Глава 5

Массивы, структуры и объединения SystemVerilog

SystemVerilog добавляет несколько расширений в Verilog для представления больших объёмов данных. Конструкция массива расширена как в представлении данных, так и в операциях над ними. Типы структура и объединение добавлены в Verilog для представления группы переменных.

В этой главе представлены:

- Структуры
- Объединения
- Операции над структурами и объединениями
- Неупакованные массивы
- Упакованные массивы
- Операции над массивами
- Цикл foreach для массива
- Системные функции для работы с массивами
- Системная функция $bits “sizeof”

5.1 Структуры

Данные часто представлены логическими группами сигналов, такими как сигналы управления для протокола, или сигналы состояния. Язык Verilog не имеет удобного механизма для сбора сигналов в одну группу. Разработчикам приходится использовать специальные методы группирования, такие как именные соглашения, где каждый сигнал в группе начинается или заканчивается определёнными символами.

Структуры определяются с использованием С – подобного синтаксиса.

SystemVerilog добавляет С – подобные структуры в Verilog. Структура есть удобный способ группирования схожей информации. Структура объявляется с использованием ключевого слова **struct.** Члены структуры могут быть разного типа, включая типы определяемые пользователем и константные типы. Пример объявления структуры:

```
struct {
  int           a, b;     // 32-bit variables
  opcode_t opcode;        // user-defined type
  logic [23:0] address;   // 24-bit variable
  bit           error;    // 1-bit 2-state var.
} Instruction_Word;
```

Нельзя использовать С подобный “tag”.

Синтаксис объявления структуры в SystemVerilog очень похож на синтаксис языка С. Но есть одно отличие, в языке С можно ставить “tag” после ключевого слова struct перед фигурной скобкой, SystemVerilog делать этого не позволяет.

Структуры есть сбор переменных и констант.

Структура это переменные и константы под одним именем. К структуре можно обращаться используя имя структуры. Каждый член структуры имеет имя по которому к нему можно обратиться. Доступ к члену структуры такой же как в языке С.

```
<structure_name>.<variable_name>
```

На пример, чтобы присвоить значение члену структуры:

```
Instruction_Word.address = 32'hF000001E;
```

Структуры отличаются от массивов.

Массивы состоят из элементов с одинаковым типом и размером, структуры состоят из переменных и констант, которые могут быть разного типа и размера. Другое отличие это то что доступ к элементам массива осуществляется через индекс, а к членам структуры через их имена.

5.1.1 Объявления структуры

Переменные или сети (соединения) могут быть определены как структура.

Структуры могут быть переменными или сетями

К структуре можно обращаться как к единому целому, либо к отдельным её членам. Структура как единое целое может быть объявлена с использованием ключевого слова **var**. Структура также может быть определена как сеть, используя любой сетевой тип **wire** или **tri**. Когда определён сетевой тип, все члены структуры должны быть типами с 4-мя состояниями.

```
var struct {                    // structure variable
  logic [31:0] a, b;
  logic [ 7:0] opcode;
  logic [23:0] address;
} Instruction_Word_var;

wire struct {                   // structure net
  logic [31:0] a, b;
  logic [ 7:0] opcode;
  logic [23:0] address;
} Instruction_Word_net;
```

Объявление структуры как тип **var** или **net** опционально. Если тип не указан, то тип структуры **var**.

```
struct {                        // structure variable
  logic [31:0] a, b;
  logic [ 7:0] opcode;
  logic [23:0] address;
} Instruction_Word_var;
```

Обратите внимание, хотя структура как единое целое может быть объявлена как тип net, типы net не могут использоваться внутри структур. Сети, могут быть сгруппированы под одним именем, используя SystemVerilog интерфейсы, которые рассматриваются в главе 10.

Типизированные и анонимные структуры

Структуры могут быть типами определяемые пользователем

Типы, определяемые пользователем, могут быть созданы из структур, используя ключевое слово **typedef**, как обсуждалось в секции 4.1. Объявление структуры как типа определяемого пользователем не выделяет память для хранения данных. Перед тем как значения будут присвоены членам структуры такого типа, переменная типа определяемого пользователем должна быть объявлена.

```
typedef struct {  // structure definition
  logic [31:0] a, b;
  logic [ 7:0] opcode;
  logic [23:0] address;
} instruction_word_t;

instruction_word_t IW;  // structure allocation
```

Когда структура объявлена без использования **typedef**, то к структуре обращаются как к анонимной.

```
struct {
  logic [31:0] a, b;
  logic [ 7:0] opcode;
  logic [23:0] address;
} instruction;
```

Локальные и разделяемые структурные определения.

Структурные определения могут быть разделяемыми, используя пакеты или $unit

Типизированная структура может быть определена внутри модуля или интерфейса, что позволяет использовать её по всей области блока (модуля или интерфейса). Если типизированную структуру необходимо использовать более чем в одном блоке, или как порт модуля или интерфейса, тогда определение структуры следует поместить в пакет и импортировать в блоки или в пространство $unit. Типизированные структуры могут быть определены напрямую в пространстве $unit. Определения в пакетах и в $unit обсуждались в секции 2.1 и 2.2 глава 2.

5.1.2 Присваивание значений структурам

Инициализация структур

Структуры могут быть инициализированы, используя список значений

Члены структуры могут быть инициализированы на момент инстанцирования структуры, используя значения заключённые между символами `'{ }`. Число значений в скобках должно соответствовать числу членов в структуре.

```
typedef struct {
   logic [31:0] a, b;
   logic [ 7:0] opcode;
   logic [23:0] address;
} instruction_word_t;

instruction_word_t IW = '{100, 3, 8'hFF, 0};
```

Похожий синтаксис используется для определения структуры констант или структуры параметров.

Синтаксис списка значений в SystemVerilog не такой как в языке C.

SystemVerilog использует символы '{ } чтобы обозначить список значений, в языке C используются { }. Ранние версии стандарта SystemVerilog использовали для обозначения списка значений скобки { } как в C. Окончательная версия IEEE SystemVerilog изменила обозначение списка значений к '{ }, чтобы отличать список значений от операции конкатенации { }.

Присваивание значений членам структуры

Три способа присвоения структурам

Значение может быть присвоено члену структуры через его имя.

```
typedef struct {
  logic [31:0] a, b;
  logic [ 7:0] opcode;
  logic [23:0] address;
} instr_t;

instr_t IW;

always @(posedge clock, negedge resetN)
  if (!resetN) begin
    IW.a = 100;  // reference structure member
    IW.b = 5;
    IW.opcode = 8'hFF;
    IW.address = 0;
  end
  else begin
    ...
  end
```

Присвоение структурных выражений структурам

Структурное выражение заключается внутри '{ }

Структуре может быть присвоено структурное выражение. Структурное выражение формируется, используя список значений, разделённых запятой и

заключённых в скобки '{ }, также как при инициализации структуры. В скобках должно быть значение для каждого члена структуры.

```
always @(posedge clock, negedge resetN)
  if (!resetN) IW = '{100, 5, 8'hFF, 0};
  else begin
    ...
  end
```

структурное выражение записывается по порядку или через имена членов структуры

Значения в структурном выражении могут быть записаны в том порядке в котором они определены в структуре. Либо, в структурном выражении могут использоваться имена членов структуры, где имя члена структуры и значение разделяются двоеточием. Когда указаны имена членов структуры, список выражения может быть составлен в любом порядке.

```
IW = '{address:0, opcode:8'hFF, a:100, b:5};
```

Нельзя смешивать оба способа в одном структурном выражении.

```
IW = '{address:0, 8'hFF, 100, 5};  // ERROR
```

Значения по умолчанию в структурных выражениях

Некоторым или всем членам структуры могут быть присвоены значения по умолчанию.

Значение по умолчанию может быть присвоено всем членам структуры, используя ключевое слово **default**.

```
IW = '{default:0}; // set all members of IW to 0
```

Значение по умолчанию может быть присвоено членам структуры определённого типа, используя ключевое слово типа. Ключевое слово **default** или ключевое слово типа отделяются от значения двоеточием.

```
typedef struct {
  real          r0, r1;
  int           i0, i1;
```

```
    logic [ 7:0] opcode;
    logic [23:0] address;
  } instruction_word_t;

  instruction_word_t IW;

  always @(posedge clock, negedge resetN)
    if (!resetN)
      IW = '{ real:1.0, default:0 };
      // assign all real members a default of 1.0
      // and all other members a default of 0
    else begin
      ...
    end
```

Значения по умолчанию должны быть совместимы с типами членов структуры (приведение типов).

Существует приоритетность присвоения значений членам структуры. Ключевое слово **default** имеет низший приоритет, далее идёт значение с определённым типом, выше которого значение явно именованного члена структуры. В следующем структурном выражении r0 присваивается значение 1.0, r1 значение 3.1415, всем остальным членам структуры присваивается значение 0.

```
typedef struct {
  real         r0, r1;
  int          i0, i1;
  logic [ 7:0] opcode;
  logic [23:0] address;
} instruction_word_t;

instruction_word_t IW;

    IW = '{ real:1.0, default:0, r1:3.1415 };
```

5.1.3 Упакованные и неупакованные структуры

Неупакованные структуры могут иметь выравнивание

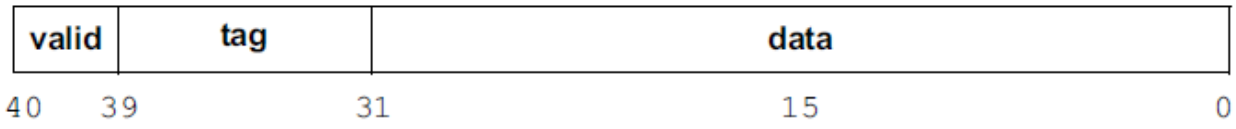
По умолчанию структура не упакована. Это означает что члены структуры рассматриваются как независимые переменные или константы, которые сгруппированы вместе под одним именем. SystemVerilog не определяет как программные инструментам следует хранить члены неупакованной структуры. Способ хранения может варироваться от одного программного инструментария к другому.

Упакованные структуры хранятся без выравнивания

Структура может быть явно объявлена как упакованная, используя ключевое слово **packed**. Упакованная структура хранит все члены как смежные биты в определённом порядке. Упакованная структура хранится как вектор, первый член структуры занимает левое наиболее значащее поле вектора. Самый правый бит последнего члена в структуре есть наименее значимый бит вектора, и нумеруется как нулевой бит. Это показано на рисунке 5-1.

```
struct packed {
  logic        valid;
  logic [ 7:0] tag;
  logic [31:0] data;
} data_word;
```

Figure 5-1: Packed structures are stored as a vector



К членам упакованной структуры можно обращаться по имени или используя часть вектора. Следующие два присвоения назначаются члену tag:

```
data_word.tag = 8'hf0;
data_word[39:32] = 8'hf0;  // same bits as tag
```

Упакованные структуры могут содержать только интегральные значения.

Все члены упакованной структуры должны быть интегральными значениями. Интегральное значение есть значение, которое может быть представлено как вектор, такое как byte, int и вектора, созданные с использованием типов **bit** или **logic.** Структура не может быть упакована, если какой-либо член структуры не может быть представлен как вектор. Это означает, что упакованные структуры не могут содержать переменные **real** или **shortreal,** неупакованные структуры, неупакованные союзы, неупакованные массивы.

Операции над упакованными структурами

Упакованные структуры рассматриваются как векторы

Поскольку упакованная структура хранится как вектор, то и операции над структурой совершаются как над вектором. Поэтому математические, логические и другие операции которые совершаются над векторами могут также совершаться над упакованными структурами.

```
typedef struct packed {
  logic        valid;
  logic [ 7:0] tag;
  logic [31:0] data;
} data_word_t;

data_word_t packet_in, packet_out;

always @(posedge clock)
  packet_out <= packet_in << 2;
```

Когда упакованной структуре присваивается список значений между скобками '{ }, как обсуждалось в секции 5.1.2, значения в списке назначаются членам структуры. В данном случае упакованная структура рассматривается как не упакованная, нежели как вектор. Значения внутри скобок '{ } это значения для членов структуры.

```
packet_in = '{1, '1, 1024};
```

Предыдущая строка присваивает 1 к valid, FF к tag, и 1024 к data.

Знаковые упакованные структуры

Упакованные структуры, используемые как вектор, могут быть знаковыми или беззнаковыми

Упакованные структуры могут быть объявлены с ключевыми словами **signed** или **unsigned**. Эти модификаторы работают для всей структуры в целом, и используются такие же операции как над вектором. Модификаторы не имеют влияния на каждый член структуры. Каждый член структуры рассматривается как знаковый или беззнаковый, исходя из объявленного типа этого члена. Часть упакованной структуры всегда беззнаковая, также как часть вектора в Verilog.

```
typedef struct packed signed {
  logic                valid;
  logic         [ 7:0] tag;

   logic signed [31:0] data;
} data_word_t;

data_word_t  A, B;

always @(posedge clock)
  if ( A < B )           // signed comparison
    ...
```

5.1.4 Передача структур через порты

Порты могут быть объявлены как структурный тип

Структуры могут передаваться через модуль и порты. Структуры должны быть определены как тип определяемый пользователем с использованием

typedef, который позволяет модулю или порту быть объявленным как структурный тип.

```
package definitions;
  typedef enum {ADD, SUB, MULT, DIV} opcode_t;
  typedef struct {
    logic [31:0]  a, b;
    opcode_t      opcode;
    logic [23:0]  address;
    logic         error;
  } instruction_word_t;
endpackage

module alu
(input definitions::instruction_word_t IW,
 input wire                            clock);
  ...
endmodule
```

Альтернативный стиль чтобы явно именовать содержимое пакета, это импорт пакета в пространство единицы компиляции **$unit**. Так же возможно напрямую определять пользовательские типы в пространстве **$unit**. Импортирование пакетов и использование пространства единицы компиляции **$unit** рассматривалось в главе 2.

Когда неупакованная структура передаётся через порт модуля, структура точно такого же типа должна быть на каждой стороне порта. Анонимные структуры, объявленные в разных модулях, даже если имена структур, члены и имена членов одинаковы, такие структуры имеют разные типы. Передача неупакованных структур через порты модуля более детально рассматривается в секции 9.6.2.

5.1.5 Передача структур как аргументы в задачи и функции

Структуры могут передаваться в задачи и функции

Структуры могут быть переданы как аргументы в задачи или функции. Для этого структура должна быть определена как тип определяемый пользователем с использованием **typedef**, после чего аргумент задачи или функции может быть объявлен как структурный тип.

```
module processor (...);
  ...
  typedef enum {ADD, SUB, MULT, DIV} opcode_t;

  typedef struct {  // typedef is local
    logic [31:0]  a, b;
    opcode_t      opcode;
    logic [23:0]  address;
    logic         error;
  } instruction_word_t;

  function alu (input instruction_word_t IW);
    ...
  endfunction
endmodule
```

Когда вызывается задача или функция, формальным аргументом которой является неупакованная структура, структура точно такого же типа должна передаваться в задачу или функцию. Анонимные структуры, имеющие одинаковые члены и имена членов, имеют разные типы.

5.1.6 Рекомендации по синтезу

Неупакованные и упакованные структуры синтезируемы. Синтез поддерживает передачу структур через порты модуля, и как входные/выходные аргументы задач и функций. Поддерживается присваивание значений структурам через имя члена и через список значений.

5.2 Объединения

Объединение хранит только одно значение

SystemVerilog добавляет C-подобные объединения в Verilog. Объединение это один элемент хранения, который может иметь множественное представление. Каждое представление может быть разного типа. Синтаксис объявления объединения похож на синтаксис структуры, и обращение к членам объединения такое же как у структуры.

```
union {
  int  i;
  int unsigned u;
} data;
...
data.i = -5;
$display("data is %d", data.i);

data.u = -5;
$display("now data is %d", data.u);
```

Объединения экономят память и могут улучшать производительность

Хотя синтаксис объединения и структуры похож, объединение сильно отличается от структуры. Структура может хранить несколько значений. Это группа переменных под одним именем. Объединение может хранить только одно значение. Типичное применение объединения когда значение может быть представлено несколькими разными типами, но только один тип в конкретный момент времени.

Типизированные и анонимные объединения

Объединение может быть определено как тип используя **typedef**, так же как и структура. Обращение к объединению, которое определено через пользовательский тип, осуществляется как к типизированному *typed union.* Если typedef не используется, к объединению обращаются как к анонимному *anonymous union.*

```
typedef union {    // typed union
  int  i;
  int unsigned u;
} data_t;

data_t a, b;  // two variables of type data_t
```

5.2.1 Неупакованные объединения

Неупакованное объединение может содержать переменную любого типа, включая типы **real**, неупакованные структуры и неупакованные массивы. Программные инструменты (tools) могут хранить значения в неупакованных объединениях в произвольной манере. Нет требований, что каждый инструмент хранит различные типы одинаковым способом.

Неупакованные объединения не синтезируемы. Они являются абстрактным типом, который полезен для высокоуровневой системы и уровня транзакции. Это может быть полезным для хранения типов с 4-мя, 2-мя состояниями и не синтезируемых типов, таких как real.

Чтение из неупакованного объединения члена, который отличается от последнего записанного, может являться причиной неопределённого результата.

Если значение сохранено в одном члене объединения, а читается из другого, тогда прочитанное значение не определено, и может выдавать различные результаты в разных программных инструментах.

Следующий пример не синтезируемый, но показывает, как неупакованное объединение может хранить разные типы. Пример показывает объединение которое может хранить либо тип **int** либо тип **real.** Поскольку эти типы сохранены в различные моменты, то и чтение этих типов будет корректным. Пример содержит дополнительную логику, чтобы отслеживать, как значения сохранены в объединении. Объединение является членом структуры. Второй член структуры это флаг, который устанавливается, если значение real было

сохранено в объединении. Когда значение читается, флаг может быть проверен для определения типа значения.

```
struct {
  bit is_real;
  union {
    int  i;
    real r;
  } value;
} data;
//...
always @(posedge write) begin
  case (operation_type)
    INT_OP: begin
              data.value.i <= 5;
              data.is_real <= 0;
            end
    FP_OP:  begin
              data.value.r <= 3.1415;
              data.is_real <= 1;
            end
  endcase
end
//...
always @(posedge read) begin

   if (data.is_real)
     real_operand <= data.value.r;
   else
     int_operand <= data.value.i;
 end
```

5.2.2 Тэговые объединения.

Объединение может быть объявлено как тэговое.

```
union tagged {
  int  i;
  real r;
} data;
```

Тэговые объединения содержат неявный тэговый член.

Тэговое объединение содержат неявный член который хранит тэг, этот тэг содержит имя последнего члена объединения в котором было сохранено значение. Когда значение сохраняется в тэговом объединении, используется тэговое выражение *tagged expression*, неявный тэг автоматически сохраняет информацию, в какой член было записано значение.

Использование тэгового выражения для сохранения значений в тэговых объединениях

Значение может быть записано в член тэгового объединения, используя тэговое выражение. Тэговое выражение имеет ключевое слово, за которым следует имя члена и значение. Тэговое выражение присваивается имени объединения. На пример:

```
data = tagged i 5;  // store the value 5 in
                    // data.i, and set the
                    // implicit tag
```

Для чтения значений из тэгового объединения используется имя члена.

```
d_out = data.i;  // read value from union
```

Тэговые объединения предъявляют требования к программным инструментам (tools) по использованию объединений, и генерируют сообщение об ошибке, если значение прочитано не из того члена в который была запись с использованием тэгового выражения. На пример, если в последнем тэговом выражении для записи использовался член i (int тип), то следующая строка кода даёт ошибочный результат.

```
d_out = data.r;  // ERROR: member does not match
                 // the union's implicit tag
```

После того как значение было присвоено тэговому объединению с использованием тэгового выражения, значения могут быть записаны в тот же

член, используя имя члена. Если имя члена не соответствует текущему тэгу, то результат будет ошибочным.

```
data.i = 7;  // write to union member; member
             // name must match the current
             // union tag
```

Разработчик несёт ответственность за последовательное чтение значений члена объединения, в который была осуществлена последняя запись данных. Если же логика проекта использует объединение в непоследовательном стиле, то программные инструменты (tools) должны информировать разработчика об ошибке.

5.2.3 Упакованные объединения

Все члены упакованного объединения имеют одинаковый размер

Объединение, так же как и структура, может быть объявлено как **packed**. В упакованном объединении количество бит каждого члена должно быть одинаково. Таким образом, область хранения будет представлена одинаковым количеством бит. Из-за этого ограничения упакованные объединения синтезируемы.

Упакованное объединение может хранить только интегральные значения состоящие из смежных бит. Если в упакованном объединении есть член с 4-мя состояниями, то объединение имеет 4 состояния. Упакованное объединение не может содержать переменные **real** или **shortreal**, неупакованные структуры, не пакованные объединения, или неупакованные массивы.

Упакованное объединение позволяет записывать данные, используя один формат и читать, используя другой. В процессе разработки нет необходимости отслеживать, как были сохранены данные. Потому что для сохранения данных используется одинаковое количество бит.

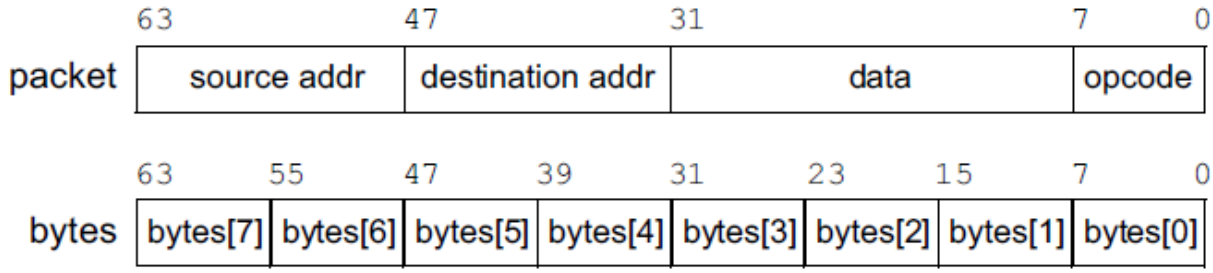
Следующий пример определяет упакованное объединение, в котором значение может быть представлено двумя способами: либо как пакет данных (используя упакованную структуру) либо как массив байт.

```
typedef struct packed {
  logic [15:0] source_address;
  logic [15:0] destination_address;
  logic [23:0] data;

   logic [ 7:0] opcode;
} data_packet_t;

union packed {
   data_packet_t  packet;    // packed structure
   logic [7:0][7:0] bytes;   // packed array
} dreg;
```

Figure 5-2: Packed union with two representations of the same storage



Так как объединение упаковано, информация будет храниться, используя одинаковое расположение бит, не смотря на то какое представление объединения используется. Это означает, что значение может быть загружено используя массив байт (поток байт из последовательного порта), и то же значение может быть прочитано, используя пакетный формат data_packet.

```
always @(posedge clock, negedge resetN)
  if (!resetN) begin
    dreg.packet <= '0;  // store as packet type
    i <= 0;
  end
  else if (load_data) begin
    dreg.bytes[i] <= byte_in; // store as bytes
    i <= i + 1;
  end

always @(posedge clock)
  if (data_ready)
    case (dreg.packet.opcode) // read as packet
    ...
```

Упакованные, тэговые объединения

Объединение может быть объявлено, одновременно, как упакованное и тэговое. В этом случае члены объединения могут иметь разный битовый размер, но должны быть интегрального типа (1 или более смежных бит). В таких объединениях разрешается только чтение члена, который соответствует члену последнего тэгового выражения записи в объединение.

```
union tagged packed {
  logic [15:0] short_word;
  logic [31:0] word;
  logic [63:0] long_word;
} data_word;
```

5.2.4 Рекомендации по синтезу

Только упакованные объединения синтезируемы

Упакованные объединения могут быть синтезируемы

Объединение хранит единственное значение, не смотря на количество представленных типов. Чтобы воплотить объединение в железе (hardware), необходимо чтобы все члены объединения сохранялись в области одинакового размера и с одинаковым расположением бит. Такие упакованные объединения синтезируемы. Неупакованное объединение не гарантирует, что каждый тип будет сохранён одинаковым способом, и поэтому оно не синтезируемо.

Упакованные, тэговые объединения синтезируемы, но на тот момент когда была написана эта книга не все компиляторы поддерживали синтез таких объединений.

5.2.5 Пример использования структур и объединений

Структуры предоставляют механизм группирования данных под одним общим именем. К каждой единице данных можно обратиться через имя, или к группе как к единому целому. Объединения позволяют использовать данные разными способами.

Следующий пример показывает простое АЛУ, которое может оперировать знаковыми или беззнаковыми значениями. В АЛУ передаётся командное слово представленное структурой: код операции, два операнда и флаг знака данных. АЛУ не может одновременно оперировать знаковыми и беззнаковыми данными. Поэтому знаковые и беззнаковые значения представлены объединением двух типов. Это позволяет одной переменной представлять как знаковые, так и беззнаковые значения. В главе 11 показан другой пример использования структур и объединений, чтобы представить сложную информацию в простой и интуитивной форме.

Example 5-1: Using structures and unions

```
package definitions;

  typedef enum {ADD, SUB, MULT, DIV, SL, SR} opcode_t;

  typedef enum {UNSIGNED, SIGNED} operand_type_t;

  typedef union packed {
    logic        [31:0] u_data;
    logic signed [31:0] s_data;
  } data_t;

  typedef struct packed {
    opcode_t         opc;
    operand_type_t   op_type;
    data_t           op_a;
    data_t           op_b;
  } instr_t;

endpackage

import definitions::*;   // import package into $unit space

module alu
(input  instr_t IW,
 output data_t  alu_out);

  always @(IW) begin
    if (IW.op_type == SIGNED) begin
      case (IW.opc)
        ADD : alu_out.s_data = IW.op_a.s_data + IW.op_b.s_data;
        SUB : alu_out.s_data = IW.op_a.s_data - IW.op_b.s_data;
        MULT: alu_out.s_data = IW.op_a.s_data * IW.op_b.s_data;
        DIV : alu_out.s_data = IW.op_a.s_data / IW.op_b.s_data;
        SL  : alu_out.s_data = IW.op_a.s_data <<< 2;
        SR  : alu_out.s_data = IW.op_a.s_data >>> 2;
      endcase
    end
    else begin
      case (IW.opc)
        ADD : alu_out.u_data = IW.op_a.u_data + IW.op_b.u_data;
        SUB : alu_out.u_data = IW.op_a.u_data - IW.op_b.u_data;
        MULT: alu_out.u_data = IW.op_a.u_data * IW.op_b.u_data;
        DIV : alu_out.u_data = IW.op_a.u_data / IW.op_b.u_data;
        SL  : alu_out.u_data = IW.op_a.u_data << 2;
        SR  : alu_out.u_data = IW.op_a.u_data >> 2;
```

```
      endcase
    end
  end
endmodule
```

5.3 Массивы

5.3.1 Неупакованные массивы

Verilog-1995 массивы

Базовый синтаксис массива Verilog:

```
<data_type> <vector_size> <array_name> <array_dimensions>
```

На пример:

```
reg [15:0] RAM [0:4095];  // memory array
```

Verilog-1995 допускает только одномерные массивы. Доступ к массиву такой же как к памяти, потому как первоначальная цель была смоделировать такие запоминающие устройства как RAM и ROM. Verilog-1995 ограничивает объявление массива типами **reg**, **integer** и **time**.

Verilog массивы

Verilog-2001 значительно расширил возможности массивов Verilog-1995, позволяя переменной или сети иметь любой тип, исключая event тип, чтобы быть объявленной как массив, а так же как многомерный массив. В Verilog-2001 переменные и сети могут использоваться в массивах.

```
// a 1-dimensional unpacked array of
// 1024 1-bit nets
wire n [0:1023];

// a 1-dimensional unpacked array of
// 256 8-bit variables
reg [7:0] LUT [0:255];

// a 1-dimensional unpacked array of
// 1024 real variables
real r [0:1023];

// a 3-dimensional unpacked array of

// 32-bit int variables
integer i [7:0][3:0][7:0];
```

Verilog ограничивает доступ к массиву одним элементом в данный момент времени

Чтение или запись в несколько элементов массива является ошибкой.

```
integer i [7:0][3:0][7:0];
integer j;

j = i[3][0][1];  // legal: selects 1 element
j = i[3][0];     // illegal: selects 8 elements
```

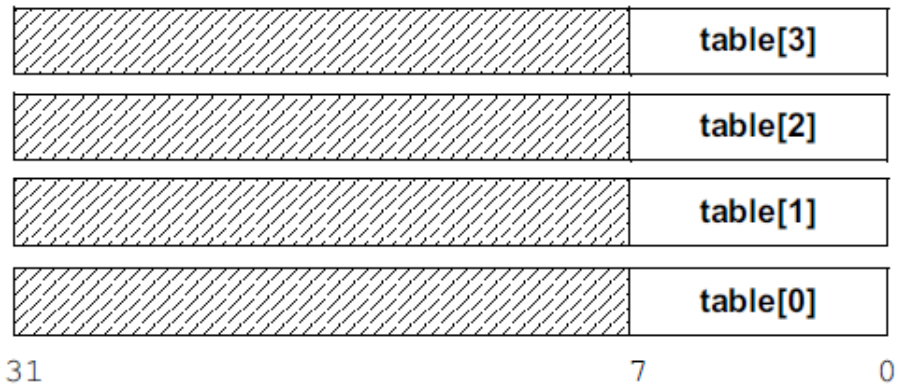
Неупакованные массивы хранят каждый элемент независимо

SystemVerilog ссылается на Verilog стиль объявления массива как не упакованного. В не упакованных массивах каждый элемент может быть сохранён независимо от других элементов, но все они сгруппированы под одним именем. Verilog не определяет как программные инструменты (tools) должны хранить элементы массива. На пример, массив элементов разрядностью 8 бит, симулятор или другое программное обеспечение может

хранить каждый 8 битный элемент в 32 битном слове. Рис 5-3 иллюстрирует этот пример.

```
wire [7:0] table [3:0];
```

Figure 5-3: Unpacked arrays can store each element independently



Расширение возможностей неупакованных массивов в SystemVerilog

SystemVerilog позволяет использовать массивы любого типа

Для неупакованных массивов SystemVerilog позволяет использовать типы **event, logic, bit, byte, int, longint, shortreal** и **real**. Так же могут использоваться типы определяемые пользователем **typedef** и типы **struct** и **enum.**

```
bit [63:0] d_array [1:128]; // array of vectors

shortreal cosines [0:89];  // array of floats

typedef enum {Mo, Tu, We, Th, Fr, Sa, Su} Week;
Week Year [1:52];  // array of Week types
```

SystemVerilog может обращаться ко всему массиву или к нескольким элементам

SystemVerilog добавляет в Verilog возможность обращения ко всему не упакованному массиву или части элементов. Часть есть один или более

смежных элементов одноразмерного массива. Это расширение даёт возможность копировать содержимое целого массива или части массива в другой массив.

Левая (количество бит элемента массива) и правая (количество элементов массива) части копируемого неупакованного массива и типы должны быть одинаковы.

Копирование в неупакованный массив нескольких элементов

Для того чтобы копировать несколько элементов в неупакованный массив, левая поле (количество бит элемента массива) и правое поле(количество элементов массива) копируемого неупакованного массива и типы должны быть одинаковы. Таким образом, тип, размер и число размерности должно быть одинаковым.

Следующий пример является корректным. Хотя числа показывающие размерности массивов разные, размер и расположение полей массивов одинаковое.

```
int  a1 [7:0][1023:0];  // unpacked array
int  a2 [1:8][1:1024];  // unpacked array

a2 = a1;          // copy an entire array

a2[3] = a1[0];  // copy a slice of an array
```

Копирование массива более детально обсуждается в этой главе позже в секции 5.3.7.

Упрощённые объявления неупакованного массива

В языке C массивы определяются размером

В языке C массивы всегда начинаются с нулевого элемента (адрес 0). Единственное что требуется для объявления массива в языке C это указать размер массива. На пример:

```
int array [20];  // a C array with addresses
                 // from 0 to 19
```

В Verilog массивы определяются диапазоном адреса

При объявлении массива необходимо задать начальный и конечный адрес размера массива.

```
int array [64:83];  // a Verilog array with
                    // addresses from 64 to 83
```

Неупакованные массивы в SystemVerilog определяются размером

SystemVerilog добавляет С-подобное объявление массива в Verilog, позволяющее определять неупакованные массивы размером, вместо начального и конечного адреса. Объявление массива:

```
logic [31:0] data [1024];
```

эквивалентно объявлению:

```
logic [31:0] data [0:1023];
```

Как и в С элементы неупакованного массива пронумерованы, начиная с 0 и заканчивая size – 1. Упрощённое объявление массива в стиле С не может быть использовано в объявлении вектора (упакованные массивы). Следующий пример есть синтаксическая ошибка.

```
logic [32] d; // illegal vector declaration
```

5.3.2 Упакованные массивы

Язык Verilog позволяет создавать векторы из однобитных типов, таких как **reg** и **wire.** Диапазон вектора ставиться перед именем вектора, тогда как диапазон неупакованного массива ставится после имени массива.

Verilog векторы есть одноразмерные упакованные массивы

SystemVerilog обращается к вектору как к упакованному массиву. В Verilog вектор есть одноразмерный упакованный массив.

```
wire [3:0] select;  // 4-bit "packed array"
reg [63:0] data;    // 64-bit "packed array"
```

В SystemVerilog разрешены много-размерные, упакованные массивы

SystemVerilog добавляет возможность объявлять многомерность в упакованном массиве.

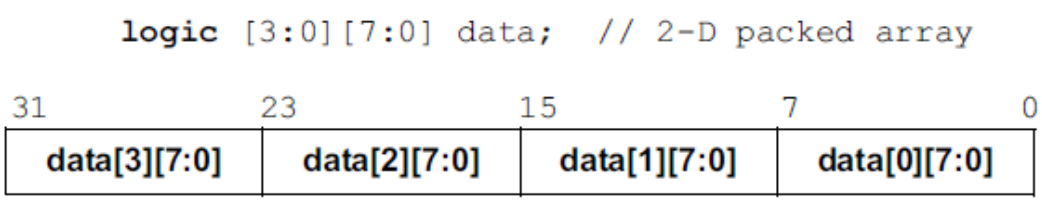
```
logic [3:0][7:0] data;  // 2-D packed array
```

Упакованные массивы не имеют выравнивания

SystemVerilog определяет как хранятся элементы упакованного массива. Весь массив может храниться как смежные биты, так же как вектор. Размерность каждого упакованного массива есть под-поле внутри вектора.

Упакованный массив, объявленный выше состоит из 4-х 8-битных под-массивов. Рис 5-4 показывает как храниться двумерный массив.

Figure 5-4: Packed arrays are stored as contiguous elements



Типы упакованного массива

Только битовые типы могут быть упакованы

Упакованные массивы могут быть сформированы, используя битовые типы (**logic**, **bit** или **reg**), упакованные массивы, упакованные структуры и упакованные объединения. Упакованные массивы также могут быть сформированы из типов Verilog (**wire**, **uwire**, **wand**, **tri**, **triand**, **trior**, **tri0**, **tri1**, или **trireg**)

```
typedef struct packed {
  logic [ 7:0] crc;
  logic [63:0] data;
} data_word;

data_word [7:0] darray;  // 1-D packed array of
                         // packed structures
```

Обращение к упакованным массивам

К упакованному массиву можно обращаться как к единому целому, как к отдельны битам и как к части массива. К многомерным упакованным массивам, также можно обращаться как к частям массива (slice). Slice есть одна или более смежных размерностей массива.

```
logic [3:0][7:0] data;  // 2-D packed array


wire [31:0] out = data;           // whole array
wire sign = data[3][7];           // bit-select
wire [3:0] nib = data [0][3:0];   // part-select
byte high_byte;
assign high_byte = data[3];       // 8-bit slice
logic [15:0] word;
assign word = data[1:0];          // 2 slices
```

Операции над упакованными массивами

Операции над упакованными массивами такие же как над вектором

Поскольку упакованные массивы хранятся как вектора, любая операция над вектором в Verilog может также осуществляться над упакованными массивами: операции над частью массива, битовые операции, операции конкатенации, математические операции и логические операции.

```
logic [3:0][15:0] a, b, result; // packed arrays
...
result = (a << 1) + b;
```

упакованные массивы используют правила для вектора Verilog

Нет семантического различия между Verilog вектором и SystemVerilog упакованным массивом. Упакованные массивы используют стандартные правила Verilog вектора для операций и выражений. Когда есть несоответствие размера в векторе, упакованный массив будет усекаться или расширяться слева, также как Verilog вектор.

5.3.3 Использование упакованных и неупакованных массивов

Способность объявлять многомерные массивы как упакованных, так и неупакованных даёт большую гибкость в представлении большого количества сложных данных. Некоторые рекомендации по использованию каждого типа массива:

Использование неупакованных массивов для моделирования памяти

Использование неупакованных массивов для моделирования:

- Массивы типов byte, int, integer, real, неупакованные структуры, неупакованные объединения, и другие типы, не являющиеся однобитовыми типами.
- Массивы, где осуществляется доступ к одному элементу в данный момент времени, такие как RAM и ROM.

```
module ROM (...);
  byte mem [0:4095]; // unpacked array of bytes
  assign data = select? mem[address]: 'z;
  ...
```

Использование упакованного массива для создания вектора с подполями

Использование упакованных массивов для моделирования:

- Вектора из однобитных типов (также как в Verilog)
- Вектора где удобно использовать доступ к подполям вектора

```
logic [39:0][15:0] packet; // 40 16-bit words

packet = input_stream;  // assign to all words
data = packet[24];      // select 1 16-bit word
tag = packet[3][7:0];   // select part of 1 word
```

5.3.4 Инициализация массивов при объявлении

Инициализация упакованного массива

упакованные массивы инициализируются также как вектора

Упакованные массивы могут быть инициализированы при объявлении, используя простое присваивание, как вектора в Verilog. Присваиваемое значение может быть константой, конкатенацией из константных значений или копирование константных значений.

```
logic [3:0][7:0] a = 32'h0;          // vector assignment
logic [3:0][7:0] b = {16'hz,16'h0}; // concatenate operator
logic [3:0][7:0] c = {16{2'b01}};   // replicate operator
```

В примере скобки { } означают оператор конкатенации.

Инициализация неупакованного массива

Неупакованные массивы инициализируются списком значений

Неупакованные массивы могут быть инициализированы при объявлении, используя список значений заключённых в скобки { } для каждой размерности массива. Этот синтаксис похож на присвоение списка значений массиву в C, с добавлением апострофа перед открывающейся скобкой. Такое использование '{ показывает список выражений, а не операцию конкатенации. В SystemVerilog нельзя опускать внутренние скобки, как это делается в языке C. Присвоение требует вложенных скобок, что соответствует размерностям массива.

```
int d [0:1][0:3] = '{ '{7,3,0,5}, '{2,0,1,6} };
// d[0][0] = 7
// d[0][1] = 3
// d[0][2] = 0
// d[0][3] = 5

// d[1][0] = 2
// d[1][1] = 0
// d[1][2] = 1
// d[1][3] = 6
```

SystemVerilog предоставляет сокращённое объявление списка значений. Внутренний список для одномерного массива может повторяться любое число раз, используя Verilog-подобный фактор копирования. Фактор копирования апострофом не сопровождается.

```
int e [0:1][0:3] = '{ 2{7,3,0,5} };
// e[0][0] = 7
// e[0][1] = 3
// e[0][2] = 0
// e[0][3] = 5

// e[1][0] = 7
// e[1][1] = 3
// e[1][2] = 0
// e[1][3] = 5
```

Операторы '{ } и '{n{ }} это не тоже что в Verilog оператор конкатенации { } и копирования {n{ }}.

скобки { } используются двумя способами

Когда инициализируется неупакованный массив, скобки { } представляют список значений. Это не операция конкатенации как в Verilog. Каждое значение в списке присваивается соответствующему элементу, следуя правилам Verilog. Это значит, безразмерные литеральные значения могут быть определены в списке также как и действительные значения. Операции конкатенации и копирования в Verilog используют скобки { } без апострофа. Эти операторы требуют, чтобы литеральные значения имели определённый размер, для создания вектора. Безразмерные числа и действительные значения не разрешены в операторах конкатенации и копирования.

Значение по умолчанию для неупакованных массивов

массив может быть инициализирован значением по умолчанию

SystemVerilog предоставляет механизм инициализации всех элементов не упакованного массива или части не упакованного массива значением по умолчанию. Значение по умолчанию определяется внутри скобок '{ } с использованием ключевого слова **default,** которое отделяется от значения двоеточием. Значение должно быть совместимо с типом массива. Значение совместимо если оно может быть приведено к типу массива.

```
int a1 [0:7][0:1023] = '{default:8'h55};
```

Неупакованный массив может быть массивом структур или другими типами определяемыми пользователем (см секцию 5.3.11). Эти конструкции могут содержать разные типы. Для того чтобы инициализировать разные типы внутри массива разными значениями, значение по умолчанию может быть определено с использованием ключевого слова для типа вместо **default.**

Присвоение по умолчанию массиву автоматически передаётся в структуры или объединения, чтобы найти переменные определённого типа. Пример определения значений по умолчанию на базе типов – секция 5.1.2

5.3.5 Присвоение значений массивам

Присвоение значений неупакованным массивам

Язык Verilog поддерживает два способа присвоения значений неупакованным массивам:

- Значение может быть присвоено одному элементу
- Значение может быть присвоено выбранному биту или части элемента

SystemVerilog расширяет Verilog двумя дополнительными способами, присвоения значений неупакованным массивам.

- Всему массиву может быть присвоен список значений.
- Слайсу (несколько смежных бит) массива может быть присвоен список значений.

Список значений определяется между скобками `'{}`, также как инициализация неупакованного массива, которая обсуждалась в секции 5.3.4.

```
byte a [0:3][0:3];

a[1][0] = 8'h5;  // assign to one element

a = '{'{0,1,2,3},
      '{4,5,6,7},
      '{7,6,5,4},
      '{3,2,1,0}};
// assign a list of values to the full array

a[3] = '{'hF, 'hA, 'hC, 'hE};
// assign list of values to slice of the array
```

Список, присваиваемых значений неупакованному массиву может определяться по умолчанию, используя ключевое слово **default.** Части массива могут быть установлены в различные значения.

```
always @(posedge clock, negedge resetN)
  if (!resetN) begin
    a = '{default:0};     // init entire array
    a[0] = '{default:4}; // init slice of array
  end
  else begin
    //...
  end
```

Присвоение значений упакованным массивам

Многомерные упакованные массивы это вектора с подполями.

Упакованные массивы это вектора, значения которым присваиваются также как векторам Verilog. Упакованным векторам значения присваиваются:

- Одному элементу массива
- Всему массиву
- Выбранной части массива
- Слайсу (несколько смежных бит) массива

```
logic [1:0][1:0][7:0] a;  // 3-D packed array

a[1][1][0] = 1'b0;    // assign to one bit

a = 32'hF1A3C5E7;     // assign to full array

a[1][0][3:0] = 4'hF; // assign to a part select

a[0] = 16'hFACE;      // assign to a slice

a = {16'bz, 16'b0};  // assign concatenation
```

5.3.6 Копирование массивов

Эта подсекция описывает правила для четырёх комбинаций присвоений массивов массивам.

Присвоение упакованных массивов упакованным массивам

Разрешается присваивать упакованные массивы упакованным массивам

Упакованный массив может быть присвоен другому упакованному массиву. Упакованные массивы рассматриваются как вектора, массивы различаются по размеру и типу. Правила присвоения для векторов, стандарта Verilog, сокращают или расширяют массивы, если есть несоответствие в размере массива.

```
bit   [1:0][15:0] a;  // 32 bit 2-state vector
logic [3:0][ 7:0] b;  // 32 bit 4-state vector
logic [15:0]      c;  // 16 bit 4-state vector
logic [39:0]      d;  // 40 bit 4-state vector

b = a;  // assign 32-bit array to 32-bit array
c = a;  // upper 16 bits will be truncated
d = a;  // upper 8 bits will be zero filled
```

Присвоение неупакованных массивов неупакованным массивам

Разрешается присваивать не упакованный массив не упакованному массиву

Не упакованные массивы могут присваиваться не упакованным массивам, если оба массива имеют одинаковую размерность, количество элементов и тип. Присвоение осуществляется копированием каждого элемента массива в соответствующий элемент массива назначения. Нет необходимости, чтобы элементы двух массивов были пронумерованы одинаково. Размерность и типы массивов должны совпадать.

```
logic [31:0] a [2:0][9:0];
logic [0:31] b [1:3][1:10];

a = b;  // assign unpacked array to unpacked
        // array
```

Присвоение неупакованных массивов с разными размерами требует приведения

Если два неупакованных массива не идентичны по конфигурации (layout), то присвоение может быть сделано с использованием операции приведения. Операции приведения представлена в этой главе, в секции 5.3.7.

Присвоение упакованных массивов неупакованным массивам

Присвоение упакованных массивов неупакованным массивам требует приведения

Упакованный массив не может быть присвоен неупакованному массиву напрямую. Даже если размерности двух массивов идентичны, упакованный массив рассматривается как вектор, который не может быть напрямую присвоен неупакованному массиву, где каждый элемент массива может храниться независимо от других элементов. Однако, присвоение может быть сделано с использованием операции приведения.

5.3.7 Копирование массивов и структур с использованием приведения

Приведение преобразует массивы во временный битовый вектор

Приведение, временно, преобразует неупакованный массив в поток битов в векторной форме. Идентификация элементов внутри массива теряется – временный вектор это просто поток бит. Затем, временный вектор может быть присвоен массиву, который является либо упакованным массивом, либо неупакованным. Общее число бит, представленное источником и массивом назначения должно быть одинаковым. Однако, размер каждого элемента в двух массивах может быть разным.

Приведение предоставляет механизм для:

- Присвоение неупакованного массива неупакованному массиву разной конфигурации (layout).
- Присвоение неупакованного массива упакованному массиву
- Присвоение упакованного массива неупакованному массиву
- Присвоение фиксированного или динамического массива динамическому массиву.
- Присвоение структуры другой структуре с разными конфигурациями (layout).

Приведение использует статический оператор приведения SystemVerilog. Приведение требует, чтобы массив назначения был представлен как тип определяемый пользователем, используя **typedef**.

```
typedef int data_t [3:0][7:0]; // unpacked type
data_t a;                      // unpacked array
int b [1:0][3:0][3:0];         // unpacked array

a = data_t'(b);    // assign unpacked array to
                   // unpacked array of a
                   // different layout
```

Операция приведения осуществляется преобразованием массива источника (или структуры) во временный вектор (поток бит), и затем присвоение группы бит каждому элементу массива назначения. Присвоение делается слева направо так, что самые левые биты, источника битового потока, назначаются первому элементу массива назначения, следующие левые биты второму элементу, и.т.д.

5.3.8 Массивы массивов

Массив может смешивать упакованную и неупакованную размерности

Обычно используется комбинация неупакованных массивов и упакованных. В Verilog, массив памяти это смешанные типы массива. В следующем примере объявляется неупакованный массив, состоящий из 64-битных векторов.

```
logic [63:0] mem [0:4095];
```

Следующий пример объявляет неупакованный массив 32-битных элементов, где каждый элемент есть упакованный массив, поделённый на 4 8-битных поля.

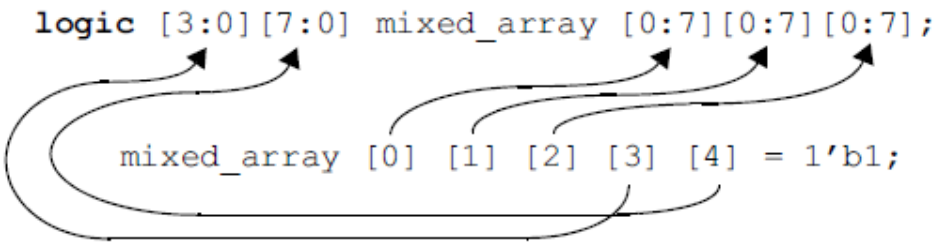
```
wire [3:0][7:0] data [0:1023];
```

Индексация массивов массивов

Неупакованная размерность индексируется перед упакованной

Когда индексируются массивы массивов, неупакованная размерность индексируется первой, слева на право. Упакованная размерность (векторные поля) индексируется второй, слева на право. Рисунок 5-5 показывает порядок, в котором выбираются размерности в смешанном многомерном массиве.

Figure 5-5: Selection order for mixed packed/unpacked multi-dimensional array



5.3.9 Использование типов определяемых пользователем в массивах

Массивы могут содержать типы определяемые пользователем

Типы, определяемые пользователем, могут быть использованы как элементы массива. Следующий пример определяет пользовательский тип для unsigned integer, и объявляет неупакованный массив из 128 unsigned integer.

```
typedef int unsigned uint;
uint u_array [0:127];  // array of user types
```

Типы, определяемые пользователем, могут быть определены из определения массива. Эти пользовательские типы, могут быть использованы в других определениях массива, создавая сложный массив.

```
typedef logic [3:0] nibble;  // packed array
nibble [31:0] big_word;      // packed array
```

Предыдущий пример эквивалентен:

```
logic [31:0][3:0] big_word;
```

Другой пример сложного массива построен из пользовательского типа:

```
typedef logic [3:0] nibble;  // packed array
typedef nibble nib_array [0:3];  // unpacked
nib_array compound_array [0:7];  // unpacked
```

Этот последний пример эквивалентен:

```
logic [3:0] compound_array [0:7][0:3];
```

5.3.10 Передача массивов через порты задачам и функциям

В Verilog к упакованному массиву обращаются как к одномерному вектору. Verilog позволяет передавать упакованные массивы через порты модуля, и использовать их как входные/выходные аргументы задач и функций. В Verilog нельзя неупакованные массивы передавать через порты модуля, задачи и функции.

SystemVerilog позволяет использовать неупакованные массивы как порты и аргументы

SystemVerilog позволяет массивы любого типа и любой размерности передавать через порты или аргументы задач и функций.

Чтобы передать массив через порт, или как аргумент задачи или функции, порт или аргумент задачи или функции должны быть объявлены как массив. Массивы, которые передаются через порт, следуют тем же правилам и ограничениям как массивы, которые присваиваются другим массивам, это обсуждалось в секции 5.3.6.

```
module CPU (...);
  ...
  logic [7:0] lookup_table [0:255];

  lookup i1 (.LUT(lookup_table));
  ...
endmodule

module lookup (output logic [7:0] LUT [0:255]);
  ...
  initial load(LUT);  //task call

  task load (inout logic [7:0] t [0:255]);
    ...

  endtask
endmodule
```

5.3.11 Массивы структур и объединений

Массивы могут содержать структуры и объединения

Упакованные и неупакованные массивы могут включать структуры и объединения как элементы массива. В упакованном массиве, структуры и объединения должны быть упакованными.

```
typedef struct packed {  // packed structure
  logic [31:0] a;
  logic [ 7:0] b;
} packet_t;

packet_t [23:0] packet_array;  // packed array
                               // of 24 structures


typedef struct {  // unpacked structure
  int  a;
  real b;
} data_t;

data_t data_array [23:0];  // unpacked array
                           // of 24 structures
```

5.3.12 Массивы в структурах и объединениях

Структуры и объединения могут содержать массивы

Структуры и объединения могут включать упакованные и неупакованные массивы. Упакованные структуры и объединения могут включать только упакованные массивы.

```
struct packed {           // packed structure
  logic parity;
  logic [3:0][ 7:0] data;  // 2-D packed array
} data_word;


struct {              // unpacked structure
  logic data_ready;
  logic [7:0] data [0:3];   // unpacked array
} packet_t;
```

5.3.13 Рекомендации по синтезу

Массивы и присвоения массивов синтезируемы.

- Объявления массивов – неупакованные и упакованные массивы синтезируемы. Массивы могут иметь любую размерность.

- Присваивание значений массивам – синтез поддерживает присваивание значений отдельному элементу массива, биту или нескольким бит элемента массива, части массива или всему массиву. Присваивание списка литеральных значений массиву так же синтезируемо, включая литералы с использованием ключевого слова **default.**

- Копирование массивов – синтез поддерживает прямое присвоение упакованных массивов упакованным массивам. Синтез также поддерживает прямое присвоение неупакованных массивов неупакованным массивам с одинаковой размерностью. Присвоение любого типа массива любому другому типу массива с использованием приведения также синтезируемо.

- Массивы в структурах и объединениях – использование массивов внутри структур и объединений синтезируемо. Объединения должны быть упакованными, это значит, что и массивы внутри объединения должны быть упакованными.

- Массивы из структур и объединений - Массивы из структур и массивы из объединений синтезируемы (объединения должны быть упакованными). Структура или объединение должны быть типизированы (с использованием **typedef**) для того чтобы определить массив из структуры или объединения.

- Передача массивов – массивы, передаваемые через порты модуля или как аргументы задачи или функции, синтезируемы.

5.3.14 Пример использования массивов

Следующий пример моделирует командный регистр, используя упакованный массив из 32 инструкций. Каждая инструкция есть значение, представленное упакованной структурой. Операнды внутри инструкции могут быть знаковые или беззнаковые, которые представлены объединением из двух типов. Входными значениями командного регистра являются операнды, код операции и флаг показывающий знак операндов. Выходом модели является массив из 32 инструкций.

Example 5-2: Using arrays of structures to model an instruction register

```
package definitions;
  typedef enum {ADD, SUB, MULT, DIV, SL, SR} opcode_t;
  typedef enum {UNSIGNED, SIGNED} operand_type_t;
  typedef union packed {
```

```
    logic [31:0]         u_data;
    logic signed [31:0] s_data;
  } data_t;

  typedef struct packed {
    opcode_t         opc;
    operand_type_t  op_type;
    data_t           op_a;
    data_t           op_b;
  } instr_t;

endpackage

import definitions::*;   // import package into $unit space

module instruction_register (
  output instr_t [0:31] instr_reg, // packed array of structures
  input  data_t         operand_a,
  input  data_t         operand_b,
  input  operand_type_t op_type,
  input  opcode_t       opcode,
  input  logic [4:0]    write_pointer
);

  always @(write_pointer) begin
    instr_reg[write_pointer].op_type = op_type;
    instr_reg[write_pointer].opc     = opcode;

    // use op_type to determine the operand type stored
    // in the input operand union
    if (op_type == SIGNED) begin
      instr_reg[write_pointer].op_a.s_data = operand_a.s_data;
      instr_reg[write_pointer].op_b.s_data = operand_b.s_data;
    end
    else begin
      instr_reg[write_pointer].op_a.u_data = operand_a.u_data;
      instr_reg[write_pointer].op_b.u_data = operand_b.u_data;
    end
  end
endmodule
```

5.4 Цикл обхода массива foreach

SystemVerilog добавляет цикл foreach, который может быть использован для обхода элементов одно- или многомерного массива, без значения размера массива. Аргументом для цикла **foreach** является имя массива, за которым следует список переменных, заключённых в квадратные скобки и разделённых запятой. Каждая переменная соответствует одной из размерностей массива.

```
int sum [1:8] [1:3];

foreach ( sum[i,j] )
  sum[i][j] = i + j;   // initialize array
```

Отображение переменных цикла на индексы массива определяется размерностью, как описано в секции 5.3.8. Несколько переменных цикла создают вложенные циклы, обход которых осуществляется через индексы. Внешние циклы соответствуют наиболее низким индексам. В примере выше, обход внешнего цикла осуществляется через i , а внутреннего через j .

Переменные цикла foreach не объявляются

Нет необходимости определять переменную цикла для каждой размерности массива. Вместо размерности можно показать позицию переменной, используя две запятые, без имени переменной. Пустой цикл обход массива не осуществляет. Смежные переменные пустого цикла в конце списка могут быть пропущены без дополнительных запятых.

В следующем примере функция проверяет биты для каждого байта в 128 битном векторе. Вектор представлен как двумерный упакованный массив из 16 8-битных элементов. Цикл foreach определяется одной переменной, которая представляет первую размерность ([15:0]) массива.

```
function [15:0] gen_crc (logic [15:0] [7:0] d);
  foreach (gen_crc[i]) gen_crc[i] = ^d[i];
endfunction
```

Переменные цикла являются автоматическими, только для чтения и локальными. Тип каждой переменной цикла объявлен неявно, чтобы быть совместимым с типом индекса массива, который будет типом **int** для массивов, представленных в этой книге. (В SystemVerilog также имеются ассоциативные массивы, которые могут использовать различные типы для индексов. Ассоциативные массивы не синтезируемы).

5.5 Системные функции запроса массива

Специальные системные функции для работы с массивами

SystemVerilog добавляет несколько специальных системных функций для работы с массивами. Эти системные функции позволяют писать верификационные программы, которые работают с массивом любого размера. Также они могут быть полезны в абстрактных моделях.

```
$dimensions(array_name)
```

- Возвращает число размерностей в массиве (возвращает 0 если объект не является массивом)

```
$left(array_name, dimension)
```

- Возвращает число наиболее значащих (msb) бит размерности. Размерности начинаются с числа 1, отсчёт ведётся с самой левой неупакованной размерности. После самой правой неупакованной размерности, число размерности продолжается на самой левой упакованной размерности, и заканчивается на самой правой упакованной размерности. Для массива:

```
logic [1:2][7:0] word [0:3][4:1];
```

```
$left(word,1) will return 0
$left(word,2) will return 4
$left(word,3) will return 1
$left(word,4) will return 7
```

```
$right(array_name, dimension)
```

- Возвращает число наименее значащих (lsb) бит размерности. Размерности пронумерованы также как в **$left**.

```
$low(array_name, dimension)
```

- Возвращает число младших бит размерности, которые могут быть либо наиболее значащими (msb) либо наименее значащими (lsb). Размерности пронумерованы также как в **$left.** Для массива:

```
logic [7:0] word [1:4];
```

`$low(word,1)` returns 1, and `$low(word,2)` returns 0.

```
$high(array_name, dimension)
```

- Возвращает число старших бит размерности, которые могут быть либо наиболее значащими (msb) либо наименее значащими (lsb). Размерности пронумерованы также как в **$left.**

```
$size(array_name, dimension)
```

- Возвращает общее число элементов ($high - $low + 1). Размерности пронумерованы также как в **$left.**

```
$increment(array_name, dimension)
```

- Возвращает 1 если $left больше или равно $right, и -1 если $left меньше $right. Размерности пронумерованы также как в **$left**.

Следующий фрагмент кода показывает, как некоторые системные функции для массивов могут использоваться для инкремента массива, без значения размерности массива.

```
logic [3:0][7:0] array [0:1023];

int d = $dimensions(array);
if (d > 0) begin   // object is an array
  for (int j = $right(array,1);
            j != ($left(array,1)
                  + $increment(array,1) );
            j += $increment(array,1))
    begin
      ...  // do something
    end
end
```

В этом примере:

```
$right(array,1) returns 1023
$left(array,1) returns 0
$increment(array,1) returns -1
```

Цикл for разворачивается в:

```
for (int j = 1023; j != -1; j += -1)
  begin
    ...
  end
```

Пример выше можно реализовать, используя цикл **foreach**:

```
foreach ( array[j] )
  begin
    ...
  end
```

Цикл **foreach** обсуждался ранее в этой главе, в секции 5.4. Когда обход охватывает весь массив и количество циклов известно, то лучше использовать цикла **foreach,** как наиболее простой и интуитивный, чем функции запроса массива. Преимущества функций запроса массива в том, что они предоставляют больше информации о том, как объявлен массив,

сколько размерностей массив содержит. Эта информация может быть использована для обхода частей размерностей.

Рекомендации по синтезу

Функции запроса массива синтезируемы, предоставляется фиксированный размер массива, и аргумент числа размерностей есть константа, или не определён. Это исключение не поддерживается компилятором при использовании задач или функций. Цикл foreach также синтезируем, предоставляется фиксированный размер массива.

5.6 Системная функция $bits “sizeof”

$bits такая же sizeof функция как в языке C

SystemVerilog добавляет системную функцию **$bits**, которая возвращает количество бит, представленных выражением. Выражение может содержать любой тип значения, включая упакованный и неупакованный массивы, структуры, объединения и литеральные числа. Синтаксис **$bits**:

```
$bits(expression)
```

Примеры использования **$bits**:

```
bit    [63:0] a;
logic  [63:0] b;
wire   [3:0][7:0] c [0:15];
struct packed {byte tag; logic [31:0] addr;} d;
```

- `$bits(a)` returns 64

- `$bits(b)` returns 64
- `$bits(c)` returns 512
- `$bits(d)` returns 40
- `$bits(a+b)` returns 128

Рекомендации по синтезу

Системная функция **$bits** синтезируема, аргумент функции не является динамическим массивом. Возвращаемое значение может быть определено статически и рассматривается как простое литеральное значение для синтеза.

5.7 Динамические, ассоциативные, разреженные массивы и строки

SystemVerilog добавляет типы динамического массива в Verilog:

- Динамические массивы
- Ассоциативные массивы
- Разреженные массивы
- Строки (символьные массивы)

Эти специальные типы массива не синтезируемы.

Динамические массивы не синтезируемы, и предназначены для использования в верификационных программах и для моделирования на высоких уровнях абстракции. Эта книга сфокусирована на написание моделей на SystemVerilog, которые синтезируемы. Поэтому, эти типы массива не рассматриваются в следующих секциях. Информация по этим типам массива может быть найдена в книге *SystemVerilog for Verification.*

5.8 Резюме

SystemVerilog добавляет возможность представлять сложные данные как единое целое. Структуры инкапсулируют переменные в один объект. К структуре можно обращаться как к единому целому. Обращение к членам структуры осуществляется по имени. Структуры могут быть упакованными, что позволяет манипулировать ими как единым вектором. Объединения предоставляют возможность моделировать единое пространство хранения на абстрактном уровне, где хранимое значение может быть представлено любым типом.

SystemVerilog также расширяет возможности массивов. Присвоение значений массивам может осуществляться как единому целому. Весь массив или части (slice) одной размерности массива могут быть скопированы в другой массив. Базовое объявление вектора расширено до многомерного, в форме упакованного массива. Упакованный массив, по сути есть вектор, который может иметь несколько подполей. SystemVerilog также предоставляет новые системные функции запроса массива, которые используются для определения характеристик массива.

Глава 11 содержит более расширенные примеры использования структур, объединений и массивов для представления сложных данных в кратком, интуитивном и эффективном стиле, и полностью синтезируемых.