ECE 571
Introduction to SystemVerilog
Spring 2015
Homework 3

1) Modify the testbench you created in Homework 2 to use SystemVerilog structs and queues to hold operands until the result is available from the staggered adder. Modify the testbench to wait a random number of cycles (between 1 and 10) between applying inputs from successive testcases to the module.

There is no need to change your staggered adder module.

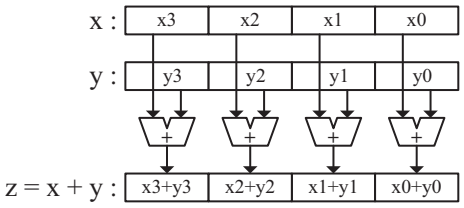
Correctness, clarity, reusability, and maintainability count.

2) Consult the Wikipedia article for "IPv6 Packet" and create a package that creates typedefs as necessary for a IPv6 fixed format packet header (implemented with a struct). Your package should include two functions as well as the typedefs. One function should accept as an argument an IPv6 packet header and display the contents of each field in a "reasonable" format. The other function should return a bit vector of length equal to the packet size in bits. Create a module that imports the package and demonstrates the use of the two functions.

3) Modern high performance microprocessors often feature SIMD (single instruction, multiple data) instructions capable of performing the same operation in parallel on multiple operands to speed up multimedia applications where operands might be 8-bit, 16-bit or 32-bit pixels, vertices, or sound samples. Thus, a processor with a single 64-bit ALU could permit a single instruction to be used to add two pairs of 32-bit operands, 4 pairs of 16-bit operands, or 8 pairs of 8-bit operands using one instruction and the same ALU.

A "partitioned" ALU is used to implement these SIMD instructions. A partitioned ALU inhibits the carry chain at appropriate places when performing SIMD operations so that the carry out of the MSB of the result of an operation doesn't carry into the adjacent operation. For example, when doing an add operation on 4 pairs of 16-bit integers, the carry chain is broken between bits 15 and 16, 31 and 32, 47 and 48.

The diagram below depicts a 64-bit ALU partitioned to perform four parallel 16-bit additions. The same ALU could be partitioned to perform eight parallel 8-bit operations by breaking the carry chain further.



The Intel x86 architecture has been extended several times to support additional SIMD capabilities (MMX, SSE-2/3/4 and AVX provided 64-, 128- and 256-bit registers respectively). The Intel x86 MMX introduced padd, an integer add instruction with several variants (paddb, paddw, paddd, paddq) that can operate on byte, word, double word, and quadword operands respectively.

In addition, the instruction can be modified to support signed or unsigned saturating addition (for example, paddsb for signed saturating addition with byte operands, paddusb for unsigned saturating addition with byte operands), though not for double word or quadword operands. Recall that saturating adds rather than overflowing (or underflowing), "saturate" by producing the largest (or smallest) representable value.

The condition codes (N, Z, V, C) are not set when performing SIMD instructions on data types smaller than quadwords.

Create a portion of a testbench for verifying a 64-bit combinational integer added capable of supporting byte, word, double work, and quadword operands. You can assume it supports only non-saturating arithmetic.

In particular, you should create and test a function that can generate two 64-bit operands to supply to the adder module ports from each of the supported operand types. For example, packing four 16-bit operands into each variable connected to the 64-bit ports of the adder module. Similarly, create and test a function that takes the 64-bit result and puts it into result values of the appropriate size so they can be checked against the expected results.