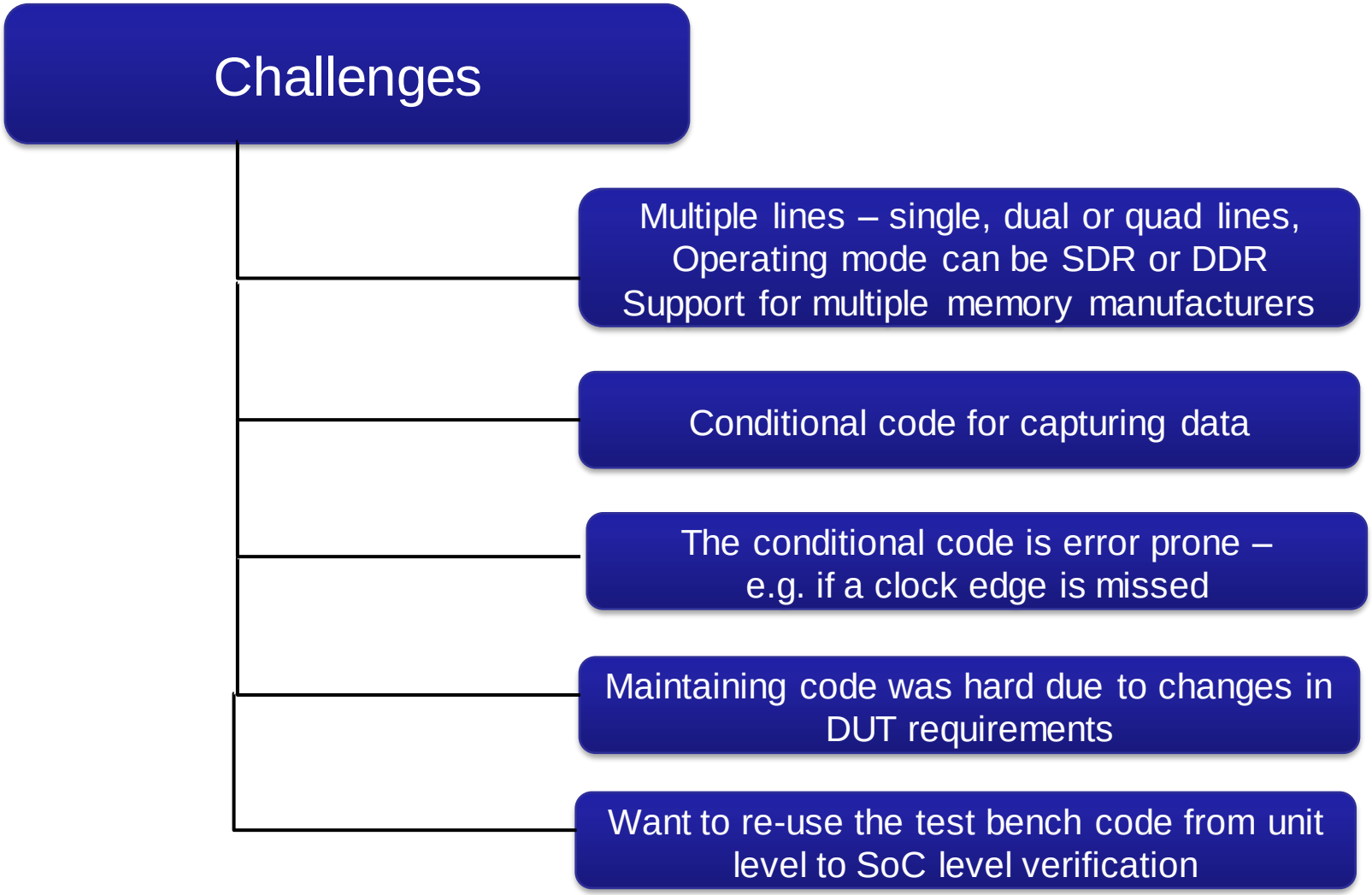
Using Specman Denali Interface (SNDI) to build a robust Data Checking Mechanism

Dr. Mike Bartley, Test and Verification Solutions

Agenda

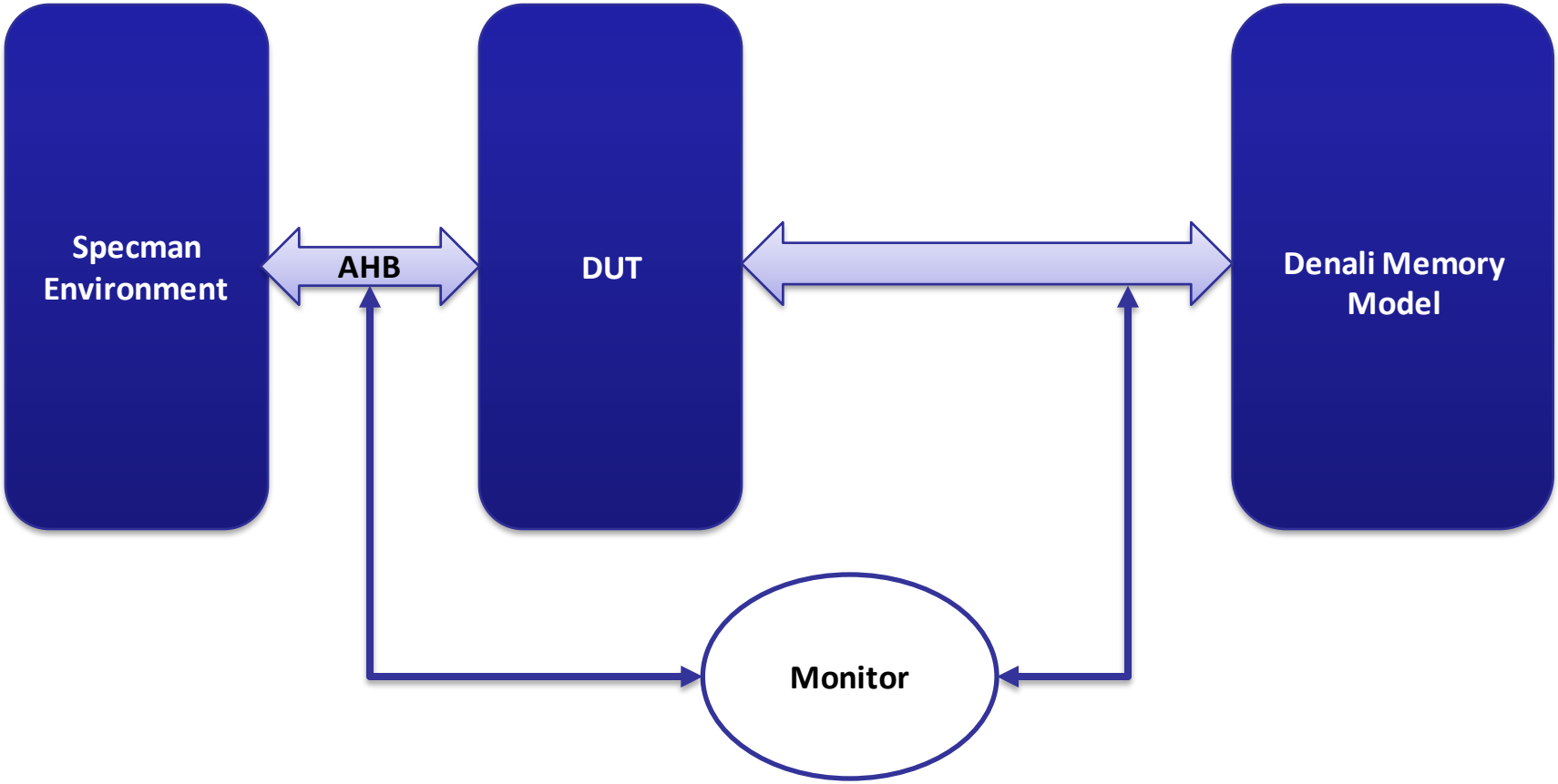
- Challenges in building data checkers for complex memory interfaces
- Overview of existing Verification environment – Verification Challenges
- The alternative – SNDI Callback mechanism
- How Callbacks work
- New Verification environment – With Callbacks
- Implementing Callbacks
- Benefits

Building data checkers for complex memory interfaces

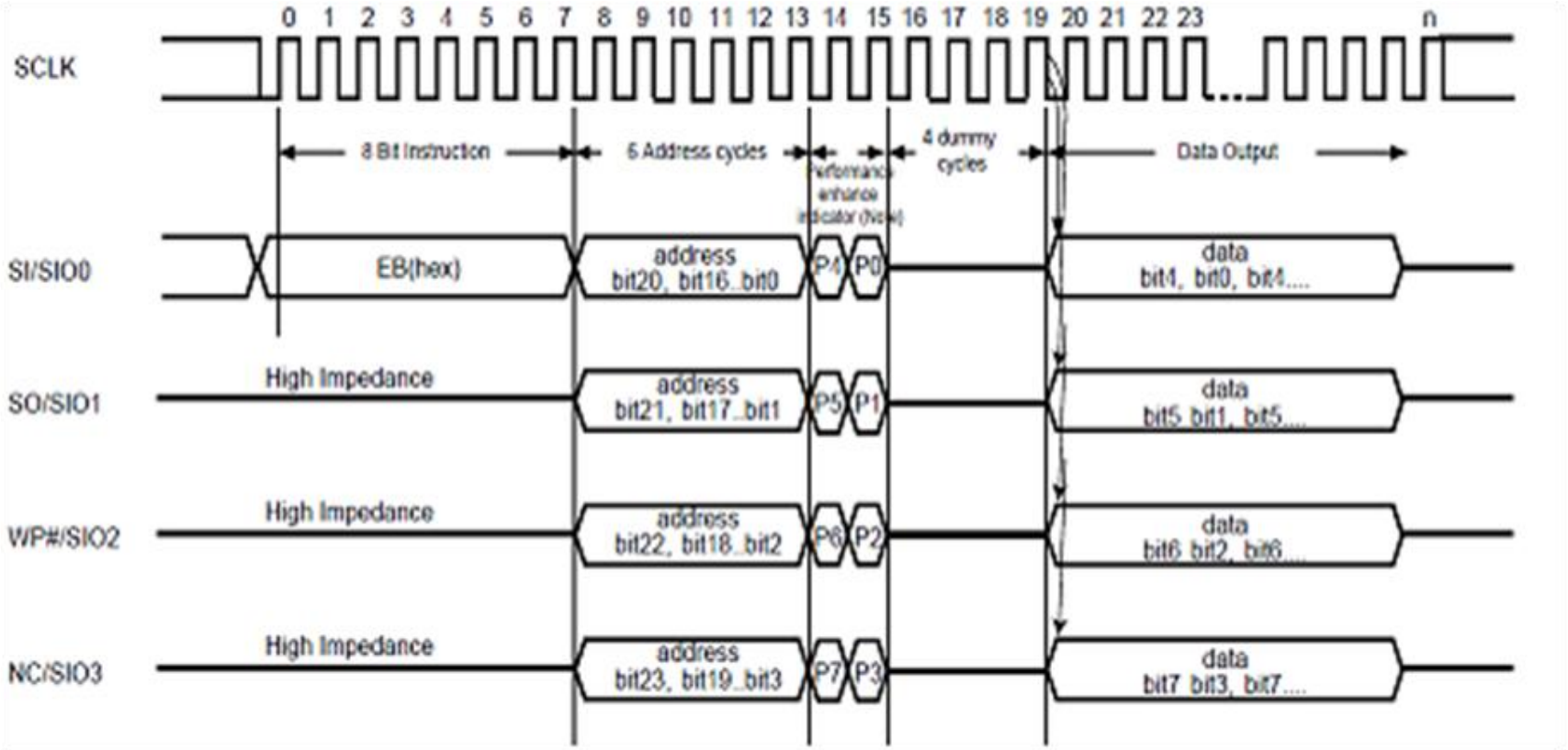


Old Verification Environment

- QuadSPI Interface
- Denali flash memory model
- Specman based environment



Verification Challenges - Complex Memory Interface



- Data can be captured on single, dual or quad lines
- Sampling in SDR or DDR mode
- Different combinations of command, address and data

Modeling of a Write operation

- Flash memory can be written only under certain conditions
  - eg. write 1s to initialize memory
- This adds to the complexity of the conditional code
- Denali model models this behavior
  - Before data is written into Denali model, a write sequence needs to be initiated

The alternative – SNDI Callback mechanism

TVS
Test and Verification Solutions

A mechanism to access or capture activity to and from Denali memory models

- Functions as a virtual scoreboard
- Simplifies testbench data management
- Provides an automatic logging mechanism

How does a callback work?

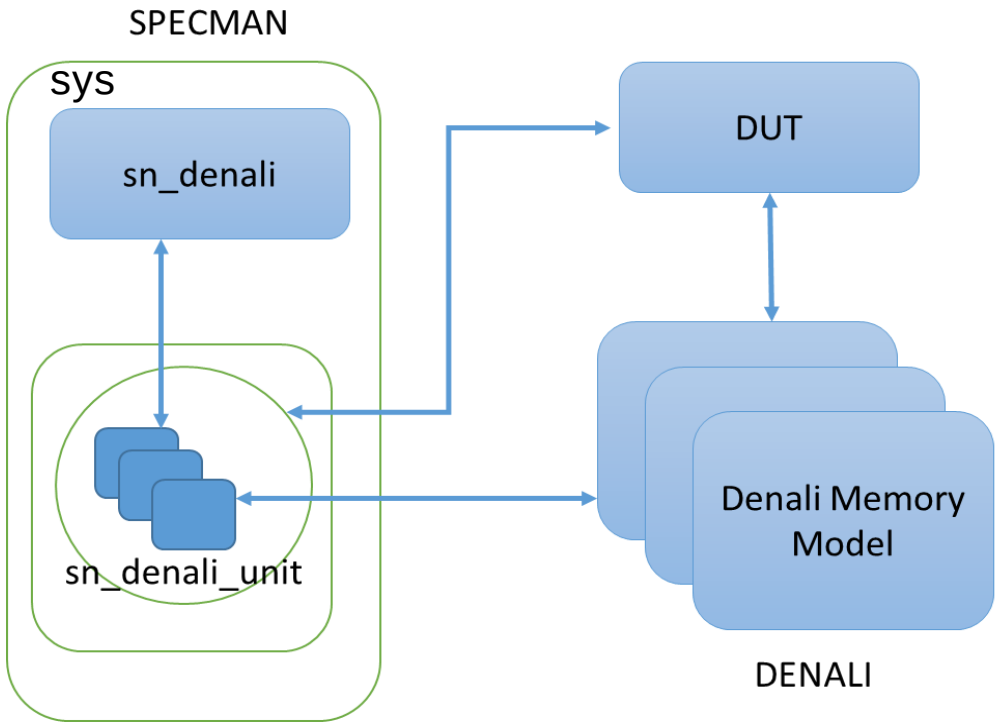
Callbacks use two predefined, user-accessible e structures for the SNDI interface : sn_denali and sn_denali_unit

sn_denali

- One global instance
- Has 4 associated methods

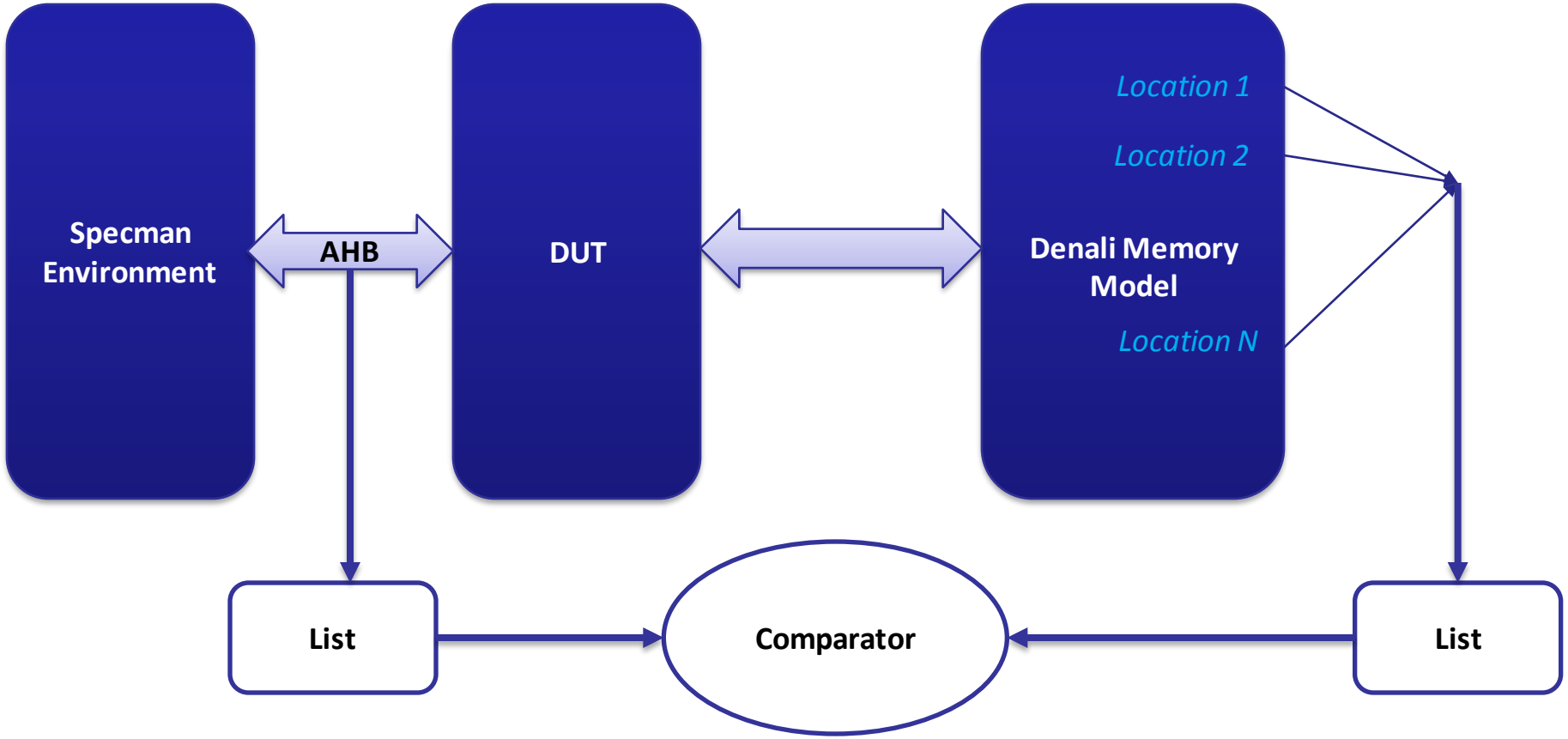
sn_denali_unit

- Abstract unit type
- Provides access to the built-in read and write methods



New Verification Environment (With Callbacks)

- Callbacks eliminate need to sample data at interface



Implementing Callbacks

- **Initialization to load the Denali shared library, initialize the Denali simulator and bind the SN_DENALI_MEMORY_UNIT instances to their API counterparts**

template macro

```
<'
SN_DENALI_MEMORY_UNIT sndi_cb using width=DWIDTH;

extend sys {

   sndi_cb_u1: sndi_cb is instance;

   keep sndi_cb_u1.hdl_path() == "~/top_specman/<SOME_DENALI_MODEL>";

   event clk is change('~/testbench/clk')@sim;
};
```

SNDI is automatically initialized when the first instance of sn_denali_unit is generated

Implementing Callbacks (Continued)

```
extend sndi_cb {
  process_cb() @access_event is {

    while TRUE {
      -- DETERMINE HOW MANY SNDI CALLBACK ITEMS NEED TO BE PROCESSED

          var n : uint = cb_data_size();
          var callback: uint;
          varaddr: uint;
          var data: uint (bits: DWIDTH);
          var access: sn_denali_access_type;
          outf("[%d] : %d DENALI callback item(s) received in %s\n", sys.time, n, e_path ());
              for callback from 0 to n-1 do {
                get_cb_data_item(callback, addr, data, access);
                print callback, addr, data, access using hex;

-- CAPTURE THE DATA DIRECTLY FROM DENALI MODEL IRRESPECTIVE OF THE ACCESS TYPE
-- THIS PROVIDES ON-THE-FLY  ACCESS INSTEAD OF BACKDOOR
-- SEPARATE CHECKS ON ADDRESSES
-- MONITOR WILL GIVE THE ACCESS TYPE FOR GENERATED DATA

          sys.<SOME_PATH>.agent.mon.CAPTURE_DATA_L.add(data);
                print sys.<SOME_PATH>.agent.mon.CAPTURE_DATA_L using hex;
                out("---------------------------------------");
              };
              out("****** [ SPECMAN/DENALI INTERFACE ] *********");
              wait [1] * cycle;  // Until the next SNDI callback
          };
      };
        run() is also {
            start process_cb();
        };
      };
```

Benefits

TVS
Test and Verification Solutions

- Eliminated lengthy lines of code
- Ensured the intended data was written and read correctly
- Easy to maintain code
- Possibility of mismatches due to malfunctioning of data checkers were eliminated
- Upto 50% savings on testbench writing and maintenance on a 4-month project