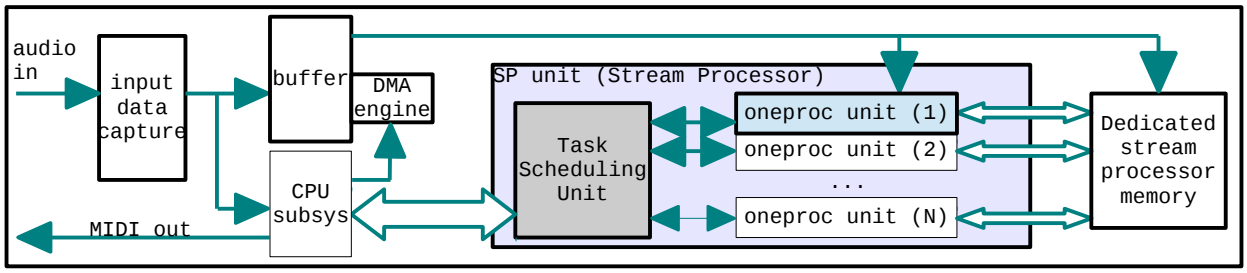
# Stream Processor Testbench

http://www.yanthia.com/online/projlets/StreamProc/index.html

This page describes a testbench for the oneproc unit in the block diagram below. The diagram depicts a system currently under design. At least two other testbenches will follow: one for the SP unit, which will reuse much of the oneproc testbench, and a second testbench for the Task Scheduling Unit.




**The transactions**: The CPU will feed the SP unit continuously, make high-level decisions on what to do next based on results from the SP unit, issue new commands, and continue on in this way until a decision threshold is reached. Each oneproc unit can take several different commands with a variety of data formats. Each oneproc has independent read and write channels, and reads can complete out-of-order w.r.t the writes. Command and data values only make sense in a limited range. All this means that the scope for *transaction* randomization is quite limited.

But the transaction class must support randomization of the communication protocol and a set of out-of-range data values such that the following design principle can be tested: The CPU might write garbage in, but the oneproc unit must always provide something out, even garbage: The oneproc unit must never hang.

Side note:

As for what this system is intended to do, please see the Voice to MIDI project. But in brief, it will make guesses about the nature of the incoming signals using both graceful heuristics and barbaric brute-force computing.

"Brute-force" means that the value of N, the number of oneproc units available, will only be limited to how many I can fit into an FPGA.

**The sequences**: Besides the scoreboard, the sequences are the most awkward part of testing the oneproc unit because the testbench must behave in a manner similar to the task scheduler, and that brings the following complications:

1) It severely limits the freedom to randomize transaction data since the command and data fields have to make sense together. This is mentioned in "The transactions" above.
2) Some write actions write a 64-byte block, some only a single 32-bit word.
3) The read channel activity requires a pair of transactions per action, as does the write channel.
4) Each computation started via the write channel needs some sort of response from the read channel.
5) The read channel doesn't know what *else* to read until after it has read the status word.

To provide the low-level sequence control and coordination along with high-level "what is the test about" requirements, a three-level sequence mechanism was used as is shown below on the left.

4 top-level virtual sequences

5 mid-level construction sequences

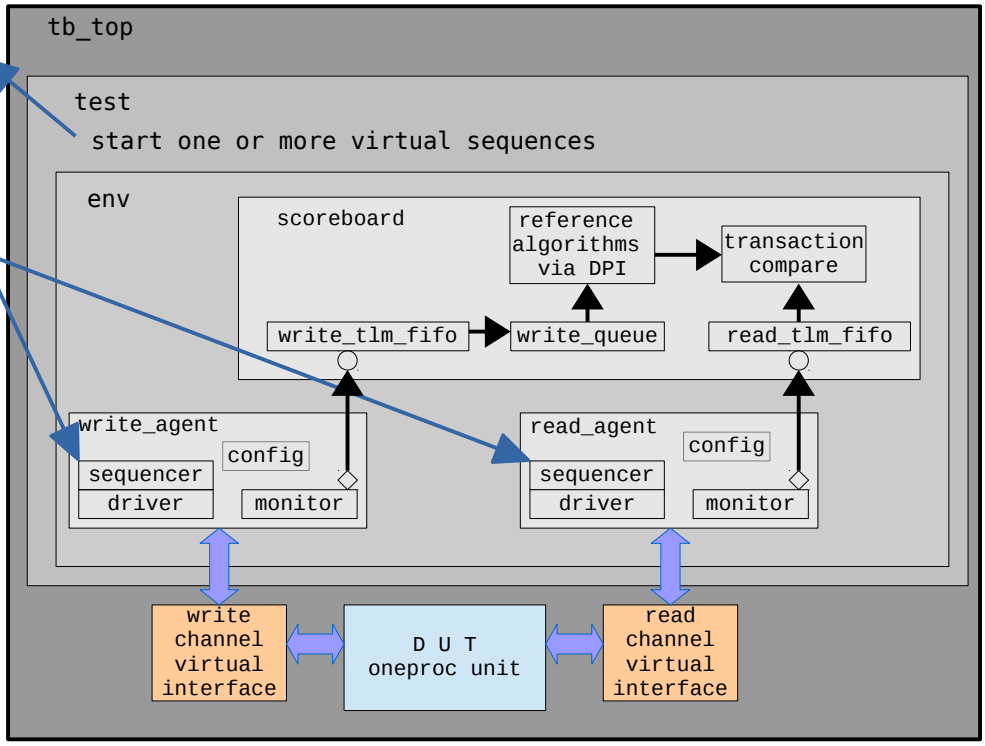
10 botom-level atomic sequences

The current oneproc functions are somewhat placeholders, although they do represent the sort of computations the final system will perform. The functions are:

**CORR**: Scan all *l*, L-length sub-strings in an array of length S, and return the maximum Pearson's r correlation coefficient found, and where in S *l* is located.

**SMOOTH**: foreach element in the array E, $E_n = (E_n + E_{n+1})/2$

**NOP**: No nothing, but do it with a programmable delay (for speeding up randomized testing of the protocol).




Dan Downs OstensiblyMyID@gmail.com Scottsdale, AZ, November 2016