Conditionals and Loops

Conditionals and Loops

- *if–else–if construct*
- *case statement*
- *forever*
- *repeat*
- *for*
- *while*
- *do while*
- *foreach*
- *break,return,continue*

Inside operator

```
logic [2:0] a;
if ( a inside {3'b001, 3'b010, 3'b100} )
```

- Without the **inside** operator, the preceding **if** decision would likely have been coded as:

```
if ( (a==3'b001) || (a==3'b010) || (a==3'b100} )
```

- The set of values can also be an array.
- **int** arr [1024];

```
if ( 255 inside {arr} )
```

➢ Tests to see if the value of 255 occurs anywhere in an array called arr.

```
int arr[1024];

for (int i=0;i < arr.size();i++) begin
 if (arr[i] == 255)
 //true code
 else
 //false code;
end
```

Static casting:

```
typedef enum {NOP=0, ADD=1, SUB=2, MULT=3, DIV=4, AND=5, OR=6, XOR=7} opcode_t;
```

```
opcode_t opcode;
logic [2:0] data=3;

always@(instruction)begin

case (instruction)
1'b0 : opcode = NOP;
1'b1 : opcode = data;
endcase
end
```

Error: Illegal assignment
data is of type logic.
Opcode is of type enum

```
opcode_t opcode;
logic [2:0] data=3;

always@(instruction)begin

case (instruction)
1'b0 : opcode = NOP;
1'b1 : opcode = opcode_t'(data);
endcase
end
```

Data=3 , Member of enum which has value 3 is MULT.
So, value 3 converted to MULT and opcode assigned with MULT.

for loop

```
logic [31:0] mem [16];

for ( int i=0;   i < 16 ; i++ )
begin
mem[i]= '0; //reset memory
$display (" mem[%0d]=%0d ", i, mem[i]);
end
```

```
integer i;
for( i=0; i<16; i=i+1 )
```

While loop

- The while-loop repeatedly executes a statement as long as a control expression is true.
- If the expression is not true at the beginning of the execution of the while-loop, the statement shall not be executed at all.

```
logic y,a,b,c,cond=1;

initial begin
   while ( cond==1'b1)  begin
      #5
      y=a+b+c;
      $display(" Value of Y=%0d ",y);
   end  //end_of_while
      $display("After while loop");
end//end_of_initial
```

```
initial begin

#20 cond=0;
end
```

Infinite loops

```
bit ctrl;
int  data,mem[100],q[$];

initial begin //T1
   while(1)    begin
      $display("Start of while loop time=%0t",$time);
      data=$random;
      mem[addr]=data;
      q_outp.push_back(data);
    end  //end_of_while
   $display("After while loop");
end
```

Infinite loop

```
initial begin  //T2
#10 ctrl=1;
end


always @(data) begin
  if( ctrl==0) data=32;
end
```

```
Output:
Start of while loop time=0

Start of while loop time=0

Start of while loop time=0

Start of while loop time=0
```

Infinite loops

```
bit ctrl;
int  mem[100],q[$];

initial begin
  while(1)     begin
    #1
    $display("Start of while loop");
    mem[addr]=data;
    q_outp.push_back(data);
 end  //end_of_while
$display("After while loop");
end
```

While loop execute after every 1-time unit

Jump statements

```
bit done;
int  mem[100],q[$];

initial    begin
  while(1)      begin
    #1 $display("Start of while loop");
     mem[addr]=data;
     if (done == 1'b1) break;

     q_outp.push_back(data);
  end  //end_of_while
$display("After while loop");
end
```

```
initial begin
#50 done=1;
end
```

breaks out of while loop

Jump statements

```
bit skip;
int  mem[100],q[$];

initial    begin
  while(1)      begin
    #1 $display("Start of while loop");
     mem[addr]=data;
     if (skip == 1'b1) continue;

     q_outp.push_back(data);
  end  //end_of_while
$display("After while loop");
end
```

```
initial begin
#50 skip=1;
end
```

Skip to the next iteration of while loop

do while loop

- The do...while-loop differs from the while-loop in that a do...while-loop tests its control expression at the end of the loop.

```
logic y,a,b,c,cond=0;

initial begin
 do
   begin
    @(posedge clk);
     y=a+b+c;
    $display("Value of Y=%0d",y);
   end
while(cond==1'b1);

$display("after while loop");
end
```

```
logic y,a,b,c,cond=0;

initial begin
while(cond==1'b1) begin
      @(posedge clk);
      y=a+b+c;
      $display("Value of Y=%0d",y);
end
$display("after while loop");
end
```

repeat loop

- The repeat-loop executes a statement a fixed number of times.

```
initial begin
 repeat(10) @ (posedge clock);
 done=1;
end

initial begin
count=10;
 repeat (count) begin
   @(posedge clk);
   op   = {$random}%4;
   inp1 = $urandom_range(10,100);
   inp2 = $urandom;
 end
end
```

1) $random generates 32-bit signed random values, The random number is a signed integer; it can be positive or negative.
2) The following code generates numbers between –59 and 59:

```
reg [23:0] rand;
rand = $random % 60;
```

3) The following code generates numbers between 0 and 59:

```
reg [23:0] rand;
rand = { $random } % 60 ;
```

foreach loop

```
logic [31:0] arr [];
string str_arr [4] = '{"cnu","seenu","ravi","ramu"};
bit [7:0] multi_arr [8] [4]; //8 arrays and each has 4 elements

initial begin
arr=new[10];
foreach( arr[i] ) begin
 arr[i]= $uradom;
$display(" arr[%0d]=%0d",i,arr[i]);
end

foreach(str_arr[i])
 $display(" str_arr[%0d]=%0s",i,str_arr[i]);

foreach(multi_arr [ j , k] )
 multi_arr[j][k] = j*k;

end
```

```
for ( int i=0;    i< arr.size();      i++ )
```

Print each index and value

```
for (int j=0; j < 8 ; j++ ) //no of arrays
  for (int k=0; j < 4 ; k++ ) //no of elements per array
    multi_arr[j][k] = j*k;
```

Streaming Operators (pack/unpack)

```
module test1;
  bit [31:0] len, len_unpack;
  bit [7:0] pack[$];

 initial begin
   len = 32'haa_bb_cc_dd;
$display("[Pack] len[7:0]=%0h",len[7:0]);
$display("[Pack] len[15:8]=%0h",len[15:8]);
$display("[Pack] len[23:16]=%0h",len[23:16]);
$display("[Pack] len[31:24]=%0h",len[31:24]);


   pack = { << 8 { len } };

   foreach(pack[k]) $display("pack[%0d]=%0h",k,pack[k]);

   { << 8 { len_unpack } } = pack;

   $display("[Unpack] len_unpack[7:0]=%0h",len_unpack[7:0]);
$display("[Unpack] len_unpack[15:8]=%0h",len_unpack[15:8]);
$display("[Unpack] len_unpack[23:16]=%0h",len_unpack[23:16]);
$display("[Unpack] len_unpack[31:24]=%0h",len_unpack[31:24]);
end
endmodule
```

```
pack.push_back(len[7:0]);
pack.push_back(len[15:8]);
pack.push_back(len[23:16]);
pack.push_back(len[31:24]);
```

```
[Pack]  len[7:0]    =dd
[Pack]  len[15:8]   =cc
[Pack]  len[23:16] =bb
[Pack]  len[31:24] =aa
```

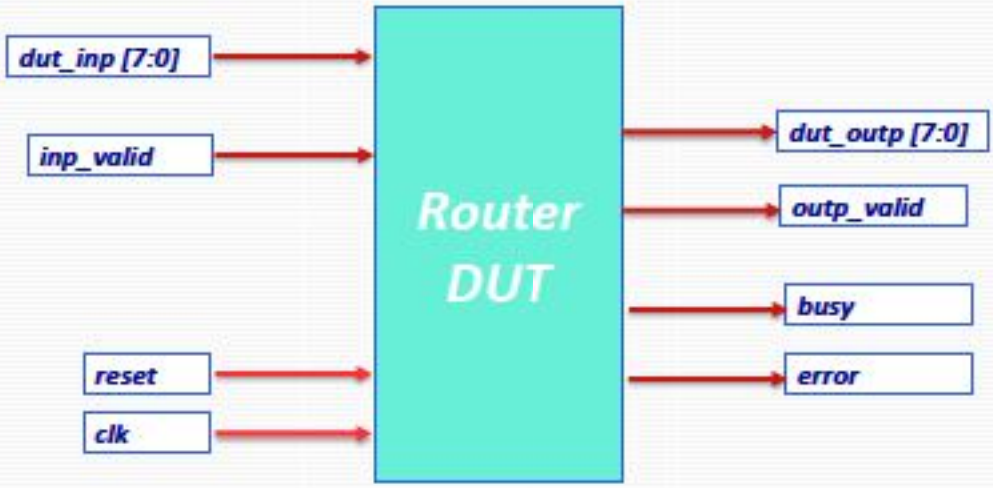
```
pack[0]=dd
pack[1]=cc
pack[2]=bb
pack[3]=aa
```

```
[Unpack] len_unpack[7:0]    =dd
[Unpack] len_unpack[15:8]   =cc
[Unpack] len_unpack[23:16] =bb
[Unpack] len_unpack[31:24] =aa
```

```
len_unpack[7:0]    = pack[0];
len_unpack[15:8]   = pack[1];
len_unpack[23:16]  = pack[2];
len_unpack[31:24]  = pack[3];
```

Packet format:

SA(1 byte)	DA(1 byte)	LEN(4 bytes)	CRC(4 bytes)	PAYLOAD (2-to-1900 bytes)

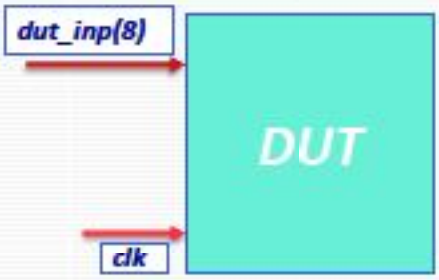


Packet format:

SA(1 byte)	DA(1 byte)	LEN(4 bytes)	CRC(4 bytes)	PAYLOAD (2-to-1900 bytes)

```
//sa ,da,len,crc,payload
typedef struct  {
bit [7:0] sa;          //source addr
bit [7:0] da;          //destination addr
bit [31:0] len;         //length of pkt
bit [31:0] crc;         //crc
bit [7:0] payload[]; //payload
} packet;
module testbench;
packet pkt;

initial begin
generate_stimulus();
drive();
end
endmodule
```




```
function void generate_stimulus();
pkt.sa=4;
pkt.da=8;
pkt.payload=new[10];
foreach(pkt.payload[i])
  pkt.payload[i]=$urandom;

pkt.len=payload.size() + 4+4+1+1;
pkt.crc=payload.sum();
endfunction
```

```
task drive();
@(posedge clk);
dut_inp=pkt.sa;
@(posedge clk);
dut_inp=pkt.da;
@(posedge clk);
dut_inp=pkt.len[7:0];
@(posedge clk);
dut_inp=pkt.len[15:8];
@(posedge clk);
dut_inp=pkt.len[23:16];
@(posedge clk);
dut_inp=pkt.len[31:24];
@(posedge clk);
dut_inp=pkt.crc[7:0];
@(posedge clk);
dut_inp=pkt.crc[15:8];
@(posedge clk);
dut_inp=pkt.crc[23:16];
@(posedge clk);
dut_inp=pkt.crc[31:24];
foreach(pkt.payload[i])
begin
@(posedge clk);
dut_inp=pkt.payload[i];
end
endtask
```

Streaming Operators(pack)

```
//sa ,da,len,crc,payload
typedef struct {
bit [7:0] sa;          //source addr
bit [7:0] da;          //destination addr
bit [31:0] len;        //length of pkt
bit [31:0] crc;        //crc
bit [7:0] payload[];  //payload
} packet;
module testbench;
packet pkt;
bit [7:0] stream[$];

initial begin
generate_stimulus();
pack();
drive();
end
endmodule
```

```
task drive();
foreach(stream[i])  begin
@(posedge clk);
dut_inp=stream[i];
end
endtask
```

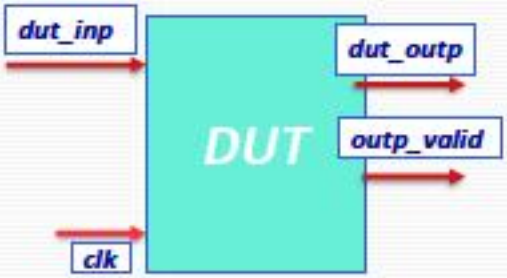
```
function void generate_stimulus();
pkt.sa=4;
pkt.da=8;
pkt.payload=new[10];
foreach(pkt.payload[i])
  pkt.payload[i]=$urandom;

pkt.len=payload.size()+4+4+1+1;
pkt.crc=payload.sum();
endfunction
```

```
function void pack();
stream = { << 8 {pkt.payload, pkt.crc, pkt.len, pkt.da, pkt.sa} };
endfunction
```

bit [7:0] stream[$]	
0	sa
1	da
2	len[7:0]
3	len[15:8]
4	len[23:16]
5	len[31:24
6	crc[7:0];
7	crc[15:8];
8	crc[23:16];
9	crc[31:24];
10	payload[0]
11	payload[1]
12	payload[2]
.....	.....
19	payload[10]

Streaming Operators (unpack)




```
initial begin
forever begin
@(posedge outp_valid);
//Not a complete code
stream_out.push_back(dut_outp);
//Write logic to collect complete packet
///Collect until outp_valid become 0
@(posedge clk);
end
end
```

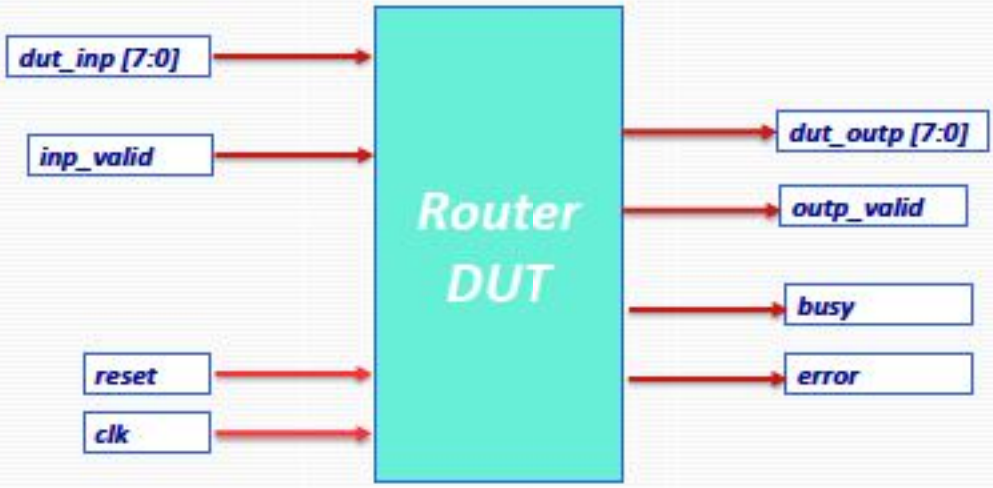
```
function void unpack();
{<< 8 {pkt.payload,pkt.crc,pkt.len,pkt.da,pkt.sa}} = stream_out

endfunction
```

bit [7:0] stream_out[$]	
0	sa
1	da
2	len[7:0]
3	len[15:8]
4	len[23:16]
5	len[31:24
6	crc[7:0];
7	crc[15:8];
8	crc[23:16];
9	crc[31:24];
10	payload[0]
11	payload[1]
12	payload[2]
.....	.....
19	payload[10]

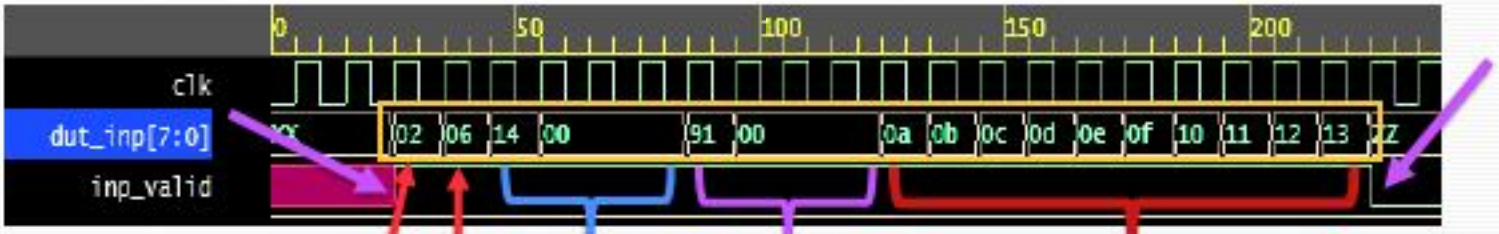
Packet format:

SA(1 byte)	DA(1 byte)	LEN(4 bytes)	CRC(4 bytes)	PAYLOAD (2-to-1900 bytes)



Router Input/Output Waveform

Input Signals

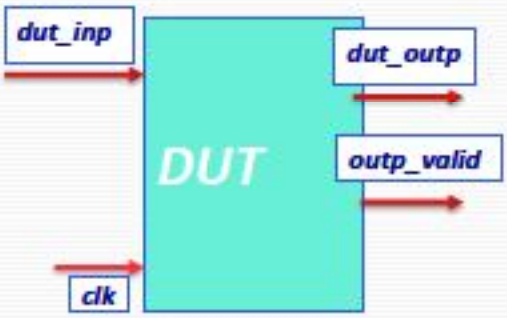



```
function void generate_stimulus();
pkt.sa=2;
pkt.da=6;
pkt.payload=new[10];
foreach(pkt.payload[i])
  pkt.payload[i]=i+10;

pkt.len=payload.size()+4+4+1+1;
pkt.crc=payload.sum();
endfunction
```

Output Signals

OutPut collection




```
initial begin//Section 8
forever begin
@(posedge outp_valid);

Loop Starts:
      //Not a complete code
      outp_stream.push_back(dut_outp);
      Write logic to collect complete packet
      Collect until outp_valid become 0

      @(posedge clk);
Loop_Ends
Unpack the packet
end//end_of_forever
```

bit [7:0] outp_stream[$]	
0	sa(2)
1	da(6)
2	len[7:0](20)
3	len[15:8] (0)
4	len[23:16] (0)
5	len[31:24] (0)
6	crc[7:0] (91)
7	crc[15:8] (0)
8	crc[23:16] (0)
9	crc[31:24] (0)
10	payload[0] (a)
11	payload[1] (b)
12	payload[2](c)
.....	.....
19	payload[10] (13)

Thank You