

# COMS31700 Design Verification Hardware Design Languages

Kerstin Eder

(Acknowledgement: Avi Ziv from the IBM Research Labs in Haifa has kindly permitted the re-use of some of his slides.)



Department of  
COMPUTER SCIENCE

## Hardware Design Languages

- Hardware Design Languages were built with simulation in mind
  - Synthesis and other back-end purposes were added at a later stage
- Most popular languages today (both are IEEE standards)
  - VHDL
  - Verilog
- VHDL:
  - Committee-designed language contracted by U.S. (DoD) (ADA-derived)
  - Functional/logic modeling and simulation language
  - Main differentiator from Verilog is types (e.g. records)
- Verilog:
  - Logic modeling and simulation language
  - Started in EDA industry in the 80's now owned by Cadence
  - Donated to IEEE as a general industry standard
  - SystemVerilog (the next generation of Verilog) is designed to improve abstraction of Verilog
    - Abstraction levels
    - Data types
    - Verification constructs
- Verilog vs. VHDL: personal preferences, EDA tool availability, commercial, business and marketing issues.

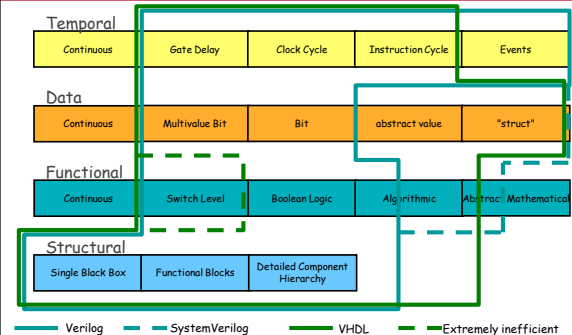
2

## Modeling Levels – Major Dimensions

- Temporal Dimension:**
  - continuous (analog)
  - gate delay
  - clock cycle
  - instruction cycle
  - events
- Data Abstraction:**
  - continuous (analog)
  - bit: multiple values
  - bit: binary
  - abstract value
  - composite value ("struct")
- Functional Dimension:**
  - continuous functions (e.g. differential equations)
  - Switch-level (transistors as switches)
  - Boolean Logic
  - Algorithmic (e.g. sort procedure)
  - Abstract mathematical formula (e.g. matrix multiplication)
- Structural Dimension:**
  - Single block box
  - Functional blocks
  - Detailed hierarchy with primitive library elements

3

## Modeling Levels – Major Dimensions



4

## Verilog for Design Verification

- Assignment calc1 design in Verilog
  - Testbench for calc1 design in Verilog
- Interactive Evita Verilog tutorial (Ch1-4,5-7):**
  - Structure of Verilog modules
  - Verilog signal values: 0, 1, x and z (4-valued logic)
  - Verilog signals:
    - nets (used for "connections", no storage capacity)
    - registers (storage capacity, similar to variables in pgr languages)
  - Verilog external signals:
    - ports (input, output or inout, port connecting rules)
  - Coding styles:
    - Structural
    - Dataflow
    - Behavioural (best for verification)

5

## Continuous Assignment

- Used in Dataflow coding style.**
- Keyword **assign** followed by optional delay declaration
- LHS (target)** can be net (scalar or vector) or concatenation of nets
  - NO registers allowed as target for assignment!
- Assignment symbol: **=**
- RHS** is an expression.
  - assign #4 Out = In1 & In2;**
- Implicit continuous assignment: wire x = ...;**
- Conditional assignment:**
  - assign Out = Sel ? In1 : In0;**
  - If Sel is 1 then In1 is assigned to Out; if Sel is 0 then Out is In0.
  - If Sel is x/z, evaluate both In1 and In0, if they are the same then Out is assigned this value, otherwise x/z.

6

## Continuous Assignment: Execution

- Continuous assignments are **always active**.
- Concurrency:**
  - When any of the operands on RHS changes, assignment is evaluated.
  - Several assignments can be executed concurrently.
  - Race conditions can occur!**
    - Two or more assignments, which operate on the same data, read and write the data concurrently.
    - Result, which might be erroneous, depends on which assignment does what when.
- Delays** specify time between change of operand on RHS and assignment of resulting value to LHS target.
  - assign #4 Out = In1 & In2;**

7

## Behavioural Coding Style

- Most advanced coding style: flexible and high-level
  - closest to programming languages
  - allows use of conditional statements, case statements, loops, etc.**Best for verification, but by no means ideal...**
- Behaviour:**
  - Actions a circuit is supposed to perform when it is active.
- Algorithmic description:** Need "variables" similar to PLs!
  - Abstraction of data storage elements - register objects:
    - reg R**; one bit register - default value x before first assignment
    - time T**; can store/manipulate simulation time
    - integer N**; by default at least 32 bit - stores values signed
    - real R**; default value is 0
    - [Other data types, e.g. arrays exist, but are out of the scope of this introduction.]

8

## Mux421: Behavioural Coding Example

```
module mux421_behavioural (Out, In0, In1, In2, In3, Sel0, Sel1);
output Out;
input In0, In1, In2, In3, Sel0, Sel1;
reg Out;
always @ (Sel1 or Sel0 or In0 or In1 or In2 or In3)
begin
    case ({Sel1,Sel0})
        2'b00 : Out = In0;
        2'b01 : Out = In1;
        2'b10 : Out = In2;
        2'b11 : Out = In3;
        default : Out = 1'bx;
    endcase
end
endmodule // mux421_behavioural
```

10

## Behavioural Blocks

- initial** and **always**
  - Can't be nested.
  - Block containing several statements must be grouped using:
    - begin ... end** (sequential) or
    - fork ... join** (concurrent)
- initial** block:
  - Used to initialise variables (registers).
  - Executed at (simulation) time 0. Only once!
- always** block:
  - Starts executing at time 0.
  - Contents is executed in infinite loop.
    - Means: Executing repeats as long as simulation is running.
  - Multiple blocks are all executed **concurrently** from time 0.

11

## Assignment in Behavioural Coding

Assignment in behavioural coding style is **procedural**:

- LHS (target) must be a register (reg, integer, real or time) - not a net, a bit or part of a vector of registers.
- NO assign keyword!**
- Must be contained within a behavioural (i.e. **initial** or **always**) block.
- NOT always active!**
  - Target register value is only changed when procedural assignment is executed according to **sequence** contained in block.
- Delays:** indicate time that simulator waits from "finding" the assignment to executing it.

12

## Blocking Assignment

(... as opposed to continuous assignment from dataflow coding style.)

```
reg A;
reg [7:0] Vector;
integer Count;
initial
begin
    A = 1'b0;
    Vector = 8'b0;
    Count = 0;
end
```

- Sequential initialisation assignment.

13

## Timing Control Evaluation

`#5 C = #10 A+B;`

Assignment delay      Intra-assignment delay

1. Find procedural assignment
2. Wait 5 time units
3. Perform A+B
4. Wait 10 time units
5. Assign result to C

- So, what is the difference between:
  - `#10 C = A+B` and
  - `C = #10 A+B`?

14

## Events and Wait

- Events** mark changes in nets and registers, e.g. raising/falling edge of clock.
  - @ **negedge** means from any value to 0
  - @ **posedge** means from any value to 1
  - @ **clk** always activates when clock changes
- Wait statement:**
  - `wait (condition) stmt;`
    - `wait (EN) #5 C = A + B;`
      - waits for EN to be 1 before `#5 C = A + B;`
- Use **wait** to block execution by not specifying a statement!
  - `wait (EN); ...`

15

## Sensitivity List

```
always @ (posedge Clk or EN)
begin ... end
```

- Allows to **suspend always blocks**.
- Block executes and suspends until signal (one or more) in **sensitivity list** changes.
- NOTE:** **or** is used to make statement **sensitive to multiple signals or events**.
- (Don't use sensitivity list to express a logical condition!)
- Common mistake:
  - Forgetting to add relevant signals to sensitivity list!

16

## Non-blocking Assignments

- Concurrency** can be introduced into sequential statements.
  - Delay is counted down before assignment,
  - BUT **control is passed to next statement immediately**.
- Non-blocking Assignments** allow to model multiple concurrent data transfers after common event.
- A blocking assignment would force sequential execution.

`A <= #1 1; B <= #2 0; (non-blocking)`

```
A x 1 1 1
B x x 0 0
Time: 0 1 2 3
```

`A = #1 1; B = #2 0; (blocking)`

```
A x 1 1 1
B x x x 0
Time: 0 1 2 3
```

17

## Approaches to Assignment - I

```
reg [7:0] MyReg;
initial
fork
#50 MyReg = 8'hFF;
#100 MyReg = 8'h01;
#150 MyReg = 8'h2F;
#200 MyReg = 8'h00;
#250 $finish;
join
```

- Concurrent blocking (=)**

```
Time:      0   50  100 150 200 250
MyReg[7:0] XX  FF  01  2F  00  00
```

Important when driving input into a DUV in a testbench!

18

## Approaches to Assignment - II

```
reg [7:0] MyReg;
initial
begin
    MyReg <= #50 8'hFF; // pass control, wait, assign
    MyReg <= #50 8'h01;
    MyReg <= #50 8'h2F;
    MyReg <= #50 8'h00;
    #250 $finish;
end
```

**Race Condition!**

- Sequential non-blocking (<=)**

```
Time:      0   50  100 150 200 250
MyReg[7:0] XX  ??
```

Important when driving input into a DUV in a testbench!

19

## Approaches to Assignment - III

```
reg [7:0] MyReg;
initial
begin
    MyReg <= #50 8'hFF; // pass control, wait, assign
    MyReg <= #100 8'h01;
    MyReg <= #150 8'h2F;
    MyReg <= #200 8'h00;
    #250 $finish;
end
```

- Sequential non-blocking (<=)

```
Time:      0  50 100 150 200 250
MyReg[7:0] XX FF 01 2F 00 00
```

Important when driving input into a DUV in a testbench!

20

## Approaches to Assignment - IV

```
reg [7:0] MyReg;
initial
begin
    #50 MyReg = 8'hFF; // wait, assign, pass control
    #50 MyReg = 8'h01;
    #50 MyReg = 8'h2F;
    #50 MyReg = 8'h00;
    #250 $finish;
end
```

- Sequential blocking (=)

```
Time:      0  50 100 150 200 250
MyReg[7:0] XX FF 01 2F 00 00
```

Important when driving input into a DUV in a testbench!

21

## HDL vs. Programming Languages

### 3 major new concepts of HDLs compared to PLs:

- Connectivity:**
  - Ability to describe a design using simpler blocks and then connecting them together.
- Time:**
  - Can specify a delay (in time units of simulator): (WHY?)
    - and #2 (Y3, ln3, Sel1, Sel0);
- Concurrency is always assumed!** (for structural style this is)
  - No matter in which order primitives/components are specified, a change in value of any input signal **activates** the component.
  - If 2 or more components are activated **concurrently**, they perform their actions **concurrently**.
  - Order of specification does not influence order of activation!
  - (NOTE: Statements inside behavioural blocks may be sequential -more later.)

22

## Tasks and Functions

## Tasks and Functions

- Both are **purely behavioural**.
  - Can't define nets inside them.
  - Can use logical variables, registers, integers and reals.
- Must be declared within a module.**
  - Are local to this module.
  - To share tasks/functions in several modules, specify declaration in separate module and use **'include** directive.
- Timing (simulation time)**
  - Tasks:**
    - No restriction on use of timing; engineer specifies execution.
  - Functions:**
    - Execute in ZERO sim time units; no timing/event control allowed.

24

## Comparing Tasks with Functions

	Tasks	Functions
Timing	can be non-zero sim time	execute in 0 sim time
Calling other tasks or functions	no limit; may enable functions	may not call tasks but may call another function <b>No recursion!</b>
Arguments	any number; any type; can't return result	at least one input; no output/inout; always results in single return value
Purpose	modularize code	react to some input with single response; only combinatorial code; use as operands in expressions

25

## Example Task

```
task factorial;
  output [31:0] f;
  input [3:0] n;
  integer count; // local variable
  begin
    f = 1;
    for (count=n; count>0; count=count-1)
      f = f * count;
  end
endtask
```

- **Invoke task:** `< task name > (list of arguments);`
  - Declaration order determines order of arguments when task is called!

26

## Example Function

```
function ParityCheck;
  input [3:0] Data;
  begin
    ParityCheck = ^Data; // bit-wise xor reduction
  end
endfunction
```

- **Result** is by default a 1 bit register assigned to implicitly declared local variable that has same name as function.
- **Function calls:**
  - Are either assigned to a variable, or
  - occur in an expression that is assigned to a variable,
  - or occur as an argument of another function call.

27

## System Tasks and Functions

- More than 100 Verilog system tasks/functions.
  - (See Evita Verilog Reference Guide for more information.)
- Can be used in any module without explicit include directive.
- **Syntax:** `$< keyword >`
- **Most important tasks for verification:**
  - `$display`, `$monitor`
  - `$time`, `$stop`, `$finish`
  - (Also with files: `$fopen`, `$fdisplay`)

28

## Summary

- Evita Verilog Tutorial [Ch1-7]
- **Verilog HDL IEEE Standard 1364**
  - Signals: internal and external (ports)
  - Different coding styles:
    - structural
    - dataflow
    - behavioural
- **HDLs: Connectivity, Time and Concurrency**
- BOOK: Verilog HDL by Samir Palnikar [in QB Library]
- **Next:**
  - Introduction to Assignment 1!

29