Лабораторный практикум

«Проектирование цифровых устройств с помощью Verilog HDL»

Лабораторная работа №1 Введение в Verilog HDL

1.1 Возникновение языков описания цифровой аппаратуры

Цифровые устройства — это устройства, предназначенные для приёма и обработки цифровых сигналов. Цифровыми называются сигналы, которые можно рассматривать в виде набора дискретных уровней. В цифровых сигналах информация кодируется в виде конкретного уровня напряжения. Как правило выделяется два уровня — логический «0» и логическая «1».

Цифровые устройства стремительно развиваются с момента изобретения электронной лампы, а затем транзистора. Со временем цифровые устройства стали компактнее, уменьшилось их энергопотребление, возрасла вычислительная мощность. Так же разительно возросла сложность их структуры.

Графические схемы, которые применялись для проектирования цифровых устройств на ранних этапах развития, уже не могли эффективно использоваться. Потребовался новый инструмент разработки, и таким инструментом стали языки описания аппаратной части цифровых устройств (Hardware Description Languages, HDL), которые описывали цифровые структуры формализованным языком, чем-то похожим на язык программирования.

Совершенно новый подход к описанию цифровых схем, реализованный в языках HDL, заключается в том, что с помощью их помощью можно описывать не только структуру, но и поведение цифрового устройства.Окончательная структура цифрового устройства получается путём обработки таких смешанных описаний специальной программой — синтезатором.

Такой подход существенно изменил процесс разработки цифровых устройств, превратив громоздкие, тяжело читаемые схемы в относительно простые и доступные описания поведения.

В данном курсе мы рассмотрим язык описания цифровой аппаратуры Verilog HDL — одни из наиболее распространённых на текущий момент. И начнём мы с разработки наиболее простых цифровых устройств — логических вентилей.

1.2 HDL описания логических вентилей

Логические вентили реализуют функции алгебры логики: И, ИЛИ, Исключающее ИЛИ, НЕ. Напомним их таблицы истинности:

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 1.1: И

a	b	$a \mid b$
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 1.2: ИЛИ

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Таблица 1.3: Исключающее ИЛИ

a	$\bar{a}$
0	1
1	0

Таблица 1.4: НЕ

Начнём знакомиться с Verilog HDL с описания логического вентиля «И». Ниже приведен код, описывающий вентиль с точки зрения его структуры:

```
1  module and_gate(
2          input  a,
3          input  b,
4          output result)
5
6  assign result = a & b;
7
8  endmodule
```

Листинг 1.1: Модуль, описывающий вентиль «И»

Описанный выше модуль можно представить как некоторый «ящик», в который входит 2 провода с названиями *«a»* и *«b»* и из которого выходит один провод с названием *«result»*. Внутри этого блока результат выполнения операции «И» (в синтаксисе Verilog записывается как «&») над входами соединяют с выходом.

Схемотично изобразим этот модуль:

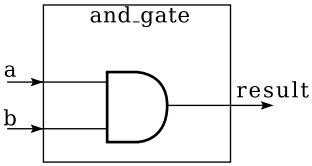

Рис. 1.1: Структура модуля «and_gate»

Аналогично опишем все оставшиеся вентили:

```
1  module or_gate(
2          input  a,
3          input  b,
4          output result)
5
6  assign result = a | b;
7
8  endmodule
```

Листинг 1.2: Модуль, описывающий вентиль «ИЛИ»

```
1 module xor_gate(
2         input  a,
3         input  b,
4         output result)
5
6 assign result = a ^ b;
7
8 endmodule
```

Листинг 1.3: Модуль, описывающий вентиль «Исключающее ИЛИ»

```
1 module not_gate(
2         input  a,
3         output result)
4
5 assign result = ~a;
6
7 endmodule
```

Листинг 1.4: Модуль, описывающий вентиль «НЕ»

В проектировании цифровых устройств логические вентили наиболее часто используются для формулировки и проверки сложных условий, например:

```
1 if ( (a & b) | (~c) ) begin
2         ...
3 end
```

Листинг 1.5: Пример использования логических вентилей

Условие будет выполняться либо когда *не* выполнено условие *«c»*, либо когда одновременно выполняются условия *«a»* и *«b»*. *Здесь и далее под условием понимается логический сигнал, отражающий его истинность.*

В качестве входов, выходов и внутренних соединений в блоках могут использоваться шины — группы проводов. Ниже приведен пример работы с шинами:

```
1  module bus_or(
2          input  [7:0] x,
3          input  [7:0] y,
4          output [7:0] result);
5
6  assign result = x | y;
7
8  endmodule
```

Листинг 1.6: Модуль, описывающий побитовое «ИЛИ» между двумя шинами

Это описание описывает побитовое «ИЛИ» между двумя шинами по 8 бит. То есть описываются восемь логических вентилей «ИЛИ», каждый из которых имеет на входе соответствующие разряды из шины *«x»* и шины *«y»*.

При использовании шин можно в описании использовать конкретные биты шины и группы битов. Для этого используют квадратные скобки после имени шины:

```
1  module bitwise_ops(
2          input  [7:0] x,
3          output [4:0] a,
4          output       b,
5          output [2:0] c);
6
7  assign a = x[5:1];
8  assign b = x[5]   | x[7];
9  assign c = x[7:5] ^ x[2:0];
10
11 endmodule
```

Листинг 1.7: Модуль, демонстрирующий битовую адресацию шин

Такому описанию соответствует следующая структурная схема, приведённая на Рис. 1.2

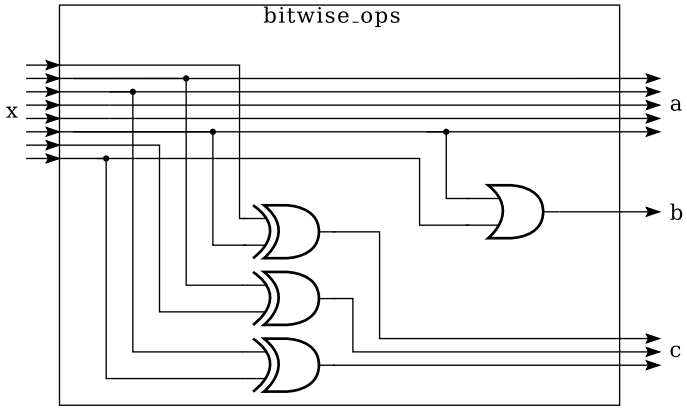

Рис. 1.2: Структура модуля «bitwise_ops»

Впрочем, реализация ФАЛ с помощью логических вентилей не всегда представляется удобной. Допустим нам нужно описать таблично-заданную ФАЛ. Тогда описания этой функции при помощи логических вентилей нам придётся сначала минимизировать её и только после этого, получив логическое выражение (которое, несмотря на свою минимальность, не обязательно является коротким), сформулировать его с помощью языка Verilog HDL. Как видно, ошибку легко допустить на любом из этих этапов.

Одно из главных достоинств Verilog HDL — это возможность описывать поведение цифровых устройств вместо описания их структуры.

Программа-синтезатор анализирует синтаксические конструкции поведенческого описания цифрового устройства на Verilog HDL, проводит оптимизацию и, в итоге, вырабатывает структуру, реализующую цифровое устройство, которое соответствует заданному поведению.

Используя эту возможность, опишем таблично-заданную ФАЛ на Verilog HDL:

```
1  module function(
2    input x0,
3    input x1,
4    input x2,
```

```
 5   output reg y);
 6
 7 wire [2:0] x_bus;
 8 assign x_bus = {x2, x1, x0};
 9
10 always @(xbus) begin
11   case (xbus)
12     3'b000:  y <= 1'b0;
13     3'b010:  y <= 1'b0;
14     3'b101:  y <= 1'b0;
15     3'b110:  y <= 1'b0;
16     3'b111:  y <= 1'b0;
17     default: y <= 1'b1;
18   endcase;
19 end;
20
21 endmodule;
```

Листинг 1.8: Пример описания таблично-заданной ФАЛ на Verilog HDL

Описание, приведённое выше, определяет y, как таблично-заданную функцию, которая равна нулю на наборах 0, 2, 5, 6, 7 и единице на всех остальных наборах.

Остановимся подробнее на новых синтаксических конструкциях:

Описание нашего модуля начинается с создания трёхбитной шины «x_bus» на строке 7.

После создания шины «x_bus», на она подключается к объединению проводов «x2», «x1» и «x0» с помощью оператора assign как показано на Рис. 1.3.

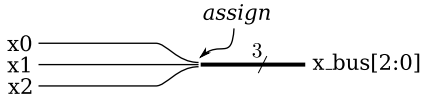

Рис. 1.3: Действие оператора **assign**

Затем начинается функциональный блок **always**, на котором мы остановимся подробнее.

Verilog HDL описывает цифровую аппаратуру, которая су-

ществует вся одновременно, но инструменты анализа и синтеза описаний являются программами и выполняются последовательно на компьютере. Так возникла необходимость последовательной программе «рассказать» про то, какие события приводят к срабатыванию тех или иных участков кода. Сами эти участки назвали процессами. Процессы обозначаются ключевым словом **always**.

В скобках после символа @ указывается так называемый *список чувствительности процесса*, т.е. те сигналы, изменение которых должно приводить к пересчёту результатов выолнения процесса.

Например, результат ФАЛ надо будет пересчитывать каждый раз, когда изменился входной вектор (любой бит входного вектора, т.е. любая переменная ФАЛ). Эти процессы можно назвать блоками, или частями будущего цифрового устройтсва.

Новое ключевое слово **reg** здесь необходимо потому, что в выходной вектор происходит запись, а запись в языке Verilog HDL разрешена только в «регистры» — специальные «переменные», предусмотренные в языке. Данная концепция и ключевое слово reg будет рассмотрено гораздо подробнее в следующей лабораторной работе.

Оператор **<=** называется оператором *неблокирующего присваивания*. В результате выполнения этого оператора то, что стоит справа от него, «помещается» («кладется», «перекладывается») в регистр, который записан слева от него. Операции неблокирующего присваивания происходят одновременно по всему процессу.

Оператор **case** описывает выбор действия в зависимости от анализируемого значения. В нашем случае анализируется значение шины «x_bus». Ключевое слово **default** используется для обозначения всех остальных (не перечисленных) вариантов значений.

Константы и значения в языке Verilog HDL описываются следующим образом: сначала указывается количество бит, затем после апострофа с помощью буквы указывается формат и, сразу за ним, записывается значение числа в этом формате.

Возможные форматы:

- b – бинарный, двоичный;

- h – шестнадцатеричный;
- d – десятичный.

Немного расширив это описание, легко можно определить не одну, а сразу несколько ФАЛ одновременно. Для упрощения записи сразу объединим во входную шину все переменные. В выходную шину объединим значения функций:

```
 1 module decoder(
 2   input  [2:0] x,
 3   output [3:0] y);
 4
 5 reg [3:0] decoder_output;
 6 always @(x) begin
 7   case (x)
 8     3'b000: decoder_output <= 4'b0100;
 9     3'b001: decoder_output <= 4'b1010;
10     3'b010: decoder_output <= 4'b0111;
11     3'b011: decoder_output <= 4'b1100;
12     3'b100: decoder_output <= 4'b1001;
13     3'b101: decoder_output <= 4'b1101;
14     3'b110: decoder_output <= 4'b0000;
15     3'b111: decoder_output <= 4'b0010;
16   endcase;
17 end;
18
19 assign y = decoder_output;
20
21 endmodule;
```

Листинг 1.9: Описание дешифратора на языке Verilog HDL

Теперь нам удалось компактно записать четыре функции, каждая от трёх переменных:

$$y_0 = f(x_2, x_1, x_0);$$
$$y_1 = f(x_2, x_1, x_0);$$
$$y_2 = f(x_2, x_1, x_0);$$
$$y_3 = f(x_2, x_1, x_0).$$

Но, если мы посмотрим на только что описанную конструк-

цию под другим углом, мы увидим, что это описание можно трактовать следующим образом: «поставить каждому возможному входному вектору x в соответствие заранее определенный выходной вектор y». Такое цифровое устройство называют *дешифратором*.

На Рис. 1.4 показано принятое в цифровой схемотехнике обозначение дешифратора.

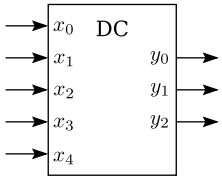

Рис. 1.4: Графическое обозначение дешифратора

Заметим, что длины векторов не обязательно должны совпадать, а единственным условием является полное покрытие всех возможных входных векторов, что, например, может достигаться использованием условия **default** в операторе **case**.

Дешифраторы активно применяются при разработке цифровых устройств. В большинстве цифровых устройств в явном или неявном виде можно встретить дешифратор.

Рассмотрим еще один интересный набор ФАЛ:

```
 1 module decoder(
 2   input [2:0] a,
 3   input [2:0] b,
 4   input [2:0] c,
 5   input [2:0] d,
 6   input [1:0] s,
 7   output reg [2:0] y);
 8
 9 always @(a,b,c,d,s) begin
10   case (s)
11     3'b00:   y <= a;
12     3'b01:   y <= b;
13     3'b10:   y <= c;
14     3'b11:   y <= d;
15     default: y <= a;
```

```
16    endcase;
17  end;
18
19  endmodule;
```

Листинг 1.10: Описание мультиплексора на языке Verilog HDL

Что можно сказать об этом описании? Выходной вектор y — это результат работы трёх ФАЛ, каждая из которых является функцией 6 переменных. Так, $y_0 = f(a_0, b_0, c_0, d_0, s_1, s_0)$.

Анализируя оператор **case**, можно увидеть, что главную роль в вычислении значения ФАЛ играет вектор s, в результате проверки которого выходу ФАЛ присваивается значение «выбранной» переменной.

Получившееся устройство называется *мультиплексор*.

Мультиплексор работает подобно коммутирующему ключу, замыкающему выход с выбранным входом. Для выбора входа мультиплексору нужен сигнал управления.

Графическое изображение мультиплексора приведено на Рис. 1.5

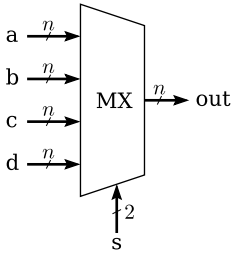

Рис. 1.5: Графическое обозначение мультиплексора

Особенно хочется отметить, что на самом деле никакой «проверки» сигнала управления не существует и уж тем более не существует «коммутации», ведь мультиплексор — это таблично-заданная ФАЛ. Результат выполнения этой ФАЛ выглядит так, как будто происходит «подключение» «выбранной» входной шины к выходной.

Приведём для наглядности таблицу, задающую ФАЛ для одного бита выходного вектора (число ФАЛ в мультиплексоре и,

следовательно, число таблиц, равняется числу бит в выходном векторе). Для краткости выпишем таблицу наборами строк вида: $f(s_1, s_0, a_0, b_0, c_0, d_0) = y_0$ в четыре столбца.

Обратите внимание, что в качестве старших двух бит входного вектора для удобства записи и анализа мы выбрали переменные «управляющего» сигнала, а выделение показывает какая переменная «поступает» на выход функции f:

$f(00\mathbf{0}000) = 0$	$f(010\mathbf{0}00) = 0$	$f(1000\mathbf{0}0) = 0$	$f(11000\mathbf{0}) = 0$
$f(00\mathbf{0}001) = 0$	$f(010\mathbf{0}01) = 0$	$f(1000\mathbf{0}1) = 0$	$f(11000\mathbf{1}) = 1$
$f(00\mathbf{0}010) = 0$	$f(010\mathbf{0}10) = 0$	$f(1000\mathbf{1}0) = 1$	$f(11001\mathbf{0}) = 0$
$f(00\mathbf{0}011) = 0$	$f(010\mathbf{0}11) = 0$	$f(1000\mathbf{1}1) = 1$	$f(11001\mathbf{1}) = 1$
$f(00\mathbf{0}100) = 0$	$f(010\mathbf{1}00) = 1$	$f(1001\mathbf{0}0) = 0$	$f(11010\mathbf{0}) = 0$
$f(00\mathbf{0}101) = 0$	$f(010\mathbf{1}01) = 1$	$f(1001\mathbf{0}1) = 0$	$f(11010\mathbf{1}) = 1$
$f(00\mathbf{0}110) = 0$	$f(010\mathbf{1}10) = 1$	$f(1001\mathbf{1}0) = 1$	$f(11011\mathbf{0}) = 0$
$f(00\mathbf{0}111) = 0$	$f(010\mathbf{1}11) = 1$	$f(1001\mathbf{1}1) = 1$	$f(11011\mathbf{1}) = 1$
$f(00\mathbf{1}000) = 1$	$f(011\mathbf{0}00) = 0$	$f(1010\mathbf{0}0) = 0$	$f(11100\mathbf{0}) = 0$
$f(00\mathbf{1}001) = 1$	$f(011\mathbf{0}01) = 0$	$f(1010\mathbf{0}1) = 0$	$f(11100\mathbf{1}) = 1$
$f(00\mathbf{1}010) = 1$	$f(011\mathbf{0}10) = 0$	$f(1010\mathbf{1}0) = 1$	$f(11101\mathbf{0}) = 0$
$f(00\mathbf{1}011) = 1$	$f(011\mathbf{0}11) = 0$	$f(1010\mathbf{1}1) = 1$	$f(11101\mathbf{1}) = 1$
$f(00\mathbf{1}100) = 1$	$f(011\mathbf{1}00) = 1$	$f(1011\mathbf{0}0) = 0$	$f(11110\mathbf{0}) = 0$
$f(00\mathbf{1}101) = 1$	$f(011\mathbf{1}01) = 1$	$f(1011\mathbf{0}1) = 0$	$f(11110\mathbf{1}) = 1$
$f(00\mathbf{1}110) = 1$	$f(011\mathbf{1}10) = 1$	$f(1011\mathbf{1}0) = 1$	$f(11111\mathbf{0}) = 0$
$f(00\mathbf{1}111) = 1$	$f(011\mathbf{1}11) = 1$	$f(1011\mathbf{1}1) = 1$	$f(11111\mathbf{1}) = 1$

Пример задания лабораторной работы

Пример формируется...

Лабораторная работа №6
FLASH память

2.1 Виды энергонезависимой памяти

Ни один из блоков цифровых устройств, которые мы рассмотрели ранее не способен хранить информацию при отсутствии питания.

Чтобы решить эту проблему, на заре вычислительной техники, данные в цифровое устройство после подачи питания загружали с таких носителей, как перфокарты и, позже, магнитные ленты. Ещё позже для целей хранения информации при отсутствии питания были разработаны накопители на гибких магнитных дисках — дискетах и жёстких магнитных дисках — «HDD». На данный момент для хранения данных при отсутствии питания наиболее широко применяется FLASH-память.

Энергонезависимые накопители информации обладают как преимуществами, так и недостатками по сравнению с энергозависимой RAM-памяти.

Как правило, энергонезависимая память существенно уступает по скорости работы RAM-памяти. Это ограничение удалось преодолеть только недавно: в 2016 году была представлена постоянная память, где информация хранится в виде спина электрона. Такая память по скорости работы не уступает современной RAM-памяти, такой как DDR5. Но подобная память ещё долгое время будет оставаться недоступной для рядового пользователя из-за высокой стоимости и малой степени освоения технологии серийного производства 10нм.

2.2 Принципы работы FLASH-памяти

В качестве элемента хранения информации FLASH-память использует транзистор с плавающим затвором, где состояние затвора определяет бит хранимой информации, изображённый на Рис.**??**

Рассмотрим работу такого транзистора.

Как вы видите, он содержит два затвора: управляющий и плавающий. Плавающий полностью находится в диэлектрике и при этом способен накапливать электроны. От величины накопленного заряда меняется «лёгкость» с которой транзистор открывается — т.е. величина напряжения «управляющий затвор—исток», при которой через транзистор начнёт течь ток.

Для хранения информации используют следующий принцип (см.Рис.**??**): чтобы считать информацию, на управляющий затвор подаётся напряжение чтения — среднее между самым сильным (в диэлектрике нет электронов) и самым слабым (в диэлектрике максимум электронов). Если транзистор открывается, значит в плавающем затворе были электроны и мы считаем, что в нём записан «0», если не открывается, значит электронов в плавающем затворе нет и записана «1».

Осталось понять как можно «заставить» электроны попадать в плавающий затвор, ведь он изолирован диэлектриком. Не вдаваясь в подробности скажем, что если подать достаточно высокое напряжение «управляющий затвор—сток», то у электронов хватит энергии, чтобы «перескочить» диэлектрик и попасть в плавающий затвор. А если изменить полярность этого напряжения, то можно заставить электроны покинуть плавающий затвор (см.Рис.**??**).

Самое важное в этой идее то, что если электроны попали в плавающий затвор они не могут самостоятельно покинуть его через диэлектрик и будут оставаться там в течении многих лет. Таким образом и достигается сохранение записанной информации при отсутствии питания.

Теперь мы знаем, что для того чтобы записать или считать информацию из FLASH-памяти надо использовать большую разность потенциалов. Но на самом деле транзистор устро-

ен таким образом, что энергия, необходимая чтобы «загнать» электроны в плавающий затвор меньше энергии, необходимой, чтобы их «выгнать». Это делается чтобы при чтении значения электроны не покидали плавающий затвор.

При такой организации становится сложно обеспечить очистку каждого транзистора в отдельности, поэтому обычно стирается целая группа ячеек.

Из-за особенностей транзистора с плавающим затвором, которые мы рассмотрели можно выделить следующие характерные черты FLASH-памяти:

- Запись значения возможна только из логической «1» в логический «0»;
- Удаление информации возможно только из группы ячеек одновременно (блока);
- Удаление и запись информации приводят к деградации ячеек памяти;
- Чтение также приводит к деградации ячеек памяти, но в меньшей степени.

На Рисунке**??** изображена общая структура FLASH-памяти. Как видно она практически не отличается от RAM-памяти: из ячеек строится матрица, контролируемая управляющим блоком. А сам управляющий блок обеспечивает коммуникацию с внешними устройствами, дешифрацию адреса и управление записью и чтением массива элементов памяти. Подключение FLASH-памяти и управление ей со стороны цифрового устройства полностью зависит от того, как реализован блок управления — доступ к содержимому FLASH-памяти может быть синхронный или асинхронный, по последовательной или параллельной шине, с разделением шин адреса и данных или без него.

2.3 Чип FLASH-памяти S29AL032D

Для практического знакомства с FLASH-памятью мы спроектируем контроллер микросхемы S29AL032D. Именно эта микросхема установлена на отладочной плате Altera DE1.

Основным источником информации о любой микросхеме

служат технические условия (англ. datasheet). В большинстве случаев в этом документе содержатся все необходимые сведения для использования микросхемы: электрические параметры, размеры и тип корпуса, информация о выводах, и многие другие сведения. В том числе datasheet содержит данные о протоколах информационного обмена.

В нашем случае микросхема уже подключена, поэтому из всего datasheet нас прежде всего интересует каким образом необходимо взаимодействовать с данной микросхемой, чтобы записать или считать данные.

Для разработки контроллера следует ознакомиться со следующими разделами документа:

11. Commands Definitions;
12. Write Operation Status;
17. AC Characteristics.

Далее будут приведены необходимые выдержки из документа, однако настоятельно рекомендуем ознакомиться с ним.

2.3.1 Операция чтения

Для чтения данных из чипа S29AL032D не требуется никакой дополнительной подготовки. Временная диаграмма чтения приведена в пункте 17.2 datasheet и представлена на Рис.**??**

Времена, указанные на диаграмме, приведены в Табл. 2.1

Обозн.	Описание	Min	Max
t_{RC}	Продолжительность цикла чтения	70нс	90нс
t_{ACC}	Задержка Адрес — Данные	70нс	90нс
t_{CE}	Задержка Выбор Чипа — Данные	70нс	90нс

Таблица 2.1: Временные характеристики операции чтения S29AL032D