

Hardware/Software Co-Design

Introductory Laboratory Report

Dot product using `axil_macc` IP

1. Consider the C-based synthesis of the `axil_macc` IP.

a) Indicate the performance (latency and minimum clock period) estimates for your IP. How many clock cycles does the synthesized multiplier take?

As shown in Figure 1, the estimated minimum clock period is 6.912 ns, the total latency is 30 ns and the number of cycles is three.

b) List the main I/O ports of your IP and relate them to the AXI-Lite interface channels.

As shown in Figure 2, the main I/O ports are `a`, `b`, `c` and `instr`. These ports are accessible to the processor as memory-mapped registers via the AXI interface `s_axi_BUS1`, whose addresses are listed in the **SW-to-HW Mapping** table. Additionally, Vitis added an extra memory-mapped register `c_ctrl`, which can be used by the processor to verify if the computation of `c` has completed.

2. Consider the C/RTL co-simulation of the `axil_macc` IP and the RTL waveforms obtained.

a) Explain briefly how the AXI-Lite interface works and which AXI-Lite signals are active when the input data and control values are written to the IP, for the first multiplication-accumulation of the simulation.

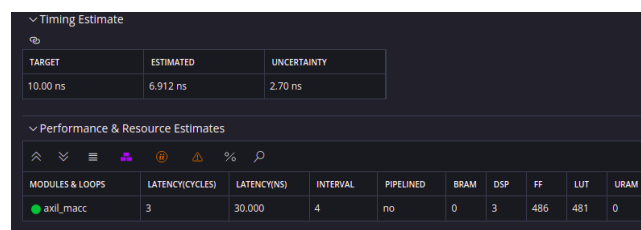
The AXI-Lite interface supports read and write transactions, each following a simple handshake protocol using valid/ready signaling. The valid signal is asserted by the source to indicate the information is available, while the ready signal is asserted by the destination to show it is ready to accept the information. A handshake completes when both valid and ready are high simultaneously.

The Write Transaction is composed of 3 parts: *Write Address Phase*, *Write Data Phase* and *Write Response Phase* - each one has a separated channel for realize the handshake - which sends address, write data and write response as information. Only in the *Write Response Phase* the slave acts as the source, otherwise it works as the destination.

The Read Transaction is composed of 2 parts: *Read Address Phase*, *Read Data Phase*. Each phase uses a separate channel to perform a handshake, transferring the write address, data, and response, respectively. Only in the *Read Data Phase* the slave acts as the source, otherwise it works as the destination.

As shown in Figure 3, a write transfer begins with the master setting the write address and asserting the address valid signal. The slave then responds by asserting the address ready signal. Once the address has been accepted, the master proceeds to place the write data on the bus and asserts the write data valid signal. Then, the slave responds by asserting the write data ready signal. After, the slave asserts the valid and response signals as 1 and 00, respectively, showing that the transaction succeeds. After the master responds with a ready signal equal to 1, finishing the entire transaction.

This process is repeated three times in the figure, corresponding to the transmission of the values for `a`, `b`, and `instr`.



Timing Estimate		
TARGET	ESTIMATED	UNCERTAINTY
10.00 ns	6.912 ns	2.70 ns

Performance & Resource Estimates									
MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
axil_macc	3	30.000	4	no	0	3	486	481	0

Figure 1: Timing and Resource Estimation in Vitis

> HW Interfaces			
v SW I/O Information			
v Top Function Arguments			
ARGUMENT	DIRECTION	DATATYPE	
a	in	int*	
b	in	int*	
c	out	int*	
instr	in	int*	
v SW-to-HW Mapping			
ARGUMENT	HW INTERFACE	HW TYPE	HW INFO
a	s_axi_BUS1	register	name=a offset=0x10 range=32
b	s_axi_BUS1	register	name=b offset=0x18 range=32
c	s_axi_BUS1	register	name=c offset=0x20 range=32
c	s_axi_BUS1	register	name=c_ctrl offset=0x24 range=32
instr	s_axi_BUS1	register	name=instr offset=0x30 range=32

Figure 2: I/O Ports of the IP in Vitis

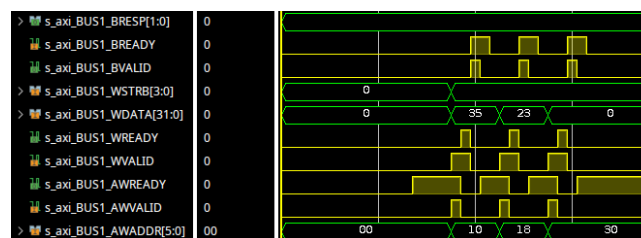


Figure 3: AXI-Lite Write Transaction Handshake

b) Indicate at which time the read transaction of the first partial result value is accomplished. What are the signals (and signal values) that define when the transfer takes place?

The master receives the first data at 595 ns. As shown in Figure 4, this read transaction involves the use of the Read Address and Read Data channels, specifically the valid, ready, and data/address signals. The transaction happens when both valid and ready are equal to 1.

c) Indicate at which time the read transaction of the final done control signal is accomplished. What are the signals (and signal values) that define this transaction?

The master receives the final data at 5095 ns. As shown in Figure 5, this read transaction involves the use of the Read Address and Read Data channels, specifically the valid, ready, data/address signals, the same signals of the previous question. The transaction happens when both valid and ready are equal to 1.

3. Consider your PS+PL system

a) Which base address has been assigned to the axil_macc IP?

The axil_macc IP base address is 0x4000_0000.

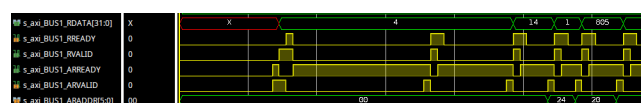


Figure 4: Read Transaction of the first partial result

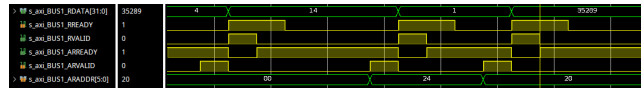


Figure 5: Read Transaction of the final result

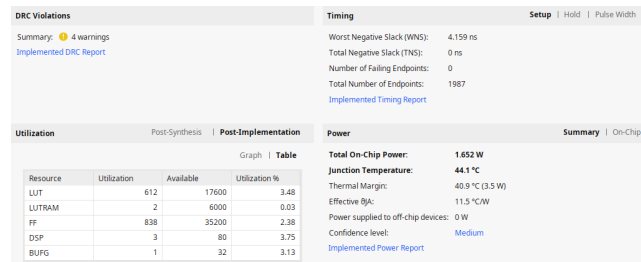


Figure 6: Timing and Resource Utilization in Vivado

b) Indicate the number (total and per component) of resources needed to implement your PL design (in terms of LUTs, FFs and DSPs).

As shown in Figure 6, the number of LUTs, FFs and DSPs utilized are 612 (two implemented as LUTRAM for shift-logic), 838 and 3, respectively. No BRAMs are used.

c) Compare the real resource consumption of the axil_macc IP with the high-level estimate (from Vitis HLS).

From Figure 1, the Vitis estimation are 3 DSPs, 486 FFs and 481 LUTs. This difference happens because Vivado considers the device's physical limitations, meaning the digital architecture might need to be spread out and decomposed into multiple sections of the LUT in order to meet demands. While Vitis performs only out-of-context synthesis, which does not take the target board's characteristics into account. The number of DSP's stays the same, because our core logic - the multiply and accumulate - is the same.

d) Could your PL system use a faster clock frequency? Justify briefly.

As shown in Figure 6, the design has a Worst Negative Slack (WNS) of 4.159 ns, indicating that the clock period could theoretically be reduced to approximately 6 ns. However, to operate at this higher frequency, the board must support generating such a clock using PLLs.

Matrix product IP

4. Consider the HLS matprod IP for full matrix multiplication, with an AXI-Lite interface using 3 BRAM_36K = 6 BRAM_18K local memories. Consider the matrix elements as 32-bit integers. In one execution, the new IP does a full matrix multiplication. The dimensions of the matrices are configurable, up to the defined MEM_SIZE (per matrix).

a) Do a C/RTL Co-simulation, using the 4x4 matrices in file m4.bin (add the input matrix file as a project testbench file) and using MEM_SIZE = 16. Indicate at which time the read transaction of the data read of the last element of the result matrix is accomplished.

The last read transaction finishes at 5.505 ns.

Implement the IP (using MEM_SIZE = 1024) in a PS+PL system.

b) Indicate the number (total and per component) of resources needed to implement the PL part of the design.

As shown in Figure 7, the total number of LUTs, FFs, DSPs and BRAMs utilized are 849, 1081, 6 and 3, respectively. The AXI-Lite interface spends 381 LUTs and 564 FFs. The Matrix-Product circuit uses 452 LUTs, 484 FFs, 6 DSPs and 3 BRAMs. The System Reset utilizes 17 LUTs and 33 FFs.

Name	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	Block RAM Tile (60)	DSPs (80)	Bonded IOPADs (130)	BUFGCTRL (32)
design_1_wrapper	849	1081	318	845	4	3	6	130	1
design_1_i[design_1]	849	1081	318	845	4	3	6	0	1
axi_smc[design_1_i]	381	564	174	380	1	0	0	0	0
axi_matprod1_0[di]	452	484	161	450	2	3	6	0	0
processing_system7	0	0	0	0	0	0	0	0	1
rst_ps7_0_100M[de]	17	33	9	16	1	0	0	0	0

Figure 7: Resource Utilization per Component in Vivado

Estimated Quality of Results

Timing Estimate

TARGET	ESTIMATED	UNCERTAINTY
10.00 ns	6.912 ns	2.70 ns

Performance & Resource Estimates

☒ Modules
 ☒ coops
 ☒ Hide empty columns

VIOLATION TYPE	DISTANCE	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
		2008	2.008E4	-	2002	-	loop auto-new	6	6	1152	1129	0

Figure 8: Timing and Resource Estimation in Vitis

c) Compare the real resource consumption of the axi_matprod IP with the high-level estimate (from Vitis HLS).

From Figure 8, the Vitis estimation are 6 DSPs, 1152 FFs and 1129 LUTs. This difference happens because Vivado considers the device's physical limitations, meaning the digital architecture might need to be spread out and decomposed into multiple sections of the LUT in order to meet demands. The number of DSP's stays the same, because our core logic - the matrix multiplication - is the same.

d) Estimate the maximum clock frequency that can be used in the PL.

By 7, the PL has a Setup Worst Negative Slack (WNS) of 3.537 ns, so the estimated maximum clock frequency is 6.463 ns.

Complete the C application to execute the matrix multiplication on your HW/SW system.

Add a function to compare the results between the software-only and the HW/SW application.

Place all the sections of your program on the OCM.

e) Measure the execution time of the HW/SW application (executing on the Zynq device on your board) for 25×25 matrices.

Console Output ⇒ Run in Zybo

Execution took 1009744 clock cycles.
SW Execution took 1553.45 us.

Execution took 470040 clock cycles.
(25) HW Execution took 723.14 us.

f) Compare the measured performance with that of the software-only 25×25 matrix multiplication.

As we can see from the output above, there is a speed-up of 2.15 when comparing to the software version - which runs on the on-board processor - this speed-up is due to having a custom hardware accelerator dedicated to the matrix multiplication, instead of relying in a general purpose processor to run a set of instruction. We could further speed-up the process if we used DMA, where the PL would directly fetch the matrix values from memory with a High Performance AXI-Lite Bus instead of have the processor feeding us said values with a General Purpose AXI-Lite Bus.

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 3.537 ns	Worst Hold Slack (WHS): 0.025 ns	Worst Pulse Width Slack (WPWS): 4.020 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 2924	Total Number of Endpoints: 2924	Total Number of Endpoints: 1097
All user specified timing constraints are met.		

Figure 9: Timing Summary in Vivado

5. Estimate the performance improvements that could be achieved if the input matrix elements were 8-bit integers.

We can easily say that we would get a considerable speed-up because the arithmetic units necessary to compute the multiplication would be 4 times smaller. Therefore, ideally, we could get a 4x speed-up. Surprisingly, when modifying the base `matprod` project to support `int8_t` instead of `int32_t`, the speed-up did not change - the execution times were the same.

There are some possible reasons for this. First, both versions ran at the same clock frequency, although the version using smaller data types could, in theory, support a higher clock rate. Second, the processor's AXI-Lite interface has a 32-bit data width. Simply reducing the data size to 8 bits led to underutilization of the bus — only 8 out of 32 bits were used per transfer. A similar underutilization occurred within the FPGA: the same number of operations were performed, but on smaller data types, without exploiting parallelism.

To approach the ideal 4x speed-up, data-level parallelism must be used. For example, each 32-bit bus transfer should carry four 8-bit values, and the FPGA should process these values concurrently.