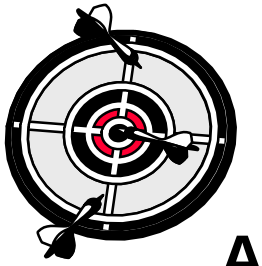
Agenda

DAY 3

8	Object Oriented Programming (OOP) – Inheritance
9	Inter-Thread Communications 
10	Functional Coverage 
11	SystemVerilog UVM Preview
CS	Customer Support

Unit Objectives



After completing this unit, you should be able to:

- **Establish order of execution using events**
- **Avoid resource collision with Semaphores**
- **Pass data between threads via Mailbox**

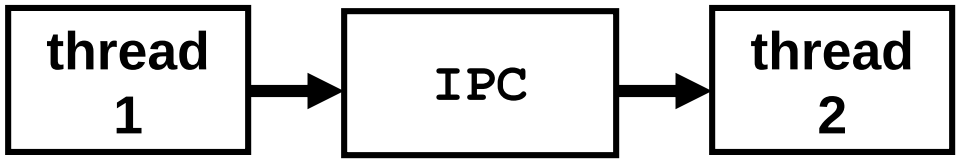
Inter-Thread Communications (ITC)

- Concurrent threads require communication to establish control for sequence of execution
- Three types are covered in this unit

Event based

Resource sharing

Data passing



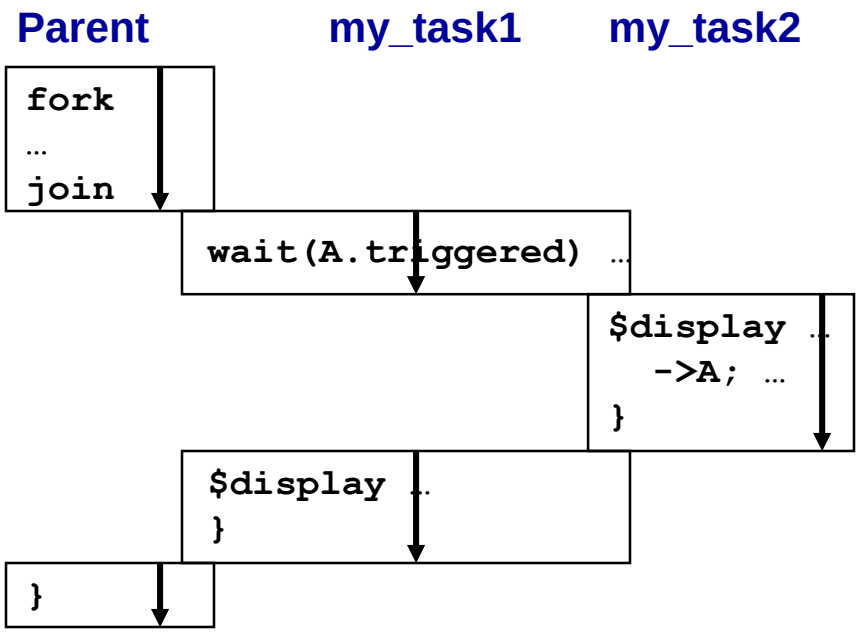
Event Based ITC

- **Synchronize operation of concurrent threads via event variables:**
  - Thread waits for event to be triggered
  - An executing thread triggers the event to enable the waiting thread to be placed into the READY queue
- **Mainly used to control sequence of execution**

Event Based ITC Example

Controlling order of execution.

```
initial begin
  event A;
  fork
     my_task1();
     my_task2();
  join
end
task my_task1();
  wait(A.triggered);
  $display("Print 2nd");
endtask: my_task1
task my_task2();
  $display("Print 1st");
  ->A;
endtask: my_task2
```



Event Wait Syntax

- **Wait until event has been triggered (one shot):**

```
@(event_var [, event2]);
```

  - Will be satisfied by events which occur after the execution of this statement

- **Wait until event has been triggered (persistent):**

```
wait(event_var.triggered);
```

  - Similar to @(event_var), but will also be satisfied by events which happened earlier during the same simulation time
  - Can be used to eliminate potential race condition

Trigger Syntax

`->eventN;`

- `event` variables are handles
- Assigning an event to another makes both event the same event

Example:

```
event a, b, c;
a = b;
-> c;
-> a; // also triggers b
-> b; // also triggers a
a = c;
b = a;
-> a; // also triggers b and c
-> b; // also triggers a and c
-> c; // also triggers a and b
```

Controlling Termination of Simulation

```
program automatic test(router_io.TB rtr_io);
  event DONE;
  initial begin
    fork
      gen();
      check();
    join_none
    for (int i=0; i<16; i++) begin
      int j = i;
      fork
        send(j);
        recv(j);
      join_none
    end
    wait(DONE.triggered);
  end
  ...
endprogram:
```

```
task check();
  forever begin
    ...
    if ($get_coverage() == 100)
      ->DONE;
  end
endtask: chec
```

Blocking statement to prevent termination of simulation until done

Trigger event when termination condition is detected

Resource Sharing ITC

- **Synchronize operation of concurrent threads via access to shared resources:**
  - Thread requests for a shared resource before executing a critical section of code
  - Thread waits if the requested resource is unavailable
  - Thread resumes execution when the requested resource becomes available

Semaphores

- **A Semaphore is a bucket in which keys can be deposited and removed:**
  - A thread tries to acquire keys from a semaphore bucket
  - Thread execution waits until keys requested are available in the semaphore bucket
  - When the requested keys becomes available, thread execution resumes and the requested keys removed from the semaphore bucket
- **Mainly used to prevent multiple threads from accessing the same hardware signal or using same software resource**

Semaphores

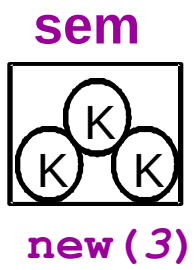
- **Semaphores are supported via built-in semaphore class:**

```
class semaphore;
  function new(int keyCount = 0);
  task put(int keyCount = 1);

  // the following is blocking:
  task get(int keyCount = 1);

  // the following is non-blocking: 1 for success 0 for failure
  function int try_get(int keyCount = 1);
endclass
```

Using Semaphores

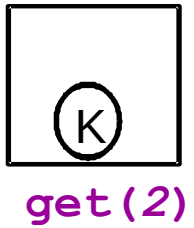
- **Semaphore buckets created using the constructor `new()`**

```
semaphore sem
sem = new(3);          // bucket with 3 keys is created
```



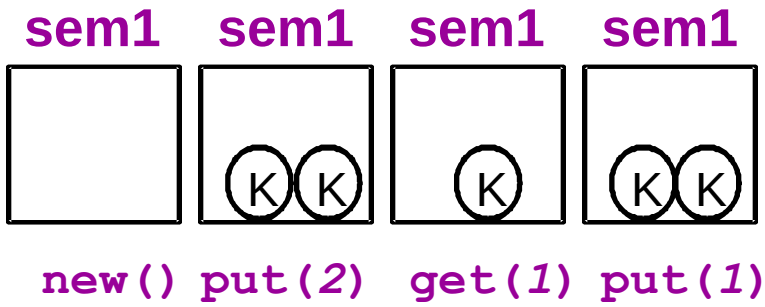
- **Acquiring Keys**

```
sem.get(2);                // bucket with 1 key left
if (!sem.try_get(2))       // does not block
  $display("Failed");
sem.get(2);                //blocks until 2 keys available
```



- **Returning/Creating Keys**

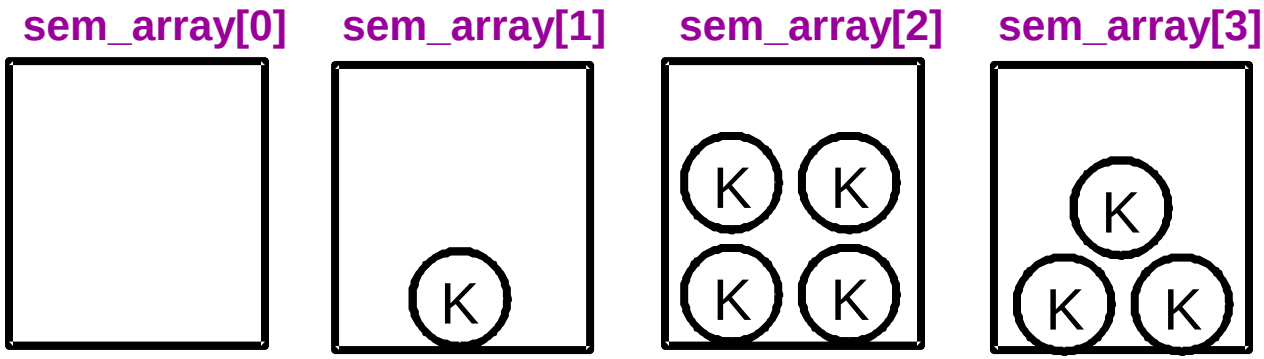
```
semaphore sem1
sem1 = new(); // 0 keys
sem1.put(2);  // Create 2 keys
sem1.get(1);  // Use 1 key
sem1.put(1);  // Return 1 key
```



Using Semaphore Arrays

- Semaphores in an array must be constructed before use

```
semaphore sem_array[];
sem_array = new[4];    // array of 4 null semaphore handles created
sem_array[2].put(2);   // ERROR: semaphore bucket does not exist!
foreach (sem_array[i])
   sem_array[i] = new(i);  // construct semaphore with i keys
sem_array[2].put(2);   // okay.  sem_array[2] now has 4 keys
```



Arbitration Example

```
program automatic test(router_io.TB rtr_io);
  semaphore sem[];
  ...;
  sem = new[16];
  foreach(sem[i])
    sem[i] = new(1);
  ...;
  task send();
    sem[da].get(1);
    send_addrs();
    send_pad();
    send_payload();
    sem[da].put(1);
  endtask: send
  ...;
endprogram: test
```

Create semaphore array to represent each output port

Construct each individual semaphore bucket

Block if others are driving the chosen output port

Re-deposit keys when done with driving port

Mailbox

- **Messages are passed between threads via mailbox:**
  - A thread sends data by putting messages a mailbox
  - A thread retrieves data by getting messages from the mailbox
  - If a message is not available, the thread can wait until there is a message to retrieve
  - Thread resumes execution once the message becomes available
- **Mainly used for passing data between concurrent threads**

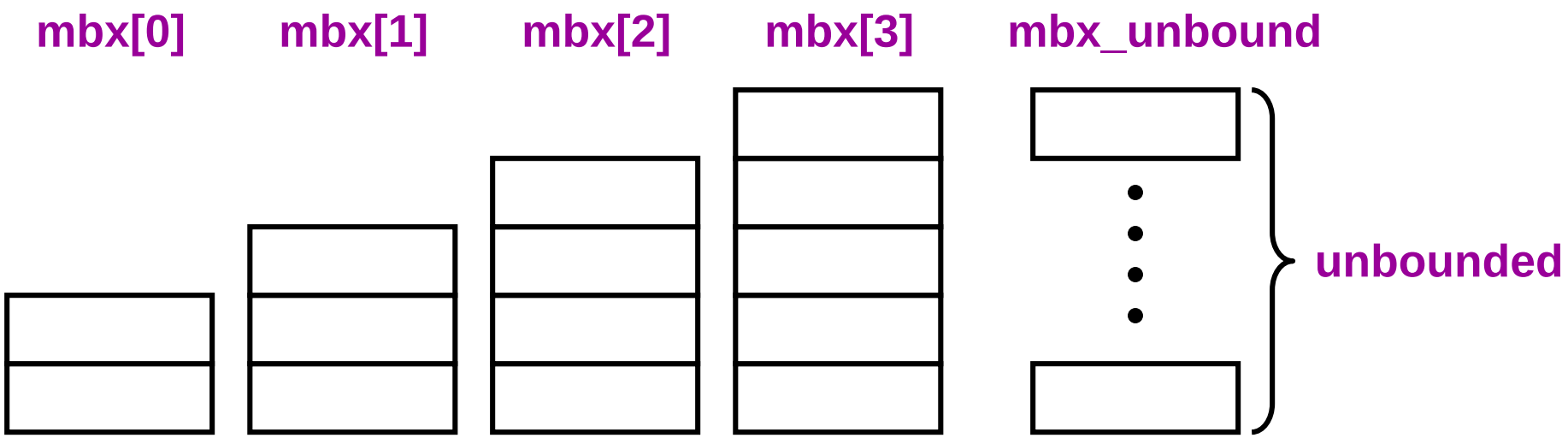
Mailbox Class

- **Mailboxes are supported via built-in mailbox class:**

```
class mailbox #(type T = dynamic_type);
  function new(int bound = 0);
  function int num();     // return # of messages
  task put(T message);    // wait if mailbox full
  task get(ref T message); // wait if no message
  task peek(ref T message); // wait if no message
  function int try_put(T message);
  function int try_get(ref T message);
  function int try_peek(ref T message);
endclass
```

Creating Mailboxes

```
mailbox #(Packet) mbx[];          //Packets only mailbox array
mailbox #(instr_e) mbx_unbound;          //enum type instr_e
mbx_unbound = new();              // mailbox size is unbounded
mbx = new[4];          // array of 4 null mailbox handles created
for (int i=0; i<mbx.size(); i++) begin
  mbx[i] = new(i+2);          // bound to max of i+2 messages
end
```



Putting Messages into Mailboxes

```
task put(message);  // block if exceeds bound
function int try_put(message);  // non-blocking
```

Example:

mbox


```
int status;
typedef enum {ADD=1, SUB, MUL, DIV} instr_e;
instr_e instr = ADD;
mailbox #(instr_e) mbox = new(1);
Packet pkt = new();
mbox.put(instr);
status = mbox.try_put(instr);//status = 0 - mailbox full
mbox.put(pkt);                    //incorrect type - compiler error
```

Retrieving Messages from Mailboxes (1/2)

```
task get(ref message);  // block if mailbox is empty
function int try_get(ref message);  // non-blocking
// if successful, return number of messages in mailbox before try_get,
// 0 if no message
```

Example:

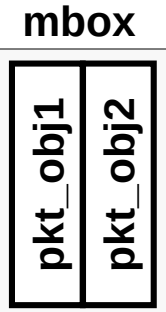

```
int status;
typedef enum {ADD=1, SUB, MUL, DIV} instr_e;
instr_e instr1 = MUL, instr2; Packet pkt;
mailbox #(instr_e) mbox = new(1);
mbox.put(instr1); mbox.get(instr2);
$display("instr2 = %s", instr2.name());// instr2 = MUL
status = mbox.try_get(instr2);  // status = 0
status = mbox.try_put(instr1);  // status = 1
mbox.get(instr2);                    // instr2 = MUL
mbox.get(instr2);                 // blocking – mailbox empty
mbox.get(pkt);                    // incorrect type - compiler error
```

Retrieving Messages from Mailboxes (2/2)

```
task peek(ref message);  // block if mailbox is empty
function int try_peek(ref message);  // non-blocking
// if successful, return number of message in mailbox before try_peek,
// 0 if no message
```

Example:

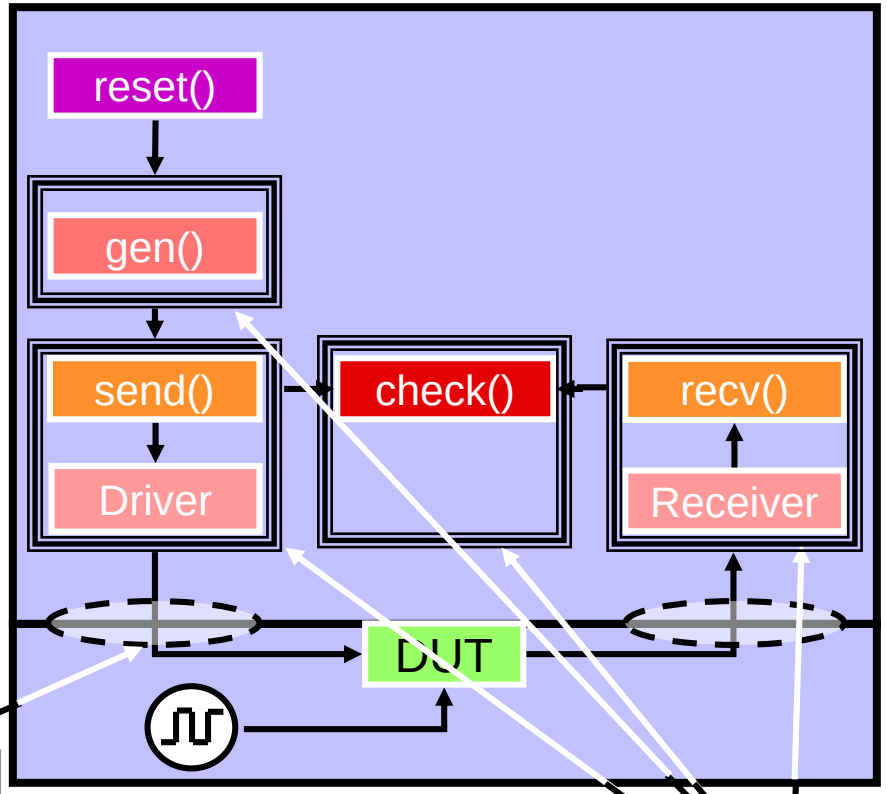
```
Packet pkt_obj1 = new();
Packet pkt_obj2 = new(), pkt2drv;
mailbox #(Packet) mbox = new();
mbox.put(pkt_obj1);
status = mbox.try_put(pkt_obj2);          // status = 1
$display("%0d messages", mbox.num());     // 2 messages
mbox.peek(pkt2drv);                       // still 2 messages left
$display(mbox.try_peek(pkt2drv));         // displays 2
$display(mbox.try_get(pkt2drv));          // displays 2
```



Lab 5 Introduction

120 min

Expand to broad-spectrum verification



Expand to exercise all ports concurrently

Compile & Simulate

semaphore-based resource mgmt

Encapsulate in OOP Class

Unit Objectives Review

Having completed this unit, you should be able to:

- **Establish order of execution using event flag**
- **Avoid resource collision with Semaphores**
- **Pass data via Mailbox**