# ЛАБОРАТОРНАЯ РАБОТА №2

# ПРОЕКТИРОВАНИЯ КОМБИНАЦИОННЫХ И ПОСЛЕДОВАТЕЛЬНЫХ СХЕМ НА ПЛИС

## Цель работы

Получение базовых знаний и навыков работы с языком описания аппаратуры Verilog для проектирования комбинационных и последовательных схем на ПЛИС.

## Теоретическая часть

### Комбинационные и последовательные схемы

Цифровые схемы разделяются на комбинационные (combinational) и последовательные (sequential). Выходы *комбинационных схем* зависят только от текущих значений на входах; другими словами, такие схемы комбинирует текущее значение входных сигналов для вычисления значения на выходе. Поведение комбинационной логической схемы можно определить с помощью набора функций вывода. Например, логический элемент – это комбинационная схема.

Комбинационные логические элементы часто группируются в «строительные блоки», используемые для создания сложных систем. Это позволяет абстрагироваться от излишней детализации уровня логических элементов и подчеркнуть функцию «строительного блока». Мультиплексоры и дешифраторы являются часто используемыми блоками.

Схема является комбинационной, если она состоит из соединенных между собой элементов и выполнены следующие условия:

- Каждый элемент схемы сам является комбинационным;
- Каждое соединение схемы является или входом, или подсоединено к одному единственному выходу другого элемента схемы;
- Схема не содержит циклических путей: каждый путь в схеме проходит через любое соединение не более одного раза.

Логическая схема, выходное значение которой зависит от входных значений, а также от хранимой информации, называется *последовательной* логической схемой. Следовательно, последовательные логические схемы обладают памятью. Основным условием для построения последовательных схем является присутствие обратной связи.

К числу наиболее распространённых последовательных схем, которые используются, в частности, в составе аппаратных средств, относятся триггеры, а также реализуемые на их основе регистры и счётчики.

**Verilog для комбинационных и последовательных схем**

Для проектирования комбинационных логических схем можно использовать все три уровня абстракции в Verilog. А для проектирования последовательных схем оптимальным является уровень абстракции Behavioral. Потому что, тип данных *reg* для хранения значений используется вместе с блоком *always*, который, как мы знаем, используется при моделировании на *поведенческом* уровне.

Во время проектирования на уровне *gate-level* должно быть известно внутренняя структура комбинационной схемы. Это значит, нужно знать, на основе каких логических элементов оно было создано. Этот уровень абстракции подходить для проектирования небольших комбинационных схем.

Уровень *data flow* акцентируется на использовании сигналов, связывающих элементы комбинационной логики. Во время проектирования будут использоваться операторы побитовые, логические и тернарные. Первые два вида операторов понятны в использовании. А вот в использовании тернарного оператора есть свои нюансы. Тернарный оператор является оператором использующий три операнда. Синтаксис тернарного оператора:

$$condition\ ?\ expression1 : expression2$$

$condition$ – это место для логического выражения. Если ответом выражение является *true,* выполняется первое выражения *(*$expression1$*),* а если *false,* то соответственно второе выражения *(*$expression1$*).* Пример использования тернарного оператора:

$$y = (x > 10)\ ?\ 1 : 0;$$

Если значение *x* будет больше 10, то *y* будет равно 1, а если *x* будет меньше 10, то *y* присвоится значение 0.

*Поведенческий* уровень дает возможность проектирования и комбинационных и последовательных логических схем. Операции в поведенческом проектировании концентрируются в блоке *always.* Каждый из блоков *always* предоставляет собой отдельный поток обработки данных, и все они выполняются параллельно.

При работе с простыми логическими элементами входные/ выходные значения сигналов являются одноразрядными. Но при работе с комбинационными схемами использование одноразрядных (скалярных) сигналов является недостаточным. Для решения этого используется многоразрядные сигналы (шины), называемым вектором (*vector*). *Vector* является набором скалярных сигналов. Например, если предоставить 5 разрядный сигнал $w$, его можно объявить, как

$$wire\ [4{:}\,0]\ w;$$

Это определяет $w$ как набор из пяти сигналов w[4], w[3], w[2], w[1] и w[0], каждая из которых является однобитным. Когда мы объявляем вектор или порт, часть в скобках (4: 0 в приведенном выше примере) задает диапазон индекса для элементов вектора. Первое значение - это индекс самого левого элемента, а второе значение - это индекс самого правого элемента.

Проектирования схем на поведенческом уровне можно представить как составления алгоритма работы устройства. А составлений алгоритма подразумевает частое принятие решений на основе определенных условий. И, при проектировании комбинационных и последовательных схем на основе ее таблицы истинности определяется, какое значение получится на выходе. Для реализации условий в Verilog используется условные операторы *if* и *case.* Синтаксис оператора *if:*

- Первый тип условного оператора без блока *else.* Оператор выполняется или нет

  $$if\ (< expression >)\ true\_statement\ ;$$

- Второй тип условного оператора с блоком else. Оценивается либо утверждение истинное, либо ложное

  $$if\ (< expression >)\ true\_statement;$$
  $$else\ false\_statement\ ;$$

- Третий вид условного оператора, вложенные *if-else-if*. Выбор из нескольких операторов, но исполняется только один из них

  $$if\ (< expression1 >)\ true\_statement1\ ;$$
  $$else\ if\ (< expression2 >)\ true\_statement2\ ;$$
  $$else\ if\ (< expression3 >)\ true\_statement3\ ;$$
  $$\ldots$$
  $$\ldots$$
  $$else\ default\_statement\ ;$$

Выражения $< expression >$ оценивается, если оно будет истинным (1 или ненулевое значение), то выполнится $true\_statement$. Однако, если это ложное (ноль), то выполняется $false\_statement$. $true\_statement$ или $false\_statement$ может быть одним оператором или блоком из нескольких выражений. Блок из нескольких выражений должен быть сгруппирован, как правило, с использованием ключевых слов *begin* и *end.*

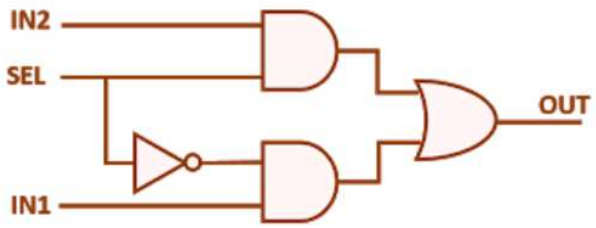
Вложенные *if-else-if* может быть громоздким, если существует много вариантов. Кратчайший вариант достижение того же результата является использование оператора *case.* Синтаксис оператора *case:*

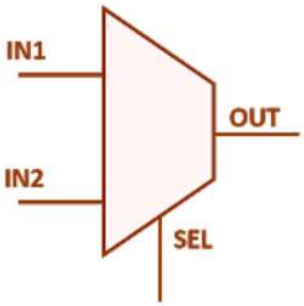
$case\ (expression)$
$alternative1: statement1;$
$alternative2: statement2;$
$alternative3: statement3;$
$\ldots$
$\ldots$
$default: default\_statement;$
$endcase$

Каждый из $statement$ может быть отдельным выражением или блоком нескольких выражений. Блок из нескольких выражений должен быть сгруппирован, как правило, с использованием ключевых слов *begin* и *end.* Выражение $expression$ сравнивается с альтернативами в их порядке написания. Если ни одна из альтернатив не совпадает, выполняется $default\_statement$. $default\_statement$ является необязательным.

**Проектирование схем в Verilog**

Для примера комбинационной схемы возьмем схему мультиплексора 2x1 и ее таблицу истинности (рис. 1).




| SEL | OUT |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Рисунок 1. схема мультиплексора и ее таблица истинности**

Проектирования схемы на уровне:

***Gate level***

```
module logic(
    output OUT,
    input IN1,
    input IN2,
    input SEL
    );

     wire w1, w2, w3;
```

```
        not (w1, SEL);
        and (w2, SEL, in2);
        and (w3, w1, in1);

        or (OUT, w2, w3);
    endmodule
```

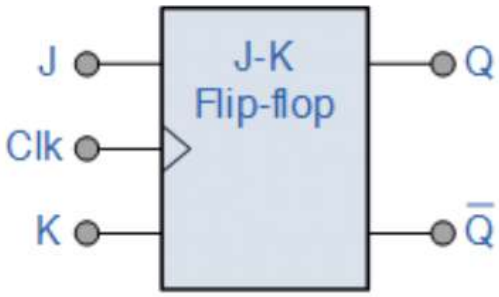
***Data flow***

```
    module logic(
        output OUT,
        input IN1,
        input IN2,
        input SEL
        );
        assign OUT = (~SEL & IN1) | (SEL & IN2);
        //ИЛИ
        assign OUT = (SEL == 1) ? IN2 : IN1;
    endmodule
```

***Behavioral***

```
    module logic(
        output reg OUT,
        input IN1,
        input IN2,
        input SEL
        );
        always@(SEL or IN1 or IN2)
        begin
            case(SEL)
            1'b0 : OUT = IN1;
            1'b1 : OUT = IN2;
            endcase
        end
    endmodule
```

Для примера последовательной схемы возьмем схему JK триггера и ее таблицу истинности (рис. 2).




| J | K | CLK | $Q_t$ |
|---|---|---|---|
| 0 | 0 | ↑ | $Q_{t-1}$ |
| 0 | 1 | ↑ | 0 |
| 1 | 0 | ↑ | 1 |
| 1 | 1 | ↑ | $\overline{Q}_{t-1}$ |

**Рисунок 2. JK триггер ее таблица истинности**

Проектирования схемы на уровне:

***Behavioral***

```
module logic(
    output reg Q,
    input J,
    input K,
    input CLK
    );
     initial
     begin
          Q = 1'b0;
     end

    always@(posedge CLK)
    begin
          case({J, K})
          2'b00 : Q = Q;
          2'b01 : Q = 1'b0;
          2'b10 : Q = 1'b1;
          2'b11 : Q = ~Q;
        endcase
    end
endmodule
```

В программе, порт *CLK* обеспечивает тактирования схемы и является синхронным входом. Ключевое слово *posedge* фиксирует передний фронт сигнала. В нашем случае блок *always* включается каждый раз, когда фиксирует передний фронт тактирующего сигнала *CLK*.

**Практическая часть**

**Задания №1.**

1. Открыть программу ISE Design Suite;
2. Создать новый проект **File → New Project;**
3. Добавить в проект рабочий файл **Project →New Source→Verilog Module;**
4. Спроектировать схему дешифратора 3x8 использовав один из уровнь абстракции на выбор*;*
5. Синтезировать проект, запустив проверку **Synthesize-XST;**
6. Открыть подпрограмму для назначения реальных входов/выходов **Tools→PlanAhead→I/O Pin Planning (PlanAhead) – Post-Synthesis…**;
7. Запустить проверку **Implement Design;**
8. При успешных двух предыдущих проверках запустить генерацию исполняемого файла **Generate Programming File;**
9. Открыть подпрограмму для записи сгенерированного файла в ПЛИС **Tools→iMPACT…**;
10. Подключить ПЛИС к компьютеру через USB;
11. Запрограммировать ПЛИС.

**Задания №2**

Спроектировать схему демультиплексора 1x4.

**Задания №3**

Спроектировать схему дешифратора предназначенного для семисегментного элемента.

**Задания №4**

Написать программу работы триггеров RS, T и D.