SystemVerilog - Типове Данни и Обекти

# Типове Данни

**Тип данни** (data type) - определя набор от стойности и операции, които могат да бъдат извършвани с тях

Типовете данни се използват при деклариането на **обекти** - например променливи, параметри или вериги (nets). Също така, те участват и при декларирането на **потребителски** типове данни.

## Целочислени типове данни

shortint 2-state data type, 16-bit signed integer

int 2-state data type, 32-bit signed integer

longint 2-state data type, 64-bit signed integer

byte 2-state data type, 8-bit signed integer or ASCII character

bit 2-state data type, user-defined vector size, unsigned

**logic 4-state data type, user-defined vector size, unsigned**

reg 4-state data type, user-defined vector size, unsigned (same as logic)

integer 4-state data type, 32-bit signed integer

time 4-state data type, 64-bit unsigned integer

Базови стойности

0 — логичека нула или неистина

1 — логическа единица или истина

x — неизвестна стойност

z — високоимпедансно състояние

Деклариране

**logic** reset\_n;

**logic** [17:0] data\_bus;

Представяне на целочислени константи

**n’Fddd…**

където

n - е броя на битовете

F - бройна система : b (двоична), o (осмична), d (десетична),h (шестнайсетична).

ddd - цифри, легални за указаната бройна система

Примери

2’b01 // 2-битово двоично число

20’hB36F // 20-битово шестнайсетично число

16’b0101\_1110\_0101\_0011 // 16-битово двоично число

267 // 32-битово десетично число

**Сила**

Освен **стойност**, някой обекти в SystemVerilog имат и **сила:**



## Тип данни string

Представляват подредена колекция от символи. Дължината им може да варира по време

симулация.

**string** s0 = "Hello World";

bit [11:0] b = 12'ha41;

**string** s2 = string'(b); // sets s2 to 16'h0a41

## Дефинирани от потребителя (user-defined) типове данни

Пример

typedef logic [3:0] bcd\_t;

module bcd\_adder(

 input bcd\_t op1, op2,

 output bcd\_t summ

);

## Изброими типове данни

Пример

typedef enum logic [2:0] {ADD,SUB,MULT,DIV,SL,SR} opcode\_t;

Пример

вариант 1 - именован изброим тип

 typedef enum logic [2:0] {

 IDLE = 2'b00,

 OP = 2'b01,

 DONE = 2'b11,

 XX = 2'bxx} foo\_t;

 **foo\_t** state, state\_next;

вариант 2 - анонимен изброим тип

enum logic [1:0] {

 IDLE = 2'b00,

 OP = 2'b01,

 DONE = 2'b11,

 XX = 2'bxx} state, state\_next;

always\_ff @(posedge clock, posedge reset)

if (reset) begin

 state <= IDLE;

 n <= 0;

end

else begin

 state <= state\_next;

 n <= n\_next;

end;

## Константи

Константите са именувани обекти, чиите стойност не се променят.

parameter, localparam

Пример

module counter #(**parameter N = 8**) (

 input clock,

 input reset,

 output logic [N-1:0] cnt

);

always\_ff @(posedge clock, posedge reset)

 if (reset)

 cnt <= 0;

 else

 cnt <= cnt+1;

endmodule

module counter\_test\_2;

 **localparam size = 2;**

 logic clock, reset;

 logic [size-1:0] cnt;

 // Instantiate the Unit Under Test (UUT)

 counter **#(.N(16))** uut (.\*);

 initial begin

 clock = 0;

 forever #50 clock = ~clock;

 end

 initial begin

 reset = 1;

 #100 reset = 0;

 @(negedge clock);

 assert (cnt == 0) else $error("cnt: expected=%1d, actual=%1d", 0, cnt);

 @(negedge clock);

 assert (cnt == 1) else $error("cnt: expected=%1d, actual=%1d",1,cnt);

 @(negedge clock);

 assert (cnt == 2) else $error("cnt: expected=%1d, actual=%1d",2,cnt);

 @(negedge clock);

 assert (cnt == 3) else $error("cnt: expected=%1d, actual=%1d",3,cnt);

 @(negedge clock);

 assert (cnt == 0) else $error("cnt: expected=%1d, actual=%1d",0,cnt);

 $display("Test passed");

 $finish;

 end

endmodule

# Типове Обекти

## Вериги и променливи

Съществуват два основни типа обекти: **вериги** (nets) и **променливи** (variables). Те се различават по това как им се присвояват стойности и как съхраняват присвоените им стойности.

**Веригите** получават стойност от:

* оператори за напрекъснато присвояване (assign)
* изходи на примитиви
* портове на модули.

Веригите могат да имат повече от един драйвер.

**Променливите** получават стойност от един или няколко оператора за **процедурно присвояване**.

**Типове вериги**

**Веригите моделират физически връзки** между структурните единици на проекта - например логически елементи.

Една верига **не съхраняват** присвоената и стойност. Нейната стойност се определя от стойностите на нейните драйвери, като се използва специфична разрешаваща функция.

Верига, която не е свързана с драйвер, се намира във високоимпедансно състояние (z).

Типът на веригата определя каква разрешаваща функция ще се използва при определяне на състоянието и.



**Деклариране на вериги**

**<тип верига> <тип данни> <размерност> <име>**

**wire** logic [7:0] data;

**wire** xyz; // implicit **logic** data type

**wire** [15:0] bus;

**Деклариране на променливи**

**Променливите се изполват за моделиране на запомнящи елементи.** Променливата помни стойността, която и е присвоена до следващото присвяване. Присвояване на променлива се извършва в процедура.

Примери

shortint s1, s2[0:9];

logic [15:0] register;

logic clear, enable;

## Масиви

**Пакетирани и непакетирани масиви**

пакетиран масив - размерността е указана **преди** името на обекта.

непакетиран масив - размерността е указана **след** името на обекта.

bit [7:0] c1; // пакетиран масив от 8 бита

real u [7:0]; // непакетиран масив, чиито елемети са от тип real

Едномерните пакетирани масиви се наричат **вектори**.

**Пакетирани масиви**

Пакетираният масив е механизъм за разделяне на елементите на вектор на групи, които да бъдат адресирани като елементи на масив. Пакетирания масив се представя като **последователни битове в паметта**.

Пакетираният масив може да се състои само от **еднобитови типове данни: bit или logic.**

Пример

logic [639:0] input\_stream; // пакет от 640 бита

logic [39:0][15:0] packet; // 40 16-битови думи

packet = input\_stream; // присвояване на всички думи от packet

data = packet[24]; // избор на една 16-битова дума

tag = packet[3][7:0]; // избор на част от дума

**Непакетирани масиви**

Непакетираният масив може да се състои от произволни типове данни.

Елементите на такъв масив **не са разположени в последователни клетки от паметта**.

int Array[0:7][0:31]; // array declaration using ranges

int Array[8][32]; // array declaration using sizes

**Операции върху масиви**

Операции върху пакетирани и непакетирани масиви

четене/запис на целият масив

 A = B

четене/запис на сегмент (slice) от масив

 A[i:j] = B[i:j]

четене/запис на елемент от масив

 A[i] = B[j]

Сравняване за равенство на масиви или сектори от масиви

 A == B

 A[i:j] != B[i:j]

(\*) Масивите A и B са с еднакви размерности и типове на елементите.

Следните опрации са разрешени само за **пакетирани масиви**.

 присвояване от цяло число

 А = 8’h7f;

 аритметични операции

 A + 2

**Описание на памети**

Пример - памет от 256 8-битови думи с индекси от 0 до 255

logic [7:0] mema [0:255];

mema[5] = 0; // запис в думата на адрес 5

data = mema[addr]; // четене на адрес индексиран от addr

**Многомерни масиви**

Многомерните масиви са “масиви от масиви”.

В декларацията на маногомерен масив, дименсиите преди името на обекта се наричат пакетирани, а тези след него - непакетирани.

Пример

// 10 елемента от по четири 8-битови байта (всеки елемент е пакетиран в 32 бита)

bit [3:0] [7:0] data [1:10];

data[9] = data[8] + 1; // събиране на 32 битови елементи

joe[7][3:2] = joe[6][1:0]; // копиране на два байта

При адресирането, пакетираните дименсии ([3:0] [7:0]) следват непакетираните ([1:10]).

## Структури

Служат за групиране на логически свързани сигнали

Пример

typedef struct { // structure definition

 logic [31:0] op\_a, op\_b;

 logic [ 7:0] opcode;

 logic [23:0] address;

} instruction\_word\_t;

instruction\_word\_t instr;

Пример - аномимна структура

struct {

 logic [31:0] op\_a, op\_b;

 logic [ 7:0] opcode;

 logic [23:0] address;

} instr;

Пример - адресиране на елементи от структура

instr.op\_a = 20;

instr.op\_b = 5;

instr.opcode = 8'h0f;

instr.address = 0;

или

instr = '{20, 5, 8'h0f, 0};

или

instr = '{op\_a:20, op\_b:5, address:0, opcode:8'h0f};

## Обединения

Обединение е област от паметта, която може да има различни представяния от различни типове.

Пример

module example\_union;

 union {

 int x;

 int unsigned y;

 } data;

 initial begin

 data.x = -5;

 $display("data is %d", data.x);

 data.y = -5;

 $display("now data is %d", data.y);

 end

endmodule

Симулация

run -all

data is -5

now data is 4294967291

exit