Transaction Level Assertions in UVM

Steve Holloway

Dialog Semiconductor

June 23rd, 2015

SNUG Reading

Agenda

Transaction Level Modelling (TLM)

SystemVerilog Assertions

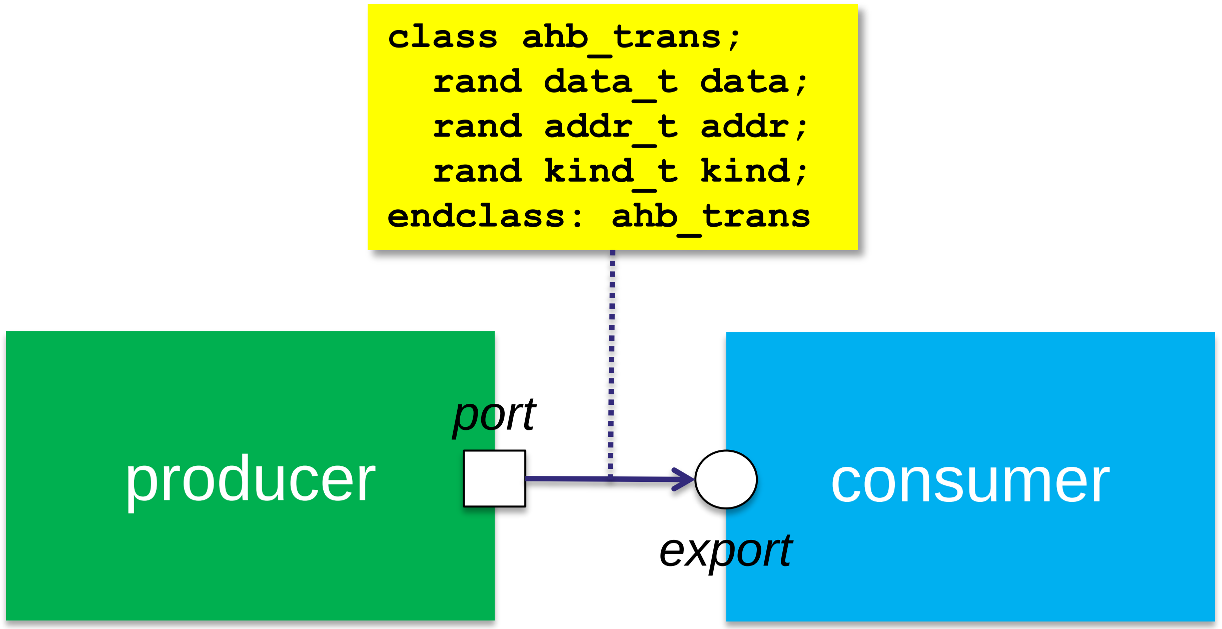
Transaction Level Assertions

UVM Split Monitor

Example TL Assertions

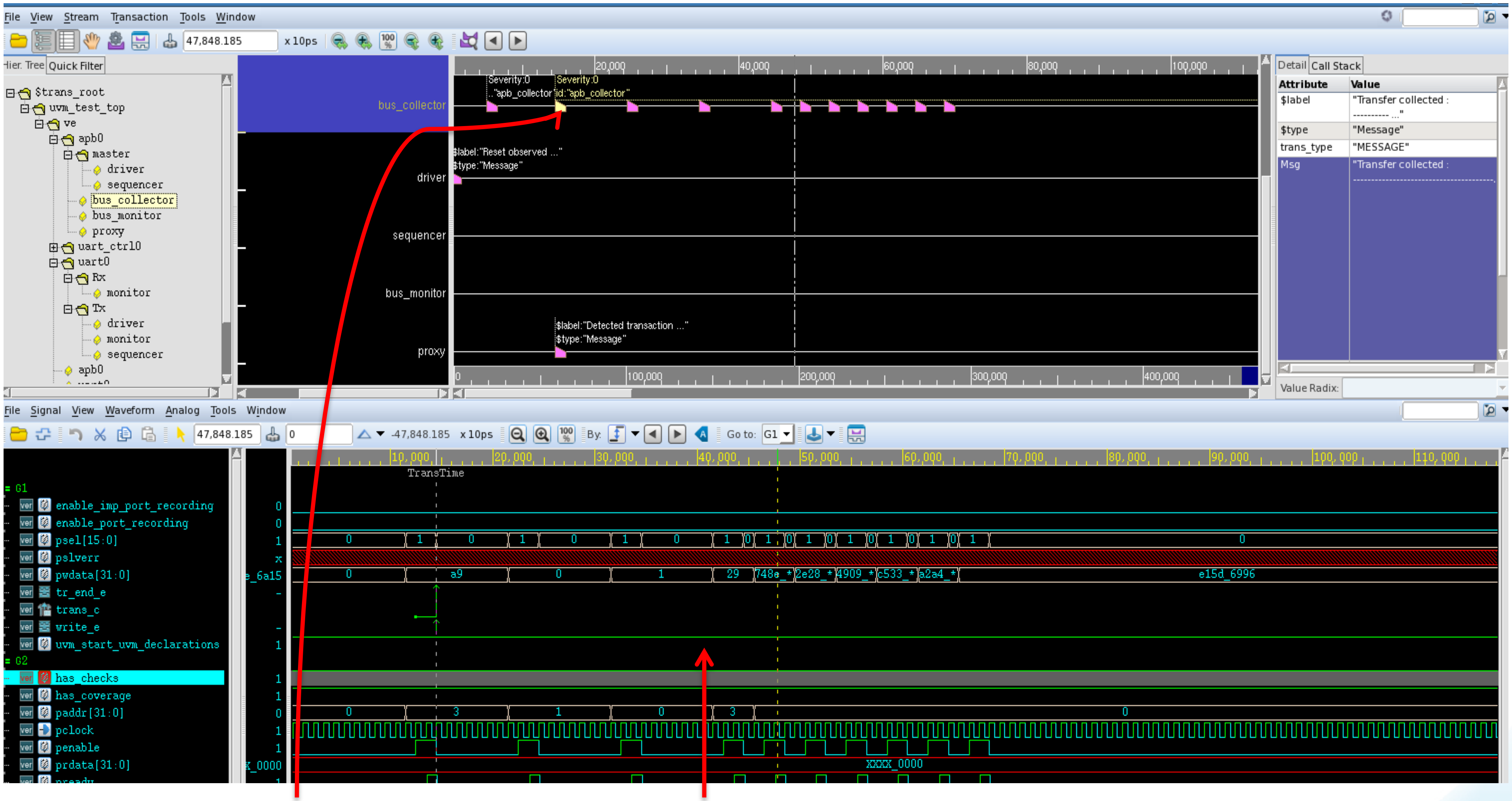
Transaction Level Modelling

- Abstract data types for bus transactions
- “Plug and play” connectivity between components
- Separate communication from implementation
- Easy debug and visualization



```
class ahb_trans;
  rand data_t data;
  rand addr_t addr;
  rand kind_t kind;
endclass: ahb_trans
```

Transaction Level Modelling

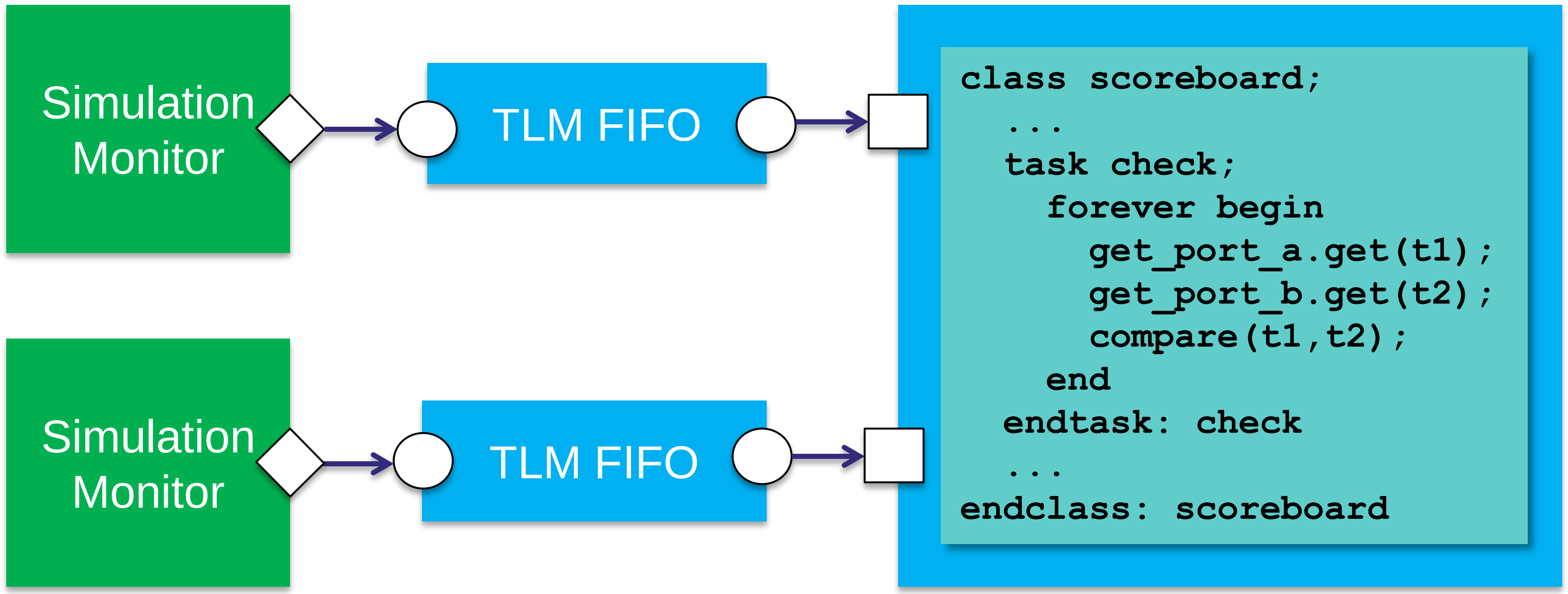


Transaction class

Bus signals

UVM and TLM

- Central concept for testbench communication
- Transaction based stimulus and checking
- Procedural code in scoreboard does TLM checks

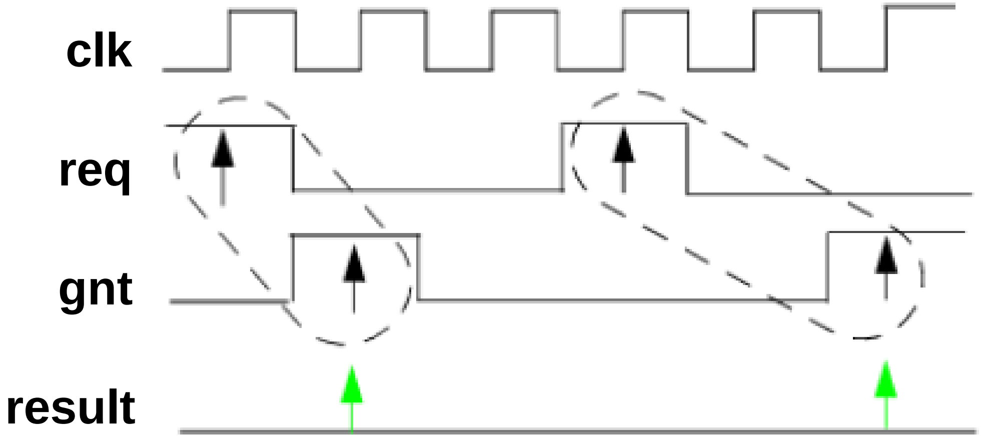


```
class scoreboard;
  ...
  task check;
    forever begin
      get_port_a.get(t1);
      get_port_b.get(t2);
      compare(t1,t2);
    end
  endtask: check
  ...
endclass: scoreboard
```

SystemVerilog Assertions

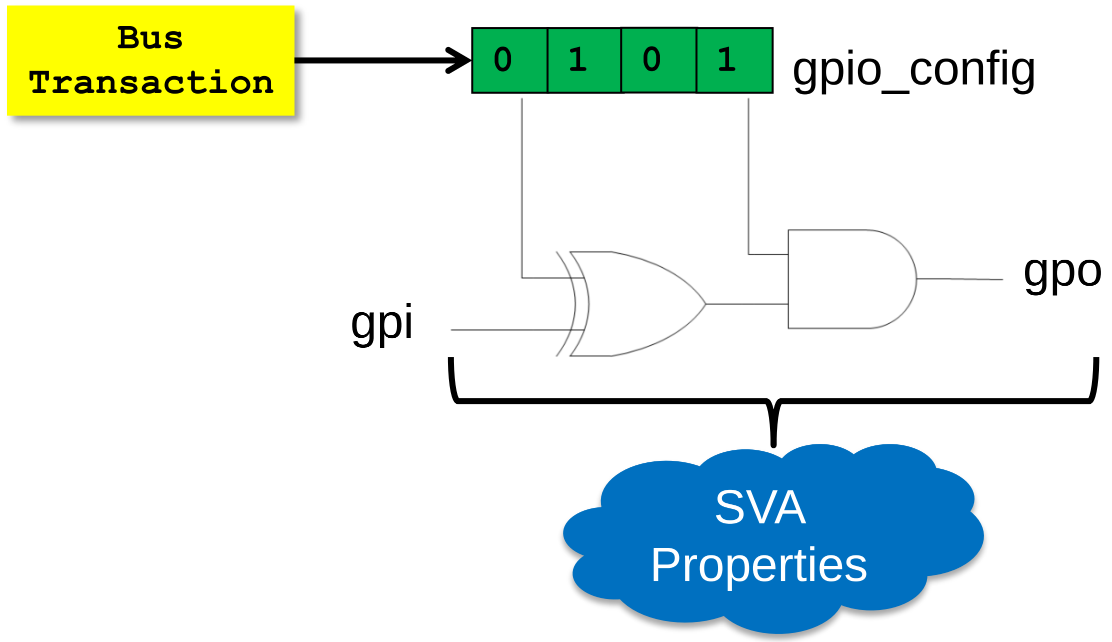
- Well – suited to temporal checks and complex sequences
- Concise compared to equivalent procedural code
- Declarative syntax – avoids re-coding DUT

```
if_req_eventually_gnt: assert property (
   @(posedge clk)
   disable iff(rst)
   (req |-> ##[1:2] gnt)
);
```

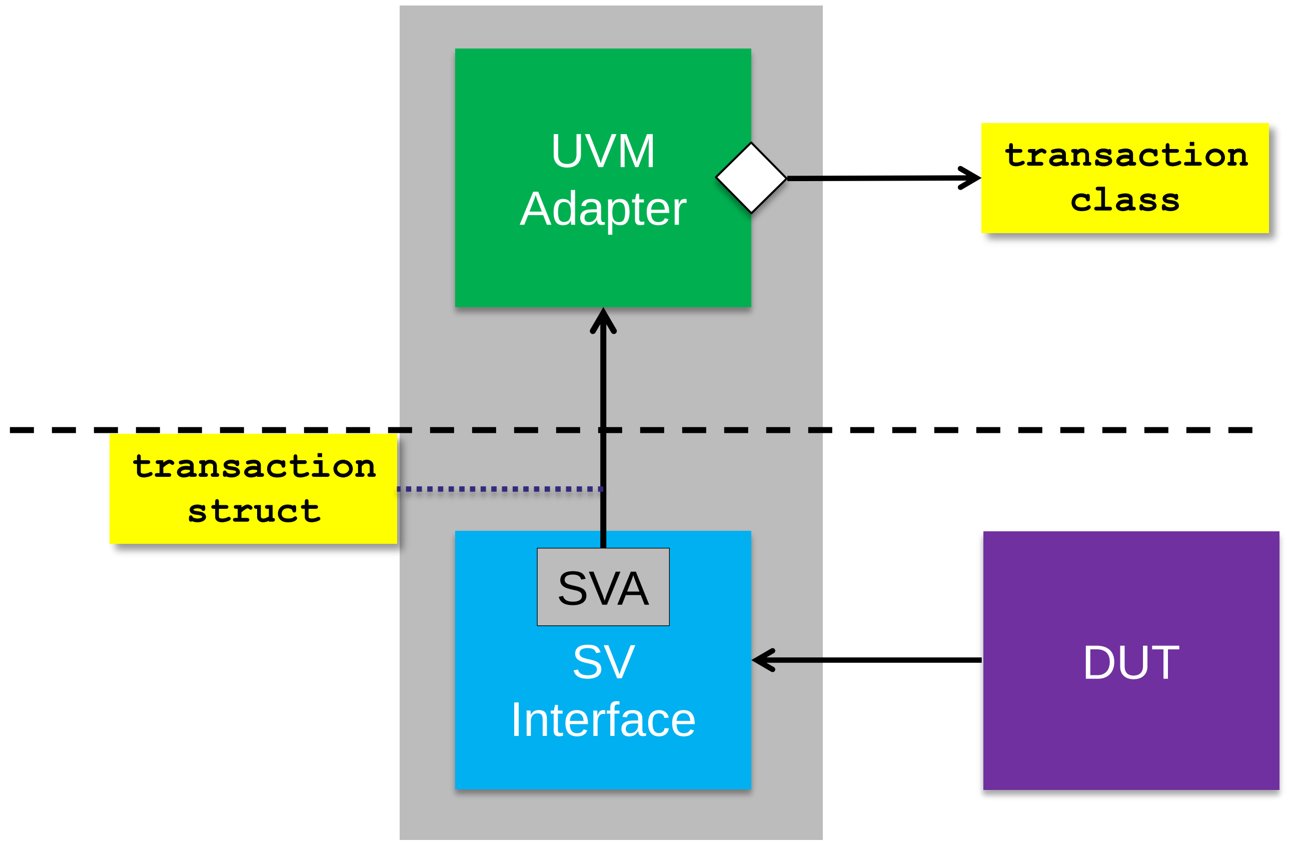


UVM and SVA

- Clear partition between temporal (SVA) and TL checking
- Data checking in UVM scoreboard
- Control checking in testbench with SVA
- How can we straddle this partition?



UVM Split Monitor

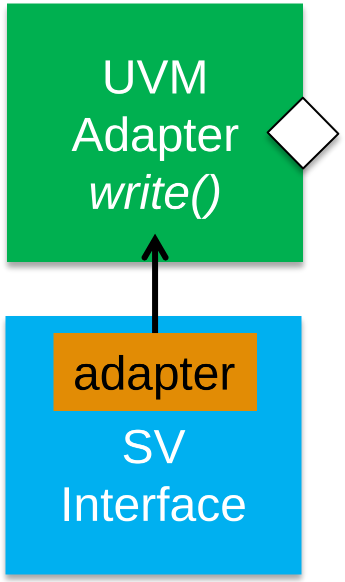


Split Monitor Interface

```
interface apb_if (input pclock, input preset);

  logic [ADDR_WIDTH-1:0] paddr;
  logic                  prwd;
  logic [DATA_WIDTH-1:0] pwdata;
  ...

  // Interface transaction struct
  apb_if_tr tr;

  // Handle to the UVM adapter
  apb_adapter adapter;
 ...

endinterface : apb_if
```

```
typedef struct {
  logic [31:0] addr;
  logic [31:0] data;
  logic        dir;
  time         start;
  time         finish;
} apb_if_tr;
```

Time stamps



Detecting transactions

- Local variables to store transaction attributes
- Function call to *write()* at end of transaction

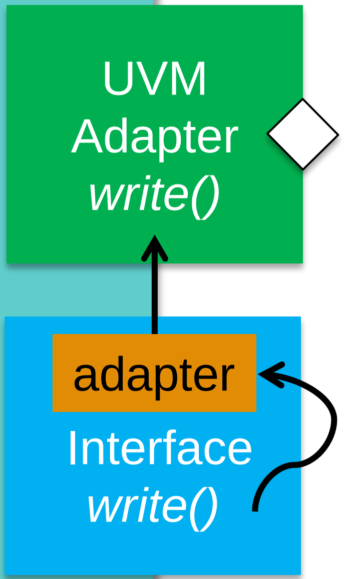
```
sequence apb_trans_s;
  logic [ADDR_WIDTH-1:0] addr;
  logic [DATA_WIDTH-1:0] data;
  logic dir;
  time  start;
  (<trans start condition>, addr = paddr, dir = prwd, start = $time)
  ##1
  ...
  (<trans end condition>, write(addr, data, dir, start));
endsequence: apb_trans_s
```

- Sequence cover directive

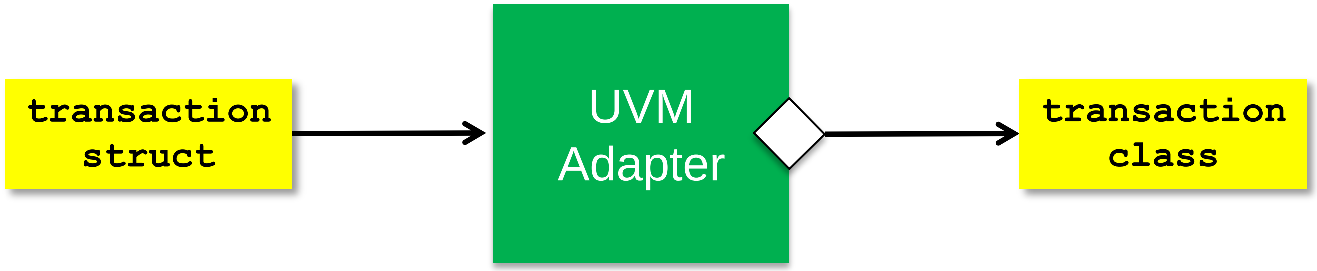
```
apb_trans_c : cover sequence(apb_trans_s);
```

Interface *write()* function

```
function void write(logic [PADDR_WIDTH-1:0]  addr,
                    logic [PWDATA_WIDTH-1:0] data,
                    logic dir,
                    time  start);

    // Assign struct members
    tr.addr   = addr;
    tr.data   = data;
    tr.dir    = dir;
    tr.start  = start;
    tr.finish = $time;

    // Call UVM adapter write() method
    adapter.write(tr);

endfunction: write
```



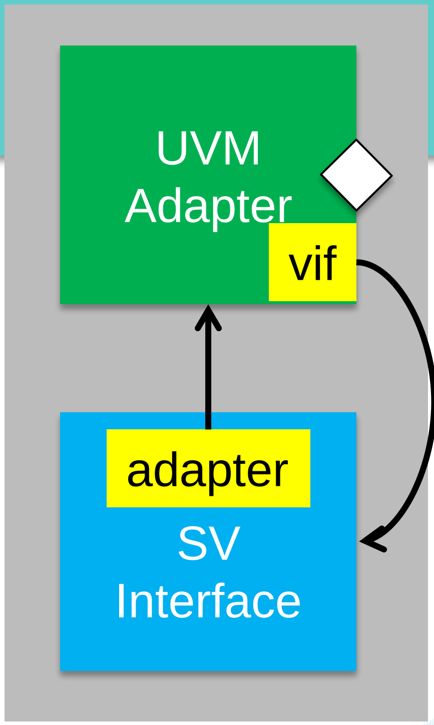
UVM Adapter

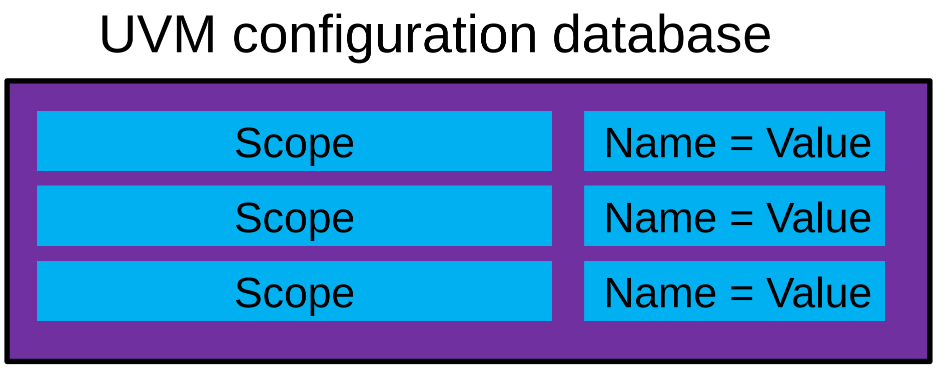


```
function void apb_adapter::write(apb_if_tr tr);
  // Create UVM transaction
  apb_transfer apb_tr = new;

  // Fill uvm transaction with info from struct
  apb_tr.addr      = tr.addr;
  apb_tr.data      = tr.data;
  apb_tr.direction = tr.dir ? APB_WRITE : APB_READ;

  // Set start and end times for transaction recording
  void'(begin_tr(apb_tr, .begin_time(tr.start)));
  void'(end_tr(apb_tr,   .end_time(tr.finish)));

  // Send transaction to analysis port
  ap.write(apb_tr);

endfunction: write
```

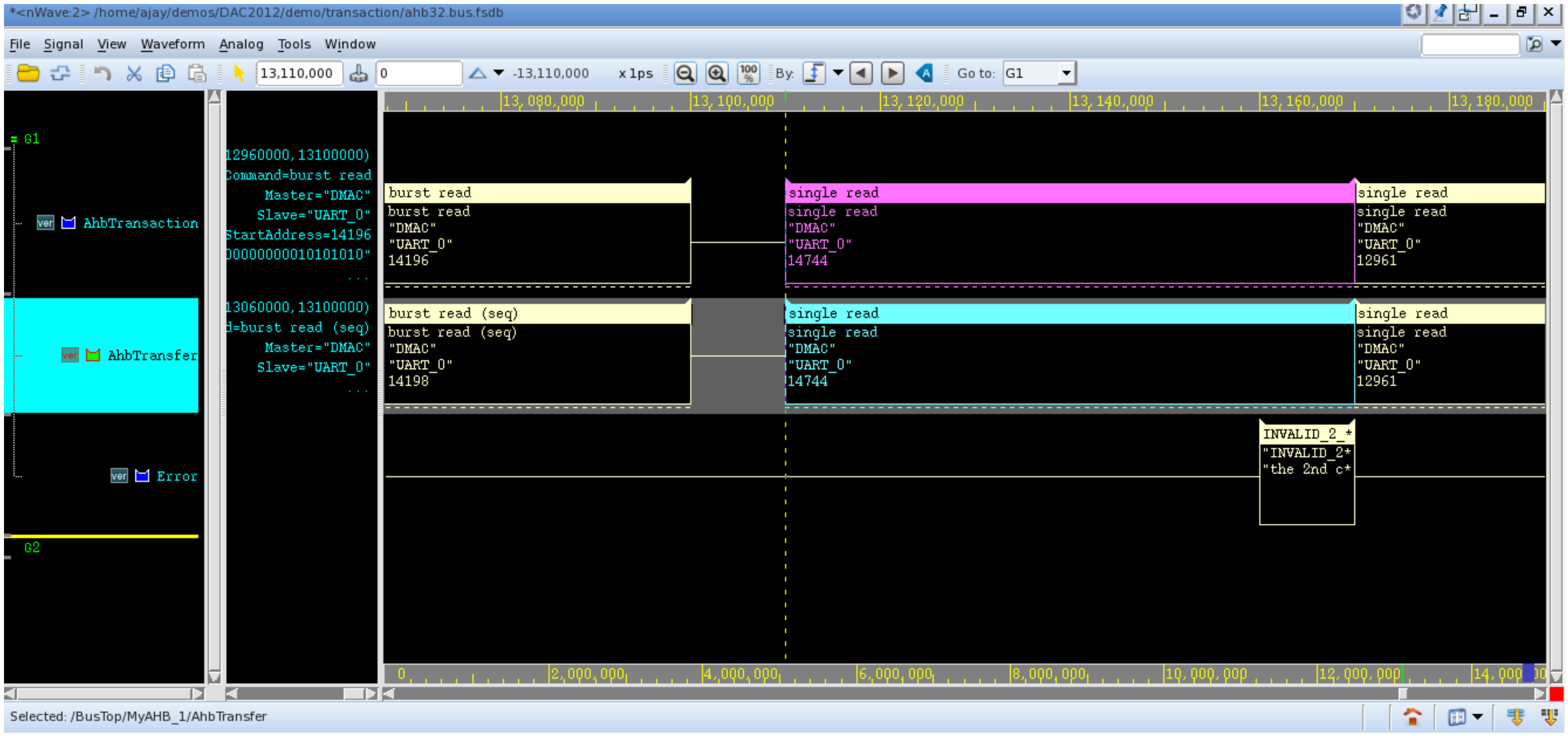
Connecting the Adapter

```
function void apb_adapter::build_phase(uvm_phase phase);
  super.build_phase(phase);

  // get the config setting for the virtual interface
  if (!uvm_config_db#(virtual apb_if)::get(this, "", "vif", vif))
    `uvm_fatal("NOVIF", {get_full_name(), ".vif"})

  // assign the back-pointer to the adapter
  vif.adapter = this;
endfunction : build_phase
```





Split Monitor Transactions



```
12960000,13100000)
Command=burst read
   Master="DMAC"
   Slave="UART_0"
StartAddress=14196
00000000010101010"
             ...
13060000,13100000)
d=burst read (seq)
   Master="DMAC"
   Slave="UART_0"
             ...
```

burst read	single read	single read
burst read	single read	single read
"DMAC"	"DMAC"	"DMAC"
"UART_0"	"UART_0"	"UART_0"
14196	14744	12961

burst read (seq)	single read	single read
burst read (seq)	single read	single read
"DMAC"	"DMAC"	"DMAC"
"UART_0"	"UART_0"	"UART_0"
14198	14744	12961

```
INVALID_2_*
"INVALID_2*
"the 2nd c*
```

Selected: /BusTop/MyAHB_1/AhbTransfer

Examples: Checking GPIO

- Register access sequence

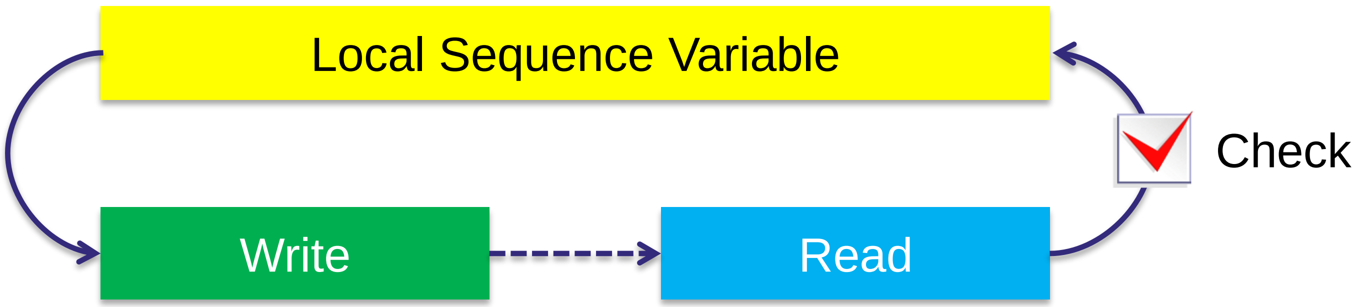
```
sequence apb_acc_s(addr, dir);
  @(apbi0.tr.start)
  (apbi0.tr.addr === addr)&&(apbi0.tr.dir === dir);
endsequence: apb_acc_s
```

- GPIO property

```
property chk_gpio_reg;
  logic [PWDATA_WIDTH-1:0] data;
  @(apbi0.tr.start)
  (apb_acc_s(`GPIO_ADDR, `APB_WR), data = apbi0.tr.data)##1
  |->
  @(posedge clk)
  (dut.gpio === data);
endproperty: chk_gpio_reg
```

Examples: Register Write-Read

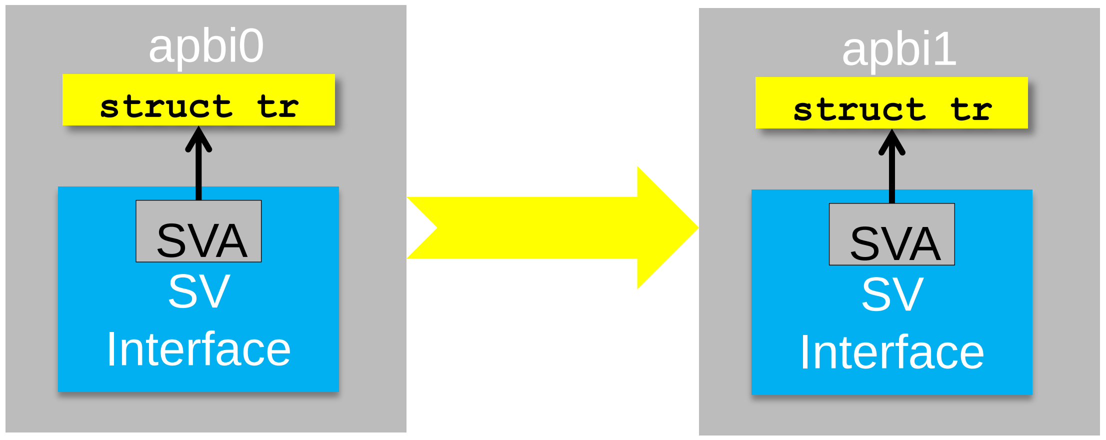
```
property chk_wr_rd(addr);
  logic [PWDATA_WIDTH-1:0] data;
  @(apbi0.tr.start)
  (apb_acc_s(addr, `APB_WR), data = apbi0.tr.data) ##[1:$]
   apb_acc_s(addr, `APB_RD)
  |->
  (apbi0.tr.data === data);
endproperty: chk_wr_rd
```



```
chk_wr_rd_reg0_a : assert property(chk_wr_rd(`REG_0_ADDR));
```

Examples: Scoreboarding

```
property chk_scbd;
  logic [PADDR_WIDTH-1:0]  addr;
  logic [PWDATA_WIDTH-1:0] data;
  logic dir;
  @(apbi0.tr.start)
  (1, addr = apbi0.tr.addr, data = apbi0.tr.data, dir = apbi0.tr.dir)
  ##1
  |->
  @(apbi1.tr.start)
  (apbi1.tr.addr === addr) &&
  (apbi1.tr.data === data) &&
  (apbi1.tr.dir  === dir);
endproperty: chk_scbd
```



Conclusion

- You can raise the abstraction level for SVA to TLM
- Split monitor concept can straddle TLM and SVA
- Backwardly compatible with standard UVM
- More flexibility to choose in which domain to place checks
- Some TL checks can be done succinctly with SVA
- Data – based checking also possible with SVA
- A complementary approach, not a UVM replacement

Thank You