DAY-66
#100DAYSRTL

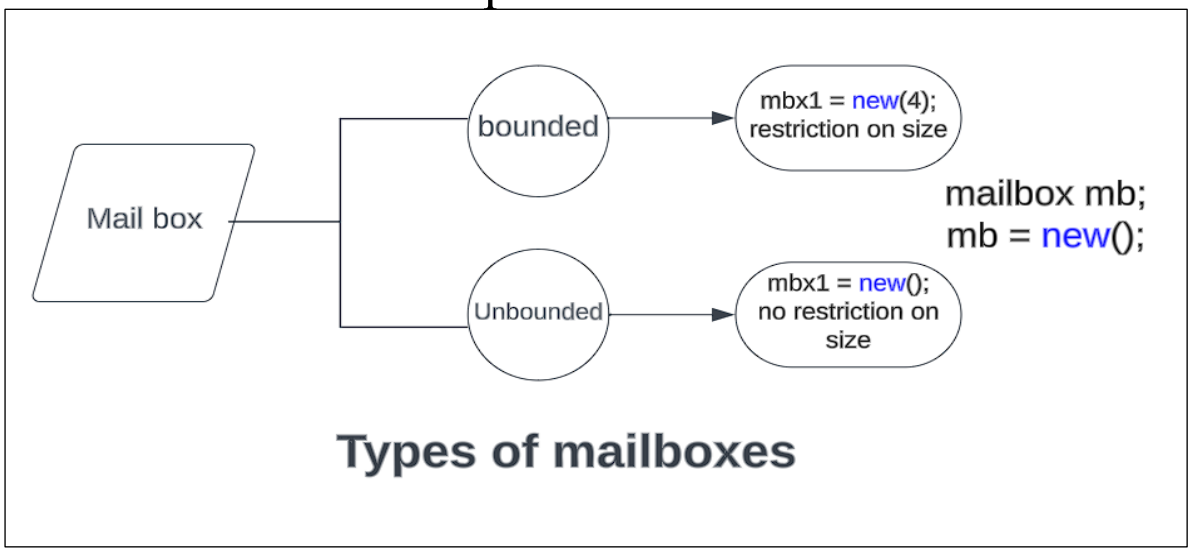
“System Verilog: IPC:Mailbox ”

“Introduction”:-

In System Verilog IPC(Inter process communication), when creating multiple processes that run simultaneously, they might need to communicate or share resources. To facilitate this, System Verilog introduced tools like mailboxes and semaphores. Mailboxes, specifically, are useful in test benches where various components run in separate processes and need a way to communicate with each other.

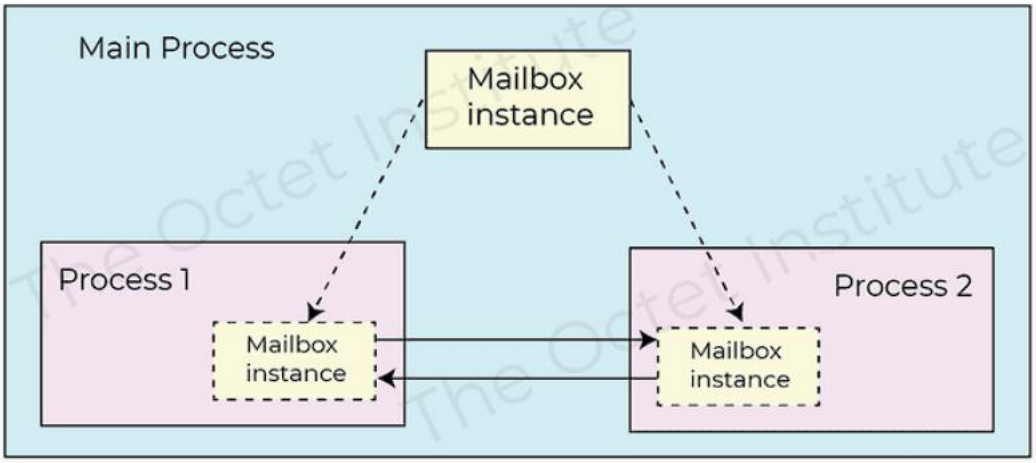
“Mailbox”:-

Mailboxes are like a direct line of communication between two parallel processes. Picture them as a real mailbox: just like how you send and receive letters, in a mailbox, there's a sender and a receiver. The receiver can only get mail when the sender sends it. If the mail hasn’t arrived yet, the receiver has two options. They can wait for the mail or they can go and do other tasks and then later check for the .Similarly, if the mailbox is full the sender can't add new mail until there's space available.



“Establishing IPC via Mailbox”:-

There will be a parent process which will start the sub-process. Mailbox is defined and declared inside the parent process. This mailbox handle can then be used by the sub-processes to communicate with each other**.**



“Methods of mailbox”:-

1. **new() -** This function creates a new instance of the mailbox and returns the handle. If no parameter is passed unbounded mailbox is created else bounded mailbox is created which can store max number of elements as specified.
2. **put() –** This is a blocking task and is used to write some data to the mailbox. If the mailbox is full it will wait for the mailbox to have some space.
3. **try_put() -** This is a non-blocking function. This will try to write a message in the mailbox. If the mailbox is full or due to some reason it cannot write it will return a status value of 0 or else return a positive value.
4. **get() –** This is a blocking task and is used to get data from the mailbox. This method also deletes the data from the mailbox after retrieving it.
5. **try_get() –** Same as get() method but this is non-blocking function. The function returns positive number if data is retrieved from the mailbox else returns zero.
6. **peek() –** This is similar to the get() method in nature but this task does not delete the data from the mailbox. As the name suggests it just peeks into the mailbox and gets back data without changing any contents of the mailbox
7. **try_peek() –** Similar to peek() method but is non-blocking function. Returns a positive number if any data can be retrieved from the mailbox else returns 0.

8. **num()** – This function returns the number of data elements that are currently present in mailbox.

Code Practising:-

```
module mailbox_demo();
    mailbox mbx1;
    initial begin
        mbx1 = new(4);
        fork
            begin:prcs_1
                bit [7:0] a;
              repeat(5) begin
                  a = $urandom_range(100);
                    mbx1.put(a);
                    $display("[%0t] Sent data = %0d", $time, a);
                    #5;
                end
            end
            begin:prcs_2
                bit [7:0] a;
                repeat(5) begin
                    mbx1.get(a);
                    $display("[%0t] Received data = %0d", $time, a);
                    #10;
                end
                if(mbx1.try_get(a))
                    $display("[%0t] Received data = %0d", $time, a);
                else
                    $display("No element to retrieve");
            end
        join
    end
endmodule
```

Result:-

```
[0] Sent data = 33
[0] Received data = 33
[5] Sent data = 75
[10] Received data = 75
[10] Sent data = 13
[15] Sent data = 87
[20] Received data = 13
[20] Sent data = 69
[30] Received data = 87
[40] Received data = 69
No element to retrieve
Simulation has finished.
```

“Parameterized Mailbox”:-

Parameterized mailboxes offer a solution by being more specific. They work like a labeled container, where you define the type of data that can be sent. Just as a package marked with clear instructions on its content, parameterized mailboxes allow senders to only send data of a specified type. This restriction prevents mismatched data and potential errors, ensuring that the receiver is always prepared to handle the expected content.

```
mailbox#([data_type]) mbx;
mbx = new();
```

“Mailbox Vs Queue”:-

- ✓ The queue is an array, memory can be operated like a FIFO or LIFO using the non-blocking methods, whereas the Mailbox is a class that can be used to communicate between TB components for sharing the transactions or data.
- ✓ Mailbox has both blocking and non-blocking methods. Mailbox is implemented with queues and semaphores.