International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 4, April - 2013

Design Of Cryptographic Processor For Security Algorithm Operations

1 M. Praveen Kumar 2 R. Ashwitha 3 N. Jyosna 4 M. Pavani

#1Assistant Professor, Department of Electronics and Communication Engineering, Vignana Bharathi Institute of Technology, Aushapur, Ghatkesar, RangaReddy, (A.P.), India.

#2#3#4UG Students(ECE), Department of Electronics and Communication Engineering, Vignana Bharathi Institute of Technology, Aushapur, Ghatkesar, RangaReddy, (A.P.), India.

Abstract
In this paper we are performing various arithmetic and logical operations such as addition, multiplication, shift operations, matrix multiplications, fixed coefficient multiplier, mix column transformation, multiplier X (2X+1), circular shift operations for arbitery irreducible polynomial. We are performing these operations in a cryptographic processor. The main aim of our project is to reduce the power consumption, increase the speed of operation and reduce the programming length.

1. INTRODUCTION

As data hacking is more prominent during the data transmission because of increase in the technology here comes the necessity of the secure data transmission. Cryptography is the science of protecting the data, which provides means and methods of converting data into unreadable form. Cryptography refers to encryption and decryption of data. It’s used to maintain confidentiality, data integrity, non-repudiation, authentication of data. Cryptography is broadly classified into two types.

a. Symmetric Key Cryptography.
b. Asymmetric Key Cryptography.

Symmetric Key Cryptography uses a single key for both the encryption and decryption of data. These may use either stream ciphers or block ciphers. In stream ciphers a single bit data is encrypted where as in a block a group of bits are encrypted as a single unit.

Asymmetric Key Cryptography uses different keys for both encryption and decryption of data in which one is a public key and the other private key. If the data is encrypted by a public key only the corresponding secret private key is used to decrypt the data.

We can implement the above mentioned instruction in a normal processor also but it has more number of addressing modes, lower throughput and more power consumption. So we have designed our own cryptographic processor.

In the processor designed by us we don’t have addressing modes we just use load and store instructions for performing the calculations. We are using 32 bit registers for performing the operations and storing the results. Hardware is simplified and the machine cycles time is reduced.

The rest of the paper states about the instructions from algorithms, operations performed, cryptographic processor, description about the modules and final our conclusion about the theme of the project with the references which have helped for successful completion of the project.

2. CRYPTOGRAPHIC ALGORITHMS

In our paper we have used the different instructions and we have taken those instructions from AES, RC6 algorithms.
AES uses block cipher symmetric key cryptography. It has fixed block size of 128 bits and the key size of 128,192,256 bits. Key size refers to number of repetitions required to convert plain text into cipher text. We have taken addition, multiplication, fixed coefficient multiplier, mix column transformation, circular shift operation, matrix multiplication, Shift operations. RC6 also has a block size of 128 bits but it can be parameterized to support a wide variety of key lengths, key sizes. We have taken multiplier X (2X+1) instruction for performing this operation.

3. OPERATIONS

Modulo addition

The addition operation is performed by adding the coefficients of corresponding powers of polynomials. The addition is performed by doing XOR (denoted by ⊕) between the numbers i.e. 0⊕0=0, 1⊕0=1, 1⊕1=0.

Modular multiplication 2^8

In this normal multiplication is performed and the resultant is done modulo with an irreducible polynomial. For the AES algorithm the irreducible polynomial is
M(x) =x^8.

Mix column transformation

In this the 32 bit data is divided into groups(say 4) then for each column one bit is shifted and then xor operation is done with it again.
For example
input[31:0]A
mixcolumn={C3,C2,C1,C0};
a0=A [7:0];
a1=A [15:8];

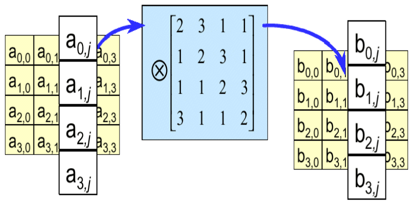

$a_{0,0}$ $a_{0,1}$ $a_{0,j}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,j}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,j}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,j}$ $a_{3,3}$
⊗
2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
$b_{0,0}$ $b_{0,1}$ $b_{0,j}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,j}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,j}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,j}$ $b_{3,3}$

Fig.1 Mix column Transform

a2=A [23:16];
a3=A [31:24];
C0 = (a0<<1) ^ ((a1<<1) ^a1) ^ (a2) ^ (a3);
C1 = (a0) ^ (a1<<1) ^ ((a2<<1) ^a2) ^ (a3);
C2 = (a0) ^ (a1) ^ (a2<<1) ^(a3<<1)^ (a3);
C3 = (a0<<1) ^a0) ^ (a1) ^ (a2) ^ (a3<<1);

Matrix Multiplication

In this we take 2 different matrices and the corresponding multiplication is done just by shifting it.
For example consider a matrix
Input [31:0] A
Input [31:0] B;
Matrix multiplication= {d3, d2, d1, d0};
a0=A [7:0];
a1=A [15:8];
a2=A [23:16];
a3=A [31:24];
b0=B [7:0];
b1=B [15:8];
b2=B [23:16];
b3=B [31:24];
d0= (a0*b0) ^ (a3*b1) ^ (a2*b2) ^ (a1*b3);
d1= (a1*b0) ^ (a0*b1) ^ (a3*b2) ^ (a2*b3);
d2= (a2*b0) ^ (a1*b1) ^ (a0*b2) ^ (a3);
d3= (a3*b0) ^ (a2*b1) ^ (a1*b2) ^ (a0*b3);

4. CRYPTOGRAPHIC PROCESSOR

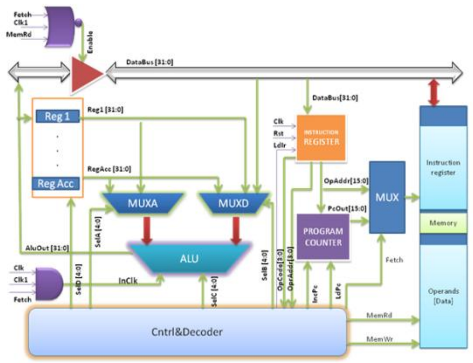


Fig.2 Cryptographic processor

This is the architecture of the cryptographic processor; this is a 32 bit pipelined processor. Here there is a separate memory for load/store instructions.

This Harvard style of architecture can either be used with two completely different memory spaces, a single dual port memory space with separate data and instruction. Three stages of pipelining have been incorporated in the design which increases the speed of operation.

The processor presented instruction set and uses a single instruction – single data (SISD) execution order.

Its main Blocks are

1. Sixteen 32-bit general purpose registers.
2. Multiplexers.
3. Instruction Register.
4. Program Counter.
5. Memory.
6. Buffer.
7. Control and Decoder.
8. ALU with basic arithmetic and logical operations.

General purpose registers:

General purpose registers (GPR’s) store and save operands and result during program execution. ALU and memories must be able to write/read those registers, so a set of sixteen 32 bit registers are used. The values in the any two registers are the operands for the ALU which perform the operations and other registers store the results of each operation after the execution. Fig.3 indicates how the registers are filled with the ALU output based on the selection line from the control and decoder.

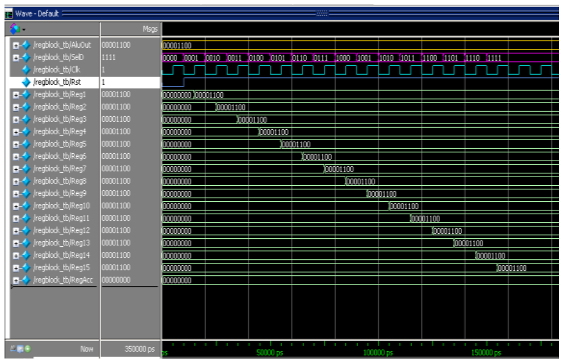


Fig.3 General Purpose registers output waveform

Fig.4 shows how the data flows in the General Purpose Registers. It receives the data from control and decoder and gives its output to MUXA and MUXD which performs further operations.

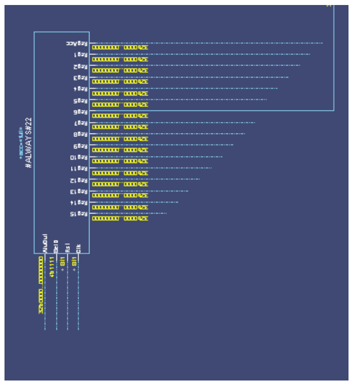


Fig.4 General Purpose registers Data flow

Multiplexers:

Here we use 3 multiplexers namely MUXA, MUXD, MUX. MUXA and MUXD are used to select one register from the 16 registers. The selected registers are used as operands to perform operations in the ALU. MUX is used to select one of the outputs from instruction register or program counter. The selected output is given as input to the memory.Fig.5 represents the MUXA output waveform through which we can see how the Registers are selected based on the selection lines.

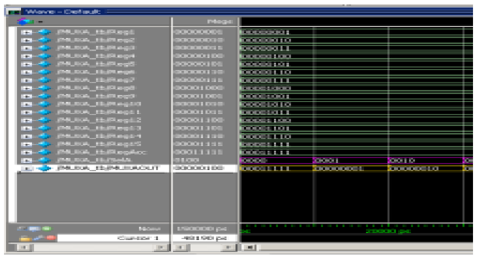


Fig.5 MUXA output waveform

Fig.6. indicates the data flow of MUXA where it receives the input from the Register block and gives its output to the ALU.

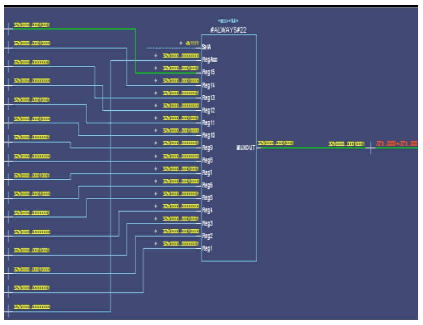


Fig.6 MUXA dataflow

Fig.7 indicates the output waveform of MUXD. The purpose of using MUXD is to fetch the operands as we require at least two operands to perform the operations.

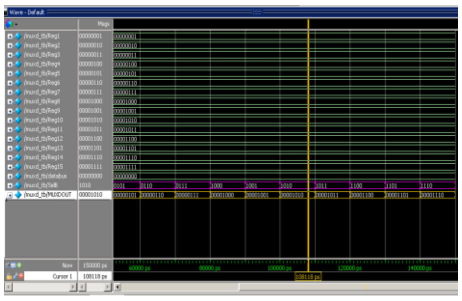


Fig.7 MUXD Output Waveform

Fig.8 indicates the dataflow of MUXD where it receives input from Register block and gives output to ALU.

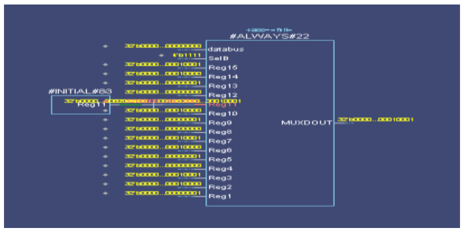


Fig.8 MUXD Dataflow

Fig.9 indicates the output waveform of MUX. Here based on the Fetch input the operational address or program counter output is selected.

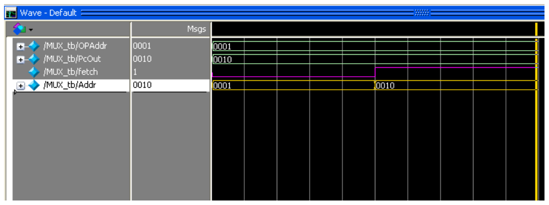


Fig.9 MUX Output waveform.

Fig.10 indicates the dataflow of MUX where the data flows from instruction register to the memory module.

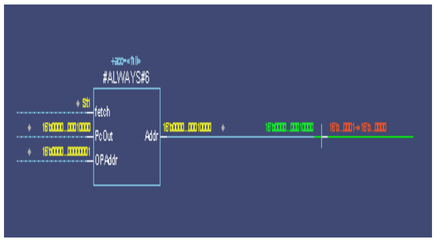


Fig.10 MUX dataflow

Instruction Register:

Instruction register store the instruction read from the program memory and give it as an input to control and decoder. It separates the Operation Code, Source register Address, Operand address and operands. These values are given to the general purpose registers, Multiplexers and ALU to execute the command. Fig.11 indicates the output of Instruction register.

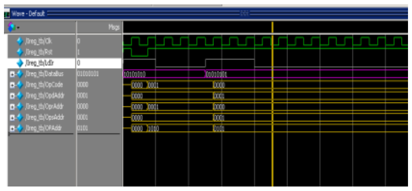


Fig.11 Instruction Register Output Waveform

Fig.12 indicates how the data flows from the data bus to the program counter and also the MUX based on the selection lines from the control and decoder. It also indicates how the source and destination address is moved into the control and decoder.

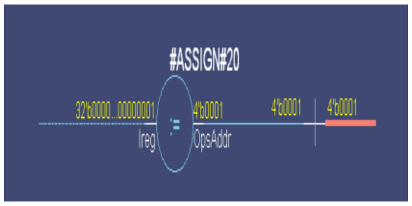


Fig.12 Instruction Register Dataflow

Program counter:

In most processors, PC is incremented after fetching an instruction, and holds the memory address of the next instruction that would be executed. Instructions are usually fetched sequentially from memory, but control transfer instructions change the sequence by placing a new value in PC. It takes input from instruction register and control and decoder and produces the output and gives it to MUX.Fig.13 indicates the Program counter Output.

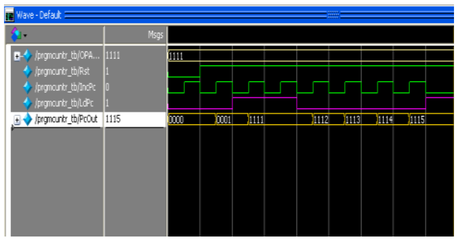


Fig.13 Program Counter Output Waveform

Fig.14 indicates the dataflow of program counter where PC output is loaded with operational address or else it is incremented based on the Reset and Load PC value.

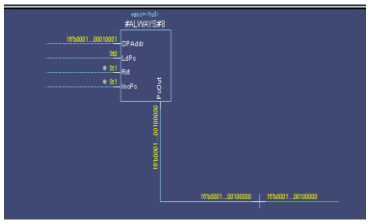


Fig.14 Program Counter Dataflow

CONTROL AND DECODER:

This works on the principle of finite state machines (FSM). The main operations performed by it are reset, instruction fetch, decode& load, address setup, operand fetch and store the result. It will be in ideal state when no operations are performed.Fig.15 indicates the output of the Control and decoder.

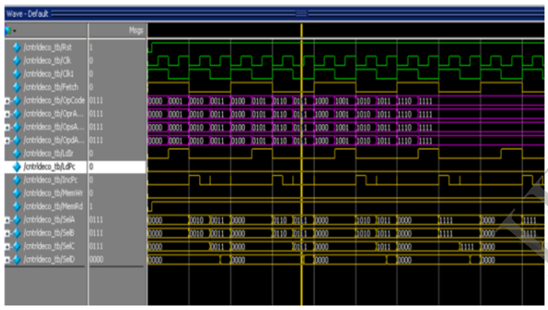


Fig.15 Control and decoder output waveform

Fig.16 indicates the how the selection lines are emerged from the Control and decoder to different modules such as SelA for MUXA, Sel B for MUXD, SelC for ALU and SelD for Register block, Fetch for MUX.

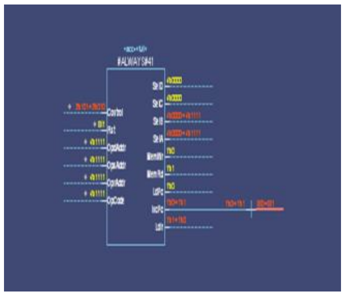


Fig.16 Control and decoder dataflow

Memory:

In this Architecture Separate memory for instructions (program) and data is used. Simultaneous Accesses to memory can be performed by using different stages of Pipelining.Fig17 indicates the output of the memory block.

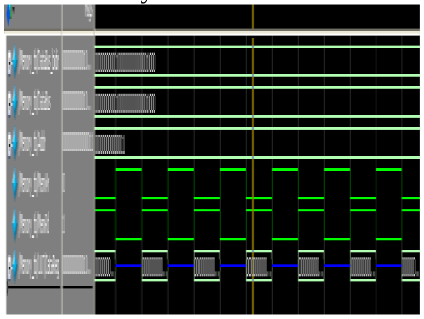


Fig.17 Memory output waveform

Fig.18 indicates how the data is read from a Text file and it stores the operands in the register block.
The Obtained outputs are again store back to the memory through data bus from Register block.

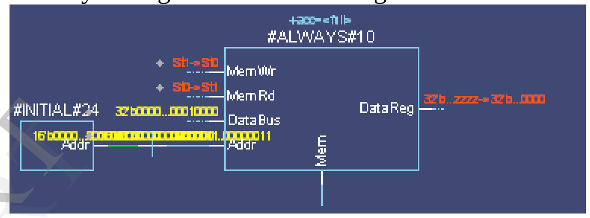


Fig.18 Memory data flow

ALU:

ALU fetches the data from memory and performs the operations according to control and decoder input, performs the operations as mentioned results are stored in the register block.Fig.19 indicates the output waveform of ALU

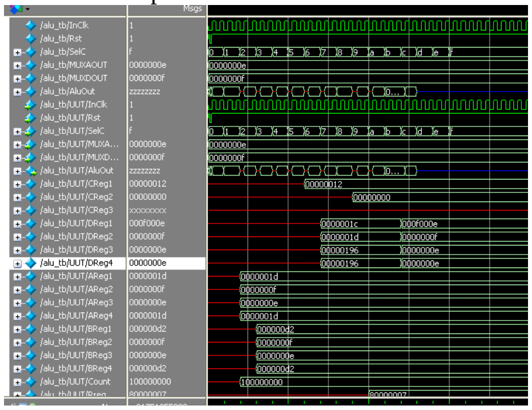
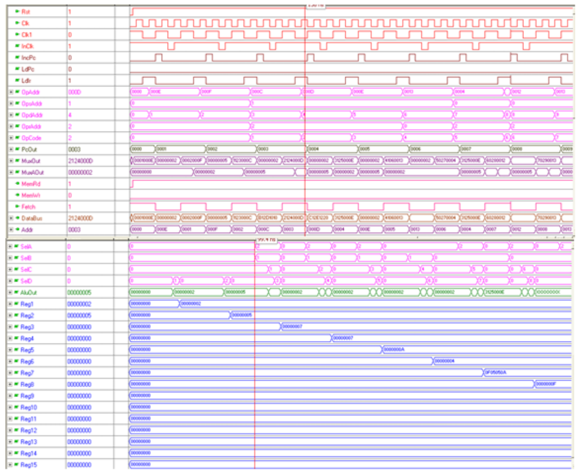


Fig.19 ALU output waveform

TOP MODULE: This links all the modules and produces the output.Fig.20 indicates the output of the Top module. Here we can see that various operations are performed taking operands from the Register Block and outputs are stored in the Registers.



5. CONCLUSION:

Thus we have designed a 32 bit cryptographic processor where different arithmetic and logical operations are performed using different symmetric key cryptographic algorithms. We have done the simulations using Model Sim, Active HDL and verified through Xilinx. By this we have reduced the power consumption and it is achieved in a lower area with less propagation delay by reducing the number of instruction cycles.

6. FUTURE SCOPE:

In order to perform differential and integration operations we need to use Advanced pipelining techniques to this processor which is the future scope of the project.

7. REFERENCES:

[1] Antonio H. Zavala "*RISC Based Architecture for Computer Hardware Introduction* **Edición**,, 2011 **IEEE.**

[2] NIST, "*Advanced Encryption Standard (AES), (FIPPUB 197)*", November 26, 2001, http://csrc.nist.gov/publications/.

[3] A. Rudra et. al., "*Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic*", Proc.CHES2001, LNCS *Vol. 2162, pp.175-188, 2001*.

[4] Rohit Sharma, Vivek Kumar Sehgal, Nitin Nitin1, Pranav Bhasker, Ishita Verma, 2009, "*Design And Implementation Of 64-Bit RISC Processor Using Computer Modeling And Simulation*, pp. 568 – 573.

[5] R. Uma / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com *Vol. 2, Issue 2, Mar-Apr 2012, pp.053-058 Design and Performance Analysis of 8-bit RISC Processor using Xilinx Tool*

[6] IEEE TRANSACTIONS on very large scale integration (VLSI) systems, vol. 18, No 8, August 2010 1145 *A High-Performance Unified-Field Reconfigurable Cryptographic Processor* Jun-Hong Chen, Ming-Der Shieh, *Member, IEEE*, and Wen-Ching Lin.

[7] *FPGA Implementations of the RC6 Block Cipher* Jean-Luc Beuchat Laboratoire de l'Informatique du arall´elisme, Ecole Normale Sup´erieure de Lyon,46, All´ee d'Italie, F–69364 Lyon Cedex 07,Jean-Luc.Beuchat@ens-lyon.fr.

[8] *Some Guidelines for Implementing Symmetric-Key Cryptosystems on Reconfigurable-Hardware* Arturo ³az-P¶erez, Nazar A. Saqib, and Francisco Rodrguez-Henriquez Computer Science Section, Electrical Engineering Department Centro de Investigacion y de Estudios Avanzados del IPN Av. Instituto Politecnico Nacional No. 2508, Mexico D.F.*f*nabbas@ computacion.cs.cinvestav.mx, adiaz, Francisco @cs. cinvestav.mx*g*.

[9] Imyong lee, Dongwook Lee, Kiyoung choi "ODALRISC: *A Small, Low power and Configurable 32-bit RISC processor"* International SOC design conference 2008.

[10]. Wayne Wolf, *FPGA Based System Design* , Prentice Hall, 2005.

[11] R. Razdan and M.D. Smith, "*A High-Performance Micro architecture with Hardware-Programmable Functional Units*,"Proc. Micro-27, IEEE Computer Society, 1994, pp. 172-180.

[12]. Vincen t P. Heuring, and Ha rry F. Jordan, "*Computer Systems Design and Architecture*", 2n d E dition, 2003.