A Systematic IP Verification Solution of Complex Memory Management for Storage SOC

Jinsong Liu, jinsong@micron.com
Shuhui Wang, wangshuhui@micron.com
Data Struct Requirement

- Alignment – 8/16/32/64/128-bit aligned.
- Entry Width – 8/16/32/64/128-bit or User-defined transaction.
- Size – User-defined.
- Locality – Nearby/Broad.
Features and Benefits

✓ Provide system memories modeling for complex SOC.
✓ Address mapping of system memories can be configurable.
✓ Support different SOC buses like APB, AHB, AXI or other internal system bus.
✓ Allocate memory with different requirements of alignment / width / size / locality / mode.
✓ Load and store user-defined transaction.
✓ Collect functional coverage to make sure all legal memory regions are fully covered.
✓ Support debug mode. Allocation and deallocation, load and store operations can be dumped.
uvm_mem_mam Extension

uvm_mem_mam supports:
- Allocation Alignment – Byte aligned.
- Allocation Mode – Thrifty.
- Allocation Locality – Nearby.
- Read and Write – Data struct: uvm_reg_data

Extension from uvm_mem_mam:
✓ Allocation Alignment – 8/16/32/64/128-bit aligned.
✓ Allocation Mode – Thrifty and Greedy.
✓ Allocation Locality -- Nearby and Broad.
✓ Read and Write – Data struct: User-defined transaction and 8/16/32/64/128-bit entry width.
## Implementation – Configuration I

### Memory Manager Configuration API

<table>
<thead>
<tr>
<th>API Prototyping</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bit configure (smm_cfg cfg_p)</code></td>
<td>Configure system memory manager, this API is usually called initially after the environment has been built. (For example, call the API in <code>connect_phase()</code> in IP environment.)</td>
</tr>
</tbody>
</table>

### Memory Manager Configuration Data Struct

<table>
<thead>
<tr>
<th>Field</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>start_offset</code></td>
<td>The start address of system memory address space that is to be allocated.</td>
</tr>
<tr>
<td><code>end_offset</code></td>
<td>The end address of system memory address space that is to be allocated.</td>
</tr>
<tr>
<td><code>alloc_mode</code></td>
<td>Allocation mode, by default this field is randomized between the below modes:</td>
</tr>
<tr>
<td></td>
<td>GREEDY: Allocate un-allocated memory region with priority.</td>
</tr>
<tr>
<td></td>
<td>THRIFTY: Allocate just released memory region with priority.</td>
</tr>
<tr>
<td><code>locality</code></td>
<td>Locality for memory allocation, by default this field is randomized between the below modes:</td>
</tr>
<tr>
<td></td>
<td>BROAD: Randomly allocate regions throughout system memory.</td>
</tr>
<tr>
<td></td>
<td>NEARBY: Allocate regions adjacent to allocated regions with priority.</td>
</tr>
</tbody>
</table>
Example: Memory Manager Configuration

```cpp
function void connect_phase(uvm_phase phase);
    smm_cfg cfg_v;
    system_mem_manager smm;

    super.connect_phase(phase);
    //Create a singleton memory manager.
    smm = system_mem_manager::get();

    //Configure address space for system memory.
    cfg_v = smm_cfg::type_id::create("cfg_v");
    cfg_v.randomize(with {
        start_offset = 32'h0000_0000;
        end_offset = 32'hFFFF_FFFF;
    });
    smm.configure(cfg_v);

    //Pass shadow memory to memory manger.
    smm.mem_h = mbp_slave_agent[0].slave_mem;
endfunction
```
## Implementation – Allocation/Deallocation I

### Memory Allocation/Deallocation APIs

<table>
<thead>
<tr>
<th>API Prototyping</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| bit alloc_mem (input int byte_size, input mem_manager_policy policy, output bit[31:0] start_addr) | Allocate memory segments with the specified size and policy (alignment, allocation strategy and etc.), and the returned start_addr is the starting address of the allocated memory.  
Return 0 for failures                                                                 |
| bit dealloc_mem (input bit[31:0] start_addr)                                    | De-allocate memory segments which have been allocated before. Note that the start address should be the same as that of the corresponding alloc_mem function and all the allocated memory segment will be released.  
Return 0 for failures                                                                 |
| bit reserve_mem (input int byte_size,input[31:0] start_addr)                   | Allocate one memory segment with specified size and starting address. The memory segment can’t be allocated unless it is released. This API can also be used to pre-allocate some memory regions initially which can’t be allocated. For example, address maps for block registers need to be reserved.  
Return 0 for failures. (i.e. The memory segment has been allocated before)       |
## Implementation – Allocation/Deallocation II

### Memory Allocation Policy Data Struct

<table>
<thead>
<tr>
<th>Field</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_min_offset</td>
<td>Specify the minimum memory offset that can be allocated. This field and the below alloc_max_offset can be used to define a sub-region to be allocated.</td>
</tr>
<tr>
<td>alloc_max_offset</td>
<td>Specify the maximum memory offset that can be allocated.</td>
</tr>
<tr>
<td>addr_align</td>
<td>Allocation address alignment:</td>
</tr>
<tr>
<td></td>
<td>BYTE_ALIGN: allocated address should be byte aligned</td>
</tr>
<tr>
<td></td>
<td>WORD_ALIGN: allocated address should be 2-byte aligned</td>
</tr>
<tr>
<td></td>
<td>DWORD_ALIGN: allocated address should be 4-byte aligned</td>
</tr>
<tr>
<td></td>
<td>QWORD_ALIGN: allocated address should be 8-byte aligned</td>
</tr>
<tr>
<td></td>
<td>EWORD_ALIGN: allocated address should be 16-byte aligned</td>
</tr>
</tbody>
</table>
Example: Memory Allocation/Deallocation

```verilog
task body();
    system_mem_manager smm;
    mem_manager_policy mmp;
    bit [31:0] addr;

    mmp = new();
    smm = system_memory_manager::get();

    mmp.addr_align = SMM_BYTEALIGN;
    mmp.alloc_min_offset = 32'hFFF3_0000;
    mmp.alloc_max_offset = 32'hFFF3_F000;
    assert (smm.alloc_mem(4096, mmp, addr));
    `uvm_info(get_name(), $sformatf("Allocated addr=0x%0h", addr), UVM_LOW)

    smm.dealloc_mem(addr);
    `uvm_info(get_name(), $sformatf("De-allocate addr=0x%0h", addr), UVM_LOW)
endtask
```

Example: Memory Reservation

```verilog
function reserve_csr_mem_region();
    smm.reserve_mem(CSR_MEM_SIZE, CSR_BASE_ADDR);
endfunction
```
## Implementation – Load and Store APIs I

### Load and Store User-defined Transaction

<table>
<thead>
<tr>
<th>API Prototyping</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit store_usr_data(input bit[31:0] start_addr, input base_usr_data usr_data)</td>
<td>Backdoor store data with user-defined format (extended from base_usr_data) into specified memory address. Return 0 for failures</td>
</tr>
<tr>
<td>bit load_usr_data(input bit[31:0] start_addr, inout base_usr_data usr_data)</td>
<td>Backdoor load data with user-defined format (extended from base_usr_data) from specified memory address. Return 0 for failures</td>
</tr>
</tbody>
</table>

### Virtual Function in Base Class of User-defined

<table>
<thead>
<tr>
<th>Virtual Function</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>int get_byte_size();</td>
<td>This function must be extended in child classes to define the byte size of user defined data.</td>
</tr>
<tr>
<td>void unpack_bytes(input bit[7:0] byte_in[]);</td>
<td>This function must be extended in child classes to convert input byte array into user-defined data. The input byte array size should match what is defined in the above get_byte_size() function. This function will be called by load_usr_data API.</td>
</tr>
<tr>
<td>void pack_bytes(output bit[7:0] bytes_out[]);</td>
<td>This function must be extended in child classes to convert user-defined data into byte array. The byte array size is also defined in the above get_byte_size() function. This function will be called by store_usr_data API.</td>
</tr>
</tbody>
</table>
Example: User-defined Transaction Extends From Base_User_Data

class dec_bufferlist_txn extends base_usr_data;

    rand Dec_BufferList_DW0 dw0;
    rand Dec_BufferList_DW1 dw1;

    `uvm_object_utils_begin(dec_bufferlist_txn)
    `uvm_field_int(dw0, UVM_ALL_ON);
    `uvm_field_int(dw1, UVM_ALL_ON);
    `uvm_object_utils_end

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

    // This function is mandatory to be implemented to specify the byte size of this txn.
    function int get_byte_size();
        return 8;
    endfunction

//This function will be called by load_usr_data().
function void unpack_bytes(input bit[7:0] byte_in[]);
    assert(byte_in.size() == get_byte_size()); //check input array size
dw0={byte_in[3], byte_in[2], byte_in[1], byte_in[0]};
dw1={byte_in[7], byte_in[6], byte_in[5], byte_in[4]};
endfunction

//This function will be called by store_usr_data().
function void pack_bytes(output bit[7:0] bytes_out[]);
    bytes_out = new[get_byte_size()]; //create an array with expected size.
    {bytes_out[3], bytes_out[2], bytes_out[1], bytes_out[0]} = dw0;
    {bytes_out[7], bytes_out[6], bytes_out[5], bytes_out[4]} = dw1;
endfunction

function string convert2string();
    string s;
    s = { s, $sformatf("dw0: %p\n", dw0 ) };
    s = { s, $sformatf("dw1: %p\n", dw1 ) };
    return s;
endfunction
endclass
Example: Store User-defined Transaction

```verilog
rand dec_bufferlist_txn buf_h;
bit [31:0] cwl_addr = 0;
bit suc;

buf_h = new;
if (!buf_h.randomize()) 'uvm_error(get_name(), "buf_h randomization failed")

suc = smm.store_usr_data(cwl_addr, buf_h);
if (suc == 0) 'uvm_error(get_name(), "store buf_h failed")
```
Example: Load User-defined Transaction

```verbatim
define dec_bufferlist_txn load_buf_list(bit [31:0] addr);
    bit suc;
    base_usr_data  base_usr_data_h;
    dec_bufferlist_txn buf_h;

    buf_h = new();

    //It is mandatory to pass a handle with “base_usr_data” type instead of its
    //extended type as the 2nd argument of load_usr_data().
    //It is recommended to $cast the extended item to a “base_usr_data” handle
    //and pass it to the load_usr_data() function. After calling load_usr_data(),
    //$cast the “base_usr_data” handle back to the extended item.
    if (!$cast(base_usr_data_h, buf_h))
        `uvm_error(get_name(), "child entity cast to parent handle failed.")

    suc = smm.load_usr_data(addr, base_usr_data_h);

    if (!$cast(buf_h, base_usr_data_h))
        `uvm_error(get_name(), "parent handle cast to child entity failed.")

    if (suc == 0)
        `uvm_error(get_name(), "load_usr_data failed")

    return buf_h;
endfunction
```