1. DESIGN AND IMPLEMENTATION
2. *System Architecture*

We give credit to Google’s TPU as inspiration for our own architecture. Given the level of design we are at, resources describing Google’s TPU proved very helpful in implementing our own.

A basic overview of our architecture is shown below. Commands and data flow in from the host interface, an Avalon AXI bus. These commands are decoded into one of the following functions:

* + *Write Weight Memory* - Data on the bus is written into a specified location in Weight Memory space
  + *Write Input Memory* - Data on the bus is written into a specified location in Input Memory space
  + *Fill Weight FIFO’s* - A set of weights is read from weight memory into the weight FIFO’s
  + *Drain Weight FIFO’s* - The set of weights currently held in the weight FIFOs is loaded into the systolic array
  + *Matrix Multiply* - A set of inputs is piped into the systolic array and multiplied with the set of weights currently held in the array.
  + *Read Output Memory* - A specified word from memory is read to the host.

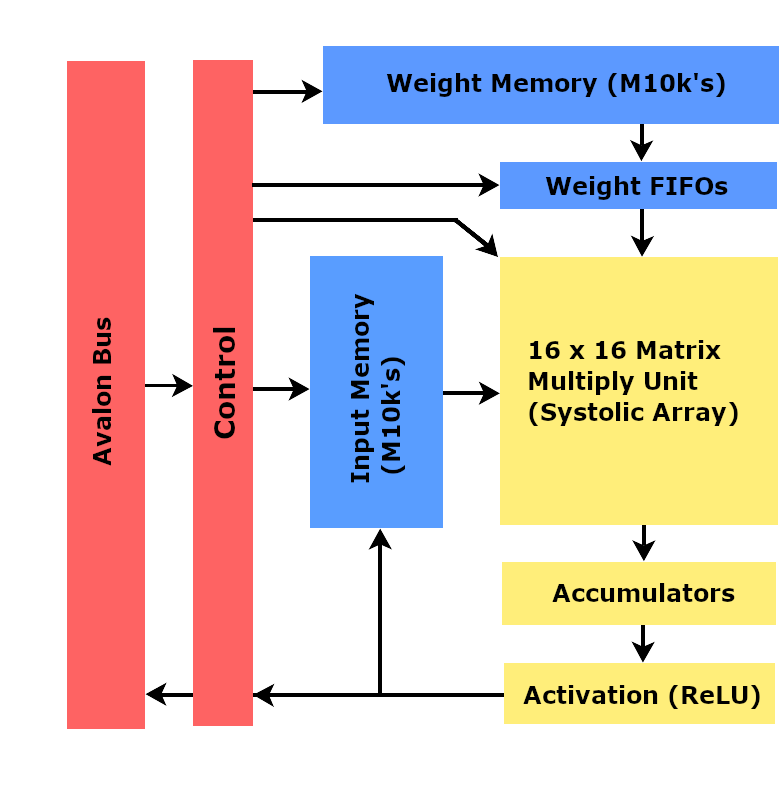


Fig. 2: High level view of system Architecture

1. *Systolic Array*

Our TPU features a weight stationary systolic array, meaning a set of weights may be loaded in once but used for many operations. The array is fully pipelined, performing a 16 x 16 matrix multiply in just 32 cycles. It is composed of many processing elements (PEs), which contain a small amount of memory and control logic, and a single multiply accumulate data path.

A complete Matrix Multiply starts at the top left corner of the Systolic array, and is piped diagonally downward. In the first cycle of a multiply, input memory supplies data for only the top left PE. After one cycle, the first PE activates its neighbors below and to the right, creating the diagonally downward piping.

Each PE holds one element of the input matrix in any given cycle, and passes that element to its right neighbor every cycle. The multiply accumulate result is passed downward to the neighbor below every cycle. Each PE then multiplies its input element with the weight element stored in it, then adds that value to the sum being supplied from its above neighbor. Note that memory interfaces only exist at the edges of the systolic array.

Multiplication results appear at the bottom of the systolic array 16 cycles after a multiplication is started, and continue flowing out for 16 cycles. The flow of data is illustrated below, showing a scaled down version of our systolic array.

1. *Control*

The control logic was where we spent the most time debugging, and is what takes up most of the ALM’s used in the final implementation. We use a single main control module which interfaces directly with the bus, decodes signals, then drives appropriate control bits in smaller control modules. We have control for reading input and weight memories, in which a counter is used to facilitate the loading of one entire matrix in 16 clock cycles.

An output memory control module reads signals from the input memory control, and will begin storing outputs from the systolic array into the output memory module. For both reading input and weight memory as well as writing output memory, the user provides a base address in which the first row of a matrix will be stored, and control iterates sequentially through the next 15 addresses until an entire matrix has been stored.

FIFO control logic facilitates the loading of weights into FIFO’s above the systolic array. It works in tandem with the weight memory control, driving FIFO signals as the weight memory control provides data. Note that all

control modules receive their inputs from the bus control module, which can be considered the master controller in the TPU.