Verilog introduction

Opriţoiu Flavius
flavius.opritoiu@cs.upt.ro

October 2, 2024

Introduction

Objectives:

- Design combinational circuits using Verilog

Reading:

- Lukasz Strozek: "Verilog Tutorial - Edited for CS141", Laboratory notes, [Stro05]

Verilog Hardware Description Language (HDL):

- Describe hardware realizations in a textual representation
- Permits designing a system at different abstractions: algorithm versus transistor level
- Permits model verification before integrating a digital device
- Synthesis tools translate Verilog models into hardware realizations

Verilog modules

Verilog description of a digital system consists of *modules*.

A Verilog module definition consists of:

- module's name, preceded by the `module` keyword
- a list of module's inputs and outputs, enclosed in parentheses
- module's implementation
- the final `endmodule` keyword

Module's inputs and outputs are collectively referred as *ports*.

Throughout this laboratory, a module's implementation can include the following type of statements:

- *instances* of other Verilog modules
- continuous assignments, introduced by the `assign` keyword
- `always` blocks

Continuous assignment - `assign`

Have the following format:

```
assign <signal> = <expression>;
```

- *signal* is either a single signal or a concatenation of signals
- *expression* refers to a valid Verilog expression

For the `assign` statement, the left hand side is updated whenever a signal in the right hand side changes.

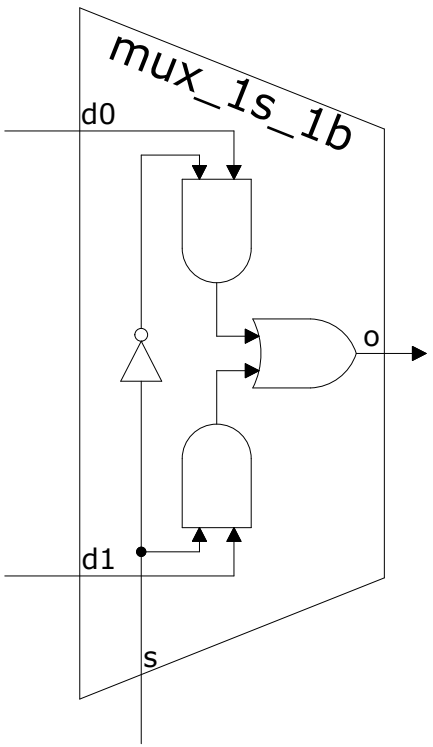
1-bit 2-to-1 multiplexer

Exercise: Implement a 1-bit 2-to-1 multiplexer using the Verilog.

Solution: The Verilog implementation and the graphical symbol of the 1-bit 2-to-1 multiplexer are depicted bellow

```
1  module mux_1s_1b (
2      input d0 ,
3      input d1 ,
4      input s ,
5      output o
6  );

8    assign o = ((~s) & d0) | (s & d1);
9  endmodule
```

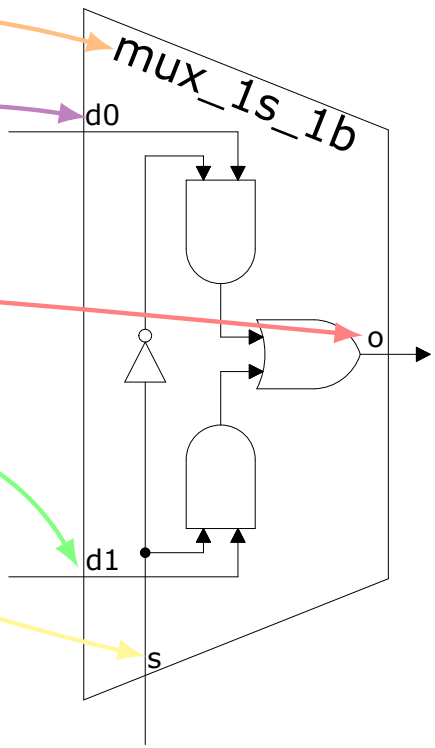


1-bit 2-to-1 multiplexer (contd.)

Components of the Verilog module definition for a 1-bit 2-to-1 multiplexer:

```
1  module mux_1s_1b (
2      input d0,
3      input d1,
4      input s,
5      output o
6  );

8      assign o = ((~s) & d0) | (s & d1);
9  endmodule
```

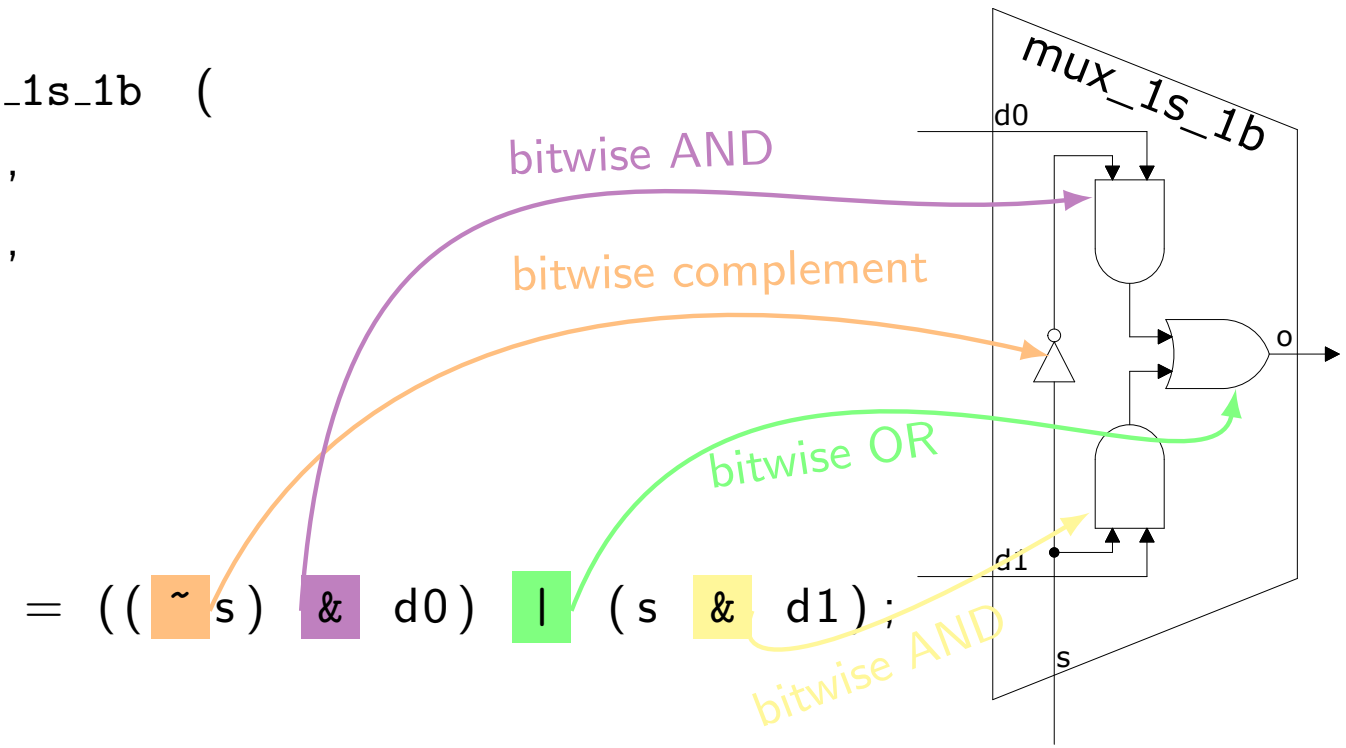


1-bit 2-to-1 multiplexer (contd.)

Module's implementation uses the Verilog continuous assignment statement, `assign`. The right hand side uses Verilog's bitwise boolean operators $\sim$, `&` and `|`.

```
1  module mux_1s_1b (
2      input d0 ,
3      input d1 ,
4      input s ,
5      output o
6  );

8    assign o = ((~s) & d0) | (s & d1);
9  endmodule
```



Verilog busses

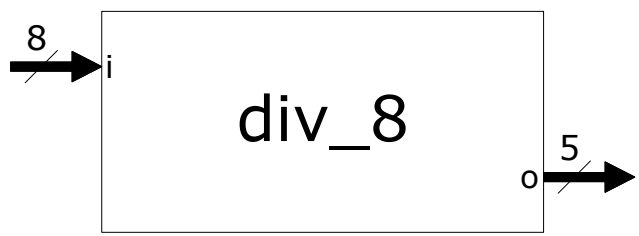
A Verilog *bus*, or a *vector* is a signal consisting of a collection of wires. It is defined by specifying the highest and the lowest bit ranks, in square brackets, followed by the bus name.

Exercise: Design a device for computing the quotient for division by 8 of an integer, unsigned, 8-bit number.

Solution:

```
1  module div_8 (
2    input [7:0] i,
3    output [4:0] o
4  );

6    assign o = i[7:3];
7  endmodule
```



Notă: The quotient for division by $8 = 2^3$ of an unsigned integer is obtained by eliminating the least significant 3 bits.

Verilog *part-select* operator

The Verilog *part-select* operator permits selecting a contiguous set of lines from a bus. The part-select mechanism specifies the upper and the lower bus ranks, between square brackets, delivering all bus' lines in between them.

Exercise: Design o module with a selection line *s* and a 64-bit input *d*. If *s* is active, the output is set to the most significant 32 bits of input *d*, otherwise the output is set to the bits between ranks 47 and 16, inclusive, of the input *d*.

Solution:

```
1  module bus_select (
2    input [63:0] d,
3    input s,
4    output [31:0] o
5  );

7    assign o = s ? d[63:32] : d[47:16];
8  endmodule
```

Verilog *concatenate* operator

The Verilog *concatenate* operator permits constructing busses. Concatenation is constructed as a list of signals, separated by commas, enclosed between curly brackets. The leftmost signal occupy the most significant bit positions of the new bus and the rightmost signal occupy the least significant ranks.

Exercise: Design a module for reversing the bit order of a 4-bit value received at input.

Solution:

```
1  module reverse_4b (
2    input [3:0] i,
3    output [3:0] o
4  );

6    assign o = {i[0], i[1], i[2], i[3]};
7  endmodule
```

Verilog arithmetic operators

Verilog provides the following binary arithmetic operators: +, −, *, /, %, for the modulus operator and ** for exponentiation. The unary arithmetic operators, + and −, are used for controlling the sign of their operands.

Exercise: Construct an 8-bit binary adder with no carry input.

Solution:

```
1  module add_8b (
2    input [7:0] x,
3    input [7:0] y,
4    output [7:0] z,
5    output co
6  );

8    assign {co, z} = x + y;
9  endmodule
```

A modulo-2^8 adder has the same implementation, except eliminating the *co* output.

Conditional operator

Verilog's *conditional operator* has the following format:

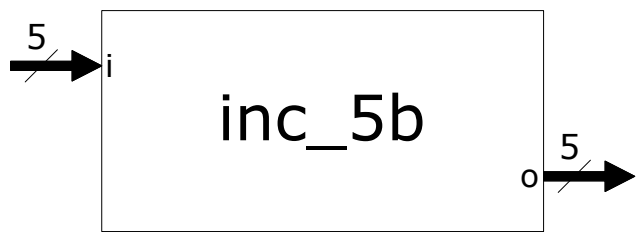
```
expression ? expression_true : expression_false
```

Exercise: Design a device for incrementing a 5-bit, unsigned integer, number at the input. If the number at the input is 31, deliver it unchanged to the output.

Solution:

```
1  module increment_5b (
2    input [4:0] i,
3    output [4:0] o
4  );

6    assign o = (i == 31) ? i : i + 1;
7  endmodule
```



Notă: The conditional operator implements the conditional instruction (*if* <condition> *then* ...), just like the multiplexer.

Verilog *replication* operator

The Verilog *replication* operator replicates an expression a number of times, concatenating all copies into a bus. Operator's format is {r{e}}, replicating expression *e* a number of *r* times.

Exercise: Construct a module for sign extension of a signed, 8-bit binary integer to 32-bits.

Solution:

```
1  module sign_extend (
2    input [7:0] i,
3    output [31:0] o
4  );

6    assign o = {{24{i[7]}}, i};
7  endmodule
```

In computer systems, a number's sign is represented in the most significant position (the leftmost bit). Sign extension, in Two's Complement (C2), copies sign bit the required number of times.

Verilog bitwise and reduction operators

Verilog's *bitwise operators* have vectors operands. The following table presents their symbols, functions and arity.

Symbol	Function	Arity
$\sim$	bitwise complementation	unary
&	bitwise AND	binary
|	bitwise OR	binary
^	bitwise EXOR	binary
^$\sim$	bitwise XNOR	binary

The *reduction* operators have a single vector operand. They generate a single-bit output obtained by applying the respective operator over all bits of the vector. The reduction operators are &, |, $\sim$& for NAND reduction, $\sim$| for NOR reduction, $\widehat{}$ and $\sim\widehat{}$ or $\widehat{}\sim$ for XNOR reduction.

Verilog bitwise and reduction operators (contd.)

Exercise: Construct a module implementing function $o(x) = \max(0, x)$ for signed numbers on 8 bits.

Solution:

```
1  module max (
2    input [7:0] x,
3    output [7:0] o
4  );
5    assign o = {8{~x[7]}} & x;
6  endmodule
```

Exercise: Construct a module for calculating the even parity bit of an 7-bit signal (even parity bit is the modulo-2 sum of all bits).

Solution:

```
1  module parity (
2    input [6:0] i,
3    output p
4  );
5    assign o = ^i;
6  endmodule
```

Other Verilog operators

Verilog relational operators: <, >, <=, >=, == (for equality), != (for inequality), === (for case equality) and !== (for case inequality). Case equality and inequality operators handle Verilog four-valued signals. In Verilog, besides having a values of 0 or 1, a signal can be undefined, marked with symbol x or in high impedance, marked with symbol z. For signals $A = 1x01$ and $B = 1x01$, $A === B$ evaluates to true, while $A == B$ evaluates to false.

Verilog logic operators: &&, || and ! (for logic negation).

Verilog shift operators are: << (for left shifting), <<< (for arithmetic left shifting of signed values), >> (for right shifting) and >>> (for arithmetic right shifting of signed values). Relational and logic operators return a one-bit value of either 1, for a true result, or 0, for a false result.

Verilog constants

Verilog constants' format:

`<bit_width>'<radix_specifier><value>`

where

- `bit_width`: decimal, positive integer representing the number of bits allocated to the constant; optional
- `radix_specifier`: can be `b` for binary, `o` for octal, `d` for decimal and `h` for hexadecimal; optional, with decimal being default
- `value` the constant's value expressed in the specified radix

Some Verilog constant examples are presented in the table bellow:

Verilog Constant	Stored as
3'b110	110
8'b0010_1101	00101101
5'd6	00110
10'h9e	0010011110

References

[Stro05] L. Strozek. Verilog Tutorial - Edited for CS141. [Online]. Available: https://wiki.eecs.yorku.ca/course_archive/2013-14/F/3201/_media/verilog-tutorial_harvard.pdf (Last accessed 20/07/2016).