FPGA assignment

Title:

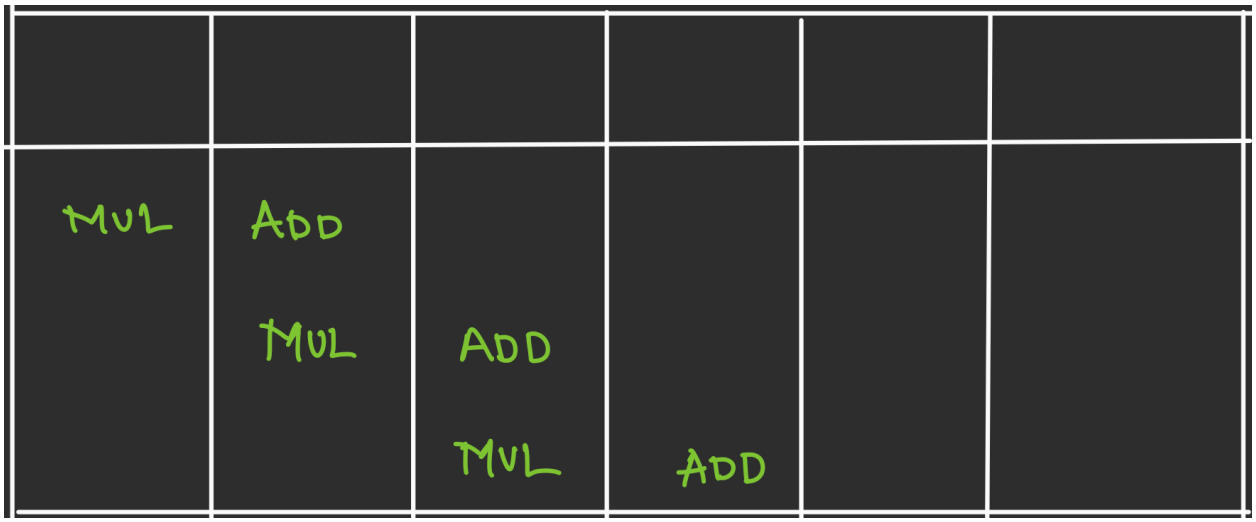
Team members : 1) Yash Sengupta(IMT2022532)
2) Teerth Bhalgat(IMT2022586)
3) Niranjan Gopal(IMT2022543)

Github Repo Link: https://github.com/YashSengupta19/FPGA_Project_Pipelined_MAC

Introduction

How we interpreted what are the things that needed to be pipelined?

The most trivial case of pipelining is doing the Multiply operation in one clock cycle and doing the addition operation in the next clock cycle.



However if we have many bits, then we can also pipeline the multiplication stage and even the accumulation stage.

- Multiplication Stage: To pipeline the multiplication stage, we need to propagate some information to the next stage:
  - **Inputs to calculate the partial products**
  - **Carry propagation from previous stage**
  - **Partial sums, which need to be added to partial products to get correct output.**
- Accumulation Stage: To pipeline the accumulation stage, we need to propagate the **carry bits** to the subsequent stages.

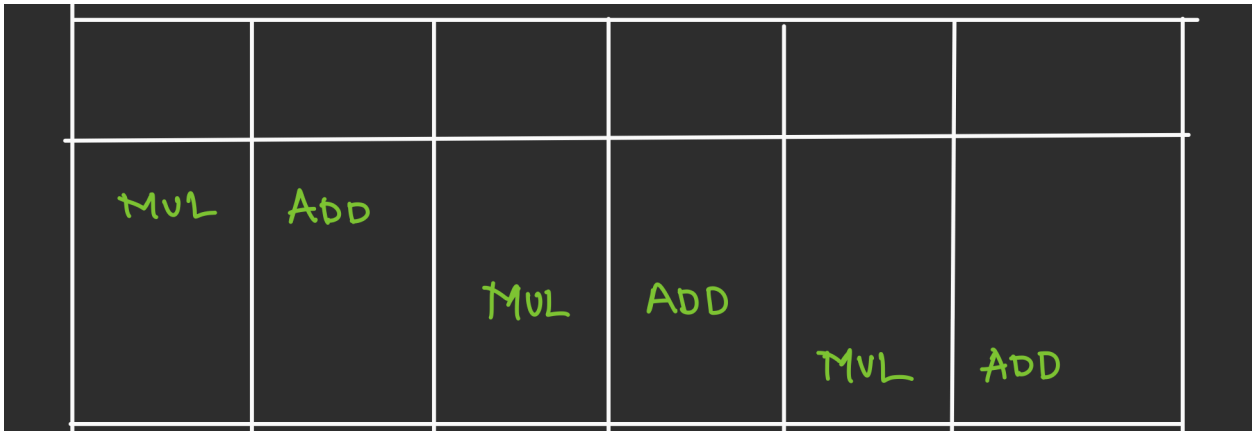
Our Interpretation of Pipeline and Non-pipelined multiplication:

$$
\begin{array}{cccc}
 & a_1 & a_0 & \\
\times & b_1 & b_0 & \\
\hline
 & a_1b_0 & a_0b_0 & \\
a_1b_1 & a_0b_1 & & \\
\hline
P_3 & P_2 & P_1 & P_0
\end{array}
$$

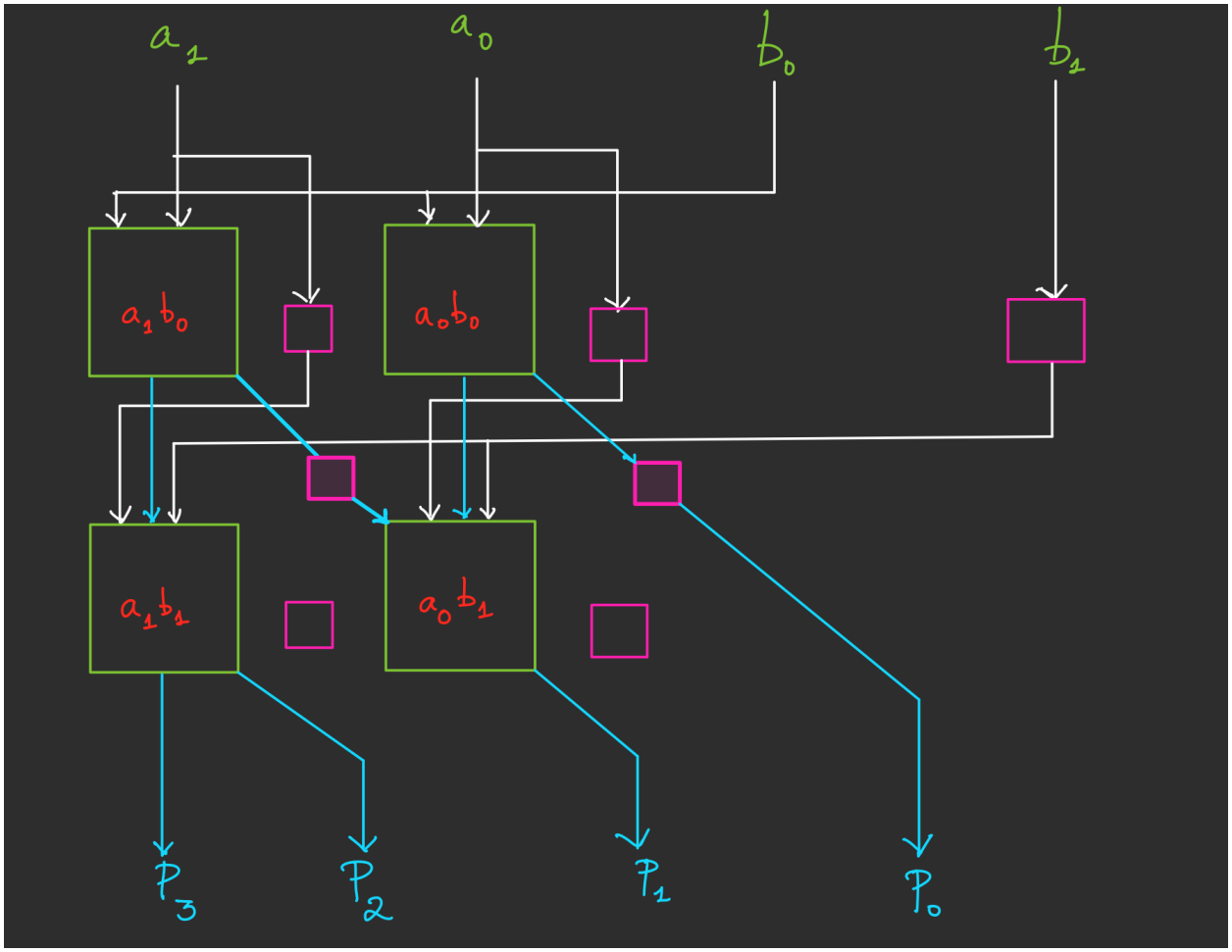
$$S_1 \left(a_1b_0 + a_0b_1\right)$$

$$S_2 \left(a_1b_1 + c_1\left(a_1b_0 + a_0b_1\right)\right)$$

$$c_2\left(a_1b_1 + c_1\left(a_1b_0 + a_0b_1\right)\right)$$



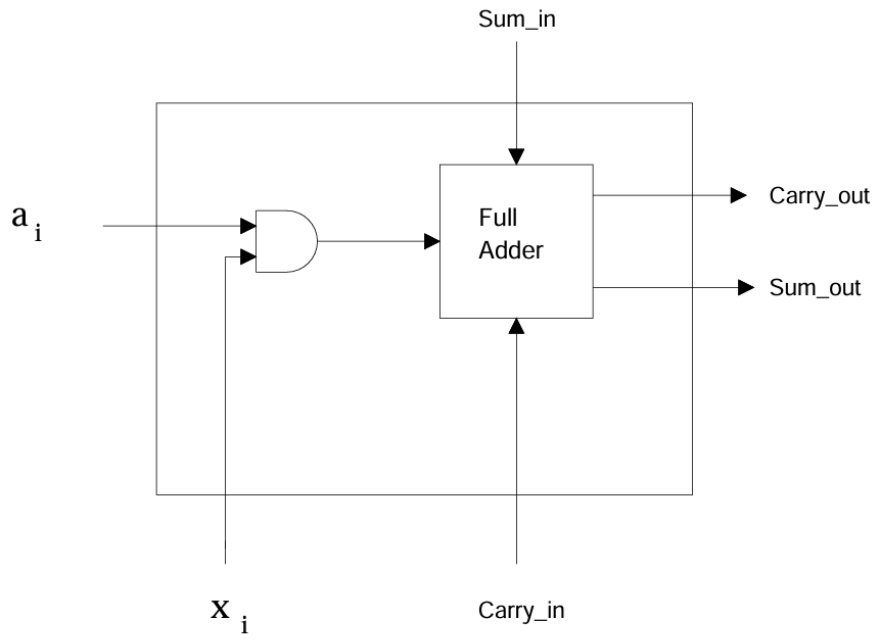
This above diagram represents how non-pipelined architecture works where after a MAC operation is done, then the next one starts.



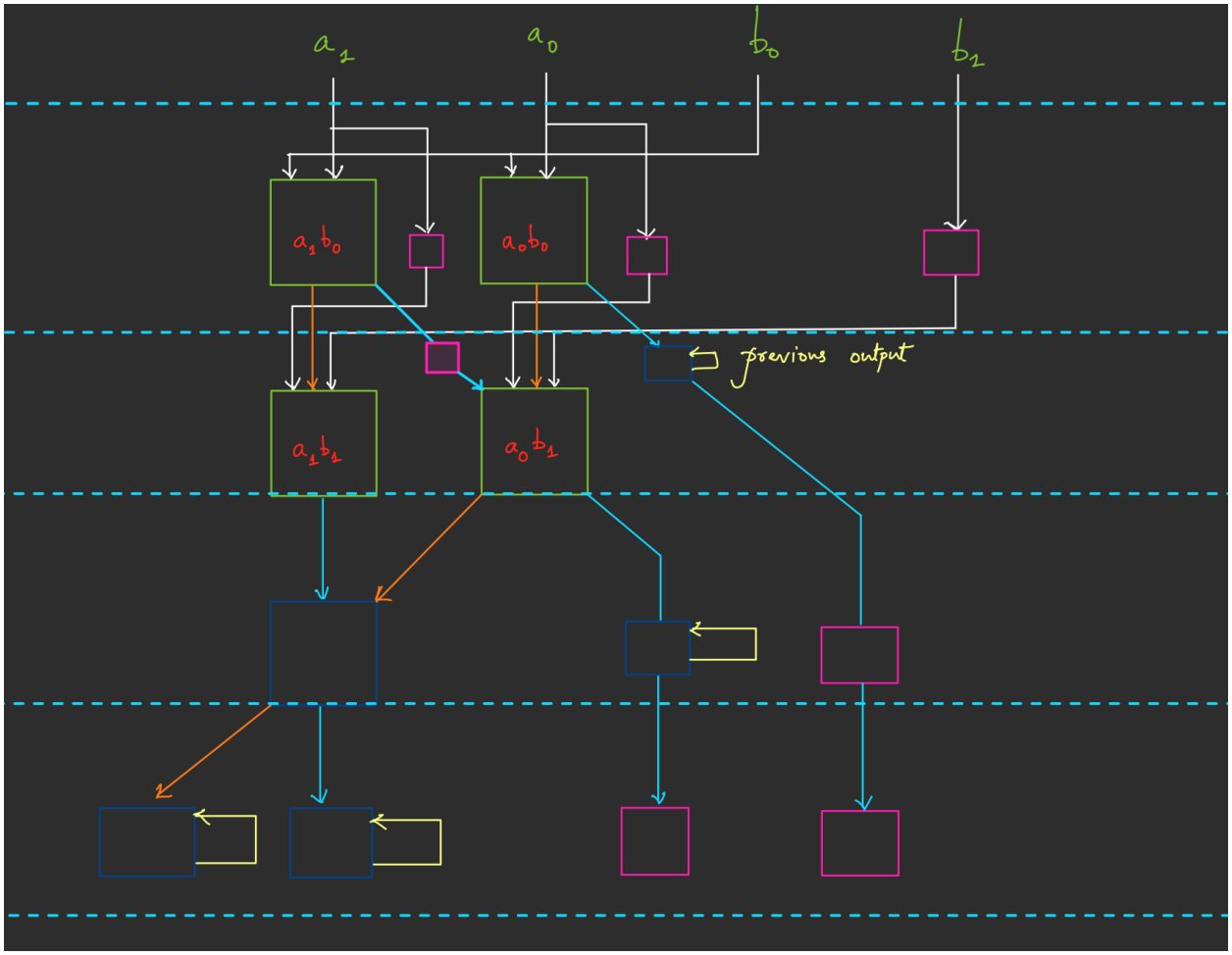
This diagram is our interpretation of how a pipelined multiplier is working so as to generate a 4 bit result from a 2 bit multiplication.

We have taken reference from the research paper “FPGA IMPLEMENTATION OF DIGITAL FILTERS”.

This is the basic cell :-



This cell used in the form of a systolic array helps in calculating the required partial products at each clock cycle and adding with the required carry bits and other partial products.



In this diagram, we are extending our logic of multiplier to MAC, where we are accumulating the bits as they are getting calculated and the carry is getting propagated onto the next stage.

DIFFICULTIES WE FACED IN THE PROJECT:

1. One of the main difficulties we faced in the project was to differentiate between the different pipeline stages, what to pipeline, what operations to perform in a single stage, etc. This was even more difficult when we used the research paper for reference trying to accumulate the result while trying to calculate the product simultaneously.

   So what we did was we broke down the multiply and accumulate

as two separate steps. First we tried to understand and implement pipelined multipliers only. Once we were able to do that we referred to the paper again and saw how it was accumulating the results and then implementing the results.

2. The second place where we got stuck was the verilog implementation. At each stage some outputs were not shown or considered and then some other outputs need to be passed on to the pipeline registers. In the paper it's not clearly mentioned which output is going to which next block or which things are we storing in pipelined registers. So at one point it got very confusing.

   So what we did regarding that started with a simple 2x2 MAC, understood its working and then moved forward with the larger design. Once we drew the entire architecture by hand and understood how each part was propagating through the stages, we were able to implement it in Verilog.

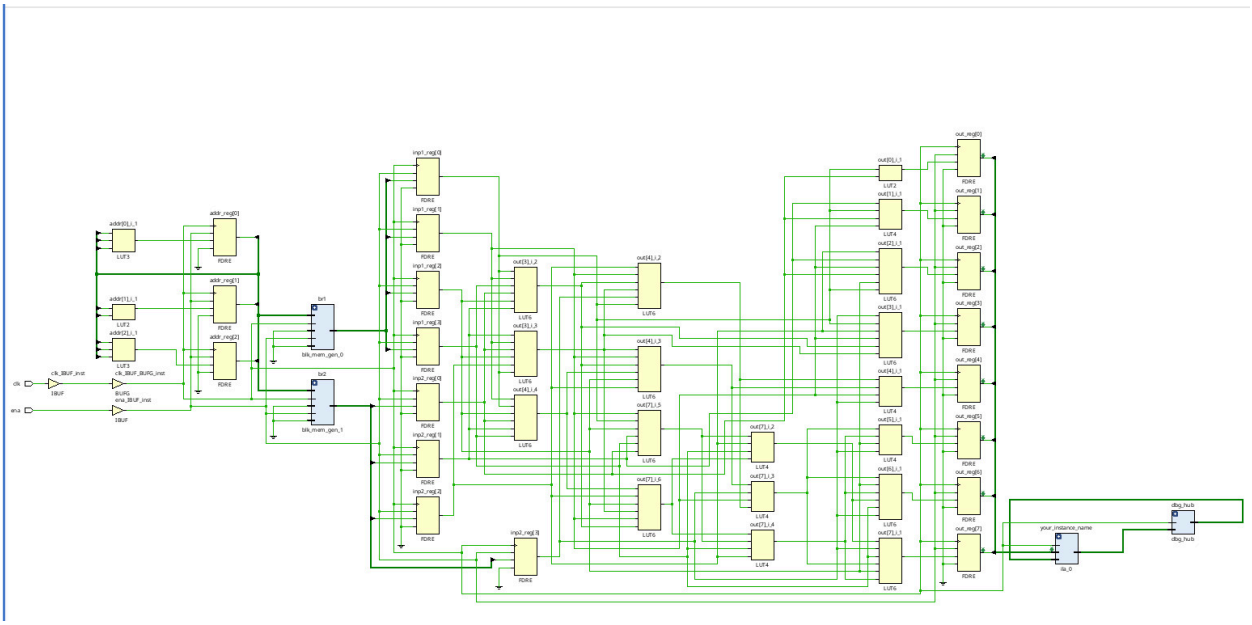
3. Another place where we faced difficulty towards the end was verifying the fact that the implemented design was actually faster than the non-pipelined case. While we thought looking at the ILA and observing results at each clock cycle would give us some insights regarding this, we realized that since both the designs are running on the same clock, we cannot really compare them.

   So what we have done is we have compared the theoretical clock frequency of both the designs using the Worst Negative Slack.
   We figured that the Max_clk_frequency_non_pipelined was less than that of the Max_clk_frequency_pipelined one. This means

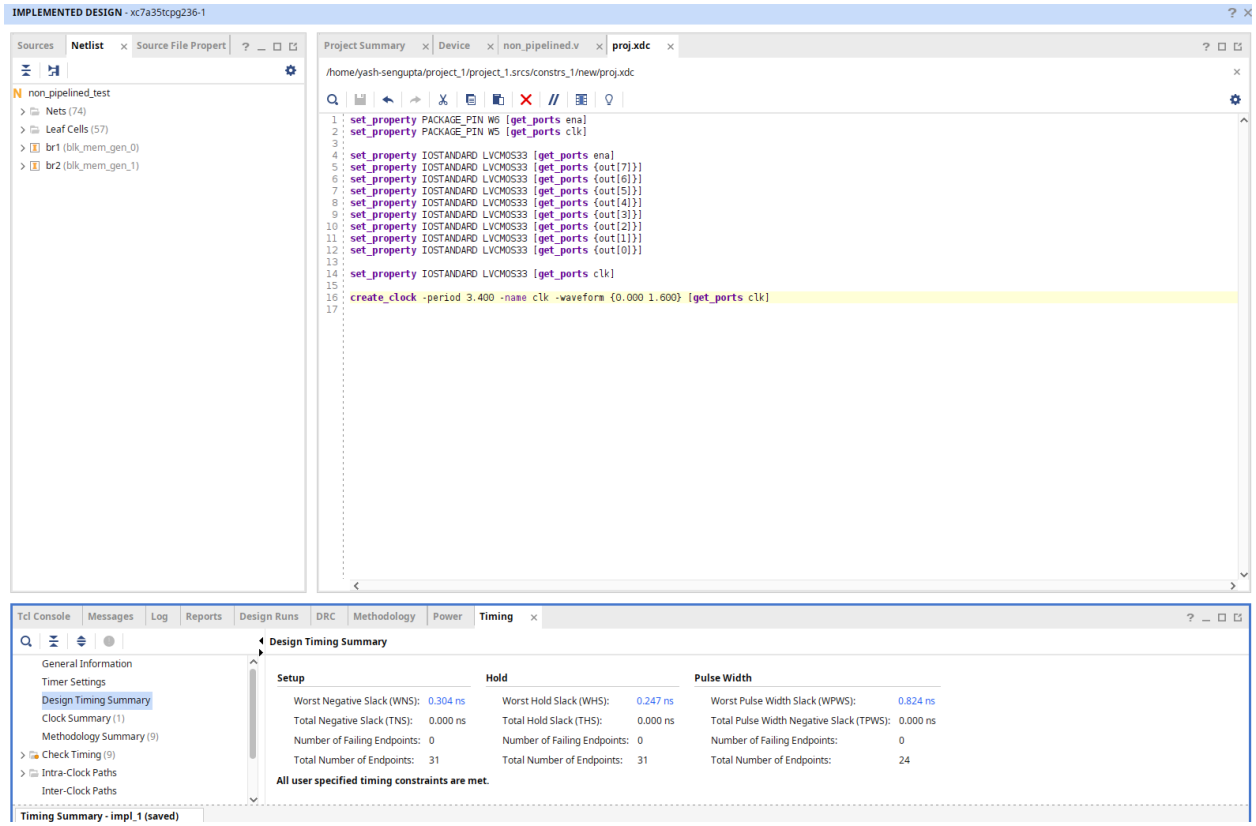
that the non-pipelined design takes more time to produce an output as compared to the pipelined one.

i) Non-Pipelined MAC

Schematic:



Timing summary:



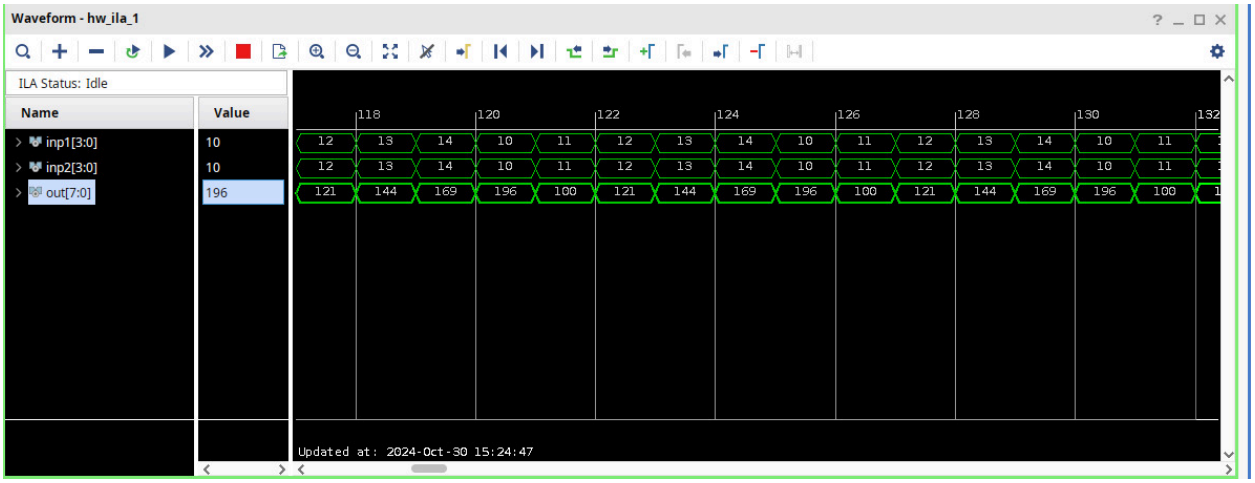
Clock Frequency Calculation:

Time period: 3.4ns
Worst Negative Slack: 0.304ns
Clock frequency: 1 / (3.4 - 0.304) = 3.22 x 10^8 Hz

Report utilization:

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
non_pipelined_test	19	19	10	19	1	10	1
br1 (blk_mem_gen_0)	0	0	0	0	0.5	0	0
U0 (blk_mem_gen_0_	0	0	0	0	0.5	0	0
br2 (blk_mem_gen_1)	0	0	0	0	0.5	0	0
U0 (blk_mem_gen_1_	0	0	0	0	0.5	0	0

ILA snapshot



ii) Exact pipelined MAC

Timing summary:

```
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN V17 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports reset]

create_clock -period 2.200 -name clk -waveform {0.000 1.100} [get_ports clk]
```

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.297 ns	Worst Hold Slack (WHS):	0.121 ns	Worst Pulse Widt	
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width	
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing	
Total Number of Endpoints:	68	Total Number of Endpoints:	68	Total Number of E	

Timing constraints are not met.

Clock Frequency Calculation:

Time period: 2.2ns
Worst Negative Slack: 0.27ns
Clock frequency: 1 / (2.2 - 0.297) = 5.25 x 10^8 Hz

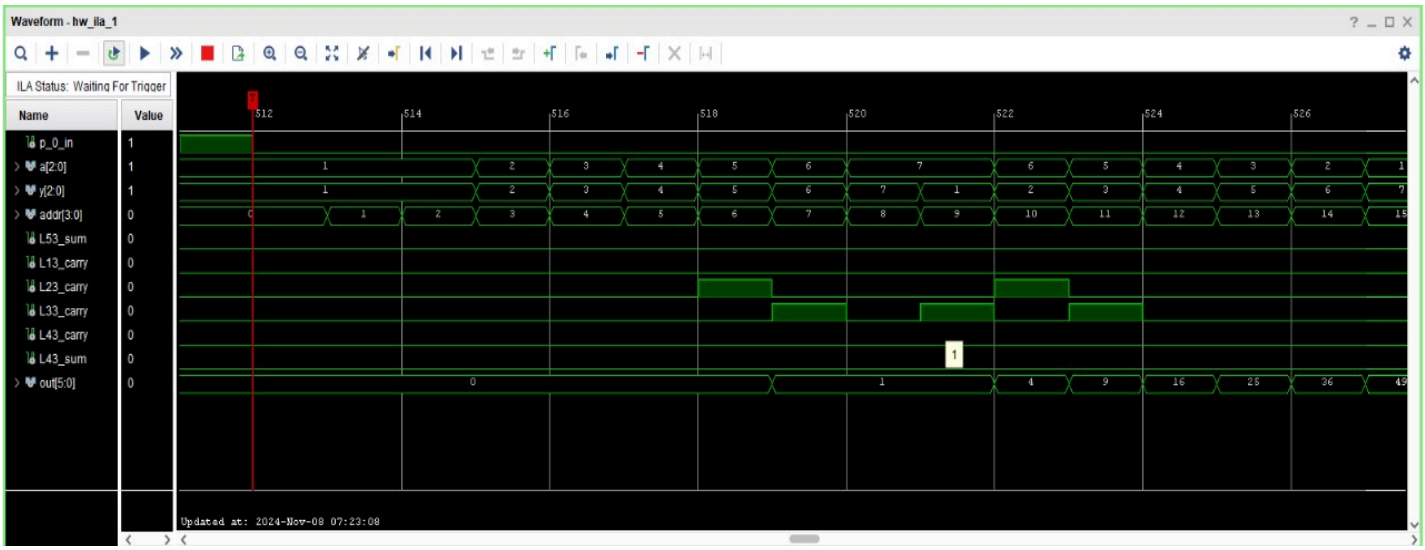
- So we can see that the clock frequency is more than in the case of a non-pipelined case as now we are performing many operations in parallel.

Report utilization:

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
pipelined_MAC	31	49	19	27	4	1	8	1
a_i (blk_mem_gen_0)	0	0	0	0	0	0.5	0	0
CAM_32 (combinational_array_multiplier)	2	0	2	2	0	0	0	0
CAM_33 (combinational_array_multiplier_0)	2	0	2	2	0	0	0	0
x_i (blk_mem_gen_1)	0	0	0	0	0	0.5	0	0

- As we can see from here the utilization of resources in pipelined is a lot more as compared to non-pipelined one which is obvious as in pipelined multiple instructions are running in parallel.

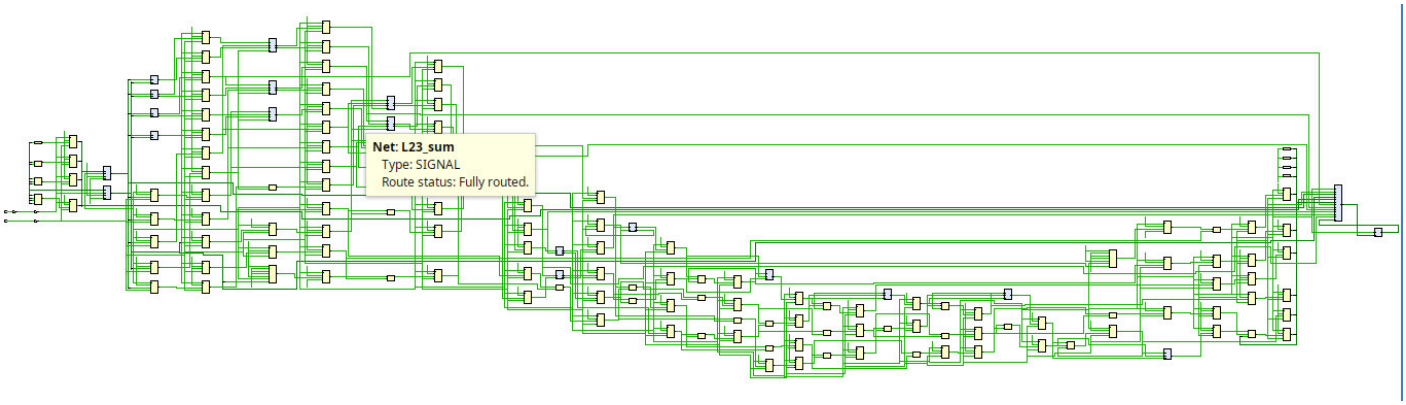
ILA snapshot:



- In this ILA diagram, if we are reading addresses at each clock cycle, we can assume one address is one clock.
- In pipelined structure, we can see that we are reading an address and the corresponding value and much later in some clock cycle we are getting the corresponding output. But then in the subsequent clock cycles, we are getting one output.

iii) Pipelined Approximate MAC

Schematic:



This is the pipelined MAC implementation as it has been specified in the research paper.

Timing summary:

Design Timing Summary

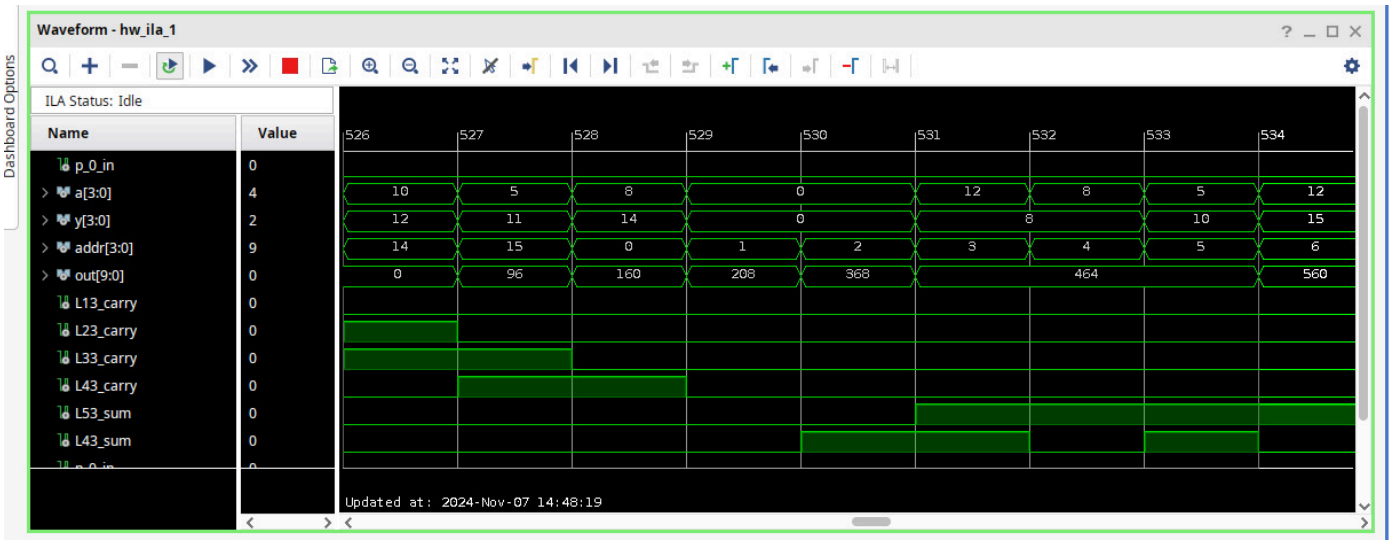
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	26.778 ns	Worst Hold Slack (WHS):	0.097 ns	Worst Pulse Width Slack (WPWS):	15.250 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1036	Total Number of Endpoints:	1028	Total Number of Endpoints:	483

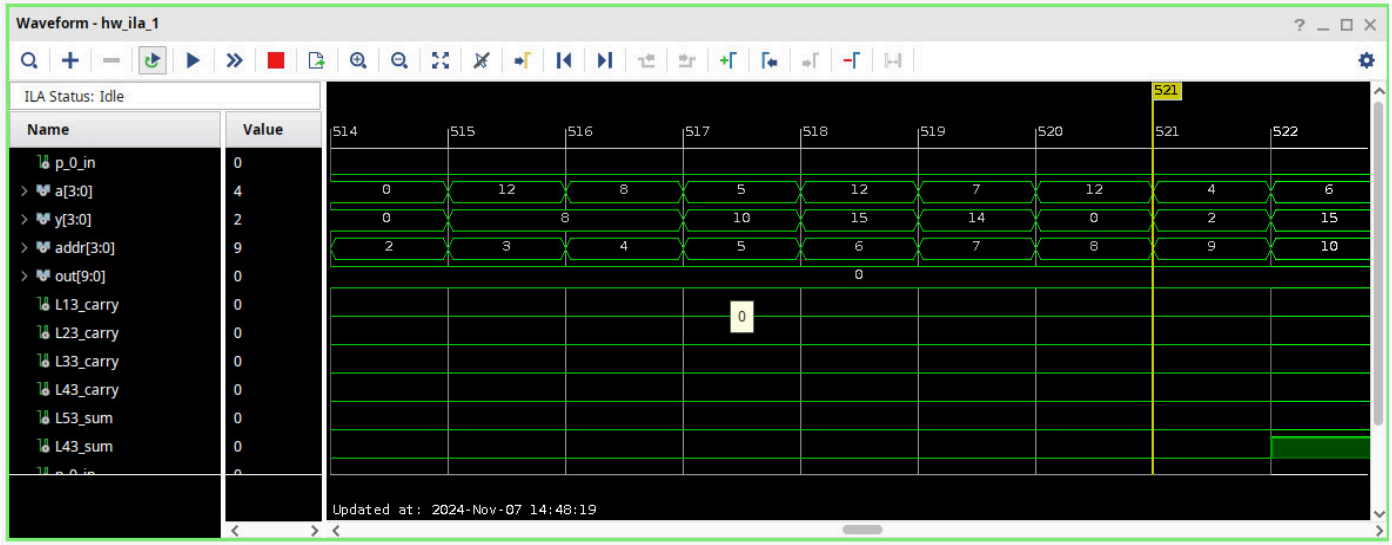
All user specified timing constraints are met.

Report utilization:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
pipelined_MAC	1565	2446	3	780	1405	160	2	2	2	1
a_i (blk_mem_gen_0)	0	0	0	0	0	0	0.5	0	0	0
CAM_11 (combinational_array_multiplier__10)	1	0	0	1	1	0	0	0	0	0
CAM_12 (combinational_array_multiplier__11)	1	0	0	1	1	0	0	0	0	0
CAM_13 (combinational_array_multiplier__12)	1	0	0	1	1	0	0	0	0	0
CAM_14 (combinational_array_multiplier__13)	0	0	0	0	0	0	0	0	0	0
CAM_22 (combinational_array_multiplier__16)	1	0	0	1	1	0	0	0	0	0
CAM_23 (combinational_array_multiplier__14)	1	0	0	1	1	0	0	0	0	0
CAM_24 (combinational_array_multiplier_0)	1	0	0	1	1	0	0	0	0	0
CAM_32 (combinational_array_multiplier_1)	1	0	0	1	1	0	0	0	0	0
CAM_33 (combinational_array_multiplier__15)	1	0	0	1	1	0	0	0	0	0
CAM_34 (combinational_array_multiplier_2)	1	0	0	1	1	0	0	0	0	0
CAM_42 (combinational_array_multiplier_3)	1	0	0	1	1	0	0	0	0	0
CAM_43 (combinational_array_multiplier)	1	0	0	1	1	0	0	0	0	0
CAM_44 (combinational_array_multiplier_4)	1	0	0	1	1	0	0	0	0	0
dbg_hub (dbg_hub)	449	741	0	235	425	24	0	0	1	1
FA71 (full_adder_5)	1	0	0	1	1	0	0	0	0	0
FA81 (full_adder_6)	1	0	0	1	1	0	0	0	0	0
FA91 (full_adder_7)	1	0	0	1	1	0	0	0	0	0
FA101 (full_adder)	1	0	0	1	1	0	0	0	0	0
HA52 (half_adder__11)	2	0	0	1	2	0	0	0	0	0
HA53 (half_adder__12)	2	0	0	1	2	0	0	0	0	0
HA64 (half_adder)	2	0	0	1	2	0	0	0	0	0
x_i (blk_mem_gen_1)	0	0	0	0	0	0	0.5	0	0	0
your_instance_name (ila_0)	1070	1606	3	529	936	134	1	0	0	0

ILA snapshots:


Waveform - hw_ila_1
Dashboard Options
ILA Status: Idle
Name
Value
p_0_in
a[3:0]
y[3:0]
addr[3:0]
out[9:0]
L13_carry
L23_carry
L33_carry
L43_carry
L53_sum
L43_sum
526
527
528
529
530
531
532
533
534
Updated at: 2024-Nov-07 14:48:19


Waveform - hw_ila_1
ILA Status: Idle
Name
Value
p_0_in
a[3:0]
y[3:0]
addr[3:0]
out[9:0]
L13_carry
L23_carry
L33_carry
L43_carry
L53_sum
L43_sum
514
515
516
517
518
519
520
521
522
Updated at: 2024-Nov-07 14:48:19