ФГБОУ ВПО «Воронежский государственный
технический университет»

А.В. Барабанов

ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА ЯЗЫКАХ VHDL И VERILOG

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

Воронеж 2015

УДК 681.3

Барабанов А.В. Проектирование цифровых устройств на языках VHDL и Verilog: учеб. пособие / А.В. Барабанов. Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2015. 167 с.

В учебном пособии содержатся практические примеры реализации цифровых устройств различного назначения на языках проектирования VHDL и Verilog.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего профессионального образования по направлению 230100.62 «Информатика и вычислительная техника» (профиль «Вычислительные машины, комплексы, системы и сети»), дисциплинам «Автоматизированное проектирование вычислительных систем», «Программирование на языках VHDL/ Verilog».

Табл. 5. Ил. 50. Библиогр.: 15 назв.

Рецензенты: кафедра вычислительной математики и прикладных информационных технологий Воронежского государственного университета (зав. кафедрой д-р техн. наук, проф. Т.М. Леденева);
д-р техн. наук, проф. О.Н. Чопоров

ВВЕДЕНИЕ

В данном учебном пособии основное внимание уделено практике проектирования устройств цифровой обработки сигналов, микропроцессорных систем, распространенных интерфейсов и программно-аппаратных средств защиты информации. При создании таких устройств используются высокоуровневые языки программирования VHDL и Verilog.

Устройства, как правило, реализуются на программируемых логических интегральных схемах (ПЛИС). ПЛИС и инструментальные средства разработки проектов на их основе представляют собой платформу для создания реконфигурируемых высокопроизводительных цифровых систем и устройств с минимальными материальными затратами и сокращенным временем на проектирование.

Расширение сферы применения ПЛИС определяется растущим спросом на устройства с быстрой перестройкой выполняемых функций, сокращением проектно-технологического цикла изделий, наличием режимов изменения внутренней структуры в реальном масштабе времени, повышением быстродействия, снижением потребляемой мощности, разработкой оптимизированных сочетаний с микропроцессорами и сигнальными процессорами (DSP), а также снижением цен на эти устройства.

Задача учебного пособия – помочь в освоении основных технических приемов при работе с современными системами проектирования, тестирования. В данное время испытывается недостаток литературы, которая служила бы студентам практическим руководством в изучении современных средств моделирования и проектирования цифровых устройств на языках VHDL и Verilog.

ГЛАВА 1. ПРОЕКТИРОВАНИЕ УСТРОЙСТВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

1.1. Этапы проектирования цифровых устройств с использованием языков VHDL и Verilog

Языки VHDL и Verilog обладают некоторыми важными отличительными характеристиками [14,15]:

– Проекты цифровых устройств (ЦУ), созданные с помощью языка VHDL и Verilog, имеют, как правило, иерархическую структуру.

– Описание модуля может иметь поведенческую или структурную форму.

– Моделирование алгоритма работы проекта основывается на событийном принципе управления.

– VHDL/Verilog –проект выполняет моделирование протекания параллельных процессов в электрических схемах, временной анализ сигналов и их параметров.

– VHDL/Verilog поддерживаются инструментальными средствами синтеза многих производителей программного обеспечения.

– Характеристики языков VHDL/Verilog как специализированных языков описания ЦУ реализуются с помощью следующих средств:

– библиотеки;

– проекты;

– подпрограммы: функции, процедуры;

– скалярные типы данных: перечислимые, числовые, физические;

– программные элементы данных: константы, переменные, сигналы, порты, идентификаторы;

– математические операции: логические, отношений, арифметические;

– программные операции: установка значений сигналов, присвоение значений переменным, реализация связи портов и сигналов;

– математические выражения: логические, алгебраические, логико–алгебраические;

– операторы объявления программных элементов данных;

– операторы комбинаторной логики;

– операторы регистровой логики.

VHDL/Verilog –проект описывает ЦУ, учитывая его многогранность, поведение, структуру, функциональные и физические свойства. VHDL и Verilog существу являются языками параллельного программирования, т.е. в их конструкциях существуют операторы, соответствующие логическим вентилям. Эти операторы обрабатываются (т.е. вычисляются) по параллельному принципу и называются операторами параллельной обработки (ОПО) (concurrence operator). Программа, написанная на языках VHDL/Verilog, моделирует физическое поведение системы, сигналы в которой распространяются мгновенно. Такая программа позволяет формировать временную спецификацию (время задержки распространения сигнала на логическом элементе), а также описывать систему как соединение различного ряда компонентов или функциональных блоков.

Аппаратная реализация проекта ЦУ с использованием языков VHDL/Verilog протекает в соответствии со следующими этапами:

1) Разработка иерархической схемы проекта. Использование языков VHDL/Verilog позволяет разбить проект на модули и определить их интерфейсы.

2) Программирование. Запись VHDL/Verilog –кода для модулей и их интерфейсов.

3) Компиляция. Анализ программного кода VHDL/Verilog –проекта для выявления синтаксических ошибок, а также проверка его совместимости с другими модулями. В ходе компиляции также собирается внутренняя информация о структуре проекта, которая необходима для моделирования работы проектируемого ЦУ.

4) Моделирование. Определение и применение входных воздействий к откомпилированному коду проекта с наблюдением выходных реакций. Моделирование может выполняться как в форме функционального контроля, так и в качестве одного из этапов верификации завершенного проекта.

5) Синтез. Преобразование VHDL/Verilog –описания в набор примитивов или логических элементов, которые могут быть реализованы с учетом конкретной технологии.

6) Компоновка, монтаж и разводка. Отображение проекта на карте синтезирующих элементов.

7) Временной анализ. Получение фактических задержек цифровой схемы проекта с учетом длины соединений, электрических нагрузок и других известных факторов.

Проект любого ЦУ на языках VHDL/Verilog – это программа, которая содержит ключевые и зарезервированные слова.

В пособии приведены основные сведения и отдельные реализации проектов на языках VHDL/Verilog: **цифрового фильтра**, микропроцессорной системы, распространенных протоколов связи электронных устройств, криптопроцесоров.

1.2. Применение цифровой фильтрации

Цифровой фильтр — в электронике любой фильтр, обрабатывающий цифровой сигнал с целью выделения и/или подавления определённых частот этого сигнала. Цифровые фильтры на сегодняшний день применяются в спектральном анализе, обработке изображений, обработке видео, обработке речи и звука и многих других приложениях.

Частоты, на которых работают фильтры, нередко достигают нескольких сотен мегагерц и более. Поэтому, растет ширина полос фильтров. Увеличивается порядок фильтра, нередко достигая четырехзначных чисел, постепенно возрастает и разрядность обрабатываемых данных. Это ведет к

увеличению объема вычислений, а значит, и к резкому росту аппаратных затрат.

При реализации цифровых фильтров с использованием программируемых логических интегральных схем (ПЛИС) встает вопрос о минимизации количества составных элементов, необходимых для построения требуемого фильтра.

Одним из наиболее критичных элементов в ПЛИС является умножитель. В состав каждой ПЛИС входит фиксированное количество умножителей, которые могут быть использованы для построения определенного количества систем. Поэтому для оптимального использования ресурсов ПЛИС необходимо минимизировать не порядок фильтра, а число его ненулевых коэффициентов [1]. Различают:

- (КИХ-фильтры), также известные как фильтры скользящего среднего значения (СС);

- фильтры с бесконечной импульсной характеристикой (БИХ-фильтры), также известные как фильтры авторегрессионного скользящего среднего значения (АСС);

- нелинейные фильтры.

Традиционная классификация фильтров начинается с учетом вида их импульсных характеристик. Эта классификация фильтра базируется на диапазонах частот, которые являются полосами прозрачности или заграждения:

- фильтры нижних частот (ФНЧ) (Lowpass) передают составляющие в нижнем диапазоне частот и уменьшают по амплитуде составляющие верхних частот;

- фильтры верхних частот (ФВЧ) (Highpass) передают составляющие в верхнем диапазоне частот и уменьшают по амплитуде составляющие нижних частот;

- полосовые фильтры (ПФ) (Bandpass) передают составляющие по частоте, соответствующие некоторой полосе частот;

- заграждающие фильтры (ЗФ) (Bandstop) уменьшают амплитуды составляющих в некоторой полосе частот [1].

На рис. 1.1 показаны зависимости коэффициентов передачи от частоты идеальных фильтров:

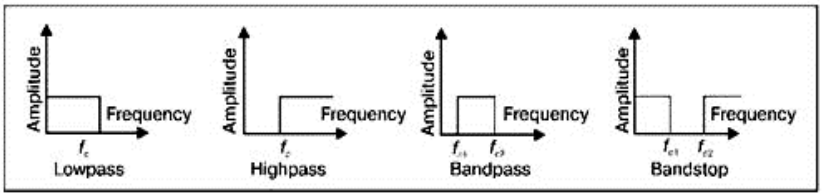


Рис. 1.1. Частотные характеристики идеальных фильтров

Цель проектирования фильтров заключается в определении этих граничных частот.

Полоса пропускания идеального фильтра - диапазон частот гармонических составляющих входного сигнала, которые проходят через фильтр без изменения амплитуд. Полоса задержания идеального фильтра - диапазон частот гармонических составляющих, которые не проходят через фильтр.

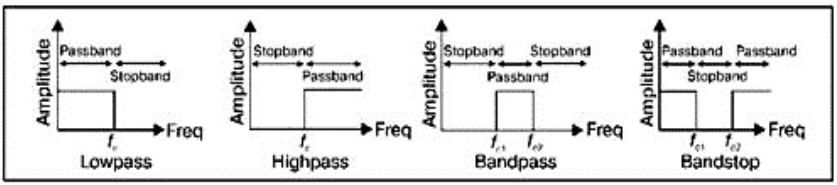


Рис. 1.2. Полоса пропускания (Passband) и полоса задержания (Stopband) для различных фильтров и для каждого типа идеального фильтра

Фильтры, частотные характеристик которых показаны на рис. 1.2, имеют следующие полосы пропускания и полосы задержания:

- ФНЧ и ФВЧ имеют одну полосу пропускания и одну полосу задержания;

- ПФ имеет одну полосу пропускания и две полосы задержания;

- ЗФ имеет две полосы пропускания и одну полосу задержания. Практически реализуемых идеальных фильтров не существует.

Для создания оптимального фильтра необходимо выполнить постановку, формализацию и решение задачи синтеза структуры фильтров, удовлетворяющих заданной совокупности показателей качества и ограничений. В настоящее время главным критерием при проектировании таких систем является минимизация среднеквадратичной ошибки. В зависимости от того, какими уравнениями описывается состояние системы, оптимальные (квазиоптимальные) фильтры подразделяются на линейные и нелинейные. Первые обычно базируются на оптимальном фильтре Калмана, а вторые - на многоканальных цифровых фильтрах.

При разработке адаптивного фильтра структура и параметры устройства подстраиваются под априорно неизвестную структуру исследуемого процесса с целью возможно более эффективного решения поставленной задачи. Как правило, это рекуррентная процедура пересчета вектора отсчетов импульсной характеристики. Однако априорно устойчивые адаптивные алгоритмы, как правило, чрезвычайно сложны в реализации, а более простые алгоритмы (например, известный метод наименьших квадратов - МНК) могут расходиться. Тем не менее, сегодня наиболее распространенными являются алгоритмы на основе МНК и рекуррентного метода МНК (так называемые РНК-алгоритмы), а также методы, базирующиеся на нелинейной теории устойчивости.

В практике реализации ЦФ часто встречаются ситуации, когда применение корректных с математической точки зрения процедур обработки становится нецелесообразным из-за неоправданно высоких затрат на их аппаратное воплощение. В

таких случаях принято использовать эвристическую фильтрацию [2].

В настоящее время большинство разработчиков использует для реализации алгоритмов обработки цифровые сигнальные процессоры (ЦСП) общего назначения. Их главными преимуществами являются гибкость базирующихся на них систем, а также возможность реализации адаптивных и обучающих алгоритмов.

Вместе с тем ЦСП имеют и ряд недостатков, которые, безусловно, приходится учитывать при разработке новых изделий. Во-первых, тактовая частота ЦСП не превышает 30-50 МГц, что ограничивает их область применения. Во-вторых, каждое семейство ЦСП имеет собственные коды команд, поэтому практически невозможен перенос реализованного на одном ЦСП алгоритма на процессоры других семейств или создание универсальных библиотек алгоритмов. В-третьих, при реализации сложных структур приходится увеличивать число процессоров и обеспечивать их работу в мультипроцессорном режиме.

Как известно, альтернативой ЦСП при реализации ЦФ являются программируемые логические интегральные схемы. Основные преимущества ПЛИС перед ЦСП:

– высокое быстродействие;

– возможность реализации сложных параллельных алгоритмов;

– наличие средств САПР, позволяющих проводить полное моделирование систем;

– возможность программирования или изменения конфигурации;

– совместимость при переводе алгоритмов на уровне языков описания аппаратуры (VHDL, AHDL, Verilog и др.);

– совместимость по уровням и возможность реализации стандартного интерфейса;

– наличие библиотек программ, описывающих сложные алгоритмы.

Архитектура ПЛИС как нельзя лучше приспособлена для реализации таких операций, как умножение, свертка и т. п. В настоящее время быстродействие ПЛИС достигло 200-300 МГц, что позволяет реализовать многие алгоритмы в радиочастотном тракте (в частности, алгоритмы фильтрации, которые применяются в устройствах с конечной импульсной характеристикой)[3].

1.3. Реализация фильтра с конечной импульсной характеристикой на базе ПЛИС XILINX

1.3.1. Структурная схема КИХ-фильтра

Проектирование КИХ-фильтров базируется, в первую очередь, на том, что частотная характеристика фильтра определяется импульсной характеристикой, а во-вторых, на том, что коэффициенты фильтра определяются его квантованной импульсной характеристикой. На вход КИХ-фильтра подается одиночный импульс, и по мере прохождения этого импульса через элементы задержки, на выходе поочередно формируются коэффициенты фильтра. На базе ПЛИС Xilinx Spartan 3 с использованием языка VHDL реализуется алгоритм цифровой фильтрации КИХ-фильтра нижних частот, обеспечивающего эффективную фильтрацию входного сигнала.

Разрабатываемый КИХ-фильтр имеет структуру, изображенную на рис. 1.3.

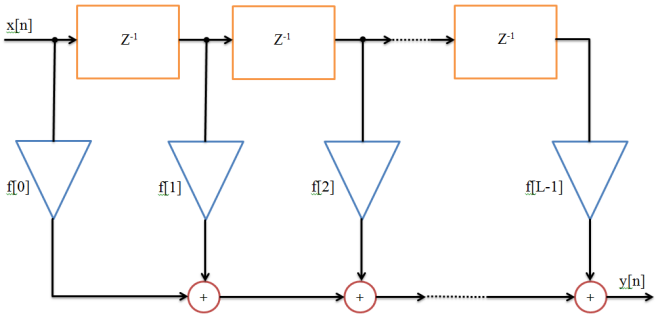


Рис. 1.3. Структурная схема разрабатываемого КИХ-фильтра

На рисунке: x[n] – входные данные, y[n] – выходные данные, f[0 – (L-1)] – блоки обработки данных, Z-1 – блоки задержки, также на рис. 1.3. изображены сумматоры, обеспечивающие на выходе результирующие данные.

1.3.2. Организация входных и выходных данных

Разрабатываемая система цифровой обработки сигналов способна фильтровать сигналы, как в аналоговой, так и в цифровой форме. Но сам процесс обработки сигнала производится над цифровой формой сигнала. В этом случае возникает потребность в использовании дополнительного модуля - аналого-цифрового преобразователя (АЦП). Сигнал поступает сначала на АЦП, в котором происходит его преобразование в цифровую форму, а затем оцифрованный сигнал поступает на вход ПЛИС.

По завершению процесса фильтрации, обработанный сигнал поступает на выход ПЛИС, находясь в цифровой форме. Если же для дальнейшей работы необходим сигнал в аналоговой форме, то можно использовать цифро-аналоговый преобразователь (ЦАП), на вход которого подается цифровой сигнал и, пройдя процесс преобразования, на выходе получится отфильтрованный сигнал в аналоговой форме. Схема подключения АЦП и ЦАП совместно с ПЛИС приведена на рис. 1.4.

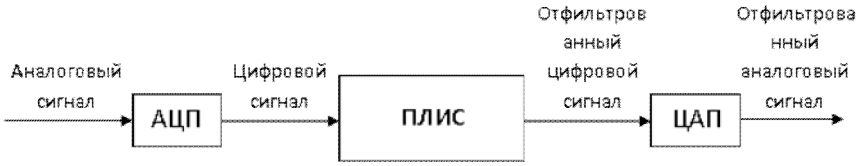


Рис. 1.4. Схема подключения АЦП и ЦАП к ПЛИС

Для решения проблемы представления входных и выходных данных в нужной форме можно воспользоваться встроенными в ПЛИС АЦП и ЦАП, но тогда необходимо к

разъемам расширения ПЛИС подключить дополнительный модуль, который сможет обеспечить ввод и вывод данных в аналоговой форме через соответствующие порты, так как используемая ПЛИС изначально их не имеет.

1.3.3. Алгоритм функционирования КИХ-фильтра

Алгоритм функционирования разрабатываемого КИХ-фильтра приведен на рис. 1.5.

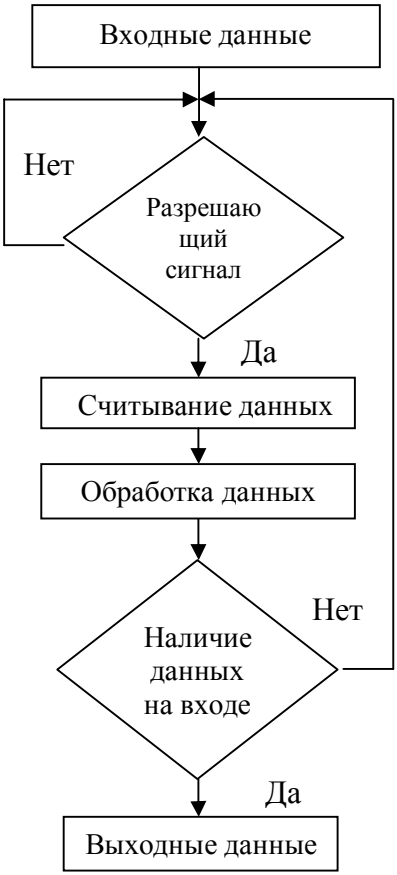


Рис. 1.5. Алгоритм функционирования КИХ-фильтра

Входные данные разделяются на вектора, количество которых зависит от длины фильтра, в соответствующие промежутки времени они считываются и обрабатываются. Так как векторов может быть несколько, то производится проверка на наличие данных на входе. На этапе обработки элементы вектора перемножаются с соответствующими коэффициентами фильтра и суммируются. В результате получаются выходные (отфильтрованные) данные.

Так как перемножение элементов вектора входного сигнала на соответствующие коэффициенты производится последовательно, написанная на языке VHDL программа получается слишком объемной из-за большого порядка КИХ-фильтра. В данном случае для хранения элементов вектора и коэффициентов используются отдельные массивы, в которых каждому элементу вектора входного сигнала соответствует свой коэффициент. Последовательное перемножение элементов массивов организовано с использованием циклов, что в свою очередь позволяет уменьшить объем программы и повысить ее эффективность.

На первом этапе работы алгоритма происходит проверка наличия разрешающего сигнала на входе, в качестве переменной, хранящей значение разрешающего сигнала, используется переменная Ce, переменная CLK представляет собой сигнал синхронизации, событие CLK'Event необходимо для проверки перехода уровня сигнала синхронизации из состояния «0» в состояние «1». Только при выполнении этих условий начинается процесс считывания и обработки сигнала.

На следующем этапе происходит преобразование коэффициентов фильтра в битовый вектор указанной длины, посредством использования функции conv_std_logic_vector(x,y), где x – переменная, y – длина вектора. Процесс преобразования происходит в цикле, для того чтобы обработать каждый элемент массива коэффициентов фильтра. Переменная order хранит в себе разрядность фильтра, переменная a – переменная счетчика цикла.

Далее происходит считывание данных, находящихся в очереди в блоке задержки, а также запись данных в блок задержки. Pipeline – массив блока задержки, в котором хранятся считанные данные. D – вход, на который подаются данные.

На следующем этапе выполняется непосредственная обработка данных, посредством выборки необходимых составляющих считанного сигнала из блока задержки и их перемножения с соответствующими коэффициентами. Результаты записываются в массив MultRes.

Далее происходит преобразование результатов перемножения, в массиве AddRes происходит последовательное суммирование данных.

Предпоследний элемент массива AddRes и будет являться итогом обработки входного сигнала, именно он и подается на выход Q.

Текст программы, написанной на языке VHDL, приведен в прил. 1.

1.3.4. Проектирование фильтра в САПР ISE

На первом этапе после запуска ISE 12.2 необходимо создать новый проект, указав его название, место хранения, а также тип проектирования (на языке VHDL, либо с помощью редактора схем). Затем следует указать модель ПЛИС, для которой создается проект. На рис. 1.6 показано окно с выбором ПЛИС и ее параметров.

После выбора параметров и указания названия проекта осуществляется переход в главное окно САПР ISE, в котором в рабочей области проекта необходимо ввести текст программы на языке VHDL, рис. 1.7.

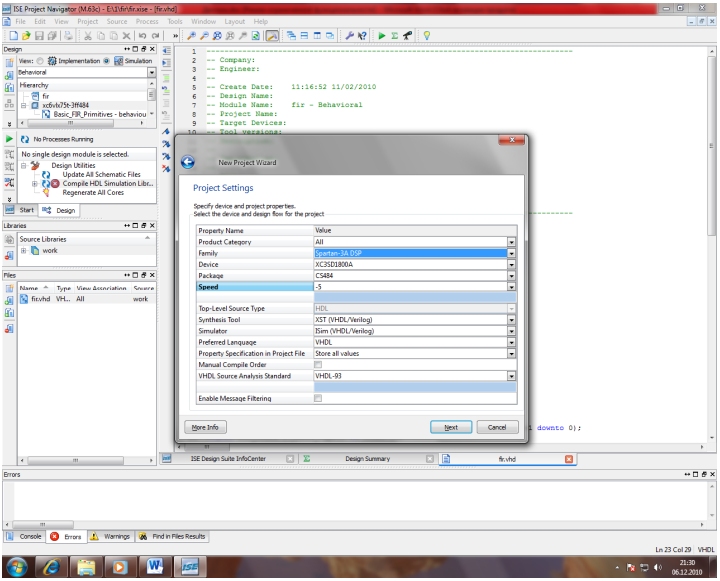


Рис. 1.6. Параметры нового проекта

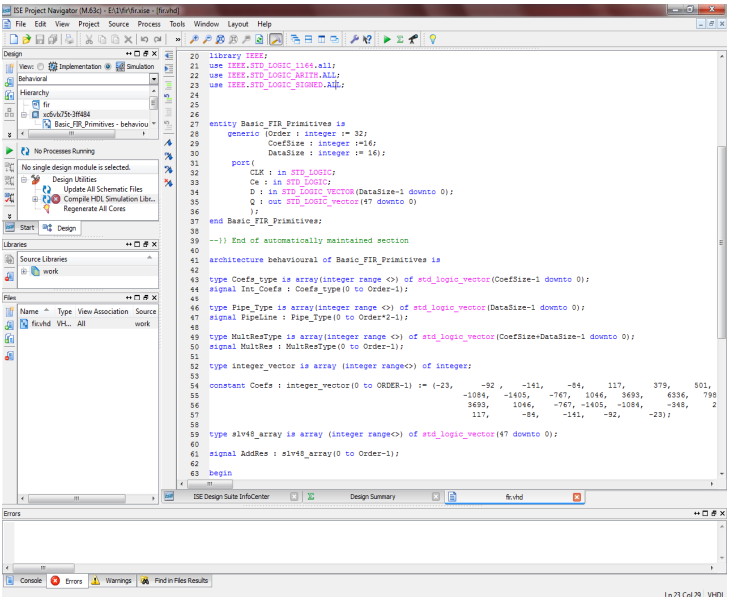


Рис. 1.7. Рабочая область проекта

После завершения ввода текста программы необходимо на уровне представления проекта пройти все пункты меню Processes, расположенного в левой части главного окна САПР ISE, рис. 1.8. В результате САПР производит проверку корректности всех введенных значений, поиск синтаксических ошибок, а также производит сравнение требуемых ресурсов для программы с доступными ресурсами выбранной модели ПЛИС. В случае возникновения ошибки или предупреждения появляется подсказка в нижней части главного окна с номером строки, в которой найдена ошибка, а также со ссылкой для поиска подсказки на сайте технической поддержки.

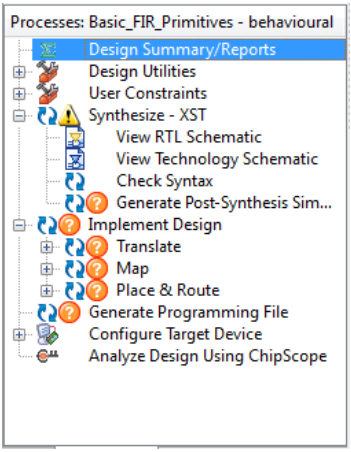


Рис. 1.8. Меню Processes

Для проверки проекта на работоспособность необходимо перейти в режим симуляции, в котором посредством встроенной утилиты ISim можно провести тестирование, рис. 1.9.

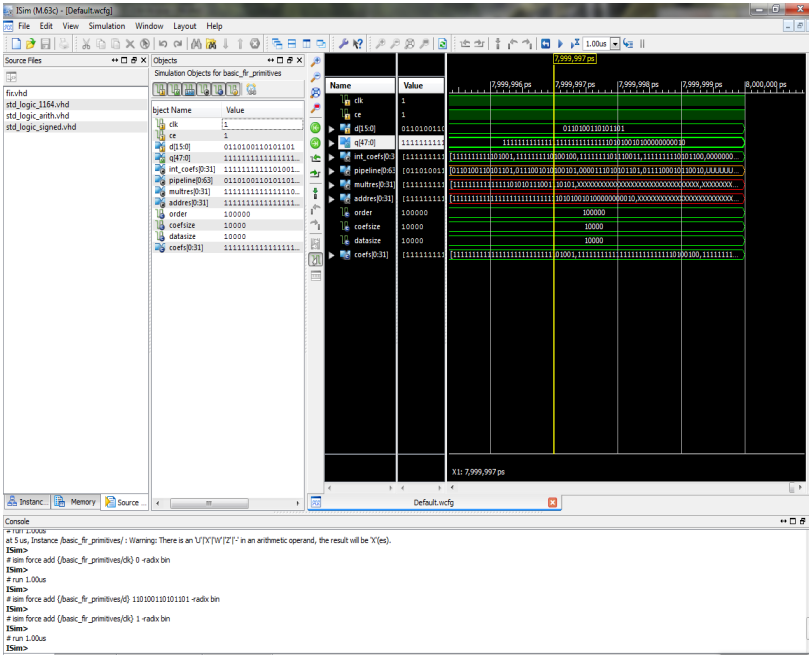


Рис. 1.9. Утилита ISim

На временных диаграммах достаточно наглядно прослеживается работа программы.

После завершения этапа проектирования необходимо подключить ПЛИС к компьютеру и произвести запись проекта при помощи САПР ISE.

ГЛАВА 2. МЕТОДЫ И СРЕДСТВА РАЗРАБОТКИ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

В базовом составе микропроцессорной системы (МПС) генератор тактовых импульсов задаёт временной интервал, который является единицей измерения (квантом) продолжительности выполнения команды. Чем выше частота, тем при прочих равных условиях более быстродействующей является МПС. МП, ОЗУ и ПЗУ - это неотъемлемые части системы. Интерфейсы ввода и вывода - устройства сопряжения МПС с блоками ввода и вывода информации.

Для измерительных приборов характерны устройства ввода в виде кнопочного пульта и измерительных преобразователей (АЦП, датчиков, блоки ввода цифровой информации). Устройства вывода обычно представляют цифровые табло, графический экран (дисплей), внешние устройства сопряжения с измерительной системой. Все блоки МПС связаны между собой шинами передачи цифровой информации. В МПС используют магистральный принцип связи, при котором блоки обмениваются информацией по единой шине данных. Количество линий в шине данных обычно соответствует разрядности МПС (количеству бит в слове данных). Шина адреса применяется для указания направления передачи данных - по ней передаётся адрес ячейки памяти или блока ввода-вывода, которые получают или передают информацию в данный момент. Шина управления служит для передачи сигналов, синхронизирующих всю работу МПС [4].

Главная особенность микропроцессора - возможность программирования логики работы. Поэтому МПС используются для управления процессом измерения (реализацией алгоритма измерения), обработки опытных данных, хранения и вывода результатов измерения и пр.

2.1. Методы разработки микропроцессорных систем

К основным преимуществам микропроцессорных средств измерения относятся:

– Многофункциональность. Замена измерительного комплекса одним, многофункциональным.

– Повышение точности по сравнению с обычными цифровыми приборами при прочих равных условиях достигается за счет исключения систематических погрешностей в процессе самокалибровки: коррекция смещения нуля, учет собственной АЧХ прибора, учет нелинейности преобразователей.

– Уменьшение влияния случайных погрешностей (путем проведения многократных измерений с последующей обработкой выборки — усреднением, вычислением мат. ожидания и пр.). Выявление и устранение грубых погрешностей. Вычисление и индикация оценки погрешности прямо в процессе измерения.

– Компенсация внутренних шумов и повышение чувствительности средства измерения. Простое усреднение сигнала на входе прибора требует достаточно большого времени tуср. Один из вариантов — проведение многократных измерений и усреднение результатов с целью компенсации случайной составляющей измерительного сигнала.

– Расширение измерительных возможностей путем широкого использования косвенных и совокупных измерений (поскольку результат обработки появляется на индикаторе сразу после проведения измерения). Косвенные измерения включают в себя вычисления результата по опытным данным по известному алгоритму. Совокупные измерения предполагают измерение нескольких одноименных физических величин путем решения системы уравнений, получаемых при прямых измерениях сочетаний этих величин.

– Упрощение и облегчение управления прибором. Все управление производится с кнопочной панели, выносные

клавиатуры используют редко. Автоматизация установок прибора приводит к упрощению его использования (выбор пределов измерения, автоматическая калибровка и пр.). В ряде приборов используется контроль за ошибочными действиями оператора — индикация его неверных действий на табло или экране. Упрощает измерения визуализация результатов на экране в удобном виде, с дополнительными шкалами. Ряд приборов предусматривает вывод результатов на печатающее устройство или портативный носитель информации [5].

Ресурсов современных ПЛИС типа FPGA (Field Programmable Gate Array) уже хватает для реализации в них микропроцессоров и даже целых МПС. ПЛИС может быть использована и как платформа для создания прототипов для функционального моделирования систем, которые впоследствии будут реализованы в СБИС, так и как основа для реализации систем как готовых продуктов. Первый вариант экономически целесообразен для многосерийной продукции, а второй - для среднесерийной и мелкосерийной. Преимущества МПС, реализованных в ПЛИС, над системами, выполненными с помощью множества дискретных микросхем средней и высокой интеграции очевидны: быстродействие, надежность, упрощение конструкции печатных плат и т.п.

Объединение инструментальных средств разработки программного обеспечения с инструментальными средствами разработки аппаратного обеспечения может стать важным преимуществом при разработке устройства. Существуют пять различных инструментов, которые используются для разработки приложений на базе микроконтроллеров, и объединение их функций может существенно облегчить процесс проектирования:

- редактор исходных текстов;
- компилятор/ассемблер;
- программный симулятор;
- аппаратный эмулятор;
- программатор.

Наличие в микропроцессорной системе как аппаратных, так и программных средств обуславливает ряд специфических особенностей, присущих процессу ее создания. В отличие от традиционного подхода, когда все функции, возлагаемые на устройство, достигаются чисто аппаратными средствами, при аппаратно-программной реализации выполняемые функции оптимально располагаются между программными и аппаратными средствами микропроцессорной системы.

Процесс разработки встраиваемых микропроцессорных систем на основе ядра MicroBlaze, предназначенных для реализации в FPGA фирмы Xilinx, можно интерпретировать в виде двух маршрутов последовательных этапов проектирования. Первый маршрут — разработка аппаратной части встраиваемой микропроцессорной системы. Второй — проектирование программных средств [5].

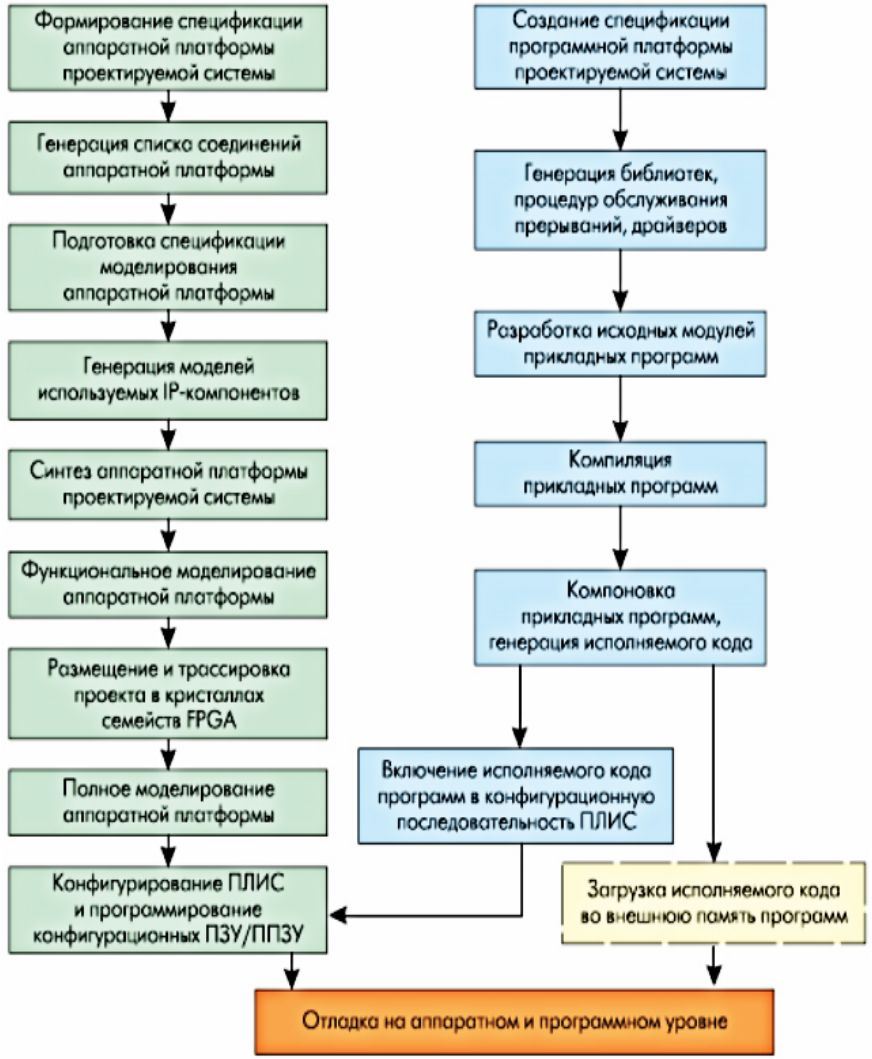


Рис. 2.1. Маршруты проектирования

2.2. Разработка структуры микропроцессорной системы

2.2.1. Этапы проектирования микропроцессорных систем

При проектировании многопроцессорных систем, содержащих несколько типов микропроцессорных наборов, необходимо решать вопросы организации памяти, взаимодействия с процессорами, организации обмена между устройствами системы и внешней средой, согласования функционирования устройств, имеющих различную скорость работы, и т. д. Примерная последовательность этапов, типичных для создания микропроцессорной системы:

- формализация требований к системе;
- разработка структуры и архитектуры системы;
- разработка и изготовление аппаратных средств и программного обеспечения системы;
- комплексная отладка и приемосдаточные испытания.

На первом этапе составляются внешние спецификации, перечисляются функции системы, формализуется техническое задание (ТЗ) на систему, формально излагаются замыслы разработчика в официальной документации.

На втором этапе определяются функции отдельных устройств и программных средств, выбираются микропроцессорные наборы, на базе которых будет реализована система, определяются взаимодействие между аппаратными и программными средствами, временные характеристики отдельных устройств и программ.

На третьем этапе, после определения функций, реализуемых аппаратурой, и функций, реализуемых программами, схемотехники и программисты одновременно приступают к разработке и изготовлению соответственно опытного образца и программных средств. Разработка и изготовление аппаратуры состоят из разработки структурных и принципиальных схем, изготовления прототипа, автономной отладки. Разработка программ состоит из разработки алгоритмов; написания текста исходных программ; трансляции

исходных программ в объектные программы; автономной отладки.

На последнем этапе происходит отладка созданной системы и устранение ошибок [7].

2.2.2. Уровни представления микропроцессорной системы

Микропроцессорная система может быть описана на различных уровнях абстрактного представления. Микропроцессорная система может быть описана, например, на одном из следующих уровней абстрактного представления: "черный ящик"; структурный; программный; логический; схемный. В процессе разработки системы происходит переход от одного уровня ее представления к другому, более детальному.

На уровне "черного ящика" микропроцессорная система описывается внешними спецификациями; перечисляются внешние характеристики.

Структурный уровень создается компонентами микропроцессорной системы: микропроцессорами, запоминающими устройствами, устройствами ввода/вывода, внешними запоминающими устройствами, каналами связи. Микропроцессорная система описывается функциями отдельных устройств и их взаимосвязью, информационными потоками.

Программный уровень разделяется на два подуровня: команд процессора и языковой. Микропроцессорная система интерпретируется как последовательность операторов или команд, вызывающих то или иное действие над некоторой структурой данных.

Логический уровень присущ исключительно дискретным системам. На этом уровне выделяются два подуровня: переключательных схем и регистровых пересылок. Подуровень переключательных схем образуется вентилями и построенными на их основе операторами обработки данных.

Переключательные схемы подразделяются на комбинационные и последовательностные; первые в отличие от последних не содержат запоминающих элементов. Поведение системы на этом уровне описывается алгеброй логики, моделью конечного автомата, входными/выходными последовательностями 1 и 0. Комбинационные схемы представляются таблицей истинности, в которой каждому входному набору значений сигналов ставится в соответствие набор значений сигналов на выходах.

Последовательностные схемы могут описываться диаграммами или таблицами входов/выходов, в которых определены взаимно однозначные соответствия между входами схемы, внутренними состояниями (комбинациями значений элементов памяти) и выходами. Подуровень регистровых пересылок характеризуется более высокой степенью абстрагирования и представляет собой описание регистров и передачу данных между ними. Он включает в себя две части: информационную и управляющую. Информационная часть образуется регистрами, операторами и путями передачи данных. Управляющая часть определяет зависящие от времени сигналы, инициирующие пересылку данных между регистрами.

Схемный уровень образуется резисторами и конденсаторами. Показателями поведения системы на этом уровне служат напряжение и ток, представляемые в функции времени или частоты [7].

Микропроцессорная система на уровне черного ящика подключается к COM-порту компьютера, для отображения результатов выполнения заложенных функций. Так же в системе присутствует блок кнопок, осуществляющих некоторые функции системы. Микропроцессорная система на уровне черного ящика может иметь следующий вид (рис. 2.2).

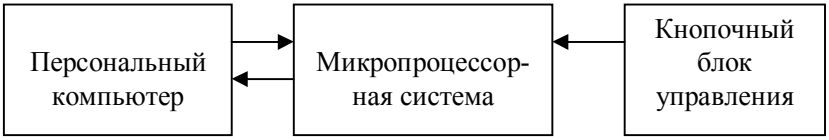


Рис. 2.2. Микропроцессорная система на уровне "черного ящика"

На структурном уровне микропроцессорная система должна осуществлять взаимодействие с ЭВМ посредством последовательного порта, осуществлять тестирование памяти, обрабатывать нажатие кнопок, производить расчет квадратного корня. Исходя из этих требований, составим структурную схему микропроцессорной системы (рис. 2.3).

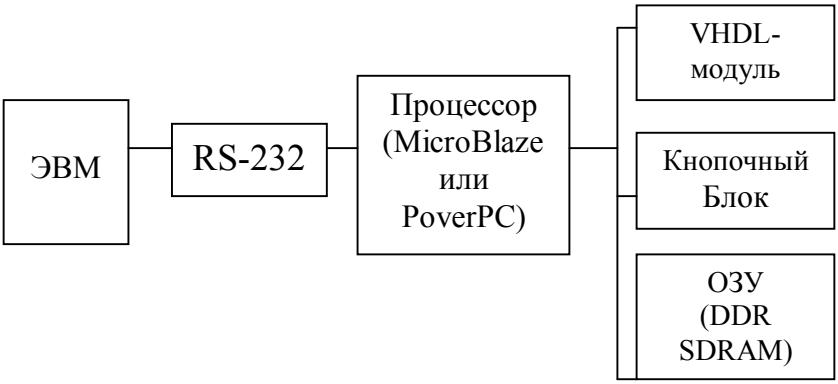


Рис. 2.3. Структурный уровень микропроцессорной системы

На программном уровне осуществляется проектирование алгоритма функционирования микропроцессорной системы (рис. 2.4).

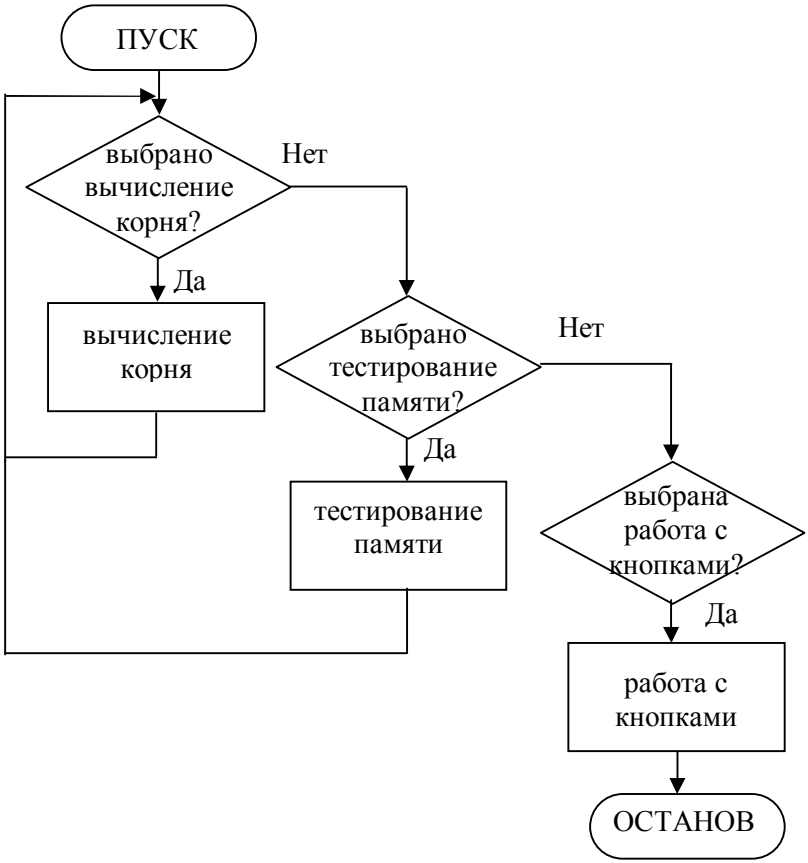


Рис. 2.4. Схема алгоритма работы микропроцессорной системы

2.2.3. Реализация микропроцессорной системы

Микропроцессорная система состоит из нескольких модулей. Структура взаимодействия модулей системы показана на рис. 2.5.

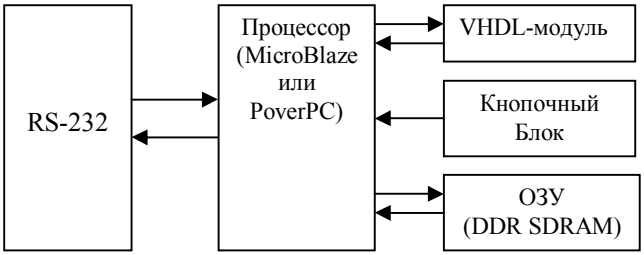


Рис. 2.5. Структура взаимодействия модулей микропроцессорной системы

Для реализации микропроцессорной системы используется отладочная плата Virtex-4 ML403.

Данная плата имеет необходимый набор периферийных устройств, позволяющих выполнять требуемые функции. Ядро микропроцессорной системы, алгоритм ее функционирования и VHDL-модуль прошиваются в ПЛИС. “Привязка” микропроцессорной системы к отладочной плате рассмотрена на рис. 2.6.

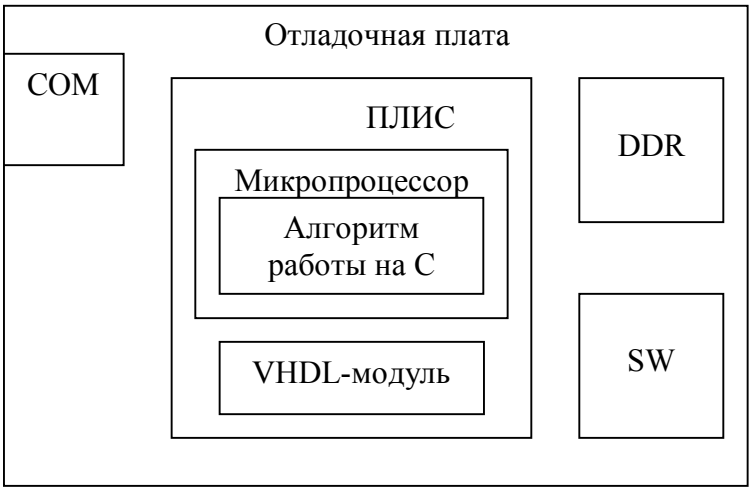


Рис. 2.6. “Привязка” микропроцессорной системы к отладочной плате

2.3. Разработка микропроцессорной системы

2.3.1. Создание проекта микропроцессорной системы

Работа с комплексом средств автоматизированного проектирования Xilinx embedded development kit начинается с запуска управляющей оболочки Xilinx platform studio с помощью кнопки пуск (start). После нажатия кнопки пуск (start) в открывшейся панели меню необходимо выбрать строку программы (programs), а затем в предложенном списке выбрать группу программ xilinx embedded development kit, в которой нужно выделить строку xilinx platform studio и щелкнуть на ней левой кнопкой мышки. При успешном выполнении указанных операций на экране монитора отображается основное окно xilinx platform studio, подробная структура которого рассмотрена ранее.

Для создания нового проекта следует выполнить команду file основного меню управляющей оболочки xilinx platform studio, а затем во всплывающем меню выбрать строку

new project. В результате открывается всплывающее меню следующего уровня иерархии, строки которого предоставляют разработчику два механизма подготовки нового проекта. На данном этапе можно создать новый проект или создать проект, используя настройки ранее созданных проектов. Создадим новый проект.

При выборе строки base system builder запускается «мастер» формирования новых аппаратных платформ base system build wizard, который позволяет проводить весь процесс разработки нового проекта в интерактивном режиме. Данный метод наиболее целесообразно применять при выполнении проектов, для аппаратной реализации которых используются отладочные и демонстрационные платы, поддерживаемые пакетом xilinx edk (рис. 2.7).

Для создания нового проекта обязательно должны быть определены следующие исходные данные:

- название проекта;
- диск и каталог, в котором предполагается расположить проект;
- семейство ПЛИС, на базе которого разрабатывается система;
- тип кристалла, используемого для реализации проектируемой системы;
- тип корпуса выбранной ПЛИС;
- категория быстродействия выбранного кристалла.

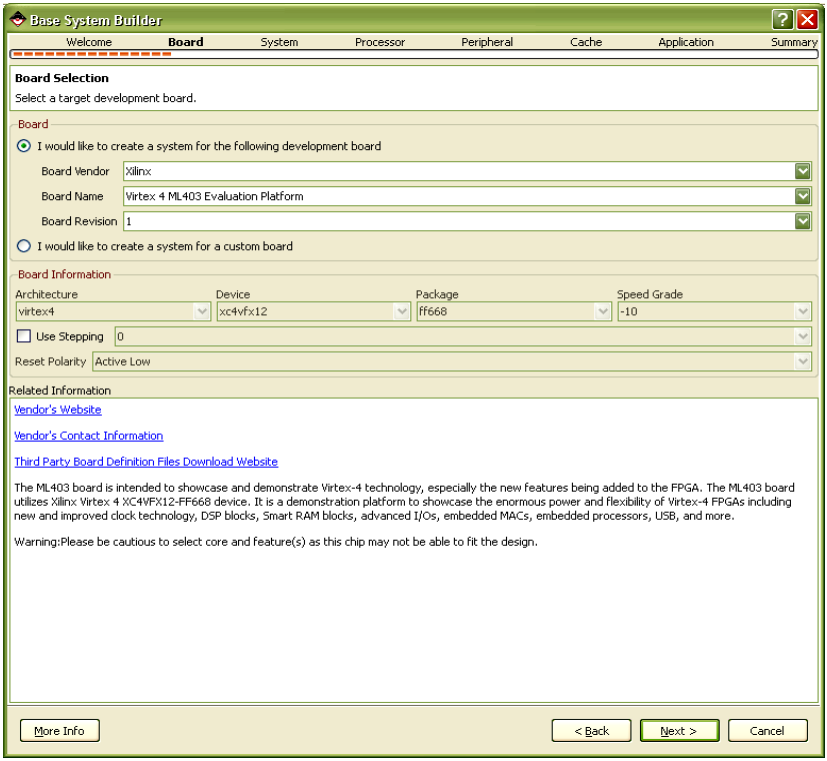


Рис. 2.7. Выбор аппаратной платформы микропроцессорной системы

Семейство ПЛИС, тип кристалла, тип корпуса ПЛИС и категория быстродействия кристалла определяются с помощью соответствующих полей выбора. При выборе названия платы эти поля заполняются автоматически.

На следующем этапе необходимо выбрать однопроцессорную или двухпроцессорную систему требуется создать (рис. 2.8).

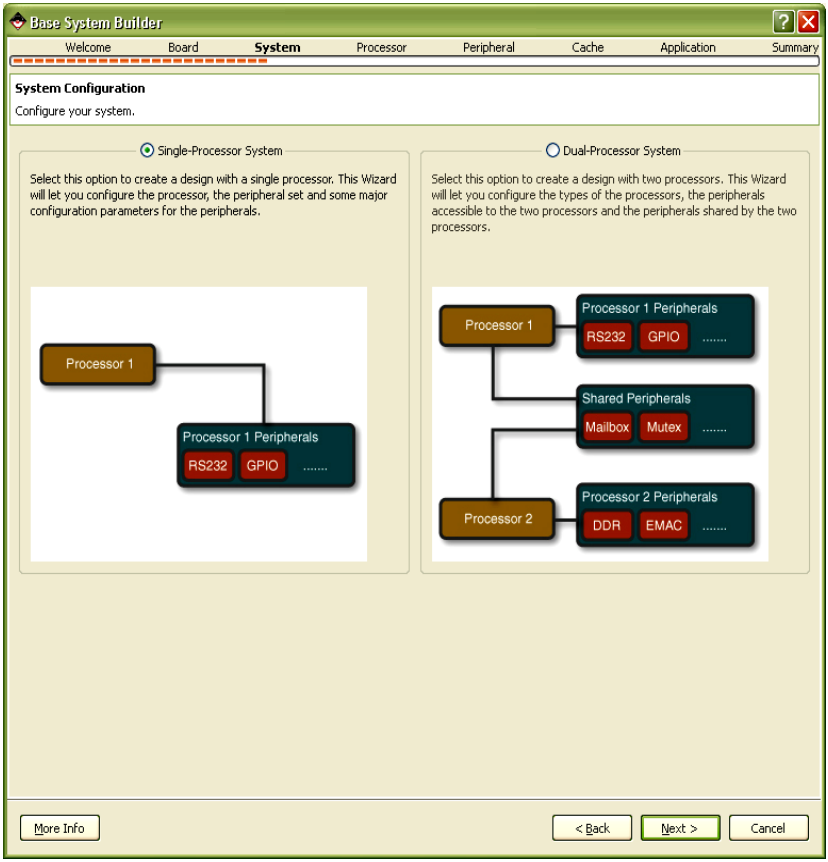


Рис. 2.8. Выбор числа процессоров микропроцессорной системы

На следующем шаге можно выбрать тип процессора: аппаратный процессор PowerPC или софт-процессор MicroBlaze. Выберем софт-процессор MicroBlaze с внутренней памятью 32Кб и перейдем к следующему этапу.

На этом этапе можно добавить или удалить ядро управления переферийным оборудованием, расположенным на плате, а также осуществить его предварительную настройку (рис. 2.9).

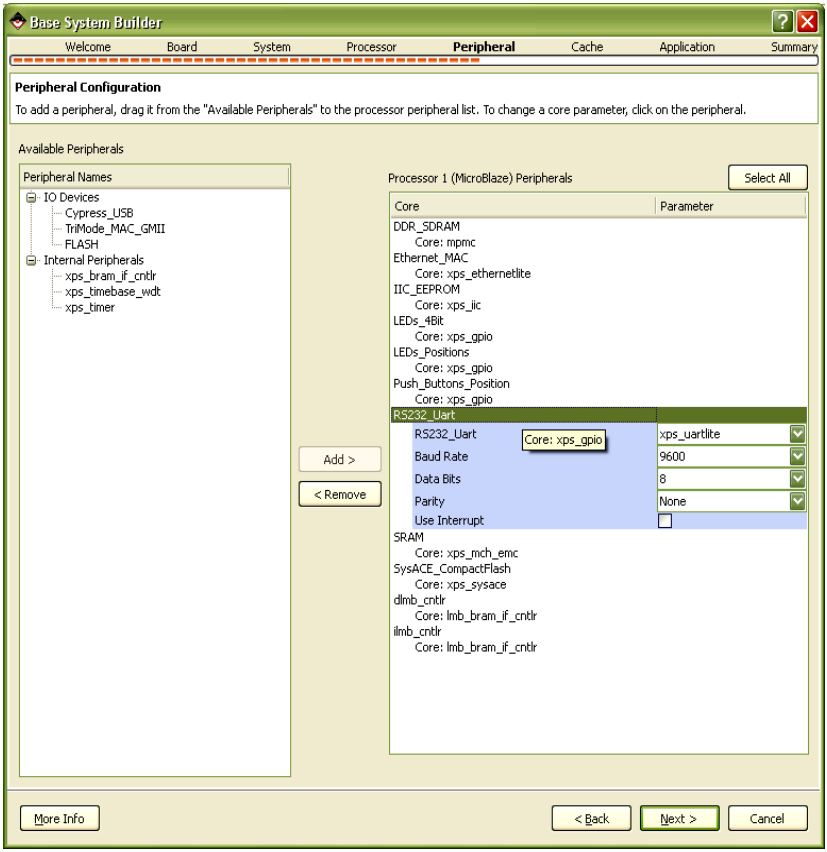


Рис. 2.9. Создание и редактирование периферийного оборудования

После выбора оборудования можно создать блоки кеш-памяти для команд и данных микропроцессора. Причем кеш-память маленького объема создается на базе LUT, а память большого объема на основе RAM.

На последнем этапе можно просмотреть дерево проекта и пространство адресов оборудования. После этого завершим процесс создания нового проекта микропроцессорной системы.

На основе выбранных настроек Xilinx EDK формирует блок-схему микропроцессорной системы (рис. 2.10) [7].

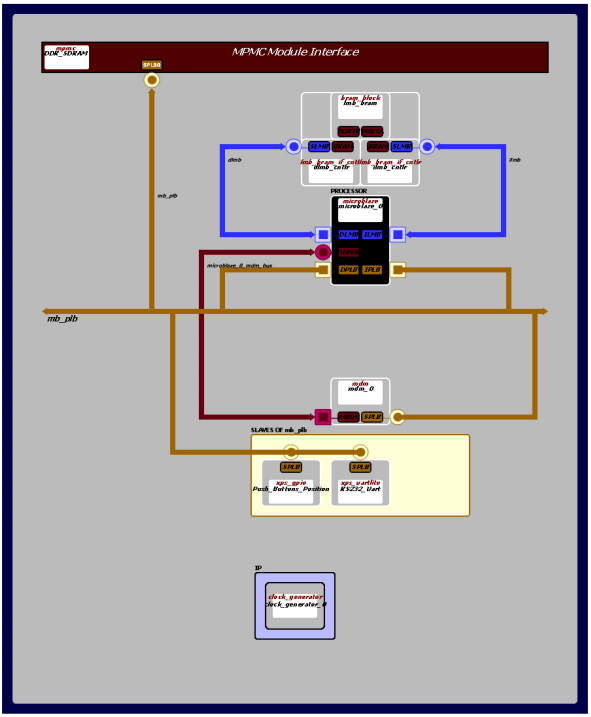


Рис. 2.10. Блок-схема микропроцессорной системы в Xilinx EDK

После определения всех необходимых параметров создаваемого проекта можно перейти к следующему этапу разработки встраиваемой микропроцессорной системы на основе ядра MicroBlaze — созданию спецификации аппаратной платформы.

2.3.2. Создание спецификации аппаратной платформы

Спецификация аппаратной платформы встраиваемой микропроцессорной системы MHS (Microprocessor Hardware

Specification) представляет собой файл с расширением MHS, в котором в текстовом формате описывается конфигурация и параметры аппаратных средств, проектируемой системы. Язык описания спецификации аппаратной платформы имеет много общего с языками описания аппаратуры HDL (Hardware Description Language) высокого уровня VHDL и Verilog. В частности, интерфейсные цепи разрабатываемой системы и ее компонентов описываются в виде портов. Как и в языках HDL высокого уровня понятие сигнала соответствует физической цепи, которая соединяет компоненты микропроцессорной системы. Каждый компонент, включаемый в состав спецификации аппаратной платформы, описывается с помощью следующей конструкции.

BEGIN <идентификатор_компонента>

Команда1<параметр1_команды1>=<значение_параметра 1> [,<параметр2_команды1>=<значение_параметра2>,…, параметрM_команды>= <значение_ параметраM>]

…

КомандаN<параметр1_командыN>=<значение_параметр а1> [,<параметр2_командыN>=<значение_параметра2>,…, <параметрK_командыN> = <значение_ параметраK>]

END

В приведенной конструкции квадратными скобками выделены параметры команд, наличие которых зависит от типа команды и не является обязательным.

Ключевое слово BEGIN обозначает начало описания нового компонента системы. Далее следует совокупность команд, с помощью которых указываются параметры компонента, включаемого в состав спецификации. При необходимости к данной совокупности добавляются команды, задающие границы адресного пространства, которое используется описываемым компонентом. Кроме того, здесь же присутствует группа команд, определяющих подключение данного компонента к шинам и цепям разрабатываемой системы. Общее число команд N в этой конструкции

обусловливается в первую очередь типом применяемого компонента. Количество параметров, указываемых в каждой команде, определяется типом этой команды. В дальнейшем параметры команд будем называть ключами, чтобы не путать с параметрами компонентов, включаемых в состав разрабатываемой системы. Блок описания компонента проектируемой системы завершается ключевым словом END.

В текущей версии спецификации аппаратной платформы используется три команды: PARAMETER, PORT и BUS_INTERFACE. С помощью команды PARAMETER определяются значения различных параметров спецификации и применяемых компонентов. Кроме того, эта команда используется для указания диапазона адресного пространства, используемого компонентами разрабатываемой микропроцессорной системы. Все параметры, определяемые командой PARAMETER, можно разделить на две группы: глобальные и локальные. Глобальные параметры относятся ко всей спецификации аппаратной платформы. Команды, которые определяют значения глобальных параметров, располагаются вне блоков описания компонентов BEGINEND.

Примером глобального параметра является параметр VERSION, который указывает номер версии спецификации аппаратной платформы встраиваемой микропроцессорной системы. Значение этого параметра обычно задается в начале файла MHS.

Локальные параметры относятся только к соответствующим компонентам проектируемой системы. Команды, определяющие значения локальных параметров, располагаются внутри блоков описания компонентов BEGIN-END. Значение параметра HW_VER указывает номер версии используемого IP-компонента.

Номера версий соответствующего IP-компонента указаны в файле документации для этого компонента. В качестве примера можно привести команду «PARAMETER HW_VER = 1.00.a», которая указывает на то, что в

соответствующем блоке BEGIN-END описывается экземпляр компонента версии 1.00.a.

С помощью параметра INSTANCE задается идентификатор описываемого экземпляра используемого IP-компонента. Для определения значения этого параметра следует включить в состав соответствующего блока описания BEGIN-END.

Диапазон адресов памяти, используемый описываемым экземпляром некоторого IP-компонента проектируемой системы, задается в виде базового и максимального значений адреса. Размер адресного пространства, выделяемого для компонента микропроцессорной системы, должен быть равен 2 в степени N. При этом последние N бит в значении базового адреса должны быть равны нулю. Значение базового адреса определяется с помощью параметра C_BASEADDR. Значения адреса, как правило, задаются в шестнадцатеричном представлении. Например, командная строка «PARAMETER C_BASEADDR= 0xFFFF0000» выделяет для описываемого экземпляра компонента диапазон адресного пространства, начинающийся с адреса FFFF0000. Верхняя граница адресного пространства памяти, используемого компонентом, указывается в виде значения параметра C_HIGHADDR.

Команда PORT предназначена для описания взаимосвязей между компонентами проектируемой системы, а также внешнего интерфейса системы. Эта команда может быть как глобальной, так и локальной. Глобальная команда PORT используется для описания внешних портов микропроцессорной системы, а также для определения значений постоянных сигналов. Глобальные команды PORT располагаются вне блоков описания компонентов BEGIN-END. С помощью ключа DIR указывается тип порта (сигнала), соответствующий направлению передачи данных. Для обозначения входного порта используется одно из следующих значений ключа DIR: INPUT, IN, I. В командах, описывающих выходные порты, ключ DIR должен принимать одно из трех

значений: OUTPUT, OUT, O. При описании двунаправленного интерфейсного порта указывается одно из двух возможных значений ключа DIR: INOUT или IO. Использование локальной команды PORT демонстрируют следующие примеры:

- команда «PORT Clk = sys_clk» описывает подключение порта Clk некоторого IP-компонента к цепи sys_clk;
- команда «PORT = net_gnd» описывает подключение порта In_reset некоторого IP-компонента к цепи «земля»;
- команда «PORT = ""» описывает порт Out_en, который находится в неподключенном состоянии.

Команда BUS_INTERFACE используется для описания взаимосвязей компонентов, осуществляемых на основе шинной архитектуры. Для подключения к шинам различные компоненты микропроцессорной системы используют одни и те же группы цепей (сигналов). При использовании команды PORT для их описания потребовалось бы достаточно большое количество командных строк, повторяющихся для каждого компонента, что привело бы к существенному увеличению времени подготовки и объема файла спецификации аппаратной платформы. Избежать этого позволяет применение для описания подключения компонентов к шинам проектируемой микропроцессорной системы команды BUS_INTERFACE.

Типовая структура спецификации аппаратной платформы проектируемой микропроцессорной системы, как правило, содержит следующие разделы:

- команда, указывающая номер версии спецификации MHS;
- описание внешнего интерфейса (глобальных портов) разрабатываемой системы;
- определение портов, подключаемых к цепям питания и земли;
- декларация и определение значений постоянных сигналов;

- блоки описания микропроцессорного ядра и компонентов, включаемых в состав разрабатываемой системы.

На основе этих данных можно создавать и редактировать спецификацию аппаратной платформы. Приведем в качестве примера часть созданной спецификации.

```
PARAMETER VERSION = 2.1.0
PORT fpga_0_RS232_Uart_RX_pin=
pga_0_RS232_Uart_RX_pin, DIR=I
PORT fpga_0_RS232_Uart_TX_pin=
pga_0_RS232_Uart_TX_pin, DIR=O
BEGIN xps_uartlite
 PARAMETER INSTANCE = RS232_Uart
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT RX = fpga_0_RS232_Uart_RX_pin
 PORT TX = fpga_0_RS232_Uart_TX_pin
END
```

2.3.3. Топологические ограничения проекта

Для получения законченного описания аппаратных средств, проектируемой микропроцессорной системы, необходимо кроме спецификации MHS (Microprocessor Hardware Specification) подготовить дополнительную информацию о топологических и временных ограничениях, которая используется средствами синтеза, размещения и трассировки проекта в кристалле.

Дополнительная информация, используемая программами синтеза, размещения и трассировки проекта аппаратной части микропроцессорной системы, задается в

форме файла временных и топологических ограничений User Constraints File (UCF). Этот файл имеет текстовый формат, каждая строка которого представляет собой выражение, описывающее соответствующий параметр (ограничение проекта). Для создания файла временных и топологических ограничений и внесения в него информации можно использовать встроенный текстовый редактор управляющей оболочки Xilinx Platform Studio (XPS) или специальную программу Constraints Editor, которая входит в состав систем проектирования серии Xilinx ISE (Integrated Synthesis Environment).

Программа Constraints Editor в диалоговом режиме, автоматически формирует соответствующие выражения для описания ограничений проекта. Кроме того, для этих целей может использоваться редактор назначения выводов кристалла и топологических ограничений PACE (Pinout and Area Constraints Editor), который также входит в состав САПР серии ISE.

В файлах ограничений проектов встраиваемых микропроцессорных систем наиболее часто применяются два типа выражений. Ограничения первого типа позволяют установить соответствие между внешними цепями проектируемой системы и номерами выводов ПЛИС. Выражения второго типа используются для определения максимального значения периода сигнала синхронизации.

Параметр LOC позволяет осуществить закрепление выводов перед трассировкой, а также явно указать конфигурируемый логический блок (Configurable Logic Block, CLB) для реализации элементов проекта. Для привязки внешних цепей проектируемой системы (подключаемых к контактам кристалла) к требуемым выводам ПЛИС используется следующий формат выражения:

```
NET fpga_0_RS232_Uart_RX_pin LOC=W2;
NET fpga_0_RS232_Uart_TX_pin LOC=W1;
```

2.3.4. Разработка спецификации программных средств

Спецификация программной платформы встраиваемой микропроцессорной системы Microprocessor Software Specification (MSS) представляет собой файл с расширением MSS. Данный файл имеет текстовый формат и содержит описание конфигурации и параметров программных компонентов проектируемой системы (операционной системы, библиотек и драйверов). Спецификация MSS определяет драйверы для каждого периферийного модуля, стандартные устройства ввода-вывода и программы обработки прерываний. Спецификация программных средств микропроцессорной системы является исходной информацией для генератора библиотек Library Generator (Libgen).

Язык и структура описания программных средств встраиваемой микропроцессорной системы в общих чертах напоминает язык описания спецификации аппаратной платформы MHS (Microprocessor Hardware Specification), который был рассмотрен выше.

Каждый компонент, включаемый в состав MSS, описывается с помощью конструкции, которая начинается с ключевого слова BEGIN и заканчивается ключевым словом END.

Типовая структура спецификации программной платформы проектируемой микропроцессорной системы, как правило, содержит следующие разделы:

- команда, указывающая номер версии спецификации MSS;
- описание используемой операционной системы;
- определение параметров микропроцессорного ядра;
- блоки описания драйверов периферийных устройств;
- блоки описания используемых библиотек.

На основе этих данных можно создавать и редактировать спецификацию программной платформы. Приведем в качестве примера часть созданной спецификации.

PARAMETER VERSION = 2.2.0

```
BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.12.b
 PARAMETER HW_INSTANCE = microblaze_0
 PARAMETER COMPILER = mb-gcc
 PARAMETER ARCHIVER = mb-ar
END
BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = RS232_Uart
END
```

2.3.5. Разработка программы управления МПС

Для разработки программы управления во вкладку Sources проекта необходимо добавить файл с расширением *.C.

Каждая прикладная программа для встраиваемой микропроцессорной системы на основе ядра MicroBlaze разрабатывается в виде соответствующего программного проекта (Software project). Такой проект может включать следующие компоненты:

- файлы, содержащие исходный текст прикладной программы (source files);
- заголовочные файлы (header files);
- файл директив для компоновщика (linker scripts).

Для того чтобы открыть исходный файл в интегрированном текстовом редакторе средств проектирования XST, на странице Applications следует дважды щелкнуть левой кнопкой мыши на строке с его названием. При этом в управляющей оболочке XPS открывается новое рабочее окно редактирования, на закладке которого указано название выбранного файла. В этом окне с помощью клавиатуры формируем исходный текст прикладной программы.

Программа выполняет циклическое чтение данных, поступающих из параллельного порта ввода-вывода, вызывает модуль вычисления квадратного корня, разработанный на языке VHDL, осуществляет тестирование памяти, фиксирует нажатие кнопок. Данные о выполнении функций передаются по последовательному порту RS-232 персональный компьютер. После завершения редактирования исходного файла необходимо сохранить его, используя команду Save из всплывающего меню, которое открывается при выборе пункта File основного меню управляющей оболочки.

Для отображения поступающих данных и сообщений на экране монитора персонального компьютера может использоваться программа HyperTerminal, которая входит в состав операционных систем семейства Windows NT.

Процесс компиляции и компоновки прикладной программы выполняется в автоматическом режиме. Информация о его результатах отражается на странице Output встроенного окна консольных сообщений управляющей оболочки XPS. При обнаружении ошибок сведения о них выводятся на странице Errors [5].

2.3.6. Разработка модуля вычисления квадратного корня

Для разработки модуля вычисления квадратного корня воспользуемся программой CORE Generator. Для этого создадим проект ISE. Выберем пункт меню Tools-CORE Generator… и выберем модуль вычисления квадратного корня (рис. 2.11).

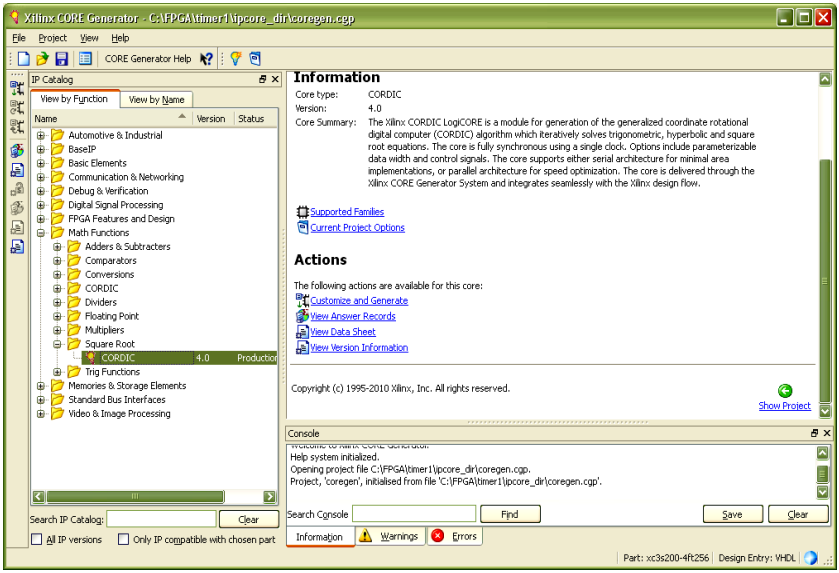


Рис. 2.11. Выбор модуля вычисления квадратного корня

После загрузки модуля осуществим его настройку (рис. 2.12).

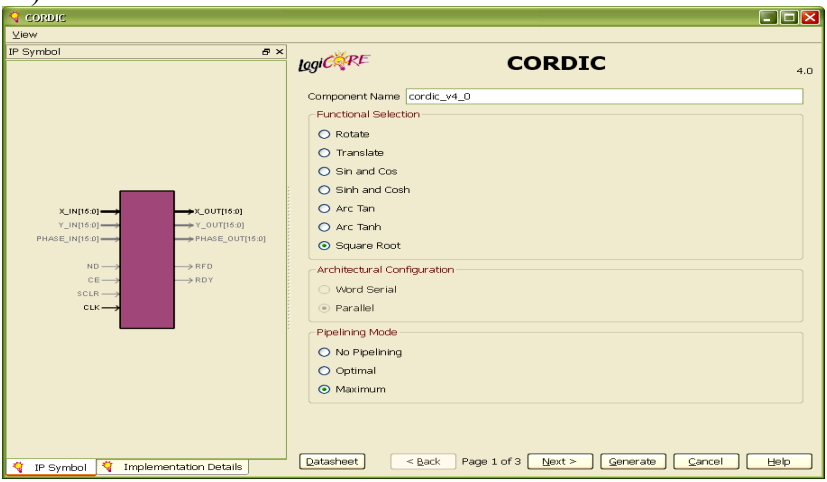


Рис. 2.12. Настройка модуля вычисления квадратного корня

После завершения настройки нажмем кнопку Generate. Модуль будет создан. Для его использования необходимо

написать проект на языке VHDL, который осуществит подключение данного модуля к проекту ISE. Для этого необходимо описать интерфейс данного модуля, его компоненту и архитектуру.

В общем виде объявление интерфейса выглядит следующим образом:

```
entity NAME_ENTITY is
port (-- Здесь указываются порты);
begin --ставится, если далее следуют параллельные операторы интерфейса
-- Параллельные операторы
end NAME_ENTITY;
```

Архитектура содержит две основные части:

- часть, содержащую описания (декларации);
- часть, содержащую исполняемые операторы.

В общем виде архитектура выглядит следующим образом:

```
architecture NAME_ARCHITECTURE of NAME_ENTITY is
-- описания типов данных;
-- функции и процедуры;
-- компоненты более низкого уровня иерархии;
-- описания сигналов и глобальных переменных.
begin
-- Здесь содержатся исполняемые операторы.
end NAME_ARCHITECTURE;
```

Использование компонентов позволяет создавать несколько объектов с одним интерфейсом. В общем виде описание компонента выглядит следующим образом:

```
component COMPONENT
port (  -- Здесь указываются порты);
end component;
```

На основе этого был разработан VHDL-модуль вычисления квадратного корня.

После тестирования работоспособности полученный модуль необходимо подключить к проекту EDK и выполнить подключение шин данных (рис. 2.13).

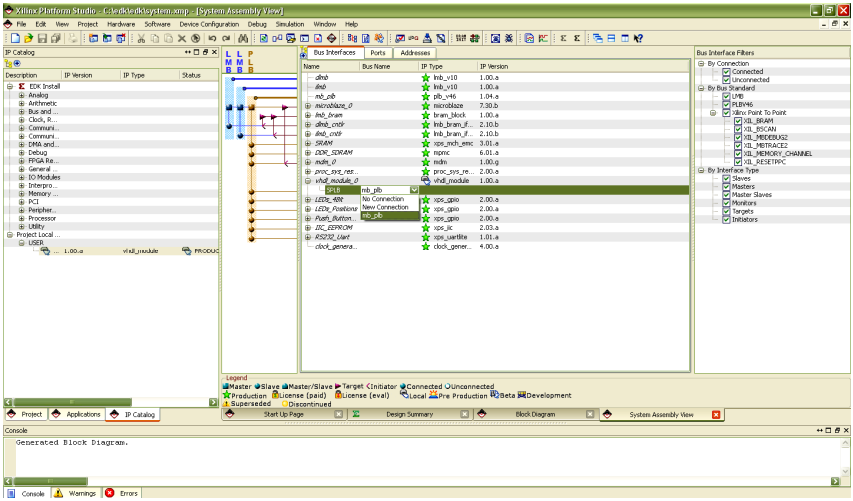


Рис. 2.13. Подключение VHDL-модуля

Также необходимо настроить *.MPD файл. В этом файле осуществляется подключение и настройка модуля. В качестве примера приведем ту часть, где осуществляется его настройка.

```
## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION IP_GROUP = MICROBLAZE:PPC:USER
OPTION DESC = VHDL_MODULE
OPTION STYLE = MIX
OPTION RUN_NGCBUILD = TRUE
```

2.3.7. Формирование списка соединений

Данный этап включает в себя две фазы. В первой фазе осуществляется генерация модулей HDL-описаний в

соответствии с подготовленной спецификацией аппаратной платформы проектируемой системы. Вторая фаза представляет собой процедуру синтеза аппаратной части встраиваемой микропроцессорной системы, в процессе которой производится трансформация исходных модулей HDL-описания в список цепей (netlist), выполненный на низком логическом уровне. Элементы низкоуровнего описания, формируемого в процессе синтеза, должны соответствовать архитектуре семейства ПЛИС, выбранного для реализации проекта. Синтезированный список цепей должен быть максимально адаптирован к ресурсам используемого кристалла, что обеспечивает его наиболее эффективное отображение средствами размещения и трассировки на физическом уровне. В процессе синтеза выполняется оптимизация полученного списка цепей в соответствии с выбранным критерием. Основным результатом фазы синтеза является формирование файла NGC, который представляет собой описание проекта аппаратной части системы на низком логическом уровне в двоичном формате. Процедура синтеза выполняется теми средствами, которые были выбраны при определении параметров проекта с помощью опции Synthesis Tool. В качестве инструмента синтеза, применяемого по умолчанию, используются средства Xilinx Synthesis Tool (XST), которые являются составной частью системы автоматизированного проектирования Foundation ISE.

Этап генерации списка соединений аппаратной платформы выполняется в автоматическом режиме. Для активизации этого процесса следует выбрать команду Generate Netlist из всплывающего меню Tools управляющей оболочки Xilinx Platform Studio или воспользоваться кнопкой быстрого доступа, расположенной на оперативной панели управления XPS. Информация о ходе выполнения каждой фазы рассматриваемого этапа проектирования отображается во встроенном окне консольных сообщений на странице с закладкой Output. В случае обнаружения ошибок

соответствующие сообщения выводятся на странице Errors окна консольных сообщений, а предупреждения — на странице Warnings.

При успешном завершении всех процедур данного этапа создается файл NGC, который содержит результаты синтеза проекта. Следующим этапом разработки является функциональное моделирование аппаратной части. Этот процесс выполняется в рамках пакета моделирования ModelSim и практически не отличается от моделирования различных цифровых устройств, проектируемых на основе ПЛИС фирмы Xilinx.

2.3.8. Компиляция и компоновка проекта МПС

Этап реализации (Implementation) проекта аппаратной части встраиваемой микропроцессорной системы, реализуемой на базе ПЛИС семейств FPGA, включает в себя три фазы: трансляция (Translate), отображение логического описания проекта на физические ресурсы кристалла (MAP), размещение и трассировка (Place and Route). В результате выполнения этапа создается двоичный файл, который описывает использование физических ресурсов кристалла для реализации компонентов (функций) разрабатываемой системы и выполнения необходимых соединений между ними. Этот файл затем используется в качестве исходного для генерации конфигурационной последовательности ПЛИС. При достижении успешных результатов размещения и трассировки можно перейти непосредственно к этапу формирования конфигурационной последовательности для проекта аппаратной платформы разрабатываемой микропроцессорной системы. Информация о ходе его выполнения отображается в окне консольных сообщений и строке состояния Навигатора проекта. При успешном завершении этого процесса, отмеченном соответствующей пиктограммой в строке Generate Programming File в окне процессов, создается файл конфигурационного битового потока (имеющий расширение

.bit) для аппаратной части разрабатываемой микропроцессорной системы. После окончания этапа реализации можно закрыть окно Навигатора проекта и вернуться в среду управляющей оболочки XPS для выполнения дальнейших этапов разработки.

После создания всех необходимых исходных файлов можно приступить к компиляции и компоновке прикладной программы. Процесс компиляции и компоновки прикладной программы выполняется в автоматическом режиме. Информация о его результатах отражается на странице Output встроенного окна консольных сообщений управляющей оболочки XPS. При обнаружении ошибок сведения о них выводятся на странице Errors.

В случае успешного завершения процедур компиляции и компоновки создается файл executable.elf, который содержит исполняемый код прикладной программы. Полученный файл используется для дополнения файла конфигурационной последовательности, который был сформирован на этапе реализации аппаратной части разрабатываемой микропроцессорной системы.

Чтобы включить исполняемый код прикладной программы в конфигурационную последовательность ПЛИС, реализующую функции аппаратной части микропроцессорной системы, следует выполнить команду Tools основного меню управляющей оболочки XPS.

При этом на экран выводится меню, в котором нужно выбрать команду Update Bitstream. Аналогичную функцию выполняет кнопка быстрого доступа, которая находится на оперативной панели управления XPS. При исполнении данной команды в конфигурационной последовательности проекта микропроцессорной системы производится перезапись содержимого блочной памяти ПЛИС. Информация о выполнении процесса преобразования конфигурационной последовательности отображается на странице Output встроенного окна консольных сообщений. При отсутствии

ошибок создается новый файл конфигурационной последовательности download.bit, который записывается в раздел implementation рабочего каталога проекта разрабатываемой микропроцессорной системы. Этот файл может непосредственно использоваться для конфигурирования кристалла FPGA.

Управление процессом загрузки конфигурационных данных в ПЛИС может осуществляться двумя способами: непосредственно из оболочки XPS или в среде программы Impact, которая входит в состав САПР серии Xilinx ISE.

Загрузка конфигурационной последовательности проекта разрабатываемой микропроцессорной системы в ПЛИС осуществляется с помощью программы Impact и загрузочного JTAG-кабеля, подключаемого к параллельному порту (LPT) или к шине USB компьютера.

2.3.9. Отладка и тестирование

Для проверки функционирования аппаратной части разработанной системы сбора данных и тестовой программы воспользуемся отладочной платой из инструментального комплекта Virtex-4 ML403. Прежде всего, к этой плате необходимо подключить загрузочный кабель, а также кабель, соединяющий разъем интерфейса с разъемом последовательного порта персонального компьютера. Далее следует подать напряжение питания на отладочную плату. После этого следует активизировать приложение HyperTerminal (Hypertrm.exe) и создать новое подключение, указав соответствующий COM-порт компьютера и установив требуемые параметры протокола передачи данных. Учитывая значения параметров последовательного асинхронного приемопередатчика, которые были заданы при создании спецификации аппаратной платформы проектируемой системы MHS (Microprocessor Hardware Specification), необходимо установить следующие настройки COM-порта:

- скорость передачи, равную 9600 бит/с;

- количество передаваемых бит данных — 8;
- количество стоповых бит — 1;
- контроль на четность или нечетность отсутствует.

Выполнив загрузку конфигурационной последовательности проекта, следует убедиться в его работоспособности. Для этого выполняется отработка всех функций микропроцессорной системы. Взаимодействие микропроцессорной системы и персонального компьютера осуществляется посредством программы HyperTerminal. При подключении микропроцессорной системы к COM-порту компьютера система выдает приветствие и ожидает выбора режима работы (рис. 2.14).

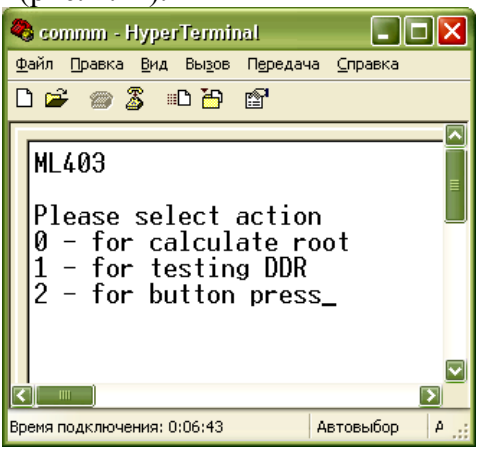


Рис. 2.14. Выбор режима работы микропроцессорной системы

Поочередно проводится проверка функционирования всех блоков микропроцессорной системы (рис. 2.15-2.17).

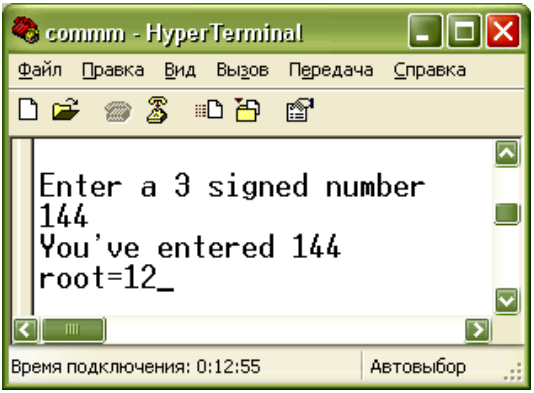


Рис. 2.15. Модуль вычисления квадратного корня

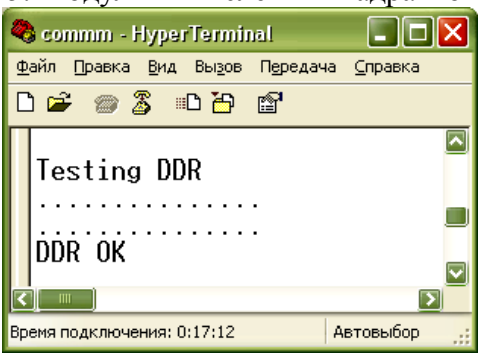


Рис. 2.16. Модуль тестирования памяти

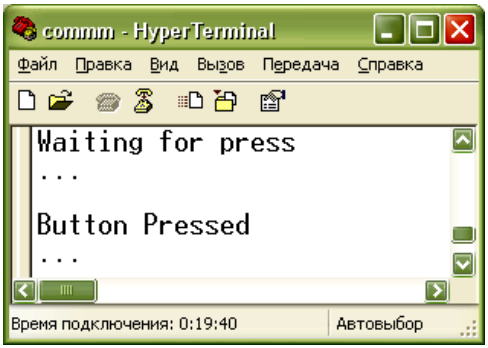


Рис. 2.17. Модуль обработки нажатия кнопок

Алгоритм функционирования микропроцессорной системы являет собой бесконечный цикл. После отработки любого из модулей система переходит в режим ожидания выбора следующего модуля. Это завершающий этап разработки микропроцессорной системы. Система готова к использованию. Т.к. алгоритм ее функционирования написан на языке С, то данная система может быть достаточно легко модифицирована и дополнена. Использование VHDL-модулей позволяет создавать функциональные блоки, узконаправленного действия, что делает систему очень удобной для решения многих прикладных задач.

Полный листинг данного модуля представлен в прил. 2.

ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРОТОКОЛОВ СВЯЗИ ЭЛЕКТРОННЫХ УСТРОЙСТВ

В настоящее время широкое распространение получили следующие протоколы связи электронных устройств CAN, LIN, UART и др.

CAN (англ. *Controller Area Network* — сеть контроллеров) — стандарт промышленной сети, ориентированный прежде всего на объединение в единую сеть различных исполнительных устройств и датчиков. Режим передачи — последовательный, широковещательный, пакетный.

LIN - последовательный коммуникационный протокол. LIN протокол отличается от CAN протокола тем, что его возможности меньше, а область применения уже, чем у CAN протокола. А его сильной стороной является то, что он проще и дешевле, что важно для приложений, не требующих больших возможностей CAN протокола.

Универсальный асинхронный приёмопередатчик (УАПП, англ. Universal Asynchronous Receiver-Transmitter (UART)) — узел вычислительных устройств, предназначенный для связи с другими цифровыми устройствами. Преобразует

заданный набор данных в последовательный вид так, чтобы было возможно передать их по однопроводной цифровой линии другому аналогичному устройству. При этом интервалы времени между передаваемыми блоками данных не являются постоянными: блоки данных выделяются с помощью стартовых и стоповых битов (Асинхронная передача данных). Метод преобразования хорошо стандартизован и широко применялся в компьютерной технике.

Обзор распространенных протоколов связи электронных устройств

Протокол CAN

CAN разработан компанией Robert Bosch GmbH в середине 1980-х и в настоящее время широко распространён в промышленной автоматизации, технологиях «умного дома», автомобильной промышленности и многих других областях. Стандарт для автомобильной автоматики.

Непосредственно стандарт CAN от Bosch определяет передачу в отрыве от физического уровня — он может быть каким угодно, например, радиоканалом или оптоволокном. Но на практике под CAN-сетью обычно подразумевается сеть топологии «шина» с физическим уровнем в виде дифференциальной пары, определённым в стандарте ISO 11898. Передача ведётся кадрами, которые принимаются всеми узлами сети. Для доступа к шине, выпускаются специализированные микросхемы — драйверы CAN шины.

Синхронная шина, с типом доступа Collision Resolving (CR, разрешение коллизии), который в отличие от Collision Detect (CD, обнаружение коллизии) сетей детерминировано (приоритетно) обеспечивает доступ на передачу сообщения, что особо ценно для промышленных сетей управления. Передача ведётся кадрами. Полезная информация в кадре состоит из идентификатора длиной 11 бит (стандартный формат) или 29 бит (расширенный формат, надмножество

предыдущего) и поля данных длиной от 0 до 8 байт. Идентификатор говорит о содержимом пакета и служит для определения приоритета при попытке одновременной передачи несколькими сетевыми узлами.

Для абстрагирования от среды передачи спецификация CAN избегает описывать двоичные значения как «0» и «1». Вместо этого применяются термины «рецессивный» и «доминантный», при этом подразумевается, что при передаче одним узлом сети рецессивного бита, а другим доминантного, принят будет доминантный бит. Например, при реализации физического уровня на радиоканале отсутствие сигнала означает рецессивный бит, а наличие — доминантный; тогда как в типичной реализации проводной сети рецессив бывает при наличии сигнала, а доминант, соответственно, при отсутствии.

Стандарт сети требует от «физического уровня», фактически, единственного условия: чтобы доминантный бит мог подавить рецессивный, но не наоборот. Например, в оптическом волокне доминантному биту должен соответствовать «свет», а рецессивному — «темнота». В электрическом проводе может быть так: рецессивное состояние — высокое напряжение на линии (от источника с большим внутренним сопротивлением), доминантное — низкое напряжение (все узлы сети «подтягивают» линию на землю). Если линия находится в рецессивном состоянии, перевести её в доминантное может любой узел сети (включив свет в оптоволокне или закоротив высокое напряжение). Наоборот — нельзя (включить темноту нельзя) [8].

Виды кадров используемые в CAN:

– Кадр данных (data frame) — передаёт данные;

– Кадр удаленного запроса (remote frame) — служит для запроса на передачу кадра данных с тем же идентификатором;

– Кадр перегрузки (overload frame) — обеспечивает промежуток между кадрами данных или запроса;

– Кадр ошибки (error frame) — передаётся узлом, обнаружившим в сети ошибку.

Кадры данных и запроса отделяются от предыдущих кадров межкадровым промежутком.

Таблица 3.1

Базовый формат кадра данных протокола CAN

Поле	Длина (в битах)	Описание
Начало кадра	1	Сигнализирует начало передачи кадра
Идентификатор	11	Уникальный идентификатор
Запрос на передачу (RTR)	1	Должен быть доминантным
Бит расширения идентификатора (IDE)	1	Должен быть доминантным
Зарезервированный бит (r0)	1	Резерв
Длина данных (DLC)	4	Длина поля данных в байтах (0-8)
Поле данных	0-8 байт	Передаваемые данные (длина в поле DLC)
Контрольная сумма (CRC)	15	Контрольная сумма всего кадра
Разграничитель контрольной суммы	1	Должен быть рецессивным
Промежуток подтверждения (ACK)	1	Передатчик шлёт рецессивный, приёмник вставляет доминанту
Разграничитель подтверждения	1	Должен быть рецессивным
Конец кадра (EOF)	7	Должен быть рецессивным

Первые 7 бит идентификатора не должны быть все рецессивными.

Таблица 3.2

Расширенный формат кадра данных

Поле	Длина (в битах)	Описание
Начало кадра	1	Сигнализирует уачало передачи кадра
Идентификатор A	11	Первая часть идентификатора
Подмена запроса на передачу (SRR)	1	Должен быть рецессивным
Бит расширения идентификатора (IDE)	1	Должен быть рецессивным
Идентификатор B	18	Вторая часть идентификатора
Запрос на передачу (RTR)	1	Должен быть доминантным
Зарезервированные биты (r1 и r0)	2	Резерв
Длина данных (DLC)	4	Длина поля данных в байтах (0-8)
Поле данных	0-8 байт	Передаваемые данные (длина в поле DLC)
Контрольная сумма (CRC)	15	Контрольная сумма всего кадра
Разграничитель контрольной суммы	1	Должен быть рецессивным
Промежуток подтверждения (ACK)	1	Передатчик шлёт рецессивный, приёмник вставляет доминанту
Разграничитель подтверждения	1	Должен быть рецессивным
Конец кадра (EOF)	7	Должен быть рецессивным

Идентификатор получается объединением частей A и B.

Преимущества протокола CAN:

– возможность работы в режиме жёсткого реального времени;

– простота реализации и минимальные затраты на использование;

– высокая устойчивость к помехам;

– арбитраж доступа к сети без потерь пропускной способности;

– надёжный контроль ошибок передачи и приёма;

– широкий диапазон скоростей работы;

– большое распространение технологии, наличие широкого ассортимента продуктов от различных поставщиков.

Недостатки:

– небольшое количество данных, которое можно передать в одном пакете (до 8 байт);

– большой размер служебных данных в пакете (по отношению к полезным данным);

– отсутствие единого общепринятого стандарта на протокол высокого уровня, однако же, это и достоинство. Стандарт сети предоставляет широкие возможности для практически безошибочной передачи данных между узлами, оставляя разработчику возможность вложить в этот стандарт всё, что туда сможет поместиться. В этом отношении CAN подобен простому электрическому проводу. Туда можно «затолкать» любой поток информации, который сможет выдержать пропускная способность шины. Известны примеры передачи звука и изображения по шине CAN (Россия).

Протокол реализуется ИМС AT89C51CC03 производства фирмы Atmel, микроконтроллерами семейства PICmicro18 фирмы MICROCHIP и SJA1000 фирмы Philips.

Протокол LIN

Основные свойства шины LIN:

– один ведущий, множество подчиненных;

– низкая стоимость полупроводниковых приборов;

– длина шины до 40 м;

– реализуется с помощью простого последовательного интерфейса uart/sci;

– для синхронизации подчиненных узлов не требуются кварцевые или керамические резонаторы;

– гарантия времени ожидания для передачи сигналов;

– напряжение на шине в пассивном состоянии 9...18 в (подключенные к шине узлы должны выдерживать повышение напряжения до 40 в);

– однопроводная линия связи;

– скорость передачи данных до 20 кбит/с.

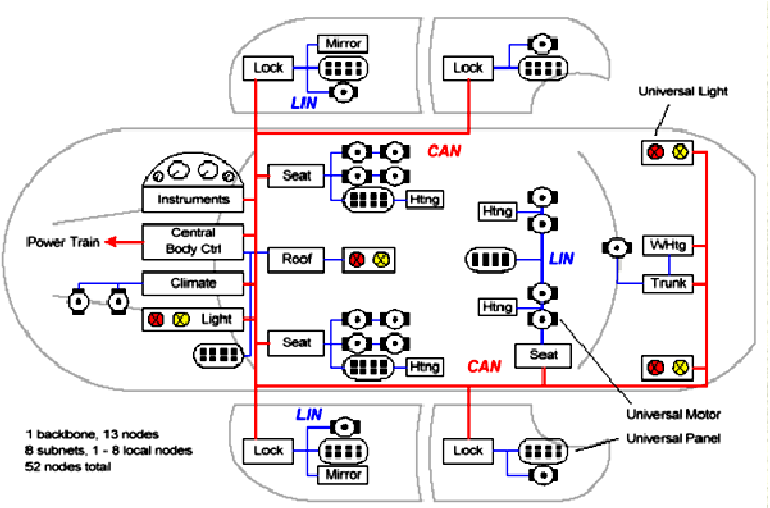


Рис. 3.1. Применение сетей LIN

Протокол LIN не использует управление доступом к шине. За передачу всех сообщений отвечает ведущий. Все подчиненные (ведомые) могут реагировать на сообщения

ведущего или других узлов сети, но отвечать они могут только будучи адресованными ведущим и получив от него разрешение.

Шина LIN состоит из одного канала, по которому передаются синхросигналы и данные. Физическая среда этого канала - однопроводная линия, подключенная через подтягивающий резистор к шине питания VCC. Высокий или "рецессивный" сигнал на шине показывает, что она свободна, а низкий или "доминантный" - что она занята. Обычно в автомобильных системах VCC является положительным потенциалом.

LIN протокол не определяет процедуру подтверждения для ведомых задач. Ведущая задача использует свою собственную подчиненную задачу для определения того, что посланный кадр сообщения идентичен кадру, полученному подчиненной задачей. Если обнаруживается несоответствие, то кадр может быть передан еще раз.

В соответствии с требованиями к уровню паразитных ЭМИ, возникающих при передаче данных по однопроводным линиям связи, скорость передачи данных равняется 20 кбит/с.

Вся информация, передаваемая по LIN шине, разделена на кадры. Как показано в рис. 3.2, кадр сообщения состоит из следующих полей:

- Разрыв синхронизации;
- Поле синхронизации;
- Идентификационное поле;
- Поле данных;
- Поле контрольной суммы.

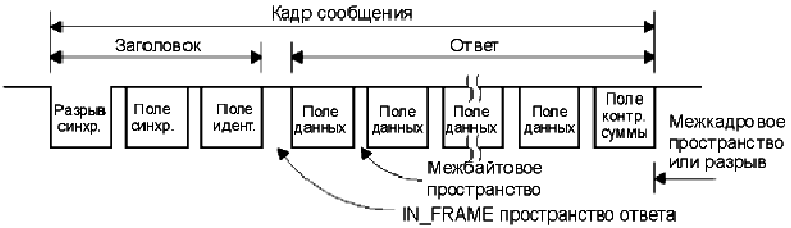


Рис. 3.2. Кадр сообщения LIN

Кадр сообщения состоит из двух частей: "заголовок", посланный ведущим и "ответ", который может формироваться как ведущей, так и подчиненной задачей.

Формат байтовых полей, идентичен формату данных стандарта UART с кодированием 8N1. Это означает, что каждое байтовое поле содержит старт бит, 8 бит данных, стоп бит и не содержит бита проверки на четность. Таким образом, каждое байтовое поле содержит 10 бит (Tbit). Как показано на рис. 3.3, старт бит указывает на начало байтового поля и является "доминирующим", в то время как стоп бит является "рецессивным". Восемь информационных битов могут быть как "доминирующими", так и "рецессивными".

Рис. 3.3. Байтовое поле LIN

Поле разрыва синхронизации указывает на начало передачи кадра сообщения. Это поле всегда передается ведущей задачей и указывает на необходимость подчиненным подготовиться к приему поля синхронизации. Поле разрыва синхронизации состоит из двух частей: первая часть имеет доминирующей (низкий) уровень, длительность которого

должна быть не менее 13 периодов байтового поля (Tbit), а вторая - рецессивной (высокий) уровень, длительность которого должна быть не менее четырех Tbit. Вторая часть поля необходима для обеспечения возможности обнаружения стартового бита следующего поля синхронизации.

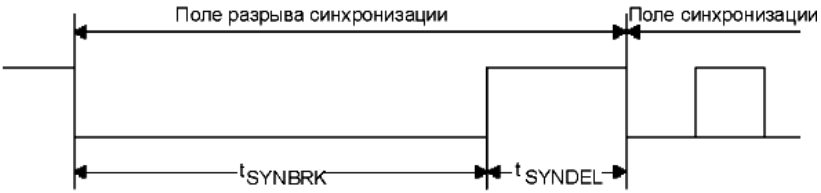


Рис. 3.4. Поле разрыва синхронизации

Длительность первого поля выбрана таким образом, чтобы обеспечить различие между полем разрыва синхронизации и максимально возможной допустимой последовательностью доминантных битов в пределах одного кадра данных. Например, поле данных, состоящее из одних нулей, не должно быть ошибочно принято за поле разрыва синхронизации.

После отправляется поле синхронизации. Поле синхронизации содержит сигналы, необходимые для синхронизации задающих генераторов подчиненных устройств сети. Поле синхронизации является байтовым полем, имеющем значение "0x55". Временная диаграмма этого поля приведена на рис. 3.5. Как видно из рисунка, поле синхронизации содержит пять спадающих фронтов (пять переходов от рецессивного к доминантному состоянию). Эти спадающие фронты должны использоваться подчиненными устройствами для настройки скорости обмена данными.

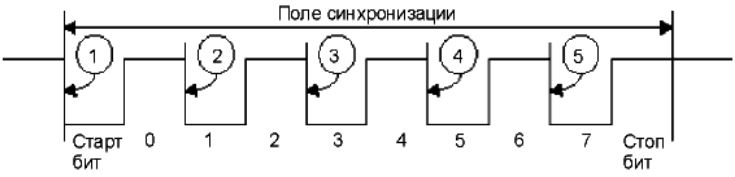


Рис. 3.5. Поле синхронизации

Идентификационное поле содержит информацию о содержимом и длине сообщения. Как показано на рис. 3.6, это поле разделено на три секции: идентификационные биты (4 бита), служебные биты, указывающие на длину сообщения (2 бита) и биты проверки на четность (2 бита). Таким образом происходит деление 64 идентификационных номеров на четыре комплекта, каждый из которых содержит 16 идентификаторов.

Рис. 3.6. Идентификационное поле

В соответствии с LIN протоколом количество полей данных в кадре данных определяется по приведенной ниже табл. 3.3.

Таблица 3.3

Количество полей данных в кадре данных

ID5	ID4	NDATA (количество полей данных)
0	0	2
0	1	2
1	0	4
1	1	8

Как видно из табл. 3.3 есть две группы с двумя полями данных, одна группа с четырьмя полями данных и одна группа с восьмью полями данных. Обратите внимание, что в идентификационном поле не описывается длина сообщения, а только указывается содержание кадра сообщения. Это позволяет приемникам подчиненных задач определить все ли были приняты данные или нет. Последние два бита идентификационного поля являются битами проверки на четность. LIN использует алгоритм смешанного контроля четности, который гарантирует, что идентификационное поле никогда не будет состоять из одних "рецессивных" или "доминантны" символов. Следует заметить, что этот алгоритм позволяет только обнаруживать ошибки, но не исправляет их.

Биты проверки на четность формируются в соответствии с приведенными ниже формулами:

P0 = ID0 xor ID1 xor ID2 xor ID4

P1 = ID1 xor ID3 xor ID4 xor ID5

Кадр данных включает в себя от двух до восьми полей данных, каждое из которых содержит восемь бит. Передача данных осуществляется старшим значащим битом (СЗБ) вперед. Поля данных записываются соответствующими подчиненными задачами. Так как нет никакого управления доступом к шине, то только одна подчиненная задача должна реагировать на идентификатор. Все остальные подчиненные задачи могут только принимать ответное сообщение и выполнять соответствующие действия.

Рис. 3.7. Поле данных LIN

Последним полем в кадре сообщения является поле контрольной суммы. Этот байт содержит инвертированную

сумму по модулю 256 всех байтов данных (кадр данных не включает идентификатор). Эта сумма рассчитывается путем суммирования с переносом всех байтов и инвертирования полученного результата. Свойства инвертированной суммы по модулю 256 таково, что если ее сложить с суммой всех байтов данных, то результат будет равен «0xFF».

Рис. 3.8. Поле контрольной суммы

LIN протокол позволяет находиться в режиме ожидания, чтобы быть совместимым со стандартами и экологическими ограничениями. Когда транспортное средство не используется, потребление всего транспортного средства должно быть меньше, чем несколько миллиампер, чтобы не разряжать аккумулятор. Таким образом, каждый блок управления должен войти в спящий режим или дежурный. Для перевода устройства в дежурный режим ему необходимо отправить кадр дежурного режима. Структура кадра дежурного режима идентична обыкновенному кадру сообщения, за исключением того, что идентификационный байт равняется "0x80". Содержимое полей данных не определено и может использоваться для передачи в системе различных параметров.

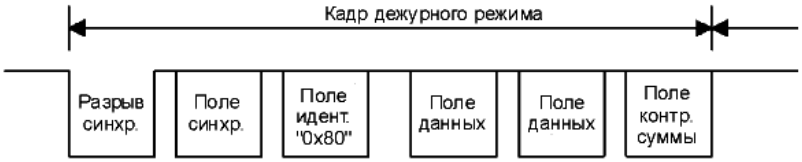


Рис. 3.9. Кадр дежурного режима

Режим сна (дежурный режим) может быть закончен

путем посылки любой подчиненной задачей сигнала активизации (пробуждения). Сигнал пробуждения возможно сформировать только в том случае, когда шина находится в режиме сна и внутренние узлы отслеживают его. Сигнал пробуждения - это кодовая комбинация "0x80". В зависимости от того, синхронизирована ли подчиненная задачи с ведущей задачей, скорость ее может быть отличной от скорости приемника ведущего устройства. По этой причине приемник ведущего устройства может идентифицировать переданный сигнал "0x80" как "0xC0", "0x80" или "0x00". Все эти сигналы будут восприняты как правильный сигнал пробуждения, и подчиненные устройства будут ожидать поля разрыва синхронизации от ведущего. Если в течение 128 Tbit не будет обнаружен этот сигнал, то сигнал пробуждения будет игнорирован. Так будет повторяться трижды. Перед посылкой нового запроса на пробуждения ведущее устройство будет ждать не менее 15 000 Tbit.

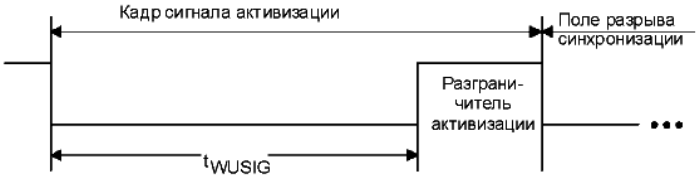


Рис. 3.10. Кадр сигнала активизации

LIN протокол обеспечивает обнаружение ошибок, как в идентификационном поле, так и в поле данных. Исправление ошибок не предусмотрено, поэтому единственный путь - автоматический повтор передачи поврежденных данных. Для выяснения правильности приема ведущий опрашивает подчиненных.

LIN протокол не определяет процедуру подтверждения для корректности принятия сообщения. Управляющий модуль ведущего устройства сравнивает переданные данные с полученными от своей собственной подчиненной задачи. Если данные идентичны, то принимается решение о правильности

передачи данных. Если же имеется расхождение в этих данных, то ведущий может повторить передачу сообщения.

Если несоответствие обнаруживает подчиненное устройство, то эта информация сохраняется и ведущему посылается запрос. Эта неверная информация может быть передана как часть кадра данных [8].

Программная реализация протокола LIN представлена в прил. 2.

UART

UART представляет собой логическую схему, с одной стороны подключенную к шине вычислительного устройства, а с другой имеющую два или более выводов для внешнего соединения.

UART может представлять собой отдельную микросхему или являться частью большой интегральной схемы. Используется для передачи данных через последовательный порт компьютера, часто встраивается в микроконтроллеры.

Передача данных в UART осуществляется по одному биту в равные промежутки времени. Этот временной промежуток определяется заданной скоростью UART и для конкретного соединения указывается в бодах (битах в секунду). Существует общепринятый ряд стандартных скоростей: 300 бод, 600 бод, 1200 бод, 2400 бод, 4800 бод, 9600 бод, 19200 бод, 38400 бод, 57600 бод, 115200 бод, 230400 бод, 460800 бод, 921600 бод.

Помимо собственно информационного потока UART автоматически вставляет в поток синхронизирующие метки, так называемые стартовый и стоповый биты. При приеме эти лишние биты удаляются из потока. Обычно стартовый и стоповый биты обрамляют один байт информации (8 бит), однако встречаются реализации UART которые позволяют передавать по 5,6,7, 8 или 9 бит. Обрамленные стартом и стопом биты являются минимальной посылкой. Некоторые реализации UART позволяют вставлять два стоповых бита при

передаче для уменьшения вероятности рассинхронизации приемника и передатчика при плотном трафике. Приемник игнорирует второй стоповый бит, воспринимая его как короткую паузу на линии.

Принято соглашение что пассивным (в отсутствие потока данных) состоянием входа и выхода UART является логическая 1. Стартовый бит всегда логический 0, поэтому приемник UART ждет перепада из 1 в 0 и отсчитывает от него временной промежуток в половину длительности бита (середина передачи стартового бита). Если в этот момент на входе все еще 0, то запускается процесс приема минимальной посылки. Для этого приемник отсчитывает 9 битовых длительностей подряд (для 8-бит данных) и в каждый момент фиксирует состояние входа. Первые 8 значений являются принятыми данными, последнее значение проверочное (стоп-бит). Значение стоп-бита всегда 1, если реальное принятое значение иное UART фиксирует ошибку.

Для формирования временных интервалов передающий и приемный UART имеют источник точного времени (тактирования). Точность этого источника должна быть такой, чтобы сумма погрешностей (приемника и передатчика) установки временного интервала от начала стартового импульса до середины стопового импульса не превышала половины (а лучше хотя бы четверти) битового интервала. Для 8-бит посылки 0,5/9,5 = 5% (в реальности не более 3%). Поскольку эта сумма ошибок приемника и передатчика плюс возможные искажения сигнала в линии, то рекомендуемый допуск на точность тактирования UART не более 1,5%.

Поскольку синхронизирующие биты занимают часть битового потока то результирующая пропускная способность UART не равна скорости соединения. Например, для 8-битных посылок синхронизирующие биты занимают 20% потока, что для физической скорости 115 200 бод дает битовую скорость данных 92160 бит/сек или 11 520 байт/сек.

Многие реализации UART имеют возможность автоматически контролировать целостность данных методом контроля битовой четности. Когда эта функция включена последний бит данных ("бит четности") контролируется логикой UART и содержит информацию о четности количества единичных бит в посылке.

В старые времена устройства с UART могли быть настолько медлительными что не успевали обрабатывать поток принимаемых данных. Для решения этой проблемы модули UART иногда снабжались отдельными выходами и входами управления потоком. При заполнении входного буфера логика принимающего UART выставляла на соответствующем выходе запрещающий уровень и передающий UART приостанавливал передачу.

Позже управление потоком возложили на коммуникационные протоколы и надобность в отдельных линиях управления потоком постепенно исчезла.

Логическая схема UART имеет входы-выходы с логическими уровнями, соответствующими полупроводниковой технологии схемы: КМОП, ТТЛ и т.д. Такой физический уровень может быть использован в пределах одного устройства, однако непригоден для коммутируемых длинных соединений по причине низкой защищенности от электрического разрушения и помехоустойчивости. Для таких случаев были разработаны специальные физические уровни, такие, как токовая петля, RS-232, RS-485, LIN и тому подобные.

Существуют физические уровни UART для сложных сред. В некотором смысле стандартный компьютерный телефонный модем также можно назвать специфическим физическим уровнем асинхронного интерфейса. Существуют специальные микросхемы проводных модемов, сделанных специально как физический уровень асинхронного интерфейса (т.е. протокольно прозрачные). Выпускается также

радиоканальный физический уровень в виде модулей радиоприемников и радиопередатчиков.

Идея асинхронной передачи данных появилась в те далекие времена, когда о стандартизации еще мало заботились и лучшее что можно было ожидать от поставщиков разрозненных решений так это открытой публикации алгоритмов работы своих изделий. Собственно поэтому стандарта UART как такового нет, но логика работы UART описана как часть продукта во многих других стандартах: токовая петля, RS-232, ISO/IEC 7816 и т.п.

UART реализован в устройствах фирмы Texas Instruments MSP430x12xx, MSP430x13xx, MSP430x15x, MSP430x14x и MSP430x16x.

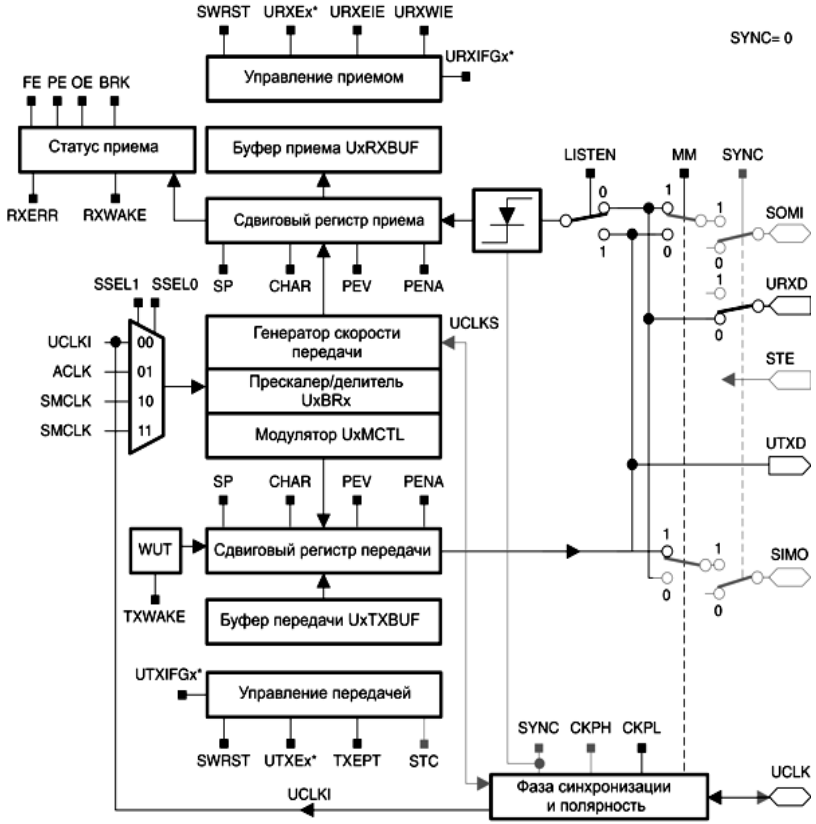


Рис. 3.11. Блок-схема UART MSP430

Программная реализация протокола UART представлена в прил. 3.

ГЛАВА 4. ПРОЕКТИРОВАНИЕ ПРОГРАММНО-АППАРАТНЫХ СРЕДСТВ ЗАЩИТЫ КОМПЬЮТЕРНОЙ ИНФОРМАЦИИ

4.1. Методы защиты компьютерной информации

Существуют различные методы защиты информации:

– скрыть канал передачи информации, используя нестандартный способ передачи сообщений;

– замаскировать канал передачи закрытой информации в открытом канале связи, например, спрятав информацию в безобидном «контейнере» с использованием тех или других стенографических способов либо обмениваясь открытыми сообщениями, смысл которых согласован заранее;

– существенно затруднить возможность перехвата передаваемых сообщений, используя специальные методы передачи по широкополосным каналам, сигнала под уровнем шумов, либо с использованием «прыгающих» несущих частот и т.п.

В отличие от перечисленных методов криптография не «прячет» передаваемые сообщения, а преобразует их в форму, недоступную для понимания противником.

Также существуют и другие проблемы защиты передаваемой информации. Например, при полностью открытом обмене возникает проблема достоверности полученной информации. Для ее решения необходимо обеспечить:

– проверку и подтверждение подлинности содержания источника сообщения;

– предотвращение и обнаружение обмана и других умышленных нарушений со стороны самих участников информационного обмена [9].

4.1.1. Аппаратные средства защиты информации

Аппаратные средства защиты компьютерной информации - это различные электронные,

электромеханические и другие устройства, непосредственно встроенные в блоки автоматизированной информационной системы или оформленные в виде самостоятельных устройств и сопрягающиеся с этими блоками.

Основные функции аппаратных средств защиты:

–запрещение несанкционированного (неавторизованного) внешнего доступа (удаленного пользователя, злоумышленника) к работающей автоматизированной информационной системе;

– запрещение несанкционированного внутреннего доступа к отдельным файлам или базам данных информационной системы, возможного в результате случайных или умышленных действий обслуживающего персонала;

– защита активных и пассивных (архивных) файлов и баз данных, связанная с не обслуживанием или отключением автоматизированной информационной системы;

– защита целостности программного обеспечения.

Эти задачи реализуются аппаратными средствами защиты информации с использованием метода управления доступом (идентификация, аутентификация и проверка полномочий субъектов системы, регистрация и реагирование) [10].

4.1.2. Программные средства защиты информации

Кроме программ шифрования и криптографических систем, существует много других доступных внешних средств защиты информации.

Межсетевые экраны (также называемые брандмауэрами или файрволами — от нем. Brandmauer, англ. firewall — «противопожарная стена»). Между локальной и глобальной сетями создаются специальные промежуточные серверы, которые инспектируют и фильтруют весь проходящий через них трафик сетевого/транспортного уровней. Более защищенная разновидность метода — это способ маскарада (masquerading), когда весь исходящий из локальной сети

трафик посылается от имени firewall-сервера, делая локальную сеть практически невидимой.

Proxy-servers (proxy — доверенность, доверенное лицо). Весь трафик сетевого/транспортного уровней между локальной и глобальной сетями запрещается полностью — маршрутизация как таковая отсутствует, а обращения из локальной сети в глобальную происходят через специальные серверы-посредники. Очевидно, что при этом обращения из глобальной сети в локальную становятся невозможными в принципе. Этот метод не дает достаточной защиты против атак на более высоких уровнях — например, на уровне приложения (вирусы, код Java и JavaScript).

VPN (виртуальная частная сеть) позволяет передавать секретную информацию через сети, в которых возможно прослушивание трафика посторонними людьми. Используемые технологии: PPTP, PPPoE, IPSec [10].

4.1.3. Смешанные аппаратно-программные средства защиты информации

Организационные средства складываются из организационно-технических (подготовка помещений с компьютерами, прокладка кабельной системы с учетом требований ограничения доступа к ней и др.) и организационно-правовых (национальные законодательства и правила работы, устанавливаемые руководством конкретного предприятия). Преимущества организационных средств состоят в том, что они позволяют решать множество разнородных проблем, просты в реализации, быстро реагируют на нежелательные действия в сети, имеют неограниченные возможности модификации и развития. Недостатки — высокая зависимость от субъективных факторов, в том числе от общей организации работы в конкретном подразделении.

По степени распространения и доступности выделяются программные средства, другие средства применяются в тех

случаях, когда требуется обеспечить дополнительный уровень защиты информации[10].

4.2. Обзор существующих методов шифрования

Криптоалгоритм – это последовательность математических или алгоритмических преобразований, производимых над блоками исходных данных для получения зашифрованного блока данных, недоступного для прочтения сторонними лицами.

В зависимости от наличия либо отсутствия ключа, кодирующие алгоритмы делятся на тайнопись и криптографию. От соответствия ключей шифрования и дешифрования алгоритмы делятся на симметричные и асимметричные. В зависимости от типа используемых преобразований – на подстановочные и перестановочные. В зависимости от размера шифруемого блока – на потоковые и блочные шифры. В отношении криптоалгоритмов существует несколько схем классификации, каждая из которых основана на группе характерных признаков. Таким образом, один и тот же алгоритм "проходит" сразу по нескольким схемам, оказываясь в каждой из них в какой-либо из подгрупп.

Каждый пользователь компьютера, сам того не подозревая, использует механизмы криптозащиты. Ядро этих механизмов - алгоритмы шифрования (табл. 4.1). Обычно пользователю нет необходимости изучать все тонкости криптографической науки, но, безусловно, полезно знать, как "устроены" эти алгоритмы и каковы их сравнительные характеристики [11].

Таблица 4.1

Алгоритмы, используемые в криптографии

Симметричные		С открытым ключом	Хэш-функции
с блочным шифрованием	с поточным шифрованием		
DES, New DES, AES, Blowfish, RC2, CAST, ГОСТ 28147-89	RC4, ARC4, DESS, IBAA, JEROBOAM, ISAAC, Rabbit	RSA, ECDH	MD2, MD5, SHA-1

4.2.1. Симметричные алгоритмы шифрования

Если при шифровании информации с помощью специального алгоритма и ключа ее расшифровка происходит с помощью того же самого алгоритма и ключа, такой метод шифрования называется симметричным (рис. 4.1). Отсюда и два других названия данного метода - с помощью симметричных алгоритмов, или алгоритмов с симметричным ключом. Встречается и еще одно название данного метода - криптография с секретным ключом**.**

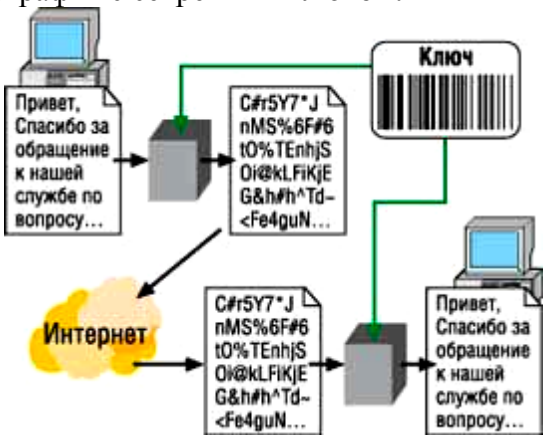


Рис. 4.1. Симметричный ключ используется и для шифрации, и для дешифрации

В симметричных алгоритмах шифрования применяются два вида шифрования - блочный и поточный.

При шифровании по блокам кодирование данных выполняется после разбиения их на блоки, каждый из которых шифруется отдельно с помощью одного и того же ключа. Если размер исходной информации не кратен размеру блока, то последний блок дополняется символами-заполнителями. Особенность данного способа шифрования - в том, что одни и те же данные шифруются одним ключом, т. е. одинаково. Как правило, такие алгоритмы используют при шифровании файлов, баз данных и электронной почты (табл. 4.2).

Таблица 4.2

Характеристики алгоритмов с блочным шифрованием

Название	Длина ключа, бит	Размер обрабатывае-мых блоков, бит	Число раундов
AES (Rijndael)	128, 192 или 256	128, 192 или 256	10, 12, 14
Blowfish	32-448	64	16
CAST-128	128	64	16
CAST-256	256	128	16
DES	56	64	16
IDEA	128	64	8
NewDES	120	64	17
RC2	до 1024	64	16
RC5	до 2048	32, 64 или 128	0...255
ГОСТ 28147-89	256	64	16 или 32

При поточном шифровании каждый байт обрабатывается индивидуально, однако, хотя ключ в данном случае одинаков для всех байтов, для шифрования используются псевдослучайные числа, сгенерированные на основании ключа. Характерная особенность этого способа шифрования - высокая производительность, благодаря чему он применяется при шифровании потоков информации в каналах связи (так, один из элементов протокола SSL выполняет поточное шифрование по алгоритму RC4).

В настоящее время наиболее распространены алгоритмы DES, Triple DES, AES, а в России - ГОСТ 28147-89. Достаточно широко применяются также алгоритмы Blowfish, Safer, RC2 и RC5 и CAST.

DES (Digital Encryption Standard) - это блочный шифр, использующий 56-разрядный ключ. Алгоритм был разработан в конце 70-х гг. прошлого века исследователями из IBM и National Security Agency (NSA). В 80-х эксперты полагали, что алгоритм не имеет слабых мест, но с появлением быстродействующих компьютеров в 90-х его репутация несколько пострадала - стала возможной атака методом перебора ключа (ключ DES был взломан специалистами компании Electronic Frontier Foundation в 1999 г. менее чем за 24ч).

Triple DES - усовершенствованный блочный алгоритм DES. Принцип его работы не отличается от применяемого в DES, а усиление достигается благодаря трехкратному (triple) шифрованию одного блока алгоритмом DES. Три 56-разрядных ключа, используемых в данном процессе, объединяются алгоритмом в один 168-разрядный ключ. И хотя время атаки перебором при обычной мощности компьютера составляет несколько миллиардов лет, что говорит о хорошей стойкости алгоритма, в некоторых публикациях описаны способы сокращения времени атаки - до уровня перебора 108-разрядного ключа. Сегодня существует также вариант Triple DES с "двойным" DES-ключом размером 112 бит, и он

применяется чаще.

AES (Advanced Encryption Standard) - еще один блочный алгоритм, который был разработан бельгийскими исследователями Винсентом Риджменом (Vincent Rijmen) и Джоан Димен (Joan Daemen) и принят в качестве стандарта Национальным институтом стандартов и технологий (NIST) 2 октября 2000 г. Конкурс на новый стандарт был объявлен тремя годами ранее, причем среди его условий значился обязательный отказ разработчиков от права интеллектуальной собственности, что позволяло сделать новый стандарт открытым и применять его без отчислений авторам.

ГОСТ 28147-89 в США часто называют русским аналогом DES. Но по сравнению с DES ГОСТ 28147-89 значительно более криптостоек и сложен. Он был разработан в одном из институтов КГБ в конце 1970-х гг., статус официального стандарта шифрования СССР получил в 1989 г., после распада СССР принят в качестве стандарта Российской Федерации. ГОСТ 28147-89 оптимизирован для применения в программных реализациях, использует вдвое больше DES-раундов шифрования с гораздо более простыми операциями, а длина ключа у него в пять раз больше [11].

4.2.2. Асимметричные алгоритмы шифрования

Если в симметричном алгоритме для шифрования и расшифрования применяется один и тот же ключ, то в асимметричном - фактически два (рис. 4.2), один для шифровки, другой - для расшифровки данных. Ключи составляют неразрывную пару (ключи из разных пар никогда не смогут работать совместно), и потому для такой пары часто употребляется термин "асимметричный ключ", а каждый ключ из пары называют половиной асимметричного ключа.

Ключ, который зашифровывает данные, называется открытым, а ключ, который их расшифровывает, - секретным. Таким образом, чтобы получить важную информацию,

достаточно сгенерировать два ключа. Один - секретный - спрятать в надежном месте, второй - открытый - можно свободно распространять и даже выкладывать в общедоступных местах, например, на сайте. Если кто-то захочет послать секретную информацию, то предварительно он просто зашифрует данные открытым ключом. Расшифровать ее удастся только секретным ключом.

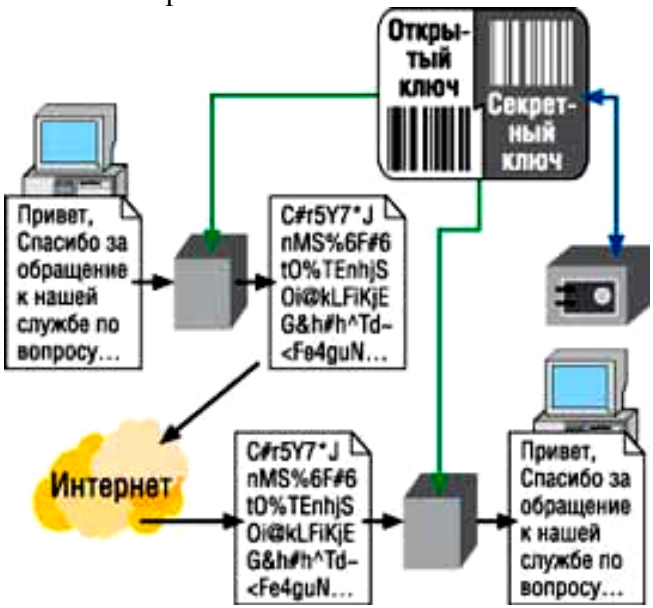


Рис. 4.2. Две части асимметричного ключа - одна для кодирования, другая для расшифровки

Недостаток работы с открытыми ключами - обработка информации требует много времени, в несколько десятков раз больше, чем при работе с симметричными алгоритмами. Поэтому при передаче больших объемов данных их шифруют с симметричным ключом, а затем сам симметричный ключ шифруют с помощью открытого ключа (рис. 4.3). Таким образом, поскольку шифрование открытым ключом применяется только к симметричному ключу, время обработки немногим отличается от симметричного шифрования. К тому

же при такой схеме симметричный ключ работает только в одном сеансе (а потому получил название сеансового), а его обработка выполняется автоматически программными средствами, следовательно, нет необходимости даже извещать пользователя о существовании симметричного ключа.

Еще один серьезный недостаток алгоритма симметричного шифрования - проблема подмены открытых ключей.

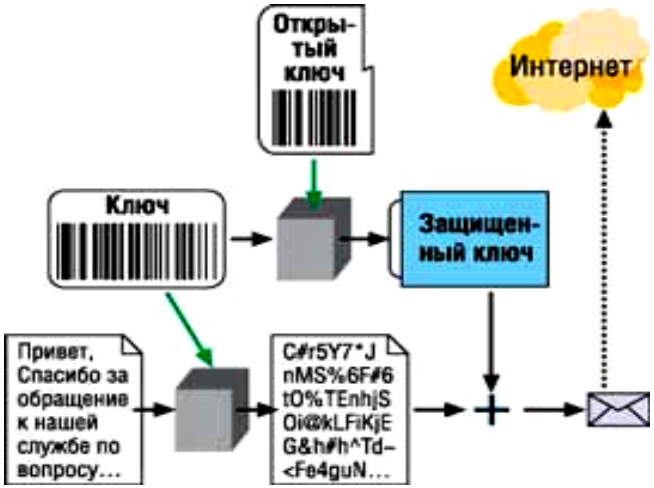


Рис. 4.3. После зашифровки данных симметричный ключ шифруется открытым ключом и объединяется с зашифрованными данными

DH (Diffie-Hellman) был создан Уитфилдом Диффи и Мартином Хеллманом в 1976 г. По сути Диффи и Хеллман предложили схему, по которой путем обмена открытой информацией можно создать совместно используемый секретный ключ. Фактически алгоритм DH - это не алгоритм шифрования, а схема распределения ключей и алгоритм создания симметричного сеансового ключа. Согласно DH, каждая из взаимодействующих сторон обладает секретным и открытым значениями ключа. При объединении секретного значения с другим открытым каждый пользователь сможет

создать один и тот же секретный ключ.

Эллиптический алгоритм ECDH (Elliptic Curve Diffie-Hellman) появился в 1985 г. благодаря трудам Нила Коблица (Neal Koblitz) из Вашингтонского университета и Виктора Миллера (Victor Miller) из исследовательского центра Томаса Уотсона корпорации IBM. В своих работах независимо друг от друга они пришли к выводу, что эллиптические функции, используемые в математике, можно с успехом применять в криптографии с открытым ключом. В настоящее время, помимо ECDH, существует и активно реализуется еще немало алгоритмов, базирующихся на эллиптических функциях [11].

4.2.3. Криптопроцессоры шифрования

Целочисленный мультипликативный криптопроцессор можно реализовывать на базе операций умножения (деления) и оперирует с исходными целочисленными данными.

Подстановочный символьный криптопроцессор шифрует текстовую информацию по методу подстановки (замены) символов открытого текста. В результате шифрования исходное слова заменено другим. При процессе шифрования один символ замещается на другой. Криптопроцессор представляет собой реализацию конечного автомата, каждому состоянию которого соответствует отдельная процедура замены символа криптопроцессора.

Перестановочный битовый криптопроцессор реализовывается на процедуре шифрования и дешифрования текстовой информации, используя метод перестановки битов. В каждом коде символа, представленного в виде логического вектора, реализуется перестановка двух пар битов: первая пара содержит 7-й и 6-й биты, а вторая -3-й и 2-й.

Дешифрование зашифрованного логического вектора осуществляется путем перестановки битов.

Криптопроцессор с многоуровневой системой шифрования, формульный метод. В этом криптографическом процессе информация шифруется на двух уровнях. На первом

уровне применена комплексная двухступенчатая операция (нелинейное формульное выражение) для получения шифра ключей (kluch1, kluch2), к которым применена операция деления.

На втором уровне для получения окончательного шифра data_shifr_out над промежуточным шифром десятичных ASCII-кодов исходных текстовых данных выполняется дополнительное преобразование с использованием глобального ключа. Это дополнительное преобразование реализуется посредством операции сложения.

Символьный аддитивный криптопроцессор heализовывается на базе операций сложения (вычитания). Поскольку прямые арифметические операции над данными такого типа запрещены, решить проблему корректной работы криптопроцессора можно с помощью функций преобразования типа. С целью обеспечения переносимости криптопроцессора между разными САПР в программном коде не используются библиотечные функции преобразования [12].

4.3. Разработка криптоситемы с использованием аппаратных методов шифрования

4.3.1. Работа криптосистемы и её взаимодействие с ПК

Криптосистема с использованием аппаратных методов шифрования реализована на инструментальном модуле Xilinx Spartan3 Starter Kit. Обмен данными осуществляется через COM-порт компьютера имеющего разъём DB-9. На модуле Spartan3 Starter Kit установлен преобразователь уровней RS-232, обеспечивающий возможность непосредственного подключения к последовательному порту через разъем DB-9 установленного на плате, при реализации универсального асинхронного приемопередатчика UART на основе ПЛИС. Ниже на рис. 4.4, представленная связь между ПК и ПЛИС, обмен данными между которыми осуществляется через интерфейс RS232.

На плате присутствуют два вида индикации,

обеспечивающих возможность визуального контроля функционирования разрабатываемой криптоситемы. На семисегментных индикаторах отображается информация о режиме работы криптоситемы, что соответствует H-шифрование и d-дешифрование. На светодиодах отображается выбранный метод шифрования/дешифрования.

Последовательность светодиодов считать слева на права, и что соответствует:

1. Подстановочный символьный криптопроцессор.
2. Перестановочный битовый криптопроцессор.
3. Криптопроцессор на основе гаммирования.
4. Подстановка и перестановка.
5. Подстановка и гамма.
6. Перестановка и гамма.

Для методов с использованием гаммы используются 8 ползунковые переключатели, которые используются для ручной установки гаммы, которая впоследствии применяется как ключ для шифрования и дешифрования текстовой информации.

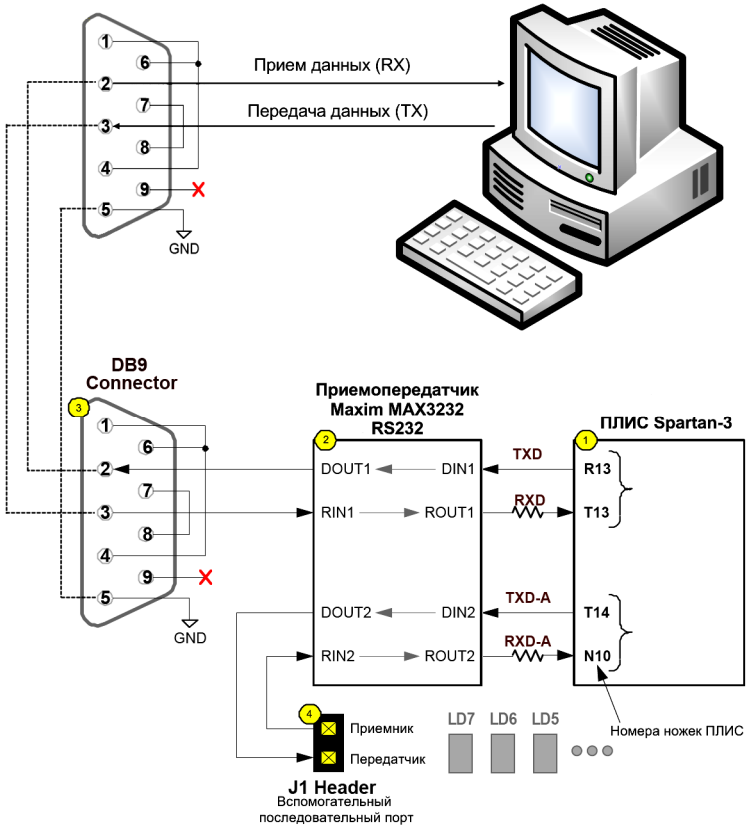


Рис. 4.4. Взаимодействие ПК и ПЛИС Spartan-3 посредством стандарта RS232

На рис. 4.4 отмечено:

1. 1.2В серия семейства ПЛИС с архитектурой FPGA (Field-Programmable Gate Arrays) Spartan-3 специально разработана для использования в электронных устройствах, рассчитанных на большие тиражи и невысокую стоимость комплектующих. Технологический процесс 90 нм. Аппаратных ядер нет, есть поддержка синтезируемых ядер микропроцессоров MicroBlaze (32-бит) и PicoBlaze (8-бит).

2. Приемопередатчик Maxim MAX3232 RS232 связывает

между собой ПЛИС и разъем DB9, который соединён с COM – портом компьютера. Также данная микросхема включает в себя преобразователь напряжения RS-232.

3. Последовательный порт RS-232, имеющий разъём DB9 female (мамка) к которому подключается кабель “Прямой 1:1 9f/9m”.

4. Дополнительный последовательный канал RS-232 созданный на двух выходах. Это 1- приёмник и 2-передатчик.

Структурная схема рассматриваемого модуля изображена на рис. 4.5. Основными элементами структуры модуля Spartan-3 Starter Board являются:

– ПЛИС XC3S200 в корпусе FT256, на основе которой реализуется проектируемая система;

– программируемое в системе ППЗУ серии Platform Flash XCF02S, предназначенное для хранения конфигурационных данных ПЛИС;

– схема загрузки конфигурационных данных;

– схема управления конфигурированием ПЛИС;

– блок синхронизации, предназначенный для формирования внешних тактовых сигналов;

– схема формирования питающих напряжений;

– внешнее высокоскоростное ОЗУ;

– блок светодиодных индикаторов;

– четырехзначный дисплей, выполненный на основе 7-сегментных светодиодных индикаторов;

– блок ползунковых переключателей;

– блок кнопочных переключателей;

– схема преобразования уровней сигналов интерфейса RS-232;

– стандартные разъемы интерфейсов RS-232, PS/2 и VGA;

– три 40-контактных разъема расширения.

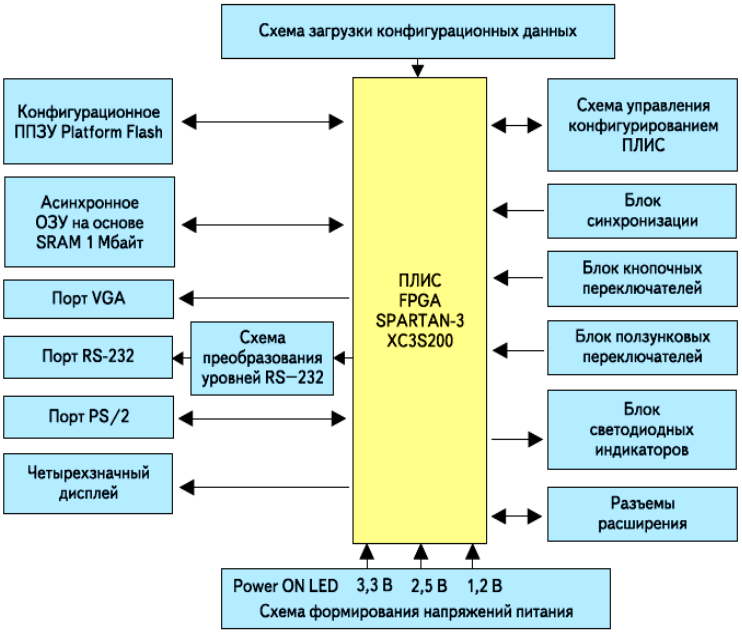


Рис. 4.5. Структурная схема инструментального модуля Spartan3 Starter Kit

4.3.2. Разработка криптопроцессоров

Криптографическая система защиты информации основана на различных алгоритмах, подстановочного, перестановочного, гаммирования и комбинированных. Криптографические методы реализовываются на языке проектирования VHDL позволяющую шифровать и дешифровать текстовую информацию на базе плис Spartan3, которая должна обеспечивать решение триединой задачи:

- конфиденциальность информации (доступность ее только для того, кому она предназначена);
- целостность информации (ее достоверность и точность, а также защищенность от преднамеренных непреднамеренных искажений);

- готовность информации (использование в любой момент, когда в ней возникает необходимость).

Криптографическая защита в большинстве случаев является более эффективной и дешевой. Конфиденциальность информации при этом обеспечивается шифрование передаваемых документов. На основе обзора криптопроцессоров и видов шифрования мы выбрали наиболее подходящие для реализации алгоритмы [11].

4.3.3. Подстановочный символьный криптопроцессор

Криптопроцессор шифрует текстовую информацию по методу подстановки (замены) символов открытого текста. В результате шифрования исходное слово заменено другим. При процессе шифрования один символ открытого текста заменяет на некоторый другой. В данном криптопроцессоре используется одноалфавитный шифр подстановки (шифр простой замены) – шифр, при котором каждый символ открытого текста заменяется на некоторый, фиксированный символ другого алфавита. Криптопроцессор представляет собой реализацию конечного автомата, каждому состоянию которого соответствует отдельная процедура замены символа криптопроцессора.

На рис. 4.6 проиллюстрирован сам принцип работы подстановочного метода, используемый в данном криптопроцессоре.

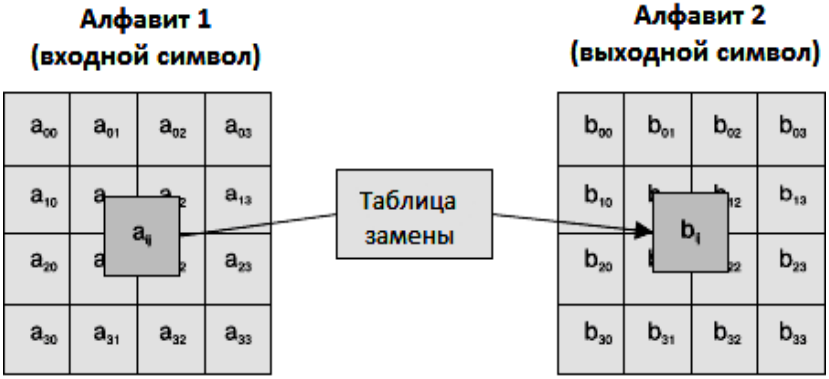


Рис. 4.6. Табличная замена каждого байта массива

Данный метод реализуется в отдельном пакете двумя различными процедурами для шифрования и дешифрования текстовой информации. В зависимости от выбранного режима, из головного модуля происходит обращение к той или иной процедуре.

Для реализации подстановочного криптопроцессора в обеих процедурах используется оператор выбора CASE, который всегда выполняет только один блок последовательных операторов из нескольких имеющихся в теле оператора блоков на основе значения задающего выражения. Выбирается тот блок, у которого сопровождающее значение тождественно равно текущему значению задающего выражения.

Значение задающего выражения представляет собой одномерный массив битов, входного слова. При выполнении оператор case сравнивает текущее вычисленное значение выражения с каждым указанным в теле оператора значением. Когда сравниваемые значения тождественно равны, происходит выбор соответствующего блока последовательных операторов. Каждый блок имеет только своё единственное значение. Ниже приведена часть листинга процедуры, выполняющей замену символов:

```
Procedure shzm(signal i: in std_logic_vector(7 downto 0);
               signal o: out std_logic_vector(7 downto 0)) is
begin
        case i is
------------Входящий символ-------Выходящий символ
when    "00100000" => o <="01001100";  --        SP      L
when    "00100001" => o <="01001101";  --        !       M
when    "00100010" => o <="01001110";  --        "       N
when    "11111101"      => o <="10101000";  --   э Ё
when    "11111110"      => o <="11000110";  --   ю Ж
when    "11111111"      => o <="11000111";  --   я       З       when
others => o <= i;
        end case;
end shzm;
```

В алфавите используются все русские и английские символы, а также цифры и различные знаки, используемые на стандартных клавиатурах ПК.

4.3.4. Перестановочный битовый криптопроцессор

Криптопроцессор основывается на шифровании и дешифрования текстовой информации, используя метод перестановки битов каждого слова. В каждом коде символа, представленного в виде логического вектора, реализуется перестановка двух пар битов: первая пара содержит 6-й и 5-й биты, а вторая - 3-й и 2-й. Дешифрование зашифрованного логического вектора осуществляется путем перестановки битов, аналогично шифрования. На рис. 4.7 представлена схема перестановки битов.

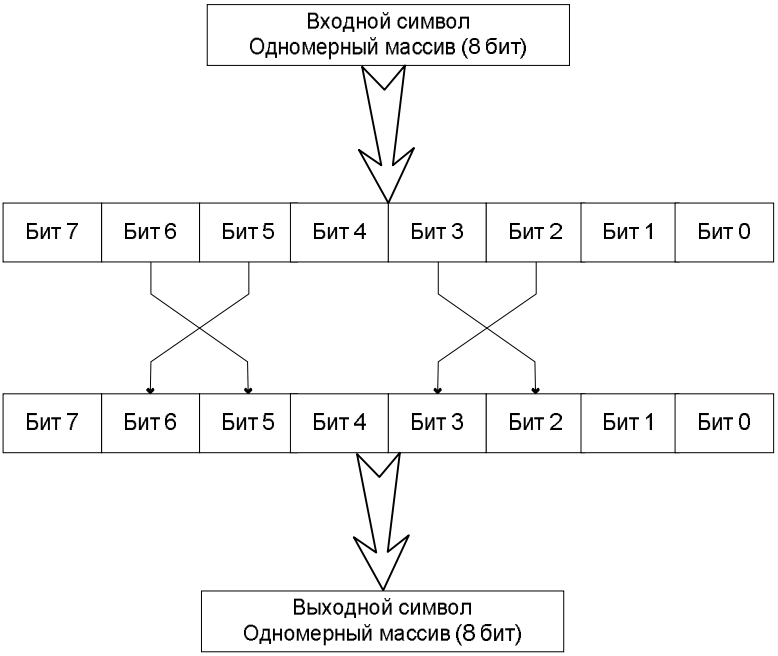


Рис. 4.7. Принцип перестановки двух пар битов

Метод перестановки двух пар битов реализован в отдельном пакете (perest.vhd) в виде одной функции, которая выполняет работу как шифратора, так и дешифратора для перестановочного битового криптопроцессора.

Листинг функции представлен ниже:

```
function per(signal i: in std_logic_vector(7 downto 0)) return std_logic_vector is
variable o,sh:std_logic_vector(7 downto 0);
begin
sh:= i(7)&i(5)&i(6)&i(4)&i(2)&i(3)&i(1 downto 0);
if sh="10100111"or sh="10101100"or sh="10101010"or sh="10000110"or
sh="10110001"or sh="10011010" then                    o:=i;
        else
                o:=sh;
        end if;
return o;
end function per;
```

В данной функции входящий символ в виде одномерного массива в 8 бит, записывается по битно в выходной одномерный массив, согласно последовательности реализованной в алгоритме. После чего новый полученный одномерный массив проверяется на условие, что его 8 бит данных не совпадают с битами уже зарезервированных 8 битными словами, используемые для управления внешним аппаратным шифратором, через ПК. Если выходной массив оказывается тождественно равным, хотя бы с одним из зарезервированных слов, тогда он возвращает выходному значению, прежнее значение, поступающее на вход для обработки [13].

4.3.5. Криптопроцессор на основе гаммирования

Метод шифрования, основанный на «наложении» гамма-последовательности на открытый текст. При котором суммируются 8 бит данных открытого текста с заранее определенной гаммой, непосредственно введенной с внешнего устройства шифрования. Дешифрирование закрытого текста происходит по аналогии с шифрованием, с обратным знаком «наложения» гаммы.

Шифрование и дешифрование представляет собой две функции реализованных в отдельном пакете. На рис. 4.8 и рис. 4.10 представлены алгоритмы работы данного метода для шифрования и дешифрования текстовой информации соответственно.

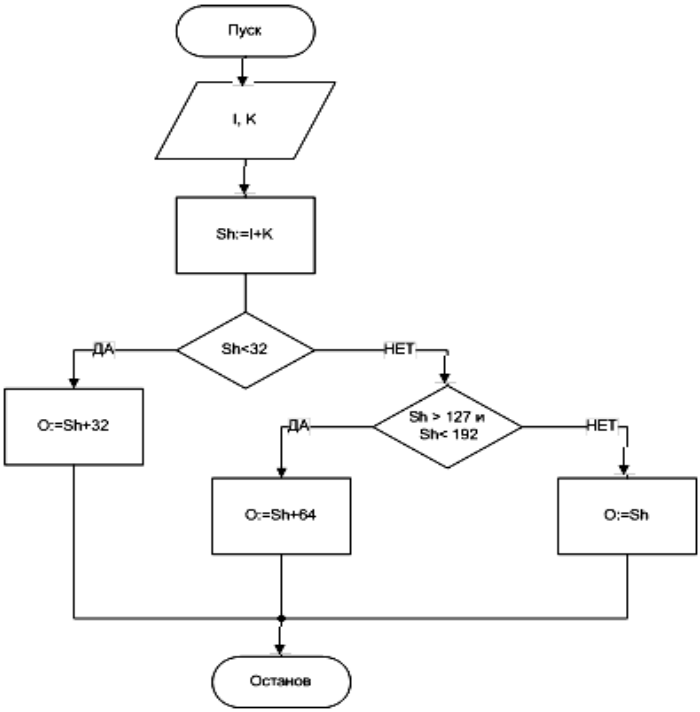


Рис. 4.8. Алгоритм шифрования с использованием гаммы

На рис. 4.8 I - входной символ открытого текста, K - значение гаммы, вводимое с устройства, O - выходной символ зашифрованного слова, Sh – промежуточный результат шифрования, 32, 192, 127, 64 - константы.

С помощью констант мы устанавливаем определённые рамки, за которые текст не должен уходить, и если такое случается то происходит смещение бит на определённое значения, согласно условию в алгоритме, что предотвращает сбои и появление неожиданных результатов при дешифрации. Листинг шифрования для данного метода приведен ниже:

```
function shkl(signal i,k: in std_logic_vector(7 downto 0)) return
std_logic_vector is
                variable o: std_logic_vector(7 downto 0);
                variable sh:std_logic_vector(7 downto 0);
        begin
Sh:=(i+k);
        if (sh < s32) then o:=sh+twenty;
        elsif (sh > s127)and(sh < s192) then        o:=sh+fourty;
        else                        o:=sh;
        end if;
                return o;
end function shkl;
```

С помощью проверок, мы оставляем символы из промежутка от 32 до 127 и от 192 до 255 включительно, данные символы можно увидеть из рис. 4.9, на котором представлена таблица символов ASCII CP-1251. В ОС Windows, для кириллицы используется именно данный стандарт CP-1251, по которому символ преобразуется в двоичный 8 битный код и передается посредством порта RS-232 на ПЛИС Spartan-3 для дальнейшей обработки информации [13].

При дешифрировании закрытого текста, результат проверяется в два этапа. На первом этапе определяется, к какому диапазону принадлежит символ. Если от 32 до 64, то возникает вероятность, что при шифровании открытого текста было смещения вправо на 32 бита. Предполагая данный факт, смещаем полученный результат на 32 бит влево, чтобы получить требуемый результат. Если же символ находится в диапазоне от 192 до 255 включительно, то смещаем его на 64 бита влево, так как в процессе шифрования, возможно, было смещение вправо на 64 бита. В остальных случаях просто вычитается гамма.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				©	Ё	§	Є	·		°						
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1																
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2		!	"	#	$	%	&		(	)	*	+				/
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}]	~	□
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	Ъ	Г	‚	ѓ	„	…	†	‡		‰	Љ	‹	Њ	Ќ	Ћ	Џ
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	ђ	‘	’	"	"		–	—		™	љ	›	њ	ќ	ћ	џ
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A		Ў	ў	J	¤	Ґ	¦	§	Ё	©	Є	«		-	®	Ї
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	°	±	І	і	ґ	µ		·	ё	№	є	»	j	S	s	ї
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	▪ 223
E	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Рис. 4.9. Кодовая таблица символов ASCII CP-1251

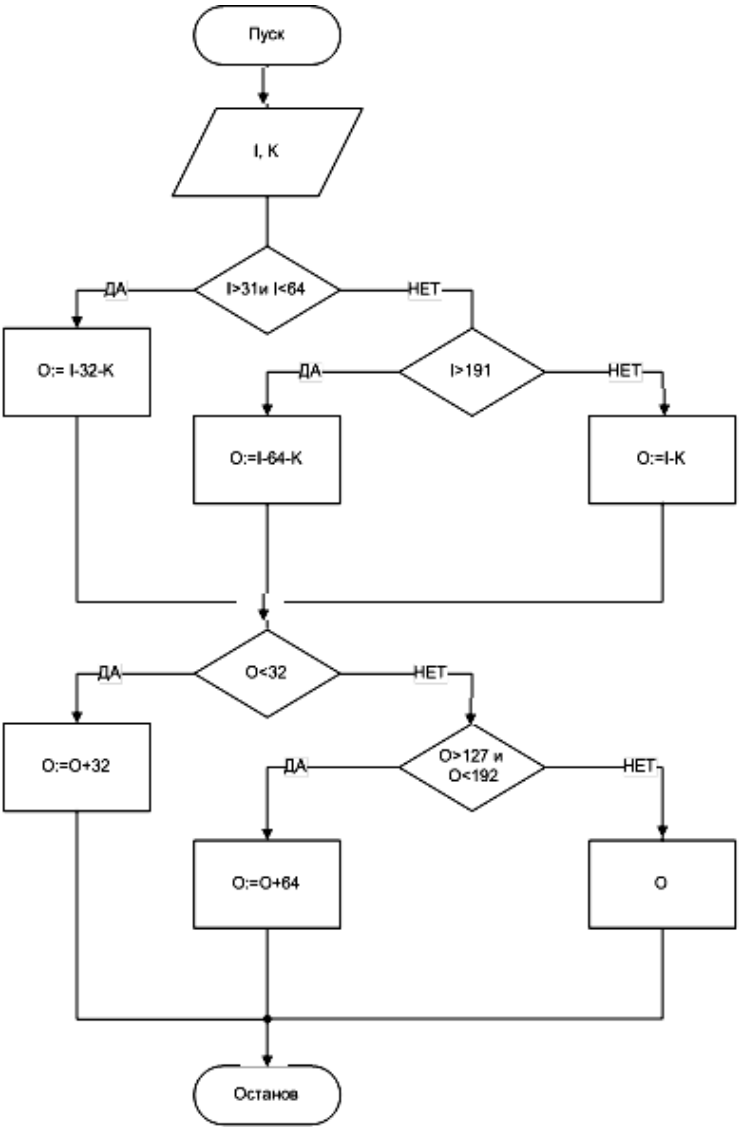


Рис. 4.10. Алгоритм дешифрования с использованием гаммы

На сто процентов утверждать, что результат соответствует открытому тексту нельзя, поэтому на втором

этапе проверяется полученный результат, аналогично проверке, которая проводилась при шифровании информации.

Листинг дешифрации:

```
function dhkl(signal i,k: in std_logic_vector(7 downto 0)) return
std_logic_vector is
        variable o: std_logic_vector(7 downto 0);
    begin
        if i>s31 and i<s64 then o:=i-twenty-k;
        elsif i>s191 then o:=i-fourty-k;
        else o:=i-k;
        end if;
        if o<s32 then    o:=o+twenty;
        elsif (o>s127 and o<s192) then
            o:=o+fourty;
        end if;
return o;
end function dhkl;\
```

Листинг реализации, функций и процедур всех криптопроцессоров приводится в прил. 4.

4.3.6. Комбинированные криптопроцессоры

Шифрование комбинированными методами основывается на результатах, полученных К. Шенноном. Наиболее часто применяются такие комбинации, как подстановка и гамма, перестановка и гамма, подстановка и перестановка.

Особенность данного алгоритма состоит в том, что при большом объеме шифртекста частотные характеристики символов шифртекста близки к равномерному распределению независимо от содержания открытого текста. Шифрование комбинированными методами является наиболее стойкой к криптоанализу, что увеличивает защиту информации в несколько раз.

Благодаря тому, что все криптопроцессоры выполнены в отдельных процедурах и функциях, и объединены в один пакет, подключенный к главному модулю как библиотека, дает нам

возможность реализовывать данные методы с минимальными затратами времени и труда. Реализация данных методов выполняется с помощью последовательных обращений к той или иной процедуре/функции.

К примеру комбинация подстановка и перестановка осуществляется всего лишь двумя обращениями:

```
shzm(dout,zm);
conv_buf<=per(zm);
```

Входной символ открытого текста, записанный в сигнал dout вызывает процедуру подстановки shzm, результат выполнения процедуры записывается в переменную zm. Далее результаты переменной zm передаем для обработки функции per, в которой происходит перестановка двух пар битов. После чего результат передается пользователю по COM-порту [12].

4.3.7. Разработка модуля управления криптопроцессорами

Данный модуль предназначен для выбора определенного криптопроцессора, и его режима работы шифрование/дешифрование. Принцип работы представлен на рис. 4.11.

Рис. 4.11. Принцип управления криптороцессорами

Принцип работы управления заключается в том, что когда поступает на вход сигнал, состоящий из 8 бит, он начинает обрабатываться параллельно в нескольких процессах.

Выбор метода и режима, выполнена в виде так называемой «защелки», которая устанавливает свое значение в определенное состояние, если входной сигнал оказался тождественно равным любому из перечисленных значений описанных в условии. В листинге видно, что при определенном значении, «защелке» Var и MD присваивается определенное значение, которое на протяжении всего времени будет оставаться в одном и том же положении, пока не поступит определенный сигнал, который изменит значение.

```
----Выбор метода
process(clk,dout)begin
        if(clk'event and clk='1')then
                if dout="10100111"then-- § подстановочный
                        var<=1; lg1<='1'; lg2<='0'; lg3<='0';
                        lg4<='0'; lg5<='0';          lg6<='0';
                elsif dout="10101100"then-- ¬ перестановочный
                        var<=2; lg1<='0'; lg2<='1';
                        lg3<='0'; lg4<='0'; lg5<='0'; lg6<='0';
                                . . . . .
                        end if;

                        if dout="10000110"then-- † шифрование
MD<='0';an0<="0111"; sn<="0001001";
                        elsif
dout="10110001"then--± дешифрование
MD<='1'; an0<="0111"; sn<="0100001";
                        end if;
                end if;
        end process;
```

Сигналы lg1-lg6 выполняют роль зажигания определенных светодиодов на плате Spartan-3 Starter Kit, для контроля процесса работы внешнего шифратора. Сигнал sn, отображает на семисегментном индикаторе состояние режима, что соответствует “H”- шифрование, “d” – дешифрование.

Имея фиксацию «защелки», реализуем процесс шифрования и дешифрования. В соответствии с определенными значениями «защелки» Var и MD, в результате

работы процесса происходит вызов той или иной процедуры или функции, где реализован криптопроцессор, находящийся в отдельном пакете. Часть процесса реализации обращения к криптопроцессорам приведен ниже:

```
process(var,MD)
begin
if (var=2) then
                    if dout="10101100"then
                    conv_buf<=dout;
          else
--Перестановочный битовый криптопроцессор
--Перестановка 2х пар битов: 7-й и 6-й биты, 3-й и 2-й биты.
                    if (MD='0') then
                              if dout="10000110"then
                               conv_buf<=dout;
                              else
                               conv_buf(7downto 0)<=per(dout);
                              end if;
                    elsif (MD='1') then
                              if dout="10110001"then
                               conv_buf<=dout;
                              else
                               conv_buf(7 downto 0)<=per(dout);
                              end if;
                    end if;
          end if;
. . . . .
conv_buf<=dout;
end if;
end process;
```

Специальные сигналы, использующиеся для «защелки», после установки сигнала в определенное значения, подаются на выход, без применения к ним каких-либо операций.

Алгоритм работы управления криптопроцессорами изображен на рис. 4.12.

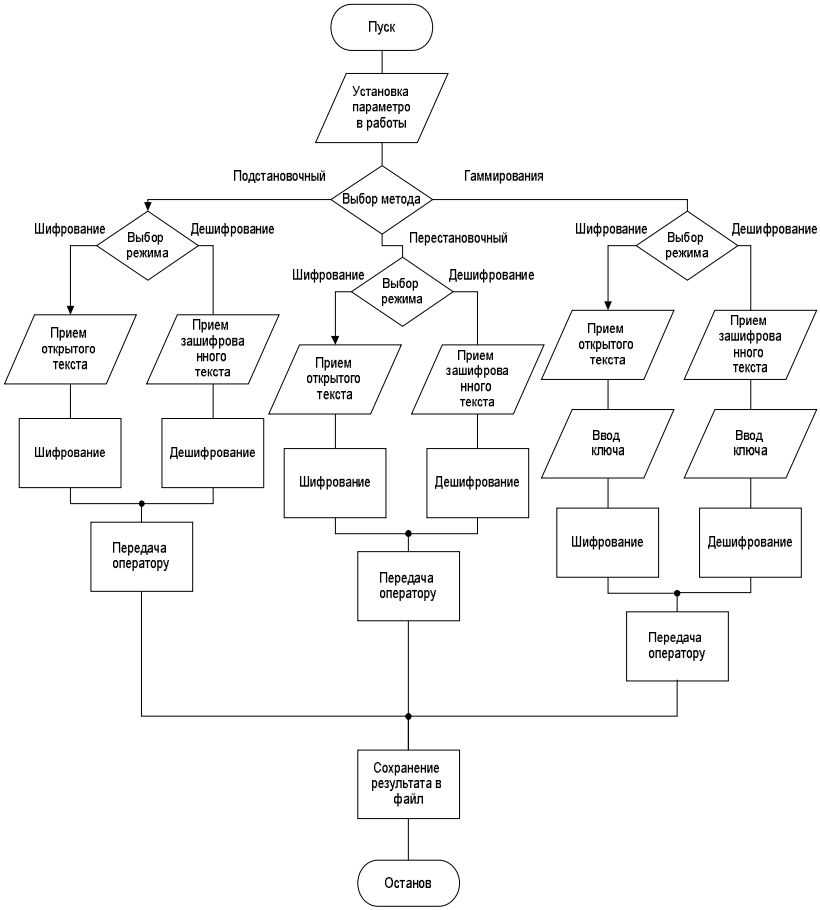


Рис. 4.12. Алгоритм работы управления криптопроцессорами

Листинг управления криптопроцессорами приведен в прил. 5.

Управление работой ПЛИС осуществляется с помощью сервисной программы, в которой задаются параметры работы устройства, и обмен информацией.

Проект реализован на модуле Spartan-3 Starter Kit с установленной на ней ПЛИС Spartan3, параллельные порты компьютера и модуля Spartan3, соединенные загрузочным

кабелем JTAG, служащий для записи прошивки в ПЛИС. Так же к Spartan-3 Starter Kit подсоединяется последовательный кабель имеющий интерфейс RS232, для приема и передачи данных.

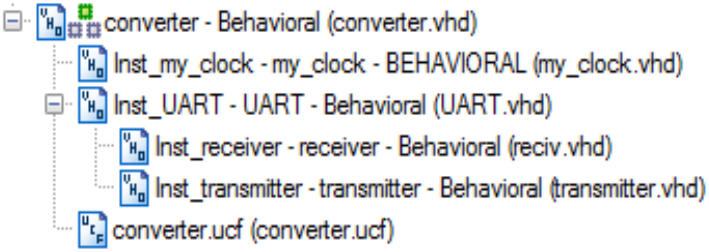


Рис. 4.13. Дерево модулей программы

– модуль Converter- главный модуль, управляющий остальными модулями, объединенных в проект;

– модуль My_clock – задаёт скорость работы(приёма \ передачи) плис на скорости 38400 создан в программе Xilinx Core Generator (программа автоматической генерации модулей);

– модуль UART – связывает модули Receiver и Transmitter в единый Универсальный Асинхронный Приемо-Передатчик;

– модуль Receiver – принимает поступающие биты с COM-порта, отправленные с компьютера, и сохраняет в памяти по 8 бит;

– модуль Transmitter – отправляет уже преобразованные биты на COM-порт, для дальнейшего отображения на компьютере, над которыми были произведены операции шифрация/дешифрация;

– UCF – в нём указывается связь входных\выходных сигналов программы с самой ПЛИС.

Так же к данному проекту подключена библиотека perest.vhd, в которой реализованы все методы шифрования и дешифрования информации.

Исходный код программы всей системы написан на языке VHDL и представляет собой алгоритм передачи информации в ПЛИС через COM-порт и дальнейшей её обработки.

Управление работой ПЛИС осуществляется с помощью сервисной программы, в которой задаются параметры работы устройства и обмена информацией.

ЗАКЛЮЧЕНИЕ

В России в последнее время заметно возрождение интереса к развитию высокотехнологичных производств и, как следствие, к использованию современных средств автоматизация проектирования в электронике (EDA). Рынок ECAD насыщен разнообразными прикладными средствами проектирования.

Службы САПР на предприятиях могут учесть местные специфические требования путем адаптации приобретаемых средств и разработки дополнительных программ, используя инструментальные среды разработки приложений. Также необходимо отметить важность и актуальность интеграции САПР различных производителей.

В работе проведено знакомство с САПР Xilinx ISE Design Suite 12.2; реализация различных проектов на языках VHDL и Verilog, поведенческое моделирование и проверка правильности цифровых устройств функционирования при помощи комплекта ПЛИС Spartan 3 Starter Kit.

Использование пакета ModelSim эффективно для моделирования работы HDL–устройств по заданным тестовым наборам. Тестовые наборы могут задаваться как в виде тестовых окружений, описанных на HDL языках, так и в виде временных диаграмм.

Приведенные практические примеры проектирования устройств цифровой обработки сигналов, микропроцессорных систем, распространенных интерфейсов и программно-аппаратных средств защиты информации позволяют слушателям лучше освоить программирование на высокоуровневых языках VHDL и Verilog и использовать полученные навыки для подготовки курсовых и дипломных проектов.

СПИСОК СОКРАЩЕНИЙ

CAD – (Computer Aided Design) – система автоматизированного проектирования.

CAE – (Computer Aided Engineering) – автоматизированное конструирование.

CAM – (Computer Aided Manufacturing) – система автоматизированного производства.

CALS – (Continious Acquisition and Life–Cycle Support) – информационная поддержка изделия на всех этапах жизненного цикла.

CPLD – (Complex Progammable Logic Device) – программируемые логические микросхемы.

ECAD – (Electronic Computer–Aided Design) – автоматизированная компьютерная система проектирования в электронике.

EDA – (Electronic Design Automation) – автоматизация проектирования в электронике.

FPGA – (Field Programmable Gate Array) – программируемые вентильные матрицы.

HDL – (Hardware Description Language) – язык описания аппаратуры

HSDL – (Hierarchical Scan Description Languages) – специальный язык, являющийся подмножеством VHDL

LRM – (Language Reference Manual) – стандарт языка Verilog

PLI – (Program Language Interface) – стандарт языка Verilog

RTL – (Register Transfer Level) – уровень регистровых задач

VHSIC – (Very high speed integrated circuits) – язык описания аппаратуры интегральных схем

W2W – (Wafer to Wafer) – «пластина к пластине»

ЕСКД – единая система конструкторской документации.

ИМС – Интегральная микросхема

ИС – Интегральная схема

КД – конструкторская документация.

ПО – программное обеспечение.

РЭА– радиоэлектронная аппаратура.

САПР –система автоматизированного проектирования.

СБИС – сверхбольшая интегральная схема.

FSM – (Finite State Machine) - конечный автомат.

XST – (Xilinx Synthesis Technology**)** – технология синтеза Xilinx.

ASCII – (American Standard Code for Information Interchange) - таблица для печатных символов и некоторых специальных кодов.

PLD – (Programmable logic device) - электронный компонент, используемый для создания цифровых интегральных схем.

CPLD – (Complex programmable logic device) - сложные программируемые логические устройства, содержат крупные программируемые логические блоки — макроячейки, соединённые с внешними выводами и внутренними шинами.

FPGA – (Field-programmable gate array) - содержат блоки умножения-суммирования, которые широко применяются при обработке сигналов, а также логические элементы и их блоки коммутации.

ASIC – (Application-specific integrated circuit) - интегральная схема, специализированная для решения конкретной задачи.

ПРИЛОЖЕНИЕ 1

Листинг программы на языке VHDL – «КИХ-фильтр»

```
library IEEE; {подключение библиотек}
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity Basic_FIR_Primitives is {описание интерфейсной части проекта}
  generic (Order : integer := 32;CoefSize : integer :=16; DataSize : integer := 16);
 port( CLK : in STD_LOGIC; Ce : in STD_LOGIC; D : in STD_LOGIC_VECTOR(DataSize-1 downto 0); Q : out STD_LOGIC_VECTOR(47 downto 0) );
end Basic_FIR_Primitives;
architecture behavioural of Basic_FIR_Primitives is {описание архитектуры}
{описание пользовательских типов данных}
type Coefs_type is array(integer range <>) of std_logic_vector(CoefSize-1 downto 0);
signal Int_Coefs : Coefs_type(0 to Order-1);
type Pipe_Type is array(integer range <>) of std_logic_vector(DataSize-1 downto 0);
signal PipeLine : Pipe_Type(0 to Order*2-1);
type MultResType is array(integer range <>) of std_logic_vector(CoefSize+DataSize-1 downto 0);
signal MultRes : MultResType(0 to Order-1);
type integer_vector is array (integer range<>) of integer;
constant Coefs : integer_vector(0 to ORDER-1) :=(
 -23,  -92, -141,  -84,  117,  379,  501,  270,
-348, -1084, -1405, -767, 1046, 3693, 6336, 7987,
7987, 6336, 3693, 1046, -767, -1405, -1084, -348,
 270,  501,  379,  117,  -84, -141,  -92,  -23);
type slv48_array is array (integer range<>) of std_logic_vector(47 downto 0);
signal AddRes : slv48_array(0 to Order-1);
begin {основная программа}
process (CLK) {при поступлении сигнала синхронизации}
begin
  if (CLK'Event and CLK = '1') then
    if (Ce = '1') then
```

```
                for a in 0 to order-1 loop
Int_Coefs(a)<=conv_std_logic_vector(Coefs(a),CoefSize);
                end loop;
                for a in 1 to order*2-1 loop
                   PipeLine(a)<=PipeLine(a-1);
                end loop;
                PipeLine(0)<=D;
                for a in 0 to order-1 loop
                   MultRes(a)<=signed(PipeLine(a*2+1))* signed(Int_Coefs(a));
                end loop;
                AddRes(0)<=sxt(MultRes(0),48);
                for a in 1 to order-1 loop
                   AddRes(a)<=signed(MultRes(a))+ signed(AddRes(a-1));
                end loop;   end if;  end if;
        end process; {завершение процесса}
        Q<=AddRes(order-1); {вывод данных}
        end behavioural;
```

ПРИЛОЖЕНИЕ 2

Листинг программы на языке Verilog – «Программная реализация протокола LIN»

Содержимое файла lin_slave.v

```
module lin_slave(          clk,clk_lin, min_byte, port_in, rx, ve          rst,     cvar,
final);
input wire clk_lin;input wire clk; input wire port_in;
input wire rst; input wire [15:0] min_byte;
input wire [2:0] cvar; output reg [7:0]rx;
output reg [15:0] over; utput reg [2:0] final;
parameter break = 3'b001; parameter syncr = 3'b010;
parameter id = 3'b100; parameter dat = 3'b011;
parameter chek = 3'b110; parameter nothing = 3'b111;
reg [15:0] count1 = 0; reg [7:0] count2 = 0;
reg [9:0] rdata = 0; reg past; reg flag;
always @(posedge (clk)) past = port_in;
always @(posedge (clk))
begin
        if (rst)
        begin           //rx = 7'b01011101;
        End
        case (cvar)
```

```
break:
        begin
        if (port_in == 0) count1 = count1 + 1;
        if ((count1 == min_byte)&(port_in == 1))
                begin
                        final = break; count1 = 0;
                end
        end
syncr:
begin
                if (port_in == 0) flag = 1;
                if (flag == 1)
                begin
                        count1 = count1 + 1;
        if (past ^ port_in) count2 = count2 + 1;
        if (count2 == 9) over = count1 >> 3;
                        if (count2 == 10)
                        begin
                                count2 = 0;
                                count1 = 0;
                                final = syncr;
                        end
                end
        end
id:     begin
                /*if (clk_lin == 1)
                Begin
                        count1 = count1 + 1;
                        if (count1 > 3)
                        begin
                        rdata[9] = port_in;
                        rdata[9:0] = rdata [9:0] >> 1;
                        end
                        if (count1 == 11)
                        begin
                         count1 = 0; final = id;
                        end
                end*/
        end
nothing:
        begin
        end
```

```
        endcase
end
endmodule
```

Содержимое файла lin_master.v

```
module lin_master( clk, port, data, state, rst, number, fin, pid );
        input wire clk; input wire [7:0] data; input wire [7:0] pid;
input wire [2:0] state; input wire [3:0] number; input wire rst; output reg port;
output reg [2:0] fin; parameter break = 3'b001;
parameter syncr = 3'b010; parameter id = 3'b100;
parameter dat = 3'b011; parameter chek = 3'b110;
parameter nothing = 3'b111;
reg [7:0] count = 0; reg [7:0] count_dat = 0;
reg [9:0] sync = 0; reg [9:0] ident = 0;
reg [9:0] cheksum = 0; reg [9:0] dataf = 0;
always @(posedge (clk))
begin
        if (rst)
        begin
count = 0;count_dat = 0; sync = 0; ident = 0; cheksum = 0;    dataf = 0;
        end
        case (state)
        break:
                begin
                        if (count < 13)
                        begin
                                port =0; count = count + 1;
                        end else
                        begin
                        fin = break; count = 0;
                        port = 1; sync = 10'b1010101010;
                        end
                end
        syncr:
                begin
                port = sync[0]; sync [9:0] = sync [9:0] >> 1;
                        count = count + 1;
                        if (count == 10)
                        begin
                ident [0] = 0;      ident [6:1] = pid [5:0];
ident [7] = pid[0] ^ pid[1] ^ pid[2] ^ pid[4];
ident [8] = ~(pid[1] ^ pid[3] ^ pid[4] ^
```

```
pid[5]); ident [9] = 1;
                                        fin = syncr; count = 0;
                                end
                        end
                id:     begin
                                port = ident[0];
                                ident [9:0] = ident [9:0] >> 1;
                                count = count + 1;
                                if (count == 10)
                                begin   fin = id; count = 0;
                                end
                        end
dat:                    begin
                                if (count == 0)
                                begin
                        dataf [0] = 0;      dataf [8:1] = data[7:0];
dataf [9] = 1;
                cheksum = cheksum + data;
                if (cheksum[8] == 1)cheksum = cheksum - 9'hff;
                                end
                                if (count < 10)
                                begin
                                port = dataf[0];                                dataf  [9:0]
= dataf [9:0] >> 1; count = count + 1;
                                end
                                if (count == 10)
                                begin
                                count = 0; count_dat = count_dat + 1;
                                end
                                if (count_dat == number)
                                begin
                                        fin = dat; count_dat = 0;
                                        cheksum = (~cheksum) << 1;
                                        cheksum[9] =1; cheksum[0]=0;
                                end
                        end
                chek:
                        begin
                                port = cheksum[0];
                                cheksum [9:0] = cheksum [9:0] >> 1;
                                count = count + 1;
                                if (count == 11)
```

```
                        begin     count = 0;  fin = nothing;
                        end
                end
        nothing:
                begin
                        port = 1;
                end
        default: fin = nothing;
        endcase
end
endmodule
```

Содержимое файла clk_lin.v

```
module clk_lin(clk,set,clk_lin);
input wire clk; input wire [15:0]set;
output reg clk_lin;
reg [15:0]count = 0; reg [15:0]pred = 0;
always @(posedge (clk))
begin
        if (set != 0)
        begin
                pred <= set;
                if (count ==  pred)
                begin
                        clk_lin <= 1; count = 0;
                end else begin
                        clk_lin <= 0; count = count +1;
                end
        end
end
endmodule
```

Содержимое файла lin_manager.v

```
module lin_manager(lin_state, clk, finaly, slave_state,          finaly_slave, start);
input wire clk; input wire [2:0] finaly;
input wire [2:0] finaly_slave; input wire [2:0] start;
parameter header = 3'b001; parameter response = 3'b010;
parameter frame = 3'b100; parameter break = 3'b001;
parameter syncr = 3'b010; parameter id = 3'b100;
parameter dat = 3'b011; parameter chek = 3'b110;
parameter nothing = 3'b111; output reg [2:0] lin_state;
output reg [2:0] slave_state;
```

```
always @ (posedge (clk))
begin
        case(start)
        header:
                begin
                        lin_state = break;
                        if (finaly == break) lin_state =  syncr;
                        if (finaly == syncr) lin_state =  id;
                        if (finaly == id) lin_state =  nothing;
                end
        response:
                begin
                lin_state = dat;
                if (finaly == dat) lin_state =  chek;
                if (finaly == nothing) lin_state =  nothing;
                end
        frame:
                begin
                        lin_state =  break;
                        if (finaly == break) lin_state =  syncr;
                        if (finaly == syncr) lin_state =  id;
                if (finaly == id) lin_state =  dat;
                if (finaly == dat) lin_state =  chek;
                if (finaly == nothing) lin_state =  nothing;
                                                slave_state = break;
                if (finaly_slave == break) slave_state = syncr;
                if (finaly_slave == syncr) slave_state = id;
                if (finaly_slave == id) slave_state = nothing;
                end
        endcase
end
endmodule
```

Содержимое файла lin_test.v

```
module lin_test;
wire clk_out; wire out; wire [2:0] lin_fin_trans;
wire [2:0] slave_fin_trans; wire [15:0] over_out;
wire [2:0] lin_state_out; wire [2:0] slave_state_out;
reg C; reg C1; reg reset; reg [15:0] mbyte; reg [7:0] dt;
reg [7:0] pd; reg [3:0] numb; reg [2:0] lin_start;
initial
```

```
begin
        C = 0;    C1 = 0;
        dt = 8'b10010010; pd = 8'b11101100;
        numb = 8; mbyte = 6500; lin_start = 3'b100;
end
always #50  C = ~C;
always #25000  C1 = ~C1;
clk_lin lclk (.clk(C), .set(over_out), .clk_lin(clk_out));
lin_master master (.pid(pd), .fin(lin_fin_trans), .port(out), .clk(C1),
.state(lin_state_out), .data(dt), .rst(reset), .number(numb));
lin_slave slave (.final(slave_fin_trans), .over(over_out),
.cvar(slave_state_out),.clk_lin(clk_out), .clk(C), .min_byte(mbyte),
.port_in(out));
lin_manager manager (.start(lin_start), .finaly(lin_fin_trans),
.clk(C),.slave_state(slave_state_out), .finaly_slave
(slave_fin_trans),.lin_state(lin_state_out));
endmodule
```

ПРИЛОЖЕНИЕ 3

Листинг программы на языке Verilog – «Простой UART»

```
module uart ( reset, txclk, ld_tx_data, tx_data, tx_enable, tx_out, tx_empty, rxclk,
uld_rx_data, rx_data, rx_enable, rx_in, rx_empty);
// Объявление портов
input       reset; input       txclk ; input       ld_tx_data ;
input  [7:0] tx_data; input       tx_enable;
output      tx_out; output      tx_empty; input    rxclk;
input            uld_rx_data;  output  [7:0]  rx_data;  input            rx_enable;    input
rx_in;
output      rx_empty;
// Внутренние переменные
reg [7:0]    tx_reg; reg  tx_empty;
reg         tx_over_run; reg [3:0]    tx_cnt;
reg         tx_out; reg [7:0]    rx_reg;reg [7:0]    rx_data;
reg [3:0]    rx_sample_cnt; reg [3:0]    rx_cnt;
reg rx_frame_err; reg   rx_over_run; reg         rx_empty;
reg         rx_d1; reg         rx_d2; reg         rx_busy;
// Логика UARTприема
always @ (posedge rxclk or posedge reset)
if (reset) begin
  rx_reg       <= 0;   rx_data      <= 0;   rx_sample_cnt <= 0;
```

```
 rx_cnt      <= 0;  rx_frame_err <= 0;  rx_over_run  <= 0;  rx_empty     <= 1;
rx_d1        <= 1;  rx_d2        <= 1;  rx_busy      <= 0;
else begin
 // Синхронный и асинхронный сигнал
 rx_d1 <= rx_in;  rx_d2 <= rx_d1;
// Загрузка принятых данных
 if (uld_rx_data) begin
rx_data  <= rx_reg;    rx_empty <= 1;
end
 // Приём данных только тогда, когда разрешен приём
 if (rx_enable) begin
  // Проверка того, что получено начало кадра
if (!rx_busy && !rx_d2) begin
    rx_busy      <= 1; rx_sample_cnt <= 1; rx_cnt       <= 0;
  end
  // Начало кадра обнаружено, переходим к остальным данным
if (rx_busy) begin
     rx_sample_cnt <= rx_sample_cnt + 1;
if (rx_sample_cnt == 7) begin
if ((rx_d2 == 1) && (rx_cnt == 0)) begin
         rx_busy <= 0;
else begin
         rx_cnt <= rx_cnt + 1;
// Начало сохранения данных приёма
         if (rx_cnt > 0 && rx_cnt < 9) begin
rx_reg[rx_cnt - 1] <= rx_d2;
         end
         if (rx_cnt == 9) begin             rx_busy <= 0;
// Проверка конца кадра, т.е. кадр принят корректно
if (rx_d2 == 0) begin
             rx_frame_err <= 1;
else begin
             rx_empty    <= 0;
rx_frame_err <= 0;
             rx_over_run  <= (rx_empty) ? 0 : 1;
end
end
end
end
end
end
 if (!rx_enable) begin    rx_busy <= 0;
```

```
end
end

// UART передатчик
always @ (posedge txclk or posedge reset)
if (reset) begin
  tx_reg       <= 0;   tx_empty      <= 1;
  tx_over_run   <= 0; tx_out        <= 1;   tx_cnt        <= 0;
end else begin
if (ld_tx_data) begin
      if (!tx_empty) begin
        tx_over_run <= 0;
      end else begin
        tx_reg   <= tx_data;
tx_empty <= 0;
end
end
if (tx_enable && !tx_empty) begin
    tx_cnt <= tx_cnt + 1;
    if (tx_cnt == 0) begin       tx_out <= 0;
end
if (tx_cnt > 0 && tx_cnt < 9) begin      tx_out <= tx_reg[tx_cnt -1];
    end
    if (tx_cnt == 9) begin        tx_out <= 1;
tx_cnt <= 0;
      tx_empty <= 1;
end
end
if (!tx_enable) begin tx_cnt <= 0;
end
end
endmodule
```

ПРИЛОЖЕНИЕ 4

Листинг программы на языке VHDL - «Микропроцессорная система»

Содержимое файла logic.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity logic is
  port  (     i : in std_logic_vector(15 downto 0);
              o : out std_logic_vector(7 downto 0)   );
end entity logic;
architecture IMP of logic is
signal o1:std_logic_vector(8 downto 0);
signal rdy : std_logic := '0';
COMPONENT sq_root is
  port (   rdy : out STD_LOGIC;
    x_out : out STD_LOGIC_VECTOR ( 8 downto 0 );
    x_in : in STD_LOGIC_VECTOR ( 15 downto 0 ) );
end component;
begin
o <= o1(7 downto 0);
sroot : sq_root
port map ( rdy => rdy, x_out => o1,            x_in => i   );
end IMP;
```

Содержимое файла user_logic.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;
-- Definition of Generics:
--   C_SLV_DWIDTH              -- Slave interface data bus width
--   C_NUM_REG                 -- Number of software accessible registers
-- Definition of Ports:
--   Bus2IP_Clk                -- Bus to IP clock
--   Bus2IP_Reset              -- Bus to IP reset
--   Bus2IP_Data               -- Bus to IP data bus
```

```
--   Bus2IP_BE                -- Bus to IP byte enables
--   Bus2IP_RdCE               -- Bus to IP read chip enable
--   Bus2IP_WrCE               -- Bus to IP write chip enable
--   IP2Bus_Data              -- IP to Bus data bus
--   IP2Bus_RdAck             -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck             -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error             -- IP to Bus error response
entity user_logic is
  generic
  (  C_SLV_DWIDTH   : integer             := 32;
    C_NUM_REG      : integer             := 8 );
  Port ( i : out std_logic_vector(15 downto 0);
            o : in std_logic_vector(7 downto 0);
    Bus2IP_Clk  : in  std_logic;  Bus2IP_Reset : in  std_logic;
    Bus2IP_Data in  std_logic_vector(0 to C_SLV_DWIDTH-1);     Bus2IP_BE
: in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE     : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE     : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                  : out std_logic_vector(0 to C_SLV_DWIDTH-1);
IP2Bus_RdAck : out std_logic;     IP2Bus_WrAck: out std_logic; IP2Bus_Error
: out std_logic );
  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";
end entity user_logic;
architecture IMP of user_logic is
signal slv_reg0: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg1: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg2: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg3: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg4: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg5: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg6: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg7: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg_write_sel: std_logic_vector(0 to 7);
signal slv_reg_read_sel  : std_logic_vector(0 to 7);
signal slv_ip2bus_data: std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_read_ack: std_logic;
signal slv_write_ack: std_logic;
begin
_WrCE/Bus2IP_RdCE   Memory Mapped Register
  --                 "1000"   C_BASEADDR + 0x0
```

```
 --                "0100"   C_BASEADDR + 0x4
 --                "0010"   C_BASEADDR + 0x8
 --                "0001"   C_BASEADDR + 0xC
 slv_reg_write_sel <= Bus2IP_WrCE(0 to 7);
 slv_reg_read_sel  <= Bus2IP_RdCE(0 to 7);
 slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or
Bus2IP_WrCE(2) or Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5)
or Bus2IP_WrCE(6) or Bus2IP_WrCE(7);
 slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2)
or Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5) or
Bus2IP_RdCE(6) or Bus2IP_RdCE(7);
 -- implement slave model software accessible register(s)
 SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
 begin
  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
   if Bus2IP_Reset = '1' then
    slv_reg0 <= (others => '0'); slv_reg1 <= (others => '0');
    slv_reg2 <= (others => '0'); slv_reg3 <= (others => '0');
    slv_reg4 <= (others => '0'); slv_reg5 <= (others => '0');
    slv_reg6 <= (others => '0');  slv_reg7 <= (others => '0');
   else   i <= slv_reg0;  slv_reg1 <= o;
  case slv_reg_write_sel is
      when "10000000" =>
       for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
        if ( Bus2IP_BE(byte_index) = '1' ) then
         slv_reg0(byte_index*8          to          byte_index*8+7)          <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
         end if;
       end loop;
      when "01000000" =>
       --for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
       --  if ( Bus2IP_BE(byte_index) = '1' ) then
       --                slv_reg1(byte_index*8     to     byte_index*8+7)     <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
       --  end if;
       --end loop;
      when "00100000" =>
       for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
        if ( Bus2IP_BE(byte_index) = '1' ) then
         slv_reg2(byte_index*8          to          byte_index*8+7)          <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
         end if;
```

```
          end loop;
        when "00010000" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
            if ( Bus2IP_BE(byte_index) = '1' ) then
              slv_reg3(byte_index*8             to             byte_index*8+7)             <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
            end if;
          end loop;
        when "00001000" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
            if ( Bus2IP_BE(byte_index) = '1' ) then
              slv_reg4(byte_index*8             to             byte_index*8+7)             <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
            end if;
          end loop;
        when "00000100" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
            if ( Bus2IP_BE(byte_index) = '1' ) then
              slv_reg5(byte_index*8             to             byte_index*8+7)             <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
            end if;
          end loop;
        when "00000010" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
            if ( Bus2IP_BE(byte_index) = '1' ) then
              slv_reg6(byte_index*8             to             byte_index*8+7)             <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
            end if;
          end loop;
        when "00000001" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
            if ( Bus2IP_BE(byte_index) = '1' ) then
slv_reg7(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
            end if;
          end loop;
        when others => null;
      end case;
    end if;
  end if;
 end process SLAVE_REG_WRITE_PROC;
 -- implement slave model software accessible register(s) read mux
```

```
  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1,
slv_reg2, slv_reg3, slv_reg4, slv_reg5, slv_reg6, slv_reg7 ) is
  begin
    case slv_reg_read_sel is
      when "10000000" => slv_ip2bus_data <= slv_reg0;
      when "01000000" => slv_ip2bus_data <= slv_reg1;
      when "00100000" => slv_ip2bus_data <= slv_reg2;
      when "00010000" => slv_ip2bus_data <= slv_reg3;
      when "00001000" => slv_ip2bus_data <= slv_reg4;
      when "00000100" => slv_ip2bus_data <= slv_reg5;
      when "00000010" => slv_ip2bus_data <= slv_reg6;
      when "00000001" => slv_ip2bus_data <= slv_reg7;
      when others => slv_ip2bus_data <= (others => '0');
    end case;
  end process SLAVE_REG_READ_PROC;
  ------------------------------------------
  -- Example code to drive IP to Bus signals
  ------------------------------------------
  IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
                  (others => '0');
  IP2Bus_WrAck <= slv_write_ack;
  IP2Bus_RdAck <= slv_read_ack;
  IP2Bus_Error <= '0';
end IMP;
```

Содержимое файла vhdl_module.vhd

```
library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;
use proc_common_v3_00_a.ipif_pkg.all;
library plbv46_slave_single_v1_01_a;
use plbv46_slave_single_v1_01_a.plbv46_slave_single;
library vhdl_module_v1_00_a;
use vhdl_module_v1_00_a.user_logic;
entity vhdl_module is
  generic
  ( C_BASEADDR: std_logic_vector     := X"FFFFFFFF";
    C_HIGHADDR  : std_logic_vector     := X"00000000";
    C_SPLB_AWIDTH: integer             := 32;
```

```
  C_SPLB_DWIDTH  : integer            := 128;
  C_SPLB_NUM_MASTERS  : integer       := 8;
  C_SPLB_MID_WIDTH : integer          := 3;
  C_SPLB_NATIVE_DWIDTH: integer   := 32;
  C_SPLB_P2P  : integer               := 0;
  C_SPLB_SUPPORT_BURSTS  : integer   := 0;
  C_SPLB_SMALLEST_MASTER : integer        := 32;
  C_SPLB_CLK_PERIOD_PS : integer          := 10000;
  C_INCLUDE_DPHASE_TIMER  : integer  := 1;
  C_FAMILY  : string   := "virtex5"  );
port (   SPLB_Clk : in  std_logic;    SPLB_Rst: in  std_logic;
  PLB_ABus: in  std_logic_vector(0 to 31);
  PLB_UABus  : in  std_logic_vector(0 to 31);
  PLB_PAValid              : in  std_logic;
  PLB_SAValid              : in  std_logic;
  PLB_rdPrim               : in  std_logic;
  PLB_wrPrim               : in  std_logic;
  PLB_masterID : in  std_logic_vector(0 to C_SPLB_MID_WIDTH-1);
  PLB_abort                : in  std_logic;
  PLB_busLock              : in  std_logic;
  PLB_RNW                  : in  std_logic;
  PLB_BE         : in  std_logic_vector(0 to C_SPLB_DWIDTH/8-1);
  PLB_MSize                : in  std_logic_vector(0 to 1);
  PLB_size                 : in  std_logic_vector(0 to 3);
  PLB_type                 : in  std_logic_vector(0 to 2);
  PLB_lockErr              : in  std_logic;
  PLB_wrDBus       : in  std_logic_vector(0 to C_SPLB_DWIDTH-1);
  PLB_wrBurst              : in  std_logic;
  PLB_rdBurst              : in  std_logic;
  PLB_wrPendReq            : in  std_logic;
  PLB_rdPendReq            : in  std_logic;
  PLB_wrPendPri            : in  std_logic_vector(0 to 1);
  PLB_rdPendPri            : in  std_logic_vector(0 to 1);
  PLB_reqPri               : in  std_logic_vector(0 to 1);
  PLB_TAttribute           : in  std_logic_vector(0 to 15);
  Sl_addrAck               : out std_logic;
  Sl_SSize                 : out std_logic_vector(0 to 1);
  Sl_wait                  : out std_logic;
  Sl_rearbitrate           : out std_logic;
  Sl_wrDAck                : out std_logic;
  Sl_wrComp                : out std_logic;
  Sl_wrBTerm               : out std_logic;
```

```
    Sl_rdDBus     : out std_logic_vector(0 to C_SPLB_DWIDTH-1);
    Sl_rdWdAddr                : out std_logic_vector(0 to 3);
    Sl_rdDAck                  : out std_logic;
    Sl_rdComp                  : out std_logic;
    Sl_rdBTerm                 : out std_logic;
    Sl_MBusy                                          :  out  std_logic_vector(0  to
C_SPLB_NUM_MASTERS-1);
    Sl_MWrErr                                         :  out  std_logic_vector(0  to
C_SPLB_NUM_MASTERS-1);
    Sl_MRdErr                                         :  out  std_logic_vector(0  to
C_SPLB_NUM_MASTERS-1);
    Sl_MIRQ                                           :  out  std_logic_vector(0  to
C_SPLB_NUM_MASTERS-1)
    -- DO NOT EDIT ABOVE THIS LINE -);
  attribute SIGIS : string;
  attribute SIGIS of SPLB_Clk      : signal is "CLK";
  attribute SIGIS of SPLB_Rst      : signal is "RST";
end entity vhdl_module;
-- Architecture section
architecture IMP of vhdl_module is
            COMPONENT logic
            PORT(   i : IN std_logic_vector(15 downto 0);
                        o : OUT std_logic_vector(7 downto 0));
            END COMPONENT;
                        signal i : std_logic_vector (15 downto 0);
            signal o : std_logic_vector (7 downto 0);
  -- Array of base/high address pairs for each address range
  constant ZERO_ADDR_PAD                  : std_logic_vector(0 to 31) := (others
=> '0');
  constant  USER_SLV_BASEADDR                       :  std_logic_vector        :=
C_BASEADDR;
  constant  USER_SLV_HIGHADDR                       :  std_logic_vector        :=
C_HIGHADDR;
  constant IPIF_ARD_ADDR_RANGE_ARRAY        : SLV64_ARRAY_TYPE
:=
    (ZERO_ADDR_PAD & USER_SLV_BASEADDR,  -- user logic slave space
base address ZERO_ADDR_PAD & USER_SLV_HIGHADDR    -- user logic
slave space high address );
  -- Array of desired number of chip enables for each address range
  constant USER_SLV_NUM_REG    : integer  := 8;
  constant USER_NUM_REG: integer:= USER_SLV_NUM_REG;
```

```
  constant IPIF_ARD_NUM_CE_ARRAY              : INTEGER_ARRAY_TYPE
:=
    ( 0  => pad_power2(USER_SLV_NUM_REG)  -- number of ce for user logic
slave space );
  -- Ratio of bus clock to core clock (for use in dual clock systems)
  -- 1 = ratio is 1:1
  -- 2 = ratio is 2:1
 constant IPIF_BUS2CORE_CLK_RATIO       : integer      := 1;
  -- Width of the slave data bus (32 only)
constant USER_SLV_DWIDTH: integer  := C_SPLB_NATIVE_DWIDTH;
constant IPIF_SLV_DWIDTH    : integer  := C_SPLB_NATIVE_DWIDTH;
  -- Index for CS/CE
  constant USER_SLV_CS_INDEX             : integer            := 0;
  constant USER_SLV_CE_INDEX             : integer            :=
calc_start_ce_index(IPIF_ARD_NUM_CE_ARRAY, USER_SLV_CS_INDEX);
  constant USER_CE_INDEX  : integer   := USER_SLV_CE_INDEX;
  -- IP Interconnect (IPIC) signal declarations
  signal ipif_Bus2IP_Clk               : std_logic;
  signal ipif_Bus2IP_Reset             : std_logic;
     signal ipif_IP2Bus_Data     : std_logic_vector(0 to IPIF_SLV_DWIDTH-1);
  signal ipif_IP2Bus_WrAck             : std_logic;
  signal ipif_IP2Bus_RdAck             : std_logic;
  signal ipif_IP2Bus_Error             : std_logic;
signal ipif_Bus2IP_Addr : std_logic_vector(0 to C_SPLB_AWIDTH-1);
          signal ipif_Bus2IP_Data  : std_logic_vector(0 to IPIF_SLV_DWIDTH-1);
  signal ipif_Bus2IP_RNW     : std_logic;
signal ipif_Bus2IP_BE : std_logic_vector(0 to IPIF_SLV_DWIDTH/8-1);
  signal ipif_Bus2IP_CS : std_logic_vector(0 to
((IPIF_ARD_ADDR_RANGE_ARRAY'length)/2)-1);
  signal ipif_Bus2IP_RdCE              : std_logic_vector(0 to
calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1);
  signal ipif_Bus2IP_WrCE              : std_logic_vector(0 to
calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1);
signal user_Bus2IP_RdCE : std_logic_vector(0 to USER_NUM_REG-1);
signal user_Bus2IP_WrCE : std_logic_vector(0 to USER_NUM_REG-1);
signal user_IP2Bus_Data : std_logic_vector(0 to USER_SLV_DWIDTH-1);
  signal user_IP2Bus_RdAck             : std_logic;
  signal user_IP2Bus_WrAck             : std_logic;
  signal user_IP2Bus_Error             : std_logic;
begin
  -- instantiate plbv46_slave_single
```

```
 PLBV46_SLAVE_SINGLE_I                    :                entity
plbv46_slave_single_v1_01_a.plbv46_slave_single
   generic map
   (                C_ARD_ADDR_RANGE_ARRAY                    =>
IPIF_ARD_ADDR_RANGE_ARRAY,
    C_ARD_NUM_CE_ARRAY            => IPIF_ARD_NUM_CE_ARRAY,
C_SPLB_P2P        => C_SPLB_P2P,
    C_BUS2CORE_CLK_RATIO=> IPIF_BUS2CORE_CLK_RATIO,
    C_SPLB_MID_WIDTH        => C_SPLB_MID_WIDTH,
    C_SPLB_NUM_MASTERS    => C_SPLB_NUM_MASTERS,
    C_SPLB_AWIDTH           => C_SPLB_AWIDTH,
    C_SPLB_DWIDTH           => C_SPLB_DWIDTH,
    C_SIPIF_DWIDTH          => IPIF_SLV_DWIDTH,
    C_INCLUDE_DPHASE_TIMER       => C_INCLUDE_DPHASE_TIMER,
C_FAMILY     => C_FAMILY    )
   port map
   (     SPLB_Clk              => SPLB_Clk,
    SPLB_Rst              => SPLB_Rst,
    PLB_ABus              => PLB_ABus,
    PLB_UABus             => PLB_UABus,
    PLB_PAValid           => PLB_PAValid,
    PLB_SAValid           => PLB_SAValid,
    PLB_rdPrim            => PLB_rdPrim,
    PLB_wrPrim            => PLB_wrPrim,
    PLB_masterID          => PLB_masterID,
    PLB_abort             => PLB_abort,
    PLB_busLock           => PLB_busLock,
    PLB_RNW               => PLB_RNW,
    PLB_BE                => PLB_BE,
    PLB_MSize             => PLB_MSize,
    PLB_size              => PLB_size,
    PLB_type              => PLB_type,
    PLB_lockErr           => PLB_lockErr,
    PLB_wrDBus            => PLB_wrDBus,
    PLB_wrBurst           => PLB_wrBurst,
    PLB_rdBurst           => PLB_rdBurst,
    PLB_wrPendReq         => PLB_wrPendReq,
    PLB_rdPendReq         => PLB_rdPendReq,
    PLB_wrPendPri         => PLB_wrPendPri,
    PLB_rdPendPri         => PLB_rdPendPri,
    PLB_reqPri            => PLB_reqPri,
    PLB_TAttribute        => PLB_TAttribute,
```

```
    Sl_addrAck                => Sl_addrAck,
    Sl_SSize                  => Sl_SSize,
    Sl_wait                   => Sl_wait,
    Sl_rearbitrate            => Sl_rearbitrate,
    Sl_wrDAck                 => Sl_wrDAck,
    Sl_wrComp                 => Sl_wrComp,
    Sl_wrBTerm                => Sl_wrBTerm,
    Sl_rdDBus                 => Sl_rdDBus,
    Sl_rdWdAddr               => Sl_rdWdAddr,
    Sl_rdDAck                 => Sl_rdDAck,
    Sl_rdComp                 => Sl_rdComp,
    Sl_rdBTerm                => Sl_rdBTerm,
    Sl_MBusy                  => Sl_MBusy,
    Sl_MWrErr                 => Sl_MWrErr,
    Sl_MRdErr                 => Sl_MRdErr,
    Sl_MIRQ                   => Sl_MIRQ,
    Bus2IP_Clk                => ipif_Bus2IP_Clk,
    Bus2IP_Reset              => ipif_Bus2IP_Reset,
    IP2Bus_Data               => ipif_IP2Bus_Data,
    IP2Bus_WrAck              => ipif_IP2Bus_WrAck,
    IP2Bus_RdAck              => ipif_IP2Bus_RdAck,
    IP2Bus_Error              => ipif_IP2Bus_Error,
    Bus2IP_Addr               => ipif_Bus2IP_Addr,
    Bus2IP_Data               => ipif_Bus2IP_Data,
    Bus2IP_RNW                => ipif_Bus2IP_RNW,
    Bus2IP_BE                 => ipif_Bus2IP_BE,
    Bus2IP_CS                 => ipif_Bus2IP_CS,
    Bus2IP_RdCE               => ipif_Bus2IP_RdCE,
    Bus2IP_WrCE               => ipif_Bus2IP_WrCE   );
        Inst_logic: logic PORT MAP(i => i, o => o );
-- instantiate User Logic
USER_LOGIC_I : entity vhdl_module_v1_00_a.user_logic
  generic map   ( C_SLV_DWIDTH   => USER_SLV_DWIDTH,
    C_NUM_REG     => USER_NUM_REG )
  port map   (   i => i, o => o,
    Bus2IP_Clk                => ipif_Bus2IP_Clk,
    Bus2IP_Reset              => ipif_Bus2IP_Reset,
    Bus2IP_Data               => ipif_Bus2IP_Data,
    Bus2IP_BE                 => ipif_Bus2IP_BE,
    Bus2IP_RdCE               => user_Bus2IP_RdCE,
    Bus2IP_WrCE               => user_Bus2IP_WrCE,
    IP2Bus_Data               => user_IP2Bus_Data,
```

```
    IP2Bus_RdAck                => user_IP2Bus_RdAck,
    IP2Bus_WrAck                => user_IP2Bus_WrAck,
    IP2Bus_Error                => user_IP2Bus_Error    );
  -- connect internal signals
  ipif_IP2Bus_Data <= user_IP2Bus_Data;
  ipif_IP2Bus_WrAck <= user_IP2Bus_WrAck;
  ipif_IP2Bus_RdAck <= user_IP2Bus_RdAck;
  ipif_IP2Bus_Error <= user_IP2Bus_Error;
  user_Bus2IP_RdCE      <=      ipif_Bus2IP_RdCE(USER_CE_INDEX      to
USER_CE_INDEX+USER_NUM_REG-1);
  user_Bus2IP_WrCE      <=      ipif_Bus2IP_WrCE(USER_CE_INDEX      to
USER_CE_INDEX+USER_NUM_REG-1);
end IMP;
```

ПРИЛОЖЕНИЕ 5

Листинг программы на языке VHDL «Управление криптопроцессорами»

Содержимое файла Converter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_bit.all;
use ieee.numeric_std.all;
library work;
use work.perest.all;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;
entity converter is
    Port ( CLK0:in STD_LOGIC;    RST : in  STD_LOGIC;
RX : in  STD_LOGIC;     TX : out  STD_LOGIC;
              kl:in std_logic_vector(7 downto 0):="00000000";
              sn:out std_logic_vector(6 downto 0):="1111111";
              an0:out std_logic_vector(3 downto 0):="1111";
                                  leg0:out STD_LOGIC;
                                  leg1:out STD_LOGIC;
                                  lg1:out STD_LOGIC:='0';
                                  lg2:out STD_LOGIC:='0';
                                  lg3:out STD_LOGIC:='0';
```

```
                                lg4:out STD_LOGIC:='0';
                                lg5:out STD_LOGIC:='0';
                                lg6:out STD_LOGIC:='0');
        end converter;
architecture Behavioral of converter is
        COMPONENT my_clock
        PORT(  CLKIN_IN : IN std_logic;
                RST_IN : IN std_logic;
                CLKDV_OUT : OUT std_logic;
                CLKIN_IBUFG_OUT : OUT std_logic;
                CLK0_OUT : OUT std_logic;
                LOCKED_OUT : OUT std_logic);
        END COMPONENT;
        COMPONENT UART
PORT( CLK : IN std_logic;CLK0 : in  STD_LOGIC; RST : IN std_logic;RX : IN
std_logic;
RD : IN std_logic; WR : IN std_logic;
        DIN : IN std_logic_vector(7 downto 0);
        TX : OUT std_logic;        TX_FULL : OUT std_logic;
        RX_EMPTY:OUT std_logic;
FRAME_ERR : out  STD_LOGIC;
        DOUT : OUT std_logic_vector(7 downto 0));
        END COMPONENT;
        signal CLK,CLK0_OUT:std_logic;
        signal CLK_UART:std_logic:='0';
        signal X_EMPTY,RX_EMPTY_OLD:std_logic:='1';
        signal FRAME_ERR:std_logic;
        signal Q:std_logic:='0';
        signal TX_FULL,TX_FULL_OLD:std_logic:='1';
        signal DOUT:std_logic_vector(7 downto 0);
        signal conv_buf:std_logic_vector(7 downto 0);
        signal dir:std_logic_vector(1 downto 0);
        signal n:integer range 0 to 3:=0;
        signal MD:std_logic:='0';
        signal i:integer range 0 to 1:=0;
        signal var:integer range 0 to 6:=0;
        signal s,k,t: integer:=0;
        signal zm,kk: std_logic_vector(7 downto 0);
begin
  leg0<=dir(1); leg1<=dir(0);
        Inst_my_clock: my_clock PORT MAP(
                CLKIN_IN => CLK0, RST_IN => '0',
```

```
                CLKDV_OUT => CLK,
                CLK0_OUT =>CLK0_OUT          );
        Inst_UART: UART PORT MAP(
                CLK => CLK_UART,CLK0 => CLK0_OUT,
                RST => RST,     RX => RX,RD => '1',
                WR => Q, DIN => conv_buf, TX => TX,
                TX_FULL => TX_FULL,
                RX_EMPTY => RX_EMPTY,
                FRAME_ERR =>FRAME_ERR,
DOUT => DOUT );
        process(CLK)begin
        if(CLK'event and clk='1')then
                if(n=3)then
                 CLK_UART<=not CLK_UART; n<=0;
                Else n<=n+1;
                end if;
        end if;
end process;
----Выбор метода
process(clk,dout)begin
        if(clk'event and clk='1')then
if dout="10100111"then-- § подстановочный --var<="01";
                                        var<=1; lg1<='1'; lg2<='0';
                                        lg3<='0'; lg4<='0'; lg5<='0';
                                        lg6<='0';                       elsif
dout="10101100"then-- ¬ перестановочный --var<="11";
                                        var<=2; lg1<='0';
                                        lg2<='1'; lg3<='0';
                                        lg4<='0';lg5<='0';
                                        lg6<='0';
        elsif dout="10101010"then--  Є гаммирование --var<="10";
                                        var<=3;lg1<='0';
                                        lg2<='0'; lg3<='1';
                                        lg4<='0'; lg5<='0';
                                        lg6<='0';
elsif  dout="10011010"then--комбинированный  подстановка  и  перестановка
        --var<="10";
                                        var<=4; lg1<='0';
                                        lg2<='0'; lg3<='0';
                                        lg4<='1'; lg5<='0';
                                        lg6<='0';
```

```
elsif dout="10100101"then-- Ґ комбинированный подстановка и гамма      --
var<="10";
                                        var<=5; lg1<='0';
                                        lg2<='0'; lg3<='0';
                                        lg4<='0';lg5<='1';
                                        lg6<='0';
elsif dout="10011100"then-- њ комбинированный перестановка и гамма     --
var<="10";
                                        var<=6;lg1<='0';
                                        lg2<='0'; lg3<='0';
                                        lg4<='0';lg5<='0';
                                        lg6<='1';
                              end if;
                    if dout="10000110"then-- † шифрование
                              MD<='0';
                              an0<="0111";
                              sn<="0001001";
elsif dout="10110001"then--± дешифрование
                              MD<='1';
                              an0<="0111";
                              sn<="0100001";
                    end if;
          end if;
end process;
-----------------------------------------------------
process (clk) begin
if(clk'event and clk='1')then
kk<=kl;
k<=tran2_10(kk);
end if;
end process;
-----------------------------------------------------
process(var,MD)
begin
if (var=2) then
                    if dout="10101100"then
                    conv_buf<=dout;
          else
--Перестановочный битовый криптопроцессор
--Перестановка 2х пар битов: 7-й и 6-й биты, 3-й и 2-й биты.
                    if (MD='0') then
                              if dout="10000110"then  conv_buf<=dout;
```

```
                    else
                     conv_buf(7 downto 0)<=per(dout);
                    end if;
                elsif (MD='1') then
                    if dout="10110001"then    conv_buf<=dout;
                    else
                      conv_buf(7 downto 0)<=per(dout);
                    end if;
                end if;
        end if;
-------------------------------------------------
elsif (var=1) then
        if dout="10100111"then
                conv_buf<=dout;
        else
---- Подстановочный криптопроцессор
                if (MD='0') then
                    if dout="10000110"then
                            conv_buf<=dout;
                    else
                            --conv_buf<=shifr;
                            shzm(dout,conv_buf);
                    end if;
                elsif (MD='1')then
                    if dout="10110001"then
                            conv_buf<=dout;
                    else
                            dhzm(dout,conv_buf);
                            --conv_buf<=deshifr;
                    end if;
                end if;
        end if;
----гаммирование
elsif (var=3) then
        if dout="10101010"then
                conv_buf<=dout;
        else
                if (MD='0') then
                 if dout="10000110"then
            conv_buf<=dout;
                else conv_buf<=shkl(dout,kl);
                end if;
```

```
                elsif (MD='1')then
                        if dout="10110001"then   conv_buf<=dout;
                        else
                        conv_buf<=dhkl(dout,kl);
                        --conv_buf<=tran10_2(s);
                        end if;
                end if;
        end if;
        elsif (var=4) then
        if dout="10011010"then   conv_buf<=dout;
        else
---- комбинированный криптопроцессор подстановка и перестановка
                if (MD='0') then
                        if dout="10000110"then   conv_buf<=dout;
                        else
                                shzm(dout,zm);
                                conv_buf<=per(zm);
                        end if;
                elsif (MD='1')then
                        if dout="10110001"then   conv_buf<=dout;
                        else
                                zm<=per(dout); dhzm(zm,conv_buf);
                        end if;
                end if;
        end if;
        elsif (var=5) then
        if dout="10100101"then
                conv_buf<=dout;
        else
--комбинированный подстановка гамма
                if (MD='0') then
                        if dout="10000110"then
                                conv_buf<=dout;
                        else
                                shzm(dout,zm);
                                conv_buf<=shkl(zm,kl);
                        end if;
                elsif (MD='1')then
                        if dout="10110001"then
                                conv_buf<=dout;
                        else
                                zm<=dhkl(dout,kl);
```

```
                                        dhzm(zm,conv_buf);
                                end if;
                        end if;
                end if;
                elsif (var=6) then
                if dout="10011100"then
                        conv_buf<=dout;
                else
--комбинированный перестановка гамма
                        if (MD='0') then
                                if dout="10000110"then
                                        conv_buf<=dout;
                                else
                                        zm<=per(dout);
                                        conv_buf<=shkl(zm,kl);
                                end if;
                        elsif (MD='1')then
                                if dout="10110001"then
                                        conv_buf<=dout;
                                else
                                        zm<=dhkl(dout,kl);
                                        conv_buf<=per(zm);
                                end if;
                        end if;
                end if;
else
                conv_buf<=dout;
end if;
end process;
PROCESS(CLK0_OUT)begin
if(CLK0_OUT'event and CLK0_OUT='1')then
if(TX_FULL='0' and TX_FULL_OLD='1')then         Q<='0';
elsif(RX_EMPTY='1' and RX_EMPTY_OLD ='0')then Q<='1';
end if;
RX_EMPTY_OLD<=RX_EMPTY; TX_FULL_OLD<=TX_FULL;
                        end if;
end process;
end Behavioral;
```

Содержимое файла perest.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
package perest is
constant s32:integer := 16#20#; constant s127:integer := 16#7f#;
constant s192:integer := 16#c0#; constant s63:integer := 16#3f#;
constant s128:integer := 16#80#; constant s223:integer := 16#df#;
constant s31:integer := 16#1f#; constant s191:integer := 16#bf#;
constant s64:integer := 16#40#;
constant twenty:std_logic_vector(7 downto 0):= X"20"; --32
constant fourty:std_logic_vector(7 downto 0):= X"40"; --64
----Перестановочный битовый криптопроцессор
function per(signal i: in std_logic_vector(7 downto 0)) return std_logic_vector;
--Перевод из двоичного числа в десятичное
function tran2_10( signal i: in std_logic_vector(7 downto 0)) return integer;
--Перевод из десятичного в двоичное число
function tran10_2(signal i: in integer) return std_logic_vector;

function shkl(signal i,k:in std_logic_vector(7 downto 0)) return std_logic_vector;
function dhkl(signal i,k: in std_logic_vector(7 downto 0)) return
std_logic_vector;
-----Подстановочный криптопроцессор
--Шифрование
Procedure shzm(signal i: in std_logic_vector(7 downto 0);
                                    signal o: out std_logic_vector(7 downto 0));

-----Подстановочный криптопроцессор
--Дещифрование
Procedure dhzm(signal i: in std_logic_vector (7 downto 0);
                                    signal o: out  std_logic_vector(7 downto 0));
end perest;
package body perest is

--Перестановочный битовый криптопроцессор
function per(signal i: in std_logic_vector(7 downto 0)) return std_logic_vector is
variable o,sh:std_logic_vector(7 downto 0);
begin
sh:= i(7)&i(5)&i(6)&i(4)&i(2)&i(3)&i(1 downto 0);
if sh="10100111"or sh="10101100"or sh="10101010"or sh="10000110"or
sh="10110001"or sh="10011010"or sh="10100101"or sh="10011100" then
        o:=i;
else o:=sh;
end if;
```

```
return o;
end function per;
--Перевод из двоичного числа в десятичное
function tran2_10( signal i: in std_logic_vector(7 downto 0)) return integer is
variable x7,x6,x5,x4,x3,x2,x1,x0,numb: integer;
        begin
        if i(7)='1' then x7:=2**7;
        else    x7:=0;
        end if;
        if i(6)='1' then x6:=2**6;
        else x6:=0;
        end if;
        if i(5)='1' then x5:=2**5;
        else x5:=0;
        end if;
        f i(4)='1' then     x4:=2**4;
        else    x4:=0;
        end if;
        if i(3)='1' then x3:=2**3;
        else    x3:=0;
        end if;
        if i(2)='1' then x2:=2**2;
        else    x2:=0;
        end if;
        if i(1)='1' then x1:=2;
        else x1:=0;
        end if;
        if i(0)='1' then x0:=1;
        else x0:=0;
        end if;
        numb:=x7+x6+x5+x4+x3+x2+x1+x0;
return numb;
end function tran2_10;
--Перевод из десятичного в двоичное число
function tran10_2(signal i: in integer) return std_logic_vector is
variable k,n,numbtmp: integer;
variable T: std_logic_vector(7 downto 0);
begin
                numbtmp:=i;
                For n in 7 downto 0 loop
                        k:=numbtmp/(2**n);
                        Numbtmp:=numbtmp rem (2**n);
```

```
                    If k=1 then
                      T(n):='1';
                    Else
                      T(n):='0';
                    End if;
                End loop;
return T;
end function tran10_2;
----------Метод с вводом ключа с ПЛИС
---шифрация
function shkl(signal i,k: in std_logic_vector(7 downto 0)) return
std_logic_vector is
variable o: std_logic_vector(7 downto 0);
variable sh:std_logic_vector(7 downto 0);
--      variable uslsh:std_logic_vector(1 downto 0);
--      variable FRAME_ERR:std_logic;
Begin Sh:=(i+k);
        if (sh < s32) theno:=sh+twenty;
        elsif (sh > s127) and (sh < s192) then
o:=sh+fourty;
        else o:=sh;
end if;
        return o;
end function shkl;
--!дешифрация
function dhkl(signal i,k: in std_logic_vector(7 downto 0)) return
std_logic_vector is
                variable o: std_logic_vector(7 downto 0);
begin

        if i>s31 and i<s64 then o:=i-twenty-k;
        elsif i>s191 then o:=i-fourty-k;
        else o:=i-k;
        end if;
        if o<s32 then           o:=o+twenty;
        elsif (o>s127 and o<s192) then o:=o+fourty;
        end if;
return o;
end function dhkl;
-----Подстановочный криптопроцессор
--Шифрование
Procedure shzm(signal i: in std_logic_vector(7 downto 0);
```

```
signal o: out std_logic_vector(7 downto 0)) is
begin
        case i is

        ----------Входящий символ-------Выходящий символ
when    "00100000"      => o <= "01001100"      ;--     SP L
when    "00100001"      => o <= "01001101"      ;--     !  M
when    "00100010"      => o <= "01001110"      ;--     "  N
when    "00100011"      => o <= "01001111"      ;--     #  O
when    "00100100"      => o <= "01000000"      ;--     $  @
when    "00100101"      => o <= "01000001"      ;--     %  A
when    "00100110"      => o <= "01000010"      ;--     &  B
when    "00100111"      => o <= "01000011"      ;--     '  C
when    "00101000"      => o <= "01101100"      ;--     (  l
when    "00101001"      => o <= "01101101"      ;--     )  m
when    "00101010"      => o <= "01101110"      ;--     *  n
when    "00101011"      => o <= "01101111"      ;--     +  o
when    "00101100"      => o <= "00110100"      ;--     ,  4
when    "00101101"      => o <= "00110101"      ;--     -  5
when    "00101110"      => o <= "00110110"      ;--     .  6
when    "00101111"      => o <= "00110111"      ;--     /  7
when    "00110000"      => o <= "01100000"      ;--     0  `
when    "00110001"      => o <= "01100001"      ;--     1 a
when    "00110010"      => o <= "01100010"      ;--     2 b
when    "00110011"      => o <= "01100011"      ;--     3 c
when    "00110100"      => o <= "01011000"      ;--     4 X
when    "00110101"      => o <= "01011001"      ;--     5 Y
when    "00110110"      => o <= "01011010"      ;--     6 Z
when    "00110111"      => o <= "01011011"      ;--     7 [
when    "00111000"      => o <= "00100100"      ;--     8 $
when    "00111001"      => o <= "00100101"      ;--     9 %
when    "00111010"      => o <= "00100110"      ;--     : &
when    "00111011"      => o <= "00100111"      ;--     ; '
when    "00111100"      => o <= "01110000"      ;--     < p
when    "00111101"      => o <= "01110001"      ;--     = q
when    "00111110"      => o <= "01110010"      ;--     > r
when    "00111111"      => o <= "01110011"      ;--     ? s
when    "01000000"      => o <= "01010100"      ;--     @ T
when    "01000001"      => o <= "01010101"      ;--     A U
when    "01000010"      => o <= "01010110"      ;--     B V
when    "01000011"      => o <= "01010111"      ;--     C W
when    "01000100"      => o <= "00111100"      ;--     D <
```

```
when    "01000101"      => o <= "00111101"      ;--     E =
when    "01000110"      => o <= "00111110"      ;--     F >
when    "01000111"      => o <= "00111111"      ;--     G ?
when    "01001000"      => o <= "01110100"      ;--     H t
when    "01001001"      => o <= "01110101"      ;--     I u
when "01001010"         => o <= "01110110"      ;--     J v
when    "01001011"      => o <= "01110111"      ;--     K w
when    "01001100"      => o <= "01111100"      ;--     L |
when    "01001101"      => o <= "01111101"      ;--     M }
when    "01001110"      => o <= "01111110"      ;--     N ~
when    "01001111"      => o <= "00110011"      ;--     O 3
when    "01010000"      => o <= "00100000"      ;--     P SP
when    "01010001"      => o <= "00100001"      ;--     Q !
when    "01010010"      => o <= "00100010"      ;--     R "
when    "01010011"      => o <= "00100011"      ;--     S #
when    "01010100"      => o <= "01101000"      ;--     T h
when    "01010101"      => o <= "01101001"      ;--     U i
when    "01010110"      => o <= "01101010"      ;--     V j
when    "01010111"      => o <= "01101011"      ;--     W k
when    "01011000"      => o <= "01001000"      ;--     X H
when    "01011001"      => o <= "01001001"      ;--     Y I
when    "01011010"      => o <= "01001010"      ;--     Z J
when    "01011011"      => o <= "01001011"      ;--     [ K
when    "01011100"      => o <= "00101100"      ;--     \ ,
when    "01011101"      => o <= "00101101"      ;--     ] -
when    "01011110"      => o <= "00101110"      ;--     ^ .
when    "01011111"      => o <= "00101111"      ;--     _ /
when    "01100000"      => o <= "01010000"      ;--     ` P
when    "01100001"      => o <= "01010001"      ;--     a Q
when    "01100010"      => o <= "01010010"      ;--     b R
when    "01100011"      => o <= "01010011"      ;--     c S
when    "01100100"      => o <= "00111000"      ;--     d 8
when    "01100101"      => o <= "00111001"      ;--     e 9
when    "01100110"      => o <= "00111010"      ;--     f :
when    "01100111"      => o <= "00111011"      ;--     g ;
when    "01101000"      => o <= "01111000"      ;--     h x
when    "01101001"      => o <= "01111001"      ;--     i y
when    "01101010"      => o <= "01111010"      ;--     j z
when    "01101011"      => o <= "01111011"      ;--     k {
when    "01101100"      => o <= "01011100"      ;--     l \
when    "01101101"      => o <= "01011101"      ;--     m ]
when    "01101110"      => o <= "01011110"      ;--     n ^
```

```
when    "01101111"      => o <= "01011111"      ;--     o _
when    "01110000"      => o <= "00101000"      ;--     p (
when    "01110001"      => o <= "00101001"      ;--     q )
when    "01110010"      => o <= "00101010"      ;--     r *
when    "01110011"      => o <= "00101011"      ;--     s +
when    "01110100"      => o <= "01000100"      ;--     t D
when    "01110101"      => o <= "01000101"      ;--     u E
when    "01110110"      => o <= "01000110"      ;--     v F
when    "01110111"      => o <= "01000111"      ;--     w G
when    "01111000"      => o <= "01100100"      ;--     x d
when    "01111001"      => o <= "01100101"      ;--     y e
when    "01111010"      => o <= "01100110"      ;--     z f
when    "01111011"      => o <= "01100111"      ;--     { g
when    "01111100"      => o <= "00110000"      ;--     | 0
when    "01111101"      => o <= "00110001"      ;--     } 1
when    "01111110"      => o <= "00110010"      ;--     ~ 2
when    "11000000"      => o <= "11001011"      ;--     А Л
when    "11000001"      => o <= "11001100"      ;--     Б М
when    "11000010"      => o <= "11001101"      ;--     В Н
when    "11000011"      => o <= "11010111"      ;--     Г Ч
when    "11000100"      => o <= "11011000"      ;--     Д Ш
when    "11000101"      => o <= "11011001"      ;--     Е Щ
when    "10101000"      => o <= "11101011"      ;--     Ё л
when    "11000110"      => o <= "11101100"      ;--     Ж м
when    "11000111"      => o <= "11101101"      ;--     З н
when    "11001000"      => o <= "11010100"      ;--     И Ф
when    "11001001"      => o <= "11010101"      ;--     Й Х
when    "11001010"      => o <= "11010110"      ;--     К Ц
when    "11001011"      => o <= "11100011"      ;--     Л г
when    "11001100"      => o <= "11100100"      ;--     М д
when    "11001101"      => o <= "11100101"      ;--     Н е
when    "11001110"      => o <= "11111101"      ;--     О э
when    "11001111"      => o <= "11111110"      ;--     П ю
when    "11010000"      => o <= "11111111"      ;--     Р я
when    "11010001"      => o <= "11000011"      ;--     С Г
when    "11010010"      => o <= "11000100"      ;--     Т Д
when    "11010011"      => o <= "11000101"      ;--     У Е
when    "11010100"      => o <= "11100000"      ;--     Ф а
when    "11010101"      => o <= "11100001"      ;--     Х б
when    "11010110"      => o <= "11100010"      ;--     Ц в
when    "11010111"      => o <= "11101000"      ;--     Ч и
when    "11011000"      => o <= "11101001"      ;--     Ш й
```

```
when    "11011001"      => o <= "11101010"      ;--     Щ к
when    "11011010"      => o <= "11101110"      ;--     Ъ о
when    "11011011"      => o <= "11101111"      ;--     Ы п
when    "11011100"      => o <= "11110000"      ;--     Ь р
when    "11011101"      => o <= "11111010"      ;--     Э ъ
when    "11011110"      => o <= "11111011"      ;--     Ю ы
when    "11011111"      => o <= "11111100"      ;--     Я ь
when    "11100000"      => o <= "11001000"      ;--     а И
when    "11100001"      => o <= "11001001"      ;--     б Й
when    "11100010"      => o <= "11001010"      ;--     в К
when    "11100011"      => o <= "11110001"      ;--     г с
when    "11100100"      => o <= "11110010"      ;--     д т
when    "11100101"      => o <= "11110011"      ;--     е у
when    "10111000"      => o <= "11011010"      ;--     ё Ъ
when    "11100110"      => o <= "11011011"      ;--     ж Ы
when    "11100111"      => o <= "11011100"      ;--     з Ь
when    "11101000"      => o <= "11110100"      ;--     и ф
when    "11101001"      => o <= "11110101"      ;--     й х
when    "11101010"      => o <= "11110110"      ;--     к ц
when    "11101011"      => o <= "11000000"      ;--     л А
when    "11101100"      => o <= "11000001"      ;--     м Б
when    "11101101"      => o <= "11000010"      ;--     н В
when    "11101110"      => o <= "11110111"      ;--     о ч
when    "11101111"      => o <= "11111000"      ;--     п ш
when    "11110000"      => o <= "11111001"      ;--     р щ
when    "11110001"      => o <= "11011101"      ;--     с Э
when    "11110010"      => o <= "11011110"      ;--     т Ю
when    "11110011"      => o <= "11011111"      ;--     у Я
when    "11110100"      => o <= "10111000"      ;--     ф ё
when    "11110101"      => o <= "11100110"      ;--     х ж
when    "11110110"      => o <= "11100111"      ;--     ц з
when    "11110111"      => o <= "11001110"      ;--     ч О
when    "11111000"      => o <= "11001111"      ;--     ш П
when    "11111001"      => o <= "11010000"      ;--     щ Р
when    "11111010"      => o <= "11010001"      ;--     ъ С
when    "11111011"      => o <= "11010010"      ;--     ы Т
when    "11111100"      => o <= "11010011"      ;--     ь У
when    "11111101"      => o <= "10101000"      ;--     э Ё
when    "11111110"      => o <= "11000110"      ;--     ю Ж
when    "11111111"      => o <= "11000111"      ;--     я З
when  "00001010"  => o <=                 "00001010"      ; --    LF
when  "00001101"  => o <=                 "00001101"      ; --    CR
```

```
when others => o <= i;
        end case;
end shzm;


-----Подстановочный криптопроцессор
--Дешифрование
Procedure dhzm(signal i: in std_logic_vector(7 downto 0);   signal   o:   out
std_logic_vector(7 downto 0)) is
begin
        case i is
when    "01001100"      => o <= "00100000"      ;--     L SP
when    "01001101"      => o <= "00100001"      ;--     M !
when    "01001110"      => o <= "00100010"      ;--     N "
when    "01001111"      => o <= "00100011"      ;--     O #
when    "01000000"      => o <= "00100100"      ;--     @ $
when    "01000001"      => o <= "00100101"      ;--     A %
when    "01000010"      => o <= "00100110"      ;--     B &
when    "01000011"      => o <= "00100111"      ;--     C '
when    "01101100"      => o <= "00101000"      ;--     l (
when    "01101101"      => o <= "00101001"      ;--     m )
when    "01101110"      => o <= "00101010"      ;--     n *
when    "01101111"      => o <= "00101011"      ;--     o +
when    "00110100"      => o <= "00101100"      ;--     4 ,
when    "00110101"      => o <= "00101101"      ;--     5 -
when    "00110110"      => o <= "00101110"      ;--     6 .
when    "00110111"      => o <= "00101111"      ;--     7 /
when    "01100000"      => o <= "00110000"      ;--     ` 0
when    "01100001"      => o <= "00110001"      ;--     a 1
when    "01100010"      => o <= "00110010"      ;--     b 2
when    "01100011"      => o <= "00110011"      ;--     c 3
when    "01011000"      => o <= "00110100"      ;--     X 4
when    "01011001"      => o <= "00110101"      ;--     Y 5
when    "01011010"      => o <= "00110110"      ;--     Z 6
when    "01011011"      => o <= "00110111"      ;--     [ 7
when    "00100100"      => o <= "00111000"      ;--     $ 8
when    "00100101"      => o <= "00111001"      ;--     % 9
when    "00100110"      => o <= "00111010"      ;--     & :
when    "00100111"      => o <= "00111011"      ;--     ' ;
when    "01110000"      => o <= "00111100"      ;--     p <
when    "01110001"      => o <= "00111101"      ;--     q =
when    "01110010"      => o <= "00111110"      ;--     r >
```

```
when    "01110011"    => o <= "00111111"    ;--    s ?
when    "01010100"    => o <= "01000000"    ;--    T @
when    "01010101"    => o <= "01000001"    ;--    U A
when    "01010110"    => o <= "01000010"    ;--    V B
when    "01010111"    => o <= "01000011"    ;--    W C
when    "00111100"    => o <= "01000100"    ;--    < D
when    "00111101"    => o <= "01000101"    ;--    = E
when    "00111110"    => o <= "01000110"    ;--    > F
when    "00111111"    => o <= "01000111"    ;--    ? G
when    "01110100"    => o <= "01001000"    ;--    t H
when    "01110101"    => o <= "01001001"    ;--    u I
when    "01110110"    => o <= "01001010"    ;--    v J
when    "01110111"    => o <= "01001011"    ;--    w K
when    "01111100"    => o <= "01001100"    ;--    | L
when    "01111101"    => o <= "01001101"    ;--    } M
when    "01111110"    => o <= "01001110"    ;--    ~ N
when    "00110011"    => o <= "01001111"    ;--    3 O
when    "00100000"    => o <= "01010000"    ;--    SP P
when    "00100001"    => o <= "01010001"    ;--    ! Q
when    "00100010"    => o <= "01010010"    ;--    " R
when    "00100011"    => o <= "01010011"    ;--    # S
when    "01101000"    => o <= "01010100"    ;--    h T
when    "01101001"    => o <= "01010101"    ;--    i U
when    "01101010"    => o <= "01010110"    ;--    j V
when    "01101011"    => o <= "01010111"    ;--    k W
when    "01001000"    => o <= "01011000"    ;--    H X
when    "01001001"    => o <= "01011001"    ;--    I Y
when    "01001010"    => o <= "01011010"    ;--    J Z
when    "01001011"    => o <= "01011011"    ;--    K [
when    "00101100"    => o <= "01011100"    ;--    , \
when    "00101101"    => o <= "01011101"    ;--    - ]
when    "00101110"    => o <= "01011110"    ;--    . ^
when    "00101111"    => o <= "01011111"    ;--    / _
when    "01010000"    => o <= "01100000"    ;--    P `
when    "01010001"    => o <= "01100001"    ;--    Q a
when    "01010010"    => o <= "01100010"    ;--    R b
when    "01010011"    => o <= "01100011"    ;--    S c
when    "00111000"    => o <= "01100100"    ;--    8 d
when    "00111001"    => o <= "01100101"    ;--    9 e
when    "00111010"    => o <= "01100110"    ;--    : f
when    "00111011"    => o <= "01100111"    ;--    ; g
when    "01111000"    => o <= "01101000"    ;--    x h
```

```
when    "01111001"    => o <= "01101001"    ;--    y i
when    "01111010"    => o <= "01101010"    ;--    z j
when    "01111011"    => o <= "01101011"    ;--    { k
when    "01011100"    => o <= "01101100"    ;--    \ l
when    "01011101"    => o <= "01101101"    ;--    ] m
when    "01011110"    => o <= "01101110"    ;--    ^ n
when    "01011111"    => o <= "01101111"    ;--    _ o
when    "00101000"    => o <= "01110000"    ;--    ( p
when    "00101001"    => o <= "01110001"    ;--    ) q
when    "00101010"    => o <= "01110010"    ;--    * r
when    "00101011"    => o <= "01110011"    ;--    + s
when    "01000100"    => o <= "01110100"    ;--    D t
when    "01000101"    => o <= "01110101"    ;--    E u
when    "01000110"    => o <= "01110110"    ;--    F v
when    "01000111"    => o <= "01110111"    ;--    G w
when    "01100100"    => o <= "01111000"    ;--    d x
when    "01100101"    => o <= "01111001"    ;--    e y
when    "01100110"    => o <= "01111010"    ;--    f z
when    "01100111"    => o <= "01111011"    ;--    g {
when    "00110000"    => o <= "01111100"    ;--    0 |
when    "00110001"    => o <= "01111101"    ;--    1 }
when    "00110010"    => o <= "01111110"    ;--    2 ~
when    "11001011"    => o <= "11000000"    ;--    Л А
when    "11001100"    => o <= "11000001"    ;--    М Б
when    "11001101"    => o <= "11000010"    ;--    Н В
when    "11010111"    => o <= "11000011"    ;--    Ч Г
when    "11011000"    => o <= "11000100"    ;--    Ш Д
when    "11011001"    => o <= "11000101"    ;--    Щ Е
when    "11101011"    => o <= "10101000"    ;--    л Ё
when    "11101100"    => o <= "11000110"    ;--    м Ж
when    "11101101"    => o <= "11000111"    ;--    н З
when    "11010100"    => o <= "11001000"    ;--    Ф И
when    "11010101"    => o <= "11001001"    ;--    Х Й
when    "11010110"    => o <= "11001010"    ;--    Ц К
when    "11100011"    => o <= "11001011"    ;--    г Л
when    "11100100"    => o <= "11001100"    ;--    д М
when    "11100101"    => o <= "11001101"    ;--    е Н
when    "11111101"    => o <= "11001110"    ;--    э О
when    "11111110"    => o <= "11001111"    ;--    ю П
when    "11111111"    => o <= "11010000"    ;--    я Р
when    "11000011"    => o <= "11010001"    ;--    Г С
when    "11000100"    => o <= "11010010"    ;--    Д Т
```

```
when    "11000101"    => o <= "11010011"    ;--    Е У
when    "11100000"    => o <= "11010100"    ;--    а Ф
when    "11100001"    => o <= "11010101"    ;--    б Х
when    "11100010"    => o <= "11010110"    ;--    в Ц
when    "11101000"    => o <= "11010111"    ;--    и Ч
when    "11101001"    => o <= "11011000"    ;--    й Ш
when    "11101010"    => o <= "11011001"    ;--    к Щ
when    "11101110"    => o <= "11011010"    ;--    о Ъ
when    "11101111"    => o <= "11011011"    ;--    п Ы
when    "11110000"    => o <= "11011100"    ;--    р Ь
when    "11111010"    => o <= "11011101"    ;--    ъ Э
when    "11111011"    => o <= "11011110"    ;--    ы Ю
when    "11111100"    => o <= "11011111"    ;--    ь Я
when    "11001000"    => o <= "11100000"    ;--    И а
when    "11001001"    => o <= "11100001"    ;--    Й б
when    "11001010"    => o <= "11100010"    ;--    К в
when    "11110001"    => o <= "11100011"    ;--    с г
when    "11110010"    => o <= "11100100"    ;--    т д
when    "11110011"    => o <= "11100101"    ;--    у е
when    "11011010"    => o <= "10111000"    ;--    Ъ ё
when    "11011011"    => o <= "11100110"    ;--    Ы ж
when    "11011100"    => o <= "11100111"    ;--    Ь з
when    "11110100"    => o <= "11101000"    ;--    ф и
when    "11110101"    => o <= "11101001"    ;--    х й
when    "11110110"    => o <= "11101010"    ;--    ц к
when    "11000000"    => o <= "11101011"    ;--    А л
when    "11000001"    => o <= "11101100"    ;--    Б м
when    "11000010"    => o <= "11101101"    ;--    В н
when    "11110111"    => o <= "11101110"    ;--    ч о
when    "11111000"    => o <= "11101111"    ;--    ш п
when    "11111001"    => o <= "11110000"    ;--    щ р
when    "11011101"    => o <= "11110001"    ;--    Э с
when    "11011110"    => o <= "11110010"    ;--    Ю т
when    "11011111"    => o <= "11110011"    ;--    Я у
when    "10111000"    => o <= "11110100"    ;--    ё ф
when    "11100110"    => o <= "11110101"    ;--    ж х
when    "11100111"    => o <= "11110110"    ;--    з ц
when    "11001110"    => o <= "11110111"    ;--    О ч
when    "11001111"    => o <= "11111000"    ;--    П ш
when    "11010000"    => o <= "11111001"    ;--    Р щ
when    "11010001"    => o <= "11111010"    ;--    С ъ
when    "11010010"    => o <= "11111011"    ;--    Т ы
```

```
when    "11010011"      => o <= "11111100"      ;--     У ь
when    "10101000"      => o <= "11111101"      ;--     Ё э
when    "11000110"      => o <= "11111110"      ;--     Ж ю
when    "11000111"      => o <= "11111111"      ;--     З я
when  "00001010"        => o <= "00001010"      ; --    LF
when  "00001101"        => o <= "00001101"      ; --    CR
          when others => o <= i;
          end case;
          end dhzm;
          end perest;
```

Содержимое файла my_clock.vhd

```
library ieee; use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL; library UNISIM;
use UNISIM.Vcomponents.ALL;
entity my_clock is
  port ( CLKIN_IN : in std_logic; RST_IN : in std_logic;
CLKDV_OUT: out std_logic; CLKIN_IBUFG_OUT : out std_logic;
CLK0_OUT : out std_logic; LOCKED_OUT: out std_logic);
end my_clock;
architecture BEHAVIORAL of my_clock is
 signal CLKDV_BUF: std_logic;signal CLKFB_IN : std_logic; signal
CLKIN_IBUFG: std_logic;
 signal CLK0_BUF:std_logic;signal GND_BIT : std_logic;
begin
  GND_BIT <= '0';
  CLKIN_IBUFG_OUT <= CLKIN_IBUFG;
  CLK0_OUT <= CLKFB_IN;
  CLKDV_BUFG_INST : BUFG
    port map (I=>CLKDV_BUF, O=>CLKDV_OUT);
  CLKIN_IBUFG_INST : IBUFG
    port map (I=>CLKIN_IN, O=>CLKIN_IBUFG);
CLK0_BUFG_INST : BUFG
    port map (I=>CLK0_BUF, O=>CLKFB_IN);  DCM_INST : DCM
  generic map( CLK_FEEDBACK => "1X",
   CLKDV_DIVIDE => 10.0, CLKFX_DIVIDE => 1,
   CLKFX_MULTIPLY => 4, CLKIN_DIVIDE_BY_2 => FALSE,
CLKIN_PERIOD => 10.000,
    CLKOUT_PHASE_SHIFT => "NONE",
    DESKEW_ADJUST=> "SYSTEM_SYNCHRONOUS",
     DFS_FREQUENCY_MODE => "LOW",
     DLL_FREQUENCY_MODE => "LOW",
```

```
    DUTY_CYCLE_CORRECTION => TRUE,
     FACTORY_JF => x"8080",  PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=>CLKFB_IN, CLKIN=>CLKIN_IBUFG,
DSSEN=>GND_BIT, PSCLK=>GND_BIT, PSEN=>GND_BIT,
PSINCDEC=>GND_BIT, RST=>RST_IN, CLKDV=>CLKDV_BUF,
CLKFX=>open, CLKFX180=>open, CLK0=>CLK0_BUF,
CLK2X=>open, CLK2X180=>open, CLK90=>open,
CLK180=>open, CLK270=>open,
LOCKED=>LOCKED_OUT, PSDONE=>open, STATUS=>open);
end BEHAVIORAL;
```

UART.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity UART is
   Port ( CLK : in STD_LOGIC; CLK0 : in STD_LOGIC; RST : in
STD_LOGIC; RX : in STD_LOGIC; RD : in STD_LOGIC; WR : in
STD_LOGIC; DIN : in STD_LOGIC_VECTOR (7 downto 0); TX : out
STD_LOGIC; TX_FULL : out STD_LOGIC; RX_EMPTY : out
STD_LOGIC; FRAME_ERR: out STD_LOGIC;DOUT : out
STD_LOGIC_VECTOR (7 downto 0));
end UART;

architecture Behavioral of UART is
        COMPONENT receiver
        PORT( CLK : IN std_logic; RST : IN std_logic; RX : IN
std_logic; RD_UART : IN std_logic; RX_EMPTY : OUT std_logic;
FRAME_ERR : out STD_LOGIC; DOUT : OUT std_logic_vector(7 downto 0);
n8: out std_logic);
        END COMPONENT;
        COMPONENT transmitter
        PORT( CLK : IN std_logic; RST : IN std_logic;
        DIN : IN std_logic_vector(7 downto 0); WR_UART : IN std_logic;
TX_FULL : OUT std_logic; TX : OUT std_logic);
        END COMPONENT;
signal rx_empt:std_logic;
signal data:std_logic_vector(7 downto 0);
signal clk_out:std_logic;
signal n8:std_logic;
```

```
begin
Inst_receiver: receiver PORT MAP( CLK => CLK,
RST => RST,     RX => RX, RD_UART => RD,     RX_EMPTY                => rx_empt,FRAME_ERR =>FRAME_ERR, DOUT => data, n8=>n8);
Inst_transmitter: transmitter PORT MAP( CLK => CLK,
RST => RST,     DIN => DIN,     WR_UART => WR,         TX_FULL         => TX_FULL, TX => TX);
RX_EMPTY<=rx_empt; DOUT<=data;
end Behavioral;
```

-- Receiver.vhd

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- any Xilinx primitives in this code.
--library UNISIM; --use UNISIM.VComponents.all;
entity receiver is
   Port ( CLK : in  STD_LOGIC; RST: in  STD_LOGIC; RX : in  STD_LOGIC; RD_UART : in  STD_LOGIC; RX_EMPTY : out  STD_LOGIC;  FRAME_ERR : out  STD_LOGIC:='0'; DOUT : out  STD_LOGIC_VECTOR (7 downto 0):="11111111";  n8: out std_logic);
end receiver;
architecture Behavioral of receiver is
type state_type is (idle,start,data,stop);
signal state_reg,state_next:state_type:=idle;
signal s,s_next:integer range 0 to 31:=0;
signal n,n_next:integer range 0 to 511:=0;
signalfifo_reg,fifo_next,  dout_reg,  dout_next:  std_logic_vector(7  downto 0):="11111111";
signal rx_reg,rx_next:std_logic:='0';
signal frame_error,frame_err_next:std_logic:='0';
signal n8_reg,n8_next:std_logic:='1';
begin
--STATE
process(CLK,RST)
begin
if(RST='1')then
          state_reg<=idle; fifo_reg<=(others=>'0');
          dout_reg<=(others=>'0');  s<=0;    n<=0;
          frame_error<='0';          rx_reg<='0';       n8_reg<='1';
elsif(CLK'event and CLK='1')then
```

```
state_reg<=state_next; s<=s_next;  n<=n_next;
fifo_reg<=fifo_next;        dout_reg<=dout_next;
rx_reg<=rx_next;frame_error<=frame_err_next; n8_reg<=n8_next;
end if;
end process;
process(state_reg,s,n,fifo_reg,rx,RD_UART,dout_reg,rx_reg,frame_error)
begin
state_next<=state_reg; s_next<=s; n_next<=n;
fifo_next<=fifo_reg; dout_next<=dout_reg; rx_next<=rx_reg; n8_next<=n8_reg;
RX_EMPTY<='0';
frame_err_next<=frame_error;
case state_reg is
when idle=>
        RX_EMPTY<='1';
        if(rx='0' and RD_UART='1')then    s_next<=0;
                state_next<=start;
        end if;
when start=>
        FRAME_ERR_NEXT<='0';
        if(s=7)then         s_next<=0; n_next<=0;
                state_next<=data;
        else      s_next<=s+1;
        end if;
when data=>
        if(s=13)then        rx_next<=rx;      s_next<=s+1;
        elsif(s=14)then   s_next<=s+1;
                if(rx/=rx_reg) then           frame_err_next<='1';
                end if;
        elsif(s=15)then
                if(rx=rx_reg)then
                        fifo_next<=rx&fifo_reg(7 downto 1);
                        if(n=7)then         state_next<=stop;
                                s_next<=0; n_next<=0;
                        else n_next<=n+1; s_next<=0;
                        end if;
                else      FRAME_ERR_NEXT<='1';
                end if;
        else      s_next<=s+1;
        end if;
when stop=>
        if(s=15)then state_next<=idle; dout_next<=fifo_reg;
        else s_next<=s+1;
```

```
            end if;
end case;
end process;
dout<=dout_reg; n8<=n8_reg;
FRAME_ERR<=frame_error;
end Behavioral;
```

-- **Transmitter.vhd**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating ---- any Xilinx primitives in this code.
--library UNISIM; --use UNISIM.VComponents.all;
entity transmitter is
   Port ( CLK : in  STD_LOGIC;  RST : in  STD_LOGIC;           DIN : in STD_LOGIC_VECTOR (7 downto 0); WR_UART : in  STD_LOGIC; TX_FULL : out  STD_LOGIC; TX : out  STD_LOGIC);
end transmitter;
architecture Behavioral of transmitter is
type state_type is (idle,start,data,stop);
signal state_reg,state_next:state_type;
signal fifo,fifo_next:std_logic_vector(7 downto 0);
signal s,s_next:integer range 0 to 15;
signal n,n_next:integer range 0 to 7;
begin
process(CLK,RST)
begin
if(RST='1')then fifo<=(others=>'0');          state_reg<=idle;
elsif(CLK'event and CLK='1')then
            fifo<=fifo_next;  state_reg<=state_next;
            s<=s_next;          n<=n_next;
end if;
end process;
process(DIN,WR_UART,FIFO,STATE_REG,s,n)
begin
state_next<=state_reg;
fifo_next<=fifo; s_next<=s; n_next<=n;
TX_FULL<='0';
case state_reg is
when idle =>       TX<='1';                    TX_FULL<='1';
```

```
        if(WR_UART='1')then
                s_next<=0;
                state_next<=start;
                fifo_next<=DIN;
        end if;
when start =>   TX<='0';
        if(s=15)then
                s_next<=0; n_next<=0; state_next<=data;
        else s_next<=s+1;
        end if;
when data =>
        TX<=FIFO(0);
        if(s=15)then    fifo_next<='0'&fifo(7 downto 1);
                if(n=7)then state_next<=stop; s_next<=0;
                else    n_next<=n+1; s_next<=0;
                end if;
        else s_next<=s+1;
        end if;
when stop =>    TX<='1';
        if(s=15) then   state_next<=idle;
        else    s_next<=s+1;
        end if;
end case;
end process;
end Behavioral;
```

ПРИЛОЖЕНИЕ 6

Пример вариантов индивидуальных заданий на языке Verilog

Вариант 1

Описать на языке Verilog однобитный цифровой компаратор, определяющий равенство двух входных сигналов. Таблица истинности и логическое уравнение однобитного цифрового компаратора:

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

$$y = x_1 x_2 \vee \overline{x}_1 \overline{x}_2$$

```
//*****************************
// Модифицированное описание 1-битного компаратора
//*****************************
// Описание модуля
module_rtl_comparer
                    // Описание портов ввода-вывода
                    (
                        input input1,
                        input input2,
                        output equal
                    );
                    // Опиание вспомогательных сигналов
                    reg local_equal;
                    // Описание тела модуля
                    always @*
                      begin
                        if (input1== input2)
                            local_equal=1'b1;
                       else
                            local_egual=1'b0;
                      end
                    assign equal=local_equal;
endmodule; rtl_comparer
```

Вариант 2

Описать на языке Verilog однотактный синхронный D-триггер. Таблица истинности:

D	Q(t)	Q(t+1)
0	0	0
0	1	0
1	0	1
1	1	1

```
//*****************
// Опиание триггера D-типа
//*****************
module d_flip_flop (
                        input clk,
                        input data_in,
                        output reg data_out
                        );
                    // Описание вспомогательных сигналов
                    // Описание тела модуля
                    always @(posedge clk)
                        data_out<=data_in;
endmodule: d_flip_flop
```

Вариант 3

Описать на языке Verilog восьмиразрядный двоичный счетчик. Блок будет прибавлять единицу к своему значению каждый такт. В случае асинхронного сброса или сигнала сброса, счетчик принимает свое значение равным нулю. В случае переполнения, счетчик продолжает считать с нулевого значения.

```
// ******************
// Двоичный счетчик
// ******************
// Описание модуля
module counter
   // Описание портов ввода-вывода
   (
      input clk,
      input reset,
      input to_zero,
      output [7:0] data
   );
   // Описание вспомогательных сигналов
   reg data_reg;
   wire data_plus_one;
   // Описание тела модуля
   always @(posedge clk, posedge reset)
      begin
         if (reset)
            data_reg <= 0;
         else if (to_zero)
            data_reg <= 0;
         else
            data_reg <= data_plus_one;
      end

   assign data_plus_one = data_reg + 1'b1;

   assign data = data_reg;
endmodule : counter
```

Вариант 4

Описать на языке Verilog N-битный цифровой компаратор, определяющий равенство двух входных сигналов. Разрядность N компаратора должна передаваться как параметр. Таблица истинности и логическое уравнение цифрового компаратора:

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

$$y = x_1 x_2 \vee \overline{x}_1 \overline{x}_2$$

```
//********************
//N-битный компаратор
//********************
//Описание модуля
module rtl_comparer_params
        // Описание параметров
        #(
        parameter WIDTH=5
        )
        // Описание портов ввода-выода
        (
        input [WIDTH-1:0] input1,
        input [WIDTH-1:0] input2,
        output equal;
        );
        // Описание вспомогательных сигналов
        reg local_equal;
        // Описание тела модуля
        always @*
                begin
                if(input1=input2)
                local_equal=1'b1;
                else
                local_equal=1'b0;
                end
        assign egual = local_equal;
endmodule: rtl_comparer_params
```

Вариант 5

Описать на языке Verilog однотактный синхронный D-триггер с асинхронным сбросом. Таблица истинности:

D	Q(t)	Q(t+1)
0	0	0
0	1	0
1	0	1
1	1	1

```
//************************
// Описаниетриггера D-типа
// с ассинхроным сбросом
//************************
module d_flip_flop_with_reset(
                                input clk,
                                input reset,
                                input data_in,
                                output reg data_out
                                );
  |// Описание вспомогательных сигналов
  // Описание тела модуля
always @(posedge clk, posedge reset)
        if (reset)
            data-out<=0;
    else
        data_out<=data_in;
endmodule : d-flip_flop_with_reset
```

Вариант 6

Описать на языке Verilog полный дешифратор из 3 в 8. Таблица истинности устройства приведены ниже.

Inputs			Outputs							
X	Y	Z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	**1**	0	0	0	0	0	0	0
0	0	1	0	**1**	0	0	0	0	0	0
0	1	0	0	0	**1**	0	0	0	0	0
0	1	1	0	0	0	**1**	0	0	0	0
1	0	0	0	0	0	0	**1**	0	0	0
1	0	1	0	0	0	0	0	**1**	0	0
1	1	0	0	0	0	0	0	0	**1**	0
1	1	1	0	0	0	0	0	0	0	**1**

```
//-------------------------------------------------------
// Описание дешифратора
//-------------------------------------------------------
module decoder_using_case (
   binary_in   , //  3 bit binary input
   decoder_out , //  8-bit  out
   enable        //  Enable for the decoder
   );
input [2:0] binary_in  ;
input  enable ;
output [7:0] decoder_out ;

reg [7:0] decoder_out ;

always @ (enable or binary_in)
begin
  decoder_out = 0;
  if (enable) begin
    case (binary_in)
      4'h0 : decoder_out = 8'b00000001;
      4'h1 : decoder_out = 8'b00000010;
      4'h2 : decoder_out = 8'b00000100;
```

```
      4'h3 : decoder_out = 8'b00001000;
      4'h4 : decoder_out = 8'b00010000;
      4'h5 : decoder_out = 8'b00100000;
      4'h6 : decoder_out = 8'b01000000;
      4'h7 : decoder_out = 8'b10000000;
    endcase
  end
 end

endmodule
```

Вариант 7

Описать на языке Verilog простейшее АЛУ, которое на вход будет принимать два 32-х разрядных операнда. Устройство должно выполнять операции сложения, умножения, логического умножения и логического сложения.

```
//**************
// Описание АЛУ
//**************
//Описание модуля
module rtl_alu
      //Описание портов ввода-вывода
(
       input [31:0] operand1,
       input [31:0] operand2,
       input [2:0] cmd,
      output [31:0] result,
      output error
);
//Описание вспомогательных сигналов
  reg [31:0] local_result;

//Описание тела модуля
always@*
    begin
       case(cmd)
            3'b000: local_result=operand1 + operand2;
            3'b001: local_result=operand1 - operand2;
            3'b010: local_result=operand1 & operand2;
            3'b011: local_result=operand1 | operand2;
            3'b100: local_result=operand1 <<1;
            3'b101: local_result=operand1 >>1;
      default: local_result = ((cmd == 3'b111))
  endcase
end
assign error =((cmd==3'b110)|| (cmd==3'b111))
          ?1'b1 : 1' b0;
          assign result = local result;
endmodule : rtl_alu
```

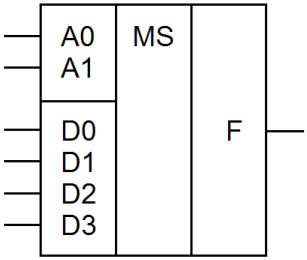
Вариант 8

Описать на языке Verilog восьмиразрядный двоичный счетчик с параллельной загрузкой и синхронным сбросом. Блок будет прибавлять единицу к своему значению каждый такт. В случае синхронного сброса счетчик принимает свое значение равным нулю. При наличии сигнала загрузки, в счётчик загружается значение с информационных входов.

```
//-------------------------------------------------------
// Счётчик с параллельной загрузкой
//-------------------------------------------------------
module up_counter_load (
out     , // Output of the counter
data    , // Parallel load for the counter
load    , // Parallel load enable
enable  , // Enable counting
clk     , // clock input
reset     // reset input
);
//----------Output Ports--------------
output [7:0] out;
//------------Input Ports--------------
input [7:0] data;
input load, enable, clk, reset;
//------------Internal Variables--------
reg [7:0] out;
//-------------Code Starts Here-------
always @(posedge clk)
if (reset) begin
                    out <= 8'b0 ;
end else if (load) begin
                    out <= data;
end else if (enable) begin
                    out <= out + 1;
end
endmodule
```

Вариант 9

Описать на языке Verilog мультиплексор 4 в 1. Устройство имеет 4 сигнальных входа, один управляющий вход и один выход. Мультиплексор позволяет передавать сигнал с одного из входов на выход, при этом выбор желаемого входа осуществляется подачей соответствующей комбинации управляющих сигналов. УГО мультиплексора изображено ниже.



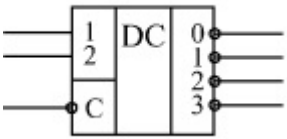

```
// ----------------------------------------------------------------
// Мультиплексор 4x1
// ----------------------------------------------------------------
// Truth table for 4x1 MUX
// S1  S0  | D
// --------+-------
//  0   0  | input0
//  0   1  | input1
//  1   0  | input2
//  1   1  | input3


module Mux (
   Input0,
   Input1,
   Input2,
   Input3,
   Sel,
   Data_out
   );

  input [3:0] Input0;
  input [3:0] Input1;
  input [3:0] Input2;
```

```
  input [3:0] Input3;
  input [1:0] Sel;
  output [3:0] Data_out;
  reg [3:0] Data_out;

  // constant declaration
  parameter S0 = 2'b00;
  parameter S1 = 2'b01;
  parameter S2 = 2'b10;
  parameter S3 = 2'b11;

  always @ (Sel or Input0 or Input1 or Input2 or Input3)
  begin
    case(Sel)
      S0: begin
        Data_out <= Input0;
      end
      S1: begin
        Data_out <= Input1;
      end
      S2: begin
        Data_out <= Input2;
      end
      S3: begin
        Data_out <= Input3;
      end
    endcase
  end
endmodule
```

Вариант 10

Описать на языке Verilog полный дешифратор из 2 в 4. Таблица истинности и УГО устройства приведены ниже.

Входные сигналы		Выходные сигналы			
X_1	X_0	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1




```
//-----------------------------------------------------
// Описание дешифратора
//-----------------------------------------------------
module decoder_using_case (
   binary_in   , //  2 bit binary input
   decoder_out , //  4-bit  out
   enable        //  Enable for the decoder
   );
input [1:0] binary_in  ;
input  enable ;
output [3:0] decoder_out ;

reg [3:0] decoder_out ;

always @ (enable or binary_in)
begin
  decoder_out = 0;
  if (enable) begin
    case (binary_in)
      2'h0 : decoder_out = 4'b0001;
      2'h1 : decoder_out = 4'b0010;
      2'h2 : decoder_out = 4'b0100;
      2'h3 : decoder_out = 4'b1000;
    endcase
  end
end
endmodule
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Основы цифровой обработки сигналов [Текст]: Курс лекций / А.И. Солонина, Д.А. Улахович, С.М. Арбузов и др. - СПб.: БХВ-Петербург, 2003.-232 с.
2. VHDL: справочное пособие по основам языка [Текст] / В. П. Бабак, А.Г. Корченко, Н.П. Тимошенко, С.Ф. Филоненко. – М.: ИД «Додэка – XXI», 2008. – 224 с.
3. Солонина, А.И. Алгоритмы и процессоры цифровой обработки сигналов [Текст] / А.И. Солонина, Д.А. Улахович. - СПб.: БХВ-Петербург, 2002. – 464 с.
4. Зотов В.Ю. Проектирование встраиваемых микропроцессорных систем на основе ПЛИС фирмы Xilinx [Текст] / В.Ю. Зотов. – М.: Горячая линия - телеком, 2006 – 520 с.
5. Суворова, Е. А. Проектирование цифровых систем на VHDL [Текст] / Е. А. Суворова. - СПб.: БХВ-Петербург, 2007.-478 с.
6. Суворова, Е. А. Проектирование цифровых систем на VHDL [Текст] / Е.А. Суворова, Ю.Е. Шейнин. – СПб.: БХВ-Петербург. 2003. – 576 с.
7. Бибило, П.Н. Основы языка VHDL [Текст] / П.Н. Бибило. – М.: Горячая линия - телеком, 2003 – 380 с.
8. Максфилд, К. Проектирование на ПЛИС: курс молодого бойца [Текст]: пер. с англ. / К. Максфилд. - М.: ИД «Додэка –XXI», 2007.– 408 с.
9. Молдовян, А.А. Криптография: скоростные шифры [Текст] / А.А. Молдовян, Н.А. Молдовян, Н.Д. Гуц, Б.В Изотов. – СПб.: БХВ-Петербург, 2002.-394 с.
10. Молдовян, Н.А. Криптография: от примитивов к синтезу алгоритмов [Текст] / Н.А. Молдовян, А.А. Молдовян, М.А. Еремеев. – СПб.: БХВ-Петербург, 2004.-534 с.
11. Смарт, Н. Мир программирования: криптография / Н. Смарт. – М.: Техносфера, 2005. - 310 с.

12. Малюк, А.А. Введение в защиту информации в автоматизированных система [Текст]: учеб. пособие / А.А. Малюк, С.В. Пазизин, Н.С. Погожин. - 2-е изд. – СПб.: БХВ – Петербург, 2004 – 356 с.

13. Основы криптографии [Текст]: учеб. пособие / А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин. - М.: Гелиос АРВ, 2002.- 453с.

14. Основы автоматизации проектирования, тестирования и управления жизненным циклом изделий [Текст]: учеб. пособие / В.Ф. Барабанов, А.Д. Поваляев, С.Л. Подвальный, С.В. Тюрин. – Воронеж: Научная книга, 2011. – 165 с.

15. Основы проектирования цифровых устройств на языках VHDL и Verilog [Текст]: учеб. пособие / В.Ф. Барабанов, С.Л. Подвальный, Н.И. Гребенникова, В.В. Сафронов. – Воронеж: ФГБОУ ВПО «ВГТУ», 2012. –216 с.

ОГЛАВЛЕНИЕ

Учебное издание

Барабанов Александр Владимирович

ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА ЯЗЫКАХ VHDL И VERILOG

В авторской редакции

Подписано в печать 19.03.2015.
Формат 60×84/16. Бумага для множительных аппаратов.
Усл. печ. л. 10,5. Уч.-изд. л. 8,8. Тираж 250 экз.
Заказ № 36.

ФГБОУ ВПО «Воронежский государственный
технический университет»
394026 Воронеж, Московский просп., 14