2.6. Структуры данных Verilog

При определении типа объектов данных в Verilog-программе (переменные, сигналы и пр.) следует учитывать способ их последующей интерпретации в электронных устройствах. Язык Verilog имеет два основных типа сигналов – цепи и регистры. Их главное отличие состоит в том, что цепь при отключении драйвера переходит в неопределенное состояние. Напомним, что драйвером называется устройство (модуль) или процесс (блок поведенческих операторов), содержащий операторы, формирующие и изменяющие значение соответствующего сигнала. Как цепи, так и регистры представляют собой двоичные векторы, каждый из разрядов которых может принимать одно из четырех значений:

`0`	Логический ноль (булева «Ложь»)
`1`	Логическая единица (булева «Истина»)
`x`	Неизвестное значение
`z`	Высокий импеданс

Кроме того, для разрешения конфликтов между драйверами, присоединенными к общей шине, допускается указание уровня приоритета конкретного сигнала. Возможные уровни перечислены ниже в порядке убывания приоритета:

```
supply, strong, pull,
large, weak, medium,
small, highz
```

Если к одной шине подключены два сигнала с различным уровнем приоритета, то в соответствии с правилами перекрытия шина принимает значение, соответствующее сигналу с большим приоритетом. Если же к одной шине подключить два сигнала с равными приоритетами, то результат примет значение x, т.е. неизвестный. Приоритеты сигналов используются для задач моделирования реальных процессов и не влияют на синтез логической цепи на основе Verilog-кода.

2.6.1. Цепи

Сигналы типа «цепь» обычно используются для соединения отдельных устройств или модулей (структурных компонентов) внутри проекта. Наиболее распространенный тип цепей определяется ключевым словом **wire**. Для более адекватного моделирования процессов перекрытия сигналов возможно также использование отличающихся уровнем приоритета цепей типа **wand, wor, tri, triand, trior, trireg** и др. Аппаратно цепь можно представить как одноразрядную линию связи, соединяющую проектируемое устройство с внешним оборудованием либо соединяющую отдельные структурные компоненты внутри устройства. По умолчанию цепь принимает значение z (за исключением цепей **trireg**, принимающих по умолчанию значение x). Как уже отмечалось ранее, при отключении своего драйвера цепь принимает значение x (неизвестный сигнал).

Примеры создания цепей типа **wire** показаны ниже:

```
wire a;                    // Создана цепь a
wire X1,X2;     // Созданы цепи X1, X2
wire Q = 1b'0;// Создана цепь Q с фиксированным значением 0
```

Как видно из этих примеров, одним оператором **wire** можно сформировать несколько сигналов (вторая строка, сигналы *X1*, *X2*), сформированному сигналу *Q* придать фиксированное значение *Q* = 0 (третья строка).

Следует помнить, что при попытке изменения в других местах Verilog-программ сигналов с предварительно заданными фиксированными значениями будут возникать конфликты, разрешение которых будет осуществляться в соответствии с вышеупомянутыми правилами перекрытия.

Если в дальнейшем в рассматриваемом примере попытаться присвоить сигналу *Q* значение 1, то произойдет наложение логического нуля и логической единицы от различных источников (драйверов), в результате которого сигнал *Q* получит значение x, т.е. неизвестный.

2.6.2. Регистры

Регистры представляют собой элементы, способные сохранять данные при отключении их источника. Не следует путать термин «регистр», обозначающий тип сигнала в языке Verilog, и электронное устройство с аналогичным названием. Verilog-регистр, в отличие от соответствующего устройства, не требует наличия тактирующего сигнала и может изменить значение в любой момент при поступлении новых данных. Таким образом, регистр в Verilog просто обозначает сигнал, не требующий драйвера для поддержки своего значения, и может быть интерпретирован средствами логического синтеза при аппаратной реализации несколькими различными способами (как триггер или как набор триггеров различных типов, как собственно регистр и т.д.).

Для создания регистра общего назначения используется ключевое слово **reg**:

```
reg W;
reg a, b; // создание нескольких регистров одним оператором
```

Регистр, так же, как и цепь, представляет собой одноразрядный двоичный сигнал.

Следует обратить внимание, что присваивание фиксированных значений при создании регистра (как в случае с цепями) не допускается. Это связано с особенностями работы непрерывного оператора присваивания и будет рассмотрено ниже в соответствующем разделе.

2.6.3. Векторы

Цепи и регистры могут группироваться в шины, называемые в языке Verilog векторами. Вектор не следует отождествлять с массивом, так как, во-первых, каждый элемент вектора может иметь длину не более 1 бита и, во-вторых, вектор можно рассматривать как целый объект данных, так и в качестве набора отдельных элементов (по аналогии с массивами).

Векторы регистров и цепей создаются путем введения индекса элементов вектора в оператор объявления сигнала непосредственно перед его идентификатором:

```
тип_сигнала [левый_бит : правый_бит] идентификатор;
```

например:

```
module vectors ( );          //Заголовок программы

reg [7:0] W, S;//создание вектора регистров (7-й бит – старший)
reg [0:7] K;         //создание вектора регистров (0-й бит – старший)
wire [7:0] S; //создание вектора цепей (7-й бит – старший)

initial begin

W = 8'b0100_0000;
K = 8'b0000_0010;
W[0] = 1'b1; // Обращение к элементу вектора W

S = W + K; // Обращение к векторам как к единым сигналам

end

endmodule                    // Конец программы
```

В приведенном примере проиллюстрирован процесс создания векторов из цепей и регистров, а также использования векторов в виде их отдельных элементов и в виде единого целого.

2.6.4. Целые числа

В языке Verilog предусмотрены специальные возможности для интерпретации регистров как целых или действительных чисел, а также для хранения информации о модельном времени.

Любую информацию можно хранить в регистрах общего назначения (например, типа **reg**), однако более удобным представляется использование для хранения целых чисел регистров специального типа **integer**. Кроме того, регистры типа **integer** (в отличие от регистров **reg**) позволяют оперировать с отрицательными числами, которые автоматически транслируются в дополнительный код [8]. Регистры типа **integer** имеют 32 двоичных разряда. Примеры объявления сигналов типа **integer** представлены ниже:

```
integer A;
integer M1, M2;
```

2.6.5. Действительные числа real

Действительные числа в Verilog, как и в VHDL, используются только при поведенческом моделировании электронных устройств и не поддерживаются средствами синтеза логических цепей. Однако для упрощения и ускорения разработки моделей высокого уровня абстракции, а также для составления кода тех частей проекта, которые не требуют решения задачи синтеза (например, генераторы тестовых последовательностей в испытательных стендах), наличие в стандарте языка Verilog действительных чисел и средств их обработки является очень полезным.

Регистры действительного типа и действительные константы создаются при помощи ключевого слова **real**:

```
real A; //Создан регистр действительного типа
real Pi = 3.14; //Создана действительная константа
```

Действительные числа могут задаваться как в формате с фиксированной запятой, так и с плавающей запятой:

```
A = 83.375          //Формат с фиксированной запятой
B = 1.2e-5          //Формат с плавающей запятой
C = 4.13e10   //Формат с плавающей запятой
```

Явное указание выделяемого для действительного сигнала количества разрядов в языке Verilog не поддерживается.

Присваивание целочисленному регистру действительного значения приводит к округлению последнего к ближайшему целому числу. Таким образом, язык Verilog, в отличие от таких строго типизированных языков, как VHDL или Ада, допускает неявное преобразование типов данных.

2.6.6. Массивы

В языке Verilog допускается использование массивов, состоящих из элементов типа **reg**, **integer**, **time,** а также из векторов элементов этих же типов. При этом массивы могут быть только одномерными и не могут состоять из действительных чисел. Важно различать между собой массивы и векторы: вектор длиной в *n* бит является одним *n*-битным элементом, а массив такой же длины представляет собой *n* однобитных элементов. Таким образом, массив векторов не является двухмерным массивом, так как обеспечивает доступ только к векторам в целом, а не к их фрагментам.

Массив элементов создается следующим образом:

```
Тип_элемента Идентификатор_массива[Диапазон_индексов]
```

Диапазон индекса массива, так же как и вектора, задается через двоеточие «:» и может быть возрастающим или убывающим. Примеры создания массивов:

```
// Массив из 11 целых чисел:
integer Data[0:10]
```

```
// Массив Byte из 100 восьмиразрядных векторов типа reg:
reg [7:0] Byte [1:100]
```

При обращении к элементу массива его индекс, так же, как и в языке Паскаль, прибавляется к идентификатору массива (справа, в квадратных скобках) следующим образом:

```
Идентификатор_массива[Индекс_элемента]
```

например:

```
integer A [1:20];
reg Data [0:100];

. . .

A[1] = 10;//Обращение к 1-му элементу массива A
P = Data[8]; //Обращение к 8-му элементу массива Data
```