Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

А.А. Бомбизов

ПОВЕДЕНЧЕСКОЕ ОПИСАНИЕ

Методические указания к выполнению
лабораторной и самостоятельной работы
по дисциплине «Проектирование систем на кристалле»

Томск
2020

УДК 681.3 (075.32)
ББК 32.973стд1-02
Б 803

Рецензент:
Тренкаль Е.И., доцент кафедры конструирования узлов и деталей радиоэлектронной аппаратуры ТУСУР, канд. техн. наук

Бомбизов, Александр Александрович
Б 803 Поведенческое описание: методические указания к выполнению лабораторной и самостоятельной работы по дисциплине «Проектирование систем на кристалле» / А.А. Бомбизов. – Томск. гос. ун-т систем упр. и радиоэлектроники, 2020. – 20 с.

Настоящее методическое указание по выполнению лабораторной и самостоятельной работы по дисциплине «Проектирование систем на кристалле».

Методическое пособие содержит краткое описание основ поведенческого программирования на примере реализации таких компонентов, как счетчик импульсов, делитель частоты, дешифратор и мультиплексор.

Одобрено на заседании каф. КУДР, протокол № 234 от 5 марта 2022 г.

УДК 681.3 (075.32)
ББК 32.973стд1-02

1 Введение

В предыдущей работе были разобраны этапы создания проекта в среде Vivado 2019 для FPGA, а так же основные моменты разработки модуля верхнего уровня с использованием инструментов структурного программирования. Но это лишь мизерная доля возможностей, которыми обладает связка HDL и ПЛИС.

Целью настоящей работы является освоение основ поведенческого программирования на примере реализации таких компонентов, как счетчик импульсов, делитель частоты, дешифратор и мультиплексор. И получение базовых навыков работы со встроенным симулятором.

2 Краткая теория

Для начала необходимо рассмотреть описание и принцип работы тех компонентов, которые будут использоваться в лабораторной работе.

В цифровой схемотехнике счетчик импульсов – электронное устройство, предназначенное для подсчета числа импульсов, поданных на вход. Количество поступивших импульсов выражается в двоичной системе счисления. Счетчики импульсов являются некоторой разновидностью регистров (счетные регистры) и строятся на триггерах и логических элементах. Простейшая реализация счетчика на D-триггерах изображена на рисунке 1.

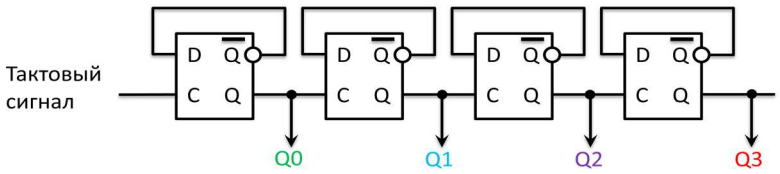


Рисунок 1 – Счетчик на D-триггерах

Основными показателями счетчиков являются коэффициент счета $K_c = 2^n$ – число импульсов, которое может быть сосчитано счетчиком. Например, счетчик, состоящий из четырех триггеров, может иметь максимальный коэффициент счёта $2^4 = 16$. Таким образом, степень определяет количество триггеров, которое требуется для хранения числа выбранной разрядности. Для четырехтриггерного счетчика минимальный выходной код – 0000, максимальный – 1111 (то есть от 0 до 15), а при коэффициенте счёта K_c = 10 выходной счет останавливается при коде 1001 = 9 и на следующем импульсе сбрасывается в 0.

Работу схемы иллюстрируют временные диаграммы, приведенные на рисунке 2. При поступлении первого импульса первый триггер переходит в

состояние $Q0 = 1$, то есть в счетчике будет записан цифровой код 0001. При поступлении второго счетного импульса первый триггер переключится в состояние логического нуля, а второй в единицу, то есть в счетчике будет записано число 2 с кодом 0010 и т.д.

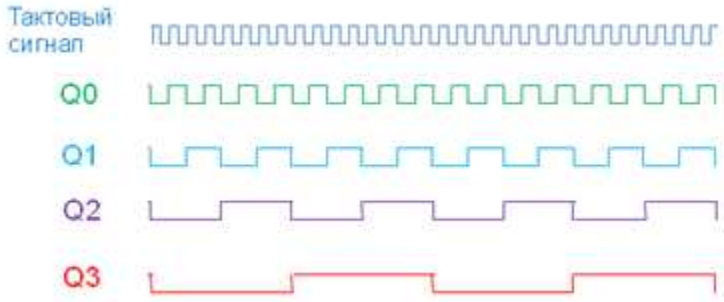


Рисунок 2 – Временная диаграмма счетчика на D-триггерах

В процессе работы двоичного счетчика частота следования импульсов на входе каждого последующего триггера уменьшается вдвое по сравнению с частотой его входных импульсов, поэтому счетчики могут применяться в качестве делителя частоты.

При поступлении шестнадцатого импульса все триггеры обнулятся. Для принудительного обнуления счетчика может быть предусмотрен вход сброса. Условно-графическое изображения счетчика импульсов на схеме представлено на рисунке 3.

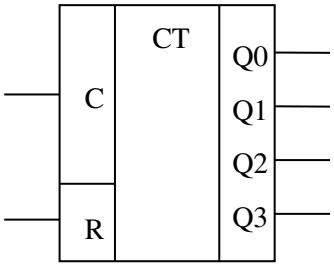


Рисунок 3 – Обозначение счетчика импульсов с фиксированным коэффициентом счета $K_c = 16$

На рисунке 4 изображена структурная схема счетчика импульсов с регулируемым коэффициентом счета. Здесь результат подсчета импульсов счетчиком непрерывно сравнивается с заданным коэффициентом счета. Если значения совпадут, то с минимальной задержкой будет выполнен сброс счетчика и, соответственно, установка значений $Q_{0..N}$ в начальное состояние. При этом схема может быть выполнена на логике исключающего ИЛИ и И-НЕ.

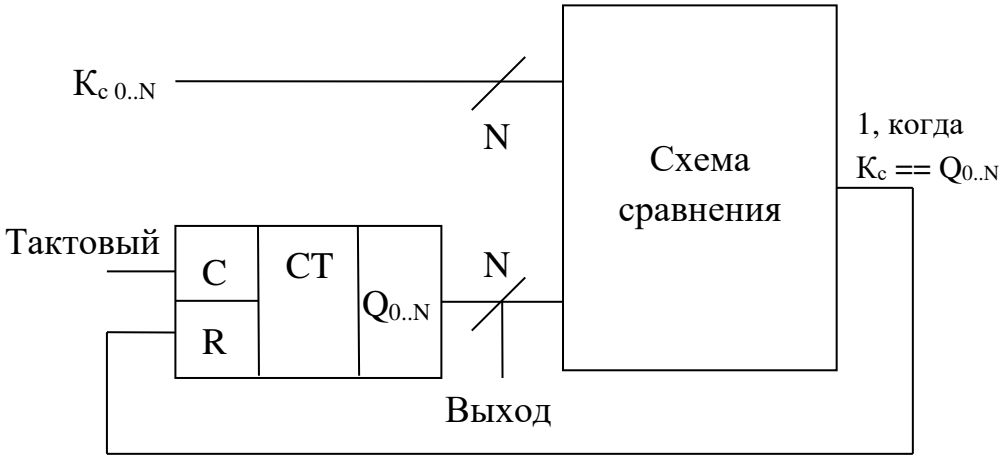


Рисунок 4 – Обобщенная структурная схема счетчика импульсов с регулируемым коэффициентом счета

Делитель частоты представляет собой соединение счетчика импульсов с регулируемым коэффициентом счета и однобитного счетчика как показано на рисунке 5. То есть при каждом сбросе счетчика делитель будет менять свое значение с единицы на ноль и обратно. Таким образом, коэффициент деления составит $K_c/2$. Либо счетчик в делителе может быть реализован для срабатывания как по фронту тактового импульса, так и по спаду, в этом случае коэффициент деления будет равен K_c.

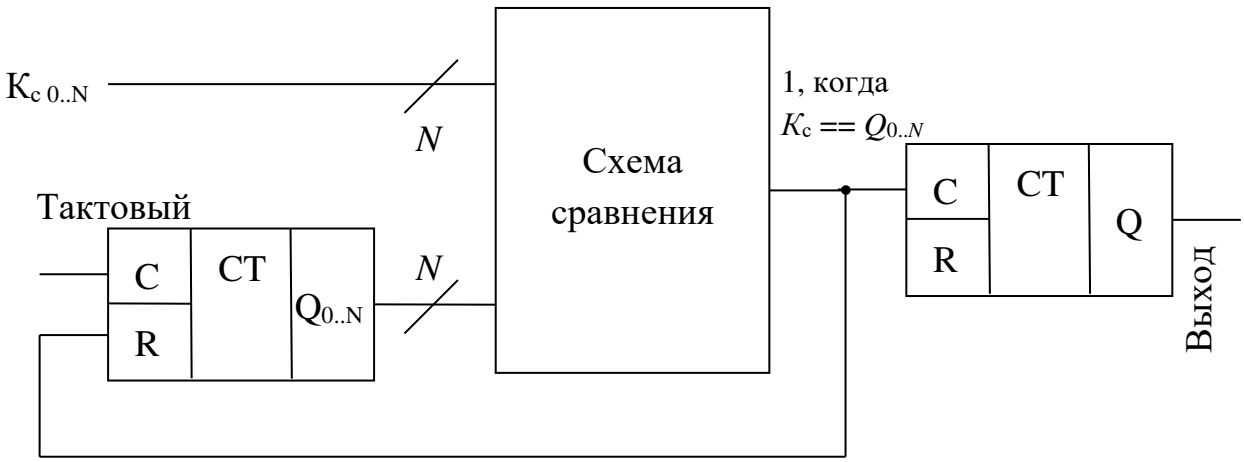
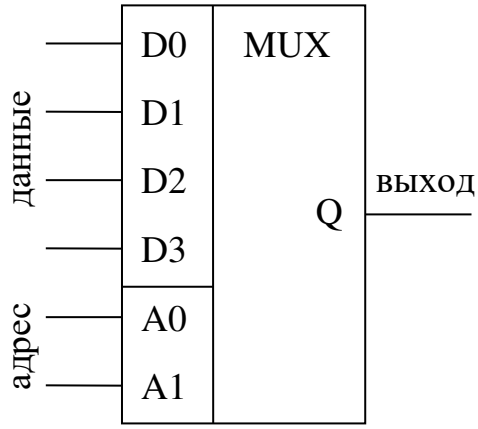


Рисунок 5 – Обобщенная структурная схема делителя частоты с регулируемым коэффициентом счета

Мультиплексор – устройство, в котором выход соединяется с одним из входов, в соответствии с кодом адреса. Т мультиплексор представляет собой электронный переключатель или коммутатор.

Структурная схема и таблица состояний мультиплексора изображена на рисунке 6.



A1	A0	Q
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Рисунок 6 – Условно-графическое обозначение и таблица состояний мультиплексора

Шифратор (называемый также кодером) осуществляет преобразование сигнала в цифровой код, чаще всего десятичных чисел в двоичную систему счисления.

Подача сигнала на один из входов приводит к появлению на выходах n-разрядного двоичного числа, соответствующего номеру входа. Например, при подаче импульса на 4-й вход, на выходах возникает цифровой код 100 (рисунок 7, а).

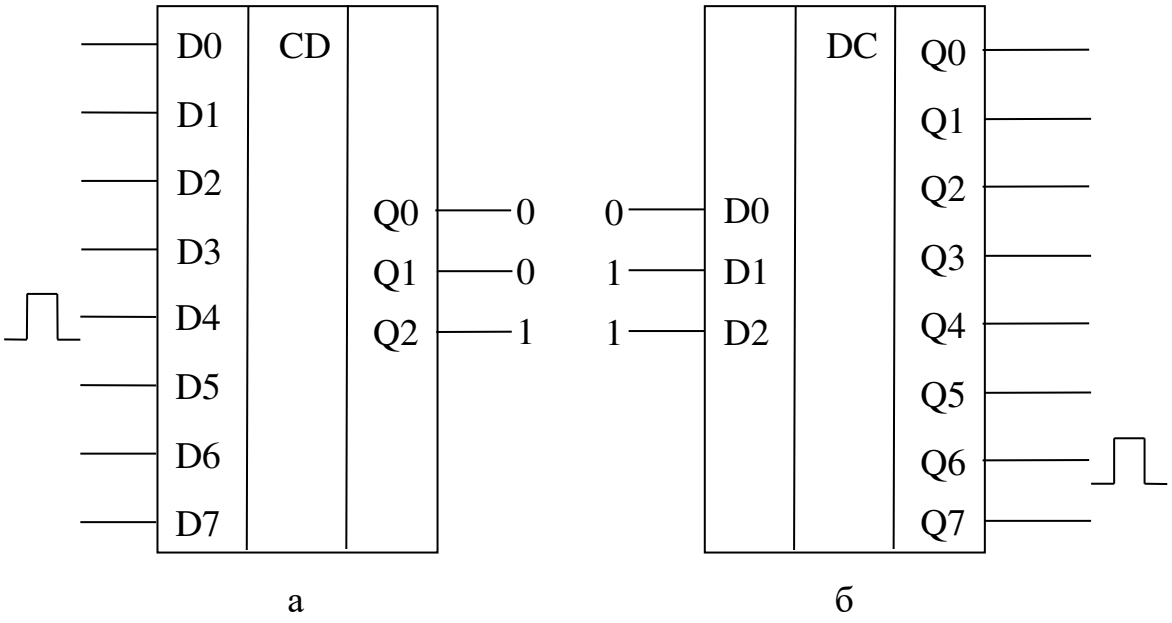


Рисунок 7 – Условно-графическое обозначение шифратора (а) и дешифратора (б)

Разработка и моделирование проекта с использованием логических вентилей, модулей и примитивов тесно связаны с логической структурой,

но эти конструкции не обеспечивают необходимый уровень абстракций для описания комплекса высокоуровневых аспектов системы. Для решения таких задач используется поведенческое описание, позволяющее в полной мере использовать возможности языка, то есть переходу от описания логической структуры к описанию поведения объекта. Поведенческое описание широко используется в таких объектах как блоки инициализации, процесс-блоки, задачи и функции. Характеризуется главным признаком – последовательным выполнением операторов. Такое описание должно быть локализовано между парой ключевых слов *begin* – *end* (если операторов более одного). Всё что находится вне этой пары относится к структурному описанию. Основные операторы такого описания в целом очень похожи на выражения обычных языков программирования: ветвление, выбор, циклы.

Блоки инициализации необходимы для установки начальных состояний и вводятся ключевым словом *initial*. Они выполняются в самом начале моделирования, в момент времени 0. Эти блоки, как правило, не поддерживаются системами синтеза, но широко используются в процессе симуляции работы модулей. Формальный синтаксис такого блока выглядит так:

```
// для одного оператора
initial<оператор>;
// для группы операторов
initial
begin
<группа операторов>;
end
```

При логическом моделировании часто приходится реализовывать устройства с учетом предыстории их функционирования, то есть предполагается наличие памяти, которая в комбинационной логике не предусмотрена. Для устройств с памятью особая роль отводится фактору времени, который в синхронных схемах естественным образом учитывается тактами. Такты определяют моменты смены состояний и синхронизируют соответствующую функцию. Простейшим примером синхронной схемы является D-триггер. Для асинхронной – встроенные примитивы и их комбинации, рассмотренные в прошлой лабораторной работе.

Процесс-блок предназначен для реализации синхронного процесса, который может повторяться многократно. Блок выполняет процесс всегда, когда некоторая переменная или группа переменных изменяют своё значение или происходит какое-либо событие.

```
// для одного оператора
always @(<список чувствительности>)
<оператор>;
// для группы операторов
always @(<список чувствительности>)
begin
<группа операторов>;
end
```

Опциональное поле *<список чувствительности>* содержит список событий, при возникновении которых блок начинает выполняться. Формальный синтаксис списка чувствительности следующий:

@(<событие 1> or <событие 2> or … <событие N>)

@(<событие 1>, <событие 2>, … <событие N>)

<событие N> – представляет собой переменную (неявный тип *event*). Для описания событий могут использоваться два вспомогательных ключевых слова *posedge* и *negedge*, чтобы указать с каким фронтом сигнала (переменной) данное событие связано: с возрастающим (из 0 в 1 – *posedge*) или спадающим (из 1 в 0 – *negedge*). Возрастающий фронт может определяться и как переход переменной из неопределенного состояния в состояние логической единицы, а спадающий – как переход из лог. 1 в неопределенное состояние. Если эти ключевые слова не указать, то событием станет любое изменение переменной. Ключевые слова *posedge* и *negedge* всегда применяются при создании регистровой логики для фиксации фронта или спада тактового сигнала. Далее приведен список чувствительности на примере модуля D-триггера со сбросом.

```
module Dtr (input clock, input data, input reset, output reg q);
  always @(posedge clock or posedge reset)
  begin
    //сюда попадем, если clock или reset переключатся в лог. 1
    if(clock==1)//запомнить данные, если пришел тактовый сигнал
      q <= data;
    else
      if(reset==1)//обнулить данные, если пришел сигнал сброса
        q<=0;
  end
endmodule
```

Нужно отметить, что в данном примере используется новый тип данных для описания выходного порта *reg*.

В первой лабораторной работе было рассмотрено непрерывное присвоение, которое изменяет состояние цепи немедленно при изменении состояния входных операндов. Помимо этого существует процедурное присвоение, используемое в поведенческом описании. Оно изменяет состояние регистровых переменных под управлением некоторой процедуры, в которой присвоение имеет место.

В отличие от цепей регистры представляют собой элементы хранения данных. Ключевое слово для данного типа *reg*. Они сохраняют своё состояние от одного назначения до другого. По умолчанию регистровые данные принимают неизвестное состояние (x).

Объявление регистров осуществляется следующим образом:

reg[<разрядность>] <список переменных>;

где *reg* – ключевое слово, определяющее тип переменной.

Примером может служить объявление 4-битного параллельного регистра:

reg[3:0] r1;

или массива 6-битных регистров, состоящего из 20 элементов:

reg[5:0] r2[0:19];

Фактически данное описание эквивалентно определению 120-битного регистра. Как и с регистрами можно описать массив цепей или шин.

Следует отметить приемы работы с шинами (многоразрядными регистрами). Допустим, имеется выходная 8-битная шина:

wire[7:0] wOut;

и две 4-битые шины:

wire[3:0] wIn1=4'b1100;

wire[3:0] wIn2=4'b1000;

то соединение двух шин в одну выполняется с использованием операции конкатенации:

assign wOut={wIn2,wIn1};

после чего в *wOut* будет значение 8'b10001100.

И обратная задача: разделение шины на две.

assign wIn2=wOut[7:4];

assign wIn1=wOut[3:0];

Различают два вида процедурных присвоений: блочное и внеблочное. Эти виды определяют различное поведение процедуры в последовательных блоках. Блочное присвоение (знак =) выполняется немедленно, в том месте, где оно встретилось в тексте, при этом переменная сразу меняет свое значение и может быть изменена неоднократно в одной ветви алгоритма.

Внеблочное присвоение (<=) изменяет значение переменной только в момент выхода из блока, поэтому, чтобы не было конфликтов в одной цепи, может быть только одна операция внеблочного присвоения. Другим отличием блочного от внеблочного является то, что в первом случае группа присвоений выполняется последовательно (если компилятор не оптимизирует), во втором – параллельно, что может сократить временные затраты. Поэтому по возможности рекомендуется по умолчанию использовать внеблочное присвоение.

В языке verilog существует удобный инструмент для конфигурирования модуля – параметры. Параметры не относятся ни к регистрам, ни к цепям. Они не могут быть переменными – это всегда константы. Объявляются параметры следующим образом:

parameter <список значений>;

При объявлении параметр всегда должен быть проинициализирован и содержать выражения присваивания, разделенные запятой. Пример применения параметра приведен ниже, который представляет собой модификацию D-триггера в параллельный регистр со сбросом, разрядность которого устанавливается параметром.

```
module DtrParam (input clock, input[BIT_DEPTH-1:0] data, input reset,
   output reg[BIT_DEPTH-1:0] q);
   parameter BIT_DEPTH=2;
   always @(posedge clock or posedge reset)
   begin
      //сюда попадем, если clock или reset переключатся в лог. 1
      if(clock==1)//запомнить данные, если пришел тактовый сигнал
         q <= data;
      else
         if(reset==1)//обнулить данные, если пришел сигнал сброса
            q<=0;
   end
endmodule
```

Объявление:

DtrParam #(8)DtrParam_inst(.clock(clk),.data(data),.reset(reset),.q(out));

где *#(8)* устанавливает значение параметра (*BIT_DEPTH*) равное 8.

Формальный синтаксис описания экземпляра модуля приведен ниже:

<имя модуля> [<список параметров>] <список экземпляров>;

имя модуля – обозначает имя модуля, экземпляр которого создается. Опциональное поле *список параметров* предназначено для модификации параметров модуля. Синтаксис этого поля выглядит следующим образом:

#(<значение1>, <значение 2>,…,<значение N>)

Список экземпляров представляется следующим образом:

<имя экземпляра>(<список соединений>)

Имя экземпляра должно быть уникальным в пределах модуля. *Список соединений* может быть организован двумя способами:

1) список состоит из имен цепей, следующих друг за другом в порядке, определенном порядком портов модуля, как и в большинстве языках программирования при вызове функций. Имена соединительных цепей разделяются запятыми. Такой порядок называется список соединений с позиционным соответствием;

2) Ключевое соответствие: для каждой цепи прямо указывается порт, к которому цепь должна быть подключена. Для этого используется следующая инструкция:

.<имя порта>(<имя цепи>)

Стоит отметить, что Vivado использует введенные в модуле параметры и автоматически их добавляет в форму настройки модуля (рисунок 8) при выполнении блочного проектирования (будет использовано в 4-й лабораторной работе).

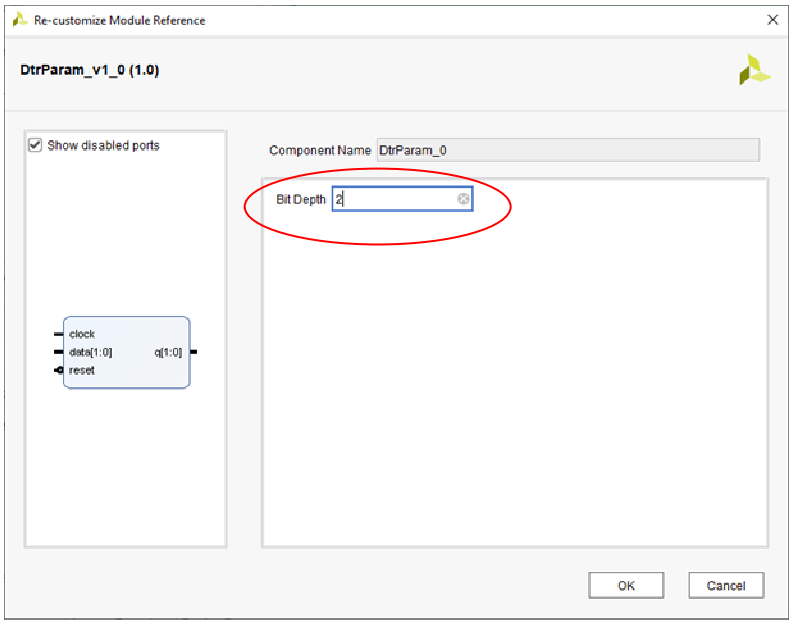


Рисунок 8 – Окно настройки модуля

Одним из самых необходимых инструментов для организации поведенческого описания является ветвление. Синтаксис этого оператора очень похож на синтаксис на языке Си. Общий вид оператора выглядит следующим образом:

```
if(<условие 1 выполняется?>)
[begin]
   <действия>
[end]
else
  if(<условие 2 выполняется?>)
    <действие>
```

Стоит отметить, что при наличии только одного действия ключевые слова *begin* и *end* необязательны.

Дополнительно можно применять краткие условия, причем как при работе с регистрами, так и при непрерывном присваивании.

```
assign var1 = (условие) ? var2:var3;
```

То есть, если условие выполнится, то к переменной *var1* будет подключена переменная *var2*, иначе *var3*.

Как правило, прежде чем применить разработанный модуль в общем проекте и тем более в «железе» должно быть выполнено его полное тестирование. Допустим, есть разработанный модуль *DtrParam*, который представляет собой параллельный регистр. Необходимо исследовать его поведение во всех ситуациях, то есть комбинациях входных значений.

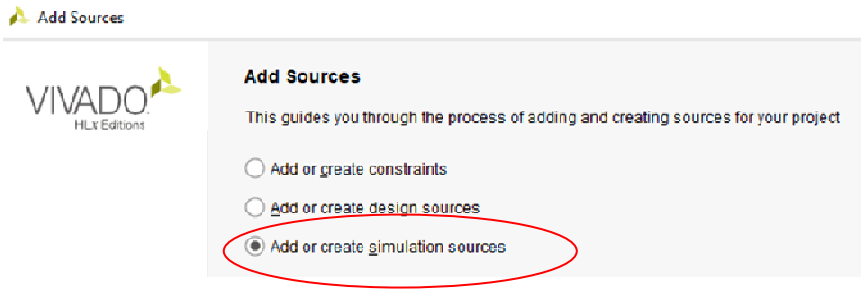
План тестирования:

1) на тактовом входе постоянная рабочая частота;

2) с определенной периодичностью изменяются входные данные для регистра;

3) в некоторый момент необходимо послать сигнал сброса для проверки реакции регистра.

Для начала тестирования должен быть организован тестовый стенд (TestBench), для эмуляции входных воздействий по заданному алгоритму. Для этого необходимо создать исходный файл для организации симуляции, как показано на рисунке 9.



Принято давать тестовым стендам специфические имена, характеризующие их назначение. К примеру, TestBench для *DtrParam* может быть назван *tbDtrParam,* как показано на рисунке 10.

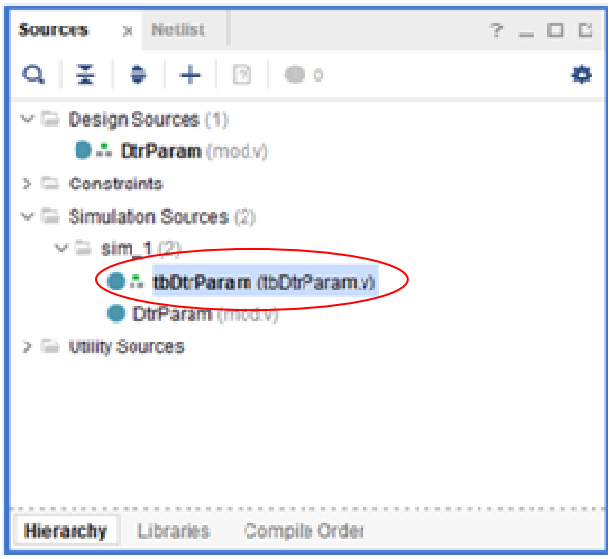


Рисунок 10 – Окно настройки модуля

У тестовых модулей, как правило, не должно быть портов ввода-вывода. Ниже приведено содержание модуля согласно запланированному сценарию.

```
`timescale 10ns / 10ns
//где timescale шаг_симуляции/шаг_моделирования - директива определения временной шкалы
//шаг_симуляции - цена деления одного шага симуляции
//шаг_моделирования - цена деления одного шага моделирования, например, для оценки времени выполнения операции в долях шага симуляции. Чем сильнее отличается шаг моделирования от шага симуляции, тем дольше выполняется моделирование, поэтому если не требуется такая оценка, то разницу стоит свести к нулю.
//определение модуля тестового стенда
module tbDtrParam();
  //Определение требуемых для симуляции переменных (регистров и цепей)
```

```
        reg clk=0; //Регистр для организации тактового сигнала
        reg[4:0] data=0;//Регистр данных, который в данном тесте будет инкрементироваться и соединятся с тестируемым модулем
        reg reset=0;//Регистр для организации сброса для тестового модуля DtrParam
        wire[2:0] out;//Выходная цепь
        //Объявление модуля
        //#(3) - устанавливает значение параметра разрядности (BIT_DEPTH=3)
        DtrParam #(3)DtrParam_inst(.clock(clk),.data(data),.reset(reset),.q(out));
        //Определение переменной-события evReset, будет служить для организации сценария со сбросом
        event evReset;
        //Определение блока первичной инициализации.
        //Здесь будет описан сценарий возникновения события сброса через 12 шагов
        //после запуска симуляции. Количество шагов не имеет значения, нужно
        //отличное от нуля для проверки реакции тестируемого модуля
        initial
        begin
          #12 //Ожидание 12 шагов симуляции
          ->evReset;//Запуск события сброса
        end
        //Определение процесса для организации тактового сигнала
        always//Без параметров. Блок вызывается при каждом шаге симуляции
        begin
          //Данный блок представляет собой бесконечный цикл.
          #1//ожидание в 1 шаг
          clk<=!clk;//переворот значения регистра, чтобы организовать 0,1,0,1,0,1....
          //На появление единицы в clk должен реагировать модуль DtrParam, так как
          //организованные в нем триггеры изменяют своё значение только при
          //изменении тактового сигнала с 0 на 1.
        end
        //Определение процесс-блока для реакции на любое изменение тактового сигнала
        always @(clk)
          data<=data+1;//изменение данных в регистре данных
        //Определение процесс-блока для реакции на событие сброса
        always @(evReset)
        begin
          reset<=1;//установка в единицу
          #1//и через шаг
          reset<=0;//сброс в ноль
        end
      endmodule
```

После ввода текста модуля необходимо запустить симуляцию, как показано на рисунке 11.

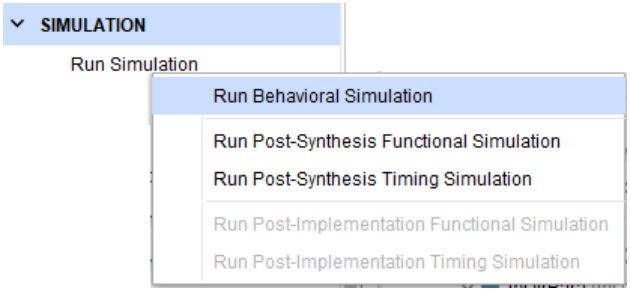


Рисунок 11 – Окно настройки модуля

Результат симуляции будет выведен на график, как показано на рисунке 12.

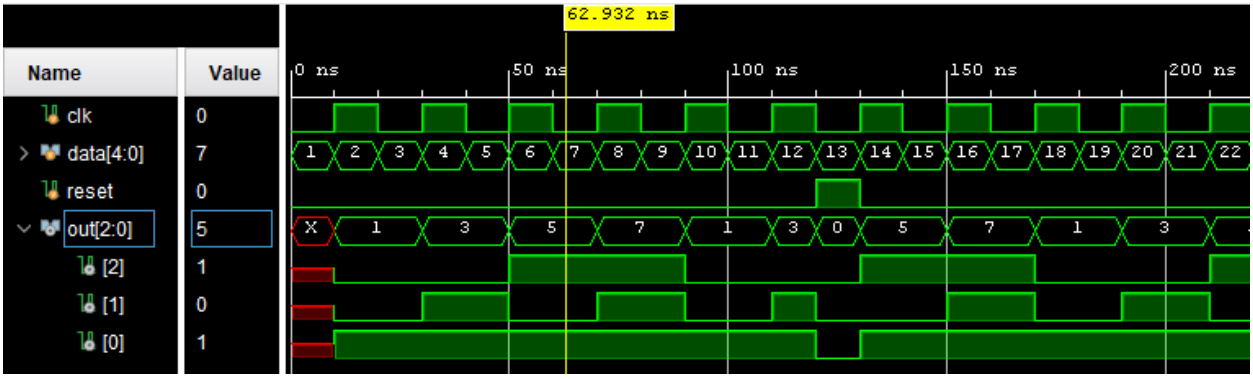


Рисунок 12 – Окно настройки модуля

В столбце Name отображаются используемые при симуляции регистры и цепи, причем шина может быть развернута на отдельные линии (см. out[2:0] – [2], [1], [0]). В столбце Value отображается последнее значение цепей и регистров на момент окончания симуляции или в позиции маркера (показано желтым цветом на рисунке 12). Полученная диаграмма должна быть проанализирована и верифицирована в соответствии со спецификацией на модуль и запланированной работой тестового стенда.

1) значение регистра clk соответствует работе процесса тактового сигнала, а именно начальное состояние 0, с задержкой в 1 шаг (цена деления 10 нс) происходит переворот;

2) параметр data соответствует работе процесса для реакции на изменение тактового сигнала, а именно инкрементирование значения регистра при любом изменении значения в регистре clk;

3) параметр reset соответствует сценарию возникновения события через 12 шагов после запуска симуляции и реакции процесс-блока на это событие, а именно через 120 нс выполняется переключение регистра в состояние логической единицы, затем через 10 нс возврат в логический ноль;

4) далее выполняется верификация работы самого модуля DtrParam (параллельного регистра со сбросом):

а) в начальном состоянии до появления тактового сигнала на выходе (out) было неопределенное состояние, что отмечено красным цветом и символом X;

б) так как регистр захватывает значение со входа (data) только по фронту тактового сигнала, то в момент захвата на входе было значение 1, что отразилось и удерживалось (параллельный регистр – это ячейка памяти) на выходе, при возникновении следующего нарастания фронта DtrParam захватит значение 3 и т.д.;

в) в момент возникновения сигнала сброса (фронт импульса) происходит сброс регистра DtrParam, что отражено значением 0 на выходе;

г) при появлении следующего тактового сигнала происходит захват значения на входе, что отражается значением 5 на выходе регистра;

д) отличия значений на входе и на выходе объясняются различной разрядностью регистра для организации входных данных и тестируемого модуля. То есть модулем DtrParam захватываются только младшие 3 бита, так как при объявлении был определен 3-битный регистр *#(3)*.

Таким образом, с использованием встроенных в vivado средств симуляции проведена верификация разработанного модуля DtrParam.

Примечание.

Если выполняется задача реализации синхронного модуля с наличием внешнего тактового входа в списке чувствительности

```
module Mod (input clock,…);
   …
   always @(posedge clock…)
…
```

то следует отметить, что на микросхеме FPGA контакты разделены на блоки ввода вывода (рисунок 13, выделены бордовым, зеленым, желтым, фиолетовым).

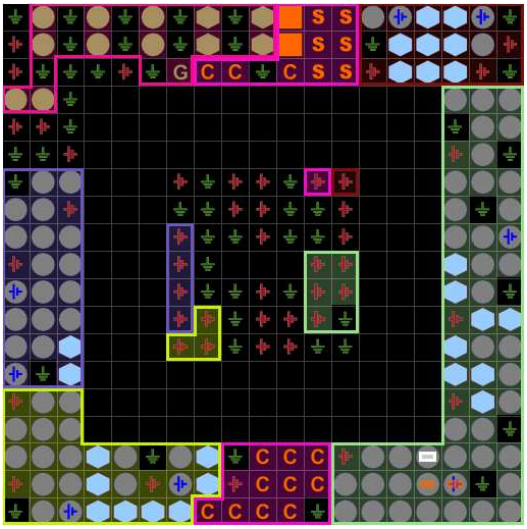


Рисунок 13 – Контакты FPGA

Внутри каждого блока присутствуют выводы общего назначения (серые контакты) и выводы для организации тактового сигнала (голубые контакты). Причем некоторые линии тактового сигнала попарно сгруппированы для организации дифференциального входа (положительный и отрицательный вход). Эти входы снабжены блоками управления тактовых сигналов, что обеспечивает равномерное распределение тактового сигнала по кристаллу.

При работе с отладочной платой потребуется обеспечивать ввод тактового сигнала, например, от кнопки, которая подключена к контакту U18 (серая контактная площадка). Во время трассировки и компоновки будет выведено сообщение об ошибке:

[Place 30-574] Poor placement for routing between an IO pin and BUFG. If this sub optimal condition is acceptable for this design, you may use the CLOCK_DEDICATED_ROUTE constraint in the .xdc file to demote this message to a WARNING. However, the use of this override is highly discouraged. These examples can be used directly in the .xdc file to override this clock rule.

<set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clock_IBUF]>

Это сообщение указывает на выбор «неоптимального» входа для организации тактового сигнала.

Решениями данной проблемы может быть два:

1) перенос цепи clock на тактовый вход (неприемлемо в рамках данной лабораторной работы). Может быть использовано при предварительном моделировании перед трассировкой печатной платы для устройства, содержащего FPGA;

2) если маршрут приемлем, то добавление в файл ограничений (Sources->Constraints->.xdc) строчки

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clock]

где *clock* – это тактовый вход модуля.

3 Порядок выполнения работы

В ходе данной работы необходимо создать, верифицировать на симуляторе и испытать на отладочной плате модули счетчика, делителя частоты, дешифратора, и шинного мультиплексора.

Для этого потребуется выполнить следующие действия:

3.1 Изучите предложенный в п. 2 теоретический материал.

3.2 Создайте проект и добавьте в него модуль счетчика импульсов. В модуле необходимо предусмотреть количество и виды портов согласно теоретическому материалу.

3.3 Параметризируйте счетчик, то есть добавьте параметры разрядность (по умолчанию 4) и коэффициент счета (по умолчанию 10). Выход должен быть представлен как шина данных.

3.4 Реализуйте функциональность счетчика в процесс-блоке.

3.5 Выполните компиляцию, симуляцию и верификацию модуля. Для симуляции необходимо разработать модуль тестового стенда.

3.6 Выполните синтез модуля и назначьте на вход тактового сигнала кнопку BTNU, сброса – BTND, на выход LD0–LD3.

3.7 Выполните генерацию файла прошивки, запрограммируйте ПЛИС и проверьте работоспособность.

3.8 Добавьте и запрограммируйте модуль делителя частоты. Сделать параметрами коэффициент деления и разрядность встроенного регистра счета.

3.9 Сделайте его модулем верхнего уровня.

3.10 Выполните компиляцию, симуляцию, верификацию и его проверку в «железе».

3.11 Добавьте и запрограммируйте модуль 2-разрядного мультиплексора шин. То есть входными и выходными должны быть не отдельные линии, а шины. Разрядность шин мультиплексора должна быть параметризируемой (по умолчанию 4). Адресные входы должны быть представлены в виде шины адреса.

3.12 Выполните компиляцию, симуляцию, верификацию и его проверку в «железе».

3.13 Выполните синтез модуля мультиплексора. К каждой шине мультиплексора подключите переключатели отладочной платы: для нулевой шины SW0–SW3, для 1-й – SW4–SW7 и т.д. К адресным входам A0 подключить кнопку BTNL, A1 – BTNR. К выходу светодиоды LD0–LD3.

3.14 Добавьте и запрограммируйте модуль 2-разрядного дешифратора. Выход должен быть представлен в виде шины данных.

3.15 Выполните компиляцию, симуляцию, верификацию и его проверку в «железе».

3.16 Оформите отчет, содержащий титульный лист, введение, ход выполнения работы, ответы на контрольные вопросы и выводы.

3.17 Защитите отчет у преподавателя.

4 Контрольные вопросы

4.1 Назначение процесс-блока.

4.2 В чем разница между поведенческим и структурным описанием?

4.3 Какое назначение типа данных регистра?

4.4 Что такое список чувствительности и как он организовывается?

4.5 Как описать цепь, представляющую собой шину данных?

4.6 Как описать 16-битный регистр?

4.7 Назначение и принцип действия делителя частоты.

4.8 Какое назначение коэффициента счета и разрядности счетчика?

4.9 Как организовать простой D-триггер?

4.10 Какое назначение симуляции?

4.11 Что такое тестовый стенд?

4.12 Что такое параметр и как его использовать?

4.13 Что такое событие в симуляции?

4.14 Назначение задержек в симуляции.

4.15 Какие отличия между блочным и внеблочным присвоением?

4.16 Какой порядок создания экземпляра модуля?

Список литературы

1 Basys3™ FPGA Board Reference Manual. URL: https://reference.digilentinc.com/_media/reference/programmable-logic/basys-3/basys3_rm.pdf (дата обращения: 31.01.2020);

2 Product Tables and Product Selection Guide. URL https://www.xilinx.com/support/documentation/selection-guides/cost-optimized-product-selection-guide.pdf#A7 (дата обращения);

3 Добавление testbench'ей на языке Verilog в проект Vivado URL: https://www.srns.ru/wiki/Добавление_testbench'ей_на_языке_Verilog_в_проек

т_Vivado (дата обращения 01.02.2020);
4 Basys3™ Schematic. URL: https://reference.digilentinc.com/_media/reference/programmable-logic/basys-3/basys-3_sch.pdf (дата обращения: 31.01.2020).