ECE520

ASIC DESIGN

PROJECT REPORT

RC6 ENCRYPTION - DECRYPTION ENGINE

Completed by

S N MOHILE AND A K VARMA

snmohile and akvarma@unity.ncsu.edu

Instructor:

DR. P D FRANZON

NORTH CAROLINA STATE UNIVERSITY

TABLE OF CONTENTS

ABSTRACT

This report describes the successful implementation of the RC6 block cipher using the verilog hardware description language. The main aim of this project is to have a fast encryption-decryption engine. Although the complete design was not synthesized, an approximate speed of the engine was obtained by the synthesis of separate modules.

The main feature of our design are:

1) The use of fast adder and multiplier.
2) Design reuse -We created modules of Adder. Multiplier etc. Which allowed us reduce the hardware by reusing or instantiating the modules.
3) A well partitioned design - allowing us to work in modules and giving the whole design a structure.

Although we couldn't synthesize our complete design, we were capable in synthesizing all the individual modules. The Clock cycle for each separate modules and the corresponding speed in MHz is shown in the table below. Also shown is the area of the individual modules.

Module	Clock cycles(ns)	Speed(MHz)	Area(micrometer^2)
Adder/subtractor	49	20.4	314532
Multiplier	73	13.69	2611089
Parassign	8.5	117.64	192186
Rotate	60	13.69	891855
Counter	9	111.11	73723.5

PROJECT OBJECTIVE

The main objective of the project is to have a fast encryption decryption engine using the RC6 algorithm.
To achieve the above mentioned goal, we designed our own adders and multipliers and a module that implemented the rotate function.
The ripple carry adder from the design-ware library is too slow for 32 bit addition - as such - we designed our own 32 bit carry look ahead adder about which we talk about later in our report.
The multiplier we designed is a parallel multiplier shift which is also significantly faster than the multiplier available in the design ware library of synopsys.

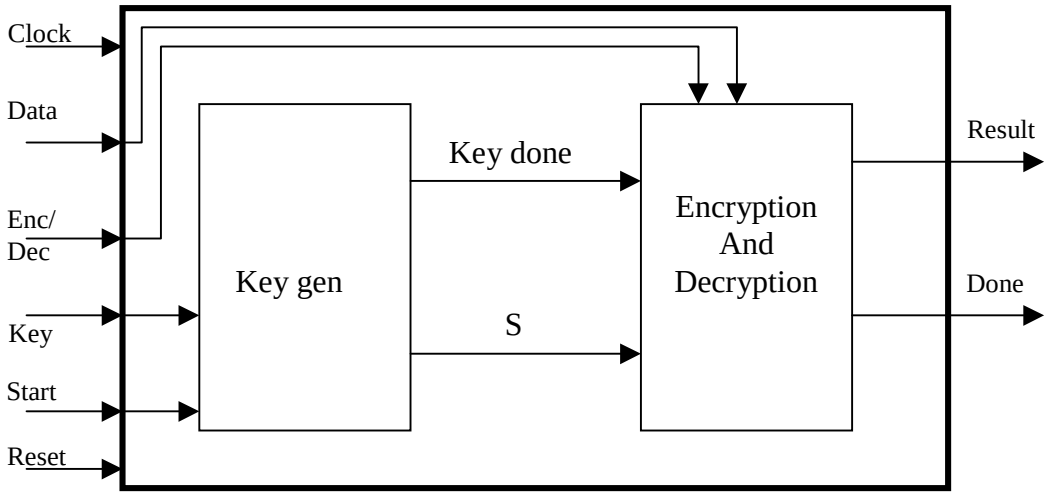
The rotate function was achieved by designing the Barrel shifter from Douglas Smith's HDL Chip Design.

INTRODUCTION

As the internet gathers more importance in today's world, the importance of secure data transfer becomes even more important. Credit Card numbers, bank account details etc - are all now being transferred over the world wide web. With the advancement of technology, the possibility of breaking todays encryption techniques are becoming increasingly easier. As such the National Institute of Standards and Technology (NIST) has been working with the industry and the cryptographic community to develop an Advanced Encryption Stander(AES) that is both secure and fast.
RC6 block cipher from the RSA laboratories is one such new technology that has been chosen to become the 'next generation' encryption standard.

FEATURES OF RC6

RC6 is the successor to RC5. After a greater understanding of RC5's structure and operations and possible 'theoretical' attacks, cryptologists at RSA labs came up with RC6. RC6 is said to be one of the most promising algorithm that is both secure and fast.
RC6 is specified as RC6-w/r/b where w is word size, r is the number of rounds and b is the length of encryption key in bytes.
Values of w/r/b play an important in Encryption and Decryption. The degree of encryption is based on the number of rounds(r) in RC6 algorithm. Thus a user requiring high encryption would prefer higher values of r. Values of w and b depend on many reasons and will be discussed in implementation issues.
The values considered in this project are w =32, r =20, b =16
Our design is summed up in the following block diagram



RC6 can be divided into 3 distinct subdivisions.

1) the key generation module
2) the encryption module and
3) the decryption module

- **THE KEY GENERATION**

In Key scheduling algorithm more words are derived from the user-defined key for the use for encryption and decryption. The user supplies a key of b bytes. The key is derived using two Magic constants P and Q. The value of P is B7E15163 and is derived from the binary expansion of e-2, where e is base of the natural logarithm function. The value of Q is 9E3779B9 and is derived from (phi)-1, where (phi) is the Golden ratio.
The algorithm can be mathematically represented as follows:

INPUT: User-supplied b byte key reloaded into 4-word array, L[0,..,3].
Number r=20 of rounds

OUTPUT: 32 bit round keys S[0,...,43]

ALGORITHM:

```
S[0]=P
for i=1 to 43 do
        S[i]=S[i-1]+Q
X=Y=i=j=0
v=3*max{4,44}=132
for s=1 to 132 do
{
        X=S[i]=(S[i]+X+Y)<<<3
        Y=L[j]=(l[j]+X+Y)<<<(X+Y)
        i=(i+1) mod 44
        j=(j+1) mod 4
}
```

- **ENCRYPTION**

The plain text is stored in 4 registers A,B,C,D each of 32 bit. The 44 subkeys are used to encrypt the plain text to give cipher text. The first two and last two steps as seen in the algorithm are called pre-whitening and post-whitening steps respectively. Without these steps the plain text reveals part of the input to first round of encryption. These steps help to disguise this and thus the encryption is made complex.
The algorithm can be mathematically represented as follows:

INPUT: Plain text stored in four 32-bit registers
Number r=20 of rounds
32-bit round keys S[0,...,43]

OUTPUT: Cipher text stored in A,B,C,D

ALGORITHM:

```
B=B+S[0]
D=D+S[1]
for i=1 to 20
{
    t=(B*(2B+1))<<<5
    u=(D*(2D+1))<<<5
    A=((A exor t)<<<u)+S[2i]
    C=((C exor u)<<<t)+S[2i+1]
    (A,B,C,D) = (B,C,D,A)
}
A=A+S[42]
C=C+S[43]
```

- **DECRYPTION**

The Cipher text is converted to plain text again using the 44 subkeys. The decryption process is just the reverse of encryption process.

The algorithm can be mathematically represented as follows:

INPUT: Cipher text stored in four 32-bit registers
Number r=20 of rounds
32-bit round keys S[0,...,43]

OUTPUT: Plain text stored in four registers

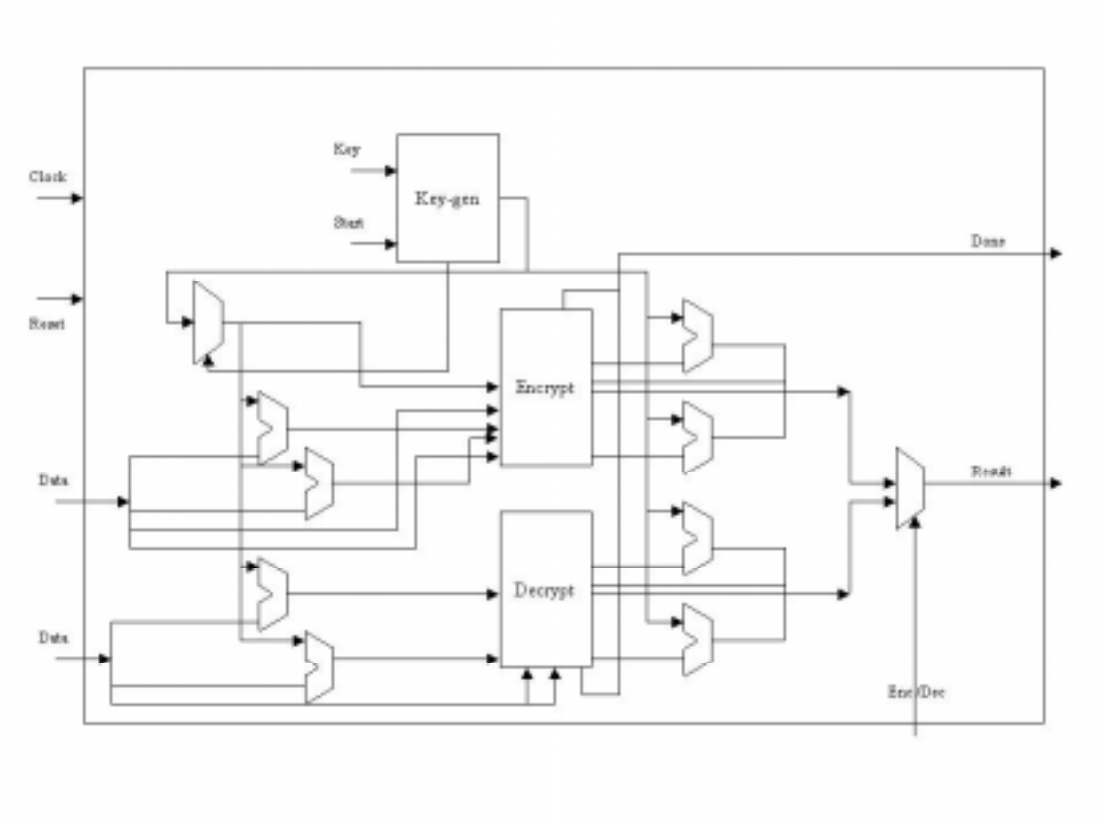
ALGORITHM:

```
C=C-S[43]u=(D*(2D+1))<<<5
B=B-S[42]

for i=20 downto 1
{
    (A,B,C,D) = (D,A,B,C)
    u=(D*(2D+1))<<<5
    t=(B*(2B+1))<<<5
    C=((C-S[2i+1])>>>t) exor u
    A=((A-S[2i])>>>u) exor t
}
```

D=D-S[1]
B=B-S[0]

Our Entire Encrypt Decrypt module is shown in the following block diagram.



IMPLEMENTING THE RC6 ALGORITHM

The RC6 algorithm can be distinctly divided into 3 different subdivisions as mentioned above. Our main approach for this design was to reuse as much design as possible. As such we designed

1) a module that does the rotate, (used 10 times)
2) a multiplier , (used 08 time)
3) an adder,(used 16 time)
4) a module that does the parallel assignment in encryption and decryption (used 2 times)
5) the logic within the for loop in the encryption module (used 20 times)
6) the logic within the for loop in the decryption module (used 20 times)

DESIGNING THE RC6 ENGINE

Initially we started with the C-code of RC6 Encryption-Decryption and compared our results with the standard results. Then we proceeded with implementing the Verilog code with '+','*' etc and compiled our code to get accurate results.
Then our main objective was to reduce the hardware and increase the speed of the circuit. We thought of building separate modules for addition/subtraction, multiplication, rotate, parallel assignment etc and using the same design by instantiating it. Thus we finally implemented a Verilog code with separate modules. We also checked our results for random data input.

PERFORMANCE OF THE ENCRYPTION/DECRYPTION ENGINE

- **PERFORMANCE OF THE ADDER/SUBTRACTOR MODULE**

We implemented a 32-bit carry look-ahead adder subtractor by recursively expanding the carry term to each stage. Recursive expansion allows the carry expression for each individual stage to be implemented in a two-level AND-OR expression. This reduces the carry signal propagation delay (the limiting factor in a standard ripple carry adder) to produce a high-performance addition circuit.
Clock period of Add/Subtract unit is 49 ns
Speed of the Add/Subtract module is 20.4 MHz.
Area of the unit 314532 micrometer ^{2}

- **PERFORMANCE OF THE ROTATE MODULE**

We implemented a 32 bit right-left rotate by concatenating 1'b0 with 32-bit input word. After every one shift the lsb i.e 0^{th} bit gets the value of msb i.e 32^{nd} bit and this repeat for a required number of shifts.
Clock Period of Rotate unit is 60 ns
Speed of Rotate module is 16.66 MHz.
Area of the unit 891855 micrometer ^{2}

- **PERFORMANCE OF THE MULTIPLIER MODULE**

We have implemented a shift and add multiplication algorithm. The process consists of looking at each successive bit of the multiplier in turn, starting with the lsb. If the multiplier is a logic 1, the multiplicand is copied down, otherwise zeros are copied down. The number copied down in successive lines are shifted one position to the left from the previous number. Finally, the numbers are added and their sum provides the product

Clock Period of multiplier unit is 73 ns
Speed of multiplier module is 13.69 MHz.
Area of the unit 2611089 micrometer ^{2}

RESULTS OBTAINED

Presented here are 10 values of the plain text, output result and user input key.
The user input key and the plain text were generated using the Random function in verilog so that the program can be thoroughly checked for any bug.

Compiling source file "varma_mohile.v"
Highest level modules:
test_fixture

datain = 06b97b0d46df998db2c2846589375212values that were geneerated
dataout = 06b97b0d46df998db2c2846589375212
key = 12153524c0895e818484d609b1f05663

datain = 76d457ed462df78c7cfde9f9e33724c6
dataout = 76d457ed462df78c7cfde9f9e33724c6
key = 00f3e30106d7cd0d3b23f1761e8dcd3d

datain = 8932d61247ecdb8f793069f2e77696ce
dataout = 8932d61247ecdb8f793069f2e77696ce
key = e2f784c5d513d2aa72aff7e5bbd27277

datain = 96ab582db2a72665b1ef62630573870a
dataout = 96ab582db2a72665b1ef62630573870a
key = f4007ae8e2ca4ec52e58495cde8e28bd

datain = cb203e968983b81386bc380da9a7d653
dataout = cb203e968983b81386bc380da9a7d653
key = c03b228010642120557845aacecccc9d

datain = 0effe91de7c572cf118449230509650a
dataout = 0effe91de7c572cf118449230509650a
key = 359fdd6beaa62ad581174a02d7563eae

datain = 20c4b341ec4b34d83c20f378c48a1289
dataout = 20c4b341ec4b34d83c20f378c48a1289
key = e5730aca9e314c3c7968bdf2452e618a

datain = de7502bc150fdd2a85d79a0bb897be71
dataout = de7502bc150fdd2a85d79a0bb897be71

key = 75c50deb5b0265b6634bf9c6571513ae

datain = bf23327e0aaa4b1578d99bf16c9c4bd9
dataout = bf23327e0aaa4b1578d99bf16c9c4bd9
key = 42f2418527f2554f9dcc603b1d06333a

datain = 7c6da9f8dbcd60b7cfc4569fae7d945c
dataout = 7c6da9f8dbcd60b7cfc4569fae7d945c
key = 312307622635fb4c4fa1559f47b9a18f

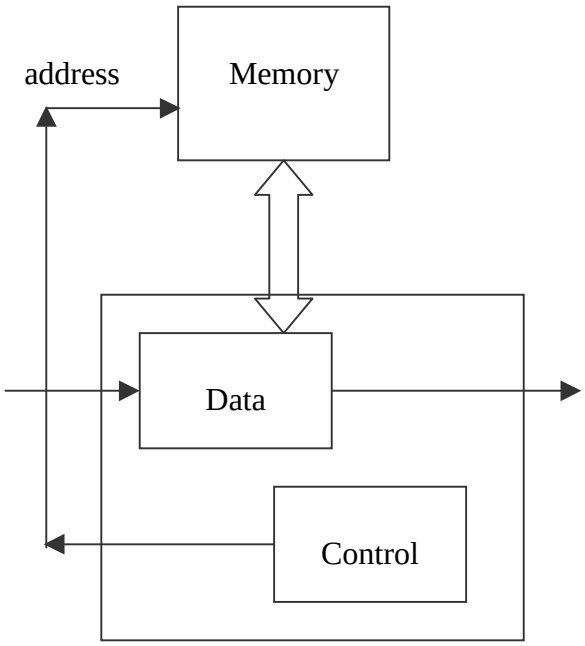
L1301 "varma_mohile.v": $finish at simulation time 91200
0 simulation events (use +profile or +listcounts option to count) + 3786 accelerated events
CPU time: 0.2 secs to compile + 7.3 secs to link + 1468.1 secs in simulation
End of VERILOG-XL 2.6.36 Apr 26, 2000 20:34:05

DESIRED IMPROVEMENTS IN THE DESIGN

The design we implemented was build keeping in mind the speed requirements of the encryption decryption engine. Some improvements that can be made to this design are

A) Implementation of the control logic with FSM s. This would clearly separate the control and data-path in the design. There could be a master FSM that controls the slave FSMs. The slave FSM can be present in each distinct module of the engine ie the Key Scheduler module, the Encryption module and the Decryption Module. We realize that making modifications in a design with FSM would be far easier than making modifications in a design without FSM.

B) Another modification that could be made - and one that is highly recommended - is removing the memory array from the main module and dumping it to the test fixture. This will vastly improve the synthesis time and results. Rather than wasting valuable resources in allocating memory, Synopsys can concentrate on more important issues of logic optimization and meeting timing and area constraint. This could have been done by passing the write values as outputs of the main module and also passing the read values as inputs of the main module. Along with these values, the address at which the memory values need to be written to and read from also needs to be passed. The design is illustrated in the diagram below.



DISCUSSION AND CONCLUSION

The RC6 Encryption/Decryption engine works for any given user key and plain text. We were successful in implementing the engine and testing the design for pre-synthesis but the design could not be completely synthesized, although we were able to synthesize each individual module of the project. The individual timing details and area are shown below.

Module	Clock cycles(ns)	Speed(MHz)	Area(micrometer^2)
Adder/subtractor	49	20.4	314532
Multiplier	73	13.69	2611089
Parassign	8.5	117.64	192186
Rotate	60	13.69	891855
Counter	9	111.11	73723.5

The main reason for not being able to synthesize our verilog code was that we had a memory array to synthesize in our key generation module. Memory arrays are simple to use but extremely hard to synthesize.
We could have implemented our memory outside the blocks that we wanted to synthesize. This has been suggested in the "Desired Improvements in the Design" part of the report. The following are included in the appendix part of the report.

1) Schematics for each synthesized module.
2) Waveform for a sample plain text and user key.
3) The view_command.log file for varma_mohile.v
4) Our Synthesis script
5) The varma_mohile.v – Our Verilog code for the RC6 encryptiuon/decryption engine
6) Test fixture

LIST OF REFERENCES AND BIBLIOGRAPHY

1. HDL CHIP DESIGN
   Douglas J. Smith Doone publications Jan 2000
2. VERILOG STYLES FOR SYNTHESIS
   Paul D. Franzon, D.R. Smith, Jan 2000.
3. Carry Lookahead adder from
   http://www.altera.com/html/atlas/examples/vhdl/v_cl_addr.html

APPENDIX

- **VERILOG FILE**
- **PRESYNTHESIS WAVEFORMS**
- **DC FILE - SYNTHESIS FILE**
- **POST SYNTHESIS SCHEMATICS OF DIFFERENT MODULES**
- **VIEW COMMAND.LOG FILE**
- **TEST FIXTURE**