2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 15-16, 2024

Solving verification challenges for complex devices with a limited number of ports using Debugports

Shyam Sharma, Cadence Design Systems

Shravan Soppi, Cadence Design Systems

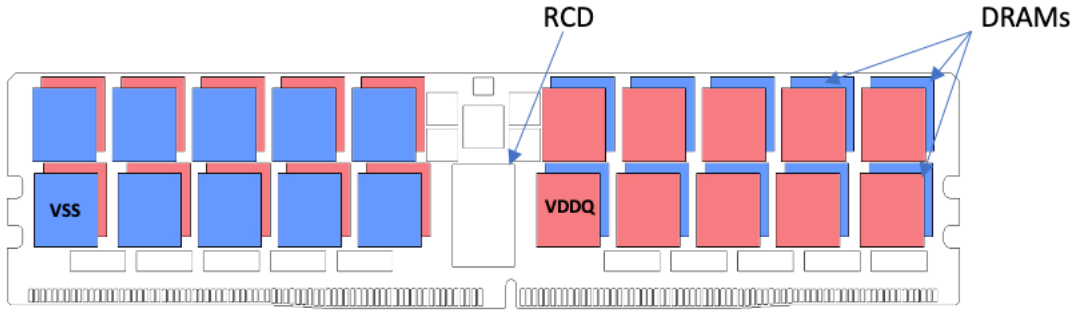
accellera SYSTEMS INITIATIVE

Agenda

- Background and User requirement.
- Solution :
  - Debugports Overview and advantages.
  - Debugports for Hierarchal VIPs for SV
- Additional Complexity of hierarchical VIP device instances in systemc
- Results and Customer feedback

Background and user requirements

- Complex Verification IP models needs internal signal and logic block visibility for
  - View command and data flow
  - Debug/isolate design and stimulus errors.
- It should be viewable on simulator waveform viewer.
- Should be simulator independent and work with XM/VCS/QuestaSim as well as open source simulators like OSCI SystemC when applicable.
- User should be able to access the change events in simulator debugger.
- Internal component signal representation should have similar look and feel to rest of the port signals.



DDR5 RDIMM Raw Card

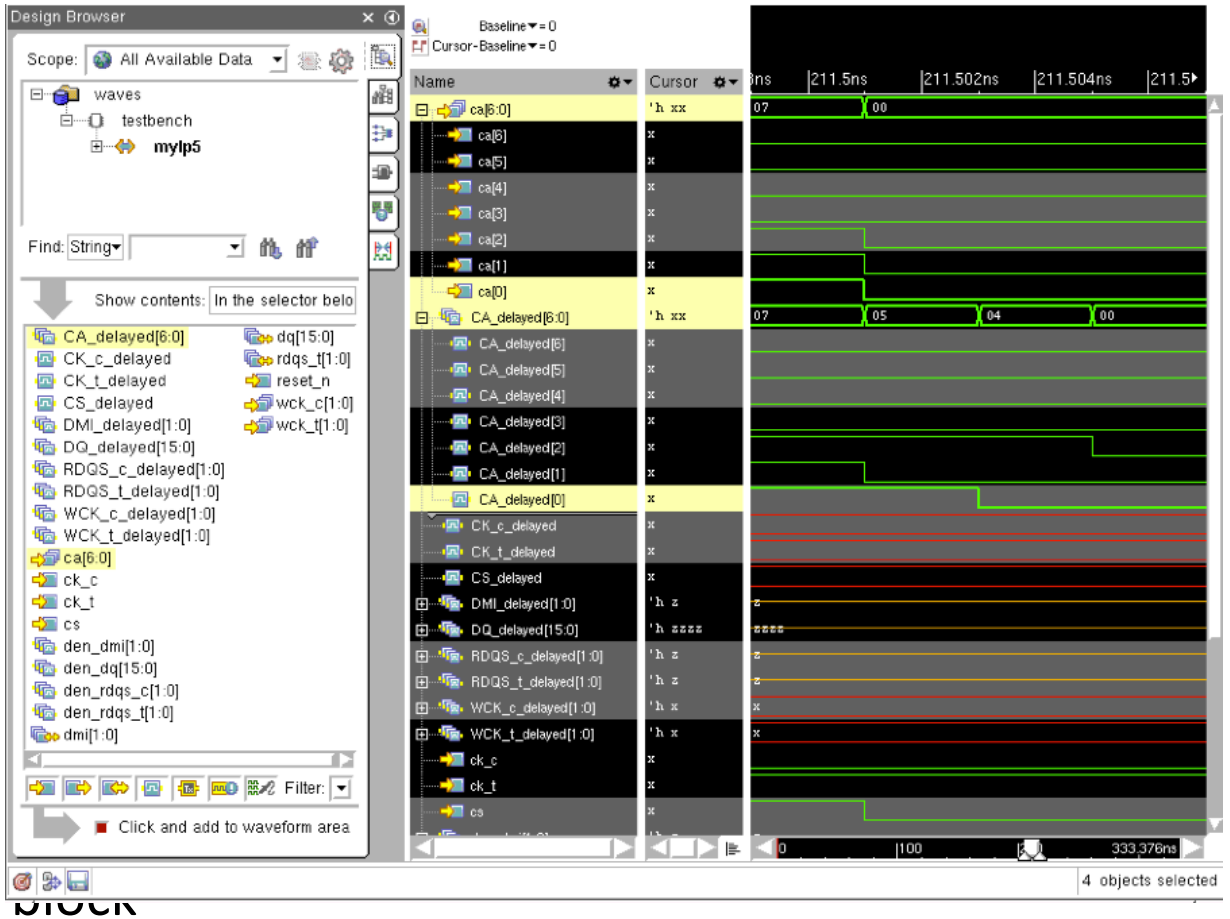
What is a debugport?

- A debugport is an HDL object that accurately reflects internal model state
  - Signal Debugports: Represents an internal signal of the model that is not a port.
    a) Show Components of a VIP that contains other VIP instances. These are usually standard Component Specification defined pins.
    b) Board Delay signals where the signal at VIP port is not the same as VIP usage as there is a transmission delay from Port connector to the internal signal that Device/Initiator/Target actually use.
    c) Internal signals that are useful to see for user like Internal Write Leveling pulses, Internal clocks etc.
  - Variable Debugports: Represents an internal logic block of the model.

    Typically used to display variables that model already tracks.
    a) Commands
    b) Read/Write latencies
    c) Composite timing parameters or equations.
    d) Per bit setup/hold/pulse windows or errors.

Debugports Advantages

- Easy to Use
  - Available in the Design hierarchy
  - Easy to generate using PV (batch command or using GUI)
- Supports SV, SV-UVM, SystemC
- Simulator independent (works with VCS/XM/MTI).
- Small performance impact
  - We saw ~4% performance impact with one of the test with signal debugports enabled.
- Users can treat a debugport just like any other HDL object
  - Send it to the waveform viewer
  - Add it to an event control of an always block
  - Print its value from a $display



Debugports Example: DDR5 DIMM component signal display in SV

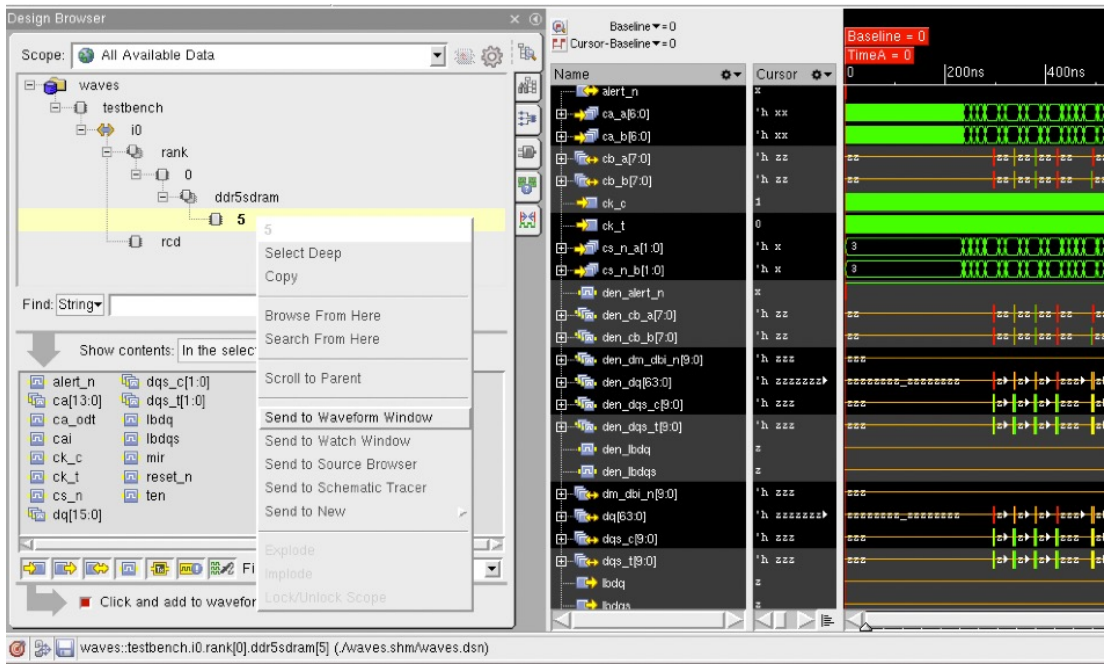
- Modify the VIP SV wrapper to add the logic variables representing the internal component ports in same hierarchical structure.

```
//
//   ddr5dimmm.
//           RCD
//           rank[0].ddr5sdram[0]
//            ..
//           rank[0].ddr5sdram[19]
//           rank[1].ddr5sdram[0]
//            ..
//           rank[1].ddr5sdram[19]
//
   localparam numRanks             =  2;
   localparam numDdr5sdramComponents = 10;

       // ------------------------------
       // <<--- BEGIN RCD DEBUGPORTS --->>
       generate
         begin : rcd
           var logic derror_in_a_n;
           var logic alert_n;
           var logic drst_n;
           var logic dck_t;
           var logic dck_c;
config_ddr5dimm_h1o1su.sv
           var logic derror_in_b_n;
           var logic dlbs_b;
           var logic dlbd_b;
           var logic qrst_a_n;
           var logic qrst_b_n;
           var logic bck_a_t;
           var logic bck_a_c;
           var logic brst_a_n;
           var logic bck_b_t;
           var logic bck_b_c;
           var logic brst_b_n;

           initial
              $__internal__vip_set_access(derror_in_a_
a_c,qcck_a_t,qcck_a_c,qdck_a_t,qdck_a_c,qack_b_t,qack_b
a_n,dca_a,dpar_a,dcs_b_n,dca_b,dpar_b,bcs_a_n,bcom_a,bc
qbcs_b_n,qbca_b,qlbs,qlbd,dlbs_a,dlbd_a,derror_in_b_n,d
_t,bck_b_c,brst_b_n);
```

- Update the wrapper logic variable.
- VPI apis provide very easy access to the variables and update them.



Complexity of hierarchical VIP devices instances in systemC

- Complexity of representation of hierarchical VIP devices in wrapper
  - Creation of wrappers representing the multi-layered sub-components can be confusing.
- Complexity of Data Update with similarly named sub-components.
  - Multiple Dram and Data buffer sub-components are present under each DIMM. Each having pins with same names. It can be difficult to identify which pin belongs to which instance if not properly grouped and named.
- Each sub-instance does not get any connection from any region as they can be just re-presentative instances with no actual drivers. As those instances are internal to VIP.
- There can be multiple instances of same kind with same signal names. Mapping systemC wrapper instances with correct internal instance of VIP is a challenge.

Solution

- Create wrapper for each sub-instances with same naming convention as VIP internal instances. (This is auto generated by our wrapper generator tool)
- A unique identifier for each instance is generated based on its full hierarchical path name.
- The main VIP instance holds a list of all the sub-instances created within its hierarchy. And are created before VIP instance is instantiated.
- While creating VIP internal instances, the library references the wrapper instance and map the unique identifiers to internal VIP instances using full path names.

```
void jedec_ddr5rdimm_32gbyte_2r_x4_6400an :: instantiate (const char *c, const char *n )
 {
  int numPins, i;
  int busSize;
  DENALImodeT pinMode;
  char *pinName;
  std::string cInst(n);
  std::string compName;
  denaliInst = (InstanceStruct*) calloc(1,sizeof(InstanceStruct));
  systemCsetCurrentDenaliInstance(denaliInst);
  systemCaddDenaliInstance(denaliInst);
  denaliInst->classobj = this;
  denaliInst->instList = new std::vector<model *>();
  denaliInst->instList->push_back(this);
  denaliInst->dpEvents = new std::unordered_map<int, std::vector<Debugport_Event *>>();

  //dram_debugports[][]
  for (i=0;i<rank;i++) {
    for (int j=0; j<components; j++){
      compName = cInst+"_rank["+std::to_string(i)+"]_ddr5sdram["+std::to_string(j)+"]";
      dram_debugports[i][j] = new jedec_ddr5_8g_x4_6400an_debugports(compName.c_str());
      dram_debugports[i][j]->setParent(denaliInst);
      denaliInst->instList->push_back(dram_debugports[i][j]);
    }
  }

  //rcd_debugport
  compName = cInst+"_rcd";
  rcd_debugport = new jedec_ddr5rcd_6400_debugports(compName.c_str());
  rcd_debugport->setParent(denaliInst);
  denaliInst->instList->push_back(rcd_debugport);
```

Solution – sub-instance wrappers

```
struct ddr5rcd_debugport : public model {
  sc_signal<sc_logic> dck_t;
  unsigned int dck_tQuark;
  sc_signal<sc_logic> dck_c;
  unsigned int dck_cQuark;
  sc_signal<sc_logic> drst_n;
  unsigned int drst_nQuark;
  sc_signal<sc_lv<jedec_ddr5rcd_6400_pin_size_dcs_a_n> > dcs_a_n;
  sc_lv<jedec_ddr5rcd_6400_pin_size_dcs_a_n> dcs_a_nTmp;
  unsigned int dcs_a_nQuark;
  sc_signal<sc_lv<jedec_ddr5rcd_6400_pin_size_dca_a> > dca_a;
  sc_lv<jedec_ddr5rcd_6400_pin_size_dca_a> dca_aTmp;
  unsigned int dca_aQuark;
  sc_signal<sc_logic> dpar_a;
  unsigned int dpar_aQuark;

  void updateDebugports(void);

  SC_HAS_PROCESS(ddr5rcd_debugport);
  ddr5rcd_debugport( sc_module_name nm):model(nm)
  {
    SC_METHOD(updateDebugports);
    sensitive << dp_event;

    dck_tQuark =  g_quark_from_string("dck_t");
    dck_cQuark =  g_quark_from_string("dck_c");
    drst_nQuark =  g_quark_from_string("drst_n");
    dcs_a_nQuark =  g_quark_from_string("dcs_a_n");
    dca_aQuark =  g_quark_from_string("dca_a");
    dpar_aQuark =  g_quark_from_string("dpar_a");

    instQuark = g_quark_from_string(nm);
  }
};
```

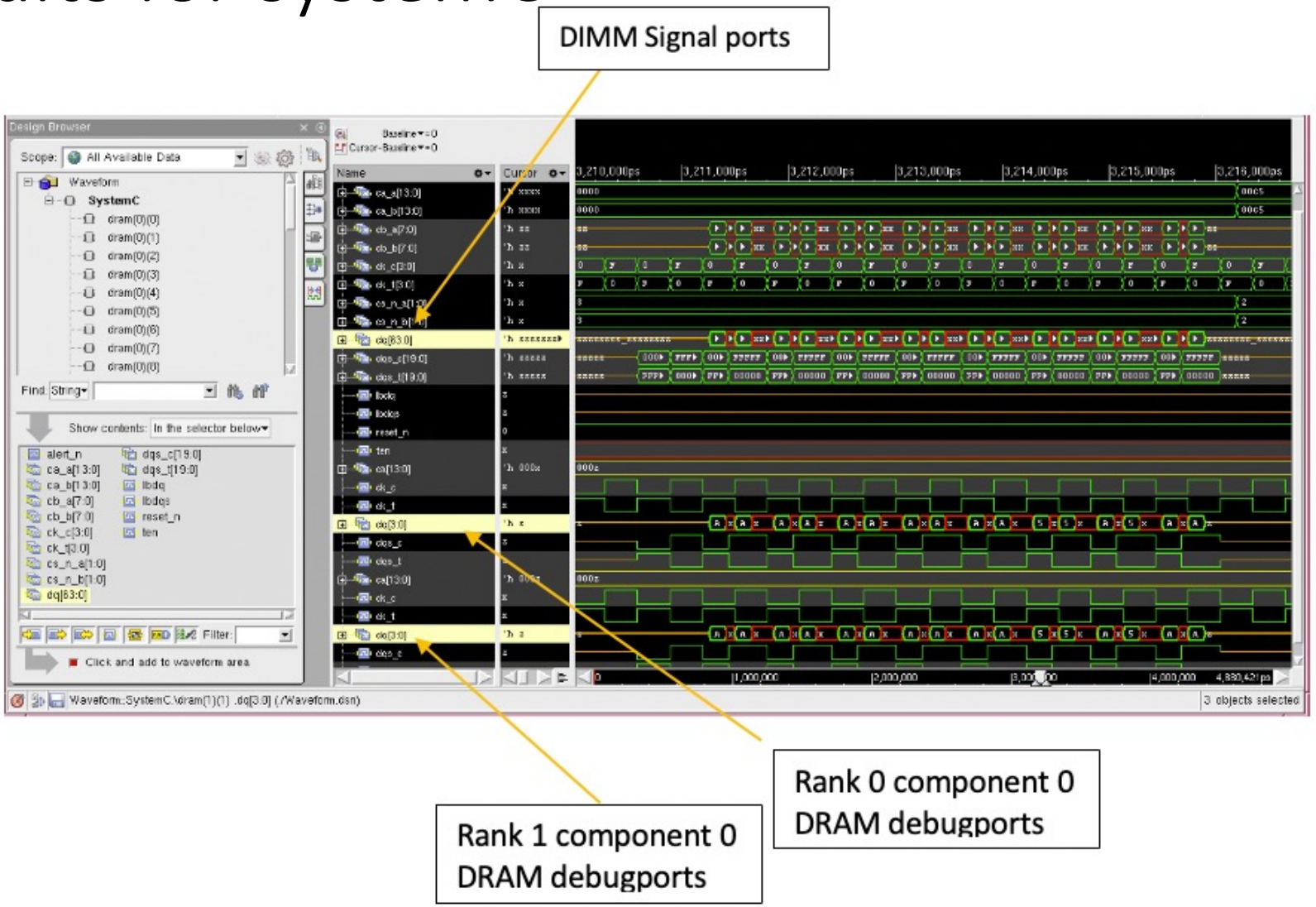
```
void ddr5rcd_debugport::updateDebugports(){
  auto eventList = (*p->dpEvents)[instQuark];

  while (!eventList.empty()) {
    Debugport_Event *tmpData = eventList.back();

    eventList.pop_back();

    int tmpQuark = g_quark_from_string(tmpData->name);
    if (dcs_b_nQuark == tmpQuark) {
        for (int i = 0; i < tmpData->width; i++) {
           dcs_b_nTmp[i] = DEN2SC( tmpData->value[i]);
        }
        dcs_b_n.write(dcs_b_nTmp);
    }
    else if (dcs_a_nQuark == tmpQuark) {
        for (int i = 0; i < tmpData->width; i++) {
           dcs_a_nTmp[i] = DEN2SC( tmpData->value[i]);
        }
        dcs_a_n.write(dcs_a_nTmp);
    }
      delete(tmpData);
  }
  (*p->dpEvents)[instQuark] = eventList;
}
```

Results for systemC



Debugports is being used by several of our customers who has provided positive feedback on improvements in debuggability of their tests.

Questions