JOSHUA MANUEL LOUISE KEMPIS 4 BS CpE October 18, 2022

Project 1 Second Progress Report

Option 1: SRT Division Technique

Version 1 Implementation

The logic behind the first version of the division hardware provides a basic flow of how the inputs are treated. The Divisor and the Dividend are the inputs. The value in the Dividend gets processed and ends up being the Remainder in the final output. The Quotient receives a shift left the operation to its values when certain conditions are met.

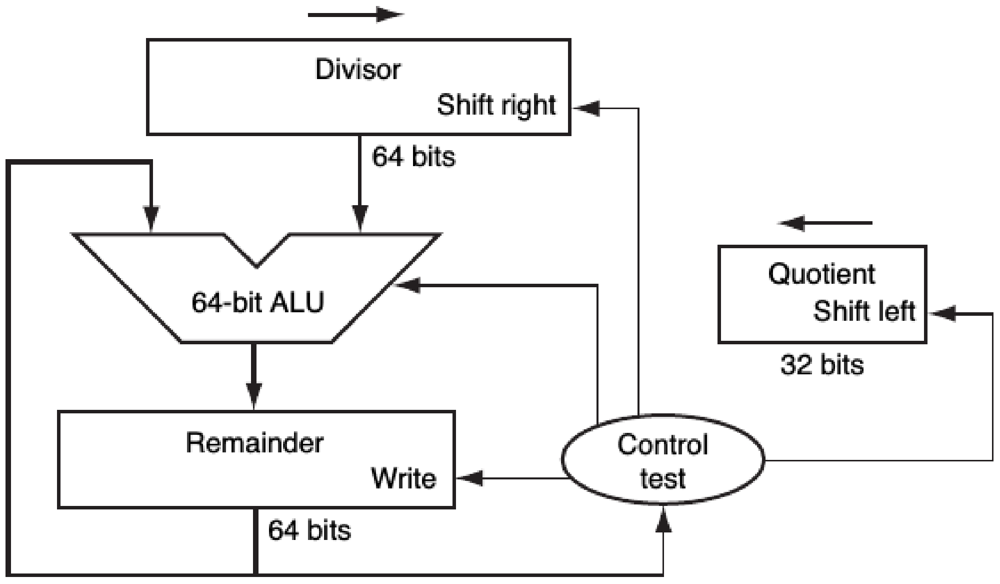


FIGURE 3.9 First version of the division hardware. The Divisor register, ALU, and Remainder register are all 64 bits wide, with only the Quotient register being 32 bits. The 32-bit divisor starts in the left half of the Divisor register and is shifted right 1 bit each iteration. The remainder is initialized with the dividend. Control decides when to shift the Divisor and Quotient registers and when to write the new value into the Remainder register.

The first version (Test #1) has been observed to have a few bugs regarding the division of values with 0 remainders or whenever the value of the divisor is a factor in the dividend. Because of this, a few adjustments to the logic being the ALU portion of the flowchart was made. This was then compiled into a C++ code to observe how these values operate when performing the SRT division under a one-bit quotient at a time.

Test #2 C++ Implementation of Version 1 Process

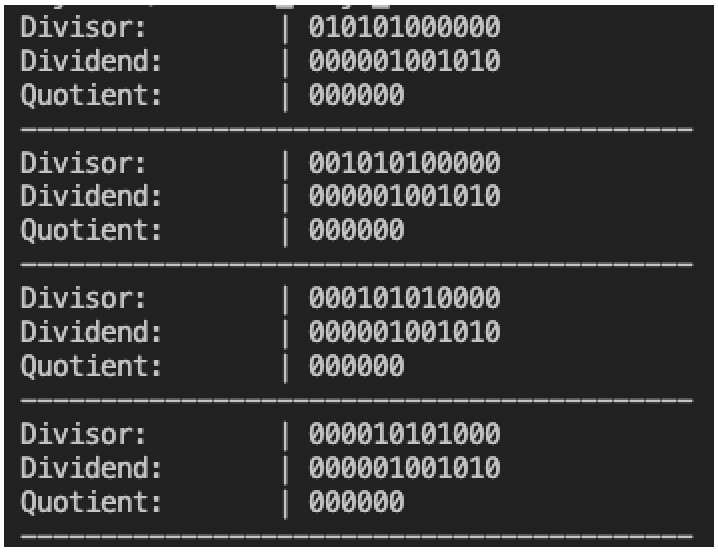
A C++ program was written that implements the division algorithm found in the flowchart. This involves taking inputs and applying them as bitset inputs. These bitset values will be values used in performing the shift, and subtraction operations in the following division algorithm. Given the same examples from the Version 1 implementation, two inputs 21 (divisor) and 74 (dividend) will be used to find the quotient and the remainder following the format of the division algorithm. These inputs are placed in a 12-bit bitset and the quotient will initially be 0 in a 6-bit bitset.

The program will initialize the values as such:

```
(Divisor) 21 = 0b010101’000000
(Dividend/Remainder) 74 = 0b000001’001010
(Quotient) 0 = 0b000000
```

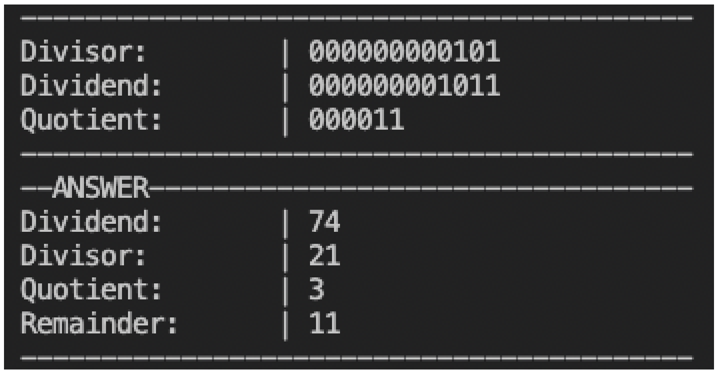
Note that during the initialization process, the value of the divisor 0b010101 was shifted into the upper half of the 12-bit bitset divisor.

INITIALIZATION



```
Divisor:          | 010101000000
Dividend:         | 000001001010
Quotient:         | 000000
------------------------------------------
Divisor:          | 001010100000
Dividend:         | 000001001010
Quotient:         | 000000
------------------------------------------
Divisor:          | 000101010000
Dividend:         | 000001001010
Quotient:         | 000000
------------------------------------------
Divisor:          | 000010101000
Dividend:         | 000001001010
Quotient:         | 000000
------------------------------------------
```

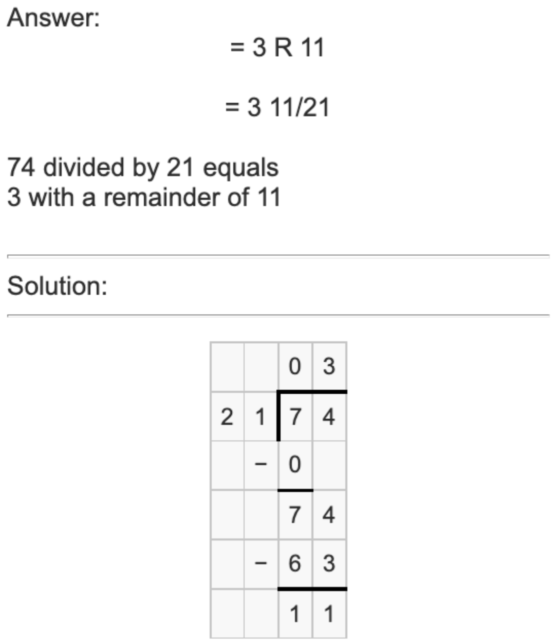
.
.
.



```
------------------------------------------
Divisor:          | 000000000101
Dividend:         | 000000001011
Quotient:         | 000011
------------------------------------------
--ANSWER----------------------------------
Dividend:         | 74
Divisor:          | 21
Quotient:         | 3
Remainder:        | 11
------------------------------------------
```

OUTPUT

Performing the basic structure of the SRT Division provides us with the output of Quotient = 3; Remainder = 11 from the given input values. Comparing it with manual division:



Answer:

= 3 R 11

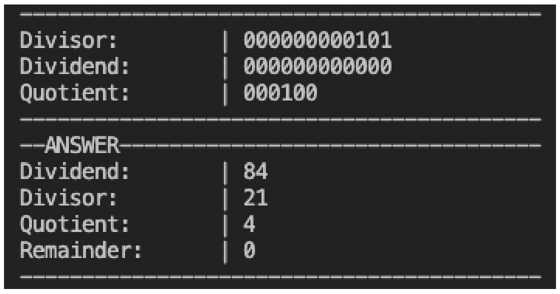
= 3 11/21

74 divided by 21 equals
3 with a remainder of 11

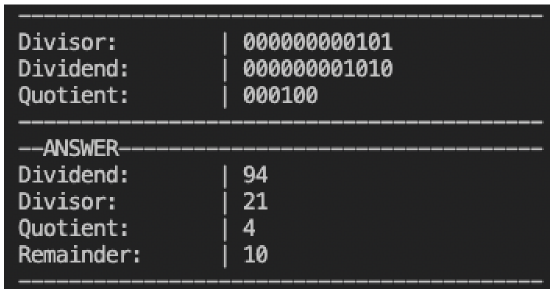
Solution:

		0	3
2	1	7	4
	-	0	
		7	4
	-	6	3
		1	1

With several other Iterations of the code, a successful 6-bit division program was successfully simulated. This program enables division through a series of shifting and basic add/subtract functions. NO “ / “ operator was used in the program.

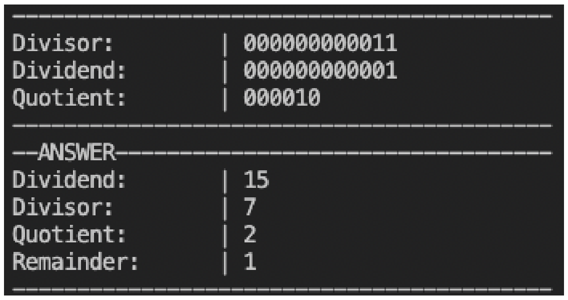


```
------------------------------------------
Divisor:        | 000000000101
Dividend:       | 000000000000
Quotient:       | 000100
------------------------------------------
--ANSWER----------------------------------
Dividend:       | 84
Divisor:        | 21
Quotient:       | 4
Remainder:      | 0
------------------------------------------
```

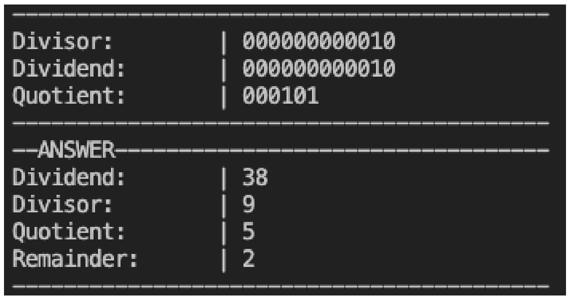


```
------------------------------------------
Divisor:        | 000000000101
Dividend:       | 000000001010
Quotient:       | 000100
------------------------------------------
--ANSWER----------------------------------
Dividend:       | 94
Divisor:        | 21
Quotient:       | 4
Remainder:      | 10
------------------------------------------
```

Error correction still needs to be implemented:



```
------------------------------------------
Divisor:        | 000000000011
Dividend:       | 000000000001
Quotient:       | 000010
------------------------------------------
--ANSWER----------------------------------
Dividend:       | 15
Divisor:        | 7
Quotient:       | 2
Remainder:      | 1
------------------------------------------
```



```
------------------------------------------
Divisor:        | 000000000010
Dividend:       | 000000000010
Quotient:       | 000101
------------------------------------------
--ANSWER----------------------------------
Dividend:       | 38
Divisor:        | 9
Quotient:       | 5
Remainder:      | 2
------------------------------------------
```

Errors encountered which may need adjustments

Conclusion

The project as of the moment is around 40% complete with plenty more needed to be done. Several debugging and error catching needs to be performed in order to implement a proper division algorithm with little to no deviation from the actual arithmetic value.

Task	Description	Status
Project Research	Search for projects on simulating SRT division in C++ to understand how it works and how it may be applied on Verilog.	DONE
Testing Similar Programs	Test programs coded in C++ and check how the lookup table process works	DONE
Preliminary Code Implementation	Begin Coding the basic structure of the project	PARTIAL
Operations	Perform operations based on instructions specified in the project	ONGOING
Output & Feedback	Produce feedback and outputs in the program	TBD
Debugging	Code polishing	TBD