УДК 629.7.05
***Ю.Г. Бондарос, д. т. н.*** *(ФГУП ГосНИИ Авиационных систем).*
***К.А. Маковкин, В.Я. Чучупал, к.ф.м.н.*** *(ВЦ им. Дородницына РАН).*

# Разработка системы распознавания команд речевого интерфейса пилота для интегрированной модульной авионики (ИМА)

Работа выполнена при поддержке РФФИ. Грант РФФИ 06-08-01534-а

*Описаны алгоритмы системы распознавания команд речевого интерфейса пилота по стандартам интегрированной модульной авионики (ИМА, Integrated Modular Avionics – IMA). Определены возможные для разработчика допущения и ограничения в реализации алгоритмов.*

## Введение

Традиционно системы бортового радиоэлектронного оборудования разрабатывались, используя сделанные по специальному техническому заданию изделия и программное обеспечение. Однако, в последнее время, высокая себестоимость систем с учётом их полного жизненного цикла вынуждает разработчиков авиационной техники кроме специально заказанных систем рассматривать создание систем, основанных на готовых коммерческих продуктах. Одновременно, наблюдается тенденция перехода к универсальным вычислительным платформам, которые могут использовать приложения различного типа, а также могут исполнять комплексные приложения одновременно. Этот подход известен как ИМА. Различные архитектуры программного обеспечения ИМА, несмотря на отличия, обладают одними и теми же признаками высокого уровня:

- наличие общих подсистем обработки
- абстрактность программного обеспечения.

Общие подсистемы обработки должны позволить комплексным приложениям совместно и многократно использовать одни и те же самые вычислительные ресурсы. Это позволяет уменьшить число загружаемых подсистем, которые полностью не используются, и обеспечивает более эффективное использование ресурсов системы, оставляя возможность для расширения в будущем.

Абстрактность программного обеспечения должна освободить приложение от зависимости не только от архитектуры основной шины и от архитектуры используемого оборудования. Это позволяет использовать приложения на различных платформах, а также позволяет заменять устаревшие архитектуры на новые. Хотя уже разработано много стандартов для ИМА [1-12], документ RTCA DO-255 "Requirements Specification for Avionics Computer Resource (ACR)" [3] и спецификация ARINC 653 [4] содержат наиболее важную информацию для разработчиков прикладного программного обеспечения (ППО) для ИМА. Документ RTCA DO-255 содержит описание требований к бортовым компьютерным ресурсам (ACR). К ресурсам ACR относятся совместно используемые вычислительные ресурсы, базовое программное обеспечение и средства формирования сигналов, необходимые для взаимодействия с различными бортовыми системами.

ARINC 653 определяет образец выполнения программного обеспечения ИМА на высоком уровне. ARINC 653-1 - сокращение для Draft 3 of Supplement 1 to ARINC Specification 653: Avionics Application Standard Software Interface , изданного 15 июля 2003 г. Технические требования ARINC 653-1 стали стандартом в октябре 2003 года. В этом документе описана базовая операционная среда прикладного программного обеспечения, используемого в составе ИМА и традиционной авионики серии ARINC 700. Главной целью данной спецификации является описание интерфейса общего назначения APEX (APplication/EXecutive) между операционной системой (ОС) бортовых компьютерных ресурсов и прикладным программным обеспечением. В спецификацию входят требования к интерфейсу между прикладным программным обеспечением и ОС а также список сервисов, которые должны позволять прикладным программам контролировать диспетчеризацию, связь и данные о состоянии внутренних обрабатываемых элементов. В этой спецификации описаны данные, обмен которыми производится статическим (за счёт конфигурации) и динамическим (за счёт сервисов) путём, а также сервисы, предоставляемые операционной системой и используемые прикладными программами. Спецификация не диктует требования относительно конкретной реализации каких-либо аппаратных средств или программного обеспечения системы.

Значительная часть этого документа посвящена оперативным средствам, действующим в отношении встраиваемых прикладных программ авиационных электронных систем. Список сервисов включает минимальные функции, которые должно выполнять ППО, то есть представляет собой стандартный авиационный интерфейс. Этот интерфейс призван быть максимально общим, так как слишком сложные интерфейсы или системно-ориентированные интерфейсы не универсальны. Спецификации интерфейса APEX, касающиеся программного обеспечения, не зависят от языка высокого уровня,

поэтому для следования этому интерфейсу система может использовать разные компиляторы и языки.

Эти и другие стандарты ИМА, предъявили новые требования к структуре и выполнению программного обеспечения систем реального времени, обеспечиваемых коммерческими разработчиками.

Реализация концепции ИМА с открытой архитектурой должна дать разработчику программного обеспечения возможность сертифицировать свою продукцию в России и за рубежом по упрощенным процедурам. Сокращение времени и затрат на сертификацию изделий может быть достигнуто за счет использования задела и стандартных технических решений, которые уже одобрены АР МАК и EASA.

## Операционная система реального времени.

Одной из задач базового модуля в любой системе ИМА является поддержка одной или нескольких прикладных программ в составе авиационной электронной системы и обеспечение независимой работы этих приложений. Эта задача может быть корректно решена, если в системе обеспечить разделение, то есть разделить прикладные программы авиационной электронной системы с точки зрения выполняемых функций – прежде всего, для ограничения распространения последствий отказов (то есть, чтобы отказ функции, реализуемой в одном из разделов, не мог вызвать отказа функций, реализуемой в других разделах), для упрощения верификации, подтверждения и сертификации.

Технические требования к операционной системе для ИМА определяют две важных концепции. Это пространственное и временное разделение, определяющие требования по разъединению для приложений, запускаемых одновременно на одной и той же вычислительной платформе (модуле ядра). Различные приложения, запущенные в ИМА, не должны иметь возможность лишать друг друга совместно используемых прикладных ресурсов, которые обеспечиваются вычислительной платформой. Обычно это достигается с помощью раздельных виртуальных контекстов памяти, которые приводятся в исполнение модулем управления памятью процессора (Memory Management Unit - MMU). Эти контексты названы разделами. Они содержат приложение со своей собственной динамической памятью, а также стек для процессов приложения. В единой среде ППО раздел - это, обычно, то же самое, что программа: он включает данные, свой собственный контекст, атрибуты конфигурации и т. д. В больших приложениях может быть несколько разделов.

Временное разделение гарантирует, что ни одно приложение не может использовать процессор дольше, чем предназначено. Если исполнение подразумевает основанное на временном разделении планирование, каждый раздел спланирован для временного интервала определенной ширины, другим разделам выделены свои временные интервалы. В конце временного интервала для данного раздела, программатор вынуждает переключение контекста на следующий раздел в плане. В пределах своего временного интервала, раздел может использовать свою собственную стратегию планирования. Эта модель достаточно гибка, чтобы использовать существующие или новые приложения ИМА, разработанные автономно, но неизбежно имеется проблема включения в план дополнительных разделов и проверки, что границы и графики не нарушены.

На Рис. 1 представлена архитектура программного обеспечения для ИМА. Главная операционная система (ОС) взаимодействует непосредственно с вычислительной платформой (модулем ядра). Она обеспечивает управление глобальными ресурсами, управляя каждым приложением (разделом) и контролируя его исправность. ОС раздела используется в каждом разделе для индивидуального планирования и управления ресурсами. Связь ОС раздела с главной ОС - через отдельный интерфейс пересылки сообщений. Такая архитектура обеспечивает гибкость, невозможную в варианте исполнения с монолитным ядром. Разделы - контейнеры, в которые могут быть помещены процессы, объекты и ресурсы, с разделением, приводимым в исполнение модулем управления памятью (Memory Management Unit - MMU). Каждый домен защиты имеет свой собственный стек и локальную динамическую память, которая не может быть захвачена приложениями, запускаемыми из других разделов. Домены защиты также предотвращают влияние ошибочных обращений к памяти от приложений, запускаемых из других разделов.

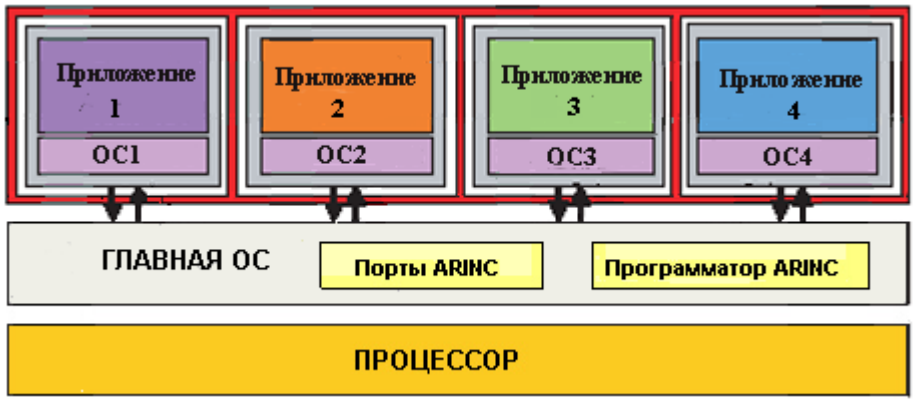


*Рис. 1: Архитектура системы ИМА*

ОС авиационного назначения должна обеспечивать высокую надежность, гарантируя невозможность полного отказа системы. Кроме того, вероятность сбоя в ней должна быть мала. Качество (включая требования по безопасности и защищенности данных) ОС и ППО быть доказано.

ОС LynxOS-178 разработана фирмой LynxWorks как коммерческий продукт с открытым интерфейсами и отвечающий требованиям ARINC 653. Основу LynxOS-178 составляет LynxOS 3.0. LynxOS-178 версии 2.0 поддерживает [13]:

- до 16 разделов;
- до 64 процессов внутри каждого раздела;
- до 51 потока внутри каждого процесса;
- диспетчеризацию реального времени потоков внутри раздела;
- функции POSIX межпроцессорного взаимодействия внутри раздела.

POSIX [4] - стандарт, позволяющий приложениям быть перенесенным в ИМА. POSIX - название семейства связанных стандартов (примерно 15 документов), установленных IEEE по определению интерфейса прикладных программ (API) для программного обеспечения, совместимого с вариантами операционной системы Unix. Формально эти стандарты обозначены как IEEE 1003 или как ISO/IEC 9945. Их разработка началась примерно в 1985 году. Термин POSI расшифровывается как Portable Operating System Interface, а X отражает принадлежность API к Unix.

Фирмой Wind River Systems разработана операционная система реального времени VxWorks [6]. Подобные операционные системы разработаны другими фирмами: QNX, VRTX, pSOS, Windows-CE, Nucleus RTX и т.д. Приложения могут быть новыми, разработанными с нуля, или уже существующими. Они могут быть разработаны на различных языках программирования и могут использовать различные модели планирования. Обеспечение поддержки разнородных (гетерогенных) приложений, запускаемых как индивидуальный раздел, позволяет приложениям на разных языках выполняться под ОС раздела.

## Интерфейс ARINC 653

ARINC 653 определяет требования к обеспечению интерфейса APplication EXecutive (APEX), который может использоваться, когда комплексные приложения должны совместно использовать единый процессор и при этом ни одно приложение не может вызвать сбой другого в случае неисправности.

ARINC 653 APEX, иногда называемый ARINC 653 API, обеспечивает универсальный интерфейс между операционной системой и прикладным программным обеспечением. ARINC 653 API также обеспечивает уровень абстракции, который скрывает подробности выполнения ОС по ARINC 653 от приложения и основной архитектуры модуля ядра. Цель состоит в том, чтобы иметь одни и те же определения и представления интерфейса на языках Ada и C.

Интерфейс операционной системы по ARINC 653 обеспечивает определённые гарантии разработчику ППО. В нём описываются сервисы системы и структуры данных с заданной семантикой, которые можно будет надёжно использовать, если ППО будет работать на базе операционной системы, соответствующей необходимым требованиям. Таким образом, существует два типа соответствия стандарту ARINC 653 - соответствие данной реализации операционной системы и соответствие ППО.

Интерфейс APEX, являющийся элементом связи между ППО и ОС, определяет набор средств, которые будут обеспечиваться системой для управления диспетчеризацией, связью и данными о состоянии внутренних обрабатываемых элементов. С точки зрения ППО интерфейс APEX можно рассматривать как спецификацию на языке высокого уровня, а с точки зрения ОС как определение параметров и механизмов входа (например, вызовов внутренних прерываний). APEX может включать транслятор спецификации с языка высокого уровня в соответствующий механизм входа. Эта трансляция зависит от реализации ОС, аппаратной платформы и, возможно, также от того, какой компилятор используется для компиляции ППО. Предполагается, что поставщик ОС создал пакет прикладных программ, который будет состоять из набора процедур (функций, подпрограмм и т. д.), которые будут иметь доступ к машинным средствам низкого уровня. Эти процедуры будут вызваться через интерфейс APEX.

Реализация интерфейса APEX не требует определения способа, которым компилятор должен обрабатывать вызовы. Кроме того, нет необходимости в том, чтобы в ППО и ОС использовался один и тот же язык. Реальный механизм, с помощью которого функции вызываются на машинном уровне - это работа компилятора, используемого программами ОС.

Существует множество альтернативных способов реализации интерфейса APEX. Одним из вариантов является использования специального или обязательного компилятора для программного обеспечения системы ИМА. Другим вариантом является использование ассемблера в составе прикладной программы для доступа к функциям ОС.

Третьим вариантом является набор процедур, с помощью которых осуществляется трансляция вызовов кода прикладной программы в вызовы ОС. Использование специального компилятора - это самый дорогостоящий способ, который, кроме того, характеризуется недостатком гибкости и возможными трудностями с будущим сопровождением. Включение ассемблера в прикладные программы не решает всех задач, так как при этом будет очень трудно обеспечить переносимость программ. В последнем третьем варианте, ППО может быть независимо от реализации, если будет обеспечен интерфейс для трансляции в ОС. Он может считаться элементом, отдельным от ППО. В этом случае сама прикладная программа становится переносимой.

Для разработки ППО авиационного назначения обычно используют и рекомендуют использовать язык Ada. Документ ARINC 613 «Руководство по использованию языка программирования Ada для систем авионики» даёт рекомендации по использованию языка программирования Ada в прикладных программах для коммерческой авиации. Он касается использования языка программирования Ada при разработке, тестировании и сопровождении цифровой авиационной электроники для коммерческой авиации.

Части раздела ППО, непосредственно использующие сервисы APEX, и спецификация интерфейса APEX должны быть написаны на одном и том же языке. Поэтому интерфейс должен иметь столько отдельных спецификаций на языках высокого уровня, сколько потребуется языков для разработки ППО. Эти спецификации должны отвечать определенным требованиям. Каждая спецификация должна представлять собой полное описание интерфейса APEX с точки зрения прикладной программы. Все спецификации должны обеспечивать сервисы с одной и той же семантикой независимо от языка. Это требование позволит одной и той же ОС поддерживать разные спецификации. Синтаксис сервисов, описанных на разных языках, может различаться, так как существуют различия в контроле типов и описательном охвате разных языков. Каждая спецификация должна компилироваться независимо. Прикладные программы, выполняемые на каком-то конкретном базовом процессорном модуле, не обязательно должны использовать одну и ту же спецификацию (то есть написанную на одном и том же языке).

Каждый раздел в системе по ARINC 653 представляет отдельное приложение и использует область памяти, которая ему специально выделена. Точно так же APEX назначает специальный выделенный интервал времени для каждого раздела, таким образом создавая разделение по времени. Каждый раздел по ARINC 653 поддерживает многозадачный режим.

Операционная система реального времени LynxOS-178 приспособлена к интерфейсу APEX по ARINC 653-1. LynxOS-178 обеспечивает следующие формы обслуживания систем в соответствии со стандартом ARINC 653-1: управление разделами, управление процессом, управление временем.

Соответствие стандарту ARINC позволяет авиационному производителям быть уверенными в том, что предлагаемое программное обеспечение будет совместимо и взаимозаменяемо.

## Конфигурирование системы

Одна из целей ARINC 653 заключается в том, чтобы определить среду, которая обеспечивала бы полную переносимость и возможность многократного использования исходного кода прикладных программ. Для этого ППО должно быть интегрировано в такую конфигурацию системы, которая удовлетворяла бы функциональным требованиям прикладных программ и требованиям функций летательного аппарата (ЛА) к доступности и надёжности системы. За интеграцию должна отвечать единая ответственная организация - системный интегратор. Созданная в итоге конфигурация должна давать возможность каждому разделу получать доступ к необходимым ресурсам и обеспечивать, чтобы удовлетворялись требования каждой прикладной программы к доступности и надёжности системы. Таким образом, системный интегратор должен знать, какие требования предъявляются к времени выполнения процессов, использованию памяти и внешним интерфейсам со стороны каждого раздела, который будет включаться в интегральную систему.

Конфигурирование процессов внутри разделов входит в обязанности разработчика ППО. Системный интегратор должен обеспечить, чтобы все разделы имели доступ к внешним данным, необходимым для правильной работы системы. Внутри системы обмен данными осуществляется в форме сообщений. Каждое сообщение должно быть уникальным и идентифицируемым. Интегратор должен обеспечить совместимость формата сообщений для всех разделов, отправляющих и принимающих сообщения.

Архитектура ARINC 653 гарантирует доступность ресурса с помощью системы и записей конфигурации раздела. Их назначение состоит в том, чтобы позволить конфигурирование множества приложений ИМА, разработанных одним или большим количеством разработчиков, в единую систему ИМА, которая конфигурирована

интегратором системы. Записи конфигурации раздела определяют характеристики каждого приложения в терминах требований к памяти, к процессору и к использованию порта ARINC. Записи конфигурации системы определяют возможности и мощности платформы ИМА, а также ссылаются на записи конфигурации индивидуальных разделов. Эта схема позволяет интегратору систем гарантировать, что требования приложений совместимы с характеристиками платформы и что индивидуальные приложения не превышают ассигнованные им ресурсы. ARINC Specification 653 обеспечивает определение структуры высокого уровня и содержание записей конфигурации.

Связь внутри раздела по ARINC 653 ответственна за связь между процессами, находящимися в одном разделе.

## Система контроля и перезапуска

ARINC 653 определяет концепцию контроля состояния (Health Monitoring - HM) в пределах систем ИМА. Она отвечает за "контроль исправности аппаратных средств, приложения и операционной системы", и её роль - помочь "изолировать неисправности и предотвратить размножение неисправности". Это требование требует сложного, в масштабе системы, контроля состояния HM, который имеет возможность отслеживать ошибки и выполнять переконфигурирование и восстановление. Отклик на индивидуальную неисправность зависит от характера неисправности, ее серьезности и стратегии управления обработкой ошибок, определенной интегратором систем. Система контроля состояния (HM System – HMS) это сложная подсистема, которая является встроенной частью архитектуры. Она выполняет все требования ARINC 653 и обеспечивает дополнительные функции, определяемые интеграторам систем и предназначенные использовать динамическое переконфигурирование (в частности, основанное на режиме планирование).

Архитектура HMS состоит из сервера HM для всей системы, включая главную ОС, и агентов (исполнительных устройств) HM, которые находятся в индивидуальном разделе. HMS обрабатывает события, требующие её внимания. Неисправность представлена в программном обеспечении тревогой. Структура также поддерживает второй тип событий, сообщение, которое может использоваться для регистрации или для конфигурирования интегратором систем. Структура обеспечивает способность выполнять контроль исправности на трех уровнях, HM процесса, HM раздела и HM модуля ядра путем трёх типов услуг: обнаружение тревоги, регистрация тревоги и отклик на тревогу. Чтобы способствовать мобильности, структура использует определения для кодов ошибки по

ARINC 653, которые включают события типа пропущенного крайнего срока, числовой ошибки, незаконного запроса, сбоя питания, и т.д. Сигнальный ответ зависит от уровня ошибок. На уровне модуля это может включать перезагрузку или закрытие системы. На уровне раздела это может приводить к перезагрузке раздела. При создании раздела используется холодный старт (размещение и инициализация объектов раздела), горячий старт используется при повторном старте или перезапуске раздела (объекты раздела не размещаются заново, а только повторно инициализируются). При возникновении ошибок процесса, реакция управляема приложением, и поэтому предпринятое действие зависит от типа ошибки и его контекста.

Примером комплексного раздела (приложения) ИМА может служить речевой интерфейс пилота ЛА, например, самолета или вертолета.

**Речевой интерфейс пилота.**

Речевой интерфейс – альтернативное средство обеспечения диалога пилота с авионикой ЛА, облегчающее работу пилота в напряженном информационном поле за счёт использования наиболее естественного для человека средства передачи информации - речи. ПО речевого интерфейса должно уметь (реализовать внутренние функции): выполнять сопряжение пилота с речевым интерфейсом с помощью микрофона, ларингофона, телефона; преобразовывать и кодировать речь; распознавать говорящего (идентифицировать, верифицировать); понимать его речь; синтезировать искусственные речевые сигналы и выполнять сопряжение речевого интерфейса с бортовым процессором. Эти внутренние функциональные возможности позволят речевому интерфейсу выполнить следующие задачи (внешние функции):

- идентифицировать говорящего и осуществить по речевому паролю доступ к определенным функциям управления после проверки аутентичности речи пилота;
- распознать команды управления авионикой ЛА (например, управления светом и температурой в кабине, рабочей частотой средств связи, навигацией, двигателем);
- осуществлять контроль работоспособности пилота;
- вести речевой диалог пилота с авионикой ЛА (запросить и получить в виде речи параметры движения и местоположения, температуру в кабине и т. д.);
- передавать аварийно-предупредительные сообщения;
- накапливать и хранить аудиоданные переговоров в кабине пилота;
- осуществлять голосовую связь.

Основные преимущества речевого канала:

- речь как является самой естественной формой представления информации, не требующей перекодировки.
- глаза и руки пилота в момент речевого ввода могут быть сосредоточены на другой задаче.
- речь является доминирующей формой представления информации, так как обладает высокой степенью внушаемости.
- речевые технологии относятся к наиболее дешевым, так как стоимость оборудования определяется стоимостью программного обеспечения. Остальное оборудование (кроме небольшого дисплея обратной связи) уже есть на борту.

Словарь речевого интерфейса зависит от класса ЛА и содержит примерно:

100-120 слов для речевых команд дискретного управления;

120-150 аварийно-предупредительных сообщений;

100-150 диалоговых фраз;

120-180 информационных сообщений;

Общий объем речевых терминов не превышает 500- 600 словоформ и фраз.

Типовые требования, предъявляемые к характеристикам алгоритма распознавания раздельно произносимых команд имеют вид:

- ✓ Точность распознавания команд – не ниже 95%.
- ✓ Время распознавания команды <= 0,2 с.
- ✓ Темп распознаваемой речи: 100 слов/мин.
- ✓ Темп синтезируемой речи: 120 -160 слов/мин.
- ✓ Полное время подстройки системы распознавания к голосу конкретного пилота – не более 1 - 1,5 часов.

Речевой интерфейс пилота в терминах ИМА можно рассматривать как комплексное приложение, состоящее из нескольких разделов. Наиболее сложной частью задачи алгоритмического построения речевого интерфейса является создание раздела (приложения) - системы распознавания команд пилота.

## Структура системы распознавания команд

Система речевого управления содержит систему распознавания команд, преобразователь результатов распознавания речевых команд в сигналы управления объектом, а также подсистему мониторинга, которая позволяет контролировать визуально, на дисплее, работу системы и осуществлять настройки системы. Упрощенная функциональная схема системы распознавания команд приведена на Рис. 2.

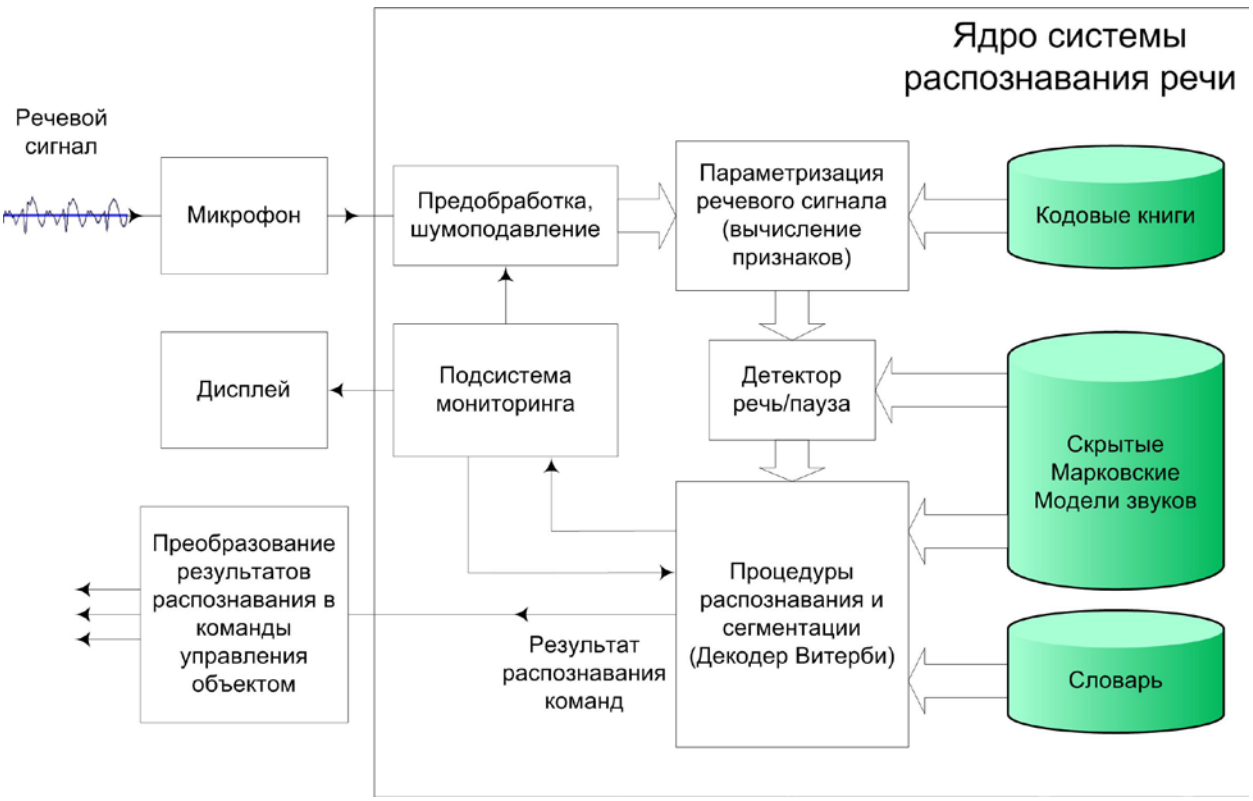


*Рис. 2. Упрощенная функциональная схема системы речевого управления*

Основой описания речевого сигнала рассматриваемой системы на акустико-фонетическом уровне является алфавит звуков (точнее, основных вариантов произношения звуков - аллофонов) русской речи. На этом алфавите, в частности, записаны варианты произношения команд (произносительные транскрипции). Алфавит звуков универсален в том смысле, что произношение любого слова или словосочетания может быть достаточно точно записано его символами. Каждый звук – элемент алфавита имеет модель его произношения. Модели звуков можно рассматривать как элементарные “кирпичики”, из которых строится модель произношения высказывания. Модели звуков – Марковские цепи с лево-правой структурой из 3-4 (в зависимости от фонетического качества) состояний.

Задание словаря системы распознавания осуществляется в текстовой форме – каждое слово (ключевое или заполнитель) определяется своей произносительной транскрипцией. Например, подмножество команд управления может определяться следующим образом:

```
пауза        [.]  sil
слева             s l’ e^ v a
справа            s p r a^ v a
```

| | | |
|---|---|---|
| **больше** | | **b o l’ sh e** |
| **борт** | | **b o r t** |
| **включить** | | **v k l’ u ch’ i^ t’** |
| **выключить** | | **v y k l’ u ch’ i t’** |
| **двигатель** | | **d v’ i g a t’ e^ l’** |
| **дыхание** | **[ ]** | **d y h a^ n’ i je** |
| **закрылки** | | **z a k r y^ l k’ i** |
| **кабина** | | **k a b’ i^ n a** |
| **кислород** | | **k’ w0 s l a r o^ t** |
| **кондиционер** | | **k a n’ d’ i ts y a n’ e^ r** |
| **меньше** | | **m’ e^ n’ sh e** |
| **фара** | | **f a^ r a** |
| **обороты** | | **a b a r o^ t y** |
| **свет** | | **s v’ e^ t** |
| **стартер** | | **s t a r t’ o r** |
| **шасси** | | **sh a s’: i** |

Здесь использована транскрипция произнесения команд в алфавите, который использован в работе [15]. Символ | - разделитель альтернативных транскрипций, а в скобках [] указано визуальное представление команды, для отображения в протоколе или на экране, если оно отлично от орфографической записи команды.

Поскольку речевой сигнал моделируется на уровне звуков, алфавит которых универсален, то словарь системы распознавания , в принципе, неограничен и легко может быть модифицирован. С точки зрения вероятности правильного распознавания, при конструировании словаря важно выбрать акустически различные команды, корректно задать произносительную транскрипцию команд и описать грамматику рабочего языка. Описание грамматики осуществляется в форме Бэкуса-Наура и соответствует рекомендациям консорциума W3C (см. http://www.w3.org).

Рассмотрим, для примера, фрагмент грамматики прикладного языка, которая соответствует словарю команд управления:

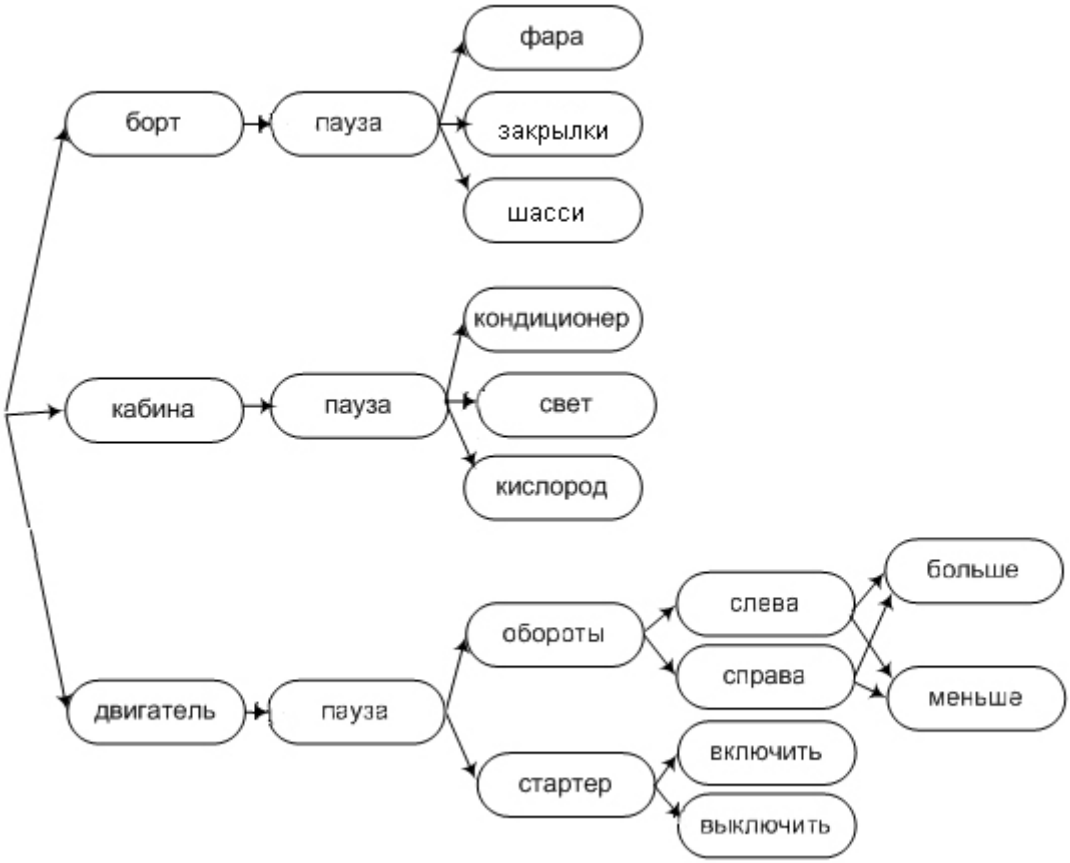


*Рис. 3 Фрагмент грамматики прикладного языка.*

Формальное описание фрагмента грамматики для ввода в систему распознавания речи имеет следующий вид:

```
$сторона = слева | справа;
$состояние = включить | выключить;
$уровень = больше | меньше;
$борт = борт [пауза] фара |закрылки | шасси;
$кабина = кабина [пауза] кондиционер |свет | кислород;
$двигатель= двигатель [пауза] (обороты $сторона $уровень | стартер $состояние);
$речь = [$кабина | $борт | $двигатель]
$неречь = пауза|дыхание;
($неречь | $речь)
```

Здесь вертикальная черта (|) соответствует альтернативному выбору команды, а квадратные скобки ([]) – необязательному использованию команды.

Обучение системы распознавания заключается в оценке параметров Марковских моделей звуков, в частности условных вероятностей наблюдения параметров в данном состоянии, вероятностей перехода между состояниями, а также параметров модели языка – вероятностей следования команд. Для рассматриваемой системы оценка параметров

Марковских моделей, как правило, не требуется, поскольку система содержит набор готовых акустических моделей. Однако, если акустическая обстановка, в которой планируется использовать систему распознавания, существенно отличается от той, для которой выполнялась оценка параметров (косвенно на это может указывать снижение точности распознавания), тогда имеет смысл провести переоценку параметров моделей звуков. Эта процедура выполняется отдельным модулем обучения в соответствии с общей технологией оценки параметров Марковских моделей и схематически представлена на Рис.4. На вход программы обучения подается достаточно представительный фонетический или орфографически транскрибированный речевой материал (неоднократное произнесение всех команд), который собран в актуальной акустико-фоновой обстановке.

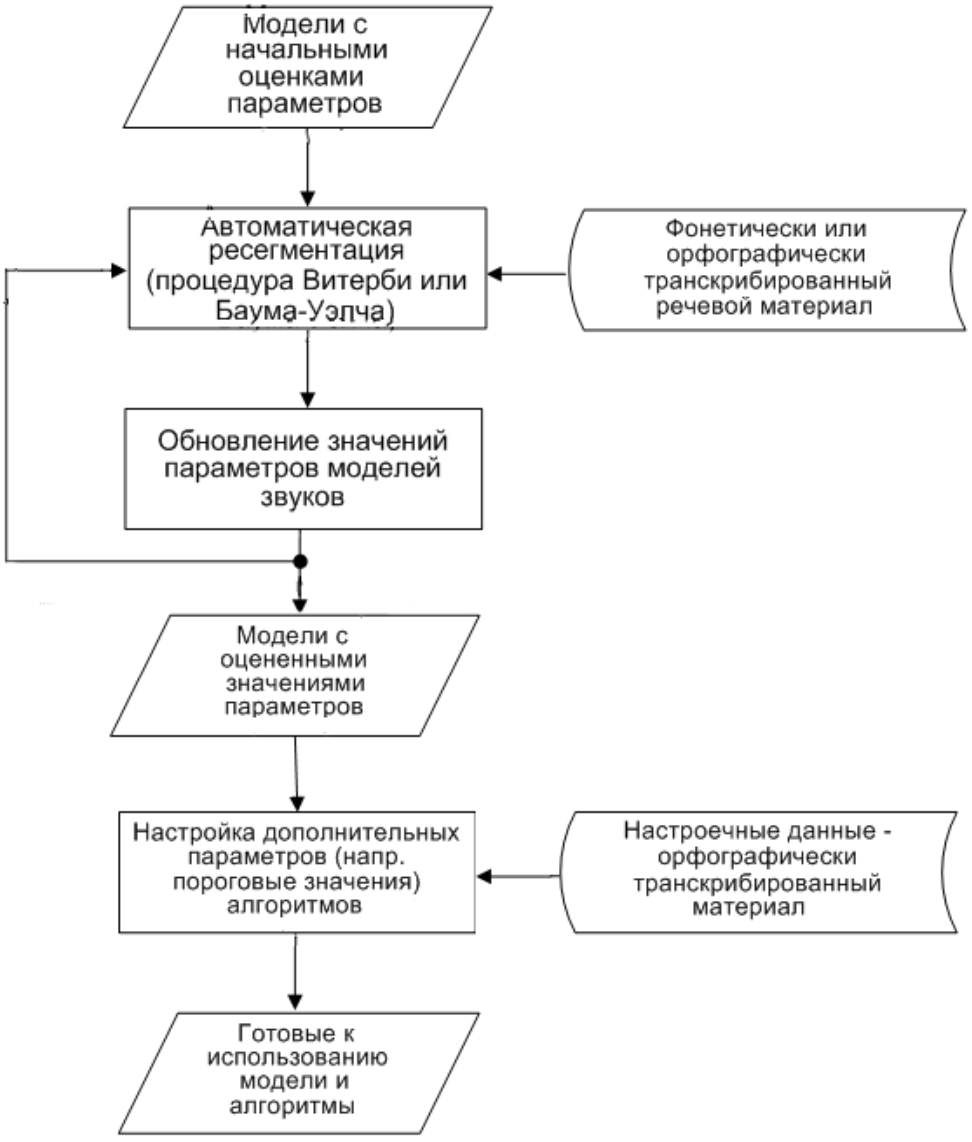


*Рис. 4. Блок-схема процедуры переоценки параметров Марковских моделей звуков.*

## Распознавание речевых команд

В соответствии со схемой на Рис. 2 при распознавании речевого высказывания сигнал проходит несколько этапов обработки. Рассмотрим эти этапы подробнее.

***Предобработка и шумоподавление.*** На этом этапе происходит очистка речевого сигнала от шума. Задача этой операции состоит в повышении отношения сигнал/шум и соответственно повышении точности распознавания. В системе используется шумоподавление, основанное на теории оптимальной фильтрации, которое осуществляется в две стадии (см. Рис. 5.): первоначальной очистки входного сигнала и дополнительного динамического шумоподавления, которое зависит от соотношения сигнал-шум обрабатываемого сигнала. Шумоподавление происходит на основе последовательной блочной обработки сигнала. После буферизации входного сигнала выполняется оценка линейного спектра в блоке оценки спектра. На следующем шаге происходит сглаживание вычисленных спектров по времени и вычисление средней плотности спектра мощности. Затем, в следующем блоке, происходит оценка коэффициентов фильтра Винера, используя оценку спектра на текущем фрейме и оценку спектра шума. Оценка спектра шума происходит в блоке детектора речи (VADNest). Полученные линейные коэффициенты фильтра сглаживаются вдоль оси частот гребенкой фильтров. Результатом этой операции являются коэффициенты, распределенные по шкале мелов. В следующем блоке посредством обратного косинусного преобразования вычисляется импульсный отклик такого фильтра. Входной сигнал фильтруется полученным фильтром. На самом последнем этапе происходит удаление постоянной составляющей очищенного от шума сигнала. На обоих стадиях сигнал проходит идентичную обработку за исключением блока усиления фильтра на второй стадии шумоподавления.

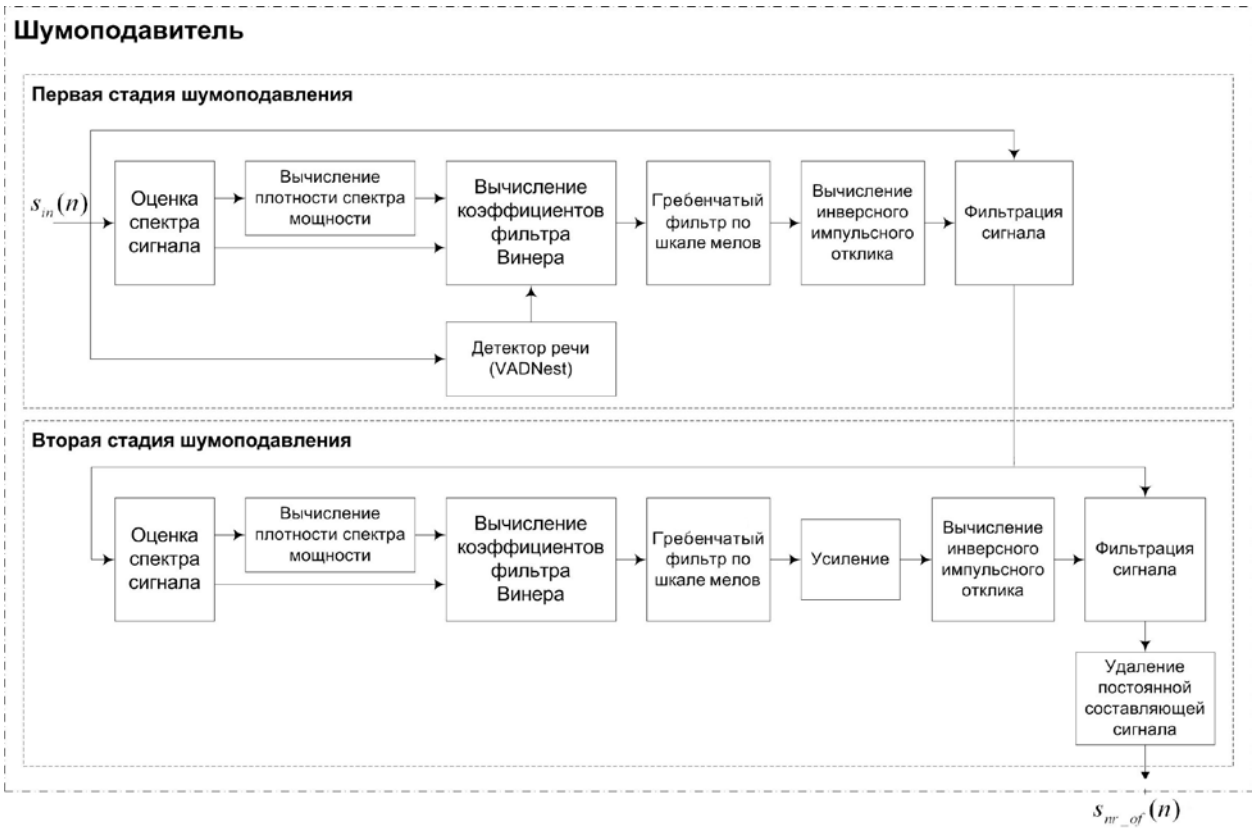


*Рис 5. Схема блока шумоподавления*

***Параметризация речевого сигнала (вычисление признаков).*** На этом этапе происходит параметризация речевого сигнала гребенкой из 16-ти полосовых кепстральных фильтров распределенных по шкале мелов [14.]. Кепстральное представление хорошо зарекомендовало себя в системах распознавания речи [17], т.к. оно снижает размерность представления, достаточно информативно и адекватно моделирует источник речевого сигнала. Кроме того, для полноценного описания речевого сигнала в систему параметров включены компоненты, описывающие временную динамику сигнала. Эти компоненты представляют собой первую производную по времени от вычисленных кепстральных коэффициентов [18].

***Детектор речь/пауза.*** Детектор речь/пауза необходим для снижения вычислительной нагрузки на блок распознавания путем отбрасывания участков, не содержащих полезного сигнала. В системе распознавания используется детектор речь/пауза, основанный на технологии скрытых марковских моделей (СММ) [19]. Этот алгоритм, используя одну модель для шума и одну модель для речи, анализирует каждый фрейм входного сигнала и вычисляет логарифм правдоподобия для каждой из этих моделей. При превышении некоторого порога правдоподобия принимается решение о наличии речевого сигнала. Такой алгоритм демонстрирует приемлемое качество при временной сложности, которая позволяет использовать его в системах реального времени.

***Процедуры распознавания и сегментации (декодер Витерби).*** В этом блоке выполняется распознавание речевого высказывания, представленного конкатенацией СММ звуков. СММ состоит из марковской цепи с конечным числом состояний, которая моделирует временные изменения сигнала, и конечного множества выходных распределений вероятностей, которые позволяют моделировать спектральные вариаций сигнала. В качестве базовых единиц системы используются машинное представление звуков, для каждого из которых создана отдельная СММ, при этом произнесение каждого звука описывается последовательностью векторов спектральных характеристик речевого сигнала. С лингвистической точки зрения звуки, для которых строятся модели, соответствуют контексто-зависимых акустическим реализациям фонем. марковская модель слова из словаря системы получается конкатенацией элементарных моделей звуков.

В процессе распознавания неизвестного высказывания необходимо найти наиболее подходящую модель (то есть произносительную транскрипцию команды) $M^*$, которая максимизирует вероятность $P(M^* \mid X, \Theta)$ при фиксированном множестве параметров модели $\Theta$ и наблюдаемой в данный момент последовательности векторов спектральных характеристик речевого сигнала *X*::

$$M^* = \arg\max_{M \in \Re} P(M \mid X, \Theta)$$

Метод нахождения наилучшей модели основан на использовании модифицированной процедуры Витерби [16].

## Вычислительная платформа системы распознавания команд

С точки зрения операционной сложности система представляет собой комплекс разнородных вычислительных схем, сочетающих в себе, как чисто вычислительные алгоритмы (например, предобработка, шумоподавление, параметризация речевого сигнала), так и специальные алгоритмы – поиск, работа с таблицами, деревьями, списками и другими сложными структурами данных (например, процедуры распознавания и сегментации). Причем, эти специальные алгоритмы далеки от линейных и представляют структуры с множеством ветвлений, циклов и рекурсий. С точки зрения временной сложности система представляет собой систему реального времени. Высокая операционная сложность и жесткие ограничения по быстродействию системы требуют тщательного выбора вычислительной платформы, которая максимально соответствовала бы решаемой задаче.

Хотя постоянно происходит значительный рост производительности процессоров и появление новых вычислительных архитектур так же повышающих производительность процессоров, опережающими темпами растет операционная сложность алгоритмов. Поэтому, разработчикам встраиваемых систем постоянно приходится сталкиваться с недостатком вычислительной мощности, что приводит к необходимости заниматься оптимизацией системы с целью достичь ее максимального быстродействия. Такая оптимизация для системы распознавания команд состоит из двух частей. С одной стороны, система распознавания команд имеет целый ряд параметров, которые влияют на временную сложность используемых алгоритмов. Это частота дискретизации сигнала, длина окна и шаг окна обработки, размерности кодовых книг, количество моделей звуков, объём словаря и т.д. Управляя этими параметрами, можно значительно снизить требования к временной сложности алгоритмов. Однако, практически все эти параметры оказывают влияние на точность распознавания, причем снижение временной сложности ведет к снижению точности распознавания. Таким образом, при проектировании встроенной системы распознавания необходимо находить оптимальный набор параметров, чтобы достичь требуемой точности с минимальными требованиями к временной сложности. С другой стороны, стремясь повысить производительность процессоров, разработчики используют специфические вычислительные архитектуры, к которым, зачастую, очень чувствительны вычислительные схемы применяемых алгоритмов и используемых структур данных. Это, прежде всего, размер адресуемого слова, размер кэша программ и данных, организация вычислительного конвейера, система команд и т.д. Поэтому при проектировании системы необходимо учитывать архитектурные особенности используемого процессора и в соответствии с этим строить схему вычислений, т.е. выполнять оптимизацию программного кода. В этой ситуации существенно возрастает важность используемых компиляторов в процессе разработки системы, которые облегчают оптимизацию программного кода при выполнении на определенном процессоре. Использование хорошего компилятора не только может повысить производительность системы, но и сделать ее более мобильной. К сожалению, пока качество оптимизации компиляторов оставляет желать лучшего. Поэтому программистам приходится выполнять оптимизацию «вручную». Существуют лишь самые общие принципы оптимизации, эффективность которых сильно зависит от архитектуры выбранной вычислительной платформы. Выработать общие принципы оптимального построения систем распознавания команд очень сложно.

Отмеченная выше разнородность вычислительных схем системы распознавания команд наводит на мысль об использовании двухпроцессорного или двухядерного

решения при выборе вычислительной платформы. На наш взгляд наиболее соответствующей вычислительной архитектурой для реализации системы распознавания команд во встроенном исполнении является двухядерная архитектура, в которой в одном кристалле интегрированы два вычислительных ядра. Обычно это сигнальный процессор и RISC (Reduced Instruction Set Computer) процессор. Типичный пример такой архитектуры это семейство процессоров OMAP591x или DaVinci фирмы Texas Instruments. На Рис. 6 приведена структурная схема процессора OMAP5912, которая взята из документации фирмы Texas Instruments.

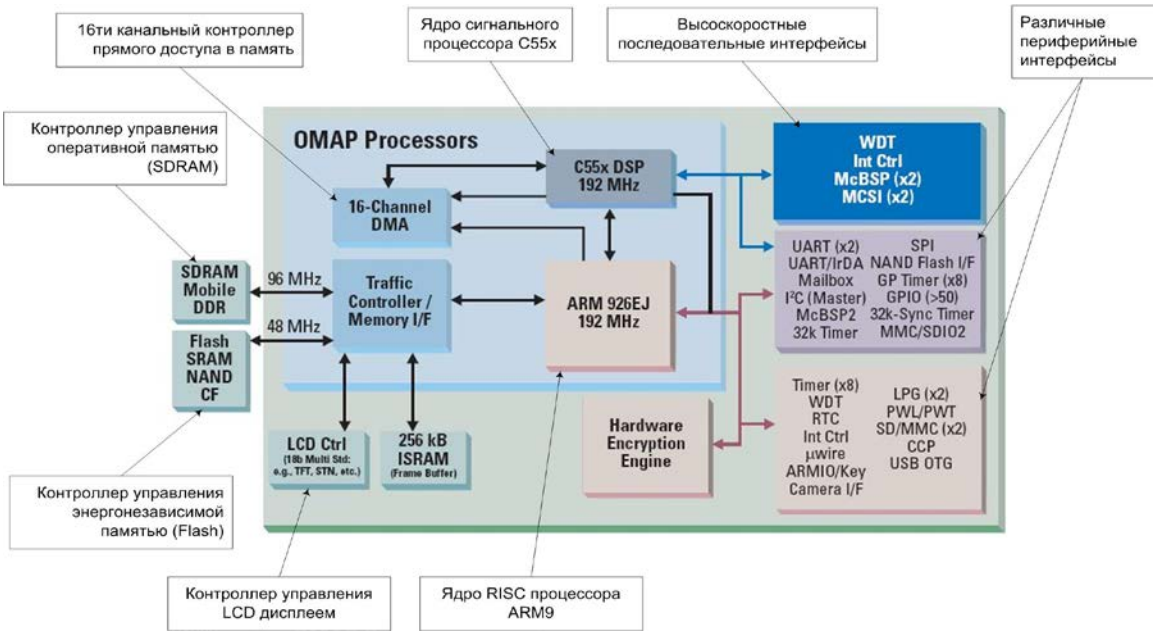


*Рис. 6 Структурная схема процессора OMAP5912*

Такие процессоры представляют собой однокристальный компьютер с интегрированным в одну микросхему широким спектром периферии, например, последовательным интерфейсом к аудио кодеку, контроллером дисплея и т.д. Это приводит к значительному уменьшению количества напаянных элементов, что повышает надежность работы изделия. Кроме того, использование такой архитектуры ОС приложения позволяет значительно увеличить производительность системы в целом за счет реализации параллельных вычислений и использования для разнородных вычислений соответствующих процессоров. Структурная схема реализации системы распознавания команд в таком одноплатном исполнении выглядит, как показано на Рис. 7. Изображенная на Рис. 7 схема функционирует следующим образом. Речевой сигнал с микрофона поступает на аудио кодек системы, который оцифровывает его и передает в вычислительный модуль, в котором он попадает на высокоскоростные последовательные

интерфейсы сигнального процессора, с помощью которых происходит ввод сигнала в процессор. Последовательные интерфейсы управляются драйвером аудиокодека. Далее сигнальный процессор выполняет очистку оцифрованного сигнала от шума и параметризацию речевого сигнала (вычисляет кепстральные параметры). После этого, полученные векторы параметров квантуются с помощью кодовых книг. Результаты этих вычислений передаются на RISC ядро, где они попадают на детектор речь/пауза, который выделяет из потока сигнала участки, содержащие речевые высказывания, и передает их на декодер Витерби, в котором происходит сегментация и распознавание высказывания. Результат распознавания передается в декодер команды/сигнал, который

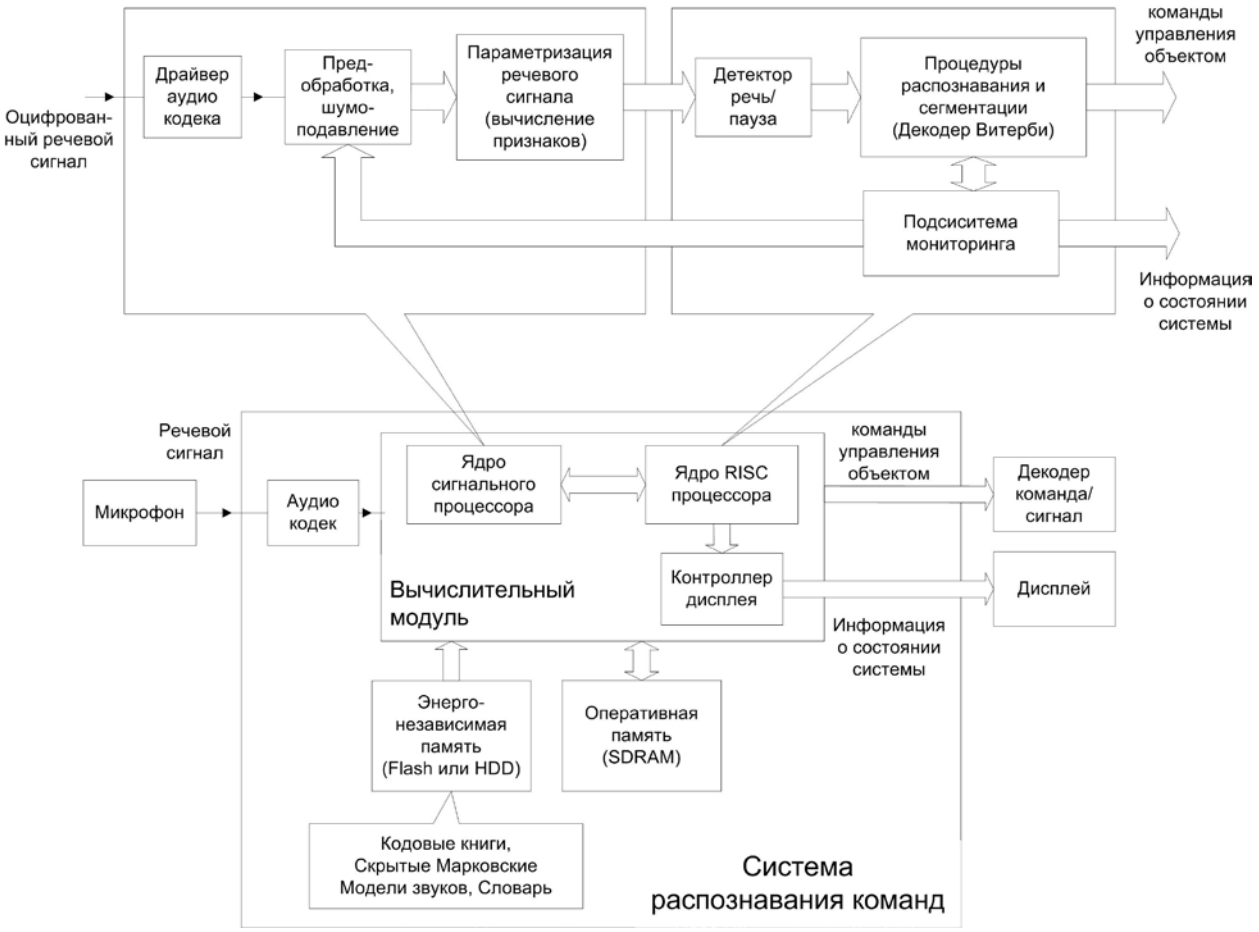


*Рис. 7. Схема реализации системы речевого управления на двухядерной вычислительной платформе.*

перекодирует полученные распознанные речевые команды в последовательность управляющих воздействий на объект управления. При этом, сама программа распознавания, кодовые книги, СММ модели звуков, словарь системы должны располагаться в энергонезависимой памяти, например, на жестком диске или флэш

памяти. Кроме того, система имеет дисплей обратной связи и вспомогательный интерфейс, который необходим для смены словаря, настройки параметров и т.п.

По нашим оценкам, при использовании такой двухъядерной или двухпроцессорной архитектуры, без учета затрат на ОСРВ, вычислительная платформа должна обладать следующими ресурсами:

| Тип ресурса | Величина ресурса |
|---|---|
| *Сигнальный процессор* | |
| Производительность | 50-200 MIPS |
| Размер оперативной памяти | 2 Мбайта |
| Размер флэш памяти | 4 Мбайта |
| *RISC процессор* | |
| Производительность | 100-200 МГц |
| Размер оперативной памяти | не менее 16 Мбайт |
| Размер флэш памяти | 32 Мбайта |

**Среда ОСРВ системы распознавания команд**

Чтобы соответствовать стандарту ARINC 653, ППО должно быть способно работать с использованием только тех ресурсов операционной системы, которые описаны в стандарте APEX. Если речь идёт о неописанном поведении или поведении, зависящем от реализации, то чтобы соответствовать данному стандарту, ППО должно согласовываться с любым поведением, которое может быть у операционной системы, соответствующей данному стандарту. Что касается символьных констант, то чтобы соответствовать данному стандарту, ППО должно воспринимать любые значения, допускаемые этим стандартом; оно не должно запрашивать констант, выходящих за этот минимум, даже если реализация системы позволяет их использовать.

Интерфейс ППО, описанный в соответствующем разделе выше, выполняет операции, необходимые для поддержания базовой многозадачной работы системы. Включение дополнительных сервисов, соответствующих конкретным системам, не обязательно будет являться нарушением этого стандарта, но делает ППО, использующее эти дополнительные сервисы, менее переносимым.

Архитектура, лежащая в основе раздела, сходна с архитектурой многозадачных приложений в главных компьютерах коллективных систем общего назначения. Каждый

раздел состоит из одного или нескольких параллельно выполняемых процессов, совместно осуществляющих доступ к ресурсам процессора в зависимости от требований прикладной программы. Каждый процесс имеет исключительные идентификационные признаки или атрибуты, управляющие диспетчеризацией, синхронизацией и общим выполнением программы. Библиотечные процедуры требуется включить в программный код прикладных программ, чтобы соблюсти требования отказоустойчивого разделения на разделы.

Прикладная программа, отправляющая сообщение другой прикладной программе или получающая сообщение от другой прикладной программы, не содержит явных данных о положении своего собственного раздела или раздела своего адресата. Информация, необходимая для правильной маршрутизации сообщения от источника до точки назначения, содержится в конфигурационных таблицах, которые разрабатываются и обновляются системным интегратором, а не разработчиком отдельных прикладных программ. Системный интегратор конфигурирует среду таким образом, чтобы обеспечивалась правильная маршрутизация сообщений между разделами базового модуля и между базовыми модулями.

Раздел должен быть способен предупредить нормальное обновление диспетчеризации процессов операционной системой, если его процессы достигают критических сегментов или ресурсов, совместно используемых несколькими процессами одного и того же раздела. Эти критические сегменты могут быть определёнными областями памяти, какими-то физическими устройствами или просто обычными вычислениями и действиями какого-то процесса.

Монитор состояния (НМ) активируется, если ППО вызывает сервис ошибки в приложении, а также, если операционная система или аппаратные средства обнаруживают отказ. Действия по восстановлению работоспособности системы после отказа зависят от уровня ошибки. Действия по восстановлению работоспособности системы в случае ошибок уровня модуля и раздела описываются системным интегратором в таблицах НМ модуля и в таблицах НМ раздела. Действия по восстановлению работоспособности системы в случае ошибок уровня процесса описываются разработчиком ППО в специальном процессе обработки ошибок.

Процесс обработки ошибок представляет собой специальный апериодический процесс, имеющий самый высокий уровень приоритета, который запускается операционной системой при обнаружении ошибки уровня процесса (например, при нарушении срока завершения, цифровой ошибке, переполнении буфера). Он выполняется

во время окна раздела. Он не может быть приостановлен или остановлен другим процессом. Его приоритет не может быть изменён. Он может прервать по приоритету любой выполняющийся процесс независимо от его уровня приоритета, даже если прерывание по приоритету дезактивировано (уровень блокировки не равен нулю). Если процесс обработки ошибок не создан, то производятся действия по восстановлению работоспособности системы, указанные в таблице НМ раздела.

Процесс обработки ошибок пишется разработчиком ППО. Он может остановить и перезапустить процесс, в котором произошёл отказ, перезапустить целый раздел или отключить раздел. Однако процесс обработки ошибок не может использоваться для исправления ошибок (например, ограничить значение в случае переполнения). Процесс, в котором произошёл отказ, может быть продолжен, только если речь идёт об ошибке ППО или о нарушении срока завершения. Процесс обработки ошибок не может вызывать сервисы, связанные с блокировкой. Он может передать информацию о контексте ошибки на функцию НМ для проведения обслуживания. Идентификатор процесса, в котором произошёл отказ, код ошибки, адрес ошибки и сообщение об отказе могут быть выведены. Процесс обработки ошибок отвечает за передачу сведений об отказах. Если во время выполнения процесса обработки ошибок произойдёт какой-то отказ (например, нарушение памяти, отказ операционной системы), то процесс обработки ошибок не сможет быть надёжно выполнен в контексте данного раздела. В этом случае автоматически начинает выполняться действие по восстановлению работоспособности системы, предусмотренное таблицей НМ раздела.

Переносимость прикладных программ может быть обеспечена, если они написаны независимо от реализации аппаратных средств и их конфигурации. То есть в ППО должны максимально использоваться стандартные сервисы. Также в ППО следует по возможности избегать и других видов зависимости от реализации других разделов.

Для конфигурирования раздела в комплексное приложение интегратор должен иметь по разделу, как минимум, следующие данные: требования к памяти, период, продолжительность, тип сообщений, которые будут отправляться разделом, тип сообщений, который будут приниматься разделом

Учитывая, все перечисленные требования предъявляемы к ППО, работающему под управлением ОСРВ (например, LynxOS-178), реализация системы распознавания речевых команд с эмулированной ОС раздела выглядит, как показано на Рис. 8. Приложение системы распознавания команд функционирует как один раздел системы в виде четырех процессов:

1. Монитор состояния системы. Процесс, который является агентом системного контроля состояния (HM). Главной задачей этого процесса является контроль за корректной работой всех компонентов системы распознавания. Другими словами, этот процесс должен контролировать правильность и время завершения всех процессов раздела, т.е. например, контролировать время заполнения буфера речевого сигнала и если оно превысит допустимые пределы, то фиксировать сбой при вводе речевого сигнала. Или, например, при превышении времени, отведенного на распознавание команды, либо прекращать поиск наилучшей гипотезы, либо вызывать перезапуск процесса распознавания, а если и это невозможно, то инициировать перезагрузку всего раздела, в котором находится система распознавания.

2. Распознавание речи. Процесс, который, собственно, реализует систему распознавания речевых команд. Внутри этого процесса выполняются два потока. Первый, осуществляет ввод речевого сигнала через драйвер аудиокодека, вычисляет параметры и выполняет шумоподавление. Второй, выполняет распознавание параметризованного речевого сигнала на основе имеющихся моделей звуков и словаря.

3. Формирование команд управления объектом по результатам распознавания речевого высказывания. Этот процесс выполняет функцию преобразования результата распознавания в сигналы управления.

4. Отображение текущего состояния. Задачей этого процесса является формирование обратной связи с пилотом, а именно отображением информации на дисплее о правильности и качестве (степени «уверенности») распознавания.

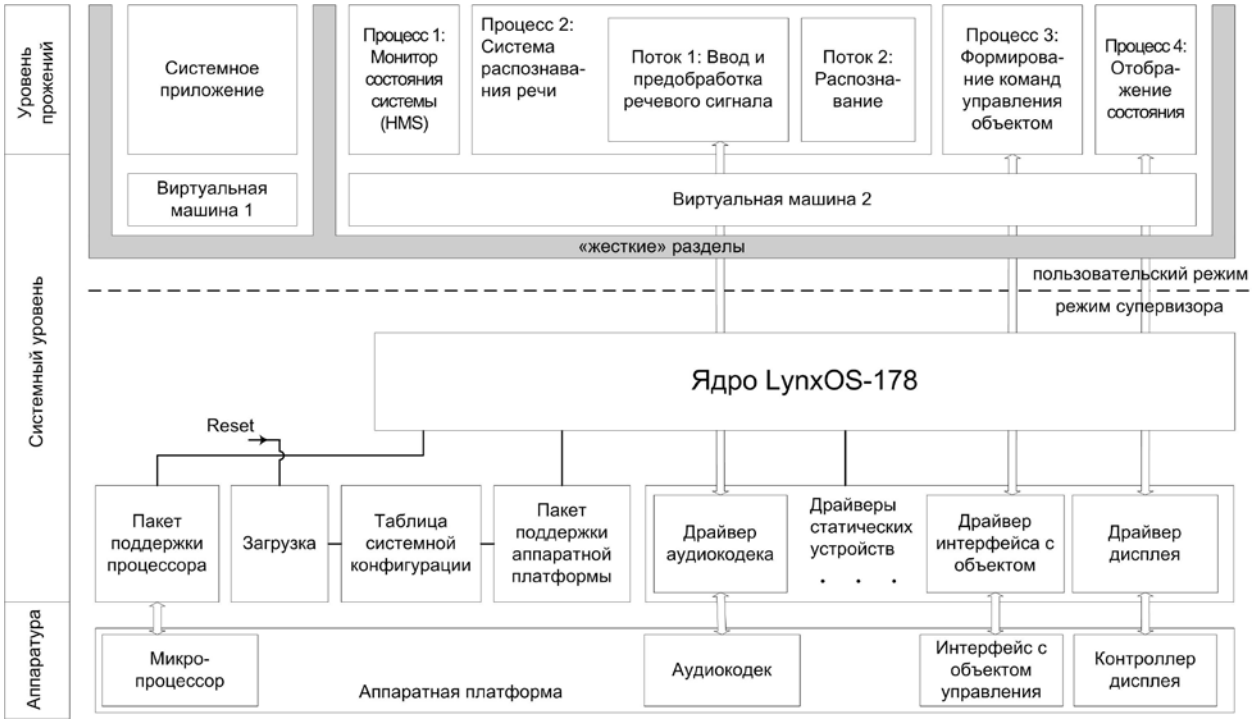

*Рис. 8. Схема реализации системы речевого управления с эмулированной ОС раздела*

## Заключение

Архитектуры и стандарты ИМА находятся еще в стадии развития, что представляет существенные трудности для разработчиков ППО. Создание платформы по ARINC 653 с интерфейсом, имеющим спецификации на основных языках высокого уровня (C, Visual C++, Delphi) обеспечит коммерческого разработчика операционной системой реального времени и основными инструментальными средствами, необходимыми для успешной разработки критичных по безопасности приложений ИМА, освобождая его от проблем отслеживания стандартов.

Анализ текущего состояния технологий, используемых при создании микропроцессорных систем, показал, что на сегодняшний день существует широкий спектр мощных вычислительных архитектур, которые позволяют реализовывать сложные алгоритмы системы речевого управления во встроенном варианте.

## ЛИТЕРАТУРА

1. RTCA DO-297 Integrated Modular Avionics (ИМА) Development Guidance and Certification Considerations. (Интегрированная модульная авионика (ИМА). Руководство по разработке и сертификации.). RTCA Inc.

2. RTCA DO-254 Design Assurance Guidance for Airborne Electronic Hardware. ( Руководство по гарантированному обеспечению проектирования бортового электронного оборудования.). RTCA Inc.

3. КТ-254 Квалификационные требования к проектированию и сертификации электронной аппаратуры.

3 RTCA DO-255, "Requirements Specification for Avionics Computer Resource (ACR)," RTCA Inc.

4 ARINC Specification 653, "Avionics Application Software Standard Interface," January 1, 1997, ARINC

5. ARINC Report 665-3. Loadable Software Standards. ( Отчет ARINC 665-3.Стандарты загружаемого программного обеспечения).

6 Parkinson P. Safety Critical Software Development for Integrated Modular Avionics. White Paper, Wind River Systems.

7 POSIX Specification, ANSI/IEEE POSIX 1003.1-1995 и стандарт ISO/IEC 9945-1:1996.

8 ISO/IEC 15408: 1999 "Information technology - Security techniques - Evaluation criteria for IT security."
9 DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," RTCA.
10. КТ-178А, КТ-178В. Требования к программному обеспечению бортовой аппаратуры и систем при сертификации авиационной техники.
11. ГОСТ Р ИСО МЭК 51904-2002. Программное обеспечение встроенных систем. Общие требования к разработке и документированию.
12. MIL-STD-498. Software Development and Documentation.
13. LynxOS-178 – коммерческая ОСРВ для авиации. - PC Week/RE, No22, 2005, с. 1- 4
14. Joseph W. Picone, “Signal Modeling Techniques in Speech Recognition”, Proceedings of the IEEE, Vol. 81, No.9, pp.1215-1247, September 1993.
15. Design and implementation of the Russian telephone speech database. In Proc. of Int. Workshop “Speech and Computer”, SPECOM’99, Moscow, 1999, pp.179-181.
16. L.R. Rabiner, «A tutorial on hidden Markov models and selected application in speech recognition», Proceedings of the IEEE vol. 77, 2, 1989, pp. 257-286. Русский перевод: Л.Р. Рабинер, «Скрытые марковские модели и их применение в избранных приложениях при распознавании речи: Обзор» ТИИЭР том 77, № 2 февраль 1989, стр. 86-120
17 .Steven Davis and Paul Mermelstein, “Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences”, IEEE Trans. Acoustics, Speech, and Signal Processing, vol. 28, No. 4, pp. 357-366, August 1980.
18. Young S., Jansen J., Odell J., Ollason D. & Woodland P. «HTK - Hidden Markov Toolkit», Entropic Cambridge Research Laboratory, 1995
19. Acero, A., Crespo, C., De la Torre, C., Torrecilla, J.C, 1993. «Robust HMM-based endpoint detector». In: Proceedings of the EuroSpeech'93, Berlin, 21-23 September 1993, Vol. 3, p.p. 1551-1554.

Автор: Бондарос Юлия Григорьевна.
Ст. научн. сотр. ФГУП ГосНИИАС

_____

“____”_____2006 г.

Рабочий адрес:
Россия, 125319, Москва, ул. Викторенко, дом 7.
Тел.: 157-93-21
Факс: 157-50-97
E-mail: bondaros@gosniias.ru

Домашний адрес:
Россия, 121609, Москва, ул. Осенний бульвар, дом 15. кв. 212.
Тел.: 413-06-10

Паспорт: 45 02 616437
Выдан 21.03.2002 ОВД “Крылатское” гор. Москвы

Автор: Маковкин Константин Александрович.
Мл. научн. сотр. ВЦ им. А.А Дородницына РАН

_____

“____”_____2006 г.

Рабочий адрес:
Россия, 119991 г. Москва, ГСП-1, ул. Вавилова, д. 40.
Тел.: 137-13-27
Факс: 135-61-59
E-mail: makovkin@ccas.ru

Домашний адрес:
Россия, 109388 г. Москва, ул. Гурьянова, дом 79, кв.64.
Тел.: 354-89-31

Паспорт: 45 06 709844
Выдан 04.11.2003 ОВД района Печатники г. Москвы
код подразделения: 772-080

Автор: Чучупал Владимир Яковлевич.
Нач. сектора ВЦ им. А.А Дородницына РАН

_____

“____”_____2006 г.

Рабочий адрес:
Россия, 119991 г. Москва, ГСП-1, ул. Вавилова, д. 40.
Тел.: 137-13-27
Факс: 135-61-59
E-mail: chuchu@ccas.ru

Домашний адрес:
Россия, 105523 г. Москва, ул. Плеханова д.29, к.1, кв.112.
Тел.: 652-51-82

Паспорт: 4599 863621
Выдан 22.03.2000 ОВД Северное Измайлово г. Москвы
код подр. 772-047