A study and practical implementation of hardware security for RISCV architectures on Zynq 7000 (ZedBoard) FPGA

DESIGN PROJECT

EEE F377

Author:

Sai kartik 2020A3PS0435P

May 8, 2024

Certificate

This is to certify that the report entitled "A study and practical implementation of hardware security for RISCV architectures on Zynq 7000 (ZedBoard) FPGA" and submitted by Sai Kartik ID No. 2020A3PS0435P, in partial fulfilment of the requirements of EEE F377 Design Project, embodies the work done by him under my supervision.

Signature of Supervisor

Date: May 8, 2024

Dr Nitin Chaturvedi

Associate Professor, BITS - Pilani, Pilani Campus

Acknowledgements

The success of any task depends on the encouragement and guidance of many people. I take this opportunity to express my sincere gratitude to the people who have been instrumental in enabling the successful completion of this project.

I wish to express my deepest gratitude and thanks to my guide, **Dr. Nitin Chaturvedi,** Associate Professor, Department of Electrical and Electronics Engineering, for providing me with his valuable insights and constantly monitoring the work's development by setting up precise deadlines. Without his constant guidance and support, this project would not have materialised. I would also like to thank my project partner, Rajeev Rajagopal, for his efforts in completing this work.

I would also like to express my sincere gratitude to the Department of Electrical and Electronics Engineering, BITS Pilani, Pilani Campus, for allowing me to work on this project and providing various resources that were instrumental in completing this work.

RISC-V

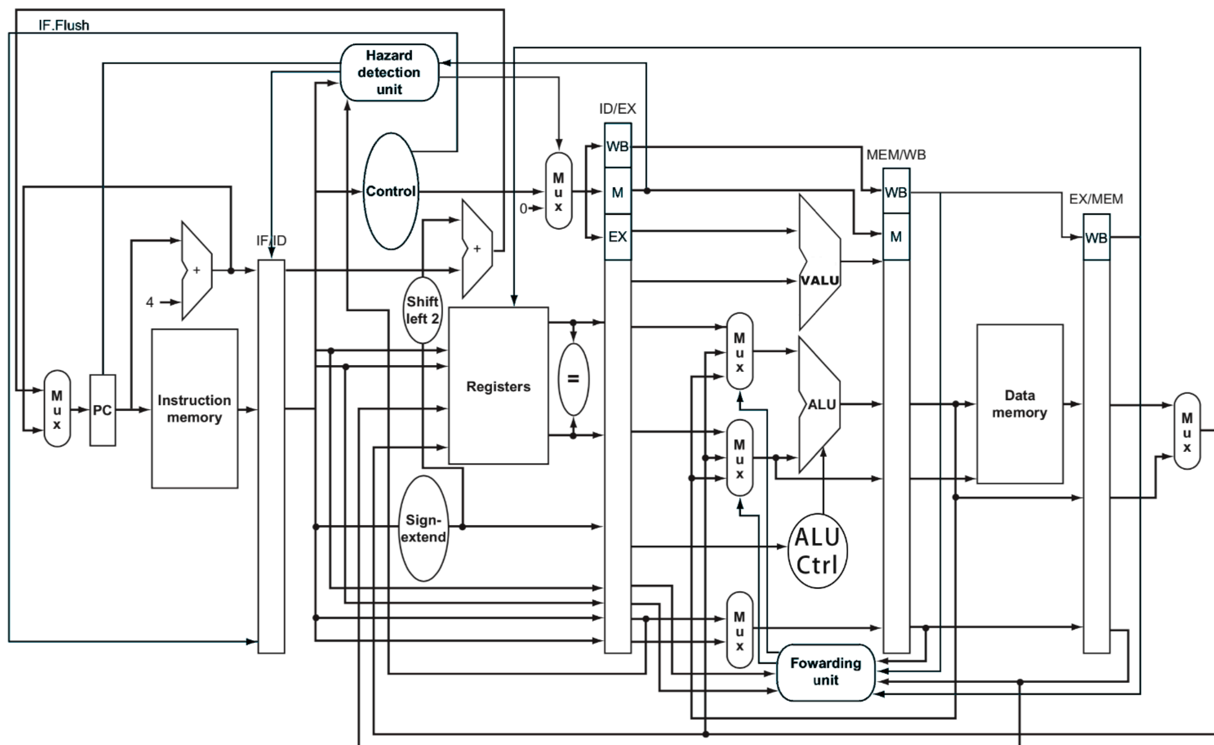
To provide an efficient solution to the growing need for security, open-source architectures such as RISC-V are widely implemented on edge-computing devices and platforms.

A soft-core implementation of the RISC-V architecture was made on the Zynq-7000 FPGA (Zedboard). The required resources for this activity are in their official GitHub repository.

Using the Berkely Bootloader (bbl), we can boot a RISC-V-Linux image on the Zynq-7000 with live access to the root terminal.

Another implementation of the RISC-V architecture can be found in this repository. This implementation follows an RTL approach to designing the CPU. Since the code is open source and freely available, we can modify it to our requirements, adding custom instructions if necessary.

Below is the block diagram of the module implementation:



This implementation supports all the basic instructions

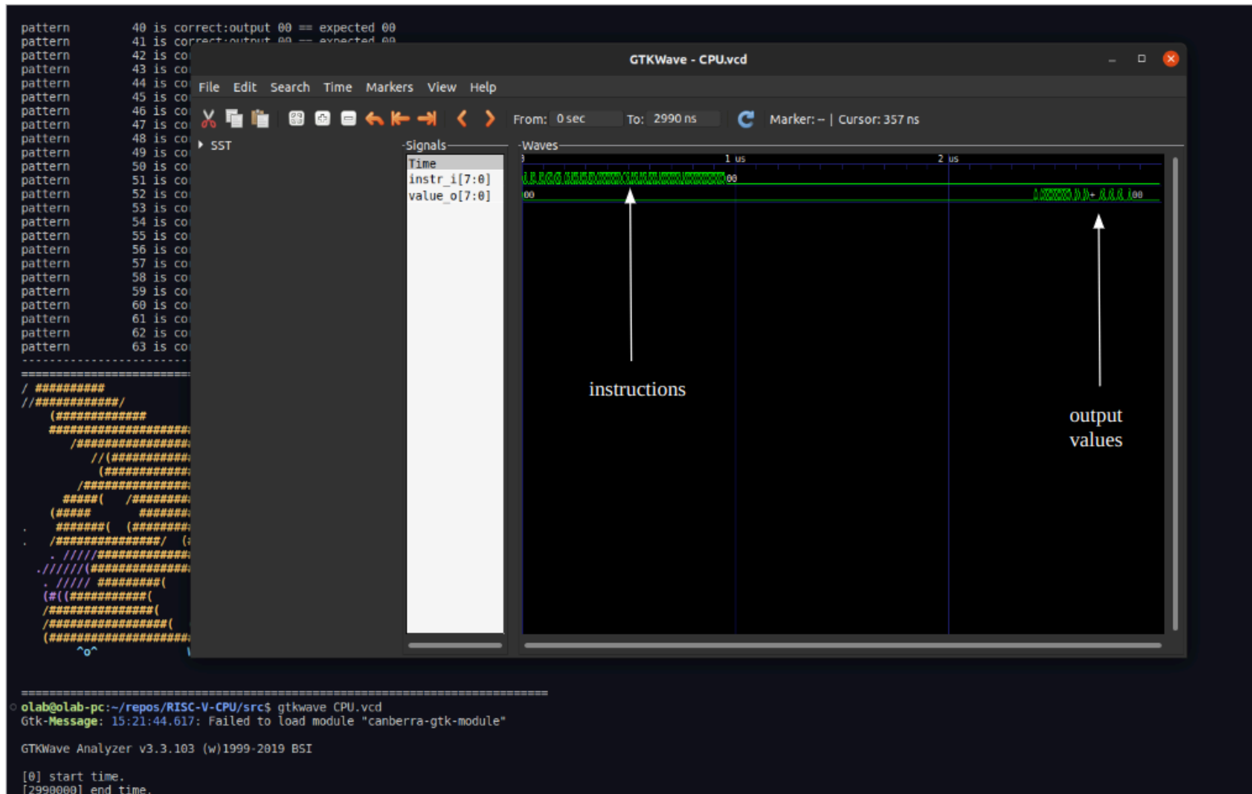
- addi
- sub
- or
- and
- lw
- sw
- beq
- mul

As well as vector arithmetic with commands:

- vsum
- vsub
- vsm (vector scalar multiplication)
- vdp (vector dot product)

Upon compiling the source files with Icarus Verilog, we can verify that the CPU successfully executes the pre-built instructions and gives the expected output.

```
olab@olab-pc:~/repos/RISC-V-CPU/src$ iverilog -o test CPU.v testbench.v
olab@olab-pc:~/repos/RISC-V-CPU/src$ vvp test
WARNING: testbench.v:57: $readmemb(../dat/instruction2.txt): Not enough words in the file for the requested range [0:257].
VCD info: dumpfile CPU.vcd opened for output.
-------------------------- [ Simulation Starts !! ] --------------------------
pattern          0 is correct:output 00 == expected 00
pattern          1 is correct:output 00 == expected 00
pattern          2 is correct:output 00 == expected 00
pattern          3 is correct:output 00 == expected 00
pattern          4 is correct:output 00 == expected 00
pattern          5 is correct:output 00 == expected 00
pattern          6 is correct:output 00 == expected 00
pattern          7 is correct:output 01 == expected 01
pattern          8 is correct:output 00 == expected 00
pattern          9 is correct:output 00 == expected 00
pattern         10 is correct:output 03 == expected 03
pattern         11 is correct:output db == expected db
pattern         12 is correct:output 23 == expected 23
pattern         13 is correct:output 6b == expected 6b
pattern         14 is correct:output 6e == expected 6e
pattern         15 is correct:output 12 == expected 12
pattern         16 is correct:output 23 == expected 23
```



Below is the RISC-V ISA format for different types of instructions.

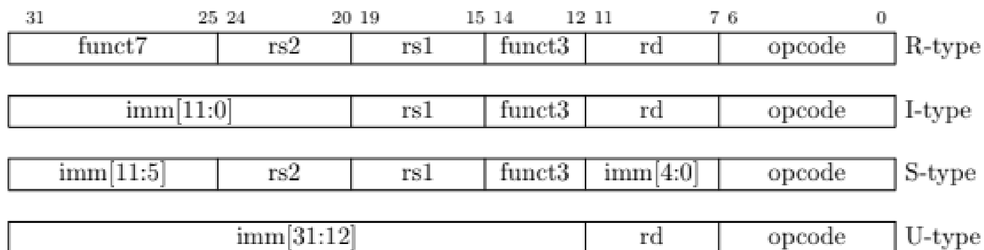


Figure 2.2: RISC-V base instruction formats. Each immediate subfield is labeled with the bit position (imm[x]) in the immediate value being produced, rather than the bit position within the instruction's immediate field as is usually done.

Upon investigation of the code base, it can be found that new instructions can be added with ease, given the knowledge of the control signals.

```
always@(*)begin

  case(Op_i)

  7'b0010011 : begin //addi
  ALUOp_o = 2'b11;
  ALUSrc_o = 1'b1;
  RegWrite_o = 1'b1;
  MemRd_o = 1'b0;
  MemWr_o = 1'b0;
  MemToReg_o = 1'b0;
  immSelect_o = 1'b0;
  end

  7'b0110011 : begin //others
  ALUOp_o = 2'b10;
  ALUSrc_o = 1'b0;
  RegWrite_o = 1'b1;
  MemRd_o = 1'b0;
  MemWr_o = 1'b0;
  MemToReg_o = 1'b0;
  immSelect_o = 1'b0;
  end

  7'b1100011 : begin //beq
  ALUOp_o = 2'b01;
  ALUSrc_o = 1'b1;
  RegWrite_o = 1'b0;
  MemRd_o = 1'b0;
  MemWr_o = 1'b0;
  MemToReg_o = 1'b0;
  immSelect_o = 1'b0;
  end

  7'b0000011 : begin //lw
  ALUOp_o = 2'b00;
  ALUSrc_o = 1'b1;
  MemRd_o = 1'b1;
  MemToReg_o = 1'b1;
  RegWrite_o = 1'b1;
  MemWr_o = 1'b0;
  immSelect_o = 1'b0;
  end
```

The control signals' description is given below:
ALUOp_o: ALU operation to be performed
ALUSrc_o: Choice of input source to the ALU
RegWrite_o: Whether or not to write to registers
MemRd_o: Whether or not to read memory
MemWr_o: Whether or not to write to memory
MemToReg: Data source from memory/registers
immSelect_o: 1 for sw, 0 for others

Encryption Algorithms

BLOWFISH ENCRYPTION ALGORITHM

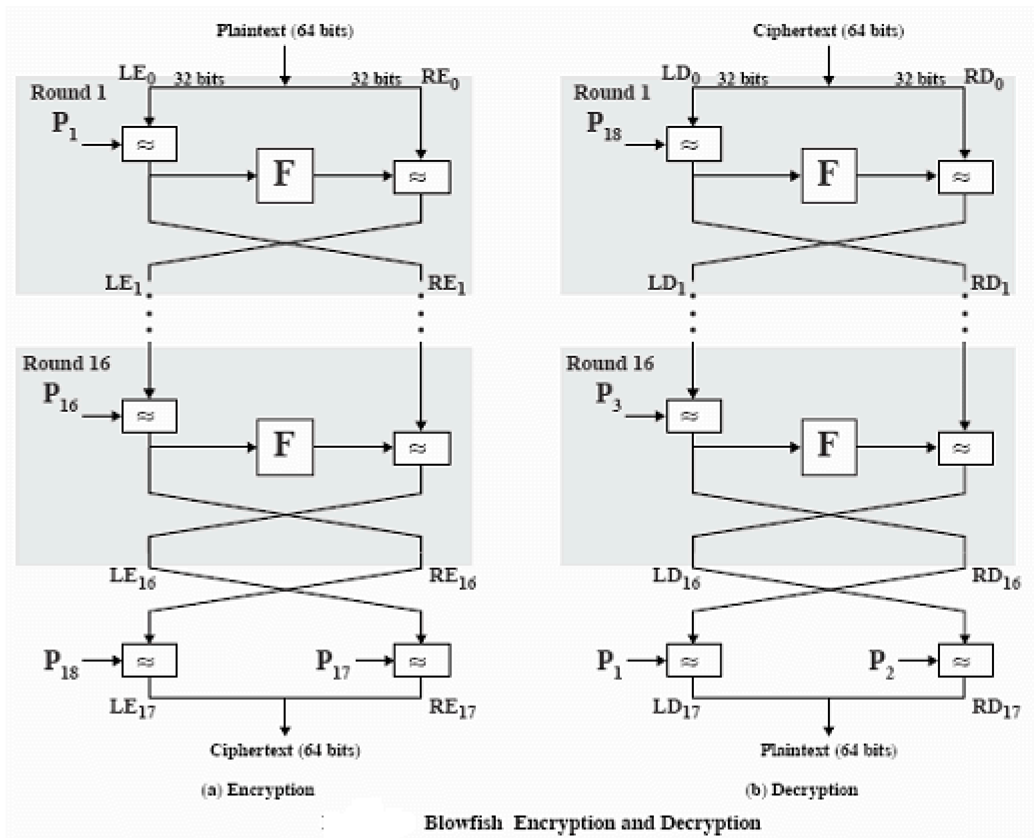
This symmetric cypher splits messages into blocks of 64 bits and encrypts them individually. Blowfish is known for its tremendous speed and overall effectiveness. Meanwhile, vendors have taken full advantage of its free availability in the public domain. You'll find Blowfish in software categories ranging from e-commerce platforms for securing payments to password management tools, where it protects passwords. It's one of the more flexible encryption methods available. Blowfish is significantly faster than DES and IDEA, unpatented, and is free for all uses.

It uses a variable-length key, from 32 bits to 448 bits. It consists of 16 Feistel-like iterations, where each iteration operates on a 64-bit block that's split into two 32-bit words. Blowfish uses a single encryption key to both encrypt and decrypt data.

Mechanism of the Blowfish cypher:

1. Initially, the input 'Hi world' consists of seven characters plus one space, equal to 64 bits or 8 bytes.
2. The input is split into 32 bits. The left 32 bits — "Hi w"— are XORed with P1, which is generated by key expansion to create a value called P1. (Note: P denotes prime number, which is not divisible except by 1 and itself.)
3. Then, PI runs through a transformative function (F In), in which the 32 bits are split into 4 bytes each and passed to the four s-boxes.
4. The first Iwo values from the first two S-boxes are added to each other and XORed with the third value from the third S-box.
5. This result Is added to the output of the fourth S-box to produce 32 bits as output.
6. The output of F In is XORed with the right 32 bits of the input message —'orld' — to produce output F1'.
7. Then, F1'. replaces the left half of the message, while P1' replaces the right half.
8. This same process is repeated for successive members of the P-array for 16 rounds in total.
9. Finally, after 16 rounds, the outputs P16'. and F16' are XORed with the last two entries of the P-array, i.e., P17 and Pl8. They are then recombined to produce the 64-bit ciphertext of the input message.

A flowchart representation of the blowfish algorithm has been shown below.



Blowfish Encryption and Decryption

There are some downsides to using Blowfish for encryption, including the following:

- Speed is affected when changing keys.
- The key schedule takes a long time.
- The small 64-bit block size makes the algorithm vulnerable to birthday attacks, a class of brute-force attacks.
- Each new key requires preprocessing equivalent to 4 KB of text, which affects its speed, making it unusable for some applications.

RSA ENCRYPTION ALGORITHM:

RSA is a public-key encryption algorithm and the standard for encrypting data sent over the internet. It is also one of the methods used in PGP and GPG programmes. Unlike Triple DES, RSA is considered an asymmetric algorithm because it uses a pair of keys. You have a public key to encrypt the message and a private key to decrypt it. RSA encryption results in a massive batch of mumbo jumbo that takes attackers a lot of time and processing power to break.
https://www.geeksforgeeks.org/rsa-algorithm-cryptography/?ref=lbp

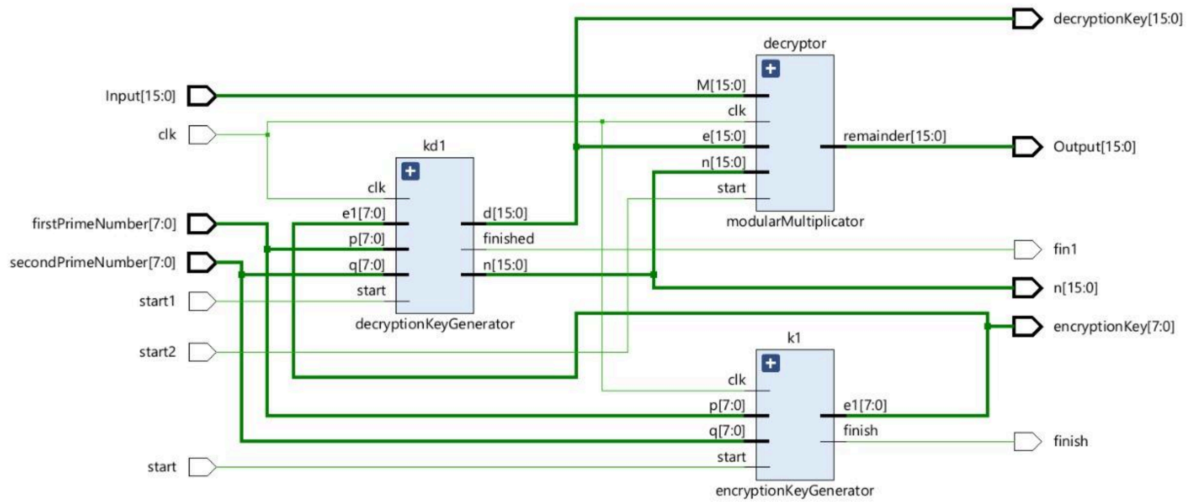
The public key is used for encryption, and the private key is used for decryption. Decryption cannot be done using a public key. The two keys are linked, but the private key cannot be derived from the public key. The public key is well known, but the private key is secret and is known only to the user who owns it.

- Choose p = 3 and q = 11
- Compute n = p * q = 3 * 11 = 33
- Compute φ(n) = (p - 1) * (q - 1) = 2 * 10 = 20
- Choose e such that 1 < e < φ(n) and e and φ (n) are coprime. Let e = 7
- Compute a value for d such that (d * e) % φ(n) = 1. One solution is d = 3 [(3 * 7) % 20 = 1]
- Public key is (e, n) => (7, 33)
- Private key is (d, n) => (3, 33)
- The encryption of $m = 2$ is $c = 2^7 \% 33 = 29$
- The decryption of $c = 29$ is $m = 29^3 \% 33 = 2$

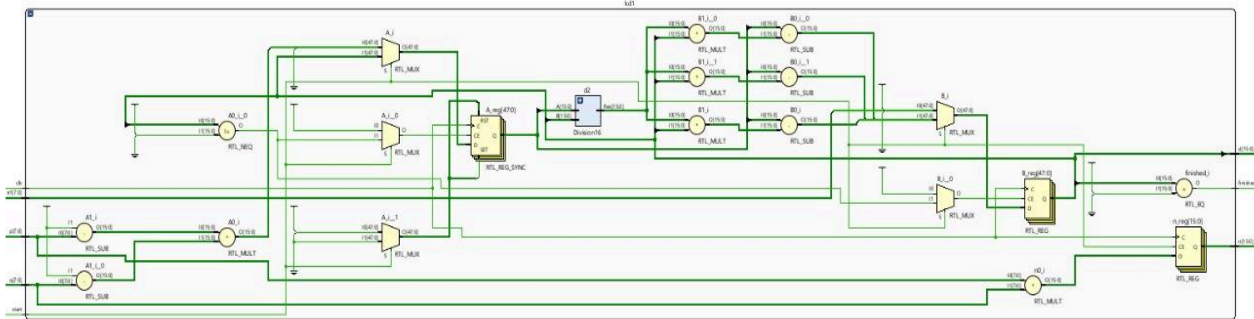
Disadvantages:

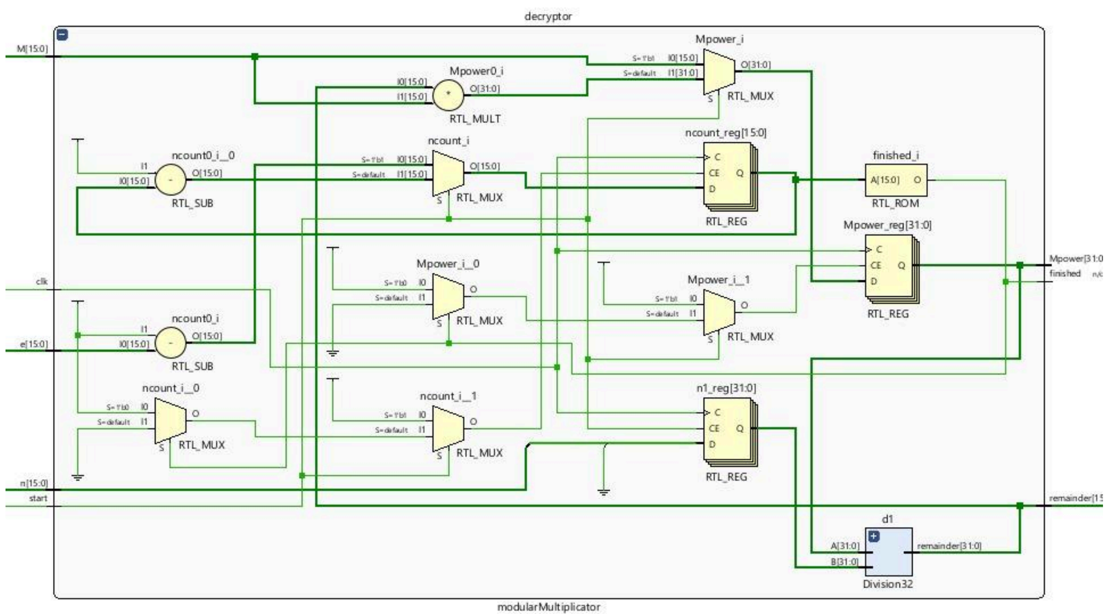
- Slow processing speed: The RSA algorithm is slower than other encryption algorithms, especially when dealing with large amounts of data.
- Large key size: The RSA algorithm requires large key sizes to be secure, requiring more computational resources and storage space.
- Vulnerability to side-channel attacks: The RSA algorithm is vulnerable to side-channel attacks, which means an attacker can use information leaked through side channels, such as power consumption, electromagnetic radiation, and timing analysis, to extract the private key.
- Limited use in some applications: Due to its slow processing speed, the RSA algorithm is unsuitable for applications requiring constant encryption and decryption of large amounts of data.
- Complexity: The RSA algorithm is a sophisticated mathematical technique that some individuals may find challenging to comprehend and use.
- Key Management: The secure administration of the private key is necessary for the RSA algorithm, although in some cases, this can be difficult.

The following figure shows the block diagram representation of the RSA algorithm implemented in Verilog:

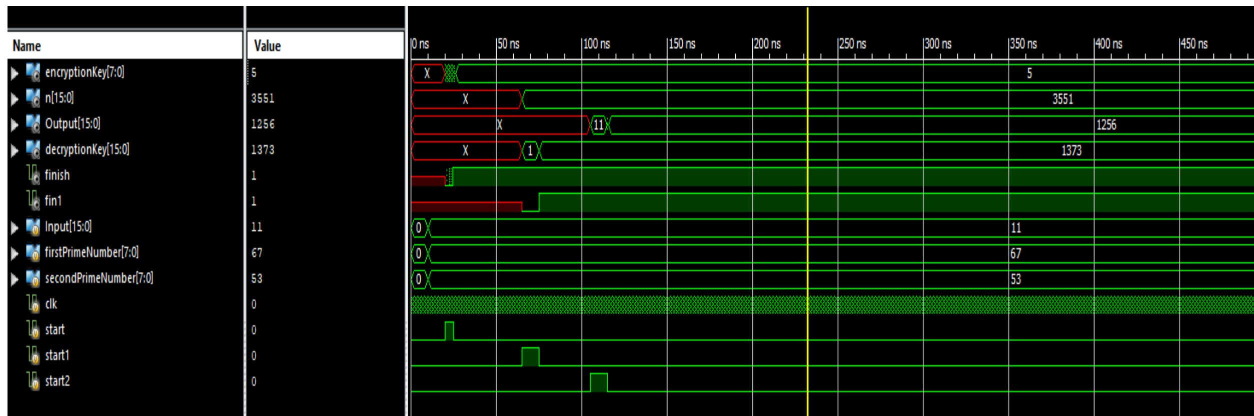


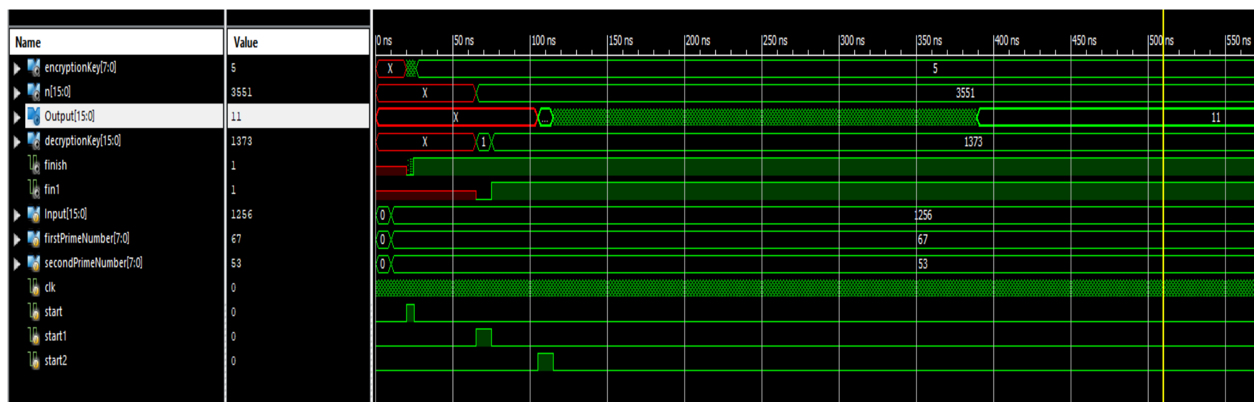
The different modules of the RSA hardware module are shown below:





The timing diagrams, as simulated on Vivado, for the RSA encryption module for the encryption algorithm and the decryption algorithm have been shown below, respectively:





Future Scope

We have observed and detailed a possible solution for implementing the same to achieve better results and performance regarding encryption and security applications.

To advance this project, we can further investigate the various encryption modules available and interface them with hardware accelerators. A soft-core implementation can also be used with programmable logic to decrease computational time on complex calculations while performing an encryption or decryption operation.

RISC-V architecture offers distinct advantages for securing edge computing devices. Its openness and transparency allow for thorough scrutiny and verification of security mechanisms, which is vital for ensuring resilience against vulnerabilities in hostile environments. Customizability enables tailored security features such as cryptographic accelerators, while minimalistic instruction sets reduce the attack surface, which is crucial for resource-constrained devices. Additionally, features like privilege levels and memory protection ensure robust isolation, while support for secure boot and trusted execution

safeguards against unauthorised access and ensures the integrity of critical operations. Furthermore, the collaborative ecosystem around RISC-V promotes sharing security best practices, fostering the development of robust security solutions for edge computing.