(T)EE2026

Digital Fundamentals

Introduction to Verilog

Massimo Alioto

Dept of Electrical and Computer Engineering

Email: ***massimo.alioto@nus.edu.sg***

Outline

- Introduction: Hardware Description Languages
- Introduction to Verilog
  - modules, inputs, outputs
  - operators
  - definition of Boolean expressions

Hardware Description Languages

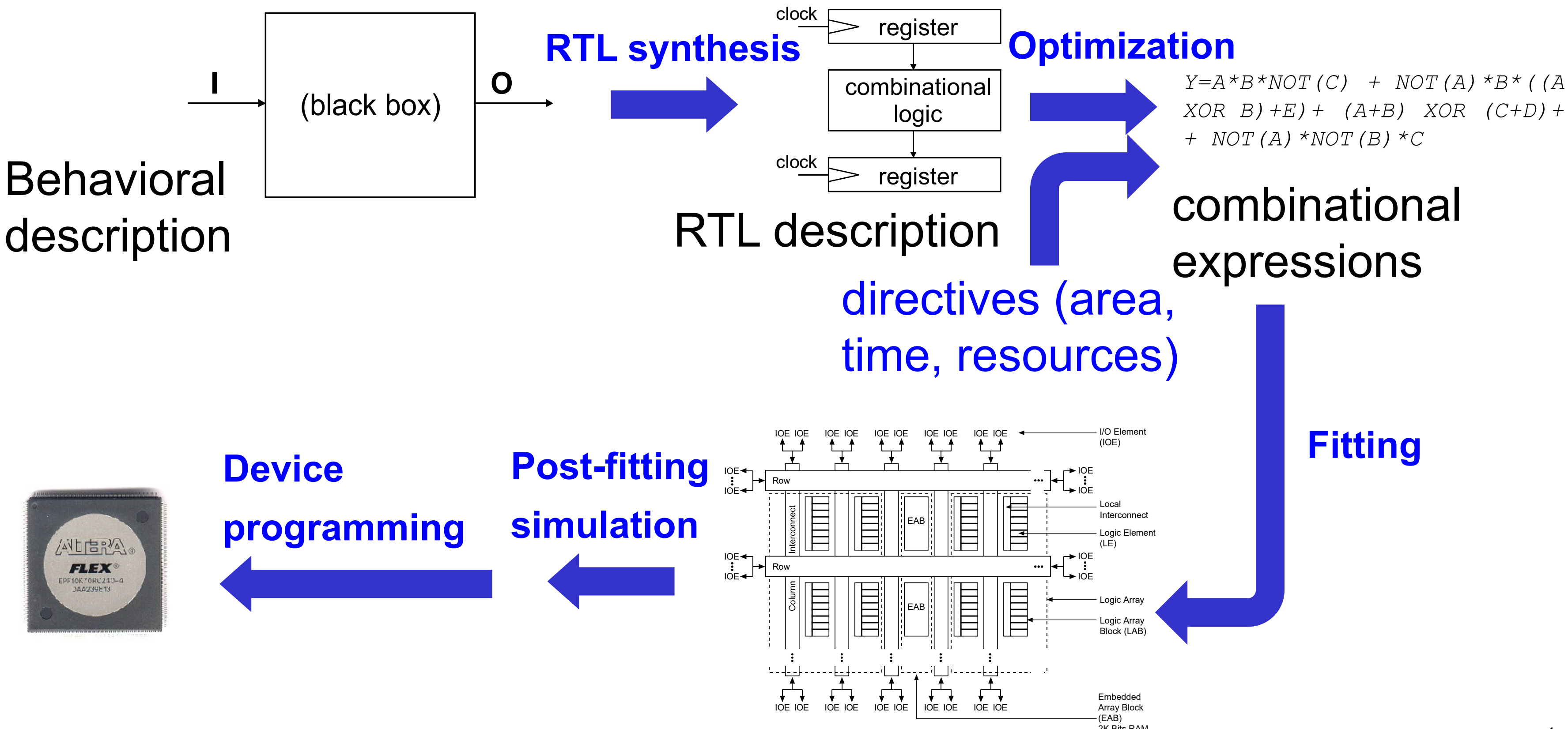
- HDL = software programming language to model the intended operation of a piece of hardware
- HDL vs programming languages

HDLs	Programming languages
• concurrent execution	• sequential execution
• account for timing	• no timing information
• designed for HW modeling	• permit HW modeling
• designed for HW synthesis	• can be potentially used to synthesize HW (difficult)

- Why using HDLs? Same language to...
  – describe, verify/debug, designing a digital system

Automated Design Flows

- Automated translation of HDL into programmable chip design (all steps are iterative)



Verilog

- Verilog was developed by Gateway Design Automation as a simulation language in 1983
- Cadence purchased Gateway in 1989 and placed the Verilog language in the public domain. Open Verilog International (OVI) was created to develop the Verilog language as an IEEE standard
- Verilog is a fairly simple language to learn, especially if you are familiar with C
- References
  - D. Harris, S. Harris, Digital Design and Computer Architecture (1st ed.), Morgan Kaufmann
  - *http://www.asic-world.com/verilog/verilog_one_day.html*

Commenting Code in Verilog

- Single line comments:
  - begin with "//" and end with a carriage return
  - may begin anywhere on the line
- Multiple line comments:
  - begin with "/*" and end with a "*/"
  - may begin and end anywhere on the line
  - everything in between is commented out
- Coding style tip
  - use single line comments for comments
  - reserve multi-line comments for commenting out a section of code

Example

```
module pound_one;
reg [7:0] a,a$b,b,c; // register declarations
reg clk;

initial
 begin
       clk=0;  // initialize the clock
       c = 1;
       forever  #25 clk = !clk;
 end
/*  This section of code implements
     a pipeline */
always @ (posedge clk)
 begin
       a = b;
       b = c;
 end
endmodule
```

Identifiers in Verilog

- Verilog is case sensitive

  Variable CASE_SENSITIVE = 0

  Variable case_sensitive = 1

  different variables

- Identifiers = names assigned by the user to Verilog objects (modules, variables, tasks...)
- Contain any sequence of letters, digits, a dollar sign '$' , and the underscore '_' symbol.
  - first character must be a letter or underscore
  - escaped identifiers start with a backslash (\) and end with white space
  - examples: legal_identifier, _OK,OK_, OK_123, CASE_SENSITIVE, case_sensitive; $_BAD,123_BAD

Logic Values

- Verilog has 4 logic values:
  - ‘0’ represents zero, low, false, not asserted.
  - ‘1’ represents one, high, true, asserted.
  - ‘z’ or ‘Z’ represent a high-impedance value, which is usually treated as an 'x' value.
  - ‘x’ or ‘X’ represent an uninitialized or an unknown logic value--an unknown value is either '1' , '0' , 'z' , or a value that is in a state of change.

Data Types

Nets

- physical connections between devices
- **wire** and **tri** (identical, typically: distinguish tristate nodes)
- no memory: continuously assigned (driven) by an assignment

```
wire a,b;                    // scalar wires
wire [7:0] in_bus;           // multi-bit wire (bus)
```

Reg

- **reg**: storage devices, variables
- on LHS of an assignment, reg is updated immediately and holds its value until changed again

```
reg a;                       // scalar reg variable
reg [7:0] in_bus;            // vectored reg variable
```

Parameters

- constants

```
parameter [2:0] a = 1; // 3-bit (little endian, [0:2] ok – big endian)
parameter width = 8; // default width for parameterizable design
```

Numbers

- Constant numbers are integer or real constants .
- Integers may be sized or unsized
  - Syntax: <size>'<base><value> where:
    - <size> is the number of <u>bits</u>
    - <base is b or B (binary), o or O (octal), d or D (decimal), h or H (hex)
    - <value> is 0-9 a-f A-F x X z Z ? _
    - Examples: 2'b01, 6'o243, 78, 4'ha
- Default radix is decimal, i.e. 1=1'd1
- underscores (_) are ignored (use them like commas)
- When <size> is less than <value> - the upper bits are truncated, e.g. 2'b101->2'b01, 4'hfcba->4'ha

Examples

• 3.14	decimal notation
• 6.4e3	scientific notation for 6400.0
• 16'bz	16 bit z (z is extended to 16 bits)
• 83	unsized decimal
• 8'h0	8 bits with 0 extended to 8 bits
• 2'ha5	2 bits with upper 6 bits truncated (binary equivalent = 01)
• 2_000_000	2 million
• 16'h0x0z	16'b0000xxxx0000zzzz

Operators

- **Verilog operators (in increasing order of precedence)**
  - ?: (conditional – cond_expr ? true_expr : false_expr)
  - || (logical or – used when checking conditions)
  - && (logical and – used when checking conditions)
  - | (bitwise or)
  - ~| (bitwise nor)
  - ^ (bitwise xor)
  - ^~ (or ~^) (bitwise xnor, equivalence)
  - & (bitwise and)
  - ~& (bitwise nand)
  - == (logical) != (logical) === (case – including x and z) !== (case – including x and z)
  - < (lower than)
  - <= (lower than or equal to)
  - > (greater than)
  - >= (greater than or equal to)
  - << (shift left)
  - >> (shift right)
  - + (addition), - (subtraction), * (multiply), / (divide), % (modulus)

if bitwise operands have different bit width, the shorter one is zero-extended in MSBs

Code Structure: Modules

- The **module** is the basic unit of code in the Verilog language
  - describes any block of HW with inputs/outputs
  - includes interfaces, functionality, timing
- Example

```
module holiday_1(sat,sun,weekend);
  input sat, sun;
  output weekend;
  assign weekend = sat | sun;
endmodule
```

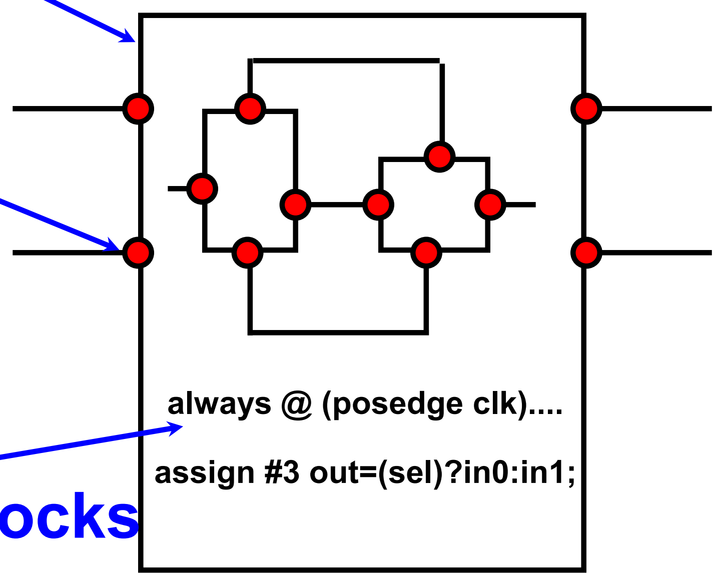
```
module name (port_names);
    module port declarations
    data type declarations
    procedural blocks
    continuous assignments
    user defined tasks & functions
    primitive instances
    module instances
    specify blocks
endmodule
```

Pictorial Summary of Module Structure

Module

Ports
interfaces

Body
Instances
Concurrent blocks
functionality



Covers different levels of abstraction
behavioral vs structural description

Design partitioning
manages hierarchy (sub-modules), enables modularity (reusable) and regularity (many times)

Module Port Declaration

- Scalar (1-bit) port declaration
  - *port_direction port_name*, *port_name*...;
- Vector (multiple-bit) port declaration
  - *port_direction* [*port_size*] *port_name*, *port_name*...;

  - *port_direction*: input, inout (bi-directional) or output
  - *port_name*: legal identifier
  - *port_size*: is a range from [msb:lsb]

```
input a, into_here, george;          // scalar ports
input [7:0] in_bus, data;            //vectored ports
output [31:0] out_bus;               //vectored port
inout [maxsize-1:0] a_bus;           //parameterized port
```

Port Connection Rules

- Inputs
  - internally, must be of *net* data type
  - externally, may be connected to *reg* or *net* data type

inputs need to be specified continuously (otherwise, value would be unclear)

- Outputs
  - internally may be of *net* or *reg* data type
  - externally must be connected to a *net* data type.

output may be generated by combinational or register

- Inouts (most restrictive btwn inputs and outputs)
  - internally must be of *net* data type
  - externally must be connected to a *net* data type

Module Instantiation

- A module may be instantiated within another module, there may be multiple instances of the same module
- ports of instances are either by order or by name
  - use by order unless there are lots of ports, by name for libraries and other peoples code (cannot mix the two in one instantiation)

syntax for instantiation with port order:
module_name instance_name **(signal, signal,...);**

just use same order as module of the instance

syntax for instantiation with port name:
module_name instance_name **(.port_name(signal), .port_name (signal),...**);

explicit original names (+ actual signal)

```
module example (a,b,c,d);
input a,b;
output c,d;
. . . .
endmodule

example ex_inst_1(in_1, in_2, w, z);
example ex_inst_2(in_1, in_2, , z);  // skip a port (sets input to default value specified
within the instantiated module – ex.: input c = 1;)
example ex_inst_3 (.a(w), .d(x), .c(y), .b(z)); // w, x, y and z = wires around instance
```