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

DANIEL GUNASEKARAN - Reg. No.: URK21CS1003

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

under the supervision of

Dr. GETZI JEBA LEELIPUSHPAM, Associate Professor

and

MR. ABHISHEK NANDY, Intel Industry Mentor

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

CONTENTS

Introduction

Introduction

Computing division is more difficult than any other arithmetic operations like addition, subtraction, and multiplication. This is due to the complexity and ineffective parallelization of division algorithms in general. Division is an iterative algorithm where the result corresponding to the quotient must be shifted to the remainder utilizing a Euclidean measure. This makes it challenging to parallelize the process because intermediate results of one phase are used as inputs to the following step. On the contrary, multiplication can be accelerated by breaking it down into a series of bit manipulation techniques.

To expedite division and lower the number of clock cycles needed to complete the operation, fast division algorithms are necessary. This is crucial since division is a frequently utilized operation in a variety of computing tasks, and cutting down on the time needed to conduct division can greatly enhance a system's overall performance.

Problem Statement

Design and Implementation of slow and fast division algorithms in Computer Architecture

Objectives

- Determine and define the design's inputs, outputs, and states.
- Any necessary assumptions should be made and documented.
- For the design, create a state diagram and label each state change.
- Create an algorithm for the state diagram's implementation on a Moore or Mealy machine.
- Create Verilog code to put the design into practice.
- Use test scenarios to confirm the design's functionality.
- Analyze the data and make judgements based on them.
- Make sure the design complies with the requirements.
- Create a design block diagram with the correct inputs and outputs.
- Examine and improve the Finite State Machine (Mealy/Moore) in accordance with the design requirements.
- Create appropriate test cases to simulate and validate the design's functioning.
- Analyze the time and summarize the design.
- Run the design on a Virtual Lab Software

Slow Division

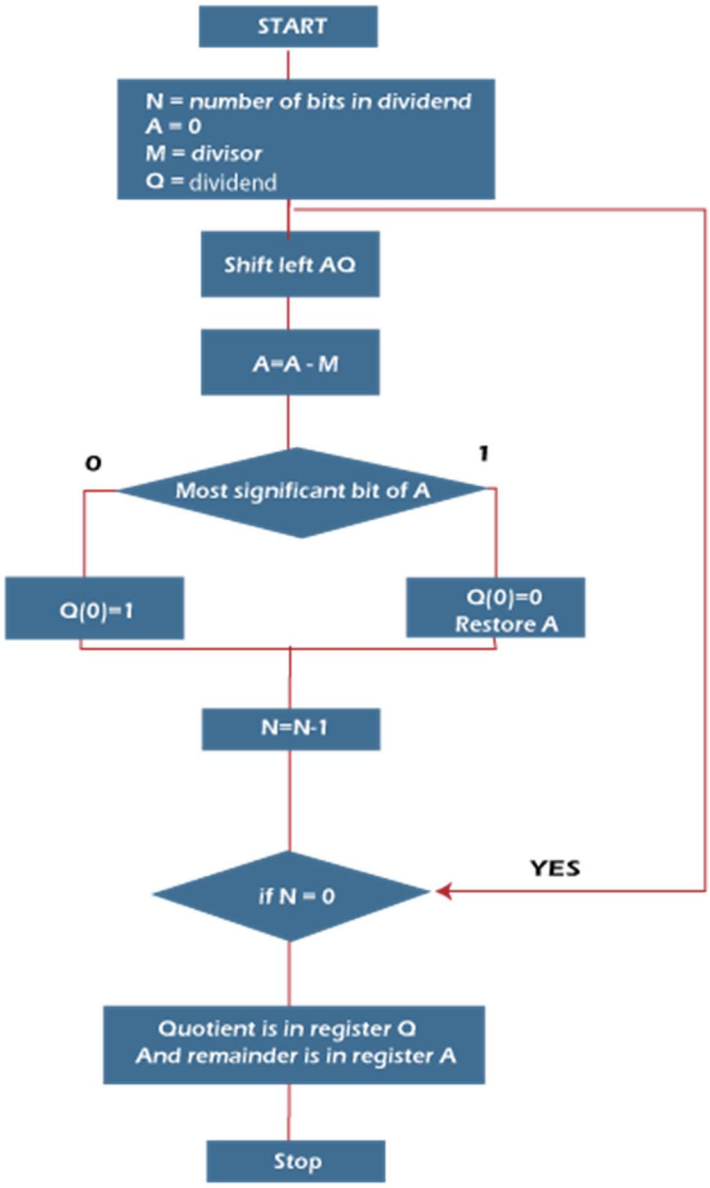
Restoring Division Method

Two unsigned numbers can be divided using restoring division. It restores the value of the Accumulator (A) after each or a few iterations, which is why it is termed restoring. By default, the divisor is reduced without considering the size of the dividend set. It is known as trial subtraction.

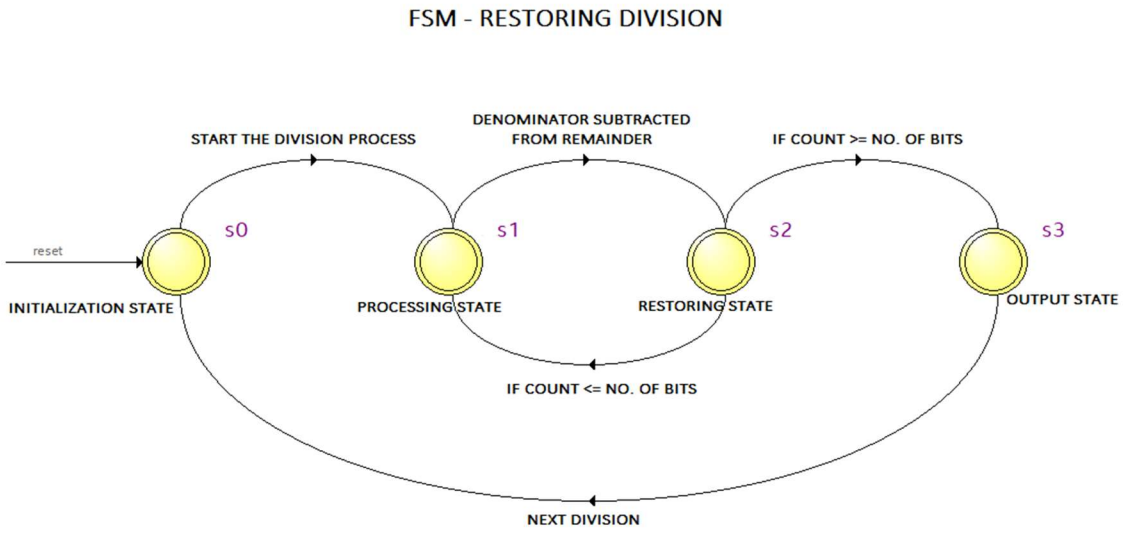
Algorithm

- With the relevant values (Q = Dividend, M = Divisor, A = 0, n = the number of bits in dividend), the registers are initialized.
- Registers A and Q's contents are combined and relocated to the left.
- The contents of registers M and A are subtracted, and the resulting value is kept in register A.
- The A's most important bit is verified. The least significant bit of Q is toggled to 1 if it is 0. If it is 1, on the other hand, the least significant bit of Q is changed to 0, restoring the value of register A to what it was before to the subtraction with M.
- The counter n's value is decreased.
- If the current value of n decreases to zero, the loop is ended; otherwise, step 2 is repeated.
- The registers Q and A hold the remainder and quotient, respectively.

Flow Chart


START
N = number of bits in dividend
A = 0
M = divisor
Q = dividend
Shift left AQ
A=A - M
0
1
Most significant bit of A
Q(0)=1
Q(0)=0
Restore A
N=N-1
YES
if N = 0
Quotient is in register Q
And remainder is in register A
Stop

FSM Diagram



The states are S0, S1, S2, and S3: -

- **S0**: This is the initialization state. The FSM enters this state when it is reset. In this state, the count, quotient register, and done signal are reset to 0. The denominator is loaded and aligned, and the remainder is set to the numerator. The FSM then moves to state s1.

- **S1**: This is the processing state. In this state, the denominator is subtracted from the remainder. The FSM then moves to state s2.

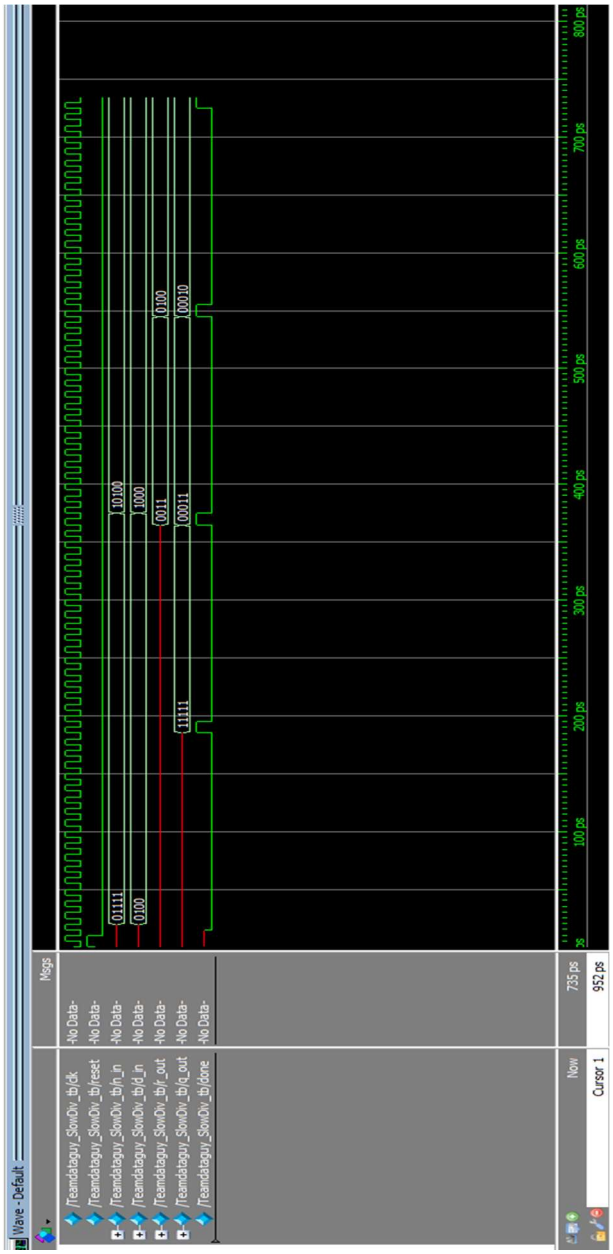
- **S2**: This is the restoring state. In this state, if the remainder is negative, it is restored to its previous value by adding back the denominator. The quotient register is shifted left by 1 bit, with its least significant bit set to 0 if the remainder was negative or 1 otherwise. The count is incremented and the denominator is shifted right by 1 bit. If the count has reached 8, indicating that the division is complete, the FSM moves to state s3. Otherwise, it moves back to state s1 for further processing.

- **S3**: This is the output state. In this state, the final values of the quotient and remainder are outputted and the done signal is set to 1. The FSM then moves back to state s0 to start a new division.

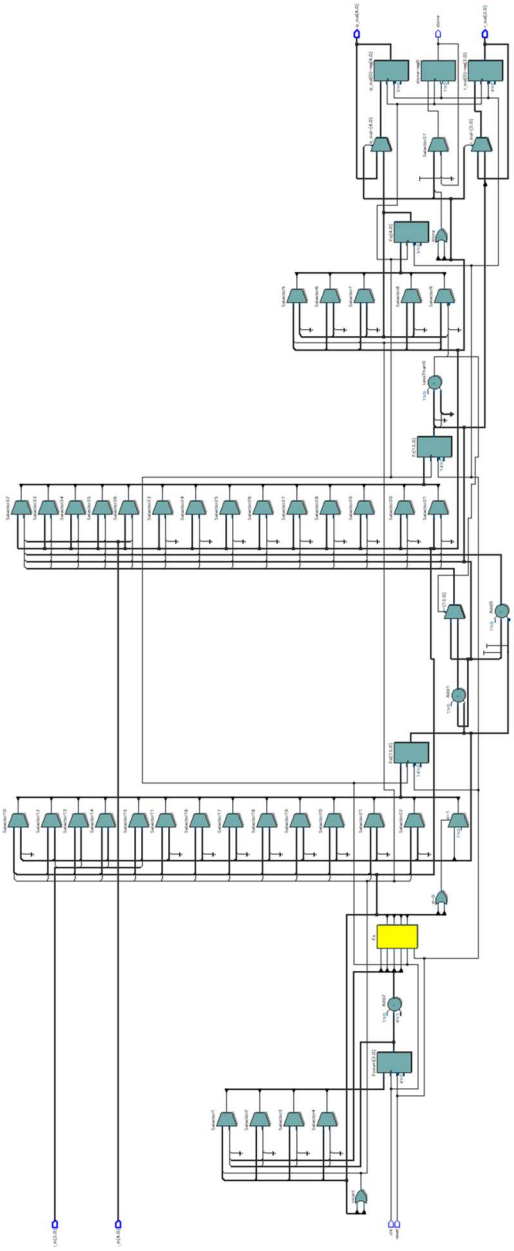
Analysis and Elaboration

RTL Simulation

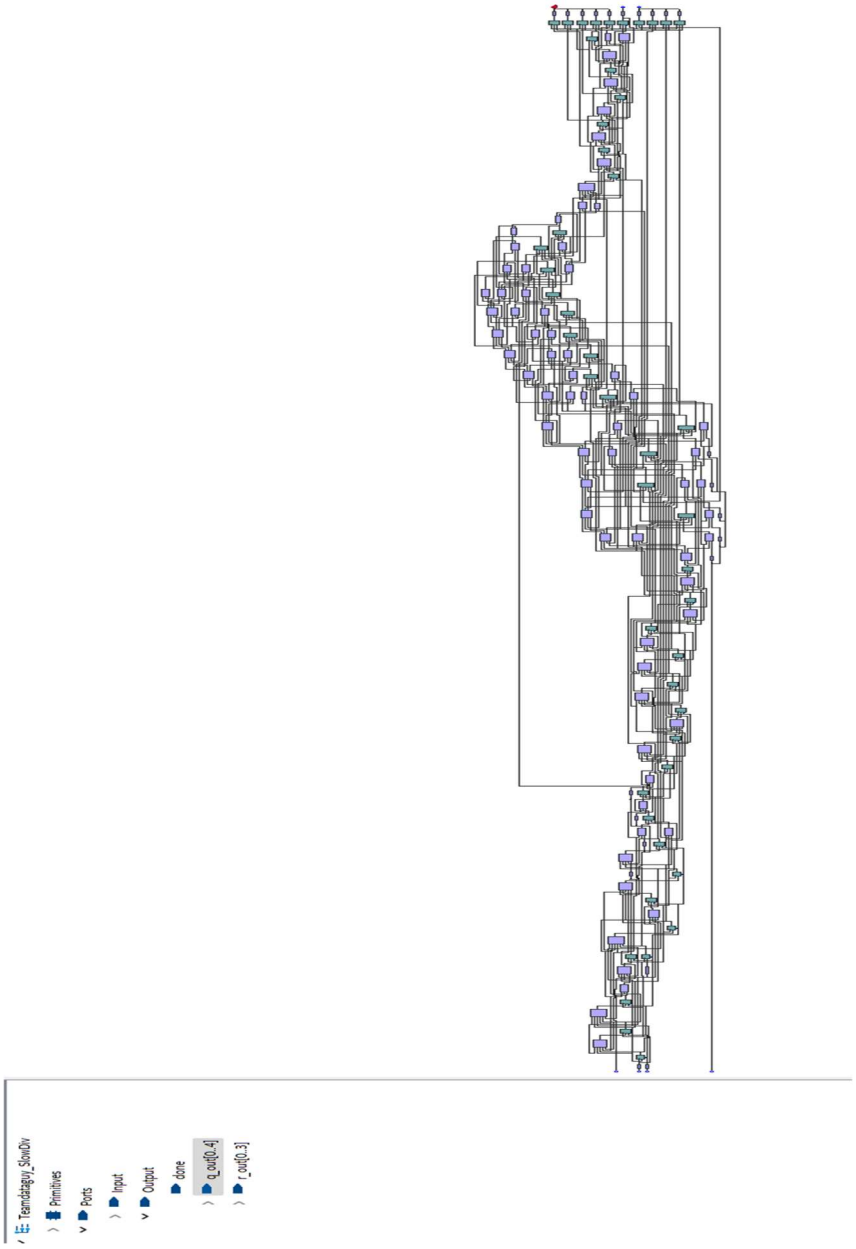
```
# Test case 1: 15 /  4 =  3 R  3
# Test case 2: 20 /  8 =  2 R  4
# ** Note: $finish    : C:/Intel_Unnati/Teamdataguy_SlowDiv/Teamdataguy_SlowDiv_tb.v(42)
#    Time: 735 ps  Iteration: 0  Instance: /Teamdataguy_SlowDiv_tb
```



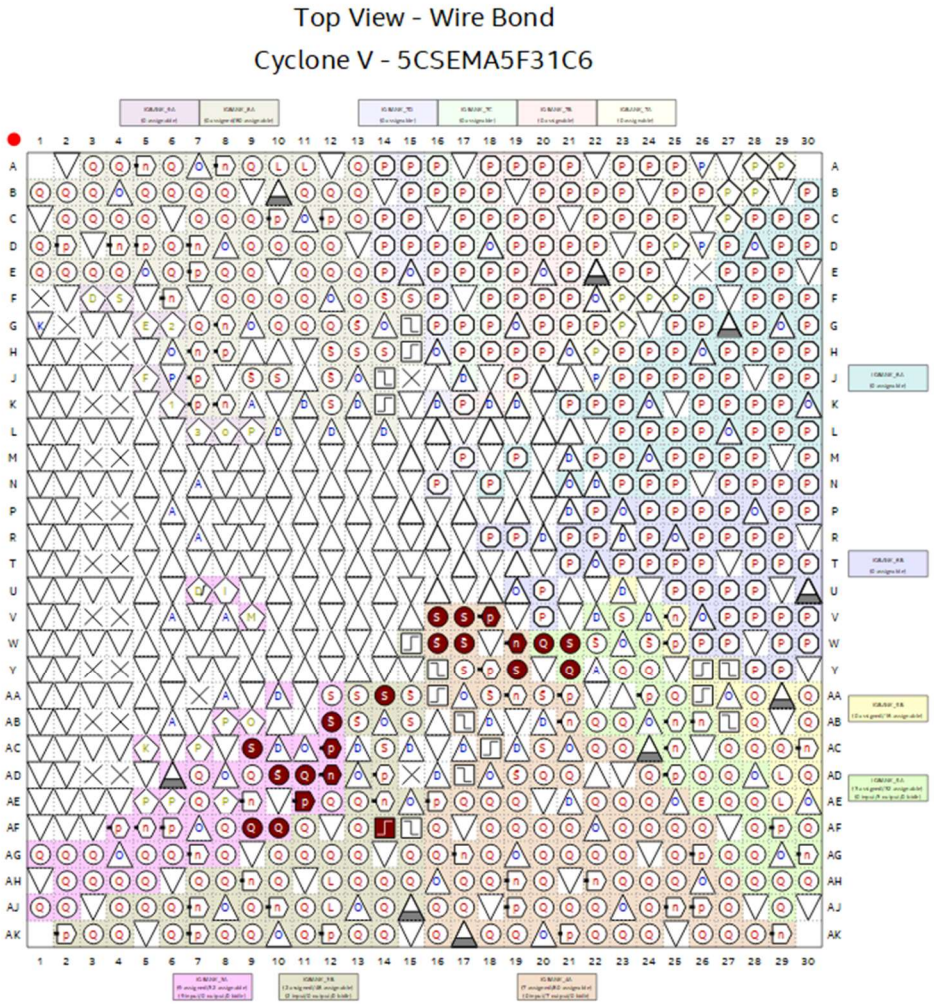
RTL Viewer



Technology Map Viewer (Post-Mapping)



Pin Planning



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
clk	Input	PIN_AF14	3B	B3B_N0	PIN_AF14	3.3-V LVTTL		16mA (default)
d_in[3]	Input	PIN_AD10	3A	B3A_N0	PIN_AD10	3.3-V LVTTL		16mA (default)
d_in[2]	Input	PIN_AC9	3A	B3A_N0	PIN_AC9	3.3-V LVTTL		16mA (default)
d_in[1]	Input	PIN_AE11	3A	B3A_N0	PIN_AE11	3.3-V LVTTL		16mA (default)
d_in[0]	Input	PIN_AD12	3A	B3A_N0	PIN_AD12	3.3-V LVTTL		16mA (default)
done	Output	PIN_Y21	5A	B5A_N0	PIN_Y21	3.3-V LVTTL		16mA (default)
n_in[4]	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	3.3-V LVTTL		16mA (default)
n_in[3]	Input	PIN_AF10	3A	B3A_N0	PIN_AF10	3.3-V LVTTL		16mA (default)
n_in[2]	Input	PIN_AF9	3A	B3A_N0	PIN_AF9	3.3-V LVTTL		16mA (default)
n_in[1]	Input	PIN_AC12	3A	B3A_N0	PIN_AC12	3.3-V LVTTL		16mA (default)
n_in[0]	Input	PIN_AB12	3A	B3A_N0	PIN_AB12	3.3-V LVTTL		16mA (default)
q_out[4]	Output	PIN_W17	4A	B4A_N0	PIN_W17	3.3-V LVTTL		16mA (default)
q_out[3]	Output	PIN_V18	4A	B4A_N0	PIN_V18	3.3-V LVTTL		16mA (default)
q_out[2]	Output	PIN_V17	4A	B4A_N0	PIN_V17	3.3-V LVTTL		16mA (default)
q_out[1]	Output	PIN_W16	4A	B4A_N0	PIN_W16	3.3-V LVTTL		16mA (default)
q_out[0]	Output	PIN_V16	4A	B4A_N0	PIN_V16	3.3-V LVTTL		16mA (default)
r_out[3]	Output	PIN_W21	5A	B5A_N0	PIN_W21	3.3-V LVTTL		16mA (default)
r_out[2]	Output	PIN_W20	5A	B5A_N0	PIN_W20	3.3-V LVTTL		16mA (default)
r_out[1]	Output	PIN_Y19	4A	B4A_N0	PIN_Y19	3.3-V LVTTL		16mA (default)
r_out[0]	Output	PIN_W19	4A	B4A_N0	PIN_W19	3.3-V LVTTL		16mA (default)
reset	Input	PIN_AA14	3B	B3B_N0	PIN_AA14	3.3-V LVTTL		16mA (default)

Analysis and Synthesis Summary

Analysis & Synthesis Status	Successful - Tue Jul 04 23:48:44 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Teamdataguy_SlowDiv
Top-level Entity Name	Teamdataguy_SlowDiv
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	48
Total pins	21
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Fitter

Fitter Summary

Fitter Status	Successful - Tue Jul 04 23:52:23 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Teamdataguy_SlowDiv
Top-level Entity Name	Teamdataguy_SlowDiv
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	38 / 32,070 (< 1 %)
Total registers	51
Total pins	21 / 457 (5 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total RAM Blocks	0 / 397 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Power Utilization Summary

Power Analyzer Status	Successful - Sat Jul 08 10:34:40 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Teamdataguy_SlowDiv
Top-level Entity Name	Teamdataguy_SlowDiv
Family	Cyclone V
Device	5CSEMA5F31C6
Power Models	Final
Total Thermal Power Dissipation	421.04 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	411.23 mW
I/O Thermal Power Dissipation	9.81 mW

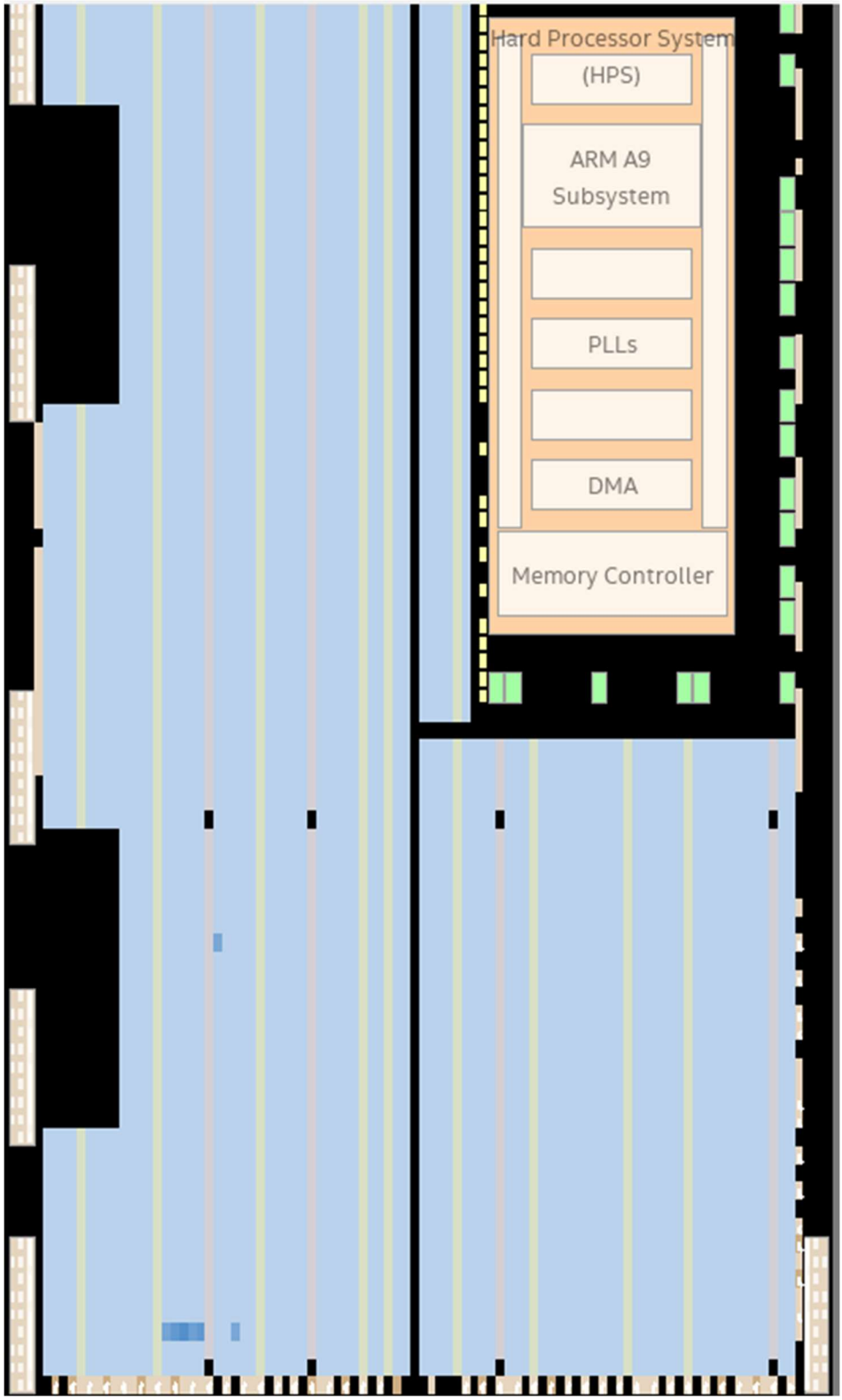
Resource Usage Summary

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	38 / 32,070	< 1 %
2	> ALMs needed [=A-B+C]	38	
3			
4	Difficulty packing design	Low	
5			
6	> Total LABs: partially or completely used	7 / 3,207	< 1 %
7			
8	> Combinational ALUT usage for logic	72	
9	Combinational ALUT usage for route-throughs	3	
10			
11	> Dedicated logic registers	51	
12			
13	Virtual pins	0	
14	> I/O pins	21 / 457	5 %
15			
16	> Hard processor system peripheral utilization		
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	1	
27	SERDES Transmitters	0 / 100	0 %
28	SERDES Receivers	0 / 100	0 %
29	JTAGs	0 / 1	0 %
30	ASMI blocks	0 / 1	0 %
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)	0.0% / 0.1% / 0.0%	
37	Peak interconnect usage (total/H/V)	0.7% / 0.9% / 0.2%	
38	Maximum fan-out	51	
39	Highest non-global fan-out	38	
40	Total fan-out	451	
41	Average fan-out	2.67	

Chip Planner



Assembler

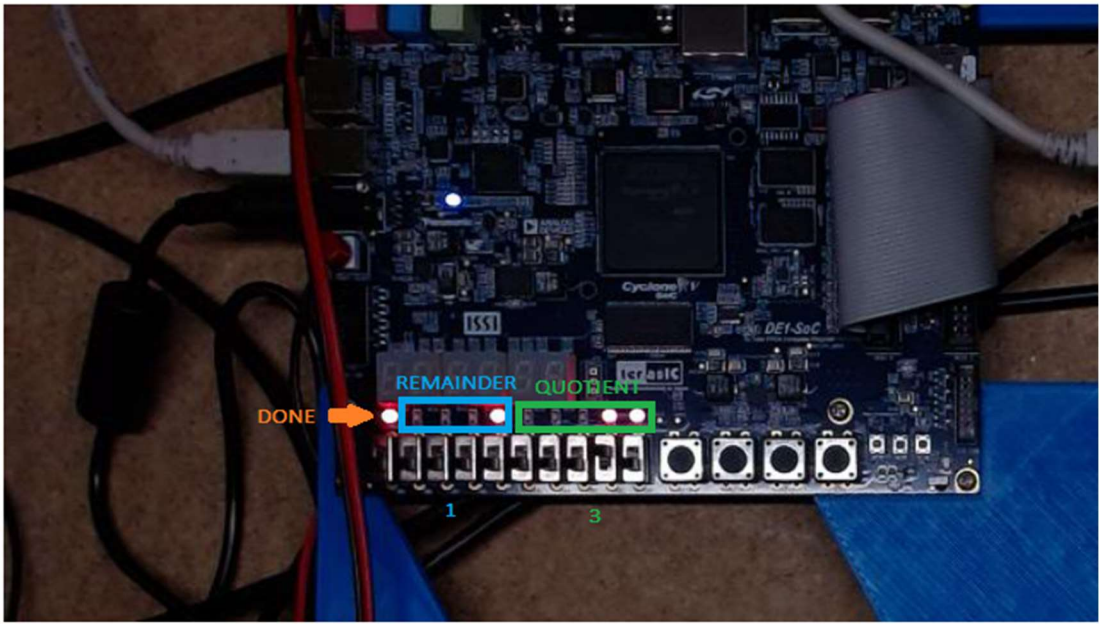
Generate Programming Files

Assembler Status	Successful - Tue Jul 04 23:54:42 2023
Revision Name	Teamdataguy_SlowDiv
Top-level Entity Name	Teamdataguy_SlowDiv
Family	Cyclone V
Device	5CSEMA5F31C6

Teamdataguy_SlowDiv	04/07/2023 23:54	SOF File	6,534 KB

Sample Input and Output on Simulator

Input: Dividend = 7 [00111] and Divisor = 2 [0010]
Expected Output: Quotient = 3[00011] and Remainder = [0001]



Fast Division

Look-up Table Method

Fast division Using a short lookup table and one multiplication operations is a technique that can speed up the division process. There are two steps in the algorithm. The table lookup and the initial multiplication are handled simultaneously in the first phase. The second step involves doing the second multiplication to produce the quotient.

This division generates 2m-bit results that are certain to be correct to one ulp (unit in the final place) using a single multiplier and a lookup table of 2m+1 bits. The fundamental technique can quickly produce results by employing a multiplier and a tiny lookup table.

Algorithm

Algorithm: Fast Division using Lookup Table

Input: xin (8-bit dividend), yin (8-bit divisor), clk (clock input), enbl (enable input)
Output: xyout (16-bit output, where the first 8 bits correspond to the integer part and the next 8 bits correspond to the fractional part)

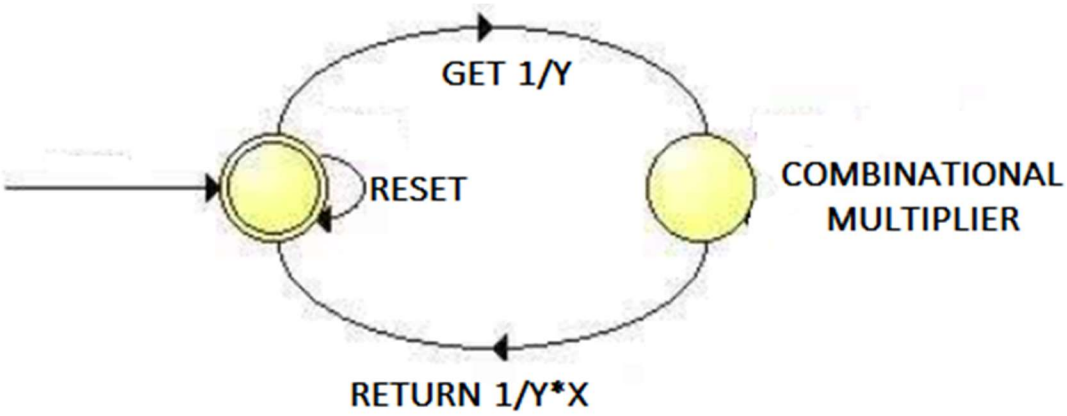
1. Initialize internal registers xin(8-bit), yin (8-bit), and xyout (16-bit)
2. Instantiate a mult module with inputs out, xin, and a temporary wire temp, and output product
3. On changes in the value of yin:
a. Use a case statement to implement a lookup table that assigns a value to the out register based on the value of yin
4. On the positive edge of the clock:
a. If enbl is high:
i. If xin is equal to yin, assign the value 16'h0100 to the register xyouti
ii. Else if yin is equal to 8'h01, assign {xin, 8'h00} to the register xyouti
iii. Else, assign the value of the temporary wire temp to the register xyouti
5. Assign the value of register xyouti to the output wire xyout

Algorithm: Combinatorial Multiplier

Input: multiplier (8-bit), multiplicand (8-bit)
Output: product (16-bit)

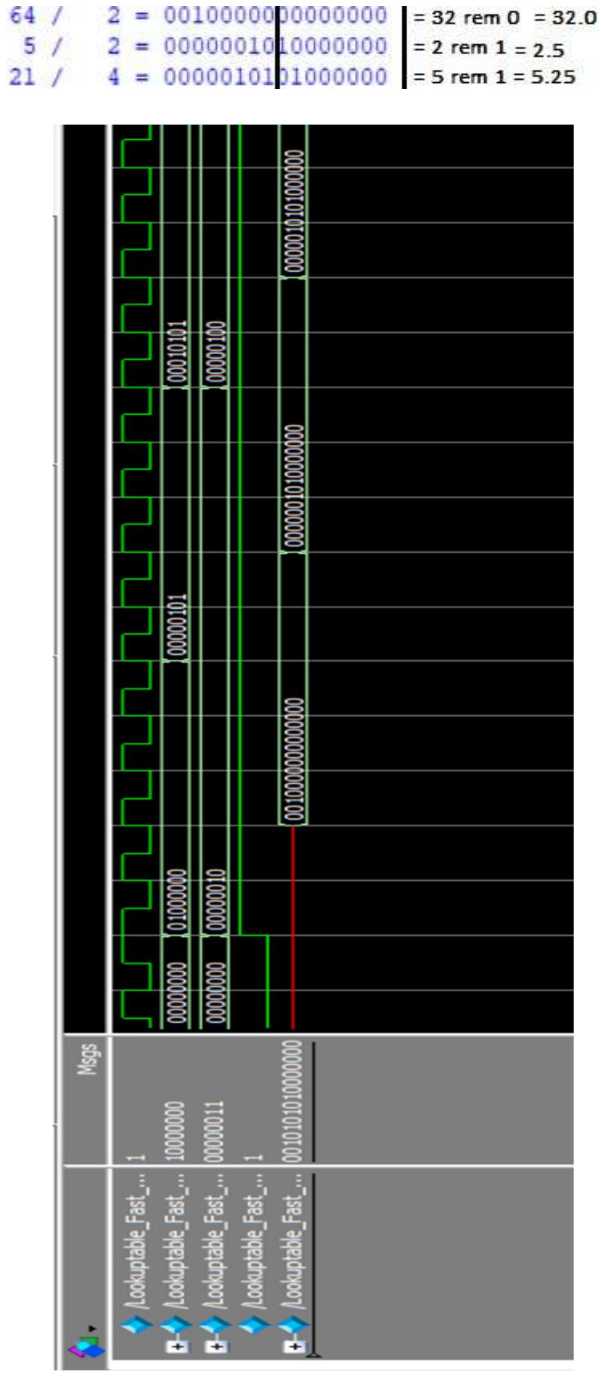
1. Initialize internal register product
2. On changes in either of the inputs (multiplier or multiplicand):
a. Initialize the value of register product to 0
b. For each bit of the multiplier:
i. If the current bit of the multiplier is 1, add (multiplicand << i) to the value of register product

FSM Diagram

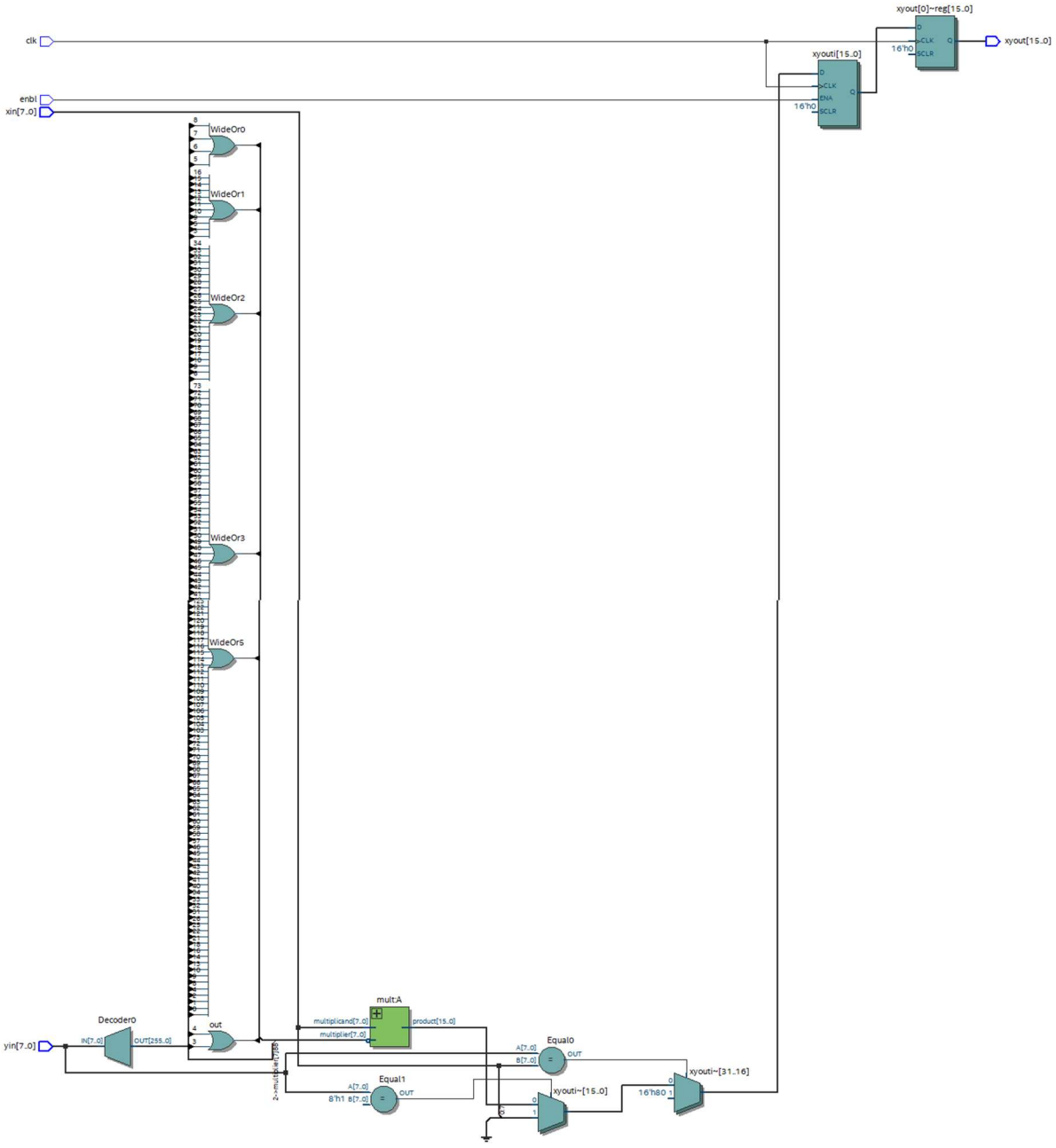


Analysis and Elaboration

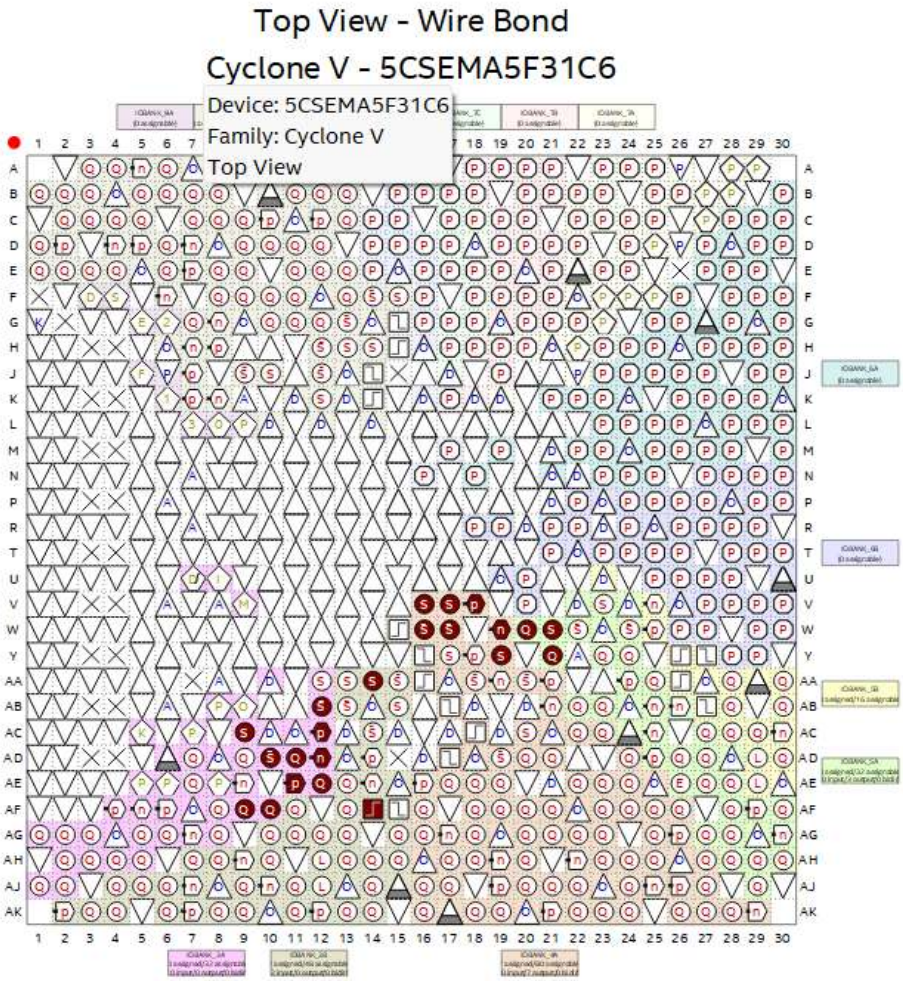
RTL Simulation



RTL Viewer


clk
enbl
xin[7..0]
yin[7..0]
WideOr0
WideOr1
WideOr2
WideOr3
WideOr5
Decoder0
IN[7..0]
OUT[255..0]
out
mult:A
multiplicand[7..0]
multiplier[7..0]
product[15..0]
Equal0
A[7..0]
B[7..0]
OUT
Equal1
8'h1
xyouti~[15..0]
xyouti~[31..16]
16'h80
xyouti[15..0]
16'h0
xyout[0]~reg[15..0]
xyout[15..0]

Pin Planning



Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	urrent Streng
yin[4]	Input	PIN_AE12	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
yin[3]	Input	PIN_AD10	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
yin[2]	Input	PIN_AC9	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
yin[1]	Input	PIN_AE11	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
yin[0]	Input	PIN_AD12	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
xyout[11]	Output	PIN_Y21	5A	B5A_N0	3.3-V LVTTL		16mA ...ault)
xyout[10]	Output	PIN_W21	5A	B5A_N0	3.3-V LVTTL		16mA ...ault)
xyout[9]	Output	PIN_W20	5A	B5A_N0	3.3-V LVTTL		16mA ...ault)
xyout[8]	Output	PIN_Y19	4A	B4A_N0	3.3-V LVTTL		16mA ...ault)
xyout[7]	Output	PIN_W19	4A	B4A_N0	3.3-V LVTTL		16mA ...ault)
xyout[6]	Output	PIN_W17	4A	B4A_N0	3.3-V LVTTL		16mA ...ault)
xyout[5]	Output	PIN_V18	4A	B4A_N0	3.3-V LVTTL		16mA ...ault)
xyout[4]	Output	PIN_V17	4A	B4A_N0	3.3-V LVTTL		16mA ...ault)
xyout[3]	Output	PIN_W16	4A	B4A_N0	3.3-V LVTTL		16mA ...ault)
xyout[2]	Output	PIN_V16	4A	B4A_N0	3.3-V LVTTL		16mA ...ault)
xin[4]	Input	PIN_AD11	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
xin[3]	Input	PIN_AF10	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
xin[2]	Input	PIN_AF9	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
xin[1]	Input	PIN_AC12	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
xin[0]	Input	PIN_AB12	3A	B3A_N0	3.3-V LVTTL		16mA ...ault)
enbl	Input	PIN_AA14	3B	B3B_N0	3.3-V LVTTL		16mA ...ault)
clk	Input	PIN_AF14	3B	B3B_N0	3.3-V LVTTL		16mA ...ault)

Analysis and Synthesis Summary

Flow Status	Successful - Thu Jul 13 21:02:28 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Lookuptable_Fast_Division
Top-level Entity Name	Lookuptable_Fast_Division
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	32
Total pins	34
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Fitter

Fitter Summary

Flow Status	Successful - Thu Jul 13 21:05:20 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Lookuptable_Fast_Division
Top-level Entity Name	Lookuptable_Fast_Division
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	85 / 32,070 (< 1 %)
Total registers	32
Total pins	34 / 457 (7 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

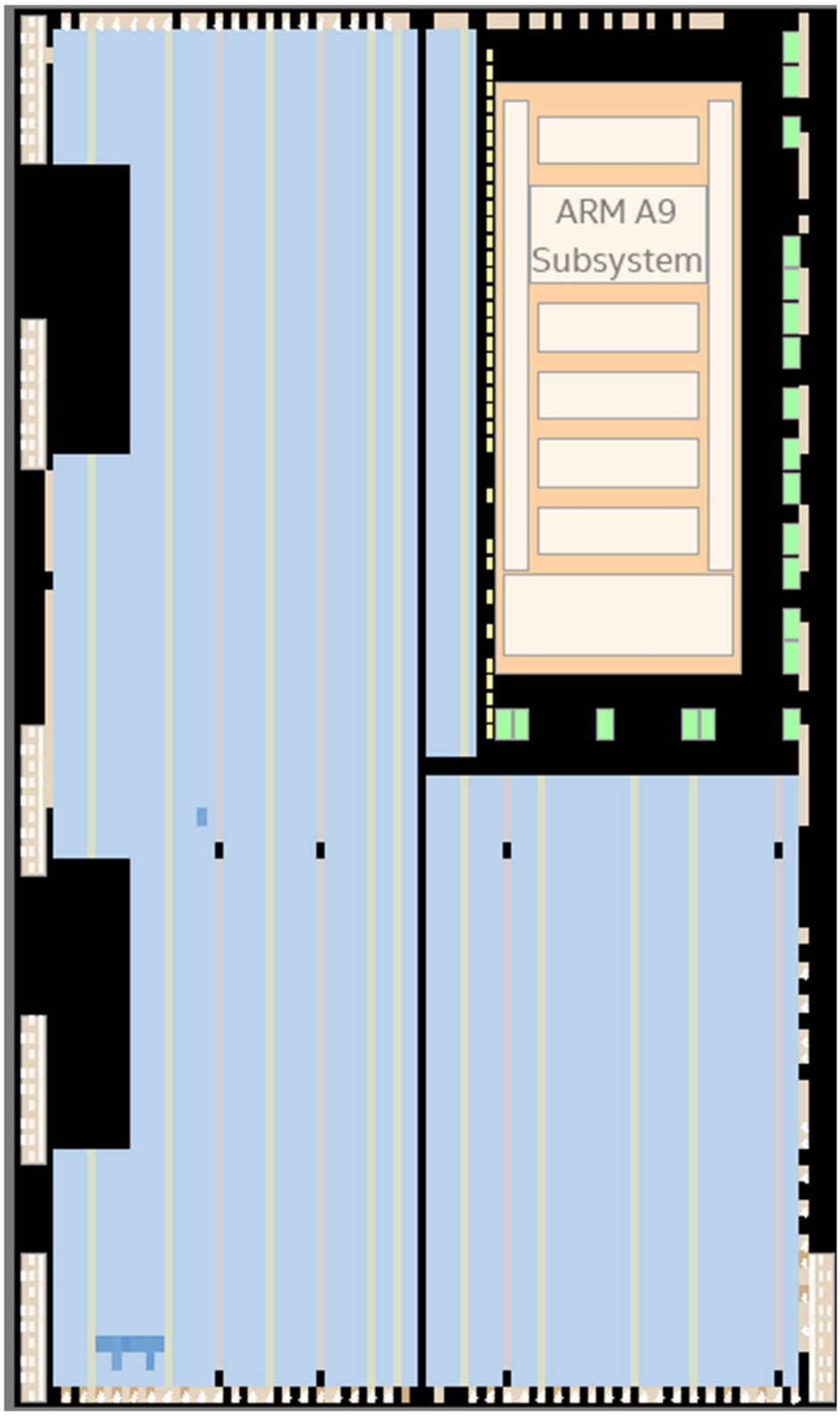
Power Utilization Summary

Power Analyzer Status	Successful - Thu Jul 13 21:06:29 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Lookuptable_Fast_Division
Top-level Entity Name	Lookuptable_Fast_Division
Family	Cyclone V
Device	5CSEMA5F31C6
Power Models	Final
Total Thermal Power Dissipation	421.39 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	411.23 mW
I/O Thermal Power Dissipation	10.15 mW

Resource Usage Summary

	Resource	Usage	%
1	Logic utilization (ALMs ... / total ALMs on device)	85 / 32,070	< 1 %
2	> ALMs needed [=A-B+C]	85	
3			
4	Difficulty packing design	Low	
5			
6	> Total LABs: partially or completely used	11 / 3,207	< 1 %
7			
8	> Combinational ALUT usage for logic	156	
9	Combinational ALUT usage for route-throughs	2	
10			
11	> Dedicated logic registers	32	
12			
13	Virtual pins	0	
14	> I/O pins	34 / 457	7 %
15			
16	> Hard processor system peripheral utilization		
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	1	
27	SERDES Transmitters	0 / 100	0 %
28	SERDES Receivers	0 / 100	0 %
29	JTAGs	0 / 1	0 %
30	ASMI blocks	0 / 1	0 %
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)	0.1% / ... / 0.0%	
37	Peak interconnect usage (total/H/V)	1.2% / ... / 0.7%	
38	Maximum fan-out	86	
39	Highest non-global fan-out	86	
40	Total fan-out	840	
41	Average fan-out	3.24	

Chip Planner



Assembler

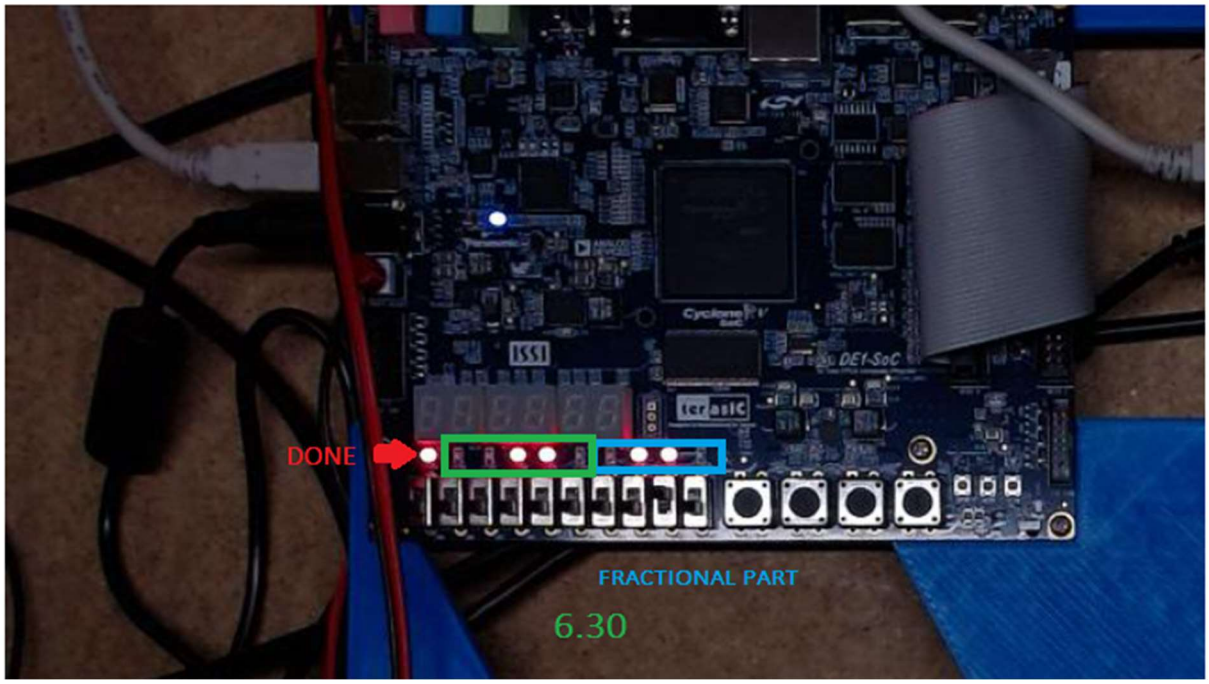
Generate Programming Files

Assembler Status	Successful - Thu Jul 13 21:11:56 2023
Revision Name	Lookuptable_Fast_Division
Top-level Entity Name	Lookuptable_Fast_Division
Family	Cyclone V
Device	5CSEMA5F31C6

Lookuptable_Fast_Division	13/07/2023 21:11	SOF File	6,534 KB

Sample Input and Output on Simulator

Input: Dividend = 19 [10011] and Divisor = 3 [0011]
Expected Output: 00110.0011 = 6.33..



Conclusions and Further Scope

By subtracting the divisor from the partial remainder and restoring it if the result is negative, the slow division process known as "restoring division" provides one digit of the quotient every iteration. Although it is easy to create, it has a large latency and needs many cycles for each digit.
Lookup table division is a quick division method that produces the quotient by performing two multiplication operations on a tiny lookup table. It has a Taylor series expansion foundation and has a two-cycle accuracy limit. Although it uses less hardware than restoring division, the table's memory footprint is higher.

Let us look at some data: -

	SLOW DIVISION	FAST DIVISION
TIME TAKEN	200 ps	40 ps

Here we can see that: -

- As predicted the Fast division algorithm takes significantly less time as compared to the slow division
- Although the method used here is highly space consuming it requires more memory while executing in real time i.e., it will be blocking more Random-access memory for computing a division algorithm
- This is why we need less space consuming fast division Algorithms
- Combining the advantages of both restoring division and lookup table division in a hybrid approach could potentially reduce the latency and memory usage while maintaining accuracy and flexibility. By switching between the two algorithms depending on the input size and precision requirements, the hybrid approach could provide an efficient solution for a wide range of division problems.

Future Scope: -

Combining the benefits of both techniques and employing a hybrid strategy that alternates between restoring division and lookup table division based on the input size and precision requirements is a potential area for future development. While preserving accuracy and flexibility, this might lower latency and memory utilization. Which is our Ultimate Goal

Reference

- https://www.javatpoint.com/restoring-division-algorithm-for-unsigned-integer
- https://www.geeksforgeeks.org/restoring-division-algorithm-unsigned-integer
- https://www.intel.com/content/www/us/en/programmable/customertraining/webex/Verilog/presentation_html5.html
- https://www.intel.com/content/www/us/en/docs/programmable/683082/22-1/systemverilog-state-machine-coding-example.html
- https://www.intel.com/content/www/us/en/docs/programmable/683082/22-1/state-machine-hdl-guidelines.html
- https://en.wikipedia.org/wiki/Division_algorithm#:~:text=Fast%20division%20methods%20start%20with,allow%20using%20fast%20multiplication%20algorithms.
- https://www.codingninjas.com/studio/library/introduction-to-division-algorithm-in-computer-architecture
- https://cs.stackexchange.com/questions/67999/how-fast-is-this-division-algorithm
- Wong, D.C. & Flynn, Michael. (1991). Fast division using accurate quotient approximations to reduce the number of iterations. 191 - 201. 10.1109/ARITH.1991.145559.