Студент \_\_\_\_ИУ8-81\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Ю. Сидоров**

(Подпись, дата) (И.О.Фамилия)

Руководитель ВКРБ, **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Сотников**

начальник сектора НИИ ИСУ, к.т.н. (Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.И. Бауман**

(Подпись, дата) (И.О.Фамилия)

*2018 г.*

# АННОТАЦИЯ

В настоящей работе описывается процесс исследования, проектирования и внедрения системы цифрового обмена блока чувствительных элементов гидролокационного комплекса, предназначенной для стандартизированного обмена данными между блоком чувствительных элементов и блоком управления о ориентировании в пространстве – показания акселерометров и гироскопов.

Проанализированы вопросы структурной организации такой системы. Проведён анализ существующих технологий и приведено обоснование выбора конкретных инструментов для решения поставленных задач, показаны отдельные наиболее важные аспекты их использования в прототипе системы. Рассмотрены ключевые моменты проектирования и реализации системы. Осуществлена разработка технологии тестирования и приведены результаты её применения к разработанному прототипу.

Система должна собирать показания датчиков, обмениваться данными в реальном времени исходя из протокола информационного взаимодействия БЧЭ, а также визуализировать их для конечного пользователя.

# ABSTRACT

*The present work describes the process of research, design and implementation of a digital interchange system for a block of sensitive elements of a sonar complex intended for standardized data exchange between a block of sensitive elements and a control unit for orientation of space - readings of accelerometers and gyroscopes.*

*The questions of the structural organization of such a system are analyzed. The analysis of existing technologies is carried out and the substantiation of a choice of concrete tools for the decision of the put problems is resulted, the separate most important aspects of their use in a prototype of system are shown. The key moments of design and implementation of the system are considered. The development of testing technology is carried out and the results of its application to the developed prototype are shown.*

*The system must collect sensor readings, exchange data in real time based on the BSE information interaction protocol, and also visualize them for the end user.*

# РЕФЕРАТ

Записка 65 с., 11 табл., 28 рис., 26 ист.

БЛОК ЧУВСТВИТЕЛЬНЫХ ЭЛЕМЕНТОВ, БЛОК УПРАВЛЕНИЯ, ШИНА CAN, ПРОТОКОЛ CANOPEN, СТЕК ПРОТОКОЛА CANOPENNODE, STELLARIS, GY-521, ОБМЕН ДАННЫМИ

Объектом разработки является прототип системы цифрового обмена блока чувствительных элементов гидролокационного комплекса с блоком управления.

На сегодняшний день блоки чувствительных элементов используются повсеместно там, где важна скорость и надежность передачи данных. Поэтому разработка такой система имеет актуальный характер.

Цель работы — на основе анализа существующих решений определить структуру системы цифрового взаимодействия блока чувствительных элементов и разработать аппаратно-программные компоненты её прототипа.

В процессе работы проведён разбор существующих интерфейсов взаимодействия блоков чувствительных элементов, подбор компонентов системы, их адаптация друг ко другу и согласование интерфейсов, тестирование работоспособности созданного прототипа. Применение реально существующего протокола информационного взаимодействия блока чувствительного элемента и протокола высокого уровня для организации промышленных сетей позволило достичь актуальности и универсальности системы.

Предложенное решение может применяться как система ориентирования в пространстве в составе гидролокационного комплекса. Также, полученная система может быть без особых затруднений интегрирована в другие комплексы и системы, которые построены на основе протокола CANOpen.

СОДЕРЖАНИЕ

[ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ 7](#_Toc516665737)

[ВВЕДЕНИЕ 9](#_Toc516665738)

[1 Анализ предметной области и интерфейсов взаимодействия блоков чувствительных элементов 11](#_Toc516665739)

[1.1 Сравнительный анализ стандартов передачи данных 11](#_Toc516665740)

[1.1.1 Стандарт RS-485 11](#_Toc516665741)

[1.1.2 Стандарт CAN 12](#_Toc516665742)

[1.1.3 Сравнение Ethernet, RS-485, CAN и выбор оптимального решения 13](#_Toc516665743)

[1.2 Сравнительный анализ протоколов высокого уровня для промышленной сети CAN 16](#_Toc516665744)

[1.2.1 Описание протокола DeviceNet 17](#_Toc516665745)

[1.2.2 Описание протокола CANopen 18](#_Toc516665746)

[1.3 Определение особенностей реализации разрабатываемой системы 25](#_Toc516665747)

[1.3.1 Описание БЧЭ 25](#_Toc516665748)

[1.3.1.1 Общая структура БЧЭ 25](#_Toc516665749)

[1.3.1.2 Описание протокола взаимодействия БЧЭ с конечным пользователем 27](#_Toc516665750)

[1.3.2 Определение архитектуры системы 28](#_Toc516665751)

[1.3.3 Выбор аппаратных средств 29](#_Toc516665752)

[1.3.3.1 Выбор датчиков 29](#_Toc516665753)

[1.3.3.2 Выбор микроконтроллеров 30](#_Toc516665754)

[1.3.4 Выбор программных средств 32](#_Toc516665755)

[1.3.4.1 Выбор стека для реализации протокола CANopen 32](#_Toc516665756)

[1.3.4.2 Выбор среды и средств разработки программного обеспечения 33](#_Toc516665757)

[1.4 Постановка задач 33](#_Toc516665758)

[2 Проектирование и разработка системы 35](#_Toc516665759)

[2.1 Разработка функциональной схемы 35](#_Toc516665760)

[2.2 Разработка программного модуля реализации CanOpen 38](#_Toc516665761)

[2.3 Проектирование сети CANopen 40](#_Toc516665762)

[2.4 Разработка ПО микроконтроллеров 42](#_Toc516665763)

[2.5 Разработка диаграммы взаимодействия пользователя с системой 52](#_Toc516665764)

[2.6 Разработка диаграмм компоновки программного обеспечения 53](#_Toc516665765)

[2.7 Расчет скорости передачи данных 55](#_Toc516665766)

[3 Разработка технологии тестирования работоспособности системы 57](#_Toc516665767)

[3.1 Тестирование работоспособности компонентов системы 57](#_Toc516665768)

[3.2 Тестирование протокола передачи данных высокого уровня 60](#_Toc516665769)

[ЗАКЛЮЧЕНИЕ 62](#_Toc516665770)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 63](#_Toc516665771)

ПРИЛОЖЕНИЕ А. Техническое задание…………………………………..66

ПРИЛОЖЕНИЕ Б. Руководство пользователя…………………………….75

ПРИЛОЖЕНИЕ В. Исходные коды программ……………………………..84

ПРИЛОЖЕНИЕ Г. Копии листов графической части……………………120

# ОПРЕДЕЛЕНИЯ, СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ

БЧЭ – Блок чувствительных элементов;

БУ– Блок управления;

CAN – Controller Area Network - стандарт промышленной сети, ориентированный, прежде всего, на объединение в единую сеть различных исполнительных устройств и датчиков;

CANfestival – кросс-платформенный фреймворк CANopen с открытым исходным кодом;

CANopen – открытый сетевой протокол высокого уровня для подключения встраиваемых устройств в бортовых транспортных и промышленных сетях;

CANopenNode – кросс-платформенный фреймворк CANopen с открытым исходным кодом;

CRC - Cyclic redundancy check - алгоритм нахождения контрольной суммы, предназначенный для проверки целостности данных;

CSMA/CD CSMA/CD - Carrier Sense Multiple Access with Collision Detection - множественный доступ с прослушиванием несущей и обнаружением коллизий;

NMT – Объекты управления сетью;

OSI – Open Systems Interconnection basic reference model - Базовая Эталонная Модель Взаимодействия Открытых Систем;

PDO – Объекты данных процесса;

RS-485 –  Recommended Standard 485 – стандарт физического уровня для асинхронного интерфейса;

SDO – Сервисные объекты данных;

UART – Universal Asynchronous Receiver/Transmitter – Универсальный асинхронный приёмопередатчик;

МК – Микроконтроллер;

CiA – CAN in Automatization - международное сообщество пользователей и производителей решений на основе CAN;

СiA 301 - Спецификация прикладного уровня и коммуникационного профиля CANopen;

EVB – оценочная плата Stellaris LM3S8962 Evaluation Board;

SSI – Synchronous Serial Interface - синхронно-последовательный интерфейс;

JTAG – Joint Test Action Group - специализированный аппаратный интерфейс на базе стандарта IEEE 1149.1;

VCP – Виртуальный COM порт;

COB-ID – Идентификатор коммуникационного объекта CANopen;

ПО – Программное обеспечение

# ВВЕДЕНИЕ

Ни для кого не секрет, что мы живем в эпоху развития технологий. Каждый день изобретается что-то новое, совершенствуются старые разработки. При этом постепенно увеличивается техническая сложность изобретений человека. Автомобили, корабли, самолеты, ракеты – всё требует тщательного контроля протекания технических процессов для правильного функционирования. Для этого используются разнообразные датчики, различающиеся по методу измерения, измеряемым параметрам, принципу действия и т.д.

Но все датчики объединяет одно – информацию с них нужно передавать на управляющий блок. Для реализации данной задачи было изобретено много различных стандартов и интерфейсов передачи данных.

А что если предполагается использовать несколько датчиков и исполняемых устройств, служащих для выполнения комплексной задачи, в едином корпусе или, допустим, предполагается их использование в экстремальных условиях? В таком случае датчики и исполняемые устройства объединяют в единый модуль – блок чувствительных элементов (далее БЧЭ), который защищает встроенные датчики от условий окружающей среды и, можно сказать, объединяет их в некоторую локальную сеть [1].

Использование БЧЭ предъявляет особые требования к надежности передачи данных, ведь зачастую БЧЭ применяются там, где ошибки могут быть фатальны, например, в подводных лодках, самолетах, на атомной электростанции [2].

Также, в виду наличия большой функциональности, работой БЧЭ нужно управлять. Для выполнения этих целей используется блок управления (далее БУ). В качестве блока управления может выступать как персональный компьютер с наличием специального программного обеспечения, так и запрограммированный под конкретную задачу микроконтроллер.

Целью данной работы является разработка системы цифрового обмена БЧЭ гидролокационного комплекса, предназначенная для:

* контроля показаний датчиков;
* стандартизированного обмена информационными сообщениями (показания датчиков, сообщения об ошибках, сообщения состояния) между БЧЭ и БУ;

Для достижения поставленной цели необходимо решить следующие задачи:

* провести сравнительный анализ интерфейсов взаимодействия БЧЭ и на основе этого выбрать наиболее подходящий способ для обмена данными;
* на основе протокола информационного взаимодействия блока БЧЭ гидролокационного комплекса реализовать свой собственный протокол;
* провести обзор технических средств для моделирования системы;
* рассмотреть различные инструментальные средства разработки прикладных программ;
* разработать программное обеспечение для блока чувствительных элементов и блока управления;
* провести тестирование и отладку программного обеспечения компонентов системы;

# 1 Анализ предметной области и интерфейсов взаимодействия блоков чувствительных элементов

# 1.1 Сравнительный анализ стандартов передачи данных

Прежде чем приступать к разработке системы управления БЧЭ, необходимо рассмотреть все доступные интерфейсы взаимодействия и выбрать наиболее подходящий.

Есть несколько основных стандартов передачи данных, которые применяются для БЧЭ: RS-485 и CAN [2]. Т.к. БЧЭ используются в неблагоприятных условиях, то критически важным критерием при выборе стандарта данных следует уделить особое внимание обеспечению надежности передаваемых данных. Для этого рассмотрим представленные стандарты передачи данных подробнее.

# 1.1.1 Стандарт RS-485

Стандарт RS-485, разработанный в 1983 году, служит для создания многоабонентской связи между устройствами с интерфейсом UART. Устройство, подключенное к шине RS-485, обменивается данными с другим устройством в полудуплексном режиме. Несмотря на то, что полудуплексный режим связи предполагает двунаправленный обмен данными, в определенный момент времени только одно устройство может передавать данные, а другие устройства работают в режиме прослушивания (приема) [4].

RS-485 не является протоколом, он только определяет базовые правила и физический канал для обмена данными, позволяя передавать последовательные сообщения используя многоабонентскую шину, причем, содержимое сообщений полностью определяется пользователем. Сказанное также означает, что, например, структура коммуникационного кадра, схема адресации узлов, механизмы предотвращения коллизий, а также другие задачи, должны быть реализованы разработчиком в виде программной реализации протокола.

# 1.1.2 Стандарт CAN

Локальная сеть контроллеров (CAN) была разработана в 1980-х годах компанией Robert Bosch GmbH. Первоначально CAN была предназначена для автомобильной промышленности [3].

В настоящее время CAN используется во многих прикладных областях, таких как автоматизация промышленного производства, медицина, транспорт и т.д.

В отличие от RS-485, CAN определяет не только физическую среду связи, но и обеспечивает все необходимые механизмы для адресации пакетов данных (сообщений), предотвращения коллизий, обнаружения ошибок в переданных данных, автоматического повтора передачи поврежденных сообщений, обеспечения целостности данных во всех узлах сети. Также, CAN определяет структуру кадра данных с идентификатором сообщения, байтами данных и контрольными байтами. Как и в RS-485, узел CAN может взаимодействовать с любым другим узлом в полудуплексном режиме.

Сообщение, главным образом, включает идентификатор (ID), определяющий приоритет сообщения по отношению к предотвращению коллизий и до восьми байтов данных. Сообщение передается по шине последовательно и принимается всеми узлами. Если шина свободна, любой узел может начать передачу. В случае, когда одновременно два или более узлов начинают передавать сообщения, сообщение с более высоким приоритетом, который определяется наиболее значащим ID (имеет большее количество доминантных битов, т.е. нулей), перезапишет сообщения других узлов с менее значащими ID. Это гарантирует, что только значащие сообщение передано и принято всеми другими узлами. Данный механизм относится к методу арбитража шины, основанному на приоритете. Сообщения с численно меньшими значениями ID имеют более высокий приоритет и в случае одновременной передачи сообщений, передаются первыми.

Для принятия окончательного решения по выбору стандарта передачи данных, необходимо проанализировать их более подробно.

# 1.1.3 Сравнение Ethernet, RS-485, CAN и выбор оптимального решения

Обычно RS-485 работает посредством модуля UART. Обмен данными выполняется отдельными символами, которые записываются или читаются из 8-битового регистра. Следовательно, структура телеграммы должна быть реализована программно.

В отличие от RS-485, CAN оперирует со стандартизованным форматом телеграммы, который кроме 0-8 байтов данных, также содержит контрольную информацию для адресации и целостности данных (проверка CRC). Пользователь только определяет идентификатор сообщения и полезные данные, все остальное добавляется так называемым CAN контроллером. В качестве CAN контроллера может выступать как внешняя микросхема, так и внутренний модуль микроконтроллера.

Каждое CAN сообщение имеет собственный приоритет, определяемый идентификатором сообщения. Сообщения с более высоким приоритетом могут рассматриваться CAN контроллером как более предпочтительные для передачи и, соответственно, передаваться прежде чем сообщения с более низким приоритетом. Между узлами CAN шины, сообщения с более высоким приоритетом передаются первыми.

Следовательно, CAN является решением, удовлетворяющим условиям реального времени. Зная приоритет сообщения, который определяется его ID, независимо от загрузки шины или производительности других узлов можно определить максимальное время ожидания передачи для каждого сообщения.

В отличие от CAN, RS-485 не может детектировать никаких коллизий сообщений, а это значит, что предотвращение коллизий должно гарантироваться прикладным ПО. Обычно предотвращение коллизий в сетях RS-485 достигается при взаимодействии master/slave (ведущий/ведомый): ведущий узел последовательно опрашивает все ведомые узлы. Время ожидания передачи зависит от количества и времени реакции всех узлов, поэтому в RS-485 оно намного выше, чем в CAN.

Арбитраж на основе сообщений позволяет CAN работать в многоабонентском режиме без соблюдения дополнительных мер предосторожности. В случае RS-485, это только возможно с использованием специального протокола, такого как «token ring» («кольцо с маркерным доступом»), который должен быть реализован программно.

В CAN реализован усовершенствованный механизм управления обработкой ошибок. Если сообщение было получено узлом неправильно (ошибка CRC или формата фрейма), телеграмма «разрушается» получателем с помощью кадра ошибок и помечается как «недействительная» для всех узлов. Это действие инициирует автоматическую реакцию в CAN контроллере - повторить процесс передачи. Процедура гарантирует получение всеми узлами только «действительных» сообщений. Узлы, которые повторно передают некорректные данные или установили некорректность полученных данных, автоматически отключаются от взаимодействия в CAN шине. Таким образом, неисправный CAN контроллер или неисправные монтажные соединения не могут надолго нарушить передачу потока данных.

Все эти совместно выполняемые меры (короткие сообщения, дифференциальная передача, обнаружение ошибок и их устранение, отключение неисправных узлов) делают CAN отказоустойчивой, защищенной и надежной сетью.

Поэтому, CAN применяется во многих критических и отказоустойчивых системах транспортных средств, кораблей, лифтов, медицинских устройств, самолетов и промышленных заводов.

В таблице 1 приведена сравнительная характеристика описываемых стандартов [5].

Таблица 1 – Сравнительная характеристика стандартов CAN и RS-485.

|  |  |  |
| --- | --- | --- |
| **Функция** | **RS-485** | **CAN** |
| Интерфейс | UART | CAN-контроллер |
| Поддерживаемые уровни модели OSI | Физический уровень | Физический и канальный уровни |
| Обнаружение конфликтов данных | Нет | Да, CSMA/CD |
| Максимальная скорость передачи | 10 Мбит/сек (до 12 м) | 1 Мбит/сек (до 50 м) |
| Максимальная длина шины | 1200 м (при 100 Кбит/сек) | 1600 м (при 50 Кбит/сек) |
| Поддерживаемые правила арбитража шины | Master/Slave (ведущий/ ведомый) или «кольцо с маркерным доступом» | Multimaster (многомастер- ный) и все его производные, например, Master/Slave или «кольцо с маркерным доступом» |
| Максимальное количество данных в кадре | Не ограничено | 8 байт |
| Примеры распространенных протоколов | Modbus RS-485, Profibus | CANopen, DeviceNet |

Поскольку протокол CAN полностью реализован аппаратно, нагрузка на микроконтроллер значительно ниже (только одно прерывание на сообщение). В RS-485, обычно, одно прерывание на принятый символ, в дополнение к программной обработке протокола (включая механизмы обеспечения целостности данных), что приводит к существенной загрузке CPU. В зависимости от протокола и скорости передачи данных, а также требований приложения, для RS-485 требуются более мощные и дорогие CPU. Даже более низкие скорости передачи приводят к существенному влиянию на производительность CPU и его стоимость.

CAN контроллеры могут аппаратно фильтровать принимаемые сообщения по идентификатору сообщения. Таким образом, за счет приема только сообщений, обрабатываемых данным узлом, снижается загрузка CPU.

Также можно выделить и дополнительные преимущества стандарта CAN:

* если применяется CAN, интерфейс UART микроконтроллера остается свободным и его можно использовать для отладки или других вспомогательных целей (например, printf);
* вследствие высокой степени стандартизации CAN, PC/CAN интерфейсы (USB, PCIe, ...), включая программную поддержку (драйверы ПО), доступны по низким ценам. Следовательно, CAN легко доступен для приложений, работающих на ПК;
* для ПК доступны различные инструментальные средства анализа, моделирования, конфигурирования, обеспечивающие доступ от физического до прикладного уровней, что существенно упрощает процесс разработки и отладки;
* многие поставщики предлагают различные протоколы прикладного уровня, такие как CANopen, DeviceNet. Таким образом, разработчик может использовать надежное ПО прикладного уровня практически для всех доступных микроконтроллеров, что существенно позволяет снизить риск, ускорить продвижение на рынок и снизить затраты на разработку.

# 1.2 Сравнительный анализ протоколов высокого уровня для промышленной сети CAN

Так как протокол CAN существует уже довольно длительное время, то к настоящему времени было создано множество протоколов для организации промышленной сети.

Блоки чувствительных элементов, в целях обеспечения безопасности, зачастую основаны на собственных протоколах передачи данных, доступ к которым ограничен. Поэтому необходимо провести анализ существующих известных протоколов высокого уровня.

Самыми популярными на сегодняшний день являются протоколы DeviceNet и CANopen [12][13].

# 1.2.1 Описание протокола DeviceNet

DeviceNet предназначен для объединения промышленных устройств в единую сеть с общей шиной. DeviceNet - сетевое решение, которое предоставляет полноценную инфраструктуру для установления соединений между устройствами с поддержкой полной взаимозаменяемости устройств разных производителей [7][11].

DeviceNet – это закрытый стандарт, и для его использования производители обязаны приобретать лицензии для разработки DeviceNet совместимых устройств. Любой может приобрести спецификацию DeviceNet в ODVA. Любая компания может стать членом ODVA, и принимать участие в рабочей группе по дальнейшему усовершенствованию стандарта, но за это придется заплатить, что не вписывается в рамки реализуемого в дальнейшем проект [8].

Помимо этого, сложность реализации стандарта DeviceNet на устройства Stellaris (а именно с ними планируется работать в дальнейшем) нельзя адекватно оценить, поскольку сначала нужно приобрести доступ к данному стандарту.

Протокол CANopen лишен вышеуказанных недостатков, поэтому рассмотрим его подробнее.

# 1.2.2 Описание протокола CANopen

CANopen - это открытая промышленная сеть созданная на основе Controller Area Network (CAN). Стандарт CAN (ISO 11898) описывает два нижних уровня эталонной модели ISO/OSI, CANopen описывает остальные пять. Документ The CANopen Application Layer and Communication Profile (CiA DS 301) определяет каким образом устройства обмениваются данными и описывает интерфейс к нижележащим уровням сети [6].

Основная область применения СANopen - встроенные распределенные системы управления реального времени (embedded networks) [9]. СANopen фактически является стандартом и наиболее широко применяемым протоколом при создании современных систем управления в машиностроении (обрабатывающие станки различного назначения, термпопласт-автоматы, полиграфическое оборудование), железнодорожном транспорте (DIN 25002-2), специальном транспорте, сложном медицинском оборудовании, лифтах.

На Рисунке 1 показана функциональная схема связи двух узлов с помощью шины CAN и протокола CANopen.

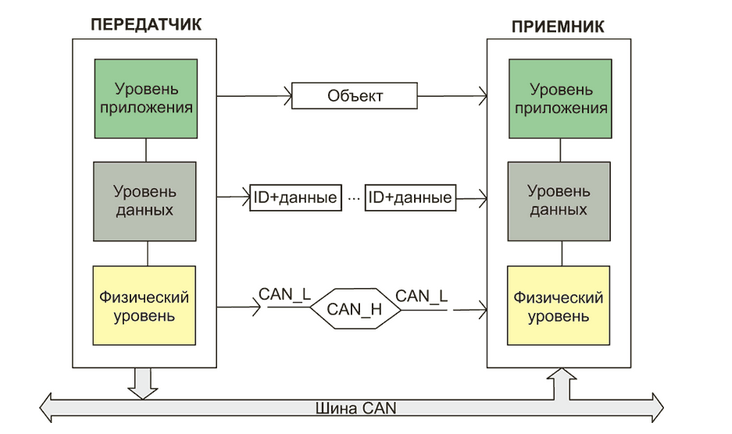


Рисунок 1 *-* Коммуникационные уровни при соединении двух узлов [10].

Основной функциональной единицей протокола является объект. Под объектом может пониматься набор данных, несущих информацию о параметрах (например, показания датчика температуры), конфигурации узла или сети, возникших ошибках и т.п. Каждый узел CANopen содержит центральную базу данных всех объектов – Словарь Объектов (Object Dictionary).

Функционирование CANopen полностью базируется на этом Словаре: там содержатся данные процесса, параметрические данные, а также вся необходимая информация для работы узла в сети. Приложение или микропрограмма узла взаимодействует со Словарём (записывает и считывает данные), а сервисы CANopen обеспечивают взаимодействие между Объектами Словаря разных узлов.

По своей сути, Словарь Объектов – это связующее звено между приложением и передаваемой на физическом уровне информацией (см. Рисунок 2). С каждым устройством, использующим интерфейс CANopen, производитель должен предоставить файл с расширением \*.eds (Electronic DataSheet), содержащий словарь объектов и дополнительную информацию.

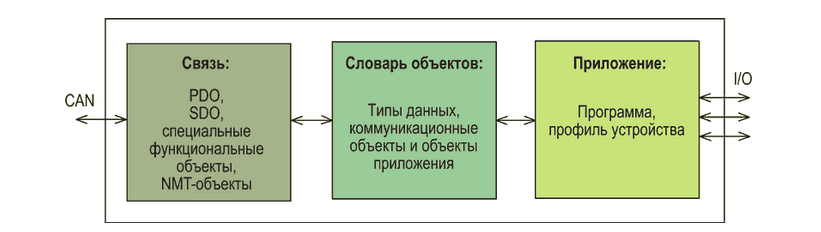


Рисунок 2 – Модель устройства с интерфейсом CANopen [10].

CANopen-устройство имеет три условных составляющих:

* программный модуль обработки протокола и пакетов интерфейса;
* словарь объектов;
* программное обеспечение на уровне приложения.

Модуль обработки протокола непосредственно отвечает за передачу и прием коммуникационных объектов по шине. Словарь объектов описывает все типы данных, коммуникационные объекты и объекты приложения, используемые в данном устройстве. Программное обеспечение на уровне приложения выполняет функции внутреннего управления и обеспечивает связь с другими устройствами, не использующими шину CAN.

Каждый объект в словаре имеет 16-разрядный индекс и 8-разрядный подиндекс.

Протокол CANopen предполагает существование следующих типов объектов:

* сервисные объекты данных (SDO);
* объекты данных процесса (PDO);
* специальные функциональные объекты: объект синхронизации (SYNC), временная метка, срочное сообщение (EMCY);
* сообщения управления сетью, например Layer Management (LMT) и Network Management (NMT) сообщения.

SDO обеспечивают доступ к элементам словаря объектов, то есть, с помощью SDO можно читать и записывать информацию в словарь объектов. При связи посредством SDO между двумя устройствами устанавливается соединение типа «точка-точка», причем будет реализована клиент-серверная модель коммуникации, где устройство-клиент передает необходимое сообщение, а устройство-сервер выдает соответствующий этому сообщению ответ, в соответствии со схемой, изображенной на Рисунке 3.

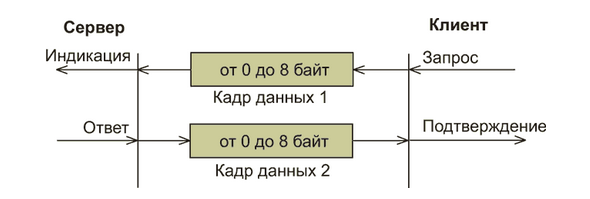


Рисунок 3 – Клиент-серверная модель [6].

При организации SDO-связи клиент с помощью мультиплексора, содержащего индекс и подиндекс объектного словаря, может предоставлять серверу определенный набор данных, которые будут передаваться. В основном передача SDO реализуется как сегмент-трансферт (segment transfer), при котором передается последовательность сегментов в количестве до 127.

Для непосредственной передачи полезных данных технологического процесса (температура, скорость, ток, напряжение и т.п.) в реальном времени используются PDO. Передача PDO осуществляется широковещательно, при этом применяется модель производитель-потребитель (producer-consumer), изображенная на Рисунке 4.

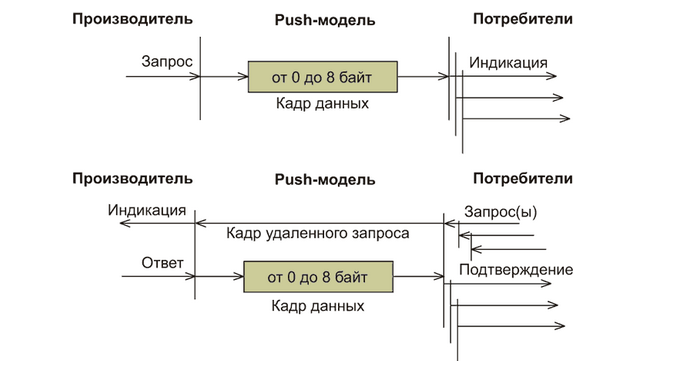


Рисунок 4 – Модель производитель – потребитель [6].

В объектном словаре различаются два типа PDO – для передачи данных (TPDO) и для приема (RPDO). Устройства, в конкретный момент времени выдающие PDO на шину, называются производителями, а принимающие эти PDO – потребителями. PDO также описываются в объектном словаре устройства. Типы данных и отображение объектов в PDO описываются структурой, называемой PDO-отображение (PDO-mapping). С помощью SDO на стадии инициализации можно изменить количество PDO и отображение объектов внутри них. Все PDO описываются структурным параметром (или параметром отображения) и параметром связи, характеризующим коммуникационные возможности PDO. Индексы этих параметров определяются в соответствии c спецификацией прикладного уровня и коммуникационного профиля CANopen (СiA 301) [17].

С помощью одного PDO можно передать от 1 до 8 байт данных. В одной сети CANopen может присутствовать до 512 TPDO и до 512 RPDO.

PDO могут передаваться как синхронно, так и асинхронно относительно объекта синхронизации SYNC, выдающегося на шину с определенной периодичностью. Это проиллюстрировано Рисунком 5. Синхронные PDO передаются в рамках установленного интервала времени после появления на шине SYNC-объекта.

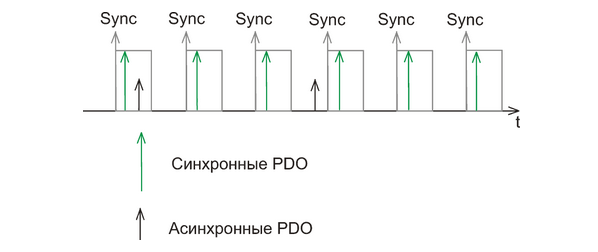


Рисунок 5 – Принцип передачи синхронных и асинхронных PDO [6].

Асинхронные PDO передаются без какой-либо связи с объектом синхронизации. Различают также три режима вызова PDO (Рисунок 6):

* по событию или по таймеру: механизм передачи PDO запускается после возникновения какого-либо внутреннего события или по срабатыванию таймера устройства;
* по удаленному запросу: в этом случае устройство начинает передачу PDO после получения кадра удаленного запроса от другого устройства на шине;
* синхронная передача (цикличная или ацикличная): как уже отмечалось ранее, связана с появлением на шине SYNC-объекта.

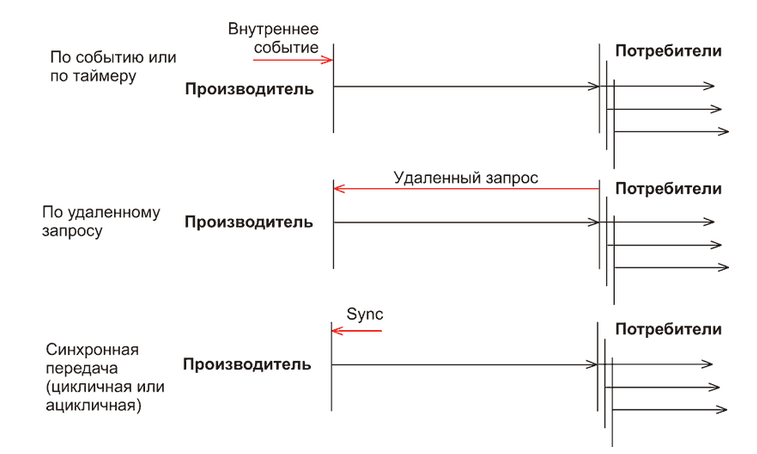


Рисунок 6 – Три режима вызова PDO [6].

Другими объектами, без которых немыслимо существование CANopen-сети, являются NMT-объекты, позволяющие управлять работой этой сети. Вначале стоит отметить, что в конкретный момент времени устройство должно находиться в одном из четырех состояний (см. Рисунок 7): инициализация (Initialization), готовность (Pre-operational), работа (Operational) или остановка (Stopped).

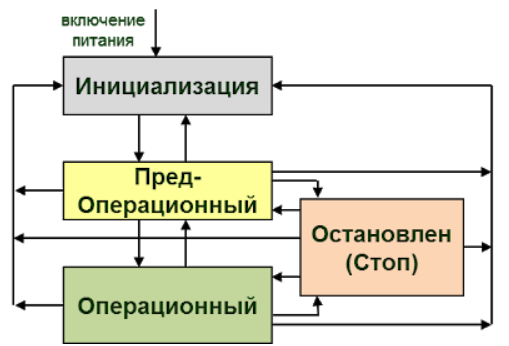


Рисунок 7 – Возможные состояния узла сети CANopen [6]

При включении устройство проходит этап внутренней инициализации, и после ее успешного завершения переходит в состояние готовности. В этом состоянии уже возможно осуществить настройку CANopen-узла с помощью SDO. Затем узел может перейти в рабочее состояние. Для этого необходимо, чтобы Master сети (передача NMT сообщений происходит в соответствии с моделью Master-Slave) передал широковещательное сообщение Start\_remote\_node. ID NMT-сообщений равен 0, поскольку они должны иметь самый высокий приоритет в сети. В Таблице 2 представлено описание NMT-сообщений.

Таблица 2 – Описание NMT-сообщений [10]

|  |  |  |
| --- | --- | --- |
| Сообщение | Обозначение | Команда в составе формата сообщения |
| Запуск удаленного узла | Start\_remote\_node | 1 |
| Остановка удаленного узла | Stop\_remote\_node | 2 |
| Вход в состояние готовности | Enter\_pre\_operational | 128 |
| Сброс узла | Reset\_node | 129 |
| Сброс связи | Reset\_communication | 130 |

Формат NMT-сообщения также предполагает наличие ID того узла, которому адресовано сообщение. В случае равенства этого параметра 0 сообщение будет адресовано всем узлам сети, то есть, например, формат 0/1/0 будет означать запуск всех узлов на шине, а 0/2/9 – остановку узла с ID=9.

Для повышения надежности функционирования сети имеются объекты срочных сообщений (Emergency Object или EMCY). Их передача осуществляется при возникновении внутренних ошибок какого-либо узла. Срочное сообщение передается в сеть лишь один раз после возникновения определенной ошибки, и, как бы долго состояние активной ошибки не присутствовало, нового соответствующего ей EMCY передано не будет. Только при возникновении новых ошибок могут быть переданы соответствующие им EMCY. Механизм передачи срочных сообщений не является обязательным для сети CANopen, но при грамотном его применении он позволит вовремя определить и устранить неисправность узла.

Все Slave-устройства в составе сети CANopen могут посылать специальное сообщение о своей готовности функционирования в сети. Это сообщение начальной загрузки (boot-up message) дает понять Master-устройству, что внутренний сетевой статус Slave-узла перешел из режима инициализации в режим готовности к работе. Передача boot-up-сообщения является также необязательной, но желательной процедурой, поскольку Master будет знать, что конкретное Slave-устройство уже можно настраивать с помощью SDO или переводить его в режим работы.

Интерфейсом CANopen предусмотрены два протокола контроля функционирования сети: протокол караула узлов (Node guarding protocol) и протокол контрольного тактирования (Heartbeat protocol). В первом случае выделенный NMT-мастер опрашивает Slave-устройства через одинаковые промежутки времени, называемые guard time. В ответ каждое Slave-устройство посылает сообщение, содержащее его сетевой статус. Время ожидания подобного сообщения может быть настроено индивидуально для каждого узла. Если узел по истечении заданного времени не получил запрос от Master-устройства, на его стороне с помощью сервиса Life Guarding Event возникнет ошибка, свидетельствующая об отсутствии сторожевого запроса. Если удаленный запрос передачи не был подтвержден за время сторожевого ожидания, или указанный в ответном сообщении статус Slave-устройства не соответствует ожидаемому, со стороны Master-устройства возникнет ошибка караула узла, сообщаемая с помощью сервиса Node guarding event.

Heartbeat-протокол позволяет контролировать функционирование сети без необходимости посылки Slave-устройствами удаленных ответов. В данном случае узел, сконфигурированный на широковещательную передачу Heartbeat-сообщения, является производителем контрольных тактов. Остальные устройства, настроенные на прием Heartbeat-сообщения, являются потребителями контрольных тактов, и в случае если за время ожидания контрольного такта (Heartbeat Consumer Time) Heartbeat-сообщение не пришло, генерируется ошибка контрольного тактирования. Оба рассмотренных протокола контроля функционирования сети являются взаимоисключающими, то есть в сети можно использовать лишь один из них. Heartbeat-протокол имеет более высокий приоритет, и по умолчанию предполагается использование именно его.

# Определение особенностей реализации разрабатываемой системы

# 1.3.1 Описание БЧЭ

# 1.3.1.1 Общая структура БЧЭ

Блок чувствительных элементов – это электромеханическое устройство, выполняющее измерения, преобразования и выдачу потребителю полученной информации о действующих на изделие угловых скоростях и кажущихся линейных ускорениях.

БЧЭ, на основании которого основана разрабатываемая система, используется в гидролокационных системах, например, в торпедах подводных лодок для управления траекторией движения.

Основными измерительными элементами БЧЭ являются следующие датчики:

* три одноосных гироскопа (для осей Ох, Оу, Оz) соответственно;
* три одноосных акселерометра (для осей Ох, Оу, Оz) соответственно;
* три сенсора температуры.

Упрощенная структурная схема БЧЭ представлена на рисунке 8.



Рисунок 8 – Упрощенная структурная схема БЧЭ

T1, T2, T3 – сенсоры температуры.

БА – Блок Акселерометров, состоит из трех одноосевых акселерометров с диапазоном измеряемых ускорений ± 16 g.

БГ – Блок Гироскопов, состоит из трех одноосевых гироскопов с диапазоном угловых скоростей 180-300 °/с.

ППС – Плата Преобразования Сигналов, представляет собой микроконтроллер. Собирает информацию с датчиков, выполняет требуемые функциональные вычисления и обеспечивает связь с конечным потребителем по последовательному каналу передачи данных CAN в соответствии с протоколом информационного взаимодействия БЧЭ с потребителем (описан ниже).

# 1.3.1.2 Описание протокола взаимодействия БЧЭ с конечным пользователем

Обмен между БЧЭ и потребителем осуществляется по каналу CAN стандартными пакетами данных с 11-разрядным полем арбитража со скоростью передачи 1 Мбит/с. БЧЭ посылает пакет ответа по каналу только в том случае, если предварительно был получен запрос на данный пакет. После ответа на запрос БЧЭ ожидает следующий запрос и т.д.

Состав пакетов «Запрос» и «Ответ» приведён в таблице 3

Таблица 3 – Состав пакетов «Запрос» и «Ответ»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Номер  флага | Запрос | | Ответ | | | Параметр | Обозначение |
| Поле данных  (hex) | DLC | Поле арбитража | | DLC |
| 28-18 разряды  (hex) | RTR |
| 0 | 1 | 8 | 22 | 0 | 8 | Кватернион | λ0, λX, λY, λZ |
| 1 | 2 | 8 | 2 | 0 | 8 | Угловая скорость | ωX, ωY, ωZ |
| 2 | 4 | 8 | 42 | 0 | 8 | Интеграл линейного ускорения | vX, vY, vZ |
| 3 | 8 | 8 | С2 | 0 | 4 | Слово состояния БЧЭ | − |
| 4 | 10 | 8 | A2 | 0 | 8 | Линейное ускорение | aX, aY, aZ |
| 5 | 20 | 8 | 422 | 0 | 8 | Температура | Т1, Т2, Т3 |
| 6 | 40 | 8 | 82 | 0 | 8 | Интеграл линейного ускорения | VX, VY |
| 62 | 0 | 4 | Vz |
| 7 | 80 | 8 | 402 | 0 | 4 | Резерв | − |
| 8 | 100 | 8 | − | − | − | Reset | − |
| 9 | 200 | 8 | − | − | − | Сброс λ | − |
| 10 | 400 | 8 | − | − | − | Выключение | − |

Так как предполагается использовать штатный режим работы БЧЭ, то будут задействованы только флаги 0, 1, 4, 5, 8.

Структура пакета данных приведена в таблице 4.

Таблица 4 – Структура пакета данных

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SOF | Поле арбитража  (Arbitration field) | | | | | | Поле управления  (Control field) | | | | | | Поле данных  (Data field) | | | | | | CRC field | | |
| Standart ID | | | | | | IDE | R0 | DLC | | | | Байт 0 | | | Байт 1 | … | Байт 7 | 14 бит | … | 0 бит |
| 28 бит | 27 бит | … | 19 бит | 18 бит | RTR | 3 бит | 2 бит | 1 бит | 0 бит | 7 бит | … | 0 бит |

# 1.3.2 Определение архитектуры системы

Поскольку блок чувствительных элементов – дорогостоящее (порядка 1.5 – 2 млн. рублей) и доступ к работе с ними физическим лицам невозможен [2], было решено имитировать БЧЭ и БУ с помощью микроконтроллеров.

Исходя из вышеизложенного анализа БЧЭ, необходим мониторинг показаний трех типов параметров:

* линейные ускорения по осям координат Ox, Oy и Oz (ωx, ωy, ωz);
* угловые скорости по осям координат Ox, Oy и Oz (Ax, Ay, Az);
* температура.

Для измерения линейных ускорений необходимо выбрать один или несколько датчиков акселерометров, для измерения угловых скоростей – один или несколько датчиков гироскопов.

Данные с датчиков должны передаваться на микроконтроллер, который в дальнейшем будет предоставлять информацию конечному пользователю.

Микроконтроллеры, имитирующие работу БЧЭ и БУ, должны быть оснащены приемо-передатчиком CAN для обмена данными. Также необходимо реализовать работу протокола CANopen на основе протокола информационного взаимодействия БЧЭ с пользователем (см п. 1.3.1.2).

Блок управления должен формировать запросы к БЧЭ, поэтому нужно предусмотреть наличие элементов управления.

Для удобства отладки программного обеспечения и графического представления данных конечному пользователю необходимо передавать отладочные данные на персональный компьютер.

На основе вышеизложенных доводов была представлена структурная схема разрабатываемой системы:

Рисунок 9 – Структурная схема системы

Пользователь может формировать управляющие запросы к БЧЭ посредством элементов управления БУ, а также просматривать отладочную информацию по сети CANopen и ее элементам, используя персональный компьютер.

# 1.3.3 Выбор аппаратных средств

После анализа предметной области и получения требований к оборудованию осталось выбрать элементную базу. Под элементной базой подразумевается микроконтроллеры и датчики.

# 1.3.3.1 Выбор датчиков

Проанализировав требования к необходимым датчикам и ассортимент радиомагазинов, выбор остановился на модуле GY-521 [14]. В состав данного модуля входит микросхема MPU6050, которая оснащена трёхосевыми акселерометром и гироскопом, а также сенсором температуры. Очень удобно, что все датчики, требуемые для реализации разрабатываемой системы, собраны в единый модуль, который, ко всему прочему, не имеет аналогов по цене (порядка 200 рублей).

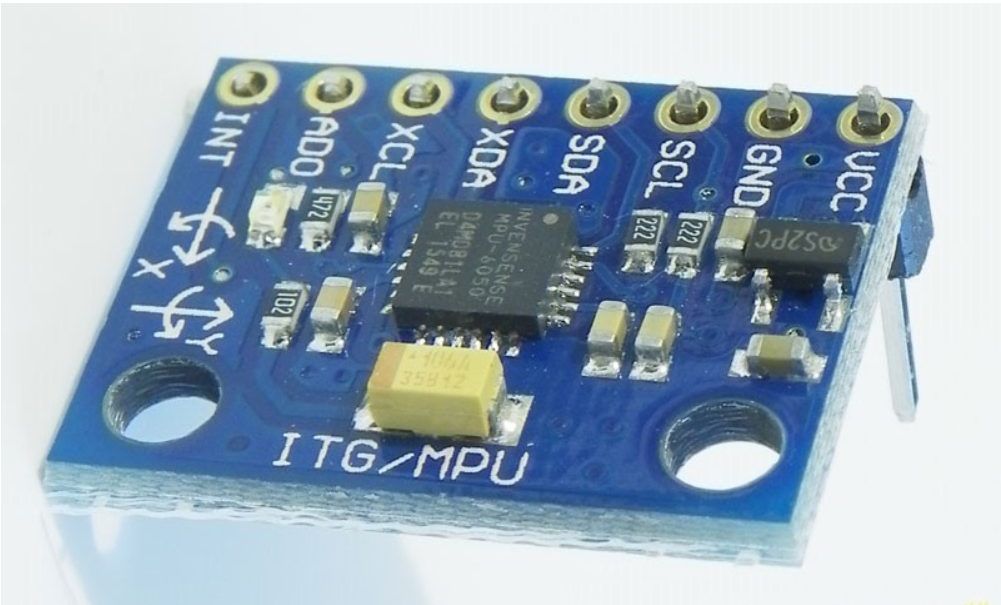


Рисунок 10 – Модуль GY-521

Модуль GY-521 включает понижающий линейный стабилизатор для питания от 5В, красный светодиод питания и обвязку, обеспечивающую работу по протоколу I2C.

Характеристики MPU6050:

* 16-битный АЦП;
* напряжение питания 3-5В;
* поддержка протокола «I2C»;
* диапазон ускорений: ± 2 ± 4 ± 8 ± 16g;
* диапазон «гиро»: ± 250 500 1000 2000 °/s.

Таким образом, один единственный модуль сочетает в себе все необходимые датчики, которые удовлетворяют требованиям технического задания.

# 1.3.3.2 Выбор микроконтроллеров

Так как предполагается реализовывать протокол CANopen, то микроконтроллеры для реализации БЧЭ и БУ, должны обладать интерфейсом CAN и достаточно высокой производительностью. Помимо этого, для подключения модуля GY-521, микроконтроллер для реализации БЧЭ должен поддерживать передачу данных по интерфейсу I2C.

Необходимым требованием также будет являться наличие или возможность подключения дисплея для визуализации данных.

Исходя из всех перечисленных требований, а также из-за наличия опыта работы (в ходе курсовых и лабораторных работ) выбор пал на микроконтроллеры семейства Stellaris компании Texas Instruments, а именно, на оценочные платы LM3S8962.

Микроконтроллер обладает следующими важными особенностями [15]:

* 32-разрядная RISC архитектура, основанная на ARM CortexTM-M3 v7M. Работа осуществляется на частоте до 50 МГц;
* Блок защиты памяти (MPU);
* Интегрированный контроллер вложенных векторных прерываний (NVIC);
* 42 канала прерываний с 8 уровнями приоритетов;
* 256 Кбайт встроенной программируемой flash-памяти с временем выборки – один цикл тактовой частоты;
* SRAM объемом 64 кБайт;
* 32-разрядных таймера общего назначения;
* Интегрированный модуль Ethernet;
* Модуль CAN-Интерфейса;
* Полностью программируемые 16C550-совместимые UART;
* До 42 линий GPIO, в зависимости от пользовательской конфигурации;
* Встроенный стабилизатор с низким падением напряжения;
* 512 байт встроенное ОЗУ.
* 8 – битный таймер с предделителем входной частоты.
* 16 – разрядный таймер с предделителем входной частоты, функцией сравнения, захвата и режимом ШИМ.
* Энергопотребление на частоте 4 МГц при питающем напряжении 3 В составляет 3 мА.
* 32 программируемых линии ввода-вывода (четыре 8-битных порта).
* Диапазон питающего напряжения 2,7 ÷ 6 В.

Расположение основных модулей на оценочной плате приведено на рисунке 11.

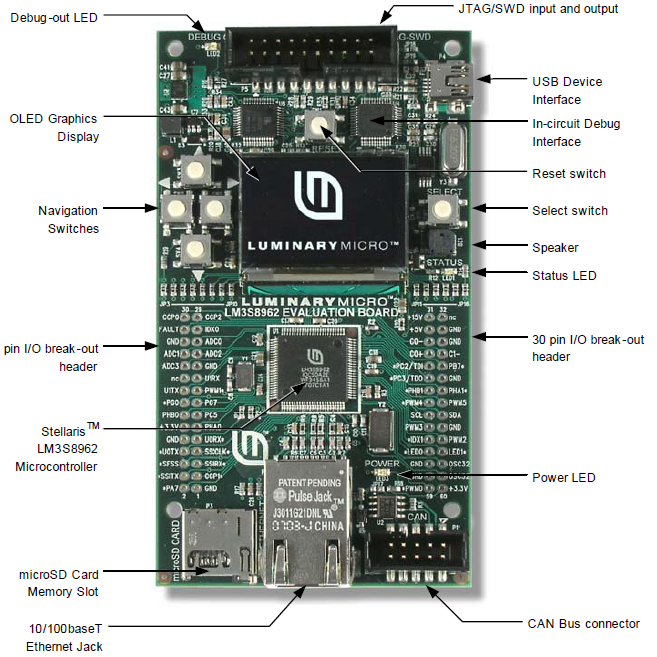


Рисунок 11 – Расположение модулей на оценочной плате Stellaris LM3S8962 Evaluation Board [15].

# 1.3.4 Выбор программных средств

# 1.3.4.1 Выбор стека для реализации протокола CANopen

Написание полного стека CANopen с нуля может вызвать определенные затруднения и потребовать неприемлемых в некоторых случаях затрат времени, поэтому сегодня уже существует немалое количество библиотек для различных платформ, позволяющих адаптировать конкретное устройство для работы в сети CANopen. На сегодняшний день существует множество фреймворков CANopen, однако самыми популярными из них являются CANfestival и CANopenNode.

Оба варианта находятся в свободном доступе и легко портируются на различные микроконтроллеры. Однако, по стеку CANopenNode в различных источниках больше информации по настройке и использованию, и на форуме компании Texas Instruments рекомендательно советуют использовать стек CANopenNode для микроконтроллеров семейства Stellaris, с которыми и планируется работать.

Помимо этого, CANfestival изначально разрабатывался для 8-ми и 16-ти разрядных микроконтроллеров, а микроконтроллеры семейства Stellaris построены на 32-х разрядной архитектуре ARM, что вызовет дополнительные сложности в портировании.

Таким образом, использование стека CANopenNode для реализации протокола CANopen является наиболее предпочтительным.

# 1.3.4.2 Выбор среды и средств разработки программного обеспечения

Для разработки программ для оценочных плат Texas Instruments пользуются спросом среды разработки CodeComposer Studio и uVision Keil, причем первая рекомендована компанией TI [15]. Также для микроконтроллеров семейства Stellaris для облегчения программирования поддерживается микропрограммное обеспечение StellarisWare, представляющее собой набор различных подключаемых классов-функций для облеserгчения программирования типовых задач. Проанализировав техническое задание, было решено использовать среду разработки CodeComposer Studio и микропрограммное обеспечение StellarisWare, Поскольку задачи, поставленные в рамках дипломной работы, попадают под типовые для StellarisWare, вследствие чего не нужно будет работать напрямую с регистрами, это сильно ускорит процесс написания кода.

# 1.4 Постановка задач

На основании вышеизложенного можно сделать вывод, что разработка системы цифрового обмена блока чувствительных элементов является актуальной. Реализуемый протокол для стека CANopen может быть широко востребованным для потребителей, заинтересованных в организации промышленной сети.

Для эффективного выполнения системой своих функций она должна решать следующие задачи:

* формировать запрос к БЧЭ на выдачу требуемых данных;
* управлять работой БЧЭ посредством БУ;
* контролировать показания датчиков;
* выводить полученные данные на дисплей, а также выводить отладочную информацию на персональный компьютер пользователя и отвечать следующим требованиям:

1. информационный обмен данными системы осуществляется на основе протокола информационного взаимодействия БЧЭ с пользователем;
2. система должна быть построена на основе протокола CANopen;
3. датчики должны удовлетворять требованиям, описанным в п. 1.3.1.1.

Для реализации системы цифрового обмена блока чувствительных элементов гидролокационного комплекса необходимо решить следующие задачи:

* провести обзор стандартов передачи данных, используемых в БЧЭ;
* провести обзор технических средств разработки управления БПЛА и рассмотреть различные инструментальные средства разработки прикладных программ;
* провести анализ элементной базы;
* реализовать протокол CANopen для микроконтроллеров Stellaris;
* разработать структурную, функциональную и принципиальную схемы разрабатываемой системы;
* разработать модуль блока управления;
* разработать модуль блока чувствительных элементов;
* объединить модули в единую систему (CANopen), провести тестирование и отладку программ.

# 2 Проектирование и разработка системы

# 2.1 Разработка функциональной схемы

На основе ТЗ и результатов анализа требований к разрабатываемой системе была разработана функциональная схема МК-системы (Приложение Б).

На функциональной схеме МК-системы можно выделить два блока:

* блок чувствительных элементов;
* блок управления.

Передача данных между БЧЭ и БУ осуществляется с помощью приемо-передатчиков CAN со скоростью 1 Мбит/сек, в соответствии с протоколом информационного взаимодействия БЧЭ с пользователем. На оценочных платах LM3S8962 Evaluation Board имеется стандартный CAN приемопередатчик и 10-контактный разъем CAN, который наиболее часто используется для подключения к CAN сети.

Встроенный терминальный резистор сопротивлением 120 Ом обеспечивает заглушку для шины.

В состав БЧЭ входит модуль GY-521, который собирает данные с датчиков (гироскоп, акселерометр, сенсор температуры), преобразует их через встроенный аналого-цифровой преобразователь (АЦП) и записывает их в сенсорный регистр. Затем обработанные данные поступают на последовательный интерфейс I2C.

Оценочная плата Stellaris EKS-LM3S8962 имеет в своем составе OLED-дисплей семейства RIT P14201, который будет задействован для визуализации данных и работы компонентов системы в целом. Данный дисплей обладает несколькими важными особенностями:

* разрешение 128x96 точек;
* высокая контрастность (типовое значение 500:1);
* высокая яркость (120 кд/м2);
* время отклика ниже 10 мкс.

Дисплей RIT P14201 имеет встроенную интегральную схему контроллера с последовательным и параллельным интерфейсами. На оценочной плате используется синхронный последовательный интерфейс (SSI), так как платы EVB разведены производителем так, что вход E/RD подтянут к земле. При этом, если не требуется считывать данные из контроллера дисплея, то необходима всего одна линия данных.

Пользовательские кнопки (Up, Down, Right, Left, Select) используются для навигации по меню (Up, Down), выводимому на дисплей, выбора значений для формирования запросов для БЧЭ (Select), а также для вывода отладочной информации в порт UART0.

В отладочных платах LM3S8962 реализован виртуальный COM порт (VCP), который позволяет приложениям Windows (таким как HyperTerminal) обмениваться данными с блоком UART0 микроконтроллера LM3S8962 через USB интерфейс.

Следует отметить, что у данного микроконтроллера есть встроенный интерфейс отладки (ICDI), изображенный на рисунке 5. В его составе есть USB-интерфейс, который позволяет перепрограммировать микроконтроллер без использования специального программатора.

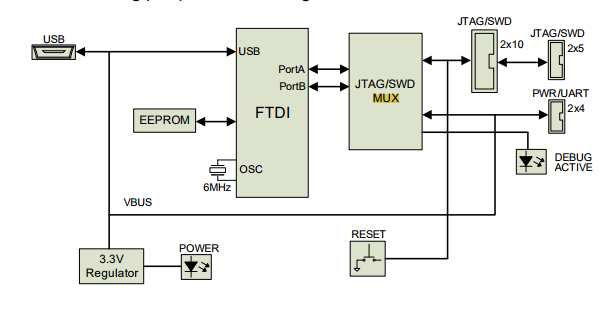


Рисунок 12 – Встроенный интерфейс отладки

Микроконтроллер поддерживает программирование и отладку с использованием USB, JTAG, SWD. JTAG использует сигналы TCK, TMS, TDI и TDO. Для SWD требуется меньше сигналов: SWCLK, SWDIO. Отладчик знает, какой из протоколов отладки будет использоваться. Так как в комплекте к оценочной плате идет 20-пиновый шлейф-соединитель, то отладка и программирование будет происходить по JTAG.

Оценочная плата поддерживает несколько конфигураций аппаратных средств отладки. В таблице 5 показаны эти конфигурации.

Таблица 5 - Конфигурация аппаратных средств отладки ознакомительной платы Stellaris LM3S8962

|  |  |  |  |
| --- | --- | --- | --- |
| **Режим** | **Функция отладки** | **Используется** | **Выбирается посредством…** |
| 1 | Внутренний ICDI | Отладка встроенного  микроконтроллера  LM3S8962 через USB  интерфейс. | Режим по умолчанию |
| 2 | ICDI-выход на разъеме JTAG/SWD | EVB используется как  преобразователь USB в SWD/JTAG для внешнего  микроконтроллера. | Подключения внешнего  отлаживаемого модуля и  запуска отладочного  программного обеспечения. |

Продолжение таблицы 5

|  |  |  |  |
| --- | --- | --- | --- |
| **Режим** | **Функция отладки** | **Используется** | **Выбирается посредством…** |
| 3 | Вход разъема JTAF/SWD | Для пользователей,  которые предпочитают  внешний отладочный  интерфейс (ULINK, JLINK,  etc.) для EVB | Подключения внешнего  отладчика к разъему  JTAG/SWD. |

# 2.2 Разработка программного модуля реализации CanOpen

Поскольку готового порта стека CANopenNode для микроконтроллеров семейства Stellaris на данный момент не существует, первым делом было решено портировать стек на основе существующих решений. За основу был взят порт для 32-х разрядных микроконтроллеров PIC32, поскольку в основе этих микроконтроллеров лежит ядро Cortex-M3, как и в микроконтроллерах Stellaris.

В состав библиотеки CANopenNode для любого микроконтроллера входят заголовочные и исполняемые файлы, описанные в таблице 6 [16].

Таблица 6 – Состав библиотеки CANopenNode

|  |  |
| --- | --- |
| **Название файла** | **Назначение** |
| CANopen.h | Заголовочный файл, который определяет переменные и структуры, необходимые для реализации объектов протокола CANopen |
| CO\_errors | Заголовочный файл для обработки ошибок |
| CO\_stack.c | Совокупность реализаций всех объектов протокола CANopen (PDO, SDO, NMT и т.д) |
| CO\_driver.h  CO\_driver.c | Драйвер работы с модулем CAN, зависит от типа микроконтроллера |

Изменять пришлось только драйвер работы с модулем CAN (CO\_driver), а, поскольку программировать системы решено, используя Stellarisware, то большинство функций было реализовано с использованием компонентов этой библиотеки.

В итоге в драйвере работы с модулем CAN были реализованы следующие функции:

Таблица 7 – Функции драйвера работы с модулем CAN (CO\_driver)

|  |  |
| --- | --- |
| **Название функции** | **Описание** |
| CO\_DriverInit() | Функция, вызываемая в начале исполняемой программы; инициализирует драйвер |
| CO\_ProcessDriver() | Функция, вызываемая циклически в теле программы; контролирует работу драйвера |
| CO\_SetupCAN() | Функция, инициализирующая CAN контроллер |
| CANIntHandler | Обработчик прерываний для CAN контроллера |
| CO\_CANrxIsr() | Функция, вызываемая при прерывании, когда получено новое CAN сообщение |
| CO\_CANtxIsr() | Функция, вызываемая при прерывании, когда отправлено CAN сообщение |
| CO\_OD\_Read() | Функция, вызываемая SDO сервером для чтения данных из объектного словаря клиента |
| CO\_OD\_Write() | Функция, вызываемая SDO сервером для записи данных в объектный словарь клиента |

В результате реализации вышеуказанных функций был успешно портирован стек протокола CANopenNode.

# 2.3 Проектирование сети CANopen

Для проектирования узлов сети CANopen разработчиками CANopenNode было создано приложение Object Dictionary Editor (см. рисунок 13). С помощью него можно по своему усмотрению сконфигурировать объектный словарь для узла сети CANopen.

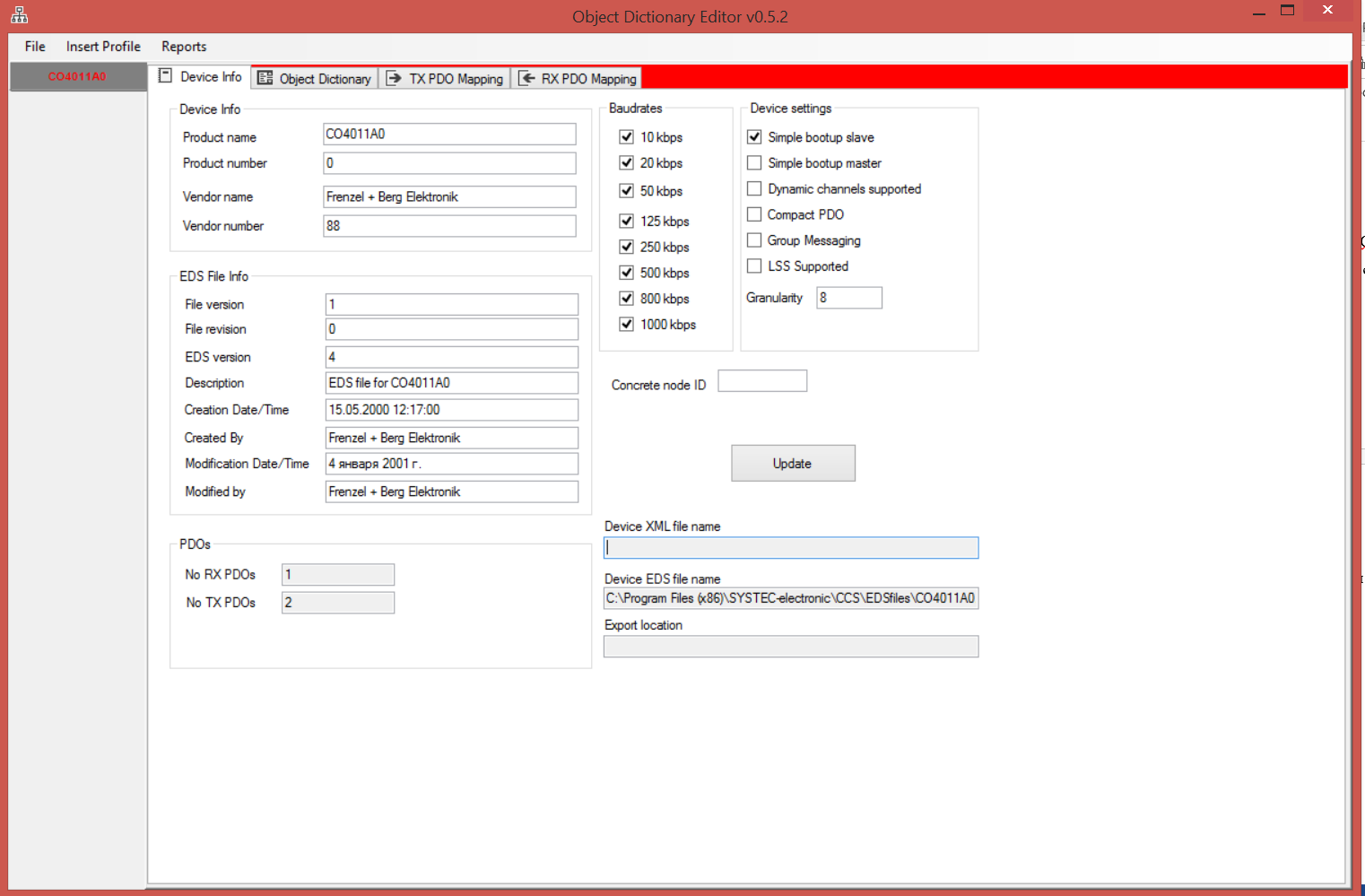


Рисунок 13 – Пример экранной формы Object Dictionary Editor

Разрабатываемая система состоит из двух узлов (БЧЭ и БУ), а всего доступно 128 (от 0 до 127) идентификаторов узла, поэтому можно выбрать любую пару идентификаторов. Было решено выбрать для них следующие идентификаторы:

* для БЧЭ – Node Id = 2.
* для БУ – Node Id = 1

В соответствии с протоколом информационного взаимодействия БЧЭ с БУ, скорость передачи данных необходимо установить в 1 Мбит/Сек.

Поскольку БУ должен управлять работой БЧЭ, то необходимо использовать сервис NMT. В данном случае БУ – Master, а БЧЭ – Slave.

Для того, чтобы диагностировать работу Slave – узла (БЧЭ), используем сервис Heatrbeat. БЧЭ будет посылать сообщения состояния каждую секунду узлу NMT Master, то есть блоку управления. NMT Slave отправляет в сеть сообщение состояния с COB-ID=NODEID+700hex. В поле данных этого сообщения указывается состояние NMT Slave:

* 0 – BootUp,
* 3 – Stopped,
* 5 – Operational,
* 7Fhex - Pre-Operational.

Так как в рамках поставленных задач для БЧЭ необходимо обмениваться показаниями датчиков, словом состояния и данными ошибках, то потребуется сконфигурировать PDO объекты объектного словаря. Исходя из таблицы 4 и условия, что БЧЭ предполагается использовать в штатном режиме, необходимо реализовать флаги 0, 1, 4, 5, 8. Соответственно, по 5 TPDO и RPDO.

Согласно протоколу CAN, сообщение, основная цель которого – инициализировать передачу сообщения другим узлом – кадр запроса (Remote Frame). Однако в CANopen в основном используются кадры данных (Data Frame). К тому же БЧЭ тоже должен обмениваться только кадрами данных, согласно протоколу информационного взаимодействия.

В таблице 8 представлены параметры для TPDO и RPDO объектов модулей БЧЭ и БУ соответственно, которые были установлены исходя из протокола информационного взаимодействия

Таблица 8 – Параметры отображения для TPDO и RPDO объектов модуля БЧЭ

|  |  |  |
| --- | --- | --- |
| **Параметр для БЧЭ (параметр для БУ)** | **Назначение** | **Значение** |
| TPDO 0 COB-ID (RPDO 0 COB-ID) | Кватернион | 0x43 |

Продолжение таблицы 8

|  |  |  |
| --- | --- | --- |
| **Параметр для БЧЭ (параметр для БУ)** | **Назначение** | **Значение** |
| TPDO 1 COB-ID (RPDO 1 COB-ID) | Угловая скорость, рад/с | 0x5E3 |
| TPDO 2 COB-ID (RPDO 2 COB-ID) | Линейное ускорение, м/ | 0x643 |
| TPDO 3 COB-ID (RPDO 3 COB-ID) | Температура | 0x663 |
| TPDO 4 COB-ID (RPDO 4 COB-ID) | Программный Reset МК | 0x6C3 |
| RPDO 0 COB-ID (TPDO 0 COB-ID) | Запрос данных от БУ к БЧЭ | 0x202 |

# 2.4 Разработка ПО микроконтроллеров

Разработка программ для МК будет вестись на языках С/С++. Для их отладки используется отладочная плата EVB и среда CodeComposer Studio 6.2.

На рисунке 14 представлена общая схема алгоритма программы МК для БЧЭ.

Как видно из структуры, сначала происходит установка частоты микроконтроллера. Производителем (TI) рекомендована рабочая частота в 8 МГц, ее и установим. Коэффициент предделителя установим в 1.

Далее происходит инициализация всех основных модулей, необходимых для реализации функционала:

* UART консоль;
* OLED – Дисплей;
* Пользовательские кнопки;
* I2C контроллер;
* CAN Драйвер;
* Драйвер MPU 6050.

Установка приоритетов прерываний устанавливается следующим образом: больший приоритет имеют CAN прерывания, затем прерывания таймера T0 и прерывания I2C.

После разрешается вызов всех прерываний и программа переходит в вечный цикл, где вызывается функция обработки магистрали CANopen (описана в файле CANopen\_stack.c) и функция исполняемого процесса.

Как уже было сказано ранее, оценочные платы EVB могут использовать UART0 в качестве виртуального COM-порта.

Для инициализации UART-консоли необходимо разрешить работу пинов порта ввода/вывода, отвечающих за UART0 (пины PA0 и PA1). Затем задается режим работы для этих пинов и после UART0 инициализируется для работы в режиме консоли (для этого используется готовая функция UARTStdioInit() из библиотеки StellarisWare).

Для инициализации OLED дисплея вызывается функция RIT128x96x4Init(1000000), объявленная в StellarisWare. Данная функция инициализирует интерфейс SSI для дисплея OLED и настраивает встроенный контроллер SSD1329. Далее задаем тактовый импульс Main OSC равной 8 МГц.

Для инициализации таймера Timer0, приемопередатчиков CAN и I2C также вызываются соответствующие функции из StellarisWare.

а) б)

Рисунок 14 - Общая схема алгоритма программы МК для БЧЭ (а) и БУ (б)

Процесс инициализации UART – консоли представлен на рисунке 15.



Рисунок 15 – Схема алгоритма инициализации UART- консоли

Пример кода для иллюстрации процесса инициализации UART-консоли представлен ниже:

void

InitConsole(void)

{

// Включить порт А

//

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//Конфигурирование пинов PA1 и PA2 для работы в режиме UART

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Инициализация UART

//

UARTStdioInit(0);

}

Процесс инициализации I2C-шины для модуля БЧЭ представлен на рисунке 16.



Рисунок 16 – Схема алгоритма инициализации I2C-шины

Пример кода для иллюстрации процесса инициализации I2C-шины для БЧЭ представлен ниже:

void I2C\_Init()

{

//Разрешить работу I2C

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0);

//Перезапуск I2C

SysCtlPeripheralReset(SYSCTL\_PERIPH\_I2C0);

//Включение порта B

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB);

// Конфигурирование пинов порта B для работы в режие I2C

GPIOPinConfigure(GPIO\_PB2\_I2C0SCL);

GPIOPinConfigure(GPIO\_PB3\_I2C0SDA);

GPIOPinTypeI2CSCL(GPIO\_PORTB\_BASE, GPIO\_PIN\_2);

GPIOPinTypeI2C(GPIO\_PORTB\_BASE, GPIO\_PIN\_3);

// Конфигурирование Мастер-узла

I2CMasterInitExpClk(I2C0\_MASTER\_BASE,SysCtlClockGet(), false);

}

Процесс инициализации исполняемого процесса представлен на рисунке 17.



Рисунок 17 – Схема алгоритма инициализации исполняемого процесса.

При инициализации исполняемого процесса система сбрасывает значения цифровых входов. Это необходимо делать согласно стандарту CiA 401, описывающему профиль CANopen для модулей ввода/вывода, которым и является блок чувствительных элементов [17]. Также установим все порты ввода/вывода как цифровые.

Процесс инициализации драйвера MPU6050 изображен на рисунке 18.

При инициализации драйвера MPU6050 настраивается работа акселерометра и гироскопа, а именно выбирается значение линейных ускорений (± 16g) и рабочее значение для угловых скоростей гироскопа (±500 ° / s).

Согласно [18], изначально модуль GY-521 находится в режиме сна, т.е. не осуществляет передачу данных на шину I2C. Для вывода его из режима сна необходимо записать в регистр по адресу 0x6B в 7 бит 0, таким образом, модуль сенсоров перейдет в операционное состояние.



Рисунок 18 – Схема алгоритма инициализации драйвера MPU6050

При запуске, модуль БЧЭ должен иметь предоперационное состояние. В таком состоянии он не передает PDO объекты и ожидает управляющей NMT команды от блока управления.

На рисунке 19 изображен процесс сброса соединения CAN при запуске узла.



Рисунок 19 – Схема алгоритма процесса сброса соединения CAN

Первым делом узел переводится в предоперационное состояние, в котором он не отправляет никаких объектов PDO. Далее каждый узел подгружает из объектного словаря параметры конфигурации CANopen (Bitrate, Node Id, PDO, NMT, Heartbeat), очищает массивы для хранения передаваемых/принимаемых данных и запускает работу шины CAN. При этом БУ автоматически переводится в операционное состояние, поскольку объектный словарь сконфигурирован для него как для NMT Мастера. Далее БУ переводит БЧЭ в операционное состояние, в котором тот уже будет готов к обмену информационными данными.

После всех подготовительных этапов программа попадает в цикл, где вызывается функция пользовательского процесса (см. рисунок 20 для БЧЭ, рисунок 21 – для БУ), в которой выполняется все взаимодействие пользователя с системой.



Рисунок 20 – Схема алгоритма вызова исполняемого процесса для БЧЭ

Пользовательский процесс вызывается циклически, поэтому каждый раз сбрасываются состояния кнопок. Затем выполняемые действия целиком зависят от нажатых кнопок.

Кнопки UP\_SW и DOWN\_SW блока управления используются для обхода списка формируемых запросов в БЧЭ. Выбор/отмена нужных запросов осуществляется при нажатии на кнопку SELECT\_SW. Если был выбран хотя бы один запрос, то формируется объект TPDO и отсылается в сеть CANopen.



Рисунок 21 – Схема алгоритма вызова исполняемого процесса для БУ

БЧЭ ожидает входного запроса от БУ, и, если такой поступил, разбирает полученное сообщение и формирует ответ для БЧЭ.

При нажатии LEFT\_SW для БЧЭ и БУ выводятся элементы CAN протокола в консоль UART, при нажатии RIGHT\_SW – в консоль выводятся элементы протокола CANopen.

Процесс считывания показаний датчиков изображен на рисунке 22.



Рисунок 22 – Схема алгоритма считывания показаний датчиков модуля GY-521

Согласно [18], показания датчиков модуль GY-521 записывает в регистры, каждый из которых описан в таблице 9.

Таблица 9 – Регистры данных модуля GY-521.

|  |  |  |
| --- | --- | --- |
| **Адрес** | **Имя регистра** | **Содержимое регистра** |
| 0x3B | ACCEL\_XOUT\_H | Старший байт показаний акселерометра (ось oX) |
| 0x3C | ACCEL\_XOUT\_L | Младший байт показаний акселерометра (ось oX) |
| 0x3D | ACCEL\_YOUT\_H | Старший байт показаний акселерометра (ось oY) |
| 0x3E | ACCEL\_YOUT\_L | Младший байт показаний акселерометра (ось oY) |
| 0x3F | ACCEL\_ZOUT\_H | Старший байт показаний акселерометра (ось oZ) |
| 0x40 | ACCEL\_ZOUT\_L | Младший байт показаний акселерометра (ось oZ) |
| 0x41 | TEMP\_OUT\_H | Старший байт показаний сенсора температуры |

Продолжение таблицы 9

|  |  |  |
| --- | --- | --- |
| **Адрес** | **Имя регистра** | **Содержимое регистра** |
| 0x42 | TEMP\_OUT\_L | Младший байт показаний сенсора температуры |
| 0x43 | GYRO\_XOUT\_H | Старший байт показаний гироскопа (ось oX) |
| 0x44 | GYRO\_XOUT\_L | Младший байт показаний гироскопа (ось oX) |
| 0x45 | GYRO\_YOUT\_H | Старший байт показаний гироскопа (ось oY) |
| 0x46 | GYRO\_YOUT\_L | Младший байт показаний гироскопа (ось oY) |
| 0x47 | GYRO\_ZOUT\_H | Старший байт показаний гироскопа (ось oZ) |
| 0x48 | GYRO\_ZOUT\_H | Младший байт показаний гироскопа (ось oZ) |

Чтобы начать процесс считывания данных, БЧЭ подключается по шине I2C к модулю GY-521 по адресу 0x68 [18] и считывает данные последовательно из регистров 0x3B – 0x48, при этом проверяя, свободна ли шина I2C.

# 2.5 Разработка диаграммы взаимодействия пользователя с системой

Для описания особенностей взаимодействия пользователя с системой, а именно обмена показаниями датчиков, была сформирована диаграмма взаимодействия пользователя с системой (см. рисунок 23).

Диаграмма детально описывает последовательность взаимодействия пользователя с системой при формировании запроса. Переход по доступным элементам для запроса осуществляется нажатием кнопок UP\_SW и DOWN\_SW, подтверждение выбора запроса осуществляется при нажатии на кнопку SELECT\_SW.

Далее блок управления формирует запрос и посылает его блоку чувствительных элементов в виде TPDO объекта. БЧЭ снимает показания данных с датчиков и формирует ответное сообщение, содержащее различный набор данных в зависимости от запроса.

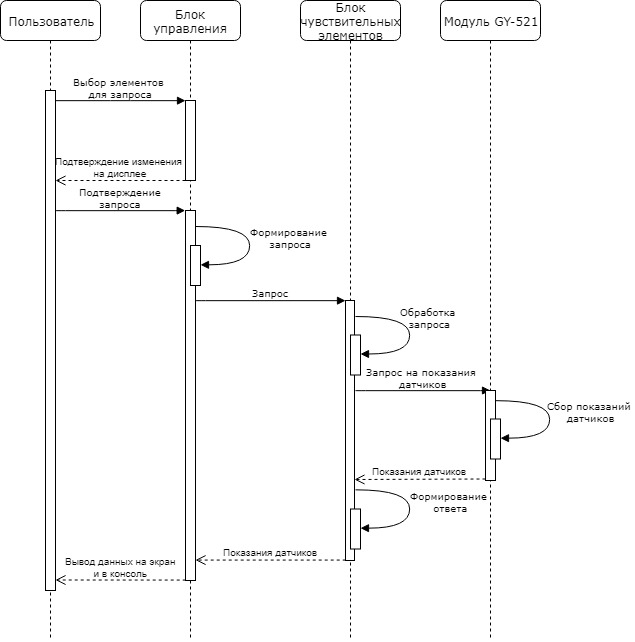


Рисунок 23 – Диаграмма взаимодействия пользователя с системой

# 2.6 Разработка диаграмм компоновки программного обеспечения

В виду наличия большого количества программных компонентов системы было решено разработать диаграммы компоновки компонентов БЧЭ и БУ.

Диаграммы компонентов показывают то, как выглядит программный продукт на физическом уровне, то есть из каких частей он состоит и как эти части взаимосвязаны между собой.

На рисунках 24-25 представлены диаграммы компоновки файлов, содержащих исходный код для программной реализации модулей БЧЭ и БУ соответственно



Рисунок 24 – Диаграмма компоновки программного модуля БЧЭ



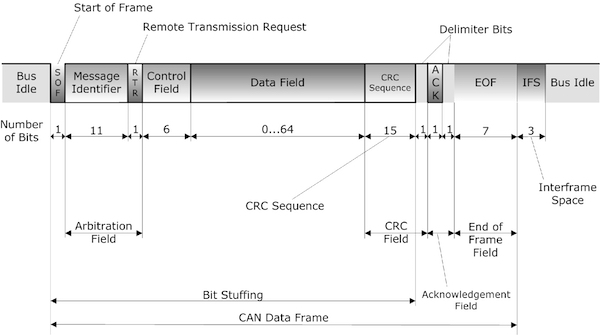
Рисунок 25 – Диаграмма компоновки программного модуля БУ

Основным исполняемым файлом для обоих модулей является Main.c, из которого уже вызываются компоненты библиотеки CANopenNode (см. таблицу 6) файлы пользовательских процессов (для БЧЭ Block\_of\_sensors.c и mpu6050\_sensors.c, для БУ – Block\_of\_control.c)

# 2.7 Расчет скорости передачи данных

Как было описано ранее, для передачи данных используются интерфейсы I2C и CAN. Для расчета скорости передачи данных будем учитывать только пропускную способность без времени задержки нажатия кнопок, поскольку задержки по времени при этом крайне малы. Также, учитывая высокую рабочую частоту МК LM3S8962 (до 50 МГц, базовая частота – 8 МГц) и LM3S8962 (до 25 МГц, базовая – 8 МГц), не будем заносить в расчеты время задержки выполнения команд.

Приемопередатчик CAN настроен на работу при скорости 1 Мбит/с, согласно протоколу работы БЧЭ. Длина линии передачи данных в таком случае ограничивается расстоянием в 40 метров, что более чем достаточно в рамках макета.

  
Рисунок 24 – Формат стандартного кадра CAN данных

Исходя из рисунка 24, можно посчитать размер кадра CAN стандартного формата. Будем учитывать максимально возможный размер области данных (8 байт). Пользуясь рисунком, был вычислен размер кадра, он составляет 111 бит.

(1)

I2C шина микроконтроллера LM3S8962 (модуль БЧЭ) настроена на частоту приема 400 кбит/с, на которой работает модуль GY-521 [18].

На рисунке 25 изображена стандартная последовательность данных на шине при считывании данных из Slave-устройста.

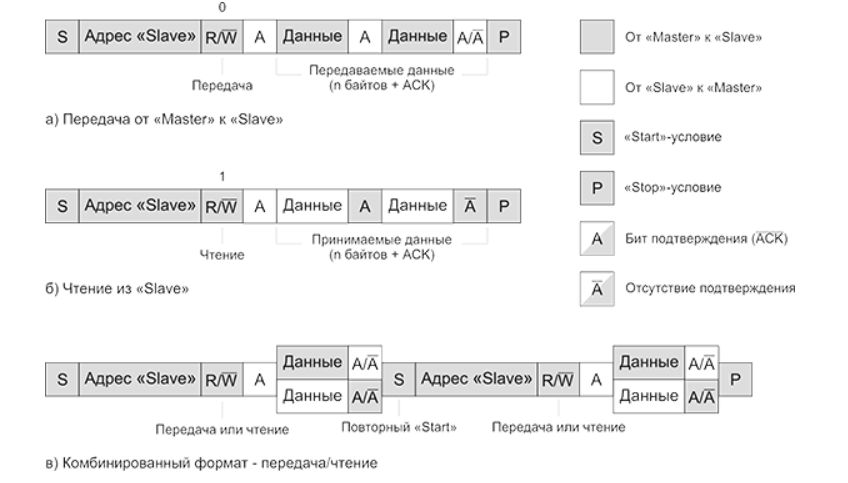


Рисунок 25 – Формат операций чтения/записи I2C шины

Т.к считывается суммарно 14 байт данных, а также посылается байт адреса узла Slave, то

(2)

Исходя из формул (1) и (2) и особенностей разрабатываемого ПМПИ, рассчитаем скорость передачи данных в одно направление:

(3)

# 3 Разработка технологии тестирования работоспособности системы

Тестирование системы – проверка соответствия системы заявленным характеристикам, сравнение ожидаемого результата с полученным.

Для стабильной работы системы необходима исправная работа модулей БЧЭ и БУ, а также правильно сконфигурированная сеть CANOpen.

# 3.1 Тестирование работоспособности компонентов системы

Отладка программы также производилась в среде CodeComposer Studio, рекомендованной разработчиком оценочных наборов Stellaris EKS-LM3S8962 Evaluation Kit. В режиме отладки среда имеет вид, изображенный на рисунке 30.

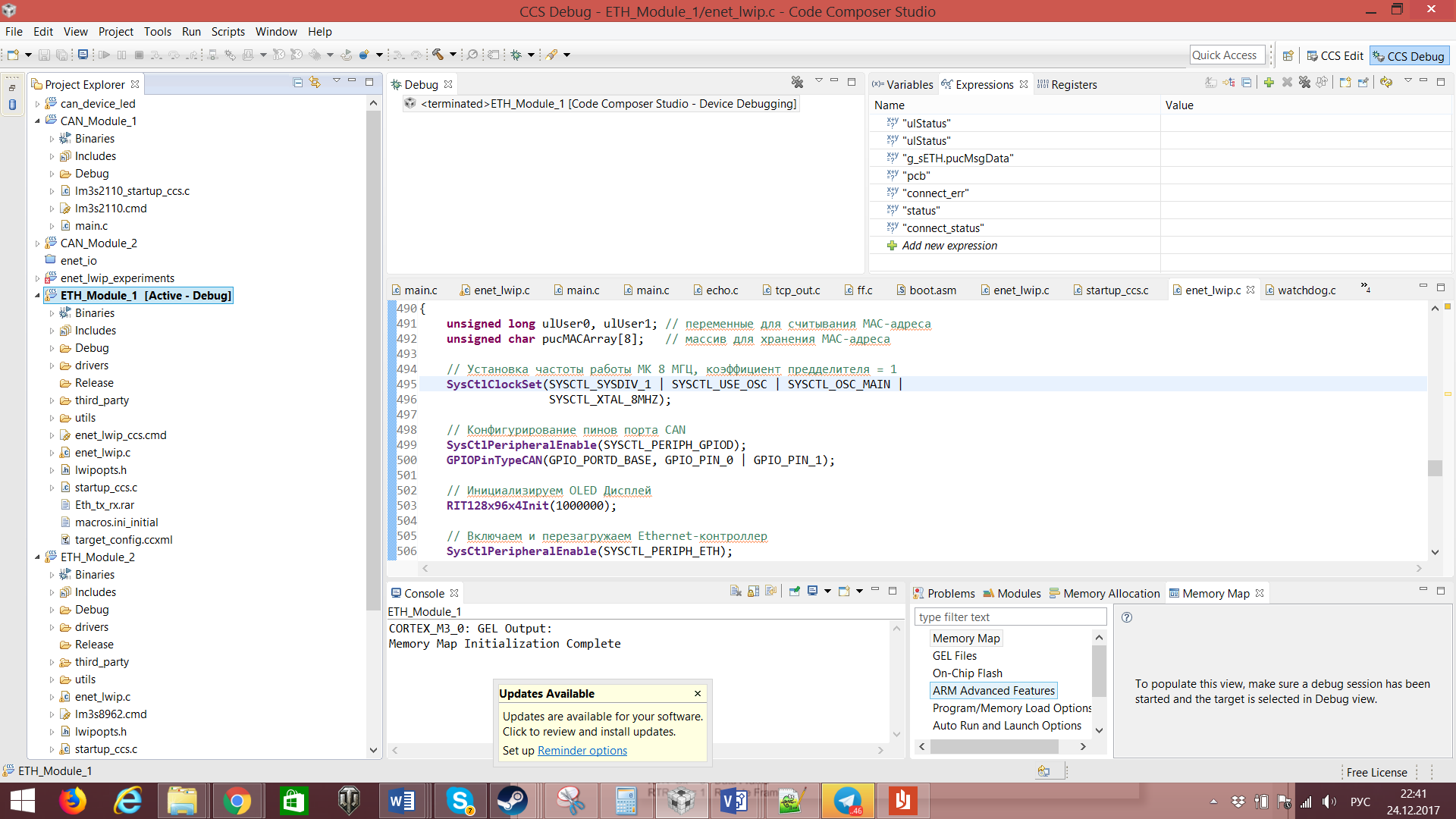


Рисунок 26 – Пример экранной формы CodeComposer Studio

В ходе нескольких отладочных сессий с активным использованием точек останова в прерываниях, программа была полностью отлажена. Также, как средство отладки был использован вывод отладочной информации на дисплей, расположенный на оценочных платах EVB, а также в UART-консоль.

Несмотря на доступ к исходному коду было принято решение проведения тестирования методом функционального тестирования («чёрный ящик»), без структурного тестирования («белый» или «серый ящик»). Таким образом, можно достичь максимальной схожести использования системы в реальных условиях и выявить самые явные недоработки системы.

Результаты тестирования работы программной части БЧЭ приведены в таблице 10.

Таблица 10 – Тестирование программной части модуля БЧЭ

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение теста | Описание теста | Ожидаемый результат | Полученный результат |
| 1 | 2 | 3 | 4 |
| Тестирование работы меню выбора запроса к БУ | Навигация по меню нажатием кнопок UP\_SW и DOWN\_SW | Корректная навигация по меню выбора запроса к БУ | Корректная и наглядная навигация по меню выбора запроса |
| Тестирование настроек конфигурации CANopen – сети и CAN-шины | Вывод в UART-консоль настроек конфигурации CANopen – сети и CAN-шины | Настройки конфигурации CANopen – сети и CAN-шины соответствуют Объектному Словарю БЧЭ | Настройки конфигурации CANopen – сети и CAN-шины соответствуют Объектному Словарю БЧЭ |
| Тестирование обмена данными с БУ | Формирование выбранного в меню запроса по нажатию кнопки SELECT\_SW и отправление его на шину CAN, получение ответа исходя из запроса | Корректное формирование выбранного в меню запроса и получение соответствующих запросу данных | Корректное формирование выбранного в меню запроса и получение соответствующих запросу данных |

Тестирование не выявило ошибок.

Аналогичным образом было произведено функциональное тестирование блока управления. Результаты тестирования приведены в таблице 11.

Таблица 11 – Тестирование программной части модуля БЧЭ

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение теста | Описание теста | Ожидаемый результат | Полученный результат |
| 1 | 2 | 3 | 4 |
| Тестирование чтения показаний датчиков | Чтение показаний датчиков модуля GY-521 | Корректно изменяющиеся данные при изменении положения и угла наклона модуля GY-521, а также при нагревании и охлаждении сенсора температуры | Корректно изменяющиеся данные |
| Тестирование настроек конфигурации CANopen – сети и CAN-шины | Вывод в UART-консоль настроек конфигурации CANopen – сети и CAN-шины | Настройки конфигурации CANopen – сети и CAN-шины соответствуют Объектному Словарю БУ | Настройки конфигурации CANopen – сети и CAN-шины соответствуют Объектному Словарю БУ |

Тестирование не выявило ошибок в программном обеспечении КУ.

Во всех тестах система показала свою работоспособность и корректное реагирование действия пользователя системы.

# 3.2 Тестирование протокола передачи данных высокого уровня

Для тестирования протокола передачи данных высокого уровня необходимо анализировать данные, которые передаются по сети CAN и стандартизированы согласно протоколу CANOpen. Для этого используются анализаторы CAN-шины.

Т.к. разрабатываемая система построена на основе протокола CANOpen, то и анализатор сети такой, который поддерживает соответствующий протокол.

Одним из самых популярных [20] анализаторов сети CANOpen является устройство USB-to-CAN v2, которое, с одной стороны, подключается к шине CAN и анализирует данные, передаваемые по шине, а с другой – по интерфейсу USB к персональному компьютеру. Внешний вид данного анализатора представлен на рисунке 27



Рисунок 27 – Анализатор сети CANOpen USB-to-CAN v2

Далее, с помощью специального программного обеспечения MiniMon v3 by IXXAT, написанного специально для данного анализатора, можно наглядно анализировать пакеты данных на экране монитора (см. рисунок 28) (предварительно подключив пины CANL и CANH к сети CAN микропроцессорной системы, используя [15][21])

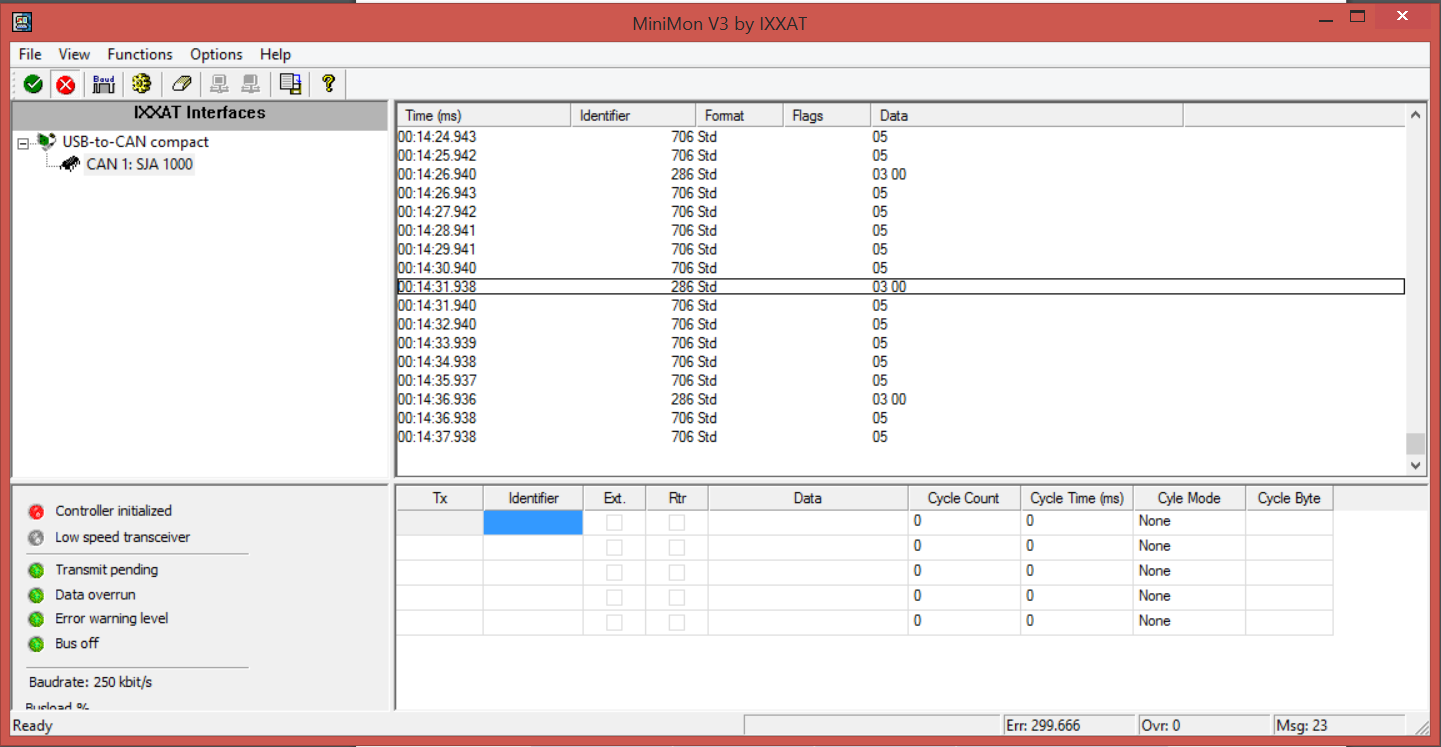


Рисунок 28 – Процесс анализа сети CANOpen, используя программное обеспечение MiniMon v3 by IXXAT

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы было выполнено следующее:

* проведен аналитический обзор интерфейсов взаимодействия БЧЭ с пользователем;
* изучен способ организации промышленной сети для стандартизированного обмена данными на основе протокола CANopen;
* изучен и адаптирован протокол информационного взаимодействия БЧЭ с потребителем;
* разработана структурная и функциональная схемы системы цифрового обмена БЧЭ гидролокационного комплекса;
* разработан программный модуль реализации стека CANopenNode для микроконтроллеров семейства Stellaris;
* разработаны рабочие компоненты системы и объединены в сеть CANopen.
* проведено тестирование и отладка компонентов системы.

В планах дальнейшее усовершенствование системы путем увеличения компонентов сети CANopen, а именно увеличение управляемых компонентов системы, содержащих различные датчики, для построения гидролокационного комплекса.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Блок Чувствительных Элементов (БЧЭ) [Электронный ресурс] // Техногаз. URL: https://texnogaz.ru/blok-chuvstvitelnyh-elementov-bche (дата обращения 05.10.2017).
2. Блоки чувствительных элементов навигационной системы [Электронный ресурс] // Корпорация Ориентир. URL: http://orientir.tech/inertial-measurement-unit.html (дата обращения 05.10.2017).
3. Введение в протокол CAN [Электронный ресурс] // CAN – технологии. URL: <http://can.marathon.ru/page/can-protocols/canbus/canintro> (дата обращения 05.10.2017).
4. RS-485 [Электронный ресурс] // Википедия – свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/RS-485 (дата обращения 06.10.2017).
5. Сравнение CAN и RS-485 [Электронный ресурс] // Дейтамикро – CAN и смежные технологии. URL: http://www.datamicro.ru/download/CAN\_vs\_RS485[EA,%20rus].pdf (дата обращения 11.10.2017).
6. Обзор протокола CANopen [Электронный ресурс] // Microsin.net. URL: <http://microsin.net/programming/arm/canopen-overview.html> (дата обращения 11.10.2017).
7. DeviceNet [Электронный ресурс] // Википедия – свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/DeviceNet (дата обращения 11.10.2017).
8. DeviceNet network protocol [Электронный ресурс] // Nerds in control. URL: http://control.com/thread/1241972766 (дата обращения 11.10.2017).
9. Использование сети CAN и стека CANopen [Электронный ресурс] // Хабрахабр. URL: <https://habrahabr.ru/post/144435> (дата обращения 10.10.2017).
10. Протокол высокого уровня CANopen [Электронный ресурс] // РадиоЛоцман. URL: <https://www.rlocman.ru/review/article.html?di=147764#A4> (дата обращения 11.10.2017).
11. Введение в протокол DeviceNet [Электронный ресурс] // CAN – технологии. URL: <http://can.marathon.ru/page/can-protocols/devicenet/devicenetintro> (дата обращения 10.11.2017).
12. CAN протоколы высокого уровня [Электронный ресурс] // Справочник по электронным компонентам. URL: <http://www.gaw.ru/html.cgi/txt/publ/interface/can.htm> (дата обращения 10.11.2017).
13. Обзор и сравнение популярных открытых промышленных сетей [Электронный ресурс] // plc4good. URL: <http://plc4good.org.ua/post_print.php?id=29> (дата обращения 10.11.2017).
14. Акселерометр и гироскоп MPU6050 [Электронный ресурс] // cxem.net Сайт паяльник. URL: http://cxem.net/mc/mc324.php?sa=X&ved=0CBcQ9QEwAWoVChMIoOTw1cr0xgIVyKtyCh0bawoK (дата обращения 20.11.2017).
15. Stellaris LM3S8962 Evaluation Board User’s Manual, Texas Instruments, 2010, 36 с.
16. CANopenNode Manual, Janez Paternoster, 2005, 10 C.
17. CANopen профиль CiA 401, Марафон. Изделия IO Remote 2015, 55 C.
18. MPU-6050 Register Map and Descriptions, InvenSense, 2013. 47 с.
19. Интерфейс I2C [Электронный ресурс] // СпецПромДизайн. URL: <https://spd.net.ru/Article/I2C.aspx> (дата обращения 10.11.2017).
20. Интерфейс IXXAT USB-to-CAN v2 [Электронный ресурс] // Автоматизация в промышленности. URL: http://avtprom.ru/news/2014/02/03/interfeis-ixxat-usb-can-
21. USB-to-CAN v2 – Руководство пользователя, Дэйтамикро, 2014. 12 с.
22. ГОСТ 2.105-95 ЕСКД. Общие требования к текстовым документам.
23. ГОСТ 7.9-95 Реферат и аннотация. Общие требования.
24. ГОСТ 7.12-1993 Сокращение слов на русском языке
25. ГОСТ 7.1-2003 Библиографическая запись.
26. Иванова Г.С. Технология программирования. М.: Изд-во МГТУ им.Н.Э.Баумана, 2002. – 241 с.

# ПРИЛОЖЕНИЕ А Техническое задание

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Компьютерные системы и сети»

УТВЕРЖДАЮ

Зав. кафедрой ИУ6,

д.т.н., проф. \_\_\_\_\_\_\_\_А.В. Пролетарский

" \_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201 г.

Система цифрового обмена блока чувствительных элементов гидролокационного комплекса

Техническое задание

Листов 8

Руководитель,

к.т.н., доцент \_\_\_\_\_\_\_\_\_\_\_ Сотников А. А.

Исполнитель,

студ. гр. ИУ6-81 \_\_\_\_\_\_\_\_\_\_\_ Сидоров В. Ю.

2018 г.

1. Введение

Настоящее техническое задание распространяется на разработку программно-аппаратного комплекса, используемого для автоматизированного и стандартизированного обмена данными между блоком чувствительных элементов (БЧЭ) и блоком управления и предназначенного для организаций или групп лиц, заинтересованных в организации промышленной сети.

На сегодняшний день с уверенностью можно сказать, что датчики окружают нас повсюду, например, датчики в автомобиле, датчики системы умный дом и так далее. А что если предполагается использовать несколько датчиков и исполняемых устройств в едином корпусе или, допустим, предполагается их использование в экстремальных условиях? В таком случае датчики и исполняемые устройства объединяют в единый модуль – БЧЭ, который защищает встроенные датчики от условий окружающей среды и, можно сказать, объединяет их в некоторую локальную сеть.

БЧЭ распространены повсеместно и включают в себя целый набор датчиков, поэтому мониторинг показаний датчиков имеет важный характер. Разрабатываемая система основана на использовании общепринятого протокола высокого уровня, который стандартизирует обмен данными, поэтому она легко может быть интегрирована в любую промышленную сеть.

2. Основания для разработки

Система цифрового обмена БЧЭ гидролокационного комплекса разрабатывается на основе учебного плана кафедры ИУ6.

3. Назначение разработки

Основное назначение системы заключается организации промышленной сети, состоящей из БЧЭ и блока управления, для автоматизированного и стандартизированного обмена данными по протоколу высокого уровня. В состав БЧЭ входят такие датчики: акселерометр, гироскоп, датчик температуры. Пользователями системы могут быть организации, заинтересованные в организации промышленной сети для мониторинга показаний датчиков.

4. Исходные данные, цели и задачи

4.1. Исходные данные

4.1.1. Исходными данными для разработки является протокол информационного взаимодействия блока БЧЭ-Р-600М с потребителем.

4.2. Цель работы

Целью работы является проект системы цифрового обмена БЧЭ гидролокационного комплекса.

4.3. Решаемые задачи

4.3.1. Анализ требований технического задания, исследование предметной области.

4.3.2. Анализ интерфейсов взаимодействия БЧЭ.

4.3.3. Выбор датчиков, микроконтроллеров для имитации БЧЭ.

4.3.4. Выбор средств разработки.

4.3.5. Разработка электрической функциональной схемы.

4.3.6. Разработка электрической принципиальной схемы.

4.3.7. Разработка алгоритмов работы блока чувствительных элементов и блока управления.

4.3.8. Реализация компонентов с учетом выбранных средств и их тестирование.

5. Требования К СИСТЕМЕ

5.1. Требования к функциональным характеристикам

Система должна обеспечивать выполнение следующих функций:

5.1.1. Для пользователя

* просмотр результатов показаний датчиков БЧЭ в режиме реального времени;
* цифровой обмен данными между БЧЭ и блоком управления;
* управление передаваемой информацией и режимом работы БЧЭ;
* вывод отладочной информации;

5.1.2. Исходные данные:

* показания датчиков;
* период опроса датчиков;
* режим работы БЧЭ;
* пользовательский запрос к БЧЭ на выдачу данных.

5.1.3. Результаты:

* визуализация показаний датчиков;
* ответ от БЧЭ.

5.2. Требования к надежности

5.2.1. Предусмотреть контроль вводимой информации.

5.2.2. Предусмотреть защиту от некорректных действий пользователя.

5.3. Условия эксплуатации

5.3.1. Условия эксплуатации в соответствие с СанПиН 2.2.2/2.4.1340-03.

5.4. Требования к составу и параметрам технических средств

5.4.1. Блок чувствительных элементов и блок управления должны работать в режиме реального времени.

5.4.2. Блок чувствительных элементов и блок управления должны иметь интерфейс передачи данных CAN.

5.4.3. Блок чувствительных элементов и блок управления должны поддерживать протокол высокого уровня CANopen.

5.4.4. Датчик температуры должен работать с точностью до знака после запятой.

5.4.5. Датчик гироскопа должен работать в трех системах координат (Оx, Оy, Оz).

5.4.6. Датчик акселерометра должен работать в трех системах координат (Оx, Оy, Оz).

5.5. Требования к информационной и программной совместимости

5.5.1. Пакеты данных от блока чувствительных элементов и блока управления должны быть совместимы с программным обеспечением – анализаторами CAN-сети.

5.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

5.7. Требования к транспортированию и хранению

Требования к транспортировке и хранению не предъявляются.

6. Требования к документации

6.1. Разрабатываемые программные модули должны быть самодокументированы, т.е. тексты программ должны содержать все необходимые комментарии.

6.2. Разрабатываемое программное обеспечение должно включать справочную систему.

6.3. В состав сопровождающей документации должны входить:

6.3.1. Расчетно-пояснительная записка на 60 листах формата А4 (без приложений).

6.3.2. Техническое задание (Приложение A).

6.4. Графическая часть должна быть выполнена на 6 листах формата А1 (копии формата А3, А4 включить в качестве приложений к расчетно-пояснительной записке):

6.4.1. Постановка задачи, структурная схема системы цифрового обмена блока чувствительных элементов гидролокационного комплекса.

6.4.2. Функциональная схема элементов системы.

6.4.3. Структурная схема БЧЭ, состав и структура пакетов данных

6.4.3. Схемы алгоритмов.

6.4.4 Диаграмма взаимодействия пользователя с системой и диаграммы компоновки.

6.4.5. Конфигурирование объектного словаря и тестирование протокола передали данных CANOpen

7. Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название этапа | Срок,  даты, % | Отчетность |
| 1. | Разработка технического задания | 01.01.2018-05.05.2018  8 % | Техническое задание |
| 2. | Анализ требований и уточнение спецификаций (эскизный проект) | 06.02.2018-05.03.2018  20% | Спецификации программного обеспечения. |
| 3. | Проектирование структуры программного обеспечения, проектирование компонентов (технический проект) | 06.03.2018-31.03.2018  40% | Схема структурная системы и спецификации компонентов, схемы алгоритмов, диаграмма последовательности взаимодействия. |
| 4. | Реализация компонент и автономное тестирование компонентов.  Сборка и комплексное тестирование.  Оценочное тестирование и (рабочий проект). | 01.04.2018-30.04.2018  80% | Тексты программных компонентов.  Тесты, результаты тестирования. |
| 5. | Разработка программной документации | 01.05.2018-27.05.2018  90% | Программная документация. |
| 6. | Подготовка доклада и предзащита | 28.05.2018- 19.06.2018  95 % | Доклад |
| 7. | Защита проекта | 20.06.2018  100 % |  |

8. Порядок контроля и приемки

8.1. Порядок контроля

Контроль выполнения осуществляется руководителем еженедельно.

8.2. Порядок защиты

Защита осуществляется перед государственной экзаменационной комиссией (ГЭК).

8.3. Срок защиты

Срок защиты определяется в соответствии с планом заседаний ГЭК.

9. ПРИМЕЧАНИЕ

В процессе выполнения работы возможно уточнение отдельных требований технического задания по взаимному согласованию руководителя и исполнителя.

# ПРИЛОЖЕНИЕ Б Руководство пользователя

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Компьютерные системы и сети (ИУ6)

Система цифрового обмена блока чувствительных элементов гидролокационного комплекса

Руководство пользователя

Листов 8

Студент гр. ИУ6-81 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сидоров В.Ю.

(Подпись, дата) (И.О. Фамилия)

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сотников А.А.

(Подпись, дата) (И.О. Фамилия)

*2018 г.*

СОДЕРЖАНИЕ

[1 Инструкция по работе системы для пользователя 3](#_Toc517069471)

[1.1 Навигация по меню и формирование запроса для блока чувствительных элементов, используя блок управления 3](#_Toc517069472)

[1.2 Анализирование сети CANOpen 4](#_Toc517069473)

[1.2.1 Установка драйвера для IXXAT USB-to-CAN v2 5](#_Toc517069474)

[1.2.3 Процесс анализирования сети CANOpen 6](#_Toc517069475)

**1 Инструкция по работе системы для пользователя**

Основные функции по работе с системой можно разделить на 2 части: навигация по меню и формирование запроса для блока чувствительных элементов, используя блок управления и анализирование сети CANopen.

**1.1 Навигация по меню и формирование запроса для блока чувствительных элементов, используя блок управления**

Навигация по меню осуществляется с помощью навигационных кнопок UP\_SWITCH и DOWN\_SWITCH с использованием информации на дисплее микроконтроллера LM3S8962. Формирование запроса осуществляется нажатием кнопки SELECT\_SWITCH микроконтроллера.

Внешний вид микроконтроллера представлен на рисунке Б.1.

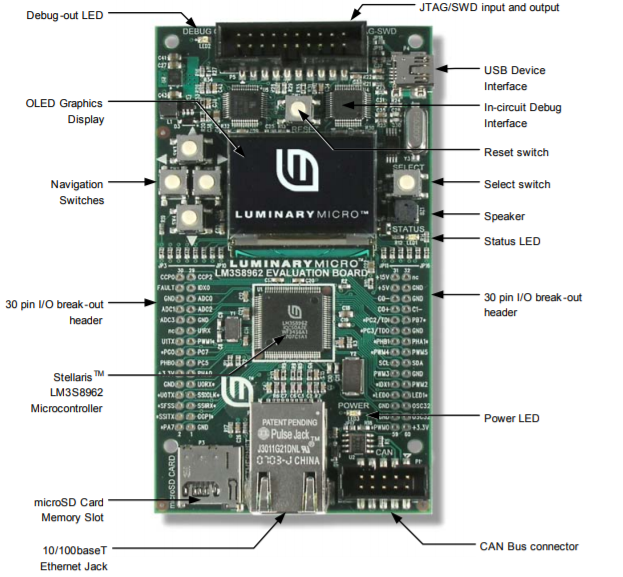


Рисунок Б.1 – Внешний вид микроконтроллера LM3S8962

Все возможные для формирования запросы, представленные в меню навигации, отражены в таблице Б.1.

Таблица Б.1 - Возможные для формирования запросы

|  |  |
| --- | --- |
| Параметр | Обозначение |
| Кватернион | λ0, λX, λY, λZ |
| Угловая скорость | ωX, ωY, ωZ |
| Линейное ускорение | aX, aY, aZ |
| Температура | Т1, Т2, Т3 |
| Reset | − |

**1.2 Анализирование сети CANOpen**

Для просмотра и анализирования пакетов, обмен которыми происходит внутри сети CANOpen, необходимо использовать анализатор сети. Предполагается использование анализатора сети IXXAT USB-to-CAN v2, изображенного на рисунке Б.2



Рисунок Б.2 – Анализатор сети CANOpen IXXAT USB-to-CAN v2

Перед началом работы, необходимо подключить устройство к ПК и установить необходимый драйвер.

**1.2.1 Установка драйвера для IXXAT USB-to-CAN v2**

Перед началом установки необходимо перейти на официальных сайт программного обеспечения для данного устройства (<https://www.ixxat.com/support/file-and-documents-download/drivers/vci-v3-driver-download>) и скачать утилиты установки драйвера VCI 3.5.4.4543 Installation.

При запуске установочной утилиты появится всплывающее окно, изображенное на рисунке Б.3

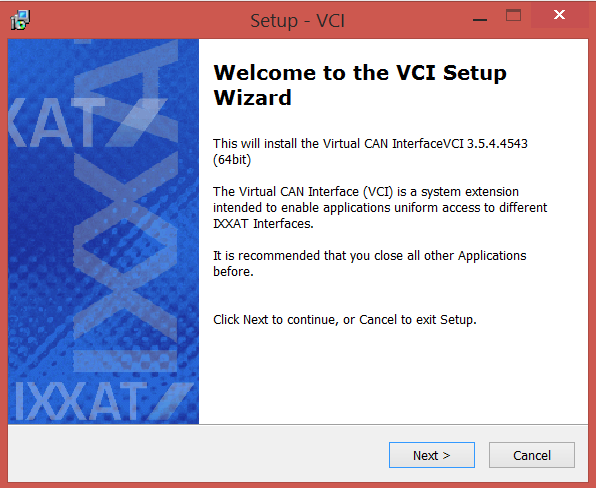


Рисунок Б.3 – Мастер установки драйвера VCI

Для установки ПО следуйте следующим шагам:

* В приветственном окне выберите Next;
* Примите лицензионное соглашение;
* В окне выбора компонентов выберите драйвер VCI V3.

**1.2.2 Установка программного обеспечения MiniMon V3**

После установки драйвера для устройства CANOpen IXXAT USB-to-CAN v2, необходимо установить программное обеспечение для анализирования CANOpen-сети.

Для этого необходимо перейти на сайт [www.ixxat.com](http://www.ixxat.com) и скачать установщик MiniMon V3 by IXXAT.

Для установки ПО следуйте следующим шагам:

* В приветственном окне выберите Next;
* Примите лицензионное соглашение;
* В окне выбора директории установки компонентов выберите необходимый путь.

**1.2.3 Процесс анализирования сети CANOpen**

После установки программного обеспечения, описанного выше, необходимо подсоединить устройство CANOpen IXXAT USB-to-CAN v2 к шине CAN. Данное устройство имеет интерфейс Sub-D9M, изображенный на рисунке Б.4

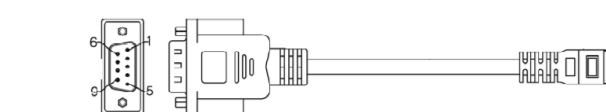


Рисунок Б.4 – Интерфейс Sub-D9M

Назначение пинов вывода представлено в таблице Б.2

|  |  |
| --- | --- |
| Сигнал | Пин |
| CAN High | 7 |
| CAN Low | 2 |
| GND | 3,6 |

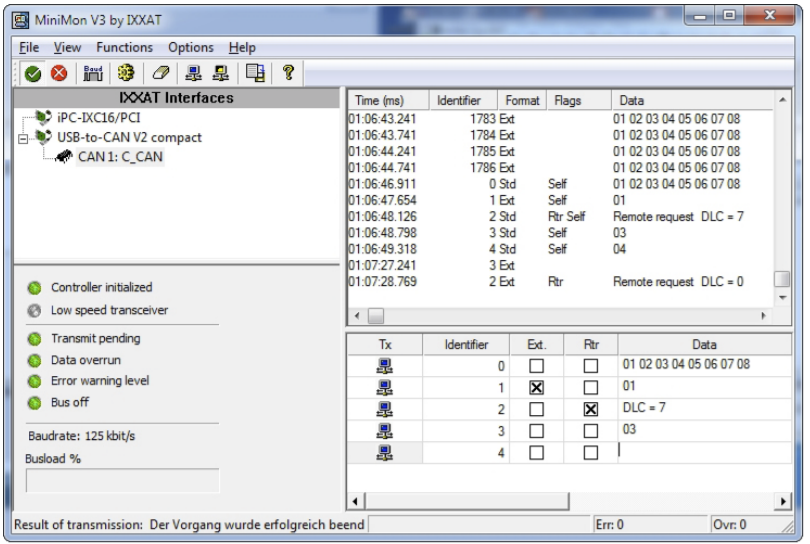
Главное окно MiniMon V3 изображено на рисунке Б.5

Рисунок Б.5 - Главное окно MiniMon V3

В списке доступных CAN-интерфейсов (Рисунок Б.6) необходимо выбрать USB-to-CAN v2.

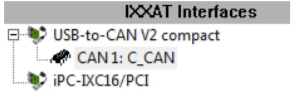


Рисунок Б.6 - Список доступных CAN-интерфейсов MiniMon V3

В окне приема сообщений (рисунок Б.7) отображается информация о полученных CAN объектах: метка времени, состояние, идентификатор и данные.

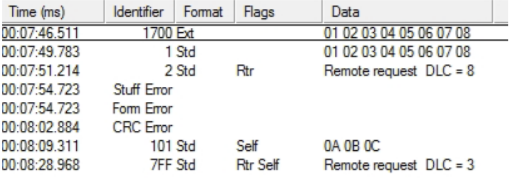


Рисунок Б.5 – Окно приема сообщений MiniMon V3

Описание колонок окна приема сообщений представлено в таблице Б.3

Таблица Б.3 - Описание колонок окна приема сообщений

|  |  |
| --- | --- |
| Название колонки | Значение |
| Time (Время) | Время приема сообщения – разрешение 0.001 секунд |
| Identifier (Идентификатор) | Идентификатор сообщения, отображается в шестнадцатеричном формате |
| Format (Формат) | Std – стандартный CAN формат (11 разрядный идентификатор)  Ext – расширенный CAN формат (29 разрядный идентификатор) |
| Flags (Флаги) | Дополнительные сведения о сообщении:  Ovr – После этого сообщения были потери сообщений  Rtr – Сообщение удаленного запроса  Self – Сообщение самоприема, передаваемое miniMon |
| Data (Данные) | Данные CAN сообщения, отображаемые побайтно в шестнадцатеричном формате |

# ПРИЛОЖЕНИЕ В Исходные коды программ

**Модуль БЧЭ – main.c**

**#include** "EK-LM3S8962-HW-Main.h"

**#include** "utils/uartstdio.h"

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_ints.h"

**#include** "inc/hw\_sysctl.h"

**#include** "inc/hw\_can.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/can.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/systick.h"

**#include** "drivers/rit128x96x4.h"

**#include** "CO\_driver.h"

**#include** "CANopen.h"

**#include** "debug\_info.h"

**#include** "mpu6050\_sensors.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Interrupt priority definitions. The top 3 bits of these values are

// significant with smaller values indicating higher priority interrupts.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#define** SET\_INT\_PRIORITY(i) (i << 5)

**#define** CAN\_INT\_PRIORITY SET\_INT\_PRIORITY(0) // highest priority

**#define** TIMER0B\_INT\_PRIORITY SET\_INT\_PRIORITY(1) // lower priority

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Counter to count the number of timer interrupts that have been called.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**volatile** **unsigned** **long** g\_ulCounter = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! This function sets up Timer0B to be used as a 1ms CANopen processing timer.

//!

//! \attention

//! The SysTick interrupt \b \*cannot\* be used for the 1ms timer because

//! the CANopenNode code disables the timer interrupts around critical

//! regions. If a SysTick interrupt is disabled when its next periodic

//! interrupt is due, the interrupt will be lost, not pended!

//! However, a standard timer interrupt will only be delayed while its

//! interrupt is disabled, and will occur as soon as its interrupt

//! is reenabled.

//!

//! \return None.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**static** **void** **InitTimer0B**(**void**)

{

//

// The Timer0 peripheral must be enabled for use.

//

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

//

// Configure Timer0B as a 16-bit periodic timer.

//

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_16\_BIT\_PAIR | TIMER\_CFG\_B\_PERIODIC);

//

// Set the Timer0B load value to 1ms.

//

**TimerLoadSet**(TIMER0\_BASE, TIMER\_B, **SysCtlClockGet**() / 1000);

//

// Clear any pending interrupt flag.

//

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMB\_TIMEOUT);

//

// Configure the Timer0B interrupt for timer timeout.

//

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMB\_TIMEOUT);

//

// Enable the Timer0B interrupt on the processor (NVIC).

//

**IntEnable**(INT\_TIMER0B);

//

// Enable Timer0B.

//

**TimerEnable**(TIMER0\_BASE, TIMER\_B);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! The interrupt handler for the Timer0B interrupt.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** **Timer0BIntHandler**(**void**)

{

//

// Clear the timer interrupt flag.

//

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMB\_TIMEOUT);

//

// Update the Timer0B interrupt counter.

//

g\_ulCounter++;

**CO\_Timer1msIsr**();

**if** ((g\_ulCounter & 0x0000001f) == 0)

{

TOGGLE\_STATUS\_LED();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Configure the CAN and enter a loop to implement CANopen processing.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int** **main**(**void**)

{

//

// Set the clocking to run from the PLL.

//

**SysCtlClockSet**(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN |

SYSCTL\_XTAL\_8MHZ);

//

// Disable interrupts to the processor.

//

**IntMasterDisable**();

**#ifdef** UART\_DEBUG

//

// Set up the serial console to use for displaying messages.

//

**InitConsole**();

**UARTprintf**("\n\nCANopenNode Tutorial Sensor\n");

**UARTprintf**("==========================\n");

**#endif** // UART\_DEBUG

//

// Initialize the OLED display.

//

RIT128x96x4Init(1000000);

RIT128x96x4StringDraw("Accel X", 0, 0, 15);

RIT128x96x4StringDraw("Accel Y", 0, 8, 15);

RIT128x96x4StringDraw("Accel Z", 0, 16, 15);

RIT128x96x4StringDraw("Gyro X", 0, 24, 15);

RIT128x96x4StringDraw("Gyro Y", 0, 32, 15);

RIT128x96x4StringDraw("Gyro Z", 0, 40, 15);

RIT128x96x4StringDraw("Quat. W", 0, 48, 15);

RIT128x96x4StringDraw("Quat. X", 0, 56, 15);

RIT128x96x4StringDraw("Quat. Y", 0, 64, 15);

RIT128x96x4StringDraw("Quat. Z", 0, 72, 15);

RIT128x96x4StringDraw("Temp", 0, 80, 15);

RIT128x96x4StringDraw("State: ", 0, 88, 15);

SELECT\_SW\_INIT();

LEFT\_SW\_INIT();

RIGHT\_SW\_INIT();

UP\_SW\_INIT();

DOWN\_SW\_INIT();

STATUS\_LED\_INIT();

// setup Timer0B interrupt

InitTimer0B();

**CO\_DriverInit**(); // CANopenNode driver init

**User\_Init**(); // User init

**CO\_ResetComm**(); // Reset CAN communication

//

// Set the priorities of the interrupts used by the application.

//

**IntPrioritySet**(INT\_CAN0, CAN\_INT\_PRIORITY);

**IntPrioritySet**(INT\_TIMER0B, TIMER0B\_INT\_PRIORITY);

//

// Enable interrupts to the processor.

//

**IntMasterEnable**();

init\_mpu6050();

**while** (1)

{

**CO\_ProcessMain**();

**User\_ProcessMain**();

}

}

**Модуль БЧЭ – BlockOfSensors.c**

// CAN board is specified in Makefile

**#include** <stdlib.h>

**#include** <math.h>

**#include** "EK-LM3S8962-HW-Main.h"

**#include** "utils/uartstdio.h"

**#include** "utils/ustdlib.h"

**#include** "inc/hw\_types.h"

**#include** "drivers/rit128x96x4.h"

**#include** "mpu6050\_sensors.h"

**#include** "CANopen.h"

**#include** "debug\_info.h"

/\*\*\*\*\* Device profile for Generic I/O \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#ifdef** CO\_IO\_DIGITAL\_INPUTS

/\*0x6000\*/ **extern** tData4bytes ODE\_Read\_Digital\_Input;

**#endif**

**#ifdef** CO\_IO\_DIGITAL\_OUTPUTS

/\*0x6200\*/ **extern** tData4bytes ODE\_Write\_Digital\_Output;

**#endif**

**#ifdef** CO\_IO\_ANALOG\_INPUTS

/\*0x6401\*/ **extern** INTEGER16 ODE\_Read\_Analog\_Input[];

**#endif**

**#ifdef** CO\_IO\_ANALOG\_OUTPUTS

/\*0x6411\*/ **extern** INTEGER16 ODE\_Write\_Analog\_Output[];

**#endif**

//#ifdef CAN\_DEVICE\_LM3S8962

**static** **int** TheSelectSwitch = 0;

**static** **int** TheLeftSwitch = 0;

**static** **int** TheRightSwitch = 0;

//#endif

**static** **int** TheUpSwitch = 0;

**static** **int** TheDownSwitch = 0;

**union**

{

**float** bufTemp;

**unsigned** **long** ul;

**unsigned** **char** uc[32];

} tempUn;

**union**

{

**float** bufAccel[3];

**unsigned** **long** ul;

**unsigned** **char** uc[32];

} accelUn;

**union**

{

**float** bufGyro[3];

**unsigned** **long** ul;

**unsigned** **char** uc[32];

} gyroUn;

**union**

{

**float** bufQuaternion[4];

**unsigned** **long** ul;

**unsigned** **char** uc[32];

} quatUn;

**enum**

{

*TEMPERATURE*,

*RESET*,

*QUATERNION*,

*ACCEL*,

*GYRO*,

} eState;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

SwitchOffNode - SWITCH OFF ALL NODE OUTPUTS

Function is called after Node startup, Communication reset or after

NMT\_OPERATIONAL (this or monitored nodes) was lost.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **SwitchOffNode**(**void**)

{

**#ifdef** CO\_IO\_DIGITAL\_OUTPUTS

//CHANGE THIS LINE -> WRITE 0 TO ALL PORTS !!!

ODE\_Write\_Digital\_Output.DWORD[0] = 0;

**#endif**

**#ifdef** CO\_IO\_ANALOG\_OUTPUTS

//CHANGE THIS LINE -> WRITE 0 TO ALL PORTS !!!

ODE\_Write\_Analog\_Output[0] = 0;

ODE\_Write\_Analog\_Output[1] = 0;

**#endif**

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_Init - USER INITIALIZATION OF NODE

Function is called after start of program.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_Init**(**void**)

{

ODE\_EEPROM.PowerOnCounter++;

**#ifdef** CO\_IO\_DIGITAL\_INPUTS

//CHANGE THIS LINE -> set ports as digital inputs

ODE\_Read\_Digital\_Input.DWORD[0] = 0;

**#endif**

**#ifdef** CO\_IO\_ANALOG\_INPUTS

//CHANGE THIS LINE -> set ports as analog inputs

ODE\_Read\_Analog\_Input[0] = 0;

ODE\_Read\_Analog\_Input[1] = 0;

ODE\_Read\_Analog\_Input[2] = 0;

ODE\_Read\_Analog\_Input[3] = 0;

ODE\_Read\_Analog\_Input[4] = 0;

ODE\_Read\_Analog\_Input[5] = 0;

ODE\_Read\_Analog\_Input[6] = 0;

ODE\_Read\_Analog\_Input[7] = 0;

**#endif**

**#ifdef** CO\_IO\_DIGITAL\_OUTPUTS

//CHANGE THIS LINE -> set ports as digital outputs

**#endif**

**#ifdef** CO\_IO\_ANALOG\_OUTPUTS

//CHANGE THIS LINE -> set ports as analog outputs

**#endif**

SwitchOffNode();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_Remove - USER EXECUTION ON EXIT OF PROGRAM

Function is called before end of program. Not generally used.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_Remove**(**void**)

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_ResetComm - USER RESET COMMUNICATION

Function is called after start of program and after CANopen NMT command:

\b Reset

\b Communication.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_ResetComm**(**void**)

{

SwitchOffNode();

}

**char** cbuf[100];

**char**\* **ftoa**(**float** f)

{

**int** pos=0,ix,dp,num;

**if** ((f == +INFINITY) || (f == -INFINITY))

f = 0;

**if** (f<0)

{

cbuf[pos++]='-';

f = -f;

}

dp=0;

**while** (f>=10.0)

{

f=f/10.0;

dp++;

}

**for** (ix=1;ix<8;ix++)

{

num = (**int**)f;

f=f-num;

**if** (num>9)

cbuf[pos++]='#';

**else**

cbuf[pos++]='0'+num;

**if** (dp==0) cbuf[pos++]='.';

f=f\*10.0;

dp--;

}

**return** cbuf;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_ProcessMain - USER PROCESS MAINLINE

This function is cyclically called from main(). It is a non blocking function.

It is asynchronous. This is for longer and time consuming code.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_ProcessMain**(**void**)

{

**static** **int** sav\_up\_sw = 0;

**static** **int** sav\_down\_sw = 0;

//#ifdef CAN\_DEVICE\_LM3S8962

**static** **int** sav\_select\_sw = 0;

**static** **int** sav\_left\_sw = 0;

**static** **int** sav\_right\_sw = 0;

**float** \* buf;

**static** **unsigned** **char** state = 99;

//#endif // CAN\_DEVICE\_LM3S8962

//#ifdef CAN\_DEVICE\_LM3S8962

**if** (sav\_select\_sw != TheSelectSwitch)

{

sav\_select\_sw = TheSelectSwitch;

**if** (sav\_select\_sw)

{

/\*#ifdef UART\_DEBUG

ShowStack();

#endif

\*/

}

}

**if** (sav\_left\_sw != TheLeftSwitch)

{

sav\_left\_sw = TheLeftSwitch;

**if** (sav\_left\_sw)

{

**#ifdef** UART\_DEBUG

**ShowCANItems**();

**#endif**

}

}

**if** (sav\_right\_sw != TheRightSwitch)

{

sav\_right\_sw = TheRightSwitch;

**if** (sav\_right\_sw)

{

**#ifdef** UART\_DEBUG

**ShowState**();

**#endif**

}

}

//#endif // CAN\_DEVICE\_LM3S8962

**if** (sav\_up\_sw != TheUpSwitch)

{

sav\_up\_sw = TheUpSwitch;

**if** (sav\_up\_sw)

{

**#ifdef** CO\_IO\_DIGITAL\_INPUTS

++ODE\_Read\_Digital\_Input.BYTE[0];

**#endif**

**#ifdef** UART\_DEBUG

// ShowObjectDictionary();

**#endif**

}

}

**if** (sav\_down\_sw != TheDownSwitch)

{

sav\_down\_sw = TheDownSwitch;

**if** (sav\_down\_sw)

{

**#ifdef** CO\_IO\_DIGITAL\_INPUTS

--ODE\_Read\_Digital\_Input.BYTE[0];

**#endif**

}

}

// display variables

//#ifdef CAN\_DEVICE\_LM3S8962

RIT128x96x4Enable(1000000);

buf = main\_mpu6050();

accelUn.bufAccel[0] = buf[0];

accelUn.bufAccel[1] = buf[1];

accelUn.bufAccel[2] = buf[2];

gyroUn.bufGyro[0] = buf[4];

gyroUn.bufGyro[1] = buf[5];

gyroUn.bufGyro[2] = buf[6];

quatUn.bufQuaternion[0] = buf[7];

quatUn.bufQuaternion[1] = buf[8];

quatUn.bufQuaternion[2] = buf[9];

quatUn.bufQuaternion[3] = buf[10];

tempUn.bufTemp = buf[3];

**if** (tempUn.bufTemp != ODE\_Read\_Digital\_Input.DWORD[0])

{

CO\_DISABLE\_TMR();

RIT128x96x4StringDraw(ftoa(accelUn.bufAccel[0]), 50, 0, 15);

RIT128x96x4StringDraw(ftoa(accelUn.bufAccel[1]), 50, 8, 15);

RIT128x96x4StringDraw(ftoa(accelUn.bufAccel[2]), 50, 16, 15);

//SysCtlDelay(100);

RIT128x96x4StringDraw(ftoa(gyroUn.bufGyro[0]), 50, 24, 15);

RIT128x96x4StringDraw(ftoa(gyroUn.bufGyro[1]), 50, 32, 15);

RIT128x96x4StringDraw(ftoa(gyroUn.bufGyro[2]), 50, 40, 15);

RIT128x96x4StringDraw(ftoa(quatUn.bufQuaternion[0]), 50, 48, 15);

RIT128x96x4StringDraw(ftoa(quatUn.bufQuaternion[1]), 50, 56, 15);

RIT128x96x4StringDraw(ftoa(quatUn.bufQuaternion[2]), 50, 64, 15);

RIT128x96x4StringDraw(ftoa(quatUn.bufQuaternion[3]), 50, 72, 15);

RIT128x96x4StringDraw(ftoa(tempUn.bufTemp), 50, 80, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("complete data receive from MPU6050\n");

**UARTprintf**("Accel x = %s", ftoa(accelUn.bufAccel[0]));

**UARTprintf**(", y = %s", ftoa(accelUn.bufAccel[1]));

**UARTprintf**(", z = %s\n", ftoa(accelUn.bufAccel[2]));

**UARTprintf**("Gyro x = %s", ftoa(gyroUn.bufGyro[0]));

**UARTprintf**(", y = %s", ftoa(gyroUn.bufGyro[1]));

**UARTprintf**(", z = %s\n", ftoa(gyroUn.bufGyro[2]));

**UARTprintf**("Temperature x = %s\n", ftoa(tempUn.bufTemp));

**#endif**

CO\_ENABLE\_TMR();

}

**if** (state != CO\_NMToperatingState)

{

**#ifdef** UART\_DEBUG

**UARTprintf**("CO\_NMToperatingState = ");

**#endif**

**switch** (CO\_NMToperatingState)

{

**case** NMT\_INITIALIZING:

RIT128x96x4StringDraw("INITIALIZING ", 40, 88, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_INITIALIZING");

**#endif**

**break**;

**case** NMT\_PRE\_OPERATIONAL:

RIT128x96x4StringDraw("PRE-OPERATIONAL", 40, 88, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_PRE\_OPERATIONAL");

**#endif**

**break**;

**case** NMT\_OPERATIONAL:

RIT128x96x4StringDraw("OPERATIONAL ", 40, 88, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_OPERATIONAL");

**#endif**

**break**;

**case** NMT\_STOPPED:

RIT128x96x4StringDraw("STOPPED ", 40, 88, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_STOPPED");

**#endif**

**break**;

**default**:

**#ifdef** UART\_DEBUG

**UARTprintf**("0x%02x", CO\_NMToperatingState);

**#endif**

**break**;

}

**#ifdef** UART\_DEBUG

**UARTprintf**("\n");

**#endif**

state = CO\_NMToperatingState;

}

//#endif // CAN\_DEVICE\_LM3S8962

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_Process1msIsr - 1 ms USER TIMER FUNCTION

Function is executed every 1 ms. It is deterministic and has priority over

mainline functions.

Function of the following code (each cycle):

- Read from hardware

- If Operational state:

- If all Monitored nodes are Operational:

- Copy all Received PDOs to their mapped location

- Prepare all Transmit PDOs from their mapped location (and send them in case Change Of State)

- If Operational state of this or monitored nodes is lost, node outputs are switched off.

- Write to hardware

If TPDO is configured for Change of State transmission (Transmission\_type >= 254), then consider next

situation: On network startup one node is started first and send PDO of curent state, which will not

change soon. Another node is started later and missed PDO from first node??? Solution might be

Event timer.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_Process1msIsr**(**void**)

{

**static** **unsigned** **char** LastStateOperationalGradePrev = 0;

**unsigned** **char** LastStateOperationalGrade = 0;

//#ifdef CAN\_DEVICE\_LM3S8962

TheSelectSwitch = SELECT\_SW\_PRESSED();

TheLeftSwitch = LEFT\_SW\_PRESSED();

TheRightSwitch = RIGHT\_SW\_PRESSED();

//#endif

TheUpSwitch = UP\_SW\_PRESSED();

TheDownSwitch = DOWN\_SW\_PRESSED();

// Read from Hardware -------------------------------------------------------

//CHANGE THIS LINE -> ODE\_Read\_Digital\_Input.BYTE[0] = port\_xxx

//CHANGE THIS LINE -> ODE\_Read\_Digital\_Input.BYTE[1] = port\_xxx

//CHANGE THIS LINE -> ODE\_Read\_Digital\_Input.BYTE[2] = port\_xxx

//CHANGE THIS LINE -> ODE\_Read\_Digital\_Input.BYTE[3] = port\_xxx

**#ifdef** CO\_IO\_ANALOG\_INPUTS

//CHANGE THIS LINE -> ODE\_Read\_Analog\_Input[0...7] = ...

**#endif**

// PDO Communication

**if** (CO\_NMToperatingState == NMT\_OPERATIONAL)

{

LastStateOperationalGrade++;

// verify operating state of monitored nodes

**#if** CO\_NO\_CONS\_HEARTBEAT > 0

**if** (CO\_HBcons\_AllMonitoredOperational == NMT\_OPERATIONAL)

{

**#endif**

LastStateOperationalGrade++;

// Read RPDOs -------------------------------------------------------------

// Following code realizes Static RPDO Mapping

**#if** CO\_NO\_RPDO > 0

**#ifdef** CO\_IO\_DIGITAL\_OUTPUTS

**if** (CO\_RPDO\_New(0))

{

CO\_DISABLE\_CANRX\_TMR();

ODE\_Write\_Digital\_Output.DWORD[0] = CO\_RPDO(0).DWORD[0];

CO\_RPDO\_New(0) = 0;

CO\_ENABLE\_CANRX\_TMR();

}

**#endif**

**#endif**

**#if** CO\_NO\_RPDO > 1

**#ifdef** CO\_IO\_ANALOG\_OUTPUTS

**if** (CO\_RPDO\_New(1))

{

CO\_DISABLE\_CANRX\_TMR();

ODE\_Write\_Analog\_Output[0] = CO\_RPDO(1).WORD[0];

ODE\_Write\_Analog\_Output[1] = CO\_RPDO(1).WORD[1];

CO\_RPDO\_New(1) = 0;

CO\_ENABLE\_CANRX\_TMR();

}

**#endif**

**#endif**

**#if** CO\_NO\_CONS\_HEARTBEAT > 0

} // end if(CO\_HBcons\_AllMonitoredOperational == NMT\_OPERATIONAL)

**#endif**

// Write TPDOs ------------------------------------------------------------

// Following code realizes Static TPDO Mapping

// Transmission is Synchronous or Change of State, depends on Transmission\_type.

// Inhibit timer and Periodic Event Timer can be used.

**#if** CO\_NO\_TPDO > 0

**#endif**

**#if** CO\_NO\_TPDO > 1

**#ifdef** CO\_IO\_DIGITAL\_INPUTS

**if** ((CO\_TPDO\_InhibitTimer[1] == 0) &&

(CO\_TPDO(1).DWORD[0] != ODE\_Read\_Digital\_Input.DWORD[0]))

{

CO\_TPDO(1).DWORD[0] = tempUn.ul;

**if** (ODE\_TPDO\_Parameter[1].Transmission\_type >= 254)

{

**CO\_TPDOsend**(1);

}

}

**#elif** defined CO\_IO\_ANALOG\_INPUTS

**if** ((CO\_TPDO\_InhibitTimer[1] == 0) && (

(CO\_TPDO(1).WORD[0] != ODE\_Read\_Analog\_Input[0]) ||

(CO\_TPDO(1).WORD[1] != ODE\_Read\_Analog\_Input[1]) ||

(CO\_TPDO(1).WORD[2] != ODE\_Read\_Analog\_Input[2]) ||

(CO\_TPDO(1).WORD[3] != ODE\_Read\_Analog\_Input[3])))

{

CO\_TPDO(1).WORD[0] = ODE\_Read\_Analog\_Input[0];

CO\_TPDO(1).WORD[1] = ODE\_Read\_Analog\_Input[1];

CO\_TPDO(1).WORD[2] = ODE\_Read\_Analog\_Input[2];

CO\_TPDO(1).WORD[3] = ODE\_Read\_Analog\_Input[3];

**if** (ODE\_TPDO\_Parameter[1].Transmission\_type >= 254)

{

CO\_TPDOsend(1);

}

}

**#endif**

**#endif**

} // end if (CO\_NMToperatingState == NMT\_OPERATIONAL)

**if** (LastStateOperationalGrade < LastStateOperationalGradePrev) // NMT\_OPERATIONAL (this or monitored nodes) was just lost

{

SwitchOffNode();

}

LastStateOperationalGradePrev = LastStateOperationalGrade;

}

**Модуль БЧЭ – mpu6050\_sensors.c**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** "sensorlib/hw\_mpu6050.h"

**#include** "sensorlib/hw\_ak8975.h"

**#include** "sensorlib/i2cm\_drv.h"

**#include** "sensorlib/ak8975.h"

**#include** "sensorlib/mpu6050.h"

**#include** "sensorlib/comp\_dcm.h"

**#include** "inc/hw\_ints.h"

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_sysctl.h"

**#include** "inc/hw\_can.h"

**#include** "inc/hw\_i2c.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/i2c.h"

**#include** "driverlib/rom.h"

**#include** "driverlib/rom\_map.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/uart.h"

**#include** "utils/uartstdio.h"

**#include** "driverlib/pin\_map.h"

**#define** SLAVE\_ADDRESS 0x68

**#define** AMBIENT\_TEMPERATURE 0x3B

**#define** DEVICE\_ID\_REGISTER 0x3B

**#define** MANUFACTURE\_ID\_REGISTER 0x07

**#define** GPIO\_PB2\_I2C0SCL 0x00010801

**#define** GPIO\_PB3\_I2C0SDA 0x00010C01

**float** \*pfEulers, \*pfQuaternion;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Global Instance structure to manage the DCM state.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

tCompDCM g\_sCompDCMInst;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Global instance structure for the ISL29023 sensor driver.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

tMPU6050 g\_sMPU6050Inst;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The factors used to convert the acceleration readings from the MPU6050 into

// floating point values in meters per second squared.

//

// Values are obtained by taking the g conversion factors from the data sheet

// and multiplying by 9.81 (1 g = 9.81 m/s^2).

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**static** **const** **float** g\_fMPU6050AccelFactors[] =

{

0.00059875, // Range = +/- 2 g (16384 lsb/g)

0.00119751, // Range = +/- 4 g (8192 lsb/g)

0.00239502, // Range = +/- 8 g (4096 lsb/g)

0.00479004 // Range = +/- 16 g (2048 lsb/g)

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The factors used to convert the acceleration readings from the MPU6050 into

// floating point values in radians per second.

//

// Values are obtained by taking the degree per second conversion factors

// from the data sheet and then converting to radians per sec (1 degree =

// 0.0174532925 radians).

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**static** **const** **float** g\_fMPU6050GyroFactors[] =

{

1.3323124e-4f, // Range = +/- 250 dps (131.0 LSBs/DPS)

2.6646248e-4f, // Range = +/- 500 dps (65.5 LSBs/DPS)

5.3211258e-4f, // Range = +/- 1000 dps (32.8 LSBs/DPS)

0.0010642252f // Range = +/- 2000 dps (16.4 LSBs/DPS)

};

uint32\_t ui32SysClock\_2;

**int** a = 0;

uint32\_t pui32DataTx[3];

uint32\_t pui32DataRx[3];

**float** \* X;

**float** sBufAcc[3];

**float** sBufTemp;

**float** sBufGyro[3];

**float** \* spBufQuaternion;

uint16\_t UpperByte\_X = 0;

uint16\_t LowerByte\_X = 0;

uint16\_t UpperByte\_Y = 0;

uint16\_t LowerByte\_Y = 0;

uint16\_t UpperByte\_Z = 0;

uint16\_t LowerByte\_Z = 0;

**void** **I2C\_Init**()

{

//Разрешить работу I2C

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_I2C0);

//Перезапуск I2C

**SysCtlPeripheralReset**(SYSCTL\_PERIPH\_I2C0);

//Включение порта B

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOB);

// Конфигурирование пинов порта B для работы в режие I2C

**GPIOPinConfigure**(GPIO\_PB2\_I2C0SCL);

**GPIOPinConfigure**(GPIO\_PB3\_I2C0SDA);

**GPIOPinTypeI2CSCL**(GPIO\_PORTB\_BASE, GPIO\_PIN\_2);

**GPIOPinTypeI2C**(GPIO\_PORTB\_BASE, GPIO\_PIN\_3);

// Конфигурирование Мастер-узла

**I2CMasterInitExpClk**(I2C0\_MASTER\_BASE,**SysCtlClockGet**(), false);

}

**void** **xAxis\_R**(uint8\_t ambient\_temp\_reg)

{

**I2CMasterDataPut**(I2C0\_MASTER\_BASE, ambient\_temp\_reg);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

}

**float** \* **xAxis**()

{

**static** **float** sRes[7];

**float** fFactor;

int16\_t rtemp\_MPU;

//

// Get the conversion factor for the current data format.

//

fFactor = g\_fMPU6050AccelFactors[0];

**I2CMasterSlaveAddrSet**(I2C0\_MASTER\_BASE, SLAVE\_ADDRESS, true);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_START);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

UpperByte\_X = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

LowerByte\_X = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

UpperByte\_Y = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

LowerByte\_Y = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

UpperByte\_Z = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

LowerByte\_Z = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

//

// Convert the Accelerometer values into rad/sec

//

sRes[0] = ((**float**)(**short**)((UpperByte\_X<<8) | LowerByte\_X) \* fFactor);

sRes[1] = ((**float**)(**short**)((UpperByte\_Y<<8) | LowerByte\_Y) \* fFactor);

sRes[2] = ((**float**)(**short**)((UpperByte\_Z<<8) | LowerByte\_Z) \* fFactor);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

UpperByte\_X = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

LowerByte\_X = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

rtemp\_MPU = (((int16\_t) UpperByte\_X) << 8) | LowerByte\_X;

sRes[3] = (**float**)(rtemp\_MPU)/340+36.53;

//

// Get the conversion factor for the current data format.

//

fFactor = g\_fMPU6050GyroFactors[0];

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

UpperByte\_X = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

LowerByte\_X = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

UpperByte\_Y = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

LowerByte\_Y = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

UpperByte\_Z = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_FINISH);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE));

LowerByte\_Z = **I2CMasterDataGet**(I2C0\_MASTER\_BASE);

sRes[4] = ((**float**)(**short**)((UpperByte\_X<<8) | LowerByte\_X) \* fFactor);

sRes[5] = ((**float**)(**short**)((UpperByte\_Y<<8) | LowerByte\_Y) \* fFactor);

sRes[6] = ((**float**)(**short**)((UpperByte\_Z<<8) | LowerByte\_Z) \* fFactor);

**return** sRes;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Called by the NVIC as a result of GPIO port M interrupt event. For this

// application GPIO port M pin 3 is the interrupt line for the MPU9150

//

// For BoosterPack 2 Interface use Port M pin 7.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void**

**GPIOPortBIntHandler**(**void**)

{

**unsigned** **long** ulStatus;

//

// Get the status flags to see which pin(s) caused the interrupt.

//

ulStatus = **GPIOPinIntStatus**(GPIO\_PORTB\_BASE, true);

//

// Clear all the pin interrupts that are set

//

**GPIOPinIntClear**(GPIO\_PORTB\_BASE, ulStatus);

//

// Check if this is an interrupt on the MPU6050 interrupt line.

//

// For BoosterPack 2 use Pin 7 instead.

//

**if**(ulStatus & GPIO\_PIN\_7)

{

//

// MPU6050 Data is ready for retrieval and processing.

//

//MPU6050DataRead(&g\_sMPU6050Inst, MPU6050AppCallback, &g\_sMPU6050Inst);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Called by the NVIC as a result of I2C0 Interrupt. I2C0 is the I2C connection

// to the MPU6050.

//

//

// For BoosterPack 2 interface change this to the I2C8 interrupt location in

// the vector table in the startup file.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void**

**MPU6050I2CIntHandler**(**void**)

{

//

// Pass through to the I2CM interrupt handler provided by sensor library.

// This is required to be at application level so that I2CMIntHandler can

// receive the instance structure pointer as an argument.

//

//I2CMIntHandler(&g\_sI2CInst);

}

**void** **init\_mpu6050**(**void**)

{

ui32SysClock\_2 = **SysCtlClockGet**();

I2C\_Init();

//SysCtlDelay(500);

pui32DataTx[0] = 0x6B; //MPU6050 PWR\_MGMT\_1 Register

pui32DataTx[1] = 0x00; //Set to 0x00 the register PWR\_MGMT\_1

**I2CMasterSlaveAddrSet**(I2C0\_MASTER\_BASE, SLAVE\_ADDRESS, false);

**I2CMasterDataPut**(I2C0\_MASTER\_BASE, pui32DataTx[0] );

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE)){}

pui32DataRx[0]=**I2CSlaveDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterDataPut**(I2C0\_MASTER\_BASE, pui32DataTx[1] );

**I2CMasterControl**(I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_CONT);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE)){}

pui32DataRx[1]=**I2CSlaveDataGet**(I2C0\_MASTER\_BASE);

**I2CMasterControl**( I2C0\_MASTER\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH );//Finish the Burst sending

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE)) {}

pui32DataRx[2]=**I2CSlaveDataGet**(I2C0\_MASTER\_BASE);

//

// Initialize the DCM system. 50 hz sample rate.

// accel weight = .2, gyro weight = .8, mag weight = .2

//

CompDCMInit(&g\_sCompDCMInst, 1.0f / 50.0f, 0.2f, 0.6f, 0.2f);

}

**float** \* **main\_mpu6050**(**void**)

{

**I2CMasterSlaveAddrSet**(I2C0\_MASTER\_BASE, SLAVE\_ADDRESS, false);

xAxis\_R(0x3B);

**while**(**I2CMasterBusy**(I2C0\_MASTER\_BASE)){}

X = xAxis();

sBufAcc[0] = X[0];

sBufAcc[1] = X[1];

sBufAcc[2] = X[2];

sBufGyro[0] = X[4];

sBufGyro[1] = X[5];

sBufGyro[2] = X[6];

sBufTemp = X[3];

CompDCMAccelUpdate(&g\_sCompDCMInst, sBufAcc[0], sBufAcc[1],

sBufAcc[2]);

CompDCMGyroUpdate(&g\_sCompDCMInst, sBufGyro[0], sBufGyro[1],

sBufGyro[2]);

CompDCMStart(&g\_sCompDCMInst);

//

// Get Quaternions.

//

spBufQuaternion = CompDCMComputeQuaternion(&g\_sCompDCMInst, pfQuaternion);

X[7] = spBufQuaternion[0];

X[8] = spBufQuaternion[1];

X[9] = spBufQuaternion[2];

X[10] = spBufQuaternion[3];

**return** X;

}

**Модуль БУ – main.c**

// CAN board is specified in Makefile

**#include** "EK-LM3S8962-HW-Main.h"

**#include** "utils/uartstdio.h"

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_ints.h"

**#include** "inc/hw\_sysctl.h"

**#include** "inc/hw\_can.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/can.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/systick.h"

**#include** "drivers/rit128x96x4.h"

**#include** "CO\_driver.h"

**#include** "CANopen.h"

**#include** "debug\_info.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Interrupt priority definitions. The top 3 bits of these values are

// significant with smaller values indicating higher priority interrupts.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#define** SET\_INT\_PRIORITY(i) (i << 5)

**#define** CAN\_INT\_PRIORITY SET\_INT\_PRIORITY(0) // highest priority

**#define** TIMER0B\_INT\_PRIORITY SET\_INT\_PRIORITY(1) // lower priority

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Counter to count the number of timer interrupts that have been called.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**volatile** **unsigned** **long** g\_ulCounter = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! This function sets up Timer0B to be used as a 1ms CANopen processing timer.

//!

//! \attention

//! The SysTick interrupt \b \*cannot\* be used for the 1ms timer because

//! the CANopenNode code disables the timer interrupts around critical

//! regions. If a SysTick interrupt is disabled when its next periodic

//! interrupt is due, the interrupt will be lost, not pended!

//! However, a standard timer interrupt will only be delayed while its

//! interrupt is disabled, and will occur as soon as its interrupt

//! is reenabled.

//!

//! \return None.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**static** **void** **InitTimer0B**(**void**)

{

//

// The Timer0 peripheral must be enabled for use.

//

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

//

// Configure Timer0B as a 16-bit periodic timer.

//

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_16\_BIT\_PAIR | TIMER\_CFG\_B\_PERIODIC);

//

// Set the Timer0B load value to 1ms.

//

**TimerLoadSet**(TIMER0\_BASE, TIMER\_B, **SysCtlClockGet**() / 1000);

//

// Clear any pending interrupt flag.

//

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMB\_TIMEOUT);

//

// Configure the Timer0B interrupt for timer timeout.

//

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMB\_TIMEOUT);

//

// Enable the Timer0B interrupt on the processor (NVIC).

//

**IntEnable**(INT\_TIMER0B);

//

// Enable Timer0B.

//

**TimerEnable**(TIMER0\_BASE, TIMER\_B);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! The interrupt handler for the Timer0B interrupt.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** **Timer0BIntHandler**(**void**)

{

//

// Clear the timer interrupt flag.

//

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMB\_TIMEOUT);

//

// Update the Timer0B interrupt counter.

//

g\_ulCounter++;

**CO\_Timer1msIsr**();

**if** ((g\_ulCounter & 0x0000001f) == 0)

{

TOGGLE\_STATUS\_LED();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Configure the CAN and enter a loop to implement CANopen processing.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int** **main**(**void**)

{

//

// If running on Rev A2 silicon, turn the LDO voltage up to 2.75V. This is

// a workaround to allow the PLL to operate reliably.

//

**if** (REVISION\_IS\_A2) // needs #include "inc/hw\_sysctl.h"

{

**SysCtlLDOSet**(SYSCTL\_LDO\_2\_75V);

}

//

// Set the clocking to run from the PLL.

//

**SysCtlClockSet**(CO\_OSCILLATOR\_FREQ | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_8MHZ);

//

// Disable interrupts to the processor.

//

**IntMasterDisable**();

**#ifdef** CAN\_DEVICE\_LM3S2110

//

// Enable the pull-ups on the JTAG signals.

// (This appears to be needed for reliable operation!?)

//

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_JTAG\_PINS);

GPIOPadConfigSet(JTAG\_BASE,

JTAG\_TCK | JTAG\_TMS | JTAG\_TDI | JTAG\_TDO,

GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

**#endif** // CAN\_DEVICE\_LM3S2110

**#ifdef** UART\_DEBUG

//

// Set up the serial console to use for displaying messages.

//

**InitConsole**();

**UARTprintf**("\n\nCANopenNode Tutorial Command\n");

**UARTprintf**("============================\n");

**#endif** // UART\_DEBUG

//

// Initialize the OLED display.

//

RIT128x96x4Init(1000000);

//

// Hello!

//

RIT128x96x4StringDraw(" ", 8, 0, 0); // remove bogus pixel on Board #1

RIT128x96x4StringDraw("CANopenNode", 20, 20, 15);

RIT128x96x4StringDraw("Tutorial Command", 20, 32, 15);

RIT128x96x4StringDraw("Temperature = ", 0, 48, 15);

RIT128x96x4StringDraw("Cooler: ", 0, 58, 15);

RIT128x96x4StringDraw("Heater: ", 0, 66, 15);

RIT128x96x4StringDraw("State: ", 0, 76, 15);

SELECT\_SW\_INIT();

LEFT\_SW\_INIT();

RIGHT\_SW\_INIT();

UP\_SW\_INIT();

DOWN\_SW\_INIT();

STATUS\_LED\_INIT();

// setup Timer0B interrupt

InitTimer0B();

**CO\_DriverInit**(); // CANopenNode driver init

**User\_Init**(); // User init

**CO\_ResetComm**(); // Reset CAN communication

//

// Set the priorities of the interrupts used by the application.

//

**IntPrioritySet**(INT\_CAN0, CAN\_INT\_PRIORITY);

**IntPrioritySet**(INT\_TIMER0B, TIMER0B\_INT\_PRIORITY);

//

// Enable interrupts to the processor.

//

**IntMasterEnable**();

**while** (1)

{

**CO\_ProcessMain**();

**User\_ProcessMain**();

}

}

**Модуль БУ – BlockOfControl.c**

// CAN board is specified in Makefile

**#include** <stdlib.h>

**#include** "EK-LM3S8962-HW-Main.h"

**#include** "utils/uartstdio.h"

**#include** "utils/ustdlib.h"

**#include** "drivers/rit128x96x4.h"

**#include** "CANopen.h"

**#include** "debug\_info.h"

**#define** RemoteTemperature CO\_RPDO(0).DWORD[0]

**#define** NMT\_MASTER CO\_TXCAN\_USER

**#define** DigOut\_COOLER\_IND\_INIT() PC6\_PIN\_INIT()

**#define** DigOut\_COOLER\_IND\_ON() TURN\_ON\_PC6()

**#define** DigOut\_COOLER\_IND\_OFF() TURN\_OFF\_PC6()

**#define** DigOut\_HEATER\_IND\_INIT() PD3\_PIN\_INIT()

**#define** DigOut\_HEATER\_IND\_ON() TURN\_ON\_PD3()

**#define** DigOut\_HEATER\_IND\_OFF() TURN\_OFF\_PD3()

**#define** DigInp\_Button\_INIT() SELECT\_SW\_INIT()

**#define** DigInp\_Button() SELECT\_SW\_PRESSED()

**#define** DigOut\_COOLER\_IND(i) **if** (i) { DigOut\_COOLER\_IND\_ON(); } **else** DigOut\_COOLER\_IND\_OFF()

**#define** DigOut\_HEATER\_IND(i) **if** (i) { DigOut\_HEATER\_IND\_ON(); } **else** DigOut\_HEATER\_IND\_OFF()

**static** **int** TheSelectSwitch = 0;

**static** **int** TheLeftSwitch = 0;

**static** **int** TheRightSwitch = 0;

**static** **int** TheUpSwitch = 0;

**static** **int** TheDownSwitch = 0;

**static** **char** \* Temperature = 0;

**union**

{

**float** bufTemp;

**unsigned** **long** ul;

**unsigned** **char** uc[32];

} bufUn;

**char** cbuf[100];

**char**\* **ftoa**(**float** f)

{

**int** pos=0,ix,dp,num;

**if** (f<0)

{

cbuf[pos++]='-';

f = -f;

}

dp=0;

**while** (f>=10.0)

{

f=f/10.0;

dp++;

}

**for** (ix=1;ix<8;ix++)

{

num = (**int**)f;

f=f-num;

**if** (num>9)

cbuf[pos++]='#';

**else**

cbuf[pos++]='0'+num;

**if** (dp==0) cbuf[pos++]='.';

f=f\*10.0;

dp--;

}

**return** cbuf;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_Init - USER INITIALIZATION OF NODE

Function is called after start of program.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_Init**(**void**)

{

ODE\_EEPROM.PowerOnCounter++;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_Remove - USER EXECUTION ON EXIT OF PROGRAM

Function is called before end of program. Not generally used.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_Remove**(**void**)

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_ResetComm - USER RESET COMMUNICATION

Function is called after start of program and after CANopen NMT command:

\b Reset

\b Communication.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_ResetComm**(**void**)

{

// Prepare variables for custom made NMT master message.

// arguments to CO\_IDENT\_WRITE(CAN\_ID\_11bit, RTRbit/\*usually 0\*/)

CO\_TXCAN[NMT\_MASTER].Ident.WORD[0] = CO\_IDENT\_WRITE(0, 0);

CO\_TXCAN[NMT\_MASTER].NoOfBytes = 2;

CO\_TXCAN[NMT\_MASTER].NewMsg = 0;

CO\_TXCAN[NMT\_MASTER].Inhibit = 0;

}

**#ifdef** UART\_DEBUG

//! Output temperature info.

**void** **ShowTemps**(**void**)

{

**UARTprintf**("\nRemoteTemperature = %d\n", RemoteTemperature);

}

**#endif** // UART\_DEBUG

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_ProcessMain - USER PROCESS MAINLINE

This function is cyclically called from main(). It is a non blocking function.

It is asynchronous. This is for longer and time consuming code.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_ProcessMain**(**void**)

{

**static** **int** sav\_up\_sw = 0;

**static** **int** sav\_down\_sw = 0;

**static** **int** sav\_select\_sw = 0;

**static** **int** sav\_left\_sw = 0;

**static** **int** sav\_right\_sw = 0;

**char** tempString[5];

**static** uint16\_t saveTemp = 9999;

**static** **unsigned** **char** status = 99;

**static** **unsigned** **char** state = 99;

**if** (sav\_select\_sw != TheSelectSwitch)

{

sav\_select\_sw = TheSelectSwitch;

**if** (sav\_select\_sw)

{

/\*#ifdef UART\_DEBUG

ShowStack();

#endif

\*/

}

}

**if** (sav\_left\_sw != TheLeftSwitch)

{

sav\_left\_sw = TheLeftSwitch;

**if** (sav\_left\_sw)

{

**#ifdef** UART\_DEBUG

**ShowCANItems**();

**#endif**

}

}

**if** (sav\_right\_sw != TheRightSwitch)

{

sav\_right\_sw = TheRightSwitch;

**if** (sav\_right\_sw)

{

**#ifdef** UART\_DEBUG

**ShowState**();

**#endif**

}

}

**if** (sav\_up\_sw != TheUpSwitch)

{

sav\_up\_sw = TheUpSwitch;

**if** (sav\_up\_sw)

{

**#ifdef** UART\_DEBUG

// ShowObjectDictionary();

**#endif**

}

}

**if** (sav\_down\_sw != TheDownSwitch)

{

sav\_down\_sw = TheDownSwitch;

**if** (sav\_down\_sw)

{

**#ifdef** UART\_DEBUG

ShowTemps();

**#endif**

}

}

// display variables

//DigOut\_COOLER\_IND(STATUS\_COOLER);

//DigOut\_HEATER\_IND(STATUS\_HEATER);

**if** (saveTemp != bufUn.bufTemp)

{

CO\_DISABLE\_TMR();

saveTemp = bufUn.bufTemp;

CO\_ENABLE\_TMR();

RIT128x96x4StringDraw(ftoa(bufUn.bufTemp), 86, 48, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("Temperature = %s\n", tempString);

**#endif**

}

**if** (state != CO\_NMToperatingState)

{

**#ifdef** UART\_DEBUG

**UARTprintf**("CO\_NMToperatingState = ");

**#endif**

**switch** (CO\_NMToperatingState)

{

**case** NMT\_INITIALIZING:

RIT128x96x4StringDraw("INITIALIZING ", 40, 76, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_INITIALIZING");

**#endif**

**break**;

**case** NMT\_PRE\_OPERATIONAL:

RIT128x96x4StringDraw("PRE-OPERATIONAL", 40, 76, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_PRE\_OPERATIONAL");

**#endif**

**break**;

**case** NMT\_OPERATIONAL:

RIT128x96x4StringDraw("OPERATIONAL ", 40, 76, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_OPERATIONAL");

**#endif**

**break**;

**case** NMT\_STOPPED:

RIT128x96x4StringDraw("STOPPED ", 40, 76, 15);

**#ifdef** UART\_DEBUG

**UARTprintf**("NMT\_STOPPED");

**#endif**

**break**;

**default**:

**#ifdef** UART\_DEBUG

**UARTprintf**("0x%02x", CO\_NMToperatingState);

**#endif**

**break**;

}

**#ifdef** UART\_DEBUG

**UARTprintf**("\n");

**#endif**

state = CO\_NMToperatingState;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\*

User\_Process1msIsr - 1 ms USER TIMER FUNCTION

Function is executed every 1 ms. It is deterministic and has priority over

mainline functions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **User\_Process1msIsr**(**void**)

{

**static** **unsigned** **int** DeboucigTimer = 0;

TheSelectSwitch = SELECT\_SW\_PRESSED();

TheLeftSwitch = LEFT\_SW\_PRESSED();

TheRightSwitch = RIGHT\_SW\_PRESSED();

TheUpSwitch = UP\_SW\_PRESSED();

TheDownSwitch = DOWN\_SW\_PRESSED();

**if** (DigInp\_Button())

{

DeboucigTimer++;

}

**else**

{

DeboucigTimer = 0;

}

**switch** (DeboucigTimer)

{

**case** 1000: // button is pressed for one seccond

**if** (CO\_NMToperatingState == NMT\_OPERATIONAL)

{

CO\_TXCAN[NMT\_MASTER].Data.BYTE[0] = NMT\_ENTER\_PRE\_OPERATIONAL;

CO\_NMToperatingState = NMT\_PRE\_OPERATIONAL;

}

**else**

{

CO\_TXCAN[NMT\_MASTER].Data.BYTE[0] = NMT\_ENTER\_OPERATIONAL;

CO\_NMToperatingState = NMT\_OPERATIONAL;

}

CO\_TXCAN[NMT\_MASTER].Data.BYTE[1] = 0; // all nodes

**CO\_TXCANsend**(NMT\_MASTER);

**break**;

**case** 5000: // button is pressed for 5 seconds

// reset all remote nodes

CO\_TXCAN[NMT\_MASTER].Data.BYTE[0] = NMT\_RESET\_NODE;

CO\_TXCAN[NMT\_MASTER].Data.BYTE[1] = 0; // all nodes

**CO\_TXCANsend**(NMT\_MASTER);

**break**;

**case** 5010: // button is pressed for 5 seconds + 10 milliseconds

CO\_Reset(); // reset this node

**break**;

}

bufUn.ul = RemoteTemperature;

}

**Приложение Г**

**Копии листов графической части**

Листов 6

В графическую часть дипломного проекта входят:

1. Постановка задачи , структурная схема системы цифрового обмена блока чувствительных элементов гидролокационного комплекса (лист 1).
2. Функциональная схема элементов системы (лист 2).
3. Структурная схема БЧЭ , состав и структура пакетов данны (лист 3).
4. Схемы алгоритмов (лист 4).
5. Диаграмма взаимодействия пользователя с системой и диаграммы компоновки (лист 5).
6. Конфигурирование объектного словаря и тестирование протокола передали данных CANOpen (лист 6).