© 2015 IJSRSET | Volume 1 | Issue 6 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Implementation of Optimized Floating Point Arithmetic Unit on Reconfigurable Logic

Sonam Pardhi, Nitesh Dodkey
Department of Electronics and Communication Engineering, Surbhi group of Institute, Madhya Pradesh, India

ABSTRACT

This paper presents the FPGA implementation of a Decimal Floating Point (DFP) arithmetic unit. The design performs addition, subtraction and multiplication on 64-bit operands that use the IEEE 754-2008 DPD encoding of DFP numbers. The design uses an equal bypass adder, this adder reduces the power consumption and it also reduces the delay by reducing the gate count. The design also uses barrel shifter instead of sequential shifter to reduce delay. Also 64 bit parallel BCD multiplier is used to perform fixed point multiplication. The proposed DFP arithmetic unit supports operations on the decimal64 format and it is easily extendable for the decimal128 format.

Keywords : Floating point addition, Floating point multiplication, Floating point subtraction, FPGA, Delay, Area overhead, IEEE P754-2008

I. INTRODUCTION

The binary floating point (BFP) arithmetic has certain flaws namely; it cannot provide correct decimal rounding and cannot precisely represent some decimal fractions such as 0.001, 0.0475 etc [1]. There are many applications where a precision is required such as billing, insurance, currency conversion, banking and some scientific applications. European Union requires that currency conversion to and from EURO is to be calculated to six decimal digits [2]. One study estimates that errors generating from BFP arithmetic can sum up to a yearly billing of over dollar 5 million for a large billing organization [3]. Therefore decimal floating point (DFP) arithmetic becomes very important in many current and future applications as it has ability to represent decimal fractions precisely. DFP arithmetic also has the ability to provide correct decimal rounding that will mimic the manual rounding.

Applications which cannot tolerate errors generating from BFP arithmetic, these application use software platforms to perform DFP arithmetic [1]. There are many software packages which are available for example: the java BigDecimal library [5] and IBM's decNumber library [4]. Also Intel published results for a decimal arithmetic library which uses Binary integer decimal (BID) encoding. These software packages are good enough for current applications, but trends towards globalization and e-commerce are increasing, so faster response of these systems is required. Software designs to these systems may be inadequate with the increasing performance demands of future systems. So hardware implementation of these systems is the need of the hour.

In 2008, the IEEE 754-1985 floating point standard has been revised and the new standard called the IEEE 754-2008 floating point standard was setup [6], which includes specifications for DFP formats, encoding and operations. The IEEE 754-2008 standard includes an encoding format for DFP numbers in which the significand and the exponent (and the payloads of NaNs) can be encoded in two ways namely; binary encoding and decimal encoding. [7]

Both the encoding formats break a number into a sign bit s, an exponent E, and a p-digit significand c. The value encoded is $(-1)^s \times 10^E \times c$. In both formats the range of possible values is identical, but the significand c is encoded differently. In the decimal encoding, it is encoded as a series of p decimal digits using the densely packed decimal encoding (DPD). In the binary encoding also known as binary integer decimal (BID) encoding, it is encoded as a binary number.

In this paper a floating point arithmetic unit is proposed. This floating point arithmetic unit is IEEE P754 – 2008 complaint and based on densely packed decimal (DPD) encoding for DFP arithmetic. The proposed floating point arithmetic unit uses low power equal bypass adder to reduce the power consumption of the design. A parallel 64 bit (16 x 16 digits) BCD multiplier is used to reduce delay.

II. METHODS AND MATERIAL

A. Decimal Floating Point Representation

In IEEE 754-2008, the value of a finite DFP number with an integer significand is

$$V = (-1)^{s} \times 10^{q} \times c$$

where ‘S’ is the sign, ‘q’ is the unbiased exponent, and ‘C’ is the significand. The precision or the length of the significand is denoted as ‘p’, which is equal to 7, l6, or 34 digits, for decimal32, decimal64, or decimall28, respectively. Figure 1 shows the double precision decimal64 Decimal Floating Point format.

Sign (S)	Combination (w + 5) (G)	Trailing Significand (C)
1 Bit	13 bits	50 bits

Figure 1: Decimal 64 – Decimal floating point format DPD encoded

The l-bit Sign Field, S indicates the sign of a number. The (w+5)-bit Combination Field, G provides the most significand digit (MSD) of the significand and a non-negative biased exponent, E such that E = q + bias. The exponent is almost always encoded in binary. The G Field also indicates special values, such as Not-a-Number (NaN) and infinity (00). The remaining digits of the significand are specified in the t-bit Trailing Significand Field, T. Table 1 shows the combination field.

Table 1: Combination Field

Number type	**Combination field**	**Exponents Bits**	**Significand MSD**
finite	a b c d e	a b	0 c d e
finite	11 a b c	a b	1 0 0 e
infinite	1 1 1 1 0	. .	. . . .
NaN	1 1 1 1 1	. .	. . . .

B. Decimal Floating Point Arithmetic Unit

Decimal floating point arithmetic unit performs three operations on IEEE P754-2008 decimal64 numbers namely, addition, subtraction and multiplication. Addition and subtraction operation on floating point operands are accomplished using a combined adder/subtractor unit, whereas multiplication on floating point unit is performed using a separate multiplication unit. Figure 2 shows the high level block diagram of floating point arithmetic unit.

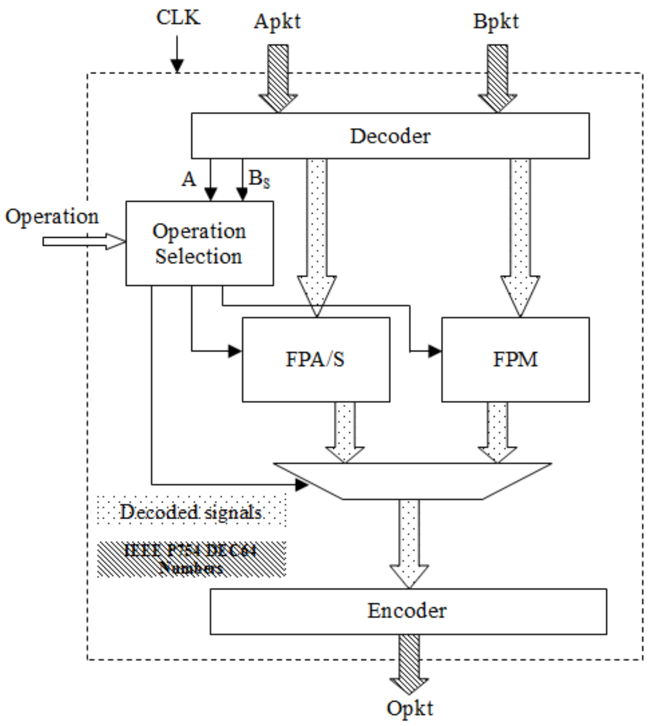


Figure 2 : High Level Block diagram – Floating point Unit

The inputs to the decoder block are two 64 bit IEEE P754-2008 floating point numbers encoded in DPD namely “Apkt” and “Bpkt”. Sign (A_s, B_s), exponent (A_e, B_e) and mantissa (A_m, B_m) information is extracted from two input packets by first converting the input information in DPD and then DPD is converted to BCD. After the decoding process the exponent (A_e, B_e) and Mantissa (A_m, B_m) are BCD numbers.

An operation selection block is used to determine the correct operation, when the 2 bit “operation” input is “00” then the selected operation is floating point addition and output of FPA/S (Floating point adder/subtractor) is assigned to the encoder block, when the operation input is “01” then the selected operation is floating point subtraction and also in this case output of FPA/S is assigned to encoder, when the operation input is “10” then the selected operation is multiplication and output of FPM (Floating point multiplier) is assigned to encoder block and when operation input is “11” then no operation is selected and output “Opkt” becomes 0.

FPA/S and FPM blocks are used to perform floating point addition/subtraction and floating point multiplication respectively. These blocks are explained in section 4 and section 5 respectively. An encoder block is used to encode the sign, exponent and mantissa of output to IEEE P754-2008 decimal64 format. The encoder block first converts the BCD exponent and mantissa into DPD and then convert the DPD numbers to decimal64 format.

C. Decimal Floating Point Adder/Subtractor

Figure 3 explains the algorithm for adding two decimal floating point numbers encoded in DPD dec64 format.

```
Step 1: Decode the inputs A and B to obtain
            ($A_s$,$A_E$,$A_m$) and ($B_s$,$B_E$,$B_m$)
Step 2: Determine effective operation (EOP)
            EOP <= As XOR Bs;
            EOP = 0 → Addition
            EOP = 1 → Subtract
Step 3: if Ae < Be, then Swap A and B
Step 4: Calculate d <= Ae – Be
Step 5: Shift right ‘Bm’ by d
Step 6: Add ‘d’ to ‘Be’
Step 7: Compute Zc <= $A_m \pm B_m$ (Depends on EOP)
Step 8: Ze <= Ae
Step 9: Zs <= Sign(greater(A,B))
Step 10: Encode to IEEE P754-2008 decimal64 format
```

Figure 3: Floating Point Addition - Algorithm

Figure 4 shows the floating point adder architecture for decimal floating point number system. The decoder generates mantissa (Am,Bm), exponent(Ae,Be) and sign (As,Bs).

The XOR gate determines the effective operation (EOP) by xoring the two signs (As, Bs). If eop signal is zero then the effective operation is addition and if the eop signal is 1 then the effective operation is subtraction. This eop signal is assigned to the BCD adder/subtractor unit.

A comparator unit is used to identify the larger of two numbers, if Ae > Be then swap signal is assigned to 0, and if Ae < Be, then swap signal is assigned to 1. Also the two exponents Ae and Be is subtracted and assigned to RSA (right shift amount) signal.

Swapping logic is used to assign the larger to the L channel and the smaller number to the S channel. When Swap signal is 0 then number A is larger than B, so Lm is assigned with Am, Le is assigned with Ae and Ls is assigned with As. Similarly so Sm is assigned with Bm, Se is assigned with Be and Ss is assigned with Bs. And if swap signal is 0 then B is greater than A and hence L channel is assigned with B and S channel is assigned with A.

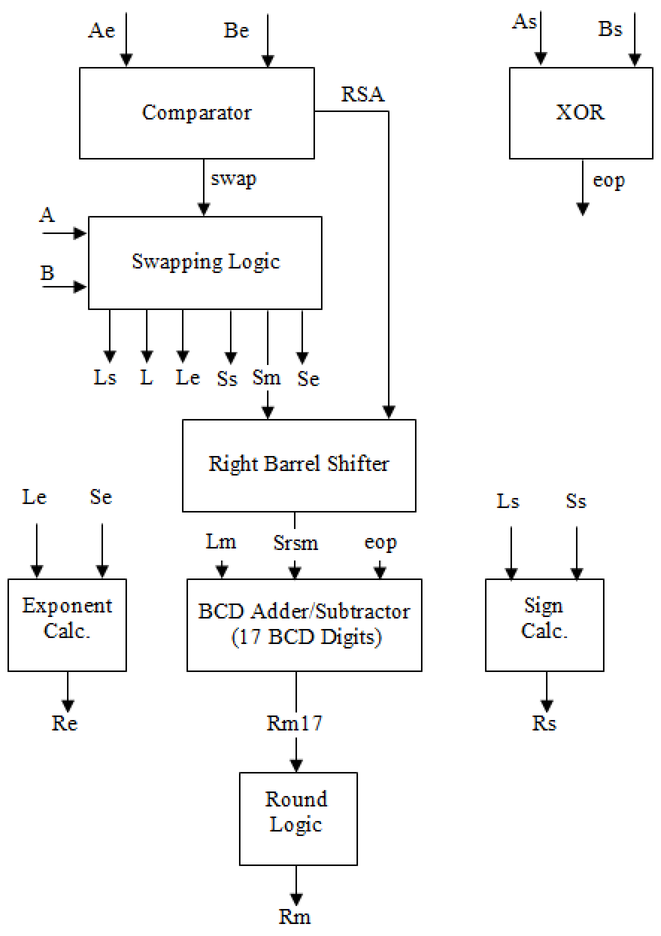
Now the smaller mantissa Sm is shifted right using a low delay barrel shifter, the right shifting amount is determined by RSA signal generated in comparator unit. The output is Srsm(small right shifted mantissa).

Next the two mantissas Srsm and Lm are operated. The operation is determined by eop signal generated earlier using XOR gate. The two mantissas are added/subtracted using 17 digit BCD adder/subtractor. Here a low power low delay full adder circuit is used to implement a BCD adder/subtractor to reduce the power consumption and delay of the design.

Figure 5 shows the 4 bit BCD adder, this BCD adder uses two 4 bit ripple carry adder, these 4 bit ripple carry adder uses conventional full adder.

Figure 6 shows the conventional full adder. All the logic gates in this design are applied with inputs all the time and this consumes power at all times, also the gate count for sum is two and the gate count for carry is three. We are replacing the conventional full adder by equal bypassing full adder. Figure 7 shows the low power low delay equal bypassed full adder.

In this full adder when input ‘A’ and input ‘B’ are equal then the output of XOR gate is ‘0’, this makes the control input of tri-state buffer ‘0’, now the output of tri-state inverter is high impedance ‘Z’, this blocks one channel of the multiplexer and reduces the power consumption. And if the two inputs ‘A’ and ‘B’ are different then the output of the XOR gate is ‘1’ and control input of tri-state inverter is also ‘1’, the input ‘C’ is complemented. Also at all times the gate count of Cout is reduces to 1, this is 2 less than the conventional full adder. So the power consumption and delay of the BCD adder is reduced.



Legend

A(77 bit): Number A {1 bit sign, 12 bit exp, 64 bit mantissa}
B(77 bit): Number B {1 bit sign, 12 bit exp, 64 bit mantissa}
Am (64 bit): 16 digit mantissa of number A
Bm (64 bit): 16 digit mantissa of number B
As (1 bit): Sign of number A
Bs (1 bit): Sign of number B
Ae (12 bit): 3 digit exponent of number A
Be (12 bit): 3 digit exponent of number B
RSA (4 bit): Right shift amount
eop (1 bit): effective operation – add/sub
swap(1 bit): swap signal
Lm (64 bit): 16 digit mantissa of number L
Sm (64 bit): 16 digit mantissa of number S
Ls (1 bit): Sign of number L
Ss (1 bit): Sign of number S
Le (12 bit): 3 digit exponent of number L
Se (12 bit): 3 digit exponent of number S
Rm17 (68 bit): 17 digit mantissa of number R
Rm (64 bit): 16 digit mantissa of number R
Rs (1 bit): Sign of number R
Re (12 bit): 3 digit exponent of number R

Figure 4: Floating Point Adder / Subtractor architecture

The rounding logic unit truncates the 17 digit mantissa generated by the BCD adder/subtractor to 16 digit mantissa Rm.

The exponent calculation unit generates the output exponent. The large exponent Le is assigned to the result exponent Re if truncation is not needed, if the output mantissa is truncated then the resultant exponent is added with 1.

The sign calculation unit generates the output sign. The sign of greater number is assigned to the resultant sign Rs. Here the sign of greater number is Ls. The three information Rm, Re and Rs are applied to the encoder block for encoding it to decimal64 number.

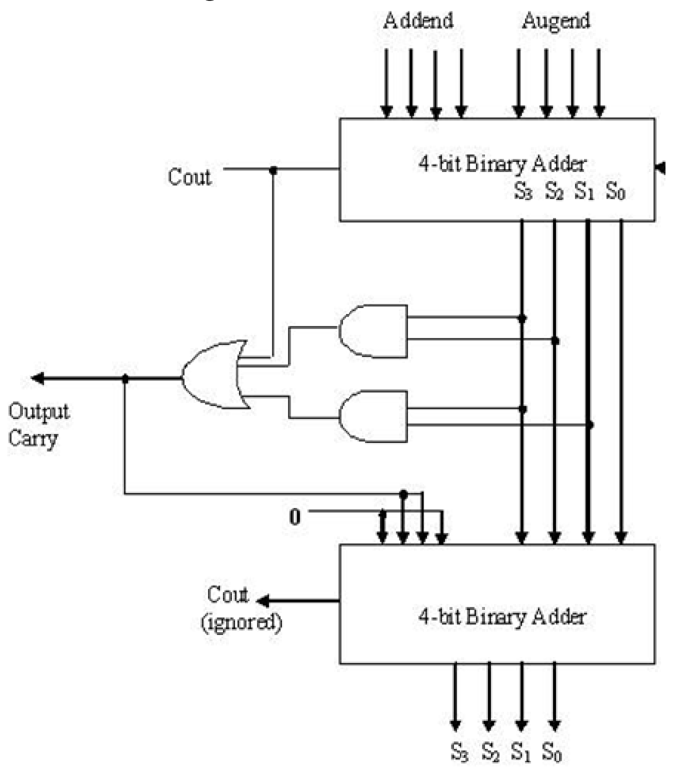


Figure 5: BCD Adder

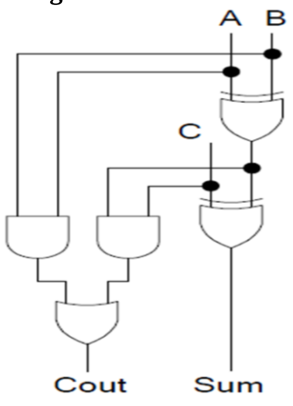


Figure 6: Conventional Full Adder

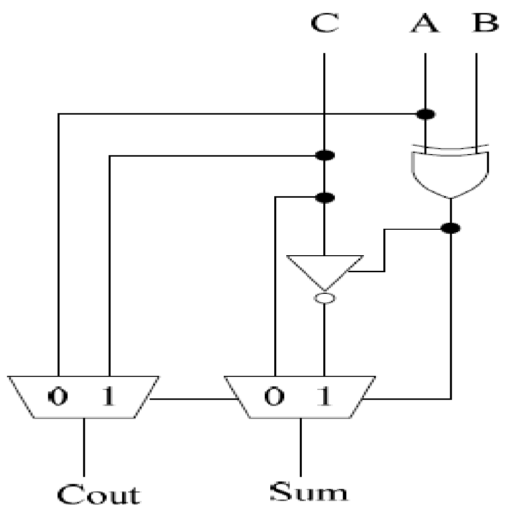


Figure 7 : Low Power – Low Delay Full Adder

D. Floating Point Multiplier

Figure 8 depicts the basic algorithm for floating point multiplication.

1. *Extract As, Ae, Am, Bs, Be,Bm by decoding the incoming packets.*
2. *Rs <= As XOR Bs*
3. *Re <= Ae + Be – bias*
4. *Product <= Am * Bm*
5. *Truncate product to generate 16 digit mantissa Rm*
6. *Generate appropriate flags InF, NaN, Zero, OF, UF.*
7. *Encode to output result format.*

Figure 8 : Floating Point Multiplication - Algorithm

Figure 9 shows the architecture of floating point multiplier.

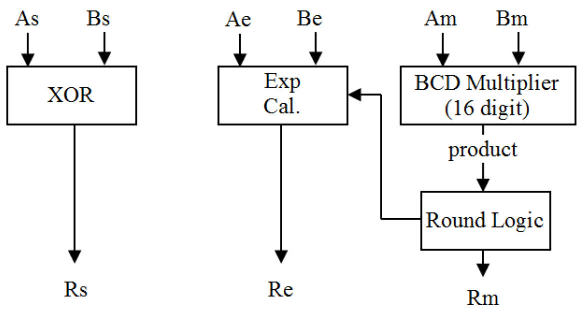


Figure 9 : Floating Point Multiplication - Architecture

The sign (As, Bs), exponent (Ae, Be) and mantissa (Am, Bm) information is extracted from the decimal64 number in decoder unit. This information along with the a clk (not shown in figure) is used in floating point multiplier to generate the result R.

First the two sign bits As and Bs are XORED to generate the result sign bit Rs. The exponent is generated by adding the two input exponents Ae and Be. The input exponents are biased and hence the result of the addition is further subtracted with the bias value. In decimal64 format the bias value is 398. So the result exponent is calculated as:

$$Re = Ae + Be – bias$$

Parallel to the sign and exponent calculation, the product is generated by multiplying the two mantissas Am and Bm. This multiplication is accomplished by a 16 digit parallel BCD multiplier.

Figure 10 represents an area optimized BCD digit multiplier. This multiplier produces the result of multiplication in binary and we need a binary to BCD converter shown in figure 11.The B is the higher nibble of the multiplication and C is the lower nibble of multiplication. Figure 12 shows the parallel multiplication process of a 4 x 4 BCD multiplier. Xi and yi are single digit BCD numbers. These numbers are multiplied using the single digit BCD multiplier shown in figure 10-11. pyixiH and pyixiL are higher and lower nibble of multiplication respectively.

				x3	**x2**	**x1**	**x0**
				y3	**y2**	**y1**	**y0**
				P0003L	P0002L	P0001L	p0000L
			P0003H	P0002H	P0001H	P0000H	
			P0103L	P0102L	P0101L	P0100L	
		P0103H	P0102H	P0101H	P0100H		
		P0203L	P0202L	P0201L	P0200L		
	P0203H	P0202H	P0201H	P0200H			
	P0303L	P0302L	P0301L	P0300L			
P0303H	P0302H	P0301H	P0300H				
P7	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

Figure 12: 4 x 4 BCD Multiplication process

Figure 13 depicts the 4 x 4 multiplier architecture to implement the algorithm shown figure12. In the process of floating point multiplication this 4x4 multiplier is extended to implement 16 x 16 multiplier. The multiplication is the critical operation the design of floating point arithmetic unit and hence in order to reduce we have used a parallel BCD multiplier.

The result of the 16 x 16 BCD multiplier is a 32 digit BCD mantissa. But Decimal64 only supports 16 digit of mantissa. so a rounding logic unit is incorporated to round off the least significant digits and adjust the exponent accordingly. The rounding unit generated the 16 digit mantissa Rm. Now the sign Rs, exponent Re and mantissa Rm are assigned to the encoder unit.

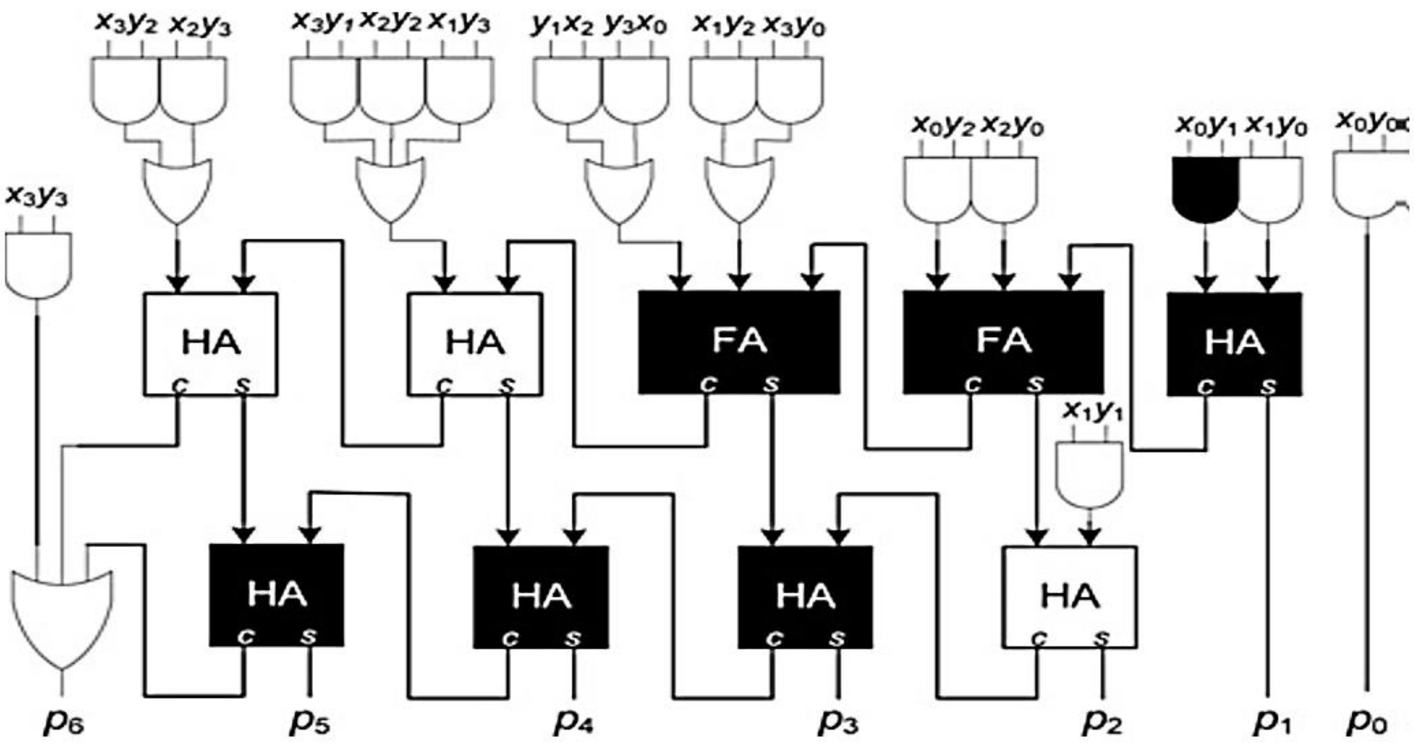


Figure 10 : Area optimized BCD digit multiplier

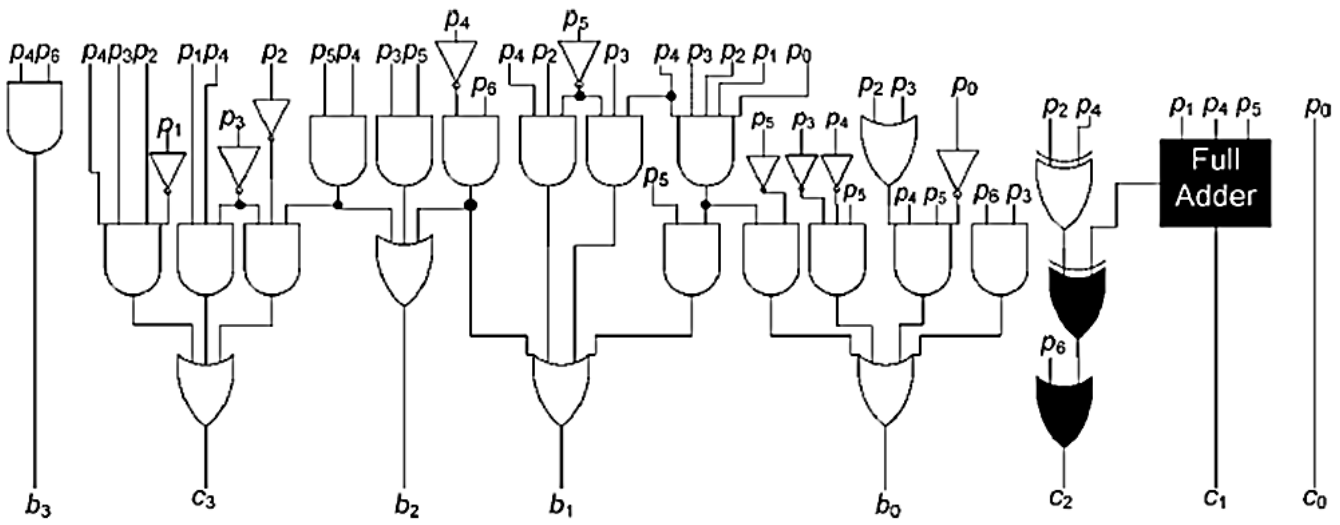


Figure 11 : Binary Product to BCD Converter

III. RESULTS AND DISCUSSION

All logics were described in VHDL. The design has been implemented on Xilinx Virtex-5 device XC-5VLX30FF324-3. Resource utilization is shown in table 1. As seen from table 1 that the critical module of the design is the FPM (floating point multiplier) with large resource usage and delay. The maximum operating frequency of the floating point arithmetic unit is 45.935 MHz.

Table 1: Device Utilization Summary

Module	Slice Registers	Slice LUT's	Delay
Decoder	4	138	4.04ns
FPA/S	5	652	45.39ns
FPM	-	15895	81.96ns
Encoder	109	72	-
Arithmetic unit	274	16778	-

Table 2 shows the power consumption summary of floating point arithmetic unit. Dynamic power depends mainly on the clock rate and hence it is increasing with clock frequency. The power consumption at maximum operating frequency (MOF) is 634mW.

Table 2: Power Consumption

S.no	Frequency	Static	Dynamic	Total
1	20 Mhz	380 mw	115 mw	495mw
2	30 Mhz	381 mw	168 mw	549mw
3	40 Mhz	382 mw	220 mw	602mw
4	45.9 Mhz (MOF)	382 mw	252 mw	634mw

IV. CONCLUSION

In this work implementation of IEEE P754-2008 decimal floating point arithmetic unit is accomplished. The advantage of decimal floating point arithmetic over binary floating point arithmetic is that decimal floating point representation does not has errors with binary floating point arithmetic like rounding error, arithmetic error and errors associated when representing fraction decimal numbers.

The arithmetic unit performs three operations, floating point addition, floating point subtraction and floating point multiplication. A combined floating addition and subtraction unit is used in the design; the combined floating point adder/subtractor unit uses a low power low delay full adder to implement the BCD adder/subtractor. The floating point multiplication unit uses a parallel 16 x 16 BCD multiplier. Parallel multiplier is opted because of its low delay compared to serial multiplier. It is observed from table 1 and table 2 that the 16 x16 BCD digit multiplier is most critical module of the design, its takes the most resources and has the maximum delay. In future we can improve the performance of the design by optimizing the 16 x 16 BCD digit multiplier.

V. REFERENCES

[1]. M. F. Cowlishaw, "Decimal Floating-Point: Algorism for Computers," Proc. IEEE 16th Symp. Computer Arithmetic, pp. 104-111, 2003.

[2]. IBM Corporation, The 'Telco' benchmark, http://speleotrove.com/ Decimal/telcoSpec.html, 2005.

[3]. D.-G. for Economic and F. A. C. from the Commission to the European Council, "Review of the Introduction of Euro Notes and Coins," EURO PAPERS, Apr. 2002.

[4]. M.F. Cowlishaw The decNumber library, v3.68. IBM, http://speleotrove.-com/decimal/decnumber.pdf, 2013.

[5]. S. Microsystems BigDecimal Class, Java 2 Platform Standard ed. 5.0, APISpecification,http://docs.oracle.com/javase/1.5.0/docs/api/java/math/BigDecimal.html, 2013.

[6]. M. Cornea, C. Anderson, J. Harrison, P.T.P. Tang, E. Schneider, and C. Tsen, "A Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic Using the Binary Encoding Format," Proc. IEEE 18th Symp.

[7]. ANSI/IEEE 754-1985, "Standard for Binary Floating-Point Arithmetic".

[8]. R. K. Yu, G.B. Zyner, 167 MHz radix-4 floating point multiplier, Proceedings 12th Symposium on Computer Arithmetic, 1995, pp. 149-154.

[9]. C. Gamez, R. Pang, Apparatus and method for rounding operands, U.S. patent 5258943, 1993.

[10]. M. Saishi, T. Minemaru, Multiplication circuit having rounding function, U.S. patent 5500812, 1996.

[11]. Guy Even, Silvia M. Mueller, Peter-Michael Seidel "A dual precision IEEE Floating-point multiplier" Elsevier INTEGRATION, the VLSI journal 29 (2000) 167-180.

[12]. C. Tsen, M.J. Schulte, and S.G. Navarro, “Hardware Design of a Binary Integer Decimal Based IEEE P754 Rounding Unit,” Proc. IEEE 18th Int’l Conf. Application-Specific Systems, Architectures and Processors, pp. 115-121, 2007.

[13]. B.J. Hickmann, A. Krioukov, M.J. Schulte, and M.A. Erle, “A Parallel IEEE P754 Decimal Floating-Point Multiplier,” Proc. IEEE 25th Int’l Conf. Computer Design, 2007.

[14]. C. Tsen, S.G. Navarro, M.J. Schulte, B. Hickmann, and K. Compton, “A Combined Decimal and Binary Floating-Point Multiplier,” Proc. IEEE 20th Int’l Conf. Application-Specific Systems, Architectures, and Processors, pp. 8-15, 2009.

[15]. J. Di and J. S. Yuan, “Power-aware pipelined multiplier design based on 2-dimensional pipeline gating,” in *13th Great Lakes Symposium on VLSI*. ACM, 2003, pp. 64–67.

[16]. Sunjoo Hong, Taehwan Roh and Hoi-Jun Yoo, “a 145w 8×8 parallel multiplier based on optimized bypassing architecture”, department of electrical engineering, Korea advanced institute of science and technology (KAIST), Daejeon, Republic of Korea, IEEE, pp.1175-1178, 2011.

[17]. Yin-Tsung Hwang, Jin-Fa Lin, Ming-Hwa Sheu and Chia-Jen Sheu, “low power multipliers using enhenced row bypassing schemes”, department of electronic engineering, National Yunlin University of science & technology, Touliu, Yunlin, Taiwan, IEEE, pp.136-140, 2007.

[18]. George Economakos, Dimitris Bekiaris and Kiamal Pekmestzi, “a mixed style architecture for low power multipliers based on a bypass technique”, national technical University of Athens, school of electrical and computer engineering, heroon polytechniou 9, GR-15780 Athens, Greece, IEEE, pp.4-6, 2010.

[19]. Meng-Lin Hsia and Oscal T.-C. Chen, “low power multiplier optimized by partial-product summation and adder cells”, dept. of electrical engineering, national chung cheng University, chia-yi, 621, Taiwan, IEEE, pp.3042-3045, 2009. [12] P. C. H. Meier, “analysis and design of low power digital multipliers”, Ph.D. thesis, Carnegie Mellon University, dept. of electrical and computer engineering, Pittsburgh, Pennsylvania, 1999.

[20]. Carlos Minchola, Martin Vazquez and Gustavo Sutter “A FPGA IEEE 754 2008 decimal floating point adder subtractor” 2011 IEEE.

[21]. Yanyu Ding, Deming Wang, Jianguo Hu and Hongzhou Tan, “A Low power Parallel Multiplier Based on Optimized-Equal-Bypassing-Technique”, Third International Conference on Information Science and Technology March, 2013 IEEE, China