**Лабораторная работа №1**

**Введение в Verilog. Логические вентили. Функции алгебры логики. Дешифраторы. Мультиплексоры.**

Наш курс называется «Проектирование цифровых устройств на Verilog», но что такое «цифровое устройство»? Что такое Verilog?

Цифровые устройства — это устройства, предназначенные для приёма и обработки цифровых сигналов, то есть таких сигналов, которые можно рассматривать в виде набора дискретных уровней.

Цифровые устройства стремительно развиваются с момента изобретения электронной лампы, а затем транзистора. Со временем цифровые устройства стали компактнее, уменьшилось их энергопотребление. Настолько же разительно возросла сложность их структуры и их возможности.

С ростом сложности цифровых устройств, графические схемы, которые применялись для проектирования, уже не могли эффективно использоваться. Потребовался новый инструмент разработки и таким инструментом стали языки описания аппаратной части цифровых устройств, которые описывали структуру формализованным языком, чем-то похожим на язык программирования.

Совершенно новый подход к описанию цифровых схем, реализованный в языках описания цифровой аппаратуры, заключается в том, что с помощью языка описания можно описывать не только структуру, но и поведение цифрового устройства. Структура в таком случае получается в результате обработки описания специальной программой — синтезатором. Этот подход существенно изменил процесс разработки цифровых устройств, превратив громоздкие, тяжело читаемые схемы в относительно простые и доступные описания поведения.

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

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

**Операция «И» (AND, · )**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

**Операция «ИЛИ» (OR, + )**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

**Операция «Исключающее ИЛИ» (XOR, , сложение по модулю 2)**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

**Операция «НЕ» (NOT,)**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 1 |
| 1 | 0 |

Теперь опишем логические вентили, реализующие эти функции на языке Verilog.

Код Verilog выделяется моноширинным шрифтом Consolans. Здесь и далее в коде описания цифровых устройств жирным шрифтом отмечены ключевые слова и символы языка Verilog.

Начнём с описания логического вентиля, выполняющего операцию «И»:



Описанный выше модуль можно представить как некоторый «ящик», в который входит 2 провода с названиями *x* и *y* 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.

Аналогично опишем все остальные логические вентили.

Вентиль «ИЛИ»:

|  |
| --- |
| **module** or\_gate(  **input** x,  **input** y,  **output** result);  **assign** result = x | y;  **endmodule**; |

Вентиль «Исключающее ИЛИ»:

|  |
| --- |
| **module** xor\_gate(  **input** x,  **input** y,  **output** result);  **assign** result = x ^ y;  **endmodule**; |

Вентиль «НЕ»:

|  |
| --- |
| **module** not\_gate(  **input** x,  **output** result);  **assign** result = ~x;  **endmodule**; |

В поведенческом описании цифровых устройств логические вентили наиболее часто используются для формулирования и проверки сложных условий, например:

|  |
| --- |
| **if** ((a&b)|(~c)) **begin**     …  **end**; |

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** or\_bus(  **input** [7:0]x,  **input** [7:0]y,  **output** [7:0]result);  **assign** result = x | y;  **endmodule**; |

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*.

Проиллюстрируем это описание рисунком схемы, ему соответствующей:



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

|  |
| --- |
| **module** or\_bus(  **input** [7:0]x,  **output** [4:0]a,  **output** b,  **output** [2:0] c);  **assign** a = x[6:2];  **assign** b = x[5] | x[7];  **assign** c = x[7:5] ^ x[2:0];  **endmodule**; |

Такому описанию соответствует следующая схема:



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 придется 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. 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:

|  |
| --- |
| **module** function(  **input** x0,  **input** x1,  **input** x2,  **output reg** y);  **wire** [2:0]x\_bus;  **assign** x\_bus = {x2, x1, x0};  **always @**(xbus) **begin**  **case** (xbus)      3’b000: y <= 1’b0;     3’b010: y <= 1’b0;     3’b101: y <= 1’b0;     3’b110: y <= 1’b0;     3’b111: y <= 1’b0;  **default**: y <= 1’b1;  **endcase;**  **end;**  **endmodule**; |

Описание, приведенное 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 трехбитной шины *x\_bus.*

После создания шины *x\_bus,* она подключается к объединению проводов *x2* *x1* и *x0*.



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 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 указывается формат (b – бинарный, двоичный / h – шестнадцатеричный / d – десятичный) и сразу же за ним значение числа.

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:

|  |
| --- |
| **module** decoder(  **input** [2:0] x,  **output** [3:0]y);  **reg** [3:0] decoder\_output;  **always @**(x) **begin**  **case** (x)     3’b000: decoder\_output <= 4’b0100;     3’b001: decoder\_output <= 4’b1010;     3’b010: decoder\_output <= 4’b0111;     3’b011: decoder\_output <= 4’b1100;     3’b100: decoder\_output <= 4’b1001;     3’b101: decoder\_output <= 4’b1101;     3’b110: decoder\_output <= 4’b0000;     3’b111: decoder\_output <= 4’b0010;  **endcase;**  **end;**  **assign** y = decoder\_output;  **endmodule**; |

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]);*

Но, если мы посмотрим на только что описанную конструкцию 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**.

В цифровой схемотехнике принято обозначать дешифратор следующим образом:



Заметим, что длины векторов не 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 однин интересный набор ФАЛ:

|  |
| --- |
| **module** decoder(  **input** [2:0] a,  **input** [2:0] b,  **input** [2:0] c,  **input** [2:0] d,  **input** [1:0] s,  **output reg** [2:0]y);  **always** **@**(a,b,c,d,s) **begin**  **case** (s)     3’b00: y <= a;     3’b01: y <= b;     3’b10: y <= c;     3’b11: y <= d;  **endcase;**  **end;**  **endmodule**; |

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.

Ниже показано как изображается мультиплексор на рисунках цифровых сем:



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:

|  |  |  |  |
| --- | --- | --- | --- |
| *f*(00**0**000) = 0  *f*(00**0**001) = 0  *f*(00**0**010) = 0  *f*(00**0**011) = 0  *f*(00**0**100) = 0  *f*(00**0**101) = 0  *f*(00**0**110) = 0  *f*(00**0**111) = 0  *f*(00**1**000) = 1  *f*(00**1**001) = 1  *f*(00**1**010) = 1  *f*(00**1**011) = 1  *f*(00**1**100) = 1  *f*(00**1**101) = 1  *f*(00**1**110) = 1  *f*(00**1**111) = 1 | *f*(010**0**00) = 0  *f*(010**0**01) = 0  *f*(010**0**10) = 0  *f*(010**0**11) = 0  *f*(010**1**00) = 1  *f*(010**1**01) = 1  *f*(010**1**10) = 1  *f*(010**1**11) = 1  *f*(011**0**00) = 0  *f*(011**0**01) = 0  *f*(011**0**10) = 0  *f*(011**0**11) = 0  *f*(011**1**00) = 1  *f*(011**1**01) = 1  *f*(011**1**10) = 1  *f*(011**1**11) = 1 | *f*(1000**0**0) = 0  *f*(1000**0**1) = 0  *f*(1000**1**0) = 1  *f*(1000**1**1) = 1  *f*(1001**0**0) = 0  *f*(1001**0**1) = 0  *f*(1001**1**0) = 1  *f*(1001**1**1) = 1  *f*(1010**0**0) = 0  *f*(1010**0**1) = 0  *f*(1010**1**0) = 1  *f*(1010**1**1) = 1  *f*(1011**0**0) = 0  *f*(1011**0**1) = 0  *f*(1011**1**0) = 1  *f*(1011**1**1) = 1 | *f*(11000**0**) = 0  *f*(11000**1**) = 1  *f*(11001**0**) = 0  *f*(11001**1**) = 1  *f*(11010**0**) = 0  *f*(11010**1**) = 1  *f*(11011**0**) = 0  *f*(11011**1**) = 1  *f*(11100**0**) = 0  *f*(11100**1**) = 1  *f*(11101**0**) = 0  *f*(11101**1**) = 1  *f*(11110**0**) = 0  *f*(11110**1**) = 1  *f*(11111**0**) = 0  *f*(11111**1**) = 1 |

**Оператор case в Verilog используется ТОЛЬКО для описания дешифраторов, мультиплексоров и таблично заданных ФАЛ.**

**Задание лабораторной работы:**

Описать на языке Verilog цифровое устройство, функционирующее согласно следующим принципам:

1. Ввод информации происходит с переключателей SW
2. SW[3:0] должны обрабатываться дешифратором «DC1», согласно индивидуальному заданию
3. SW[7:4] должны обрабатываться дешифратором «DC2», согласно индивидуальному заданию
4. Реализовать дешифратор «DC-DEC», преобразующий число, представленное в двоичном коде в цифру, отображаемую на семи сегментном индикаторе. Руководствоваться при этом нужно следующими соображениями:  
    - Семи сегментный индикатор подключается к шине HEX0[6:0]  
    - Диоды на семи сегментном индикаторе загораются при подаче на них низкого напряжения (0 - горит, 1 - не горит)  
    - Картинка соответствия проводников в шине диодам индикатора:
5. С помощью мультиплексора реализовать следующую схему подключения:  
   Если SW[9:8] = 00, на дешифратор DC-DEC поступает выход DC1  
   Если SW[9:8] = 01, на дешифратор DC-DEC поступает выход DC2  
   Если SW[9:8] = 10, на дешифратор DC-DEC поступает выход  
   логической функции f  
   Если SW[9:8] = 11, на дешифратор DC-DEC поступает SW[3:0]

Выполнив описание модуля на языке Verilog необходимо построить временные диаграммы его работы с помощью САПР Altera Quartus.

Привязать входы модуля к переключателям SW, отладочной платы, а выход к шине HEX0[6:0], получить прошивку для ПЛИС и продемонстрировать её работу.