Matthew Hardenburgh

Design

Part 1 Procedural Design

For this implementation I used an if statement inside an always block to check if the year is a leap year. I first converted the BCD to binary. If the year is NOT a century and it is divisible by 4, or, if the year IS a century and divisible by 400, then it is a leap year. We then set LY to 1 If it is a leap year.

For the test bench I created a loop that iterated through every year starting from 1582, up to 4818, the years specified in the homework 1. At each iteration, I first isolate each digit of the year and set it to YM, YH, YT, YO respectively. Then, I calculate if the year is a leap year and set a reg variable called expected to 1'b1 if yes, else 1'b0. If expected value is equal to LY, the output of the leap year module, then the test passes.

Part 2 Divbyfour and Iszero

DivisibleByFour

I used a brute force method for this approach, with some tricks. There are only two decimal digits so the highest number that can be input to the module is 99. Given a BCD input for YT and YO, I check if the binary representation is equal to pre-determined value. That pre-determined value, being a value that is divisible by four. For example, 4*4 = 16 is divisible by 4, so I check if the BCD for YT = 1 and TO = 6. Again, to show another example, 4*14 = 56 is divisible by 4, so I check if the BCD for YT = 5 and TO = 6. I'm sure there is better solution, but I was running out of time, so I went with this approach.

For the testbench I followed a similar approach to the testbench for LeapYear. I loop from 0 up to 99, convert each number to BCD and feed it into the DUT, check if the number is divisible by 4, and then the iteration passes if the expected result is the same as the output from the module.

IsZero

This one seemed straightforward, check if each input wire is equal to 0. I not'ed each of the inputs and collected the inputs at a final AND gate. The gate would be true if the inputs were zero. Y = (~BCD[0]) & (~BCD[1]) & (~BCD[2]) & (~BCD[3]).

For the testbench I followed a similar approach to the testbench for LeapYear. I loop from 0 up to 9, convert each number to BCD, and feed it into the DUT, check if each number is

equal to 0, and then the iteration passes if the expected result is the same as the output from the module.

Part 3 Data Flow Design

I mention my approach to the submodules in part 2, but for dataflow I use the inverter operator , ~, instead of a not gate. For the leap year module, I had to break down the algorithm for checking if a set of BCD inputs is a leap year into two parts. First, I check if the lower two digits are divisible by four and that both are not zero (that it is NOT a century), Y = (lower two digits are divisible by 4) & (~(YT is zero & YO is zero)). Then I check if the upper two digits are divisible by four and that lower digits are both zero (that it IS a century), X = (upper two digits are divisible by 4) & (YT is zero) & (YO is zero)

Part 4, Structural Design

Converted what I had done in part 3 to gate level design. Nothing changed algorithmically.