2024 IEEE 9th International Conference for Convergence in Technology (I2CT)
Pune, India. Apr 5-7, 2024

Design of 32-Function- 32-Bit Arithmetic and Logical Unit (ALU)

[1]Mamatha I, [2]Raksha R, [3]Dinesh P
[1]*Department of Electronics and Communication Engineering, Mangalore Institute of Technology & Engineering, Badaga Mijar, Moodbidri, Karnataka, India, 574225*
[2,3] *Department of Electrical and Electronics Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India*
[1]*mamraj78@gmail.com* , [2]*raksha.r1996@gmail.com,* [3]*patildp786@gmail.com,*

2024 IEEE 9th International Conference for Convergence in Technology (I2CT) | 979-8-3503-9447-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/I2CT61223.2024.10543825

Abstract **- An Arithmetic logic Unit (ALU) is used in arithmetic, logical function in all processor. It is also an important subsystem in digital system design. Arithmetic Logic Unit (ALU) is one of the most important components of any system and is used in many appliances like calculators, cell phones, and computers. A 32-bit ALU was designed using Verilog HDL and 32 functions were implemented using 5 select lines. Along with the primitive arithmetic and logical operations, shifting, parity checkers, increment and decrement operations, and code converters are included in this design.**

Index Terms –FPGA, ALU, code converters, arithmetic, logical operations, BCD.

I. INTRODUCTION

An Arithmetic and Logical Unit (ALU) is the fundamental core of all processors which along with the control unit and memory forms the Central Processing Unit (CPU). As the name suggests, an ALU performs arithmetic and logical operations and has two inputs, known as operands and one output. These operations are primitively carried out by a combinational circuitry. The major requirement of any ALU is that it should be designed in such a way that it consumes less power, compute with high speed and have high accuracy while producing the results. This in turn increases the reliability of the designed ALU.

The processor capacity is defined in terms of number of bits an ALU can process at the same time. Initially, 4-bit or 8-bit ALU's were constructed using FPGA's which could perform primitive arithmetic operations such as Addition, Subtraction, Multiplication and Division and logical operations such as AND, OR, NOT, NAND, NOR, EXOR and EXNOR. However, with the advancements in technology, the present day ALU's are expected to handle more number of bits at once and also perform other operations such as parity checking, bitwise operations, comparator and code converter.

Field Programmable Gate Arrays (FPGAs) are integrated circuits with Logical Elements, Input/Output ports and programmable interconnects. Various operations can be programmed using an FPGA, for example, in [1], it is used to implement a MIL-STD 1553B serial bus is used in spacecraft for information exchange between subsystems. The flexibility of an FPGA can be seen in various applications that it can be implemented, as seen in [2], where an efficient 16-bit Multiply and Accumulate (MAC) Unit to perform Vedic Mathematics. This shows that an FPGA is capable of performing the functions that it has been programmed for. A simple 4-bit high-speed ALU is discussed in [2], while [3] discusses an efficient method to implement a 4-bit multiplier. In [4], power reduction with respect to frequency is discussed in detail for an 8-bit ALU over different I/O standards. Also, clock gating techniques is applied in ALU to achieve reduction of clock and dynamic power consumption [5]. A 32-bit extended ALU architecture is proposed in [6]. [7], [9] respectively propose 16 and 32-bit ALU designs using reversible gates. Other work related to ALU's can be found in [10-11].

In this paper, a 32-bit 32-function ALU is designed in Verilog HDL using MODELSIM Tool catering to the needs of the present-day requirements from an ALU. Along with the basic arithmetic and logical operations, this ALU is designed to perform bitwise operations, shifting operations, comparator and has two code converters, binary to grey and binary to BCD.

II. METHODOLOGY

The combinational circuit of the ALU can be designed to perform various functions. Here, 32 functions are proposed for the ALU to perform and hence, 5 select lines are required.

A. Proposed Truth Table

The truth table for the proposed ALU is given in Table I.

B. Operations Performed

1) Arithmetic Operations

One of the most primitive operations expected from an ALU are arithmetic operations.

Here, the ALU is designed to perform 4 operations i.e.,

a) Reset

This operation resets the OUTPUT value to 32'h00000000.

b) Addition

This operation adds the two operands.

c) Subtraction

This operation subtracts the two operands.

TABLE I
PROPOSED TRUTH TABLE

ALU_SELECT_LINE	OPERATION
0	RESET
1	ADDITION
2	SUBTRACTION
3	MULTIPLICATION
4	DIVISION
5	INCREMENT A
6	DECREMENT A
7	LOGICAL_AND
8	LOGICAL_OR
9	LOGICAL_NOT A
10	BITWISE_AND
11	BITWISE_OR
12	BITWISE_NAND
13	BITWISE_NOR
14	BITWISE_EXOR
15	BITWISE_EXNOR
16	PARITY CHECKER A
17	PARITY CHECKER B
18	1'S COMPLIMENT OF A
19	2'S COMPLIMENT OF A
20	1'S COMPLIMENT OF B
21	2'S COMPLIMENT OF B
22	LEFT SHIFT A, B TIMES
23	RIGHT SHIFT A, B TIMES
24	(A>B)?
25	(A==B)?
26	BINARY TO GREY A
27	BINARY TO GREY B
28	GREY TO BINARY A
29	GREY TO BINARY B
30	BINARY TO BCD A
31	BINARY TO BCD B

d) Multiplication

This operation multiplies the two operands and stores the lower 32 bits as the OUTPUT value.

e) Division

This operation divides the two operands and stores the integer quotient as the OUTPUT value.

The additional arithmetic operations include:

f) Increment A

In C syntax, this operation performs A++.

b. Decrement A

In C syntax, this operation performs A--.

2) Logical Operations

The other primitive operation is the logical operation and the ALU is programmed to perform the following logical operations which gives results just like a C programming syntax.

a) Logical AND

If both the operands are greater than 32'h00000000, then, its OUTPUT is 32'h00000001, else, the value stored is 32'h00000000.

b) Logical OR

If either of the operands are greater than 32'h00000000, then, its OUTPUT is 32'h00000001, else, the value stored is 32'h00000000.

c) Logical NOT A

If the value of operand A is greater than 32'h00000000, then its OUTPUT value is 32'h00000000, else, the value stored is 32'h00000001. Note that Logical NOT is a unary operation.

3) Bitwise Operations

a) Bitwise AND

This operation checks the corresponding bit values of both the operands and performs AND operation. It produces a 32-bit result by following the truth table given in Table II.

TABLE II
AND GATE TRUTH TABLE

A[i]	B[i]	ALU_OUTPUT[i]
0	0	0
0	1	0
1	0	0
1	1	1

b) Bitwise OR

This operation checks the corresponding bit values of both the operands and performs OR operation. It produces a 32-bit result by following the truth table given in Table III.

TABLE III
ORGATE TRUTH TABLE

A[i]	B[i]	ALU_OUTPUT[i]
0	0	0
0	1	1
1	0	1
1	1	1

c) Bitwise NAND

This operation checks the corresponding bit values of both the operands and performs NAND operation. It produces a 32-bit result by following the truth table given in Table IV.

TABLE IV
NAND GATE TRUTH TABLE

A[i]	B[i]	ALU_OUTPUT[i]
0	0	1
0	1	1
1	0	1
1	1	0

d) Bitwise NOR

This operation checks the corresponding bit values of both the operands and performs NOR operation. It produces a 32-bit result by following the truth table given in Table V.

TABLE V
NOR GATE TRUTH TABLE

A[i]	B[i]	ALU_OUTPUT[i]
0	0	1
0	1	0
1	0	0
1	1	0

e) Bitwise EXOR

This operation checks the corresponding bit values of both the operands and performs EXOR operation. It produces a 32-bit result by following the truth table given in Table VI.

TABLE VI
EXOR GATE TRUTH TABLE

A[i]	B[i]	ALU_OUTPUT[i]
0	0	0
0	1	1
1	0	1
1	1	0

f) Bitwise EXNOR

This operation checks the corresponding bit values of both the operands and performs EXNOR operation. It produces a 32-bit result by following the truth table given in Table VII.

TABLE VII
PROPOSED TRUTH TABLE

A[i]	B[i]	ALU_OUTPUT[i]
0	0	1
0	1	0
1	0	0
1	1	1

g) Parity Checker A

It performs a Unary EXOR operation with the bit-stream of operand A. If the number of 1's in the bit-stream is even in count, then, it produces an OUTPUT value of 32'h00000000, else, the result is 32'h00000001.

h) Parity Checker B

It performs a Unary EXOR operation with the bit-stream of operand B. If the number of 1's in the bit-stream is even in count, then, it produces an OUTPUT value of 32'h00000000, else, the result is 32'h00000001. The parity checking operation is a part of error detection mechanism, and proves to be an efficient tool when long bit-streams are transmitted from Transmitter end to Receiver end in a communication channel.

4) Compliment Operations

a) 1's Compliment of A

It performs Binary NOT Operation on Operand A.

b) 2's Compliment of A

It performs Binary NOT Operation on Operand A and adds 1 to the result.

c) 1's Compliment of B

It performs Binary NOT Operation on Operand B.

d) 2's Compliment of B

It performs Binary NOT Operation on Operand B and adds 1 to the result.

5) Shifting Operations

a) Left Shift A, by B times

In C syntax, it performs (A<<B).

b) Right Shift A , by B times

In C syntax, it performs (A>>B).

6) Comparator Operations

a) (A>B)?

This operation follows the following pseudocode:

```
if(A>B)
    ALU_OUTPUT= 32'hFFFFFFFF;
else
    ALU_OUTPUT= 32'h00000000;
```

b) (A==B)?

This operation follows the following pseudocode:

```
if(A==B)
    ALU_OUTPUT= 32'hFFFFFFFF;
else
    ALU_OUTPUT= 32'h00000000;
```

7) Code Converters

a) Binary to Gray A

When a long sequence of binary numbers are generated, it may cause an error during the transition from one number to the next. This is eliminated in Gray code since there is only one bit changes its value during any transition between two numbers. Also known as cyclic codes, these codes prove to be useful in applications like error detectors, K-Maps, optical encoders, etc. The steps followed to convert a binary code to a grey code are:

- Gray_A[MSB]= Binary_A[MSB]
- Gray_A[i]=Binary_A[i-1]^Binary_A[i]

b) Binary to Gray B

The same explanation holds good for Operand B,

- Gray_B[Gray]= Binary_B[Gray]
- Gray_B[i]=Binary_B[i-1]^Binary_B[i]

c) Gray to Binary A

The steps to be followed for conversion from Gray to Binary are:

- Binary_A[31]=Gray_A[31]
- Binary_A[i]=Gray_A[31] ^ Gray_A[30]^…..^Gray_A[i]

d) Gray to Binary B

The same explanation holds good for Operand B, i.e.,

- Binary_B[31]=Gray_B[31]
- Binary_B[i]=Gray_B[31] ^ Gray_B[30]^…..^Gray_B[i]

e) Binary to BCD A

Binary Coded Decimal represents a decimal number from 0 to 9 in binary. It typically takes 4 bits to represent one decimal number. Unlike hexadecimal to binary, where 10 or A is represented as 1010, in BCD, it requires 8 bits to represent the same, i.e., 0001 0000. The algorithm used to perform this conversion is called Double Dabble Algorithm which is also known as Shift-And-Add-3 Algorithm. The algorithm follows these steps:

a. Shift the binary number left one bit.
b. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
c. Number of iterations= Number of binary bits.

Therefore, for a 12 bit resultant BCD number, the pseudocode is given in Fig 1.

f) Binary to BCD B

The above steps are followed for Operand B as well.

```
for (i = 0; i < 8; i = i+1)
      begin
        BCD_A = {BCD_A[10:0],bin[7-i]};

        if(i < 7 && BCD_A[3:0] > 4)
          BCD_A[3:0] = BCD_A[3:0] + 3;
        if(i < 7 && BCD_A[7:4] > 4)
          BCD_A[7:4] = BCD_A[7:4] + 3;
        if(i < 7 && BCD_A[11:8] > 4)
          BCD_A[11:8] = BCD_A[11:8] + 3;
      end
    end
```

Fig. 1. Pseudocode for Double Dabble Algorithm

III. Results and Simulations

The operations presented in Table I are coded in Verilog HDL using Behavioral Model of abstraction. This provides the highest abstraction level and follows a syntax similar to that of C Programming. The sample inputs fed are:

A= 32'h0000FF0A
B=32'h0000EEEE.

TABLE VIII
SIMULATED OUTPUTS

OPERATION	ALU_OUTPUT
RESET	32'h00000000
ADDITION	32'h0001EDF8
SUBTRACTION	32'h0000101C
MULTIPLICATION	32'hEE08674C
DIVISION	32'h00000001
INCREMENT A	32'h0000FF0B
DECREMENT A	32'h0000FF09
LOGICAL_AND	32'h00000001
LOGICAL_OR	32'h00000001
LOGICAL_NOT A	32'h00000000
BITWISE_AND	32'h0000EE0A
BITWISE_OR	32'h0000FFEE
BITWISE_NAND	32'hFFFF11F5
BITWISE_NOR	32'hFFFF0011
BITWISE_EXOR	32'h000011E4
BITWISE_EXNOR	32'hFFFFEE1B
PARITY CHECKER A	32'h00000000
PARITY CHECKER B	32'h00000000
1'S COMPLIMENT A	32'hFFFF00F5
2'S COMPLIMENT A	32'hFFFF00F6
1'S COMPLIMENT B	32'hFFFF1111
2'S COMPLIMENT B	32'hFFFF1112
LEFT_SHIFT A, B TIMES	32'h00000000
RIGHT_SHIFT A, B TIMES	32'h00000000
(A>B)?	32'hFFFFFFFF
(A==B)?	32'h00000000
BINARY TO GRAY A	32'h0000808F
BINARY TO GRAY B	32'h00009999
GRAY TO BINARY A	32'h0000AA0C
GRAY TO BINARY B	32'h0000B4B4
BINARY TO BCD A	32'h00065290
BINARY TO BCD B	32'h00061166

The functionality of the design are validated by obtaining the sample results through ModelSim simulator as shown in Fig.2(a)-Fig.2(h) for the various operations.

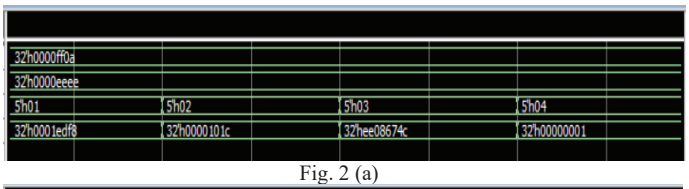

Fig. 2 (a)

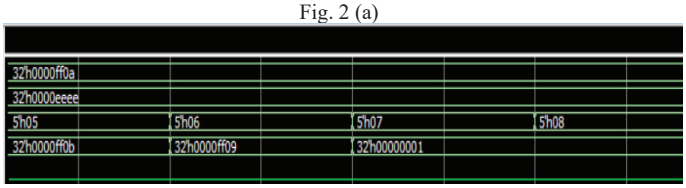

Fig. 2(b).

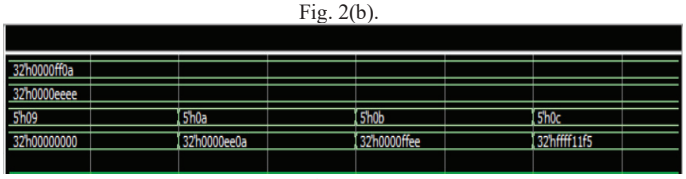

Fig. 2(c)

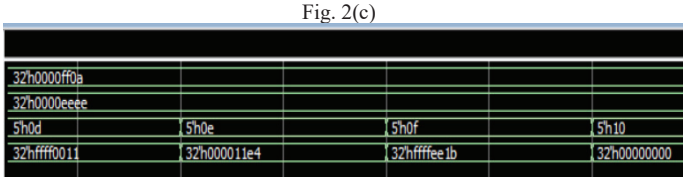

Fig. 2(d)

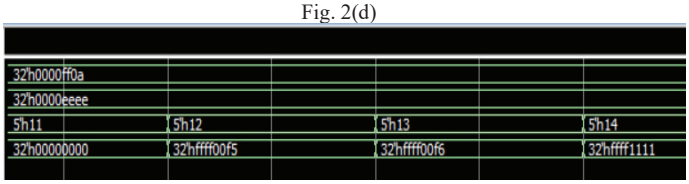

Fig.2(e)

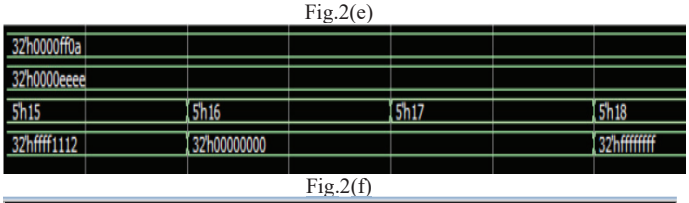

Fig.2(f)

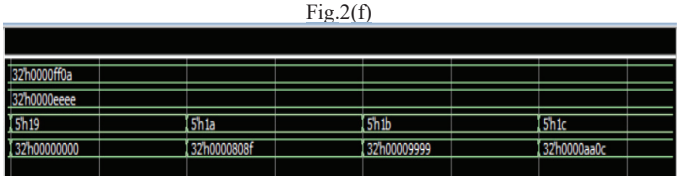

Fig. 2(g)

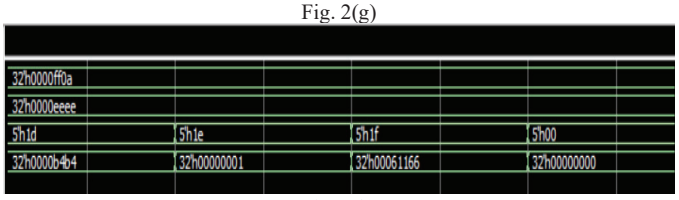

Fig. 2(h)

Fig 2(a)- Fig 2(h). MODELSIM Snapshots of ALU_OUTPUT

IV. Conclusion and Future Scope

In this paper, a 32-bit 32-function ALU was successfully simulated using MODELSIM. Along with the primitive operations, comparators, bitwise operations, shifting operations and compliment operations were performed. Additionally, code converters such as binary to grey and grey to binary for 32 bits have been successfully coded and implemented. An added feature i.e., binary to BCD has also been simulated successfully.

This work can be extended by increasing the number of bits to meet the real-time requirements. Also, floating point and signed numbers can be taken into consideration for computations. The result of n bit multiplication is of 2n bits. However, in our work, on last n bits are stored in the result. This can be avoided. Lastly, fast computing MAC computing ALU's which are required for DSP operations can be implemented.

References

[1] D. P. V. Rao, A. Raja, R. Karthikeyan, K. V. Pal, S. V. Tharun and B. K. Priya, "Design and Implementation of 1553B Bus Controller on FPGA," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp. 418-421.

[2] A. K. Panigrahi, S. Patra, M. Agrawal and S. Satapathy, "Design and Implementation of a high speed 4bit ALU using BASYS3 FPGA Board," 2019 Innovations in Power and Advanced Computing Technologies (i-PACT), Vellore, India, 2019.

[3] P. Martha, N. Kajal, P. Kumari and R. Rahul, "An efficient way of implementing high speed 4-Bit advanced multipliers in FPGA," 2018 2nd International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), Kolkata, 2018, pp. 1-5.

[4] B. Pandey, J. Yadav, Y. K. Singh, R. Kumar and S. Patel, "Energy efficient design and implementation of ALU on 40nm FPGA," 2013 International Conference on Energy Efficient Technologies for Sustainability, Nagercoil, 2013, pp. 45-50.

[5] B. Pandey, J. Yadav, M. Pattanaik and N. Rajoria, "Clock gating based energy efficient ALU design and implementation on FPGA," 2013 International Conference on Energy Efficient Technologies for Sustainability, Nagercoil, 2013, pp. 93-97.

[6] N. Gaur, A. Mehra, D. Kamboj and D. Tyagi, "A novel implementation of 32 bit extended ALU Architecture at 28nm FPGA," 2016 International Conference on Emerging Trends in Communication Technologies (ETCT), Dehradun, 2016, pp. 1-4.

[7] L. Viswanath and , "Design and Analysis of 16 Bit Reversible ALU", Journal of Computer Engineering (IOSRJCE, vol. 1, no. 1, 2012, pp. 46-53.

[8] S. M. Swamynathan and V. Banumathi, "Design and analysis of FPGA based 32 bit ALU using reversible gates," 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE), Karur, 2017, pp. 1-4.

[9] Kumar, T., Pandey, B., Das, T., & Chowdhry, B. S. (2014). Mobile DDR IO standard based high performance energy efficient portable ALU design on FPGA. Wireless Personal Communications, vol.76, no.3, 569-578.

[10] Suhaili, S., & Sidek, O.,"Design and implementation of reconfigurable alu on FPGA", In 3rd International Conference on Electrical & Computer Engineering ICECE, 2004, pp. 28-30.

[11] Kaliamurthy, S., & Sir, M. U. S. "VHDL design of FPGA arithmetic processor", Global Journal of Research In Engineering, vol.11, no.6-F, 2011.