Implementation of an Arithmetic Logic Unit using Area Efficient Carry Look-Ahead Adder and Booth's Multiplier

Sarwagya Chaudhary

Suresh Gyan Vihar University, Dept. of Electronics and Communication Engineering, Jaipur, India

Abstract: ***An arithmetic logic unit acts as the basic building block or cell of a central processing unit of a computer. It is a digital circuit, comprised of the basic electronic components, which is used to perform various arithmetic, logic and integral operations. .The purpose of this work is to propose the design of an 8-bit ALU which supports 4-bit multiplication. The functionalities of the ALU in this study consist of addition, subtraction, increment, decrement, AND, OR, NOT, XOR, NOR, two's complement generation, multiplication. The adder in the ALU is implemented using a Carry Look Ahead adder joined by a ripple carry approach. The design of the multiplier is achieved using the Booth's Algorithm. The proposed ALU can be designed by using Verilog or VHDL and can also be designed on Cadence Virtuoso Platform.***

Keywords: Arithmetic Logic Unit, Booth Multiplier, Carry Look-Ahead Adder, VLSI

1. Introduction

A conventional ALU can be used to perform basic arithmetic and logic operations such as AND, OR, NOT, ADD, Subtract. The ALU takes two operands and performs the desired operation between them. A control signal is used to select the output from the operations that have been performed. The control unit is designed by using a multiplexer which selects the required operation. All the operations are performed in one cycle but only the one that is required in the output is selected by the multiplexer. An ALU does not perform multiplication between two operands. Extra circuitry is required along with the ALU which increases the chip area. In this paper we propose the design of an ALU which supports multiplication. The multiplier is designed using the Booth's algorithm Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers by using two's complement notation. Conventional array multiplier requires a large number of devices. Thus the complexity of the circuit increases. Implementing the booth's algorithm is a more efficient approach. It performs signed multiplication and the circuit complexity is considerably lower than any other type of multiplier circuit. Thus it contributes in building a faster logic circuit with a lower area requirement. The adder unit is constructed by using a Carry look-ahead adder. A conventional ripple-carry adder introduces a considerable amount of delay. The Carry look-ahead adder takes care of the problem and is thus better equipped to perform faster operations.

2. Previous Work

2.1 Adder

There have been various adder circuits used in previous designs and other ALUS to perform the addition operation. Conventional ripple-carry adders have been used while designing logic circuits. When two individual operands are applied as inputs to the adder, it takes some time before giving out the valid output. This happens because each full adder in the combination introduces a certain delay. If one full adder introduces a delay of time 't' then for an 'n' bit circuit the total delay after which Cout is obtained is '(n-1)t'. The delay incurred depends on the size of the operands. Thus when higher bit numbers are used the delay becomes highly unacceptable. [1]

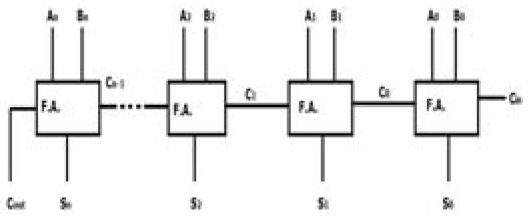


Figure 1: Ripple carry adder

2.2 Multiplier

Multiplication of two four-bit unsigned binary numbers X3X2X1X0 and Y3Y2Y1Y0 by using an array multiplier is shown in Fig. 2. [2]. Full Adder block is the basic building block and total number of full adder blocks required is 4x3=12. Output generated by the full adder block is (a) SUM = X XOR Y XOR CI. (b) Carry Out= X.Y + Y.CI + CI.X. The partial products are realized using an AND Gate.

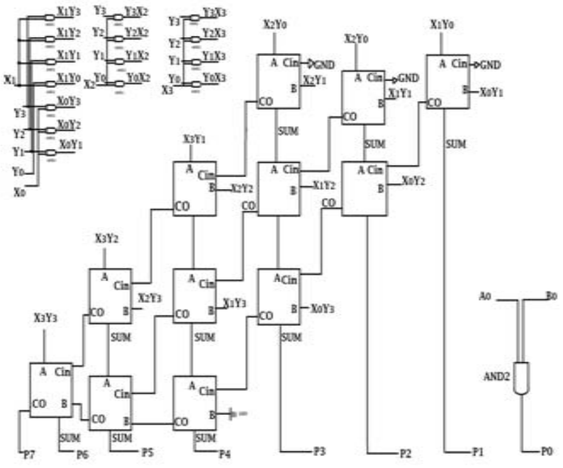


Figure 2: Array multiplier

3. Arithmetic Logic Unit supporting Booth Multiplication

Functionalities performed by the proposed Arithmetic Logic Unit are NOT, NAND, NOR, AND, OR, Exor, Addition, Increment, Two’s complement generation, Subtraction, Decrement, Multiplication. The Inverter, NAND and NOR blocks can be realized in the conventional manner. The inverter circuit consists of a PMOS and an NMOS transistor where the drain of the pmos is connected to the supply voltage and the nmos source is grounded. The inverter output is taken between the pmos source and nmos drain. An n-bit inverter can be designed from the single bit inverter. The NAND and NOR circuits, both, constitute a pull-up and pull-down framework for output generation. The pull-up block in both the logic circuits is made up of PMOS transistors whereas NMOS transistors make up the pull-down block. The drains of the pmos are connected together which, in turn, are connected to the supply voltage. The source of both nmos transistors are connected to the ground. The output is taken between the source of the pull-up block and the drain of the pull-down block. The pull-up network of NAND gate consists of two PMOS transistors connected in parallel with each other and the pull-down consists of two NMOS transistors connected in series with each other. Similarly the NOR circuit’s pull-up network has two PMOS transistors in series and the pull-down network has which are connected to two NMOS transistors in parallel. An n-bit NOT, NAND and NOR block can be erected from these three circuits. [3]. The AND and the OR block can be derived from the NAND and NOR blocks by utilizing the inverter. This prevents hardware complexity thus reducing the total Area thereby making the ALU more efficient.

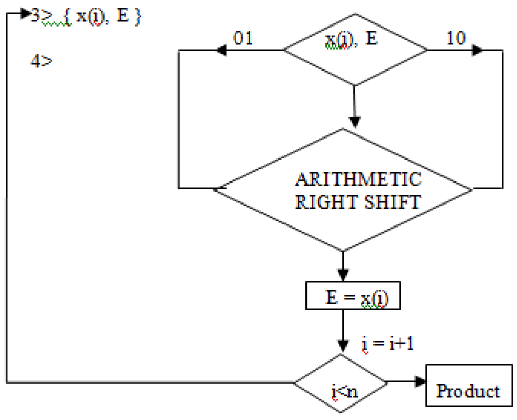
3.1 Adder

In digital adders the speed of addition is limited by the time required to propagate the carry signal through the adder. In an elementary adder the generation of sum for each bit position takes place in a sequentially only after the preceding bit position has been summed and a carry is propagated into the next position. A carry look-ahead helps us in eliminating the delay caused by the propagation of the Carry signal in a binary adder. The delay caused in the addition operation is because of the carry signal. The expression for the carry signal of a ripple carry adder is [4]
Ci+1 = AiBi + AiCi + BiCi (1)
This can be written as
Ci+1 = AiBi + Ci(Ai + Bi) (2)
Ci+1 = Gi + PiCi (3)
The value of function Gi is equal to 1 when both Ai and Bi are 1. Gi is independent of Ci and is guaranteed to generate an output so it is known as the generate function. The value of function Pi is equal to 1 when either Ai or Bi or both are 1. A carry out is produced when Ci is equal to 1. This leads to the carry in of 1 propagating through the ith stage. Thus Pi is termed as the propagate function. On expansion of the expression for the carry signal we get
Ci+1 = Gi + Pi(Gi-1 + Pi-1Ci-1) (4)
Ci+1 + Gi + PiGi-1 + PiPi-1Ci-1 (5)
If we stretch this expansion to the 0th stage then we get the following expression.
Ci+1 = Gi + PiGi-1 + PiPi-1Gi-2 + ….+ PiPi-1..P2P1G0 +PiPi-1…P1P0C0 (6)

This expression is the representation of a two level AND-OR combination which can calculate the value of Ci+1 at a much higher speed. It defines the functioning of the carry look-ahead adder.

3.2 Multiplier

Latency, area, design complexity, throughput are some of the factors which help us to choose. The booth multiplier algorithm accelerates multiplication [5]. Let’s take an example where we see the results of the algorithm for signed and unsigned multiplication. Considering two operands x and y which are the multiplier and the multiplicand respectively. The values of x and y are varied in three cases and the result of the multiplication is checked using the algorithm.
Case 1 x = 3, y = 5. Product = +15
Case 2 x = -3, y = -5. Product = +15
Case 3 x = -3, y = 5. Product = -15
1> Read x, y
2> i = 0, z = 0, E = 0



E = x(i)
i = i+1
i<n Product
Three parameters will remain constant for all the cases.
i = 0, E = 0, z = 00000000
E is he bit which demonstrates multiplication. Z stores the value after each loop.

Case 1(Unsigned multiplication)
x = (3)10, y = (5)10.
Product, P = x.y =(15)10
x = (0011)2, y = (0101)2
let y1 be the two’s complement of y, therefore y1 = -y = therefore y1 = -y = (1011)2

Step 1. {x(0),E} = {1,0}
z = z – y = z + y1
The bits of y or y1 are added from the MSB of z.
00000000
+1011
10110000
After Arithmetic Right shift, z = 11011000
i = i + 1= 0 + 1 = 1
E = x(0) = 1
Step 2. {x(1),E} = {1,1}After arithmetic right shift , z = 11101100
i = i + 1 = 1 + 1 = 2
E = x(1) = 1

Step 3. {x(2),e} = {0,1}
z = z + y
11101100
+ 0101
00111100

After Arithmetic Right shift, z = 00011110
i = i + 1 = 2 + 1 = 3
E = x(2) = 0

Step 4. {x(3),E} = {0,0}
Result after arithmetic right shift
z = (00001111)2
z = (15)10, which is equal to the product of the two operands.

Case 2 (Signed Multiplication)
x = (-3)10, y = (-5)10.
Product, P = x.y = (15)10
x = (1101)2, y = (1011)2
let y1 be the two's complement of y
therefore y1 = -y = (0101)2

Step 1. {x(0),E} = {1,0}
z = z – y = z + y1
The bits of y or y1 are added from the MSB of z.
00000000
+0101
01010000
After Arithmetic Right shift, z = 00101000
i = i + 1= 0 + 1 = 1
E = x(0) = 1

Step 2. {x(1),E} = {0,1}
z = z + y
00101000
+ 1011
110 11000

Result after arithmetic right shift
z = 11101100

i = i + 1 = 1 + 1 = 2
E = x(1) = 0

Step 3. {x(2),e} = {1,0}
z = z + y
11101100
+ 1011
00011100

After Arithmetic Right shift, z = 00011110
i = i + 1 = 2 + 1 = 3
E = x(2) = 1

Step 4. { x(3),E} = {1,1}
Result after arithmetic right shift
z = (00001111)2
z = (15)10, which is equal to the product of the two operands.

Case 3 (Signed multiplication)
x = (-3)10, y = (5)10.
Product, P = x.y = (-15)10
x = (1101)2, y = (0101)2
let y1 be the two's complement of y,
therefore y1 = -y = (1011)2

Step 1. {x(0),E} = {1,0}
z = z – y = z + y1
The bits of y or y1 are added from the MSB of z.
00000000
+1011
10110000
After Arithmetic Right shift, z = 11011000

i = i + 1= 0 + 1 = 1
E = x(0) = 1

Step 2. {x(1),E} = {0,1}
z = z + y
11011000
+ 0101
001 01000

Result after arithmetic right shift
z = 00010100

i = i + 1 = 1 + 1 = 2
E = x(1) = 0

Step 3. {x(2),e} = {1,0}
z = z – y = z + y1
00010100
+ 1011
11000100

After Arithmetic Right shift, z = 11100010
i = i + 1 = 2 + 1 = 3
E = x(2) = 1

Step 4. { x(3),E} = {1,1}
Result after arithmetic right shift
z = (11110001)2
z = (-15)10, which is equal to the product of the two operands.

Thus the booth's multiplication algorithm helps us perform multiplication much faster than any other conventional multiplier. It reduces hardware complexity. It requires fewer devices than an array multiplier.

3.4 Subtractor

We construct the subtractor circuit by implementing the carry look-ahead adder and an inverter. For an n-bit number a complement wrt 2N is what the two's complement means. The result obtained after we subract the number by 2N . The two's complement exhibits the behaviour of the negative of the original number. [6]

3.5 Incrementer and Decrementer unit

The incrementer and decrementer units are built using the adder and subtractor units. For incrementer one of the addends is given the input value logic '1'. For decrementer unit the subtrahend of the subtractor is provided with logic '1' value.

3.6 Two's complement generation

The two's complement generator circuit is achieved by using an inverter and an incrementer. The input operand is passed through an inverter which inverts its digits. The output of the inverter is given as input to the incrementer. The incrementer adds logic 1 to the inverted bits. Thus the output generated is the two's complement of the input bit.
Fig. 3 shows the block diagram for the proposed ALU. The multiplexer is used to select the desired output. All operations are performed in a single cycle but only the required operation is selected by the multiplexer. Figure 3 shows the block diagram of the proposed arithmetic logic unit.

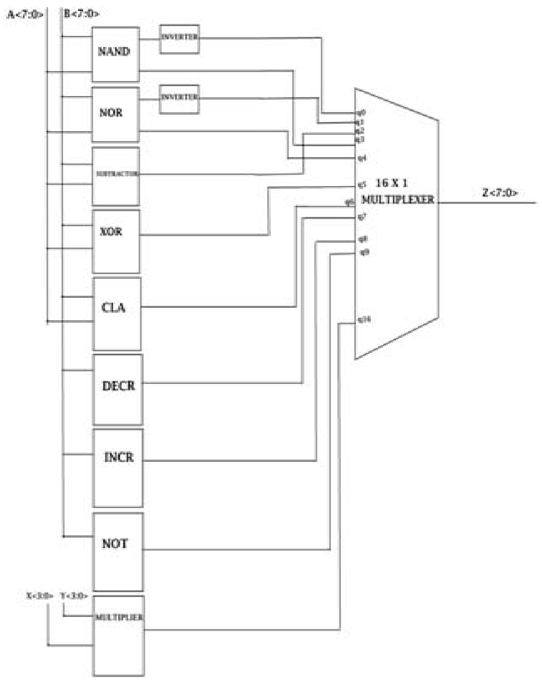


Figure 3: Block diagram of ALU

Operation	*Opcode*	*Operation*	*Opcode*
AND	0	SUM	110
OR	1	Decrement	111
Subtract	10	Increment	1000
NAND	11	NOT	1001
NOR	10	Two's Compliment	1010
XOR	101	Multiply	1111

Table 1: Operations and their Opcodes

4. Conclusion

The need for smaller-sized integrated circuits is evolving at a very fast rate. The VLSI field requires logic circuits with reduced complexity and circuit area and a higher processing rate. The proposed ALU consists of an adder circuit which provides lesser delay than a conventional adder. The Booth's multiplication algorithm reduces the hardware complexity and provides the output faster than other multipliers. The AND and OR units are realized using NAND and NOR blocks thereby minimizing the area required by the logic circuit. Less area and faster operation makes the ALU ideal for an efficient logic circuit. It can be coded using Verilog or VHDL. We can also design it using the Cadence Virtuoso platform.

References

[1] Akbar Bemana, A new Simulation if a 16 bit Ripple-Carry Adder and a 16-bit skip Carry Adder, International Journal of Scientific and Technology Research, Volume 1, Issue 2, March 2012

[2] U.Sreenivasulu & T.Venkata Sridhar, Implementation of a four bit ALU using Low Power and Area efficient Carry Select Adder, International Conference on Electronics and Communication Engineering, 20th May 2012, Bangalore

[3] Neil H.E Weste, Cmos VLSI Design: A circuit and Systems Perspective(Pearson Education India, 1 Sep 2006)

[4] Mousam Halder, Jagannath Samanta, Performance Analysis of High Speed Low Power Carry-Look Ahead Adder Using Different Logic Styles, International Journal of Soft Computing and Engineering, Vol 2, Issue 6 Jan-2013

[5] Fazal Noorbasha, VLSI Implementation of 200 MHz, 8 bit, 90nm CMOS Arithmetic and Logical Unit Processor, Leonardo Electronic Journal of Practices and Technologies 19 July-December 2011.

[6] Michael Andrew Lai, Arithmetic Units for a high performance Digital signal processor, University of California, Davis, 2002