2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

One Stop Solution of DFT Register Modelling in UVM

Rui Huang

Advanced Micro Devices

AMD

DFT Test Access

- DFT design is becoming more and more complex.
- Currently IEEE 1149.1, IEEE 1687 and 1500 protocols are used to integrate DFT IP into SoC.

- This approach makes the DFT test access network complex and it needs a series of complex shift operations to access a TDR (Test Data Register).

To Simplify It...

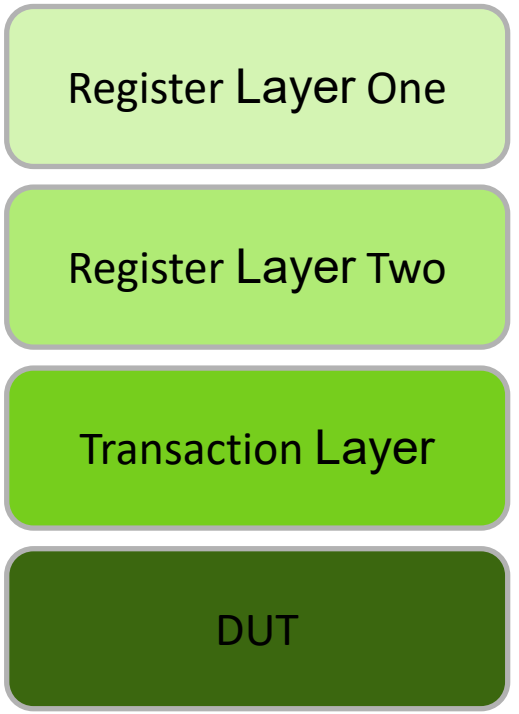
- Lift up DFT TDR access in RAL (Register Abstraction Level):
  - Test writers can focus on test sequences.
  - Tests can be easily migrated from block level to system level.

The Problem...

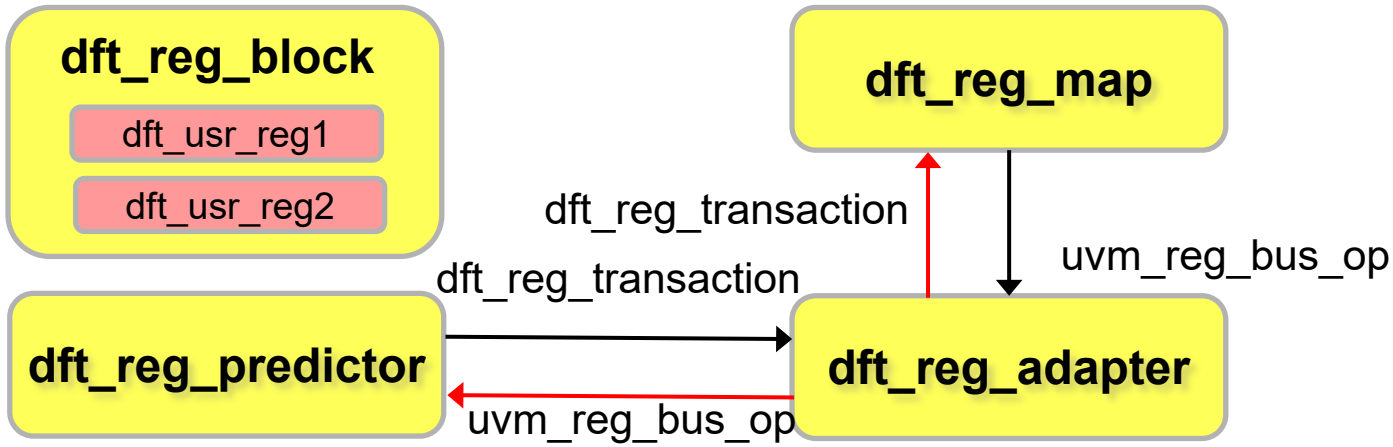
- How to model DFT TDR in RAL in a universal way so that it can be applied in different projects?

Universal Framework

- Layered structure of DFT TDR modelling in UVM:
  - Register layer one converts abstracted register operation to generic *dft_reg_transaction*.
  - Register layer two converts generic *dft_reg_transaction* to a series of *jtag_transactions*.
  - Transaction layer: *jtag_transactions* are used to drive and sample JTAG port of the DUT.



Register Layer One




```
class dft_reg_transaction extends uvm_sequence_item;
    bit [`DFT_REG_WIDTH-1:0]   address;
    bit                        wr_data_queue[$];
    int unsigned               reg_length;
                    ...
endclass: dft_reg_transaction
```

Register Layer One: DFT TDR Modelling

not uvm_reg

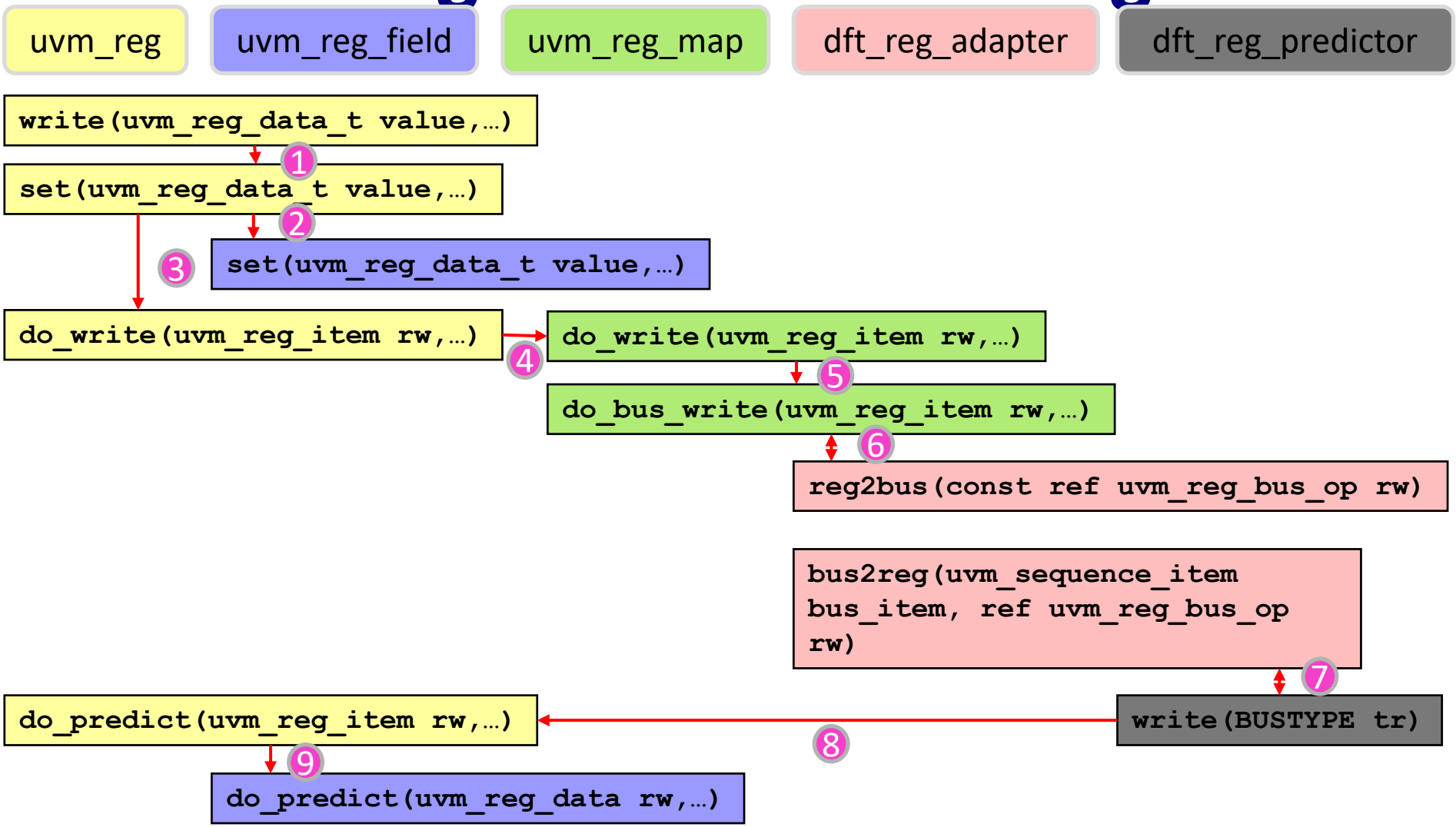
```
class ieee1149_gopphy_crsel_reg extends dft_reg;
   rand uvm_reg_field add_data;
   rand uvm_reg_field crsel_cmd;
   . . .
   virtual function void build();
      add_data = uvm_reg_field::type_id::create( "add_data" );
      add_data.configure( .parent                ( this ),
                          .size                  ( 8 ),
                          . . .
                          .is_rand               ( 1    ),
                          .individually_accessible( 0   ) );

endfunction: build
endclass: ieee1149_gopphy_crsel_reg
```

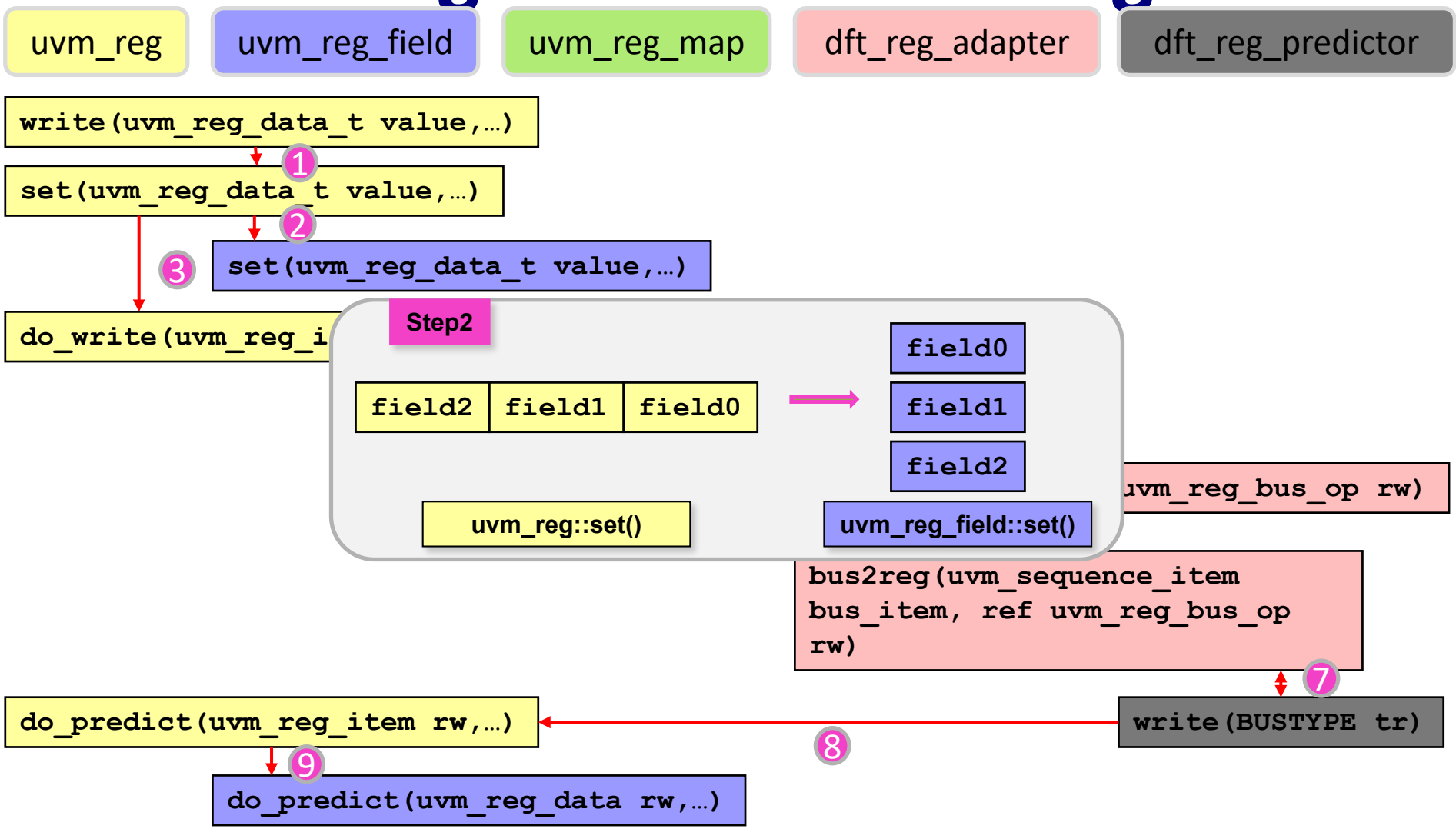
Register Layer One: How to Model Ultra-long Length TDR

- We override UVM_REG_DATA_WIDTH macro to the value identical to the length of the longest TDR in the system.
- Current UVM solution has storage waste issue:
  - The width of uvm_reg_data_t is decided by UVM_REG_DATA_WIDTH. And uvm_reg_data_t is instantiated and used almost everywhere in the RAL-related components.
  - Every field extends from uvm_reg_field also uses uvm_reg_data_t to store value. But normally the width of a filed is not as wide as thousands of bit.

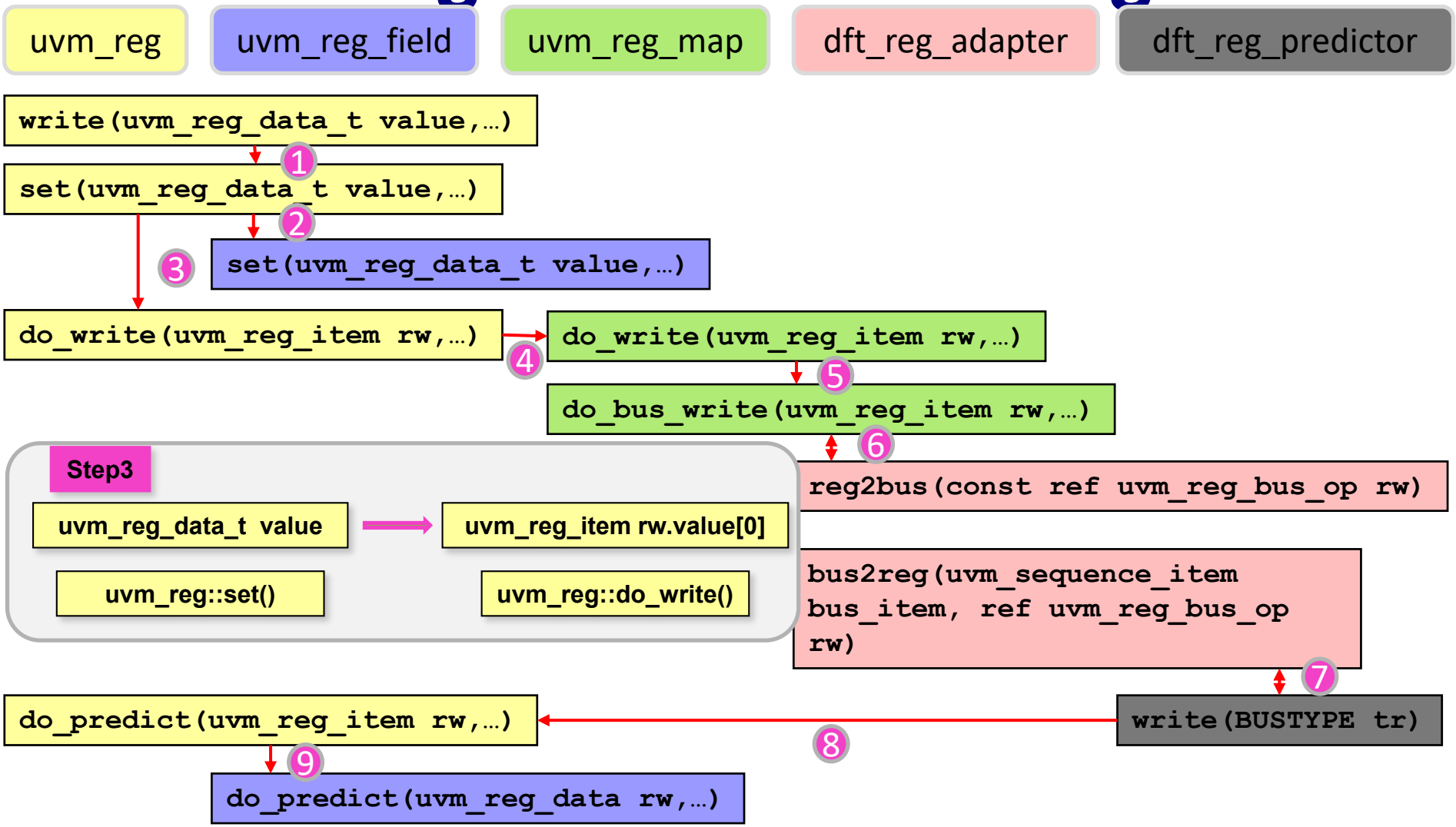
Solution for Ultra-long Length TDR Modelling



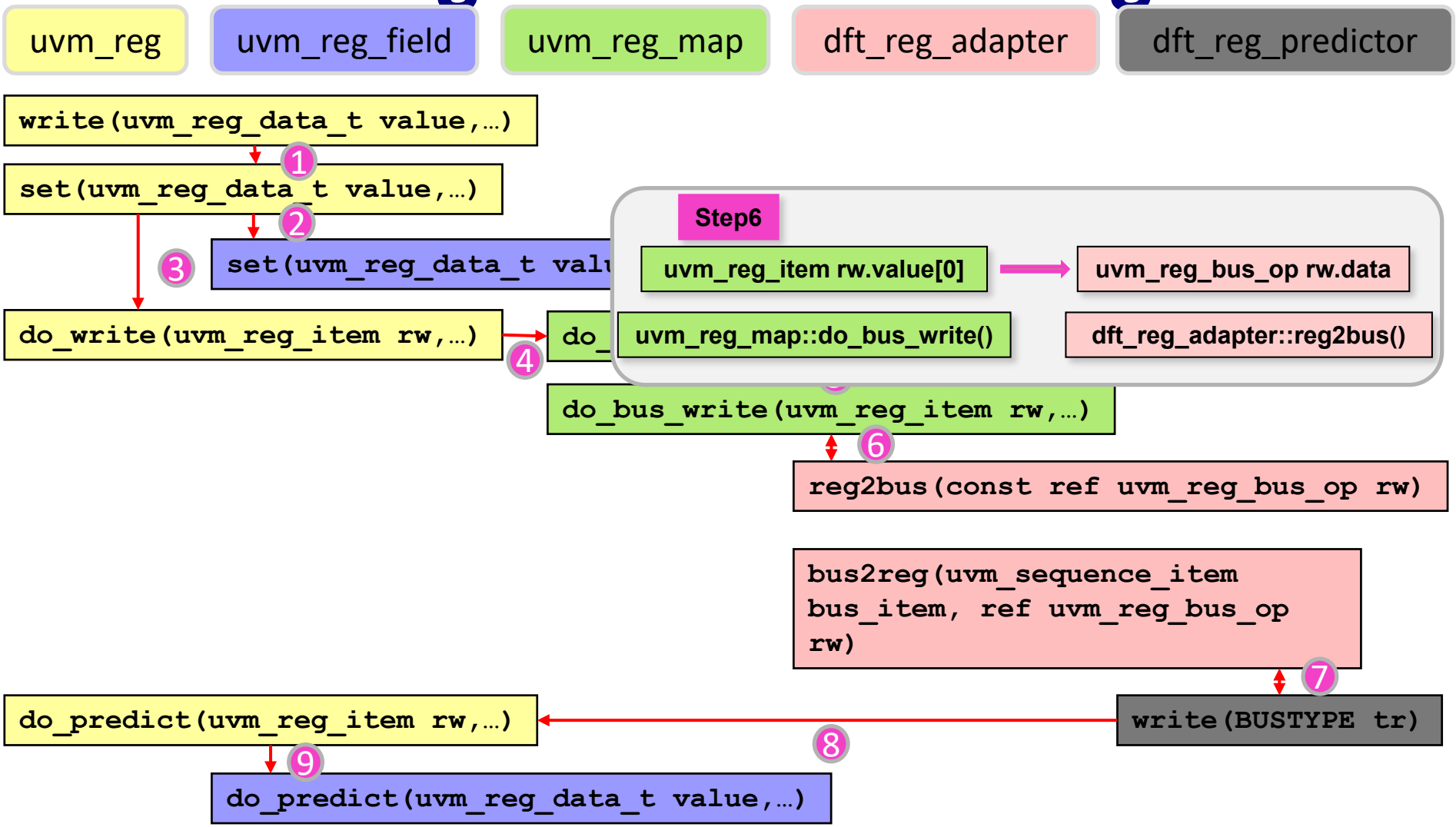
Solution for Ultra-long Length TDR Modelling



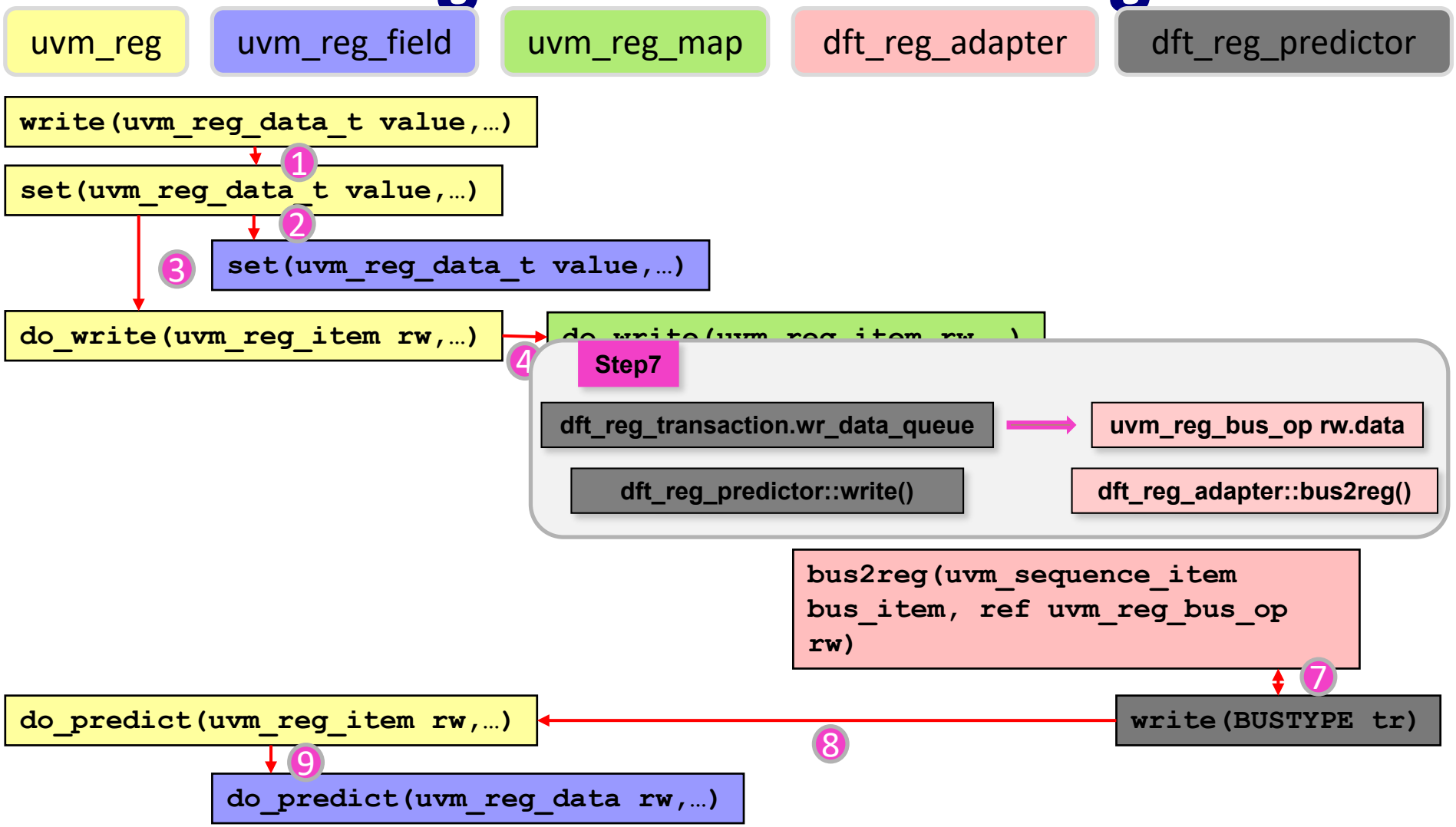
Solution for Ultra-long Length TDR Modelling



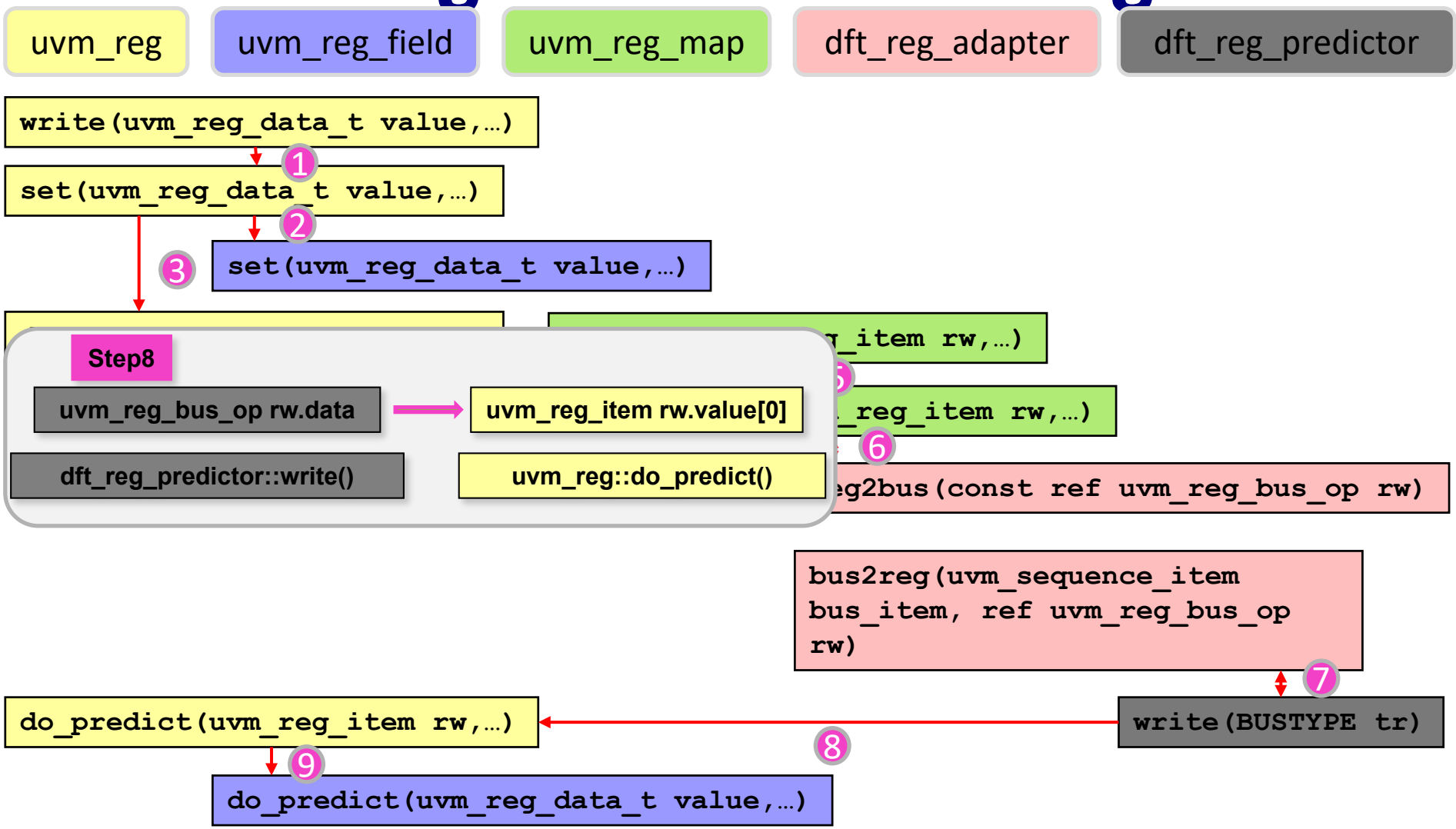
Solution for Ultra-long Length TDR Modelling



Solution for Ultra-long Length TDR Modelling

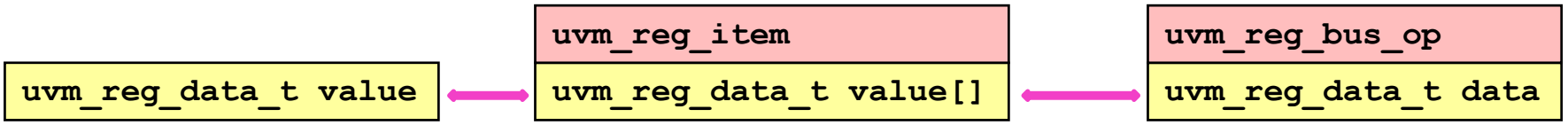


Solution for Ultra-long Length TDR Modelling

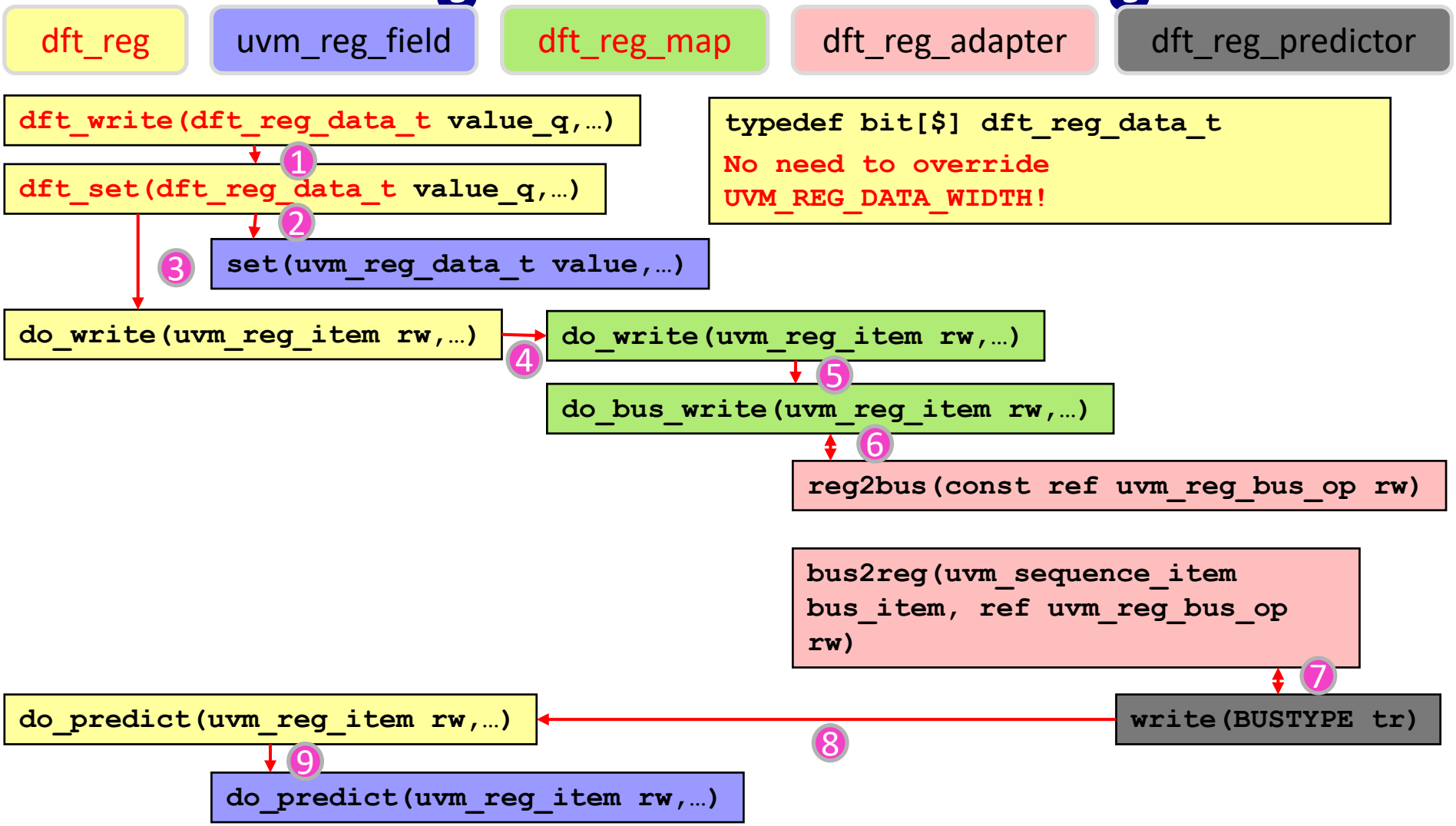


Solution for Ultra-long Length TDR Modelling

```
typedef bit unsigned [`UVM_REG_DATA_WIDTH-1:0] uvm_reg_data_t
```

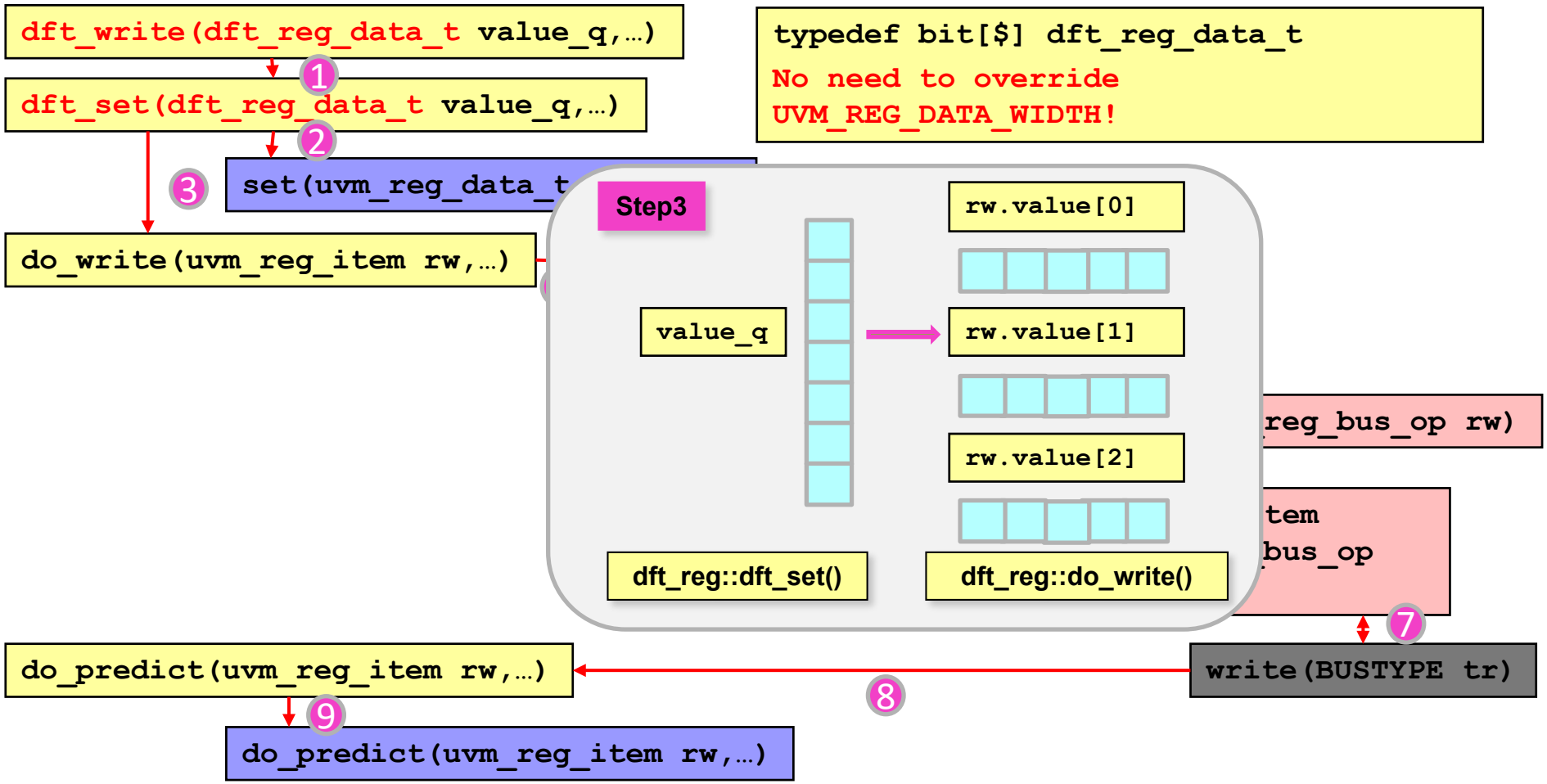


Solution for Ultra-long Length TDR Modelling

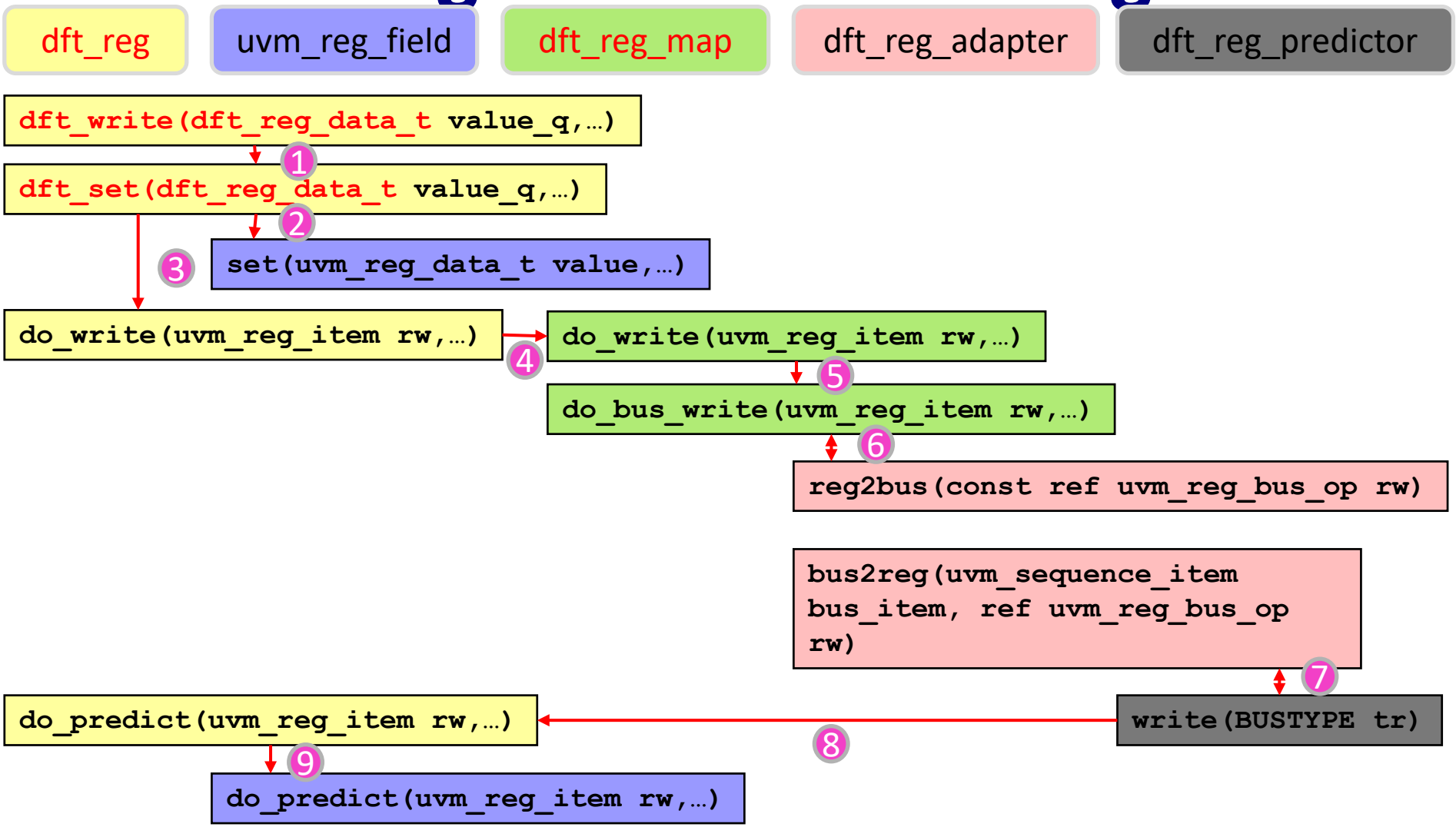


Solution for Ultra-long Length TDR Modelling

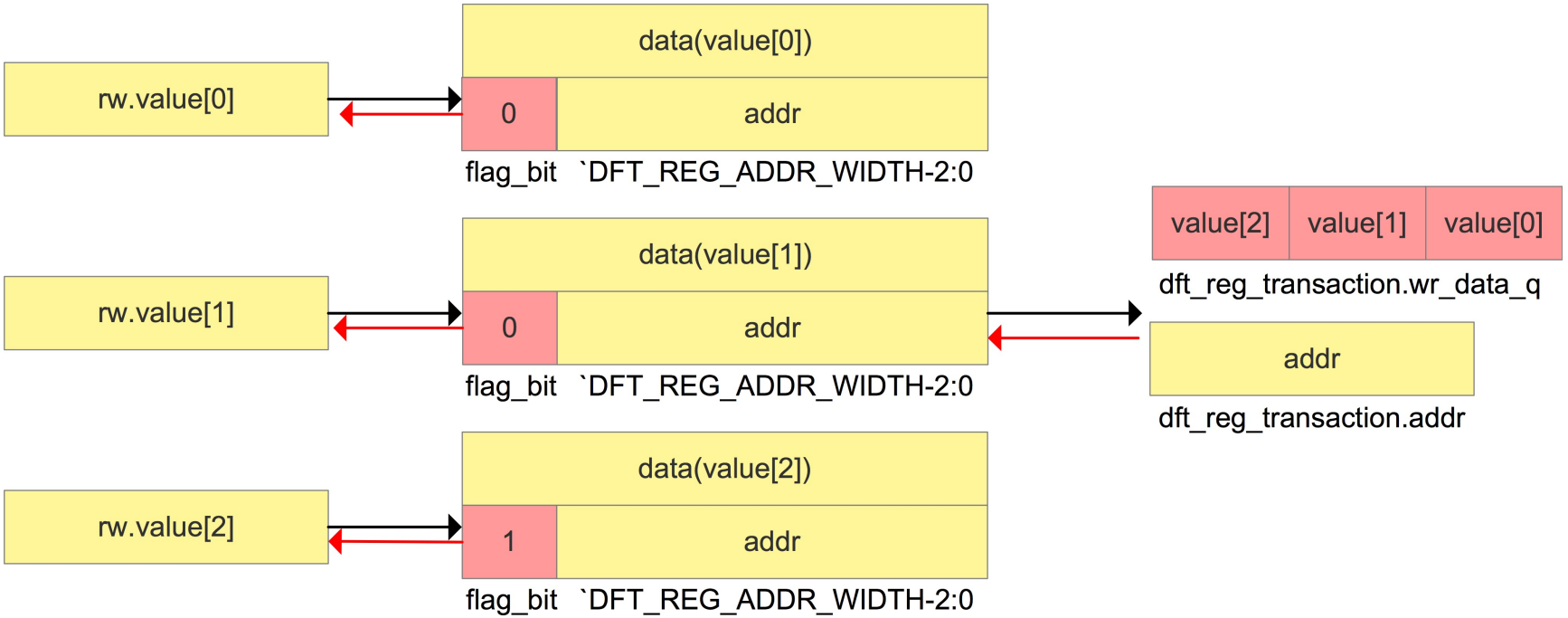
dft_reg | uvm_reg_field | dft_reg_map | dft_reg_adapter | dft_reg_predictor



Solution for Ultra-long Length TDR Modelling

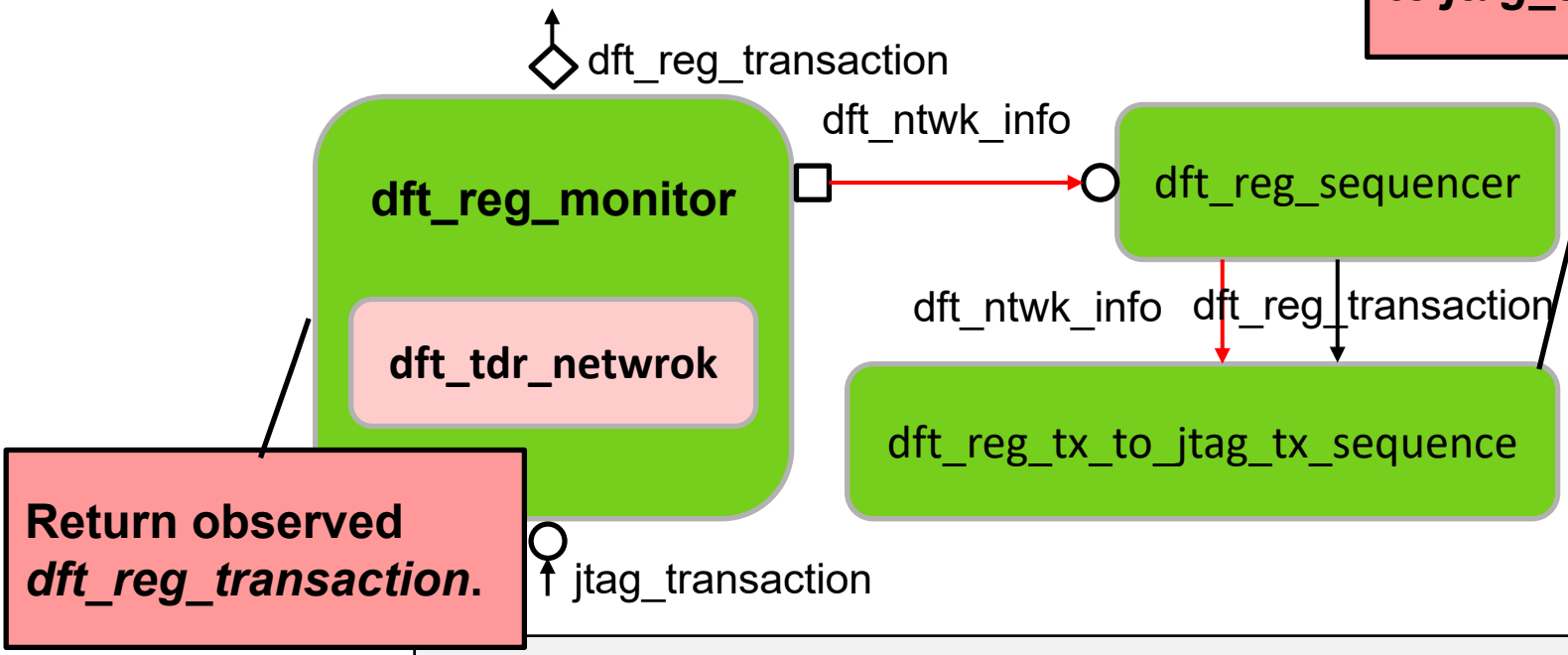


Register Layer One: Data Conversion Process



Register Layer Two

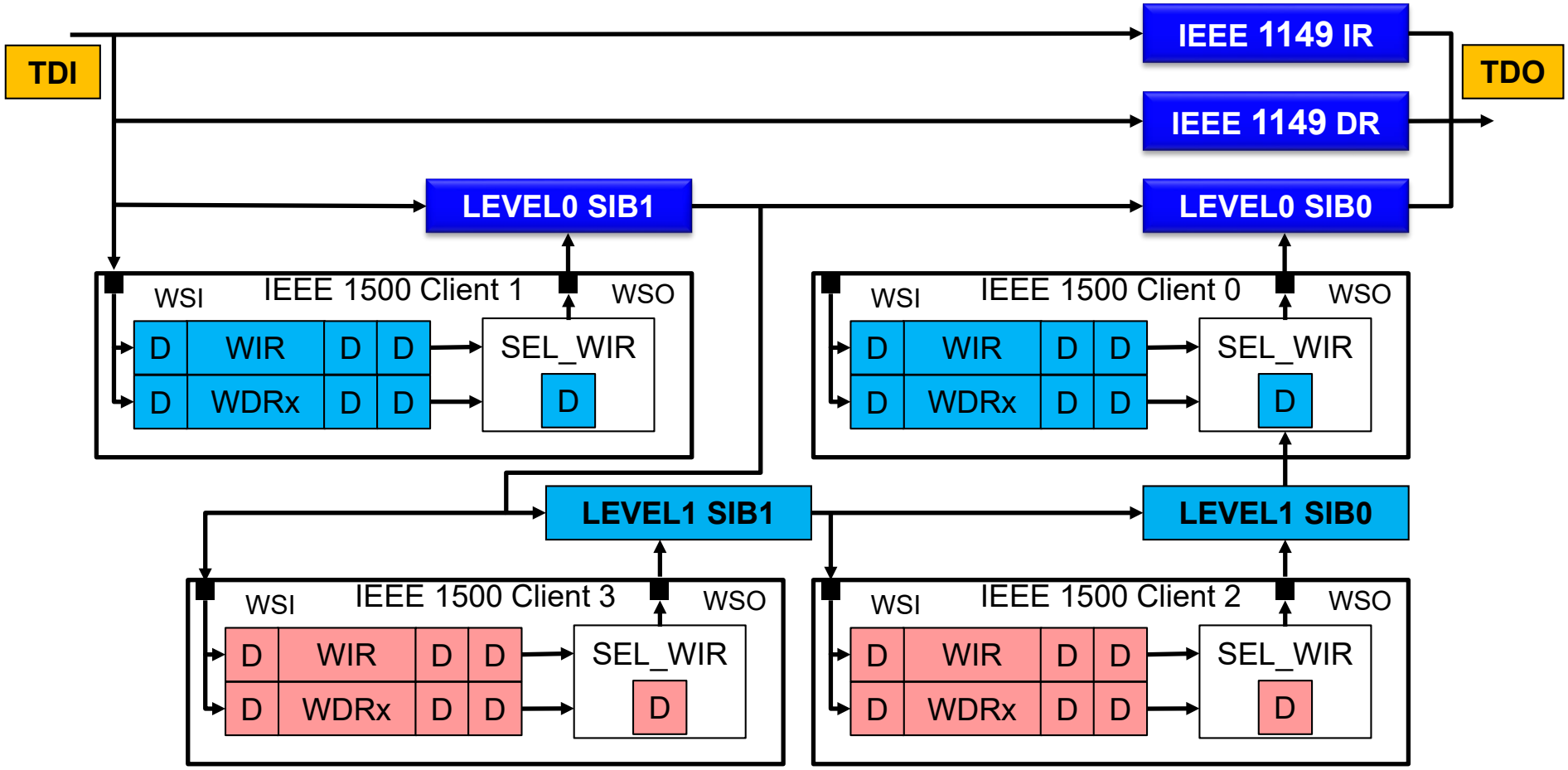
Converts generic *dft_reg_transaction* to *jtag_transactions*.



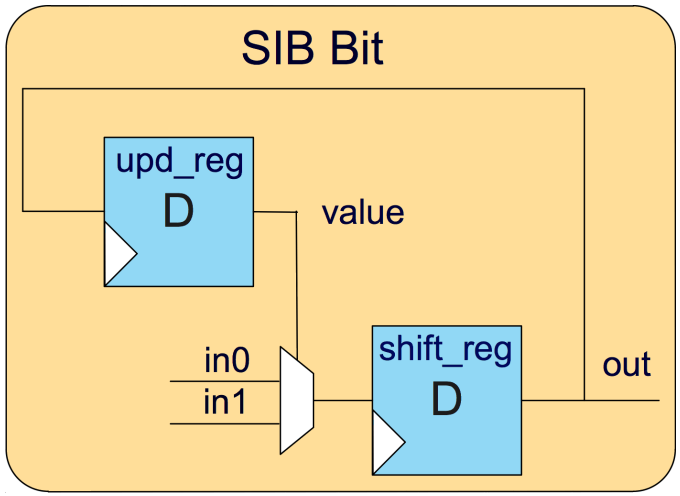
Return observed *dft_reg_transaction*.

```
class jtag_transaction extends uvm_sequence_item;
    bit                         o_ir[];
    bit                         o_dr[];
    bit                         tdo_dr_queue[$];
    bit                         tdo_ir_queue[$];
                    ...
endclass: jtag_transaction
```

Register Layer Two:
Example of DFT Test Access Network



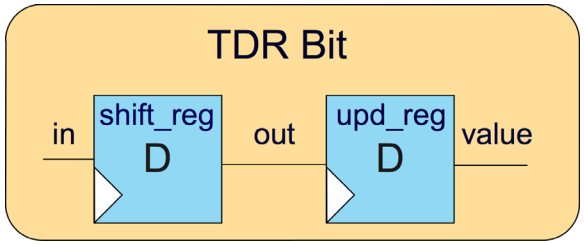
Register Layer Two: Access Network Element Modelling



The *out_update* function simulates the active clock edge on shift register and the *value_update* function simulates the active clock edge on update register.

```
class sib_node extends uvm_object;
   `uvm_object_utils(sib_node)
   bit     in0;
   bit     in1;
   bit     value;
   bit     out;
   function new(string name = "sib_node");
     super.new(name);
   endfunction : new
   function void out_update ();
      out = value ? in1 : in0;
   endfunction: out_update
   function void value_update ();
      value = out;
   endfunction: value_update
endclass : sib_node
```

Register Layer Two: Access Network Element Modelling




```
class reg_node extends uvm_object;
   `uvm_object_utils(reg_node)
   bit    in;
   bit    is_selwir;
   bit    value;
   bit    out;
   function new(string name = "reg_node");
     super.new(name);
   endfunction : new
   function void out_update ();
      out = in;
   endfunction: out_update
   function void value_update ();
      value = out;
   endfunction: value_update
endclass : reg_node
```

Register Layer Two: Access Network Modelling Example

- The functional equivalent access network can be constructed by instantiating following properties and a series of conditional statements.

```
sib_node   lvl0_sib0, lvl0_sib1, lvl1_sib0, lvl1_sib1;
reg_node   lvl0_sel_wir0, lvl0_sel_wir1;
reg_node   lvl1_sel_wir0, lvl1_sel_wir1;
reg_node   lvl0_wir[`IEEE1500_IR_WIDTH], lvl0_wdr_dynmc[];
reg_node   lvl1_wir[`IEEE1500_IR_WIDTH], lvl1_wdr_dynmc[];
```

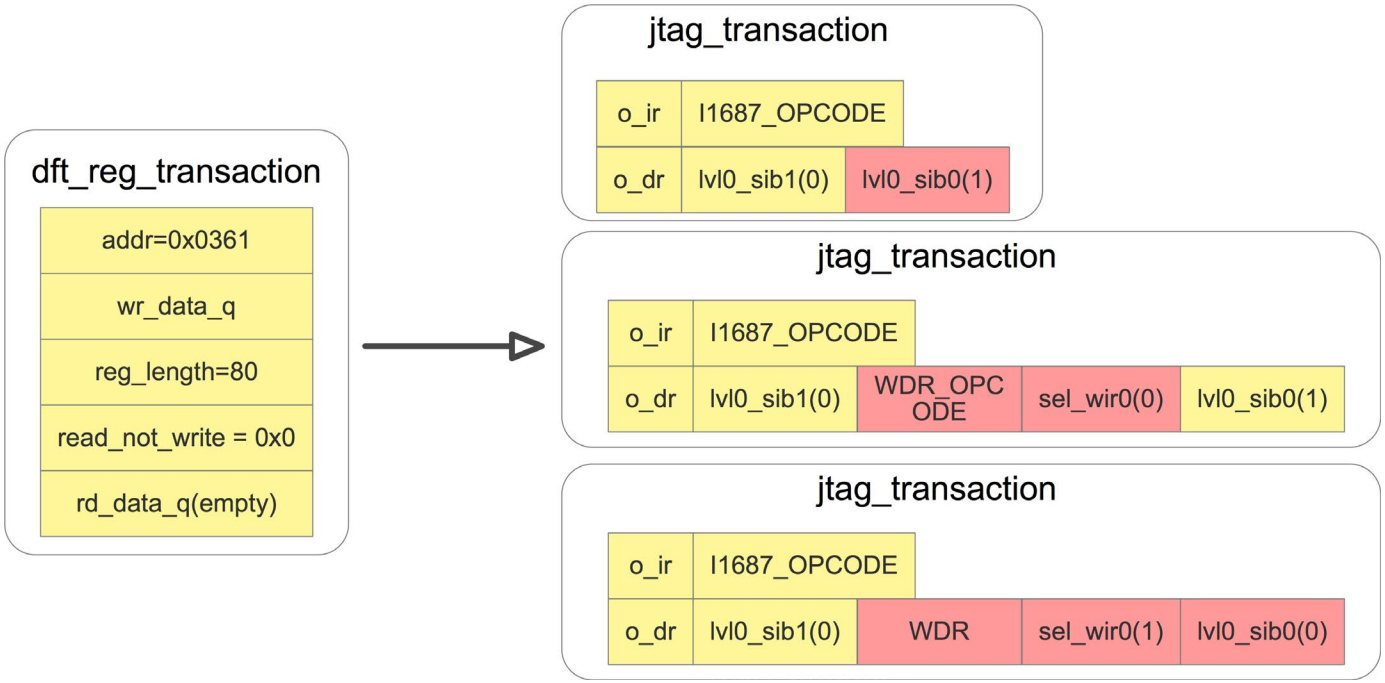
Register Layer Two: Access Network Modelling Example

```
if { ir == `P1687 &&
     {lvl1_sib1.value,lvl1_sib0.value} == 2'b00 &&
     {lvl0_sib1.value,lvl0_sib0.value} == 2'b01 &&
      lvl0_sel_wir0.vallue == 1}
the client0's WIR is being select.
if { ir == `P1687 &&
     {lvl1_sib1.value,lvl1_sib0.value} == 2'b00 &&
     {lvl0_sib1.value,lvl0_sib0.value} == 2'b01 &&
      lvl0_sel_wir0.vallue == 0}
the client0's WDR is being select.

The length of the WDR can be calculated by the observed
jtag_transaction.tdo_dr_queue length and the network structure.
```

Register Layer Two:

dft_reg_tx_to_jtag_tx conversion sequence



The dft_reg_tx_to_jtag_tx sequence decodes the dft_reg_transaction.addr to get the TDR's location in the test access network.

Results

A TDR is defined to have X bits width and each field is one bit width. Instance 200 such TDRs and write these 200 TDRs in sequence.

	512	1024	2048	4096
Current UVM RAL Solution	2350 MB	2760 MB	4213 MB	9264 MB
Memory Saving Solution	2075 MB	2345 MB	2981 MB	4327 MB
Memory Saving	11.7%	15.0%	29.2%	53.3%

Results

A TDR is defined to have X bits and each field is one bit. Instance 400 such TDRs and write these 400 TDRs in sequence.

	512	1024	2048	4096
Current UVM RAL Solution	2504 MB	3542 MB	6452 MB	16064 MB
Memory Saving Solution	2335 MB	2960 MB	4067 MB	6421 MB
Memory Saving	6.8%	16.4%	37.0%	60.0%

Summary

- We defined a universal layered framework to model DFT TDR in UVM.
  - Register Layer One and Transaction Layer can be used in different projects.
  - In Register Layer Two, dft_reg_network component and dft_reg_tx_to_jtag_tx_sequence can be easily adjusted according to specific test access network architecture.
- We resolved the issue of simulation memory waste in modelling ultra-long length TDR with current UVM solution (up to 60% memory saving).

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!