A Generic Approach to Handling Sideband Signals

Markus Brosch, Salman Tanvir

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- What is a Sideband UVC ?
- Advantages
- Motivation
- Architecture
- Code
- Conclusion
- Q&A

What is a Sideband UVC ?

Protocol related signals

Protocol independent signals, so called Sideband Signals
e.g. Reset, HW-Config, IRQ/Alarms

Sideband-UVC:
A generic interface UVC that encapsulates the infrastructure
for driving/monitoring protocol independent signals

Advantages

- Unified sideband signal handling
- Fosters reusability/portability
- Time saving
- High code quality

Motivation

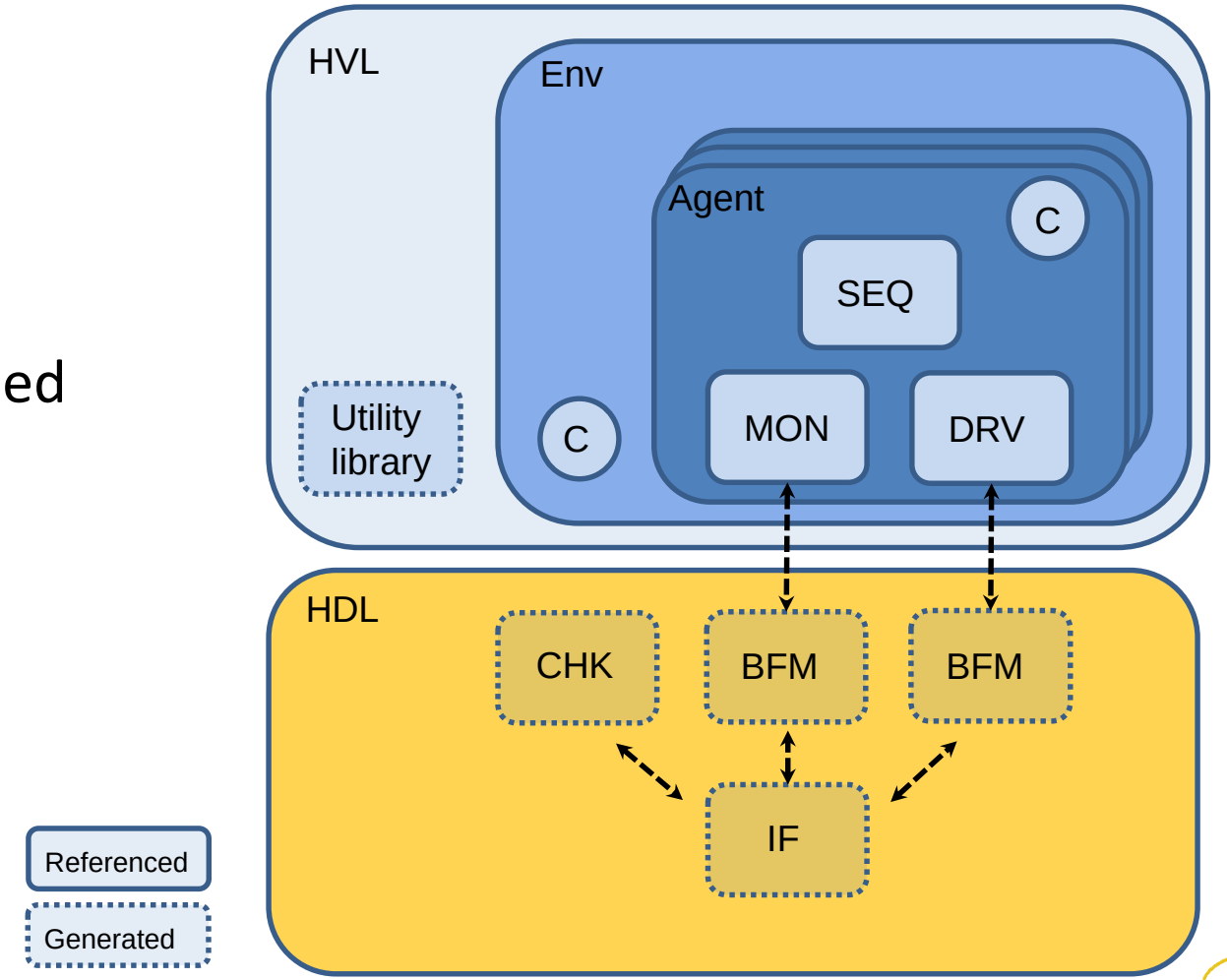
- Propose a SV UVM Sideband UVC Architecture
  - Signal Genericity: Support for multiple data types/sideband interfaces
  - Reusability: Efficient reuse across multiple projects
  - Out of the box monitoring/driving capability
  - Minimal code set

Architecture

- Split transactors (transaction level/BFM)
- Testbench to DUV connection (Abstract class approach)
- Transaction modelling
- Driving
- Monitoring
- Checks

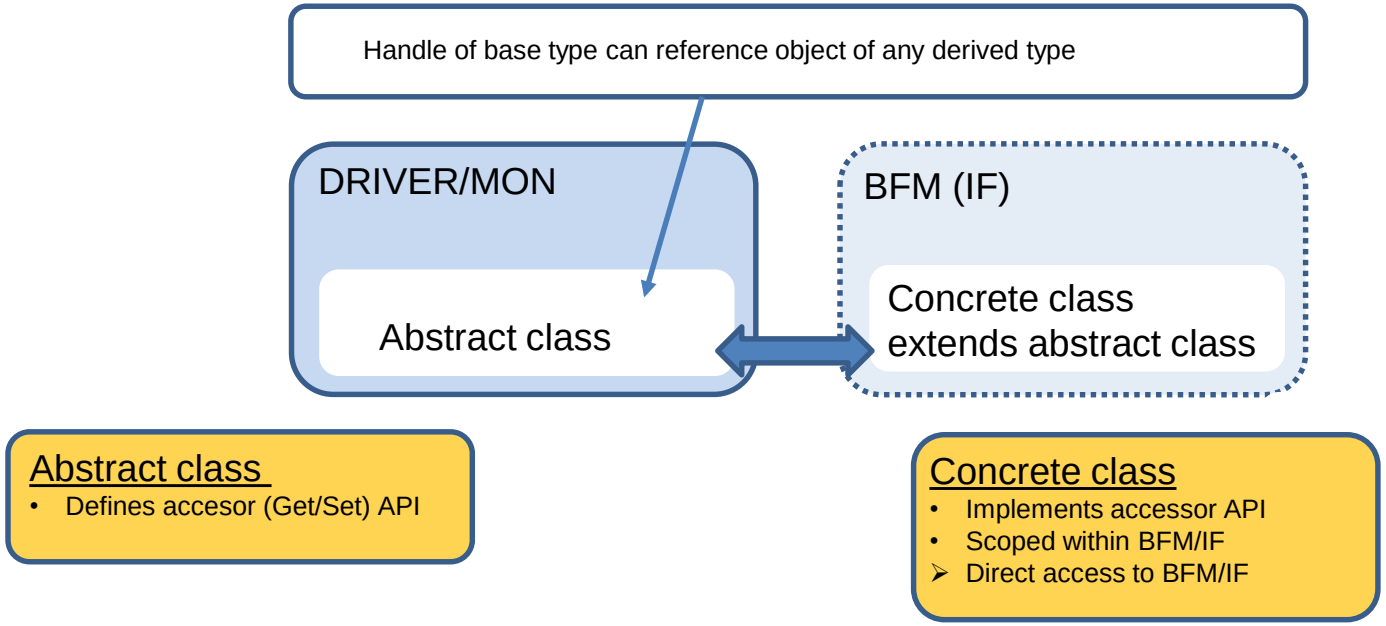
Split Transactors

- A sideband UVC needs to be extended by the user
- Propose to split the UVC into:
  - Reusable transactional class based component
  - User extendable (generated) interface based BFM part

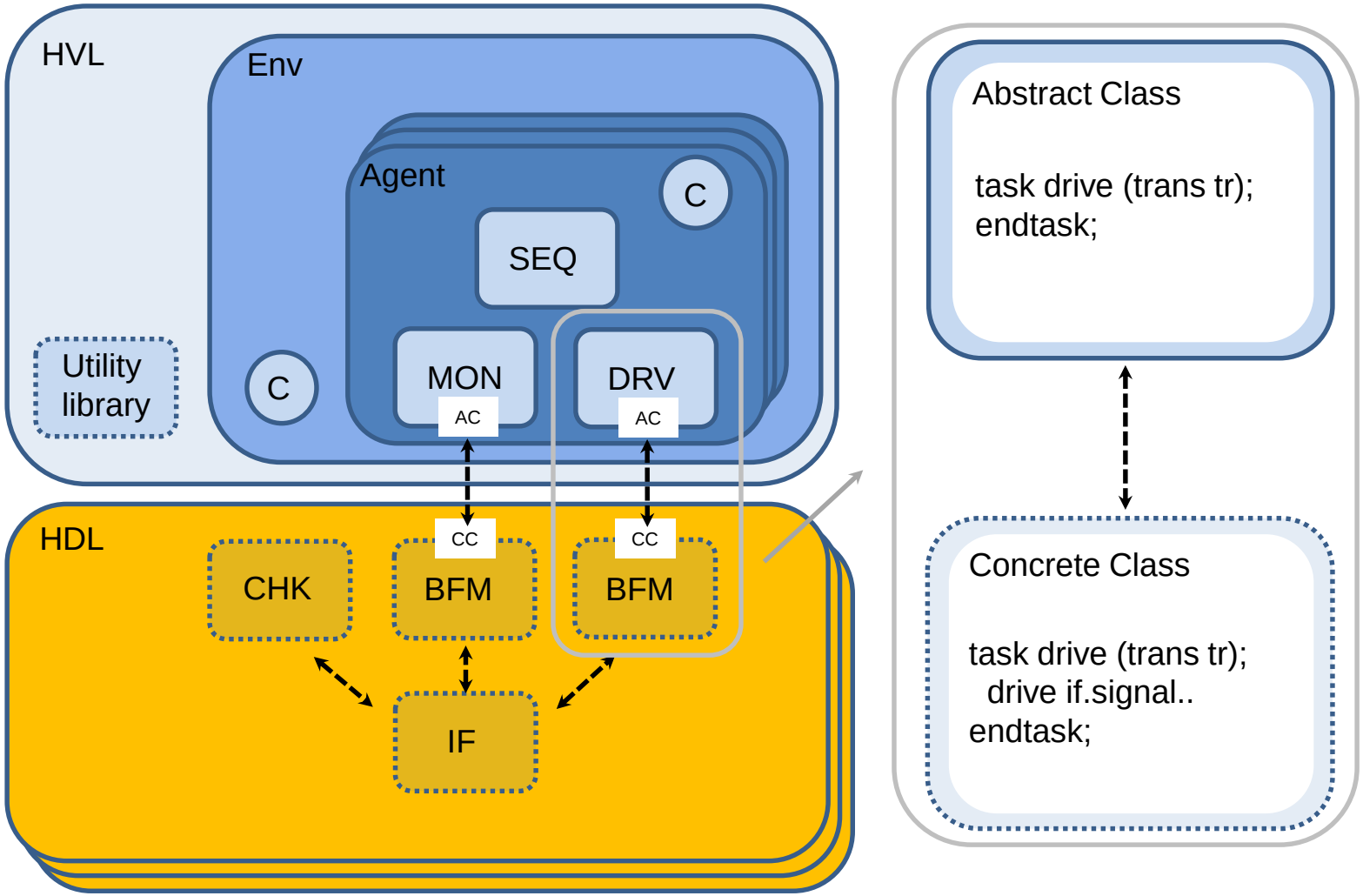


Testbench to DUV connection

- Virtual interface connections are type specific
  - Different interface types cannot be referenced
- Support of different interfaces is key for a Sideband UVC.
- The abstract class construct of SV solves the type specificity problem



Architecture



Transaction modelling

- Common data structure required to store the value of all signals of interest

Dynamic arrays	Parameterized classes
Concise code base Single transaction item Single line encode/decode	Larger code base Every signal type needs a parametrized extension Casting required for encode/decode
Additional string field needed for representation	Values are immediately visible

Transaction modelling

- Dynamics arrays

```
class ifx_sideband_monitor_item extends uvm_transaction;
  logic data[];                                    //signal value as array of logic
  string signal_id;                                //signal identifier
  ifx_sideband_item_kind_enum item_kind ;          //signal kind Real or Integer
  string context_id;                               //context identifier  env_name + agent_name + optional agent_idx
  string value;                                    //signal value as string
....

class ifx_sideband_driver_item extends uvm_sequence_item;
  logic data[];
  string signal_id;
  ifx_sideband_item_kind_enum item_kind;
  rand ifx_sideband_sync sync;                     //reference signal for Synchronous driving
  string value;                                    //signal value as string
....
```

How can signal data be encoded into and decoded from dynamic arrays?

Driver Encoding/Decoding

	Integral	Real
Encoding	tr.data[] = {<<{seq.signal_value}};	tr.data[] = {<<{$realtobits(seq.signal_value)}};
Decoding	s_if.signal = {<<{tr.data[]}};	logic[63:0] real_bits; real_bits = {<<{tr.data[]}}; s_if.signal = $bitstoreal(real_bits);

- Streaming operator used for encoding/decoding of signal data
- User does not need to take care of streaming the data
  - Driver: generated macros/sequences automate this

Monitor Encoding/Decoding

	Integral	Real
Encoding	tr.data[] = {<<{s_if.signal}};	tr.data[] = {<<{$realtobits(s_if.signal)}};
Decoding	mon.signal_mirror = {<<{tr.data[]}};	logic[63:0] real_bits; real_bits = {<<{tr.data[]}}; mon.signal_mirror = $bitstoreal(real_bits);

- Encoding handled automatically in monitor BFM
- Monitored transactions can be decoded:
  - Manually
  - Using a generated helper class

Monitor Encoding/Decoding

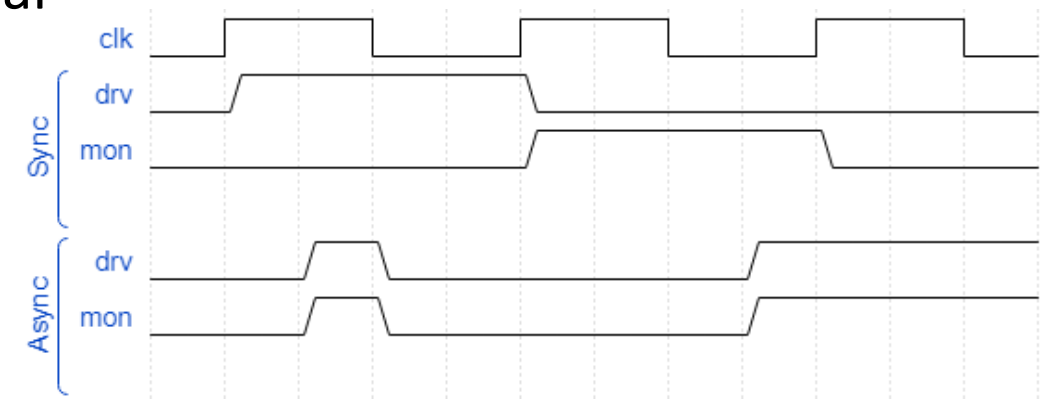
```
class module_monitor extends uvm_env;

  ifx_sideband_test_in_mirror test_in_0_sideband;
  bit                         reset;

  uvm_analysis_imp_sideband#(ifx_sideband_monitor_item, module_monitor) sideband_export;

  function void write_sideband(ifx_sideband_monitor_item t);
    // Option 1: Manual extraction
    if(t.context_id == "test_in_0" && t.signal_id == "reset" ) begin
      reset = {<<{t.data}};
    end

    // Option 2: Mirror helper class
    test_in_0_sideband.mirror(t);
  endfunction
endclass
```

mirror object encapsulates all monitored fields

mirror method automatically decodes transaction

Driving and Monitoring

- Driving
  - Synchronous to another sideband signal
  - Asynchronous (immediate)
- Monitoring
  - Synchronous
  - Asynchronous
  - Waiting tasks (to wait until a specified trigger event)
  - Sampling function (return immediate sample)



Checks

- Basic integrity assertions can be generated
  - Proper reset behavior
  - Valid signals (e.g. not X/Z)
- Help finding bugs very early in the design cycle

Code

- Code generation
  - Driving Sequence/Macro
  - Monitoring Helper Class
- Usage examples

Code generation

- UVC is structured into a re-usable and a project specific part.
- Reusable part referenced from a library.
  - Class based UVC (HVL environment, transaction, abstract class, agent, monitor, driver)
- Project specific part is generated.
  - HDL interfaces.
  - Checker interface.
  - Monitor/driver BFM's.
  - Driving sequence/macro library.
  - Monitoring helper class.
- Code generator based on Python (Mako).
- Excel/Json input specification file.

Excel Input

	A	B	C	D	E	F	G	H	I
1	**Agent**	**Signals**	**Type**	**Signal kind**	**Drv Init Value**	**Asynchronous Monitor**	**Synchronous Monitor**	**Checker**	**Comment**
2	agentY	reset	bit	active	0				reset signal
3		signal_a	real	active	0.0	DISABLE	posedge,signal_b	ref_rst,posedge,1.5	
4							negedge,signal_b		
5							,signal_b		
6							level==3,signal_b		
7		signal_b	logic [9:0]	active	0			x,z	
8								ref_rst,posedge,12	
9	agentZ	reset	bit	active	0				
10		signal_a	real	active	0.0	DISABLE	posedge,signal_b	ref_rst,posedge,1.5	
11							negedge,signal_b		
12							,signal_b		
13							level==3,signal_b		
14		signal_b	logic [9:0]	active	0			x,z	
15								ref_rst,posedge,12	
16									

Driving Sequence (integral)

```
//------------------------------------------------------------------------------
// Sequence to drive Sideband Env: envX, Agent: agentY, Signal: signal_b
//------------------------------------------------------------------------------
class ifx_sideband_envX_agentY_signal_b_seq extends ifx_sideband_default_seq;

  rand logic [9:0] value;

  `uvm_object_utils(ifx_sideband_envX_agentY_signal_b_seq)

  function new(string name = "ifx_sideband_envX_agentY_signal_b_seq");
      super.new(name);
  endfunction

  virtual function ifx_sideband_driver_item get_tr();
    get_tr           = new();
    get_tr.data      = {<<{value}};
    get_tr.value     = $sformatf("%x",local::value);
    get_tr.item_kind = INTEGRAL;
    get_tr.signal_id = "signal_b";
  endfunction
endclass
```

Driving Sequence (real)

```
//------------------------------------------------------------------------------
// Sequence to drive Sideband Env: envX, Agent: agentY, Signal: signal_a
//------------------------------------------------------------------------------
class ifx_sideband_envX_agentY_signal_a_seq extends ifx_sideband_default_seq;

  rand real value;

  `uvm_object_utils(ifx_sideband_envX_agentY_signal_a_seq)

  function new(string name = "ifx_sideband_envX_agentY_signal_a_seq");
      super.new(name);
  endfunction

  virtual function ifx_sideband_driver_item get_tr();
    logic [63:0] real_bits_v = $realtobits(value);
    get_tr           = new();
    get_tr.data      = {<<{real_bits_v}};
    get_tr.value     = $sformatf("%f",local::value);
    get_tr.item_kind = REAL;
    get_tr.signal_id = "signal_a";
  endfunction
endclass
```

Driving Macro (integral)

```
`define do_sideband_on_with(SEQR,WIDTH,SIG,CONSTRAINTS={}) \
  begin \
  ifx_sideband_default_seq __seq = ifx_sideband_default_seq::type_id::create(); \
  ifx_sideband_driver_item __item = ifx_sideband_driver_item::type_id::create(); \
  logic [WIDTH-1:0] value ; \
  if (!std::randomize(value) with CONSTRAINTS) begin \
    `uvm_warning("RNDFLD", "Randomization failed in do_sideband_on_with action") \
  end\
  if (!__item.randomize()  ) begin \
    `uvm_warning("RNDFLD", "Randomization failed in do_sideband_with action") \
  end\
  __item.signal_id = `"SIG`"; \
  __item.item_kind = INTEGRAL; \
  __item.data      = {<<{value}}; \
  __seq.tr = __item; \
  __seq.start(SEQR,this,-1,0); \
  end
```

Reference base macro

```
`define do_sideband_envX_agentY_reset(SEQR,CONSTRAINTS={}) \
    begin \
        `do_sideband_on_with(SEQR,1,reset,CONSTRAINTS) \
    end

`define do_sideband_envX_agentY_signal_a(SEQR,CONSTRAINTS={}) \
    begin \
        `do_sideband_real_on_with(SEQR,signal_a,CONSTRAINTS) \
    end
```

Generated macros (per signal)

Monitoring Helper Class

```
class ifx_sideband_envX_agentY_mirror#(int AGENT_IDX = 0) extends uvm_object;

  string context_id = "envX_agentY";

  //default signals
  bit reset;
  real signal_a;
  logic [9:0] signal_b;

  function void mirror(ifx_sideband_monitor_item t);
    logic[63:0] real_bits_v;
    if (t.item_kind == REAL) begin
      real_bits_v = {<<{t.data}};
    end
    if (t.context_id == context_id) begin
      case(t.signal_id)
          "reset": reset = {<<{t.data}};
          "signal_a": signal_a = $bitstoreal(real_bits_v);
          "signal_b": signal_b = {<<{t.data}};
          default:  `uvm_fatal(get_type_name(), {"UNDEFINED SIGNAL: ",t.signal_id})
      endcase
    end
  endfunction
endclass
```

Mirrored interface signals

Automated mirroring

Usage Examples

```
task seq_dummy::body();
  ifx_sideband_envX_agentY_signal_a_seq    envX_agentY_signal_a_seq;
  ifx_sideband_envX_agentY_signal_b_seq    envX_agentY_signal_b_seq;

  //------------------------------------------------------------------------------
  `uvm_info(get_type_name(), "START SEQ_DUMMY", UVM_LOW)
  //------------------------------------------------------------------------------
  ....
    `uvm_do_on_with(envX_agentY_signal_a_seq,p_sequencer.sideband_envX_agentY_sequencer,{value == 2.0;});
    `do_sideband_envX_agentY_signal_b(p_sequencer.sideband_envX_agentY_sequencer,       {value == 1.0;})
  ....
```

```
//------------------------------------------------------------------------------
`uvm_info(get_type_name(), "START SEQ_DRIVER_SYNC", UVM_LOW)
//------------------------------------------------------------------------------

// Synchronous driving usage examples

// Use wait tasks to synchronize and then drive asynchronously (drive signal a to 1.0 at first posedge of reset)
p_sequencer.sideband_envX_agentY_sequencer.agent_cfg.mon_proxy.wait_edge_integral_signal(POS_EDGE,"reset",1);
`uvm_do_on_with(envX_agentY_signal_a_seq,p_sequencer.sideband_envX_agentY_sequencer,{value == 1.0;});

// Macro example( signal a being driven to 2.0 at 2nd negedge of reset)
`do_sideband_seq_sync_on_with(ifx_sideband_envX_agentY_signal_a_seq,p_sequencer.sideband_envX_agentY_sequencer,
                              "reset",{value==2.0;tr.sync.ev_cnt ==2;tr.sync.edge_kind == NEG_EDGE;})

//------------------------------------------------------------------------------
`uvm_info(get_type_name(), "END SEQ_DRIVER_SYNC", UVM_LOW)
//------------------------------------------------------------------------------
```

Conclusion

- Fully functional SV Sideband UVC developed
  - Genericity (Support for multiple data types/sideband interfaces)
  - Concise code set (HVL referenced, HDL regenerated only)
    - HVL referenced
    - HDL regenerated only
  - Out of the box monitoring/driving capability
- Split transactor/abstract class concept can be reused to develop generic interface UVC'S
  - Adapt base transaction per interface protocol
  - Extend base transaction/BFMS to support other protocol flavors

Questions