

Vectorr ○

[← vector3](#) (✓) (/wiki/vector3)[vector4 ○](#) (/wiki/vector4) →

Given an 8-bit input vector [7:0], reverse its bit ordering.

See also: Reversing a longer vector (/wiki/Vector100r).

Module Declaration

```
module top_module(  
    input [7:0] in,  
    output [7:0] out  
);
```

Hint...

- `assign out[7:0] = in[0:7];` does not work because Verilog does not allow vector bit ordering to be flipped.
- The concatenation operator may save a bit of coding, allowing for 1 assign statement instead of 8.

Write your solution here

[\[Load a previous submission\]](#) ▼[Load](#)

```
1 module top_module(  
2     input [7:0] in,  
3     output [7:0] out  
4 );  
5     assign out = {in[0],in[1],in[2],in[3],in[4],in[5],in[6],in[7]};  
6     //assign out[7:0] = in[0:7];  
7  
8 endmodule  
9
```

[Submit](#)[Submit \(new window\)](#)[Upload a source file...](#) ▼

Solution

[Show solution](#)

```
1 module top_module (  
2     input [7:0] in,  
3     output [7:0] out
```

```

4 );
5
6 assign {out[0],out[1],out[2],out[3],out[4],out[5],out[6],out[7],out[8]}
7
8 /*
9 // I know you're dying to know how to use a loop to do this:
10
11 // Create a combinational always block. This creates combinational
12 // as sequential code. for-loops describe circuit *behaviour*
13 // inside procedural blocks (e.g., always block).
14 // The circuit created (wires and gates) does NOT do any iteration
15 // AS IF the iteration occurred. In reality, a logic synthesizer
16 // figure out what circuit to produce. (In contrast, a Verilog simulator
17 // during simulation.)
18 always @(*) begin
19     for (int i=0; i<8; i++) // int is a SystemVerilog type.
20         out[i] = in[8-i-1];
21 end
22
23
24 // It is also possible to do this with a generate-for loop. (These loops
25 // but are quite different in concept, and not easy to understand.
26 // of "things" (Unlike procedural loops, it doesn't describe circuit
27 // module instantiations, net/variable declarations, and procedural
28 // a procedure). Generate loops (and genvars) are evaluated before
29 // blocks as a form of preprocessing to generate more code, verilog
30 // In the example below, the generate-for loop first creates the circuit
31 // synthesized.
32 // Note that because of its intended usage (generating code as a
33 // on how you use them. Examples: 1. Quartus requires a generate block
34 // attached (in this example, named "my_block_name"). 2. Inside a
35 generate
36     genvar i;
37     for (i=0; i<8; i = i+1) begin: my_block_name
38         assign out[i] = in[8-i-1];
39     end
40 endgenerate
41 */
42
43 endmodule
44

```

vectorr — Compile and simulate

Running Quartus synthesis. [Show Quartus messages...](#)

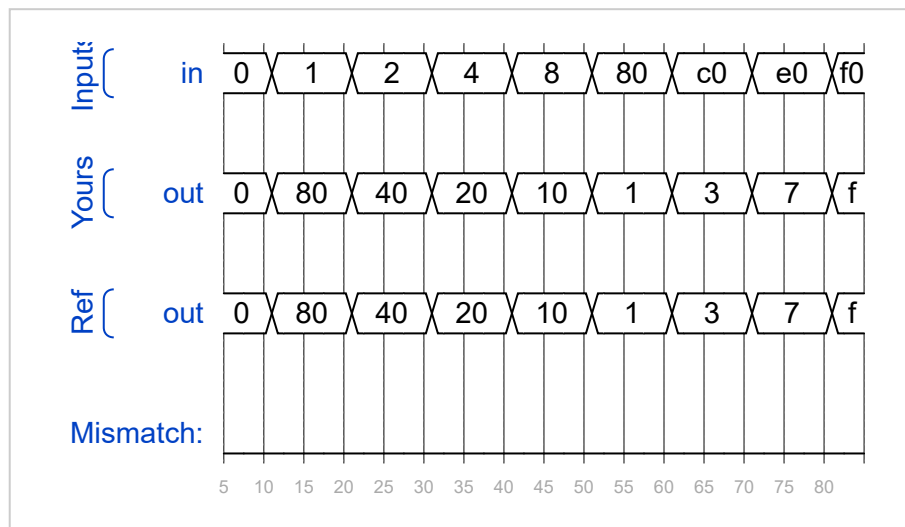
Running ModelSim simulation. [Show Modelsim messages...](#)

Status: Success!

You have solved 17 problems. [See my progress...](#)

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).



← vector3 ☒ (/wiki/vector3)

vector4 ☐ (/wiki/vector4) →

Retrieved from "http://hdlbits.01xz.net/mw/index.php?title=Vectorr&oldid=1253
(http://hdlbits.01xz.net/mw/index.php?title=Vectorr&oldid=1253)"

Problem Set Contents

▸ Getting Started

▼ Verilog Language

▸ Basics

▼ Vectors

- ☒ Vectors (/wiki/vector0)
- ☒ Vectors in more detail (/wiki/vector1)
- ☒ Vector part select (/wiki/vector2)
- ☒ Bitwise operators (/wiki/vectorgates)
- ☒ Four-input gates (/wiki/gates4)
- ☒ Vector concatenation operator (/wiki/vector3)
- ☐ **Vector reversal 1 (/wiki/vectorr)**
- ☐ Replication operator (/wiki/vector4)
- ☐ More replication (/wiki/vector5)

▸ Modules: Hierarchy

▸ Procedures

▸ More Verilog Features

▸ Circuits

▸ Verification: Reading Simulations

▸ Verification: Writing Testbenches

