Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Оренбургский государственный университет»

А.В. Хлуденев

СРЕДСТВА РАЗРАБОТКИ И ОТЛАДКИ ПРОГРАМ ДЛЯ МИКРОКОНТРОЛЛЕРОВ

Учебное пособие

Рекомендовано ученым советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по образовательным программам высшего образования по направлениям подготовки 11.03.03 Конструирование и технология электронных средств и 11.03.04 Электроника и наноэлектроника

Оренбург
2019

УДК 621.3 : 681.31
ББК 32.85+32.973
Х 60

Рецензент - доктор технических наук, профессор В.Н. Булатов

Хлуденев, А.В.
Х 60 Средства разработки и отладки программ для микроконтроллеров [Электронный ресурс]: учебное пособие / А.В. Хлуденев; Оренбургский гос. ун-т. - Оренбург: ОГУ, 2019.
ISBN 978-5-7410-2400-3

В учебном пособии рассмотрены принципы организации и функционирования инструментальных средств, используемых для разработки и отладки программ для встраиваемых устройств на основе микроконтроллеров.

Учебное пособие предназначено для обучающихся по образовательным программам высшего образования по направлениям подготовки 11.03.03 Конструирование и технология электронных средств и 11.03.04 Электроника и наноэлектроника при изучении дисциплины «Отладочные средства микропроцессорных систем».

УДК 621.3 : 681.31
ББК 32.85+32.973

ISBN 978-5-7410-2400-3

Содержание

Введение

Многие технические средства, окружающие современного человека содержат микропроцессоры (МП). Они используются не только для построения компьютеров, но и встраиваются в бытовые и медицинские приборы, мобильные телефоны и другие устройства связи, транспортные средства, оборудование различного назначения (промышленного, оборонного и других).

Первые модели МП не планировалось использовать для построения компьютеров, они предназначались для замены «жесткой логики», то есть таких цифровых электронных устройств, алгоритм работы которых полностью определяется их структурой. Значительная часть этих устройств были предназначены для управления различными объектами или процессами и получили название – контроллеры. При этом переход с одного алгоритма управления на другой нельзя было изменить без изменения структуры устройства. Такие устройства строились на основе интегральных схем (ИС) низкой и средней степени интеграции, а требуемая для нужного алгоритма структура формировалась путем выбора соответствующего набора ИС и связей между их выводами. Конструктивно такие устройства выполнялись на нескольких, связанных между собой, печатных платах, на которых устанавливались десятки ИС.

В 1965 году Гордон Мур сформулировал закон, в соответствии с которым полупроводниковым компаниям за счет совершенствования технологии удается удваивать число интегральных транзисторов на кристалле каждые полтора – два года. Справедливость этого закона подтверждается по настоящее время.

В начале 70-х годов прошлого века прогресс в области микроэлектроники уже позволял изготавливать устройства достаточно высокого уровня сложности всего на одном или нескольких кристаллах ИС большой степени интеграции (БИС). Серьезным сдерживающим фактором построения устройств с жесткой логикой на БИС был экономический. Затраты на подготовку производства новой модели БИС окупаются только при достаточно крупных объемах производства, а спрос на конечные изделия, в которых планировалась использовать эти БИС, не

покрывал этих затрат. Данная ситуация получила название – кризис микроэлектроники.

Массовый спрос на БИС этого сегмента можно было обеспечить только на основе программируемости. Микропроцессоры как раз и явились первым продуктом решения этой проблемы. За основу первого МП инженеры фирмы Intel взяли структуру процессора с классической архитектурой фон-Неймана. Процессор I4004 (1971 год) имел четырехразрядную сетку данных и нашел ограниченное применение. Зато появление восьмиразрядной модели I8080 (1974 год) и ее модернизированного варианта I8085 (1976 год) можно назвать технологической революцией, сопоставимой по влиянию с изобретением транзистора.

Программируемость МП, как и любого процессора, обеспечивается возможностью изменения выполняемой последовательности машинных операций. БИС МП при неизменной структуре можно использовать для решения широкого круга задач путем реализации требуемого алгоритма в виде последовательности выполняемых машинных инструкций. Модели I8080 и I8085 были очень популярны, кроме Intel их выпускали и другие полупроводниковые компании, в СССР производились их полные аналоги К580ИК80 (затем К580ВМ80А) и К1821ВМ85. Самым удачным клоном I8080 и его основным конкурентом стал МП Z80 компании Zilog.

Попытки построить на восьмиразрядном МП компьютер закончились лишь созданием «игрушек для взрослых», получивших название - бытовой микрокомпьютер. Миллионы энтузиастов в то время осваивали на этой платформе азы микропроцессорной техники. Бытовой микрокомпьютер «Хобби» на базе МП К580ВМ80 выпускался в восьмидесятые годы по линии конверсии оборонных отраслей промышленности на Оренбургском аппаратном заводе.

Первые модели персональных компьютеров профессионального уровня строили уже на шестнадцатиразрядных процессорах следующего поколения – это IBM PC на базе I8086 (1976 год) и его модификации I8088 и Apple Macintosh на базе Motorola 68000 (1979 год).

Основным профессиональным направлением применения восьмиразрядных МП оставалась замена жесткой логики. Тем не менее, конструкция программируе-

мых микропроцессорных контроллеров оставалась достаточно сложной. На печатной плате необходимо было разместить не только БИС МП, но также другие БИС микропроцессорного комплекта, включающего БИС контроллера прерываний, прямого доступа к памяти, последовательных и параллельных портов, таймеров, шинных формирователей и, конечно же, БИС памяти для хранения кода программы и данных. Такие изделия получили название одноплатные контроллеры.

Новым этапом в развитии элементной базы микропроцессорной техники явилась разработка Intel первых однокристальных микроконтроллеров (МК) семейства MCS-48 (1976 год). На одном кристалле удалось разместить полную микропроцессорную систему, включающую центральное процессорное устройство (ЦПУ), встроенный тактовый генератор, память программ и память данных, таймеры, программируемые порты. Для построения работоспособной микропроцессорной системы к БИС МК необходимо было подключить только несколько внешних элементов.

Следующей разработкой Intel стали МК семейства MCS-51 (1980 год). Архитектура и система команд MCS-51 оказались настолько удачными, что стали на долгое время стандартом. И хотя Intel давно перестала выпускать БИС этого семейства, десятки полупроводниковых компаний в настоящее время производят многочисленные клоны этих моделей уже на основе современных технологий.

На современном рынке средств автоматизации присутствуют одноплатные промышленные компьютеры, одноплатные контроллеры, программируемые логические контроллеры (ПЛК). Так как в отличие от персональных компьютеров их невозможно использовать изолированно от среды, в которой они должны работать, то какие микропроцессорные устройства (МПУ) получили название - встраиваемые системы (embedded systems). Областью применения встраиваемых систем также являются авионика, станки с числовым программным управлением, банкоматы, платежные терминалы, телекоммуникационное оборудование.

Специфика встраиваемых МПУ проявляется не только в их применении, но и разработке. В отличие от устройств на жесткой логике встраиваемые МПУ являются аппаратно-программными системами. Наличие аппаратных (hardware) и про-

граммных (software) компонент обусловливает ряд особенностей, характерных для процессов их проектирования. Структура и конфигурация аппаратного обеспечения МПУ, как правило, жестко детерминированы производителями элементной базы микропроцессорной техники. Специфику функций и решаемых задач МПУ, в основном, определяет программное обеспечение (ПО). Поэтому большая часть трудоемкости проектирования МПУ приходится на разработку встроенного ПО.

Разработка ПО для встраиваемых систем проходит в условиях, значительно отличающихся от условий, в которых разрабатывают ПО для компьютеров. В настоящее время разработчики встраиваемых приложений (эмбеддеры), также как и программисты персональных компьютеров (PC), в отличие от первопроходцев, не формирует исполняемые коды программ вручную, а используют для этого специализированные средства – интегрированные среды разработки (IDE – Integrated Development Environment). В обоих случаях эти среды разработки работают на платформе компьютера. Но основное отличие состоит в том, что для PC-программиста компьютер – это одновременно и инструментальная и целевая платформы. Другими словами, на одном и том же компьютере можно разрабатывать и исполнять программный код. Эмбеддеры, за редким исключением, лишены возможности разрабатывать ПО на целевой аппаратной платформе, так как эта платформа непригодна для работы с системой разработки:

- не обладает достаточными вычислительными ресурсами;

- не оборудована необходимыми средствами человеко-машинного интерфейса (клавиатура, дисплей);

- очень часто не имеет операционной системы.

Системы разработки, которые используют PC-программисты, например Microsoft Visual Studio или Delphi, называют резидентными, а системы, которые используют эмбеддеры, кросс - платформенными. Исключением являются случаи, когда МП целевой системы совместим по системе команд с процессором инструментального (обычно персонального) компьютера. Такими МП являются модели ряда I80x86, иногда используемые в одноплатных промышленных компьютерах.

Кросс - платформенные системы также используют разработчики приложений для мобильных устройств.

Специфика кросс – платформенных IDE заключается в использовании кросс - компиляторов и реализации средств отладки программ. Для исполнения кода программы используют либо программный симулятор целевого процессора, либо сам целевой процессор (или его заменяющий). Таким образом, отладчик кросс – платформенных IDE должен совместно работать либо с программным симулятором, либо с целевым процессором. Во втором случае требуются специальные технические средства для сопряжения инструментального компьютера и целевого процессора, находящегося в своей рабочей среде. Такая отладка программы называется внутрисхемной.

Для загрузки кода программы в память целевого процессора используются средства внутрисхемного программирования и загрузчики. Поэтому еще одной функцией кросс - платформенных IDE является управление средствами программирования.

В настоящее время наиболее распространенной элементной базой для построения встраиваемых систем являются МК. Также используются МП, системы на кристалле, системы в корпусе и системы в модуле. На современном рынке МК представлены множеством 8-разрядных, 16-разрядных и 32-разрядных моделей. Причем 32-разрядные МК в основном составляют конкуренцию 16-разрядным. Спрос на восьмиразрядные МК остается стабильно высоким. Их широкое применение в самых разнообразных приборах и устройствах бытового и промышленного применения можно объяснить оптимальным для этих задач сочетанием цена - качество.

Лабораторный практикум дисциплины построен на платформах восьмиразрядного МК PIC16F877A производства Microchip и 32-разрядного МК K1986BE92QI производства АО «ПКК Миландр».

Компания Microchip стабильно входит в пятерку лидеров производителей МК. Восьмиразрядные МК семейств PIC16 широко используются в системах электропривода, системах управления, интеллектуальных преобразователях электро-

энергии, автомобильной электронике, медицинской, измерительной и бытовой технике, технических средствах систем безопасности. Именно на моделях семейства PIC16F87x специалисты Microchip отрабатывали технологию встроенной теневой отладки. Только для этого семейства информация о реализации модуля встроенной теневой отладки была официально опубликована. Компания Microchip предлагает использовать разработчикам МПУ на платформах PIC16 – PIC18, PIC24, dsPIC, PIC32 свободно распространяемую интегрированную инструментальную систему MPLAB IDE. Основные инструменты этой системы рассматриваются в учебном пособии.

МК семейства K1986BE9x построены на лицензированном процессорном ядре ARM Cortex-M3. Компания «ПКК Миландр» предлагает использовать для разработки программ на платформе МК семейства K1986BE9x инструментальные средства сторонних компаний:

- CodeMaster-ARM - интегрированную среду разработки от ООО "КБ Фитон";

- пакет разработки RealView Microcontroller Development Kit (MDK ARM) компании Keil, состоящий из компилятора C/C++ ARM RealView и интегрированной среды разработки Keil µVision.

Для использования этих инструментальных средств требуется приобрести лицензию, но имеются бесплатные версии с ограниченными возможностями, которых вполне достаточно для изучения дисциплины. В лабораторном практикуме используется система Keil µVision v.5.

1 Задачи и средства проектирования МПУ

1.1 Маршрут проектирования МПУ

Исходные данные на проектирование МПУ представляются в форме технического задания (ТЗ), которое содержит основные требования к выполняемым функциям, значениям основных параметров и условиям эксплуатации. Также задаются требования к основным технико-экономическим показателям. На рисунке 1.1 приведен типовой маршрут проектирования МПУ.

Рисунок 1.1 – Маршрут проектирования МПУ

На этапе системно-алгоритмического проектирования МПУ из общей проблемы, часто поставленной абстрактно и независимо от технической реализации, формулируются четкие, конкретные технические требования к МПУ и выделяются

выполняемые функции на основе принятых для технического описания МПУ терминов и определений параметров, характеристик, режимов работы. Функциональная спецификация определяет набор функций, которые должно выполнять МПУ для удовлетворения требованиям ТЗ и обеспечения взаимодействия между МПУ и внешней средой (обслуживающим персоналом или пользователями, исполнительными устройствами, датчиками и т.д.). Последнее определяет наличие и количество индикаторных элементов, органов управления (переключателей, кнопок, клавиатуры), аналоговых и дискретных каналов ввода и вывода информации.

Далее разработчик должен определить конфигурацию аппаратной и программной компонент МПУ, а также распределить основные функции между ними. То есть, требуется решить, какие функции будут выполняться периферийными модулями, а какие функции будут выполняться процессорным ядром МК под управлением программы. Важно оптимально распределить выполняемые функции между аппаратными и программными средствами, тщательно учитывая особенности, достоинства и недостатки реализации функций каждой компонентой МПУ.

К преимуществам программной реализации можно отнести:

- широкие функциональные возможности;

- возможность оперативной перенастройки МПУ на новые условия, задачи, объекты путем изменения только ПО.

Вследствие этого МПУ возможно будут иметь следующие достоинства:

- незначительные затраты на развитие и модернизацию;

- более низкие показатели по энергопотреблению, стоимости и, возможно, по габаритам и массе;

- высокую технологичность и малые затраты ресурсов на производство;

- расширенные возможности для унификации ряда близких по назначению МПУ.

В то же время программная реализация функций имеет по сравнению с аппаратной ряд особенностей, которые могут оказывать существенное влияние на распределение функций. К их числу можно отнести:

- время выполнения функций может оказаться неприемлемо большим для задач реального времени;

- сложность программной реализации функций сопряжения с объектами;

- ограниченные ресурсы памяти для хранения кода программы и данных.

По результатам распределения функций окончательно формулируются требования к параметрам используемого МК. При выборе модели МК учитываются следующие основные характеристики:

- разрядность;

- особенности архитектуры и производительность;

- набор команд и способов адресации;

- требования к источнику питания и потребляемая мощность в различных режимах;

- объем памяти программ и данных и возможность их расширения;

- наличие и возможности периферийных модулей и портов;

- возможность внутрисхемного программирования;

- возможность поставки в различных корпусах;

- стоимость в различных вариантах исполнения;

- наличие полной документации;

- наличие и доступность эффективных средств разработки и отладки программ;

- количество и доступность каналов поставки, возможность замены изделиями других фирм.

Номенклатура выпускаемых моделей МК исчисляется тысячами типов изделий различных фирм. Современная стратегия модульного проектирования обеспечивает потребителя разнообразием моделей МК с одним и тем же процессорным ядром. Такое разнообразие открывает перед разработчиком возможность выбора оптимального МК, не имеющего чрезмерной функциональной избыточности, что минимизирует стоимость комплектующих элементов. Допущенные на данном этапе просчеты могут впоследствии привести к необходимости смены модели МК.

Дальнейшая разработка аппаратных и программных средств выполняется параллельно. На этапе разработки структуры МПУ окончательно определяется состав встроенных в МК и внешних периферийных модулей, протоколы обмена между модулями. Выполняется проработка электрических схем и конструкции МПУ. Часто бывает целесообразно на основании полученных результатов изготавливать не опытный образец МПУ, а его прототип в упрощенном конструктивном исполнении. Тестирование опытного образца или прототипа направлено на выявление и исправление ошибок, допущенных при разработке схем и конструкций.

На этапе разработки программных средств определяются состав и связи программных модулей, язык программирования. На этом же этапе осуществляется выбор средств разработки и отладки программ.

Разработка встроенного ПО для МПУ, аналогично разработке прикладных программ компьютеров, включает следующие этапы:

- анализ выполняемых функций и требований к ПО;

- формализация задач и функций, включая определение входов и выходов алгоритмических и программных модулей, конкретных процессов обработки, формулирование системных ограничений (эксплуатационных, точности обработки данных, времени формирования результата, объема памяти данных и программ);

- алгоритмизация (построение алгоритма, удовлетворяющего требованиям постановки задачи и спецификаций);

- кодирование или собственно программирование, заключающееся в формировании исходного текста программы на выбранном языке программирования.

В результате трансляции и компоновки программы получают загрузочный модуль в машинных кодах целевого МК. Автономная отладка программы направлена на поиск и исправление допущенных ошибок. Трансляторы позволяют выявить и локализовать только формальные ошибки, связанные с нарушениями правил грамматики выбранного языка программирования. Основным методом поиска логических и других неформальных ошибок является тестирование программы. При выполнении автономной отладки тестирование проводят на программно-

логической модели МК. Полнота тестирования зависит от правильного планирования и выбора тестовых данных.

Возможности программно-логических моделей МК ограничены. Например, они не позволяют достоверно учитывать асинхронность изменений сигналов на внешних выводах МК, нестабильность частоты тактового генератора и рассогласование фронтов синхросигналов. Поэтому для полной и эффективной проверки МПУ требуется выполнять комплексную отладку после загрузки машинного кода программы в память программ МПУ.

Комплексная отладка является наиболее сложным этапом разработки МПУ и занимает особое место в процессе проектирования, так как выполняется при непосредственном взаимодействии программных и аппаратных средств. Исправление найденной ошибки в программе требует выполнения повторной трансляции и перепрограммирования памяти программ. Необходимость экспериментальной проверки взаимодействия программы с модулями ввода-вывода и внешними устройствами в реальном времени требуют применения специальных средств комплексной отладки. Окончательную доводку МПУ выполняют при тестировании в составе изделия. По оценкам [2] все виды отладки МПУ часто занимают более 50 % полного времени проектирования МПУ.

1.2 Средства разработки МПУ

Предметом дисциплины «Отладочные средства микропроцессорных систем» являются инструментальные средства, используемые для разработки МПУ, а также задачи разработки, при решении которых эти средства активно применяются. В последующих разделах учебного пособия основное внимание уделено средствам разработки программного обеспечения МПУ, которые используются для решения задач:

- кодирования алгоритмов;
- тестирования и автономной отладки программ;
- загрузки программ в память МПУ;
- комплексной отладки программ.

Для разработки аппаратных средств МПУ обычно используют универсальные средства проектирования электронных устройств – системы автоматизированного проектирования (САПР), позволяющие выполнять разработку и верификацию электрических схем и конструкций. Для автономного тестирования прототипов и опытных образцов МПУ обычно используют универсальные измерительные приборы:

- мультиметры;
- осциллографы;
- логические пробники;
- логические анализаторы.

Примеры применения этих средств подробно изложены в книгах [1, 2].

Разработка и изготовление макета или прототипа МПУ требуют существенных материальных затрат. Также требуется потратить время. Себестоимость технологической подготовки производства печатных плат практически не зависит от объемов производства. То есть, стоимость изготовления по заказу одной – двух плат будет несущественно отличаться от стоимости мелкой серии. А себестоимость одной платы в серии будет многократно ниже. Такая же ситуация со сборкой узлов на основе печатных плат. При серийном производстве сборку выполняют технологические автоматы, а единичные экземпляры собирают вручную.

По этой причине выгоднее использовать в качестве прототипа отладочные платы (evaluation board, evaluation kit) , выпускаемые серийно. Они предназначены для обеспечения возможности оценки функционирования и отладки МПУ на базе определённых моделей МК. Это позволяет существенно ускорить начало выполнения этапа комплексной отладки.

В качестве примера на рисунке 1.2 приведен внешний вид контроллера-конструктора KIT-PIC03 производства компании "КТЦ-МК". Контроллер-конструктор KIT-PIC03 предназначен для макетирования устройств, проектируемых на базе МК семейств PIC16, и представляет собой печатную плату, на которой размещены:

- панельки DIP28 и DIP18 для установки МК семейств PIC16 и кварцевый резонатор;

- модуль PC1602AR-IEH-B жидко - кристаллического индикатора (ЖКИ);

- 16-ти кнопочная клавиатура со схемой управления;

- интегральные схемы EEPROM AT24C64 и часов реального времени PCF8583;

- преобразователь уровней интерфейса RS-232 (ADM202) и разъем DB-9 для интерфейса RS-232;

- разъем питания, разъем RJ-12 для подключения внутрисхемного отладчика – программатора.

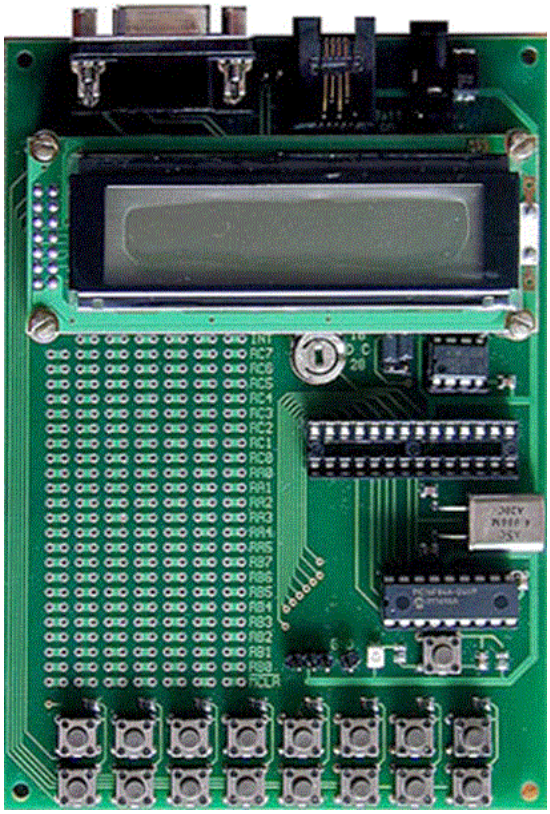


Рисунок 1.2 – Контроллер-конструктор KIT-PIC03

Как правило, отладочные платы содержат макетное поле с подведенными к нему выводами портов МК. На контактные площадки макетного поля можно дополнительно установить необходимые для разрабатываемого МПУ элементы. Для выполнения электрических соединений используют проводной монтаж.

Для изучения микропроцессорной техники удобнее использовать отладочные платы, содержащие необходимые средства для программирования МК и выполнения комплексной отладки программ. На рисунке 1.3 приведен внешний вид отладочного комплекса, используемого в лабораторном практикуме. На основной отладочной плате установлен МК PIC16F877A, кварцевый резонатор, преобразователь уровней и разъем DB-9 для интерфейса RS-232, разъем для подключения источника питания. На этой же плате размещены элементы внутрисхемного отладчика – программатора, являющегося функциональным аналогом устройства PICKit-2. Для подключения к инструментальному компьютеру используется скоростной интерфейс USB. Выводы портов МК подключены к разъемам, через которые можно подключать периферийные платы.

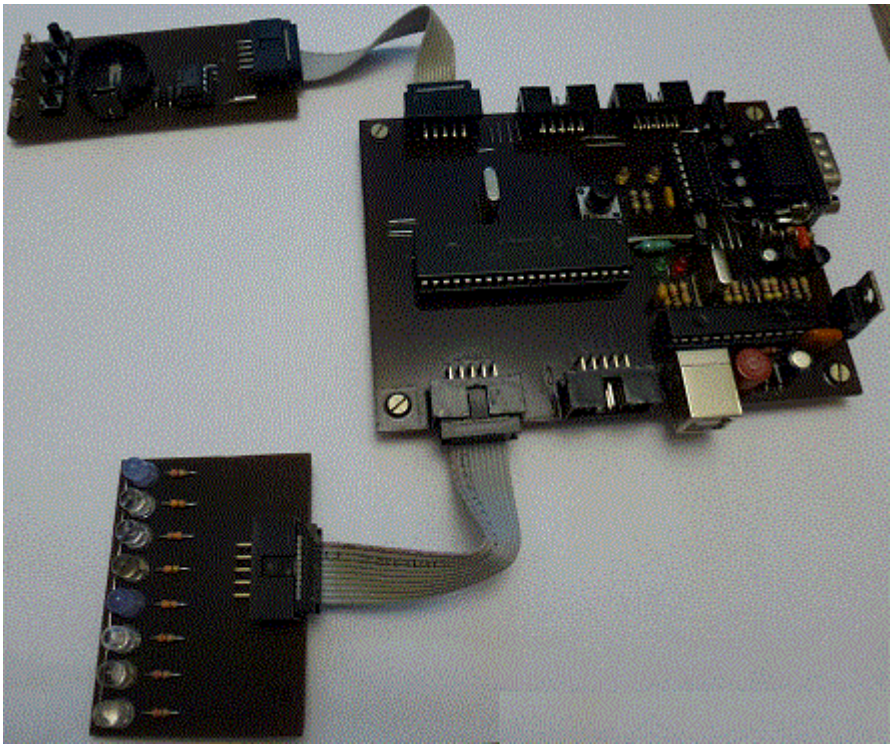


Рисунок 1.3 – Отладочный стенд на PIC16F877A

В лабораторном практикуме дисциплины также используются отладочные платы LDM-K1986BE92QI-H. На плате установлен МК фирмы ЗАО «ПКК Миландр» с ядром ARM Cortex-M3. Отладочная плата LDM-K1986BE92QI предназначена для обучения основам проектирования электронных устройств на основе МК

K1986BE92QI, отладки проектов, сокращения времени выхода новых продуктов на рынок [3].

Отладочная плата в полной комплектации содержит драйверы внешней периферии: RS232, CAN, RS485, разъем USB Host, панель для литиевой батареи, генератор часов реального времени, ЖКИ индикатор, отладочные разъемы JTAG A и JTAG B. Общий вид полной комплектации отладочной платы LDM-K1986BE92QI приведен на рисунке 1.4. На плате также установлены 5 кнопок управления и 4 пользовательских светодиода, 3 движковых переключателя режима программирования FLASH, кнопки RESET и WAKEUP, разъем USB-UART преобразователя (UART-загрузчик), разъем интерфейса USB Host, преобразователи напряжения питания. Все выводы микроконтроллера выведены на 2 разъема XS9, XS10. Питание платы осуществляется от внешнего стабилизированного источника или от USB порта персонального компьютера.

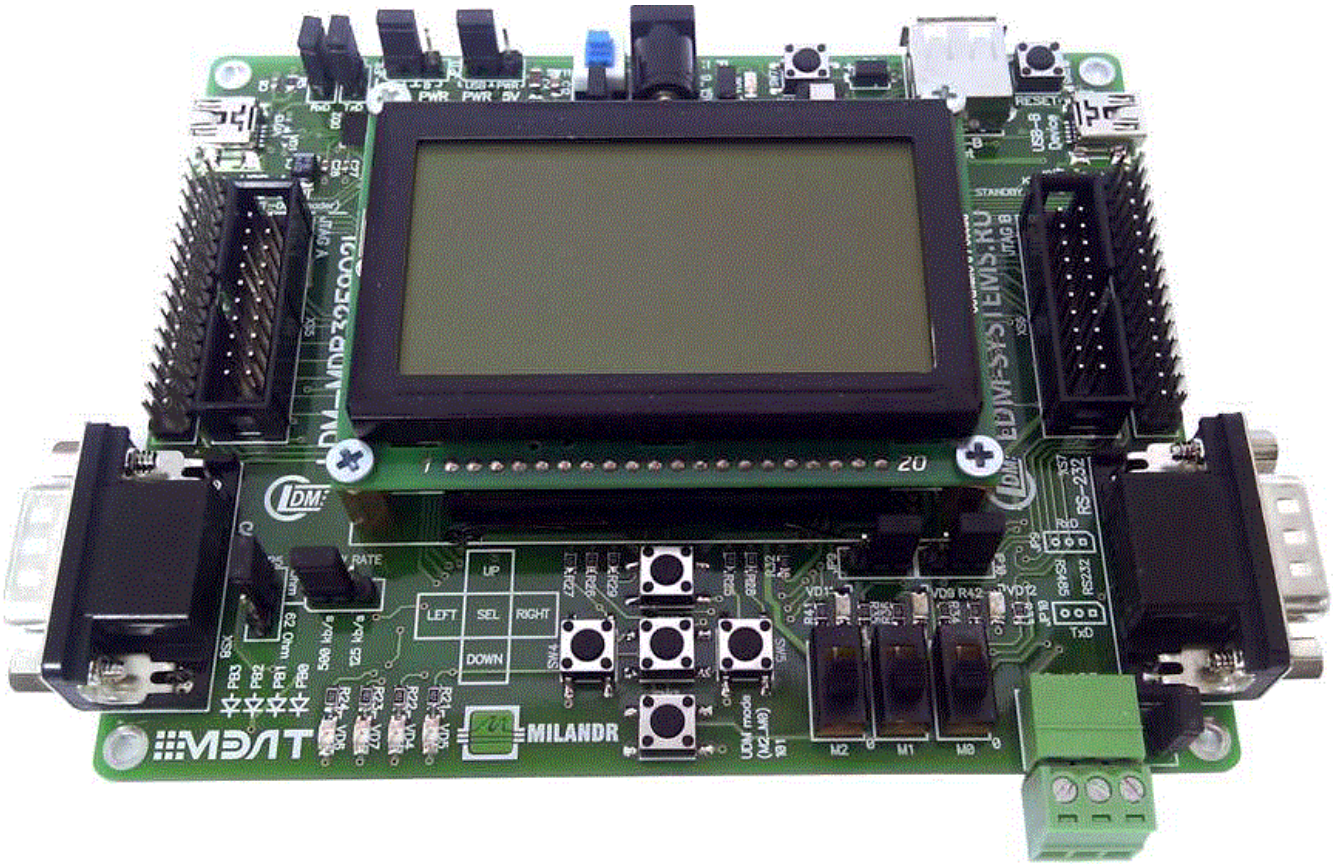


Рисунок 1.4 – Отладочный стенд - LDM-K1986BE92QI-H

1.3 Контрольные вопросы к разделу 1

Что понимают под встраиваемыми системами?

Перечислите основные особенности разработки электронных устройств на основе микроконтроллеров.

Перечислите основные этапы проектирования МПУ.

Поясните содержание основных этапов проектирования МПУ.

Поясните причины, по которым требуется выполнять этапы автономной и комплексной отладки программного обеспечения МПУ.

Чем отличаются этапы автономной и комплексной отладки программного обеспечения МПУ?

Чем отличаются резидентные и кросс - платформенные средства разработки программного обеспечения?

Что понимают под прототипом МПУ?

Какие преимущества дает применение отладочных плат?

2 Средства кодирования и трансляции программ

2.1 Языки программирования

Средства, используемые для формирования загрузочных кодов программ МПУ, прошли примерно такой же путь развития, как и аналогичные средства для компьютеров. Но ручное кодирование программ использовалось очень недолго, так как к моменту появления первых МП для компьютеров уже были разработаны эффективные языки программирования и средства трансляции. Была разработана теория формальных языков и грамматик, на основе которой разрабатывались эти средства. Поэтому достаточно быстро появились аналогичные средства в кросс - платформенном варианте реализации [4].

Основным средством кодирования алгоритмов являются алгоритмические языки или языки программирования. Эти языки относятся к формальным языкам, которые в отличие от естественных языков имеют существенно более простые и строгие грамматические правила. Это позволяет относительно легко построить средства для автоматического перевода исходных текстов в машинный код программы процессора. При этом формальный язык определятся двумя способами:

- порождающей грамматикой;
- распознавателем.

Порождающая грамматика определяет лексику и синтаксис языка и ориентирована на программистов. Программисты должны знать грамматику выбранного языка и формировать исходные тексты в соответствии с правилами этой грамматики. *Распознавателем* называют алгоритм, позволяющий выполнить проверку корректности исходного текста на соответствие правилам грамматики и построение последовательности операций процессора, эквивалентных операторам языка. Кодирование полученной последовательности позволяет сформировать загрузочный код программы. Этот алгоритм лежит в основе работы трансляторов.

Такой способ построения языков и трансляторов позволяет исключить получение любого сомнительного результата трансляции в виде кода программы по причине грамматических ошибок в исходном тексте. Трансляторы работают таким

образом, что загрузочный код может быть сформирован только после исправления всех грамматических ошибок.

Языки программирования делятся на две основные группы:

- языки программирования высокого уровня
- языки программирования низкого уровня.

Языки программирования низкого уровня иначе называют языками ассемблера. Основой любого языка ассемблера является мнемоническое представление машинных инструкций МП или МК. Очевидно, что для процессоров с различными системами команд необходимо использовать различные языки ассемблера. Ассемблеры для одного и того же процессора могут различаться между собой дополнительными возможностями.

Языки программирования высокого уровня позволяют при трансляции заменять один оператор несколькими машинными командами. Это позволяет повысить производительность труда программистов. Кроме того, использование языков высокого уровня позволяет с меньшими затратами переносить программы с одной аппаратной платформы на другую. Очевидно, что при этом необходимо использовать трансляторы для соответствующего процессора.

Загрузочный код программы, полученной в результате трансляции с языка ассемблера, обладает наилучшими показателями эффективности, которая оценивается затратами памяти для хранения кода и временем выполнения. При кодировании того же алгоритма на языке высокого уровня эти показатели будут уступать. Повысить эффективность можно, используя оптимизирующие трансляторы. Но и в этом случае не удается получить показатели эффективности, как при использовании языка ассемблера.

Среди языков высокого уровня наилучшие показатели эффективности удается получить при использовании языка Си. Особенные свойства этого языка объясняются тем, что его специально разрабатывали для написания операционной системы Unix. По этой причине в подавляющем большинстве случаев классический вариант языка ANSI C и языки C++, C# используются для разработки программ для встраиваемых МПУ. Альтернативой является использование языков ассемблера.

Основу формальных грамматик составляют правила, последовательное применение которых позволяет строить синтаксически и лексически правильные конструкции языка. Эти правила можно сформулировать в различных формах:

- средствами аппарата математической лингвистики;

- формальными системами, ориентированными на профессиональных прикладных программистов;

- текстовом описании в произвольной форме.

В математической лингвистике формальной порождающей грамматикой ***G*** называется четверка

$$G = (V_T, V_A, S \in V_A, R),$$

где V_T – *терминальный,* основной алфавит **(**символы этого алфавита называются *терминалами,* из них строятся цепочки**,** *порождаемые грамматикой)*;

V_A - *нетерминальный,* вспомогательный алфавит**,** символы этого алфавита называются *нетерминалами* и используются при построении цепочек; они могут входить в промежуточные цепочки, но не должны входить в результат порождения;

S - *начальный нетерминал* грамматики;

R - множество *правил вывода* или *порождающих правил*.

Нетерминалы используются для обозначения определяемых конструкций языка, а терминалы обозначают элементы языка, из которых эти конструкции можно построить.

Для формального описания порождающих правил в теории формальных грамматик используется *метаязык*, с помощью которого можно задавать описание формальных языков. Для описания общих свойств грамматик обычно используют символическую форму задания правил, предложенную Н. Хомским. Она предусматривает использование в качестве элементов нетерминального словаря отдельных символов. В символической форме правила грамматик имеют вид $\alpha \to \beta$, где α и β - цепочки, построенные из букв алфавита $V_T \cup V_A$, который называют полным алфавитом грамматики ***G***. Знак ‘$\to$’ отделяет левую часть пра-

вила от правой части и читается “порождает” или “есть по определению” (α порождает β).

Рассмотрим пример определения правил записи целых чисел. Отрицательные целые числа записывают со знаком ’ –‘, а положительные без знака или со знаком ‘+’. Любую цифру от ‘0’ до ‘9’обозначим терминалом d, тогда терминальный алфавит грамматики будет включать три символа $V_T = \{d, ' -', '+'\}$. Определяемому понятию – “целое” поставим в соответствие начальный нетерминал S. Понятие “целое без знака” обозначим нетерминалом A, а цепочку, состоящую из цифр, нетерминалом B. Алфавит нетерминалов $V_A = \{S, A, B\}$.

Так как целое число может быть записано и одной цифрой, то правила грамматики будут иметь вид

$S \to '+'A | '-'A | dB$,

$A \to dB$,

$B \to dB | \varepsilon$.

Символ ’|’ читается как ИЛИ, символ ‘ε’ обозначает пустую цепочку. Алгоритм распознавания в форме графа конечного автомата приведен на рисунке 2.1.

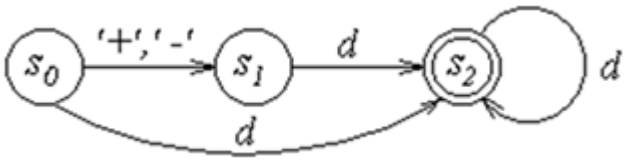


Рисунок 2.1 – Граф распознавателя

Состояния автомата s_0, s_1, s_2 поставлены в соответствие нетерминалам грамматики S, A, B. Автомат начинает анализ цепочки символов в состоянии s_0. Только при анализе цепочки, состоящей из цифр и возможно, начинающейся с символов ‘+’ или ’ –‘, автомат завершит работу в выделенном состоянии s_2 .

Для практикующих программистов разработчики формальных языков часто предлагают использовать альтернативные формы представления правил грамматики. Например, в предложенной Бэкусом и Науром форме (БНФ) была задана грам-

матика языка Алгол-60. Н. Вирт грамматику языка Паскаль представил в форме синтаксических диаграмм.

В БНФ вместо нетерминалов используются названия определяемых конструкций, заключенные в угловые скобки. В качестве разделителя используют символ ‘::=’.

Пример определения лексических правил записи целого числа в БНФ:

<целое> ::= <целое без знака> | + <целое без знака> | - <целое без знака>

<целое без знака> ::= <цифра> | <целое без знака> <цифра>

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

2.2 Язык ассемблера PIC16

Синтаксически программа на ассемблере состоит из последовательности строк. Строка – основная единица ассемблерной программы. Синтаксис строки

[<метка>] <мнемоника> [<операнд>{,<операнд>}] [;<комментарий>]

Необязательные элементы строки указаны в квадратных скобках, повторяющиеся элементы – в фигурных скобках. Метка должна начинаться в колонке номер один. Мнемоника может начинаться в колонке два или далее. Комментарий отделяется символом ‘;’. Максимальная длина строки 255 символов. Один или несколько пробелов должны отделять метку и мнемонику или мнемонику и операнды. Операнды разделяются запятой.

Главным и обязательным элементом строки является мнемоника, под которой понимается машинная инструкция, директива ассемблера или макрокоманда

<мнемоника>::=<инструкция>|<директива>|<макрокоманда>

Инструкции являются машинными командами, представленными в мнемонической форме

<инструкция>::=ADDWF|ANDWF| ... |SUBLW|XORLW

Полный перечень инструкций МК семейства PIC16 и подробное описание можно найти в [5]. Необязательная метка в строке инструкции используется как символьное представление адреса ячейки памяти, в которой располагается ее код. Метка

может начинаться с буквы или нижнего подчеркивания ‘_’ и содержать буквы и цифры. Длина метки может быть до 32 символов.

Вместо констант в качестве параметров допускается использовать выражения. Выражения могут содержать константы, символы или любые комбинации констант и символов, разделенных арифметическими операторами. В ассемблере MPASM используются следующие форматы выражений:

- текстовая строка;

- числовые константы и Radix;

- арифметические операторы и приоритеты;

- High / Low операторы.

Допустимые форматы представления констант приведены в таблице 2.1

Таблица 2.1 – Форматы констант

Тип	Синтаксис	Пример
Десятичная	D'<цифры>' или .<цифры>	D'100' или .100
16-ричная	H'<цифры>' или 0x<цифры>	H'9f' или 0x9f
Восьмеричная	O'<цифры>'	O'777'
Двоичная	B'<цифры>'	B'00111001'
Символьная	'<символ>' или A'<символ>'	"C" или A'C'

Поле комментария может использоваться программистом для текстового или символьного пояснения логической организации программы. Поле комментария не обрабатывается ассемблером, поэтому в нем можно применять любые символы.

Директивы ассемблера служат для управления процессом трансляции программы

<директива>::=BANKISEL|BANKSEL| ... |VARIABLE|WHILE

Существует пять основных типов директив в языке ассемблера PIC16:

- директивы контроля – управляют созданием разделов условно компилированного кода;

- директивы данных – управляют распределением памяти и назначением символических имен переменным и константам;

- директивы листинга – определяют формат и состав файла листинга, позволяют указывать заголовки, нумеровать страницы и настраивать другие параметры;

- макро директивы – управляют работой макросов и распределением данных в теле макроса;

- директивы объектного файла – используются только при создании перемещаемого объектного кода.

Полный перечень директив и их полное подробное описание приведены в документе [6].

Директива MACRO используется для определения макроса

<метка> macro [<аргумент>{,<аргумент>}]

Метка в строке директивы MACRO определяет имя макроса, а аргументами являются имена передаваемых в макрос параметров. В тело макроса могут быть включены любые инструкции и директивы. Ассемблер обрабатывает инструкции и директивы тела макроса до тех пор, пока не встретит директиву EXITM или ENDM.

Например, можно определить макросы псевдо инструкций условных переходов

```
jz      macro       address
        btfsc       STATUS,Z
        goto        address
        endm
 jnz    macro       address
        btfss       STATUS,Z
        goto        address
        endm
```

Однажды определенный макрос может вызываться макрокомандой в любой части программы в пределах исходного модуля. Синтаксис макрокоманды вызова макроса:

<имя макроса> [<значение аргумента>{,<значения аргумента>}]

При вызове макроса, аргументы в теле макроса будут заменены их значениями везде, где встречается имя данного аргумента. Например, вызов макроса условного переход на метку m1

```
        jz      m1
```

Рекомендуемые структуры исходного текста программ на языке Ассемблера для МК PIC16 приводятся в шаблонах системы MPLAB, которые находятся в директории ..\MPASM Suite\Template.

Варианты шаблонов для получения абсолютного кода программы находятся в папке ..\Code. Для МК типа PIC16F877A шаблон находится в файле 16F877ATEMP.asm. Данный шаблон с комментариями на русском языке приведен ниже:

```
list    p=16f877,st = OFF       ; директива листинга
#include <p16f877.inc>          ; определение переменных МК

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC & _WRT_ENABLE_ON & _LVP_OFF & _DEBUG_ON & _CPD_OFF
; директива установки разрядов слова конфигурации
;***** Определение переменных
w_temp      EQU     0x70        ; переменные для сохранения контекста
status_temp EQU     0x71        ;
; директивы определения остальных переменных
;*****************************************************************
            ORG     0x000           ; вектор старта
            clrf    PCLATH          ; очистка программного счетчика
            goto    main            ; переход на начало программы

            ORG     0x004           ; вектор прерывания
            movwf   w_temp          ; сохранение контекста
            movf    STATUS,w        ;
            movwf   status_temp     ;
; операторы подпрограммы обработки прерывания
            movf    status_temp,w   ; восстановление контекста
            movwf   STATUS          ;
            swapf   w_temp,f
            swapf   w_temp,w        ;
            retfie                  ; возврат из подпрограммы

main
; операторы программы
            END                     ; директива окончания программы
```

В общем случае текст программы на языке ассемблера PIC16 состоит из следующих разделов:

- директивы листинга list, директивы #include включения файла с определением переменных конкретной модели МК;

- директивы _CONFIG формирования слова конфигурации;

- директив определения переменных программы EQU или CBLOCK;

- директивы указания вектора старта ORG;

- оператора переход на начало программы;

- директивы указания вектора прерывания ORG;

- операторов подпрограммы обработки прерывания;

- операторов программы;

- директивы окончания программы END.

Если в программе не используется механизм прерываний, то из шаблона можно удалить раздел подпрограммы обработки прерывания и команду перехода на операторы основной программы (goto main).

2.3 Набор инструментов MPASM

Основным инструментальным средством для разработки встроенного программного кода для всех МК производства Microchip является интегрированная среда MPLAB IDE [7]. Для МК семейств PIC16 – PIC18 среда MPLAB IDE включает набор инструментов MPASM Toolsuite [6], состоящий из:

- ассемблера MPASM;

- компоновщика (редактора связей) MPLINK;

- библиотекаря MPLIB.

Ассемблер MPASM выполняет лексический и синтаксический анализ исходных текстов программы, а также выполняет построение объектного кода программы. Исходные тексты программ, содержащих подпрограммы, могут быть сохранены полностью в одном файле или в нескольких файлах. Например, в одном файле сохраняется основная программа, а в другом подпрограммы.

В первом случае MPASM может сформировать абсолютный объектный код, в котором все ячейки памяти программ и данных будут иметь абсолютные физиче-

ские адреса (рисунок 2.2). Во втором случае для каждого исходного модуля MPASM сформирует перемещаемый объектный код, в котором все ячейки памяти программ и данных будут иметь относительные адреса (рисунок 2.3).

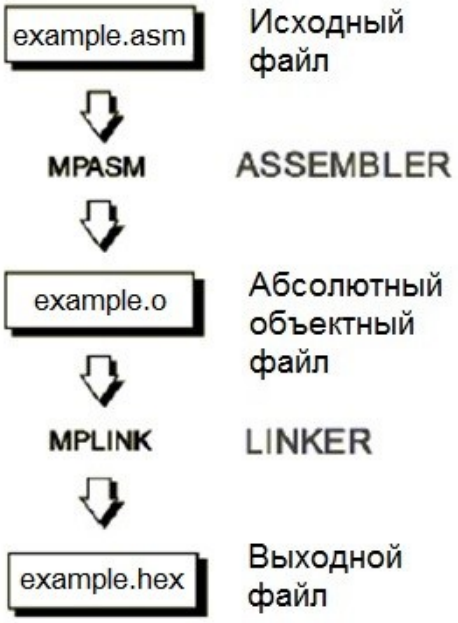


Рисунок 2.2 – Трансляция в абсолютный объектный код

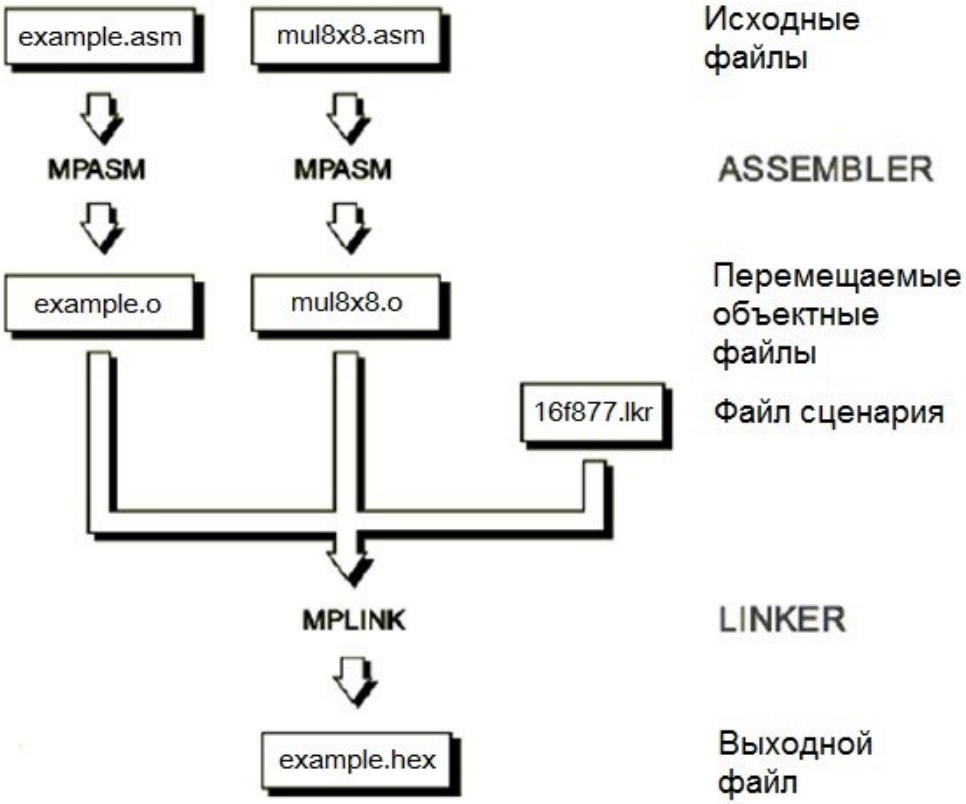


Рисунок 2.3 – Трансляция в перемещаемый объектный код

Компоновщик MPLINK формирует загрузочный модуль программы в Intel HEX формате. При работе с перемещаемыми объектными модулями компоновщик формирует абсолютные адреса, при этом он использует файл сценария, в котором описано распределение основных сегментов памяти программ и памяти данных для конкретной модели МК. Компоновщик решает конфликты адресов и обеспечивает, что программа или данные не будут размещаться в пространстве адресов, которое уже занято. Компоновщик MPLINK также формирует файл с символьной информацией для отладчика MPLAB IDE.

Исходные тексты программы должны быть сохранены в файлах с расширением .ASM. Расширения файлов, используемых и формируемых программами MPASM Toolsuite, приведены в таблице 2.2.

Таблица 2.2 - Расширения файлов MPASM Toolsuite

Расширение	Назначение
.ASM	Входной файл ассемблера для MPASM <source_name>.ASM
.OBJ	Выходной файл объектного кода из MPASM <source_name>.OBJ
.LST	Выходной файл листинга, генерируемый ассемблером MPASM: <source_name>.LST
.ERR	Выходной файл ошибок из MPASM: <source_name>.ERR
.MAP	Выходной файл распределения памяти из MPLINK: <source_name>.MAP
.HEX	Выходной файл загрузочного кода в HEX формате из MPLINK: <source_name>.HEX
.LIB	Библиотечный файл, созданный MPLIB: <source_name>.LIB
.LNK	Выходной файл компоновщика: <source_name>.LNK
.COF	Выходной файл для отладчика. Формируются MPLINK: <source_name>.COF

Листинг представляет собой текстовый файл в формате ASCII, который содержит отчет ассемблера о результате трансляции. Если текст программы не содержит лексических и синтаксических ошибок, то файл листинга содержит абсолютный или перемещаемый объектный код в шестнадцатеричном представлении.

Компоновщик MPLINK формирует карту использования памяти программ и памяти данных. В таблице символов перечисляются все символы, которые есть в программе, и где они определены. Карта использования памяти дает представление о расходовании памяти МК.

Для записи кода программы в энергонезависимую память МК используются программаторы, поэтому загрузочный код программы представляют в стандартном для программаторов формате INHX8M (Intel HEX формат). В этом формате 14-разрядные команды представляются побайтно. Так как по каждому адресу содержится только один байт, объем адресов увеличен вдвое. Каждая строка начинается с 9 знаковой приставки и заканчивается 2-х знаковой контрольной суммой.

:BBAAAATTHHHH....HHHCC

где: BB - количество байт в строке;

AAAA - адрес записи данных;

TT - указатель конца файла (00 – данные, 01 – конец файла);

HH - байт данных или младший/старший байт слова команды;

CC - контрольная сумма, которая является дополнением ко всем предшествуемым байтам в строке.

В качестве примера рассмотрим результаты трансляции программы, циклически выполняющей умножение однобайтных чисел, последовательно вводимых через порт ввода PORTB. Программа состоит из основного модуля main и подпрограммы умножения mpy. В первом варианте программы оба модуля находились в одном файле LR1.asm. Текст программы построен по шаблону 16F877ATEMP.asm. Идентификаторы используемых переменных определены директивой CBLOCK, начало сегмента кода (вектор старта) задано директивой ORG.

Листинг ассемблера

```
MPASM  5.30          LR1.ASM   10-21-2018  21:59:56         PAGE  1
LOC   CODE    LINE          SOURCE TEXT
  VALUE
              00001         LIST    p=16F877a, st=OFF
              00002         #include "P16F877A.INC"
              00003         CBLOCK 0x20    ; Структура данных
  00000020    00004         Mulcnd         ; множимое
  00000021    00005         Mulplr         ; множитель
  00000022    00006         H_byte         ; старший байт произведения
```

```
  00000023    00007         L_byte          ; младший байт произведения
  00000024    00008         Count           ; счетчик циклов
              00009         ENDC
              00010
0000          00011         ORG     0x000   ;вектор старта
0000   0806   00012 main    movf    PORTB,w
0001   00A1   00013         movwf   Mulplr
0002   0806   00014         movf    PORTB,w
0003   00A0   00015         movwf   Mulcnd
0004   2006   00016         call    mpy     ;вызов подпрограммы
0005   2800   00017         goto    main
              00018 ;************* Подпрограмма умножения ************
0006   01A2   00019 mpy     clrf    H_byte
0007   01A3   00020         clrf    L_byte
0008   3008   00021         movlw   8
0009   00A4   00022         movwf   Count
000A   0820   00023         movf    Mulcnd,w
000B   1003   00024         bcf     STATUS,C
000C   0CA1   00025 Loop    rrf     Mulplr,f
000D   1803   00026         btfsc   STATUS,C
000E   07A2   00027         addwf   H_byte,f
000F   0CA2   00028         rrf     H_byte,f
0010   0CA3   00029         rrf     L_byte,f
0011   0BA4   00030         decfsz  Count,f
0012   280C   00031         goto    Loop
0013   3400   00032         retlw   0
              00033 ;***********************************************
              00034         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)
0000 : XXXXXXXXXXXXXXXX XXXX------------ ---------------- ----------------
All other memory blocks unused.
Program Memory Words Used:    20
Program Memory Words Free:  8172
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

В результате трансляции по схеме на рисунке 2.2 MPASM получил абсолютный объектный код программы (столбец CODE), для размещения которого было использовано 20 ячеек памяти программ с адресами в диапазоне 0x0000 – 0x0013. Для размещения данных были использованы ячейки с адресами в диапазоне 0x020 – 0x024.

Компоновщик MPLINK сформировал загрузочный модуль программы в Intel HEX формате:

```
:020000040000FA
:100000000608A1000608A00006200028A201A301FE
:100010000830A40020080310A10C0318A207A20CAA
:08002000A30CA40B0C28003412
:00000001FF
```

Карта памяти, построенная компоновщиком:

```
     MPLINK 4.30, Linker
     Linker Map File - Created Sun Oct 21 21:40:47 2018
     Section Info
Section      Type      Address   Location   Size (Bytes)
---------  --------  ---------  ---------  ---------
.org_0       code     0x000000    program   0x000000
.org_1       code     0x000000    program   0x000028

     Program Memory Usage
Start         End
---------   ---------
0x000000    0x000013
20 out of 8456 program addresses used, program memory utilization is 0%

     Symbols - Sorted by Address
Name    Address   Location      Storage File
----- ---------  ---------     --------- ---------
main  0x000000    program     static C:\PIC\Lr\lr1.asm
mpy   0x000006    program     static C:\PIC\Lr\lr1.asm
loop  0x00000c    program     static C:\PIC\Lr\lr1.asm
```

Во втором варианте модули программы были сохранены в двух файлах:

- example.asm – модуль основной программы;

- mul8x8.asm – подпрограмма умножения.

Текст программы построен по шаблону 16F877ATEMPO.asm. Идентификаторы используемых переменных определены директивой UDATA, внешние и глобальные метки и идентификаторы переменных определены директивами EXTERN и GLOBAL, начало сегмента кода (вектор старта) задано директивой CODE.

Листинг ассемблера, полученный после трансляции модуля основной программы.

```
MPASM  5.30          EXAMPLE.ASM   11-28-2018  18:50:20         PAGE  1
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE
                     00001         LIST    p=16F877a, st = OFF
                     00002         #include "P16F877A.INC"
                     00003
  0000               00004         EXTERN  Mulcnd, Mulplr, H_byte, L_byte
  0000               00005         EXTERN  mpy
                     00006
                     00007 PROG1   CODE                    ;сегмент PROG1
0000   0806          00008 main    movf    PORTB,w
0001   00??          00009         movwf   Mulplr
0002   0806          00010         movf    PORTB,w
0003   00??          00011         movwf   Mulcnd
0004   2???          00012         call    mpy             ;вызов подпрограммы
0005   2???          00013         goto    main
                     00014         END
```

```
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

В результате трансляции по схеме на рисунке 2.3 MPASM получил перемещаемый объектный код модуля (столбец CODE), для размещения которого было использовано 6 ячеек памяти программ с относительными адресами в диапазоне 0x0000 – 0x0005. Листинг ассемблера, полученный после трансляции подпрограммы умножения:

```
MPASM  5.30          MUL8X8.ASM   10-21-2018  22:02:40         PAGE  1
LOC   CODE     LINE        SOURCE TEXT
  VALUE
                00001         LIST    p=16F877a, st = OFF
                00002         #include "P16F877A.INC"
                00003         UDATA          ; Структура данных
0000            00004 Mulcnd  RES     1      ; множимое
0001            00005 Mulplr  RES     1      ; множитель
0002            00006 H_byte  RES     1      ; старший байт произведения
0003            00007 L_byte  RES     1      ; младший байт произведения
0004            00008 Count   RES     1      ; счетчик циклов
                00009
  0000          00010         GLOBAL  Mulcnd, Mulplr, H_byte, L_byte
                00011
                00012 PROG1   CODE       ; сегмент кода PROG1
  0000          00013         GLOBAL  Mpy
0000   01??     00014 Mpy     clrf    H_byte
0001   01??     00015         clrf    L_byte
0002   3008     00016         movlw   8
0003   00??     00017         movwf   Count
0004   08??     00018         movf    Mulcnd,W
0005   1003     00019         bcf     STATUS,C     ; Очистка флага C.
0006   0C??     00020 Loop    rrf     Mulplr,f
0007   1803     00021         btfsc   STATUS,C
0008   07??     00022         addwf   H_byte,f
0009   0C??     00023         rrf     H_byte,f
000A   0C??     00024         rrf     L_byte,f
000B   0B??     00025         decfsz  Count,f
000C   2???     00026         goto    Loop
000D   3400     00027         retlw   0
                00028         END
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

В результате трансляции по схеме на рисунке 2.3 MPASM получил перемещаемый объектный код подпрограммы умножения, для размещения которого было использовано 14 ячеек памяти программ с относительными адресами в диапазоне 0x0000 – 0x000D. Для хранения данных ассемблер выделил ячейки с относительными адресами в диапазоне 0x0000 – 0x0004.

Компоновщик MPLINK выполнил сборку программы и сформировал загрузочный модуль в Intel HEX формате:

```
:020000040000FA
:100000000608A1000608A00006200028A201A301FE
:100010000830A40020080310A10C0318A207A20CAA
:08002000A30CA40B0C28003412
:044000000034003454
:00000001FF
```

Карта памяти, построенная компоновщиком, показывает, что распределение ячеек памяти программ и данных получилось таким же, как и в первом варианте.

```
     MPLINK 4.30, Linker
Linker Map File - Created Sun Oct 21 22:02:41 2018
                  Section Info
Section     Type      Address     Location  Size(Bytes)
---------  ---------  --------  ---------  ---------
PROG1      code       0x000000    program   0x000028
.cinit     romdata    0x002000    program   0x000004
.udata     udata      0x000020       data   0x000005

Program Memory Usage
Start         End
---------   ---------
0x000000    0x000013
0x002000    0x002001
22 out of 8453 program addresses used, program memory utilization is 0%

Symbols - Sorted by Address
Name    Address   Location    Storage File
-----  ---------  ---------  --------- ---------
main   0x000000    program    static C:\PIC\Lr\example.asm
mpy    0x000006    program    extern C:\PIC\Lr\mul8x8.asm
loop   0x00000c    program    static C:\PIC\Lr\mul8x8.asm
Mulcnd 0x000020       data    extern C:\PIC\Lr\mul8x8.asm
Mulplr 0x000021       data    extern C:\PIC\Lr\mul8x8.asm
H_byte 0x000022       data    extern C:\PIC\Lr\mul8x8.asm
L_byte 0x000023       data    extern C:\PIC\Lr\mul8x8.asm
Count  0x000024       data    static C:\PIC\Lr\mul8x8.asm
```

Менеджер библиотек MPLIB позволяет создавать и модифицировать файлы библиотек. Библиотечный файл является коллекцией объектных модулей, которые размещены в одном файле. MPLIB использует объектные модули с именем типа "filename.o" формата COFF (Common Object File Format). Использование библиотечных файлов упрощает компоновку программы, делает ее более структурированной и облегчает ее модификацию.

2.4 Работа ассемблера

На рисунке 2.4 приведена схема, поясняющая работу двухпроходного ассемблера. Как правило, интегрированные системы разработки и их основные компоненты, включая трансляторы, предназначены для разработки программ для определенного набора семейств МК. Так как МК различных семейств могут иметь различные наборы инструкций, то и языки ассемблера в этой части будут отличаться. Поэтому ассемблер использует таблицы машинных команд и директив ассемблера, которые формируются для каждого семейства МК.

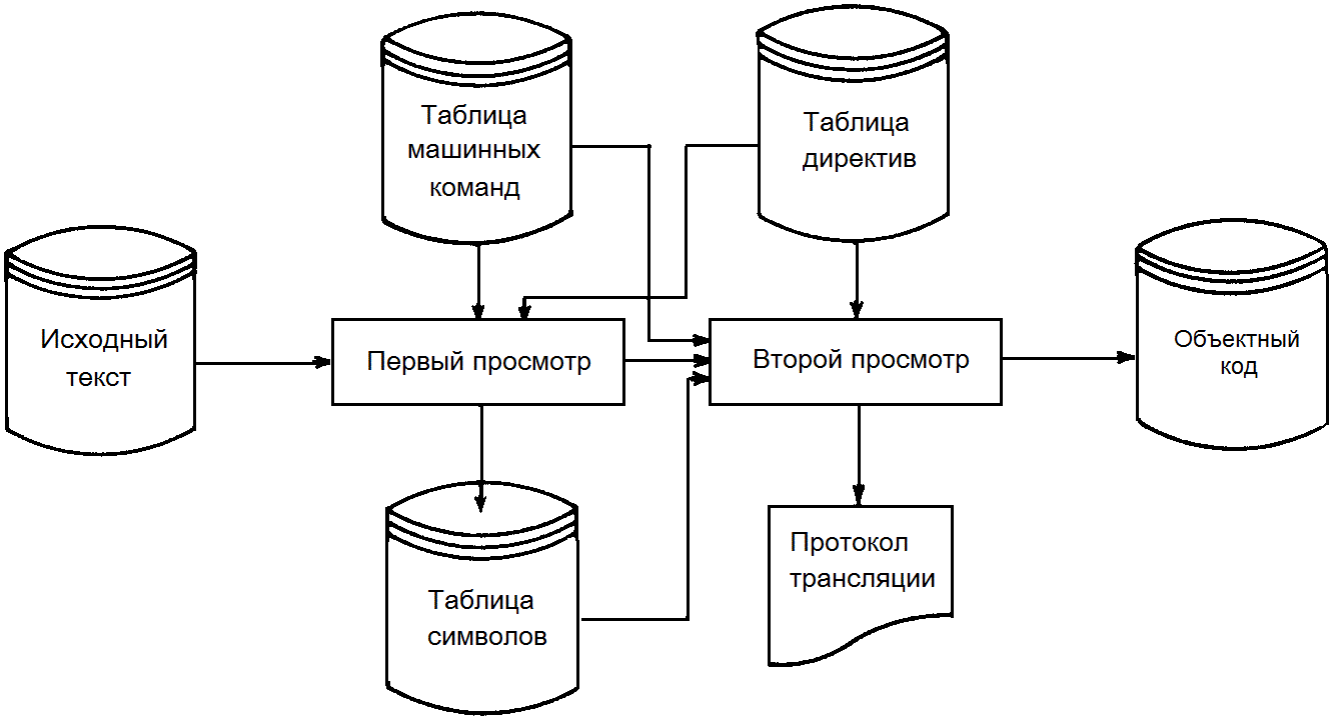


Рисунок 2.4 – Работа ассемблера

Цель первого прохода ассемблера - формирование таблицы символов. Ассемблер просматривает каждую строку текста и определяет тип содержащейся мнемоники: машинной команды или директивы ассемблера. Поиск выполняется в загруженных таблицах машинных команд и директив. Если ассемблер встречает в поле метки или поле параметра строки лексические единицы, не являющиеся предопределенными словами языка ассемблера, они заносятся в таблицу символов. Под символами здесь подразумеваются метки команд и идентификаторы данных.

Алгоритм первого прохода ассемблера при генерации абсолютного объектного кода приведен на рисунке 2.5. Переменная программного счетчика PC используется для формирования адресов ячеек памяти программ. По директиве ORG ассемблер модифицирует значение программного счетчика, по директиве EQU присваивает символу идентификатора значение операнда – адрес ячейки памяти данных. По директиве END работа ассемблера на первом проходе завершается. При нахождении символа в поле метки машинной команды ему присваивается значение PC, и он заносится в таблицу символов. Перед чтением следующей строки исходного текста значение переменной PC инкрементируется.

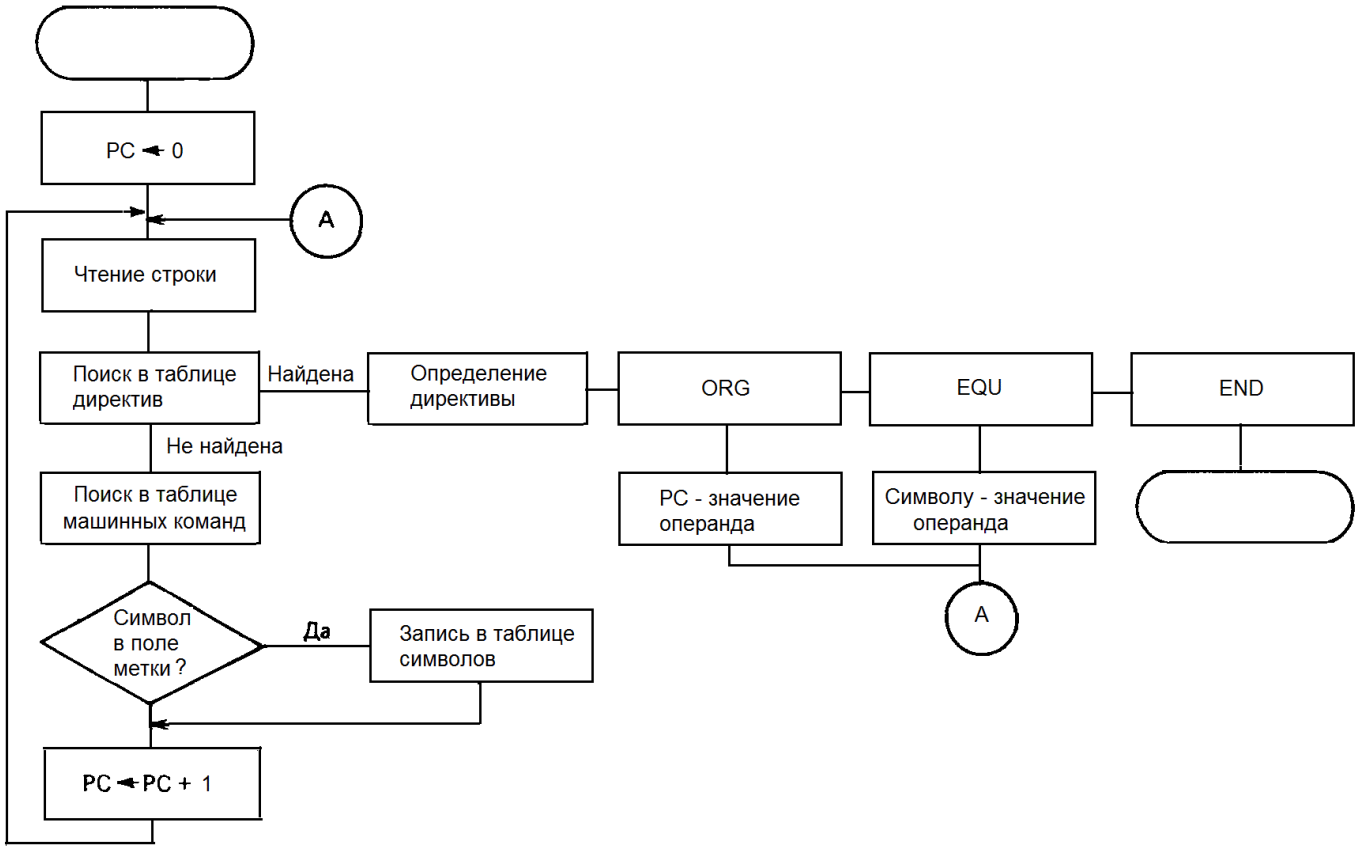


Рисунок 2.5 – Схема алгоритма первого прохода ассемблера

На втором проходе трансляции (рисунок 2.6) ассемблер повторно просматривает каждую строку текста и определяет тип содержащейся мнемоники: машинной команды или директивы ассемблера. Если найдена мнемоника машинной команды, то она заменяется ее машинным кодом. Сформированная команда заносится в файл объектного кода, добавляется запись в файл листинга.

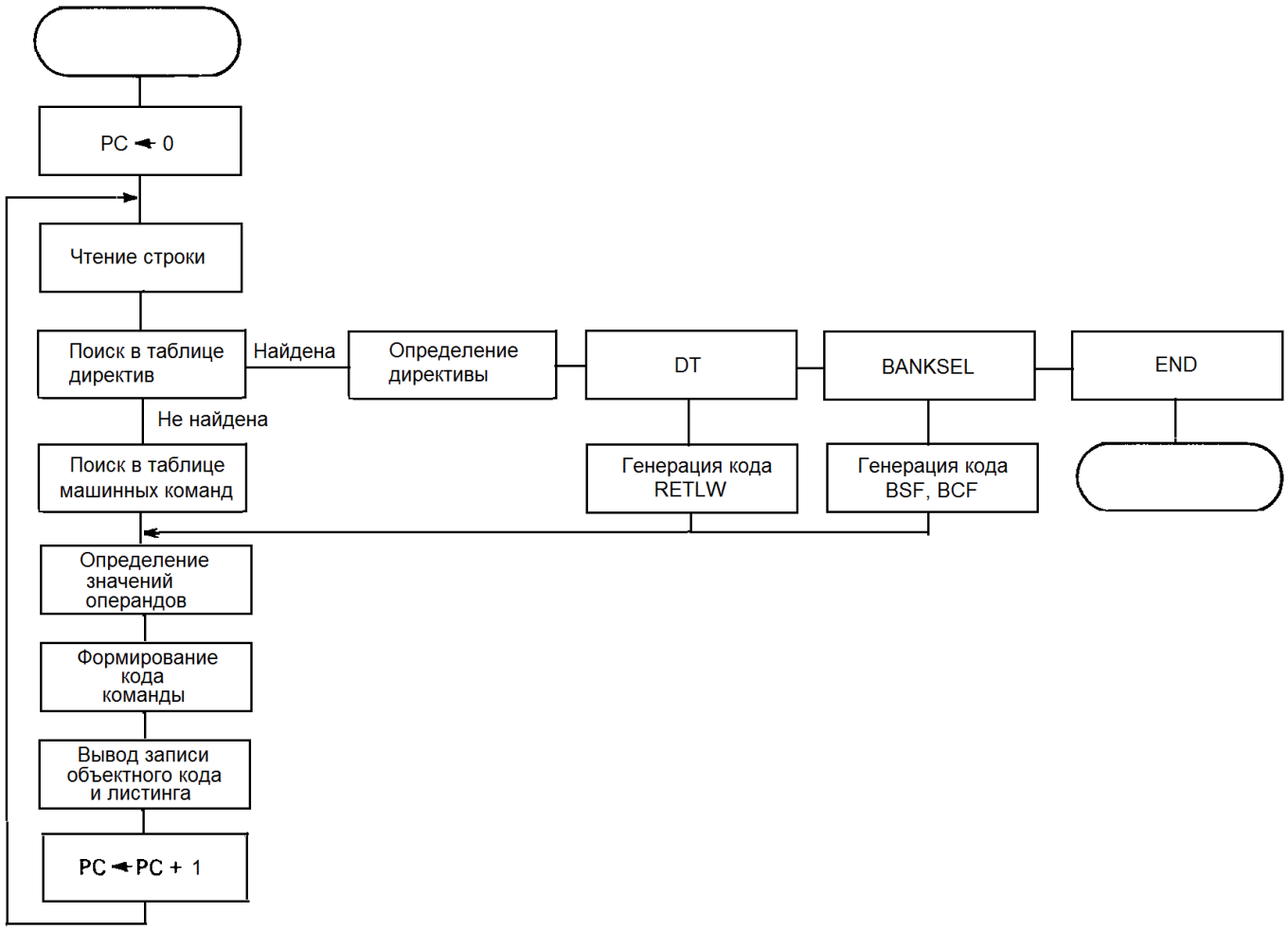


Рисунок 2.6– Схема алгоритма второго прохода ассемблера

Если найдена директива генерации кода, то ассемблер ее выполняет. При этом символы идентификаторов ассемблер заменяет адресами ячеек памяти данных. Например, по директиве

[<label>] DT <expr>[{,<expr>}]

ассемблер генерирует серию команд RETLW для 8-разрядных значений <expr>. Каждое значение <expr> сохраняется в отдельной команде RETLW. Таким образом, в памяти программ можно сформировать таблицу данных. По директиве

BANKSEL <var>

ассемблер генерирует последовательность команд bsf, bcf, выполняемых над разрядами RP1, RP0 регистра STATUS, тем самым обеспечивается выбор активного банка в адресном пространстве памяти данных. Например, по директиве

BANKSEL TRISB

ассемблер сформирует код команд

```
bsf     STATUS,RP0
bcf     STATUS,RP1
```

В результате будет активирован банк 1, в котором располагается регистр TRISB. По директиве END работа ассемблера на втором проходе завершается.

2.5 Работа компоновщика

Работа компоновщика на примере формирования загрузочного кода из двух объектных модулей приведена на рисунке 2.7 . На первом проходе компоновщик выполняет поиск внешних символов, определяемых директивой EXTERN, и отождествляет их с точно такими же глобальными символами, определяемыми директивой GLOBAL. Компоновщик формирует таблицу внешних символов. При этом каждой паре внешних и глобальных символов назначается абсолютный адрес ячейки памяти программ (для меток) или памяти данных (для идентификаторов)

$$A_{abs} = A_{rel} + A_{base}, \quad (1)$$

где A_{rel} – относительный адрес глобального символа;

A_{base} - базовый адрес, назначенный объектному модулю.

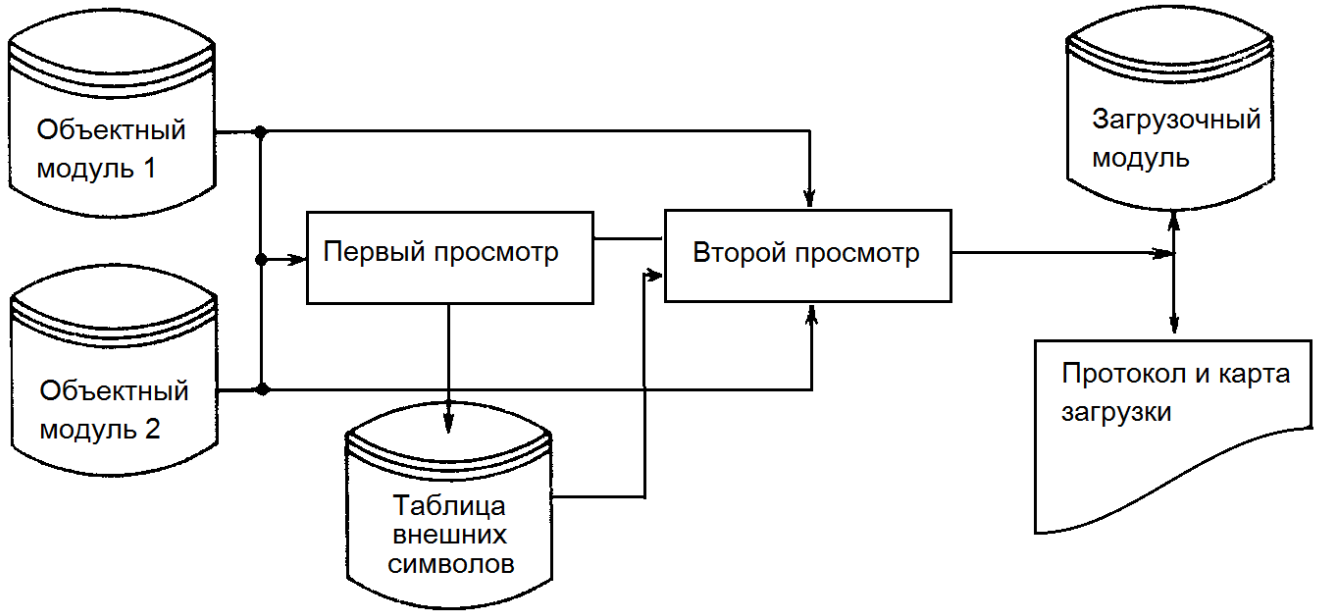


Рисунок 2.7 – Работа компоновщика

На втором проходе компоновщик формирует загрузочный код программы, выстраивая коды команд по полученной последовательности абсолютных адресов памяти программ. При этом в поле параметров команд компоновщик заносит полученные абсолютные адреса ячеек памяти данных для операндов и абсолютные адреса ячеек памяти программ для точек переходов.

2.6 Программирование МК PIC16 на языке Си

Используемые для программирования МК диалекты языка Си незначительно отличаются от стандарта ANSI C. Эти отклонения от стандарта, ориентированного на разработку программ для компьютеров, вызваны особенностями архитектуры и использования МК:

- код исполняемой программы хранится в ПЗУ;

- память данных разделена на банки, а реализованные виды адресации позволяют обращаться только к ячейкам активного банка;

- регистры специальных функций, управляющие работой периферийных модулей, и регистры портов часто располагаются в адресном пространстве памяти данных;

- примитивность реализации системы прерываний МК;

- в МК невысокой производительности, как правило, не используются операционные системы, поэтому программа должна напрямую взаимодействовать с процессорным ядром, портами и периферийными модулями.

Основы программирования на языке Си изложены в многочисленных источниках, поэтому в учебном пособии не рассматриваются. Рассмотрим особенности программирования на языке Си МК семейства PIC16 при использовании компилятора PICC Lite компании HI-TECH [8].

Компилятор PICC Lite является бесплатной ограниченной (без оптимизации) версией компилятора PICC HI-TECH и может быть интегрирован в среду системы MPLAB IDE. В состав PICC входит несколько стандартных библиотек. Специфические особенности конкретного МК отражены в файле заголовка. Компилятор за-

грузит файлы заголовков, предназначенных для используемого в проекте МК, например, PIC16F877A, при выполнении директив препроцессора

```
#include <pic.h>
#include <pic1687x.h>
```

Поддержка МК заключается в назначении семантических имен регистрам специальных функций, отдельным битам и описании сервисных макросов.

Настроить разряды слова конфигурации МК при его программировании поможет макрос CONFIG. Если необходимо запретить сторожевой таймер, отключить защиту кода и данных и настроить контроллер на работу с тактовым генератором типа XT, то макрос будет иметь параметры:

```
__CONFIG(WDTDIS & XT & UNPROTECT);
```

МК семейства PIC16 могут выполнять операции с отдельными битами регистров (инструкции BSF, BCF). Компилятор PICC будет генерировать код, состоящий из этих инструкций, при трансляции операторов

```
int cell;
cell |=0x40;
```

Для установки или сброса разряда в регистре можно использовать макросы:

```
#define bitset (var, bitn) ((var) |= 1<< (bitn))
#define bitclr (var, bitn) ((var) &= ~(1<< (bitn)))
```

Необходимость побитовой обработки также поддерживается типом данных bit – одноразрядное целое и возможностью использовать двоичные литералы в формате 0b<число>, например 0b101001010. Переменные типа bit объявляются стандартными средствами:

```
static  bit  flag;
```

не могут передаваться в функцию в качестве аргумента, но функция может возвращать значение типа бит. Переменные этого типа ведут себя как переменные типа unsigned char, но при этом могут принимать значение 0 или 1. Поэтому их удобно использовать для определения различных логических переменных и флагов. Такой подход значительно позволяет экономить оперативную память МК. Битовым переменным можно присваивать результат операций отношения

```
flag = char_var != 0;
```

Возможность создавать битовые переменные может быть использована, для удобного доступа к отдельным битам регистров специальных функций. При описании битовых переменных с указанием абсолютного адреса необходимо указывать абсолютный номер бита в непрерывном битовом поле, которое образуется из последовательности регистров. Например, 5-тий бит в ячейке памяти с адресом 3 будет иметь абсолютный номер равный 3*8+5. Например, макрос:

```
#define PORTBIT(adr, bit) ((unsigned)(&adr)*8+(bit))
```

позволяет объявлять линии порта ввода, как битовые переменные:

```
static unsigned char ptKeys @ &PORTB;
static bit btLeft  @ PORTBIT(ptKeys, 0);
static bit btRight @ PORTBIT(ptKeys, 1);
static bit btDown @ PORTBIT(ptKeys, 2);
static bit btUp    @ PORTBIT(ptKeys, 3);
```

Для эффективного размещения и доступа к переменным компилятору необходимо сообщить дополнительную информацию:

- будет ли переменная изменяться программой (константы);

- может ли переменная изменяться помимо программы (порты и регистры специальных функций);

- должна ли переменная обнулиться при запуске программы;

- в каком банке памяти должна размещаться переменная.

PICC компилятор поддерживает стандартные квалификаторы const и volatile. Квалификатор const указывает на неизменность объекта. Любая попытка изменить содержимое переменной, помеченной квалификатором const, приведет к выдаче компилятором соответствующего сообщения. Все такие объекты должны инициализироваться при описании, и в дальнейшем их значения не могут быть изменены:

```
const int version = 5;
```

Квалификатор volatile сообщает компилятору, что нельзя гарантировать сохранность значения переменной между двумя удачными обращениями к ней из программы. Имеются в виду регистры специальных функций, значение которых могут изменять периферийные модули. Кроме этого, этим квалификатором необ-

ходимо помечать переменные, которые изменяются подпрограммами обработки прерываний. Пример:

```
volatile unsigned char P_A @ 0x05;
```

Дополнительные квалификаторы persistent, bank1, bank2, bank3 учитывают специфику МК. По умолчанию любая переменная, если она не была явно инициализирована, обнуляется. Чтобы исключить очистку переменной при рестарте программы нужно воспользоваться квалификатором persistent. Существует несколько библиотечных функций, которые позволяют проверять и инициализировать такие переменные. При использовании для локальных переменных необходимо добавить ключевое слово static, например:

```
void func (void)
{
static persistent int intvar2;
// Тело функции.
}
```

Квалификаторы bank1, ... bank3 – используются для размещения переменных в указанном банке памяти данных, например:

```
static bank3 unsigned char Cell;
```

Для описания функции - обработчика прерывания необходимо воспользоваться квалификатором interrupt. Компилятор будет транслировать эту функцию специальным образом. Прежде всего, будут сохранены и восстановлены все регистры, которые были задействованы в обработчике, для выхода из функции будет использоваться инструкция retfie. Функция обработчик прерываний должна возвращать значение типа void и не должна иметь аргументов. Пример простейшего обработчика прерываний:

```
long tick_cnt;
void interrupt tc_int(void)
{
++tick_cnt;
}
```

Так как в МК PIC16 реализован только один вектор прерываний, то и функция обработчика должна быть единственной в программе. Она должна выполнять анализ источника прерывания и выполнять соответствующие действия.

Для управления системой прерываний в PICC предусмотрены функции:

```
ei();          // разрешить все прерывания
di();          // запретить все прерывания
```

и возможность управлять соответствующими разрядами регистров специальных функций:

```
ADIE = 1;    // разрешить прерывание от АЦП
PEIE = 1;    // разрешить все прерывания от периферийных устройств
```

Пример программы на языке Си, аналогичной рассмотренным в подразделе 2.3 программам на языке ассемблера:

```
#include <pic.h>
#include <math.h>
#include <pic1687x.h>

main(void)
{
unsigned char mulcnd;
unsigned char mulplr;
unsigned int  product;
while (1){
        mulcnd=PORTB;
        mulplr=PORTB;
        product=mulcnd*mulplr;
        }
}
```

Сравнительные результаты трансляции при использовании компилятора PICC Lite и ассемблера приведены в таблице 2.3.

Таблица 2.3 – Сравнительные результаты компиляции

Транслятор	Затраты памяти программ, ячеек	Затраты памяти данных, ячеек	Время выполнения основного цикла, машинных циклов
Компилятор PICC Lite	78	14	247
Ассемблер MPASM	20	5	79

2.7 Программирование МК K1986BE9x на языке Си

В пакет разработки RealView MDK ARM входит компилятор C/C++ ARM RealView [9]. Как и для восьмиразрядных МК, специфика программирования на языке Си МК с 32-разрядным ядром Cortex-M3 связана с управлением периферийными модулями. Загрузка файла заголовка для МК семейства K1986BE9x выполнятся по директиве препроцессора

#include "MDR32Fx.h"

Для того чтобы использовать порты ввода-вывода, нужно предварительно сконфигурировать их соответствующим образом. На самом низком уровне работа с портами ввода-вывода осуществляется с помощью специальных регистров микроконтроллера. Эти регистры доступны как ячейки памяти, расположенные по определенным адресам. Для облегчения труда программиста разработчики МК предоставляют специальные библиотеки функций для работы с периферийными устройствами. Для МК семейства K1986BE9x разработана библиотека «MDR32F9Qx Standard Peripherals Library, содержащая функции для работы со всеми встроенными периферийными модулями. При этом программисту не надо напрямую обращаться к регистрам, достаточно вызвать подходящую функцию.

Для работы с портами ввода-вывода к проекту должен быть подключен библиотечный модуль MDR32F9Qx_port.c. Для удобства настройки портов ввода-вывода рекомендуется использовать специальную структуру типа PORT_InitTypeDef. Тип PORT_InitTypeDef описан в заголовке MDR32F9Qx_port.h:

```
typedef struct
{ uint16_t   PORT_Pin;
             PORT_OE_TypeDef PORT_OE;
             PORT_PULL_UP_TypeDef PORT_PULL_UP;
             PORT_PULL_DOWN_TypeDef PORT_PULL_DOWN;
             PORT_PD_SHM_TypeDef PORT_PD_SHM;
             PORT_PD_TypeDef PORT_PD;
             PORT_GFEN_TypeDef PORT_GFEN;
             PORT_FUNC_TypeDef PORT_FUNC;
             PORT_SPEED_TypeDef PORT_SPEED;
             PORT_MODE_TypeDef PORT_MODE;
} PORT_InitTypeDef;
```

В поле PORT_Pin типа uint16_t указывают, какие линии подлежат конфигурированию. Каждый бит в этом поле отвечает за отдельную линию ввода-вывода. Линии нумеруются, начиная с нуля. Так, если указать единицу в 4-м разряде, т.е. занести в поле значение 0x0010, это будет означать, что конфигурируют линию 4. Можно указать единицы сразу в нескольких разрядах, например, во 2-м, 3-м и 5-м. Тогда все настройки будут, соответственно, относится к линиям 2, 3 и 5. Для этого в поле PORT_Pin необходимо записать значение 0x002C.

Для работы с цифровым портом необходимо задать функцию цифрового входа или выхода. Для задания направления используется поле PORT_OE типа PORT_OE_TypeDef. Если требуется сделать вывод цифровым входом, то:

PortInitStructure.PORT_OE = PORT_OE_IN;

а если цифровым выходом, то:

PortInitStructure.PORT_OE = PORT_OE_OUT;

В поле PORT_SPEED типа PORT_SPEED_TypeDef указывают скорость работы линии вывода. Возможны следующие значения:

– PORT_OUTPUT_OFF – выход выключен;

– PORT_SPEED_SLOW – низкая скорость (фронт порядка 100 нс, частота до 5 МГц);

– PORT_SPEED_FAST – высокая скорость (фронт порядка 20 нс, частота до 25 МГц);

– PORT_SPEED_MAXFAST – предельно высокая скорость (фронт порядка 10 нс, частота до 50 МГц).

К линиям порта PB3 – PB0 в отладочном стенде подключены светоизлучающие диоды. Подпрограмма-функция InitPortLed для начальной инициализации линий PB3 - PB0 на вывод может быть реализована следующим образом:

```
#define LEDS_MASK    (0xF)
#define LED0_OFS     (0)

void InitPortLed(void) {
    MDR_PORTB->FUNC &= ~((0xFF << (LED0_OFS << 1)));   // Port
    MDR_PORTB->ANALOG |= LEDS_MASK;                    // Digital
    MDR_PORTB->PWR |= (0xAA << (LED0_OFS << 1));       // Slow
```

```
    MDR_PORTB->RXTX &= ~LEDS_MASK;
    MDR_PORTB->OE |= LEDS_MASK;
}
```

Управлять состоянием линий порта можно путем установки или сброса с помощью битовых масок соответствующих разрядов в регистре MDR_PORTB->RXTX.

Схема подключения в отладочном стенде кнопок джойстика к линиям портов МК приведена на рисунке 2.8.

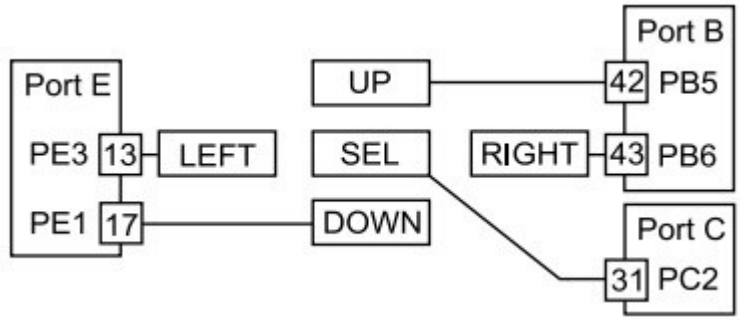


Рисунок 2.8 – Подключение кнопок джойстика

Функция InitPortJoystick для начальной инициализации линий трех портов, которые используются для считывания информации с кнопок джойстика:

```
void InitPortJoystick(void) {
   MDR_PORTB->FUNC  &= ~((0xF << (5 << 1)));  // UP - PB5, RIGNHT - PB6
   MDR_PORTB->ANALOG |= 0x3 << 5;             // Digital

   MDR_PORTC->FUNC &= ~((0x3 << (2 << 1)));   // SEL - PC2
   MDR_PORTC->ANALOG |= 0x1 << 2;             // Digital

   MDR_PORTE->FUNC &= ~((0x33 << (1 << 1)));  //DOWN - PE1, LEFT - PE3
                                               // (0b110011)
   MDR_PORTE->ANALOG |= 0x5 << 1;             // Digital
}
```

Подпрограмма-функция GetKey возвращает унитарный код нажатой кнопки:

```
int GetKey(void)
{
 int data;
   data = 0;
   if(~(MDR_PORTC->RXTX) & (1<<2)) data |= 1;          // SEL      PC2
```

```
    if(~(MDR_PORTB->RXTX) & (1<<6)) data |= (1<<1);   // RIGHT  PB6
    if(~(MDR_PORTE->RXTX) & (1<<3)) data |= (1<<2);   // LEFT   PE3
    if(~(MDR_PORTB->RXTX) & (1<<5)) data |= (1<<3);   // UP     PB5
    if(~(MDR_PORTE->RXTX) & (1<<1)) data |= (1<<4);   // DOWN   PE1
    return (data);
}
```

При работе с механическими кнопками всегда возникает проблема подавления «дребезга», который проявляется в виде многократной реакции МПУ при однократном нажатии на кнопку. В МПУ используют программные способы подавления «дребезга», которые основаны на введении временной задержки между последовательными опросами соответствующих линий порта ввода. Для формирования необходимых интервалов времени необходимо использовать прерывания от встроенных таймеров. Иначе МК не сможет при формировании задержки выполнять другую работу.

Процессор с ядром Cortex-M3 имеет системный таймер SysTick, выполняющий обратный счет от загруженного в него значения до нуля; перезагрузка значения в регистр LOAD происходит по следующему фронту синхросигнала, затем счёт продолжается. Для инициализации таймера SysTick необходимо выбрать источник тактового сигнала, разрешить прерывания и включить его. В регистр LOAD необходимо загрузить значение задержки. Чтобы настроить таймер на прерывание раз в одну миллисекунду, включим тактирование от HCLK – это частота тактирования ядра микроконтроллера. Она равна 8 МГц или 8000000 тактов в секунду. Тогда количество тактов в одной миллисекунде составит 800. Так как один такт используется для загрузки таймера, то это значение необходимо уменьшить на единицу. Разряды регистра управления необходимо настроить на выбранный режим работы:

```
#define CLKSOURCE (1<<2) //Источник синхросигнала: 0 - LSI, 1 - HCLK
#define TCKINT  (1<<1)        //Разрешает запрос на прерывание
#define ENABLE (1<<0)         //Разрешает работу таймера
void Init_SysTick (void) //Прерывание раз в миллисекунду.
{
        SysTick->LOAD |= (8000000/1000)-1;
        SysTick->CTRL |= CLKSOURCE|TCKINT|ENABLE;
}
```

Контроллер вложенных векторных прерываний NVIC представляет собой подсистему, управляющую аппаратными прерываниями. NVIC позволяет задать каждому прерыванию определенный приоритет, в соответствии с которым будет определяться порядок обработки запросов на прерывания, поступивших одновременно. При появлении запроса прерывания от системного таймера SysTick, управление будет передаваться на вектор прерывания SysTick_Handler. Для обработки прерывания можно использовать функцию SysTick_Handler, выполняющую обратный отсчет миллисекунд:

```
volatile uint32_t  Delay_dec = 0;
void SysTick_Handler (void)
{
 if (Delay_dec) Delay_dec--;
}
```

Функция задержки, использующая данное прерывание, принимает длительность задержки (в миллисекундах) Delay_ms_Data, копирует ее в переменную Delay_dec. После того, как задержка выполнена, функция завершает свою работу.

```
void Delay_ms (uint32_t Delay_ms_Data)
{
Delay_dec = Delay_ms_Data;
while (Delay_dec) {};
}
```

2.7 Вопросы для самоконтроля к разделу 2

Какие средства используются для кодирования алгоритмов?

Каким образом определяются формальные языки?

Дайте определение порождающей грамматики.

Поясните роль терминального и нетерминального алфавитов в формальных грамматиках.

Поясните назначение механизма распознавания.

Чем отличаются алгоритмические языки высокого и низкого уровня?

Перечислите преимущества и недостатки использования языков высокого уровня.

Перечислите преимущества и недостатки использования языков ассемблера.

Перечислите основные лексические единицы языка ассемблера PIC16.

Перечислите основные виды мнемоник языка ассемблера PIC16.

Чем отличаются инструкции и директивы языка ассемблера PIC16?

Перечислите основные типы директив языка ассемблера PIC16.

Приведите примеры использования директив.

Что понимают под объектным и загрузочным кодом программы?

Чем отличаются абсолютный и перемещаемый объектный код?

Какой инструмент среды MPLAB IDE формирует объектный код программы?

Каким образом в среде MPLAB IDE выполняется трансляция программы сложной структуры в абсолютный код?

Каким образом в среде MPLAB IDE выполняется трансляция программы сложной структуры в перемещаемый код?

Какие директивы языка ассемблера PIC16 используют при трансляции в абсолютный и перемещаемый код?

Поясните назначение компоновщика.

Поясните работу ассемблера на первом и втором проходе трансляции.

Поясните работу компоновщика на первом и втором проходе.

В чем проявляется специфика кодирования на языке Си для МК?

Каким образом в программе на языке Си выполняют инициализацию портов и периферийных модулей?

Перечислите допустимые типы данных в языке Си.

Как связаны между собой данные и типы?

Как объявить переменную в языке Си?

Как объявить одномерный массив в языке Си?

Как инициализировать одномерный массив в языке Си?

Что такое оператор в языке Си? Выражение?

Чем в языке Си оператор отличается от выражения?

Какие в языке Си бывают операторы?

Приведите примеры операторов и выражений в языке Си.

Приведите пример классификации операций в языке Си.

Какие в языке Си бывают операции инкремента/декремента?

Как можно в языке Си классифицировать операторы?

Приведите примеры использования операторов ветвления языка Си.

Приведите примеры использования операторов цикла языка Си.

Каким образом в языке Си можно описать бесконечный цикл?

Для чего в языке Си используются указатели?

Как связать в языке Си указатель и массив?

Как в языке Си объявить указатель на массив?

3 Автономная отладка программ

3.1 Виды верификации программ

Можно выделить три основных вида верификации программ: визуальный, статический и динамический контроль (рисунок 3.1).

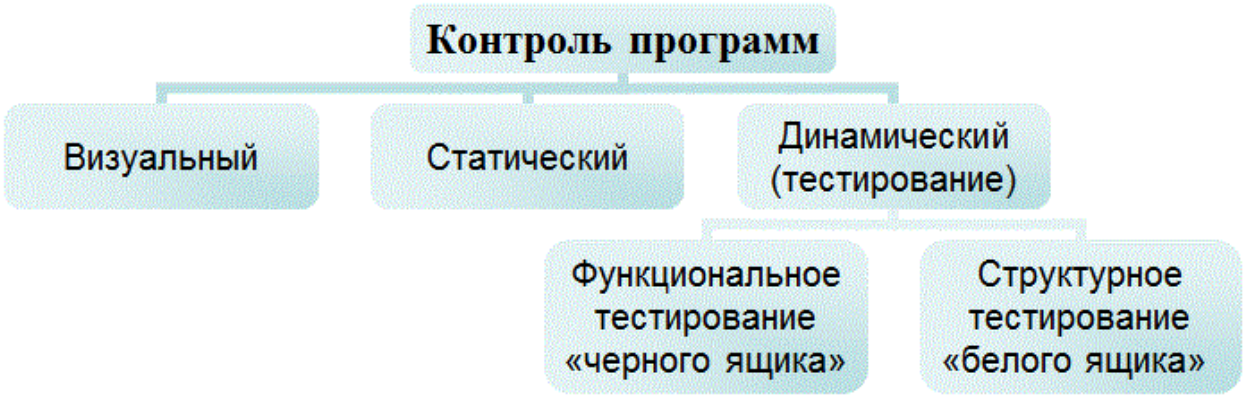


Рисунок 3.1 – Виды верификации программ

Визуальный контроль – предполагает чтение исходного текста программы, причем особое внимание уделяется следующим элементам:

- соответствию комментариев тексту программы;
- условиям в операторах ветвления и цикла;
- сложным логическим выражениям;
- возможности незавершения локальных циклов.

Второй этап визуального контроля - сквозной контроль программы (ее ручная прокрутка на нескольких заранее подобранных простых тестах).

Статический контроль - это проверка текста программы с помощью компилятора, при котором проверяется соответствие текста программы синтаксическим правилам языка программирования. При компиляции программы удается найти ошибки в написании меток, идентификаторов, ошибки в распределении адресного пространства памяти программ и данных.

Второй формой синтаксического контроля может быть контроль структурированности программ, то есть проверка выполнения соглашений и ограничений структурного программирования.

Третья форма статического контроля - контроль правдоподобия программы, при котором выявляют синтаксически корректные конструкции с признаками логических ошибок. Примеры неправдоподобных ситуаций:

- использование в программе значений неинициализированных переменных;

- наличие в программе переменных, подпрограмм, меток, не используемых в программе;

- наличие в программе фрагментов, никогда не выполняющихся;

- наличие в тексте программы локальных бесконечных циклов.

Динамический контроль программы - это проверка правильности программы при ее выполнении, т.е. тестирование. На этот этап приходится около 50 % общей стоимости разработки программного обеспечения.

Тестирование - процесс выполнения программы с целью обнаружения в ней ошибок. Выполнение программы без ошибок не свидетельствует об их отсутствии. Локализация и последующее исправление ошибки возможны только при ее проявлении. Проявление ошибки зависит от различных условий, то есть при одних условиях ошибку удается обнаружить, а при других условиях она себе не проявляет. Поэтому основной принцип тестирования предполагает выполнение заранее спланированных экспериментов, направленных на проверку выполнения программы в различных условиях. “Удачным” считается тест, при выполнении которого удалось обнаружить ошибку в программе.

Тестирование программы основано на сопоставлении реального хода выполнения программы и получаемых результатов с правильным ожидаемым выполнением программы и правильными результатами. Поэтому планирование любого тестового эксперимента обязательно включает формирование эталонной модели выполнения программы или ее части.

Основной проблемой обеспечения эффективности тестирования является его полнота. Частичное тестирование программы не может гарантировать отсутствие в ней ошибок. Зависимость трудоемкости полного тестирования от сложности программы носит нелинейный характер. Поэтому на практике обеспечить полноту тестирования сложных программ часто не представляется возможным. Поэтому и в

отлаженных программах могут быть ошибки. При этом некоторые ошибки могут ни разу себя не проявить не только при разработке МПУ, но и в процессе эксплуатации. Другие ошибки при эксплуатации проявляют себя в виде отказов техники.

Так получается, что завершающий этап тестирования сложных программ для офисных компьютеров выполняют пользователи. Поступающие разработчикам сведения об ошибках, претензии и рекламации содержат ценную информацию, позволяющую локализовать и исправлять остающиеся в программных продуктах ошибки. Исправленные и дополненные версии программ существенно надежнее α и β версий.

Требования к надежности встроенного программного обеспечения МПУ более жесткие. Многие встраиваемые МПУ работают в необслуживаемых режимах, круглосуточно, длительное время. Программное обеспечение таких МПУ должно обеспечивать автоматическое восстановление после сбоев (используя для этого сторожевые таймеры), восстановление работы после провалов или перерывов электропитания. Сложность и стоимость исправления ошибок на этапе эксплуатации существенно выше, чем при разработке МПУ. Поэтому вопросам тестирования встроенного программного обеспечения на этапах автономной и комплексной отладки уделяют особое внимание.

Различают функциональное и структурное тестирование. При *функциональном* тестировании программа рассматривается, как “черный ящик” (то есть ее текст не принимается во внимание). Происходит проверка соответствия поведения программы ее внешней функциональной спецификации. Исчерпывающее функциональное тестирование не всегда возможно, поэтому используют тесты с достаточной вероятностью обнаружения ошибок в программе.

При *структурном* тестировании программа рассматривается как “белый ящик” (текст открыт для использования). Происходит проверка логики программы. Полнота тестирования будет определяться возможностью проверки всех возможных путей в схеме алгоритма программы. Единственным средством, которое предоставляют отладчики интегрированных систем разработки, являются *профи-*

лировщики, позволяющие определять только частоту выполнения различных операторов программы.

Даже если предположить, что удалось полностью покрыть пути при структурном тестировании программы, в ней могут тоже содержаться ошибки. Возможные причины:

1) неправильно выполнена алгоритмизация, поэтому программа не соответствует внешней спецификации;

2) проявление ошибок зависит от значений данных.

Таким образом, ни структурное, ни функциональное тестирование не может быть исчерпывающим.

Известны два подхода к тестированию сложных программ, состоящих из множества модулей: пошаговое и монолитное тестирование. При *монолитном* тестировании сначала все модули тестируются по отдельности, а затем объединяются для совместного тестирования. При *пошаговом* тестировании очередной модуль для своего тестирования подключается к набору уже проверенных модулей.

Для автономного тестирования каждого модуля требуется модуль - драйвер, имитирующий вызов тестируемого модуля, и модули - заглушки, имитирующие работу вызываемых модулей. При пошаговом тестировании модули проверяются совместно, поэтому требуются или только драйверы, или только заглушки.

При пошаговом тестировании возможны две стратегии подключения модулей: нисходящая и восходящая. *Нисходящее* тестирование начинается с главного модуля программы, а затем последовательно подключают модули, вызываемые из уже протестированных модулей. Неподключенные вызываемые модули заменяют заглушками. При *восходящем* тестировании проверка программы начинается с терминальных модулей. Далее тестируют модуль, который вызывает уже протестированные модули. Неподключенные модули соседнего верхнего уровня заменяются драйверами.

3.2 Условия контролепригодности

В области технической диагностики понятие *контролепригодности* определяется, как свойство изделия обеспечивать достоверную оценку его технического состояния и раннее обнаружение неисправностей и отказов.

Возможность выполнения автономной и комплексной отладки программ также определяется свойствами контролепригодности. Для проведения отладки методом тестирования исполнитель программы должен обладать свойствами управляемости, наблюдаемости, предсказуемости.

Управляемость - свойство исполнителя программы, при котором его поведение поддается управлению, например, имеется возможность остановить выполнение программы в определенном состоянии, и затем снова продолжить ее выполнение. *Наблюдаемость* - свойство исполнителя программы, позволяющее проследить за его поведением, сменой его внутренних состояний. *Предсказуемость* - свойство исполнителя программы, позволяющее установить его в состояние, из которого все последующие состояния могут быть предсказаны.

Основным средством обеспечения контролепригодности исполнителя программы является *отладчик* (debugger) интегрированной среды разработки. Функции отладчика:

- обеспечивать управление исполнителем отлаживаемой программы;

- собирать информацию о работе исполнителя программы, обрабатывать и представлять на различных уровнях абстракции.

Основные инструменты обеспечения управляемости:

- пуск и останов исполнения программы по команде разработчика;

- пошаговое исполнение программы;

- останов выполнения программы по условию (простые и сложные точки останова - breakpoints).

Основные инструменты обеспечения наблюдаемости:

- отображение содержимого выбранных областей памяти программ и памяти данных, стека;

- отображение содержимого программного счетчика, регистров специальных функций и регистров общего назначения, флагов (окна наблюдения);

- измерение времени выполнения отдельных участков отлаживаемой программы (секундомер);

- формирование и отображение трассы программы (последовательности выполняемых команд с результатами их выполнения);

- отображение значений сигналов на линиях портов ввода-вывода (логический анализатор).

Отладчики современных интегрированных систем разработки, как правило, обеспечивают высокоуровневую символьную отладку программ, под которой понимают отображение очередности выполняемых операторов или команд, установки точек останова непосредственно в окне редактора с исходным текстом программы.

В кросс - платформенных системах разработки исполнителем кода отлаживаемой программы может являться либо целевой МП или МК, либо программный симулятор. Технически проще реализовать совместную работу отладчика и симулятора.

3.3 Программные симуляторы

Под *программным симулятором* понимают функциональную модель целевого процессора или МК, состоящую из областей памяти, выделяемых для хранения кода программы и данных, рабочих регистров, а также алгоритма, реализующего процесс выборки, декодирования и исполнения команд целевого процессора. Симуляторы МК также содержат функциональные модели периферийных модулей (параллельных и последовательных портов, интервальных таймеров).

На рисунке 3.2 приведена потоковая модель данных, представляющая совместную работу симулятора и отладчика в составе интегрированной среды разработки. В этой модели прямоугольниками представлены области памяти для хранения:

- кода отлаживаемой программы;

- данных, используемых целевой программной;

- адресов точек останова;

- трассы целевой программы;

- моделей внешних воздействий (стимулов);

- количества затраченных на выполнение целевой программы машинных циклов (таймер - секундомер).

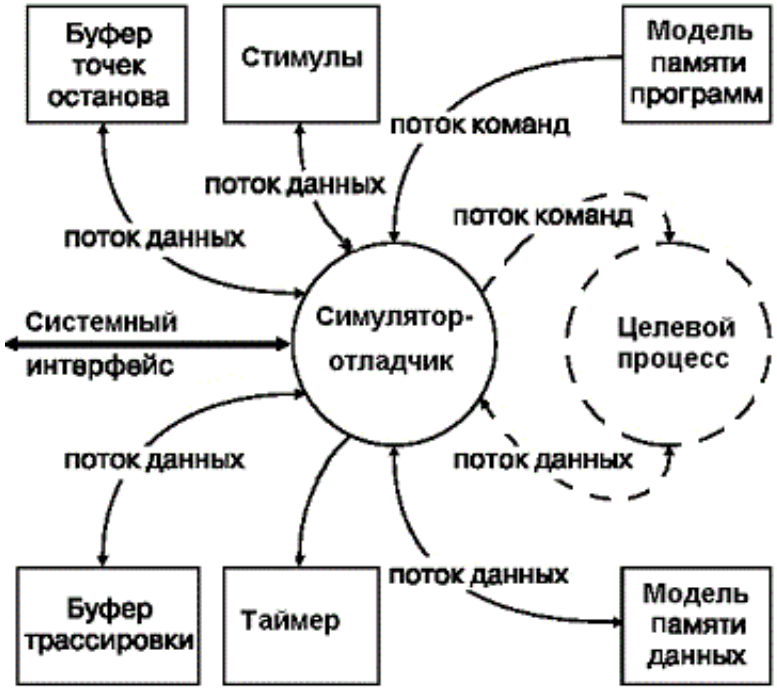


Рисунок 3.2 - Работа симулятора-отладчика

Окружность представляет процесс выполнения инструментальным компьютером алгоритмов отладчика и симулятора, а стрелки – потоки команд и данных отлаживаемой программы. Через системный интерфейс отладчик взаимодействует с экранным интерфейсом интегрированной среды разработки, средствами которого обеспечивается отображение областей памяти в соответствующих окнах и управление через экранное меню отладчика. При выполнении алгоритма симулятора выполняется модификация содержимого областей памяти, соответствующая работе целевого процессора. Этот результат можно интерпретировать, как следствие виртуального процесса выполнения отлаживаемой программы целевым процессором.

В качестве примера рассмотрим применение средств автономной отладки системы MPLAB IDE для тестирования программы, циклически выполняющей умножение однобайтных чисел, последовательно вводимых через PORTB. Программа состоит из основного модуля main и подпрограммы умножения mpy. Вызывающая программа main передает подпрограмме значения множимого mulcnd и множителя mulplr. Подпрограмма возвращает двухбайтовое произведение в ячейках H_byte и L_byte. Окно MPLAB IDE с исходным текстом программы, в котором отображается установленные точки останова и указатель выполняемой команды, приведены на рисунке 3.3.

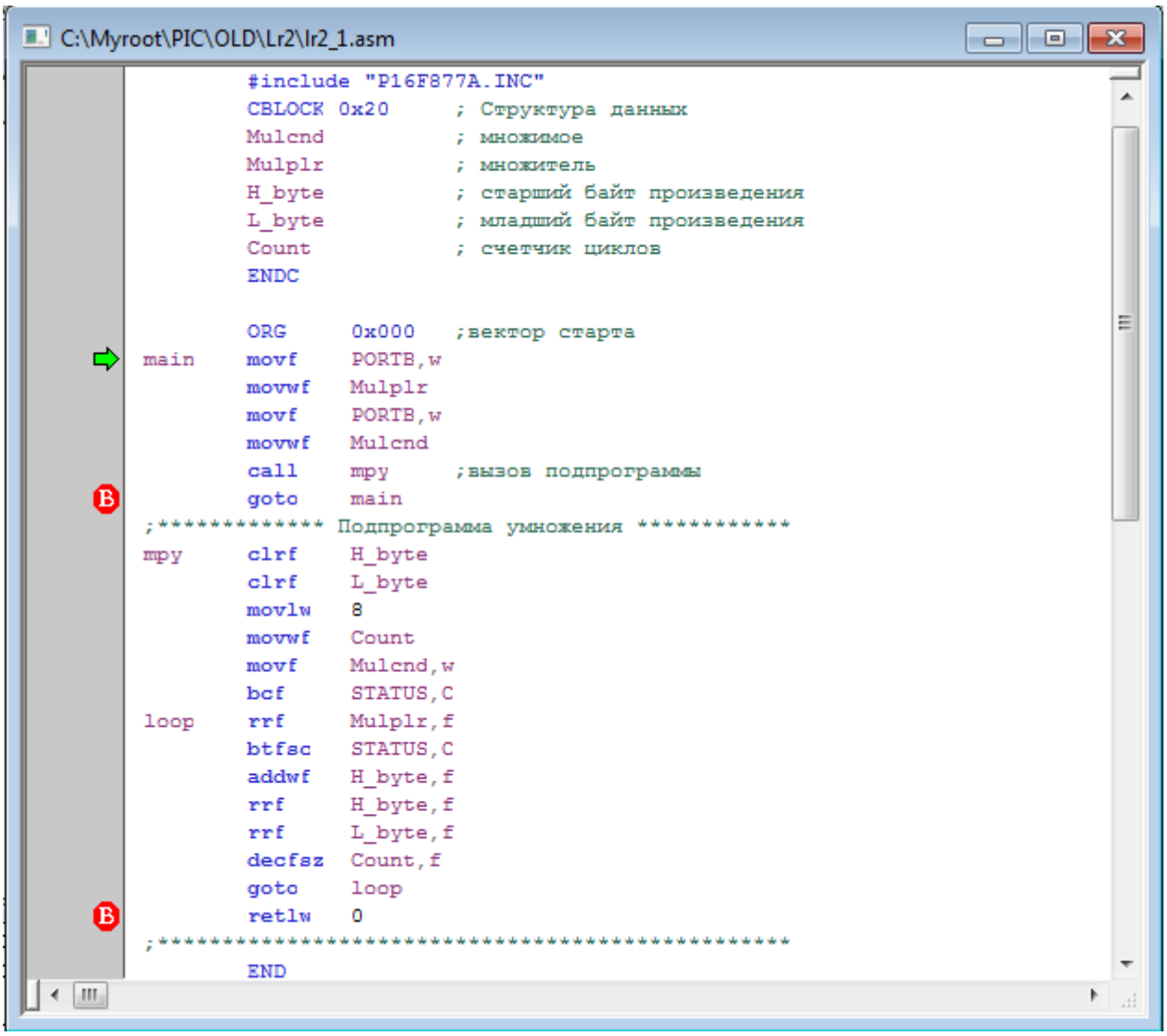


Рисунок 3.3 – Окно с исходным текстом

Для полного функционального тестирования подпрограммы mpy необходимо выполнить $2^8 \cdot 2^8 = 2^{16}$ вариантов тестирования для значений mulcnd от 0 до 255 и mulplr от 0 до 255.

План структурного тестирования подпрограммы mpy должен строиться на основе анализа схемы алгоритма (рисунок 3.4). В подпрограмме реализован метод последовательной обработки разрядов множителя со сдвигом накопленного произведения. Чтобы проверить все пути в схеме алгоритма достаточно выполнить проверку умножения значений mulcnd на значение mulplr, часть разрядов кода которого равна 1, а другая часть 0. Например, можно взять код числа 0x5A. Полнота проверки будет определяться также выбором значений множимого, для которых выполнение операции сложения будет происходить как с установкой флага переноса, так и без установки.

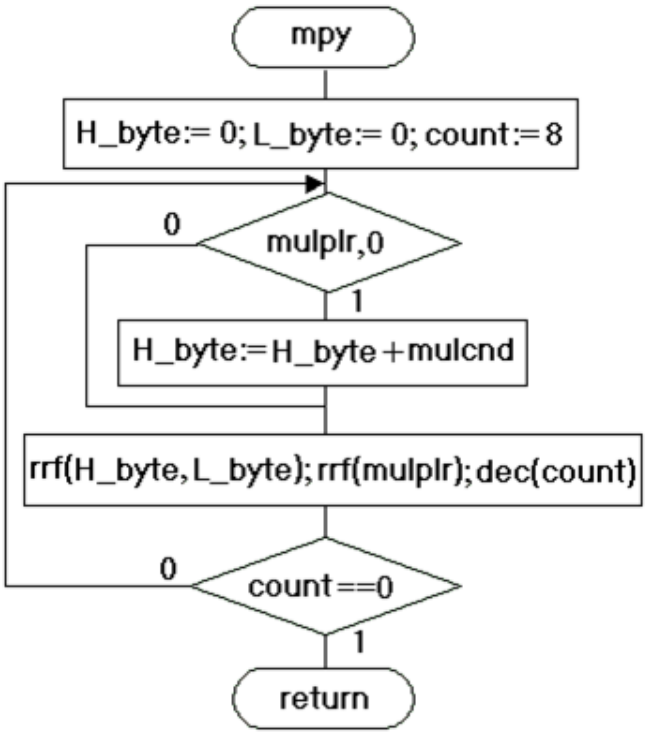


Рисунок 3.4 – Схема алгоритма подпрограммы умножения

Сформируем удобную среду для отладки программы. Для тестирования программы необходимо иметь средства имитации ввода данных через PORTB. Для этих целей можно использовать стимулы регистра типа *Register Injection*. Текстовый файл стимула регистра состоит из колонки значений, которые будут поочеред-

но передаваться регистру PORTB. С учетом 16-разрядного выравнивания в текстовый файл в одну колонку введем значения четырех 16-ричных значений, которые планируем ввести в PORTB:

0x005A
0x00A5
0x003C
0x00C3

Числа в нечетных строках будут использованы как значения множителя, а в четных строках – как значения множимого.

Пункт меню *Debugger>Stimulus>New Workbook* открывает окно настройки стимула регистра. В открывшемся окне на закладке *Register Injection* в указанных столбцах зададим значения:

- *Reg/Var* - PORTB;
- *Trigger* – Demand;
- *Data Filename* – имя файла стимула;
- *Wrap* – Yes;
- *Format* - Hex.

Стимул активируется нажатием кнопку *Apply*, сохранить стимул регистра можно командой *Save* (рисунок 3.5). Контролировать текущие значения в интересующих регистрах и ячейках памяти данных можно в окне наблюдения, которое открывается командой *View> Watch* (рисунок 3.6).

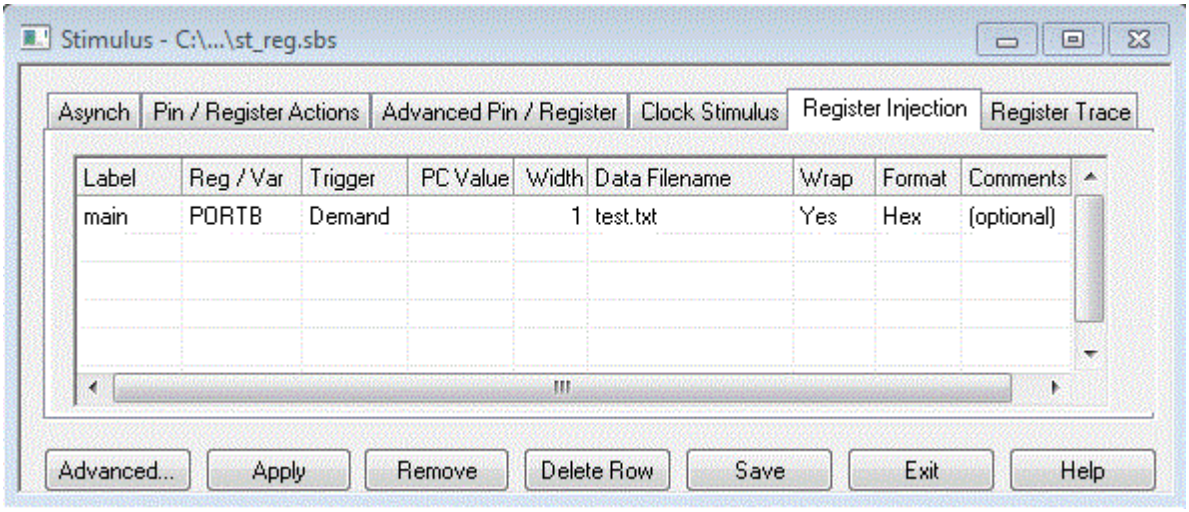


Рисунок 3.5 – Окно настройки стимулов

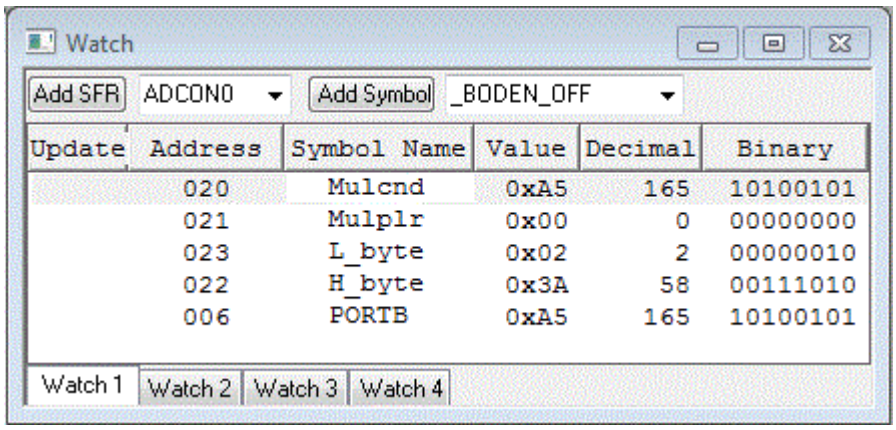


Рисунок 3.6 – Окно наблюдения

Открыть окно памяти трассы можно командой *View>Simulator Trace* (рисунок 3.7). Каждая запись памяти трассы содержит:

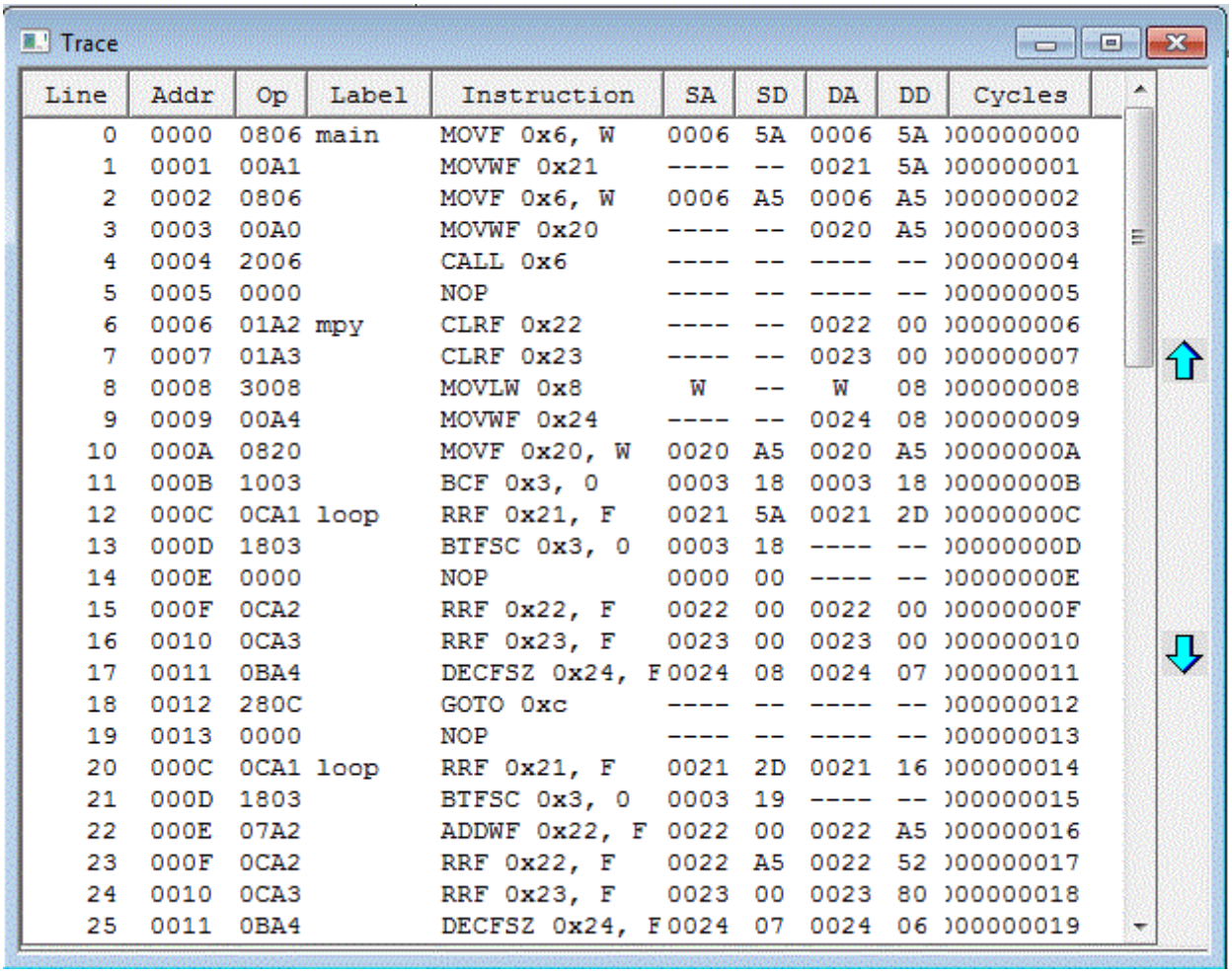
- адрес команды (Addr);
- код команды (Op);
- метку (Label);
- дизассемблерное представление команды (Instruction);
- адрес и содержимое ячейки источника (SA и SD);
- адрес и содержимое ячейки приемника (DA и DD);
- текущее время в циклах (Cycles).

Содержимое буфера трассировки можно сохранить в файле командой *Output to File*, очистить буфер трассировки можно командой *Clear Trace Buffer*.

Программные симуляторы являются доступным, удобным и эффективным кросс-платформенным инструментом отладки программ. Но, как и любая другая модель, симулятор не может абсолютно достоверно воспроизвести работу целевого процессора. Ограниченность симуляторов заключается в строго синхронном алгоритме моделирования целевого процессора. Все процессы рассматриваются для дискретных моментов времени, привязанных к машинному циклу. Таким же образом рассматриваются процессы в периферийных модулях и внешней среде. Имиторы внешних воздействий – стимулы изменяют свои состояния с привязкой к машинным циклам процессорного ядра. В действительности процессы, протекающие во внешней среде, асинхронны по отношению к машинным циклам. Изменения

формируемых МПУ выходных сигналов на внешних выводах периферийных модулей задержаны относительно фронтов тактовых импульсов процессорного ядра.

Поэтому успешно тестированная с помощью симулятора программа может в реальном МПУ выполняться иначе. Могут потребоваться изменения в коде программы. Эту доработку можно выполнить только при отладке программы на целевой аппаратной платформе.



Line	Addr	Op	Label	Instruction	SA	SD	DA	DD	Cycles
0	0000	0806	main	MOVF 0x6, W	0006	5A	0006	5A	00000000
1	0001	00A1		MOVWF 0x21	----	--	0021	5A	00000001
2	0002	0806		MOVF 0x6, W	0006	A5	0006	A5	00000002
3	0003	00A0		MOVWF 0x20	----	--	0020	A5	00000003
4	0004	2006		CALL 0x6	----	--	----	--	00000004
5	0005	0000		NOP	----	--	----	--	00000005
6	0006	01A2	mpy	CLRF 0x22	----	--	0022	00	00000006
7	0007	01A3		CLRF 0x23	----	--	0023	00	00000007
8	0008	3008		MOVLW 0x8	W	--	W	08	00000008
9	0009	00A4		MOVWF 0x24	----	--	0024	08	00000009
10	000A	0820		MOVF 0x20, W	0020	A5	0020	A5	0000000A
11	000B	1003		BCF 0x3, 0	0003	18	0003	18	0000000B
12	000C	0CA1	loop	RRF 0x21, F	0021	5A	0021	2D	0000000C
13	000D	1803		BTFSC 0x3, 0	0003	18	----	--	0000000D
14	000E	0000		NOP	0000	00	----	--	0000000E
15	000F	0CA2		RRF 0x22, F	0022	00	0022	00	0000000F
16	0010	0CA3		RRF 0x23, F	0023	00	0023	00	00000010
17	0011	0BA4		DECFSZ 0x24, F	0024	08	0024	07	00000011
18	0012	280C		GOTO 0xc	----	--	----	--	00000012
19	0013	0000		NOP	----	--	----	--	00000013
20	000C	0CA1	loop	RRF 0x21, F	0021	2D	0021	16	00000014
21	000D	1803		BTFSC 0x3, 0	0003	19	----	--	00000015
22	000E	07A2		ADDWF 0x22, F	0022	00	0022	A5	00000016
23	000F	0CA2		RRF 0x22, F	0022	A5	0022	52	00000017
24	0010	0CA3		RRF 0x23, F	0023	00	0023	80	00000018
25	0011	0BA4		DECFSZ 0x24, F	0024	07	0024	06	00000019

Рисунок 3.7 – Окно трассы программы

3.4 Вопросы для самоконтроля к разделу 3

Перечислите основные виды верификации программ.

Что является целью тестирования программы?

Как выполняется отладка программ, имеющих сложную структуру?

Чем отличаются технологии восходящей и нисходящей отладки?

Перечислите условия контролепригодности.

Приведите примеры средств, обеспечивающих выполнение условий контролепригодности.

Перечислите основные инструментальные средства и режимы работы отладчика системы MPLAB IDE.

Перечислите основные режимы отладки программ в среде MPLAB IDE?

Поясните содержание трассы программы?

Как в среде симулятора можно имитировать изменения внешних сигналов на линиях портов ввода МК?

Каким образом можно оценить длительность формируемых временных интервалов с использованием симулятора?

Перечислите основные режимы работы отладчика системы MPLAB IDE.

Как выполняется настройка среды MPLAB IDE на требуемый режим отладки?

Как выполняется настройка логического анализатора системы MPLAB IDE?

Чем отличаются сложные точки останова от простых? В каких случаях их целесообразно использовать?

Как можно выполнять автономную отладку программ, использующих механизм прерываний?

4 Средства загрузки

4.1 История развития

Особенностью встраиваемых МПУ является использование энергонезависимой полупроводниковой памяти для хранения кода программы. При этом исполняемый код программы непосредственно считывается из ячеек постоянного запоминающего устройства (ПЗУ или ROM). При появлении первых МП и МК использовались однократно-программируемые ПЗУ (PROM) и репрограммируемые ПЗУ с ультрафиолетовым стиранием (EPROM) (рисунок 4.1). Для загрузки полученного кода программы необходимо было использовать программаторы (рисунок 4.2).

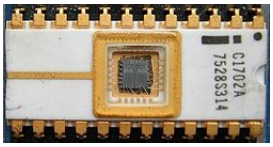


Рисунок 4.1 - EPROM

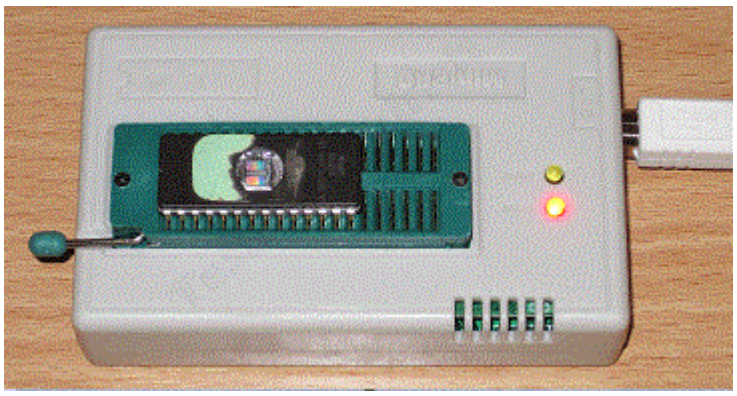


Рисунок 4.2 – Программатор ПЗУ

Таким образом, на физическом уровне загрузка кода программы в память МПУ сводилась к процедуре программирования ПЗУ. И если для загрузки уже отлаженного кода существенных неудобств это не вызывало, то с программными кодами, требующими выполнения комплексной отладки, возникали серьезные проблемы. Для исправления каждой найденной в программе ошибки было необходимо программировать новый экземпляр PROM или выполнять длительный процесс стирания и повторного программирования EPROM. Процедуру стирания источни-

ком ультрафиолетового света длиной волны 253 нм и световым потоком не менее 15 Вт·с/см^2 требовалось выполнять в течение 20 – 30 минут. При этом корпуса интегральных схем ПЗУ приходилось устанавливать на печатных платах в специальные панельки, допускающих возможность их извлечения для стирания и программирования.

Существенно снизить трудоемкость и продолжительность перезагрузки кода программы удалось за счет применения эмуляторов ПЗУ. *Эмулятор ПЗУ* - программно-аппаратное средство, позволяющее замещать ПЗУ программ на плате отлаживаемого МПУ. Простейшие эмуляторы ПЗУ представляли собой интегральную схему статического оперативного запоминающего устройства (ОЗУ) с резервным питанием от батареи или конденсатора с достаточно большой емкостью. Так как чаще использовались интегральные схемы ОЗУ, изготовленные по технологии КМОП, то для защиты их выводов от перенапряжений при коммутации с включенным питанием использовались буферные ТТЛШ схемы на линиях адреса, данных и управления (рисунок 4.3). Применение таких эмуляторов по-прежнему требовало использовать программаторы, но исключало необходимость выполнять длительную процедуру стирания.

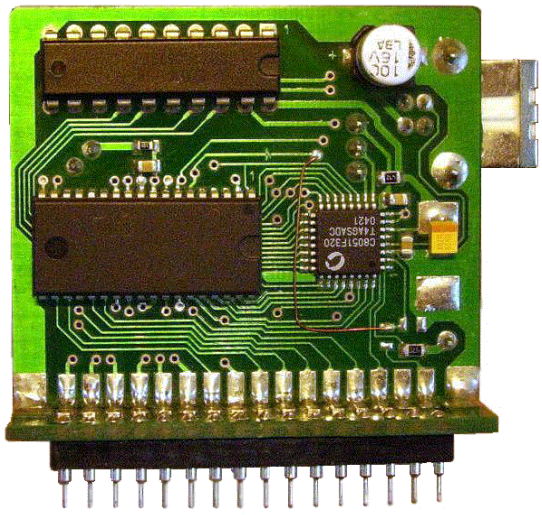


Рисунок 4.3 – Простой эмулятор ПЗУ

Позднее появились эмуляторы ПЗУ, позволяющие отказаться от использования программаторов. Они содержали быстрые электронные переключатели выво-

дов интегральных схем ОЗУ между шинами отлаживаемого МПУ и интерфейсным каналом инструментального компьютера, с которого выполнялась загрузка кода программы. Фотография и функциональная схема эмулятора ПЗУ, устанавливаемого в системную шину ISA персонального компьютера, приведены на рисунках 4.4 и 4.5.

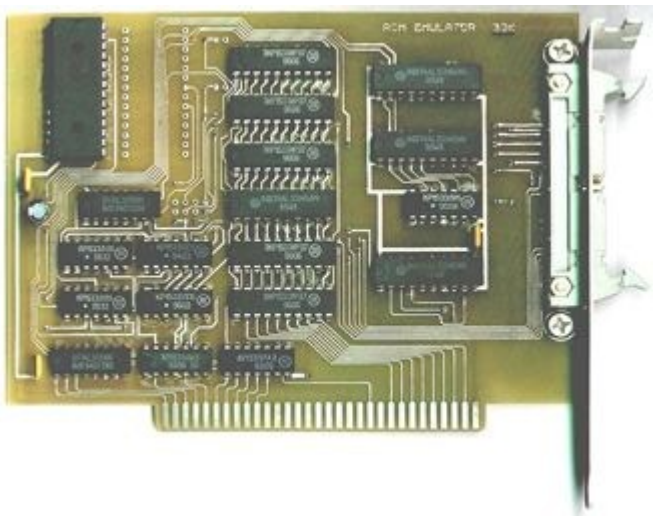


Рисунок 4.4 – Эмулятор ПЗУ

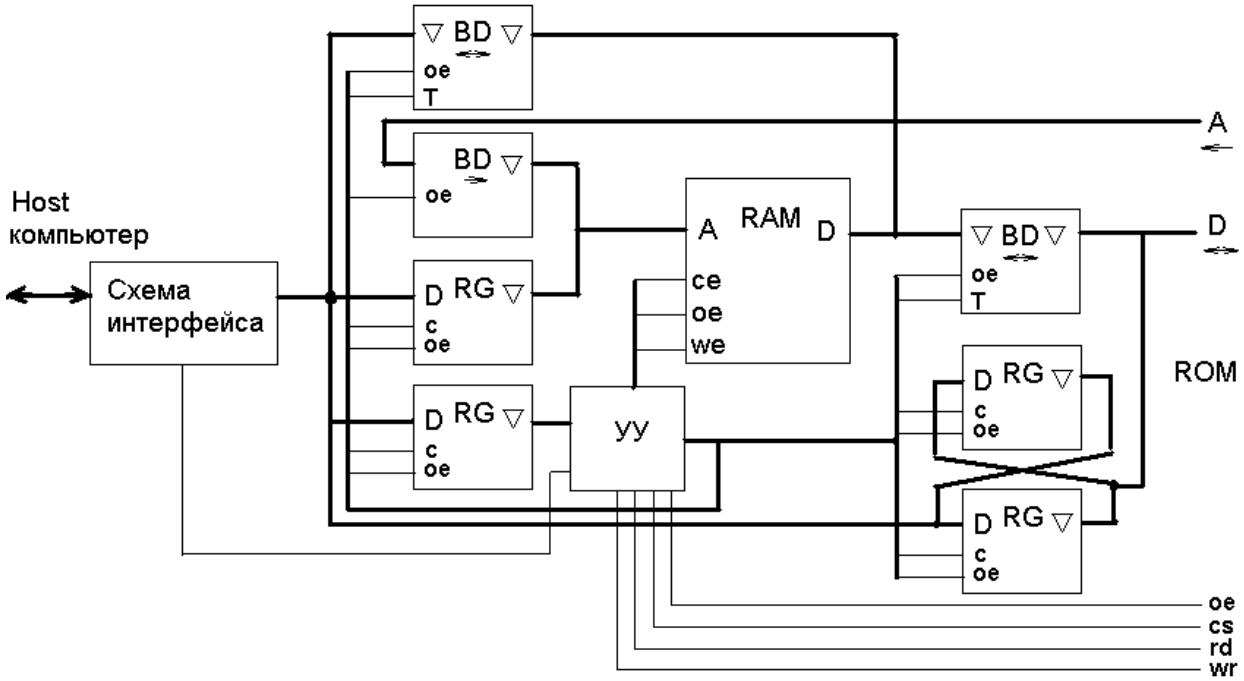


Рисунок 4.5 – Схема эмулятора ПЗУ

Данная конструкция широко использовалась для загрузки программного кода в МК семейства MCS51 с внешней памятью программ. При этом также обеспечивалась реализация некоторых отладочных функций.

4.2 Внутрисхемное программирование

Появление МК со встроенной Flash памятью программ существенно изменило процедуру загрузки программного кода. Так как, для программирования Flash ПЗУ не обязательно использовать программатор, то при перепрограммировании интегральные схемы могут оставаться на своих платах. Программирование МК в среде МПУ, в составе которого он должен работать, получило название *внутрисхемное* программирование.

Современные МК содержат на кристалле все необходимое для их программирования, включая высоковольтные генераторы стирания - программирования, управляющие автоматы для выполнения внешних команд и реализации поллинга. Для краткости этот набор средств будем называть – *встроенный программатор*. Значительная часть функций стирания – программирования выполняется встроенным программатором. Внешний программатор должен лишь формировать необходимую последовательность команд и вместе с данными передавать ее в МК, используя для этого какой-либо интерфейс и соответствующий протокол. Встроенный программатор должен декодировать и выполнять команды, поступающие от внешнего программатора (рисунок 4.6).

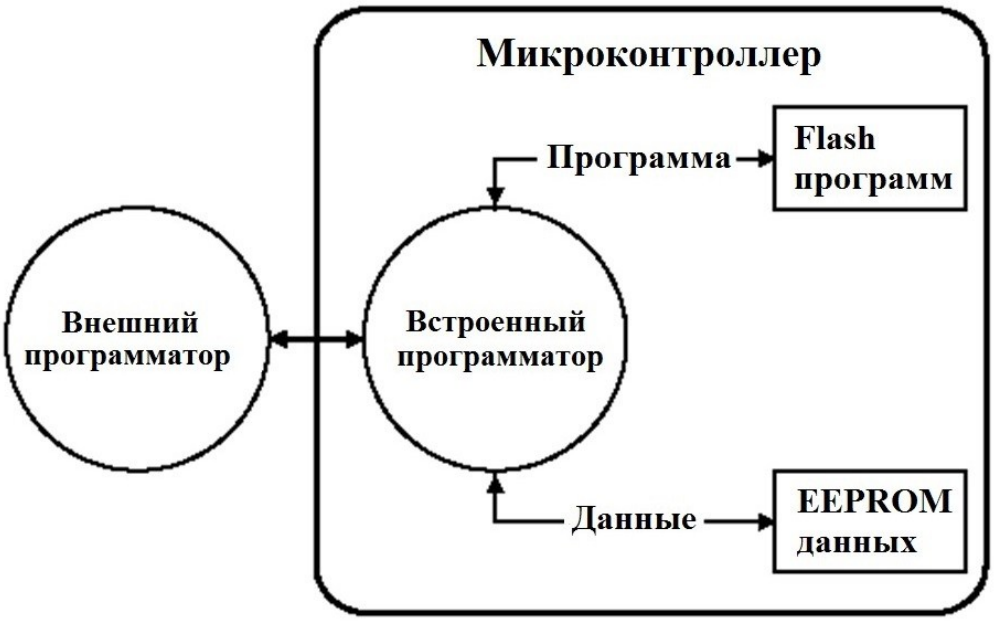


Рисунок 4.6 – Модель внутрисхемного программирования

Рассмотрим реализацию внутрисхемного программирования в МК семейства PIC16. Внутрисхемное программирование доступно для всех моделей МК, память программ которых реализована в виде One-Time-Programmable (OTP) или Flash ПЗУ [10].

Для своих МК Microchip разработала протокол внутрисхемного последовательного программирования - In-Circuit Serial Programming (ICSP). Для семейства PIC16F87x спецификация [11] предусматривает различные алгоритмы ICSP для моделей PIC16LF87x с рабочим диапазоном напряжения питания от 2.2 В до 5.5 В (алгоритм 1) и моделей с рабочим диапазоном напряжения питания от 4.5 В до 5.5 В (алгоритм 2). Для каждого алгоритма доступны методы высоковольтного и низковольтного программирования. Эти методы отличаются способом перевода МК в режим программирования. При использовании высоковольтного программирования для перевода МК в режим программирования необходимо на вывод #MCLR подать потенциал V_{DDP} (13 ± 0.5) В при удержании на линиях RB6 и RB7 низкого потенциала. При использовании низковольтного программирования (LVP) для перевода МК в режим программирования необходимо на вывод #MCLR подать потенциал питания V_{DD} , потенциал на выводе RB3 изменить от нуля до V_{DD}. Разрешить/запретить использование низковольтного программирования можно путем установки/сброса разряда LVP в слове конфигурации (Configuration word).

В режиме программирования доступно программирование Flash ROM программ, EEPROM данных и ячеек памяти конфигурации. Эти области памяти для внешнего программатора отображаются в общем адресном пространстве, как показано на рисунке 4.7. Ячейки Flash ROM (до 8K) располагаются по адресам 0000h – 1FFFh. Для памяти конфигурации выделены адреса 2000h – 200Fh, однако физически реализованы только ячейки 2000h – 2007h. Ячейки EEPROM данных (до 256) располагаются по адресам 2100h – 21FFh.

Ячейки с адресами в диапазоне 0x2000 – 0x2003 предназначены для размещения кода идентификации (ID). Рекомендуется использовать только четыре младших разряда каждой из этих ячеек, так как для некоторых моделей МК после установки защиты кода программ они читаются в расшифрованном виде. Ячейка с

адресом 0x2006 выделена для хранения кода идентификации кристалла (Device ID), по которому внешнее устройство может определить тип модели МК.

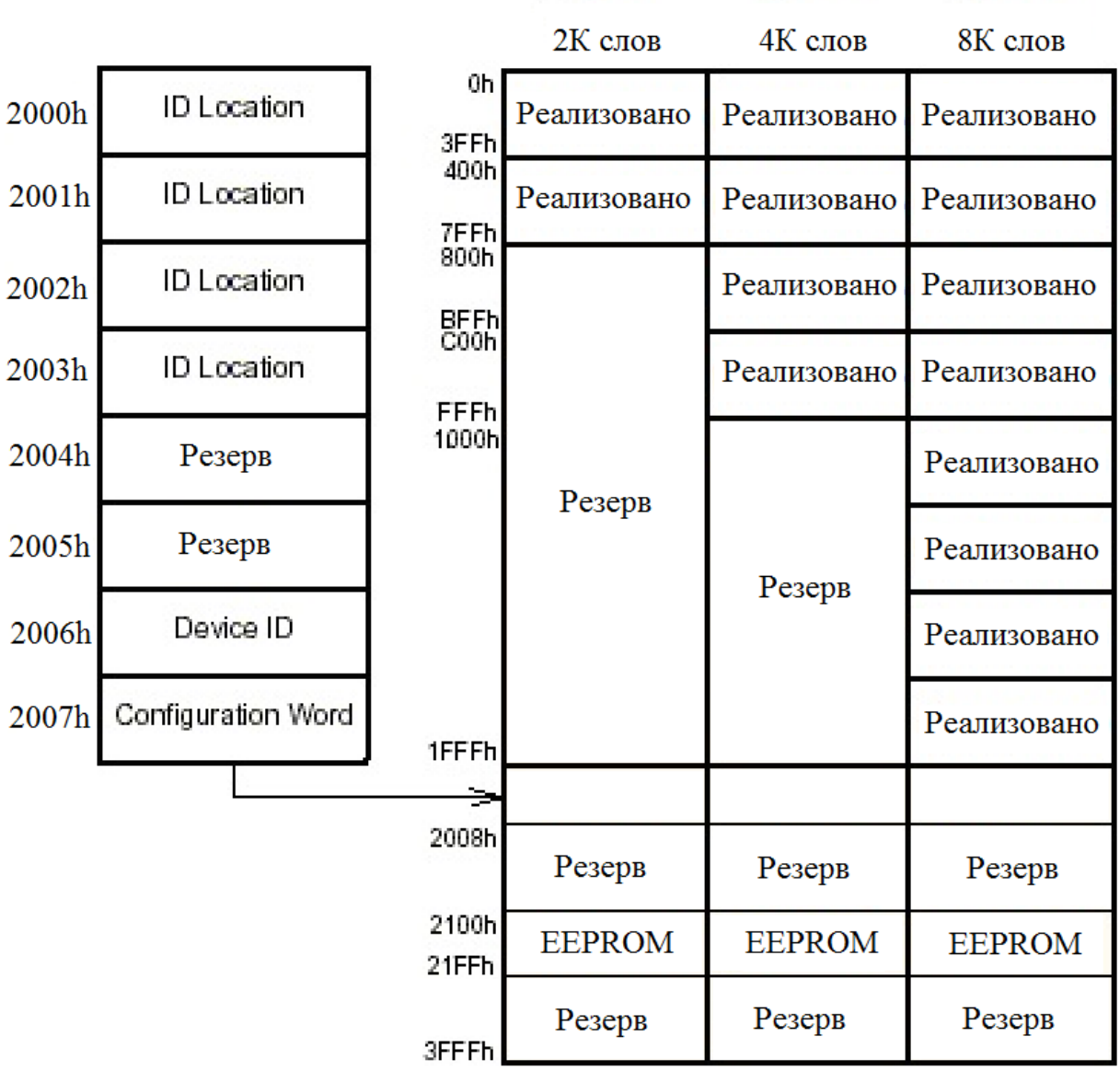


Рисунок 4.7 – Адресное пространство программатора

Для передачи команд от внешнего программатора и обмена данными используется последовательный синхронный полудуплексный интерфейс. После передачи команды чтения / загрузки через паузу не менее 1 мкс следуют данные. Каждый разряд передаваемой по линии вывода RB7 команды или данных сопровождается синхроимпульсом на линии RB6. Прием разрядов команд / данных происходит по отрицательному фронту синхроимпульса. Все команды и данные передаются младшими разрядами вперед. Набор команд приведен в таблице 4.1.

Таблица 4.1 – Команды внутрисхемного программирования

Команда	Код	Данные	Диапазон V_{DD}, В
Load Configuration (загрузка конфигурации)	x x 0 0 0 0	0, данные(14), 0	2.2 – 5.5
Load Data for Program Memory (загрузка памяти программ)	x x 0 0 1 0	0, данные(14), 0	2.2 – 5.5
Read Data from Program Memory (чтение памяти программ)	x x 0 1 0 0	0, данные(14), 0	2.2 – 5.5
Increment Address (инкремент адреса)	x x 0 1 1 0		2.2 – 5.5
Begin Erase Programming Cycle (начало цикла стирания – программирования)	0 0 1 0 0 0		2.2 – 5.5
Begin Programming Only Cycle (начало цикла программирования)	0 1 1 0 0 0		4.5 – 5.5
Load Data for Data Memory (загрузка памяти данных)	x x 0 0 1 1	0, данные(14), 0	2.2 – 5.5
Read Data from Data Memory (чтение памяти данных)	x x 0 1 0 1	0, данные(14), 0	2.2 – 5.5
Bulk Erase Setup 1 (стирание накопителя 1)	0 0 0 0 0 1		4.5 – 5.5
Bulk Erase Setup 2 (стирание накопителя 2)	0 0 0 1 1 1		4.5 – 5.5

Загрузка конфигурации. После получения этой команды программный счетчик устанавливается на адрес 0x2007. Принятое слово конфигурации программируется в память конфигурации. После этого существует только один путь вернуться к памяти программ – выйти из режима программирования / верификации путем подачи низкого потенциала на линию #MCLR.

Загрузка в память программ. После получения этой команды встроенный программатор будет программировать в очередную ячейку памяти программ принятый 14 разрядный код. Временная диаграмма передачи команды приведена на рисунке 4.8.

Загрузка в память данных. После получения этой команды встроенный программатор будет программировать в очередную ячейку памяти данных 8 младших разрядов принятого 14 разрядного кода.

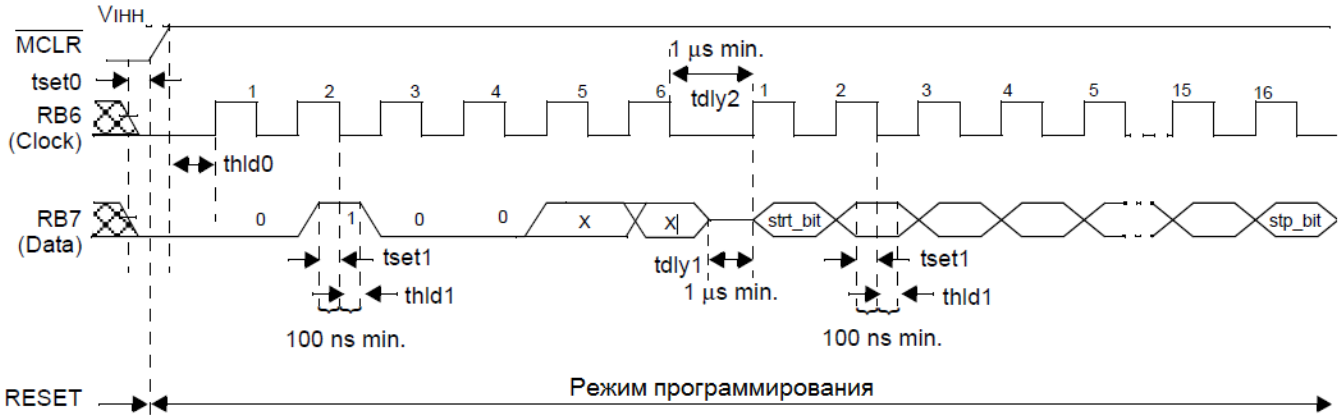


Рисунок 4.8 – Команда загрузки в память программ

Чтение памяти программ. После получения этой команды встроенный программатор будет считывать и передавать 14 разрядный код из очередной ячейки памяти программ. Линия RB7 после паузы 1 мкс в момент формирования второго положительного фронта синхроимпульса переключается в режим выхода. Временная диаграмма приведена на рисунке 4.9.

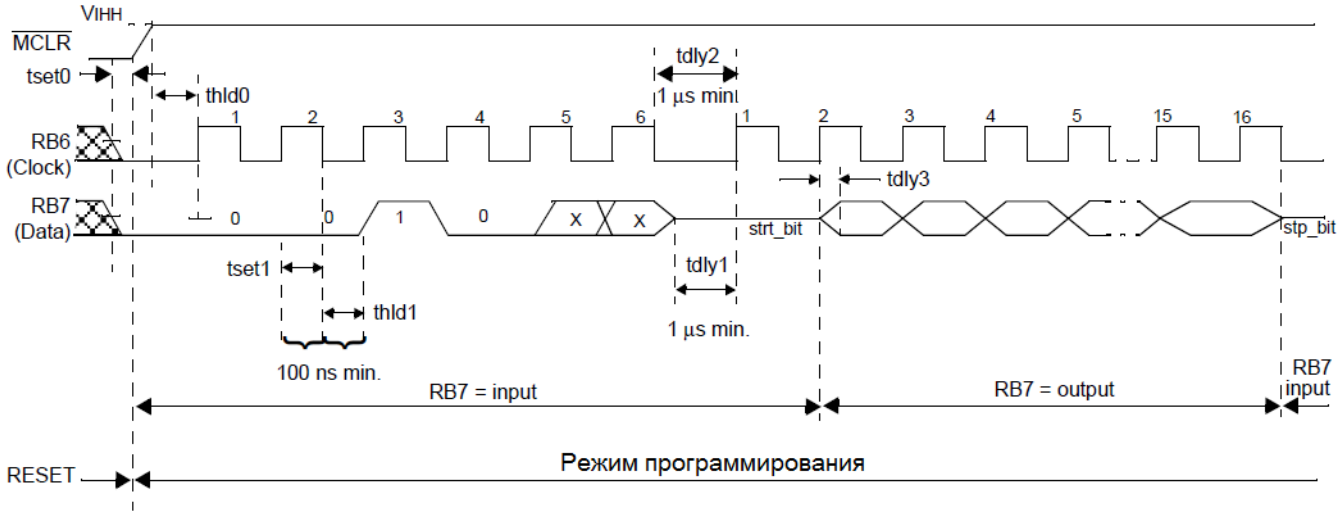


Рисунок 4.9 – Команда чтения памяти программ

Чтение памяти данных. После получения этой команды встроенный программатор будет читать и передавать 14 разрядный код. При этом только первые 8 разрядов являются актуальными данными из очередной ячейки памяти данных.

Инкремент адреса. Содержимое программного счетчика МК инкрементируется. Временная диаграмма приведена на рисунке 4.10.

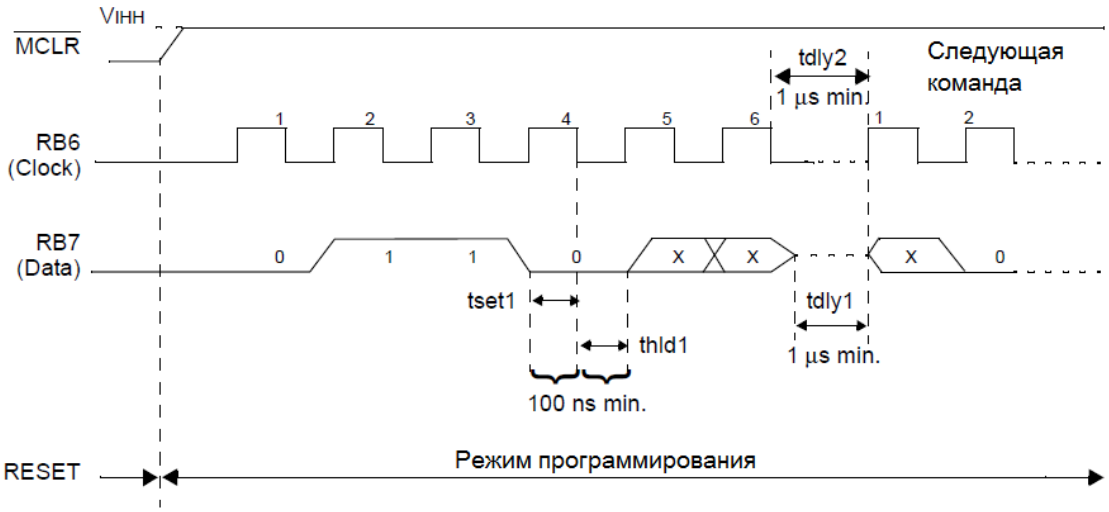


Рисунок 4.10 – Команда инкремента адреса

Начало цикла стирания-программирования. Эта команда должна подаваться перед каждой операцией программирования. Программирование выбранной области памяти будет начинаться после получения и декодирования команды. Модуль тайминга встроенного программатора выполняет стирание перед программированием. Внешний программатор должен обеспечивать ожидание завершения циклов стирания и программирования.

Начало программирования. Команда подобна команде начала цикла стирания-программирования за исключением того, что стирание ячейки не выполняется. Предполагается, что перед серией команд начало программирования выполняется команда стирание накопителя памяти.

Необходимую последовательность команд формирует внешний программатор, функции которого выполняет аппаратно-программный комплекс, состоящий из инструментального компьютера, программного модуля Programmer системы MPLAB IDE, а также специального модуля, например ICD2. Так как этот модуль используется также и для внутрисхемной отладки, то он получил название внутрисхемный отладчик-программатор. Принцип действия ICD2 будет рассмотрен в подразделе 5.

На рисунке 4.11 представлен алгоритм высоковольтного программирования памяти программ (или данных) моделей PIC16LF87x с рабочим напряжением питания от 2.2 В до 5.5 В (алгоритм 1). В алгоритме 1 для перепрограммирования од-

ной ячейки отдается команда начала цикла стирания – программирования, и эти операции последовательно выполняются для одной ячейки по текущему адресу. Команда инкремента адреса подается после паузы, рассчитанной на завершение этих операций стирания и программирования.

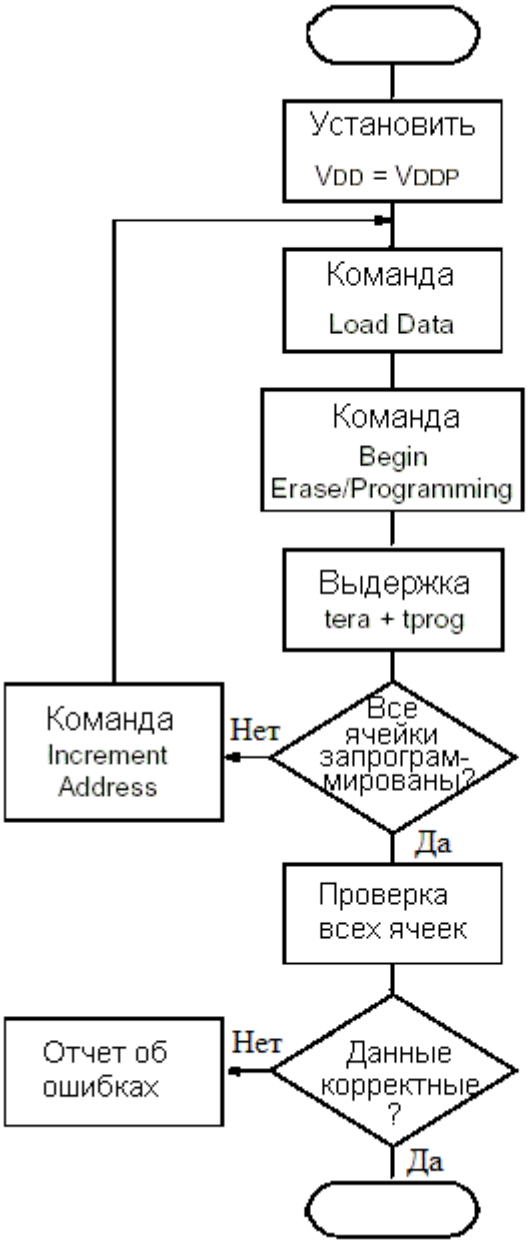


Рисунок 4.11 - Алгоритм 1

На рисунке 4.12 представлен алгоритм высоковольтного программирования памяти программ (или данных) моделей PIC16F87x с рабочим напряжением питания от 4.5 В до 5.5 В (алгоритм 2). В алгоритме 2 сначала отдается команда начала цикла стирания всего накопителя, а затем в цикле подается последовательность

команд программирования одной ячейки и инкремента адреса. Команда инкремента адреса подается после паузы, рассчитанной на завершение операции программирования ячейки.

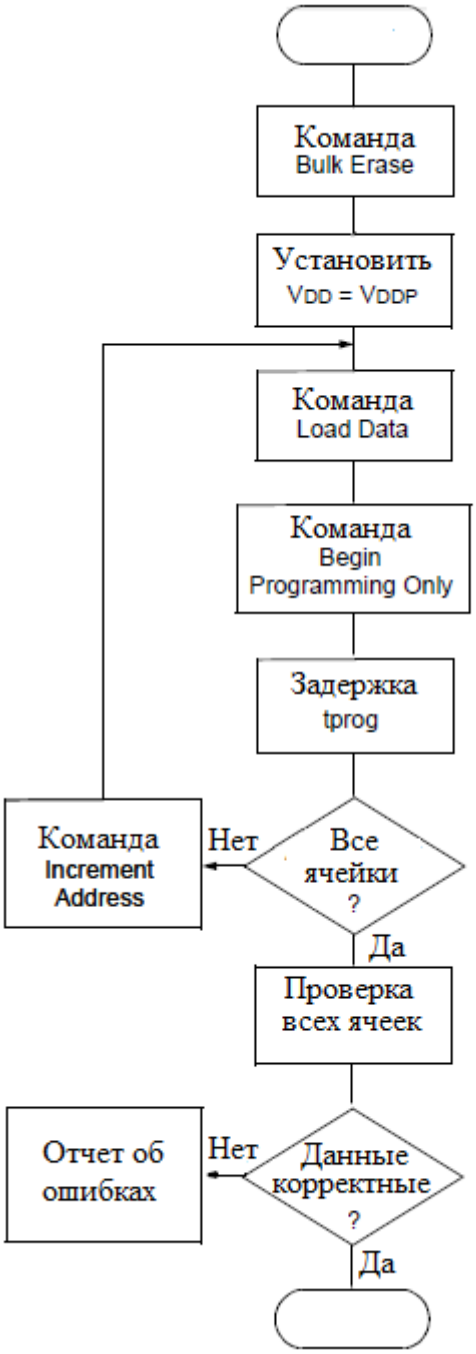


Рисунок 4.12 – Алгоритм 2

В зависимости от состояния разрядов защиты кода в слове конфигурации стирание памяти программ и памяти данных будет выполняться различными процедурами.

Когда защита памяти программ и данных не установлена, они должны стираться отдельно, используя следующие процедуры. Если установлена защита или памяти программ, или памяти данных, то стирание обоих накопителей выполняется одной процедурой. Эти процедуры не стирают слово конфигурации и коды идентификации.

Стирание накопителя памяти программ:

1) выполнить команду загрузки памяти программ с «1» во всех разрядах (0x3FFF);

2) выполнить команду стирание накопителя 1;

3) выполнить команду стирание накопителя 2;

4) выполнить команду начало цикла стирания – программирования;

5) задержка 8 мс;

6) выполнить команду стирание накопителя 1;

7) выполнить команду стирание накопителя 2.

Стирание накопителя памяти данных:

1) выполнить команду загрузки памяти данных с «1» во всех разрядах (0x3FFF);

2) выполнить команду стирание накопителя 1;

3) выполнить команду стирание накопителя 2;

4) выполнить команду начало цикла стирания – программирования;

5) задержка 8 мс;

6) выполнить команду стирание накопителя 1;

7) выполнить команду стирание накопителя 2.

Если защита кода была установлена, все защищенные ячейки памяти программ и памяти данных читаются как «0», и дальнейшее программирование запрещено. Ячейки кода идентификации и слова конфигурации читаются декодированными и могут репрограммироваться нормально. В этом случае используется следующая процедура стирания:

1) выполнить команду загрузки конфигурации с «1» во всех разрядах (0x3FFF);

2) выполнить команду инкремента адреса, чтобы установить адрес ячейки слова конфигурации (0x2007);

3) выполнить команду стирание накопителя 1;

4) выполнить команду стирание накопителя 2;

5) выполнить команду начало цикла стирания – программирования;

6) задержка 8 мс;

7) выполнить команду стирание накопителя 1;

8) выполнить команду стирание накопителя 2.

Для внешнего программатора оно находится в адресном пространстве памяти программ, для МК семейства PIC16F87x - по адресу 0x2007. Значения разрядов слова конфигурации можно задать директивой __CONFIG в исходном тексте программы. После трансляции их значения можно наблюдать и редактировать в окне *Configure> Configuration Bits* системы MPLAB IDE. Требуемые значения разрядов для обеспечения внутрисхемной отладки приведены в таблице 4.2.

Таблица 4.2 - Разряды слова конфигурации

Пункт	Опция
Oscillator	Режим тактового генератора, например, кварцевый резонатор до 4 МГц (XT)
Watchdog Timer	Сторожевой таймер (WDT) должен быть выключен (Off)
Power Up Timer	Таймер включения (PWRT) должен быть включен (On).
Brown Out Detect	Детектор дрожания напряжения питания (BOD) должен быть выключен (Off)
Low Voltage Program	Низковольтное программирование должно быть запрещено (Disabled)
Flash Memory Write	Запись Flash Memory должна быть разрешена (Enabled)
Data EE Read Protect	Защита данных EEPROM должна быть выключена (Off)
Code Protect	Защита кода программы должна быть выключена (Off)

Соответствующие параметры директивы ассемблера:

```
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_ OFF & _PWRTE_ON
& _XT_OSC & _WRT_ENABLE_ON & _LVP_OFF & _DEBUG_ON &
_CPD_OFF
```

4.3 Вопросы для самоконтроля к разделу 4

Как физически выполняется память программ МК?

Каким образом выполняется загрузка кода программы в память программ МК?

Поясните функциональное назначение программатора ПЗУ при разработке МПУ.

Поясните смысл термина – внутрисхемное программирование.

Какие существуют ограничения на использование внутрисхемного программирования?

Поясните, чем отличаются методы высоковольтного и низковольтного программирования МК семейств PIC16.

Поясните функции встроенного и внешнего программатора при внутрисхемном программировании.

Поясните работу интерфейса ICSP.

Перечислите команды протокола ICSP.

Как отличаются алгоритмы внутрисхемного программирования МК семейств PIC16 с различными диапазонами рабочих напряжений?

5 Средства комплексной отладки

5.1 Внутрисхемные эмуляторы

Внутрисхемный эмулятор – программно-аппаратное средство, способное замещать собой целевой процессор в МПУ, обеспечивая при этом выполнение условий контролепригодности. Внутрисхемный эмулятор - это наиболее мощное и универсальное отладочное средство.

Обычно стыковка внутрисхемного эмулятора с отлаживаемой системой производится при помощи эмуляционного кабеля со специальным переходным разъемом. Эмуляционный разъем устанавливается вместо целевого МП или МК в панельку на плате отлаживаемой системы (рисунок 5.1). Если МК невозможно удалить из отлаживаемой системы, то использование эмулятора возможно, только если этот МК имеет отладочный режим, при котором все его выводы находятся в третьем состоянии. В этом случае для подключения эмулятора используют специальный адаптер-клипсу, который подключается непосредственно к выводам эмулируемого МК.

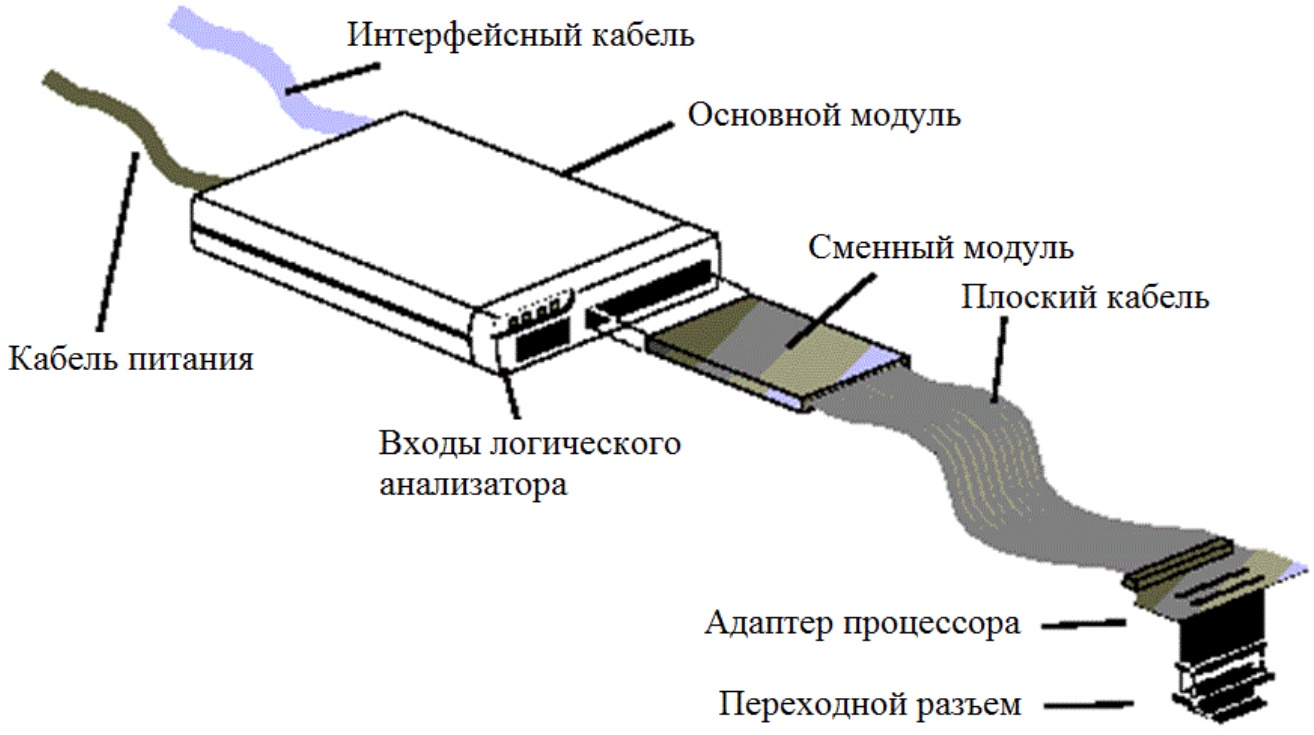


Рисунок 5.1 – Подключение внутрисхемного эмулятора

Для повышения универсальности эмуляторы, как правило, разрабатываются не под конкретную модель МК, а под определенный набор семейств и состоят из основного модуля эмулятора, сменных процессорных модулей (обычно поддерживают какое-либо семейство МК) и сменных адаптеров (выбор определяется количество портов ввода-вывода и набором периферийных устройств). В некоторых случаях еще необходим переходной разъем для конкретного типа корпуса целевого МК. Таким образом, эмуляционная платформа состоит из нескольких модулей и для каждой поддерживаемой модели МК необходимо подбирать определенный набор модулей.

Основой внутрисхемного эмулятора является эмуляционный МК, который является, по сути, специализированным МК, способным заменить определенное множество моделей МК одного семейства (рисунок 5.2). В отличие от серийных моделей МК эмуляционный кристалл имеет дополнительные выводы для доступа к системным шинам и процессорному ядру. При отладке с помощью эмулятора программа выполняется из внешнего ОЗУ и ход программы полностью контролируется эмулятором под управлением среды разработки.

Эмуляционные МК вместе со схемой сопряжения и тактирования располагаются в сменных модулях. В основном модуле располагаются остальные функциональные узлы:

- память программ на основе статического ОЗУ;
- память трассы выполнения программы;
- таймер для измерения времени выполнения участков программы;
- блок точек останова для хранения адресов точек останова и условий останова;
- устройство управления, связанное интерфейсным кабелем с инструментальным компьютером.

Внутрисхемный эмулятор работает под управлением отладчика интегрированной среды разработки. При этом разработчику МПУ доступны все стандартные отладочные функции.

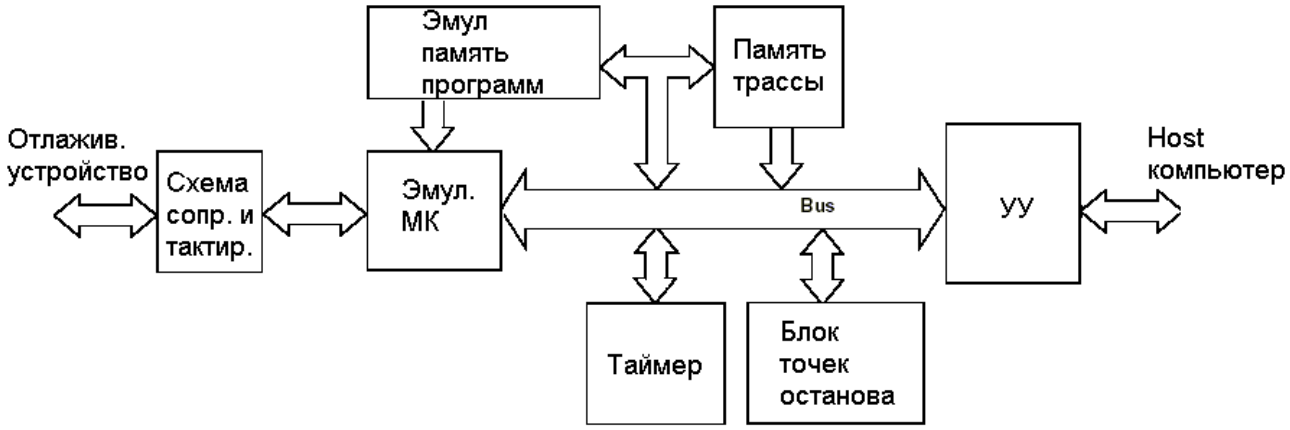


Рисунок 5.2 – Схема внутрисхемного эмулятора

Внутрисхемные эмуляторы обеспечивают разработчика практически всеми необходимыми инструментами для эффективной отладки программ. К примеру, эмулятор MPLAB ICE 2000 от компании Microchip (рисунок 5.3) предоставляет следующие возможности:

- максимальная тактовая частота эмулируемого контроллера до 25МГц;

- до 65535 точек останова;

- использование до 8 логических пробников;

- буфер трассировки до 32Кб с сохранением в реальном времени лога адресов выполняемых команд, обращения к регистрам и памяти данных, а также состояний логических пробников;

- триггеры и точки останова могут быть настроены на единичное событие, множество событий или последовательность событий.

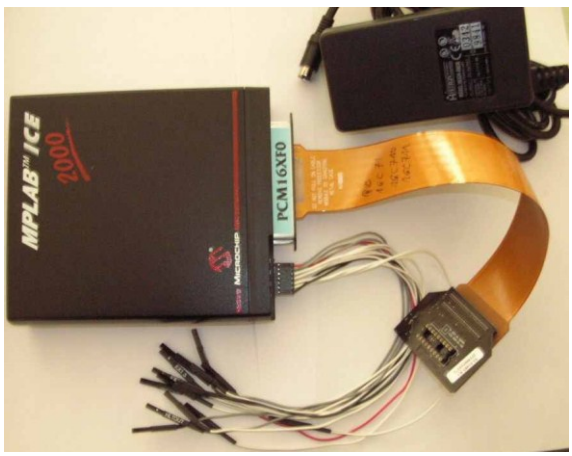


Рисунок 5.3 – Внутрисхемный эмулятор MPLAB ICE 2000

На рисунке 5.4 приведена фотография адаптера DA16C5x, поддерживающего МК семейства PIC16C5x.

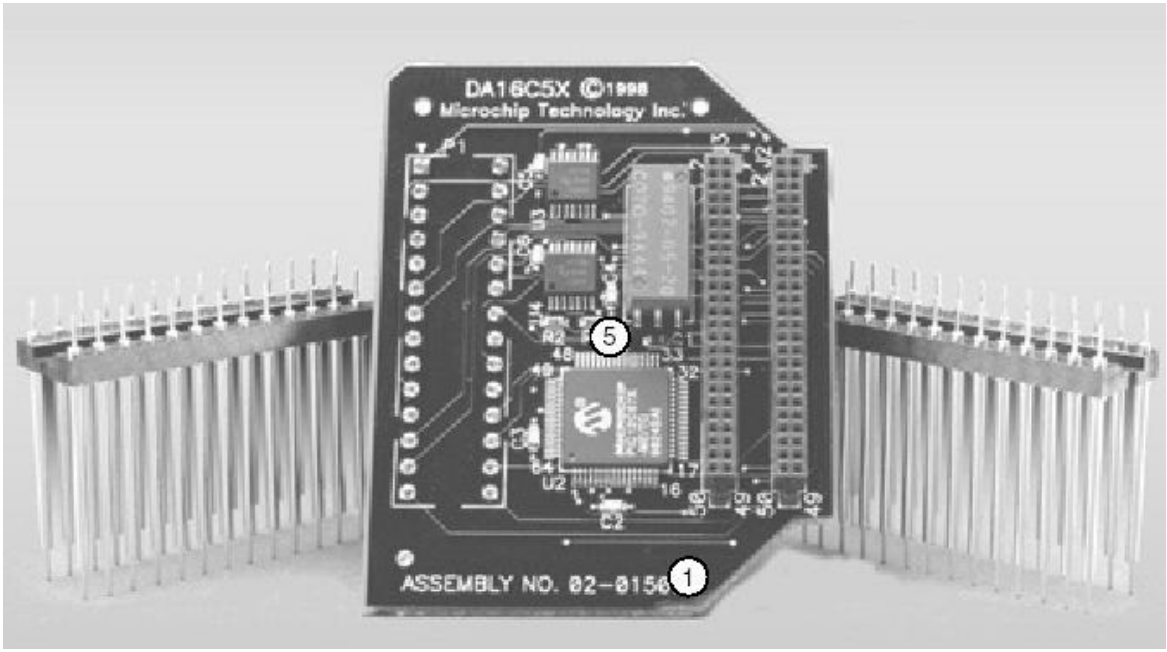


Рисунок 5.4 - Адаптер DA16C5x

Внутрисхемные эмуляторы, несмотря на очевидные достоинства, имеют и очевидные недостатки:

- высокая стоимость специализированных эмуляционных кристаллов;

- эмуляционный кристалл не является точной копией целевого МК, поэтому возможны несоответствия в работе эмулятора и целевого МК;

- эмуляционный кристалл располагается удаленно от платы МПУ, поэтому задержки сигналов в проводниках ограничивают предельную тактовую частоту МУ на уровне 25 – 50 МГц.

Эти недостатки являются причиной отказа ведущих производителей МК от использования внутрисхемных эмуляторов, как основного средства комплексной отладки. В настоящее время внутрисхемные эмуляторы можно рассматривать, как часть истории развития микропроцессорной техники. Тем не менее, высокая репутация внутрисхемных эмуляторов у разработчиков МПУ толкает производителей средств комплексной отладки на использование аббревиатуры ICE (In-Circuit Emulator) для наименования современных инструментальных средств отладки, не использующих эмуляционных кристаллов.

5.2 Встроенные средства отладки МК PIC16F

Начало применения Flash ПЗУ в качестве памяти программ МК совпало с появлением встроенных средств отладки. Совершенствование полупроводниковой технологии позволило производителям интегральных схем МК размещать на кристаллах не только встроенные средства для программирования Flash и EEPROM, но и встроенные средства отладки, обеспечивающие выполнение условий контролепригодности. При этом, как правило, для этих встроенных средств используются общие интерфейсные каналы для связи с внешним оборудованием. По этой причине обычно используется один интерфейсный канал и общее внешнее оборудование, необходимое для выполнения внутрисхемного программирования и внутрисхемной отладки. В настоящее время разработчикам МПУ доступны разнообразные устройства от простых внутрисхемных отладчиков – программаторов до более дорогих и сложных устройств, которые часто продолжают называть внутрисхемными эмуляторами. Но важно понимать, что возможности этих средств могут быть реализованы только при наличии соответствующей поддержки на кристаллах МК. Весь набор средств, включая отладчик интегрированной среды разработки, внутрисхемные программаторы – отладчики (эмуляторы) для краткости будем называть внешним отладчиком. В этом подразделе основное внимание уделим достаточно простой и эффективной реализации встроенных средств отладки МК PIC16F - модуля теневой отладки – Background Debug Module (BDM) [12]. Расположение BDM на кристалле PIC16F877 показано на рисунке 5.5.

Встроенный модуль отладки МК семейства PIC16 обеспечивает минимальный набор отладочных функций:

- возможность инициировать и останавливать процесс исполнения целевой программы по команде от внешнего отладчика;

- обеспечивать возможность исполнения целевой программы по шагам;

- останавливать процесс исполнения целевой программы при достижении точки останова;

- выполнять чтение (модификацию) регистров специальных функций и регистров общего назначения, других ячеек памяти целевого МК.

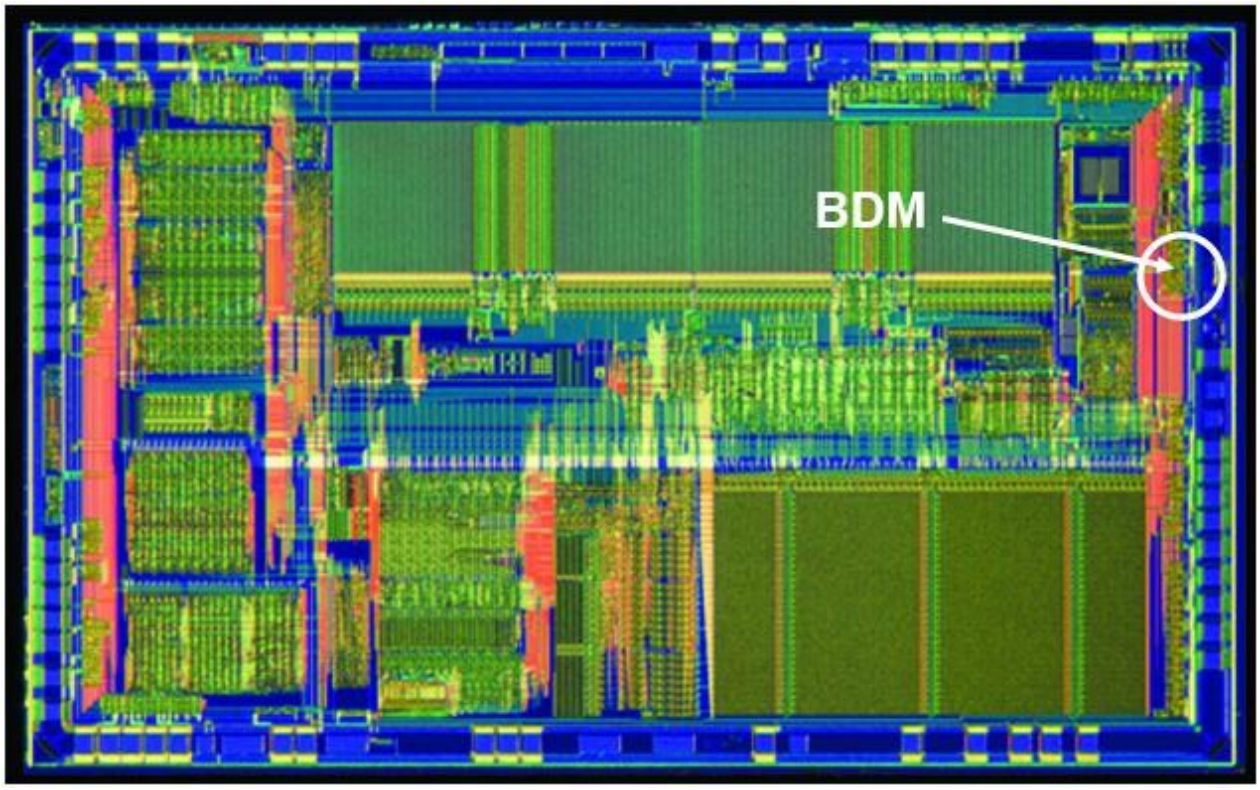


Рисунок 5.5 – План кристалла PIC16F877

Для управления узлами BDM используются слово конфигурации и регистры отладочных функций, представленные на рисунке 5.6. Работа BDM разрешена, если в разряд 11 (DEBUG) слова конфигурации Config сброшен в ‘0’. Запись слова конфигурации выполняется при программировании МК. Если программирование МК выполняется из меню Programmer системы MPLAB IDE, в разряд DEBUG записывается ‘1’. Если программирование выполняется из меню Debugger системы MPLAB IDE, в разряд DEBUG записывается ‘0’, при этом в память программ наряду с отлаживаемым приложением записывается код обработчика отладочного прерывания - Debug Executive (DE).

Слово конфиурации - Config (адрес 2007h)

CP1	CP0	DEBUG	-	WRT	CPD	LVP	BODEN	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
bit13													bit0

Рисунок 5.6 – Слово конфигурации

В этом случае в памяти МК находится код двух программ:

- целевой программы (отлаживаемого приложения);

- обработчик отладочного прерывания DE.

Одновременно МК может выполнять одну из этих программ. Модель внутрисхемной отладки приведена на рисунке 5.7. При выполнении обработчика отладочного прерывания DE, целевой процесс остановлен, но обеспечивается работа инструментального интерфейса с внешним отладчиком.

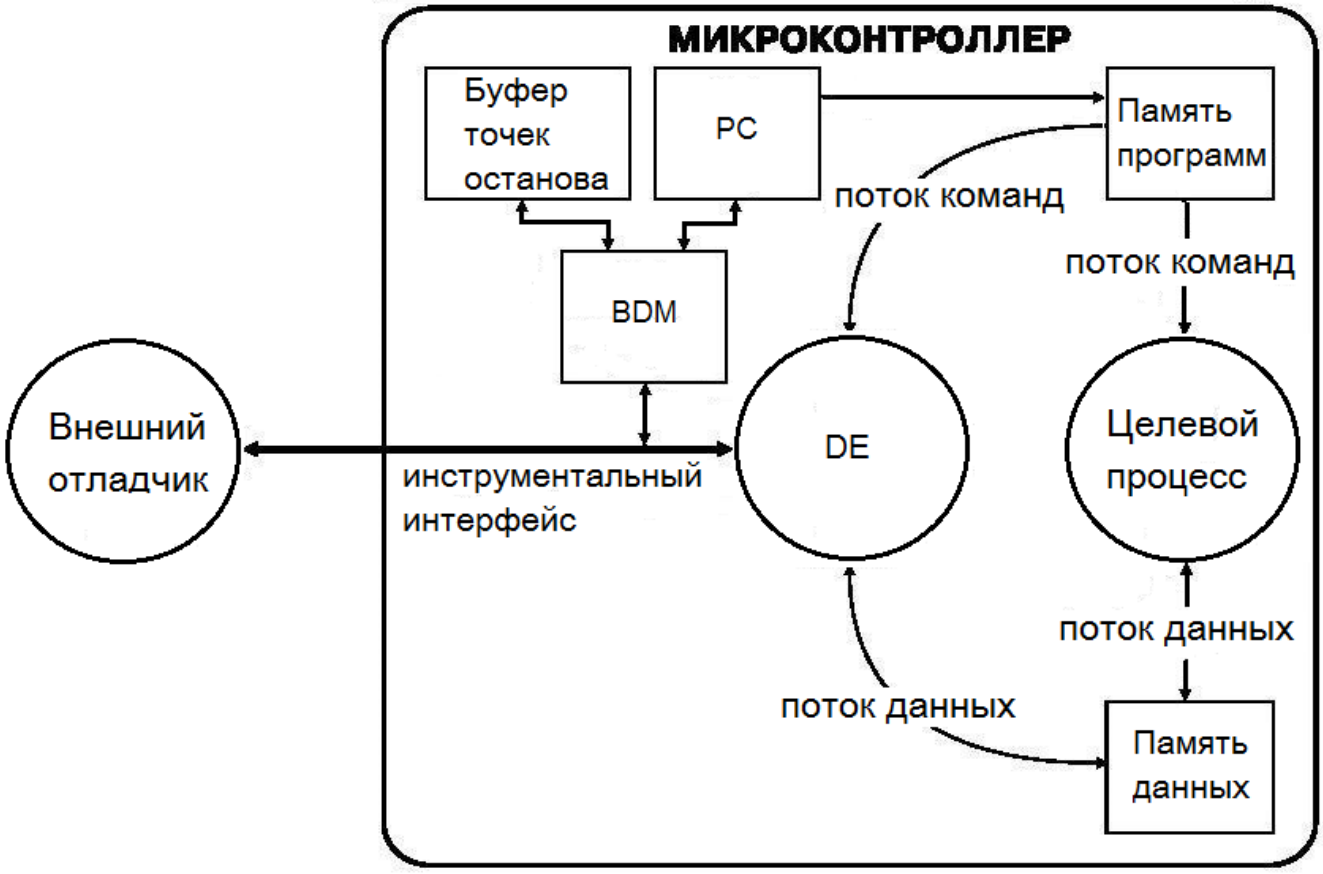


Рисунок 5.7 – Модель внутрисхемной отладки

Для МК семейства PIC16F87x обработчик отладочного прерывания DE загружается в ячейки памяти программ по адресам в диапазоне 0x1F00 ...0x1FFF. Поэтому эта область памяти программ не должна использоваться отлаживаемым приложением. Кроме этого, не должны использоваться ячейки памяти данных по адресам 0x70, 0x1EB... 0x1EF – они также используются обработчиком отладочного прерывания. Для функций внутрисхемной отладки также резервируется один уровень стека – для хранения адреса точки возврата из отладочного прерывания и ли-

нии порта RB6, RB7, используемые для организации синхронного последовательного полудуплексного интерфейса с внешним внутрисхемным отладчиком-программатором In Circuit Debugger (ICDx). Линия RB6 (ICDC) задействована для тактирования, линия RB7 (ICDD) – для обмена данными.

Таким образом, МК PIC16 может находиться в одном из четырех режимов:

- RESET (сброс) при наличии на линии #MCLR низкого потенциала Vil;

- ICSP (внутрисхемное программирование): при наличии на линии #MCLR потенциала высоковольтного программирования Vpp;

- USER (пользовательский) – выполняет код отлаживаемой программы;

- INBUG (отладочный режим) – выполняет код DE.

Микроконтроллер работает в режимах USER и INBUG при наличии на линии #MCLR потенциала питания Vdd. Пока МК выполняет код обработчика отладочного прерывания (режим INBUG), он может получать от внешнего отладчика команды:

- Read/Write - чтение/запись регистра;

- Step - выполнение одиночного шага целевой программы;

- Run – передача управления отлаживаемой программе (переход в пользовательский режим USER).

Для управления узлами BDM используются регистры отладочных функций, представленные на рисунке 5.8. Состояние разряда INBUG регистра IckBug (адрес 18Eh) определяет режим работы – пользовательский или отладочный. Состояние разряда FREEZ определяет работу периферийных модулей МК в отладочном режиме INBUG. Если разряд FREEZ установлен, то работа периферийных модулей останавливается.

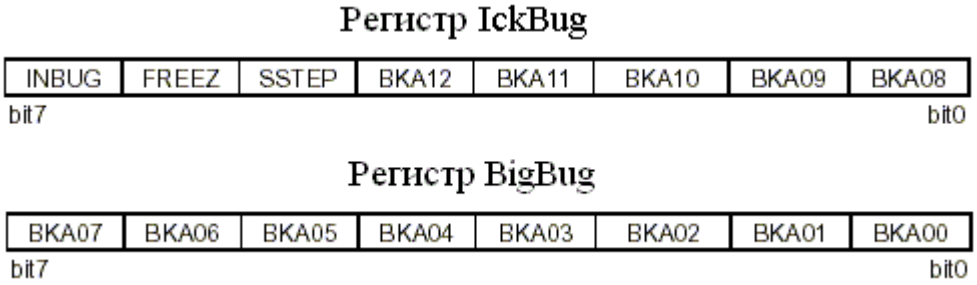


Рисунок 5.8 – Регистры отладочных функций

При поступлении команды *Step*, устанавливается разряд SSTEP регистра IckBug, МК переходит в режим USER (пользовательский), но выполняет только одну инструкцию отлаживаемого приложения, после чего переходит в режим IN-BUG.

В пользовательском режиме модуль BDM не может принимать и выполнять команды от внешнего отладчика. Поэтому все переходы в отладочный режим происходят по специальному немаскируемому прерыванию, недоступному в отлаживаемой программе и возникающему при следующих отладочных событиях:

- останов по команде Halt;
- достижение точки останова;
- пошаговое выполнение программы.

На рисунке 5.9 приведены временные диаграммы, поясняющие переход в отладочный режим по команде Halt.

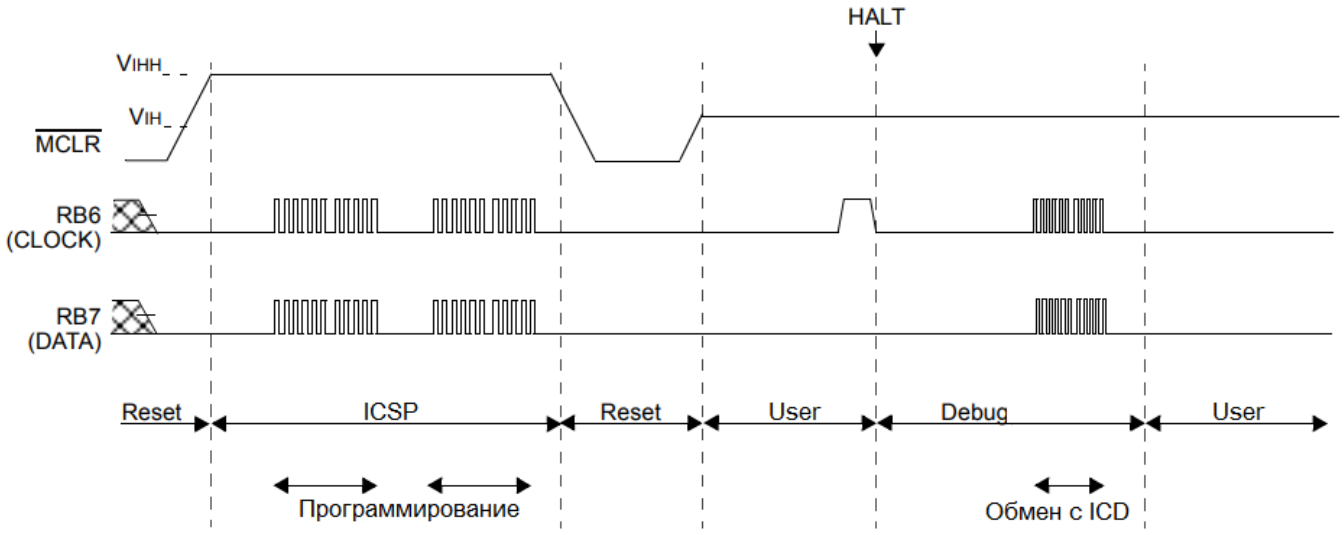


Рисунок 5.9 – Переход в отладочный режим

Внешний отладчик по команде Halt формирует на линии RB6 (ICDC) одиночный импульс, по отрицательному фронту которого устанавливается разряд IN-BUG регистра IckBug, выполняется отладочное прерывание. Адрес точки возврата помещается в стек, в программный счетчик PC загружается вектор отладочного прерывания 0x2004. В ячейке по этому адресу находится код инструкции goto, па-

раметр которой указывает на начальный адрес обработчика отладочного прерывания.

Отладочное прерывание также может произойти при достижении точки останова. Для МК семейств PIC16F доступна только одна простая точка останова (по совпадению адреса инструкции). Дополнительные расширенные точки останова отсутствуют.

В отладочном режиме внешний отладчик устанавливает точку останова путем записи адреса ячейки памяти программ BKA12-BKA0 в разряды 4-0 регистра IckBug и разряды 7-0 регистра BigBug (адрес 18Fh), доступные только в режиме INBUG. Во время работы МК в пользовательском режиме, модуль BDM сравнивает содержимое программного счетчика (текущий адрес памяти программ) с адресом точки останова BKA12-BKA0. При их совпадении генерируется отладочное прерывание.

МК семейства PIC18F имеют до трех простых точек останова и одну расширенную точку останова (по совпадению данных). МК семейства PIC18F Extended имеет три расширенных точки останова. МК семейства PIC18F “J” имеет еще больше возможностей:

- останов по срабатыванию сторожевого таймера;
- остановка по вхождению в режим SLEEP.

5.3 Внутрисхемные отладчики-программаторы ICDx

Для обеспечения взаимодействия BDM с отладчиком системы MPLAB IDE необходимы специальные аппаратные средства поддержки, в качестве которых используются внутрисхемные отладчики - программаторы. Фирма Microchip разработала несколько моделей внутрисхемных отладчиков для своих МК.

Первая модель MPLAB ICD имела ограниченные возможности и предназначалась для работы с МК только семейства PIC16F87x. Поддержка этой модели компанией Microchip прекратилась при разработке версии системы MPLAB IDE 6.0. Поэтому последней версией системы MPLAB IDE, поддерживающей работу

MPLAB ICD, является MPLAB IDE 5.7.40. Принципиальная электрическая схема и монтажная схема MPLAB ICD были опубликованы, также свободно распространялся код firmware – встроенного программного обеспечения для МК PIC16F876, на базе которого был построен MPLAB ICD. Поэтому изготовить самостоятельно такой отладчик могли многие разработчики микропроцессорной техники. Также получили распространение клоны этого устройства, в частности, внутрисхемный отладчик MICD.

Вторая модель MPLAB ICD2 также выполнена на МК PIC16F876, но принципиально отличается тем, что имеет перезагружаемый для конкретной модели целевого МК модуль firmware. Это позволило расширить тип поддерживаемых моделей МК PICmicro. Принципиальным ограничением является наличие Flash памяти программ и поддержка используемой версией системы MPLAB IDE. В данной модели внутрисхемного отладчика реализованы два интерфейса для связи с инструментальным компьютером – RS232 и USB.

Упрощенным аналогом MPLAB ICD2 является разработка отечественного производства MICD2, в этой модели для связи с инструментальным компьютером используется только интерфейс RS232. Данная модель внутрисхемного отладчика используется в лабораторном практикуме дисциплины.

Схема подключения отладчика MICD2 к компьютеру и отлаживаемому микропроцессорному устройству (МПУ) - отладочному стенду приведена на рисунке 5.10. Питание отлаживаемого устройства может осуществляться от отладчика MICD2, который, в свою очередь, получает питание от блока питания (AC-DC адаптера 9 В).

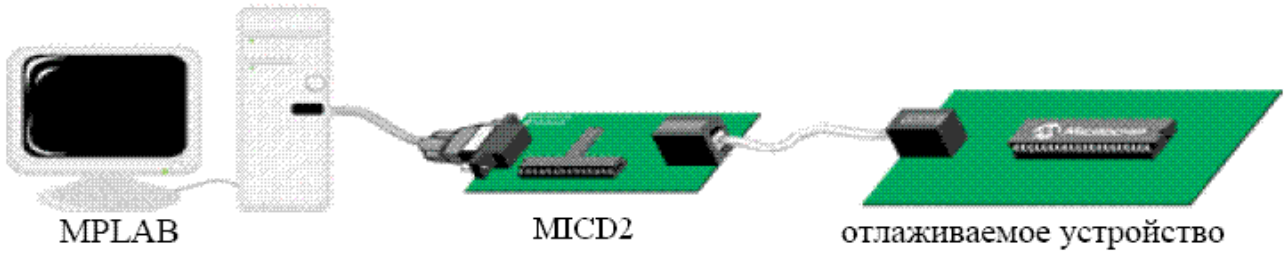


Рисунок 5.10 – Подключение отладчика MICD2

Структурная схема внутрисхемных отладчиков ICD и ICD2 приведена на рисунке 5.11.

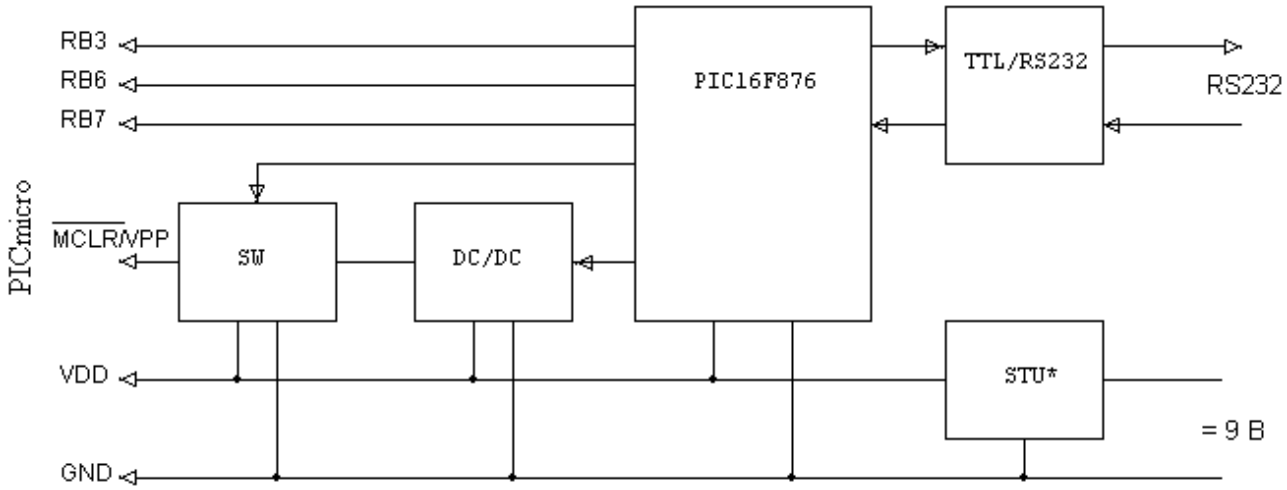


Рисунок 5.11 - Структурная схема внутрисхемных отладчиков ICDx

Протокол обмена внешнего отладчика и встроенного BDM является протоколом типа «ведущий-ведомый»:

- отладчик MPLAB IDE инициирует обмен данными с внутрисхемным отладчиком ICDx;

- внутрисхемный отладчик ICDx инициирует обмен данными с модулем BDM. Реализация более сложных асинхронных протоколов обмена потребовала бы больше пользовательских ресурсов МК.

Основой внутрисхемного отладчик ICDx является МК PIC16F876, непосредственно связанный с линиями отладочного интерфейса RB6 (ICDC) и RB7 (ICDD). Модуль асинхронного приемопередатчика UART МК PIC16F876 через преобразователь уровней связан по каналу RS232 с инструментальным компьютером. Основной функцией внутрисхемного отладчика является реализация канала связи. Это мост между двумя различными последовательными каналами - отладочным интерфейсом МК и интерфейсом RS232 инструментального компьютера.

Второй функцией внутрисхемного отладчик ICDx является управление режимами работы МК по линии MCLR. Аналоговый трехпозиционный ключ позволяет подавать на вывод MCLR потенциал нуля (брос), Vdd (работа в пользовательском или отладочном режиме) или Vpp (режим программирования). Необходимый уровень напряжения Vpp формируется повышающим DC-DC преобразователем.

Повышение быстродействия внутрисхемных отладчиков обеспечивается за счет использования скоростных интерфейсных каналов и сокращение времени на преобразование передаваемых пакетов сообщений. В отладчике – программаторе MPLAB ICD2 был добавлен скоростной интерфейсный канал USB (рисунок 5.12). Аналогичные решения были применены при построении отладчиков – программаторов PicKit2 и PicKit3 (рисунок 5.13).

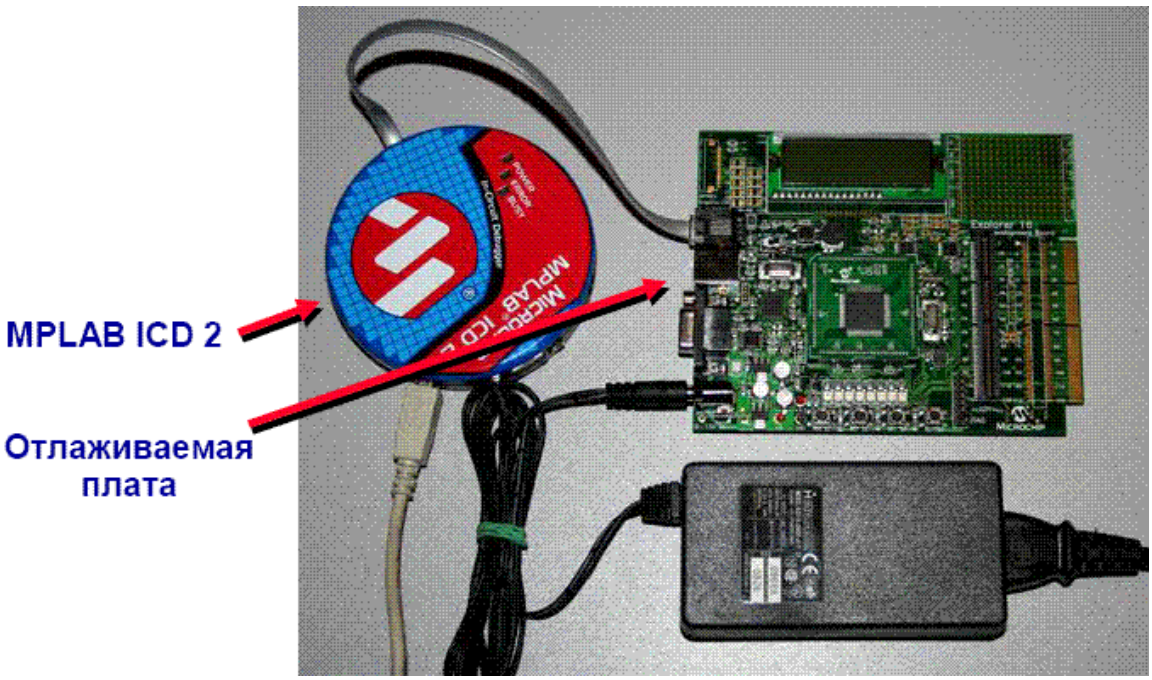


Рисунок 5.12 – MPLAB ICD2

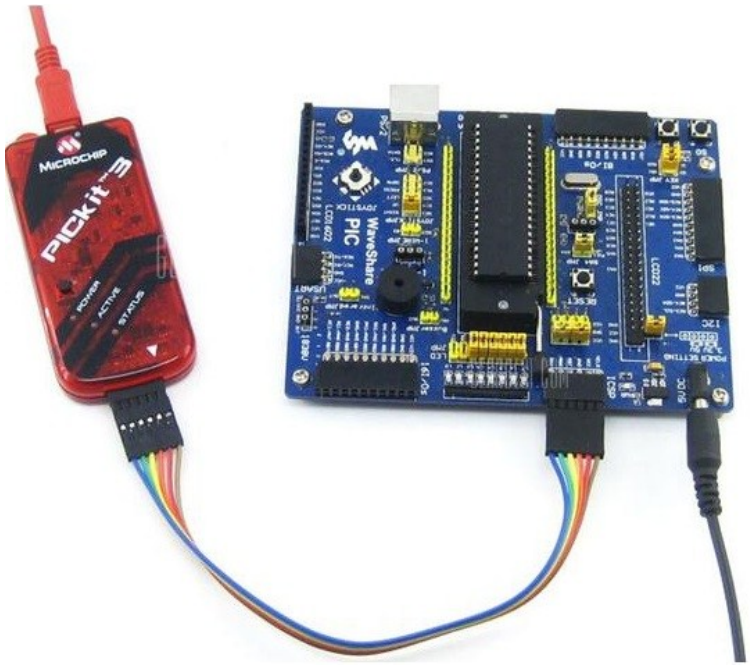


Рисунок 5.13 – PICKit-3

Высокоскоростная модель внутрисхемного отладчика нового поколения MPLAB ICD3 (рисунок 5.14) позволяет программировать и отлаживать все 8-разрядные Flash PIC-micro, а также все линейки 16-и и 32-разрядных МК и цифровых сигнальных контроллеров dsPIC. MPLAB ICD3 обеспечивает скорость программирования Flash и отладки не менее чем в 15 раз более высокую, чем модель ICD2. Такое повышение быстродействия обеспечено за счет реализации интерфейсного моста не на МК, на платформе БИС программируемой логики.

Рисунок 5.14 – ICD-3

Для более производительных и функционально-сложных МК, прежде всего реализованных на основе 32-разрядных процессорных ядрах используют и более сложные встроенные отладочные средства. Реализуемый при этом набор отладочных функции ничем не уступает возможностям внутрисхемных эмуляторов. Возможностей рассмотренных внутрисхемных отладчиков становится недостаточно для эффективной реализации этих функций. Необходимо использовать более скоростные отладочные устройства. Компания Microchip для поддержки МК семейств PIC32, реализованных на процессорном ядре MIPS32, разработала такое устройство, но позиционирует его, как внутрисхемный эмулятор REAL ICE (рисунок 5.15).

REAL ICE поддерживает все функции внутрисхемной отладки, но отличается от устройств серии ICDx и PICkitx, значительно более высоким быстродействием и выполнять функции внутрисхемного эмулятора, такие как:

- трассировка программного кода и данных;

- триггеры останова по условному внешнему событию;

- программные точки останова;

- интерфейс мониторинга и управления данными (DMCI).

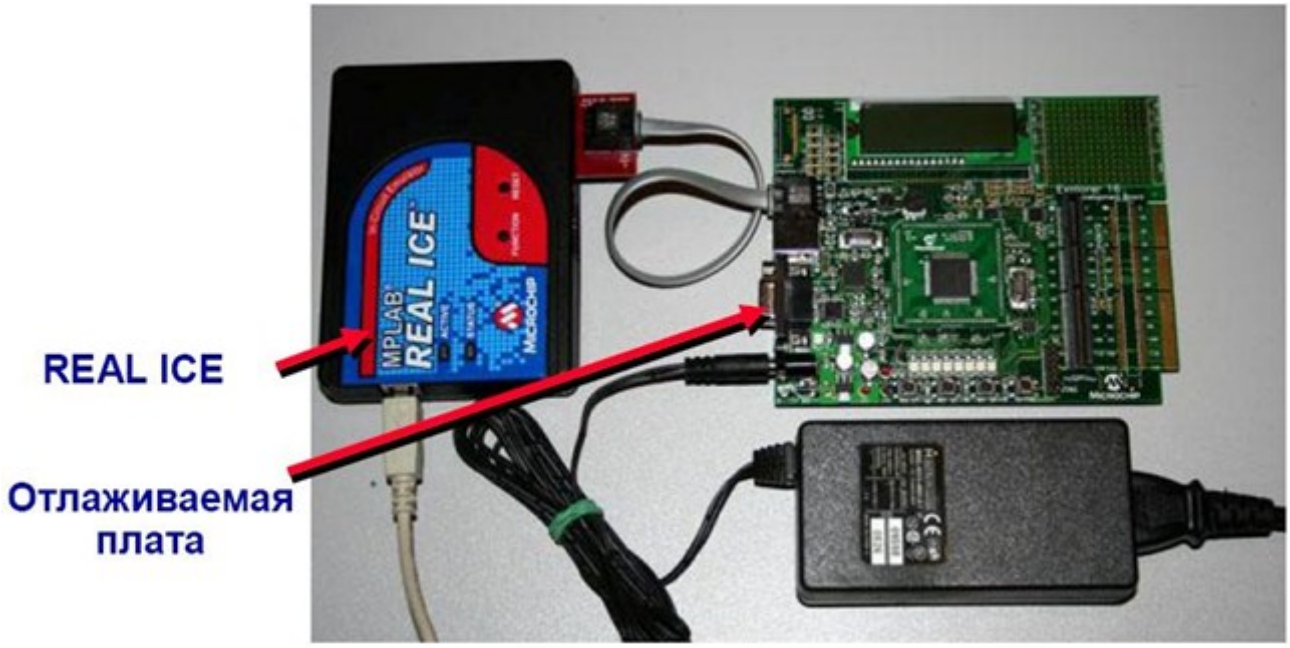


Рисунок 5.15 – REAL ICE

Трассировка программного кода позволяет просмотреть последовательность выполнения команд в реальном времени. Всего может быть зафиксировано до 22 млн. строк. А трассировка данных позволяет анализировать историю изменений переменных, значений регистров или сигналов на выводах контроллера в процессе работы устройства. Для этого в коде программы ставятся специальные метки:

Log(id, x),

где id – номер используемого канала, который определяется автоматически;

x – переменная, значение которой фиксируется при трассировке.

Компилятор автоматически вставляет в код служебные команды, обеспечивающие трассировку. Текущее значение трассируемой переменной будет заноситься в историю каждый раз, когда программа будет проходить через метку. Потом историю можно просмотреть в окне Trace системы MPLAB IDE.

Совместно с обработчиком особых ситуаций (except handler), к которым относятся: прерывания, ошибки адреса, ошибки шины, точки останова, ошибки со-

процессора, арифметическое переполнение, деление на ноль и т.д., трассировка позволяет определить ветку программы, выполнение которой привело к ошибке.

Восемь каналов триггеров могут использоваться для останова или сброса МК по изменению состояния логического внешнего сигнала, а также для формирования сигнала нужного уровня на выходе одного из каналов в точке останова.

Сильным ограничением на возможности встроенных отладочных средств МК является количество аппаратные точек останова. Real ICE поддерживает программные точки останова. В отличие от аппаратных точек останова они вставляются в код программы при компиляции. Их применение позволяет избавиться от эффекта проскальзывания, свойственного всем внутрисхемным отладчикам, и снимает ограничение на количество точек останова.

Интерфейс мониторинга и управления данными (DMCI) позволяет строить графики изменения интересующих переменных программы или регистров в процессе работы устройства, а также принудительно задавать значения этих переменных во время выполнения программы (рисунок 5.16).

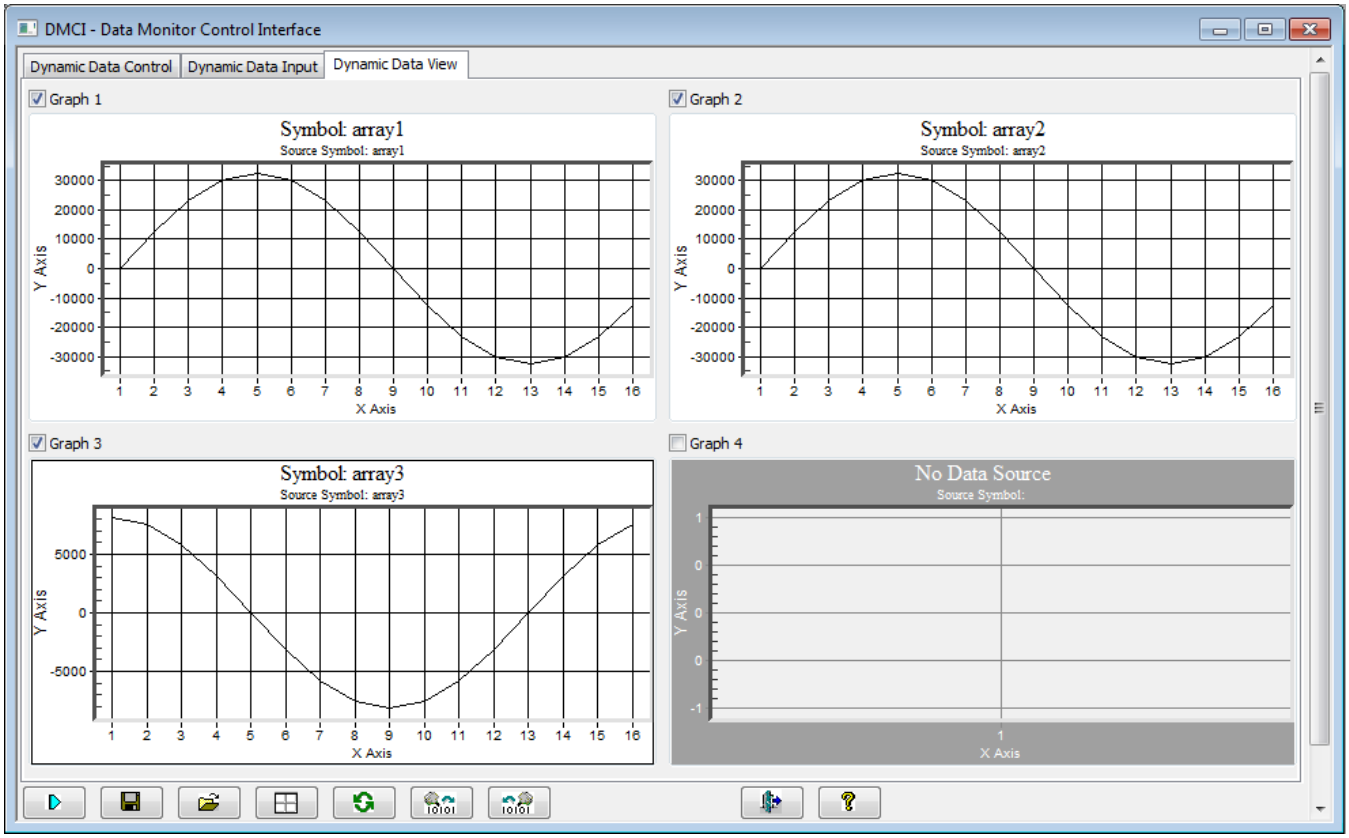


Рисунок 5.16 – Окно DMCI

5.4 Встроенные средства отладки с интерфейсом JTAG

В отличие от компании Microchip многие из их конкурентов не стали разрабатывать собственные отладочные интерфейсы, а адаптировали для этих целей стандартные инструментальные интерфейсы. Наиболее удачные решения были получены на основе интерфейса JTAG. Интерфейс стандарта IEEE 1149.1, был разработан группой JTAG для выполнения технической диагностики аппаратных средств. Он позволяет определять исправность ИС и их внешних цепей на основе метода граничного сканирования. Технические возможности этих средств позволили на их основе разработать встроенные средства программирования и отладки МК.

На рисунке 5.17 приведена функциональная схема встроенной отладочной системы МК Atmega128 семейства AVR. JTAG-интерфейс задействует 4 вывода МК. Эти выводы образуют "Порт доступа к функциям тестирования" (TAP):

-TMS – выбор режим тестирования (используется для управления TAP-контроллером);

-TCK – сигнал синхронизация;

-TDI - тестовый ввод данных (в регистр инструкции или регистры данных);

-TDO -тестовый вывод данных (из регистра инструкции или регистра данных).

TAP-контроллер – цифровой автомат, который управляется сигналами TCK и TMS. TAP-контроллер выбирает в качестве сканируемой цепи (сдвигового регистра) между входом TDI и выходом TDO регистр JTAG-инструкции или один из нескольких регистров данных. В регистре инструкции сохраняются JTAG-инструкции, которые управляют поведением регистра данных.

Регистр идентификации (ID), регистр пропуска и регистры цепи граничного сканирования и данных используются для технической диагностики аппаратных средств. JTAG интерфейс программирования (состоит из нескольких физических и виртуальных регистров данных) используется для последовательного внутрисхемного программирования МК.

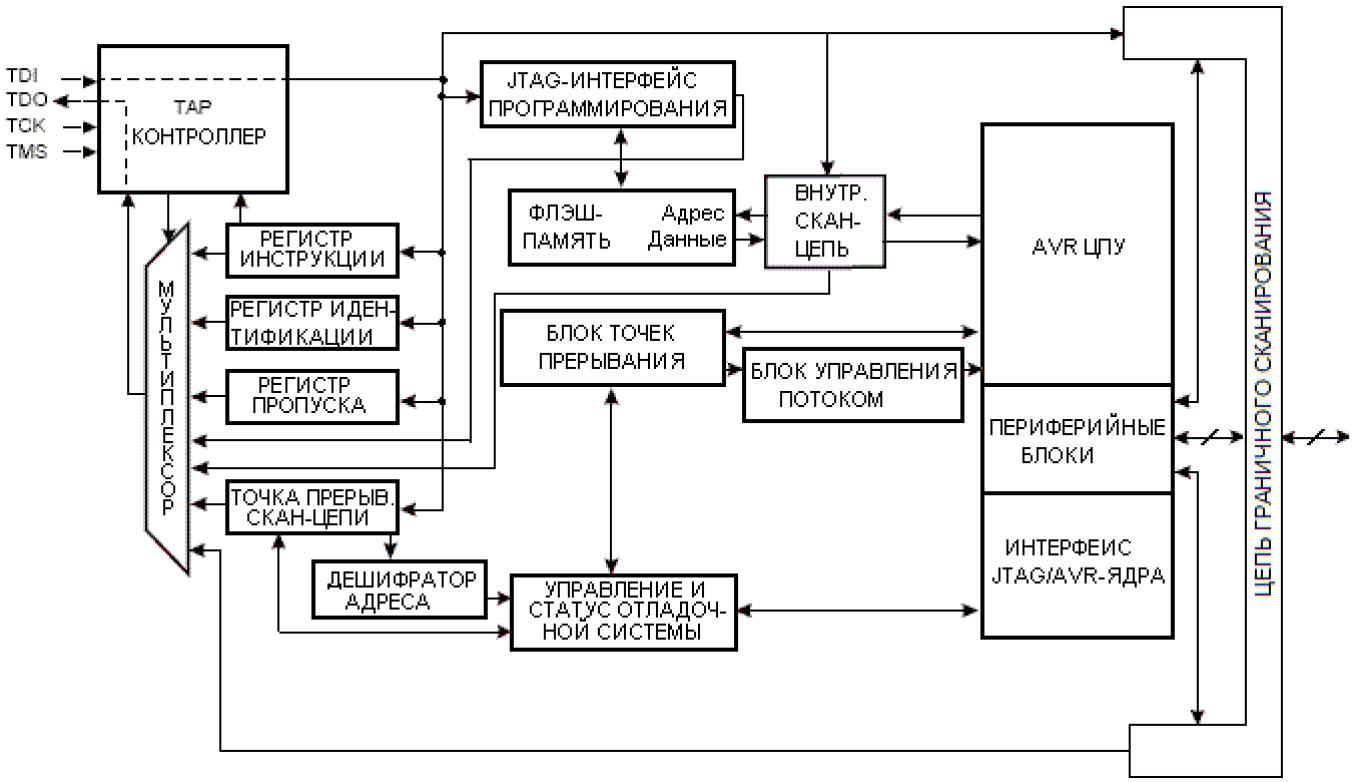


Рисунок 5.17 - Встроенная отладочная система МК Atmega128

Если конфигурационный бит JTAGEN не запрограммирован, то выводы TAP-порта выполняют функции порта ввода-вывода, а TAP-контроллер находится в состоянии сброса. Если же бит JTAGEN запрограммирован, а также сброшен бит JTD в регистре MCUCSR, то к входам порта TAP подключаются внутренние подтягивающие резисторы и разрешается работа интерфейса JTAG для граничного сканирования и программирования. Когда TAP-контроллер не выполняет сдвиг данных, выход TDO находится в третьем состоянии и должен быть подключен к внешнему подтягивающему резистору. Системой встроенной отладки в дополнении к сигналам интерфейса JTAG используется вывод RESET, состояние которого отслеживает отладчик для определения возникновения условия внешнего сброса. Отладчик может установить низкий уровень на выводе RESET, поэтому источник внешнего сброса должен иметь открытый выход.

Аппаратная часть отладочной системы включает:

- цепь внутреннего сканирования;

- блок точек прерывания;

- блок управления отладочной системы;

- интерфейс между центральным процессорным устройством (AVR ЦПУ) и JTAG-системой.

Все операции чтения или записи, необходимые для работы отладчика, реализуются путем выполнения AVR-инструкций через цепь внутреннего сканирования. ЦПУ отправляет результат в память ввода-вывода по указанному адресу. которая является частью интерфейса между AVR ЦПУ и JTAG-системой.

Блок точек прерывания поддерживает прерывания по изменению программного потока, пошаговое прерывание, две точки прерывания в памяти программ и две комбинированных точки прерывания.

Для разрешения работы порта доступа к функциям тестирования (TAP) необходимо запрограммировать конфигурационный бит JTAGEN, конфигурационный бит OCDEN, не установлены биты защиты программы. В противном случае, отладочная система могла бы служить способом считывания защищенной программы.

Интегрированная среда разработки AVR Studio предоставляет разработчику полное управление выполнением программы AVR-МК. Для подключения к инструментальному компьютеру необходимо использовать внутрисхемный отладчик - программатор AVR ICE.

Средства высокоуровневой отладки позволяют отслеживать выполнение программы на уровне исходного кода или на уровне дизассемблера. Пользователь может запустить программу на исполнение, выполнить программу пошагово как с переходом к обработке подпрограмм, так и без, выполнить шаг назад, выполнить программу до позиции курсора, остановить программу, а также сбросить текущий сеанс выполнения программы (эквивалентно сбросу микроконтроллера). Кроме того, пользователь может установить неограниченное число точек останова по адресу памяти программ (используя инструкцию BREAK) и до двух точек останова при обращении к адресам памяти программ.

Для более производительных МК разработали расширенную версию встроенного отладчика, названную EJTAG (Enhanced JTAG). На рисунке 5.18 приведена структурная схема отладочной системы для МП с архитектурой MIPS32. Для упрощения физической реализации, снижения стоимости и обеспечения совмести-

мости в EJTAG используется интерфейс, аналогичный JTAG. Дополнительная логика, размещенная на кристалле, обеспечивает управление, установку точек останова, трассировку счетчика инструкций в реальном времени.

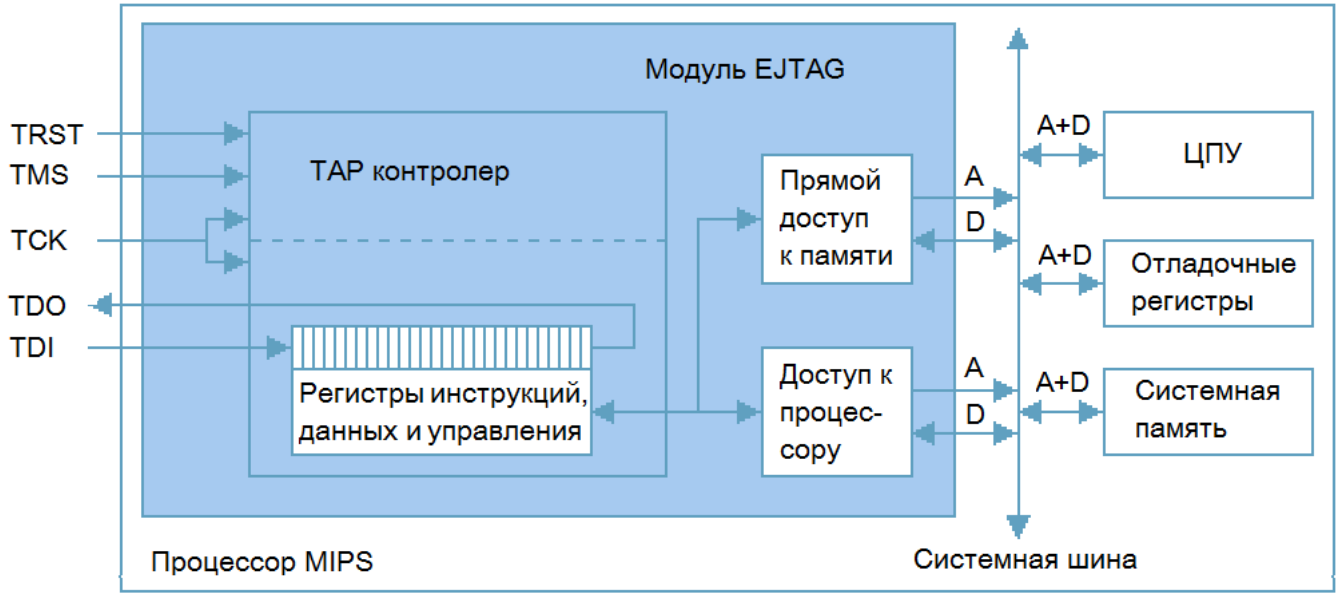


Рисунок 5.18 - Структура отладочных средств EJTAG

Передача данных через TAP-порт происходит в последовательном коде с частотой 40 МГц. Управление работой EJTAG осуществляется с помощью набора внутренних 32-разрядных регистров, которые используются для определения ресурсов, необходимых для отладки и сбора информации о текущем состоянии. Работа встроенного модуля EJTAG контролируется с помощью внешнего EJTAG-адаптера, служащего интерфейсом между инструментальным компьютером и целевым МП.

Средства отладки EJTAG требуют высокой интеграции с процессорным ядром. Для обеспечения процесса отладки блок центрального процессора (ЦП) должен поддерживать специальный режим отладки. Одна из наиболее важных особенностей - использование специального отладочного исключения, приоритет которого выше приоритетов всех других исключений. Оно возникает в случае, когда установлена программная точка останова, выдана инструкция пошагового исполнения или зарегистрировано отладочное событие JtagBrk в EJTAG. Инструкция программной точки останова (SDBBP) определена в наборе команд архитектуры

MIPS32. Для простых точек останова отладчик может подменять инструкции приложения на программные точки останова для генерации отладочного исключения.

Когда исключение возникает, ЦП переходит в специальный отладочный режим, в котором нет ограничений на доступ к ядру процессора и памяти, и обычные исключения, такие как ошибка адреса или прерывание, замаскированы.

Обработчик исключения обеспечивается отладочной системой и может запускаться через EJTAG-порт, используя доступ к процессору, или включается в код приложения по выбору разработчика.

Специальные отладочные регистры DEBUG, DEPC и DESAVE содержат отладочную информацию. Регистр DEBUG показывает источник отладочного исключения и других стандартных исключений, которые могут возникать одновременно. Он также используется для установки пошагового режима. Регистр DEPC (Debug Exception Program Counter) содержит адрес точки возникновения исключения. Он используется для возврата в исполняемую программу после завершения режима отладки. Регистр DESAVE (Debug Exception Save Register) может хранить дополнительную информацию, освобождая 32-разрядные регистры процессора общего назначения от задачи сохранения обработчика исключения. В этом случае обработчик исключения не затрагивает содержимое регистров процессора общего назначения.

В спецификации EJTAG существуют несколько типов аппаратных точек останова. Они останавливают нормальную работу процессора и переводят его в отладочный режим. Останов возникает в случае, когда на шинах адреса, данных или управления происходят определенные операции. Отладочное исключение возникает до выполнения операции, сохраняя тем самым содержимое регистров или памяти. Аппаратные точки останова, в отличие от программных, могут ассоциироваться с определенным адресом на шине, так что могут использоваться для доступа к любой области памяти. Всего определено три типа аппаратных точек останова: по команде, по данным и по шине процессора.

5.5 Встроенные средства отладки Cortex-M3

Традиционно в процессорах с архитектурой ARM для отладки использовался интерфейс JTAG. В процессорные ядра Cortex-M3 встроена новая система отладки CoreSight [13]. Система отладки CoreSight предоставляет широкий набор отладочных функций:

- останов и пошаговое выполнение программы;
- аппаратные точки останова;
- поддержка команд точки останова;
- точки наблюдения при обращении к конкретному адресу, диапазону адресов или значению;
- чтение и запись регистров;
- исключение Debug monitor;
- обращения к памяти во время работы ядра;
- трассировка команд (при использовании модуля ETM);
- трассировка данных;
- программная трассировка (посредством модуля ITM);
- профилирование (посредством модуля DWT).

В процессорах с ядром Cortex-M3 для отладки используется специальный интерфейс Debug Interface (рисунок 5.19). Внешние средства отладки могут использовать для подключения порт JTAG или скоростной двухпроводный интерфейс Serial Wire (SWD). Совмещенный порт получил обозначение SWJ. Отладочная система CoreSight содержит средства управления отладкой. В контроллере прерываний NVIC имеется блок регистров Run Control для управления основными отладочными операциями, включая останов и пошаговое выполнение программ. К средствам управления отладкой относятся модуль точек останова Breakpoint Unit и модуль доступа к памяти Memory Access Unit [14].

Также отладочная система CoreSight содержит средства управления трассировкой:

- блок трассировки данных - Data Watchpoint & Trace Unit (DWT);

- блок генерации отладочных сообщения Instrumentation Trace Macrocell (ITM);

- блок трассировки команд (опция) - Embedded Trace Macrocell (ETM).

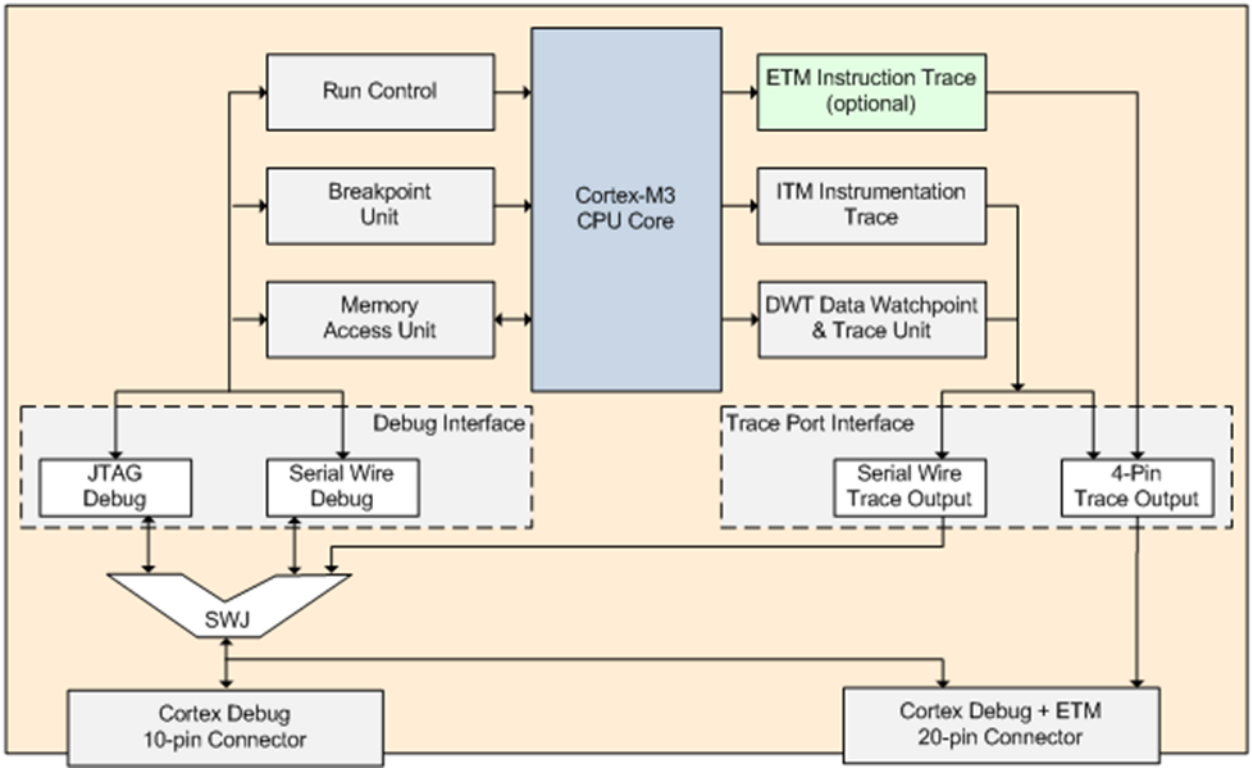


Рисунок 5.19 - Система отладки CoreSight

Модули трассировки DWT и ITM через интерфейс порта трассировки Trace Port Interface могут быть связаны однопроводной линией SWO с отладочным портом SWJ (SWV-режим) или высокоскоростным четырехпроводным каналом трассировки (синхронный режим). Опциональный модуль ETM может быть связан только с четырехпроводным каналом трассировки. Внешние отладочные устройства могут подключаться к отладочному порту (Cortex Debug) через стандартный 10-контактный разъемный соединитель. Или к совмещенному порту отладки и трассировки (Cortex Debug +ETM) через 20-контактный разъемный соединитель.

В процессоре Cortex-M3 предусмотрено два режима выполнения отладочных операций:

1) режим останова, в котором процессор полностью прекращает выполнение программы;

2) режим исключения Debug monitor , в котором отладочные операции выполняются в обработчике данного исключения. В этом режиме могут возникать исключения с более высоким приоритетом. Исключение Debug monitor имеет программируемый приоритет. Это исключение может быть вызвано различными событиями отладки.

В режим останова исполнение команд и системный таймер SYSTICK останавливаются, поддерживаются пошаговые операции с возможностью возникновение и обработка прерываний.

В режиме монитора отладки процессор выполняет код обработчик исключения Debug monitor, счётчик таймера SYSTICK продолжает работать. Новые прерывания могут прерывать выполнение обработчика монитора отладки при их более высоком приоритете, поддерживаются пошаговые операции.

Во время отладки возможно разместить одну или несколько точек останова по адресам инструкций программы или констант. При выполнении команды или при чтении значения, расположенного по такому адресу, генерируется событие отладки, которое приводит к останову программы (для отладки в режиме останова) или генерации исключения Debug monitor (для отладки в режиме монитора). После этого можно проконтролировать содержимое регистров, памяти, продолжить пошаговую отладку программы.

Система трассировки ядра Cortex-M3 содержит до трёх источников трассировочных данных — это модули ETM , ITM и DWT. Модуль ITM является опциональным, поэтому некоторые МК с ядром Cortex-M3 не поддерживают трассировку команд. Данные от этих источников передаются по общему каналу. Во время работы каждому источнику трассировки присваивается уникальный 7-битный идентификатор (ATID), который вместе с пакетом данных пересылается по каналу трассировки. Этот идентификатор позволяет хосту отладки разделить полученные пакеты на отдельные потоки данных трассировки.

Модуль DWT содержит в своём составе четыре компаратора, каждый из которых может быть сконфигурирован для реализации:

- аппаратной точки наблюдения (генерирует событие точки наблюдения, переводящее процессор в один из режимов отладки);

- триггера ETM (модуль ETM вставляет в поток трассировки триггер-пакет);

- триггера события выборки счётчика команд;

- триггера события выборки адреса.

Модуль ITM имеет следующие возможности:

- программа может напрямую записывать сообщения в порты стимулов модуля ITM и выводить их в виде данных трассировки;

- модуль DWT может генерировать пакеты трассировки и выводить их через модуль ITM;

- модуль ITM может генерировать пакеты временных меток, которые вставляются в поток данных трассировки, что помогает вести хронометраж событий.

Основной функцией модуля ITM является поддержка вывода отладочных сообщений (используя функцию printf). В модуле ITM имеется 32 порта стимулов, в которые различные процессы могут выводить свою информацию. Последующее отделение сообщений различных процессов друг от друга осуществляется хостом отладки. В интегрированной среде разработки μVision имеется специальный модуль ITM Viewer для сбора и отображения выводимой информации.

Если процессор содержит модуль ETM, и он включён, то в процессе трассировки генерирует пакеты трассировки команд. Для уменьшения объёма передаваемых данных модуль ETM сообщает только о командах переходов, выполненных процессором. Поскольку хост отладки имеет копию двоичного образа программы, то этих данных ему достаточно для восстановления последовательность команд, выполненных процессором.

5.7 Вопросы для самоконтроля к разделу 5

Поясните принцип работы внутрисхемного эмулятора.

Что понимают под эмуляционным кристаллом процессора?

Чем отличаются аппаратные и программные точки останова?

Каким образом обеспечиваются условия контролепригодности за счет применения внутрисхемных эмуляторов?

Перечислите основные преимущества и недостатки внутрисхемных эмуляторов.

Что понимают под встроенными средствами отладки?

Поясните принцип работы модуля теневой отладки МК PIC16F.

Что понимают под отладочным прерыванием?

Какие функции выполняет обработчик отладочного прерывания?

Поясните назначение регистров модуля теневой отладки МК PIC16F.

Какие функции выполняют внутрисхемные отладчики – программаторы?

Что понимают под отладочным интерфейсом?

Приведите примеры реализации отладочных интерфейсов.

Перечислите основные режимы работы отладчика системы MPLAB IDE. Как выполняется настройка среды MPLAB IDE на требуемый режим отладки?

Каким образом можно отслеживать выполнение программы с использованием встроенных средств отладки?

Поясните организацию встроенной отладочной системы CoreSight.

Поясните назначение модулей системы CoreSight.

Перечислите функциональные возможности встроенной отладочной системы CoreSight.

Какие средства системы CoreSight обеспечивают выполнение условий управляемости и наблюдаемости?

Список использованных источников

1 Микропроцессоры [Текст] : в 3 кн.: учеб. для втузов / под ред. Л. Н. Преснухина. - М. : Высш. шк., 1986. Кн. 3 : Средства отладки, лабораторный практикум и задачник / Н. В. Воробьев, В. Л. Горбунов, А. В. Горячев. - 1986. - 350 с. : ил. - Библиогр.: с. 342-344.

2 Уильямс, Г.Б. Отладка микропроцессорных систем: пер. с англ. [Текст] / Г.Б. Уильямс. - М: Энергоатомиздат, 1988. - 254 с.

3 Отладочная плата LDM-K1986BE92QI ARM Cortex-M3: описание продукта [Электронный ресурс]. DS-K1986BEQI-1-3, 2015. - 51 с. . – Режим доступа : https://ldm-systems.ru/f/doc/catalog/LDM-K1986BE92QI-H/LDM-K1986BE92QI.pdf /. – 26.04.2018.

4 Белов, А.М. Средства автоматизации программирования микропроцессорных устройств [Текст] / А.М. Белов, Е.А. Иванов, Л.Л. Муренко; Под ред. В.Г. Домрачева. – М.: Энергоатомиздат, 1988. – 120 с.

5 Яценков, В. С. Микроконтроллеры MicroCHIP [Текст] : практ. рук. / В. С. Яценков.- 2-е изд., испр. и доп. - Москва : Горячая линия-Телеком, 2008. - 280 с. : ил. - (Современная электроника).

6 MPASM [Электронный ресурс]: Руководство пользователя. – М.: ООО «Микро-Чип», 2001. – 183 с. – Режим доступа: http://www.microchip.ru/files/d-sheets-rus/mpasm.pdf /. – 26.04.2018.

7 MPLAB IDE [Электронный ресурс]: User’s Guide DS51519B.– Microchip, 2006. – 277 p. – Режим доступа: http://www.cis.upenn.edu/~lee/06cse480/data/MPLAB_IDE_User_guide.pdf/.– 27.04.2018.

8 HI-TECH PICC-Lite Compiler [Электронный ресурс]: User’s manual. – HI-TECH Software, 2007. – Режим доступа : https://is.muni.cz/el/1433/jaro2010/PB171/um/PICC_manual.pdf /. – 27.06.2010.

9 Копытин, С. Программный пакет разработки для ARM-микроконтроллеров RealView Keil. / С. Копытин, К. Дорофеев // Компоненты и технологии. – 2008. - № 9. - С. 73 – 76.

10 In-Circuit Serial Programming™ (ICSP™) Guide. [Электронный ресурс]: Tutorial DS30277C. – Microchip Technology Inc., 2000. – Режим доступа : https://www.elnec.com/sw/30277d.pdf .- 27.11.2018.

11 EEPROM Memory Programming Specification. [Электронный ресурс]: Tutorial DS39025F. – Microchip Technology Inc., 2002. – Режим доступа : http://pdf.eicom.ru/datasheets/microchip_pdfs/pic16f87x_eeprom_memory_programming_specification/pic16f87x_eeprom_memory_programming_specification.pdf____/. – 27.11.2018.

12 On-Chip Debugger Specification. [Электронный ресурс]: Tutorial DS51242A. – Microchip Technology Inc., 2001. – Режим доступа : https://www.mikrocontroller.net/attachment/111659/On_Chip_Debug_Spec_51242a.pdf /. – 27.11.2018.

13 Копытин, С. Отладка микроконтроллеров на базе процессоров Cortex-M3 / С. Копытин, К. Дорофеев // Компоненты и технологии. – 2009. - № 4. - С. 84 – 86.

14 Ю, Джозеф. Ядро Cortex-M3 компании ARM. Полное руководство [Текст]/ Джозеф Ю; пер. с англ. А. В. Евстифеева. — М. : Додэка-XXI, 2012. — 552 с. : ил. — (Мировая электроника). — Доп. тит. л. англ.