

## Course Syllabus

*(~10 minute breaks near the top of each hour)  
(Lab time is scheduled for "Lunch & Lab" and again near the end of the day)*

### Day One

#### (1) SystemVerilog Features for Advanced Digital Design

Includes a few short slides to provide a quick review of SystemVerilog resources available to designers.

- Verilog & SystemVerilog Keywords
- SystemVerilog Books & Resources-Includes free IEEE PDF version of SystemVerilog Standard
- SystemVerilog Enhancements Strategy & High-Level Methodology
- You don't have to swallow the whole SystemVerilog elephant all at once.

#### (1a) Review of Important Verilog Coding Guidelines

Important features that most self- and poorly-taught engineers do not understand about SystemVerilog enhancements, and the RTL design issues they are intended to fix. The features are fully explained with historical context to help even experienced designers to avoid common RTL design mistakes.

- reg -vs- wire (the reg-mistake explained by Verilog developer, Phil Moorby).
- blocking -vs- nonblocking usage to avoid RTL simulation race conditions.
- New useful SystemVerilog topics that all designers should understand - each of these topics is required for advanced RTL design techniques shown later in the course:
  - logic type - new near-universal type - versus wire type
  - Only use wire-type for multi-driver and resolved logic designs
  - Typedefs and recommended naming conventions to enhance RTL design
  - Multi-field structs for enhanced design documentation and design reuse
  - Enumerated types (advanced RTL design styles, especially opcodes and FSM design)
  - Basic SystemVerilog packages for designers - especially useful for FSM design
- **LABS:** Multiple SystemVerilog types, typedefs, type-casting and logic labs

#### (2) Latches & Priority Encoders

Improving synthesis is defined as faster clock speeds, smaller logic or both. Detailed description of two synthesis problem areas: latches and priority encoders. Detailed description of the synthesis directives "full\_case" and "parallel\_case", and SystemVerilog replacements, priority & unique.

- Improve synthesis? Make tools work harder or change RTL code? Changing RTL works best.
- Improve synthesis? Think hardware or creative abstract algorithms? Think hardware!
- always blocks & sensitivity lists
- V2K1 @\* and comma-separated sensitivity lists
- Problem area #1 - Generating unexpected latches
- Why do fully coded case statements with case-default still give latches?
- Using initial default assignments to avoid latches and improve synthesis - always works
- Latch inference reports and intentional latch naming conventions

- Problem area #2 - Generating large-slow priority encoders
- Priority encoders from if-else testing of unrelated logic
- Why can case statements produce priority encoders?
- "full\_case parallel\_case," the "evil twins!" - Causes simulation-synthesis mismatches
- SystemVerilog priority & unique - usage guidelines to avoid simulation-synthesis mismatches
- Summary of latch & priority encoder guidelines

### (3) Combinational Logic Design Techniques I - Synthesizable Coding Styles

RTL coding styles for combinational logic, including problems and inefficiencies that arise from poor coding styles. Includes good Verilog-2001 (V2K1) and much better SystemVerilog combinational logic enhancements. Explains reasons behind SystemVerilog enhancements. Numerous combinational labs demonstrate many potential problem areas related to common combinational coding styles.

- Throughout this section - emphasis on fastest designs / smallest area / coding styles to avoid
- New SystemVerilog operators
- Continuous assignments
- Logic specific processes (always\_type blocks) document designer intent
- always\_comb / always\_latch / always\_ff
- Added design checks using always\_type blocks - unknown by most RTL designers
- always @\* -vs- always\_comb and why always\_comb is better
- void functions to document larger combinational blocks
- Best coding style of huge combinational logic? always\_comb & void functions
- Combinational sensitivity issues with always @\*
- Design encapsulation through void functions
- **LABS:** Combinational RTL labs I

### (4) Implicit .\* and .name Port Instantiation

- Implicit port connections can reduce top-level ASIC and FPGA coding efforts by more than 70% and simultaneously enforce greater port type checking. Many engineers fear debugging .\* designs, but this section shows how to expand ports for debug and why .\* enforces better design styles.

- Verilog-2001 positional & named ports
- SystemVerilog .\* implicit ports
- SystemVerilog .name implicit ports
- Implicit port connection rules & comparisons - includes IEEE 1800 latest updates
- Strong port-type checking
- How .\* fixes old Verilog port connection problems
- New debugging techniques - automatic expansion of .\* ports - auto-schematic generation
- Synthesis approach - analyze subblocks first then analyze upper blocks.
- Advantages & disadvantages
- **LAB:** implicit port instantiation labs

## Day Two

### (5) Combinational Logic Design Techniques II - Synthesizable Coding Styles

Additional RTL coding styles for combinational logic, including more problems and inefficiencies that occur from poor coding styles. More combinational labs demonstrate many potential problem areas related to common combinational coding styles.

- Throughout this section - emphasis on fastest designs / smallest area / coding styles to avoid
- Synthesizable and non-synthesizable Verilog constructs
- Bitwise -vs- logical operators
- Tasks, functions, void functions & automatic void functions
- Tri-state drivers
- Bi-directional busses
- **LABS:** Combinational RTL labs II

### (6) Sequential Logic Design Techniques - Synthesizable Coding Styles

This section covers coding styles for sequential logic. Common sequential RTL styles that are problematic. Inferring efficient designs using adders and other large resources is also detailed. Also discusses and includes advantages and disadvantages of instantiation.

- Throughout this section - emphasis on fastest designs / smallest area / coding styles to avoid
- Edge-sensitive sensitivity list
- Basic asynchronous & synchronous resets
- Additional flip-flop coding styles
- Poor synthesis results from mixing flipflop styles in the same always block
- Simulation issue with async set & reset flipflop coding styles
- Simulation/synthesis differences
- Simulation efficiency
- Register banks
- Memories
- Instantiating Blocks
- **LABS:** Sequential RTL labs
- **NOTE** - One of the Labs is a counter with diagnostic carry inputs to accelerate testing of the 16-bit counter.

### (7) Advanced Design Topics - Resource Sharing / Register Rebalancing / Clock Gating / Multi-Cycle Paths / High Speed Design

Multiple advanced topics are shown in this section.

- Large resources
- Resource sharing
- Register re-balancing
- Re-balancing techniques (1) Automated (2) Manual RTL coding
- Latency & throughput design
- Latency & throughput tradeoffs
- Introductory clock-gating techniques
- High-level clock gating / hand coded

- Low-level clock gating / tool insertion
- Multi-cycle paths and where they are used

## (8) Synchronous & Asynchronous Reset Design

Detailed material for selection and usage of synchronous and asynchronous reset design taken from actual design experiences.

- Synchronous vs. asynchronous resets
- Reset removal metastability
- Asynchronous reset synchronizer circuitry
- Reset distribution trees and techniques
- Reset Domain Crossing & synchronization

## (9) Introduction to SystemVerilog Interfaces

Interfaces are a powerful new form of abstraction and this section details how they work for design and verification. This section also discusses when *and* when not to use interfaces.

- Interface usage overview
- Interfaces -vs- structs
- How interfaces work
- 4 requirements for good interface usage
- Interfaces - legal & illegal usage
- Interface constructs
- Interface modports
- **LAB:** simple interface testbench lab

## (10) Finite State Machine (FSM) Design

Fundamental and advanced coding styles for state machines. Includes important considerations for coding designs for easy debug and optimal synthesis. Advantages of putting encodings into a package. Why parameter or enum-type state definitions are used instead of `define. Binary encoded, and efficient onehot coding styles are presented. FSMs with combinational outputs and sequential outputs are also presented.

- Introduction to state machines and 7 different FSM design styles
- Moore & Mealy styles
- Three always block coding style - registered outputs
- Two always blocks implementation - combinational outputs
- Output assignments using always blocks and continuous assignments
- One always block implementation - Inefficient - avoid this style
- Four always block coding style - registered outputs - for improved synthesis results
- Indexed one-hot implementation - registered outputs
- Encoded one-hot implementation - Inefficient - avoid this style
- Output encoded style - registered outputs
- Modified Mealy FSM to register outputs
- Efficiencies

- Labs: State machine labs experimenting with different coding styles and the priority / unique for synthesis

### Day 3

#### (11) Multi-clock Clock Domain Crossing (CDC) using SystemVerilog

Very advanced design techniques from Cliff's award-winning presentations on the efficient implementation of multi-clock CDC techniques. These materials are not specific to SystemVerilog but solutions are shown using SystemVerilog syntax (advanced techniques that all design engineers should know - the techniques you did not learn in college).

- Metastability & synchronizers - synchronizing 1-bit signals
- Passing multiple control signals - synchronizing multi-bit signals or busses
  - Consolidation
  - Edge detection
  - Controlled synchronization - multicycle path formulations (MCP)
  - FIFO synchronizer
  - Gray codes & Gray code counters
- Clock Domain Recovery
- Design partitioning - design & synthesis techniques
  - Naming conventions
  - Synthesis scripting & timing analysis issues
- Simulation issues
  - X-propagation issues
  - Synopsys command for SDF files
  - Multi-SDF files
  - ASIC/FPGA vendor cells and models
  - Simulation model to expose synchronization problems
- LAB: MCP controlled synchronization lab

#### (12) Multi-clock FIFO Design using SystemVerilog

Very advanced design techniques from Cliff's award-winning presentations on the efficient implementation of multi-clock FIFO designs. These materials are not specific to SystemVerilog but solutions are shown using SystemVerilog syntax (advanced techniques that all design engineers should know). Engineers have told Cliff that this FIFO from Cliff's paper is used in many commercial designs.

- Multi-clock FIFO design - large section on design and FIFO issues
- Two different Gray code counter styles
- LAB: 2-clock FIFO lab

### **(13) Design for Reuse / IP Design**

Various techniques and recommendations related to IP design and design reuse.

- Design reuse - start simple / migrate to reuse
- Another look at SystemVerilog interfaces
- Design for reuse for IP:
  - Scale-ability
  - Modularity / partitioning
- Parameterizable designs
  - Smart use of parameters
  - generate statements (added to Verilog-2001 and over-used)
  - Arrays of instance (part of Verilog-1995 / not widely known / better and simpler syntax for contiguous range of logic such as instantiating IO pads for a bus / easily parameterized / better and easier syntax than generate statements)
  - Comparing generates to arrays of instance - pros & cons of each
- Guidelines

### **(14) Design For Test (DFT) Techniques**

Teach the fundamentals of Design For Test (DFT). This section on DFT teaches engineers the basics of DFT and the right questions to ask about the DFT techniques used in a design.

- What is DFT?
- DFT modes of operation
- RTL coding for DFT
- DFT steps
- DFT - where can it go wrong?
- Reset considerations for DFT
- Scan for multi-clock designs