INTRODUCTION

Challenges in Verification

- Random for fast distribution
- Constrained for accuracy
- Controlled for corner case scenarios

Objective: Faster simulation for quick closure of verification while achieving all of the above

Memory Controller

- Controls flow of data between host memory and dynamic memory
- Host memory is divided into pages
- Pages are blocks of contiguous addresses
- Page aligned address: Multiple of page size (n), Max log(n) least significant zeros
- Unaligned address: Offset from start address = log(n) significant bits

PROPOSED APPROACHES FOR GENERATION OF ADDRESS BLOCKS

Approach I - Store all the addresses in a page

```cpp
for(i = 0; i < 1000; i = i + 1) begin
    used_address_list.push_back(address_generator_h.addr);
}
```

Approach II - Compress the address range

```cpp
for(i = 0; i < 1000; i = i + 1) begin
    start_address = address_generator_h.addr + address_generator_h.offset;
    end_address = start_address + address_generator_h.block_size;
    used_address_list.push_back(address_generator_h.addr);
}
```

Approach III - Move overlap check outside constraint

```cpp
for(i = 0; i < 1000; i = i + 1) begin
    if (address_generator_h.block_size > (offset == 0)) begin
        start_address = address_generator_h.addr + address_generator_h.offset + address_generator_h.block_size - address_generator_h.page_size;
        end_address = address_generator_h.addr + address_generator_h.offset + address_generator_h.block_size - address_generator_h.page_size;
        used_address_list.push_back(address_generator_h.addr);
    end
```

Approach IV - Use Verilog’s Surandom range function

```cpp
for(i = 0; i < 1000; i = i + 1) begin
    if (address_generator_h.offset < 1) begin
        start_address = address_generator_h.addr + rand_range(0, address_generator_h.block_size);
        end_address = address_generator_h.addr + rand_range(0, address_generator_h.block_size);
        used_address_list.push_back(address_generator_h.addr);
    end
```

CONCLUSION – A COMPARATIVE ANALYSIS

- Memory blocks constrained within range (start_address, end_address)
- Alignment control exercised using the bit align set to 1 (offset = 0) or 0 (offset = 0)
- Constraint offset = addr[OFFSET_WIDTH-1:0] can be replaced by offset = addr % page_size
- Overlap of memory blocks avoided by maintaining a queue of unused address blocks, used_address_list

Verdict

Using a for loop while populating the queue results in a huge amount of run time rendering the method highly inefficient. It is suitable only if the amount of data transfer involved is less.

Approach II - Memory block range compressed to [start_address/page_size, end_address/page_size]

- Actual address retrieved as (Generated address * Page size + Offset)

Verdict

Because the address range is now compressed, the queue used_address_list is populated with a single address instead of all addresses within a page. Constraint solver takes much lesser time since search list is very short.

Application of the most efficient address generator for 128 MB data transfer

- Comparative analysis highlights Approach III as the best suited for efficient constrained random generation of address blocks
- Performance of same approach studied by applying it to generate address blocks capable of holding data up to 128 MB and more
- Around 43,529 address blocks, aligned and unaligned, generated in random for the same
- A very nominal CPU time of 2.64 seconds recorded for the entire simulation