Assumption 1: The pseudo code for *Mayo Verify* specifies the inputs as the expanded public key (EPK) and signature (SIG). (This is excluding the message input (M) to *Mayo Verify* because it was out of the scope of my project guidelines). In the reference C implementation EPK and SIG are already allocated and populated. The function inputs are pointers to the respective arrays. To make all bytes of EPK and SIG present in my design from on onset both EPK and SIG would have to be provided in parallel. That isn't feasible given that combined the EPK and SIG total 71 Kbytes. Therefore, my design assumes that EPK and SIG are read in byte by byte from a data input interface with a width of 8-bits.

Implication of Assumption 1: To store decode and store EPK in a RAM in such a format as to satisfy assumption 2 for every 32 bytes read from the input at minimum 128 clock cycles are needed to write it to the internal RAM *epk_storage_RAM*. Therefore, the EPK cannot be input at one byte per clock cycle, since the design would need an impractically large buffer to prevent overruns. To alleviate this problem the design uses three ports to facilitate the data input interface. 1) *data_in,* the 8-bit data bus, 2) *rd_data_in* (read data in) a 1-bit input port acting as a flag to indicate the circuit should be reading the data on *data_in,* and 3) *rdy_data_in* (ready for data in) a 1-bit output port from the design indicating that the circuit is ready to receive the next 32-bytes from data_*in*. If *rd_data_in* is set high before *rdy_data_in* all data preceding *rdy_data_in* will be ignored.

Additionally, using this 1 byte at a time input port means that the overwhelming majority of execution time for this circuit is reading and storing the inputs. This could be reduced by increasing the width of *data_in,* it could be changed to 32 bytes without any major modifications needed to the code. But for the same of a simple input I kept it at 1 byte.

Assumption 2: My design aimed to reduce the number of clock cycles needed to compute the main loop of algorithm (pseudo code lines 22-26).

Implication of Assumption 2: The original design had the circuit computing *u* in one clock cycle. That design had me computing 64, (1x66) * (66x66) * (66x1) matrix multiplications in parallel. Simulating this design too an incompletable amount of time as was abandoned. The submitted version of the design instead only compute a single index of *u* per clock cycle, reducing the resources needed compared to design one by a factor of 64. To achieve the (1x66) * (66x66) * (66x1) matrix computation of the pseudo code arrays, ST, P, and S respectively, a specific slice of P needed to accessed based on a single index. In software P is an (66x66x64) 3d matrix. Where P is derived from the EPK. The first 32 bytes provide P(1,1,:), the next P(1,2,:) and so on until P is populated. To store P in a RAM where each address is a 66x66 slice of P, it takes 128 clock cycles to update the RAM for a specific row, column combination for all slices. (*See epk_slice_storage*).

The point being, in order to reduce the computation time of the main loop the time needed to read and store the input is significantly increased as a function of input length. Given that the EPK is so long, this assumption drastically hurts the overall runtime of the design. If I were to create a version 2.0 of this design I wouldn't have made assumption 2.

Assumption 3: The design is generic about MAYO versions.

Implication of Assumption 3: All constants referenced in this report and schematics are in reference to the MAYO1 implementation. The HDL design itself is completely generic about all MAYO parameters, they are defined as constants in a universally imported package, *PKG_matrix_types.vhd*. I did not have time to test other MAYO implementations, but there is no reason to think they wouldn't work.