**Chapter 8, Advanced OOP and Testbench Guidelines homework**

The purpose of this homework is to create a testbench to generate IPV4 packets. IPV4 packet headers have the format in Figure 1. A payload of 64-bits is included in the packet as well.



Figure 1: IPV4 packet format

The fields in the header are described in Table 1.

|  |  |  |
| --- | --- | --- |
| **Field** | **Width** | **Meaning** |
| Version | 4 | The Version field indicates the format of the internet header. Must equal 4. |
| IHL | 4 | Internet Header Length is the length of the internet header in 32 bit words. Must be 5. |
| Type of Service | 8 | The Type of Service provides an indication of the abstract parameters of the quality of service desired.  bits [2:0]: Precedence  111 - Network Control  110 - Internetwork Control  101 - CRITIC/ECP  100 - Flash Override  011 - Flash  010 - Immediate  001 - Priority  000 - Routine  bit 3: 0 = Normal Delay, 1 = Low Delay.  bit 4: 0 = Normal Throughput, 1 = High Throughput.  bit 5: 0 = Normal Relibility, 1 = High Relibility.  bits [7:6]: Reserved for Future Use. |
| Total Length | 16 | Total Length is the length of the datagram, measured in octets, including internet header and data. |
| Identification | 16 | An identifying value assigned by the sender to aid in assembling the fragments of a datagram. |
| Flags | 3 | bit 0: reserved, must be zero  bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.  bit 2: (MF) 0 = Last Fragment, 1 = More Fragments. |
| Fragment Offset | 13 | This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero. |
| Time to Live | 8 | This field indicates the maximum time the datagram is allowed to remain in the internet system. |
| Protocol | 8 | This field indicates the next level protocol used in the data portion of the internet datagram. |
| Header Checksum | 16 | A CRC checksum on the header only. It is calculated by:  {Version, IHL, Type of Service} ^ Total Length ^ Identification ^ {Flags, Fragment Offset} ^ {Time to Live, Protocol} ^ Source IP Address[31:16] ^ Source IP Address[15:0] ^ ^ Destination IP Address[31:16] ^ Destination IP Address[15:0] |
| Source IP Address | 32 | The source address. |
| Destination IP Address | 32 | The destination address. |

Table 1: IPV4 Header field description

The testbench will:

1. Create a header class defining the header fields of an IPV4 packet.
2. Create a data class defining the payload of an IPV4 packet.
3. Extend the abstract class, *base\_packet*, in to create a packet class.
4. Extend the packet class to create a bad packet class. The bad packet class will corrupt the Header Checksum Field for 2% of the packets.
5. Using the packet class as a blueprint create a Generator class that will randomize the blueprint pattern. Provide the ability to replace a good packet with a bad packet in the Generator. The generator will generate 1000 packets and place them in a mailbox called *gen2drv*.
6. Create callbacks to:
   1. Set the version to 3 for 1% of the packets
   2. Send the packet to a scoreboard for comparison by calling scoreboard function *compare\_expected*. The scoreboard is provided for you.
7. The driver will get a packet from mailbox *gen2drv*, check the callback queue, and transmit the packet. The transmit task will consume 10ns.
8. Using a test registry define 3 test classes:
   1. *TestGood* which simply generates packets with no corruption
   2. *Test\_v3* which uses the callback referenced in 6.a to generate packets with the version set to 3.
   3. *TestBad* which uses the bad packet class referenced in 4 to generate packets with a corrupted Header Checksum.
9. Using a test registry provide the ability to run each of the 3 tests without recompiling.
10. At the end of the test print out the number of packets compared.

Deliverables:

1. All of your code in tarred/zipped electronic format including scripts, project if used, etc.
2. If not using scripts the command line used to run the 3 tests.
3. Clear directions on running your solution.
4. Copy of the transcript window for the 3 tests. It is expected that:
   1. For *TestGood* the transcript will simply contain the number of packets compared with no errors
   2. For *Test\_v3* the transcript will contain the number of packets compared with about 10 packets containing version errors
   3. For *TestBad* the transcript will contain the number of packets compared with about 20 packets containing header checksum errors.

|  |
| --- |
| virtual class base\_packet;  rand header\_class header;  rand data\_class data;    static int count; // Number of instance created  int id; // Unique transaction id  function new();  id = count++; // Give each object a unique ID  header = new();  data = new();  endfunction // new    pure virtual function base\_packet copy();  pure virtual function void display();  pure virtual function void calc\_header\_checksum();  endclass |

Code Example 1: Abstract base\_packet