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	$a \vert b$
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:

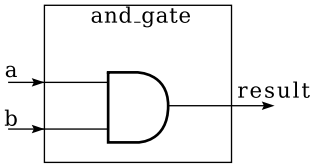

Рис. 1.1: Структура модуля «and_gate»

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:

```
module bus_or(
        input  [7:0] x,
        input  [7:0] y,
        output [7:0] result);

assign result = x | y;

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:

```
module bitwise_ops(
        input  [7:0] x,
        output [4:0] a,
        output       b,
        output [2:0] c);

assign a = x[5:1];
assign b = x[5]   | x[7];
assign c = x[7:5] ^ x[2:0];

endmodule
```

Листинг 1.7: Модуль, демонстрирующий битовую адресацию шин

Такому описанию соответствует следующая структурная схема:

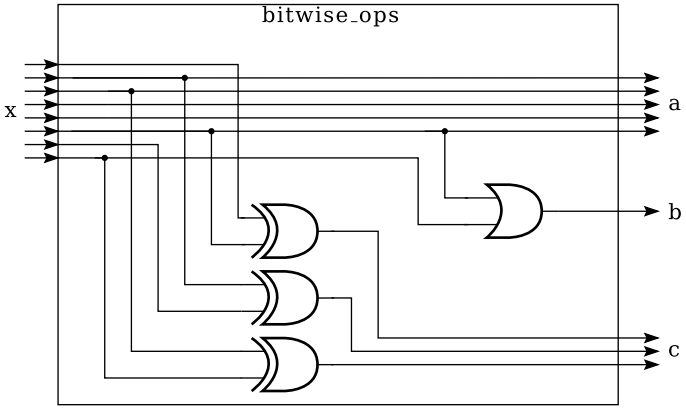

Рис. 1.2: Структура модуля «bitwise_ops»

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 на строке 8.

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

Verilog HDL описывает цифровую аппаратуру, которая существует 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 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

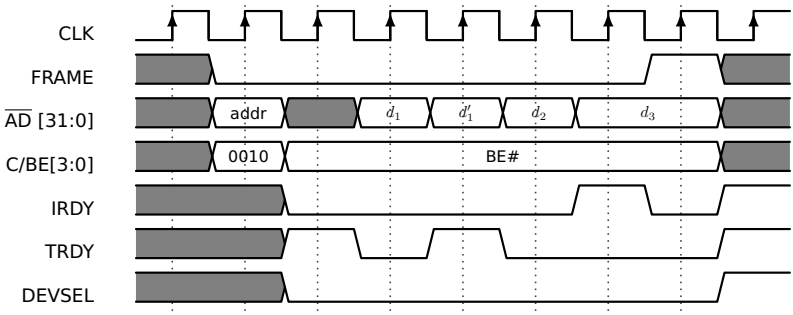

Рис. 1.3: Временная диаграмма операции чтения шины PCI

Лабораторная работа №6 FLASH память

2.1 Возникновение FLASH-памяти

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

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 жёстких магнитных дисках, которые известны нам по аббревиатуре «HDD».

Энергонезависимые накопители информации обладают как преимуществами, так и недостатками по сравнению с энергозависимой памятью.

Как правило, энергонезависимая память существенно уступает по скорости работы RAM-памяти. Это ограничение удалось преодолеть только недавно: в 2016 году широкой общественности 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.

На данный момент наиболее широко используемая энергонезависимая память - это память типа FLASH.

В качестве элемента хранения информации такая память использует транзистор с плавающим затвором, где состояние затвора определяет бит хранимой информации.

Рис. 2.1: Ячейка FLASH-памяти

Из-за использования транзистора с плавающим затвором, у FLASH-памяти есть характерные особенности:

- Запись значения возможна только из логической «1» в логический «0»;
- Удаление информации возможно 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.

Из-за физических процессов, протекающих в транзисторах во время записи значений в память и очистки содержимого.