YAMM
Yet Another Memory Manager

Andrei Vintila, Ionut Tolea

Amiq Consulting

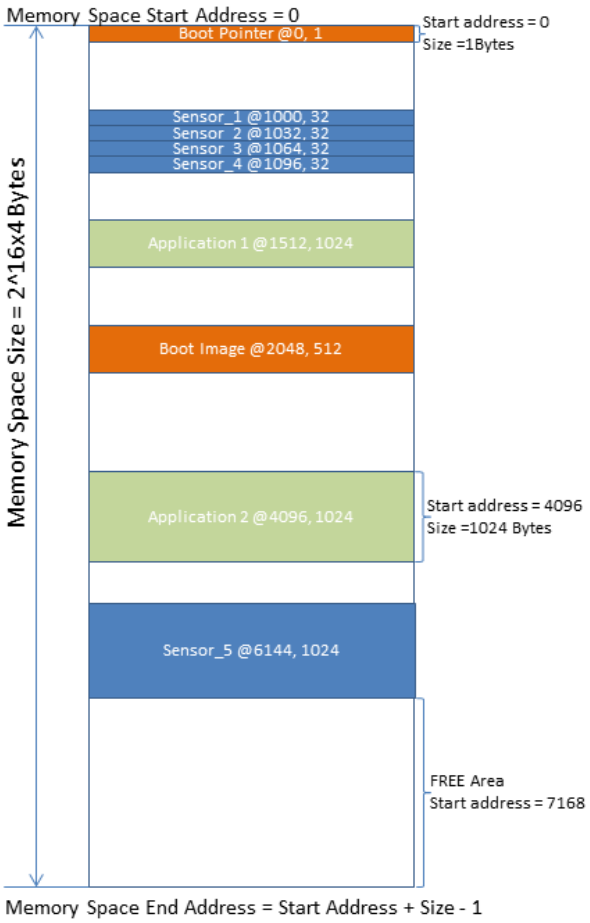
accellera SYSTEMS INITIATIVE

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Theory
  - Memory Management Introduction
  - YAMM Overview: Features, Algorithm, Datatypes, API
- Comparison with UVM_MAM
  - Feature-wise
  - Performance-wise
- Examples

Memory Management – What is it?



Memory Management Requirements

Real Life	Verification
Provide memory buffers to programs	Support real life use cases
Prevent memory corruption	Provide randomization support
Reduce fragmentation	Provide debug support
	Not necessarily a memory model

YAMM Features (1)

- Implementations for SystemVerilog, C++
- Everything is a buffer
- Provides API for retrieval of allocated buffers
- Supports multiple allocation modes
- Supports memory granularity and address alignment
- Provides buffer content management

YAMM Features (2)

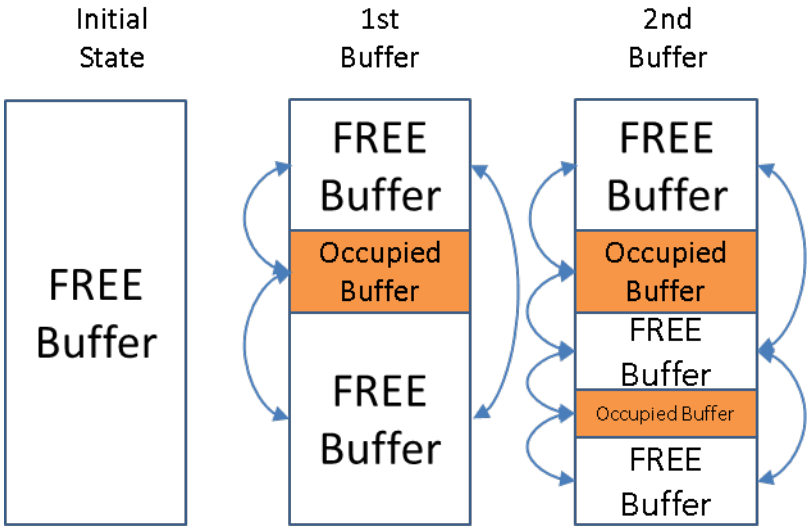
- Buffers represent address spaces themselves
- Memory can be written to file or pretty-printed
- Provides usage and fragmentation statistics
- Can be easily extended for specific use cases

YAMM Data Types

- yamm_buffer
  - Contains all data and functions
- yamm
  - Top level instance of the memory manager
  - Inherits yamm_buffer
- yamm_access
  - Optional usage
  - Used to model a basic access (start address, size)

YAMM Overview - Algorithm

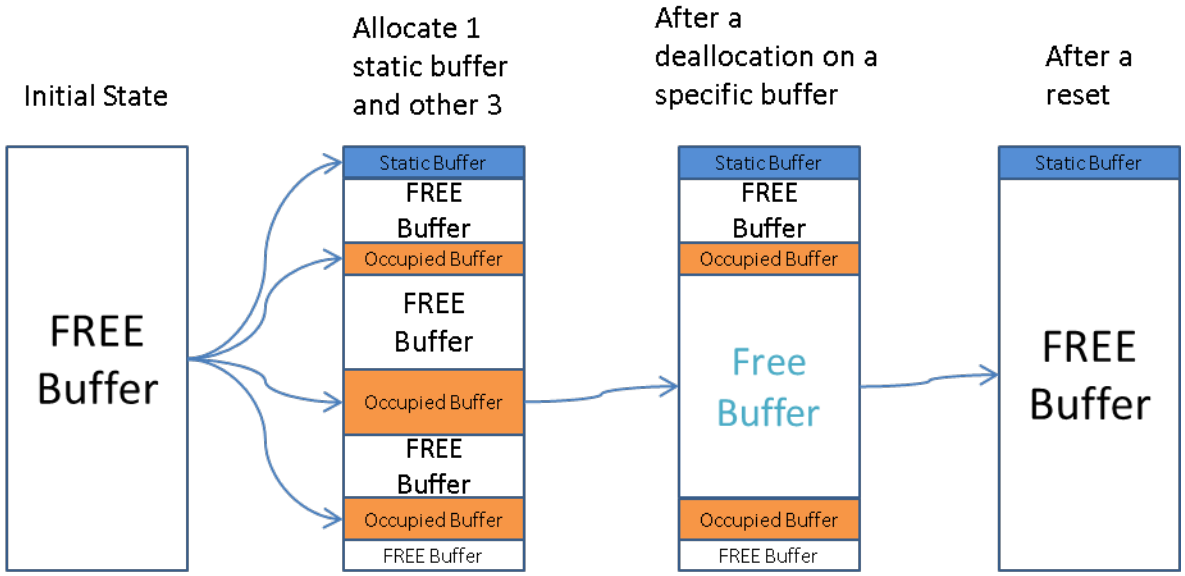
- After initialization the memory map will contain a single FREE buffer
- All buffers in a memory map are chained in a double linked list



YAMM Overview – API (1)

- Allocation
  - Manual allocation: insertion
  - Automatic allocation rules:
    - RANDOM_FIT
    - FIRST_FIT (and FIRST_FIT_RND)
    - BEST_FIT (and BEST_FIT_RND)
    - UNIFORM_FIT
- Deallocation
  - It can be done on a specific buffer or address

YAMM Overview – API (2)

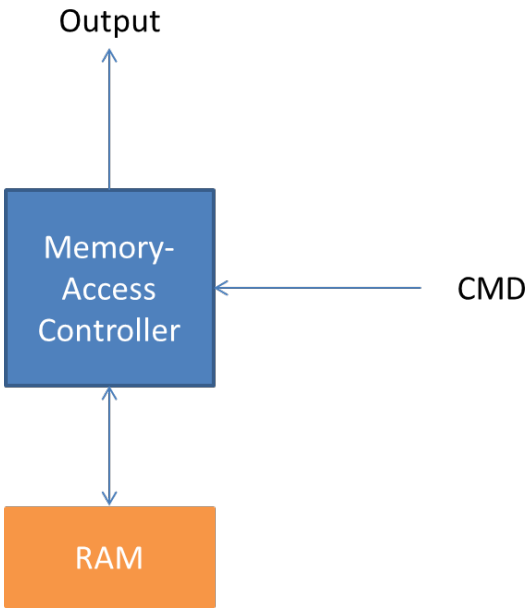


YAMM Overview – API (3)

- Search functions
  - Retrieve buffers by name
  - Retrieve buffers by address
  - Retrieve buffers in address range
  - Retrieve buffers by access
- Debug functions
  - Return the memory map allocation as a string
  - Dump memory map to file
  - Provide usage and fragmentation statistics

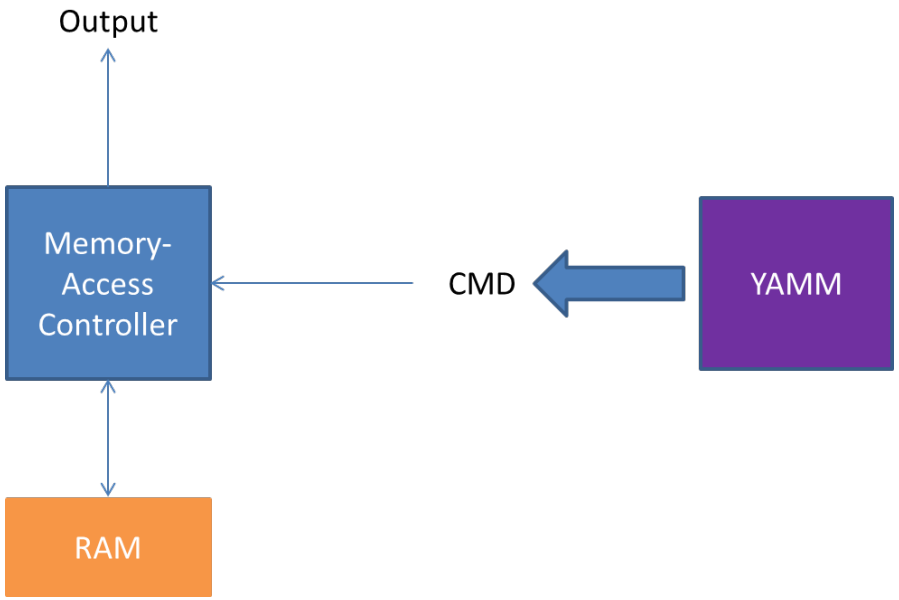
YAMM – Use case

- A typical use case would be a DUT that handles accesses to a memory with specific priorities.
- When using constraint random overlapping accesses can be generated which increases the difficulty of checking.



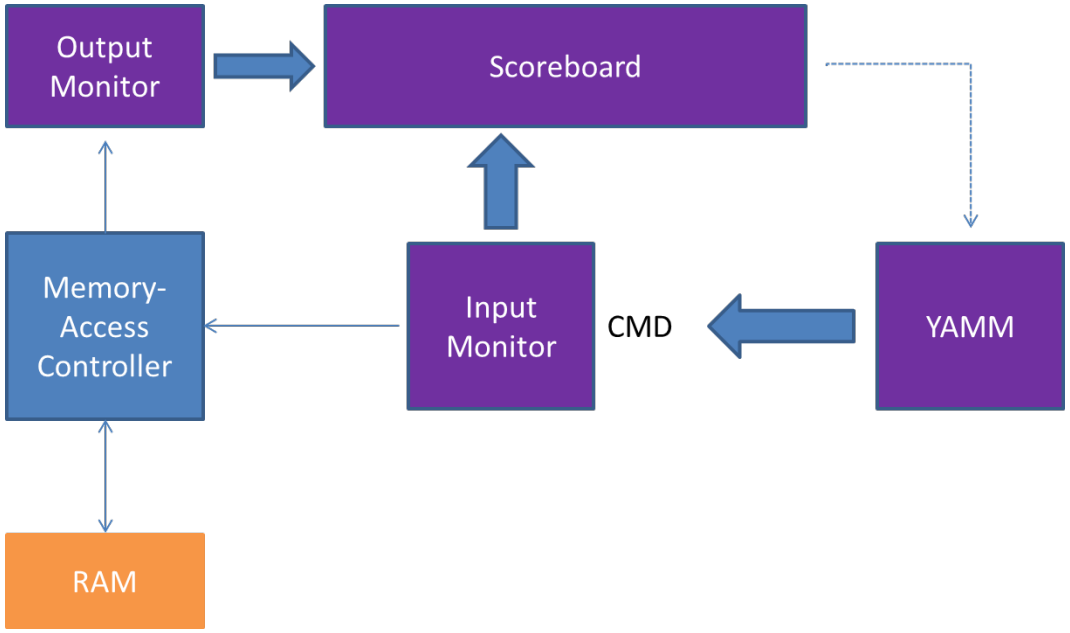
YAMM – Use case (2)

- YAMM can be easily used to generate interesting scenarios, by providing the buffers in which the accesses are done (address and size)



YAMM – Use case (3)

- Checking is made much easier when YAMM is used to control the buffers in which the writes/reads are done.



MAM – UVM Solution

- Oriented towards memory modeling
- MAM is used with UVM_MEM to reserve specific memory regions
- Supports only “greedy” allocation mode
- Specific regions can be freed or entire memory can be wiped
- Memory state can be returned as a string

MAM vs YAMM – Feature Wise (1)

- Memory
  - UVM_MAM is linked to UVM_MEM which provides the memory locations used for storing data
  - YAMM top level as well as every individual buffer contains a memory map composed of multiple buffers that can store simple data
- Allocation
  - UVM_MAM can only allocate on previously unallocated memory (2 allocation modes)
  - YAMM can allocate new buffers in either unallocated memory or inside an already allocated buffer (6 allocation modes)

MAM vs YAMM – Feature Wise (2)

- Finding buffers
  - UVM_MAM provides only an iterator; user must implement the search functions
  - YAMM provides support for finding and modifying buffers by different criteria

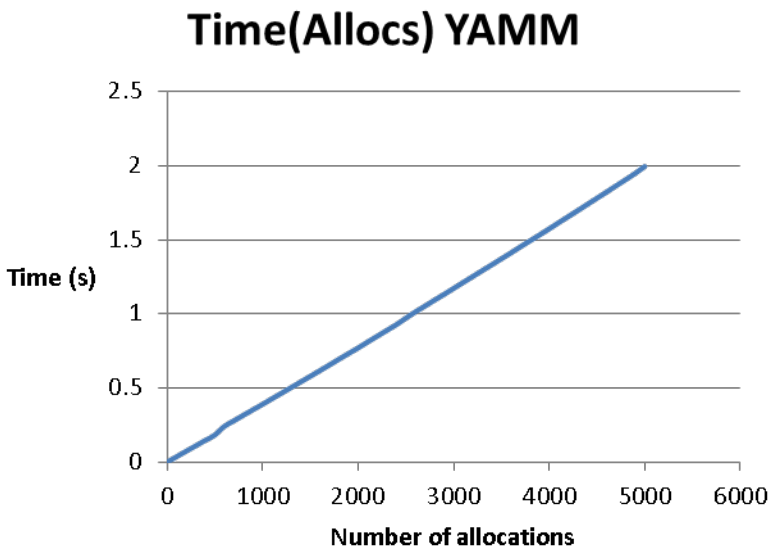
MAM vs YAMM – Feature Wise (3)

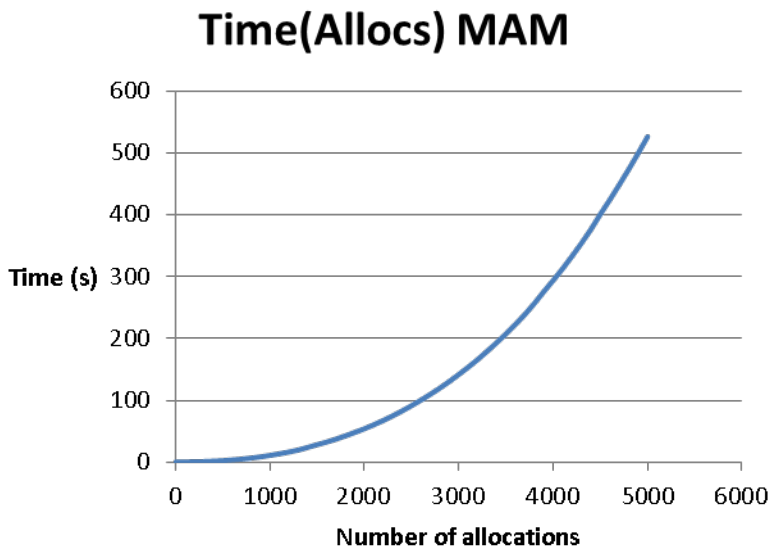
- Ease of use
  - UVM_MAM is complex and rather hard to use and for features beyond reserving and freeing regions user has to go to objects higher in hierarchy
  - YAMM has a more user friendly API, memory map can be accessed by calling functions on the top level and also specific regions can be accessed by calling same functions on the chosen buffers

Performance test - Parameters

- Memory space of 1G
- Allocation of 5000 buffers of size 100
- Measuring the time taken for allocation every 100 allocations
- For MAM, request_region() with default policy
- For YAMM, allocate_by_size() with RANDOM_FIT allocation mode was used

Performance test – MAM vs YAMM





More than 200x the speed

Examples – Initialization Example

```
yamm new_memory;
yamm_size_width memory_size = 10000;
new_memory = new;
new_memory.build("memory_name", memory_size);
```

Examples – Sequence Example

```
class user_sequence extends uvm_sequence;
   rand int unsigned access_size;
   ...
   task body();
      yamm_buffer buffer =
p_sequencer.user_memory.allocate_by_size(access_size
);
      `uvm_do_with(user_item, {
         address == buffer.get_start_addr();
         size == buffer.get_size();
         data == buffer.get_contents();
      })
   endtask
endclass
```

Examples – Scoreboard example

```
class user_scoreboard;
   yamm user_memory;
   ...
   //function checks if the current access is done
to a previously allocated address
   function void check_access(user_item item);
      if(user_memory.get_buffer(item.addr) == null)
         `uvm_error(get_name(), "Access detected to
a non-allocated memory address!")
   endfunction
endclass
```

YAMM Availability

- Open Source under Apache 2.0 license
- Available in both SystemVerilog and C++ as well
- Blog: www.amiq.com/consulting/blog
- Github: www.github.com/amiq-consulting/yamm

Questions