МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

А.О. Ключев, Д.Р. Ковязина,
П.В. Кустарев, А.Е. Платунов

АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА ВСТРАИВАЕМЫХ СИСТЕМ

Учебное пособие

Санкт-Петербург

2010

Ключев, А.О., Ковязина Д.Р., Кустарев, П.В., Платунов, А.Е. Аппаратные и программные средства встраиваемых систем. Учебное пособие [Текст] / А.О. Ключев, П.В. Кустарев, А.Е. Платунов. – СПб.: СПбГУ ИТМО, 2010. – 290 с.

Учебное пособие является введением в проблематику организации аппаратных и программных средств встраиваемых вычислительных систем (ВВС). Рассматриваются базовые понятия и основные характеристики ВВС, элементная база, аппаратные и программные средства, используемые для простроения ВВС, особенности реализации и программирования современных контроллеров ВВС на примере учебного контроллера SDK-1.1.

Для подготовки бакалавров по направлению 230100 «Информатика и вычислительная техника», магистров по программам 230100.68.13 «Сети ЭВМ и телекоммуникации», 230100.68.31 «Проектирование встроенных вычислительных систем», 230100.68.32 «Системотехника интегральных вычислителей. Системы на кристалле».

Рекомендовано к печати ученым советом факультета КТиУ, протокол №11 от 08.06.2010 г.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

Оглавление

Введение

Данная книга является результатом обобщения материалов учебных курсов, связанных с микропроцессорной техникой и информационно-управляющими системами, которые читаются на кафедре вычислительной техники СПбГУ ИТМО в течение ряда лет. Первая часть книги базируется на курсе лекций «Информационно-управляющие системы» и содержит обзорный материал, позволяющий понять основные особенности встраиваемых систем. Во второй части книги подробно рассматривается типичный контроллер на примере учебного микропроцессорного стенда SDK-1.1, дано описание основных блоков, разобраны примеры программ, описан инструментарий для программирования. При описании различных блоков, мы старались вначале делать общее описание (принцип работы, концепцию) и только затем описывать особенности конкретной реализации.

Целью создания данной книги явилась необходимость собрать в одном издании обширную и разнородную информацию о типовых аппаратных и программных решениях в области построения встраиваемых вычислительных систем, полезную студентам при изучении курса «Информационно-управляющие системы», а также специальных курсов магистерской подготовки.

Хочется заметить, что эта книга не рассматривается в качестве единственного пособия для изучения курса студентами. Настоятельно рекомендуется изучение дополнительной литературы.

Авторы надеются, что студенты, изучающие данное пособие, стремятся стать высококвалифицированными специалистами в области вычислительной техники. Для того чтобы стать таким специалистом мало знать конкретные технологии, мало уметь программировать, нужно *научиться получать новые знания*, понимать *суть предмета* и выстроить у себя в голове *систему знаний*. К сожалению, в современной литературе по вычислительной технике в основном освещаются практические вопросы, даются рецепты «Как сделать?», не объясняя никаким образом, почему именно так следует это делать. Мы рекомендуем ряд книг, которые содержат по большей части именно суть и концепции, а не только практические рекомендации и примеры.

По вопросам организации современных вычислительных устройств мы рекомендуем книгу «Архитектура компьютера» Таненбаума [49]. Тем, кто хочет глубже изучить язык программирования Си, необходимый для выполнения ряда лабораторных работ, можно рекомендовать книги Кернигана и Ричи «Язык программирования Си» [39] и Кернигана и Пайка «Практика программирования» [38]. Для более глубокого изучения языков программирования, их отличий и принципов построения, рекомендуется знакомство с книгами Н.Н. Непейводы [42, 43] и Роберта У. Себесты [47]. Для

того чтобы понять особенности проектирования систем и научиться эффективно управлять проектами[1] рекомендуем книги Демарко [32, 33] и Фредерика Брукса [27]. О принципах создания сложных систем можно узнать из книги Г. Буча «Объектно-ориентированный анализ и проектирование с примерами на языке C++» [28].

Материал данного курса, безусловно, выходит за рамки области встраиваемых вычислительных систем. Необходимо понимать, что в основе вычислительной техники лежит сравнительно небольшой объем фундаментальных знаний, которые являются универсальными и применимы в различных областях. Вам нужно попытаться добраться именно до таких знаний.

[1] Не забывайте, что вы будущие руководители!

1 Введение во встраиваемые вычислительные системы

Данная глава посвящена обзору базовых понятий и особенностей, аппаратно-программных механизмов обеспечения свойств встраиваемых вычислительных систем.

1.1 Определения, особенности, классификация

1.1.1 Встраиваемые системы

Зарождение встроенных систем происходило в начале пятидесятых годов. В то время, компьютеры выполнялись на громоздкой элементной базе, были крайне ненадёжны. Для нормальной работы таким машинам требовались идеальные условия эксплуатации. Класс вычислительных систем, предназначенных для управления и максимально оторванных от объекта управления, называли информационно-управляющими системами (ИУС). С появлением компьютерных сетей примерно в 70-х годах, появилась возможность строить распределенные или сетевые ИУС. Появление интегральных микросхем, а также микропроцессоров дало возможность приблизить ИУС непосредственно к объекту управления, или даже встроить в него ЭВМ. Так появились первые встроенные системы (Embedded System). Постепенно, по мере удешевления элементной базы и увеличения степени её интеграции и увеличения надёжности вычислительных устройств появилась возможность устанавливать ЭВМ в разные места объекта управления, объединяя все вычислительные узлы в единую контроллерную сеть. В процессе дальнейшего развития, благодаря еще большей миниатюризации и диффузии с объектом управления появились так называемые киберфизические системы, (CPS, Cyber Physical System). CPS характеризуются глубоким сращиванием с механическими, оптическими, химическими и биологическими системами. Итак, по степени проникновения вычислительной системы в объект управления можно выделить:

- Информационно-управляющие системы (ИУС).
- Распределенные информационно-управляющие системы (РИУС).
- Встроенные системы (Embedded System, ES).
- Сетевые встроенные системы (Networked Embedded System, NES).
- Киберфизические системы (Cyber Physical System, CPS).

В последнее время, из-за прогресса в области вычислительной техники, смысл термина встроенная система достаточно сильно видоизменился и размылся. По мере развития техники происходила эволюция обозначения класса управляющих компьютерных систем: от информационно-управляющей системы к встроенной, от встроенной к встроенной сетевой, а от встроенной сетевой – киберфизической. В процессе развития, происходила плавная интеграция вычислительной системы и объекта управления. Если первые информационно-управляющие системы представляли собой систему,

практически не связанную с объектом управления, то современные киберфизические системы очень и очень тесно интегрированы с объектом управления.

Киберфизическая система Cyber Physical System, (CPS)– специализированная вычислительная система, имеющая физические средства взаимодействия (электрические, химические, оптические, механические, биологические и т.п.) с объектом контроля и управления, выполняющая одну функцию. В качестве вычислительной платформы для реализации киберфизической системы может исапользоваться любое компьютерное оборудование, включая оборудование класса SOHO (например, персональный компьютер или КПК) [11, 12, 14].

Существует множество определений термина «встроенная система» (embedded system), приведем некоторые из них [5, 21]:

- Встроенные вычислительные системы (ВВС) – специализированные (заказные) вычислительные системы (ВС), непосредственно взаимодействующие с объектом контроля или управления и объединенные с ним единой конструкцией.
- Встроенная вычислительная система – специализированная информационно-управляющая система (ИУС) для выполнения определенного набора функций.
- Встроенная вычислительная система – любая система, которая использует компьютер как элемент, но чья основная функция не есть функция компьютера. Примеры ВВС: DVD-проигрыватель, светофорный объект, банкомат, паркомат и т.д.
- Встраиваемой системой можно считать любую вычислительную систему, которая не является ПК, портативным компьютером (laptop) или большим универсальным компьютером (mainframe computer).
- Встроенная вычислительная система – устройство, которое включает в себя программируемый компьютер, но не является при этом компьютером общего назначения.
- Встроенная вычислительная система – практически любая вычислительная система, не являющаяся настольным компьютером.
- Встроенная система – система специального назначения, в которой вычислительный элемент полностью встраивается в устройство, которым она управляет. В отличие от универсального компьютера, встроенная система выполняет одну или несколько предопределенных задач, обычно с очень конкретными требованиями. В техническом смысле встроенная система взаимодействует с окружающей средой контролируемым образом, удовлетворяя ряд требований на способность реагировать в смысле качества и своевременности. Как правило, она должна удовлетворять требованиям реализации, таким как стоимость, потребляемая мощность и использование ограниченных физических

ресурсов. В идеале она должна взаимодействовать со средой в течение всей жизни объекта.

Как правило, встроенная система является частью более крупной системы или встраивается непосредственно в объект управления. Встроенные системы – это системы «глубоко интегрированные» с объектами физического мира [21]. Их элементы практически всегда ограничены по ресурсам. Это системы длительного жизненного цикла, часто автономные. Масштаб этих систем по размерам и сложности меняется в очень широких пределах. Эти системы рассчитаны на непрофессиональных пользователей и вместе с тем, часто выполняют критически важные функции.

Встраиваемые вычислительные системы можно классифицировать:

- по области применения/назначению;
- по различному соотношению информационных и управляющих функций, т.е. система преимущественно информационная (система сбора данных) или управляющая (система автоматического управления);
- по пространственной локализации аппаратных блоков:
  a) пространственно локализованные;
  b) пространственно рассредоточенные.
- по различному соотношению вычислительной (обработка данных) и коммуникационной (функция ввода-вывода данных) составляющей;
- по степени участия человека:
  a) автоматические системы – системы, в которых оператор выполняет только функции начальной настройки и оперативной корректировки параметров и режимов работы системы. Функции сбора данных, передачи и исполнения команд управления, оперативной выработки команд управления происходят без участия человека;
  b) автоматизированные системы – системы, в которых оператор частично или в полном объеме обеспечивает оперативную обработку данных и формирование команд управления исполнительными устройствами (например, телеуправление).
- по организации обработки данных/вычислений (централизованные/ децентрализованные);
- по распараллеливанию на уровне задач и/или функций между физическими/логическими модулями системы.

Особенность работы встроенной системы состоит в наличии необходимости работы в реальном масштабе времени (или просто в реальном времени).

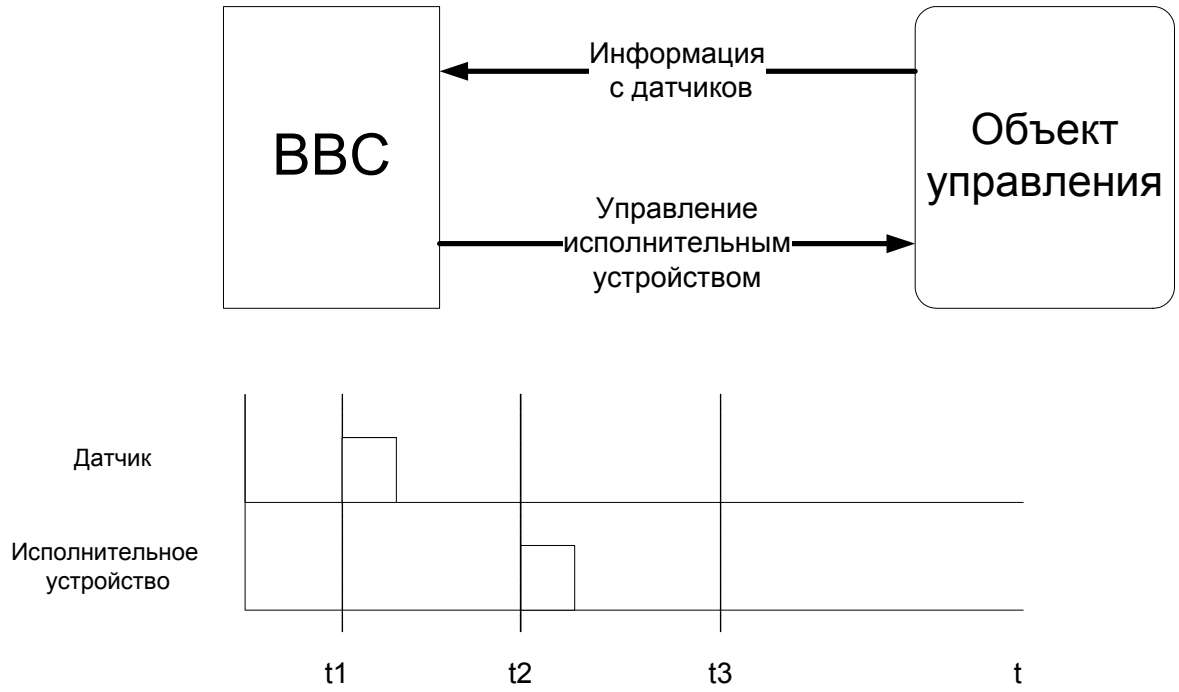


Рисунок 1. Работа в реальном масштабе времени

На рисунке показано три времени: t1 – время получения сигнала с датчика, t2 – время выдачи управляющего воздействия на исполнительное устройство, t3 – крайний срок выдачи управляющего воздействия. Если по какой-либо причине выдача управляющего сигнала задержится, сигнал будет выработан после t3, управляющий сигнал будет бесполезен или даже вреден. В качестве пример рассмотрим систему управления стеклоподъемником в автомобиле. Если ВВС игнорирует сигнал датчика положения стекла, либо стекло, либо подающий стекло механизм могут быть испорчены.

1.1.2 Примеры встраиваемых систем

Диапазон реализаций ВВС, действительно, очень велик. В него попадают и простейшие устройства уровня домашнего таймера, и сложнейшие распределенные иерархические системы, управляющие критически важными объектами на огромных территориях.

- Телекоммуникационные системы, сетевое оборудование (коммутаторы, маршрутизаторы, ADSL модемы и т.п.);
- Бытовая электроника (сотовые телефоны, КПК, игровые консоли, цифровые фотоаппараты, электрочайники, микроволновые печи, посудомоечные машины и пр.);
- Современное медицинское и спортивное оборудование;
- Транспортная автоматика (от автомобильных до авиационных систем), авионика, системы управления городским дорожным движением;
- Системы телемеханики (системы управления наружным освещением, контроля и учета электроэнергии и других энергоресурсов, управления и мониторинга энергообъектов);

- Системы мониторинга, навигации, слежения, бортовые системы для военных и космических применений;
- «Умный дом» («интеллектуальное здание») на основе технологий сенсорных сетей.

Важно, что, проектируя ВВС, разработчик всегда создает специализированную вычислительную систему независимо от степени соотношения готовых и заново создаваемых решений. В сферу его анализа попадают все уровни организации системы. Он имеет дело не с созданием приложения в готовой операционной среде при наличии мощных и удобных инструментальных средств, а с созданием новой специализированной ВС в условиях жестких ограничений самого разного плана.

Безусловно, часть задач в области создания ВВС удается решать шаблонными способами, особенно если речь идет о развитии или модификации уже готовой системы. Но даже в этом случае требуется использование качественной вычислительной платформы, мощного специализированного инструментария, тщательная верификация и тестирование продукта.

Задачи создания ВВС, которые не укладываются по тем или иным причинам в рамки шаблонных решений, постоянно требуют совершенствования методов и средств проектирования.

1.1.3 Реальное время

Система реального времени – вычислительная система с гарантированным временем реакции на события [21].

Система реального времени (СРВ) – любая вычислительная система, в которой время формирования выходного воздействия является существенным. Примеры СРВ: управление технологическими процессами, встроенные вычислительные системы, кассовые торговые системы и т.д.

Принципиальное отличие информационных систем (Information Technology) от систем реального времени (real-time) в трактовке параметра «реакция вход-выход»: «The right answer late is wrong» («Правильный ответ поздно = неправильный»).

К особенностям встроенных систем относится необходимость обеспечения надежности, безопасности и гарантированного времени реакции. Соблюдения гарантированного времени ответа обычно называют работой в реальном времени.

ВВС получает информацию об объекте управления посредством датчиков. В ответ на полученную информацию ВВС вырабатывает управляющее воздействие и передает и объекту управления через устройство сопряжения с объектом. Время, протекающее между получением информации от объекта управления и выдачей сигнала управления от встроенной системы, мы назовем временем реакции.

Система реального времени не должна быть обязательно быстрой. Это распространенное заблуждение. Система реального времени должна выдавать управляющие сигналы в ответ на информацию, поступающую от датчиков в гарантированные промежутки времени.

По степени важности последствий несоблюдения времени реакции обычно выделяют две группы систем реального времени:

- Система мягкого реального времени;
- Система жесткого реального времени.

Система мягкого реального времени (soft real-time system) – задержки задаются средними величинами. Имеет место в организации бизнес-процессов и в торговле.

Система жесткого реального времени **-** это система реального времени, невыполнение временных ограничений в которой приводит к катастрофическим последствиям для целевой функции системы. Имеет место в случае военных и космических применений.

1.1.4 Надежность

Надежность – свойство объекта выполнять заданные функции, сохраняя во времени значения установленных эксплуатационных показателей в заданных пределах, соответствующим заданным режимам и условиям использования, технического обслуживания, ремонта, хранения и транспортирования [29].

Надежность является комплексным свойством, которое в зависимости от назначения объекта и условий его эксплуатации может включать в себя безотказность, долговечность, ремонтопригодность и сохраняемость в отдельности или определённое сочетание этих свойств как для объекта (здесь под объектом понимается определённое средство измерения), так и для его частей.

Показатель надежности количественно характеризует, в какой степени данному объекту присущи определенные свойства, обусловливающие надежность. Одни показатели надежности (например, технический ресурс, срок службы) могут иметь размерность, ряд других (например, вероятность безотказной работы, коэффициент готовности) являются безразмерными.

Живучесть – свойство объекта сохранять работоспособность (частично или полностью) при наличии дефектов или повреждений определенного вида.

Работоспособное состояние (работоспособность) – состояние объекта, при котором значения всех параметров, характеризующих способность выполнять заданные функции, соответствуют требованиям нормативно-технической и (или) конструкторской (проектной) документации.

Отказ – событие, заключающееся в нарушении работоспособного состояния объекта. Критерий отказа – отличительный признак или

совокупность признаков, согласно которым устанавливается факт возникновения отказа.

Типы отказов:

- Отказы функционирования (выполнение основных функций объектом прекращается, например, поломка зубьев шестерни);
- Отказы параметрические (некоторые параметры объекта изменяются в недопустимых пределах, например, потеря точности станка).

Природа отказов:

- Случайные, обусловленные непредусмотренными перегрузками, дефектами материала, ошибками персонала или сбоями системы управления и т. п.;
- Систематические, обусловленные закономерными и неизбежными явлениями, вызывающими постепенное накопление повреждений: усталость, износ, старение, коррозия и т. п.

Основные признаки классификации отказов:

- Характер возникновения;
- Причина возникновения;
- Характер устранения;
- Последствия отказов;
- Дальнейшее использование объекта;
- Легкость обнаружения;
- Время возникновения.

Характер возникновения отказов:

- Внезапный отказ – отказ, проявляющийся в резком (мгновенном) изменении характеристик объекта;
- Постепенный отказ – отказ, происходящий в результате медленного, постепенного ухудшения качества объекта.

Внезапные отказы обычно проявляются в виде механических повреждений элементов (трещины – хрупкое разрушение, пробои изоляции, обрывы и т. п.) и не сопровождаются предварительными видимыми признаками их приближения. Внезапный отказ характеризуется независимостью момента наступления от времени предыдущей работы.

Постепенные отказы связаны с износом деталей и старением материалов.

Причина возникновения отказов:

- Конструкционный отказ, вызванный недостатками и неудачной конструкцией объекта;
- Производственный отказ, связанный с ошибками при изготовлении объекта по причине несовершенства или нарушения технологии;

- Эксплуатационный отказ, вызванный нарушением правил эксплуатации.

Характер устранения отказов:

- Устойчивый отказ;
- Перемежающийся отказ (возникающий/исчезающий). Последствия отказа – легкий отказ (легкоустранимый);
- Средний отказ не вызывает отказы смежных узлов (вторичные отказы);
- Тяжелый отказ вызывает вторичные отказы или приводит к угрозе жизни и здоровью человека.

По дальнейшему использование объекта отказы делятся на:

- Полные отказы, исключающие возможность работы объекта до их устранения;
- Частичные отказы, при которых объект может частично использоваться.

По легкости обнаружения отказы могут быть очевидными (явными) и скрытыми (неявными).

По времени возникновения отказы бывают:

- Приработочными, то есть возникающими в начальный период эксплуатации;
- Возникающими при нормальной эксплуатации;
- Износовые, то есть вызванные необратимыми процессами износа деталей, старения материалов и пр.

1.1.5 Распределенные встроенные системы

Распределенная встроенная система – пространственно рассредоточенная встроенная система. Такие системы характеризуются наличием слабой связи между компонентами.

Тенденция усложнения ВВС проявляется, прежде всего, в том, что большинство систем реализуются в виде многопроцессорных распределенных вычислительных систем или контроллерных сетей. Это дополнительно усложняет задачу проектировщика. Рассмотрим основные свойства современных распределенных ВВС.

Множество взаимодействующих узлов (более двух). Интерес сегодня представляют системы с единицами тысяч взаимодействующих встроенных компьютеров.

Работа в составе систем управления без участия человека. В таких системах оператор может присутствовать, он может получать информацию и частично иметь возможность воздействовать на работу системы, однако основной объем управления выполняет распределенная ВВС.

Степень функциональной и пространственной децентрализации управления может меняться в широких пределах. Вычислительные элементы ВВС выполняют задачи, отличные от задач вычислений и коммуникаций общего назначения. Распределенные ВВС используются в составе больших по масштабу технических объектов (например, самолет или здание) или взаимодействуют с объектами естественной природы (например, комплексы мониторинга окружающей среды).

Распределенные ВВС могут характеризоваться узлами с ограниченным энергопотреблением, иметь фиксированную или гибкую топологию, выполнять критичные для жизнедеятельности человека функции, требовать высокотехнологичной реализации или создаваться как прототип.

1.1.6 Пирамида автоматизации

Пирамида автоматизации (Computer Integrated Manufacturing pyramid) – модель, объединяющая все сферы деятельности современного предприятия в единую информационную среду [9, 10, 17].

Объединение всех уровней в единую систему позволяет:

1. Оптимизировать используемую информацию за счет централизации и упорядочивания потоков данных.
2. Интегрировать бизнес процессы, процессы управления материалами, разработками и производством в единую систему.
3. Получить доступ ко всем данным, существующим на предприятии для анализа с целью оптимизации управленческих процессов.

Результатом внедрения систем комплексной автоматизации является снижение себестоимости продукции, существенное увеличение качества и снижение длительности циклов производства. Количество уровней пирамиды автоматизации обычно колеблется от трех до пяти. Существуют разные трактовки одних и тех же терминов в различных системах. В трехуровневой модели CIM есть уровни стратегического планирования, тактического исполнения и управления операциями. В четырехуровневой модели есть уровни данных, управления, устройств и датчиков и исполнительных устройств. Мы рассмотрим CIM на примере пятиуровневой модели [17].

Верхние уровни пирамиды автоматизации часто строятся по принципам, применимым для обычных информационных систем. Не все системы содержат все пять уровней. Простые системы могут содержать в себе один или несколько нижних уровней пирамиды.

1.1.6.1 Уровень предприятия (1)

Верхний уровень отвечает за следующие вещи:

- Управление ресурсами (ERP, MRP, MRP II, JIT)
- Управление производством изделий (FMS)
- Управление продукцией (PDM)

- Планирование и прогнозирование (OLAP)
- Разработка новых изделий (CAD/CAM, CAE)

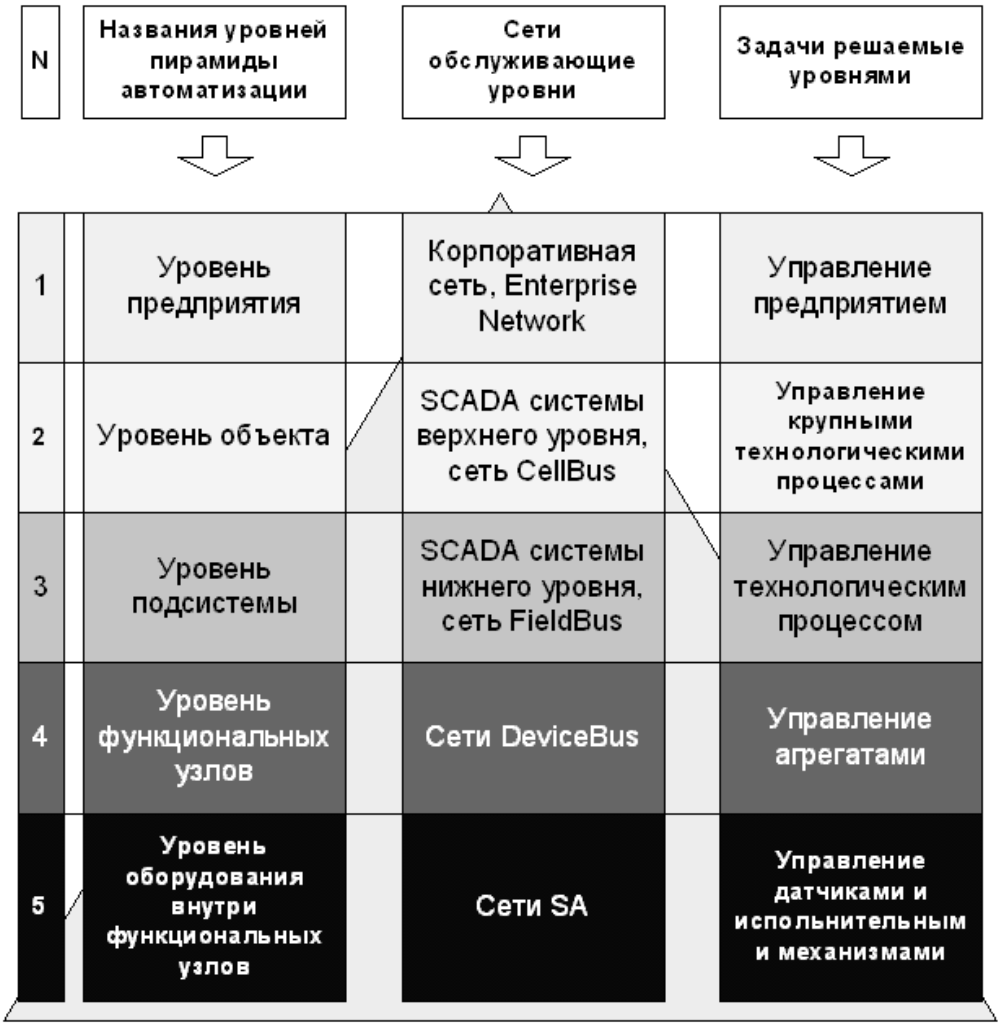


Рисунок 2. Пирамида автоматизации

На этом уровне мы имеем обычную корпоративную сеть (на базе Ethernet, HSE FIELDBUS, ATM и т.п.), состоящую из комплекса ЛВС общего назначения, файловых серверов, СУБД и различных систем автоматизации. Уровень характеризуется сравнительно низкими требованиями к надежности и безопасности к оборудованию. Возможно использование офисных компьютеров для рабочих станций и серверов. Оборудование функционирует в обычных офисах.

1.1.6.2 Уровни объекта (2) и подсистемы (3)

Следующие два уровня отвечают за управление технологическими процессами. Более верхний уровень отвечает за крупные промышленные объекты. Более мелкий – за сравнительно небольшие технологические процессы (составные части крупных). На небольших предприятиях эти два уровня сливаются в один уровень управления. На уровне объекта и подсистемы систему управления строят на базе сети CellBus. В большинстве случаев это обычный Ethernet или HSE FIELDBUS. В узлах сети CellBus находятся SCADA-системы (Supervisory Control and Data Acquisition). SCADA-системы уровня

объекта управляют SCADA-системами уровня подсистемы. Компьютерное оборудование этих двух уровней, как правило, находится в защищенных помещениях (например, в диспетчерских).

1.1.6.3 Уровень функциональных узлов (4)

Уровень функциональных узлов предназначен для управления агрегатами, т.е. некими сложными, составными механизмами. Главным элементом этого уровня являются логические контроллеры – вычислительные модули, управляющие, задающие логику функционирования и координирующие работу этих агрегатов (в конечном счете, объектов управления). В качестве логических контроллеров обычно выступают программируемые логические контроллеры (ПЛК или PLC), промышленные компьютеры и сетевые модули ввода-вывода. Алгоритм работы задается программным обеспечением логического контроллера. Программирование обычно осуществляется конечным пользователем – не разработчиком системы – на специальных «технологических» языках (например, на языке релейных схем, функциональных блоков и других). В пределах этого уровня используются сети, входящие в категорию DeviceBus. Связь со SCADA-системами осуществляется через сети Fieldbus.

1.1.6.4 Уровень оборудования функциональных узлов (5)

Самый нижний уровень предназначен для сбора информации с датчиков и управления исполнительными устройствами [25]. На этом уровне работают интеллектуальные устройства сопряжения с объектом (УСО) или сетевые модули ввода-вывода. В качестве сети используются сети SA Bus (Sensor Actuator). К оборудованию последних двух уровней предъявляются повышенные требования к надежности и безопасности. Контроллеры и сети находятся в непосредственной близости от объекта управления, подвержены действию различных помех, агрессивных сред и вибраций.

1.1.6.4.1 Устройства ввода-вывода

Обычно, в ВВС устройство ввода-вывода (УВВ) содержит некоторое количество дискретных и аналоговых входов и выходов.

Система ввода-вывода является тем блоком, ради которого обычно делается информационно-управляющая система. Если «лицом» офисной системы (если так можно выразиться) является дисплей с соответствующей графической оболочкой (десктопом), то «лицом» ИУС является система ввода-вывода. Системы ввода вывода могут быть весьма сложны по конструкции и дороги по цене. Очень часто, в системы ввода-вывода встраиваются дополнительные электронные, электрические и электромеханические устройства для защиты объекта управления или ИУС в различных нештатных ситуациях.

1.1.6.4.2 Устройство сопряжения с объектом

Устройства сопряжения с объектом (УСО), которые также называют модулями ввода-вывода, выполняют функции адаптера датчиков и исполнительных устройств. Они имеют специальные аппаратные каскады сопряжения с оконечными устройствами и поддерживают алгоритмы управления ими. УСО могут выполнять функции первичной обработки данных с датчиков: фильтрацию, усреднение и накопление. УСО являются подчиненными по отношению к логическим контроллерам и самостоятельно не реализуют каких-либо алгоритмов контроля объекта управления. По способу взаимодействия с логическим контроллером УСО делятся на:

- Локальные УСО, конструктивно совмещенные с логическим контроллером;
- Удаленные или сетевые УСО, взаимодействующие с логическим контроллером по сетевому каналу, конструктивно независимые от логических контроллеров.

В отличие от УВВ, УСО работает с реальным объектом в условиях помех, высоких напряжений и больших токов (например, управление двигателем переменного тока). Как правило, это устройство гальванически развязано с объектом управления. В качестве УСО могут выступать дискретные (битовые) входы и выходы, аналоговые входы и выходы и т.п. Без гальванической развязки управляющий вычислительный комплекс будет либо постоянно сбоить из-за помех, либо сгорит при появлении повышенного напряжения на входе УСО.

Устройство сопряжения с объектом является водоразделом, отделяющим «тепличный» мир вычислительного устройства от достаточно агрессивной окружающей среды. УСО должно не только сопрягать различные уровни сигналов, например 24 Вольт необходимые для включения реле и 5 Вольт (TTL уровень) на выходе битового порта ввода-вывода микроконтроллера, но и не пропускать высокие напряжения во внутренние цепи контроллера.

В некоторых системах (например, в медицинских) главным является сохранение объекта управления, т.к. от этого напрямую зависит жизнь человека. Такие системы тестируются на степень защиты человека от электрической травмы из-за некорректной работы УСО.

1.2 Механизмы реального времени

1.2.1 Таймер

Таймер позволяет производить отсчет временных интервалов заданной продолжительности. Принцип действия таймера основан на двоичном счетчике, с возможностью предварительной записи исходного значения. После каждого

такта синхросигнала счетчик прибавляет или отнимает единицу от имеющегося у него значения. При достижении нуля (то есть при переполнении), счетчик вырабатывает активный уровень на выходе. Как правило, выходной сигнал таймера заводят на вход запроса прерывания микропроцессора или контроллера прерываний.

В большинстве современных встроенных систем таймеры используются в качестве основы для организации системы разделения времени на базе переключателя задач. В данном случае таймер используется в паре с механизмом прерываний [5].

1.2.2 Устройство захвата-сравнения

Устройство захвата-сравнения предназначено для измерения периодов и длительностей импульсов, скважности, а также для генерации импульсов. С помощью этого модуля можно получить широтно-импульсную (ШИМ) и фазовую модуляцию (ФМ).

1.2.3 Сторожевой таймер

Допустим, в нашей системе есть некий процесс, выполняющий какую-либо полезную задачу. Назовем его прикладным или наблюдаемым процессом. Если процесс выполняет важную целевую функцию, то прекращение его деятельности может привести к остановке работы всей встроенной системы, то есть к сбою. Если не принять особых мер по обнаружению таких ситуаций, встроенная система может работать некорректно в течении достаточно долгого времени. Для вывода систем из сбойного состояния и приведения её в нормальный режим функционирования обычно используют сторожевой таймер.

Таким образом, сторожевой таймер является механизмом защиты системы от сбоев.

Рассмотрим работу сторожевого таймера подробнее. В работе участвуют три процесса: наблюдаемый прикладной процесс, процесс сторожевого таймера и служебный процесс, реализующий механизм защиты системы от аварийных ситуаций. Суть механизма сторожевого таймера состоит в проверке критерия, по которому можно определить, что наблюдаемый процесс работает нормально. Если сторожевой таймер обнаруживает, что с наблюдаемым процессом все в порядке, то ничего не происходит. Если сторожевой таймер определил, что с наблюдаемым процессом что-то не так, происходит передача информации системе разрешения аварийных ситуаций, которая, в свою очередь, принимает решение о дальнейшей судьбе наблюдаемого процесса.

В самом примитивном варианте в качестве сторожевого таймера выступает обычный вычитающий счетчик. Во время инициализации в счетчик записывается какое-либо значение. Если в процессе работы в счетчик эпизодически вносится новая константа, то ничего не происходит. Если же прикладной процесс не успевает записать константу, и счетчик успевает

досчитать до нуля, вырабатывается сигнал аппаратного рестарта и процессор перезапускается. Естественно, простой вариант реализации сторожевого таймера не является 100% гарантией от выхода системы из сбойного состояния.

Рассмотрим типичную ошибку, допускаемую программистами при использовании простого сторожевого таймера. Для того, чтобы контролируемый процесс прекратил работу, достаточно вставить код обновления счетчика сторожевого таймера внутрь цикла проверки, который будет всегда давать ложное значение из-за какой-либо ошибки (в аппаратуре или программе).

```
while( device_ready() == 0 ) reset_watchdog_timer();
```

Если функция device_ready всегда будет возвращать ноль, то мы бесконечное время будем сбрасывать сторожевой таймер, и система будет всегда находиться в состоянии сбоя.

Нужно заметить, что в большинстве микроконтроллеров реализована самая простая схема работы сторожевого таймера. Для её нормальной работы нужна дополнительная, сложная программная обработка.

Ещё одним неудобством простой схемы сторожевого таймера является выработка аппаратного сигнала рестарта (RESET). В большинстве случаев гораздо корректнее производить повторную инициализацию или рестарт одной или нескольких взаимосвязанных частей системы, а полный горячий рестарт осуществлять только при каких-либо фатальных сбоях.

1.2.4 Система прерываний

Прерывание в классической трактовке – прекращение выполнения текущей команды или последовательности команд для обработки некоторого события обработчиком прерывания, с последующим возвратом к выполнению прерванной программы.

Прерывание можно определить как механизм межпроцессного взаимодействия, включающий в себя механизмы передачи данных (тип прерывания) и управления (пуск/останов текущего процесса).

Система прерываний является неотъемлемой частью любой вычислительной системы и предназначена для обеспечения быстрой реакции процессора на ряд ситуаций, требующих его внимания, которое может возникать как в самом процессоре, так и за его пределами.

Прерывания следует рассматривать не только и не столько как реакцию процессора на аномальные ситуации, а как естественный процесс, с помощью которого реализуется поддержка большинства необходимых механизмов, таких как виртуальная память, ввод/вывод и т. п. По образному выражению Питера Нортона: "Прерывание – это движущая сила компьютера".

Система прерываний представляет собой комплекс аппаратных и программных средств. Аппаратные средства системы прерываний обычно называются блоком или контроллером прерываний. В ПК это PIC (Programmable Interrupt Controller), т.е. отдельная микросхема 8259A. В некоторых случаях контроллер прерываний интегрируется в кристалл микропроцессора. Управление контроллером прерываний осуществляется через регистры. Каждому прерыванию можно задать приоритет, численно определяющий важность события [21]. Программные средства систем прерываний представляют собой специальные программы – обработчики прерываний (interrupt handler). Как правило, адреса обработчиков располагаются в специальной таблице, так называемой таблице векторов прерываний.

Назначение системы прерывания – реагировать на определенные события путем прерывания работы процессора по выполнению программы и переключения процессора на выполнение другой программы, обслуживающей соответствующую ситуацию. В момент возникновения определенного события (причины) формируется сигнал прерывания, который поступает в процессор и инициирует специальную операцию – операцию прерывания, обеспечивающую прерывание одной программы и переключение процессора на выполнение другой программы.

1.2.4.1 Классификация прерываний

В зависимости от источника возникновения сигнала прерывания делятся на:

- асинхронные или внешние (аппаратные) — события, которые исходят от внешних источников (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, внешнего интерфейса, АЦП и других;
- внутренние — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода: деление на ноль или переполнение, обращение к недопустимым адресам или недопустимый код операции. Такого рода прерывания еще называются исключительными ситуациями (exceptions);
- программные (частный случай внутреннего прерывания) — инициируются исполнением специальной инструкции в коде программы. Программные прерывания как правило используются для обращения к функциям встроенного программного обеспечения драйверов и операционной системы.

Аппаратные прерывания могут возникать в произвольные моменты времени и являются асинхронными по отношению к выполняемой программе.

С помощью аппаратных прерываний осуществляется взаимодействие процессора с периферийными устройствами, а также сообщается о различных ошибках аппаратуры (например, ошибка памяти, ошибка передачи по шине и т.

п.) или об аварийном отключении питания. Во втором случае иногда такие прерывания называют исключительными ситуациями (exceptions).

Реагируя на аппаратное прерывание, процессор должен идентифицировать его источник, сохранить минимальный контекст прерываемой программы и переключаться на специальную программу – обработчик прерывания (interrupt handler), который может быть оформлен как процедура или задача.

Действия обработчика прерывания, называемые обслуживанием прерывания, заключается в том, чтобы правильно отреагировать на прерывание конкретного источника (например, поместить символ нажатой клавиши в буфер, произвести инкремент системных часов и т. п.).

После завершения обслуживания прерывания процессор возвращается к выполнению прерванной программы, и она должна продолжиться таким образом, как будто прерывания не было.

К основным ситуациям, возникающим вне процессора и приводящим к прерыванию, относятся:

- запросы от управляемого объекта (являются типичными для управляющих систем), т.е. запросы от ВУ:
  a) ВУ, готовые к обмену, требуют реакции процессора для организации программно-управляемой передачи данных;
  b) завершение работы ВУ или КВВ по передаче данных;
  c) особая (аварийная) ситуация в ВУ или КВВ.
- запросы прерываний от других процессоров для обеспечения синхронизации вычислительных процессов, протекающих в рамках многопроцессорной системы.

Программные прерывания, в отличие от аппаратных, появляются синхронно по отношению к выполняемой программе.

Причинами программных прерываний могут служить особые ситуации, возникающие при выполнении программы, препятствующие нормальному продолжению программы и требующие специального обслуживания (переполнение, нарушение защиты памяти, отсутствие нужной страницы в оперативной памяти и т.п.), а также специальные команды типа INT n (n — номер прерывания), являющиеся генераторами программных прерываний. Эти команды обычно используются для вызова определенных функций ОС.

Обработка исключительных ситуаций (exception handling) — механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её

базового алгоритма. В русском языке также применяется более короткая форма термина: «обработка исключений».

Во время выполнения программы могут возникать ситуации, когда состояние данных, УВВ или компьютерной системы в целом делает дальнейшие вычисления в соответствии с базовым алгоритмом невозможным или бессмысленными. Классические примеры подобных ситуаций:

- Нулевое значение знаменателя при выполнении операции целочисленного деления. Результата у операции быть не может, поэтому ни дальнейшие вычисления, ни попытка использования результата деления не приведут к решению задачи.
- Ошибка при попытке считать данные с внешнего устройства. Если данные не удаётся ввести, любые дальнейшие запланированные операции с ними бессмысленны.
- Исчерпание доступной памяти. Если в какой-то момент система оказывается не в состоянии выделить достаточный для прикладной программы объём оперативной памяти, программа не сможет работать нормально.
- Появление сигнала аварийного отключения электропитания системы. Прикладную задачу, по всей видимости, решить не удастся, в лучшем случае (при наличии какого-то резерва питания) прикладная программа может озаботиться сохранением данных.
- Появление на входе коммуникационного канала данных, требующих немедленного считывания. Чем бы ни занималась в этот момент программа, она должна перейти к чтению данных, чтобы не потерять поступившую информацию.

1.2.4.2 Функции системы прерываний и их реализация

Функции системы прерываний:

1. Прием и хранение запросов прерываний от многих источников.
2. Выделение наиболее приоритетного запроса из множества поступивших.
3. Проверка возможности обработки запросов центральным процессором (проверка замаскированности запросов или сравнение уровня приоритетности запросов с так называемым порогом прерываний).
4. Сохранение состояния (контекста) прерываемой программы.
5. Вызов обработчика прерываний.
6. Собственно обработка прерываний (выполнение программы обработки прерываний).
7. Восстановление состояния (контекста) прерванной программы и возобновление ее выполнения.

Этапы 1-5 выполняются аппаратными средствами компьютера автоматически при появлении запроса прерывания. Этап 7 также выполняется аппаратно по команде возврата из обработчика прерывания.

Процедура опроса источников прерываний с целью выделения наиболее приоритетного (полинг/polling) может быть реализована как на аппаратном, так и на программном уровнях.

Программный полинг реализуется специальной программой, которая последовательно опрашивает триггеры запросов, объединенных, как правило, в единый регистр с целью поиска первого установленного бита.

Аппаратный полинг может быть реализован либо на основе многотактной схемы, в основу которой положен двоичный счетчик, либо с помощью однотактной схемы, которую обычно называют дейзи-цепочка.

Отношение процессора к поступившим запросам прерываний может быть выражено с помощью одного из двух механизмов:

- механизм масок;
- порог прерываний.

1.2.5 Часы реального времени

Во многих, сравнительно мощных микроконтроллерах есть встроенный блок часов реального времени (Real-Time Clock, RTC). Часы позволяют автоматически отслеживать переход через границу минут, часов, суток, отслеживают високосные года и автоматически переходят на летнее время [5]. Как правило, блок RTC создают на элементной базе, обеспечивающей пониженное энергопотребление. Для подсчета времени в RTC используются специальные кварцевые резонаторы с частотой 32,768 кГц. У микроконтроллера обычно существует возможность подключения дополнительного электропитания (например, литиевой батарейки или ионистора). Пример микроконтроллера со встроенным RTC – Philips LPC 2000, на базе ядра ARM7. На этом контроллере построен учебный стенд SDK-2.0.

Для того, чтобы часы реального времени могли выдавать точное астрономическое время, должны быть соблюдены следующие условия:

1. У часов должно быть собственное автономное питание, чтобы кратковременные или длительные перебои с питанием не приводили к сбросу астрономического времени.
2. Для обеспечения точности хода, часы должны калиброваться, так как у кварцевых резонаторов есть некоторый разброс параметров.
3. Точность хода часов зависит от параметров окружающей среды. Больше всего на точность хода влияет температура, так как при изменении температуры немного изменяется частота кварцевого резонатора. Для обеспечения точности хода необходимо предусмотреть калибровочные значения для различных температурных диапазонов.

Необходимо помнить, что часы реального времени не являются абсолютно надежным устройством. Проблемы могут возникнуть как на уровне интерфейса с часами (у большинства RTC нет контроля целостности передаваемых

данных), в кварцевом резонаторе или в самих часах. Если целевая функция системы сильно зависит от астрономического времени, необходимо предусмотреть несколько источников точного времени образуя мажоритар, а при считывании данных с часов реального времени проверять границы диапазонов считанных значений и проверять (например, с помощью таймера), идут часы или нет. Из-за аппаратных сбоев, часы реального времени могут не просто остановиться, но также из-за разного рода проблем может измениться серьезно тактовая частота, что приведет к серьезному замедлению или ускорению хода часов.

Необходимо заметить, что на точность хода часов реального времени влияет топология проводников на печатной плате. Проблема возникает из-за того, что часы реального времени делают, как правило, в виде устройства с пониженным энергопотреблением. Пониженное энергопотребление является следствием высокого импеданса между выводами микросхемы, а такие схемы чувствительны к помехам. Кроме того, на точность часов оказывает влияние паразитная ёмкость печатного монтажа. Для минимизации влияния паразитной ёмкости необходимо специальным образом организовывать разводку печатной платы.

Далее будет приведен конкретный пример организации и работы часов реального времени PCF8583 [14], которые установлены в контроллере SDK-1.1 (устройство данного контроллера будет рассматриваться в следущих главах).

1.2.5.1 Основные характеристики PCF8583

- I^2C-интерфейс.
- ОЗУ емкостью 240 байт для пользовательских данных.
- Календарь на 4 года.
- Функция будильника/сигнализации и определения переполнения.
- Поддержка 12- или 24-часовой формата времени.
- Внешний тактовый генератор – 32,768 кГц или 50 Гц.
- Автоматическое наращивание адреса при работе с памятью.
- Программируемые функции будильника, таймера, счетчика событий и прерывания.

1.2.5.2 Описание

Часы/календарь PCF8583 содержит оперативную память объемом 256 байт. Адреса и данные передаются последовательно через двунаправленную шину I^2C. Встроенный регистр адреса автоматически наращивается после чтения или записи каждого байта данных [14].

Адресный вывод A0 используется для настройки адресов устройств, что позволяет подсоединять к одной шине I2C две микросхемы PCF8583.

Встроенная микросхема генератора, работающая на частоте 32,768 кГц, и первые 8 байт оперативной памяти используются для часов, календаря и функций счетчика. Следующие 8 байт могут быть запрограммированы на использование в качестве регистров сигнализации (функция будильника), или же к ним можно обращаться как к свободным адресам памяти. Остальные 240 байт относятся к оперативной памяти.

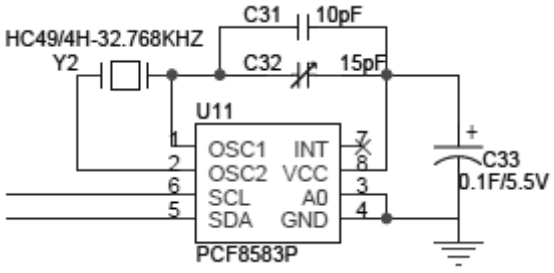


Таблица 1. Назначение выводов микросхемы PCF8583.

Обозначение	Вывод	Описание
OSC1 (OSCI)	1	Вход генератора на частоте 50 Гц или вход для импульса по событию.
OSC2 (OSCO)	2	Выход генератора.
A0	3	Адресный вход.
VCC	4	Отрицательный импульс.
SDA	5	Последовательная линия данных.
SCL	6	Последовательная линия синхронизации.
INT	7	Выход прерывания с открытым стоком (активный низкий уровень выходного сигнала).
Vdd	8	Положительный импульс.

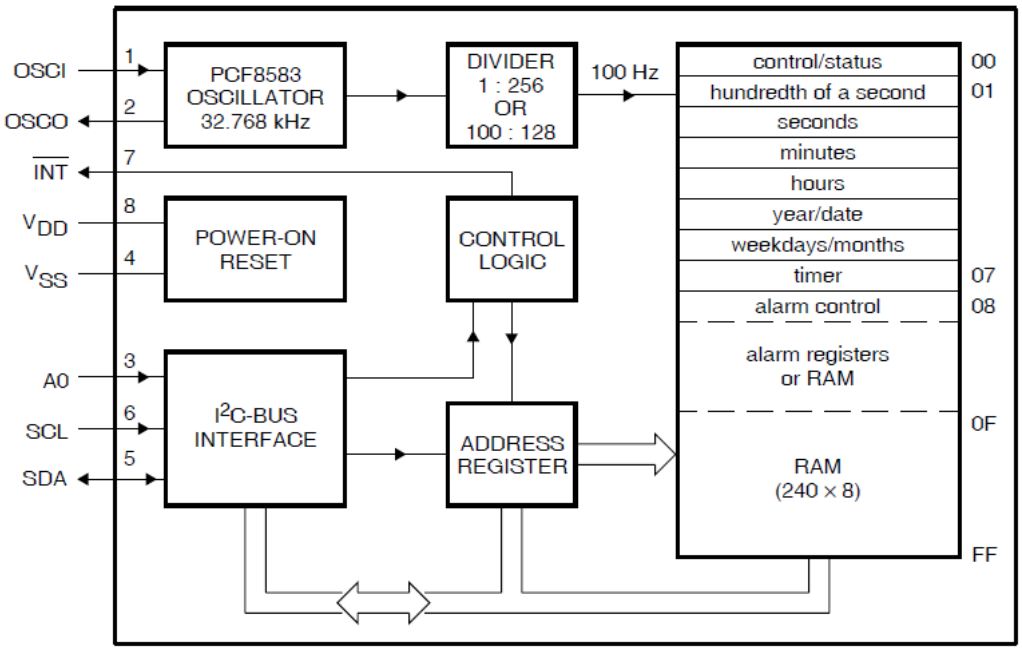


Рисунок 3. Структурная схема часов / календаря PCF8583

Обозначения:

- PCF8583 OSCILLATOR – тактовый генератор;
- POWER-ON RESET – сброс по включению питания;
- I2C-BUS INTERFACE – интерфейс шины I2C;
- DIVIDER – делитель;
- CONTROL LOGIC – логика управления;
- ADDRESS REGISTER – адресный регистр.

Таблица 2. Память RTC: регистровая модель

Назначение ячеек		**Адрес**
Регистр управления / состояния		00h
Режим работы «часы»	***Режим работы «счетчик событий»***	
Регистры-счетчики		
Сотые доли секунды (BCD)	Младший байт счетчика событий (BCD)	01h
Секунды (BCD)	Средний байт счетчика событий (BCD)	02h
Минуты (BCD)	Старший байт счетчика событий (BCD)	03h
Часы (BCD)	Свободный	04h
Год / дата (BCD)	Свободный	05h
Дни недели / месяц (BCD)	Свободный	06h
Таймер (счет от 0 до 99 BCD), значение 1 тика определяется настройками регистра 08h	Таймер (счет от 0 до 99 в формате BCD), значение 1 тика определяется настройками регистра 08h	07h
Будильник/сигнализация		
Регистр управления будильником/сигнализацией		08h
Регистры сигнализации		
Количество сотых долей секунды для сигнализации (BCD)	Младший байт количества событий для сигнализации (BCD)	09h
Количество секунд для сигнализации (BCD)	Средний байт количества событий для сигнализации (BCD)	0Ah
Количество минут для сигнализации (BCD)	Старший байт количества событий для сигнализации (BCD)	0Bh
Количество часов для сигнализации (BCD)	Свободный	0Ch
День месяца (года) для сигнализации (BCD)	Свободный	0Dh
Месяц, день недели для сигнализации (BCD)	Свободный	0Eh
Значение таймера для сигнализации (от 0 до 99 сотых долей секунды, секунд, минут, часов, дней в формате BCD), значение 1 тика определяется настройками регистра 08h	Значение таймера для сигнализации (таймер инкрементируется от 0 до 99 каждый импульс-событие, каждые 100 импульсов, каждые 10000, каждый 1000000, в формате BCD), значение 1 тика определяется настройками регистра 08h	0Fh
ОЗУ данных		10h-FFh

Микросхема PCF8583 содержит оперативную память объемом 256 байт с 8-битным адресным регистром, осуществляющим автоматическое

инкрементирование адреса, встроенную микросхему генератора (частота 32,768 кГц), делитель частоты, последовательную двунаправленную шину I^2C и схему, осуществляющую сброс по включению питания.

Первые 16 байт ОЗУ (адреса памяти от 00 до 0F) представляют собой адресуемые 8-битовые регистры специального назначения. Первый регистр (адрес 00) используется в качестве регистра управления/состояния. Регистры по адресам с 01 по 07 – счетчики для функций часов. Регистры, расположенные по адресам с 08 по 0F, могут быть запрограммированы в качестве регистров сигнализации (функция будильника) или использованы как обычные регистры памяти (когда сигналы отключены).

1.2.5.3 Режимы работы часов

При программировании регистра управления/состояния может быть установлен один из режимов работы часов [14]:

- на частоте 32,768 кГц (часы);
- на частоте 50 Гц (часы);
- режим счетчика событий.

В том случае, если выбран режим часов, сотые доли секунды, секунды, минуты, часы, дата, месяц (календарь на 4 года) и дни недели хранятся в двоично-десятичном формате в ОЗУ часов.

Режим счетчика событий используется для подсчета импульсов, подаваемых на вход генератора OSCI (к выводу OSCO ничего не подключается). Кроме того, в этом режиме часы можно настроить таким образом, чтобы они считали входящие события и генерировали логический «0» на выходе INT при переполнении. Этот сигнал можно завести на вход внешнего прерывания микроконтроллера и работать с часами как со счетчиком событий по прерыванию.

При чтении одного из регистров-счетчиков (адреса с 01h по 07h) содержимое всех счетчиков стробируется в регистры-защелки в начале цикла чтения. Таким образом, предотвращаются ошибки чтения счетчика часов. При записи в один регистр-счетчик с другими регистрами-счетчиками ничего не происходит.

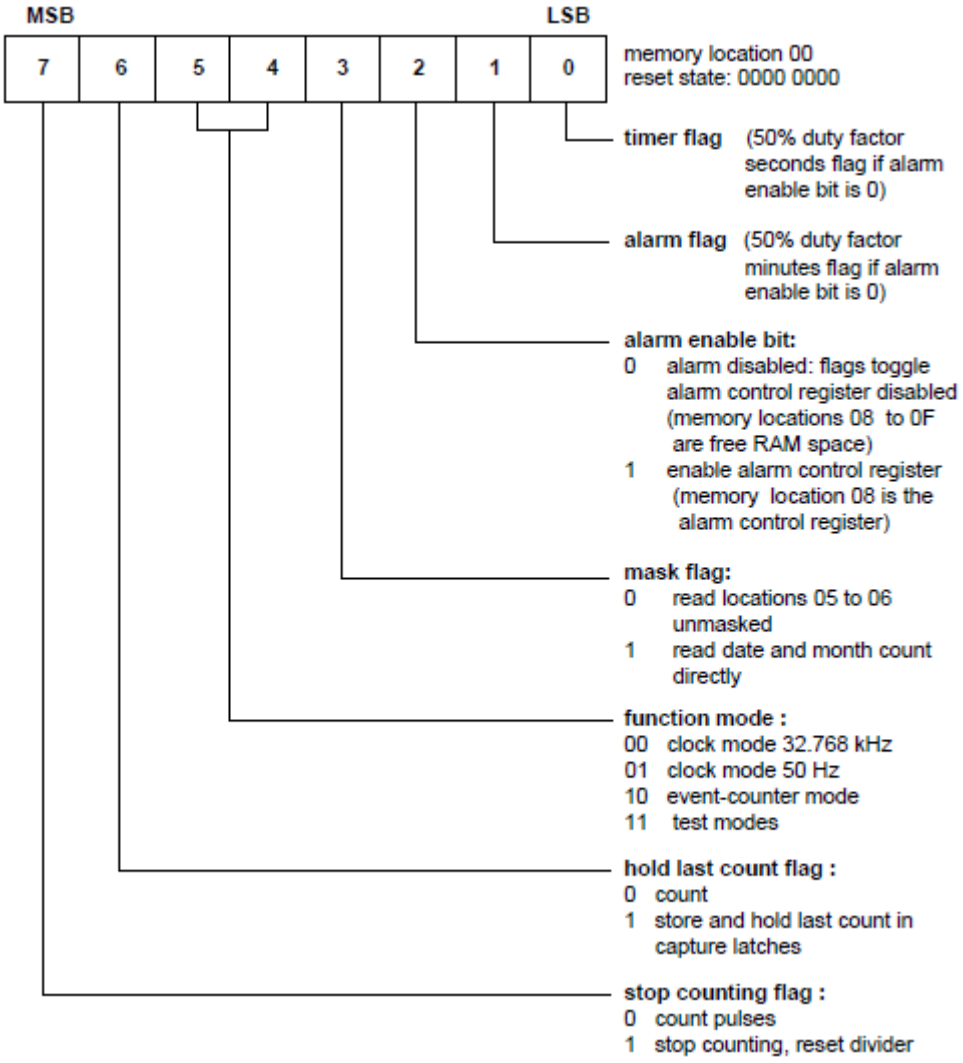


Рисунок 4. Регистр управления/состояния

1.2.5.4 Регистры-счетчики

В режиме работы «часы» можно устанавливать 12- и 24-часовые форматы времени путем изменения соответствующих битов регистра-счетчика часов.

Год и день месяца упакованы в памяти RTC по адресу 05h. Дни недели и месяцы хранятся по адресу 06h. Настройка регистра управления / состояния позволяет считывать дату напрямую путем маскирования в регистрах-счетчиках битов лет и дней недели. Это дает возможность пользователю напрямую считывать дату и месяц.

Таблица 3. Длины циклов счетчиков времени, режим часов.

Единицы	**Счетный цикл**	**Переход к началу цикла**	**Содержимое счетчика месяцев**
Сотые доли секунды	от 00 до 99	99 → 00	-
Секунды	от 00 до 59	59 → 00	-
Минуты	от 00 до 59	59 → 00	-
Часы (24 часа)	от 00 до 23	23 → 00	-

Часы (12 часов)	12 AM	-	-
	от 01 AM до 11 AM	-	-
	12 PM	-	-
	от 01 PM до 11 PM	11 PM → 12 AM	-
Дата	от 01 до 31	31 → 01	1, 3, 5, 7, 8, 10 и 12
	от 01 до 30	30 → 01	4, 6, 9 и 11
	от 01 до 29	29 → 01	2, год = 0
	от 01 до 28	28 → 01	2, год = 1, 2, 3
Месяцы	от 1 до 12	12 → 01	-
Год	до 0 до 3	-	-
Дни недели	от 0 до 6	6 → 0	-
Таймер	от 00 до 99	нет перехода	-

1.2.5.5 Будильник

При установке в регистре управления/состояния бита, разрешающего работу сигнализации, активируется регистр управления будильником (адрес 08h).

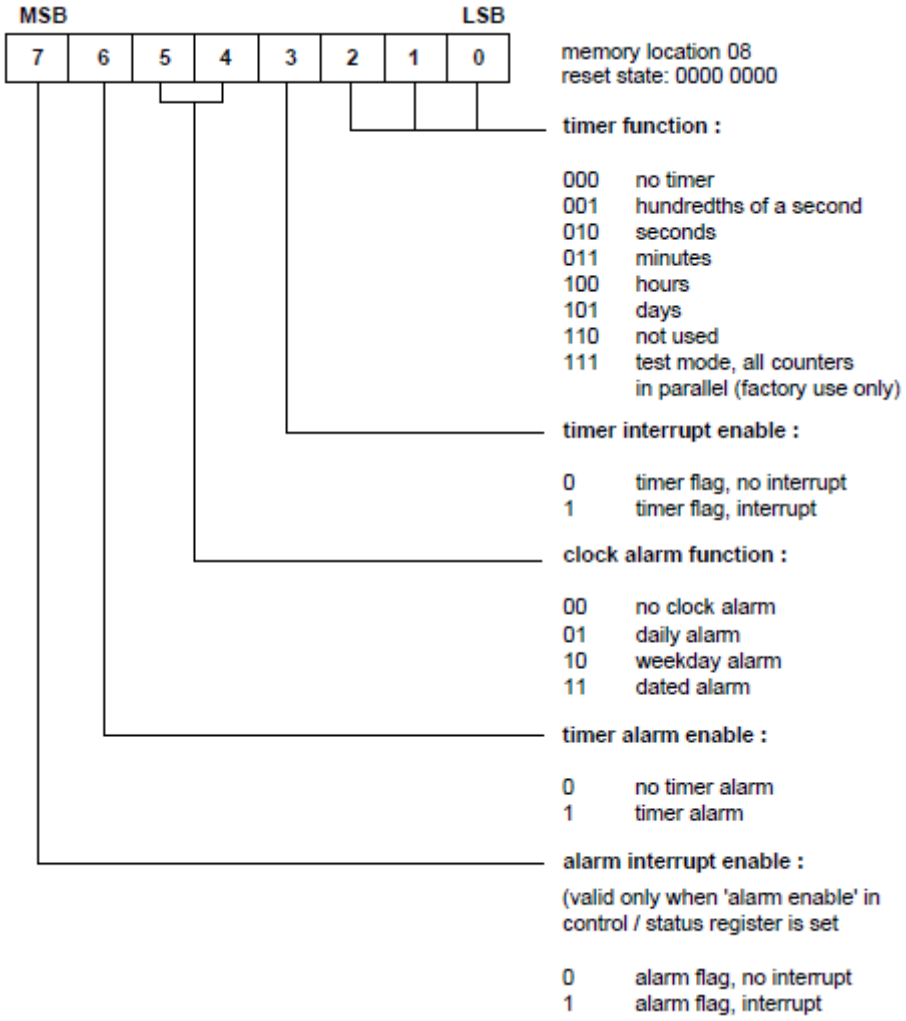


Рисунок 5. Регистр управления будильником (режим работы «часы»)

При помощи регистра управления будильником можно настроить его работу таким образом, чтобы он «звонил» в режиме работы «часы» при наступлении определенной даты, ежедневно в одно и то же время, по дням недели и по интервалам времени: каждое n сотых долей секунды, каждые n секунд, каждые n минут, часов, дней, n – значение, записываемое в соотетствующий регистр таймера (0Fh).

Кроме того, можно настроить будильник таким образом, чтобы на выходе INT в заданное время устанавливался сигнал «0». Этот сигнал можно завести на вход внешнего прерывания микроконтроллера и работать с часами-будильником по прерыванию (в учебном стенде SDK-1.1 этот выход никуда не подключен).

Если сигнал отключен (то есть бит 2 регистра управления/состояния равен 0), регистры сигналов (адреса с 08h по 0Fh) могут быть использованы как свободные ячейки памяти.

1.2.5.6 Регистры сигнализации

Все регистры сигнализации размещены со смещением 08h относительно соответствующих регистров-счетчиков.

Генерация сигнала происходит тогда, когда каждый бит регистра сигнализации совпадает с аналогичным битом соответствующего регистра-счетчика. Если речь идет о срабатывании будильника по дате, игнорируются биты года и дня недели. При генерации ежедневного сигнала будильника игнорируются биты месяца и даты. Если будильник настроен по дню недели, то из регистра сигнализации дней недели/месяца (0Eh) будут выбраны соответствующие дни недели.

1.2.5.7 Таймер

Таймер (адрес 07h) ведет отсчет от 0 (или от запрограммированного пользователем значения) до 99 в BCD формате. При переполнении таймер устанавливается в 0. Флаг таймера (младший бит регистра управления/состояния) устанавливается при переполнении таймера. Этот флаг сбрасывается программным путем. Инвертированное значение этого флага может быть передано внешнему прерыванию (выход INT) путем установки бита 3 регистра управления будильником.

Кроме того, сигнализация по таймеру может быть запрограммирована установкой бита разрешения сигнала по таймеру (бит 6 регистра управления будильника). Флаг сигнала (бит 1 регистра управления/состояния) устанавливается, когда значение таймера равно числу, указанному в регистре сигнализации по таймеру (адрес 0Fh). Если установлен бит разрешения прерывания по сигналу (бит 7 регистра управления будильником), то

инвертированное значение флага сигнала может быть передано на внешнее прерывание (выход INT).

Разрешение таймера программируется с помощью 3 младших битов регистра управления будильником: сотые доли секунды, секунды, минуты, часы, дни.

1.2.5.8 Режим счетчика событий

Режим счетчика событий устанавливается с помощью битов 4 и 5 регистра управления/состояния. Режим счетчика используется для подсчета импульсов, подаваемых на вход генератора OSCI (вывод OSCO остается неподключенным).

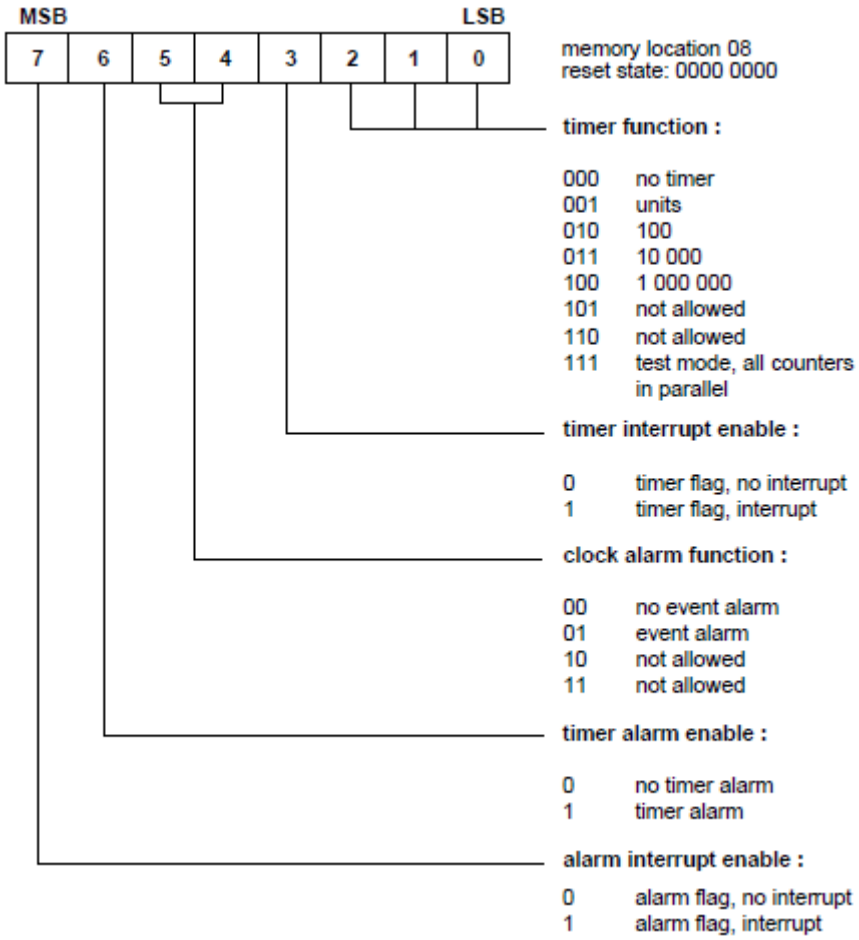


Рисунок 6. Регистр управления будильником (режим работы «счетчик событий»)

Счетчик событий хранит до 6-значное десятичное число, которые располагается в формате BCD в памяти по адресам 01h-03h (соответственно, младший, средний, старший байт). Таким образом, может быть посчитано до 1 миллиона событий. Сигнализация счетчика событий срабатывает в том случае, если содержимое регистра-счетчика совпадает со значением, хранящимся по адресам 09h-0Bh, и при этом сигнализация по событию разлешена (биты 4 и 5 регистра управления будильником установлены соответственно в 0 и 1). При этом по сигнализации устанавливается флаг сигнала (бит 1 регистра

управления/состояния). Инвертированное значение этого флага может быть передано на вывод прерывания микросхемы (INT) – для этого надо установить бит разрешения прерывания по сигналу в регистре управления будильником.

В этом режиме таймер (адрес 07h) инкрементируется при наступлении каждого первого, сотого, десятитысячного и миллионного события в зависимости от значений, установленных в битах 0, 1 и 2 регистра управления будильником (08h). Соответственно, сигнализация по таймеру в режиме счетчика событий срабатывает, когда содержимое регистра-счетчика таймера (07h) совпадает со значением, хранящимся по адресу 0Fh, и при этом сигнализация по таймеру разлешена (бит регистра управления будильником установлены соответственно).

1.2.5.9 Вывод прерывания INT

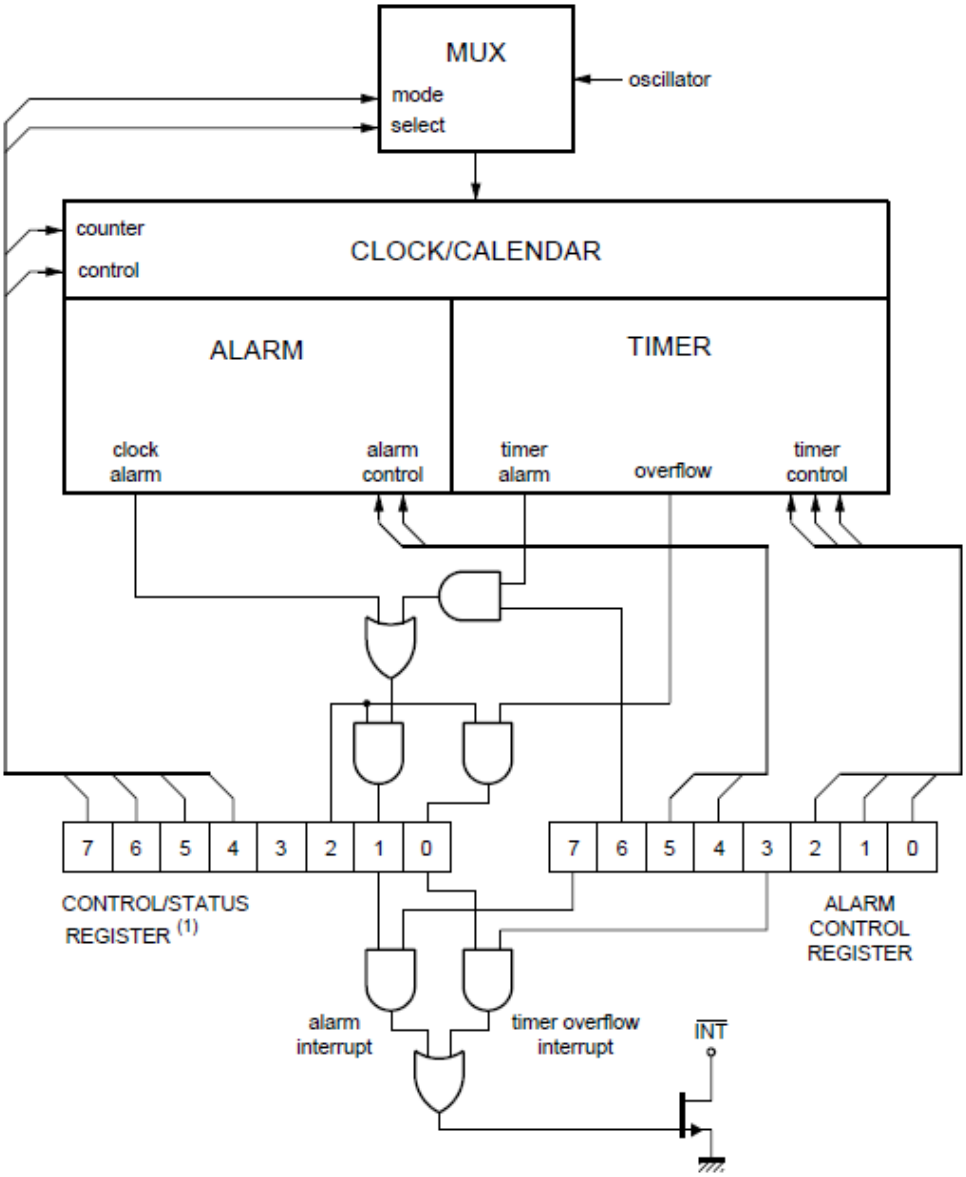


Рисунок 7. Логика работы прерываний по сигналу и по таймеру.

Условия для генерации на выходе INT активного сигнала (логический «0») определяются путем программирования регистра управления будильником (08h). К таким условиям относятся сигнализация по часам, сигнализация по таймеру, переполнение таймера, сигнализация счетчика событий. Прерывание происходит в том случае, когда установлен флаг сигнала или флаг таймера и при этом разрешено соответствующее прерывание. В любом случае прерывание очищается только программным путем: очисткой флага, вызвавшего прерывание.

1.2.6 Система контроля питания

Система питания является фундаментом для любой электронной схемы. К сожалению, во встроенных системах не всегда удается добиться качественного вторичного электропитания при использовании автономных источников энергии (аккумуляторов, химических элементов питания и т.п.), бортовой сети (например, в автомобиле), при наличии большого количества помех (например, на производстве). Система контроля питания предназначена для обеспечения надежного функционирования микроконтроллера в условиях нестабильного питающего напряжения [40].

Сразу после включения питания устройства на плате начинаются переходные процессы. Нарастание напряжения происходит не мгновенно и не линейно, время установки стабильного напряжения питания зависит от схемы и составляет обычно десятки-сотни миллисекунд. На этот момент времени система контроля питания задерживает старт микроконтроллера. Если старт не задержать, микроконтроллер, получая нестабильное питание, может давать сбои в работе и часто рестартовать.

В процессе работы система контроля питания постоянно проверяет уровень напряжения в цепях питания. Если уровень отклоняется от заданной величины, система контроля питания вырабатывает прерывание. Обработчик прерывания может корректно завершить работу встраиваемой системы, например, при внезапном пропадании питания.

Что можно сделать в обработчике прерывания, когда произошел сбой питания? Можно попытаться сохранить в энергонезависимой или обычной памяти контроллера текущее состояние прикладной программы, чтобы после возобновления подачи электроэнергии продолжить работу с прерванного места. Естественно, для реализации такого механизма защиты от сбоев питания вы должны реализовать свою программу так, чтобы в ней были четко выражены состояния ее работы. Другими словами, при проектировании такого рода программ очень полезно использовать конечные автоматы.

Задача супервизора питания – отследить уровень питающего напряжения и выдать цифровой сигнал, если уровень не соответствует заданному критерию. К чему могут привести изменения питающего напряжения?

Так как схема СнК не является чисто активной нагрузкой, при включении питания начинаются так называемые переходные процессы. Уровень питания устанавливается не сразу, а в течении нескольких десятков - сотен миллисекунд, при этом в ряде случаев могут возникать хаотичные колебания напряжения. В результате, микропроцессор и программируемая логика может десятки раз включаться и выключаться отрабатывая начальный участок алгоритма. Такое поведение схемы может отрицательно сказываться на энергонезависимой памяти устройства, так и на объекте управления, подключенном к УСО системы ввода-вывода. Аналогичные переходные процессы обычно возникают при выключении устройства или при внезапном и кратковременном пропадании питающего напряжения (например из-за плохого контакта).

Самым простым способом борьбы с этим явлением, является использование супервизора питания, который держит сигнал RESET в активном состоянии на время работы переходных процессов.

Обычно, для защиты от переходных процессов супервизоры питания снабжают компаратором с триггером Шмидта, а также счетным таймером, для организации регулируемой или постоянной задержки выходного сигнала.

Очень серьёзную проблему вызывает подача нескольких напряжений питания на схему. Какие проблемы могут возникнуть? Во-первых, при некорректной подаче питающего напряжения снижается надежность электронных компонентов, что в свою очередь снижает срок службы СнК. Это происходит из-за того, что некоторое время после включения, электронным компонентам приходится работать в недопустимых условиях (например, только с одним источником питания вместо трёх). Во-вторых, возможны ситуации, когда некорректная подача нескольких напряжений на плату вызывает превышение допустимых токов или напряжений на выводах микросхем, что может привести к фатальным последствиям.

Из-за некорректной подачи нескольких напряжений может возникнуть тирристорный эффект, приводящий к временному повышенному энергопотреблению или даже выходу устройство из строя. Возможен так же конфликт системных шин, возникающий из-за того, что при старте часть системных шин работает встык друг другу (обе шины работают на выход одновременно).

Решение проблем, связанных с несколькими питающими напряжениями также возможно с помощью супервизора питания.

1.2.7 Встроенная FLASH-память

Встроенная FLASH-память в основном предназначена для хранения программ и сравнительно больших объемов данных. Объем памяти, в зависимости от типа микроконтроллера, может достигать от единиц до сотен килобайт.

FLASH-память не позволяет перезаписывать отдельные байты. Адресное пространство памяти разбито на участки по аналогии с жесткими дисками, называемыми секторами. Для записи новой информации необходимо с помощью специальной команды стереть целый сектор или все сектора сразу. Один из секторов называется загрузочным. Обычно он располагается в той части адресного пространства микроконтроллера, с которого стартует процессор. Адрес старта - очень важный параметр, его обязательно нужно знать для всех используемых микроконтроллеров.

Необходимо помнить, что количество циклов записи и стирания ограничено и достигает десятков тысяч - миллионов раз, и если вы собираетесь использовать FLASH память для хранения данных необходимо предусмотреть.

Для обеспечения операции стирания и записи используются команды, отправляемые по специальным адресам адресного пространства FLASH-памяти.

Среди главных достоинств этой памяти можно назвать следующие:

- Энергонезависимость, т.е. способность хранить информацию при выключенном питании (энергия расходуется только в момент записи данных);
- Информация может храниться очень длительное время (десятки лет);
- Сравнительно небольшие размеры;
- Высокая надежность хранения данных, в том числе устойчивость к механическим нагрузкам;
- Не содержит движущихся деталей (как в жестких дисках).
- Основные недостатки флэш-памяти:
- Невысокая скорость передачи данных (в сравнении с динамической оперативной памятью);
- Незначительный объем (по сравнению с жесткими дисками);
- Ограничение по количеству циклов перезаписи (хотя эта цифра в современных разработках очень высока - более миллиона циклов).

1.2.8 Контроллер прямого доступа к памяти

Обмен данными микропроцессора с медленнодействующими периферийными устройствами (ПУ) обычно организуется по прерываниям или по программному опросу. Однако при передаче между основной и внешней памятью микро-ЭВМ больших блоков данных (десятки байт и более) производительность процессора в этих режимах является недостаточной.

Скорость передачи данных в режиме программного ввода-вывода ограничивается только процессором. Поэтому для передачи данных между устройствами внешней памяти и ОЗУ разработан специальный метод передачи данных без участия процессора, получившего название прямого доступа к памяти (Direct Memory Access, DMA). Аппаратные средства реализации канала ПДП называются контроллером прямого доступа к памяти (КПДП).

DMA-контроллер содержит несколько регистров, доступных центральному процессору для чтения и записи. Обычно эти регистры задают порт (или канал), который должен быть использован; направление переноса данных (чтение/запись); единицу переноса (побайтно/пословно); число байтов, которое следует перенести; адрес.

Необходимо отметить, что контроллер ПДП используется не только для передачи данных между ПУ и памятью, но и из памяти в память, и из ПУ в ПУ.

В идеальном случае режим ПДП совершенно не должен влиять на действия процессора, но для этого потребуется сложный и дорогой тракт в основную память вычислительной системы. Поэтому в большинстве систем используется временное разделение (мультиплексирование) общей системной шины между процессором и КПДП.

Разработано две разновидности ПДП: режим без пропусков тактов микропроцессора и режим с пропуском тактов микропроцессора.

В первом режиме реализации прямой доступ осуществляется без участия процессора (параллельно микропроцессору). Для этого используются те интервалы машинных циклов, в течение которых микропроцессор не обращается к основной памяти. Процессор (или дополнительная схема) идентифицирует эти интервалы для КПДП специальным сигналом, означающим доступность системной шины. Производительность процессора в этом режиме не уменьшается, но для каждого типа процессора потребуется свой контроллер ПДП. С другой стороны, сами передачи будут носить нерегулярный характер ввиду отсутствия у некоторых команд этих интервалов, что приведет к уменьшению скорости передачи данных в режиме ПДП.

Во втором способе реализации КПДП полностью ″захватывает″ системную шину на время передачи, при этом процессор отключается от системной шины и переходит в режим холостого хода. Таким образом, передачи ПДП осуществляются путем пропуска тактов процессора в выполняемой программе. При выполнении передач ПДП содержимое внутренних регистров процессора не модифицируются, поэтому его не нужно запоминать в памяти, а затем восстанавливать, как при обработке прерываний. Выполнение программы осуществляется сразу после окончания ПДП. Тем не менее, в условиях интенсивных передач ПДП эффективная производительность процессора уменьшается.

Обычно блоками ПДП снабжаются сравнительно мощные модели микроконтроллеров. В основном ПДП используется для взаимодействия памяти с устройствами ввода-вывода, которые могут создать большой поток данных: сетевыми контроллерами, UART, ЦАП и АЦП.

Как правило, блок ПДП программисты встроенных систем, особенно начинающие, стараются обходить стороной, считая его излишне сложным. На самом деле, использование ПДП не является более сложным, чем, например, использование системы прерываний, а эффект от применения ПДП может

разгрузить центральный процессор и увеличить производительность контроллера.

1.2.9 Средства понижения энергопотребления

Практически все современные микроконтроллеры имеют встроенные средства понижения энергопотребления, позволяющие отключать не используемые в данный момент блоки, понижать тактовую частоту процессора и переходить в различные режимы сна.

2 Технические средства встраиваемых систем

В данной главе рассматривается устройство таких компонентов встраиваемых вычислительных систем как вычислительный блок, устройства ввода-вывода, сетевой блок. Приводятся примеры конкретных реализаций.

2.1 Элементная база микропроцессорной техники для встраиваемых применений

2.1.1 Процессор

Процессор – элемент вычислительной системы, устройство для выборки команд из памяти и выполнения действий, предписанных командами; устройство, осуществляющее процесс обработки информации. В ряде случаев процессором также называют программные средства, предназначенные для обработки информации (например, текстовый процессор, языковой процессор). Процессоры (в смысле устройств) можно классифицировать по разным критериям, например, по способу организации функционирования (конвейерные, матричные), характеру обрабатываемой информации, по назначению и т.д.

Процессор характеризуется размером и количеством адресных пространств, шириной внутренней и внешней шины данных, системой команд, способом обработки прерываний, наличием адресного селектора для подключения внешних устройств, наличием ПДП, системы управления питанием. В высокопроизводительных ядрах могут присутствовать такие механизмы как конвейеры, кэш-память, устройство предсказания ветвлений, несколько АЛУ [6].

Программируемость процессора – не обязательное свойство.

Процессоры могут быть:

1. Непрограммируемые, не программно реализованные;
2. Программируемые и программно реализованные;
3. Непрограммируемые, но программно реализованные;
4. Программируемые, но не программно реализованные.

Программируемый процессор – процессор, у которого есть система команд. Его можно настроить на решение той или иной задачи.

Функции непрограммируемого процессора раз и навсегда зафиксированы.

Процессоры могут строиться как аппаратные блоки или по принципу программно-управляемых устройств.

2.1.2 Классификация процессоров

Процессоры целесообразно классифицировать по функциональной направленности, функциональной гибкости, способу реализации [6]. В первом

случае говорят об универсальных (общего назначения) и специализированных (ПВВ, графические, математические, обработки сигналов) процессорах. Первые призваны решать различные задачи и имеют широкую область применения, тогда как вторые ориентированы на решение узкого круга задач. Универсальные процессоры характеризуются: способностью обрабатывать большое число команд; системой команд (СК): если система команд позволяет решить любую задачу, то процессор универсальный. Нужно анализировать систему команд на ее сбалансированность, формы данных, способы адресации. Хотя универсальность процессора – вопрос относительный.

Также можно выделить центральные, периферийные и сервисные процессоры. Центральный процессор осуществляет общее управление вычислительной системой: производит основную обработку данных, обмен ими с другими элементами ВС, а также управляет работой элементов ВС. Периферийный процессор выполняет лишь часть функций вычислительной системы: управляет и обменивается данными с устройствами ввода-вывода (процессор ввода-вывода), может участвовать в вычислительном процессе (обрабатывать часть данных). Сервисный (обслуживающий) процессор обычно не участвует в основном вычислительном процессе и выполняет функции контроля и обслуживания: выполняет инструментальные функции (доставка и отладка программного обеспечения, настройка оборудования), осуществляет контроль правильности функционирования системы, измерение параметров окружающей среды (температура, влажность), напряжения питания и т.п. В ВС один и тот же процессор может выполнять функции как периферийного, так и сервисного процессора.

Степень функциональной гибкости, или возможность настройки процессора на выполнение конкретной функции из допустимого множества в каждый момент времени, определяется возможностью и оперативностью его программирования. Если в процессе эксплуатации функция может быть перенастроена, то такой процессор называется программируемым (programmable processor), в противном случае мы имеем дело с непрограммируемым, "жестким" устройством (dedicated processor – близкий, но не точный термин, другой возможный вариант – hardware accelerator). Сложность механизма программного управления может изменяться в очень широких пределах.

Рассмотрим пример. Промышленность выпускает большое число тактовых генераторов, часть из которых может формировать различные наборы частот в зависимости от состояния управляющих входов или записанных в специальные запоминающие ячейки кодов коэффициентов деления. Если пользователю в составе системной платы ПК генератор попадает с жестко запаянными значениями на входах (или с запрограммированными в ячейках кодами без интерфейса для их изменения), то такое устройство следует считать специализированным ("жестким") процессором. Если же на вашей плате установлены переключатели для задания частоты, то мы имеем дело с

программируемым устройством. Конечно, обычно тактовый генератор, пускай и программируемый, процессором не называют. Но разве кто-то может однозначно провести границу функциональной сложности в этом вопросе?

По способу реализации процессоры, прежде всего, следует делить на аппаратно-реализованные (или аппаратные, hardware) и программно-реализованные (соответственно, программные, software). Каждая из указанных групп реализаций, в свою очередь, имеет множество вариантов. Вполне пригодным для практического использования критерием программной реализации процессора следует считать наличие в его составе хотя бы одной программно-реализованной части.

Как на практике классифицировать способ реализации устройства по аппаратно-программному признаку? Довольно часто программную реализацию связывают с использованием принципа программного управления при проектировании устройства. Также широко распространено мнение об обязательном свойстве последовательной интерпретации управляющей информации устройством для отнесения его к разряду программно-реализованных.

Общим критерием аппаратной или программной реализации устройства (функции) предлагается считать степень избыточности присутствующей в устройстве регулярной структуры (блока постоянной или оперативной памяти, логической матрицы, операционных элементов и т.д.). Из данного определения следует, что не только организация управляющей части устройства определяет способ его реализации. Например, табличный функциональный преобразователь также может быть отнесен к категории программных устройств. Явным случаем аппаратной реализации является отсутствие в устройстве регулярных структур вообще (нерегулярная "жесткая" логика, или по-другому, "клубок схем"). На практике сегодня, в силу ряда причин, устройство создают на основе избыточных регулярных структур, а затем фиксируют его в получившемся виде, либо выполняют еще один шаг – удаляют неиспользованные элементы регулярной структуры (целевая компиляция, кремниевая компиляция и так далее в соответствии с контекстом). В первом случае мы будем иметь дело с программно-реализованным устройством, во втором случае программно-реализованная версия на этапе проектирования заменяется аппаратной реализацией ("жесткая логика").

Таким образом, можно говорить о четырех базовых вариантах процессоров:

- "жесткий" (непрограммируемый) процессор с аппаратной реализацией (dedicated hardware processor);
- программируемый процессор с аппаратной реализацией (programmable hardware processor);
- "жесткий" процессор с программной реализацией (dedicated software processor);

- программируемый процессор с программной реализацией (programmable software processor).

2.1.3 Микропроцессор и микроконтроллер

Традиционно он трактуется как программируемый процессор в интегральном исполнении. Однако в связи с развитием технологии ASIC, ASSP, PLD (ПЛИС) логичнее микропроцессором называть все четыре группы процессорных элементов в случае их интегрального исполнения.

Вычислительные системы на верхнем уровне рассмотрения могут быть представлены тремя группами элементов: обработчики, устройства памяти, интерфейсы. С группой обработчиков связаны термины "процессор" и "контроллер". Предлагается процессором называть обрабатывающий элемент, функции которого в рамках прикладной задачи еще не зафиксированы. В зависимости от закрепленной прикладной (по отношению к данному элементу, а не к системе в целом) функции процессор будет играть роль контроллера (т.е. устройства управления), умножителя, супервизора, диспетчера и т.д. В свою очередь контроллеры могут быть самого различного назначения: памяти, принтера, последовательного интерфейса, технологического процесса и другие. Еще один важный термин – "микроконтроллер", следует понимать как контроллер, построенный на основе микропроцессорной элементной базы. Микроконтроллеры могут быть однокристальными, одноплатными, программируемыми, логическими, промышленными, универсальными и т.д. Микроконтроллер в одном кристалле содержит микропроцессор и набор периферийных устройств и контроллеров: контроллер прерываний, таймеры, контроллер сети, контроллер последовательного канала, контроллер памяти, контроллер ПДП и т.д.

2.1.4 Классификация микроконтроллеров

Существует множество способов, с помощью которых можно производить классификацию микроконтроллеров [5, 21].

- По разрядности различают 8, 16 и 32 разрядные микроконтроллеры.
- По возможностям в области обработки сигналов можно рассматривать обычные микроконтроллеры и DSP-микроконтроллеры.
- По области применения различают следующие микроконтроллеры: автомобильные, промышленные, для контроллерных сетей, управления двигателями, управления беспроводными сетями.

По объему вычислительных ресурсов условно можно выделить четыре характерные группы микроконтроллеров.

- Периферийные процессоры – Microchip PIC 10, PIC12, PIC16, PIC18, PIC24, Atmel AT90xxxx и т.п.
- Универсальные 8-ми и 16-ти разрядные ОМЭВМ – Intel MCS51, Siemens SAB 5xx, Atmel Mega10x и т.п.

- Универсальные 16-ти и 32 разрядные ОМЭВМ – Fujitsu FR-50, ARM7 и т.п.
- Универсальные однокристальные 32-х разрядные микроконтроллеры и процессоры Freescale MPC560xx, ARM9, ARM11 и т.п.

Для первой категории процессоров характерны следующие особенности:

- небольшой объем памяти данных (десятки – сотни байт);
- небольшой объем памяти программ (единицы – десятки килослов);
- сравнительно высокое быстродействие;
- система команд RISC;
- низкое энергопотребление;
- малое число выводов;
- невозможность подключения внешней памяти;

Старшие модели микроконтроллеров могут иметь в своем составе сетевые контроллеры. Основная идея в этих контроллерах – обеспечение создания устройств с низким энергопотреблением и минимальным количеством компонентов на плате.

Для второй категории процессоров характерна возможность использования внешней (внекристальной) памяти. Отличает эту категорию низкая цена и небольшие вычислительные ресурсы. Производительность таких микроконтроллеров как правило значительно ниже, чем у первой категории. Контроллеры этого типа применяются в основном в простых и дешевых устройств, не предъявляющих повышенных требований по производительности и энергопотреблению, но имеющих повышенные требования по объему программного кода и требуемой памяти данных.

Третья категория процессоров имеет развитые аппаратные средства для повышения производительности обработки информации, гораздо более мощный, по сравнению с первой и второй категориями центральный вычислитель и расширенное адресное пространство. Начиная с этой категории в состав микроконтроллеров производители начинают наиболее активно включать сетевые контроллеры. В настоящее время это наиболее распространенные микроконтроллеры.

В четвертой категории процессоров характерно применение механизмов защиты памяти и большое адресное пространство, что позволяет без особых проблем применять операционные системы реального времени. От третьей категории их также отличает более высокая производительность.

2.1.5 Программируемые логические интегральные схемы

Программируемая логическая интегральная схема (PLD, Programmable Logic Device) – электронный компонент, состоящий из логических ячеек и конфигурируемой схемы соединений. Основное назначение – построения реконфигурируемых цифровых схем. В отличие от обычных интегральных

схем, функциональность не определяется на этапе изготовления раз и навсегда и может создаваться и изменяться конечным пользователем (инженером), исходя из своих нужд.

ПЛИС, в первом приближении, представляет собой множество однотипных логических элементов, соединяемых с помощью специальных коммутационных матриц. Соединение и инициализация элементов осуществляется посредством бинарного образа, загружаемого в конфигурационную память ПЛИС. Файлы конфигурации (бинарные образы) генерируются с помощью САПР производителя конкретной ПЛИС и являются его интеллектуальной собственностью. На аппаратной базе FPGA можно реализовывать системы на кристалле. Описания проектов возможно делать как на языках структурно-функционального описания аппаратуры (Verilog,VHDL), так и при помощи более высокоуровневых языков, таких как SystemC.

Применение ПЛИС:

- Связывающая логика (glue logic). ПЛИС выступает как средство обеспечения совместимости нескольких интерфейсов. Изначально это достигалось с помощью логики 74 и 40 серий, а в более сложных случаях с помощью CPLD и FPGA. Примеры: декодер шины адреса, расширитель портов.
- Для шифрования интеллектуальной собственности электронных схем
- Для прототипирования ИС и в малосерийном производстве
- Цифровая обработка сигналов, изображений
- Криптография
- Высокопроизводительные вычисления

2.1.6 Программируемая логическая матрица

PAL (Programmable Array Logic) – программируемая логическая матрица (ПЛМ). Представляет собой простую программируемую логическую интегральную схему (ПЛИС).

ПЛМ делаются на базе технологии ПЗУ, то есть они программируются однократно. Внутри, ПЛМ представляют собой несколько логических элементов «И» с перемычками на входах, выходы которых подключены фиксированным образом к элементам «ИЛИ». Процесс программирования заключается в пережигании перемычек (на рисунке они показаны волнистой линией) с целью получения нужной булевой функции.

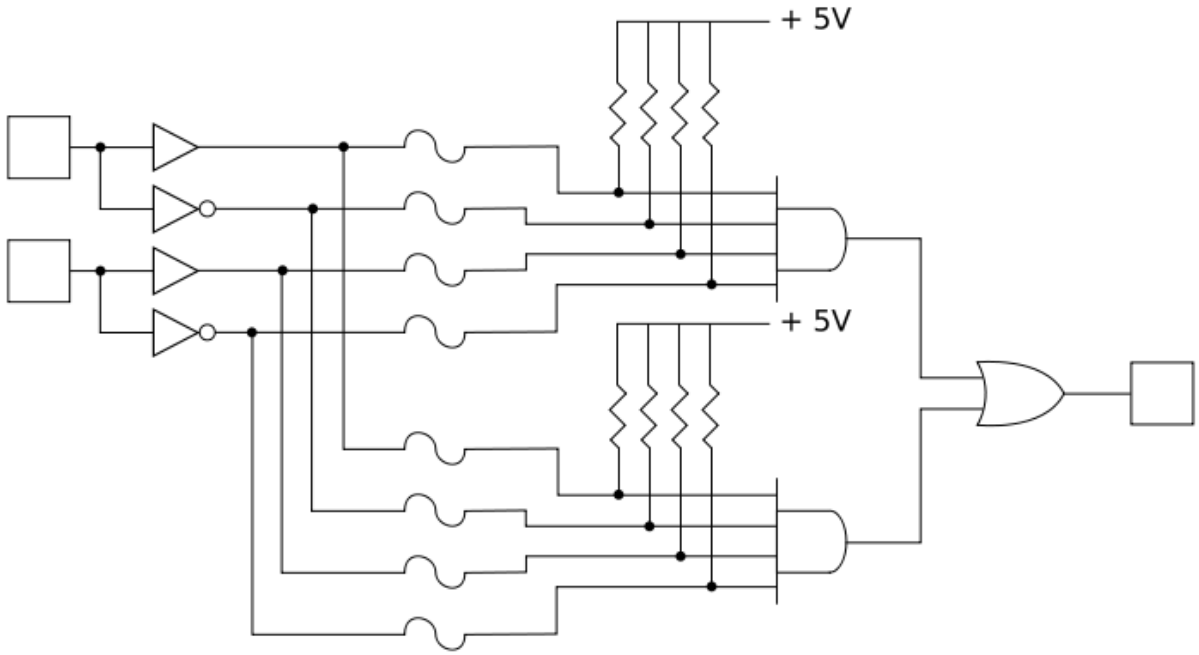


Рисунок 8. Фрагмент ПЛМ: входной и выходной буфер, матрица И (2 терма), матрица ИЛИ

2.1.7 CPLD

CPLD (complex programmable logic device) – электронное устройство, принадлежащее к классу программируемых электронных схем (ПЛИС) и находящееся по сложности между FPGA и PAL. CPLD состоят из блоков логических вентилей, объединенных программируемой коммутационной матрицей. В отличие от FPGA, схемы CPLD делают обычно на базе энергонезависимой памяти. В последнее время, различия между CPLD и FPGA постепенно стираются.

2.1.8 FPGA

FPGA (Field-Programmable Gate Array) представляет собой множество однотипных логических элементов, соединить которые можно с помощью программируемой коммутационной схемы. Основная цель создания FPGA состоит в том, чтобы позволить проектировщику с помощью относительно простых технологий, в лабораторных условиях получить на кристалле достаточно сложное и при этом ещё и работающее цифровое устройство. Итак, что мы выигрываем при использовании технологии FPGA? Во-первых, мы избавляемся от сложного производства интегральных схем. Во-вторых, упрощается проектирование кристалла из-за отсутствия проблем с топологией микросхемы и взаимным влиянием различных узлов. Что мы теряем? Мы теряем производительность, снижаем надежность, увеличиваем чувствительность к помехам, увеличиваем энергопотребление кристалла и получаем меньшую отдачу от использования площади кристалла из-за роста количества вентилей в 20..30 раз.

К основным направлениям применения FPGA можно отнести: прототипирование, симуляцию и изготовление мелко и среднесерийных

устройств. При достаточно больших партиях устройств становится экономически выгодным изготовление ASIC.

Схемы строятся на баз логических элементов. Каждый логический элемент FPGA состоит из двух основных частей: программируемого логического элемента (так называемый LUT — Lookup Table) и триггера на выходе.

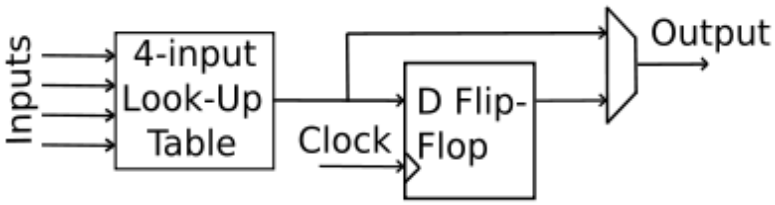


Рисунок 9. Логический элемент FPGA

Один логический элемент состоит из нескольких десятков логических вентилей сделанных в базисе 2 И-НЕ или 2 ИЛИ-НЕ. Программируемый логический элемент содержит в себе таблицу истинности, позволяющую реализовать любую комбинационную схему. Как правило, такого рода элементы выполняют в виде набора регистров памяти. Так называемая конфигурационная память FPGA, содержит в себе информацию необходимую для соединения логических элементов и таблицы истинности для LUT.

Логические элементы, составляющие FPGA, принадлежат уровню RTL. Их соединение хорошо описывается структурными VHDL и Verilog. Используемая модель вычислений — модель дискретных событий.

По мере совершенствования технологий производства интегральных схем появляется тенденция к укрупнению базовых элементов FPGA. В большинство современных моделей добавляют память, элементы арифметико-логических устройств, умножители и целые процессорные ядра.

2.1.9 Системы-на-кристалле

Система-на-кристалле (System-on-Chip, SoC) – в общем случае системы, на едином кристалле которых интегрированы процессор (процессоры, в том числе специализированные), некоторый объем памяти, ряд периферийных устройств и интерфейсов, — то есть максимум того, что необходимо для решения задач, поставленных перед системой. Выражение "система на кристалле" не является, строго говоря, термином. Это понятие отражает общую тенденцию к повышению уровня интеграции за счет интеграции функций.

Производительность приборов класса "система-на-кристалле" в значительной мере зависит от эффективности взаимодействия всех встроенных компонентов и от эффективности их взаимодействия с внешним, относительно прибора, миром. В первую очередь это связано с различием в быстродействии встроенных компонентов, в особенности организации интерфейсов.

Системы на кристалле обычно состоят из трех основных цифровых системных блоков: процессор, память и логика. Процессорное ядро реализует

поток управления, когда каждой управляющей программой однозначно устанавливаются последовательности выполнения операций обработки данных, что позволяет задавать один из возможных алгоритмов работы всей интегральной схемы. Память используется по ее прямому назначению — хранение кода программы процессорного ядра и данных. Наконец, логика используется для реализации специализированных аппаратных устройств обработки и прохождения данных, состав и назначение которых определяются конечным приложением — потока данных.

Реальная система на кристалле содержит как минимум все три перечисленных блока, что исключает применение многочисленных отдельных интегральных схем и реализацию интерфейсов связи между ними. Однокристальное конфигурируемое или программируемое решение допускает оперативное изменение своей внутренней аппаратной структуры и конечного предназначения как на этапе производства, так и в полевых условиях, непосредственно в проекте. Такие интегральные схемы были отнесены к группе изделий системного уровня интеграции, но получили другое название — Configurable System on a Chip или CSoC. Поскольку термин CSoC не стандартизован, то существуют и другие названия изделий этого класса — System on Programmable Chip (SoPC), Programmable System on a Chip (PSoC) или просто SoC, что определяется вкусом и желаниями конкретного производителя микросхем.

Типовая встраиваемая система, построенная на базе SoC, содержит различные наборы следующих интерфейсов и контроллеров:

- Системная шина и контроллеры шин LPC/ISA, PCI, PCMCIA;
- Контроллеры управления NOR/NAND Flash, SDRAM, SRAM, DDR;
- Контроллер Ethernet;
- Последовательные интерфейсы UART, SPI/SSP/uWire, RS-232, RS-422/RS-485, CAN;
- Беспроводные интерфейсы WiFi/IEEE802.11, ZigBee, Bluetooth, IrDA;
- Интерфейсы поддержки Flash-карт памяти: SD/MMC, CompactFlash, MemoryStick;
- Контроллер LCD STN/TFT/OLED;
- Контроллер матричной клавиатуры;
- Модули беспроводной передачи данных GSM/GPRS, CDMA;
- Модули приема сигналов спутниковых навигационных систем GPS, Glonass;
- Аппаратные поддержки плавающей точки, шифрования, DRM и т.п.;
- Аудио- и видеоинтерфейсы.

2.2 Модульный принцип организации процессора ВВС

2.2.1 Типовая структура процессора для встраиваемых систем

В настоящее время выпускается большое количество разнообразных по структуре и функциям процессоров для применения во встроенных системах. Эта номенклатура постоянно расширяется, чтобы обеспечить решение специфических задач в различных прикладных задачах. Возможность разработки и производства новых моделей в сжатые сроки обеспечивает *модульный принцип структурной организации*.

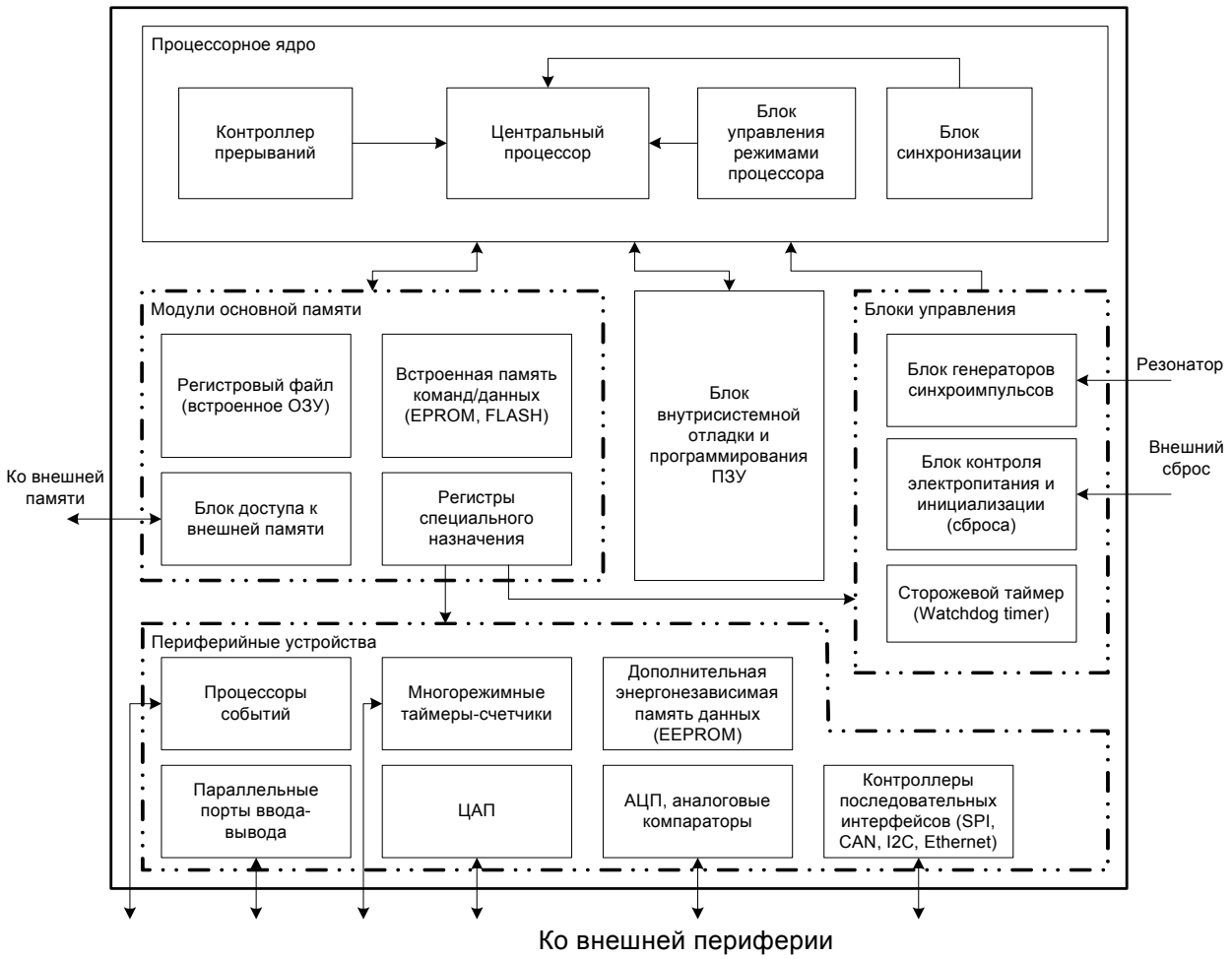


Рисунок 10. Типовая структура процессора для встроенных систем.

При модульном принципе построения все процессоры одного семейства содержат в себе одинаковый базовый функциональный блок – процессорное ядро, и изменяемый функциональный блок.

Базовый блок (процессорное ядро) включает [6]:

- Центральный процессор;
- Внутренние магистрали адреса, данных и управления;
- Блок формирования множества сигналов с различными фазами и частотами для синхронизации центрального процессора и внутренних магистралей.

- Блок управления режимами работы процессора, который может настраивать процессор на активный режим, режим обработки прерываний, несколько режимов пониженного энергопотребления, режим рестарта.

Процессорное ядро является основным отличительным признаком архитектуры определенного семейства процессоров, поэтому его (ядро) называют по названию семейства. Например, ядро MCS-51 или ядро PIC16.

Изменяемый функциональный блок включает:

- Модули памяти различных типов: оперативную память данных типа SRAM, постоянную память команд (программ) типов ROM, EPROM или FLASH, энергонезависимую память данных типа EEPROM;
- Модули периферийных устройств;
- Модули управления и синхронизации.

В различных микросхемах семейства может иметься различный набор модулей изменяемого функционального блока. Общую совокупность модулей, реализованных в микросхемах одного семейства, называют *библиотекой периферийных модулей* данного семейства. В данную библиотеку входят, как говорилось, не только периферийные, но и модули памяти, встроенные генераторы синхронизации, блок контроля электропитания и формирования сигналов рестарта системы в случае сбоев или «внешнего сброса», модули внутрисхемной отладки и программирования. В последнее время активно развивается направление «System-On-Chip», когда конечный пользователь сам может формировать структуру специализированного процессора из предоставленной библиотеки периферийных модулей, а также самостоятельно разрабатывать новые модули [6].

2.2.2 Процессорное ядро

Техническое решение процессорного ядра определяют следующие параметры [6]:

- Архитектурные – набор регистров, организация памяти, способы адресации операндов в памяти, система команд для обработки этих данных.
- Схемотехнические решения – схемы регистров, АЛУ, схемы управления магистралями и т.п. Схемотехника определяет также внутреннюю диаграмму функционирования – последовательность перемещения данных по магистралям между регистрами, памятью, АЛУ.
- Технология производства – определяет допустимую сложность схемы, максимальную частоту переключений, энергопотребление.

В современных процессорах для встраиваемых систем реализуют как CISC-архитектуру (Motorola HC11, Intel MCS-51, AMD Am186 и др.), так и RISC-архитектуру (MicrochipPIC, Atmel AVR, Triscend E7-ARM).

Производительность процессорного ядра определяется комплексом факторов:

- Частотой тактирования межмодульных магистралей адреса и данных Fbus. Она определяется из частоты генератора синхронизации Fxclk по соотношению, индивидуальному для каждого процессорного ядра. Например, для MCS51 – Fxclk/Fbus = 12 и при частоте генератора 12МГц ядро работает на частоте 1МГц; для Am186ES - Fxclk/Fbus = 1; для Motorola HC08 существует режим умножения входной частоты и Fxclk/Fbus < 1.
- Количеством пересылок регистр-регистр за единицу времени. Для RISC-процессоров это одна пересылка за такт шины, для CISC – 1..3 пересылки (они медленнее).
- Производительностью при выполнении операций наиболее используемым в конкретном алгоритме управления. Например, для ПИД – регуляторов – это операции умножения/деления; для простых конечных автоматов – это логические операции.
- Временем вызова/возврата подпрограммы обработки прерывания. Этот параметр значим для функционирования в режиме жесткого реального времени и определяет максимальную интенсивность обрабатываемых событий.

2.2.3 Организация прерываний в управляющих процессорах

Источниками прерываний могут быть:

1. Внешние источники. Запрос передается перепадом напряжения на входе (из «1» в «0» или из «0» в «1») или определенным уровнем напряжения («0» или «1») на внешнем входе запроса прерывания.
2. Внутренние источники – встроенные модули памяти (обычно от модуля EEPROM) или модули периферийных устройств:
   a) Таймеры/счетчики. Запрос вырабатывается по переполнению;
   b) Блоки захвата/сравнения. Запрос по событию входного захвата или равенства при выходном сравнении.
   c) АЦП. Запрос по завершению преобразования.
   d) Аналоговые компараторы. Запрос по изменению соотношения уровней входных сигналов.
   e) Приемопередатчики последовательных интерфейсов (RS-232 (SIO), SPI, I^2C, USB, CAN, Ethernet, HDLC и т.п.). Запрос вырабатывается:

- По приему байта или пакета и доступности новых принятых данных;
- По завершению передачи байта или пакета и освобождению передатчика.

3. Программные прерывания.

Организация прерываний в процессорах для управляющих систем ничем принципиально не отличается от универсальных процессоров. В различных семействах управляющих процессоров реализованы различные механизмы обработки прерываний:

1. Векторный с жестким приоритетом (ST7, AVR, Am186).
2. Векторный с программируемым приоритетом (MCS-51, M16C, i386EX).
3. Векторный с динамической таблицей векторов (M16C).
4. С общим вектором (механизм полинга) (PIC).

Укрупненная схема блока обработки запросов прерываний (механизм масок)

Механизм масок основан на использовании специального бита для каждого запроса прерываний, с помощью которого разрешается или запрещается обработка прерываний, связанных с этим запросом. В процессоре семейства Intel функцию маски выполняет бит IF, с помощью которого разрешается (IF=1) или запрещается (IF=0) обработка запросов внешних прерываний (как правило, от ВУ). При сброшенном флаге IF принято говорить, что прерывание замаскировано.

Запросы прерываний от ВУ формируются с помощью PIC (Programmable Interruption Controller), который связан линией запроса с CPU. Запросы от PIC поступают в CPU на внешний вход INTR.

Для управления прерываниями используется маска прерываний (см. рис. 4), представляющая собой двоичное слово M = m1 m2 … mk с числом разрядов, равным числу маскируемых причин прерывания. Если разряд маски mk = 0, то прерывание по причине k запрещено (замаскировано), если разряд маски mk = 1, то прерывание по причине k разрешено (не замаскировано). Маска прерываний хранится в процессоре, куда она загружается командой УСТАНОВИТЬ МАСКУ A, где A – адрес. По этой команде слово с адресом A загружается в качестве маски в процессор и определяет отношение процессора к сигналам прерывания. Если все разряды маски равны нулю, процессор не реагирует ни на одну причину прерывания.

В простейших процессорах используется следующий способ маскирования прерываний. В систему команд компьютера вводятся две системные команды ЗАПРЕТИТЬ ПРЕРЫВАНИЯ и РАЗРЕШИТЬ ПРЕРЫВАНИЯ, выполнение

которых приводит к запрещению и разрешению прерываний одновременно по всем причинам.

Команды, маскирующие прерывания, относятся к группе привилегированных команд.

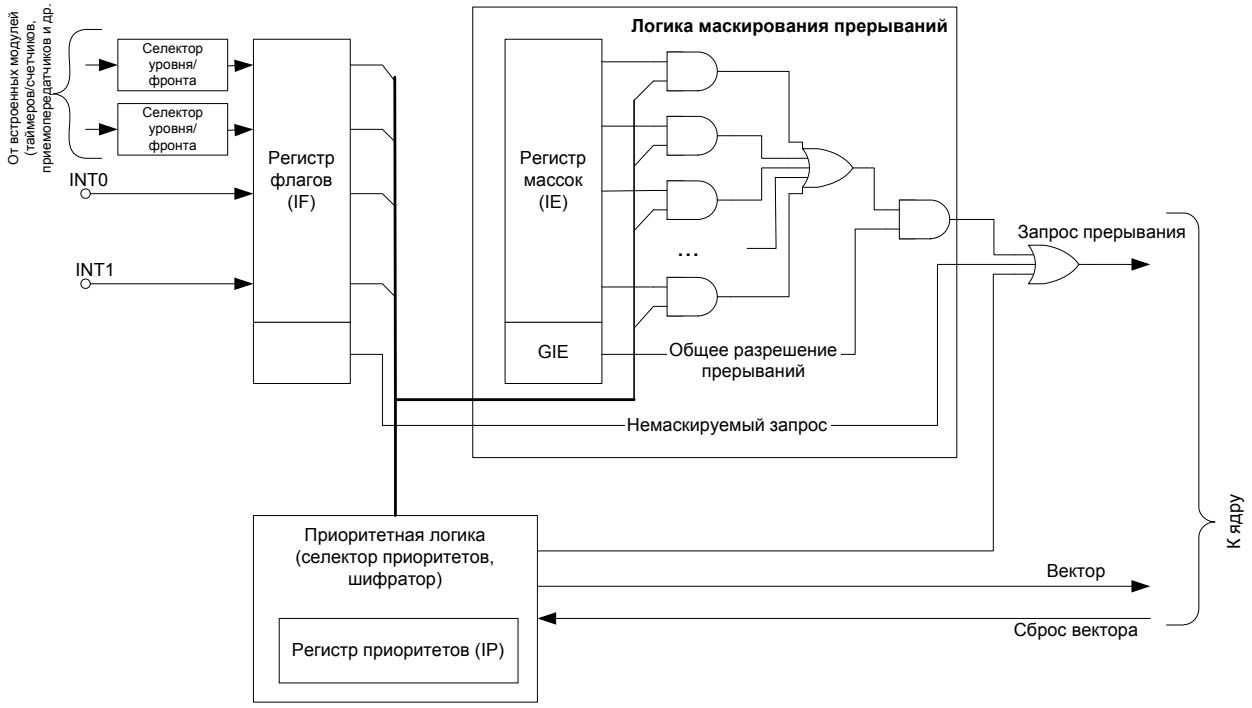


Рисунок 11. Блок обработки запросов прерываний

На рисунке селектор уровня/фронта внешнего сигнала запроса прерывания выбирает событие, по которому вырабатывается запрос прерывания от внешнего сигнала («внешнее прерывание»). Возможны следующие настройки: по перепаду (фронт или спад сигнала) или по уровню. При возникновении запроса прерывания в регистре флагов (Interrupt Flag, IF) устанавливается бит, соответствующий источнику. Логика маскирования прерываний разрешает или запрещает выработку запросов от определенных источников или от всех источников сразу. Для разрешения прерывания необходимо установить в «1» соответствующий бит регистра масок (Interrupt Enable, IE) и бит общего разрешения прерываний (Global Interrupt Enable, GIE). Логика маскирования никак не влияет на немаскируемый запрос прерывания (Non-Maskable Interrupt, NMI). Приоритетная логика (Interrupt Priority, IP) вырабатывает вектор для наиболее приоритетного запроса и передает его вычислительному ядру синхронно с сигналом запроса прерывания; отслеживает приоритет запроса, находящегося в обработке и вытесняет (прерывает) данных запрос, если пришел другой запрос с более высоким приоритетом.

2.2.4 Модули памяти

Память – совокупность устройств, предназначенных для хранения программ, обрабатываемой информации (данных), промежуточных или окончательных результатов вычислений.

Важнейшие характеристики памяти – емкость, быстродействие и стоимость. Емкость ЗУ определяется предельным количеством информации, размещаемым в ЗУ, и исчисляется в кило-, мега- и гигабайтах. Быстродействие ЗУ характеризуется затратами времени на чтение запись информации при обращении к ЗУ. Стоимость ЗУ – это затраты средств в денежном выражении на хранение всего объема информации, определяемого емкостью ЗУ. Для сравнения качества ЗУ различных типов используется характеристика, называемая удельной стоимостью и равная стоимости ЗУ, деленной на емкость ЗУ. Удельная стоимость имеет размерность, например, доллар/Мбайт.

В зависимости от назначения и особенностей реализации устройств памяти, по-разному подходят и к вопросам их классификации.

Критерии классификации:

1. По назначению;
2. По виду физического носителя (технология производства);
3. По организации доступа (адресный: произвольный, прямой (циклический), последовательный; ассоциативный доступ);
4. По возможности записи и перезаписи;
5. По энергозависимости/энергонезависимости;
6. По типу интерфейса;
7. По типу организации адресного пространства;
8. По удалённости и доступности для центрального процессора (первичная, вторичная, третичная память).

Термин «модуль памяти» подразумевает объединение собственно массивов ячеек памяти со специальными аналоговым и цифровыми схемами управления режимами записи–стирания, со схемами (и иногда источниками) электропитания, с регистрами управления режимами.

В качестве примера приведем классификацию модулей памяти по критерию энергозависимости. В этом случае модули памяти делятся на ПЗУ и ОЗУ.

Модули ПЗУ бывают:

- ПЗУ масочного типа (MaskROM) – записывается на заводе-изготовителе и не может быть изменено пользователем. Обладают высоким качеством хранения. Самый дешевый тип ПЗУ. Используются для изделий, выпускаемых большими партиями в несколько десятков тысяч штук.
- ПЗУ, однократно программируемые пользователем (One-Time Programmable ROM (OTPROM)). Программируется пользователем. При выпуске на заводе все ячейки имеют значения FFh. Подачей импульсов напряжения в биты могут быть записаны «0», но обратная запись – в «1» уже не возможна. Имеют высокое качество хранения при строгом соблюдении режимов программирования (уровни напряжения, временная диаграмма, режимы проверки), в противном случае через

некоторое время (месяцы-годы) биты могут самопроизвольно «распрограммироваться» - перейти в состояние «1». Дешевое ПЗУ. Используется для небольших партий изделий.

- ПЗУ программируемое пользователем, со стиранием ультрафиолетовыми или рентгеновскими лучами (EPROM). Допускается многократное перепрограммирование (несколько десятков раз). Технология программирования схожа с OTPROM, но может быть осуществлено стирание всех запрограммированных в «0» ячеек в состояние «1» под УФ лучами. Для того на корпусе есть специальное окно из кварцевого стекла. Нарушение режимов стирания программирования приводит к резкому сокращению числа циклов перепрограммирования и времени хранения. Очень дорогая память (примерно на порядок дороже чем OTPROM). Используется в отладочных образцах.
- ПЗУ программируемое пользователем с электрическим стиранием (Electrically Erasable Programmable ROM – EEPROM или E^2PROM). Допускается перезапись произвольной ячейки. При этом стирание выполняется автоматически, прозрачно для пользователя. Число циклов перезаписи до 10000..100000 шт. Значительное время хранения (годы .. 10 лет). Однако, блоки EEPROM имеют ограниченный объем (байты…десятки кБ), в связи с чем их почти всегда используют как память данных. Пример организации и принципа работы EEPROM [18] будет рассмотрен подробнее в подразделе 2.2.4.1.
- ПЗУ с электрическим стиранием типа FLASH является модификацией EEPROM со значительно увеличенным объемом. Для увеличения объема удалены схемы стирания каждого бита по отдельности и стирание выполняется страницами размером от десятков байт до десятков кБ. Также доступно стирание блоками по несколько страниц или всей памяти разом. Такой режим работы (страничное стирание) неудобен для хранения данных, но приемлем для записи программ. Поэтому память FLASH используется в качестве памяти программ. Объем встраиваемой FLASH-памяти – десятки-сотни кБ. Число циклов перепрограммирования – до 100000. Время хранения – до 10 лет. Время стирания –десятки ms на килобайт, время программирования – десятки мкс на байт. Напряжение питания от 1.8 В. ПЗУ типа FLASH в настоящее время выходит на ведущие позиции в секторе встраиваемых и внешних модулей (микросхем) памяти ПЗУ.

В качестве встроенного ОЗУ в большинстве случаев используются модули статической памяти (Static Random Access Memory, SRAM). Ядро микросхемы статической оперативной памяти представляет собой совокупность триггеров – логических устройств, имеющих два устойчивых состояния, одно из которых условно соответствует логическому нулю, а другое – логической единице. Другими словами, каждый триггер хранит один бит информации.

К достоинствам триггера по сравнению с конденсатором в динамических ОЗУ можно отнести:

- состояния триггера устойчивы и при наличии питания могут сохраняться бесконечно долго, в то время как конденсатор требует периодической регенерации;
- триггер, обладая мизерной инертностью, без проблем работает на частотах вплоть до нескольких ГГц, тогда как конденсаторы "сваливаются" уже на 75-100 МГц.

К недостаткам триггеров следует отнести их высокую стоимость и низкую плотность хранения информации. Если для создания ячейки динамической памяти достаточного всего одного транзистора и одного конденсатора, то ячейка статической памяти состоит как минимум из четырех, а в среднем шести – восьми транзисторов, поэтому статические ЗУ в 4...5 раз дороже динамических и приблизительно во столько же раз меньше по информационной емкости. Их достоинством является высокое быстродействие, а типичной областью применения – схемы кэш-памяти.

В динамических ОЗУ (DRAM) данные хранятся в виде зарядов конденсаторов, образуемых элементами МОП-структур. Саморазряд конденсаторов ведет к разрушению данных, поэтому они должны периодически (каждые несколько миллисекунд) регенерироваться. В то же время плотность упаковки динамических элементов памяти в несколько раз превышает плотность упаковки, достижимую в статических RAM.

Регенерация данных в динамических ЗУ осуществляется с помощью специальных контроллеров. Разработаны также ЗУ с динамическими запоминающими элементами, имеющие внутреннюю встроенную систему регенерации, у которых внешнее поведение относительно управляющих сигналов становится аналогичным поведению статических ЗУ. Такие ЗУ называют квазистатическими.

Динамические ЗУ характеризуются наибольшей информационной емкостью и невысокой стоимостью, поэтому именно они используются как основная память компьютеров.

Выбор статического ОЗУ в качестве модуля встроенной основной памяти, а не динамического, определяется возможностью хранения данных при снижении частоты вплоть до полной остановки процессора. Такой режим используется для энергосбережения, например, при питании от батарейки.

На современном этапе модули встроенного ОЗУ не обладают значительным объемом – единицы иногда десятки килобайт. В случае мощных систем требующих больших объемов ОЗУ подключаются внешние микросхемы памяти. Это могут быть как микросхемы статического ОЗУ, так и динамическое ОЗУ (DRAM, SDRAM). В последнем случае процессор (например, AMD Am186ED, Mitsubishi M16C и др.) имеет модуль интерфейса динамического

ОЗУ, который поддерживает интерфейс классических микросхем DRAM, или SDRAM, а так же регенерацию динамической памяти.

Второй важной особенностью современных модулей ОЗУ – низкое, примерно 1 В, напряжение хранения информации. Это позволяет сохранять данные при провалах питания. С этой же целью иногда в модуль статического ОЗУ включают батарейку электропитания, позволяющую хранить данные до 10 лет (семейство DS5000 Dallas Semiconductor).

2.2.4.1 Энергонезависимая память E^2PROM: историческая справка

В 1974 году в Intel пришел Джордж Перлегос (George Perlegos), грек по происхождению и будущий основатель компании Atmel. Под его руководством в 1983 была разработана микросхема EEPROM (кодовое название 2816). Основой EEPROM стал транзистор с плавающим затвором, изобретенный в той же Intel Доном Фрохманом (Don Frohman). И в дальнейшем, несмотря на смены технологических эпох, принцип устройства ячейки энергонезависимой памяти остался неизменным – какой бы способ стирания и записи ни использовался.

Ячейка памяти представляет собой МОП-транзистор с плавающим затвором, который окружен диоксидом кремния. Сток транзистора соединен с «землей», а исток подключен к напряжению питания с помощью резистора. В стертом состоянии (до записи) плавающий затвор не содержит заряда, и МОП-транзистор закрыт. В этом случае на истоке поддерживается высокий потенциал, и при обращении к ячейке считывается логическая единица. Программирование памяти сводится к записи в соответствующую ячейки логических нулей. Программирование осуществляется путем подачи на управляющий затвор высокого напряжения. Этого напряжения должно быть достаточно, чтобы обеспечить пробой между управляющим и плавающим затвором, после чего заряд с управляющего затвора переносится на плавающий. МОП-транзистор переключается в открытое состояние, закорачивается исток с землей. В этом случае при обращении к ячейке считывается логический нуль. Такой метод записи называется «инжекцией горячих электронов», слой окисла между плавающим затвором и подложкой при этом составляет 50 нм. Стирание содержимого ячейки происходит путем электрического соединения плавающего затвора с «землей».

Таким образом, в EEPROM образца 1980-х запись производилась «горячей инжекцией», а стирание – «квантовым туннелированием». Оттого микросхемы эти были довольно сложны в эксплуатации и требовали два, а то и три питающих напряжения, причем подавать их при записи и стирании требовалось в определенной последовательности.

Позже, в электрически стираемой памяти Джордж Перлегос предложил использовать "квантовый эффект туннелирования Фаулера-Нордхейма". За этим непонятным названием кроется довольно простое по сути (но очень сложное с физической точки зрения) явление: при достаточно тонкой пленке изолятора (в пределах 10 нм) электроны, если их слегка подтолкнуть подачей

не слишком высокого напряжения в нужном направлении, могут просачиваться через барьер, не перепрыгивая его.

Основным преимуществом использования памяти EEPROM заключается в возможности ее многократного перепрограммирования без удаления из платы. Такой способ программирования получил название «In-System Programming» или «ISP». При этом сокращаются затраты на программирование.

В процессе перезаписи, слой окисла постепенно накапливает захваченные электроны, которые со временем могут переходить в плавающий затвор. Тем самым уменьшается различие между пороговыми напряжениями соответствующими лог. «1» и «0». После достаточного количества циклов перезаписи, различие становится слишком маленьким для распознания. Как правило, минимальное число перезаписей доходит до сотен тысяч и миллиона раз.

Во время записи, электроны, инжектируясь в плавающий затвор, могут дрейфовать через изолятор, особенно при повышении температуры, и при этом возможна потеря заряда, то есть информация ячейки стирается. Производители обычно гарантируют, что данные будут храниться не менее 10 лет.

Далее будет рассматриваться память EEPROM AT24Cxx (Atmel) [18], которая установлена в учебном стенде SDK-1.1. Архитектура этот стенда – пример простейшего контроллера встраиваемых систем.

2.2.4.2 Основные характеристики EEPROM AT24Cxx

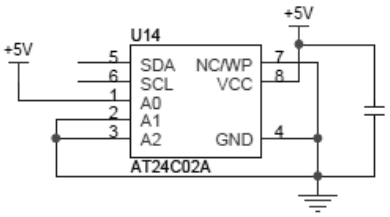
- Внутреннее ПЗУ 128x8 (1 K), 256x8 (2 K), 512x8 (4 K), 1024x8 (8 K) или 2048x8 (16 K).
- Двухпроводной последовательный интерфейс (I2C).
- Двунаправленный протокол передачи данных.
- Совместимость по частоте 100 кГц (1.8 V, 2.5 V, 2.7 V) и 400 кГц (5 V).
- Вывод защиты записи, обеспечивающий аппаратную защиту данных.
- Поддержка страничной записи в 8-байтном (1 K, 2 K), и 16-байтном (4 K, 8 K, 16 K) режимах.
- Поддержка неполной страничной записи.
- Самосинхронизирующийся цикл записи (максимум – 10 мс).
- Высокая надежность:
  - ♦ Продолжительность работы 1 миллион циклов перезаписи;
  - ♦ Сохранение данных в памяти в течение 100 лет.
- Автоматическая градуировка и возможность работы в широком диапазоне температур.

2.2.4.3 Описание

Микросхема AT24C01A / 02 / 04 / 08 / 16 – это 1024 / 2048/ 4096/ 8192/ 16384 бит последовательной памяти E^2PROM (128 / 256 / 512 / 1024 / 2048 байт), которая может быть перезаписана с помощью электрических сигналов и считана программным путем. Данное устройство предназначено для применения в промышленных и коммерческих областях, где важным условием является низкое энергопотребление и малое напряжение питания.

Таблица 4. Назначение выводов микросхемы.

Вывод	Назначение
A0 – A2	Адресные входы.
SDA	Последовательная передача данных.
SCL	Линия синхронизации.
WP	Защита от записи.
NC	Не используется.



SERIAL CLOCK (SCL) – линия синхронизации.

Вход SCL используется при передаче в E^2PROM (положительный фронт) и отправке данных на любое внешнее устройство (отрицательный фронт).

SERIAL DATA (SDA) – линия последовательной передачи данных.

SDA – вывод для двунаправленной последовательной передачи данных. Это вывод со свободным стоком, к нему можно подключать любое количество открытых коллекторов или коллекторов со свободным стоком.

Выводы A2, A1 и A0 – это адресные входы устройств, разработанные для микросхем AT24C01A и AT24C02. К одной шине может быть подключено до 8 1К/2К – устройств.

Микросхема AT24C04 использует для фиксированной адресации два вывода A2 и A1, что позволяет подключить к одной шине до четырех таких микросхем. Вывод A0 не используется.

Микросхема AT24C08 использует для фиксированной адресации только вывод A2, что позволяет подключить к одной шине до двух таких микросхем. Выводы A0 и A1 не используются.

Микросхема AT24C16 не использует адресные выводы. К одной шине можно подключить только одно устройство. Выводы A0, A1 и A2 не используются.

Схемы семейства AT24C01A / 02 / 04 / 16 имеют вывод защиты от записи (WP), с помощью которого можно защитить аппаратные данные. Этот вывод используется для обычных операций чтения/записи в случае, если он подключен к заземлению (GND). Когда на этот вывод подается напряжение, свойство защиты от записи проявляется так, как показано в таблице ниже.

Таблица 5. Защита от записи в схемах семейства AT24C01A/02/04/16.

Состояние вывода	Защита массива данных				
	24C01A	24C02	24C04	24C08	24C16
Напряжение	Полностью (1К)	Полностью (2К)	Полностью (4К)	Обычные операции чтения/записи	Верхняя половина массива (8К)
Земля	Обычные операции чтения/записи				

2.2.4.4 Организация памяти

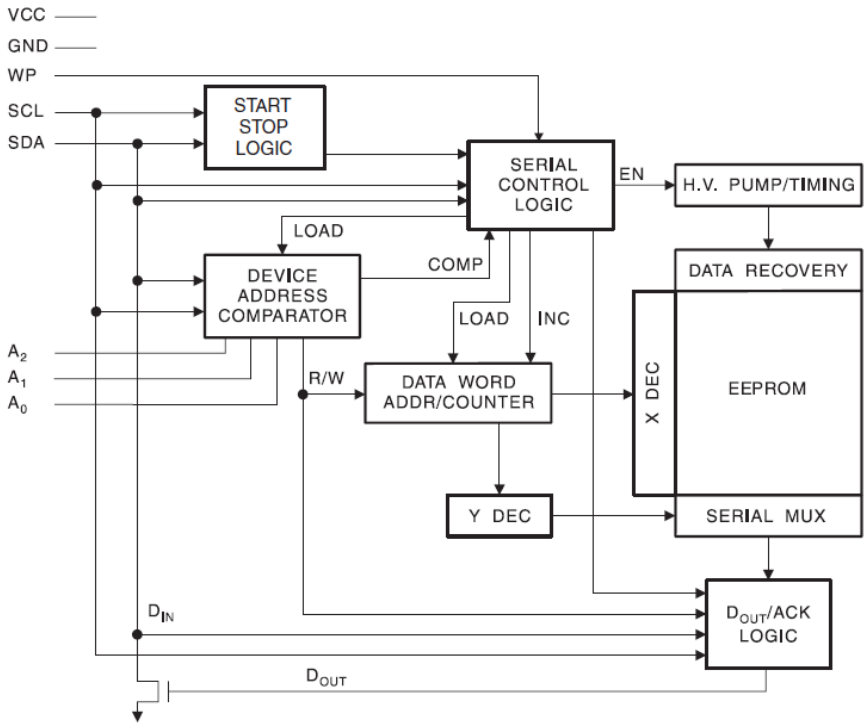


Рисунок 12. Структурная схема памяти EEPROM

AT24C01A, 1K последовательная E²PROM. Внутренняя память, состоящая из 128 однобайтовых страниц общим объемом в 1 К, для произвольного доступа к которой требуются 7-битные адреса.

AT24C02, 2K последовательная E²PROM. Внутренняя память, состоящая из 256 однобайтовых страниц общим объемом в 2К, для произвольного доступа к которой требуются 8-битные адреса.

AT24C04, 4K последовательная E²PROM. Внутренняя память объемом в 4К, состоящая из 256 страниц по 2 байта каждая. Для произвольного доступа к данным требуются 9-битные адреса.

AT24C08, 8K последовательная E²PROM. Внутренняя память объемом в 8К, состоящая из 4 блоков. Каждый блок содержит 256 4-байтных страниц. Для произвольного доступа к данным требуется 10-битная адресация.

AT24C16, 16K последовательная E²PROM. Внутренняя память объемом в 16K, состоящая из 8 блоков. Каждый блок содержит 256 8-байтных страниц. Для произвольного доступа к данным необходима 11-битная адресация.

Обозначения на схеме:

- Start-stop logic – логика «пуск-останов».
- Device Address Comparator – блок сравнения адреса.
- Serial Control Logic – последовательная логика управления.
- Data Word Addr/Counter – адрес/счетчик слова данных.
- H.V. Pump/Timing – генератор подкачки заряда / тактирование.
- Data Recovery – восстановление данных.
- Serial Mux – мультиплексор последовательной передачи.
- Dout/ACK Logic – логика подтверждения передачи сигнала.

2.2.4.5 Адресация модулей EEPROM

Микросхемы E²PROM с объемом памяти 1 K, 2 K, 4 K, 8 K и 16 K после перехода в старт-состояние должны получать слово (8 бит) с адресом устройства. Только тогда микросхема сможет произвести операцию чтения или записи (см. рисунок).

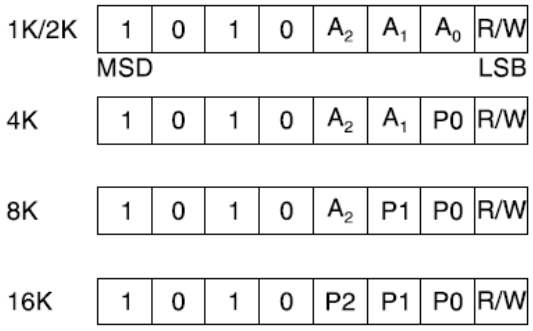


Рисунок 13. Адреса устройств

Первые четыре бита слова адреса представляют собой обязательную последовательность «1010». Данная последовательность одинакова для всех устройств E²PROM (данной серии, производства Atmel). Следующие 3 бита представляют собой адреса устройств A2, A1 и A0 для 1K/2K E²PROM. Эти биты соответствуют входам с аналогичными названиями.

E²PROM с объемом памяти 4 К использует только биты адресов A2 и A1, а P0 представляет собой адрес страницы памяти. Оба бита адресов устройств соответствуют выходам на микросхеме с аналогичными названиями. Вывод A0 не подключен.

Адресный байт для E²PROM с объемом памяти 8 К содержит только один бит адреса устройства A2, а биты P1 и P0 используются для адресации страницы памяти. Бит A2 соответствует выводу A2 на микросхеме. Выводы A1 и A0 не подключены.

В E^{2}PROM с 16 K памяти биты P0, P1 и P2 представляют собой адрес страницы памяти в устройствах 4 K, 8 K и 16 K. Выводы A0, A1 и A2 не подключены.

Восьмой бит адреса устройств используется для выбора режима чтения/записи. Если бит равен 1, происходит чтение, иначе – запись. После сравнения адресов устройств E^{2}PROM выдает 0. Если сравнение не было произведено, микросхема возвращается в режим ожидания.

2.2.4.6 Операция записи

Запись байта

После того, как E^{2}PROM получит адрес ячейки и подтвердит возможность приема, должна происходить операция записи. Получив адрес и ответив выдачей «0», устройство примет первые 8 бит данных. Затем E^{2}PROM выдает «0». Адресующее устройство (например, микроконтроллер) должно остановить процесс записи путем выдачи стоп-сигнала. В этот момент E^{2}PROM начинает цикл записи в постоянную память. До тех пор, пока запись не будет завершена, отключаются все входы и E^{2}PROM не реагирует ни на какие сигналы (см. рисунок).

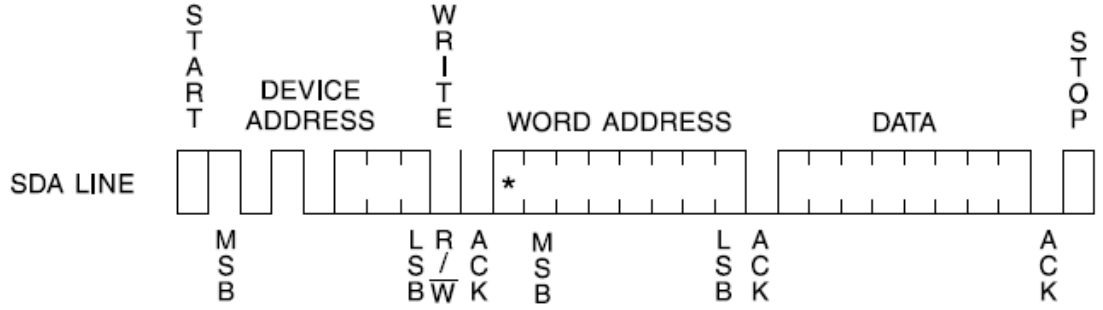


Рисунок 14. Запись байта

Список обозначений:

- WRITE – запись;
- ACK – сигнал подтверждения;
- SDA LINE – линия передачи данных SDA;
- DATA – данные;
- DEVICE ADDRESS – адрес устройства;
- WORD ADDRESS – адрес ячейки памяти.

Страничная запись

1K/2K E^{2}PROM может производить страничную запись (по 8 байт), а устройства с объемом памяти в 4 K, 8 K и 16 K производят 16-байтную запись.

Процесс страничной записи инициируется так же, как запись одного байта, отличие в том, что микроконтроллер после передачи первого слова не выдает стоп-сигнал.

Вместо этого, как только E^{2}PROM подтвердит получение первого слова данных, микроконтроллер может передать ему еще до 7 (1 К / 2 К) или 15 (4 К, 8 К, 16 К) слов данных. После получения каждого слова E^{2}PROM будет выдавать на линии «0» (подтверждение). Микроконтроллер прекращает страничную запись, выдавая стоп-сигнал (см. рисунок).

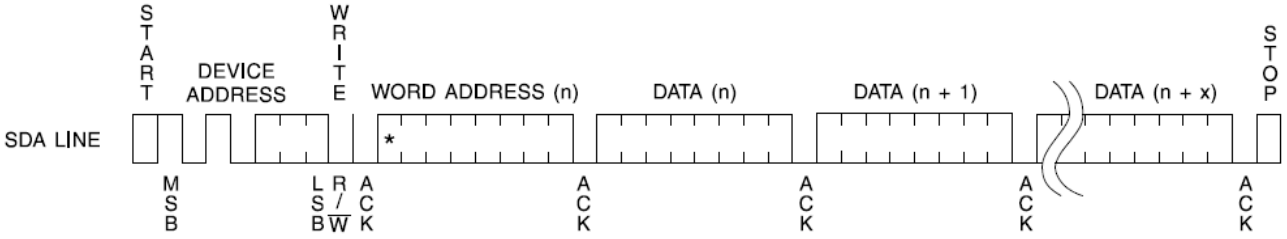


Рисунок 15. Запись страницы

Список обозначений:

- WRITE – запись;
- ACK – сигнал подтверждения;
- SDA LINE – линия передачи данных SDA;
- DATA – данные;
- DEVICE ADDRESS – адрес устройства;
- WORD ADDRESS – адрес ячейки памяти.

Каждый раз при получении слова данных E^{2}PROM инкрементирует младшие 3 (1 K / 2 K) или 4 (4 K, 8 K, 16 K) адресных бита. Старшие адресные биты не инкрементируются.

Во время записи последовательности байт счетчик адреса «перепрыгивает» с последнего байта текущей страницы на первый байт той же самой страницы.

Опрос устройства

Как только E^{2}PROM начнет внутренне тактируемый цикл записи и отключит свои входы, можно инициировать запрос на подтверждение получения данных. Этот процесс включает отправку слова с адресом устройства, а затем выдачу стоп-сигнала. Бит чтения/записи устанавливается в зависимости от требуемой операции. E^{2}PROM выставит «0» – это позволит продолжить запись или чтение, только после завершения своего внутреннего цикла.

2.2.4.7 Операция чтения

Операция чтения инициируется точно так же, как и операция записи, за тем исключением, что бит чтения/записи в слове адреса устройства устанавливается равным 1. Существует три операции чтения: чтение с текущего адреса, чтение в режиме произвольного доступа и последовательное чтение.

Чтение с текущего адреса

Внутренний счетчик адреса содержит последний адрес, к которому производилось обращение во время операции чтения или записи, увеличенный на единицу. Этот адрес остается корректным в промежутке между операциями до тех пор, пока микросхема работает (питание включено). Во время чтения счетчик адреса «перепрыгивает» с последнего байта последней страницы памяти на первый байт первой страницы.

После передачи микроконтроллером байта адреса устройства с битом чтения/записи, установленным в «1», и подтверждения его модулем E^2PROM, следующим микроконтроллеру передается байт данных по текущему адресу памяти. Микроконтроллер в ответ сигнализирует об окончании чтения: посылает NO ACK и стоп-сигнал (см. рисунок).

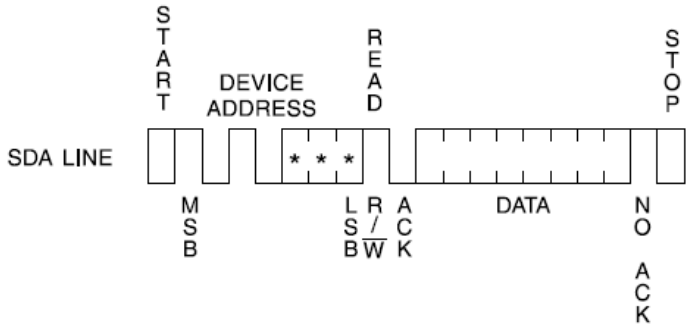


Рисунок 16. Чтение с текущего адреса

Список обозначений:

- READ – чтение;
- ACK – сигнал подтверждения;
- NO ACK – отсутствие подтверждения;
- SDA LINE – линия передачи данных SDA;
- DATA – данные;
- DEVICE ADDRESS – адрес устройства.

Чтение в режиме произвольного доступа

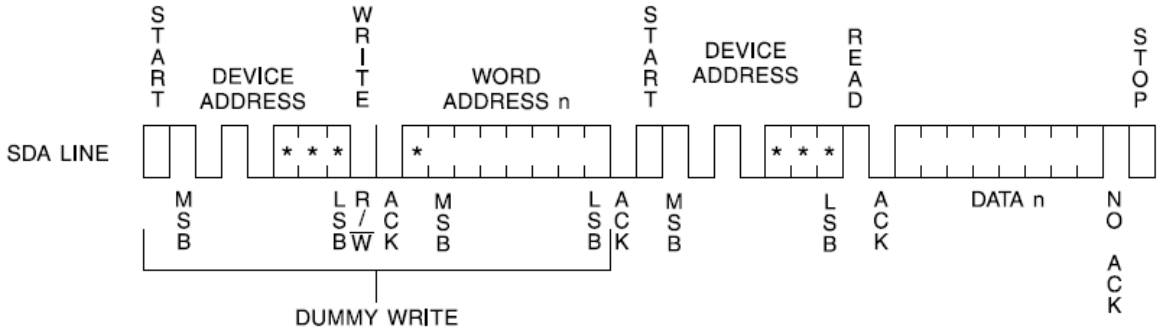


Рисунок 17. Чтение в режиме произвольного доступа

*Эти биты для памяти EEPROM емкостью 1 К могут быть любыми.

Список обозначений:

- READ – чтение;
- WRITE – запись;
- ACK – сигнал подтверждения;
- SDA LINE – линия передачи данных SDA;
- DATA – данные;
- DEVICE ADDRESS – адрес устройства;
- WORD ADDRESS – адрес ячейки памяти;
- DUMMY WRITE – холостая запись (установка счетчика адреса).

Как только слово адреса устройства и адрес слова данных будут приняты E^2PROM, микроконтроллер должен сгенерировать еще один старт-сигнал. После чего начинается чтение с текущего адреса, только что установленного (см. предыдущий раздел).

Чтение в режиме последовательного доступа

Последовательное чтение данных инициируется в процессе либо чтения с текущего адреса, либо чтения в режиме произвольного доступа – чтением последовательности байтов данных из EEPROM. Когда счетчик адреса достигнет последнего байта последней страницы памяти, он «перепрыгнет» на первый байт первой страницы. Операция последовательного чтения будет остановлена в том случае, если микроконтроллер сигнализирует об окончании чтения: посылает NO ACK и стоп-сигнал (см. рисунок).

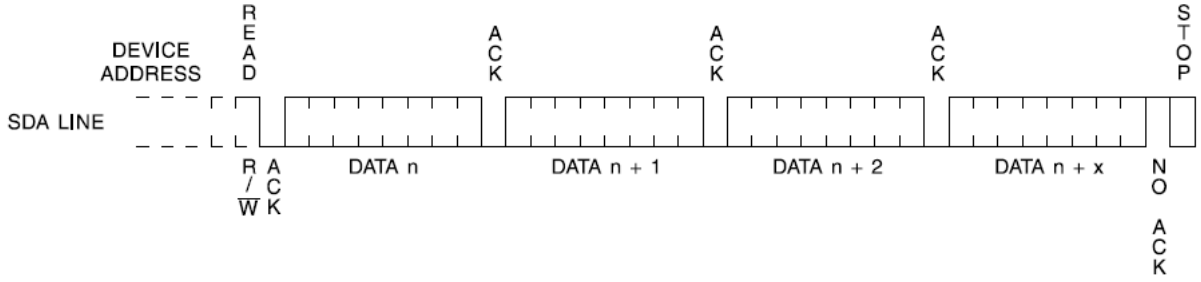


Рисунок 18. Последовательное чтение

Список обозначений:

- READ – чтение;
- ACK – сигнал подтверждения;
- NO ACK – отсутствие подтверждения;
- SDA LINE – линия передачи данных SDA;
- DATA – данные;
- DEVICE ADDRESS – адрес устройства.

2.2.5 Порты ввода-вывода

Каждый процессор для встраиваемых применений имеет некоторое количество внешних линий ввода-вывода, подключенных к внешним выводам микросхемы и называемых внешними портами. Одиночные (одноразрядные, состоящие из одной линии) порты ввода-вывода объединяются в группы, обычно, по 4, 8 или 16 линий, которые называются параллельными портами. Разрядность параллельных портов может быть нестандартной, например, 5-разрядный порт у микроконтроллера PIC16F84.

Через порты процессорное ядро взаимодействует с различными внешними устройствами – считывает значения входных сигналов и устанавливает значения выходных сигналов.

Во встраиваемых системах в качестве внешних устройств чаще всего рассматриваются датчики, исполнительные устройства, устройства ввода-вывода данных оператором, устройства внешней памяти.

По типу сигнала различают порты:

1. Дискретные (цифровые) – используются для ввода-вывода дискретных значений логического «0» или «1».

В большинстве современных процессоров для встраиваемых применений поддерживается как независимое управление каждой линией параллельного порта, так и групповое управление всеми разрядами. Так как схемотехника отдельных линий в рамках одного 4-х, 8-ми или 16-разрядного порта одинакова, то дальше будет рассматриваться устройство и функционирование одиночного разряда.

2. Аналоговые – через них вводятся сигналы на вход АЦП или других аналоговых схем и выводятся выходные сигналы ЦАП или других аналоговых схем.

Аналоговые порты (или перестраиваемые порты в аналоговом режиме) – используются подключения внешних сигналов к ЦАП, АЦП или аналоговым компараторам, встроенным приемопередатчикам. В режиме работы с ЦАП, АЦП или компаратором порты обычно позволяют вводить сигнал в диапазоне от 0В- до Uпит+ (индексы + и – означают чуть больше и чуть меньше, примерно на 200..300мВ). В режиме приемопередатчика параметры сигналов определяются конкретным интерфейсом. В большинстве случаев аналоговые или цифровые линии подключения к приемопередатчикам вообще не называют портами, хотя они по схемотехнике и по месту в структуре процессора близки к универсальным портам ввода-вывода. Реализация входных и выходных каскадов зависит от схемы АЦП, компаратора, ЦАП или приемопередатчика.

3. Перестраиваемые – настраиваются на аналоговый или цифровой режим работы.

По направлению передачи сигнала различают:

1. Однонаправленные порты, предназначенные только для ввода (входные порты, порты ввода) или только для вывода (выходные порты, порты вывода).
2. Двунаправленные порты, направление передачи которых определяется в процессе программно-управляемой настройки схемы.
3. Порты с альтернативной функцией. Отдельные линии этих портов связаны со встроенными периферийными устройствами, такими, как таймер, контроллеры последовательных приемопередатчиков. Если соответствующий периферийный модуль не задействован, то линии можно использовать как обычные порты, если модуль активизирован, то связанные с ним линии автоматически или «вручную» (программно) конфигурируются в соответствии с функциональным назначением и не могут быть использованы в качестве универсальных портов ввода-вывода. В некоторых случаях порты могут использоваться только для связи с периферийным модулем (например, входы АЦП в некоторых процессорах).

По алгоритму обмена различают порты:

1. С программно управляемым (программным) вводом-выводом – установка и считывание данных определяется только ходом вычислительного процесса. Нет защиты от повторного считывания-записи одного и того же (не изменившегося) значения на выводе и считывания-записи во время переходного процесса на выводе.
2. Со стробированием – каждая операция ввода вывода подтверждается импульсом синхронизации (стробом) со стороны источника сигнала (при выводе – процессор, при вводе – внешнее устройство). Считывание информации приемником происходит только по стробу, что позволяет защититься от приема данных во время переходного процесса входного сигнала. Пример: порт PSP (Parallel slave port) в ОКМЭВМ PICmicro.
3. С полным квитированием. Данный режим чаще всего используется для обмена данными с другой вычислительной системой по параллельной шине. Кроме сигналов синхронизации со стороны передатчика используются сигналы подтверждения (готовности к следующему обмену) со стороны приемника. Это позволяет управлять интенсивностью обмена обоим взаимодействующим сторонам и предотвращает потерю данных, когда одна из них перегружена. Пример порта с квитированием – порт LPT персонального компьютера. Во встроенных модулях процессоров данный режим чаще всего реализуется программно-аппаратно.

2.2.5.1 Однонаправленные порты

Схема однонаправленного порта ввода представлена на рисунке.

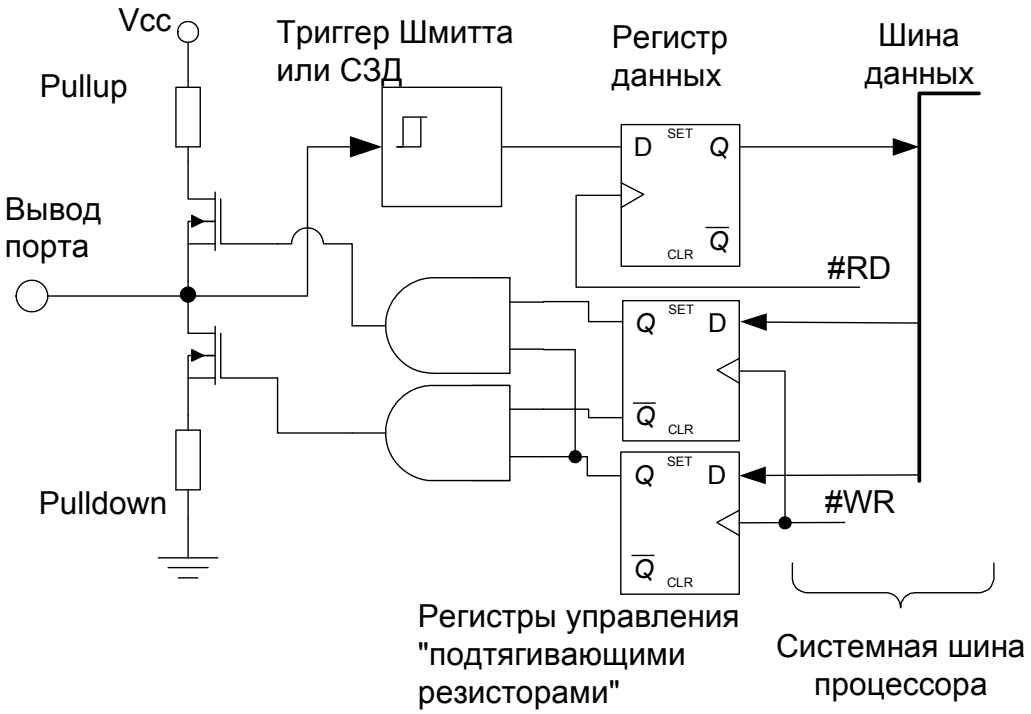


Рисунок 19. Однонаправленный порт ввода

Внешние данные считываются через вывод порта (ножку микросхемы), проходят через триггер Шмитта (ТШ) или схему защиты от дребезга (СЗД) и по внутреннему сигналу чтения фиксируются в регистре данных, с выхода которого, в свою очередь, данные считываются процессором.

ТШ (используется в большинстве процессоров для встроенного применения) имеет гистерезис по уровню входного напряжения и предотвращает многократное переключение входных схем при пологом фронте сигнала или помехах.

СЗД (например, в семействе Zilog Z8) вводит инерционность переключения и отсекает реакцию на короткие по длительности импульсы. Используется для защиты от помех.

К входу также могут подключаться так называемые «резисторы поддержки» логической «1» (Pull-up) или логического «0» (Pull-down). Эти резисторы предназначены для переведения входов в устойчивое состояние «0» или «1» и предотвращения произвольных переключений от помех в моменты, когда на них (входы) не подается внешний сигнал, например, неиспользуемых и неподключенных к внешним схемам входов («открытых входов»). Через специальные управляющие регистры «схемы поддержки» могут быть отключены полностью или включены в режим Pull-up или Pull-down.

Все перечисленные блоки – триггер Шмитта, СЗД и «схемы поддержки» используются для защиты от случайных переключений в результате помех и помогают снизить энергопотребление, которое резко возрастает в момент переключений входных схем.

Порты вывода бывают:

- с двухтактной выходной схемой (комплементарные);
- с однотактной выходной схемой и внутренней нагрузкой;
- с открытым выходом (открытым коллектором или стоком).

Порты вывода с двухтактной выходной схемой являются самыми распространенными и реализованы, например, в семействах Atmel AVR, Microchip PICmicro, AMD AM186, Motorola HC08, HC11 и многих других.

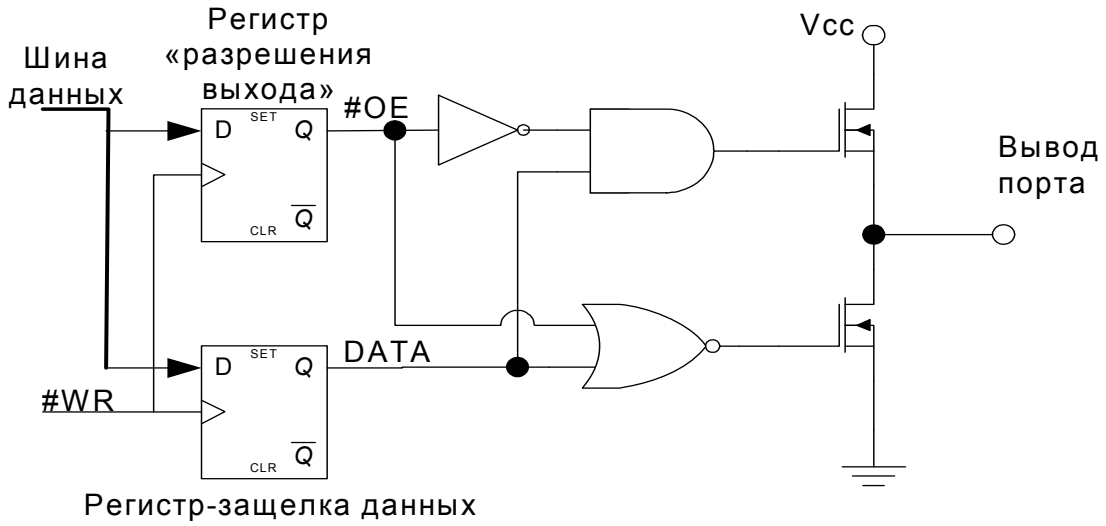


Рисунок 20. Порт вывода с двухтактной схемой

Рассмотрим функционирование данной схемы.

Выходные данные записываются в регистр-защелку данных по внутреннему сигналу записи #WR и через простейшую логическую схему управляют выходными транзисторами. Если в регистр записано значение «1», то открыт верхний по схеме транзистор, а нижний закрыт: на выходе Vcc, то есть «1». Если в регистр записано значение «0», то открыт нижний по схеме транзистор, а верхний закрыт: выход соединен с минусовой шиной питания, то есть там установлен «0».

Верхний по схеме регистр управляет сигналом #OE - «разрешение выходов». Если в регистр записан «0», то схема работает, как было описано выше. Если записана «1», то оба транзистора закрываются и схема переводится в «высокоомное» состояние (Z-состояние). В этом состоянии выходное сопротивление порта очень высокое и он фактически «оторван» от микропроцессора. Это необходимо:

- Если к выходному порту подключены выходы других схем и необходимо разделять линии передачи данных с этими устройствами. Например: наш процессор используется как периферийный контроллер и его выходной порт подключен к периферийной шине другого процессора (мастера), и к шине также подсоединены еще несколько периферийных контроллеров;
- В схемах двунаправленных портов (см. ниже).

Достоинства:

Значительный максимальный втекающий (в состоянии «0») и вытекающий (в состоянии «1») ток выхода: 2..6mA для каскадов с нормальной нагрузочной способностью (например, Fujitsu MB90) и 5..30mA для каскадов с повышенной нагрузочной способностью (например, PICmicro, AVR). Встречаются отдельные микросхемы со сверхвысокой нагрузочной способностью – до 60..90mA (например, PIC17). Большой выходной ток позволяет непосредственно с ножки, без схем усиления и согласования сигнала, управлять достаточно мощной нагрузкой: светодиодами, реле, мощным электронным ключем (транзистор, тиристор). Это значительно упрощает схему устройств.

Недостатки:

- При программировании необходимо управлять дополнительным битовым регистром «разрешение выхода»;
- Значительное энергопотребление и уровень помех при переключении. Последний особо зависит от скорости переключения. Для ограничения токов в момент переключений иногда используют специальные демпфирующие схемы. Однако они снижают быстродействие портов. Наибольшее применение демпфирующие схемы находят в портах ПЛИС в силу их особо высокого быстродействия;
- Относительно сложная внутренняя схема повышающая сложность и стоимость микросхемы в целом. Однако на нынешнем этапе, в связи с успехами технологии производства микросхем, это уже не является проблемой.

Порты вывода с однотактной выходной схемой и внутренней нагрузкой применяются, например, в семействе MCS-51. Они имеют более простую внутреннюю схему.

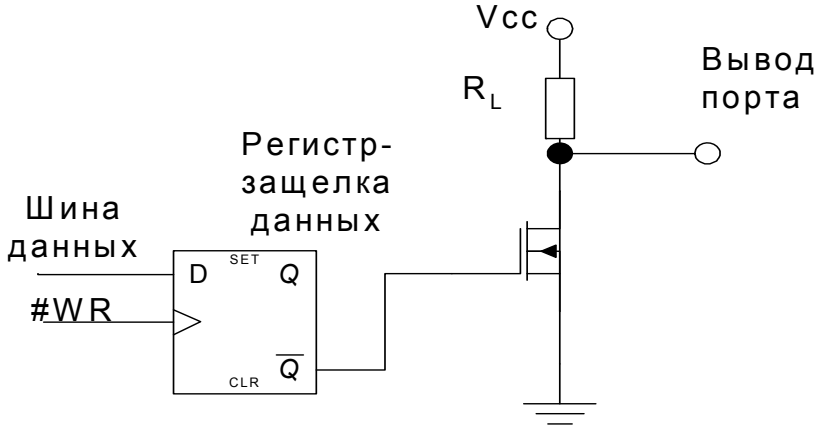


Рисунок 21. Порт вывода с однотактной схемой

Когда в регистр-защелку записано значение «1», транзистор закрыт и на выходе через резистор R_L устанавливается Vcc – логическая «1». Когда же в регистр-защелку записан «0», открывается транзистор и соединяет выход с минусовой шиной питания, то есть там устанавливается «0». При этом резистор R_L оказывается подключенным между шинами питания. Во избежания

высокого тока через резистор и его перегрева, сопротивление делают достаточно высоким – 10..100кОм. Высокое сопротивление резистора позволяет непосредственно соединять несколько выходов, не опасаясь их встречного включения, так как если «0» на одном из выходов «подсадит» «1» на другом, то мощность, выделяемая на «подсаженном» резисторе будет мала, он не перегреется и каскад нбе выйдет из строя.

Достоинства:

- Необходимо управлять только одним регистром;
- Простая схема;
- Возможность без дополнительных схем организовать подключение на одну внешнюю шину несколько таких выходов. Легко построить квазидвунаправленный порт ввода-вывода (см. ниже).

Недостатки:

- Малый вытекающий ток (в состоянии «1»), ограниченный резистором R_L – сотни uA. Это не дает управлять относительно мощными нагрузками без дополнительных каскадов усиления либо требует обеспечивать, чтобы активным был сигнал со значением «0» («управление нулем»).

Порты вывода с открытым выходом (открытым коллектором или стоком)

Применяются во многих семействах микропроцессоров, например, AMD Am186 (там это один из режимов порта), PICmicro. Выходной каскад построен по однотактной схеме с внешней нагрузкой. Принцип функционирования аналогичен описанному для однотактного выходногокаскада.

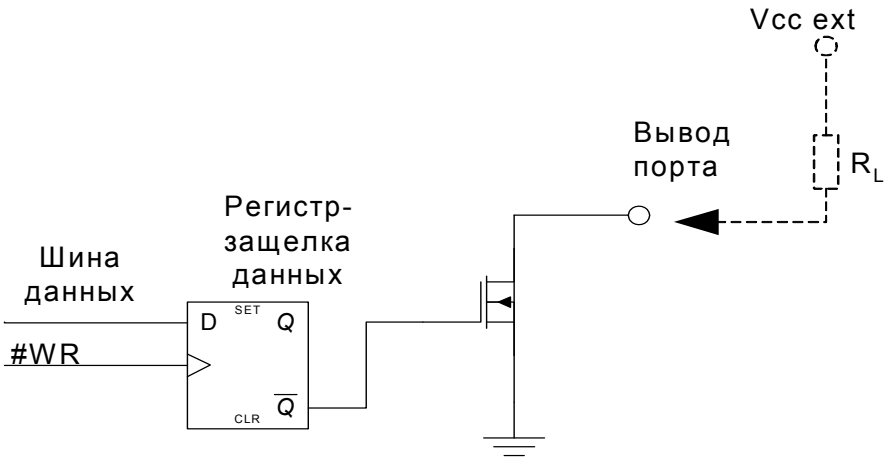


Рисунок 22. Порт вывода с открытым выходом

Достоинства:

- Внешнее напряжение питания нагрузки Vcc ext может быть иным – большим или меньшим, чем питание микропроцессора. Это может быть удобным для сопряжения схем с различными уровнями логической «1», например, 3.3В и 5В. Если внешнее напряжение достаточно высокое, то можно непосредственно управлять высоковольтной нагрузкой.

Например, анонсирован микроконтроллер семейства PICmicro допускающий подключение внешнего напряжения Vcc ext до 15В при питании ядра 2..6В.

- Необходимо управлять только одним регистром;
- Простая схема;
- Возможность без дополнительных схем организовать подключение на одну внешнюю шину несколько таких выходов. При этом можно подбирать требуемое сопротивление R_L, например, стандарт I^2C требует чтобы сопротивление было 2.2кОм. Легко построить квазидвунаправленный порт ввода-вывода (см. ниже).

Недостатки:

- Требуется внешняя нагрузка;
- Малый вытекающий ток (в состоянии «1»), ограниченный внешним нагрузочным резистором.

2.2.5.2 Двунаправленные порты и порты с альтернативной функцией

Самой простой схемой двунаправленного порта является квазидвунаправленный порт со схемой, аналогичной схеме порта вывода с однотактным выходным каскадом.

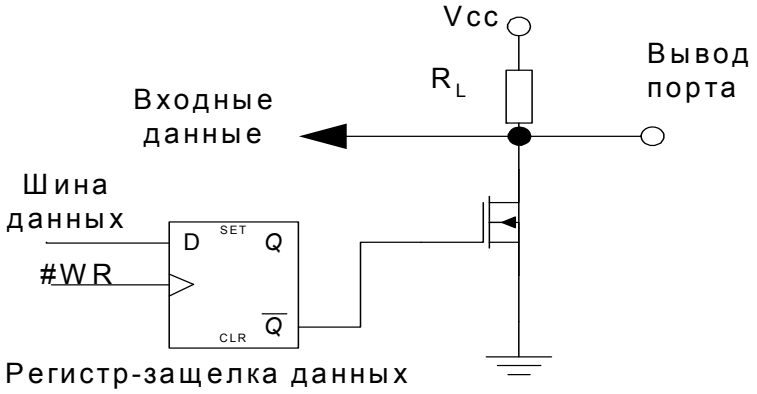


Рисунок 23. Квазидвунаправленный порт

Регистр входных данных (на схеме не показан) подключен к внешнему выводу порта. Перед считыванием входных данных необходимо предварительно записать «1» в регистр-защелку выходных данных. Это закроет транзистор и исключит влияние порта вывода на входной сигнал. Резистор R_L останется подключенным к входному сигналу и будет являться для него дополнительной нагрузкой, однако, так как сопротивление резистора велико (10..100 кОм), то даже на маломощный входной сигнал данная нагрузка не окажет заметного влияния. Схема квазидвунаправленного порта используется в семействе MCS-51.

Более часто используется схема переключаемого двунаправленного порта с комплементарным выходным каскадом.

Она объединяет схемы порта ввода и порта вывода с двухтактной выходной схемой, описанные выше. Переключение порта в режим ввода осуществляется записью «1» в регистр «вход/выход». В этом случае (как было указано при описании порта вывода) оба транзистора переводятся в закрытое состояние и порт вывода не влияет на входной сигнал. В двунаправленных портах резисторы pull-up и pull-down подключаются только в режиме ввода, для чего на вход соответствующей схемы управления подключается выход регистра «вход/выход» («1» - ввод).

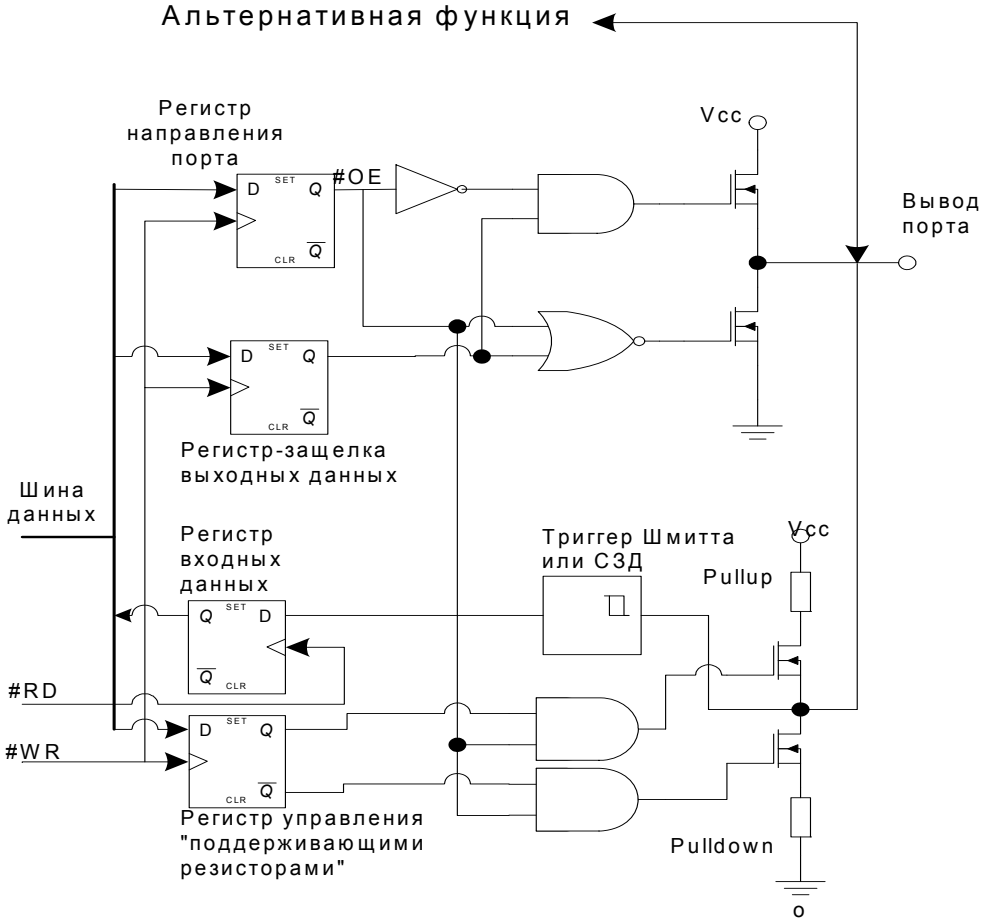


Рисунок 24. Переключаемый двунаправленный порт с комплементарным выходным каскадом

Кроме исполнения функции порта ввода-вывода, внешние выводы микросхемы могут быть задействованы для связи с внутренними периферийными модулями микропроцессора, а так же с подсистемами процессорного ядра, схем памяти и управления (с контроллером прерываний, блоком интерфейса внешней памяти и т.п.). Данные функции называются альтернативными. Обычно, когда вывод порта используется для выполнения альтернативной функции основные схемы переводятся в состояние ввода или вообще отключаются.

2.2.6 Таймеры-счетчики

Таймеры-счетчики предназначены для:

- Подсчета временных интервалов (режим таймера);
- Подсчета числа импульсов («внешних событий») на специальном внешнем входе (режим счетчика).

Режим таймера

Тактирование счетчика выполняется от сигнала внутренней синхронизации процессора Fint. Обычно это частота процессорных циклов формируемая от основного генератора. Подсчет временных интервалов выполняется в периодах сигнала Fint.

Предделитель используется для снижения тактовой частоты, подаваемой на регистр-счетчик. Это позволяет подсчитывать в более длительные интервалы, но увеличивает шаг дискретизации, а соответственно уменьшает точность. Предделитель может быть с фиксированным или программируемым коэффициентом деления. У программируемых предделителей обычно выбирается коэффициент деления из ряда 1, 2, 4, 8, …

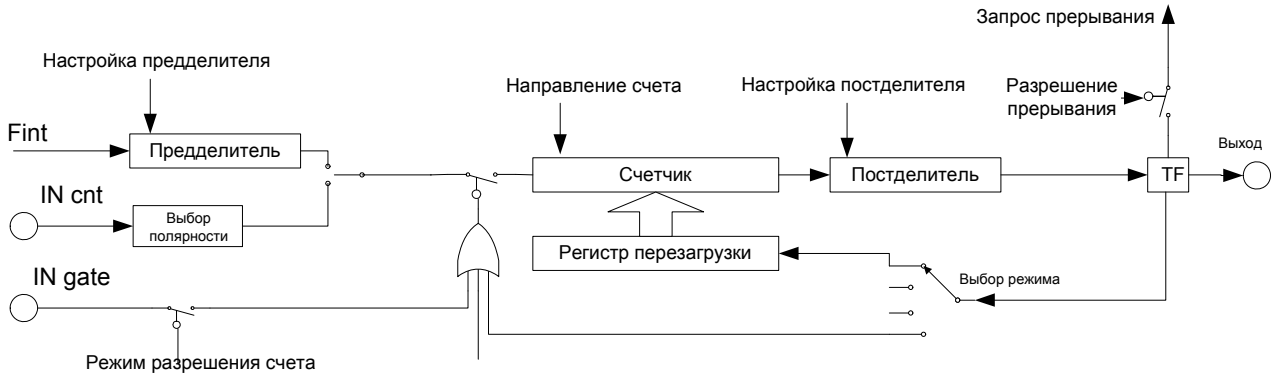


Рисунок 25. Структурная схема таймера-счетчика

Регистр-счетчик накапливает (считает) значение временного интервала в единицах входных тактов счетчика (после предделителя). Разрядность регистра-счетчика определяет разрядность всего таймера-счетчика.

Постделитель встречается достаточно редко (PICmicro) и служит для увеличения периода установки флага переполнения TF. Обычно постделитель – это дополнительные разряды регистра-счетчика недоступные по чтению-записи. Постделитель обычно программируемый на разные коэффициенты деления как и предделитель.

TF – флаг переполнения таймера. Устанавливается при переходе всех разрядов регистра счетчика-постделителя из 1 в 0. Обычно используется для указания окончания временного интервала. По нему может вырабатываться запрос прерывания.

От флага TF идет цепь обратной связи, задающая режим работы таймера:

1. Однократный счет: после переполнения в регистр-счетчик загружается значение 0 и счет останавливается. Запуск следующего цикла – специальной командой из программы;
2. Циклический счет с полным циклом: после переполнения в регистр-счетчик загружается значение 0 и счет начинается снова. Полный цикл счета таймера будет 2^k тактов, где k – разрядность счетчик + постделитель.
3. Циклический счет с автоперезагрузкой: после переполнения в регистр счетчик загружается значение из регистра перезагрузки. Таким образом счет можно начинать не с 0 и уменьшается (программируется) длительность цикла таймера.

Во многих процессорах имеется специальный вывод INgate, который выполняет функцию разрешения счета внешним сигналом. С помощью этого механизма легко подсчитывать длительность временного интервала, определяемого длительностью импульса на входе INgate.

Режим счетчика

В отличие от режима таймера, в режиме счетчика выбирается тактирование от внешнего импульсного сигнала, подаваемого на вход INcnt. При этом подсчитываются импульсы внешнего сигнала. Инкрементация или декрементация счетчика происходит по перепаду (фронту) сигнала. Фронт сигнала в данном случае называют «внешним событием». Полярность фронтов можно программировать.

В остальном функционирование в режимах счетчика и таймера аналогично.

2.2.6.1 Программируемые таймеры в микроконтроллере с ядром Intel MCS-51

Микроконтроллер ADuC812 имеет три программируемых 16-битных таймера/счетчика: Таймер 0, Таймер 1, Таймер 2. Каждый таймер состоит из двух 8-битных регистров THX и TLX. Все три таймера могут быть настроены на работу в режимах “таймер” или "счетчик".

В режиме "таймер" регистр инкрементируется каждый машинный цикл, т.е. можно рассматривать это как подсчет машинных циклов. Так как машинный цикл состоит из 12 перепадов напряжения на тактовом входе микроконтроллера, частота инкрементирования таймера в 12 раз меньше тактовой частоты микроконтроллера (соответствующего кварцевого резонатора).

В режиме "счетчик" регистр инкрементируется по перепаду из "1" в "0" внешнего входного сигнала, подаваемого на вывод микроконтроллера T0, T1 или T2. Когда опрос показывает высокий уровень в одном машинном цикле и

низкий уровень в следующем, счетчик увеличивается на 1. Таким образом, на распознавание периода требуются два машинных цикла, максимальная частота подсчета входных сигналов равна 1/24 частоты кварцевого резонатора. На длительность периода входных сигналов ограничений сверху нет. Для гарантированного прочтения входной сигнал должен удерживать значение 1, как минимум, в течение одного машинного цикла микроконтроллера.

Для конфигурации и контроля Таймеров используются 3 регистра специального назначения: TMOD и TCON для управления Таймерами 0 и 1, T2CON для управления Таймером 2.

Схемы управления Таймерами 0 и 1 идентичны (оба Таймера входят в ядро Intel MCS-51, Таймер 2 – нет). Далее рассмотрим принцип работы этих таймеров.

Таймер 0 и Таймер 1 могут работать в четырех режимах работы:

- режим 0: 13-битный таймер
- режим 1: 16-битный таймер
- режим 2: 8-битный автоперезагружаемый таймер
- режим 3: Таймер 0 как 2 раздельных 8-битных таймера.

Кроме того, Таймер 1 можно использовать для задания скорости передачи (baud rate) последовательного порта [1, 51].

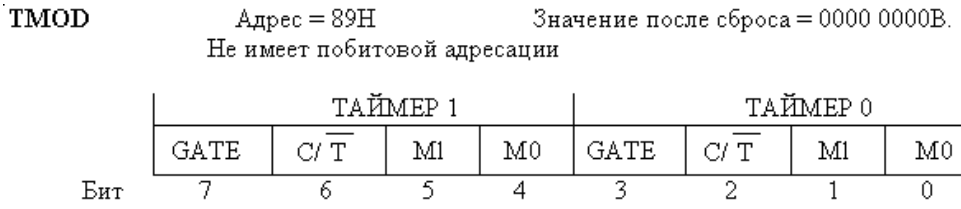


Рисунок 26. Формат регистра управления режимами работы таймеров TMOD.

Так как управление таймерами 0 и 1 полностью идентично, то приведём назначение битов по именам:

Название	**Позиция бита**	**Назначение**
GATE	TMOD.7 для таймера 1 и TMOD.3 для таймера 0	Управление блокировкой таймера от сигнала INTx. Если бит установлен в 1, то таймер/счетчик "x" разрешен до тех пор, пока на входе "lNTx" высокий уровень и бит управления "TRx" установлен. Если бит сброшен в 0, то T/C разрешается, как только бит управления "TRx" устанавливается в 1.
C/T	TMOD.6 для таймера 1 и TMOD.2 для таймера 0	Бит выбора режима таймера или счетчика событий. Если бит сброшен в 0, то таймер работает от внутреннего генератора, если установлен в 1, то работает от внешних сигналов на входе "Tx".

M1	TMOD.5 для таймера 1 и TMOD.1 для таймера 0	Выбор режима работы таймера
M0	TMOD.4 для T/C1 и TMOD.0 для T/C0	

M1	M0	
0	0	13 битный таймер/счетчик "TLx" работает как 5-битный предварительный делитель
0	1	16 битный таймер/счетчик. "THx" и "TLx" включены последовательно
1	0	8-битный автоперезагружаемый таймер/счетчик. "THx" хранит значение, которое должно быть перезагружено в "TLx" каждый раз по переполнению
1	1	Таймер/счетчик 1 останавливается. Таймер/счетчик 0: TL0 работает как 8-битный таймер/счетчик, и его режим определяется управляющими битами таймера 0. TH0 работает только как 8 битный таймер, и его режим определяется управляющими битами таймера 1

Схема управления Таймерами 0 и 1 идентична и для таймера T0 приведена на рисунке. В приведенной схеме заштрихованным прямоугольником обозначены внешние выводы микросхемы микроконтроллера [1, 51]:

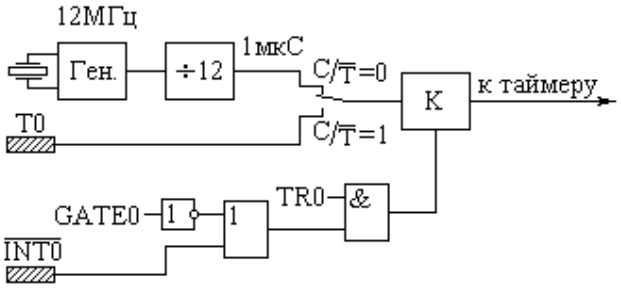


Рисунок 27. Схема управления Таймерами 0 или 1

Из схемы видно, что таймер может включаться и выключаться битами TRx. Таким образом, можно уменьшать потребление микросхемы и уровень помех, создаваемый ею. Учитывая, что счетчики таймеров переключаются на высокой частоте, то они могут потреблять до половины тока потребления микроконтроллера. Следует отметить, что при включении и после сброса микроконтроллера работа таймеров запрещена.

Есть возможность управлять работой таймера извне при помощи внешнего вывода T0 для таймера T0 или T1 для таймера T1. Для этого необходимо записать в бит GATEx логическую единицу (не забыв при этом разрешить работу таймера при помощи бита TRx). Установка GATE=1приводит к тому, что работа таймера контролируется внешним входом INTx, что позволяет измерять длительность импульса.

Кроме того, таймер может синхронизироваться от внешнего генератора. Для этого в бит управления C/T нужно записать логическую единицу.

Биты включения таймеров TR0 и TR1 размещены в регистре TCON (control - управлять), а биты GATE и C/T в регистре TMOD. Формат регистра TCON приведён на следующем рисунке [1, 51]:

Рисунок 28. Формат регистра управления режимами работы таймеров TCON

Символ	Позиция	Назначение
TF1	TCON.7	Флаг переполнения таймера 1. Устанавливается аппаратно при переполнении таймера/счетчика. Сбрасывается при обслуживании прерывания аппаратно.
TR1	TCON.6	Бит управления таймера 1. Устанавливается/сбрасывается программой для пуска/останова.
TF0	TCON.5	Флаг переполнения таймера 0. Устанавливается аппаратурно. Сбрасывается при обслуживании прерывания
TR0	TCON.4	Бит управления таймера 0. Устанавливается / сбрасывается программой для пуска/останова таймера/счетчика.
IE1	TCON.3	Флаг внешнего прерывания 1. Устанавливается аппаратно, когда детектируется срез внешнего сигнала INT1. Сбрасывается при обслуживании прерывания.
IT1	TCON.2	Бит управления типом прерывания 1. Устанавливается / сбрасывается программно для определения типа запроса прерывания INT1 (по спаду/по низкому уровню).
IE0	TCON.1	Флаг внешнего прерывания 0. Устанавливается по срезу сигнала INT0. Сбрасывается при обслуживании прерывания.
IT1	TCON.0	Бит управления типом прерывания 0. Устанавливается / сбрасывается программно для определения типа запроса прерывания INT0 (по спаду/по низкому уровню).

Режим 0 (13-битный Таймер/Счетчик)

В данном режиме Таймер X работает как 13-битный суммирующий таймер/счётчик. Этот таймер/счётчик состоит из 8 бит регистра THx и младших 5 бит регистра TLx, где x в обозначении регистра заменяется на 0 или 1 в зависимости от того таймера, которым мы управляем. Старшие 3 бита регистров TLx не определены и игнорируются. Таким образом, этот режим можно трактовать как 8-битный таймер/счетчик с предварительным делителем (на 32) на входе. Установка запускающего таймер флага TR0 или TR1 не

очищает эти регистры. Работе таймера 0 или таймера 1 в режиме 0 соответствует схема:

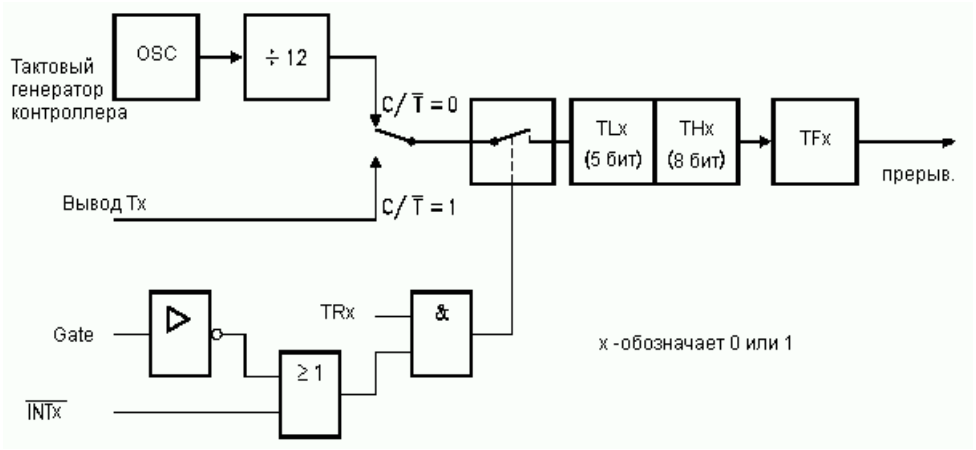


Рисунок 29. Работа таймера в режиме 0.

Когда содержимое счетчика изменяется из состояния все "1" в состояние все "0" , то устанавливается (принимает значение "1") флаг прерывания таймера TF0 или TF1.

Режим 1 (16-битный Таймер/Счетчик)

В первом режиме работы Таймер X работает как шестнадцатиразрядный таймер/счётчик. Режим 1 похож на режим 0, за исключением того, что в регистрах таймера использует все 16 бит. В этом режиме регистры THx и TLx также включены друг за другом. Работе таймера 0 или таймера 1 в режиме 1 соответствует схема:

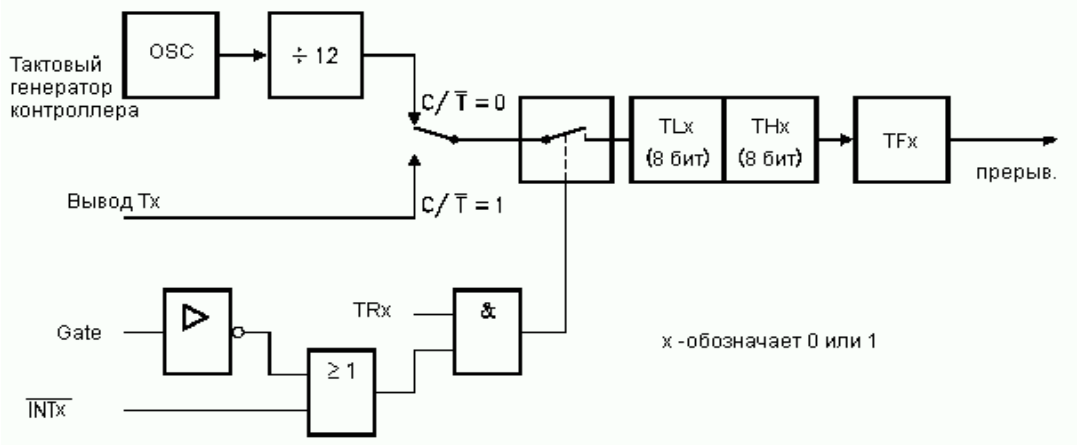


Рисунок 30. Работа таймера в режиме 1.

Нулевой и первый режимы работы Таймеров 0 и 1 предназначены для формирования одиночного интервала времени. Если возникает необходимость формировать последовательность интервалов времени для периодических процессов, то загрузка регистров TH0 и TL0 для задания нужного интервала времени производится программно, что для коротких интервалов времени может привести к значительным затратам процессорного времени.

Для формирования последовательности одинаковых интервалов времени используется режим работы таймера с автоперезагрузкой – режим 2.

Режим 2 (8-битный Таймер/Счетчик с автоперезагрузкой)

В режиме 2 регистр таймера TLx работает как 8-битный счетчик с автоматической перезагрузкой начального значения из регистра THx в регистр TLx. Переполнение регистра TLx не только устанавливает флаг TFx, но и загружает регистр TLx содержимым регистра THx, который предварительно инициализируется программно. Перезагрузка не изменяет содержимое регистра THx. Работе таймера 0 или таймера 1 в режиме 2 соответствует схема:

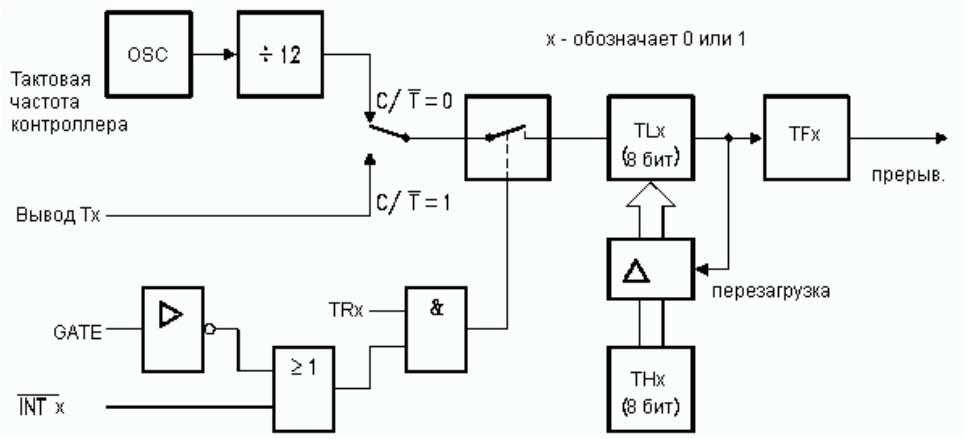


Рисунок 31. Работа таймера в режиме 2.

Режим 3 (Два 8-битных Таймера/Счетчика)

Работа в режиме 3 имеет отличия для таймеров 0 и 1. Таймер 1 в этом режиме просто останавливает свой счет. Тот же эффект даст установка TR1=0. Логика работы таймера 0 в режиме 3 показана на схеме:

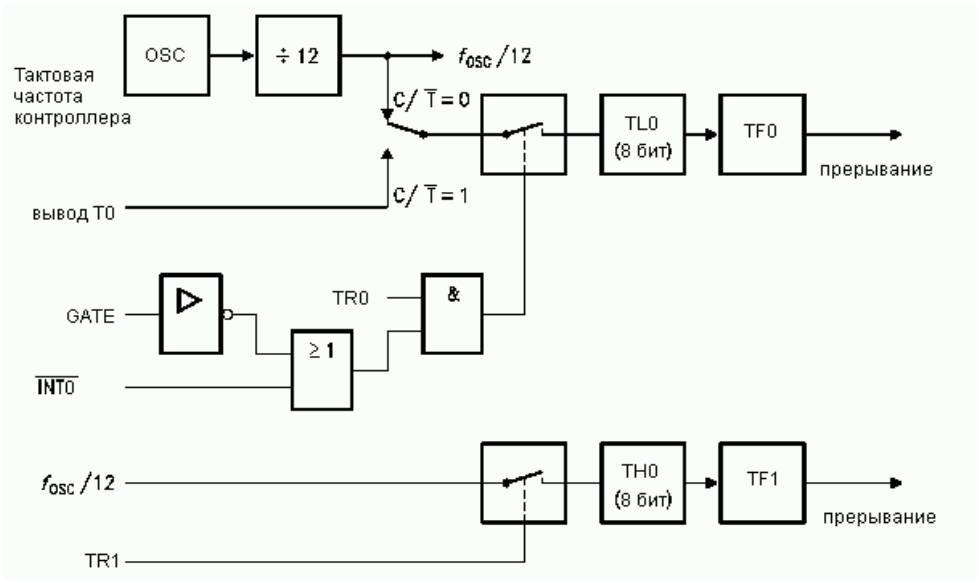


Рисунок 32. Работа таймера в режиме 3.

Таймер 0 в режиме 3 устанавливает TL0 и TH0 как два разных счетчика. Счетчик на базе TL0 использует биты управления таймера 0: C/T, GATE, TR0, INT0, TF0. TH0 зафиксирован в режиме таймера (считающего машинные

циклы), и использует для управления биты TR1 и TF1 таймера 1. Таким образом, прерывание от переполнения регистра TH0, будет обозначено флагом TF1.

Режим 3 предназначен для приложений, которым нужен дополнительный 8-битный таймер/счетчик. Когда таймер 0 работает в режиме 3, таймер 1 может быть выключен установкой его в режим 3, или может быть оставлен включенным для использования в качестве генератора тактовых импульсов для последовательного интерфейса, или для любого приложения, которому не требуется прерывание именно от таймера 1.

Настройка таймера на заданную частоту

Задача настройки таймера на заданную частоту во встраиваемых системах обычно связана с организацией системного времени.

В нашем случае под этим подразумевается настройка таймера в режиме «таймер» таким образом, чтобы его переполнения происходили через одинаковые интервалы времени (1 мс, 5 мс, 10 мс и т.д.), так называемые кванты времени.

Расчет необходимой частоты работы таймера может быть произведен по следующей формуле:

$$F = \frac{f_{osc}}{12 \cdot Counts},$$

где F – необходимая частота, f_{osc} – частота микроконтроллера (в стенде SDK-1.1 f_{osc} = 11,0592 МГц), $Counts$ – количество тиков (счетов) таймера для достижения частоты F.

Так как таймеры у нас суммирующие, то регистры таймера (THx, TLx) нужно инициализировать следующих кодом:

$$T_{timer} = Counts_{\max} - Counts,$$

где $Counts_{max}$ – максимальное количество тиков в таймере, которое определяется по разрядности таймера, его режиму работы. Например, если таймер 16-битный (режим 1), то $Counts_{max}$ = 65536.

Если в стенде SDK-1.1 необходимо настроить Таймер 0 на частоту 1000 Гц, то для этого нужно использовать 16-битный таймер (режим 1). По формулам получится, что $Counts$ = 921 (чуть больше) и T_{timer} = 64615 (FC67h). Таким образом, регистры Таймера 0 должны быть инициализированы так: TH0 = FCh, TL0 = 67h. В подразделе 6.4 приведен пример программирования таймера 0 в стенде SDK-1.1.

Использование таймера в качестве измерителя ширины импульсов

Известно, что измерение длительности импульса можно произвести, подсчитав импульсы эталонной частоты. Принцип измерения длительности импульсов иллюстрируется рисунком [51]:

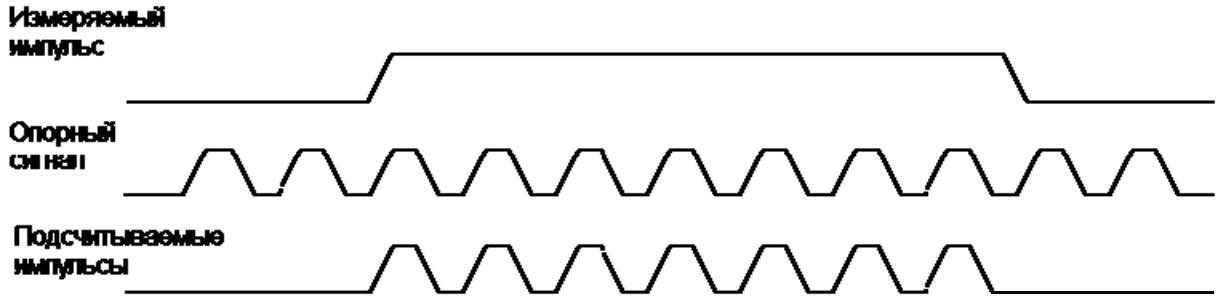


Рисунок 33. Принцип измерения длительности импульсов

Для измерения длительности импульса измеряемый сигнал подаётся на вывод микроконтроллера INTx и в бит управления GATE записывается разрешающий сигнал логической единицы. Таймер/счётчик настраивается в режим таймера записью в бит C/Tx логического нуля. Содержимое таймера обнуляется.

Если теперь на вход микроконтроллера INT0 подать импульс с неизвестной длительностью, то в регистрах TH0 и TL0 будет записана его длительность в микросекундах.

Использование таймера в качестве частотомера

Известно, что измерение частоты можно произвести, подсчитав количество периодов неизвестной частоты за единицу времени. Принцип измерения частоты иллюстрируется рисунком [51]:

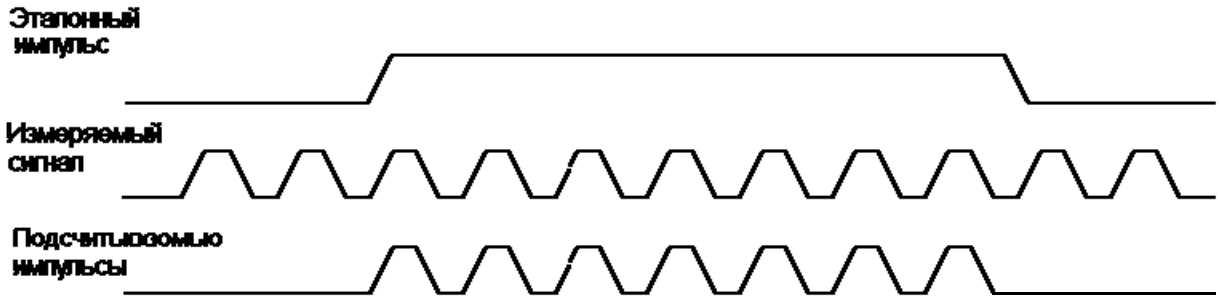


Рисунок 34. Принцип измерения частоты

Для измерения частоты измеряемый сигнал подаётся на вывод микроконтроллера Tx. Таймер/счётчик настраивается в режим счётчика записью в бит C/Tx логической единицы. Содержимое таймера обнуляется. Таймер включается на строго определённый интервал времени. Этот интервал задаётся оставшимся таймером.

Если теперь на вход микроконтроллера T0 подать сигнал с неизвестной частотой, то в регистрах TH0 и TL0 будет записана его частота в килогерцах.

2.2.6.2 Модули таймеров-счетчиков со схемами входного захвата, выходного сравнения и выработки сигналов с ШИМ

Модули Capture/Compare/PWM (CCP) являются развитием структуры таймеров-счетчиков и выполняют схожие функции, однако требуют меньшей программной поддержки, более гибки в настройке на различные задачи, позволяют достигнуть более высокого быстродействия. Наибольшую эффективность они обеспечивают при работе с внешними периодическими или непериодическими сигналами при решении следующих задач:

- Фиксация времени (момента) внешнего события (фронта);
- Определение частоты и длительности импульсов внешнего сигнала, фазового сдвига нескольких сигналов;
- Формирование одиночных импульсов с программируемой длительностью.
- Формирование на одном или нескольких выводах периодических последовательностей импульсов и программируемой частотой, длительностью, фазовым сдвигом (в случае нескольких выходных сигналов);
- Формирование сигналов с широтно-импульсной модуляцией (ШИМ, PWM). При ШИМ частота сигнала остается постоянной, а длительность положительного и отрицательного импульсов программируется. Основная характеристика сигнала с ШИМ является скважность: отношение периода к длительности положительного импульса. Для меандра скважность равна 2. Модуль ШИМ с подключенной к его выходу интегрирующей цепочкой образует простейший ЦАП. Такое использование модулей ШИМ является основным во встраиваемых системах.

Все перечисленные функции выполняются модулями CCP автономно, а вмешательство программиста требуется только на этапе настройки режимов модуля.

Схема выходного сравнения (Output Compare)

Многоразрядный цифровой компаратор непрерывно сравнивает изменяющийся во времени код таймера-счетчика с кодом, который записан в регистре сравнения. В момент равенства этих кодов устанавливается флаг OCF (Output Compare Flag) и изменяется сигнал на выводе OCO (Output Compare Output). Возможны три варианта изменения сигнала, которые могут быть настроены программно: установка «1», установка «0», инвертирование сигнала на выводе OCO (OCO <= #OCO). По установке флага OCF может быть сброшен

(обнулен) или перезагружен определенным значением регистр-счетчик. Кроме того, по установке флага OCF может быть выработан запрос прерывания, если данное прерывание разрешено. Запрос прерывания может вырабатываться и при переполнении таймера-счетчика.

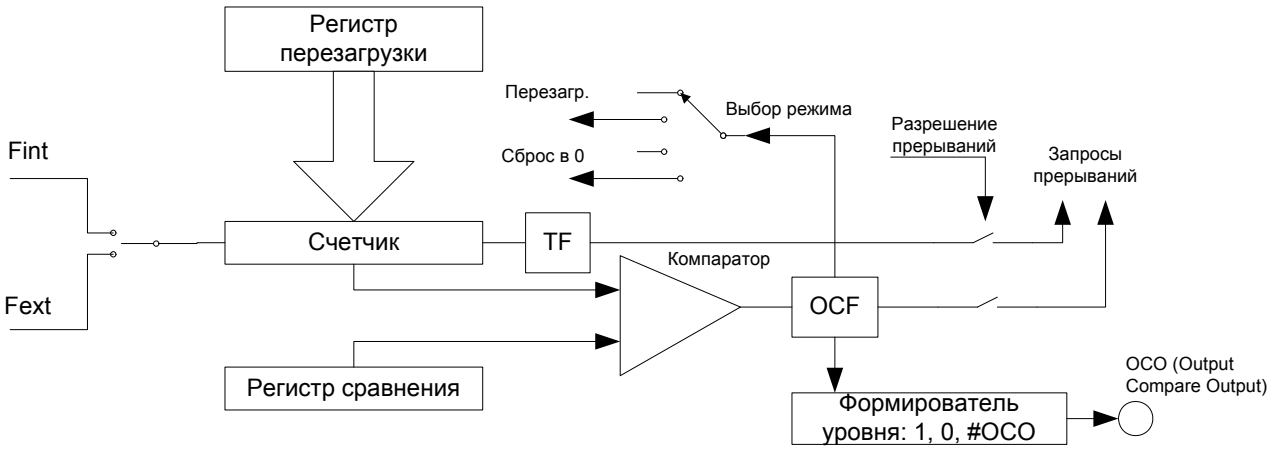


Рисунок 35. Модуль выходного сравнения (Output Compare)

Рассмотрим примеры типовых применений модуля CCP в режиме выходного сравнения:

1. Формирование сигнала с определенной частотой: формирователь уровня настраивают на режим инверсии OCO, управление таймером-счетчиком в режим сброса по флагу OCF, в регистр сравнения – значение, равное полупериоду формируемой частоты. По каждому событию сравнения раз в полупериод порт OCO инвертируется и формируется передний или задний фронт сигнала.
2. Формирование одиночного импульса определенной длительности: формирователь уровня настраивают на режим установки OCO в «0», в регистр сравнения – длительность импульса, таймер обнуляем и одновременно устанавливаем порт OCO в «1» (передний фронт). По событию сравнения порт обнуляется (задний фронт).
3. Ожидание определенного числа импульсов на счетном входе (сигнал Fext) таймера-счетчика: таймер настраиваем в режим счетчика, обнуляем, в регистр сравнения записываем требуемое число импульсов, разрешаем прерывание по событию сравнения (по флагу OCF). После прохождения заданного числа импульсов будет выработан запрос прерывания.
4. Делитель входной частоты на заданное число N, кратное двум: таймер-счетчик переключаем в режим счетчика, устанавливаем обнуление счетчика по флагу OCF, формирователь уровня настраивают на режим инверсии OCO, в регистр сравнения записываем значение N/2.

Схема входного захвата (Input Capture)

Функцию входного захвата поддерживают микроконтроллеры семейств (Atmel), 8051GB(Intel), AVR(Atmel), PIC16(Microchip), ST7, ST9 (SGS-T), HC08, HC11 (Motorola) и многие другие.

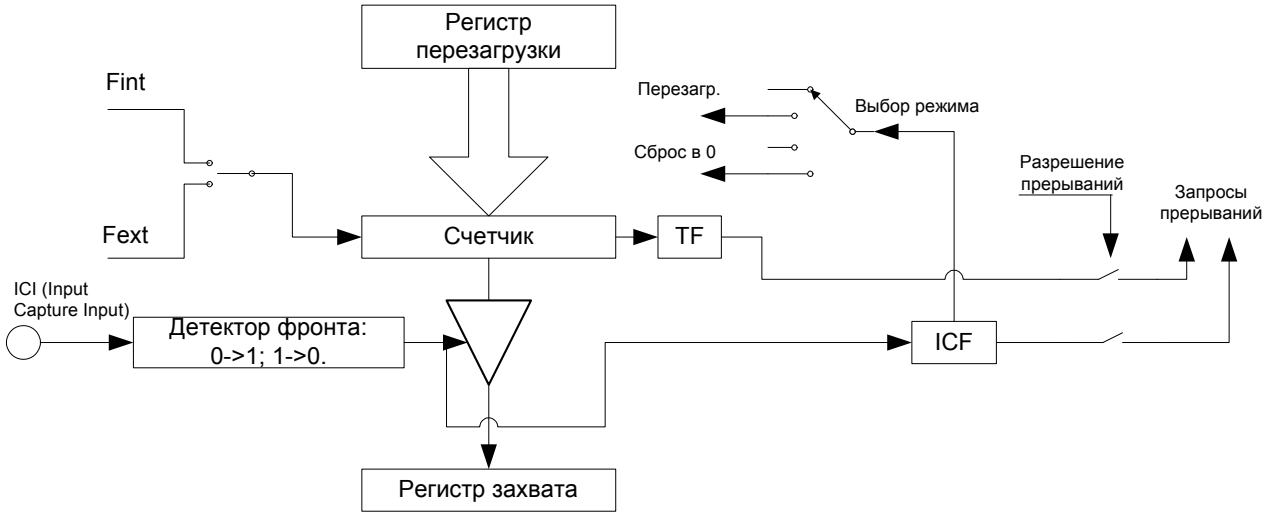


Рисунок 36. Модуль входного захвата (Input Capture)

Данная схема предназначена для фиксации времени возникновения внешнего события: когда на внешнем выводе ICI происходит событие (перепад), определяемый настройкой схемы «детектора фронта», то текущее значение регистра-счетчика переписывается в регистр захвата, откуда может быть прочитано программно. Во многих реализациях захват может быть программно-управляемым – по команде обращения к специальному регистру.

Тактирование регистра-счетчика чаще выбирается от сигнала внутренней синхронизации процессора Fint, то есть счетная часть модуля Input Capture настроена на режим подсчета времени – таймера. Но так же можно использовать и внешнее тактирование. По событию захвата устанавливается флаг ICF, может вырабатываться запрос прерывания. Кроме этого может быть перезагружен «0» или определенным значением регистр-счетчик.

С помощью схемы входного захвата удобно:

1. Определять период/частоту сигнала на входе ICI;
2. Фиксация относительного времени возникновения различных событий.

Схема выработки сигнала с ШИМ

Данная схема является модифицированным вариантом схемы выходного сравнения (Output Compare). Разница в том, что выходом управляет как компаратор, так и схема фиксации переполнения регистра-счетчика. Передний фронт сигнала с ШИМ (0→1) формируется по событию сравнения (когда регистр-счетчик равен регистру сравнения). Задний фронт (1→0) – по переполнению регистра-счетчика.

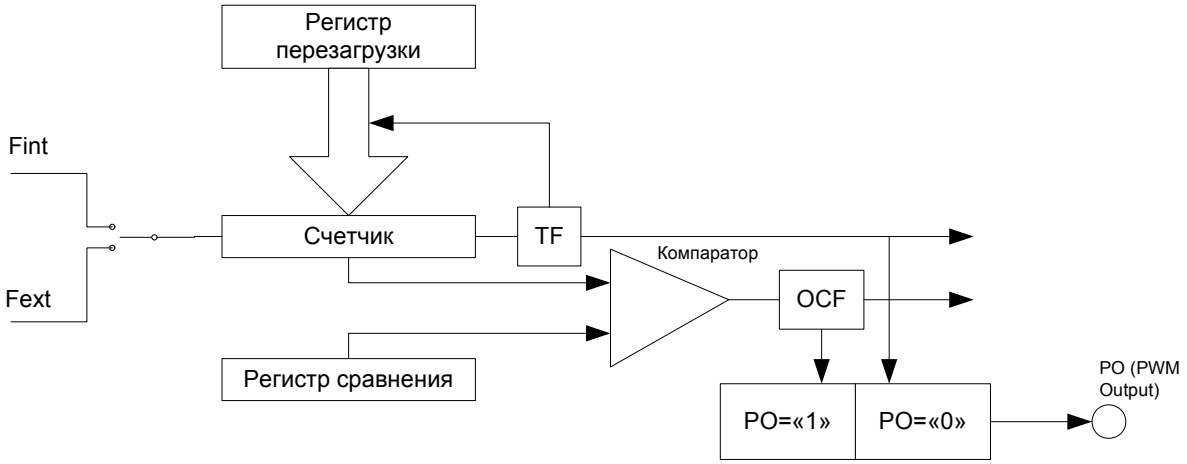


Рисунок 37. Модуль генератора сигнала ШИМ

Период сигнала с ШИМ равен частоте переполнения таймера и задается содержимым регистра перезагрузки. Длительность положительного импульса в периоде определяется как разница (максимального значения регистра-счетчика +1) и содержимого регистра сравнения.

В различных процессорах могут использоваться схемы генераторов ШИМ немного отличающиеся от данной.

Процессоры событий

Под управлением единого счетчика могут быть объединены несколько каналов входного захвата/ выходного сравнения/ формирования сигналов ШИМ. Каждый из каналов может быть индивидуально настроен на один из перечисленных режимов. Такие сложные блоки называют *процессорами событий*, а также: массивом программируемых счетчиков – PCA (Programmable Counter Array) (Intel), блоком CAPCOM (Infineon), блоком TIM8 (Motorola).

Рисунок 38. Блок процессора событий

Процессоры событий позволяют формировать взаимно синхронизированные выходные сигналы: с фиксированным сдвигом фаз или считывать временные сдвиги между событиями, как частный случай – сдвиг фазы.

2.2.7 Аналого-цифровой преобразователь

Модуль аналого-цифрового преобразования (АЦП, Analog-to-digital converter, ADC) предназначен для ввода в процессор аналоговых сигналов с датчиков физических величин и преобразования значения напряжения этих сигналов в двоичный код с целью дальнейшей программной обработки [25]. Простейшим одноразрядным двоичным АЦП является компаратор.

Характеристики:

- Разрешение АЦП – минимальное изменение величины аналогового сигнала, которое может быть преобразовано данным АЦП. Обычно измеряется в вольтах, поскольку для большинства АЦП входным сигналом является электрическое напряжение.
- Разрядность АЦП характеризует количество дискретных значений, которые преобразователь может выдать на выходе.
- Частота дискретизации.
- Точность.
- Скорость преобразования.

Аналоговый сигнал является непрерывной функцией времени, в АЦП он преобразуется в последовательность цифровых значений. Следовательно, необходимо определить частоту выборки цифровых значений из аналогового сигнала. Частота, с которой производятся цифровые значения, получила название частоты дискретизации АЦП.

Непрерывно меняющийся сигнал с ограниченной спектральной полосой подвергается оцифровке (то есть значения сигнала измеряются через интервал времени T — период дискретизации) и исходный сигнал может быть точно восстановлен из дискретных во времени значений путём интерполяции. Точность восстановления ограничена ошибкой квантования. Однако в соответствии с теоремой Котельникова-Шеннона точное восстановление возможно только, если частота дискретизации выше, чем удвоенная максимальная частота в спектре сигнала.

Поскольку реальные АЦП не могут произвести аналого-цифровое преобразование мгновенно, входное аналоговое значение должно удерживаться постоянным, по крайней мере, от начала до конца процесса преобразования (этот интервал времени называют время преобразования). Эта задача решается путём использования специальной схемы на входе АЦП – устройства выборки-хранения – УВХ. УВХ, как правило, хранит входное напряжение в конденсаторе, который соединён со входом через аналоговый ключ: при замыкании ключа происходит выборка входного сигнала (конденсатор

заряжается до входного напряжения), при размыкании – хранение. Многие АЦП, выполненные в виде интегральных микросхем, содержат встроенное УВХ.

Полученное в результате преобразования значение записывается в регистр данных (РД). АЦП, интегрированные на кристалл процессора, обычно строят по схеме последовательного приближения. Время преобразования обычно составляет несколько десятков микросекунд, в зависимости от частоты тактирования АЦП. Завершение процесса преобразования отмечается установкой флага Fацп и (если разрешено) вырабатывается запрос прерывания. В современных управляющих процессорах и микроконтроллерах наиболее распространены АЦП с разрядностью 8, 10, реже 12 и совсем редко 14 и 16 бит.

Аналоговый коммутатор выбирает один из возможных аналоговых входов (выводов) и подключает его к входу внутреннего АЦП для преобразования. При последовательной выборке каналов создается имитация многоканального АЦП. Применение действительно многоканальных АЦП резко повышает энергопотребление и стоимость процессора и обычно не используется (Если требуется несколько каналов и высокая скорость преобразования, то используют микросхему внешнего АЦП).

Код выбора канала может формироваться программно, то есть программист «вручную» переключается между каналами, или аппаратно (автоматически), последовательно перебирая каналы (режим сканирования).

Для большего удобства использования модуля АЦП в режиме сканирования могут быть реализованы несколько регистров данных (Fujitsu MB90, Intel 8051GB), по одному на канал. Программисту будет достаточно считывать данные из регистра, соответствующего требуемому каналу. При этом код выбора канала параллельно подается на адресные входы блока регистров данных.

Источник опорного напряжения Vref и коммутатор Vref

Опорное напряжение Vref определяет диапазон значений напряжения на аналоговых входах и разрешающую способность АЦП, равную $Vref/2^n$, где n – разрядность АЦП. Если значение напряжение на входе не велико, то точность преобразования может быть увеличена путем уменьшения Vref. Диапазон допустимых значений Vref обычно находится в рамках значения напряжения питания процессора.

Могут быть использованы опорные источники следующего типа:

1. Внешний, подключаемые через специальные выводы микросхемы;
2. Внутренний фиксированный или программируемый (с помощью встроенного ЦАП).

Подключение к АЦП внешнего или внутреннего источников выполняется с помощью коммутатора Vref.

Коммутатор сигнала запуска АЦП позволяет выбрать способ запуска процесса преобразования, а также определяет один из возможных режимов работы АЦП:

1. Периодического преобразования. В этом режиме АЦП запускается периодическим сигналом от основного тактового генератора или встроенного таймера.
2. Если сигнал запуска подать на двоичный счетчик, выходами подключенный к управляющим входам аналогового коммутатора и адресным линиям блока регистров данных, то таким образом легко реализовать режим последовательного сканирования каналов.
3. Внешнего запуска. Запуск осуществляется внешним сигналом, что позволяет четко определить момент считывания значения аналогового напряжения со входа.
4. Программно управляемого запуска, по установке специального бита.

Блок управления модулем АЦП конфигурирует и синхронизирует функционирование других (вышеперечисленных) блоков, управляется программно, через регистры специального назначения.

Аналоговый компаратор

Аналоговый компаратор используется для сравнения напряжения двух внешних аналоговых сигналов или для сравнения напряжения внешнего аналогового сигнала с образцовым напряжением, вырабатываемым внутри процессора. Могут быть запрограммированы различные уровни образцового напряжения. Результат сравнения кодируется битом в регистре специального назначения, например, “1” – вход А больше или равно чем В, “0” – вход А меньше чем В. В случае изменения соотношения изменяется значение бита, а также может быть установлен флаг и выработан запрос прерывания.

Структура блока аналогового компаратора приведена ниже.

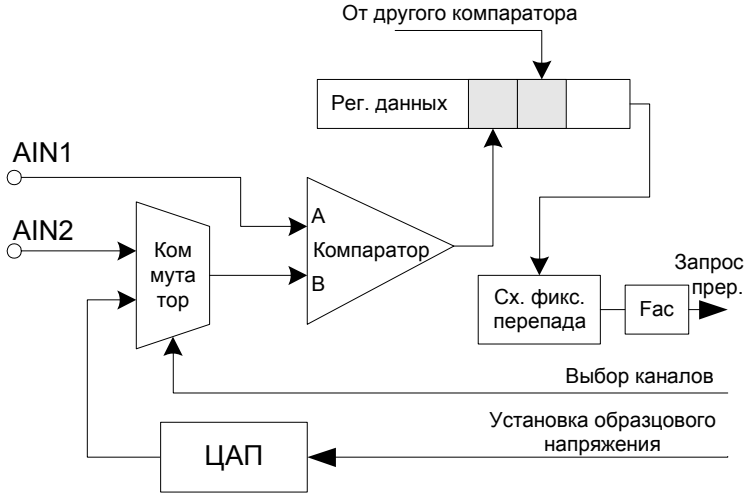


Рисунок 39. Модуль аналогового компаратора

Аналоговый коммутатор входов выбирает аналоговые сигналы для сравнения. Один сигнал берется с внешнего входа AIN1, в качестве второго берется или сигнал с внешнего входа AIN2, или образцовое внутреннее напряжение, которое вырабатывается с помощью ЦАП.

ЦАП – программируемый генератор образцового напряжения.

Регистр данных – программно доступный регистр, в битах которого сохраняются результаты сравнения одного или нескольких компараторов.

Схема фиксации перепада определяет изменение одного из бит в регистре данных (выхода одного из компараторов) и вырабатывает по этому событию запрос прерывания.

Пример использования аналогового компаратора:

- Контроль превышения допустимых значений температуры, давления, тока, напряжения и других физических величин. Физическая величина преобразуется в напряжение с помощью датчика и контролируется с помощью аналогового компаратора. Порог сравнения устанавливается встроенным генератором образцового напряжения.
- Обнаружение (формирование) фронтов внешних сигналов.
- Встроенные схемы контроля напряжения питания системы.

2.2.7.1 Классификация АЦП

Рисунок 40. Классификация АЦП по методам преобразования

В настоящее время известно большое число методов преобразования напряжение-код [25]. Эти методы существенно отличаются друг от друга потенциальной точностью, скоростью преобразования и сложностью аппаратной реализации.

В основу классификации АЦП положен признак, указывающий на то, как во времени разворачивается процесс преобразования аналоговой величины в цифровую. В основе преобразования выборочных значений сигнала в цифровые эквиваленты лежат операции квантования и кодирования. Они могут осуществляться с помощью либо последовательной, либо параллельной, либо последовательно-параллельной процедур приближения цифрового эквивалента к преобразуемой величине.

Параллельные АЦП

АЦП этого типа осуществляют квантование сигнала одновременно с помощью набора компараторов, включенных параллельно источнику входного сигнала. На рисунке показана реализация параллельного метода АЦ-преобразования для 3-разрядного числа. С помощью трех двоичных разрядов можно представить восемь различных чисел, включая нуль. Необходимо, следовательно, семь компараторов. Семь соответствующих эквидистантных опорных напряжений образуются с помощью резистивного делителя.

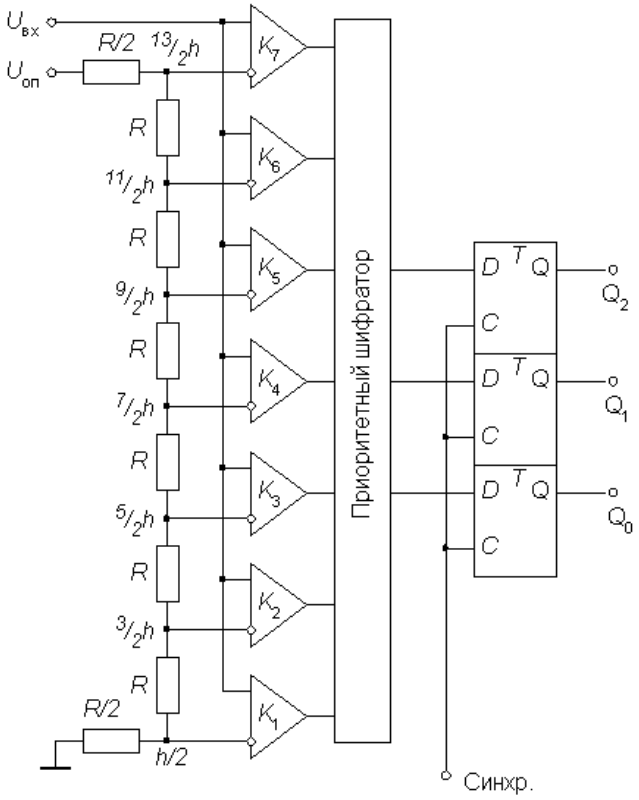


Рисунок 41. Схема параллельного АЦП

Если приложенное входное напряжение не выходит за пределы диапазона от 5/2h, до 7/2h, где h=Uоп/7 – квант входного напряжения, соответствующий единице младшего разряда АЦП, то компараторы с 1-го по 3-й устанавливаются

в состояние 1, а компараторы с 4-го по 7-й – в состояние 0. Преобразование этой группы кодов в трехзначное двоичное число выполняет логическое устройство, называемое приоритетным шифратором.

АЦП последовательного приближения

АЦП последовательного приближения (successive approximation architecture, SAR) или АЦП с поразрядным уравновешиванием содержит компаратор, вспомогательный ЦАП и регистр последовательного приближения. АЦП преобразует аналоговый сигнал в цифровой за N шагов, где N — разрядность АЦП.

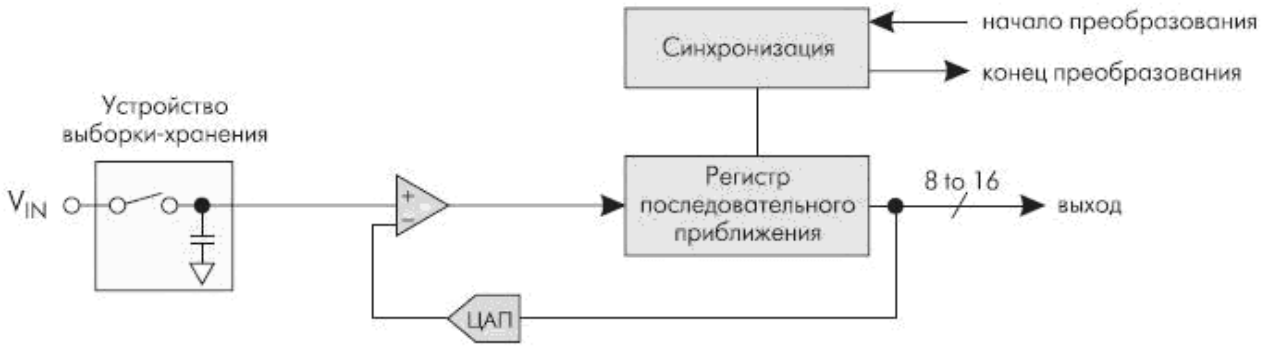


Рисунок 42. АЦП последовательного приближения

На каждом шаге определяется по одному биту искомого цифрового значения, начиная от старшего значащего разряда (СЗР) и заканчивая младшим значащим разрядом (МЗР). Последовательность действий по определению очередного бита заключается в следующем. На вспомогательном ЦАП выставляется аналоговое значение, образованное из битов, уже определённых на предыдущих шагах; бит, который должен быть определён на этом шаге, выставляется в 1, более младшие биты установлены в 0. Полученное на вспомогательном ЦАП значение сравнивается с входным аналоговым значением. Если значение входного сигнала больше значения на вспомогательном ЦАП, то определяемый бит получает значение 1, в противном случае 0. Таким образом, определение итогового цифрового значения напоминает двоичный поиск. АЦП этого типа обладают одновременно высокой скоростью и хорошим разрешением. Однако при отсутствии устройства выборки хранения погрешность будет значительно больше (представьте, что после оцифровки самого большого разряда сигнал начал меняться).

Характеристика АЦП последовательного приближения: невысокая скорость преобразования, невысокая цена и низкое энергопотребление.

2.2.8 Цифро-аналоговый преобразователь

Цифро-аналоговый преобразователь (digital-analog converter, DAC) предназначен для преобразования числа, представленного, как правило, в виде двоичного кода, в напряжение или ток, пропорциональные этому числу.

Схемотехника цифро-аналоговых преобразователей весьма разнообразна [25]. На рисунке ниже представлена общая классификация ЦАП по способам преобразования входного кода и схемам формирования выходного сигнала.

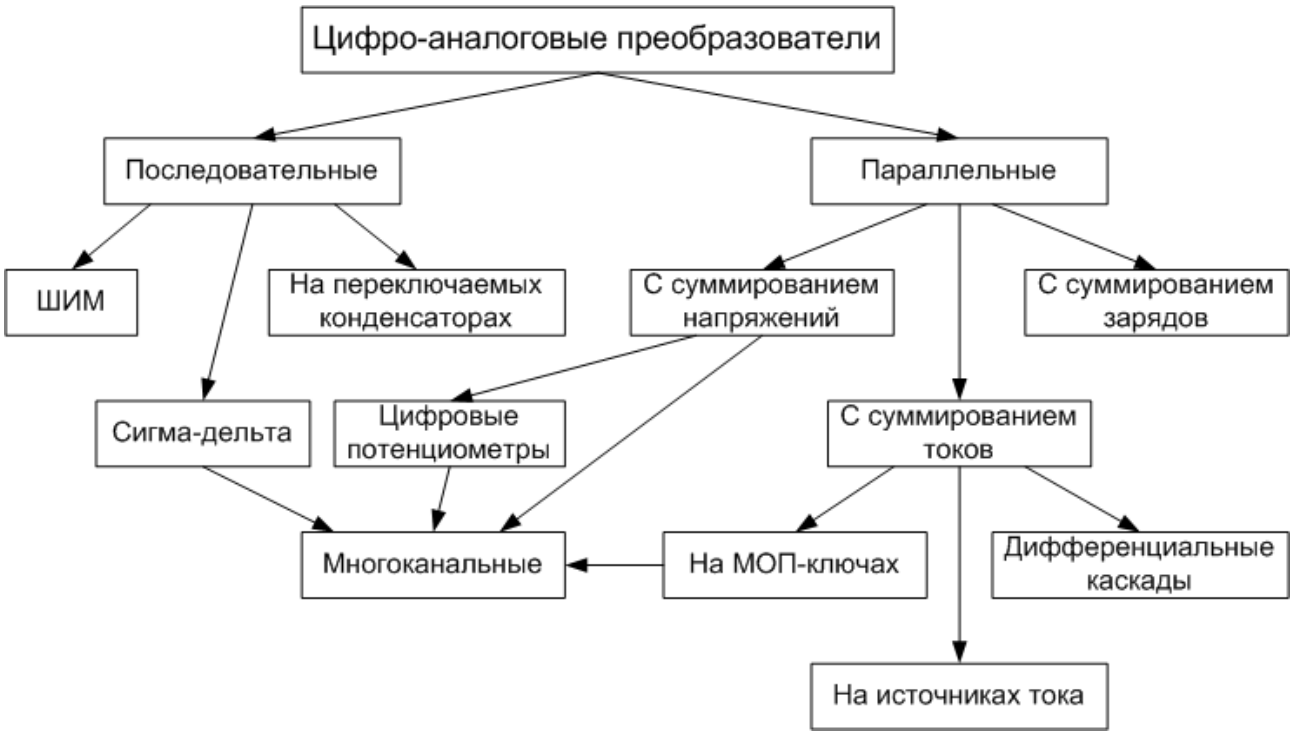


Рисунок 43. Обобщенная классификация ЦАП

Дальнейшую классификацию цифро-аналоговых преобразователей можно провести по ряду специфических признаков, например:

- по роду выходного сигнала: преобразователи с токовым выходом или с выходом по напряжению;
- по типу цифрового интерфейса: с последовательным вводом или с параллельным вводом;
- по числу ЦАП на кристалле: одноканальные и многоканальные;
- по быстродействию: низкого, среднего и высокого быстродействия;
- по разрядности.

Матрица R-2R – самый распространенный метод цифро-аналогового преобразования. Матрица работает по принципу деления входного напряжения на входах. Матрица имеет число входов по числу разрядов регистра данных. На каждый вход через ключ может быть подано опорное напряжение Vref или 0В. Ключи управляются разрядами регистра данных: “1” – на матрицу подается Vref, “0” – подается 0В.

Коммутатор опорного напряжения Vref позволяет выбрать внешний или встроенный источник опорного напряжения.

В регистр данных записывается цифровой код. Регистр данных определяет разрядность ЦАП.

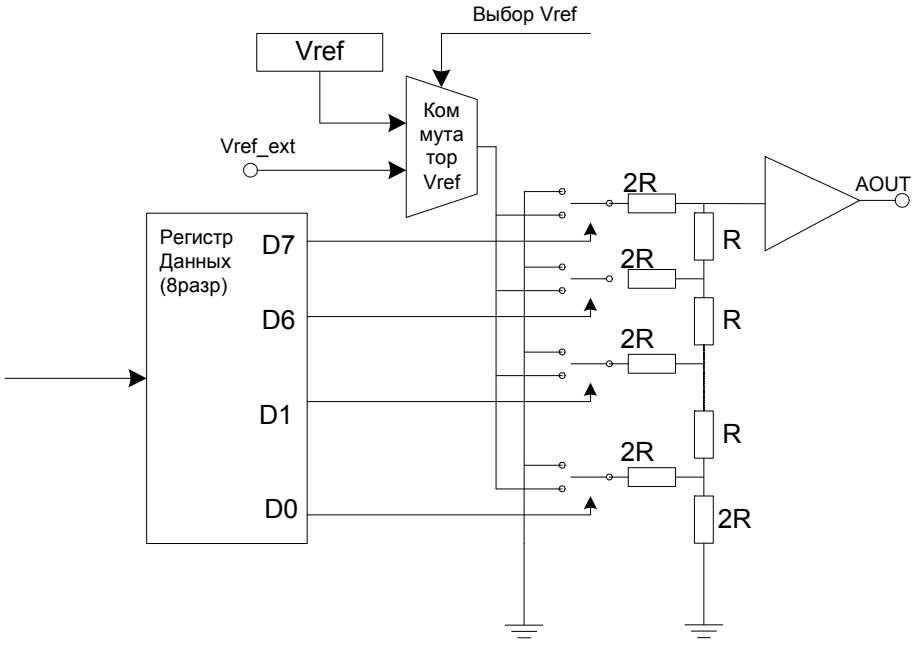


Рисунок 44. Модуль ЦАП с типом преобразования «Матрица R-2R»

На практике ЦАП применяется для управления различными исполнительными устройствами (приводами) и системами: электродвигателями постоянного тока с переменной скоростью вращения, источниками питания с управляемым напряжением, различными индикаторами и т.п. С помощью ЦАП можно синтезировать аналоговые сигналы различной формы, например, синусоидальной.

2.2.9 Контроллеры последовательных интерфейсов

Контроллеры последовательных интерфейсов ориентированы на решение следующих задач:

- Связь встраиваемой микропроцессорной системы с системой управления верхнего уровня: промышленным или офисным компьютером, программируемым контроллером. Наиболее часто для этих целей используют интерфейсы RS-232C, RS-422, USB, IrDA.
- Связь с внешними по отношению к микропроцессору периферийными микросхемами (памяти EEPROM, часов реального времени (RTC) и т.д.), а также с различными датчиками с последовательным цифровым выходом. Для этих целей наиболее часто применяются интерфейсы SPI, I^2C, MicroWire, uLAN и другие.
- Интерфейс связи с локальной сетью в распределенных информационно-управляющих системах. В этой сфере находят применение интерфейсы RS-232C, RS-485, I^2C, uLAN, CAN, Ethernet.
- Внутрисистемное программирование резидентной памяти программ (OTPROM, EPROM, FLASH) или данных (EEPROM) у процессоров для встраиваемых применений. Обычно для этого используется интерфейс

RS-232C (ADuC (Analog Devices), MB90Fxxx (Fujitsu), MSP430 (Texas Instruments)) или SPI (AVR(Atmel)).

В настоящее время встроенные контроллеры последовательных интерфейсов имеются почти у всех встраиваемых процессоров, исключая простейшие 8-16 выводные микросхемы. У большинства процессоров имеются несколько таких модулей одного или различных типов.

Среди контроллеров последовательного обмена стандартом «де-факто» стал модуль универсального синхронно-асинхронного приемопередатчика (Universal Synchronous/Asynchronous Receiver and Transmitter, USART). В названии часто опускают слово «синхронный» и модуль не совсем корректно именуется UART (чисто асинхронные приемопередатчики сейчас встречаются достаточно редко). Характеристики последовательного порта UART не позволяют производить приём и передачу данных за пределы печатной платы. Для связи с другими устройствами, сигнал от UART необходимо пропустить через приёмопередатчик, работающий в одном из стандартов:

- RS-232;
- RS-485;
- RS-422.

Обычно модули UART в асинхронном режиме поддерживают протокол обмена для интерфейса RS-232 (8N1 или 9N1); в синхронном режиме – нестандартные синхронные протоколы, в некоторых случаях – протокол SPI.

Приемопередатчик – преобразователь уровня, как правило, выполненный в интегральном исполнении. Предназначен для преобразования электрических сигналов из уровня ТТЛ в уровень, соответствующий физическому уровню определенного стандарта.

Контроллер UART обычно содержит:

- Источник тактирования (обычно с увеличенной частотой тактирования по сравнению со скоростью обмена, чтобы иметь возможность отслеживать состояние лини передачи данных в середине передачи бита).
- Входные и выходные сдвиговые регистры.
- Регистры управления приемом/передачей данных; чтением/записью.
- Буферы приема/передачи.
- Параллельная шина данных для буферов приема/передачи.
- FIFO буферы памяти (опционально).

Упрощенная структура приемопередатчика типа UART представлена на рисунке.

Генератор скорости обмена представляет собой делитель внутренней тактовой частоты процессора Fint с плавно или пошагово (дискретно) программируемым коэффициентом деления. При «плавном» программировании можно настраивать требуемую скорость вне зависимости (в

определенных пределах) от частоты Fint. Для этого используется стандартный или специально выделенный таймер-счетчик в режиме автоперезагрузки. В случае «фиксированных» коэффициентов деления для поддержания стандартного ряда скоростей необходимо выбирать определенную частоту тактирования процессора.

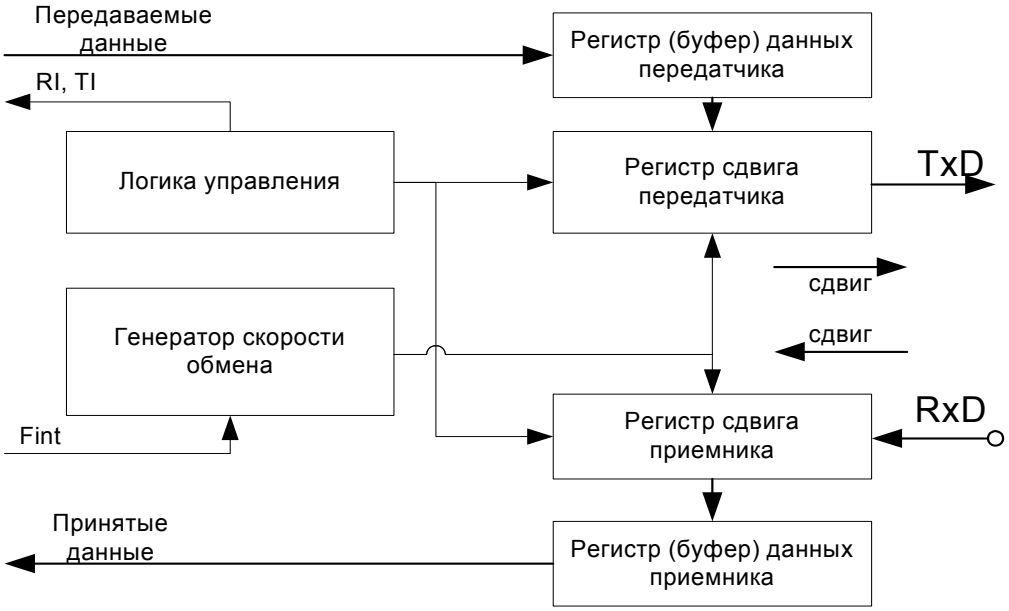


Рисунок 45. Модуль UART

С выхода генератора скорости сигнал синхронизации поступает на вход тактирования приемного и передающего сдвиговых регистров, которые осуществляют последовательную выдачу/прием бит данных с заданной скоростью. Полностью принятый байт попадает в регистр – буфер данных приемника. Байт для передачи помещается в сдвиговый регистр из буфера передатчика.

Процессы приема и передачи в асинхронном режиме UART происходят независимо. Таким образом, поддерживается дуплексный режим обмена. Однако требуется, чтобы приемник и передатчик были настроены на одну скорость.

Более простым является функционирование в синхронном режиме. Здесь каждый принимаемый/передаваемый бит стробируется специальным сигналом и нет необходимости точно согласовывать скорость приемника и передатчика.

Ошибки UART:

- Overrun Error (ошибка из-за повышенной скорости передачи, переполнение буфера приема)

Эта ошибка случается, когда приемник UART не успевает обрабатывать приходящие из канала символы, т. е. буфер переполняется.

- Framing Error (ошибка кадрирования)

Эта ошибка случается, когда фиксируется некорректное состояние линии данных в момент передачи старт- или стоп-бита. Например, после передачи 8

бит данных приемник ожидает перехода линии в стоп-состояние, но этого не происходит.

- Break Condition (сигнал прерывания передачи, разрыва связи)

Этот сигнал информирует о том, что входная линия данных находилась в неизменном нулевом состоянии в течение времени, больше передачи одного символа. В буфере приема нулевой байт. Некоторые устройства используют такую последовательность, чтобы сообщить передатчику, например, о переходе на другую скорость обмена данными.

Пример организации и работы контроллера последовательного канала микроконтроллера ADuC812 (Intel MCS-51) рассмотрен в следующем подразделе этой главы. В разделе 6.3 приведен пример программирования последовательного канала в учебном стенде SDK-1.1.

Еще более упрощается функционирование в режиме SPI: приемник и передатчик работают синхронно: приему одного бита соответствует передача одного бита, начало передачи байта совпадает с началом приема, за сеанс обмена происходит прием одного байта и передача одного байта.

В большинстве случаев приемопередатчики работают с входными и выходными сигналами уровней TTL. Формирование физических сигналов с уровнями напряжения и тока, соответствующими реализуемому интерфейсу, выполняется с помощью специальных микросхем – трансиверов или адаптеров физического интерфейса. Например: MAX232 (MAXIM) – RS-232C, MAX485 (MAXIM) – RS-422/485, PCA82C251 (Philips) – CAN.

Кроме рассмотренных приемопередатчиков USART во встраиваемых процессорах широко используются другие интерфейсы, например, USB, CAN.

2.2.9.1 Контроллер последовательного интерфейса в микроконтроллере с ядром Intel MCS-51

Последовательный порт в микроконтроллерах MCS-51 позволяет осуществлять последовательный дуплексный ввод-вывод в синхронном и асинхронном режимах с разными скоростями обмена. Помимо обычного ввода-вывода, в нем предусмотрена аппаратная поддержка взаимодействия нескольких микроконтроллеров.

Схематически контроллер последовательного порта представлен ниже на рисунке. Как видно из рисунка, регистр SBUF, доступный пользователю для чтения и записи, есть на самом деле два регистра, в один из которых можно только записывать, а из другого – читать. Контроллер позволяет дуплексный обмен данными, т.е. одновременно может передавать и принимать информацию по линиям TxD (P3.1) и RxD (P3.0) соответственно.

Передача инициируется записью в SBUF байта данных. Этот байт переписывается в сдвиговый регистр, из которого пересылается бит за битом.

Как только байт будет переслан целиком, устанавливается флаг TI, сигнализирующий о том, что контроллер готов к передаче очередного байта.

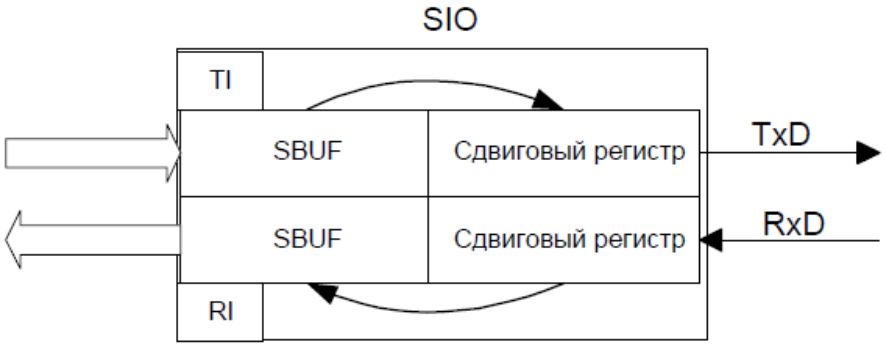


Рисунок 46. Схема контроллера UART МК ADuC812 (Intel MCS-51)

Прием осуществляется в обратном порядке: после начала процесса приема принятые биты данных последовательно сдвигаются в сдвиговом регистре, пока не будет принято их установленное количество, затем содержимое сдвигового регистра переписывается в SBUF и устанавливается флаг RI. После этого возможен прием следующей последовательности бит. Необходимо отметить, что запись принятого байта из сдвигового регистра в SBUF происходит только при условии, что RI=0, поэтому при заборе пользовательской программой очередного байта из SBUF ей необходимо сбрасывать этот флаг, иначе принятый в сдвиговый регистр, но не записанный в SBUF, следующий байт будет безвозвратно утерян.

Контроллер последовательного порта может работать в одном из четырех режимов (один синхронный и три асинхронных), различающихся скоростями обмена (бод) и количеством передаваемых/принимаемых бит:

- Режим 0 (синхронный): данные передаются и принимаются через RxD, TxD является синхронизирующим (выдает импульсы сдвига). Скорость в этом режиме фиксирована (1/12 частоты тактового генератора).
- Режим 1 (8 бит данных, асинхронный, переменная скорость). Передаются 10 бит: старт-бит, 8 бит данных (SBUF) и стоп-бит.
- Режим 2 (9 бит данных, асинхронный, фиксированная скорость). Передаются 11 бит: старт-бит, 8 бит (SBUF), бит TB8/RB8 (посылка/прием соответственно) и 1 стоп-бит. Биты TB8 и RB8 содержатся в регистре SCON (биты 3 и 2 соответственно), первый устанавливается программно, а второй содержит 9-й бит принятой комбинации. С помощью них можно организовать, например, контроль четности или обмен с двумя стоп-битами.
- Режим 3 (9 бит данных, асинхронный, переменная скорость). То же, что и режим 2, только скорость обмена переменная.

В режиме 1 и 3 используются Таймер 0 и/или Таймер 1 для настройки скорости обмена.

Режим контроллера UART, а также другие параметры его работы задаются в регистре специального назначения SCON (98h).

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Таблица 6. Описание регистра специального назначения SCON

Бит	Описание
SM0 SM1	Режим работы UART: SM0 SM1 Выбранный режим работы 0 0 Режим 0 0 1 Режим 1 1 0 Режим 2 1 1 Режим 3
SM2	Включает поддержку взаимодействия нескольких микроконтроллеров в режимах 2 и 3. В Режиме 0 бит SM2 должен быть сброшен (SM2=0). При приеме в Режиме 1 при SM2=1 RI не устанавливается, если не был принят правильный стоп-бит; если же SM2=0 RI установится как только будет принят байт данных. Если SM2=1, то при приеме в режимах 2 и 3 RI не устанавливается в том случае, если принятый 9-й бит (RB8) равен 0.
REN	(Receiver Enable) Если установлен, то разрешен прием данных. Устанавливается и сбрасывается программно.
TB8	(Transmitter Bit 8) 9-й бит посылаемых данных в режимах 2 и 3. Устанавливается и сбрасывается программно.
RB8	(Receiver Bit 8) 9-й бит принимаемых данных в режимах 2 и 3. В режиме 1, если SM2=0, в RB8 записывается принятый стоп-бит (точнее то, что было принято в момент приема стоп-бита). В режиме 0 не используется.
TI	(Transmitter Interrupt) Флаг завершения посылки. Устанавливается аппаратно по завершении посылки 8-го бита данных в режиме 0 или стоп-бита в других режимах. При ES = 1 (бит 4 регистра IEN0 (0A8h)) происходит прерывание №4 (вектор 023h), когда TI устанавливается контроллером в 1. Сбрасывается программно.
RI	(Receiver Interrupt) Флаг завершения приема. Устанавливается аппаратно по завершении приема 8-го бита данных в режиме 0 или стоп-бита в других режимах. При ES = 1 (бит 4 регистра IEN0 (0A8h)) происходит прерывание №4 (вектор 023h), когда RI устанавливается контроллером в 1. Сбрасывается программно.

Как было сказано выше, скорости обмена в разных режимах различаются. В двух из них скорости фиксированы, а в двух других переменные. Рассмотрим подробнее способы задания скорости обмена в каждом из режимов.

- Режим 0. Фиксированная скорость обмена. В этом режиме скорость вычисляется следующим образом:

$baudrate = \frac{Core\_Clk}{12}$, где *Core_Clk* есть внутренняя тактовая частота МК.

- Режим 2. Скорость обмена зависит от значения бита SMOD (старший бит в регистре PCON (087h)). Если SMOD=0 (по умолчанию), то

скорость определяется как 1/64 тактовой частоты. SMOD=1 удваивает это значение. В общем случае:

$$baudrate = \frac{2^{SMOD} \times f_{osc}}{64}$$

- Режимы 1 и 3. В этих режимах порт можно синхронизировать от двух источников: от Таймера 1 или Таймера 2, или обоих (один для передачи, другой для приема байта данных). Источник синхронизации определяется значением бита BD (старший бит регистра ADCON(0D8h)): если BD=1, то используется встроенный генератор импульсов, в противном случае используется таймер 1.

При использовании Таймера 1 скорость обмена определяется временем переполнения таймера и значением бита SMOD:

$$baudrate = \frac{2^{\mathrm{SMOD}} \times \mathrm{TM1OVrate}}{32},$$

где TM1OVrate – время переполнения Таймера 1, зависящее от режима его работы. Таймер 1 может быть установлен как «таймер» или как «счетчик», в любом режиме работы. Прерывание от таймера при этом обычно запрещается. Чаще всего таймер 1 устанавливают как «таймер» в режиме 2 («auto-reload»): для этого старшая тетрада регистра TMOD (089h) должна равняться 0010b. При таком использовании после каждого переполнения регистра таймера TL1 в него загружается значение из регистра TH1. Скорость обмена в этом случае вычисляется по формуле:

$$baudrate = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12 \times (256 - \mathrm{TH1})}$$

В следующей таблице приведены способы задания некоторых стандартных скоростей обмена:

Скорость	**f_{osc}, МГц**	**SMOD**	**Таймер 1 (TH1)**
19200 бод	11.059	1	FD
9600 бод	11.059	0	FD
4800 бод	11.059	0	FA
2400 бод	11.059	0	F4

2.2.10 Подсистема синхронизации

Подсистема синхронизации отвечает за формирование устойчивых сигналов синхронизации внутренних блоков процессора и внешних цепей (блоков) управляющих вычислительных систем, построенных на данном процессоре.

К внутренним блокам относятся:

- Вычислительное ядро;

- Периферийные устройства (таймеры/счетчики, блоки CCP, АЦП, ЦАП, приемопередатчики и др.);
- Схемы рестарта («сброса»).

Для синхронизации внешних схем (периферийных контроллеров, интерфейсных микросхем, блоков программируемой логики и др.) из процессора на специальную ножку выводится сигнал синхронизации.

Обобщенная структура подсистемы синхронизации встраиваемых процессоров приведена на рисунке.

Подсистема синхронизации включает блок генераторов синхроимпульсов (основной и вспомогательный генераторы, схему формирователя выходного сигнала синхронизации) и формирователь внутренних синхросигналов процессора (коммутаторы К1 и К2, делители, схему «сброса»).

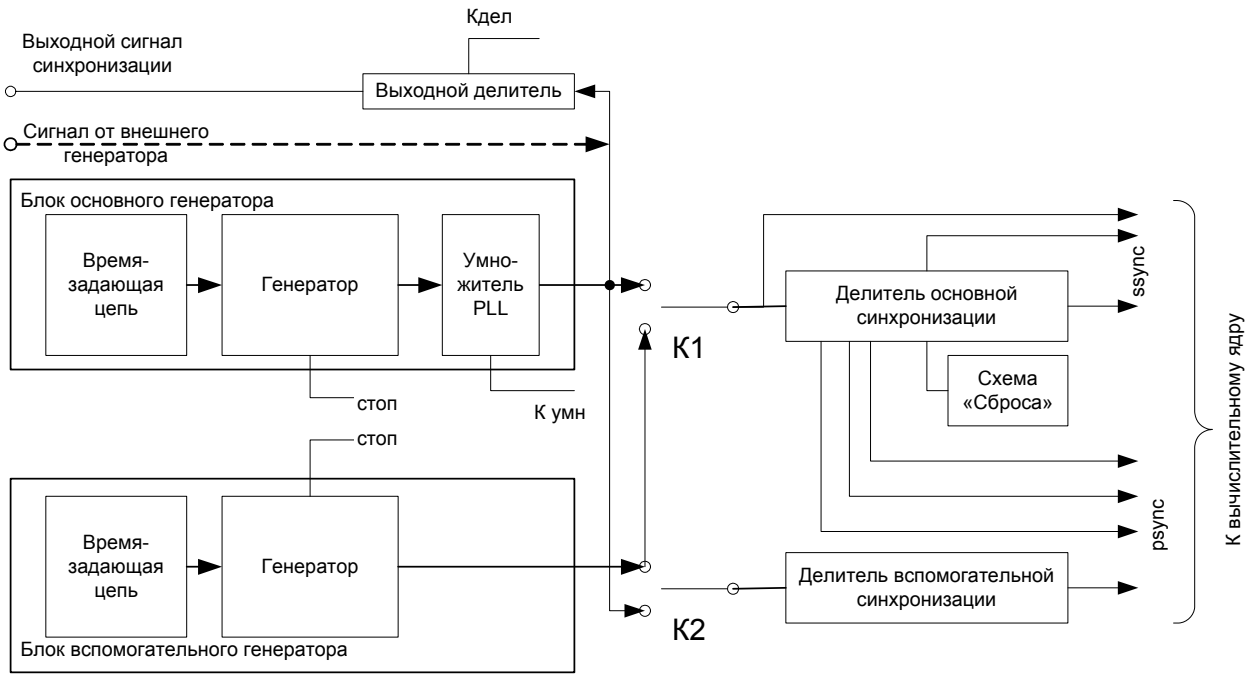


Рисунок 47. Подсистема синхронизации

Блок основного генератора вырабатывает сигналы синхронизации для вычислительного ядра, большинства периферийных устройств, схемы рестарта. Блок включает собственно схему генератора, внешнюю или встроенную времязадающую цепочку для генератора и (не всегда) схему умножителя частоты.

Основная частота синхронизации может изменяться в очень широких пределах – от десятков килогерц до десятков и сотен мегагерц. Невысокая частота (до 1 МГц) используется в системах с пониженным энергопотреблением.

В зависимости от выбранной рабочей частоты, требований точности и стабильности параметров сигнала синхронизации необходимо могут быть использованы *различные типы времязадающих цепочек*, которые в свою очередь требуют перенастройки режимов работы генераторов. Выбор режима

работы генератора и типа времязадающей цепочки осуществляется программированием специальных конфигурационных бит во встроенной памяти программ микропроцессора.

Времязадающие цепочки подключаются к специальным выводам микропроцессора. В некоторых моделях имеются встроенные цепочки. Наиболее часто используются следующие типы времязадающих цепочек:

1. Кварцевый резонатор: частоты от десятков килогерц до десятков мегагерц, высокая стабильность частоты (погрешность – сотые/тысячные доли процента), относительно высокая цена;
2. Пьезокерамический резонатор: частоты от десятков килогерц до единиц мегагерц, средняя стабильность частоты (погрешность – десятые доли процента), невысокая цена;
3. LC-цепь: частоты единицы-сотни килогерц, средняя стабильность частоты (погрешность – десятые доли/единицы процента), невысокая цена;
4. RC-цепь: частоты единицы-сотни килогерц, низкая стабильность частоты (погрешность – единицы процента), низкая цена, часто реализуется как встроенная времязадающая цепочка;

В случаях высокой частоты (свыше 30 МГц) рекомендуется встроенный генератор отключать полностью и подключать внешний. Так же можно поступать, если от одного внешнего генератора синхронизируется несколько схем, включая процессор.

Работа генератора на высокой частоте ведет за собой следующие трудности: сложность «запуска» встроенного генератора (на частотах свыше 30 МГц), специальные требования к трассировке и качеству печатных плат, высокий уровень помех от внешних высокочастотных цепей (например, цепей подключения кварцевого резонатора), невозможность оперативной перестройки частоты. Для избежания этих проблем почти во всех 16/32-разрядных процессорах используются встроенные цифровые умножители частоты с программируемым коэффициентом умножения. Наиболее распространенным умножителем на сегодняшний день является схема синтезатора частоты с фазовой автоподстройкой (PLL).

От основного генератора синхросигнал выводится на ножку микросхемы и может использоваться для тактирования внешних схем. Для снижения частоты выходного сигнала в этой цепи может использоваться управляемый или фиксированный делитель.

Блок вспомогательного генератора обеспечивает тактирование части периферийных устройств, обычно таймеров-счетчиков (PICmicro, ATmega, Fujitsu MB90), а в некоторых режимах может принимать на себя функции основного генератора (синхронизацию ядра, периферии, схем «сброса»). Вспомогательный генератор обычно работает на частотах до 1 МГц. В случае использования его как базы часов реального времени – на частоте 32768 Гц.

По структуре вспомогательный генератор аналогичен основному генератору, но почти никогда не используется умножитель частоты.

Ядром формирователя внутренних сигналов являются делители частоты основного и вспомогательного генераторов. С их выходов берутся сигналы тактирования ядра и схемы сброса ssync и сигналы синхронизации периферийных модулей psync. Обычно все внутренние сигналы (ssync и psync) получаются делением частоты генераторов на фиксированный коэффициент, но в некоторых процессорах коэффициенты деления могут программироваться, например, если необходимо снизить частоту тактирования ядра в режимах пониженного энергопотребления.

Коммутаторы синхросигналов К1 и К2 используются для выбора источника тактирования внутренних схем процессора: основного или вспомогательного генератора. В обычном режиме большинство подсистем синхронизируется от основного генератора, а вспомогательный используется как временная база таймеров, часов реального времени или сторожевого таймера. В случае нестабильности работы основного генератора или при необходимости перейти на более низкие частоты функционирования, например, в режимах энергосбережения, можно подключить вход делителя основного генератора на выход вспомогательного генератора.

2.2.11 Механизмы начальной инициализации встроенной памяти

Механизмы начальной инициализации (начальной загрузки) обеспечивают запись программного кода, данных или конфигурационных параметров во встроенную энергонезависимую память процессора или однокристальной микроЭВМ. Процесс начальной инициализации предполагает работу с «голой» аппаратурой, т.е. без помощи какой-либо инструментальной программы (загрузчика), исполняющейся в рабочем режиме процессора.

В качестве записываемого программного кода выступает или более высокоуровневый загрузчик или непосредственно прикладная программа.

Данные – обычно начальные значения рабочих параметров (уставок).

Конфигурационные параметры настраивают режимы работы аппаратуры процессора. К ним могут относиться:

- тип генератора (кварцевый, на пьезокерамическом резонаторе, LC или RC);
- используемые подсистемы сброса при сбое электропитания, автоматического сброса при включении питания (Power On Reset);
- использование сторожевого таймера (Watch Dog Timer);
- флаги защиты внутренней памяти от несанкционированного копирования;
- использование и разрядность шины внешней памяти;
- адрес старта программы (вектор сброса);

- и т.д.

Классификация механизмов начальной инициализации представлена на рисунке ниже.

Встроенная программа загрузчика (Bootsrap loader) – специальная программа, записанная при производстве процессора в специальный блок встроенной памяти программ ПЗУ. При выполнении bootstrap loader принимает записываемые программы или данные через последовательный порт (обычно порт UART) и записывает их в память процессора.

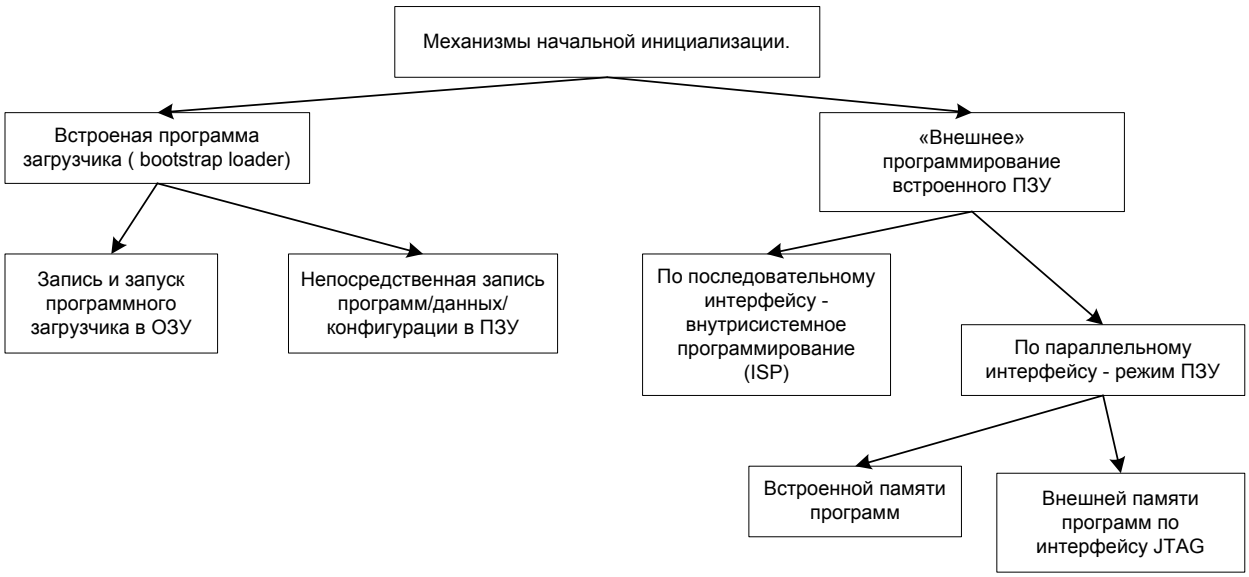


Рисунок 48. Механизмы инициализации встроенной памяти

Возможны несколько вариантов сохранения загруженной программы:

1. Программа загружается в ОЗУ и сразу после этого ей передается управление. Это должен быть загрузчик, в свою очередь принимающий и записывающий во встроенное или внешнее ПЗУ (обычно это – FLASH-память) прикладную программу. После рестарта управление передается прикладной программе.
2. Программа записывается непосредственно во встроенное ПЗУ и начинает исполняться после перезапуска в нормальном режиме. В качестве загруженной программы может выступать загрузчик или целевой код. Такой режим используется, например, в семействах MB90F (Fujitsu), MSP430 (Texas Instruments).

Переход в режим bootstrap loader обычно выполняется подачей специального кода на конфигурационные выводы с одновременным рестартом процессора.

2.2.11.1 Внешнее программирование встроенного ПЗУ

В данном режиме процессор со встроенным ПЗУ (OTP, EEPROM, FLASH) рассматривается как обычная микросхема ПЗУ с последовательным или параллельным интерфейсом. Ядро процессора при этом отключено; некоторые

семейства, например, AVR (Atmel), требуют, чтобы функционировал генератор синхроимпульсов. Переключение процессора в режим внешнего программирования обычно осуществляется подачей на специальный вывод напряжения программирования с уровнем около 12В или подачей специального кода на конфигурационные выводы с одновременным рестартом процессора (аналогично переходу в режим bootsrap loader).

В зависимости от типа интерфейса различают два варианта внешнего программирования:

1. Программирование по параллельному интерфейсу (поддерживается семействами AVR, MCS-51, Z8). Процессор или микроЭВМ программируется вне целевой системы, в специальных программаторах. Порты ввода-вывода и/или сигналы внешней шины используются в качестве линий адреса/данных/управления ПЗУ.

Достоинства:

- Простые алгоритмы программирования;
- Высокая степень защиты от случайного перепрограммирования в системе;

Недостатки:

- Необходимо использовать панельку для микропроцессора или программировать однократно перед монтажом печатной платы системы. Последнее не позволяе модифицировать (обновлять) программное обеспечение системы.

2. Программирование по последовательному интерфейсу (поддерживается семействами AVR, PICmicro). В этом режиме адреса, данные и команды доступа к ПЗУ (записи, чтения, проверки и другие) передаются по специальному или стандартному последовательному интерфейсу.

Достоинства:

- Небольшое количество сигналов (2-5 шт.) позволяет подключать программатор к микросхеме, установленной на плате, и программировать ее, не отключая от схемы. В связи с этим режим последовательного программирования часто называют режимом внутрисистемного программирования (In System Programming, ISP).
- Неограниченность числа различных команд, которые можно передавать в последовательном коде, позволяет значительно увеличить функциональные возможности программатора.

Недостатки:

- Относительно сложный протокол (алгоритм) программирования.

В настоящее время в режиме внутрисистемного программирования микропроцессорных систем с внешним или внутренним ПЗУ начинает широко использоваться последовательный интерфейс JTAG (IEEE-1049). JTAG – интерфейс граничного сканирования, позволяющий устанавливать на ножках микросхемы сигналы с определенным значением и считывать значения сигналов, установленные внешними схемами или внутренними подсистемами процессора. С помощью интерфейса JTAG имитируется диаграмма записи во внешнее ПЗУ или во внутреннее ПЗУ (процессор должен находиться в режиме внешнего параллельного программирования).

2.3 Сетевые интерфейсы встраиваемых систем

Данный раздел посвящен обзору сетевых промышленных проводных и беспроводных, локальных и глобальных интерфейсов, которые применяются в современных встраиваемых системах для организации канала связи между компонентами самой ВВС или с другими информационными системами верхних уровней управления (см. пирамиду автоматизации).

2.3.1 Последовательный интерфейс I²C

В бытовой технике, телекоммуникационном оборудовании и промышленной электронике часто встречаются похожие решения, в, казалось бы, никак не связанных изделиях. Например, практически каждая система включает в себя:

- Некоторый «умный» узел управления, обычно однокристалльная микроЭВМ.
- Узлы общего назначения, такие как буферы ЖКИ, порты ввода-вывода, RAM, EEPROM или преобразователи данных.
- Специфические узлы, такие как схемы цифровой настройки и обработки сигнала для радио- и видео- систем, или генераторы тонального набора для телефонии.

Для того, чтобы использовать эти общие решения к выгоде конструкторов и производителей (технологов), а также для увеличения эффективности аппаратуры и упрощения схемотехнических решений, Philips в 1980 году разработала простую двунаправленную двухпроводную шину для эффективного «межмикросхемного» (inter-IC) управления. Шина так и называется – Inter-Integrated Circuit, или IIC (I²C) шина [7, 18]. В настоящее время ассортимент продукции Philips включает более 150 КМОП и биполярных I²C-совместимых устройств, функционально предназначенных для работы во всех трех вышеперечисленных категориях электронного оборудования. Все I²C-совместимые устройства имеют встроенный интерфейс, который позволяет им связываться друг с другом по шине I²C. Это конструкторское решение разрешает множество проблем сопряжения различных устройств, которые обычно возникают при разработке цифровых систем.

Основной режим работы шины I²C – 100 кбит/с; 10 кбит/с в режиме работы с пониженной скоростью. Заметим, что стандарт допускает тактирование с частотой вплоть до нулевой. Для адресации I²C-устройств используется 7 бит.

После пересмотра стандарта в 1992 году становится возможным подключение ещё большего количества устройств на одну шину (за счёт возможности 10-битной адресации), а также большую скорость до 400 кбит/с в скоростном режиме. Соответственно, доступное количество свободных узлов выросло до 1008. Максимальное допустимое количество микросхем, подсоединенных к одной шине, ограничивается максимальной емкостью шины в 400 пФ.

Версия стандарта 2.0, выпущенная в 1998 году представила высокоскоростной режим работы со скоростью до 3.4 Мбит/с с пониженным энергопотреблением. Последняя версия 2.1 2000 года включила лишь незначительные доработки.

1 октября 2006 года были отменены лицензионные отчисления за использование протокола I²C. Однако, отчисления сохраняются для выделения эксклюзивного подчинённого адреса на шине I²C.

Список возможных применений I²C:

- доступ к модулям памяти (RAM, EEPROM, Flash и др.);
- доступ к низкоскоростным ЦАП/АЦП;
- работа с часами реального времени (RTC);
- регулировка контрастности, насыщенности и цветового баланса мониторов;
- управление интеллектуальными звукоизлучателем (динамиками);
- управление ЖКИ, в том числе в мобильных телефонах;
- чтение информации с датчиков мониторинга и диагностики оборудования, например, термостат центрального процессора или датчик скорости вращения вентилятора охлаждения процессора;
- информационный обмен между микроконтроллерами.

2.3.1.1 Концепция шины I²C

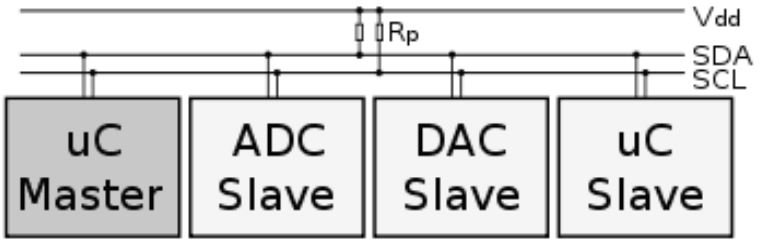


Рисунок 49. Пример соединения устройств по шине I²C: один ведущий – микроконтроллер, три ведомых устройства – АЦП, ЦАП, МК

I²C использует две двунаправленных линии с открытым стоком – последовательная линия данных (SDA, англ. Serial DAta) и последовательная линия тактирования (SCL, англ. Serial CLock), обе нагруженные резисторами. Максимальное напряжение +5В, часто используется +3,3В, однако допускаются и другие напряжения (не менее +2В). Шина I²C поддерживает любую технологию изготовления микросхем (НМОП, КМОП, биполярную).

Каждое устройство распознается по уникальному адресу – будь то микроконтроллер, ЖКИ-буфер, память или интерфейс клавиатуры – и может работать как передатчик или приёмник, в зависимости от назначения устройства. Обычно ЖКИ-буфер – только приёмник, а память может как принимать, так и передавать данные. Кроме того, устройства могут быть классифицированы как ведущие и ведомые при передаче данных. Ведущий – это устройство, которое инициирует передачу данных и вырабатывает сигналы синхронизации. При этом любое адресуемое устройство считается ведомым по отношению к ведущему. Классическая адресация включает 7-битное адресное пространство с 16 зарезервированными адресами. Это означает до 112 свободных адресов для подключения периферии на одну шину [7, 18].

Таблица 7. Термины, используемые в спецификации I²C

Термин (англ.)	**Термин (рус.)**	**Описание**
Transmitter	Передатчик	Устройство, посылающее данные в шину
Receiver	Приемник	Устройство, принимающее с шины
Master	Ведущий	Начинает пересылку данных, вырабатывает синхроимпульсы, заканчивает пересылку данных
Slave	Ведомый	Устройство, адресуемое ведущим
Multi-master	–	Несколько ведущих могут пытаться захватить шину одновременно, без нарушения передаваемой информации
Arbitration	Арбитраж	Процедура, обеспечивающая Multi-master
Synchronization	Синхр.	Процедура синхронизации двух устройств

Возможность подключения более одного микроконтроллера к шине означает, что более чем один ведущий может попытаться начать пересылку в один и тот же момент времени [7, 18]. Для устранения хаоса, который может возникнуть в данном случае, разработана процедура арбитража. Эта процедура основана на том, что все I²C-устройства подключаются к шине по правилу монтажного И.

Генерация синхросигнала – это всегда обязанность ведущего; каждый ведущий генерирует свой собственный сигнал синхронизации при пересылке данных по шине [7, 18]. Сигнал синхронизации может быть изменен, только если он “вытягивается” медленным ведомым устройством (путем удержания линии в низком состоянии), или другим ведущим, если происходит столкновение.

2.3.1.2 Принцип работы шины I²C

Вследствие различных технологий микросхем (КМОП, НМОП, биполярная), которые могут быть подключены к шине, уровни логического нуля (НИЗКИЙ) и логической единицы (ВЫСОКИЙ) не фиксированы и зависят от соответствующего уровня Vdd. Один синхроимпульс генерируется на каждый пересылаемый бит [7, 18].

Данные на линии SDA должны быть стабильными в течение ВЫСОКОГО периода синхроимпульса. ВЫСОКОЕ или НИЗКОЕ состояние линии данных должно меняться, только если линия синхронизации в состоянии НИЗКОЕ.

Рисунок 50. Пересылка бита по шине I²C

Данные по линии SDA передаются байтами, при этом каждый байт должен оканчиваться битом подтверждения. Количество байт, передаваемых за один сеанс связи, не ограничено. Данные передаются, начиная со старшего бита. Если приёмник не может принять еще один целый байт, пока он не выполнит какую-либо другую функцию (например, обслужит внутреннее прерывание), он может удерживать линию SCL в НИЗКОМ состоянии, переводя передатчик в состояние ожидания. Пересылка данных продолжается, когда приёмник будет готов к следующему байту и отпустит линию SCL (опять срабатывает правило монтажного И).

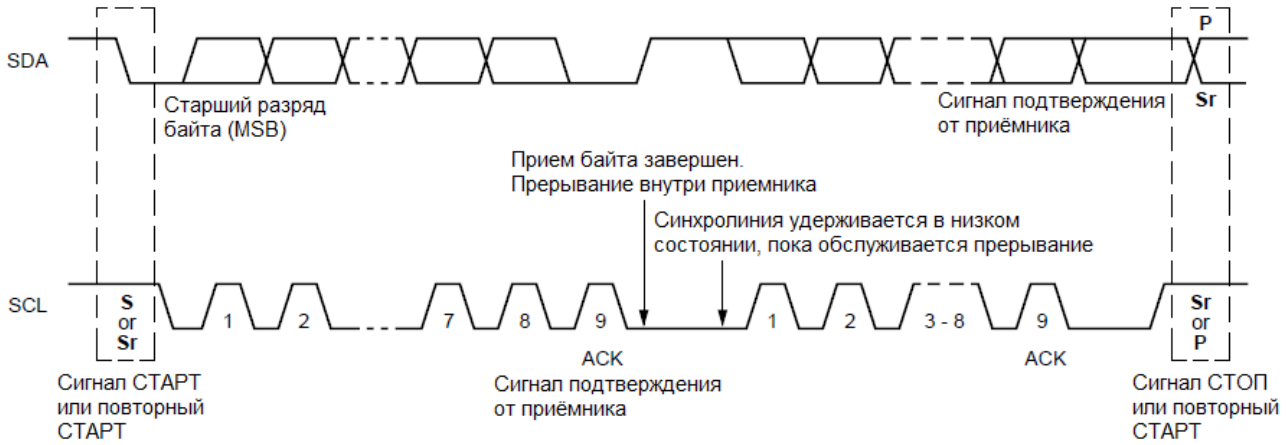


Рисунок 51. Пересылка данных по шине I²C

2.3.1.3 Сигналы СТАРТ и СТОП

Процедура обмена данными по шине I²C начинается с того, что ведущий формирует состояние СТАРТ – ведущий генерирует переход сигнала линии

SDA из ВЫСОКОГО состояния в НИЗКОЕ при ВЫСОКОМ уровне на линии SCL [7, 18]. Этот переход воспринимается всеми устройствами, подключенными к шине как признак начала процедуры обмена. Процедура обмена завершается тем, что ведущий формирует состояние СТОП – переход состояния линии SDA из НИЗКОГО состояния в ВЫСОКОЕ при ВЫСОКОМ состоянии линии SCL. Состояния СТАРТ и СТОП всегда вырабатываются ведущим. Считается, что шина занята после фиксации состояния СТАРТ. Шина считается освободившейся через некоторое время после фиксации состояния СТОП.

Определение сигналов СТАРТ и СТОП устройствами, подключенными к шине достаточно легко, если в них встроены необходимые цепи. Однако микроконтроллеры без таковых цепей должны осуществлять считывание значения линии SDA как минимум дважды за период синхронизации для того, чтобы определить переход состояния.

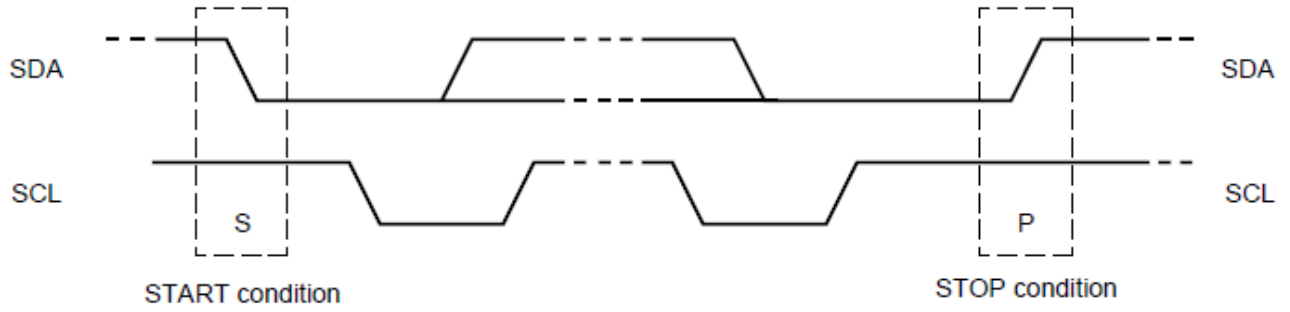


Рисунок 52. СТАРТ и СТОП состояния

2.3.1.4 Подтверждение

Подтверждение при передаче данных обязательно, кроме случаев окончания передачи ведомой стороной. Соответствующий импульс синхронизации генерируется ведущим. Передатчик отпускает (ВЫСОКОЕ) линию SDA в течение синхроимпульса подтверждения. Приёмник должен удерживать линию SDA в течение ВЫСОКОГО состояния синхроимпульса подтверждения в стабильно НИЗКОМ состоянии [7, 18]. Конечно, время установки и удержания также должны быть приняты во внимание (электрические и временные параметры).

Таким образом передача 8 бит данных от передатчика к приемнику завершаются дополнительным циклом (формированием 9-го тактового импульса линии SCL), при котором приемник выставляет НИЗКИЙ уровень сигнала на линии SDA, как признак успешного приема байта.

В том случае, когда ведомый-приёмник не может подтвердить свой адрес (например, когда он выполняет в данный момент какие-либо функции реального времени), линия данных должна быть оставлена в ВЫСОКОМ состоянии. После этого ведущий может выдать сигнал СТОП для прерывания пересылки данных. Если в пересылке участвует ведущий-приёмник, то он должен сообщить об окончании передачи ведомому-передатчику путем не подтверждения последнего байта. Ведомый-передатчик должен освободить

линию данных для того, чтобы позволить ведущему выдать сигнал СТОП или повторить сигнал СТАРТ.

2.3.1.5 Синхронизация

При передаче посылок по шине I²C каждый ведущий генерирует свой синхросигнал на линии SCL [7, 18]. Данные действительны только во время ВЫСОКОГО состояния синхроимпульса.

Синхронизация выполняется с использованием подключения к линии SCL по правилу монтажного И. Это означает, что ведущий не имеет монопольного права на управление переходом линии SCL из НИЗКОГО состояния в ВЫСОКОЕ. В том случае, когда ведомому необходимо дополнительное время на обработку принятого бита, он имеет возможность удерживать линию SCL в низком состоянии до момента готовности к приему следующего бита. Таким образом, линия SCL будет находиться в НИЗКОМ состоянии на протяжении самого длинного НИЗКОГО периода синхросигналов.

Устройства с более коротким НИЗКИМ периодом будут входить в состояние ожидания на время, пока не кончится длинный период. Когда у всех задействованных устройств кончится НИЗКИЙ период синхросигнала, линия SCL перейдет в ВЫСОКОЕ состояние. Все устройства начнут проходить ВЫСОКИЙ период своих синхросигналов. Первое устройство, у которого кончится этот период, снова установит линию SCL в НИЗКОЕ состояние. Таким образом, НИЗКИЙ период синхролинии SCL определяется наидлиннейшим периодом синхронизации из всех задействованных устройств, а ВЫСОКИЙ период определяется самым коротким периодом синхронизации устройств.

Механизм синхронизации может быть использован приемниками как средство управления пересылкой данных на байтовом и битовом уровнях.

На уровне байта, если устройство может принимать байты данных с большой скоростью, но требует определенное время для сохранения принятого байта или подготовки к приему следующего, то оно может удерживать линию SCL в НИЗКОМ состоянии после приема и подтверждения байта, переводя таким образом передатчик в состояние ожидания.

На уровне битов, устройство, такое как микроконтроллер без встроенных аппаратных цепей I²C или с ограниченными цепями, может замедлить частоту синхроимпульсов путем продления их НИЗКОГО периода. Таким образом скорость передачи любого ведущего адаптируется к скорости медленного устройства.

2.3.1.6 Форматы обмена данными по шине I²C (7-битный адрес)

После сигнала СТАРТ посылается адрес ведомого. После 7 бит адреса следует бит направления данных (R/W), «ноль» означает передачу (запись), а «единица» – прием (чтение) [7, 18]. Пересылка данных всегда заканчивается сигналом СТОП, генерируемым ведущим. Однако, если ведущий желает

оставаться на шине дальше, он должен выдать повторный сигнал СТАРТ и затем адрес следующего устройства.

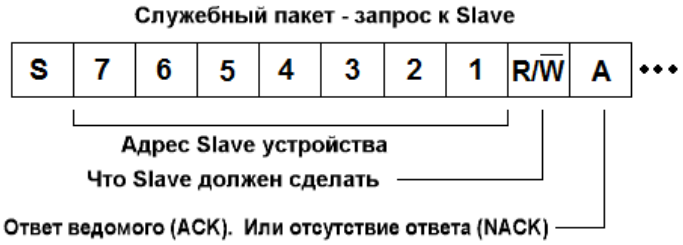


Рисунок 53. Первый байт после СТАРТ-состояния (адресный байт)

При таком формате посылки возможны различные комбинации чтения/записи. Подробнее диаграммы чтения/записи данных приведены в подразделах 2.2.4.6, 2.2.4.7 на примере памяти EEPROM с интерфейсом I^2C.

2.3.1.7 Арбитраж

Арбитраж помогает решать конфликтные ситуации во время передачи данных по I^2C, когда присутствует несколько ведущих (режим мультимастера). Ведущий может начинать пересылку данных только, если шина свободна. Если один ведущий передает на линию данных НИЗКИЙ уровень, в то время как другой – ВЫСОКИЙ, то последний отключается от линии, так как состояние SDA (НИЗКОЕ) не соответствует ВЫСОКОМУ состоянию его внутренней линии данных [7, 18].

Вследствие того, что арбитраж зависит только от адреса и данных, передаваемых соревнующимися ведущими, не существует центрального ведущего, а также приоритетного доступа к шине.

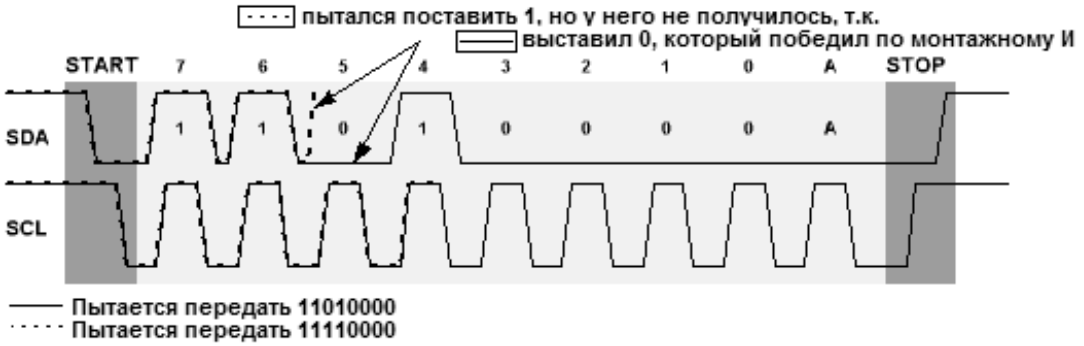


Рисунок 54. Арбитраж между двумя ведущими (случай одновременной передачи данных)

Что же будет, когда два ведущих начнут передачу одновременно? Тут опять помогает свойство монтажного И: оба мастера бит за битом передают адрес ведомого, потом данные, кто первый выставит на линию «0», тот и побеждает в этой конфликтной ситуации. Так что очевидно, что самый важный адрес должен начинаться с нулей, чтобы тот, кто к нему пытался обращаться, всегда выигрывал арбитраж. Проигравшая же сторона вынуждена ждать, пока шина не освободится.

Таким образом, арбитраж может продолжаться до окончания адреса, а если ведущие адресуют одно и то же устройство, то в арбитраже будут участвовать и

данные. Вследствие такой схемы арбитража при столкновении данные не теряются.

Ведущему, проигравшему арбитраж, разрешается выдавать синхроимпульсы на шину SCL до конца байта, в течение которого был потерян доступ.

Если в устройство ведущего также встроены и функции ведомого и он проигрывает арбитраж на стадии передачи адреса, то он немедленно должен переключиться в режим ведомого, так как выигравший арбитраж ведущий мог адресовать его.

2.3.1.8 Достоинства шины I²C

- Требуется только две линии – линия данных (SDA) и линия синхронизации (SCL) Каждое устройство, подключённое к шине, может быть программно адресовано по уникальному адресу. В каждый момент времени существует простое отношение ведущий/ведомый: ведущие могут работать как ведущий-передатчик и ведущий-приёмник.
- Шина позволяет иметь несколько ведущих, предоставляя средства для определения коллизий и арбитраж для предотвращения повреждения данных в ситуации, когда два или более ведущих одновременно начинают передачу данных. В стандартном режиме обеспечивается передача последовательных 8-битных данных со скоростью до 100 кбит/с, и до 400 кбит/с в “быстром” режиме.
- Встроенный в микросхемы фильтр подавляет всплески, обеспечивая целостность данных.
- Максимальное допустимое количество микросхем, подсоединённых к одной шине, ограничивается максимальной емкостью шины 400 пФ.

Это лишь некоторые преимущества. Кроме того, I²C-совместимые микросхемы увеличивают гибкость системы, позволяя простое конструирование вариантов оборудования и легкую модернизацию для того, чтобы поддерживать разработки на современном уровне [7, 18]. Таким образом, целое семейство оборудования может быть разработано, основываясь на базовой модели. Модернизация оборудования или расширение его функций (например, дополнительная память, дистанционное управление и т.п.) может быть произведена путем простого подключения соответствующей микросхемы к шине. Если требуется большая ПЗУ, то дело лишь в выборе микроконтроллера с большим объемом ПЗУ. Поскольку новые микросхемы могут замещать старые, легко добавлять новые свойства в оборудование или увеличивать его производительность путем простого отсоединения устаревшей микросхемы и подключения к шине новой.

2.3.2 Интерфейс RS-485

RS-485 (Recommended Standard 485, Electronics Industries Association 485, EIA-485) – стандарт передачи данных по двухпроводному полудуплексному многоточечному последовательному каналу связи.

Стандарт RS-485 совместно разработан двумя ассоциациями: Ассоциацией электронной промышленности (EIA — Electronics Industries Association) и Ассоциацией промышленности средств связи (TIA — Telecommunications Industry Association). Ранее EIA маркировала все свои стандарты префиксом «RS» (Recommended Standard — Рекомендованный стандарт). Многие инженеры продолжают использовать это обозначение, однако EIA/TIA официально заменил «RS» на «EIA/TIA» с целью облегчить идентификацию происхождения своих стандартов. На сегодняшний день, различные расширения стандарта RS-485 охватывают широкое разнообразие приложений, этот стандарт стал основой для создания целого семейства промышленных сетей широко используемых в промышленной автоматизации.

В стандарте RS-485 для передачи и приёма данных часто используется единственная витая пара проводов. Передача данных осуществляется с помощью дифференциальных сигналов. Разница напряжений между проводниками одной полярности означает логическую единицу, разница другой полярности — ноль.

RS-485 имеет следующие особенности:

- возможность объединения несимметричных и симметричных цепей,
- параметры качества сигнала, уровень искажений (%),
- методы доступа к линии связи,
- протокол обмена,
- аппаратную конфигурацию (среда обмена, кабель),
- типы соединителей, разъёмов, колодок, нумерацию контактов,
- качество источника питания (стабилизация, пульсация, допуск),
- отражения в длинных линиях.

Электрические и временные характеристики интерфейса RS-485:

- 32 приёмопередатчика при многоточечной конфигурации сети (на одном сегменте, максимальная длина линии в пределах одного сегмента сети: 1200 метров).
- Только один передатчик активный.
- Максимальное количество узлов в сети — 250 с учётом магистральных усилителей.

Характеристика скорость обмена/длина линии связи (зависимость экспоненциальная):

- 62,5 кбит/с 1200 м (одна витая пара)
- 375 кбит/с 300 м (одна витая пара)

- 500 кбит/с
- 1000 кбит/с
- 2400 кбит/с 100 м (две витых пары)
- 10000 кбит/с 10 м

Примечание: Скорости обмена 62,5 кбит/с, 375 кбит/с, 2400 кбит/с оговорены стандартом RS-485. На скоростях обмена свыше 500 кбит/с рекомендуется использовать экранированные витые пары.

Тип приёмопередатчиков — дифференциальный, потенциальный. Изменение входных и выходных напряжений на линиях A и B: Ua (Ub) от −7В до +12В (+7В).

2.3.2.1 Согласование и конфигурация линии связи

При больших расстояниях между устройствами, связанными по витой паре и высоких скоростях передачи начинают проявляться так называемые эффекты длинных линий. Причина этому – конечность скорости распространения электромагнитных волн в проводниках. Скорость эта существенно меньше скорости света в вакууме и составляет немногим больше 200 мм/нс. Электрический сигнал имеет также свойство отражаться от открытых концов линии передачи и ее ответвлений. Грубая аналогия - желоб, наполненный водой. Волна, созданная в одном конце, идет по желобу и, отразившись от стенки в конце, идет обратно, отражается опять и так далее, пока не затухнет. Для коротких линий и малых скоростей передачи этот процесс происходит так быстро, что остается незамеченным. Однако, время реакции приемников - десятки/сотни нс. В таком масштабе времени несколько десятков метров электрический сигнал проходит отнюдь не мгновенно. И если расстояние достаточно большое, фронт сигнала, отразившийся в конце линии и вернувшийся обратно, может исказить текущий или следующий сигнал. В таких случаях нужно каким-то образом подавлять эффект отражения.

У любой линии связи есть такой параметр, как волновое сопротивление Zв. Оно зависит от характеристик используемого кабеля, но не от длины. Для обычно применяемых в линиях связи витых пар Zв=120 Ом. Оказывается, что если на удаленном конце линии, между проводниками витой пары включить резистор с номиналом равным волновому сопротивлению линии, то электромагнитная волна дошедшая до "тупика" поглощается на таком резисторе. Отсюда его названия – согласующий резистор или "терминатор".

Большой минус согласования на резисторах – повышенное потребление тока от передатчика, ведь в линию включается низкоомная нагрузка. Поэтому рекомендуется включать передатчик только на время отправки посылки. Есть способы уменьшить потребление тока, включая последовательно с согласующим резистором конденсатор для развязки по постоянному току. Однако, такой способ имеет свои недостатки. Для коротких линий (несколько десятков метров) и низких скоростей (меньше 38400 бод) согласование можно вообще не делать.

Эффект отражения и необходимость правильного согласования накладывают ограничения на конфигурацию линии связи.

Линия связи должна представлять собой один кабель витой пары. К этому кабелю присоединяются все приемники и передатчики. Расстояние от линии до микросхем интерфейса RS-485 должно быть как можно короче, так как длинные ответвления вносят рассогласование и вызывают отражения.

В оба наиболее удаленных конца кабеля (Zв=120 Ом) включают согласующие резисторы Rt по 120 Ом (0.25 Вт). Если в системе только один передатчик и он находится в конце линии, то достаточно одного согласующего резистора на противоположном конце линии.

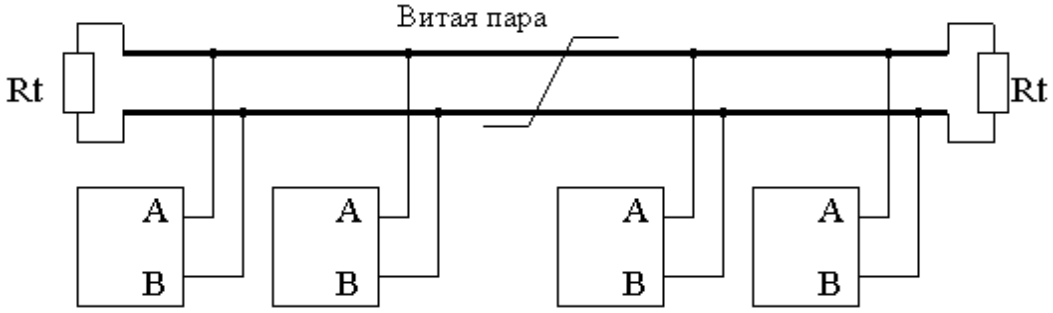


Рисунок 55. Согласование линии связи

2.3.2.2 Защитное смещение

Как уже упоминалось, приемники большинства микросхем RS-485 имеют пороговый диапазон распознавания сигнала на входах A-B - ±200мВ. Если |Uab| меньше порогового (около 0), то на выходе приемника RO могут быть произвольные логические уровни из-за несинфазной помехи. Такое может случиться либо при отсоединении приемника от линии, либо при отсутствии в линии активных передатчиков, когда никто не задает уровень. Чтобы в этих ситуациях избежать выдачи ошибочных сигналов на приемник UART, необходимо на входах A-B гарантировать разность потенциалов Uab > +200мВ. Это смещение при отсутствии входных сигналов обеспечивает на выходе приемника логическую "1", поддерживая, таким образом, уровень стопового бита.

Добиться этого просто - прямой вход (A) следует подтянуть к питанию, а инверсный (B) - к "земле".

Величины сопротивлений для резисторов защитного смещения (Rзс) нетрудно рассчитать по делителю. Необходимо обеспечить Uab > 200мВ. Напряжение питания – 5. Сопротивление среднего плеча – 120Ом//120Ом//12КОм на каждый приемник – примерно 57 Ом (для 10 приемников). Таким образом, выходит примерно по 650 Ом на каждый из двух Rзс. Для смещения с запасом - сопротивление Rзс должно быть меньше 650 Ом. Традиционно ставят 560 Ом.

Обратите внимание: в расчете номинала Rзс учитывается нагрузка. Если на линии висит много приемников, то номинал Rзс дожен быть меньше. В длинных линиях передачи необходимо так же учитывать сопротивление витой пары, которое может "съедать" часть смещающей разности потенциалов для удаленных от места подтяжки устройств. Для длинной линии лучше ставить два комплекта подтягивающих резисторов в оба удаленных конца рядом с терминаторами.

Функция безотказности

Многие производители приемопередатчиков заявляют о функции безотказности (failsafe) своих изделий, заключающейся во встроенном смещении. Следует различать два вида такой защиты:

- Безотказности в открытых цепях (Open circuit failsafe). В таких приемопередатчиках применяются встроенные подтягивающие резисторы. Эти резисторы, как правило, высокоомные, чтобы уменьшить потребление тока. Из-за этого необходимое смещение обеспечивается только для открытых (ненагруженных) дифференциальных входов. В самом деле, если приемник отключен от линии или она не нагружена, тогда в среднем плече делителя остается только большое входное сопротивление, на котором и падает необходимая разность потенциалов. Однако, если приемопередатчик нагрузить на линию с двумя согласующими резисторами по 120 Ом, то в среднем плече делителя оказывается меньше 60 Ом, на которых, по сравнению с высокоомными подтяжками, ничего существенного не падает. Поэтому, если в нагруженной линии нет активных передатчиков, то встроенные резисторы не обеспечивают достаточное смещение. В этом случае, остается необходимость устанавливать внешние резисторы защитного смещения, как это было описано выше.
- Истинная безотказность (True failsafe). В этих устройствах смещены сами пороги распознавания сигнала. Например: -50 / -200 мВ вместо стандартных порогов ±200 мВ. То есть при Uab>-50мВ на выходе приемника RO будет логическая "1", а при Uab<-200 - на RO будет "0". Таким образом, и в разомкнутой и в пассивной линии при разности потенциалов Uab близкой к нулю, приемник выдаст "1". Для таких приемопередатчиков внешнее защитное смещение не требуется. Тем не менее, для лучшей помехозащищенности все-таки стоит дополнительно немного подтягивать линию.

Сразу виден минус внешнего защитного смещения – через делитель постоянно будет протекать ток, что может быть недопустимо в системах малого потребления. В таком случае можно сделать следующее:

1. Уменьшить потребление тока, увеличив сопротивления Rзс. Хотя производители приемопередатчиков и пишут о пороге распознавания в 200мВ, на практике вполне хватает 100мВ и даже меньше. Таким образом, можно сразу увеличить сопротивления Rзс раза в два-три.

Помехозащищенность при этом несколько снижается, но во многих случаях это не критично.

2. Использовать true failsafe приемопередатчики со смещенными порогами распознавания. Например, у микросхем MAX3080 и MAX3471 пороги: -50мВ / -200мВ, что гарантирует единичный уровень на выходе приемника при отсутствии смещения (Uab=0). Тогда внешние резисторы защитного смещения можно убрать или значительно увеличить их сопротивление.
3. Не применять без необходимости согласование на резисторах. Если линия не будет нагружена на 2 по 120 Ом, то для обеспечения защитного смещения хватит подтяжек в несколько килоом в зависимости от числа приемников на линии.

Для опторазвязанной линии подтягивать следует к питанию и "земле" изолированной линии. Если не применяется опторазвязка, подтягивать можно к любому питанию, так как делитель создаст лишь небольшую разность потенциалов между линиями A и B. Нужно только помнить о возможной разности потенциалов между "землями" устройств, расположенных далеко друг от друга.

2.3.2.3 Исключение приема при передаче в полудуплексном режиме

При работе с полудуплексным интерфейсом RS-485 (прием и передача по одной паре проводов с разделением по времени) можно забыть, что UART контроллера полнодуплексный, то есть принимает и передает независимо и одновременно.

Обычно во время работы приемопередатчика RS-485 на передачу, выход приемника RO переводится в третье состояние и ножка RX контроллера (приемник UART) "повисает в воздухе". В результате, во время передачи на приемнике UART вместо уровня стопового бита ("1") окажется неизвестно что, и любая помеха будет принята за входной сигнал. Поэтому нужно либо на время передачи отключать приемник UART (через управляющий регистр), либо подтягивать RX к единице. У некоторых микроконтроллеров это можно сделать программно – активировать встроенные подтяжки портов.

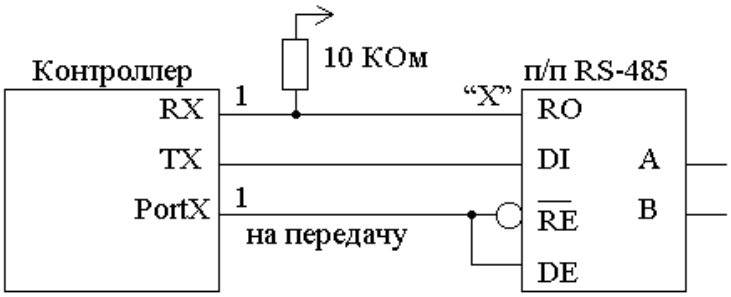


Рисунок 56. Схема подключения приемопередатчика RS-485 к микроконтроллеру

2.3.3 Интерфейс CAN

CAN (Controller Area Network) – последовательный протокол связи, который эффективно поддерживает распределенное управление в реальном масштабе времени с высоким уровнем безопасности. Рзработан фирмой BOSCH. Режим передачи – последовательный, широковещательный, пакетный. Стандарт не описывает физический уровень, но чаще всего используется сеть с топологией шина на базе дифференциальной пары, стандарта ISO 11898. Передача ведётся кадрами, которые принимаются всеми узлами сети. Метод доступа к среде передачи данных с разрешением конфликтов приоритетно обеспечивает доступ на передачу сообщения. Полезная информация в кадре состоит из идентификатора длиной 11 бит (стандартный формат) или 29 бит (расширенный формат) и поля данных длиной от 0 до 8 байт. Идентификатор говорит о содержимом пакета и служит для определения приоритета при попытке одновременной передачи несколькими узлами.

Область применения – от высокоскоростных сетей до дешевых мультиплексных шин. В автоматике, устройствах управления, датчиках используется CAN со скоростью до 1 Mbit/s.

Для достижения прозрачности проекта и гибкости реализации, CAN был подразделен на различные уровни согласно модели ISO/OSI:

- Уровень передачи данных (Data Link Layer)
- Подуровень логического управления линией (LLC)
- Подуровень управления доступом к среде передачи (MAC)
- Физический Уровень (Physical Layer)

Область LLC подуровня:

- обеспечение сервиса для передачи данных и для удалённого запроса данных.
- решение, какие сообщения, полученные LLC подуровнем, должны быть фактически приняты.
- обеспечение средствами для управления восстановлением и уведомления о перегрузке.

Область MAC подуровня главным образом - протокол передачи, то есть: арбитраж, проверка на ошибки, сигнализация и типизация ошибок. Внутри MAC подуровня решается, является ли шина свободной для начала новой передачи или возможен только приём данных. В MAC подуровень также включены некоторые элементы битовой синхронизации. Всё это находится внутри MAC подуровня и не имеет никакой возможности к модификации. Область физического уровня - фактическая передача битов между различными узлами с соблюдением всех электрических правил.

Внутри одной сети, физический уровень одинаков для всех узлов. Однако существует свобода в выборе физического уровня.

Основные характеристики:

- приоритетность сообщений;
- гарантированное время отклика;
- гибкость конфигурации;
- групповой прием с синхронизацией времени;
- система непротиворечивости данных;
- multimaster;
- обнаружение ошибок и их сигнализация;
- автоматическая ретрансляция испорченных сообщений, как только шина снова станет свободной;
- различие между нерегулярными ошибками и постоянными отказами узлов и автономное выключения дефектных узлов.
- Послойная архитектура CAN-сети согласно модели OSI.
- Физический уровень определяет, как сигналы фактически передаются и, следовательно, имеет дело с описанием битовой синхронизации и кодирования битов. Внутри этой спецификации характеристики передатчика / приемника физического уровня не определены, чтобы позволить среде передачи и реализации уровня сигнала быть оптимизированными для конкретных систем.
- MAC подуровень представляет собой ядро протокола CAN. Он передает сообщения, полученные от LLC подуровня, и принимает сообщения, которые будут переданы к LLC подуровню. MAC подуровень ответственен за арбитраж, подтверждение, обнаружение ошибок и их сигнализацию.
- LLC подуровень имеет отношение к фильтрации сообщений, уведомлению о перегрузке и управлению восстановлением [2].

2.3.4 Промышленный Ethernet

Industrial Ethernet (промышленный Ethernet) – стандартизованный (IEEE 802.3 и 802.11) вариант Ethernet для применения в промышленности. Сеть с процедурой доступа CSMA/CD. Industrial Ethernet обычно используется для обмена данными между программируемыми контроллерами и системами человеко-машинного интерфейса, реже для обмена данных между контроллерами и, незначительно, для подключения к контроллерам удаленного оборудования (датчиков и исполнительных устройств). Широкому применению Ethernet в последних задачах препятствует суть метода CSMA/CD, делающая невозможным гарантию обмена небольшим количеством информации (единицы байт) с высокой частотой (миллисекундные циклы обмена).

В последнее время является одной из самых распространённых промышленных сетей. Широко применяется при автоматизации зданий и в областях, не требующих высокой надёжности [31].

2.3.4.1 Протоколы реального времени

Для обеспечения гарантированного времени реакции используют протоколы реального времени:

- Profinet
- EtherCAT
- Ethernet Powerlink
- Ether/IP

Эти протоколы в различной степени модифицируют стандартный стек TCP/IP, добавляя в него:

- функции синхронизации
- новые алгоритмы сетевого обмена
- диагностические функции
- методы самокорректировки

Канальный и физический уровни Ethernet при этом остаются неизменными. Что позволяет использовать протоколы реального времени в существующих сетях Ethernet с использованием стандартного сетевого оборудования.

2.3.4.2 Резервирование каналов и кольцевая топология

Для обеспечения защиты каналов связи от единичного отказа необходимо их резервировать. Резервирование неизбежно ведет к возникновению кольцевых участков сети – замкнутых маршрутов. Стандарт Ethernet, предусматривает только древовидную топологию и не допускает кольцевых, так как это приводит к зацикливанию пакетов.

Современные коммутаторы, как правило, поддерживают дополнительный прокол Spanning Tree Protocol (STP, IEEE 802.1d), который позволяет создавать кольцевые маршруты в сетях Ethernet. Постоянно анализируя конфигурацию сети, STP автоматически выстраивает древовидную топологию, переводя избыточные коммуникационные линии в резерв. В случае нарушения целостности построенной таким образом сети (обрыв связи, например), STP в считанные секунды включает в работу необходимые резервные линии, восстанавливая древовидную структуры сети. Этот протокол не требует первичной настройки и работает автоматически.

Более мощная разновидность данного протокола - Rapid Spanning Tree Protocol (RSTP, IEEE 802.1w), позволяющая снизить время перестройки сети до нескольких миллисекунд. Протоколы STP и RSTP позволяют создавать произвольное количество избыточных линий связи и являются обязательным функционалом для промышленных коммутаторов, применяемых в резервированных сетях.

2.3.4.3 Отличия от обычного Ethernet

- Стандарты на кабели и разъемы, удовлетворяющие специфическим требованиям промышленности: усиленное экранирование, стойкость к агрессивным средам и т. п.
- Специальные стандарты и устройства для связи с подвижными объектами: гибкие кабели, устройства беспроводной связи
- Дополнение стека протоколов TCP/IP протоколом RFC 1006 обеспечивает регулярную и частую передачу по сети небольших объемов информации, что характерно для обмена данными между промышленными контроллерами
- С помощью специальных коммутаторов можно организовать кольцевую топологию, которая при обрыве восстанавливает связь, то есть находит новый путь для передачи данных значительно быстрее, чем применяемый в обычных сетях "алгоритм избыточного дерева"
- Частое использование наряду со стеком протоколов TCP/IP Специфического стека протоколов ISO Transport Protocol

2.3.5 Интерфейс LIN

LIN (Local Interconnect Network) – стандарт промышленной сети, разработаны консорциумом европейских автопроизводителей и других известных компаний, включая Audi AG, BMW AG, Daimler Chrysler AG, Motorola Inc., Volcano Communications Technologies AB, Volkswagen AG и VolvoCar Corporation. Протокол LIN предназначен для создания дешёвых локальных сетей обмена данными на коротких расстояниях. Он служит для передачи входных воздействий, состояний переключателей на панелях управления и так далее, а также ответных действий различных устройств, соединённых в одну систему через LIN, происходящих в так называемом «человеческом» временном диапазоне (порядка сотен миллисекунд).

Основные задачи, возлагаемые на LIN консорциумом европейских автомобильных производителей - объединение автомобильных подсистем и узлов (таких как дверные замки, стеклоочистители, стеклоподъёмники, управление магнитолой и климат-контролем, электролюк и так далее) в единую электронную систему. LIN-протокол утверждён Европейским Автомобильным Консорциумом как дешёвое дополнение к сверхнадёжному протоколу CAN.

LIN и CAN дополняют друг друга и позволяют объединить все электронные автомобильные приборы в единую многофункциональную бортовую сеть. Причём область применения CAN - участки, где требуется сверхнадёжность и скорость; область же применения LIN - объединение дешёвых узлов, работающих с малыми скоростями передачи информации на коротких дистанциях и сохраняющих при этом универсальность, многофункциональность, а также простоту разработки и отладки. Стандарт LIN включает технические требования на протокол и на среду передачи данных. Как последовательный протокол связи, LIN эффективно поддерживает

управление электронными узлами в автомобильных системах с шиной класса «А» (двунаправленный полудуплексный), что подразумевает наличие в системе одного главного (master) и нескольких подчинённых (slave) узлов.

2.3.6 Технология PLC

PLC (Power Line Communication/Carrier) – относительно новая телекоммуникационная технология категории «последняя миля». Так называемый «Интернет из розетки», базируется на использовании внутридомовых и внутриквартирных электросетей для высокоскоростного информационного обмена. В этой технологии, основанной на частотном разделении сигнала, высокоскоростной поток данных разбивается на несколько низкоскоростных, каждый из которых передается на отдельной частоте с последующим их объединением в один сигнал. При этом PLC-устройства могут «видеть» и декодировать информацию, хотя обычные электрические устройства – лампы накаливания, двигатели и т. п. – даже «не догадываются» о присутствии сигналов сетевого трафика и работают в обычном режиме.

Основой технологии Power Line является использование частотного разделения сигнала, при котором высокоскоростной поток данных разбирается на несколько относительно низкоскоростных потоков, каждый из которых передается на отдельной поднесущей частоте с последующим их объединением в один сигнал. Реально в технологии Power Line используются 84 поднесущие частоты в диапазоне 4-21 Мгц.

PLC включает BPL (Broadband over Power Lines – широкополосная передача через линии электропередачи), обеспечивающий передачу данных со скоростью более 1 Мбит в секунду, и NPL (Narrowband over Power Lines – узкополосная передача через линии электропередач) с намного меньшими скоростями передачи данных.

При передаче сигналов по бытовой электросети могут возникать большие затухания в передающей функции на определенных частотах, что может привести к потере данных. В технологии PowerLine предусмотрен специальный метод решения этой проблемы – динамическое включение и выключение передачи сигнала (dynamically turning off and on data-carrying signals). Суть данного метода заключается в том, что устройство осуществляет постоянный мониторинг канала передачи с целью выявления участка спектра с превышением определенного порогового значения затухания. В случае обнаружения данного факта, использование этих частот на время прекращается до восстановления нормального значения затухания.

Существует также проблема возникновения импульсных помех (до 1 микросекунды), источниками которых могут быть галогенные лампы, а также включение и выключение мощных бытовых электроприборов, оборудованных электрическими двигателями.

PDSL – технология семейства xDSL, обеспечивающая симметричную передачу данных со скоростью до 2Мбит/с по силовым кабелям (4-20 кВ), параллельно с транспортируемым электричеством. Подключение оборудования PDSL к высоковольтным линиям осуществляется посредством устройств сопряжения, которые устанавливаются в трансформаторных шкафах.

2.3.6.1 Преимущества

- Не требуется прокладка кабеля, заключение его в короба, сверление; стен и опорных конструкций;
- Простота использования;
- Скорость монтажа.

2.3.6.2 Преимущества PLC по сравнению с Wi-Fi

- Не требует настроек;
- Более стабильная связь;
- Большая безопасность информации;
- Подходит для передачи Multicast-трафика, например, IPTV;
- На качество связи не влияет материал и толщина стен в квартире;
- В РФ не требуется регистрация оборудования в Роскомнадзоре.

2.3.6.3 Недостатки

- Нарушение радиоприема в помещениях, где работают PLC-модемы, особенно на средних и коротких волнах, но на очень небольшом расстоянии порядка 3-5 метров от модема.
- Пропускная способность сети по электропроводке делится между всеми её участниками. Например, если в одной Powerline-сети две пары адаптеров активно обмениваются информацией, то скорость обмена для каждой пары будет составлять примерно по 50% от общей пропускной способности.
- На стабильность и скорость работы PLC влияет качество выполнения электропроводки, наличие стыков из разных материалов (например, медного и алюминиевого проводника), а также просто количество соединений проводника.
- Не работает через сетевые фильтры и ИБП, не оборудованные специальными розетками "PLC READY".
- На качество связи могут оказывать отрицательное влияние дешевые энергосберегающие лампы, тиристорные диммеры, импульсные блоки питания и зарядные устройства. Максимальное влияние на скорость в сети перечисленные устройства оказывают при подключении в непосредственной близости от PLC-модема.
- Неясные правовые аспекты использования данной технологии в Российской Федерации.

2.3.7 Технология M2M

M2M (Machine-to-Machine) – межмашинная коммуникация. Обычно системы M2M строятся на базе сотовых GPRS модемов. Связь осуществляется через встроенный в модуль модема стек TCP/IP. В связи с стем, что скорость обмена по сотовым сетям не очень велика, есть существенные задержки сигнала и возможны обрывы связи, основным применением таких систем является:

- Управление медленными процессами на больших территориях (системы управления наружным освещением, АСКУЭ, системы «умный дом», системы автоматизации ЖКХ);
- Сбор телеметрической информации с территориально-распределенных систем;
- Системы сигнализации для стационарных и подвижных объектов;
- Сбор информации о местонахождении подвижных объектов (учет пробега транспорта и так далее).

2.3.8 Стандарт ARINC 429

ARINC 429 - стандарт на компьютерную шину для применения в авионике. Разработан фирмой ARINC. Стандарт описывает основные функции и необходимые физические и электрические интерфейсы для цифровой информационной системы самолёта. Сегодня, ARINC 429 является доминирующей авиационной шиной для большинства хорошо экипированных самолётов.

ARINC 429 является двухпроводной шиной данных. Соединительные проводники – витые пары. Размер слова составляет 32 бита, а большинство сообщений состоит из единственного слова данных. Спецификация определяет электрические характеристики, характеристики обмена данными и протоколы. ARINC 429 использует однонаправленный стандарт шины данных (линии передачи и приёма физически разделены). Сообщения передаются на одной из трёх скоростей: 12,5, 50 или 100 Кбит/сек. Передатчик всегда активен, он либо передаёт 32-битовые слова данных или выдаёт «пустой» уровень. На шине допускается не более 20 приёмников, и не более одного передатчика.

2.3.9 Стандарт MIL-STD-1553

MIL-STD-1553 (MIL-STD-1553B) – стандарт Министерства обороны США, распространяется на магистральный последовательный интерфейс (МПИ) с централизованным управлением, применяемый в системе электронных модулей. ГОСТ 26765.52-87, ГОСТ Р 52070-2003, МКИО - российский аналог американского военного стандарта MIL-STD-1553 (MIL-STD-1553B).

Изначально разрабатывался по заказу МО США для использования в военной бортовой авионике, однако позднее спектр его применения существенно расширился, стандарт стал применяться и в гражданских

системах. Особенностью интерфейса является двойная избыточная линия передачи информации, полудуплексный протокол <команда-ответ> и до 31 удалённого абонента (оконечного устройства). Каждая линия управляется своим контроллером канала.

Стандарт устанавливает требования к:

- составу технических средств интерфейса;
- организации контроля передачи информации;
- характеристикам линии передачи информации (ЛПИ);
- характеристикам устройств интерфейса;
- интерфейсу с резервированием.

Впервые опубликован в США как стандарт ВВС в 1973 году, применён на истребителе F-16. Принят в качестве стандарта НАТО - STANAG 3838 AVS. В новейших самолетах заменяется стандартом IEEE 1394b.

2.3.9.1 Физический уровень

Одна шина состоит из пары проводов с волновым сопротивлением 70-85 Ом при частоте 1 МГц. Для соединения используется круглый разъём, по центральной ножке которого передаётся сигнал, закодированный Манчестерским кодом. Принимающее и передающее оконечные устройства подключаются к шине с использованием трансформаторной развязки, а не задействованные подключения отделяются с использованием пары изолирующих резисторов, развязанных через трансформатор. Это уменьшает влияние короткого замыкания и добавляет уверенности, что через шину ток не течёт по корпусу самолёта. Манчестерский код используется для того, чтобы передавать сигнал данных и сигнал синхронизации по одной паре проводников, а также для исключения любых постоянных составляющих, задерживаемых трансформаторной развязкой. Пропускная способность канала составляет 1 Мбит/с. Допуск на погрешность и долговременный дрейф пропускной способности составляет 0,1 %; краткосрочная стабильность тактовых импульсов должна быть в пределе 0,01 %. Амплитуда входного напряжения передатчика должна составлять 18-27 В.

Избыточность сообщений в системе передачи информации передачи может быть достигнута за счёт использования двух или трёх независимых каналов (проводников), к которым подключены все устройства на шине. Эти меры предосторожности приняты для того, чтобы можно было задействовать дублирующий контроллер шины, в случае отказа используемого в текущий момент.

Также существует вторая версия стандарта, известная как MIL-STD-1773, в которой в качестве канала передачи информации используется оптоволокно, имеющее меньший вес и лучшие показатели по электромагнитной совместимости.

Типичная шина MIL-STD-1553B может состоять из:

- Двух каналов (основного и резервного);
- Контроллера шины;
- Оконечных устройств;
- Монитора канала.

2.3.9.2 Контроллер шины

На одной шине может быть всего один контроллер в любой момент времени. Он является инициатором всех сообщений по этой шине.

Контроллер:

- Оперирует командами из списка в своей внутренней памяти;
- Командует оконечным устройствам послать или принять сообщения;
- Обслуживает запросы, получаемые от оконечных устройств;
- Фиксирует и восстанавливает ошибки;
- Поддерживает историю ошибок.

2.3.9.3 Оконечные устройства

Оконечные устройства служат для:

- Организации взаимодействия шины и подключаемой подсистемы;
- Организации моста между двумя шинами.

2.3.9.4 Монитор канала

Монитор канала отличается от оконечного устройства тем, что не может передавать сообщения по шине. Его роль заключается в мониторинге и записи транзакций по шине, без вмешательства во взаимодействие контроллера и оконечных устройств. Эта запись может быть использована для последующего анализа.

3 Программное обеспечение и инструментальные средства встраиваемых систем

Данная глава посвящена обзору программного обеспечения (ПО) встраиваемых систем, языков программирования, используемых для разработки ПО ВВС, специализированных инструментальных средств. Особое внимание уделяется обсуждению проблем проектирования программного проекта в общем и в случае встраиваемых систем, особенностям управления такого рода проектами.

3.1 Особенности программного обеспечения ВВС

3.1.1 Основные определения

Программное обеспечение – незафиксированная (soft – мягкий) часть системы, которую можно изменить. Неизменяемые системы (hard – твердый), к примеру, сетевой коммутатор, имеющий в своем составе ПО (даже целые ОС), тем не менее считается аппаратным обеспечением.

Операционная система реального времени (ОС РВ) – это средство распределения и выделения ресурсов встроенной системы.

Программируемый логический контроллер (ПЛК, PLC) – контроллер, программируемый конечным пользователем, а не профессионалом в области программирования. ПЛК обычно выпускаются в виде наборов модулей – конструкторов, из которых пользователь сам строит систему. В состав ПЛК входит, как правило, процессорный модуль и несколько модулей ввода-вывода.

3.1.2 Особенности ПО ВВС

К особенностям программного обеспечения встроенных систем, как уже говорилось, мы относим:

- Реальное время;
- Надёжность;
- Безопасность;
- Малые ресурсы аппаратуры (память, быстродействие, электропитание);
- Тяжелые условия эксплуатации платформы.

Программное обеспечение встроенных систем может быть построено следующими способами:

- Специально под задачу (специализированное ПО);
- На базе операционной системы реального времени;
- На базе ОС общего назначения;
- На базе виртуальной машины программируемого логического контроллера.

3.1.3 Операционные системы реального времени

ОС РВ в проектировании является некоторой постоянной составляющей, вынесенной за скобки после анализа множества монолитных реализаций программного обеспечения ВВС.

Что, по сути, дает применение ОС РВ во ВВС? Во-первых, это средство распределения ресурсов между прикладными процессами и средство организации этих процессов. Во-вторых, это отлаженный (то есть с минимальным количеством ошибок) программный код с полезной функциональностью. В-третьих, ОС РВ, как правило, является архитектурой с заведомо известными плюсами и минусами. В-четвертых, это средство для организации связи с достаточно большой номенклатурой аппаратных средств (различных контроллеров, периферийных устройств). Самостоятельная поддержка множества протоколов обмена, различных процессоров и контроллеров, как правило, оказывается нерентабельной для большинства компаний, создающих ВВС, что также определяет использование готовых ОС РВ.

Какие минусы может принести использование ОС РВ во ВВС? Естественно, большинство ОС РВ, присутствующие на рынке, разрабатывалась как относительно универсальные системы. Универсальность, как правило, означает избыточность функций и, следовательно, необходимость в дополнительных аппаратных ресурсах для поддержки этих функций. При использовании в проекте готовой ОС РВ существует возможность получения закрытой системы, то есть системы со скрытой внутренней структурой. Против использования такого "черного ящика" есть много аргументов. Самым сильным из них является невозможность проверки системы (например, при сертификации) на отсутствие серьезных ошибок и разного рода неучтенного, "шпионского" программного кода.

В последнее время популярен способ проектирования систем на базе шаблонов. Так в частности, в HW/SW CoDesign проектах используют заготовки ОС РВ (планировщики, переключатели процессов и другие). Эти шаблоны используются на этапе архитектурного проектирования. В результате на выходе системы проектирования разработчик получает монолитный код. Такой подход лишен большинства недостатков, присущих использованию универсальных (или покупных) ОС РВ.

Итак, основными причинами, заставляющими применять ОС РВ в составе программного обеспечения ВВС, будем считать:

- Необходимость использования готовой, надежной и предсказуемой платформы (выделение из множества программ стандартной составляющей, поддерживающей унификацию, стандартизацию, модульность);
- Необходимость обеспечения параллельного функционирования прикладных процессов;

- Необходимость обеспечения защиты процессов друг от друга;
- Необходимость в готовых драйверах периферийных устройств, вычислительной сети [40].

3.1.4 Программируемые логические контроллеры

Программируемый логический контроллер (ПЛК, PLC) – контроллер, программируемый конечным пользователем, а не профессионалом в области программирования. ПЛК обычно выпускаются в виде наборов модулей – конструкторов, из которых пользователь сам строит систему. В состав ПЛК входит, как правило, процессорный модуль и несколько модулей ввода-вывода.

3.1.4.1 Особенности ПЛК

- ПЛК позволяет добиться работы в реальном масштабе времени без операционной системы реального времени.
- Программы для ПЛК надёжнее программ, написанных с помощью обычных языков и с сипользованием обычных компиляторов для встроенных систем.
- Центральный процессор, точнее его регистры и система команд недоступна пользователю.

3.1.4.2 Варианты построения систем на базе ПЛК

Существует два основных варианта построения систем на базе ПЛК.

В первом варианте в ПЛК предусмотрены специальные разъемы расширения, в которые можно вставлять пассивные (т.е. без собственного процессора) модули ввода-вывода. Такой вариант предпочтителен, когда нужно сконцентрировать большую вычислительную мощность и большое число входов-выходов в одном месте.

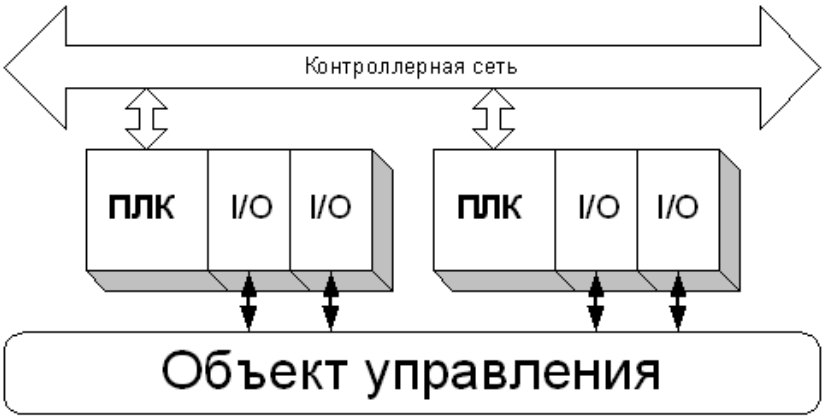


Рисунок 57. ПЛК с модулями расширения, подключаемыми через шину

Во втором варианте ПЛК не имеет своих входов выходов вообще или имеет их ограниченное количество. Дополнительное количество входов-выходов обеспечивается за счет подключения модулей ввода-вывода через специальную промышленную сеть. Последний вариант интересен тем, что позволяет достаточно гибко изменять масштаб системы управления, оставляя разработчикам значительную свободу в выборе решений.

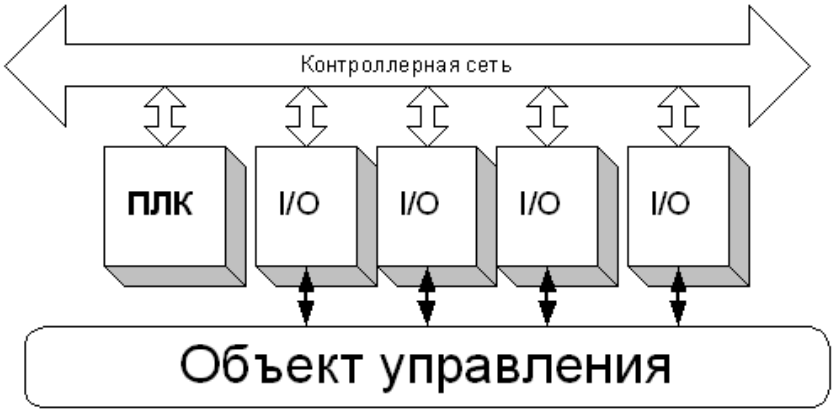


Рисунок 58. ПЛК с сетевыми модулями расширения

3.1.4.3 Особенности программирования ПЛК

Как правило, люди, программирующие ПЛК, являются не профессиональными программистами, а специалистами в какой-либо предметной области. Чаще всего ПЛК используются в АСУТП в качестве промышленного контроллера. Программирование ПЛК ведётся с помощью специальных языков программирования IEC1131-3, IEC61131-3, IEC-61499 и др., позволяющих полностью изолировать уровень системного программирования от программиста, достичь весьма высокой надежности функционирования и работы в реальном масштабе времени.

3.1.4.4 Варианты реализации ПЛК

Существует два полярных варианта реализации ПЛК.

Soft PLC

В первом случае, в качестве аппаратной базы берется обычный промышленный компьютер и снабжается операционной системой реального времени или DOS для индустриальных приложений (для компьютеров на базе процессора Intel). Далее, на этом промышленном компьютере запускается специальная программа – виртуальная машина ПЛК, реализующая одну или несколько вычислительных моделей, используемых в языках программироваания для ПЛК. В результате мы получаем так называемый Soft PLC. Этот вариант построения ПЛК интересен своей гибкостью. Конечный пользователь может в широких пределах менять характеристики программного обеспечения. Недостатками такого решения являются высокая цена

компонентов системы. Вам придется покупать промышленный компьютер, операционную систему и виртуальную машину ПЛК. Кроме того, если вы не специалист по операционным системам реального времени, вы можете получить достаточно низкие характеристики системы.

Специализированный ПЛК

Во втором случае в качестве аппаратной базы используется не промышленный компьютер, а специализированный контроллер. Все необходимое программное обеспечение уже прошито в ПЗУ на заводе изготовителе. Пользователю обычно остается только работы по конфигурации сети и разработке прикладной программы. Обычный ПЛК можно реализовать как Soft PLC, закрытый для изменения пользователя, или как специализированную вычислительную машину, с аппаратной поддержкой моделей вычислений, используемых в языках программирования ПЛК.

Достоинством системы на базе специализированного ПЛК является низкая стоимость, простота использования и высокая надежность. К недостаткам можно меньшую расширяемость аппаратной части специализированного контроллера и невозможность изменения системного программного обеспечения конечным пользователем.

3.1.4.5 Цикл ПЛК

В основе работы программируемого логического контроллера лежит циклическое исполнение программы.

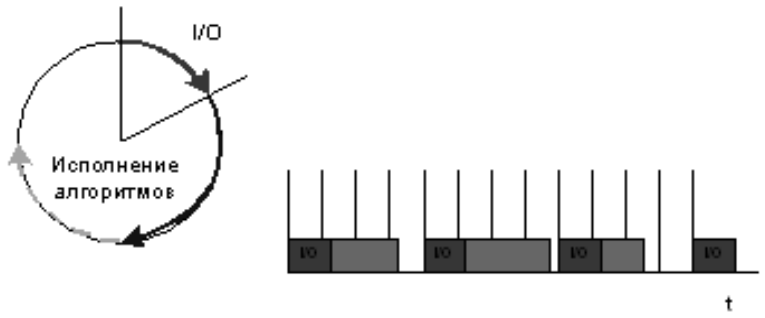


Рисунок 59. Цикл ПЛК

В начале цикла исполняется ввод-вывод. Происходит обмен между различными устройствами по сети, получение информации с датчиков и вывод информации на исполнительные устройства. После этого происходит исполнение алгоритмов управления. Через некоторое время цикл повторяется снова. Особенность такого решения состоит в том, что обмен с устройствами ввода-вывода происходит через строго определенные промежутки времени. Что это нам дает? А дает нам это гарантированное время реакции системы, т.е. мы получаем систему реального времени достаточно простым и надежным способом. Естественно, у любого метода есть свои недостатки. К недостаткам такого подхода можно отнести достаточно большое время простоя центрального процессора. Для того, чтобы цикл ПЛК был постоянным

необходимо, чтобы суммарное время ввода-вывода и время выполнения алгоритмов управления было всегда меньше периода цикла.

3.1.4.6 Области применения ПЛК

ПЛК хорошо приспособлены для широкого диапазона задач автоматики. Очень часто, это автоматизация промышленных процессов на производстве, где цена разработки и поддержки автоматических систем оказывается гораздо выше суммарной стоимости самих систем автоматики, и, там где требуются изменения в системах в течении периода их эксплуатации. ПЛК содержат устройства ввода/вывода совместимые с промышленными регуляторами и приводами, поэтому практически не требуют электротехнического проектирования, а большинство задач укладывается в описание требуемых наборов операций в нотации релейной логики или диаграмм состояний. ПЛК могут быть легко перенастроены на выполнение различных задач. Их цена, как правило, оказывается ниже в сравнении со заказными контроллерами. Хотя, в случае систем массового потребления, применение специализированных управляющих систем является более целесообразным, т.к. позволяет оптимально выбрать элементарную базу и сократить бесполезную избыточность.

3.1.4.7 Сравнение с микроконтроллерами

Решения на базе микроконтроллеров будут эффективней, в сравнении с ПЛК, там, где выпускаются сотни или тысячи устройств (повышенная цена разработки окупается за счет большого объема продаж) и там, где конечному пользователю не потребуется изменять алгоритмы управления. Примером могут служить системы автомобильной автоматики, миллионы устройств выпускаются ежегодно и только их производители занимаются программированием контроллеров. Также ПЛК могут оказаться неприменимыми в тех областях, где предъявляются повышенные требования к вычислительным ресурсам. В управлении химическим производством часто требуется реализация сложных алгоритмов управления, с которыми не справляются даже самые производительные ПЛК. Высокая скорость и точность вычислений требуется в бортовых системах авиации и военной техники.

3.2 Языки программирования

3.2.1 Основные определения

Язык – система знаков, сопряженная с правилами их связывания и служащая коммуникативным целям. Язык – средство, позволяющее передавать человеческие мысли и образы. Формальный язык позволяет однозначно воспринимать описание, неформальный допускает неоднозначности.

Язык программирования – язык, позволяющий выразить человеческие мысли и образы в формальном виде, однозначно воспринимаемом вычислительным устройством в рамках конкретной модели вычислений.

К составным частям языка можно отнести: лексику, синтаксис и семантику.

Лексика – описание элементов языка (его элементарных конструкций). Например, лексемами языка Си являются: for, if, int, char, while,+, -,...

Синтаксис – совокупность правил написания языковых элементов (лексем).

Семантика – смысловое наполнение языковых конструкций, связь между элементами языка и их значением.

Существуют языки, в которых лексемы и синтаксис очень близки, а семантика – различна. Обычно так происходит, если в основе языков лежат различные модели вычислений.

Существует несколько подходов к определению семантики языков программирования. Наиболее широко распространены разновидности следующих трёх: операционного, денотационного (математического) и деривационного (аксиоматического). При описании семантики в рамках операционного подхода обычно исполнение конструкций языка программирования интерпретируется с помощью некоторой воображаемой (абстрактной) ЭВМ. Деривационная семантика описывает последствия выполнения конструкций языка с помощью языка логики и задания пред- и постусловий. Денотационная семантика оперирует понятиями, типичными для математики – множества, соответствия, а также суждения, утверждения и др.

Статическая типизация – приём, широко используемый в языках программирования, при котором переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменён позже (переменная или параметр будут принимать, а функция – возвращать значения только этого типа). Примеры статически типизированных языков – Ада, Си++, Паскаль.

Динамическая типизация – приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов. Примеры языков, где есть динамическая типизация – Smalltalk Python, Ruby, PHP, Perl, JavaScript, Object Pascal, Lisp, xBase.

3.2.2 Классификация языков

Языки можно классифицировать по следующим критериям:

- Место в жизненном цикле проектирования (язык программирования, язык спецификации);
- Степень формальности (формальный язык, неформальный язык);
- Используемая модель вычислений;
- Полнота по Тьюрингу;
- Способ реализации (интерпретатор или компилятор);
- Способ типизации (динамическая или статическая);
- Тип используемой семантики (операционная, денотационная деривационная).

3.2.3 Языки спецификации и программирования

Программирование ВВС представляет собой сложный многоуровневый процесс, в котором языковые средства играют ключевую роль. В зависимости от этапа и характера использования языки принято делить на языки специфицирования (этап проектирования) и языки программирования (этап реализации), однако существует мнение, что любой формальный язык, в принципе, можно использовать на любом из этих этапов. В отличие от ВС общего назначения, в ВВС на этапе создания языкового описания некоторого объекта зачастую невозможно сказать, в каком виде этот объект будет реализован - как традиционная программа для программируемого процессора, как конфигурация ПЛИС или как специализированный аппаратный блок. Исходя из такой особенности применения языков при проектировании ВВС, наиболее общим и удобным следует считать взгляд на языки с позиции использования их для специфицирования средств ВВС для различных целей и на разных уровнях представления. Именно такой подход к классификации языков был предложен в, что естественным образом следует из концепции многоязыкового проектирования ВВС. Ниже мы рассмотрим языки, используемые при создании ВВС, именно с этих позиций. Основное внимание будет уделено средствам, используемым для проектирования на системном уровне.

Языки используются во время одного из самых важных шагов проектирования системы: на этапе спецификации системы. Существует множество языков спецификации. Каждый из них имеет свои преимущества, но превосходит другие только в пределах ограниченной прикладной области. Отбор языка - вообще компромисс между несколькими критериями, такими как выразительная мощь языка, возможности автоматизации, обеспеченные моделью, лежащей в основе языка, и наличие инструментальных средств и методов, поддерживающих язык. В ряде случаев полезно использовать несколько языков для спецификации различных модулей одного проекта. Многоязычные решения требуются для проектирования гетерогенных систем,

где различные части принадлежат разным прикладным категориям, например, управление/данные или непрерывный/дискретный.

Все инструментальные средства проектирования системы используют языки в качестве входных данных. Они обычно используют промежуточную форму, чтобы выполнить обработку и преобразование начальной спецификации. В основе языкового представления лежат модели вычисления, суть которых была представлена в предыдущем разделе. Модели вычислений могут быть ориентированы на данные или на управление. В обоих случаях, они могут быть синхронными или асинхронными.

3.2.4 Полнота по Тьюрингу

В теории вычислимости[2] исполнитель (множество вычисляющих элементов) называется тьюринг-полным, если на нём можно реализовать любую вычислимую функцию. Другими словами, для каждой вычислимой функции существует вычисляющий её элемент (например, машина Тьюринга) или программа для исполнителя, а все функции, вычисляемые множеством вычислителей, являются вычислимыми функциями (возможно, при некотором кодировании входных и выходных данных). Название пошло от Алана Тьюринга, который придумал абстрактный вычислитель - машину Тьюринга и дал определение множества функций, вычислимых посредством машин Тьюринга. Большинство широко используемых языков программирования - тьюринг-полные. Это касается как императивных языков, таких как Паскаль, так и функциональных (Haskell) и языков логического программирования (Prolog). Полными по Тьюрингу являются также грамматики общего вида. Примерами неполных по Тьюрингу формализмов являются конечные автоматы, примитивно рекурсивные функции, контекстно-свободные и регулярные грамматики.

3.2.5 Модель вычислений

Модель вычислений, вычислительная модель (model of computation, MOC) – набор законов взаимодействия элементов вычислительной системы.

Модель вычислений – набор правил организации вычислительного процесса, в рамках которых возможен его формальный анализ.

Модель вычислений – набор формальных правил, в рамках которых организована взаимосвязь и поведение множества составляющих атомарных частей модели некоторой вычислительной системы.

---

[2] Теория вычислимости – это раздел современной математики и теории вычислений, возникший в результате изучения понятий вычислимости и невычислимости. Изначально теория была посвящена вычислимым и невычислимым функциям и сравнению различных моделей вычислений. Сейчас поле исследования теории вычислимости расширилось - появляются новые определения понятия вычислимости и идёт слияние с математической логикой, где вместо вычислимости и невычислимости идёт речь о доказуемости и недоказуемости (выводимости и невыводимости) утверждений в рамках каких-либо теорий.

Модель вычислений – строго определенная парадигма (набор правил), описывающая протекание вычислительного процесса, способы обмена данными, взаимодействия между отдельными функциональными элементами.

Модель вычислений – недвусмысленный формализм для представления спецификаций проекта и проектных решений.

Модель вычислений – математическая модель, демонстрирующая пользователю вычислительные возможности вычислителя и правила их использования.

Теория вычислимости и теория сложности вычислений трактует модель вычисления не только как определение множества допустимых операций, использованных для вычисления, но также и относительных издержек их применения. Охарактеризовать необходимые вычислительные ресурсы время выполнения, объём памяти, а также ограничения алгоритмов или компьютера можно только в том случае, если выбрана определённая модель вычислений. В модельно-ориентированной инженерии модель вычислений и её выбор дают ответ на вопрос, как ведёт себя система в целом, если известно поведение её отдельных частей. При асимптотической оценке сложности вычислений модель вычислений определяется через допустимые примитивные операции, для каждой из которых известна её цена.

Известен целый ряд моделей вычислений, зависящих от набора применяемых операций и их вычислительной сложности. Они распадаются на следующие большие категории: абстрактные машины (абстрактные вычислители), используемые для доказательства вычислимости и получения верхней границы вычислительной сложности алгоритма и модели принятия решений, используемые для получения нижней границы сложности вычислений для алгоритмических задач.

Примеры языков принадлежащих различным моделям вычислений:

- Си, Pascal, Ada – императивная модель или модель Фон-Неймана;
- VHDL, Verilog – модель дискретных событий;
- Prolog, Рефал – сентенциональная модель;
- XML – иерархическая модель данных;
- SQL – реляционная модель;
- Lisp – функциональная модель [40].

3.2.6 Стиль программирования

Стиль программирования – внутренне концептуально согласованная совокупность средств, базирующаяся на некоторой логике построения программ. Впервые этот термин введен в книге "Основания программирования".

Стиль программирования следует отличать от стиля кодирования, который сводится к определенному способу форматирования исходного текста.

Термины программирование и программа необходимо определить более точно, чтобы не возникало неоднозначности. В настоящее время под программой понимается последовательность действий, совершаемых машиной Фон-Неймана, а в качестве стиля программирования рассматривается базирующийся на машине Фон-Неймана структурный стиль программирования. В таком контексте непонятно, можно ли считать программой текст, написанный на языке VHDL или Verilog, а также тексты, описывающие системы, работающие в других моделях вычислений.

Непейвода приводит следующую классификацию стилей программирования:

- сентенциальное программирование (Рефал, Prolog);
- функциональное программирование (Lisp);
- автоматное программирование;
- событийное программирование;
- структурное программирование (Си);
- параллельное программирование;
- объектно-ориентированное программирование [42, 43].

3.2.7 Стиль программирования, модель вычислений, платформа

Стиль программирования находится над моделью вычислений, так как модель вычислений задает набор правил, в рамках которых реализуется тот или иной стиль. Между решаемой задачей, стилем программирования и моделью вычислений не должно быть концептуальных противоречий. В противном случае возникает резкий скачок сложности проектируемой системы, увеличиваются количество ошибок, сроки выполнения, быстро заканчивается бюджет проекта. К сожалению, большинство программистов не воспринимает эту проблем, что обычно приводит к увеличению проектных бюджетов и огромному количеству ошибок.

В качестве примера нарушения гармонии между стилем программирования и моделью вычислений можно привести ОС РВ, в которых искусственно создается потоковая модель вычислений, а в качестве стиля программирования используется структурное программирование (чаще всего на языке Си). Аналогичные проблемы есть в современных операционных системах общего назначения (Microsoft Windows 2000/XP/Vista, Linux, FreeBSD, Mac OS и т.д.) и в системах программирования для них (Java, C#, C++ и т.д.). В своей статье "Проблемы с потоками" Эдвард Ли подробно освящает эту проблему [13].

Чтобы избежать противоречий, используют способ проектирования на базе платформ, в котором каждый слой имеет свою модель вычислений и гармонично связанный с ним стиль программирования. Каждый слой системы при этом является фундаментом для последующих слоев и перемешивания понятий, встречающегося в современных ОС РВ, не происходит [40].

3.2.8 Критерии оценки языков

Профессор Роберт У. Себеста из университета штата Колорадо, в своей книге «Основные концепции языков программирования» [47], к основным критериям оценки языков программирования относит:

- Удобочитаемость;
- Лёгкость создания программ;
- Надёжность.

К основным характеристикам языков программирование можно отнести:

- Простоту (минимум языковых конструкций);
- Ортогональность (способность языка порождать новые языковые конструкции, например, структуры данных, с помощью небольшого количества элементарных конструкций);
- Поддержку абстракции (выделение главного);
- Выразительность (понятно и компактно).

Таблица 8. Критерии оценки языков программирования

	Критерии		
Характеристики	**Удобочитаемость**	**Легкость создания**	**Надёжность**
Простота, ортогональность	●	●	●
Управляющие структуры	●	●	●
Типы и структуры данных	●	●	●
Синтаксическая структура	●	●	●
Поддержка абстракции		●	●
Выразительность		●	●
Проверка типов			●
Обработка исключительных ситуаций			●
Ограниченное совмещение имен			●

3.2.8.1 Удобочитаемость

Удобочитаемость (Readability) – лёгкость чтения и понимания программ, написанных на языке программирования. Необходимо понимать, что программа будет понятной и простой, если она написана на языке, подходящем для данной предметной области. В данном случае речь идёт о таких понятиях как модель вычислений и стиль программирования.

На удобочитаемость программ оказывает влияние простота языка. Чем больше в языке различных языковых конструкций, тем сложнее его использовать. Программисты, использующие большие языки программирования (например, такие как C++), очень часто используют только некоторое подмножество языковых конструкций. Необходимо заметить, что излишняя простота языка также отрицательно сказывается на читаемости программы. Например, ассемблер очень простой язык, но понять программу,

написанную на ассемблере в несколько раз сложнее, чем программу написанную на языке Си или Паскаль.

Пример программы, осуществляющей вывод текста “Hello World” на очень простом языке brainfuck. Этот язык имеет всего 8 команд и по своей идее очень близок к Машине Тьюринга.

```
++++++++++ [>+++++++>++++++++++>+++>+<<<<-]>++
 .>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.
 ------.--------.>+.>.
```

Как видите, с удобочитаемостью тут всё понятно. Естественно, язык Brainfuck был придуман как шутка. Основной целью создания такого языка было обеспечение максимально возможной простоты компилятора.

Еще одной характеристикой языка, снижающей его удобочитаемость, является множественность свойств, то есть наличие нескольких способов совершения каких-либо действий.

Третьей проблемой является перегрузка операторов, то есть наличие у одного и того же символа, обозначающего операцию, нескольких значений. Если программист будет перегружать операторы, без каких либо разумных оснований, программа может стать очень труднопонимаемой.

3.2.8.2 Лёгкость создания программ

Лёгкость создания программ – характеризует удобство создания программ в заданной области. Очень часто, при использовании языков, содержащих множество различных языковых конструкций, возникает ситуация, когда программисты либо забывают, либо используют неправильно языковые конструкции. В результате получаются менее изящные и менее эффективные программы.

С одной стороны, использование небольшого количества элементов и согласованных правил их использования (ортогональность) лучше, чем применение большого количества примитивов. С другой стороны, слишком ортогональная структура также усложняет использование языка.

Поддержка абстракции, позволяет определять и затем использовать сложные структуры и операции, позволяя игнорировать мелкие детали. Простой пример абстракции – подпрограмма.

Выразительность языка позволяет осуществлять написание программ с помощью более ёмких и компактных языковых конструкций. Сравните:

```
Count = Count + 1;
```

и

```
Count++;
```

3.2.8.3 Надёжность

Программа считается надёжной, если она соответствует своему предназначению в любых условиях.

Одним из важнейших факторов, влияющих на надёжность языка, является проверка типов. Языки, не имеющие (или допускающие отсутствие) проверки типов позволяют программистам делать грубые ошибки.

```
if( $s1 =~ /\ [/ )
{
      if( $Found==1 )
      {
            $Found=0;
            $N=0;
            if( ( $s1 =~ /\b$substring\b/ ) )
            {
                  $Found=1;
            }
            $Buf [ $N++ ] = $_;
      }
      else
      {
            $Buf [$N++] = $_;
            if( ( $c1 =~ /\b$substring\b/ ) )
            {
                  $Found=1;
            }
      }
}
```

Например, в приведенном выше примере программы на языке Perl, ошибка в имени переменной ($c1 вместо $s1) не будет замечена компилятором[3].

3.2.9 Требования к языкам для управляющих систем

Требования к языкам формируются исходя из общих требований: надёжность, реальное время.

Приветствуются:

- Простота;
- Выразительность;
- Ортогональность.

К примеру, Perl не подходит, между языками Си и Си++ выбираем Си. Почему?

3.2.10 Краткий обзор языков, используемых при проектировании ВВС

3.2.10.1 Язык программирования Си

Си (англ. C) – императивный язык программирования или язык, в основе которого лежит модель вычислений Фон-Неймана. Используется практически на всех уровнях пирамиды автоматизации. Наибольшее распространение получил на нижних (3 и 4) уровнях пирамиды.

[3] В случае, если не использовать use strict

Простота, легкость переноса компиляторов на разные аппаратные платформы, наличие указателей и битовых операций делает удобным использование языка Си для системного программирования, программирования встроенных систем и микроконтроллеров. Аскетичность языка Си с одной стороны, функциональность и мощь с другой, хорошо укладываются в принцип KISS[4].

Язык программирования Си разработан в начале 1970-х годов сотрудниками Bell Labs Кеном Томпсоном и Денисом Ритчи как развитие языка Би. Си был создан для использования в операционной системе (ОС) UNIX. С тех пор он был перенесен на многие другие операционные системы и стал одним из самых используемых языков программирования.

Си ценят за его эффективность; он является самым популярным языком для создания системного программного обеспечения. Его также часто используют для создания прикладных программ. Несмотря на то, что Си не разрабатывался для новичков, он активно используется для обучения программированию. В дальнейшем синтаксис языка Си стал основой для многих других языков.

Для языка Си характерны лаконичность, современный набор конструкций управления потоком выполнения, структур данных и обширный набор операций.

Многие элементы Си потенциально опасны, а последствия неправильного использования этих элементов зачастую непредсказуемы. Керниган говорит: «Си – инструмент, острый, как бритва: с его помощью можно создать и элегантную программу, и кровавое месиво». В связи со сравнительно низким уровнем языка многие случаи неправильного использования опасных элементов не обнаруживаются и не могут быть обнаружены ни при компиляции, ни во время исполнения. Они часто приводит к непредсказуемому поведению программы. Иногда в результате неграмотного использования элементов языка появляются уязвимости в системе безопасности. Необходимо заметить, что использования многих таких элементов можно избежать.

Чаще всего источником ошибки является обращение к несуществующему элементу массива. Несмотря на то, что Си непосредственно поддерживает статические массивы, он не имеет средств проверки индексов массивов (проверки границ). Например, возможна запись в шестой элемент массива из пяти элементов, что, естественно, приведёт к непредсказуемым результатам. Частный случай такой ошибки называется ошибкой переполнения буфера. Ошибки такого рода приводят к большинству проблем с безопасностью.

Другим потенциальным источником опасных ситуаций служит механизм указателей. Указатель может указывать на абсолютно любой объект в памяти,

[4] Принцип "KISS" (англ. Keep It Simple, Stupid- "будь проще, тупица") - процесс и принцип проектирования, при котором простота системы декларируется в качестве основной цели и/или ценности.

включая даже и сам машинный код программы, что может приводить к непредсказуемым эффектам. Несмотря на то, что большинство указателей, как правило, указывают на безопасные места, они легко могут быть передвинуты в уже небезопасные области памяти с помощью арифметики указателей; память, на которую они указывают, может быть освобождена и использована по-другому («висячие указатели»); они могут быть не инициализированы («дикие указатели»); или же они просто могут получить любое значение путём приведения типов или присваивания значения другого повреждённого указателя. Другие языки пытаются решить эти проблемы путём использования более ограниченных типов – ссылок.

Одна из таких проблем заключается в том, что автоматически и динамически создаваемые объекты не инициализируются, поэтому в начале они имеют такое значение, какое осталось в памяти, выделенной для них от ранее удалённых объектов. Такое значение полностью непредсказуемо, оно меняется от одной машины к другой, от запуска к запуску, от вызова функции к вызову. Если программа попытается использовать такое неинициализированное значение, то придёт к непредсказуемому результату. Большинство современных компиляторов пытаются обнаружить эту проблему в некоторых случаях.

Функции с переменным количеством аргументов также являются потенциальным источником проблем. В отличие от обычных функций, имеющих прототип, стандартом не регламентируется проверка функций с переменным числом аргументов. Если передаётся неправильный тип данных, то возникает непредсказуемый, если не фатальный результат. Например, семейство функций printf стандартной библиотеки языка Си, используемое для генерации форматированного текста для вывода, хорошо известно своим потенциально опасным интерфейсом с переменным числом аргументов, которые описываются строкой формата. Проверка типов в функциях с переменным числом аргументов является задачей каждой конкретной реализации такой функции, однако многие современные компиляторы проверяют типы в каждом вызове printf, генерируя предупреждения в случаях, когда список аргументов не соответствует строке формата. Следует заметить, что невозможно статически проконтролировать даже все вызовы функции printf, поскольку строка формата может создаваться в программе динамически, поэтому, как правило, никаких проверок других функций с переменным числом аргументов компилятором не производится. Для помощи программистам на Си в решении этих и многих других проблем было создано большое число отдельных от компиляторов инструментов. Такими инструментами являются программы дополнительной проверки исходного кода и поиска распространённых ошибок, а также библиотеки, предоставляющие дополнительные функции, не входящие в стандарт языка, такие как проверка границ массивов или ограниченная форма сборки мусора.

Ещё одной распространённой проблемой является то, что память не может быть использована снова, пока она не будет освобождена программистом с помощью функции free(). В результате программист может случайно забыть освободить эту память и продолжить её выделять, занимая всё большее и большее пространство. Это обозначается термином утечка памяти. Наоборот, возможно освободить память слишком рано, но продолжать её использовать. Из-за того, что система выделения может использовать освобождённую память по-другому, это ведёт к непредсказуемым последствиям. Эти проблемы решаются в языках со сборкой мусора. С другой стороны, если память выделяется в функции и должна освобождаться после выхода из функции, данная проблема решается с помощью автоматического вызова деструкторов в языке C++, или с помощью локальных массивов, используя расширения C99.

Основная масса инструментального обеспечения для разработки встраиваемых систем продается за деньги. Стоимость компиляторов или наборов инструментальных средств составляет от сотен до нескольких тысяч евро.

Сравнительно небольшое количество производителей микропроцессоров предлагает свои компиляторы бесплатно (для примера AVR Studio фирмы Atmel и Softune Workbench фирмы Fujitsu).

Коммерческие компиляторы имеют следующие характерные особенности:

- Официальная поддержка;
- Хорошая документация;
- Меньшее количество ошибок в генерируемом коде (к сожалению, не всегда и не у всех);
- Улучшенная оптимизация кода;
- Более дружелюбный интерфейс пользователя.

Как правило, начинающие разработчики гораздо проще разбираются в коммерческих инструментальных средствах.

Некоторое количество компиляторов, используемых во встроенных системах, выпускается под лицензией GNU GPL. GNU General Public License (иногда переводят, как, например, Универсальная общественная лицензия GNU, Универсальная общедоступная лицензия GNU или Открытое лицензионное соглашение GNU) – лицензия на свободное программное обеспечение, созданная в рамках проекта GNU в 1988 г. Её также сокращённо называют GNU GPL, или даже просто GPL, если из контекста понятно, что речь идёт именно о данной лицензии (существует довольно много других лицензий, содержащих слова «general public license» в названии).

Цель GNU GPL – предоставить пользователю права копировать, модифицировать и распространять (в том числе на коммерческой основе) программы (что по умолчанию запрещено законом об авторских правах), а также гарантировать, что и пользователи всех производных программ получат вышеперечисленные права. Принцип «наследования» прав называется

«копилефт» (транслитерация английского «copyleft») и был придуман Ричардом Столлмэном. По контрасту с GPL, лицензии собственнического ПО очень редко дают пользователю такие права и обычно, наоборот, стремятся их ограничить, например, запрещая восстановление исходного кода.

Использование свободно распространяемых инструментальных средств порождает некоторое количество проблем.

- Существует необходимость в достаточно высоком начальным уровне подготовки разработчика.
- Свободно-распространяемые компиляторы генерируют менее эффективный код.
- Официальная поддержка отсутствует (тем не менее, присутствует неплохая неофициальная).

В целом, инструментальные средства выпущенные в рамках лицензии GPL являются в некоторых случаях очень неплохой альтернативой коммерческим продуктам [38, 39, 46].

3.2.10.2 Язык программирования C++

Си++ (англ. C++) – компилируемый строго типизированный язык программирования общего назначения. Поддерживает разные парадигмы программирования: процедурную, обобщённую, функциональную; наибольшее внимание уделено поддержке объектно-ориентированного программирования.

В 1990-х годах язык стал одним из наиболее широко применяемых языков программирования общего назначения.

При создании Си++ стремились сохранить совместимость с языком Си. Большинство программ на Си будут исправно работать и с компилятором Си++. Си++ имеет синтаксис, основанный на синтаксисе Си.

Нововведениями Си++ в сравнении с Си являются:

- Поддержка объектно-ориентированного программирования через классы;
- Поддержка обобщённого программирования через шаблоны;
- Дополнения к стандартной библиотеке;
- Дополнительные типы данных;
- Исключения;
- Пространства имён;
- Встраиваемые функции;
- Перегрузка операторов;
- Перегрузка имён функций;
- Ссылки и операторы управления свободно распределяемой памятью.

Язык возник в начале 1980-х годов, когда сотрудник фирмы "Bell Laboratories" Бьярне Строуструп придумал ряд усовершенствований к языку Си под собственные нужды. До начала официальной стандартизации язык развивался в основном силами Строуструпа в ответ на запросы программистского сообщества. В 1998 году был ратифицирован международный стандарт языка Си++: ISO/IEC 14882:1998 "Standard for the C++ Programming Language"; после принятия технических исправлений к стандарту в 2003 году нынешняя версия этого стандарта - ISO/IEC 14882:2003.

Название "Си++" происходит от Си, в котором унарный оператор ++ обозначает приращение.

В книге "Дизайн и развитие C++" Бьярне Строуструп описывает некоторые правила, которые он использовал при проектировании Си++. Знание этих правил может помочь понять, почему Си++ такой, каким он стал. Вот некоторые из этих правил (подробности можно найти в "Дизайне и развитии C++").

Си++:

- Разработан как универсальный язык со статическими типами данных, эффективностью и переносимостью языка Си.
- Разработан так, чтобы непосредственно и всесторонне поддерживать множество стилей программирования (процедурное программирование, абстракцию данных, объектно-ориентированное программирование и обобщённое программирование).
- Разработан так, чтобы давать программисту свободу выбора, даже если это даёт ему возможность выбирать неправильно.
- Разработан так, чтобы максимально сохранить совместимость с Си, тем самым делая возможным лёгкий переход от программирования на Си.
- Избегает таких особенностей, которые зависят от платформы или не являются универсальными.
- Не накладывает никакой избыточной нагрузки на программу, не использующую какие-либо возможности.
- Разработан так, чтобы не требовать слишком усложнённой среды программирования [48].

Для встроенных систем разработан упрощенный диалект языка C++ - embedded C++. К основным отличиям относится отсутствие ряда следующих возможностей:

- Множественное наследование;
- Виртуальные базовые классы;
- Информация о типах на этапе исполнения (typeid);
- Новый стиль преобразования типов (static_cast, dynamic_cast, reinterpret_cast, const_cast);
- Пространства имён;

- Исключения;
- Шаблоны.

3.2.10.3 Платформа Java

Java ("Ява", произносится "Джава") – объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения.

Программы на Java могут быть транслированы в байт-код, выполняемый на виртуальной джава-машине (JVM) – программе, обрабатывающей байтовый код и передающей инструкции оборудованию, как интерпретатор, но с тем отличием, что байтовый код в отличие от текста обрабатывается значительно быстрее.

Достоинство подобного способа выполнения программ – в полной независимости байт-кода от ОС и оборудования, что позволяет выполнять Java приложения на любом устройстве, которое поддерживает виртуальную машину. Другой важной особенностью технологии Java является гибкая система безопасности, благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание. Это позволяет пользователям загружать программы, написанные на Java, на их компьютеры (или другие устройства, например, мобильные телефоны) из неизвестных источников, при этом не опасаясь заражения вирусами, пропажи ценной информации, и т. п.

Часто к недостаткам этого подхода относят то, что исполнение байт-кода виртуальной машиной может снижать производительность программ и алгоритмов, реализованных на языке Java. Данное утверждение справедливо для первых версий виртуальной машины Java, однако в последнее время оно практически потеряло актуальность. Этому способствовал ряд усовершенствований: применение технологии JITs (Just-In-Time compiler), позволяющей переводить байт-код в машинный код во время исполнения программы с возможностью сохранения версий класса в машинном коде, широкое использование native-кода в стандартных библиотеках, а также аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM).

Внутри Java существуют 3 основных семейства технологий:

- J2EE или Java EE (начиная с v1.5) – Java Enterprise Editon, для создания программного обеспечения уровня предприятия;
- J2SE или Java SE (начиная с v1.5) – Java Standard Editon, для создания пользовательских приложений, в первую очередь – для настольных систем;
- J2ME, Java ME или Java Micro Edition, для использования в устройствах, ограниченных по вычислительной мощности, в том числе мобильных телефонах, PDA, встроенных системах.

Прежняя версия JVM от Microsoft (аналог SUN JVM v.1.1.3) во многом отступает от стандартов языка, предложенных Sun Microsystems, с целью проприетарной поддержки платформы Windows. Впоследствии это явилось поводом для судебных исков со стороны Sun Microsystems к Microsoft. В настоящее время между двумя компаниями достигнуты договорённости вплоть до снятия взаимных судебных претензий и произведено взаимное кросс-лицензирование технологий. По версии Microsoft будет поддерживаться спецификация MS-J# соответствующая спецификации SUN-JVM J2SE.

Java 2 Micro Edition

Java 2 Micro Edition (J2ME) – подмножество технологий фирмы Sun Microsystems, основанное на концепции Java-платформы и предназначенное для выполнения приложений, написанных на языке Java на устройствах бытовой электроники, например мобильных телефонах, персональных органайзерах, цифровых телевизионных ресиверах и т. п. Основой J2ME является виртуальная машина, способная исполнять байт-код языка Java.

J2ME задумана для того, чтобы обеспечить эффективное исполнение Java-приложений на устройствах бытовой электроники, отличительной особенностью которых является ограниченная вычислительная мощность, ограниченный объём памяти, малый размер дисплея, питание от портативной батареи, а также низкоскоростные и недостаточно надёжные коммуникационные возможности. Типичный современный мобильный телефон содержит внутри 32-разрядный RISC-процессор с тактовой частотой 50 МГц, имеет объём оперативной памяти около 4 Мб, цветной дисплей размером 2 дюйма и имеет возможность GPRS-соединения с Интернетом со скоростью до 172 Кб/с, которое при этом фундаментально ненадёжно, поскольку скорость передачи данных может неожиданно упасть, или соединение может быть вообще полностью потеряно.

J2ME специфицирует две базовые конфигурации, которые определяют требования к виртуальной машине или, иначе говоря, определяют подмножество стандартного языка Java, которое виртуальная машина способна выполнять, а также минимальный набор базовых классов: CLDC (Connected Limited Device Configuration – конфигурация устройства с ограниченными коммуникационными возможностями) и CDC (Connected Device Configuration –

конфигурация устройства с нормальными коммуникационными возможностями).

J2ME также определяет несколько так называемых профилей (profiles), которые дополняют и расширяют упомянутые выше конфигурации, в частности определяют модель приложения, возможности графического интерфейса, а также коммуникационные функции (например доступ к Интернету) и др.

В настоящее время самой распространённой конфигурацией является CLDC, для которого разработан профиль MIDP (Mobile Information Device Profile – профиль для мобильного устройства с информационными функциями). MIDP определяет понятие мидлета (MIDlet) – компактного Java-приложения, что делает его пригодным для передачи по сети и установки на мобильном устройстве. Другим популярным профилем для J2ME/CLDC является DoJa, разработанный фирмой NTT DoCoMo для её собственного сервиса iMode. iMode весьма распространён в Японии, и в меньшей степени в Европе и на Дальнем Востоке.

Конфигурация CLDC успешно используется в большинстве современных мобильных телефонов и портативных органайзеров. По данным компании Sun Microsystems к концу 2004 года в мире было выпущено более 570 миллионов мобильных устройств с поддержкой этой конфигурации Java. Это делает J2ME доминирующей технологией Java в мире. Объёмы производства мобильных телефонов значительно превышают количество других компьютерных устройств, способных исполнять приложения на Java (например, персональных компьютеров).

3.2.10.4 Платформа .NET

.NET Framework – программная технология от компании Microsoft, предназначенная для создания обычных программ и веб-приложений.

Одной из основных идей Microsoft .NET является совместимость различных служб, написанных на разных языках. Например, служба, написанная на C++ для Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на Delphi; на C# можно написать класс, наследованный от класса, написанного на Visual Basic .NET, а исключение, созданное методом, написанным на C#, может быть перехвачено и обработано в Delphi. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок.

.NET является патентованной технологией корпорации Microsoft. Тем не менее, после заключения договоренности с компанией Novell, технология Mono была признана как реализация .NET на Unix-подобных системах (GNU/Linux, Mac OS X). Однако договорённость касается Novell и клиентов Novell, также технологии ASP.NET, ADO.NET и Windows.Forms не были стандартизированы ECMA/ISO и использование их в Mono находится под угрозой претензий со

стороны Microsoft (претензии возможны только в странах, где существуют патенты на программное обеспечение). Mono предоставляет реализацию ASP.NET, ADO.NET и Windows.Forms, но в то же время рекомендует обходить эти API.

Приложения также можно разрабатывать в текстовом редакторе и использовать консольный компилятор.

Подобно технологии Java, среда разработки .NET создаёт байт-код, предназначенный для исполнения виртуальной машиной. Входной язык этой машины в .NET называется MSIL (Microsoft Intermediate Language), или CIL (Common Intermediate Language, более поздний вариант), или просто IL. Применение байт-кода позволяет получить кроссплатформенность на уровне скомпилированного проекта (в терминах .NET: сборка), а не только на уровне исходного текста, как, например, в C. Перед запуском сборки в среде исполнения CLR байт-код преобразуется встроенным в среду JIT-компилятором в машинные коды целевого процессора. Также существует возможность скомпилировать сборку в родной (native) код для выбранной платформы с помощью поставляемой вместе с .NET Framework утилиты NGen.exe.

3.2.10.5 Язык программирования ADA

Ада (Ada) – язык программирования, созданный в 1979-1980 годах в результате проекта, предпринятого Министерством обороны США с целью разработать единый язык программирования для так называемых встроенных систем (то есть систем управления автоматизированными комплексами, работающими в реальном времени). Имелись в виду, прежде всего, бортовые системы управления военными объектами (кораблями, самолётами, танками, ракетами, снарядами и т. п.). Перед разработчиками не стояло задачи создать универсальный язык, поэтому решения, принятые авторами Ады, нужно воспринимать в контексте особенностей выбранной предметной области.

Ада – это структурный, модульный, объектно-ориентированный язык программирования, содержащий высокоуровневые средства программирования параллельных процессов. Синтаксис Ады унаследован от языков типа Algol или Паскаль, но расширен, а также сделан более строгим и логичным. Ада - язык со строгой типизацией, в нём исключена работа с объектами, не имеющими типов, а автоматические преобразования типов сведены к абсолютному минимуму.

Для удовлетворения требованиям надёжности язык построен таким образом, чтобы как можно большее количество ошибок обнаруживалось на этапе компиляции. Кроме того, одним из требований при разработке языка была максимально лёгкая читаемость текстов программ, даже в ущерб лёгкости написания. Результатом такого подхода стал несколько «тяжеловесный» синтаксис и множество ограничений, часто воспринимаемых профессиональными программистами как «глупые» и «ненужные». Именно это

привело к формированию представления об Аде как о сложном, малопонятном и неудобном в использовании языке. Это представление верно лишь отчасти: написание простой программы на Аде действительно требует больше времени, чем на других, менее формальных языках, типа Си, но отладка и сопровождение программ, особенно крупных и сложных, значительно упрощается. По утверждению Стефена Цейгера, разработка программного обеспечения на Аде в целом обходится на 60 % дешевле, а разработанная программа имеет в 9 раз меньше дефектов, чем при использовании языка Си.

Язык Ада используется в США и Европе в разработке сложных больших проектов, главным образом, встроенных систем, причём далеко не только в военных приложениях.

3.2.10.6 Язык программирования Esterel

Esterel – императивный и параллельный язык, который имеет хорошо определенную формальную базу и законченную реализацию. Фундаментальная концепция ESTEREL – событие. Событие соответствует посылке или получению сигналов, которые передают данные.

Esterel основан на синхронной модели. Этот синхронизм упрощает обоснование времени и гарантирует детерминизм.

Пример программы на языке Esterel:

```
module ABRO:
input A, B, R;
output O;

loop
  [ await A || await B ];
 emit O
each R

end module
```

Приведенная выше программа осуществляет генерацию события ‘O’ только при наличии обоих событий на входах ‘A’ и ‘B’. Сброс программы осуществляется через вход ‘R’.

В настоящее время Esterel используется в системе программирования Esterel Studio (Synfora), предназначенной для проектирования систем-на-кристалле.

3.2.10.7 Язык программирования Lustre

Lustre – декларативный язык программирования, основанный на модели SDF (synchronous dataflow) и предназначенный для программирования реактивных систем. Язык начал разрабатываться в 1984 году, во Франции.

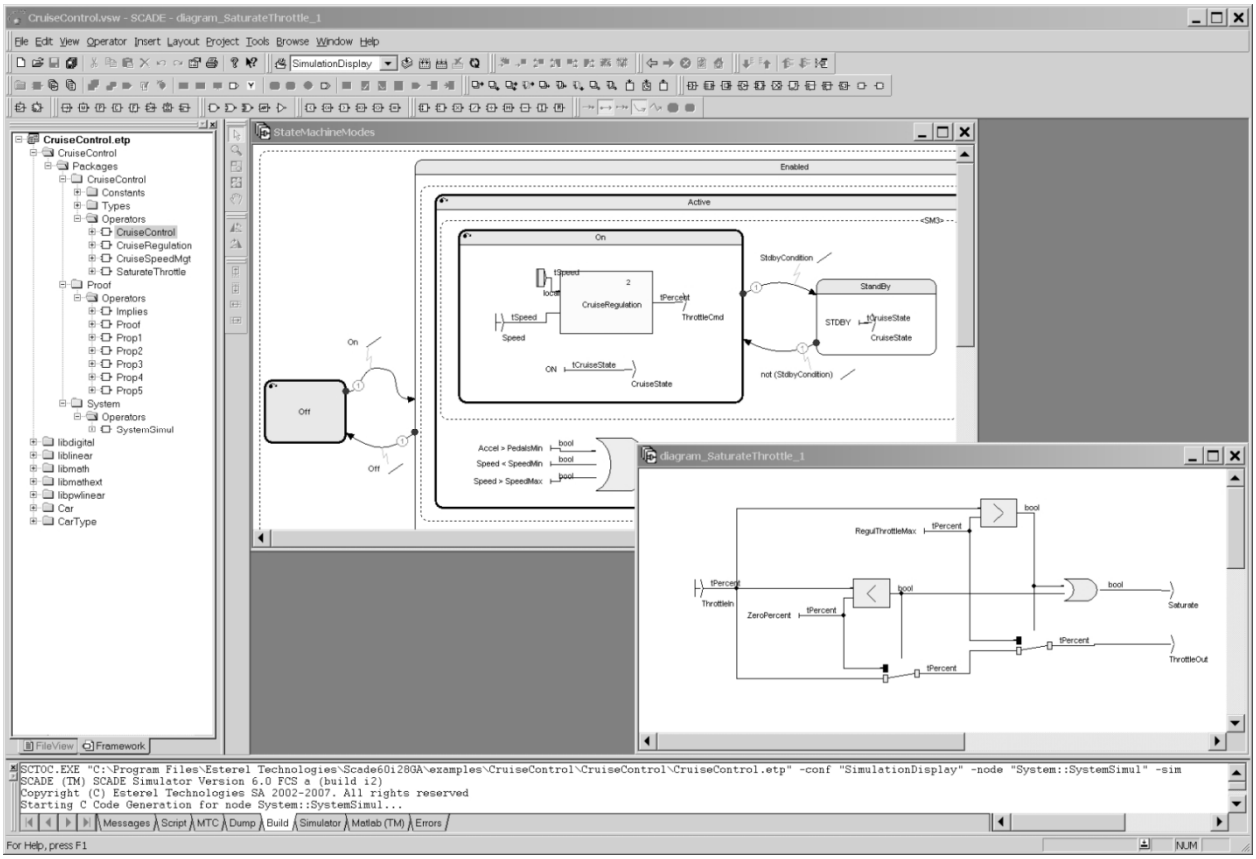


Рисунок 60. Внешний вид системы SCADE, в котрой используется язык Lustre

Язык Lustre активно используется для реализации ответственных приложений в авиакосмической промышленности. На рисунке выше представлен внешний вид системы SCADE, использующей в своей основе этот язык.

3.3 Инструментальные средства отладки и тестирования ВВС

3.3.1 Симулятор

Симулятор – система для полной или частичной имитации поведения и структуры какого-либо объекта. Симулятор относится к инструментальным средствам отладки, тестирования и верификации программных и аппаратных компонент вычислительной системы.

В случае программной реализации симулятор можно исполнять на инструментальной машине. В программировании встроенных систем и СнК чаще всего используется симулятор процессора. Внешне такой симулятор выглядит как обычный отладчик.

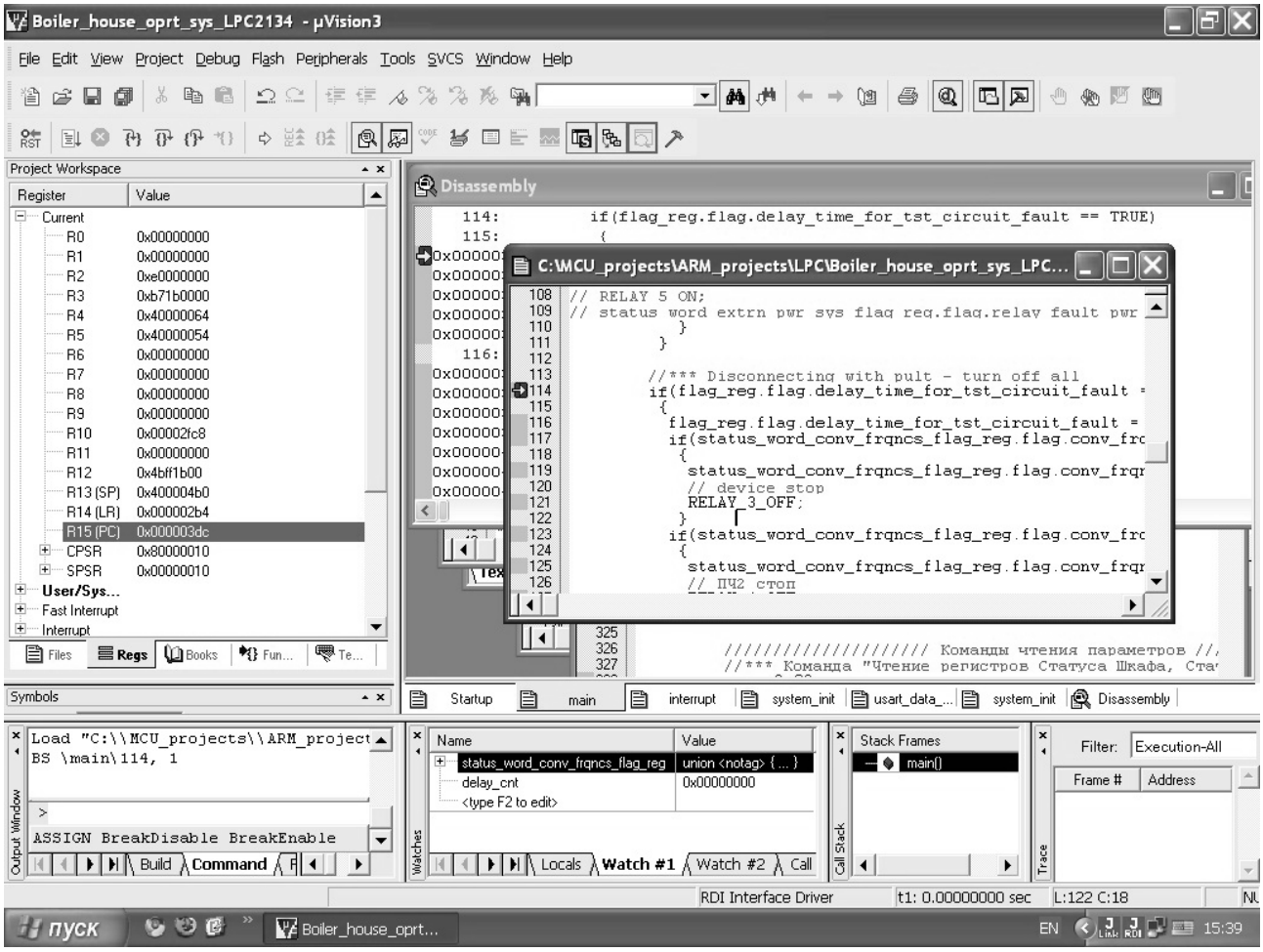


Рисунок 61: Внешний вид симулятора-отладчика Keil Sofrware

При симуляции различных ASIC, процессоров, графических акселераторов используют программно-аппаратные симуляторы, выполненные на базе FPGA. Такой подход позволяет существенно сократить время тестирования и верификации по сравнению с чисто программной симуляцией. Стоимость программно-аппаратного симулятора значительно выше стоимости программного.

К достоинствам симулятора можно отнести возможность моделирования окружающей среды исследуемой системы, что позволяет производить работу в реальном масштабе времени с точки зрения наблюдателя, находящегося внутри исследуемой системы.

К сожалению, в симуляторах всегда присутствует инструментальная погрешность, возникающая из-за неточности моделирования, абстракции, ошибок реализации и так далее.

3.3.2 Внутрисхемный эмулятор

Внутрисхемный эмулятор – это аппаратное устройство, используемое при отладке, обычно выполняемое в форме микропроцессора с дополнительными контактами.

Внутрисхемные эмуляторы подключаются к отлаживаемой или тестируемой системе вместо целевого микропроцессора или микроконтроллера и позволяют гибко управлять поведением системы на протяжении процесса отладки, собирать данные о состоянии ее различных объектов, выполнять программы пользователя в различных режимах: в режиме реального времени (непрерывное выполнение программы с заданного адреса), в пошаговом режиме, в режиме с остановками функционирования по заданному условию. Зачастую они позволяют эмулировать не только целевой процессор, но и память, тактовый генератор, устройства ввода-вывода.

Рисунок 62: Внутрисхемный эмулятор для процессоров фирмы Atmel

Применение внутрисхемных эмуляторов позволяет решить почти все проблемы, связанные с отладкой и тестированием программного обеспечения и аппаратуры. К сожалению, крупным недостатком внутрисхемных эмуляторов является их очень высокая стоимость.

Внутрисхемные эмуляторы являются средством, заменяющим на аппаратном уровне часть целевой системы. В настоящее время распространены два основных варианта:

- эмулятор процессора;
- эмулятор ПЗУ.

В последнее время, с появлением ОКМЭВМ со встроенной постоянной памятью (в виде FLASH или OTP), второй вариант эмуляторов начал постепенно выходить из употребления. В принципе, внутрисхемный эмулятор

имеет тот же набор функций, что и программный симулятор. Приведем основные отличия:

- отладка возможна на реальном оборудовании (что не исключает возможности программной имитации окружения);
- отладка может производиться в реальном времени.

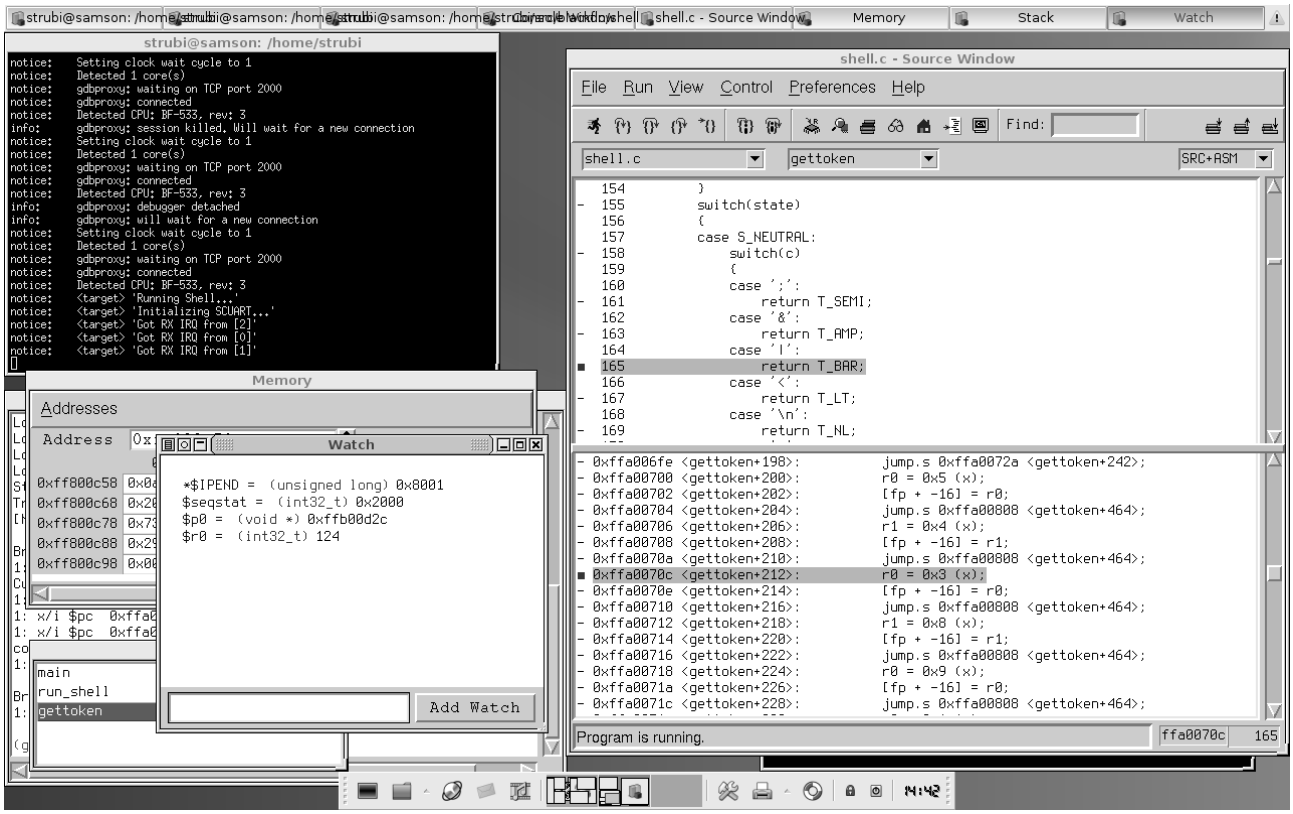


Рисунок 63: Отладка с помощью бесплатного отладчика GDB-Insight

Интересным вариантом внутрисхемной эмуляции является JTAG. В классических эмуляторах процессора для проведения отладки центральный процессор заменяется на эмулирующую головку (у ряда новых процессоров эмулирующую головку можно подключать непосредственно к впаянному кристаллу). Это приводит к необходимости ставить на плату панель под процессор, что уменьшает надежность системы. При использовании технологии JTAG, эмулятор подключается к плате через специальный технологический разъем. При этом процессор не вынимается. JTAG позволяет отключить ядро процессора и управлять шиной адреса, данных и управления напрямую. К сожалению, для управления всеми выводами процессора необходимо передавать через порт JTAG большое количество информации. Поэтому отладка в реальном времени (на частоте работы процессора) невозможна.

Большим достоинством JTAG является аппаратная простота эмулятора. Например, в самом простом случае, достаточно подключить порт JTAG к параллельному LPT порту обычного ПК.

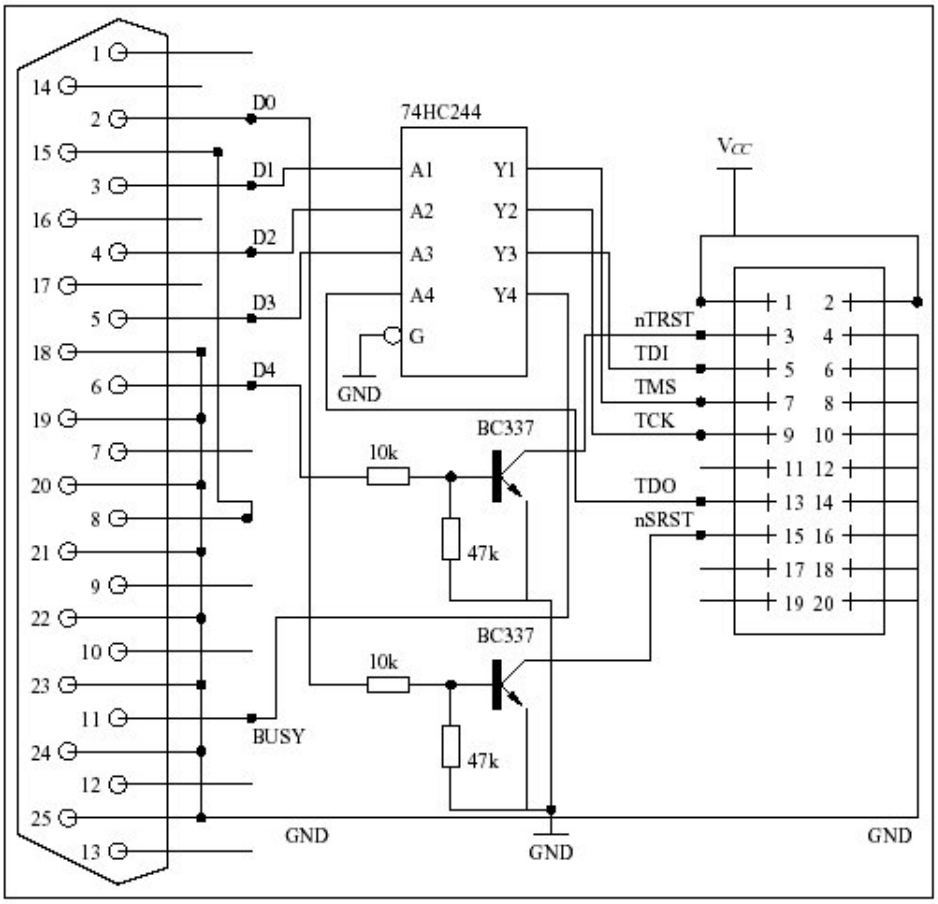


Рисунок 64: Адаптер Wiggler для отладки через JTAG. Схема электрическая принципиальная.

3.3.3 IEEE 1149.1 JTAG - механизм граничного сканирования

JTAG предназначен для решения следующего перечня основных задач:

- начальное тестирование, которое выявляет технологические дефекты изготовления;
- доставка необходимой конфигурации для программируемых компонент;
- поддержка различного рода отладочных механизмов (статических или динамических) и режима мониторинга.
- Гибкость использования механизма граничного сканирования достигается рядом особенностей стандарта, основными из которых являются:

- возможность параллельной согласованной работы нескольких устройств, поддерживающих данный стандарт (количество ограничивается электрическими параметрами интерфейса);
- возможность расширения самого механизма (введение дополнительных команд и форматов данных).

Первое определяет возможность использования механизма граничного сканирования при разработке многопроцессорных систем с гомогенной или гетерогенной структурой. Второе расширяет круг задач, где может эффективно применяться механизм, позволяя решать, например, задачи отладки, сбора диагностики и мониторинга.

3.3.3.1 Реализация JTAG-инструментария

Решая задачу схемотехнического проектирования встроенной системы, проектировщики закладывают средства начального тестирования и внутрисхемной инициализации. Для этого разрабатывается специализированное устройство или сервисный механизм инструментально-технологического характера. Функциональность и состав инструментального обеспечения определяются особенностями проекта. Инструментальные кросс-средства представляют собой совокупность программных средств разработки и аппаратных интерфейсов, которые обеспечивают доступ к целевому объекту с инструментальной машины. Ресурсов инструментальной машины обычно хватает для реализации интерпретатора языкового описания механизма граничного сканирования при проведении большинства работ с макетным образцом системы на этапах частичной инициализации и начальной отладки. Иначе обстоит дело с опытными образцами системы, когда при проектировании стараются сокращать выделенные инструментальные каналы, формируя объединенные сервисные механизмы. При этом начальная инициализация происходит с помощью ранее апробированных вычислительных компонент, которые совмещают в себе целевую функциональность и поддержку инструментального режима. Например, инструментальный канал и канал для поддержки механизма граничного сканирования могут быть объединены. В этом случае размещение интерпретатора на инструментальной машине приведет к снижению эффективности работы из-за ограничений канала. Актуальной становится реализация интерпретатора средствами “сервисного” контроллера в составе целевой системы, который будет выполнять резидентную программу поддержки механизма граничного сканирования. Решение в пользу такой резидентной реализации интерпретатора будет зависеть от вычислительной мощности контроллера и сложности интерпретируемого языка. На сегодняшний день многие производители предоставляют такого рода интерпретаторы в исходных текстах, для различных аппаратных платформ. Типовой вариант JTAG – системы приведен на рис. 5. Эффективность использования технологии граничного сканирования во многом зависит от решения двух вопросов: описания структуры цепочки JTAG и

формулировки алгоритма работы с описанной цепочкой. Под JTAG-цепочкой (scan path) понимается полная стандартная тест-шина, полученная последовательным соединением сигналов TDI и TDO нескольких компонент. Это понятие используется при решении следующих задач:

- описания отдельных микросхем, поддерживающих механизм граничного сканирования;
- описания структуры целевой системы с точки зрения механизма граничного сканирования;
- описания (выделения) цели работы конкретного алгоритма (программирование, тестирование).

Обычно в реальной системе имеется несколько микросхем, связанных в одну JTAG-цепочку. Как отмечалось выше, для описания цепочек и их иерархии используются языки BSDL и HSDL. Как средства описания они эффективны, но практически не поддержаны доступными инструментальными средствами. В результате цепочку JTAG-устройств разработчик задает тем или иным неформальным образом, что ведет к резкому росту трудоемкости при использовании механизма граничного сканирования. Задача выделения цели не входит в число стандартных задач описания структуры JTAG-цепочки. Однако на практике приходится очень редко работать с регистром граничного сканирования BSR (Boundary Scan Register), равным объединению BSR всей цепочки. Обычно для конкретного теста или алгоритма требуется не более десятка ячеек в BSR или только одна микросхема в составе цепочки. Стандартного средства такого описания, по-видимому, не существует. Это объясняется, скорее всего, спецификой описания (по факту) JTAG-цепочки и кругом решаемых JTAG задач. Есть упоминания о средствах программирования, например, flash-памяти, с помощью JTAG, что требует выделения линий адреса, данных и управления во всем BSR. Однако в известных реализациях это делается неформальным образом. Справедливости ради, необходимо отметить, что в стандарте HSDL имеется возможность «выделять» отдельные ячейки BSR, назначая им произвольные имена. Но это нельзя назвать выделением цели в чистом виде, поскольку алгоритм, составленный по такому описанию, по-прежнему будет оперировать полным BSR. Что касается разработчика, то ему при работе также придется рассматривать всю JTAG-цепочку (явным или неявным образом), даже если используется только несколько выводов одной из микросхем.

3.3.4 Измерение производительности программ

Профилировка (профилирование) – измерение производительности как всей программы в целом, так и отдельных ее фрагментов, с целью нахождения «горячих» точек (Hot Spots) – тех участков программы, на выполнение которых расходуется наибольшее количество времени.

Профилировщик (профайлер, profiler) – основной инструмент оптимизатора программ. Программный код ведет себя как в известной

пословице самый медленный верблюд, который определяет скорость каравана, то есть производительность приложения определяется самым узким его участком. Программисты нуждаются в инструментальных средствах, чтобы проанализировать их программы и идентифицировать критические участки программы.

Профилировщики помогают определить, как долго выполняются определенные части программы, как часто они выполняются, или генерировать дерево вызовов (call graph). Типично эта информация используется, чтобы идентифицировать те части программы, которые работают больше всего. Эти трудоёмкие части могут быть оптимизированы, чтобы выполняться быстрее. Это общая методика для отладки.

3.3.4.1 Цели и задачи профилировки

Основная цель профилировки – исследовать характер поведения программы во всех её точках. Под «точкой», в зависимости от степени детализации, может подразумеваться как отдельная машинная команда, так целая конструкция языка высокого уровня (например, функция, цикл или одна–единственная строка исходного текста).

Большинство современных профилировщиков поддерживают следующий набор базовых операций:

- определение общего времени исполнения каждой точки программы (total [spots] timing);
- определение удельного времени исполнения каждой точки программы ([spots] timing);
- определение причины и/или источника конфликтов и пенальти (penalty information);
- определение количества вызовов той или иной точки программы ([spots] count);
- определение степени покрытия программы ([spots] covering).

3.3.4.2 Общее время исполнения

Сведения о времени, которое тратится на выполнение каждой точки программы, позволяют выявить её наиболее «горячие» участки. Правда, здесь необходимо сделать одно уточнение. Непосредственный замер покажет, что, по крайней мере, 99,99% всего времени выполнения профилируемая программа проводит внутри функции main, при этом «горячей» является отнюдь не сама main, а вызываемые ею функции. Чтобы не вызывать у программистов недоумения, профилировщики обычно вычитают время, потраченное на выполнение дочерних функций, из общего времени выполнения каждой функции программы.

3.3.4.3 Удельное время выполнения

Если время выполнения некоторой точки программы не постоянно, а варьируется в тех или иных пределах (например, в зависимости от рода обрабатываемых данных), то трактовка результатов профилировки становится неоднозначной, а сам результат – ненадежным. Для более достоверного анализа требуется: а) определить действительно ли в программе присутствуют подобные «плавающие» точки и, если да, то: б) определить время их исполнения в лучшем, худшем и среднем случаях.

3.3.4.4 Определение количества вызовов

Оценивать температуру точки можно не только по времени ее выполнения, но и частоте вызова. Например, пусть у нас есть две «горячие» точки, в которых процессор проводит одинаковое время, но первая из них вызывается сто раз, а вторая – сто тысяч раз. Нетрудно догадаться, что, оптимизировав последнюю хотя бы на 1%, мы получим колоссальный выигрыш в производительности, в то время как, сократив время выполнения первой из них вдвое, мы ускорим нашу программу всего лишь на четверть.

Таким образом, часто вызываемые функции в большинстве случаев имеет смысл «инлайнить» (от английского in-line), т. е. непосредственно вставить их код в тело вызываемых функций, что сэкономит какое-то количество времени.

3.3.4.5 Определение степени покрытия

Покрытие – это процент реально выполненного кода программы в процессе его профилировки. Такая информация, в первую очередь, нужна тестерам, чтобы убедиться, что весь код программы протестирован целиком и в ней не осталось никаких «темных» мест. С другой стороны, оптимизируя программу, очень важно знать, какие именно ее части были профилированы, а какие нет. В противном случае, многих «горячих» точек можно просто не заметить только потому, что соответствующие им ветки программы вообще ни разу не получили управления.

3.3.5 Анализ исходного кода

Анализатор кода (code analyzer, code reviewing software) – программное обеспечение (ПО), которое помогает обнаружить ошибки (уязвимые места) в исходном коде программы.

Анализ кода (code analysis) – это близкий родственник обзора кода (code review).

Code review – это систематическая проверка исходного кода, проводящаяся для того, чтобы обнаружить ошибки, допущенные на стадии разработки, улучшить качество программы и навыки разработчиков.

Обычно code review включает участие:

- человека, который код писал;
- человека (или людей), которые этот код могут читать и понимать, насколько хорошо он удовлетворяет общим и конкретным критериям.

Общие критерии представляют собой стандарты кодирования (coding standard). Конкретные критерии подразумевают знания требований, для удовлетворения которых код написан.

Процедура анализа кода отличается от тестирования. При тестировании программа проверяется на некотором наборе входных данных с целью выявления несоответствия действительного поведения программы специфицированному. Однако спецификация может определять поведение программы лишь на подмножестве множества всех возможных входных данных. Таким образом, не все ошибки могут быть определены при помощи тестирования. Для этого и нужно проводить анализ кода, который позволяет обнаружить такие ошибки, а точнее уязвимые места исходного кода: переполнение буферов, неинициализированная память, указатели (null-pointer), «утечка» памяти, состояние гонок и др. Примеры такого рода ошибок можно увидеть в статье, посвященной статическому или динамическому анализу кода.

Можно сказать, что анализ исходного кода – это процесс получения информации о программе из ее исходного кода или объектного кода. Исходный код – это статическое, текстовое, удобочитаемое, исполняемое описание компьютерной программы, которое может быть скомпилировано автоматически в исполняемый код.

Анализ исходного кода включает три компонента:

1. Парсер (parser), который выполняет синтаксический разбор исходного кода и преобразует результаты анализа в одну или несколько форм внутреннего представления. Большинство парсеров (синтаксических анализаторов) основываются на компиляторах.
2. Внутреннее представление, которое абстрагирует конкретный аспект программы и представляет его в форме, пригодной для выполнения автоматического анализа. Например, переменные заменяются соответствующими типами данных. Некоторые внутренние представления создаются непосредственно парсерами, другие требуют результатов предыдущего анализа. Классическими примерами таких представлений является граф потоков управления (control-flow graph, CFG), дерево вызовов (call graph), абстрактное синтаксическое дерево (abstract syntax tree, AST), форма статического единственного присваивания (static single assignment, SSA).

3. Анализ внутреннего представления. Анализ может производится по различным направлениям:
    - Статический или динамический. Статический анализ кода (static code analysis) производится без реального выполнения программ. Результаты такого анализа применимы и одинаковы для всех исполнений программы. В отличие от статического анализа, динамический анализ кода (dynamic code analysis) производится при помощи выполнения программ на реальном или виртуальном процессоре. Результаты такого анализа более точные, но гарантируются только для конкретных входных данных. Утилиты динамического анализа могут требовать загрузки специальных библиотек или даже перекомпиляцию программного кода.
    - Контекстно-зависимый (context-sensitive) межпроцедурный анализ.
    - Потоково-зависимый (flow-sensitive) анализ.

Кроме того, анализаторы кода можно классифицировать следующим образом.

- Автоматические анализаторы, которые проверяют исходный код в соответствии с предопределенным набором правил и по результатам проверки создают отчеты.
- Различные виды браузеров, которые визуализируют структуру (архитектуру) ПО, таким образом, помогают лучше ее понять.

Примером ошибок в программе, которые может обнаружить статический анализатор, являются ошибки, связанные с переносом программ на 64-битные системы. В результате чего в приложении могут проявиться новые ошибки. Другие примеры можно найти в различных статьях [36].

При расчете необходимой для массива памяти использовался явно размер типа элементов. На 64-битной системе этот размер изменился, но код остался прежним:

```
size_t ArraySize = N * 4;
intptr_t *Array = (intptr_t *)malloc(ArraySize);
```

Некоторая функция возвращала значение -1 типа size_t в случае ошибки. Проверка результата была записана так:

```
size_t result = func();
if (result == 0xffffffffu) {
// error
}
```

На 64-битной системе значение -1 для этого типа выглядит уже по-другому и проверка не срабатывает.

Арифметика с указателями – постоянный источник проблем. Но в случае с 64-битными приложениями к уже известным добавляются новые проблемы.

Рассмотрим пример:

```
unsigned short a16, b16, c16;
char *pointer;
…
pointer += a16 * b16 * c16;
```

Как видно из примера, указатель никогда не сможет получить приращение больше 4 гигабайт, что хоть и не диагностируется современными компиляторами как ошибка, но приведет в будущем к неработающим программам. Можно привести значительно больше примеров потенциально опасного кода.

Все эти и многие другие ошибки были обнаружены в реальных приложениях во время переноса их на 64-битную платформу.

3.3.5.1 Обеспечение корректности программного кода: обзор существующих решений

Существуют различные подходы к обеспечению корректности кода приложений. Перечислим наиболее распространенные из них: тестирование с помощью юнит-тестов, динамический анализ кода (во время работы приложения), статический анализ кода (анализ исходных текстов). Нельзя сказать, что какой-то один вариант тестирования лучше других – все эти подходы обеспечивают различные аспекты качества приложений.

Юнит-тесты предназначены для быстрой проверки небольших участков кода, например, отдельных функций и классов. Их особенность в том, что эти тесты выполняются быстро и допускают частый запуск. Из этого вытекают два нюанса использования такой технологии. Во-первых, эти тесты должны быть написаны. Во-вторых, тестирование выделения больших объемов памяти (например, более двух гигабайт) занимает значительное время, поэтому нецелесообразно, так как юнит-тесты должны отрабатываться быстро.

Динамические анализаторы кода (лучший представитель – это Compuware BoundsChecker) предназначены для обнаружения ошибок в приложении во время выполнения программы. Из этого принципа работы и вытекает основной недостаток динамического анализатора. Для того чтобы убедиться в корректности программы, необходимо выполнить все возможные ветки кода. Для реальной программы это может быть затруднительно. Но это не значит, что динамический анализ кода не нужен. Такой анализ позволяет обнаружить ошибки, которые зависят от действий пользователя и не могут быть определены по коду приложения.

Статические анализаторы кода (как, например, Gimpel Software PC-lint и Parasoft C++test) предназначены для комплексного обеспечения качества кода и содержат несколько сотен анализируемых правил. В них также есть некоторые из правил, анализирующих корректность 64-битных приложений. Однако, поскольку это анализаторы кода общего назначения, то их использование для

обеспечения качества 64-битных приложений не всегда удобно. Это объясняется, прежде всего, тем, что они не предназначены именно для этой цели. Другим серьезным недостатком является их ориентированность на модель данных, используемую в Unix-системах (LP64). В то время как модель данных, используемая в Windows-системах (LLP64), существенно отличается от нее. Поэтому применение этих статических анализаторов для проверки 64-битных Windows-приложений возможно только после неочевидной дополнительной настройки.

Некоторым дополнительным уровнем проверки кода можно считать наличие в компиляторах специальной диагностики потенциально некорректного кода (например, ключ /Wp64 в компиляторе Microsoft Visual C++). Однако этот ключ позволяет отследить лишь наиболее некорректные конструкции, в то время как многие из также опасных операций он пропускает.

3.3.6 Инструментальные средства отладки ОС РВ eCos

В ОС РВ eCos для отладки программного обеспечения используется Redboot.

Redboot (акроним от «Red Hat Embedded Debug and Bootstrap») – это приложение с открытым исходным кодом, которое использует слой абстракции оборудования (HAL) операционной системы реального времени eCos для загрузки программного обеспечения или прошивки (firmware) во встроенных вычислительных системах. Redboot предоставляется под GPL-совместимой лицензией eCos License.

Redboot предоставляет широкий набор инструментов для загрузки и исполнения программ в целевых встроенных системах, в том числе Embedded Linux и eCos приложений, через последовательный канал или Ethernet соединение, а также инструменты для манипулирования параметрами целевой системы. Redboot также обеспечивает простую файловую систему для модулей флэш-памяти, которая может быть использована для загрузки исполнимых образов. Он может быть использован при разработке продукции (для поддержки отладки), а также для использования в конечной продукции (для загрузки приложений по сети или с флэш-памяти).

Возможности Redboot:

- Поддержка загрузочных скриптов;
- Простой интерфейс командной строки для управления и конфигурирования Redboot, доступный через последовательный канал или Ethernet соединение по протоколу telnet;
- Встроенные заглушки GDB для соединения с отладчиком на хост-машине через последовательный или сетевой Ethernet интерфейс (ограничивается локальной сетью) для отладки приложений на целевой встроенной системе;

- Атрибутная конфигурация – пользовательский контроль и возможность изменения таких аспектов, как системное время и дата (если используется), статический IP адрес и т.д.;
- Конфигурируемый и расширяемый, специально для адаптации под целевую платформу;
- Поддержка загрузки по сети, включая установку и загрузку через BOOTP, DHCP и TFTP;
- Поддержка загрузки программ через последовательный интерфейс посредством протоколов XModem и YModem;
- Самотестирование при запуске.

Redboot может быть использован в качестве общей системы отладки и контроля загрузки программного обеспечения для любых встроенных систем и операционных систем. Например, с соответствующими дополнениями Redboot может заменить широко используемые программы BIOS персональных компьютеров.

3.4 Разработка программного продукта

3.4.1 Жизненный цикл проекта

Жизненный цикл проекта (Project Life Cycle) – последовательность фаз проекта, задаваемая исходя из потребностей управления проектом.

В рамках методологии Института управления проектами (Project Management Institute) жизненный цикл проекта имеет 5 фаз:

- Инициация;
- Планирование;
- Выполнение;
- Контроль и мониторинг;
- Завершение.

При моделировании по принципу "водопада" работа над проектом движется линейно через ряд фаз, таких как:

- анализ требований (исследование среды);
- проектирование;
- разработка и реализация подпроектов;
- проверка подпроектов;
- проверка проекта в целом.

Недостатками такого подхода являются накопление возможных на ранних этапах ошибок к моменту окончания проекта и, как следствие, возрастание риска провала проекта, увеличение стоимости проекта.

Итеративный подход – выполнение работ параллельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы. Проект при этом подходе в каждой фазе развития проходит повторяющийся цикл: Планирование - Реализация - Проверка - Оценка (plan-do-check-act cycle).

Преимущества итеративного подхода:

- снижение воздействия серьезных рисков на ранних стадиях проекта, что ведет к минимизации затрат на их устранение;
- организация эффективной обратной связи проектной команды с потребителем (а также заказчиками, стейкхолдерами) и создание продукта, реально отвечающего его потребностям;
- акцент усилий на наиболее важные и критичные направления проекта;
- непрерывное итеративное тестирование, позволяющее оценить успешность всего проекта в целом;
- раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта;
- более равномерная загрузка участников проекта;
- эффективное использование накопленного опыта;
- реальная оценка текущего состояния проекта и, как следствие, большая уверенность заказчиков и непосредственных участников в его успешном завершении.

Пример реализации итеративного подхода – методология разработки программного обеспечения, созданная компанией Rational Software.

В модели Боэма рассматривается зависимость эффективности проекта от его стоимости с течением времени. На каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка.

Моделирование жизненного цикла проекта инкрементным методом (прототипный жизненный цикл) предусматривает разработку последовательности всё более усложняющихся прототипов системы [40].

3.4.2 Общие проблемы проектирования

Ниже мы приведем некоторые фрагменты из книги Брукса «Мифический человеко-месяц или как создаются программные системы» [27].

3.4.2.1 Сложность проектирования и разработчики- одиночки

Что лучше, работать одному или в составе рабочей группы?

В жизни часто встречаются умельцы, заявляющие, что они смогут быстро сделать продукт за «50 баксов». В принципе, они наверно правы (тем более, что

есть положительные примеры таких продуктов), но существует ряд проблем, хорошо описанных Бруксом в книге «Мифический человеко-месяц»:

- Программа может быть создана одним программистом;
- Для создания комплекса (нескольких взаимосвязанных программ) понадобится усилий в 3 раза больше;
- Для создания продукта (хорошо отлаженной и задокументированной программы) понадобится усилий в 3 раза больше.

Итак, получается для создания комплексного программного продукта понадобится в 9 раз больше усилий. Этот факт нужно учитывать, именно эти вещи отличают продукт от поделки, сделанной за вечер на коленке. На самом деле, слово программа можно заменить словом изделие или встроенная система. Суть от этого не изменится. Уже давно замечено, что идеи, изложенные Бруксом в его книге, весьма универсальны и применимы в различных областях.

Да, безусловно, производительность работы рабочей группы может быть ниже, чем производительность одного разработчика. В чем выигрыш? В параллелизме, то есть в возможности распарраллеливании работ.

3.4.2.2 Оценка времени проектирования

Почти все программистские проекты страдают скорее из-за нехватки времени, нежели из-за отсутствия каких-либо других ресурсов. Почему эта причина бедствий является столь всеобщей?

Во-первых, наши методы оценки весьма несовершенны. Строго говоря, они отражают некоторое неявно высказываемое и в корне неверное допущение, что все будет идти хорошо.

Во-вторых; наши методы оценки ошибочно путают усилия с достижениями, прячась за допущение, что человек и месяц взаимозаменяемы.

В-третьих, отсутствие уверенности в наших оценках ведет к отсутствию у руководителей программистских проектов вежливого упрямства, свойственного шеф-повару ресторана "Аптуан".

В-четвертых, управление ходом разработки плохо организовано. Методы, давно опробованные и даже рутинные в других технических дисциплинах, в технологии программирования рассматриваются как радикальные новшества.

В-пятых, когда обнаруживается отставание от графика, естественная (и традиционная) реакция руководителя - добавить рабочей силы. А это, аналогично попытке заливать огонь бензином,- только ухудшает дело, причем значительно. Чем сильнее огонь, тем больше требуется бензина, круг замыкается, и последствия плачевны.

Все программисты — оптимисты. Исходя из этого, необходимо с осторожностью относиться к их прогнозам.

Стоимость проекта действительно зависит от числа людей и от числа месяцев, но его успешность - нет. Следовательно, человеко-месяц как единица измерения объема работы является опасным и вводящим в заблуждение мифом. Этот миф основывается на предпосылке, что люди и месяцы взаимозаменяемы.

Ни одна часть графика работ не связана так сильно ограничениями на их последовательность, как отладка компонент и комплексная отладка. Очевидно, что требуемое время зависит от числа встречаемых ошибок и легкости их обнаружения. Будучи оптимистами, мы обычно ожидаем, что ошибок будет меньше, чем это оказывается в действительности. Именно поэтому отладка чаще всего не укладывается в график.

3.4.2.3 Использование новых технологий

После провала проекта очень хочется найти какую-либо новую технологию, которая позволит быстро и радикально повысить скорость работы команды разработчиков. К сожалению, процесс увеличения производительности требует значительных затрат и на практике происходит значительно медленнее, чем хотелось бы. Фредерик Брукс называет такую «волшебную технологию» серебряной пулей и пишет по этому поводу следующее: «*Серебряных пуль не только не видно в настоящее время, но в силу самой природы программного обеспечения маловероятно, что они вообще будут найдены - не будет изобретений, способных повлиять на продуктивность создания, надежность и простоту программного обеспечения так, как электроника, транзисторы и интегральные схемы - на аппаратное обеспечение компьютеров*».

3.4.3 Повторное использование

Без применения каких-либо методик повторного использования распространение заготовок за пределы рабочей группы практически невозможно.

Между рабочими группами всегда существует некий "потенциальный барьер", препятствующий распространению полезной информации. Барьер формируется, как правило, из-за нежелания сотрудничать, различных уровней подготовки сотрудников, разных стилей руководства, отличных друг от друга подходов к проектированию и т.д. Проникновение информации через барьер возможно только при наличии достаточно большой энергии, прикладываемой с обеих сторон для организации совместной деятельности.

Практика показывает, что при определенной сложности компонента сделать такой же значительно проще с нуля, чем перенять чужой опыт. Почему это происходит?

- Исходные тексты могут ответить на вопрос "как это сделано", но они никогда не ответят на вопрос "почему это именно так сделано".
- Сколько-нибудь сложная система требует наличия проектной документации, чтобы в ней можно было разобраться и поддержки со стороны разработчиков, чтобы можно было узнать то, что непонятно.
- Необходимо понимание общей концепции системы для того, чтобы можно было понять частности.

К сожалению, в низкобюджетных разработках не делают сколько-нибудь серьезной проектной документации из-за недостатка времени, исходные тексты, как правило, делаются без комментариев, стиль кодирования оставляет желать лучшего. Информация о концепции не изложена на бумаге, а хранится только в одном месте - в голове ведущего разработчика или архитектора проекта. Общая занятость коллектива не позволяет тратить время на объяснения тонкости работы какого-либо компонента другой рабочей группе.

Перечисленные проблемы решаются, если увеличить бюджет проекта. В ряде случаев такое экстенсивное решение может сделать проект нерентабельным. К примеру, требуется сделать уникальное устройство со сравнительно небольшим тиражом и небольшим бюджетом разработки. Выполнение такой работы крупной фирмой невозможно из-за плохой рентабельности, а мелкая фирма может не справиться с работой из-за проблем с повторным использованием [40].

3.4.4 Информация для будущих руководителей

Ниже, мы позволим себе привести несколько цитат из книги Де Марко «Deadline. Роман об управлении проектами», которые могут быть Вам полезны, как инженерам и будущим руководителям проектов. Основная мысль этой книги состоит в том, что главное в любом проекте это *люди*. Данная книга написана в виде художественного произведения и в конце глав делаются некоторые выводы, которые и приведены ниже. Если вам по какой-либо причине не нравится такое изложение, существует более серьёзная книга того же автора на аналогичную тему: «Человеческий фактор. Успешные проекты и команды» [32, 33].

3.4.4.1 Безопасность и перемены

1. Если человек не чувствует, что находится в безопасности, он будет противиться переменам.
2. Перемены необходимы руководителю для успешной работы (наверняка они необходимы и в любой другой деятельности).
3. Неуверенность заставляет человека избегать риска.

4. Избегая риска, человек упускает все новые возможности и выгоды, которые могли бы принести ему перемены.
5. Человека легко запугать прямыми угрозами, но также можно просто дать ему понять, что при случае с ним могут обойтись грубо и жестоко. Эффект будет таким же.

3.4.4.2 Отрицательная мотивация

1. Угрозы – самый неподходящий вид мотивации, если вас волнует производительность сотрудников.
2. Чем бы вы ни угрожали, задача все равно не будет выполнена, если с самого начала вы отвели на ее выполнение слишком мало времени.
3. Более того, если люди не справятся, вам придется выполнить свои обещания.

3.4.4.3 Части тела, необходимые для управления проектами

Для руководства нужны сердце, нутро, душа и нюх.

Так что:

- Руководить надо сердцем;
- Чувствовать нутром;
- Вкладывать в команду и проект душу;
- Иметь нюх на всякую ерунду и бессмыслицу.

3.4.4.4 Повышение производительности

1. Не существует никаких краткосрочных мер, которые позволили бы быстро повысить производительность роботы.
2. Повышение производительности – результат долгосрочных усилий.
3. Любые средства для повышения производительности, которые обещают немедленный результат, на самом деле не что иное, как «птичье молоко».

3.4.4.5 Управление рисками

1. Чтобы управлять проектом, достаточно управлять его рисками.
2. Создайте список рисков для каждого проекта.
3. Отслеживайте те риски, которые являются причиной провала проекта, а не только конечные риски.
4. Оцените вероятность возникновения и стоимость каждого риска.
5. Для каждого риска определите показатель – симптом, по которому можно определить, что риск превращается в проблему.

6. Назначьте специального человека для управления рисками, и пусть он не поддерживает девиз «Мы можем все!», который культивирует начальство.
7. Создайте доступные (возможно, анонимные) каналы для сообщения плохих новостей руководству.

3.4.4.6 Играй в защите

1. Сокращайте потери.
2. Успех проекта можно скорее обеспечить сокращением ненужных усилий, чем стремлением к новым победам.
3. Чем раньше вы прекратите ненужную работу, тем лучше для всего проекта.
4. Не пытайтесь создавать новые команды без необходимости; поищите в коллективе уже сложившиеся и сработавшиеся команды.
5. Оставляйте команды работать вместе и после окончания проекта (если они сами того хотят), чтобы у пришедших вам на смену руководителей было меньше проблем с плохо срабатывающимися командами.
6. Считайте, что команда, которая хочет продолжать работать вместе и дальше, - это одна из основных целей любого проекта.
7. День, который мы теряем в начале проекта, значит так же много, как и день, потерянный в конце.
8. Есть тысяча и один способ потратить день зря и ни одного, чтобы вернуть этот день обратно.

3.4.4.7 Сбор метрических данных

1. Определяйте размер каждого проекта.
2. Не усердствуйте поначалу с выбором единицы измерения – если впоследствии вам предстоит работать с реальными данными, для начала сойдут и абстрактные единицы.
3. Стройте сложные метрики на основе простых (тех, которые легко подсчитать в любом программном продукте).
4. Собирайте архивные данные, чтобы считать производительность труда по уже законченным проектам.
5. Работайте над формулами вычисления сложных синтетических метрик до тех пор, пока полученные результаты не будут наиболее точно отражать отношение абстрактных единиц к указанному в архивных данных объему работ.
6. Проведите через всю архивную базу данных линию тренда, которая будет показывать ожидаемый объем работ в виде отношения значений сложных синтетических метрик.

7. Теперь для каждого нового проекта достаточно будет высчитать значение синтетической метрики и использовать ее при определении ожидаемого объема работ.
8. Не забывайте об «уровне помех» на линии производительности и используйте его, как индикатор при определении допустимых отклонений от общей траектории.

3.4.4.8 Что дает давление сверху

1. Люди не начнут быстрее соображать, если руководство будет давить на них.
2. Чем больше сверхурочной работы, тем ниже производительность.
3. Немного давления и сверхурочной работы могут помочь сконцентрироваться на проблеме, понять и почувствовать ее важность, но длительное давление всегда плохо.
4. Возможно, руководство так любит применять давление, потому что просто не знает, как еще можно повлиять на ситуацию, или же потому, что альтернативные решения кажутся им слишком сложными.
5. Ужасная догадка: давление и сверхурочная работа призваны решить только одну проблему – сохранить хорошую мину при плохой игре.

3.4.4.9 Сердитый начальник

1. Злость и неуважение заразительны. Когда высшее руководство демонстрирует злость и неуважение к подчиненным, руководители среднего звена начинают копировать их поведение. Точно так же дети, которых наказывали в детстве, часто впоследствии становятся жестокими родителями.
2. Неуважение и злоба, по мнению некоторых начальников, должны заставить подчиненных лучше работать. Это типичная политика «кнута и пряника». Но разве когда-нибудь неуважение со стороны начальства приводило к тому, что люди начинали лучше работать?
3. Если начальник демонстрирует неуважение к подчиненным, это признак того, что он не может больше занимать свою должность (а вовсе не признак того, что у него плохие подчиненные).

3.4.4.10 Туманные спецификации

1. Неясность спецификации говорит о том, что между участниками проекта есть неразрешенные конфликты.
2. Спецификация, в которой нет списка типов входящей и исходящей информации, не должна даже приниматься к рассмотрению. Это значит, что она попросту ничего не специфицирует.

3. Никто никогда не скажет вам, что спецификация плоха. Люди скорее будут обвинять себя в неспособности понять написанное, чем ругать авторов спецификации.

3.4.4.11 Конфликт

1. Проект, в котором участвуют несколько сторон, обязательно столкнется с конфликтом интересов.
2. Процесс создания и распространения программных систем – прямо-таки рассадник всевозможных конфликтов.
3. В большинстве компаний, где создается программное обеспечение, никто специально не занимается вопросом решения конфликтов.
4. Конфликт заслуживает понимания и уважительного отношения. Конфликт не имеет ничего общего с непрофессиональным поведением.
5. Донесите до каждого, что постараетесь учитывать интересы всех участников, и проследите, чтобы так оно и было.
6. Тяжело договариваться. Гораздо легче выступать посредником.
7. Объявите заранее, что если интересы конфликтующих сторон полностью или частично противоположны, то поиск решения будет переложен на посредника.
8. Не забывайте: мы находимся по одну сторону баррикад. По другую сторону находится сама проблема.

3.4.4.12 Кто такой катализатор проекта

1. Существуют люди-катализаторы. Они помогают создать здоровую команду, отношения, боевой дух. Даже если бы они больше ничего не делали (а как правило, они делают куда как много), их роль в проекте остается одной из наиболее важных.
2. Посредничество – еще одна сфера, в которой люди-катализаторы просто незаменимы. Впрочем, посредничеству можно научиться, это не очень сложно.
3. Первым шагом к посредничеству должна быть маленькая церемония. Например, можно произнести фразу: «Вы позволите мне попробовать помочь вам решить этот спор?»

3.4.4.13 Человеку свойственно ошибаться

Нам кажется, что самое страшное – не знать чего-то. На самом деле гораздо хуже быть уверенным, что знаешь, когда на самом деле это не так.

3.4.4.14 О персонале

1. Если в самом начале проект делает большая команда, это снижает эффективность самой ответственной части работы - определения

архитектуры системы (потому что всем разработчиком надо скорее дать какую-то работу).
2. Если работу раздать людям и командам еще до того, как завершится стадия дизайна продукта, не получится создать простые и эффективные модели взаимодействия между людьми и рабочими группами.
3. Это приведет к потере независимости, увеличению числа собраний и совещаний, общему недовольству.
4. В идеале было бы хорошо сначала набрать маленькую команду, которая создала бы продуманную архитектуру системы, а уже потом, на последнюю, шестую часть времени разработки в эту команду можно было бы добавить новый персонал (который работал бы непосредственно над кодированием).
5. Ужасное предположение: кажется, те команды, перед которыми не ставят жестких сроков, заканчивают работу быстрее!

3.4.4.15 Проблемы социологии

1. Собрания должны быть небольшими. Для этого нужно сделать так, чтобы люди не боялись пропускать ненужные им собрания. Самый простой способ – заранее опубликовать повестку дня, а потом всегда строго ее придерживаться.
2. Каждому проекту нужна какая-то церемония или ритуал.
3. С помощью церемоний можно концентрировать внимание собравшихся на основных целях и задачах совещания: сократить состав рабочей группы, повысить качество программного кода и т.п.
4. Защищайте людей от оскорблений и ругани Начальства.
5. Запомните: в работе страх = гнев. Те руководители, которые любят кричать на своих подчиненных и всячески унижают и оскорбляют их, на самом деле просто чего-то очень боятся.
6. Наблюдение: если бы для всех проявление грубости и злости к подчиненным всегда значило бы, что начальник просто боится, то никто из начальников не стал бы так себя вести просто из страха, что его страх станет заметен! (Это, конечно, не решает проблем такого руководителя, но, по крайней мере, оберегает его подчиненных.)

3.4.4.16 О патологической политике (еще раз)

1. Эту патологию невозможно вылечить снизу.
2. Не стоит терять время или подвергать себя опасности, чтобы проверить предыдущий постулат на собственном опыте.
3. Иногда единственным выходом из ситуации становится выжидание. Попробуйте подождать, пока проблема не разрешится сама по себе или пока вы не найдете способ уйти от нее в сторону.
4. Чудеса, конечно, случаются, но лучше все же на них не рассчитывать.

3.4.4.17 Злоба и скупость

1. Злоба и скупость – вот формула, которую начинают применять в плохих компаниях те, кто несет ответственность за неудачи в бизнесе.
2. Злоба и скупость прямо противоположны истинным целям любой хорошей компании – быть щедрыми и заботливыми по отношении к своим сотрудникам.
3. Когда вы подмечаете в компании проявления злобы и скупости, знайте, их настоящая причина – страх и боязнь провала.

3.4.4.18 Основы здравого смысла

1. У проекта должно быть два срока сдачи – запланированный и желаемый.
2. Эти сроки должны быть разными.

3.4.5 Особенности проектирования встроенных систем

В целом, проектирование встроенных систем мало отличается от проектирования сложных, программно-аппаратных комплексов. Основные отличия в следующем:

1. Встроенная система – это не программа и не аппаратура по отдельности. Это сложное устройство, состоящее из аппаратной и программной части, помещенное в корпус. Во встроенной системе всё взаимосвязано.
2. Встроенная система должна уметь работать с реальными объектами управления, с объектами реального мира, в реальном масштабе времени. Встроенная система должна уметь противоборствовать агрессивному окружению (температура, давление, влажность, вибрации, излучения).
3. Встроенная система должна быть надёжна, так как чинить ее, в случае поломки, как правило, будет некому. Основная часть встроенных систем не имеет даже интерфейса пользователя и вмонтированы они в самые недра различных установок.
4. В процессе проектирования встроенной системы используется специфическая элементная база и инструментальные средства.

Исходя из этого, на процесс проектирования накладываются следующие ограничения:

1. Проектирование встроенной системы производится как проектирование комплексной системы.
2. В процессе проектирования учитываются различные аспекты, в том числе такие как:

   а) Энергопотребление;

b) Температурный диапазон;
c) Устойчивость к вибрациям.

3. В процессе проектирования встроенной системы есть точка невозврата, появляющаяся при фиксировании выбора аппаратной (неизменяемой) части.

4 Устройство современного контроллера на примере SDK-1.1

Данная глава посвящена архитектуре учебно-лабораторного стенда SDK-1.1, который является примером простейшего контроллера встраиваемых систем. На этом примере демонстрируется реализация тех базовых технических приемов по устройству встраиваемых систем, которые обсуждались в предыдущих главах.

4.1 Назначение стенда

Учебный стенд представляет собой микропроцессорный контроллер, построенный на базе однокристальной микро-ЭВМ ADuC812 [1] (SDK-1.1/S на ADuC842) и имеющий в своем составе разнообразные, типичные для современных встроенных систем, устройства, предназначенные для ввода, обработки и вывода информации в цифровом и аналоговом виде. SDK-1.1 можно применять в качестве аппаратной базы для обучения основам современной микропроцессорной техники и программируемой логики в университетах, колледжах, физико-математических школах и на предприятиях.

Необходимость в подобных стендах возникает из-за того, что современные персональные компьютеры, к сожалению, достаточно плохо позволяют демонстрировать студентам все тонкости организации вычислительного процесса. Во-первых, вся «начинка» современного компьютера скрыта от пользователя операционной системой. Во-вторых, аппаратная база компьютеров часто меняется, что сильно затрудняет поддержку учебных материалов в актуальном состоянии. В-третьих, аппаратура современных компьютеров общего назначения весьма сложна и на подробное её изучение может просто не хватить времени, отведенного на курс учебной программой.

Учебный стенд SDK-1.1 позволяет изучить основные принципы функционирования вычислительной машины, заостряя внимание студентов на самых важных моментах, не отвлекаясь на моменты второстепенные [50].

4.2 Состав стенда

В состав учебного стенда SDK-1.1 входят:

1. Микроконтроллер ADuC812 (Analog Devices), 8 Кб FLASH, 256 байт ОЗУ, 640 байт EEPROM.
2. Внешнее ОЗУ 128 Кб (с возможностью расширения до 512 Кб), подключение к МК ADuC812 по системной шине; используется для хранения пользовательских программ и данных.
3. Расширитель портов ввода–вывода – ПЛИС MAX3064 (Altera), подключение к МК ADuC812 по системной шине.
4. Внешняя EEPROM–память 256 байт, подключение к МК ADuC812 по интерфейсу I2C.

5. Часы реального времени – PCF8583 (Philips), подключение по интерфейсу I2C.
6. Консоль оператора (подключение через ПЛИС к МК ADuC812):
   - Символьный жидкокристаллический индикатор (ЖКИ) WH1602B-YGK-CP (Winstar Display), 16 * 2
   - Матричная клавиатура, 4 * 4
   - Звуковой излучатель – 1 шт.
   - Управляемые светодиоды – 8 шт.
   - Ручные переключатели тестовых сигналов для аналоговых и дискретных портов ввода: коммутатор аналоговых каналов (подключен напрямую к МК ADuC812) и стимулятор дискретных портов.
7. Интерфейсы:
   - Оптически развязанный приемопередатчик инструментального канала RS–232C (для связи с персональным компьютером).
   - Интерфейс JTAG (IEEE 1149.1) для контроля периферийной шины и портов, реализованных в ПЛИС MAX3064.

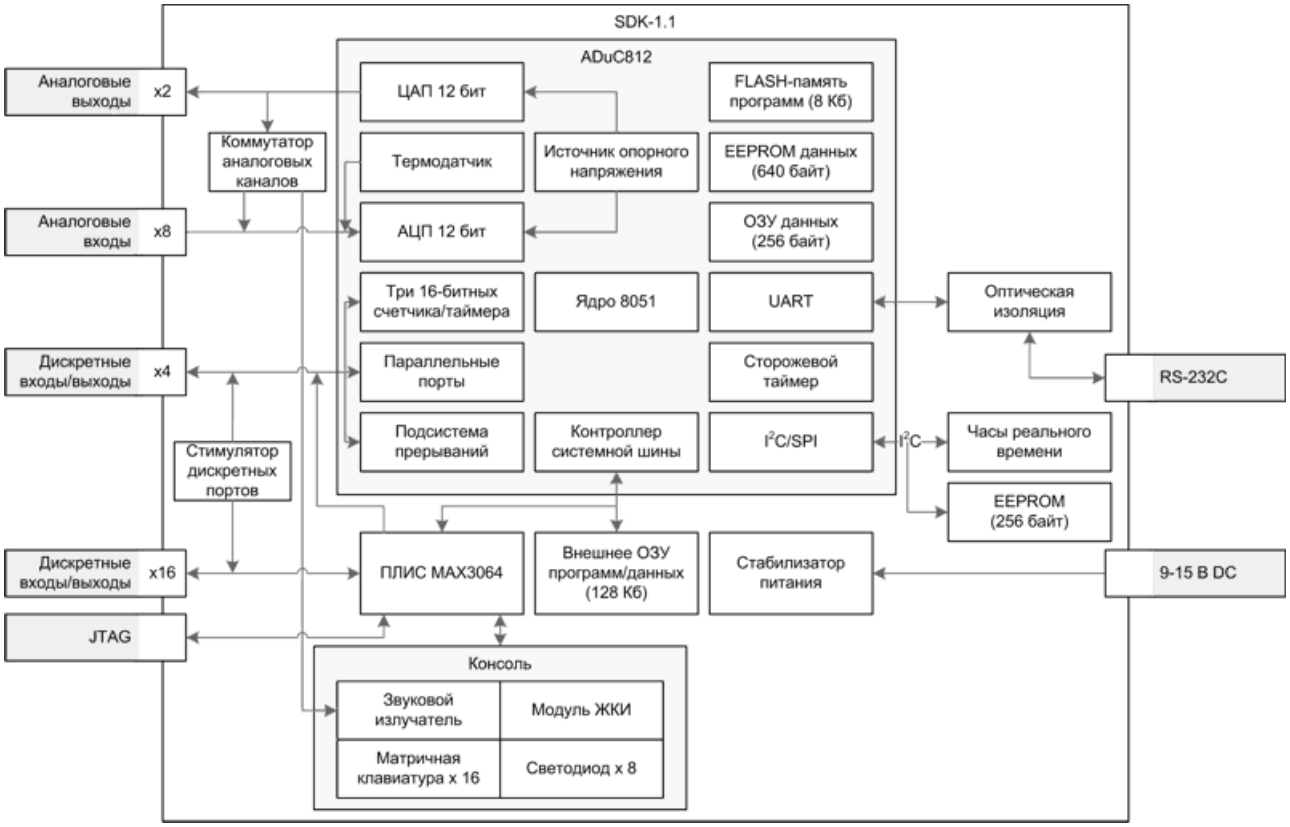


Рисунок 65. Структура аппаратной части учебного стенда SDK-1.1

4.3 Разъемы стенда и назначение выводов

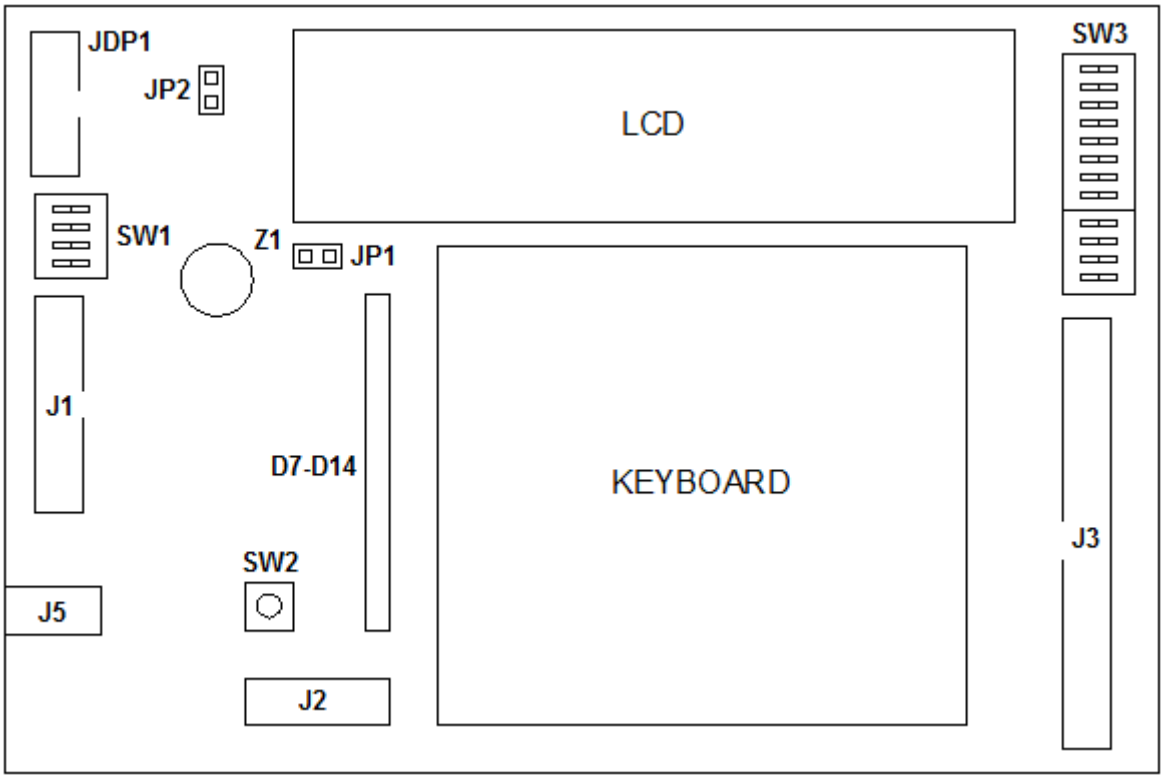


Рисунок 66. Схематическое изображение стенда SDK-1.1.

На рисунке представлено схематическое изображение лицевой панели стенда SDK-1.1. Расшифровка обозначений на схеме дана в таблице.

Таблица 9. Расшифровка обозначений на схеме лицевой панели стенда SDK-1.1.

Элемент	Описание
LCD	Жидкокристаллический индикатор WH1602B-YGK-CP.
KEYBOARD	Матричная клавиатура AK1604A-WWB.
Z1	Звуковой пьезокерамический излучатель.
SW2	Кнопка сброса RESET.
J5	Разъем питания стенда 10-14 В типа «JACK», полярность безразлична.
JDP1	Разъем последовательного порта стенда.
J1	Выводы каналов АЦП и ЦАП.
SW1	Переключатель, замыкающий каналы 0 и/или 1 ЦАП на входы соответствующих (0, 1) каналов АЦП.
J3	16 линий параллельного порта ПЛИС MAX и 4 линии параллельного порта P3 микроконтроллера ADuC812 (INT0/1, T0/1).
SW3	Набор переключателей, замыкающих соответствующие выводы J3 на корпус (переключение в лог. «0»).
J2	Выводы JTAG-интерфейса ПЛИС MAX.
JP1	Перемычка, замыкающая вывод PSEN микроконтроллера ADuC812 на корпус.
JP2	Разъемы подключения внешней батареи питания часов реального времени PCF8583.
D7-D14	Набор сигнальных светодиодов.

Перемычка JP1. Перемычка предназначена для замыкания вывода PSEN микроконтроллера ADuC812 через резистор 1 КОм на корпус. По сигналу RESET или при включении питания микроконтроллер ADuC812 анализирует состояние этого вывода и если он находится в лог. «0» (перемычка замкнута), то запускается встроенная в микроконтроллер процедура перезаписи внутренней Flash-памяти.

Разъем JP2. Разъем предназначен для подключения внешней батареи питания + 5 В часов реального времени PCF8583. Если батарея не подключена, питание часов осуществляется через ионистор емкостью 0.1 ф. Назначение выводов относительно надписей на плате представлено на рисунке.

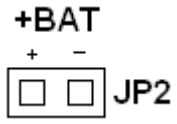


Рисунок 67. Разъем JP2: назначение выводов.

Разъем JDP1. Этот разъем предназначен для подключения кабеля асинхронного последовательного интерфейса, связывающего стенд с COM-портом персонального компьютера. Назначение выводов представлено на рисунке.

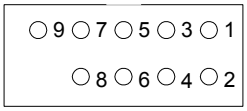


Рисунок 68. Выводы разъема JDP1.

К разъему подключается коммуникационный кабель RS-232, на одном конце которого находится стандартный для персонального компьютера разъем DB25F, а на другом – IDC10F. Распайка кабеля представлена на рисунке.

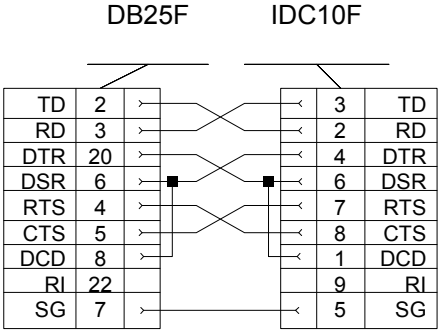


Рисунок 69. Распайка кабеля.

Разъем J2. Разъем предназначен для программирования ПЛИС MAX3064 (MAX3128) через интерфейс JTAG (IEEE 1149.1). Нумерация выводов относительно надписи «J2» на плате показана на рисунке.

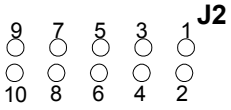


Рисунок 70. Нумерация выводов разъема J2.

Таблица 10. Выводы разъема J2.

№ вывода	Описание (JTAG)
1	TCK
2	Корпус
3	TDO
4	+ 5 V
5	TMS
6	-
7	-
8	#RESET
9	TDI
10	Корпус

Разъем J1. Разъем представляет собой набор входов восьмиканального АЦП и выводов двухканального ЦАП микроконтроллера ADuC812.

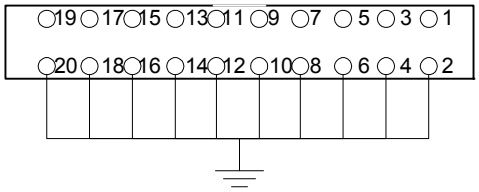


Рисунок 71. Разъем J1.

Таблица 11. Назначение выводов разъема J1.

№ вывода	Описание
1	Канал 0 ЦАП
3	Канал 1 ЦАП
5	Вход канала 0 АЦП
7	Вход канала 1 АЦП
9	Вход канала 2 АЦП
11	Вход канала 3 АЦП
13	Вход канала 4 АЦП
15	Вход канала 5 АЦП
17	Вход канала 6 АЦП
19	Вход канала 7 АЦП
Четные 2-20	Корпус

Напряжение, подаваемое на входы АЦП, делится на два при поступлении на соответствующие выводы микроконтроллера ADuC812 (ADC0-ADC7).

На панели стенда смонтирован переключатель SW1, замыкающий соответственно канал 0 ЦАП на вход канала 0 АЦП (перемычка «1» в положении ON) и канал 1 ЦАП на вход канала 1 АЦП (перемычка «2» в положении ON).

Внимание! Замыкание каналов ЦАП на корпус при ненулевом напряжении на них может привести к выходу микроконтроллера ADuC812 из строя.

Разъем J3. Разъем представляет собой выводы параллельного порта ПЛИС MAX 3064 (MAX3128) и 4 линии порта P3 микроконтроллера ADuC812. Нумерация выводов представлена на рисунке.

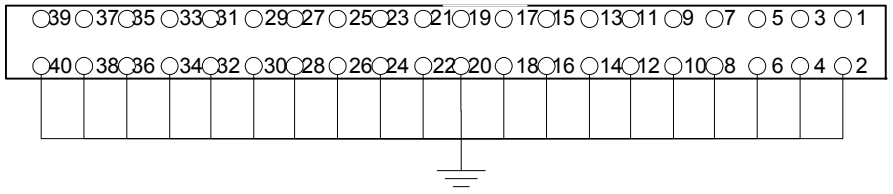


Рисунок 72. Нумерация выводов разъема J3.

Таблица 12. Назначение выводов разъема J3.

№ вывода	Описание
1	Вход INT0 ADuC812.
3	Вход INT1 ADuC812.
5	Вход T0 ADuC812.
7	Вход T1 ADuC812.
9	Вывод линии 0 параллельного порта ПЛИС MAX.
11	Вывод линии 1 параллельного порта ПЛИС MAX.
13	Вывод линии 2 параллельного порта ПЛИС MAX.
15	Вывод линии 3 параллельного порта ПЛИС MAX.
17	Вывод линии 4 параллельного порта ПЛИС MAX.
19	Вывод линии 5 параллельного порта ПЛИС MAX.
21	Вывод линии 6 параллельного порта ПЛИС MAX.
23	Вывод линии 7 параллельного порта ПЛИС MAX.
25	Вывод линии 8 параллельного порта ПЛИС MAX.
27	Вывод линии 9 параллельного порта ПЛИС MAX.
29	Вывод линии 10 параллельного порта ПЛИС MAX.
31	Вывод линии 11 параллельного порта ПЛИС MAX.
33	Вывод линии 12 параллельного порта ПЛИС MAX.
35	Вывод линии 13 параллельного порта ПЛИС MAX.
37	Вывод линии 14 параллельного порта ПЛИС MAX.
39	Вывод линии 15 параллельного порта ПЛИС MAX.
Четные 2-40	Корпус.

Поскольку напряжение питания ПЛИС MAX3064 (MAX3128) составляет 3.3 В, уровнем лог. «1» на линиях параллельного порта ПЛИС считается напряжение около 3 В. Уровнем лог. «1» на линиях INT0/1, T0/1 микроконтроллера ADuC812 (выводы 1, 3, 5, 7) считается напряжение около 5 В.

Внимание! Прямое замыкание линий параллельного порта ПЛИС или порта P3 ADuC812 на корпус при ненулевом напряжении на них может привести к выходу соответствующих микросхем из строя.

Для принудительного «обнуления» линий INT0/1, T0/1 в схему введен набор переключателей SW3-1; для принудительного «обнуления» линий 0-7 параллельного порта ПЛИС в схему введен набор переключателей SW3-2,

замыкающих соответствующие линии через резисторы 100 Ом на корпус. Для того, чтобы принудительно «обнулить» соответствующую линию, необходимо установить соответствующий переключатель в положение «ON». Соответствие номеров переключателей и линий приведено в таблице.

Таблица 13. Соответствие DIP-переключателей и линий параллельного порта ПЛИС и МК ADuC812

№ переключателя	Линия
Набор переключателей SW3-1	
1	Вход INT0 ADuC812 (P3.2).
2	Вход INT1 ADuC812 (P3.3).
3	Вход T0 ADuC812 (P3.4).
4	Вход T1 ADuC812 (P3.5).
Набор переключателей SW3-2	
1	Линия 0 параллельного порта ПЛИС MAX.
2	Линия 1 параллельного порта ПЛИС MAX.
3	Линия 2 параллельного порта ПЛИС MAX.
4	Линия 3 параллельного порта ПЛИС MAX.
5	Линия 4 параллельного порта ПЛИС MAX.
6	Линия 5 параллельного порта ПЛИС MAX.
7	Линия 6 параллельного порта ПЛИС MAX.
8	Линия 7 параллельного порта ПЛИС MAX.

4.4 Обзор компонентов принципиальной электрической схемы SDK-1.1

4.4.1 Микроконтроллер ADuC812

Микроконтроллер ADuC812 является клоном Intel 8051 (8052) со встроенной периферией, а значит, является представителем Гарвардской архитектуры.

Основные характеристики:

- Рабочая частота 11.0592 МГц.
- 8 Кб Flash (10000-50000 циклов доступа к памяти/стирание-запись-чтение) для хранения программ. В стенде SDK-1.1 в этой памяти располагается резидентный загрузчик и системная таблица векторов прерываний.
- 256 байт ОЗУ данных.
- 640 байт программируемого EEPROM со страничной организацией (160 страниц по 4 байта, 10000-50000 циклов доступа к памяти/стирание-запись-чтение) для хранения данных (например, различных настроек).
- Адресное пространство памяти программ 64 Кб.
- Адресное пространство внешней памяти данных 16 Мб.

- Четыре 8-разрядных порта ввода-вывода (три двунаправленных, один порт ввода).
- Три 16-битных таймера/счетчика и таймер WatchDog.
- 8-канальный 12-битный АЦП, который может работать в режиме (максимальная частота выборки-дискретизации 200 КГц).
- 2-канальный 12-битный ЦАП.
- Внутренний термодатчик.
- Режим управления питанием.
- Универсальный асинхронный приемопередатчик (UART).
- Интерфейс I2C (используется в стенде SDK-1.1), интерфейс SPI (не используется в стенде SDK-1.1).

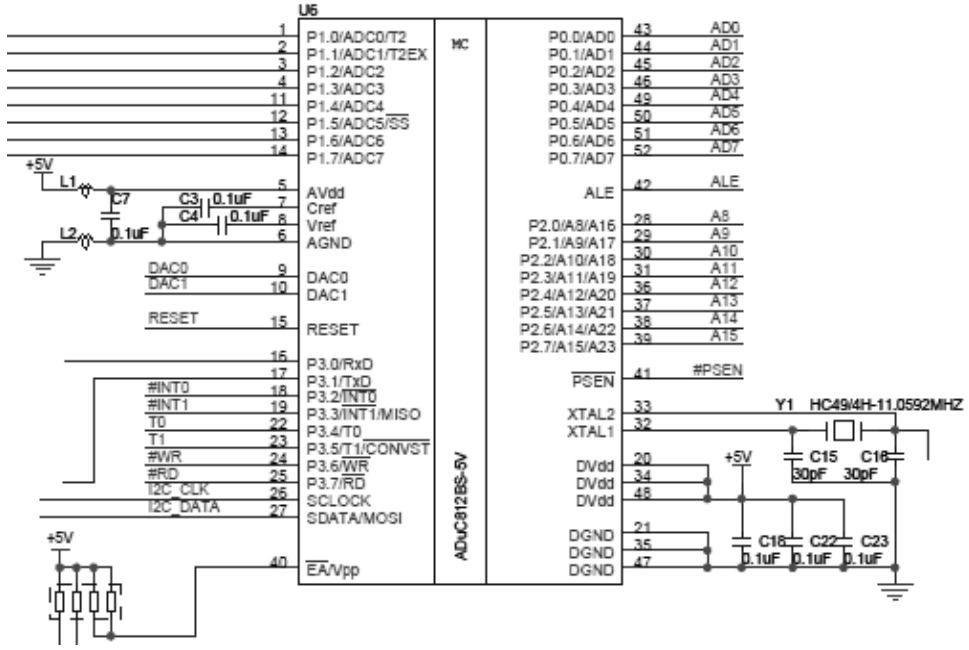


Рисунок 73. Микроконтроллер ADuC812 на принципиальной электрической схеме стенда SDK-1.1

4.4.2 Внешняя память программ и данных

Внешняя память программ и данных стенда SDK-1.1 – статическое ОЗУ (SRAM), имеющее страничную организацию и предназначенное для размещения пользовательских программ и данных. Размер страницы – 64 Кб. Первая страница (страница 0) доступна для выборки и команд, и данных микроконтроллером ADuC812. Остальные страницы доступны только для размещения данных (логическое адресное пространство внешней памяти данных составляет 16 Мб). Однако, реально для пользовательских программ доступно не 64 Кб, а 52 Кб, т.к. в младшие адреса отображается 8 Кб Flash-памяти ADuC812 (адреса 0000h-1FFFh). Кроме того, 4 Кб зарезервировано резидентным загрузчиком МК (адреса F000h-FFFFh).

Если в пользовательской программе используются прерывания, то ее рекомендуется загружать с адреса 2100h, т.к. в пространстве адресов 2000h-2100h располагается пользовательская таблица векторов прерывания.

Физическое адресное пространство внешней памяти данных в стенде SDK-1.1 ограничено 512 Кб (т.е. не может быть 16 Мб), потому что начиная с 8 страницы располагается адресное пространство ПЛИС (MAX 3064).

Внешняя память программ и данных подключается к МК ADuC812 по системной шине (как и ПЛИС).

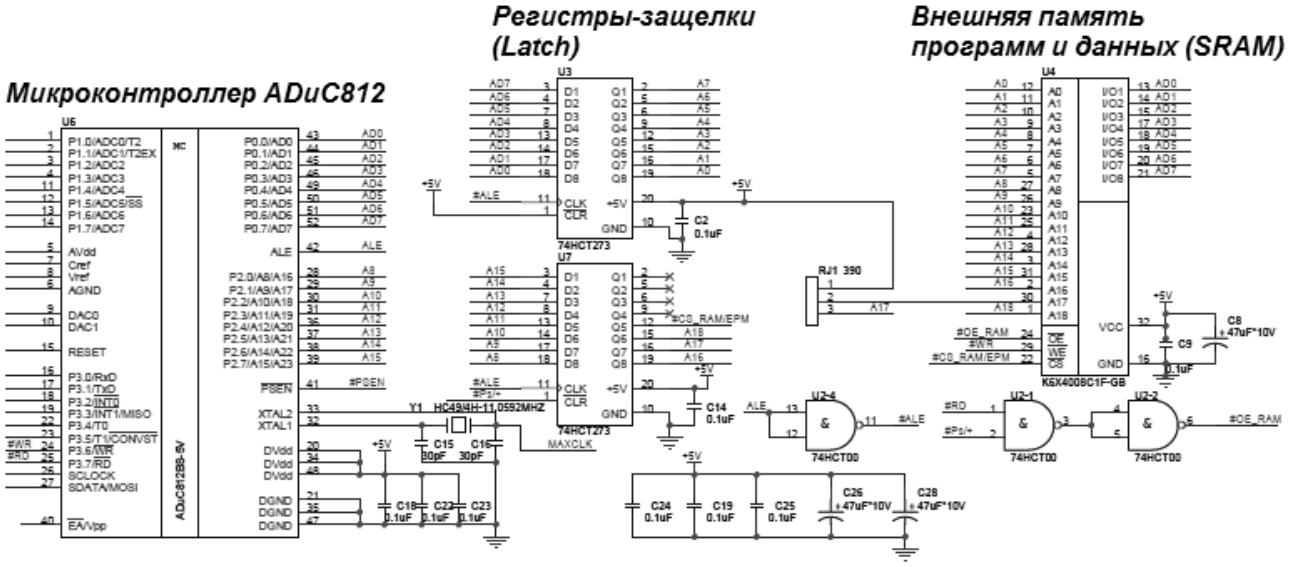
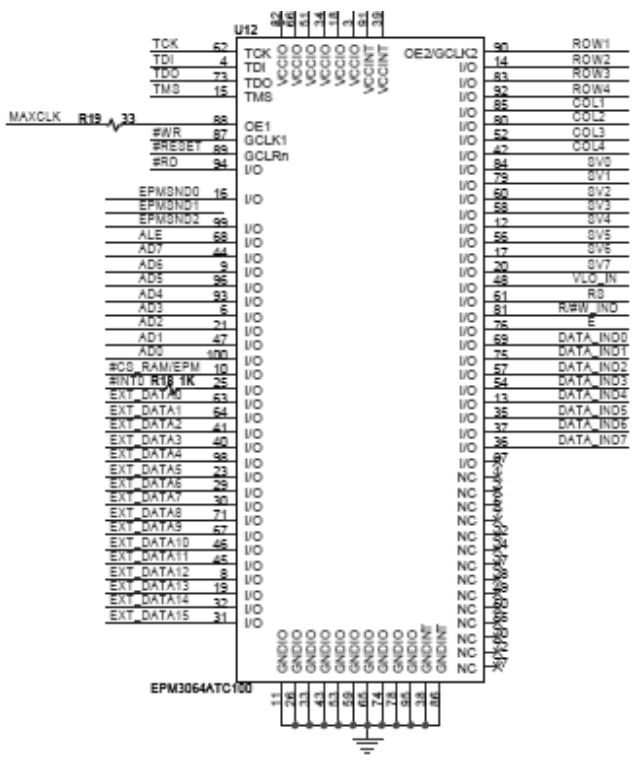


Рисунок 74. Подключение внешней памяти программ и данных к МК ADuC812

4.4.3 Расширитель портов ввода-вывода на базе ПЛИС



В SDK-1.1 используется программируемая логическая интегральная схема (ПЛИС) семейства MAX3000A фирмы Altera (EPM3064A) как расширитель портов ввода-вывода.

К ПЛИС подключены:

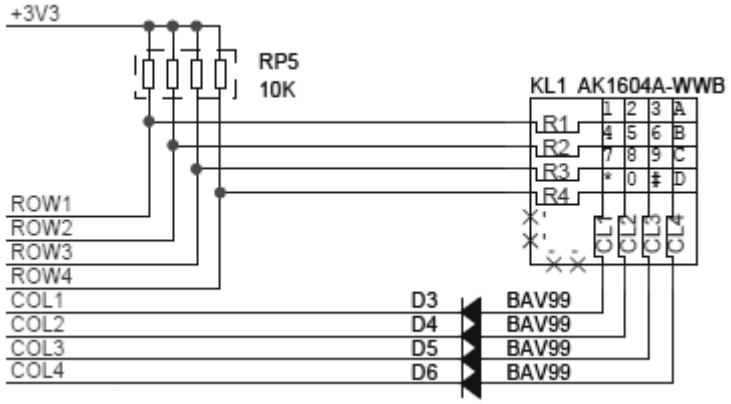
- Клавиатура
- ЖКИ
- Линейка светодиодов
- Звуковой излучатель
- 16 дискретных портов ввода-вывода

Для программиста расширитель портов представлен в виде нескольких однобайтовых регистров находящихся в начале восьмой страницы внешней памяти данных.

Далее будет представлен обзор периферийных устройств, подключенных к расширителю портов ввода-вывода, их обозначение на принципиальной электрической схеме стенда SDK-1.1.

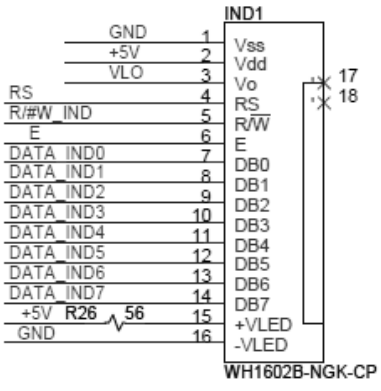
4.4.3.1 Матричная клавиатура

Клавиатура подключена через расширитель портов ввода-вывода.



Клавиатура организована в виде матрицы 4x4. Доступ к колонкам и рядам организован как чтение/запись определенного регистра ПЛИС (4 бита соответствуют 4 колонкам, другие 4 бита – рядам). Ряды ROW1..ROW4 подключены к плюсу питания через резисторы. Это обеспечивает наличие логической единицы при отсутствии нажатия (чтение регистра ПЛИС). На столбцы клавиатуры COL1..COL4 подают логический ноль (запись через регистр ПЛИС). При нажатии на кнопку происходит изменение значения сигнала на входе соответствующего ряда с единицы на ноль.

4.4.3.2 Жидкокристаллический индикатор

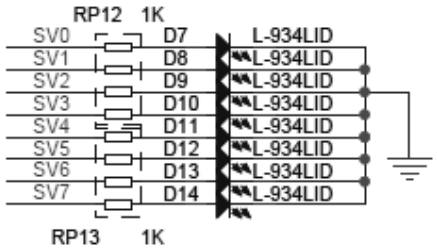


ЖКИ работает в текстовом режиме (2 строки по 16 символов), имеет подсветку (цвет желто-зеленый). Основные характеристики:

- Габариты: 80x36x13.2 мм.
- Активная область 56.21x11.5 мм.
- Размеры точки 0.56x0.66 мм; размеры символа 2.96x5.56 мм.
- Встроенный набор 256 символов (ASCII + кириллица).
- Генератор символов с энергозависимой памятью на 8 пользовательских символов.

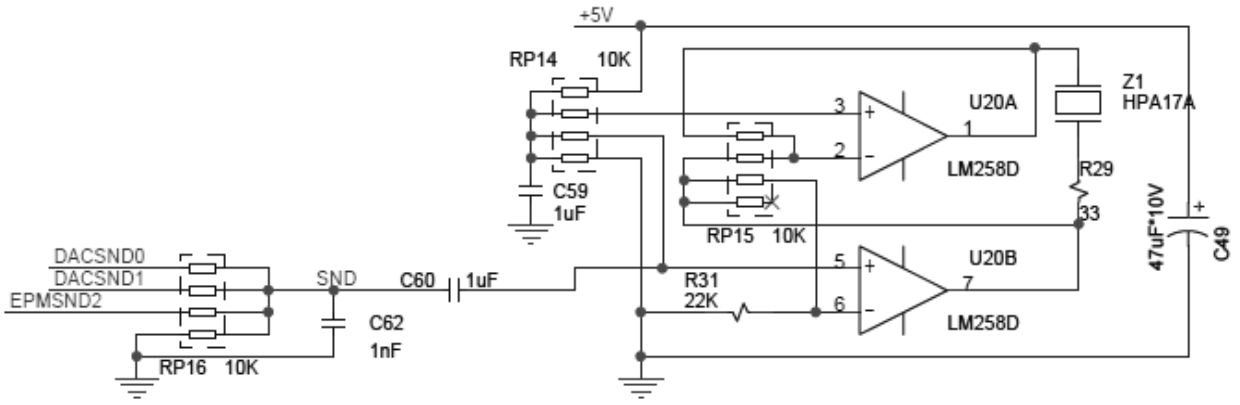
Подробнее о ЖКИ можно прочитать в подразделе 4.6.9.

4.4.3.3 Светодиодные индикаторы



Светодиодные индикаторы подключены к расширителю портов ввода-вывода. Так как все катоды светодиодов подключены к корпусу, для зажигания светодиодов необходимо подать напряжение +5В (лог. «1») на соответствующий анод. Резисторы R12..R13 ограничивают ток, текущий через порт ввода-вывода и светодиод. В данном случае приблизительный ток можно вычислить по закону Ома: I = U/R, I = 3.3/1000 = 3.3 мА. От силы тока зависит яркость горения светодиода. Если ток сделать очень большим, то порт ввода-вывода или светодиод могут выйти из строя.

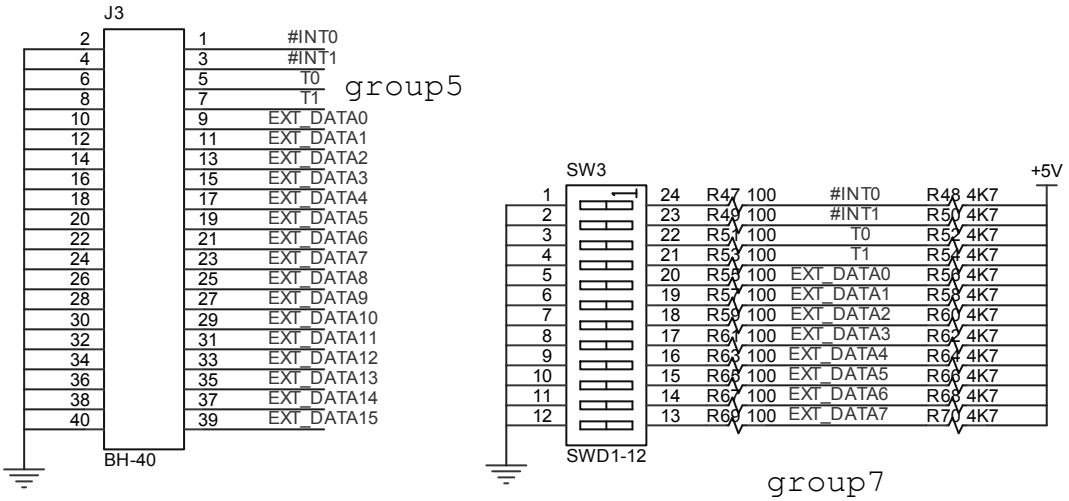
4.4.3.4 Звукоизлучатель



В SDK-1.1 используется пьезоэлектрический звукоизлучатель HPA17A (Z1). Выходы EPMSND0..EPMSND2 подключены к расширителю портов ввода-вывода (в SDK-1.1R3/R4/R5 сигналы EPMSND0 и EPMSND1 не подключены).

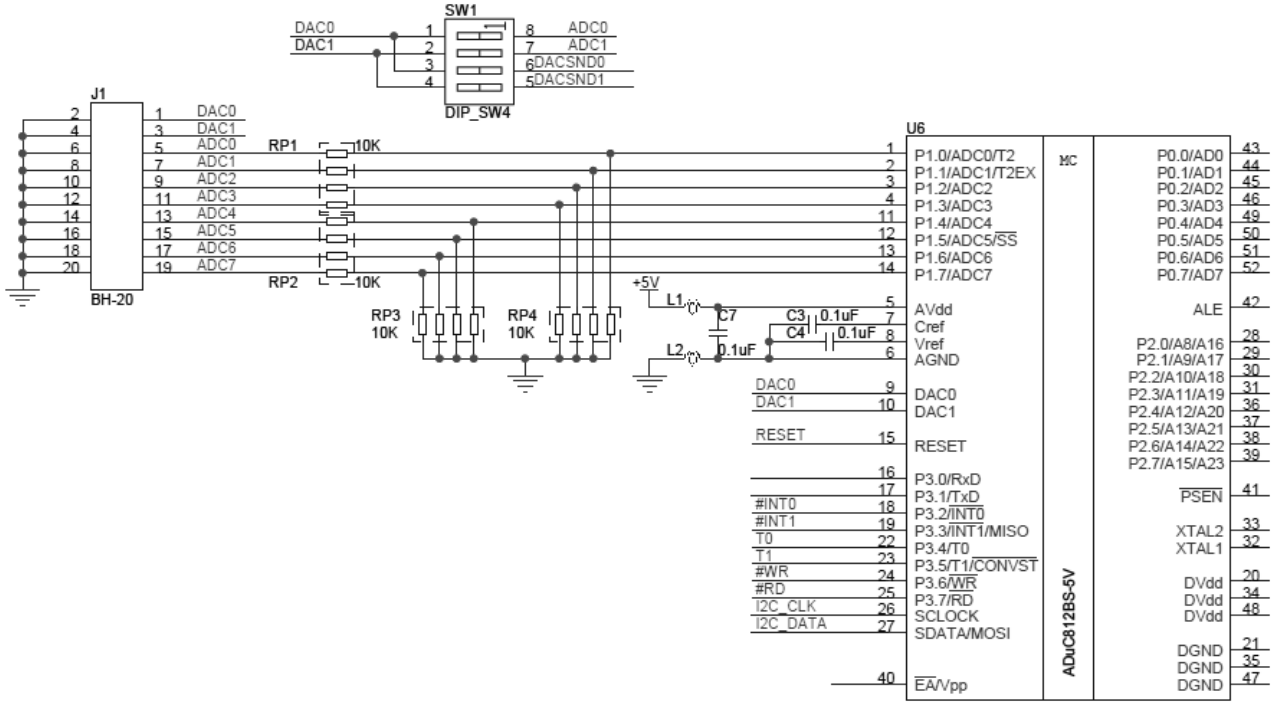
4.4.3.5 Дискретные входы-выходы

Дискретные входы-выходы предназначены для ввода и вывода информации, представленной в двоичном виде. Сигнал на входе или выходе дискретного порта может принимать значение логического нуля или единицы. В SDK-1.1 дискретные порты выведены на разъем J3. Эти порты можно использовать для подключения модулей SDX-0.9 или каких-либо других внешних устройств. Кроме этого, к дискретным входам-выходам подключены DIP-переключатели (SW3), позволяющие задавать фиксированные значения сигналов на входах. По умолчанию все входы притянуты к логической единице (через резисторы на +5В). При замыкании переключателя SW3 на выбранном входе появляется логический ноль.



Дискретные входы-выходы не имеют гальванической изоляции. Логическому нулю соответствует 0В, а логической единице +5В (уровни TTL). Нагрузочная способность дискретных портов ввода-вывода, подключенных к разъему J3, невелика, так как на разъем выведены порты ADuC812 без каких-либо дополнительных усилителей.

4.4.4 Аналоговые входы-выходы



ADuC812 имеет в своем составе 8 быстродействующих 12-разрядных АЦП и 2 12-разрядных ЦАП. Для коррекции зависимости параметров ЦАП и АЦП от температуры в ADuC812 встроен термодатчик. Все входы ЦАП и выходы АЦП выведены на разъем J1. Кроме того, выходы DAC0 и DAC1 можно замкнуть на входы ADC0 и ADC1 с помощью переключателя SW1.

4.4.5 Особенности реализации последовательного канала в стенде SDK 1.1

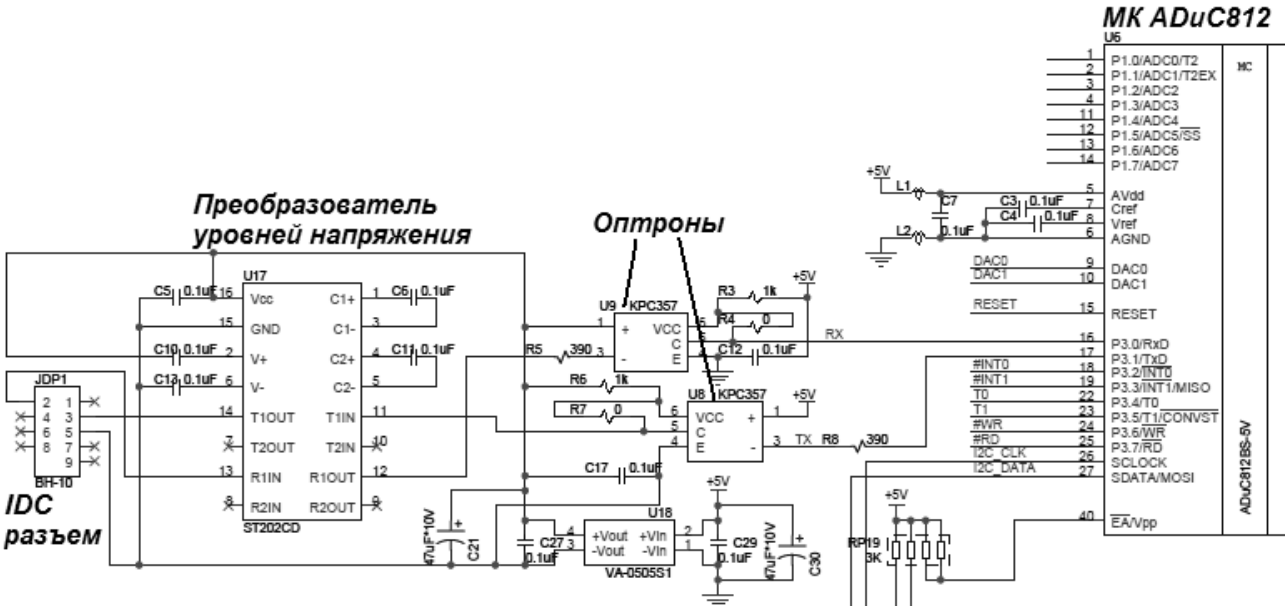
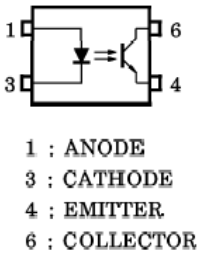


Рисунок 75. Гальванически изолированный последовательный интерфейс SDK-1.1

В SDK-1.1 последовательный канал гальванически развязан. Гальваническая изоляция или гальваническая развязка – разделение электрических цепей посредством не проводящего ток материала. Блок гальванической изоляции не всегда присутствует в трактах подобного вида. Гальваническая изоляция нужна для защиты ядра ВС от помех, от разности напряжений при коммутации (установке оборудования). Реализуется с помощью трансформаторной изоляции или с помощью оптоэлектронной схемы. Недостаток трансформаторов состоит в том, что они работают только на переменном токе. Оптоэлектронные схемы (оптопары) состоят из светоизлучающих приборов (диоды) и фотоприёмников (фоторезисторы, фототранзисторы). Оптопары работают хорошо только на полярном подключении, что неудобно при передаче аналоговых сигналов. Гальваническая изоляция позволяет защитить SDK-1.1 от высоких напряжений, различных наводок и подключать его к ПК во время работы.



Реализована гальваническая изоляция на базе двух оптронов U8 и U9 (TLP 181). Оптрон TLP181 состоит из светодиода (выводы 1,3) и фототранзистора (выводы 6,4). Если через светодиод пустить ток, то он начинает излучать свет. Свет падает на PN переход фототранзистора и открывает его. Когда свет гаснет, фототранзистор закрывается. Гальваническая изоляция достигается как раз за счет того, что между двумя элементами оптрона нет никакой связи кроме оптической.

Выход передатчика последовательного канала TxD попадает на катод оптрона U8. Далее сигнал попадает на преобразователь уровней напряжений (из ТТЛ в ±12 В).

4.4.6 I²C-устройства

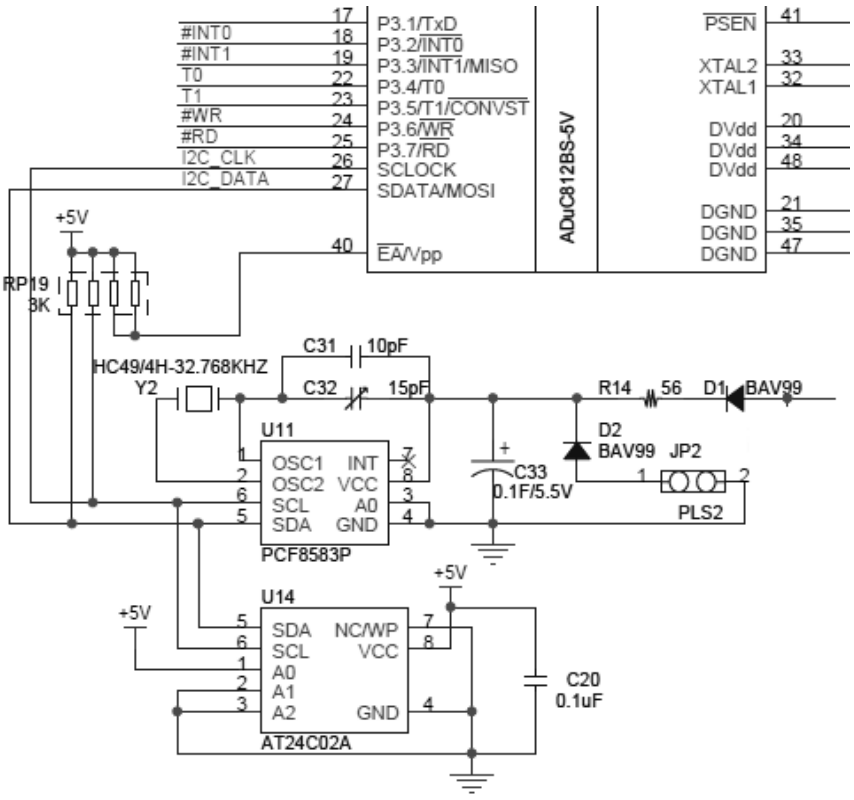


Рисунок 76. I2C-устройства стенда SDK-1.1

В стенде SDK-1.1 два устройства подключены по шине I²C (см. подраздел 2.3.1) к микроконтроллеру ADuC812: часы реального времени PCF8583 (U11) и EEPROM AT24C02A (U14).

EEPROM – электрически стираемое перепрограммируемое постоянное запоминающее устройство. Объем памяти EEPROM (AT24C02A, Atmel), установленной в стенде SDK-1.1, составляет 256 байт (2 Кбит).

Основные характеристики:

- Возможность перезаписи до 1 млн. раз.
- Возможность побайтовой и постраничной записи (в текущей конфигурации размер страницы составляет 8 байт).

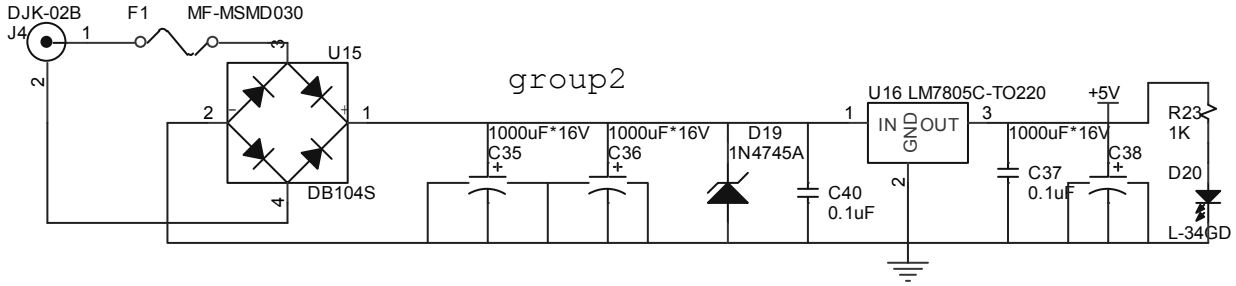
Подробное описание организации и принципа работы данной микросхемы EEPROM приведено в подразделе 2.2.3.

Микросхема PCF8583 (Philips) – часы реального времени (часы/календарь) с памятью объемом 256 байт, работающие от кварцевого резонатора с частотой 32,768 кГц. Из 256 байт памяти собственно часами используются только первые 16 (8 постоянно обновляемых регистров-защелок на установку/чтение даты/времени и 8 на будильник), остальные 240 байт доступны для хранения данных пользователя. Точность измерения времени – до сотых долей секунды.

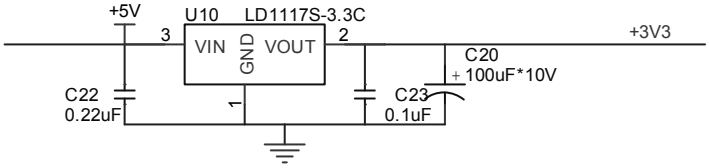
Подробное описание организации и принципа работы RTC PCF8583 приведено в подразделе 2.2.31.2.5.

4.4.7 Источник питания

Переменное (15..16В) или постоянное (9..10В) напряжение от внешнего источника питания попадает на диодный мост U15 через разъем J4. Сердцем встроенного в SDK 1.1 источника питания является микросхема LM7805C. Эта микросхема является интегральным стабилизатором с защитой от перегрева и короткого замыкания. Выходное напряжение – 5В ± 2%, выходной ток до 1 А.



Стабилитрон D19 (1N4745A) предназначен для защиты LM7805C и электролитических емкостей от превышения входного напряжения (напряжения пробоя стабилитрона – 16В). Электролитические конденсаторы C35 и C36 необходимы для сглаживания пульсаций входного напряжения. Электролитический конденсатор C38 необходим для поддержки работоспособности SDK-1.1 при кратковременных пропаданиях напряжения питания. Емкости C40 и C37 необходимы для фильтрации высокочастотных помех, их использование определяется штатной схемой включения LM7805C.



Напряжение 3.3 В для питания ПЛИС формируется с помощью стабилизатора U10 (LD1117S).

4.4.8 Схема сброса

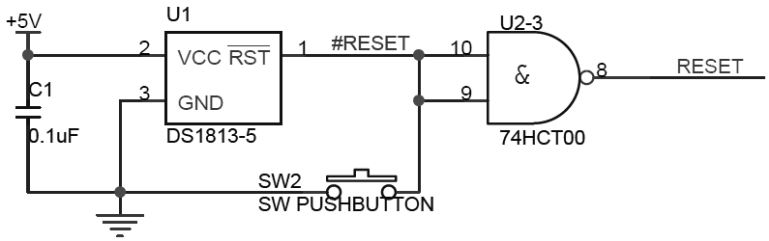
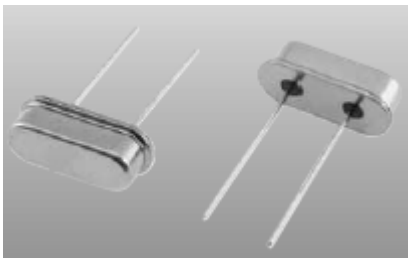


Схема сброса предназначена для формирования качественного сигнала RESET после включения питания, после нажатия кнопки RESET или после выключения питания. Проблема состоит в том, что при старте контроллера после включения питания или при выключении питания возможны различные переходные процессы, способные привести к некорректному исполнении программ или порче содержимого ОЗУ. Супервизор питания (U1) DS1813 обеспечивает формирование сигнала RESET на 150 мс, т.е. на время, достаточное для окончания всех переходных процессов.

4.4.9 Кварцевые резонаторы



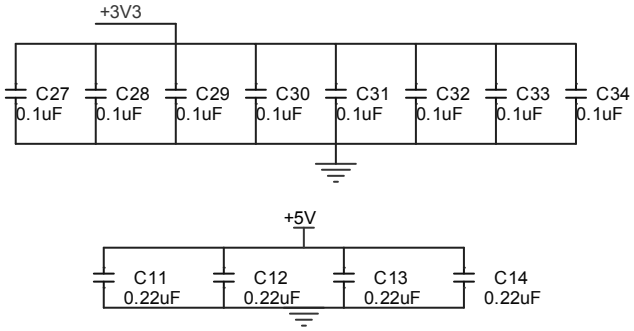
Кварцевые резонаторы – устройства, использующие пьезоэлектрический эффект для возбуждения электрических колебаний заданной частоты. При совпадении частоты приложенного напряжения с одной из собственных механических частот кварцевого вибратора в приборе возникает явление резонанса, приводящее к резкому увеличению проводимости. Обладая среди резонаторов самой высокой добротностью Q~105-107 (добротность колебательного LC-контура не превышает 102, пьезокерамики – 103), кварцевые резонаторы имеют также высокую температурную стабильность и низкую долговременную нестабильность частоты ($10^6..10^8$). Кварцевые резонаторы применяются в генераторах опорных частот, в управляемых по частоте генераторах, селективных устройствах: фильтрах, частотных дискриминаторах и т.д.

Y1	Y2
Y1 HC49/4H-11.059MHZ	Cry ctal8x3-32.768 Y2

В SDK-1.1 два кварцевых резонатора. Y1 служит для тактирования ADuC812 (11,0592 МГц), а Y2 для тактирования часов реального времени (32,768 КГц).

4.4.10 Фильтрующие емкости

Фильтрующие емкости равномерно распределены по всей поверхности печатной платы. Каждый конденсатор соединяет плюс питания с корпусом. Фильтрующие емкости шунтируют высокочастотные помехи, возникающие в цепях питания 3.3 и 5В.



Шунтирование происходит из-за того, что активное сопротивление емкости тем меньше, чем выше частота сигнала.

$Xc = \frac{1}{2\pi fC}$, где Xc – активное сопротивление конденсатора, f – частота, C – емкость.

Для постоянного напряжения сопротивление конденсатора близко к бесконечности, а для переменного напряжения высокой частоты – конденсатор является резистором с низким сопротивлением.

4.5 Микроконтроллер ADuC812

Микроконтроллер ADuC812 является клоном Intel 8051 (8052) со встроенной периферией, а значит, является представителем Гарвардской архитектуры.

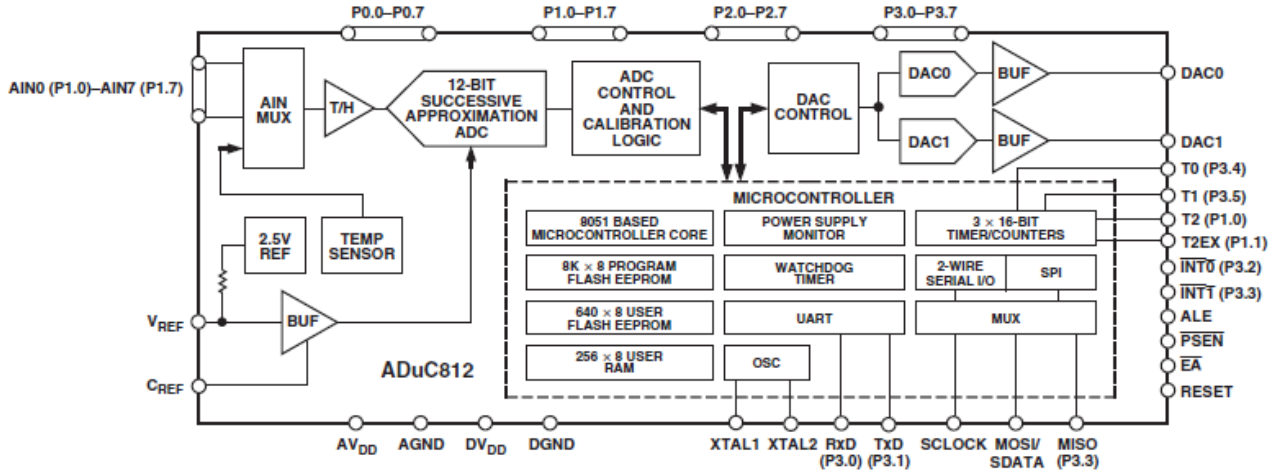


Рисунок 77. Структурная схема микроконтроллера ADuC812

4.5.1 Система команд микроконтроллера ADuC812

Микроконтроллер ADuC812 (как и все представители семейства MCS-51) является типичным микропроцессорным устройством с CISC-архитектурой. Система команд включает 111 команд. В машинном коде команда занимает один, два или три байта в зависимости от типа адресации. Команды выполняются за один, два или четыре (умножение и деление) машинных цикла.

Система команд микроконтроллера предоставляет большие возможности обработки данных, обеспечивает реализацию логических, арифметических операций, а также управление в режиме реального времени.

В этой системе команд реализована побитная, потетрадная (4 бита), побайтовая (8 бит) и 16-разрядная обработка данных. Микросхемы семейства MCS-51 – это 8-разрядные микропроцессоры, а это означает, что ПЗУ, ОЗУ, регистры специального назначения, АЛУ и внешние шины имеют байтовую организацию. Двухбайтовые данные используются только регистром-указателем (DPTR) и счетчиком команд (PC).

Способы адресации операндов

При определении способа адресации операндов в команде необходимо учитывать, что адресация для каждого операнда команды своя. В общем случае адресация источника и приемника могут не совпадать [51].

Неявная адресация. При неявной адресации регистр источник или регистр приёмник подразумевается в самом коде операции. Например:

```
    03 RR A ;Сдвинуть содержимое аккумулятора вправо
    D4 DA A ;Произвести десятичную коррекцию результата суммирования
    E8 MOV A, R0 ;В первом операнде использована неявная адресация, а во втором
– регистровая
```

Регистровая адресация используется для обращения к восьми рабочим регистрам выбранного банка рабочих регистров, а также для обращения к регистрам A, B, AB (сдвоенному регистру), DPTR, и к флагу переноса C. Номер регистра записывается в трех младших битах команды. Например:

```
    F8 MOV R5, A ;в первом операнде использована регистровая адресация, а во
втором – неявная
```

Прямая байтовая адресация используется для обращения к ячейкам внутренней памяти (ОЗУ) данных (адреса 0:127) и к регистрам специального назначения (адреса 128:256). Адрес ячейки памяти помещается во второй байт команды. Например:

```
    E520 MOV A, 20h ;во втором операнде использована прямая байтовая адресация,
а в первом – неявная
    8D15 MOV 15h,R6 ;в первом операнде использована прямая байтовая адресация,
а во втором – регистровая
```

Прямая битовая адресация используется для обращения к отдельно адресуемым 128 битам, расположенным в ячейках с адресами 20H-2FH, и к отдельно адресуемым битам регистров специального назначения. Например:

```
    D220 SETB 20h ;использована прямая битовая адресация
    C215 CLR 15h ;использована прямая битовая адресация
```

Косвенно-регистровая адресация используется для обращения к ячейкам внутреннего ОЗУ данных. В качестве регистров-указателей адреса используются регистры R0, R1 выбранного банка регистров. Например:

```
    E6 MOV A,@R0 ;В первом операнде использована неявная адресация, а во втором
- косвенно-регистровая
    F7 MOV @R1,A ;В первом операнде использована косвенно-регистровая
адресация, а во втором – неявная
```

Косвенно-регистровая адресация используется также для обращения к внешней памяти данных. В этом случае с помощью регистров-указателей R0 и R1 (рабочего банка рабочих регистров) выбирается ячейка из блока 256 байт внешней памяти данных. Номер блока предварительно задается содержимым порта P2. Например:

```
    E2 MOVX A,@R0 ;В первом операнде использована неявная адресация, а во
втором - косвенно-регистровая
    F3 MOVX @R1,A ;В первом операнде использована косвенно-регистровая
адресация, а во втором – неявная
```

Если в качестве регистра-указателя используется 16-разрядный указатель данных (DPTR), то можно выбрать любую ячейку внешней памяти данных объемом до 64 Кбайт. (В некоторых моделях микроконтроллеров семейства

MSC-51 таким образом можно обращаться к внутренней памяти данных объемом более 256 байт).

```
    E0 MOVX A,DPTR;В первом операнде использована неявная адресация, а во
втором - косвенно-регистровая
    F0 MOVX DPTR,A;В первом операнде использована косвенно-регистровая
адресация, а во втором – неявная
```

Косвенно-регистровая адресация по сумме базового и индексного регистра (содержимое аккумулятора А) упрощает просмотр таблиц, записанных в памяти программ. Любой байт из таблицы может быть выбран по адресу, определяемому суммой содержимого DPTR или PC и содержимого А, например:

```
    83 MOV A, @A+PC ;В первом операнде использована неявная адресация, а во
втором - косвенно-регистровая
    93 MOV A, @A+DPTR ;В первом операнде использована неявная адресация, а во
втором - косвенно-регистровая.
```

Непосредственная адресация позволяет выбрать из адресного пространства памяти программ константы, явно указанные в команде, например:

```
    7414 MOV A, #14h ;В первом операнде использована неявная адресация, а во
втором - непосредственная
    902048 MOV DPTR, #2048h ;В первом операнде использована неявная адресация,
а во втором - непосредственная
```

4.5.2 Порты ввода-вывода микроконтроллера ADuC812

Порт можно определить как точку, через которую осуществляется взаимодействие с каким-либо блоком в системе ввода-вывода, многоразрядный вход или выход устройства. *Порт ввода-вывода* – это логическая адресуемая единица системы ввода-вывода, которая характеризуется, в первую очередь, следующими тремя признаками:

- адресом;
- форматом данных, пересылаемых через него (под форматом данных подразумевается как их разрядность, так и положение значащих разрядов);
- набором допустимых с ним операций (чтение, запись или и то, и другое).

Порты P0, P1, P2, P3 ADuC812 являются квазидвунаправленными портами ввода-вывода и предназначены для обеспечения обмена информацией микроконтроллера с внешними устройствами, образуя 32 линии ввода-вывода. Каждый из портов содержит восьмиразрядный регистр, имеющий байтовую и битовую адресацию для установки (запись «1») или сброса (запись «0») разрядов этого регистра с помощью программного обеспечения. Выходы этих регистров соединены с внешними ножками микросхемы.

Кроме работы в качестве обычных портов ввода-вывода, внешние выводы портов P0…P3 могут выполнять ряд дополнительных (альтернативных) функций.

Порт P0 может быть использован для организации шины адреса/данных при работе микроконтроллера с внешней памятью данных или программ (см. принципиальную схему стенда SDK-1.1), при этом через него выводится младший байт адреса (A0–A7), выдается из микроконтроллера или принимается в микроконтроллер байт данных.

Порт P1 – аналоговые входы.

Порт P2 может быть использован для организации шины адреса при работе микроконтроллера с внешней памятью данных или программ, при этом через него выводится старший байт адреса (A8–A15) для доступа к памяти программ; средний и старший байт адреса (A8 – A15, A16 – A23) для доступа к памяти данных.

Каждая линия *порта P3* имеет индивидуальную альтернативную функцию, которая может быть задействована простым обращением к устройству, соединенному с ножкой порта:

- P3.0 RxD – вход последовательного порта (UART).
- P3.1 TxD – выход последовательного порта (UART).
- P3.2 INT0 используется как вход 0 внешнего запроса прерываний.
- P3.3 INT1 используется как вход 1 внешнего запроса прерываний.
- P3.4 T0 используется как вход счетчика внешних событий 0.
- P3.5 T1 используется как вход счетчика внешних событий 1.
- P3.6 WR – строб записи во внешнюю память данных.
- P3.7 RD – строб чтения из внешней памяти данных.

4.5.3 Организация памяти программ микроконтроллера ADuC812

Память программ предназначена для хранения программ и имеет отдельное от памяти данных адресное пространство объемом 64 Кб (Гарвардская архитектура), причем для некоторых микросхем семейства MCS-51 для хранения программ на кристалле микроконтроллера расположено ППЗУ (например, Flash в ADuC812). Это ППЗУ отображается в область младших адресов памяти программ. Учитывая, что выполнение программы после сброса микроконтроллера всегда начинается с нулевого адреса памяти программ, то при включении питания начнет выполняться программа, записанная во внутреннем ППЗУ микроконтроллера. Микроконтроллеры, не имеющие внутреннего ППЗУ (например, КР1816ВЕ31 и КР1830ВЕ31) могут работать только с внешней микросхемой ПЗУ емкостью до 64 Кб (при использовании портов P1 и P3 в качестве расширителя адреса объем подключаемой ПЗУ может быть увеличен до 1 Гб). Микроконтроллеры семейства MCS-51 имеют внешний вывод EA (External Access Enable), с помощью которого можно запретить работу внутренней памяти, для чего необходимо подать на вывод EA

логический «0» (соединить этот вывод с корпусом). При этом внутренняя память программ отключается и, начиная с нулевого адреса, все обращения происходят к внешней памяти программ [1, 51].

Распределение памяти программ микроконтроллера ADuC812 представлено ниже:

Рисунок 78. Адресное пространство памяти программ

Доступ к внешней памяти программ осуществляется в двух случаях:

- при действии сигнала EA=0 независимо от адреса обращения,
- в любом случае, если программный счетчик (PC) содержит число большее, чем максимальная ячейка внутренней памяти программ (больше 1FFFh, т.е. больше 8 Кб Flash).

В стенде SDK-1.1 EA = 1 (см. рисунок), поэтому при подаче питания или после перезапуска начинает исполняться загрузчик во Flash-памяти МК ADuC812. Этот загрузчик ждет пользовательскую программу, которую (в качестве исполнительного модуля) «получает» по коммуникационному интерфейсу RS-232 от персонального компьютера. Далее загрузчик записывает эту программу во внешнюю память программ и данных и передает ей управление (по умолчанию, по адресу 2100h), и она исполняется.

4.5.4 Организация памяти данных микроконтроллера ADuC812

Несмотря на то, что это самое маленькое адресное пространство из рассматриваемых, оно устроено наиболее сложным образом.

Внутреннее ОЗУ данных предназначено для временного хранения информации, используемой в процессе выполнения программы, и занимает 256 восьмиразрядных ячеек, с адресами от 00h до FFh.

Регистры специального назначения (Special Function Registers, SFR) занимают адреса внутренней памяти данных с 80h по FFh. SFR служат для управления, конфигурирования и передачи/приема данных от периферийных устройств, т. е. выполняют функцию интерфейса между процессором и периферией на кристалле. Так как адреса SFR совпадают со старшими адресами

внутреннего ОЗУ данных, то имеются особенности при использовании этих адресов внутренней памяти данных [1, 51].

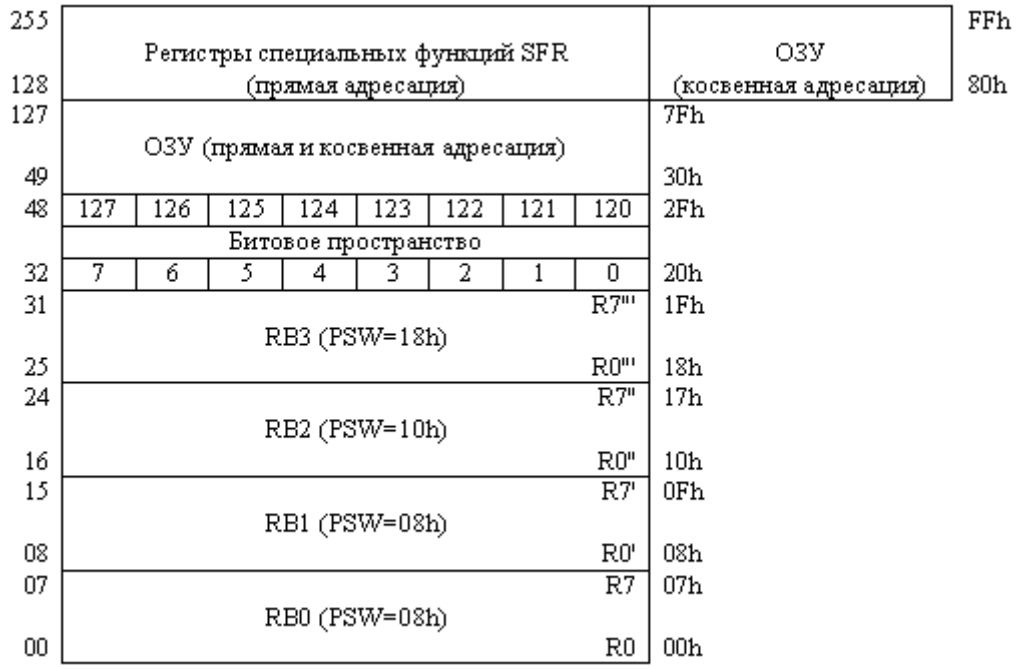


Рисунок 79. Внутренняя память данных микроконтроллера ADuC812

Система команд микроконтроллера позволяет обращаться к ячейкам внутренней памяти данных при помощи прямой и косвенно-регистровой адресации. При обращении к ячейкам памяти с адресами 00h-7Fh использование любого из этих видов адресации будет производить выборку одной и той же ячейки памяти. При обращении к ячейкам ОЗУ с адресами 80h-FFh следует воспользоваться косвенно-регистровой адресацией. Учитывая, что работа со стеком ведется при помощи косвенной адресации, то имеет смысл размещать в этой области памяти стек. Если же требуется обратиться к регистрам специальных функций, то нужно использовать прямую адресацию.

Регистры общего назначения (РОН, General-Purpose Registers) позволяют писать самые эффективные программы. У микроконтроллеров семейства MCS-51 программисту доступны восемь регистров. Более того, в этом семействе микроконтроллеров есть целых четыре набора (банка) регистров с именами RB0 - RB3. Банк регистров состоит из восьми восьмиразрядных регистров с именами R0, R1, …, R7. Несколько банков регистров служат для организации независимой работы нескольких параллельно выполняемых программ. Переключение банков регистров производится при помощи двух особых бит регистра слова состояния программы PSW (RS0 и RS1). Если организация нескольких параллельных потоков обработки данных не нужна, то можно пользоваться только нулевым банком регистров, включающимся автоматически после включения питания и сброса микроконтроллера, остальные ячейки памяти использовать как обычное ОЗУ.

Следующие после банков регистров внутреннего ОЗУ данных 16 ячеек памяти (адреса 20h-2Fh) образуют область памяти, к которой возможна как

байтовая, так и битовая адресация. В этих ячейках располагаются 128 программных флагов (битовых ячеек памяти). Обращение к отдельным битам этих ячеек возможно по их битовым адресам.

4.5.5 Программирование внутренних ППЗУ микроконтроллера ADuC812

Внутренняя Flash-память микроконтроллера ADuC812 отображается на младшие 8 Кб адресного пространства памяти программ. Она может быть перепрограммирована двумя способами:

- Внутрисхемное программирование (In-Circuit Programming). Для этого сигнальная линия PSEN (Program Store Enable) заземляется, а программа загружается через последовательный канал RS-232 во Flash (на персональном компьютере работает специальная инструментальная программа – загрузчик). Так в стенде SDK-1.1 перешивается загрузчик (UL3 или HEX-202), при этом на ПК запускается программатор Flash для микроконтроллера ADuC812 (dl.exe от Analog Devices).
- Параллельное программирование реализуется с использованием специальных программаторов других производителей. При этом порты P0, P1, P2 используются в качестве шины адреса и данных, линия ALE (Address Latch Enable) – сигнал записи, порт P3 – конфигурационный регистр, который определяет действия (стирание, запись и т. д.).

Внутренняя EEPROM-память программируется при помощи SFR-регистров: EDATA1-4 – 4 регистра данных для чтения/записи страницы; EADRL – регистр адреса страницы; ECON – регистр управления командами доступа к памяти (чтение, запись, стирание, верификация и др.).

И Flash, и EEPROM имеют ограниченное число циклов перезаписи (10000-50000 циклов доступа к памяти/стирание-запись-чтение в ADuC812).

4.5.6 Система прерываний ADuC812

В микроконтроллере ADuC812 девять источников прерываний с двумя уровнями приоритетов. Когда происходит прерывание, значение программного счетчика помещается в стек, а в сам счетчик загружается адрес соответствующего вектора прерывания. Адреса векторов указаны в таблице:

Прерывание	Наименование	Адрес вектора	Приоритет
PSMI	Монитор источника питания ADuC812	43H	1
IE0	Внешние прерывание INT0	03H	2
ADCI	Конец преобразования АЦП	33H	3
TF0	Переполнение Таймера/Счетчика 0	0BH	4
IE1	Внешние прерывание INT1	13H	5
TF1	Переполнение Таймера/Счетчика 1	1BH	6
I2CI/ISPI	Прерывание от I2C/SPI	3BH	7

RI/TI	Прерывание от UART	23H	8
TF2/EXF2	Переполнение Таймера/Счетчика 2	2BH	9

Регистр IE

Регистр специального назначения IE (адрес A8h) – регистр разрешения прерываний. Все биты регистра устанавливаются и сбрасываются пользователем.

Бит	**Название**	**Опписание**
7	EA	0 – запрещены, 1 – разрешены прерывания от всех источников
6	EADC	0 – запрещены, 1 – разрешены прерывания от АЦП
5	ET2	0 – запрещены, 1 – разрешены прерывания от Таймера 2
4	ES	0 – запрещены, 1 – разрешены прерывания от UART
3	ET1	0 – запрещены, 1 – разрешены прерывания от Таймера 1
2	EX1	0 – запрещено, 1 – разрешено внешние прерывание 1
1	ET0	0 – запрещены, 1 – разрешены прерывания от Таймера 0
0	EX0	0 – запрещено, 1 – разрешено внешние прерывание 0

Регистр IE2

IE2 (SFR адрес A9h) регистр разрешения вторичных прерываний

Бит	**Название**	**Опписание**
7-2	-	Зарезервированы для использования в будушем
1	ET0	0 – запрещены, 1 – разрешены прерывания от монитора источника питания
0	EX0	0 – запрещено, 1 – разрешено прерывания от SPI/I2C

Регистр IP

Регистр специального назначения IP (адрес B8h) – регистр приоритета прерываний. Все биты регистра устанавливаются и сбрасываются пользователем. После включения питания по умолчанию содержит 00h.

Бит	**Название**	**Описание**
7	PSI	0 –“высокий”, 1 – “низкий” приоритет прерывания от SPI/I2C
6	PADC	0 –“высокий”, 1 – “низкий” приоритет прерывания от АЦП
5	PT2	0 –“высокий”, 1 – “низкий” приоритет прерывания от Таймера 2
4	PS	0 –“высокий”, 1 – “низкий” приоритет прерывания UART
3	PT1	0 –“высокий”, 1 – “низкий” приоритет прерывания от Таймера 1
2	PX1	0 –“высокий”, 1 – “низкий” приоритет внешнего прерывания 1
1	PT0	0 –“высокий”, 1 – “низкий” приоритет прерывания от Таймера 0
0	PX0	0 –“высокий”, 1 – “низкий” приоритет внешнего прерывания 0

Для каждого источника прерывания программист может задать один из двух уровней приоритета: высокий и низкий. Прерывание с высоким уровнем приоритета может прерывать обслуживание прерывания с низким уровнем приоритета, а если прерывания с разными уровнями происходят одновременно, то прерывание с высоким приоритетом будет обслужено первым. Обслуживание прерывание не может быть прервано прерыванием с таким же уровнем приоритета. Если два прерывания с одинаковым уровнем приоритета происходят одновременно, то порядок их обработки определяется с помощью следующей таблицы:

Источник	Приоритет	Описание
PSMI	1(Наивысший)	Монитор источника питания ADuC812
IE0	2	Внешние прерывание INT0
ADCI	3	Конец преобразования АЦП
TF0	4	Переполнение Таймера/Счетчика 0
IE1	5	Внешние прерывание INT1
TF1	6	Переполнение Таймера/Счетчика 1
I2CI+ISPI	7	Прерывание от I2C/SPI
RI+TI	8	Прерывание от UART
TF2+EXF2	9 (Низший)	Переполнение Таймера/Счетчика 2

4.5.7 Особенности обработки прерываний в стенде SDK-1.1

Прерывания ADuC812 имеют вектора в диапазоне 0003h-0043h, которые попадают в область младших адресов памяти программ. Это пространство соответствует 8Кб (0000h-2000h) Flash-памяти.

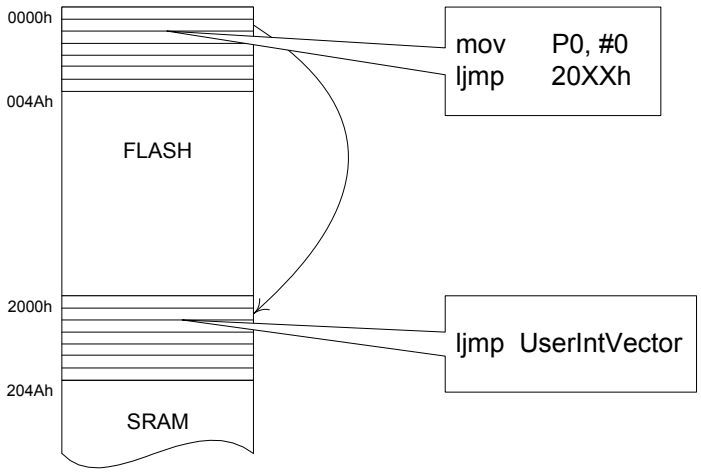


Рисунок 80. Использование прерываний в SDK-1.1

В стенде SDK-1.1 пользователь не имеет возможности записи во Flash-память (запись программ осуществляется во внешнюю память программ и данных), следовательно, не может подставить свои процедуры обработки прерываний (точнее, команды перехода к процедурам) по адресам, соответствующим векторам прерываний.

Проблема использования прерываний в пользовательских программах решается следующим образом:

1. По адресам (0003h-0043h) векторов прерываний во Flash-памяти SDK-1.1 располагаются команды переходов на вектора пользовательской таблицы, размещенной в адресах 2003h-2043h.
2. По адресам векторов пользовательской таблицы пользователем указываются команды переходов на процедуры обработки прерываний.

Приведем пример помещения собственного вектора в пользовательскую таблицу. Пусть требуется осуществить обработку прерываний от таймера 0 (прерывание 0Bh). В программу на языке Си (компилятор SDCC) можно вставить следующий код:

```
// Обработчик прерывания от таймера 0
void T0_ISR(void) __interrupt (1)
{
// Действия, выполняемые обработчиком
}

////////////////////////// SetVector ///////////////////////////
// Функция, устанавливающая вектор прерывания в пользовательской таблице
// прерываний.
// Вход:      Vector - адрес обработчика прерывания,
//            Address - вектор пользовательской таблицы прерываний.
// Выход:     нет.
// Результат: нет.
////////////////////////////////////////////////////////////////
void SetVector(unsigned char xdata * Address, void * Vector)
{
 unsigned char xdata * TmpVector; // Временная переменная
 // Первым байтом по указанному адресу записывается
 // код команды передачи управления ljmp, равный 02h
 *Address = 0x02;
 // Далее записывается адрес перехода Vector
 TmpVector = (unsigned char xdata *) (Address + 1);
 *TmpVector = (unsigned char) ((unsigned short)Vector >> 8);
 ++TmpVector;
 *TmpVector = (unsigned char) Vector;
 // Таким образом, по адресу Address теперь
 // располагается инструкция ljmp Vector
}

void main(void)
{
/*…*/
// Установка вектора в пользовательской таблице
SetVector(0x200B, (void *) T0_ISR);
// Разрешение прерываний от таймера 0
ET0=1; EA=1;
/*…*/
```

Примеры программ для стенда SDK-1.1, демонстрирующие работу с прерываниями, таймерами приведены в подразделе 6.4.

4.6 Расширитель портов ввода-вывода на базе ПЛИС

В SDK-1.1 используется программируемая логическая интегральная схема (ПЛИС) семейства MAX3000A фирмы Altera (EPM3064A) как расширитель портов ввода-вывода. Микросхема EPM3064A подключена к системной шине МК ADuC812. Адресная линия A19 используется как сигнал CS (chip select) для ПЛИС. Микросхема EPM3064A выбирается, когда на линии A19 логическая единица. Физический адрес ПЛИС таким образом равен 80000h, что соответствует восьмой странице внешней памяти.

Микросхема EPM3064A относится к типу CPLD (complex programmable logic device) и имеет следующие характеристики: энергонезависимая (EEPROM-based), 1250 вентилей, 64 макроячейки, 4 логических блока, 220 МГц, 100 перезаписей.

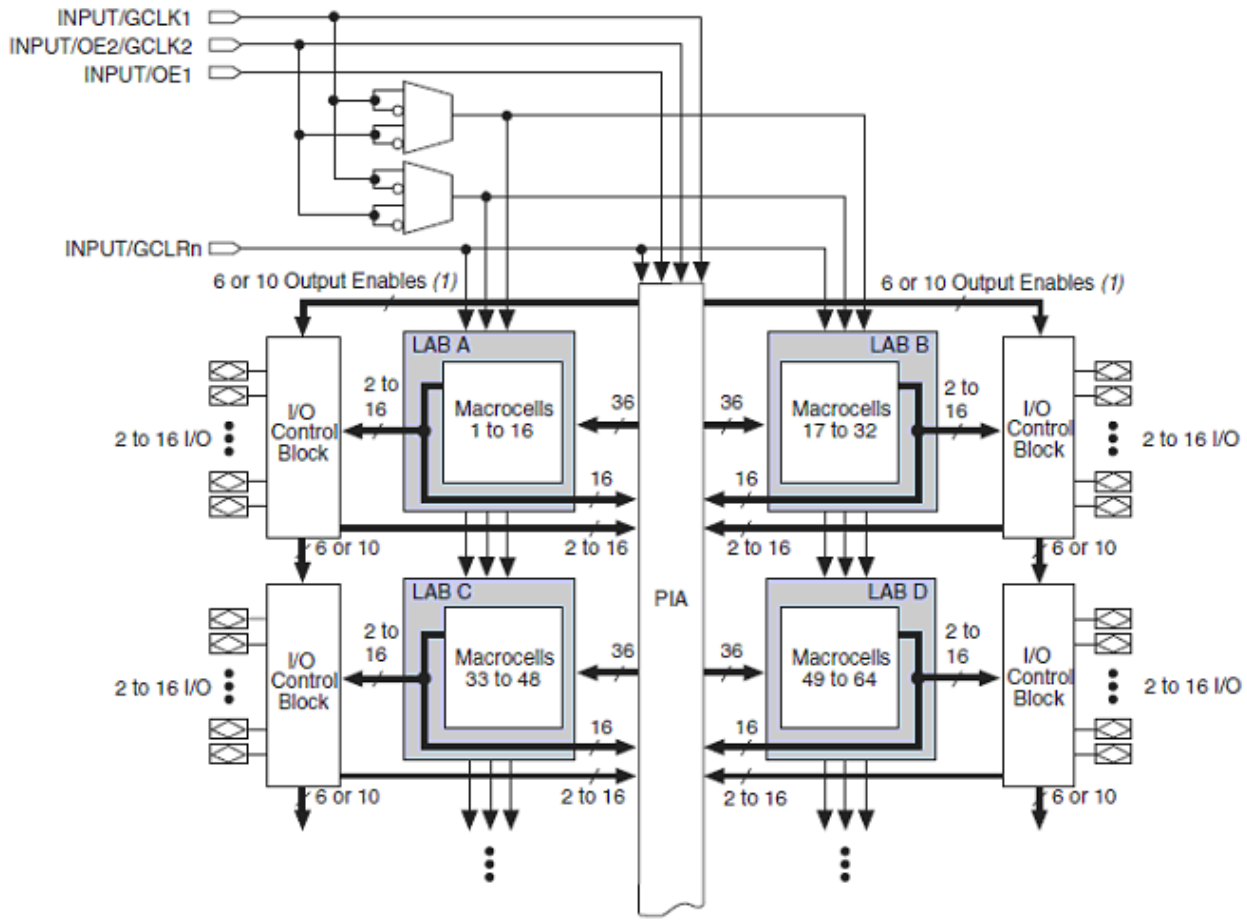


Рисунок 81. Структурная схема ПЛИС EPM3064A

ПЛИС семейства MAX 3000A содержат:

- Логические блоки (Logic array blocks, LABs), которые обеспечивают построение как комбинационных цепей, так и схем с элементами памяти.
- Макроячейки (Macrocells). Макроячейка (МЯ) содержит три функциональных блока: матрица логических элементов (вентилей), матрица распределения термов, программируемый регистр

(регистровый выход). Массив элементов И является программируемым, массив элементов ИЛИ – фиксированным.
- Логический расширитель термов (Expander product terms). Если для реализации функции МЯ недостаточно числа ее собственных термов, то можно воспользоваться дополнительными ресурсами двух типов ЛР: общий (разделяемый) и параллельный.
- Программируемая матрица соединений (Programmable interconnect array, PIA)
- Блоки ввода-вывода (I/O control blocks)

В очень упрощенном виде ПЛИС представляет собой набор макроячеек и механизм для организации связи между ними (PIA). Информация о связях между макроячейками хранится в энергонезависимой памяти находящейся внутри самой микросхемы. Для программирования EPM3064A используется специальная САПР QuartusII (Altera). Логическая схема расширителя портов ввода-вывода была нарисована в этой САПР и преобразована в базис макроячеек ПЛИС и, далее, в конфигурационный файл, необходимый для соединения нужных логических ячеек ПЛИС. Конфигурационный файл доставляется в память ПЛИС через интерфейс JTAG (IEEE 1149.1).

В стенде SDK-1.1 ввод-вывод данных осуществляется с помощью портов микроконтроллера (см. подраздел 4.5.2) и микросхемы ПЛИС, которая имеет 8 регистров, отображаемых во внешнее адресное пространство данных процессора. В старых версиях стенда SDK-1.1 используется упрощенная конфигурация регистров микросхемы ПЛИС. В описании регистров будут указаны различия между полной и упрощенной конфигурациями регистров ПЛИС. К ПЛИС в SDK-1.1 подключены:

- Клавиатура
- ЖКИ
- Линейка светодиодов
- Звуковой излучатель
- 16 дискретных портов ввода-вывода

Таблица 14. Перечень регистров расширителя портов ввода-вывода

Адрес	Регистр	Доступ	Назначение
080000H	KB	R/W	Регистр клавиатуры.
080001H	DATA_IND	R/W	Регистр шины данных ЖКИ.
080002H	EXT_LO	R/W	Регистр данных параллельного порта (разряды 0..7).
080003H	EXT_HI	R/W	Регистр данных параллельного порта (разряды 8..15).
080004H	ENA	W	Регистр управления портами ввода-вывода, звуком, сигналом INT0 и прерыванием от клавиатуры.
080006H	C_IND	W	Регистр управления ЖКИ.
080007H	SV	W	Регистр управления светодиодами.

4.6.1 Регистр клавиатуры KB

Адрес 080000H. Значение после сброса xxxx1111B (полная конфигурация) и xxxx0000B (упрощенная конфигурация).

Таблица 15. Регистр клавиатуры KB

7	6	5	4	3	2	1	0
R	R	R	R	R/W	R/W	R/W	R/W
ROW				COL			

Таблица 16. Назначение битов регистра KB

Биты	Поле	Описание
0..3	COL	Поле предназначено для сканирования клавиатуры (колонки матрицы). Сканирование производится посредством записи логического «0» в один из разрядов поля. В полной конфигурации ПЛИС из этой тетрады считывается записанное ранее в нее значение. В упрощенной конфигурации всегда возвращается «0».
4..7	ROW	Поле предназначено для считывания данных с клавиатурной матрицы (строки). Если ни одна из кнопок в строке не нажата, все биты поля ROW содержат логические «1». Если кнопка нажата и на ее колонку подан логический «0», то в поле ROW также появится логический «0».

Данный регистр можно использовать для определения типа конфигурации. Если при старте программы записать в тетраду COL регистра KB ненулевое значение, а затем его считать, то в полной конфигурации будет считано записанное значение, а в упрощенной – нули.

В полной конфигурации ПЛИС работа с клавиатурой может быть организована по прерыванию: 6-й бит (KB) регистра ENA заведен на вход внешнего прерывания INT0 МК ADuC812. Внешнее прерывание будет возникать, как только на клавиатуре будет нажата хоть одна клавиша (любая). Таким образом, чтобы работать с клавиатурой по ее собственному прерыванию, необходимо:

- Разрешить прерывание от клавиатуры (6-й бит регистра ENA должен быть «1»);
- Настроить внешнее прерывание INT0 так, чтобы оно работало по спаду, а не по уровню (регистр специального назначения TCON);
- Разрешить внешнее прерывание INT0 (бит EX0 = 1 в регистре специального назначения IE);
- Написать обработчик внешнего прерывания INT0;
- Переключатели SW3 должны быть в положении OFF, а 5-й бит регистра ENA (ПЛИС) должен быть «1»;
- Написать алгоритм сканирования клавиатуры.

4.6.2 Регистр шины данных ЖКИ DATA_IND

Адрес 080001H. Значение после сброса 00000000B.

Таблица 17. Регистр шины данных ЖКИ DATA_IND

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 18. Назначение битов регистра DATA_IND

Биты	Поле	Описание
0..7	D0..D7	Регистр DATA_IND позволяет устанавливать данные на шине данных ЖКИ и считывать их оттуда. Для организации взаимодействия с ЖКИ (формирования временных диаграмм чтения и записи) необходимо использование регистра C_IND.

4.6.3 Регистр данных параллельного порта EXT_LO

Адрес 080002H. Значение после сброса 00000000B.

Таблица 19. Регистр данных параллельного порта EXT_LO

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 20. Назначение битов регистра EXT_LO

Биты	Поле	Описание
0..7	D0..D7	Регистр EXT_LO позволяет считывать и записывать биты 0..7 параллельного порта. Для того чтобы данные из регистра попали на выход, необходимо установить бит EN_LO в логическую «1» (см. регистр ENA). Для чтения данных необходимо установить этот бит в логический «0».

4.6.4 Регистр данных параллельного порта EXT_HI

Адрес 080003H. Значение после сброса 00000000B.

Таблица 21. Регистр данных параллельного порта EXT_HI

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 22. Назначение битов регистра EXT_HI

Биты	Поле	Описание
0..7	D0..D7	Регистр EXT_HI позволяет считывать и записывать биты 8..15 параллельного порта. Для того, чтобы данные из регистра попали на выход, необходимо установить бит EN_LO или EN_HI в логическую «1» (см. регистр ENA). Для чтения данных необходимо установить этот бит в логический «0».

4.6.5 Регистр управления ENA

Адрес 080004H. Значение после сброса x0100000B.

Таблица 23. Регистр управления ENA

7	6	5	4	3	2	1	0
-	W	W	W	W	W	W	W
-	KB	INT0	EPMSND2	EPMSND1	EPMSND0	EN_HI	EN_LO

Таблица 24. Назначение битов регистра ENA

Биты	Поле	Описание
0	EN_LO	В полной конфигурации бит EN_LO нужен для управления младшими 8 разрядами (биты 0..7) 16-разрядного порта ввода-вывода. Если записать в EN_LO логический «0», то порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из EXT_LO. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистр EXT_LO, попадают на выход порта ввода-вывода. В упрощенной конфигурации этот бит управляет всеми 16 разрядами порта ввода-вывода. Если записать в EN_LO логический «0», то весь порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из регистров EXT_LO и EXT_HI. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистры EXT_LO и EXT_HI, попадают на выход порта ввода-вывода.
1	EN_HI	В полной конфигурации бит EN_HI нужен для управления старшими 8 разрядами (биты 8..15) 16-разрядного порта ввода-вывода. Если записать в EN_HI логический «0», то порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из EXT_HI. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистр EXT_HI, попадают на выход порта ввода-вывода. В упрощенной конфигурации бит EN_HI не влияет на функционирование порта ввода-вывода. Все управление портом производится битом EN_LO.
2..4	EPMSND0-EPMSND2	Выход звукового ЦАП. Задает уровень напряжения на динамике. Позволяет формировать звуковые сигналы различной тональности и громкости.
5	INT0	При записи логического «0» в этот бит на вход INT0 ADuC812 также попадает логический «0». Бит можно использовать для

		формирования внешнего прерывания для микроконтроллера.
6	KB	В полной конфигурации при записи логического «0» прерывание от клавиатуры запрещается. Если бит установлен в «1», то прерывание от клавиатуры разрешено. В упрощенной конфигурации бит KB всегда равен нулю, т.е. прерывание клавиатуры запрещено.

4.6.6 Регистр управления ЖКИ C_IND

Адрес 080006H. Значение после сброса xxxxx010B.

Таблица 25. Регистр управления ЖКИ C_IND

7	6	5	4	3	2	1	0
-	-	-	-	-	W	W	W
-	-	-	-	-	RS	RW	E

Таблица 26. Назначение битов регистра C_IND

Биты	Поле	Описание
0	E	Бит управления входом «E» ЖКИ. Наличие положительного импульса на входе «E» позволяет зафиксировать данные на шине ЖКИ (данные, сигналы RW и RS к этому моменту должны быть уже установлены).
1	RW	Бит переключения шины данных ЖКИ на чтение или запись: 0 – запись, 1 – чтение.
2	RS	Бит переключения режимов команды/данные ЖКИ: 1 – данные, 0 – команды.

4.6.7 Регистр управления светодиодами SV

Адрес 080007H. Значение после сброса 00000000B.

Таблица 27. Регистр управления светодиодами SV

7	6	5	4	3	2	1	0
W	W	W	W	W	W	W	W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 28. Назначение битов регистра SV

Биты	Поле	Описание
0..7	D0..D7	Биты управления светодиодами. Подача логической «1» зажигает светодиоды, «0» – гасит.

4.6.8 Логическая схема ПЛИС: доступ к периферийным устройствам

Мы уже обсудили такие понятия, как реализация расширителя портов ввода-вывода на базе ПЛИС, интерфейс подключения ПЛИС к МК ADuC812,

регистровая модель ПЛИС для программирования подключенных к ней периферийных устройств. Теперь рассмотрим логическую схему ПЛИС, которая нам поможет понять, что же происходит при доступе к тому или иному периферийному устройству внутри самой ПЛИС. Для примера возьмем зажигание светодиодов.

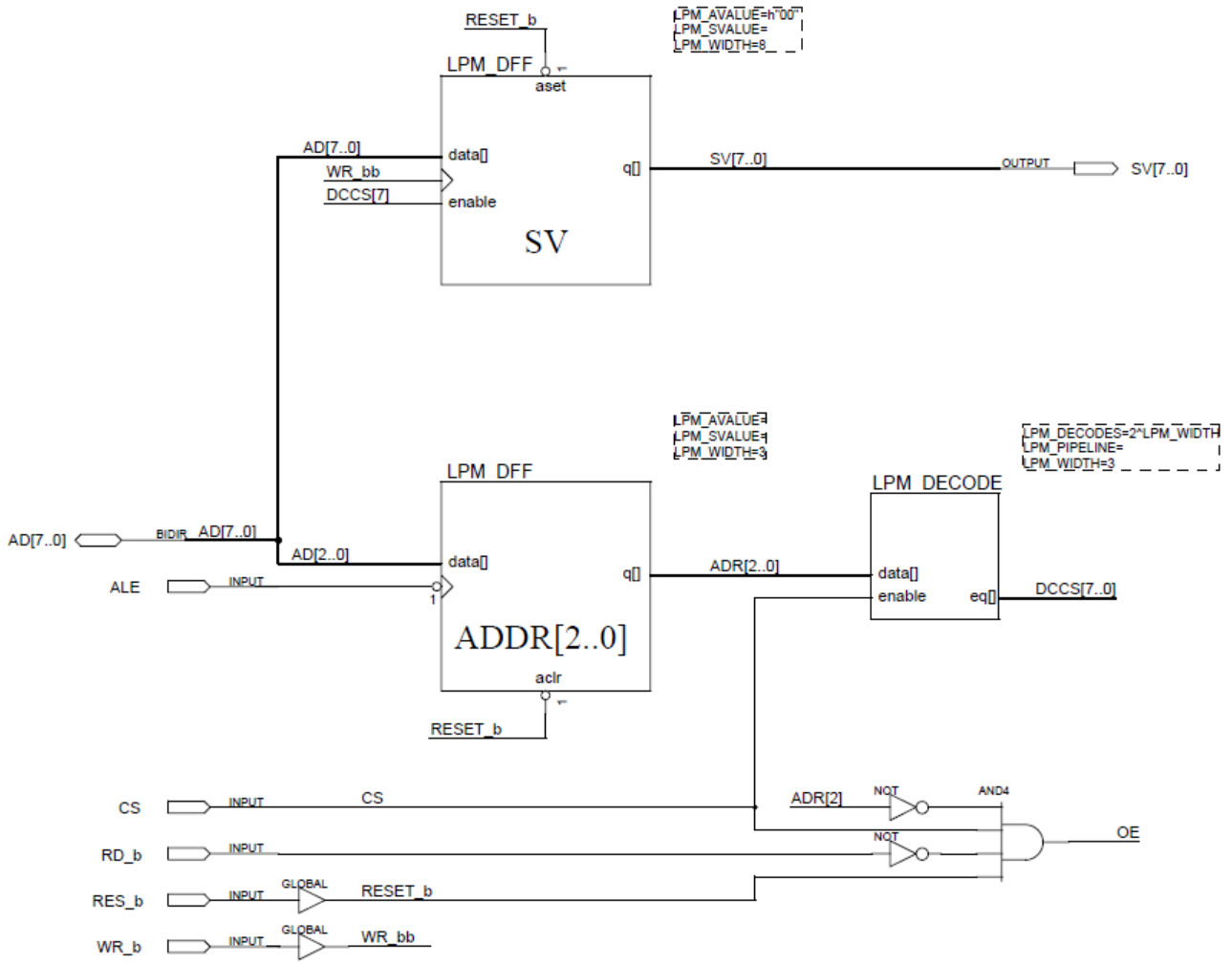


Рисунок 82. Логическая схема ПЛИС (регистр управления светодиодами)

МК ADuC812 выбирает восьмую страницу внешней памяти и в ней седьмой регистр, т.е. полный адрес составляет 0x080007. Младшая часть этого адреса выставляется на мультиплексированную шину адреса/данных порта P0 и попадает на вход ПЛИС (AD[7..0]). Младшие три бита адреса регистра поступают на вход регистра-защелки, который сохраняет этот адрес по сигналу ALE. Далее три разряда (ADR[2..0]) попадают на вход дешифратора, который по сигналу CS активизирует один из выходных разрядов в соответствии с преобразованием в позиционный код «1 из N». В случае светодиодов DCCS[7] становится равным «1» и поступает на вход регистра-защелки. По сигналу записи WR байт с шины данных (от порта P0) сохраняется в регистре и подается на выводы ПЛИС SV[7..0], к которым, в свою очередь, подключены светодиоды.

Вопросы доступа к регистрам ПЛИС на программном уровне рассматриваются в подразделе 6.2 на примере программирования светодиодных индикаторов.

4.6.9 Жидкокристаллический индикатор

4.6.9.1 Историческая справка

В настоящее время жидкокристаллические индикаторы (ЖКИ) являются наиболее распространённым видом индикаторов. Хотя сами жидкие кристаллы (ЖК) были известны химикам еще с 1888 г., но только 1960-х годов началось их практическое использование. В 1990 г. Де Жен получил Нобелевскую премию за теорию жидких молекулярных кристаллов.

Термином жидкий кристалл обозначается мезофаза[5] между твердым состоянием и изотропным жидким состоянием, при этом мезофаза сохраняет фундаментальные свойства присущие двум состояниям материи. Жидкие кристаллы, с одной стороны, обладает текучестью как изотропная жидкость, с другой стороны, сохраняет определенный порядок в расположении молекул (как кристалл).

В отдельных случаях мезофаза оказывается стабильной в широкой области температур, включая комнатную, тогда говорят о жидких кристаллах. Большинство жидких кристаллов образуются стержневыми молекулами.

Обычно жидкокристаллический дисплей представляет собой стеклянную кювету толщиной меньше 20 мкм, в которую помещен жидкий кристалл. Направление молекул жидкого кристалла может быть задано обработкой поверхностей кюветы таким образом, чтобы молекулы ЖК выстраивались в определенном направлении – параллельно плоскости кюветы или перпендикулярно к ней. Один из способов обработки поверхности заключается в нанесении на нее тонкого слоя твердого полимера и последующего «натирания» его в одном направлении.

Используя различные ориентации направления молекул жидкого кристалла первоначально с помощью поверхностного упорядочения, а затем с помощью электрического поля, можно сконструировать простейший дисплей. Жидкокристаллический дисплей состоит из несколько слоев, где ключевую роль играют две стеклянные панели, между которыми помещён жидкий кристалл.

На панели наносятся бороздки. Бороздки получаются в результате размещения на стеклянной поверхности тонких пленок из прозрачного пластика, который затем специальным образом обрабатывается. Бороздки расположены таким образом, что они параллельны на каждой панели, но перпендикулярны бороздкам соседней панели. Соприкасаясь с бороздками,

[5] Мезофаза – промежуточная фаза состояния вещества, в котором оно имеет свойства жидкости и твёрдого тела одновременно.

молекулы в жидких кристаллах ориентируются одинаково по всей поверхности. В результате направление ориентации молекул жидкого кристалла поворачивается от верхней панели к нижней на 90°, вращая, таким образом, плоскость поляризации света. Изображение формируется при помощи поляризационных плёнок, размещённых над и под жидкокристаллическим дисплеем. Если оси поляризации этих плёнок перпендикулярны друг другу, то дисплей будет прозрачным.

На стеклянные панели наносится тонкий слой металла, образующий электроды. Если теперь к электродам подвести напряжение, то молекулы жидкого кристалла развернутся вдоль электрического поля, вращение плоскости поляризации исчезнет, и свет не сможет пройти через поляризационные плёнки.

Напряжение, необходимое для поворота директора составляет обычно 2В-5В. Важно, что действие электрического поля не связано с дипольным моментом молекулы и поэтому не зависит от направления поля. Это позволяет использовать для управления индикатором переменное поле. Постоянное поле может приводить к электролизу жидкого кристалла и, в конечном итоге, выходу прибора из строя.

Электроды на жидкокристаллический индикатор наносятся в виде точек, пиктограмм или сегментов для отображения различных видов информации, как это уже обсуждалось ранее.

Важным параметром индикатора является время релаксации - время, необходимое для возвращения молекул жидкого кристалла в исходное состояние после выключения поля. Оно определяется поворотом молекул и составляет 30-50 мс. Такое время достаточно для работы различных индикаторов, но на несколько порядков превышает время, необходимое для работы компьютерного монитора.

Время релаксации резко зависит от температуры жидкокристаллического индикатора. Именно временем релаксации определяется минимальная температура использования жидкокристаллических индикаторов. Время релаксации современных жидкокристаллических индикаторов при температуре -25°С достигает нескольких секунд. Это время смены информации неприемлемо для большинства практических приложений.

Не менее важным параметром жидкокристаллического индикатора является контрастность изображения. При нормальной температуре контрастность изображения достигает нескольких сотен. При повышении температуры контрастность изображения падает и при температуре порядка +50°С изображение становится практически неразличимым.

Следующий параметр, характеризующий жидкокристаллический индикатор – это угол обзора. Угол обзора жидкокристаллического индикатора существенно зависит от скважности динамического режима индикации. Чем больше скважность, тем меньше получается угол обзора индикатора.

В современных жидкокристаллических компьютерных мониторах используется специальный метод формирования статического формирования изображения при динамическом способе его подачи на дисплей. Это TFT технология. При использовании этой технологии около каждого элемента изображения формируется запоминающий конденсатор и ключевой транзистор, который подключает этот конденсатор к цепям формирования изображения только в момент подачи информации именно для этого элемента изображения.

Особенностью работы жидкокристаллического индикатора является то, что на него следует подавать переменное напряжение. Это связано с тем, что при подаче на жидкокристаллический индикатор постоянного напряжения происходит электролиз жидкого кристалла и индикатор выходит из строя.

Напряжение для работы жидкокристаллического индикатора формируется логическими элементами, поэтому обычно используется прямоугольное колебание со скважностью равной двум. Его легко можно получить на выходе делителя частоты на два.

Теперь вспомним, что логические сигналы содержат постоянную составляющую. Ее можно убрать, подав сигнал на выводы жидкокристаллической ячейки в противофазе друг другу.

Если ячейку жидкокристаллического индикатора следует оставить прозрачной, то на ее выводы подаются синфазные напряжения. В результате разность потенциалов получается равной нулю [35].

4.6.9.2 Подключение ЖКИ

В ЖКИ SDK-1.1 есть специальный контроллер, формирующий необходимые напряжения на входах матрицы и осуществляющий динамическую индикацию. Для работы с этим контроллером реализован простейший интерфейс, описанный ниже. Матрица имеет 80 входов по горизонтали и 16 входов по вертикали[6].

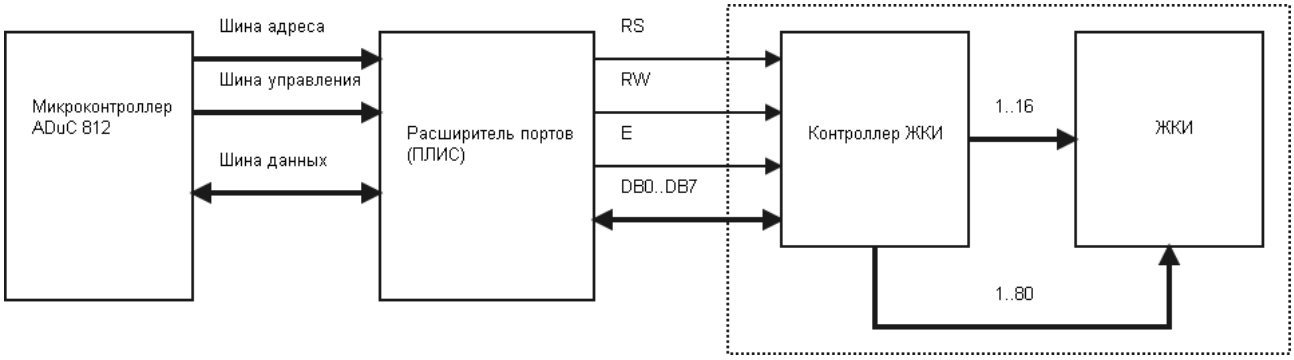


Рисунок 83. Схема подключения контроллера ЖКИ к МК ADuC812

[6] Чтобы уменьшить количество проводников используется динамическая индикация. Принцип работы динамической индикации аналогичен принципу, используемому при опросе клавиатуры учебного стенда.

Матрица ЖКИ состоит из 32 знакомест (2 строки по 16 символов) размером 5 точек по горизонтали и 8 точек по вертикали. Для отображения различных символов внутри контроллера ЖКИ есть знакогенератор[7].

Таблица 29. Интерфейс ЖКИ

Обозначение	Описание
RS	Переключение между регистрами команд и данных. 1 – данные, 0 – команды
R/W	1– чтение (из контроллера ЖКИ), 0 – запись (в контроллер ЖКИ)
E	Разрешающий сигнал (1 – активный уровень). Если сигнал E = 0, то контроллер ЖКИ игнорирует все остальные сигналы.
DB0	Бит данных 0
DB1	Бит данных 1
DB2	Бит данных 2
DB3	Бит данных 3
DB4	Бит данных 4
DB5	Бит данных 5
DB6	Бит данных 6
DB7	Бит данных 7

4.6.9.3 Контроллер ЖКИ

Основными компонентами контроллера ЖКИ являются память DDRAM (Data Display RAM), память CGRAM (Character Generator RAM), память CGROM (Character Generator ROM), счетчик адреса, регистр команд IR (Instruction Register), регистр данных DR (Data Register). Регистр команд предназначен для записи в него таких операций, как очистка дисплея, перемещение курсора, включение/выключение дисплея, а также установка адреса памяти DDRAM и CGRAM, для последующего их выполнения. Регистр данных временно хранит данные, предназначенные для записи или чтения из DDRAM или CGRAM (символы). Эти два регистра можно выбрать с помощью регистрового переключателя RS (Register Select).

Таблица 30. Варианты значений сигналов RS и R/W

RS	R/W	Команда
0	0	IR используется для внутренних команд (очистка дисплея и т.д.).
0	1	Считывание флага занятости (DB7) и счетчика адреса (от DB0 до DB7).
1	0	Запись данных в DDRAM или CGRAM (из регистра данных в DDRAM или CGRAM).
1	1	Чтение данных из DDRAM или CGRAM (из DDRAM или CGRAM в регистр данных).

[7] Знакогенератор – специальное устройство, содержащее в себе ПЗУ (или ОЗУ) с битовыми картами с изображениями различных символов. Каждому изображению символа ставится в соответствие его код.

4.6.9.4 Память данных ЖКИ (DDRAM)

Эта память используется для хранения данных, выводимых на дисплей. Один символ представлен в виде 8-битного кода. Объем памяти составляет 80×8 битов или 80 символов.

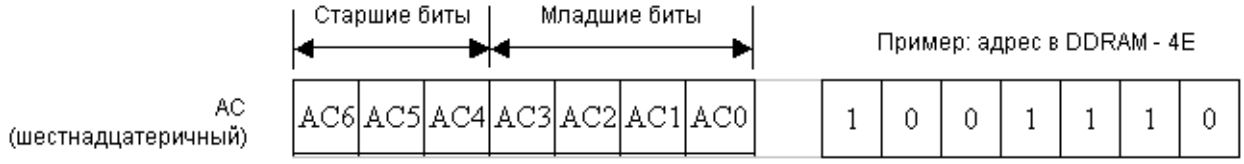


Рисунок 84. Адресация памяти DDRAM.

Ниже приведена схема соответствия между адресами DDRAM и позициями ЖКИ.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Дисплей: 2 строки по 16 символов

Рисунок 85. Соответствие между адресами DDRAM и позициями ЖКИ.

4.6.9.5 Знакогенератор, встроенный в ПЗУ (CGROM)

CGROM генерирует символы размером 5×8 или 5×10 точек на основе 8-битных кодов символов.

4.6.9.6 Знакогенератор, встроенный в ОЗУ (CGRAM)

В CGRAM пользователь может программно генерировать символы. Можно определить 8 символов размером 5×8 точек и 4 символа размером 5×10 точек. Коды символов нужно записывать в DDRAM по адресам, отображенным в таблице.

4.6.9.7 Счетчик адреса (AC)

Счетчик адреса (AC) назначает адреса и DDRAM, и CGRAM.

4.6.9.8 Флаг занятости (BF)

Если флаг занятости равен 1, это значит, что БИС занята выполнением внутренних операций и следующая команда не может быть принята. Если RS=0 и R/W=1, содержимое флага занятости передается в бит DB7. Следующая команда должна быть записана только при значении флага занятости, равном 0.

4.6.9.9 Таблица команд контроллера ЖКИ

Команда	Код операции										Описание
	RS	**R/W**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	
Очистка экрана	0	0	0	0	0	0	0	0	0	1	Запись “00H” в DDRAM и установка адреса DDRAM на “00H” из AC.
Возврат в начало строки	0	0	0	0	0	0	0	0	1	*[8]	Установка адреса DDRAM на “00H” из AC и возврат курсора в начало строки, если он был смещен. Содержимое DDRAM не меняется.
Начальные установки	0	0	0	0	0	0	0	1	I/D	SH	Задает направление перемещения курсора и разрешает сдвиг сразу всех символов.
Дисплей ON/OFF	0	0	0	0	0	0	1	D	C	B	Устанавливает / отключает биты, отвечающие за режим дисплея (D), отображение курсора (C), мерцание курсора (B).
Передвиж. курсора по экрану	0	0	0	0	0	1	S/C	R/L	*	*	Установка бита движения курсора и смещения всех символов, указание направления смещения без изменения данных в DDRAM.
Функц. установки	0	0	0	0	1	DL	N	F	*	*	Установка длины данных (DL:8-бит/4-бита), количества строк на дисплее (N:2-строки или 1) и размера символов (F:5×11 точек/5×8 точек).
Установка адреса CGRAM	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Установка адреса CGRAM в счетчик адреса.
Установка адреса DDRAM	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Установка адреса DDRAM в счетчик адреса.
Чтение флага занятости и адреса	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Прочитав флаг занятости, можно определить, занят ли контроллер выполнением внутренних операций. Также можно прочесть содержимое счетчика адреса.
Записать данные в память	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Запись данных во внутреннюю память (DDRAM/CGRAM).

[8] «*» – Не имеет значения

Чтение данных из памяти	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Чтение данных из внутренней памяти (DDRAM/CGRAM).

4.6.9.10 Операции чтения и записи команд/данных

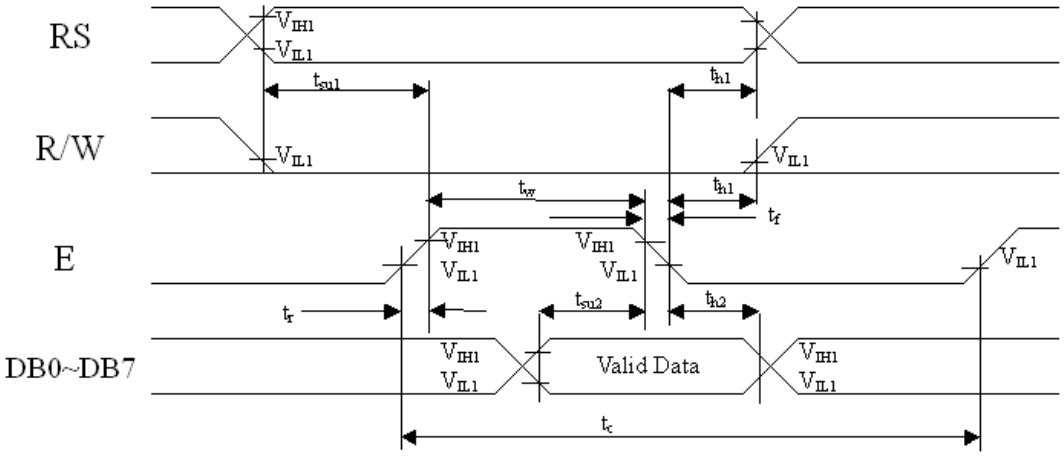


Рисунок 86. Запись команды/данных в ЖКИ

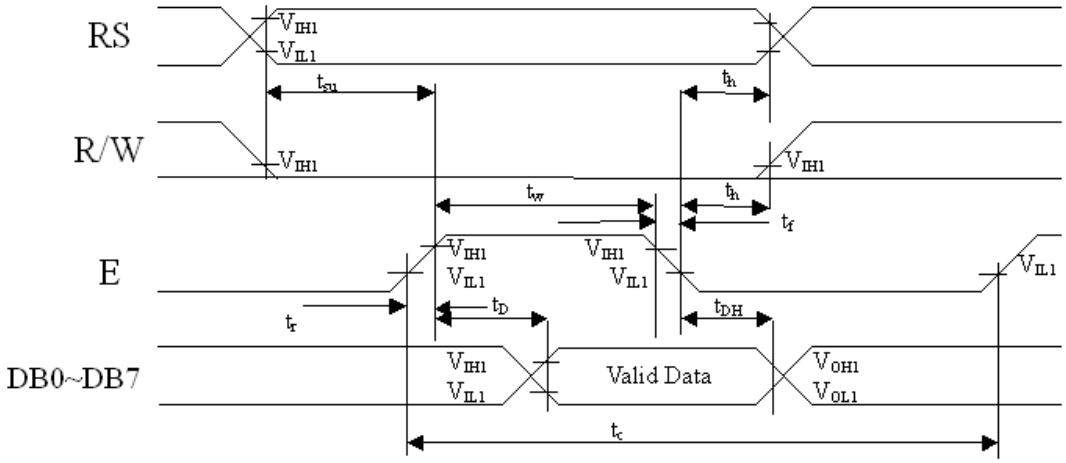


Рисунок 87. Чтение данных из ЖКИ

Пример программирования ЖКИ стенда SDK-1.1 приведен в подразделе 6.5.

4.7 Внешняя память программ и данных

Внешняя память программ и данных стенда SDK-1.1 (см. подраздел 4.4.2) – статическое ОЗУ (SRAM), имеющее страничную организацию и предназначенное для размещения пользовательских программ и данных. Подключается к МК ADuC812 по системной шине (как и ПЛИС).

Так как МК ADuC812 имеет Гарвардскую архитектуру, то доступ к внешней памяти по чтению команд и данных организован по-разному.

Используется одна и та же шина адреса и данных, а шина управления отличается.

Сначала рассмотрим цикл чтения команды из внешней памяти программ:

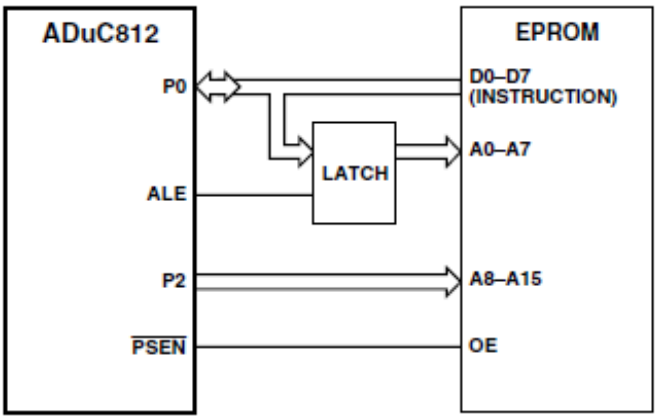


Рисунок 88. Интерфейс подключения внешней памяти программ к МК ADuC812

В такой схеме для портов P0 и P2 должна быть выбрана альтернативная функция (т.е. не порты ввода-вывода), при этом порт P0 превращается мультиплексированную шину адреса/данных. Сначала через P0 передается младший байт счетчика команд, который запоминается регистром-защелкой (Latch) по синхроимпульсу ALE (Address Latch Enable). Через P2 передается старший байт счетчика команд. Далее вырабатывается сигнал PSEN (Program Store Enable), по которому из внешней памяти программ считывается команда в ADuC812 через P0.

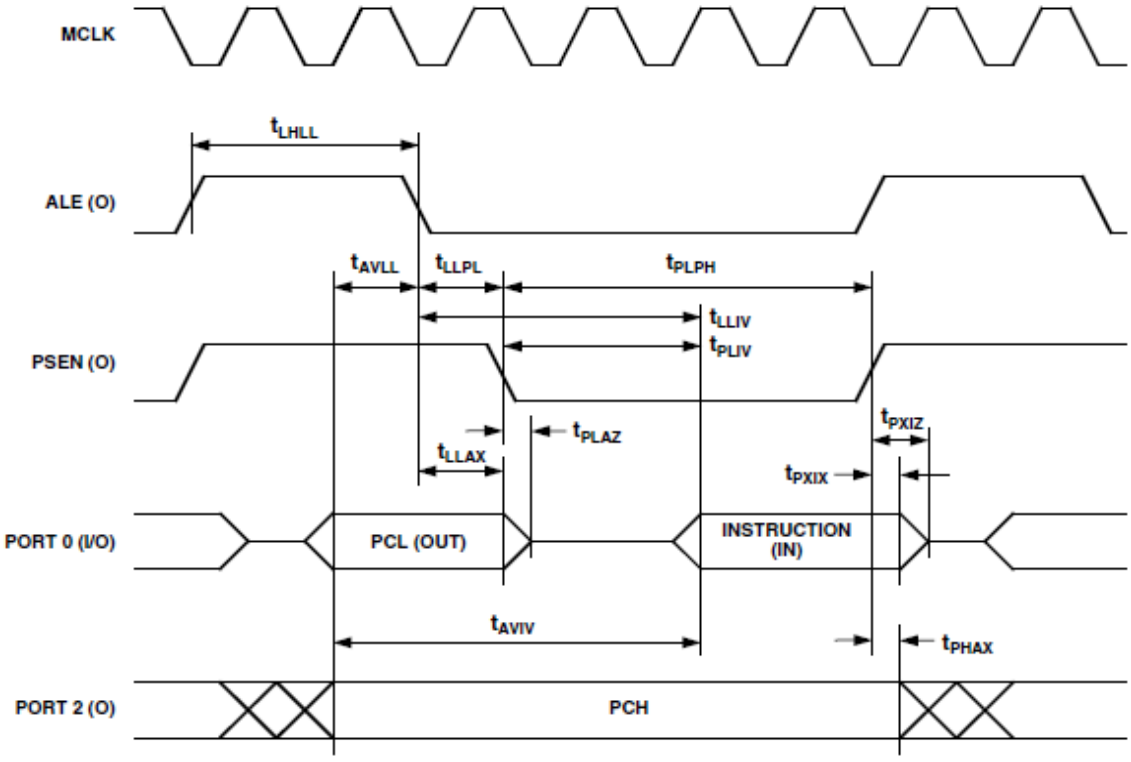


Рисунок 89. Диаграмма цикла чтения команды из внешней памяти программ

Рассмотрим схему подключения внешней памяти данных объемом 16 Мб (больше 64 Кб) к МК ADuC812:

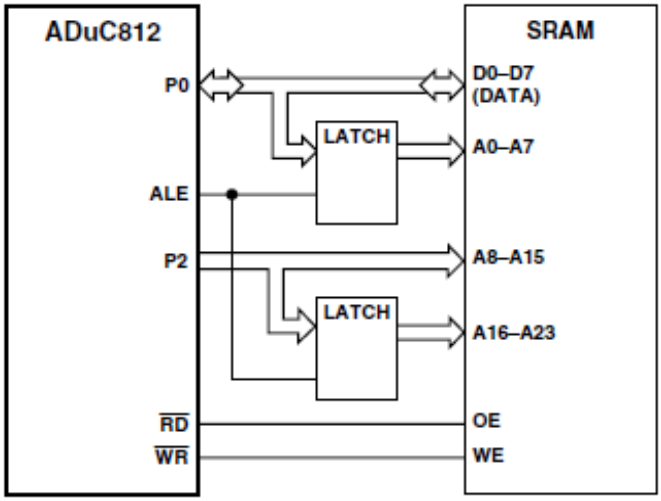


Рисунок 90. Интерфейс подключения внешней памяти данных (16 Мб) к МК ADuC812

В такой схеме подключения используется два регистра-защелки. Цикл чтения байта данных организован следующим образом: через порт P0 выставляется младший байт адреса (содержимое SFR-регистра DPL), а через порт P2 — старший байт адреса (регистр DPP). Каждый байт адреса сохраняется по сигналу ALE в соответствующем регистре-защелке. Далее через порт P2 выставляется средний байт адреса (регистр DPH), и по сигналу RD ADuC812 читает байт данных через порт P0.

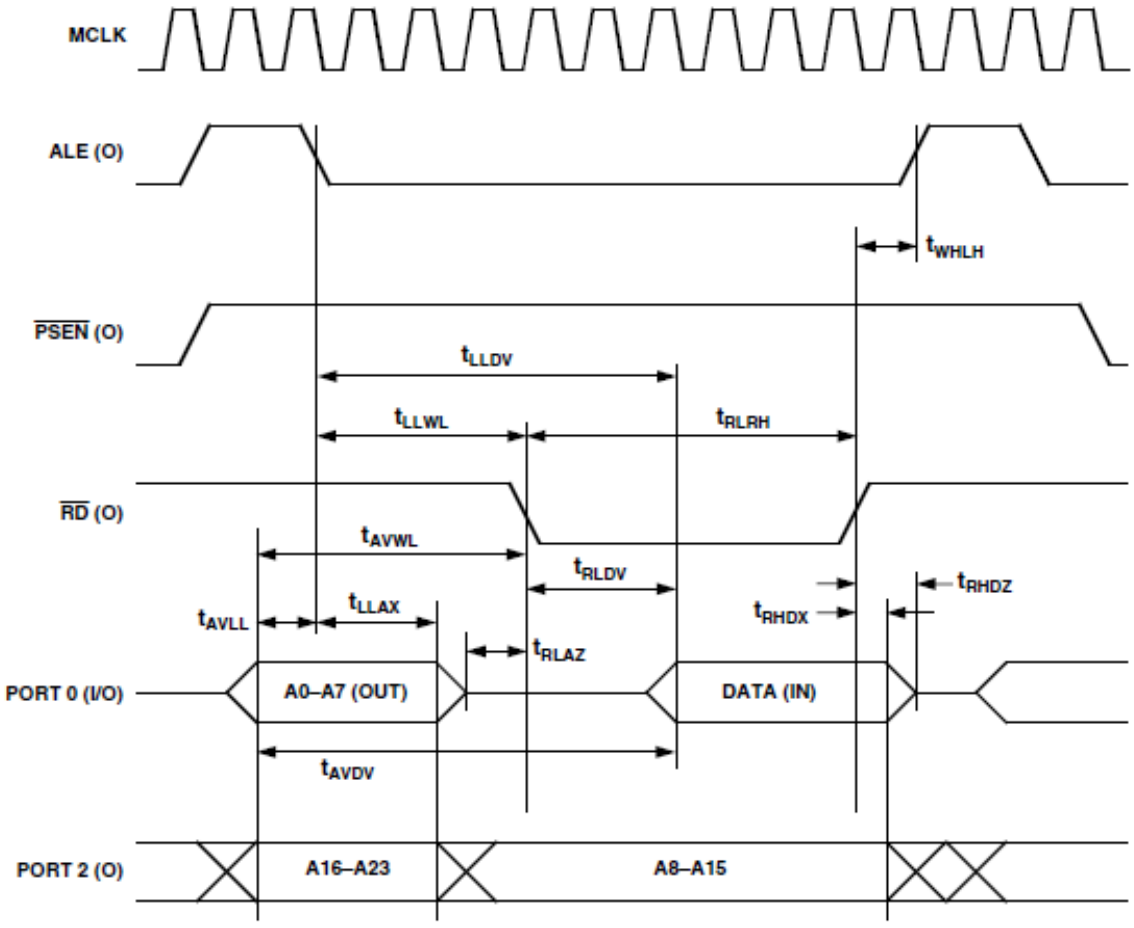


Рисунок 91. Диаграмма цикла чтения байта из внешней памяти данных

Если второго регистра-защелки нет, то объем доступной внешней памяти данных равен 64 Кб.

Цикл записи байта во внешнюю память данных выполняется аналогично, только вместо сигнала RD используется сигнал WR, а через порт P0 передается записываемый байт.

Таким образом, для обращения к внешней памяти данных и к памяти программ используются одни и те же шина адреса и шина данных, но разные управляющие сигналы. Для чтения памяти программ вырабатывается сигнал PSEN (Program Store Enable), а для чтения памяти данных вырабатывается сигнал RD. Для записи информации в память данных вырабатывается сигнал WR. То есть память программ доступна только для чтения, а память данных доступна и для чтения, и для записи любой информации, записанной в двоичном коде.

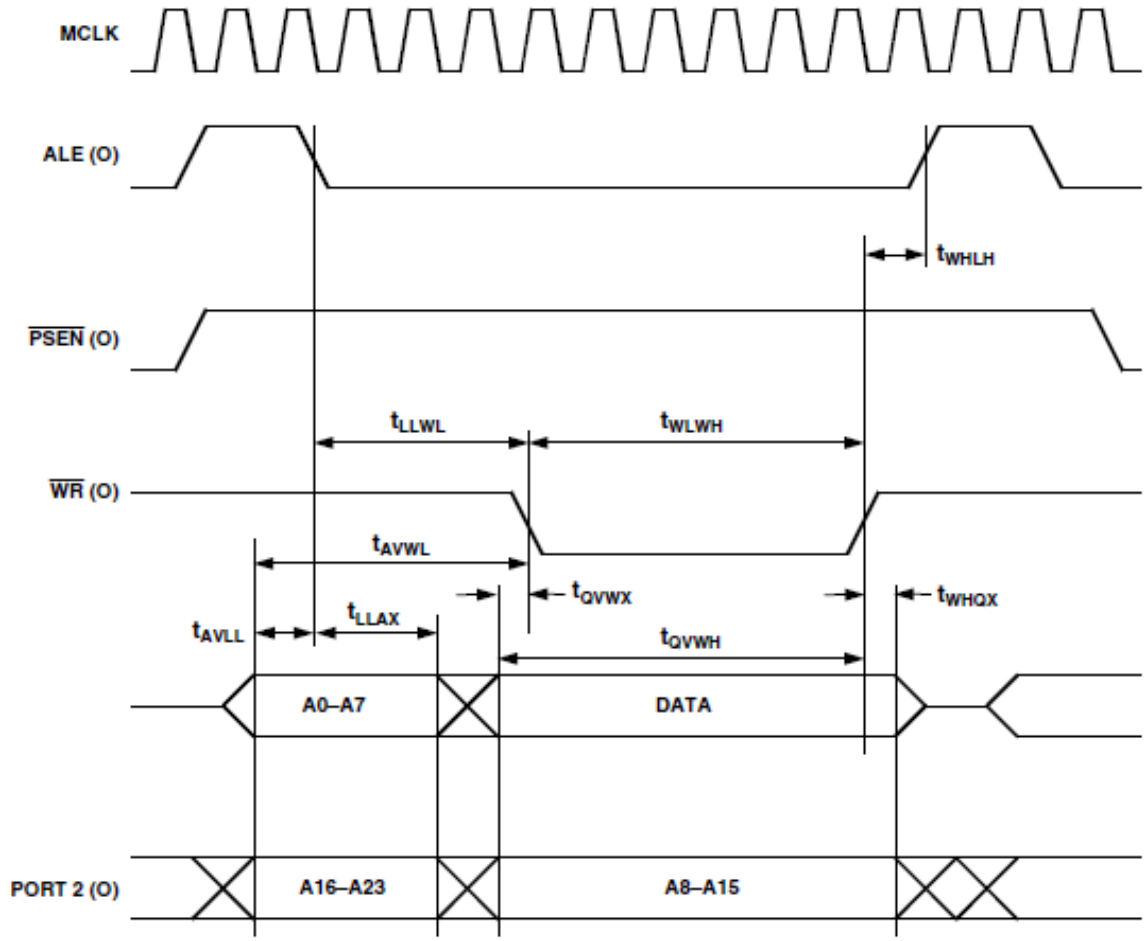


Рисунок 92. Диаграмма цикла записи байта во внешнюю память данных

5 Инструментальные средства для работы со стендом SDK-1.1

В данной главе рассматривается инструментальная цепочка программирования контроллера SDK-1.1, в которую входят бесплатные с открытым исходным кодом инструментальные средства по разработке программного обеспечения и доставке исполняемых модулей в учебный стенд SDK-1.1.

5.1 Программирование стенда SDK-1.1

Для программирования стенда может использоваться любой транслятор ассемблера или C для ядра 8051, например, SDCC или пакет µVision от Keil Software.

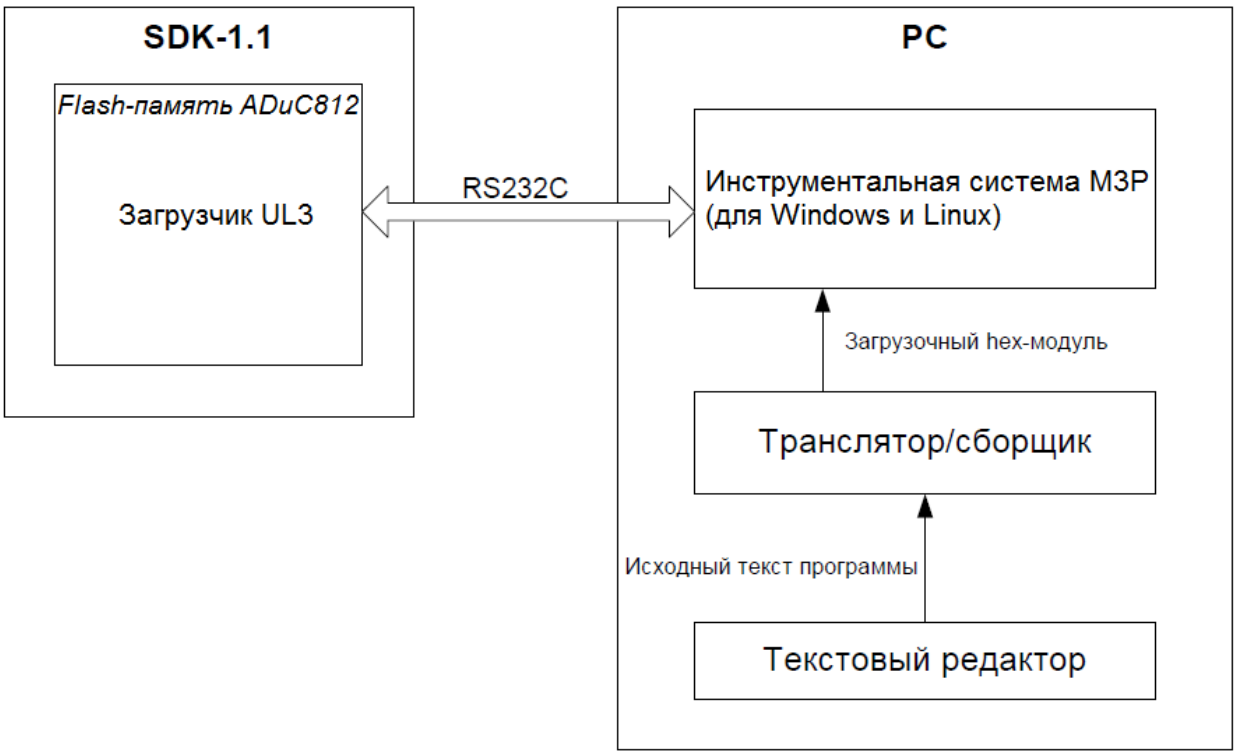


Рисунок 93. Этапы программирования стенда SDK-1.1

Основные этапы программирования стенда (с использованием компилятора SDCC):

- Подготовка программы в текстовом редакторе или среде программирования.
- Транслирование исходного текста, сборка и получение загрузочного HEX-модуля программы при помощи компилятора SDCC.
- Подготовка и загрузка исполняемого модуля в стенд через интерфейс RS-232C с помощью инструментальной системы (M3P). Под подготовкой понимается преобразование загрузочного модуля из HEX-формата в бинарный образ.

- Прием и обработка исполняемого модуля резидентным загрузчиком UL3, который находится во Flash-памяти стенда SDK-1.1. Загрузчик записывает программу во внешнюю память программ и данных по указанному адресу и передает ей управление тоже по указанному адресу.

Далее будут рассмотрены все программные средства описанной инструментальной цепочки программирования стенда SDK-1.1, которые используются на стороне персонального компьютера.

5.2 Компилятор SDCC

SDCC (Small Device C Compiler) – бесплатный с открытым исходным кодом, переносимый, оптимизирующий ANSI C компилятор для 8-разрядных микроконтроллеров на базе Intel MCS51, Maxim 80DS390, Zilog Z80, Motorola 68HC08. Распространяется под лицензией GNU GPL. SDCC использует бесплатные, способные полностью перенастраиваться на новые платформы ассемблер и линкер. SDCC обладает расширенным набором команд, основанном на оригинальном наборе команд микроконтроллеров на базе Intel 8051, что позволяет ему более эффективно использовать аппаратные возможности конкретной платформы [16].

5.2.1 Опции командной строки компилятора

Ключ	Описание
`-I`	Добавляет каталог «директория» в начало списка каталогов, используемых для поиска заголовочных файлов. Ее можно использовать для подмены системных заголовочных файлов, подставляя ваши собственные версии, поскольку эти директории просматриваются до директорий системных заголовочных файлов. Если вы используете более чем одну опцию '-I', директории просматриваются в порядке слева направо; стандартные системные директории идут после.
`-c`	Компилировать или ассемблировать исходные файлы, но не линковать. Конечный вывод происходит в форме объектного файла для каждого исходного файла.
`-mmcs51`	Выбирает семейство микроконтроллеров Intel MCS51, для которого производится компиляция. Опция необязательна, т.к. это семейство выбирается по умолчанию. Для других микроконтроллеров соответствующие ключи компиляции необходимо указывать (-mds390, -mz80, -mhc08 и др.)

`--model-small`
`--model-medium`
`--model-large`
	Выбирает модель памяти. Модель памяти small выбирается по умолчанию. В моделях medium и large все переменные без модификатора памяти попадают по умолчанию во внешнее ОЗУ (XRAM). В модели памяти small все переменные без модификатора памяти попадают во внутреннюю память данных (RAM).

`--xstack`	Позволяет располагать стек во внешней памяти (первые 256 байт), в

	сегменте pdata.
-o <path/file>	Указывает путь/имя исполняемого файла (загрузочного модуля), в который получается в результате сборки проекта. По умолчанию это файл в формате Intel HEX.
--stack-auto	Указывает, что все функции в исходных кодах являются реентерабельными (см. раздел ниже). По умолчанию все функции являются нереентерабельными, в том числе и функции по работе с 16- и 32-разрядными переменными (операции умножения, деления и т.д.), которые входят в стандартную библиотеку SDCC и используются неявно (можно увидеть только в lst-файлах).
--code-loc <Value>	Расположение памяти программ (кода). По умолчанию адрес = 0. Значение стартового адреса (<Value>) может быть указано как в шестнадцетеричной, так и в десятичной системе счисления: --code-loc 0x8000 или --code-loc 32768.
--xram-loc <Value>	Расположение внешней памяти данных. По умолчанию адрес = 0. Значение стартового адреса (<Value>) может быть указано как в шестнадцетеричной, так и в десятичной системе счисления: --xram-loc 0x8000 или --xram-loc 32768.
--stack-loc <Value>	Расположение стека. По умолчанию стек располагается после сегмента данных во внутреннем ОЗУ (например, для МК ADuC812 вершина стека = 0x07 после старта). Значение стартового адреса (<Value>) может быть указано как в шестнадцетеричной, так и в десятичной системе счисления: --stack-loc 0x20 или --stack-loc 32.

5.2.2 Классы памяти

Компилятор SDCC поддерживает инструмент, позволяющий управлять механизмами использования памяти в микроконтроллерах семейства Intel MCS51 и создавать мощные и гибкие программы. Этот инструмент – классы памяти. Каждая переменная может принадлежать к одному из 7 классов памяти. Класс памяти указывается при помощи специального модификатора.

Таблица 31. Классы памяти (расширение языка Си для МК Intel MCS51)

Модификаторы памяти	**Описание**
data	Внутренняя память данных с прямой адресацией; самая быстрая работа с переменными (128 байт). Класс памяти по умолчанию для модели памяти small.
xdata	Внешняя память данных (64Кб-16Мб). Класс памяти по умолчанию для модели памяти large.
idata	Внутренняя память данных с косвенной адресацией; доступ ко всему адресному пространству (128/256 байт).
pdata	Внешняя память данных с косвенной адресацией (256 байт). Класс памяти по умолчанию для модели памяти medium.
code	Память программ (64 Кб).
bit	И класс памяти, и нестанадртный тип данных. Бит-адресуемая внутренняя память данных (128 бит) в диапазон адресов 20h-2Fh.
sfr/sfr16/sfr32/sbit	И класс памяти, и нестанадртный тип данных. Служит для определения регистров специального назначения (sfr – 8 разрядов,

	sfr16 – 16 разрядов, sfr32 – 32 разряда) и их битов (sbit). Эти ключевые слова используются для создания заголовочных файлов, позволяющих обращаться к регистрам специального назначения по именам.

Обращение к внутренней памяти данных происходит гораздо быстрее, чем к внешней. Поэтому переменные, которые используются чаще других, следует размещать во внутренней памяти, а остальные – во внешней. Далее приведены примеры объявления переменных в разных классах памяти:

```
__data unsigned char test_data;
__xdata unsigned char test_xdata;
__idata unsigned char test_idata;
__pdata unsigned char test_pdata;
__code unsigned char test_code;
__bit test_bit;


__sfr __at (0x80) P0; // регистр специального назначения P0 по адресу 0x80

/* 16-разрядный регистр специального назначения для Таймера 0
   Старший байт значения находится по адресу 0x8C, младший по адресу 0x8A*/
__sfr16 __at (0x8C8A) TMR0;

__sbit __at ( 0xD7 ) CY; /* CY (Carry Flag, флаг переноса в SFR PSW) */
```

Если в объявлении переменной модификатор памяти не указан, выбирается модель памяти, установленная по умолчанию. Аргументы функции и переменные класса памяти auto, которые не могут быть размещены в регистрах, также хранятся в области памяти, установленной по умолчанию.

Модель памяти, выбираемая в качестве модели по умолчанию, устанавливается с помощью опций компилятора small, medium и large.

5.2.3 Абсолютная адресация

Переменным можно присваивать не только класс памяти (data, xdata, code) но и абсолютный адрес расположения.

```
__xdata __at (0x7ffe) unsigned int chksum;
```

В примере, приведенном выше, переменная chksum будет размещена по адресу 7ffeh во внешней памяти XDATA. Необходимо заметить, что компилятор не резервирует места под переменные, определенные таким образом и проверка на отсутствия пересечений с другими данными полностью лежит на программисте. Для проверки можно использовать файлы с расширениями .lst, .rst и .map.

Если вы произведете инициализацию памяти, линкер сможет обнаружить пересечение данных.

```
__code __at (0x7ff0) char Id [5] = ''SDCC'';
```

В случае использования устройств ввода-вывода, расположенных в адресном пространстве внешней памяти, необходимо использовать ключевое слово volatile, для того, чтобы оптимизатор компилятора не заменил обращение к устройству, обращением к регистру общего назначения.

```
volatile __xdata __at (0x8000) unsigned char PORTA_8255;
```

В SDCC допускается указание абсолютного адреса расположения бита, для битовых переменных.

```
__bit __at (0x02) bvar;
```

Использование абсолютной адресации битовой памяти может вызвать путаницу, и оправдано только, если вам хочется написать универсальную программу (см. пример ниже) для нескольких комплектов аппаратных средств.

```
extern volatile __bit MOSI;  /* master out, slave in */
extern volatile __bit MISO;  /* master in, slave out */
extern volatile __bit MCLK;  /* master clock */

unsigned char spi_io(unsigned char out_byte)
{
  unsigned char i=8;
  do
  {
    MOSI = out_byte & 0x80;
    out_byte <<= 1;
    MCLK = 1;
    if (MISO)
      out_byte += 1;
    MCLK = 0;
  } while(-i);
  return out_byte;
}
```

Далее мы видим варианты определения битов для разных аппаратных средств:

```
// вариант 1
  __bit __at (0x80) MOSI;  /* I/O port 0, bit 0 */
  __bit __at (0x81) MISO;  /* I/O port 0, bit 1 */
  __bit __at (0x82) MCLK;  /* I/O port 0, bit 2 */
// вариант 2
  __bit __at (0x83) MOSI;  /* I/O port 0, bit 3 */
  __bit __at (0x91) MISO;  /* I/O port 1, bit 1 */
  __bit __at (0x92) MCLK;  /* I/O port 1, bit 2 */
```

5.2.4 Реентерабельность

В зависимости от модели памяти и количества свободного места, автоматические переменные и параметры функции могут быть помещены в стек или в пространство внешней или внутренней памяти. Последний вариант делает функцию нереентерабельной. Для того, чтобы разместить автоматические переменные в стеке можно воспользоваться опцией компилятора --stack-auto (или в тексте программы #pragma stack-auto) или ключевым словом reentrant при определении функции.

```
unsigned char foo(char i) __reentrant
{
  ...
}
```

Необходимо помнить, что в архитектуре MCS51 стек имеет очень небольшой объем. Поэтому опцией --stack-auto необходимо пользоваться экономно и с осторожностью. Для решения проблемы с размером стека, можно явно указывать тип памяти для автоматической переменной и её абсолютный адрес.

```
unsigned char foo()
{
  __xdata unsigned char i;
  __bit bvar;
  __data __at (0x31) unsigned char j;
  ...
}
```

5.2.5 Оверлеи

В нереентерабельных функциях одна и та же область памяти может быть использована повторно. SDCC использует оверлеи по умолчанию. Для отключения режима работы с оверлеями необходимо использовать #pragma nooverlay.

```
#pragma save
#pragma nooverlay

void set_error(unsigned char errcd)
{
  P3 = errcd;
}

#pragma restore
void some_isr () __interrupt (2)
{
  ...
  set_error(10);
  ...
}
```

В приведенном примере использование errcd без #pragma nooverlay привело бы к непредсказуемым последствиям.

5.2.6 Обработчики прерываний

Обработчик прерывания в SDCC имеет следующий вид:

```
void timer_isr (void) __interrupt (1) __using (1)
{
    ...
}
```

Ключевое слово __interrupt определяет номер вектора прерываний, а слово __using - номер используемого регистрового банка. Явное указание номера регистрового банка позволяет уменьшить объем данных, сохраняемых в стеке

при вызове обработчика. Предполагается естественно, что этот регистровый банк кроме обработчика никто не будет использовать.

Если обработчик прерывания изменяет какие-либо глобальные переменные, они должны быть определены с использованием ключевого слова volatile.

```
/* Номера обработчиков прерываний для ЬК ADuC812 в стенде SDK-1.1:
адрес = (номер * 8) + 3 */

#define IE0_VECTOR   0    /* 0x03 external interrupt 0 */
#define TF0_VECTOR   1    /* 0x0b timer 0 */
#define IE1_VECTOR   2    /* 0x13 external interrupt 1 */
#define TF1_VECTOR   3    /* 0x1b timer 1 */
#define SI0_VECTOR   4    /* 0x23 serial port 0 */
```

5.2.7 Критические секции

Внутри критической секции SDCC генерирует код, который запрещает (в начале секции) и восстанавливает в исходное состояние (в конце) все прерывания. Необходимо помнить, что в большинстве случаев запрещать все прерывания слишком накладно.

```
int foo () __critical
{
    ...
    ...
}
```

Ключевое слово __critical может использоваться совместно с ключевым словом reentrant.

Ключевое слово __critical может использоваться для защиты отдельных переменных.

```
__critical { i++; }
```

5.2.8 Семафоры

Архитектура MCS51 позволяет проводить атомарные действия над битовыми переменными, что позволяет успешно реализовать простой бинарный семафор. SDCC генерирует такой код, если используется приведенный ниже шаблон исходного текста.

```
volatile bit resource_is_free;

if (resource_is_free)
{
  resource_is_free=0;
  ...
  resource_is_free=1;
}
```

Пример использования бинарного семафора.

```
char x = 0;
volatile bit resource_is_free;

void sem( void )
{
  if (resource_is_free)
  {
    resource_is_free = 0;
    x = 10;
    resource_is_free = 1;
  }
}
```

Генерируемый SDCC код.

```
  jbc  _resource_is_free,00106$
  ret
00106$:
  mov  _x,#0x0A
  setb _resource_is_free
  ret
```

5.2.9 Ассемблерные вставки

Компилятор SDCC позволяет использовать ассемблерные вставки. Попробуем оптимизировать программу, представленную ниже.

```
unsigned char __far __at(0x7f00) buf [0x100];
unsigned char head, tail;

void to_buffer( unsigned char c )
{
  if( head != (unsigned char)(tail-1) ) buf [ head++ ] = c;
}
```

В примере оптимизированной функции хорошо видно, что для прямого включения ассемблерного кода необходимо использовать директивы _asm и _endasm.

```
void to_buffer_asm( unsigned char c )
{
  _asm
  mov r2,dpl
;buffer.c if( head != (unsigned char)(tail-1)
  mov a,_tail
  dec a
  mov r3,a
  mov a,_head
  cjne a,ar3,00106$
  ret
00106$:
;buffer.c buf [ head++ ] = c;
  mov r3,_head
  inc _head
  mov dpl,r3
  mov dph,#(_buf >> 8)
  mov a,r2
  movx @dptr,a
00103$:
  ret
```

```
  _endasm;
}
```

Внутри ассемблерной вставки возможно использование любых директив, понятных ассемблеру.

5.2.10 Использование меток

Внутри функции можно определять метки вида nnnn$, где n – число от 0 до 100. Метки, используемые в языке Си, не видны внутри ассемблерных вставок и наоборот. Метки в ассемблерных вставках внутри разных функций также не видны друг для друга.

```
foo() {
    /* Некоторый код на Си */
    _asm
     ; Некоторый ассемблерный код
     ljmp 0003$
    _endasm;
  /* Еще код на Си */
  clabel:  /* Встроенный ассемблер не видит эту метку */
    _asm
    0003$: ; Эта метка доступна только из встроенного ассемблера
    _endasm ;
    /* Еще код на Си */
  }
```

5.2.11 Директива __naked

Директива __naked позволяет исключить генерацию вводной части функции. Предполагается, что за сохранение контекста отвечает программист.

```
volatile data unsigned char counter;

void simpleInterrupt(void) __interrupt (1)
{
  counter++;
}

void nakedInterrupt(void) __interrupt (2) __naked
{
  _asm
   inc _counter ;Инкремент не меняет флагов, нет необходимости сохранять
                ;psw
   reti; Неоходимо явно указывать reti
  _endasm;
}
```

Без __naked получается такой код:

```
_simpleInterrupt:
  push  acc
  push  b
  push  dpl
  push  dph
  push  psw
  mov   psw,#0x00
  inc   _counter
  pop   psw
  pop   dph
```

```
  pop   dpl
  pop   b
  pop   acc
  reti
```

С __naked код выглядит так:

```
_nakedInterrupt:
    inc  _counter
    reti
```

5.2.12 Формат Intel HEX

Intel HEX – формат файла. Основным отличием этого формата от таких монстров, как ELF и COFF является крайняя простота. Формат позволяет хранить только образ памяти. Ни о каком перемещаемом коде и возможности хранения объектных файлов в этом формате речи не идет.

В настоящий момент этот формат в основном используется при программировании встроенных систем. Большинство компиляторов и линкеров умеют выдавать загрузочный модуль в этом формате. Строки файла представляют собой *текстовые записи*, в которых закодированы адреса расположения, команды и данные в шестнадцатеричной системе счисления. Изначально, HEX формат использовался для работы с перфоленточными загрузчиками. В настоящее время HEX формат используют для программирования различных контроллеров и связи с программаторами ППЗУ.

Каждую строку в HEX файле называют записью. Она состоит из следующих элементов:

- Двоеточие (:).
- Число байтов данных, содержащихся в этой записи. Занимает один байт (две шестнадцатеричных цифры), что соответствует 0…255 в десятичной системе.
- Начальный адрес блока записываемых данных – 2 байта. Этот адрес определяет абсолютное местоположение блока в EPROM.
- Один байт, обозначающий тип записи.
  - 0x00 – блок данных;
  - 0x01 – конец файла;
  - 0x02 – адрес сегмента (см. архитектуру процессора Intel x86);
  - 0x03 – стартовый адрес сегмента (см. архитектуру процессора Intel x86);
  - 0x04 – старшая часть линейного (32-разрядного) адреса;
  - 0x05 – стартовый адрес, старшая часть линейного (32-разрядного) адреса.
- Байты данных (их число указывается в поле 2).

- Последний байт в записи является контрольной суммой. Если сумма всех байтов в строке (без учёта переноса) равняется 00, строка считана правильно.
- Строка заканчивается стандартной парой CR/LF (0Dh 0Ah).
- Файл всегда завершается командой 01, (получается запись вида «:00000001FF»).

Пример HEX-файла:

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

Недостатки формата:

- Ненадежный контрольный код (вероятность ошибки 1/256).
- Нет суммарного контрольного кода для всего файла.
- Получается большой файл при HEX кодировании, что отрицательно сказывается на скорости передачи файлов в контроллер.

5.3 Инструментальная система M3P

M3P – кроссплатформенная инструментальная система со встроенным интерпретатором языка FORTH. Система M3P предназначена для решения следующего ряда задач:

- Отладки, тестирование и внутрисистемного программирования встроенных систем;
- Интеграции инструментальных средств в единую систему;
- Связывания разнородных инструментальных средств посредством языка сценариев.

5.3.1 Язык FORTH

В языке Forth используется обратная польская запись. Вместо переменных, операнды обычно хранятся в стеке. Например, для вычисления выражения 2 + 3 вам придётся написать следующее:

```
2 3 +
```

В начале на стек попадает число ‘2’, потом ‘3’. Операция ‘+’ осуществляет сложение двух чисел. Результат попадает на стек, где его можно посмотреть с помощью оператора ‘.’.

В языке Forth можно создавать новые функции. Для этого служат операторы ‘:’ и ‘;’. Напрмер, для вывода строки “Hello world!” вам придётся написать такую программу. Оператор.” выводит на экран строку, а переводит строку.

```
: hello ." Hello world!" cr ;
```

Для запуска программы напишите hello и нажмите Enter.

Forth хорошо подходит для написания сравнительно небольших программ [22, 25, 36].

5.3.2 Основные команды M3P

```
.TITLE    [FORTH] Вывод информации о данной программе.
//        [FORTH] Комментарий до конца строки
--        [FORTH] Комментарий до конца строки
(         [FORTH] Комментарий до закрывающей круглой скобки
BYE       [FORTH] Выход из программы
+         [FORTH] Сложение A и B. A B +
-         [FORTH] Вычитание (A-B). A B -
*         [FORTH] Умножнение A на B. A B *
/         [FORTH] Деление A на B. A B /
%         [FORTH] Остаток от деления A на B. A B %
AND       [FORTH] Логическое И между A и B: A B AND
OR        [FORTH] Логическое ИЛИ между A и B: A B OR
XOR       [FORTH] Исключающее ИЛИ между A и B: A B XOR
NOT       [FORTH] Инверсия A: A NOT
WORDS     [FORTH] Вывод списка команд и контекстов.
.         [FORTH] Вывод числа с вершины стека в текущей системе счисления
D.        [FORTH] Вывод числа в десятичной системе счисления
H.        [FORTH] Вывод числа в шестнадцатиричной системе счисления
2H.       [FORTH] Вывод числа в шестнадцатиричной системе счисления в
          формате .2X
4H.       [FORTH] Вывод числа в шестнадцатиричной системе счисления в
          формате .4X
8H.       [FORTH] Вывод числа в шестнадцатиричной системе счисления в
          формате .8X
:         [FORTH] Начало определения новой команды
;         [FORTH] Завершение определения новой команды
 [        [FORTH] Переход в режим исполнения
]         [FORTH] Переход в режим компиляции
C,        [FORTH] Компиляция байта на вершину словаря
,         [FORTH] Компиляция 16-ти разрядного слова на вершину словаря
DUMP      [FORTH] Вывод шестандцатеричного дампа памяти: addr n dump
HERE      [FORTH] Выдача на вершину стека адреса вершины словаря (адрес
          свободного места)
BEGIN     [FORTH] Начало цикла BEGIN-AGAIN: begin test again
AGAIN     [FORTH] Конец цикла BEGIN-AGAIN
DO        [FORTH] Начало цикла DO-LOOP (аналог цикла for для языка C):
          10 0 do i . loop
LOOP      [FORTH] Конец цикла DO LOOP
EXIT      [FORTH] Завершение цикла (аналог break в языке C)
WHILE     [FORTH] Проверка предусловия для цикла BEGIN-WHILE-REPEAT
REPEAT    [FORTH] Конец цикла BEGIN-WHILE-REPEAT
UNTIL     [FORTH] Конец цикла BEGIN-UNTIL (цикл с проверкой в конце)
I         [FORTH] Внешний счетчик цикла DO-LOOP
J         [FORTH] Средний счетчик цикла DO-LOOP
K         [FORTH] Внутренний счетчик цикла DO-LOOP
IF        [FORTH] Часть условного ветвления IF-THEN-ELSE
THEN      [FORTH] Часть условного ветвления IF-THEN-ELSE
ELSE      [FORTH] Часть условного ветвления IF-THEN-ELSE
>         [FORTH] Кладет на стек ИСТИНУ (не 0) если A > B
<         [FORTH] Кладет на стек ИСТИНУ (не 0) если A < B
>=        [FORTH] Кладет на стек ИСТИНУ (не 0) если A >= B
<=        [FORTH] Кладет на стек ИСТИНУ (не 0) если A <= B
==        [FORTH] Кладет на стек ИСТИНУ (не 0) если A = B
```

```
<>          [FORTH] Кладет на стек ИСТИНУ (не 0) если A != B
ABORT       [FORTH] Прерывает выполнение текущей программы
ALLOT       [FORTH] Захватывает N байт памяти в словаре от текущего
            свободного места
R>          [FORTH] Переносит слово из стека данных в стек возвратов
>R          [FORTH] Переносит слово из стека возвратов в стек данных
R@          [FORTH] Копирует слово с вершины стека возвратов на вершину стек
            данных
R!          [FORTH] Заменяет слово на вершине стека возвратов
SWAP        [FORTH] Меняет два слова на стеке данных местами
DUP         [FORTH] Дублирует вершину стека данных
DROP        [FORTH] Убирает вершину стека данных
S.          [FORTH] Выдает содержимое стека данных
R.          [FORTH] Выдает содержимое стека возвратов
KEY         [FORTH] Выдает код нажатой клавиши (getch() из языка C)
CR          [FORTH] Выводит на консоль коды CR LF
!           [FORTH] Запоминает слово X в словаре по адресу A: X A !
@           [FORTH] Читает слово из словаря по адресу A: A @
ROT         [FORTH] Производит ротацию трех верхних элементов стека
OVER        [FORTH] Дублирует второй сверху элемент стека данных
"           [FORTH] Завершение текстовой строки
."          [FORTH] Вывод текстовой строки на консоль на этапе исполнения
DISFORTH    [FORTH] Дизассемблирование словаря
VARIABLE    [FORTH] Задание переменной
CONSTANT    [FORTH] Задание константы
ALL         [FORTH] Выключение контекстов
FORTH       [FORTH] Переход к контексту FORTH
INST        [FORTH] Переход к контексту INST
COM         [FORTH] Переход к контексту COM
USER        [FORTH] Переход к контексту USER
'           [FORTH] Кладет на стек адрес компиляции команды. Применяется для
            disforth: ' test disforth
EMIT        [FORTH] Вывод символа X на консоль. X emit
+TERM       [FORTH] Включение вывода символов на консоль. Параметров нет.
-TERM       [FORTH] Выключение вывода символов на консоль. Параметров нет.
+ECHO       [FORTH] Включение дублирования консольного вывода в файл (см.
            команду echo). Параметров нет.
-ECHO       [FORTH] Выключение дублирования консольного вывода в файл (см.
            команду echo). Параметров нет.
ECHO        [FORTH] Создание файла для хранения копии консольного вывода (см.
            команды +echo и -echo). echo <имя_файла>
\ECHO      FORTH] Закрытие файла для хранения копии консольного вывода (см.
            команды echo, +echo и -echo). Параметров нет.
>>          [FORTH] Сдвиг числа X вправо на 1 бит. X >>
<<          [FORTH] Сдвиг числа X влево на 1 бит. X <<
?TERMINAL   [FORTH] Возвращает ИСТИНУ (не 0), если нажата кнопка на консоли
LFILE       [FORTH] загружает скрипт: lfile <имя>
@TIME       [FORTH] Кладет на стек дату и время в формате ANSI (4 байта)
.CTIME      [FORTH] Распечатывает дату и время в форме "Tue Sep 03 20:33:17
            2002". Исходные данные необходимо предоставить с помощью @time
HELP        [FORTH] Вывод справки по команде: help <имя_команды>
HELPS       [FORTH] Вывод справки по всем командам текущего контекста
HELPALL     [FORTH] Вывод справки по всем командам
SYSTEM"     [FORTH] Передача строки командному интерпретатору ОС. На стеке
            остается код возврата (errorlevel): system" строка "
DIR         [FORTH] Вывод списка файлов на консоль (вызывается системная
            команда dir или ls). dir filemask
CLOCK       [FORTH] Время в мс от начала запуска программы. Команда может
            использоваться совместно с командой ShowClock.
SHOWCLOCK   [FORTH] Выдает на консоль время в секундах (с точностью до
            десятых) прошедшее с момента запуска clock
SLEEP       [FORTH] Подвешивание потока на заданное в мс время: time_ms Sleep
```

```
VERSION_CHECK   [FORTH] Контроль версии (защита от использования новых
                скриптов старыми интерпретаторами). Если на стеке лежит
                версия большая (более  старшая) чем версия данной
                программы, то происходит завершение  работы
FORGET     [FORTH] Забыть указанное определения и все определения заданные
           позже. Имена из словаря исчезают, а память не освобождается.
           forget name
.(         [FORTH] Вывод текстовой строки до закрывающей скобки (аналог
           команды ECHO). Команда испольуется только вне определений через
           двоеточие. .(string )
OPENCHANNEL  [COM] Открытие последовательного канала: 9600 openchannel com2
CLOSECHANNEL [COM] Закрытие последовательного канала
WSIO         [COM] Вывод символа в последовательный канал
RSIO         [COM] Чтение символа из последовательного канала
?RSIO        [COM] Выдает ИСТИНУ, если в буфере приема есть символ
DEBUG        [COM] Переключение системы в отладочный режим: 1 debug
TERM         [INST] Включение эмулятора терминала:
             0 term - ASCII,
             1 - HEX,
             3 - DEC
HB166        (->) file1.hex file2.bin
             HEX - BIN преобразователь для 64K.
             пример: hexbin file.hex file.bin
HB32         (->) filename.hex filename.bin
             HEX - BIN преобразователь.
             пример: hexbin file.hex file.bin
HB32o        (->) filename.hex filename.bin
             HEX - BIN преобразователь. В отличии от HB32 отрезает пустое
             пространство из начала файла.
             пример: hexbin file.hex file.bin
HB64         addr len (->) filename.hex filename.bin
             HEX - BIN преобразователь для 64K (аналог
             hb166) Позволяет получить бинарный образ из
             фрагмента HEX файла. В отличии от HB166
             заполняет пустые места кодом 0xFF.

             пример: 0 2048 hexbin file.hex file.bin

HB_FRAGMENT32      addr len (->) filename.hex filename.bin

             HEX - BIN преобразователь для файлов с
             32-разрядным Intel HEX. Позволяет получить бинарный образ из
             фрагмента HEX файла. Заполняет пустые места кодом 0xFF.

             пример:
             0 2048 hb_fragment32 file.hex file.bin
```

5.3.3 Циклы

```
-- Бесконечный цикл BEGIN AGAIN
: test1 begin
    ." test1 "
  again ;

-- Цикл со счетчиком DO LOOP, значение счетчика цикла принимает значения от
-- 0 до 9. Аналог на языке C: for(i=0;i<10;i++) { }

: test2 10 0  do
      i .
    loop ;

-- Цикл с проверкой в начале, продолжается если перед while
-- ИСТИНА
```

```
-- (не 0)

: test3 begin  1 while

    ." test3 "
  repeat ;

-- Цикл с проверкой в конце, продолжается пока перед repeat
-- на стеке ЛОЖЬ (0)


: test4 begin

    ." test4 "

  0 until ;
```

5.3.4 Условные ветвления

```
-- Операторы IF THEN

: test3
  2 == if
    " это двойка... " type cr
     then ;


-- Операторы IF ELSE THEN

: test4
  2 == if
    " это двойка... " type cr
     else
    " это не двойка" type cr
     then ;
```

5.3.5 Переменные и константы

```
-- Константы

0x41 constant ADDR1

: test5
  ADDR1 h. cr
  ;


-- Переменные

variable var1

: test6

  0x15 var1 ! -- Записываем в переменную

  var1 @ h. cr  -- Читаем из переменной

  ;
```

5.3.6 Загрузка файла в SDK-1.1

При помощи этого m3p-скрипта (load.m3p) производится загрузка исполняемого файла в учебный стенд SDK-1.1:

```
terminateonerror
-- Открытие com-порта на скорости 9600 бит/с:
-- com1 под MS Windows, /dev/ttyS0 под Linux
9600 openchannel com1
: wait
    cr cr
    ." Включите питание и нажмите кнопку RESET на стенде SDK." cr cr
    ." Ожидание перезапуска... "
    begin rsio dup emit 109 == until
    ." Ok" cr cr
;
    wait

    T_RAM

    0x2100       write   test_serial.bin
    0x2100       jmp

    0 term

bye
```

5.4 Утилита make

Утилита make, как известно, используется для сборки сравнительно больших проектов. Раньше, когда компьютеры не обладали такими вычислительными мощностями как сейчас, время компиляции исходных текстов было настолько велико, что возникла необходимость в раздельной и выборочной компиляции. В чём суть этой раздельной компиляции? Процесс разбивался на два этапа: превращение исходных текстов в объектные модули и сборка этих объектных модулей в загрузочный модуль. Далее, если производить компиляцию выборочно, то есть если не компилировать все исходные тексты подряд, а компилировать только те, которые претерпели изменения, можно получить существенную экономию времени сборки всего проекта.

Для того, чтобы прочувствовать необходимость такой выборочной компиляции сейчас, пожалуй, необходимо скомпилировать что-то действительно большое. К примеру, ядро операционной системы Linux, или собрать из исходных текстов Open Office. Правда, помня о такой маркетинговой уловке производителей железа как закон Мура, я полагаю, с неуклонным ростом вычислительных ресурсов этот пример станет не очень удачным уже достаточно скоро.

Зачем же нам тогда make, спросите вы? Ведь большинство проектов для встроенных систем весьма аскетичны, а скомпилировать три десятка модулей на C или C++ современный компьютер может за считанные секунды?

Ответ очень прост: make достаточно мощный, *специализированный* инструмент для работы с проектами.

Со специализацией инструментов я столкнулся в полной мере, когда попробовал использовать язык Perl для обработки текстов вместо обычного на тот момент для меня C++. Результат меня тогда поразил. За три дня на язке Perl был выполнен объем работы, которые потребовал от меня две недели на C++! Это при том, что C++ я уже использовал давно, а Perl видел в первый раз в жизни. Perl специально создавали как средство для обработки текстов. C++ не помогла библиотека STL, хотя вроде как набор механизмов у инструментов был примерно одинаковый. Разве что в C++ у меня не использовались библиотеки для разбора регулярных выражений.

В чём же оказалась сила Perl? В первую очередь – в простоте. Краткая синтакическая конструкция, вызывающая у новичка головную боль и мысли о распечатке дампа памяти, делает то, что на C++ потребует написания полутора десятков строк текста. Нам известно, что уровень языка не влияет на количество ошибок. Влияние оказывает количество строк, то есть чем больше строк, тем больше ошибок и наоборот. В программе на Perl строк было мало и я продвигался к завершению проекта быстрыми темпами.

Итак, вернемся к make. Основная цель утилиты – сократить время, необходимое для описания того, что надо сделать с проектом. Предполагается при этом, что вы работаете с проектом самостоятельно, без каких либо IDE.

Что нужно делать с проектом во время сборки?

- Убрать старые и ненужные файлы, объектные модули, всякий мусор, исполняемые файлы и т.д. Это полезно делать перед архивацией проекта или перед отдачей проекта системе контроля версий.
- Запаковать проект в архив, при этом желательно переписать его в пару-тройку мест, а чтобы не запутаться – назвать архив так же, как называется проект.
- Скомпилировать исходные тексты.
- Собрать загрузочный модуль.
- Вызвать необходимые конвертеры для получения формата, удобоваримого для доставки в контроллер.
- Вызвать программу для доставки загрузочного модуля в контроллер.

Ко всему этому хочется добавить, что проектов у вас много, а вы один. Большинство описанных выше действий вполне себе стандартно и хочется вынести их за скобки. Утилита make позволяет описать параметры сборки проекта в виде переменных, а из этих переменных задать последовательность действий. Получается, что переменные от проекта к проекту практически не меняются, а вот их содержимое может изменяться.

Что мы можем указать в виде таких переменных? Попробуем привести список.

- Имя проекта. Это имя можно использовать как имя архива, а также как имя исполняемого или загрузочного модуля.
- Дополнительные части для имени проекта, такие как дата и время компиляции, порядковый номер сборки, тип проекта.
- Имя компилятора. В области встроенных систем считается вполне нормальным частый переход на разные микроконтроллеры и, соответственно, использование разных компиляторов раз в несколько месяцев.
- Опции компилятора. Флаги оптимизации, специфические опции, местоположение заголовочных файлов и т.п.
- Опции линкера. Наименование библиотек, адреса памяти программ и кода и т.д.
- Архиватор и ключи для архивации.
- Имя программы и ее ключи для доставки ПО в контроллер.
- Список этот далеко не полный, его можно продолжать.

5.4.1 Использование make

По своей структуре makefile (обычно так называют программу для make) можно разделить на две части: описание переменных и описание действий над этими переменными. Программа для make очень напоминает форму Бэкуса-Наура (БНФ, по-английски Backus-Naur form, BNF), позволяющую описать синтаксис языка программирования. В БНФ одна конструкция последовательно определяется через другие. Такую запись, как в make, можно увидеть в знаменитом компиляторе компиляторов YACC. Для того, чтобы вам было проще понять make, я вам рекомендую почитать литературу про БНФ.

С переменными все просто, формат записи следующий:

```
PROJECT = test_sdk11
```

Для использования переменной достаточно указать имя переменной в скобках со знаком доллара:

```
$(PROJECT)
```

Вместо PROJECT будет подставлено test_sdk11. Часть makefile, отвечающая за действия, состоит из ряда правил, каждое из которых имеет следующий вид:

```
Цели: зависимость1 зависимость2 ...
 команда 1
 команда 2
 ...
```

Цель (target) – это то, ради чего существует данное правило. Если это файл, то утилита make учитывает факт его наличия и время его создания. Зависимости показывают, от чего данное правило зависит. Любое изменение зависимости приведет к запуску команды. Зависимости могут быть описаны ниже как цели. Команды – перечень команд, которые исполняются при

исполнении правила. В общем случае их может и не быть, так же как может и не быть зависимостей.

Пример правила без зависимостей:

```
clean:
    -rm -f $(NAME).hex \
        $(NAME).bin \
        $(NAME).map \
        $(NAME).lst
```

В данном правиле производится стирание файлов, имя которых состоит из содержимого переменной NAME и расширений .hex, .bin, .map и .lst. Необходимо заметить, что оно будет работать совершенно неожиданным образом, если в каталоге будет находиться файл с именем clean.

Значение переменных в makefile задается с помощью знака присваивания. В переменную попадает все содержимое строки.

```
LFLAGS = --code-loc 0x2100 --xram-loc 0x6000 --stack-loc 0x80
```

Возможно продление строки с помощью обратного слэша. Кроме того, возможно использование ранее определенных переменных при определении новых.

```
LIST_SRC = \
$(SRC_DIR)/led.c \
$(SRC_DIR)/max.c \
$(SRC_DIR)/test_led.c
```

В круглых скобках возможно производить различные действия, с помощью достаточно большого количества функций. Например, возможна переделка списка исходных текстов в список объектных модулей:

```
LIST_OBJ = $(LIST_SRC:.c=.rel)
```

С помощью функции shell возможен вызов командного интерпретатора. Результатом его работы является строка, полученная через стандартный вывод и присваиваемая переменной.

```
DATE    = $(shell date +%d-%m-%g_%H-%M-%S)
VERSION = $(shell cat VERSION)
```

В документации к утилите make подробно описаны функции, которые можно использовать при определении переменных [34].

5.4.2 Пример makefile для Windows

На примере программирования учебного стенда SDK-1.1 будет продемонстрирована работа с утилитой make в ОС Microsoft Windows.

Для компиляции проекта необходимо войти в каталог содержащий файл makefile и запустить команду **make.**

Для загрузки программы в SDK-1.1 необходимо запустить команду **make load.** Для очистки каталога от файлов, полученных в результате компиляции, необходимо запустить команду **make clean**. Чтобы запустить эмулятор терминала необходимо запустить команду **make term**. Необходимо иметь в виду, что по умолчанию в примерах используется последовательный канал com1. Если ваш SDK-1.1 подключен к другому порту, необходимо в **makefile** и в m3p-скрипте load.m3p (см. предыдущие разделы) заменить com1 на имя вашего порта.

```
# ------------------------------------------------------------------
# Имя проекта

NAME   = test_led

# Настройки компилятора и линкера

CC     = sdcc
CFLAGS = -I./INCLUDE -c --stack-auto
LFLAGS = --code-loc 0x2100 --xram-loc 0x6000 --stack-auto --stack-loc 0x80

PROJNAME = ${PROJECT}-${VERSION}-${BUILD}-${TYPE}
TARBALL  = ${PROJNAME}.tar

# Настройки M3P

M3P           = m3p
COMPORT = com1
COMLOG  = $(COMPORT)_log.txt
BAUD      = 9600

# Каталоги с исходными текстами

SRC_DIR = SRC
# ------------------------------------------------------------------
all: test_led

clean:
 del $(NAME).hex
 del $(NAME).bin
 del $(NAME).map
 del $(NAME).mem
 del $(NAME).lnk
 del pm3p_*.txt
 del com?_log.txt
 del $(TARBALL).gz
 del $(SRC_DIR)\*.asm
 del $(SRC_DIR)\*.rel
 del $(SRC_DIR)\*.rst
 del $(SRC_DIR)\*.sym
 del $(SRC_DIR)\*.lst

load:
 $(M3P) lfile load.m3p


term:
 $(M3P) echo $(COMLOG) $(BAUD) openchannel $(COMPORT) \
   +echo 6 term -echo bye

LIST_SRC = \
$(SRC_DIR)/led.c \
```

```
$(SRC_DIR)/max.c \
$(SRC_DIR)/test_led.c

LIST_OBJ = $(LIST_SRC:.c=.rel)

test_led : $(LIST_OBJ) makefile
 $(CC) $(LIST_OBJ) -o test_led.hex $(LFLAGS)
 $(M3P) hb166 test_led.hex test_led.bin bye


$(LIST_OBJ) : %.rel : %.c makefile
 $(CC) -c $(CFLAGS) $< -o $@
```

5.4.3 Пример makefile для Linux

Так выглядит **makefile** в случае программирования учебного стенда SDK-1.1 в ОС Linux:

```
# ---------------------------------------------------------------------
# Имя проекта

NAME   = test_led

# Настройки компилятора и линкера

CC   = sdcc
CFLAGS = -I./INCLUDE -c --stack-auto
LFLAGS = --code-loc 0x2100 --xram-loc 0x6000 --stack-auto --stack-loc 0x80

PROJNAME = ${PROJECT}-${VERSION}-${BUILD}-${TYPE}
TARBALL = ${PROJNAME}.tar

# Настройки M3P

M3P           = m3p
COMPORT = /dev/ttyS0
COMLOG  = $(COMPORT)_log.txt
BAUD    = 9600

# Каталоги с исходными текстами

SRC_DIR = SRC
# ---------------------------------------------------------------------
all: test_led

clean:
 -rm -f $(NAME).hex \
             $(NAME).bin \
             $(NAME).map \
             $(NAME).mem \
             $(NAME).lnk \
             pm3p_*.txt \
             com?_log.txt \
             $(TARBALL).gz \
             $(SRC_DIR)/*.asm \
             $(SRC_DIR)/*.rel \
             $(SRC_DIR)/*.rst \
             $(SRC_DIR)/*.sym \
             $(SRC_DIR)/*.lst
```

```
load:
 $(M3P) lfile load.m3p

term:
 $(M3P) echo $(COMLOG) $(BAUD) openchannel $(COMPORT) \
   +echo 6 term -echo bye

LIST_SRC = \
$(SRC_DIR)/led.c \
$(SRC_DIR)/max.c \
$(SRC_DIR)/test_led.c

LIST_OBJ = $(LIST_SRC:.c=.rel)

test_led : $(LIST_OBJ) makefile
 $(CC) $(LIST_OBJ) -o test_led.hex $(LFLAGS)
 $(M3P) hb166 test_led.hex test_led.bin bye

$(LIST_OBJ) : %.rel : %.c makefile
 $(CC) -c $(CFLAGS) $< -o $@
```

5.5 Система контроля версий

Система контроля версий решает сразу несколько задач для разработчика встроенного ПО:

- Позволяет сохранять в надёжном месте весь его проект (выполняет функции интеллектуального архиватора).
- Позволяет предоставить доступ к проекту для всех его участников, а также регламентировать доступ для посторонних людей.
- Позволяет отслеживать все изменения в проекте, возвращаться к старым версиям файлов и вести параллельно сразу несколько вариантов одного проекта.

Большинство опытных монстров и зубров в области программирования могут возразить, что для сравнительно несложных программных проектов, существующих в области встроенных систем, система контроля версий – лишняя бюрократическая процедура, отнимающая время и не дающая ничего взамен. Хочу вас заверить, что это не так. Даже если вы работаете один, и вам не нужно совместно с кем-то работать над одним проектом, система контроля версий оказывается гораздо удобнее банального архиватора. С помощью такой, к примеру, команды:

```
svn co http://vash_server.vash_domen.ru/repos/proect
```

вы можете получить доступ к своим исходным текстам везде, где есть компьютеры и выход в Интернет. С помощью простой команды `svn commit` вы можете отправить свои изменения обратно на сервер. Вам не нужно больше бояться, что вирусы уничтожат ваши файлы, или что у вас сломается жесткий диск. Все данные хранятся на удаленном сервере и всё, что вы можете потерять, – это несколько часов работы. Естественно, все это будет работать

именно так, если вы позаботитесь о регулярном резервном копировании вашего сервера.

В настоящее время системы контроля версий интегрированы в различные интегрированные среды разработки (IDE). Наиболее распространенными системами контроля версий, широко применяемыми на момент написания этой книги, являются CVS и Subversion. Эти системы очень похожи. CVS появилась раньше, чем Subversion и хорошо себя зарекомендовала. Subversion в настоящий момент активно пробивается в лидеры, в ней исправлено несколько концептуальных ошибок, имеющихся в ее предшественнике SVN. Чем пользоваться вам – решайте сами. По сути, все системы контроля версий очень похожи, и чем пользоваться – дело привычки или каких-то иных предпочтений.

5.5.1 Работа с системой контроля версий

Работа с системой контроля версий начинается с создания репозитория. Репозиторий – это специализированная база данных, в которой хранятся файлы вашего проекта, вносимые изменения, информация о создании и удалении файлов, информация о пользователях и т.п. В основном, репозиторий рассчитан на хранение обычной текстовой информации, хотя можно хранить и двоичные файлы (к примеру, исполняемые, документацию в формате PDF или DOC). Почему не использовать обычную СУБД, к примеру, такую как MySQL или Oracle? Ответ прост: проблема в эффективности. Репозиторий системы контроля версий специально ориентирован на хранение текстовых документов и изменений в них. Хранение таких данных (весьма большого объема) в обычной базе данных вызовет неэффективное использование дискового пространства и замедление работы. Обычно в репозитории хранится разница между текущей и предыдущей версией файла. Для того, чтобы посмотреть на то, как может выглядеть внутренности репозитория, изучите работу программы `diff`.

После создания репозитория можно добавлять и стирать в нем файлы, получать содержимое репозитория на свою машину, проверять обновления и отправлять изменения обратно. Типичный цикл работы на примере Subversion может выглядеть примерно так:

```
svn co http://194.85.162.173/repos/ul3
```

Эта строка позволяет забрать с сервера 194.85.162.173 из репозитория repos/ul3 ваш проект. Вы можете отредактировать нужные вам файлы. Для того, чтобы убедиться, что ваши товарищи не добавили в проект чего-то нового можно вызвать команду:

```
svn update
```

Исполнение этой команды приводит к тому, что вы забираете из репозитория все новые файлы, появившиеся там после того, как вы скачали версию себе на машину. Если вы умудрились редактировать с кем-то один и тот же файл, система контроля версий должна предупредить вас о конфликте.

Разрешение конфликта делается в соответствии с документацией на используемый вами инструмент.

Для записи изменений, сделанных вами в репозиторий воспользуйтесь следующей командой:

```
svn commit
```

Данная команда отправит в репозиторий только те файлы, которые вы изменили. Если файлы были изменены кем-то еще, вам будет сообщено о конфликте. В процессе запуска команды будет вызван текстовый редактор, и вам будет предложено описать сделанное изменение. Пожалуйста, не ленитесь писать по существу. Эти записи потом вам сильно помогут разобраться в проекте, когда пройдет существенное время, и вы все основательно позабудете.

О более сложных вариантах использования систем контроля версий я рекомендую читать самостоятельно в специальной литературе [41].

6 Примеры программирования стенда SDK-1.1

6.1 Приступаем к работе

Для работы со стендом SDK-1.1 вам понадобится:

- Персональный компьютер или ноутбук, работающий под операционными системами MS Windows, Linux или Mac OS (в принципе, инструментальные средства должны работать в любой Unix подобной среде).
- Коммуникационный кабель RS-232 (возможно вам понадобится переходник USB2Com).
- SDCC, бесплатный компилятор языка Си для микроконтроллеров (этот компилятор можно свободно скачать в сети Интернет для разных платформ, есть исходные тексты).
- Утилита make (для Windows можно воспользоваться пакетом Cygwin, в Linux и MacOS можно воспользоваться версиями GNU make).
- Инструментальная система GM3P (исходные тексты и скомпилированная версия может быть свободно получена на сайте Научно-образовательного направления «Встроенные вычислительные системы» кафедры вычислительной техники СПбГУ ИТМО http://embedded.ifmo.ru).
- Исходные тексты примеров (могут быть скачаны на сайте http://embedded.ifmo.ru).
- Если вы любите работать в IDE, скачайте себе бесплатную (и весьма при этом удобную) среду разработки Eclipse с плагином для работы с исходными текстами на языках C/C++.

Все перечисленные компоненты, как вы уже поняли, кроссплатформенные и будут работать практически везде.

На Интернет-форуме http://embedded.ifmo.ru/forum приветствуется обсуждение особенностей программирования учебного стенда SDK-1.1 (а также и других стендов) в специальных разделах.

Если вы знакомы со сборкой программ из исходных текстов в ОС Linux, для вас не составит особого труда разобраться с тем, как можно скомпилировать программу и загрузить ее в учебный стенд. В противном случае, рекомендуется изучить разделы данного пособия, посвященные инструментальному обеспечению (sdcc, make, m3p). В качестве дополнительного материала можно порекомендовать руководство пользователя для SDCC (на английском языке), множество статей и книг на русском и английском языках по утилите make, а также руководство пользователя и исходные тексты утилиты M3P, которые можно скачать с сайта http://embedded.ifmo.ru. Не знающим язык Си хочется порекомендовать книгу Кернигана и Ричи «Язык программирования Си» [39].

Итак, мы берем стенд SDK-1.1 в руки и начинаем работать. ***Необходимо помнить, что все подключения и отключения коммуникационного кабеля RS-232 необходимо производить при выключенном питании стенда SDK-1.1!***

Порядок работы со стендом очень прост:

1. Установите себе компилятор SDCC, утилиту make и M3P.
2. Подключите учебный стенд к своему компьютеру с помощью кабеля RS-232 (кабель входит в комплект поставки стенда). В случае подключения через USB вам понадобится переходник USB2Com.
3. Подключите к стенду прилагаемый источник питания.
4. Вставьте источник питания в розетку (сеть 220В, 50 Гц). На стенде должен загореться светодиод POWER.
5. Скачайте тестовый пример и войдите в каталог с тестовым примером.
6. Откорректируйте название вашего последовательного порта в makefile. Для Windows это будет com1, com2,…, для Linux /dev/ttyS0, /dev/ttyS1 и так далее. Если у вас один встроенный порт, то это скорее всего com1 или /dev/ttyS0.
7. Скомпилируйте тестовый проект. Для этого нужно в каталоге, в котором находится файл makefile, запустить команду make.
8. Загрузите получившийся файл в учебный стенд. Для этого нужно выполнить команду make load.
9. Если всё прошло нормально, вы увидите эмулятор терминала, запущенный в консоли вашего компьютера, а программа, загруженная в SDK-1.1, начнет исполняться.

6.2 Программирование светодиодных индикаторов

Светодиодные индикаторы (8 шт.) в стенде SDK-1.1 подключены не напрямую к микроконтроллеру ADuC812, а через расширитель портов, выполненный на базе ПЛИС. За связь со светодиодами в расширителе портов отвечает 8-разрядный регистр SV, который находится по адресу 080007h (на 8-й странице внешней памяти данных 7-я ячейка), значение после сброса 00000000B, доступен только для записи (см. подраздел 4.4.3).

Для доступа к регистрам ПЛИС нужно переключить страничный регистр DPP на 8 страницу памяти. Адреса регистров внутри страницы находятся в диапазоне от 0 до 7.

Доступ к регистрам возможен через указатель:

```
unsigned char xdata *regnum
```

Необходимо помнить, что при переключении страниц становятся недоступными все данные, размещенные в странице 0 (т.е. переменные в программе объявленные с модификатором памяти xdata).

Для того, чтобы избежать проблем со страничным регистром DPP, нужно использовать специальные функции для доступа к ПЛИС, которые перед

началом работы с регистрами ПЛИС будут запоминать старое значение страничного регистра, а по окончании работы возвращать его обратно.

Нужно следить, чтобы передаваемые в регистры ПЛИС значения хранились во внутренней памяти микроконтроллера (DATA, IDATA). Убедиться, что передаваемая информация не содержится во внешней памяти контроллера (XDATA), достаточно легко: так как для доступа к внешней памяти в микроконтроллерах семейства Intel MCS-51 используется регистр специального назначения DPTR, нужно просто просмотреть листинг программы и убедиться в том, что для доступа к переменным компилятор не использует DPTR.

Пример программы:

```
#include "aduc812.h"

#define MAXBASE 0x8  // Номер страницы внешней памяти (xdata),
                     // куда отображаются регистры расширителя портов в/в.

/*---------------------------------------------------------------------
                                  write_max
---------------------------------------------------------------------
Запись в нужный регистр ПЛИС ALTERA MAX3064(3128)

Вход:    regnum - адрес (номер) регистра
         val   - записываемое значение
Выход:     нет
Результат: нет
Описание:  Производится запись в регистр (порт) ПЛИС ALTERA MAX3064(3128)
           путем переключения адресуемой страницы памяти на страницу, где
           расположены (куда отображаются) порты ввода-вывода ПЛИС.
---------------------------------------------------------------------*/
void write_max( unsigned char xdata *regnum, unsigned char val )
{
unsigned char oldDPP = DPP;

  DPP   = MAXBASE;
  *regnum = val;
  DPP   = oldDPP;
}

/*---------------------------------------------------------------------
                                  read_max
---------------------------------------------------------------------
Чтение из нужного регистра ПЛИС ALTERA MAX3064(3128)

Вход:    regnum - адрес (номер) регистра
Выход:   нет
Результат: прочитанное из регистра значение
Описание:  Чтение из порта ПЛИС ALTERA MAX3064(3128)
      путем переключения адресуемой страницы памяти на страницу, где
      расположены (куда отображаются) порты ввода-вывода ПЛИС
Пример:
---------------------------------------------------------------------*/
unsigned char read_max( unsigned char xdata *regnum )
{
unsigned char oldDPP=DPP;
unsigned char val;

  DPP = MAXBASE;
```

```
  val = *regnum;
  DPP = oldDPP;

  return val;
}

/*---------------------------------------------------------------------
                                    leds
---------------------------------------------------------------------
Зажигание линейки светодиодов

Вход: on - управление светодиодами. Каждый бит переменной отвечает за один
      светодиод: 1 - зажигает, 0 гасит светодиод
Выход:     нет
Результат: нет
Описание:  Производится доступ к регистру расширителя портов SV с помощью
           функции write_max. Состояние светодиодов хранится в регистре
           old_led.
---------------------------------------------------------------------*/

void leds( unsigned char on )
{
  write_max( SV, on );

  old_led = on;
}

/*---------------------------------------------------------------------
                                    main
---------------------------------------------------------------------*/
// Задержка на заданное количество мс
void delay ( unsigned long ms )
{
volatile unsigned long i, j;
    for( j = 0; j < ms; j++ )
    {
        for( i = 0; i < 50; i++ );
    }
}

void main( void )
{
int i;
    for( i = 0; i < 3; i++ )
    {
        leds( 0xFF );  delay( 100 );
        leds( 0x00 );  delay( 100 );
    }
    led( 0, 1 ); delay( 300 );
    led( 1, 1 ); delay( 300 );
    led( 2, 1 ); delay( 300 );
    led( 3, 1 ); delay( 300 );
    led( 4, 1 ); delay( 300 );
    led( 5, 1 ); delay( 300 );
    led( 6, 1 ); delay( 300 );
    led( 7, 1 ); delay( 300 );
    delay( 1000 );

    for( i = 0; i < 3; i++ )
    {
        leds( 0xFF );  delay( 100 );
        leds( 0x00 );  delay( 100 );
    }
```

```
    while( 1 )
    {
        leds( 0x55 );  delay( 500 );
        leds( 0xAA );  delay( 500 );
    }
}
```

6.3 Программирование последовательного канала

Простейшим из способов организации последовательного обмена является асинхронный обмен с программной проверкой готовности. Примером такого обмена может служить работа с контроллером последовательного канала (UART) МК ADuC812 в стенде SDK-1.1 «по опросу» (см. раздел 2.2.9.1). Если требуется переслать байт, то: 1) сбрасывается TI; 2) в SBUF записывается нужный байт данных; и 3) ожидается, пока TI не будет установлен контроллером. С приемом здесь сложнее: нужно постоянно проверять флаг RI и, если он установлен, то читать принятый байт из SBUF и сбрасывать RI. Такой способ удобен, когда разработчику четко известно, когда произойдет прием данных и когда его завершать. Неудобен он тем, что время выполнения самой программы напрямую зависит от скорости обмена. В самом деле, чем медленнее скорость, тем дольше по времени цикл ожидания готовности к приему/посылке следующего байта.

Пример программы:

```
/*--------------------------------------------------------------------
                                  init_sio
--------------------------------------------------------------------
Инициализирует последовательный канал на заданной скорости.

Вход:   char speed - скорость. Задается константами, описанными в
        заголовочном файле sio.h
        bit sdouble - дублирование скорости: 0 - не дублировать скорость,
        заданную аргументом speed; 1 - дублировать.
Выход:     нет
Результат: нет
-------------------------------------------------------------------- */
void init_sio( unsigned char speed )
{
  TH1    = speed;
  TMOD   |= 0x20; //Таймер 1 будет работать в режиме autoreload
  TCON   |= 0x40; //Запуск таймера 1
  SCON   = 0x50;  //Настройки последовательного канала
  ES     = 0;     //Запрещение прерываний от приемопередатчика
}

/*--------------------------------------------------------------------
                                  RSioStat
--------------------------------------------------------------------
Возвращает ненулевое значение, если буфер приема не пуст

Вход:      нет
Выход:     нет
Результат: 0 - буфер приема пуст,
           1 - был принят символ
-------------------------------------------------------------------- */
unsigned char rsiostat(void)
```

```
{
  return RI;
}

/*---------------------------------------------------------------------
                                    wsio
---------------------------------------------------------------------
Отправляет символ по последовательному каналу

Вход:      unsigned char c - символ, который нужно отправить
Выход:     нет
Результат: нет
--------------------------------------------------------------------- */
void wsio( unsigned char c )
{
  SBUF = c;
  TI  = 0;
  while( !TI );
}

/*---------------------------------------------------------------------
                                    rsio
---------------------------------------------------------------------
Дожидается приема символа из последовательного канала и возвращает его.

Вход:      нет
Выход:     нет
Результат: принятый символ
--------------------------------------------------------------------- */
unsigned char rsio(void)
{
  while( !RI );
  RI = 0;
  return SBUF;
}

/*---------------------------------------------------------------------
                                    type
---------------------------------------------------------------------
Выводит ASCIIZ-строку в последовательный канал

Вход:      char *str - указатель на строку
Выход:     нет
Результат: нет
--------------------------------------------------------------------- */
void type(char * str)
{
  while( *str ) wsio( *str++ );
}

/*---------------------------------------------------------------------
                                    main
---------------------------------------------------------------------*/
void main( void )
{
unsigned char c;

    init_sio( S9600 );

    type("Тест драйвера SIO для стенда SDK-1.1\r\n");
    type("Нажимайте кноки для тестирования... \r\n");

    while( 1 )
```

```
    {
        if( rsiostat() )
        {
            c = rsio();

            switch( c )
            {
            case '1': type("\r\ntest 1\r\n"); break;
            case '2': type("\r\ntest 2\r\n"); break;
            case '3': type("\r\ntest 3\r\n"); break;

            default: wsio( c );          break;
            }
        }
    }
}
```

При организации асинхронного обмена по прерыванию при приеме байта с линии происходит прерывание и передача управления соответствующей программе-обработчику, который читает принятый байт из порта данных контроллера UART и, к примеру, помещает его в специальный буфер-очередь принятых байт, доступный прерванной программе (см. раздел А.2.4). По завершении процедуры обработки прерывания управление передается в прерванную программу, которая при желании (в любом удобном месте алгоритма) может забрать принятый байт. С другой стороны, при завершении отправки контроллером очередного байта также происходит прерывание, сигнализирующее о том, что байт послан и в буфер UART можно поместить новые данные. Обработчик прерывания, при наличии данных в исходящей очереди, записывает очередной байт в порт данных контроллера и запускает посылку. Основная же программа может, не заботясь о готовности или неготовности контроллера принять очередной байт, может спокойно помещать данные в исходящий буфер, а всю работу с устройством выполнит обработчик прерываний, когда оно (устройство) будет готово.

6.4 Программирование таймера

Микроконтроллер ADuC812 имеет три программируемых 16-битных таймера/счетчика: Таймер 0, Таймер 1, Таймер 2. Каждый таймер состоит из двух 8-битных регистров THX и TLX. Все три таймера могут быть настроены на работу в режимах “таймер” или "счетчик". Организация и принцип работы Таймера 0 и 1 рассмотрена в подразделе 2.2.6.1.

Чтобы настроить таймер на определенную частоту (работа по прерыванию) в МК ADuC812 стенда SDK-1.1, необходимо выполнить следующую последовательность действий:

- Выбрать один из 4-х режимов работы таймера (регистр TMOD);
- Настроить таймер на заданную частоту (инициализировать регистры THx, TLx);
- Включить таймер (регистр TCON);

- Написать обработчик прерывания от таймера и установить вектор прерывания в пользовательской таблице прерываний (внешняя память данных начиная с адреса 2003h);
- Разрешить прерывания от выранного таймера (регистр IE);
- Разрешить все прерывания (регистр IE).

В приведенном примере программы реализована анимация на светодиодных индикаторах. Исходные коды драйвера ПЛИС смотрите в разделе 6.2.

```
unsigned long __systime = 0;

//////////////////////////////////////////////////////////////////////////
// Инициализация Таймера 0 (1000Гц)
//////////////////////////////////////////////////////////////////////////
void InitSystimer0( void )
{
 TCON = 0x00;      // Выключение таймера 0 (и таймера 1)
 TMOD = 0x01;      // Выбор режима работы 16-разрядный таймер
 TH0 = 0xFC;       // Инициализация таймера 0:
 TL0 = 0x67;       // настройка на частоту работы 1000 Гц (чуть больше)
 TCON = 0x10;      // Включение таймера 0
}


//////////////////////////////////////////////////////////////////////////
// Чтение милисекундного счетчика
//////////////////////////////////////////////////////////////////////////
unsigned long GetMsCounter( void )
{
 unsigned long res;

 ET0 = 0;
 res = __systime;
 ET0 = 1;

 return res;
}

//////////////////////////////////////////////////////////////////////////
// Возвращает прошедшее время (от момента замера)
//////////////////////////////////////////////////////////////////////////
unsigned long DTimeMs( unsigned long t2 )
{
    unsigned long t1 = ( unsigned long )GetMsCounter();

    return t1 - t2;
}

//////////////////////////////////////////////////////////////////////////
// Задержка в милисекундах
//////////////////////////////////////////////////////////////////////////
void DelayMs( unsigned long ms )
{
 unsigned long t1 = ( unsigned long )GetMsCounter();

 while ( 1 )
 {
       if ( DTimeMs( t1 ) > ms ) break;
 }
}
```

```
////////////////////////// T0_ISR ////////////////////////////////
// Обработчик прерывания от таймера 0.
//////////////////////////////////////////////////////////////////
void T0_ISR( void ) __interrupt ( 1 )
{
 // Время в милисекундах
 __systime++;

 TH0 = 0xFC;       // Инициализация таймера 0:
 TL0 = 0x67;       // настройка на частоту работы 1000 Гц (чуть больше)
}

////////////////////////// SetVector /////////////////////////////
// Функция, устанавливающая вектор прерывания в пользовательской таблице
// прерываний.
// Вход:      Vector - адрес обработчика прерывания,
//            Address - вектор пользовательской таблицы прерываний.
// Выход:     нет.
// Результат: нет.
//////////////////////////////////////////////////////////////////
void SetVector(unsigned char xdata * Address, void * Vector)
{
 unsigned char xdata * TmpVector; // Временная переменная
 // Первым байтом по указанному адресу записывается
 // код команды передачи управления ljmp, равный 02h
 *Address = 0x02;
 // Далее записывается адрес перехода Vector
 TmpVector = (unsigned char xdata *) (Address + 1);
 *TmpVector = (unsigned char) ((unsigned short)Vector >> 8);
 ++TmpVector;
 *TmpVector = (unsigned char) Vector;
 // Таким образом, по адресу Address теперь
 // располагается инструкция ljmp Vector
}

void main( void )
{
 unsigned char light = 1;

 InitSystimer0();

 // Установка вектора в пользовательской таблице
 SetVector( 0x200B, (void *)T0_ISR );
 // Разрешение прерываний от таймера 0
 ET0 = 1; EA = 1;

    while( 1 )
 {
       leds( light );
       if( light == 0xFF ) light = 1;
       else light |= light << 1;
       DelayMs( 300 );
 }
}
```

6.5 Программирование ЖКИ

6.5.1 Работа с ЖКИ

Работать с ЖКИ достаточно просто, так как оперировать вам придётся всего с 4 регистрами. Необходимо помнить, что контроллер ЖКИ в SDK-1.1 подключен не напрямую к микроконтроллеру, а через расширитель портов, выполненный на базе ПЛИС. За связь с ЖКИ, в расширителе портов отвечают первые два регистра:

1. *DATA_IND* отвечает за выдачу информации на шину данных (через этот регистр можно передавать команды контроллеру и данные;
2. *C_IND* отвечает за формирование сигналов E, R/W и RS, позволяющих регулировать обмен на шине между расширителем портов и контроллером ЖКИ.

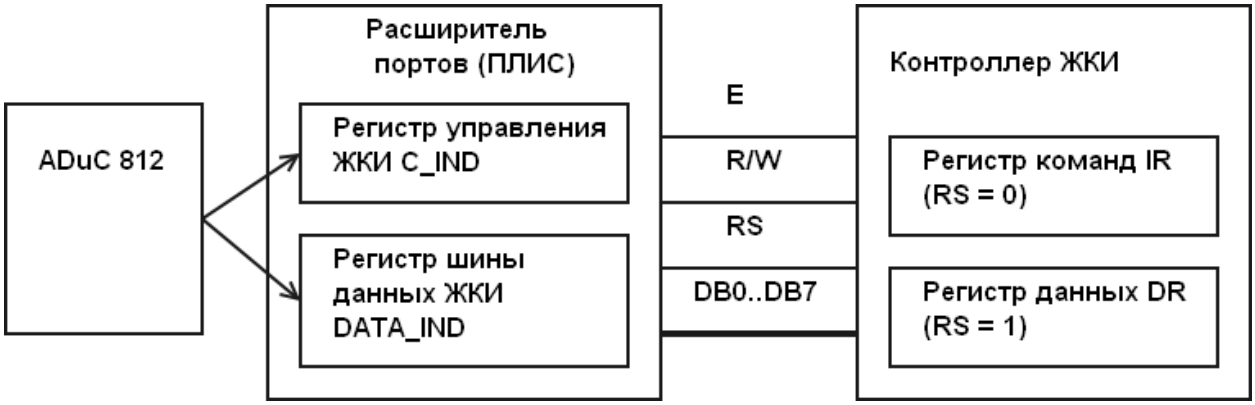


Рисунок 94. Регистры, необходимые для работы с ЖКИ: слева регистры расширителя портов, справа, регистры контроллера ЖКИ. Для доступа к регистрам контроллера ЖКИ вы должны сформировать на шине сигналы, с помощью регистров расширителя портов.

Вся работа с индикатором сводится к нескольким простым вещам:

1. Первым шагом вы записываете команду или данные (коды выводимых символов) в регистр DATA_IND расширителя портов. После этого, содержимое этого регистра появляется на шине данных контроллера ЖКИ (DB0..DB7). Контроллер на эти данные естественно не реагирует, так как сигнал ‘E’ (Enable) нами еще не выставлен в активный уровень (логическая ‘1’).
2. Вторым шагом, вы должны разрешить работу с шиной с помощью сигнала ‘E’ (логическая ‘1’), выставить сигнал записи (логический ‘0’ на линии ‘W’) и указать тип регистра, с которым вы будете работать в контроллере ЖКИ на линии RS. Если вы передаёте данные, то на сигнал RS нужно подать ‘1’, если команду, то ‘0’.

Вторая пара регистров находится в контроллере ЖКИ: это регистр команд (IR) и регистр данных (DR). Контроллер ЖКИ умеет выполнять несколько простых команд, например, таких как включение и выключение дисплея, очистка дисплея, позиционирование курсора и так далее. Подробнее, о

контроллере ЖКИ и соответствующих регистрах ПЛИС можно почитать в разделе 4.6 и его подразделах.

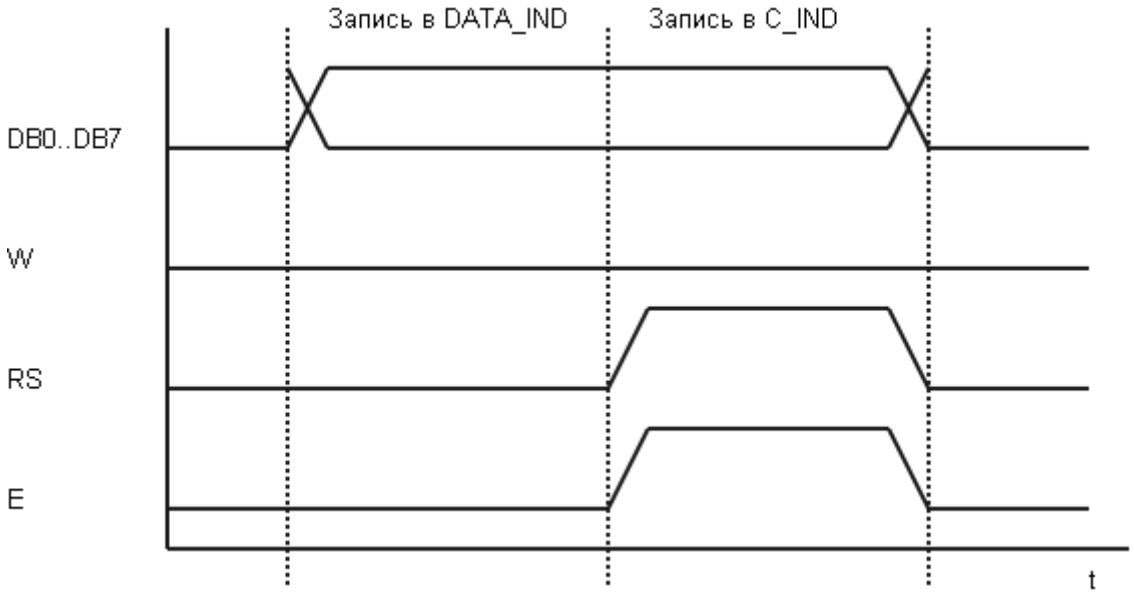


Рисунок 95. Временная диаграмма для записи в регистр данных (DR) контроллера ЖКИ. На первом шаге мы производим загрузку регистра DATA_IND (при этом у нас появляются данные на выходах DB0..DB7). На втором шаге мы формируем нужные нам управляющие сигналы W, RS и E, производя запись в регистр C_IND расширителя портов.

6.5.2 Пример программы

В приведенном ниже исходном тексте есть некоторые упрощения, сделанные для того, чтобы сделать более понятными основные операции, необходимые вам для работы с ЖКИ:

1. Необходимо помнить, что в учебном стенде SDK-1.1 начальная инициализация контроллера ЖКИ уже выполнена в загрузчике. В реальной системе вам придется программировать ее самостоятельно.
2. Время выполнения команд контроллером ЖКИ не равно нулю и это нужно учитывать в своей программе. В самом простом варианте можно сформировать задержки после выполнения команд (что и сделано в функции main()). Величину времени различных команд можно посмотреть в разделе «Жидкокристаллический индикатор».

В приведенном исходном тексте нет описания функции работы с регистрами ПЛИС (write_max). Драйвер ПЛИС представлен в разделе 6.2.

```
/* Пример работы с ЖКИ для учебного стенда SDK-1.1

    Логическая '1' на входе ‘E’ позволяет передавать и принимать данные между
микроконтроллером и контроллером ЖКИ. Логический ноль на входе RW позволяет
записывать данные в контроллер ЖКИ, сигнал RS позволяет переключать регистры
контроллера ЖКИ: 1 – данные, 0 – команды.
    */
    #define E    1      // Сигнал разрешения 'E'
    #define R    2      // Сигнал чтения
    #define RS   4      // Сигнал переключения в режим передачи данных
```

```
#define DATA_IND         0x1   // Регистр шины данных ЖКИ (ПЛИС)
#define C_IND            0x6   // Регистр команд ЖКИ (ПЛИС)

#define CLEAR            0x01  // Команда очистки дисплея

/*---------------------------------------------------------------------
                                lcd_clear
---------------------------------------------------------------------
Очистка дисплея.

Вход:      нет
Выход:     нет
Результат: нет
--------------------------------------------------------------------- */
void lcd_clear( void )
{
// Записываем в регистр шины данных ЖКИ ( DATA_IND )
// команду очистки дисплея CLEAR

 write_max( DATA_IND, CLEAR );

// Создаём строб записи, для этого на сигнальную линию 'E'
// выдаём короткий импульс

 write_max( C_IND, E );

// положительной полярности, на линии 'RS' держим
// '0', чтобы попасть в регистр команд

 write_max( C_IND, 0 );
}

/*---------------------------------------------------------------------
                                lcd_putch
---------------------------------------------------------------------
Вывод символа на ЖКИ.

Вход:      char c - выводимый символ
Выход:     нет
Результат: нет
--------------------------------------------------------------------- */
void lcd_putch( char c )
{

// Записываем в регистр шины данных ЖКИ ( DATA_IND ) код символа

 write_max( DATA_IND, c );

// Создаём строб записи, для этого на сигнальную линию 'E'
// выдаём короткий импульс

 write_max( C_IND, RS | E );

// положительной полярности, на линии 'RS' держим '1',
// чтобы попасть в регистр данных

 write_max( C_IND, RS   );
}

/*---------------------------------------------------------------------
                                  main
---------------------------------------------------------------------*/
```

```
void main( void )
{
 unsigned short i = 0;

 lcd_clear();

 for( i = 0; i < 300; i++ );

 lcd_putch( 'H' );
 lcd_putch( 'e' );
 lcd_putch( 'l' );
 lcd_putch( 'l' );
 lcd_putch( 'o' );
 lcd_putch( '!' );

 while( 1 );
}
```

Приложение А. Оформление документации и исходных текстов

А.1 Общие требования к отчётам

Текст отчёта оформляется в соответствии с ГОСТ 2.105-95.

Отчёт по лабораторной работе должен содержать:

1. *Титульный лист* с указанием названия университета, названия лабораторной работы, фамилии и инициалов авторов работы. В нижней части титульного листа укажите год.
2. *Номера страниц.* Номера страниц указываются в середине нижней части каждой страницы. В OpenOffice и MS Word нумерация страниц производится автоматически.
3. *Оглавление.* Оглавление вставляется автоматически, если вы используете стили Заголовок1, Заголовок2 и так далее (OpenOffice и MS Word).
4. *Введение или цель работы.* Этот раздел должен содержать информацию, позволяющую оценить стороннему человеку содержание данной работы.
5. *Техническое задание.* В техническом задании опишите то, что вам нужно сделать в вашей работе.
6. *Описание архитектуры.* Опишите фрагменты учебного стенда, которые вам предстоит запрограммировать, попытайтесь изобразить *концепцию* работы вашей программы. Описывайте только самое важное и интересное.
7. *Результаты работы.* Перечислите задачи, решенные вами, опишите возникшие сложности и проблемы.
8. *Список литературы* оформляется по следующим ГОСТам:
   a) ГОСТ 7.82-2001. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления
   b) ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления
9. *Приложение с исходными текстами.* В приложении должны быть приведены исходные тексты, оформленные по приведенным в данном пособии правилам кодирования на языке Си и снабженные комментариями.

Для основного текста используйте шрифт Times New Roman 12, полтора интервала, выравнивание по ширине. Для исходных текстов используйте

моноширинный шрифт `Courier New 10`, один интервал, выравнивание по левому краю.

A.2 Описание архитектуры

После того, как вычислительную технику захлестнул так называемый "кризис сложности", начались попытки систематизации и структурирования процессов проектирования. Кризис сложности проявлялся как сильный рост количества ошибок при увеличении размера проекта. Этот рост количества ошибок был нелинейным, и после определенного этапа количество ошибок становилось настолько большим, что ставило под вопрос завершение проекта.

Постепенно стало понятно, что блок-схема алгоритма, применявшаяся для описания сравнительно сложных программ, уже не дает никакого эффекта в крупных проектах. В работах Дейкстры и Вирта, а далее в работах Йордона, Росса и др., появилось предложение описывать программные системы в виде совокупности структурных и поведенческих составляющих. Появились технологии проектирования, объединенные общим подходом к проектированию системы. Система разрабатывалась на основе декомпозиции (разделения) общих сущностей на более частные. Была предложена так называемая абстракция, т.е. выделение существенных для проектировщика деталей проекта и сокрытие второстепенных.

Технологии проектирования, возникшие в этот период, были названы структурными. В области программного обеспечения этот подход к проектированию вылился в разделение программ на процедуры и структуры данных. Программирование разделилось на несколько этапов. Добавилось так называемое архитектурное проектирование. Вирт в своих работах писал о сильном семантическом (смысловом) разрыве между языком программирования и прикладной задачей. Первым шагом к архитектурному проектированию было предложение описывать программы в виде обычного текста, на обычном, человеческом языке. Позже такие описания стали называть структурным английским (русским и т.д.) или вербальными (речевыми).

Недостатком вербального описания программного обеспечения является низкая степень формализации, т.е. практическая невозможность математического доказательства правильности тех или иных утверждений, выраженных таким способом.

Росс, Йордон и другие ученые предложили ряд способов изображения архитектуры программного обеспечения. В основном, для описания программного обеспечения использовались поведенческие схемы, рассматривающие такие категории, как «данные», «процесс», «передача управления», «хранилище данных» (DFD, CFD, SADT). Для описания баз данных было придумано структурное описание на основе зависимостей различных сущностей друг от друга (например, диаграмма сущность-связь). Ранние методики проектирования не учитывали такого фактора, как время.

Позднее стали использоваться такие средства, как диаграммы взаимодействия и временные диаграммы. Для ответственных частей программного обеспечения изредка начали применять описание в виде конечных автоматов и, а позже – сетей Петри (в примитивном виде это конечные автоматы с временными задержками).

Структурный подход использовался и в проектировании аппаратного обеспечения. Аппаратная система описывалась структурной схемой, функциональной схемой, схемой электрической принципиальной и набором временных диаграмм. Необходимо заметить, что аппаратное обеспечение разрабатывалось, как правило, отдельно от программного.

В таком виде практически без изменений структурный подход к проектированию существует до сих пор. В конце 80-х, начале 90-х годов появилось объектно-ориентированное проектирование. В целом это развитие структурного подхода к проектированию. Отличие состоит в том, что вместо декомпозиции имеет место быть эволюция, т.е. развитие; кроме этого, система представляется в виде множества так называемых объектов, наделенных и структурными, и поведенческими составляющими. Специальные объектно-ориентированные языки программирования C++, SmallTalk и другие позволяют описывать программу, начиная от некоторого «зародыша». Зародыш постепенно растет, обретая в процессе проектирования новые свойства и методы (структура и поведение, как раньше).

В принципе, ничего особенно революционного в этом подходе нет. Декларируемое удобство состоит в том, что человеку (в данном случае проектировщику) проще оперировать объектами реального мира, чем абстрактными функциями. Собственно, кроме этой естественности восприятия, в объектно-ориентированном проектировании ничего нового нет. При использовании структурных методов архитектурного проектирования можно добиться тех же результатов, без использования объектно-ориентированных языков проектирования. В основном, шумиха, поднятая вокруг ООП, – это не более чем реклама.

Последним достижением в области проектирования является CoDesign, или совместное проектирование программы и аппаратуры. Как правило, в методиках проектирования CoDesign используется объектно-ориентированный подход.

A.2.1 Полнота описания системы

При проектировании любой системы, в том числе ИУС, важна полнота описания. И в структурном подходе и в объектно-ориентированном, полнота описания обеспечивается за счет совместного применения двух понятий: структуры и поведения. Такое разделение наверно не является единственным и самым удобным для проектирования. Но оно существует уже более двадцати лет, используется в десятках методик проектирования и является классическим.

Зачем нужна полнота описания? Представим себе, что мы строим дом (избитый пример, его все приводят :). Если это сарай (или собачья конура, как в книге у Г. Буча), то для строительства нам достаточно словесно описать систему (коробка, с односкатной крышей, примерно три на пять и т.д.) или представить картинку на уровне рисунка 5-ти летнего ребенка. Строитель все поймет, т.к. система простая. Здесь полнота описания не нужна, т.к. детали просты и интуитивно понятны. В голове строителя есть архитектурный шаблон, по которому заполняются пробелы в нашем описании. Шаблон у него образовался со временем, на основе опыта строительства большого количества сараев в прошлом.

Теперь, представим, что мы строим большой торговый центр или супермаркет. Для того, чтобы объяснить строителям, что мы собственно хотим построить, мы должны представить подробный план строительства, учитывающий примерно следующий перечень разных вещей:

- Внешний вид здания;
- План этажей;
- Применяемые материалы;
- Схему электрических, отопительных и других коммуникаций;
- Последовательность (технологию) сборки;
- Календарный план, учитывающий строительные ресурсы и т.д.

Только после перечисления практически всех вопросов, которые могут задать вам строители, можно начинать работу по возведению здания. Если мы упустим какой-либо момент в описании здания, рабочие либо построят не то, что мы хотели, либо работа просто встанет.

Шаблоны, которые есть в голове у рядовых строителей, не подходят для решения всей задачи (они умеют класть кирпичи, ставить окна, штукатурить и т.п.). Во-первых, они не строили раньше таких торговых центров, во-вторых, эти шаблоны слишком мелки, т.к. в голову человека не помещается целиком информация о крупном проекте. Архитектор представляет, как построить все здание, но сам строить он, как правило, не умеет.

Аналогичные проблемы возникают и при проектировании достаточно сложных ИУС.

A.2.2 Взгляд на систему с разных позиций

Что такое структура и поведение? Структура системы – это совокупность частей (элементов и подсистем) и связи между ними. Поведение системы – это изменение структурных составляющих (подсистем и элементов), а также связей между ними во времени.

Для описания двух этих понятий, люди издавна используют графические изображения. К сожалению, пока не существует способа изображения на одной

картинке всей структуры или поведения вычислительной системы. Приходится рассматривать и структуру, и поведение с разных позиций.

Структура может быть представлена следующими способами [28]:

- Совокупность блоков системы и интерфейсов (объект A соединен с объектом B с помощью интерфейса I2C);
- Совокупность объектов и зависимостей (объект «дом» зависит от объекта «электростанция»);
- Схема наследования классов (класс X происходит от классов Y и Z);
- Схема включения классов (агрегация, класс Q включает в себя объекты S и D);

Поведение можно представить в виде [28]:

- Блок-схемы алгоритма;
- Конечного автомата;
- Временной диаграммы (процессограммы, диаграммы взаимодействий и т.п.);
- Потоковой диаграммы (DFD, CFD и т.п.).

Эти перечни не претендуют на полноту. В книгах по проектированию вычислительных систем можно найти другие способы описания структуры и поведения.

A.2.3 Структурная схема

Структура (от лат. structura – строение, расположение, порядок) – это совокупность устойчивых связей объекта, обеспечивающих его целостность и тождественность самому себе, т.е. сохранение основных свойств при различных внешних и внутренних изменениях. Данное определение говорит о наличии связей внутри объекта, и отмечает их значимую роль в образовании целостной и постоянной сущности.

Структурная схема – это графическое описание системы, представленное в виде совокупности блоков и интерфейсов между ними. Как правило, структурные элементы вычислительной системы обозначают прямоугольниками, а связи между ними (интерфейсы) стрелками. В прямоугольниках и над стрелками записывают названия элементов и интерфейсов. Стрелки показывают направленность интерфейсов. Часто на стрелках показывают разрядность цифровых интерфейсов, напряжения для шин питания и т.п. Иногда, толщиной стрелки показывают сложность или производительность интерфейса.

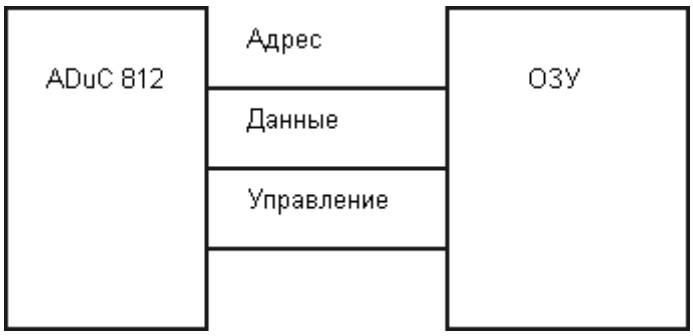


Рисунок 96. Пример структурной схемы.

С помощью структурных схем можно изображать не только аппаратные решения, но и программные (в принципе, с помощью структурной схемы можно изобразить все, что имеет структуру).

A.2.4 Диаграмма потоков данных

Диаграмма потоков данных (Data Flow Diagram, DFD) позволяет графически отобразить один из поведенческих аспектов системы [28]. С помощью DFD вычислительная система представляется в виде множества процессов и потоков данных между процессами.

Процесс – последовательная смена состояний, явлений, ход развития чего-то.

Процесс – вычислительная единица, предназначенная для преобразования потока информации.

- Процесс имеет начальную и конечную стадии своей жизни.
- Процесс протекает во времени.
- Процесс может предавать данные или управлять другим процессом.
- Процесс может принимать данные или управляться другим процессом.

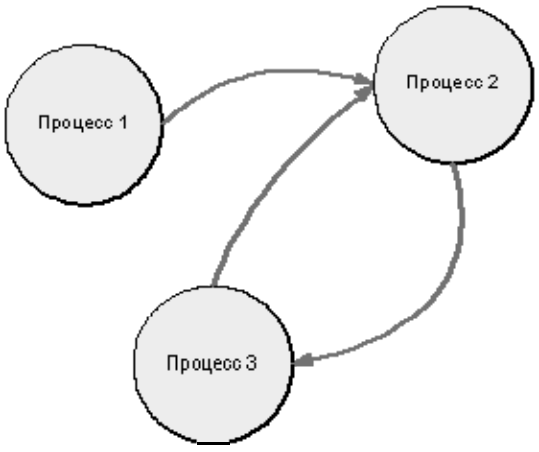


Рисунок 97. Диаграмма потоков данных

В рамках нотации потоковых диаграмм DFD и CFD процессы изображают кружком. В DFD процессы взаимодействуют посредством передачи данных (сплошные стрелки), а в CFD – посредством передачи управления (пунктирные стрелки). Стрелки показывают направление и способ передачи данных или

управления (синхронный или асинхронный). Очень часто, при проектировании ОС РВ смешивают DFD и CFD в рамках одного рисунка.

В качестве примеров процесса можно предложить: процесс или поток (нить, thread), использующийся в Windows или Unix, обработчик прерывания в учебном стенде SDK-1.1.

Для примера рассмотрим потоковую модель драйвера последовательного канала, работающего по прерыванию (асинхронный обмен по прерыванию).

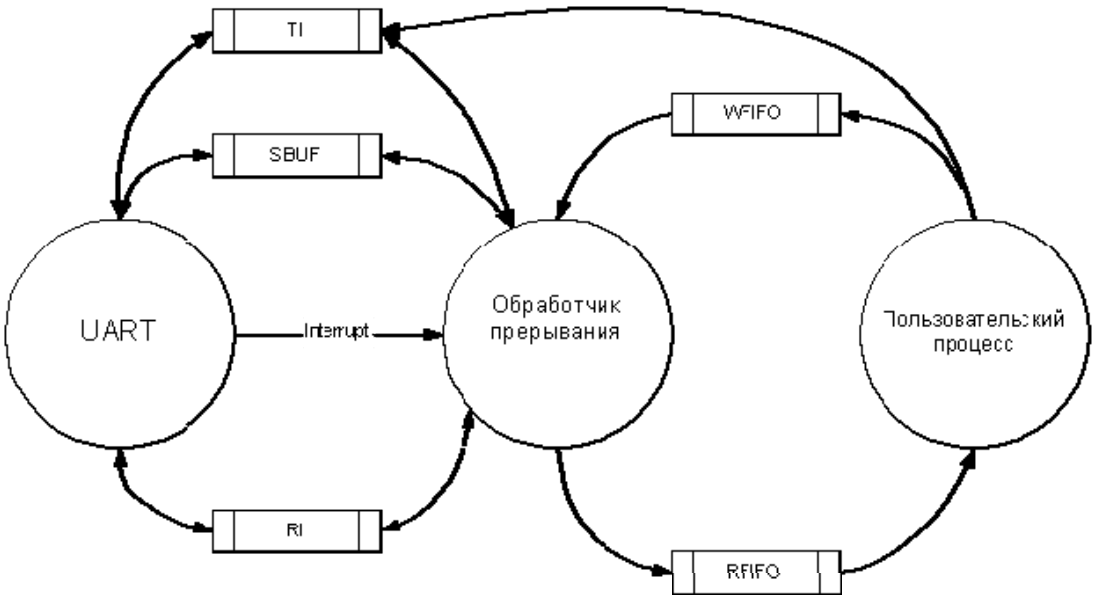


Рисунок 98. Потоковая модель драйвера последовательного канала для SDK-1.1

В этой модели есть три процесса:

- Контроллер последовательного канала UART;
- Обработчик прерывания;
- Пользовательский процесс.

Информация, полученная от UART, записывается процессом «Обработчик прерываний» в очередь RFIFO. Пользовательский процесс может забрать байт из FIFO тогда, когда у него для этого будет возможность (т.е. очередь не будет пуста). После записи байт в WFIFO пользовательский процесс может заниматься своими делами. Обработчик прерывания заберет байт из WFIFO тогда, когда буфер передатчика UART опустеет (при этом обработчик будет вызван).

A.2.5 Конечный автомат

Что представляет собой *конечный автомат*? Автомат может запоминать свое состояние. Такой автомат называется *автоматом с памятью*. Выход автомата без памяти не зависит от предыдущих значений входов, а зависит только от текущих значений входов. Выход автомата с памятью зависит от предыдущих воздействий. Это принципиальное отличие автомата без памяти от автомата с памятью.

Основное понятие автомата с памятью – это его состояние. Если значение выходов автомата без памяти зависит только от входов, то значение выходов автомата с памятью зависит не только от входов, но и от состояния автомата. Состояний у автомата может быть множество. Автомат с памятью, имеющий конечное число состояний, называется *конечным автоматом*. Автомат, который может помнить не только свое текущее состояние, но сохранять и затем восстанавливать свои предыдущие состояния, называется *автоматом с магазинной памятью* [29].

Существует множество определений термина «конечный автомат» (finite-state machine), приведем некоторые из них [29, 52]:

- Конечный автомат – это модуль, имеющий конечное число возможных состояний и функционирующий в дискретном времени.
- Конечный автомат (в теории алгоритмов) – математическая абстракция, позволяющая описывать пути изменения состояния объекта в зависимости от его текущего состояния и входных данных, при условии, что общее возможное количество состояний конечно.
- Конечный автомат (в объектно-ориентированном программировании) – спецификация последовательности состояний, через которые проходит в течение своей жизни объект, или взаимодействие в ответ на происходящие события (а также ответные действия объекта на эти события).
- Конечный автомат (в аппаратной реализации) – модель логического устройства с определенной памятью, предназначенная для обработки информации.
- Конечный автомат – математическая модель устройства с конечной памятью. Конечный автомат перерабатывает множество входных дискретных сигналов во множество выходных сигналов.

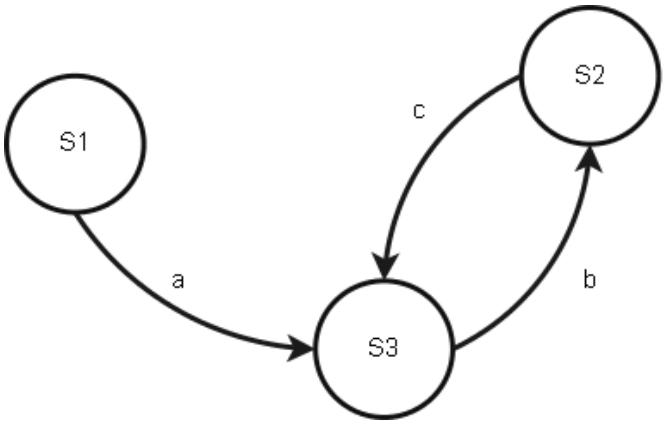


Рисунок 99. Графическое изображение конечного автомата

В данном примере у автомата три состояния: S1, S2 и S3. По условию a автомат переходит из состояния S1 в состояние S2, по условию c автомат переходит из состояния S2 в состояние S3 и по условию b автомат переходит в состояние S2.

Таблица 32. Табличная форма записи конечного автомата

Состояние/условие	a	b	c
S1	S3		
S2			S3
S3		S2	

В левой колонке показаны состояния, в шапке таблицы – условия перехода. В пересечении строки и столбца показано новое состояние, в которое должен перейти автомат их указанного состояния по данному условию [52].

Реализация конечного автомата в виде программы выглядит следующим образом:

```
// Коды состояний
#define S1 0
#define S2 1
#define S3 2

// Глобальные переменные, через которые мы получаем условия
// (например, из обработчика прерывания)
extern int a;
extern int b;
extern int c;

int main( void )
{
int state = S1;

…
  while( 1 )
  {
    switch( state )
    {
    case S1: if( a ) state = S3; break;
    case S2: if( c ) state = S3; break;
    case S3: if( b ) state = S2; break;
    }
  }
return 0;
}
```

Идея конечного автомата развилась из близкого понятия, введенного в 1936 году Аланом Тьюрингом (см. Машина Тьюринга). К достоинствам конечного автомата можно отнести простоту реализации. К сожалению, далеко не все можно представить в рамках этой модели вычислений. Во многих случаях, количество состояний в модели становится настолько большим, что делает ее использование практически невозможным.

Известно деление конечных автоматов на автомат Мура и автомат Мили. В автомате Мура выходные сигналы зависят только от текущего состояния автомата (действия совершаются в состояниях), в автомате Мили выходные сигналы зависят и от текущего состояния, и от входных сигналов автомата (действия совершаются при переходах между состояниями).

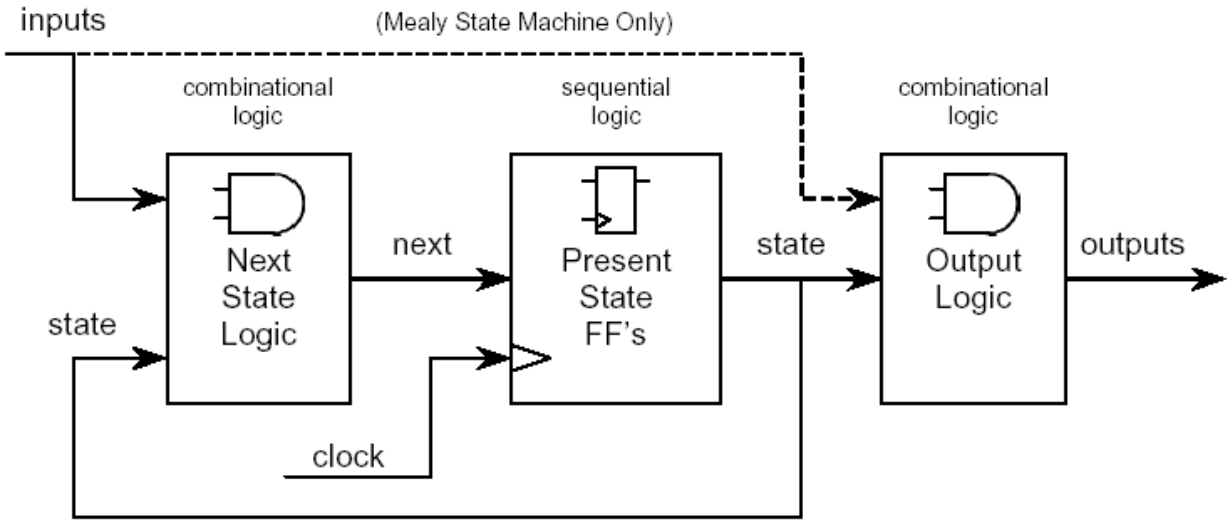


Рисунок 100. Диаграмма конечного автомата

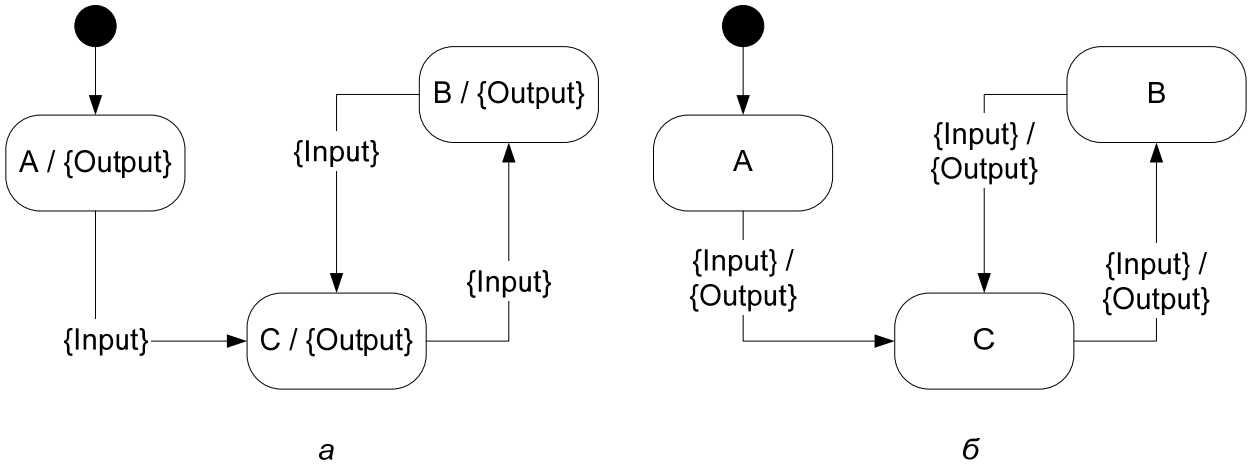


Рисунок 101. а) Конечный автомат Мура; б) Конечный автомат Мили

Автоматная теория, являясь частью теории алгоритмов, позволяет трактовать конечные автоматы как алгоритмы с конечной памятью, многие свойства которых можно исследовать безотносительно от способа реализации автоматов (которые могут быть, например, как аппаратными, так и программными). Таким образом, допустимо констатировать о классе задач, решаемых автоматами, так: если и только если задача алгоритмически разрешима, возможно построить конечный автомат, решающий эту задачу. Перефразируя, можно сказать, что с помощью конечного автомата потенциально возможно решить любую алгоритмически разрешимую задачу.

Исходя из трактовки автоматов, как алгоритмов с конечной памятью, центральной проблемой становится изучение возможностей автоматов в терминах множеств слов, с которыми они работают. Допустимо выделить два основных аспекта работы автоматов:

- Автоматы распознают входные слова, то есть отвечают на вопрос, принадлежит ли поданное на вход слово данному множеству (автоматы-распознаватели).
- Автоматы преобразуют входные слова в выходные, то есть реализуют автоматные отображения (автоматы-преобразователи).

Автоматы-распознаватели принципиально ничем не отличаются от автоматов-преобразователей, то есть допустимо один аспект свести к другому. Но при сведении, понятия и проблемы, важные при первом аспекте, оказываются либо несущественными, либо сильно видоизмененными во втором.

Конечные автоматы широко используются на практике, например в синтаксических анализаторах, а также в других случаях, когда количество состояний объекта и переходов между ними сравнительно невелико.

A.2.5.1 Состояния

Состояние – абстрактный многозначный термин, в общем, обозначающий множество стабильных значений переменных параметров объекта.

Свойства состояния:

- описывает переменные свойства объекта;
- стабильно до тех пор, пока над объектом не будет произведено действие;
- если над объектом будет произведено некоторое действие, его состояние может измениться.

Состояние автомата – это состояние системы, за функционирование которой отвечает автомат. Состояние системы определяется значением всех управляющих переменных. Переменные системы – это изменяющиеся свойства, а управляющие переменные – это те переменные, которые влияют на поведение системы

Состояние представляет собой итоговый результат поведения системы. Например, только что включенный в сеть телефон находится в начальном состоянии: его предыдущее поведение несущественно, при этом он готов к тому, чтобы позвонить или принять звонок. Если кто-нибудь поднимет трубку, телефон перейдет в состояние готовности к набору номера; в этом состоянии мы не ожидаем, что телефон зазвонит, но приготовились к беседе с одним или несколькими абонентами. Если кто-либо наберет ваш номер, а телефон находится в начальном состоянии (трубка положена), то когда вы поднимете трубку, телефон перейдет в состояние с установленным соединением, и вы сможете поговорить со звонившим.

В любой момент времени состояние объекта определяет набор свойств (обычно статический) объекта и текущие (обычно динамические) значения этих

свойств. Под "свойствами" подразумевается совокупность всех связей и атрибутов объекта. Мы можем обобщить понятие состояния так, чтобы оно было применимо и к объекту, и к классу, так как все объекты одного класса "живут" в одном пространстве состояний. Это пространство может представлять собой неопределенное, хотя конечное множество возможных (но не всегда ожидаемых или желаемых) состояний. Каждое состояние должно иметь имя.

A.2.5.2 Переходы

Событием мы называем любое происшествие, которое может быть причиной изменения состояния системы. Изменение состояний называется переходом. На диаграмме переходов и состояний он изображается дугой. Каждый переход соединяет два состояния. Состояние может иметь переход само в себя; обычно есть несколько различных переходов в одно и тоже состояние, но все переходы должны быть уникальны в том смысле, что ни при каких обстоятельствах не может произойти одновременно два перехода из одного состояния.

Например, в поведении гидропонной теплицы играют роль следующие события:

- Посажена новая партия семян.
- Урожай созрел и готов к сбору.
- Из-за плохой погоды упала температура в теплице.
- Отказало охлаждающее устройство.
- Наступил заданный момент времени.

Идентификация событий, подобных этим, позволяет определить границы поведения системы и распределить обязанности по осуществлению этого поведения между отдельными классами.

Каждое из первых четырех перечисленных выше событий, вероятно, вызывает некоторое действие – например, начало или остановку выполнения некоторого плана сельскохозяйственных работ по посеву, включение нагревателя или посылку сигнала тревоги технику, обслуживающему систему. Отсчет времени – это другое дело: хотя секунды и минуты не имеют значения (посевы растут, очевидно, не так быстро), наступление нового часа или суток может вызвать некоторый сигнал, например, включить/выключить лампочки и изменить температуру в теплице, чтобы имитировать смену дня и ночи, необходимую для роста растений.

Действием мы называем операцию, которая, с практической точки зрения, требует нулевого времени на выполнение. Например, включение сигнала тревоги – действие. Обычно действие означает вызов метода, порождение другого события, запуск или остановку процесса. Деятельностью мы называем операцию, требующую некоторого времени на свое выполнение. Например,

нагрев воздуха в теплице – деятельность, запускаемая включением нагревателя, который может оставаться включенным неопределенное время, до тех пор, пока не будет выключен явной командой.

Модель событий, передающих сообщения, которую предложил Харел, концептуально безупречна, но ее нужно приспособить к объектному подходу. При анализе мы можем давать предварительные названия событиям и действиям, в общих чертах отражая наше понимание предметной области [28].

A.2.5.3 Пример использования

Рассмотрим более сложный пример практического использования конечных автоматов в программировании

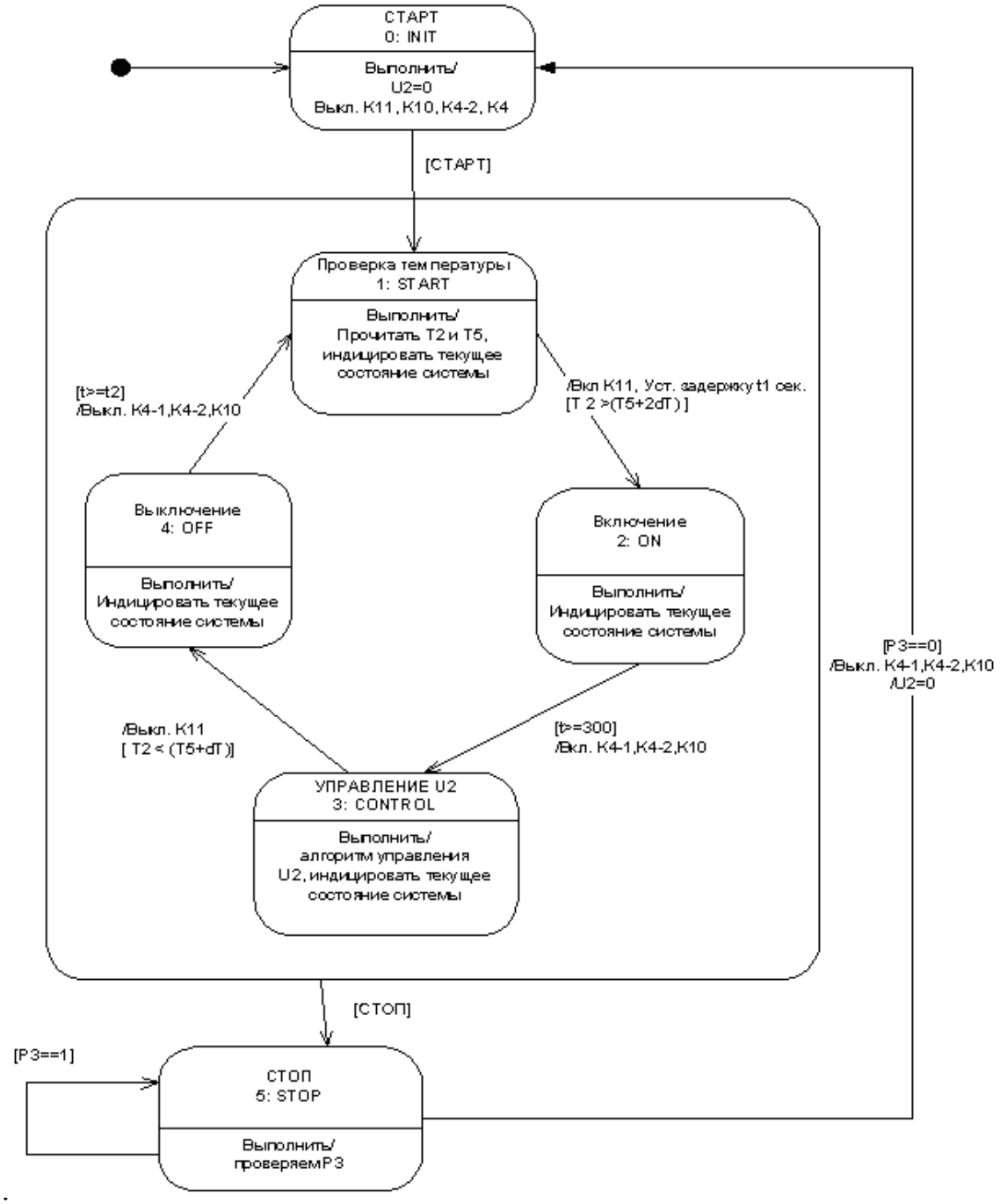


Рисунок 102. Конечный автомат, описывающий работу устройства

Данная диаграмма состояний описывает алгоритм управления некоторым устройством.

Как преобразовать ее в привычный исходный текст на языке Си? Достаточно просто. Для начала, заведем переменную, в которой мы будем держать состояние нашего конечного автомата. Сразу замечу: переменная должна быть статической, т.е. она ни в коем случае не должна находиться в регистре процессора или в стеке.

```
unsigned char State=0;   // Состояние автомата.
```

Далее, описываем сам автомат. Переменные Ki – это дискретные выходы. Pi – дискретные входы. Символом Ti мы обозначим значение, снятое с термодатчика, ti – время.

Для реализации автомата на языке Си проще и нагляднее воспользоваться оператором SWITCH. Каждый CASE является одним из состояний автомата. Переход из состояния в состояние осуществляется путем записи номера нового состояния в переменную State.

```
switch(State)

 {
 case 0: // INIT
       U2=0;
       K11=0; K10=0; K4_2=0; K4=0;
       if(START) State=1;      // Переход в состояние 1
       break;
 case 1: // START
       // Читаем температуры T5 и T2
       // :
       if(STOP) State=5;
       if(T2 > (T5+2*dT) )
       {
             State=2;
             K11=1;
             // Установить задержку для таймера t1
       }
       break;
 case 2: // ON
       // Индицируем текущее состояние системы
       // :
       if(STOP) State=5;

       if(t>=300)
       {
             K4_1=1, K4_2=1, K10=1;
             State=3;
       }
       break;
 case 3: //CONTROL
       // Выполняем алгоритм управления нашим устройством
       // :
       if(STOP) State=5;

       if(T2 < (T5+dT) )
       {
             K11=0;
```

```
            State=4;
      }
      break;
case 4: //OFF
      // Индицируем текущее состояние системы
      // :
      if(STOP) State=5;

      if(t >=t2)
      {
            K4_1=0, K4-2=0,K10=0;
            State=1;
      }
      break;
case 5: // STOP
      // Останов системы
      // :
      // Выполнить проверку P3
      if(P3==0)
      {
            K4-1=0, K4-2=0, K10=0, U2=0;
            State=0;
      }
      break;
}
```

Как видно из рисунка и исходного текста на языке С, автомат может находиться в шести состояниях. Переход от состояния к состоянию осуществляется после анализа условий. Условия (показаны в квадратных скобках на дугах) реализованы оператором IF. Если на дугах указано действие (отмечается символом '/') то оно исполняется внутри IF, в процессе перехода. Действия, указанные внутри прямоугольников выполняются каждый раз, при попадание в данное состояние.

Как видно из исходного текста, оператор SWITCH , запущенный один раз выполняет только один шаг конечного автомата. Для того, чтобы выполнить всю программу необходимо каким-то образом повторять вызов описанной выше конструкции. Это может быть выполнено в виде банального оператора цикла WHILE или более сложным способом. В частности автоматы удобно использовать внутри обработчиков прерывания от таймера, т.к. время выполнения программы в таком случае достаточно просто проанализировать [52].

В чем удобство использования конечных автоматов? Во-первых, – простота реализации. Во-вторых, детерминированность и, как следствие детерминированности – тестопригодность. В каждый момент времени выполняется конкретный шаг алгоритма. Состояние такого автомата описывается всего одной переменной, если известно состояние переменной – однозначно известен текущий шаг.

К сожалению, автоматные модели далеко не идеальны. Самым крупным недостатком автоматной модели является трудность описания сложных систем с большим числом состояний.

A.3 Стиль кодирования для языка Си

Одним из важнейших факторов, влияющих на способность программы к развитию, является ее понятность. Одним из существенных факторов понятности программы, в свою очередь, является информативность исходного текста. Если исходный текст не является хорошо читаемым, то есть написан без соблюдения определенного стиля и системы и представляет собой "мешанину" операторов и знаков препинания, то вносить изменения в него очень сложно даже автору. Такая программа, безусловно, не является информативной.

Сложности модификации значительно возрастают по прошествии времени и при необходимости работать с чужой программой. Рассмотрим ряд требований и рекомендаций, позволяющих выработать хороший стиль оформления программ, повышающий ее информативность. Изложение будем вести на примере языка C/C++, хотя все, сказанное ниже, относится к любому алгоритмическому языку.

A.3.1 Количество операторов в строке

Для улучшения читаемости исходного текста программы рекомендуется писать не более одного оператора в строке, что вызвано особенностями человеческого восприятия текста. Кроме того, это облегчает пошаговую отладку в символьных отладчиках. Не следует опасаться того, что программа слишком вырастет в длину, так как реальные программы и без того настолько длинны, что несколько "лишних" страниц (или даже десятков страниц) не меняют общую ситуацию. Выигрыш же в понятности с избытком покрывает увеличение длины.

Таблица 33. Количество операторов в строке

Неправильно	Правильно

`int *ptr; ptr = new int [100];`
`ptr [0] = 0;`
	`int *ptr;`
	`ptr  = new int [100];`
	`ptr [0] = 0;`

Два оператора в строке вполне допустимы, если второй подчинен первому, причем является единственным подчиненным, например:

```
for( i=0; i < size; i++ ) m [i] = 0;
```

Использование двух и более операторов в строке не только допустимо, но и желательно, если это позволяет подчеркнуть некую систему в локальной последовательности операторов, например:

```
x1 = Tr1 [0]; y1 = Tr1 [1]; z1 = Tr1 [2];
x2 = Tr2 [0]; y2 = Tr2 [1]; z2 = Tr2 [2];
x3 = Tr3 [0]; y3 = Tr3 [1]; z3 = Tr3 [2];
```

A.3.2 Отступы

Правильное использование отступов являются ключевым методом обеспечения читаемости. Идея состоит в том, что отступы зрительно показывают подчиненность (иерархию) операторов. При этом директивы препроцессора (`#include`, `#define` и т.д.), описания классов, структур, типов, глобальных данных и определения функций всегда имеют наивысший приоритет, то есть начинаются с крайней левой позиции, например:

```
#include <stdio.h>

#define NAME_SIZE 256

int main()
{
...
}
```

Основные правила использования отступов таковы.

Правило 1. *Операторы одного уровня иерархии должны иметь равный отступ.*

Таблица 34. Правило 1. Равный отступ

Неправильно	Правильно

`printf( "Enter dimension: " );`
`  scanf( "%d", &dim );`
`  ptr = new int [dim];`
`ptr [0] = 0;`
	`printf( "Enter dimension: " );`
	`scanf( "%d", &dim );`
	`ptr  = new int [dim];`
	`ptr [0] = 0;`

Правило 2. *Подчиненные операторы должны быть сдвинуты вправо по отношению к управляющему оператору, образуя следующий уровень иерархии.*

Таблица 35. Правило 2. Сдвиг подчинённых операторов

Неправильно	Правильно

`if( f == NULL )`
`printf( "No file\n" );`
`else`
`printf( "Start..\n" );`
	`if( f == NULL )`
	`  printf( "No file\n" );`
	`else`
	`  printf( "Start..\n" );`

Правило 3. *Размер сдвига должен быть постоянным.*

Таблица 36. Правило 3. Постоянный размер сдвига подчинённых операторов

Неправильно	Правильно

`if( ptr == NULL )`
`  return -1;`
`for( i=0; i<dim; i++ )`
`  ptr [i] = i;`
	`if( ptr == NULL )`
	`  return -1;`
	`for( i=0; i<dim; i++ )`
	`  ptr [i] = i;`

Размер сдвига не должен быть ни слишком мал, ни слишком велик. Оптимальная величина составляет 2-5 пробелов. Наиболее часто для сдвигов используют табуляцию, устанавливая при этом для нее желаемый шаг. Последняя возможность поддерживается большинством интегрированных сред разработчика.

A.3.3 Операторные скобки

Существует два основных стиля расстановки операторных скобок. При использовании первого стиля открывающаяся скобка помещается на той же строке, что и управляющая конструкция, а закрывающаяся – строго на уровне управляющей конструкции:

```
int factorial( int n ) {
  if( n > 1 )
    return n * factorial( n-1 );
  if( n < 0 ) {
    fprintf( stderr, "Factorial error: negative argument\n" );
    return -1; //Заведомо невозможный результат
  }
  return 1;
}
```

Второй подход покажем на том же примере:

```
int factorial( int n )
{
  if( n > 1 )
    return n * factorial( n-1 );
  if( n < 0 )
  {
    fprintf( stderr, "Factorial error: negative argument\n" );
    return -1; //Заведомо невозможный результат
  }
  return 1;
}
```

Как видно, отличие состоит в положении открывающейся скобки. Однако закрывающаяся скобка в обоих случаях должна находиться на уровне управляющего оператора или описания. Многим специалистам представляется, что второй подход оправдан в большей мере, так как обеспечивает улучшенную наглядность. Открывающаяся и закрывающаяся скобки при этом располагаются строго друг под другом, что помогает находить начало и конец составного оператора, функции или описания класса.

Заметим, что наличие или отсутствие скобок не влияет на наличие и размер отступов, что видно в приведенных примерах.

Изредка можно встретить еще и третий стиль, подобный второму, но характеризующийся расположением скобок где-нибудь внутри отступа:

```
int factorial( int n )
 {
  if( n > 1 )
    return n * factorial( n-1 );
  if( n < 0 )
```

```
    {
     fprintf( stderr, "Factorial error: negative argument\n" );
     return -1; //Заведомо невозможный результат
    }
  return 1;
}
```

Этот стиль, несомненно, хуже второго, поскольку не только не привносит какой-либо дополнительной ясности, но и несколько снижает ее. Во-первых, выбор положения скобок где-то внутри отступа произволен и ничем не обоснован. Во-вторых, количество вертикальных зрительных линий отступов из-за этого удваивается и составляет вместо обычных 4-5 линий 8-10, что снижает наглядность без повышения информативности.

A.3.4 Пробелы

Особенность зрительного восприятия человека такова, что пробелы распознаются лучше других знаков синтаксиса. Поэтому отдельные элементы текста необходимо отделять пробелами, несмотря на то, что первые, возможно, уже отделены другими знаками препинания (скобки, запятые, точки с запятой и т.д.). В особенности важно отделять стоящие рядом операторы и списки аргументов функций:

Таблица 37. Расстановка пробелов

Неправильно	Правильно
`while(i++<dim)` `move(a,b,ptr [base+off*i]);`	`while( i++ < dim )` `move( a, b, ptr [base + off*i] );`

Дополнительные пробелы могут быть также использованы для выравнивания сходных по смыслу или однотипных частей выражений с целью улучшения наглядности, например, при объявлении переменных и для серии присваиваний:

```
int  a, size;
char *buf;
float lenght1, lenght2;

. . .

a       = 1;
lenght1 = GetLength();
lenght2 = 0;
size    = (int) lenght1;
```

Практику правильного использования пробелов можно также изучить по всем остальным примерам данного подраздела.

A.3.5 Пустые строки

Использование пустых строк является важным средством для выделения участков программы. При этом имеет смысл отделять:

1. Определения переменных:

```
char str [80];
int counter = 0;

fgets( str, 79, infile);
counter++;
```

2. Последовательности однотипных инструкций или директив:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define NAME_SIZE 256
#define MAX_LEN 3000
```

3. Функции:

```
int main()
{
. . .
}

char *get_name(FILE *f)
{
. . .
}
```

4. Любые логически завершенные блоки кода:

```
printf( "Enter size and delta: " ); //Блок ввода данных
scanf( "%d", &size );
scanf( "%f", &delta );

for( i=0; i<size; i++ ) //Блок использования данных
{
  a [i] -= delta;
  b [i] += delta;
}
```

A.3.6 Имена

Типичной ошибкой начинающих является стремление давать всем переменным неосмысленные однобуквенные имена, например m, n, a, s, p и т.п. Это глубоко порочная практика, поскольку при этом теряется сам смысл понятия имя. Однобуквенные имена принято давать только индексам. Исключением являются случаи, когда количество переменных в процедуре очень мало (порядка 1-3 переменных), и смысл их хорошо понятен из контекста или комментариев. Пример – описанная выше функция factorial. Все переменные, имеющие сколько-нибудь важное значение в программе, необходимо снабжать именами, в той или иной мере характеризующими их назначение, например, filename (имя файла), int_vector (целочисленный вектор),

size (размер), sum (сумма), maximum (максимум) и т.п. При плохом знании английского языка можно использовать звуковые аналогии русским буквам, например nazv_faila, razmer, summa и т.п. Это не в полной мере соответствует представлению о "хорошем стиле", но, несомненно, гораздо нагляднее использования совсем бессмысленных имен. Еще в большей мере сказанное относится к именам функций и классов. Такие имена часто делают довольно длинными для улучшения понимаемости их смысла, а отдельные компоненты имен начинают с большой буквы, например ObjectList, ArcSet и т.п. Имена функций при этом рекомендуется начинать с глагола, например GetPersonName, SetNewDate и т.п. Альтернативным способом выделения компонент в сложных названиях является использование символа подчеркивания, например add_record, copy_object и т.п. Все сказанное вполне относится и к именам файлов с программами. Имена файлов должны нести обязательную смысловую нагрузку, поясняя свое "содержимое". Например, заголовочный файл, содержащий описание класса vector логичнее всего назвать vector.h или vector_description.h, а файл, содержащий реализацию методов этого класса – vector.cpp или vector_implementation.cpp. Все современные операционные системы (UNIX, Windows и т.д.) поддерживают длинные имена файлов, вследствие чего искусственно упрощать и укорачивать их нет необходимости. Более того, при разработке крупного продукта количество исходных файлов измеряется сотнями. В этой ситуации невозможно "помнить", в каком файле что находится, а значит длинные и понятные имена жизненно необходимы. Появляется также необходимость организации хранения файлов в виде дерева. В этом случае, указанные в примере файлы могут храниться в отдельном подкаталоге vector и именоваться vector\vector.h и vector\vector.cpp.

A.3.7 Комментарии

Время, потраченное на написание комментариев, многократно окупится при любых модификациях программы. Однако комментировать все подряд, включая самоочевидные действия, как в следующем примере, тоже не стоит:

```
size = 10; //Присвоить size значение 10
for( i=0; i<size; i++) //Цикл по i от 0 до size
{
  . . .
}
```

Такого рода комментарии только загромождают программу.

Комментировать следует:

1. Заголовок файла, описывая содержимое данного файла;
2. Заголовок функции, поясняя назначение ее аргументов и смысл самой функции;
3. Вводимые переменные и структуры данных;
4. Основные этапы и особенности реализуемых алгоритмов;

5. Любые места, которые трудны для быстрого понимания, в особенности использование различных программных "трюков" и нестандартных приемов.

Предметный указатель

Ф

Ц

Ч

Ш

Ю

Я

Литература

1. ADuC812: MicroConverter, Multichannel 12-Bit ADC with Embedded Flash MCU Data Sheet (Rev E, 04/2003) [Electronic resource] / Norwood: Analog Devices Inc., 2003. – Режим доступа: http://www.analog.com/static/imported-files/data_sheets/ADUC812.pdf, свободный. – Загл. с экрана.
2. Barr, M., Massa, A.N. Programming embedded systems: with C and GNU development tools [Text] / M. Barr, A.N Massa. – 2nd ed. – Sebastopol: O'Reilly Media Inc., 2006. – 301 p. – ISBN 0596009836
3. CAN Specification Version 2.0 [Text]. – Stuttgart: Robert Bosch GmbH, 1991. – 73 p.
4. GNU make manual [Electronic resource]. – Режим доступа: http://www.gnu.org/software/make/manual/html_node/index.html, свободный. – Загл. с экрана.
5. Heath, S. Embedded systems design. EDN series for design engineers [Text] / S. Heath. – 2nd ed. – Oxford: Elsevier Science, 2003. – 430 p. – ISBN 0 7506 5546 1
6. Hennessy, J.L., Patterson, D.A., Goldberg, D. Computer architecture: a quantitative approach [Text] / J.L. Hennessy, D.A. Patterson, D. Goldberg. – 3rd ed. – San Francisco: Morgan Kaufmann, 2003. – 883 p. – ISBN 1-55860-596-7
7. I²C-bus specification and user manual (Rev. 03, 06/2007) [Electronic resource]. – Режим доступа: http://www.nxp.com/documents/user_manual /UM10204.pdf, свободный. – Загл. с экрана.
8. Information Technology for Manufacturing: Reducing Costs and Expanding Capabilities [Text] / K. Ake [et al.]. – CRC Press, 2003. – 328 p. – ISBN 1574443593
9. Koenig, D. Computer-Integrated Manufacturing: Theory and Practice [Text] / D. Koenig. – Hemisphere: Taylor & Francis, 1990. – 248 p. – ISBN 0891168745
10. Koren, Y. Computer Control of Manufacturing Systems [Text] / Y. Koren. – McGraw-Hill Education (ISE Editions), 1984. – 304 p. – ISBN 0-07-035341-7
11. Lee, A. Edward. Cyber Physical Systems: Design Challenges [Text] / Edward A. Lee. – International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC). – May, 2008. – 7 p.
12. Lee, A. Edward. Cyber-Physical Systems – Are Computing Foundations Adequate? [Text] / Edward A. Lee. – Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap. – October 16-17, 2006, Austin, TX. – 9 p.
13. Lee, A. Edward. The Problem with Threads [Text] / Edward A. Lee. – EECS Department University of California, Berkeley Technical Report No. UCB/EECS-2006-1. – Jan. 10, 2006. – 19 p.

14. PCF8583 – Clock/calendar with 240 x 8-bit RAM [Electronic resource] / The Netherlands: Philips Semiconductors, 1997 (Jul 15). – Режим доступа: http://www.nxp.com/documents/data_sheet/PCF8583.pdf, свободный. – Загл. с экрана.
15. Sangiovanni-Vincentelli, A. Quo Vadis SLD: Reasoning About the Trends and Challenges of System Level Design [Text] / A. Sangiovanni-Vincentelli. – Proceedings of the IEEE. – 95(3), 2007. – P. 467-506.
16. SDCC Compiler User Guide [Electronic resource]. – Режим доступа: http://sdcc.sourceforge.net/doc/sdccman.html/, свободный. – Загл. с экрана.
17. Schultz, T.W. C and the 8051 [Text] / T.W. Schultz. – Otsego: PageFree Publishing Inc., 2004. – 3rd ed. – 412 p. – ISBN 1-58961-237-X
18. The I²C-bus specification (version 2.1, 2000) [Electronic resource]. – Режим доступа: http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf, свободный. – Загл. с экрана.
19. Two-wire Serial EEPROM AT24C02A/AT24C04A [Electronic resource] / Atmel Corporation, 2007. – Режим доступа: http://www.atmel.com/dyn/resources/prod_documents/doc5083.pdf, свободный. – Загл. с экрана.
20. Waldner, J.-B. CIM: Principles of Computer-Integrated Manufacturing [Text] / Jean-Baptiste Waldner. – John Wiley & Sons Inc., 1992. – 206 p. – ISBN 047193450X
21. Wolf, W.H. Computers as Components: Principles of Embedded Computing Systems Design [Text] / W.H. Wolf. – San Francisco: Morgan Kaufmann, 2005. – 656 p. – ISBN 978-0-12-369459-1
22. Zurawski, R. Embedded systems handbook. Industrial information technology series [Text] / R. Zurawski. – Boca Raton: CRC Press, 2006. – 1160 p. – ISBN 0-8493-2824-1
23. Баранов, С.Н., Ноздрунов, Н.Р. Язык Форт и его реализации [Текст] / С.Н. Баранов, Н.Р. Ноздрунов. – Л.: Машиностроение, 1988. – 157 с. – ISBN 5-217-00324-3
24. Бень, Е.А. RS-485 для чайников [Электронный ресурс] / Е.А. Бень. – Иваново: «ТехноСфера», 2003. – Режим доступа: http://www.ivtechno.ru/news_16.html, свободный. – Загл. с экрана.
25. Болл, Стюарт Р. Аналоговые интерфейсы микроконтроллеров [Текст] / Стюарт Р. Болл. – М.: Издательский дом «Додэка-XX1», 2007. – 360 с: ил. – ISBN 978-5-94120-142-6
26. Броуди, Л. Начальный курс программирования на языке Форт [Текст] / Л. Броуди: пер. с англ.; предисл. И.В. Романовского. – М.: Финансы и статистика, 1990 – 352 с. – ISBN 5-279-00252-6
27. Брукс, Ф. Мифический человеко-месяц или как создаются программные системы [Текст] / Ф. Брукс. – СПб.: Символ-Плюс, 2010. – 304 с. – ISBN 5-93286-005-7, ISBN 0-201-83595-9

28. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами на языке C++ [Текст] / Г. Буч. – СПб.: Бином, Невский диалект, 1998. – 560 с. – ISBN 0-8053-5340-2, ISBN 5-7989-0067-3
29. Ваганов, С.А. Автоматное программирование во “Флоре” [Электронный ресурс] / С.А. Ваганов. – СПб.: СПбГУ ИТМО, 2002. – Режим доступа: http://is.ifmo.ru/progeny/vaganov/, свободный. – Загл. с экрана.
30. ГОСТ 27.002-89. Надежность в технике. Основные понятия. Термины и определения [Текст]. – Введ. 1990-07-01. – М.: ИПК Издательство стандартов, 2002. – 64 с.
31. Гук, М.Ю. Аппаратные средства IBM PC. Энциклопедия [Текст] / М.Ю. Гук. – 3-е изд. – СПб.: Питер, 2006. – 1072 с.: ил. – ISBN 5-469-01182-8
32. Демарко, Т. Deadline. Роман об управлении проектами [Текст] / Т. Демарко. – М.: Манн, Иванов и Фербер, 2010. – 352 с. – ISBN 978-5-91657-076-2
33. Демарко, Т., Листер, Т. Человеческий фактор. Успешные проекты и команды [Текст] / Т. Демарко, Т. Листер. – СПб.: Символ-Плюс, 2009. – 256 с. – ISBN 978-5-93286-061-8, ISBN 5-93286-061-8, ISBN 0-932633-43-9
34. Игнатов, В. Эффективное использование GNU Make [Электронный ресурс] / В. Игнатов. – М.: Центр Информационных Технологий, 2000. – Режим доступа: http://www.citforum.ru/operating_systems/gnumake/index.shtml, свободный. – Загл. с экрана.
35. История компьютера. Жидкокристаллические индикаторы [Электронный ресурс]. – Режим доступа: http://chernykh.net/content/view/659/, свободный. – Загл. с экрана.
36. Карпов, А., Рыжков, Е. 20 ловушек переноса кода Си++ на 64-битную платформу [Электронный ресурс]. – Тула: ООО "СиПроВер", 2007. – Режим доступа: http://www.viva64.com/content/articles/64-bit-development/?f=20_issues_of_porting_C++_code_on_the_64-bit_platform_rus.html&lang=ru&content=64-bit-development, свободный. – Загл. с экрана.
37. Келли, М., Спайс, Н. Язык программирования Форт [Текст] / М. Келли, Н. Спайс: пер. с англ. – М.: Радио и связь, 1993. – 320 с. – ISBN 5-256-00438-7
38. Керниган, Б., Пайк, Р. Практика программирования [Текст] / Б. Керниган, Р. Пайк. – М.: Вильямс, 2004. – 288 с.: ил. – ISBN 5-8459-0679-2, ISBN 0-201-61586-X
39. Керниган, Б., Ритчи, Д. Язык программирования C [Текст] / Б. Керниган, Д. Ритчи. – 5-е изд. – М.: Вильямс, 2009. – 304 с. – ISBN 978-5-8459-0891-9, ISBN 5-8459-0891-4, ISBN 0-13-110362-8
40. Ключев, А.О. Программное обеспечение встроенных вычислительных систем [Текст] / А.О. Ключев [и др.]. – СПб.: СПбГУ ИТМО, 2009. – 212 с.

41. Коллинз-Сассман, Б., Фитцпатрик, У.Б., Пилато, К.М. Управление версиями в Subversion [Электронный ресурс] / Бен Коллинз-Сассман, Брайан У. Фитцпатрик, К. Майкл Пилато. – Режим доступа: http://svnbook.red-bean.com/nightly/ru/index.html, свободный. – Загл. с экрана.
42. Непейвода, Н.Н. Стили и методы программирования: Курс лекций: Учеб. пособие [Текст] / Н.Н. Непейвода. – М.: Интернет-Университет информационных технологий (ИНТУИТ), 2005. – 320 с. – ISBN 5-95560-023-0
43. Непейвода, Н.Н., Скопин, И.Н. Основания программирования [Текст] / Н.Н. Непейвода, И.Н. Скопин. – М.-Ижевск: Институт компьютерных исследований, 2003. – 868 с. – ISBN 5-93972-299-7
44. Олссон, Г., Пиани, Д. Цифровые системы автоматизации и управления [Текст] / Г. Олссон, Д. Пиани. – СПб.: Невский Диалект, 2001. – 557 с. – ISBN 5-7940-0069-4, ISBN 5-7940-0082-1
45. Пентус, А.Е., Пентус, М.Р. Математическая теория формальных языков [Текст] / А.Е. Пентус, М.Р. Пентус. – М.: Интернет-Университет информационных технологий (ИНТУИТ), 2006. – 248 с. – ISBN 5-9556-0062-0
46. Подбельский, В.В., Фомин, С.С. Программирование на языке Си [Текст] / В.В. Подбельский, С.С. Фомин. – 2-е, доп. изд. – М.: Финансы и статистика, 2005. – 600 с.: ил. – ISBN 5-279-02180-6, ISBN 5-279-02180-2
47. Себеста, Роберт У. Основные концепции языков программирования [Текст] / Роберт У. Себеста; пер. с англ. – 5-е изд. – М.: Вильямс, 2001. – 672 с. – ISBN 5-8459-0192-8 (рус.), ISBN 0-201-75295-6 (англ.)
48. Страуструп, Б. Язык программирования C++. Специальное издание [Текст] / Б. Страуструп. – 3-е изд. – СПб.: Бином, Невский Диалект, 2008. – 1104 с. – ISBN 5-7989-0226-2, ISBN 5-7940-0064-3, ISBN 0-201-70073-5
49. Таненбаум, Э. Архитектура компьютера [Текст] / Э. Таненбаум. – 5-е изд. – СПб.: Питер, 2007. – 844 с: ил. – ISBN 5-469-01274-3.
50. Учебный стенд SDK-1.1. Руководство пользователя (Версия 1.0.11) [Электронный ресурс]. – СПб.: СПбГУ ИТМО, 2009. – Режим доступа: http://embedded.ifmo.ru/sdk/sdk11/doc/sdk11_userm_v1_0_11.pdf, свободный. – Загл. с экрана.
51. Цикл лекций по микроконтроллерам семейства MCS-51 [Электронный ресурс]. – Режим доступа: http://digital.sibsutis.ru/content.htm, свободный. – Загл. с экрана.
52. Шалыто, А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления [Текст] / А.А. Шалыто. – 2-е изд. – СПб.: Наука, 1998. – 628 с.

x

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра ВТ СПбГУ ИТМО создана в 1937 году и является одной из старейших и авторитетнейших научно-педагогических школ России.

Традиционно основной упор в подготовке специалистов на кафедре делается на фундаментальную базовую подготовку в рамках общепрофессиональных и специальных дисциплин, охватывающих наиболее важные разделы вычислительной техники.

Кафедра является одной из крупнейших в университете. Учебными курсами и научно-исследовательскими работами руководят 8 профессоров и 16 доцентов. На кафедре обучаются более 500 студентов и 30 аспирантов.

Кафедра имеет собственные компьютерные классы и специализированные исследовательские лаборатории, оснащенные современной вычислительной и оргтехникой, уникальным инструментальным и технологическим оборудованием, измерительными приборами и программным обеспечением.

В 2007-2008 гг. коллективом кафедры была успешно реализована инновационная образовательная программа СПбГУ ИТМО по научно-образовательному направлению «Встроенные вычислительные системы».

Начиная с 2009 года кафедра вычислительной техники является активным участником реализации программы развития национального исследовательского университета СПбГУ ИТМО, вошла в состав крупнейшего в СПбГУ ИТМО научно-исследовательского центра «Интеллектуальные системы управления и обработки информации».

Аркадий Олегович Ключев

Динара Раисовна Ковязина

Павел Валерьевич Кустарев

Алексей Евгеньевич Платунов

АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА ВСТРАИВАЕМЫХ СИСТЕМ

Учебное пособие

В авторской редакции

Дизайн Д.Р.Ковязина
Верстка Д.Р.Ковязина
Редакционно-издательский отдел Санкт-Петербургского государственного университета информационных технологий, механики и оптики
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати 18.06.2009
Заказ №
Тираж 100 экз.
Отпечатано на ризографе

Ключев А.О., Ковязина Д.Р., Кустарев П.В., Платунов А.Е.

АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА ВСТРАИВАЕМЫХ СИСТЕМ

УЧЕБНОЕ ПОСОБИЕ

Санкт-Петербург
2010

Редакционно-издательский отдел
Санкт-Петербургского государственного
университета информационных технологий,
механики и оптики
197101, Санкт-Петербург, Кронверкский пр., 49