# **Hardware Description Languages**

Most common HDLs used in practice are:

Verilog (see interactive Evita\_Verilog tutorial) and VHDL

#### Historically 1. Design with Boolean Equations

- · basic building blocks: gates and flip-flops
- $\bullet$  express relationship between input and output with logic equations

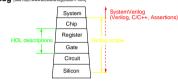
#### impractical for large designs

- Evolution to 2. Schematic-Based Design
   allows additional logic blocks not just gates and flip-flops
- enables formation of hierarchical designs and graphical representation.
- ++ (used to be) very popular easy to use
- \_\_ but too limited for increasing design complexity (density)
- Schematics with >6000 gates are incomprehensible!

erstin Eder 1 University of Bristol October 24, 2008

## **Different Levels of Description**

NEW [June 2002]: Accellera approves next-generation of Verilog: SystemVerilog | See http://www.accellera.org/press17.html]



"SystemVerilog represents the most significant advancement to the Verilog language, since its acceptance as an IEEE standard," said Dennis Brophy, Accellera Chairman. "It takes Verilog to the next abstraction level - the architecture and behavioral design of a system - and adds to it powerful assertions that allow designers and system architects to build and verify full systems."

Kerstin Eder 2– iv University of Bristol October 24, 2008

# Verilog Signal Values

In Verilog, signals have a finite set of values: (SIMPLIFICATION!)

- 0. 1 represent False and True (as normal)
- X or x stands for "unknown", i.e. either 0 or 1 or z
- Z or z high impedance (equivalent to cutting off the signal source)
- Distinguish different signal strengths, depending on technological considerations.

#### \*Verilog standard defines truth tables for 4-valued logic.

			-												_	
and	0	1	Х	z	or	0	1	Х	z	xor	0	1	х	z	not: input	output
0	0	0	0	0	0	0	1	Х	х	0	0	1	х	х	0	1
1	0	1	Х	Х	1	1	1	1	1	1	1	0	х	Х	1	0
Х	0	х	х	х	х	х	1	х	х	х	х	х	х	х	Х	х
Z	0	х	х	х	Z	Х	1	Х	х	Z	Х	х	х	х	Z	Х

(See Evita\_Verilog tutorial Chapter 3 for more details.)

Kerstin Eder 6 University of Bristol October 24, 2008

# **Verilog External Signals**

External signals are called ports.

• Declare direction and type (width) of ports.

Direction: Ports can be input, output or inout.

■ What would be an example (use) for a bi-directional port?

### Signal specification:

key\_word {vector\_range} signal\_name{, signal\_name}\*;
where key word is:

- signal\_type for internal signals, i.e. wire or reg
- · direction for ports (external signals)

# **Different Levels of Description**



Gerstin Eder 2— i University of Bristol October 24, 2008

## Verilog HDL

- HDL developed in 1984/5 by Philip Moorby.
- Simple, intuitive and effective way of describing digital circuits.
- Intended for modelling, simulation and analysis.
- Language became property of Gateway Design Automation.
- Cadence Design Systems acquired Gateway Design Automation.
- Verilog is a registered trademark of Cadence Design Systems, Inc.
- Language open to public since 1990.
- Since 1995: IEEE Standard 1364

Verilog vs. VHDL: personal preferences, EDA tool availability, commercial, business and marketing issues.

\*Read more in interactive Evita\_Verilog tutorial!

Kerstin Eder 3 University of Bristol October 24, 2008

# Verilog Signals I

Nets: (most common declaration with keyword wire)

- Represent physical connections between hardware elements.
- No storage capacity.
- Value is determined by driver of net or high impedance (i.e. net is not connected to a driver).

#### Registers:

- Can store values even when disconnected.
- · Assigned values are kept until new value is assigned.
- \*Same role as variables in programming languages.
- · Different to digital registers, which are clocked.
- Verilog reg has no clocking signal!

Kerstin Eder 7 University of Bristol October 24, 200

#### ₩ By default, Verilog assumes all ports are nets of type wire!

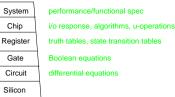
- · Wires must be driven at all times.
- Not good enough to hold value of output signals until next clock.
- Can define an output as reg (instead of wire).
- -registered output port needs additional declaration!

■ module processor (Clk, Reset, RW, Data, Addr); input Clk, Reset; output RW; inout [15:0] Data; // inout ports cannot be registered output [19:0] Addr; reg RW reg [19:0] Addr;

· Only output ports can be declared as registers!

# **Different Levels of Description**

# Behaviour:



For Verification we are interested in specifying Behaviour.

Kerstin Eder 2- ii University of Bristol October 24, 2008

# System Descriptions in Verilog

A Verilog module represents a system



Module definitions cannot be nested!

Kerstin Eder 4 University of Bristol October 24, 2008

# Verilog Signals II

Both registers and nets can be:

- Scalar: single line signals clock signal
- wire clk;
- Vector: multiple line signals busses
  - transmits signals comprising of several logic values
  - bits are individually indexed and ranked [msb: lsb]

■ wire [3:0] bus\_1 4 bit wide bus; msb @ 3, lsb @ 0
■ wire [0:7] bus\_2 8 bit wide bus; msb @ 0, lsb @ 7

■wire [0:7] bus\_2 8 bit wide bus; msb @ 0, lsb @ 7
■wire [-2:2] bus\_3 5 bit wide bus; msb @ -2, lsb @ 2

Syntax for bit select on vectors: vector\_name[bit\_number] or

vector\_name[msb : lsb]

bus\_2[0:1]

(See Evita\_Verilog tutorial Chapter 5 for more details.)

Kerstin Eder 8- i University of Bristol October 24, 200

# Coding Styles Structural style:

Structural style

- Use primitives and lower-level module instantiations.
   (-logic gates and Verilog primitives)
- ++ Very close to physical implementation.
- ++ Easy (straightforward) synthesis.

#### Dataflow style:

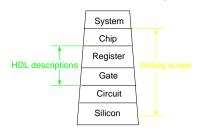
• Define output signals in terms of input signal transformations.

#### Behavioural style:

- Specify algorithmically the expected behaviour of the system.
- Close to natural language description.
- \_\_ Most difficult to synthesise!

Best for verification.

# **Different Levels of Description**



Verilog covers several design levels.

Kerstin Eder 2- iii University of Bristol October 24 2008

### Verilog: The very Basics

- Verilog is case-sEnsiTIVE!
- The <module\_name> is the formal identifier of a module.

  \$ (Not the name of the file that contains the module.)
- The <port\_list> contains names of ports only.
- · Compiler directives:
- 'include
- Can be anywhere in the code.
- No item can be used before it is declared.

Kerstin Edge 5 University of Briefel October 24 2008

# Verilog Signals II

Both registers and nets can be:

■wire [-2:2] bus 3

- Scalar: single line signals clock signal
- wire clk;
- Vector: multiple line signals busses
- transmits signals comprising of several logic values
- bits are individually indexed and ranked [msb: lsb]

■ wire [3:0] bus\_1 4 bit wide bus; msb @ 3, lsb @ 0
■ wire [0:7] bus\_2 8 bit wide bus; msb @ 0, lsb @ 7

5 bit wide bus; msb @ -2, lsb @ 2

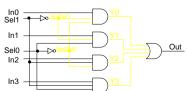
Syntax for bit select on vectors:  $vector\_name[bit\_number]$  or  $vector\_name[msb:lsb]$ 

■bus\_2[0:1] two most significant bits of bus\_2

(See Evita\_Verilog tutorial Chapter 5 for more details.)

Kerstin Eder 8- ii University of Bristol October 24, 2008

# Structural Coding Style



Systems are composed of lower-level module instantiations and primitives.

Use internal  $\boldsymbol{wires}$  to connect sub-components.

BEWARE of large number of details!

Eder 9 University of Bristol October 24, 2008 Kerstin Eder 10 University of Bristol October 24, 2008 Kerstin Eder 11 University of Bristol October 24, 2008 Kerstin Eder 12 University of Bristol October 25, 2008 Kerstin Eder 12 University of Bristol October 26, 2008 Kerstin Eder 15 University of Bristol October 26, 2008 Kerstin Eder 15 University of Bristol October 26, 2008 Kerstin Eder 16 University of Bristol October 26, 2008 Kerstin Eder 15 University of Bristol October 26, 2008 Kerstin Eder 16 University of Bristol October 26, 2008 Kerstin Eder 17 University of Bristol October 26, 2008 Kerstin Eder 18 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 University of Bristol October 26, 2008 Kerstin Eder 19 Univ

### Structural Coding Style

```
module mux_421_structural (Out, In0, In1, In2, In3, Sel1, Sel0);
 output Out.
 input InO. Inl. In2. In3. Sell. Sel0:
 wire NotSell. NotSell:
 wire VO. V1. V2. V3:
 not (NotSel0, Sel0);
 not (NotSell, Sell):
 and (Y0, In0, NotSell, NotSel0):
 and (Y1, In1, NotSell, Sel0);
 and (Y2, In2, Sell,
                       NotSel0);
 and (Y3, In3, Sell,
                       Sel0):
 or (Out, Y0, Y1, Y2, Y3);
endmodule // mux421 structural
How does this work?
```

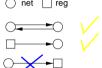
University of Bristol October 24, 2008

## Port Connecting Rules

\*Set of rules defines how to connect ports.

input and inout ports:

- Must be of type net both internally and externally. output ports:
- Must be of type reg or net internally; must be connected to a net externally



All ports can be scalars or vectors.

# Continuous Assignment - Syntax

- Keyword assign followed by optional delay declaration
- LHS 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.

# **Behavioural Coding Style**

- ++ most advanced coding style: flexible and high-level
- ++ closest to programming languages
- ++ allows use of conditional statements, loops, case 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 1

### Structural Coding Style

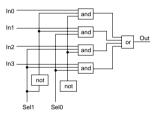
```
module mux 421 structural (Out, In0, In1, In2, In3, Sel1, Sel0);
 output Out.
 input InO. In1. In2. In3. Sell. Sel0:
 wire NotSell. NotSell:
 wire VO. V1. V2. V3:
 not FirstNot (NotSel0, Sel0); // optional name of instance or
 not SecondNot (NotSell, Sell): // Design Structure [ModelSim]
     #2 (Y0. In0. NotSell. NotSel0):
 and
 and #2 (Y1, In1, NotSell, Sel0);
 and #2 (Y2, In2, Sell, NotSel0);
 and #2 (Y3, In3, Sell, Sel0);
 or #4 OutputOr (Out, Y0, Y1, Y2, Y3); // optional delay
```

How does this work?

endmodule // mux421 structural

University of Bristol October 24, 2008

### **Dataflow Coding Style**



Define output signals in terms of input signal transformations.

• Use logical view: logical operators and assignment statements.

# Continuous Assignment - Syntax

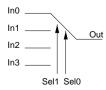
- Keyword assign followed by optional delay declaration
- LHS can be net (scalar or vector) or concatenation of nets
- NO registers allowed as target for assignment!
- Assignment symbol: =
- RHS is an expression.
- assign #4 Out = In1 & In2;
- Implicit continuous assignment: wire x = ...;

#### ■ 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.

# **Behavioural Coding Style**

Mux 4 to 1 behaviour:



Behavioural specification encapsulated in initial and always behavioural blocks.

# **HDLs 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.
- - Can specify a delay (in time units of simulator): (WHY2)
- and #2 (Y3, In3, Sel1, Sel0); (for structural style this is)
- Concurrency is always assumed! - 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)

# **Dataflow Coding Style**

```
module mux421_dataflow (Out, In0, In1, In2, In3, Sel0, Sel1);
 input InO. In1. In2. In3. SelO. Sell:
   (~Sel1 & ~Sel0 & In0)
   (~Sel1 & Sel0 & In1)
   (Sel1 & ~Sel0 & In2)
  (Sel1 & Sel0 & In3);
```

endmodule // mux421 dataflow

\* Verilog has various operators: logical, bit-wise, reduction, arithmetic, conditional and concatenation.

(See Reference Guide of Evita\_Verilog tutorial for more details.)

### **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.
- Delays specify time between change of operand on RHS and assignment of resulting value to LHS target.

```
■assign #4 Out = In1 & In2;
```

- 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.

Hierarchical Design

Building blocks for complex designs: Primitives and Modules.

Modules can be instantiated in same way as primitives.

Simplest parameter passing: In order of ports in port list.

\*Important for Verilog Testbench Architecture.

· Constants are by default at least 32 bit wide.

■ -8'd10 equivalent to -(8'd10)

■ -10 2's complement of decimal 10

(- Ports may remain unconnected; just leave position blank.)

- To get hex, oct, bin precede constant with 'h. 'o. 'b.

■8'b1 represents 0000 0001 - padded with zeros

NOTE: Registers store negative values as 2's complement!

Instead of hard-coded constants, use parameter declarations:

- To down-size a constant precede it with positive integer.

To enhance readability, use underscores: 186\_444\_189 is 186444189!

■ parameter BUSSIZE = 8; and then reg [BUSSIZE-1:0] Data Bus;

• Modules are more flexible than primitives:

- can have arbitrary number of ports.

· Modules can instantiate other modules

⇒ Multi-level hierarchical designs

· Modules cannot be nested!

Verilog Constants

· Constants are by default decimal.

- Can be of any complexity, and

- \* 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;

# Mux421: Behavioural Coding Example

```
module mux421_behavioural (Out, In0, In1, In2, In3, Sel0, Sel1);
 output Out:
 input InO, In1, In2, In3, SelO, 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'bl1 : Out = In3;
    default : Out = 1'bx;
```

endmodule // mux421 behavioural

# Behavioural Blocks - initial and always

- · Can't he 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.
- Multiple blocks are all executed concurrently from time 0.

# 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:

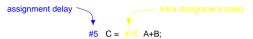
- Used to initialise variables (registers).
- Executed at (simulation) time 0. Only once!

#### 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.

University of Bristol October 24, 2008

# **Timing Control**



- 1. Find procedural assignment
- 2. Wait 5 time units
- 3. Perform A+B
- 4. Wait 10 time units

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

5. Assign result to C

University of Bristol October 24, 2008

# Approaches to assignment

```
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 (=)



\*Important when driving input into a DUV in a testbench!

# 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 ONE sim time unit; no timing/event control allowed

# Assignment in Behavioural Spec

#### Assignment 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
- . Delays: indicate time that simulator waits from "finding" the assignment to executing it.

#### **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;
```

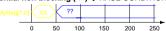
₩ Use wait to block execution by not specifying a statement!

■ wait (EN); ...

# Approaches to assignment

```
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;
```

#### Sequential non-blocking (<=) ₩ RACE CONDITION!



\*Important when driving input into a DUV in a testbench!

# Comparing Tasks and Functions

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

# **Blocking Assignment**

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

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

\* Sequential initialisation assignment.

# 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!

# Approaches to assignment

```
reg [7:0] MyReg;
initial
hegin
 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 (<=)



\*Important when driving input into a DUV in a testbench!

# **Example Task**

```
task factorial;
 output [31:01 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

University of Bristol October 24, 2008

### **Behavioural Constructs for Coding**

#### Conditionals:

if (expression\_true) true\_branch; else false\_branch;

# Case

University of Bristol October 24, 2009

```
case ({_,...,_})
pattern: ...;
default : ...;
endcage
```

Loops: forever, repeat, while, for

See Verilog reference card for syntax!

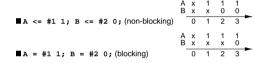
# Non-blocking Assignments

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 seguential execution.

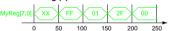


University of Bristol October 24 2005

# Approaches to assignment

```
reg [7:0] MyReg;
initial
hegin
 #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 (=)



\*Important when driving input into a DUV in a testbench!

# **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.

# **System Tasks/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. System tasks facilitate synthesis, but are not synthesizable.

Syntax: \$< keyword >

#### Most important tasks for verification:

- \$display, \$monitor
- \$time, \$stop, \$finish
- (• Also with files: \$fopen, \$fdisplay)

# **Summary**

Evita\_Verilog Tutorial [Ch1-7] in (less than) 2\*50min!

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!

Kerstin Eder 37 University of Bristol October 24, 2008 Kerstin Eder 38 University of Bristol October 24, 2008