International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 9, September- 2014

IEEE 754 Floating Point Multiplier using Carry Save Adder and Modified Booth Multiplier

K. Lavanya
M.Tech (EMBEDDED SYSTEMS)
Electronics and Communication Engineering,
SR. Engineering, college,
Warangal, India.

K. Navatha
Assistant Professor
Electronics And Communication Engineering
SR. Engineering, college,
Warangal, India

***Abstract*-This paper describes design and implementation of IEEE-754 single precision floating point using carry save adder (CSA) and modified booth algorithm(MBA) and design is compared in terms of speed, area and power consumption. Adders were used to increase the speed; multipliers will help in reducing power, area and also reduces generation of partial products and speed up operation which helps in producing accurate results. Floating point design involves detection of exceptions and trap overflow and underflow. The algorithm is implemented in Verilog HDL and simulation results observed by using Xilinx ISE8.1i. Verification has been implemented on Spartan 3E.**

***Keywords*- Carry save adder, floating point multiplier, modified booth algorithm, Verilog HDL**.

I. INTRODUCTION

Floating point multiplier (FPU) is one of the most common elements in digital applications such as Digital signal processing, control units and image processing because of its fast and rapid execution of arithmetic operations in a circuit. Floating point (FP) acts as co-processor that comprises arithmetic operations, adders and multiplications. Floating point unit is used in all computer processors which play an important role in processor to perform math's coprocessor which is designed to carry out operations on floating point numbers, and it plays a very important role in high performance applications such as microprocessor, digital signal processing (DSP) [1] and FIR filters. Math processor will help in increasing speed and capability of a processor. In many applications programmers has to put effort for maintaining proper scaling of numbers to maintain precision and also to avoid overflow for such type of applications we use floating point which is specifically used to control the desirability of dynamic ranges of fixed point operations.

Now a day in many general purpose computer architectures, one or more floating point units (FPU) are integrated with central processing unit (CPU). Floating point unit is an execution unit which is mainly designed to perform mathematical operations like addition, subtraction and multiplications and it also performs various transcendental functions such as trigonometric or exponential calculations and these operations are done through software library routines in many advanced processors

In this paper single precision floating point multiplier unit is implemented using pipelining and without pipelining method. The multiplier and adder units are implemented using modified booth multiplier and carry save adder (CSA) [7]. Carry save adder is one of the fastest adder used in digital circuits increase speed and reduces area, power, and delay modified booth multiplier will help in increasing speed and reduce generation of partial products by this it reduce complexity of multiplication. Floating point (FP) using pipelining method will improve continuous clock cycle results and reduces consumption, latency and increases speed.

II. IEEE754 FLOATING POINT UNIT

2.1 IEEE-74FLOATINGPOINT STANDARDS

IEEE-754 standard is s standard representation established by IEEE and widely used standard for floating point computation. Single precession floating point represents computer format and it occupies 32-bits in a computer memory and represents a wide range of values by using a floating point. In IEEE 754-2008, the 32-bit with base 2 formats [2][6] is referred as single precision or binary 32.

The standard basically has four types and they are

- *Arithmetic format: it is a* set of binary and decimal floating point numbers which has finite numbers that contains signed zero, subnormal and infinite numbers and special value called" not a number"(NaN).
- *Interchange format*: it is a bit string or encodings that are mainly used to exchange floating point data in a compact and efficient form.
- *Rounding rules:* properties should be satisfied while doing arithmetic operations and conversion of any numbers on arithmetic formats.
- *Exception handling:* It indicates any exceptional conditions like overflow, underflow etc., while doing operations.

III. IEEE-754 FLOATING POINT FORMAT

IEEE is a technically used standard followed by many hardware and software implementation. Single precision floating point standard represents 32bits (4bytes) in computer number format, most significant bit will start from left this single precision is having three basic components sign, exponent and mantissa sign bit width is 1 bit, exponent of width is 8bits and mantissa of 24 bits out of which 23 bits are explicitly stored [6] and 1 bit is implicitly stored, sign bit is used represent sign of floating point number where sign(s=o positive numbers=1 negative number)

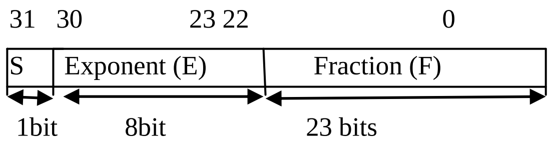


Fig 3.1 32 Bit single precession floating point

The Number representation of single precession [9].
Value = (-1) s *2E-127 * 1.M (normalized) when E>0
S= sign bit (0 for positive, 1 for negative)
e=unbiased exponent; e=E-127(bias)
E=exponent it can be signed or unsigned integer, 8-bit signed integer it ranges fromE_{min} = - 128 to E_{max} =127(2'compliment) and for unsigned it ranges from e_{min} =0 to E_{max} =255 which is the accepted biased form in IEEE 754 single precession. In this exponent with value 127 represents actual zero. The true mantissa will have 23 fraction bits to the right of binary point and implicit to left of binary point with value 1unless exponent bit is stored with zeros. In memory we can see only 23 fraction bits in mantissa even though it has 24bit.If E>0 and E<255 and has 1 in MSB of significant then that number is said to be normalized number.

3.1 FLOATING POINT MULTIPLICATION ALGORTHIM

Floating point multiplications are used in many digital circuits and signal processing computations, floating point multiplication design involves overflow and underflow[1][5][9]. In this normalized floating point number have the form of Z=X*Y= (- 1S) * 2 (E - *Bias*) * (1.M).
To multiply two floating point numbers it involves following steps [8]:

- If one or both operand is equal to zero I results zero otherwise
- Multiplying the significant bits in mantissa(X*Y)
- Adding the exponent of two numbers and then the result will be subtracted from bias (E1+E2-Bias).
- Obtains the sign bit (X XOR Y).
- Normalizing the result if needed, by shifting mantissa to right and incrementing exponent result.
- Rounding the results to the allowed mantissa bits.
- Checking for overflow and underflow

1. Overflow occurs when result is greater than maximum exponent.
2. Underflow, when result is smaller than minimum exponent.

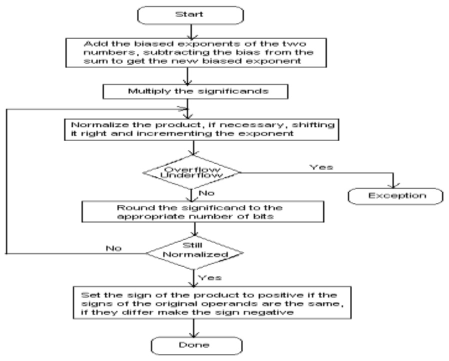

Start
Add the biased exponents of the two numbers, subtracting the bias from the sum to get the new biased exponent
Multiply the significands
Normalize the product, if necessary, shifting it right and incrementing the exponent
Overflow Underflow
Yes
Exception
No
Round the significand to the appropriate number of bits
No
Still Normalized
Yes
Set the sign of the product to positive if the signs of the original operands are the same, if they differ make the sign negative
Done

Fig 3. 2. Floating point multiplication flow chart.

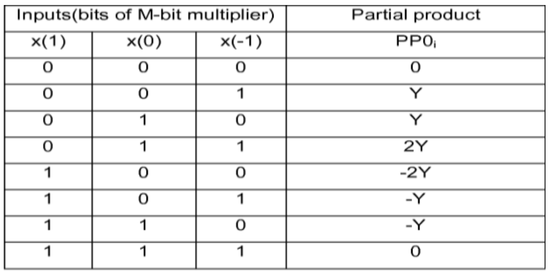
3.1.1. Radix-4Modified booth encoder

Booth algorithm is an important algorithm that is used to implement signed number multiplication, which treats both positive and negative numbers uniformly [3]. The disadvantages that occurred in booth algorithm can be overcome by using this Modified booth encoder technique, which was proposed by O.L.Macsorely.in modified booth it is possible to reduce partial products by half i.e. for n-multiplier bits we get n/2+1 partial products[4]. Modified booth is twice faster than normal booth multiplier and one of the efficient way by which it can reduce partial products by grouping two consecutive bits and uses two operands to perform signed bit. The booth encoded operand is called multiplier and other operand is multiplicand. In this technique it compares 3bits at a time by using overlapping technique, grouping of bits are taken from LSB, and first block use only two bits and for third bit it will assume as zero [3]. The functional operation of radix-4 modified booth is shown in table1.

Radix-4 booth algorithm is shown below [3]:

1. Extend the sign bit 1 position if necessary to ensure that n is even.
2. Assume a zero at right end of LSB of a multiplier
3. According to the values of each block taken in multiplier will take partial products by 0, +y,-y, +2y,-2y.

Table1: Radix4 Modified Booth Encoder



Inputs(bits of M-bit multiplier)			Partial product
x(1)	x(0)	x(-1)	PP0$_i$
0	0	0	0
0	0	1	Y
0	1	0	Y
0	1	1	2Y
1	0	0	-2Y
1	0	1	-Y
1	1	0	-Y
1	1	1	0

3.2 *Carry save Adder*

Carry save adder (CSA) design [7] is used in high speed multioperand adder. Carry save adder is a standalone full adder and carries out number of partial products, without any carry chain operations. Carry save adder is mainly used to calculate partial products that are generated by integer multipliers. By using carry save adder it is possible to reduce delay. In carry save adder N-bits contain N-disjoint full adders [3] in which it contains single sum and carry which depends on inputs. Carry save adder is known as (3:2) counter or compressor that is it takes 3 inputs and produces 2outputs without any carry chain. Carry save adder not only solve adder problems but it also solves different problems that occurs in a digital system, carry save adder can compress n-bits by taking any type of adder in place of full adder we take ripple carry adder as ripple adder is cascaded with full adders we use ripple adder as sub block in carry save adder.

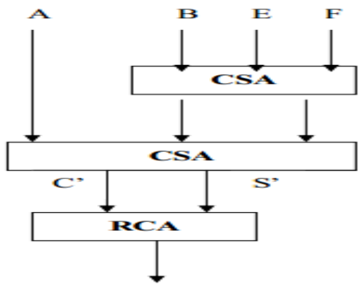


Fig.3.3 Carry save Adder

3.2.1 *Ripple Carry Adder*

Another known adder is ripple carry adder (RCA) and it is composed of cascaded full adders for n-bit adder as shown in above fig .ripple carry adder is constructed with the help of full adder blocks and these adders are arranged in series. The carry out of one stage is fed directly to the carry-in of the next stage. For an n-bit parallel adder it requires n full adders. It is used to obtain the final sum and carry output by adding partial products. In this adder at each stage input (c_{in})is carry output (c_{out})of previous adder stages[7] i.e., each carry bit ripples to next full adder and is called as ripple carry adder(RCA)

Logic equations:

G_i=carry generation function

P_i=carry propagation function

$G_i = X_iY_i$ $\qquad$ $P_i = X_iXORY_i$

$C_{i+1} = G + P_ic_i$ $\qquad$ $S_i = P_iXORC_i$

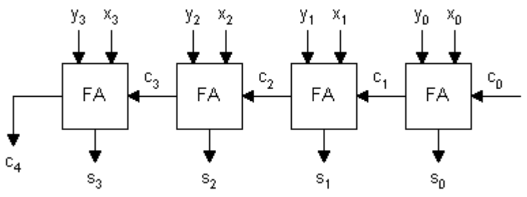


Fig 3.4.Rripple carry adder

3.3.*Floating point multiplier using pipeling*

pipeline technique will help a circuit to operate at higher clock rates by dividing lage task into small subtasks without overlapping,it helps in reducing the complexity of a circuit.pipeline multiplier[10] is suitable to perform higher arthimatic operations such as digital singals.Floating point multiplier architecture is mainly used to reduce delay.pipelined architecture provides fast response and also consumes more area,pipeline produce a flexible data path which is easy o modify then fixed datapath and reduces maintenance time,and it also reduces clock period and increases latency[10].In this floating point multipler using pipeline use modified booth multiplier of 26bits and it is divide into 3 stage pipeline.in this pipeline stage one represent check zero module stage two will check for sign bits and stage three represents normalizer and finall results.

To multiply two operand in pipeline multiplier it involves following steps

1. Check for zero module that is operands are checked whether zero_flag has zero or one if it as zero then it will initiallise to all stages and resultant output will be zero,if zero_flag is one in any operand then input will assign IEEE754 format and connected to check sign,add exponent and mantissa
2. Check sign will represent whether operand bits used in pipeline multiplier hast positive sign or negative sign bits i.e.,if both operands have same sign bit then it represent positive sign otherwise negative sign.
3. Add exponent will show whether module is in active state or not (if zero_flag is set then active otherwise zero will be passed to next state and set zero_flag to 0)
4. Multiply mantissa will check for zero_flag.if zero is set then no caluculations will be done.if mantissa is set to 0 then multiplier operation will be done, if mantissa has 1 hen it shows that execution is done.
5. Normalizer will check for overflow and underflow.

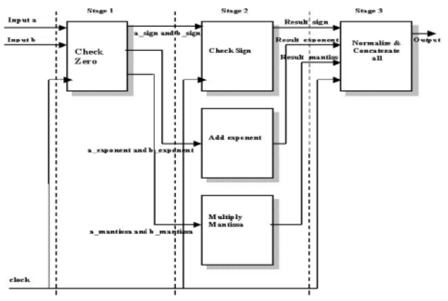


Fig 3.5.Stage Pipeline Multiplier

IV. SIMULATION RESULTS

Simulation results shown below are implemented using Verilog HDL using Xilinx ISE 8.1i.and implemented in spatarn3E.

TABLE.2.SYNTHESIS REPORT FOR CARRY SAVE ADDER

Types of Parameters	*Used and Available*
Number of slices	10 out of 960
Number of 4 input LUTs	17 out of 1920
Number of bonded IOs	25 out of 66
XORs	12
IOs	26
Delay	13.121ns

TABLE.3.SYNTHESIS REPORT FOR RADIX4 MODIFIED BOOTH MULTIPLIER

Types of Parameters	*Used*
Number of slices	1244
Number of 4 input LUTs	2332
Bonded IOs	105
Adders	53
Registers	52
Multiplexers	13
Flip-flops	52
IO Buffers	104
OFFSET	107.637ns
Clock buffer	1

TABLE.4. SYNTHESIS REPORT FOR FLOATING POINT WITH PIPELINING AND WITHOUT PIPELINING

Types of parameters	*Floating point without pipeline*	*Floating point with pipeline*
Number of slices	1921	1099
Number of 4 input LUTSs	3629	2033
Number of bonded IOs	99	99
IOs	99	99
IO buffers	98	98
Adders	53	53
Flip-flops	193	193
Registers	193	199
Comparators	4	4
Multiplexers	13	13
Xors	25	25
Clock period	163.749ns	6.625ns
Clock frequency	6.107Mhz	159.899Mhz
delay	81.874	6.254ns

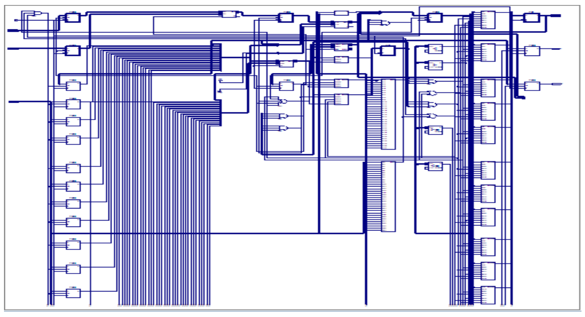


Fig.4.1. RTL Schematic of Floating Point Multiplier without Pipelining

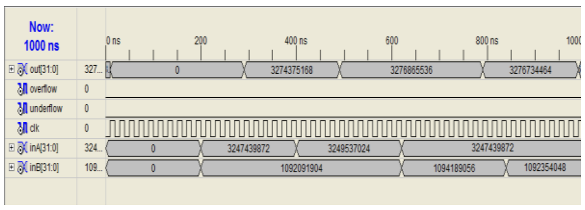


Fig.4.2. Simulation Results for Floating Point Multiplier without Pipeline

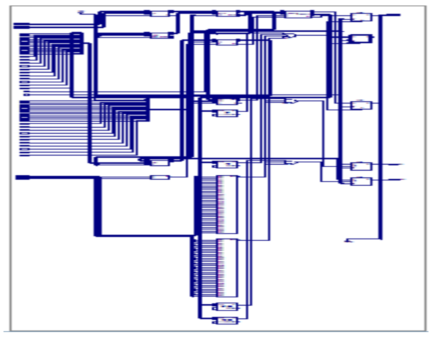


Fig.4.3.RTL of Floating Point Pipelined Multiplier.

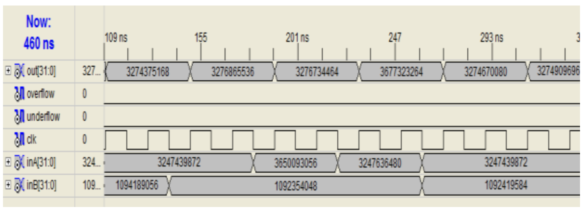


Fig.4.4.Simulation Results for Floating Point using pipeline

V. CONCLUSION

In this paper we implemented single precision floating point multiplier with and without pipelining using Xilinx 8.1i and is implemented in Spartan 3E,and observed that floating point multiplier with pipeline is having high speed performance. It has less delay and less clock period then floating point multiplier without pipelining and also pipelining has reduced complexity and offers high throughput with low latency. It can be implemented in any type of processor and is also suitable for system on chip(Soc) prototyping.

REFERENCES

1. P. Belanovi´c and M. Leeser. A Library of Parameterized Floating Point Modules and Their Use. In Proceedings, International Conference on Field Programmable Logic and Applications, Montpelier, France, Aug. 2002.
2. IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.
3. Radix-4 Encoder Booth Multiplier
4. A High-Speed Multiplication Algorithm Using Modified Partial Product Reduction Tree P. Asadee, International Journal of Electrical and Electronics Engineering, **4**: 4, (2010).
5. D. Sangwan and M. K. Yadav, “Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic”, in International Journal of Electronics Engineering, **(2010)**, pp. 197-203.
6. An ANSI/ IEEE Standard for Radix-Independent Floating-Point Arithmetic, Technical Committee on microprocessor of IEEE computer society, October, 1987.
7. P.Sreenivasulu, Dr. K.Srinivasa Rao, Malla Reddy and Dr.A.Vinay Babu ―Energy and Area efficient Carry Select Adder on a reconfigurable hardware‖ International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 ,Vol. 2, Issue 2, Mar-Apr 2012pp.436-440.
8. IEEE Floating Point Representation of Real Number, Fundamentals of Computer Science. Link: http://www.math.grin.edu/~stone/courses/fundamentals/ieee- reals.html
9. L. Louca, T. A. Cook and W. H. Johnson, “Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs”, Proceedings of 83rd IEEE Symposium on FPGAs for Custom Computing Machines (FCCM‟96), **(1996)**, pp. 107–116.
10. Amine Bermark, Guixuan Liang and Qingzheng, “A High-speed 32-bit Signed/ Unsigned Pipelined Multiplier”, Department of Electronics and Computer Engineering, Honkong University of Science and Technology, Hong Kong, China.