Software Piracy Protection mechanism using FPGAs

Ravi Tandon
Department of Computer Science and Engineering,
IGET, Abhipur, Mohali, Punjab, India
Goyalravi12@gmail.com

Abstract: Software itself is a form of data and as such is vulnerable to thefts and misuse. Threats to particular piece of software can originate from a variety of sources. A substantial problem from an economic perspective is the unauthorized copying and the redistribution of applications, otherwise known as software piracy. For these reasons, software protection is considered one of the more important unresolved research challenges in security today. While many previously developed protection schemes have provided a strong level of security, their overall effectiveness has been hindered by the lack of transparency to the user in terms of performance overhead. The goal of this research is to make piracy more difficult by combining the advantages of both hardware and software techniques. There are all sorts of solutions: Embedded code in the software that disables copying, code that makes use of non-copy able aspects of the original disk, simple key lock hardware "dongles" that the software needs to run. This paper will determine some of the tools and methodologies that can be used to prevent software Piracy.

1. INTRODUCTION

There are basically 2 types of software Protection mechanisms. The first mechanism is by copy prevention and the second mechanism is by installation prevention. Copy prevention is prevention of software copying as is. Installation prevention is preventing any user to install the software without knowing certain things. The implementation details will be discussed in the following subsection.[1]

1.1 Copy prevention

There are several methods to achieve copy prevention. Identifying all of them would be very difficult. In the following subsection, we will thus present only the basic rules on which most of the existing technologies are based.

Copy prevention of the actual software used:

File copying is an easy task; everyone can just copy files and put them on different computers. In DOS operating system, most software's can be easily copied to another machine and will work fine. Nowadays, with Windows operating system, most software will require installation. The installation by itself does not prevent copying; however there are some system files and/or registry modification performed during the installation process. Thus, it is not as easy for the software to just being copied over to a different machine. The basic assumption here is the malicious user does not have the original software to perform installation. If user tries to just copy the whole files of the software, it just simply cannot work. The drawback of this method is the availability of other software (for example Norton Utilities and uninstall program) that could track what file is added or modified during the installation process. Further, it can also keep a backup of registry prior to installation and then compare the registry after registration to track what additional line is added to the registry. Thus, the user will know what files and registry addition are required to run the software on different machines. Another method being implemented is to add an encrypted file according to the machine where the software is installed. During installation, some information about the local machine is taken (example: machine name, user name used, or such that would very likely unique to each machine). The information is then encrypted and placed into a file or as part of registry. Each time the software is to be run, the encrypted information is decrypted and compared. If it matches, then the software will run. Applying this method disable the previous copying techniques to function properly. The only method to break this technique is by hacking the software and bypasses the checking function.

Copy prevention prior to installation of software's:

While the above methods try to prevent copying of the actual software, another method is to copy the installation software. In this way, if malicious user knows the serial number, he can install it to any machines he desires. Depending upon the media to which the installation software is at, there are several ways to prevent copying. On CD-ROM/DVD media, encryption mechanism can be implemented. There is a portion of the CD-ROM/DVD, which is used solely for key location. This location of the media cannot be copied or written by common CD/DVD writers, but only by commercial writers. An example of this is Sony PlayStation. Every Sony CD has a key location, which is accessed for originality prior to game loading. If the key is not there, the game just simply will not run. The way around this is by using a commercial quality copier that copies the whole CD as is. However, this is an expensive method and typically used by large (illegal) copier firm such as those in Taiwan or China. Another way around this, which is commonly used, is by putting in an original PlayStation CD during initial startup during the time when the key portion is accessed. After the key read, the machine will need some time to verify the key before reading the data portion. At that time, the original CD is taken out and the copied CD is inserted. Since the key is all the same for current PlayStation CD, with one original PlayStation CD, all copied CDs of any games can be played. On disk media (diskette or hard disk), copy prevention is almost impossible. One method that used to be implemented is to allow for only one time installation. After the first installation, the file is just then being corrupted. However, this present more hassle since commonly software needs to be installed again in the future (broken hard drive, operating system crash, etc). Further, it is very inconvenience for the valid user to contact the software vendor each time he/she needs or wants to reinstall the software. The common method of prevention is by installation prevention as will be discussed in the next section.

1.2 Installation prevention

Since copy prevention is difficult on certain media, most vendors add software installation prevention nowadays. The term “installation prevention” is referred to installation by users who don’t have the license to use it or should not have the software installed on more than one machine. In any cases, copy prevention method need to be applied on each of the installation prevention method. In the past, many programs could be installed without any authorization method, thus allowing anyone to install or reinstall any software as soon as they acquire its installation package. Hence, anyone could just install a program to multiple machines or get a copy and install it right away. Software vendors recognize the chance of their copyrighted software’s being used without any control or checking. Thus several method of prevention is being implemented and will be discussed in this subsection. The main distinction is **software, hardware, and combination method.** The hardware methods actually perform some software process as well, which indicate that it is not purely hardware only.

Software method

Serial number identification is the most common method used. Upon the purchase of the software, user will receive a serial number, which the installation package will ask upon setup/installation. The serial number is a required preliminary step before installation can be performed. In the beginning, the serial number is just being checked against a serial number list inside the installation package. However, user can still install it to multiple machines or makes copies of the installation package and gives it to others. Thus the media used to contain the installation package should have some copy prevention mechanism. For this, the serial list file is encrypted in some method with the key hidden in the installation software as well. On such cases, hackers have to find the key to decrypt the file, which make it more difficult to modify the serial list file. If the hacker can decrypt the file, he/she will still get a hold of the serial number. Thus, the serial list being used is not the actual number but hashed or encrypted numbers. Upon installation, the serial number entered is hashed or encrypted according to the installation package implementation and being compared to that in the serial list file. In such case, the hacker will not get the actual serial number list. Nevertheless, they can still bypass the checking process.

The next generation of this method is then being introduced which was to apply the **hashed/encrypted serial number (hSN)** into the registry or a file after the installation process is finished. Each time the actual software is

run; it takes the hSN from the registry/file and compares it again with its list. Thus, there have to be an hSN inputted during installation process or entered manually by the installer to the registry/file. In the latter case, a hacker needs to hack both the installation package and the actual software to bypass both checking processes. This makes it more difficult to hack the installation package and distribute it further. More advanced hackers are capable to find the portion of the checking program inside the installation package that will actually form the checking process after installation and modify or bypass it. However, the difficulty depends on what encryption applied on the actual software in the installation package. Typically installation package also have a checking program that will recheck whether the decryption performed correctly by checking each actual software file's size. Thus, it is much more difficult to hack although not impossible. Certainly the latter serial number method provides better copyright protection. Nevertheless, if the hacker/user knows the serial number, this method is almost useless. The basic assumption here is authorized user is not going to install it to multiple machines or provides the installation package and serial number to anyone else. A newer method that is being introduced is that upon installation, the installer will grab some information about the computer (similar to copy protection). User will still need to provide a serial number. During installation, some information about the local machine is taken (example: machine name, user name used, serial number of the software, or such that would very likely unique to each machine). The information is then encrypted and being showed to the user. Depending on the implementation, the installation package or installed software will then asks for a password. In some cases, the software is not being installed if the password is not supplied. In others, the software is being installed but will not work without the correct password. The user needs to contact the software vendor with the encrypted information. The vendor can be contacted by email (the nowadays still commonly used method), phone, or web-based script. The process can be automated or handled by vendor support people. In any case, the vendor will provide the password back to the user. Basically the encrypted information is used for a key for certain algorithm. The password is the result of the algorithm. The installed software or the installation package also has this algorithm and has the password information. If the password supplied by the user from the vendor is correct, then the installation will proceed or the software will run, depending on the implementation. Further, by having the user contact the vendor, the vendor can register the software used. A hacker can try to discover the required password by identifying the software portion that perform the algorithm and run a separate process of that algorithm. The hacker can then get the password and feed it into the password file. Due to this problem, a modified method is used where it is based on zero knowledge techniques. The software/installer doesn't know what the password is supposed to be, but it sends a question as part of the encrypted information. The vendor needs to solve the question that will be the password.

Hardware method

The serial number method is software only implementation. This method uses hardware implementation in addition to software implementation. The hardware is designed to perform certain function and produce result to the calling software within specified period of time.

Hardware key (dongle)

A long known method is to use hardware key (dongle). A dongle is a hardware based security device that attaches to the serial or parallel printer port of a desktop computer or into the PC-Card host of a laptop. It uses codes and passwords embedded inside the key to control access to software applications. Software integrated with a dongle will only run when that dongle is attached to the computer. Most dongles are fully transparent, which means a printer cable or other dongles can be attached together in a "daisy-chain", without interfering with the dongles functionality or information exchange. Dongle can be attached to any computer port: parallel port, serial port, ADB port, USB port, and Mac ISA Extension card. Dongles can be sophisticated algorithmic devices. Basically what happened is an implementation of serial number method above, but instead of giving the serial number information to the user to enter, it is embedded in the dongles. Typically, serial number method is also applied as well as dongle method as extra precaution. Further, this method is mostly used on the more expensive software/network based software. Single user software is unlikely to apply this method, unless the expensive one because it is quite expensive. A way to attack this method can be by making a dongle copy or stealing the dongle, which is very difficult to do. Another attack is by disabling the checking process as that of serial number check implementation. If

the process that calls towards the dongle can be disabled then the installed software can run fine. However, this method usually has multiple calls toward the dongle which makes it significantly more difficult to hack. Nevertheless, if the result of the dongle is observed and the result produced tends to be similar, the information required can be used to perform duplication attack. The result is saved into a file where the same information is used again. Each time the software asks for verification, small software redirects the request from asking the dongle to getting the information from the file. On some occasion, the software needs to be hacked to allow information gathering from file instead of dongle. Another attack is to replicate the process performed by the dongle in software. However, typically software performs much slower than hardware depending of the encrypt/decrypt algorithm, thus it might not yield the response in the required time. The software still needs to be hacked or redirected in similar manner as before. Hacker can also perform the similar attack as serial number checking bypass as discussed above. As the dongle method, it usually is performed multiple times in the software thus making it more difficult to break. Further each check will require fast feedback from the dongle, making software process attack almost impossible. The latter implementation is to include timestamp info added, which makes the dongle process each authentication request, making it almost impossible for the duplicate attack and software attack implementation.

1.3 Combination method

This research work is using combination of both software and hardware implementation. Combining the two methods makes it more difficult for hacker to stripe the authentication process from the software. In this approach encryption and decryption algorithms are used in such a way that encryption is carried out at software end and decryption is carried out at hardware end and finally the software execution / installation will take place after the verification of result.[2]

2. DESIGN METHODOLOGY

The general design cycle for this work consisted of the following steps:

- Implementation of RSA encryption algorithm on Java and VB.NET Platform.
- Implementation of RSA Decryption algorithm in VHDL for FPGA (dongle) programming.
- Serial and USB interface between computer and FPGA (dongle) using UART and USB Protocol in VHDL, VB.NET and Java.
- Software testing using GUI interface in Java and VB.NET.
- On line (WEB based) software testing in VB.NET.

3. HARDWARE AND SOFTWARE TOOLS

3.1 Software Tools :

- Xilinx ISE and MODELSIM software for FPGA Programming
- JCreator and jdk for java platform
- VB.NET and SQL server
- MOXA USB cable drivers
-

3.2 Hardware Tools:

- VIRTEX 4 (XC4VFX12) FPGA kit
- SPARTAN 3 (XC3S200) FPGA kit
- MOXA USB cable for USB interface
- RS232 serial cable for serial interface
- JTAG downloading cable

4. THE RSA ALGORITHM IMPLEMENTATION

4.1 RSA Algorithm:

The RSA algorithm was invented by Rivest, Shamir, and Adleman. Let p and q be two distinct large random primes. The modulus n is the product of these two primes: n = p.q. Euler's totient function of n is given by

$$\Phi(n) = (p - 1)(q - 1)$$

Now, select a number $1 < e < \Phi(n)$ such that
$\gcd(e, \Phi(n)) = 1$ and compute d with

$$d = e\text{-}1 \bmod \Phi(n)$$

Using the extended Euclidean algorithm here, e is the public exponent and d is the private exponent. Usually one selects a small public exponent, e.g., $e = 2^{16} + 1$. The modulus n and the public exponent e are published. The value of d and the prime numbers p and q are kept secret. Encryption is performed by computing

$$C = M^{e} \pmod n$$

M is the plaintext such that $0 \leq M < n$. The number C is the cipher text from which the plaintext M can be computed using

$$M = C^{d} \pmod n$$

4.2 Technique Used For RSA Implementation on FPGA

Five-To-Two CSA Architecture

The first of these variants is based on a five-to-two CSA. The inputs to the algorithms are A1, A2, B1, B2, n where (A1, A2) and (B1, B2) are the Carry save representations of A, B respectively and 'n' is the modulus. This type of adder comprises three levels of carry save logic.

$$SUM = X1 \text{ xor } X2 \text{ xor } X3$$
$$CARRY = (X1 \text{ and } X2) \text{ or } (X1 \text{ and } X3) \text{ or } (X2 \text{ and } X3)$$

The sum of the bit vectors *SUM* and *CARRY* is equal to the sum of the three input bit vectors *X1*, *X2* and *X3*. An outline diagram of the five-to-two CSA operation is shown in Figure 1. The carry save representation of the five input operands is output from the register after only one clock cycle using this approach.

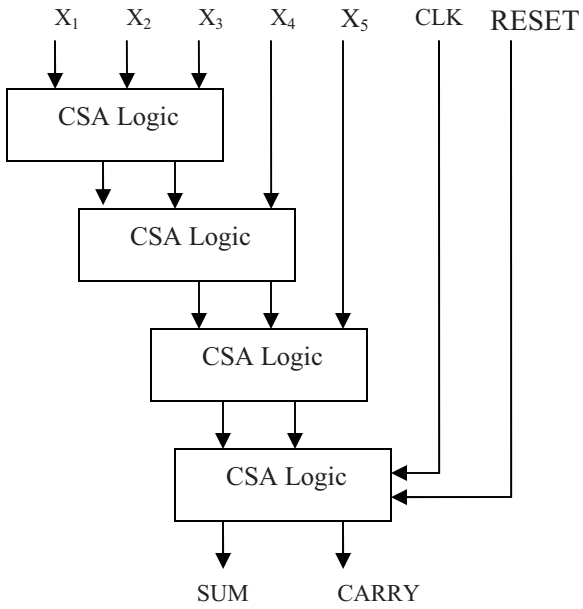

Figure 1 Block Diagram of five-to-two CSA

ALGORITHM I: *Five-to-two CSA Montgomery Multiplication[3]*
(A1, A2, B1, B2, n)
S1 [0] = 0;
S2 [0] = 0;
For i in 0 to k-1 loop
$qi = (S1[i]_0 + S2[i]_0) + (A_i * (B1_0 + B2_0)) \bmod 2$;
S1[i+1], S2[i+1] =
CSR $(S1[i] + S2[i] + A_i * (B1 + B2) + q_i * n) \text{ div } 2$;
End loop;

The author has also demonstrated the four-to-two CSA approach. It has been shown that an RSA processor using the five-to-two multiplier can carry out encryption in only $(k+2)\ (e_k+3)$ clock cycles, where k is the modulus bit length and e_k is the public exponent bit length. The Fermat prime, $F4=2^{16}+1$, is used as the public exponent. RSA decryption can be performed in $k/2+2)\ (d_k/2+3)$ clock cycles, where d_k is the private exponent bit length, also using the five-to-two multiplier. The Chinese Remainder Theorem (CRT) technique is used in this case. It has also been shown that using a four-to-two CSA with two extra registers rather than a five-to-two CSA leads to a useful reduction in the critical path of the multiplier, albeit at the expense of a small increase in circuitry. For operand lengths of 1536-bits and greater, the percentage gain in data throughput rate outweighs the percentage increase in silicon area. Moreover, for a 2048-bit operand length, typical of what is required in many future generation applications, the gain in data throughput is 27.9% compared with a 9.9% increase in area.

4.3 RSA FPGA Design Verification

The functionality of RSA design is captured generically in VHDL and implemented into the Xilinx Spartan-3 and Virtex-4 series of FPGAs for varying operand lengths of 512, 1024 and 2048. Synthesis and simulation has been done using ISE 8.2i. The simulation waveform has been shown in Figure2 and figure3 clearly showing the

encryption and decryption of message using RSA algorithm. Here the public and private key pair is {7, 23}. This can be calculated manually.[4]

$$\text{Cipher} = 11^{23} \text{ mod1 } 187 = 88.$$
$$\&$$
$$\text{Message} = 88^{7} \text{ mod } 187 = 11$$

Hence the cipher-text once generated is clearly converted back to plaintext without any loss of precision. This can be clearly verified from the simulation waveforms.

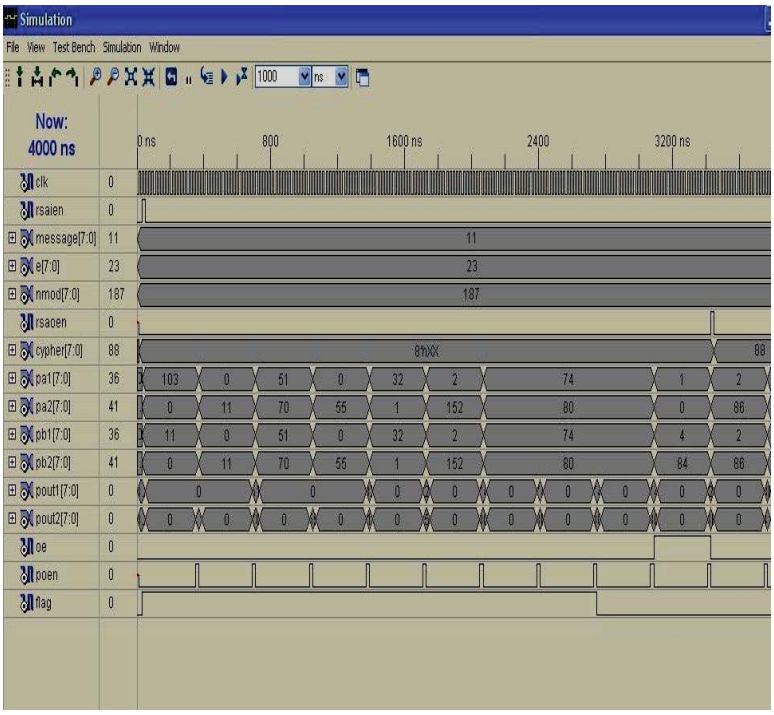

Figure 2 RSA Encryption

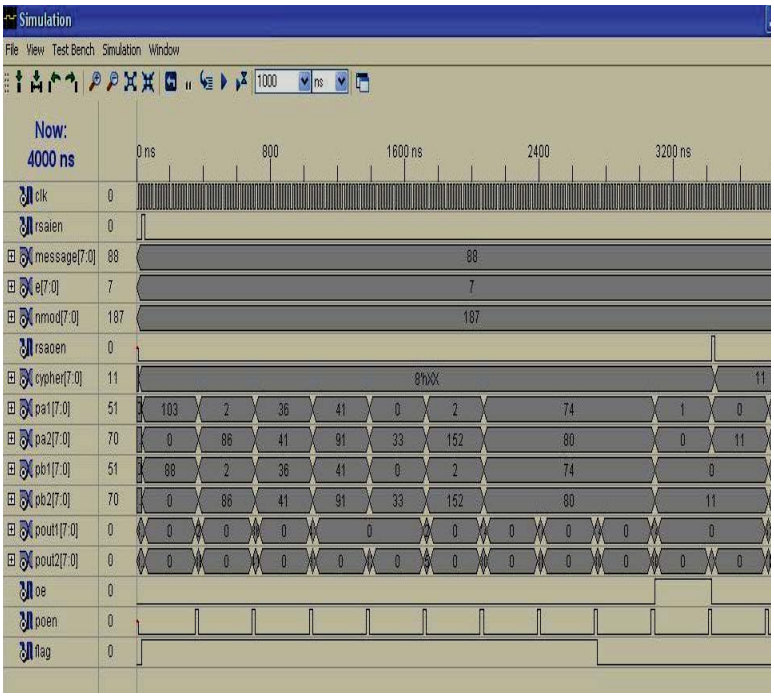

Figure 3 RSA Decryption

4.4 DESIGN FLOW of RSA implementation

Following flow chart will explain the design flow of RSA Processor. The flow chart is based on five-to two CSA approach. [5][6]

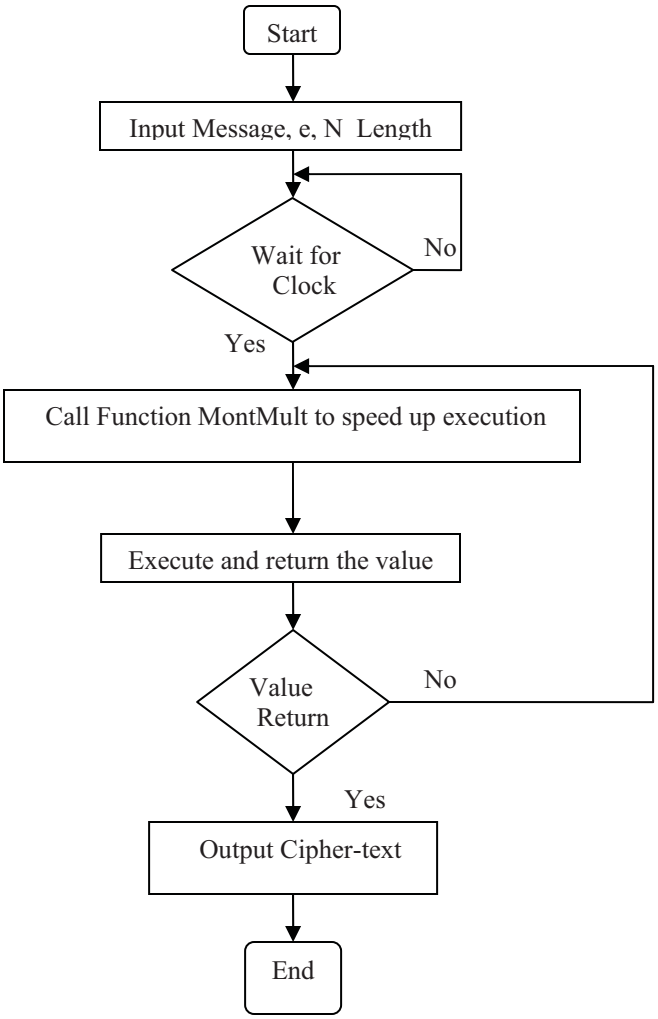

Figure 4 Flowchart of the Previous Design

5. SERIAL AND USB INTERFACE

5.1 Receiver

This module receives data from Computer and using a state machine it stores it in a serial in parallel out shift register. It has to take in care the start and stop bits that accompany the data bit and remove them and store the data. These start and stop bits are used to control our state machine. The control part runs at 16 times the baud rate to minimize the errors due to clock of FPGA and baud rate. The control part samples the data received in the middle of the baud cycle and pushed the data in the shift register. Idle data line is represented by logic 1 and state machine starts sampling when data line becomes logic 0. When start bit is detected, Q0 goes to Q1 and Q1 waits for 6 clock cycles so that data is sampled in middle of baud cycle (Clock is 16 times baud rate). State which makes us wait for 14 clock cycles just brings us to next baud cycle. Q8 state represents that data which was received was not correct as data line was not idle but Q9 state means that data has been received correctly.[7][8]

5.2 Transmitter

The state machine of transmitter appends the data bit with a start bit (a transition from 1 to 0) and for the next 8 clock cycles it serially streams out the 8 data bits at the baud rate mutually agreed upon by the transmitter of the FPGA and by the receiving end (PC). The baud rate is set initially so that clock of FPGA and the baud rate are equal within 1 percent. In the state machine after appending the start bit it sends 8 data bits and then appends the end bit. After that, it makes the line idle. Thus, this state machine follows the protocol for serial data transfer [9].

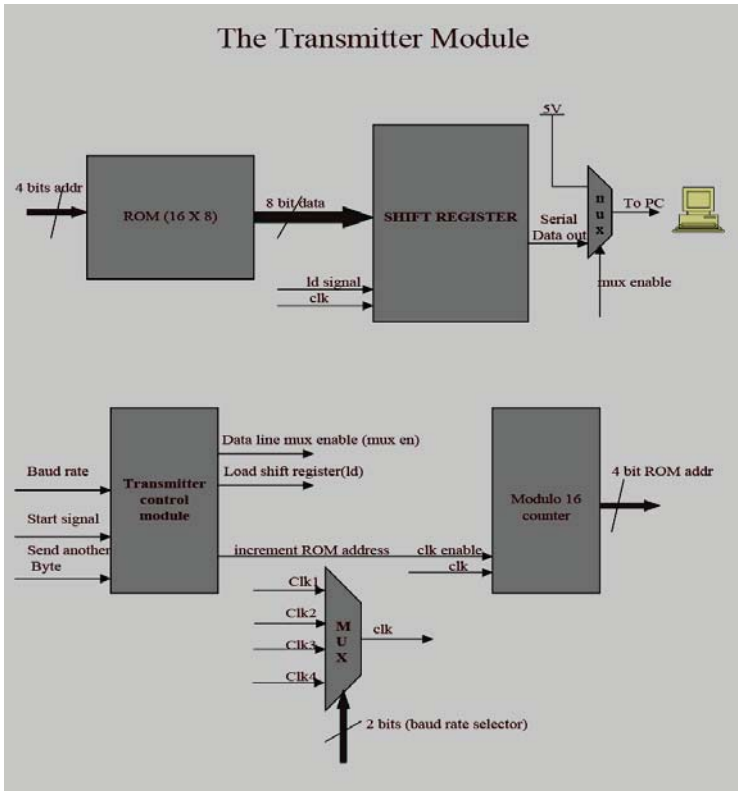

Figure 5: Transmission Protocol

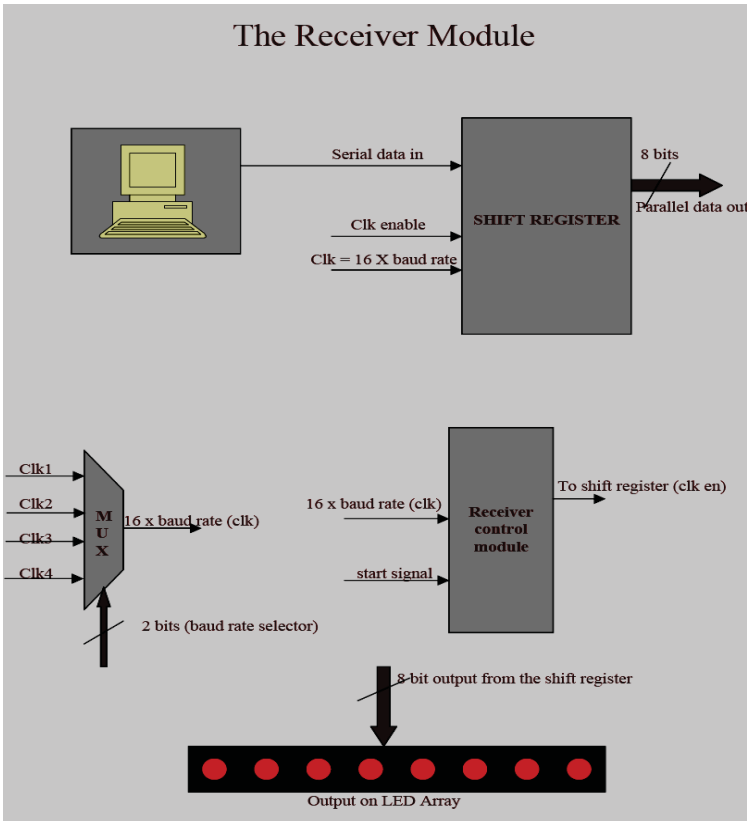

Figure 6: Reception Protocol

6 FINAL SOFTWARE TESTING

The window based application is designed in such a way that Software Piracy can be stopped .At the back end of the application a task manager is tracing out all the processes whose status is set to “Running” and re- filter out all that processes saved by User within database during previous transaction and the one that he have just provided to the application, these processes is further delivered to routines for making their status Closed. The database is maintaining a log carrying all the processes provided by User, and during startup of the application it blocks them all if required hardware is missing. If the application finds Dongle presence it stops all its back end routines and set the status of the processes to “Running”. Besides this, the application is communicating with virtual COM Ports created through Mosa Utility Driver provided with this application; in each installation of this driver the system get a new virtual port. The application needs user side feedback for determining the COM Port and one initiative to run the hardware simulation and rest of the task is done through .NET. [10] - [14]

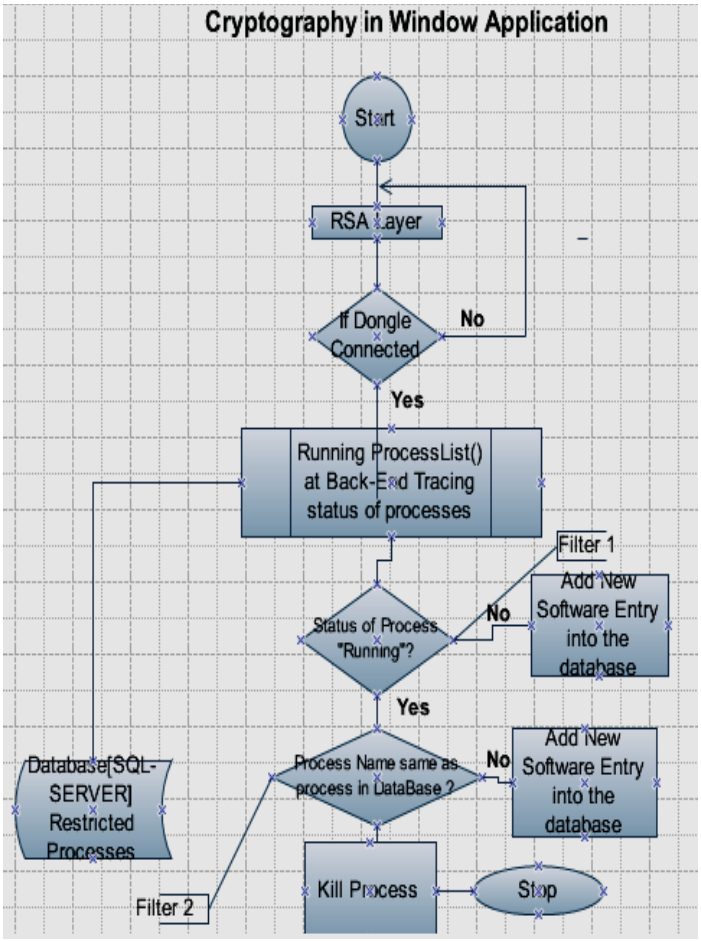

Figure 7: Flow sequence of software testing

7 RESULTS AND CONCLUSION

The entire design on FPGA kit have different throughputs .It gives an through put of 1.06Gb/sec in Virtex 4 kit and 0.544Gb/sec in Spatan 3 kit as shown graphically in figure 8 . Thus this design can be used in preventing software piracy in this way that software execution can only be possible if the FPGA device is connected.

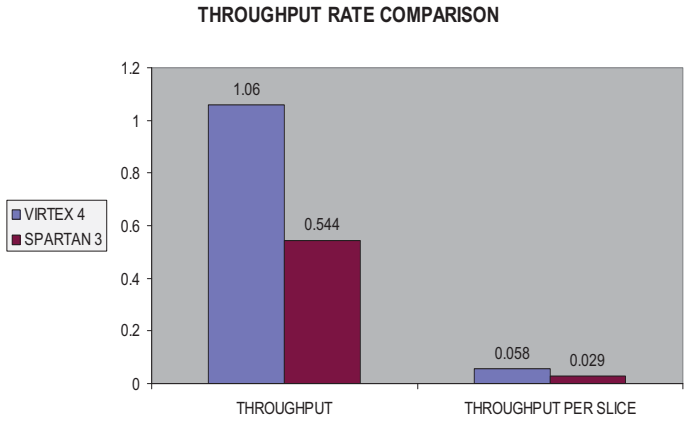

Figure 8 Through put rate comparison of Algorithm on Virtex 4 and Spartan 3 kit

REFERENCES

[1] William Stallings ,"Cryptography and Network Security, Principles and practices" Edition 3d, Pearson Education

[2] M.K. Hani, H.Y.Wen, A.Paniandi, "Design and implementation of Private and Public key Crypto processor for next generation IT Security applications", Malaysian journal of computer Science, vol.19 (1), 2006

[3] P. Montgomery, "Modular multiplication without trial division, "Mathematics of Computation, No.44, pp.519-521,1985.

[4] A. Menezes, P.Van Oorschot, and S.Vanstone, "Handbook of applied Cryptography", CRC Press, 1996

[5] R. J. Hwang, F.F.Su, Y.S.Yeh, C.Y. Chen," An Efficient Decryption Method for RSA Cryptosystem", Proceedings of 19th international conference on Advanced Information networking and Applications, 2005

[6] Xinmiao Zhang, Student Member, IEEE, and Keshab K. Perhi, Fellow, IEEE, "High-Speed VLSI Architectures for the AES Algorithm," *Proc. IEEE, 2004*.

[7] Nele Mentens, Lejla Batina, Bart Preneel and Ingrid Verbauwhede, "A Systematic Evaluation of Compact hardware Implementations for the Rijndael S-Box," 2005.

[8] Alireza Hobjat, Student Member, IEEE and Ingrid Verbauwhede, Senior Member, IEEE "Area-Throughput Trade-Offs for fully Pipelined 30 to 70 Gbits/s AES Processors", *IEEE 2006*.

[9] National Institute of Standards and Technology. Specification for the Advanced Encryption Standard (AES). *FIPS PUB 197*, available at http://csrc.nist.gov, 2001.

[10] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalist. Presented at *Proc. 3rd AES Conf. (AES3)*. [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html

[11] M. McLoone and J. V. McCanny, "Rijndael FPGA implementation utilizing look-up tables," in *IEEEWorkshop on Signal Processing Systems*, Sept. 2001, pp. 349–360.

[12] Daniel Cazzulino, "Web Programming using VB.NET"

[13] Ganelon, "Visual Basic . Net Black Book"

[14] www.codeworks.it/net/VBNetRs232.htm.