FPGA BASED-QUBIT PROCESSOR IMPLEMENTATION USING RSA ALGORITHM

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

In

Electronics and Communication Engineering

by

Devakinandan Palla	Sourodipto Bhowmick	Abhishek Kumar
19BEC0812	19BEC0803	19BEC0752

Under the guidance of

Dr. Sakthivel Ramachandran

SENSE

VIT, Vellore.

April, 2023

DECLARATION

I hereby declare that the thesis entitled **"FPGA BASED-QUBIT PROCESSOR IMPLEMENTATION USING RSA ALGORITHM"** submitted by us, for the award of the degree *of Bachelor of Technology in Electronics and Communication* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Sakthivel Ramachandran.

We further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date: 12/04/23

Sourodipto Bhowmick

Signature of the Candidate

Devakinandan Palla

Signature of the Candidate

Abhishek Kumar

Signature of the Candidate

Dr. Sakthivel Ramachandran
Signature of the Guide

CERTIFICATE

This is to certify that the thesis entitled "**FPGA BASED-QUBIT PROCESSOR IMPLEMENTATION USING RSA ALGORITHM**" submitted by **Devakinandan Palla(19BEC0812), Sourodipto Bhowmick(19BEC0803) and Abhishek Kumar(19BEC0752),**SENSE,VIT, for the award of the degree *of Bachelor of Technology in Electronics and Communication,* is a record of bonafide work carried outby him / her under my supervision during the period, 01. 12. 2022 to 19.04.2023, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 14/04/2023

Dr. Sakthivel Ramachandran

Signature of the Guide

Internal Examiner

External Examiner

Dr. Noor Mohammed V

Head of Department

Electronics and Communication Engineering

ACKNOWLEDGEMENTS

We would like to thank all the people who helped and supported us with writing this bachelor's research project.

We wish to express our sincere thanks to Dr. G. Viswanathan, founder, and chancellor of the college, for furnishing us with all the necessary facilities.

We take this opportunity to thank our Head of Department, Department of SENSE for consistent encouragement.

We would like to express our sincere gratitude to our guide Dr. Sakthivel Ramachandran- School of Electronics and Communication Engineering for his kind supervision. His guidance helped us in executing our research work. Without his help, it wouldn't have been possible to carry on this project.

Close to our guide, we would like to thank the rest of our professors for their comprehensive advice and insightful comments which helped us widen our research from a different point of view. We are extremely grateful to our parents and siblings and the love and support they have given us throughout the duration of this project.

Finally, we have a special mention to all our friends in VIT for their constant encouragement.

Devakinandan Palla

Sourodipto Bhowmick

Abhishek Kumar

Executive Summary

The RSA algorithm is a widely used public-key encryption algorithm that is used to secure data transmitted over networks. In recent years, there has been a growing interest in implementing RSA encryption on FPGAs (Field Programmable Gate Arrays) due to their ability to provide high-speed computation and low power consumption.

In this project, we propose the FPGA implementation of the RSA algorithm using Verilog and Qiskit for 64-bit encryption. The project involves the development of a Verilog-based hardware implementation of the RSA algorithm, along with a software implementation using Qiskit, a quantum computing SDK developed by IBM.

The hardware implementation of the RSA algorithm is achieved using a modular arithmetic approach, which allows for efficient computation of large integer numbers. The Verilog code is optimized for FPGA implementation, with a focus on minimizing the power consumption and maximizing the performance of the design.

The software implementation of the RSA algorithm using Qiskit involves the usage of a quantum simulator to simulate the quantum circuits required for the algorithm. The Qiskit code is used to generate the quantum circuits and simulate their behavior. The results of the project show that the FPGA implementation of the RSA algorithm using Verilog and Qiskit is capable of achieving high-speed computation and low power consumption.

The 64-bit encryption is achieved in a time-efficient manner with negligible errors. Overall, this project demonstrates the potential of using FPGAs and quantum computing in combination to implement encryption algorithms such as RSA. The project has practical applications in securing sensitive data transmitted over networks and can pave the way for further research in the field of cryptography and quantum computing.

CONTENTS

List of Figures

List of Tables

List of Abbreviations

RSA	Rivest, Shamir, Adleman
HFT	High Frequency Trading
MAC	Message Authentication Module
IBM	Internal Business Machines
SDK	Software Development Kit
FPGA	Field Programmable Gate Arrays
HDL	Hardware Descriptive Language
ASIC	Application Specific Integrated Circuits
QFT	Quantum Fourier Transform
QPE	Quantum Phase Estimation
RTL	Resistor Transistor Level
ALM	Adaptive Logic Module
LUT	Look Up Table

Symbols and Notations

Φ	Phi
$\equiv$	Identical
%	Modulus
π	pi

1. INTRODUCTION

1.1. OBJECTIVE

The project aims to have an outlook on compromising security for the sake of high performance whilst using RSA algorithm. This can be used in applications which do not really have much securityconcerns or the risks of less performance outweigh that of security issues.

1.2. MOTIVATION

The reason we choose RSA is its tougher ability to crack, which possesses a challenge to break it down. For example, take the classic example of Alice and Bob. Alice wants to send Bob a message, and thus she encrypts the message with a security key. But unfortunately, Bob does not have access to the secret key to unlock it. Alice now is unable to send Bob the secret key securely, since it gets hackers an easy job to decrypt or corrupt the message. Thus, now what she does is, by using RSA algorithm, she makes both of them generate a pair of public key and a private key, that are mathematically linked to each other on their computers. The public key is used to encrypt the data, whereas only the private key is used to decrypt the data.

Even though the links are related to each other, one cannot be derived from each other. Therefore, what Alice and Bob now do is, exchange each other's public key, and then send the message after encryption. Thus, Bob is now able to open his message through his unique private key, and not even Alice can decrypt it. We thus used this above technique because despite our potential threats of using a small bit key size, the kind of algorithm which we used is branded and reputed enough and is still pretty strong enough.

1.3 BACKGROUND

High performance precedence over security has many applications and fields demanding. To list a few, take High Frequency Trading sector where high-frequency trading (HFT), financial institutions use powerful algorithms to make trades in fractions of a second. In order to achieve the necessary speed, security protocols may be relaxed, such as reducing encryption strength or using weaker authentication mechanisms. This can make HFT vulnerable to cyber-attacks, but the potential profits outweigh the risks for some traders. Another such application would be Video Streaming which require a lot of bandwidth to deliver high quality video to users.

To ensure smooth playback, video streaming services may sacrifice some security measures such as using weaker encryption algorithms or not using encryption at all. This can make the video stream vulnerable to eaves dropping, but it may be deemed an acceptable trade-off for delivering a seamless user experience.

It is important to note that in all of these cases, the potential risks should be carefully considered before compromising security measures for high performance. Usage of secure key exchange protocols: One way to mitigate the risks of a smaller RSA key size is to use a secure key exchange protocol, such as Diffie-Hellman key exchange or Elliptic Curve Diffie-Hellman key exchange, to establish a shared secret key between the communicating parties. This shared secret key can then be used to encrypt and decrypt messages, providing an additional layer of security.

Implementation of multi-factor authentication: multi-factor authentication can be used to ensure that only authorized users are able to access the system or data being encrypted. This can include using passwords, biometrics, or other authentication factors in addition to the RSA key.

Usage of message authentication codes (MACs): A message authentication code (MAC) can be used to verify the authenticity of the message and ensure that it has not been tampered with. This can be particularly useful in situations where the key size is too small to provide strong encryption, as it provides an additional layer of security.

Usage of a trusted hardware module: A trusted hardware module, such as a hardware security module (HSM), can be used to store and manage RSA keys securely. This can help to protect the key from unauthorized access or tampering, even if the key size is small.

Implementation of access controls and monitoring: Access controls and monitoring can be used to ensure that only authorized users have access to the system or data being encrypted, and to detect and respond to any unauthorized access attempts or security breaches.

It’s important to note that while these techniques can help to mitigate the risks associated with a smaller RSA key size, they are not a replacement for using a larger key size when strong security is required.

2. PROJECT DESCRIPTION AND GOALS

The project aims to have an outlook on compromising security for the sake of high performance. Thus, this can be used in applications which do not really have much security concerns or the risks of less performance, outweigh that of security issues. High frequency trading is a major area where the above requirements can be fulfilled. In high-frequency trading (HFT), financial institutions use powerful algorithms to make trades in fractions of a second. In order to achieve the necessary speed, security protocols may be relaxed, such as reducing encryption strength or using weaker authentication mechanisms. This can make HFT vulnerable to cyber-attacks, but the potential profits outweigh the risks for some traders, where this reduced key size can come into play. In order to achieve this goal, we have used a 64-bit RSA algorithm using the best suited multiplier algorithm, the Booth's multiplier and simple Square and Multiply algorithm.

We have obtained an excellent throughput with a very good frequency along with a low-cost implementation due to its reduced complexity. We have also provided adequate methods to counter security concerns due to its extremely small key size, and therefore wish that our proposals shall be useful. The tools that we have used for the implementation are ModelSim, Intel Quartus II Prime(FPGA) using Verilog HDL code.

We have also done a supplementary study on how Quantum computing has been successful in breaking the most complex of algorithms up to 1024 bits in hours, which has indirectly pushed us to stick to a lower 64- bit encryption size with alternate proposals to enhance security while keeping the speed and performance intact. For this, we have used Qiskit software to study and observe.

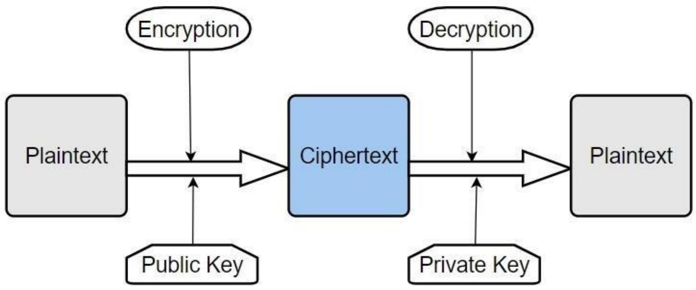

Figure 1: Block Diagram of RSA

3. TECHNICAL SPECIFICATIONS

3.1 SOFTWARES USED

MODELSIM- ModelSim is a hardware simulation and verification tool used in the design and testing of digital logic circuits. It is a software program that provides a comprehensive environment for simulating and debugging digital designs.

ModelSim provides a range of features, including waveform viewers, code coverage analysis, and automated testbench generation, to aid in the debugging and testing of digital designs. It also supports a wide range of simulation models, including interactive simulation,batch simulation, and post-synthesis simulation.

QUARTUS PRIME- Quartus Prime is an integrated development environment (IDE) for the design and verification of digital circuits. It is a software tool developed by Intel (formerly Altera) and is used for designing, simulating, synthesizing, and verifying Field-

Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). Quartus Prime supports the hardware description languages (HDLs) VHDL, Verilog, and System Verilog, and it provides a range of features to aid in the design and verification of digital circuits. These features include a graphical user interface (GUI) for designing and editing circuits, a compiler for synthesizing HDL code into digital circuits, a simulator for testing the circuits, and a place-and-route tool for mapping the circuits to FPGA or ASIC architectures.

QISKIT - Qiskit is an open-source software development kit (SDK) for building and running quantum programs. It is developed by IBM and is one of the most popular SDKs for programming quantum computers.

Qiskit provides a range of tools and resources for working with quantum computing, including a quantum circuit composer, a quantum simulator, and access to real quantum hardware through the IBM Quantum Experience. It also provides a range of tutorials, exercises, and learning resources to help users get started with quantum computing.

IBM JUPYTER NOTEBOOK- To use IBM Jupyter Notebook, the user can create an account on IBM Watson Studio and launch the environment. From there, one can create a new Jupyter Notebook and start writing code in our preferred programming language. IBM Jupyter Notebook allows users to import data, visualize it, perform statistical analysis, and build machine learning models all within the same environment. The user can also share your work with others by exporting your notebooks or publishing them on platforms such as GitHub orKaggle.

3.2 CODING LANGUAGES USED

VERILOG HDL- Verilog Hardware Description Language (HDL) is a hardware description language used in digital circuit design and verification. It is one of the most widely used hardware description languages and is used to design digital systems such as integrated circuits, field-programmable gate arrays (FPGAs), and other digital logic devices. Verilog HDL is a text-based language that allows designers to describe the behavior of digital circuits in a high-level manner. It is used to describe the structure and behavior of digital circuits, including modules, signals, and registers. Verilog HDL also supports timing and delay modeling, which is essential for designing digital circuits that operate at high speeds. To obtain FPGA design and waveforms, we have used this coding language.

PYTHON- Python is one of the most widely used programming languages in quantum computing. It is often used in conjunction with quantum computing frameworks and libraries, such as Qiskit, which provide an easy-to-use interface for working with quantum algorithms and quantum hardware.

Python is a popular choice for quantum computing because of its ease of use, readability, and large ecosystem of libraries and tools. It is also a high-level language, which makes it easy to express complex quantum algorithms in a simple and concise way. In quantum computing, Python is used for a wide range of tasks, including designing and simulating quantum circuits, developing and testing quantum algorithm. We have the used to same to obtain our RSA circuit on quantum level using qubits in IBM JUPYTER Notebook.

4. DESIGN APPROACH AND DETAILS

4.1 DESIGN APPROACH/ MATERIALS & METHODS

In the RSA algorithm, each user has a pair of keys: a public key and a private key. The public key is shared with everyone, while the private key is kept secret. The public key is used to encrypt messages, while the private key is used to decrypt messages. The algorithm works as follows:

Choose two large prime numbers p and q and a message signal m.
Calculate n = p*q.Calculate phi(n) = (p-1)*(q-1).
Choose a public exponent e such that 1 < e < phi(n) and e is coprime with phi(n).
Calculate a private exponent d such that d*e ≡ 1 (mod phi(n)).
The public key is (n,e). The private key is (n,d).
To encrypt a message m, compute c = m^e mod n.
To decrypt a ciphertext c, compute m = c^d mod n

RSA encryption and decryption are mutual inverses and commutative due to symmetry which is present in modular arithmetic. Hence the Encryption engine contains both the operation of Encryption and Decryption. Here is the mathematics involved in modular arithmetic: A and B are congruent modulo m if and only if A–B is divisible by m. It is written as: A = B mod m Where m is a positive integer known as modulus. The sign "≡" indicates congruence. Modular exponentiation operation has a series of modular multiplication and squaring operation.

This simplification is based on an algorithm called square and multiply algorithm. This algorithm is based on scanning the bit of the exponent from the left to the right. In every iteration, i.e., for every exponent bit, the current result is squared when the currently scanned exponent bit has the value 1, a multiplication of the current result by M is obtained following the squaring. This algorithm can be represented in pseudo code as shown below.

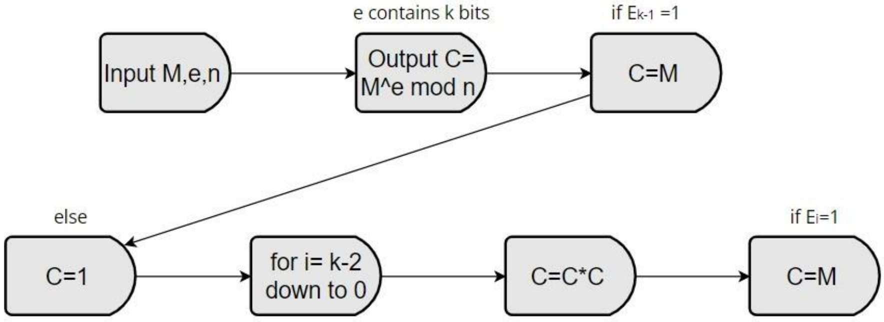

Figure 2: Square and Multiply Algorithm

The modular multiplication problem is the computation of P= A × B (mod n), with integers A, B, and n. It is usually considered that A and B are positive integers with 0 A, Bn. The square or multiplication operation is just a simple multiplication process. There are many good approaches to perform multiplication such as Multiply then divide, Interleaving multiplication and reduction, etc. But here Montgomery's algorithm is used. It avoids the "division" operation and uses" shift and addition" operations to do modular multiplication. Suppose A and B are two k-bit positive integers, respectively. Let Ai and Bi are the bit of A and B, respectively. The algorithm is as follows:

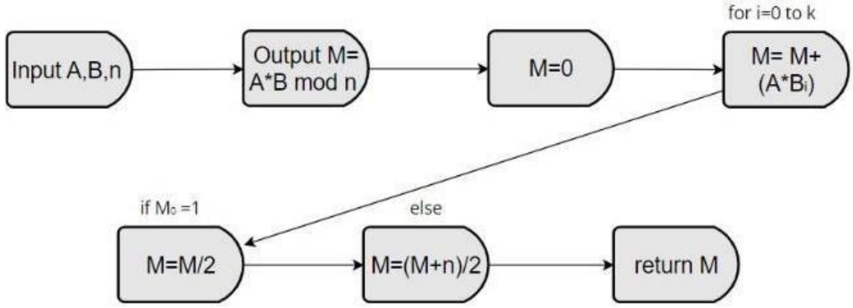

Figure 3: Shift and Addition Algorithm

In our project carry save adder is used for performance of both addition and subtraction operation. Subtraction means addition of a number with 2's complement of one more number. Here a select lines are used to perform the selection of adder or subtracter. If A, B are positive integers, and S is the result then S=A+B, when select line is 0 else S=A-B will be the output.

All the above theorems and algorithms play a vital role in RSA algorithm key generation and getting the encrypted and decrypted values.

4.2 CODES AND STANDARDS

The modular operations code defines a module named "mod_operationrsa" that implements a basic arithmetic operation using two 32-bit input values "in1" and "in2". The module has inputs for clock and reset signals, as well as other control signals such as "op_rdy" and "sign". The output of the module is a 64-bit value named "out" and an output signal named "out_rdy", which indicates when the output is ready. The module uses a set of registers and wires to implement the arithmetic operation, including are register for the output value, a register for a temporary output value, and registers for the input values. Within this block, the module uses a set of if-else statements to perform the arithmetic operation based on the input values and control signals. Overall, this Verilog HDL code implements a basic arithmetic operation that can be used in RSA encryption and decryption algorithms. It covers both encryption and decryption engines.

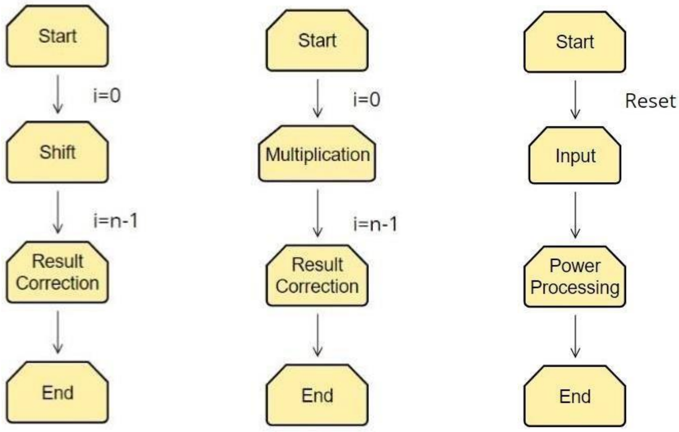

Figure 4: (from left to right) Booth Multiplier, Array Multiplier, Power Multiplier

The power module seems to implement a power function that computes data1 raised to the power of data2. The inputs to the module include a clock signal (clk), a reset signal (reset), two data inputs (data1 and data2), and an input ready signal (in_rdy). The outputs of the module include the output signal (out) and an output ready signal (out_rdy). The module uses a sequential always block triggered on the positive edge of the clock signal. The loop iterates data2 times, multiplying temp by data1 on each iteration. The count variable keeps track of the number of iterations performed so far. If count reaches data2, the loop terminates and out_rdy is set to true. Otherwise, out_rdy is kept false. Once the loop is complete, the result is stored in temp, which is then output on the out signal.

Coming to the quantum computing part, the code starts by defining some parameters required for RSA encryption such as the prime numbers p and q, the public exponent e, the modulus n, and the quototient of n (phi n). It then defines a message to be encrypted. The codethen creates a quantum circuit with a number of qubits equal to the bit length of n (num bits), and one less classical bit (num bits_1) to store the estimated phase. The message is encrypted by computing the ciphertext c using the RSA encryption formula (c = message^(e) mod n). The binary representation of the ciphertext is then obtained and the qubits corresponding to the binary digits of 1 are put into a superposition state using Hadamard gates. The next step is to perform the quantum phase estimation algorithm (QPE) to estimate the phase angle corresponding to the eigenvector of a unitary operator. In this case, the unitary operator is chosen to be a controlled rotation operator that rotates the phase of the state of the last qubit by an angle proportional to the binary representation of the ciphertext. The angle of rotation is chosen

to be theta = 2*pi / (2^num_bits).

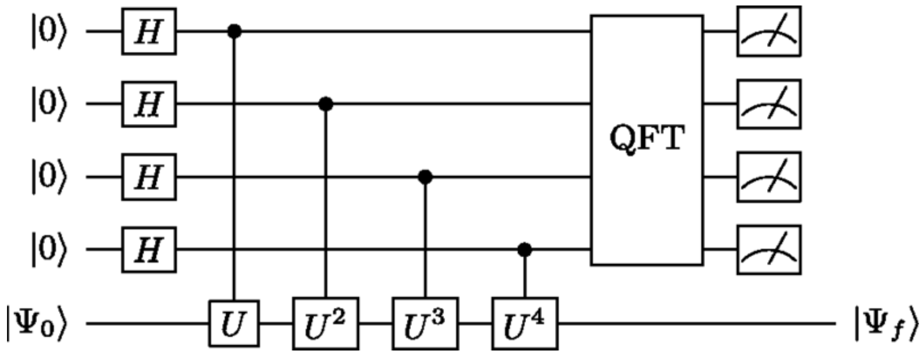

Figure 5: Quantum Phase Estimation Algorithm

4.3 CONSTRAINTS, ALTERNATIVES AND TRADEOFFS

Constraints:

1) Resource limitations of the FPGA chip, including available memory, number of logic elements, and power consumption.
2) Compatibility issues between the Verilog and Qiskit platforms.
3) Time constraints for completing the project within a given timeframe.

Alternatives:

1) Alternative FPGA chips with higher logic density and more memory could be used to enhance the performance of the design.
2) Alternative simulation tools could be used instead of Qiskit for the software implementation of the RSA algorithm.
3) Alternative encryption algorithms such as Elliptic Curve Cryptography (ECC) could be considered as they have been shown to provide similar security with smaller key sizes and lower computational overhead.

Tradeoffs:

1) Security vs performance and power consumption: The primary tradeoff committed to this project was prioritizing higher performance with lower power consumption at the cost of a standard secure device.
2) Hardware versus software implementation: The hardware implementation may provide higher performance, but the software implementation may be more flexible and easier to modify.
3) Design complexity versus performance :A more complex design may provide higher performance, but may also consume more resources and require longer development time.

5. SCHEDULE, TASKS AND MILESTONES

Table 1: Progress of Work

S. No.	Activity	Status	Date
1	LITERATURE SURVEY	COMPLETED	01/02/22- 20/12/22
2	PROBLEM IDENTIFICATION	COMPLETED	22/12/22- 10/01/23
3	IMPLEMENTING VERILOG CODE ON MODELSIM	COMPLETED	15/01/23- 18/02/23
4	OBTAINING RTL DESIGN ON QUARTUS PRIME	COMPLETED	19/02/23- 21/02/23
5	IMPLMENTING PYTHON CODE ON QISKIT	COMPLETED	28/02/23- 20/03/23
6	OBTAINING QUANTUM CIRCUIT	COMPLETED	23/03/23- 25/03/23
7	FINAL ANALYSIS OF PROJECT	COMPLETED	26/03/23- 27/03/23
8	RESEARCH PAPER IN IEEE FORMAT	COMPLETED	28/03/23- 01/04/23
9	POSTER	COMPLETED	07/04/23- 11/04/23
10	FINAL REPORT	COMPLETED	12/04/23- 14/04/23

6. PROJECT DEMONSTRATION

The entire RSA algorithm engine has been performed in 4 different modules in Verilog HDL. Below are screenshots of each module

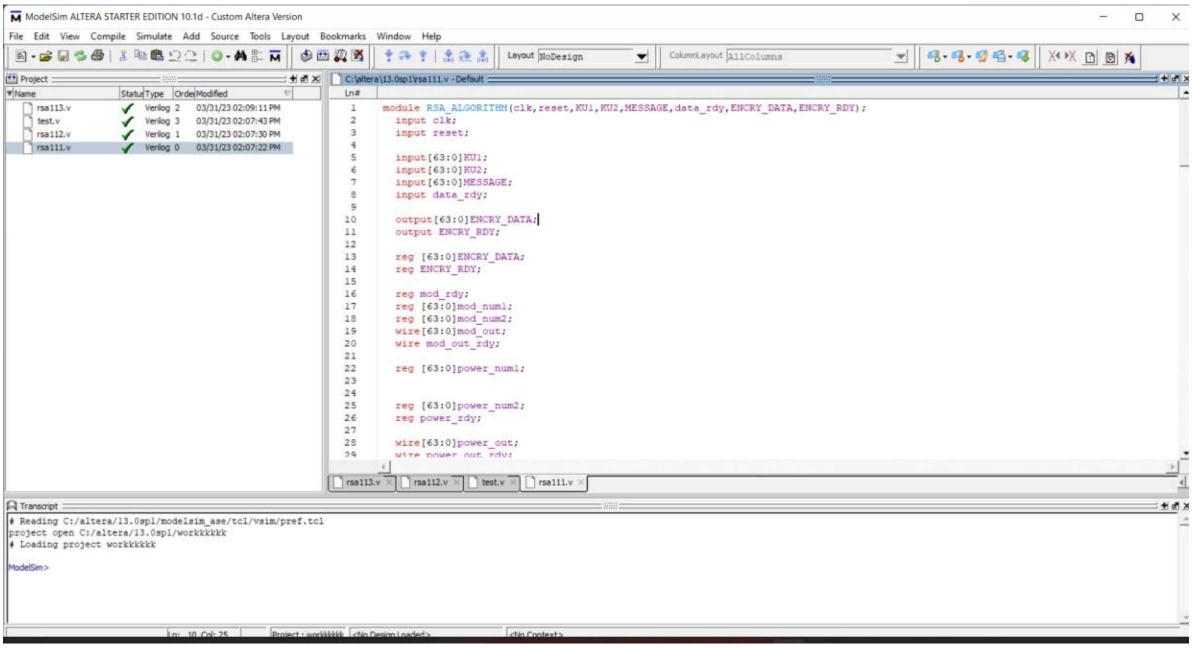

Figure 6: RSA Engine Module

The above module is implemented for performing the basic RSA Encryption and Decryption process.

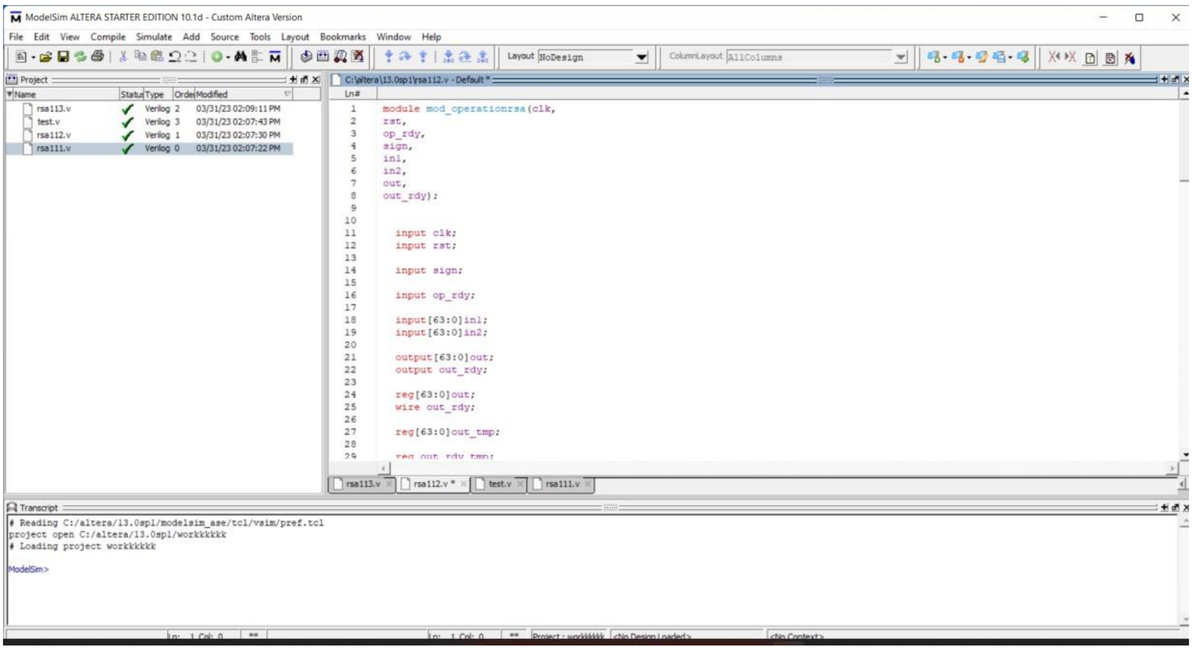

Figure 7: Modular Operations Module

The above module performs the basic modular operations and set as top module.

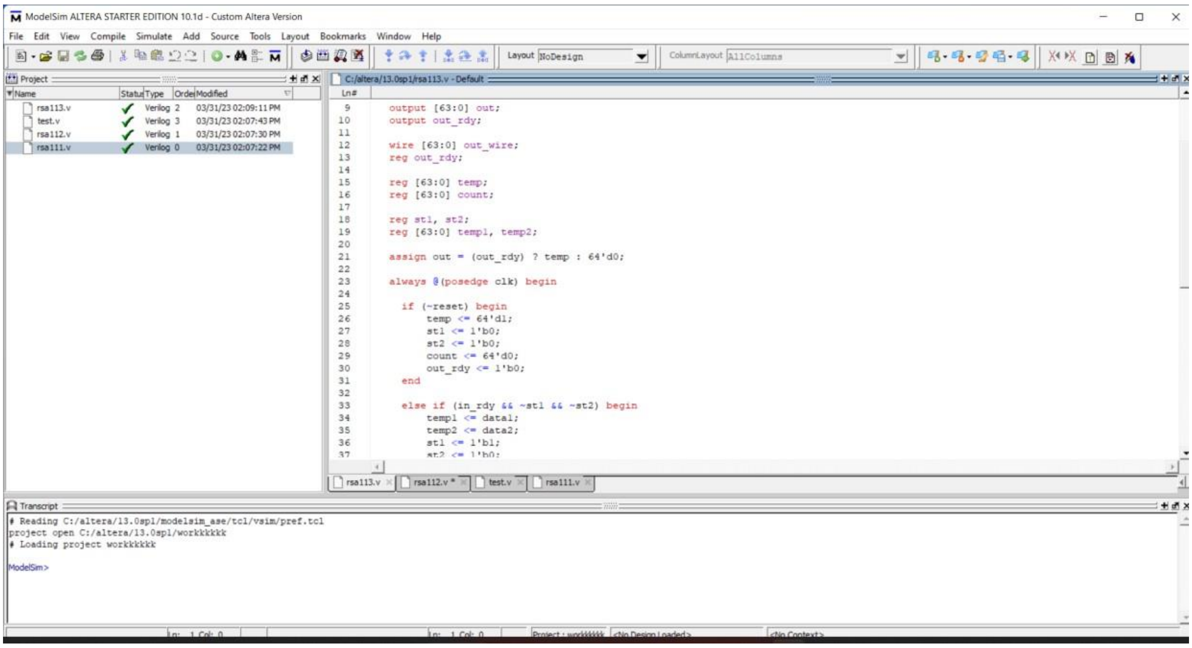

Figure 8: Power Process Module

The above module performs all the power operations

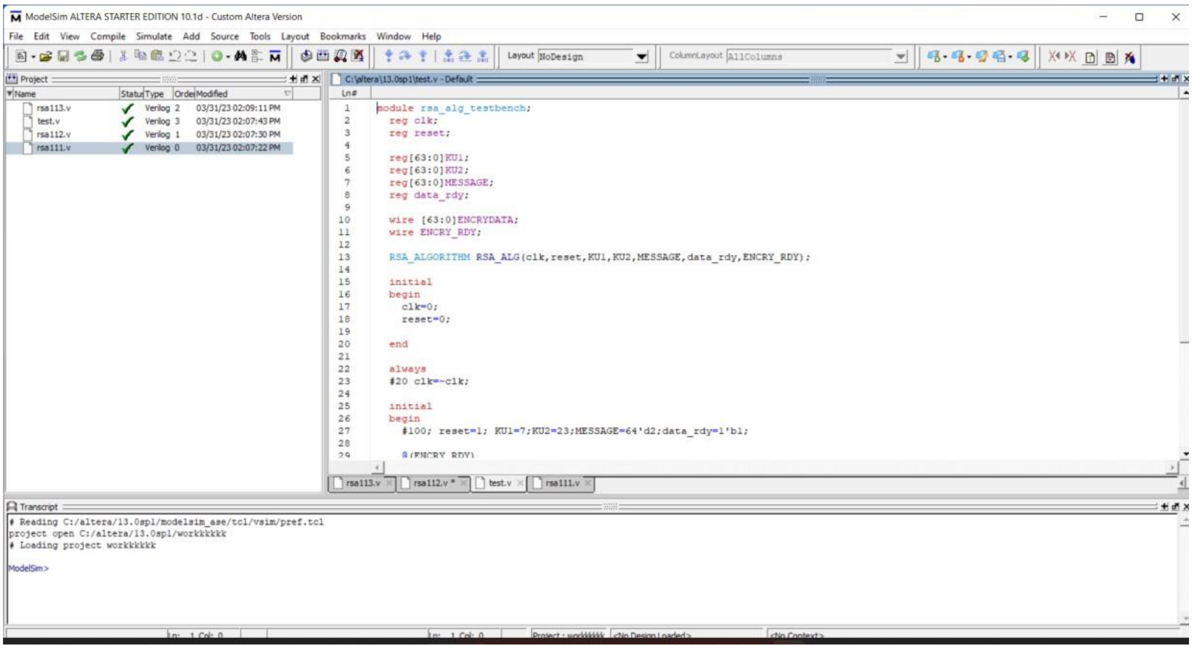

Figure 9: Testbench Module

The above code is for taking two sample prime numbers as input and obtaining the desired values from all basic RSA operations

Table 2: Input Signals

ENCRY_DATA	the encrypted message
ENCRY_RDY	a signal indicating that the ENCRY_DATA output is valid
mod_num1,2	registers holding the input to the modular operation
power_num1,2	registers holding the input to the power operation
mod_rdy	a signal indicating that the modular operation has completed
mod_out	the output of the modular operation
power_out:	the output of the power operation
power_rdy:	a signal indicating that the power operation has completed
count	a counter used to keep track of the number of iterations in the power operation
power_value	a register holding the output of the power operation
mod_value	register holding the output of the modular operation

Table 3: Output Signals

l

Clk	the clock signal
Reset	the reset signal
KU1	the first part of the public key
KU2	the second part of the public key
MESSAGE	the message to be encrypted
data_rdy	signal indicating that the MESSAGE input is ready for processing

The above tables contain the input signals and output signals required for generating the wave on Modelsim waveform window.

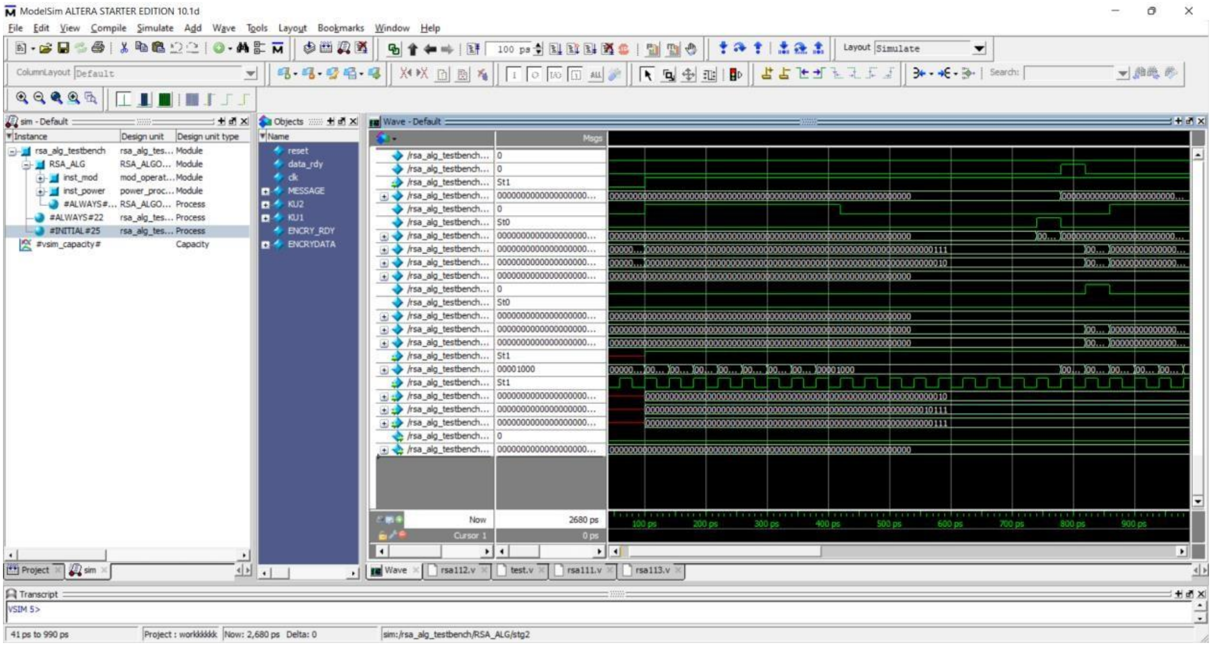

Figure 10: Waveform with input and output signals

The Verilog code has been further implemented on Quartus Prime Software to obtain the RTL Schematic and perform analysis of transistors, timing values, etc.

For the code to run in this software we set “mod_operations_rsa” module as top-level entity as it performs the main modular operations.

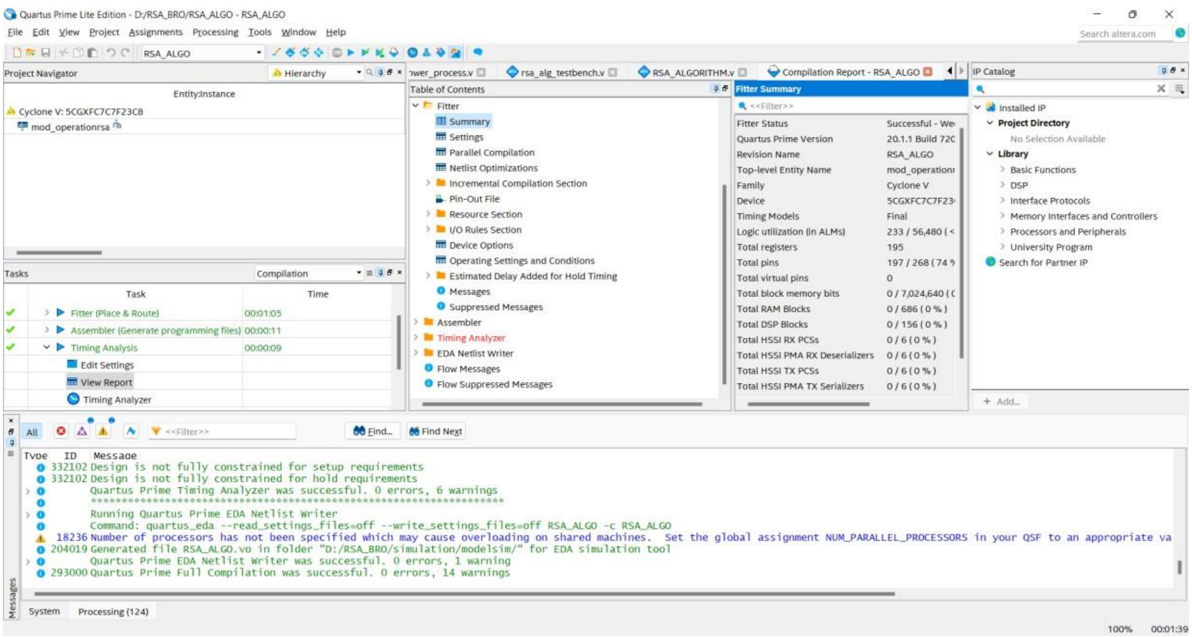

Figure 11: QUARTUS implementation

On IBM Quantum Lab, a python code has been implemented at quantum level to obtain the circuit at quantum level using classical and quantum registers.

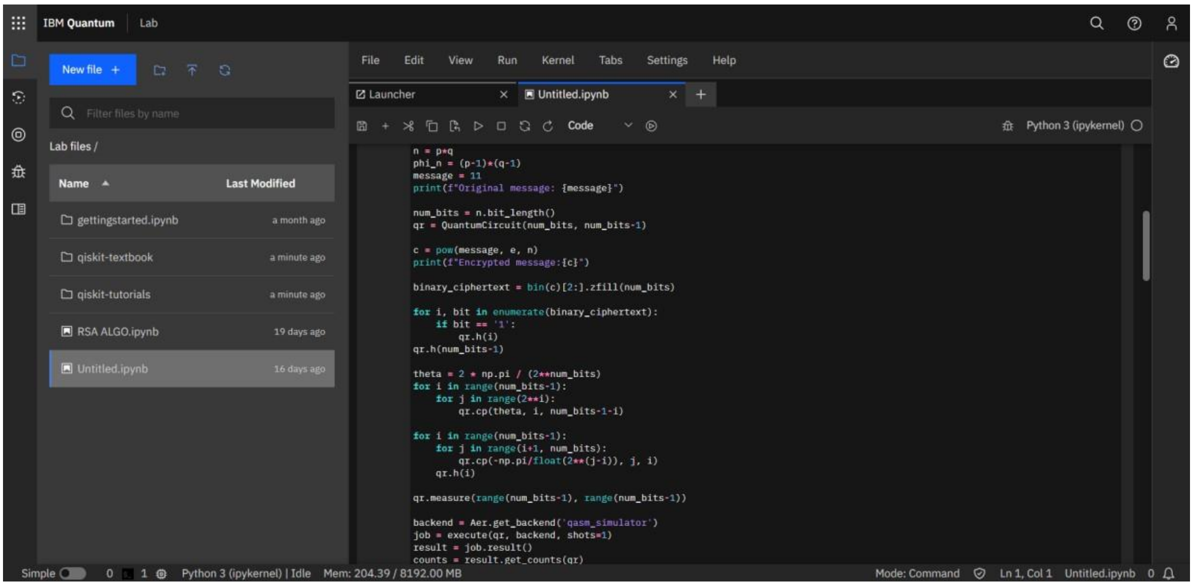

Figure 12: Python implementation on IBM Quantum Lab

Some standard QISKIT libraries have been used for the code to run:

from qiskit import QuantumCircuit, transpile
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *
from qiskit_aer import AerSimulator

We load our IBM account with the following line:

service = QiskitRuntimeService(channel="ibm_quantum")

6. RESULT & DISCUSSION

Following are results for the RSA algorithm implemented on Verilog.
Numerical Analysis of the ModelSim Testbench:
We have taken 2 prime numbers P= 5 and Q= 7,along with a message signal M=2:
Now calculating for N= P*Q= 5*7=35
Phi(N)=(P-1)(Q-1)= 4*6= 24
Next, we choose e such that GCD{Phi(N),e}= 1; 24=2*2*2*3; e cannot be 2,3,4, choose e=5; [Greatest Common Divisor] Next, we find the encrypted Cipher text by applying C= (M^e)mod(N):
(2^5)mod35= 32mod35 = 32;
Thus, the encrypted value we obtained is 32(or)100000.

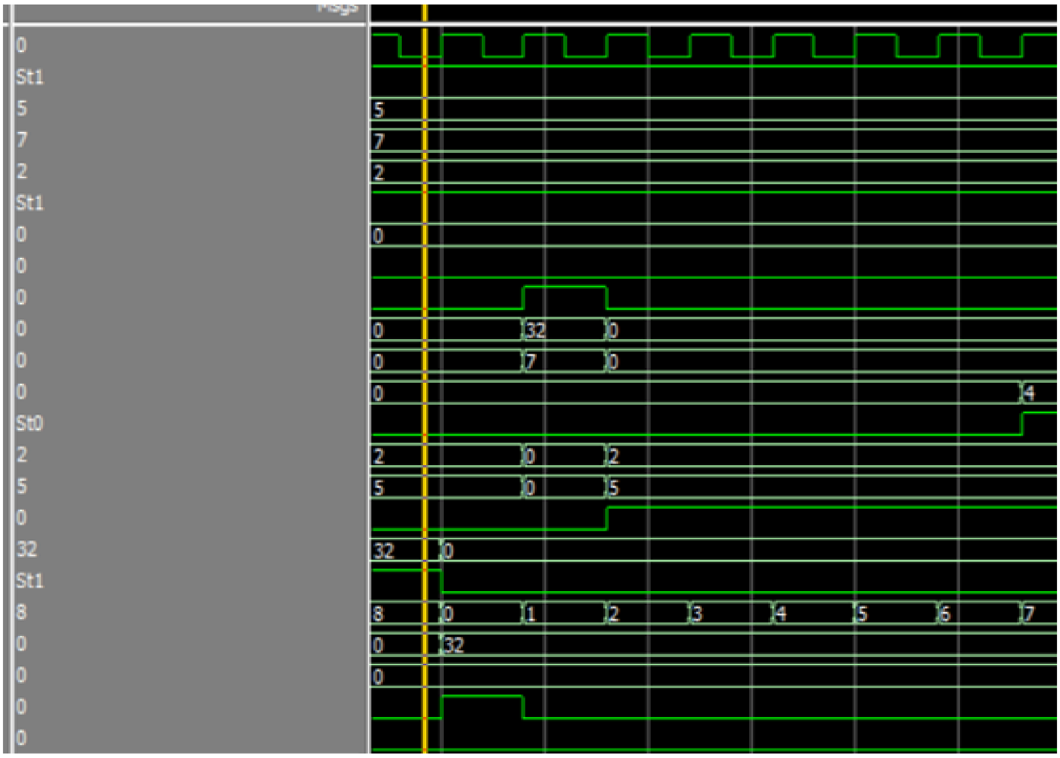

Figure 13: Modelsim Waveform for P= 5, Q=7

Like the previous case,
We gather prime numbers, P=3, Q= 23 and a message signal M=6

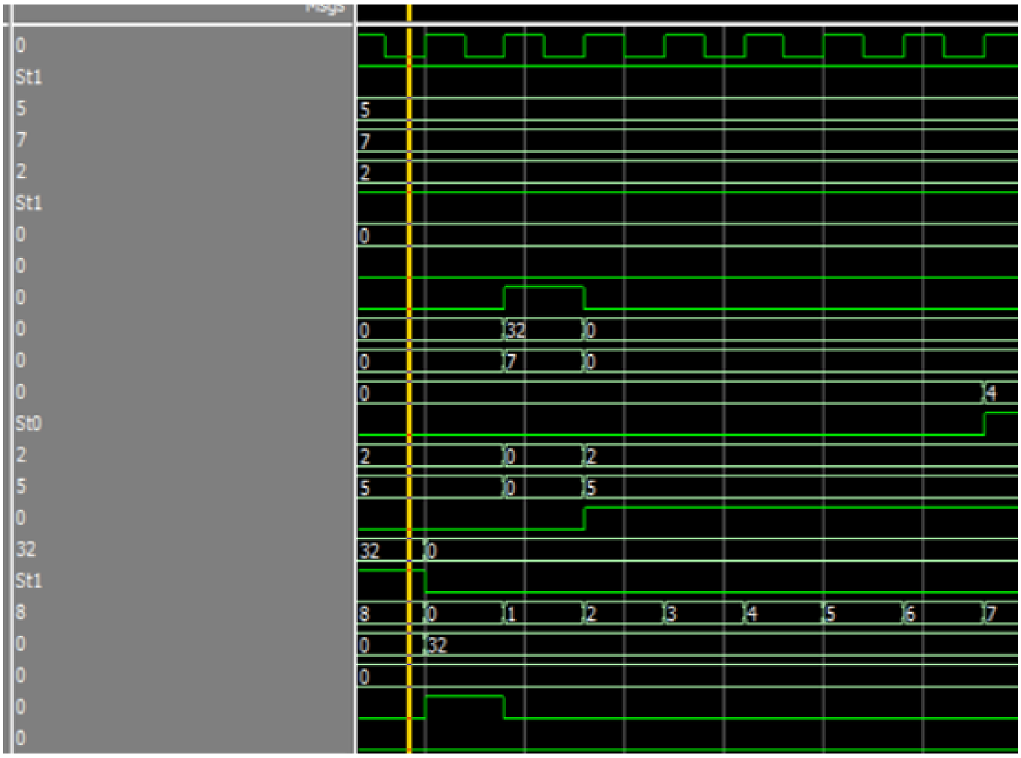

Figure 14: Waveform obtained on Modelsim for P= 3, Q= 23

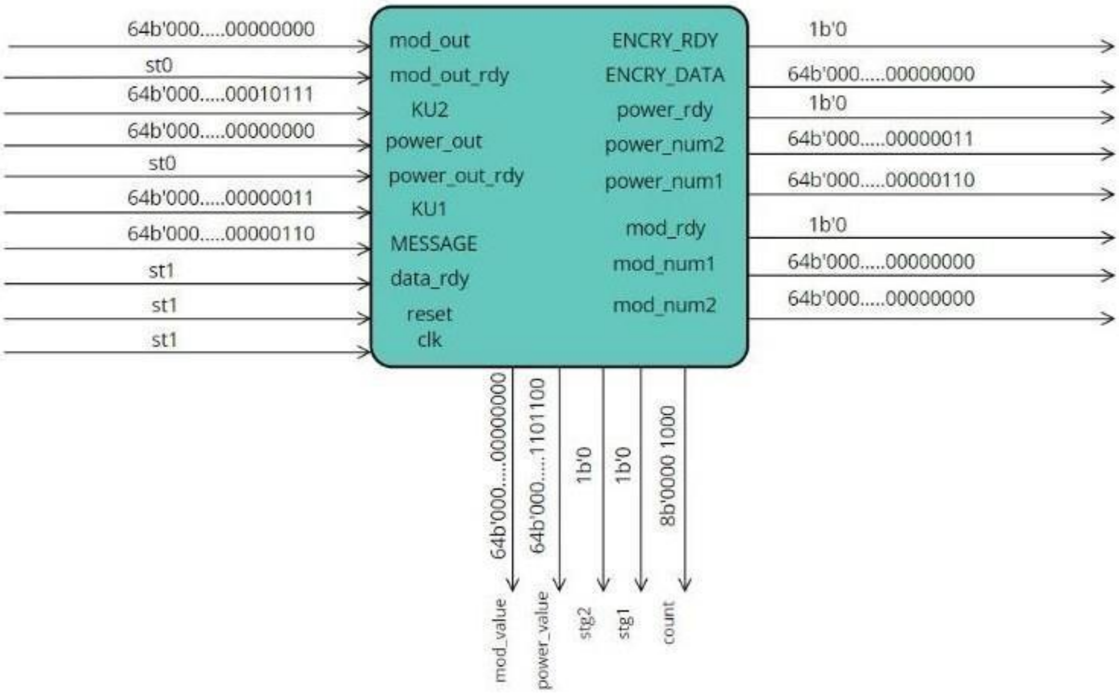

Figure 15: Dataflow of the Verilog code

A very complex RTL design has been obtained on QUARTUS PRIME. From the design we were able to obtain timing values and analysis and synthesis of various parameters. We have shared a drive link for the circuit with a minor portion of it shown below:

https://drive.google.com/file/d/1F_OVW9MJLHppKOwaic5apt22jmrbq_CJ/view?usp=sharing

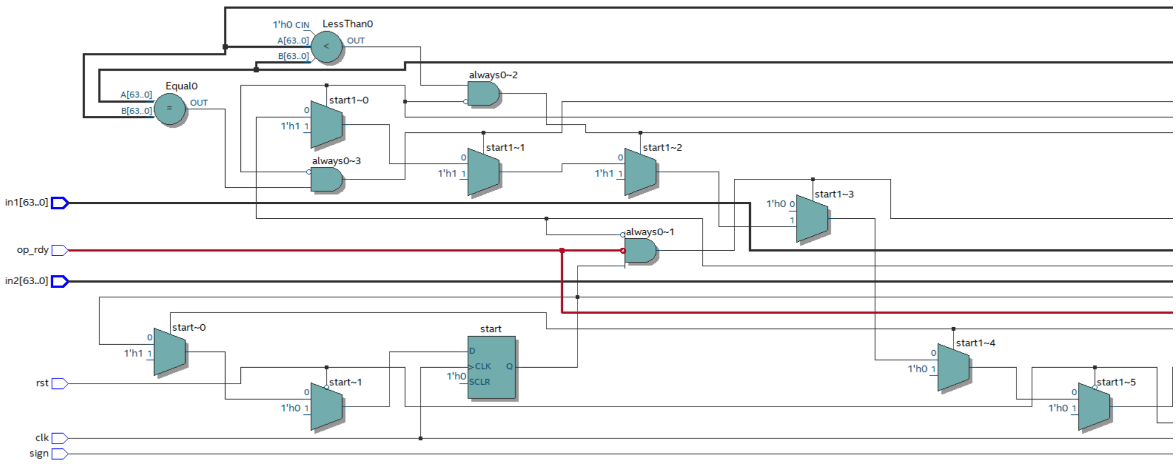

Figure 16: RTL design

Table 4. Analysis and Synthesis Usage Report Summary

COMPONENT	Nos.
Estimate of Logic utilization (ALMs needed)	216
Registers	195
Total I/O Pins	197
Input ports	68
Output ports	33
Combinational ALUT usage for logic	301
Total fan-out	2389
Number Detected on Machine	1231
Maximum Allowed	8
Processor 1,2-4	100%, 0%
ALMs used for LUT logic	30

Table 5: Timing Analysis

PERIOD	1000
MAXIMUM FREQUENCY	1000 MH
RISE	0.0
FALL	0.500
SPEED GRADE	4

Table 6: Register Statisics

Number of registers using Synchronous Clear	195
Number of registers using Synchronous Load	193
Number of registers using Clock Enable	192
Number of registers using Asynchronous Clear Number of registers using Asynchronous Load	0

Maximum Frequency: It refers to the maximum clock speed at which a digital circuit can operate without violating its timing requirements. It is typically measured in MHz or GHz. Number of slice flip flops: Slices are the basic building blocks of digital circuits in FPGA (Field Programmable Gate Arrays). A slice flip-flop is a memory element that can store a single binary bit value, and it is used to synchronize the data with the clock. The number of slice flip-flops in an FPGA design is an important metric that affects its performance and resource utilization.

Number of LUTs: LUT (Lookup Table) is a small memory unit used to store the logic functions in digital circuits. The number of LUTs in an FPGA design is a measure of its complexity and resource utilization.

Period: It refers to the time taken by a clock signal to complete one cycle. The period is the inverse of the clock frequency and is typically measured in nanoseconds.

Speed Grade: In FPGA design, the speed grade refers to the maximum operating frequency at which an FPGA device can operate. The higher the speed grade, the faster thedevice can operate.

Table 7. Comparison Parameters from our work

Parameter	Value
Maximum Frequency	1000 MHz
Number of slice flip flops	195
Number of LUTs	301
Period	1.000 ns
Speed Grade	4

From the comparative analysis of the papers[16-20], we have inferred the following:

The first paper involves 128 bit and used Bit-Serial Systolic Algorithm. Our inference from this comparison was our sample had a higher frequency, lesser slices, LUTs and lower time period.

The second paper involves 64 bit and used right to left algorithm. Our inference from this comparison was our sample had a higher frequency, lesser slices, LUTs and lower time period.

The third paper involves 128-bit simple shift and add algorithm. Our inference from this comparison was our sample had a higher frequency, lesser slices, LUTs and lower time period.

The fourth paper involves 1024-bit secured communication system based on Montgomery algorithm. Our inference from this comparison was our sample had a higher frequency, lesser slices, LUTs and lower time period.

The fifth paper involves 64-bit hardware base application RSA encryption algorithm. Our inference from this comparison was our sample had a higher frequency, lesser slices, LUTs and lower time period. We have tabulated these comparisons and have also obtained percentage increase/decrease with respect to our findings:

Table 8. Comparison with exisiting papers

Parameters	Original	Paper-1[16]	Paper 2[17]	Paper-3[18]	Paper-4[19]	Paper-5[20]
Maximum Frequency	1000 MHz	101.061 MHz	58 MHz (128 bits)	81.06 MHz	69.093MHz	79.54 MHz
Number of Slice flip flops	195	2366	432	2943	8956	277
Number of LUTs	301	4325	568	4325	8032	3083
Period	1.000 ns	9.85 ns	12.473 ns	9.879ns	14.473ns	12.60 ns
Speed Grade	-4	-4	-4	-4	Nil	Nil

Table 8. Comparative percentage increase of original paper

Comparative percentage increase of original paper	Paper-1[16] (%)	Paper-2[17] (%)	Paper-3[18] (%)	Paper-4[19] (%)	Paper-5[20] (%)
Maximum Frequency	888.939	1620.69	1137.48%	1343.83	1155.88
Number of Slice flip flops	-91.76	-54.86	-93.43	-97.83	-29.64
Number of LUTs	-93.045	-47.07	-93.03	-96.255	-90.263
Period	-89.847	-91.9827	-89.88	-93.0096	-92.06
Speed Grade	same	same	same	-	-

After the above section, we have done a brief study on quantum computing and its ability to break down high bit algorithms. We could not expand this section of our project due to the limitations of the Qubits allowed. The following results have been obtained from on Qiskit when we gave a simple test input to it.

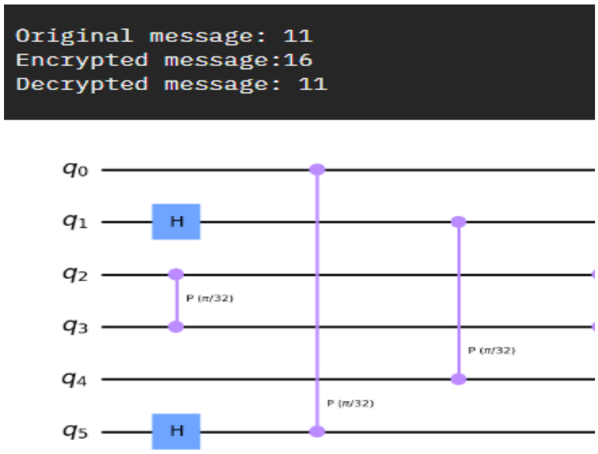

Figure 17: Results obtained on Qiskit

Here we have taken p = 5 q = 7 e = 23

The circuit generated by the circuit drawer function visual- izes the quantum circuit that implements the RSA encryption scheme using the Quantum Phase Estimation (QPE) algorithm. The circuit has a total of num bits qubits, where the first num bits-1 qubits are used to perform the QPE and the last qubit is used to implement the RSA encryption. The circuit starts by applying the Hadamard gate to the qubits corresponding to the ’1’ bits in the binary representation of the ciphertext. Then, the Hadamard gate is applied to the qubit corresponding to the public key parameter ’e’. Finally, the Inverse Quantum Fourier Transform (QFT) is applied to obtain the phase estimation result, which is used to calculate the decryption key using the multiplicative inverse function. The decrypted message is then obtained by applying the RSA decryption algorithm using the calculated decryption key.

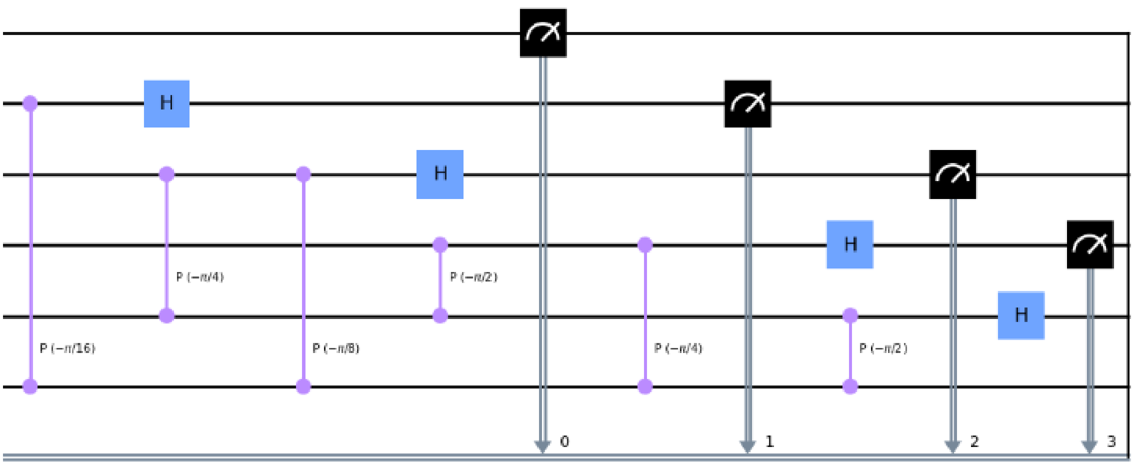

Figure 18: Quantum circuit of the algorithm

7. Summary

The Verilog level coding for RSA Encryption algorithm is done in 4 different modules. The desired waveforms have been obtained from the values in test bench. The maximum clock frequency is found to be about . We have implemented this in accordance to the fact that we are compromising security for the rewards of higher performance. As a result, we have used the best possible multiplier algorithm for the number of bits it is used. The key size although low can be compensated by techniques such as usage of secure key protocols, implementation of a multi factor authentication, usage of message authentication codes or a trusted hardware module. We have further proved our point by taking 5 samples of various bit sizes, and upon comparision, we have found out that our samples took less area and provided high frequency. Due to its less complexity, we have concluded that this design consumes less power, has a higher performance, but at the cost of good security. We have also studied how quantum computing has broke down every bit size up to 1024 bits, making every one of them vulnerable to be broken down in few hours or sometimes even minutes.

While this design sample has a fair share of advantages, it is still not recommended for its security concerns, and it is thus expected for the user to compensate the low bit key size with the above recommended compensation techniques, given in the introduction section for lost security aspect,while also keeping it appropriate by keeping the respective context in mind.

Regarding the quantum study, python code has been implemented in Qiskit to obtain the Encrypted and Decrypted message. A quantum level circuits are obtained using quantum phase estimation and Fourier Transforms.

8. References

[1] . M. Nanditha, U. Sanath Rao, M. Murali, R. Swathi and K. Chethana, "Design and Analysis of Digital Circuits using Quantum Cellular Automata and Verilog," *2020 7th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2020, pp. 205-210, doi: 10.23919/INDIACom49435.2020.9083709.

[2] N. Messaoudi, C. Crocker and M. Almendros, "A Hardware-Accelerated Qubit Control System for Quantum Information Processing," *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, Segovia, Spain, 2020, pp. 1-5, doi: 10.1109/DCIS51330.2020.9268643.

[3] K. Bertels *et al*., "Quantum Computer Architecture: Towards Full-Stack Quantum Accelerators," *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2020, pp. 1-6, doi: 10.23919/DATE48585.2020.9116502.

[4] Pilch, J., Długopolski, J. An FPGA-based real quantum computer emulator. *J Comput Electron*

[5] N. M. Kosaraju, M. Varanasi and S. P. Mohanty, "A high-performance VLSI architecture for advanced encryption standard (AES) algorithm," *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*, Hyderabad, India, 2006, pp. 4 pp.-, doi: 10.1109/VLSID.2006.9.

[6] K. K. Soni and A. Rasool, "Cryptographic Attack Possibilities over RSA Algorithm through Classical and Quantum Computation," *2018 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, Tirunelveli, India, 2018, pp. 11-15, doi: 10.1109/ICSSIT.2018.8748675.

[7] Bonnetain, X., Naya-Plasencia, M., & Schrottenloher, A. (2019). Quantum Security Analysis of AES. *IACR Transactions on Symmetric Cryptology*, *2019*(2), 55–93

[8] Khalid, M., Mujahid, U., Jafri, A. *et al.* An FPGA-based hardware abstraction of quantum computing systems. *J Comput Electron* **20**, 2001–2018 (2021).

[9] Y. Xu *et al*., "QubiC: An Open-Source FPGA-Based Control and Measurement System for Superconducting Quantum Information Processors," in *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1-11, 2021, Art no. 6003811, doi: 10.1109/TQE.2021.3116540.

[10] Yuchen Yang, Zhongtao Shen, Xing Zhu, Ziqi Wang, Gengyan Zhang, Jingwei Zhou, Xun Jiang, Chunqing Deng, Shubin Liu

[11] M. Huang, K. Gaj and T. El-Ghazawi, "New Hardware Architectures for Montgomery Modular Multiplication Algorithm," in *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 923-936, July 2011, doi: 10.1109/TC.2010.247.

[12] A. P. Fournaris and O. Koufopavlou, "A new RSA encryption architecture and hardware implementation based on optimized Montgomery multiplication," *2005 IEEE International Symposium on Circuits and Systems*, Kobe, Japan, 2005, pp. 4645-4648 Vol. 5, doi:

[13] A. P. Fournaris, "Fault and simple power attack resistant RSA using Montgomery modular multiplication," *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, Paris, France, 2010, pp. 1875-1878, doi: 10.1109/ISCAS.2010.5537879.

[14] Bertels, Koen & Ashraf, Imran & Nane, Razvan & Fu, Xiang & Riesebos, L. & Varsamopoulos, Savvas & Mouedenne, A. & van Someren, Hans & Sarkar, Aritra & Khammassi, Nader. (2019). Quantum Computer Architecture: Towards Full-Stack Quantum Accelerators.

[15] Y. Song, X. Hu, W. Wang, J. Tian and Z. Wang, "High-Speed and Scalable FPGA Implementation of the Key Generation for the Leighton-Micali Signature Protocol," *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, Daegu, Korea, 2021, pp. 1-5, doi: 10.11

[16] Sahu, Sushanta & Pradhan, Manoranjan. (2011). FPGA Implementation of RSA Encryption System. International Journal of Computer Applications. 19. 10-12. 10.5120/2391- 3173.

[17] Shams, Rehan & Khan, Fozia & Umair, Mohammad. (2013). Cryptosystem An Implementation of RSA Using Verilog. International Journal of Computer Networks and Communications Security.

[18] Yuliang, Deng and Mao Zhigang. "FPGA Implementation of RSA Encryption System." (2011).

[19] Ankit Anand, Pushkar Praveen, 2012, Implementation of RSA Algorithm on FPGA, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 01, Issue 05 (July 2012)

[20] Jyoti Kalia1 and Vikas Mittal2 1PG scholar, 1Department of Electronics and Communication Engineering, MMU University, Mullana – 133207, Haryana, India. 2Department of Electronics and Communication Engineering, MMU University, Mullana – 133207, Haryana, India International Journal of Electronics Engineering Research. ISSN 0975-6450 Volume 9, Number 8 (2017)