Министерство образования и науки Российской Федерации

Калужский филиал федерального государственного бюджетного

образовательного учреждения высшего образования

«Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)»

(КФ МГТУ им Н.Э. Баумана)

**В.Н. Коновалов, И.В. Коновалов**

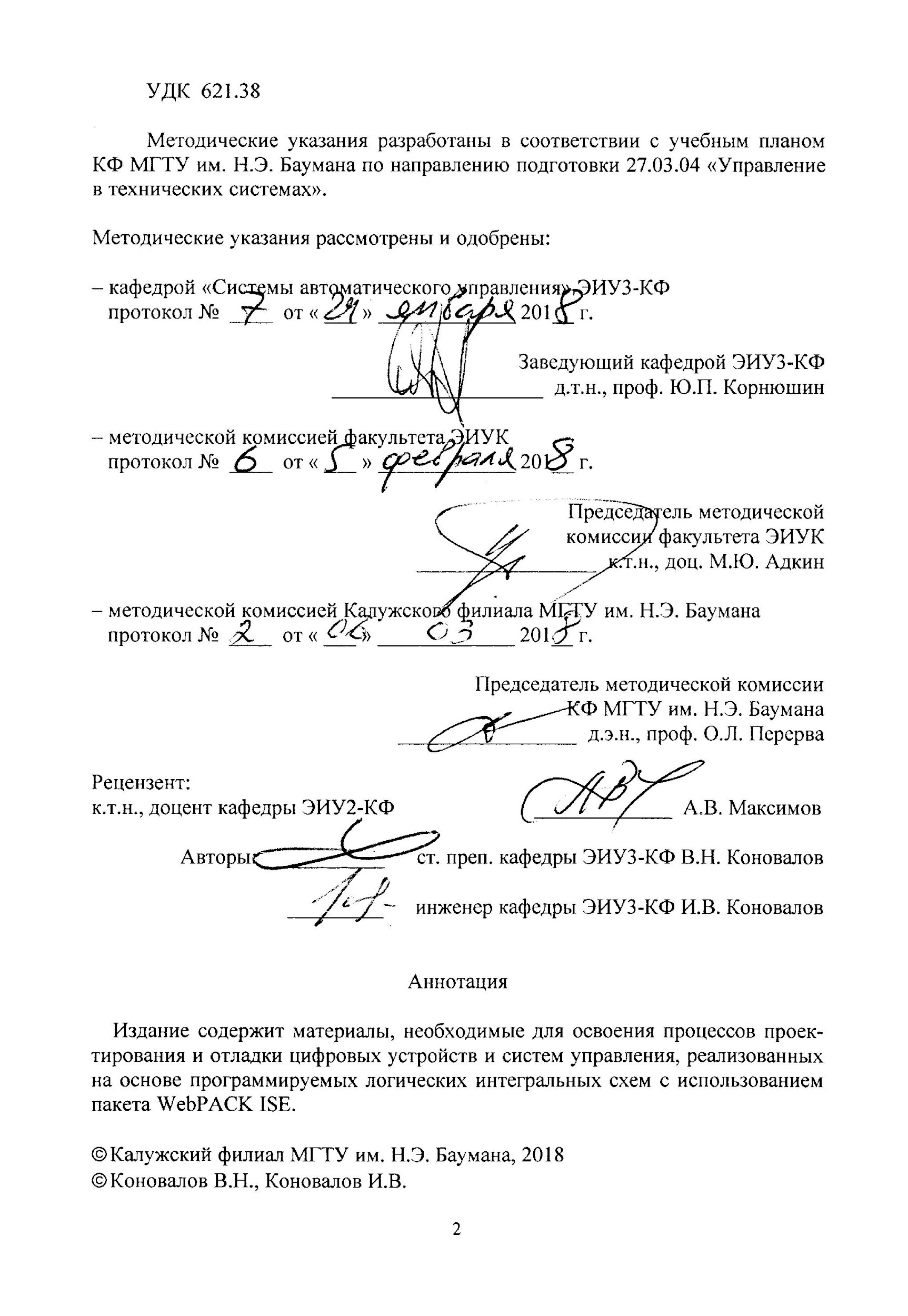
**ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ**

**НА ОСНОВЕ ПЛИС ФИРМЫ XILINX**

**Лабораторный практикум**

**по дисциплине «Конструирование приборов»**

Калуга, 2018

УДК 621.38

Методические указания разработаны в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 27.03.04 «Управление в технических системах».

Методические указания рассмотрены и одобрены:

– кафедрой «Системы автоматического управления» ЭИУ3-КФ

протокол № \_\_\_\_ от « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_\_ г.

Заведующий кафедрой ЭИУ3-КФ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ д.т.н., проф. Ю.П. Корнюшин

– методической комиссией факультета ЭИУК

протокол № \_\_\_\_ от « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_\_ г.

Председатель методической

комиссии факультета ЭИУК

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к.т.н., доц. М.Ю. Адкин

– методической комиссией Калужского филиала МГТУ им. Н.Э. Баумана

протокол № \_\_\_\_ от « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_\_ г.

Председатель методической комиссии

КФ МГТУ им. Н.Э. Баумана

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ д.э.н., проф. О.Л. Перерва

Рецензент:

к.т.н., доцент кафедры ЭИУ2-КФ \_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Максимов

Авторы: \_\_\_\_\_\_\_\_\_ ст. преп. кафедры ЭИУ3-КФ В.Н. Коновалов

\_\_\_\_\_\_\_\_\_ инженер кафедры ЭИУ3-КФ И.В. Коновалов

Аннотация

Издание содержит материалы, необходимые для освоения процессов проектирования и отладки цифровых устройств и систем управления, реализованных на основе программируемых логических интегральных схем с использованием пакета WebPACK ISE.

© Калужский филиал МГТУ им. Н.Э. Баумана, 2018

© Коновалов В.Н., Коновалов И.В.

**СОДЕРЖАНИЕ**

[ПРЕДИСЛОВИЕ 7](#_heading=h.gjdgxs)

[1. ЛАБОРАТОРНАЯ РАБОТА № 1. ЗНАКОМСТВО С ЛАБОРАТОРНЫМ СТЕНДОМ ЛСЦ-003 И ПОЛЬЗОВАТЕЛЬСКИМ ИНТЕРФЕЙСОМ ОСНОВНЫХ ПРОГРАММ ПАКЕТА WebPACK ISE 9](#_heading=h.30j0zll)

[Структурная схема стенда ЛСЦ-003 9](#_heading=h.kgcv8k)

[Конструкция стенда ЛСЦ-003 13](#_heading=h.3znysh7)

[Маршрут проектирования цифровых устройств на основе ПЛИС фирмы Xilinx 16](#_heading=h.2et92p0)

[Начало работы с пакетом WebPACK ISE 16](#_heading=h.tyjcwt)

[Создание нового проекта в САПР WebPACK ISE 18](#_heading=h.3dy6vkm)

[Создание принципиальной схемы в среде редактора ECS 22](#_heading=h.1t3h5sf)

[Ввод временных и топологических ограничений проекта 30](#_heading=h.4d34og8)

[Синтез проекта с использованием средств XST пакета WebPACK ISE 32](#_heading=h.2s8eyo1)

[Размещение и трассировка проектов, реализуемых на базе семейств ПЛИС CPLD фирмы Xilinx, в пакете WebPACK ISE 33](#_heading=h.17dp8vu)

[Программирование ПЛИС семейств CPLD с использованием модуля iMPACTпакета WebPACK ISE 35](#_heading=h.3rdcrjn)

[Задачи и порядок проведения работы 39](#_heading=h.26in1rg)

[Вопросы для самоконтроля и подготовки к защите лабораторной работы № 1 40](#_heading=h.lnxbz9)

[ЛАБОРАТОРНАЯ РАБОТА № 2. СОЗДАНИЕ ПРОЕКТА ЦИФРОВОГО nУСТРОЙСТВА НА ПРИМЕРЕ СИНТЕЗА СХЕМЫ ПОЛНОГО ДЕШИФРАТОРА СЕМИСЕГМЕНТНОГО СВЕТОДИОДНОГО ИНДИКАТОРА 41](#_heading=h.35nkun2)

[Синтез схемы дешифратора 41](#_heading=h.1ksv4uv)

[Задачи и порядок проведения работы 44](#_heading=h.44sinio)

[Вопросы для самоконтроля и подготовки к защите лабораторной работы № 2 45](#_heading=h.2jxsxqh)

[ЛАБОРАТОРНАЯ РАБОТА № 3. МОДЕЛИРОВАНИЕ СПРОЕКТИРОВАННОГО ЦИФРОВОГО УСТРОЙСТВА С ИСПОЛЬЗОВАНИЕМ ПАКЕТА MODELSIM 46](#_heading=h.z337ya)

[Этапы моделирования цифровых устройств, разрабатываемых на основе кристаллов ПЛИС семейств CPLD фирмы Xilinx 46](#_heading=h.3j2qqm3)

[Подготовка тестового модуля проекта 47](#_heading=h.1y810tw)

[Структура тестового модуля проекта 48](#_heading=h.4i7ojhp)

[Создание тестового модуля проекта в форме временных диаграмм 49](#_heading=h.2xcytpi)

[Установка значений параметров функционального моделирования проекта 58](#_heading=h.1ci93xb)

[Функциональное моделирование проекта в программе ModelSim 60](#_heading=h.3whwml4)

[Временное моделирование проекта, реализуемого на базе ПЛИС семейств CPLD 68](#_heading=h.2bn6wsx)

[Задачи и порядок проведения работы: 70](#_heading=h.qsh70q)

[Вопросы для самоконтроля и подготовки к защите лабораторной работы № 3 71](#_heading=h.3as4poj)

[ЛАБОРАТОРНАЯ РАБОТА № 4. РЕАЛИЗАЦИЯ ЦИФРОВЫХ УСТРОЙСТВ НА ПРОСТЫХ ЛОГИЧЕСКИХ ЭЛЕМЕНТАХ 72](#_heading=h.1pxezwc)

[Покрытие комбинационных схем двухвходовыми логическими элементами 72](#_heading=h.34g0dwd)

[Задачи и порядок проведения работы: 74](#_heading=h.2p2csry)

[Вопросы для самоконтроля и подготовки к защите лабораторной работы № 4 75](#_heading=h.147n2zr)

[ЛАБОРАТОРНАЯ РАБОТА № 5. ОПИСАНИЕ ЦИФРОВЫХ УСТРОЙСТВ В СРЕДЕ ПАКЕТА WEBPACK ISE НА ЯЗЫКЕ VHDL 77](#_heading=h.3o7alnk)

[Язык описания схем VHDL 77](#_heading=h.23ckvvd)

[Структура программы на языке VHDL 78](#_heading=h.ihv636)

[Описание интерфейса проектируемой схемы 80](#_heading=h.32hioqz)

[Архитектурные тела в языке VHDL 82](#_heading=h.1hmsyys)

[Операторы блоков в языке VHDL 85](#_heading=h.41mghml)

[Процессы в языке VHDL 86](#_heading=h.2grqrue)

[Типы, переменные и константы в языке VHDL 87](#_heading=h.vx1227)

[Атрибуты 95](#_heading=h.3fwokq0)

[Процедуры и функции на языке VHDL 96](#_heading=h.1v1yuxt)

[Библиотеки и пакеты на языке VHDL 97](#_heading=h.4f1mdlm)

[Операторы управления в языке VHDL 98](#_heading=h.2u6wntf)

[Задержки сигналов в языке VHDL 102](#_heading=h.19c6y18)

[Моделирование схемных соединений 103](#_heading=h.3tbugp1)

[Пример описания дешифратора для семисегментного индикатора на языке VHDL 105](#_heading=h.28h4qwu)

[Задачи и порядок проведения работы 109](#_heading=h.nmf14n)

[Вопросы для самоконтроля и подготовки к защите лабораторной работы № 5 110](#_heading=h.1jlao46)

[ЛАБОРАТОРНАЯ РАБОТА № 6. ИССЛЕДОВАНИЕ ОСОБЕННОСТЕЙ СХЕМОТЕХНИЧЕСКОЙ РЕАЛИЗАЦИИ И РАБОТЫ ТИПОВЫХ КОМБИНАЦИОННЫХ УСТРОЙСТВ 112](#_heading=h.1mrcu09)

[Двоичные дешифраторы 112](#_heading=h.46r0co2)

[Наращивание размерности дешифраторов 115](#_heading=h.2lwamvv)

[Мультиплексоры 117](#_heading=h.111kx3o)

[Наращивание размерности мультиплексоров 119](#_heading=h.3l18frh)

[Типовые комбинационные устройства, представленные в библиотеке символов пакета WebPACK ISE 122](#_heading=h.206ipza)

[Задачи и порядок проведения работы: 123](#_heading=h.4k668n3)

[Вопросы для самоконтроля и подготовки к защите лабораторной работы № 6 124](#_heading=h.2zbgiuw)

[ЛАБОРАТОРНАЯ РАБОТА № 7. ИСПОЛЬЗОВАНИЕ ПРОРАММНОГО ПАКЕТА DECOMPOSER ДЛЯ ПРОЕКТИРОВАНИЯ КОМБИНАЦИОННЫХ СХЕМ 125](#_heading=h.1egqt2p)

[Общие сведения о программном пакете Decomposer 125](#_heading=h.3ygebqi)

[Абстрактный синтез сумматора по модулю 16 126](#_heading=h.2dlolyb)

[Детализация схемы до уровня двухвходовых блоков на примере синтеза разряда двоичного сумматора 138](#_heading=h.sqyw64)

[Задачи и порядок проведения работы 145](#_heading=h.3cqmetx)

[Вопросы для самоконтроля и подготовки к защите лабораторной работы № 7 147](#_heading=h.1rvwp1q)

[ОБЩИЕ ПРАВИЛА БЕЗОПАСНОСТИ ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНЫХ РАБОТ 148](#_heading=h.4bvk7pj)

[ЛИТЕРАТУРА 149](#_heading=h.2r0uhxc)

[**ПРЕДИСЛОВИЕ**](http://www.xilinx.com)

Учебный план дисциплины «Конструирование приборов» включает выполнение цикла лабораторных работ, связанных с проектированием, реализацией и отладкой цифровых устройств на основе программируемых логических интегральных схем ([ПЛИС](https://ru.wikipedia.org/wiki/%D0%9F%D0%9B%D0%98%D0%A1)) фирмы Xilinx.

В ходе лабораторных работ студенты осуществляют практическую реализацию и испытание разработанных схем на лабораторных стендах ЛСЦ-003, выполненных на основе программируемой логической интегральной схемы xc95288xl семейства CPLD фирмы [Xilinx](http://www.xilinx.com).

Микросхемы программируемой логики в настоящее время рассматриваются как наиболее перспективная элементная база для построения цифровой аппаратуры разнообразного назначения. Перспективность ПЛИС базируется на ряде их достоинств, к числу которых можно отнести следующие:

– универсальность и связанный с нею высокий спрос со стороны потребителей, что обеспечивает массовое производство;

– низкая стоимость, обусловленная массовым производством и высоким процентом выхода годных микросхем;

– высокое быстродействие и надежность как следствие реализации на базе передовых технологий;

– разнообразие конструктивного исполнения, поскольку обычно одни и те же кристаллы поставляются в разных корпусах;

– разнообразие в выборе напряжений питания и параметров сигналов ввода/вывода, а также режимов снижения мощности, что особенно важно для портативной аппаратуры с автономным питанием;

– наличие разнообразных, хорошо развитых и эффективных программных средств автоматизированного проектирования, малое время проектирования и отладки проектов;

– простота модификации проектов на любых стадиях их разработки.

Схемы комбинационных устройств, синтезированные студентами в ходе выполнения домашних работ, предусмотренных учебным планом, в дальнейшем используются в качестве исходного материала при выполнении соответствующих лабораторных работ:

«Реализация цифровых устройств на простых логических элементах» и

«Исследование особенностей схемотехнической реализации и работы типовых комбинационных устройств».

**Целями лабораторного практикума являются:**

– знакомство с современными методами логического синтеза комбинационных устройств на основе методов многоуровневой декомпозиции;

– знакомство с современными средствами САПР в области цифровой электроники;

– закрепление практических навыков логического синтеза и анализа отдельных функциональных узлов комбинационного типа, построенных на основе логических элементов заданного базиса;

**После выполнения лабораторного практикума студенты смогут:**

– выполнять логический синтез и анализ цифровых комбинационных устройств с использованием методов декомпозиции без привязки схемы к логическому базису;

– строить схемы цифровых устройств с использованием схемотехнического редактора ECS пакета WebPACK ISE;

– проводить программирование спроектированного устройства в ПЛИС xc95288xl семейства CPLD с использованием модуля iMPACT;

– самостоятельно проводить практические испытания цифровых устройств и выполнять оценку их работоспособности по результатам анализа схемы и проведенных испытаний.

**1. ЛАБОРАТОРНАЯ РАБОТА № 1. ЗНАКОМСТВО С ЛАБОРАТОРНЫМ СТЕНДОМ ЛСЦ-003 И ПОЛЬЗОВАТЕЛЬСКИМ ИНТЕРФЕЙСОМ ОСНОВНЫХ ПРОГРАММ ПАКЕТА WebPACK ISE**

**Структурная схема стенда ЛСЦ-003**

Структурная схема лабораторного стенда ЛСЦ-003 представлена на рис.1.



Рис. 1. Структурная схема стенда ЛСЦ-003

Основным элементом конструкции стенда, непосредственно предназначенным для физической реализации цифровых схем, является программируемая логическая интегральная схема (ПЛИС) фирмы Xilinx xc95288xl. Указанная ПЛИС построена на основе архитектуры CPLD (Complex Programmable Logic Device), содержит в своей структуре 288 макроячеек (6 400 эквивалентных логических вентилей) и относится к семейству xc9500xl, являющемуся результатом дальнейшего развития микросхем xc9500 на основе применения технологии 0,35 мкм. Основными особенностями семейства xc9500xl являются:

– снижение напряжения питания «ядра» кристалла до 3.3 В;

– повышение производительности (возможность реализации проектов с системными частотами до 208 МГц);

– расширение возможностей функциональных блоков;

– применение усовершенствованной матрицы переключений Fast CONNECT;

– наличие схем удержания последнего состояния в блоках ввода/вывода;

– применение усовершенствованных технологий защиты конфигурационных данных от несанкционированного копирования и случайного стирания;

– сравнительно мощные выходные буферы, обеспечивающие максимальный ток до 24 мА;

– совместимость с 3,3 В и 5 В логикой по входу и выходу, позволяющая использовать ПЛИС в схемах со смешанным питанием;

– минимальная задержка распространения сигнала от входного контакта до выходного через комбинационную логику – 5 нс;

– возможность раздельного управления длительностью фронтов выходных сигналов для каждого вывода ПЛИС, позволяющая снизить уровень помех на выходах кристалла;

– возможности фиксации пользовательских выводов перед трассировкой;

– возможность установки программируемого режима пониженной потребляемой мощности для каждой макроячейки;

– не менее 10 000 циклов перепрограммирования;

гарантированный срок хранения запрограммированной

– конфигурации не менее 20 лет;

полная поддержка протокола периферийного сканирования в соответствии со стандартом IEEE 1149.1 (JTAG);

– снижение стоимости кристаллов.

Описанные выше особенности микросхем xc95288xl позволяют реализовывать на их основе широкий спектр цифровых устройств и систем без использования дополнительных элементов (например, внешних ПЗУ для хранения конфигурационных последовательностей).

Для подачи на исследуемую цифровую схему входных воздействий в стенд включены блоки формирования входных сигналов БФВС1, БФВС2 и БФВС3, позволяющие генерировать статические (A, B, C, D, T, Q) и динамические (импульсные) (C1, C2, SET, RESET) логические сигналы. Уровни указанных сигналов можно наблюдать визуально на дискретных светодиодных индикаторах.

Кроме того блоки БФВС позволяют формировать сетку опорных тактовых сигналов (ТИ1, ТИ2, ТИ3) с частотами 1 МГц, 1 КГц и 1 Гц, которые в случае необходимости могут использоваться для синхронизации и тактирования исследуемых цифровых схем.

Блок БФВС2 может работать в режиме аналогово-цифрового преобразователя (АЦП), при этом на его выходах формируется цифровой код, соответствующий величине входного напряжения VIN, поступающего с блока сопряжения с внешними устройствами (БСВУ).

Блок БФВС3 имеет входной разъем ГИ для подключения внешнего генератора с целью подачи на исследуемую схему импульсных сигналов TST.

Блок сопряжения с внешними устройствами (БСВУ) позволяет подключить к стенду ЛСЦ-003 дополнительные элементы (двигатели постоянного тока, вентильные и шаговые двигатели, электромеханические исполнительные устройства, различные датчики и др.), которые могут потребоваться для моделирования и реализации сложных систем автоматического управления. Для управления мощными исполнительными устройствами необходимо использовать внешние усилительные и ключевые элементы.

Блок индикации БИ7-4 представляет собой два сдвоенных семисегментных индикатора со статическим управлением, которые позволяют наблюдать состояния на выходах исследуемых схем в виде изображения шестнадцатеричных или десятичных цифр. При этом дешифратор для управления индикаторами должен быть реализован средствами ПЛИС.

Блок индикации и контроля БИК представляет собой 16 дискретных светодиодов, которые позволяют визуально наблюдать состояния на соответствующих выводах ПЛИС. Кроме того, в состав блока БИК входят два разъема CTL1 и CTL2, к которым может быть подключен двухлучевой осциллограф OSC для наблюдения динамических (импульсных) сигналов. Для подключения выходов CTL1 и CTL2 к различным внутренним точкам исследуемых схем можно использовать мультиплексоры, входящие в структуру ПЛИС. Такой подход позволяет контролировать статические и динамические сигналы на выходах сложных цифровых устройств и систем.

Блок интерфейса USB БИ USB построен на основе специализированной микросхемы FT245BM, преобразующей сигналы шины USB в восьмиразрядный параллельный интерфейс, представленный однобайтным портом шины данных и сигналами управления и синхронизации. Выводы параллельного порта и управляющие сигналы подключены к выводам ПЛИС.

Блок интерфейса LPT БИ LPT обеспечивает сопряжение персонального компьютера (ПК) со стендом ЛСЦ-003. При этом все сигналы стандартного LPT-порта ПК подаются на выводы ПЛИС.

Указанные интерфейсные блоки обеспечивают возможность передачи данных между ПК и реализованным на ПЛИС устройством с целью проведения его автоматизированного тестирования или реализации автоматических цифровых систем с ЭВМ в контуре управления.

Блок программирования БПР построен на основе буферных элементов sn74ls125 и может совместно с блоком БИ LPT работать в одном из двух задаваемых пользователем режимов:

1) «ПРОГРАММИРОВАНИЕ». В этом режиме обеспечивается передача конфигурационных файлов из ПК в ПЛИС с использованием сигналов TMS, TCK, TDI и TDO интерфейса JTAG. Такие файлы формируются на этапе проектирования цифровых устройств специализированными средствами САПР (например, пакетом WebPACK ISE для ПЛИС фирмы Xilinx).

2) «ИНТЕРФЕЙС». В этом режиме обеспечивается непосредственное подключение сигналов LPT-порта к выводам ПЛИС для сопряжения исследуемых цифровых устройств с ПК.

Блок питания БП реализован на микросхемах LM1117 и служит для получения стабилизированных напряжений +5 В (VCC5A и VCC5B) и +3,3 В (VCC3.3) для питания стенда. Входное напряжение VPOW подается от внешнего нестабилизированного источника постоянного тока, имеющего на выходе напряжение 7 … 8 В при токе нагрузки 0,8 … 1 А.

Описанная выше структура лабораторного стенда ЛСЦ-003 позволяет решать учебные задачи различного содержания и степени сложности, связанные с моделированием отладкой и демонстрацией широкого спектра цифровых устройств и систем управления.

**Конструкция стенда ЛСЦ-003**

Конструктивно стенд ЛСЦ-003 выполнен в виде двухсторонней печатной платы размером 200 х 300 мм. Схема расположения основных элементов представлена на рис. 2.



Рис. 2. Схема расположения основных элементов стенда ЛСЦ-003

В левой части платы расположены тактовые кнопки, посредством нажатия на которые формируются 16 статических входных сигналов, разделенных на четыре группы – A, B, C и D. Состояния указанных сигналов индицируются расположенными рядом с каждой кнопкой светодиодными индикаторами. Каждое нажатие на тактовую кнопку изменяет состояние соответствующего входного сигнала на противоположное. Опрос состояния тактовых кнопок и формирование входных сигналов ПЛИС и сигналов управления светодиодами производится с помощью трех микроконтроллеров ATmega48, работающих по специальным программам, загруженным в их память. Такое конструктивное решение позволило избавиться от ненадежных и дорогих двухпозиционных переключателей, обычно используемых в подобных случаях. Кроме того, на плате нанесены номера выводов ПЛИС (*PXXX*), к которым подключены соответствующие элементы схемы, что значительно облегчает пользователям ввод топологических ограничений при создании проектов.

В верхней части платы расположены соединительные разъемы:

– для подключения LPT-порта ПК (DRB-25MA) с целью программирования ПЛИС и обмена данными со стендом,

– для сопряжения с внешними устройствами (DRB-15MA),

– для соединения с портом USB ПК,

– для подключения внешнего блока питания стенда.

Кроме того, в центре верхней части платы находится тактовая кнопка и два индикаторных светодиода (LPT и PROG), управляющие режимом работы программатора. При выборе одного из режимов зажигается соответствующий светодиод. Справа от указанных элементов на плате расположены микросхемы интерфейсного блока USB – FT245 и 93C46.

В правой части платы расположены четыре сдвоенных семисегментных индикатора со статическим управлением (IND1 … IND4) и 16 дискретных светодиодов, разделенных на четыре группы – E, F, G и H, которые позволяют наблюдать состояния на выходах исследуемых схем.

В нижней части платы расположены тактовые кнопки, посредством нажатия на которые формируются четыре импульсных сигнала (CLK1, CLK2, SET, RESET) и 6 статических входных сигналов, разделенных на две группы – T и Q. Группы сигналов T и Q могут использоваться для управления мультиплексорами, позволяющими подключать к расположенным здесь же разъемам CTL1 и CTL2 (типа BNC) различные внутренние точки исследуемых схем с целью визуального наблюдения динамических (импульсных) сигналов на экране двухлучевого осциллографа.

В центре нижней части платы находится разъем TST (типа BNC) для подключения к стенду импульсного генератора. В левом нижнем углу платы расположена кнопка ADC (и соответствующий индикаторный светодиод), посредством нажатия на которую блок БФВС2 переводится в режим аналогово-цифрового преобразователя (АЦП), при этом на его выходах вместо логических сигналов C и D формируется восьмиразрядный цифровой код, соответствующий величине входного напряжения VIN, поступающего с блока сопряжения с внешними устройствами (БСВУ).

В центре платы расположен основной конструктивный элемент стенда, непосредственно предназначенный для физической реализации цифровых схем – ПЛИС xc95288xl. Кроме того, в центральной части платы находятся три микросхемы стабилизаторов напряжения LM1117, каждая из которых припаивается к специальной медной подложке-радиатору для отвода выделяющегося при работе тепла.

Такая конструкция стенда позволяет выполнить все соединения в виде печатных проводников на плате и избавиться от проводных и кабельных линий, что в конечном итоге позволит автоматизировать сборочно-монтажные операции при производстве.

**Маршрут проектирования цифровых устройств на основе ПЛИС фирмы Xilinx**

Для создания цифрового устройства на базе ПЛИС Xilinx необходимо выполнить следующую последовательность операций:

1. Создать новый проект, указав семейство, тип ПЛИС и средств синтеза;

2. Разработать описание проектируемого устройства в схемотехнической, алгоритмической или текстовой форме;

3. Выполнить синтез устройства;

4. Провести проверку проекта методом функционального моделирования;

5. Выполнить размещение и трассировку проекта в кристалл;

6. Провести окончательную верификацию проекта методом временного моделирования;

7. Загрузить конфигурационные данные проекта в кристалл (выполнить программирование ПЛИС);

Операции функционального и временного моделирования не являются обязательными, но позволяют значительно сократить общее время разработки устройства за счет раннего обнаружения возможных ошибок. Ниже рассматривается выполнение необходимых операций проектирования при использовании схемотехнического описания проекта.

**Начало работы с пакетом WebPACK ISE**

Для начала работы с пакетом WebPACK ISE необходимо запустить *Навигатора проекта (Project Navigator),* основоне окно которого показано на рис. 3. Оно содержит кроме стандартных элементов четыре встроенных окна:

– окно исходных модулей проекта (*Sources in Project*);

– окно необходимых процессов для выбранного исходного модуля проекта (*Processes for Current Source*);

– окно консольных сообщений программных модулей (*Console*);

– окно редактора текстовых HDL-описаний проекта.

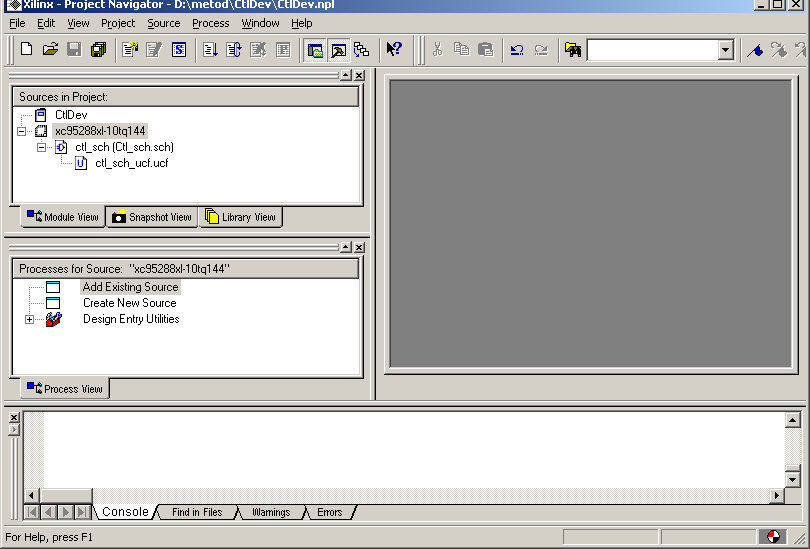


Рис. 3. Основное окно *Навигатора проекта* пакета WebPACK ISE

В окне исходных модулей проекта отображается иерархическая структура, состоящая из модулей, в которых содержится описание проектируемого устройства и описание тестовых воздействий, используемых в процессе моделирования. Каждый тип модуля имеет соответствующее графическое обозначение – пиктограмму.

Окно процессов показывает маршрут обработки выделенного исходного модуля в процессе проектирования устройства. Таким образом, в данном окне подробно отображаются все этапы процесса разработки и программирования ПЛИС.

Последовательность и содержание этапов определяется типом исходного модуля и семейством ПЛИС. Каждый этап может быть активизирован непосредственно в окне процессов. Для этого достаточно дважды щелкнуть левой кнопкой мыши на строке с названием соответствующей процедуры в окне процессов. При активизации процесса также выполняются процедуры, которые предшествуют и оказывают влияние на исполняемый процесс. После выполнения процедуры в строке с ее названием выводится пиктограмма, обозначающая характер завершения процесса.

Возможны следующие варианты завершения процедур:

 – текущий процесс выполнен успешно;

 – процедура исполнена без ошибок, но имеются предупреждения;

 – при выполнении процесса обнаружены ошибки.

Более подробные сведения о результатах выполнения процесса содержатся в соответствующем отчете. Для просмотра отчета следует дважды щелкнуть левой кнопкой мыши на строке с названием отчета.

Окно консольных сообщений предназначено для вывода информации программных модулей пакета, работающих в консольном режиме. Здесь отображаются сообщения об ошибках и предупреждения.

Окно интегрированного текстового редактора становится активным, если для проектируемого устройства или используемых библиотек выбран способ описания на языке HDL.

**Создание нового проекта в САПР WebPACK ISE**

Для создания нового проекта нажмите *File>New Project*, как показано на рис. 4.

|  |  |
| --- | --- |
| а) | б) |

Рис. 4. Создание нового проекта в САПР WebPACK ISE

В открывшейся диалоговой панели нужно определить исходные данные, необходимые для создания проекта:

– название проекта (*Project Name*);

– путь, в котором предполагается расположить проект (*Project Location*);

– семейство ПЛИС, на базе которого разрабатывается устройство (*Device Family*) (XC9500XL CPLDs);

– тип кристалла (*Device*) (xc95288xl);

– корпус (*Pacage*) (TQ144);

– быстродействие (*Speed Grade*) (-10);

– средство синтеза (*Design Flow*) (XST VHDL).

Значение EDIF используется в случае, если проект представлен в виде списка цепей в формате EDIF, например, при импорте из других систем проектирования.

В случае успешного создания нового проекта его название отображается в строке заголовка окна *Навигатора проекта*, а окно исходных модулей приобретает вид, показанный на рис. 5.

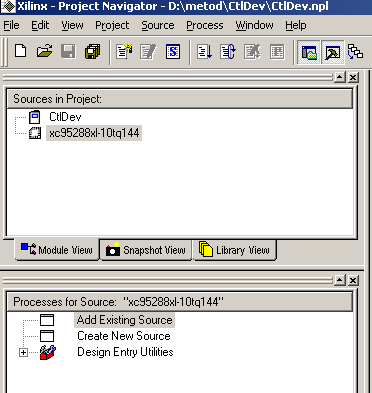


Рис. 5. Рабочая область нового проекта

Окно исходных модулей нового проекта содержит две строки: заголовка и описания проекта.

Далее необходимо создать модули исходного описания проектируемого устройства. Ниже рассматривается схемотехнический способ описания проектируемого цифрового устройства.

**Создание нового модуля исходного описания проекта в пакете WebPACK ISE**

Для создания нового модуля исходного описания проекта следует выбрать команду *New Source* из раздела *Project* основного меню. В открывшейся диалоговой панели, показанной на рис. 6.

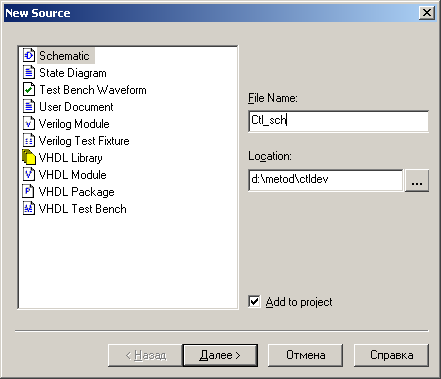


Рис. 6. Диалоговая панель установки параметров

нового исходного модуля описания проекта

Необходимо выбрать тип нового модуля, задать его имя и указать место расположения файла на диске. Для разработки принципиальной схемы следует выбрать тип создаваемого исходного модуля *Schematic*. Место расположения создаваемого модуля на диске указывается в поле редактирования *Location* диалоговой панели. По умолчанию предлагается рабочий каталог текущего проекта. Если флаг индикатора *Add to project* находится в установленном состоянии, то созданный модуль автоматически включается в состав текущего проекта.

При успешном создании модуля открывается информационная панель параметров создаваемого исходного модуля проекта .На этой панели отображается информация обо всех параметрах созданного модуля. После создания модуля автоматически производится запуск программы пакета, которая используется для работы с соответствующим типом исходного описания проектируемого устройства. В рассматриваемом случае открывается окно схемотехнического редактора *Shematic Editor* ECS.

**Создание принципиальной схемы в среде редактора ECS**

В процессе создания принципиальной схемы проектируемого устройства необходимо выполнить следующие операции:

1. Ввод символов элементов схемы;

2. Соединение компонентов схемы с помощью проводников и шин;

3. Ввод названий цепей и шин;

4. Установка маркеров цепей, используемых для подключения «внешних» элементов или выводов ПЛИС.

При открытии окна схемотехнического редактора (рис. 7) активизирован режим выбора объекта, установленный по умолчанию. В этом режиме осуществляется выделение, перемещение и удаление элементов схемы, а также просмотр и редактирование их параметров.

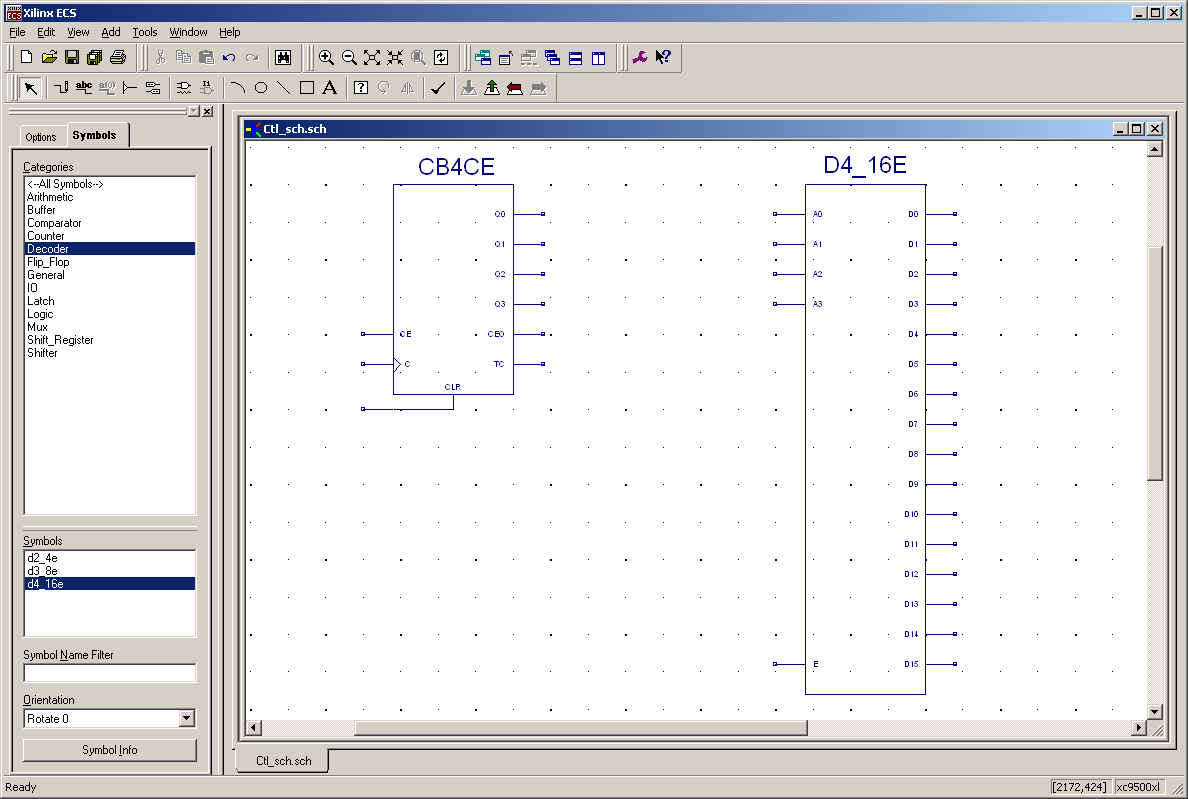


Рис. 7. Расположение символов компонентов на поле чертежа

Для ввода символов компонентов создаваемой схемы следует нажать кнопку  на инструментальной панели или выбрать команду *Symbol*, которая находится во всплывающем меню *Add*. Режим ввода символов компонентов также автоматически включается при выборе символа в панели библиотек, которая находится в правой части окна ECS (рис. 7). Вначале следует выбрать нужную функциональную группу символов библиотеки компонентов в поле *Categorie*s. Далее в поле *Symbols* выбирается искомый компонент, после чего курсор мыши следует переместить на поле чертежа. При этом к курсору мыши привязывается контурное изображение выбранного компонента, которое перемещается вместе с указателем. Для получения повернутого или зеркального изображения символа на поле чертежа следует нажать кнопку активизации выпадающего списка возможных типов ориентации в правой части поля выбора *Orientation* и выбрать требуемый угол поворота и форму представления УГО щелчком левой кнопки мыши на соответствующей строке списка. После этого контурное изображение символа, привязанное к курсору, примет выбранную ориентацию. Далее следует поместить указатель с изображением элемента в соответствующее место страницы схемы и зафиксировать его, щелкнув левой кнопкой мыши.

В случае успешного выполнения указанных операций на поле чертежа появляется детальное изображение выбранного компонента (рис. 7). После ввода символа курсор по-прежнему сохраняет контурное изображение элемента, поэтому, если в схеме используется несколько экземпляров текущего выбранного компонента, то нужно поместить указатель на место предполагаемого расположения следующей копии УГО и вновь щелкнуть левой кнопкой мыши. Аналогичным образом размещаются на поле чертежа все компоненты схемы.

Для соединения компонентов схемы с помощью проводников (Wire) и шин (Bus) следует нажать кнопку  на инструментальной панели или выбрать команду *Wire* из всплывающего меню *Add.* Формирование цепи начинается с фиксации стартовой точки, которая может располагаться на свободном месте поля чертежа или совпадать с выводом одного из компонентов. Для этого следует поместить курсор мыши в требуемую точку на поле чертежа и щелкнуть левой кнопкой мыши. Затем следует переместить указатель в позицию, соответствующую точке изгиба, соединения с другим проводником или контактом компонента, а также конечной точке цепи. Фиксация сегмента осуществляется щелчком левой кнопкой мыши в конечной точке, после чего новый фрагмент цепи отображается основной линией. Завершение формирования цепи осуществляется щелчком правой кнопкой мыши после фиксации последнего сегмента цепи.

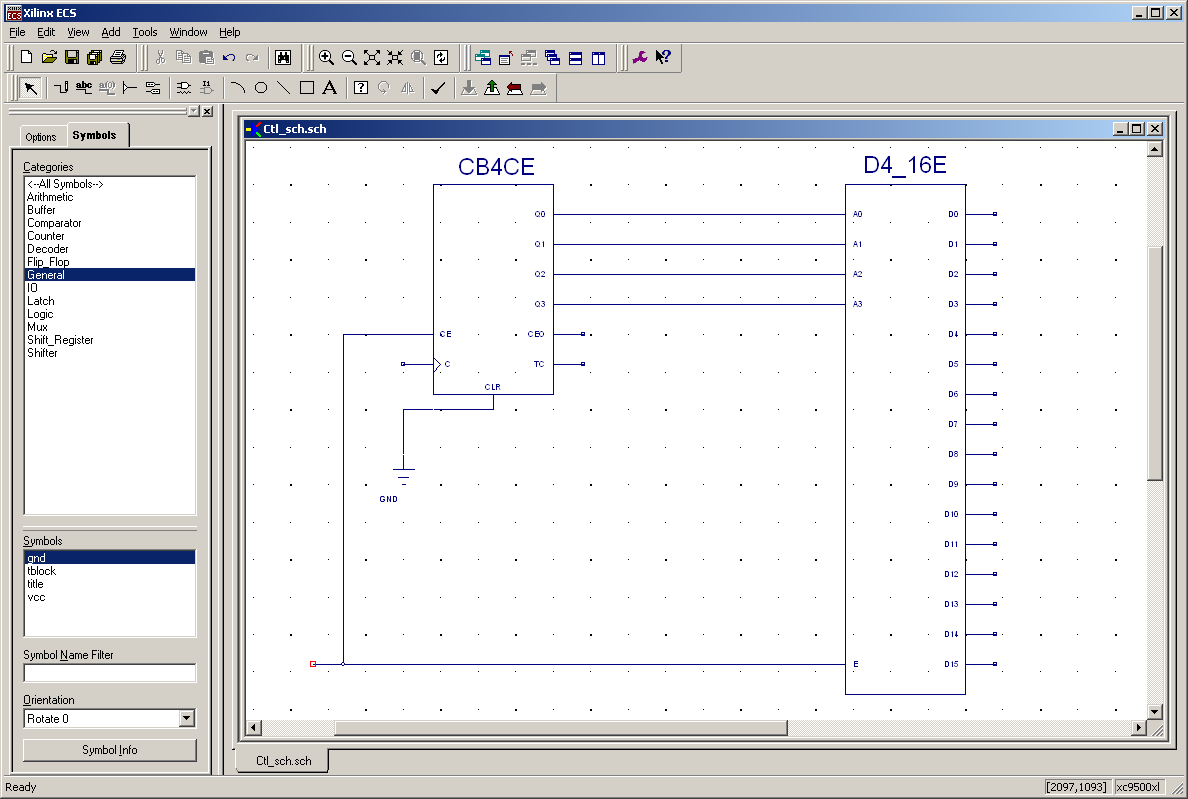


Рис. 8. Соединение компонентов схемы

с помощью проводников (цепей)

Для формирования соединений элементов схемы в виде шин необходимо выполнить следующую последовательность действий. Вначале в режиме ввода проводников создается графическое изображение шины в виде фрагмента одиночной цепи в соответствии с инструкциями, рассмотренными выше. До тех пор, пока не задано название шины в соответствующем формате (с указанием разрядности или перечислением проводников), она отображается сплошной тонкой линией, как одиночная цепь. Затем следует перейти в режим формирования отводов шины, нажав кнопку  на панели инструментов или выполнив команду *Bus Tap* из выпадающего меню *Add.* При этом к курсору присоединяется изображение отвода шины (рис. 9). Символ отвода шины может быть подключен к вертикальному или горизонтальному сегменту шины. Для получения нужной ориентации изображения отвода шины следует воспользоваться кнопкой , при каждом нажатии которой осуществляется поворот изображения на девяносто градусов по часовой стрелке.

Далее нужно указать точку подключения проводника к шине. При этом символ отвода присоединяется к изображению выбранной шины (рис. 9). После формирования всех отводов шины производится их соединение проводниками с соответствующими цепями и элементами схемы. После подключения проводников к отводам шины производится присвоение соответствующих названий этих цепей.

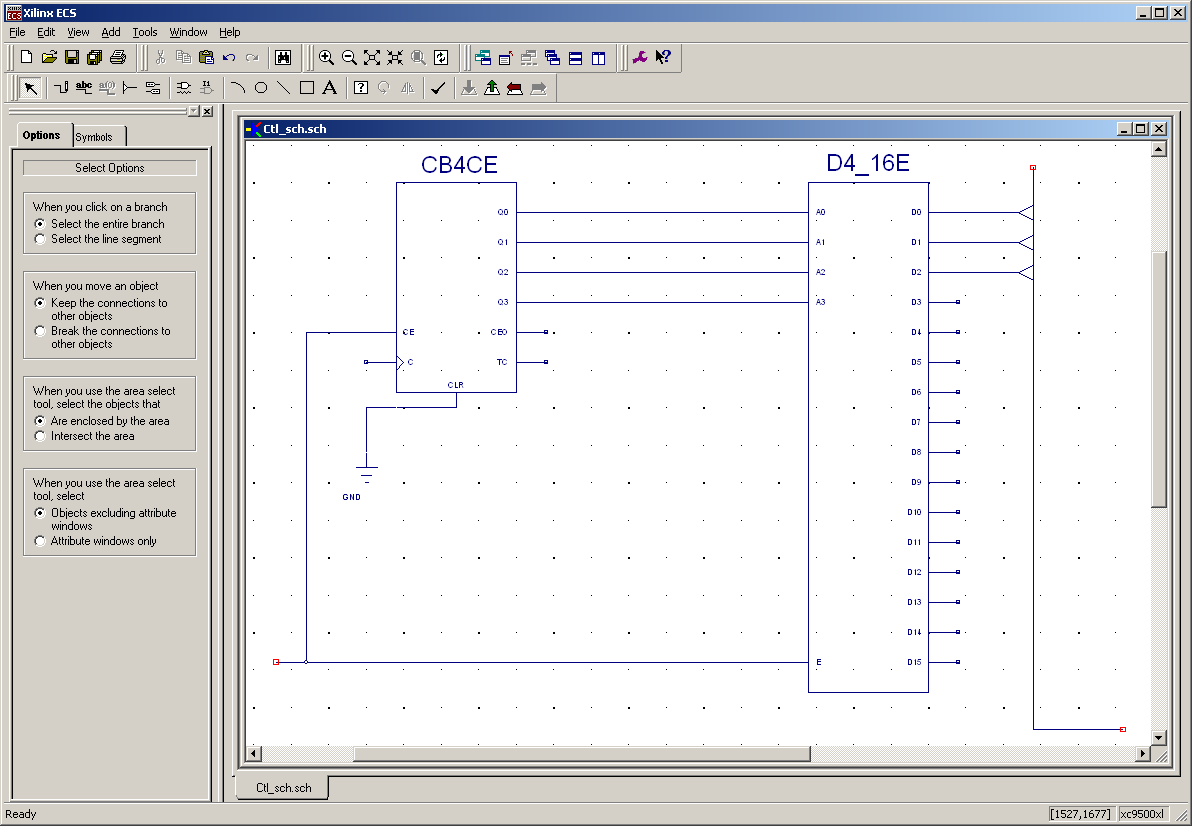


Рис. 9. Формирование соединений схемы в виде шин

При разработке схемы в редакторе ECS необходимо установить названия цепей, которые входят в состав шин или используются для соединений с «внешними» элементами. В состав названия цепи могут входить прописные и строчные буквы латинского алфавита (A – Z, a – z), цифры (0 – 9), а также символ подчеркивания («\_»). ***Название должно начинаться с буквы или цифры и не может состоять только из цифр.*** Длина названия не должна превышать 255 символов. Чтобы включить режим ввода названия цепей, следует нажать кнопку  на панели инструментов или выбрать пункт *Add*в основном меню, а затем в соответствующем всплывающем меню - строку *Net Name.* В этом режиме панель дополнительных параметров схемотехнического редактора имеет вид, показанный на рис. 10.

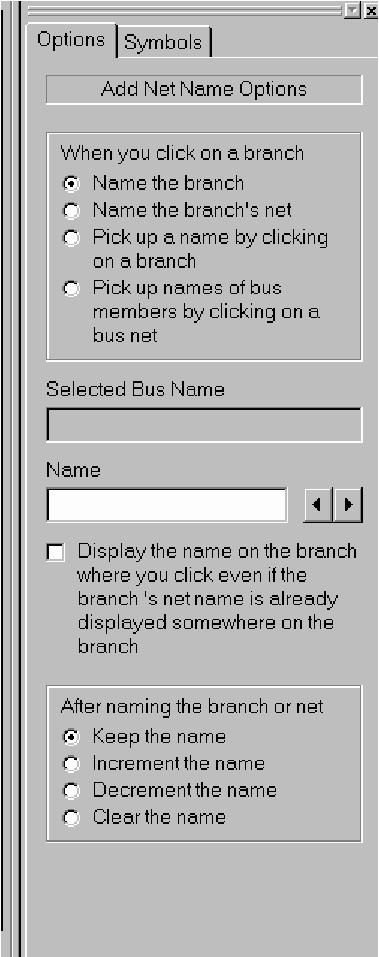


Рис.10. Вид панели дополнительных параметров

в режиме ввода названий цепей

Название цепи набирается в поле *Name* с помощью клавиатуры. Введенный текст названия привязывается к курсору мыши при перемещении последнего на поле чертежа схемы. Далее необходимо поместить указатель на изображение соответствующей цепи и щелкнуть левой кнопкой мыши. При отсутствии ошибок название цепи отображается рядом с ее изображением, в позиции, зафиксированной при щелчке левой клавиши мыши (рис. 11). ***Если на схеме присутствуют два или более проводника с одинаковыми названиями, но не имеющие точек соединений, то они объединяются электрически в одну цепь.***

|  |  |
| --- | --- |
| а) | б) |

Рис. 11. Ввод названий цепей и шин

Названия шин устанавливаются аналогично в том же режиме. Формат названий шин имеет следующий вид:

<имя шины>(<конечный номер проводника в шине>:<начальный номер проводника в шине>),

например, OUT\_Q(15:0).

Названия проводников шины указываются в формате:

<название шины>(<номер проводника в шине>),

например, OUT\_Q (5).

***В скобках вместо задания конечного и начального номеров можно через запятую перечислить названия проводников шины.***

Для автоматизированного именования цепей, соединённых с шиной, в панели дополнительных параметров следует выбрать *Pick ubnames of bus members by ckicking on a bus net,* затем выбрать курсором соответствующую шину. В поле *Selected bus name* появится имя выбранной шины, а в поле *Name* имена цепей шины, которые можно менять клавишами .

Для включения режима ввода маркеров цепей (выводов микросхемы или УГО) следует нажать кнопку  на панели инструментов или выполнить команду *I/O Marker* из вcпадающего меню *Add,* после чего к курсору привязывается изображение маркера (рис. 12). Его тип (входной *Input*, выходной *Output* или двунаправленный *Bidirectional*) указывается на панели дополнительных параметров. ***Маркеры можно присоединить только к проводникам или шинам. К выводам УГО элементов их присоединить нельзя.***

|  |  |
| --- | --- |
| а) | б) |

Рис. 12. Установка маркеров цепей

Установка маркеров шин производится аналогично.

Для проверки разработанной схемы следует воспользоваться командой *Check Schematic*. В процессе верификации осуществляется контроль целостности схемы и выполнения правил электрических соединений. После выполнения проверки открывается окно отчета, в котором отображаются сообщения о возможных ошибках и предупреждения с указанием цепи или компонента, с которым они связаны.

***По окончании работы необходимо обязательно сохранить разработанную схему.***

При рассмотрении дальнейших этапов разработки цифрового устройства на базе ПЛИС фирмы Xilinx, в качестве примера рассматривается проект, принципиальная схема которого показана на рис. 13.

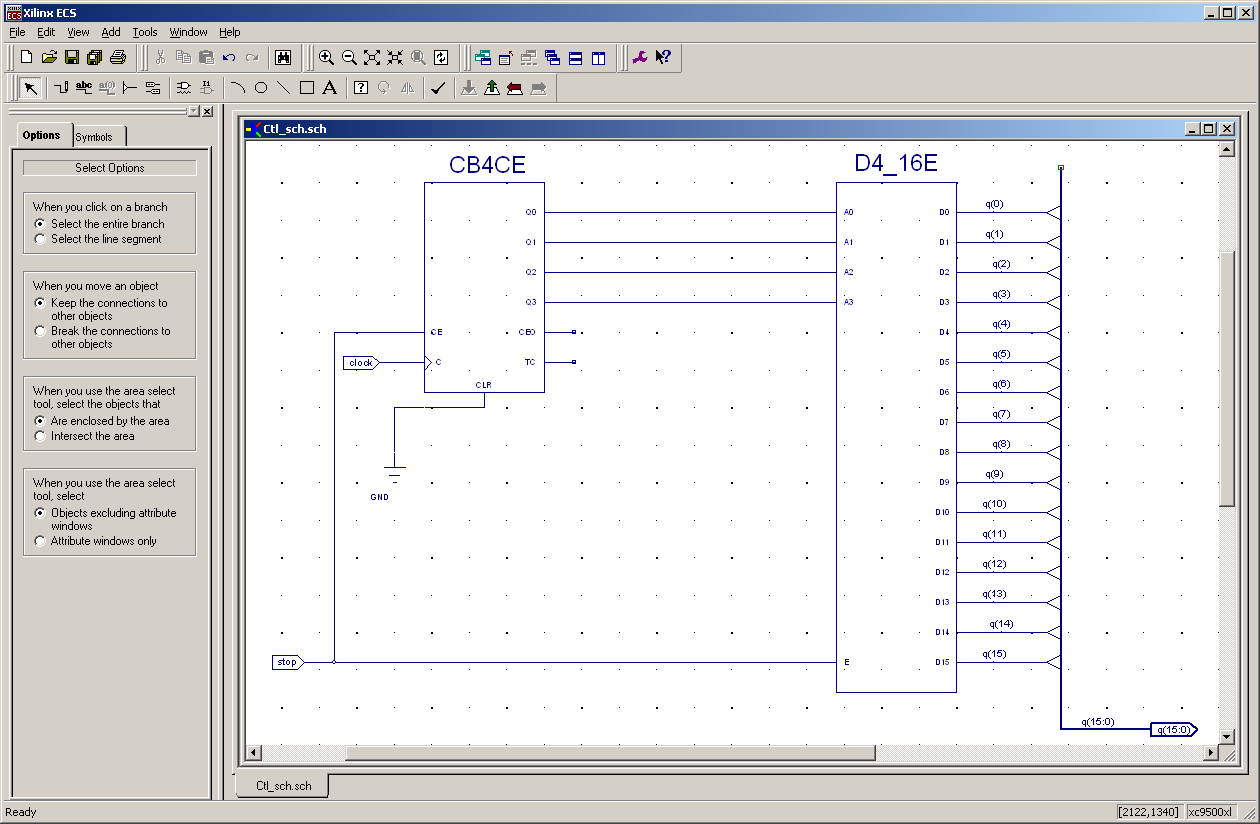


Рис. 13. Принципиальная схема устройства циклического

последовательного выбора, выполненная в редакторе ECS

**Ввод временных и топологических ограничений проекта**

Для ввода дополнительной информации, используемой программами синтеза, размещения и трассировки необходимо использовать файл User Constraints File (UCF). В нём содержится информация о цоколёвке микросхемы применительно к разрабатываемому проекту. Такая технология позволяет отвязать модули исходного описания от готовой прошивки и, таким образом, обеспечить универсальность этих модулей.

Для начала необходимо создать UCF файл. Для этого необходимо включить в проект новый модуль (см. Создание нового модуля) и выбрать тип *User Constraints.* Далее программа спросит к какому модулю необходимо привязать новый модуль. После создания UCF файла необходимо привязать маркеры схемы к выводам микросхемы. Для этого необходимо выполнить команду *Floorplan IO-Pre-Synthesiss* (рис. 14).

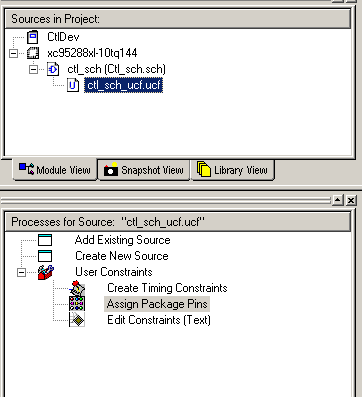


Рис. 14. Рабочая область основного окна

Навигатора проекта пакета WebPACK ISE

Откроется окно редактора выводов микросхемы, изображённое на рис. 15.

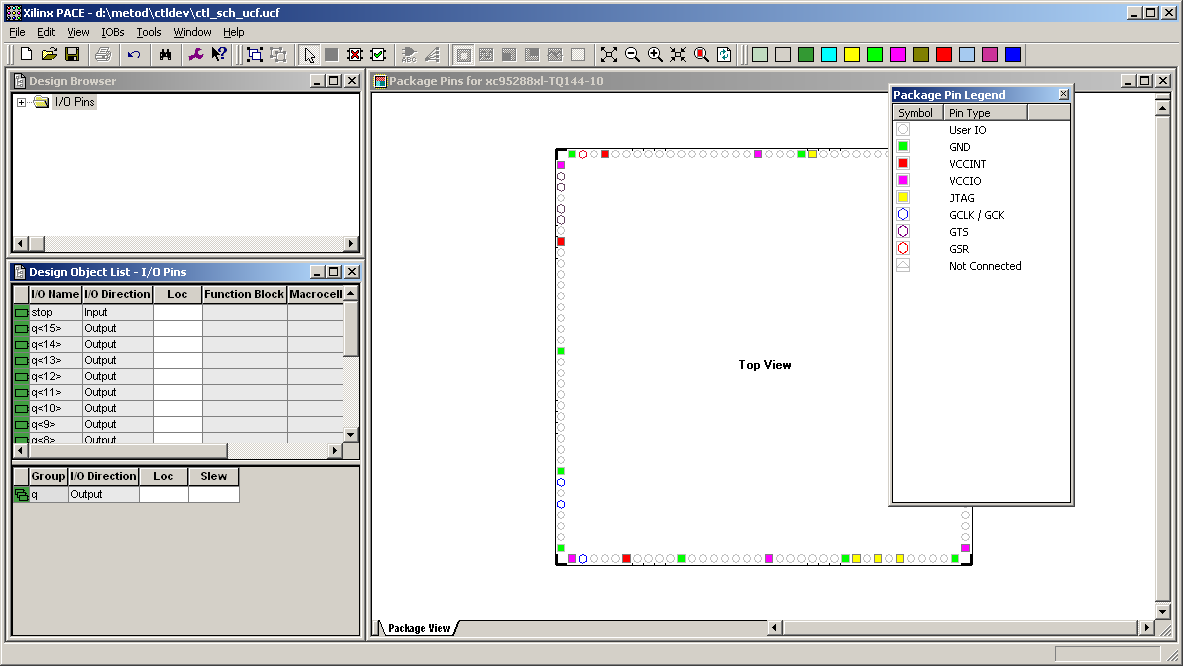


Рис. 15. Редактор выводов микросхемы

Для привязки маркеров схемы к выводам ПЛИС, необходимо указателем мыши захватить нужный вход или выход (при нажатой левой кнопке) и перетащить его на требуемый вывод схематического изображения микросхемы в правой части окна.

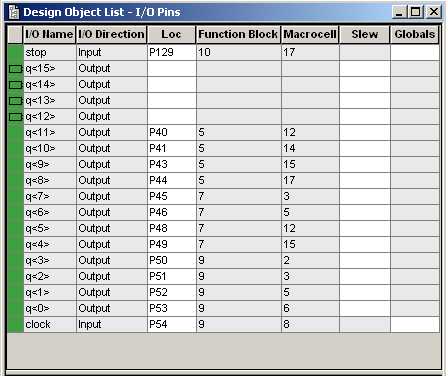


Рис. 16. Назначение выводов ПЛИС

**Синтез проекта с использованием средств XST пакета WebPACK ISE**

После подготовки принципиальной схемы и файла ограничений UCF можно приступить к выполнению синтеза, в процессе которого из файлов HDL-описаний проектируемого устройства формируется файл списка соединений в формате EDIF (Electronic Data Interchange Format). Синтезированный файл представляет собой текстовое (ASCII) описание проекта, но на более низком логическом уровне в формате, воспринимаемом программами трассировки Xilinx. При этом принципиальная схема проекта автоматически преобразуется в HDL-формат, соответствующий выбранным средствам синтеза.

**Размещение и трассировка проектов, реализуемых на базе семейств ПЛИС CPLD фирмы Xilinx, в пакете WebPACK ISE**

Этап размещения и трассировки проектов, выполняемых на базе семейств ПЛИС CPLD фирмы Xilinx, включает в себя две фазы: трансляции и распределения ресурсов кристалла для реализации проектируемого устройства. В процессе трансляции выполняется объединение всех списков соединений в формате EDIF, входящих в состав проекта, и информации обо всех ограничениях, которая содержится в файлах UCF и NCF(Netlist Constraints File). Результатом фазы трансляции является формирование логического описания проекта в терминах примитивов Xilinx низкого уровня с учетом временных и топологических ограничений, выполненного в формате NGD (Native Generic Database). На второй стадии рассматриваемого этапа производится разбиение логического описания проекта, полученного на предыдущем шаге, на блоки в соответствии с ресурсами выбранного типа ПЛИС. При этом выполняется оптимизация с целью минимизации используемых (необходимых) ресурсов кристалла с учетом заданных ограничений. В результате выполнения этапа размещения и трассировки создается бинарный файл, который описывает использование физических ресурсов кристалла для реализации функций проектируемого устройства.

Перед выполнением процедур рассматриваемого этапа можно установить значения их параметров тем же способом, что и для процесса синтеза. Однако на первых порах рекомендуется оставить значения, установленные по умолчанию.

Далее следует активизировать процесс размещения и трассировки двойным щелчком левой кнопки мыши на строке *Implement Design* в окне процедур *Навигатора проекта* (рис. 14). Информация о ходе его выполнения отображается в окне консольных сообщений.

Завершение выполнения каждой фазы этого процесса отмечается соответствующей пиктограммой в строке с ее названием и сопровождается отчетом о полученных результатах. Для просмотра отчета о выполнении трансляции следует дважды щелкнуть левой кнопкой мыши на строке *Translation Report*.

Чтобы открыть отчет о результатах выполнении размещения и трассировки следует дважды щелкнуть левой кнопкой мыши на строке *Fitter Report* (рис. 15).

Отчёт содержит семь основных разделов, в начале каждого из которых указано его название. В разделе *Resource Summary* приведена общая информация о ресурсах ПЛИС и их использовании для реализации проектируемого устройства.

В начале этого раздела указаны данные о суммарном количестве и числе используемых макроячеек, термов, регистров, выводов и функциональных блоков кристалла. Далее располагается общая статистическая информация о сигналах, выводах, макроячейках, глобальных ресурсах и режиме потребления макроячеек.

Раздел *Resources Used by Successfully Mapped Logic* содержит подробную информацию о ресурсах ПЛИС, использованных для реализации проектируемого устройства. В разделе *Function Block Resource Summary* приведены общие статистические данные об использовании функциональных блоков кристалла.

В четвёртом разделе отчета расположена подробная информация об использовании ресурсов каждого функционального блока ПЛИС. Этот раздел состоит из N самостоятельных, одинаковых по составу, частей с названиями *FB1 – FBN*, где N – количество функциональных блоков в используемом кристалле.

В разделе *Implemented Equations* представлены логические выражения, описывающие проектируемое устройство на этапе его реализации. Раздел *Device Pin Out* в наглядной форме отражает назначение всех выводов кристалла ПЛИС после загрузки конфигурационных данных проекта.

В заключительной части отчета, *Compiler Options*, приведены значения основных параметров процесса размещения и трассировки.

При получении успешных результатов размещения и трассировки можно перейти к заключительному этапу разработки проектируемого устройства.

**Программирование ПЛИС семейств CPLD с использованием модуля iMPACTпакета WebPACK ISE**

Прежде, чем приступить к программированию ПЛИС, необходимо преобразовать результаты, полученные на этапе размещения и трассировки проекта в кристалл, в формат, воспринимаемый средствами программирования. Для создания конфигурационной последовательности (файла программирования) следует дважды щелкнуть левой кнопкой мыши на строке *«Generate Programming File»*, расположенной в окне процессов *Навигатора проекта* (рис. 14).

Информация о ходе его выполнения отображается в окне консольных сообщений и строке состояния. После успешного завершения этого процесса, отмеченного соответствующей пиктограммой в строке *«Generate Programming File»* можно приступать непосредственно к программированию ПЛИС.

Для конфигурирования ПЛИС семейств CPLD, выпускаемых фирмой Xilinx, не требуется специальных аппаратных средств программирования, достаточно загрузочного кабеля.

Прежде чем приступить непосредственно к работе с модулем программирования ПЛИС iMPACT, который входит в состав пакета WebPACK ISE, рекомендуется присоединить загрузочный кабель к параллельному порту ПК через программатор и специальным контактам платы проектируемого устройства, предназначенным для конфигурирования кристаллов (JTAG-порту). Для программирования ПЛИС семейств CPLD фирмы Xilinx используются следующие сигналы, описанные в спецификации стандарта JTAG (IEEE Standard 1149.1):

– Test Data In (TDI);

– Test Mode Select (TMS);

– Test Clock (TCK);

– Test Data Out (TDO).

Эти сигналы в процессе конфигурирования подаются на одноименные выводы ПЛИС. Если в состав разрабатываемой системы входят несколько кристаллов ПЛИС, то их специальные выводы, используемые для программирования и периферийного сканирования, могут быть соединены последовательно. Таким образом, формируется последовательная цепочка периферийного сканирования кристаллов ПЛИС, подключенная к соответствующей группе контактов JTAG-интерфейса.

После этого следует подать напряжение питания на разработанное устройство. Такая последовательность обеспечивает возможность автоматического обнаружения и инициализации загрузочного кабеля и кристаллов ПЛИС при активизации программы iMPACT. Если загрузочный кабель подключается после запуска модуля программирования, то в этом случае необходимо выполнить операции установки типа и параметров используемого кабеля.

Далее следует активизировать модуль программирования двойным щелчком левой кнопки мыши на строке *«Configure Device (iMPACT)»* в окне процедур *Навигатора проекта* (рис. 14). Работа программы iMPACTв этом случае начинается с обнаружения загрузочного кабеля. Если программе не удается автоматически идентифицировать загрузочный кабель, то после соответствующего предупреждения выводится диалоговая панель ручной установки его параметров. В этой панели необходимо последовательно установить следующие параметры:

– *Communication Mode* – вид интерфейса, используемого для коммутации с ПК (тип загрузочного кабеля): Parallel, MultiLinx/USB, MultiLinx/Serial;

– *Port* – номер порта, к которому подключен кабель загрузки;

– *Baud Rate* – скорость передачи данных (только для MultiLinx/Serial). В нашем случае это *Parallel, LPT1*.

При успешном обнаружении присоединенного загрузочного кабеля производится автоматический поиск и инициализация цепочки периферийного сканирования ПЛИС, подключенной к выбранному JTAG-порту.

Информация об обнаруженной цепочке периферийного сканирования ПЛИС отображается в графической форме в рабочей области основного окна и в текстовом виде в окне регистрации сообщений программы iMPACT(рис. 17)*.* Под каждым условным графическим образом ПЛИС указывается ее тип и название соответствующего файла, содержащего конфигурационную последовательность.

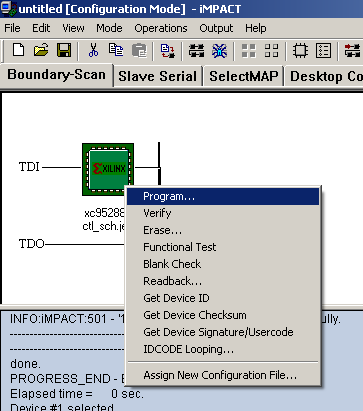


Рис. 17. Отображение цепочки периферийного сканирования

в окне модуля iMPACTпакета WebPACK ISE

Для программирования ПЛИС необходимо выбрать кристалл, поместив на его УГО курсор и щелкнув левой кнопкой мыши, после чего выполнить команду *Program* из меню *Operations* или контекстно-зависимого всплывающего меню, после активизации которого на экран выводится диалоговая панель параметров процесса загрузки конфигурации ПЛИС (рис. 18). Эта панель содержит группу общих параметров программирования кристаллов и группы опций, относящихся к конкретным семействам ПЛИС.

Ниже рассматриваются общие параметры процесса конфигурирования кристаллов семейств CPLD. Эти опции могут принимать одно из двух возможных значений: разрешающее («включено») и запрещающее («выключено»).

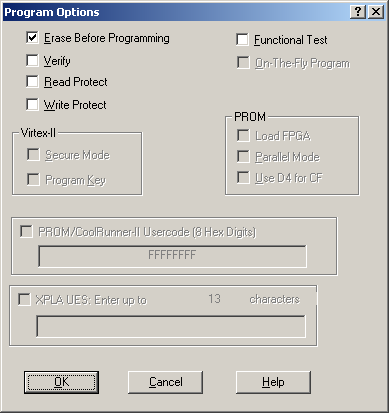


Рис. 18. Диалоговая панель параметров процесса

загрузки конфигурации ПЛИС

Параметр *Erase Before Programming* позволяет разработчику установить режим предварительного «стирания» конфигурационных данных, находящихся во внутренней энергонезависимой памяти кристалла, перед его программированием.

Значение параметра *Verify* определяет использование операции контроля конфигурационных данных в ходе программирования ПЛИС.

Параметр *Read Protect* предназначен для установки защиты от несанкционированного чтения (копирования) загружаемых конфигурационных данных. Программирование ПЛИС с использованием защиты от чтения устанавливает код секретности, который «сбрасывается» только при выполнении операции полного «стирания».

С помощью параметра *Write Protect* разработчику предоставляется возможность установки защиты от случайного перепрограммирования ПЛИС.

После установки всех необходимых значений параметров следует подтвердить их нажатием кнопки «ОК» в нижней части диалоговой панели (рис. 19), что приводит к запуску операции программирования выбранного кристалла. Завершение процесса конфигурирования отмечается соответствующими сообщениями в рабочей области и окне регистрации сообщений программы iMPACT.

**Задачи и порядок проведения работы**

1. Ознакомиться с конструкцией и основными элементами управления и индикации лабораторного стенда ЛСЦ-003;

2. Ознакомиться с пользовательским интерфейсом основных программ пакета WebPACK ISE;

3. Создать проект цифрового устройства с использованием схемотехнического редактора ECS пакета WebPACK ISE;

4. Запрограммировать спроектированное устройство в ПЛИС xc95288xl семейства CPLD с использованием модуля iMPACT;

5. Проверить правильность функционирования спроектированного устройства на лабораторном стенде ЛСЦ-003.

**Форма отчётности студентов:**

индивидуальный отчёт в электронной форме с типовым титульным листом в формате MS Word с последующей распечаткой на бумажном носителе.

**Содержание отчёта:**

1. Краткие теоретические сведения о структуре и функциональном назначении лабораторного стенда ЛСЦ-003;

2. Функциональный состав и назначение основных программ пакета WebPACK ISE;

3. Схема устройства, подготовленная для прошивки в ПЛИС;

4. Отчёт об испытаниях реализованного цифрового устройства на лабораторном стенде ЛСЦ-003;

5. Выводы в развёрнутой форме с соответствующими пояснениями.

Выполнение и защита лабораторной работы производится ***строго*** во время занятий. Защита работы – демонстрация работоспособности спроектированного цифрового устройства, собеседование с преподавателем и выполнение индивидуальных заданий.

**Вопросы для самоконтроля и подготовки к защите лабораторной работы № 1**

1. Опишите состав и функциональное назначение основных структурных блоков лабораторного стенда ЛСЦ-003;

2. Опишите последовательность действий при создании нового проекта;

3. Опишите порядок действий при наименовании шины в зависимости от поступающих на нее выходных сигналов;

4. Опишите порядок действий при загрузке конфигурационной последовательности в ПЛИС стенда ЛСЦ-003;

5. Опишите последовательность действий при проверке работоспособности созданного проекта на стенде ЛСЦ-003.

**ЛАБОРАТОРНАЯ РАБОТА № 2. СОЗДАНИЕ ПРОЕКТА ЦИФРОВОГО УСТРОЙСТВА НА ПРИМЕРЕ СИНТЕЗА СХЕМЫ ПОЛНОГО ДЕШИФРАТОРА СЕМИСЕГМЕНТНОГО СВЕТОДИОДНОГО ИНДИКАТОРА**

**Синтез схемы дешифратора**

Дешифратор – это устройство, преобразующее входной двоичный код в другой код, подаваемый на входы устройств индикации или управления.

Самый простой пример дешифратора – управление семисегментным индикатором. На вход устройства подается четырёхразрядное число, которое определяет отображаемый на индикаторе символ. К семи выходам устройства подключаются соответствующие сегменты светодиодного индикатора с общим анодом. Внешний вид индикатора, схема соединения и обозначение его сегментов приведены на рис. 19.

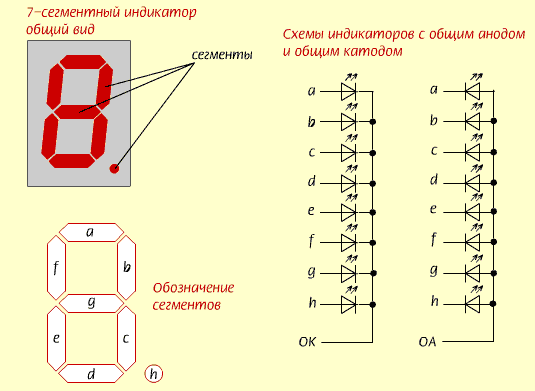


Рис. 19. Устройство семисегментного светодиодного индикатора

Перед началом синтеза семисегментного дешифратора приведем подробное описание его работы. Для того чтобы нужный сегмент индикатора с общим анодом (ОА) светился, необходимо подать на него напряжение низкого уровня (логический «0»). Пример изображения цифры 1 приведен на рис. 20.

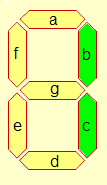


Рис. 20. Пример вывода цифры 1

Для того чтобы закодировать символы шестнадцатеричной системы счисления необходимо 4 двоичных разряда. Так как дешифратор должен управлять свечением семи светодиодных сегментов, то потребуется 7 выходных сигналов (выходных числовых последовательностей).

Для каждой цифры и каждого символа необходимо выбрать его начертание, так как это будет определять внутреннюю структуру дешифратора. В качестве примера некоторые шестнадцатеричные цифры приведены на рис. 21.

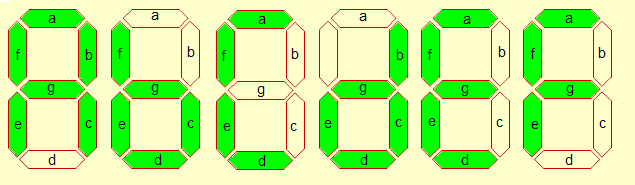


Рис. 21. Изображение шестнадцатеричных цифр A – F

После определения начертания символов переходим к проектированию схемы дешифратора. В первую очередь необходимо составить таблицу истинности, по которой производится построение и последующая минимизация СДНФ. В дальнейшем на основе минимизированных формул ДНФ можно построить принципиальные схемы блоков семисегментного дешифратора.

Таблица 1 – Фрагмент таблицы истинности проектируемого устройства

| Входные сигналы | | | | Выходные сигналы дешифратора | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X8 | X4 | X2 | X1 | a | b | c | d | e | f | g |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |  |  |  |  |  |  |  |
| 0 | 1 | 0 | 0 |  |  |  |  |  |  |  |
| 0 | 1 | 0 | 1 |  |  |  |  |  |  |  |
| 0 | 1 | 1 | 0 |  |  |  |  |  |  |  |
| 0 | 1 | 1 | 1 |  |  |  |  |  |  |  |
| 1 | 0 | 0 | 0 |  |  |  |  |  |  |  |
| 1 | 0 | 0 | 1 |  |  |  |  |  |  |  |
| 1 | 0 | 1 | 0 |  |  |  |  |  |  |  |
| 1 | 0 | 1 | 1 |  |  |  |  |  |  |  |
| 1 | 1 | 0 | 0 |  |  |  |  |  |  |  |
| 1 | 1 | 0 | 1 |  |  |  |  |  |  |  |
| 1 | 1 | 1 | 0 |  |  |  |  |  |  |  |
| 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |

Минимизацию построенных по табл. 1 формул СДНФ можно провести с использованием карт Карно.

**Задачи и порядок проведения работы**

1. Разработать описание проектируемого устройства (дешифратора семисегментного индикатора) в схемотехнической форме;

2. Выполнить логический синтез устройства в заданном элементном базисе (элементы «И-НЕ»);

3. Создать новый проект, указав семейство, тип ПЛИС и средства синтеза;

4. Выполнить ввод схемы с использованием схемотехнического редактора, размещение и трассировку проекта в кристалле;

5. Загрузить конфигурационные данные проекта в кристалл (выполнить программирование ПЛИС);

6. Провести проверку работы синтезированной схемы дешифратора семисегментного индикатора на лабораторном стенде ЛСЦ-003.

**Форма отчётности студентов:**

индивидуальный отчёт в электронной форме с типовым титульным листом в формате MS Word с последующей распечаткой на бумажном носителе.

**Содержание отчёта:**

1. Техническое задание на проектирование цифрового устройства;

2. Логические формулы (СДНФ) выходных функций дешифратора и их минимизация с использованием карт Карно или других методов;

3. Схемы отдельных сегментов, необходимые для реализации проектируемого устройства в целом;

4. Схема устройства, подготовленная для прошивки в ПЛИС;

5. Отчёт об испытаниях реализованного устройства на лабораторном стенде ЛСЦ-003 и вывод о соответствии его техническому заданию;

6. Выводы в развёрнутой форме с соответствующими пояснениями.

Выполнение и защита лабораторной работы производится ***строго*** во время занятий. Защита работы – демонстрация работоспособности спроектированного цифрового устройства, собеседование с преподавателем и выполнение индивидуальных заданий.

**Вопросы для самоконтроля и подготовки к защите лабораторной работы № 2**

1. Опишите назначение дешифраторов для управления семисегментными индикаторами;

2. Опишите принципы построения СДНФ по таблице истинности;

3. Опишите использование карт Карно для упрощения логических выражений и схем;

4. Опишите переход к логическому базису «И-НЕ»;

5. Приведите известные вам критерии сложности логических схем;

6. Опишите процесс проверки работоспособности спроектированного устройства на лабораторном стенде ЛСЦ-003.

**ЛАБОРАТОРНАЯ РАБОТА № 3. МОДЕЛИРОВАНИЕ СПРОЕКТИРОВАННОГО ЦИФРОВОГО УСТРОЙСТВА С ИСПОЛЬЗОВАНИЕМ ПАКЕТА MODELSIM**

**Этапы моделирования цифровых устройств, разрабатываемых на основе кристаллов ПЛИС семейств CPLD фирмы Xilinx**

В процессе проектирования средства пакета *WebPACK ISE* позволяют сформировать несколько видов моделей разрабатываемого устройства. После создания модулей исходного описания проекта генерируется поведенческая модель, которая позволяет выполнить их функциональную верификацию.

На этой стадии проектирования отсутствует информация о значениях задержек распространения сигналов, поэтому при функциональном моделировании можно обнаружить только логические и синтаксические ошибки в описании разрабатываемого устройства.

Таким образом, функциональное моделирование устройства позволяет выполнить предварительную верификацию проекта. На этом этапе фактически не учитываются временные характеристики и особенности архитектуры кристалла, на базе которого предполагается реализация проектируемой системы. Часто в литературе этот процесс называют моделированием на уровне регистровых передач (*Register**Transfer Level, RTL*). Для функционального моделирования проекта используется библиотека *UniSim Library*, элементы которой имеют единичные задержки.

Состав моделей, формируемых на последующих этапах процесса проектирования, зависит от типа семейства ПЛИС (*FPGA* или *CPLD*), выбранного для реализации разрабатываемого устройства. Только после проведения этапов синтеза, размещения и трассировки становится доступной информация об используемых ресурсах кристалла и задержках распространения сигналов, которая необходима для формирования более адекватной модели.

В отличие от функционального последующие этапы моделирования выполняются с применением библиотеки *SimPrim**Library*, которая содержит описание элементов на уровне ресурсов кристалла. Эта библиотека позволяет учитывать информацию о задержках распространения сигналов, которая содержится в соответствующих файлах, имеющих стандартный формат *SDF* (*Standard Delay Format*).

Так как содержание этапа реализации (*Implementation*) коренным образом отличается для семейств *CPLD* и *FPGA*, что обусловлено их архитектурными особенностями, то процесс генерации моделей, учитывающих временные характеристики используемых кристаллов, включает в себя различные фазы, в зависимости от выбранного типа ПЛИС. Для проектов, выполняемых на основе ПЛИС семейств *CPLD*, кроме функциональной модели может быть сформирована только полная временная модель устройства.

Таким образом, процесс проектирования систем на базе кристаллов семейств *CPLD* включает в себя два этапа моделирования: функционального и полного временного. Все виды моделирования цифровых устройств, разрабатываемых в среде САПР *WebPACK ISE*, выполняются с помощью системы HDL-моделирования *ModelSim*.

**Подготовка тестового модуля проекта**

Прежде чем приступать непосредственно к выполнению этапов моделирования, необходимо сформировать файл описаний тестовых воздействий, если он не был подготовлен на этапе создания исходных модулей проекта. Методы разработки тестового модуля проекта подробно рассмотрены ниже. На всех этапах моделирования может использоваться один и тот же модуль описаний тестовых воздействий.

Для выполнения этапов функционального и временного моделирования необходимо сформировать описание тестовых воздействий, используемых в процессе верификации. Подготовка тестовых модулей может осуществляться как на этапе создания исходных описаний проектируемого устройства, так и непосредственно перед активизацией процесса моделирования.

**Структура тестового модуля проекта**

Тестовый модуль проекта фактически содержит представление модели испытательного стенда для разрабатываемого устройства на языке описания аппаратуры HDL, используемом в процессе проектирования. В дальнейшем предполагается, что процесс разработки выполняется при использовании средств синтеза XST (Xilinx Synthesis Technology) VHDL. В этом случае тестовый модуль проекта имеет стандартную структуру VHDL-описания, которая включает в себя следующие элементы:

1) ссылки на используемые библиотеки и пакеты;

2) описание интерфейса объекта (ENTITY);

3) описание архитектуры объекта;

4) декларацию компонента, представляющего модуль описания верхнего уровня иерархии проектируемого устройства;

5) декларацию сигналов, используемых для подачи входных тестовых воздействий и контроля выходных реакций моделируемой системы;

6) выражение создания экземпляра компонента с подключением соответствующих сигналов;

7) формирование поведения входных тестовых сигналов.

В данном случае под объектом описания понимается модель испытательного стенда в целом, поэтому он не имеет интерфейсных сигналов (портов). Архитектура этого объекта содержит единственный компонент, представляющий разрабатываемое устройство.

«Мастер» подготовки модулей исходного описания проекта позволяет автоматически сформировать основу тестового файла, которая содержит все перечисленные выше элементы его структуры, за исключением секции, описывающей поведение входных сигналов. Формирование входных тестовых воздействий может выполняться в текстовом виде (на языке VHDL) при использовании встроенного HDL-редактора или в графической форме (в виде временных диаграмм) с помощью утилиты генерации тестов *HDL Bencher,* которая входит в состав пакета WebPACK ISE. Последний способ является наиболее наглядным и не требует знаний языков HDL. Программа *HDL Bencher* автоматически транслирует созданные временные диаграммы тестовых сигналов при их сохранении в текстовый формат.

**Создание тестового модуля проекта в форме временных диаграмм**

Для автоматического формирования основы файла тестовой последовательности необходимо активизировать режим создания нового модуля исходного описания проекта, воспользовавшись кнопкой  на оперативной панели или командой *New Source* из раздела *Project*основного меню *Навигатора проекта.* В открывшейся диалоговой панели, показанной на рис. 22, в качестве типа нового модуля следует выбрать *Test Bench Waveform.* Далее необходимо задать его имя и указать место расположения файла на диске в поле редактирования *Location.*

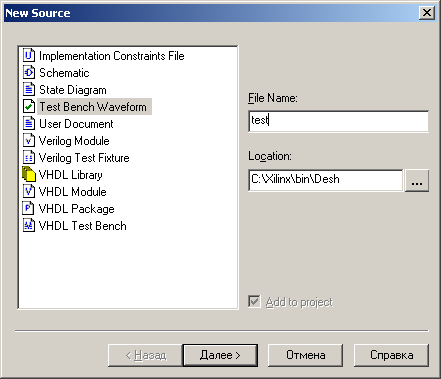


Рис. 22. Диалоговая панель установки исходных параметров

тестового модуля проекта

Установка значений всех необходимых исходных параметров создаваемого модуля завершается нажатием клавиши *Далее (Next****),***которая находится в нижней части диалоговой панели (рис. 22). В результате чего на экране отображается следующая диалоговая панель, содержащая список модулей исходного описания проекта. В предлагаемом списке следует выделить элемент (файл, содержащий схему), с которым ассоциируется создаваемый тестовый модуль, поместив курсор на соответствующую строку и щелкнув левой кнопкой мыши. Сделанный выбор подтверждается нажатием клавиши *Далее (Next),* расположенной в нижней части диалоговой панели (рис. 23).

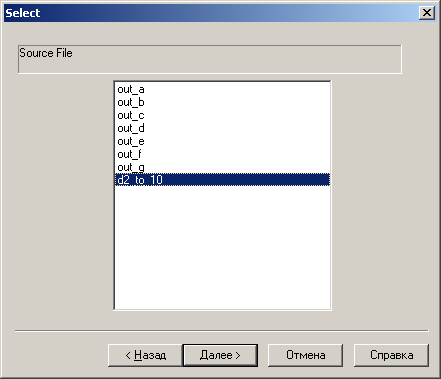


Рис. 23. Диалоговая панель выбора объекта

для генерации тестового файла

При успешном создании основы тестового модуля открывается информационная панель, показанная на рис. 24. В этой панели отображаются значения всех параметров создаваемого модуля. Если необходимо изменить значение какого-либо параметра, то кнопка *Назад (Back)* позволяет вернуться к предыдущей диалоговой панели. Процесс создания основы тестового модуля завершается нажатием кнопки *Готово (Finish),* находящейся в нижней части информационной панели (рис. 24).

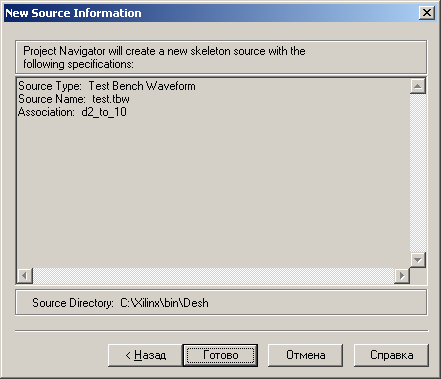


Рис. 24. Информационная панель параметров

создаваемого тестового модуля проекта

Выбор объекта завершается нажатием клавиши *Готово*, после чего открывается диалоговая панель инициализации временных параметров (рис. 25).

Временные параметры, необходимые для формирования тестовых сигналов, различаются для синхронных и асинхронных устройств. Программа HDL Bencher автоматически определяет тип устройства, для которого разрабатываются тестовые последовательности.

Для синхронного устройства необходимо, прежде всего, проконтролировать и при необходимости изменить выбор основного тактового сигнала, в поле *Single clock*, используя выпадающий список интерфейсных сигналов устройства. Затем следует выбрать единицы измерения временных параметров, используя выпадающий список возможных значений поля *Time scale*. Далее нужно определить длительность состояний высокого и низкого логических уровней периода тактового сигнала в поле редактирования *Clock high time* и *Clock low time* соответственно. Кроме того, в поле редактирования Input setup time следует указать минимальное значение времени установки входных сигналов, а в поле *Output valid delay* максимальное значение задержки выходных сигналов.

Программа HDL Bencher автоматически определяет, чем тактируется устройство – фронтом (*Rising edge*), спадом (*Falling edge*) тактового сигнала или тем и другим (*Dual edge*), но разработчик может выбрать это вручную, используя соответствующие кнопки диалоговой панели (рис. 25). При создании тестов для комбинационных устройств нужно указать только значения временных интервалов между моментами подачи входных воздействий и контроля выходных сигналов *Check outputs и Assign Inputs*. Эти параметры позволяют исключить возможные конфликты в моменты изменения входных и выходных сигналов.

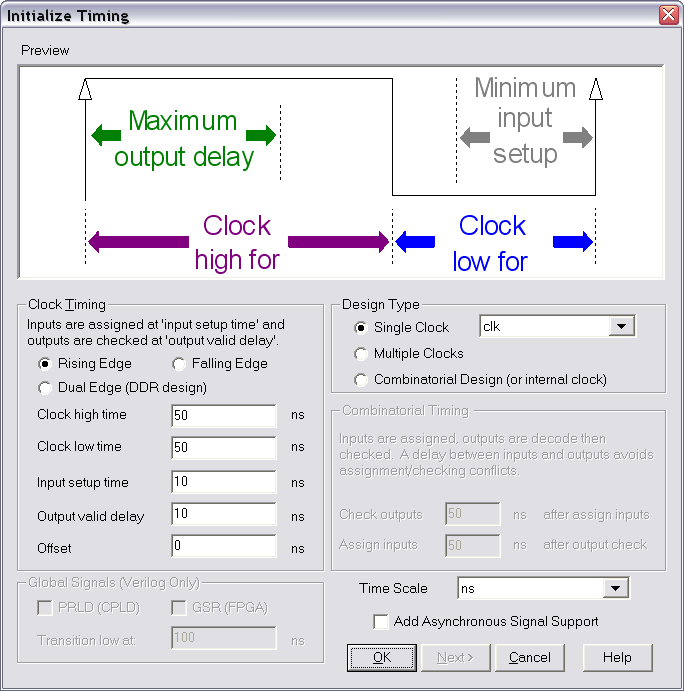


Рис. 25. Диалоговая панель инициализации временных

параметров утилиты HDL Bencher

Процесс установки требуемых значений временных параметров завершается нажатием кнопки ОК, расположенной в нижней части диалоговой панели (рис. 25), после чего в рабочей области окна программы HDL Bencher отображаются заготовки временных диаграмм (рис. 26).

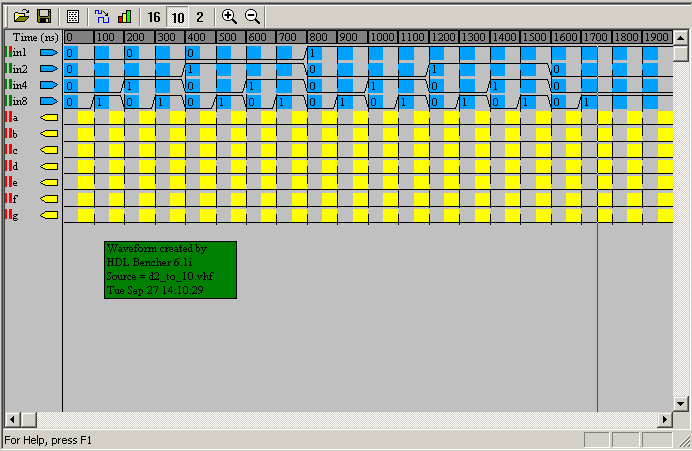


Рис. 26. Заготовки временных диаграмм

В верхней строке рабочей области окна программы *HDL Bencher* расположена временная шкала, деления которой соответствуют периодам тактового сигнала. В левом столбце отображается список названий интерфейсных сигналов устройства. Справа от названия сигнала приведена пиктограмма, обозначающая его тип: тактовый; входной; выходной и двунаправленный соответственно. Сигналы векторного типа отображаются в виде шин соответствующей разрядности.

Изменить значение сигнала в какой-либо момент времени можно несколькими способами. Чтобы переключить значение сигнала на противоположное (из состояния низкого логического уровня в высокое или наоборот), достаточно поместить курсор в требуемое место временной диаграммы и щелкнуть левой кнопкой мыши. Чтобы выбрать значение сигнала из списка возможных состояний, следует щелчком правой кнопки активизировать контекстно-зависимое всплывающее меню, в котором выбрать строку *Set Value.* В результате выполнения указанных действий на экран выводится диалоговая панель, содержащая поле выбора значения сигнала (рис. 27). Для установки требуемого значения сигнала следует нажать кнопку управления выпадающим списком возможных состояний, в котором выбрать затем соответствующую строку, после чего подтвердить выбор нажатием кнопки *ОК* (рис. 27).

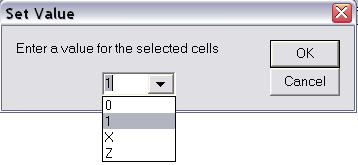


Рис. 27. Диалоговая панель выбора значения сигнала

Чтобы сформировать последовательность значений для выбранного сигнала, можно воспользоваться соответствующим «мастером» *Pattern Wizard.* Для его активизации необходимо поместить курсор мыши в исходную точку временной диаграммы и дважды щелкнуть левой кнопкой, после чего воспользоваться появившейся кнопкой *Pattern.* В диалоговой панели «мастера» *Pattern Wizard,* показанной на рис. 28, следует выбрать алгоритм переключений сигнала в поле *Choose Pattern**и* указать: начальное значение в поле редактирования *Initial Value,* альтернативное значение в поле *Other Value,* интервал переключения в поле *Toggle Every**и* количество формируемых периодов сигнала в поле *Repeat pattern.* После нажатия кнопки *ОК,* расположенной в нижней части диалоговой панели (рис. 28), сформированный фрагмент временной диаграммы отображается в рабочей области. Команды меню *Edit*и контекстно-зависимого всплывающего меню позволяют копировать в буфер и затем вставлять в нужное место выбранные фрагменты временных диаграмм.

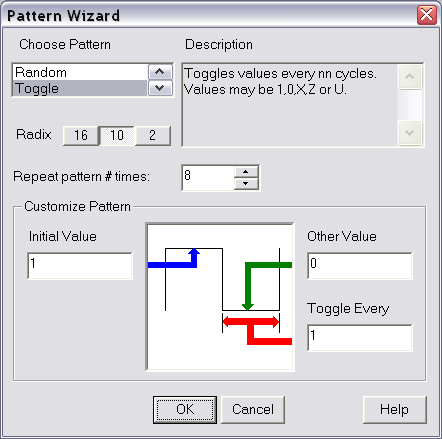


Рис. 28. Диалоговая панель «мастера» Pattern Wizard

На изображении временных диаграмм присутствует указатель конечной точки тестовой последовательности (вертикальная синяя линия), положение которого устанавливается автоматически или «вручную». Выбор режима определения окончания тестового вектора осуществляется с помощью команды *Configuration* из всплывающего меню *Options,* в результате выполнения которой на экран выводится диалоговая панель, показанная на рис. 29. Параметр *Automatically determine end of test bench* управляет выбором режима, в котором выполняется установка маркера конечной точки тестового вектора. По умолчанию используется значение «включено», соответствующее автоматическому режиму, при котором указатель располагается в начале тактового интервала, следующего за моментом последнего изменения входных сигналов. При перемещении маркера с помощью мыши происходит переключение из автоматического режима в «ручной».

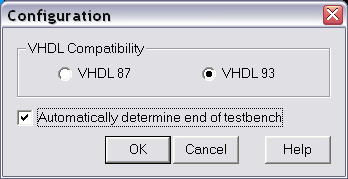


Рис. 29. Диалоговая панель параметров конфигурации

программы HDL Bencher

Параметр VHDL Compatibility, представленный в этой же диалоговой панели в виде двух кнопок, позволяет выбрать стандарт языка VHDL, в соответствии с которым формируется описание тестовой системы. По умолчанию установлен режим совместимости со стандартом IEEE VHDL Std 1076-1993.

Завершив редактирование временных диаграмм тестовых сигналов, следует сохранить их, используя команду *Save Waveform* из меню *File*. При этом, если указатель окончания тестовой последовательности был установлен до момента последнего изменения входных или контроля формирования выходных сигналов, выводится панель предупреждения, в которой предлагается перенести маркер в соответствующую позицию.

По окончании формирования тестового файла следует выделить строку с его названием в окне исходных модулей Навигатора проекта, поместив на нее курсор мыши и щелкнув левой кнопкой. В результате в окне процессов отображается интерактивный список этапов моделирования проектируемого устройства. Содержание окна процедур в этом режиме определяется видом семейства ПЛИС, выбранного для реализации проекта, но независимо от типа используемого кристалла первым в списке является этап функционального моделирования. На рис. 30 показан вид рабочей области Навигатора проекта, в которой отображаются этапы моделирования в процессе проектирования устройства на базе ПЛИС семейств CPLD при использовании средств синтеза XST VHDL. Строка *Simulate Behavioral VHDL Model* соответствует этапу функционального моделирования, a *Simulate Post-Fit VHDL Model* полного временного.

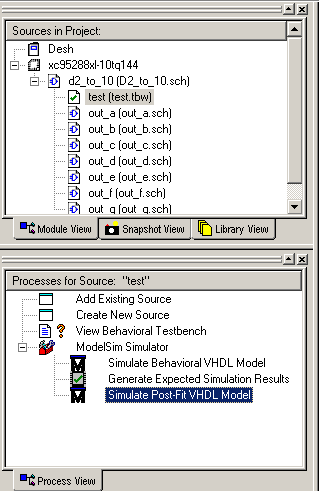


Рис. 30. Отображение этапов моделирования устройств,

в рабочей области Навигатора проекта пакета WebPACK ISE

Все этапы моделирования проекта выполняются в пакетном режиме. Поэтому далее основное внимание уделяется описанию параметров инициализации программы ModelSim и временной модели, с помощью которых осуществляется управление процессом моделирования. Состав этих параметров зависит от языка описания аппаратуры HDL и средств синтеза, используемых в процессе проектирования. В последующих разделах данной главы будут рассмотрены параметры процессов моделирования проектов, разрабатываемых на основе кристаллов семейств CPLD с применением средств синтеза XST VHDL.

**Установка значений параметров функционального моделирования проекта**

Перед запуском средств моделирования следует проконтролировать и при необходимости установить требуемые значения параметров инициализации программы *ModelSim* и соответствующей модели. Для этого нужно в окне процессов (рис. 27) щелчком левой кнопки мыши выделить строку *Simulate Behavioral VHDL Model,* после чего нажать кнопку расположенную на оперативной панели *Навигатора проекта****,***иливоспользоваться командой *Properties*контекстно-зависимого всплывающего меню, которое выводится при щелчке правой кнопки мыши. В результате выполненных действий на экране монитора отображается диалоговая панель параметров процесса функционального моделирования, вид которой показан на рис. 31.

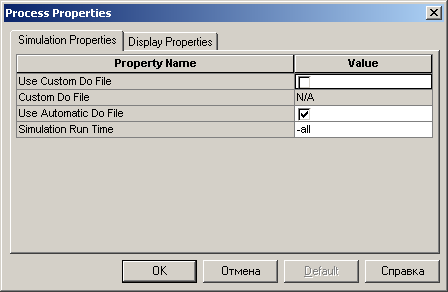


Рис. 31. Диалоговая панель параметров процесса функционального

моделирования (страница *Simulation Properties*)

Эта диалоговая панель содержит две страницы, снабженные закладками с их названиями: *Simulation Properties, Display Properties*. Каждая из этих страниц содержит соответствующую группу параметров, представленных в виде таблицы. Чтобы изменить значение параметра, следует активизировать соответствующее поле таблицы, поместив на него курсор и щелкнув левой кнопкой мыши, а затем воспользоваться кнопкой управления выпадающим списком значений или, если параметр имеет два состояния («включено»/«выключено»), щелкнуть левой кнопкой мыши на поле индикатора состояния. Для некоторых параметров требуемое значение может быть введено непосредственно с клавиатуры. Содержание страниц в диалоговой панели параметров моделирования зависит от выбранного семейства ПЛИС и языка описания HDL. Рассмотрим подробнее эти параметры для случая использования семейств ПЛИС CPLD и средств синтеза XST VHDL.

На странице *Simulation Properties* представлены общие параметры процесса моделирования. Параметр *Custom Do File* позволяет указать название командного файла, который будет выполняться при активизации средств моделирования в дополнение к файлу, формируемому автоматически пакетом WebPACK ISE. Название командного файла может быть введено непосредственно с клавиатуры после активизации поля редактирования значения этого параметра или выбрано при использовании стандартной диалоговой панели открытия файла, которая выводится при нажатии кнопки с пиктограммой в виде многоточия «...».

Значение параметра *Use Automatic Do File* разрешает или запрещает автоматическое создание и исполнение командного файла, который содержит директивы компиляции всех VHDL-файлов проекта, активизации процесса моделирования и отображения его результатов. По умолчанию установлено значение «включено», разрешающее генерацию и выполнение этого пакетного файла.

На странице *Display Properties* представлены параметры, управляющие отображением окон программы моделирования *ModelSim* при ее активизации (рис. 29). При запуске средств моделирования непосредственно из управляющей оболочки пакета WebPACK ISE автоматически открываются только те окна *ModelSim,* для которых соответствующий параметр установлен в состояние «включено». Параметр *Signal window* определяет режим отображения окна сигналов, *Wave window —* окна временных диаграмм, *Structure window –* окна структуры проекта, *Source window**–* окна исходного кода, *List window* – окна табличной формы результатов моделирования, *Variables window —* окна переменных, *Process window —* окна процессов, *Data Flow window —* окна трассировки сигналов. По умолчанию в начале сеанса моделирования автоматически открываются окна сигналов, временных диаграмм и структуры проекта (рис. 32).

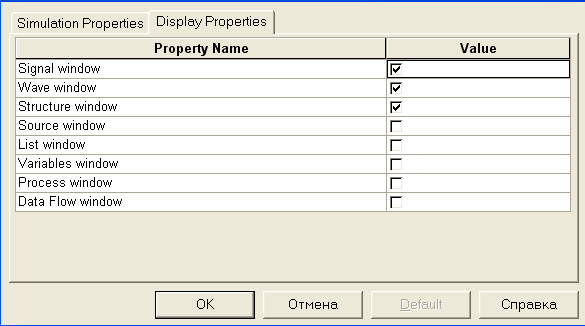


Рис. 32. Страница *Display Properties*диалоговой панели параметров

процесса функционального моделирования

Выполнив необходимые изменения параметров процесса моделирования, следует подтвердить их нажатием клавиши *ОК* в нижней части диалоговой панели (рис. 31 и 32).

**Функциональное моделирование проекта в программе ModelSim**

Процесс функционального моделирования проекта активизируется двойным щелчком левой кнопки мыши на строке *Simulate Behavioral VHDL Model*в окне процессов *Навигатора проекта.* При этом средствами пакета WebPACK ISE автоматически создается пакетный файл, который содержит последовательность команд управления программой *ModelSim,* необходимых для проведения сеанса функционального моделирования проектируемого устройства. Далее автоматически производится запуск средств моделирования, сразу после инициализации которых выполняется этот командный файл (если только параметр *Use Automatic Do File* не был установлен в состояние «выключено»). Информация о ходе выполнения пакетного файла отображается в консольной области основного окна программы *ModelSim.* При этом последовательно открываются дочерние окна этой программы, которые были указаны в параметрах инициализации. После завершения выполнения командного файла при необходимости моделирование может быть продолжено в интерактивном режиме. На рис. 33, в качестве примера, приведены результаты функционального моделирования двоично-десятичного дешифратора.

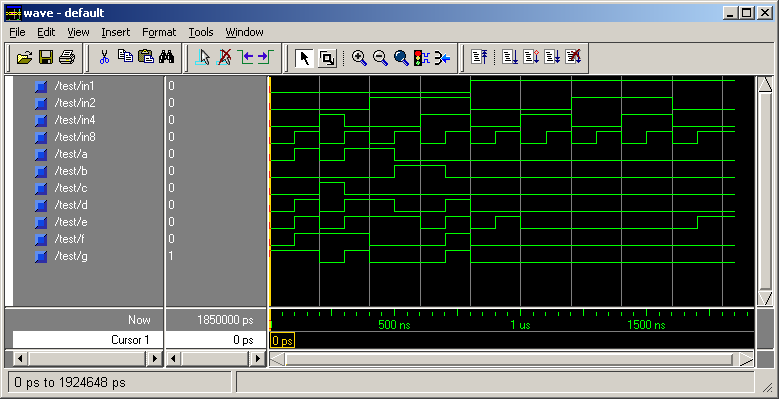


Рис. 33. Результаты функционального моделирования проекта

двоично-десятичного дешифратора

Как правило, наибольший интерес для разработчика представляют результаты моделирования, которые отображаются в графическом виде в окне временных диаграмм программы *ModelSim.* Для их просмотра, прежде всего, необходимо выбрать требуемый масштаб изображения.

Увеличение масштаба осуществляется с помощью кнопки , расположенной на оперативной панели управления окна временных диаграмм, или команды *Zoom In* из всплывающего меню *Zoom****,***доступ которому предоставляет пункт *View*основного меню. Чтобы уменьшить масштаб изображения, следует использовать кнопку быстрого доступа  на оперативной панели или команду *Zoom Out.*

Однократное применение этих инструментов приводит к изменению масштаба в два раза. Для детального просмотра выбранного фрагмента временных диаграмм следует воспользоваться кнопкой  на оперативной панели или командой *Zoom Range.* При этом, чтобы указать интересующую область изображения результатов моделирования, следует поместить курсор мыши на одну из ее границ, нажать левую кнопку мыши и, не отпуская ее, раздвинуть появившиеся курсоры до границ требуемого диапазона, после чего освободить нажатую кнопку.

Перемещение по временным диаграммам осуществляется с помощью элементов прокрутки, расположенных вдоль нижней и правой границ панели результатов моделирования в окне временных диаграмм. Кроме того, на оперативной панели имеются кнопки быстрого доступа  и  . Они обеспечивают перемещение курсора в предыдущую или следующую точку переключения выбранного сигнала. При открытии окна диаграмм сигналов курсор находится в начальной точке временной шкалы, которая соответствует нулевому значению времени моделирования. Чтобы переместить его в требуемую точку, следует расположить указатель мыши в этой позиции и щелкнуть левой кнопкой, после чего изображение курсора будет зафиксировано в новом положении. Допускается также перетаскивание курсора в новую точку с помощью мыши, которое выполняется при нажатой левой кнопке.

Для измерения временных интервалов на диаграммах сигналов, полученных в результате моделирования, следует использовать дополнительные курсоры. Чтобы добавить еще один курсор, необходимо нажать кнопку  на оперативной панели управления. При наличии нескольких указателей активный курсор изображается сплошной линией, а остальные – штриховой. Поместив курсоры на границах измеряемого временного интервала, можно определить его длительность, которая отображается в нижней части окна диаграмм в виде разности координат указателей. При необходимости можно использовать несколько дополнительных курсоров. В последующем для удаления неиспользуемых указателей можно воспользоваться кнопкой  на оперативной панели или командой *Delete Cursor* из всплывающего меню *Edit.*

В ряде случаев для большей наглядности результатов моделирования можно представить совокупность выбранных сигналов в виде шины. Для этого необходимо выделить названия соответствующих сигналов в окне временных диаграмм и выбрать команду *Combine Signals* из всплывающего меню *Tools,* после чего выводится диалоговая панель, показанная на рис. 34. В поле редактирования *Name* с помощью клавиатуры необходимо указать название формируемого виртуального сигнала (шины). Кнопки *Ascending* и *Descending* позволяют выбрать порядок следования сигналов в шине. Параметр *Remove selected signals**after combining* предоставляет возможность удаления сигналов после их объединения в виртуальную шину.

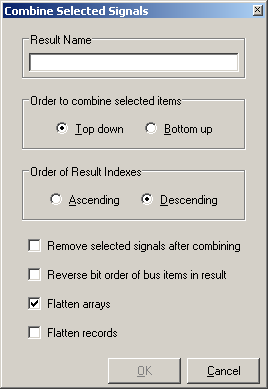


Рис. 34. Диалоговая панель создания виртуальной шины

По умолчанию для этого параметра установлено значение «выключено», запрещающее удаление из окна временных диаграмм сигналов, вошедших в состав виртуальной шины. Сформированный виртуальный сигнал отображается в списке пиктограммой в форме ромба красного цвета, в отличие от реальных сигналов, обозначаемых пиктограммой в виде квадрата синего цвета.

При автоматическом создании командного файла для сеанса моделирования средства пакета WebPACK ISE включают в список контролируемых сигналов только интерфейсные сигналы моделируемого объекта. Но в процессе отладки часто возникает необходимость отображения поведения *внутренних сигналов* в различных модулях проекта. Для этого следует в иерархической структуре проекта, представленной в окне *Structure,* выбрать интересующий объект, поместив курсор мыши на строку с его названием и щелкнув левой кнопкой мыши. При этом в окне *Signals* отобразится соответствующий список сигналов, в котором следует захватить требуемый элемент, поместив курсор мыши на строку с его названием, нажав левую кнопку, и, не отпуская ее, перетащить его в область названий сигналов окна временных диаграмм. Далее можно продолжить моделирование с текущего момента времени или повторить выполненные процедуры сначала. В последнем случае следует сбросить результаты моделирования и вернуть систему в начальное состояние, выполнив команду *Restar****t***из всплывающего меню *Run****,*** доступ к которому открывает пункт *Simulate* главного меню основного окна программы *ModelSim,* или нажав кнопку  на оперативной панели. Повторное моделирование инициируется командой *Run -All* или кнопкой быстрого доступа .

Результаты моделирования, полученные в текущем сеансе, автоматически сохраняются в виде набора данных (dataset) с именем *vsim* в формате WLF (Wave log format).

Чтобы сохранить результаты моделирования в файле с другим именем, следует выполнить команду *Save* из всплывающего меню *File* основного окна программы *ModelSim* или команду *Save Dataset* из выпадающего меню *File* окна временных диаграмм. После выбора строки *sim dataset* или *sim* открывается стандартная панель сохранения файла, в которой нужно указать с помощью клавиатуры требуемое название набора данных. Для завершения этапа функционального моделирования следует выполнить команду *End Simulation* из меню *Simulate* и закрыть основное окно программы *ModelSim****,***после чего можно переходить к временному моделированию, которое рассматривается в следующих разделах.

**Установка значений параметров временного моделирования проекта**

Для контроля и установки требуемых значений параметров временного моделирования нужно в окне процессов щелчком левой кнопки мыши выделить строку *Simulate Post-Fit VHDL Model,* после чего нажать кнопку , расположенную на оперативной панели *Навигатора проекта,* или воспользоваться командой *Properties*контекстно-зависимого всплывающего меню, которое выводится при щелчке правой кнопки мыши.

Отображаемая после этого диалоговая панель параметров временного моделирования, в отличие от функционального, содержит три страницы, снабженные закладками с их названиями: *Simulation Properties, Display Properties* и *Simulation Model Properties* (рис*.* 35).

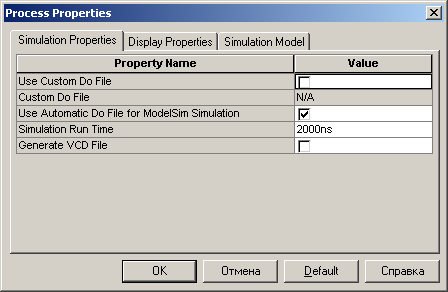


Рис. 35. Диалоговая панель параметров процесса временного

моделирования (страница *Simulation Properties*)

Страница *Simulation Properties* включает в себя все параметры инициализации программы *ModelSim* и процесса моделирования, присутствующие на одноименной странице в случае функциональной верификации. Кроме того, страница *Simulation Properties* для процесса временного моделирования содержит три дополнительных параметра.

*С* помощью параметра *Generate VCD File* осуществляется управление формированием файла результатов в формате VCD (Value Change Dump) в процессе временного моделирования. Этот файл может использоваться, в частности, программой оценки потребляемой мощности *XPower.* При использовании значения «выключено», установленного по умолчанию, автоматическое создание файла VCD в процессе временного моделирования не производится.

Страница *Display Properties* панели параметров процесса временного моделирования содержит те же опции, что и для этапа функционального моделирования (рис. 32). Значение каждого параметра, расположенного на этой странице, определяет режим отображения соответствующего дочернего окна программы *ModelSim.*

На странице *Simulation Model Properties****,***вид которой показан на рис. 36, представлены параметры временной модели проектируемого устройства.

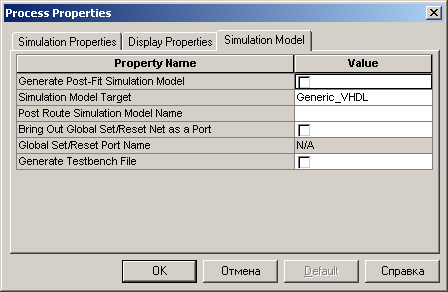


Рис. 36. Страница *Simulation Model Properties* диалоговой панели,

представляющей параметры процесса временного моделирования

Параметр *Bring**Out Global Set/Reset Net as a Port* используется для включения глобальной цепи сброса/установки кристалла в описание интерфейса моделируемого объекта. Если данный параметр находится в состоянии «включено», то глобальный сигнал сброса/установки триггерных ресурсов кристалла GSR преобразуется в формат порта объекта, который представлен в описании верхнего уровня иерархии проекта. Значение «выключено», принятое по умолчанию, соответствует обычному представлению глобальной цепи сброса/установки ПЛИС.

Значение параметра *Global Set/Reset Port Name* позволяет указать название порта, соответствующего глобальной цепи сброса/установки кристалла. Данный параметр используется только при условии, что *Bring Out Global Set/Reset Net as a Port* установлен в состояние «включено». По умолчанию название порта совпадает с идентификатором глобальной цепи сброса/установки – GSR. Новое значение этого параметра вводится с помощью клавиатуры после активизации соответствующего поля редактирования.

Значение параметра *Reset on Configuration Pulse Width* определяет длительность импульса, необходимого для моделирования компонентов, управляющих инициализацией глобальных цепей сброса/установки в начальный момент времени. По умолчанию для этого параметра используется значение 100 нс. Данная опция используется только в случае, если параметр *Bring Out Global Set/Reset Net as a Port* находится в состоянии «выключено».

Параметр *Generate Testbench File* управляет генерацией тестового файла по результатам моделирования. По умолчанию установлено значение «выключено», которое запрещает автоматическую перезапись тестового файла.

Значение параметра *Global Disable of X-generation for Simulation* разрешает или запрещает перевод регистров в неопределенное состояние (X) при нарушении временных соотношений сигналов на его входах в процессе моделирования. При установке этого параметра в состояние «включено» в случае обнаружении временных нарушений регистры сохраняют предыдущее состояние в процессе моделирования. По умолчанию установлено значение «выключено», разрешающее перевод регистров в неопределенное состояние (X) при нарушении временных ограничений для входных сигналов.

Выполненные изменения значений для параметров инициализации программы *ModelSim* и временной модели вступают в силу после нажатия клавиши ОК в нижней части диалоговой панели (рис. 36).

**Временное моделирование проекта, реализуемого на базе ПЛИС семейств CPLD**

Запуск процесса временного моделирования проекта осуществляется двойным щелчком левой кнопки мыши на строке *Simulate Post-Fit VHDL Model* в окне процессов *Навигатора**проекта*. Если этапы синтеза, размещения и трассировки были проведены успешно, то далее автоматически создается командный файл сеанса временной верификации и производится запуск системы моделирования *ModelSim,* которая сразу же после инициализации исполняет этот пакетный файл.

При отсутствии окончательных результатов этапа реализации проекта (например, в случае их удаления с помощью команды *Cleanup Project Files* из всплывающего меню *Project,* или, если размещение и трассировка не проводились) программные средства пакета WebPACK ISE автоматически выполнят все необходимые процедуры, после чего будет активизирован непосредственно процесс временного моделирования. Для проекта, в котором правильно учтены все временные факторы (выбран кристалл с необходимым быстродействием, заданы и выполнены соответствующие ограничения), результаты временного моделирования отличаются от функционального только наличием задержек между входными и выходными сигналами. В качестве примера, на рис. 37, приведены временные диаграммы, полученные в результате полного моделирования двоично-десятичного дешифратора.

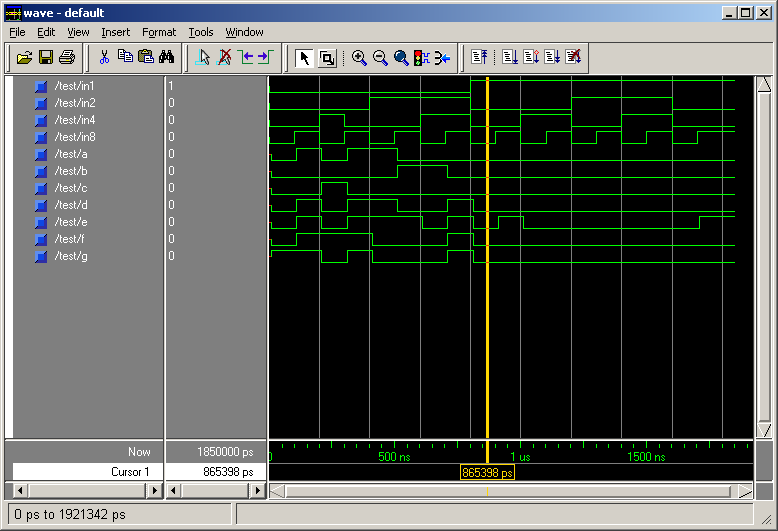


Рис. 37. Результаты временного моделирования проекта

двоично-десятичного дешифратора

Если результаты временной верификации не удовлетворяют предъявляемым требованиям, то следует проанализировать выполнение ограничений, установленных в проекте. Кроме того, рекомендуется обратить внимание на критерии оптимизации, которые были установлены для процессов синтеза и реализации *(Implementation)* проекта.

При необходимости следует внести коррективы во временные и топологические ограничения проекта и/или изменить критерии оптимизации, которая выполняется на этапах синтеза и реализации. После этого нужно повторить все необходимые этапы проектирования, включая полное временное моделирование.

Если указанные действия не приносят желаемого результата, то следует выбрать кристалл с более высоким быстродействием. С этой целью рекомендуется выполнить несколько сеансов временного моделирования, при различных значениях параметра *Change Device Speed to.*

**Задачи и порядок проведения работы:**

1. Ознакомиться с пользовательским интерфейсом пакета ModelSim 5.8;

2. Создать тестовый файл для моделирования схемы семисегментного дешифратора в виде временных диаграмм;

3. Провести функциональное и временное моделирование схемы семисегментного дешифратора с использованием программы ModelSim 5.8;

4. Оценить правильность функционирования спроектированного устройства по результатам моделирования.

**Форма отчётности студентов:**

индивидуальный отчёт в электронной форме с типовым титульным листом в формате MS Word с последующей распечаткой на бумажном носителе.

**Содержание отчёта:**

1. Изображение тестовых сигналов для моделирования семисегментного дешифратора в виде временных диаграмм;

2. Результаты функционального и временного моделирования схемы;

3. Оценка результатов моделирования;

4. Выводы в развёрнутой форме с соответствующими пояснениями.

Выполнение и защита лабораторной работы производится ***строго*** во время занятий. Защита работы – демонстрация работоспособности спроектированного цифрового устройства, собеседование с преподавателем и выполнение индивидуальных заданий.

**Вопросы для самоконтроля и подготовки к защите лабораторной работы № 3**

1. Опишите последовательность действий при создании тестового модуля;

2. С какой целью используется пакет ModelSim ?

3. Опишите процесс задания тестовых входных сигналов в программе ModelSim;

4. Опишите процедуру оценки правильности функционирования проектируемого устройства по результатам моделирования.

**ЛАБОРАТОРНАЯ РАБОТА № 4. РЕАЛИЗАЦИЯ ЦИФРОВЫХ УСТРОЙСТВ НА ПРОСТЫХ ЛОГИЧЕСКИХ ЭЛЕМЕНТАХ**

**Покрытие комбинационных схем двухвходовыми логическими элементами**

На этапе детализации синтезируемая схема делится до уровня двухвходовых блоков без учёта логического базиса, представляющего собой совокупность логических элементов, с использованием которых можно реализовать любое комбинационное устройство. Библиотека символов Symbol программного пакета WebPACK ISE включает в себя все возможные двухвходовые логические элементы (раздел Logic), которые можно разделить на три группы (типа), в зависимости от количества единиц, содержащихся в их числовых последовательностях.

**1.** Э***лементы типа «И»*** – содержат в собственных числовых последовательностях одну единицу:

– функция «И», числовая последовательность 0001;



и – функции «[ЗАПРЕТ](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B)» числовые последовательности 0010 и 0100 соответственно;



– функция «[СТРЕЛКА ПИРСА](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B5%D0%BB%D0%BA%D0%B0_%D0%9F%D0%B8%D1%80%D1%81%D0%B0)» («ИЛИ-НЕ»), числовая последовательность 1000.



**2.** ***Элементы (функции) типа «М2»*** – содержат в собственных числовых последовательностях две единицы:

(или ) – функция «[НЕРАВНОЗНАЧНОСТЬ](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B)» («НЕЭКВИВАЛЕНТНОСТЬ», «ИСКЛЮЧАЮЩЕЕ ИЛИ»), числовая последовательность 0110;



– функция «[РАВНОЗНАЧНОСТЬ](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%B2%D0%B8%D0%B2%D0%B0%D0%BB%D0%B5%D0%BD%D1%86%D0%B8%D1%8F)» («ЭКВИВАЛЕНТНОСТЬ», «ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ»), числовая последовательность 1001.



**3.** ***Элементы (функции) типа «ИЛИ»*** – содержат в собственных числовых последовательностях три единицы:

– функция «ИЛИ», числовая последовательность 0111;



и – функции «[ИМПЛИКАЦИЯ](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F)» («СЛЕДОВАНИЕ»), числовые последовательности 1011 и 1101 соответственно;



– функция «[ШТРИХ ШЕФФЕРА](https://ru.wikipedia.org/wiki/%D0%A8%D1%82%D1%80%D0%B8%D1%85_%D0%A8%D0%B5%D1%84%D1%84%D0%B5%D1%80%D0%B0)» («И-НЕ»), числовая последовательность 1110.



Структура обозначения логических блоков пакета WebPACK ISE включает следующие элементы:

– название реализуемой логической функции:

**AND** – логическая функция «И»,

**NAND** – логическая функция «И-НЕ»,

**OR** – логическая функция «ИЛИ»,

**NOR** – логическая функция «ИЛИ-НЕ»,

**XOR** – логическая функция «ИСКЛЮЧАЮЩЕЕ ИЛИ»,

**XNOR** – логическая функция «ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ».

– цифра, стоящая после названия логической функции обозначает количество входов логического элемента – от 2 до 9;

– буква B с последующей цифрой обозначает количество инвертированных входов рассматриваемого логического элемента – от B1 до B5.

Например, логический элемент **NOR5B4**, представленный на рис. 38, выполняет логическую функцию «ИЛИ-НЕ» над пятью входными переменными, четыре из которых (I0 – I3) предварительно инвертируются.

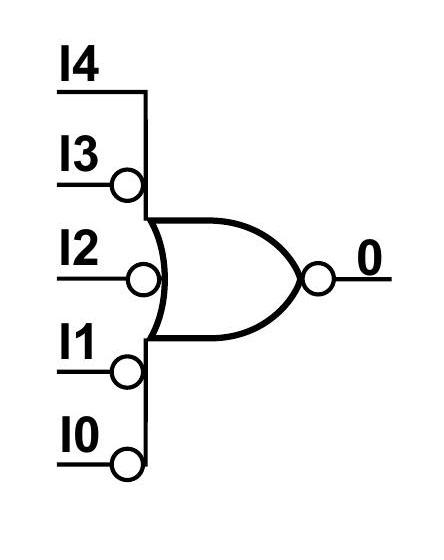


Рис. 38. Логический элемент NOR5B4

Для взаимного преобразования элементов и их числовых последовательностей можно использовать следующие алгоритмы:

1) инвертирование выходной числовой последовательности логических элементов позволяет из элементов типа «И» получить элементы типа «ИЛИ»;

2) инвертирование входной числовой последовательности с весом приводит к взаимной перестановке групп из символов в числовой последовательности элемента. При одновременном инвертировании двух и более входных числовых последовательностей применяется принцип суперпозиции – перестановки символов в выходной последовательности производятся группами в соответствии с весовыми коэффициентами инвертированных входов.



Покрытие производится от выхода к входу схемы. При полном совпадении числовых последовательностей блока и элемента производится замещение блока на элемент. В противном случае необходимо добавить инверторы на входах или выходе логического элемента для обеспечения совпадения с числовой последовательностью замещаемого блока.

В результате выполнения описанных выше процедур получается схема, состоящая из заданных логических элементов.

**Задачи и порядок проведения работы:**

1. Ознакомиться с логическими элементами, входящими в состав библиотеки символов программного пакета WebPACK ISE;

2. Реализовать полученные в домашней работе № 1 комбинационные схемы на двухвходовых логических элементах из библиотеки символов программного пакета WebPACK ISE;

3. Реализовать полученные в домашней работе № 2 комбинационные схемы на двухвходовых элементах заданного логического базиса («2И-НЕ» и «2ИЛИ-НЕ») из библиотеки символов программного пакета WebPACK ISE;

4. Провести проверку работы синтезированных схем на лабораторном стенде ЛСЦ-003 с подключением к их выходам схемы дешифратора семисегментного индикатора.

**Форма отчётности студентов:**

индивидуальный отчёт в электронной форме с типовым титульным листом в формате MS Word с последующей распечаткой на бумажном носителе.

**Содержание отчёта:**

1. Схемы синтезированных комбинационных устройств, выполненные с использованием схемотехнического редактора пакета WebPACK ISE;

2. Результаты испытания синтезированных схем на лабораторном стенде ЛСЦ-003;

3. Выводы в развёрнутой форме с соответствующими пояснениями.

Выполнение и защита лабораторной работы производится ***строго*** во время занятий. Защита работы – демонстрация работоспособности спроектированного цифрового устройства, собеседование с преподавателем и выполнение индивидуальных заданий.

**Вопросы для самоконтроля и подготовки к защите лабораторной работы № 4**

1. Опишите структуру обозначения логических элементов, входящих в состав пакета WebPACK ISE;

2. Какую логическую функцию реализует элемент NOR2B2, приведите его числовую последовательность и эквивалентный элемент из библиотеки пакета WebPACK ISE;

3. Какую логическую функцию реализует элемент NAND2B2, приведите его числовую последовательность и эквивалентный элемент из библиотеки пакета WebPACK ISE;

4. Какую логическую функцию реализует элемент XOR2, приведите его числовую последовательность;

5. Какую логическую функцию реализует элемент XNOR2, приведите его числовую последовательность.

**ЛАБОРАТОРНАЯ РАБОТА № 5. ОПИСАНИЕ ЦИФРОВЫХ УСТРОЙСТВ В СРЕДЕ ПАКЕТА WEBPACK ISE НА ЯЗЫКЕ VHDL**

**Язык описания схем VHDL**

В середине 80-х годов Министерство обороны США (U.S. Department of Defense, DoD) и Институт инженеров по электротехнике и электронике (Institute of Electrical and Electronic Engineers, IEEE) поддержали разработку довольно мощного языка описания схем VHDL. С самого начала и по настоящее время отличительными особенностями этого языка является следующее:

– проектируемые устройства можно иерархически разбивать на составные элементы;

– каждый элемент устройства имеет ясно очерченный интерфейс (для соединения его с другими элементами) и точное функциональное описание (для его моделирования);

– функциональное описание может быть основано на алгоритме, либо на реальной конструкции, которыми определяется работа элемента. Например, первоначально можно описать работу элемента посредством алгоритма, и это сделает возможной верификацию элементов более высокого уровня, в которых используется данный элемент; позднее алгоритмическое определение можно заменить структурной схемой;

– все можно моделировать: параллелизм, временные соотношения и синхронизацию тактовыми сигналами. На языке VHDL можно описать как асинхронные, так и синхронные последовательностые структуры;

– можно моделировать выполняемые устройством в целом логические действия и его временные характеристики.

Таким образом, с самого начала VHDL является языком документации и моделирования, позволяющим точно задавать и имитировать поведение цифровых систем.

**Структура программы на языке VHDL**

При создании языка VHDL имелось в виду воплотить в нем принципы структурного программирования с заимствованием идей у языков программирования Паскаль и Ада. Ключевая идея состояла в том, чтобы задать интерфейс схемного модуля, а его внутреннее устройство скрыть.

Основное описание и организация структуры VHDL показаны на рис. 39.

| **Package** – пакет (описание функций и процедур, необязательно) |
| --- |
| **Entity** – объект (описание интерфейса) |
| **Architecture** –архитектура (описание модели)  Описание функций и процедур |
| **Signal** **declaration** – описание сигналов  *параллельные операторы*   * Операторы экземпляра компонента * Операторы условного выбора * Операторы выбора и присваивания * Операторы генерации * Операторы процесса   **Variable** **declaration** – описание переменных  *последовательные операторы*   * Присвоение сигналов * Присвоение переменных * Вызов процедур, функций * Операторы **if, case, loop, next, exit, return** * Оператор ожидания **wait** |

Рис. 39. Структура проекта на VHDL

Оператор пакета является необязательным для раздела описаний. Оператор **entity** содержит описание входного и выходного интерфейса проекта, так как каждая микросхема имеет выходы или входы. Оператор **architecture** содержит описание функционирования проекта. Проект может содержать любое количество операторов пакетов, объектов и архитектур. Из структурной схемы языка видно, что архитектура содержит параллельные операторы, такие как **netlists*,*** и классические последовательные операторы. Параллельные операторы выполняются независимо от порядка, в котором они написаны. Параметры, передаваемые по значению, передаются между операторами с помощью сигналов, присвоение сигнала осуществляется драйвером.

Наиболее мощные конструкции языка встречаются внутри последовательных операторов. Такие конструкции размещаются вместо параллельных операторов, например, оператора процесса, функций и процедур. Последовательные операторы похожи на операторы языков программирования, они выполняются в том же порядке, в котором пишутся. Значения содержатся в переменных или константах. Сигналы используются для передачи входных и выходных значений процесса или от других параллельных операторов.

Как и в других языках программирования, в языке VHDL пробелы и переходы с одной строки на другую в общем случае игнорируются, и для удобства чтения их можно вставлять как угодно. *Комментарии***(comments)**начинаются с двух дефисов (--) и заканчиваются концом строки.

На рис. 40 в качестве примера приведена очень простая программа на языке VHDL для двухвходового вентиля «ЗАПРЕТ». В больших проектах объекты и архитектуры иногда бывают помещены в отдельные файлы, связь между которыми компилятор обнаруживает по их объявленным именам

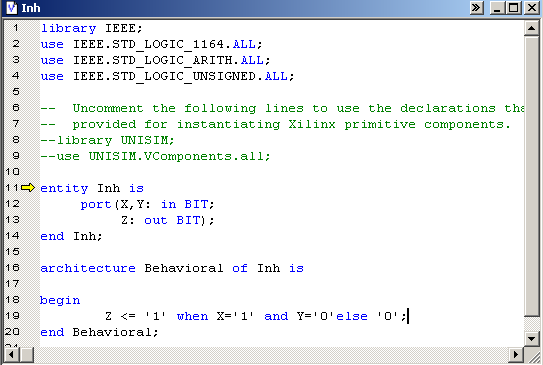


Рис. 40. Программы на языке VHDL для вентиля «ЗАПРЕТ»

В языке VHDL определено много специальных строк символов, называемых *зарезервированными**словами***(reserved words)**или *ключевыми**словами***(keywords)**.В приведенном примере имеется несколько ключевых слов: **entity,port, is, in, out, end, architecture, begin, when, else и not**. Определяемые пользователем *идентификаторы***(identifiers)**начинаются с буквы и содержат буквы, цифры и подчеркивания. (Символ подчеркивания не может следовать за другим символом подчеркивания и не может быть последним символом идентификатора). В данном примере идентификаторами являются **Inh, X, Y, BIT, Z и Behavioral. «BIT»** – это встроенный идентификатор предопределенного типа; он не считается зарезервированным словом, так как его можно переопределять. Зарезервированные слова и идентификаторы не чувствительны к регистру.

**Описание интерфейса проектируемой схемы**

В языке VHDL логическая схема представляется как объект проекта. Представляемая логическая схема может быть достаточно сложной, как, например, микропроцессор или весьма простой, как логический вентиль «2И-НЕ». Каждый объект проекта в свою очередь имеет два различных типа описаний: описание интерфейса и одно или более архитектурные тела.

В табл. 2 представлен синтаксис объявления объекта. Целью объявления объекта, помимо присвоения объекту имени, является определение сигналов внешнего интерфейса или *портов***(ports)**в части объявления объекта, которая называется *объявлением портов***(port declaration).**Кроме ключевых слов **entity, is, port и end,** объявление объекта содержит следующие элементы:

**Entity\_name** – выбираемое пользователем имя объекта;

**Signal\_names** – список выбираемых пользователем имен сигналов внешнего интерфейса, состоящий из одного имени или из большего числа имен, разделенных запятой;

**Mode** – одно из четырех зарезервированных слов, определяющих направление передачи сигнала:

а) **in** сигнал на входе объекта;

б) **out** сигнал на выходе объекта; заметьте, что значение такого сигнала нельзя «прочитать» внутри структуры объекта; он доступен только объектам, использующим данный объект;

в) **buffer** сигнал на выходе объекта, такой, что его значение можно читать также внутри структуры данного объекта;

г) **inout** сигнал, который может быть входным или выходным для данного объекта; обычно этот режим используется применительно к входам-выходам ПЛУ с тремя состояниями;

**Signal*\_*type** – встроенный или определенный пользователем тип сигнала.

**Замечание**. Обратите внимание, что после заключительного **signal*\_* type** нет точки с запятой; изменение порядка следования закрывающей скобки и точки с запятой после нее типичная синтаксическая ошибка программиста, начинающего писать на языке VHDL.

Таблица 2 – Синтаксис объявления объекта на языке VHDL

| **entity** | *entity\_name* **is** |
| --- | --- |
| **port** | (*signal\_names:* **mode** *signal\_type;* |
|  | *signal\_names:* **mode** *signal\_type;* |
|  | ………………………………….. |
|  | *signal\_names:* **mode** *signal\_type);* |
| **end** | *entity\_name* |

Порты объекта, а также направление передачи и типы сигналов – это все, что видят другие модули, использующие данный модуль.

Проиллюстрируем это понятие на примере. Описание интерфейса для схемы дешифратора, которая преобразует входной четырехразрядный код в напряжение низкого уровня на одном из 10 выходов.

**entity** Id\_1 is

**port**(A: **in** bit\_vector(0 to 3); Y: **out** bit\_vector(0 to 9));

**end** Id\_1;

**Архитектурные тела в языке VHDL**

Описание интерфейса по существу определяет только входы-выходы объекта проекта. Кроме этого необходимо иметь средства описания поведения объекта. В языке VHDL для этой цели служит так называемое архитектурное тело. Это тело может определять поведение объекта непосредственно или представлять структурную декомпозицию на более простые компоненты.

В табл. 3 представлен синтаксис объявления архитектурного тела.

Таблица 3 – Синтаксис определения архитектуры на языке VHDL

| **architecture** | *Architecture\_name* **of** *entity\_name* **is** |
| --- | --- |
|  | *type declarations* |
|  | *signal declarations* |
|  | *constant declarations* |
|  | *function definitions* |
|  | *procedure definitions* |
|  | *component declarations* |
| **begin** |  |
|  | *Concurrent\_statement* |
|  | *…………………………* |
|  | *Concurrent\_statement* |
| **end** | *Architecture\_name;* |

**Имя****объекта(entity***\_***name)**в этом определении должно быть таким же, какое раньше было присвоено объекту в объявлении объекта. *Имя архитектуры***(architecture\_name)**–это выбираемый пользователем идентификатор, обычно так или иначе связанный с именем объекта; при желании имя архитектуры может быть тем же самым, что и имя объекта.

Сигналы внешнего интерфейса архитектуры (порты) наследуются от той части объявления соответствующего объекта, где объявляются порты. У архитектуры могут быть также сигналы и другие объявления, являющиеся для нее локальными, подобно тому, как это имеет место в других языках высокого уровня. В отдельном «пакете», используемом несколькими объектами, можно сделать объявления, общие для этих объектов.

Объявления в табл. 3 могут располагаться в произвольном порядке. Начать легче всего с *объявления сигнала***(signal declaration),** которое сообщает ту же самую информацию о сигнале, какую содержит объявление порта, за исключением того, что вид сигнала не задается:

**signal** *signal\_names : signal\_type;*

В архитектуре может быть объявлено любое число сигналов, начиная с нуля, и они приблизительно соответствуют поименованным соединениям в принципиальной схеме. Их можно считывать и записывать внутри определения архитектуры и, подобно другим локальным элементам, на них можно ссылаться только в пределах данного определении архитектуры.

*Переменные***(variables)**в языке VHDL похожи на сигналы, за исключением того, что, как правило, они не имеют никакого физического смысла в схеме. Переменные используются в функциях, процедурах и процессах языка VHDL. Объявление переменной выглядит следующим образом:

**variable** *variable\_names : variable\_type;*

Ниже, в качестве примера, показано архитектурное тело для дешифратора **Id\_1:**

**Architecture** DC **of** Id\_1 **is**

**Begin**

**Process(A)**

**Begin** Y<=”1111111111”

**Case** A(0 **to** 3) **is**

**when(**“0000”**)**  => Y <=”0111111111”

**when(**“1000”**)**  => Y <=”1011111111”

**when(**“0100”**)**  => Y <=”1101111111”

**when(**“1100”**)**  => Y <=”1110111111”

**when(**“0010”**)**  => Y <=”1111011111”

**when(**“1010”**)**  => Y <=”1111101111”

**when(**“0110”**)**  => Y <=”1111110111”

**when(**“1110”**)**  => Y <=”1111111011”

**when(**“0001”**)**  => Y <=”1111111101”

**when(**“1001”**)**  => Y <=”1111111110”

**end** case;

**end** process;

**end** DC;

В архитектурном теле отсутствует программный цикл, осуществляющий сканирование входов от A(0) до A(3), вместо этого применен оператор выбора **case**, в котором управляющее выражение – битовый вектор. В зависимости от значений входного вектора A, определяется значение выходного вектора Y. Нумерация разрядов в векторе осуществляется слева направо.

**Операторы блоков в языке VHDL**

Базовым элементом описания в языке VHDL служит блок, т.е. ограниченный фрагмент текста, содержащий раздел описания и исполняемый раздел. Следовательно, архитектурное тело представляет собой блок. Однако в рамках архитектурного тела могут существовать внутренние блоки.

Рассмотрим пример. Блоки А и В вложены во внешний блок архитектурного тела. Возможно любое число уровней вложенности. Использование такой структуры позволяет иерархически представить структуру объекта и записать условия «защиты». Если условие будет иметь значение «истина», это разрешит выполнение определенных типов внутренних операторов.

Наличие метки перед оператором блока обязательно. Имя метки заканчивается двоеточием и может стоять перед оператором блока или на предыдущей строке. Наличие других операторов между оператором блока и меткой не допускается:

**architecture** Block **of** System **is**

**--------** раздел описаний типов внешнего блока

**begin**

**--------** выполняемые операторы внешнего блока

**A:block**

**--------** раздел описаний типов внутреннего блока А

**begin**

**--------** выполняемые операторы внутреннего оператора А

**end** block A;

**B:block**

**--------** раздел описаний типов внутреннего блока В

**begin**

**--------** выполняемые операторы внутреннего оператора B

**end** block B;

**end** Block;

**Процессы в языке VHDL**

Наиболее значимым элементом для моделирования на языке VHDL является оператор процесса. Оператор содержит последовательные операторы и позволяет разработчику описывать схемы на поведенческом уровне абстракции. Например:

**process(Insig)**

**variable v1:** Integer; -- описание переменных

**begin**

**v1:=**Insig; -- присвоение переменной

**v1:=**function\_name(v1+1); -- Вызов функции

**end** process;

При проектировании аппаратных средств оператор процесса применяется в двух случаях: для комбинационных и последовательностных схем. Для комбинационных схем оператор выглядит так:

**process(**signal1, signal2, …,signalN**)**

**begin**

**…**

**end** process;

А для последовательностной схемы – несколько иначе:

**process(**X\_signal**)**

**begin**

**if** X\_signal and X\_signal’evtent **then**

**…**

**end if;**

**end** process;

Для комбинационных схем определяется весь список вводимых в процесс сигналов. Это так называемый список чувствительности или сигналов запуска процесса. Если какой-либо сигнал из этого списка меняет свое значение, то процесс активизируется и выполняются операторы блока этого процесса.

Для поледовательностных схем список чувствительности, включает в себя операторы времени и условия запуска с атрибутами. Если этого списка нет, то должен быть оператор времени **wait**

**Типы, переменные и константы в языке VHDL**

Каждому сигналу, переменной и константе в программе на языке VHDL необходимо поставить в соответствие *тип***(type)**.Типом определяется множество или диапазон значений, которые может принимать данный элемент, и обычно имеется набор операторов (таких как сложение, логическое «И» и т.д.), связываемых с данным типом.

В языке VHDL есть всего лишь несколько *предопределенных типов***(predefined types);**они перечислены в табл. 4.

Таблица 4 – Предопределенные типы языка VHDL

| **bit** | **character** | **severity\_level** |
| --- | --- | --- |
| **bit\_vector** | **integer** | **string** |
| **boolean** | **real** | **time** |

Типом **integer**определяется диапазон значений целых чисел от **– 2147483647** до **+ 2147483647** (от –231 до +231–1). В некоторых реализациях языка VHDL этот диапазон может быть и шире. Типом **boolean**предусматриваются два значения: **true** и **false***.* Тип **character**содержит все символы 8-битового набора ISO, из которых первые 128 являются символами стандарта ASCII. Встроенные операторы для типов **integer** и **boolean** приведены в табл. 5.

Таблица 5 – Предопределенные операторы для типов

**integer** и **boolean** в языке VHDL

| **+** | Сложение | **and** | И |
| --- | --- | --- | --- |
| **–** | Вычитание | **or** | ИЛИ |
| \* | Умножение | **nand** | И-НЕ |
| **/** | Деление | **nor** | ИЛИ-НЕ |
| **mod** | Деление по модулю | **xor** | Исключающее ИЛИ |
| **rem** | Остаток от деления по модулю | **xnor** | Исключающее ИЛИ-НЕ |
| **abs** | Абсолютное значение | **not** | Дополнение (инверсия) |
| \*\* | Возведение в степень |  |  |

Чаще всего в типичных программах на языке VHDL используются определяемые пользователем типы **(userdefined types),** а из них самыми употребительными являются перечислимые типы **(enumerated types)**, которые определяются путем перечисления их значений. Предопределяемые типы **boolean** и **character** – это перечислимые типы. Формат объявления типа в случае перечислимого типа указан в первой строке табл. 6. Здесь **value\_list** представляет собой список (перечисление) всех возможных значений этого типа, разделяемых запятыми. Значениями могут быть определяемые пользователем идентификаторы или символы (где под «символом» понимается символ ISO, заключенный в одинарные кавычки). Идентификаторы чаще всего применяются для обозначения альтернатив или состояний конечного автомата, например:

**type** *traffic\_light\_state* **is** (*reset, stop, wait, go*);

Символы используются в очень важном случае стандартного определяемого пользователем логического типа **std\_logic**(см. табл. 7), являющегося частью стандартного пакета IEEE 1164. Этот тип включает не только ‘0’ и ‘1’, но также и семь других значений, которые оказываются полезными при моделировании логического сигнала (бита) в реальной логической схеме.

Таблица 6 – Синтаксис объявления типов и констант в языке VHDL

| **type** | *Type\_name* **is** *{value\_list);* |
| --- | --- |
| **subtype** | *Subtype\_name* **is** *type\_name start* **to** *end;* |
| **subtype** | *Subtype\_name* **is** *type\_name start* **downto** *end;* |
| **constant** | *Constant\_name: type\_name* := *value;* |

Таблица 7 – Определение типа STD\_LOGIC в языке VHDL

| **type** STD\_ULOGIC **is** | (‘**U’**,--не инициализировано |
| --- | --- |
|  | ‘**X’**,--сильное неизвестное |
|  | ‘**0’**, --сильный ноль |
|  | ‘**1’**, --сильная единица |
|  | ‘**Z’**, --высокий импеданс |
|  | ‘**W’**,--слабое неизвестное |
|  | ‘**L’**,--слабый ноль |
|  | ‘**H’**,--слабая единица |
|  | **‘-**’,--безразлично ); |
| **subtype** STD\_LQGIC **is resolved** STD\_ULOGIC; | |

Язык VHDL позволяет пользователю создавать также *подтипы***(subtypes)**согласно синтаксису, указанному в табл. 6. Значения подтипа должны быть слитным подмножеством значений, предусмотренных основным типом, начиная со **start**и кончая **end***.* Для перечислимого типа «слитность» означает расположение на соседних позициях в исходном списке значений **value*\_*list***.* Вот несколько примеров определения подтипов:

**subtype** *twoval\_logic* **is** *std\_logic* **range** ‘0’ **to** ‘1’;

**subtype** *fourval*\_*logic* **is** *std\_logic* **range** ‘X’ **to** ‘Z’;

**subtype** *negint* **is** *integer* **range** -2147483647 **to** -1;

**subtype** *bitnum* **is** *integer* **range** 31 **downto** 0;

Заметьте, что порядок следования значений в указываемом диапазоне может быть в сторону возрастания или в сторону убывания в зависимости от того, какое из ключевых слов **to**или **downto**употреблено.

В языке VHDL есть два предопределенных подтипа integer:

**subtype** *natural* **is** *integer* **range** 0 **to** *highest\_integer;*

**subtype** *positive* **is** *integer* **range** 1 **to** *highest\_integer;*

*Переменная***(variable)** в языке VHDL отслеживает состояние процесса только внутри него и вне процесса ее не видно. Синтаксис определения переменной внутри процесса подобен синтаксису объявления сигнала в архитектуре, за исключением того, что используется ключевое слово **variable***:*

**variable** *variable-names : variable-type;*

**variable** RUN: Boolean :=False;

**variable** COUNT: Integer :=0;

**variable** ADDR: bit\_vector(0 **to** 11);

*Константы***(constants)** способствуют удобству чтения программ, возможности их поддержания и сопровождения, а также переносу на какой-либо другой язык. Синтаксис *объявления констант***(constant declaration)**в языке VHDL указан в последней строке в табл. 6; его можно проиллюстрировать следующими примерами:

**constant** BUS\_SIZE: *integer* := 32;

**constant** MSB: *integer* := BUS\_SIZE-1;

**constant** Z: *character* := ‘Z’;

Обратите внимание, что значение константы может быть задано простым выражением. Константы можно использовать повсюду, где встречаются соответствующие значения.

Другую очень важную группу определяемых пользователем типов образуют *типы массивов***(array types)**.Как и в других языках, в языке VHDL *массив***(array),**по определению, – это упорядоченный набор элементов одного и того же типа, отдельные компоненты которого выбираются с помощью *индекса массива***(array index).**Возможны несколько вариантов синтаксиса объявления массива в языке VHDL; они представлены в табл. 8. В первых двух вариантах **start**и **end**являются целыми числами, которыми задается возможный диапазон изменения индекса массива и, следовательно, полное число элементов массива. В последних трех вариантах диапазоном изменения индекса массива являются все значения указанного типа **(range-type)**или подмножество этих значений.

Таблица 8 – Синтаксис объявления массивов в языке VHDL

| **type** *type\_name* **is** **array** *(start* **to** *end)* **of** *element\_type;* |
| --- |
| **type** *type\_name* **is** **array** *(start* **downto** *end)* **of** *element\_type;* |
| **type** *type\_name* **is** **array** *(range\_type)* **of** *element\_type;* |
| **type** *type\_name* **is** **array** *(range\_type* **range** *start* **to** ***end****)* **of** *element\_type;* |
| **type** *type\_name* **is** **array** *(range\_type* **range** *start* **downto** *end)* **of** *element\_type;* |

В табл. 9 приведены примеры объявления массивов. Первые два примера совсем обычны и демонстрируют задание диапазона изменения индекса в сторону возрастания и в сторону убывания. Следующий пример показывает, как можно воспользоваться константой WORD\_LEN при объявлении массива. Отсюда видно также, что границу диапазона можно задать простым выражением. Из третьего примера следует, что сам элемент массива может быть массивом; таким образом создается двумерный массив. Последний пример показывает, что множество возможных значений элементов массива можно задать, указав перечислимый тип (или подтип); в этом примере массив состоит из четырех элементов согласно данному нами чуть раньше определению типа **traffic*\_*light*\_*state*.***

Таблица 9 – Примеры объявления массивов в языке VHDL

| **type** *monthly\_count* **is** **array** (1 **to** 12) **of** *integer*;  **type** *byte* **is** **array** (7 **downto** 0) **of** STD\_L0GIC;  **constant** WORD \_LEN: *integer* := 32;  **type** *word* **is** **array** (WORD\_LEN-1 **downto** 0) **of** STD\_L0GIC;  **constant** NUM\_REGS: *integer* := 8;  **type** *reg*\_*file* **is** **array** (1 **to** NUM\_REGS) **of** *word*;  **type** *state\_count* **is** **array** (*traffic*\_*light*\_*state*) **of** *integer*; |
| --- |

Элементы массива считаются упорядоченными слева направо в том же направлении, в каком индекс пробегает свои значения. Таким образом, индексы самых левых элементов массивов типов **monthly\_count, byte, word, reg\_file**и**statecount** в табл. 9 равны 1, 7, 31, 1 и **reset** соответственно.

Обращение к отдельным элементам массивов в операторах программы на языке VHDL осуществляется путем указания имени массива и индекса элемента в круглых скобках. Если, например, М, В, W, R и S – сигналы или переменные тех пяти типов массивов, которые приведены в табл. 9, то любая из записей М(11), В(5), W(WORD\_LEN-5), R(0,0), R(0) и S (reset) является правильным указанием элемента.

*Массивы-литералы***(array literals)**можно задать, перечисляя в скобках значения элементов. Например, переменной **В** типа **byte** можно задать значение, состоящее из одних единиц, оператором

**В: = (‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’ ‘1’);**

В языке VHDL возможно и в более сжатой форме задавать значения, указывая индекс. Например, следующая запись обеспечивает присвоение единичных значений всем элементам переменной W типа word, за исключением младших разрядов каждого байта, которым присваиваются нулевые значения:

**W:= (0 => ‘0’, 8 => ‘0’, 16 => ‘0’, 24 => ’0’, others=>’1’);**

Рассмотренные правила справедливы при любом *типе элементов***(element-type)**,но литерал типа STD\_LOGIC легче всего записать в виде «строки». *Строкой***(string)**в языке VHDL является последовательность символов ISO, заключенная в двойные кавычки. Строка – это, конечно, массив символов; поэтому массиву типа STD\_LOGIC заданной длины можно присвоить значение, выраженное строкой той же длины, если только символы в строке принадлежат набору из девяти символов, которыми, по определению, исчерпываются возможные значения элементов типа STDLOGIC: ‘0’, ‘1’, ‘U’ и т.д. Таким образом, предыдущие два примера можно переписать в виде:

**В: = “11111111”;**

**W: = “11111110111111101111111011111110”;**

Можно также указывать подмножество непосредственно следующих один за другим элементов массива или, как говорят, *вырезку из массива***(array slice)**,задавая начальный и конечный индексы подмножества.

Наконец, массивы или элементы массивов можно объединять с помощью *оператора конкатенации &* **(concatenation operator)**,который соединяет массивы и элементы в том порядке, в каком они записаны слева направо. Например, запись ‘0’ & ‘1’ & “1Z” эквивалентна строке “011Z”, а выражение **В** (6 **downto** 0) & **В** (7) представляет собой циклический сдвиг 8-разрядного массива **В** на 1 разряд влево.

Самым важным типом массивов в типичной программе на языке VHDL является определяемый пользователем в соответствии со стандартом IEEE 1164 логический тип STD\_LOGIC\_VECTOR, которым задается упорядоченный набор элементов типа STD\_LOGIC.

Определение этого типа имеет вид:

**type** *std\_logic\_vector*  **is** **array** (*natural* **range** <>) **of**, *std\_logic*;

Это пример *типа массива**без ограничений***(unconstrained array type)**:диапазон возможных значений индекса массива не задан, за исключением того, что он должен быть подмножеством определенного типа, в данном случае – типа **natural**. Эта особенность языка VHDL позволяет записывать архитектуры, функции и другие элементы программ в более общем виде, до некоторой степени независимо от размеров массивов и диапазонов возможных значений индексов. Действительный диапазон значений индекса определяется в тот момент, когда сигналу или переменной ставится в соответствие этот тип.

**Атрибуты**

Атрибуты – это значения, связанные с поименованным элементом (константой, переменной, сигналом). Для построения моделей и моделирования особенно важную роль играют атрибуты. Есть несколько групп атрибутов, например, для типов, массивов, сигналов и пр.рассмотрим группу атрибутов, определенную для сигналов:

1) *S’last\_*value (прошлое значение S) – предыдущее значение, которое сигнал S имел непосредственно перед последним изменением S. Относится к тому же самому типу, что и S. Может исползоваться для проверки, не изменился ли сигнал (**if** S/=S’last\_value).

2) *S’stable(T)* – тип Boolean. Атрибут имеет истинное значение, если сигнал S стабилен в течение последних T единиц времени. Если T=0, атрибут записывается как S’stable.

3) *S’delayed(T)* есть значение, которое сигнал S имел на T единиц ранее. Относится к тому же самому типу, что и S.

4) *S’event,* тип Boolean, принимает истинное значение, если только что произошло изменение сигнала S.

Эти атрибуты полезны для обнаружения изменений сигналов и детального времени моделирования.

**Процедуры и функции на языке VHDL**

В языке VHDL описание функции осуществляется как в обычных языках программирования. Указывается имя функции и, если необходимо, входные параметры. Функция имеет оператор возврата с определенным типом возвращаемого аргумента.

Функция может быть описана, например, следующим образом:

**function** Is\_zero(n: Integer) **return** Boolean **is**

**--** описание типов, переменных, констант, подпрограмм

**begin**

**--** последовательные операторы

**if** n=0 **then** **return** True;

**else** **return** False;

**end if**;

**end;**

Возвращаемое значение может быть определено в виде выражения либо вычислено через соответствующую последовательность операторов.

Язык VHDL предусматривает также описание процедур. Процедура может иметь как внутренние, так и внешние параметры. В остальном описание процедуры аналогично описанию функции. Ниже показан пример описания процедуры:

**procedure** Count(Incr : Boolean; big : **out** Bit; num : **inout** Integer) **is**

**--** описание типов, переменных, констант, подпрограмм

**begin**

**if** Incr **then**

num := num+1;

**end** **if**;

**if** num > 101 **then**

big := ‘1’;

**else**

big := ‘0’;

**end if;**

**end**

В списке параметров указываются и входные и выходные параметры. За списком параметров следует раздел описания.

**Библиотеки и пакеты на языке VHDL**

*VHDL-библиотека* (**library**)– это место, где компилятор VHDL хранит информацию об отдельном варианте проекта, включая промежуточные файлы, используемые при анализе, моделировании и синтезе в рамках данной разработки. Место библиотеки в файловой системе компьютера зависит от реализации. Для очередного VHDL-проекта компилятор автоматически создает библиотеку под именем «work» и использует ее.

*VHDL-пакет***(package)**– это файл, содержащий определения элементов, которые могут быть использованы другими программами. В пакет можно включить элементы такого рода, как сигнал, тип, константа, функция, процедура и объявления компонентов.

Помещенные в пакет сигналы являются «глобальными» и доступны любому VHDL-объекту, использующему этот пакет. Типы и константы, упомянутые в пакете, известны в любом файле, использующем этот пакет.

Проект может «использовать» тот или иной пакет, если в начало файла, относящегося к данному проекту, помещено *предложение use***(use clause).**Например, чтобы воспользоваться всеми определениями пакета, содержащего стандарт IEEE 1164, нам следует написать:

**use ieee.std\_logic\_1164.all;**

Здесь **“ieee”** – это имя библиотеки, ранее введенное предложением library. В этой библиотеке файл с именем **“std\_logic\_ll64”** содержит желаемые определения. Приставка **“all”** велит компилятору использовать все определения этого файла. Вместо **“all”** можно написать имя какого-то одного элемента, когда необходимо использовать только его определение, например:

**use ieee.std\_logic\_1164.std\_ulogic;**

**Операторы управления в языке VHDL**

Среди операторов управления, используемых в языке VHDL, можно выделить стандартные и специальные, характерные только для данного языка как, например, **wait.** Рассмотрим основные и наиболее часто используемые операторы управления.

Оператор **if**(если)

**if** increment **and not** decrement **then**

count := count +1;

**elsif not** increment **and** decrement **then**

count := count -1;

**elsif** increment **and** decrement **then**

count : =0;

**else**

count := count;

**end** **if;**

Условия в операторе должны быть типа Boolean. Возможно наличие любого числа фраз **elsif.** Фразы **elsif** и **else** являются необязательными.

Оператор **case** (выбор) проиллюстрируем несколькими примерами.

Пример 1.

**case** X(0 **to** 1) **is**

**when** “00” => Z <= 0;

**when** “01” => Z <= 1;

**when** “10” => Z <= **not** 2;

**when** “11” => Z <= Z;

**end case;**

Пример 2.

**case** N **is**

**when** 0 => Z <= 0;

**when** 1 => Z <= 1;

**when** 2 => Z <= not 2;

**when** 3 => Z <= 2;

**end case;**

Пример 3

**case** day **is**

**when** Saturday **to** Sunday =>

work := False;

work\_out := False;

**when** Monday | Wednesday | Friday =>

work := True;

work\_out := True;

**when** **others** =>

work := True;

work\_out := False;

**end** **case**;

Оператор выбора **case** осуществляет декодирование на основе значения управляющего выражения, а затем выполняет выбранный оператор (или группу операторов). В примере 1 управляющее выражение – битовый вектор, примере 2 – целое, в примере 3 – символьное. В общем случае в качестве управляющего выражения может использоваться любой дискретный тип, как в показанных примерах, либо одномерный массив символов.

В языке VHDL существует оператор назначения сигнала с выбором, позволяющий сделать более сжатую запись:

**with** X1&X2&X3 **select**

f <= ‘0’ **after 20** ns **when** “000”,

‘1’ **after** 20 ns **when** “001”,

‘0’ **after** 20 ns **when** “010”,

‘1’ **after** 20 ns **when** “011”,

‘0’ **after** 20 ns **when** “100”,

‘1’ **after** 20 ns **when** “101”,

‘1’ **after** 20 ns **when** “110”,

‘1’ **after** 20 ns **when** “111”,

Операторы цикла реализуют те же функции, что и во всех языках программирования. Рассмотрим примеры с использованием операторов управления **for, wile, loop, next, exit:**

Пример 1

**for** i **in** 0 **to** 3 **loop**

A(i) := 2\*\*I

**end** **loop**;

Пример 2

Sum := 0;

i := 1;

Sum\_Int: **while** i<=n **loop**

sum := Sum + 1;

i := i + 1;

end loop Sum\_int;

Пример 3

Sum := 0;

i := 0;

Sum\_Int: **while** i<=N **loop**

i := i + 1;

**next** Sum\_Int **when** i=3;

Sum := Sum +1;

**end** **loop** Sum\_Int;

Пример 4

Sum := 1;

loop

VAL(X);

exit when X<0;

Sum := Sum +X;

end loop;

Пример 1 иллюстрирует базовый цикл **for***,* а пример 2 – цикл **while** и использование метки цикла. Цикл примера 2 вычисляет сумму первых **n** целых. В примере 3 производится вычисление той же самой суммы, но без числа 3, поскольку итерация пропускается при помощи оператора **next**. Пример 4 показывает потенциально бесконечный цикл, который прибавляет к накопленной сумме значения, поступающие из подпрограммы. Выход из этого цикла будет происходить в случае, когда подпрограмма VAL(X) выдаст отрицательное значение.

Два последних оператора управления – **return**(возврат) и **wait** (ожидание). Оператор return используется в процедурах и функциях.

Оператор ожидания **wait** используется для приостановки процесса на определенный период времени или до момента наступления определенного события, например, **wait** **until** clk.

**wait** **on** X1,X2 -- продолжить работу, когда изменится X1 или X2

**wait** **until** (X3=0); -- продолжить работу, когда изменится X3 из 1 в 0

**wait** **for** 100 ns; -- продолжить работу через 100 нс.

**Задержки сигналов в языке VHDL**

В цифровых системах сигналы передаются не мгновенно, а с некоторой задержкой во времени. При моделировании на языке различают два типа задержек сигналов – инерционную и транспортную.

Для моделирования такого рода задержек в распространении сигнала и используется оператор инерционной задержки **after**. Например, запись Y <= X1&X2 **after** 20 ns, означает, что значение сигнала Y изменится через 20 нс после изменения одного из сигналов X.

Транспортная задержка предполагает, что все изменения на входе будут передаваться на выход независимо от времени существования входного сигнала. В этом ее главное отличие от инерционной, работающей по принципу фильтра. Фактически при быстром изменении сигнал на входе логической схемы будет обнаружено только то значение, которое будет присутствовать по истечении задержки. При транспортной задержке временная диаграмма «вход-выход» будет более полной, но со сдвигом по времени на значение задержки.

Оператор транспортной задержки записывается следующим образом:

Y <= **transport** X **after** 3 ns.

**Моделирование схемных соединений**

В языке VHDL модель соединений компонентов для передачи и приема представляется в виде архитектурного тела с описанием интерфейса компонентов. Рассмотрим пример системы, состоящей из двух компонентов А и В, соединенных локальной шиной Lab. Пусть локальная шина имеет восемь проводников. Система имеет 8 входных линий и 16 выходных.

Интерфейс компонентов А и В имеет следующий вид:

**entity** A **is**

**port** (Ax : **in** bit\_vector(0 **to** 7); Ay : **out** bit\_vector(0 **to** 7));

**end** A;

**entity** B **is**

**port** (Bx : **in** bit\_vector(0 **to** 7); By : **out** bit\_vector(0 **to** 15));

**end** B;

Описание системы включает ее интерфейс и архитектурное тело, в котором записывается интерфейс компонента и конкретизация связей между компонентами. Для рассматриваемой в примере системы можно записать:

**entity** System **is**

**port** (X : **in** bit\_vector(0 **to** 7); Y : **out** bit\_vector(0 **to** 7));

**end** System;

**architecture** System\_AB **of** System **is**

**signal** Lab : bit\_vector(0 **to** 7); -- описание линий связи

**component** A **is** -- описание компонента A

**port** (Ax : **in** bit\_vector(0 **to** 7); Ay : **out** bit\_vector(0 **to** 7));

**end** **component**;

**component** B **is** -- описание компонента В

**port** (Bx : **in** bit\_vector(0 **to** 7); By : **out** bit\_vector(0 **to** 15));

**end** **component**;

**begin**

---- выполнение процессов, блоков

A **port** **map**(X, Lab); -- конкретизация компонента системы

B **port** **map**(Lab, Y); -- конкретизация компонента системы

**End** System\_AB;

Таким образом, объединения компонентов в систему моделируется описанием интерфейсов и передачей соответствующих значений сигналов.

Если для соединения используются отдельные единичные линии, то соответствующие им переменные могут иметь битовый тип. Все дальнейшие действия аналогичны. Но следует заметить, что групповое представление сигналов, когда оно соответствует схеме, выгоднее, чем битовое.

В тексте программы операторы **port** **map** записываются внутри архитектурного тела за операторами процессов. Приведенная в примере запись оператора определяется позиционным сопоставлением элементов отображения портов в описании компонентов. Тоже самое можно сделать при помощи ключевого соответствия, например:

S1: A **port** **map**(Ay => Lab, Ax => X);

S2: B **port** **map**(By => Y, Bx => Lab);

**Пример описания дешифратора для семисегментного индикатора на языке VHDL**

Для создания нового проекта нажмите *File>New Project*.

В открывшейся диалоговой панели нужно определить исходные данные, необходимые для создания проекта:

1) название проекта (*Project Name*);

2) путь, в котором предполагается расположить проект (*Project Location*);

3) семейство ПЛИС, на базе которого разрабатывается устройство (*Device Family*) (xc9500 xl CPLDs);

4) тип кристалла (*Device*) (xc95288xl);

5) корпус (*Pacage*) (VQ144);

Затем создаем новый модуль исходного описания проекта. Для этого следует выбрать команду *New Source* из раздела *Project* основного меню. В открывшейся диалоговой панели, показанной на рис. 41.

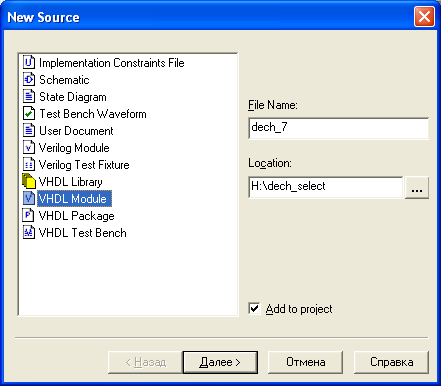


Рис. 41. Диалоговая панель установки параметров

нового исходного модуля проекта

Необходимо выбрать тип нового модуля, задать его имя и указать место расположения файла на диске. Для разработки принципиальной схемы следует выбрать тип создаваемого исходного модуля *VHDL Module*. Место расположения создаваемого модуля на диске указывается в поле редактирования *Location* диалоговой панели. По умолчанию предлагается рабочий каталог текущего проекта. Если флаг индикатора *Add to project* находится в установленном состоянии, то созданный модуль автоматически включается в состав текущего проекта. При создании *VHDL Module* можно указать имя архитектуры (*architecture\_name*), а также имя вашего объекта (*entity\_name)*

При открытии окна VHDL редактора (рис. 42) можно непосредственно приступать к написанию программы; он будет иметь следующий вид:

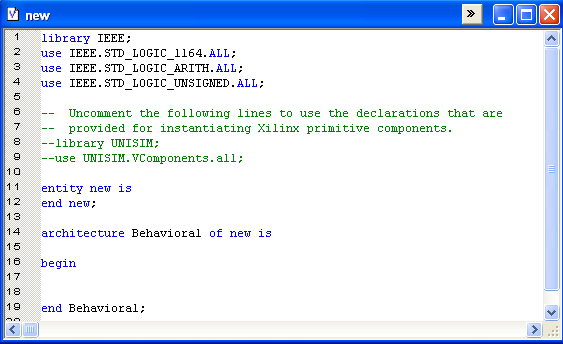


Рис. 42. Окно VHDL редактора

В только что созданном файле, как вы могли заметить, уже созданы пустые тела архитектуры и объекта, а также включены стандартные библиотеки. На рис. 43представлен один из вариантов текста программы, описывающей дешифратор семисегментного индикатора.

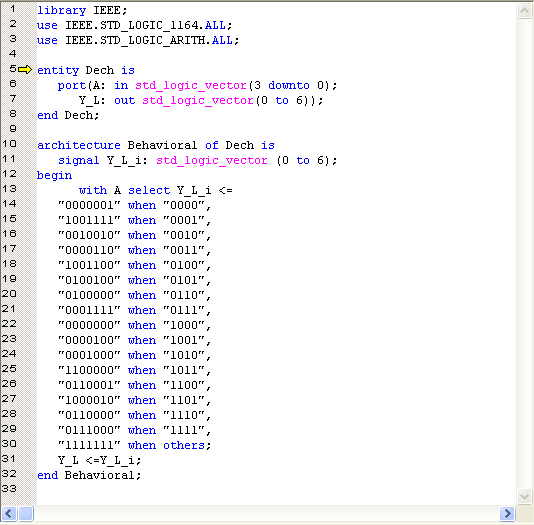


Рис. 43. VHDL-описание дешифратора семисегментного индикатора

После того как программа написана и сохранена, необходимо сформировать тестовый файл, и промоделировать дешифратор в программе ModelSim 5.8c.

На рис. 44 показан результат функционального моделирования схемы семисегментного дешифратора.

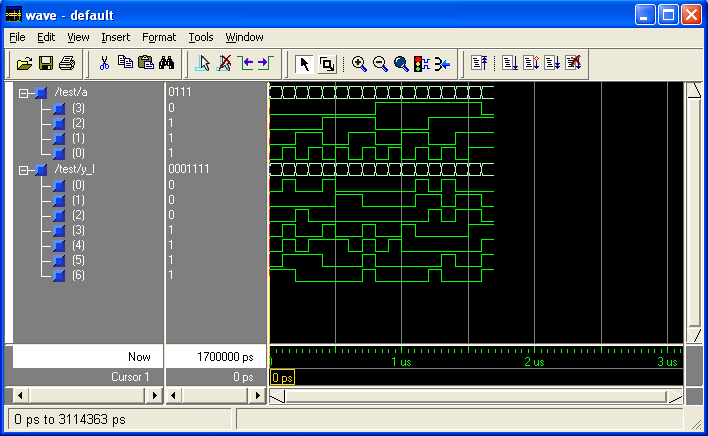


Рис. 44. Результат функционального моделирования схемы

семисегментного дешифратора.

**Задачи и порядок проведения работы**

1. Опишите несколькими способами заданное цифровое устройство на языке VHDL;

2. Проведите его моделирование в программе ModelSim;

3. Выполните реализацию и проверьте правильность функционирования спроектированного устройства на лабораторном стенде ЛСЦ-003.

Проектируемое устройство задано в виде числовой последовательности и представляет собой комбинационную логическую схему (кодопреобразователь) с 4-мя входами и 4-мя выходами. Варианты заданий указаны в табл. 10.

Таблица 10 – Варианты заданий для лабораторной работы № 5

| Номер вар. | Числовая последовательность кодопреобразователя |
| --- | --- |
| 1 | 8382 B8E1 EA6F 511D |
| 2 | 1612 71B4 B3AF C44D |
| 3 | 4341 74D2 D59F A22E |
| 4 | 1A12 B178 736F C88D |
| 5 | 4A42 E478 763F 988D |
| 6 | 1C50 1825 D35B E74C |
| 7 | 1A30 1843 B53D E72A |
| 8 | 4950 4825 D65E B719 |
| 9 | 2CA0 241A E3A7 DB8C |
| 10 | 85C0 812C DACB 7E45 |
| 11 | 8390 8249 BC9E 7D13 |
| 12 | 1650 1285 795B EED46 |
| 13 | 4360 4186 7C6D BE23 |
| 14 | 1A90 1249 B597 ED8A |
| 15 | 4AC0 421C E5C7 BD8A |

**Форма отчётности студентов:**

индивидуальный отчёт в электронной форме с типовым титульным листом в формате MS Word с последующей распечаткой на бумажном носителе.

**Содержание отчёта:**

1. Краткие теоретические сведения о структуре и способах описания цифровых устройств на языке VHDL;

2. Описание цифрового устройства, подготовленного для прошивки в ПЛИС;

3. Структурная схема, включающая синтезируемое цифровое устройство и дешифратор семисегментного индикатора;

4. Отчёт об испытаниях реализованного устройства на лабораторном стенде ЛСЦ-003 совместно с дешифратором семисегментного индикатора;

5. Выводы в развёрнутой форме с соответствующими пояснениями.

Выполнение и защита лабораторной работы производится ***строго*** во время занятий. Защита работы – демонстрация работоспособности спроектированного цифрового устройства, собеседование с преподавателем и выполнение индивидуальных заданий.

**Вопросы для самоконтроля и подготовки к защите лабораторной работы № 5**

1. Опишите основные элементы структуры проекта на языке VHDL;

2. Опишите последовательность действий при создании нового проекта;

3. Опишите синтаксис объявления объекта на языке VHDL, приведите примеры;

4. Опишите синтаксис объявления архитектуры на языке VHDL, приведите примеры;

5. Опишите типы, переменные и константы, используемые в языке VHDL, приведите примеры;

6. Опишите синтаксис объявления массивов на языке VHDL, приведите примеры;

7. Опишите синтаксис объявления процедур и функций на языке VHDL, приведите примеры;

8. Опишите основные операторы управления, используемые в языке VHDL, приведите примеры.

**ЛАБОРАТОРНАЯ РАБОТА № 6. ИССЛЕДОВАНИЕ ОСОБЕННОСТЕЙ СХЕМОТЕХНИЧЕСКОЙ РЕАЛИЗАЦИИ И РАБОТЫ ТИПОВЫХ КОМБИНАЦИОННЫХ УСТРОЙСТВ**

**Двоичные дешифраторы**

***Дешифраторы*** – это комбинационные устройства, имеющее *n* входов и *N* выходов. Дешифраторы относятся к преобразователям кодов. ***Двоичные дешифраторы*** преобразуют двоичный код в код «1 из N». При заданных значениях входных сигналов в состоянии «1» может находиться лишь один из выходов дешифратора. Если этот выход при поступлении на входы какой-либо комбинации значений входных сигналов принимает значение «1», то при поступлении на входы любой другой комбинации этот выход принимает значение «0».

Например, код «1 из N», содержащий 4 кодовых комбинации, может быть представлен следующим образом:

**1** 0 0 0

0 **1** 0 0

0 0 **1** 0

0 0 0 **1**

Из сказанного видно, что двоичный дешифратор, имеющий *n* входов, должен иметь выходов, соответствующих числу различных комбинаций в *n*-разрядном двоичном коде. Если часть входных наборов не используется, то дешифратор называют ***неполным****,* и у него число выходов меньше (десятичный дешифратор с четырьмя входами и десятью выходами).



Дешифратор характеризуется следующей системой собственных функций:



Здесь – это логические переменные, сопоставленные входам дешифратора; – переменные, сопоставленные выходам; – двоичные наборы входных переменных.



В условном обозначении дешифраторов проставляются буквы *DC* (от англ. *Decoder).* Входы дешифратора принято обозначать их двоичными весами. Кроме информационных входов дешифратор обычно имеет один или более входов разрешения работы обозначаемых как *EN* или *E* (*Enable*). На условном графическом изображении дешифратора два дополнительных поля (левое и правое) предназначены для размещения меток входов и выходов (рис. 45).

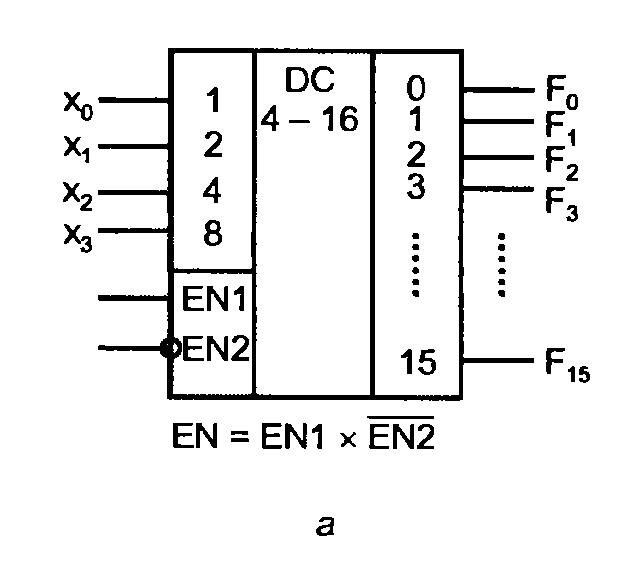


Рис. 45. Условное обозначение дешифратора в коде «1 из 16»

Схема рассматриваемого дешифратора состоит из *n*-входовых элементов «И». На входы дешифратора поступают как прямые, так и инверсные сигналы. Если же в нашем распоряжении имеются только прямые сигналы, то схему следует дополнить *n* инверторами (рис. 46).



Дешифраторы такого вида называются одноступенчатыми. Задержка прохождения логических сигналов через одноступенчатый дешифратор равняется задержке, приходящейся на один логический элемент. Малая задержка является причиной высокого быстродействия такой схемы. Недостатком одноступенчатого дешифратора является необходимость применения элементов «И» с большим числом входов, которое равно числу входов дешифратора. Многовходовые элементы «И» можно составить из элементов с небольшим числом входов, но при этом будет увеличена задержка прохождения сигналов через дешифратор.



Рис. 46. Схемотехническая реализация одноступенчатого дешифратора

При наличии входа разрешения *EN* и активном уровне сигнала на этом входе дешифратор работает описанным выше образом. При отсутствии разрешения (уровень сигнала на входе *EN* имеет неактивный (запрещающий) уровень) все выходы дешифратора пассивны. Если входов разрешения несколько, то сигнал разрешения работы образуется как конъюнкция сигналов отдельных входов. Часто дешифратор имеет инверсные выходы. В этом случае только один выход имеет нулевое значение, а все остальные – единичное. При запрещении работы дешифратора на всех его выходах будет присутствовать логическая единица.

Функционирование дешифратора со входом разрешения описывается системой конъюнкций:



**Наращивание размерности дешифраторов**

Малоразрядность стандартных дешифраторов ставит вопрос о наращивании их разрядности. Из малоразрядных дешифраторов можно построить схему, эквивалентную дешифратору большей разрядности. Для этого входное слово делится на поля. Разрядность поля младших разрядов соответствует числу входов имеющихся дешифраторов. Оставшееся поле старших разрядов служит для получения сигналов разрешения работы одного из дешифраторов, декодирующих поле младших разрядов.

В качестве примера на рис. 47 приведена схема дешифрации пятиразрядного двоичного кода с помощью дешифраторов «3-8» и «2-4». Для получения нужных 32 выходов составляется столбец из четырех дешифраторов «3-8» Дешифратор «2-4» принимает два старших разряда входного кода. Возбужденный единичный выход этого дешифратора отпирает один из дешифраторов столбца по его входу разрешения. Выбранный дешифратор столбца расшифровывает три младших разряда входного слова.

Каждому входному слову соответствует возбуждение только одного выхода. Например, при дешифрации слова на входе дешифратора первого яруса имеется код «11», возбуждающий его выход номер три (показано крестиком), что разрешает работу . На входе действует код «001», поэтому единица появится на его первом выходе, то есть на 25-ом выходе схемы в целом, что и требуется.



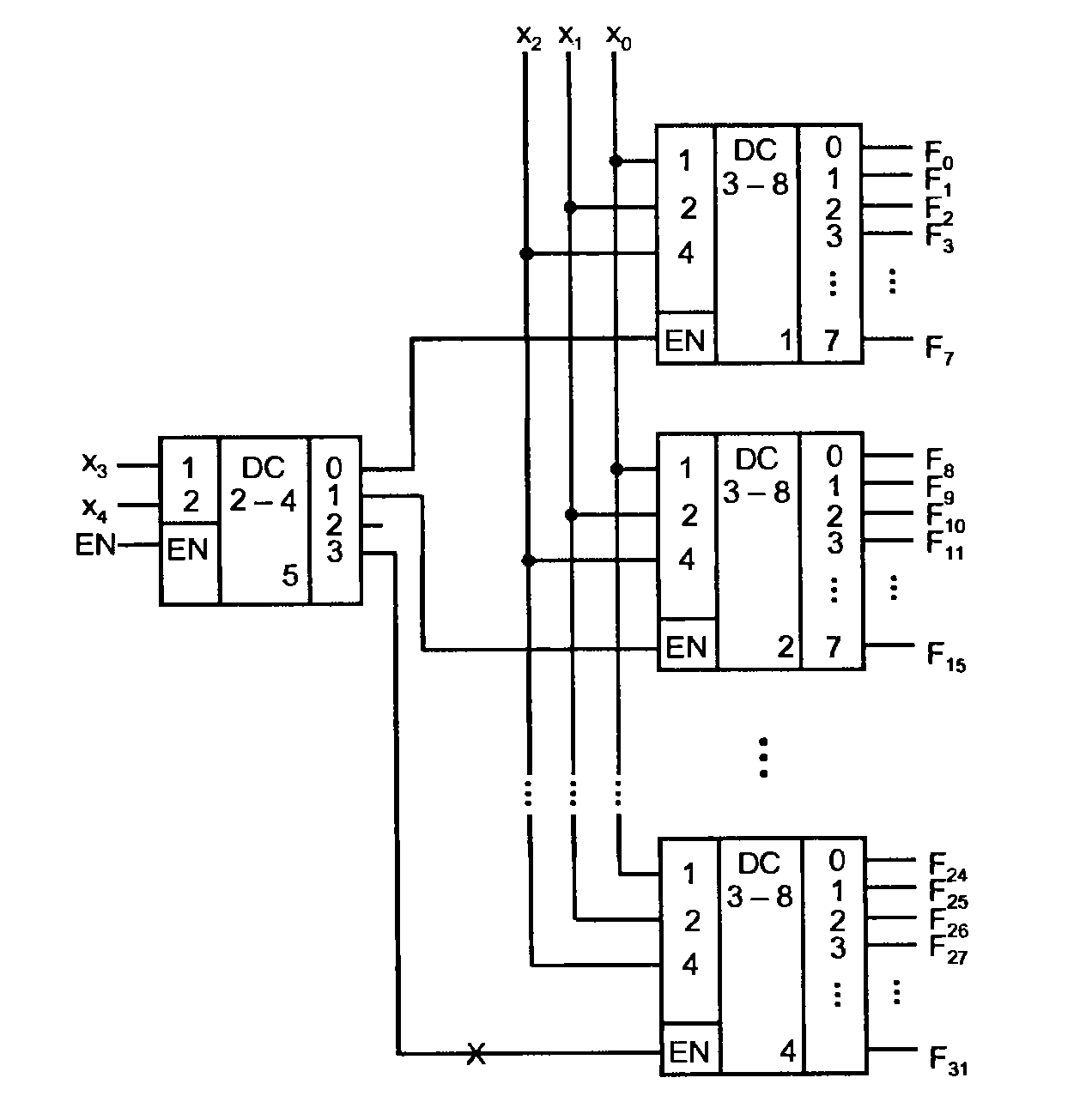


Рис. 47. Схема наращивания размерности двоичного дешифратора

Общее разрешение или запрещение работы схемы осуществляется по входу *EN* дешифратора первого яруса.

Дешифраторы совместно со схемами «ИЛИ» можно использовать для воспроизведения произвольных логических функций. Действительно, на выходах дешифратора вырабатываются все ***конъюнктивные термы*** (конституенты единицы), которые только можно составить из данного числа аргументов. Логическая функция в СНДФ есть дизъюнкция некоторого числа таких термов. Собирая нужные термы по схеме «ИЛИ», можно получить любую функцию данного числа аргументов.

На рис. 48 в качестве примера показана схема выработки двух функций и . Такое решение может быть целесообразным при необходимости выработки нескольких функций одних и тех же аргументов. В этом случае для выработки дополнительной функции добавляется только один дизъюнктор. Заметим, что для проверки правильности схемы рис. 48 удобно перевести функции и в СДНФ.



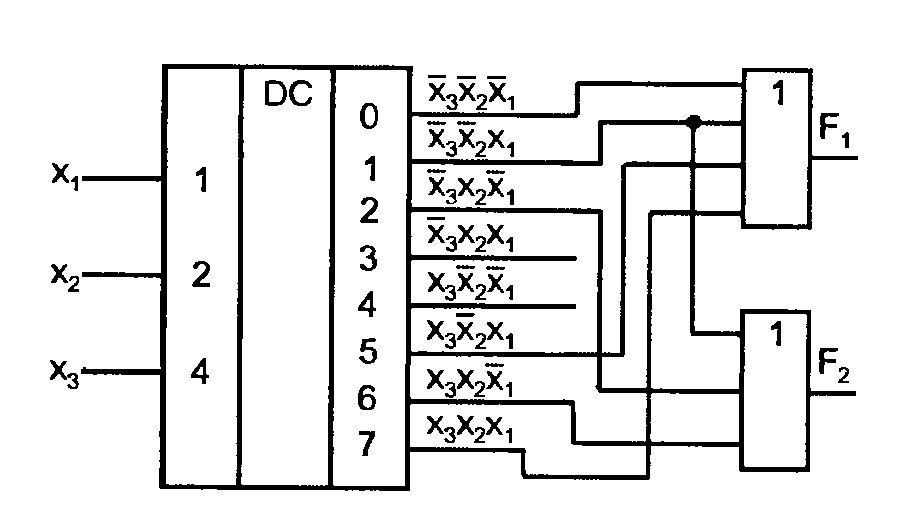


Рис. 48. Схема воспроизведения произвольных логических функций

с помощью дешифратора и дизъюнкторов

**Мультиплексоры**

Мультиплексоры осуществляют подключение одного из входных каналов к выходному под воздействием соответствующего управляющего (адресующего) слова. Мультиплексор представляет собой электронный коммутатор (переключатель), имеющий 2, 4, 8 или 16 входов данных и один информационный выход. Для управления передачей данных используются адресные входы мультиплексора – номер входа данных, подключенного в данный момент к информационному выходу, определяется двоичной комбинацией на адресных входах. Кроме того, у некоторых мультиплексоров имеется вход разрешения *E* (*Enable*), который управляет работой мультиплексора в целом (запрещает или разрешает передачу данных со входов на выход).

Входы мультиплексора делятся на две группы: информационные и адресные. Работу мультиплексора можно упрощенно представить с помощью многопозиционного ключа (рис. 49, а). Адресующий код *А* задает переключателю определенное положение, соединяя с выходом *F* один из информационных входов . При нулевом адресующем коде переключатель занимает верхнее положение , с увеличением кода на единицу переходит в соседнее положение и т. д.



|  |  |
| --- | --- |
| а) | б) |

Рис. 49. Упрощенное представление мультиплексора (а)

и реализация мультиплексора на элементах «И-НЕ»

Работа мультиплексора описывается соотношением

,



которое иногда называется мультиплексной формулой. При любом значении адресующего кода все слагаемые, кроме одного, равны нулю. Ненулевое слагаемое равно , где – значение текущего адресного кода.



Схемотехнически мультиплексор реализует электронную версию показанного выше переключателя, имея, в отличие от него, только одностороннюю передачу данных. На рис. 49, б показан мультиплексор с четырьмя информационными входами, двумя адресными входами и входом разрешения работы.

При отсутствии разрешения работы (*Е* = 0) выход *F* становится нулевым независимо от информационных и адресных сигналов.

**Наращивание размерности мультиплексоров**

Наращивание размерности мультиплексоров возможно с помощью пирамидальной структуры, состоящей из нескольких мультиплексоров. При этом первый ярус схемы представляет собою столбец, содержащий столько мультиплексоров, сколько необходимо для получения нужного числа информационных входов. Все мультиплексоры столбца адресуются одним и тем же кодом, составленным из соответствующего числа младших разрядов общего адресного кода (если число информационных входов схемы равно , то общее число адресных разрядов равно *n*, младшее поле адресного кода используется для адресации мультиплексоров первого яруса). Старшие разряды адресного кода, число которых равно , используются во втором ярусе, мультиплексор которого обеспечивает поочередную работу мультиплексоров первого яруса на общий выходной канал.



Пирамидальная схема, выполняющая функции мультиплексора «32/1» и построенная на мультиплексорах меньшей размерности, показана на рис. 50. В обозначении мультиплексоров используется сокращение MUX от англ. MULTIPLEXER.

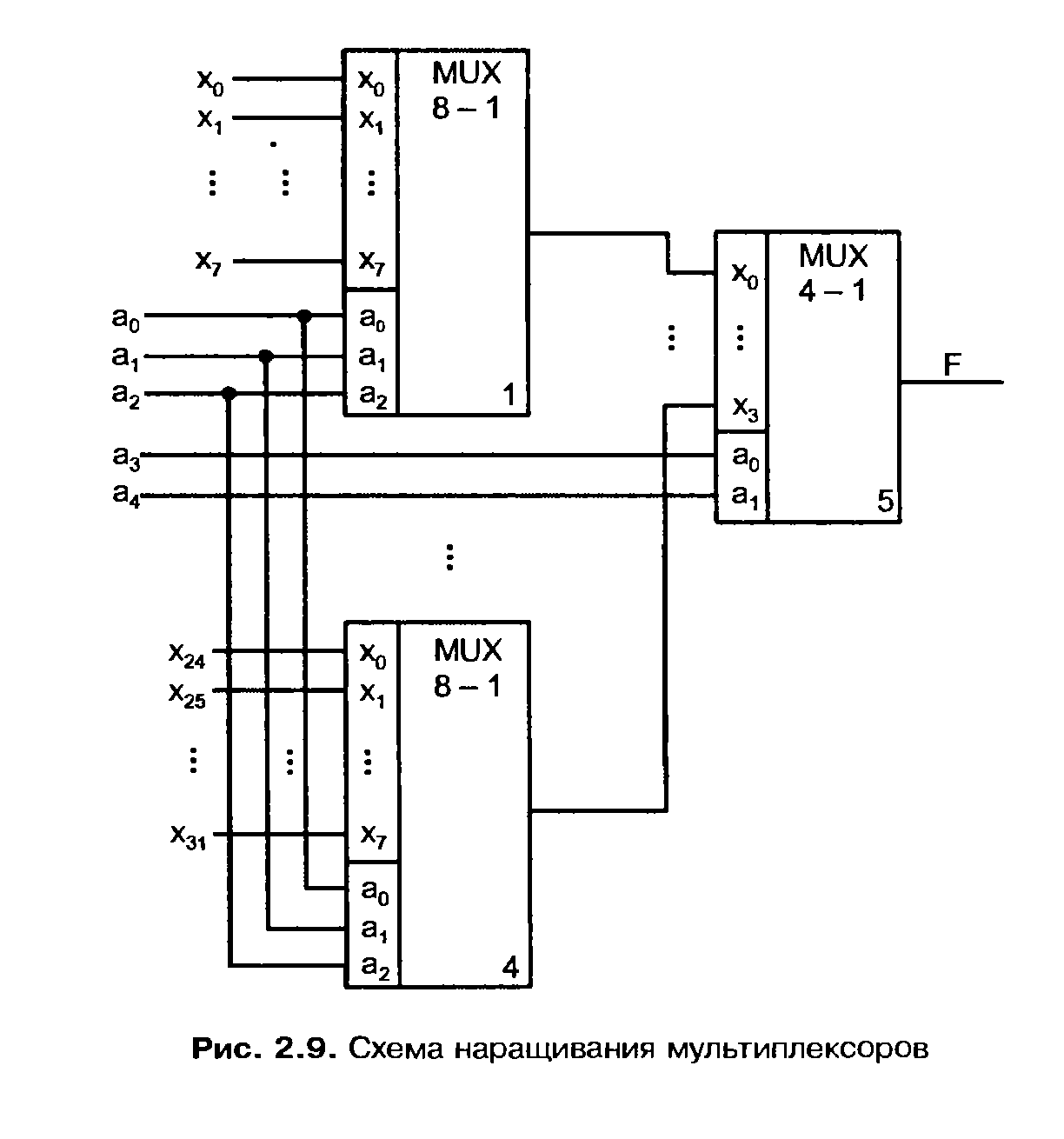


Рис. 50. Пример наращивания размерности мультиплексоров

Мультиплексоры различных типов очень удобно использовать в качестве элементов для покрытия логических схем, а сама процедура синтеза схем на мультиплексорах является сравнительно простой. Необходимо просто разложить в матрицу числовую последовательность покрываемой логической функции по всем входным аргументам, причём число аргументов должно соответствовать количеству адресных входов используемого мультиплексора. Каждая строка полученной при этом матрицы будет определять числовую последовательность, которую нужно подать на соответствующий вход данных мультиплексора.

Например, с помощью мультиплексора 4/2 можно реализовать любую логическую функцию трёх аргументов, так как при разложении её числовой последовательности в матрицу исключаются два аргумента (эти переменные подаются на адресные (управляющие) входы), а на входы данных подаются функции одного аргумента. Это могут быть либо константы, либо функция тождества, либо инверсия третьего аргумента.

С помощью мультиплексора 4/1 можно реализовать логические функции от большего, чем три, числа аргументов. При этом числовую последовательность рассматриваемой логической функции нужно разложить по различным парам аргументов и выбрать такую матрицу, числовые последовательности строк которой определяют наиболее простые остаточные логические функции, которые должны быть поданы на входы данных мультиплексора. На адресные (управляющие) входы подаются переменные, по которым произведено разложение числовой последовательности в этой матрице.

При использовании более сложных мультиплексоров (8/1 или 16/1) возможно достаточно простое покрытие функций четырёх или пяти аргументов. При этом разложение ведётся по трём (в случае использования мультиплексора 8/1) или четырём (при использовании мультиплексора 16/1) входным переменным, которые затем подаются на адресные входы мультиплексора. Каждая из строк полученной матрицы состоит из двух элементов и представляет собой либо одну из констант, либо функцию тождества, либо инверсию оставшегося невыделенным аргумента. Эти строки необходимо подать на соответствующие входы данных мультиплексоров. Переменные для построения матрицы разложения выбираются таким образом, чтобы в строках по возможности отсутствовала комбинация «10». В этом случае удаётся реализовать требуемую функцию без использования дополнительных инверторов. В противном случае потребуется как минимум один элемент для формирования инверсии оставшегося аргумента.

В общем случае для покрытия логической функции аргументов мультиплексором, имеющим адресных (управляющих) входов, необходимо построить все матрицы разложения этой функции по входным переменным. Каждая из строк такой матрицы будет представлять собой логическую функцию () аргументов. Задача в данном случае заключается в выборе такого варианта разложения, который требует формирования меньшего числа таких дополнительных функций (матрица должна содержать как можно больше строк, соответствующих константам или невыделенным аргументам).



**Типовые комбинационные устройства, представленные в библиотеке символов пакета WebPACK ISE**

Библиотека символов Symbol программного пакета WebPACK ISE включает в себя несколько разделов, содержащих типовые комбинационные устройства:

– **Arithmetic** – арифметические устройства:

**ADD** – сумматоры с различным числом разрядов – от 1 до 16;

**ACC** – накапливающий сумматор с различным числом разрядов – от 1 до 16;

**ADSU** – арифметический блок для суммирования и вычитания с различным числом разрядов – от 1 до 16;

– **Comparator** – схемы сравнения двоичных кодов с различным числом разрядов – от 2 до 16;

– **Decoder** – дешифраторы в коде «1 из N» с 4, 8 и 16 выходами;

– **Mux** – мультиплексоры с 2, 4, 8 и 16 входами.

**Задачи и порядок проведения работы:**

1. Ознакомиться с типовыми комбинационными устройствами, входящими в состав библиотеки символов программного пакета WebPACK ISE;

2. Изучить особенности схемотехнической реализации типовых комбинационных устройств с использованием встроенной справочной системы программного пакета WebPACK ISE;

3. Реализовать заданную в домашней работе № 1 комбинационную схему на мультиплексорах из библиотеки символов программного пакета WebPACK ISE;

4. Провести проверку работы синтезированной схемы на лабораторном стенде ЛСЦ-003 с подключением к выходам схемы дешифратора семисегментного индикатора.

**Форма отчётности студентов:**

индивидуальный отчёт в электронной форме с типовым титульным листом в формате MS Word с последующей распечаткой на бумажном носителе.

**Содержание отчёта:**

1. Краткие теоретические сведения о типовых комбинационных устройствах, входящих в состав библиотеки символов программного пакета WebPACK ISE;

2. Схемы синтезированных комбинационных устройств, выполненные с использованием схемотехнического редактора пакета WebPACK ISE;

3. Результаты испытания синтезированных схем на лабораторном стенде ЛСЦ-003;

4. Выводы в развёрнутой форме с соответствующими пояснениями.

Выполнение и защита лабораторной работы производится ***строго*** во время занятий. Защита работы – демонстрация работоспособности спроектированного цифрового устройства, собеседование с преподавателем и выполнение индивидуальных заданий.

**Вопросы для самоконтроля и подготовки к защите лабораторной работы № 6**

1. Опишите структуру обозначения типовых комбинационных устройств, входящих в состав пакета WebPACK ISE;

2. Опишите логическую функцию, реализуемую комбинационным устройством ADD4;

3. Опишите логическую функцию, реализуемую комбинационным устройством ADSU4;

4. Опишите логическую функцию, реализуемую комбинационным устройством ADD1, приведите его числовую последовательность;

5. Опишите логические функции, реализуемые комбинационными устройствами, входящими в группу Decoder;

6. Опишите логические функции, реализуемые комбинационными устройствами, входящими в группу Mux;

**ЛАБОРАТОРНАЯ РАБОТА № 7. ИСПОЛЬЗОВАНИЕ ПРОРАММНОГО ПАКЕТА DECOMPOSER ДЛЯ ПРОЕКТИРОВАНИЯ КОМБИНАЦИОННЫХ СХЕМ**

**Общие сведения о программном пакете Decomposer**

Программный пакет Decomposer реализует алгоритмы параллельной и последовательной декомпозиции, детализации, анализа и формирования VHDL-описания синтезированной схемы.

Исходная КЛС задаётся в виде числовой последовательности, которая может быть прочитана из файла, либо введена с клавиатуры. Прежде чем ввести последовательность, необходимо задать имя проекта, количество входов и выходов проектируемой схемы.

Файл с числовой последовательностью может быть подготовлен с помощью какой-либо программы на любом языке высокого уровня, которая в цикле перебирает в порядке возрастания все входные двоичные наборы, вычисляет значение выходной логической функции и сохраняет полученные значения в виде шестнадцатеричных цифр в файле с расширением *lof* (элементы числовой последовательности могут разделяться любыми символами: «Перевод строки», «Пробел», «Табуляция» или выводиться без разделителей).

В качестве примера рассмотрим синтез двоичного сумматора по модулю 16 с использованием программы Decomposer*.*

Синтезируемая схема имеет 9 входов и 5 выходов (рис. 51):

входы , и выходы – описываются шестнадцатеричными цифрами – 0, 1, 2 …,, , , , , (в целях экономии места используются обозначения цифр строчными латинскими буквами),



и соответственно входной и выходной сигналы переноса, которые принимают значения 1 или 0 (есть перенос или нет). Каждой из входных переменных присвоено числовое обозначение, соответствующее весовому коэффициенту во входном наборе. Индекс 0 указывает на то, что это внешняя входная переменная (вход всей схемы). Выходные переменные обозначены аналогично.



Рис. 51. Двоичный сумматор по модулю 16

Сложность исходной схемы бит.



**Абстрактный синтез сумматора по модулю 16**

Получим систему собственных функций сумматора. Наличие переноса в старший разряд будем обозначать знаком «апостроф» (') над соответствующей цифрой.

Допустим, , принимает значения 0… (15), а или 1 (всего 32 значения). Значения выхода определяются по формуле и записываются в первой строке матрицы.



Во второй строке , также принимает значения 0… (15), а или 1 и так далее до значения .



Числовая последовательность синтезируемого сумматора имеет вид строки, состоящей из 512 элементов (табл. 11).



Таблица 11 – Числовая последовательность синтезируемой схемы

| bi | 0011 | 0011 | 0011 | 0011 | 0011 | 0011 | 0011 | 0011 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pi-1 | 0101 | 0101 | 0101 | 0101 | 0101 | 0101 | 0101 | 0101 |
| ai=0 | 0112 | 2334 | 4556 | 6778 | 899a | abbc | cdde | eff0' |
| ai=1 | 1223 | 3445 | 5667 | 7889 | 9aab | bccd | deef | f0'0'1' |
| ai=2 | 2334 | 4556 | 6778 | 899a | abbc | cdde | eff0' | 0'1'1'2' |
| ai=3 | 3445 | 5667 | 7889 | 9aab | bccd | deef | f0'0'1' | 1'2'2'3' |
| ai=4 | 4556 | 6778 | 899a | abbc | cdde | eff0' | 0'1'1'2' | 2'3'3'4' |
| ai=5 | 5667 | 7889 | 9aab | bccd | deef | f0'0'1' | 1'2'2'3' | 3'4'4'5' |
| ai=6 | 6778 | 899a | abbc | cdde | eff0' | 0'1'1'2' | 2'3'3'4' | 4'5'5'6' |
| ai=7 | 7889 | 9aab | bccd | deef | f0'0'1' | 1'2'2'3' | 3'4'4'5' | 5'6'6'7' |
| ai=8 | 899a | abbc | cdde | eff0' | 0'1'1'2' | 2'3'3'4' | 4'5'5'6' | 6'7'7'8' |
| ai=9 | 9aab | bccd | deef | f0'0'1' | 1'2'2'3' | 3'4'4'5' | 5'6'6'7' | 7'8'8'9' |
| ai=a | abbc | cdde | eff0' | 0'1'1'2' | 2'3'3'4' | 4'5'5'6' | 6'7'7'8' | 8'9'9'a' |
| ai=b | bccd | deef | f0'0'1' | 1'2'2'3' | 3'4'4'5' | 5'6'6'7' | 7'8'8'9' | 9'a'a'b' |
| ai=c | cdde | eff0' | 0'1'1'2' | 2'3'3'4' | 4'5'5'6' | 6'7'7'8' | 8'9'9'a' | a'b'b'c' |
| ai=d | deef | f0'0'1' | 1'2'2'3' | 3'4'4'5' | 5'6'6'7' | 7'8'8'9' | 9'a'a'b' | b'c'c'd' |
| ai=e | eff0' | 0'1'1'2' | 2'3'3'4' | 4'5'5'6' | 6'7'7'8'' | 8'9'9'a' | a'b'b'c' | c'd'd'e' |
| ai=f | f0'0'1' | 1'2'2'3' | 3'4'4'5' | 5'6'6'7' | 7'8'8'9' | 9'a'a'b' | b'c'c'd' | d'e'e'f' |

После запуска программы Decomposer в строке меню *Файл* необходимо выбрать команду *Новый* и в открывшемся окне *Создание проекта* задать требуемые параметры – имя проекта, количество входов и выходов синтезируемой схемы и способ ввода числовой последовательности (рис. 52).

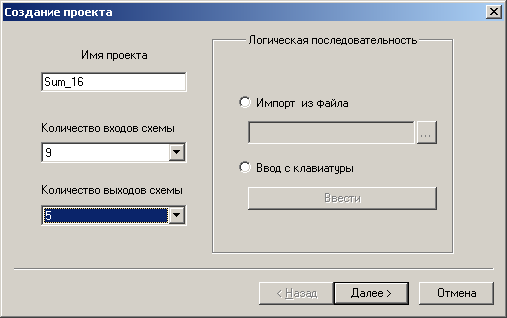


Рис. 52. Задание параметров нового проекта

Для ввода числовой последовательности с клавиатуры открывается специальное окно, изображённое на рис. 53. Элементы исходной последовательности вводятся в виде шестнадцатеричных цифр поверх элементов разметки. Для ввода элементов последовательности используются цифры 0…9 и латинские буквы A … F. Если логические функции синтезируемой схемы являются недоопределёнными, то соответствующие элементы числовой последовательности заполняются звёздочками (\*).

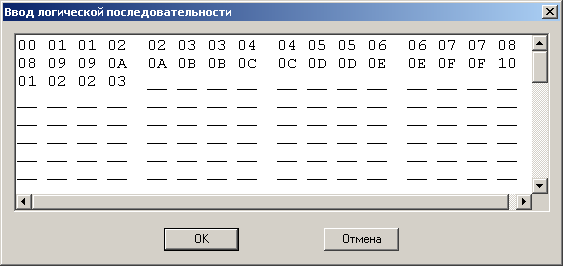


Рис. 53. Ввод исходной числовой последовательности с клавиатуры

Создание нового проекта завершается выводом информационного сообщения, содержащего все необходимые параметры проекта (рис. 54).

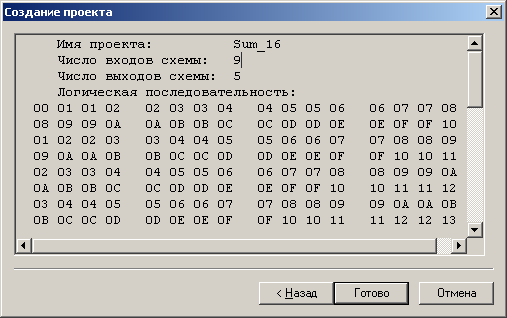


Рис. 54. Информационное окно с параметрами проекта

После нажатия кнопки *Готово* открывается главное окно программы Decomposer, содержащее графическое изображение синтезируемого устройства и его числовую последовательность (рис. 55).

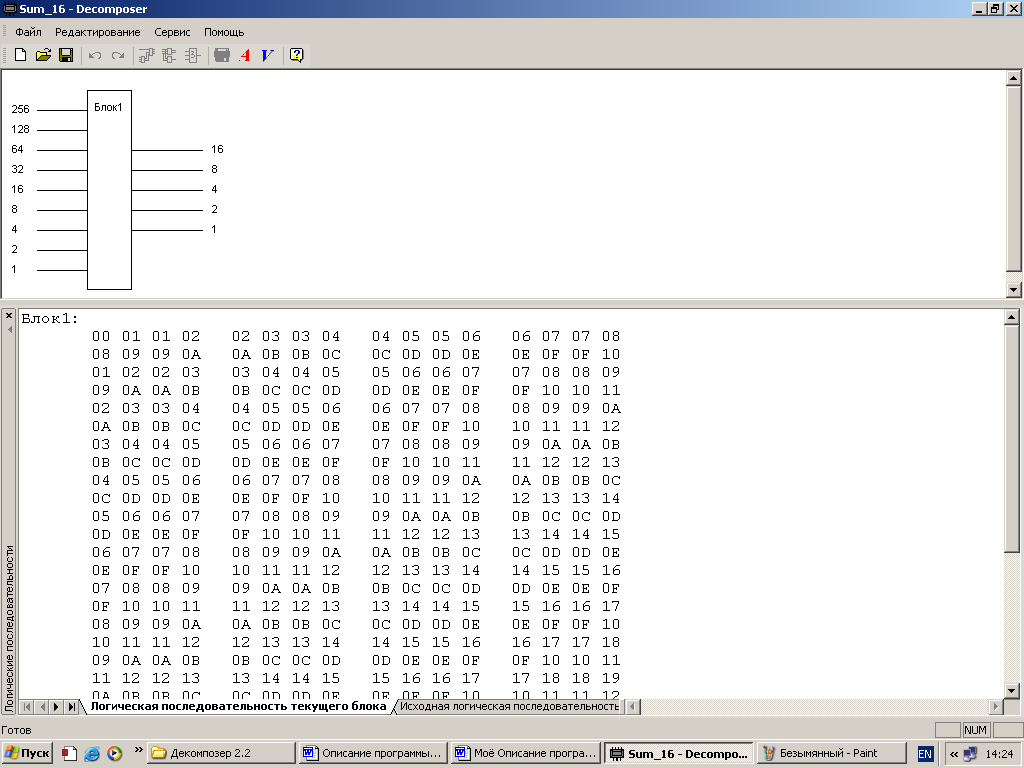


Рис. 55. Главное окно программы Decomposer

Так как схема имеет несколько выходов, попытаемся выделить параллельный блок. Для этого проверим наличие существенной или фиктивной зависимости выходных функций от всех аргументов. В программе Decomposer для этого необходимо выделить мышкой исследуемый блок и выбрать соответствующий пункт из контекстного меню или из пункта меню *Редактирование* (рис. 56).

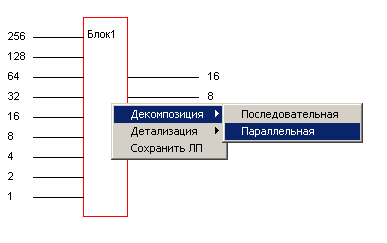


Рис. 56. Параллельная декомпозиция синтезируемой схемы

Если логическая функция (выход) существенно зависит от какой-либо входной переменной, в соответствующей клетке таблицы записывается «1», в противном случае – «0» (рис. 57).

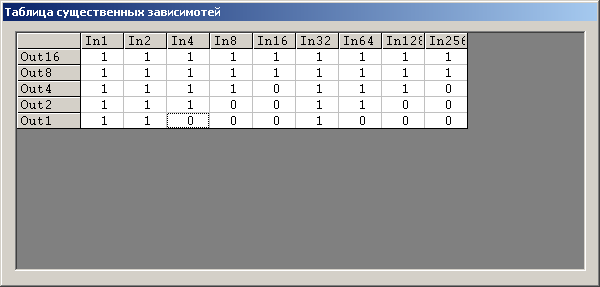


Рис. 57. Таблица анализа фиктивной зависимости

выходных функций схемы

Из построенной таблицы видно, что два младших выходных разряда ( (выход ) и (выход )) фиктивно зависят от входных переменных , , , (входы с весами , , , ), следовательно, можно выделить параллельный блок (фиктивную зависимость выхода от входов и и выхода от входов и пока учитывать не будем, поскольку нам необходимо разделить схему на два примерно равных по сложности блока). Выбор входных и выходных переменных выделяемого параллельного блока осуществляется в специальном окне программы (рис. 58).



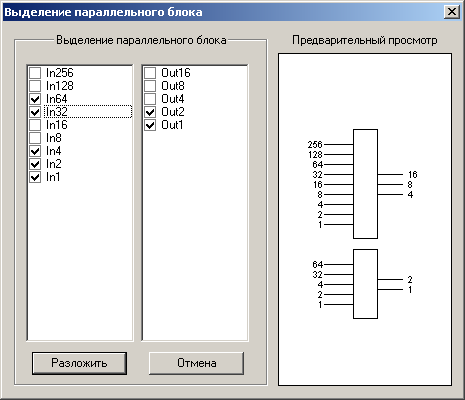


Рис. 58. Окно выделения параллельного блока

В правой части этого окна имеется возможность предварительного просмотра результатов декомпозиции. После выбора входов и выходов выделяемого блока следует нажать клавишу *Разложить*.

Одновременно с выделением параллельного блока проверяется возможность выделения последовательного блока с такими же входами (то есть проводится параллельно-последовательная декомпозиция) и на экран выводится соответствующее сообщение (рис. 59).

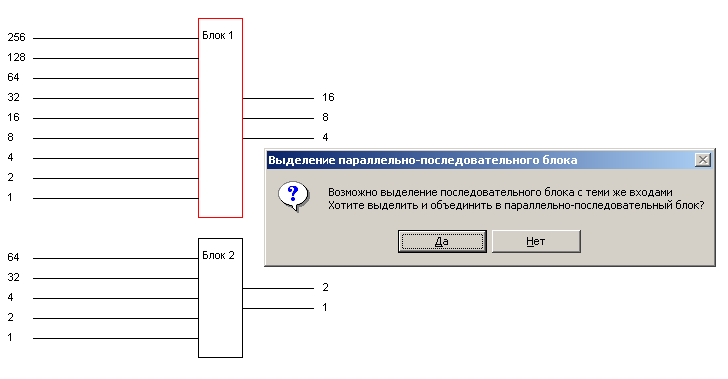


Рис. 59. Выделение параллельно-последовательного блока

В результате такого разделения схема будет иметь вид, представленный на рис. 60.

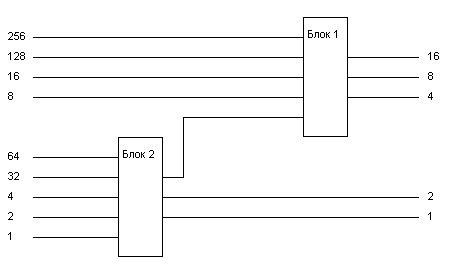


Рис. 60. Разделение схемы сумматора на 2 блока

В нижней части главного окна программы выводятся имена блоков, получившихся в результате декомпозиции и их числовые последовательности (рис. 61).

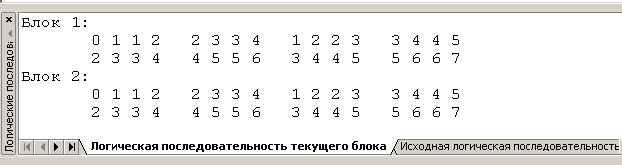


Рис. 61. Имена выделенных блоков и их

числовые последовательности

В результате выполнения этих операций получились два совершенно одинаковых блока с пятью входами и тремя выходами каждый. Сложность схемы стала равна бит.



Далее будем рассматривать декомпозицию одного из блоков, поскольку они совершенно одинаковые.

Исследуем наличие фиктивной зависимости выходов от входов для блока 2 (рис. 62).

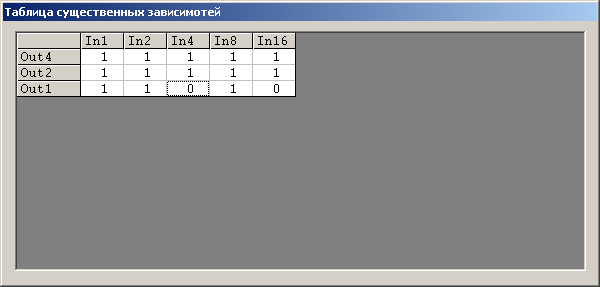


Рис. 62. Таблица анализа фиктивной зависимости

выходных функций блока 2

Из построенной таблицы видно, что младший выход рассматриваемого блока () фиктивно зависят от входных переменных и , следовательно, можно выделить параллельный блок (рис. 63). Одновременно с выделением параллельного блока производится выделение последовательного блока с такими же входами (то есть проводится параллельно-последовательная декомпозиция). Результат этих операций представлен на рис. 64.



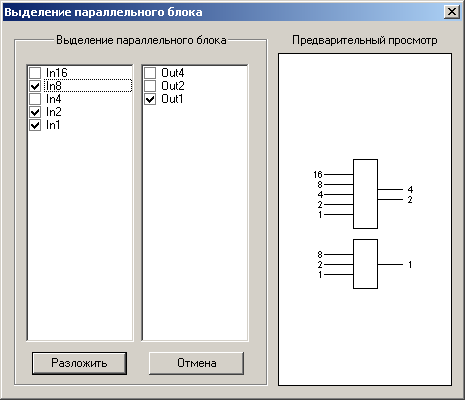


Рис. 63. Выделение параллельного блока из блока 2

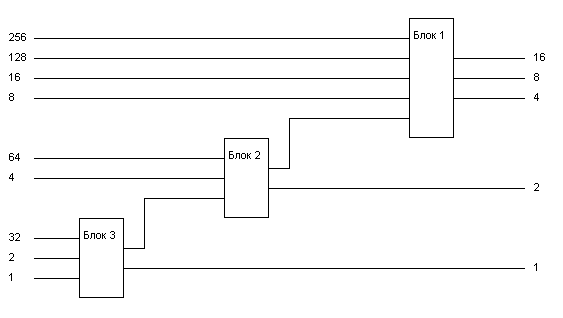


Рис. 64. Результат разделения блока 2

Получившиеся в результате этих операций блоки 2 и 3 с последовательностями представляют собой одноразрядные двоичные сумматоры (рис. 65).



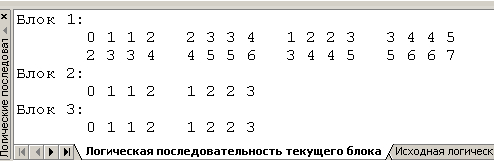


Рис. 65. Числовые последовательности выделенных блоков

Аналогично производится декомпозиция блока 1. Окончательная схема разряда двоичного сумматора по модулю 16 изображена на рис. 66.

Все блоки однотипные и характеризуются одинаковыми последовательностями – . Это одноразрядные двоичные сумматоры. Сложность схемы на этом этапе составляет бита.



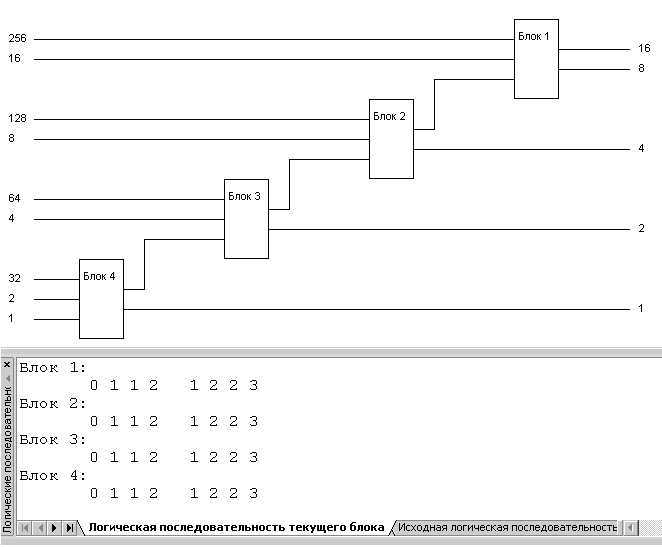


Рис. 66. Схема двоичного сумматора по модулю 16

Дальнейшая декомпозиция схемы невозможна, так как параллельные и последовательные блоки из сумматоров не выделяются (рис. 67 и 68).

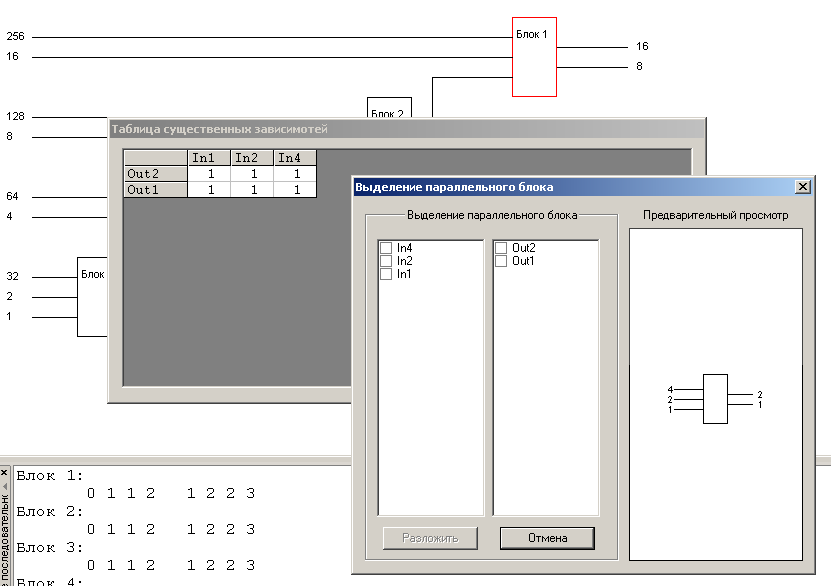


Рис. 67. Анализ фиктивной зависимости выходных

функций двоичного сумматора

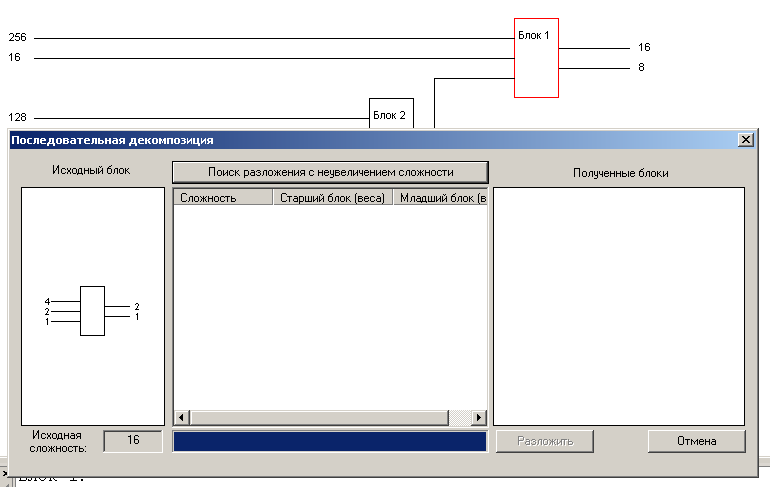


Рис. 68. Поиск вариантов последовательной

декомпозиции двоичного сумматора

Чтобы убедиться в правильности результатов синтеза схемы необходимо проделать анализ полученной схемы. Для этого можно использовать соответствующую команду из пункта меню *Сервис* или кнопку с символом ***A*** на панели инструментов. На экран выводится окно с результатами анализа, в котором размещены элементы исходной и полученной числовых последовательностей (рис. 69).

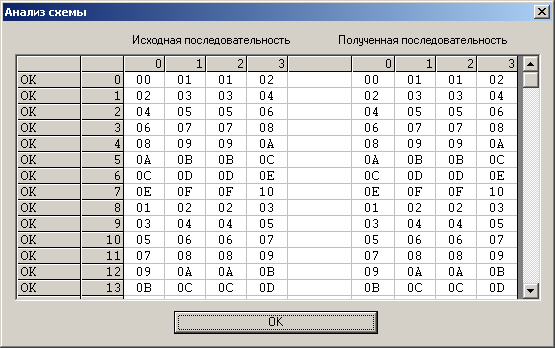


Рис. 69. Окно анализа синтезируемой схемы

Соответствующие элементы последовательностей сравниваются между собой и в случае совпадения отмечаются символами «ОК». Сравнение элементов производится с учётом возможного доопределения при синтезе недоопределённых логических функций (если в исходной последовательности на каком-либо месте находится звёздочка (\*), а в полученной последовательности – некоторая определённая комбинация, то такая ситуация считается допустимой).

Поскольку все блоки декомпозированной схемы одинаковые, то целесообразнее продолжить работу с одним из них, для чего следует использовать соответствующую команду контекстного меню и сохранить числовую последовательность в файл, а затем открыть этот файл в новом проекте (рис. 70).

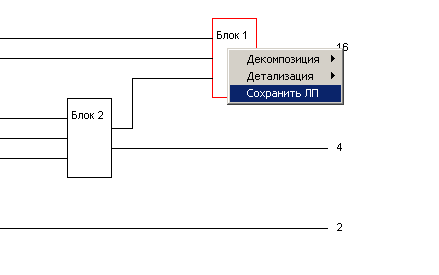


Рис. 70. Сохранение логической (числовой) последовательности

выделенного блока

**Детализация схемы до уровня двухвходовых блоков на примере синтеза разряда двоичного сумматора**

Числовая (логическая) последовательность двоичного сумматора имеет вид:



и представляет собой систему из двух логических функций, зависящих от трёх аргументов – переноса () и суммы – ().



Для реализации этого блока на двухвходовых элементах необходимо его дальнейшее разделение на более мелкие (двухвходовые) блоки – детализация. При выполнении детализации разделение может происходить без учёта критерия уменьшения сложности схемы. Задача сводится к получению блоков, соизмеримых по сложности с элементами покрытия. Детализация, так же как и декомпозиция, проводится без привязки к конкретному логическому базису.

Первый этап детализации – принудительно разделяем сумматор на одновыходовые блоки и пытаемся провести их последовательную декомпозицию (рис. 71).

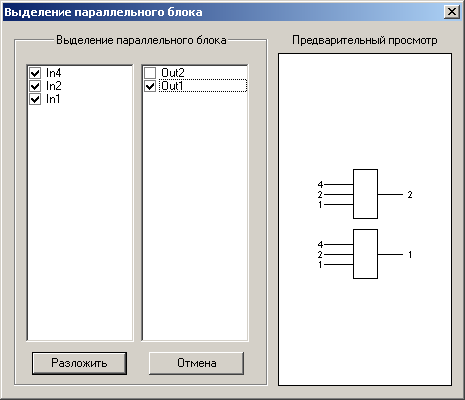


Рис. 71. Принудительное разделение сумматора

на параллельные блоки

Блок 2 (выход ), в отличие от блока 1 (выход ) может быть подвергнут последовательной декомпозиции (рис. 72).



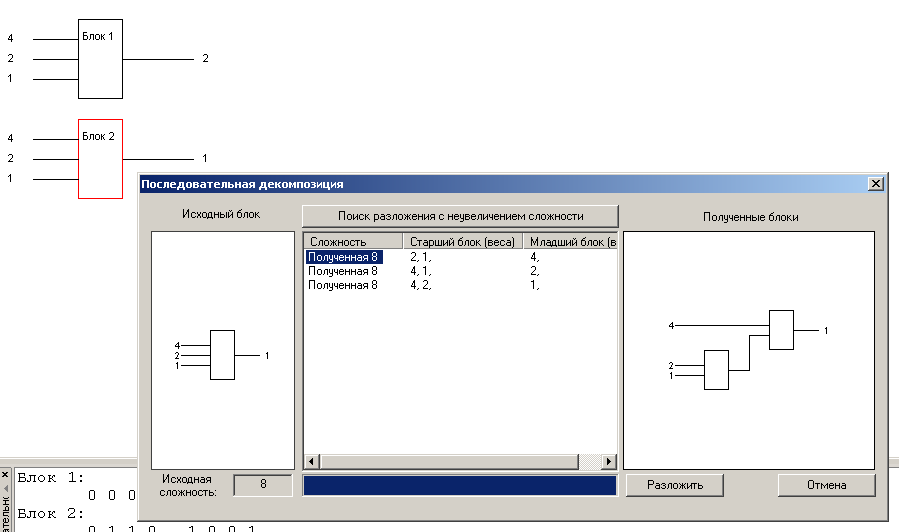


Рис. 72. Выбор варианта последовательной декомпозиции блока 2

Результат первого этапа детализации приведён на рис. 73. Полученные в результате детализации двухвходовые блоки обозначаются соответствующими символами («ИСКЛЮЧАЮЩЕЕ ИЛИ») вместо имени блока.

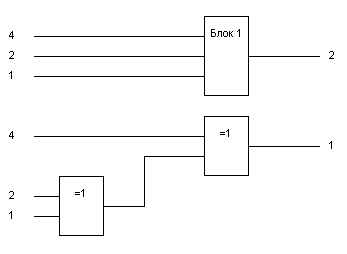


Рис. 73. Последовательная декомпозиция блока 2

Второй этап детализации – производим принудительное разделение схемы для функции , не требуя при этом уменьшения сложности. Для этого на выходе блока 1 выделяется трёхвходовый блок с использованием соответствующей команды контекстного меню (рис. 74). Возможные варианты разделения приводятся в открывающемся окне предварительного просмотра (рис 75). При выборе конкретного варианта следует учесть, что трёхвходовые блоки второго и третьего типов имеют наиболее простую окончательную реализацию на двухвходовых элементах.



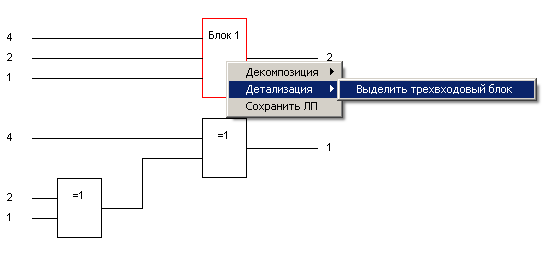


Рис. 74. Выделение трёхвходового блока при детализации

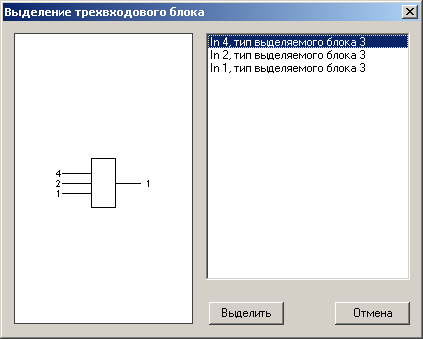


Рис. 75. Выбор варианта принудительного разделения блока 1

Результат второго этапа детализации схемы двоичного сумматора представлен на рис. 76.

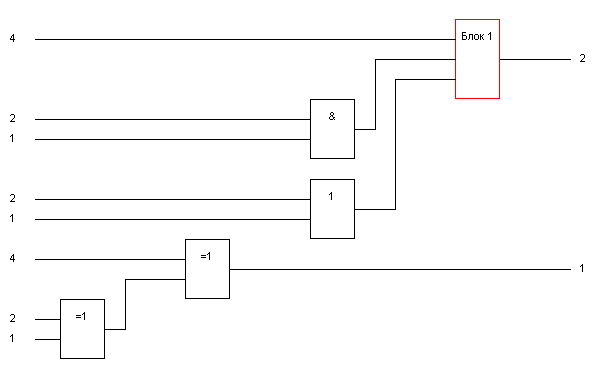


Рис. 76. Схема сумматора, полученная на 2-м этапе детализации

Третий этап детализации – разлагаем трёхвходовый блок 1 (блок третьего типа) на двухвходовые блоки (рис. 77). Выбор варианта декомпозиции производится в соответствии с типом логических элементов, заданных для покрытия схемы (при покрытии элементами «2И-НЕ» целесообразно использовать первый вариант, поскольку в противном случае потребуется поставить инвертор на выходе схемы).

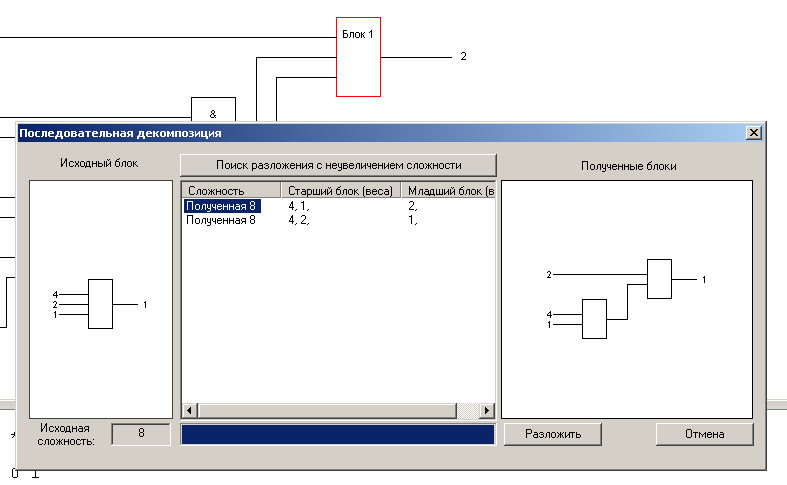


Рис. 77. Последовательная декомпозиция

трёхвходового блока 3-го типа

Окончательная детализированная схема двоичного сумматора, выполненная на двухвходовых блоках, приведена на рис. 78.

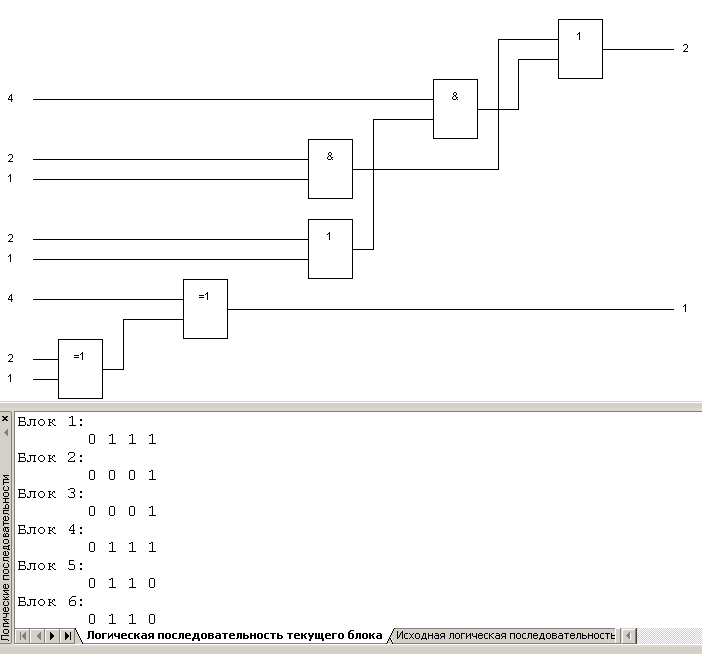


Рис. 78. Детализированная схема двоичного сумматора.

Результаты анализа позволяют судить о правильности работы синтезированной схемы (рис. 79).

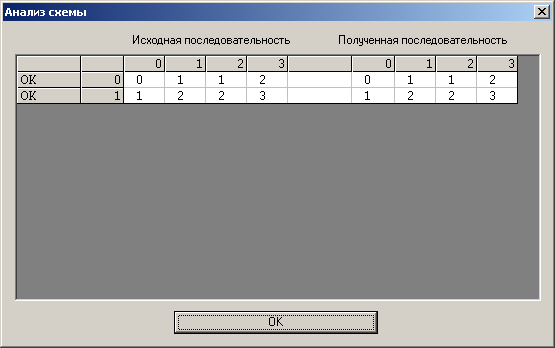


Рис. 79. Результаты анализа синтезированной схемы

двоичного сумматора

Пакет Decomposer предусматривает возможность генерации *VHDL*-кода синтезированной схемы в виде описания соответствия вход-выход для каждого из полученных блоков. Для этого можно использовать соответствующую команду из пункта меню *Сервис* или кнопку с символом ***V*** на панели инструментов.В открывшемся диалоговом окне предлагается ввести имя файла и путь для сохранения (рис. 80).

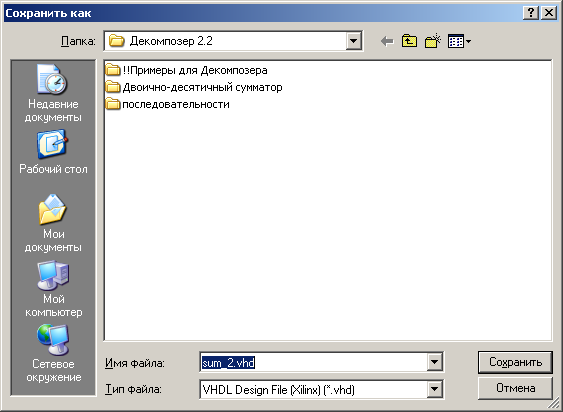


Рис. 80. Диалоговое окно для сохранения файла с VHDL-кодом

Полученный таким образом файл с *VHDL*-кодом можно использовать в других средах автоматизированного проектирования для физической реализации синтезированной схемы в виде прошивки в ПЛИС, например пакет WebPACK ISEфирмы Xilinx. Таким образом, пакет Decomposer позволяет провести полный цикл проектирования комбинационных логических схем – от исходного описания до практической реализации.

**Задачи и порядок проведения работы**

1. Изучить основные теоретические сведения о программном пакете Decomposer;

2. Изучить основные принципы работы со средой логического проектирования комбинационных схем;

3. Выполнить практические задания по синтезу цифровых устройств;

4. Провести проверку работы синтезированной схемы на лабораторном стенде ЛСЦ-003 с подключением к выходам схемы дешифратора семисегментного индикатора.

Проектируемое устройство задано в виде числовой последовательности и представляет собой комбинационную логическую схему (кодопреобразователь) с 4-мя входами и 4-мя выходами. Варианты заданий указаны в табл. 11. С использованием пакета Decomposer провести синтез логической схемы по заданным числовым последовательностям.

Таблица 11 – Варианты заданий для лабораторной работы № 7

| Номер вар. | Числовая последовательность кодопреобразователя |
| --- | --- |
| 1 | 8382 B8E1 EA6F 511D |
| 2 | 1612 71B4 B3AF C44D |
| 3 | 4341 74D2 D59F A22E |
| 4 | 1A12 B178 736F C88D |
| 5 | 4A42 E478 763F 988D |
| 6 | 1C50 1825 D35B E74C |
| 7 | 1A30 1843 B53D E72A |
| 8 | 4950 4825 D65E B719 |
| 9 | 2CA0 241A E3A7 DB8C |
| 10 | 85C0 812C DACB 7E45 |
| 11 | 8390 8249 BC9E 7D13 |
| 12 | 1650 1285 795B EED46 |
| 13 | 4360 4186 7C6D BE23 |
| 14 | 1A90 1249 B597 ED8A |
| 15 | 4AC0 421C E5C7 BD8A |

**Форма отчётности студентов:**

индивидуальный отчёт в электронной форме с типовым титульным листом в формате MS Word с последующей распечаткой на бумажном носителе.

**Содержание отчёта:**

1. Краткие теоретические сведения о декомпозиционных методах синтеза, необходимые для выполнения практического задания;

2. Результаты выполнения задания;

3. Результаты испытания синтезированной схемы на лабораторном стенде ЛСЦ-003;

4. Выводы в развёрнутой форме с соответствующими пояснениями;

Выполнение и защита лабораторной работы производится ***строго*** во время занятий. Защита работы – демонстрация работоспособности спроектированного цифрового устройства, собеседование с преподавателем и выполнение индивидуальных заданий.

**Вопросы для самоконтроля и подготовки к защите лабораторной работы № 7**

1. Опишите общий алгоритм декомпозиции логических схем;

2. Опишите алгоритм параллельной декомпозиции комбинационных устройств;

3. Опишите алгоритм последовательной декомпозиции комбинационных устройств;

4. Почему при параллельной декомпозиции сложность описания схемы всегда уменьшается?

5. Как изменяется сложность синтезируемой схемы при детализации?

**ОБЩИЕ ПРАВИЛА БЕЗОПАСНОСТИ ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНЫХ РАБОТ**

Перед началом работы в лаборатории необходимо:

– ознакомиться с инструкцией по технике безопасности при работе в лабораториях кафедры ЭИУ3-КФ МГТУ им. Н.Э. Баумана;

– ознакомиться с описаниями и инструкциями по эксплуатации используемых лабораторных стендов и измерительных приборов;

– удостовериться, что на рабочем месте отсутствуют посторонние предметы, мешающие работе на лабораторных стендах;

– проверить исправность кабелей, электрических шнуров, розеток и штепсельных вилок.

Меры безопасности при работе в лаборатории:

– при работе на стендах необходимо соблюдать все указания по безопасности, изложенные в руководствах по эксплуатации элементов и измерительных приборов, входящих в состав лабораторных стендов;

– ***запрещается проводить соединение элементов лабораторных стендов и управляющего персонального компьютера без отключения электропитания***.

**ЛИТЕРАТУРА**

1. Новиков, Ю.В. Введение в цифровую схемотехнику. [Электронный ресурс]: учебное пособие. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 392 c. – Режим доступа: <http://www.iprbookshop.ru/52187.html>. – ЭБС «IPRbooks»

2. Музылева, И.В. Основы цифровой техники. [Электронный ресурс]. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 250 c. – Режим доступа: <http://www.iprbookshop.ru/62821.html>. – ЭБС «IPRbooks»

3. Аверченков, О.Е. Основы схемотехники аналого-цифровых устройств. [Электронный ресурс]: учеб. пособие по курсу «Схемотехника ЭВМ». – М.: ДМК Пресс, 2012. – 80 с. Режим доступа: <http://e.lanbook.com/book/4139> – ЭБС «ЛАНЬ»

4. Ульрих Титце Полупроводниковая схемотехника. Том I. [Электронный ресурс]. – Саратов: Профобразование, 2017. – 826 c. – Режим доступа: <http://www.iprbookshop.ru/63579.html>. – ЭБС «IPRbooks»

5. Ульрих Титце Полупроводниковая схемотехника. Том II. [Электронный ресурс]. – Саратов: Профобразование, 2017. – 940 c. – Режим доступа: <http://www.iprbookshop.ru/63580.html>. – ЭБС «IPRbooks»

6 Аверченков, О.Е. Схемотехника: аппаратура и программы. [Электронный ресурс]. – М.: ДМК Пресс, 2012. – 588 с. – Режим доступа: <http://e.lanbook.com/book/4141> – ЭБС «ЛАНЬ»

7. Купцов, С.В. Практическая схемотехника. [Электронный ресурс]. – М.: Физматлит, 2016. – 296 с. – Режим доступа: <http://e.lanbook.com/book/91152> – ЭБС «ЛАНЬ»

8. Красовский, А.Б. Проектирование комбинационных цифровых устройств. [Электронный ресурс]. – М.: МГТУ им. Н.Э. Баумана, 2012. – 27 с. – Режим доступа: <http://e.lanbook.com/book/52372> – ЭБС «ЛАНЬ»

9. Кнышев Д.А., Кузелин М.О. ПЛИС фирмы «Xilinx»: описание структуры основных семейств [Электронный ресурс]. – М. «Додэка-XXI», 2010. – 230 с. – Режим доступа: <http://e.lanbook.com/books/element.php?pl1_id=40929>. – ЭБС «Lanbook».

10. Официальный сайт фирмы Xilinx. URL: <http://xilinx.com>.

11. Сайт официального дистрибьютора фирмы Xilinx ЗАО «КТЦ «Инлайн Групп». URL: <http://plis2.ru>.