Leveraging the UVM RAL for Memory Sub-System Verification

Tudor Timisescu, Infineon

Uwe Simm, Cadence

accellera SYSTEMS INITIATIVE

Infineon

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Intro

- Storage elements are ubiquitous

- UVM REG
  - register abstraction layer
  - programmer's view
  - integrates modeling, checking, stimulus generation and (some) coverage

- Registers vs. memories
  - *read(…)* vs. *burst_read(…)*

Why Layer

- Common language when writing tests

Vă vorbesc despre memorii.

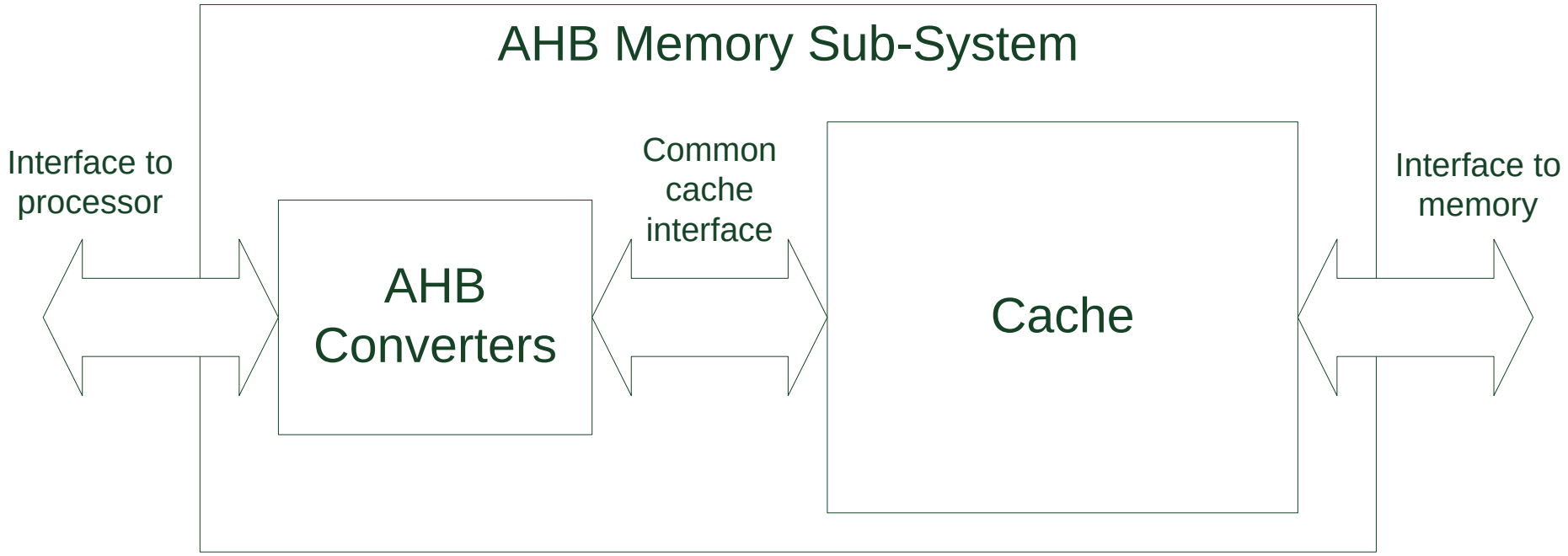
I'm talking about memories.

Ich spreche über Arbeitsspeicher.

- Reuse sequences
  - Abstract and conquer

* Images from http://polandball.wikia.com (licensed under Creative Commons).

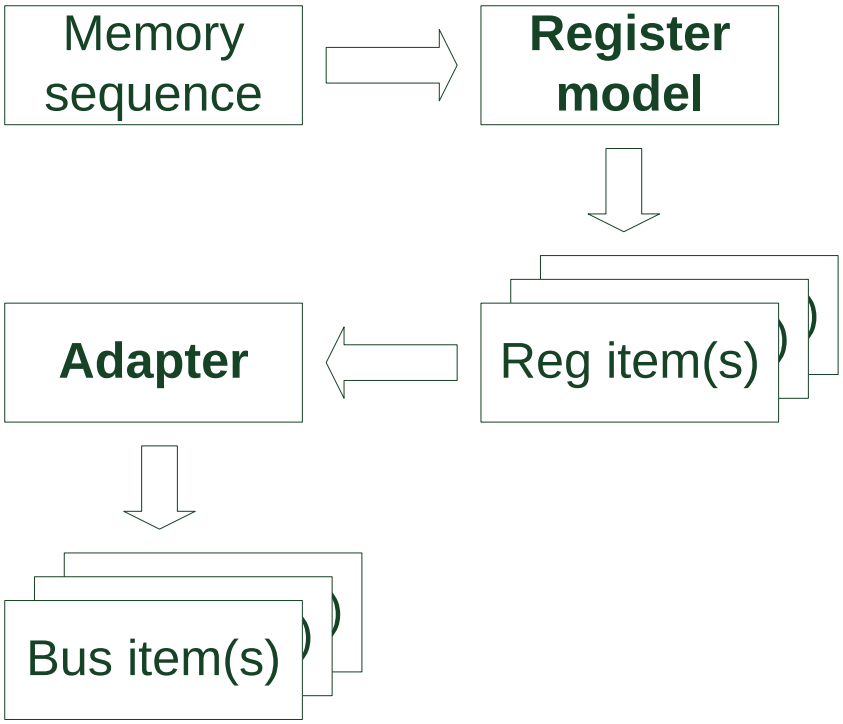
Real Life Example



- Vertical reuse (cache verified standalone)

Adapter

- register operation <-> bus item
- map chops bursts into multiple operations
  - deterministic algorithm

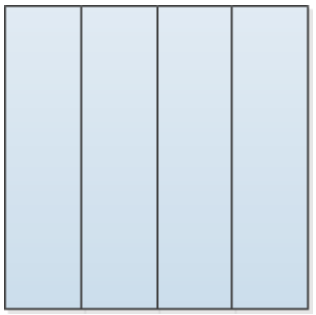


Adapter (cont'd)

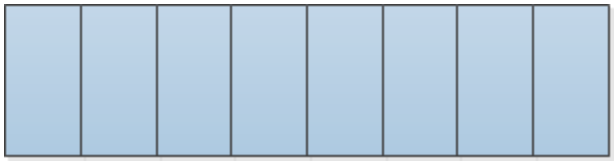
```
class ifx_ahb_reg_adapter extends uvm_reg_adapter;
  // …

  virtual function uvm_sequence_item reg2bus(const ref uvm_reg_bus_op rw);
    ifx_ahb_seq_item item = ifx_ahb_seq_item::type_id::create("item");

    if (!item.randomize() with {
      item.burst_kind == SINGLE;
      item.size == (rw.n_bits == 8 ? BYTE : (rw.n_bits == 16 ? HALFWORD : WORD));
      item.direction == rw.kind == UVM_WRITE ? WRITE : READ;
      item.addr == rw.addr;
      item.data == rw.data;
    })
      `uvm_error("RANDERR", "Randomization error")

    return item;
  endfunction
endclass
```

Burst Example

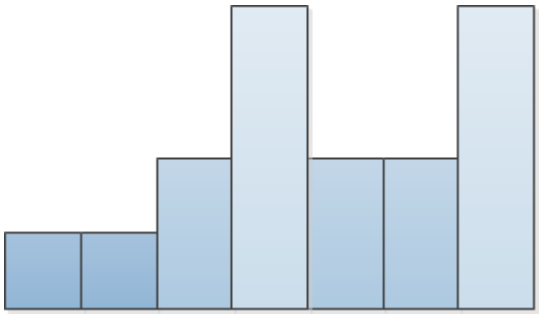
- *mem.burst_write(…)* of 16 bytes on 32 bit AHB bus



4 WORDs



8 HALFWORDs



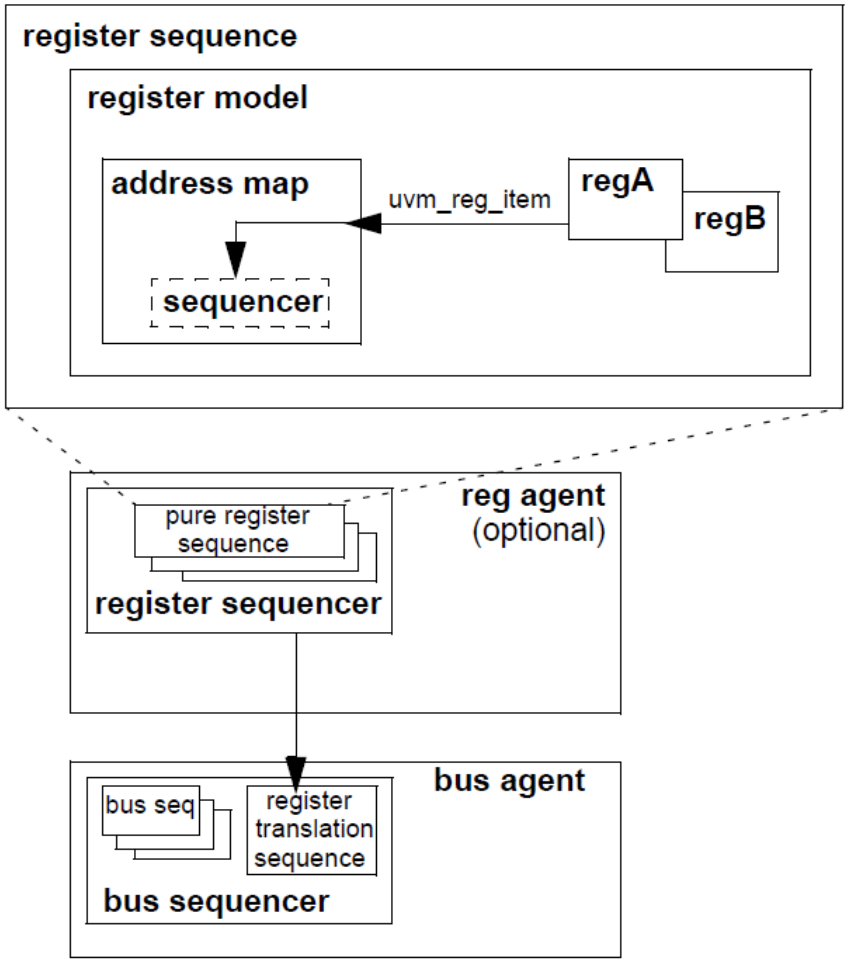
Mixture of sizes

- How many possibilities?

Requirements

- Custom translation scheme
  - Works at burst level
- Seamless integration
- Multiple DUT interfaces

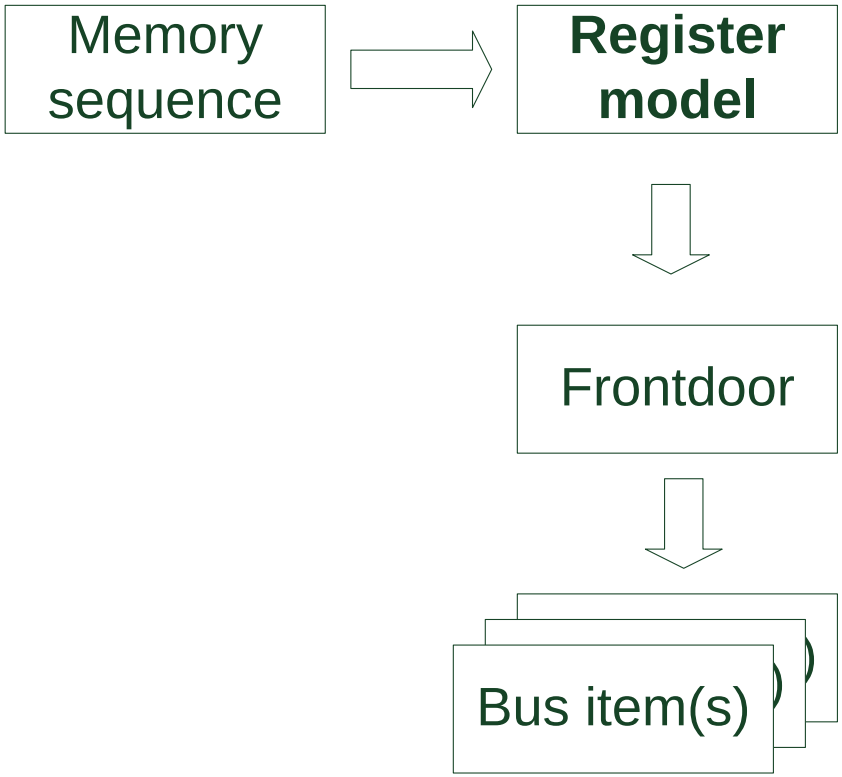
- Register sequencer
  - Only one interface
  - Cumbersome
  - Not versatile



* Image from the UVM 1.1 User's Guide.

Frontdoor

- briefly touched upon in the UVM user guide
- sequence
- intended to implement custom addressing schemes
- can be (mis)used for translation



Frontdoor (cont'd)

```
class ifx_ahb_mem_frontdoor extends uvm_reg_frontdoor;
  // ...

  virtual task body();
    bit[31:0] offset = rw_info.offset;
    int unsigned num_bytes = rw_info.value.size() * 4;

    while (num_bytes) begin
      ifx_ahb_seq_item burst = get_next_burst(offset, num_bytes);

      start_item(burst);
      finish_item(burst);

      num_bytes -= burst.get_num_bytes();
      offset += burst.get_num_bytes();
    end
  endtask
endclass
```

Keep score of transferred bytes

Move to next block

Frontdoor (cont'd)

```
class ifx_ahb_mem_frontdoor extends uvm_reg_frontdoor;
  // ...

  protected function ifx_ahb_seq_item get_next_burst(bit[31:0] offset,
    int unsigned num_bytes
  );
    ifx_ahb_seq_item burst;
    `uvm_create(burst);

    if (!burst.randomize() with {
      addr == offset;
      direction == rw_info.kind inside { UVM_READ, UVM_BURST_READ } ? READ : WRITE;
      size inside { BYTE, HALFWORD, WORD };
      (2 ** size) * burst_len <= num_bytes;
    })
      `uvm_fatal("RANDERR", "Randomization error")

    return burst;
  endfunction
endclass
```

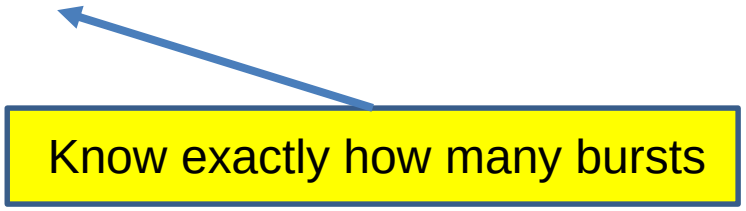
Constrain AHB burst

Constraining Randomization

- Guide randomization
  - E.g. not all burst kinds implemented
- Non-intrusive changes
  - Declare extra constraints in sub-classes
  - Polymorphism
  - UVM factory

Override 1

```
class ahb_frontdoor_single extends ifx_ahb_mem_frontdoor;
  // ...

  virtual task body();
    ifx_ahb_seq_item items[] = new[rw_info.value.size()];
    foreach (items[i]) begin
      `uvm_create(items[i])
      if (!items[i].randomize() with {
        burst_kind == SINGLE;
        size == WORD;
        // ...
      })
        `uvm_fatal("RANDERR", "Randomization error")
      `uvm_send(items[i]);
    end
  endtask
endclass

// ...
ifx_ahb_mem_frontdoor::type_id::set_type_override(
  ahb_frontdoor_single::get_type());
```



Override 2

```
class single_word_ahb_seq_item extends ifx_ahb_seq_item;
  // ...

  constraint single_word {
    burst_kind == SINGLE;
    size == WORD;
  }
endclass


// ...
ifx_ahb_seq_item::type_id::set_type_override(
  single_word_ahb_seq_item::get_type());
```

Conclusion

- Flexible
  - Registers can use adapter, memories can use frontdoor
  - Write once, tweak everywhere
  - Plays nice with native bus sequences
- Practical
  - Abstract
  - Stimulus reuse

Questions