Design of IEEE - 754 Floating point Arithmetic Processor

J. Laxmi[1], R. Ramprakash[2]

[1]M.Tech Student
CVSR College of Engineering
laxmi402.jatti@gmail.com

[2]Assistant Professor, ECE Department
CVSR College of Engineering
ramprakash.rampelli@gmail.com

Abstract: ***In this paper, we deal with the designing of a 32-bit floating point arithmetic processor for RISC/DSP processor applications. It is capable of representing real and decimal numbers. The floating point operations are incorporated into the design as functions. The logic for these is different from the ordinary arithmetic functions. The numbers in contention have to be first converted into the standard IEEE floating point standard representation before any sorts of operations are conducted on them. The floating point representation for a standard single precision number is a 32-bit number that is segmented to represent the floating point number. The IEEE format consists of four fields, the sign of the exponent, the next seven bits are that of the exponent magnitude, and the remaining 24 bits represent the mantissa sign. The exponent in this IEEE standard is represented in excess-127 format all the arithmetic functions like addition, subtraction, multiplication and division will be design by the processor. The main functional blocks of floating point arithmetic processor design includes, Arithmetic logic unit(ALU), Register organization, control & decoding unit, memory block, 32-bit floating point addition, subtraction, multiplication and division blocks. This processor IP core can be embedded many places such as co-processor for embedded DSP and embedded RISC controller. The overall system architecture will be designed using HDL language and simulation, synthesis.***

Keywords: single, dual precision, floating point, ALU, FPGA

1. Introduction

A. Floating point

In C, an operation is the effect of an operator on an expression. Specific to floating-point numbers, a floating-point operation is any mathematical operation (such as +, -, *, /) or assignment that involves floating-point numbers (as opposed to binary integer operations).Floating-point numbers have decimal points in them. The number 2.0 is a floating-point number because it has a decimal in it. The number 2 (without a decimal point) is a binary integer. Floating-point operations involve floating-point numbers and typically take longer to execute than simple binary integer operations. For this reason, most embedded applications avoid wide-spread usage of floating-point math in favor of faster, smaller integer operations. In computing, floating point describes a method of representing an approximation to real numbers in a way that can support a wide range of values. The numbers are, in general, represented approximately to a fixed number of significant digits (the mantissa) and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form:

Significant digits × base exponent

The idea of floating-point representation over intrinsically integer fixed-point numbers, which consist purely of significant, is that expanding it with the exponent component achieves the greater range. For instance, to represent large values, e.g. distances between galaxies, there is no need to keep all 39 decimal places down to femtometre-resolution, employed in particle physics. Assuming that the best resolution is in light years, only 9 most significant decimal digits matter whereas 30 others bear pure noise and, thus, can be safely dropped. This is 100-bit saving in storage. Instead of these 100 bits, much fewer are used to represent the scale (the exponent), e.g. 8 bits or 2 decimal digits. Now, one number can encode the astronomic and subatomic distances with the same 9 digits of accuracy. But, because 9 digits is 100 times less accurate than 9+2 digits reserved for scale, this is considered as precision-for-range trade-off. The example also explains that using scaling to extend the dynamic range results in another contrast with usual fixed-point numbers: their values are not uniformly spaced. Small values, the ones close to zero, can be represented with much higher resolution (1 femtometre) than distant ones because greater scale (light years) must be selected for encoding significantly larger values. That is, floating-point cannot represent point coordinates with atomic accuracy in the other galaxy, only close to the origin. The term floating point refers to the fact that their radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number. This position is indicated as the exponent component in the internal representation and floating-point can thus be thought of as a computer realization of scientific notation. Over the years, a variety of floating-point representations have been used in computers. However, since the 1990s, the most commonly encountered representation is that defined by the IEEE 754 Standard. The speed of floating-point operations, commonly referred to in performance measurements as Flops, is an important machine characteristic, especially in software that performs large-scale mathematical calculations. A number representation (called a numeral system in mathematics) specifies some way of storing a number that may be encoded as a string of digits. The arithmetic is defined as a set of actions on the representation that simulate classical arithmetic operations. There are several mechanisms by which strings of digits can represent numbers. In common mathematical notation, the digit string can be of any length, and the location of the radix point is indicated by placing an explicit "point" character (dot or comma) there. If the radix point is not specified then it is implicitly assumed to lie at the right (least significant) end of the string (that is, the number is an integer). In fixed-point systems, some specific assumption is made about where the radix point is located in the string. For example, the convention could be that the string consists of 8 decimal digits with the decimal point in the middle, so that "00012345" has a value of 1.2345. Floating point representation makes numerical computation much easier. You could write all your programs using integers or fixed-point representations, but this is tedious and error-prone. For example, you could write a program with the understanding that all integers in the program are 100 times bigger than the number they represent. The integer 2345, for example, would represent the number 23.45. As long as you are consistent, everything works. This is actually the same as using fixed point notation. In fixed point binary notation the binary point is assumed to lie between two of the bits. This is the same as an understanding that the integer the bits represent should be divided by a particular power of two. But it is very hard to stay consistent. A programmer must remember where the decimal (or binary) point "really is" in each number. Sometimes one program needs to deal with several different ranges of numbers. Consider a program that must deal with both the measurements that describe the dimensions on a silicon chip (say 0.000000010 to 0.000010000 meters) and also the speed of electrical signals, 100000000.0 to 300000000.0 meters/second. It is hard to find a place to fix the decimal point so that all these values can be represented. Notice that in writing those numbers (0.000000010, 0.000010000, 100000000.0, 300000000.0) I was able to put the decimal point where it was needed in each number. To increase the speed and efficiency of real-number computations, computers or FPUs typically represent real numbers in a binary floating-point format. In this format, a real number has three parts: a sign, a significant, and an exponent. Figure1 shows the binary floating-point format that the IA FPU uses. This format conforms to the IEEE standard.

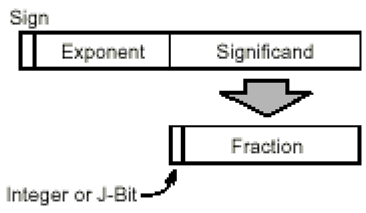


Figure 1: Binary Floating-Point Format

Processor, computer arithmetic-logic unit that uses a minimal instruction set, emphasizing the instructions used most often and optimizing them for the fastest possible execution. Software for RISC processors must handle more operations than traditional CISC [Complex Instruction Set Computer] processors, but RISC processors have advantages in applications that benefit from faster instruction execution, such as engineering and graphics workstations and parallel-processing systems. They are also less costly to design, test, and manufacture. In the mid-1990s RISC processors began to be used in personal computers instead of the CISC processors that had been used since the introduction of the micro processors.

2. Characteristics of processor

- Fixed size of the instruction, equal to the computer word length and to the width of the data bus.
- Standard execution time of all instructions, preferably with in a single CPU cycle.
- Small number of instructions, not to exceed 20.
- A Small number of 5 instruction formats.
- A Small number of 4 addressing modes.
- Memory access by load and store instruction only.
- All operations, except load and store, are register to register, within in the CPU.
- Hardwired control unit (combinational circuit).
- A relatively large (at least 32) general purpose CPU register file.

3. Proposed Floating Point Arithmetic Functions

A floating-point number is the one, which is capable of representing real and decimal numbers. The floating-point operations are incorporated into the design as functions. The logic for these is different from the ordinary arithmetic functions. The numbers in contention have to be first converted into the standard IEEE 754, 1985 floating point standard representation before any sort of operations are conducted on them. The floating-point representation for a standard single precision number is.

S	E14-E0	Ma47-Ma0

Figure 2: floating point function

A single precision number is a 32-bit number that is segmented to represent the floating-point number. The above representation is the IEEE-754 1985 standard representation. The MSB is the sign-bit i.e. the sign of the floating point number. The next eight bits are that of the exponent. The exponent in this IEEE standard is represented in excess-127 format. I.e. the exponent obtained by balancing operations is added to 0111, 1111. Therefore zero is represented by 0111, 1111. Positive numbers are represented by binary values greater than 0111, 1111 and negative numbers are represented by binary values less than it.

4. Performance Evaluation

A. System Model

In this paper, we simulated the performance of the proposed algorithm in ModelSim using Verilog. A large sequence of bits was generated randomly at the sender. The generated bits were then divided into 64-bit blocks; each block was then encrypted to a 128-bit block using M-DES. The encrypted blocks were then assumed to be transmitted over the wireless channel. The encrypted blocks received in error are then decrypted block by block. Then, the resulted sequence of bits at the receiver after decryption is compared with the sequence of bits at the sender before encryption, and the error is calculated. The same procedure is done using DES, Triple DES and Triple M-DES.

B. BER Analysis

Using simulation we show that the performance of M-DES in terms of BER is much better than DES. Initially, we studied the BER of DES, assuming one bit error only. We studied the effect of the error analytically up to the end of round three. Then it became harder to complete the sixteen rounds analytically, so we used simulations to evaluate the BER of DES, and we found that on average there was 32 bits in error out of the 64 bits. Simulations also show that in M-DES, having one or more errors at the received ciphertext block will not result in half the decrypted bits to be in error, while for DES if any bit is received in error, this will result in half the bits to be in error.

C. Security

We showed in details in the previous section how the algorithm is considered secure against brute force and differential cryptanalysis attacks. The addition of the new 80-bit key, make it impossible to crack the algorithm using brute force attack. While the addition of the new round reduces the probability of cracking the algorithm using differential cryptanalysis to almost zero.

D. Key Management

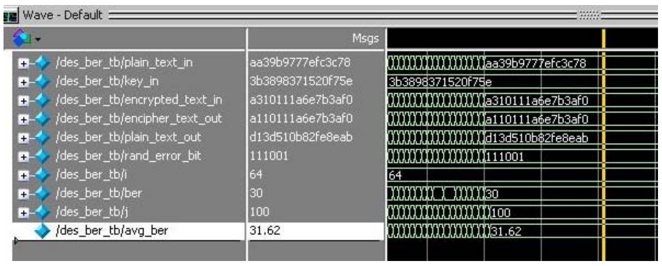
An efficient way to share the new key is to add the new 80-bit key to the old 56-bit key and deal with them as one key at the transmitter. Both the sender and the receiver will need to divide it into two keys when encryption or decryption is done.

E. Complexity

The proposed algorithm is less complex than DES in terms of the number of distinct S-Box mapping tables. However, the number of rounds and the key size of M-DES is more than DES, which might increase the complexity compared to DES.

5. Simulation Results

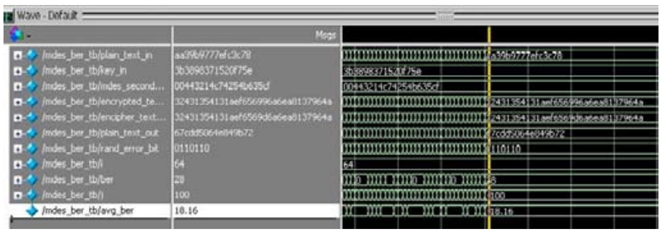
This section provides the simulation results of the BER obtained for M-DES and Triple M-DES compared to DES. Here encrypted text is the output of encryption algorithm and encipher text is the input to the decryption algorithm after transmission. Here we have considered a random error bit is the changed bit while transmission. Here i and j are the iteration variables used for the 64 bits and 100 iterations respectively. The average ber is calculated which is observed as 31.62 for an standard DES algorithm, as shown below (Fig.3):



```
Wave - Default
                                     Msgs
/des_ber_tb/plain_text_in            aa39b9777efc3c78
/des_ber_tb/key_in                   3b3898371520f75e
/des_ber_tb/encrypted_text_in        a310111a6e7b3af0
/des_ber_tb/encipher_text_out        a110111a6e7b3af0
/des_ber_tb/plain_text_out           d13d510b82fe8eab
/des_ber_tb/rand_error_bit           111001
/des_ber_tb/i                        64
/des_ber_tb/ber                      30
/des_ber_tb/j                        100
/des_ber_tb/avg_ber                  31.62
```

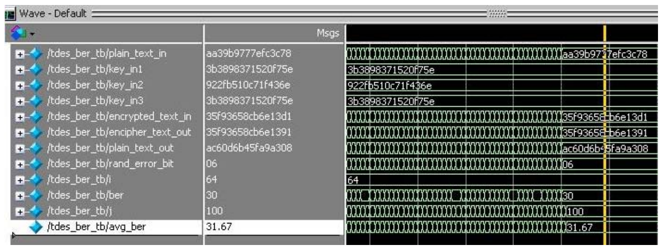
Figure 3: Average BER output for standard DES algorithm

Similarly, for M-DES algorithm requires the same 64-bit plaintext as for standard DES, but it takes a136-bit key and produces a 128-bit ciphertext. Here, the second key is the 80-bit key which is given to the Round 17. Here, we are assuming that the encrypted text differ from the encipher text by one random bit. So, here the average ber output is 18.16, which is comparatively much less than standard DES algorithm. Thus the ber performance is improved. (Fig.4)



```
Wave - Default
                                     Msgs
/mdes_ber_tb/plain_text_in           aa39b9777efc3c78
/mdes_ber_tb/key_in                  3b3898371520f75e
/mdes_ber_tb/mdes_second...          00443214c74254b635cf
/mdes_ber_tb/encrypted_te...         32431354131aef656996a6ea0137964a
/mdes_ber_tb/encipher_text...        32431354131aef6569d6a6ea0137964a
/mdes_ber_tb/plain_text_out          67cdd5064e849b72
/mdes_ber_tb/rand_error_bit          0110110
/mdes_ber_tb/i                       64
/mdes_ber_tb/ber                     28
/mdes_ber_tb/j                       100
/mdes_ber_tb/avg_ber                 18.16
```

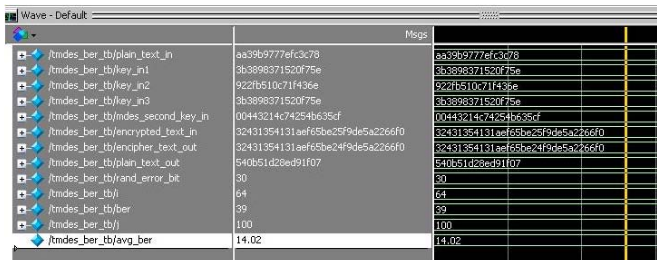
Figure 4: Average BER output for M-DES algorithm



```
Wave - Default
                                     Msgs
/tdes_ber_tb/plain_text_in           aa39b9777efc3c78
/tdes_ber_tb/key_in1                 3b3898371520f75e
/tdes_ber_tb/key_in2                 922fb510c71f436e
/tdes_ber_tb/key_in3                 3b3898371520f75e
/tdes_ber_tb/encrypted_text_in       35f93658cb6e13d1
/tdes_ber_tb/encipher_text_out       35f93658cb6e1391
/tdes_ber_tb/plain_text_out          ac60d6b45fa9a308
/tdes_ber_tb/rand_error_bit          06
/tdes_ber_tb/i                       64
/tdes_ber_tb/ber                     30
/tdes_ber_tb/j                       100
/tdes_ber_tb/avg_ber                 31.67
```

Figure 5: Average BER output for Triple DES algorithm

Similarly, we can observe the results for the triple DES algorithm (Fig.5); in this the DES algorithm is used three times. So, the average ber calculated for triple DES is 31.67, which is almost same as for the standard DES. Similarly, if we use M-DES in triple DES, i.e if we use three M-DES instead of standard DES than the average ber is obtained as 14.02(Fig.6). This is better than normal triple DES. Thus the ber performance is much more improved and the security is also increased using triple modified DES.



```
Wave - Default
                                        Msgs
/tmdes_ber_tb/plain_text_in             aa39b9777efc3c78
/tmdes_ber_tb/key_in1                   3b3898371520f75e
/tmdes_ber_tb/key_in2                   922fb510c71f436e
/tmdes_ber_tb/key_in3                   3b3898371520f75e
/tmdes_ber_tb/mdes_second_key_in        00443214c74254b635cf
/tmdes_ber_tb/encrypted_text_in         32431354131aef65be25f9de5a2266f0
/tmdes_ber_tb/encipher_text_out         32431354131aef65be24f9de5a2266f0
/tmdes_ber_tb/plain_text_out            540b51d28ed91f07
/tmdes_ber_tb/rand_error_bit            30
/tmdes_ber_tb/i                         64
/tmdes_ber_tb/ber                       39
/tmdes_ber_tb/j                         100
/tmdes_ber_tb/avg_ber                   14.02
```

Figure 6: Average BER output for Triple M-DES algorithm

Table 1: The BER comparison table for different algorithms

Architecture	*Standard*		*Modified*	
	DES	*T-DES*	*DES*	*T-DES*
BER	31.62	31.67	18.16	14.02

6. Conclusion

In this paper, we showed how well known block cipher encryption algorithms are not sufficient for the use in wireless applications, because in these applications, the signal may experience severe degradations and attenuations, which causes wrong reception of the signal and hence a catastrophic effect on the decryption process. In this paper, we proposed a new encryption mechanism based on modified DES. Using simulation we quantified the performance of the proposed M-DES versus the known standard DES and Triple DES versus Triple M-DES for encryption in a wireless communication channel. We showed that the new algorithm outperforms the standard DES and Triple DES algorithms in terms of error performance. We also showed that the new algorithm enhanced the security to a high security level which is prone to all applicable types of attacks.

References

[1] M. Haleem, C. Chetan, R. Chandramouli and K.P.Subbalakshmi, “Opportunistic Encryption: A Trade-Off between Security and Throughput in Wireless Networks,” IEEE Transaction on Dependaple and Secure Computing, vol. 4, no. 4, pp.313-324, Oct 2007.
[2] Reason, “End-to-End Confidentiality for Continuous-Media Applications in Wireless Systems,” PhD dissertation, UC Berkeley, Dec.2000.
[3] “Technical specification group services and system aspects,” 3GPP TS 55.216 V6.2.0, September 2003.
[4] Sedat Akleylek, “On the avalanche effect of MISTY1, KASUMI and KASUMI-R,” Master’s thesis, Middle East Technical University, Feb 2008.
[5] O. Dunkelman, N. Keller and A. Shamir, “A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony,” Cryptology ePrint Archive, Report 2010/013, Feb 2010.
[6] Behrouz A. Forouzan, Cryptography and Network Security, 1st Edition, Mc Graw Hill, 2008.
[7] M. Matsui, “On correlation between the order of S-boxes and the strength of DES,” in Proceedings of Eurocrypt94 (A. De Santis, ed.),no. 950 in Lecture Notes in Computer Science, pp. 366375, Springer-Verlag, 1995.
[8] E.Biham and A.Shamir, “Differential Cryptanalysis of the Full 16-round DES,” Proceedings of Crypto’92, vol. 740, Santa Barbara, CA, December 1991