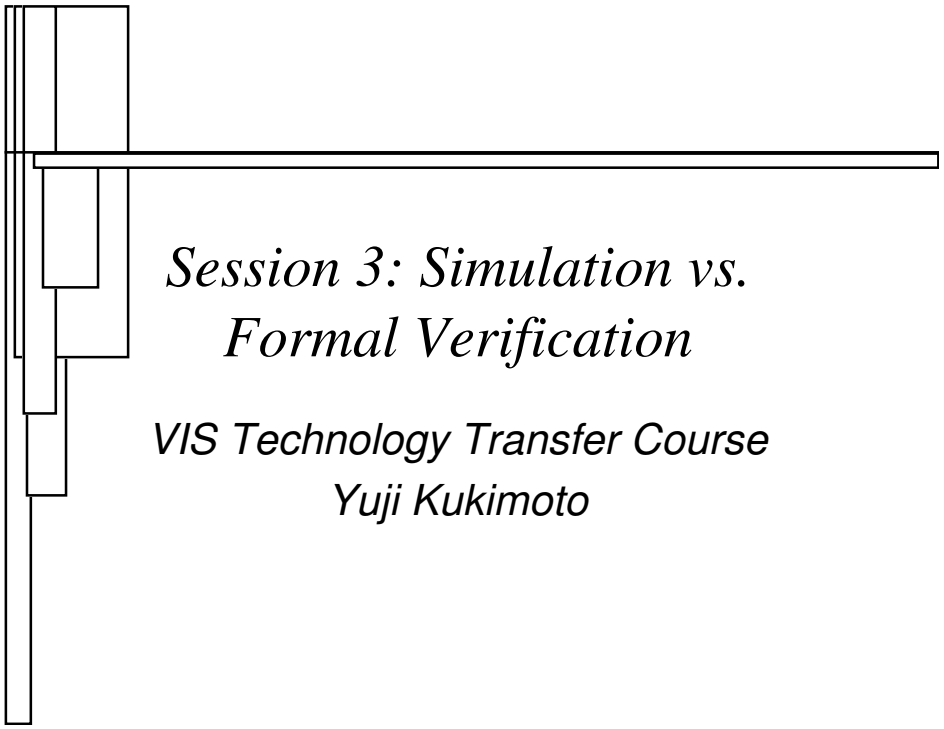
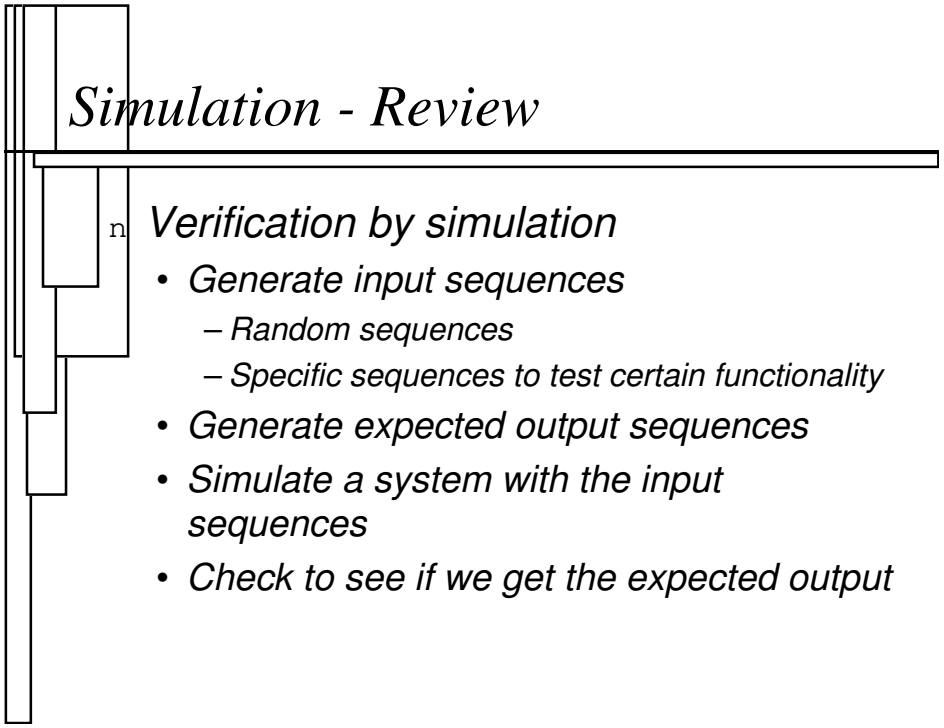


Session 3: Simulation vs. Formal Verification

VIS Technology Transfer Course

Yuji Kukimoto



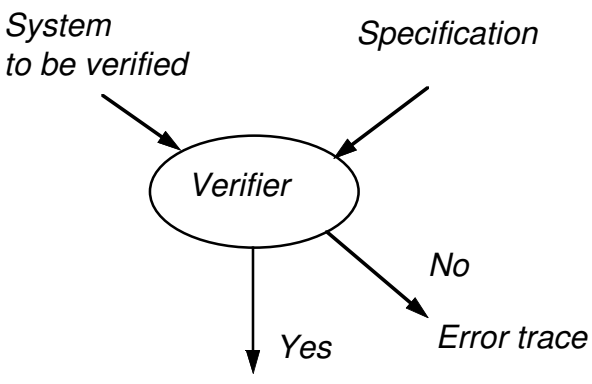
Simulation - Review

- Verification by simulation
  - Generate input sequences
    - Random sequences
    - Specific sequences to test certain functionality
  - Generate expected output sequences
  - Simulate a system with the input sequences
  - Check to see if we get the expected output

Problems in Simulation

- *Need to simulate billions of cycles*
- *Not complete*
  - *No guarantee for sequences not simulated*
- *Must generate expected response*
- *Effective simulation sequences are hard to generate*
  - *High-coverage, Corner cases*
  - *Bugs are typically introduced at locations to which designers did not pay attention*

What is Formal Verification?



Formal Verification

- *Complete with respect to a given property*
  - *Correctness is guaranteed mathematically*
- *No need to generate expected output sequences*
- *Can generate a short error trace if a property fails*
- *Formal verification is useful to find bugs in design*

Formal Verification - History

- *Theory has been around for a long time*
  - *Size of systems to be verified: 10^6*
- *A major breakthrough in 1990 (Clarke et al., CMU)*
  - *Size of systems to be verified: 10^20*
- *Subtle bugs were found in real systems by formal verification*
  - *Cache coherency protocol, Encore Gigamax (McMillan)*

Simulation vs. Formal Verification

- *Simulation*
  - *Not complete*
  - *Need to generate expected behavior*
  - *Difficult to cover corner cases*
- *Formal Verification*
  - *Complete wrt specification*
  - *No need to generate expected behavior*
  - *Corner cases are automatically taken care of*

Simulation vs. Formal Verification - cont'd

- *Simulation*
  - *CPU intensive*
    - *have to run billions of cycles*
  - *Can handle large systems*
- *Formal Verification*
  - *Most of the state-of-the-art methods are memory intensive*
    - *Intenal data structures (BDD)*
  - *Memory usage is strongly related with the size of systems to be verified*

Simulation vs. Formal Verification - cont'd

- *Simulation is still a good and fast way to verify early design*
- *Formal verification gives additional confidence in correctness*
- *Two techniques are complementary*
- *VIS supports both formal verification and simulation*

Verilog Extensions for VIS

- *Two extensions to synthesizable subset of Verilog*
  - *Enumerated types*
    - *Easy to describe system behaviors in high-level*
  - *Non-determinism*
    - *Necessary to model environments*

Enumerated Types

- *Similar to enumerated types in C*

```
/* type definition */
typedef enum {IDLE, READY, BUSY} controller_state;
```

This declares that the type "controller_state" is an enumerated type

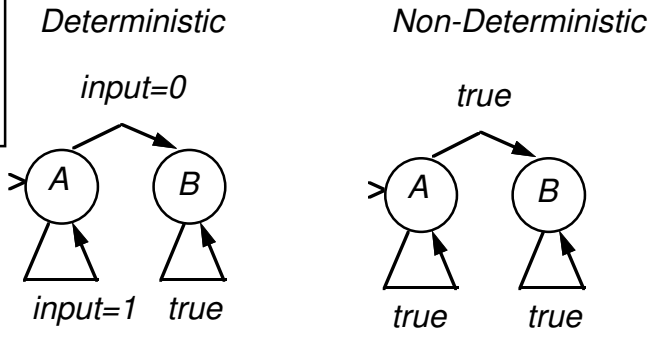
```
/* type declaration for a register variable */
controller_state reg state;
```

This says that state is a register variable of type "controller_state"

What is Non-determinism?

- *Deterministic systems*
  - *For each state-input pair, the next state and output are unique*
  - *Final systems implemented as circuits are deterministic*
- *Non-deterministic systems*
  - *There exists a state-input pair for which the next state and output are not unique*
  - *Can describe a set of behaviors*

What is Non-determinism? - cont'd



Why Non-determinism?

- Necessary to model environments
- Example:
  - Arbiter + Controllers: System to be verified
  - Clients: Environment of the system
    - A request signal is set to high non-deterministically
    - Can capture all the possible behaviors of the client with non-determinism

Non-determinism in Verilog

n

$ND construct

- *Creates a non-deterministic signal source*
- *Should only be used in an assign statement*
- *Should be used only to create non-deterministic constants*

```
/* definition of a wire variable */
wire rand_choice;
/* non-determinism */
assign rand_choice = $ND(0,1);
```