Министерство образования и науки Украины
Севастопольский национальный технический университет

Кафедра радиотехники и телекоммуникаций

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

«ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ С ИСПОЛЬЗОВАНИЕМ САПР QUARTUS II»

по дисциплине «Основы проектирования цифровых ИС»
для студентов дневной формы обучения
направления 6.0509 — Радиотехника

Севастополь
2012

УДК 621.374 (075.8)

Методические рекомендации по выполнению лабораторной работы «Проектирование комбинационных цепей с использованием САПР *Quartus II*» по дисциплине «Основы проектирования цифровых ИС» для студентов дневной формы обучения направления 0907 — радиотехника / Сост. В. В. Вертегел — Севастополь: Изд-во СевНТУ, 2012. — 31 с.

Целью рекомендаций является оказание помощи студентам-заочникам в выполнении лабораторной работы по дисциплине «Основы проектирования цифровых ИС».

Изложены краткие сведения о комбинационных цепях, методике их проектирования и о способах кодирования информации при ее обработке в цифровых узлах. Даны основные сведения о системе автоматизированного проектирования *Quartus II*. Рассмотрены вопросы проектирования с использованием *Verilog*-описаний и графического редактора указанной системы и верификации проекта с помощью сигнального редактора. Сформулированы варианты задания к лабораторной работе, требования к отчету, вопросы для самопроверки. Рекомендована дополнительная литература.

Методические рекомендации рассмотрены и утверждены на заседании кафедры радиотехники (протокол № ____ от _____ 2012 г.)

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензент:

СОДЕРЖАНИЕ

1. Цель работы и подготовка к работе

Целью работы является приобретение начальных сведений о комбинационных цепях, а также изучение некоторых возможностей проектирования таких цепей с использованием программируемых логических интегральных схем (ПЛИС) и системы автоматизированного проектирования (САПР) *Quartus* II.

Подготовка к работе включает в себя изучение данных методических указаний и рекомендованной в них литературы, подготовку ответов на приводимые вопросы для самопроверки, составление схемы проектируемого устройства (см. 4.2, 4.3) и проработку контрольного примера для нее (см. 5.4) в соответствии со своим вариантом задания (см. 4.1). Если студенту не известны простейшие логические операции и реализующие эти операции логические элементы, то до изучения материала, излагаемого ниже, необходимо ознакомиться с указанными сведениями (см., например, с. 120…126 [1]).

2. Краткие теоретические сведения

2.1. Понятие о комбинационных цепях

Узлы цифровых устройств (цифровые узлы) подразделяют на комбинационные и последовательностные. Комбинационные узлы называют также комбинационными цепями (КЦ) или комбинационными схемами, а также автоматами без памяти. Последовательностные узлы называют автоматами с памятью (АП), последовательностными цепями или схемами. КЦ и АП принципиально различаются между собой по функционированию (поведению). Значения выходных сигналов КЦ (после завершения переходных процессов, ограничивающих быстродействие любых цифровых узлов) зависят только от текущих значений входных сигналов. Как говорят, предыстория значения не имеет. В отличие от этого, значения выходных сигналов АП (состояние АП) зависят не только от значений сигналов, поступающих на их входы, но и от того состояния автомата, которое предшествует подаче этих входных сигналов и определяется ранее действовавшими на автомат входными сигналами (предысторией). Указанные функциональные (поведенческие) различия связаны со структурными различиями между КЦ и АП**:** для АП характерны внутренние обратные связи, т. е. замкнутые петли прохождения сигналов, а в КЦ такие обратные связи отсутствуют.

2.2. Способы кодирования информации при ее обработке в цифровых узлах

2.2.1. Двоичный код

Кодовые слова двоичного кода — это многоразрядные двоичные числа**:** если целое число *N* представить в виде:

$$N = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \cdots + a_0 \cdot 2^0,$$

где a_i = 0 или 1, i = 0, 1, ... , n –1, то последовательность двоичных символов a_{n-1} a_{n-2} **...** a_0 будет кодовым словом двоичного кода (n-разрядным двоичным числом) или, короче, двоичным кодом. Например, при n = 4 десятичное число 5 можно представить двоичным кодом 0101, т. е. $5_{10} = 0101_2$. При заданном значении n количество всевозможных слов двоичного кода равно 2^n .

2.2.2. Коды типа «1 из *N*»

Для кода «1 из N» кодовые слова представляют собой последовательности из N двоичных величин (нулей и единиц), т. е. являются N-разрядными (N —битными). В любом слове такого кода лишь в одном разряде может быть единица, а в остальных разрядах — нули. Т. е. кодовые слова различаются лишь позицией единицы. Количество всевозможных слов кода «1 из N» равно N .

2.2.3. Параллельные и последовательные коды. Однофазные и парафазные коды

Кодовые слова в цифровых устройствах могут передаваться в параллельном или последовательном коде. При параллельном коде все n разрядов слова передаются одновременно по n линиям (проводам). При последовательном коде разряды слова передаются поочередно (по одной линии). Все это справедливо для так называемых однофазных кодов.

Известны также парафазные коды, когда каждый разряд передается по двум линиям**:** по одной линии передается так называемое прямое значение разряда, а по другой — инверсное, т. е. каждый разряд слова в свою очередь представляется 2-разрядным словом параллельного кода.

Парафазные коды могут быть как параллельными, так и последовательными. В первом случае для передачи информации требуется 2n линий, а во втором — 2 линии.

2.2.4. Прямой, обратный и дополнительный коды

В большинстве ЭВМ разность A-B чисел A и B вычисляется путем сложения чисел A и – B , т. е. по формуле**:**

$$A - B = A + (-B). \qquad (1)$$

В связи с этим оказывается целесообразным рассмотрение прямого, обратного и дополнительного кодов. При использовании этих кодов ставится запятая перед старшим разрядом двоичного числа (целого или дробного). Перед запятой добавляется еще один

разряд, называемый знаковым разрядом.

Прямой код положительного числа (кодовое слово) совпадает с самим числом, т. е. в его знаковом разряде стоит нуль. Например, прямой код $[0{,}11001]_{пр}$ числа 0,11001 равен 0,11001или $[0{,}11001]_{пр} = 0{,}11001$. Для прямого кода отрицательного числа необходимо в знаковом разряде поставить единицу. Например, $[-0{,}11001]_{пр} = 1{,}11001$. Следовательно, нуль в прямом коде

имеет два представления**:** $[+0]_{пр} = 0{,}000\ldots$ и $[-0]_{пр} = 1{,}000\ldots$. Причем, складывая два первых прямых кода, не получим прямой код нуля, что не позволяет использовать формулу (1) для прямых кодов.

Чтобы представить в **обратном коде** двоичное число, заданное прямым кодом, необходимо сохранить значение знакового разряда, а после запятой заменить единицы на нули, а нули на единицы, если число отрицательное, и оставить разряды без изменения, если число положительное: $[0{,}11001]_{обр} = 0{,}11001$, $[-0{,}11001]_{обр} = 1{,}00110$. Нуль в обратном коде так же,

как и в прямом имеет два представления**:** $[+0]_{обр} = 0{,}000\ldots$ и $[-0]_{обр} = 1{,}111$**...**. Отметим, что $[x_{обр}]_{обр} = x_{пр}$. Существует способ сложения обратных кодов, позволяющий получить сумму также в обратном коде и использовать формулу (1). Для этого необходимо складывать обратные коды вместе со знаковыми разрядами, как обычные числа, и в случае возникновения единицы переноса из знакового разряда прибавить ее к младшему разряду суммы.

***Пример*:** $x1 = +0{,}10111$; $x2 = -0{,}00110$. $x1 + x2 = 0{,}10001$.

$[x1]_{обр} = 0{,}10111$; $[x2]_{обр} = 1{,}11001$; $[x1 + x2]_{обр} = 0{,}10001$.

$[x1]_{обр} + [x2]_{обр} =$ 0,10111

\+ 1,11001

10,10000

|_____ 1

$10{,}10001 = [x1 + x2]_{обр}$.

Дополнительный код числа совпадает с прямым, если число положительное, а для отрицательного числа ***x*** дополнительный код образуется по правилу $[x]_{доп} = [x]_{обр} + 1_{мл}$, где $1_{мл}$ — единица младшего разряда. Например, $[-0{,}11001]_{доп} = 1{,}00110 + 0{,}00001 = 1{,}00111$. Нуль в дополнительном коде имеет одно представление $[-0]_{доп} = [+0]_{доп} = 0{,}00000$. Это позволяет при ограниченном количестве разрядов использовать их для представления большего количества ненулевых значений двоичного числа. Точнее, этих значений оказывается на одно

(отрицательное) больше, чем в случае обратного кода (так, для целых чисел 1,00000 — дополнительный код отрицательного десятичного числа –32). Дополнительный код суммы чисел равен сумме дополнительных кодов слагаемых, причем единица переноса, возникающая при сложении старших (знаковых) разрядов дополнительных кодов, должна быть отброшена. Благодаря указанному свойству и формуле (1), при кодировании чисел дополнительными кодами операция вычитания может быть заменена операцией сложения.

2.2.5. Модифицированные коды

При сложении чисел в цифровых устройствах может произойти искажение старшего (знакового) разряда вследствие так называемого переполнения разрядной сетки и переноса в знаковый разряд. Такое явление возникает, в частности, если после знакового разряда в слагаемых следуют разряды дробной части числа (как в большинстве примеров, приведенных в 2.2.4), а результат сложения оказывается больше единицы. Для обнаружения ошибок, связанных с переполнением разрядной сетки и искажением знаковых разрядов, применяют специальный способ представления чисел — модифицированные коды, когда знакиизображаются двумя одинаковыми двоичными символами в старших разрядах перед запятой (двумя нулями для положительных чисел и двумя единицами для отрицательных). Существуют обратный и дополнительный модифицированные коды. Например, дополнительный мо дифицированный код $[–0,11001]_{доп\ мод} = 11,00111$. Факт переполнения обнаруживается по признаку появления числа с двумя различными знаковыми разрядами. Причем при сложении модифицированных кодов единица переноса из самого старшего знакового разряда прибавляется к младшему разряду (для обратных кодов) или отбрасывается (для дополнительных кодов).

2.3. Описание цифровых устройств при их проектировании

Проектируемое цифровое устройство (как говорят, объект проектирования или просто объект) прежде всего должно быть формально описано. Известны два основных подхода к описанию таких объектов**:** структурный (структурное описание или описание на структурном уровне) и поведенческий (поведенческое описание, описание на уровне поведения или функциональное описание).

Структурное описание — это описание устройства (его структуры) в виде совокупности компонентов и связей между компонентами. Такое описание может осуществляться в следующих формах**:**

1. Графическая форма — форма графически представленной схемы.
2. Текстовая форма — текст на общеупотребительном (например, на русском) языке

или на одном из специальных языков описания аппаратуры (например, ***Verilog***).

Поведенческое описание устройства — это описание зависимостей его состояния (выходных сигналов) от входных сигналов и от предшествующего (подаче упомянутых входных сигналов) состояния. Иначе говоря, поведенческое описание задает функцию или алгоритм, реализуемый устройством. В частности, комбинационные цепи описываются так называемыми переключательными (логическими) функциями.

Переключательной функцией называют логическую функцию, способную принимать одно из двух значений (0 или 1) в зависимости от значений нескольких ее аргументов. При проектировании КЦ с несколькими (n) выходами каждому из возможных наборов входных переменных должно быть поставлено в соответствие не одно двоичное значение функции, а определенное n-разрядное двоичное число (n двоичных значений, где $n > 1$). Тогда получается зависимость, которую можно интерпретировать как совокупность n переключательных функций. Эту совокупность называют системой переключательных функций (заметим, что значение n в общем случае не совпадает с количеством входных переменных).

Поведенческое описание может иметь следующие формы**:** 1. Описание требуемого логического преобразования таблицей функционирования (таблицей истинности). Например, табл. 1 описывает переключательную функцию F трех аргументов ($X2$, $X1$, $X0$) , которая должна быть реализована некоторой проектируемой комбинационной цепью. Для описания автоматов с памятью иногда используется представление таблиц функционирования в формах карт Карно [2].

2. Аналитическое описание логического преобразования, которое должно осуществляться проектируемым устройством (логические выражения). В частности, переключательная функция может быть представлена в так называемой совершенной дизъюнктивной нормальной форме (СДНФ). Для этого составляют дизъюнкцию тех наборов аргументов, на которых функция принимает единичное значение. Каждый из этих наборов записывается в виде конъюнкции аргументов. Причем нулевому значению какого-либо аргумента в наборе соответствует запись инверсии данного аргумента в конъюнкции (в коньюнктивном терме). Так, для функции, заданной табл. 1, СДНФ имеет вид**:**

$$F = \overline{X}_2\overline{X}_1\overline{X}_0 \vee \overline{X}_2X_1\overline{X}_0 \vee \overline{X}_2X_1X_0 \vee X_2X_1\overline{X}_0.$$

3. Описание требуемого логического преобразования с помощью временных диаграмм сигналов на входах и выходах объекта.

4. Описание объекта так называемой диаграммой состояний (см. с. 106 в [2]).

Таблица 1 — Таблица истинности комбинационной схемы

*X*2	*X*1	*X*0	*F*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

5. Текстовая форма, которая, как и при структурном описании, может представлять собой текст либо на общеупотребительном языке, либо на одном из специальных языков описания аппаратуры.

Любой из подходов к описанию объекта при любой форме может быть непосредственно использован для проектирования. Однако для не которых средств современных САПР могут оказаться предпочтительными какой-то определенный из подходов и определенная его форма. Тогда при неподходящем первоначальном описании устройства нужно перейти к предпочтительному описанию. Так, выше приведен пример перехода от табличной формы поведенческого описания КЦ к форме аналитического описания. Заметим, что в данной лабораторной работе изучается графический редактор САПР *Quartus II.* Для его применения требуется описание объекта на структурном уровне в графической форме. В связи с этим ниже (см. 2.4.2) приводится пример перехода к такому описанию от табличной формы поведенческого описания.

2.4. Мультиплексоры

Типовое условное обозначение мультиплексора представлено на рис. 1.

Здесь D_0, D_1, ..., $D_{K\text{-}1}$ — информационные входы, a_0, a_1, ..., a_{n-1} — адресные входы.

Мультиплексор можно назвать цифровым коммутатором**:** двоичное значение его выходного сигнала совпадает с двоичным значением сигнала на том информационном входе, десятичный номер которого равен двоичному числу $a_{n-1}\ a_{n-2}\ \ldots\ a_0$, поступающему в параллельном коде на адресные входы.

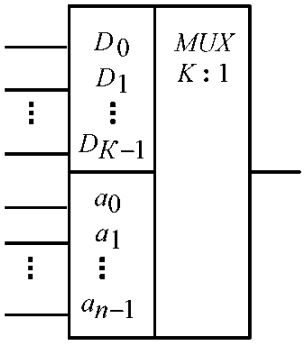


Рис. 1

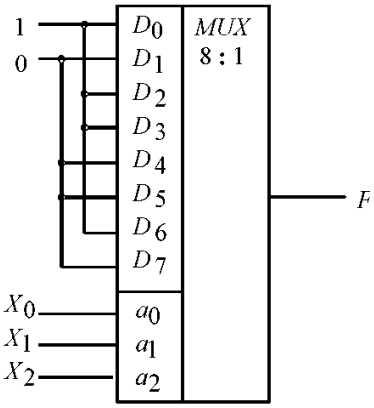


Рис. 2

При этом количество K информационных входов и количество n адресных входов связаны соотношением $K = 2^n$. В соответствии с приведенным поведенческим описанием мультиплексора (в текстовой форме) может быть составлена таблица истинности и записана СДНФ, что позволит перейти к структурному описанию в форме схемы, содержащей простейшие логические элементы. Можно убедиться, что в этой схеме будут отсутствовать обратные связи (см. 2.1).

2.4.2. Мультиплексор как универсальный логический модуль

При использовании мультиплексора в качестве универсального логического модуля (УЛМ) можно реализовать любую наперед заданную переключательную функцию. При этом входами УЛМ должны быть адресные входы мультиплексора, количество которых, следовательно, должно совпадать с количеством аргументов функции. На каждый информационный вход мультиплексора в данном случае должен подаваться постоянный сигнал: уровень логического нуля (если двоичный номер данного входа совпадает с тем набором аргументов, при котором переключательная функция принимает нулевое значение) или уровень логической единицы (в противоположном случае). На рис. 2 показана схема включения мультиплексора в качестве УЛМ, реализующего ту переключательную функцию, которая задана табл. 1. Это пример перехода от табличной формы поведенческого описания к графической форме структурного описания объекта.

2.4.3. Наращивание размерности мультиплексоров

Размерность мультиплексора (она определяется количеством информационных входов), как и размерность некоторых других цифровых узлов, можно наращивать, то есть

можно построить мультиплексор достаточно большой размерности, используя несколько мультиплексоров меньшей размерности.

Рис. 3 является примером схемы наращивания — схемой построения мультиплексора 32:1 с применением мультиплексоров, размерность которых не превышает 8:1.

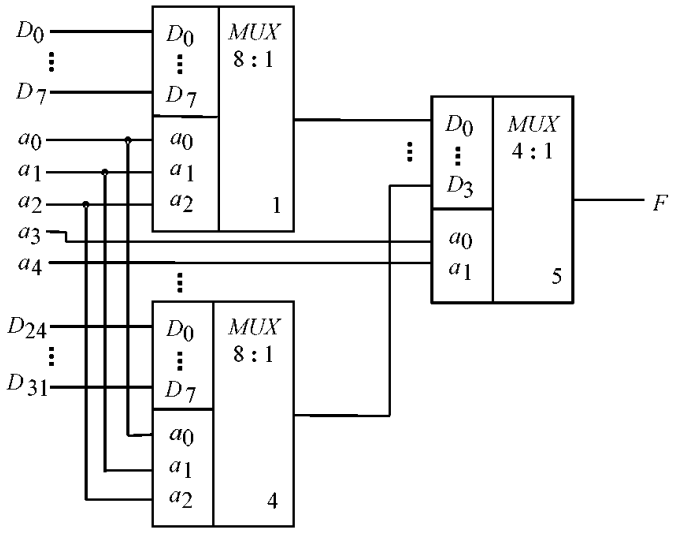


Рис. 3

2.4.4. Шинный мультиплексор

Шинный мультиплексор представляет собой цифровой узел, аналогичный мультиплексору, рассмотренному в 2.4.1. Отличия заключаются в следующем. Сигналами, подаваемыми на адресные входы, осуществляется выбор не одного информационного входа, а некоторой группы входов (шины). Шиной называют группу контактов или линий (проводов). Шинный мультиплексор имеет несколько входных информационных шин и одну выходную шину, а также уже упомянутые адресные входы. Двоичный код на адресных входах является порядковым номером выбираемой входной шины. Действующие на нее входные информационные сигналы передаются на выходную шину.

Заметим, что количество линий в шине называют ее разрядностью. Например, шина, образованная четырьмя линиями, называется 4-разрядной.

2.5. Логический синтез цифровых устройств

Логический синтез цифрового устройства — автоматизированный процесс, в котором его высокоуровневое описание на *HDL* языке (*VHDL, Verilog, SystemVerilog*) преобразуется в оптимизированный список логических вентилей (*netlist*). Этот список является схемотехнической реализацией проектируемого устройства на основе стандартных библиотек элементов, доступных в выбранной технологии, и их межсоединений.

Netlist генерируется специальными программами синтеза, которые интерпретируют *RTL* описание устройства и преобразуют его в структуру логических элементов в соответствии с ограничениями, установленными проектировщиком.

2.5.1. Синтезируемое подмножество языка *Verilog HDL*

Прежде чем приступить к описанию синтезируемого подмножества языка следует остановиться на общих принципах разработки цифровых схем на ПЛИС с использованием языков описания аппаратуры.

Для проведения синтеза необходимы три основных компонента:

- модели библиотечных элементов ПЛИС;
- синтезируемое описание на языке *HDL*;
- набор директив для средства синтеза.

Производитель микросхемы ПЛИС (*Altera*), предоставляет модели библиотечных элементов, которые могут быть использованы в схеме. Эти библиотеки направлены на использование в различных областях проектирования (моделирование, синтез, топология кристалла и т.д.) и представляются в различных форматах.

Один из форматов представляет из себя библиотеку элементов, описанных на языке описания аппаратуры и предназначенных для моделирования. В качестве примера можно привести файлы с библиотечными элементами из каталога: **\quartus\eda\sim_lib**

Таблица 2. — Библиотека моделей *ALTERA QUARTUS*

220model.v	Библиотека поведенческих (не синтезируемых) моделей параметризируемых мегафункций, таких как: lpm_dff, lpm_mux, lpm_counter и т.д.
altera_mf.v	Библиотека поведенческих (не синтезируемых) моделей автоматически генерируемых в ходе логического синтеза. lcell, global, altmult_accum и т.д.
flex10ke_atoms.v	Библиотека атомарных моделей компонентов, входящих в состав кристалла ALTERA FLEX10KE.
max_atoms.v	Библиотека атомарных моделей компонентов, входящих в состав кристалла ALTERA MAX.

В данной библиотеке содержатся элементы, описанные посредством несинтезируемых конструкций языка *Verilog HDL*. В описании библиотечных элементов (атомарных моделей) встречаются описания сквозных задержек распространения (*path delay*) и механизмы контроля временных параметров (*setup*, *hold time*). Сквозные задержки позволяют описать задержку модуля, не вдаваясь в его внутреннюю структуру. Системные функции контроля используются для проверки того, что изменения сигналов происходят в нужные моменты времени, например, для того чтобы последовательная логика не попадала в метастабильное состояние.

Не будем останавливаться подробно на этих элементах, потому что пользователю средств синтеза не нужно описывать библиотечные элементы, а в текстовом описании библиотеки обычно содержится информация о поведении элемента и его временной диаграмме. Можно сказать, что при разработке проекта СБИС или ПЛИС, возможно даже не придется вручную подключать/отключать эти элементы в структурном описании и вообще знать об их существовании.

Следующие два элемента разработки предоставляются пользователем — это синтезируемое описание на языке *Verilog* и набор директив для средства синтеза. Также, конечно, нужен набор средств разработки, включающий в себя, симулятор и средство синтеза. Следует остановиться на отличиях синтеза от компиляторов поведенческих языков. Компилятор языка (например *C*) детерминированным образом переводит операторы языка в команды машинного языка. Синтез переводит последовательные операторы языка *Verilog* в структурную схему, состоящую из библиотечных элементов. Данный процесс больше похож на перебор вариантов и выбор наилучшего, удовлетворяющего временным ограничениям и занимаемой площади. Следствием этого является то, что синтез это итеративный процесс, когда результаты предыдущего шага используются для коррекции директив для следующего шага. Методология написания директив синтеза и подход к синтезу схемы имеющей минимальное количество вентилей и максимальную тактовую частоту, является отдельным вопросом и здесь рассматриваться не будет.

Но в любом случае предоставляемое пользователем описание на языке *Verilog* должно быть правильным: оно должно быть работоспособным и синтезируемым.

Рассмотрим шаги, обычно требующиеся для разработки цифровой системы на ПЛИС:

1) подготовка синтезируемого (поведенческого/*RTL*) описания схемы (этому шагу может предшествовать моделирование на *SystemC*, *C*/*C++*, *Matlab*, или других специализированных средствах);
2) написание верификационной модели (тестовой оболочки или испытательного стенда (*testbench*)), в которой проводится полное тестирование всех режимов модели на

соответствие системным требованиям;

3) итеративная процедура синтеза поведенческой модели, результатом которой является структурная схема (список цепей или *netlist*) с использованием библиотечных элементов и файл задержек распространения (**sdf-**файла — *standard delay file*);
4) проверка работоспособности *netlist*, при этом обычно используется тот же *testbench*, что и для п. 2. Нарушение работоспособности может быть связано с нарушениями допустимых времен библиотечных элементов, гонками фронтов и пр. В зависимости от опыта разработчика происходит возврат к п.п. 1, 3 или переход к п. 5. Также полезно на этом шаге проверять результаты работы средств временного анализа (*static timing report*) для выяснения «узких мест» — критических путей проекта;
5) выполнение процедуры размещения и трассировки. Выполняется специальными средствами. Результатом является коррекция **sdf**-файла (помимо прошивки ПЛИС или разводки СБИС);
6) проверка *netlist* с новыми задержками. Эта проверка подобна п. 4, но возможен возврат к пунктам 1, 3, 5;
7) «*in-place*» оптимизация (для большинства современных ПЛИС отсутствует), производится выравнивание времен распространения сигналов — усиление/ослабление выходов ячеек, установка буферов/элементов задержки;
8) повтор пункта 6 до тех пор, пока не будут достигнуты требуемые характеристики.

Шаг 3 в этой схеме раньше выполнялся вручную, когда поведенческое описание заменялось структурным самим разработчиком. В настоящее время существуют средства синтеза, которые могут синтезировать более эффективную схему. Пользование этими средствами накладывает ограничение на использование конструкций языка в исходном коде. То есть требуется синтезируемая модель. Кроме того, чтобы многократно не переписывать поведенческую модель, следует четко представлять, какое *Verilog* описание приведет к синтезу того или иного элемента схемы.

2.5.2. Базовые правила автоматизированного синтеза

Любой элемент языка *Verilog* в результате автоматизированного синтеза может быть:

— синтезирован,

— проигнорирован,

— вызвать ошибку.

Конструкции языка могут поддерживаться полностью, частично или не поддерживаться, в зависимости от конкретного инструментального средства автоматизированного синтеза. Как правило, автоматизированные средства синтеза не

поддерживают:

— иерархические имена;

— конструкции ***initial***, ***fork-join*** или ***primitive****;*

— временной контроль ***#nn****;*

— событийный контроль поддерживается частично и только в блоках *always*.

Обычно каждое средство автоматизированного логического синтеза с языков описания аппаратуры имеет специальный документ, в котором указываются какие ограничения, вводятся на элементы языка. Перед написанием синтезируемого *RTL* кода на любом языка описания аппаратуры следует ознакомиться с этой документацией.

2.5.3. Синтез комбинаторной логики

Комбинационная логика (*Combinational logic*) или комбинационные цепи — семейство цифровых устройств, для которых выходные сигналы зависят только от текущих значений входных сигналов. Логическая функция комбинаторной логики может быть представлена как:

logic_outputs (t) = f (logic_inputs (t)).

Синтез комбинаторной логики может быть выполнен:

— на основе *Verilog* примитивов;

— на основе оператора присваивания **assign**;

— на основе *Verilog* функций (**function**);

— в процесс-блоках **always**;

— на основе *Verilog* процедур (**task**).

Ниже приведены примеры синтезируемых конструкций на основе:

1) непрерывного присвоения:

```
assign a = b + c & d;
wire b = {e,f} | g[1:0];
```

2) сигналов и операторов, описанных в теле функций (*function*)

```
function my_func;
      input depth;
      integer i,result;
      begin
      …
      end
endfunction
```

3) сигналов, формируемых в процесс-блоках **always**

```
      reg data_out;

 always @(a or b or c)

     if (b)
         data_out = a ;
      else
         data_out = c ;
```

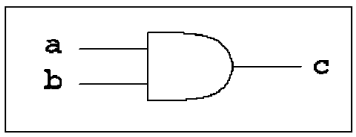
или

```
reg data_out;

always @(*)

      if (b)
            data_out = a ;
      else
            data_out = c ;
```

то есть блок ***always*** в списке чувствительности, которого перечислены все входные сигналы.

Как можно видеть в данном примере описание сигнала с ключевым словом ***reg*** не приводит к созданию регистра. Также комбинаторная логика синтезируется в случае ***case*** , если описаны все ветви выбора. При этом получается не приоритетный кодер, а набор параллельных конструкций. Также ***case*** может быть использован для синтеза мультиплексоров.

Логический элемент



1) с использованием выражения ***reg***

```
reg c;
always @ (a or b)
c = a & b;
```

2) с использованием выражения ***wire***

```
wire a,b,c;
assign c = a & b;
```

3) с использованием выражения встроенного примитива без объявления компонента

```
reg a,b;
wire c;
and (c,a,b); // output is always first in the list
```

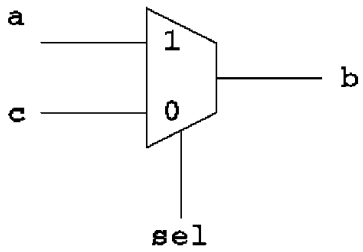
4) с использованием выражения встроенного примитива с объявлением имени компонента

```
reg a,b;
wire c;
and u1 (c,a,b); // output is always first in the list
```

4) с использованием выражения назначения

```
wire c;
reg a, b;
assign c = a & b;
```

Мультиплексор



1) с использованием процесса и оператора ***if-els***

```
always@(a or c or sel)
      if (sel == 1'b1)
            b = a;
      else
            b = c;
```

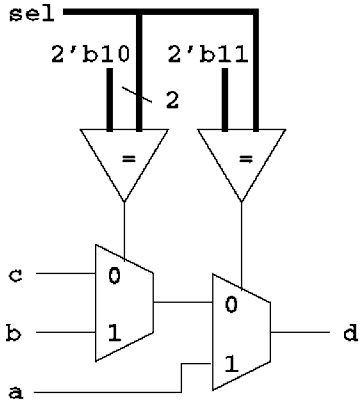
2) с использованием процесса и оператора ***case***

```
always @ (a or c or sel)
      case (sel)
            1'b1: b = a;
            1'b0: b = c;
      Endcase
```

3) с использованием тернарных выражений

```
wire b = sel ? a : c;
```

Приоритетный кодер



1) с использованием оператора ***case***

```
always @ (sel, a, b, c)
      case (sel)
            2'b11: d = a;
            2'b10: d = b;
            default:  d = c;
      endcase
```

2) с использованием выражения ***if-else***

```
always @ (sl, a, b, c)
      if (sel == 2'b11)
            d = a;
      else if (sel ==2'b10)
            d = b;
      else
            d = c;
```

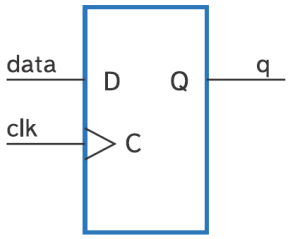
2.5.4. Синтез последовательной логики (*Sequential logic*)

К последовательной логике относится семейство цифровых устройств, у которых выходные сигналы зависят от текущих и предыдущих значений входных сигналов.

К последовательным устройствам относятся триггеры, регистры, счетчики, конечные автоматы (*FSM*), конвейеры (*pipeline*) и прочие устройства.

Пример синтезируемого описания D-триггера, работающего по переднему фронту тактового сигнала:

```
module dff (data, clk, q);
  input data, clk;
  output q;
  reg q;
  always @(posedge clk)
      q <= data;
endmodule
```



Ниже приведен пример синтезируемого поведенческого описания синхронного счетчика с входом разрешения счета и асинхронным сбросом с использованием процесс-блока **always:**

```
module counter (clk, reset, q);
     parameter WIDTH = 3;
     input clk, reset;
     output [WIDTH - 1 : 0] q;
     reg [WIDTH - 1 : 0] q;

       always @(posedge clk or negedge reset)
       if (!reset)
             q <= 3'b0;
       else
             q <= q + 1;
endmodule
```

Для конечных автоматов необходимо раздельное формирование реакции на предыдущие, текущие и будущие значения входных сигналов.

Более подробное описание синтезируемых конструкций цифровых устройств приводится в документации по применению программ синтеза цифровых устройств.

3. Краткое описание САПР *Quartus* II

3.1. Назначение САПР *Quartus II*

Эта система разработана американской фирмой *Altera* и предназначена для проектирования цифровых устройств широкого назначения на базе ПЛИС. ПЛИС представляет собой большую интегральную схему (БИС) с программируемой структурой. Это кристалл, содержащий множество логических ячеек, каждая из которых выполняет сравнительно простые логические преобразования. В некоторых семействах ПЛИС логические ячейки называют также макроячейками или логическими элементами. Соединения между ячейками внутри ПЛИС создает (программирует) пользователь с помощью программно-аппаратных средств. Благодаря высокому уровню интеграции даже одна запрограммированная ПЛИС может представлять собой весьма сложное цифровое устройство. Причем имеется возможность перепрограммирования ПЛИС.

Перечислим некоторые из средств рассматриваемой САПР.

3.2. Графический редактор системы *Quartus* II

Графический редактор, как и упоминаемый ниже текстовый редактор, является так называемым средством ввода описания проектируемого устройства. Причем данный редактор предназначен для ввода структурного описания проекта (см. 2.3) в графической форме.

Данный редактор позволяет создавать файлы проекта (с расширением **.bdf**) посредством составления схемы взаимосвязанных блоков (компонентов) при отображении ее на экране монитора. В свою очередь каждый компонент должен быть создан (программно организован) с помощью какого-либо редактора системы и должен иметь свой символ — условное графическое обозначение. В множестве таких компонентов можно выделить те, которые имеются в так называемых библиотеках системы, и те, которые создаются пользователем системы как проекты более низкого уровня иерархии.

В качестве компонентов, сосредоточенных в библиотеках системы, можно назвать примитивы, макро- и мегафункции. Эти компоненты тоже представляют собой проекты более низкого уровня иерархии, но создаются разработчиками САПР для облегчения процесса проектирования. Каждому из таких компонентов соответствует определенный (как правило, стандартный) символ. Примитивы — это простейшие программно организованные компоненты.

В библиотеке примитивов системы *Quartus II* имеются логические элементы, входные и выходные контакты, триггеры и др. Макрофункциями являются некоторые заранее спроектированные и включенные в отдельную библиотеку более сложные цифровые узлы. В

библиотеке макрофункций имеются мультиплексоры, дешифраторы, компараторы, счетчики, регистры и др. Заранее спроектированные и функционально завершенные компоненты более высокого уровня сложности, допускающие перестройку своих параметров проектировщиком-пользователем, называют мегафункциями. В библиотеке мегафункций изучаемой системы имеются, например, арифметические сумматоры (вычитатели), умножители, устройства деления чисел. Пользователь может также создавать свои собственные библиотеки компонентов.

3.3. Текстовый редактор системы *Quartus II*

Текстовый редактор является средством ввода структурного и (или) поведенческого описания проектируемого устройства с помощью языков описания аппаратуры и обеспечивает работу с файлами, имеющими следующие расширения: **.txt**, **.v**, **.vlg**, **.vqm**, **.vh**, **.verilog**, **.vhd**, **.vhdl**, **.edf**, **.edif**, **.edn**, **.tdf**, **.inc**, **.tcl**, **.c**, **.cpp**, **.h**, **.s**, **.asm**.

3.4. Сигнальный редактор и симулятор системы *Quartus II*

Сигнальный редактор в рассматриваемой системе позволяет, например, редактировать форму временных диаграмм входных воздействий для проектируемого устройства или его компонентов. После такого редактирования имеется возможность запустить симулятор системы для верификации (проверки) проекта, то есть анализа реакции проектируемого устройства или его компонентов на сигналы, заданные временными диаграммами.

При использовании сигнального редактора и симулятора создаются файлы тестирования (**.vwf**), в которых сохраняется информация о заданных временных диаграммах входных сигналов и соответствующих диаграммах выходных сигналов.

3.5. Символьный редактор системы *Quartus II*

С помощью символьного редактора можно просматривать, создавать и редактировать символ, представляющий собой условное графическое обозначение цифрового устройства, созданного в рамках какого-либо проекта.

Файл символьного редактора имеет расширение **.bsf**. Создать символ устройства можно, проделав следующий путь по меню **File → Create / Update → Create Symbol Files for Current File**. Создание символа доступно при работе с графическим и текстовым редакторами.

3.6. Редактор связей системы *Quartus II*

Редактор связей или, как его еще называют по-русски, поуровневый планировщик,

редактор трассировки позволяет просматривать и изменять размещение проектируемого устройства на кристалле ПЛИС.

3.7. Компилятор системы *Quartus II*

Компилятор является средством преобразования информации, которая содержится в файлах проекта, в информацию о соединениях между логическими ячейками внутри ПЛИС, а также средством оптимизации проектируемого устройства по критерию минимума требуемых ресурсов ПЛИС (в частности, ее ячеек) и критерию максимального быстродействия. Процесс компиляции можно запустить из меню **Processing**.

Файлы проекта — это файлы, которые содержат структурное и/или поведенческое описание проектируемого устройства и в связи с этим предназначены для обработки компилятором. Файлы проекта создаются при работе с перечисленными выше редакторами (кроме сигнального редактора, символьного редактора и редактора связей) и имеют указанные расширения. К таким файлам относят и файлы, созданные некоторыми другими САПР, совместимыми с системой *Quartus II*.

В результате компиляции и работы с некоторыми иными средствами (с сигнальным редактором, симулятором, символьным редактором, редактором связей) создаются так называемые вспомогательные файлы.

3.8. Программатор системы Quartus II

Программатор позволяет физически реализовать спроектированное устройство внутри кристалла ПЛИС, как говорят, программировать структуру ПЛИС. С помощью программатора файл конфигурации проекта с расширением **.pof**, созданный при компиляции, используется для организации соединений между ячейками ПЛИС. При этом ПЛИС должна быть связана с используемой универсальной ЭВМ специальным интерфейсом.

4. Задание на проектирование

4.1. Общая формулировка задания и его варианты

В качестве задания к данной лабораторной работе предлагается проектирование генератора сигнала псевдослучайной последовательности (ПСП), по функциональному назначению сходному с устройством, разработанном в лабораторной работе №2. ПСП представляет собой последовательность элементов d_i (i = 0, 1, 2, 3, … ∞ — номер элемента последовательности), повторяющихся с периодом N = 8, т. е. $d_{i+8} = d_i$.

Нулевой элемент последовательности $d_0 = 0$, а каждый последующий, начиная с d_1 по d_7, может принимать одно из двух значений: +1 или –1, в зависимости от варианта задания.

Чтобы записать указанный отрезок ПСП, необходимо выбрать первые четыре элемента последовательности (d_0, d_1, d_2, d_3) согласно номеру варианта задания (табл. 4.1), а последующие четыре элемента определить по следующим рекуррентным соотношениям:

$d_i = -d_{i-3} \times d_{i-2}$ — для вариантов 1…7;

$d_i = -d_{i-3} \times d_{i-1}$ — для вариантов 8…14.

Таблица 4.1 — Варианты индивидуальных заданий

№ варианта	d_0, d_1, d_2, d_3	№ варианта	d_0, d_1, d_2, d_3
1	0, –1, –1,	8	0, –1, –1,
2	0, –1, +1,	9	0, –1, +1,
3	0, +1, –1,	10	0, +1, +1,
4	0, –1, +1,	11	0, +1, +1, –
5	0, +1, +1,	12	0, +1, –1,
6	0, +1, +1,	13	0, –1, +1, –
7	0, +1, –1,	14	0, +1, –1, –

Следует отметить, что указанный отрезок представляет только первый период последовательности, а каждый последующий период ПСП должен быть его копией, при этом всю последовательность элементов достаточно ограничить четырьмя периодами.

Кроме того, каждый элемент последовательности (0, +1, –1) нужно сформировать проектируемым генератором в 4-разрядном дополнительном коде (элементу 0 соответствует код 0000, элементу +1 соответствует код 0001, а элементу –1 соответствует код 1111).

Требуется разработать указанный генератор ПСП в виде модуля на языке *Verilog* с использованием поведенческого описания компонентов схемы, провести его моделирование и синтез схемы в САПР *Quartus II*.

4.2. Поведенческое описание проектируемого устройства и его компонентов в текстовой форме

Здесь и далее в 4.3 излагаются сведения, необходимые для составления схемы проектируемого устройства (перехода к графической форме его структурного описания в связи с последующим использованием графического редактора САПР *Quartus II* в качестве средства ввода описания), а также для составления контрольного примера (см. 5.3).

Генератор необходимо построить на основе мультиплексора **MUX1** размерности 8:1 (количество информационных входов мультиплексора должно быть не меньше длины генерируемого отрезка ПСП, то есть восьми). Адресные входы мультиплексора

предназначены для подачи на них трехразрядных двоичных порядковых номеров (от 001 до 111 в параллельном коде, что соответствует десятичным значениям номеров от 1 до 7) генерируемых элементов ПСП. Сигналы на информационных входах мультиплексора следует задать по следующему принципу: для формирования элемента +1 на соответствующий его порядковому номеру информационный вход подается логическая единица, для формирования элемента –1 на соответствующий его порядковому номеру информационный вход подается логический нуль. В результате на выходе мультиплексора образуется последовательность логических единиц и нулей, которая в дальнейшем должна быть преобразована в последовательность дополнительных кодов элементов ПСП (чисел +1 и –1 соответственно).

Для формирования порядковых номеров генерируемых элементов ПСП на адресных входах генератора элементов отрезка можно использовать поведенческое описание счетчика **counter**, приведенное в п. 2.5.4. У этого модуля (рис. 4) должен быть модуль счета N = 8. Это позволит получить 3-битный счетчик тактовых импульсов (синхроимпульсов), поступающих на счетный вход **clk**, который изменяет своё состояние по передним фронтам этих импульсов при условии высокого уровня сигнала на входе асинхронного сброса **reset**.

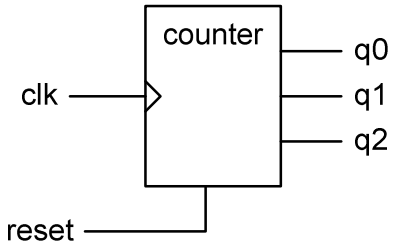


Рис. 4 — Модуль синхронного счетчика

Средства генерации сигналов **clk** и **reset** проектировать не требуется: эти сигналы должны задаваться с помощью сигнального редактора при верификации проекта.

Для преобразования последовательности логических единиц и нулей в дополнительные двоичные коды чисел +1 и –1 сигнал с выхода мультиплексора необходимо подать на формирователь. При действии на вход формирователя логической единицы должен появиться параллельный дополнительный код 0001 числа +1 на четырех выходах формирователя, а при действии логического нуля — дополнительный код 1111 числа –1. В качестве такого формирователя можно составить синтезируемое поведенческое *Verilog* описание шинного мультиплексора **MUX2** размерности 2:1 или выбрать (макрофункцию **74157** из библиотеки системы *Quartus II*).

На рис. 5 контакты **A1**, **A2**, **A3**, **A4** и **B1**, **B2**, **B3**, **B4** образуют две входные

информационные шины мультиплексора. **Y1**, **Y2**, **Y3**, **Y4** — его выходная шина. Поскольку входных шин две, то для выбора одной из них достаточно одного адресного входа. Таким адресным входом является контакт **SEL**: при логическом нуле на этом входе выбирается первая из входных шин (**A1** , **A2** , **A3**, **A4**), а при логической единице — вторая (**B1**, **B2** , **B3**, **B4**). Контакт **GN** является разрешающим входом: при **GN** = 0 мультиплексор функционирует согласно 2.4.4, а при **GN** = 1 на выходной шине нули.

Адресный вход мультиплексора (рис. 5) нужно использовать как вход упомянутого выше формирователя: именно на него подается сигнал с выхода мультиплексора **MUX1**. Если на этот вход поступает логический нуль (соответствует элементу –1 ПСП), то для появления на выходной 4-разрядной шине параллельного кода 1111 нужно на все линии входной шины **A1**, **A2**, **A3**, **A4** подать логическую единицу. Если же на упомянутый вход поступает логическая единица (соответствует элементу +1), то для его преобразования в комбинацию 0001 такая комбинация подается на шину **B1**, **B2**, **B3**, **B4**.

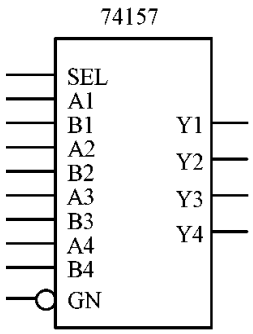


Рис. 5

Выходная шина формирователя является выходной шиной генератора.

В результате на выходной шине генератора (после подачи на счетчик **counter** сигнала **reset** высокого уровня) синхронно с тактовыми импульсами должны появляться нужные значения всех семи элементов генерируемого отрезка ПСП.

Поскольку в течение нулевого элемента последовательности необходимо на выходе ПСП генерировать параллельный код 0000, то три выхода блока **counter**, на которых в это время формируется код 000, необходимо соединить с тремя входами элемента **3НЕ-И** (**NAND3** из библиотеки примитивов *Quartus II*), подключая выход последнего ко входу **GN** шинного мультиплексора.

Таким образом, проектируемый генератор будет состоять из модуля **counter** и модуля комбинационного преобразователя кодов **encoder**, образованного мультиплексорами **MUX1**, **MUX2** и элементом **3НЕ-И**.

5. Порядок выполнения задания и дополнительные сведения о системе *Quartus II*

5.1. Начало работы над проектом

Работа в системе **Quartus II** начинается с действий, которые называют созданием проекта. Прежде всего, необходимо создать новую папку для хранения файлов этого проекта (папка должна быть индивидуальной для каждого рабочего места в лаборатории и может иметь в своем названии только цифры и латинские буквы). Затем после запуска *Quartus II* при помощи команды **New Project Wizard…** меню **File** открывается окно мастера создания проекта. В появившемся окне следует нажать **Next**. В результате открывается страница 1 мастера, имеющая название **New Project Wizard: Directory, Name and Top-Level Entity**. В верхнем поле открывшейся страницы следует ввести имя и путь к папке проекта (можно воспользоваться кнопкой с изображением многоточия справа от поля и выбрать папку с помощью специального диалогового окна), имя проекта и имя устройства, занимающего верхнюю позицию в иерархии проекта (их рекомендуется задать как **generator1**). После этого можно завершить создание проекта нажатием кнопки **Finish**. Если процесс проектирования окажется вынужденно прерванным (с сохранением полученных результатов), то для возобновления работы над проектом его необходимо открыть при помощи команды **Open Project…** меню **File**, после чего потребуется выбрать папку с файлами проекта и указать его имя.

5.2. Работа с графическим редактором

Первым этапом данной лабораторной работы является создание (ввод) структурной схемы проектируемого генератора в графическом редакторе. Для создания файла, который будет содержать структурную схему устройства (после создания проекта согласно 5.1), следует выполнить команду **New…** меню **File**. В появившемся диалоговом окне на вкладке **Device Design File** следует выбрать тип файла **Block Diagram / Schematic File** и нажать **OK**. Эти действия приведут к открытию окна графического редактора с загруженным в него файлом с расширением **.bdf**, который теперь необходимо включить в проект. Делается это следующим образом**: File → Save As…** В появившемся диалоговом окне сохранения файла следует задать его имя и установить флажок **Add file to current project**. В графическом редакторе схема создается из отдельных компонентов, которыми являются примитивы, макрофункции и мегафункции фирмы **Altera**, а также другие заранее подготовленные компоненты (в нашем случае к последним относится модуль **counter** и **encoder**). Вводятся компоненты следующим образом: на свободном участке рабочего поля нажать правую кнопку мыши и в появившемся контекстном меню выбрать пункт **Insert → Symbol…**, или выполнить двойное нажатие левой кнопки мыши на том участке рабочего поля, куда

необходимо ввести компонент. Далее в диалоговом окне нужно выбрать требуемый каталог (каталог **primitives** для примитивов; каталог **megafunctions** для мегафункций; каталог **Project** для компонентов, предварительно созданных проектировщиком; каталог **others**, где в подкаталоге **others/maxplus2** содержатся макрофункции мультиплексоров), а затем в каталоге выбирается нужный компонент. Заметим, что каталог **Project** при работе в системе *Quartus II* является обозначением папки, созданной ранее для хранения файлов проекта и имеющей произвольное название из цифр и латинских букв.

В данной лабораторной работе используются следующие компоненты. Примитивы: **NAND3** — логический элемент 3И-НЕ; **INPUT** — входной контакт; **OUTPUT** — выходной контакт; **GND** — уровень логического нуля; **VCC** — уровень логической единицы. Макрофункции: **74157** — шинный мультиплексор (рис. 5). Эти макрофункции описаны выше. Специально подготовленные для данной лабораторной работы модули **counter** и **encoder** необходимо сохранить в каталоге **…\Alt_Lib**. Из этого каталога в папку, созданную для хранения файлов проекта, необходимо скопировать файл графического редактора **counter.bdf**, **encoder.bdf**, содержащий информацию об описании компонента **encoder**, а также файлы **counter.bsf**, **encoder.bsf**, символов этих компонентов. Тогда папка с файлами проекта будет выполнять также функции каталога пользователя.

После импортирования какого-либо компонента из соответствующего каталога в рабочее поле графического редактора данный компонент (как и любой другой объект схемы, например, линию) можно перемещать по рабочему полю: щелчок левой кнопкой мыши на выбранном компоненте (выделение компонента) с последующим перемещением компонента и курсора мыши при нажатой ее левой кнопке. Соединение компонентов в проектируемой схеме можно осуществить следующим образом: переместить курсор мыши в одну из тех двух точек схемы, которые нужно соединить между собой, нажать левую кнопку мыши и, не отпуская ее, перемещать курсор ко второй из соединяемых точек. Для получения соединительной линии с несколькими изломами потребуется несколько подобных манипуляций с мышью. Укажем еще один возможный способ соединения точек схемы: несколько линий схемы (в том числе «оборванных»), которым присвоено одно и то же имя, считаются соединенными между собой. Присвоение имени какой-либо линии осуществляется следующим образом: выделить линию щелчком мыши, набрать название линии с клавиатуры (только латинскими буквами). Для последующего редактирования названия потребуется выделить его без выделения линии. Аналогичны способы соединений шинами.

В процессе реализации в графическом редакторе ранее составленной схемы проектируемого генератора (схема должна быть составлена до прихода в лабораторию)

постоянный уровень логической единицы в тех точках схемы, где он необходим, обеспечивается соединением этих точек с примитивом **VCC**. Аналогично, использование примитива **GND** обеспечивает уровень логического нуля.

До завершения работы в графическом редакторе нужно отметить входы и выходы проектируемого устройства соответственно примитивами **INPUT** и **OUTPUT**. В противном случае окажется невозможным назначение контактов реальной ПЛИС входам и выходам устройства при компиляции, упоминаемой ниже. Заметим, что возможно использование этих примитивов на входах и выходах компонента (части) устройства. В последнем случае отмеченным входам и выходам не обязательно соответствуют контакты реальной ПЛИС, а примитивы **INPUT** и **OUTPUT** (после присвоения им соответствующих названий) используются для указания связей между компонентами. После импортирования данных примитивов из библиотеки и их размещения в схеме необходимо отредактировать названия соответствующих входов и выходов. Для этого нужно нажать левую кнопку мыши дважды на названии входа или выхода, а затем изменить его (последующее изменение названия тоже возможно).

Проектируемый генератор должен иметь два входа: с названиями **reset** (для асинхронного сброса и запуска) и **clk** (для подачи тактовых импульсов). Они являются входами блока **counter**. Выходами генератора будут четыре выхода формирователя, то есть шинного мультиплексора (см. 4.2).

Кроме того, в поле **VCC** (**GND**) каждого примитива **INPUT** необходимо записать один из двух возможных вариантов уровня входного сигнала по умолчанию: **VCC** (если требуется обеспечить действие на данном входе логической единицы в отсутствие иного входного сигнала) или **GND** (если требуется обеспечить действие на данном входе логического нуля в отсутствие иного входного сигнала).

Для обозначения четырех выходов генератора целесообразно использовать графический символ шины — жирную линию. Делается это следующим образом. К каждому из четырех выходов подводится линия, «оборванная» на противоположном конце, которая затем именуется (например, выходу младшего разряда присваивается имя **OUT0,** следующий разряд именуется как **OUT1** и т. д.). Далее в любом месте рабочего поля проводится жирная линия и ей присваивается имя **OUT[3..0]**. К шине необходимо подсоединить примитив типа **OUTPUT** и назвать его тем же именем, что и шину. Жирную линию можно создать следующим образом: вначале проводится обычная линия (для построения линий целесообразно воспользоваться вертикальной панелью инструментов, расположенной в левой части экрана), затем она выделяется и на изображении выделенной линии нажимается правая кнопка мыши, в появившемся контекстном меню выбирается пункт **Bus Line**.

5.3. Компиляция проекта

После того как схема устройства будет введена средствами графического редактора необходимо осуществить компиляцию проекта.

До осуществления компиляции необходимо также выбрать семейство ПЛИС для реализации спроектированного устройства**: Assignments → Device…**, выбор требуемого семейства (рекомендуется **FLEX10K**) в поле **Family**.

Если перед запуском компилятора не оказался открытым нужный проект, то его необходимо открыть согласно 5.1. Далее запускается процесс компиляции**: Processing → Start Compilation**. Причем компилироваться будет не один упомянутый открытый файл, а вся совокупность файлов данного проекта. В результате компиляции, как уже говорилось, создается ряд вспомогательных файлов, которые не относят к файлам проекта. Запуск компиляции для данной лабораторной работы необходим лишь в связи с тем, что впоследствии потребуется осуществлять верификацию проекта (поскольку она невозможна без виртуального размещения устройства на кристалле ПЛИС).

В процессе компиляции сведения об ошибках в проекте выводятся на экран монитора. После двойного щелчка мышью на строке сообщения об ошибке открывается файл, содержащий ошибку и в нем выделяется то место, которое компилятор зафиксировал как ошибочное. Аналогично выдаются сведения об ошибках при верификации проекта с помощью сигнального редактора. В частности, одной из ошибок могло бы быть отсутствие файла **counter.bdf** в папке с файлами проекта.

5.4. Верификация проекта

Следующим этапом после компиляции является верификация проекта, то есть моделирование и проверка правильности функционирования спроектированного устройства. С этой целью создается файл временных диаграмм. Для его создания (после открытия проекта согласно 5.1) необходимо двигаться по меню **File → New → Other Files → Vector Waveform File**. При этом запустится сигнальный редактор с загруженным файлом с расширением **.vwf**. Созданный файл необходимо включить в проект тем способом, которым включался в проект созданный ранее файл графического редактора.

Проверка правильности работы устройства осуществляется следующим образом. Вначале составляется так называемый контрольный пример. Он представляет собой выбранный набор временных диаграмм входных сигналов и временные диаграммы требуемой (например, рассчитанной вручную по таблице истинности, заданной для проектирования) реакции спроектированного устройства на эти входные сигналы. На следующем этапе проверки временные диаграммы выбранных входных сигналов задаются в

сигнальном редакторе. И, наконец, последним этапом проверки является запуск моделирования (симуляции) работы устройства при заданных входных сигналах, сравнение результатов моделирования с контрольным примером и корректировка введенного описания (например, схемы) устройства в случае недопустимого различия результатов моделирования и ожидаемых (в соответствии с контрольным примером) результатов.

Контрольный пример для данной лабораторной работы должен быть создан при самостоятельной подготовке (после изучения принципов построения проектируемого устройства).

Как уже отмечалось, входными сигналами проектируемого генератора являются: последовательность тактовых импульсов **clk** и сигнал **reset**. Их временные диаграммы задаются следующим образом. В поле **Name** рабочего окна сигнального редактора нужно нажать правую кнопку мыши и в контекстном меню выбрать пункт **Insert Node or Bus…**

В появившемся диалоговом окне нажимается кнопка **Node Finder…** В следующем диалоговом окне нажимаются последовательно 3 кнопки**: List → OK** (если после нажатия кнопки **List** в поле **Nodes Found** отсутствуют строки с именами контактов устройства, то следует в поле **Filter** установить значение **Pins: all** и нажать кнопку **List** повторно). В результате информация о входах и выходах устройства, полученная в процессе компиляции, переносится в файл с расширением **.vwf** и отображается в рабочем окне сигнального редактора. В частности, в поле **Name** появляются названия всех входов и выходов устройства. Чтобы задать временную диаграмму сигнала на некотором входе, необходимо выделить этот вход (нажатием левой кнопки мыши на поле **Value**). Затем с помощью панели инструментов, расположенной в левой части окна редактора, задается требуемая (в соответствии с контрольным примером) форма сигнала.

Длительность одного периода импульсов (меандра) **clk** не рекомендуется выбирать менее 40 нс (чрезмерно малое значение данного параметра не допускается в связи с ограниченным быстродействием реальной ПЛИС). Рекомендуемые параметры сигнала **reset** указаны в 4.2.

Панель инструментов предоставляет, в частности, следующие возможности. Щелчок мышью на пиктограмме с изображением **0** позволяет задать уровень логического нуля на предварительно выделенном участке временной оси. Аналогично задается уровень логической единицы (пиктограмма **1**), неопределенное состояние (**X**) и третье состояние (**Z**). Под неопределенным состоянием понимается либо любое значение входного или выходного сигнала, либо неизвестное значение (то, которое невозможно определить при моделировании) выходного сигнала. Щелчок мышью на пиктограмме **INV** приводит к инверсии сигнала на выделенном временном интервале. Для формирования меандра

используется пиктограмма с изображением часов. Пиктограмме **C** соответствует возможность формирования на входах или выходах последовательности чисел, образующих арифметическую прогрессию с заданной разностью (с заданным параметром **Increment by**). Причем абсолютная величина этого параметра может превышать единицу только в том случае, когда упомянутая последовательность формируется на шине, образованной несколькими линиями-проводами. С использованием пиктограммы **?** можно задавать для выделенного временного интервала значения сигналов на какой-либо шине. При этом совокупность их значений задается числом, записанным в той системе счисления, которая указывается в поле **Radix** окна установки значения.

При работе с сигнальным редактором группировка линий (входов или выходов устройства) в шину осуществляется следующим образом. Прежде всего в поле **Name** выделяются линии (названия), подлежащие группировке. Затем нужно щелкнуть правой кнопкой мыши на выделенной области и в появившемся меню выбрать пункт **Group...** Далее указывается система счисления для представления данных на шине**:** двоичная (**Binary**), восьмеричная (**Octal**), десятичная при числах со знаком (**Signed Decimal**) и при числах без знака (**Unsigned Decimal**) или шестнадцатеричная (**Hexadecimal**).

Обратная операция (разгруппировка шины) реализуется выделением требуемой шины в поле **Name** с последующим нажатием правой кнопки мыши и выбором в меню пункта **Ungroup**.

После того, как будут заданы все необходимые сигналы, можно запустить симуляцию (моделирование) функционирования устройства. Для этого в меню **Processing** выбирается пункт **Start Simulation**. Временные диаграммы, полученные в результате симуляции, сохраняются в файл, имя которого имеет следующий вид **имя_vfw_файла-sim.vwf**, где **имя_vwf_файла** является именем файла временных диаграмм на входе устройства и, как правило, совпадает с именем проекта. Данный файл находится в подкаталоге **db** каталога проекта.

Сигнальный редактор дает возможность оценить быстродействие спроектированного устройства по задержкам его выходных сигналов относительно входных и по длительности переходных процессов, наблюдаемых на временных диаграммах. Для измерения длительности соответствующего временного интервала необходимо переместить мышью маркер — вертикальную синюю линию (после щелчка на верхней части маркера) к началу измеряемого интервала. После этого курсор мыши размещается в конце интервала, а в поле **Interval** верхней части экрана наблюдается значение длительности интервала.

При выполнении лабораторной работы требуется измерить задержку появления сигнала относительно соответствующего фронта импульсов **clk**.

6. Требования к отчету по работе

В качестве отчета по данной работе представляются в электронном виде и сопровождаются устными пояснениями следующие результаты: схема генератора, выполненная в графическом редакторе; ***Verilog***-описание схемы; временные диаграммы, полученные при верификации проекта; результаты измерения временной задержки.

7. Вопросы для самопроверки

1. В соответствии с какими соображениями выбирается размерность мультиплексора для выполнения задания к лабораторной работе? Сколько адресных входов должен иметь такой мультиплексор?
2. Каким образом общие принципы наращивания используются для проектирования мультиплексора?
3. Чем определяется требуемая разрядность (количество выходов) счетчика?
4. Какие сигналы и почему должны поступать на входные информационные шины шинного мультиплексора-формирователя в заданном проекте?
5. Каким образом обеспечивается формирование кода нулевого значения проектируемым генератором вне временного интервала генерации сигнала?
6. Возможна ли верификация проекта без предварительной его компиляции?
7. Какие ограничения накладываются на параметры входных сигналов генератора при верификации проекта и чем это обусловлено?

Библиографический список

1. Фролкин В.Т., Попов Л.Н. Импульсные и цифровые устройства: Учеб. пособие для вузов.— М.: Радио и связь, 1992.— 336 с.
2. Угрюмов Е.П. Цифровая схемотехника.— СПб.: БХВ, 2001.— 528 с.
3. Стешенко В.Б. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов.— М.: ДОДЭКА, 2000.— 128 с.