Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

An internship report submitted by

MOUNISH S (URK21AI1022)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

under the supervision of

Dr. T. Jemima Jebaseeli
Associate Professor

DIVISION OF COMPUTER SCIENCE AND ENGINEERING
KARUNYA INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec-3 of the UGC Act, 1956)
Karunya Nagar, Coimbatore - 641 114. INDIA

Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that the report entitled**, "Implementation of Slow and Fast Division Algorithm for Computer Architecture**" is a bonafide record of **Intel Unnati Internship** work done at during the academic year 2022-2023 by,

MOUNISH S (URK21AI1022)

in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer science and engineering of Karunya Institute of Technology and Sciences.

Guide Signature

Dr. T. Jemima Jebaseeli

Associate Professor

ACKNOWLEDGEMENT

First and foremost, I praise and thank ALMIGTHY GOD whose blessings have bestowed in me the will power and confidence to carry out my internship.

I am grateful to our beloved founders **Late. Dr. D.G.S. Dhinakaran, C.A.I.I.B, Ph.D** and **Dr. Paul Dhinakaran, M.B.A, Ph.D**, for their love and always remembering us in their prayers.

We entend Our tanks to **Dr. Prince Arulraj, M.E., Ph.D.,** our honorable vice chancellor, **Dr. E. J. James, Ph.D.,** and **Dr. Ridling Margaret Waller, Ph.D.,** our honorable Pro-Vice Chancellor(s) and **Dr. R. Elijah Blessing, Ph.D.,** our respected Registrar for giving me this opportunity to do the internship.

I would like to thank **Dr. Ciza Thomas, M.E., Ph.D.,** Dean, School of Engineering and Technology for her direction and invaluable support to complete the same.

I would like to place my heart-felt thanks and gratitude to **Dr. J. Immanuel Johnraja, M.E., Ph.D.,** Head of the Department, Division of Computer Science and Engineering for his encouragement and guidance.

I feel it a pleasure to be indebted to **Dr. T. Jemima Jebaseeli,** Associate Professor, Division of CSE & **Mr. Abhishek Nandy** for their invaluable support, advice and encouragement.

I also thank all the staff members of the School of CST for extending their helping hands to make this in Internship a successful one.

I would also like to thank all my friends and my parents who have prayed and helped me during the Internship.

CONTENTS

Team Name: Data Wizards

Team members: Ansley claret, Mounish S, Lakshman J

Introduction

In the field of computer architecture, division algorithms play a crucial role in performing arithmetic operations efficiently. Division is an essential mathematical operation that involves the partitioning of numbers into equal parts. However, division can be a computationally expensive process, especially when dealing with large numbers or complex computations. To overcome this challenge, researchers and computer architects have developed various division algorithms, each with its own advantages and trade-offs in terms of speed and complexity.

Problem Statement

Implementation of slow and fast division algorithm in computer architecture.

Objectives

The goals of slow and fast division algorithms in computer architecture are to carry out division operations precisely and efficiently. It's crucial to remember that "slow" and "fast" are relative terms that might change depending on the situation and algorithms under consideration. The following are some overarching goals for both types:

Slow Division Algorithm:

Accuracy: The slow division algorithm seeks to deliver accurate division results that are precise.
Simplicity: It concentrates on implementation simplicity to make it simpler to comprehend and program.
Resource efficiency: Although it may not be speed-optimized, it strives to utilize as little hardware or processing power as possible, making it appropriate for systems with constrained capabilities.
Compatibility: Slow division algorithms may be designed to work well on a wide range of architectures, ensuring compatibility across different platforms.

Fast Division Algorithm:

Speed: The main goal of fast division algorithms is to complete division operations as rapidly as they can, reducing the amount of time needed to receive the result.
Efficiency: To achieve high-speed division operations, fast division algorithms strive to make efficient use of hardware resources, such as processors and memory.
Optimization: To streamline the division process and cut down on the amount of computational steps, these algorithms frequently use cutting-edge approaches like parallel processing, pipelining, or approximation methods.
Scalability: Fast division algorithms are made to scale well as input sizes increase, enabling division operations to be carried out effectively even with huge operands.
Performance-focused: These algorithms place a higher priority on performance than on simplicity, making them appropriate for high-performance computing environments or for applications where quick division operations are essential.

Slow Division

Restoring division method

The restoring division algorithm is a slow division algorithm that calculates the quotient digit by digit. This algorithm will generate a quotient and a remainder after the division algorithm. Division algorithm in computer architecture uses registers for storing the numbers and calculations.

Code

```
module SA_division_algorithm(
  input wire clk,
  input wire signed [3:0] dividend,
  input wire signed [3:0] divisor,
  output reg signed [3:0] quotient,
  output reg signed [3:0] remainder
);

  always @(posedge clk) begin
    if (divisor != 0) begin
      if (dividend >= divisor) begin
        quotient <= quotient + 1;
```

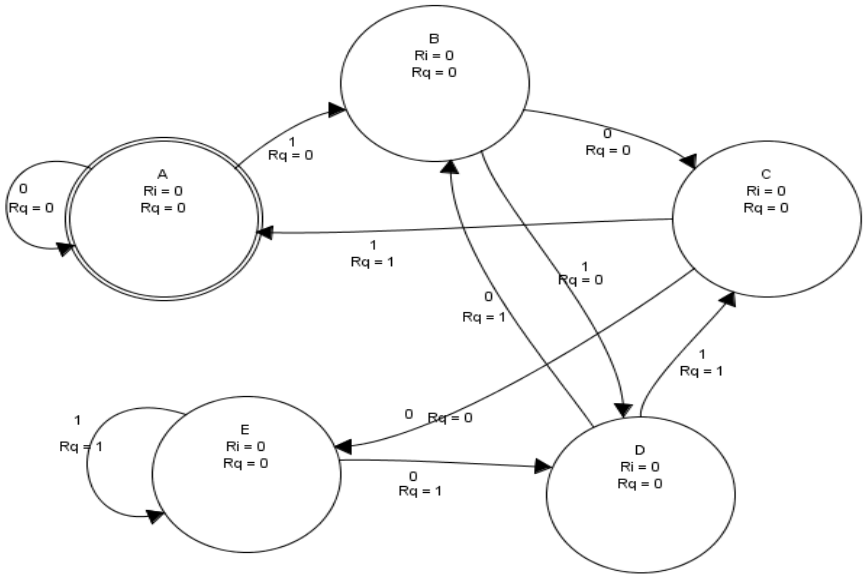
```
        remainder <= dividend - divisor;
      end else begin
        quotient <= quotient - 1;
        remainder <= dividend + divisor;
      end
    end else begin
      quotient <= 0;
      remainder <= dividend;
    end
  end

endmodule
```

Test Bench Code

```
module SA_division_algorithm_tb;
  reg clk;
  reg signed [3:0] dividend;
  reg signed [3:0] divisor;
  wire signed [3:0] quotient;
  wire signed [3:0] remainder;s

  SA_division_algorithm dut (
    .clk(clk),
    .dividend(dividend),
    .divisor(divisor),
    .quotient(quotient),
    .remainder(remainder)
  );

  initial begin
    clk = 0;
    dividend = 100;  // Set the dividend value
    divisor = 5;    // Set the divisor value

    #5;
    dividend = 50;   // Change the dividend value
    divisor = 3;    // Change the divisor value
    #10;            // Add a delay here

    #10;
    dividend = 75;   // Change the dividend value
    divisor = 8;    // Change the divisor value
    #10;            // Add a delay here

    #10;            // Add an additional delay here

    $finish;
  end
```

```
  always #5 clk = ~clk;  // Toggle the clock every 5 time units

  always @(posedge clk) begin
    $display("dividend = %d, divisor = %d, quotient = %d, remainder = %d",
      dividend, divisor, quotient, remainder);
  end

endmodule
```

FSM Model



Analysis and Elaboration

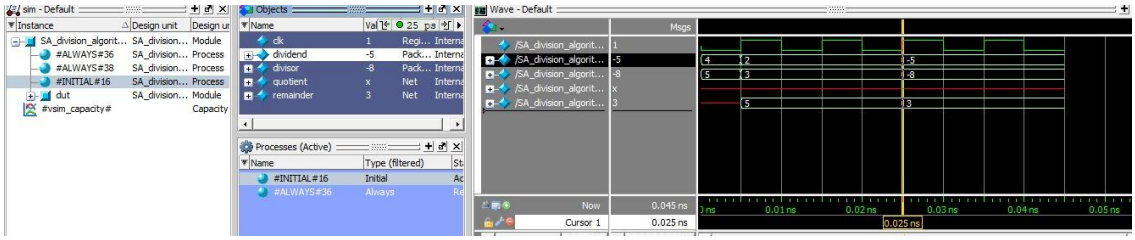
Type	ID	Message
ℹ	12021	Found 1 design units, including 1 entities, in source file sa_division_algorithm_tb.v
ℹ	12127	Elaborating entity "SA_division_algorithm" for the top level hierarchy
⚠	10230	Verilog HDL assignment warning at SA_division_algorithm.v(12): truncated value with size 32 to match size of target (4)
⚠	10230	Verilog HDL assignment warning at SA_division_algorithm.v(15): truncated value with size 32 to match size of target (4)
ℹ		Quartus Prime Analysis & Elaboration was successful. 0 errors, 3 warnings
ℹ		**********
ℹ		Running Quartus Prime Netlist Viewers Preprocess
ℹ		Command: quartus_npp SA_division_algorithm -c SA_division_algorithm --netlist_type=sgate
⚠	18236	Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM_PARALLEL_PROCESSORS in your
ℹ		Quartus Prime Netlist Viewers Preprocess was successful. 0 errors, 1 warning

Fitter

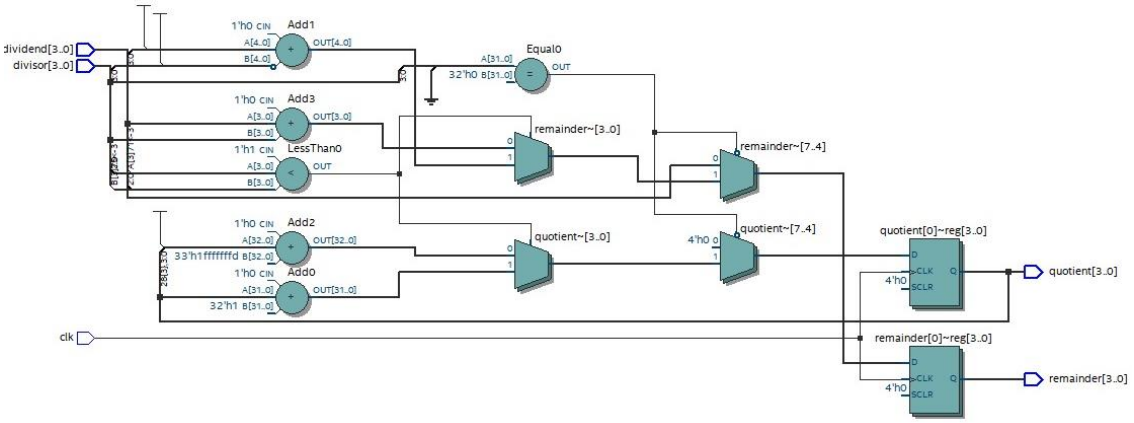
Post fitting

Type	ID	Message
ℹ	12021	Found 1 design units, including 1 entities, in source file sa_division_algorithm_tb.v
ℹ	12127	Elaborating entity "SA_division_algorithm" for the top level hierarchy
⚠	10230	Verilog HDL assignment warning at SA_division_algorithm.v(12): truncated value with size 32 to match size of target (4)
⚠	10230	Verilog HDL assignment warning at SA_division_algorithm.v(15): truncated value with size 32 to match size of target (4)
ℹ		Quartus Prime Analysis & Elaboration was successful. 0 errors, 3 warnings
ℹ		**********
ℹ		Running Quartus Prime Netlist Viewers Preprocess
ℹ		Command: quartus_npp SA_division_algorithm -c SA_division_algorithm --netlist_type=sgate
⚠	18236	Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM_PARALLEL_PROCESSORS in your
ℹ		Quartus Prime Netlist Viewers Preprocess was successful. 0 errors, 1 warning

RTL Simulator

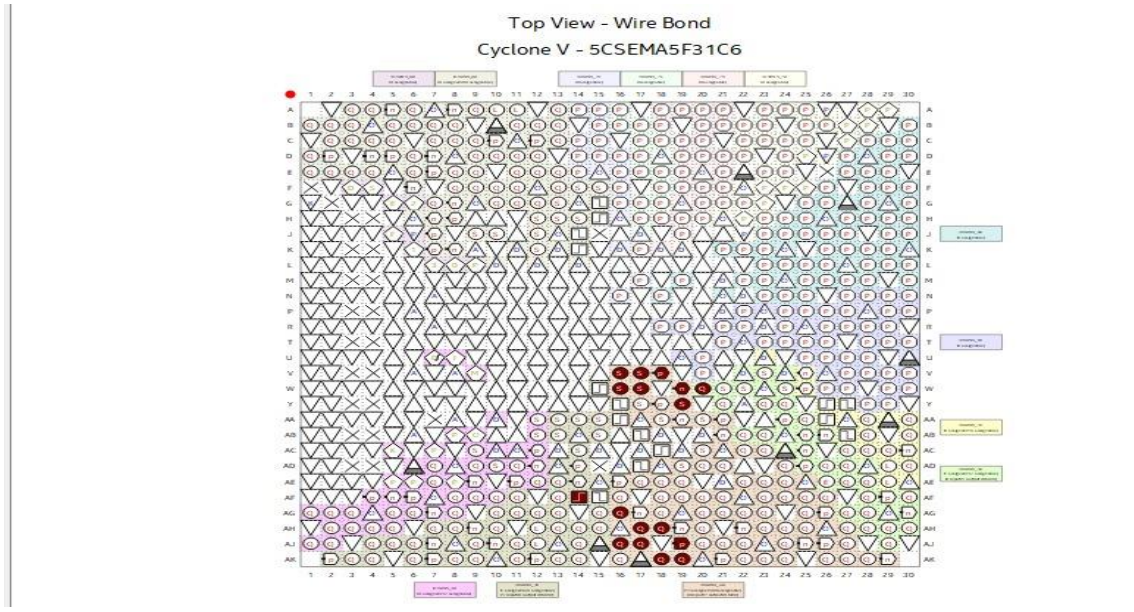


RTL Viewer

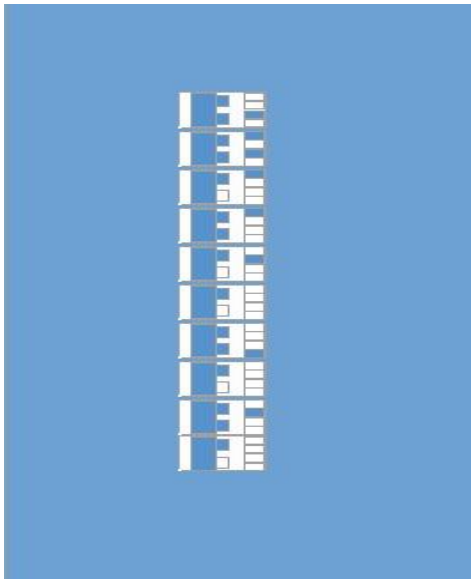


Pin planner

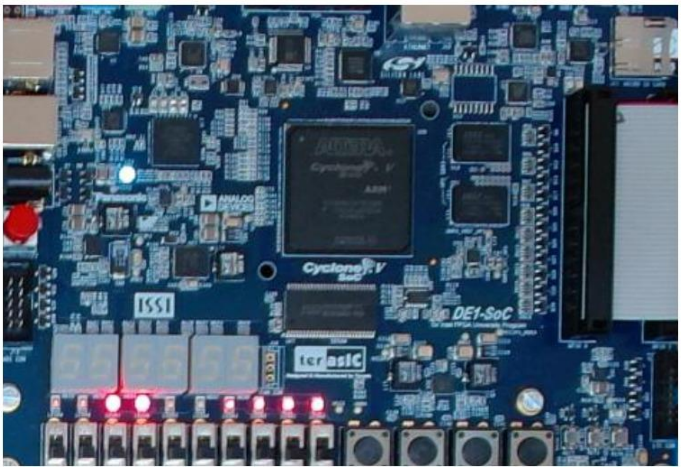
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Analog Setting	SXB/V
clk	Input	PIN_AF14	3B	B3B_N0	PIN_AF14	3.3-V LVTTL		16mA (default)				
dividend[3]	Input	PIN_AK18	4A	B4A_N0	PIN_AK18	3.3-V LVTTL		16mA (default)				
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 (default)			
quotient[2]	Output	PIN_W16	4A	B4A_N0	PIN_W16	3.3-V LVTTL		16mA (default)	1 (default)			
quotient[1]	Output	PIN_V17	4A	B4A_N0	PIN_V17	3.3-V LVTTL		16mA (default)	1 (default)			
quotient[0]	Output	PIN_V18	4A	B4A_N0	PIN_V18	3.3-V LVTTL		16mA (default)	1 (default)			
remainder[3]	Output	PIN_W17	4A	B4A_N0	PIN_W17	3.3-V LVTTL		16mA (default)	1 (default)			
remainder[2]	Output	PIN_W19	4A	B4A_N0	PIN_W19	3.3-V LVTTL		16mA (default)	1 (default)			
remainder[1]	Output	PIN_Y19	4A	B4A_N0	PIN_Y19	3.3-V LVTTL		16mA (default)	1 (default)			
remainder[0]	Output	PIN_W20	5A	B5A_N0	PIN_W20	3.3-V LVTTL		16mA (default)	1 (default)			

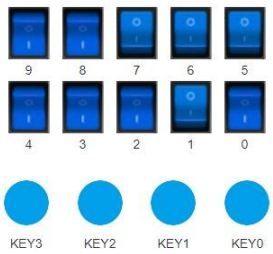


Chip planner



FGPA





Power report

Type	ID	Message
⚠	332060	Node: clk was determined to be a clock but was found without an associated clock assignment.
⚠	332068	No clocks defined in design.
ℹ	332143	No user constrained clock uncertainty found in the design. Calling "derive_clock_uncertainty"
ℹ	332154	The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
ℹ	223000	Starting Vectorless Power Activity Estimation
⚠	222013	Relative toggle rates could not be calculated because no clock domain could be identified for some nodes
ℹ	223001	Completed Vectorless Power Activity Estimation
⚠	215050	HPS power is being analyzed for a device with an HPS without HPS power.
ℹ	218000	Using Advanced I/O Power to simulate I/O buffers with the specified board trace model
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 420.65 mW
ℹ		Quartus Prime Power Analyzer was successful. 0 errors, 6 warnings

Fast Division

A lookup table is an array of data that maps input values to output values, thereby approximating a mathematical function. Given a set of input values, a lookup operation retrieves the corresponding output values from the table.

Code

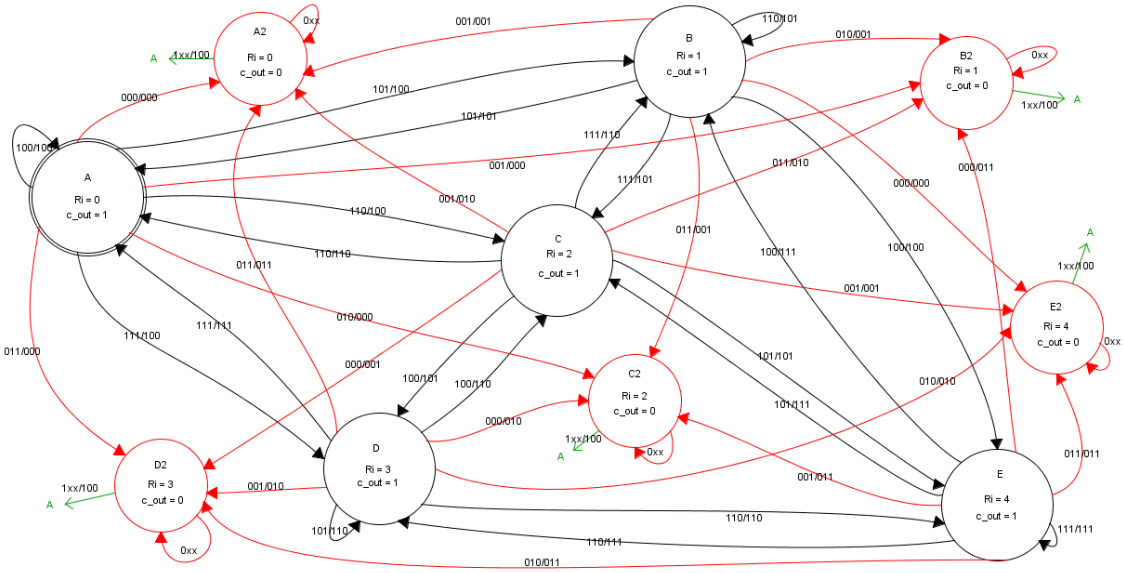
```
module fast_DA(divisor, dividend, remainder, result);

  input [2:0] divisor, dividend;
  output reg [2:0] result, remainder;

  // Variables
  reg [2:0] divisor_copy, dividend_copy;
  reg [2:0] temp;
  reg [1:0] i;

  always @(divisor or dividend)
  begin
    divisor_copy = divisor;
    dividend_copy = dividend;
    temp = 0;

    for (i = 0; i < 3; i = i + 1)
    begin
      temp = temp << 1;
      temp[0] = dividend_copy[2];
      dividend_copy = dividend_copy << 1;

      temp = temp - divisor_copy;

      if (temp[2] == 1 || temp[1] == 1)
      begin
        dividend_copy[0] = 0;
        temp = temp + divisor_copy;
      end
      else
      begin
        dividend_copy[0] = 1;
      end
    end

    result = dividend_copy;
    remainder = dividend - (divisor_copy * dividend_copy);
  end
endmodule
```

Test Bench Code

```
module division_tb;

  // Inputs
  reg [2:0] divisor;
  reg [2:0] dividend;

  // Outputs
  wire [2:0] result;
  wire [2:0] remainder;

  // Instantiate the division module
  division dut (
    .divisor(divisor),
    .dividend(dividend),
    .result(result),
    .remainder(remainder)
  );

  // Clock generation
  reg clk = 0;
  always #5 clk = ~clk;

  // Test stimuli
  initial begin
    #10;  // Wait for some time

    // Test case 1: divisor = 2, dividend = 1
    divisor = 2;
    dividend = 1;
    #10;  // Wait for division process
    $display("Dividend = %b, Divisor = %b, Quotient = %b, Remainder = %b", dividend, divisor, result, remainder);

    // Test case 2: divisor = 3, dividend = 5
    divisor = 3;
    dividend = 5;
    #10;  // Wait for division process
    $display("Dividend = %b, Divisor = %b, Quotient = %b, Remainder = %b", dividend, divisor, result, remainder);

    // Test case 3: divisor = 4, dividend = 7
    divisor = 4;
    dividend = 7;
    #10;  // Wait for division process
    $display("Dividend = %b, Divisor = %b, Quotient = %b, Remainder = %b", dividend, divisor, result, remainder);
```

```
  // Add more test cases as needed

  $finish;  // End simulation
end

always @(posedge clk)
begin
  // Display input values at each clock cycle (optional)
  $display("Clock = %t, Dividend = %b, Divisor = %b", $time, dividend, divisor);
end

endmodule
```

FSM Model

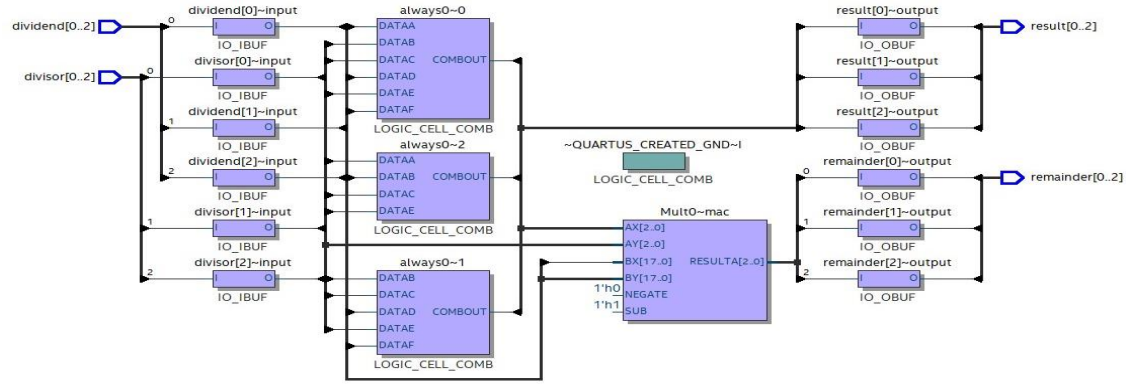


Analysis and Elaboration

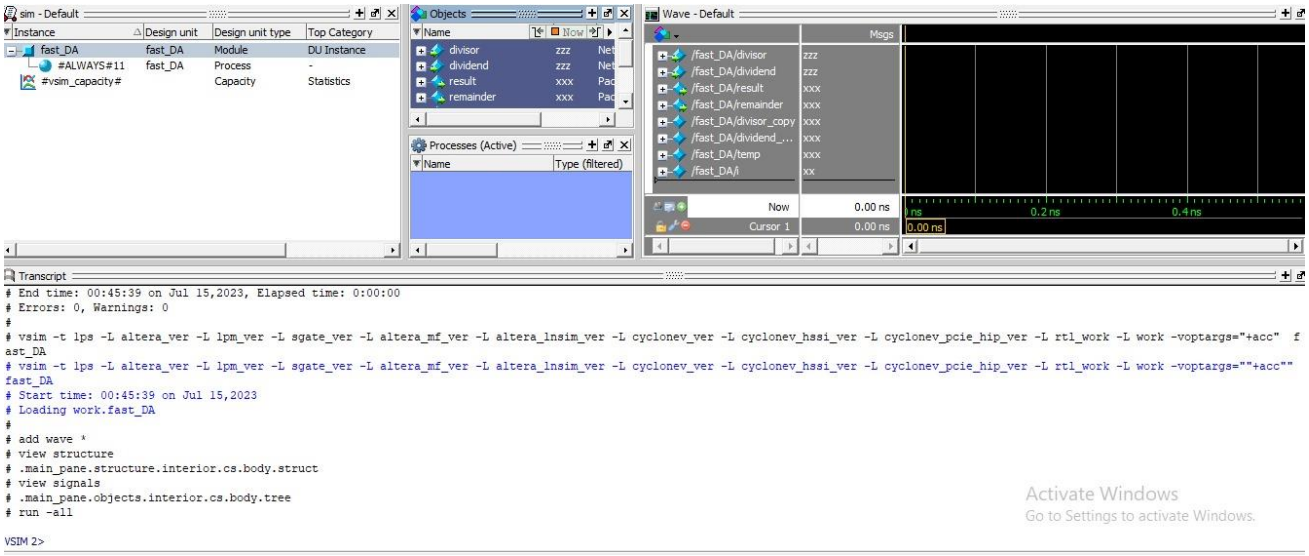
```
Info: 20030 Parallel compilation is enabled and will use 4 of the 4 processors detected
Info: 12021 Found 1 design units, including 1 entities, in source file fast_da.v
Info: 12021 Found 1 design units, including 1 entities, in source file fast_da_tb.v
Info: 12127 Elaborating entity "fast_DA" for the top level hierarchy
Warning: 10230 Verilog HDL assignment warning at fast_DA.v(17): truncated value with size 32 to match size of target (2)
Info: Quartus Prime Analysis & Elaboration was successful. 0 errors, 2 warnings
```

Fitter

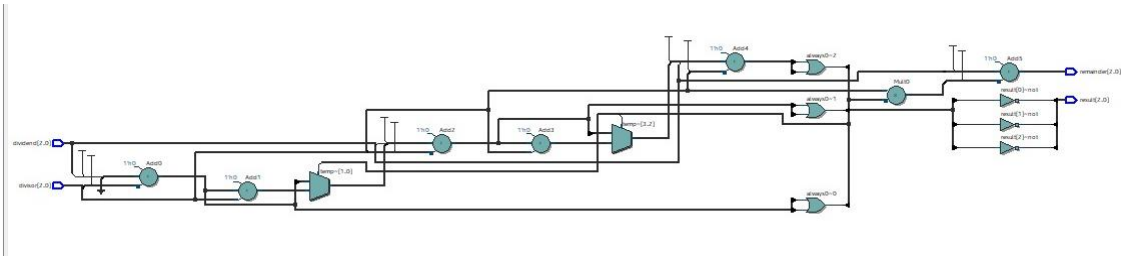
Post Fitting



RTL simulator



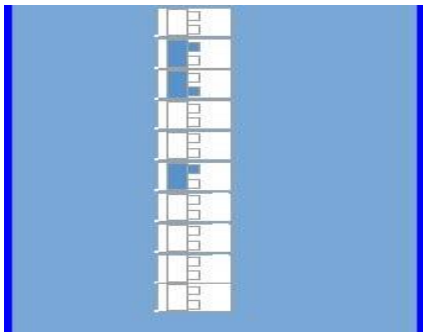
RTL viewer



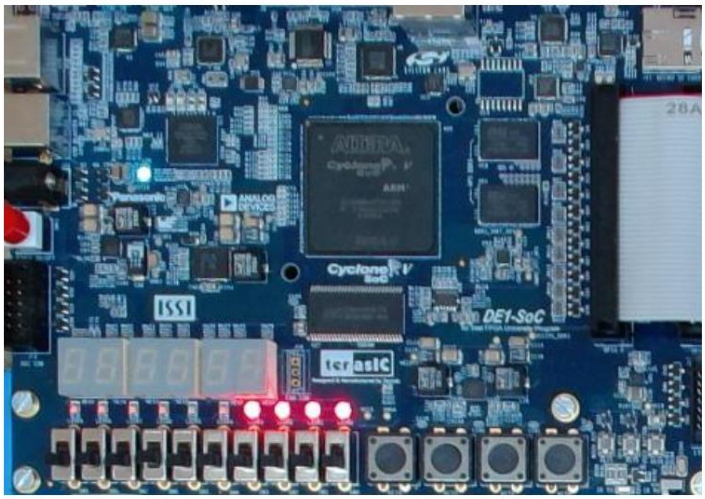
Pin Planner

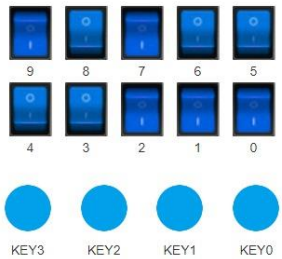
Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	er Analog Setting:	GXB/VCCT_GXB	iver I/O P
dividend[2]	Input	PIN_AK18	4A	B4A_N0	3.3-V LVTTL		16mA (default)					
dividend[1]	Input	PIN_AK19	4A	B4A_N0	3.3-V LVTTL		16mA (default)					
dividend[0]	Input	PIN_AJ19	4A	B4A_N0	3.3-V LVTTL		16mA (default)					
divisor[2]	Input	PIN_AJ17	4A	B4A_N0	3.3-V LVTTL		16mA (default)					
divisor[1]	Input	PIN_AJ16	4A	B4A_N0	3.3-V LVTTL		16mA (default)					
divisor[0]	Input	PIN_AH18	4A	B4A_N0	3.3-V LVTTL		16mA (default)					
remainder[2]	Output	PIN_V16	4A	B4A_N0	3.3-V LVTTL		16mA (default)	1 (default)				
remainder[1]	Output	PIN_W16	4A	B4A_N0	3.3-V LVTTL		16mA (default)	1 (default)				
remainder[0]	Output	PIN_V17	4A	B4A_N0	3.3-V LVTTL		16mA (default)	1 (default)				
result[2]	Output	PIN_V18	4A	B4A_N0	3.3-V LVTTL		16mA (default)	1 (default)				
result[1]	Output	PIN_W17	4A	B4A_N0	3.3-V LVTTL		16mA (default)	1 (default)				
result[0]	Output	PIN_W19	4A	B4A_N0	3.3-V LVTTL		16mA (default)	1 (default)				
<<new node>>												

Chip Planner

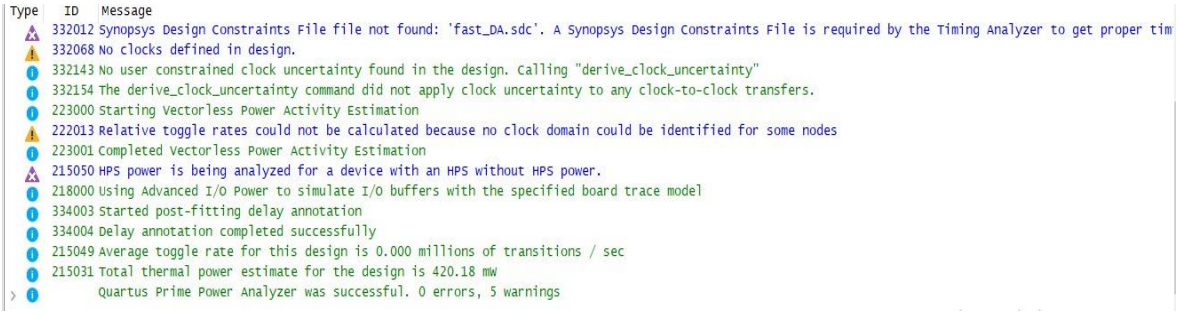


FGPA





Power Report



Type	ID	Message
⚠	332012	Synopsys Design Constraints File file not found: 'fast_DA.sdc'. A Synopsys Design Constraints File is required by the Timing Analyzer to get proper tim
⚠	332068	No clocks defined in design.
ℹ	332143	No user constrained clock uncertainty found in the design. Calling "derive_clock_uncertainty"
ℹ	332154	The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
ℹ	223000	Starting Vectorless Power Activity Estimation
⚠	222013	Relative toggle rates could not be calculated because no clock domain could be identified for some nodes
ℹ	223001	Completed Vectorless Power Activity Estimation
⚠	215050	HPS power is being analyzed for a device with an HPS without HPS power.
ℹ	218000	Using Advanced I/O Power to simulate I/O buffers with the specified board trace model
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 420.18 mW
ℹ		Quartus Prime Power Analyzer was successful. 0 errors, 5 warnings

VIDEO LINK

https://drive.google.com/file/d/1KyV1WDoUCqhp5NQkg52jfT-zO9x7hJfR/view?usp=gmail

CONCLUSION

Slow division algorithms prioritize accuracy and simplicity, making them suitable for systems with limited capabilities or applications where precision is paramount. While they may not offer the fastest performance, they excel in providing precise division results while minimizing the use of hardware resources. On the other hand, fast division algorithms focus on speed, efficiency, and performance optimization. They employ advanced techniques such as parallel processing, pipelining, or approximation methods to accelerate the division process. Fast division algorithms are particularly valuable in high-performance computing systems or applications that require rapid division operations. Hence the slow and fast division has been successfully implemented.