Development of slow and fast division algorithms for computer architecture

Intel Unnati Industrial Training – Summer 2023

Aaron Sonnie
Team: Hello World
Karunya Institute of Technology and Sciences

INTRODUCTION:

The “Hello World” Unnati internship. The internship's goal was to develop slow and fast division algorithms for computer architecture, with a focus on the Intel FPGA Cyclone V.

This internship report focuses on the development of slow and fast division algorithms for computer architecture. Division is a fundamental arithmetic operation in computing, but traditional algorithms can be inefficient, impacting computational speed and efficiency. The goal is to enhance computing capabilities and efficiency, inspiring further exploration in this crucial field of research.

Development Environment

The Intel Quartus Prime Lite platform was used to create the code. The hardware description language Verilog, which is frequently used in FPGA design, is used for the implementation.

Objectives

- To develop the Finite State machine (Mealy / Moore) based on the design specifications.
- To develop the Verilog HDL code for the developed FSM.
- To simulate and verify the design for its functionality with proper test cases.

Summary

- Week 1 - In addition to building the FSM and validating the test cases, we generated the block diagram for the given design and determined the appropriate inputs and outputs.
- Week 2 - We created the test bench, tested the specified FSM's functionality, generated the Verilog code, examined the code coverage reports, and confirmed the functionality.
- Week 3 - To accomplish adequate timing closure, we have synthesized the created HDL code, studied the area and power report, applied the appropriate timing restrictions, and examined the timing reports.
- Week 4 - We have finished the project report, published the code, functioning model, and video demo to the github site, and implemented the design on the selected FPGA.
- Goldschmidt Method - Fast Divison
- Long Division Method – Slow Division

CHAPTER - 2

GOLDSCHMIDT ALGORITHM

Algorithm

Step 1: Start with an initial approximation, denoted as "y0," for the reciprocal square root of the input number "x."

Step 2: Iterate the following steps until the desired level of accuracy is achieved:

a. Compute the reciprocal of the current approximation: "r = 1/y."

b. Multiply the reciprocal by the input number: "p = r * x."

c. Compute the arithmetic average of the reciprocal and the product: "y = (r + p) / 2."

d. Use the updated value of "y" as the new approximation for the next iteration.

Step 3: Repeat the iterations until the desired level of accuracy or a specific number of iterations is reached.

Step 6: Check the results.

Example Operation:

Let's calculate the reciprocal square root of "x = 16" using the Goldschmidt algorithm.
Start with an initial approximation, let's say "y0 = 0.25" (any reasonable initial approximation can be chosen).
Iterate the steps until convergence or desired accuracy is achieved.

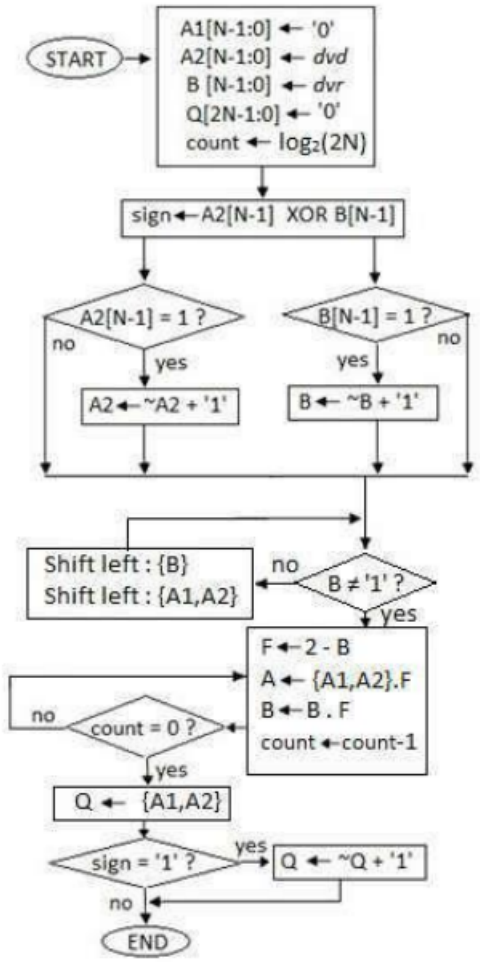
Iteration 1:
Compute the reciprocal: "r = 1 / y0 = 1 / 0.25 = 4."
Multiply the reciprocal by the input number: "p = r * x = 4 * 16 = 64."
Compute the arithmetic average: "y = (r + p) / 2 = (4 + 64) / 2 = 34."

Iteration 2:
Compute the reciprocal: "r = 1 / y = 1 / 34 ≈ 0.02941."
Multiply the reciprocal by the input number: "p = r * x ≈ 0.02941 * 16 ≈ 0.4706."
Compute the arithmetic average: "y = (r + p) / 2 ≈ (0.02941 + 0.4706) / 2 ≈ 0.25001."

Iteration 3 (optional):
Repeat the above steps with the updated approximation until the desired accuracy is achieved.
If necessary, perform additional iterations until the desired level of accuracy or convergence is achieved.
In this example, the Goldschmidt algorithm can be further iterated to improve accuracy if needed. The final approximation obtained after the desired level of accuracy or convergence is reached would be the reciprocal square root of the input number "x."

Flowchart

a) **Goldschmidt Division**



FSM

Since the Newton-Raphson division algorithm does not lend itself directly to a finite state machine (FSM) representation, it is not possible to create a traditional FSM diagram for it. The Newton-Raphson division algorithm is an iterative numerical method that relies on repeated calculations and comparisons until a desired level of precision is achieved.

Slow Division algorithm

Algorithm

Registers used: A, M, Q, n (counter)

Step 1: Write the dividend (number being divided) inside the long division symbol (÷) and the divisor (number dividing the dividend) outside the symbol.

Step 2: Begin with the leftmost digit of the dividend and perform the following steps:

a. Divide the current digit of the dividend by the divisor.

b. Write the quotient above the division symbol.

c. Multiply the quotient by the divisor and write the result below the dividend.

d. Subtract the result from the current dividend.

e. Bring down the next digit of the dividend next to the remainder obtained from the previous step.

f. Repeat steps (a) to (e) until all digits of the dividend have been processed or until the desired level of precision has been achieved.

Step 3: The final quotient written above the division symbol is the result of the division. The remainder, if any, is the value left over after dividing.

Example Operation:

Let's divide 158 by 7 using the long division algorithm.Count n = 4

```
    22
-----------
7 | 158
  - 14
  -----
    18
  - 14
  -----
     4
```

Write the dividend (158) inside the long division symbol and the divisor (7) outside the symbol.

Start with the leftmost digit (1) of the dividend: Divide 1 by 7, resulting in a quotient of 0. Write 0 above the division Symbol. Multiply the quotient (0) by the divisor (7), which equals 0. Write 0 below the 1 of the dividend. Subtract 0 from 1, leaving 1 as the remainder. Bring down the next digit (5) of the dividend next to the remainder. Repeat the steps

for the next digit (5): Divide 15 by 7, resulting in a quotient of 2. Write 2 above the division symbol. Multiply the quotient (2) by the divisor (7), which equals 14. Write 14 below the 15 of the dividend. Subtract 14 from 15, leaving 1 as the remainder. Bring down the next digit (8) of the dividend next to the remainder. Repeat the steps for the next digit (8):

Divide 18 by 7, resulting in a quotient of 2. Write 2 above the division symbol. Multiply the quotient (2) by the divisor (7), which equals 14. Write 14 below the 18 of the dividend. Subtract 14 from 18, leaving 4 as the remainder. Since all digits of the dividend have been processed, and no further digits are available, the long division is complete. The quotient is the sequence of digits written above the division symbol, which in this case is 22. The remainder is the final value left after division, which is 4. Therefore, when dividing 158 by 7 using the long division algorithm, the quotient is 22, and the remainder is 4.

Flowchart

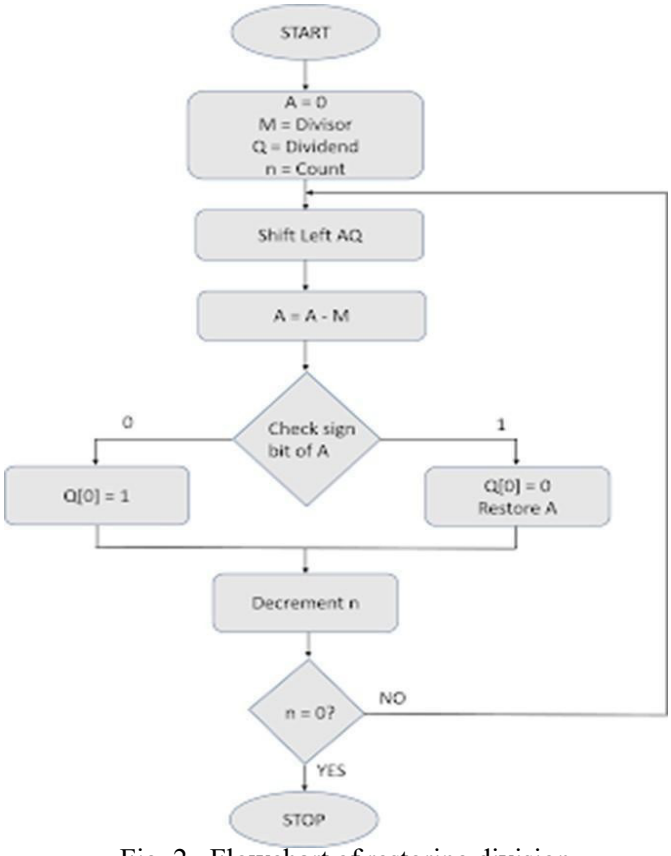


Fig. 2 - Flowchart of restoring division

FSM Diagram - MEALY MACHINE

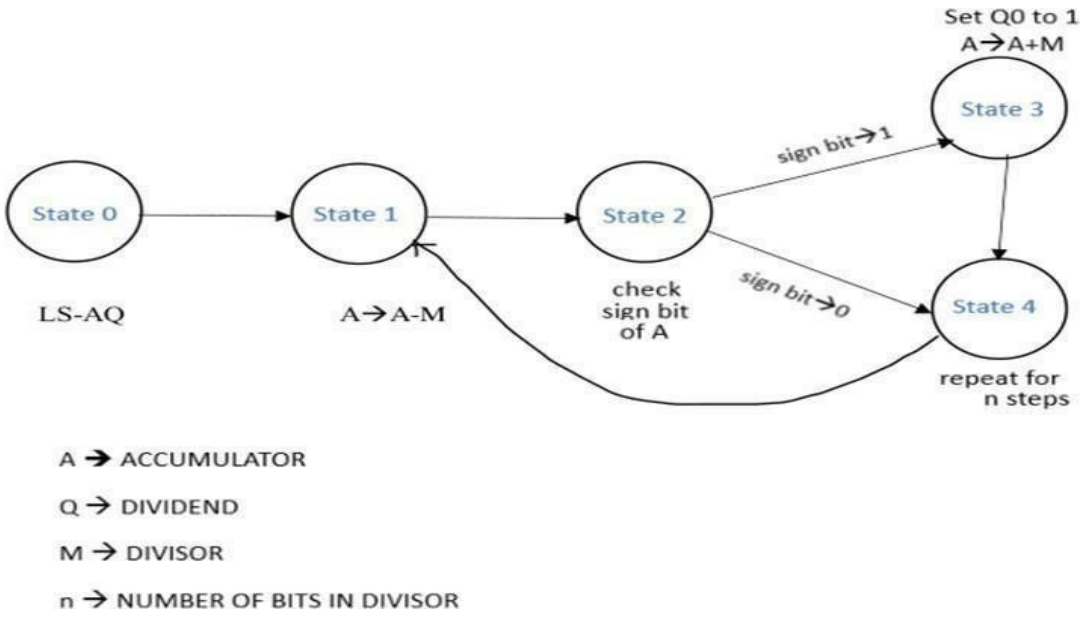


Fig. 3 - FSM of restoring division (mealy machine)

Analysis Diagram

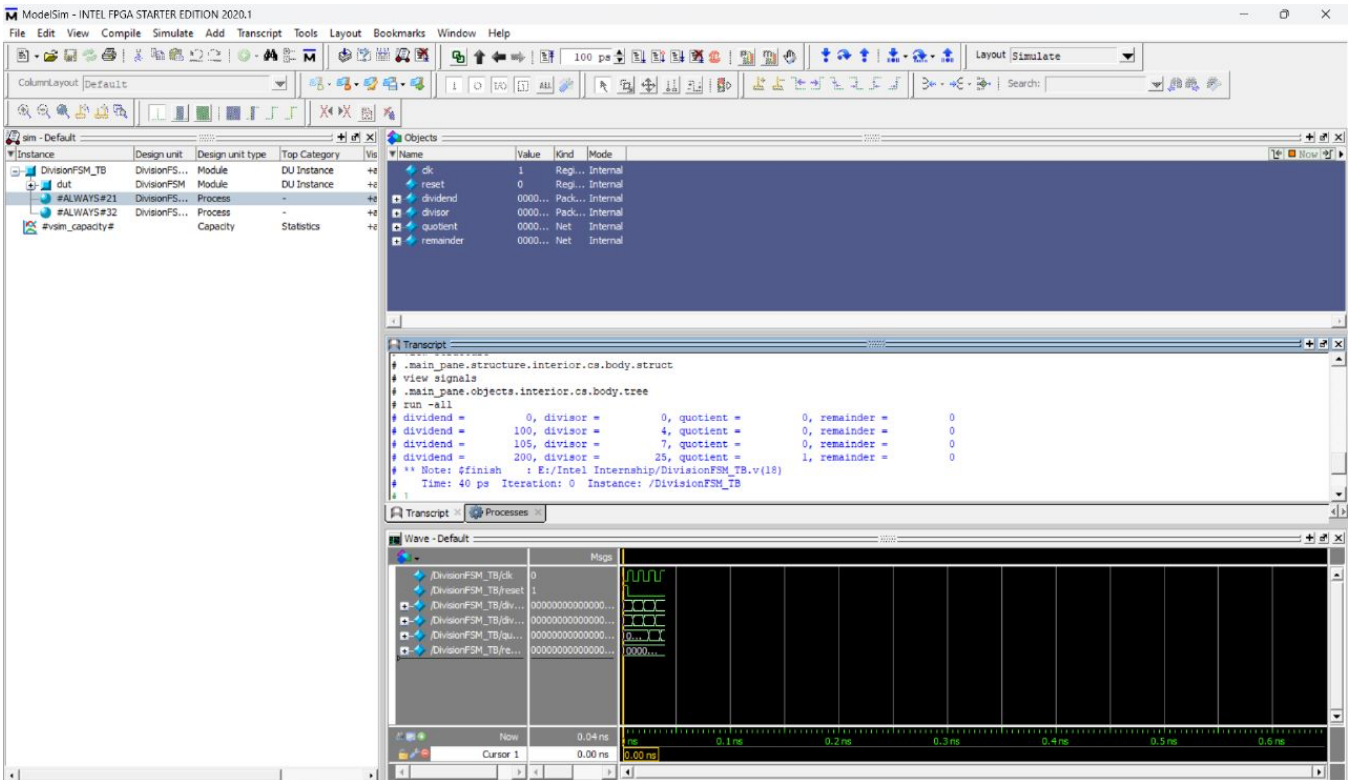


Fig. 4(a) - Analysis diagram of fast division

CHAPTER - 3

FPGA Implementation:

Fast Division Algorithm

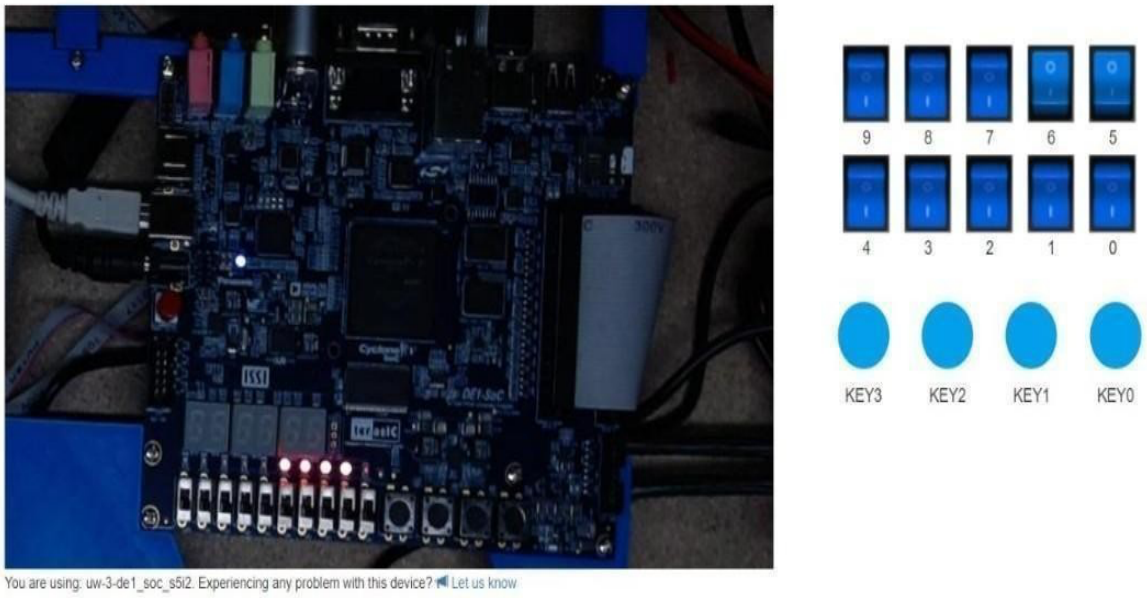


Fig. 5(a) - Labsland (fast division algorithm)

Slow Division Algorithm

Fig. 5(b) - Labsland (slow division algorithm)

Area report

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	95 / 32,070	< 1 %
2	ALMs needed [=A-B+C]	95	
1	[A] ALMs used in final placement [=a+b+c+d]	103 / 32,070	< 1 %
1	[a] ALMs used for LUT logic and registers	30	
2	[b] ALMs used for LUT logic	38	
3	[c] ALMs used for registers	35	
4	[d] ALMs used for memory (up to half of total ALMs)	0	
2	[B] Estimate of ALMs recoverable by dense packing	10 / 32,070	< 1 %
3	[C] Estimate of ALMs unavailable [=a+b+c+d]	2 / 32,070	< 1 %
1	[a] Due to location constrained logic	0	
2	[b] Due to LAB-wide signal conflicts	0	
3	[c] Due to LAB input limits	2	
4	[d] Due to virtual I/Os	0	
3			
4	Difficulty packing design	Low	
5			
6	Total LABs: partially or completely used	15 / 3,207	< 1 %
1	-- Logic LABs	15	
2	-- Memory LABs (up to half of total LABs)	0	
7			
8	Combinational ALUT usage for logic	125	
1	-- 7 input functions	0	
2	-- 6 input functions	35	
3	-- 5 input functions	2	
4	-- 4 input functions	8	
5	-- <=3 input functions	80	
9	Combinational ALUT usage for route-throughs	33	
10			

Fig. 6(a) - Area report of fast division and slow division algorithm

11	Dedicated logic registers	131	
1	-- By type:		
1	-- Primary logic registers	130 / 64,140	< 1 %
2	-- Secondary logic registers	1 / 64,140	< 1 %
2	-- By function:		
1	-- Design implementation registers	131	
2	-- Routing optimization registers	0	
12			
13	Virtual pins	0	
14	I/O pins	130 / 457	28 %
1	-- Clock pins	6 / 8	75 %
2	-- Dedicated input pins	0 / 21	0 %
15			
16	Hard processor system peripheral utilization		
1	-- Boot from FPGA	0 / 1 (0 %)	
2	-- Clock resets	0 / 1 (0 %)	
3	-- Cross trigger	0 / 1 (0 %)	
4	-- S2F AXI	0 / 1 (0 %)	
5	-- F2S AXI	0 / 1 (0 %)	
6	-- AXI Lightweight	0 / 1 (0 %)	
7	-- SDRAM	0 / 1 (0 %)	
8	-- Interrupts	0 / 1 (0 %)	
9	-- JTAG	0 / 1 (0 %)	
10	-- Loan I/O	0 / 1 (0 %)	
11	-- MPU event standby	0 / 1 (0 %)	
12	-- MPU general purpose	0 / 1 (0 %)	
13	-- STM event	0 / 1 (0 %)	
14	-- TPIU trace	0 / 1 (0 %)	

Fig. 6(b) - Area report of fast division and slow division algorithm

	Resource	Usage	%
15	-- DMA	0 / 1 (0 %)	
16	-- CAN	0 / 2 (0 %)	
17	-- EMAC	0 / 2 (0 %)	
18	-- I2C	0 / 4 (0 %)	
19	-- NAND Flash	0 / 1 (0 %)	
20	-- QSPI	0 / 1 (0 %)	
21	-- SDMMC	0 / 1 (0 %)	
22	-- SPI Master	0 / 2 (0 %)	
23	-- SPI Slave	0 / 2 (0 %)	
24	-- UART	0 / 2 (0 %)	
25	-- USB	0 / 2 (0 %)	
17			
18	M10K blocks	0 / 397	0 %
19	Total MLAB memory bits	0	
20	Total block memory bits	0 / 4,065,280	0 %
21	Total block memory implementation bits	0 / 4,065,280	0 %
22			
23	Total DSP Blocks	0 / 87	0 %
24			
25	Fractional PLLs	0 / 6	0 %
26	Global signals	2	
1	-- Global clocks	2 / 16	13 %
2	-- Quadrant clocks	0 / 66	0 %
3	-- Horizontal periphery clocks	0 / 18	0 %
27	SERDES Transmitters	0 / 100	0 %
28	SERDES Receivers	0 / 100	0 %
29	JTAGs	0 / 1	0 %
30	ASMI blocks	0 / 1	0 %

Fig. 6(c) - Area report of fast division and slow division algorithm

31	CRC blocks	0 / 1	0 %
32	Remote update blocks	0 / 1	0 %
33	Oscillator blocks	0 / 1	0 %
34	Impedance control blocks	0 / 4	0 %
35	Hard Memory Controllers	0 / 2	0 %
36	Average interconnect usage (total/H/V)	0.2% / 0.2% / 0.2%	
37	Peak interconnect usage (total/H/V)	3.5% / 3.9% / 2.6%	
38	Maximum fan-out	131	
39	Highest non-global fan-out	100	
40	Total fan-out	1318	
41	Average fan-out	2.39	

Fig. 6(d) - Area report of fast division algorithm

Power report

```
Type   ID      Message
  i 334003 Started post-fitting delay annotation
  i 334004 Delay annotation completed successfully
  i 215049 Average toggle rate for this design is 0.000 millions of transitions / sec
  i 215031 Total thermal power estimate for the design is 424.44 mW
> i        Quartus Prime Power Analyzer was successful. 0 errors, 6 warnings
```

Fig. 7- Power report of fast division algorithm

Pin planning

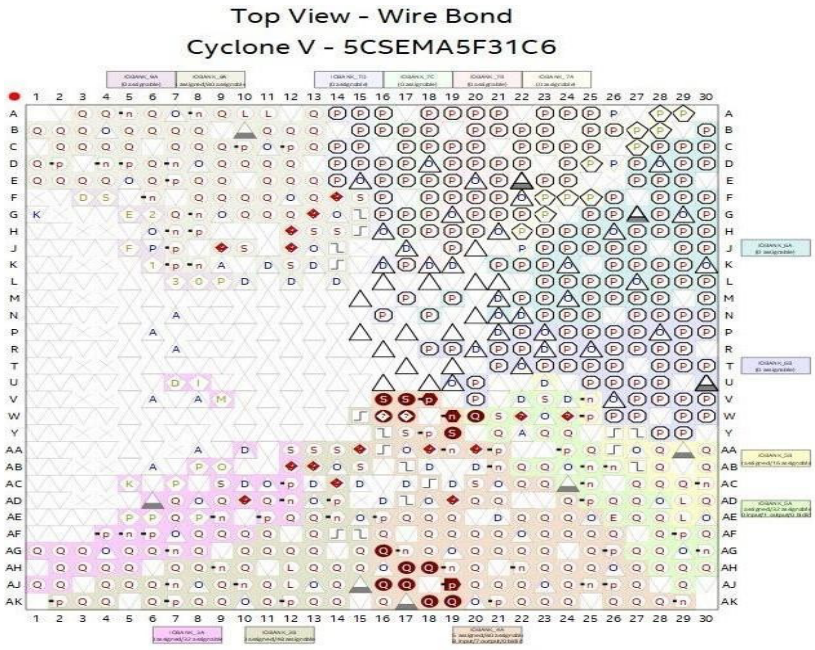


Fig. 8(a) - Pin planning of fast division algorithm

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	
dividend[3]	Input	PIN_AK18	4A	B4A_N0	PIN_AK18	3.3-V LVTTL		16mA (default)	
dividend[2]	Input	PIN_AK19	4A	B4A_N0	PIN_AK19	3.3-V LVTTL		16mA (default)	
dividend[1]	Input	PIN_AJ19	4A	B4A_N0	PIN_AJ19	3.3-V LVTTL		16mA (default)	
dividend[0]	Input	PIN_AJ17	4A	B4A_N0	PIN_AJ17	3.3-V LVTTL		16mA (default)	
divisor[3]	Input	PIN_AJ16	4A	B4A_N0	PIN_AJ16	3.3-V LVTTL		16mA (default)	
divisor[2]	Input	PIN_AH18	4A	B4A_N0	PIN_AH18	3.3-V LVTTL		16mA (default)	
divisor[1]	Input	PIN_AH17	4A	B4A_N0	PIN_AH17	3.3-V LVTTL		16mA (default)	
divisor[0]	Input	PIN_AG16	4A	B4A_N0	PIN_AG16	3.3-V LVTTL		16mA (default)	
quotient[3]	Output	PIN_V16	4A	B4A_N0	PIN_V16	3.3-V LVTTL		16mA (default)	1
quotient[2]	Output	PIN_W16	4A	B4A_N0	PIN_W16	3.3-V LVTTL		16mA (default)	1
quotient[1]	Output	PIN_V17	4A	B4A_N0	PIN_V17	3.3-V LVTTL		16mA (default)	1
quotient[0]	Output	PIN_V18	4A	B4A_N0	PIN_V18	3.3-V LVTTL		16mA (default)	1
remainder[3]	Output	PIN_W17	4A	B4A_N0	PIN_W17	3.3-V LVTTL		16mA (default)	1
remainder[2]	Output	PIN_W19	4A	B4A_N0	PIN_W19	3.3-V LVTTL		16mA (default)	1
remainder[1]	Output	PIN_Y19	4A	B4A_N0	PIN_Y19	3.3-V LVTTL		16mA (default)	1
remainder[0]	Output	PIN_W20	5A	B5A_N0	PIN_W20	3.3-V LVTTL		16mA (default)	1
<<new node>>									

Fig. 8(b) - Pin planning of fast division algorithm

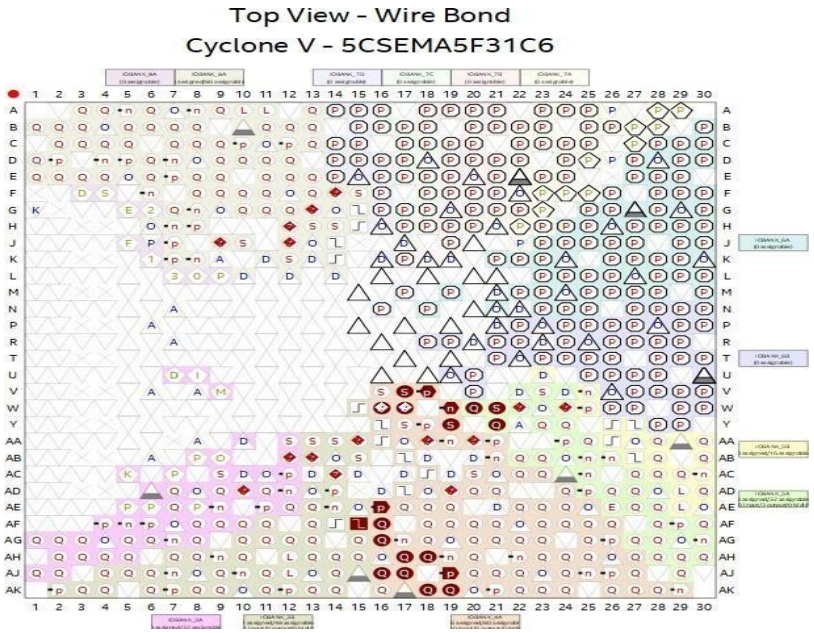


Fig. 8(c) - Pin planning of slow division algorithm

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
X[3]	Input	PIN_AK18	4A	B4A_N0	PIN_AK18	3.3-V LVTTL		16mA (default)	
X[2]	Input	PIN_AK19	4A	B4A_N0	PIN_AK19	3.3-V LVTTL		16mA (default)	
X[1]	Input	PIN_AJ19	4A	B4A_N0	PIN_AJ19	3.3-V LVTTL		16mA (default)	
X[0]	Input	PIN_AJ17	4A	B4A_N0	PIN_AJ17	3.3-V LVTTL		16mA (default)	
Y[3]	Input	PIN_AJ16	4A	B4A_N0	PIN_AJ16	3.3-V LVTTL		16mA (default)	
Y[2]	Input	PIN_AH18	4A	B4A_N0	PIN_AH18	3.3-V LVTTL		16mA (default)	
Y[1]	Input	PIN_AH17	4A	B4A_N0	PIN_AH17	3.3-V LVTTL		16mA (default)	
Y[0]	Input	PIN_AG16	4A	B4A_N0	PIN_AG16	3.3-V LVTTL		16mA (default)	
clk	Input	PIN_AE16	4A	B4A_N0	PIN_AE16	3.3-V LVTTL		16mA (default)	
quot[3]	Output	PIN_W16	4A	B4A_N0	PIN_W16	3.3-V LVTTL		16mA (default)	1 (default)
quot[2]	Output	PIN_V17	4A	B4A_N0	PIN_V17	3.3-V LVTTL		16mA (default)	1 (default)
quot[1]	Output	PIN_V18	4A	B4A_N0	PIN_V18	3.3-V LVTTL		16mA (default)	1 (default)
quot[0]	Output	PIN_W17	4A	B4A_N0	PIN_W17	3.3-V LVTTL		16mA (default)	1 (default)
rem[3]	Output	PIN_W19	4A	B4A_N0	PIN_W19	3.3-V LVTTL		16mA (default)	1 (default)
rem[2]	Output	PIN_Y19	4A	B4A_N0	PIN_Y19	3.3-V LVTTL		16mA (default)	1 (default)
rem[1]	Output	PIN_W20	5A	B5A_N0	PIN_W20	3.3-V LVTTL		16mA (default)	1 (default)
rem[0]	Output	PIN_W21	5A	B5A_N0	PIN_W21	3.3-V LVTTL		16mA (default)	1 (default)
rst	Input	PIN_AF16	4A	B4A_N0	PIN_AF16	3.3-V LVTTL		16mA (default)	
start	Input	PIN_AF15	3B	B3B_N0	PIN_AF15	3.3-V LVTTL		16mA (default)	
valid	Output	PIN_Y21	5A	B5A_N0	PIN_Y21	3.3-V LVTTL		16mA (default)	1 (default)
<<new node>>									

Fig. 8(d) - Pin planning of slow division algorithm

CONCLUSION:

This internship report focuses on the development of slow and fast division algorithms for computer architecture. Slow division algorithms, like long division, provide a fundamental understanding of division but may be inefficient for complex calculations. Fast division algorithms, such as Newton-Raphson and Goldschmidt, offer significant speed improvements, addressing performance bottlenecks. The adaptability of these algorithms to different hardware architectures, leveraging parallelism and hardware acceleration, has potential for further optimization. The report emphasizes the importance of faster division algorithms in enhancing computational efficiency and resource utilization in modern computer systems. Future research can build on these insights to develop even faster and hardware-optimized division algorithms, driving advancements in computer architecture and technology as a whole..