UVM - How, why, and is it even necessary?

Outline

- My work on this project
- SystemVerilog and UVM
- Our UVM testbench
- Results
- Discussion

My work on the project

- Familizaring myself with SV and UVM
  - No prior knowledge of either
- Developing a UVM testbench for Martin's Leros ALU
- *Investigating the SystemVerilog DPI*
- *Looking at implementing similar functionality in Scala/Chisel*

SystemVerilog

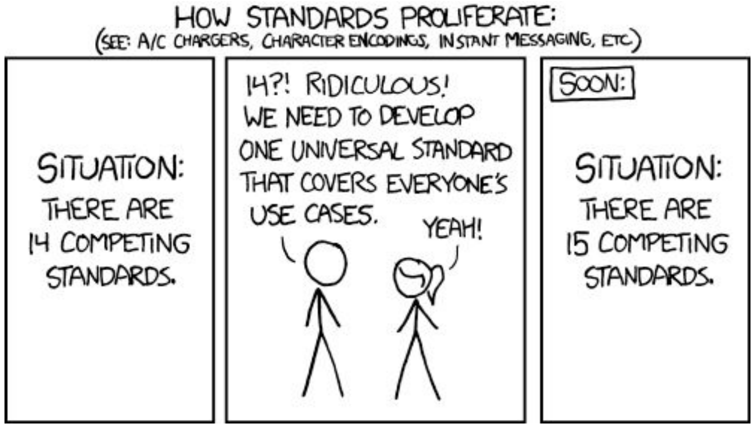
- An extension of the Verilog HDL, introduced in 2002
- Adds some much-needed features to hardware design
    - Enumerated types
    - Structs and unions (as in C)
    - “Procedural blocks” that help generate correct type of logic
- A much-needed upgrade to modernize Verilog

SystemVerilog & Verification

- SystemVerilog adds
    - Functional Coverage
    - Constrained Random Verification
    - Class-based testbench design (but only for verification, not for design)
- UVM is a class library in SystemVerilog

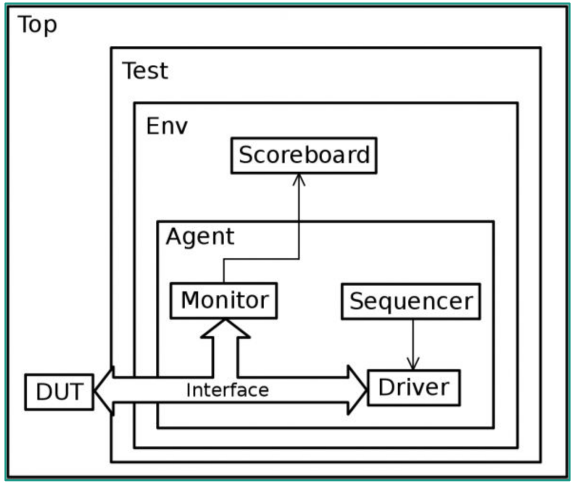
Purpose of UVM

- Standardize testbench designs between vendors
  - Previously: OVM, eRM, OSVVM, VMM
  - Promote code-reuse between testbenches
- Make testbenches more streamlined and simpler to extend
- It's HUGE


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.
Standards. By Randall Munroe. xkcd.com

Structure of a UVM Testbench

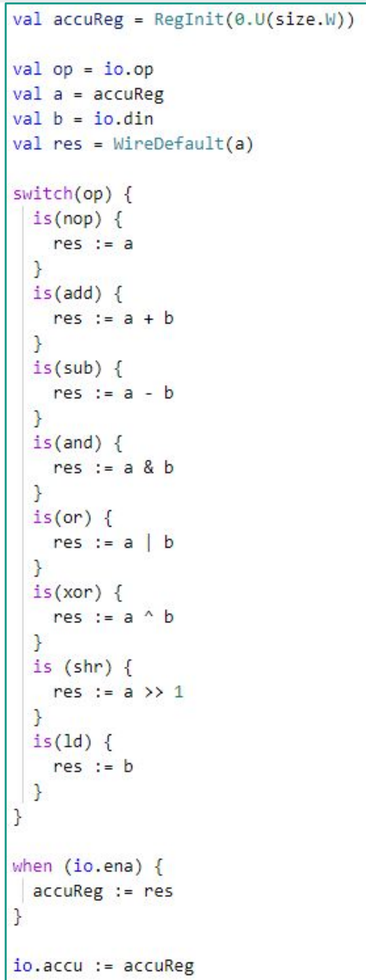
- Each component does only *one* thing
  - Encourages OOP principles
- Agents are an important building block
  - Write once, reuse everywhere



Structure of a UVM testbench. By Pedro Araújo / colorlesscube.com

Our UVM Testbench

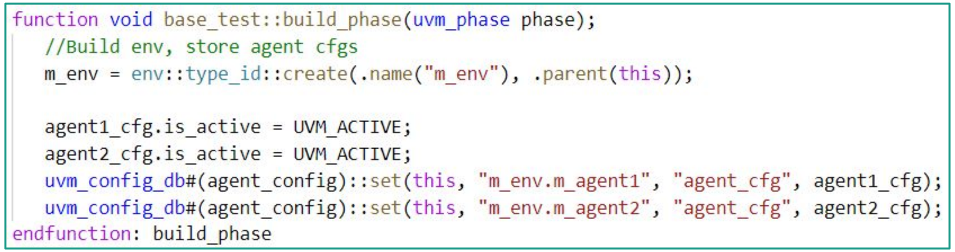
- Developed a UVM Testbench for the Leros ALU
- Tries to follow "best practice"
  - Uses base classes and inheritance as much as possible
- Uses random and constrained random inputs to test functionality
- Uses SV coverage to ensure all functionality has been tested



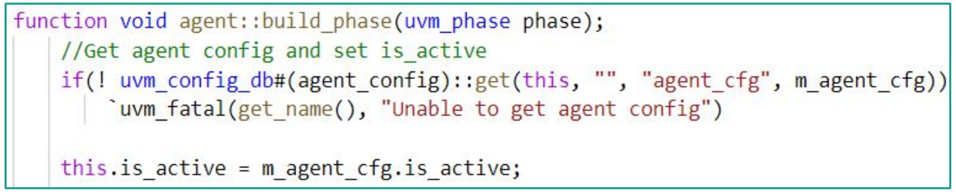
```
val accuReg = RegInit(0.U(size.W))

val op = io.op
val a = accuReg
val b = io.din
val res = WireDefault(a)

switch(op) {
  is(nop) {
    res := a
  }
  is(add) {
    res := a + b
  }
  is(sub) {
    res := a - b
  }
  is(and) {
    res := a & b
  }
  is(or) {
    res := a | b
  }
  is(xor) {
    res := a ^ b
  }
  is (shr) {
    res := a >> 1
  }
  is(ld) {
    res := b
  }
}

when (io.ena) {
  accuReg := res
}

io.accu := accuReg
```

Our UVM Testbench

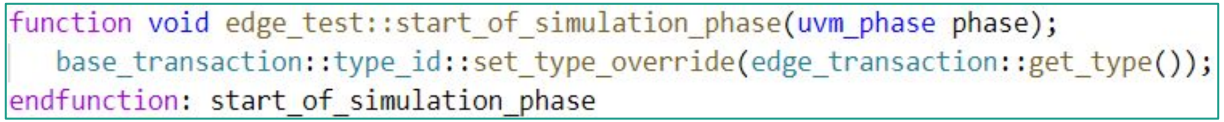
- The *UVM config DB* and *UVM Factory*are essential to the structure
    - Dynamic instantiation of correct component types
    - Pass variables to components in a dynamic fashion



```
function void base_test::build_phase(uvm_phase phase);
   //Build env, store agent cfgs
   m_env = env::type_id::create(.name("m_env"), .parent(this));

   agent1_cfg.is_active = UVM_ACTIVE;
   agent2_cfg.is_active = UVM_ACTIVE;
   uvm_config_db#(agent_config)::set(this, "m_env.m_agent1", "agent_cfg", agent1_cfg);
   uvm_config_db#(agent_config)::set(this, "m_env.m_agent2", "agent_cfg", agent2_cfg);
endfunction: build_phase
```



```
function void agent::build_phase(uvm_phase phase);
   //Get agent config and set is_active
   if(! uvm_config_db#(agent_config)::get(this, "", "agent_cfg", m_agent_cfg))
      `uvm_fatal(get_name(), "Unable to get agent config")

   this.is_active = m_agent_cfg.is_active;
```



```
function void edge_test::start_of_simulation_phase(uvm_phase phase);
   base_transaction::type_id::set_type_override(edge_transaction::get_type());
endfunction: start_of_simulation_phase
```

Results

- Just about 1000 LOC for this testbench
  - With comments and whitespace, probably 7-800 without
- Martin’s Scala tester does almost the same
  - Fits in 74 lines, with whitespace
- But SV/UVM adds CRV, coverage and guidelines

So, UVM?

- I think UVM has potential
  - **Not** the correct choice for simple designs like an ALU
  - For complex designs it makes more sense
- Has an advantage over Chisel in supporting functional coverage + CRV
- Chisel testers are simpler to write + more intuitive
  - Does Chisel scale well enough for large-scale projects?
- If we can add coverage + CRV to Chisel, it could be a game changer
- Chisel is more approachable for software-first engineers
  - Could very well give a boost to the hardware scene