ECE260B Project Report

Members: James Yuan, Anmol Nijhawan, Rishav Karki, Rahul Polisetti

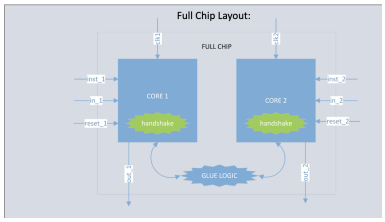
Team JARR

March 24, 2023

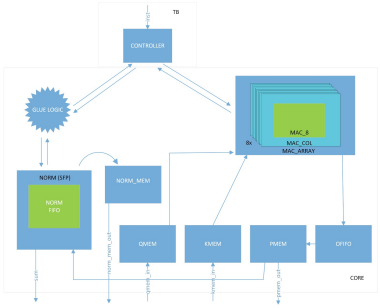
1 Baseline Design

1.1 RTL

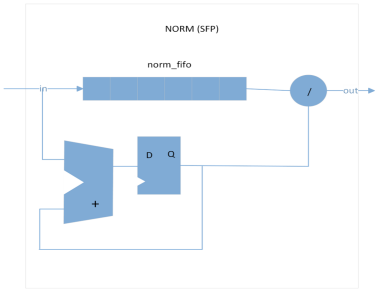
In this project, we implemented a Dual core Machine Learning Accelerator for Attention Mechanism using a 1D Vector Processor Based Architecture. The Accelerator contains two cores that communicate via a handshake protocol. Each core contains two MAC arrays, that compute the product of two matrices. An SFP Normalizer is used to normalize the products of the matrices and the final normalized data is written into a memory.



(a) Full chip block diagram



(b) Baseline core module block diagram



(c) SFP Normalizer

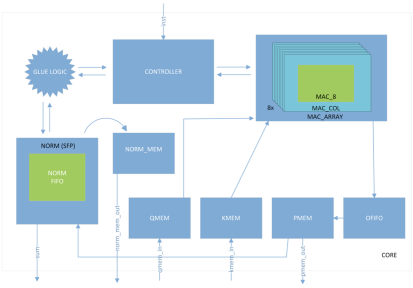
Figure 1: **Overview of Baseline Design**

In the baseline design that we developed, a single core can compute the product of two matrices of dimension $RX8$ (Q matrix) and $8X8$ (K matrix) at one shot, to generate an output matrix of dimension $R * 8$. The partial sums of the matrix multiplication are written into a FIFO (OFIFO) , and from the FIFO, the vectors are read and written into an SRAM memory (PMEM).
The testbench acts as a non synthesizable controller for the baseline RTL. It loads the SRAM

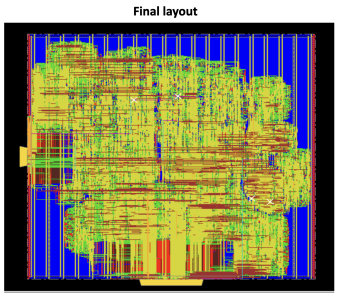
memories, KMEM and QMEM, and it also loads the MAC Array's registers with the KMEM values. Once the MAC Array's registers are loaded, the testbench begins the execution of the MAC Array, and it streams the input vectors from QMEM. The output data from the MAC Array is written into a FIFO (OFIFO). The data is read from the OFIFO and written into PMEM, once the OFIFO detects that it has valid data. Finally, the testbench reads data from PMEM for comparison with the expected values. It also loads this data in 8 shift registers and shifts out one word of data every cycle into the SFP normalizer.

The SFP normalizer latches the input stream of data, and adds the values serially to generate the sum. The input stream of data is also written into a FIFO (NORM FIFO). The testbench issues a 'div' pulse, which initiates a read transaction on the FIFO, and the read data is divided by the sum to generate the normalized output data.

2 Key Optimizations



(a) Mac Array Controller



(b) Layout

Figure 2: **Final Merged Design**

We have added the following optimizations in the final merged design :

2.1 Pipelining

Pipeline stages were added after Multiplier and in between adder stages, to meet timing. In the MAC Array, the multiplier and adder trees are on the critical paths. These paths need to be broken by adding pipeline registers on the datapath as well as the control path, to meet timing.

2.2 Multicycle Path

In the Synthesis and PNR sdc files we added a Multicycle Path of 6 cycles on the divider used in the SFP normalizer. Fixed point division is a very latency intensive operation, and adding a multicycle path on the divider logic is necessary in order to meet timing at 1 GHz clock frequency.

2.3 Handshake Protocol

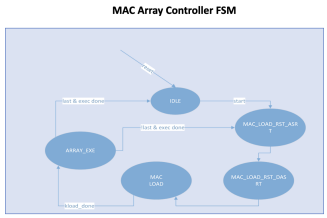
We added a controller that orchestrates the 4-Phase handshake protocol between the two cores that are asynchronous to each other. While an Asynchronous FIFO would be faster for multi-bit clock domain crossing, it would also consume more power than the handshake protocol. Also, since we only need to send the sum values from each core once every 50-60 cycles (Since the previous data needs to be read out from the FIFO every 6 cycles due to the divider's multicycle path), it gives us enough time to capture the sum from the other core, even if we use a slower protocol like the handshake protocol.

2.4 Clock Gating

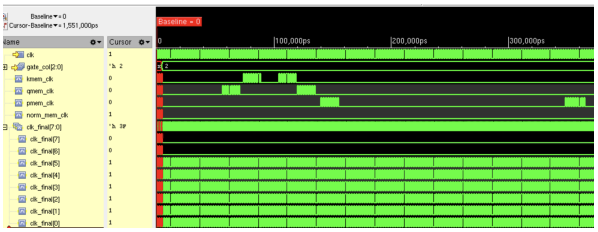
We added latch-based clock gating for the memories and the MAC Array to reduce dynamic power consumption. The ICG enable signal is latched on the negative edge of the clock, to avoid potential glitches. The memories are clock gated when the enable signals (CEN) to the memory are de-asserted. Additionally, the MAC Array controller also clock gates the columns of the MAC Array if the number of columns of the KMEM matrix are less than 8. This signal is driven by the Testbench (gate_col) currently in our design, and it indicates to the MAC Array controller that it can clock gate columns of the MAC Array that won't be used for computation.

2.5 MAC Array Controller

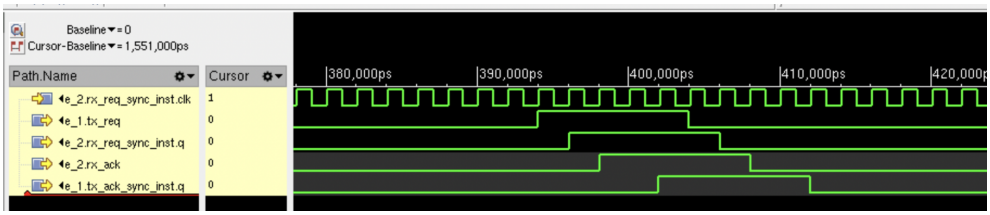
We added a controller inside core that schedules operations in the MAC Array, fetches data from the kmem and qmem memories, and writes data to the OFIFO. This controller is synthesizable and performs the same operations that the testbench performed in the baseline design.



(a) Mac Array Controller



(b) Clock Gating



(c) Handshake Protocol

Figure 3: **Optimizations**

2.6 Reconfigurable mode operation

The core can also be reconfigured to compute the products for the following input data formats - 1) 4b signed (QMEM) /4b unsigned (KMEM) data , and 2) 4b signed (QMEM) /8b unsigned (KMEM). For signed data and unsigned data multiplication, the unsigned data is sign extended to generate the correct signed data product. In 4b/8b mode, two adjacent columns of the MAC array generate partial products which need to be combined again to generate the partial sum. This is done by latching the partial sums from the odd index mac columns (1,3,5,7) and adding the latched partial sum value to the partial sums from the even index mac columns (2,4,6,8) left shifted by the bitwidth (4 bits).

$$Psum_{final}(n) = Psum_{odd}(n-1) + (Psum_{even}(n) << 4)$$

2.7 Thorough verification

We implemented a python script to enable thorough verification of our design with randomization of input vectors to improve coverage. The script takes a index and two optional

arguments R and W. R is the total cycle or the length of Q matrix (default is 24) and W is the width of the K matrix (default is 8). The script then generates random input vectors of the same data format based on the given shape. With the new data sets we were able to fully test our design.

3 Results and Observations

PNR	Area (μm^2)	Internal Power (mW)	Switching power (mW)	Leakage Power (mW)	Total Power (mW)	Slack (ps)
Part 1 (with pipeline)	335471.200	101.39	33.73	1.93	137.05	-0.000
Part 2	179521.440	54.83	14.50	0.74	70.073	-0.011
Part 3	555320.480	173.83	66.94	2.30	243.07	-0.475
Part 4	1092442.040	324.70	161.89	5.08	491.67	-0.255
Final Merged	586449.280	155.24	58.85	2.11	216.21	-0.037

Final Merged	Internal Power (mW)	Switching power (mW)	Leakage Power (mW)	Total Power (mW)
Activity =0.2	155.24	58.85	2.11	216.21
Using VCD	30.83	7.69	2.18	40.70

Figure 4: **PnR Results**

Based on the PnR results, we observed that with pipelining, we were still not able to meet timing for part 1 (**WNS between 0 and -1ps, with pipelining**), however there was considerable improvement (**WNS of -620 ps without pipelining**). Also, we were not able to meet timing for Part_3, Part_4 and Final_merged on the divider path, despite using multicycle paths. For the final merged design, we observed **a worst case negative slack of -37 ps** . We observed a **significant reduction in total power for Part_2 and Final_merged (approximately 50% from Part_1 and Part_4 respectively)**, since the numerical bit precision of the MAC Array inputs was reduced to 4 bits for the reconfigurable mode updates. This is expected, as a reduction in bit precision implies that fewer bits will toggle (reduction in dynamic power) and static leakage will also reduce as there are fewer standard cells. For Part_1 , Part_3 and Part_4 , the bit precision of the inputs is 8 bits. We also observed **81% reduction in total power consumption** in the final merged design (with clock gating) when we ran the VCD based PnR netlist simulation, and then analyzed the power consumption in Innovus for WC lib. The gate level simulation with PnR netlist is passing without any errors in functionality, when the clock frequency is reduced to 500 MHz. At 1 GHz, there are many timing violations as we were not able to achieve a WNS slack of 0. The final merged design had **10 DRC violations**, (8 short and 2 wiring).

4 Challenges faced

4.1 Debugging X-Propagation in Gate Level Simulation

Initially we ran synthesis in hierarchical mode. Gate level simulations have multiple initialization issues in hierarchical mode. It seems that many flip flops were not getting reset at the start of simulation, and we were observing x-prop. Temporarily, we were running with deposits on these flops to initialize the system. We were sourcing a tcl file in xcelium

to enable this flow. Later, we ran synthesis in flattened mode, once we figured out how to identify the names of the registers for PnR multicycle SDC constraints. In this case, we observed that the gate level simulations worked as expected without any deposits.

4.2 Power Analysis in Synthesis vs PnR

Power analysis after PnR seems inconsistent with post synthesis numbers in our final merged design, when clock gating and multicycle path is used (almost a 10x drop in power). The clock network power is absent in PnR power analysis (probably merged in some other category in PnR power reports), but in synthesis, this power contributes to around 90-95% of the total power. We suspect issues in our SDC file, and we are still debugging this. We believe additional constraints need to be specified for clock gating to get more realistic numbers.

4.3 SDC file updates

We had to update SDC file for clock domain crossing and multicycle paths. We added the following commands to handle these scenarios -

```
(Added for Clock Domain Crossing)
create_clock -name clk1 -period $clock_cycle [get_ports $clock_port1]
create_clock -name clk2 -period $clock_cycle [get_ports $clock_port2]

set_clock_groups -logically_exclusive -group [get_clocks clk1] -group [get_clocks clk2]


(Added for Multicycle Path)
set_multicycle_path -setup 6 -from [get_pins
core_instance_1/norm_instance/fifo_top_instance/fifo_instance/rd_ptr_reg[*]/clocked_on ] -to
[get_pins core_instance_1/norm_instance/out_reg[*]/next_state ]
set_multicycle_path -hold 5 -from [get_pins
core_instance_1/norm_instance/fifo_top_instance/fifo_instance/rd_ptr_reg[*]/clocked_on ] -to
[get_pins core_instance_1/norm_instance/out_reg[*]/next_state ]

set_multicycle_path -setup 6 -from [get_pins
core_instance_1_norm_instance_fifo_top_instance_fifo_instance_rd_ptr_reg_*/CP ] -to
[get_pins core_instance_1_norm_instance_out_reg_*/D ]
set_multicycle_path -hold 5 -from [get_pins
core_instance_1_norm_instance_fifo_top_instance_fifo_instance_rd_ptr_reg_*/CP ] -to
[get_pins core_instance_1_norm_instance_out_reg_*/D ]
```

Figure 5: **SDC reference commands**

5 Paths to all the deliverables

1) **Part 1**
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_1/rtl_sim – Contains RTL, and PnR Gate level VCD. To run GLS simulation : source run_gui_gls
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_1/pnr contains route.enc and all PnR timing, power and area reports.
Waveform VCD : /home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_1/rtl_sim/fullchip_tb.vcd
route.enc : /home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_1/pnr/route.enc

2)**Part 2**
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_2/rtl_sim – Contains RTL, and RTL VCD. To run RTL simulation : source run_gui
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_2/pnr contains route.enc and all PnR timing, power and area reports.
Waveform VCD : /home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_2/rtl_sim/fullchip_tb.vcd

3)**Part 3**
/home/linux/ieng6/ee260bwi23/anijhawan/project/Part_3/rtl_sim – Contains RTL, and RTL VCD.
/home/linux/ieng6/ee260bwi23/anijhawan/project/Part_3/pnr contains route.enc and all PnR timing, power and area reports.
Waveform VCD : /home/linux/ieng6/ee260bwi23/anijhawan/project/Part_3/rtl_sim/fullchip_tb.vcd

4)**Part 4**
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_4/rtl_sim – Contains RTL, and PnR Gate level VCD. To run GLS simulation : source run_gui_gls
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_4/pnr contains route.enc and all PnR timing, power and area reports.
Waveform VCD : /home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_4/rtl_sim/fullchip_tb.vcd
route.enc : /home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Part_4/pnr/route.enc

5) **Final merged design** :
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Final_merged/rtl_sim – Contains RTL, and PnR Gate level VCD. To run GLS simulation : source run_gui_gls
/home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Final_merged/pnr contains route.enc and all PnR timing, power and area reports.
Waveform VCD : /home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Final_merged/rtl_sim/fullchip_tb.vcd
route.enc : /home/linux/ieng6/ee260bwi23/rkarki/ee260b/project/Final_merged/pnr/route.enc