數位電路實驗
Lab 2
RSA256解密機

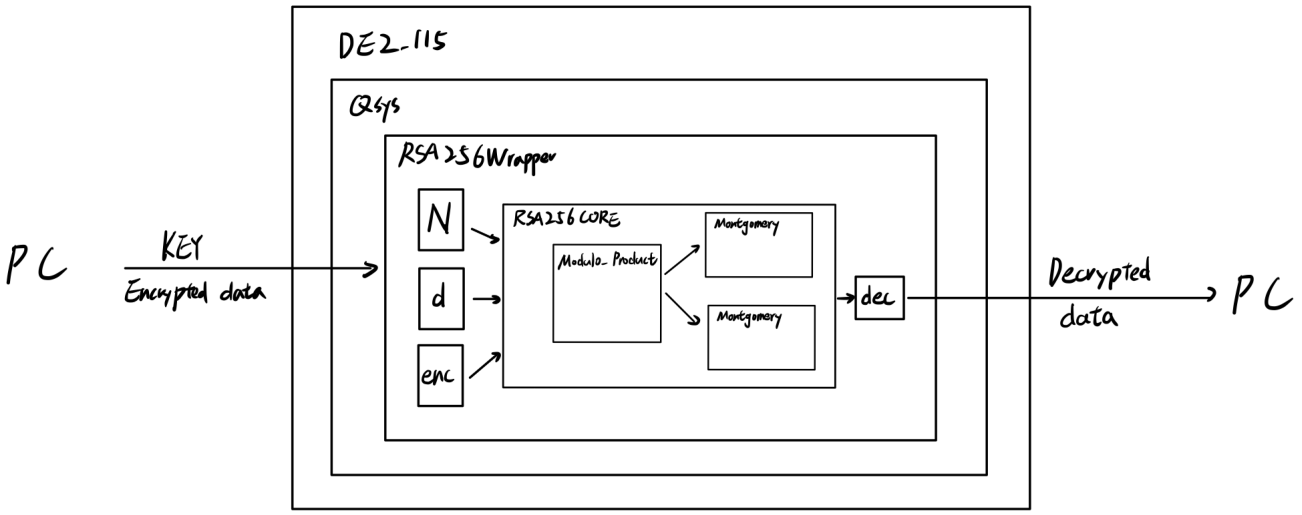
Team 02

金家逸 B10502076
王維勤 B10502010
林桓鈺 B10502013

1. Introduction:

In this experiment, we implement a machine capable of receiving RSA-256 encrypted data sent by a computer via an RS232 transmission cable and decrypting it, and then returning the decoded information back to the computer.

2. System Architecture:



(1) ModuloProduct.sv :

Calculate ab (mod k), replace multiplication with additions.

Algorithm 2 Modulo of products

```
1:  function MODULOPRODUCT(N, a, b, k)                ▷ k is number of bits of a
2:      t ← b
3:      m ← 0
4:      for i ← 0 to k do
5:          if i-th bit of a is 1 then
6:              if m + t ≥ N then
7:                  m ← m + t − N                     ▷ perform modulo operation in each iteration
8:              else
9:                  m ← m + t
10:             end if
11:         end if
12:         if t + t > N then
13:             t ← t + t − N                         ▷ perform modulo operation in each iteration
14:         else
15:             t ← t + t
16:         end if
17:     end for
18:     return m
19: end function
```

(2) MontgomeryAlgorithm.sv :

The Montgomery Algorithm is a method used for performing modular exponentiation to compute a^b mod n where n is the modulus.

Algorithm 3 Montgomery algorithm for calculating $ab2^{-256} \bmod N$

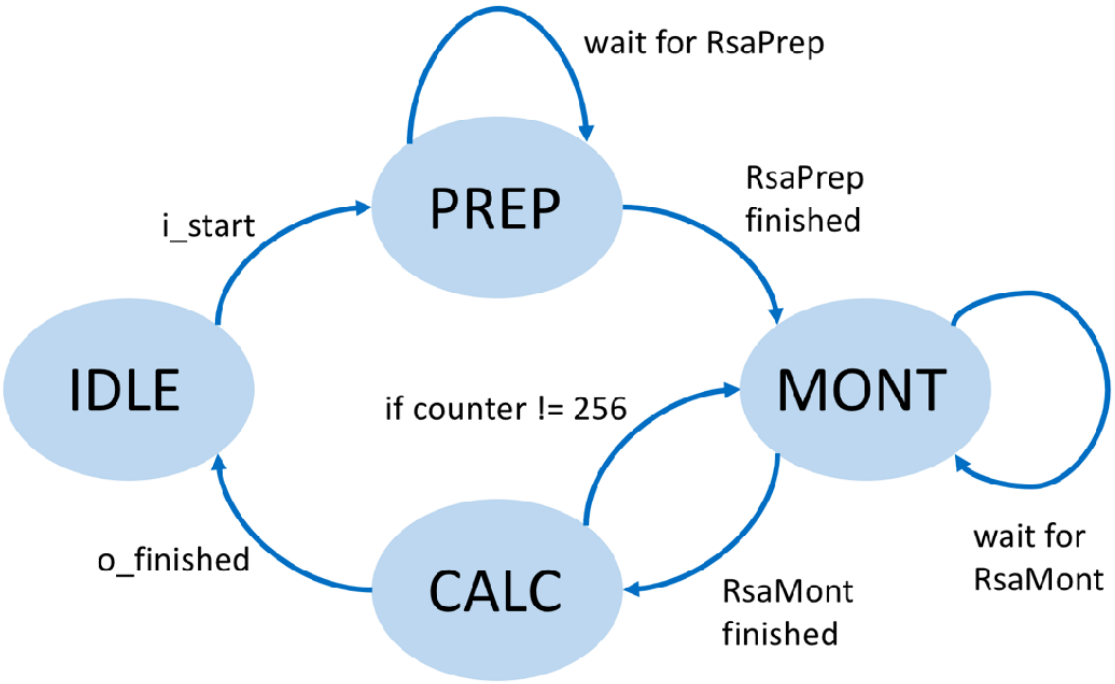
```
1:  function MONTGOMERYALGORITHM(N, a, b)
2:      m ← 0
3:      for i ← 0 to 255 do
4:          if i-th bit of a is 1 then
5:              m ← m + b
6:          end if                                    ▷ 4~6: replace multiplication with
                                                        successive addition
7:          if m is odd then
8:              m ← m + N
9:          end if
10:         m ← m/2                                   ▷ 7~10: calculate the modulo of a·2^−1
                                                        → Montgomery reduction
11:     end for
12:     if m ≥ N then
13:         m ← m − N
14:     end if
15:     return m
16: end function
```

(3) Rsa256Core.sv :

Send N, d, and the ciphertext to Rsa256Core for decryption, then transmit the decrypted plaintext back to Rsa256Wrapper.

Algorithm 4 RSA256 with exponentiation by squaring and Montgomery algorithm

```
1:  function RSA256MONT(N, y, d)
2:      t ← ModuloProduct(N, 2^256, y, 256)   t = y * 2^256
3:      m ← 1
4:      for i ← 0 to 255 do
5:          if i-th bit of d is 1 then
6:              m ← MontgomeryAlgorithm(N, m, t)   m * t * 2^-256 (modN)
7:          end if
8:          t ← MontgomeryAlgorithm(N, t, t)   t * t * 2^-256 (modN)
9:      end for
10:     return m
11: end function
```

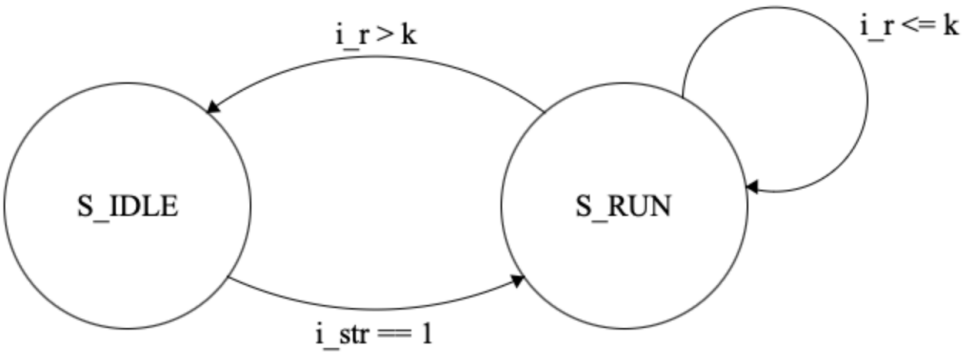


(4) Rsa256Wrapper.sv

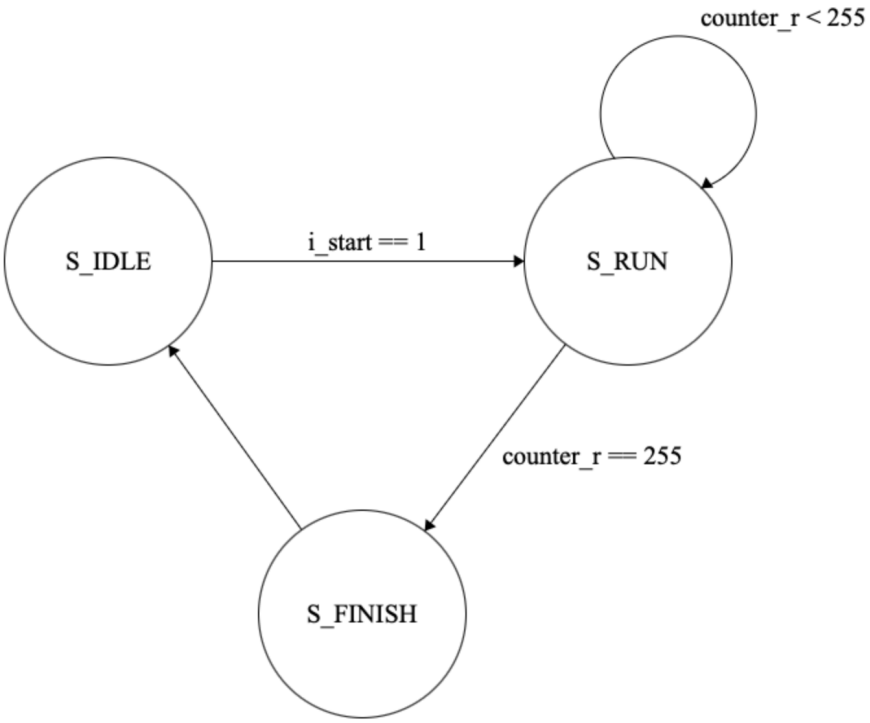
Encapsulated outside Rsa256Core, responsible for transmitting and receiving data and keys. It sends data and keys to Rsa256Core and receives the plaintext sent back from Rsa256Core.

3. Finite state machine:

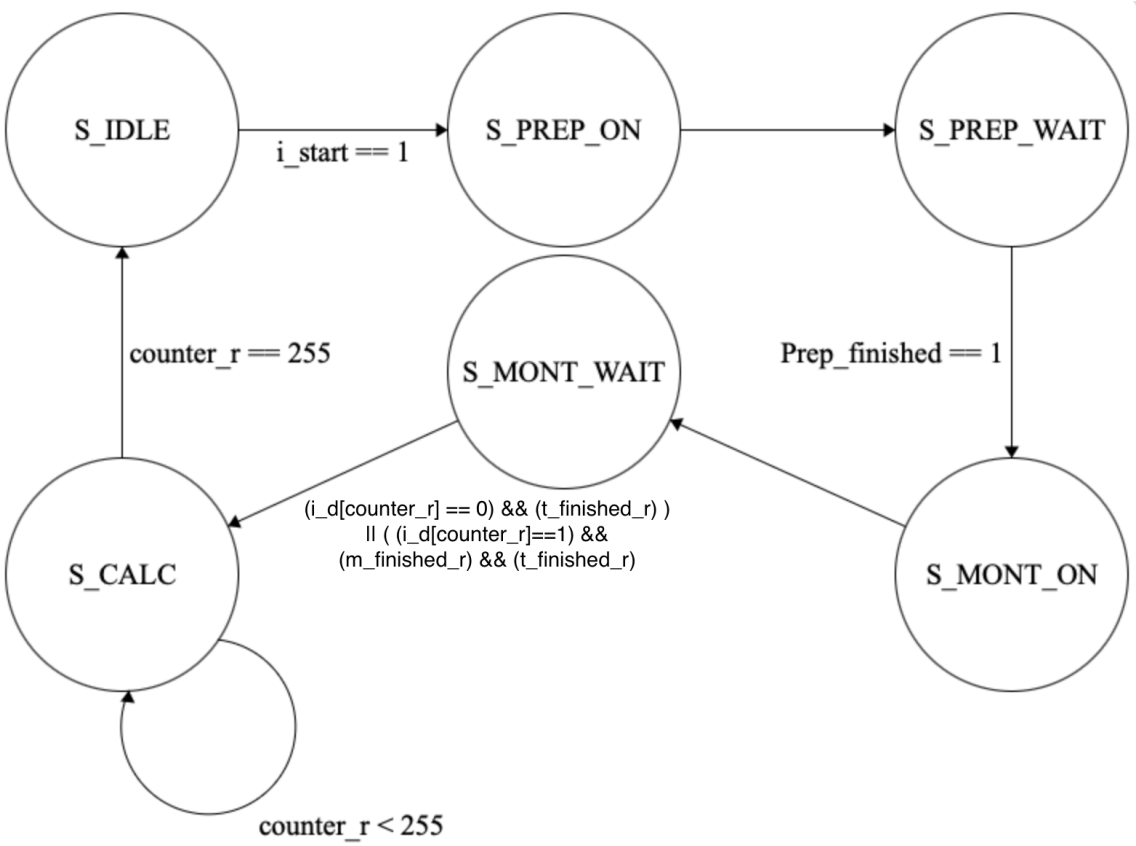
a. MouloProduct.sv



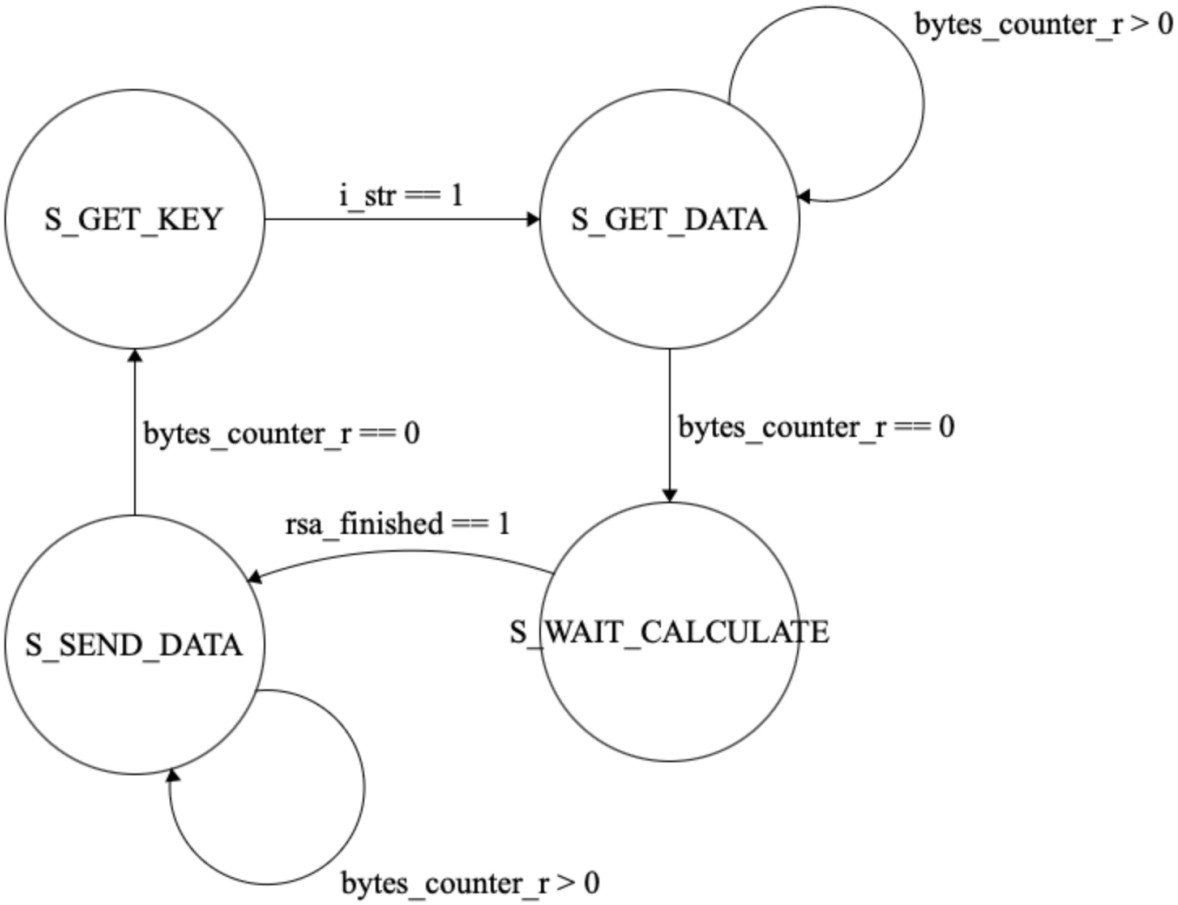
b. MontgomeryAlgorithm.sv



c. Rsa256Core.sv



d. Rsa256Wrapper.sv

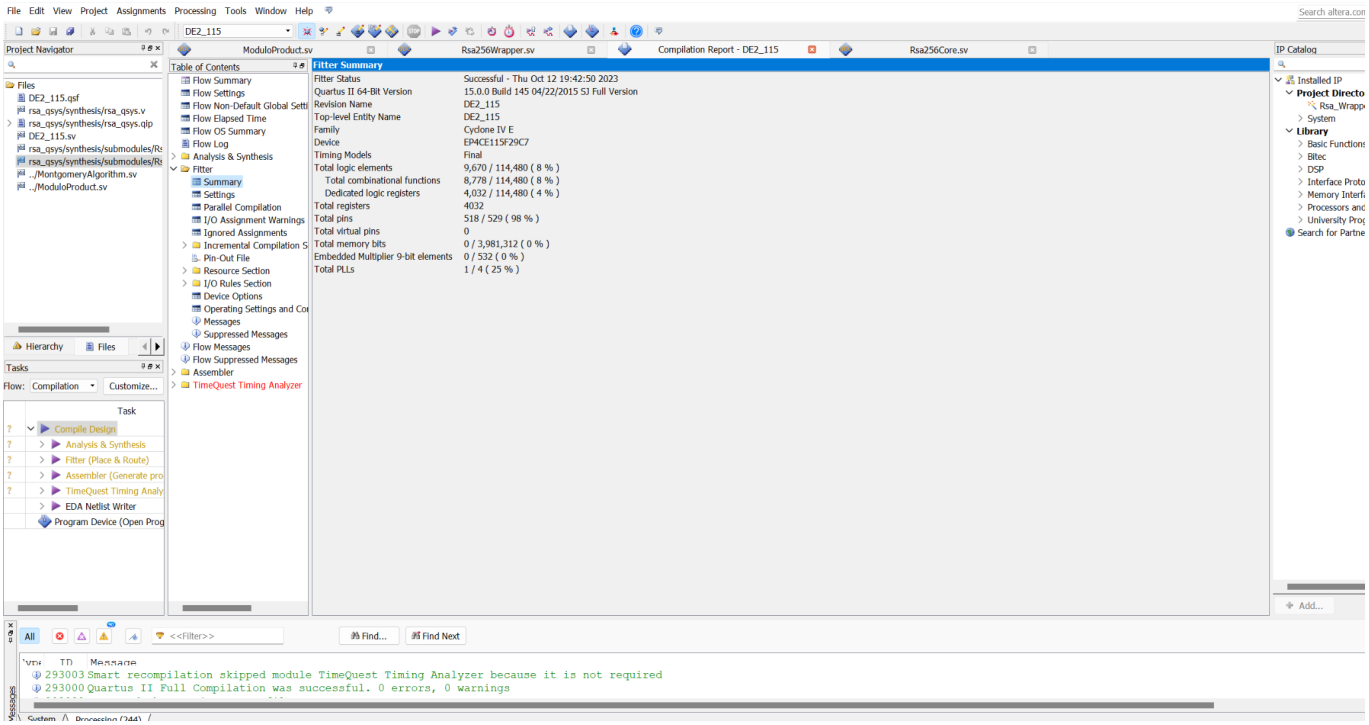


4. File Structure:

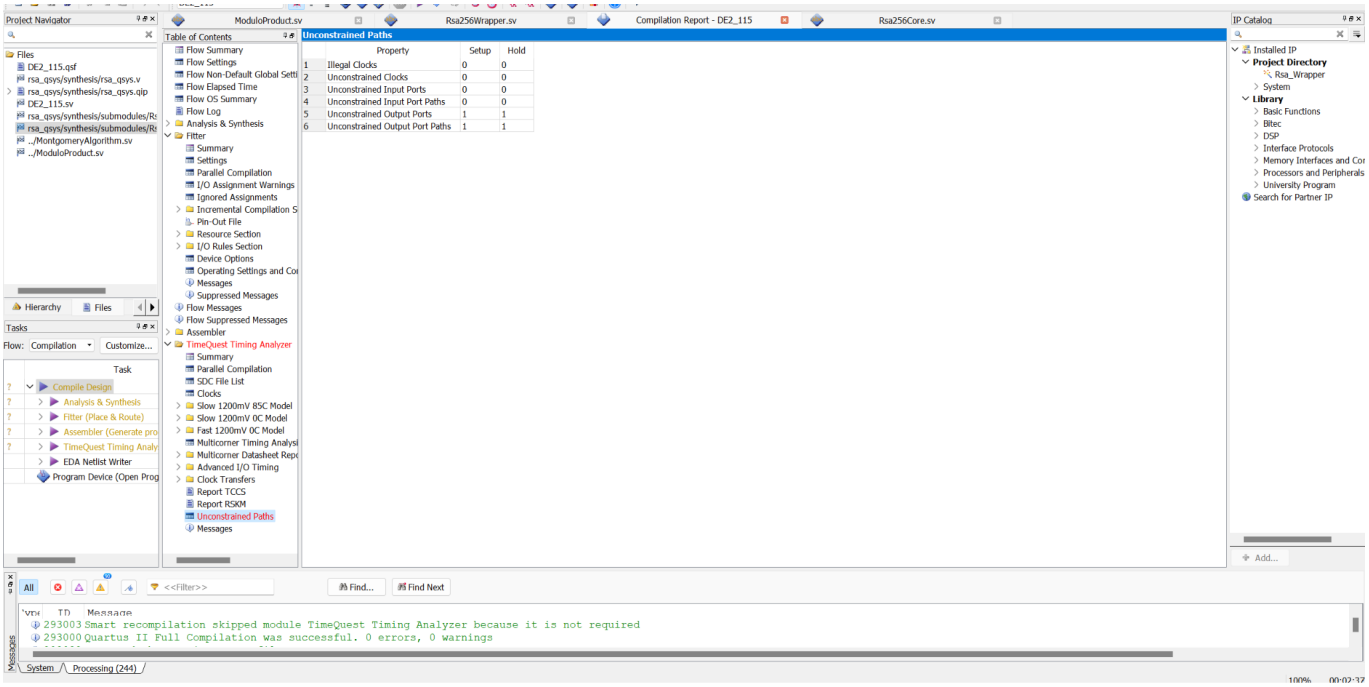
```
team02_lab2/
├── team02_lab2-report
└── src/
    ├── DE2_115/
    │   ├── DE2_115.qsf
    │   ├── DE2_115.sdc
    │   └── DE2_115.sv
    ├── MoudloProduct.sv
    ├── MontgomeryAlgorithm.sv
    ├── Rsa256Core.sv
    └── Rsa256Wrapper.sv
```

5. Screenshot:

(1) Fitter Summary



(2) Timing Analyzer



6. Problems Encountered and Solutions:

a. During the development of the Montgomery module, I overlooked the possibility of overflow during calculations, which caused continuous failures in the simulation of the RSA256 core.

b. The RS232 protocol for communication with the computer involves numerous details, such as reading addresses, determining RX and TX OK bits, and handling waitrequest, read, and write bits. These factors significantly impact the ability to read and transmit data to the PC. This debugging process highlighted the importance of reading protocol documentation thoroughly.

c. Throughout the debugging process, I often encountered issues such as getting stuck in a particular state or having one bit more or less in the output. I learned the importance of using tools like nWave to visualize waveforms, which allowed for pinpointing specific problems and addressing them effectively. Additionally, utilizing VCS simulation proved essential, saving

time by avoiding repeated uploads to the FPGA board. It also helped identify problems accurately, rather than relying on potentially misleading results from physical board operations.

d. While running on the FPGA board with Qsys, we faced challenges leading to incorrect decoding results. After extensive testing, we discovered that the issue was within the code itself, emphasizing that passing simulations does not guarantee flawless functionality on the FPGA. Furthermore, we learned that once Qsys is set up, it captures the source code, making it critical to ensure modifications are correctly recognized by Quartus, which took some time to resolve.

This experiment deepened our understanding of inter-device communication, protocol adherence, and efficient use of hardware features to simplify algorithms and accelerate computational speed, particularly in the implementation of the RSA256 algorithm.