

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 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 — один из наиболее распространённых на текущий момент. И начнём мы с разработки наиболее простых цифровых устройств — логических вентилях.

1.2 HDL описания логических вентилях

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

Таблица 1.3: Исключающее ИЛИ

a	b	$a b$
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 1.2: 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.

Схематично изобразим этот модуль:

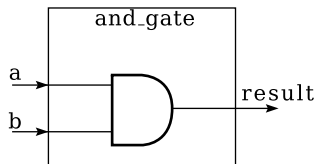


Рис. 1.1: Структура модуля «and_gate»

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 вентиляей

Условие будет выполняться либо когда *не* выполнено условие «с», либо когда одновременно выполняются условия «а» 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 вентилях не всегда представляется удобной. Допустим нам нужно описать таблично-заданную ФАЛ. Тогда описания этой функции при помощи логических вентилях 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.

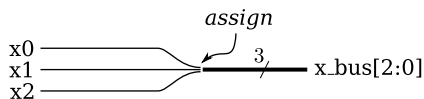


Рис. 1.3: Действие оператора **assign**

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

Verilog HDL описывает цифровую аппаратуру, которая су-

шествует 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 называли 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 выполнения 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 называть блоками, или частями будущего цифрового устройства.

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:

$$\begin{aligned}
 y_0 &= f(x_2, x_1, x_0); \\
 y_1 &= f(x_2, x_1, x_0); \\
 y_2 &= f(x_2, x_1, x_0); \\
 y_3 &= f(x_2, x_1, x_0).
 \end{aligned}$$

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.

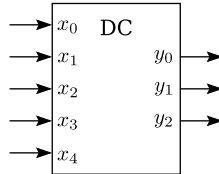


Рис. 1.4: Графическое обозначение дешифратора

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

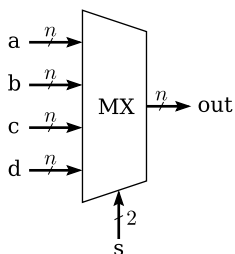


Рис. 1.5: Графическое обозначение мультиплексора

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 :

$f(000000) = 0$	$f(010000) = 0$	$f(100000) = 0$	$f(110000) = 0$
$f(000001) = 0$	$f(010001) = 0$	$f(100001) = 0$	$f(110001) = 1$
$f(000010) = 0$	$f(010010) = 0$	$f(100010) = 1$	$f(110010) = 0$
$f(000011) = 0$	$f(010011) = 0$	$f(100011) = 1$	$f(110011) = 1$
$f(000100) = 0$	$f(010100) = 1$	$f(100100) = 0$	$f(110100) = 0$
$f(000101) = 0$	$f(010101) = 1$	$f(100101) = 0$	$f(110101) = 1$
$f(000110) = 0$	$f(010110) = 1$	$f(100110) = 1$	$f(110110) = 0$
$f(000111) = 0$	$f(010111) = 1$	$f(100111) = 1$	$f(110111) = 1$
$f(001000) = 1$	$f(011000) = 0$	$f(101000) = 0$	$f(111000) = 0$
$f(001001) = 1$	$f(011001) = 0$	$f(101001) = 0$	$f(111001) = 1$
$f(001010) = 1$	$f(011010) = 0$	$f(101010) = 1$	$f(111010) = 0$
$f(001011) = 1$	$f(011011) = 0$	$f(101011) = 1$	$f(111011) = 1$
$f(001100) = 1$	$f(011100) = 1$	$f(101100) = 0$	$f(111100) = 0$
$f(001101) = 1$	$f(011101) = 1$	$f(101101) = 0$	$f(111101) = 1$
$f(001110) = 1$	$f(011110) = 1$	$f(101110) = 1$	$f(111110) = 0$
$f(001111) = 1$	$f(011111) = 1$	$f(101111) = 1$	$f(111111) = 1$

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 энергонезависимой 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 Проектирование контроллера S29AL032D

Контроллер, как мы уже знаем, предназначен для обмена информацией с внешними цифровыми устройствами. Он должен предоставлять удобный и простой интерфейс внутри нашего цифрового устройства и обеспечивать все необходимые взаимодействия с внешним устройством, избавляя остальные блоки от необходимости реализовывать эти функции.

Как должен работать наш контроллер и как он управляться?

Если мы хотим работать с памятью, то для нас наиболее важными являются операции записи и чтения данных. Тогда наиболее удобным для нас был бы уже знакомый интерфейс, похожий на RAM-память: данные для записи, данные для чтения, адрес и управляющие сигналы.

Для того чтобы начать реализовывать контроллер, нам нужно изучить операции записи и чтения в микросхемы S29AL032D

2.3.2 Операция чтения

Для чтения данных из микросхемы 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

Как мы уже знаем, единственным источником информации о времени для цифрового устройства может являться только сигнал синхронизации, заранее известной частоты. Привяжем времена, упомянутые в Таблице 2.1 к тактовому сигналу частоты 50 МГц, которым тактируется устройство. При этом учтём, что некоторые задержки могут быть равны нулю. Полученная временная диаграмма показана на Рис. ??

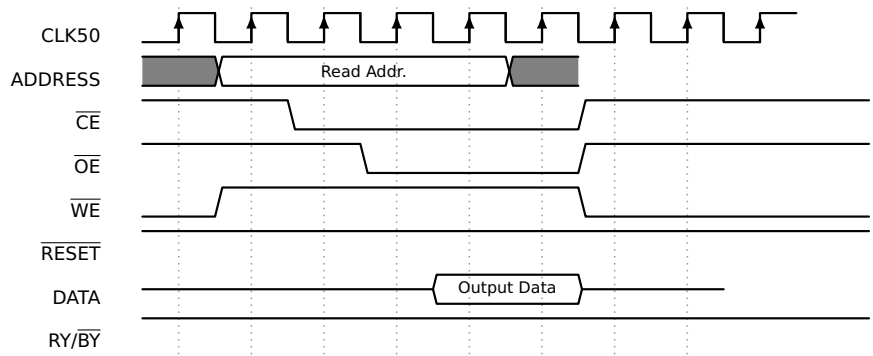


Рис. 2.1: Временная диаграмма операции чтения шины PCI

2.3.3 Операция записи

Обычно запись во flash-память - более сложная операция, чем чтение. Многие производители защищают чипы flash-памяти от случайной записи и используют для записи специальные последовательности команд. Согласно datasheet S29AL032D (разделы 7 и 11) для того, чтобы записать данные необходимо выполнить следующую последовательность из 4-х операций:

- Записать данные AA по адресу AAA;
- Записать данные 55 по адресу 555;
- Записать данные A0 по адресу AAA;
- Записать нужные данные по указанному адресу.

Согласно datasheet данные записываются не мгновенно. На то, чтобы провести операцию записи одного слова требуется порядка 11 мкс.

Также крайне важно, что при записи данных микросхема S29AL032D может менять значение с «1» на «0».

Чтобы поменять значение с «0» на «1» требуется очистка целого фрагмента памяти, называемого сектором, либо полная очистка всей микросхемы!

Для того, чтобы мы могли полноценно пользоваться микросхемой S29AL032D нам потребуется реализовать в контроллере функции очистки, поэтому изучим как происходят эти операции в данной микросхеме.

2.3.4 Операция очистки

Для очистки выбранного сектора необходимо выполнить следующую последовательность операций:

- Записать данные AA по адресу AAA;
- Записать данные 55 по адресу 555;
- Записать данные 80 по адресу AAA;
- Записать данные AA по адресу AAA;
- Записать данные 55 по адресу 555;
- Записать данные 30 по адресу сектора, который необходимо очистить.

Операция очистки сектора занимает существенное время,

и пока она не закончится, невозможно произвести запись или чтение из flash-памяти.

В datasheet на S29AL032D приведены следующие значения:

Очистка сектора - до NN мкс.

Полная очистка микросхемы - до NNN мкс.

2.3.5 Статус операции

Для того, чтобы контролировать завершение операций записи и очистки, а также отслеживать ошибки, которые могут возникнуть в процессе их выполнения необходимо получить информацию о статусе операции. Способы получения этой информации и её описание приведены в разделе 12 datasheet. Здесь мы отметим наиболее важные для нас моменты.

Так как на отладочном стенде Altera DE1, которым мы пользуемся для проведения лабораторных работ, не разведён сигнал BUSY S29AL032D, то единственным способом получения статуса является чтение информации из адреса, по которому производилась запись.

Если операция удачно завершена - то будет получено значение, из указанного адреса. Если проходила операция записи - то значение должно совпадать с тем, которое мы хотели записать, а если происходила операция очистки, то полученное значение должно содержать только единицы (8'hFF).

Если операция ещё не завершена, то во время исполнения встроенного алгоритма биты [7:2] будут содержать информацию о статусе операции.

До окончания операции записи DQ[7] будет иметь значение противоположное записываемому (инверсия при записи, или «0» при очистке) - это основной признак того, что полученные данные отражают статус операции. Информация о битах статусного пакета приведена в Таблице ??

Обратите внимание, что при повторном чтении некоторые биты статусного пакета меняют своё значение на противоположное. Это сделано, для дополнительно индикации того, что в данный момент выполняется операция.

В информации о статусе операции есть важный признак: бит DQ[4] является признаком того, что время операции пре-

высило максимально допустимое. Если этот бит принимает значение «1», то во время операции произошла какая-то ошибка. В подавляющем большинстве случаев это случается в случае записи в ячейку памяти, уже содержащую какое-то значение.

2.3.6 Проектирование контроллера Flash

Теперь, когда мы познакомились с операциями, которые предстоит выполнять контроллеру, мы можем продолжить его проектирование.

От контроллера потребуется проводить различные операции, последовательно отправлять посылки, читать и анализировать информацию о Статусе операции. Значит в качестве его основы следует применить конечный автомат. Ведь именно конечный автомат позволяет нам разделить режимы работы и реализовать последовательности действий в цифровых устройствах.

Начнём проектировать конечный автомат с начального состояния - состояния бездействия. Будем постепенно наращивать его сложность и степень детализации, уточняя некоторые особенности.

Из состояния бездействия возможны три различных перехода: операция чтения, операция записи и операция очистки.

Теперь выделим основные этапы, которые присутствуют в этих операциях. Прежде всего нас интересуют сложные операции «запись» и «очистка».

Как уже говорилось, чтобы провести запись требуется провести 4 обмена с flash-памятью. Но на этом нельзя заканчивать операцию, ведь необходимо дождаться окончания записи. К тому же во время записи могут возникнуть ошибки.

Как мы уже говорили для контроля статуса операции нам нужно считать данные из адреса, по которому производится запись и проанализировать их. Отразим это в состояниях конечного автомата. Нам также нужно учесть, что может произойти ошибка.

Раньше мы не разделяли эти состояния и всё вместе называли «запись». Но после уточнения Того, что на самом деле

происходит в устройстве мы разбили сложную операцию на более простые этапы.

Этапы операции очистки будут выглядеть абсолютно идентично. Это может означать, что некоторые решения можно спроектировать таким образом, что получится их использовать в различных ветках нашего конечного автомата.

Первое, что бросается в глаза - многократное повторение операций записи и чтения (которая используется при проверке статуса). Чтобы выполнить эти операции нужно развернуть временные диаграммы, которые мы получили на Рис. ?? и Рис. ??, соответственно. Нам хотелось бы выделить эти операции для того, чтобы упростить наш конечный автомат. Тогда можно будет представить его следующим образом:

Также можно постараться выделить чтение и анализ статуса в отдельные состояния, общие для операций записи и очистки.

Тогда структура конечного автомата приобретает следующий вид:

В состояниях W и E происходит запись значения во flash-память. В состояниях ST и R происходит чтение значения.

Можем ли мы выделить операции чтения и записи и реализовать их отдельно, чтобы затем использовать их как показано на графе переходов?

Чтобы понять это, сначала ответим на вопрос как вообще возможно реализовать эти операции в контроллере.

Для того, чтобы провести чтение, необходимо развернуть временную диаграмму, показанную на Рис. ?. Тоже относится к записи: временную диаграмму записи, привязанную к тактовому сигналу, мы получили на Рис. ?

Как мы уже обсуждали, схема которую можно использовать для разделения событий во времени - это конечный автомат. Например, с его помощью можно реализовать операцию чтения следующим образом: разобьём состояние «чтение» на этапы, которые соответствуют этапам на временной диаграмме, как представлено на Рис. ?

Мы могли бы добавить эти состояния в конечный автомат, который мы уже начали проектировать, но тогда нам пришлось бы каждое состояние чтения разбить на несколько со-

стояний, тоже самое пришлось бы сделать с записью.

Вместо этого мы можем сделать отдельные небольшие модули, которые будут выполнять эти операции, поместить автоматы в них. Например для операции чтения такой модуль будет управляться автоматом, представленным на Рис.??

Сигналом запуска для таких мини-автоматов будет признак того, что основной автомат находится в состоянии «чтение» или «запись».

Состояние «завершено» нужно для того, чтобы выработать сигнал окончания работы. Иначе «большой» автомат не будет иметь возможности «узнать» о том, что операция завершена и можно переходить в следующее состояние.

Теперь, когда мы оформили все основные идеи и общую структуру контроллера, можно преступить к его реализации на Verilog HDL.

Как всегда, начнём проектировать с интерфейса будущего контроллера - его входов и выходов. Так как контроллер будет обеспечивать доступ к памяти, мы хотели сделать его интерфейс похожим на интерфейс RAM-памяти. Но нам придётся ввести дополнительные сигналы для того, чтобы реализовать операцию очистки и индикацию ошибок.

Нам будет достаточно одного входа для адреса, так как мы не можем одновременно производить чтение и запись во flash-память.

Теперь опишем основной управляющий конечный автомат и мини-автомат чтения. Мини-автомат записи опишите самостоятельно.

Наладим связь между автоматами. Для этого определим управляющие сигналы (воздействия):

Начнём описывать исполняющую логику, которая будет задействована в различных состояниях: