RamGen: Moving Memory from Physical to the Logical domain

- Jeff Scott, Jonathan Sadowsky, Jigar Savla

{ jscott@juniper.net , jsadowsky@juniper.net, jigar.savla@gatech.edu }

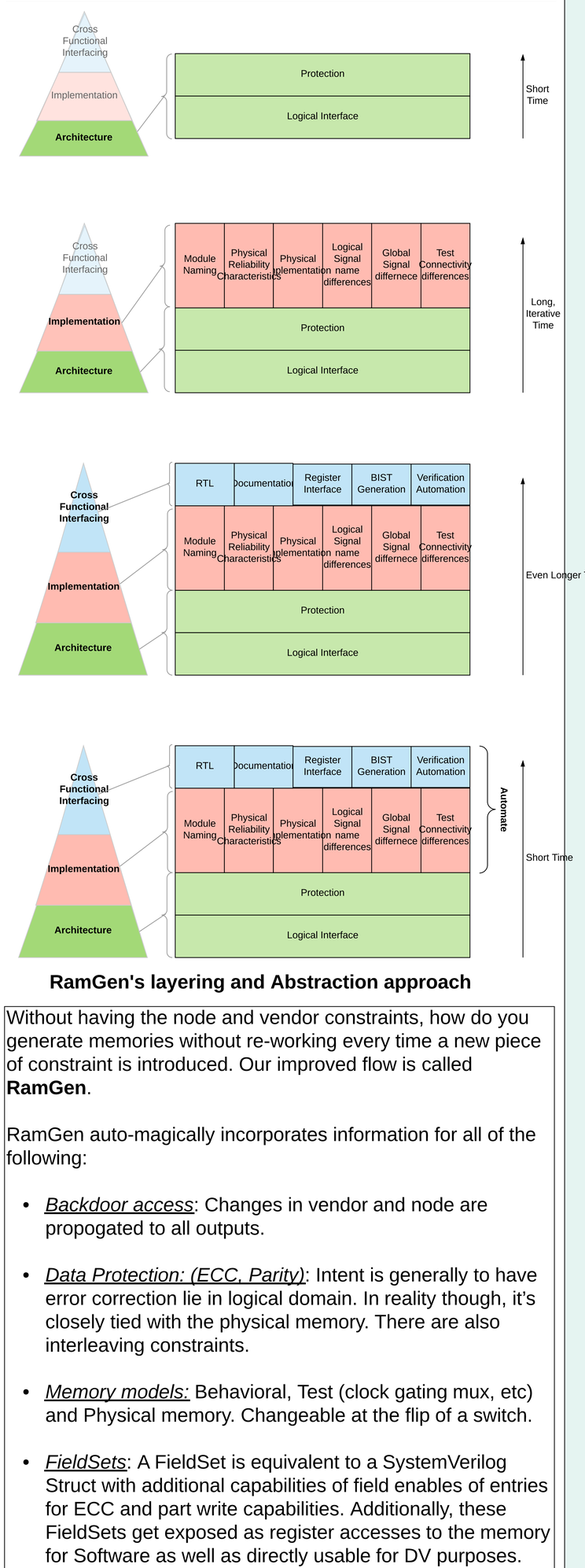
Introduction

For a memory, how does a designer give a high-level specification like width, depth and protection bits right at the start of the chip development and have it used throughout till tape-out?

Enable designers to get Simulation / Verification (DV) up and running before the memory vendor has been finalized or the physical memories have been ordered.

And once it's ordered, how do they make sure that verification doesn't have to re-do the work?

Answer is having a ***logical memory*** **instead of a** ***physical memory*****.**



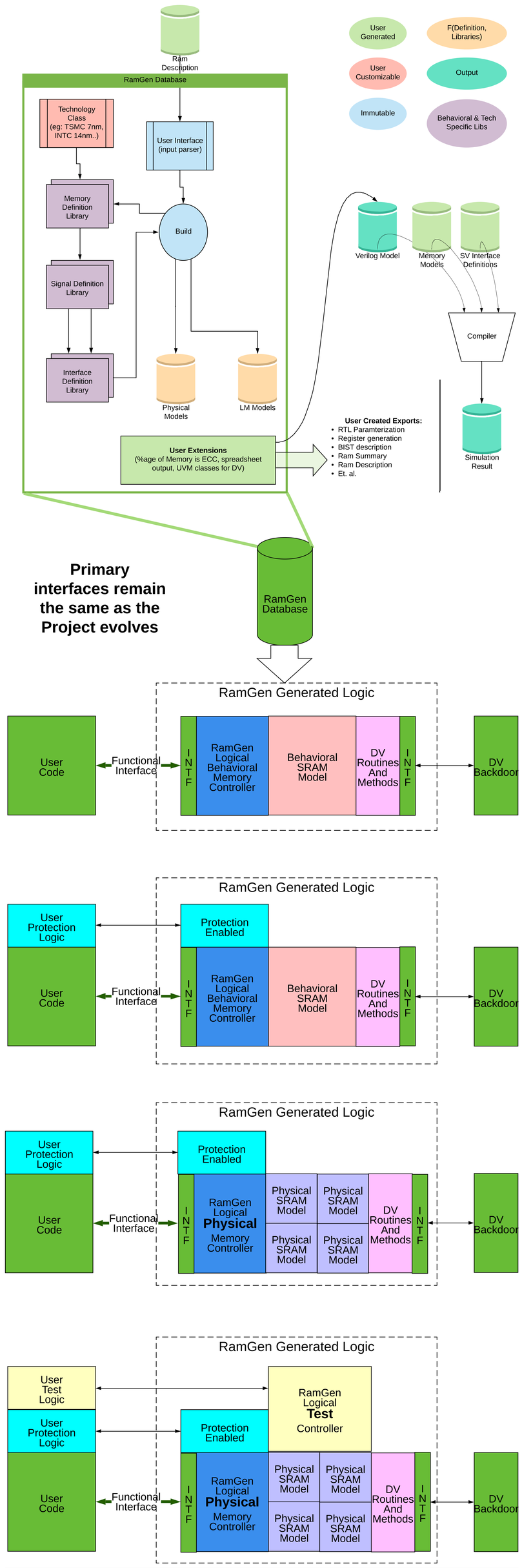
RamGen's layering and Abstraction approach

Without having the node and vendor constraints, how do you generate memories without re-working every time a new piece of constraint is introduced. Our improved flow is called **RamGen**.

RamGen auto-magically incorporates information for all of the following:

- *Backdoor access*: Changes in vendor and node are propogated to all outputs.
- *Data Protection: (ECC, Parity)*: Intent is generally to have error correction lie in logical domain. In reality though, it's closely tied with the physical memory. There are also interleaving constraints.
- *Memory models:* Behavioral, Test (clock gating mux, etc) and Physical memory. Changeable at the flip of a switch.
- *FieldSets*: A FieldSet is equivalent to a SystemVerilog Struct with additional capabilities of field enables of entries for ECC and part write capabilities. Additionally, these FieldSets get exposed as register accesses to the memory for Software as well as directly usable for DV purposes. Making memories all bits or single bit writeable. Custom sizing 4 entries of 4 bits wide. Handling ECC calculation.
- *Backdoor access*: As we change from vendor to vendor or even from one fabrication node to the other, memory capacities change. Through all of this, we would like to enable DV to get backdoor access up and running quickly. RamGen does this by presenting a logical view instead of the actual physical view. RamGen produces an output which will map the logical bits to the physical memory. As we change from say: 2x3 banked memory to 4x2 memory, DV doesn't have to tweak any parameters in their TestBenches for backdoor, RamGen will propagate the necessary changes to all its outputs, including DV, keeping them in sync.

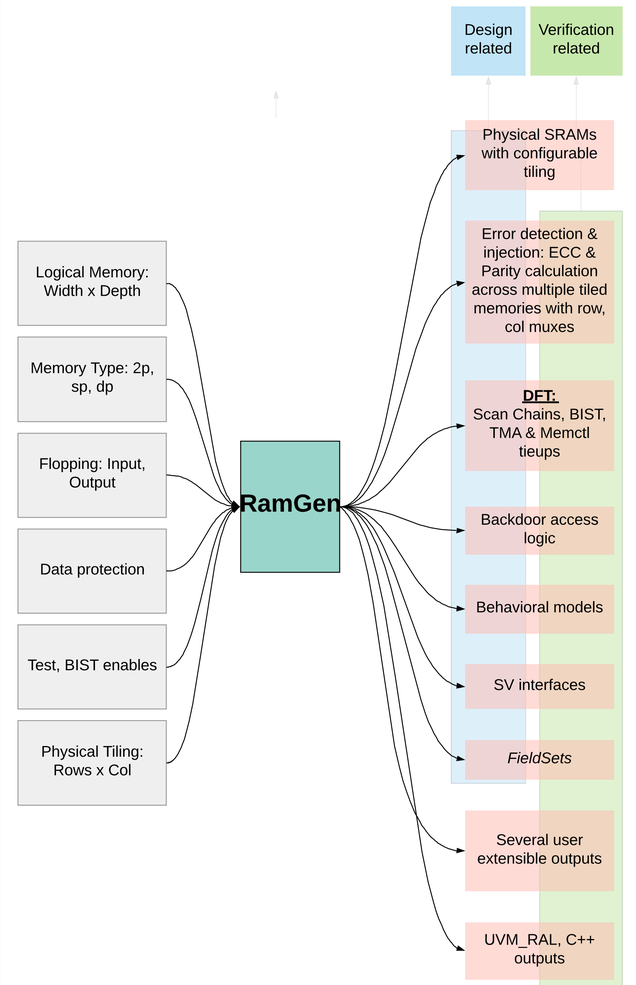
Methods



Inner workings:

- We have a Perl based hash table. It has several parameters, with a very minimum sub-set required to start prototyping. It creates SystemVerilog *interfaces* for functional and error correction. Having this makes it usable consistently for both Design and DV.
- All you really need is the Memory Type, Width, Depth, and I/O flopping. The remainder can be defaulted to off. Everything else is refinement as you go from a generic logical memory to a **design specific** logical memory (with protection and test, etc.) to a **physical memory** (with tiling, etc.).
- It also accesses a user dB to access vendor specific programmable information for other constraints which can then query specific parameters of this dynamic memory structure to make appropriate changes.

Conclusion



High level dataflow

Sample Test input:

```
$mname = "dac_2019";
$mname_uc = uc($mname);
$pad = $shrink_mem_size ? 0 : 104;
$mycfg = $shrink_mem_size ? "AB10121" : "AB10122";
$m->{$mname} = { ramgen_params => { cfg =>
"2psrfhc",
                 _field_map =>
$fsets->as_ramgen("eng_tmr_mem_t"),
                 DEPTH =>
$fsets->get_const("DAC_NUM_PAPERS"),
                 PAD => $pad,
                 memctl_group =>
"memctl_${mname}",
                 HAS_PROTECT => 1, MOD =>
"ecc",
                 mux_factor => 2,
                 opt_string => $mycfg,
                 num_cols => 2,
                 equalize_cols => 1,
                 R1 => 1, R1LP => 0,          }
};
```

Sample summary output:

```
DAC_2019_B1: 32768x20  2PSRFHC REGISTERED
LM with PROTECT
        Total Bits: 851968
        Physical:  2 memories ( 2x1 )
        Protection: 6 per entry
        Latency: 4 (1,0,0,0,0,0,0,1)
```

Experience

- For one of our complex designs, in just three days we had every memory defined, with protection, register access and the whole interrupt structure around it. Working with our other tools, the necessary uvm_regs, C++ and RTL models were also generated and successfully tested.
- We've tested this across several tape-outs and different memory vendors across multiple nodes. Designs have scaled well and enabled easy re-use.

Summary:

- Our approach here is an eco-system to enable rapid development. It's a way of defining a memory network, which is vendor agnostic.
- We take a specification and refine it over time to make the output more accurate over time.
- It automates front end memory generation. It has a set of models, makes it very easy to switch between models while the primary memory interface signals remain the same.
- We separate the Physical design decisions from the development flow. This leads to reduced code maintenance, less fragility and more portability.

ACM / IEEE's Design and Automation Conference, Las Vegas, NV | DAC 2019