

Московский государственный университет
имени М. В. Ломоносова

Физический факультет
кафедра фотоники и физики микроволн

О. Ю. Волков

ПРАКТИКУМ ПО РАДИОЭЛЕКТРОНИКЕ

ЦИФРОВЫЕ СХЕМЫ

Методическое пособие для студентов
и преподавателей практикума

Издание второе

Москва — 2016

УДК 378.162.33, 004.942, 53.083.8

ББК 22.3

Практикум по радиоэлектронике: цифровые схемы

Издание второе, переработанное и дополненное.

Учебно-методическое пособие для студентов физического факультета МГУ и преподавателей практикума по радиоэлектронике. М.: Физический факультет МГУ им. М.В. Ломоносова, 2016. — 96 с.

ISBN 978-5-8279-0104-4

Представлено переработанное описание цикла задач по цифровой электронике с введением ряда новых элементов профессиональной подготовки студентов. Программа состоит из шести задач. Каждая задача сопровождается списком контрольных вопросов и заданий, знание которых необходимо для успешного выполнения упражнений практикума.

Студенты изучают современный подход цифровой схемотехники с использованием программируемых интегральных логических схем в среде проектирования ISE — от использования простейших ПЛИС CPLD XC9572XL до более сложных ПЛИС FPGA Spartan6. В начале осваивается схематехническое описание проекта, а потом — на языке программирования VHDL. Студенты учатся описывать на VHDL логические операции, простые логические конструкции, а так же изучить работу более сложных элементов — триггеров, счетчиков, регистров и схем на их основе. Дополнительно происходит знакомство с методикой симуляции работы проекта, что позволяет тестировать и отлаживать сложные схемы за минимальное время.

Второе издание дополнено теоретическим материалом, новыми задачами и кратким изложением основ языка VHDL.

Рецензенты: д.ф.м.н. профессор *А.М. Салецкий*,
д.ф.м.н. доцент *Г.В. Белокопытов*

ВОЛКОВ Олег Юрьевич

Практикум по радиоэлектронике цифровые схемы

Объем 5,5 п.л.

Электронная версия.

Физический факультет МГУ им. М.В. Ломоносова.
119991, Москва, ГСП-1, Ленинские горы, д. 1, стр. 2

ISBN 978-5-8279-0104-4

© Волков О.Ю., 2012-2016

Содержание

Предисловие	5
7. Комбинационные логические схемы	7
7.1. Основные понятия	7
7.2. Метод упрощения логических функций	12
7.3. Комбинационные логические схемы в ПЛИС	14
7.4. Практическая часть	15
7.5. Контрольные вопросы	16
8. Триггеры	17
8.1. Основные понятия	17
8.2. Асинхронный и синхронный RS-триггеры	18
8.3. Синхронный D-триггер	20
8.4. Синхронный JK-триггер	21
8.5. Асинхронный и синхронный T-триггеры	22
8.6. VHDL описание триггеров	23
8.7. Практическая часть	24
8.8. Контрольные вопросы	26
8.9. Контрольные задания	26
9. Регистры и формирователи кода	27
9.1. Основные понятия	27
9.2. VHDL описание регистров	29
9.3. Формирование циклического кода	31
9.4. М-последовательность	32
9.5. Практическая часть	34
9.6. Контрольные вопросы	35
9.7. Контрольные задания	36

10. Сумматоры и счетчики	37
10.1. Основные понятия	37
10.2. VHDL описание счетчиков	40
10.3. Практическая часть	42
10.4. Контрольные вопросы	43
11. Системы счисления и целочисленная арифметика	45
11.1. Основные понятия	45
11.2. Умножение двоичных чисел	48
11.3. Работа с семисегментным индикатором	49
11.4. Практическая часть	50
11.5. Контрольные вопросы	51
12. Цифро-аналоговые и аналого-цифровые преобразователи . .	53
12.1. Основные понятия	53
12.2. Цифро-аналоговые преобразователи	54
12.3. Аналого-цифровые преобразователи	57
12.4. Практическая часть	59
12.5. Контрольные вопросы	60
Справочная информация	61
Программируемые логические интегральные схемы	61
Система автоматического проектирования Xilinx ISE	63
Симулятор ПЛИС ISim	65
Конструктор-ПЛИС CPLD (задачи 7-10)	66
Программирование ПЛИС через схемотехническое описание .	69
Программирование ПЛИС на языке VHDL	70
Конструктор-ПЛИС FPGA (задачи 11-12)	71
Программирование ПЛИС FPGA	76
Краткое описание языка VHDL	77
Предметный указатель	89
Литература	93

Предисловие

На отделении радиофизики физического факультета Московского государственного университета имени М.В.Ломоносова в конце 40-х годов прошлого века в дополнение к общему физическому практикуму был создан общефакультетский радиопрактикум. Его отличительной особенностью было изучение радиотехнических схем не на готовых макетах, а в процессе прохождения всех этапов создания простейших радиотехнических устройств: расчет, подготовка шасси или платы, монтаж и наладка схемы, снятие характеристик.

В дальнейшем по мере разработки новых электронных приборов (транзисторов и интегральных схем) практикум модернизировался, пополнялся новыми лабораторными работами, однако принцип индивидуальности задания и самостоятельности в монтаже схемы сохранялся.

В конце 80-х годов прошлого века радиопрактикум был преобразован в практикум по радиоэлектронике с элементами автоматизации и цифровой обработки результатов эксперимента на персональных компьютерах.

Примерно в то же время в мировой промышленности получила широкое распространение технология поверхностного монтажа компонент, что привело к миниатюризации размеров элементной базы (SMD компоненты) и стали активно развиваться программируемые логические интегральные схемы (ПЛИС).

В начале 2000-х годов появилась возможность использовать ПЛИС и SMD компоненты в разработке и создании собственных устройств не только в заводских, но и в лабораторных условиях. Это послужило стимулом для очередной модернизации радиопрактикума.

Учебное пособие содержит 12 задач. Первые 4 задачи (линейные

РС-цепи, усилитель на транзисторе, операционный усилитель, РС-генератор) — схемы аналоговой радиотехники. Пятая задача (мультивибраторы) — находится на стыке аналоговой и цифровой электроники. Шестая задача знакомит студентов со средой симуляции электронных схем, позволяющей проводить разработку и моделирование работы аппаратуры на ЭВМ. Четыре следующие (комбинационные логические схемы, триггеры, регистры и формирователи кода, сумматоры и счетчики) предназначены для знакомства с основными элементами цифровой схемотехники. Одинадцатая задача посвящена знакомству с различными системами счисления и целочисленной арифметикой. Последняя задача — цифро-аналоговые и аналого-цифровые преобразователи является очень важной для физиков-экспериментаторов, так как дает понимание условий, необходимых для стыковки аналоговых измерительных и исполнительных устройств с ЭВМ.

Вторая часть пособия охватывает цифровой раздел (задачи 7-12).

Задача 7.

Комбинационные логические схемы

Задача посвящена изучению принципов работы комбинационных логических схем и их построению на основе логических элементов.

7.1. Основные понятия

Алгебра логики — раздел математической логики, в котором изучаются логические операции над высказываниями. Чаще всего предполагается бинарная или двоичная логика.

В алгебре логики есть только две константы — 0 и 1. Переменными могут быть величины, имеющие состояния 0 и 1.

Определены три основные операции:

- НЕ — логическое отрицание, инверсия ($\bar{A}$, $\neg A$, *not* A);
- ИЛИ — логическое сложение, дизъюнкция ($\vee$, *or*, $|$, $||$);
- И — логическое умножение, конъюнкция ($\wedge$, *and*, $\&$, $\&\&$).

В алгебре логики возможны обозначения логического сложения и умножения знаками арифметического сложения и умножения, когда речь идет о высказываниях истина/ложь. Но такие обозначения невозможны в языках программирования, поэтому крайне нежелательны.

Законы алгебры логики:

- переместительный (коммутативность)

$$A \vee B = B \vee A,$$

$$A \wedge B = B \wedge A;$$

- сочетательный (ассоциативность)

$$(A \vee B) \vee C = A \vee (B \vee C),$$

$$(A \wedge B) \wedge C = A \wedge (B \wedge C);$$

- распределительный (дистрибутивность)

$$(A \vee B) \wedge C = (A \wedge C) \vee (B \wedge C),$$

$$(A \wedge B) \vee C = (A \vee C) \wedge (B \vee C);$$

- идемпотентности

$$A \vee A = A,$$

$$A \wedge A = A;$$

- свойства отрицания (комплементность)

$$A \vee \overline{A} = 1,$$

$$A \wedge \overline{A} = 0;$$

- снятия двойного отрицания (инволютивность отрицания)

$$\overline{\overline{A}} = A;$$

- инверсии (теоремы де Моргана)

$$\overline{A \vee B} = \overline{A} \wedge \overline{B},$$

$$\overline{A \wedge B} = \overline{A} \vee \overline{B};$$

- поглощения

$$A \vee (A \wedge B) = A,$$

$$A \wedge (A \vee B) = A.$$

Основные логические элементы выполняют три основных операции цифровой логики — И, ИЛИ, НЕ. С помощью этих элементов можно реализовать логические операции любой сложности. Элементы И и ИЛИ могут иметь 2 и более входов.

Элемент НЕ является *инвертором*. — Логическая 1 на выходе появляется тогда, когда на входе 0. Соединив два инвертора

последовательно получим *повторитель*. Оба элемента могут быть использованы для усиления и задержки логического сигнала.

Элемент И выполняет функцию *логического умножения (конъюнкции)*. Логическая 1 на выходе элемента И появляется при условии, когда на всех входах элемента будут сигналы логической 1. Добавление инвертора приводит к образованию элемента И-НЕ.

Элемент ИЛИ выполняет функцию *логического сложения (дизъюнкции)*. Логическая 1 на выходе появляется тогда, когда на любом его входе присутствует логическая 1. Добавление инвертора приводит к образованию элемента ИЛИ-НЕ.

Элемент ИСКЛЮЧАЮЩЕЕ_ИЛИ удовлетворяет условию *неравнозначности*. Логическая 1 на выходе элемента появляется тогда, когда на нечетном количестве входов присутствует логическая 1. Соединив элемент неравнозначности и инвертор получается элемент ИСКЛЮЧАЮЩЕЕ_ИЛИ-НЕ — элемент эквивалентности (равнозначности).

Условные изображения некоторых логических элементов приведены на рис. 7.1.

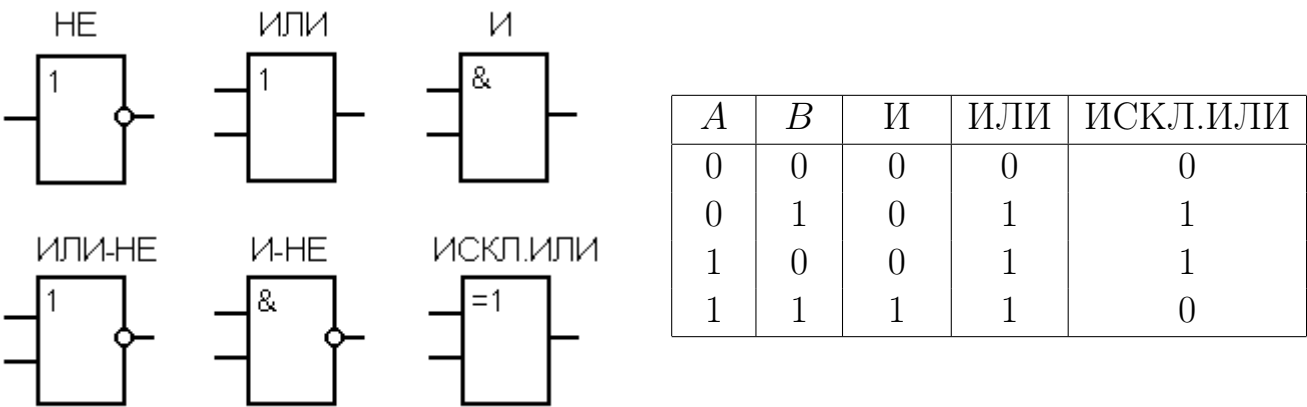


Рис. 7.1. Условные обозначения и таблица истинности некоторых логических элементов

Комбинационная логика (схема) — двоичная логика функционирования устройств комбинационного типа, когда состояние выхода однозначно определяется набором входных сигналов и не зависит от предыстории функционирования цифрового устройства. Это отличает комбинационную логику от *секвенциальной логики*.

Дешифратор или *декодер* — комбинационная схема, преобразую-

щая n -разрядный код на ее входах в одноединичный (позиционный) код. Логический сигнал активен на том выходе, порядковый номер которого соответствует коду.

Шифратор или *кодер* — комбинационная схема, выполняющая преобразование позиционного кода на ее входах в двоичный код, то есть реализующая обратную дешифратору функцию.

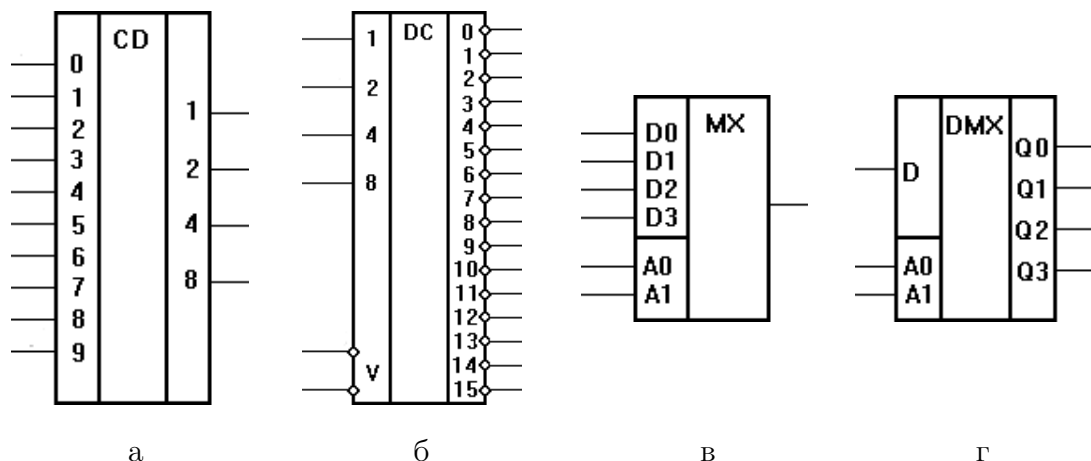


Рис. 7.2. Условные обозначения некоторых комбинационных схем (а — шифратор, б — дешифратор, в — мультиплексор, г — демультиплексор)

Мультиплексор — устройство, имеющий несколько сигнальных входов, один или более управляющих входов и один выход. Мультиплексор позволяет передавать сигнал с одного из входов на выход. Выбор желаемого входа осуществляется подачей соответствующей комбинации управляющих сигналов.

Аналоговый мультиплексор — электрически соединяют выбранный вход с выходом, обеспечивая низкое сопротивление между ними — порядка единиц Ом. Аналоговый мультиплексор иногда называют *коммутатором*, сигналы могут передаваться не только с входа на выход, но и в обратном направлении.

Цифровой мультиплексор — не образуют прямого электрического соединения между выбранным входом и выходом, а лишь передает с входа на выход логический уровень.

Демультиплексор — это устройство, в функциональном отношении противоположное цифровому мультиплексору, имеющее один сигнальный вход, один или более управляющих входов и несколько выходов. Демультиплексор передает сигнал с входа на один из

нескольких выходов в соответствии с комбинацией управляющих сигналов.

Мультиплексоры и демультимплексоры, как правило, строятся на основе дешифраторов. Условные графические изображения шифратора, дешифратора, мультиплексора и демультимплексора приведены на рис. 7.2.

Цифровой компаратор — схема, сравнивающая друг с другом два бинарных числа A и B и сообщающая является ли $A > B$, $A = B$ или $A < B$. Цифровые компараторы используются в центральных процессорах и микроконтроллерах.

Однобитовый компаратор — является самым простым цифровым компаратором, сравнивающим два однобитовых числа. Сигналы на выходах компаратора соответствуют выражениям:

$$\begin{aligned} X &= A \wedge \overline{B}; \\ Y &= (\overline{A} \wedge \overline{B}) \vee (A \wedge B); \\ Z &= \overline{A} \wedge B. \end{aligned} \quad (7.1)$$

$X = 1$ когда $A > B$, $Y = 1$ при $A = B$, а $Z = 1$ если $A < B$.

Для сравнения чисел с большей размерностью используется последовательное включение ряда однобитовых компараторов, имеющих дополнительный сигнал разрешения работы. Числа сравниваются поразрядно, начиная со старшего бита. Если биты совпадают, то разрешается сравнение следующих пар бит и так далее.

Преобразователь кодов — комбинационная схема, предназначенная для перевода чисел из одной формы записи в другую. Частным случаем такой схемы являются шифраторы и дешифраторы.

Переключатель два из трех — схема, выход которой имеет состояние 1 тогда, когда по меньшей мере 2 из 3 входов имеют состояние 1. Сигнал на выходе соответствует выражению:

$$Z = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C). \quad (7.2)$$

Схема применяется в системах повышенной надежности для сравнения сигналов, полученных от трех независимых источников.

Пороговая логическая схема — схема, в которой определенное минимальное количество переменных должно иметь состояние 1,

чтобы на выходе была 1. Переключатель два из трех является частным случаем такой схемы.

Схема контроля четности — комбинационная схема, выход которой принимает значение логической единицы если четное число входов имеют состояние 1.

Под *полной конъюнкцией* (*дизъюнкцией*) понимают операцию логического умножения (сложения), в которой участвуют все логические переменные или их инвертированные значения.

Нормальная форма ИЛИ состоит из нескольких полных конъюнкций, которые логически складываются операцией ИЛИ. Она может состоять также из одной-единственной полной конъюнкции.

Нормальная форма И состоит из нескольких полных дизъюнкций, которые логически перемножаются операцией И. Она может состоять также из одной-единственной полной дизъюнкции.

Используя нормальную форму ИЛИ можно синтезировать полную комбинационную схему, удовлетворяющую заданной таблице истинности. Количество единичных состояний равно количеству полных дизъюнкций. Такая схема часто может являться неоптимальной и может быть упрощена с помощью алгебры логики. То же относится и к нормальной форме И.

7.2. Метод упрощения логических функций

Метод *карт Карно* (*диаграмм Карно*) — служит для наглядного представления и упрощения нормальной формы ИЛИ. Карты Карно представляют таблицы истинности для полных конъюнкций. Единица в поле карты соответствует наличию полной конъюнкции. Например, выражение

$$Z = (\bar{A} \wedge B) \vee (A \wedge B) \vee (\bar{A} \wedge \bar{B}) \quad (7.3)$$

представлено на рис. 7.3,а.

Идея метода заключается в поиске соседних конъюнкций и объединения их в группы. В нашем примере мы можем выделить две пары соседних конъюнкций. Первая образована двумя конъюнкциями по

	A	$\bar{A}$
B	1	1
$\bar{B}$		1

а

	A		$\overline{A}$	
B			1	1
$\overline{B}$	1			1
	$\overline{C}$	C	$\overline{C}$	

б

Рис. 7.3. Карта Карно (а — для двух переменных, б — для трех переменных)

одной горизонтали: $(A \wedge B)$ и $(\bar{A} \wedge B)$, которые можно заменить сигналом B . Вторая — по одной вертикали: $(\bar{A} \wedge B)$ и $(\bar{A} \wedge \bar{B})$, которые заменяются сигналом $\bar{A}$. В итоге, упрощенное выражение принимает вид $Z = \bar{A} \vee B$.

Для трех переменных возможно 8 различных полных конъюнкций, следовательно карта Карно должна иметь 8 клеток. Она имеет цилиндрическую форму, поэтому клетки, находящиеся в начале и в конце одной строки считаются соседними.

На рис. 7.3,б приведена карта Карно для выражения

$$Z = (\bar{A} \wedge B \wedge C) \vee (\bar{A} \wedge B \wedge \bar{C}) \vee (\bar{A} \wedge \bar{B} \wedge \bar{C}) \vee (A \wedge \bar{B} \wedge \bar{C}). \quad (7.4)$$

Каждая группа может состоять из двух или четырех полных конъюнкций. Случай с 8 полными конъюнкциями тривиален — это бы означало наличие логической 1 всегда, вне зависимости от всех сигналов на входе.

В нашем примере можно образовать 3 группы из двух клеток. Вверху справа выделяется группа $(\bar{A} \wedge B)$, внизу — $(\bar{B} \wedge \bar{C})$, и справа можно выделить группу $(\bar{A} \wedge \bar{C})$. Третья группа является избыточной, так как первыми двумя группами все 1 уже охвачены. В результате упрощения получается выражение $Z = (\bar{A} \wedge B) \vee (\bar{B} \wedge \bar{C})$.

Карта Карно для четырех переменных состоит из 16 клеток, а каждая группа может состоять из двух, четырех или 8 полных конъюнкций (случай с 16 тривиален). Для такой карты возможны группировки по принципу расширенного соседства. Примеры таких группировок, объединяющих по 4 полные конъюнкции представлены на рис. 7.4.

Карты Карно для пяти переменных можно представить, например,

	A	$\bar{A}$		
B			1	
$\bar{B}$	1			1
	$\bar{C}$	C	$\bar{C}$	

	A	$\bar{A}$		
B	1			1
$\bar{B}$				
	$\bar{C}$	C	$\bar{C}$	

	A	$\bar{A}$		
B		1	1	
$\bar{B}$				
	$\bar{C}$	C	$\bar{C}$	

Рис. 7.4. Карты Карно для четырех переменных; показана группировка по принципу расширенного соседства

в виде двух квадратных полей по 16 клеток, а для шести — в виде четырех таких полей. Также возможно представление в виде объемных фигур. Работать с такими картами весьма сложно.

7.3. Комбинационные логические схемы в ПЛИС

Программируемые логические интегральные схемы (стр. 61) могут быть использованы для построения комбинационных логических схем. В простейших ПЛИС — CPLD содержится набор элементов И, ИЛИ, НЕ и коммутационная матрица, позволяющая соединять элементы между собой.

Функционально в ПЛИС существуют макроячейки. Каждая макроячейка включает в себя 5 элементов И, выходы которых могут быть присоединены к элементу ИЛИ. На входы любого элемента И можно подать любую комбинацию прямых или инверсных сигналов с выходов других макроячеек и входов микросхемы. Таким образом макроячейка способна реализовывать нормальную форму ИЛИ. Соседние макроячейки можно объединять, увеличивая число элементов И в нормальной форме ИЛИ. Однако следует помнить, что ресурсы ПЛИС ограничены, поэтому имеет смысл проектировать схему таким образом, чтобы она требовала минимальное количество логических элементов.

На языке программирования аппаратуры VHDL логические функции могут быть записаны следующим образом: НЕ — not, И — and, И-НЕ — nand, ИЛИ — or, ИЛИ-НЕ — nor, ИСКЛЮЧАЮЩЕЕ_ИЛИ

— xor, ИСКЛЮЧАЮЩЕЕ_ИЛИ_НЕ — xnor.

Одним из вариантов создания схемы на основе ПЛИС является ее схемотехническое описание. Программа ISE использует зарубежный стандарт отображения логических элементов, отличающийся от российского. Условные изображения некоторых логических элементов приведены на рис. 7.5.

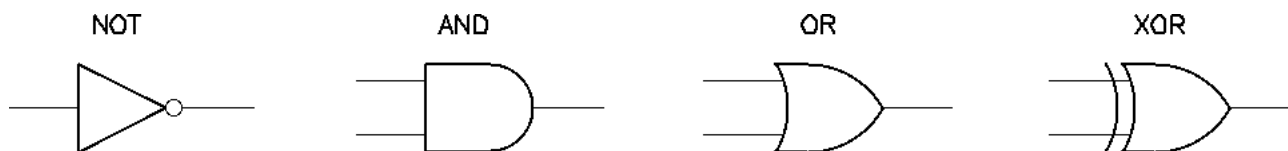


Рис. 7.5. Условные обозначения некоторых логических элементов в иностранном стандарте

7.4. Практическая часть

Перед началом выполнения упражнений необходимо на примере готового проекта освоить работу со средой ISE, программатором и макетной платой (стр. 63). Описание работы с ISE подробно рассмотрено в [3] (глава 3).

Задача выполняется на макетной плате с установленной ПЛИС XC9572XL (стр. 66). Для каждого упражнения необходимо создать собственный проект, используя схемотехническое описание. Порядок создания проекта описан в приложении на стр. 69.

Упражнения (задания с буквами выполняются по вариантам):

1. Получить функции 2И, 2ИЛИ, 2ИСКЛЮЧАЮЩЕЕ_ИЛИ из набора элементов 2И-НЕ. Входы всех трех схем подключить к кнопкам 1 и 2. Выходы — к светодиодам 1 – 3 (каждую схему к своему светодиоду). Записать таблицу истинности каждого элемента.
2. Собрать дешифратор на 4 состояния на основных логических элементах. Входы подключить к кнопкам 1 и 2, выходы — к светодиодам 1 – 4. Записать таблицу истинности.
3. Собрать схему двухбитового компаратора на основных логических элементах. Входы подключить к кнопкам 1 и 2 (первое

число) и 3 и 4 (второе число), выходы — к светодиодам 1 – 3. Записать таблицу истинности схемы.

- 4а. Собрать схему переключателя два из трех на основных логических элементах. Входы подключить к кнопкам 1 – 3, выход — к светодиоду 1. Записать таблицу истинности схемы.
- 4б. Собрать схему контроля четности для 4-битного входного числа на основных логических элементах. Входы подключить к кнопкам 1 – 4, выход — к светодиоду 1. Записать таблицу истинности схемы.

7.5. Контрольные вопросы

1. Что такое алгебра логики, основные логические операции и законы?
2. Какие вы знаете основные логические элементы? Как они обозначаются?
3. Чем отличаются комбинационная и секвенциальная логика?
4. Что такое нормальная форма ИЛИ, нормальная форма И?
5. Как работает метод упрощения логической функции с использованием карт Карно?
6. Чем отличаются шифратор, дешифратор и преобразователь кодов?
7. Чем отличаются мультиплексор, демultipлексор, коммутатор?
8. В чем отличия между пороговой логической схемой, переключателем два из трех, схемой контроля четности?
9. Что такое цифровой компаратор? Как сравниваются числа большей разрядности?

Задача 8.

Триггеры

Задача посвящена изучению принципов работы простейших триггеров и их построению на основе логических элементов.

8.1. Основные понятия

Триггерами называются логические схемы, имеющие несколько стабильных состояний, зависящих не только от сигналов на входах, но и от предыдущего состояния схемы, то есть обладающими эффектами памяти. В цифровой технике применяются триггеры, имеющие 2 стабильных состояния, называемые *двоичными триггерами*. Обычно двоичные триггеры имеют два выхода — прямой Q и инверсный $\overline{Q}$. Триггеры управляются сигналами, подаваемыми на информационные (и если есть тактовый) входы.

Информационный сигнал — сигнал, подаваемый на информационный вход триггера.

Нетактируемые или *асинхронные триггеры* изменяют свое состояние после прихода соответствующих информационных сигналов.

Тактируемые или *синхронные триггеры* реагируют на информационные сигналы только при наличии (или в момент прихода) тактового импульса на тактовом входе. Этот вход обычно обозначается буквой C (от слова Clock).

Синхронные триггеры со *статическим управлением* — тактируемые уровнем напряжения (уровнем синхроимпульса), воспринимают информационные сигналы только при наличии уровня логической единицы на тактовом входе (прямой вход C) или логического нуля

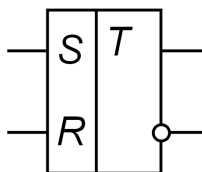
(инверсный вход $\bar{C}$).

Синхронные триггеры с *динамическим управлением* — тактируемые перепадом напряжения (фронтом синхроимпульса), воспринимают информационные сигналы только при изменении уровня на тактовом входе с 0 на 1 (прямой динамический вход C , тактирование положительным фронтом) или с 1 на 0 (инверсный динамический вход $\bar{C}$, тактирование спадом или отрицательным фронтом).

На принципиальных схемах триггеры изображают прямоугольниками, внутри которых ставятся буквы Т или ТТ, обозначающие сколько запоминающих ячеек входит в данный триггер. На специальном поле слева обозначаются входы: S, R, J, K, C и др.

Инверсные (статические) входы и выходы обозначаются кружочком ($\circ$). Динамически управляемые входы обозначаются или треугольничком или наклонной чертой. Символы $\triangleright$ или $\nearrow$ обозначают управление положительным фронтом, $\triangleleft$, $\nwarrow$ или ∇ — отрицательным фронтом.

8.2. Асинхронный и синхронный RS-триггеры



S	R	$Q(t+1)$	$\bar{Q}(t+1)$	Состояние
0	0	$Q(t)$	$\bar{Q}(t)$	хранение
1	0	1	0	установка
0	1	0	1	сброс
1	1	-	-	запрещенное

Рис. 8.1. Условное обозначение и таблица истинности асинхронного RS-триггера

Асинхронный RS-триггер (рис. 8.1) имеет 2 информационных входа — *установки* (Set) и *сброса* (Reset). При отсутствии команды сброса и установки, то есть когда на обоих входах логический 0 — триггер находится в состоянии *хранения*. Логическая 1 на входе S при отсутствии логической 1 на R (сигнал установки активен, сигнал сброса отсутствует) приводит к установке триггера в единичное состояние, то есть $Q=1$, $\bar{Q}=0$. Если присутствует сигнал сброса, а сигнал установки отсутствует — триггер переходит в нулевое

состояние, когда $Q=0$, $\overline{Q}=1$. Одновременная подача и сигнала установки и сигнала сброса переводит триггер в *запрещенное состояние*, при этом сигналы на выходах триггера зависят от его внутреннего устройства. Простейшие реализации асинхронного RS-триггера на двух элементах 2ИЛИ-НЕ (2И-НЕ) приводят к тому, что прямой и инверсный выходы принимают одинаковые значения логического 0 (логической 1 для триггера на 2И-НЕ). Переход из запрещенного состояния в состояние хранения невозможен — триггер переходит или в нулевое, или в единичное состояние, а только потом в состояние хранения.

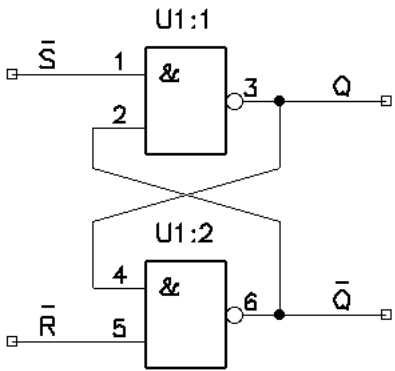
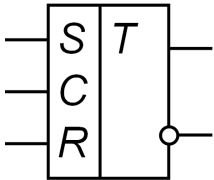


Рис. 8.2. Асинхронный $\overline{RS}$ -триггер на элементах 2И-НЕ

В положительной логике RS-триггер собирается из двух элементов 2ИЛИ-НЕ, в отрицательной — из двух 2И-НЕ. Однако работать с отрицательной логикой не всегда удобно, поэтому проще рассматривать триггер на элементах 2И-НЕ как $\overline{RS}$ -триггер, то есть триггер, у которого управляющими сигналами является не логическая 1, а логический 0. Пример такого триггера изображен на рис. 8.2.

Для надежного срабатывания RS-триггера, построенного на двух логических элементах длительность управляющих импульсов должна превышать удвоенное время задержки (быстродействие) одного элемента.



C	S	R	$Q(t+1)$	$\overline{Q}(t+1)$	Состояние
0	x	x	$Q(t)$	$\overline{Q}(t)$	хранение
1	0	0	$Q(t)$	$\overline{Q}(t)$	хранение
1	1	0	1	0	установка
1	0	1	0	1	сброс
1	1	1	-	-	запрещенное

Рис. 8.3. Условное обозначение и таблица истинности синхронного RS-триггера

Синхронный RS-триггер (рис. 8.3) имеет 2 информационных входа — установки и сброса, а также тактовый вход C. При отсутствии тактового импульса ($C=0$) или отсутствии команд сброса и установки

($R=S=0$) — триггер находится в состоянии хранения. Единичное состояние триггер принимает при одновременном наличии тактового сигнала и сигнала установки ($S=C=1, R=0$). Нулевое состояние — при наличии тактового сигнала и сигнала сброса ($R=C=1, S=0$). Запрещенное состояние триггера возникает при наличии всех трех сигналов ($S=R=C=1$), при этом поведение триггера аналогично поведению асинхронного RS-триггера в запрещенном состоянии.

Синхронный RS-триггер может быть собран из четырех элементов 2И-НЕ (в положительной логике), его схема приведена на рис. 8.4. Для надежного срабатывания такого триггера длительность управляющих импульсов должна превышать утроенное время задержки одного элемента.

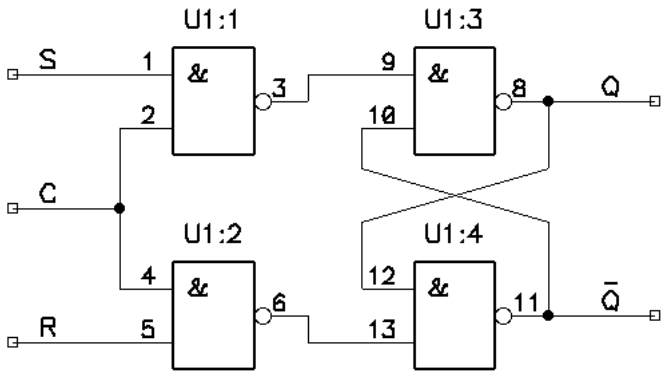
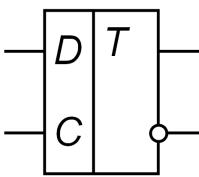


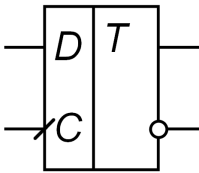
Рис. 8.4. Синхронный RS-триггер на элементах 2И-НЕ

8.3. Синхронный D-триггер



C	D	$Q(t+1)$	$\overline{Q}(t+1)$	Состояние
0	x	$Q(t)$	$\overline{Q}(t)$	хранение
1	0	0	1	сброс
1	1	1	0	установка

а



D	$Q(t+1)$	$\overline{Q}(t+1)$	Состояние
0	0	1	сброс
1	1	0	установка

б

Рис. 8.5. Условное обозначение и таблица истинности синхронного D-триггера, (а — статическое управление, б — динамическое управление)

Синхронный одноступенчатый D-триггер со статическим управлением (рис. 8.5,а) имеет один информационный вход данных (Data)

и тактовый вход C . При отсутствии тактового импульса ($C=0$) — триггер находится в состоянии хранения. Единичное состояние триггер принимает при одновременном наличии тактового сигнала и логической 1 на информационном входе D ($C=D=1$). Нулевое состояние — при наличии тактового сигнала и логического 1 на информационном входе ($C=1, D=0$).

Иными словами, при наличии тактового импульса ($C=1$) состояние триггер напрямую определяется значением на информационном входе D , то есть сигнал проходит сквозь триггер. Такое состояние называется *состоянием прозрачности*. При отсутствии тактового импульса ($C=0$) триггер переходит в состояние хранения.

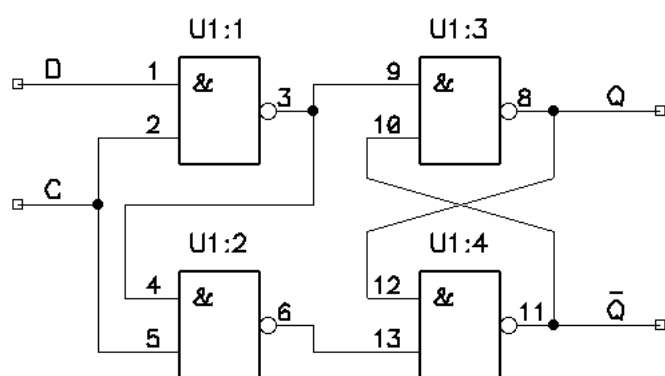


Рис. 8.6. Синхронный D-триггер со статическим управлением на элементах 2И-НЕ

Синхронный D-триггер со статическим управлением может быть собран из четырех элементов 2И-НЕ. Пример такого триггера изображен на рис. 8.6.

Синхронный D-триггер с динамическим управлением (рис. 8.5,б) имеет один информационный вход данных ($Data$) и динамический тактовый вход C .

Триггер постоянно находится в состоянии хранения, за исключением момента времени прихода очередного синхроимпульса, то есть смены состояния логического 0 на 1 на входе C . Состояния прозрачности триггер не имеет. Таблица истинности отображает поведение триггера в момент прихода тактового импульса.

8.4. Синхронный JK-триггер

Синхронный динамический JK-триггер имеет 2 информационных входа J и K , а также динамический тактовый вход C . Условное обозначение JK триггера, управляемого фронтом и его таблица истинности приведены на рис. 8.7. Триггер постоянно находится в состоянии хранения, кроме момента прихода очередного синхроим-

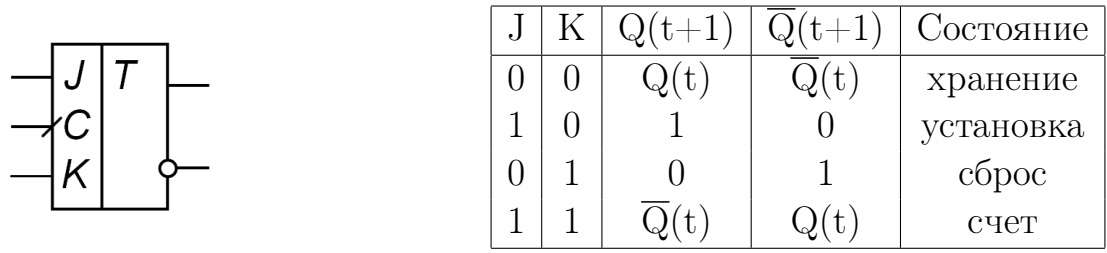


Рис. 8.7. Условное обозначение и таблица истинности синхронного JK-триггера

пульса. В этот момент времени поведение триггера определяется таблицей истинности. Когда оба информационных входа неактивны, то даже в момент прихода тактового импульса триггер продолжает находиться в состоянии хранения. Если триггер находился в нулевом состоянии, а на информационном входе J была логическая 1 — триггер переходит в единичное состояние. Если триггер находился в единичном состоянии, а на K была логическая 1 — в нулевое.

На основе стандартного JK-триггера могут быть созданы более сложные триггеры. Примером является JK-триггер, управляемый отрицательным фронтом и имеющий дополнительные асинхронные инверсные входы сброса и установки (рис. 8.8). Входы R и S имеют приоритет над логикой работы JK-триггера, то есть триггер работает как JK только когда оба сигнала R и S неактивны.

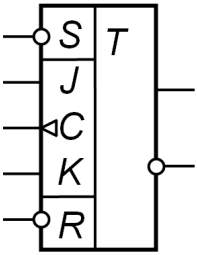


Рис. 8.8. Условное обозначение JK-триггера с дополнительными инверсными входами R и S

8.5. Асинхронный и синхронный Т-триггеры

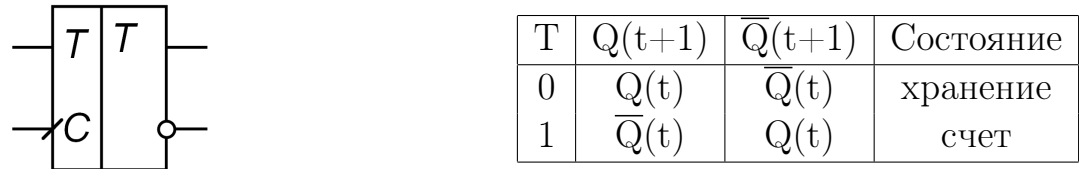


Рис. 8.9. Условное обозначение и таблица истинности синхронного Т-триггера

Асинхронный Т-триггер имеет только тактовый динамический вход и переключается на противоположное значение при приходе каждого тактового импульса. Такой триггер получается, например,

из синхронного D-триггера с динамическим управлением при соединении информационного входа D с инверсным выходом $\overline{Q}$. Также асинхронный T-триггер можно получить из JK-триггера, подав на входы J и K логическую 1.

Синхронный T-триггер отличается от асинхронного наличием входа T, разрешающего счет. Такой триггер получается из JK-триггера путем соединения J и K входов между собой (в общий вход T). Условное обозначение и таблица истинности синхронного T-триггера приведена на рис. 8.9.

T-триггер часто применяют для понижения частоты в 2 раза, при этом на T вход подают единицу, а на C — сигнал с частотой, которая будет поделена на 2. T-триггер служит основой для создания простейших счетчиков, поэтому его часто называют *счетным триггером*.

8.6. VHDL описание триггеров

Краткое описание языка VHDL приведено на стр. 77.

Работа простейших триггеров в языке VHDL может быть записана путем описания функций логических элементов, а синхронные триггеры с динамическим тактированием можно описать в виде процесса, например:

```
-- RS-триггер
Q  <= not ( R or neQ );
neQ <= not ( S or Q );
-- D-триггер, работающий по фронту синхроимпульса
process (C)
begin
    if(C'event and C = '1') then
        Q <= D;
    end if;
end process;
-- T-триггер, работающий по спаду синхроимпульса
process (C)
begin
    if(C'event and C = '0' and T = '1') then
        Q <= not Q;
    end if;
end process;
```

Правильно описанный триггер с динамическим тактированием будет выполнен на триггерах, имеющихся в ПЛИС, что гарантирует высокую скорость работы проекта.

Входы и выходы ПЛИС описываются сигналами типа STD_LOGIC внутри entity...port. Затем идет секция architecture с ссылкой на имя секции entity. Пример:

```
entity trigger is
    port (bot1, bot2 : in  STD_LOGIC;
          led1, led2 : out STD_LOGIC);
end trigger;

architecture Behavioral of trigger is
-- здесь объявляются внутренние сигналы, если они необходимы
begin
    -- здесь пишется логика работы схемы
end Behavioral;
```

Следует помнить, что значения входных сигналов можно только считывать, а выходам — только присваивать значения (считать значения выхода нельзя). Внутренние сигналы внутри архитектуры можно и устанавливать и считывать. Если схеме требуется считывание значения выходного сигнала, то следует добавить промежуточный сигнал, а его значения уже присвоить выходу:

```
-- после architecture Behavioral of ....., но перед begin
SIGNAL Q    : STD_LOGIC := '0';
SIGNAL neQ  : STD_LOGIC := '1';

-- после architecture .... и begin, но перед end Behavioral
led1 <= Q;
led2 <= neQ;
```

8.7. Практическая часть

Задача выполняется на макетной плате с установленной ПЛИС XC9572XL (стр. 66). Для каждого упражнения необходимо создать собственный проект. Упражнения, выполняемые с использованием схемотехнического описания (см. стр. 69):

1. Собрать схему асинхронного RS-триггера (рис. 8.2). Входы $\bar{R}$ и $\bar{S}$ подключить к кнопкам 1 и 2. Выходы — к светодиодам 1 и 2. Записать таблицу истинности.

2. Собрать схему синхронного RS-триггера (рис. 8.4). Входы R, S и C подключить к кнопкам 1 – 3. Выходы — к светодиодам 1 и 2. Найти состояния хранения. Записать таблицу истинности.
3. Собрать синхронный D-триггер, управляемый уровнем (рис. 8.6). Входы D и C подключить к кнопкам 1 и 2, выходы — к светодиодам 1 и 2. Записать таблицу истинности.
- 4а. Собрать D-триггер, управляемый фронтом. Вход D подключить к кнопке 1, вход C — к кнопке 2. Выходы D-триггера — к светодиодам 1 и 2. Записать таблицу истинности.
- 4б. Собрать JK-триггер, управляемый отрицательным фронтом. Входы J и K подключить к кнопкам 1 и 2, вход C — кнопке 4. Выходы JK-триггера — к светодиодам 1 и 2. Записать таблицу истинности, при этом учитывать эффектдребезга контактов кнопки тактирования — в момент нажатия происходит переходный процесс, формирующий некую последовательность случайных импульсов, что будет заметно в состоянии, когда сигналы J и K одновременно активны.

Упражнения, выполняемые на языке VHDL (см. стр. 70):

5. Собрать RS-триггер и синхронный RS-триггер. Входы R и S обоих триггеров подключить к кнопкам 1 и 2, а вход C синхронного триггера — к кнопке 3. Выходы Q и $\bar{Q}$ первого триггера — к светодиодам 1 и 2, второго — 3 и 4.
- 6а. Собрать одновременно следующие схемы: D-триггер, управляемый уровнем, входы C и D подключить к кнопкам 1 и 2, выход к светодиоду 1; D-триггер, управляемый фронтом, входы C и D подключить к кнопкам 1 и 2, выход к светодиоду 2; T-триггер, вход C подключить к выходу второго D-триггера, T — к кнопке 3, выход — к светодиоду 3.
- 6б. Собрать JK-триггер со сбросом. Входы C, R, J и K — к кнопкам 1 — 4. Выходы — к светодиодам 3 и 4. Составить таблицу истинности JK триггера, не забывая про эффектдребезга контактов.

8.8. Контрольные вопросы

1. Каким образом в сложной логической схеме можно выявить наличие триггеров?
2. Каковы известные вам виды триггеров?
3. Что такое запрещенное состояние, и в каких триггерах оно возможно?
4. Какова классификация триггеров по способу тактирования?
5. Каково отличие триггера, управляемого по фронту, от управляемого по уровню сигнала?
6. В чем особенность построения таблицы истинности для триггеров?

8.9. Контрольные задания

1. Составить таблицу истинности для триггера, имеющего прямые или инверсные входы R и S , выходы Q и $\overline{Q}$.
2. Составить таблицу истинности для триггера, имеющего прямые или инверсные входы R , S и C (тактирование уровнем).
3. Составить таблицу истинности для триггера с прямыми или инверсными входами D и C (тактирование уровнем или фронтом).
4. Составить таблицу истинности JK триггера, тактируемого положительным или отрицательным фронтом, имеющего входы J , K , $\overline{R}$ и $\overline{S}$.
5. Составить таблицу истинности JK триггера, тактируемого положительным или отрицательным фронтом, имеющего входы $J1$, $K1$, $\overline{J2}$, $\overline{K2}$.

Задача 9.

Регистры и формирователи кода

Задача посвящена изучению работы регистров и схем на их основе, таких как формирователи детерминированных и случайных бинарных последовательностей.

9.1. Основные понятия

Регистры — группа определенным образом соединенных триггеров, предназначенных для временного запоминания многоразрядных чисел и выполнения преобразований над ними. Различают последовательные, параллельные и параллельно-последовательные регистры. Триггеры, входящие в состав регистра называют *разрядами*.

Параллельные регистры или *регистры хранения* предназначены для сохранения бинарных слов (чисел) в течение определенного времени и могут быть построены на основе синхронных триггеров со статическим или динамическим управлением, тактовые входы которых объединены. Все разряды числа записываются в регистр одновременно, по сигналу синхроимпульса.

Часто выходы многоразрядных регистров дополняют схемой, отключающей их от внешней цепи при подаче определенного сигнала на отдельный управляющий вход — переводящей выходы в *третье состояние*. Это позволяет организовать работу нескольких регистров на одну общую шину данных.

Регистр хранения со статическим тактовым входом пропускает числа с входа на выход при наличии тактового импульса и запоминает последнее значение числа при пропадании синхроимпульса.

Регистр с динамическим тактовым входом — защелкивает новое значение по фронту или спаду синхроимпульса.

Последовательные или *сдвигающие регистры* принимают информацию последовательно во времени, бит за битом, сохраняют в течение определенного времени и передают далее. Такие регистры могут быть построены на синхронных триггерах с динамическим тактовым входом. Информационный вход каждого триггера, за исключением первого, подключается к выходу предыдущего разряда, а вход первого — является информационным входом (входом переноса) сдвигающего регистра. Тактовые входы всех триггеров объединяются. Выходом регистра является выход последнего разряда.

Сдвигающий регистр с параллельным выводом данных имеет выходы не только от старшего, а от всех разрядов, что позволяет передавать все разряды записанного в регистр числа и последовательно (используя только старший разряд) и одновременно.

Сдвигающий регистр с параллельным вводом данных имеет дополнительные информационные входы для каждого разряда. Он может работать в режимах параллельной и последовательной записи, переключаемых отдельным управляющим сигналом.

Реверсивный сдвигающий регистр — сдвигающий регистр, направление сдвига информации в котором может меняться при помощи управляющего сигнала.

Кольцевой сдвигающий регистр — сдвиговый регистр, вход которого замкнут на выход, обеспечивает движение по кругу предустановленной комбинации, для чего должен иметь возможность параллельной или последовательной записи в него данных.

Кольцевой сдвигающий регистр с инверсией — сдвиговый регистр, вход которого соединен с выходом через инвертор. Если начальное состояние всех триггеров нулевое, то на выходах регистра формируются набор сдвинутых по фазе меандров.

Сдвигающий регистр с линейной обратной связью — регистр сдвига, у которого входной (вдвигаемый) бит является линейной функцией состояния остальных битов регистра до сдвига. Применяется для генерации псевдослучайных битовых последовательностей.

Периодом регистра сдвига называют минимальную длину получаемой последовательности до начала ее повторения.

9.2. VHDL описание регистров

Сигналы, используемые в работе регистра удобно объявить как вектор необходимой размерности:

```
SIGNAL x : STD_LOGIC_VECTOR(3 downto 0) := "0000";
```

где выражение "3 downto 0" задает диапазон бит вектора (в нашем примере 4 бита), а "0000" — начальное значение вектора при включении питания. В вектора можно объединять не только внутренние сигналы, но также входные и выходные линии данных.

Регистр хранения, построенный на синхронных триггерах со статическим или динамическим управлением описывается в виде присвоения внутри определенного условия:

```
-- регистр хранения со статическим управлением
process (Clk, d)
begin
    if(Clk = '1') then
        Q <= d;
    end if;
end process;
```

```
-- регистр хранения с динамическим управлением
process (Clk)
begin
    if(Clk'event and Clk = '1') then
        Q <= d;
    end if;
end process;
```

где d — входной вектор данных, Q — выходной вектор.

Получить значение конкретного бита вектора можно указав номер бита в скобках, а нескольких подряд расположенных бит — указав диапазон:

```
x(0)  -- младший разряд
x(3 downto 1) -- разряды x3, x2 и x1
```

Объединение нескольких сигналов в вектор или нескольких векторов в вектор большей размерности выполняется при помощи операции конкатенации:

```
SIGNAL a, b, c : STD_LOGIC;
SIGNAL x, y    : STD_LOGIC_VECTOR(7 downto 0);

'1' & '0' & '0' & '0'      -- четырехразрядный вектор "1000"
a & b & c                  -- вектор "abc"
x(3 downto 0) & y (1) & a  -- шестиразрядный вектор
```

Работу сдвигающего регистра можно описать используя обращение к определенным битам вектора:

```
Q <= Q(2 downto 0) & V;      -- сдвигающий регистр
Q <= Q(2 downto 0) & Q(3); -- кольцевой сдвигающий регистр
```

где V — вход переноса.

Кроме того, в некоторых случаях удобно использовать операции сдвига:

- $\text{sll}(X, n)$ и $\text{srl}(X, n)$ осуществляют логический сдвиг — возвращают значение X после сдвига n раз влево и вправо соответственно, а на место появившегося бита всегда записывается бит 0 (умножение и деление беззнакового числа на 2).
- $\text{sla}(X, n)$ и $\text{sra}(X, n)$ осуществляют арифметический сдвиг значения X n раз, при сдвиге влево ведет себя как логический сдвиг, а при сдвиге вправо на место появившегося бита устанавливается бит, соответствующий знаку целого числа в дополнительном коде, то есть значение старшего бита сохраняется (умножение и деление знакового числа на 2).
- $\text{rol}(X, n)$ и $\text{ror}(X, n)$ осуществляют циклический сдвиг значения X n раз, уходящий бит переходит на место появившегося.

Значение n может быть целым числом. Если значение n отрицательно, то выполняется сдвиг в противоположную сторону, например $\text{sll}(X, -n)$ эквивалентно $\text{srl}(X, n)$ и наоборот. Аналогично ведут себя пары sla и sra , rol и ror .

Таблица 9.1. Коды Баркера

N	Последовательность
2	1 -1
3	1 1 -1
4	1 1 -1 1
5	1 1 1 -1 1
7	1 1 1 -1 -1 1 -1
11	1 1 1 -1 -1 -1 1 -1 -1 1 -1
13	1 1 1 1 1 -1 -1 1 1 -1 1 -1 1

9.3. Формирование циклического кода

Код Баркера — это числовая последовательность, состоящая из символов вида $a_i = \pm 1$, где $i = 1, 2, \dots, N$. Коды Баркера существуют только для ограниченного числа значений N и приведены в таблице 9.1. Последовательности Баркера имеют минимальный уровень боковых лепестков автокорреляционной функции из всех возможных последовательностей данной длины, поэтому код Баркера на 11 часто используется в системах передачи данных по радиоканалам.

Автокорреляционная функция (АКФ) периодического кода Баркера имеет вид:

$$r(n) = \begin{cases} 1 & (\text{для } n = 0) \\ 0 & (\text{для } 0 < n < N, N \text{ четное}) \\ \pm 1/N & (\text{для } 0 < n < N, N \text{ нечетное}) \end{cases}, \quad (9.1)$$

где $n = 0, 1, \dots, N-1$, а знак в последней строке зависит от длины последовательности N . Для непериодического кода Баркера:

$$r(n) = \begin{cases} 1 & (\text{для } n = 0) \\ 0 & (\text{для } 0 < n < N, n = N-2, N-4, \dots) \\ \pm 1/N & (\text{для } 0 < n < N, n = N-1, N-3, \dots) \end{cases}, \quad (9.2)$$

знак в последней строке зависит от длины последовательности N , а для $N = 4$ также от n .

При генерации кода Баркера вместо значений 1 и -1 используется логические 0 и 1.

Формирование любого постоянного кода можно выполнить при помощи сдвигающего регистра соответствующей разрядности, записав в него необходимый код и вращая его. Такой подход требует большого числа триггеров, а также возможности их предустановки в нужное начальное значение.

Другим способом является применение кольцевого сдвигающего регистра с инверсией и логической функции. В этом случае число триггеров можно сократить вдвое, а в качестве начального значения во все триггеры записать нули. Диаграмма работы такого регистра приведена на рис. 9.1. Сигналы на выходах регистра повторяются с периодом, вдвое большим числа разрядов. Формирование кода можно осуществить при помощи нормальной формы ИЛИ.

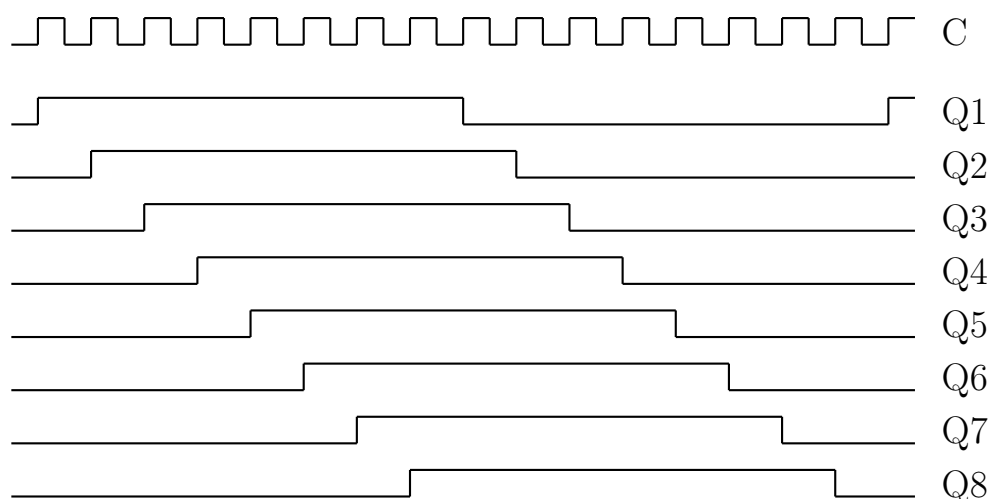


Рис. 9.1. Диаграмма работы восьмиразрядного кольцевого сдвигающего регистра с инверсией

В случае необходимости получения нечетного периода повторения сигналов следует применить сдвигающий регистр с линейной обратной связью, использующий для формирования входного бита сигналы с двух выходов регистра, например с двух старших его разрядов. При этом можно получить период повторения на единицу меньший удвоенного числа триггеров.

9.4. М-последовательность

М-последовательность или *последовательность максимальной длины* (*Maximum length sequence*) — псевдослучайная двоичная после-

довательность, порожденная регистром сдвига с линейной обратной связью и имеющая максимальный период. М-последовательности применяются в широкополосных системах связи.

Различные М-последовательности обладают следующими свойствами:

- М-последовательности являются периодическими с периодом $N = 2^n - 1$;
- количество символов, принимающих значение единица, на длине одного периода М-последовательности на единицу больше, чем количество символов, принимающих значение ноль;
- любые комбинации символов длины n на длине одного периода М-последовательности за исключением комбинации из n нулей встречаются не более одного раза, комбинация из n нулей является запрещенной — на ее основе может генерироваться только последовательность из одних нулей;
- сумма по модулю 2 любой М-последовательности с ее произвольным циклическим сдвигом также является М-последовательностью;
- периодическая автокорреляционная функция любой М-последовательности имеет постоянный уровень боковых лепестков, равный $-1/N$;
- автокорреляционная функция усеченной М-последовательности, под которой понимается непериодическая последовательность длиной в период N , имеет величину боковых лепестков, близкую к $-1/\sqrt{N}$, поэтому с ростом N величина боковых пиков уменьшается.

М-последовательности, построенные на основе n -разрядного регистра с линейной обратной связью имеют полиномы вида $x^n + \dots$, где степени полиномов приведены в таблице 9.2 (старший разряд используется во всех полиномах). Например, для $n = 3$ число 5 означает $5 = 2^2 + 2^0$, что соответствует полиному $x^3 \text{ xor } x$.

Таблица 9.2. Числа, определяющие полиномы для построения М-последовательностей.

n	Степени полинома
3	5, 6
4	9, 12
5	18, 20, 23, 27, 29, 30
6	33, 45, 48, 51, 54, 57
7	65, 68, 71, 72, 78, 83, 85, 92, 95, 96, 101, 105, 106, 114, 119, 120, 123, 126
8	142, 149, 150, 166, 175, 177, 178, 180, 184, 195, 198, 212, 225, 231, 243, 250
9	264, 269, 272, 278, 281, 300, 303, ..., 483, 485, 486, 490, 492, 497, 500, 509
10	516, 525, 531, 534, 562, 567, 576, ..., 958, 966, 969, 984, 1005, 1017, 1020
11	1128, 1136, 1188, 1192, 1280, 1300, ..., 2006, 2010, 2022, 2025, 2034, 2036
12	3592, 3586, 3232, 3144, 2820, 2584, ..., 3987, 4024, 4044, 4061, 4062, 4068

9.5. Практическая часть

Задача выполняется на макетной плате с установленной ПЛИС XC9572XL (стр. 66). Для каждого упражнения необходимо создать собственный проект (на языке VHDL). Порядок работы с симулятором ISim (упражнения 3а и 3б) описан в приложении на стр. 65.

Параметры: формируемый код, N — количество элементов кода Баркера, n — количество разрядов М-последовательности.

Упражнения:

1. Собрать 4-х разрядный регистр сдвига. Для тактирования использовать RS-триггер, подключенный к кнопкам 1 и 2. Кнопка 3 подключается ко входу переноса регистра, а кнопка 4 должна осуществлять сброс всех разрядов в нулевое состояние. Значения разрядов регистра должны отображаться на светодиодах 1–4, используя обращение к линейке светодиодов как к вектору.
2. Собрать формирователь кода, заданного преподавателем, используя кольцевой регистр сдвига с инверсией и логическую функцию. Для тактирования использовать RS-триггер, подключенный к кнопкам 1 и 2. Кнопка 4 должна осуществлять обнуление всех разрядов триггера. Результат должен отображаться на светодиоде 1.

- 3а. Собрать формирователь кода Баркера для заданного количества элементов N на основе кольцевого сдвигового регистра с установленным значением. Тактирование схемы осуществить от генератора, расположенного на макетной плате, установив на нем частоту 25 МГц. Выход подключить к светодиоду 1. На светодиод 2 вывести сигнал, соответствующий первому символу кода. Результат продемонстрировать в симуляторе ISim и на осциллографе, входы которого подключены к светодиодам.
- 3б. Собрать формирователь М-последовательности разрядности n . Тактирование схемы осуществить от генератора, расположенного на макетной плате (25 МГц). Нажатие и удержание кнопки 1 должно приостанавливать счет. Выход 4-х младших разрядов подключить к светодиодам. Результат формирования последовательности продемонстрировать в симуляторе ISim и на осциллографе. Найти период повторения последовательности.

9.6. Контрольные вопросы

1. Что такое регистр? Какие виды регистров бывают?
2. Что такое сдвигающий регистр? Что такое кольцевой сдвигающий регистр?
3. Как можно собрать схему сдвигающего и кольцевого сдвигающего регистра на D-триггерах?
4. Как можно описать сдвиговый и кольцевой регистр на VHDL?
5. Что такое код Баркера? Каковы основные свойства радиосигнала, промодулированного кодом Баркера?
6. Что такое М-последовательность? Каковы ее свойства? Как можно синтезировать М-последовательность с периодом повторения 255?

9.7. Контрольные задания

1. Опишите на языке VHDL модулятор кодом Баркера на 11 передаваемой информации. В качестве источника информации используйте кнопку 1, выход — светодиод 1. При нажатии кнопки должен передаваться инверсный код Баркера, а при отпущенной — прямой. (Состояние кнопки оценивать перед началом передачи очередной кодовой последовательности, а не при передаче каждого символа кода.)
2. Опишите на языке VHDL устройство, представляющее собой 4-разрядный сдвиговый регистр, тактируемый частотой 1 Гц (при частоте тактового генератора 100 мГц). В качестве источника информации — состояние кнопки 1. Выходы — на светодиоды 1–4.
3. Опишите на языке VHDL 4-разрядный счетчик, считающий с частотой 1 Гц в прямом направлении если управляющая кнопка отпущена и в обратном — если нажата.
4. Описать на VHDL формирователь 4-разрядных случайных чисел с периодом повторения не менее 1000. Каждое следующее число должно быть случайным полностью, а не полученное путем сдвига на 1 бит с последующим добавлением 1 случайного бита.

Задача 10.

Сумматоры и счетчики

Задача посвящена изучению простейших арифметических схем, счетчиков, делителей частоты и их применению.

10.1. Основные понятия

Сумматор — устройство, преобразующее информационные сигналы (аналоговые или цифровые) в сигнал, эквивалентный сумме этих сигналов. В цифровой технике применяют *двоичные сумматоры*.

Полусумматор может складывать два однобитовых двоичных числа, имеет два входа и два выхода. На одном выходе реализуется арифметическая сумма по модулю в данном разряде, а на другом — перенос в следующий (старший разряд). Действуют следующие правила:

$$\begin{aligned} 0 + 0 &= 0, & 0 + 1 &= 1, \\ 1 + 0 &= 1, & 1 + 1 &= 10. \end{aligned}$$

Полным сумматором называется схема, которая может складывать три однобитовых двоичных числа, при этом третий вход используется для подключения к выходу переноса сумматора младшего разряда:

$$\begin{aligned} 0 + 0 + 0 &= 0, & 0 + 0 + 1 &= 1, \\ 1 + 0 + 0 &= 1, & 1 + 0 + 1 &= 10, \\ 0 + 1 + 0 &= 1, & 0 + 1 + 1 &= 10, \\ 1 + 1 + 0 &= 10, & 1 + 1 + 1 &= 11. \end{aligned}$$

Параллельный сумматор — схема, которая может складывать два многоразрядных двоичных числа за один шаг. Для сложения числа

в младшем разряде используется полусумматор, в остальных — полные сумматоры. Для сложения многоразрядного и одnorазрядного двоичного числа (инкремента числа) во всех разрядах достаточно использовать полусумматоры.

Последовательный сумматор — схема, которая складывает два многоразрядных двоичных числа за несколько шагов, начиная с младшего разряда. Складываемые числа должны передаваться на сумматор последовательно, начиная с младшего разряда, при этом на выходе сумматора формируется сумма чисел. Выход переноса защелкивают в триггере на 1 такт и подают обратно на вход переноса.

Полувывчитатель может вычитать одно двоичное число из другого, имеет два входа и два выхода. Второй выход называется выходом займа. Действуют следующие правила:

$$\begin{aligned}0 - 0 &= 0, & 0 - 1 &= -1, \\1 - 0 &= 1, & 1 - 1 &= 0.\end{aligned}$$

Полным вычитателем называется схема, которая может к значению вычитаемого прибавить сигнал займа (подаваемый с младшего разряда) и такое увеличенное вычитаемое вычесть из уменьшаемого.

Многоразрядные вычитатели строятся аналогично сумматорам. Также вычитатель может быть заменен сумматором, если перевести вычитаемое в *дополнительный код* (изменить знак числа). Перевод в дополнительный код осуществляется инверсией всех бит числа и прибавлением 1.

Счетчик — это устройство для регистрации количества импульсов, подаваемых на его вход. *Коэффициент пересчета N* определяет возможное число состояний схемы (счетчик по модулю N).

Двоичный или *бинарный счетчик* оперирует с бинарными сигналами, то есть обрабатывает сигналы, состоящие из нулей и единиц. Почти все электронные счетчики являются двоичными, построенными на двоичных триггерах. Поэтому двоичный счетчик можно называть просто "счетчик".

Триггеры, входящие в состав счетчика называют *разрядами*.

Счетчики классифицируются по *направлению счета* — бывают

суммирующие, вычитающие и реверсивные счетчики. По модулю счета — двоичные, двоично-десятичные и др.

Асинхронные или *последовательные счетчики* строятся на триггерах (обычно Т-триггерах), включенных последовательно — выход предыдущего триггера является тактовым сигналом для следующего. При приходе очередного тактового импульса триггеры переключаются не одновременно, что связано с задержкой распространения сигнала по цепочке триггеров. Переключение начинается с младшего разряда счетчика и заканчивается на нужном разряде, в соответствии с таблицей счета. Максимальное время переключения всего счетчика пропорционально зависит от количества его разрядов.

Синхронные или *параллельные счетчики* строятся на триггерах, тактовые входы которых объединены, а логика работы обеспечивается внутренними связями. При приходе тактового импульса все триггеры (переключение которых требуется на данном шаге) изменяют свое состояние одновременно. Задержка переключения всего счетчика не зависит от числа его разрядов.

Синхронный счетчик может быть построен на основе параллельного регистра с динамическим тактовым входом и параллельного сумматора, на второй вход которого подано число 1.

В счетчика с *групповой структурой* или *параллельно-последовательных счетчиках* триггеры разбиты на группы. Каждая группа представляет собой параллельный счетчик, а между группами используются последовательные связи (входы и выходы переноса). Максимальное время переключения такого счетчика пропорционально количеству параллельных групп в его составе.

Делители частоты — цифровые последовательные устройства, выполненные по схеме счетчика, но имеющие один счетный вход и один выход. Коэффициент деления равен коэффициенту пересчета счетчика.

10.2. VHDL описание счетчиков

Современные ПЛИС позволяют создавать параллельные счетчики с большим числом разрядов, время переключения которых соответствует максимально возможному быстродействию микросхемы. При этом используются линии глобального тактирования кристалла.

Для описания работы счетчиков необходимо использовать процесс, зависящий от сигнала тактирования. Внутри процесса должно быть описано условие тактирования, а также дополнительные возможности, например сброс, разрешение счета:

```
-- реверсивный счетчик с синхронным сбросом
process (C)
begin
    if(C'event and C = '1') then
        if(R = '1') then
            n <= 0;
        elsif(E = '1') then
            n <= n + 1;
        end if;
    end if;
end process;
```

Сигнал, используемый в качестве счетчика должен быть объявлен как вектор необходимой размерности, при этом получается счетчик с коэффициентом пересчета $N = 2^n$, где n число разрядов. Например, для счетчика на 16:

```
SIGNAL n : STD_LOGIC_VECTOR(3 downto 0) := "0000";
```

Для реализации счетчика на $N < 2^n$ в описание процесса вводится проверка условия на обнуление счетчика при достижении необходимого значения. Второй вариант — использовать вычитающий счетчик с предустановкой значения при инициализации или достижении нуля.

При построении делителя частоты на N можно использовать сигнал старшего разряда счетчика, однако при этом скажность сигнала будет отличаться от 2 если $N < 2^n$. В этом случае (при необходимости формирования меандра) рекомендуется использовать делитель частоты с вдвое меньшим коэффициентом деления, подключенный ко входу счетного триггера.

Использование кнопки в качестве источника тактового сигнала счетчика осложняется проблемой, связанной с эффектом *дребезга контактов* — в момент нажатия происходит переходный процесс, формирующий некую последовательность случайных импульсов. Чтобы исключить этот эффект, необходимо использовать программный ждущий мультивибратор, в котором время задержки задается за счет деления частоты тактового генератора (сигнал Clk с частотой 25–100 мГц):

```
-- после описания IEEE библиотеки
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- после architecture Behavioral of ....., но перед begin
SIGNAL n : STD_LOGIC_VECTOR(3 downto 0) := "0000";
SIGNAL p : INTEGER := 0;

-- после architecture .... и begin, но перед end Behavioral;
process (Clk)
begin
    if(Clk'event and Clk = '1') then
        if(bot1 = '0') then
            if(p = 0) then
                n <= n + 1;
            end if;
            p <= 100000;
        elsif(p > 0) then
            p <= p - 1;
        end if;
        led <= n;
    end if;
end process;
```

Этот пример также содержит счетчик: переменная p типа integer. По умолчанию integer — 32-битное число, которое можно записывать не в двоичной, а в привычной десятичной системе. Часто удобно работать с небольшими числами, в этом случае для ограничения расходуемых ресурсов плюс (числа разрядов) можно указать используемый диапазон. Например:

```
SIGNAL p : INTEGER RANGE 0 TO 100000 := 0;
```

Особенностью выполнения упражнений задачи является обращение к линейке светодиодов как к 4-разрядному вектору. При этом, в

ucf файле необходимо ставить соответствие элемент вектора LED(3 downto 0) и ножку порта, например:

```
NET "LED(0)" LOC = "P5" ;
```

10.3. Практическая часть

Задача выполняется на макетной плате с установленной ПЛИС XC9572XL (стр. 66). Для каждого упражнения необходимо создать собственный проект (на языке VHDL). Дополнительно, работа каждого устройства проверяется в симуляторе ISim (схема устранениядребезга контактов при этом исключается). Порядок работы с симулятором описан в приложении на стр. 65.

Параметры: сборка схемы сумматора или вычитателя, f и s — частота и скважность сигнала, формируемая делителем частоты, τ — длительность импульса, формируемого ждущим мультивибратором, N — число пересчета реверсивного счетчика.

Упражнения:

1. Собрать двухразрядный двоичный сумматор (вариант а) или вычитатель (вариант б), используя элементарные логические операции (НЕ, И, ИЛИ, ИСКЛЮЧАЮЩЕЕ_ИЛИ). Первое число должно формироваться кнопками 1 и 2, второе — 3 и 4. Результат вывести на светодиоды 1–3. Записать таблицу истинности схемы.
- 2а. Собрать делитель частоты кварцевого генератора, входящего в комплект макетной платы, формирующий импульсы с заданной скважностью и частотой. Выход подключить к светодиоду 1.
- 2б. Собрать ждущий мультивибратор, запускаемый нажатием на кнопку 1 и формирующий импульс, длительностью τ . Выход подключить к светодиоду 1.
3. Собрать четырехразрядный счетчик с естественным порядком счета. Счетчик должен увеличивать свое значение на 1 при нажатии на кнопку 1. Для подавлениядребезга контактов использовать ждущий мультивибратор. Выходы подключить к

светодиодам 1–4. Записать таблицу всех возможных состояний счетчика, начиная с "0000".

- 4а. Собрать четырехразрядный реверсивный счетчик с коэффициентом пересчета N , увеличивающий значение на 1 при нажатии на кнопку 1 и уменьшающий на 1 при нажатии на кнопку 2. Для подавления дребезга контактов использовать ждущий мультивибратор. Сброс счетчика в положение "0000" должен осуществляться по нажатию на кнопку 3. Результат счета должен отображаться в двоичном коде на светодиодах 1–4.
- 4б. Собрать схему коррелятора сигнала с кодом Баркера. Схема должна считать количество совпадающих бит в 13-ти разрядном сдвиговом регистре с кодом Баркера. Результат в виде двоичного кода вывести на светодиодную линейку. Нажатие кнопок 1-4 должно приводить к сдвигу информации в регистре, с заполнением младшего разряда логическим 0, 1, значением старшего разряда (циклический сдвиг) и псевдослучайным значением соответственно. Построить в тетради функцию автокорреляции кода Баркера и функцию корреляции кода Баркера с псевдослучайной последовательностью.

10.4. Контрольные вопросы

1. Как работает полусумматор и полный сумматор, полувычитатель и полный вычитатель?
2. Как можно собрать полусумматор и полный сумматор на логических элементах?
3. Чем отличаются последовательные и параллельные многоразрядные сумматоры/вычитатели?
4. Чем отличаются синхронные и асинхронные счетчики?
5. Что понимается под счетчиком по модулю N ? Как получить делитель частоты?

-
6. Что такое реверсивный счетчик? Что такое двоично-десятичный счетчик?

Задача 11.

Системы счисления и целочисленная арифметика

11.1. Основные понятия

Двоичный или *бинарный код* — способ представления чисел, использующий в одном разряде комбинацию двух знаков (0 и 1). В этом случае разряд числа называется *бит* или *бинарный разряд*.

Позиционная система счисления — система счисления, в которой позиция цифры однозначно связана со значением числа. Зависимость является степенной.

Двоичная система счисления — позиционная система счисления с основанием 2.

Двоично-десятичный код — представляет каждый разряд десятичного числа четырьмя бинарными разрядами или 4-мя битами. Совокупность 4-х бит называется *тетрадой*. Из 16 возможных тетрад используются только 10. Таким образом, n разрядное десятичное число представляется в двоично-десятичном коде n тетрадами.

Восьмеричная система счисления — позиционная целочисленная система счисления с основанием 8. Для представления чисел в ней используются цифры от 0 до 7.

Шестнадцатеричная система счисления — позиционная система счисления по целочисленному основанию 16. В качестве цифр этой системы счисления обычно используются цифры от 0 до 9 и латинские буквы от A до F. Буквы A, B, C, D, E, F имеют значения 10, 11, 12, 13, 14, 15 соответственно.

Код Грея — система счисления, в которой два соседних значения

различаются только в одном разряде. Наиболее часто на практике применяется рефлексивный двоичный код Грея, хотя в общем случае существует бесконечное множество кодов Грея для систем счисления с любым основанием. В большинстве случаев, под термином «код Грея» понимают именно рефлексивный бинарный код Грея.

В таблице 11.1 представлены числа от 0 до 15 в различных кодах.

Таблица 11.1. Сравнение различных кодов для чисел от 0 до 15

Десятичный код (dec)	Двоичный код (bin)	Двоично-десятичный код	Восьмеричный код (oct)	Шестнадцатеричный код (hex)	Бинарный код Грея
0	0000	0000 0000	000	0	0000
1	0001	0000 0001	001	1	0001
2	0010	0000 0010	002	2	0011
3	0011	0000 0011	003	3	0010
4	0100	0000 0100	004	4	0110
5	0101	0000 0101	005	5	0111
6	0110	0000 0110	006	6	0101
7	0111	0000 0111	007	7	0100
8	1000	0000 1000	010	8	1100
9	1001	0000 1001	011	9	1101
10	1010	0001 0000	012	A	1111
11	1011	0001 0001	013	B	1110
12	1100	0001 0010	014	C	1010
13	1101	0001 0011	015	D	1011
14	1110	0001 0100	016	E	1001
15	1111	0001 0101	017	F	1000

Двоичный код является *весовым кодом*, каждому разряду которого сопоставлена степень числа 2. Двоично-десятичный код также является весовым. Весовые коды могут быть преобразованы в аналоговые сигналы и наоборот.

Код Грея является примером *невесового кода*. Невесовые коды перед преобразованием в аналоговые цифро-аналоговые сигналы должны преобразовываться в весовой код.

Дополнительный код — наиболее распространенный способ представления отрицательных целых чисел в компьютерах. Он позволяет заменить операцию вычитания на операцию сложения и сделать

операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел, чем упрощает архитектуру ЭВМ.

При записи числа в дополнительном коде старший разряд является знаковым. Если его значение равно 0, то в остальных разрядах записано положительное двоичное число, совпадающее с прямым кодом, если 1 — отрицательное двоичное число.

В таблице 11.2 представлены числа, кодируемые 8-ю битами в дополнительном коде (тип `char`, `byte`, `int8` в языках программирования). В скобках приведены соответствующие положительные числа в беззнаковом представлении (`unsigned char`, `unsigned byte`, `uint8`). Таким образом, 8 бит кодируют числа от 0 до 255 (положительные) или от -128 до 127 (со знаком), 16 разрядов — от 0 до 65535 или от -32768 до 32767, 32 разряда — от 0 до 4294967295 или от -2147483648 до 2147483647.

Таблица 11.2. Кодирование 8-битного числа в дополнительном коде

Положительные числа		Отрицательные числа	
Десятичное	Двоичное	Десятичное	Двоичное
0	00000000	-1 (255)	11111111
1	00000001	-2 (254)	11111110
2	00000010	-3 (253)	11111101
3	00000011	-4 (252)	11111100
4	00000100	-5 (251)	11111011
...			
124	01111100	-125 (131)	10000011
125	01111101	-126 (130)	10000010
126	01111110	-127 (129)	10000001
127	01111111	-128 (128)	10000000

Для инверсии числа в дополнительном коде необходимо проинвертировать все разряды числа и к результату прибавить 1.

11.2. Умножение двоичных чисел

Простейшим однобитовым *умножителем* является элемент И:

$$\begin{aligned}0 + 0 &= 0, & 0 + 1 &= 0, \\1 + 0 &= 0, & 1 + 1 &= 1.\end{aligned}$$

Матричный или *параллельный умножитель* — цифровая схема, осуществляющая умножение двух (многоразрядных) чисел с помощью двоичного каскадного сумматора. Умножение двух чисел a и b производится следующим образом:

1. Вычисление частных произведений при помощи логической операции И, а также их сдвиг в соответствии с весом произведения. Вычисление производится по формуле $m_i = 2^i ab_i$, где b_i — i -ый бит числа b .
2. Суммирование частных произведений по формуле $m = \sum_i m_i$.

При сложении многоразрядных чисел сложность схемы резко возрастает с ростом числа разрядов. Например, для умножения двух 8-ми разрядных чисел потребуется 64 2И-элемента и 8 8-битовых параллельных сумматора. Такая схема будет не только сложна, но и достаточно медленна.

Современные перемножители выполняют суммирование частных произведений в виде *дерева Уоллеса* — путем преобразования каждого трех слагаемых в два. Для того, чтобы представить сумму трех чисел (x, y, z) с помощью двух чисел, направим каждый разряд x_i , y_i и z_i на вход полного сумматора. Тогда младший бит сумматора будет i -ым битом первого числа, а старший — $(i + 1)$ -ым второго. Поскольку все полные сумматоры работают параллельно (выходы на каждом из них зависят только от собственных входов), то глубина такой схемы есть константа (не зависит от количества бит). Получившиеся в конце дерева два числа складываются обычным многоразрядным сумматором.

Последовательный умножитель строится на базе сдвиговых регистров и позволяет экономить количество логических элементов (сум-

маторов) за счет последовательного выполнения операций сложения с разными разрядами чисел.

11.3. Работа с семисегментным индикатором

Семисегментный индикатор представляет собой набор светодиодов, расположенных определенным образом (рис. 11.1).

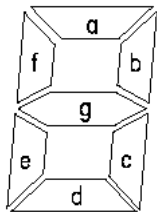


Рис. 11.1. Расположение сегментов семисегментного индикатора



Рис. 11.2. Формирование разряда шестнадцатеричного числа на семисегментном индикаторе

При помощи отдельных сегментов возможно формирование чисел восьмеричной, десятичной или шестнадцатеричной системы счисления (рис. 11.2).

Многоразрядные светодиодные индикаторы подключают к контроллерам в виде матриц. Для 4-разрядного индикатора получается матрица 4x7 (см. рис. на странице 74). Выбор нужного разряда осуществляется подачей логического 0 на линии S1–S4, а сегмента — подачей 0 на линии DA–DG. Для формирования 4-х разрядного числа на индикаторе необходимо поочередно устанавливать логический 0 на одной из линий S1–S4 (рис. 11.3), синхронно подавая информацию на линии DA–DG (таблица 11.3). Пауза между импульсами S4 и S1 позволяет снижать яркость индикатора.

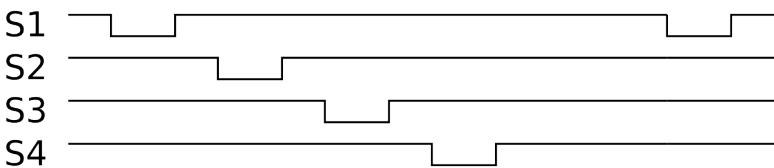


Рис. 11.3. Временная диаграмма сигналов выбора разряда индикатора

Таблица 11.3. Кодирование разрядов семисегментного индикатора (управляющие сигналы подключены к катодам светодиодов)

отображаемый символ	сегмент						
	A	B	C	D	E	F	G
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0
-	1	1	1	1	1	1	0
	1	1	1	1	1	1	1

11.4. Практическая часть

Упражнения выполняются на макетной плате с ПЛИС XC6SLX (стр. 76). Для каждого упражнения необходимо создать собственный проект (на языке VHDL).

Упражнения:

1. Собрать схему отображения 8-битного двоичного числа на семисегментном индикаторе. Двоичное число задается переключателями 1–8. Кнопки 1–4 задают систему счисления отображаемого числа: 1 — восьмиричная (000 – 377), 2 — десятичная (0 – 255), 3 — десятичная со знаком (-128 – 127), 4 — шестнадцатиричная (00 – FF). Семисегментный индикатор отображает значение двоичного числа в выбранной системе.
2. Собрать двухразрядный двоичный умножитель, используя опе-

рации И, сдвига и сумматор. Первое число должно формироваться кнопками 1 и 2, второе — 3 и 4. Результат вывести на светодиоды 1–4. Записать таблицу истинности схемы.

3. Собрать умножитель двух 8-ми разрядного беззнакового двоичного числа, задаваемое переключателями 1–8 на 4-х разрядное беззнаковое двоичное число, задаваемое кнопками 1–4. Результат вывести на семисегментный индикатор в десятичной системе счисления. Незначащие нули (слева) показываться не должны. Использовать встроенный в ПЛИС модуль умножения.

11.5. Контрольные вопросы

1. Что такое весовой, невесовой и дополнительный коды?
2. Что такое двоичный и двоично-десятичный код?
3. Что такое код Грея и где он используется?
4. Чем отличаются последовательные и параллельные умножители?
5. Что такое дерево Уоллеса и для чего оно используется?
6. Какие наиболее распространенные системы счисления вы знаете?

Задача 12.

Цифро-аналоговые и аналого-цифровые преобразователи

12.1. Основные понятия

Цифро-аналоговый преобразователь — устройство для преобразования цифрового кода в аналоговый сигнал. Полученный аналоговый сигнал является ступенчатым сигналом с определенным количеством возможных амплитудных значений.

Аналого-цифровой преобразователь — устройство, преобразующее входной аналоговый сигнал в дискретный цифровой код. АЦП чаще всего выдают результат в двоичной системе счисления или в двоично-десятичном коде.

Обычно ЦАП и АЦП являются *линейными* — когда аналоговое напряжение пропорционально значению цифрового кода.

Разрядность — количество бит, определяющих количество дискретных значений, воспроизводимых АЦП или число различных уровней выходного сигнала ЦАП. Однобитный ЦАП способен воспроизвести два (2^1) уровня, примером однобитного АЦП является компаратор. Восьмибитный ЦАП воспроизводит 256 (2^8) уровней, аналогичный ему АЦП выдает 256 различных чисел.

Эффективная разрядность — комплексная характеристика, которая отражает влияние сразу нескольких источников ошибок: шума квантования, дифференциальной нелинейности, шумов внутренних узлов тракта преобразования. Причем если известна зависимость эффективной разрядности от частоты, то это понятие учитывает и динамические погрешности. Эффективная разрядность вычисляется

по формуле:

$$ENOB = \frac{SNR(db) - 1,76}{6,02}, \quad (12.1)$$

где ENOB — Effective Number Of Bits, а SNR — отношение сигнал/шум, выраженное в децибелах.

Частота дискретизации — частота, с которой следуют цифровые значения (отчеты).

Передискретизация — изменение частоты дискретизации оцифрованного сигнала. Исходный сигнал может быть оцифрован на более высокой частоте, например, для последующей цифровой фильтрации, а затем произведена *децимация* — снижение частоты. Увеличение частоты с восстановлением промежуточных значений сигнала называется *интерполяцией*.

12.2. Цифро-аналоговые преобразователи

Максимальная частота дискретизации — максимальная частота, на которой ЦАП может работать, выдавая на выходе корректный результат. В соответствии с теоремой Найквиста — Шеннона (известной также как теорема Котельникова), для корректного воспроизведения аналогового сигнала из цифровой формы необходимо, чтобы частота дискретизации была не менее, чем удвоенная максимальная частота в спектре сигнала.

Динамический диапазон — соотношение наибольшего и наименьшего сигналов, которые может воспроизвести ЦАП, выражается в децибелах. Данный параметр связан с разрядностью и шумовым порогом.

Вес младшего разряда или *Least Significant Bit (LSB)* — величина элементарного шага или квант преобразования.

Смещение нуля — напряжение на выходе ЦАПа, когда на вход подается код, соответствующий нулевому напряжению. Выражается в процентах от полной шкалы или LSB.

Усиление — наклон характеристики преобразования ЦАП.

Дифференциальная нелинейность — это локальная характеристика, определяет насколько приращение аналогового сигнала, полученное при увеличении кода на 1 младший значащий разряд (МЗР), отличается от правильного (среднего по шкале) значения. Выражается в LSB. В идеальном случае при изменении цифрового кода на 1 аналоговый сигнал также должен измениться на 1 LSB.

Интегральная нелинейность — максимальное отклонение от прямой линии, соединяющей крайние точки шкалы. Параметр характеризует, насколько передаточная характеристика ЦАП отличается от идеальной, строго линейной характеристики. Интегральную нелинейность выражают либо в количестве квантов (МЗР), либо в процентах от полной шкалы.

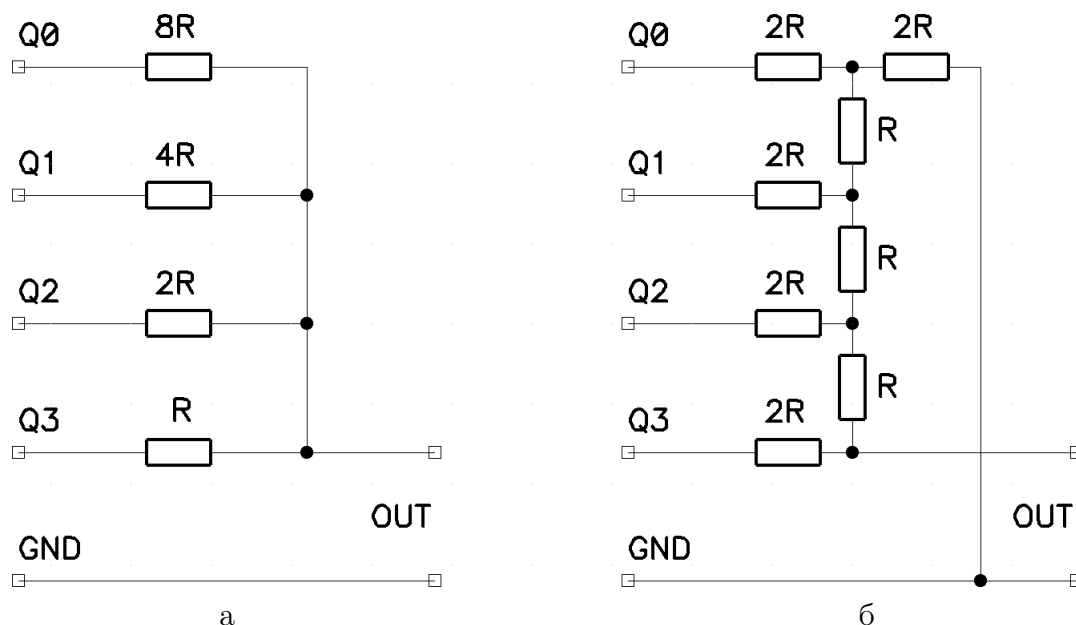


Рис. 12.1. Схемы цифро-аналоговых преобразователей на резисторах (а — взвешивающего типа, б — лестничного типа)

Типы цифро-аналоговых преобразователей:

1. *ЦАП взвешивающего типа* (рис. 12.1,а). Каждому биту преобразуемого двоичного кода соответствует резистор, подключенный на общую точку суммирования. Проводимость резистора выбирается пропорциональна весу бита в коде:

$$R_n = R_0/2^n, \quad (12.2)$$

где n — номер разряда двоичного весового кода. Таким образом, все ненулевые биты кода суммируются с весом. Взвешивающий

метод один из самых быстрых, но ему свойственна низкая точность из-за необходимости прецизионного подбора сопротивлений схемы. Обычно взвешивающие ЦАП имеют разрядность не более восьми бит.

2. *ЦАП лестничного типа* или *цепная R - $2R$ -схема* (рис. 12.1,б). В этом типе ЦАП напряжение формируется в специальной схеме, состоящей из резисторов с сопротивлениями R и $2R$, называемой *матрицей постоянного импеданса*. Она имеет два вида включения: прямое — матрица токов и инверсное — матрица напряжений. Применение одинаковых резисторов позволяет существенно улучшить точность по сравнению с обычным взвешивающим ЦАП, так как сравнительно просто изготовить набор прецизионных элементов с одинаковыми параметрами. С лазерной подгонкой резисторов на одной подложке достигается точность 20-22 бита.
3. *Широтно-импульсный модулятор (ШИМ)*. Преобразование осуществляется за счет формирования импульсного сигнала переменной скважности, причем длительность импульса пропорциональна цифровому коду. Полученная импульсная последовательность фильтруется аналоговым фильтром нижних частот.
4. *ЦАП передискретизации* — позволяют использовать ЦАП с меньшей разрядностью для достижения большей разрядности итогового преобразования. Например, сигма-дельта ЦАП состоит из цифрового интерполирующего фильтра, сигма-дельта модулятора, низкоразрядного ЦАП и аналогового фильтра нижних частот (рис. 12.2). При этом, спектр шума квантования изменяется — основную часть энергии шума переносится в высокочастотную область, где он может быть легко подавлен аналоговым фильтром относительно низкого порядка. *Сигма-дельта модулятор*, входящий в состав ЦАП представляет собой чисто цифровое устройство, которое преобразует последовательность многоразрядных слов на входе в малоразрядные двоичные коды

(рис. 12.3). Регистр в составе модулятора задерживает сигнал на один такт, который затем вычитается из входного. Полученная разность интегрируется, урезается по количеству бит и снова поступает на регистр.

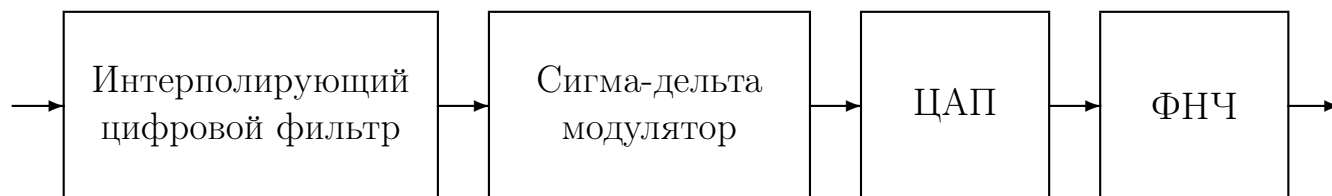


Рис. 12.2. Блок-схема сигма-дельта ЦАП



Рис. 12.3. Блок-схема сигма-дельта модулятора цифрового сигнала

12.3. Аналого-цифровые преобразователи

Разрешающая способность или *разрешение* — минимальное изменение величины аналогового сигнала, которое может быть преобразовано данным АЦП. В случае единичного измерения без учета шумов разрешение напрямую определяется разрядностью АЦП и равно разности напряжений, соответствующих максимальному и минимальному выходному коду, деленной на количество выходных дискретных значений. На практике разрешение АЦП ограничено отношением сигнал/шум входного сигнала и описывается эффективной разрядностью.

Точность АЦП — определяется ошибками квантования, нелинейностью и апертурной погрешностью (джиттером).

Производительность — частота дискретизации, с которой производятся цифровые значения.

Наложение спектров (алиасинг) — проявляется когда в оцифровываемом сигнале присутствуют составляющие, частоты которых

выше половины частоты дискретизации. Они ошибочно воспринимаются как сигналы более низких частот. Для устранения этого явления необходимо использовать фильтр нижних частот на входе АЦП.

Типы аналого-цифровых преобразователей:

1. *АЦП последовательного счета* — оцифровывает аналоговый сигнал, сравнивая его с линейно нарастающим эталонным напряжением пилообразной формы. В момент перехода через ноль эталонного напряжения запускается счетчик импульсов, а когда пилообразный сигнал достигает уровня входного сигнала — счетчик останавливается. Полученное значение считывается и подается на выход АЦП.
2. *АЦП с двойным интегрированием* работает в 2 этапа. На первом шаге происходит заряд конденсатора током, пропорциональным входному напряжению в течении определенного интервала времени. На втором — происходит разряд конденсатора до нуля постоянным током. Время разряда конденсатора соответствует измеренному значению.
3. *АЦП компенсационного типа* — сравнивает входной сигнал с компенсационным напряжением, полученным в результате обратного преобразования при помощи ЦАП. На ЦАП подаются различные значения, пока оба напряжения не сравняются. Схема может работать в нескольких режимах:
 - счет от 0 до момента совпадения напряжений;
 - с изменением направления непрерывного счета, определяемым на основании сравнения напряжений;
 - *методом последовательного приближения*, когда компенсационное напряжение формируется делением измеряемого интервала пополам, определяя значения бит последовательно, начиная со старшего.
4. *АЦП по принципу напряжение — частота*. Аналоговый сигнал преобразуется в переменное напряжение определенной частоты,

линейно зависящей от амплитуды аналогового сигнала. Затем частота измеряется цифровым способом.

5. *АЦП прямого преобразования* работают по принципу сравнения с набором фиксированных напряжений. Полученный линейный код преобразуется в двоичный при помощи шифратора. Это самый быстрый вид АЦП, но сложность его реализации сильно возрастает с увеличением разрядности. Существует несколько разновидностей таких АЦП:

- параллельные АЦП прямого преобразования, содержат по одному компаратору на каждый дискретный уровень входного сигнала;
- параллельно-последовательные АЦП прямого преобразования, позволяют значительно уменьшить количество компараторов за счет того, что содержат в своем составе k параллельных АЦП прямого преобразования — второй, третий и т. д. АЦП служат для уменьшения ошибки квантования первого АЦП путем оцифровки этой ошибки;
- полностью последовательные АЦП прямого преобразования, каждый компаратор которого определяет один бит кода, причем опорные напряжения для каждого компаратора формируются на основании более старших разрядов кода.

6. *Сигма-дельта АЦП* производит аналого-цифровое преобразование с частотой дискретизации, во много раз превышающей требуемую и путем фильтрации оставляет в сигнале только нужную спектральную полосу.

12.4. Практическая часть

Упражнения выполняются на макетной плате с ПЛИС XC6SLX (стр. 76). Для каждого упражнения необходимо создать собственный проект (на языке VHDL).

Упражнения:

1. Собрать 4-х разрядный ЦАП, работающий по принципу широтно-импульсной модуляции. Цифровой код считать с кнопок 1–4, выход подключить к светодиоду 1.
2. Собрать 4-х разрядный сигма-дельта ЦАП, без модуля интерполяции, использующий передискретизацию в 1 разряд. Цифровой код считать с кнопок 1–4, выход подключить к светодиоду 1.
3. Собрать 4-х разрядный АЦП последовательного приближения на основе ЦАП с ШИМ. Результат работы вывести на светодиоды 1–4.

12.5. Контрольные вопросы

1. Какие типы ЦАП вы знаете?
2. Какие параметры характеризуют ЦАП?
3. Как устроены ЦАП на резисторах и широтно-импульсный модулятор?
4. Какой принцип работы сигма-дельта ЦАП?
5. Какие типы АЦП вы знаете?
6. Какие параметры характеризуют АЦП?
7. Как устроены АЦП последовательного счета и АЦП с двойным интегрированием?
8. Как устроены АЦП компенсационного типа и по принципу напряжение — частота?
9. Как устроены АЦП прямого преобразования?
10. Как устроены сигма-дельта АЦП?

Справочная информация

Программируемые логические интегральные схемы

Программируемая логическая интегральная схема (ПЛИС) — электронный компонент, используемый для создания цифровых интегральных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задается посредством программирования (проектирования). Для этого используются программаторы и отладочные среды, позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры: Verilog, VHDL, AHDL и др.

ПЛИС широко используется для построения различных по сложности и по возможностям цифровых устройств. Некоторые производители ПЛИС предлагают программные процессоры для своих ПЛИС, которые могут быть модифицированы под конкретную задачу, а затем встроены в ПЛИС. Тем самым обеспечивается уменьшение места на печатной плате и упрощение проектирования самой ПЛИС.

Programmable Logic Array (PLA) — маскируемые (программируемые с помощью маски) интегральные схемы (ИС), основанные на ассоциативном постоянном запоминающем устройстве (ПЗУ). Микросхема программировалась чередованием металлических слоев в процессе производства. В качестве элементов памяти использовались JK-триггеры.

Programmable Array Logic (PAL) — программируемый массив (матрица) логики (ПЛМ). Разница между PLA и PAL состоит в доступности программирования внутренней структуры (матриц) ПЛМ.

Gate Array Logic (GAL) — это ПЛИС, имеющие программируемую матрицу И и фиксированную матрицу ИЛИ.

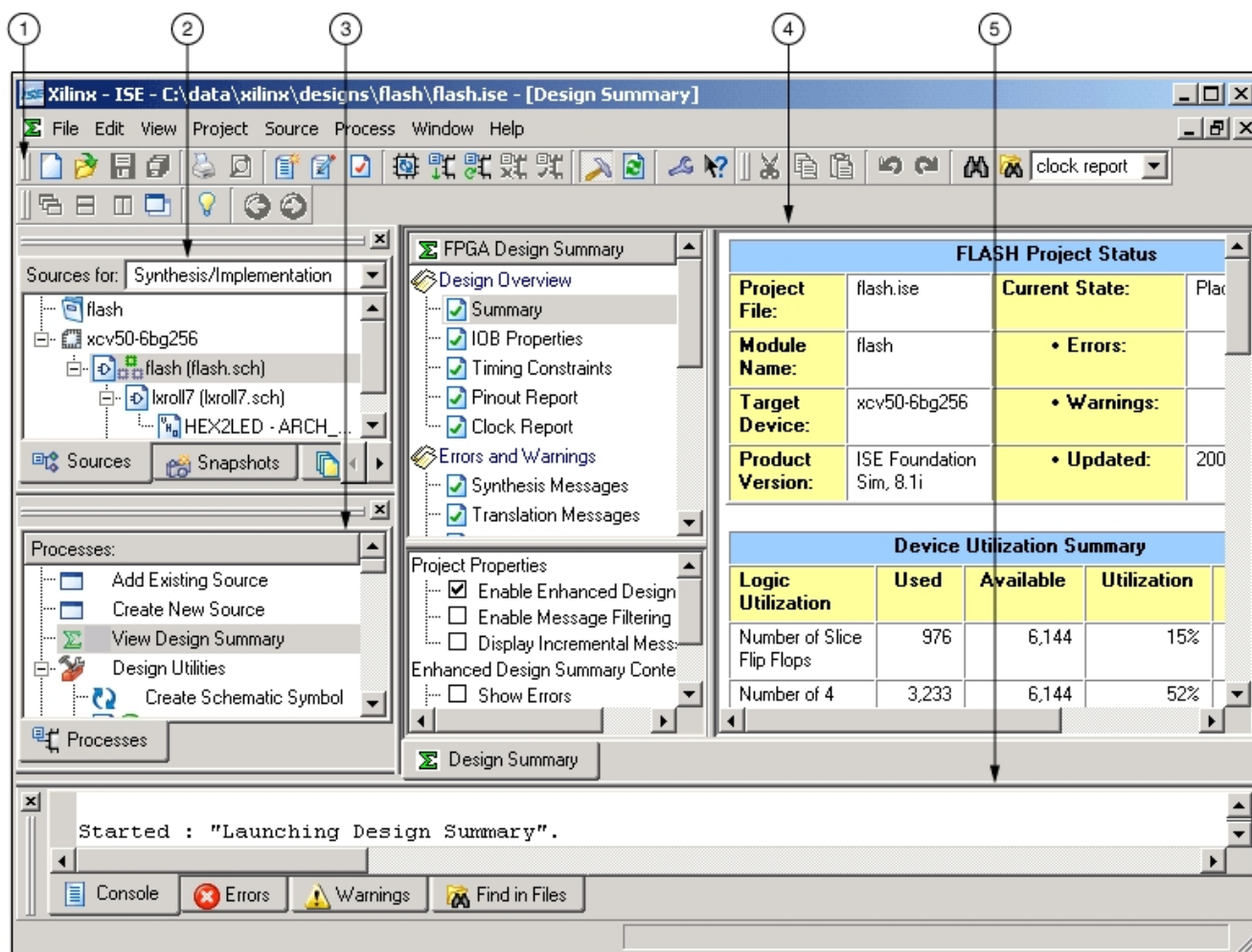
Complex Programmable Logic Device (CPLD) — сложные программируемые логические устройства, содержат относительно крупные программируемые логические блоки — макроячейки, соединенные с внешними выводами и внутренними шинами. Функциональность CPLD кодируется в энергонезависимой памяти, поэтому нет необходимости их перепрограммировать при включении.

Field-Programmable Gate Array (FPGA) — более сложная ПЛИС, содержат блоки умножения-суммирования, которые широко применяются при обработке сигналов, а также логические элементы (как правило, на базе таблиц перекодировки — таблиц истинности) и блоки коммутации. FPGA обычно используются для обработки цифровых сигналов, имеют больше логических элементов и более гибкую архитектуру, чем CPLD. Программа для FPGA хранится в распределенной памяти, которая может быть выполнена как на основе энергозависимых ячеек статического ОЗУ (подобные микросхемы производят, например, фирмы Xilinx и Altera) — в этом случае программа не сохраняется при исчезновении электропитания микросхемы, так и на основе энергонезависимых ячеек Flash-памяти или перемычек antifuse (такие микросхемы производит фирма Actel и Lattice Semiconductor) — в этих случаях программа сохраняется при исчезновении электропитания. Если программа хранится в энергозависимой памяти, то при каждом включении питания микросхемы ее необходимо заново конфигурировать при помощи начального загрузчика, который может быть встроен в саму FPGA.

Альтернативой ПЛИС FPGA являются более медленные цифровые процессоры обработки сигналов.

Система автоматического проектирования Xilinx ISE

WebPACK ISE — средство проектирования цифровых устройств.



Основное окно Навигатора проекта: 1 — инструментальная панель (Toolbar); 2 — окно описания проекта (Source window); 3 — окно процессов (Process window); 4 — рабочий стол (Workspace); 5 — окно отчетов (Transcript window).

Для подключения файла лицензий необходимо выполнить следующие действия:

1. Запустить программу ISE. Закрыть окно Tip of the Day, сняв галочку Show Tips at Startup. Закрыть окно Xilinx License Error, нажав ОК. Подождать запуска менеджера лицензий.
2. В окне Xilinx License Configuration Manager выбрать закладку Manage Licenses, нажать кнопку Load License и подключить файл

лицензий `"/opt/Xilinx/Xilinx.lic"`, закрыть все окна, относящиеся к менеджеру лицензий.

Для проверки работоспособности всей системы необходимо присоединить к программатору плату с микросхемой CPLD XC9572XL и выполнить следующие действия:

1. Запустить программу ISE.
2. Открыть тестовый проект (File — Open Project). Имя файла `"/home/student/Desktop/ISE_test/XC9572XLVQ44.xise"`.
3. Запустить программу iMPACT (Tools — iMPACT), проигнорировав предупреждение и отказавшись от опроса.
4. Двойным щелчком мыши вызвать режим сканирования оборудования Boundary Scan. Выбрать инициализацию канала (File — Initialize Chain). В окне Auto Assign Configuration Files Query Dialog поставить галочку и нажать Yes. В окне Assign New Configuration Files выбрать соответствующий файл проекта `"/home/student/Desktop/ISE_test/XC.jed"`. Закрыть окно Device Programming Properties нажав ОК.
5. Выбрать режим очистки Erase. Убедиться что 4 светодиода горят в пол накала.
6. Выбрать режим программирования Programm. 3 светодиода должны погаснуть, один гореть в четверть накала. При нажатии на кнопки состояние светодиодов должно изменяться.
7. После тестирования можно закрыть или свернуть программу iMPACT, а в окне ISE — закрыть проект (File — Close Project).

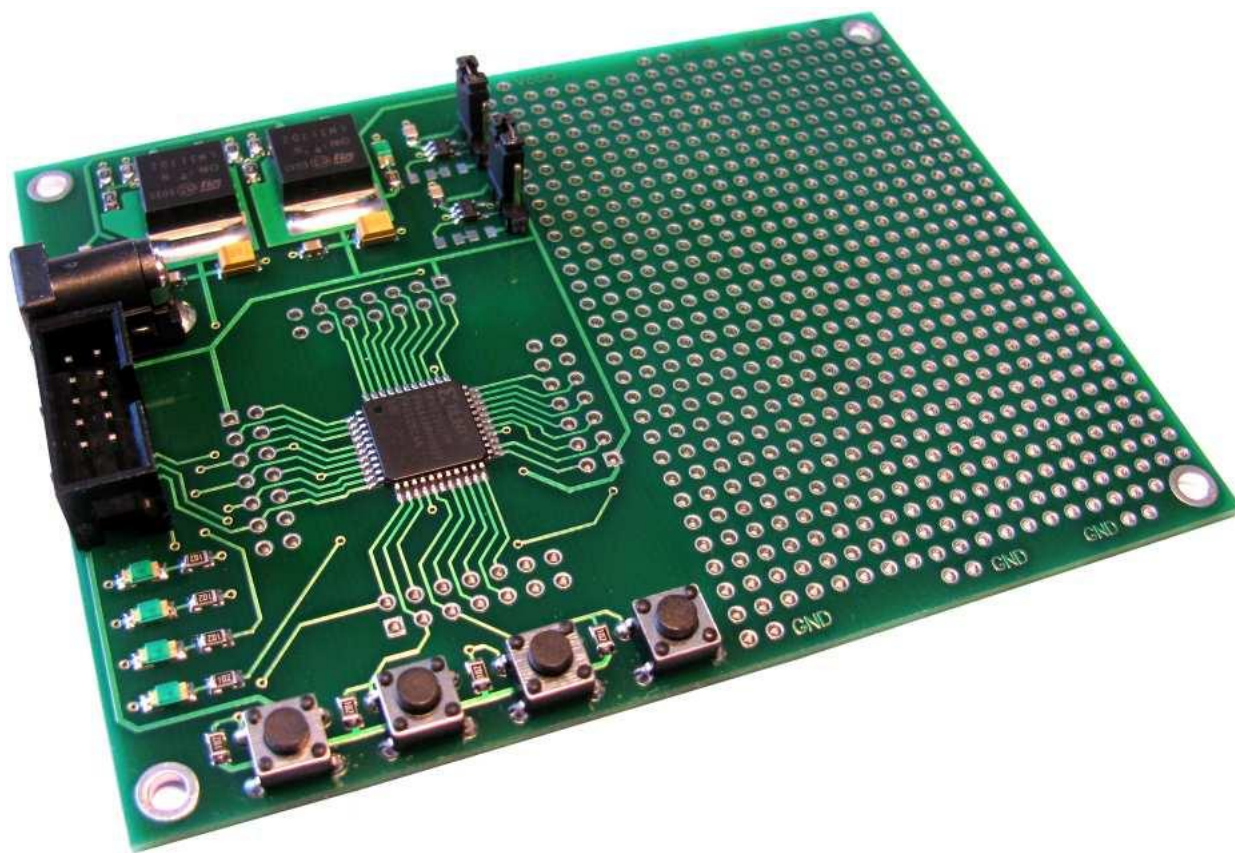
Симулятор ПЛИС ISim

Симулятор ПЛИС позволяет убедиться в правильности работы сложных проектов или отдельных алгоритмов, используемых в проекте.

Порядок работы с симулятором.

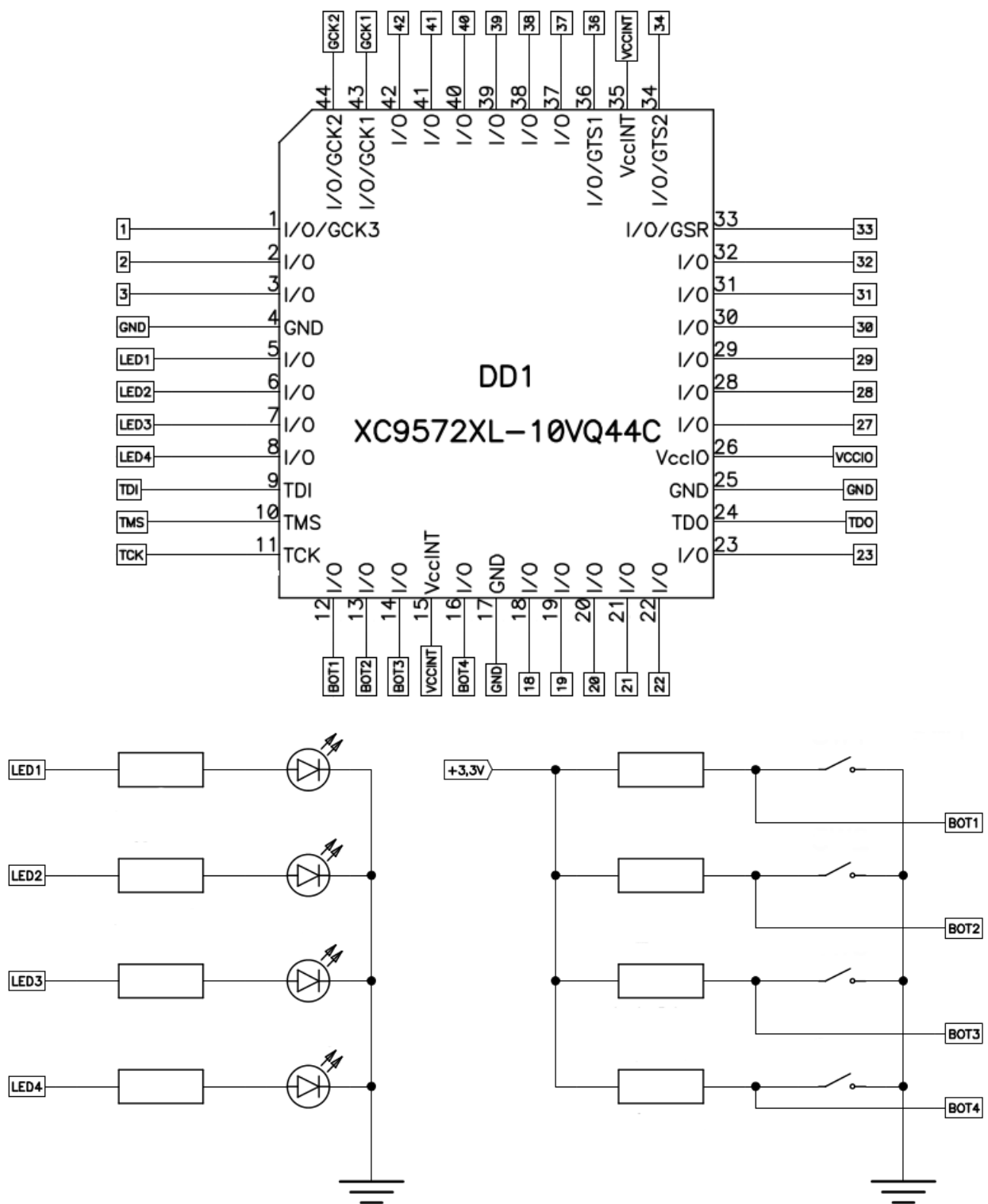
1. Запустить ISE, открыть проект или создать новый.
2. Создать новый модуль описания (Project — New Source, тип VHDL Test Bench). В результате получится шаблон работы с симулятором на языке VHDL, в котором необходимо вписать необходимый функционал.
3. В левой части окна войти в панель Design, переключить вид на симуляцию View: Simulation и активизировать файл проекта, который предполагается симулировать. При этом, в нижней части панели Design появится пункт ISim Simulator и в нем два подпункта: Behavioral Check Syntax — для проверки синтаксиса и Simulate Behavioral Model — для запуска ISim. Необходимо выполнить их последовательно.
4. В окне ISim выполнить View — Zoom — To Full View для получения обзорного вида на процесс симуляции.

Конструктор-ПЛИС CPLD (задачи 7-10)



Конструктор-ПЛИС фирмы LDM-SYSTEMS предназначен для макетирования устройств, проектируемых на ПЛИС CPLD XC9572XL фирмы Xilinx, а также сборки законченных устройств путем монтажа необходимых компонент на макетном поле платы. Питание платы осуществляется от источника постоянного тока, напряжением от 9 до 12 В. Поскольку микросхема XC9572XL требует для своей работы напряжение питания 3,3 В, оно формируется непосредственно на плате вторичными источниками питания.

Дополнительно на плате расположены четыре светодиода, четыре кнопки и два тактовых генератора, которые подключены к выводам ПЛИС. Светодиоды зажигаются подачей уровня логической 1 на цепи LED1–LED4. Кнопки при нажатии формируют сигнал логического нуля на линиях BOT1–BOT4.



Два независимых генератора формируют сигналы GCK1 и GCK2. Используя переключки можно осуществить настройку каждого из генераторов на частоты 25, 50 и 100 МГц.

25 МГц	50 МГц	100 МГц

Внешние цепи подключены к определенным выводам ПЛИС. Соответствие цепей и ножек микросхемы определяется при помощи usf файла. Пример:

```
NET "BOT1" LOC = "P12" ;  
NET "BOT2" LOC = "P13" ;  
NET "BOT3" LOC = "P14" ;  
NET "BOT4" LOC = "P16" ;  
NET "GCK1" LOC = "P43" ;  
NET "GCK2" LOC = "P44" ;  
NET "LED1" LOC = "P5" ;  
NET "LED2" LOC = "P6" ;  
NET "LED3" LOC = "P7" ;  
NET "LED4" LOC = "P8" ;
```

Поле NET означает имя цепи, LOC — номер ножки ПЛИС с добавленной вначале латинской буквой P. В usf файле должны перечисляться только используемые цепи. Пустые строки с "лишними" цепями, не задействованными в проекте, перечислять не следует (иначе проект может не работать).

Если в схеме часть входов или выходов объединены в вектор, то необходимо поставить в соответствие ножки микросхемы для каждого разряда вектора. Например, для использования линейки светодиодов в качестве вектора LED(3 downto 0) в usf файле следует прописать:

```
NET "LED(0)" LOC = "P5" ;  
NET "LED(1)" LOC = "P6" ;  
NET "LED(2)" LOC = "P7" ;  
NET "LED(3)" LOC = "P8" ;
```

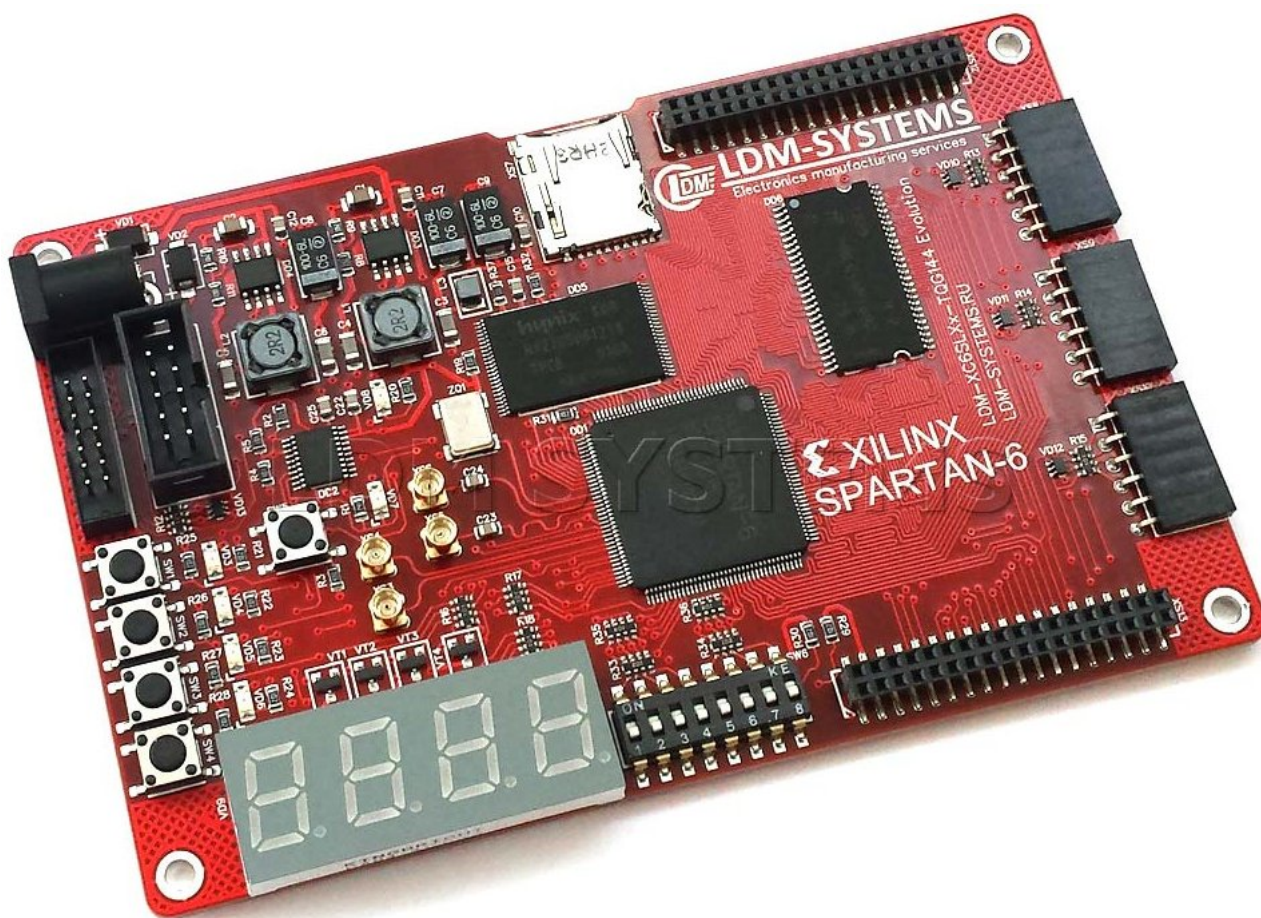
Программирование ПЛИС через схемотехническое описание

1. Запустить ISE и выбрать режим создания нового проекта New project. В окне New Project Wizard задать имя проекта (Name) и выбрать тип проекта (Top-level sources type) — Schematic. Нажать Next. Выбрать параметры микросхемы: Family — XC9500XL CPLDs, Device — XC9572XL, Package — VQ44, Speed — "-10".
2. Создать новый лист схемы (Project — New Source, тип Schematic). Затем разместить на нем необходимые библиотечные элементы (Add — Symbol). Выполнить необходимые соединения элементов (Add — Wire). Входы и выходы вывести на левый и правый край схемы соответственно, подсоединив к ним коннекторы (Add I/O Marker). Переименовать цепи, идущие к коннекторам в соответствии с назначением сигналов (двойной щелчок по коннектору, Category: Nets).
3. Составить ucf файл (Project — New Source, тип Implementation Constraints File), ставящий в соответствие названия цепей схемы и номеров выводов ПЛИС, задействованных в проекте. Пример ucf файла приведен на стр. 68.
4. Выполнить компиляцию схемы (Process — Implement Top Module) и исправить ошибки.
5. Осуществить программирование ПЛИС, для этого, запустить программу iMPACT (Tools — iMPACT), выполнить сканирование (Boundary Scan), инициализацию канала (File — Initialize Chain), в окне Assign New Configuration Files выбрать соответствующий файл проекта, очистить и запрограммировать ПЛИС (Erase, Programm).

Программирование ПЛИС на языке VHDL

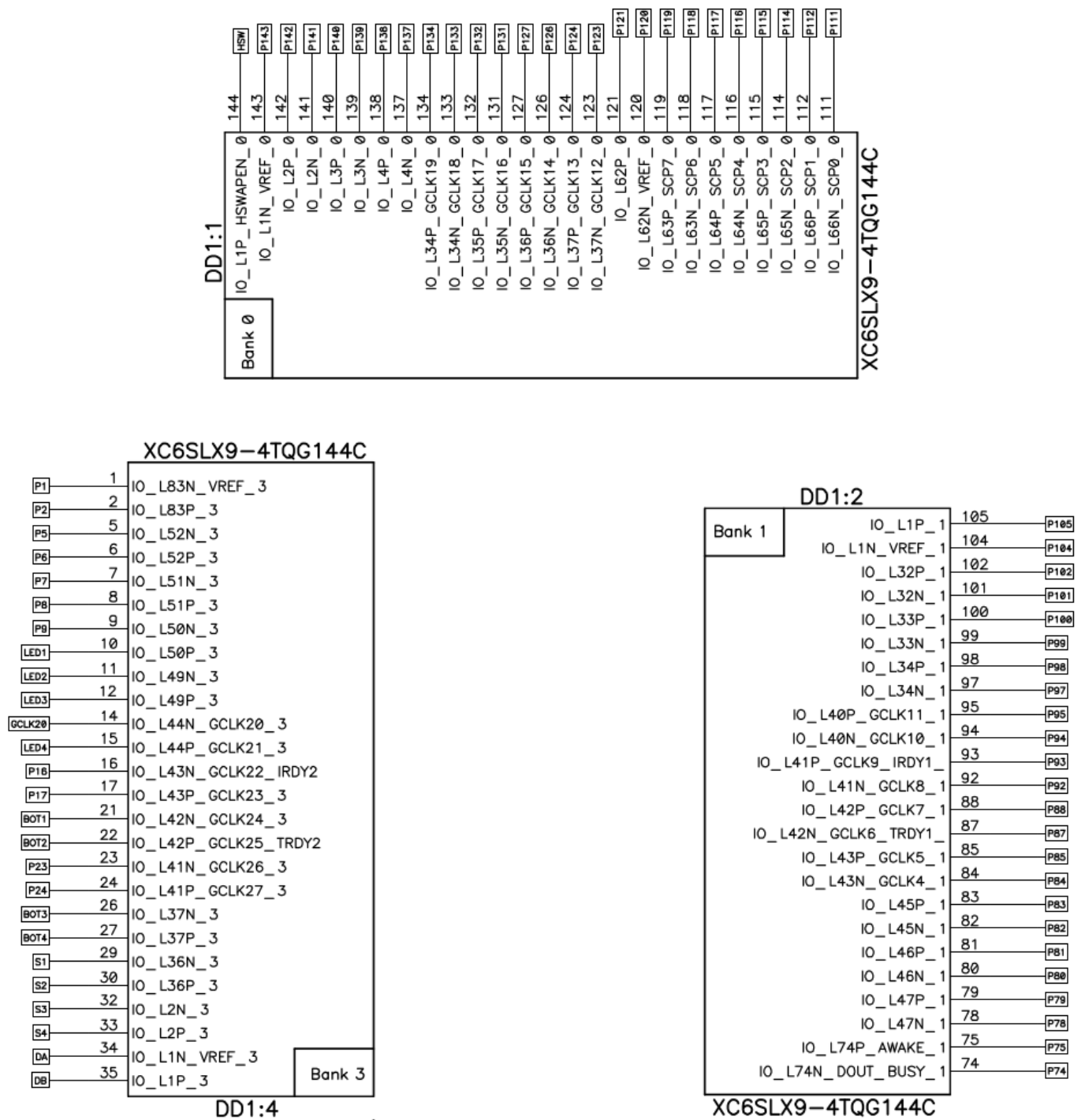
1. Запустить ISE и выбрать режим создания нового проекта New project. В окне New Project Wizard задать имя проекта (Name) и выбрать тип проекта (Top-level sources type) — HDL. Нажать Next. Выбрать параметры микросхемы: Family — XC9500XL CPLDs, Device — XC9572XL, Package — VQ44, Speed — "-10". Предпочитаемый язык (Preferred Language) — VHDL.
2. Создать новый модуль описания (Project — New Source, тип VHDL Module). Нажать Next. Указать имена цепей ввода-вывода (Port Name) и направление (Direction). Цепи, подключаемые к кнопкам — in, к светодиодам — out. В результате получится шаблон на языке VHDL, в котором необходимо вписать необходимый функционал.
3. Составить ucf файл (Project — New Source, тип Implementation Constraints File), ставящий в соответствие названия цепей схемы и номеров выводов ПЛИС, задействованных в проекте. Пример ucf файла приведен на стр. 68.
4. Выполнить компиляцию схемы (Process — Implement Top Module) и исправить ошибки.
5. Посмотреть статистическую информацию о работе синтезатора (Project — Design Summary/Reports).
6. Визуально проконтролировать работу синтезатора: как реализован проект на уровне описания цепей и сигналов (Tools — Schematic Viewer — RTL); как транслятор ISE распорядился ресурсами микросхемы (Tools — Schematic Viewer — Technology).
7. Осуществить программирование ПЛИС, для этого, запустить программу iMPACT (Tools — iMPACT), выполнить сканирование (Boundary Scan), инициализацию канала (File — Initialize Chain), в окне Assign New Configuration Files выбрать соответствующий файл проекта, очистить и запрограммировать ПЛИС (Erase, Programm).

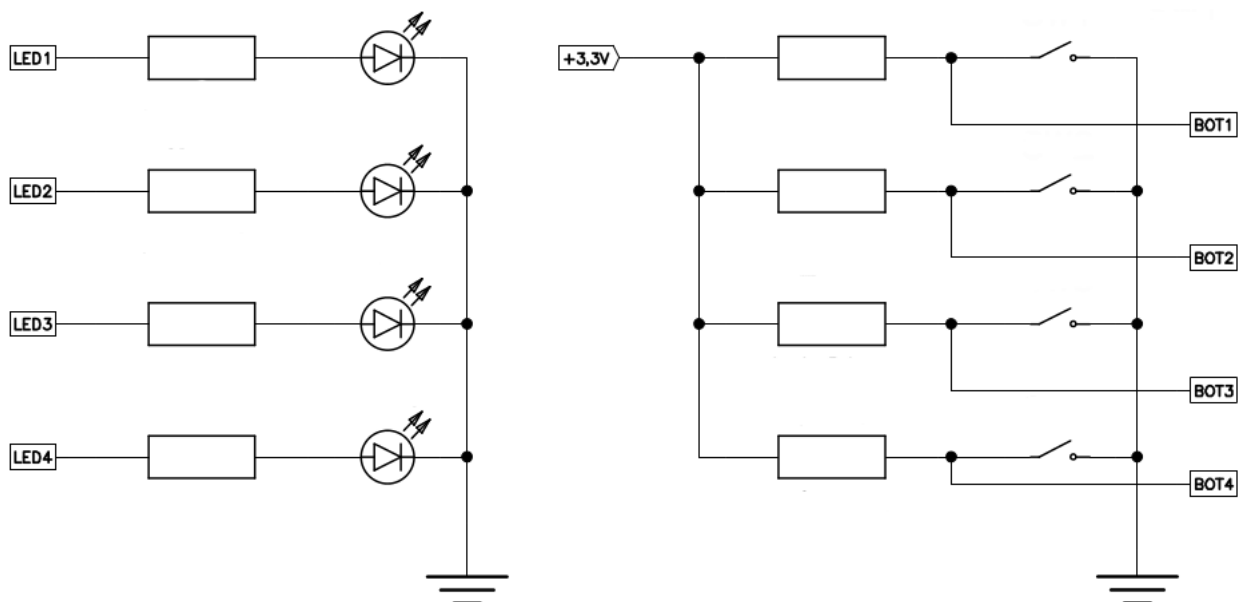
Конструктор-ПЛИС FPGA (задачи 11-12)



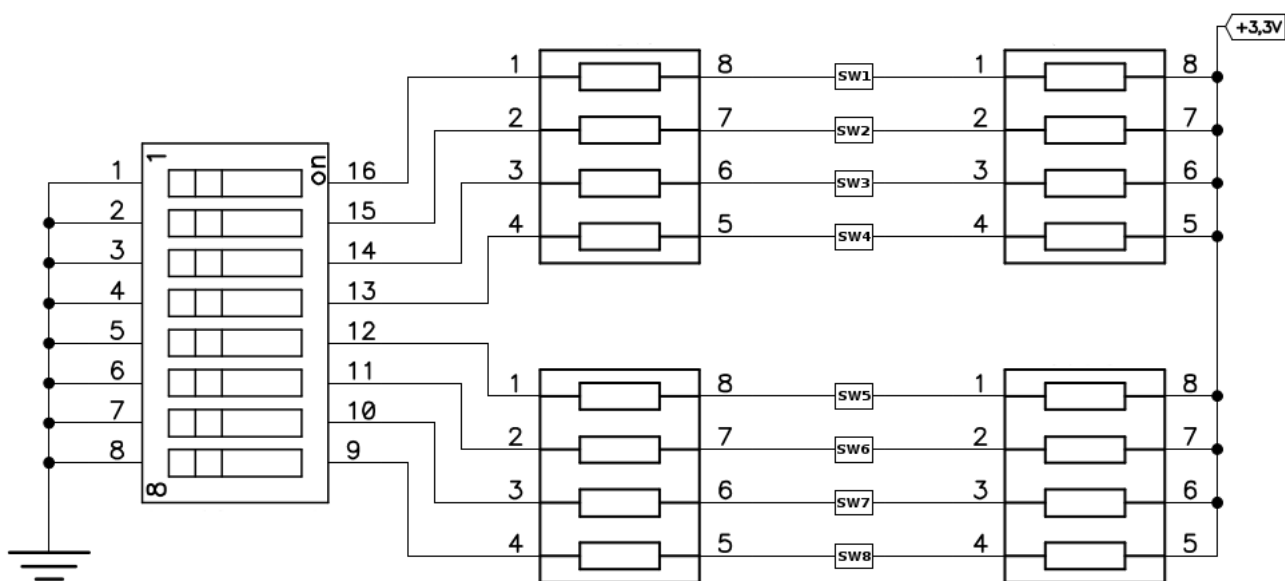
Конструктор-ПЛИС фирмы LDM-SYSTEMS предназначен для макетирования устройств, проектируемых на ПЛИС FPGA XC6SLX фирмы Xilinx, а также сборки законченных устройств путем присоединения дополнительных модулей к разъемам расширения.

Отладочная плата оснащена: 4-значным семисегментным LED дисплеем, 4-я MMCX ВЧ разъемами (2 входа и 2 выхода), 4-я кнопками, 8-ю переключателями, 4-я светодиодами, слотом для micro SD карт, 3-я разъемами с ESD защитой (по 4 линии I/O на разъем), 2-я разъемами для подключения внешних модулей расширения типа Tower (многоярусной компоновки) LDM-XC6-MEZ Tower и LDM-XC6-MEZ FT245R VGA Tower. На плате имеется встроенный тактовый генератор с частотой 25 МГц. Питание платы осуществляется от источника постоянного тока, напряжением от 9 до 12 В.





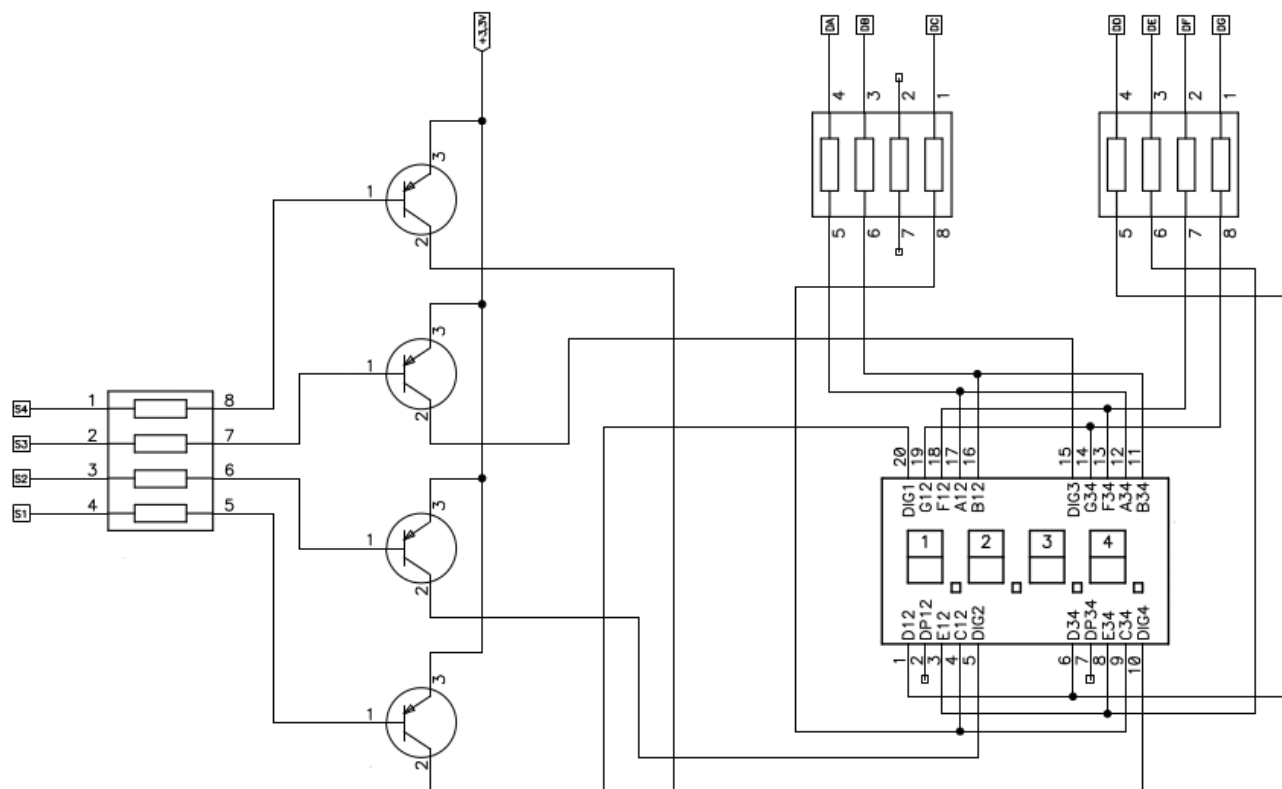
Светодиоды зажигаются подачей уровня логической 1 на цепи LED1–LED4. Кнопки при нажатии формируют сигнал логического нуля на линиях BOT1–BOT4.



Переключатели в замкнутом состоянии формируют сигнал логического нуля на линиях SW1–SW8.

Семисегментный индикатор представляет собой светодиодную матрицу 4x7. Сигналы S1–S4 соответствуют разрядам, а DA–DG — сегментам индикатора.

Генератор формирует частоту 25 МГц на линии GCLK20.



Пример содержимого ucф файла:

```

NET "BOT1"    LOC = "P21" ;
NET "BOT2"    LOC = "P22" ;
NET "BOT3"    LOC = "P26" ;
NET "BOT4"    LOC = "P27" ;
NET "SW1"     LOC = "P45" ;
NET "SW2"     LOC = "P46" ;
NET "SW3"     LOC = "P47" ;
NET "SW4"     LOC = "P48" ;
NET "SW5"     LOC = "P50" ;
NET "SW6"     LOC = "P51" ;
NET "SW7"     LOC = "P55" ;
NET "SW8"     LOC = "P56" ;
NET "GCLK20"  LOC = "P14" ;
NET "LED1"    LOC = "P10" ;
NET "LED2"    LOC = "P11" ;
NET "LED3"    LOC = "P12" ;
NET "LED4"    LOC = "P15" ;
NET "S1"      LOC = "P29" ;
NET "S2"      LOC = "P30" ;
NET "S3"      LOC = "P32" ;
NET "S4"      LOC = "P33" ;
NET "DA"      LOC = "P34" ;
NET "DB"      LOC = "P35" ;
NET "DC"      LOC = "P38" ;
NET "DD"      LOC = "P40" ;
NET "DE"      LOC = "P41" ;
NET "DF"      LOC = "P43" ;
NET "DG"      LOC = "P44" ;

```

Задача по изучению ЦАП и АЦП выполняется на ПЛИС XC6SLX с использованием модуля расширения LDM-XC6-MEZ Tower, на котором распаяны готовые элементы для построения ЦАП и АЦП. Ко входу АЦП в качестве источника сигнала подключен фотоэлемент, а выход ЦАП — к динамику.

Программирование ПЛИС FPGA

1. Запустить ISE и выбрать режим создания нового проекта New project. В окне New Project Wizard задать имя проекта (Name) и выбрать тип проекта (Top-level sources type) — HDL. Нажать Next. Выбрать параметры микросхемы: Family — Spartan6, Device — XC6SLX9, Package — TQG144, Speed — "-3". Предпочитаемый язык (Preferred Language) — VHDL.
2. Создать новый модуль описания (Project — New Source, тип VHDL Module). Нажать Next. Указать имена цепей ввода-вывода (Port Name) и направление Direction. Цепи, подключаемые к кнопкам — in, к светодиодам — out. В результате получится шаблон на языке VHDL, в котором необходимо вписать необходимый функционал.
3. Составить ucf файл (Project — New Source, тип Implementation Constraints File), ставящий в соответствие названия цепей схемы и номеров выводов ПЛИС, задействованных в проекте. Пример ucf файла приведен на стр. 74.
4. Выполнить компиляцию схемы (Process — Implement Top Module) и исправить ошибки.
5. Посмотреть статистическую информацию о работе синтезатора (Project — Design Summary/Reports).
6. Осуществить программирование ПЛИС, для этого, запустить программу iMPACT (Tools — iMPACT), выполнить сканирование (Boundary Scan), инициализацию канала (File — Initialize Chain), в окне Assign New Configuration Files выбрать соответствующий файл проекта и запрограммировать ПЛИС (Erase, Programm).

Краткое описание языка VHDL

Язык VHDL был разработан в 1983 г. для формального описания логических схем на всех этапах разработки электронных систем. Very High Speed Integrated Circuit Hardware Description Language переводится как язык описания высокоскоростных интегрированных схем. Синтаксис и особенности языка рассмотрены в [5–7].

Классы. В VHDL не предусмотрена возможность создавать новые классы, они уже встроены и их всего три:

1. Константы (constant) могут объявляться в интерфейсах, архитектурах, процессах, процедурах и функциях. После объявления константы должно быть её определение (constant const: bit := '1';). Если символов := нет в декларации константы, то константа называется задержанной. Такие константы могут быть в разделе деклараций пакета. Соответствующая полная декларация должна быть в теле пакета.
2. Переменные (variable) объявляются в области деклараций процесса, функции или процедуры и видны только там. Есть возможность сделать переменную глобальной — объявить ее в архитектуре или в интерфейсе следующим VHDL-кодом:

```
shared variable y: bit;
```

Переменным можно задавать начальное значение при объявлении:

```
variable y: bit:='1';
```

3. Сигналы (signal) введены для описания цифровых схем и обладают многими свойствами реальных сигналов в цифровой технике: скорость распространения, способы взаимодействия с другими сигналами и пр. Они объявляются в области объявлений архитектурного тела и видны во всем этом теле или в области объявлений интерфейса, тогда они видны во всех архитектурах, использующих данный интерфейс. Порты по умолчанию являются сигналами.

Объекты. Прежде чем использовать какой либо объект, его нужно объявить членом какого-нибудь класса. Например, объявляются три объекта `x`, `y` и `gnd`:

```
constant gnd: bit := '0';
variable y: bit;
signal x: bit;
```

Далее можно записать следующий VHDL код:

```
y := gnd;
x <= y;
```

Оператор присваивания для сигналов отличается от аналогичных для объектов других классов. Этот код дан для примера т. к. можно написать `x <= gnd`; или еще проще — `x <= '0'`; Без первых двух классов можно обойтись в любом синтезируемом проекте, а без класса `signal` — нет.

Комментарии отделяются от основного VHDL-кода двойным штрихом и действуют до конца строки:

```
-----
-- Пример комментариев --
-----

-- Управление индикаторами сигналов LINK
nLEDL1 <= GND when nLINKA = '0' else TRI; -- канал A
nLEDL2 <= GND when nLINKB = '0' else TRI; -- канал B
```

Типы данных определены в библиотеке `std` (по аналогии пользователь может создавать свои типы данных):

```
type boolean is (false, true); -- логический тип
type bit is ('0', '1'); -- битовый тип
type character is (...); -- символьный тип
type severity_level is (note, warning, error, failure); -- перечислимый тип
type integer is range -2147483647 to 2147483647; -- целый тип
type real is range -1.0E308 to 1.0E308; -- вещественный тип
type time is range -2147483648 to 2147483647; -- целый тип
subtype natural is integer range 0 to integer'high; -- подтип натуральных чисел
subtype positive is integer range 1 to integer'high; -- подтип положительных чисел
```

Здесь `integer'high` — наибольшее значение типа `integer`. В данном случае `integer'high` = 2147483647 и наоборот `integer'low` = -2147483647.

Массивы и вектора:

```
type string is array (positive range <>) of character;  
type bit_vector is array (natural range <>) of bit;  
signal adddr: bit_vector (7 downto 0); -- определили 8-ми битовый вектор  
constant txt: string := "VHDL"; -- определили строковую константу
```

Все библиотечные типы `std` переопределять нет необходимости, они по умолчанию подключаются к проекту.

Tun STD_LOGIC. Для описания логических схем или их моделирования были введены типы `std_ulogic` и `std_ulogic_vector`:

```
type std_ulogic is (  
    'U', -- Неинициализированный. Этот сигнал еще не был задан  
    'X', -- Сильный неизвестный сигнал  
    '0', -- Сильный 0  
    '1', -- Сильная 1  
    'Z', -- Высокий импеданс  
    'W', -- Слабый неизвестный сигнал  
    'L', -- Слабый 0  
    'H', -- Слабая 1  
    '-' -- Не имеет значения какой сигнал  
);  
type std_ulogic_vector is array (natural range <> ) of std_ulogic;  
subtype std_logic is resolved std_ulogic;  
type std_logic_vector is array (natural range <> ) of std_logic;
```

`std_ulogic` — перечислимый тип, состоящий из 9 символов. Символ 'U' обозначает неинициализированные данные (например ячейки оперативной памяти или триггеры после включения питания), данный символ используется только для моделирования. Символы 'X', '1' и '0' обозначают сильные сигналы, например на выходах микросхемы. 'Z' — это простой высокий импеданс, подходит как для моделирования, так и для синтеза, в ПЛИС такой сигнал можно реализовать только на выходных каскадах микросхемы. '-' используется только при моделировании, когда все равно какое значение имеет сигнал.

Слабые сигналы можно подавить сильными при смешивании. Они могут использоваться как нагрузка на открытом коллекторе или как сопротивление, соединенное с кнопкой. Пример:

```
out <= '0' when in = '1' else 'H';
```

Некоторые ПЛИС содержат такие резисторы в своих выходных каскадах, а некоторые синтезаторы даже подключают их, встретив такой VHDL-код. В крайнем случае, это можно сделать вручную через настройки выходов.

Подтип `std_logic` образуется из типа `std_ulogic` с помощью разрешающей функции `resolved`, которая также объявлена и определена в библиотеке `std_logic_1164`. Для разрешения данных типа `std_logic`, функция `resolved` использует разрешающую таблицу. Данная таблица определяет, какой сигнал получится в итоге "столкновения" двух сигналов типа `std_logic`. Например, если приходят сигналы '0' и '1' получается '0' (преобладает сильный сигнал). В библиотеке `std_logic_1164` переопределены операторы `and`, `or`, `nand`, `nor`, `xor`, `xnor` и `not` для типов `std_ulogic` и `std_logic`.

В отличие от библиотеки `STD`, библиотеку `std_logic_1164` нужно подключать с помощью следующих VHDL инструкций:

```
library ieee;  
use ieee. std_logic_1164.all;
```

Массивы — именованный набор однотипных переменных, доступ к которым осуществляется по индексу, могут быть одномерными и многомерными, что определяется количеством индексов. Обычно синтезаторы не поддерживают многомерные массивы, за исключением массивов векторов. В зависимости от наличия или отсутствия размерности, объявление типа массива может быть двух видов:

```
type type_name is array (range) of element_type;  
type type_name is array (type range <>) of element_type;
```

Первый вид используется для типов с заданной размерностью и при объявлении объекта размерность уже будет задана типом:

```
type BYTE is array (7 downto 0) of BIT; -- 8-ми битовый тип  
signal BYT1: BYTE; -- 8-ми битовый массив
```

Второй вид используется для типов без размерности, тогда размерность должна указываться при объявлении объекта:

```
type MATRIX is array (POSITIVE range <>) of REAL; -- вектор реальных чисел  
signal MATRIX1: MATRIX (1 to 3); -- вектор из трех реальных чисел
```

Операции. В VHDL можно использовать: логические операции (and, or, nand, nor, xor, xnor); операции сравнения (=, /=, <, <=, >, >=); операции сдвига (sll, srl, sla, sra, rol, ror); операции сложения (+, -, &); операции смены знака числа (+, -); умножение деление (*, /, mod, rem); прочие (**, abs, not).

Операции перечислены в порядке увеличения приоритета выполнения (т. е. от низшего к высшему). При равенстве приоритетов, операции выполняются слева направо и скобки дают высший приоритет.

Операторы в языке VHDL могут быть синтезируемыми и нет. Операторы делятся на последовательные и параллельные, простые и составные. Последовательные операторы в VHDL (не надо путать с последовательной логикой) определяют алгоритмы выполнения подпрограмм и процессов. Они выполняются в том порядке, в каком расположены в VHDL-коде и могут использоваться только в теле процесса, функции или процедуры. Это следующие операторы: assert (последовательный); case; exit; if; loop; next; null; report; return; wait; последовательные операторы присваивания. Примеры:

```
-- Использование операторов loop, next, exit, if, присваивание
-- Несинтезируемый фрагмент VHDL-кода
neq := 0;
L1: for i in 0 to 31 loop
  next L1 when mem1 (i) = mem2 (i);
  for j in 7 downto 0 loop
    crash := 1;
    exit L1 when (mem1 (i)(j) = 'X' or mem2 (i)(j) = 'X');
    crash := 0;
    if (mem1 (i)(j) /= mem2 (i)(j)) then neq := neq+1; end if;
  end loop;
end loop L1;
-- Использование операторов wait, case, if
-- Синтезируемый фрагмент VHDL-кода
process begin
  wait until clk;
  if (trig = '1') then
    case state is
      when red => state <= green;
      when green => state <= yellow;
      when yellow => state <= red;
    end case;
  end if;
end process;
```

Параллельные операторы в VHDL используются для определения блоков и процессов, которые описывают общее поведение или структуру проекта. Параллельные операторы VHDL, выполняются (или не выполняются) параллельно друг другу и независимо от расположения в VHDL-коде. К таким операторам относятся: `assert` (параллельный); `block`; `generate`; `process`; применение компонентов (`port map ...`); параллельные операторы присваивания.

Интерфейс — это те же входные и выходные сигналы, а также какие-нибудь передаваемые в архитектуру параметры, оформленные по правилам языка VHDL. В общем виде объявление интерфейса выглядит следующим образом:

```
entity NAME_ENTITY is
  generic (
    -- Здесь указываются параметры);
  port (
    -- Здесь указываются порты);
    -- Область объявлений
begin -- ставится, если далее следуют параллельные операторы интерфейса
  -- Параллельные операторы (обычно assert), вызовы подпрограмм
end NAME_ENTITY;
```

Приведем пример общего случая VHDL интерфейса, при этом не забываем объявлять библиотеки, если необходимо:

```
library ieee;
use ieee. std_logic_1164.all;
entity ADDER_N is
  generic (
    -- Задаем разрядность сумматора
    N: natural := 4);
  port (
    a, b: in std_logic_vector (0 to N-1); -- Операнды
    s: out std_logic_vector (0 to N-1); -- Сумма
    c: out std_logic -- Перенос
  );
  constant GND: std_logic := '0';
  constant VCC: std_logic := '1';
  constant TRI: std_logic := 'Z';
begin
  assert N > 1
  report "N is too small"
  severity error;
end ADDER_N;
```

А теперь то же самое, но без generic:

```
library ieee;
use ieee. std_logic_1164.all;
entity ADDR4 is
  port (
    a, b: in std_logic_vector (0 to 3); -- Операнды
    s: out std_logic_vector (0 to 3); -- Сумма
    c: out std_logic -- Перенос
  );
end ADDR4;
```

А может быть и пустой интерфейс. Это когда нет никаких связей с внешним миром — для создания тестовых VHDL моделей:

```
library ieee;
use ieee. std_logic_1164.all;
entity TEST_BENCH is
end TEST_BENCH;
```

Интерфейс может быть как внешним для проекта, так и внутренним, применяемым для компонентов проекта. Порты VHDL-интерфейса могут быть следующих видов: *in* — входной порт (только для чтения значения сигнала); *out* — выходной порт (только для записи значения сигнала); *inout* — двунаправленный порт (чтение и запись); *buffer* — выходной порт, значение которого можно считать; *linkage* — двунаправленный порт с ограниченным использованием (практически не используется).

Интерфейс может быть связан с несколькими архитектурами.

Архитектура содержит две основные части:

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

В общем виде архитектура выглядит следующим образом:

```
architecture NAME_ARCHITECTURE of NAME_ENTITY is
  -- Описания типов данных
  -- Функции и процедуры
  -- Компоненты более низкого уровня иерархии
  -- Описания сигналов и глобальных переменных
begin
  -- Исполняемые операторы
end NAME_ARCHITECTURE;
```

Теперь напишем простой VHDL код для компонента NAND с двумя ВХОДАМИ:

```
library ieee;
use ieee. std_logic_1164.all;

entity NAND2 is
  port (
    a, b: in std_logic;
    c: out std_logic
  );
end NAND2;

architecture aaa of NAND2 is
begin
  c <= not (a and b);
end aaa;
```

Покажем, как используются компоненты и сигналы:

```
library ieee;
use ieee. std_logic_1164.all;

entity NAND2X3 is
  port (
    a, b, c, d: in std_logic;
    q: out std_logic
  );
end NAND2X3;

architecture bbb of NAND2X3 is
  signal s1, s2: std_logic;
  component NAND2
  port (
    a, b: in std_logic;
    c: out std_logic);
  end component;
begin
  D1: NAND2 port map (a => a, b => b, c => s1);
  D2: NAND2 port map (a => c, b => d, c => s2);
  D3: NAND2 port map (a => s1, b => s2, c => q);
end bbb;
```

Данный VHDL код использует компонент NAND2, описанный ранее. Это *структурное описание*. Такой метод удобен в крупных проектах, т. е. когда большой проект разбивается на несколько компонент поменьше.

Можно написать проще и понятнее, используя *поведенческое описание*: Кроме того, учтем, что $!(a*b) * !(c*d) = (a*b) + (c*d)$. В итоге VHDL-код запишется следующим образом:

```
library ieee;
use ieee. std_logic_1164.all;

entity NAND2X3 is
  port (
    a, b, c, d: in std_logic;
    q: out std_logic
  );
end NAND2X3;

architecture ddd of NAND2X3 is
begin
  q <= (a and b) or (c and d);
end ddd;
```

Процессы. В общем виде VHDL-процесс записывается так:

```
LABEL: process (лист чувствительности)
-- область деклараций
begin
  --VHDL операторы
end process;
```

Лист чувствительности нужен исключительно для моделирования. Он говорит модели (testbench) о необходимости заново промоделировать данный процесс если изменилось значение сигнала, перечисленного в листе чувствительности. В лист чувствительности следует вносить минимально возможный перечень сигналов — от этого зависит скорость работы модели. Например, при моделировании D-триггера, управляемого фронтом следует указать сигнал тактирования, но не следует указывать сигнал данных, а при моделировании D-триггера, управляемого уровнем — оба сигнала. Это потому, что управляемый уровнем триггер может пропускать на выход сигнал данных, при разрешающем сигнале на входе тактирования, а управляемый фронтом — всегда хранит информацию до прихода следующего фронта тактирования.

Язык VHDL допускает процессы без меток, а также отсутствие листа чувствительности при наличии в теле процесса хотябы одного

оператора wait (например при моделировании в симуляторе). Пример процесса:

```
architecture arch of RegUpr is
signal regupr: std_logic_vector (7 downto 0);
begin
  process (wr, cs, addr)
  begin
    if (cs = '1' and wr = '0') then
      case addr is
        when "01" => regupr <= datain;
        when others => null;
      end case;
    end if;
  end process;
  en1 <= regupr (0);
  en2 <= regupr (1);
end arch;
```

Атрибуты — это различные характеристики объектов VHDL. Они делятся на predetermined и пользовательские. Мы рассмотрим и те и другие, хотя использование пользовательских атрибутов очень обременительно и мало что дает, но может кому-то и понравится.

Предetermined атрибуты делятся на три группы — атрибуты сигналов, атрибуты массивов и атрибуты типов.

S'active	TRUE, если было присвоение, но текущее значение еще прежнее
S'delayed(t)	Значение сигнала за время t перед вычислением атрибута
S'event	TRUE, если происходит изменение сигнала
S'last_active	Время от последнего присвоения значения сигналу до момента вычисления атрибута
S'last_event	Время от последнего изменения сигнала до момента вычисления атрибута
S'last_value	Последнее присвоенное сигналу значение
S'stable(t)	TRUE, если не происходило изменение сигнала в течение времени t
S'transaction	TRUE, если происходит очередное присвоение значения сигналу
S'quiet	FALSE, если было присвоение, но текущее значение еще прежнее
A'left(N)	Левая граница N-го индекса массива A
A'right(N)	Правая граница N-го индекса массива A
A'high(N)	Верхняя граница N-го индекса массива A
A'low(N)	Нижняя граница N-го индекса массива A
A'range(N)	Диапазон N-го индекса массива A
A'reverse_range(N)	Обратный диапазон N-го индекса массива A
A'length(N)	Длина диапазона N-го индекса массива A
T'base	Базовый тип данных
T'left	Левая граница значений T

T'right	Правая граница значений T
T'high	Верхняя граница значений T
T'low	Нижняя граница значений T
T'pos(X)	Позиция значения X в наборе значений T
T'val(N)	Значение элемента в позиции N набора значений T
T'succ(X)	Значение в наборе значений T, на одну позицию большее X
T'pred(X)	Значение в наборе значений T, на одну позицию меньшее X
T'succ(X)	Значение в наборе значений T, на одну позицию вправо от X
T'pred(X)	Значение в наборе значений T, на одну позицию влево от X

Чтобы поймать строб сигнала, нам как раз подходит атрибут S'event. S'event дает TRUE, если происходит изменение сигнала, а это ни что иное как фронт сигнала (строб). Такой атрибут любой синтезатор воспримет правильно и подключит его ко входу синхронизации триггера, что нельзя сказать о других атрибутах, которые в основном используются для моделирования или носят вспомогательные функции.

Итак, мы выбрали атрибут, но этого мало. Еще нужно указать какой фронт мы будем ловить. Для определенности, пусть нам нужно срабатывание по переднему фронту. Тогда это будет соответствовать выражению `clk'event and clk = '1'`. Обратному фронту соответствует выражение `clk'event and clk = '0'`.

Подпрограммы. В VHDL используется два вида подпрограмм — это функции и процедуры. В библиотеке `std_logic_1164` описаны различные функции, в том числе и `rising_edge ()`/`falling_edge ()` определение которых выглядит так:

```
function rising_edge (signal s: std_ulogic) return boolean is
begin
    return (s'event and s = '1');
end;

function falling_edge (signal s: std_ulogic) return boolean is
begin
    return (s'event and s = '0');
end;
```

А так как мы используем библиотеку `std_logic_1164`, то спокойно можем задействовать эти функции:

```
if rising_edge (ale) then
    ...
end if;
```

Пример процедуры:

```
procedure IM_CPU (res, w: in std_logic;  
                  a: out std_logic) is  
begin  
  if (res = '1') then  
    a := '0';  
  else  
    a := w;  
  end if;  
end;
```

Предметный указатель

Least Significant Bit, 54

Maximum Length Sequence, 32

VHDL, 77

архитектура, 83

атрибуты, 86

вектора, 79

интерфейс, 82

классы, 77

комментарии, 78

константы, 77

лист чувствительности, 85

массивы, 79, 80

объекты, 78

операторы, 81

операции, 81

переменные, 77

поведенческое описание, 85

подпрограммы, 87

процедуры, 87

процессы, 85

сигналы, 77

структурное описание, 84

тип STD_LOGIC, 79

типы данных, 78

функции, 87

Алгебра логики, 7

Алиасинг, 57

Аналого-цифровой преобразователь,
53

компенсационного типа, 58

напряжение — частота, 58

последовательного счета, 58

прямого преобразования, 59

с двойным интегрированием, 58

сигма-дельта, 59

Аналоговый мультиплексор, 10

Асинхронный счетчик, 39

Асинхронный триггер, 17

Бинарный код, 45

Бинарный разряд, 45

Бинарный счетчик, 38

Бит, 45

Быстродействие, 19

Вес младшего разряда, 54

Весовой код, 46

Восьмеричная система счисления, 45

Время задержки, 19

Вычитатель, 38

Вычитающий счетчик, 38

Групповая структура счетчика, 39

Двоичная система счисления, 45

Двоично-десятичный код, 45

Двоичный код, 45

Двоичный сумматор, 37

Двоичный счетчик, 38

Двоичный триггер, 17

Декодер, 9

Делители частоты, 39

Демультимплексор, 10

Дерево Уоллеса, 48

Децимация, 54

Дешифратор, 9

- Диаграмма Карно, 12
Дизъюнкция, 9
Динамический диапазон, 54
Динамическое управление, 18
Дифференциальная нелинейность, 54
Дополнительный код, 38, 46
Дребезг контактов, 40

Запрещенное состояние, 19

Инвертор, 8
Интегральная нелинейность, 55
Интерполяция, 54
Информационный сигнал, 17

Карта Карно, 12
Код Баркера, 31
Код Грея, 45
Кодер, 10
Кольцевой сдвигающий регистр, 28
 с инверсией, 28
Комбинационная логика, 9
Комбинационная схема, 9
Коммутатор, 10
Конъюнкция, 9
Коэффициент деления, 39
Коэффициент пересчета, 38

Линейные ЦАП/АЦП, 53
Логическое сложение, 9
Логическое умножение, 9

М-последовательность, 32
Максимальная частота
 дискретизации, 54
Матрица постоянного импеданса, 56
Матричный умножитель, 48
Меандр, 40
Метод последовательного
 приближения, 58
Модуль счета, 38
Модуль счетчика, 38
Мультиплексор, 10

Наложение спектров, 57
Направление счета, 38
Невесовой код, 46
Неравнозначность, 9
Нетактируемый триггер, 17
Нормальная форма И, 12
Нормальная форма ИЛИ, 12

Однобитовый компаратор, 11
Основные логические элементы, 8
Ошибка квантования, 57

Параллельно-последовательный
 счетчик, 39
Параллельный регистр, 27
Параллельный сумматор, 37
Параллельный счетчик, 39
Параллельный умножитель, 48
Передискретизация, 54
Переключатель два из трех, 11
Период регистра сдвига, 28
Повторитель, 8
Позиционная система счисления, 45
Полная дизъюнкция, 12
Полная конъюнкция, 12
Полный вычитатель, 38
Полный сумматор, 37
Полувывчитатель, 38
Полусумматор, 37
Пороговая логическая схема, 11
Последовательность максимальной
 длины, 32
Последовательный регистр, 28
Последовательный сумматор, 38
Последовательный счетчик, 39
Последовательный умножитель, 48
Преобразователь кодов, 11
Производительность, 57

Разрешающая способность, 57
Разрешение, 57
Разряд, 27, 38
Разрядность, 53

- Реверсивный
 сдвигающий регистр, 28
 счетчик, 38
- Регистр, 27
- Регистр хранения, 27
- Сброс, 18
- Сдвигающий регистр, 28
 кольцевой, 28
 кольцевой с инверсией, 28
 реверсивный, 28
 с линейной обратной связью, 28
 с параллельным вводом, 28
 с параллельным выводом, 28
- Секвенциальная логика, 9
- Сигма-дельта модулятор, 56
- Синхронный счетчик, 39
- Синхронный триггер, 17
 с динамическим управлением, 18
 со статическим управлением, 17
- Скважность, 40
- Смещение нуля, 54
- Состояние прозрачности, 21
- Статическое управление, 17
- Сумматор, 37
- Суммирующий счетчик, 38
- Схема контроля четности, 12
- Счетный триггер, 23
- Счетчик, 38
- Тактирование спадом, 18
- Тактирование уровнем, 17
- Тактирование фронтом, 18
- Тактируемый триггер, 17
- Терада, 45
- Точность АЦП, 57
- Третье состояние, 27
- Триггер, 17
- Умножитель, 48
- Усиление, 54
- Установка, 18
- Хранение, 18
- Цепная R-2R-схема, 56
- Цифро-аналоговый преобразователь,
 53
 взвешивающего типа, 55
 лестничного типа, 56
 передискретизации, 56
 с широтно-импульсной
 модуляцией, 56
- Цифровой компаратор, 11
- Цифровой мультиплексор, 10
- Частота дискретизации, 54
- Шестнадцатеричная система
 счисления, 45
- Широтно-импульсный модулятор, 56
- Шифратор, 10
- Эффективная разрядность, 53

Литература

1. И.Т. Трофименко, Е.В. Лебедева, Н.С. Седлецкая. Практикум по радиоэлектронике. Учеб. пособие. — М. Изд-во МГУ, 1997.
2. У. Титце, К. Шенк. Полупроводниковая схемотехника. Пер. с нем. В двух томах. — М. Додека-XXI, 2008.
3. Джонс М.Х. Электроника - практический курс (2-е издание, 2006)
4. Бойт К. Цифровая электроника. 2007 г.
5. П. Хоровиц, У. Хилл. Искусство схемотехники. Пер. с англ. — М. БИНОМ, 2014 г.
6. И.Е. Тарасов. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL Горячая Линия - Телеком, 2005, ISBN: 5-93517-242-9.
7. ГОСТ Р 50754-95 Язык описания аппаратуры цифровых систем VHDL. Описание языка.
8. Сайт по VHDL: <http://allhdl.ru/vhdl.php>
9. Описание ISE: http://ru.wikibooks.org/wiki/Xilinx_WebPACK_ISE
10. Сайт LDM-SYSTEMS: <http://www.ldm-systems.ru>
11. Е. А. Бехтенов. Цифро-аналоговые и аналого-цифровые преобразователи. Новосибирск 2010.
<http://www.inp.nsk.su/students/radio/2013/TSANI/Lab4.pdf>

ISBN 978-5-8279-0102-0

