Dual-Core Machine Learning Accelerator for Attention Mechanism

Krish Mehta† Kumar Divij† Suraj Sathya Prakash† Ishan Bansal†
†University of California San Diego

***Index Terms*—Synthesis, Placement, Routing, Gate Level Simulation, Physical Design, Machine Learning Accelerator**

***Abstract*—Synthesis followed by Placement & Routing are essential stages in the design of semiconductor chips. In this project, we start with a basic behavioural design of a single core 1D Vector Multiplication Processor and expand it to a full Dual Core Machine Learning Accelerator. We perform careful placement and routing of the complete chip and verify our design with a Gate Level Simulation with physically simulated delays.**

I. Introduction

This project report outlines the development and testing process of a single core matrix multiplier, a crucial component in numerous digital systems. The project is segmented into several stages, commencing with the design and testing of the Register Transfer Logic (RTL) for the core. This is succeeded by the synthesis and Placement and Routing (PNR) processes, with a target frequency of 1GHz. The project also entails the computation of vectors comprising 8 elements, each being an 8-bit integer. The output normalization stage ensures the accuracy of the results, which are stored in the partial sum memory (pmem). This report will provide a comprehensive overview of each stage, including the challenges encountered and the solutions implemented to surmount them. We have also introduced a number of optimizations as a measure to ensure the viability of our design and make it capable of commodification.

We have inculcated the following optimizing ideas and presented them as α's over and above the baseline design mentioned in the introduction above.

1) **Pipelined Divider:** The biggest timing bottleneck is eliminated with this enhancement
2) **Asynchronous Interface:** Capable of handling two cores on significantly different clocks
   - Toggle Synchronizers are used to detect and pass a pulse even if two clock domains are very different, e.g. 1ns and 5ns
   - Req-Ack based 4-phase handshake algorithm exchanges accumulated sum information between two cores
3) **Sparsity Aware Columnar Clockgating:** A column of MAC Array can get clockgated to save power if all weights in that column are zero. A free clock makes sure the query is propagated to the next column
4) **Pipelined SFP Accumulation:** Addition of 8 partials sums is distributed into two stages
5) **Pipelined MAC:** Addition of 8 partial products is distributed across three stages
6) **Gate Level Simulations with SDF:** Performed Gate Level Simulation to verify the functionality of the complete placed and routed design with physically simulated delays

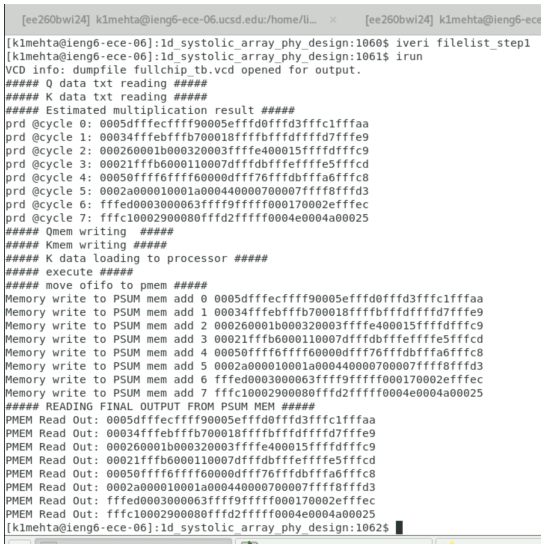

Fig. 1: Single Core RTL Verification, Baseline, No SFP

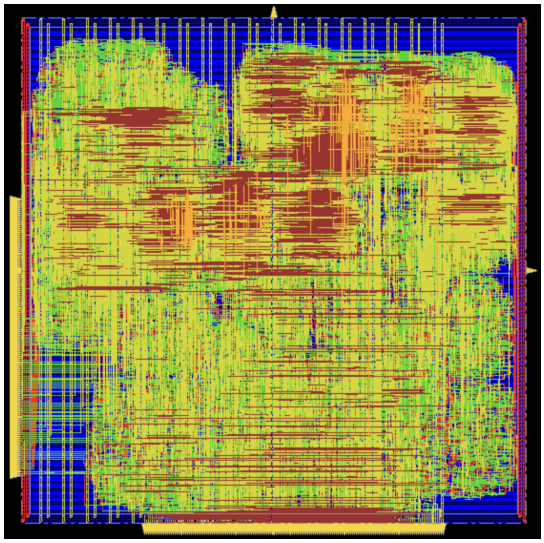

Fig. 2: Single Core Post-Routing layout, No SFP, Unoptimized

7) **Automated Verification:** Event driven testbench, with built-in checks for normalized outputs

II. **Step 1:** Single Core RTL, Verification, Synthesis & PNR

The output in $core.v$ was connected to $pmem\_out$. The output from the testbench was verified successfully in Fig. 1.

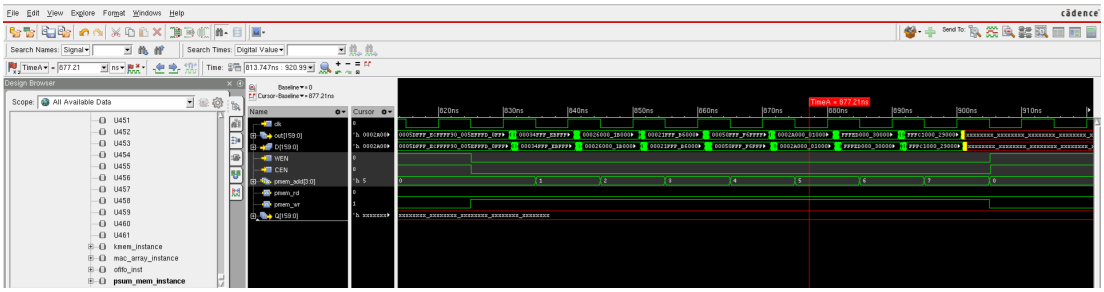

Fig. 3: Simple Core Gate Level Simulation from PNR files, **with SDF Annotation**, showing PMEM being written with MAC output values

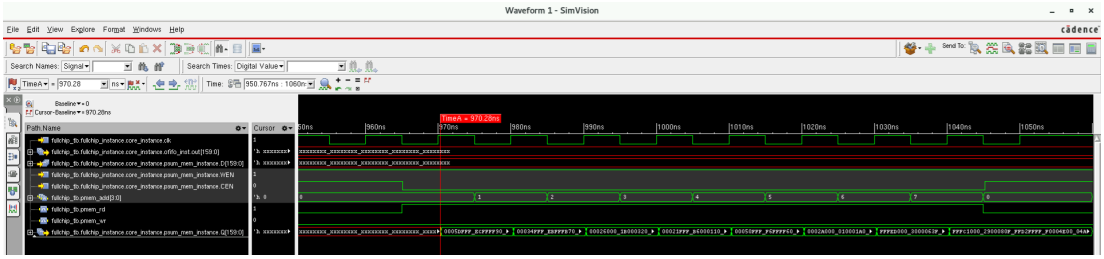

Fig. 4: Simple Core Gate Level Simulation from PNR files, **with SDF Annotation**, showing PMEM being read out to Core output pins

The single core was synthesized and PNR'd with a $4ns$ clock period to start with. As shown in Fig 2, the input pins were placed on the left side and the output on the bottom with $4\mu m$ spacing. The post-routing statistics are presented in Table I.

Setup Check	Hold Check	Area (μm^2)	Power (mW)	Core Density
-0.016 @ $4ns$	MET @ $4ns$	290305	33.40	51%

TABLE I: Single Core, Non-Heirarchical, Unoptimized, Post-Route Metrics

III. Step 2: Output Normalization

The control signals in the SFP module, acc, and div were assigned with the correct control signals $inst[17]$ and $inst[18]$, respectively. The SFP control logic to accumulate in the SFP and write back to the $pmem$ was added to the test bench. Behavioural simulation of SFP functionality is showing in Fig. 5 and Fig. 6. The output matched with the values expected by the testbench, as shown in Fig 7.

IV. Step 3: Hierarchical Synthesis Single Core

The single core was hierarchically synthesized and PNR'd with a $0.8ns$ clock period. As shown in Fig 8, the K SRAM was placed in the top left corner, below which the Q SRAM is placed. The Pmem SRAM was placed at the bottom-most level. The mac-array logic was left for the tool to place and route. The pin placement was optimized as follows:

- **Left:** Clock, Reset, Instruction and Memory inputs that go directly to the K, Q Mem or MAC inputs. K and Q Mem are flipped such that their D ports are towards the input pins of the core
- **Right:** The core's final output pins, i.e. normalized outputs which will ultimately become the output of the full chip
- **Bottom:** The partial accumulation of 8 terms that this core owns, i.e. sum_out, which is to be communicated with the SFP of another core

All the pins were spaced with $4\mu m$ spacing. The post-route statistics are presented in Table II and Table III.

Setup Check	Hold Check	Area (μm^2)	Power (mW)	Core Density
-0.862 @ $1ns$	-0.581 @ $1ns$	1122400	309.34	42%

TABLE II: Unoptimized Single Core Post-Route Metrics

However, after applying several optimizations within the core like: **Pipelined MAC, Pipelined Divider, Register based SFP with Pipelined Accumulation**, we are able to run it at much higher frequency with much smaller WNS.

Setup Check*	Hold Check*	Area (μm^2)	Power (mW)	Core Density
MET @ $0.8ns$	MET	1328400	704.27	43.5%

*reg2reg

TABLE III: Optimized Single Core Post-Route Metrics

V. Dual Core Implementation & PNR

After completing the Single Core design, we synthesized and PNR'd the complete chip, which had 2 instances of the core and synchronisation logic to handle the asynchronous interface between the two cores. As shown in Fig 14, the two cores are made with a wide aspect ratio so that **final chip is a 1:1 ratio** and can be fabricated on the die directly.

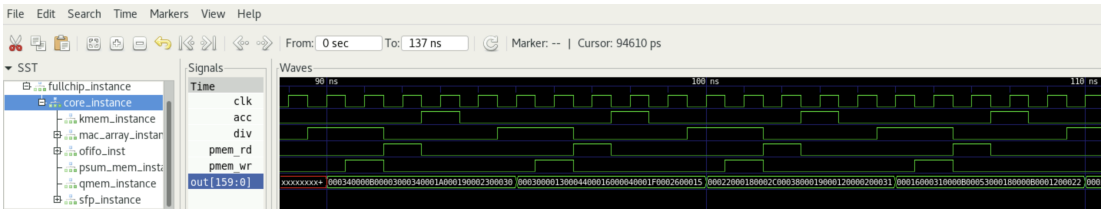

Fig. 5: Waveform to verify normalization (first 4 outputs)

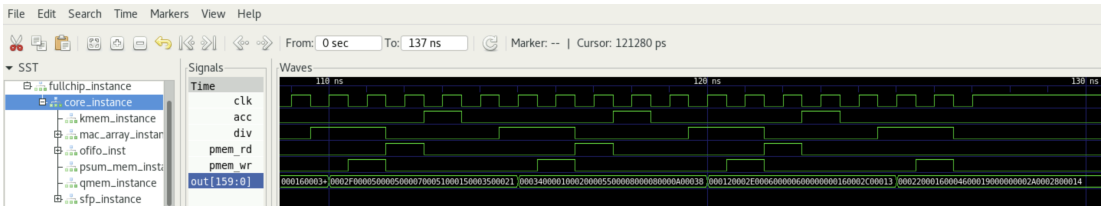

Fig. 6: Waveform to verify normalization (last 4 outputs)

Fig. 7: Output Normalization RTL Verification

Setup Check*	Hold Check*	Area (μm^2)	Power (mW)	Core Density
MET @ $0.8ns$	MET	33800	48.33	80.1%

*reg2reg

TABLE IV: K & Q Mem SRAM Post-Route Statistics

The pin placement was optimized to ensure simple and straightforward routing:

- **Right:** Inputs to the chip, i.e. instruction, memory inputs, clock, reset
- **Left:** Final output from the chip, i.e. normalized outputs and attention values

The routing between two cores is the exchange of partial accumulation values, through toggle synchronizers and the handshake mechanism.

Setup Check*	Hold Check*	Area (μm^2)	Power (mW)	Core Density
MET @ $1ns$	MET	3312400	1853.65	82%

*reg2reg

TABLE V: Full Chip Post-Route Metrics

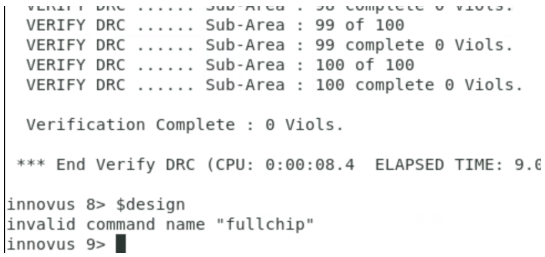

Fig. 15: The final routing of the full chip was completed with **0 DRC Errors**

VI. Functional Testbench & Gate Level Simulation

We have updated the testbench to do **automated checking** of final normalized values. The testbench calculates expected MAC and Normalized outputs based on the inputs and automatically matches then with the outputs from the design and prints a Matched/Not Matched result during the simulation.

We have rigorously performed Gate Level Simulations with **all timing checks enabled and SDF annotation** as shown by Fig. 19 and Fig. 20 and were **able to successfully verify the output of the core at 1.3ns and the fullchip at 1.8ns without any timing violations in Xcelium**. This makes it up to the mark for tape-out.

Without SDF annotation we are able to run both **core and fullchip at 1ns** and have functionally verified correct outputs at fullchip level.

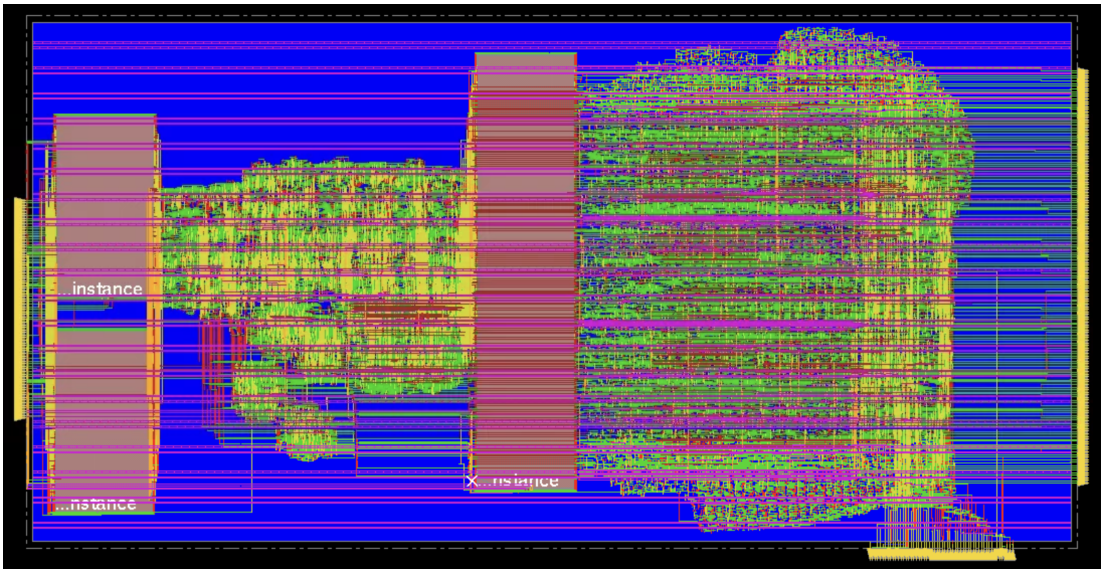

Fig. 8: Hierarchical Single Core Post-Routing layout

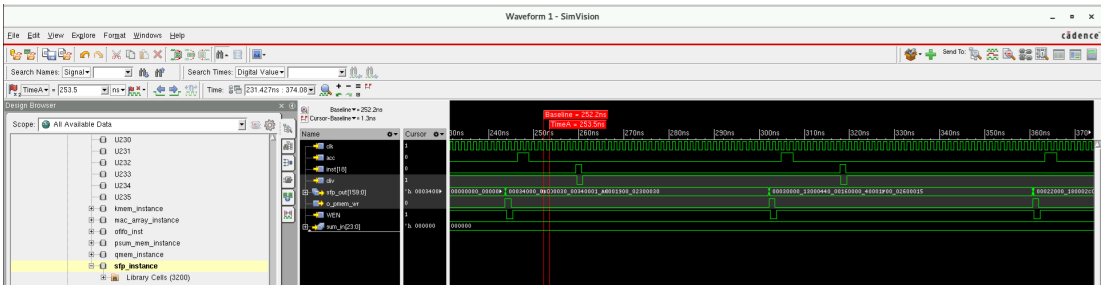

Fig. 9: Hierarchical Single Core **GLS running at 1.3ns with 91% SDF Annotation** from PNR information.

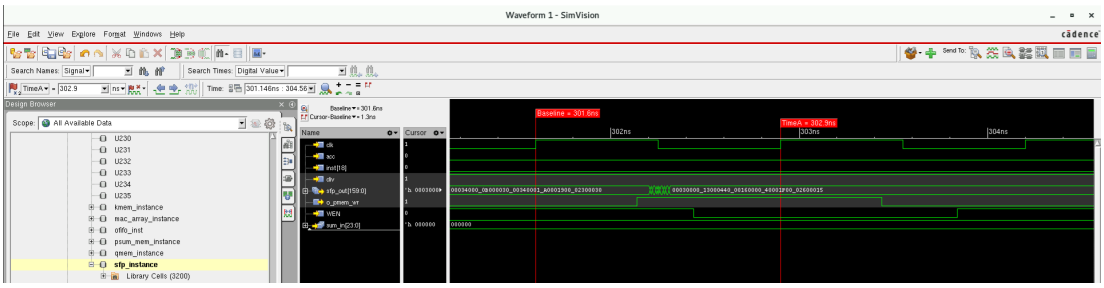

Fig. 10: SDF Annotated delays causing glitching and settling to final value well before clock positive edge. **If we go with constant 20ps delay, we can reach down to 0.8ns clock period, but we prefer to report realistic delays**

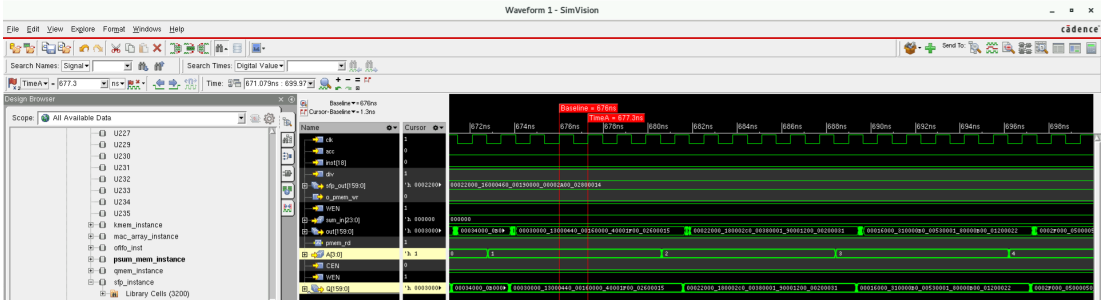

Fig. 11: The normalized values are read out from the PMEM and reflected at the output of the core. This completes Heirarchical Single Core verification through Gate Level Sim

```
  VERIFY DRC ..... Sub-Area : 41 of 45
  VERIFY DRC ..... Sub-Area : 41 complete 0 Viols.
  VERIFY DRC ..... Sub-Area : 42 of 45
  VERIFY DRC ..... Sub-Area : 42 complete 0 Viols.
  VERIFY DRC ..... Sub-Area : 43 of 45
  VERIFY DRC ..... Sub-Area : 43 complete 0 Viols.
  VERIFY DRC ..... Sub-Area : 44 of 45
  VERIFY DRC ..... Sub-Area : 44 complete 0 Viols.
  VERIFY DRC ..... Sub-Area : 45 of 45
  VERIFY DRC ..... Sub-Area : 45 complete 0 Viols.

  Verification Complete : 1 Viols.

 *** End Verify DRC (CPU: 0:00:28.0  ELAPSED TIME: 2

innovus 6> $design
invalid command name "core"
innovus 7>
```

Fig. 12: The final routing of Core was completed with only **1 DRC Error**

```
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 44 complete 0 Viols. 0 Wrngs.
  VERIFY GEOMETRY ..... SubArea : 45 of 45
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 45 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 35.00
Begin Summary ...
  Cells       : 0
  SameNet     : 1
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 1 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:35.3  MEM: 0.4M)

innovus 8> $design
invalid command name "core"
innovus 9>
```

Fig. 13: The final routing of Core was completed with only **1 Geometry Error**

VII. Optimization

A. Sparsity Aware Columnar Clock Gating

The 1D vector processing array for MAC Col consists of 8 MAC Col modules, each receiving 8 clock lines. These clock lines then pass through each MAC 8in the pipelined module. The modules operate using both free running clocks and gated clocks.

Clocks are gated when all weights in a column are zero, known as the Zero-Condition. In this case, gating masks are set to 1 only if the Mask Enable-Condition occurs, typically when kernel loading finishes. When the Zero-Condition occurs, and only the free clock is active, new data latching and MAC calculations are prevented.

On the other hand, the free-running clock remains unaffected by the gating mechanism. It ensures the proper movement of query data from the west to east within the vector processing array. This enables continuous data flow for processing, even when some columns are temporarily halted due to zero conditions.

B. Gate Level Simulation with SDF Annotation

As mentioned in Section 7, we have performed meticulous gate level simulations with 91% SDF annotations conforming

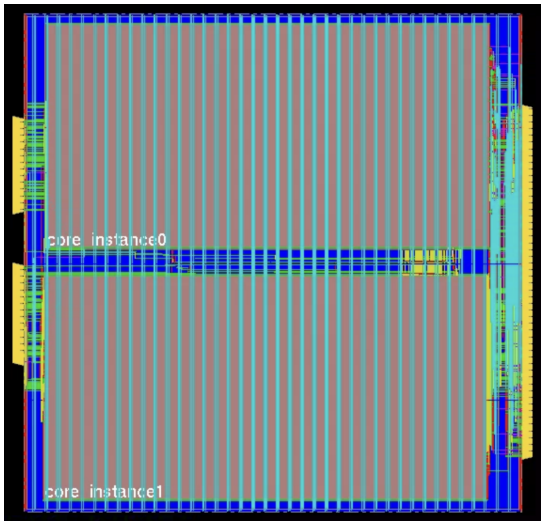

Fig. 14: Fullchip Hierarchical PNR with 2 Cores as sub-modules and Synchronization Logic between them

```
  VERIFY GEOMETRY ..... Sub-Area : 98 complete 0 Viols. 0 Wrngs.
  VERIFY GEOMETRY ..... SubArea : 99 of 100
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 99 complete 0 Viols. 0 Wrngs.
  VERIFY GEOMETRY ..... SubArea : 100 of 100
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 100 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 14.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:14.5  MEM: 22.7M)

innovus 10> $design
invalid command name "fullchip"
innovus 11>
```

Fig. 16: The final routing of the full chip was completed with **0 Geometry errors**

that the chip is prime for tape-out.

C. Creating a Multicycle Path

Division is a time-consuming step that requires us to develop an approach that allows us to send signals later than just a single cycle. Multicycle path refers to a path in a sequential circuit where the combinational logic delay between two flops can take more than one clock cycle to propagate the signal from source to destination. We have attempted to introduce six multicycle paths to enable the divider to function correctly in our project. This is for the hierarchical synthesis required for step 3. However, in further study, we discovered that introducing a pipelined divider significantly enhances performance and reduces the time for tools to synthesize the design. **Hence, our final optimized design does not need Multicycle Paths and can be easily and successfully synthesized, placed and routed at high frequencies.**

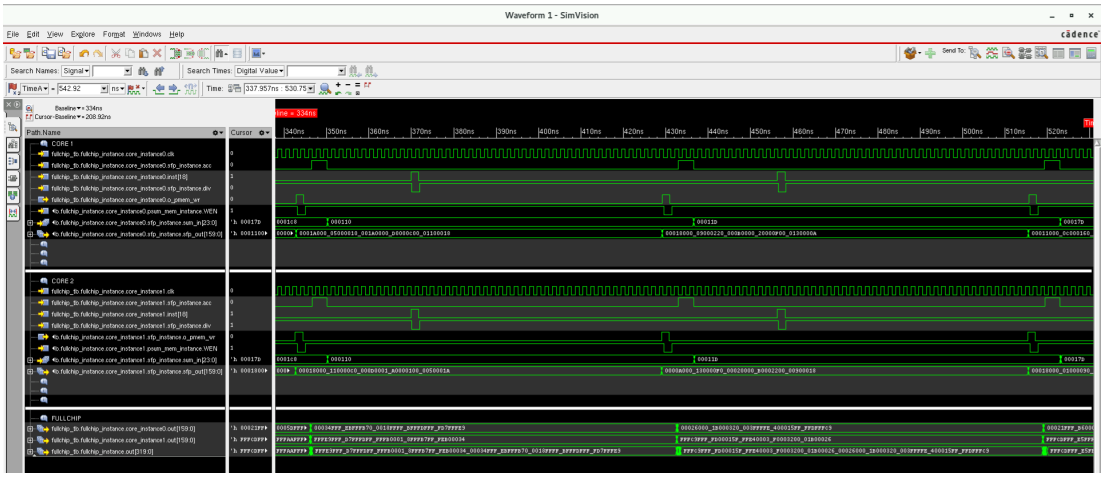

Fig. 17: Heirarchical Full Chip Gate Level Sim showing Accumulation and Pipelined Division with full Req-Ack handshaking coordination

Fig. 18: Reading out the Normalized Partial Sums from each core's PMEM to form one full concatenated output at the Fullchip level

```
Annotation completed with 0 Errors and 165174 Warnings
SDF statistics:
        No. of Pathdelays = 698298      No. of Disabled Pathdelays = 0        Annotated = 91.70% (640358/698298)
        No. of Tchecks    = 424474      No. of Disabled Tchecks    = 0        Annotated = 77.63% (329510/424474)
                                Total(T)          Disabled(D)     Annotated(A)          Percentage(A/(T-D))
        Path Delays             698298              0             640358                      91.70
              $hold              47482              0                  0                       0.00
             $width             174502              0             174502                     100.00
          $recovery              47482              0                  0                       0.00
         $setuphold             155008              0             155008                     100.00
```

Fig. 19: Successful SDF Annotation at all levels, Step 1 Core, Step 3 Core and Step 4/5 Fullchip

```
226 initial $sdf_annotate("./constraints/core_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance0, , ,"MAXIMUM","1:1:1","FROM_MTM");
227 initial $sdf_annotate("./constraints/core_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance1, , ,"MAXIMUM","1:1:1","FROM_MTM");
228
229 initial $sdf_annotate("./constraints/sram_w16_64_64bit_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance0.qmem_instance, , ,"MAXIMUM","1:1:1","FROM_MTM");
230 initial $sdf_annotate("./constraints/sram_w16_64_64bit_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance0.kmem_instance, , ,"MAXIMUM","1:1:1","FROM_MTM");
231 initial $sdf_annotate("./constraints/sram_w16_160_160bit_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance0.psum_mem_instance, , ,"MAXIMUM","1:1:1","FROM_MTM");
232
233 initial $sdf_annotate("./constraints/sram_w16_64_64bit_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance1.qmem_instance, , ,"MAXIMUM","1:1:1","FROM_MTM");
234 initial $sdf_annotate("./constraints/sram_w16_64_64bit_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance1.kmem_instance, , ,"MAXIMUM","1:1:1","FROM_MTM");
235 initial $sdf_annotate("./constraints/sram_w16_160_160bit_TC_08.sdf", fullchip_tb.fullchip_instance.core_instance1.psum_mem_instance, , ,"MAXIMUM","1:1:1","FROM_MTM");
236
```

Fig. 20: Annotation at all Heirarchical Module levels done from the Testbench

D. Asynchronous Interface

The cores can operate independently and asynchronously using an asynchronous interface, allowing them to perform MAC calculations at their own pace without being tightly coupled. This asynchronous nature enables each core to process data as soon as it becomes available, without waiting for a fixed clock cycle or synchronization signal. Additionally, an asynchronous interface can help mitigate timing mismatches and clock skew that may arise when cores operate at different speeds or have varying latencies. **In our final optimized design, the two cores can operate at wildly different frequencies like 1ns and 5ns and still coordinate the normalization process.**

E. Pipelining the MAC Unit

Performed pipelining with the boundary between the multiplication & accumulation operation to increase the throughput. We went one step ahead and broke the 7 additions into 2 pipeline stages. Control signals (instructions) were pipelined to avoid skew between the data and control paths. Multiple instructions can be processed simultaneously by pipelining the MAC unit, with each pipeline stage handling a different operation. This enables the system to perform multiple MAC calculations concurrently, significantly increasing the processing speed.

VIII. Conclusion

In this report we have presented novel approaches to optimizing a single core attention mechanism module and enhanced it to a dual core machine learning accelerator for the same. We have developed our individual cores so that they can be run on a clock of 1.3ns and a fullchip design which can run on 1.8ns with real world physical delays. These designs are also capable of running on clock frequencies over 1GHz with unannotated constant delays. With this we can successfully stake a claim on having designed and developed a chip which can be taped-out for the said specifications.

IX. Source Code

PFA paths to all source code, simulations, waveforms and routed designs on the next page.

Top Level Path: /home/linux/ieng6/ee260bwi24/		
Step	**Item**	**Path**
1: Single Core RTL and Verification	Verilog, Behavioural VCD	/k1mehta/project3/1d_systolic_array_phy_design/verilog_step1/ /k1mehta/project3/1d_systolic_array_phy_design/filelist_step1 /k1mehta/project3/1d_systolic_array_phy_design/vcd_files/step1_*
1: Single Core Synthesis and PNR	Routed design	/k1mehta/project3/1d_systolic_array_phy_design/pnr/step1_core_unoptimized_heirarchical/
1: Single Core Synthesis and PNR	Gate Level Sim, VCD	/kdivij/PROJECT/0_rtl_sim.core.step1/fullchip_tb.vcd
2. Output normalization	Verilog, Behavioural VCD	/k1mehta/project3/1d_systolic_array_phy_design/verilog_step2/ /k1mehta/project3/1d_systolic_array_phy_design/filelist_step2 /k1mehta/project3/1d_systolic_array_phy_design/vcd_files/step2_*
3. Single Core Synthesis and PNR (Unoptimized)	Routed Design	/k1mehta/project3/1d_systolic_array_phy_design/pnr/step3_single_core_unoptimized/
3. Single Core Synthesis and PNR (Optimized)	Routed Design	/k1mehta/project3/1d_systolic_array_phy_design/pnr/step3_single_core_optimized/
3. Single Core Synthesis and PNR (Optimized)	Gate Level Sim, VCD	/kdivij/PROJECT/0_rtl_sim.core_step3/fullchip_tb.vcd
4. Dual Core Synthesis and PNR 5. Optimizations	Verilog, Behavioural VCD	/k1mehta/project3/1d_systolic_array_phy_design/verilog_dualcore/ /k1mehta/project3/1d_systolic_array_phy_design/filelist_dualcore/ /k1mehta/project3/1d_systolic_array_phy_design/vcd_files/dualcore_*
4. Dual Core Synthesis and PNR 5. Optimizations	Routed design	/k1mehta/project3/1d_systolic_array_phy_design/pnr/step4_step5_fullchip_optimized/
4. Dual Core Synthesis and PNR 5. Optimizations	Gate Level Sim, VCD	/kdivij/PROJECT/0_rtl_sim.fullchip_step4/fullchip_tb.vcd