Проектирование модульной структуры нижнего уровня информационной системы учета готовой продукции

М. В. Паринов, email: parmax@mail.ru [1]
К. В. Костин, email: phantasyftw@bk.ru [1]
Н. А. Чернышов, email: chernyshovkolya1994@mail.ru [1]
П. В. Шуваев, email: asutp3@molvest.ru[2]

[1] Воронежский государственный технический университет
[2] Молочный комбинат «Воронежский»

Аннотация. *В данной работе рассматриваются особенности проектирования нижнего уровня аппаратно-программного комплекса учета готовой продукции. Рассматривается специфика разработки программного обеспечения для современных микроконтроллеров. Обсуждаются принципы программной реализации, предлагается деление на модули и реализация связей между ними. Поднимаются вопросы расширения и масштабирования системы*

Ключевые слова: *Информационная система, микроконтроллер, модульная структура, операционная система реального времени, задача, объект.*

Введение

Особенностью рассматриваемой в статье задачей является ее реализация на современном мощном микроконтроллере. Подход программирования под подобные решения достаточно близко к типовым подходам разработки программного обеспечения для настольных компьютеров и мобильных устройств. При этом он значительно отличается от классических методов программирования микроконтроллеров. Однако наличие аппаратных устройств и ограниченные ресурсы требуют специального подхода.

В данной работе рассмотрено проектирование достаточно объемной информационной подсистемы с учетом изложенных выше факторов. Выполнен анализ различных подходов. Выбран наиболее перспективный с позиции авторов, описана его реализация.

1. Выбор и обоснование методики реализации

На основании технического задания и описания распределения процессов во времени, представленном диаграммой последовательности, возможно определить модульную структуру

проектируемой подсистемы. Данная структура жестко привязана к инструментальным средствам реализации продукта и поэтому не может быть инвариантной. Основная проблема заключается в использовании специфической аппаратной платформы.

Выбранный микроконтроллер семейства STM32 [1] позволяет использовать объектно-ориентированный подход программирования. Однако в большинстве случаев в процессе разработки для подобных систем применяется процедурный подход. Это связано со значительной экономией ресурсов. В частности, использования объектов в младших моделях может привести к нехватке встроенной оперативной памяти.

Процедурное программирование [2] имеет ряд классических общеизвестных недостатков для реализации достаточно сложных систем. Они известны и не требуют анализа в данном материале. Однако специфика микроконтроллерной техники вносит дополнительные проблемы реализации достаточно сложных систем. Они следуют из привязки аппаратно-программной платформы к достаточно жесткому реальному времени.

Обеспечение работы большинства аппаратных ресурсов микроконтроллера требует соблюдение четких временных интервалов. Задача достаточно легко реализуется при необходимости реализации отдельных процессов реального времени, выполняемых последовательно по жесткому алгоритму. Однако для сложных информационных систем типовой задачей является параллельное их выполнение, а также нечеткий сценарий вызова процессов.

Управление прерываниями и тщательное планирование таймингов позволяют решить данную задачу. Однако подобный подход значительно увеличивает время разработки, а в ряде случаев даже не позволяет достичь удовлетворительных результатов без привлечения специалистов высочайшего класса.

Реализация сложных классических процедурных микроконтроллерных проектов с наличием значительного фрагмента кода реального времени в настоящее время типична только для отдельных сфер хозяйственной деятельности [3-5]. Например, для авиакосмической отрасли. Обычно в типовых проектах применяют методы декомпозиции, которые позволяют разделить сложную систему на подсистемы. Задача может быть разделена между несколькими микроконтроллерами, а также реализована с применением операционной системы, где каждая из отдельных задач является отдельным инкапсулированным процессом.

Использование избыточного числа ядер микроконтроллера является достаточно редким решением. Оно значительно удорожает

аппаратную часть и требует сложной реализации между отдельными объектами, что иногда недопустимо из-за недостаточной скорости обмена.

Операционные системы реального времени (ОСРВ) на микроконтроллерах в настоящее время крайне популярны. Они позволяют решить поставленную задачу с наименьшими трудовыми и финансовыми затратами. К недостаткам ОСРВ следует отнести некоторую сложность реализации программного кода, что требует определенного набора знаний и навыков программиста.

Следующий недостаток, который ряд специалистов считает критическим – это потеря полного контроля жесткого реального времени и возможность возникновения сбоев из-за использования в проекте сторонних программных продуктов (операционная система, драйвера и так далее). Данный недостаток является спорным. В частности, некоторые компании разрешают использование ОСРВ в гражданской авиации в составе критичных компонентов. Однако большинство современных проектов в области авиакосмической техники стараются избегать использования операционных систем.

Необходимо отметить, что разрабатываемый продукт относится к системам учета готовой продукции. Таким образом, его невозможно причислить к программам с особыми требованиями надежности. Также следует отметить, что общая архитектура системы эффективно противостоит сбоям различных подсистем. При этом авторам неизвестны исследования, которые показывают факт снижения надежности программного продукта при использовании ОСРВ.

На основании этого можно обосновать целесообразность использования операционной системы реального времени в создаваемом проекте. Также следует отметить ряд дополнительных достоинств данного подхода. Основным преимуществом является инкапсуляция задачи и ее абстрагирование от других объектов. Это значительно повышает надежность в целом, существенно снижая риск фатального сбоя микроконтроллера в целом при аварии отдельной задачи. Следующим существенным преимуществом является простота расширяемости при сохранении режима, приближенного к реальному времени.

При использовании процедурного программирования микроконтроллера в связке с ОСРВ реализуется специфическая блочная структура программы. Она похожа на объектно-ориентированную модель, но при этом, оперируя точными определениями, используемая реализация является процедурной.

2. Реализация модульной структуры подсистемы

Нам неизвестны специальные инструментарный средства описания проектных решений данного типа. Анализ типовых диаграмм UML показывает, что существующие элементы могут быть с некоторыми допущениями применены для описания модульной структуры создаваемой подсистемы. В частности, нами предложено использовать типовую диаграмму классов UML [6,7]. При этом каждый объект на диаграмме является задачей операционной системы.

Показанная на рис. 1 диаграмма позволяет представить состав каждой отдельной задачи, которая по факту является сродним классу объектом в объектно-ориентированном программировании. Связи между объектами на диаграмме соответствуют правилам построения диаграмм классов, их представление описывает назначение и особенности.

Рассмотрим структуру функциональных объектов, каждый из которых является отдельным объектом ОСРВ. Основным из них является центральный диспетчер (mCU manager). В текущей версии информационной системы он будет представлен единой задачей, реализованной без объектно-ориентированного подхода внутри. Однако чтобы показать его функциональность и сложность реализации он разбит на несколько подобъектов. Дочерние объекты связаны с родительским связью агрегации, которая показывает их подчиненную роль, выполняющую ряд определенных функций в составе более крупного структурного элемента. В дальнейшем планируется разделение этих дочерних объектов между отдельными задачами операционной системы.

Родительским элементом центрального диспетчера в основном выполняются функции, отвечающие за реализацию регулярных процессов во времени. Основными функциями является запуск планировщика (startScheduler()) и функция обновления параметров (changeSchedulerParams()).

Базовый набор параметров, с которыми запускается задача в текущей версии системы представлен следующими элементами: стартовые параметры (startParameters) фактически не являются единым информационным объектом, включая в себя множество разнообразных данных, описывающие специфику работы основного планировщика задач; базовый сценарий (baseScenario) определяет алгоритм работы центрального диспетчера, он вынесен в отдельную группу, так как чрезмерно значителен для роли параметра; расписание (timeTable) также может быть отнесено к параметрам, но вынесено в отдельную группу по причине крупной специфичной структуры, описывающей

распределение всех процессов диспетчера во времени; список процессов (processesList) в настоящее время статичен и также является параметром, но в дальнейшем при создании отдельных задач ОСРВ для каждого дочернего объекта диспетчера он будет использован при динамическом создании подчиненных задач.

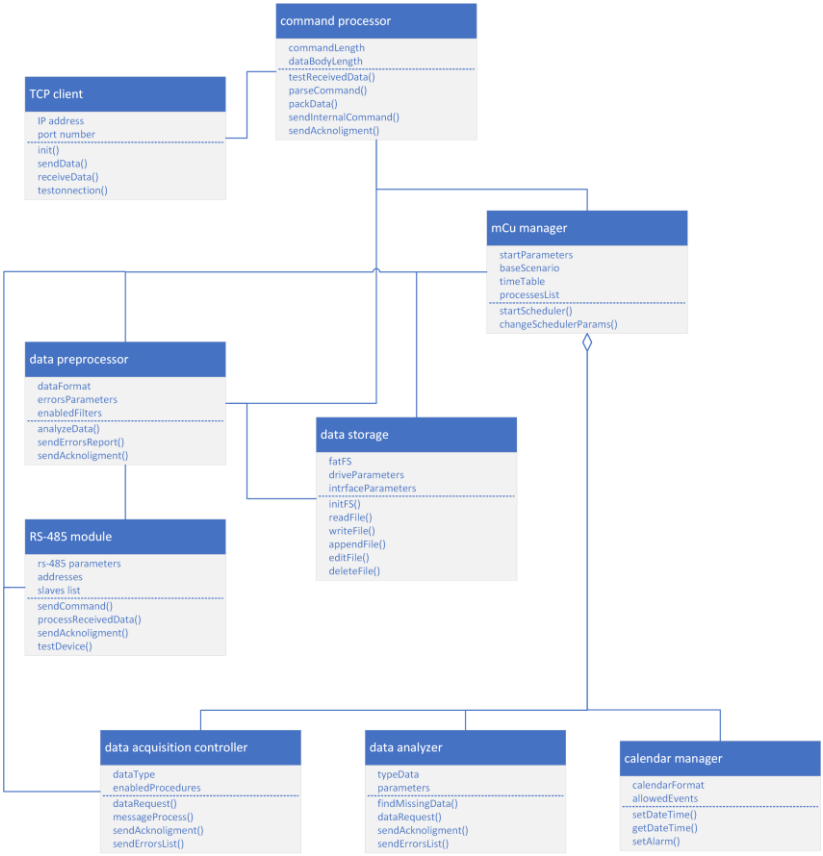

Рис. 1. Модульная структура подсистемы

Центральный диспетчер связан ассоциативностями со всеми объектами, исключая TCP клиент. Данные связи будут представлены в описании каждого ассоциированного объекта. Также необходимо отметить, что TCP клиент может получать базовые подтверждения от

центрального диспетчера, но они не показаны в виде связи в связи с их незначительно функциональностью.

Рассмотрим дочерние объекты, агрегатируемые центральным диспетчером. Наиболее важным является модуль получения данных от счетных устройств (data acquisition controller). Он отвечает за процесс получения данных от внешних устройств учета. В его функциональность входит формирование команд модулю управления внешними интерфейсами, оценка ответов от данного модуля, препроцессора обработки и модуля управления внешней памятью. На основе полученных ответов данный объект принимает решение о корректном завершении сбора счетных данных или о повторении неудачных действий.

В состав базовых параметров модуля получения данных от счетных устройств входит описание обрабатываемых типов данных (dataType) и список используемых процедур (enabledProcedures). Список процедур показывает, какие методы обработки информации будут использованы в данный момент. Они могут отличаться для различных счетных устройств и/или для различной продукции.

В качестве базовых функций модуля получения данных от счетных устройств выделим следующие: запрос данных от модуля управления внешними интерфейсами (dataRequest()) выполняет типовые аппаратноабстрагированные запросы к счетному оборудованию; процессор типовых сообщений (messageProcess()) обрабатывает ответы от смежных модулей и отвечает за решения о взаимодействии; функция отправки подтверждения (sendAcknoligment()) применяется для положительного ответа на направленные запросы; в случае выявления ошибок при отработке типовых сценариев предусмотрена функция отправки списка ошибок (sendErrorsList()).

Анализатор данных (data analyzer) выполняет анализ имеющихся данных в дисковой памяти при получении внешнего запроса. Данный объект должен определить наличие возможности выдачи валидных данных в полном объеме. Существует малая вероятность низкого качества данных. В этом случае данные должны быть запрошены со счетного устройство повторно, если срок их хранения на нем не вышел. Однако наиболее распространенной ситуацией является получение внешнего запроса, требующего данные, которые еще не были сохранены во внешней памяти. В этом случае анализатор данных должен определить, какие данные следует запросить от оборудования, а какие можно получить из имеющегося банка данных. При этом рассматриваемый модуль ответственен за все процессы управления

подчиненными модулями, а также обработку и сохранение новых данных на дисковом накопителе.

В базовой версии объекта анализатора данных мы выделим следующие параметры: типы данных (typeData), описывающие запрашиваемые данные и список общих параметров (parameters), отвечающих за функционирование модуля и параметры обработки. К функциям модуля стоит отнести: функцию поиска недостающих данных (findMissingData()), которая определяет необходимые, но отсутствующие в дисковой памяти параметры; функции запроса данных от различных источников (dataRequest()), данный набор включает несколько схожих функций; функцию формирования и отправки подтверждения операций (sendAcknoligment()); а также функцию формирования и отправки перечня ошибок при их возникновении (sendErrorsList()).

В отдельный подмодуль центрального диспетчера вынесен объект для работы со встроенным календарем (calendar manager). Он выполняет управление аппаратными ресурсами часов реального времени (RTC), а также включает ряд функций по обработке календарных событий. К базовым параметрами данного объекта относится формат календаря и часов (calendarFormat), а также список разрешенных календарных событий (allowedEvents). Базовыми функциями являются функция установки даты и времени (setDateTime()) и обратная ей функция их получения (getDateTime()), также отметить функцию установки события по времени (setAlarm()).

Модуль управления внешними интерфейсами (RS-485 module) используется для реализации абстракции запросов к счетному оборудованию, позволяющей упростить структуру команд центрального диспетчера и сделать ее аппаратнонезависимой. Фактически основная функциональность объекта определяется набором встроенных драйверов. Также модуль отвечает за взаимодействие с препроцессором полученных данных и содержит функции формирования ответов и списков ошибок.

К базовым параметрам данного модуля относятся: параметры работы интерфейса (rs-485 parameters); список адресов оборудования (addresses); список счетных устройств (slaves list), который также включает сведения, описывающие алгоритмы работы со счетными устройствами. К минимальному набору функций модуля относятся: функции отправки команд оборудованию (sendCommand()), которые являются зависимыми от конкретного типа устройств и их настроек; функции начальной обработки полученных данных (processReceivedData()) для предоставления их в другие модули в целостном виде; функции отправки подтверждений

(sendAcknoligment()); функции тестирования интерфейса и счетных устройств (testDevice()).

Модуль управления внешними интерфейсами имеет ассоциативные связи только с 2 объектами: центральным диспетчером и препроцессором обработки данных. От первого из них модуль получает управляющие команды и выдает подтверждения или сведения об ошибках. Второй модуль получает необработанные (но собранные воедино) данные, полученные от счетных устройств.

Препроцессор обработки данных (data preprocessor) используется для предварительной обработки полученных от счетных устройств данных. Она включает анализ данных на наличие грубых ошибок, а также элементы цифровой фильтрации. На основании выводов данного модуля выполняется повторный опрос оборудования в случае фатального повреждения данных. Данные также могут быть подвергнуты базовой статистической обработки. По завершению работы данного объекта выдается команда модулю обслуживания внешней памяти на запись результатов.

Базовыми параметрами препроцессора обработки данных являются: формат данных (dataFormat); параметры обрабатываемых ошибок и процедуры обработки (errorsParameters); список используемых фильтров и их параметры работы (enabledFilters). К основным функциям следует отнести следующие: запуск анализа необработанных данных (analyzeData()); отправку подтверждения (sendAcknoligment()) с последующей записью данных (функция записи отдельно не показана); функцию формирования и отправки списка ошибок (sendErrorsReport()).

Препроцессор обработки данных имеет ассоциативные связи с модулем управления внешними интерфейсами, с которым выполняется основной обмен данными, центральным диспетчером, который получает от него отчеты и подтверждения, а также модулем обслуживания внешней памяти, который работает с внутренним банком данных нижнего уровня системы.

В задачи этого модуля (data storage) входит весь набор функций по реализации работы с дисковой памятью. Их следует разделить на три большие группы: поддержка аппаратных ресурсов (в нашем случае работы с SD card по интерфейсу SPI или SDIO), поддержка среднего уровня программных библиотек для реализации базовых функций взаимодействия с файловой системой и файлами, а также функции для работы с конкретными данными.

Рассмотрим основные функции данного модуля. К ним относятся: инициализация файловой системы (initFS()); чтение файла (readFile()); запись файла, в том числе создание пустого файла при го отсутствии

(writeFile()); продолжение записи файла, в том числе создание пустого файла при го отсутствии (appendFile()); редактирование файла, замена блока данных (editFile()); удаление файла (deleteFile()) и другие.

Основными параметрами инициализации рассматриваемого объекта являются: тип файловой системы и ее параметры (fatFS); параметры и методы работы с дисковым носителем (driveParameters), параметры и режимы работы аппаратных интерфейсов для подключения внешней памяти (intrfaceParameters).

Ассоциативные связи у этого модуля имеются с описанными ранее объектами, а также с командным процессором. Они необходимы для отправки данных из внешней памяти верхнему уровню системы учета.

Командный процессор (command processor) является посредником между сетевым клиентом и другими элементами системы. Фактически он выполняет два типовых набора функций: упаковка данных для отправки посредством TCP клиента и распаковка полученных команд с последующей предварительной обработкой. В процессе распаковки внешние команды преобразуются в набор внутренних команд и определяется порядок их выполнения, а также внутренние сценарии их реализации.

Командный процессор содержит два основных базовых параметра: общая длина команды верхнего уровня системы (commandLength) и длина тела команды (dataBodyLength), которая определяет объем аргументов в команде. Эти параметры ограничивают максимальную длину, что важно при выделении памяти на обработку данных.

Основная функциональность командного процесса представлена следующим набором функций: функция базового тестирования корректности полученной команды (testReceivedData()) анализирует пришедшую от сетевого клиента команду; функция парсинга (parseCommand()) выполняет анализ поступившей команды и ее разбиение на структурные элементы; функция упаковки данных (packData()) формирует пакет с отправляемыми данными для последующей передачи клиентом; функции отправки на выполнения внутренних команд (sendInternalCommand()) реализуют исполнение инструкция после их логического анализа и преобразования парсером; для подтверждения также используется соответствующий инструментарий (sendAcknoligment()).

Фактически командный процессор ассоциативно связан только с сетевым клиентом, а также центральным диспетчером и модулем data storage.

Сетевой клиент (TCP client) выполняет только задачи сетевой коммуникации и реализует аппаратную поддержку интерфейса Ethernet.

Данный модуль жестко отделен от командного процессора по причине значительной нагрузки вычислительных ресурсов стеком сетевых протоколов, обеспечивающим работу клиента.

К основным параметрам относится IP адреса устройства и сервера системы (IP address), а также номер порта для реализации TCP обмена (port number). Функции содержат следующий базовый набор: инициализация сетевого устройства и стека протоколов TCP/IP (init()); функции отправки подготовленных данных (sendData()); функции обработки полученных от сервера данных (receiveData()); набор функций для тестирования сетевого соединения и используемого стека протоколов (testonnection()).

Заключение

Все модули реализованы в виде отдельных задач. Это позволяет повысить надежность путем инкапсуляции процессов. Для стабильной и защищенной передачи данных между отдельными задачами применены стандартные средства ОСРВ. В частности, используются очереди статических и динамических объектов. Для разграничения доступа к критичным общим объектами применяется механизм мьютексов и семафоров, также реализованный стандартной функциональностью операционной системы.

Предложенная в ходе проектирования структура может быть легко масштабирована путем добавления новых объектов и связей. Это позволяет расширять функциональность подсистемы без изменения концептуального подхода ее формирования. В дальнейшем планируется добавить ряд объектов для улучшения функциональности разработки.

Допустимым путем развития подсистемы является использование динамически создаваемых объектов, являющихся экземплярами классов в рамках отдельных задач. Это переводит продукт в класс объектно-ориентированных программ. Однако концептуальный подход в целом остается неизменным. При этом вновь созданные объекты приобретают связь композиции с ранее созданными и представленными в этой работе.

Список литературы

1. STM32 32-bit Arm Cortex MCUs [Электронный ресурс] : техническое описание. – Режим доступа : https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

2. Мешков, А. Visual C++ и MFC / А. Мешков, Ю. Тихомиров. - М.: БХВ-Петербург, 2013. - 546 с.

3. Иванов, К. К. Проектирование информационных систем / К. К. Иванов. // Молодой ученый. — 2017. — № 19 (153). — С. 22-24.

4. Костин А.А., Баженов Р.И. Разработка информационной системы учета изъятия драгоценного металла из оборудования // Современные научные исследования и инновации. 2014. № 8-1 (40). С. 108-119.

5. Манако А.Ф. Подход к построению формализованного описания информационных систем для образования и обучения // Образовательные технологии и общество. 2013. Т. 16. № 1. С. 536-546.

6. Ларман, К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку / К. Ларман. - М.: Вильямс, 2013. - 736 с.

7. Буч, Г. Язык UML. Руководство пользователя / Г. Буч , Д. Рамбо , А. Джекобсон. - М.: ДМК, 2015. - 432 с.