T

Объектно=ориентированное программирование в SystemVerilog
Определение класса в SystemVerilog
Понятие класса позволяет объединять данные вместе с подпрограммами, манипулирующими ими.
Простейший синтаксис:

```
class classname
//список аргументов
endclass *
```

Листинг Представляет класс BusTran, описывающий типичный сетевой пакет, который включает адрес источника и назначения, а также массив значений данных. Класс BusTran содержит функции для вывода содержания пакета и вычисления CRC (cyclic redundancy check) данных. Для того чтобы можно было легко определить начало и конец декларации класса, используются метки.

```
class BusTran;
bit [31:0] addr, crc, data[8];
function void display;
$display("BusTran: %h", addr);
endfunction : display
function void calc_crc;
crc = addr ^ data.xor; //xor в данном контексте это метод массива
endfunction : calc_crc
endclass : BusTran
```

В SystemVerilog классы могут быть объявлены в program, module, package или в другом месте как самостоятельные программные единицы . Они могут быть использованы в программах и модулях.
Многие группы тестировщиков размещают автономный класс или группу родственных классов в файл.
Набор классов оформляется в пакет package. Например, вместе в один пакет могут быть сгруппированы все транзакции SCS1/ATA. Пакет может быть скомпилирован отдельно, независимо от теста описания системы. Неродственные классы, такие, как транзакции, s**coreboards** (трафареты таблиц) или различные протоколы, целесообразно оформлять в отдельные файлы.

Терминология ООП
1. Класс - основной блок, содержащий подпрограммы и переменные. Аналогом в Verilog является модуль.
2. Объект - копия класса. Подобно созданию в Verilog копии модуля возможности его последующего использования.
3. Определитель (Handle) - указатель на объект. В Verilog используется имя копии модуля для обращения к сигналам и подпрограммам вне модуля. Указатель в ООП подобен адресу на объект, но он хранится в указателе, который может ссылаться на любой тип.
4. Свойства (Property) - переменные, сохраняющие данные. В Verilog - это сигналы: регистры или цепи.
5. Метод (Method) - процедурный код для манипулирования переменными, оформляемый в виде задач и функций. В Verilog модули содержат задачи и функции, а также блоки initial и always.
6. Прототипы (Prototype) - заголовки подпрограмм, содержащие имена, типы

T

и списки аргументов. Тело подпрограммы содержит выполняемый код.

. Создание новых объектов
В следующем примере b - это *определитель* (handle) объекта типа BusTran.

```
BusTran b; // Декларация объекта класса
b = new; // Выделение места под объект BusTran
```

В момент декларация определитель b инициализируется нулевым указателем null. Затем для создания объекта BusTran может быть вызвана предопределенная функция new, которая выделяет пространство под BusTran, инициализирует переменные значением по умолчанию (0 - для 2-значных переменных и X - для 4значных) и возвращает адрес объекта. В SystemVerilog для каждого класса автоматически создается функция new, предназначенная для выделения памяти и инициализации значений объектов.

SystemVerilog позволяет пользователю создавать свою функцию new
Она не имеет типа, поскольку всегда возвращает объект того же типа, что и класс.

Листинг . Определение пользовательской функции new

```
class BusTran;
logic [31:0] addr, crc, data[8];

function new;
addr = 3;
foreach (data[i])
data[i] = 5;
endfunction
endclass

  bus tran b;
 b=new;
```

Здесь функция присваивает переменным addr и data определенные значения, но оставляет для переменной crc значение по умолчанию X (SystemVerilog выделяет место для объекта автоматически). Можно указывать для аргументов функции значения по умолчанию, чтобы сделать конструктор более гибким .

Листинг . Функция new с аргументами

```
class BusTran;
logic [31:0] addr, crc, data[8];
function new(logic [31:0] addr=3, d=5);
this.addr = addr;
foreach (data[i])
data[i] = d;
endfunction
endclass
initial begin
```

T

```
BusTran b;
b = new(10); // поле data получает значение 5 no умолчанию
end
```

. **Освобождение объектов**.
Когда дескриптор прекращает указывать на объект, то он удаляется из памяти. Например:

```
BusTran b; // Создание определителя
b = new; //Выделение памяти под объект
b = new; // Выделение памяти под второй объект, первый удаляется
b = null; // Удаление второго объекта
```

Во второй строке вызов конструктора new выделяет память под объект и присваивает указатель на него дескриптору Ь. Следующий вызов конструктора new создает новый объект и записывает адрес на него в Ь, переназначая предыдущее значение и удаляя первый объект. В четвертой строке происходит удаление второго объекта.
SystemVerilog не удаляет объект, на который указывает дескриптор автоматически Удаление необходимо делать «вручную» с помощью оператора null.

.**Использование объектов**.
Для доступа к переменным и подпрограммам класса можно использовать обозначение точки Например:

```
BusTran b; // Создание определителя BusTran
b = new; // Вызов конструктора BusTran
b.addr = 32'h42; // Присвоение значения переменной
b.display(); // Вызов подпрограммы
a=b.addr;
```

. **Подпрограммы или методы класса**
Подпрограммы или методы (task или function) класса определяются внутри него. Листинг 20.7 представляет пример определения подпрограммы display() для классов BusTran и PCI_Tran. SystemVerilog вызывает подпрограмму, основываясь на типе дескриптора.
Листинг 20.7.

```
class BusTran;
bit [31:0] addr, crc, data[8];
function void display();

$display("@%0d: BusTran addr=%h, crc=%h" addr, crc);
$write("\tdata[0-7]-");
foreach (data[i]) $write(data[i]);
$display();
endfunction
endclass
class PCI_Tran;
bit [31:0] addr, data;
function void display();
$display("@%0d: PCI: addr=%h, data=%h", addr, data);
```

T

```
endfunction
endclass

BusTran b;
PCI_Tran pc;
initial begin
b = new(); // Конструктор BusTran
b.display(); // Вывод значения BusTran
pc = new(); // Конструктор транзакции PCI
pc.display();//  Вывод значений транзакции PCI
end
```

Подпрограммы в классе всегда используют автоматическое распределение памяти, следовательно, они не нуждаются в применении модификатора automatic.

Простой пример

```
// jk-flipflop as a class
module count3(clock,r, d_out);
input logic clock,r;
output logic[1:0] d_out;

class rjk;
 logic j,k,db;

  task  reset();
        db='b0;
    endtask;
  function  sync_log();
          case ({j,k})
                  'b00: return (db);
                  'b01: return ('b0);
                  'b10: return ('b1);
                  'b11: return (!db);
          endcase;

      endfunction;
  endclass;

  rjk v2=new;
    rjk v1=new;

  initial
    begin r='b0 ;clock='b0;
        #10;
          r='b1;
          forever
              begin #10;
                    clock=!clock;
              end;
    end;
```

T

```
 always _comb
   begin
v1.j=!v2.db;
  v1.k='b1;
    v2.j=v1.db;
    v2.k='b1;//v1.db;
    d_out={v2.db,v1.db};
  end;
  always @( posedge clock or negedge r)
      begin
        if(!r) begin v1.reset; v2.reset; end
        else //if (clock ='b1)
              begin
               v1.db=v1.sync_log();
               v2.db=v2.sync_log();
             end ;
     end; // always
endmodule
```

В строгом ООП доступ к переменным осуществляется только через публичные (public) методы, например get() и put(). Это повышает стабильность разрабатываемых программ, исключая случайное изменение полей, однако снижает гибкость, необходимую для построения testbench. Поэтому для языков верификации лучшим решением является наличие прямого доступа к переменным класса.

Статические и глобальные переменные

В SystemVerilog имеется возможность создавать статическую переменную, являющуюся общей для всех копий класса. В листинге 5 статическая переменная count хранит количество созданных на данный момент объектов . Она инициализируется в 0 при декларации, потому что в начале моделирования отсутствуют транзакции. Каждый раз, при создании нового объекта, значение count увеличивается на 1.

. Переменная id не является статической, поэтому каждый объект BusTran имеет свою собственную копию, и нет необходимости в создании глобальной переменной для count.

Листинг.5. Класс со статической переменной

```
class BusTran;
            static int count = 0; //Счетчик создаваемых объектов
            int id; // Уникальный ID копии объекта
            function new;
            id = count++; // Установка ID с помощью счетчика
            endfunction
endclass
BusTran ЫI, b2;
initial begin
            b1 = new; // Первая копия, id=0
            Ь2 = new; // Вторая копия, id=1
            $display("Second id=%d, count=%bM, b2.id, b2.count);
end
```

T

Статические переменные, как правило, инициализируются в момент декларации. Нельзя использовать для начальной инициализации функцию new, поскольку она вызывается каждый раз при создании новой копии класса. Для более сложной инициализации может быть использован блок initial. Необходимо только гарантировать, что статическая переменная инициализирована до создания первой копии класса.

. **Определение подпрограмм за пределами класса.**
Эмпирическое правило гласит: «Чтобы код программы был понятным, он должен быть не больше одной страницы».
В SystemVerilog подпрограмму можно разбивать на прототип (имя подпрограммы и аргументы) внутри класса и тело (процедурный код), которое может быть записано вне класса. В этом случае перед именем подпрограммы в классе добавляется слово extern. При определении тела подпрограммы перед ее именем ставится имя класса, отделенное от него двумя двоеточиями (::).
Листинг 20.8. Внешнее определение подпрограмм

```
class BusTran;
        bit [31:0] addr, crc, data[8};
        extern function void display();
endclass

          function void BusTran::display();
          $display("@%0d: BusTran addr=%h, crc=%h", addr, crc);
          $write('\tdata[0-7]=");
          foreach {data[i]) $write(data[i]);

          $display();
endfunction
class PCi_Tran;
        bit [31:0] addr, data;
          extern function void display();
endclass
function void PCI_Tran::display();
        $dispiay("@%0d: PCI: addr=%h, data=%hH, addr, data);
endfunction
```

. Правила границ видимости
SystemVerilog придерживается основных правил Verilog, но имеет ряд дополнений. Границы видимости определяются блоками кода, такими как модуль, программа, задача, функция, класс или блок begin-end. Для циклов for и foreach автоматически создается блок. Таким образом, индексные переменные являются локальными для этой области видимости.
Новые переменные могут быть определены внутри блока. SystemVerilog допускает создание переменных в безымянных блоках begin-end. Имя может быть относительным или абсолютным, инициальзация записываеися в форме в форме

$root.<идентифиатор>

Где идентификатор это имя иерархического блока , на котороый распространяется видимость
Для относительных имен SystemVerilog просматривает список границ видимости, пока не найдет совпадение.

Т

В следующем примере (листинг 20.9) применяется одно и то же имя в различных областях видимости. Имя limit соответствует глобальной переменной, переменной программы, класса, задачи и локальной переменной в блоке initial.

Листинг 20.9. Границы видимости имен

```
int limit;        // $root.limit
program p;
    int limit; //$root.p.limit
        class Foo;
            int limit, array[]; // $root.p.Foo.limit
            task print (int limit); // $root.p.Foo.print.limit

              for (int i=0; i<limit; i++)
            $display("%m: array[%0d]=%0d", i, array[i});
                endtask
          endclass
    initial begin

            int limit = $root.limit; // $root.p.$unnamed.limit
            Foo bar,
            bar = new;
            bar.array = new(limit};
            bar.print (limit);
       end
endprogram
```

Следует быть осторожным при определении переменных. Если компилятор не находит переменную внутри блока, он ищет ее извне. Это может привести к одновременному использованию одной переменной несколькими потоками, и ошибка будет обнаружена только на этапе моделирования.

T

Листинг 20.11. Неправильное использование общей переменной программы

```
program bug;
  class Buggy;
            int data[l0];
            task transmit;
                fork
                for (i=0; t<10; i++) //Переменная i не объявлена в классе
                send(data[i]);
                join_none
                endtask
        endclass
int i; // программный уровень
Buggy b;
event receive;
initial begin
          b = new;
          for (i=0; i<10; i++) b.data[i] = i;
          b.transmit;
          for (i=0; i<10; i++)
          @(recieve) $display(data[i]);
end
endprogram
```

Использование одного класса в пределах другого

Один класс может содержать копию другого, используя дескриптор в качестве объекта, подобно тому, как модуль в Verilog может содержать другой модуль. Основная идея применения вложений заключается в возможности повторного использования. Например,
каждая транзакция может содержать блок статистики с временными метками начала и завершения транзакции, а также информацию обо всех транзакциях

T

Листинг 20.12. Использование класса в классе

```
// Класс Statistics,
class Statistics;

time startT, stopT; // Время транзакции
static int ntrans = 0;// Счетчик транзакций
static time total_elapsed_time = 0;
     function time howlong;
          howlong = stopT - startT;
          ntrans++;
          total_elapsed_time+=howlong;
     endfunction

        function void start;
        startT = $time;
        endfunction
endclass
// Использование класса Statistics в классе BusTran
class BusTran;
          bit [31:0] addr, crc, data[8];
          Statistics stats; //Определитель класса Statistics
  function new();
          stats = new(); // Выделение памяти под объект stats
  endfunction
task create_packet();
……..// Заполнение пакета данными
          Stats.start(); // Передача пакета
endtask
endclass
```

Внешний класс BusTran может обращаться к элементам класса Statistics с помощью иерархического синтаксиса, например stats.start. Необходимо помнить, что значение дескриптора вложенной копии объекта равно null и вызов start приведет к ошибке. Инициализацию копий вложенных классов лучше сделать в конструкторе внешнего класса BusTran.

Порядок компиляции

Если необходимо выполнить компиляцию класса до определения вложенного в него класса, то последний может быть объявлен с помощью конструкции typedef, например, следующим образом:

```
typedef class Statistics; //Декларация класса нижнего уровня
class BusTran;
Statistics stats; // Использование класса Statistics
endclass
class Statistics; // Описание класса Statistics

endclass
```

Т

, Понятие динамического объекта

При вызове подпрограммы со скалярными переменными и использованием ключевого слова ref SystemVerilog передает адрес. Иначе - SystemVerilog передает копию переменных. При этом их изменения в подпрограмме не повлияют на оригинальные значения переменных.
В примере, представленном листингом 13, в блоке initial создается объект BusTran, который вызывает задачу transmit с определителем, указывающим на объект. С помощью определителя задача transmit может читать и записывать данные в объект. Тем не менее, если transmit попытается модифицировать дескриптор, результат не будет виден в блоке initial, поскольку не объявлен как ref. Подпрограмма может модифицировать объект, даже если аргумент дескриптора не имеет модификатора ref. Как показано ниже, функция transmit может записать timestamp в объект. Если нет необходимости модификации объекта в подпрограмме, можно передать его копию.

Листинг 13. Пассивные объекты

```
// Передача пакета в 32-битовую шину

task transmit(BusTran bt);
        CBbus.rx_data <=bt.data;
        bt.timestamp = Stime;
endtask
BusTran b;
initial begin
        b = new();// Выделение памяти под объект
        b.addr = 42; // Инициализация значения
        transmit(b); .. Передача объекта в задачу
end
```

T

Изменение определителя (указателя) в задаче

Частой ошибкой кодирования является пропуск ключевого слова ref в описании аргументов подпрограммы. Например, в следующем коде аргумент b не имеет модификатора ref, таким образом, любые его изменения не будут видны в вызывающем коде,

```
task create_packet(BusTran bt); // Неправильная декларация аргументов
                                // функции, пропущен ref

bt = new();
bt.addr = 42;
……./// Initialize other fields
endtask
BusTran b;
initial begin
// фактический параметр задачи  “b”  вроде бы замещает    “bt”
create_packet(b); // Вызов подпрограммы с ошибкой
$display(b.addr); // Приводит к ошибке, потому что b=null
end
```

Хотя create_packet изменил аргумент bt, определитель b должен оставаться равным null. Чтобы исправить ситуацию, аргумент bt должен быть объявлен как ref.

```
task create_packet(ref BusTran bt);// Правильно
……
endtask
```

T

Массивы определителей
Testbench может оперировать многими объектами. Можно создать массив дескрипторов, каждый из которых ссылается на свой собственный объект. Пример представляет способ хранения десяти транзакций шины в массиве:

```
// Использование массива дескрипторов
task generator();

BusTran barray[10];

foreach (barray[i]) begin

barray[i] = new(); // Конструктор каждого объекта
transmit(barray[i]);
end
endtask
```

Массив barray создан из дескрипторов, но не объектов, поэтому необходимо создавать каждый объект массива до его использования. Нет возможности применять функцию new для целого массива.

. Копирование объектов
Использование new для копирования объектов - простое и надежное решение. Создается новый объект, а значения всех переменных копируются в него из существующего объекта.

```
class BusTran;

bit [31:0] addr, crc, data[8];
endclass
BusTran src, dst;
initial begin

src = new; // Создание первого объекта
dst = new src; // Создание копии с помощью new
end
```

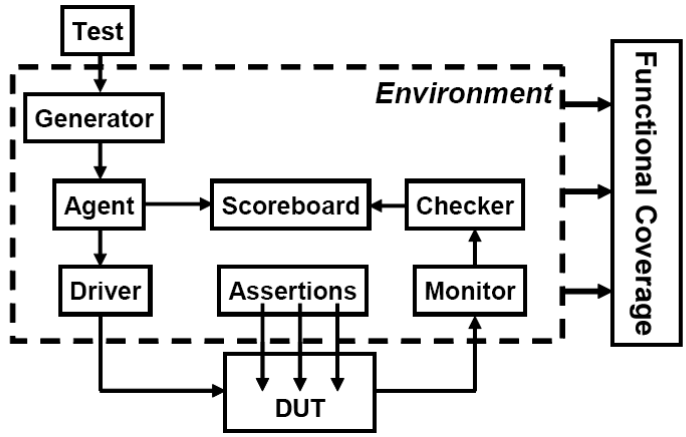
. Публичные или приватные объекты классов
Обычные языки программирования, использующие идею ООП, предполагают, что все объекты класса являются приватными (невидимыми за пределами класса), если они не объявлены как public. Для построения testbench необходима большая гибкость, поэтому в SystemVerilog все объекты являются публичными (public), если только они не объявлены как приватные (private).

Наследование и подклассы
Ключевым свойством классов SystemVerilog является наследование, которое позволяет создавать классы, основанные на других аналогах. Таким образом, они автоматически включают некоторые члены родительского класса. Например, в следующем коде Circle наследует свойства класса Shapes. Часть базового класса всегда создается первой и удаляется последней. Часть подкласса конструируется последней и удаляется первой.

T

```
class Shapes;
// Каждый объект должен содержать размер и начало координат
int totaLsize;
int [8][2] angles[];
int start_x, start_y;
bit[7:0] color;
//Для каждой фигуры необходимы функции рисования и смены цвета
task draw();
……
endtask
// Определение кода в данном месте является необязательным
// для наследуемых методов, поскольку здесь они являются
// виртуальными и будут переписаны позже в классах-наследниках

task setcolor (bit [7:0] color);
this.color = color;
endtask
endclass

class Circle extends Shapes;
bit [30:0] circle_d_data;
function new(int size = 4);

total_size = size * 3;
endfunction
task draw();
// Код, выполняющий рисование круга, записывается здесь
$display("drawing a circle..An");
endtask
endclass
 class poligon extends Shapes;
int [8] x,y[];
function new(int num);
…
endfunction;

task draw();
// Код, выполняющий рисование многоугольника

endtask ;
endclass
```

T




```
interface bus ();

logic start,stop,start_read;
 logic [7:0] datain;
   logic [7:0] dataout;

modport stim//mode for stim
(
  output start,stop,start_read,
    output datain
    );

modport monit
(input start,stop,start_read, datain, dataout);
 modport syst //mode of DUT
( input  start,stop,start_read, input datain, output dataout);
endinterface

program clocks(clock);
 output logic clock;
 initial
    forever
      begin
          #5;
         clock=1'b0;
       #5;
         clock=1'b1;
      end;
endprogram;

module top;
   logic clock, start,stop, startoutput,stopoutput;
   logic [7:0] datain,dataout;
   bus bbus();
  clocks cl(clock);
  generator g1(.clock(clock),.t(bbus));
   monitor e1(.clock(clock),.sy(bbus),.startoutput(startoutput),.stopoutput(stopoutput));
  device d1(.clock(clock),.startoutput(startoutput),.stopoutput(stopoutput), .sy(bbus));
  endmodule;
```

T

```
class buff_trans;
  rand  int length;
  constraint db_len {length inside {[2:6]};}
 rand  logic [7:0] data_block [1:10];
 logic [7:0] crc;

task get ( input logic  start,stop,// this function only accepts data and translates
        input logic [7:0] datain,
        output logic [7:0]dataout,//
         output int terms,// holds number of transactions
          input int i//;
           );
                        if (start) crc=0;
                        else  begin  if (!stop) crc= crc +datain; //inthis case stopd is argument
                                 data_block[i]=datain ;
                                 terms=i;
                               end ;
endtask;

task send (
     output logic start,stop,
     output logic[7:0] data,
      input int i );

         stop= (i==length+1) ? 1:0; // stop issued after crc transmitted
         start= (i==0)  ? 1:0;
          data= (i<=length)? data_block[i]:crc;
          if (i==1)  crc=data_block[1];
          else if (i <=length && i!=0) crc=crc+ data_block[i];
 endtask;

endclass;
```

T

```
include "myclass.sv";
module generator  (input logic clock,
bus.stim t);
  parameter block_number= 4;// number of bytes in block
  integer k,w;
  buff_trans  p;
  initial
    begin
      t.stop=0;
      t.start=0;
      t.start_read =0;
  p = new();

 @ (posedge clock) ; //at least one clock must be executed
                // before real start
  repeat (block_number) //repeat for all proposed blocks
    begin
    assert(p.randomize()); // number of bytes in block
                           for (k=0; k<=p.length+1;k++)
                             begin   @(posedge clock)
                               // sent a unit from block to dut

                                p.send(t.start,t.stop,t.datain,k);
                              end;
         repeat (5)  @ (posedge clock);// pause
          t.start_read=1; // initiate transmission from dut
          @ (posedge clock);
          t.start_read=0;
          repeat(10) @ (posedge clock);//interval to realise reading from DUT

     end;//repeate
 end  ; //initial
endmodule;
```

T

```
 include "myclass.sv";
module device (input logic clock,output logic startoutput,stopoutput,
   bus.syst sy);
  int i,j;
  buff_trans device_buff;  //not necessary for pipe
 logic  v=0,stopd=0, w=0; // v=1 enables reception,W=1 emables transmission
 integer terms;

 logic[7:0] u;
 initial device_buff=new();//not necessary for pipes
 always @ (negedge clock)
begin if  (sy.start)
          begin  v=1;i=0;w=0;
           end
      else if (stopd) begin v=0;
              device_buff.length=terms-1; // assign field in object
                   end
               stopd<=sy.stop; //half clock delay to get crc
 end;// always

always @ (negedge clock) //acception
              //methods get and send are  not necessary for pipelined devices
               if (v)  begin
                          device_buff.get(sy.start,sy.stop,sy.datain,sy.dataout,terms,i) ;
                         i++;
                         u=device_buff.crc;
                        // ?????   operate accepted data unit
                     end;
always @(posedge clock)
         if (stopd && v) begin
                         j=1;
                       assert ( sy.datain==device_buff.crc)
                       $error("transmission error");
                      end  ;
always @(posedge clock) // transnission
             if (sy.start_read ) begin w=1; i=0;end
             else if (w && i<=terms)
                begin
                 device_buff.send (startoutput,stopoutput,sy.dataout,i);
                 i++;
               end
                else  w=0;
endmodule ;
```

T

```
include "myclass.sv";

module monitor(input logic clock,bus.monit sy,
          input logic startoutput,input logic stopoutput);
                buff_trans mon_buff,result_buff;
          logic  v=0,stopd=0, stopoutputd=0,w=0; // v=1 enables reception from generator,w=1 emables
          logic[7:0]dd;                    // reception from dut
          integer terms,i,j;
          initial begin mon_buff=new();
                result_buff=new();
                 end

    always @ (negedge clock) // accepting from generator
          begin stopd<=sy.stop; //half clock delay to get crc
              if  (sy.start )
                   begin  v=1;i=0;                    end
               else if (stopd) begin v=0;
                        mon_buff.length=terms-1; // assign field in object
                             end;
              if (v)  begin
                 mon_buff.get(sy.start,sy.stop, sy.datain,dd,terms,i) ;
                   i++      ;
                  //??? - prepare predicted propper results
                  end;
           end;// always

          always @ (negedge clock) //accepting fron dut
                begin   stopoutputd<=stopoutput;
                     if(startoutput) begin w=1;j=0;
                                 end
                     else if (stopoutputd) begin w=0;
                                 result_buff.length=terms-1; // assign field in object
                           //       if (addinional condition???) assert ( predicted==real ??)
                           //                                  $error("calculation error")
                                    end ;
                     if (w) begin
                             result_buff.get( startoutput, stopoutputd,sy.dataout,dd,terms,j);
                             j++      ;
                             end;
             end;// always

             endmodule
```