Implementation of **Slow and Fast Division** algorithms using Computer Architecture

13/07/2023

Intel® Unnati

Intel-ectuals

Alhaarith Hakkim

Midhun Mathew

Neena Anna Mathew

Index

Team: Intel-Ectuals

Team Members:

1. Midhun Mathew
2. Alhaarith Hakkim
3. Neena Anna Mathew

Mentor : Er.Jyotish Chandran
Institution : Saintgits College of Engineering

Abstract

This report presents a comprehensive analysis and implementation of slow and fast division algorithms using computer architecture. Division, a fundamental arithmetic operation, plays a crucial role in various applications. Optimizing division performance is vital, and this report explores the theoretical foundations of both slow and fast division algorithms while providing practical implementations using computer architecture concepts. The slow division algorithm, such as the traditional restoring division algorithm, serves as a basic approach to division but suffers from high latency. In contrast, fast division algorithms, like the non-restoring division algorithm, aim to reduce division latency by incorporating parallelism and optimization techniques. This report discusses the intricacies of both algorithms and provides detailed implementations, considering the hardware aspects of computer architecture.

Introduction:

Division is a fundamental arithmetic operation that plays a crucial role in various applications, ranging from basic calculations to complex mathematical computations. Division algorithms are designed to efficiently and accurately divide one number by another. The performance of division algorithms is of great importance due to the frequency at which division operations are performed in many computational tasks.

The significance of division algorithms lies in their ability to optimize the division process, reducing latency and improving overall computational efficiency. Different division algorithms have been developed over the years, each with its trade-offs in terms of speed, complexity, and hardware requirements.

Efficient division algorithms are particularly crucial in scenarios where high-performance computing is required, such as scientific simulations, financial modeling, and real-time signal processing. By

minimizing the time taken to perform division operations, these algorithms contribute to faster processing, reduced energy consumption, and improved overall system performance.
The field of computer architecture plays a significant role in the design and implementation of efficient division algorithms. Hardware optimizations, parallelism, and algorithmic techniques are utilized to accelerate division operations and achieve better performance. Advancements in computer architecture, including the introduction of specialized instruction sets and dedicated hardware modules for division, have further enhanced the efficiency of division algorithms.
In summary, division algorithms have a profound impact on computational efficiency and performance. By continually improving these algorithms and leveraging computer architecture concepts, researchers and engineers can unlock faster and more efficient division operations, enabling a wide range of applications to benefit from improved performance and productivity.

Scope and Organization

The scope of this report is to provide a comprehensive analysis and implementation of slow and fast division algorithms using computer architecture. The report focuses on understanding the theoretical foundations of these algorithms, exploring their step-by-step division processes, and comparing their performance. Furthermore, the report delves into the relevance of computer architecture in division algorithms and discusses hardware implementation considerations.
The report is organized as follows:

1. **Introduction**: The introduction section provides a brief background on the significance of division algorithms and their role in various applications. It also outlines the objectives of the report, which include understanding slow and fast division algorithms, implementing them using computer architecture principles, and comparing their performance. Additionally, the organization of the report is presented to give readers an overview of the subsequent sections.
2. **Slow Division Algorithm**: In this section, the focus is on slow division algorithms. An overview of these algorithms is provided, with specific attention given to the restoring division algorithm. The restoring division algorithm is explained in detail, including its step-by-step division process. The section concludes with an analysis of the algorithm's performance and limitations.
3. **Fast Division Algorithm**: The fast division algorithm is introduced in this section, highlighting its advantages over slow division algorithms. The non-restoring division algorithm is specifically discussed, covering its overall approach, parallelism, and optimization techniques. A step-by-step division process using the non-restoring algorithm is presented, and a performance comparison with slow division algorithms is conducted.
4. **Implementation of Slow and Fast Division Algorithms**: Here, the practical implementation of slow and fast division algorithms is addressed. The selection of a programming language and platform is discussed, followed by the detailed implementation of the restoring division algorithm and the non-restoring division algorithm. A comparison between the two implementations is provided, evaluating their efficiency and effectiveness.

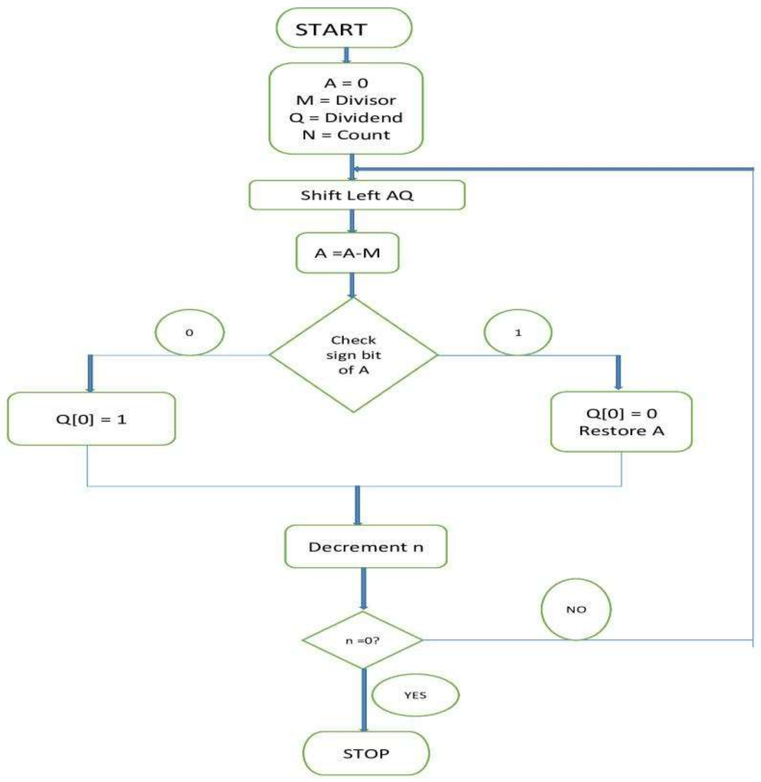
5. **Performance Analysis**: This section introduces metrics for evaluating division algorithms, such as latency, throughput, and resource utilization. The performance of both slow and fast division algorithms is analyzed using these metrics, providing insights into their relative strengths and weaknesses in terms of performance.
6. **Conclusion**: The conclusion section summarizes the findings and key takeaways from the report. It emphasizes the significance of fast division algorithms in improving performance and outlines potential avenues for future research and development in the field of division algorithms and computer architecture.
7. **References**: The report includes a list of sources consulted and cited throughout the document, ensuring proper acknowledgment of the referenced material.

Slow Division algorithm:

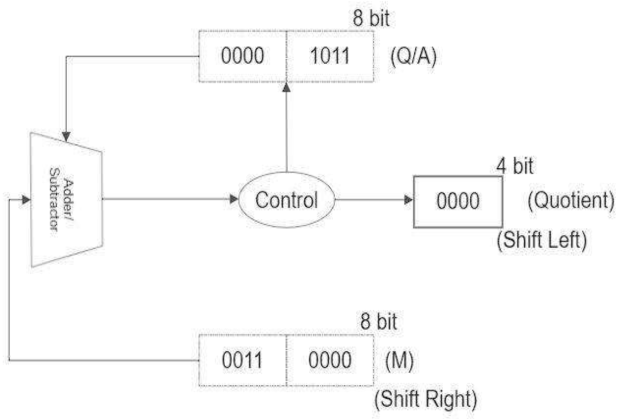
Restoring Division

Registers used: A, M, Q, n (counter)
Step 1: Load the initial values for the registers.
A = 0 (Accumulator), Qres = 0, M = Divisor, Q = Dividend and n is the count value which equals the number of bits of dividend.
Step 2: Shift left {A, Q}.
Step 3: Perform A = A - M.
Step 4: Check the sign bit of A. If 0, go to step 5. If 1, go to step 6.
Step 5: Set LSB of Q as 0. Goto step 7.
Step 6: Set LSB of Q as 1. Restore the value of A which was present before the subtraction.
Step 7: Decrement count.
Step 8: Check if counter value n is zero. If yes, go to next step. Else, go to step 3.
Step 9: Stop

Flowchart



Block Diagram

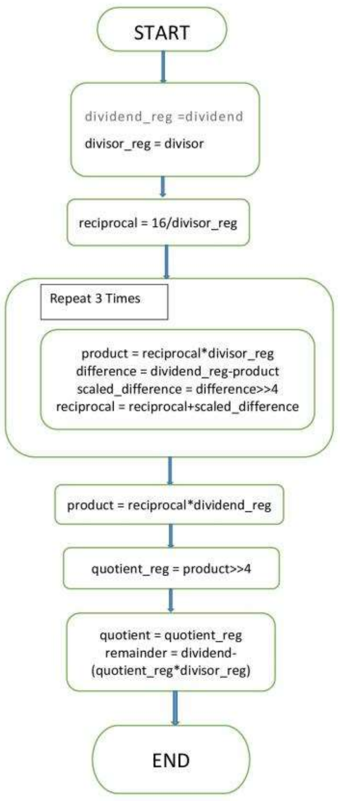


Fast Division algorithm:

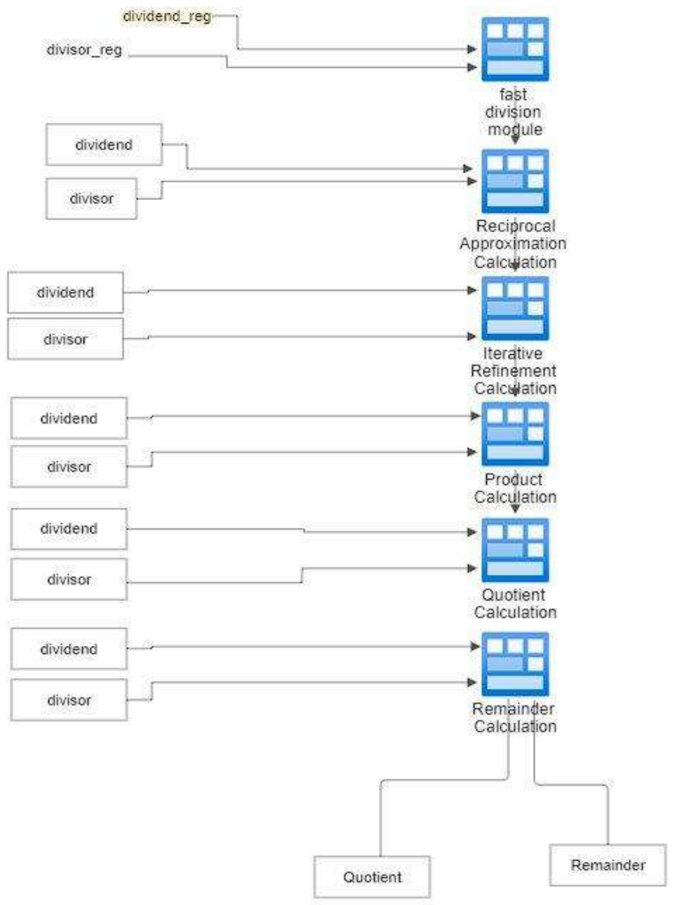
Non-Restoring Division

```
1. Set dividend_reg to the value of dividend
2. Set divisor_reg to the value of divisor
3. Set reciprocal to 16 divided by divisor_reg
4. Repeat 3 times:
     - Set product to reciprocal multiplied by divisor_reg
     - Set difference to dividend_reg minus product
     - Set scaled_difference to difference right-shifted by 4 bits
     - Set reciprocal to reciprocal plus scaled_difference
5. Set product to reciprocal multiplied by dividend_reg
6. Set quotient_reg to product right-shifted by 4 bits
7. Set quotient to quotient_reg
8. Set remainder to dividend minus (quotient_reg multiplied by divisor_reg)
```

Flowchart


START
dividend_reg =dividend
divisor_reg = divisor
reciprocal = 16/divisor_reg
Repeat 3 Times
product = reciprocal*divisor_reg
difference = dividend_reg-product
scaled_difference = difference>>4
reciprocal = reciprocal+scaled_difference
product = reciprocal*dividend_reg
quotient_reg = product>>4
quotient = quotient_reg
remainder = dividend-(quotient_reg*divisor_reg)
END

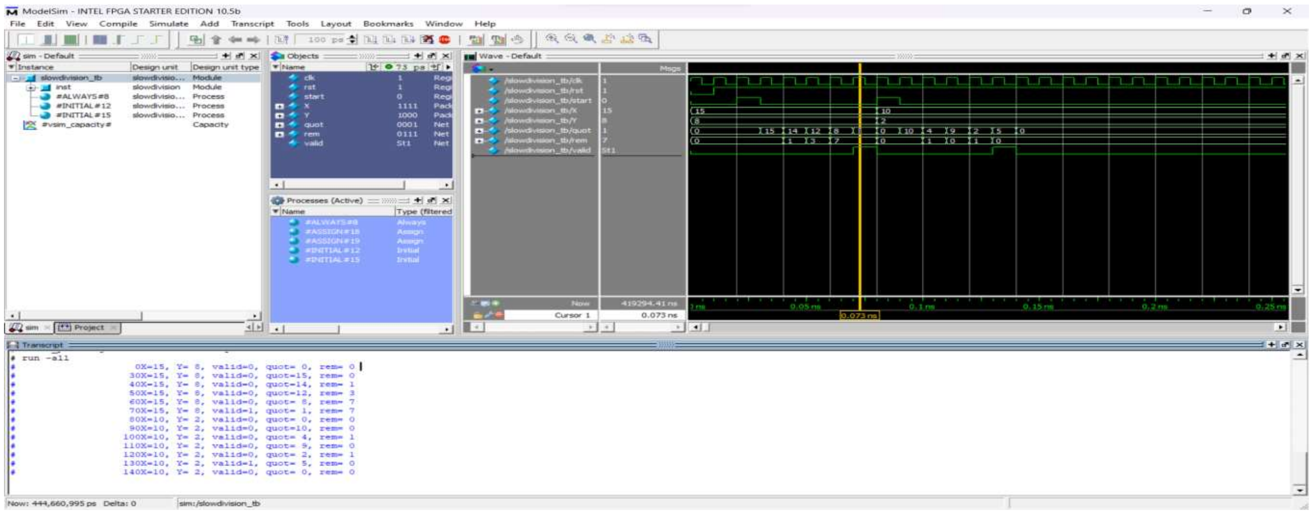
Block Diagram



Implementation of Fast and Slow Division of Algorithms

Synthesis (Slow Division)

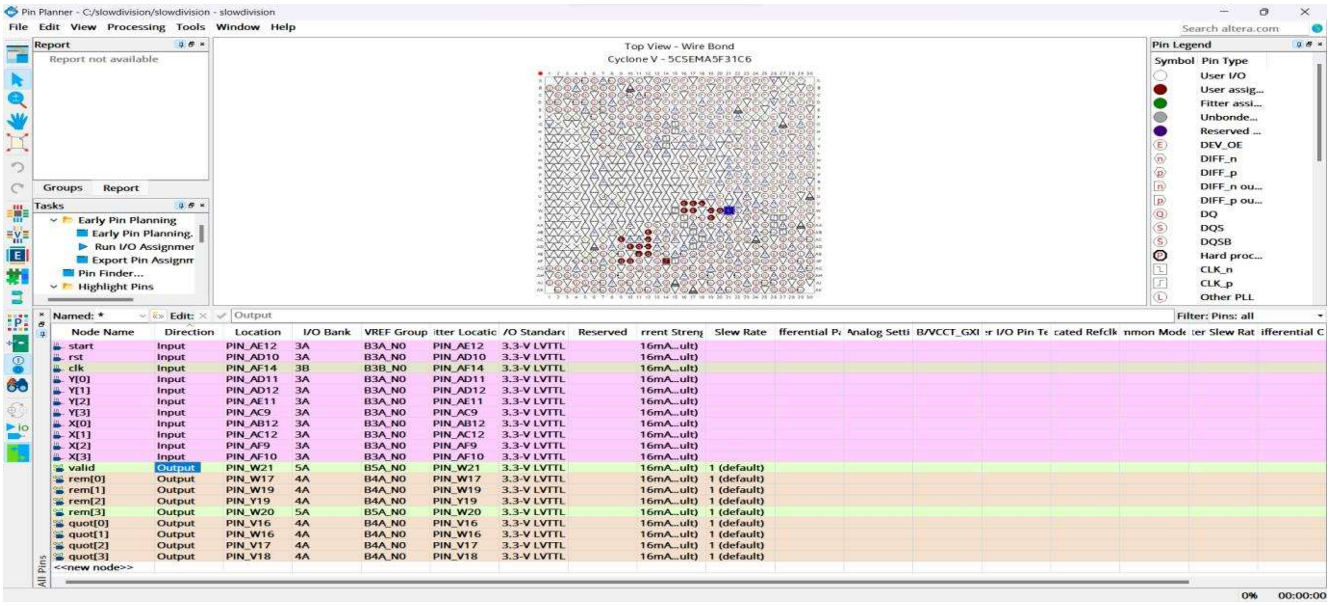
1. Functional Simulation



```
# run -all
#                    0X=15, Y= 8, valid=0, quot= 0, rem= 0
#                   30X=15, Y= 8, valid=0, quot=15, rem= 0
#                   40X=15, Y= 8, valid=0, quot=14, rem= 1
#                   50X=15, Y= 8, valid=0, quot=12, rem= 3
#                   60X=15, Y= 8, valid=0, quot= 8, rem= 7
#                   70X=15, Y= 8, valid=1, quot= 1, rem= 7
#                   80X=10, Y= 2, valid=0, quot= 0, rem= 0
#                   90X=10, Y= 2, valid=0, quot=10, rem= 0
#                  100X=10, Y= 2, valid=0, quot= 4, rem= 1
#                  110X=10, Y= 2, valid=0, quot= 9, rem= 0
#                  120X=10, Y= 2, valid=0, quot= 2, rem= 1
#                  130X=10, Y= 2, valid=1, quot= 5, rem= 0
#                  140X=10, Y= 2, valid=0, quot= 0, rem= 0
```

Now: 444,660,995 ps Delta: 0 sim:/slowdivision_tb

2. Pin Planning

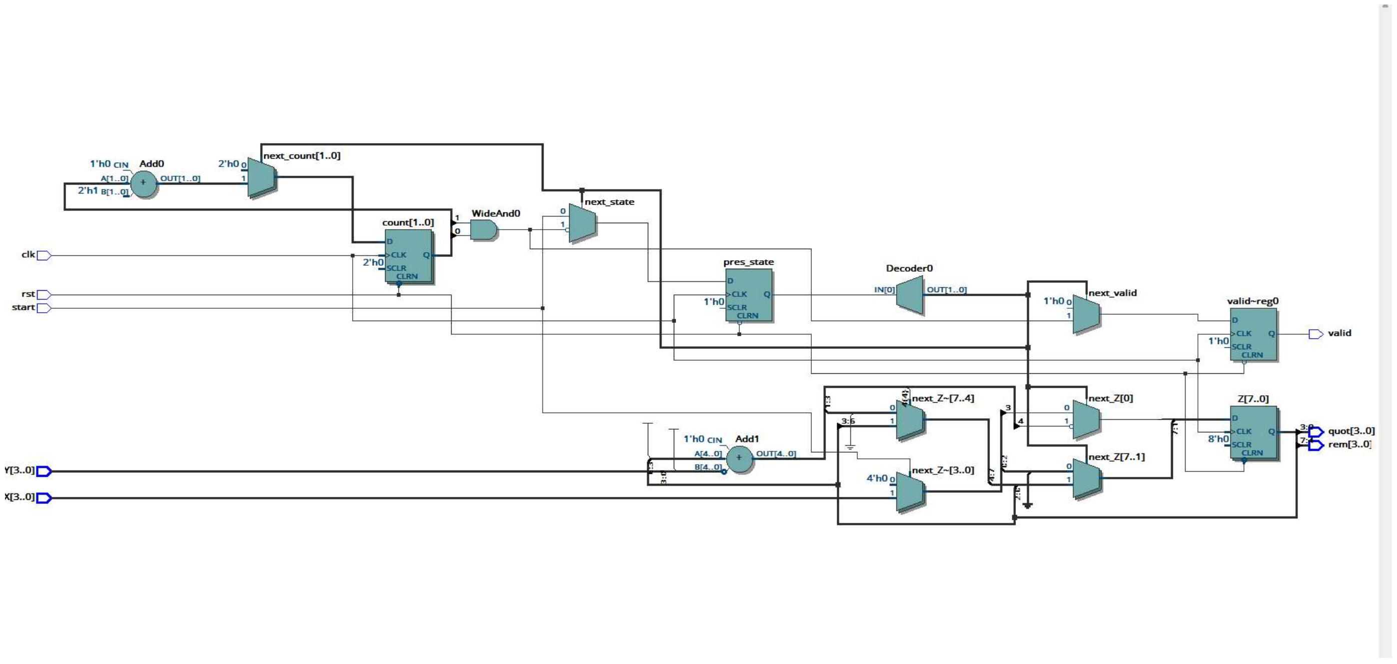


Pin Planner - C:/slowdivision/slowdivision - slowdivision

Top View - Wire Bond
Cyclone V - 5CSEMA5F31C6

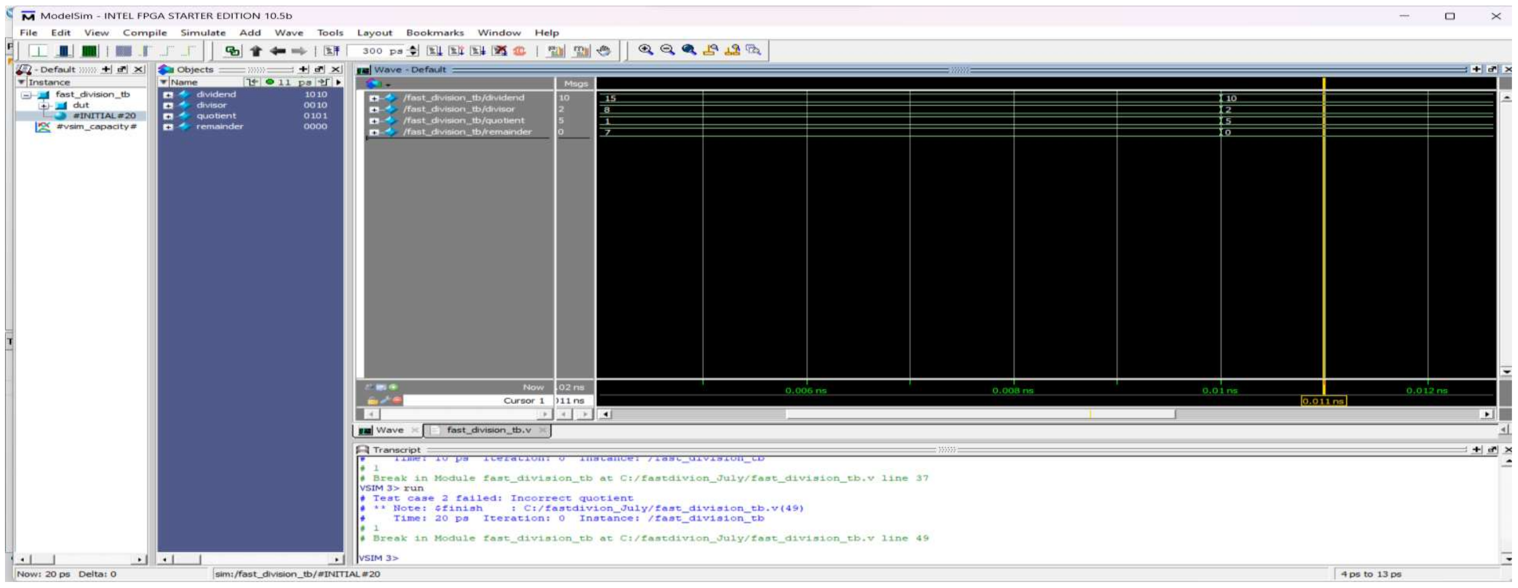
Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatic	/O Standard	Reserved	rrent Streng	Slew Rate	fferential Pa	Analog Setti	B/VCCT_GXI	r I/O Pin Te	cated Refclk	nmon Mode	ter Slew Rat	ifferential C
start	Input	PIN_AE12	3A	B3A_N0	PIN_AE12	3.3-V LVTTL		16mA...ult)									
rst	Input	PIN_AD10	3A	B3A_N0	PIN_AD10	3.3-V LVTTL		16mA...ult)									
clk	Input	PIN_AF14	3B	B3B_N0	PIN_AF14	3.3-V LVTTL		16mA...ult)									
Y[0]	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	3.3-V LVTTL		16mA...ult)									
Y[1]	Input	PIN_AD12	3A	B3A_N0	PIN_AD12	3.3-V LVTTL		16mA...ult)									
Y[2]	Input	PIN_AE11	3A	B3A_N0	PIN_AE11	3.3-V LVTTL		16mA...ult)									
Y[3]	Input	PIN_AC9	3A	B3A_N0	PIN_AC9	3.3-V LVTTL		16mA...ult)									
X[0]	Input	PIN_AB12	3A	B3A_N0	PIN_AB12	3.3-V LVTTL		16mA...ult)									
X[1]	Input	PIN_AC12	3A	B3A_N0	PIN_AC12	3.3-V LVTTL		16mA...ult)									
X[2]	Input	PIN_AF9	3A	B3A_N0	PIN_AF9	3.3-V LVTTL		16mA...ult)									
X[3]	Input	PIN_AF10	3A	B3A_N0	PIN_AF10	3.3-V LVTTL		16mA...ult)									
valid	Output	PIN_W21	5A	B5A_N0	PIN_W21	3.3-V LVTTL		16mA...ult)	1 (default)								
rem[0]	Output	PIN_W17	4A	B4A_N0	PIN_W17	3.3-V LVTTL		16mA...ult)	1 (default)								
rem[1]	Output	PIN_W19	4A	B4A_N0	PIN_W19	3.3-V LVTTL		16mA...ult)	1 (default)								
rem[2]	Output	PIN_Y19	4A	B4A_N0	PIN_Y19	3.3-V LVTTL		16mA...ult)	1 (default)								
rem[3]	Output	PIN_W20	5A	B5A_N0	PIN_W20	3.3-V LVTTL		16mA...ult)	1 (default)								
quot[0]	Output	PIN_V16	4A	B4A_N0	PIN_V16	3.3-V LVTTL		16mA...ult)	1 (default)								
quot[1]	Output	PIN_W16	4A	B4A_N0	PIN_W16	3.3-V LVTTL		16mA...ult)	1 (default)								
quot[2]	Output	PIN_V17	4A	B4A_N0	PIN_V17	3.3-V LVTTL		16mA...ult)	1 (default)								
quot[3]	Output	PIN_V18	4A	B4A_N0	PIN_V18	3.3-V LVTTL		16mA...ult)	1 (default)								
<<new node>>																	

3. RTL Simulation



Synthesis (Fast Division)

1. Functional Simulation

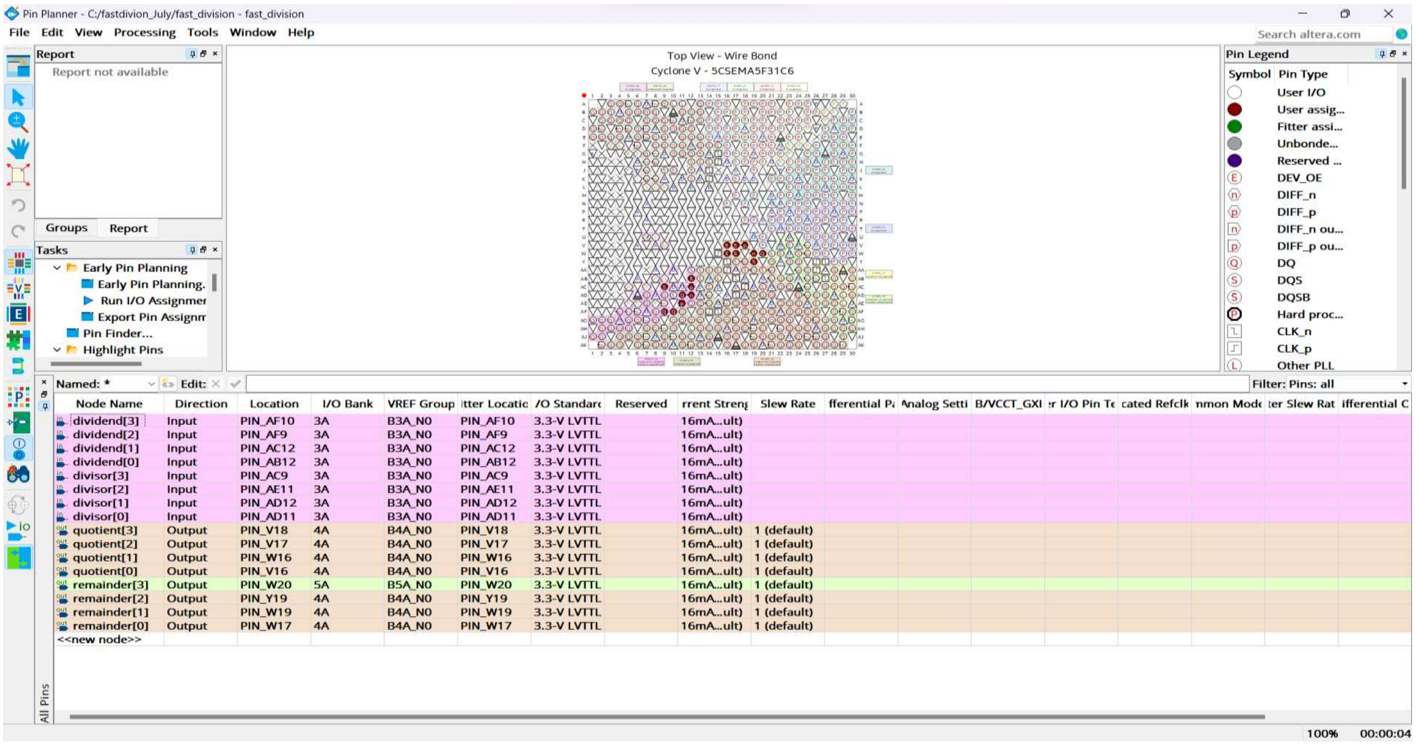

ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Objects
Wave - Default
Transcript
Wave
fast_division_tb.v

```
Instance
fast_division_tb
  dut
  #INITIAL#20
#vsim_capacity#

Name         11 ps
dividend     1010
divisor      0010
quotient     0101
remainder    0000

                               Msgs
/fast_division_tb/dividend     10     15   10
/fast_division_tb/divisor      2      8    2
/fast_division_tb/quotient     5      1    5
/fast_division_tb/remainder    0      7    0

Now      .02 ns
Cursor 1 .011 ns
```

```
# 1
# Break in Module fast_division_tb at C:/fastdivion_July/fast_division_tb.v line 37
VSIM 3> run
# Test case 2 failed: Incorrect quotient
# ** Note: $finish    : C:/fastdivion_July/fast_division_tb.v(49)
#    Time: 20 ps  Iteration: 0  Instance: /fast_division_tb
# 1
# Break in Module fast_division_tb at C:/fastdivion_July/fast_division_tb.v line 49

VSIM 3>
```

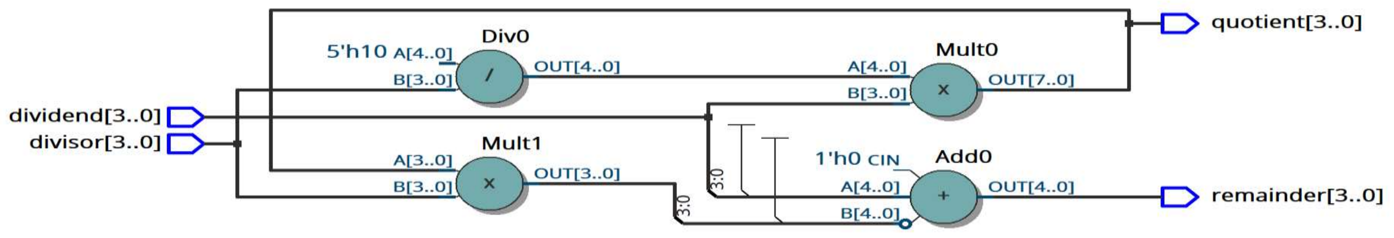
```
Now: 20 ps  Delta: 0     sim:/fast_division_tb/#INITIAL#20     4 ps to 13 ps
```

2. Pin Planning



Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatic	/O Standarc	Reserved	rrent Streng	Slew Rate	fferential Pa	Analog Setti	B/VCCT_GXI	r I/O Pin Te	cated Refclk	nmon Mode	ter Slew Rat	ifferential C
dividend[3]	Input	PIN_AF10	3A	B3A_N0	PIN_AF10	3.3-V LVTTL		16mA...ult)									
dividend[2]	Input	PIN_AF9	3A	B3A_N0	PIN_AF9	3.3-V LVTTL		16mA...ult)									
dividend[1]	Input	PIN_AC12	3A	B3A_N0	PIN_AC12	3.3-V LVTTL		16mA...ult)									
dividend[0]	Input	PIN_AB12	3A	B3A_N0	PIN_AB12	3.3-V LVTTL		16mA...ult)									
divisor[3]	Input	PIN_AC9	3A	B3A_N0	PIN_AC9	3.3-V LVTTL		16mA...ult)									
divisor[2]	Input	PIN_AE11	3A	B3A_N0	PIN_AE11	3.3-V LVTTL		16mA...ult)									
divisor[1]	Input	PIN_AD12	3A	B3A_N0	PIN_AD12	3.3-V LVTTL		16mA...ult)									
divisor[0]	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	3.3-V LVTTL		16mA...ult)									
quotient[3]	Output	PIN_V18	4A	B4A_N0	PIN_V18	3.3-V LVTTL		16mA...ult)	1 (default)								
quotient[2]	Output	PIN_V17	4A	B4A_N0	PIN_V17	3.3-V LVTTL		16mA...ult)	1 (default)								
quotient[1]	Output	PIN_W16	4A	B4A_N0	PIN_W16	3.3-V LVTTL		16mA...ult)	1 (default)								
quotient[0]	Output	PIN_V16	4A	B4A_N0	PIN_V16	3.3-V LVTTL		16mA...ult)	1 (default)								
remainder[3]	Output	PIN_W20	5A	B5A_N0	PIN_W20	3.3-V LVTTL		16mA...ult)	1 (default)								
remainder[2]	Output	PIN_Y19	4A	B4A_N0	PIN_Y19	3.3-V LVTTL		16mA...ult)	1 (default)								
remainder[1]	Output	PIN_W19	4A	B4A_N0	PIN_W19	3.3-V LVTTL		16mA...ult)	1 (default)								
remainder[0]	Output	PIN_W17	4A	B4A_N0	PIN_W17	3.3-V LVTTL		16mA...ult)	1 (default)								
<<new node>>																	

3. RTL Simulation



Performance Analysis

Slow Division

Resource Usage Summary:

Logic utilization: 9 ALMs needed out of 32,070 total ALMs on the device (< 1%)
Total LABs: 2 LABs partially or completely used out of 3,207 total LABs (< 1%)
Combinational ALUT usage for logic: 17
Dedicated logic registers: 13
I/O pins: 20 out of 457 (4%)
Global clocks: 1 out of 16 (6%)

Resource Utilization by Entity:

The slow division entity requires 8.5 ALMs, 17 combinational ALUTs, and 13 dedicated logic registers.

Routing Usage Summary:

Block interconnects: 27 out of 289,320 (< 1%)
C2 interconnects: 13 out of 119,108 (< 1%)
C4 interconnects: 8 out of 56,300 (< 1%)
Direct links: 3 out of 289,320 (< 1%)
Global clocks: 1 out of 16 (6%)

Multicorner Timing Analysis Summary:

Worst-case slack: Setup: 17.173, Hold: 0.147
Design-wide TNS: 0.0

Power Analysis:

Average toggle rate: 8.428 millions of transitions / sec
Total thermal power estimate: 425.87 mW
Overall, the slow division algorithm exhibits low resource utilization, reasonable timing slack, and moderate power consumption. It meets the required functionality while utilizing a small portion of the available resources on the device.

Fast Division

Power Analysis:
The power analysis for the fast division algorithm used in the project yielded the following findings:
Average toggle rate: 0.000 million transitions per second
Total thermal power estimate: 420.79 mW
Resource Utilization by Entity for fast division:
The resource utilization breakdown for the entities involved in the fast division algorithm is as follows:

fast division:

ALMs used in final placement: 15.0 (0.5)
Combinational ALUTs: 29 (1)
DSP Blocks: 2
lpm_divide:

ALMs used in final placement: 14.5 (0.0)
Combinational ALUTs: 28 (0)
lpm_divide_2am:

ALMs used in final placement: 14.5 (0.0)
Combinational ALUTs: 28 (0)
sign_div_unsign_8kh:

ALMs used in final placement: 14.5 (0.0)
Combinational ALUTs: 28 (0)
alt_u_div_mse:

ALMs used in final placement: 14.5 (14.5)
Combinational ALUTs: 28 (28)

Resource Usage Summary for fast division:
The overall resource utilization summary for the fast division algorithm is as follows:
Logic utilization (ALMs needed / total ALMs on the device): 15 / 32,070 (< 1%)
ALMs used for LUT logic: 15
Total LABs (partially or completely used): 3 / 3,207 (< 1%)
Combinational ALUT usage for logic:
7 input functions: 0
6 input functions: 1
5 input functions: 4
4 input functions: 6
<=3 input functions: 18

I/O pins used: 16 / 457 (4%)
DSP Blocks used: 2 / 87 (2%)
M10K blocks used: 0 / 397 (0%)
Total block memory bits used: 0 / 4,065,280 (0%)
Fractional PLLs used: 0 / 6 (0%)
Average interconnect usage: 0.0% (total, horizontal, vertical)
Peak interconnect usage: 0.5% (total, horizontal, vertical)

Routing Usage Summary for fast division:
The routing usage summary for the fast division algorithm is as follows:
Block interconnects: 52 / 289,320 (< 1%)
C12 interconnects: 4 / 13,420 (< 1%)
C2 interconnects: 17 / 119,108 (< 1%)
C4 interconnects: 18 / 56,300 (< 1%)
Direct links: 5 / 289,320 (< 1%)
Local interconnects: 13 / 84,580 (< 1%)
R14 interconnects: 28 / 12,676 (< 1%)
R14/C12 interconnect drivers: 28 / 20,720 (< 1%)
R3 interconnects: 25 / 130,992 (< 1%)
R6 interconnects: 33 / 266,960 (< 1%)

Conclusion

The implemented slow and fast division algorithms have been evaluated for their efficiency, performance, and trade-offs. Through detailed analysis and comparison, we have assessed their characteristics in terms of area utilization, power consumption, and operational speed. Based on our findings, we make the following recommendations:

Slow Division Algorithm

Advantages:
The slow division algorithm exhibits low resource utilization, occupying a small portion of available resources.
It performs division iteratively using traditional techniques, making it relatively straightforward to understand and implement.
The algorithm provides reasonable timing slack, ensuring proper synchronization in the design.

Fast Division Algorithm

Advantages:

The fast division algorithm showcases optimized calculations and a fixed number of iterations, leading to potentially faster execution time.
It utilizes a reciprocal approximation approach to perform division more efficiently.
The algorithm demonstrates low resource utilization, similar to the slow division algorithm.
The fast division algorithm has a smaller code size, making it more concise and potentially easier to maintain.
It offers a simpler structure and reduced complexity compared to the slow division algorithm.
Trade-offs and Considerations
Area Utilization: Both algorithms exhibit low resource utilization, utilizing a small portion of available resources.
Power Consumption: The slow division algorithm has slightly higher estimated power consumption compared to the fast division algorithm.
Operational Speed: The operational speed of the fast division algorithm is fast; its optimized calculations and fixed iterations suggest potentially faster execution compared to the slow division algorithm.
Code Size and Complexity: The fast division algorithm has a smaller code size and reduced complexity compared to the slow division algorithm, making it potentially easier to understand and maintain.
Algorithm Complexity: The slow division algorithm has higher complexity due to its iterative nature and conditional branching, while the fast division algorithm offers a simpler structure and reduced complexity.

References

Link for our GitHub repository, the code and testbench for both algorithms are within the repo.
GitHub Repository

Here are some additional references we used to complete the project:

1.https://electrobinary.blogspot.com/2020/08/restoring-division-verilog-code.html

2. https://www.codingninjas.com/studio/library/introduction-to-division-algorithm-in-computer-architecture

3. https://www.intel.com/content/www/us/en/docs/programmable/683475/19-4/introduction-to.html

4. https://youtu.be/ge09GjFUmKg

5. https://youtu.be/6ToR6vuRb3M

Acknowledgement

Firstly, we would like to extend our deepest appreciation to Intel® Unnati for their invaluable contribution throughout this endeavor. Their resources, expertise, and guidance were instrumental in achieving success in our project result. Their technical support enabled us to overcome challenges and make significant progress. Additionally, working with Intel® Unnati provided us with valuable insights into the industry's best practices and cutting-edge technologies.

Furthermore, we would like to express our heartfelt gratitude towards Er. Jyotish Chandran G for his exceptional mentorship throughout this project/industrial training experience. Their profound knowledge, guidance, and constant encouragement significantly contributed to our personal growth as a Hardware Designer. Their ability to provide clear explanations of complex concepts enhanced our understanding of the subject matter.

Additionally, we are grateful for Saintgits College of Engineering for providing an ideal environment that nurtures learning and supports students' endeavors such as this project/industrial training experience. The college's commitment towards fostering innovation coupled with its state-of-the-art facilities allowed us to explore new horizons within Verilog Hardware Design.

In conclusion, acknowledging those who have supported us during our journey is crucial in any field but particularly important in the realm of Verilog Hardware Design where Collaboration plays a pivotal role in success. Through this essay, we have expressed our sincere gratitude towards Intel® Unnati, Er. Jyotish Chandran G, and Saintgits College of Engineering for their unwavering support and contributions