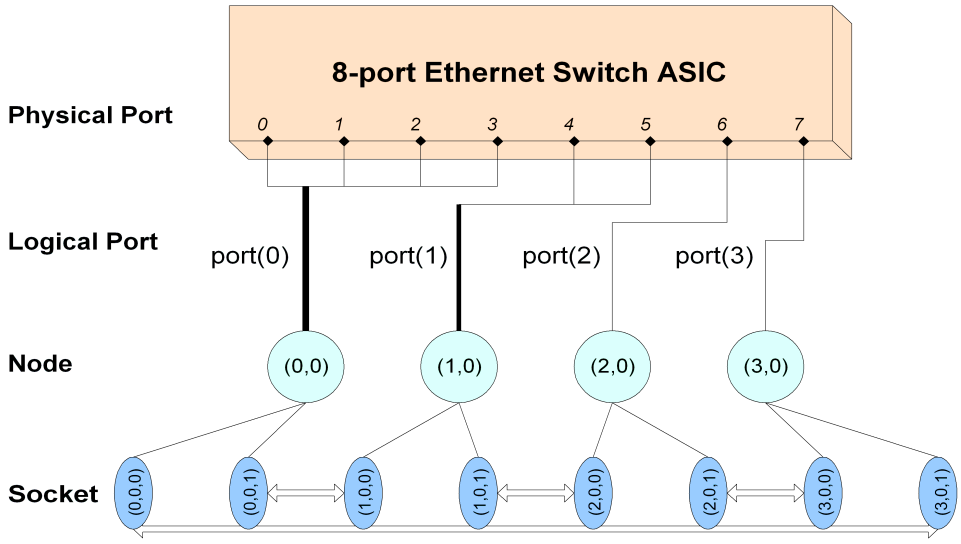
Test Case in SDL

The example is a test case for an eight-port Ethernet switch ASIC illustrated in the following figure. Eight sessions are generated in parallel in this test case, namely, port 0 and 1 pair, port 1 and 2 pair, port 2 and 3 pair, and port 3 and 0 pair. Each pair has two sessions in full duplex so that traffic pattern is a two-way ring.



There are eight TCP/IP flows that are complete TCP sessions. The object *soc* applies to the architecture simulator that generates test input vector. Properties defined under the object specify the number of total logical ports, seed for random function, data length, data pattern and total number of packets per flow. There are four logical ports named *port(0)*, *port(1)*, *port(2)*, and *port(3)*. The logical port(0) consists of four physical ports that trunk into one logical port for bandwidth sharing. The node associated with the port implies external computer connected to the port of switch. So, each node has its own IP address and a set of sockets is defined under the node for TCP transactions. Given socket sets under the node, the property of *bindSocket* makes a connection between each socket. Unless explicitly specified, socket pairs are randomly chosen by default.

Under the switch ASIC object of *ETHERNET_SWITCH*, design objects such as registers and memories are defined such as *portMode* register field. With the statement of property and value pair, configuration thread set the value of 0 to the register *portMode* during configuration stage. The DSC block example represents data flow waiting for configuration done, and then read the value of the *portMode* register of which value is printed out to console. Using built-in commands, memory entry values can be written and read. Any address space can be accessed through write and read primitives.

1.1 8-Port Switch Test Case with Embedded DSC object

```
// Topology and test vector generation
%soc
// total port used
totalPorts = 4
// random seed
randomSeed = 4587
// payload length is from 0 to 100 randomly
dataLength = [0,100]
// payload content increment by 1
dataLengthInc = 1
// data pattern
dataPattern = random
// total packets per flow
totalPackets = 600

// port 0 definitions
%port(0)
// a logical port with 4 physical ports
// (trunking)
pid = [0,1,2,3]
// the name of this port
name = mac0
// the data link type for Ethernet
linkType = mac
// total node connected to this port
totalNodes = 1
// total sockets opened per node
totalSockets = 3
// node(0,0) is the first node connected to port(0)
%node(0,0)
// property overloading
// overwrite the definition at the connected port
totalSockets = 2
// ip address
ipAddr = 192.3.4.6
%socket(0,0,0)
dataLength = [64,100]
%socket(0,0,1)
totalPackets = [2,10]

// port 1 definitions
%port(1)
pid = [4,5]
name = mac1
linkType = mac
%node(1,0)
totalSockets = 2
%socket(1,0,0)
bindSocket = socket(0,0,1)
%socket(1,0,1)

// port 2 definitions
%port(2)
pid = 6
name = mac2
linkType = mac
%node(2,0)
totalSockets = 2
%socket(2,0,0)
bindSocket = socket(1,0,1)
%socket(2,0,1)

// port 3 definitions
%port(3)
pid = 7
name = mac3
linkType = mac
%node(3,0)
totalSockets = 2
%socket(3,0,0)
bindSocket = socket(2,0,1)
%socket(3,0,1)
bindSocket = socket(0,0,0)

// the DUT object
%ETHERNET_SWITCH
// 0: xgmii, 1: gmii
portMode = 0

// DSC: dynamic simulation control
<dsc_begin>
  // wait for configuration done
  wait (TB_SC_DONE == 1);

  // read the portMode register value
  $val = read(portMode);

  // print out the value
  printf("portMode = 0x%x", $val);

  // set value to MEM_A memory
  for ($i=0; $I<16; $i++) {
    // set value to each memory entry
    write(MEM_A[$i], 0x12345678abcd+$i);
  }
  // put the value to address 0xfe008000
  $addr = 0xfe008000;
  write($addr, 0xbabeface);
  // read from the address 0xfe008000
  $val = read($addr);
<dsc_end>
```