Specifying Hardware Communication as Programs

Ernest Ng, Nikil Shyamsunder, Francis Pham, Adrian Sampson, Kevin Laeufer

ernest@cs.cornell.edu

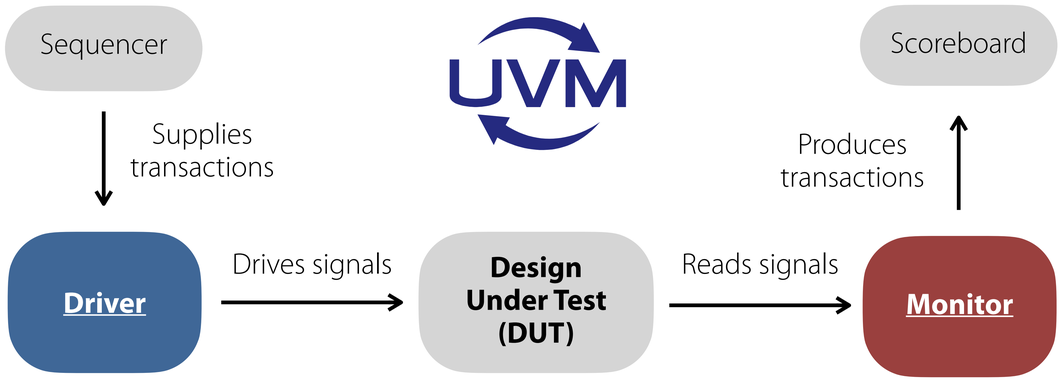
CornellBowers Computer Science

1. Problem: Testing & debugging hardware is hard

To test RTL modules, one usually implements two programs:

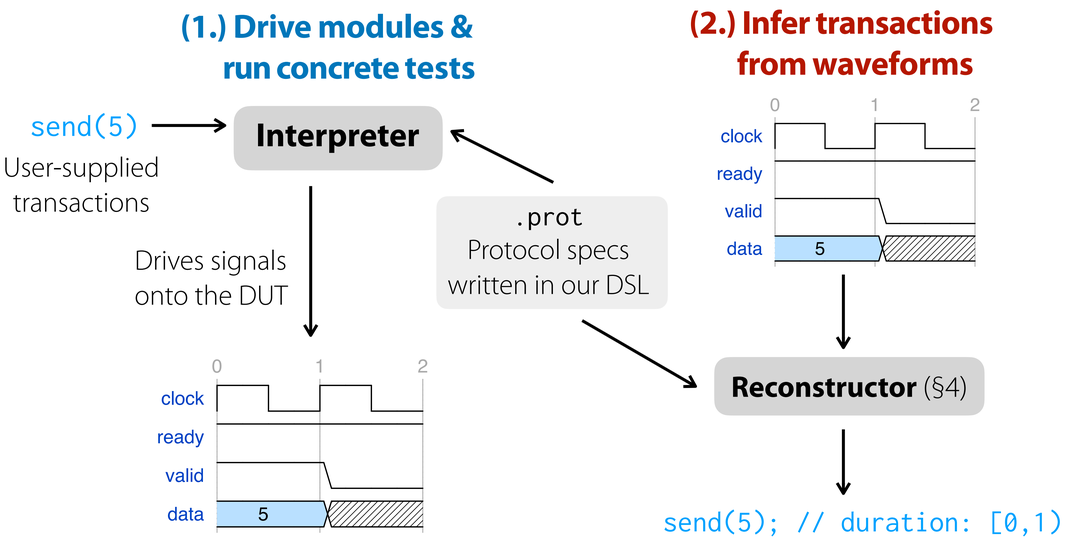
Drivers: transactions → drives signals onto DUT
Monitors: reads signals → produces transactions

Developed separately ⇒ inconsistencies!



2. Idea: Write one program to obtain both the driver & monitor

We design a domain-specific language (DSL) in which users write succinct imperative programs that specify the behavior of HW communication protocols.
Key idea: The same program can be used to do both tasks!



For task (1): We implement an interpreter for our DSL that tests whether Verilog DUT implementations abide by user-supplied protocol specs in our DSL.
For task (2): We've implemented a tool, the reconstructor (§4), which performs this!

Takeaway: Our design enables the protocol spec to serve as a single source of truth, instead of being scattered across ad-hoc tests and frameworks!

3. Example: Specifying a ready-valid handshake in our DSL

```
prot send<DUT: ReadyValid>(data: u8) {
  DUT.valid := 1;
  DUT.data := data;
  while (DUT.ready ≠ 1) { step(); }
  step();
}
```

Indicate that we have valid data to send

Wait till receiver is ready, then use one cycle to transfer data

Simple imperative semantics, like software!

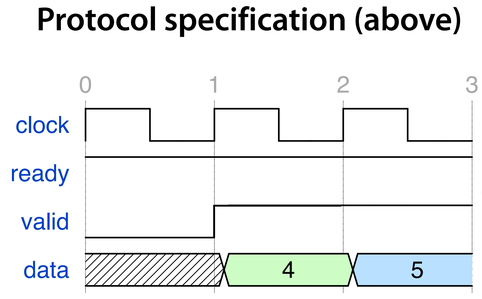
Language features:

`step()` Advances clock cycle

`fork()` Enables concurrent protocol execution (models pipelining)

4. Inferring transactions from waveforms

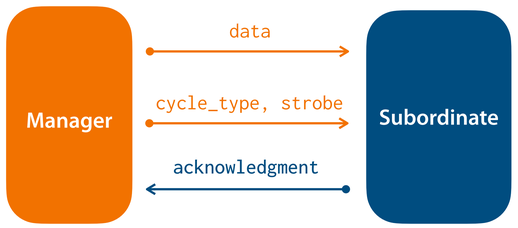
Our DSL comes equipped with a tool, the ***reconstructor***, which given a protocol specification in our DSL and a concrete waveform (execution trace), infers a transaction trace that is consistent with the waveform.



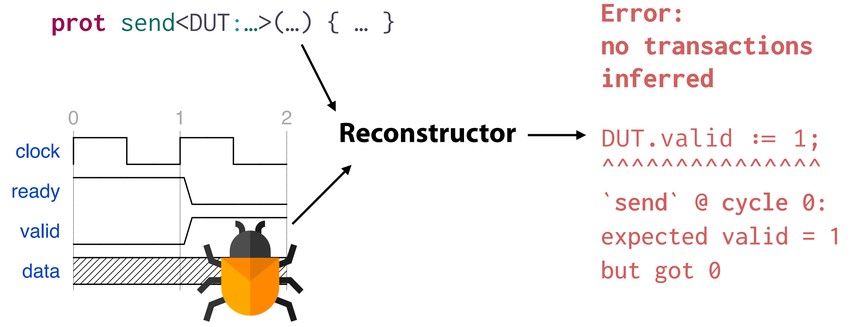
The reconstructor symbolically executes the protocol spec and determines whether assignments in the protocol are consistent with the waveform data. Inconsistent execution paths are immediately pruned, allowing transactions to be efficiently inferred.

5. Evaluation (Ongoing)

Language Expressivity:
Demonstrate that the same specification for the Wishbone interconnect allows us to both drive designs & infer transaction traces for a variety of WB implementations



Reconstructor Utility:
Demonstrate that the reconstructor can reveal protocol violation bugs (e.g. violations of the AXI-Stream spec) from an existing benchmark of FPGA bugs (Ma et al. ASPLOS '22)



6. Future Work

Extend language to model features from the full AXI & Wishbone protocols (e.g. burst mode)

Support higher-level protocols (e.g. CXL) + multiple clock domains

cucapra/protocols

arXiv:2606.13659