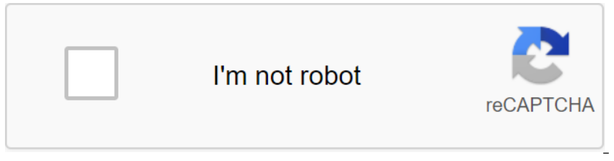

Continue

Bcd adder verilog

Useful links for E15 labs You can print the Synthesized Verilog Fast Link (pdf) (originally at Arch/files/verilogcheatsheet.pdf) In this lab you will perform many of the same tasks that you did in the last lab, but using a more advanced Verilog, for a much more compact implementation. Verilog will do a lot more nitty-gritty design work, allowing you to focus on higher-level problems. You can turn to Verilog Tutorial as you work through this lab, especially to create a new project and compile, modeling and dowloading your projects. Challenge 0: Cylon Redux Use If ... Then... Still build, we can make 4 LED cylinder chains, as shown below. It performs the same function as the module from the last lab, except for entering a 3-bit vector instead of 3 discrete inputs. Cylon module (d, x); 'cylon'd' chain (2:0)d; 3-bit input (in vector) output reg (3:0)x; 4-bit output (in vector) always@ (i) starts if (d 3'd0) if d is 0... Start x 4'b0001; // ... turn on x'0, x-1'x'x'x'0 end even if (d 3'd1) (d q 3'd5)) otherwise if d is 1 or 5... Start x No 4'b0010; // ... turn on x.1. end still if (d y 3'd2) (d q 3'd4)) otherwise, if d is 2 or 4... Start x No 4'b0100; x'2 is on the end yet if (d 3'd3) / otherwise if d is 3... Start x 4'b1000; // ... Turn x 3 at the end yet! yet... Start x No 4'bxxxx; // ... we don't care about going out. end end endmodule We can make it shorter by removing the start and end from if ... more... because each statement is only one line. Cylon module (d, x); 'cylon'd' chain (2:0)d; 3-bit input (in vector) output reg (3:0)x; 4-bit output (in vector) always@ (i) starts if (d 3'd0) if d is 0... x 4'b0001; // ... turn on x'0, x-1'x (d q 3'd5)) otherwise if d is 1 or 5... x No. 4'b0010; // ... turn on x.1. more if (d 3'd2) (d q 3'd4)) otherwise, if d is 2 or 4... x 4'b0100; x'2 is in a different place if (d q 3'd3) / otherwise, if d is 3... x 4'b1000; // ... Turn x 3 on another... x 4'bxxxx; // ... we don't care about going out. end endmodule or even easier using the deal to build instead. Cylon module (d, x); 'cylon'd' chain (2:0)d; 3-bit power output reg (3:0)x; 4-bit output (in vector) always@ (i) begins (d) / output change depending on d 3'd0: x'4'b0001; d'0, x'0'1, others from 3'd1, 3'd5: x'4'b0010; d'1 or d'5, x-1 3'd2, 3'd4: x'4'b0100; d'2 or d'4, x-2'1 3'd3: x'4'b1000; d'3, default x'3'1: x'4'bxxxx; Don't care about the endmodule endmodule Challenge 1: Working with 7 segment displays, redux. Write a module called BCD2Seven, which as bcD to 7 segment decoder. It takes as input four bits of binary number, and as output lights seven seven LED (example below shows HEX0). Remember that LED segments are active low (they are turned on when you send a low signal to them). Number (decimal) 0 1 2 3 4 5 6 7 8 9 Number (binary) 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 Display Use body statement or other behavioral designs. Suppose 4-bit inputs are a number from 0 to 9, and outputs should be not equal to input above 9. The block diagram for your module is shown below. The skeleton of the code is given. Keep the names of the entrance and output variables. BCD2Seven (D,L) input (3:0)D; Four-bit BCD 6-0'L output; Seven bits for LEDs/... Your code ... Endmodule Make sure the module works either by modeling or when connected to the hardware. Task 2: Displaying two six-digit numbers Now add a module defined in E15Counter1HzB.v that you used in the previous lab - you can refer to this lab if you want to update your memory of how it works. Use one bcd2seven module for the HEX3 drive (left six-digit) and the other for the HEX2 drive. In the picture I named the left decoder dig3 and righmost decoder dig2. The entries on dig3 are the exit, as shown below, and make sure it works as expected (meaning you should know what to expect). The first few lines of the module (in a file called doubleDigits.v) appear on the right. Run the code on the hardware to make sure it's working. doubleDigits module (CLOCK_50, SW, HEX3, HEX2); Entrance CLOCK_50; Entering the clock 50 MHz (3:0) SW; Exit 4 switches (6-0) HEX3; Output 7 Segment (6:0) HEX2; Exit 7 segments /... Your code goes here endmodule. Challenge 3: BCD Adding Write BCD adder (bcdAdder.v) using behavioral code (the algorithm is described in your text, or you can understand it for yourself). There are two 4 bit inputs and two 4-bit outputs. You will probably need it if the application. Remember, too, that this verilog code is not consistent. You can use Verilog to easily add two 4-digit BCD (binary encoded decimal) numbers to get a 5-bit binary result. A fragment of the code demonstrates. module bcdAdder (bcdIn1, bcdIn2, bcdOut1, bcdOut2); Input (3:0) bcdIn1; Enter 4 bit input (3:0) bcdIn2; Weekend reg (3-0) bcdOut1; 4 bit reg exits (3:0) bcdOut2; reg (sum 4-0); 5-bit intermediate value. Always : (I) start the amount and bcdIn1 and bcdIn2; make an addition/..... You can then use a 5-bit result in total to create two 4-bit BCD numbers with simple if... Then... more to build. I suggest you model the module to make sure it works, but it's not required for the report. Challenge 4: Putting it all together to build a module based on one in 2. 4 switches and counter outlets should go to your bcdAdder module, and bcdAdder module outputs should drive the right hex numbers. After all, HEX3 should count from 0 to 9, HEX2 must display the BCD number entered by SW-3:0, SW-3:0, should show the most significant result figure, and HEX0 should show the least significant figure. The result should be from 00 (0 added to 0) to 18 (9 added to 9). Think about the block scheme you need to connect different modules. You don't have to take into account erroneous inputs (i.e. 4 bit numbers more than 9) you should be able to use the modules from above without changing them. Show your scheme to me (or send me a video, along with the names of the people in your group). Include E15 Lab 2, Task 4 in the subject line. Challenge 5: Additional (not required) design change from task 4 to diagram that subtract the number on HEX2 from the number on HEX3 and displays the result on HEX1 and HEX0 (HEX1 is used for a negative mark if necessary). It doesn't take up much code, but takes a little more thought than just adding. Demonstrate your scheme to me with both positive and negative subtraction results (or send me a video, along with the names of the people in your group). Include E15 Lab 2, Task 5 in the theme line. Turn on: Each part of each task should be well marked and in order. Turn on one file on the moodle. Challenge 1 Include your well commented code for bcd2seven.v. (15 points) Explain how you checked that it works. (5 points) Challenge 2 Include a well-commented code for doubleDigits.v (20 points) Challenge 3: Turn on the well commented code for bcdAdder.v (20 points) Challenge 4: Finish the block chart below (or do a similar one) and include it in your report. I called the module iCanAdd - you should use the name of the module that you wrote. (15 points) Include the well commented code for the module (s) you wrote. (15 points) Demonstrate your scheme to me (or send me a video, along with the names of the people in your group). (10 points) Challenge 5: (Additionally, not required) Make a block chart and include it in your report. (5 points) Include the well commented code for the module (s) you wrote. (5 points) Demonstrate your scheme to me (or send me a video, along with the names of the people in your group). (No points, but you won't get credit if I don't see it working) - Back to the E15 Lab page, please contact me if you find any errors or other problems (such as something vaguely stated) on this 8-Bit Adder webpage used to add two 8-bit words, or two bytes, binary data. BCD is a standard form for insing binary information, where 4 bits, or nibble, represents one number in decimal, up to the number 9. For example: the decimal number 19 is represented by two bites. The first nibble, or most significant nibble, is a 4-bit representation of 1 second nibble, or least significant nibble, is a 4-bit view of 9. Combined, these two nibbles will 1 and 9, or decimal number 19. 19 0001 1001 (bcD presentation) Language used by Verilog: C as HDL is easier to understand and and Design time, since the syntax is more concise that VHDL Tools used Cadence NCSim: To compile and validate the logic of the design of team members Jonathan Frey - Group Leader William Nitsch - Testbench Coder Raashid Ansari - Source Coder NOTE: Refer detailed-report.md for a better explanation of the project. Page 2 of 8-Bit Adder is used to add two 8-bit words, or two bytes, binary data. BCD is a standard form for insing binary information, where 4 bits, or nibble, represents one number in decimal, up to the number 9. For example: the decimal number 19 is represented by two bites. The first nibble, or most significant nibble, is a 4-bit representation of 1 second nibble, or least significant nibble, is a 4-bit view of 9. Combined, these two nibbles will represent 1 and 9, or decimal number 19. 19 0001 1001 (BCD presentation) Language used by Verilog: C as HDL is easier to understand and saves design time because the syntax is more concise that VHDL Tools used Cadence NCSim: To compile and validate the logic of the design team members Jonathan Frey - Group Leader William Nitsch - Testbench Coder Raashid Ansari - Coder Source: Detailed-report.md for better explanation of the project. Project. bcd full adder verilog. bcd adder verilog code. bcd adder verilog code and testbench. bcd adder verilog code gate level. bcd adder verilog code structural. bcd adder verilog code dataflow. 8 bit bcd adder verilog code. 2 digit bcd adder verilog code