Discrete Event Simulation & VHDL

Prof. K. J. Hintz

Dept. of Electrical and Computer Engineering

George Mason University

Discrete Event Simulation

- Material from *VHDL Programming with Advanced Topics* by Louis Baker, Wiley, 1993

DES & VHDL

- VHDL is a special case of DES
  - Not continuous differential equation
  - Not finite difference equation
  - DES is discontinuous in time
    - Microprocessor/microcontroller
    - Finite State Machine
    - Language Recognizer
    - Clocked/synchronous devices in general

DES Properties

- Restricts Allowed Values of Signals
- Restricts Transitions to the Result of Events, *i.e.*, It Is Causal

Why discuss DES?

- Understanding Leads to Better and More Efficient Use of Simulation
- Queuing of Transactions May Not Reflect Reality Due to Inertial or Transport Delay
- Not All Simulations Will Be Deterministic
  - Interrupts
  - Instruction Mix
  - Intermediate values causing branching

Two Approaches to DES

- Event Scheduling
  - Simulation proceeds from event to event
  - VHDL
- Process Interaction
  - Follows customer through system
  - Suitable for queuing systems
  - SIMULA language

Queues

- Events Ordered in Time
- Many Deletions of Events From Queue May Be Expected Due to the Default Inertial Delay

Queues

- Each Driver Must Maintain Projected Waveform
  - Time and value of most recent event (`s'last_event`)
  - Next scheduled transaction
  - When event occurs, value before event must be saved (`s'last_value`)
  - Event time (`s'last_active`)

Queues

- History of Signals Is NOT Maintained by Simulation Even Though Saved in History File
  - Access is only available implicitly

Efficient Simulation

- Finite Propagation Delay Vice Delta Delay
  - Finite delay more accurately deflects world, but requires simulation to check each different delay time, therefore slow
  - A δ-delay only needs to resolve signals after one δ-delay, therefore faster
  - If all components had same delay, then no difference

Efficient Simulation

- Limit Use of **INOUT** Ports
  - Use Only When Essential
  - Usage Limits Error Checking Associated With Port Modes
  - Simulation Must Resolve Direction

Efficient Simulation

- Limit Use of Generics
  - `Generic` statements require global storage
- Minimize Feedback
  - The More Layers of Feedback There Are, the More Simulations Are Needed to Resolve the Output at Its Next State

Internal Event Queues

- Necessary to Simulate Interaction With Outside World
- Signals Cannot Be Used Because They Can't Represent External Events Put on a Queue
- VHDL Allows Coding Your Own Event Queues

Simulation Determinism

- Deterministic
  - Microprocessor with no interrupt doing well-defined task, *e.g.*, boot-up sequence, MSDOS start for x86
  - All inputs to a combinational circuit

Stochastic Simulation

- More Common
- Economic Simulation
  - Truly random #: derived from physical process
  - Pseudo-random: $U(0,1)$
    - Generators uniform, zero mean, unit variance
    - All other generators can be computed from this

Stochastic Simulation

- Random Variables
  - Empirical: fit to data
    - Component failure, Weibull
      - No justification, however fits data and easy to manipulate
  - Function of known distribution
    - Central limit theorem, summation of large number of RV approaches Gaussian
    - Maximum of a number of RV's: Gumbel Distribution

Stochastic Simulation

- Variance Reduction Techniques
  - The greater the variance in results, the less certain we are of their validity.
  - There is a square root of N improvement in variance with N, but not very efficient

Stochastic Simulation

- Correlated Sampling
  - Vary design, but use same pseudo-random sequence of inputs
  - If 2 designs are equally effective, then the simulation results should be very similar

Stochastic Simulation

- Correlated Sampling
  - Suppose Independent RVs are used to compare 2 designs and P_1 and P_2 are performance measures then a figure of merit , D= P_1 - P_2 has variance $\sigma_D{}^2 = \sigma_{P_1}{}^2 + \sigma_{P_2}{}^2$

    If not independent,

    $$\sigma_D{}^2 = \sigma_{P_1}{}^2 + \sigma_{P_2}{}^2 - 2\,\mathrm{cov}(P_1, P_2)$$
  - That is, the more similar the RVs are, the smaller is σ_d

Stochastic Simulation

- Antithetic Variables
  - Simulate using $U_s(0,1)$ RV and save value
  - Generate new sequence based on old by $1\text{-}U_1$, $1\text{-}U_2$, *etc.*
  - If U_s were particularly good sequences then you might expect $1\text{-}U_s$ to be particularly bad sequences

Stochastic Simulation

- Stratified Sampling
  - Subdivide sample space into groups with minimum variance, *e.g*, computer instruction set
    - distribution associated with business applications
    - distribution associated with scientific applications

Stochastic Simulation

- Importance Sampling
  - Devote more simulation runs to the sets of inputs which have greater variance
  - Assures that enough "rare" events occur to make sound statistical inferences

Stochastic Simulation

- Sequential Analysis
  - Adaptively modify sampling sequence, *e.g.*, simulation fails when one bit is set so only test all variations with that bit set

Stochastic Simulation

- Initialization
  - May take long time for startup transients to converge to steady state behavior
  - Analyze transient behavior separately
  - Initialize near steady state values for subsequent analysis

System Stressing

- Often interested in degradation characteristics under increasing load
  - Catastrophic failures
  - Graceful degradation
- Disable certain services to simulate their being fully occupied and not available

Simulation Validation

- Run simple cases for which results are known or calculable and compare results
- Perturb parameters to perform sensitivity analysis
  - Fisher's Law: The more optimized a system is for one environment, the worse it will do in another
- Monte-Carlo: large # of runs, average results

Concurrent Simulation

- Optimistic
  - Run parts concurrently but only check interaction at large time intervals.
    - If an event occurred, backtrack
- Conservative
  - Each parallel process proceeds no further than it can without the possibility of backtracking

VHDL Implementation of FSMs

- Why FSMs?
  - Represent combinational circuits with feedback
  - Represent memory
  - Represent state dependent behavior
  - Are ubiquitous
  - Used for
    - Regular language recognizers
    - Computer Control Units
    - Represent bus/network protocols

FSM Types

- Automata
  - Basic machine with state transitions, but no output
- Moore
  - Output is only a function of present state
- Mealy
  - Output is a function of both present state and present input

FSM Equivalence

- It can be shown that there is a behavioral equivalence between Moore and Mealy Machines
- Any Moore machine can be converted to a behaviorally equivalent Mealy machine and vice-versa

DFA/NDFA

- Deterministic Finite Automata (DFA)
  - aka, completely specified
- Non-Deterministic Finite Automata (NDFA)
  - Not all ((state, input),state) pairs are defined in transition function, δ
  - Transitions based on strings rather than single inputs
  - Different state transitions for same (state, input)

DFA/NDFA

- Any NDFA Can Be Converted to a DFA
- Net Result, Only Need to Study DFA Mealy Machine

- Insert some slides from 331_18/19 here

Inappropriate Use of FSMs

- Combinational Circuits
- Fixed Sequencers (no branching, no feedback)
  - n-phase clock
  - shift registers
  - digital FIR w/o feedback

Sample FSM in VHDL

- Insert more here

End