Journal of Computer Science of Newports Institute of Communications and Economics
Volume 5, Issue-2014, ISSN: 2226-3683

Implementation of Data Encryption Standard (DES) on FPGA

[1]FOZIA HANIF KHAN, [2]REHAN SHAMS, [3]ASIF HASAN, [4]NAWAID HASAN,

[1]Department of Computer Science, Newports Institute of Communications and Economics,
[2,3,4]Department of Telecommunication Engineering,Sir Syed University of Engineering and Technology

[1]drfozia@newports.edu.pk, [2]r.shams@ssuet.edu.pk, [3]asifhasanaskari@hotmail.com, [4]nawaidhasan@hotmail.com,

ABSTRACT

Data Encryption can be performed by using two types of algorithms. One is symmetric key and other is asymmetric key algorithm. Symmetric key algorithms are widely used due to less complexity and faster processing as compared to asymmetric key algorithm. Most commonly used symmetric key algorithm is Data Encryption standard (DES). In this paper, we present an efficient implementation of DES algorithm using High Level Language (HLL) approach. The hardware platform use for DES implementation is Spartan 3e XC3S1600E Field Programmable Gate Array (FPGA). Xilinx System Generator is a HLL tool which is used for DES implementation. System Generator provide environment similar to Simulink and provide the facility to pictorially design the system instead of writing thousands lines of code. It provides the synthesizable code of the design which can directly burns on FPGA to get the implementation of design on hardware. We manually customized our design by using conventional blocks of Xilinx System Generator to get optimum performance in terms of speed and area. Our FPGA implementation shows best performance in terms of speed and area as compared with any other software and hardware implementation counterparts, it operates on a frequency of 310.174 MHz and gives a throughput of 1.24GHz, it uses 1344 slices and 120 BRAMs.

Keywords: *DES, System Generator, FPGA*

1. Introduction

With the technological evolution of computerized systems, security and confidentiality have become one of main issues of these systems to avoid the frauds. Cryptography plays a vital role in this regard to provide privacy, authentication and integrity protection. It is not only to prevent the data from hacking but also to protect confidential data from the opponent. Nowadays cryptography is largely used in Cellular phones, ATM networks, online banking, radio modems and Smart cards etc [1]. In cryptography, the basic and known functions are encryption and decryption. Encryption is the process of hiding original information by applying a set of procedures called an encryption algorithm and a key. The message in its original form is called plaintext while the resulting text after the encryption process is called ciphertext. To retrieve the plaintext from the ciphertext, the decryption process is executed. The decryption is achieved by again applying a specific set of procedures and the same key (in case of symmetric), the ciphertext will again be converted in plaintext i.e. in its original form.

The original idea behind the Data Encryption Algorithm was developed by IBM in the 1960's and was based on Claude Shannon's concept [2]. The technique was first called as Lucifer and later refined and renamed as the DEA (Data Encryption Algorithm). In 1977 the United States Government chose the Data Encryption Standard (DES).DES was widely adopted by industry for secure communication,
In DES, 64 bit data block is converted into 64 bit cipher text using a key having size of 56 bit. The algorithm consists of 16 rounds. The same process is repeated to decrypt 64 bit cipher text into plain text of 64 bit using the same key as shown in Figure 1.

FPGAs are ideal for implementation of the cryptographic algorithms. They are reconfigurable platform that give time and cost effective solutions as compared to ASICs that are expensive and require the largest development time. [3] It provides far above the ground performance than software implementations and can be reconfigured on the fly to store the updated encryption or decryption standard. We are using Spartan 3e XCS1600E FPGA kit here for our implementation [4].

2. Data Encryption Standard

DES is a typical block cipher [5], there are two inputs to the encryption function, i.e. the plaintext and the key, and two inputs to the decryption function, i.e. the ciphertext and the key. The message (plaintext or ciphertext) must be of 64 bits and the key of 56 bits in length. The key apparently composed of sixty-four bits;

but, 8 bits are used uniquely for checking parity, and are then discarded; only fifty-six of those are effectively used by the algorithm. Processing of the plaintext proceeds in three phases [6],
The 64 bit text passes through an Initial Permutation (IP). Then 16 "rounds" of operations which involves both permutation and substitution functions that mix the data and key together in a prescribed manner [7]. The pre-output obtained from swapping the left and right halves of the output in each round, is then passed through a Permutation (IP-1) that is the inverse of the Initial permutation (IP) function, to produce the 64-bit ciphertext or plaintext. Flow diagram of algorithm is shown in Figure 2.

2.1. Initial and Final Permutation

The initial permutation occurs before round 1, it transposes the input block as shown in Table 1, this table should be read from left to right and top to bottom, like for example, according to table, first bit is replaced with fifty-eighth bit, second bit is replaced with fiftieth bit and similarly all the bits are mapped according to table [8].
The final permutation is the last step of DES algorithm and takes places after the 16 rounds. It is the inverse of initial permutation as shown in table 2.

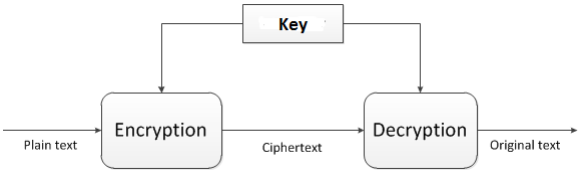

Figure 1: DES encryption and decryption

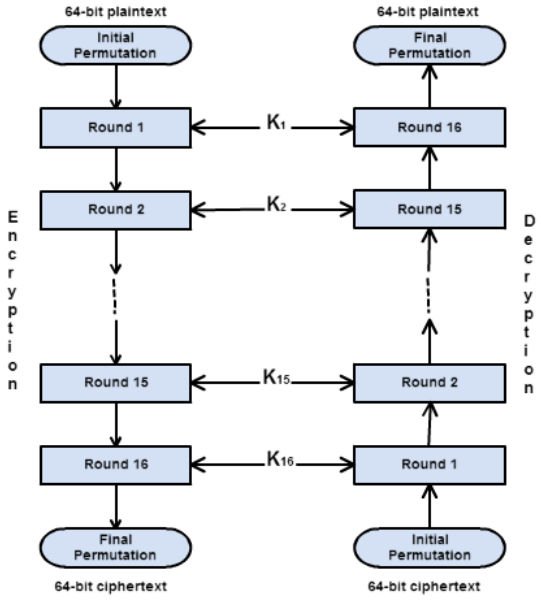

Figure 2: DES encryption and decryption algorithm

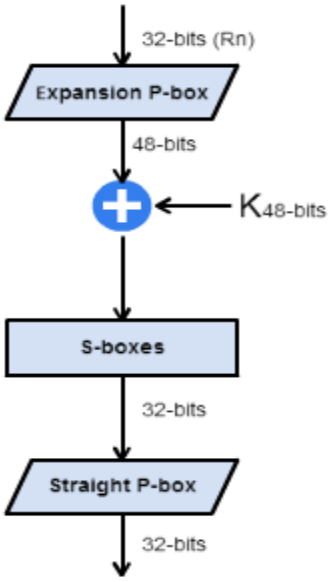

Figure 3: DES function

Table 1. Initial permutation

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Table 2. Final permutation

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table 3. Expansion Permutation

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

2.2. Rounds Structure

DES uses 16 rounds. Each round of DES is a feistel cipher, as shown in Fig. 2. The round takes L_i and R_i from previous round (or the initial permutation box) and creates L_i and R_i, which will go to the next round (or final permutation box). The R_i and round key K_i goes in the 'function' block whose output is then XOR with the L_i .These XORed bits are then swapped with the original R_i that become left half (L_{i+1}) data of the next round [9].

2.3. DES function:

The DES function applies a 48-bit key to the rightmost 32 bits (R_i) to produce a 32-bit output [10]. This function is made up of four sections: an expansion permutation-box, a whitener XOR (that adds key), a group of S-boxes, and a straight permutation-box as shown in Fig. 3.

The expansion p-box takes a block of 32 bits as input and yields a block of 48 bits as output, the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the table:

2.4 Key Generation

In key generation, 56 bits of the key are selected from the initial 64 bits by Permuted Choice 1 (PC-1), the remaining eight bits are either discarded or used as parity check bits. The 56 bits are then divided into two 28-bit halves; each half is thereafter treated separately. In successive rounds, both halves are rotated left by one or two bits (specified for each round), and then 48 subkey bits are selected by Permuted Choice 2 (PC-2), 24 bits from the left half, and 24 from the right [11]. The 56 bits are also passed on to the next round for further keys to be generated as shown in figure 4. The key schedule for decryption is similar, only the subkeys are in reverse order as compared to encryption. Apart from that change, the process is the same as for encryption [12].

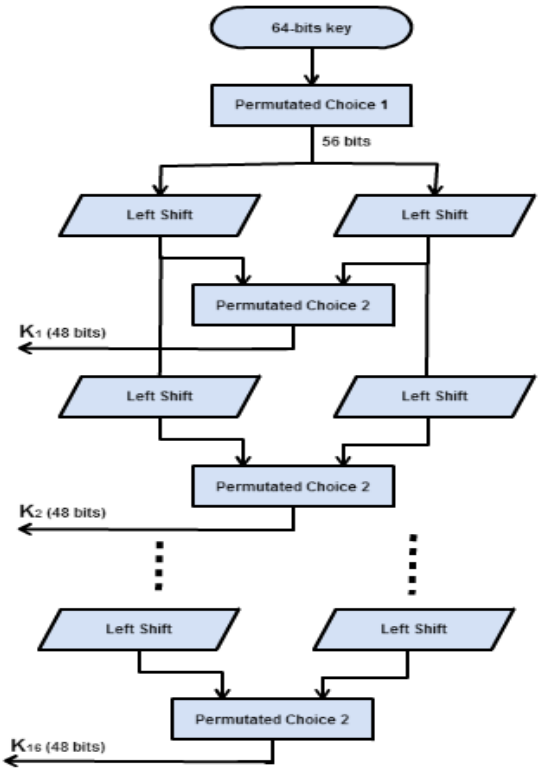

Figure 4: Key Generation

3. Implementation

In our design, we have used system generator [6] to develop single DES round as a separate subsystem, and then converted the 16 rounds in an overall subsystem as shown in figure 4. In each round the 64 bits data block is split into two 32 bit blocks namely left (Ln) and right (Rn) after which data expansion, key XOR with data, compression of data using S-Boxes, permutation and Xoring it with left side (Ln) and swapping it with right side (Rn) takes place. The 64 bits data is passed through the IP (initial permutation) and IP^{-1} (Inverse permutation) before going in and coming out of the rounds block [13].

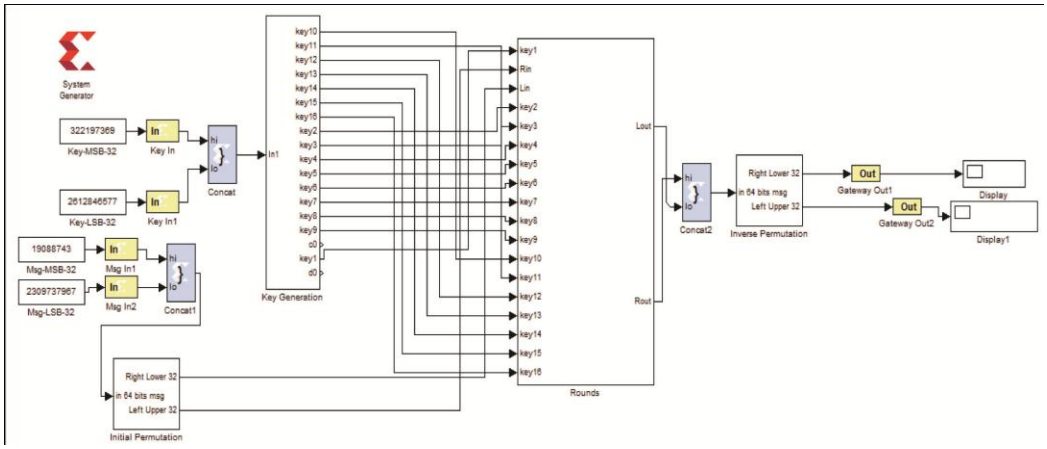

Figure 5: DES System Generator Architecture

3.1. Key Generation:

The 64 bits key enters the key generator sub system which consists of 16 rounds itself. Except the first round, each round has two inputs and three outputs. In the first round the 64 bits passes through PC1 (permutated compression 1) which produces 56 bits in two halves i.e. C_o and D_o which then goes in the circular shift blocks where the iteration takes place, then the output bits goes into second PC box which further reduces the bits to 48 bits producing them in two halves, which are then combined afterwards giving us our first key, the 56 bits produced after circular shifts goes into the second round of key generation to produce the second round key and so on producing 16 keys in total in the whole process as shown in figure 6.

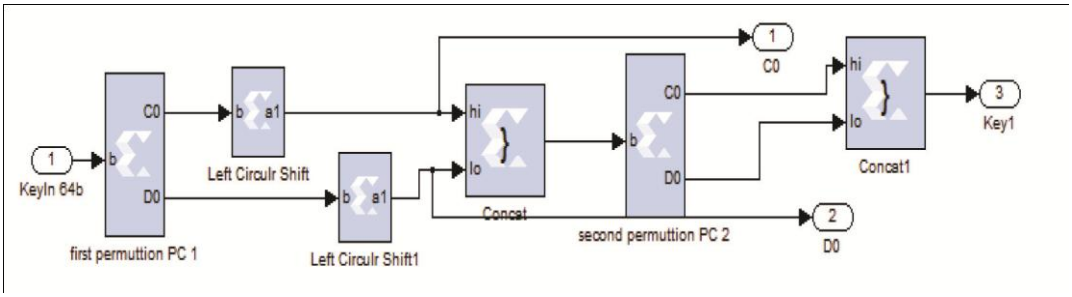

Figure 6: Structure of key gen. block

3.2. Permutation:

The permutation of data was done by using the 'bit basher' block in which the bits were arranged according to their respective tables.

3.3. Rounds Generation:

The rounds too consists of 16 sub rounds just as the key generation, each sub round has 3 inputs i.e. (the L_{in} and R_{in}) and the round key as shown in figure 7.

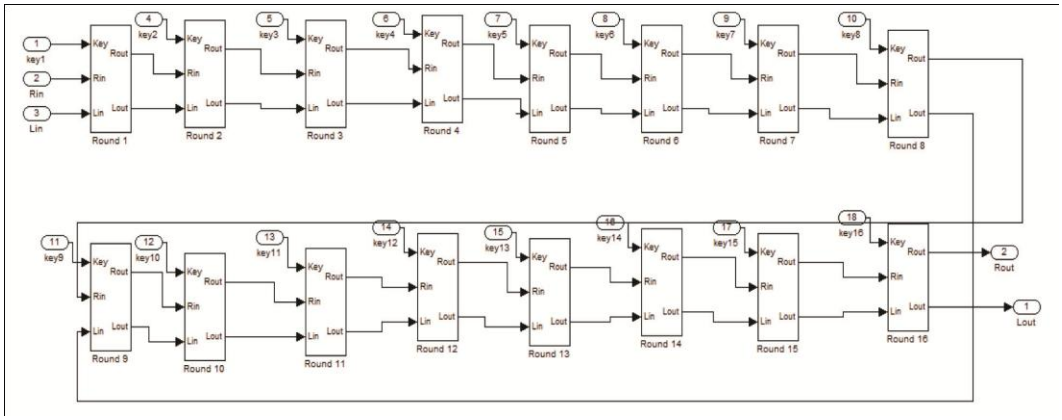

Figure 7: Internal structure of Rounds block

In each sub round as shown in figure 8, the R_{in}32 bits goes through the Expansion permutation, which expands the 32 bits to 48 bits which are then XOR with the 48 bits of round key, the output of the XOR in divided into eight sets of six bits, the six bits in each set determine the row and column of the s-box (the first and last bit determines the row and the remaining four bits determine the column) the output of the eight sets are fed into the eight S-boxes, each S box produces a single output according to the row and column it is mapped on, the eight outputs are then fed into the P-box (permutation box) that produces 32 bits which are then XOR with the L_{in} bits the output of XOR is then swapped with the L_{in} and becomes the L_{in} for the next round and the whole process goes on for 16 rounds. The output of the Round generation, after going through IP^{-1} is the ciphertext [14].

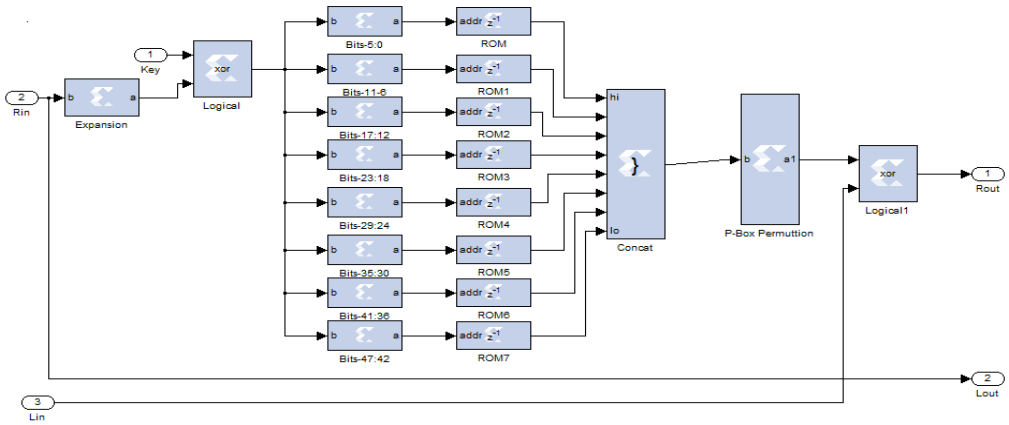

Figure 8: Structure of a single sub-round

3.4. S-Boxes:

In the sub-round block, after including the effect of key, the 48 bits data is compressed to 32 bits. This conversion is done using S-Box. The implementation of S-Boxes is performed using Xilinx single port ROM as shown in Figure 9.

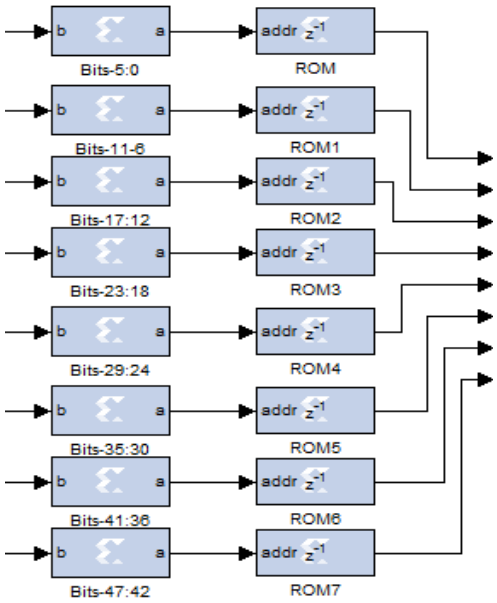

Figure 9: S-box Implementation

ROM consists of single input (address) and single output i.e. mapped value on input address. The type of memory is selected for ROM is Block RAM. One S-Box consists of a table having 64values, so the depth of memory is selected up to

64. The table of 64 values is put into ROM initial value vector. Whatever be the address given as input to ROM, it gives the value placed at that address as output. This output is a decimal value, which is concaved and then passed on to the P-box [14].

4. Implementation Results

Xilinx System Generator implements the High Level Language design of DES. The design is constructed in the block form and then its synthesizable Verilog code and hardware co-simulation block was generated by the system generator block. The design is simulated over System Generator, Xilinx ISE 14.4 and has been implemented over XC3S1600E Spartan-3E FPGA for real time results.
The hardware resources used by our design are given in Table 4.

Table 4: Hardware Resources

Slices	FlipFlops	BRAMs	LUTs	IOBs	Mults/DSP48s	TBUFs
1344	32	120	1408	192	0	0

Table 5: Comparison with other implementations

Implementation	Device	Data Paths	Freq. (f)	Throughput
David C. Fel [7]	DEC 3100	32	-	1.098 (Mbps)
David C. Fel [7]	Sun 4/280	32	-	1.227 (Mbps)
R. Stephen [8]	TMS320C6211	32	150 MHz	38.8 (Mbps)
R. Stephen [8]	TMS320C6201	32	200 MHz	52.4 (Mbps)
Our Implementation	Spartan3e 1600e	64	310.14 MHz	1.24 (Gbps)

As its clear from the table, DES has been implemented on many different platforms and techniques like CUDA [7] and OpenCL[8] but in our implementation we have used High level language tool i.e. ISE Design tool 14.4 System generator that maps directly the design in HDL code and it has more flexibility as compared to other implementations, furthermore we have generated a separate block for key generation, instead of giving fixed keys as is the case of above mentioned implementations.

5. Conclusion:

In this paper, we have discussed the High Level Language implementation of DES. Our design is efficient in comparison to other software implementations and it utilizes less hardware resource on FPGA and takes less development time.

References

[1] FU Li, PAN Ming, (2009) "A Simplified FPGA Implementation Based on an Improved DES Algorithm",School of Computer Science and Control Guilin University of Electronic Technology Guilin, China, 2009

[2] William Stalling, "Cryptography and Network Security Principles and Practices", Prentice Hall, sixth edition.

[3] MicroBlaze Development Kit Spartan-3E 1600E Edition User Guide, available athttp://www.xilinx.com/support/documentation/user_ guides/ug190.pdf

[4] FIPS-46-3, "Federal Information Processing Standards Publication FIPS-46-3, Data Encryption Standard (DES)", http://csrc.nist.gov/ publications/fips/fips46-3/fips46-3.pdf

[5] Shoaib Mughal, Dr. Arshad Aziz, "High Level Implementation of DES on fpga", Department of Electrical Engineering, NUST, http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

[6] System Generator for DSP. Getting Started Guide. *Xilinx*.

[7] D. Noer, A.P. Engsig-Karup and E. Zenner, (2011) "Improved Software Implementation of DES Using CUDA and OpenCL", 2011

[8] Mentor Graphics, (2008) "Modelsim Data sheet," available at http://modelsim.s3.amazonaws.com/modelsim-sedatasheet.pdf

[9] Prasun Ghosal, Malabika Biswas, and Manish Biswas, (2010), A Compact FPGA Implementation of Triple-DES Encryption System with IP Core Generation and On-Chip Verification, Proceedings of the 2010 International Conference on Industrial Engineering and Operations Management Dhaka, Bangladesh, January 9 – 10.

[10] Larry T. McDaniel III , (2003), An Investigation of Differential Power Analysis Attacks on FPGA-based Encryption Systems, Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Masters of Sciences in Electrical Engineering.

[11] Xu Guo, (2012), Secure and Efficient Implementations of Cryptographic Primitives, Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering, May, Blacksburg, Virginia.

[12] L.Thulasimani, M.Madheswaran, (2010), Design And Implementation of Reconfigurable Rijndael Encryption Algorithms For Reconfigurable Mobile Terminals, L. Thulasimani et. al. / (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 04, 2010, pp. 1003-1011.

[13] Ramya Krishna Addluri, (2014), An Efficient Implementation of the Blowfish Encryption Algorithm, A thesis submitted to the Graduate School of the University of Cincinnati In partial fulfillment of the requirements For the degree of Master of Science In the Department of Electrical and Computer Engineering Of the College of Engineering and Applied Sciences.

[14] Rehan Shams, Fozia Hanif Khan, Umair Jillani and M. Umair, (2012), Introducing Primality Testing Algorithm with an Implementation on 64 bits RSA Encryption Using Verilog, SSU Res .J. of Engg. & Tech. Vol. 2. Issue 1, pp. 12-17.