**Explanation of Test Cases with Clock Cycles**

1. **Test Case 1** (Clock Cycle 2-4):
   * **Description**: Processor 0 requests the bus.
   * **Expected Result**: Processor 0 is granted (Com\_Bus\_Gnt\_proc\_0 goes high).
   * **Explanation**: Since there are no competing requests, the arbiter directly grants Processor 0 on the next clock cycle.
2. **Test Case 2** (Clock Cycle 5-7):
   * **Description**: Processor 1 and Snoop 0 request simultaneously.
   * **Expected Result**: Snoop 0 is granted (Com\_Bus\_Gnt\_snoop\_0 goes high) because snoop has higher priority than processor requests.
   * **Explanation**: The arbiter prioritizes the snoop request over the processor, ensuring coherence by responding to snoops first after this processor 1 is granted request.
3. **Test Case 3** (Clock Cycle 8-10):
   * **Description**: Processor 2, Snoop 1, and a memory request are active.
   * **Expected Result**: Memory snoop is granted first (Mem\_snoop\_gnt goes high) as it has the highest priority.
   * **Explanation**: The arbiter grants memory access, maintaining cache coherence by servicing memory snoop requests before others, this is then followed by snoop 1 request and then processor request.
4. **Test Case 4** (Clock Cycle 11-13):
   * **Description**: Multiple processor requests with no snoop or memory request.
   * **Expected Result**: One of the processor grants goes high, arbitrating fairly among them (Processor 1 or Processor 3).
   * **Explanation**: With no competing snoop or memory requests, processor requests are handled, ensuring no starvation.
5. **Test Case 5** (Clock Cycle 14-18):
   * **Description**: All types of requests are active (Processors, Snoops, and Memory).
   * **Expected Result**: Memory is granted first (Mem\_snoop\_gnt), followed by snoop grants (Com\_Bus\_Gnt\_snoop\_x), and finally processor grants.
   * **Explanation**: This confirms that the arbiter properly prioritizes memory requests, followed by snoop requests, and finally processor requests. This order enforces coherence by handling higher-priority snoop and memory requests first.

Each test case confirms that the arbiter maintains priority as required for cache coherence in a multicore system. The testbench ensures that even when multiple requests are active, the arbiter follows the priority order essential for maintaining MESI protocol coherence.