**Wide Multiplier Block Documentation**

**Revision:** 1.0

**Date:** June 2021

The WideMult block implements the function, dout= A + B + C(D + 0.5(C\*E)). The input signal types and widths are:

* + A is 128 bit signed
  + B is 64 bit unsigned
  + C is 64 bit signed
  + D is 128 bit signed
  + E is 128 bit signed

The composition of signals A, D, and E are 64 fractional bits, 63 integer bits and 1 sign bit. Signals B and C do not include the fractional bits. In the final calculation, B is shifted left by 64 because it contains no fractional bits.

To meet the performance requirement of 156.25 MHz, an inferred parameterized 69x69 signed multiplier block was created with a pipeline register at each multiply stage for maximum performance. The input and output register of the Math blocks are also utilized implementing this function. The larger multiply functions were built using the 69x69 building block, configured for 64x64 and 64x65 bit widths, calculating the partial sums where required. The implementation has a latency of 37 clocks.

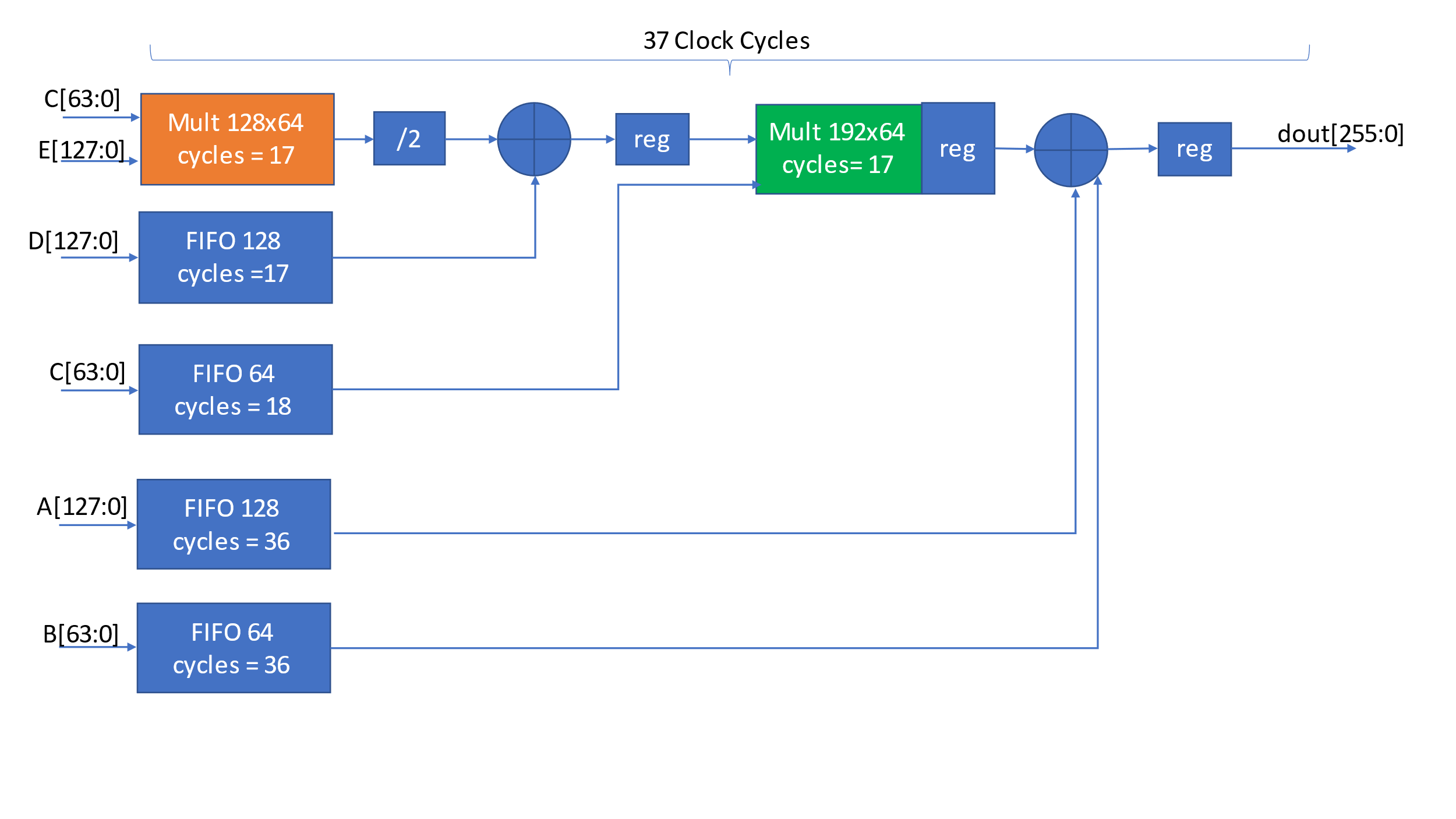
 Figure 1 - High level block diagram

Figure 1 shows the high level architecture of the function. The FIFO blocks comprised of uSRAM blocks were used as delay elements to save logic resources and the registers were used to improve performance. Two representations of the 128x64 bit multiply block are shown below, figure 2 shows the bit representation and figure 3 shows the functional block representation. Two representations of the 192x64 bit multiply block are shown below, figure 4 shows the bit representation and figure 5 shows the functional block representation. The multiply operations that include BL and BM require prepending a ‘0’ to maintain the proper signed result. These are represented as BL{0,[63:0]} in the 128x64 multiply block and BL{0,[63:0]} and BM{0,[127:64]} in the 192x64 multiply block. The divide by 2 of the 128x64 multiply is implemented as a right shift by 1. BL is the lower 64-bits. Bu is the upper 64 bits.

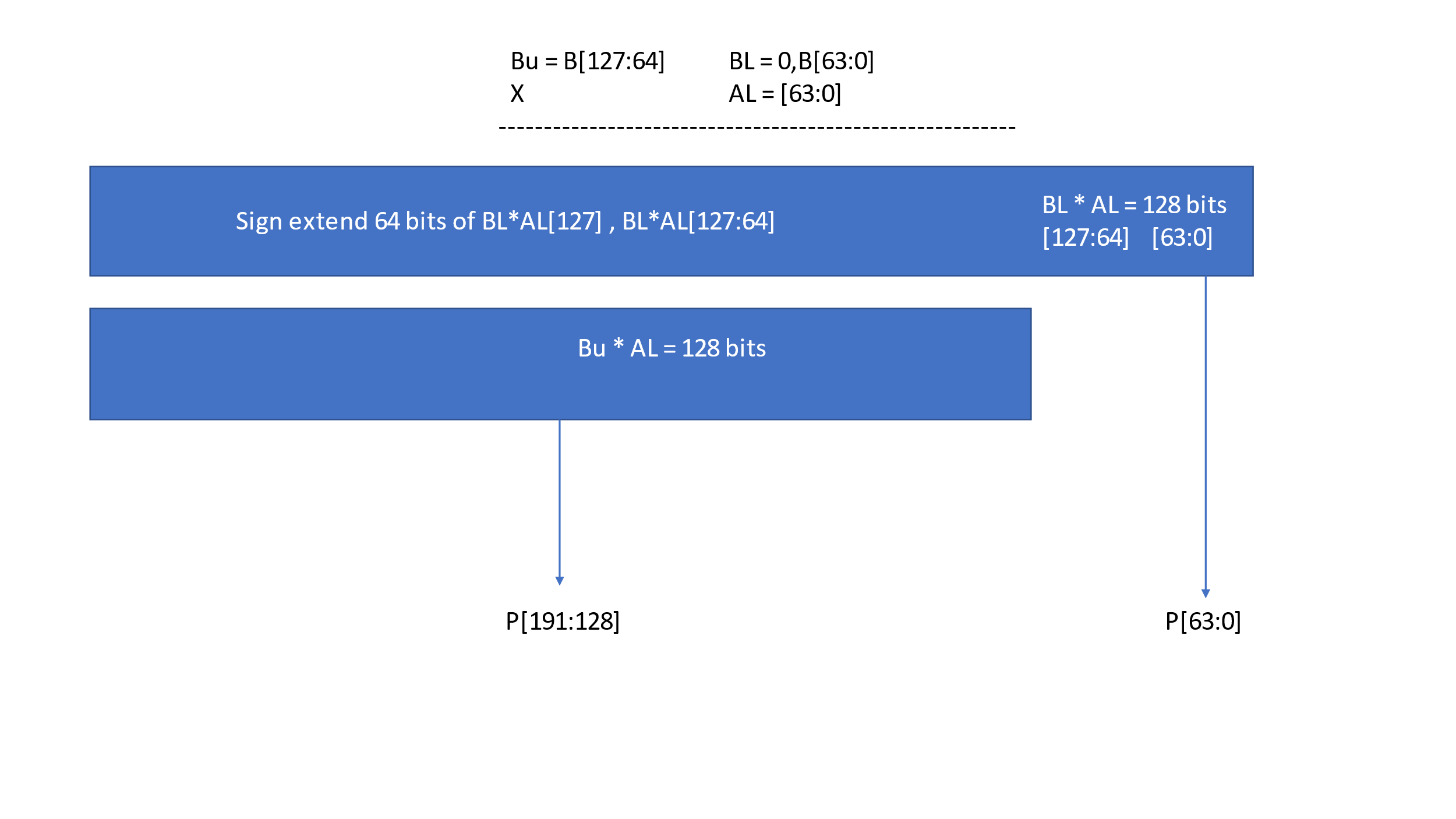


Figure 2 - 128x64 multiply block – bit view

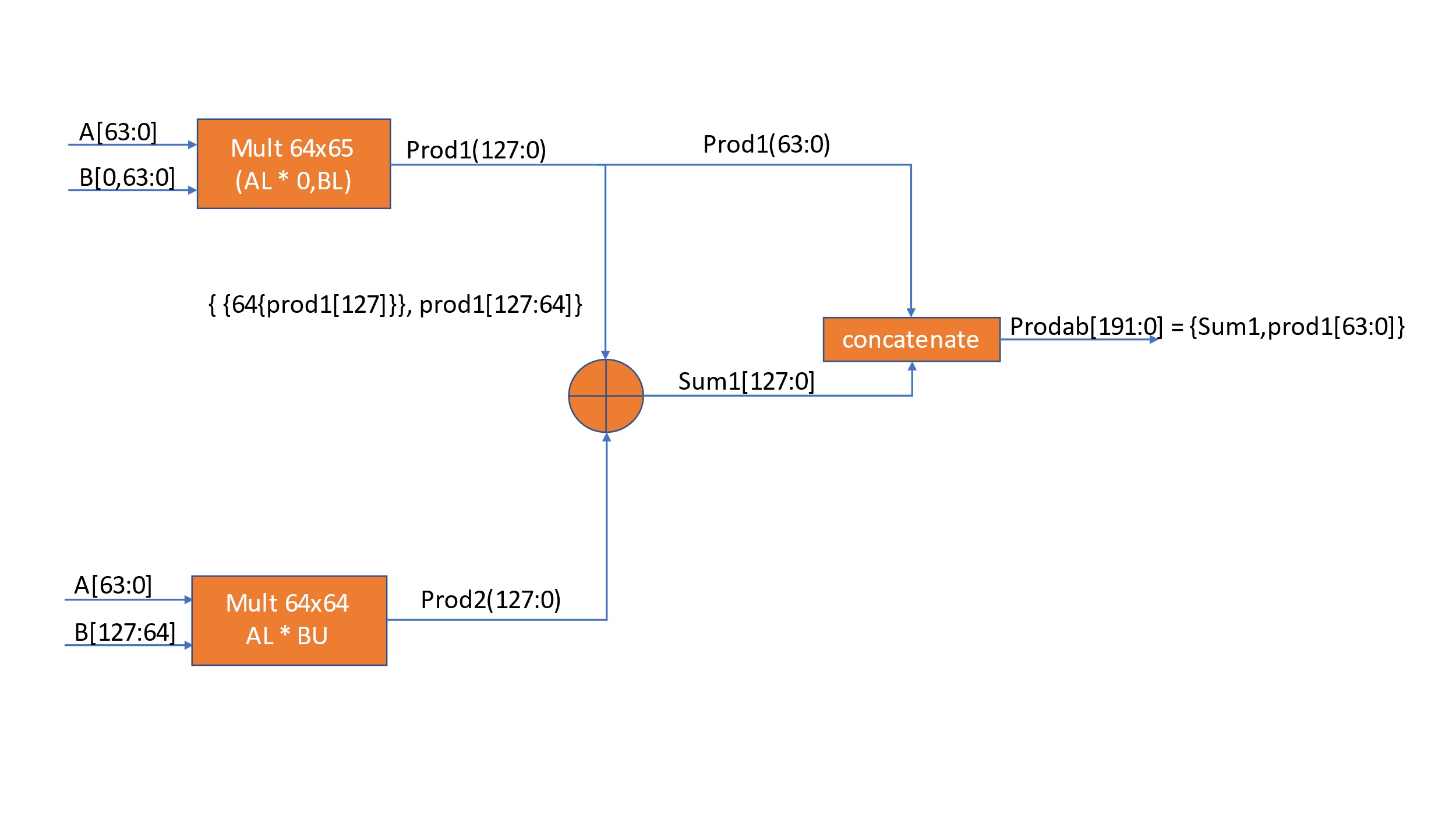


Figure 3 - 128x64 multiply block – functional block view

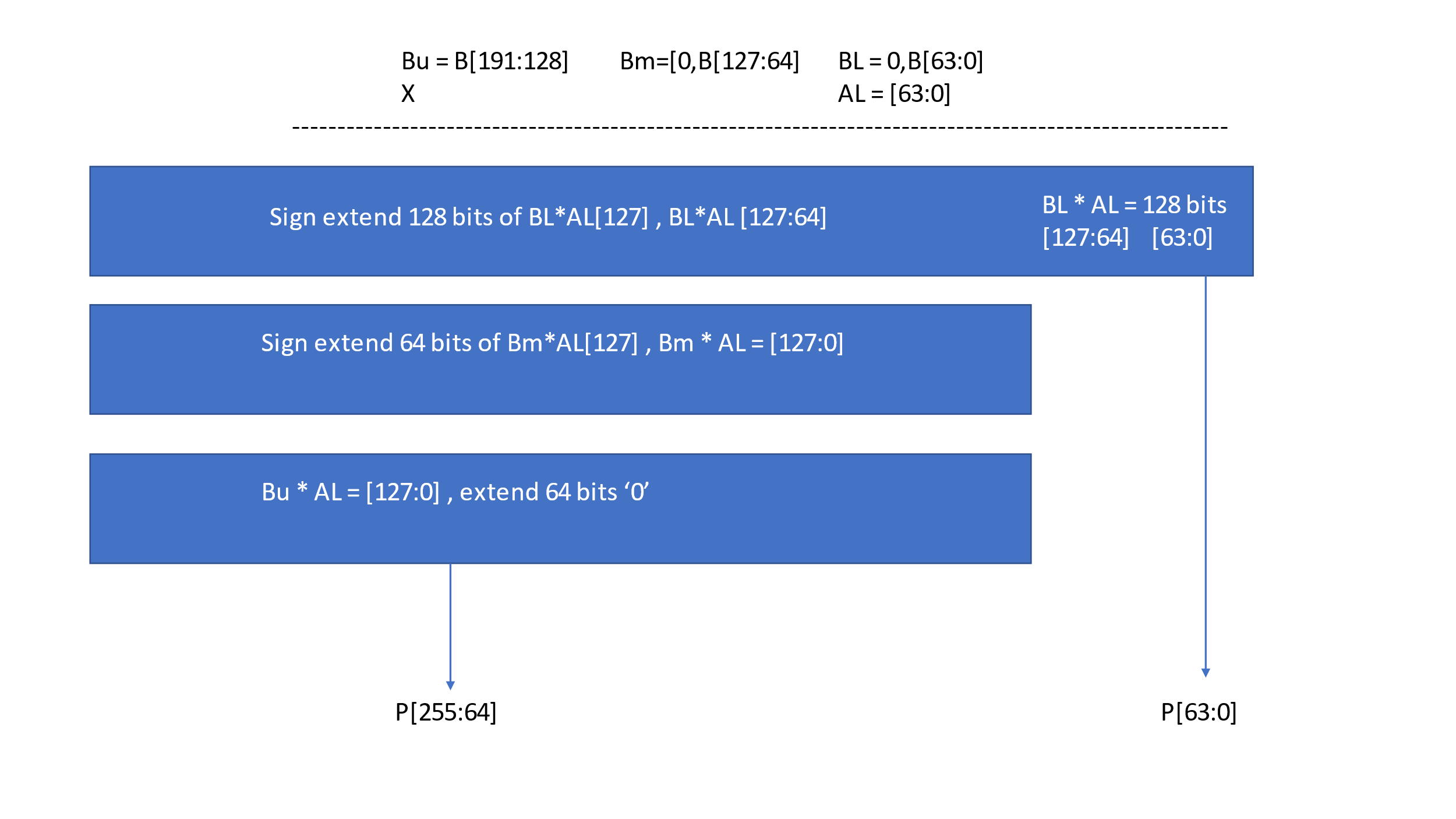


Figure 4 - 192x64 multiply block -bit view

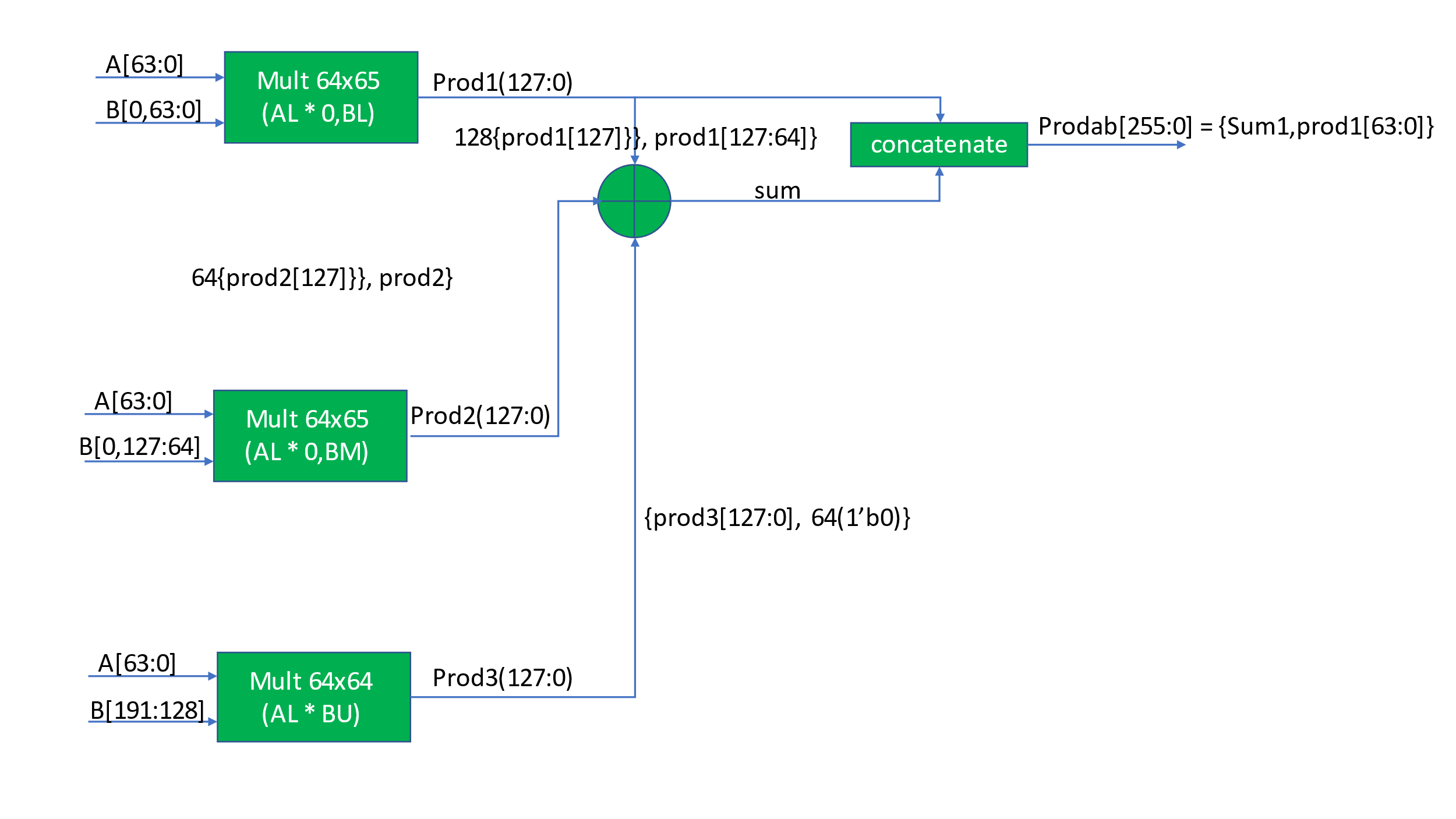


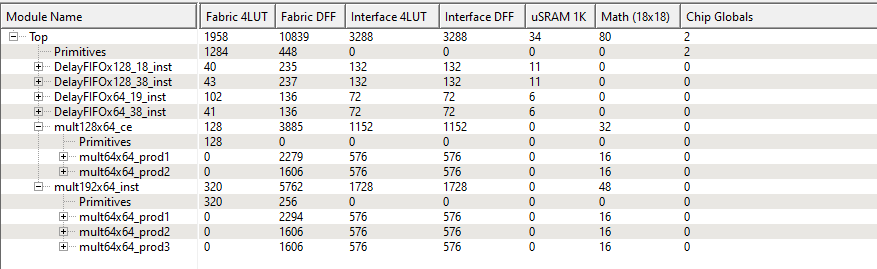
Figure 5- 192x64 multiply block -function block view

Table 1 - Port List

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Direction** | **Signal Description** | |
| resetn | Input | Asynchronous, active low | |
| clock | Input | clock | |
| A[127:0] | Input | signed data input | |
| B[63:0] | Input | unsigned data input | |
| C[63:0] | Input | signed data input | |
| D[127:0] | Input | signed data input | |
| E[127:0] | Input | signed data input | |
| WE | Input | active high write enable for FIFOs | |
| result[255:0] | Output | signed data output | |
|  |  |  |  |

**Utilization**

The chart below shows the hierarchical utilization of the design. Each of the 69x69 multiplies consumes 16 math blocks, with the entire design using a total of 80. Using FIFO block comprised of uSRAMs saved 10,552 SLE’s. The uSRAMs also require interface DFFs.



**Performance**

Static timing analysis of the block shows the max frequency of the block ~207 MHz.

**Simulation**

The top level testbench for the design is WideMultTopPf\_Tb.v located in the project stimulus directory. The do file for Modelsim is run.do located in the project simulation directory along with a wave.do file to add signals of interest to the wave window. The testbench provides a set of 47 input test vectors (as a sanity check) for each of the function inputs, a\_stim, b\_stim, c\_stim, d\_stim, and e\_stim. There are checks included to verify each stage of the calculation. The calculated result is compared to the expected result and printed out in the log window. To run the simulation, execute the do file which will run for 50 us.

You should see the following output in the simulation log window, the first correct result is 0c:

time=3437000

# result= 000000000000000000000000000000000000000000000002000000000000000a

# check\_result\_widemult=000000000000000000000000000000000000000000000002000000000000000a

#

# time=3443000

# result= 0000000000000000000000000000000000000000000000030000000000000018

# check\_result\_widemult=0000000000000000000000000000000000000000000000030000000000000018

#

# time=3450000

# result= 0000000000000000000000000000000000000000000000040000000000000034

# check\_result\_widemult=0000000000000000000000000000000000000000000000040000000000000034

The final value should be:

time=3731000

# result= 00000000000000000000000000000000e2468ad16007b2c9fffffffdb34fe912

# check\_result\_widemult=00000000000000000000000000000000e2468ad16007b2c9fffffffdb34fe912

Figure 6 shows the period to complete one full calculation, with the individual components displayed in blue. The initial multiply (128x64) occurs between cursors 1 and 2 taking 17 clock cycles. There is a one clock cycle delay after the first add shown between cursors 2 and 3. The second multiply (192x64) requires 18 clock cycles (17 cycles for the multiply and one cycle for the registered result) and is shown between cursors 3, 4, and 5. There is a one clock cycle delay after the final adder shown between cursors 5 and 6.

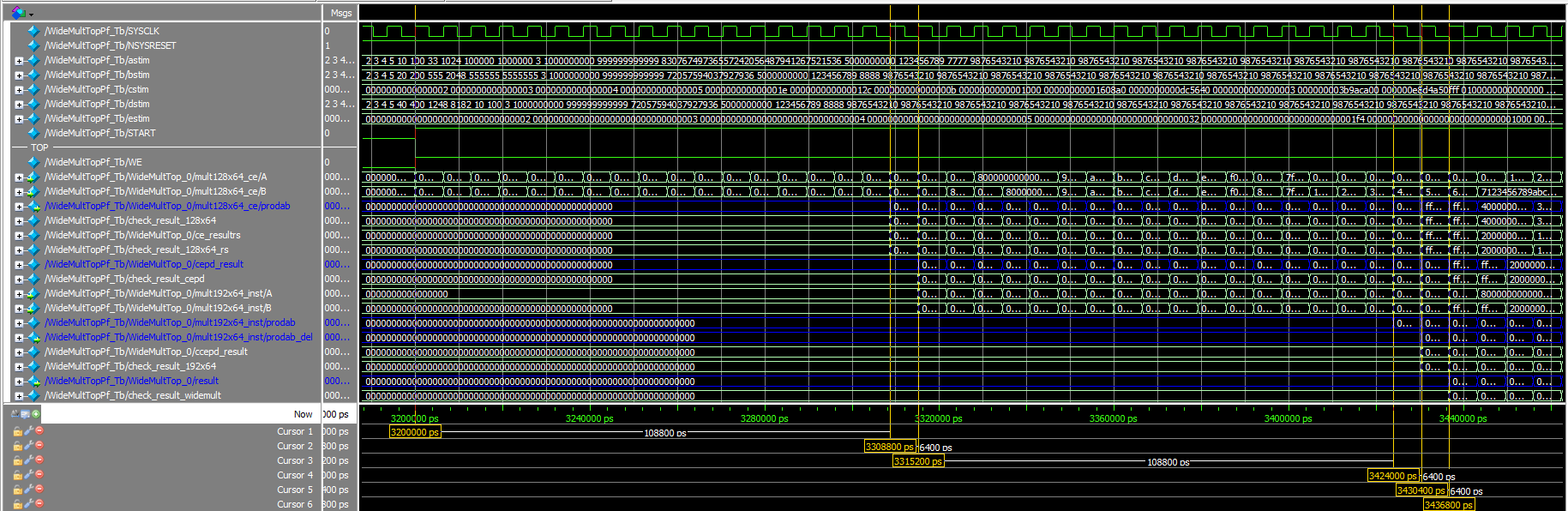


Figure 6 - Waveforms showing full calculation period

Figure 7 contains an example of 128x64 multiply output with its associated check shown in blue. The full list of calculation checks (computed using behavioral Verilog code) included in the testbench and wave file are as follows:

Check result 128x64 checks the output of the 128x64 multiply

/WideMultTopPf\_Tb/WideMultTop\_0/mult128x64\_ce/prodab

Check\_result\_128x64\_rs checks the output of the 128x64 multiply divided by 2

/WideMultTopPf\_Tb/WideMultTop\_0/ce\_resultrs

Check result\_192x64 checks the registered output of the 192x64 multiply

/WideMultTopPf\_Tb/WideMultTop\_0/ccepd\_result

Check result\_widemult checks the output of the entire calculation

/WideMultTopPf\_Tb/WideMultTop\_0/mult128x64\_ce/prodab



Figure 7 - Example of Multiply output with check