Tutorial 5

CSE313

Time consuming statements

- "**#**" : suspends code execution for absolute simulation steps (e.g, **#**3) or absolute time(e.g, **#**3ns)
- **@(**condition**)**: edge-sensitive waiting statements that suspends code execution till condition is toggled.
    - Can be Sensitive to rising edge only: **@(posedge** condition**)**
    - Can be Sensitive to failing edge only: **@(negedge** condition **)**
    - Can be Sensitive to both: **@(**condition**)**
- **wait(**condition**)**:edge-sensitive waiting statements that suspends code execution till condition is true.

Random System Functions

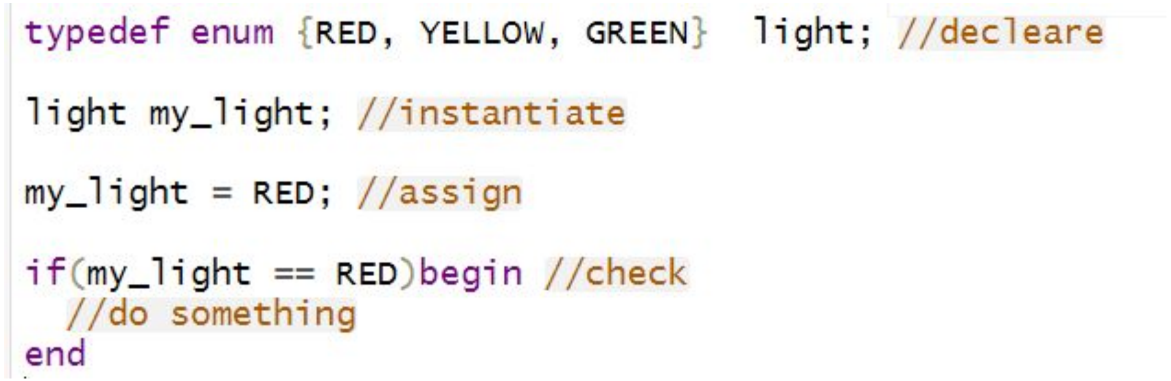
- **$random()**: returns a signed 32 bit random number each time called
  - $random(seed): it can take a certain seed to return same value given same seed

    Same Seed -> Same Random Numbers
- **$urandom()**: exactly as $random() but returns unsigned number.
- **$urandom_range(**int unsigned maxval, int unsigned minval = 0**)**: returns random number within certain range.
  - Default of minval is zero so we can only pass maxval(e.g, $urandom_range(20): return value from 0 to 20)
  - If minval is greater than maxval function will swap them (e.g, $urandom_range(20,30) and $urandom_range(30,20) have same meaning)

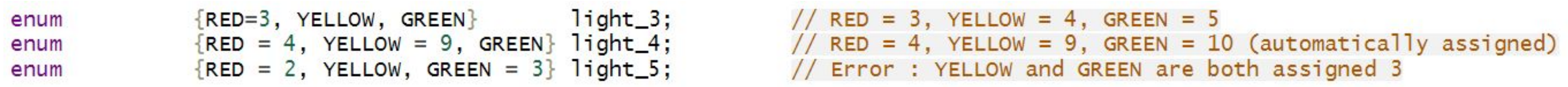
$ -> Indicate that these function uses OS system Calls

Enums

- Named integers (int data type)
- Enumerations names can’t start with number
- Every member is the incrementation by 1 from the previous member if not assigned
- First member is zero if not assigned



```
typedef enum {RED, YELLOW, GREEN}  light; //decleare

light my_light; //instantiate

my_light = RED; //assign

if(my_light == RED)begin //check
  //do something
end
```



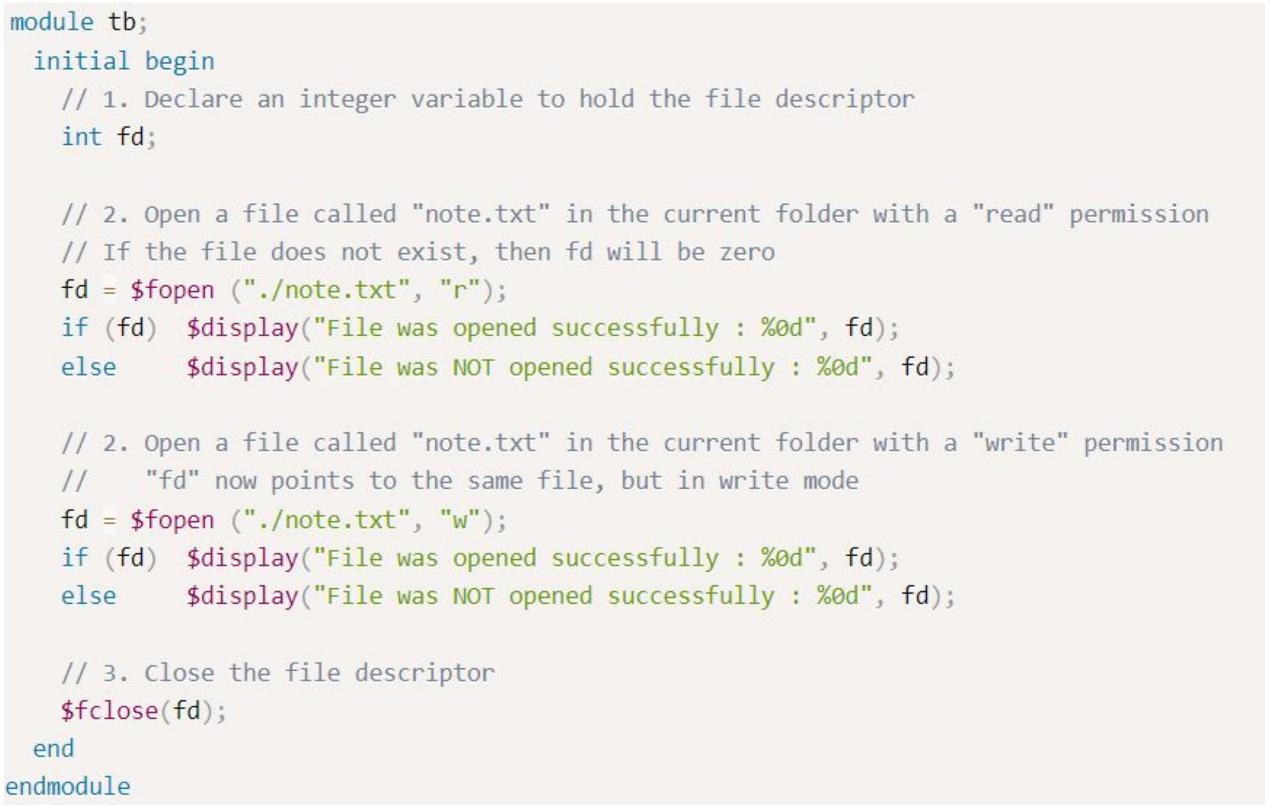
```
enum          {RED=3, YELLOW, GREEN}       light_3;         // RED = 3, YELLOW = 4, GREEN = 5
enum          {RED = 4, YELLOW = 9, GREEN} light_4;         // RED = 4, YELLOW = 9, GREEN = 10 (automatically assigned)
enum          {RED = 2, YELLOW, GREEN = 3} light_5;         // Error : YELLOW and GREEN are both assigned 3
```

File processing in systemverilog

- fd= **$fopen(**string file path, string mode**):**returns handle to the file called file descriptor
    - **"r":** Open for reading
    - **"w":** Create for writing, overwrite if it exists
    - **"a":** Create if file does not exist, else append; open for writing at end of file
- **$fclose(**fd**)**: close file

returns a 32-bit number which is the file descriptor that will be used to process to file

File processing in systemverilog



```
module tb;
  initial begin
    // 1. Declare an integer variable to hold the file descriptor
    int fd;

    // 2. Open a file called "note.txt" in the current folder with a "read" permission
    // If the file does not exist, then fd will be zero
    fd = $fopen ("./note.txt", "r");
    if (fd)  $display("File was opened successfully : %0d", fd);
    else     $display("File was NOT opened successfully : %0d", fd);

    // 2. Open a file called "note.txt" in the current folder with a "write" permission
    //    "fd" now points to the same file, but in write mode
    fd = $fopen ("./note.txt", "w");
    if (fd)  $display("File was opened successfully : %0d", fd);
    else     $display("File was NOT opened successfully : %0d", fd);

    // 3. Close the file descriptor
    $fclose(fd);
  end
endmodule
```

File processing in systemverilog

- $fdisplay(fd,string): writes in a file
- $fscanf(fd, certain format, variable list): reads line by line each time called using a certain format to extract required variable list and returns number of variable extracted
- $feof(): returns one if end of file is reached

Q3

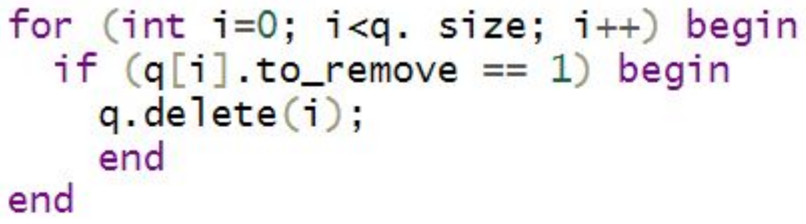
First Solution:
- Decrement the i right after the q.delete(i)

Edit:

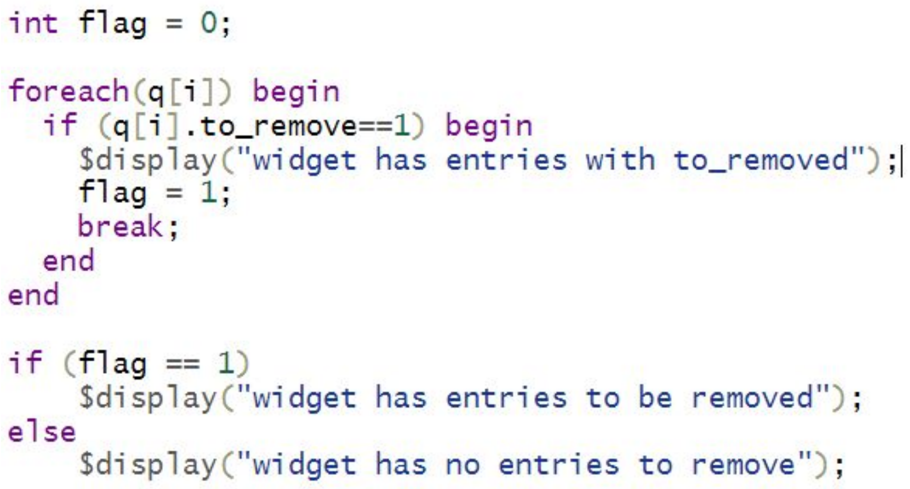
```
if (q[i].to_remove == 1)
begin
    q.delete(i);
    i--;
end
```

Second Solution:
- Change the for loop itself

Edit:
- for(int i = (q.size - 1); i >= 0; i--)



```
for (int i=0; i<q. size; i++) begin
  if (q[i].to_remove == 1) begin
    q.delete(i);
    end
end
```



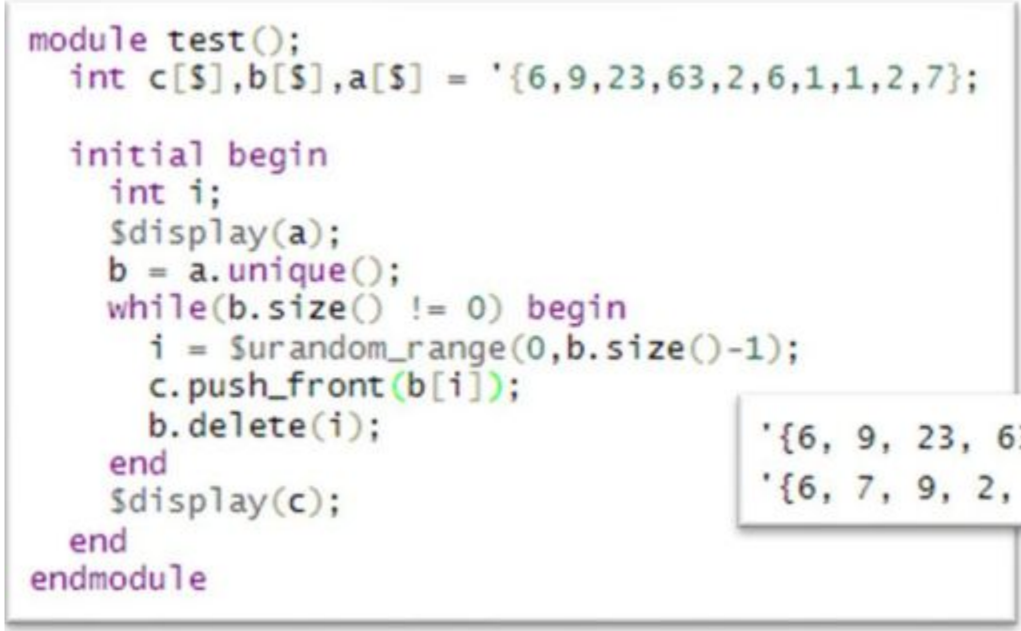
```
int flag = 0;

foreach(q[i]) begin
  if (q[i].to_remove==1) begin
    $display("widget has entries with to_removed");
    flag = 1;
    break;
  end
end

if (flag == 1)
    $display("widget has entries to be removed");
else
    $display("widget has no entries to remove");
```

Take an Example on Q[i].to_remove = {0,0,1,1,1}

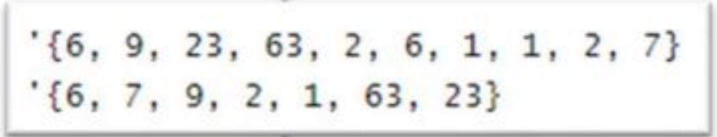
The buggy code will produce a Q like that {0,0,1}

Lab code: https://www.edaplayground.com/x/PQwK

Q13



```
module test();
  int c[$],b[$],a[$] = '{6,9,23,63,2,6,1,1,2,7};

  initial begin
    int i;
    $display(a);
    b = a.unique();
    while(b.size() != 0) begin
      i = $urandom_range(0,b.size()-1);
      c.push_front(b[i]);
      b.delete(i);
    end
    $display(c);
  end
endmodule
```



```
'{6, 9, 23, 63, 2, 6, 1, 1, 2, 7}
'{6, 7, 9, 2, 1, 63, 23}
```

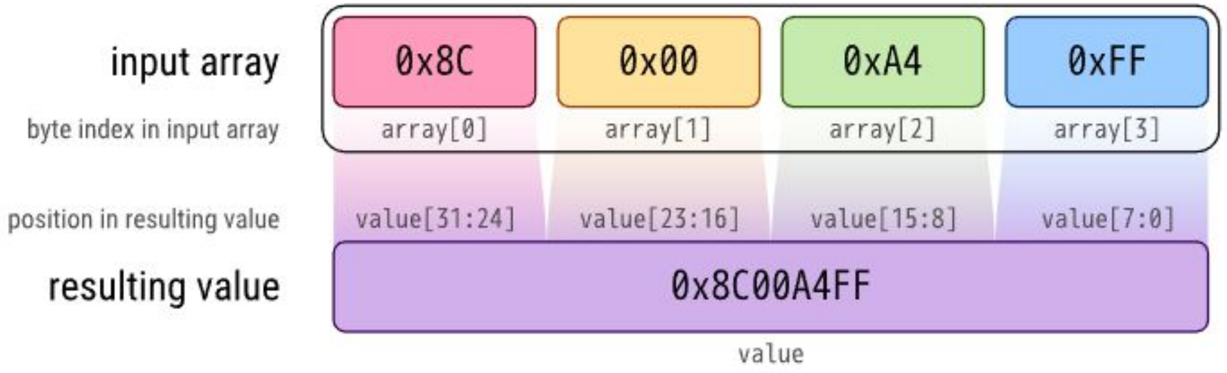
Streaming operator

```
module example_1_2;
  initial begin
    static bit [7:0] array[4] = '{ 8'h8C, 8'h00, 8'hA4, 8'hFF };
    static int       value    = {>>{array}};

    $display("value = 0x%h", value);
  end
endmodule
```

The Normal Concatenation:

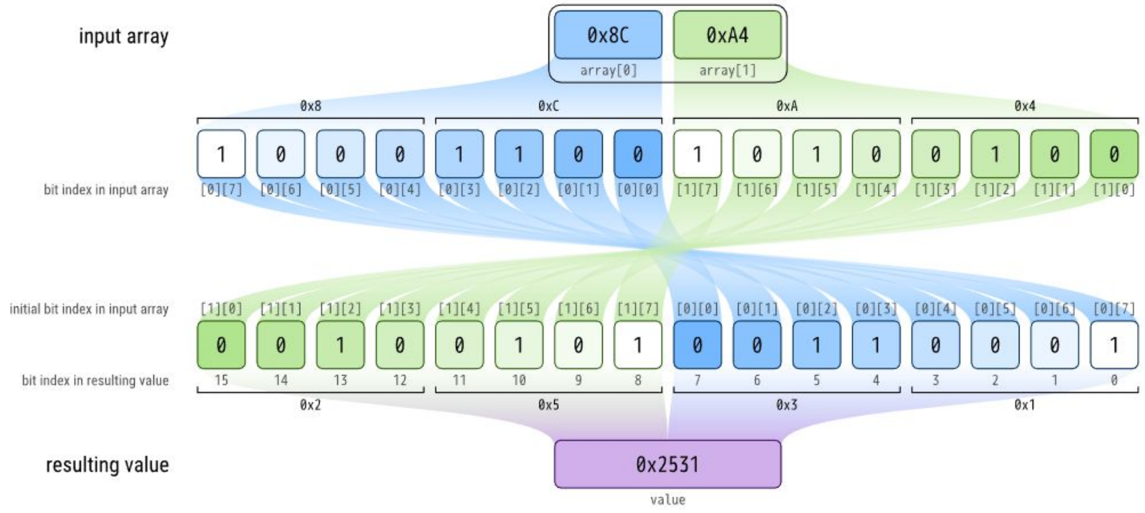
value = {array[0], array[1], array[2], array[3]};

Problem -> Not suitable for large data



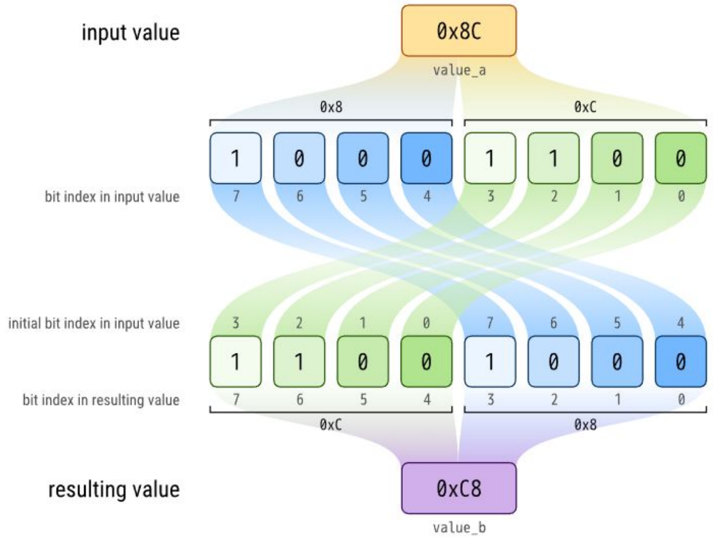
Reverse streaming operator

```
module example_5;
  initial begin
    static bit [7:0] array[2] = '{ 8'h8C, 8'hA4 };
    static shortint  value    = {<<{array}};

    $display("value = 0x%h", value);
  end
endmodule
```

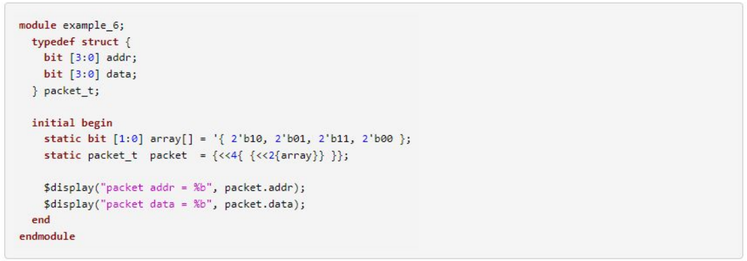


Block Reverse streaming operator

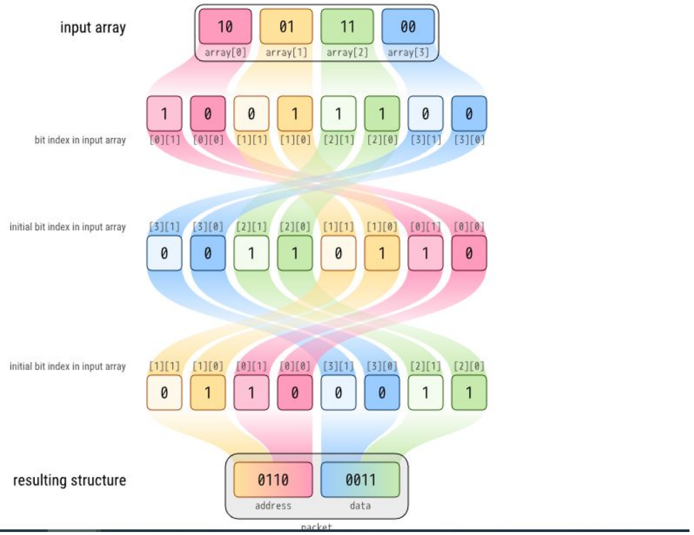
```
module example_4;
  initial begin
    static bit [7:0] value_a = 8'h8C;
    static bit [7:0] value_b = {<<4{value_a}};

    $display("value_b = 0x%h", value_b);
  end
endmodule
```



Advanced packing



```
module example_6;
  typedef struct {
    bit [3:0] addr;
    bit [3:0] data;
  } packet_t;

  initial begin
    static bit [1:0] array[] = '{ 2'b10, 2'b01, 2'b11, 2'b00 };
    static packet_t  packet  = {<<4{ {<<2{array}} }};

    $display("packet addr = %b", packet.addr);
    $display("packet data = %b", packet.data);
  end
endmodule
```



Quiz