Московский государственный университет
имени М. В. Ломоносова

Физический факультет
кафедра фотоники и физики микроволн

О. Ю. Волков

ПРАКТИКУМ ПО РАДИОЭЛЕКТРОНИКЕ

ЦИФРОВЫЕ СХЕМЫ НА СИМУЛЯТОРЕ ПЛИС

Методическое пособие для студентов
и преподавателей практикума

Москва — 2021

УДК 378.162.33, 004.942, 53.083.8
ББК 22.3

Практикум по радиоэлектронике: цифровые схемы на симуляторе ПЛИС

Учебно-методическое пособие для студентов физического факультета МГУ и преподавателей практикума по радиоэлектронике. М.: Физический факультет МГУ им. М.В. Ломоносова, 2021. — 49 с.

ISBN 978-5-8279-0104-4

Представлено переработанное описание цикла задач по цифровой электронике, ориентированное на работу с симулятором ПЛИС. Такой подход не требует наличия у студента самой микросхемы в процессе обучения, что позволяет проводить занятия как в лаборатории, так и дистанционно.

В пособие включены четыре задачи. Каждая задача сопровождается списком контрольных вопросов, знание которых необходимо для успешного выполнения упражнений практикума.

Студенты изучают современный подход цифровой схемотехники с использованием программируемых интегральных логических схем в среде проектирования ISE компании Xilinx. Методика симуляции работы проекта осваивается про помощи программы ISim.

В начале осваивается схемотехническое описания 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.

Издание основано на базе пособия Практикум по радиоэлектронике: цифровые схемы, 2016 г.

ВОЛКОВ Олег Юрьевич
Практикум по радиоэлектронике
цифровые схемы на симуляторе ПЛИС

Объем 2,8 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

Содержание

Установка системы автоматического проектирования Xilinx ISE

Программное обеспечения может быть установлено на системы Linux, Windows XP, Windows 7, а так же на Windows 8 и Windows 10 (но после установки требуются дополнительные действия). Для MAC OS надо поставить виртуальную машину, например с Linux.

Загрузка ПО осуществляется с сайта компании Xilinx по ссылке https://www.xilinx.com/downloadNav/vivado-design-tools/archive-ise.html (потребуется регистрация). Необходимо раскрыть пункт "14.7" (не пункт "14.7 Windows 10"). Затем необходимо выбрать тот архив, который лучше подходит под вашу систему.

- `Multi-File Download: ISE Design - 14.7 Full Product Installation` — разбитый на 4 части архив, предназначенный для файловых систем, которые не умеют работать с файлами более 2Г (например FAT32). Необходимо скачать все 4 файла и распаковать. Архив подходит и для Windows и для Linux.
- `ISE Design Suite - 14.7 Full Product Installation` — набор архивов для выбранной операционной системы. Необходимо скачать один из предложенных файлов — универсальный архив или укороченный под вашу операционную систему:
  - Full DVD Single File Download Image (TAR/GZIP - 7.78 GB)
  - Full Installer for Linux (TAR/GZIP - 6.09 GB)
  - Full Installer for Windows 7/XP/Server (TAR/GZIP - 6.18 GB)

Далее необходимо получить бесплатную лицензию для Webpack версии ISE по ссылке http://www.xilinx.com/getlicense

Необходимо помнить, что сам архив занимает от 6 до 8 гигабайт, а так же необходимо место для распаковки архива и инсталляции. Если

установка делается в виртуальную машину, например VirtualBox, то лучше создавать динамический диск размером не менее 50Г. Реально (для Linux) диск будет около 30Г в момент установки, пока не удалили исходный архив.

Архив не установится в папку, путь которой содержит русские буквы. При дальнейшей работе нельзя использовать русские буквы в именах фалов и путей к ним, а также всевозможные спецсимволы. Допустимы только латинские буквы, цифры и подчерк. Пример:

`C:\ISE` — сюда можно распаковать архив и поставить ISE;
`C:\ПЛИС` — нельзя распаковать архив и поставить ISE.

Для Windows 8 и Windows 10 после установки необходимо выполнить следующие действия:

1. Открыть папку `Xilinx\14.7\ISE_DS` ;
2. В ней перейти в `ISE\lib\nt64`, а затем найти и переименовать файл `libPortability.dll` на `libPortability.dll_original`;
3. В той же директории сделать копию `libPortabilityNOSH.dll` (скопировать и вставить в ту же папку, должен получится файл с именем `libPortabilityNOSH - копия.dll`);
4. В той же директории переименовать `libPortabilityNOSH.dll` в `libPortability.dll`;
5. Скопировать файл `libPortabilityNOSH - копия.dll`, вернуться в директорию `Xilinx\14.7\ISE_DS` и вставить скопированный файл в директорию `common\lib\nt64`;
6. Переименовать в той же директории файл `libPortability.dll` в `libPortability.dll_original`;
7. Переименовать там же файл `libPortabilityNOSH - копия.dll` в `libPortability.dll`.

Задача 1.

Комбинационные логические схемы

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

1.1. 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 ($\overline{A}$, $\neg A$, $notA$);
- 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 Мóргана)

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

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

- 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

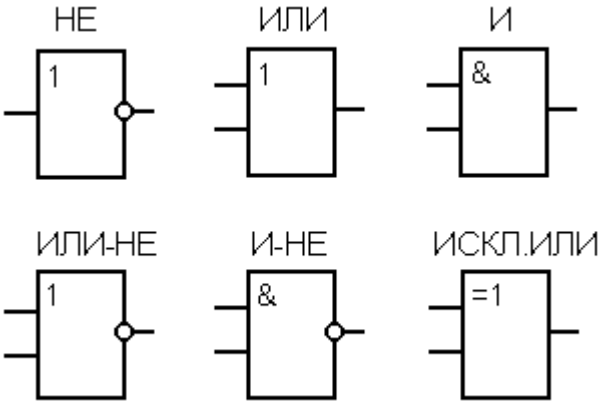
последовательно получим *повторитель*. Оба элемента могут быть использованы для усиления и задержки логического сигнала.

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. 1.1.



A	*B*	И	ИЛИ	ИСКЛ.ИЛИ
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

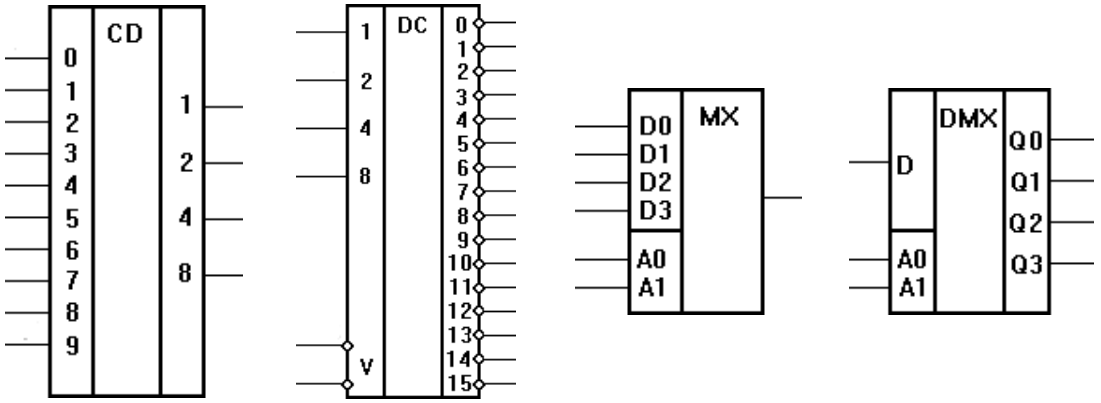
Рис. 1.1. Условные обозначения и таблица истинности некоторых логических элементов

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.



а б в г

Рис. 1.2. Условные обозначения некоторых комбинационных схем (а — шифратор, б — дешифратор, в — мультиплексор, г — демультиплексор)

Мультиплексор — устройство, имеющее 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 демультиплексоры, как правило, строятся на основе дешифраторов. Условные графические изображения шифратора, дешифратора, мультиплексора и демультиплексора приведены на рис. 1.2.

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:

$$\begin{gathered} X = A \wedge \overline{B}; \\ Y = (\overline{A} \wedge \overline{B}) \vee (A \wedge B); \\ Z = \overline{A} \wedge B. \end{gathered} \tag{1.1}$$

$X = 1$ 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:

$$Z = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C). \tag{1.2}$$

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.

Программируемые логические интегральные схемы 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 ограниченны, 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. 1.3.

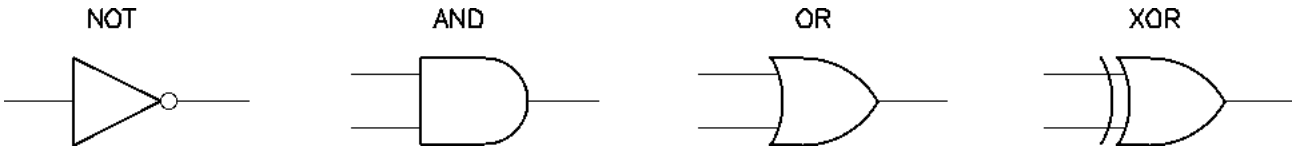


Рис. 1.3. Условные обозначения некоторых логических элементов в иностранном стандарте

1.2. Знакомство с системой Xilinx ISE

WebPACK ISE — средство проектирования цифровых устройств. Основное окно программы приведено на рис. 1.4. Цифрами обозначены: 1 — инструментальная панель (Toolbar); 2 — окно описания проекта (Source window); 3 — окно процессов (Process window); 4 — рабочий стол (Workspace); 5 — окно отчетов (Transcript window).

При первом запуске программы для подключения файла лицензий необходимо выполнить следующие дейсвия:

1. Запустить программу ISE. Закрыть окно Tip of the Day, сняв галочку Show Tips at Startup. Закрыть окно Xilinx License Error, нажав OK. Подождать запуска менеджера лицензий.

2. В окне Xilinx License Configuration Manager выбрать закладку Manage Licenses, нажать кнопку Load License и подключить файл лицензий, закрыть все окна, относящиеся к менеджеру лицензий.

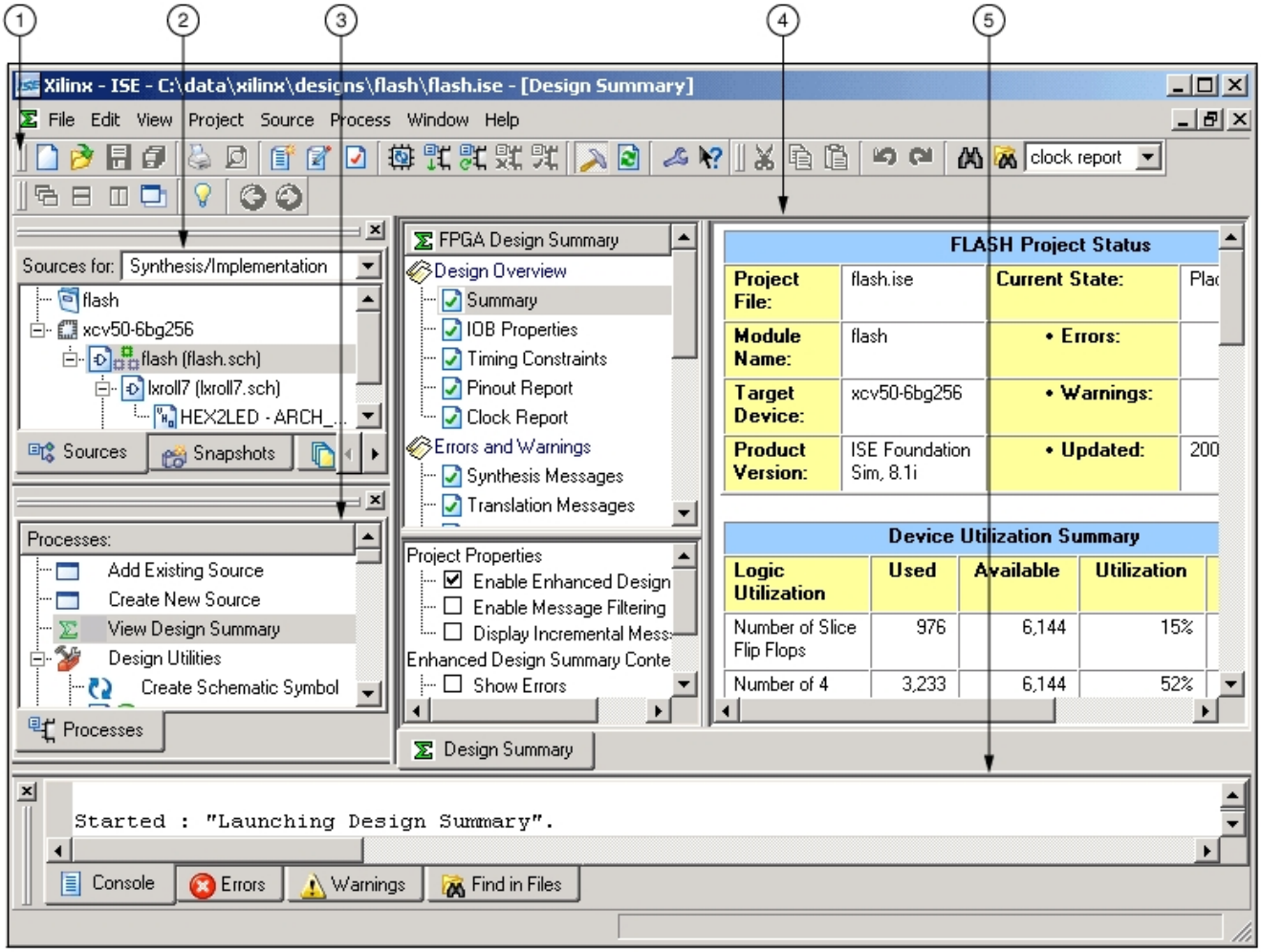


Рис. 1.4. Основное окно программы навигатора проекта ISE

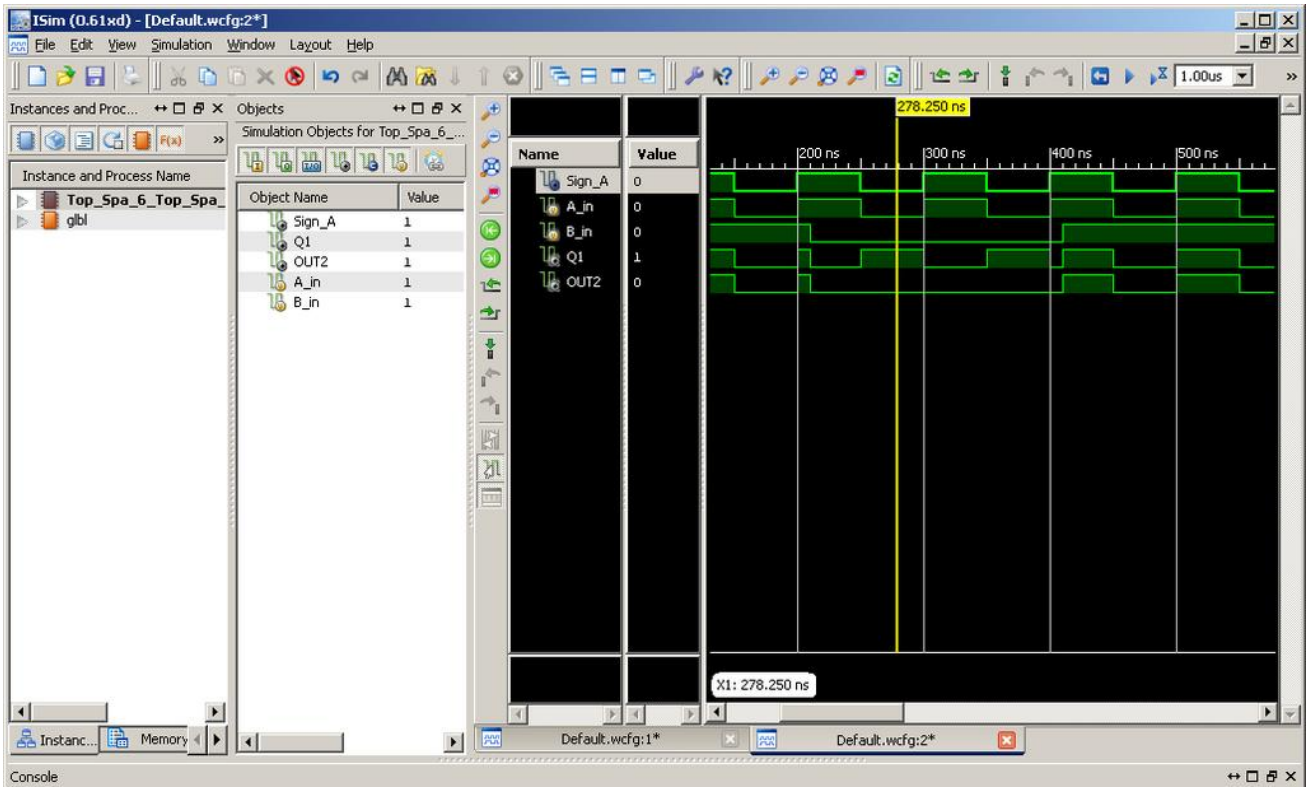


Рис. 1.5. Основное окно симулятора ISim

Система проектирования включает в себя программный симулятор ISim, основное окно которого приведено на рис. 1.5. В правой части окна выводятся результаты работы симулятора. Управление изменением масштаба по горизонтальной оси осуществляется кнопками (рис. 1.6).

Рис. 1.6. Кнопки изменения масштаба по горизонтали

1.3. Схемотехническое описание работы ПЛИС

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 sorces 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 Shematic). 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. 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 требуемый функционал.

4. В левой части окна войти в панель Design, переключить вид на симуляцию View: Simulation и активизировать файл проекта, который предполагается симулировать (Test Bench). 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.

5. В окне ISim выполнить View — Zoom — To Full View для получения обзорного вида на процесс симуляции.

Для управления входами тестируемой схемы в Test Bench файле вписывается порядок изменения уровней (внизу перед командой wait). Пример управления двумя сигналами in1 и in2:

```
in1 <= '0';   -- начальные значения, если не присвоены
in2 <= '0';   -- ранее, при объявлении
WAIT FOR 100 ns;
in1 <= '1';
WAIT FOR 100 ns;
in1 <= '0';
in2 <= '1';
WAIT FOR 100 ns;
in1 <= '1';   -- in2 уже установлен в '1';
WAIT;
```

1.4. Практическая часть

Перед началом выполнения упражнений необходимо установить и настроить среду разработки ISE (стр. 4). Описание работы с ISE подробно рассмотрено в [3] (глава 3).

Задача выполняется в программах ISE и ISim. Для каждого упражнения необходимо создать собственный проект, используя схемотехническое описание.

Упражнения:

1. Получить функции 2И, 2ИЛИ, 2ИСКЛЮЧАЮЩЕЕ_ИЛИ из набора элементов 2И-НЕ. Входы всех трех схем подключить к сигналам in1 и in2. Выходы — к сигналам outand, outor, outxor (каждую схему к своему сигналу). Записать таблицу истинности каждого элемента.

2. Собрать дешифратор на 4 состояния на основных логических элементах. Входы подключить к сигналам bit0 и bit1, выходы — к сигналам d0, d1, d2 и d3. Записать таблицу истинности.

3. Собрать схему переключателя два из трех на основных логических элементах. Входы подключить к синалам s1, s2 и s3, выход — к сигналу s. Записать таблицу истинности схемы.

4. Собрать схему контроля четности для 4-битного входного числа на основных логических элементах. Входы подключить к сигналам bit0, bit1, bit2 и bit3, выход — к сигналу p. Записать таблицу истинности схемы.

1.5. 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. Чем отличаются шифратор, дешифратор и преобразователь кодов?

6. Чем отличаются мультиплексор, демультиплексор, коммутатор?

7. В чем отличия между пороговой логической схемой и переключателем два из трех?

8. Как работает и устроена схема контроля четности?

9. Что такое цифровой компаратор? Как сравниваются числа большой разрядности?

Задача 2.

Триггеры

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

2.1. 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 $\overline{\text{Q}}$. Триггеры управляются сигналами, подаваемыми на информационные (и если есть тактовый) входы.

Информационный сигнал — сигнал, подаваемый на информационный вход триггера.

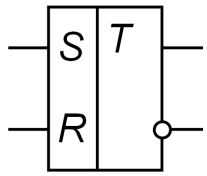
Нетактируемые или *асинхронные триггеры* изменяет свое состояние после прихода соответствующих информационных сигналов.

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 логической единицы на тактовом входе (прямой вход C) или нуля (инверсный вход 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 C, тактирование спадом или отрицательным фронтом).

На принципиальных схемах триггеры изображают прямоугольниками, внутри которых ставятся буквы T или TT.На специальном поле слева обозначаются входы: S, R, J, K, C и др. Инверсные (статические) входы и выходы обозначаются кружочком (∘). Динамически управляемые входы обозначаются или треугольничком или наклонной чертой. Символы ▷ или ╱ обозначают управление положительным фронтом, ◁, ╲ или ◁▷ — отрицательным фронтом.

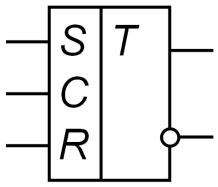


S	R	Q(t+1)	$\overline{Q}$(t+1)	Состояние
0	0	Q(t)	$\overline{Q}$(t)	хранение
1	0	1	0	установка
0	1	0	1	сброс
1	1	-	-	запрещенное

Рис. 2.1. Условное обозначение и таблица истинности асинхронного RS-триггера

Асинхронный RS-триггер (рис. 2.1) 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, $\overline{Q}$=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

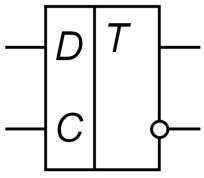
состояния в состояние хранения невозможен — триггер переходит или в нулевое, или в единичное состояние, а только потом в состояние хранения.



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

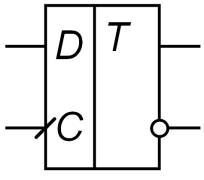
Рис. 2.2. Условное обозначение и таблица истинности синхронного RS-триггера

Синхронный RS-триггер (рис. 2.2) 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.



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

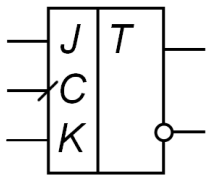
Рис. 2.3. Условное обозначение и таблица истинности синхронного D-триггера, (а — статическое управление, б — динамическое управление)

Синхронный одноступенчатый D-триггер со статическим управлением (рис. 2.3,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 0 на информационном входе (C=1, D=0).

Иными словами, при наличии тактового импульса (C=1) состояние триггера 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.

Синхронный D-триггер с динамическим управлением (рис. 2.3,б) имеет один информационный вход D 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.



J	K	Q(t+1)	$\overline{Q}$(t+1)	Состояние
0	0	Q(t)	$\overline{Q}$(t)	хранение
1	0	1	0	установка
0	1	0	1	сброс
1	1	$\overline{Q}$(t)	Q(t)	счет

Рис. 2.4. Условное обозначение и таблица истинности синхронного JK-триггера

Синхронный 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. 2.4. Триггер постоянно находится в состоянии хранения, кроме момента прихода очередного синхроимпульса. В этот момент времени поведение триггера определяется таблицей истинности. 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. Иначе состояние триггера сохраняется.

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. 2.5). 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.

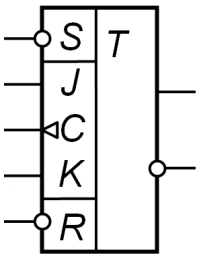
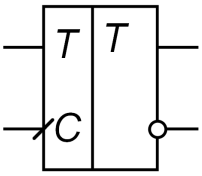


Рис. 2.5. Условное обозначение JK-триггера с дополнительными инверсными входами R и S



T	Q(t+1)	$\overline{Q}$(t+1)	Состояние
0	Q(t)	$\overline{Q}$(t)	хранение
1	$\overline{Q}$(t)	Q(t)	счет

Рис. 2.6. Условное обозначение и таблица истинности синхронного T-триггера

Асинхронный T-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. 2.6.

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*.

2.2. VHDL описание триггеров

Краткое описание языка VHDL приведено на стр. 40.

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 $\overline{\text{RS}}$-триггер, то есть триггер, у которого управляющими сигналами является не логическая 1, а логический 0. Пример такого триггера изображен на рис. 2.7 слева. Для надежного срабатывания RS-триггера, построенного на двух логических элементах длительность управляющих импульсов должна превышать удвоенное время задержки (быстродействие) одного элемента.

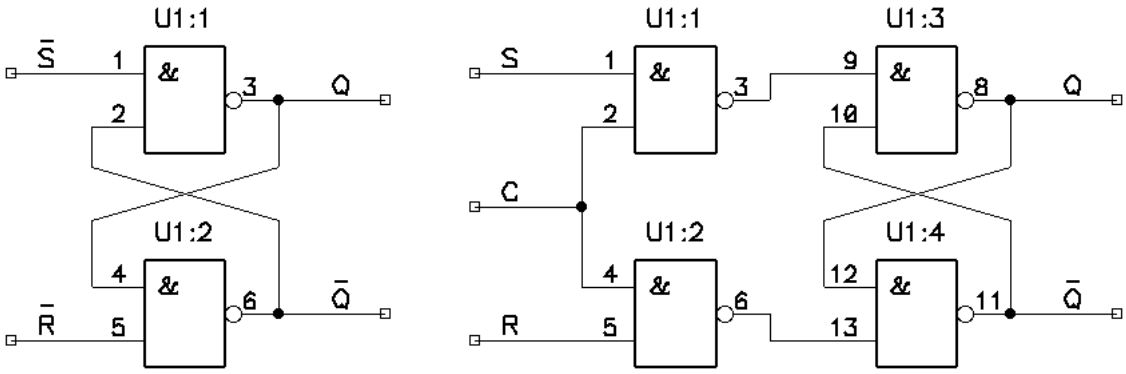


Рис. 2.7. Асинхронный $\overline{\text{RS}}$-триггер и синхронный RS-триггер на элементах 2И-НЕ

Синхронный RS-триггер может быть собран из четырех элементов 2И-НЕ, его схема приведена на рис. 2.7 справа. Для надежного срабатывания такого триггера длительность управляющих импульсов должна превышать утроенное время задержки одного элемента.

Работа простейших триггеров в языке VHDL может быть записана путем описания функций логических элементов, например RS триггер может быть записаны следующим образом:

```
-- RS-триггер
Q   <= not ( R or neQ );
neQ <= not ( S or Q );
```

При описании RS триггеров в виде процесса через условные операторы можно реализовать триггеры таким образом, чтобы вместо

запрещенного состояния у них был приоритет сброса. Такой способ следует использовать, если в логике работы схемы запрещенное состояние не используется или даже вредно.

Синхронные триггеры с динамическим тактированием описывать через логические элементы как правило не имеет смысла. Например D-триггер и T-триггер можно описать намного проще, в виде процесса:

```
-- 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 (in) можно только считывать, а выходам (out) — только присваивать значения. Внутренние сигналы внутри архитектуры можно и устанавливать

и считывать. Если схеме требуется считывание значения выходного сигнала,то следует изменить вид выхода с out на buffer (не путать с двунаправленным входом/выходом вида inout). Пример:

```
entity trigger is
    port (bot1, bot2 : in     STD_LOGIC;  -- вход, только считывание
          led1, led2 : buffer STD_LOGIC;  -- выход, установка и считывание
          led3, led4 : out    STD_LOGIC); -- выход, только установка
end trigger;
```

Другой способ — добавить промежуточный сигнал для работы схемы, 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;
```

2.3. Описание работы 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 sorces 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 требуемый функционал.

3. Визуально проконтроллировать 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).

4. 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 требуемый функционал.

5. В левой части окна войти в панель Design, переключить вид на симуляцию View: Simulation и активизировать файл проекта, который предполагается симулировать (Test Bench). 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.

6. В окне ISim выполнить View — Zoom — To Full View для получения обзорного вида на процесс симуляции.

2.4. Практическая часть

Задача выполняется в программах ISE и ISim. Для каждого упражнения необходимо создать собственный проект. Упражнения, выполняются на языке VHDL:

1. Собрать одновременно RS-триггер и синхронный RS-триггер со статическим управлением. Входы R и S обоих триггеров подключить к сигналам R и S, а вход C синхронного триггера — к C. Выходы Q и $\overline{Q}$ первого триггера — к сигналам Q1 и neQ1, второго — Q2 и neQ2. Записать таблицы истинности триггеров.

2. Собрать одновременно следующие схемы: D-триггер, управляемый уровнем, входы C и D подключить к сигналам C и D, выход к Qd1; D-триггер, управляемый фронтом, входы C и D подключить к сигналам C и D, выход к Qd2; T-триггер, вход C подключить к выходу второго D-триггера, T — к сигналу T, выход — к Qt.

3. Собрать JK-триггер со сбросом. Входы C, R, J и K — к соответствующим сигналам. Выходы — к Q и neQ. Составить таблицу истинности JK триггера.

2.5. 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. Как выглядит таблица истинности для триггера, имеющего прямые или инверсные входы R и S, выходы Q и $\overline{\text{Q}}$.
7. Как выглядит таблица истинности для триггера, имеющего прямые или инверсные входы R, S и C (тактирование уровнем).
8. Как выглядит таблица истинности для триггера с прямыми или инверсными входами D и C (тактирование уровнем или фронтом).
9. Как выглядит таблица истинности JK триггера, тактируемого положительным или отрицательным фронтом, имеющего входы J1, K1, $\overline{\text{J2}}$, $\overline{\text{K2}}$.
10. Как выглядит таблица истинности JK триггера, тактируемого положительным или отрицательным фронтом, имеющего входы J, K, $\overline{\text{R}}$ и $\overline{\text{S}}$.

Задача 3.

Регистры и формирователи кода

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

3.1. Основные понятия

Регистры — группа определенным образом соединенных триггеров, предназначенных для временного запоминания многоразрядных чисел и выполнения преобразований над ними. 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 — 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 — сдвигающий 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 — сдвигающий 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.

Таблица 3.1. 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

Код Баркера — это числовая последовательность, состоящая из символов вида $a_i = \pm 1$, где $i = 1, 2, \ldots, N$. Коды Баркера существуют только для ограниченного числа значений N и приведены в таблице 3.1.

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:

$$r(n) = \begin{cases} 1 & (\text{для } n = 0) \\ 0 & (\text{для } 0 < n < N,\ N \text{ четное}) \\ \pm 1/N & (\text{для } 0 < n < N,\ N \text{ нечетное}) \end{cases}, \qquad (3.1)$$

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

$$r(n) = \begin{cases} 1 & (\text{для } n = 0) \\ 0 & (\text{для } 0 < n < N,\ n = N-2, N-4, \ldots) \\ \pm 1/N & (\text{для } 0 < n < N,\ n = N-1, N-3, \ldots) \end{cases}, \qquad (3.2)$$

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.

М-последовательность или *последовательность максимальной длины (Maximum length sequence)* — псевдослучайная двоичная последовательность, 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 $x^n + \ldots$, где степени полиномов приведены в таблице 3.2 (старший разряд используется во всех полиномах). Например, для $n = 3$ число 5

Таблица 3.2. 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

означает $5 = 2^2 + 2^0$, что соответствует полиному $x^3\ xor\ x$.

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

```
-- 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;
```

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.

3.3. Практическая часть

Задача выполняется в программах ISE и ISim. Для каждого упражнения необходимо создать собственный проект (на языке VHDL).

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

Упражнения:

1. Собрать 4-х разрядный регистр сдвига. Для тактирования использовать сигнал C, V — вход переноса регистра, а сигнал R должен осуществлять сброс всех разрядов в нулевое состояние. Значения разрядов регистра должны отображаться в виде вектора Q(3 downto 0).

2. Собрать формирователь кода Баркера для заданного количества элементов N на основе кольцевого сдвигающего регистра. Тактирование схемы осуществить от сигнала CLK. Выход подключить к сигналу Barker. На сигнал Marker вывести импульс соответствующий началу кодовой последовательности.

3. Собрать формирователь М-последовательности разрядности n. Тактирование схемы осуществить от сигнала CLK. Сигнал STOP должен приостанавливать счет. Выход 4-х младших разрядов подключить к вектору M(3 downto 0). Найти период повторения последовательности.

3.4. 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 сдвигающий 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?

Задача 4.

Сумматоры и счетчики

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

4.1. 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:

$$0+0=0,\ \ 0+1=1,$$
$$1+0=1,\ \ 1+1=10.$$

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

$$0+0+0=0,\quad 0+0+1=1,$$
$$1+0+0=1,\quad 1+0+1=10,$$
$$0+1+0=1,\quad 0+1+1=10,$$
$$1+1+0=10,\ \ 1+1+1=11.$$

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 одноразрядного 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.

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

$$0 - 0 = 0,\ \ 0 - 1 = -1,$$
$$1 - 0 = 1,\ \ 1 - 1 = 0.$$

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 счетчиках 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.

Мультивибраторами называют электронные устройства, генерирующие электрические колебания, близкие по форме к прямоугольным.

Одновибратор или *ждущий, моностабильный мультивибратор* является генератором импульса заданной длительности. Схема имеет одно устойчивое, и одно квазиустойчивое состояние. Переход из устойчивого состояния происходит при подаче внешнего сигнала, обратный переход выполняется самопроизвольно.

Скважность — отношение периода следования прямоугольного импульса к длительности импульса.

Коэффициент заполнения — величина обратная скважности.

Меандр — периодический сигнал прямоугольной формы, у которого длительность импульса равна длительности паузы. Скважность такого сигнала равна 2.

4.2. 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.

Дла 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:

```
-- после описания IEEE библиотеки (чтобы работало сложение векторов)
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- после architecture .... и begin, но перед end Behavioral;
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:

```
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 достижении нуля.

При построении делителя частоты на N можно использовать сигнал старшего разряда счетчика, однако при этом *скважность* 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.

При организации счетчиков можно использовать тип 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 плис (числа разрядов) можно указать используемый диапазон. Например:

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

4.3. Практическая часть

Задача выполняется в программах ISE и ISim. Для каждого упражнения необходимо создать собственный проект (на языке VHDL).

Параметры: n и s — коэффициент деления и скважность сигнала, формируемая делителем частоты, τ — длительность импульса, формируемого ждущим мультивибратором, N — число пересчета реверсивного счетчика.

Упражнения:

1. Собрать двухразрядный двоичный сумматор, используя элементарные логические операции (НЕ, И, ИЛИ, ИСКЛЮЧАЮЩЕЕ_ИЛИ). Первое число должно формироваться сигналом

A(1 downto 0), второе — B(1 downto 0). Результат вывести на сигнал Y(2 downto 0). Записать таблицу истинности схемы.

2. Собрать делитель частоты сигнала CLK, формирующий импульсы с заданной скважностью и частотой. Выход подключить к сигналу Q.

3. Собрать ждущий мультивибратор, запускаемый сигналом RUN и формирующий импульс, длительностью τ. Выход подключить к сигналу Q.

4. Собрать четырехразрядный реверсивный счетчик с коэффициентом пересчета N, увеличивающий значение на 1 при подаче импульса на сигнал P и уменьшающий на 1 при по сигналу M. Сброс счетчика в положение "0000" должен осуществляться по сигналу R. Результат счета должен отображаться в двоичном коде на векторе Q(3 downto 0). Записать таблицу всех возможных состояний счетчика, начиная с "0000".

4.4. 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?

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.

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:

```
-- 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 pozitive is integer range 1 to integer'high;-- подтип положительных чисел
```

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

Тип 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.

Подтип std_logic образуется из типа std_ulogic с помощью разрешающей функции resolved, которая использует разрешающую таблицу, определяющую какой сигнал получится в итоге "столкновения" двух сигналов типа std_logic. Например, если приходят сигналы '0' и 'H' получается '0' (преобладает сильный сигнал). Библиотеку std_logic_1164 нужно подключать:

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

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:

```
-- 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;
-- 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;
```

Пример VHDL интерфейса:

```
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;
```

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); out — выходной порт (только для записи); inout — двунаправленный порт (чтение и запись); buffer — выходной порт, значение которого можно считывать. Интерфейс может быть связан с несколькими архитектурами.

Архитектура содержит две основные части, содержащие описания (декларации) и 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;
```

Простой VHDL код для компонента NAND с двумя входами может быть записан следующим образом:

```
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 $\overline{\overline{A \wedge B} \wedge \overline{C \wedge D}} = (A \wedge B) \vee (C \wedge D)$.

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;
```

Пример процесса:

```
process (BOT1,BOT2)
begin
  if(BOT1 = '0' and BOT2 = '0') then
    LED<="0001";
  elsif(BOT2 = '0') then
    LED<="0010";
  elsif(BOT1 = '0') then
    LED<="0100";
  else
    LED<="1000";
  end if;
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).

Атрибуты — это различные характеристики объектов VHDL. Они делятся на предопределенные и пользовательские. Предопределенные делятся на три группы — атрибуты сигналов, массивов и 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. Такой атрибут любой синтезатор воспримет правильно и подключит его ко входу синхронизации тригера, что нельзя сказать о других атрибутах, которые в основном используются для моделирования или носят вспомогательные функции.

Если необходимо срабатывание по переднему фронту сигнала clk следует использовать выражение clk'event and clk = '1'. Заднему 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;
```

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. Краткое описание программы Xilinx WebPACK ISE на русском языке: https://ru.wikibooks.org/wiki/Xilinx_WebPACK_ISE
9. Сайти компании Xilinx: https://www.xilinx.com