Customization of RAL Adapters and Predictors in UVM 1.2

Steven K. Sherman

AMD

September 24, 2015

Boston SNUG

Agenda

Introduction

Introduction

- This presentation …
  - Focus is on issues and concepts, not details (see paper and refs)
  - Fixes some typos in the paper
  - Developed with 1.1 and 1.2, not all 1.2 features presented
- UVM 1.2 RAL includes:
  - uvm_adapter.svh
  - uvm_predictor.svh
- "UVM 1.1 User's Guide"
  - Basic descriptions, little about customization
- General customization guidelines available
  - See Doug Smith's, "Easier RAL" SNUG paper!
  - See my last SNUG paper for a few customizations!

Why customize?

Why customize?

- Adapter and Predictor are MEANT to be customized!
  - Extend classes!
  - Override virtual functions/tasks!
  - Exercise options!
  - (Ab)use the extension object!

Thou shalt keep thy mirrored register values IN SYNC with the DUT!

Why customize?

I'll just use the default predictor and adapter!

Fine, unless you …

- Don't control everything
- Import IP
- Have specific design, security, address map or other requirements

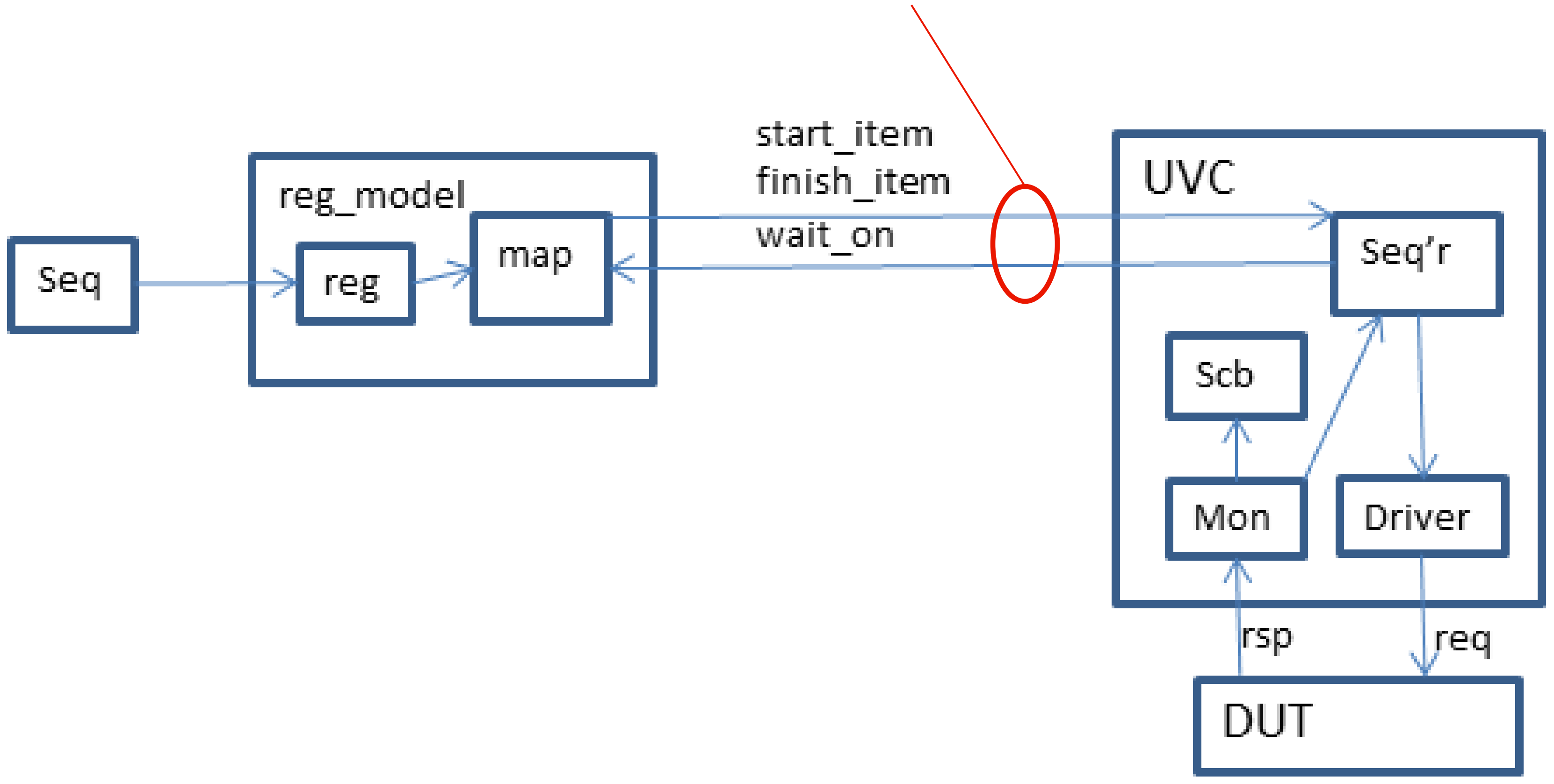
Quick Overview

RAL Adapters and Predictors

Quick Overview

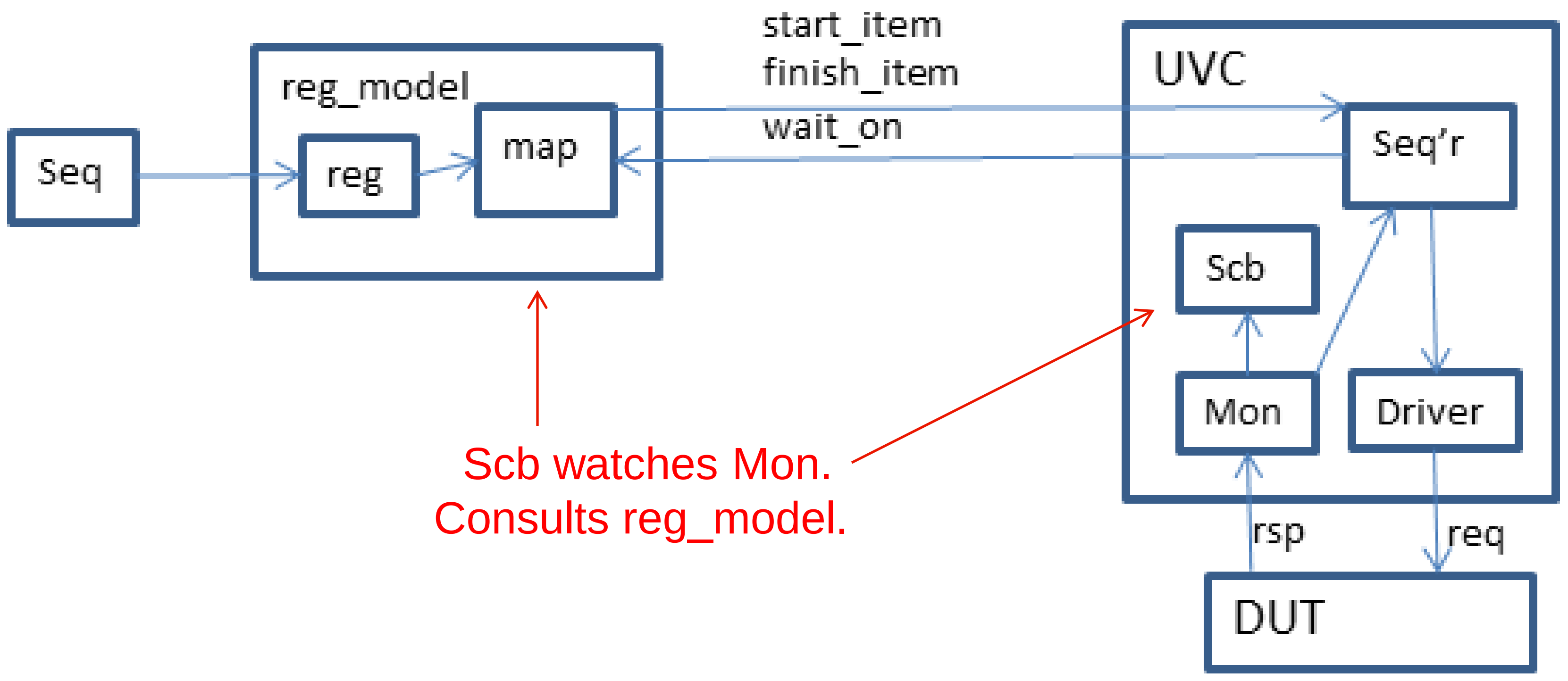
Basic System

Packets in default format.



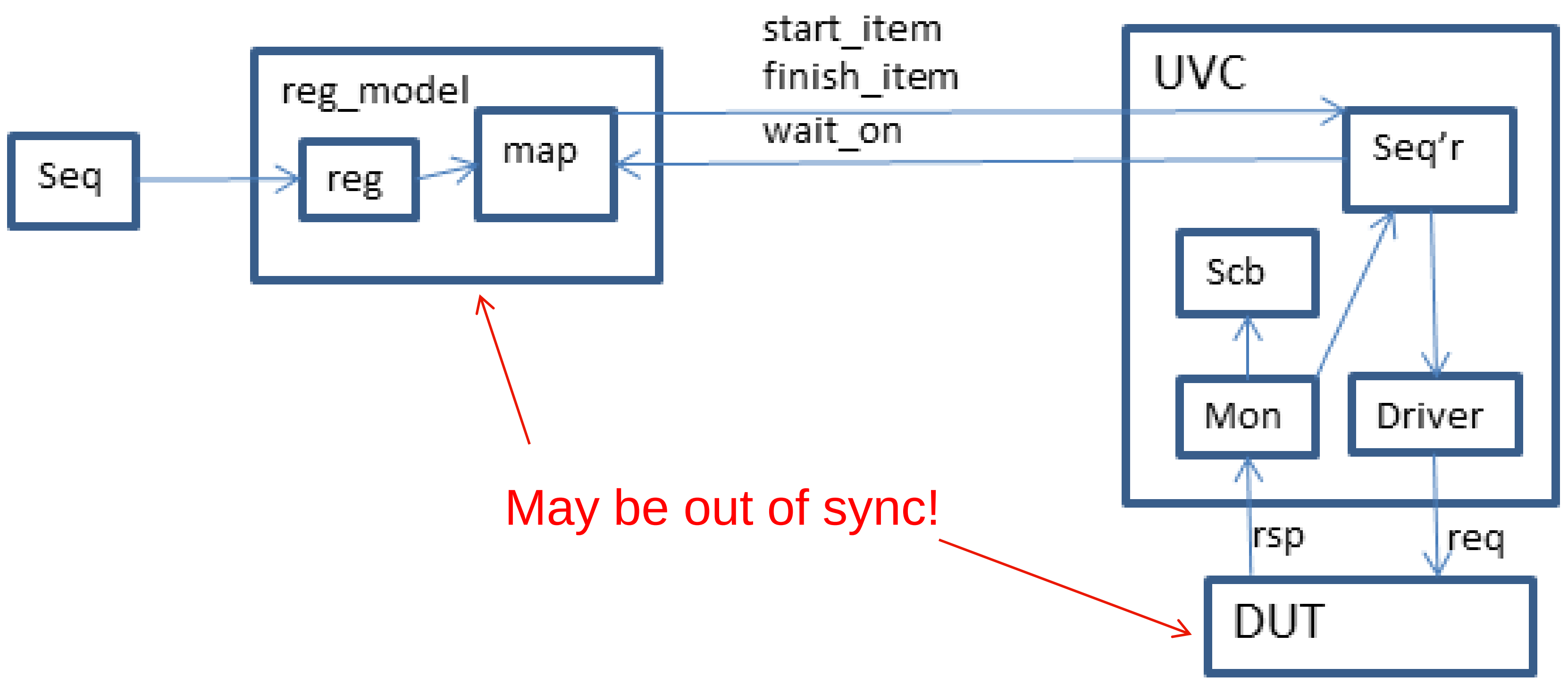
Quick Overview

Basic System



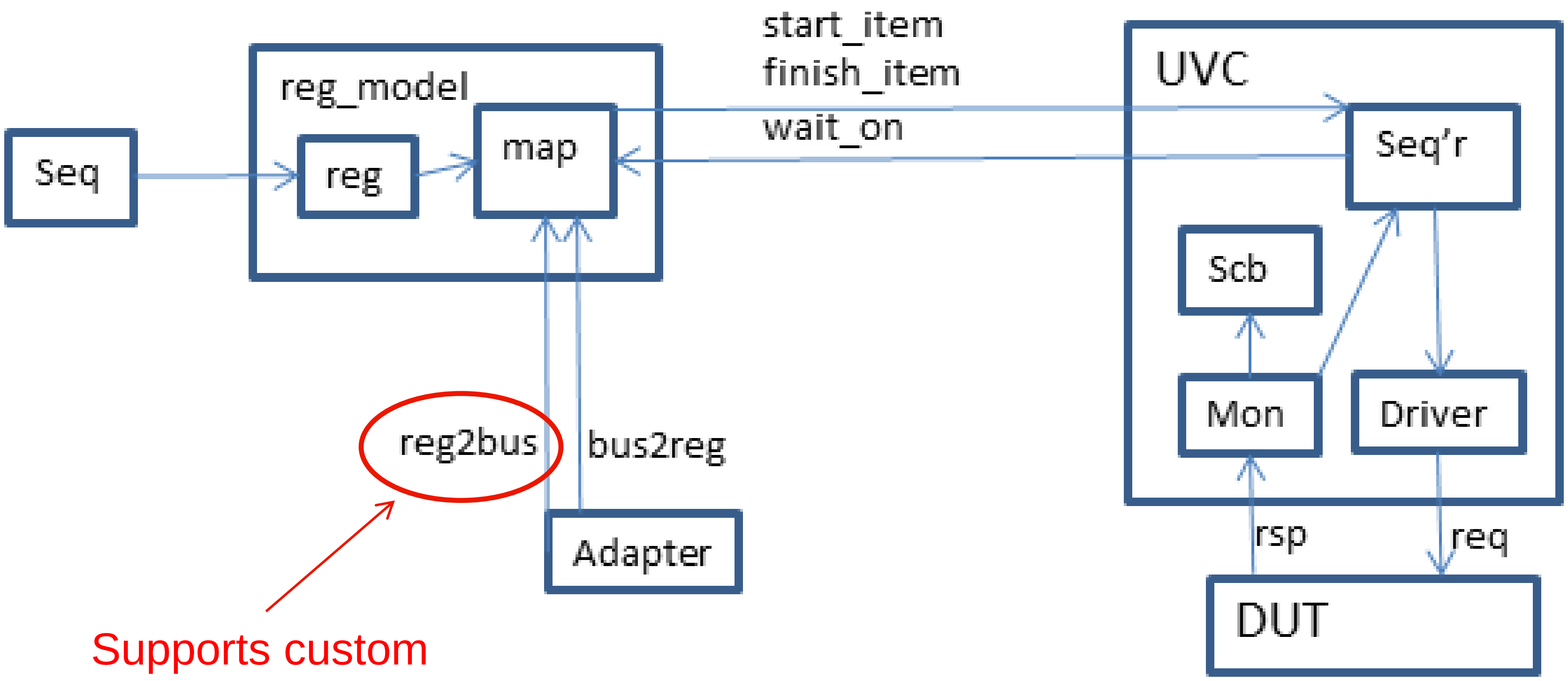
Scb watches Mon.
Consults reg_model.

Quick Overview
Basic System



Quick Overview

System with Adapter



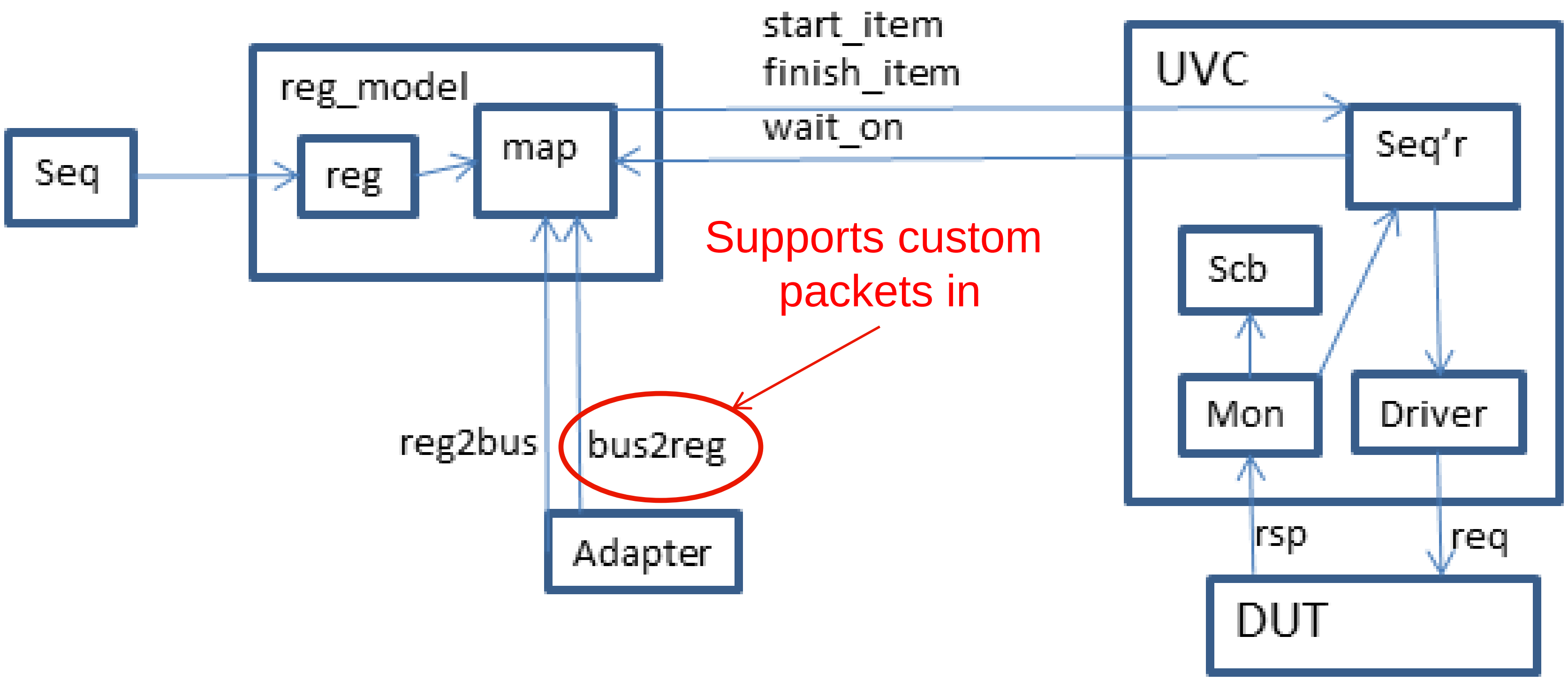
Supports custom packets out

Quick Overview
Adapter reg2bus

```
// Function: reg2bus
//
// Converts a <uvm_reg_bus_op> struct to a <uvm_tlm_gp> item.

virtual function uvm_sequence_item reg2bus(const ref uvm_reg_bus_op rw);

   uvm_tlm_gp gp = uvm_tlm_gp::type_id::create("tlm_gp",, this.get_full_name());
   int nbytes = (rw.n_bits-1)/8+1;
   uvm_reg_addr_t addr=rw.addr;

   if (rw.kind == UVM_WRITE)
      gp.set_command(UVM_TLM_WRITE_COMMAND);
   else
      gp.set_command(UVM_TLM_READ_COMMAND);

   gp.set_address(addr);

   gp.m_byte_enable = new [nbytes];
   gp.m_byte_enable_length = nbytes;

   gp.set_streaming_width (nbytes);

   gp.m_data = new [gp.get_streaming_width()];
   gp.m_length = nbytes;

   for (int i = 0; i < nbytes; i++) begin
      gp.m_data[i] = rw.data[i*8+:8];
      gp.m_byte_enable[i] = (i > nbytes) ? 8'h00 : (rw.byte_en[i] ? 8'hFF : 8'h00);
   end

   return gp;

endfunction
```

Quick Overview

System with Adapter

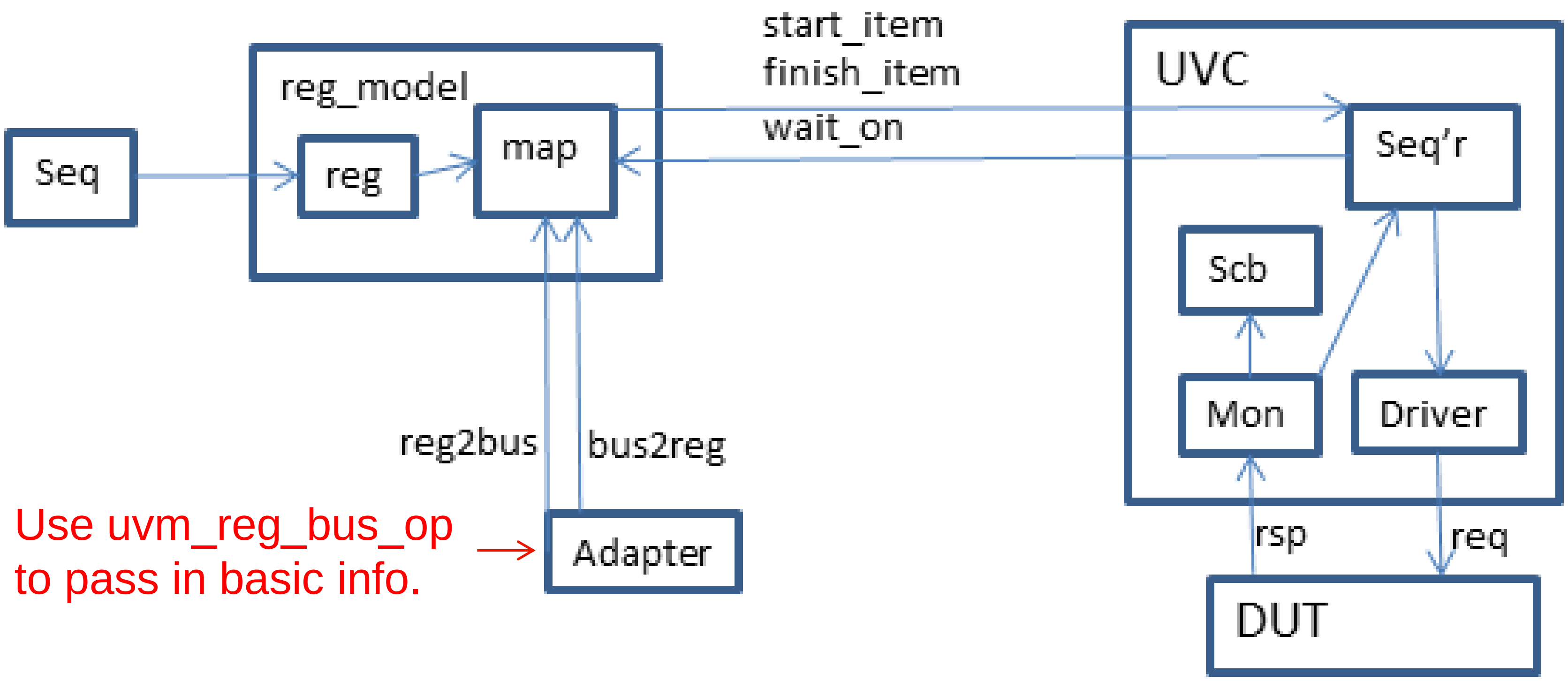


Quick Overview

Adapter bus2reg

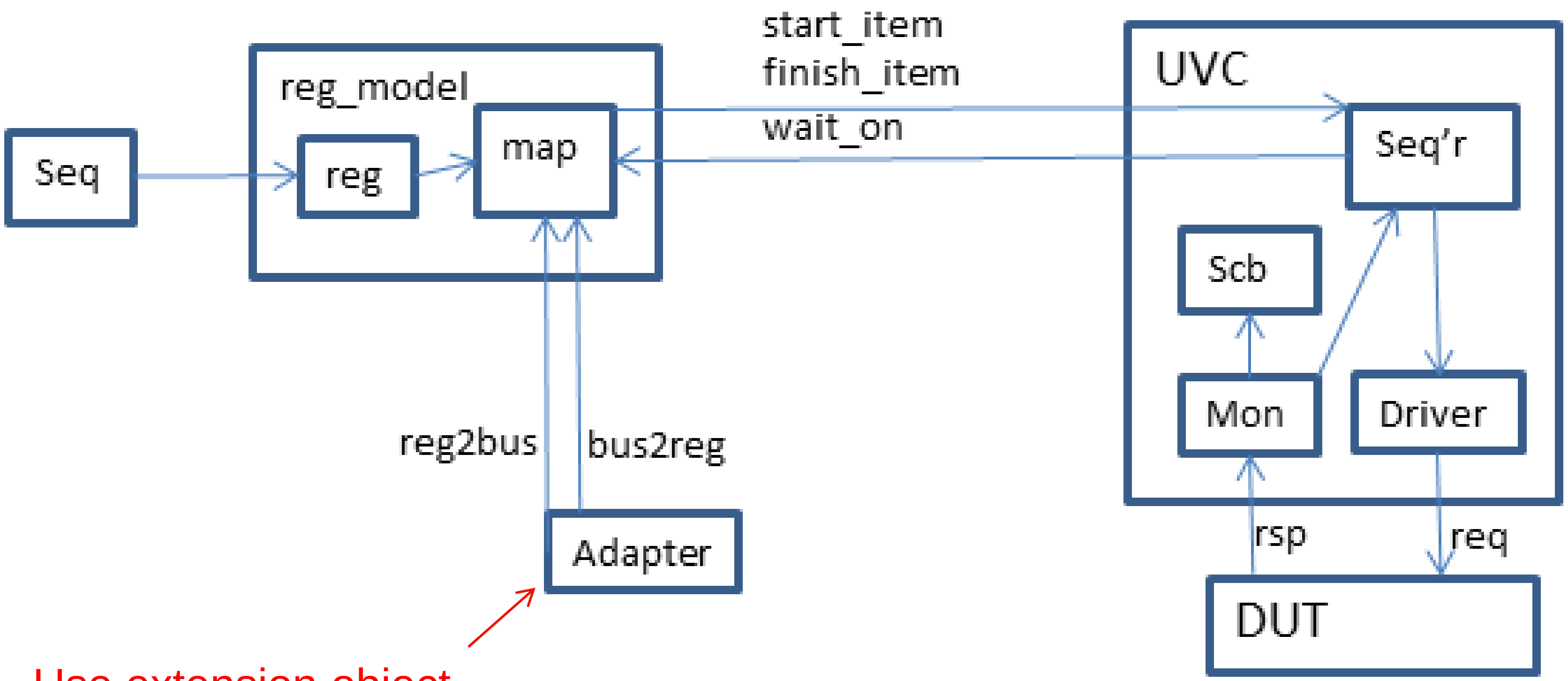
```
// Function: bus2reg
//
// Converts a <uvm_tlm_gp> item to a <uvm_reg_bus_op>.
// into the provided ~rw~ transaction.
//
virtual function void bus2reg(uvm_sequence_item bus_item,
                              ref uvm_reg_bus_op rw);

  uvm_tlm_gp gp;
  int nbytes;

  if (bus_item == null)
   `uvm_fatal("REG/NULL_ITEM","bus2reg: bus_item argument is null")

  if (!$cast(gp,bus_item)) begin
    `uvm_error("WRONG_TYPE","Provided bus_item is not of type uvm_tlm_gp")
    return;
  end

  if (gp.get_command() == UVM_TLM_WRITE_COMMAND)
    rw.kind = UVM_WRITE;
  else
    rw.kind = UVM_READ;

  rw.addr = gp.get_address();

  rw.byte_en = 0;
  foreach (gp.m_byte_enable[i])
    rw.byte_en[i] = gp.m_byte_enable[i];

  rw.data = 0;
  foreach (gp.m_data[i])
    rw.data[i*8+:8] = gp.m_data[i];

  rw.status = (gp.is_response_ok()) ? UVM_IS_OK : UVM_NOT_OK;


endfunction
```

Quick Overview
System with Adapter



Use uvm_reg_bus_op to pass in basic info. →

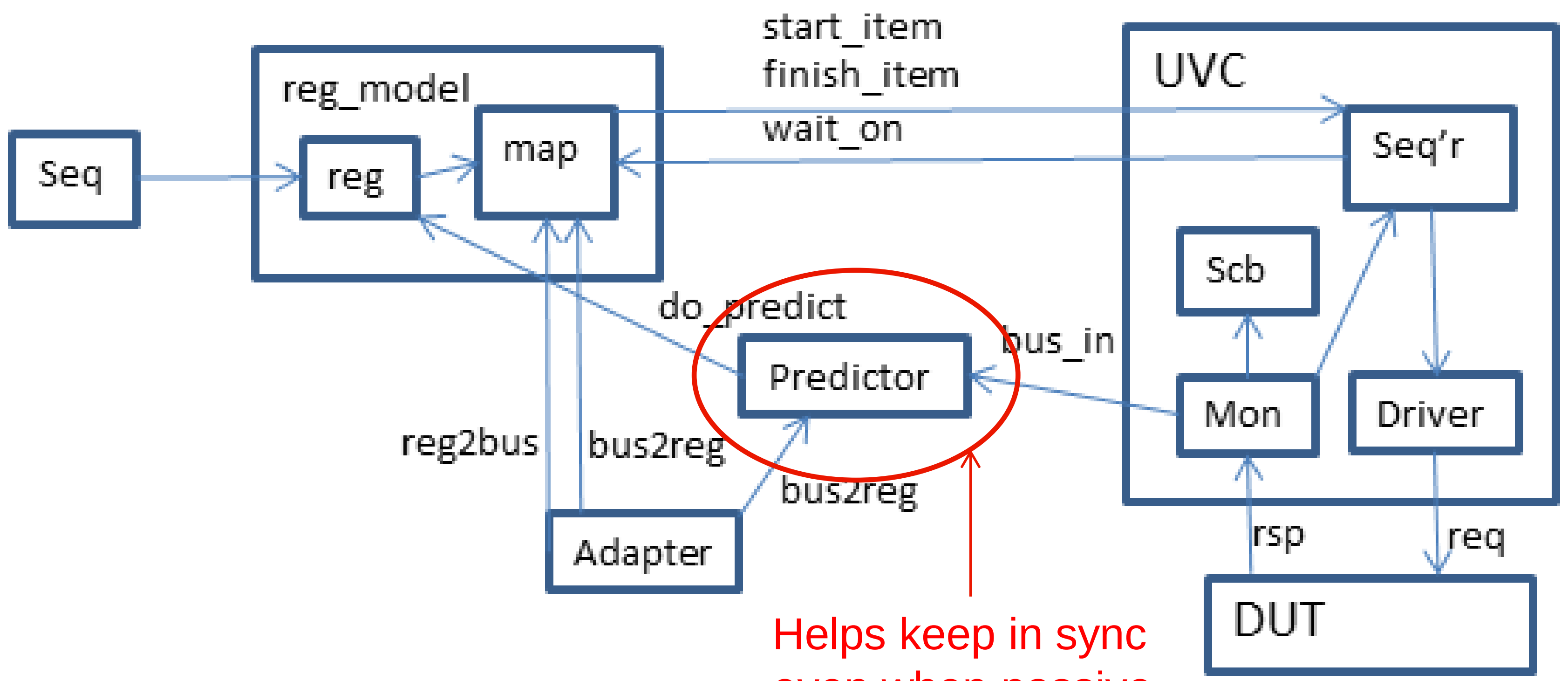
Quick Overview
System with Adapter



Use extension object
to pass in more info.

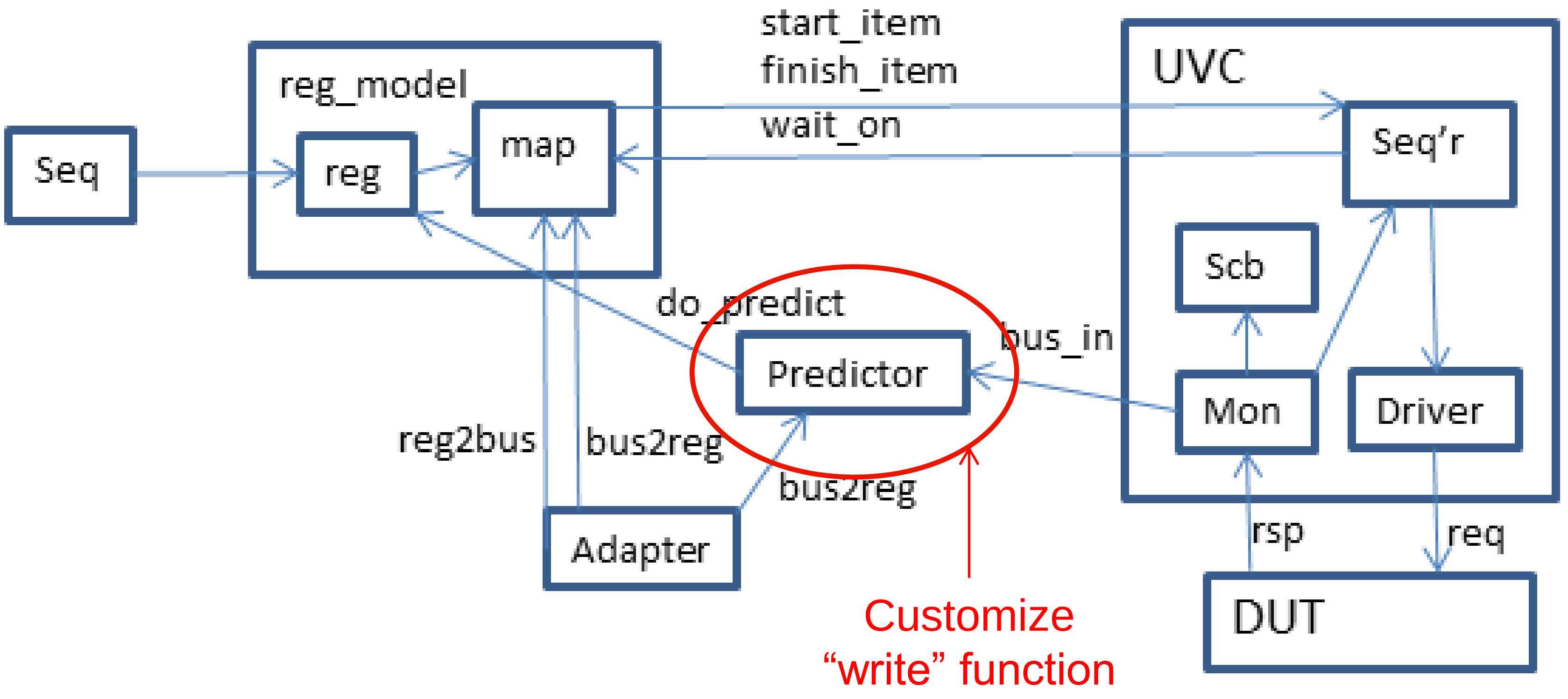
Quick Overview

System with Adapter and Predictor



Helps keep in sync even when passive (no Seq'r activity).

Quick Overview
System with Adapter and Predictor



Quick Overview
Predictor write

```
// Function- write
//
// not a user-level method. Do not call directly. See documentation
// for the ~bus_in~ member.
//
virtual function void write(BUSTYPE tr);
   uvm_reg rg;
   uvm_reg_bus_op rw;
  if (adapter == null)
   `uvm_fatal("REG/WRITE/NULL","write: adapter handle is null")

   // In case they forget to set byte_en
   rw.byte_en = -1;
   adapter.bus2reg(tr,rw);
   rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));

   // ToDo: Add memory look-up and call <uvm_mem::XsampleX()>

   if (rg != null) begin
     bit found;
     uvm_reg_item reg_item;
     uvm_reg_map local_map;
     uvm_reg_map_info map_info;
     uvm_predict_s predict_info;
     uvm_reg_indirect_data ireg;
     uvm_reg ir;

     if (!m_pending.exists(rg)) begin
       uvm_reg_item item = new;
       predict_info =new;
       item.element_kind = UVM_REG;
       item.element      = rg;
       item.path         = UVM_PREDICT;
       item.map          = map;
       item.kind         = rw.kind;
       predict_info.reg_item = item;
       m_pending[rg] = predict_info;
     end
     predict_info = m_pending[rg];
     reg_item = predict_info.reg_item;
```

Adapter Customization

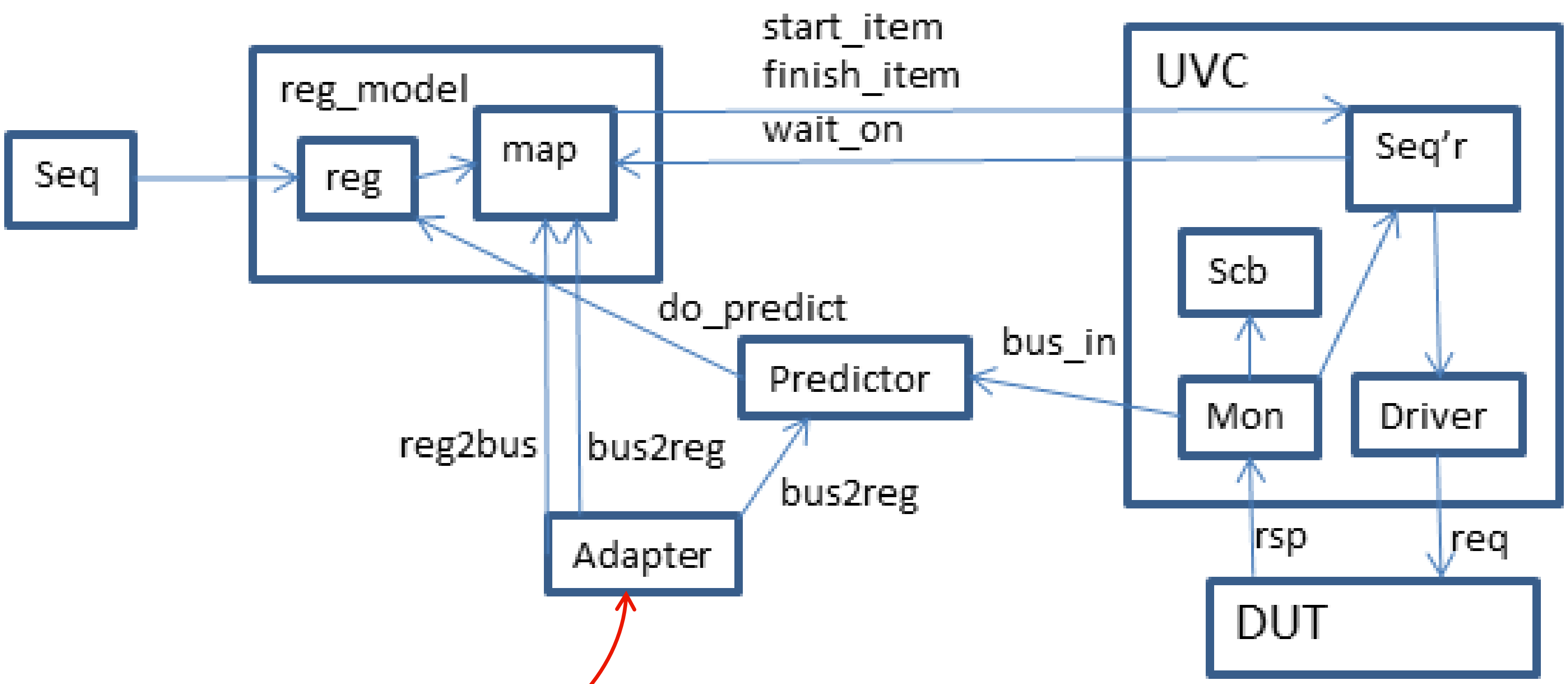
Adapter Customization

Gotchas

- provides_responses
  - Hang!
  - Ignored responses
- Extension Object
  - Lost variables
- supports_byte_enable = 1
  - *May* require customization

Adapter Customization

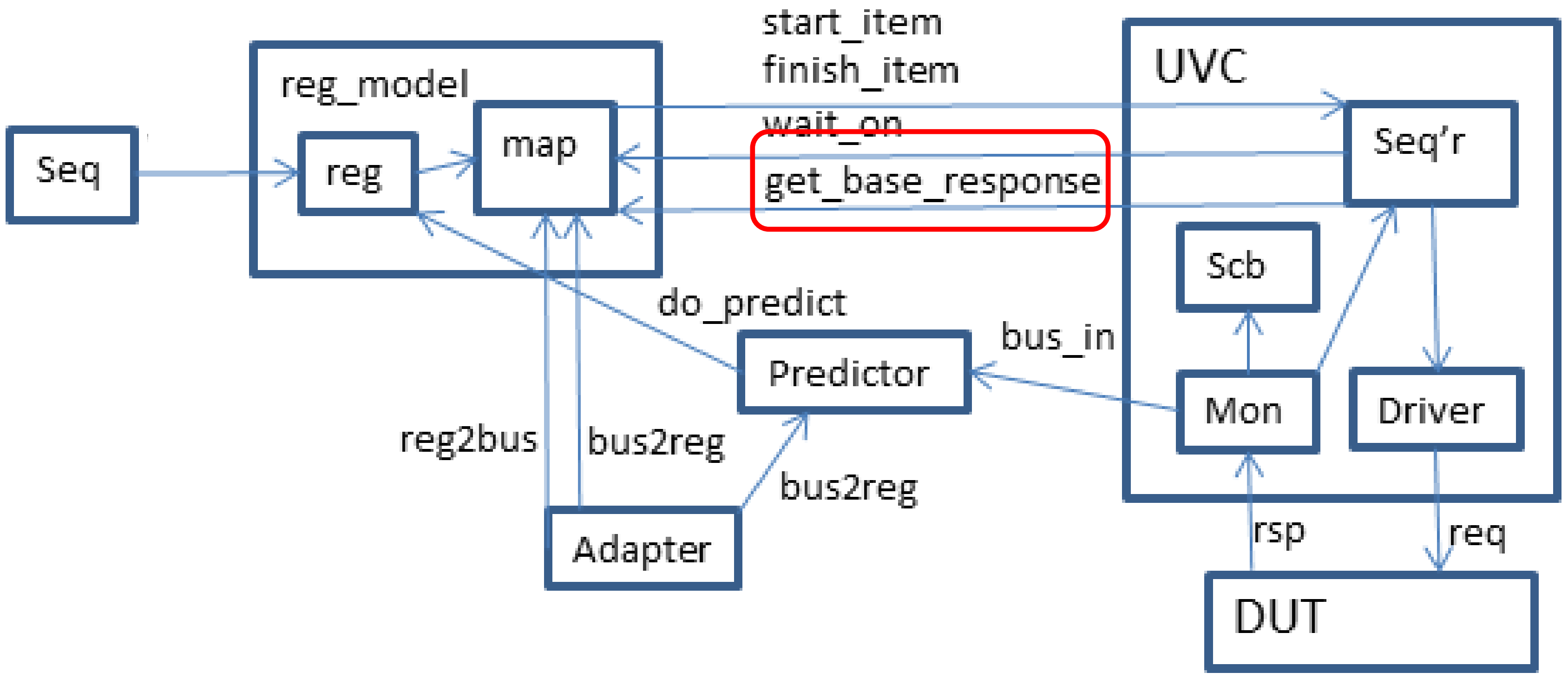
provides_responses = 0



provides_responses
stored here

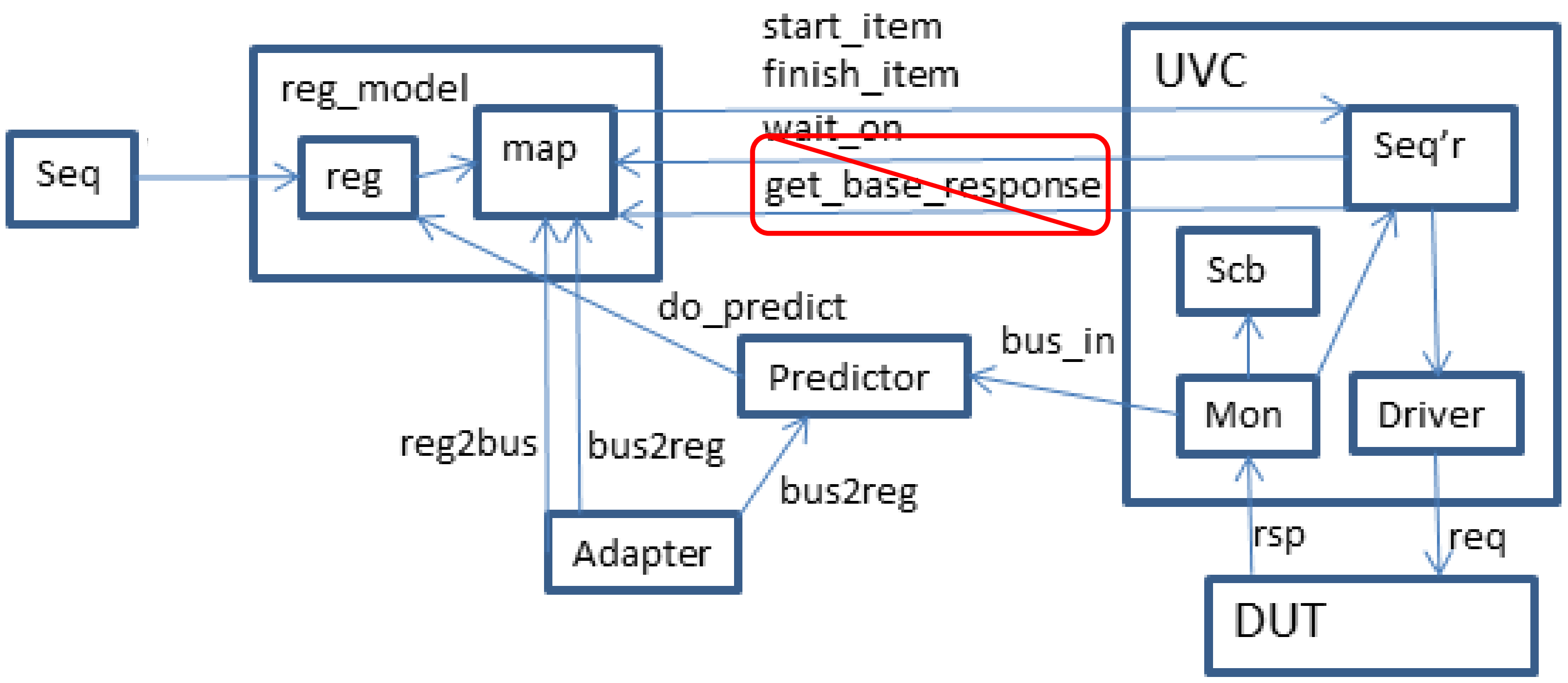
Adapter Customization

provides_responses = 1



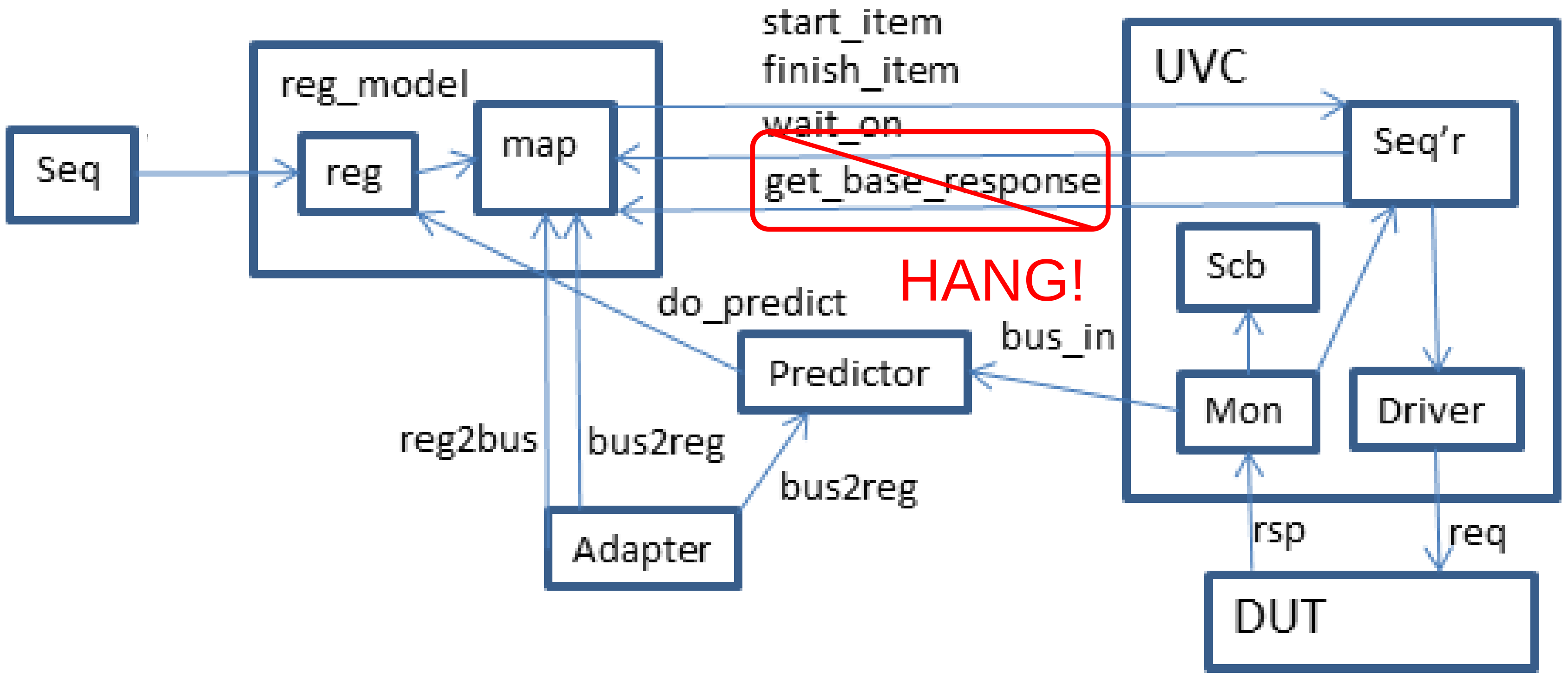
Adapter Customization

provides_responses = 1, no get_base_response



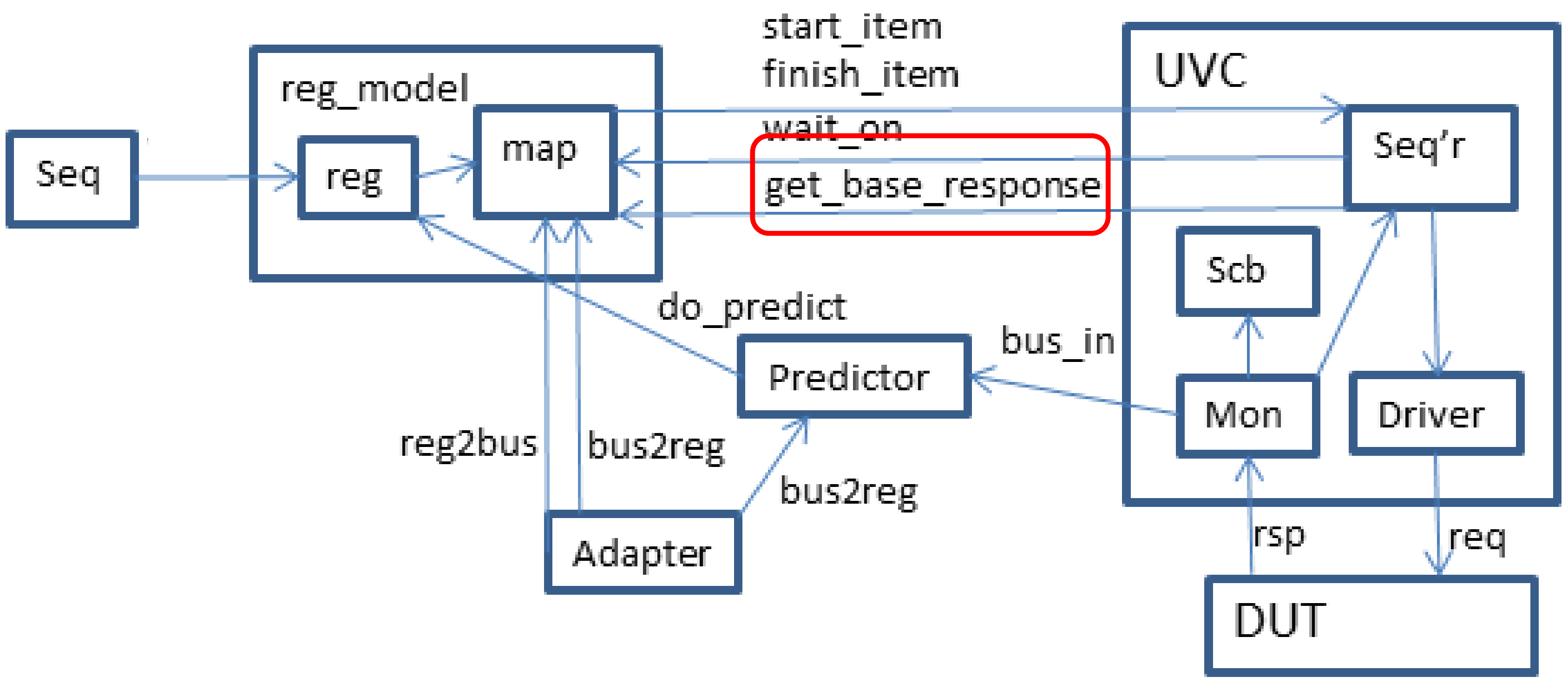
Adapter Customization GOTCHA!

provides_responses = 1, no get_base_response



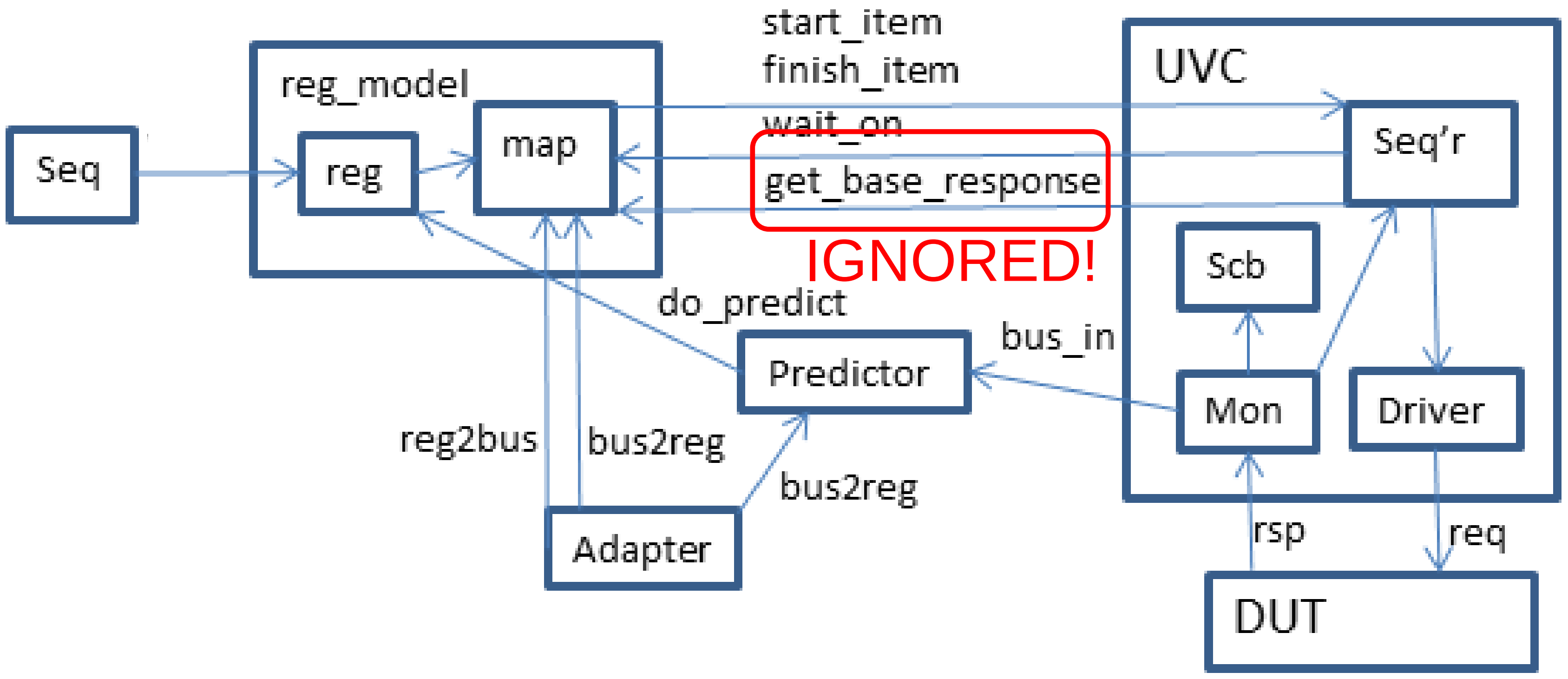
Adapter Customization

provides_responses = 0, get_base_response sent



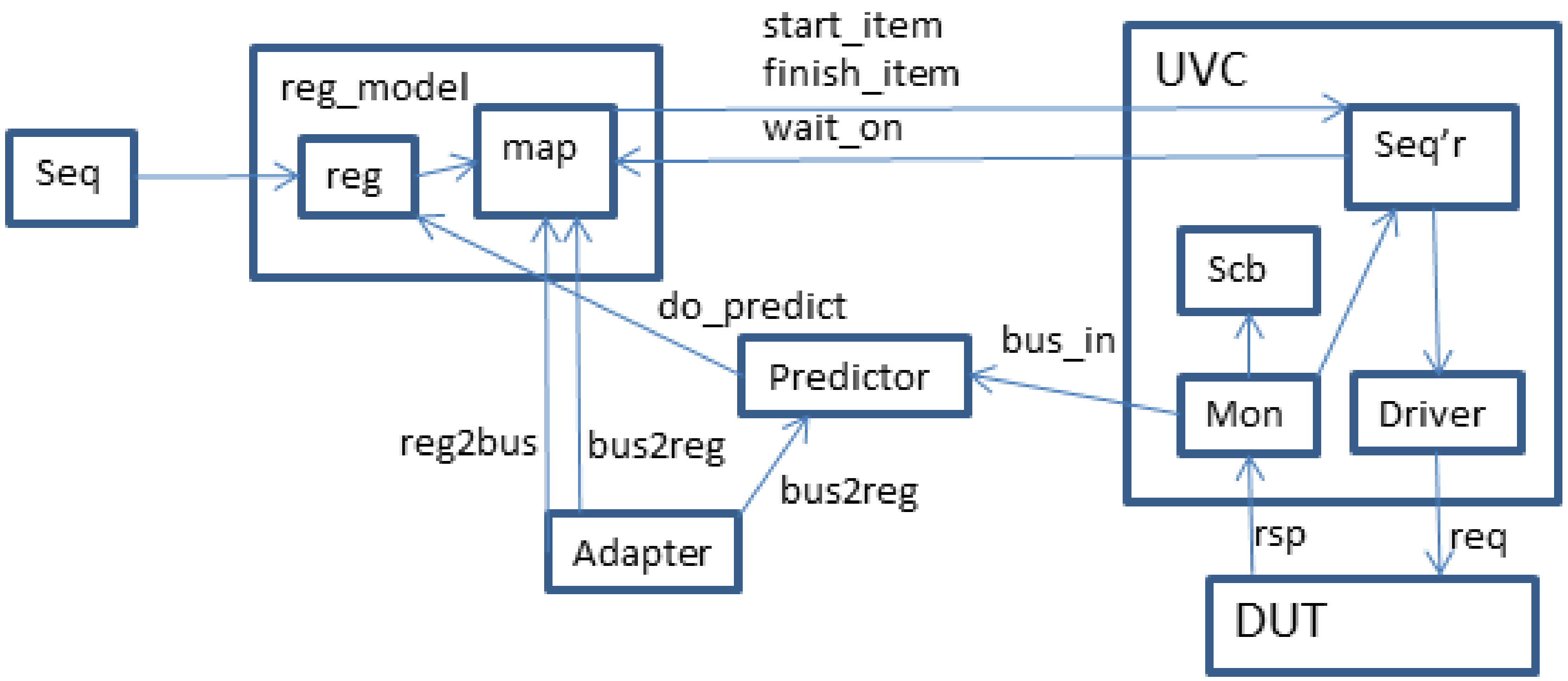
Adapter Customization GOTCHA!

provides_responses = 0, get_base_response sent



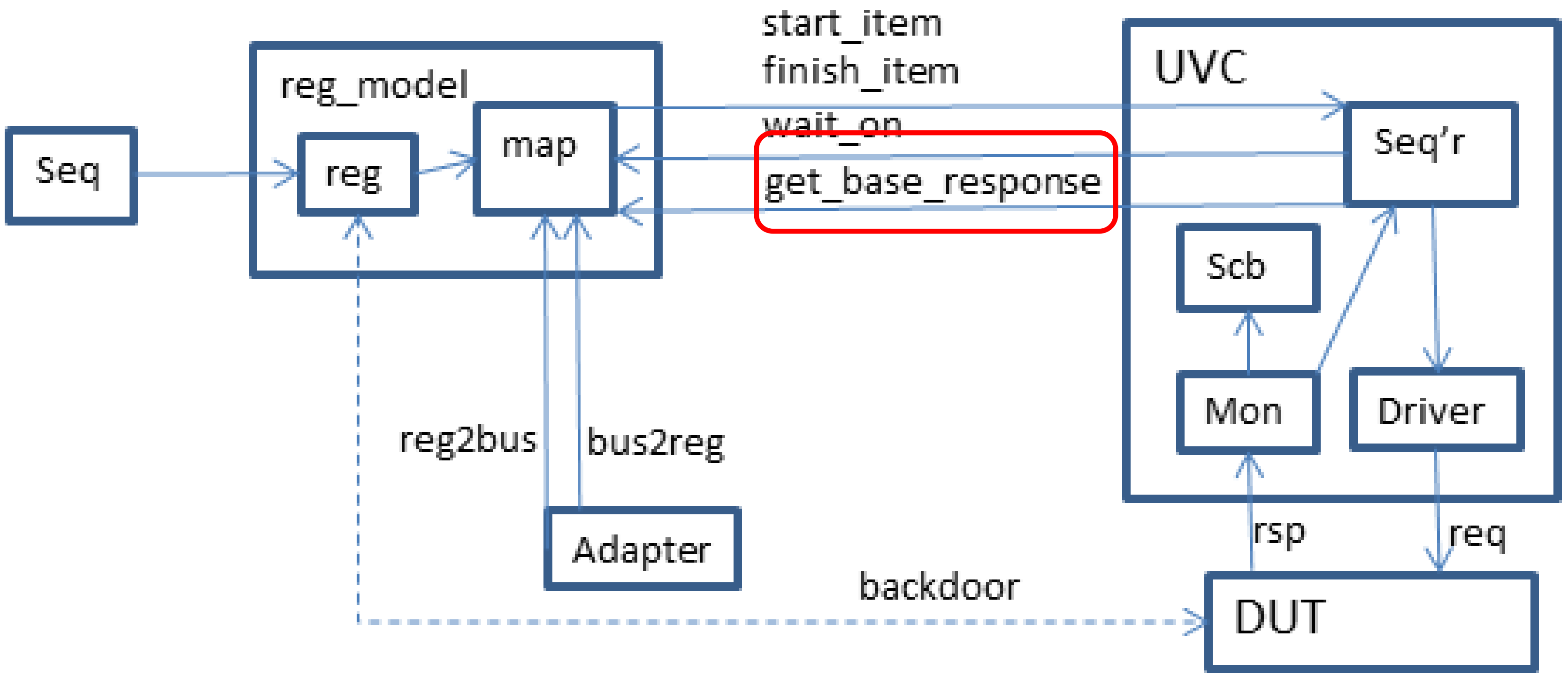
Adapter Customization

provides_responses = 0 with predictor



Adapter Customization

provides_responses = 1, no predictor



get_item in reg2bus

Passes in the extension, clone function

```
// Function: reg2bus
// Converts generic register access items to sdp bus sequence items
function uvm_sequence_item my_adapter::reg2bus(const ref uvm_reg_bus_op rw);

  my_pkg::my_extension extension;

  // Get the extension.
  uvm_reg_item item = get_item();
  if (item.extension != null) begin
    if (!$cast(extension, item.extension.clone())) begin
      `uvm_error(get_type_name(), "Extension casting failed")
    end
  end
  else begin
    `uvm_error(get_type_name(), "Null extension provided!")
  end
```

create_burst

Function within extension

```
my_extension.create_burst(data, len, byte_en);
reg_obj.write(status, data, .path(path), .parent(this),
                  .map(map), .extension(my_extension));
```

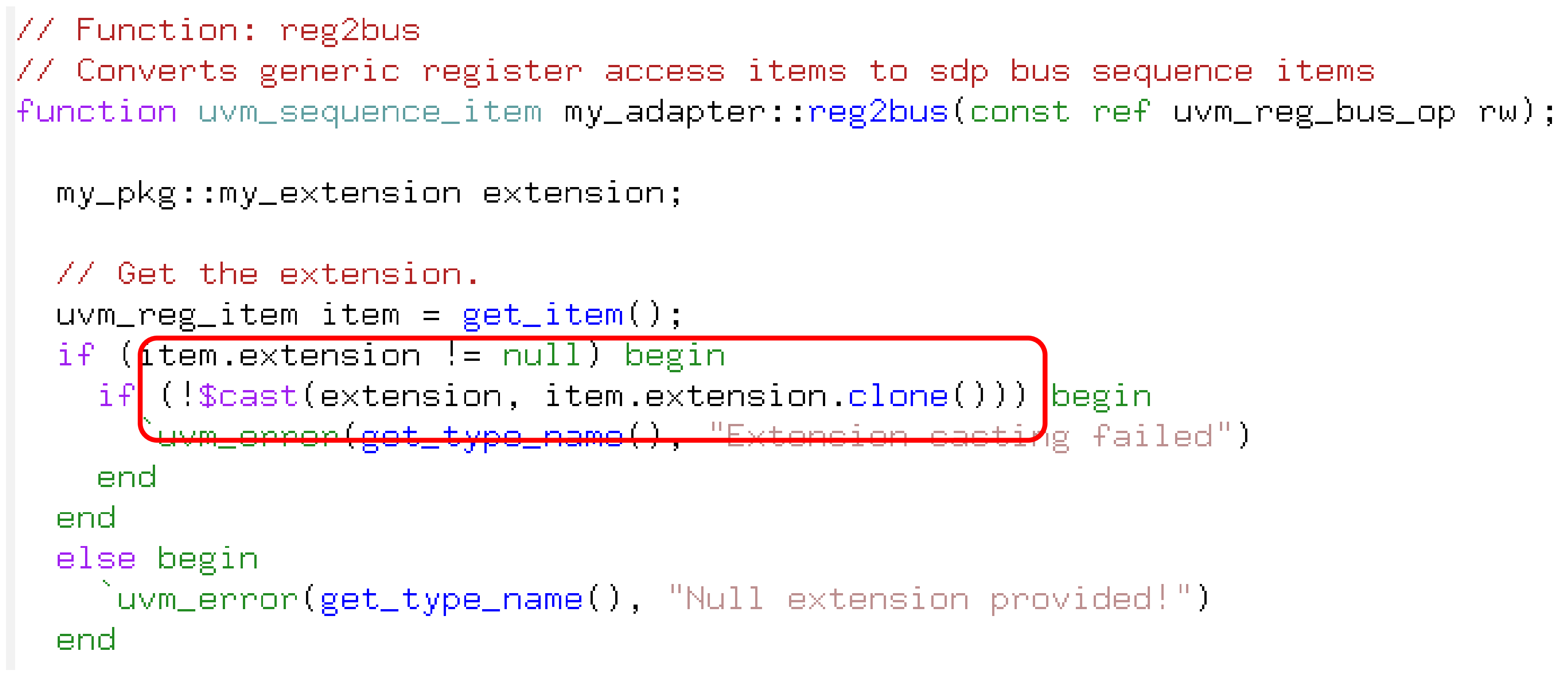
get_item in reg2bus

Passes in the extension, clone function

```
// Function: reg2bus
// Converts generic register access items to sdp bus sequence items
function uvm_sequence_item my_adapter::reg2bus(const ref uvm_reg_bus_op rw);

  my_pkg::my_extension extension;

  // Get the extension.
  uvm_reg_item item = get_item();
  if (item.extension != null) begin
    if (!$cast(extension, item.extension.clone())) begin
      `uvm_error(get_type_name(), "Extension casting failed")
    end
  end
  else begin
    `uvm_error(get_type_name(), "Null extension provided!")
  end
```

get_item in reg2bus Gotcha!

extension do_copy customization needed



```
// Function: reg2bus
// Converts generic register access items to sdp bus sequence items
function uvm_sequence_item my_adapter::reg2bus(const ref uvm_reg_bus_op rw);

  my_pkg::my_extension extension;

  // Get the extension.
  uvm_reg_item item = get_item();
  if (item.extension != null) begin
    if (!$cast(extension, item.extension.clone())) begin
      `uvm_error(get_type_name(), "Extension casting failed")
    end
  end
  else begin
    `uvm_error(get_type_name(), "Null extension provided!")
  end
```

Extension Object Class

do_copy for reg2bus clone function

```
class my_extension extends uvm_sequence_item;
    `uvm_object_utils(my_extension)
  int len = 0;
  int byte_en = -1;

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

  virtual function void do_copy(uvm_object rhs);
    my_extension _rhs;
    super.do_copy(rhs);
    void'($cast(_rhs, rhs));
    // NOTE: Copy all variables here.
    len = _rhs.len;
    byte_en = _rhs.byte_en;
  endfunction : do_copy
```

Extension Object Class

do_copy for reg2bus clone function

```
class my_extension extends uvm_sequence_item;
    `uvm_object_utils(my_extension)
  int len = 0;
  int byte_en = -1;

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

  virtual function void do_copy(uvm_object rhs);
    my_extension _rhs;
    super.do_copy(rhs);
    void'($cast(_rhs, rhs));
    // NOTE: Copy all variables here.
    len = _rhs.len;
    byte_en = _rhs.byte_en;
  endfunction : do_copy
```

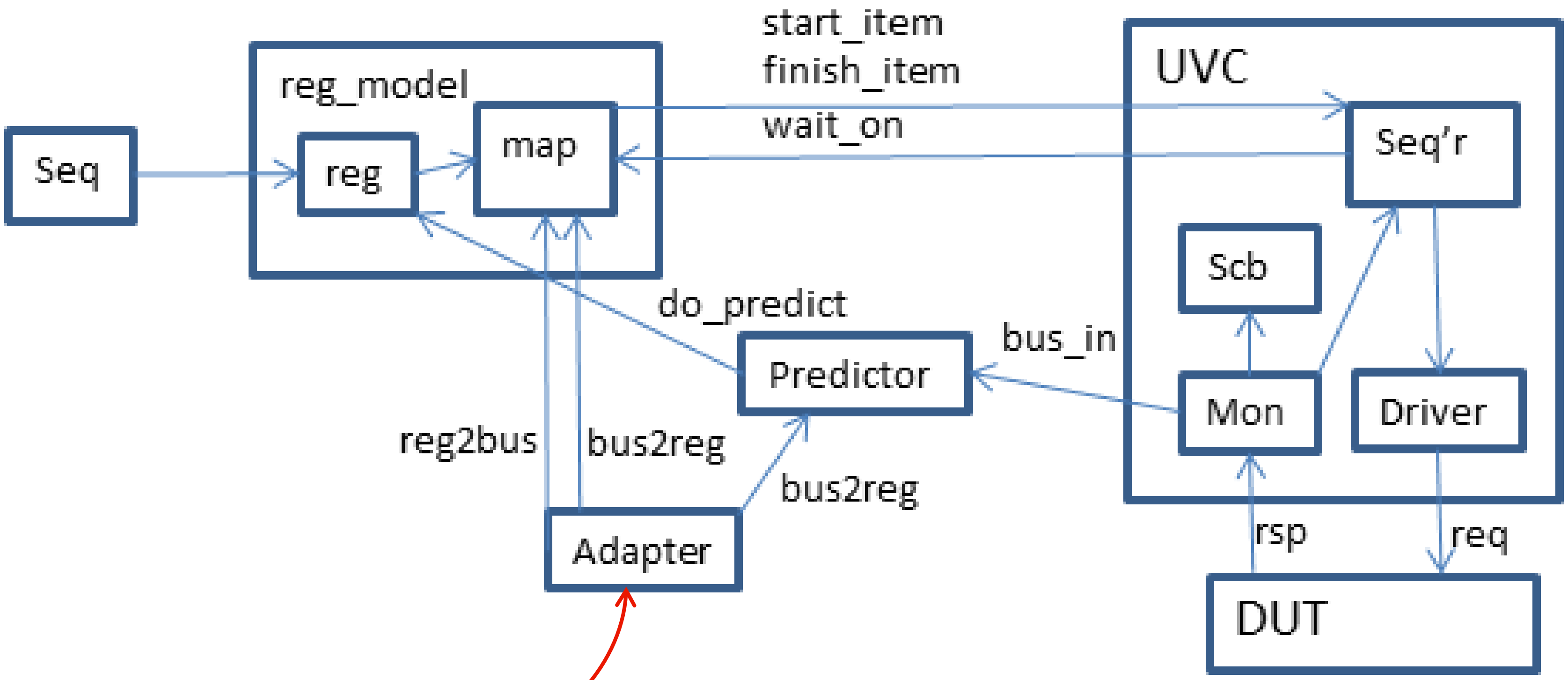
Extension Object Class

do_copy for reg2bus clone function

```
class my_extension extends uvm_sequence_item;
    `uvm_object_utils(my_extension)
  int len = 0;
  int byte_en = -1;

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

  virtual function void do_copy(uvm_object rhs);
    my_extension _rhs;
    super.do_copy(rhs);
    void'($cast(_rhs, rhs));
    // NOTE: Copy all variables here.
    len = _rhs.len;
    byte_en = _rhs.byte_en;
  endfunction : do_copy
```

Byte Enables

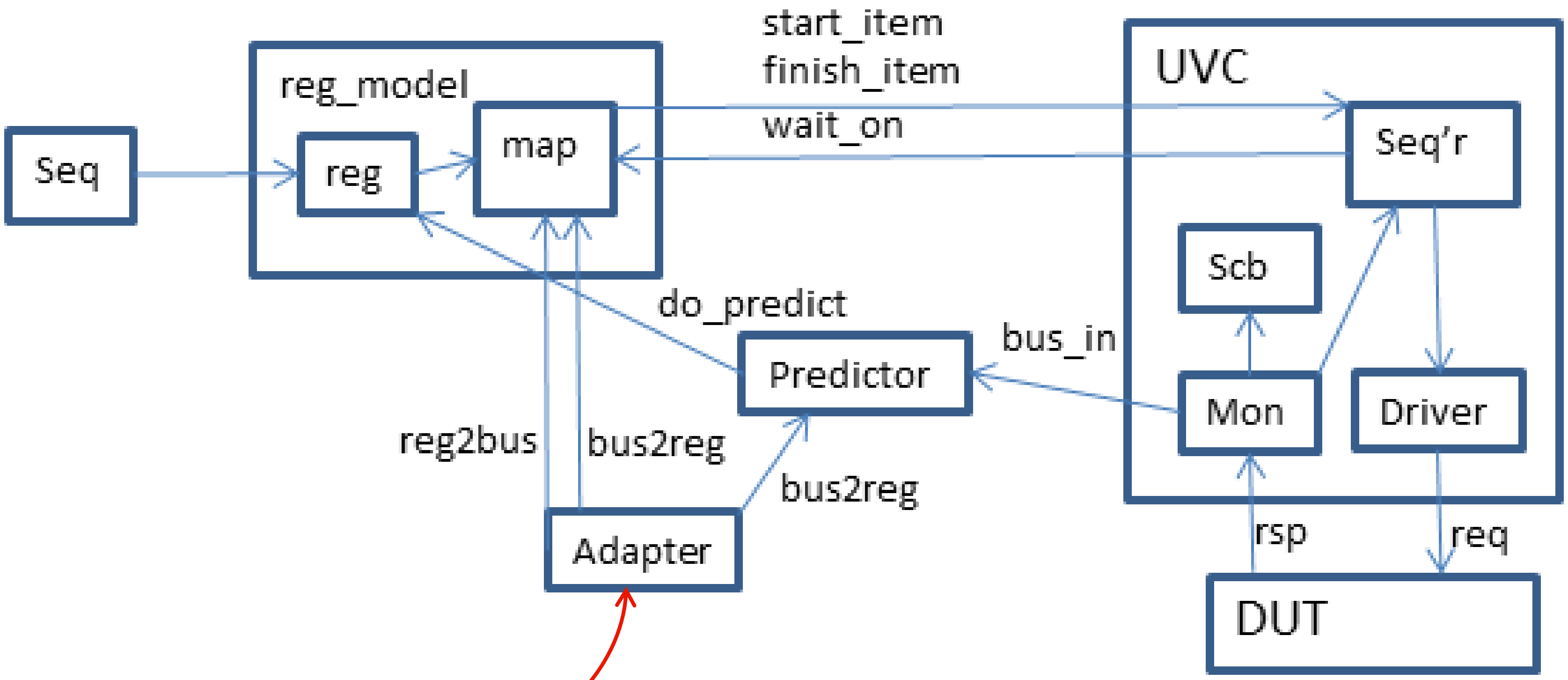
supports_byte_enable = 1



supports_byte_enable
also stored here

Byte Enables Gotcha!

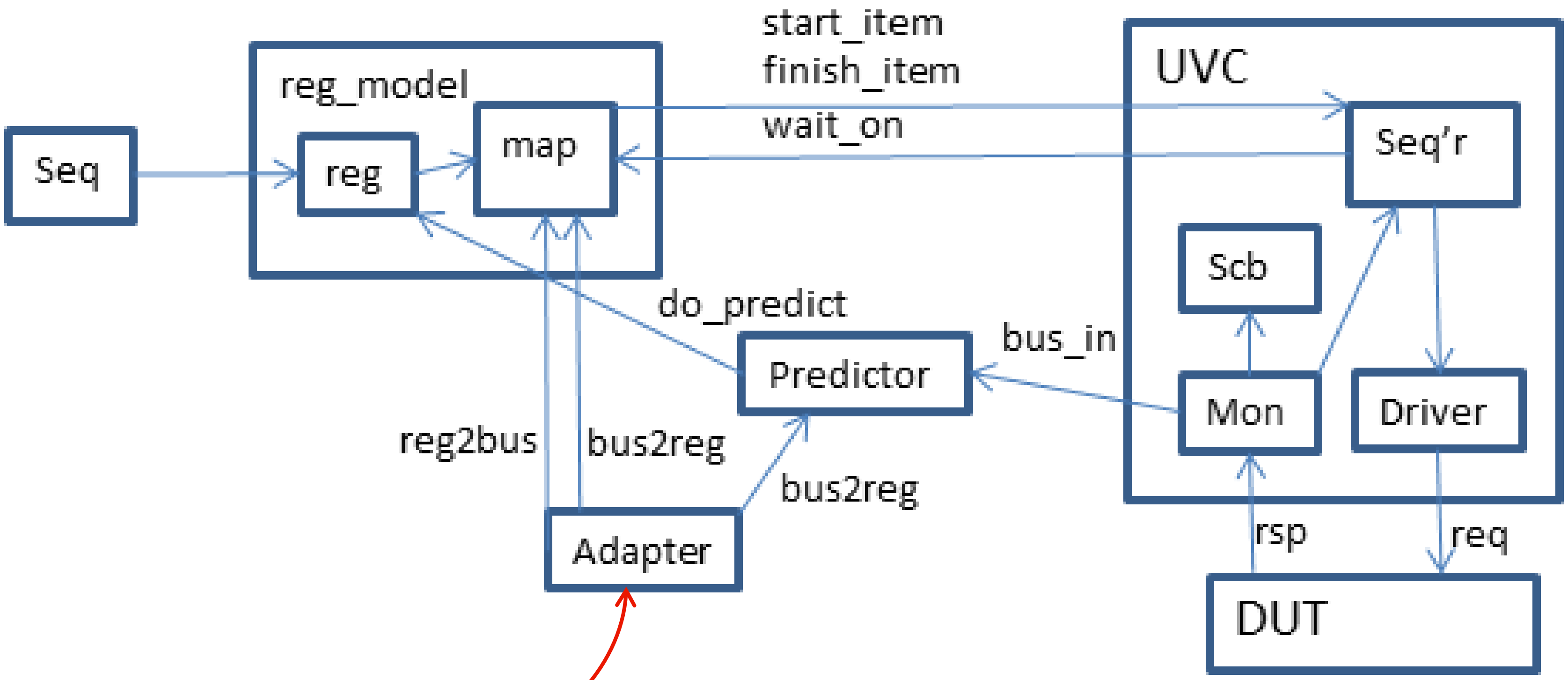
supports_byte_enable = 1



supports_byte_enable
also stored here

Byte Enables Gotcha (maybe)!

supports_byte_enable = 1



supports_byte_enable
also stored here

Byte Enables

Passed to reg2bus via extension

```
logic [63:0]  temp_byten = extension.byte_en;
```

Byte Enables

Passed to bus2reg via bus_item

```
function void my_adapter::bus2reg(uvm_sequence_item bus_item,
                                  ref uvm_reg_bus_op rw);

  my_uvc_pkg::my_data_item my_item;

  if (!$cast(my_item, bus_item)) begin
    `uvm_fatal(get_type_name(),"Provided item type is not expected")
  end

  if (my_item.req._cmd inside {my_uvc_pkg::WRITE_CMD}) begin
    rw.kind = UVM_WRITE;
    rw.addr = my_item.req._addr;
    rw.byte_en = {my_item.wdata._byten[1], my_item.wdata._byten[0]};
    rw.data = {my_item.wdata._data[1], my_item.wdata._data[0]};
    rw.status = my_item.status;
  end
```

Byte Enables

Passed to bus2reg via bus_item

```
function void my_adapter::bus2reg(uvm_sequence_item bus_item,
                                  ref uvm_reg_bus_op rw);

  my_uvc_pkg::my_data_item my_item;

  if (!$cast(my_item, bus_item)) begin
    `uvm_fatal(get_type_name(),"Provided item type is not expected")
  end

  if (my_item.req._cmd inside {my_uvc_pkg::WRITE_CMD}) begin
    rw.kind = UVM_WRITE;
    rw.addr = my_item.req._addr;
    rw.byte_en = {my_item.wdata._byten[1], my_item.wdata._byten[0]};
    rw.data = {my_item.wdata._data[1], my_item.wdata._data[0]};
    rw.status = my_item.status;
  end
```

Predictor Customization

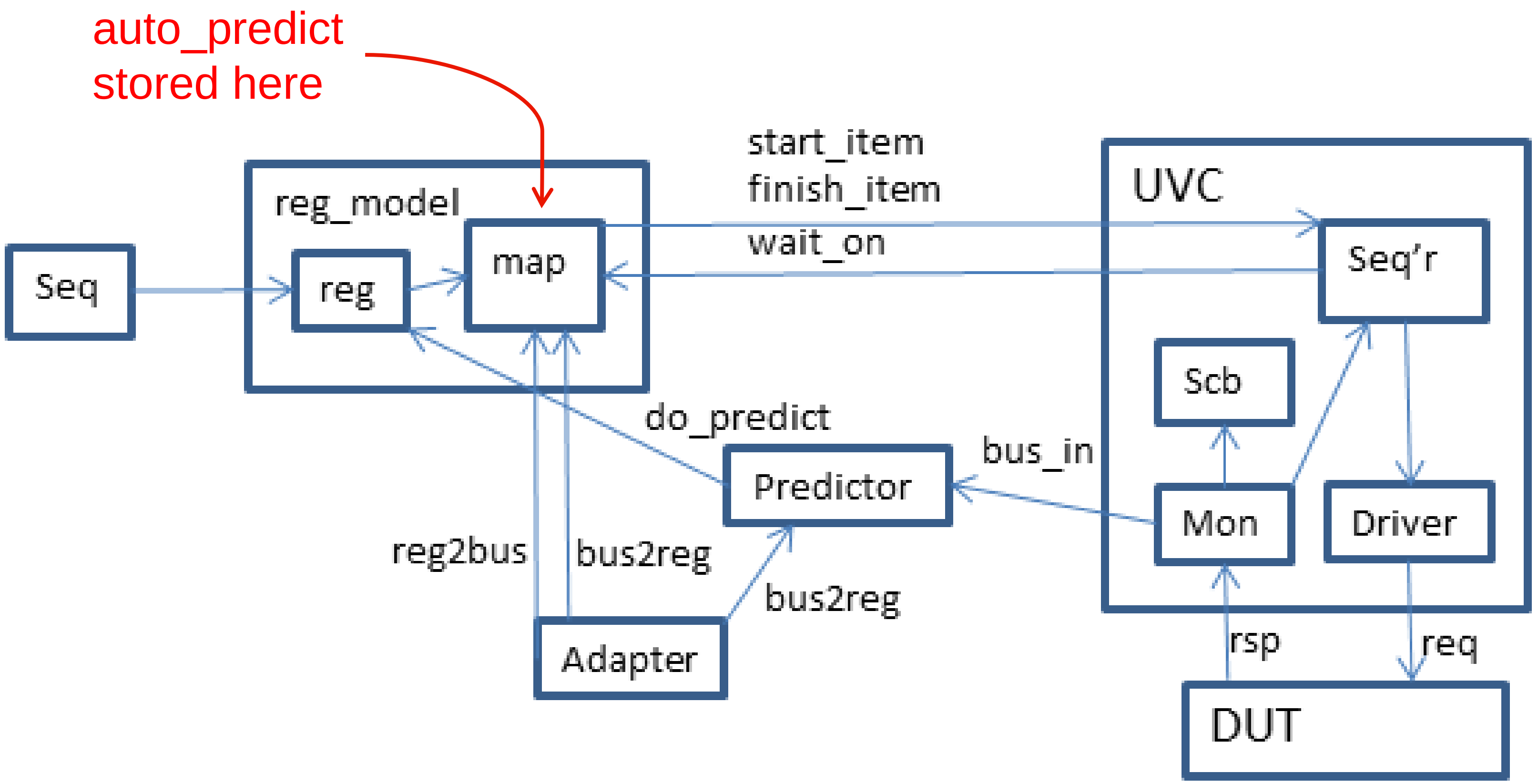
Predictor Customization

Gotchas

- auto_predict = 1
  - Sync loss
  - Backdoor read miscompare
- auto_predict = 0
  - Sync loss
- supports_byte_enable = 1
  - do_check false mismatch
  - do_predict bytes changed errantly
  - do_predict bytes skipped errantly
- “get” vs “get_mirrored_value” sync loss
- post_write callback miss
- Indirect register check miss

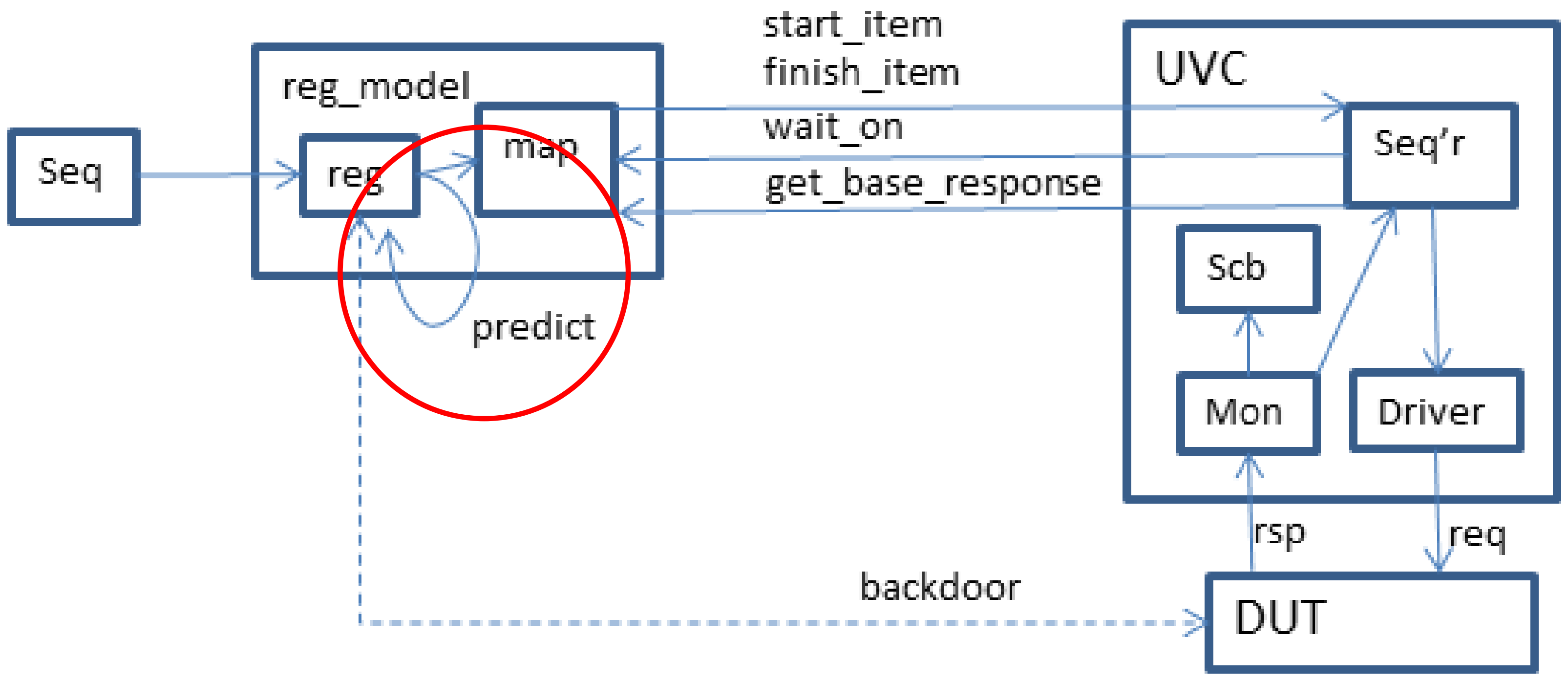
Auto Predict

auto_predict = 0, predictor present



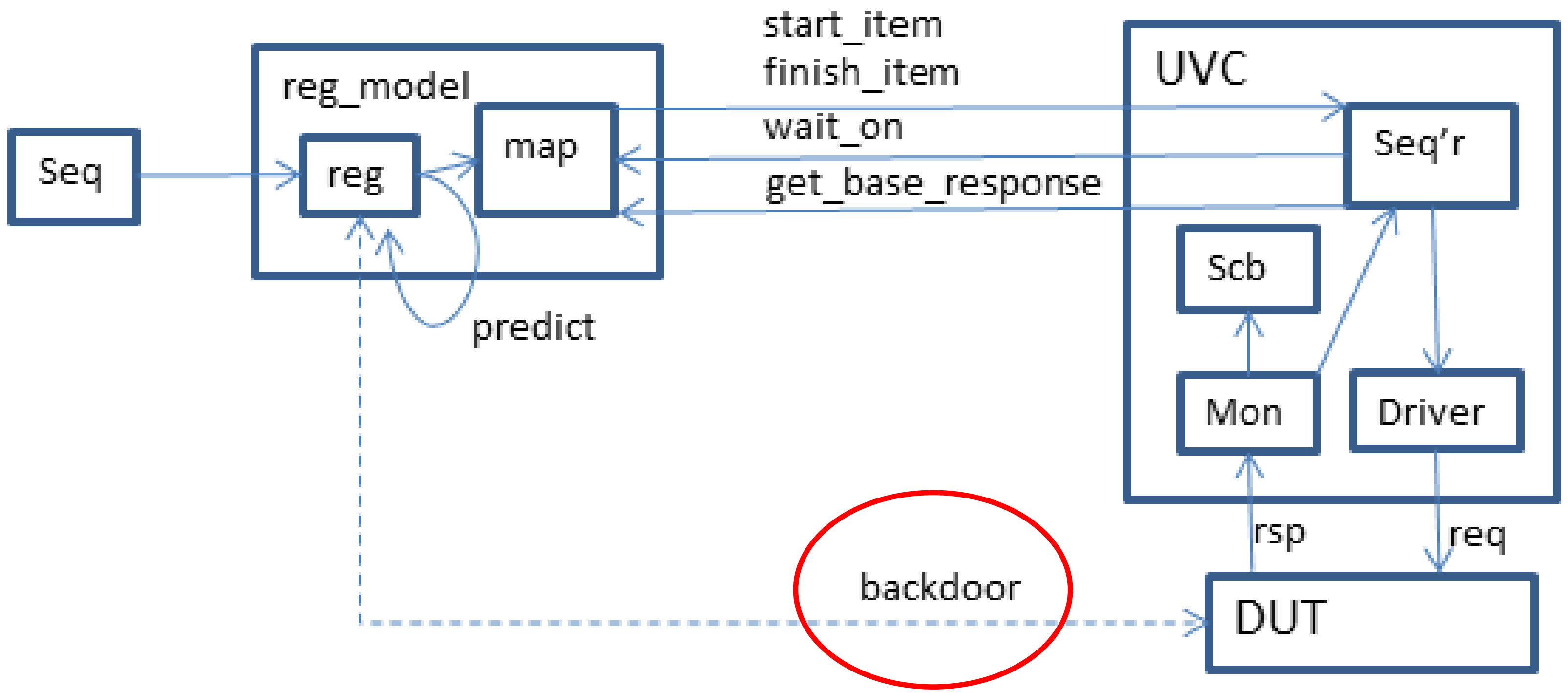
Auto Predict

auto_predict = 1, local predict, backdoor check



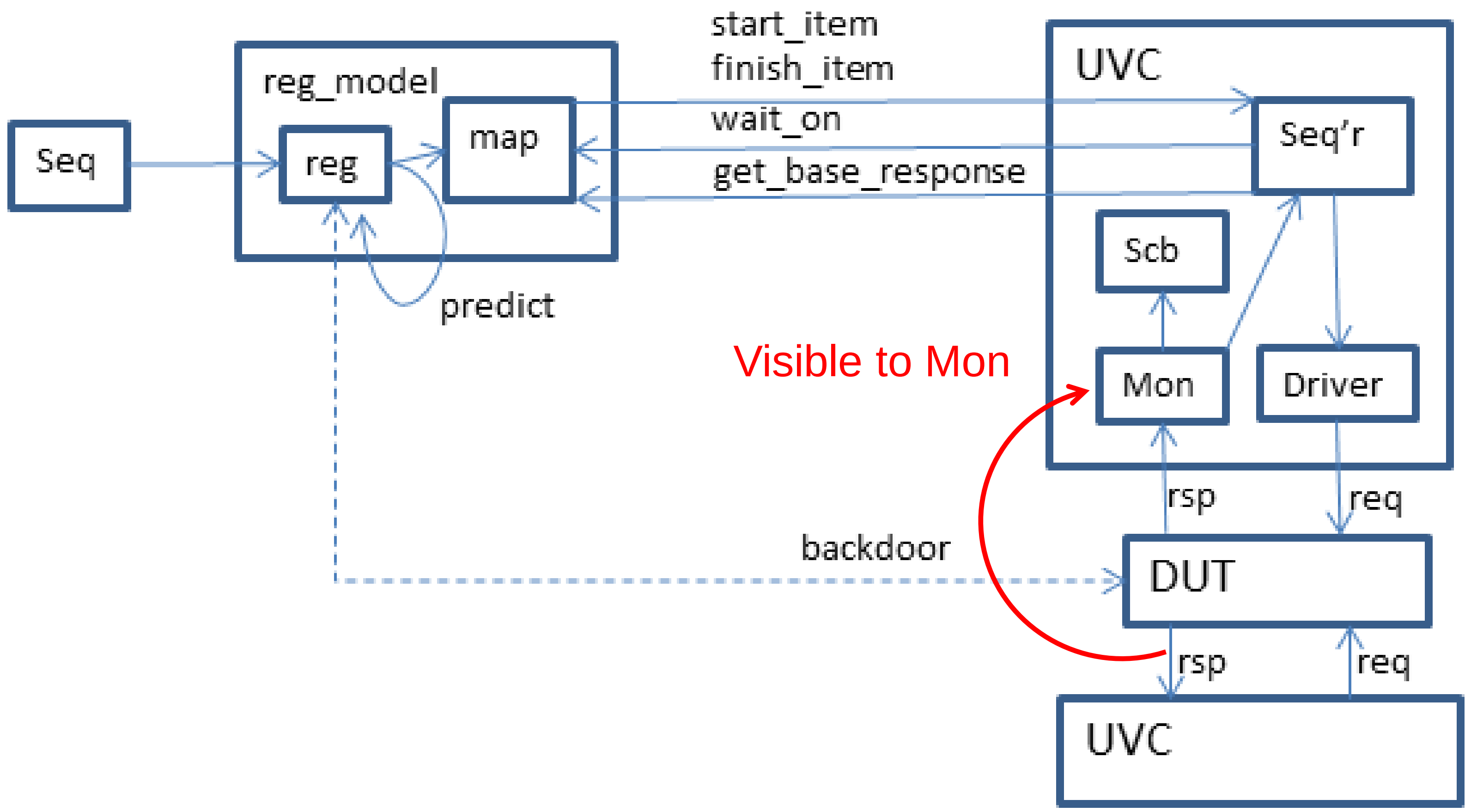
Auto Predict

auto_predict = 1, local predict, backdoor check



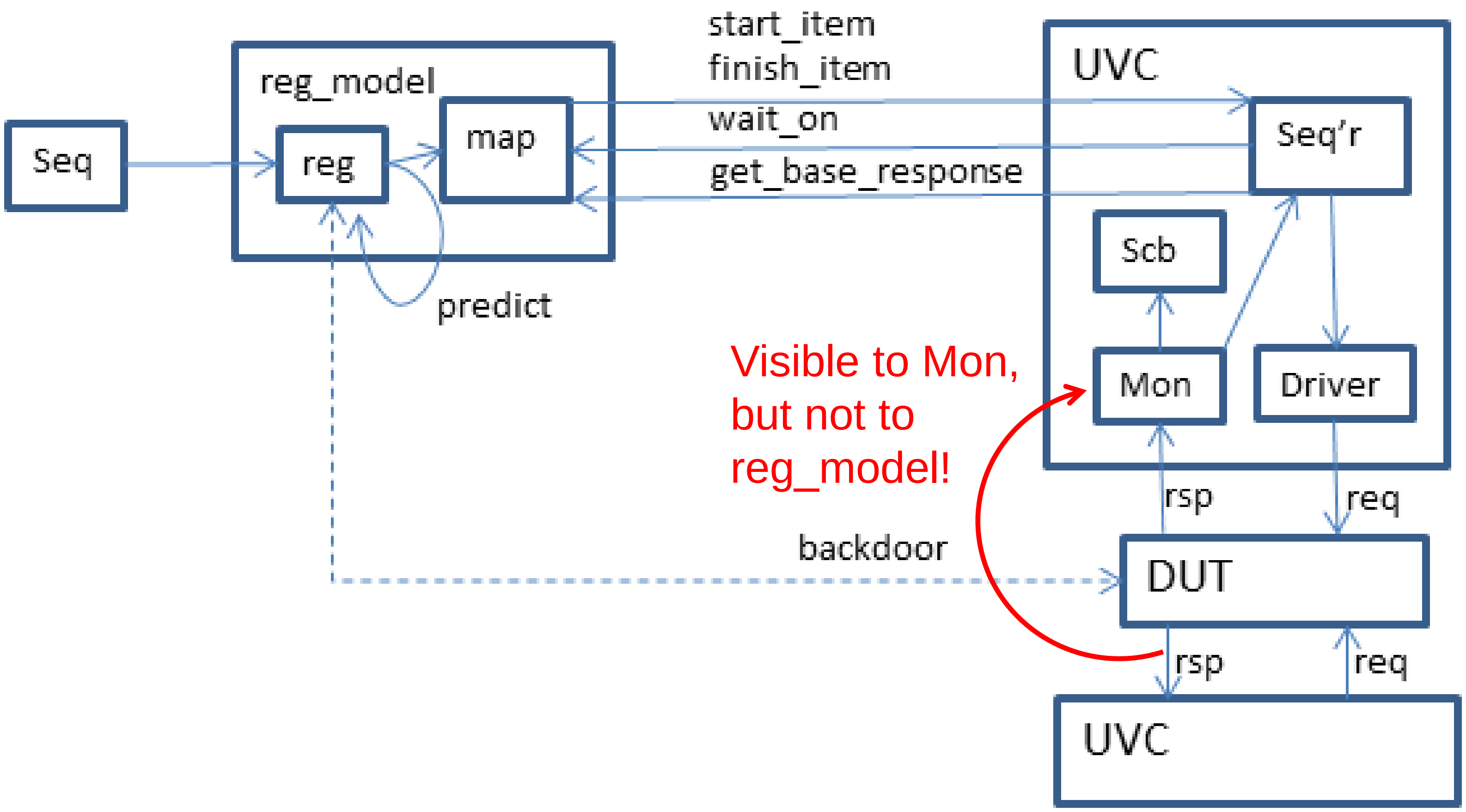
Auto Predict

auto_predict = 1, local predict, backdoor check



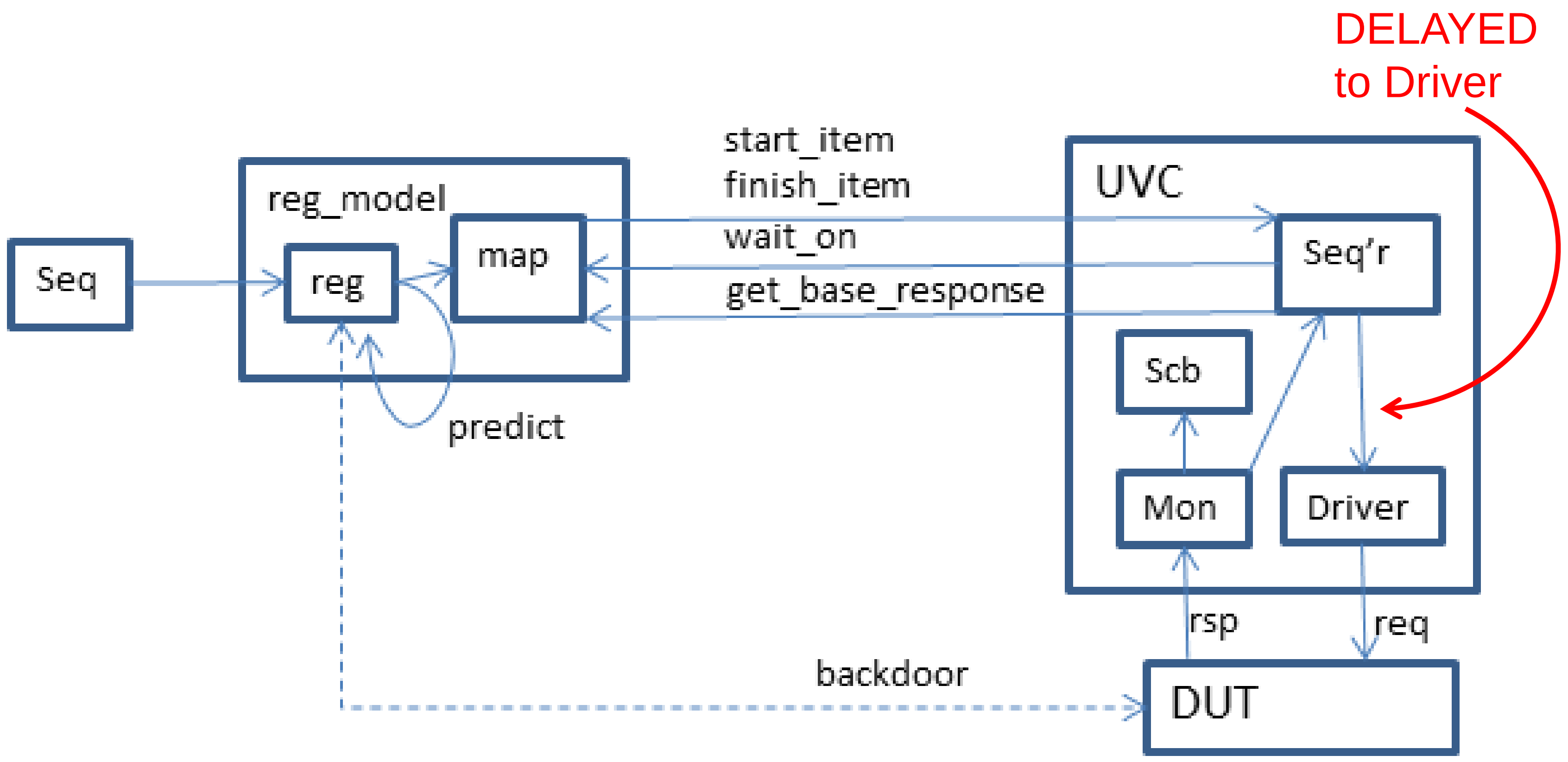
Auto Predict Gotcha!

auto_predict = 1, local predict, backdoor check



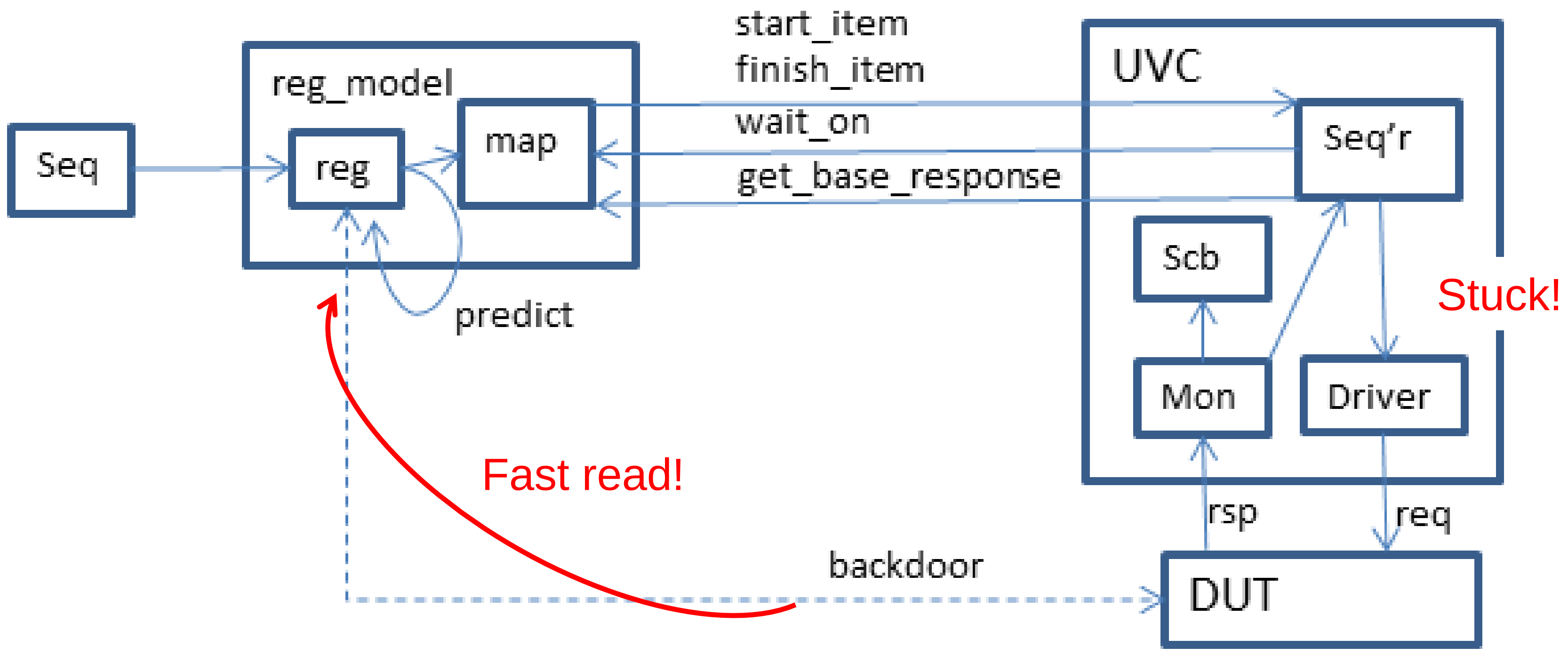
Auto Predict

auto_predict = 1, local predict, backdoor check



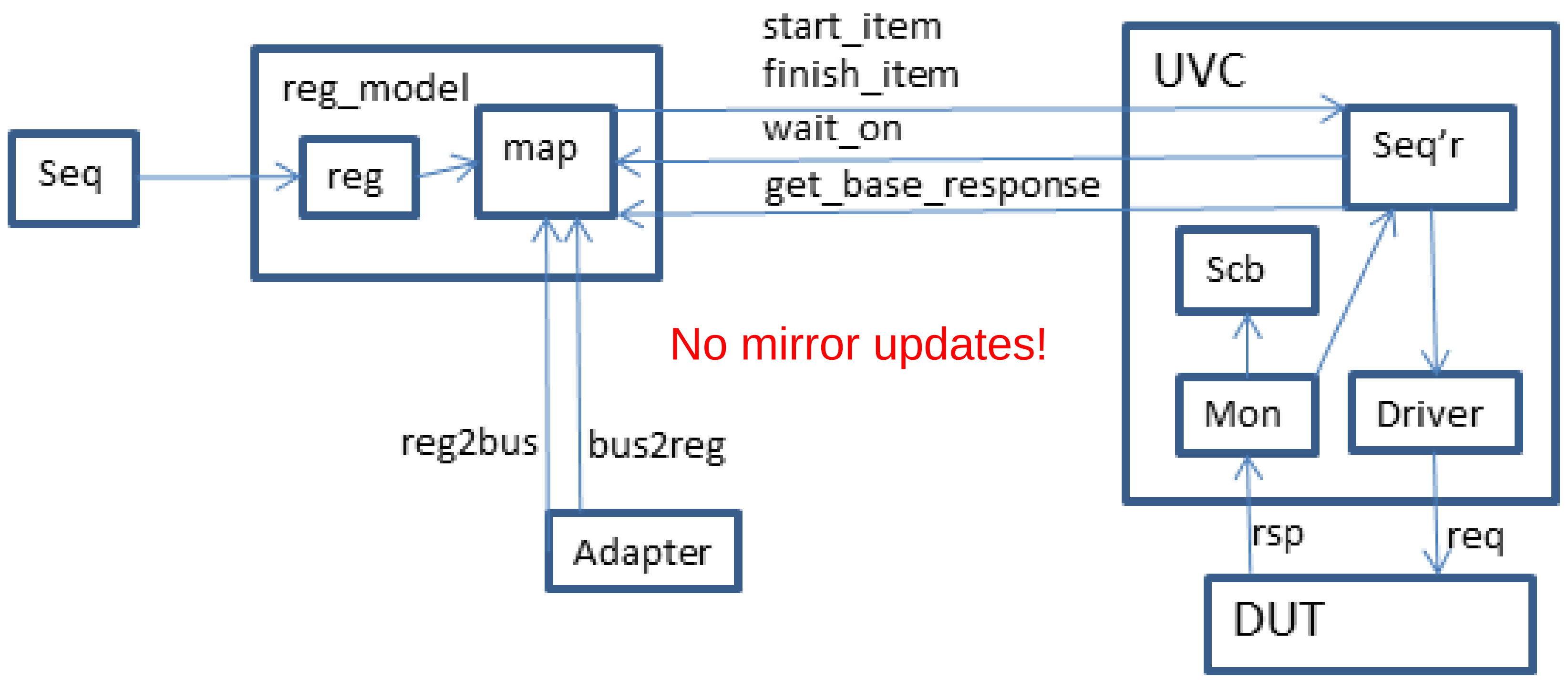
Auto Predict Gotcha!

auto_predict = 1, local predict, backdoor check



Loss of Mirror Sync

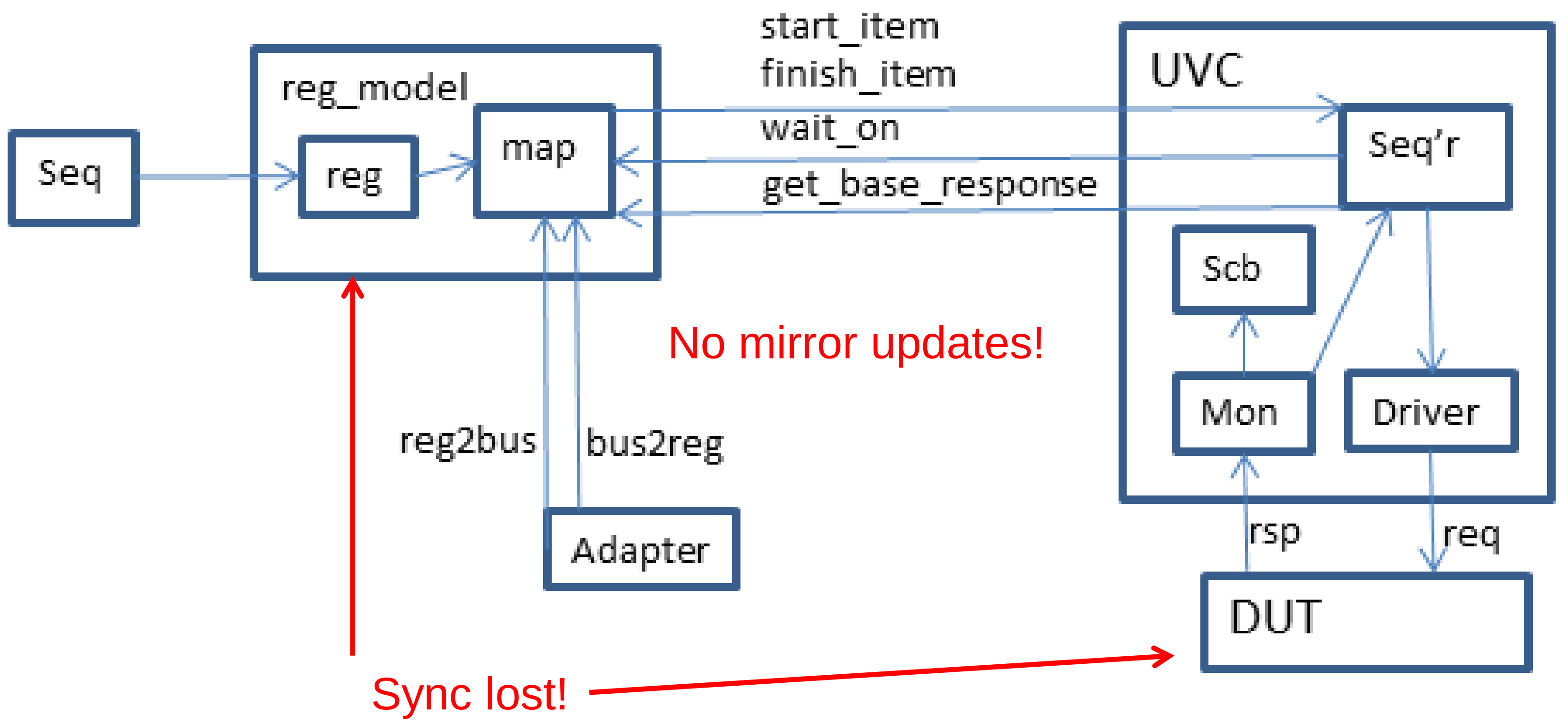
auto_predict = 0, no predictor



No mirror updates!

Loss of Mirror Sync Gotcha!

auto_predict = 0, no predictor



Predictor do_check

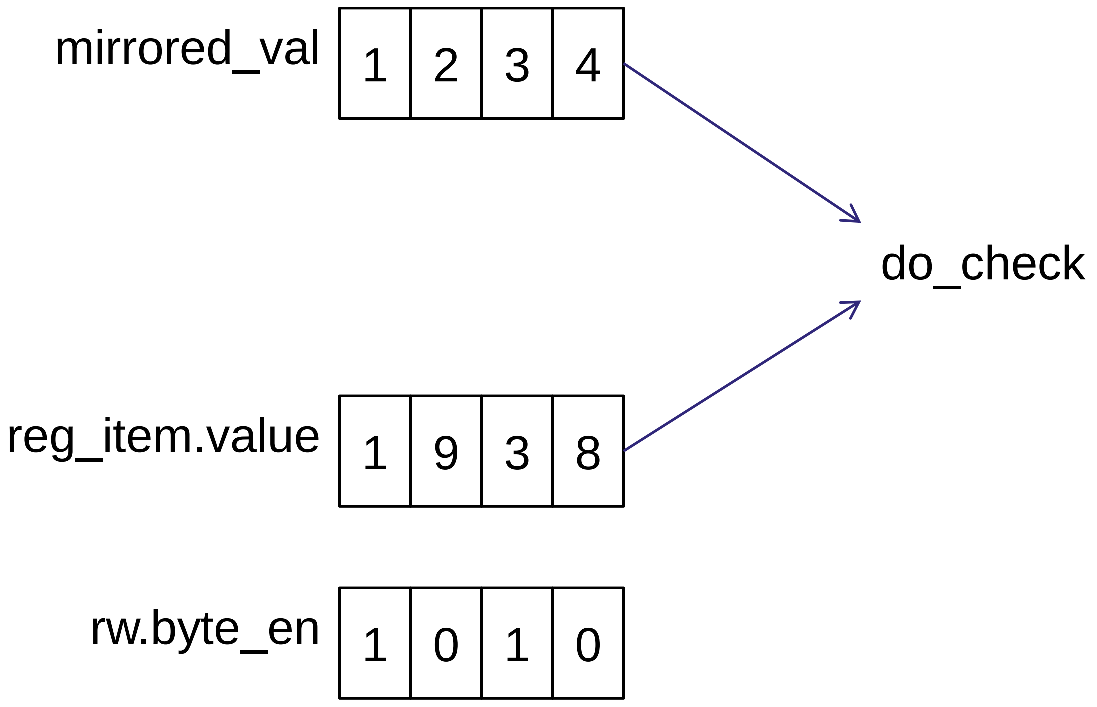
No byte enable sensitivity …

```
if (reg_item.kind == UVM_READ &&
    local_map.get_check_on_read() &&
    reg_item.status != UVM_NOT_OK) begin
   void'(rg.do_check(ir.get_mirrored_value(),
                     reg_item.value[0], local_map));
end
```

No byte enable input!

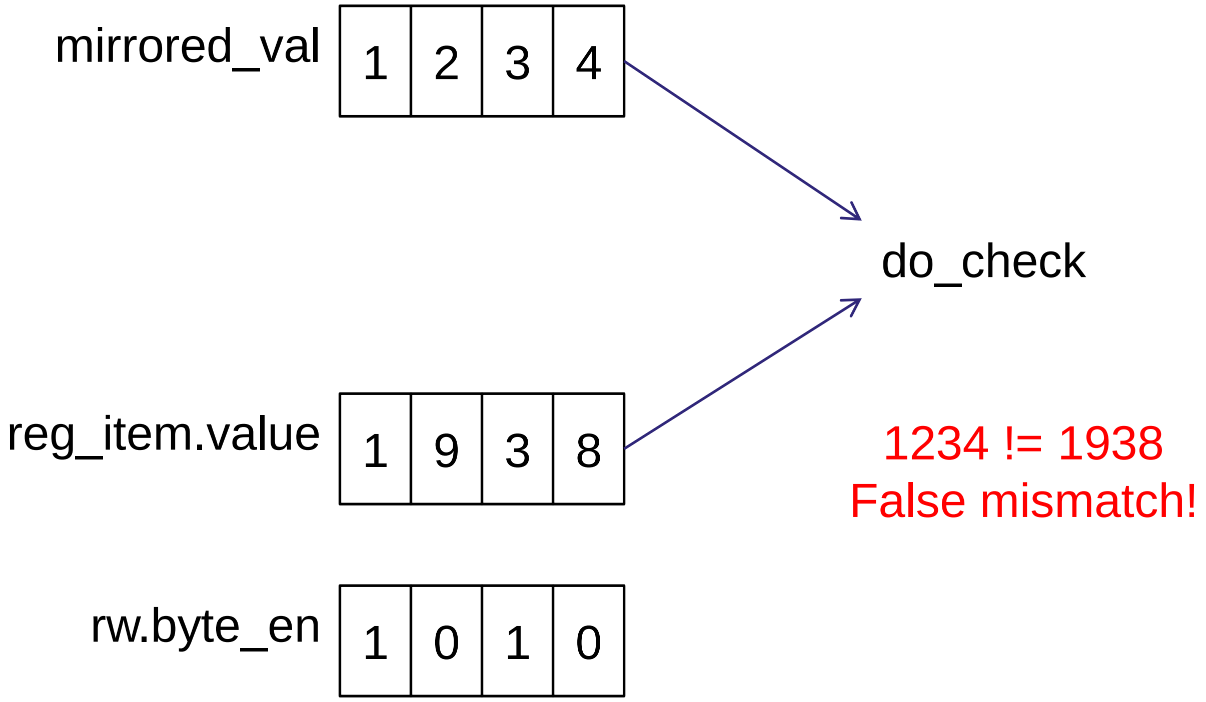
Predictor do_check

supports_byte_enable = 1 …



Predictor do_check Gotcha!

supports_byte_enable = 1 …



Byte Enable Checking

supports_byte_enable = 1

```
// Limit check per rw.byte_en.

exp_val = rg.get_mirrored_value(); // start with mirrored bytes

for(int bi = 0; bi < `MY_BYTENABLE_WIDTH; bi++) begin
  if(!rw.byte_en[bi]) begin
    // Set this expected byte value to reg_item.value[0]
    exp_val[(bi*8)+7 -: 8] = reg_item.value[0][(bi*8)+7 -: 8];
  end
end

void'(rg.do_check(exp_val, reg_item.value[0], local_map));
```

Byte Enable Checking

supports_byte_enable = 1

```
// Limit check per rw.byte_en.

exp_val = rg.get_mirrored_value(); // start with mirrored bytes

for(int bi = 0; bi < `MY_BYTENABLE_WIDTH; bi++) begin
  if(!rw.byte_en[bi]) begin
    // Set this expected byte value to reg_item.value[0]
    exp_val[(bi*8)+7 -: 8] = reg_item.value[0][(bi*8)+7 -: 8];
  end
end

void'(rg.do_check(exp_val, reg_item.value[0], local_map));
```

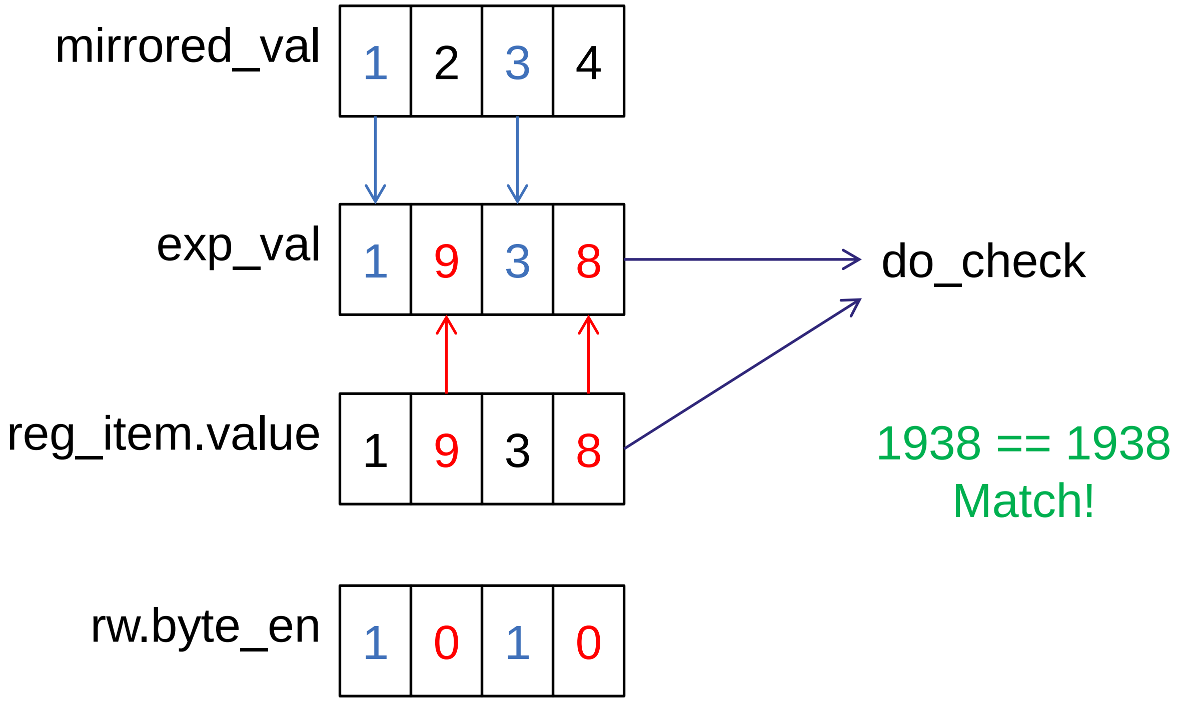
Byte Enable Checking

supports_byte_enable = 1

```
// Limit check per rw.byte_en.

exp_val = rg.get_mirrored_value(); // start with mirrored bytes

for(int bi = 0; bi < `MY_BYTENABLE_WIDTH; bi++) begin
  if(!rw.byte_en[bi]) begin
    // Set this expected byte value to reg_item.value[0]
    exp_val[(bi*8)+7 -: 8] = reg_item.value[0][(bi*8)+7 -: 8];
  end
end

void'(rg.do_check(exp_val, reg_item.value[0], local_map));
```

Predictor do_check

Copy reg_item to exp_val per !rw.byte_en …



Predictor do_predict

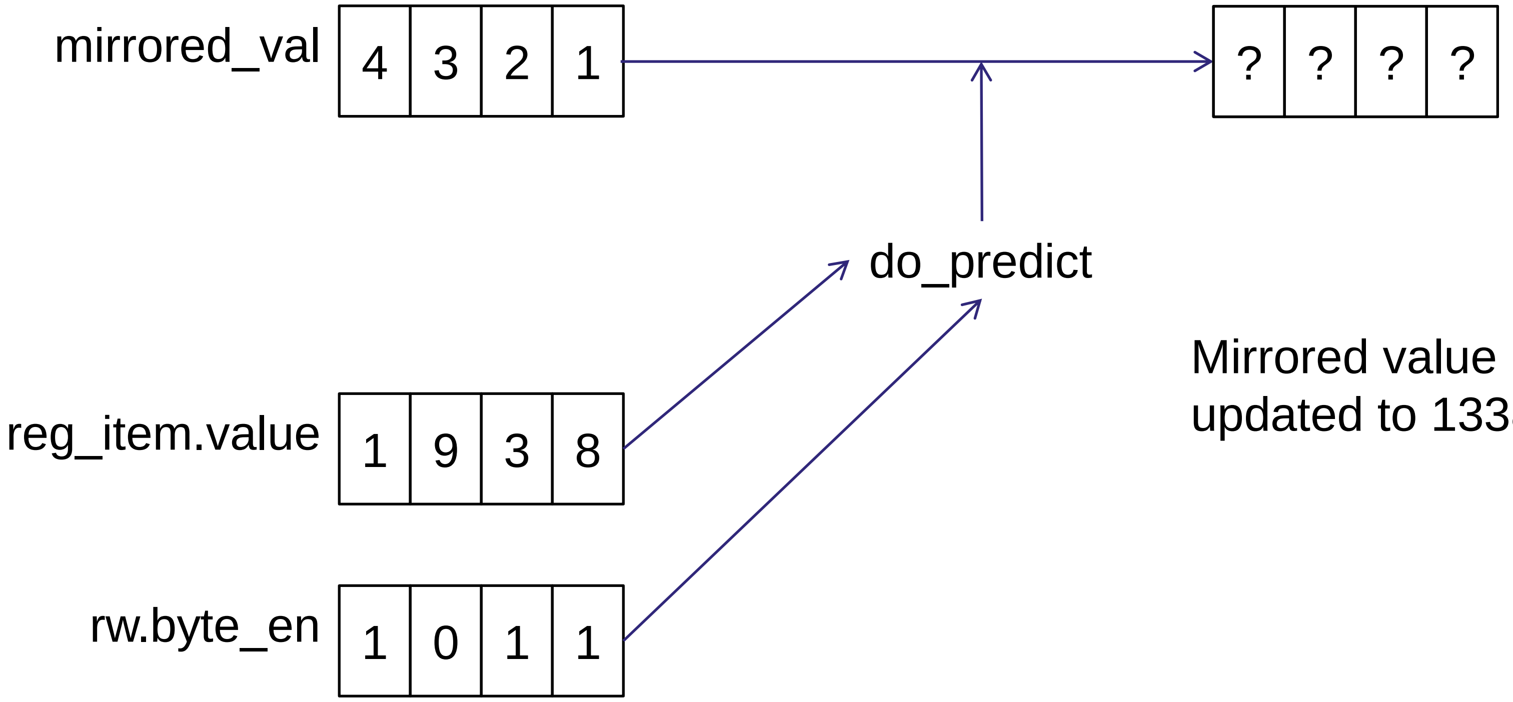
What happens with rw.byte_en (single, 32-bit field)?

```
rg.do_predict(reg_item, predict_kind, rw.byte_en);
```

Byte enable input!

Predictor do_predict

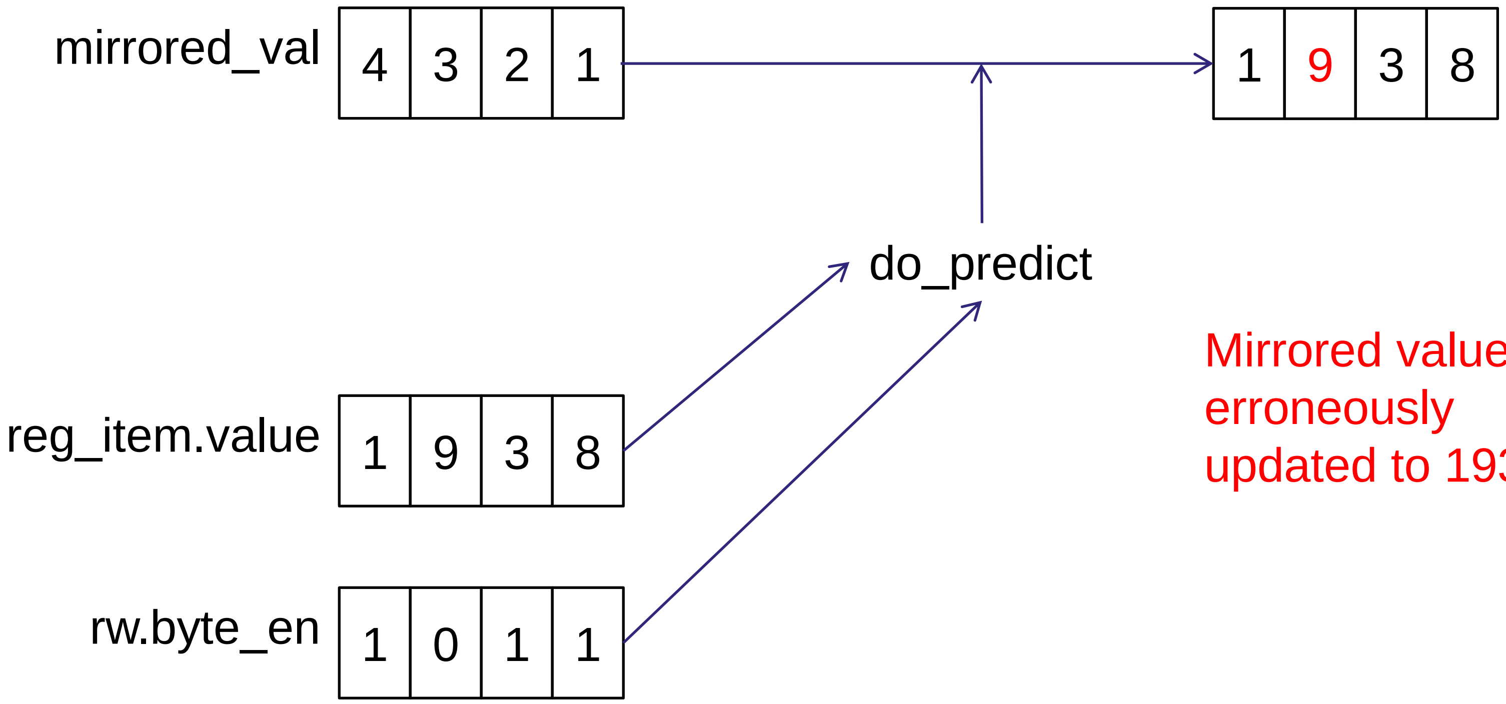
rw.byte_en == 1011 …



Mirrored value updated to 1338?

Predictor do_predict Gotcha!

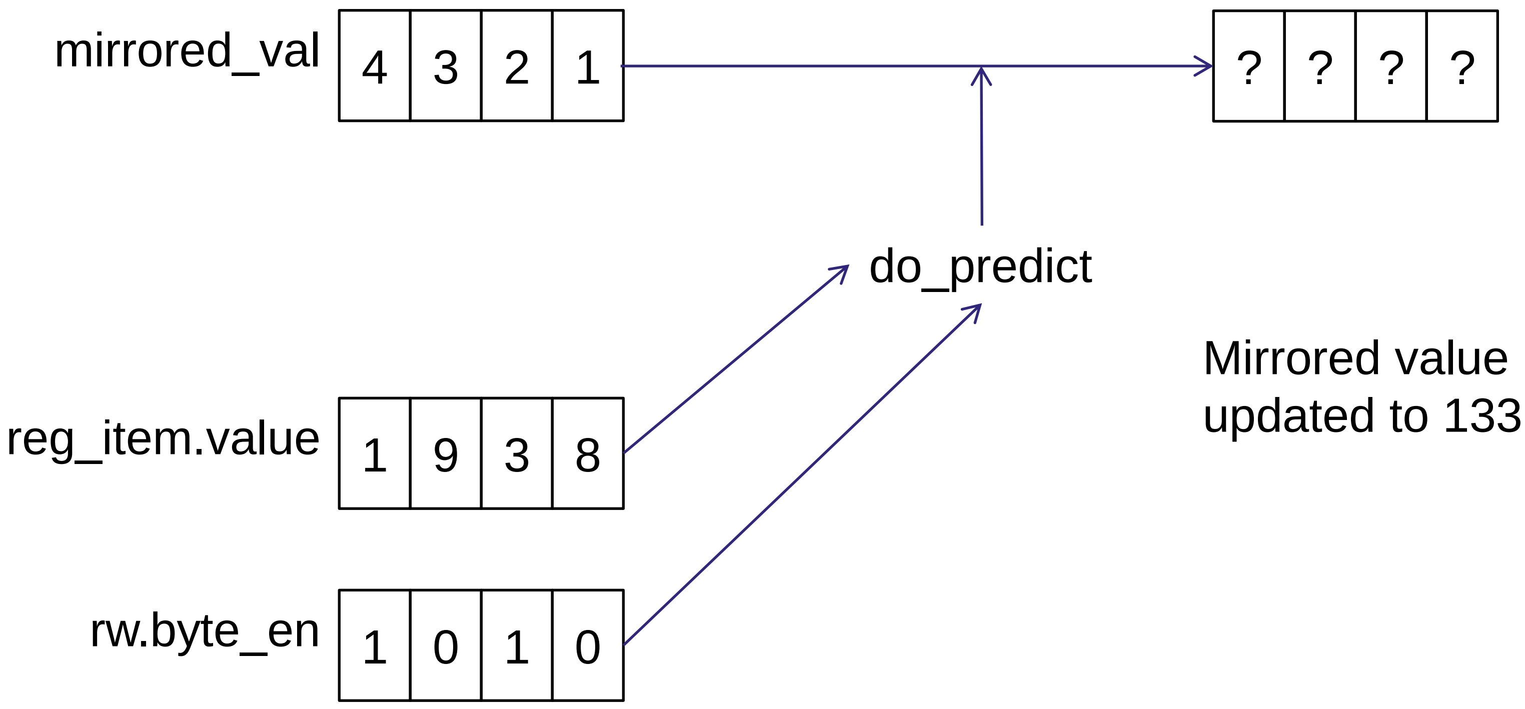
rw.byte_en == 1011 …



Mirrored value erroneously updated to 1938!

Predictor do_predict

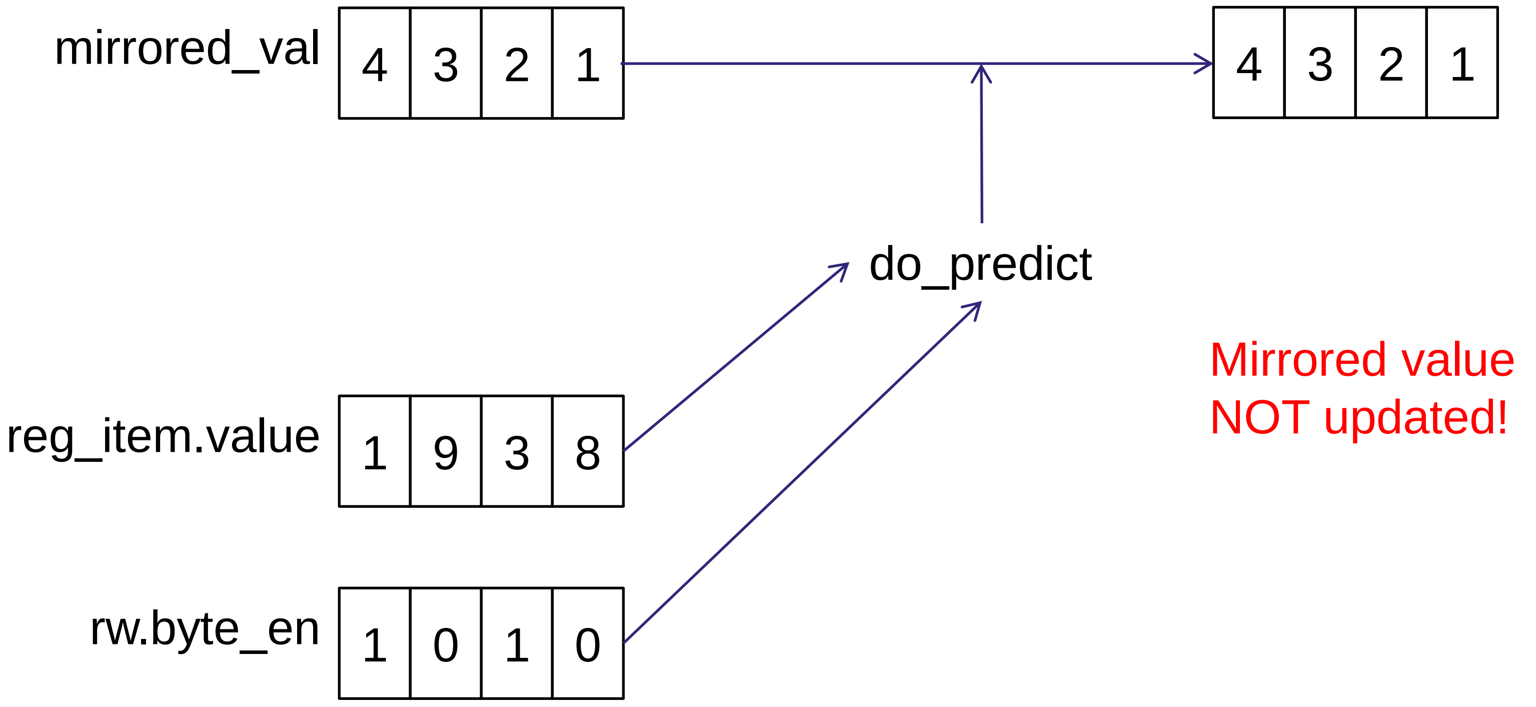
rw.byte_en == 1010 …



Mirrored value updated to 1331?

Predictor do_predict Gotcha!

rw.byte_en == 1010 …



Mirrored value
NOT updated!

uvm_reg do_predict

```
// do_predict

function void uvm_reg::do_predict(uvm_reg_item      rw,
                                  uvm_predict_e     kind = UVM_PREDICT_DIRECT,
                                  uvm_reg_byte_en_t be = -1);

   uvm_reg_data_t reg_value = rw.value[0];
   m_fname = rw.fname;
   m_lineno = rw.lineno;

if (rw.status ==UVM_IS_OK )
   rw.status = UVM_IS_OK;

   if (m_is_busy && kind == UVM_PREDICT_DIRECT) begin
      `uvm_warning("RegModel", {"Trying to predict value of register '",
                  get_full_name(),"' while it is being accessed"})
      rw.status = UVM_NOT_OK;
      return;
   end

   foreach (m_fields[i]) begin
      rw.value[0] = (reg_value >> m_fields[i].get_lsb_pos()) &
                                 ((1 << m_fields[i].get_n_bits())-1);
      m_fields[i].do_predict(rw, kind, be>>(m_fields[i].get_lsb_pos()/8));
   end

   rw.value[0] = reg_value;

endfunction: do_predict
```

uvm_reg do_predict

```
// do_predict

function void uvm_reg::do_predict(uvm_reg_item      rw,
                                  uvm_predict_e     kind = UVM_PREDICT_DIRECT,
                                  uvm_reg_byte_en_t be = -1);

   uvm_reg_data_t reg_value = rw.value[0];
   m_fname = rw.fname;
   m_lineno = rw.lineno;

if (rw.status ==UVM_IS_OK )
   rw.status = UVM_IS_OK;

   if (m_is_busy && kind == UVM_PREDICT_DIRECT) begin
      `uvm_warning("RegModel", {"Trying to predict value of register '",
                  get_full_name(),"' while it is being accessed"})
      rw.status = UVM_NOT_OK;
      return;
   end

   foreach (m_fields[i]) begin
      rw.value[0] = (reg_value >> m_fields[i].get_lsb_pos()) &
                                 ((1 << m_fields[i].get_n_bits())-1);
      m_fields[i].do_predict(rw, kind, be>>(m_fields[i].get_lsb_pos()/8));
   end

   rw.value[0] = reg_value;

endfunction: do_predict
```

uvm_reg_field do_predict

Entire field is skipped (be[0] == 0)!

```
// do_predict

function void uvm_reg_field::do_predict(uvm_reg_item      rw,
                                        uvm_predict_e     kind = UVM_PREDICT_DIRECT,
                                        uvm_reg_byte_en_t be = -1);

   uvm_reg_data_t field_val = rw.value[0] & ((1 << m_size)-1);

   if (rw.status != UVM_NOT_OK)
     rw.status = UVM_IS_OK;

   // Assume that the entire field is enabled
   if (!be[0])
     return;
```

Byte Enable Predicting

```
// Limit predict per rw.byte_en.

exp_val = rg.get_mirrored_value(); // start with mirrored bytes

for(int bi = 0; bi < `MY_BYTENABLE_WIDTH; bi++) begin
  if(rw.byte_en[bi]) begin
    // Set this expected byte value to reg_item.value[0]
    exp_val[(bi*8)+7 -: 8] = reg_item.value[0][(bi*8)+7 -: 8];
  end
end

// Modify reg_item.value[0] and enable all bytes for do_predict.

reg_item.value[0] = exp_val;
rw.byte_en = -1;

rg.do_predict(reg_item, predict_kind, rw.byte_en);
```

Byte Enable Predicting

```
// Limit predict per rw.byte_en.

exp_val = rg.get_mirrored_value(); // start with mirrored bytes

for(int bi = 0; bi < `MY_BYTENABLE_WIDTH; bi++) begin
  if(rw.byte_en[bi]) begin
    // Set this expected byte value to reg_item.value[0]
    exp_val[(bi*8)+7 -: 8] = reg_item.value[0][(bi*8)+7 -: 8];
  end
end

// Modify reg_item.value[0] and enable all bytes for do_predict.

reg_item.value[0] = exp_val;
rw.byte_en = -1;

rg.do_predict(reg_item, predict_kind, rw.byte_en);
```

Byte Enable Predicting

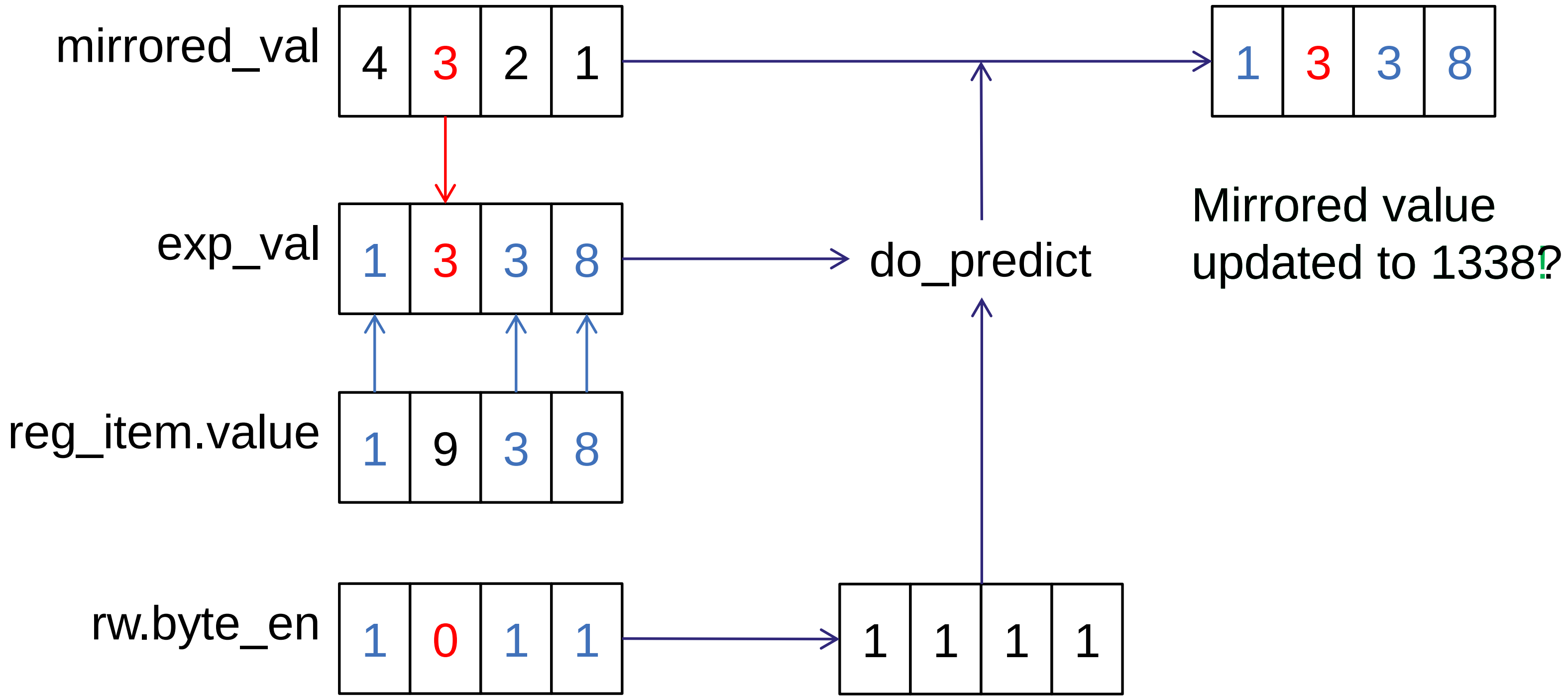
```
// Limit predict per rw.byte_en.

exp_val = rg.get_mirrored_value(); // start with mirrored bytes

for(int bi = 0; bi < `MY_BYTENABLE_WIDTH; bi++) begin
  if(rw.byte_en[bi]) begin
    // Set this expected byte value to reg_item.value[0]
    exp_val[(bi*8)+7 -: 8] = reg_item.value[0][(bi*8)+7 -: 8];
  end
end

// Modify reg_item.value[0] and enable all bytes for do_predict.

reg_item.value[0] = exp_val;
rw.byte_en = -1;

rg.do_predict(reg_item, predict_kind, rw.byte_en);
```

Byte Enable Predicting

```
// Limit predict per rw.byte_en.

exp_val = rg.get_mirrored_value(); // start with mirrored bytes

for(int bi = 0; bi < `MY_BYTENABLE_WIDTH; bi++) begin
  if(rw.byte_en[bi]) begin
    // Set this expected byte value to reg_item.value[0]
    exp_val[(bi*8)+7 -: 8] = reg_item.value[0][(bi*8)+7 -: 8];
  end
end

// Modify reg_item.value[0] and enable all bytes for do_predict.

reg_item.value[0] = exp_val;
rw.byte_en = -1;

rg.do_predict(reg_item, predict_kind, rw.byte_en);
```

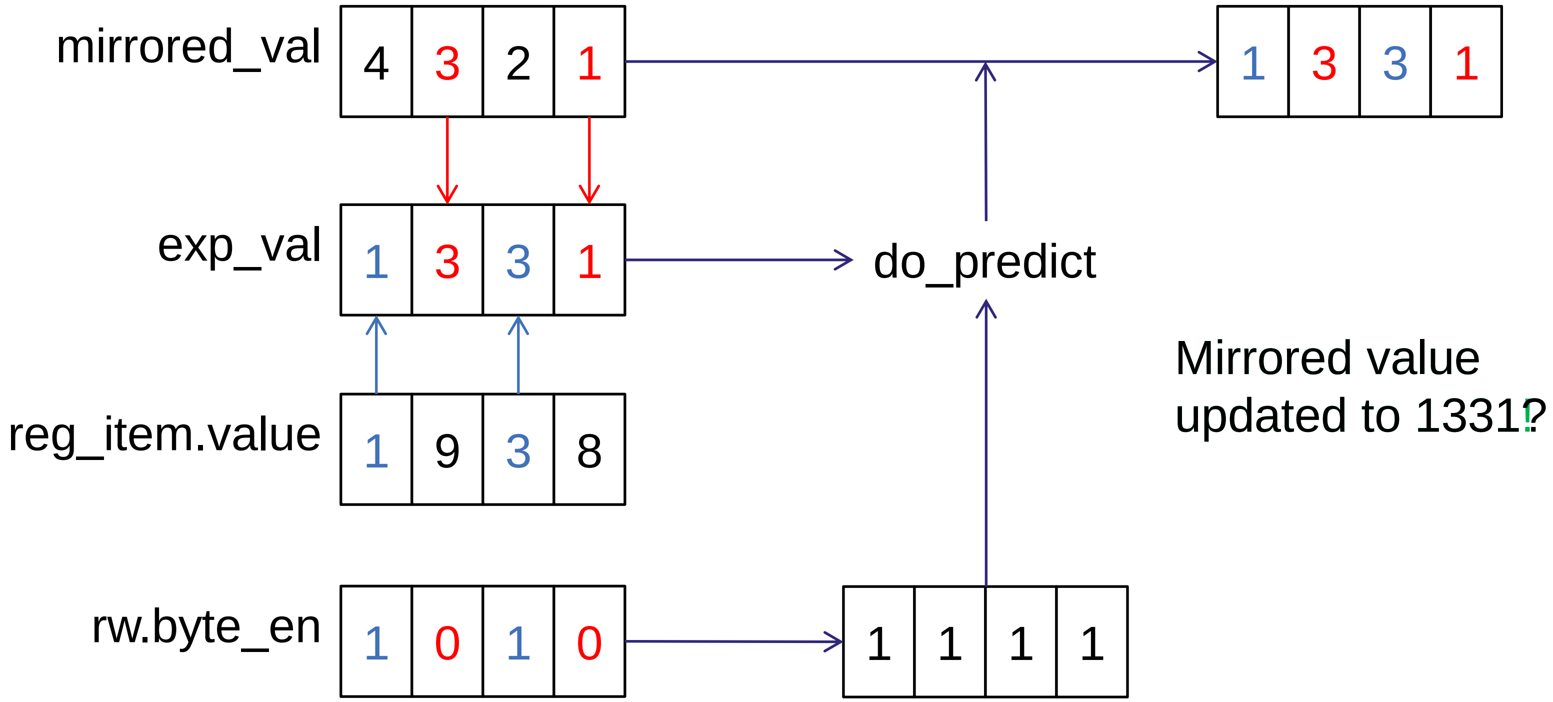
Predictor do_predict

Copy reg_item to exp_val per rw.byte_en == 1011 …

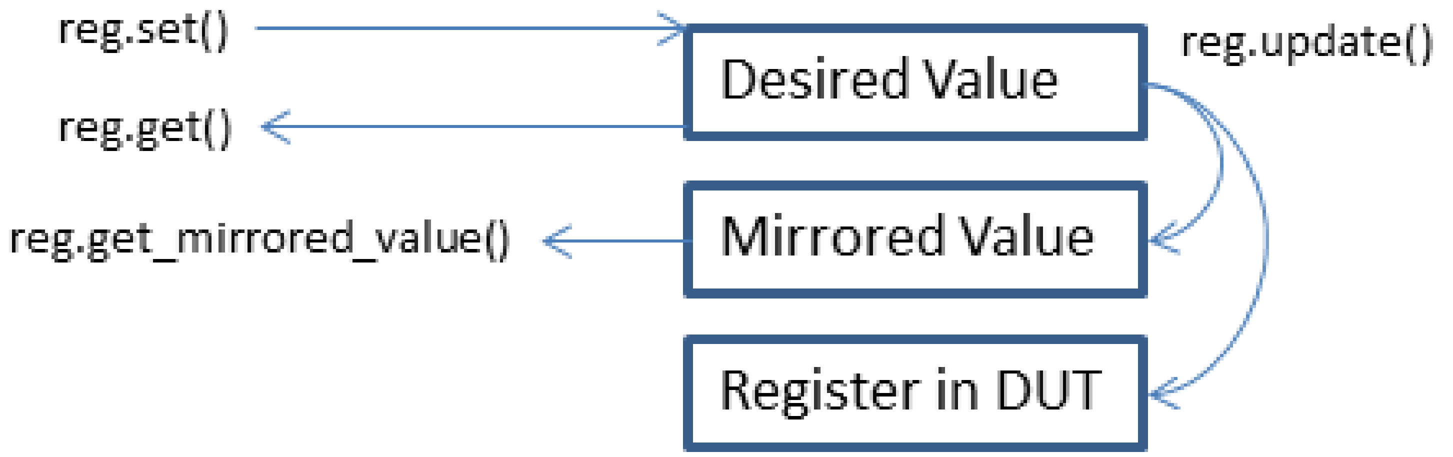


Predictor do_predict

Copy reg_item to exp_val per rw.byte_en == 1010 …

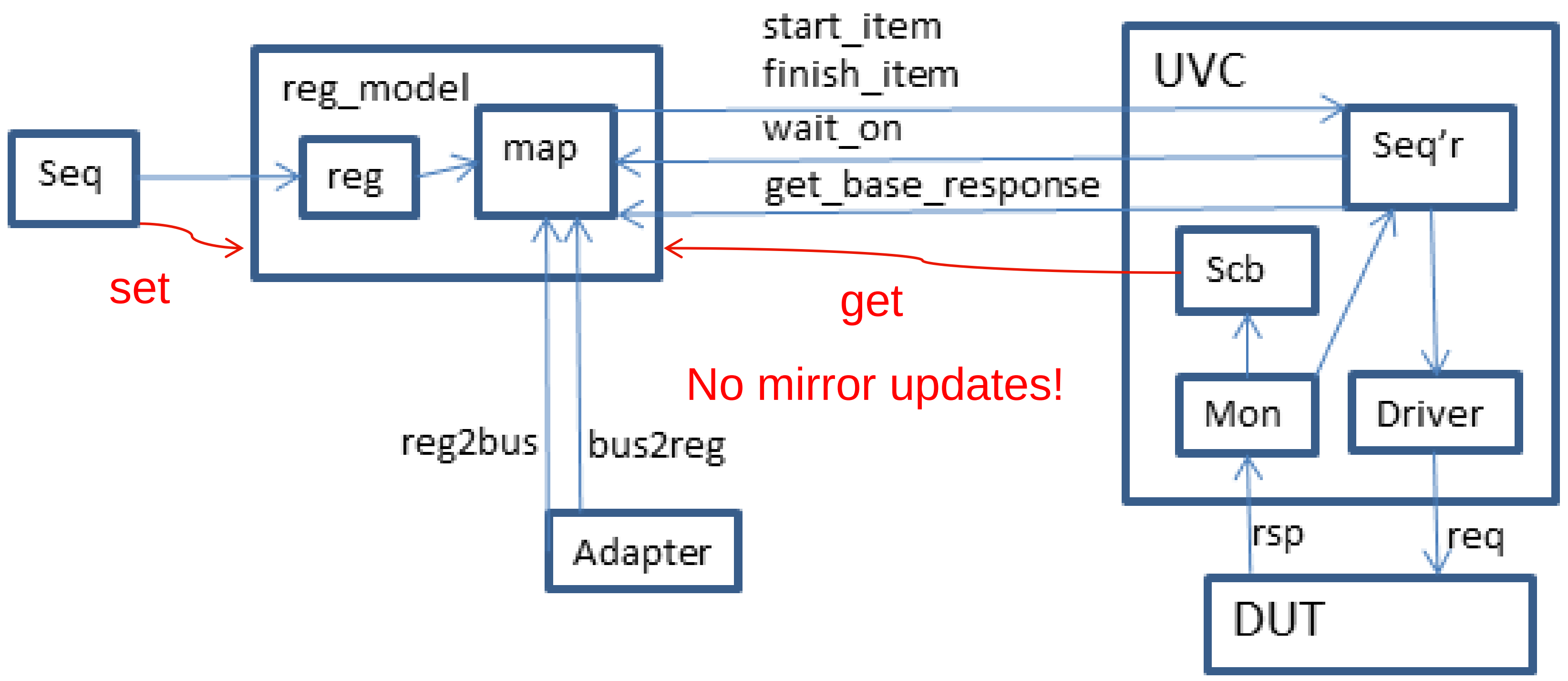


“get” vs “get_mirrored_value”



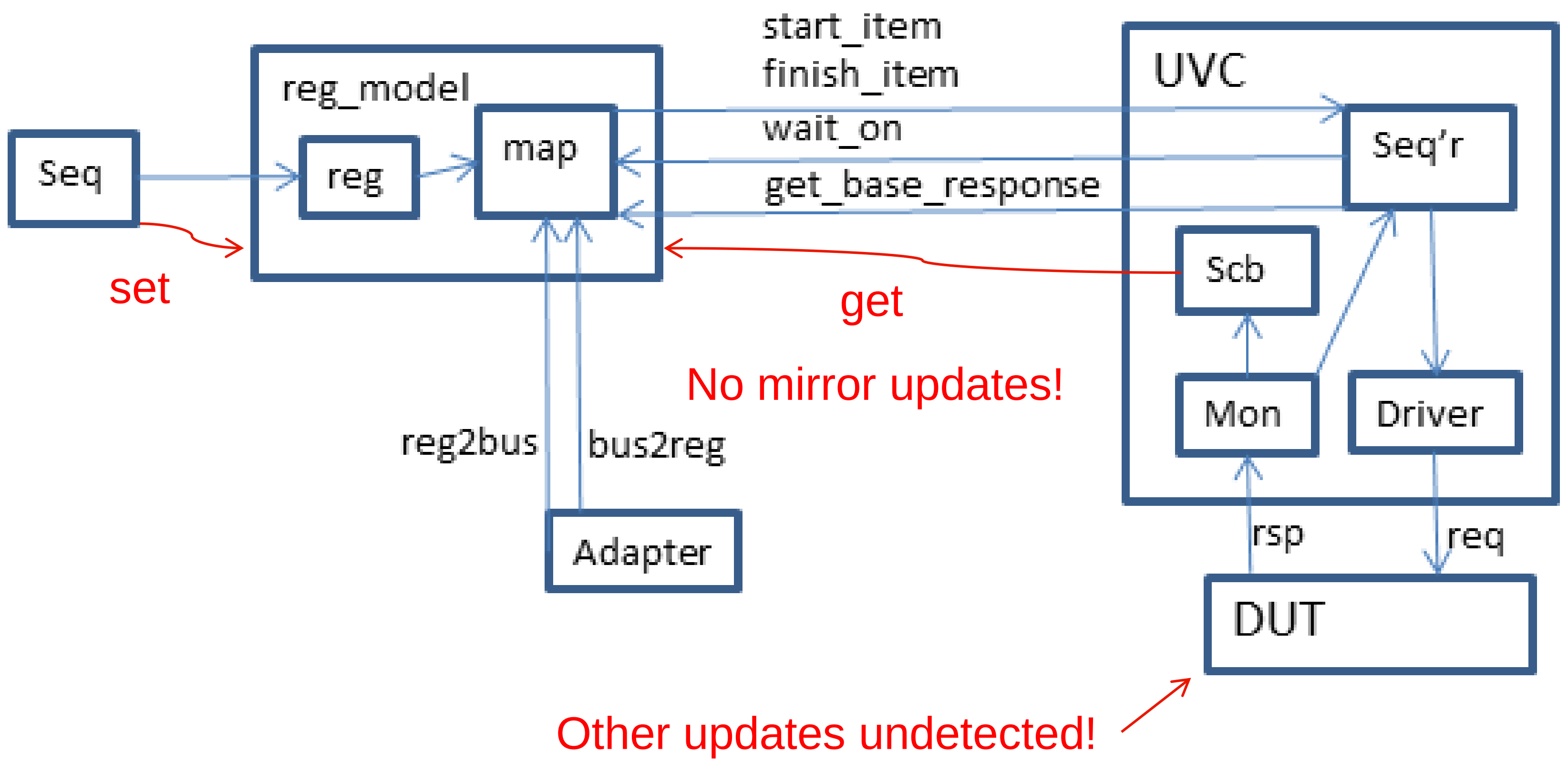
Loss of Mirror Sync (again)

auto_predict = ?, no predictor

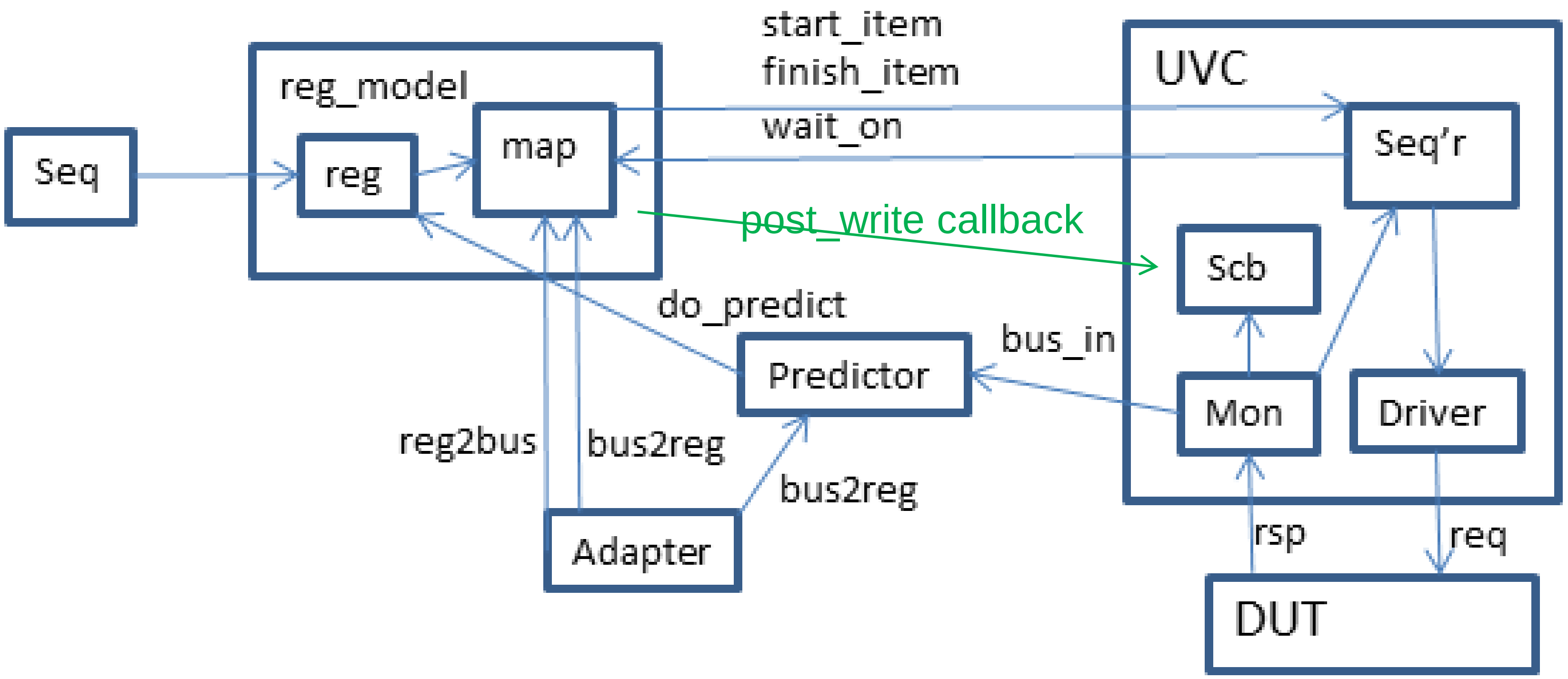


Loss of Mirror Sync (again) Gotcha!

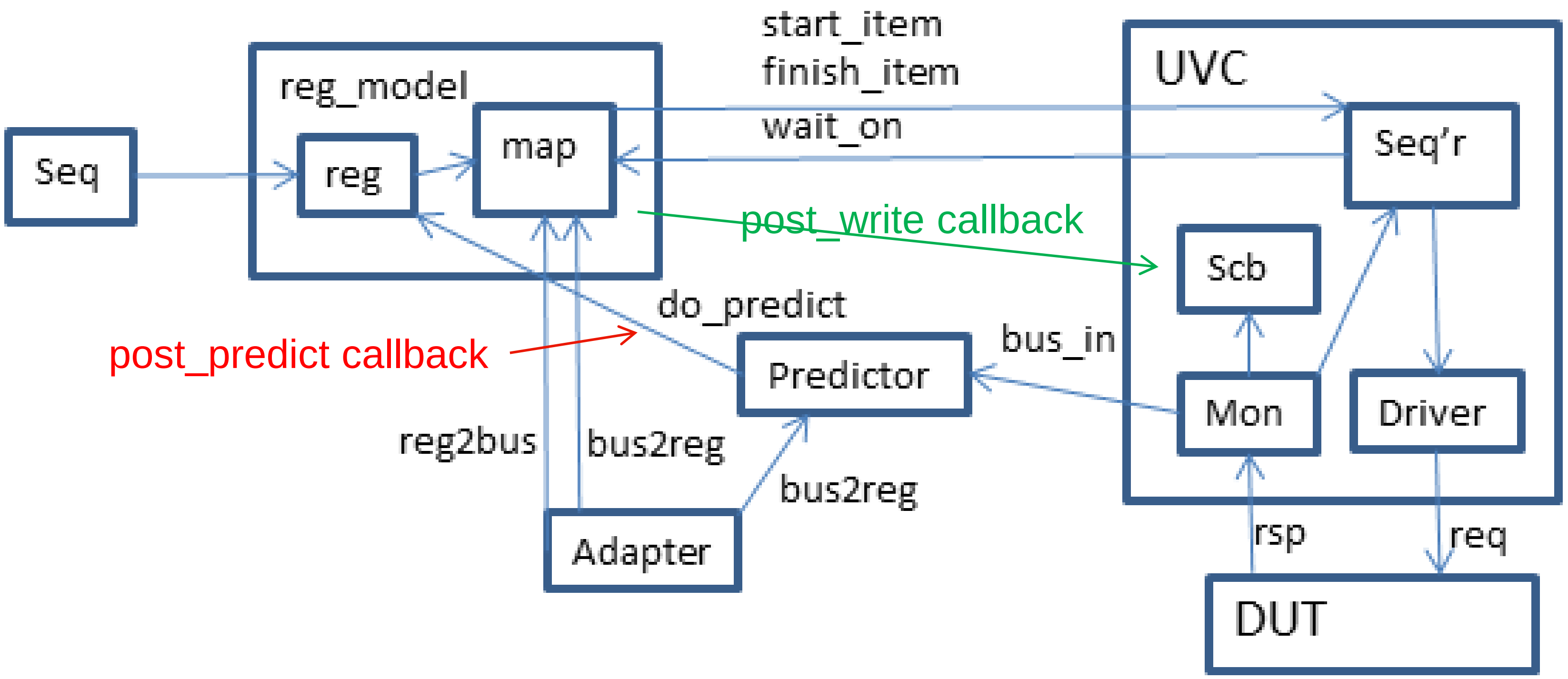
auto_predict = ?, no predictor



post_write Callbacks



post_write Callbacks Gotcha!



post_write Callbacks

Predictor calls do_callbacks

```
function void do_callbacks(uvm_reg reg_obj,
                           uvm_reg_item rw);

  // get the callback iterator
  uvm_reg_cb_iter cbs = new(reg_obj);

  // If no callbacks, then return.

  if(cbs.first() == null) begin
    return;
  end
```

post_write Callbacks

Predictor calls do_callbacks

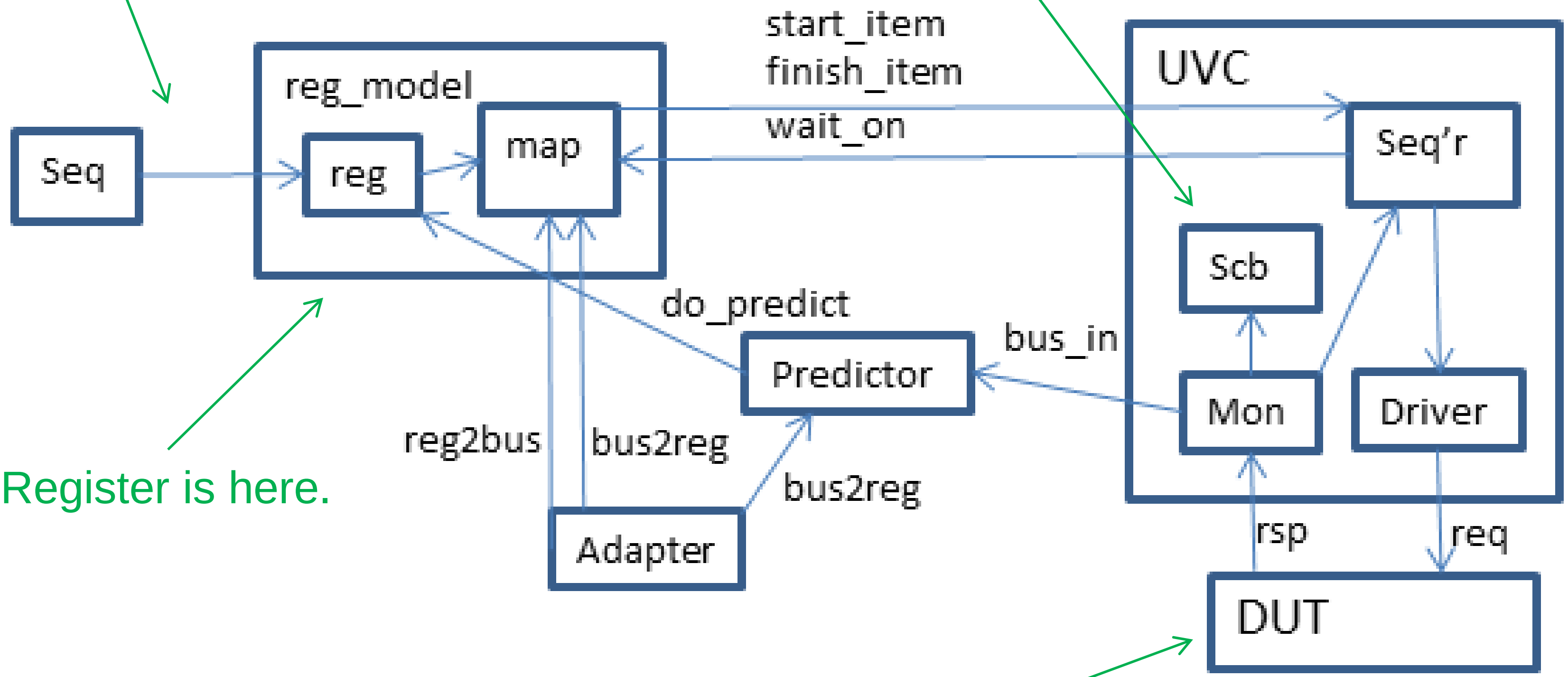
```
fork: cb_thread_001
  begin
    automatic uvm_reg_item rwf = new;
    automatic uvm_reg reg_objf;
    automatic uvm_reg_cb_iter cbsf;

    // preserve current variables
    reg_objf = reg_obj;
    cbsf = new(reg_objf);
    rwf.copy(rw);

    // process callbacks
    for (uvm_reg_cbs cb=cbsf.first();
         cb!=null; cb=cbsf.next()) begin
      if(cb != null) begin
        cb.post_write(rwf);
      end
    end

    reg_objf.post_write(rwf);
  end
join_none
```

Sequence Direct Mirror Update

Ensure post_write callback

Sequence updates mirror,

Scoreboard gets post_write callback.



Register is here.

Register is NOT here.
No actual traffic to register.
No predictor post_write.

Predictor Code Snippet

Address Filtering

```
if((tr.req._addr[47:0] >= 'hFFF1_2340_0000) &&
   (tr.req._addr[47:0] <= 'hFFF1_2340_FFFF)) begin
  special_detected = 1;
end
else begin
  special_detected = 0;
end

adapter.bus2reg(tr,rw);
```

Predictor Code Snippet

Address Filtering

```
adapter.bus2reg(tr,rw);

checked = my_ral.my_check(rw.addr);

if(checked) begin
  int old_addr = rw.addr;

  // rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));
  rg = my_ral.get_rg_by_offset(map, rw.addr, special_detected);
```

Predictor Code Snippet

Special Checks, Indirect Register Access

```
adapter.bus2reg(tr,rw);

checked = my_ral.my_check(rw.addr);

if(checked) begin
  int old_addr = rw.addr;

  // rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));
  rg = my_ral.get_rg_by_offset(map, rw.addr, special_detected);
```

Predictor Code Snippet Gotcha!

Special Checks, Indirect Register Access

```
adapter.bus2reg(tr,rw);

checked = my_ral.my_check(rw.addr);

if(checked) begin
   int old_addr = rw.addr;

   // rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));
   rg = my_ral.get_rg_by_offset(map, rw.addr, special_detected);
```

Indirect access means rg is NOT at the address checked!

Predictor Code Snippet

Special Checks, Indirect Register Access

```
adapter.bus2reg(tr,rw);

checked = my_ral.my_check(rw.addr);

if(checked) begin
  int old_addr = rw.addr;

  // rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));
  rg = my_ral.get_rg_by_offset(map, rw.addr, special_detected);

  if(old_offset != rw.addr) begin
    checked = my_ral.my_check(rw.addr);
```

Conclusion

Conclusion

- Options
  - provides_responses (bus driver, predictor)
  - supports_byte_enable (byte enable issues)
  - auto_predict (set with predictors)
- Extension object (functions, info for adapter)
- Mirrored 🙂 vs desired 😟
- post_write callbacks for do_predict
- get_rg_by_offset (reuse in predictors, indirect access)
- Address filtering (before bus2reg)
- Indirect register address double-check

Conclusion

- Adapter/Predictor customization challenges:
  - A handful of critical, large development projects …
  - Dozens of testbenches …
  - Nearly a half-dozen sets of adapters and predictors …
  - A wide variety of testbench variants …

And, after all this …

Conclusion

Mirrored register values are

IN SYNC with the DUT!

AMD

snug
Synopsys Users Group

Thank You