2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Deploying Customized Solution for Graphics Registers with UVM1.2 RAL

Roman Wang

Nigel Wang, Lunping Guo, Robert Liu, Jia Zhu

Advanced Micro Devices, Inc. Shanghai, China

AMD

Introduction

The Special Requirements in Graphics Registers.

- A: Many registers need general and sideband information to make register as multiple scopes.
  - Contain a one-dimensional or multi-dimensional register array with the parameterized depth
  - Each entry can be accessed through
    - The same address
    - Decode with different sideband information, like groups of ID fields.
- B: Many registers are highly depending on. its field or other register.
  - For example: Write disable field.

The Challenges

- RDL can only generate the RAL models in general behavior.
  How can we handle the special without touching RAL models?
  - Functional coverage/ Automate register test/ Easy to debug
- How to make UVM RAL adapter work for the special?
  - Different bus interface has different sideband information.
- RAL provides several access methods.
  - How to access the registers with sideband information?
  - Is that possible to achieve the requirements below?
    - Hide the UVM RAL knowledge as much as possible.
    - Unify the RAL access API to support both general and special registers.

Solutions

- The explicit approach for **challenge A** (our focus)
  - The template method pattern
  - The factory pattern
  - The singleton pattern
- The implicit approach for **challenge B**
  - Creating special register callback library

Mark Litterick, Marcus Harnisch, Verilab, “Advanced UVM Register Modelling – There’s More Than One Way to Skin A Reg”. DVCon USA 2014.

Explicit Approach - 1

- The template method pattern.
  - Customized and Parameterized RAL Template
    - Extending register model with a register array indexed by sideband information, like ID fields.
    - Re-constructed methods for extended register operation.
    - Defines extension functions for user to customize the decoding.

```
class cust_ral_tmpl_base #(type R=uvm_object, type T=uvm_reg, int SIZE=1) extends  T;
    T T_inst[SIZE];    // register_contents
function new(string name = "cust_ral_tmpl_base");
   super.new(name);
   foreach ( T_inst[i] ) begin
      string name = $sformatf("%s_%0d", T::get_type_name(), i);
      T_inst[i] = new(name);
   end
```

R: User extension object Type
T: Original register type
SIZE: The depth of the array

Trick:
We do NOT use the factory method by calling type_id::create()

Explicit Approach - 1

- Example of user template extension

```
class BLK1_RAL_TMPL #(type T=uvm_reg, int SIZE=8) extends cust_ral_tmpl_base #(bus_tr_t,T,SIZE);

// implement the user extension functions
function void write_extension(….);
```

transaction type (sideband information fields, like IDs)

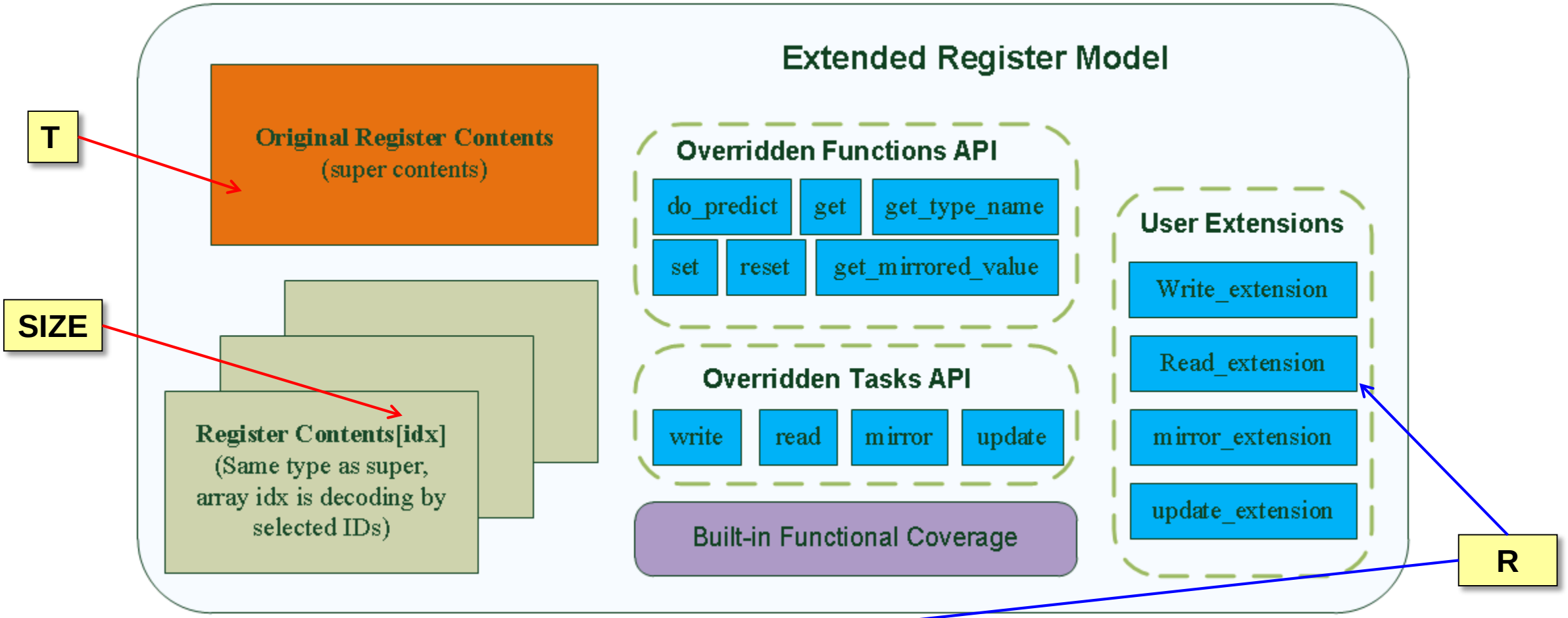
- UVM factory pattern.
  - Override the register type in the build_phase of base test layer.

```
BLK1_REG0::type_id::set_type_override(BLK1_RAL_TMPL#(BLK1_REG0,12)::get_type());
```

T

Size

Big Picture of RAL Template



virtual function void **write_extension**(input **R** extension,output int reg_idx,output int reg_idx_all);

Customized RAL Adapter

In the uvm_reg.svh

```
virtual task write(, input  uvm_object extension = null,)
```

In the uvm_reg_map.svh

```
task uvm_reg_map::do_bus_write(uvm_reg_item rw,
  ……
  adapter.m_set_item(rw);
  bus_req = adapter.reg2bus(rw_access);

  // execute bus item on the interface UVC
```

Set the reg_item in adapter

Get the reg_item in adapter

Get the user extension object

Reshaping the bus item

In the user customized RAL adapter

```
class blk_ral_adapter extends uvm_reg_adapter;
   function uvm_sequence_item reg2bus(const ref
uvm_reg_bus_op rw);
        bus_trans_item b_tr;            // bus item
        user_ext_item user_ext_tr;    // user extension
        uvm_reg_item temp_tr;
        b_tr = new("b_tr");
        user_ext_tr = new(“user_ext_tr”);
        temp_tr = get_item();
        if($cast(user_ext_tr, temp_tr.extension) &&
temp_tr.extension != null)
            b_tr.id1 = user_ext_tr.u_id1; … // user space
         else
            b_tr.addr = rw.addr;      // general assignment
```

Overridden Function Examples

- The set() and get()

```
function void set (uvm_reg_data_t  value,
                   string fname = "",  int lineno = 0);
    int reg_idx = 0;  int reg_idx_all = 0;
    if(lineno == SIZE) d
      reg_idx_all = 1;
    else
      reg_idx = lineno;
    if (reg_idx_all == 1 )
       foreach (T_inst[i]) begin
          T_inst[i].set(value[31:0], fname, lineno);
   else
         T_inst[reg_idx].set(value[31:0], fname, lineno);
```

The index of Reg array

```
function uvm_reg_data_t get (string  fname = "",
                             int  lineno = 0);
    uvm_reg_data_t reg_val;
    if(lineno < 0 || lineno >= SIZE) begin
        `uvm_error(get_type_name(), $psprintf("lineno
shoude be inside [0:%d], assign lineno to 0 by default",
SIZE-1))
        lineno = 0;
    end
    if(fname == "super")
        reg_val= super.get();
    else
        reg_val= T_inst[lineno].get();
```

Overridden Task Example

- The write()

```
virtual task write( output uvm_status_e status, … )
   uvm_reg_item rw_super;
   R trans;
   if(!$cast(trans, extension) || extension == null)
     trans = R::type_id::create("trans");
     write_extension(trans, reg_idx, reg_idx_all);
     if (!reg_idx_all ) T_inst[reg_idx].set(value[31:0]);
     else ….
     XatomicX(1);
     if(reg_idx == 0)
        set(value);
```

```
 rw_super =
uvm_reg_item::type_id::create("write_item",,get_full_na
me());
rw_super.element = this;
rw_super.extension = trans;
super.do_write(rw_super);
…
XatomicX(0);
```

Trick:
We do not directly call **super.write(),** because the super has the set(value) which will call the overridden set() function and update all indexes to entry0.

Overridden Task Example

- The mirror()

```
task mirror (output uvm_status_e status, ….)
      R trans ; int reg_idx = 0;
      uvm_reg_map    _map;
      uvm_reg_map_info _map_info;
      …..
      mirror_extension(trans,reg_idx);
      _map = super.get_local_map(map, "read()");
      if(_map == null) begin
        status = UVM_NOT_OK;
        return;
      end
      _map_info = _map.get_reg_map_info(super);
```

```
if (_map.get_reg_map_info(T_inst[reg_idx],0) == null)   begin
      _map.add_reg(T_inst[reg_idx], _map_info.offset,
_map_info.rights , _map_info.unmapped, _map_info.frontdoor
);
      _map.Xinit_address_mapX();
 end
 T_inst[reg_idx].mirror(status, check, path, map, parent, prior,
extension, fname, lineno);
```

Trick:
We refer to the super.mirror to handle the map and map_info for T_inst[idx].
It is necessary to call the map.Xinit_address_mapX which is to resolve the map when the block is locked.

UVM RAL Access Export

- *Prerequisites*

✓ The predefined base hook: translate sideband information into user extension object.

✓ Enhanced uvm_vpl_* message macros for a better debugging.

✓ RAL access atomic SV class.

- RAL Access Export (**RAE**) acts as a register access proxy.

✓ The singleton top instance (General SV class)

✓ Unique register front door access API with multiple operation modes.

✓ Support both general and special registers.

✓ Model Track Interface (MTI): dumping register access into a file.

✓ Optional atomic protect for register access API

User Defined Hook and RAL Export

```
// Front door access hook base class
virtual class ral_fd_hook_base;
    pure virtual function void fd_ral_access_extension( input bit [`MAX_ID1_W-1:0] id1 = 0, input bit [`MAX_ID2_W-1:0] id2 = 0, ….. , output uvm_object ext_extension);
    pure virtual function void vector2extension( input bit [`MAX_VECTOR_W-1:0] vector = 0,
output uvm_object ext_extension);
```

```
class ral_export;
    static local ral_export m_inst;
    ral_fd_hook_base fd_cb; …
    static function ral_export get(); …
    task automatic fd_ral_access( ….. );
endclass: ral_export

const ral_export ral_exp = ral_export::get();
```

```
// User extension
class blk1_fd_hk extends ral_fd_hook_base;
```

```
// At the end of build_phase of base test layer.
ral_exp.fd_cb = blk1_fd_hk::get();
```

Supported Front-door access Command.

- WR, WR_COMP, RD, MIRROR_W_CHK, SET_UPDATE, RAND_UPDATE, RD_M_WR, POLLING

Unique Front-door RAL Access API

```
task automatic fd_ral_access ( `uvm_ext_fline_decl
                ral_fd_cmd_t reg_access_type,
                uvm_reg rg,
                ref logic [`MAX_REG_DATA_W-1:0] rwdata,
                 input bit special = 0,
                 bit [`MAX_ID1_W-1:0] id1 = 0,
                 ……
                 int poll_num = 2,
                real poll_interval = 10ns,
                uvm_sequence_base parent = null,
                uvm_reg_map map = null,
                real timeout = 5us);
……..
if(special && (fd_cb != null) )
    fd_cb.fd_ral_access_extension (id1,…, ext_obj);
```

```
case (reg_access_type)
      WR:begin
           fork: block_WR
               begin // write access
                   job[0] = process::self();
                     rg.write(status,…,.extension(ext_obj));
               end
               begin   //Timeout
                    job[1] = process::self();
                   delayer (timeout);.
               end
           join_any
           foreach (job[j])
                if ( job[j].status != process::FINISHED )
                   job[j].kill();  …..
```

RAL Export Examples

logic [`MAX_REG_DATA_W-1:0] value

1. General register READ

ral_exp.fd_ral_access(`uvm_ext_fline_map, RD, state_regs,value);

2. Special register READ

ral_exp.fd_ral_access(`uvm_ext_fline_map,RD, cool_rd_reg, value, .**special(1), .**id1(2'b10), .id2(7'h5));

3. Special register POLL

reg_data=64'hxxxx_xxxx;

reg_data[15]=1'b1; // polling till bit15 is 1.

ral_exp.fd_ral_access(`uvm_ext_fline_map, POLL, cool_state_reg, value, .**special(1), .**id1(2'b10), .id2(7'h6), .poll_num(10), .poll_interval(1us)); // total polling 10 times and 1us interval between two polling operation.

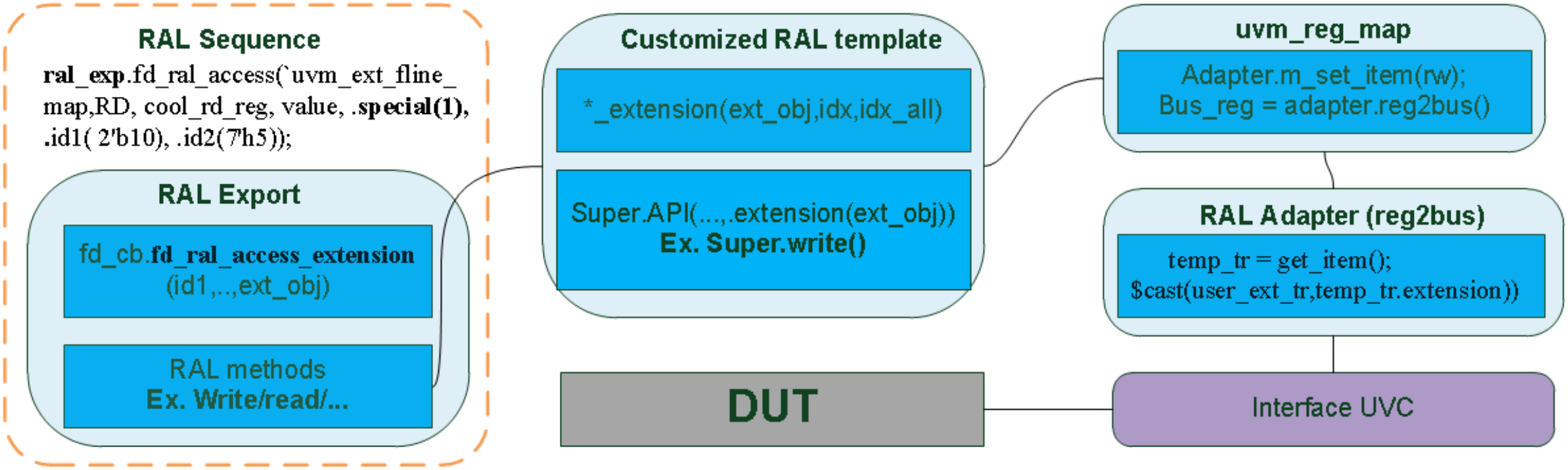
```
// Example of enhanced UVM message Marco.
`define uvm_ext_info(ID, MSG, VERBOSITY,FILE,LINE) \
begin \
if(uvm_report_enabled(VERBOSITY,UVM_INFO,ID)) \
uvm_report_info (ID, MSG, VERBOSITY,FILE,LINE, "", 1); \
end
`define uvm_ext_fline_map \
        `uvm_file \
        ,`uvm_line
`define uvm_ext_fline_decl \
        string filename, \
        int line,
```

Automate Register Test

- UVM RAL built-in sequences can’t work for the special registers based on a customized template.

```
// Example of customized uvm_reg_hw_reset_seq.
ral_fd_cb_base fd_cb;
uvm_object ext_obj;
uvm_pool#(string, id_vector_t) spec_reg_pool;
……
    protected virtual task do_block(uvm_reg_block blk);
      ……
      foreach (regs[i]) begin
        if(spec_reg_pool.exists(regs[i].get_name()))
        fd_cb.vector2extension(spec_reg_pool.get(regs[i].get_name()),ext_obj)
        regs[i].mirror(status, UVM_CHECK, UVM_FRONTDOOR, maps[d], this, .extension(ext_obj));
  ….
```

The Workflow from RAL to interface UVC



2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Conclusions

- Successfully deployed this total solution in block level UVM verification environments for graphics registers testing.
- The special register can easily model with extending RAL base template.
  - Support the special register built-in functional coverage
  - Support automate special registers testing.
- Unify the UVM RAL access API
  - Hide the UVM RAL knowledge and no gap between designer and verifier.
  - Easy to understand and reference each other.
- Improve the verification productivity.

roman.wang@amd.com