Summary: it works.

Software Implementation: The submitted software implementation *mayo_verify.m* recreates the MAYO1 algorithm in MATLAB. Recreating their provided C implementation in MATLAB was easier for me to understand the specific mathematic operations being performed, and therefore create input/output test vectors for specific operations. The C implementation obscures the details of the Galois operations. These being novel to me, it was helpful to recreate them as MATLAB function in the *+galois/* package. The same applies to mapping the MAYO specific operations (bitslice decoding, etc..) to a straight forward implementation which could then be used as my guide for specific HDL entities.

Included in the reference implementation folder is *mayo_example.mat* it contains variable values from the C implementation at various points in time. They serve as the inputs to *mayo_verify* and are used to check correctness a different point during execution.

HDL test vectors are generated by *save_test_vectors.m* this script assumes the MATLAB workspace is in the same state generated by running *mayo_verify*. The output *.ascii* files are stored in c_data to be read in by the various HDL testbenches.

Verification Strategy: A testbench for every entity was created, and submitted. With the exception of the controller modules, however, entities that could be split into a data path and controller a test bench was created for the data path and top-level combination of the data path and controller. Verification was confirmed generally using test vectors generated by *save_test_vectors.m*, in some instances the scope of the entity was so small that verification could be satisfied using a handful of values hardcoded into the testbench. All submitted testbenches run as is and produce the expected output.

Top-level Verification: The top-level entity of the design is *mayo_verify.vhd,* the corresponding testbench is *TB_mayo_verify.vhd.* It reads the two inputs EPK and SIG from their respective test vector files and feeds it to the design following the interface defined in assumption 1. Figure 1 shows an example of this interface for the first 64 bytes being read from *data_in.*

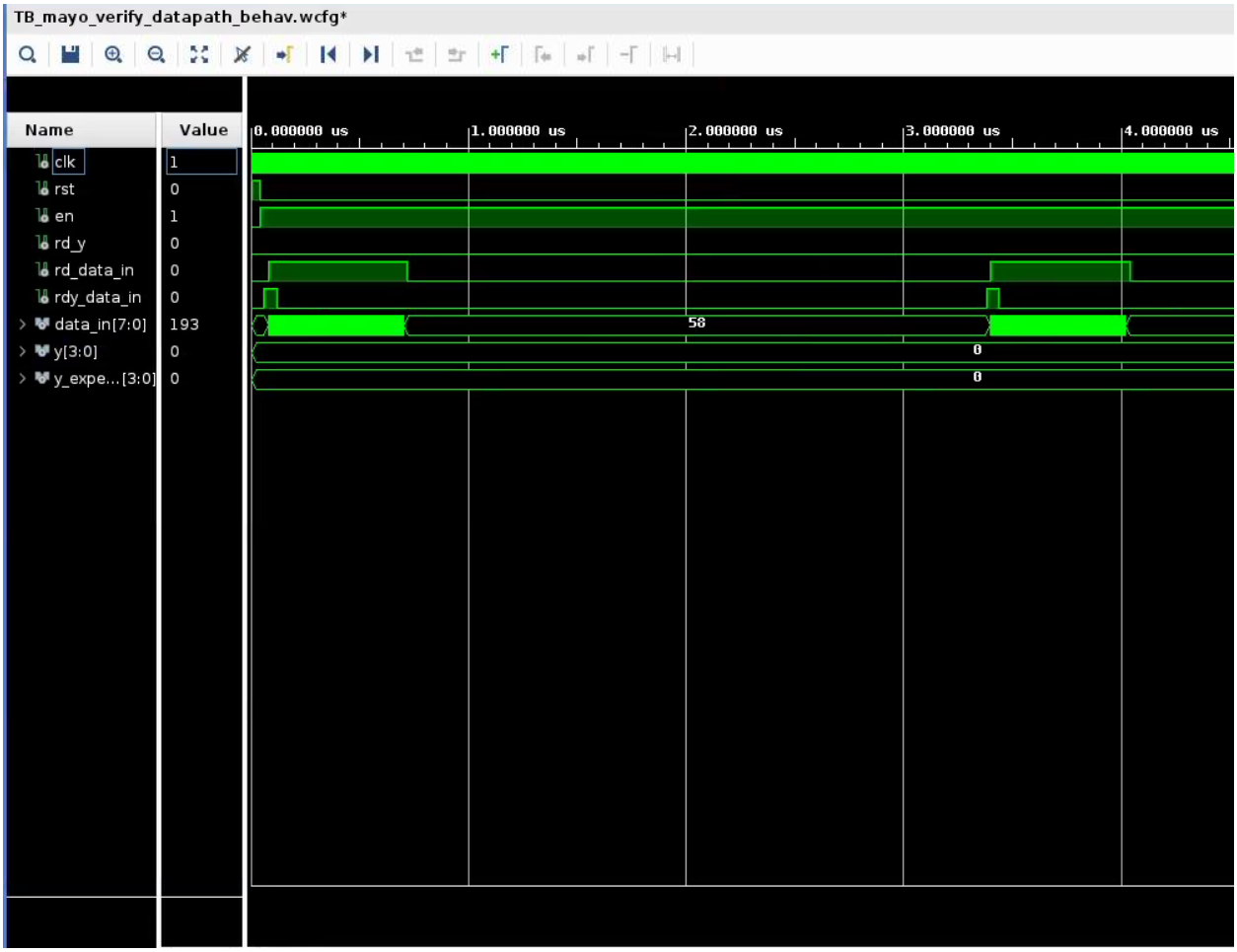

Figure 1. Data Input Interface

After a time (~8ms of simulated time since the input is so large) the design's output *rd_y* (read y) goes high for 64 clock cycles indicating the data on output bus *y* is valid. While *rd_y* is asserted *y_expected* is read from the test vector file *y_golden.ascii* and compared to y. As submitted and shown in Figure 1, *y* and *y_expected* match for the duration of *rd_y.* This is the desired behavior of the circuit.

Because of assumption 2 the circuit has an extremely large area (as discussed in results). Running *TB_mayo_verify.vhd* takes approximately 50 minutes on my machine,

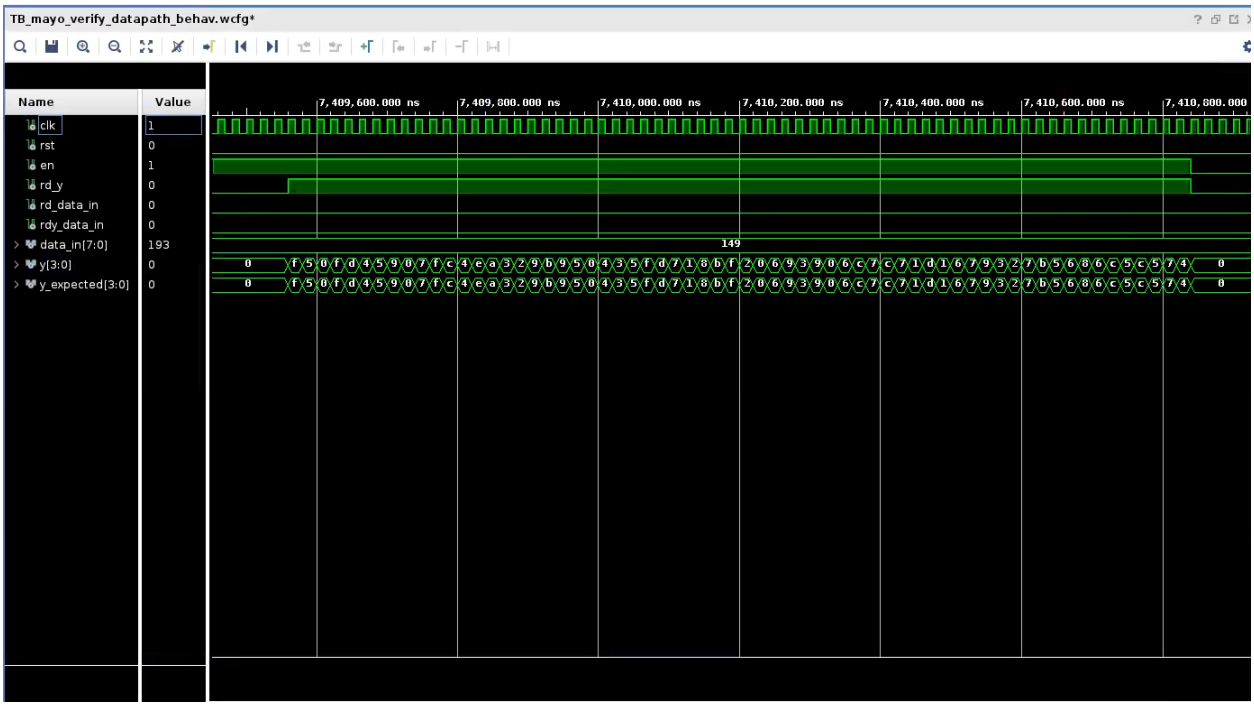

Figure 2. Comparison of *y* and *y_expected.*