

# **Verilog and FPGA Workshop**

## **Project 3 – RSA with ZYNQ**

**Name: ABABAKER NAZAR ABDELHAMID OSMAN**

**Matric No: A23KE0001**

**Link :**

[https://drive.google.com/drive/folders/1Wise3SZAqxaMWay2oY\\_0\\_SWD09AhqkLN?usp=sharing](https://drive.google.com/drive/folders/1Wise3SZAqxaMWay2oY_0_SWD09AhqkLN?usp=sharing)

**Brief , brief , brief Introduction:**

The system is designed to process inputs from two main sources: switches and a push button.

The push button plays a crucial role in controlling the encryption/decryption mode of the system, while the switches provide the data to be encrypted or decrypted.

The FPGA handles the inputs from the push button to toggle between three distinct states:

1. **State 0 (Do Nothing):** In this state, no operation is performed, and the system waits for further input or for the button to toggle to another state.
2. **State 1 (Encrypt):** When the button is pressed and toggled into this state, the FPGA triggers the RSA encryption operation. It uses the predefined RSA encryption exponent to encrypt the message received from the switches.
3. **State 2 (Decrypt):** When the button is pressed again, the FPGA switches to the decryption state, and the RSA decryption operation is triggered. The decryption is performed using the predefined RSA decryption exponent.

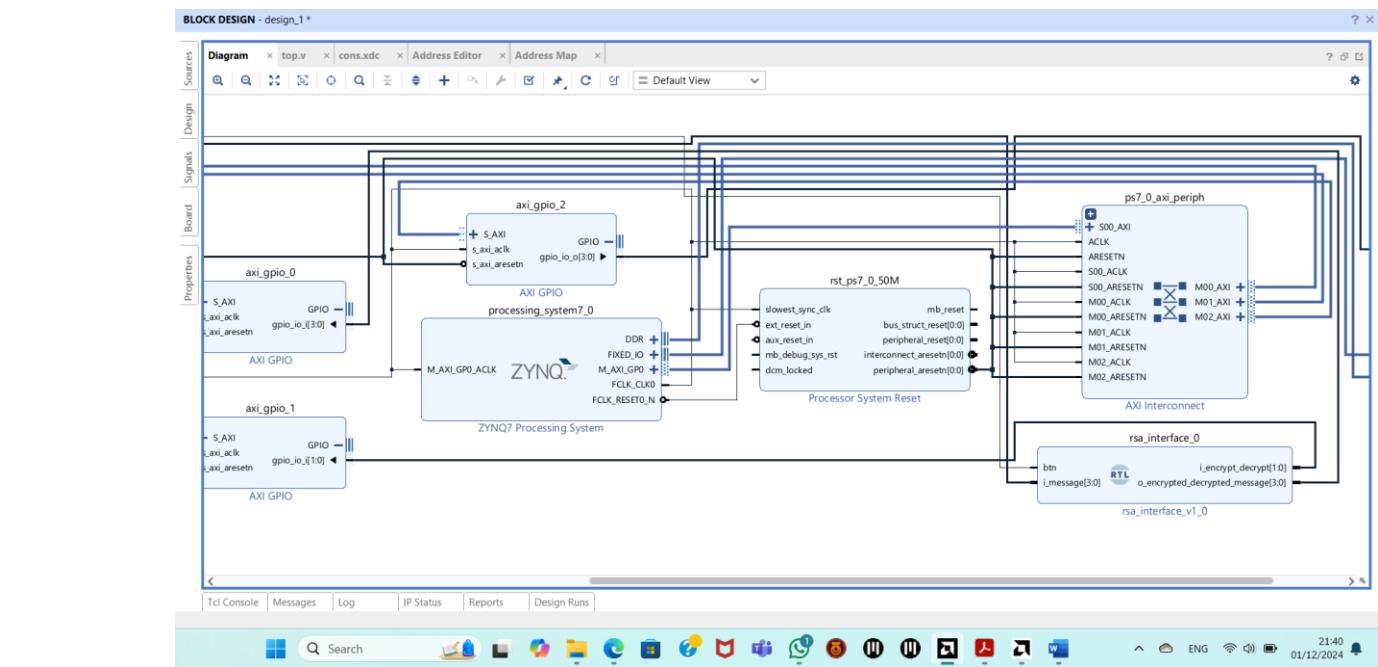
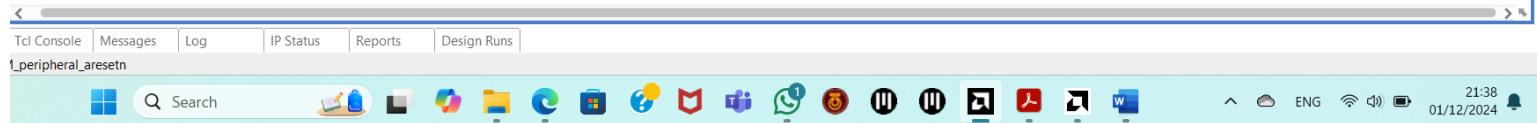
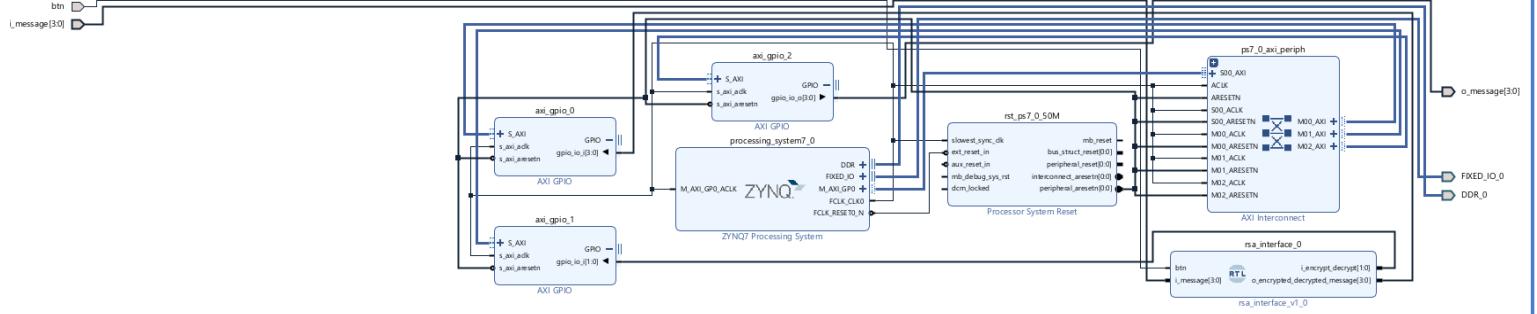
To implement this, the FPGA continuously monitors the push button input. It detects button presses and toggles between the three states (0, 1, and 2) as follows:

- Initially, the system starts in **State 0**, where no encryption or decryption takes place.
- A push button press increments the state to **State 1**, where the FPGA performs the encryption on the input message from the switches using modular exponentiation with the encryption exponent (7).
- Another push button press changes the state to **State 2**, and the FPGA performs decryption using the decryption exponent (3).
- After toggling through states 0, 1, and 2, the system resets back to **State 0**, ready to process further input.

The FPGA is responsible for detecting and handling these states. Once the appropriate mode (encryption or decryption) is selected, the FPGA sends the data to the Zynq processor, which performs the RSA cryptographic calculations. After completing the calculations, the Zynq processor sends the result back to the FPGA, which displays the outcome on the connected LEDs.

This design highlights how the FPGA interfaces with the push button to toggle between states, with each state corresponding to a different action: doing nothing, encrypting, or decrypting. This allows the user to control the system's operation based on simple input from the push button while leveraging the computational power of the Zynq processor for the RSA cryptography.

## System Block Diagram:



### Pure RSA C Code:

```
#include <stdio.h>

#define RSA_MODULO 15

#define RSA_EXPONENT_ENC 7

#define RSA_EXPONENT_DEC 3

int mod_exp(int base, int exponent, int mod) {

    int result = 1;

    base = base % mod;

    while (exponent > 0) {

        if (exponent % 2 == 1) {

            result = (result * base) % mod; }

        exponent = exponent >> 1;

        base = (base * base) % mod }

    return result; }

int main() {

    int i_message;

    int i_encrypt_decrypt = 0;

    printf("Enter a message (integer) to encrypt or decrypt:\n");

    scanf("%d", &i_message);
```

```
printf("Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit): ");

char user_input;

while (1) {

    scanf(" %c", &user_input); // Read the user's choice (with space to skip newline)

    if (user_input == '1') {

        int encrypted_message = mod_exp(i_message, RSA_EXPONENT_ENC,
RSA_MODULO);

        printf("Encrypted message: %d\n", encrypted_message);

    } else if (user_input == '2') {

        int decrypted_message = mod_exp(i_message, RSA_EXPONENT_DEC,
RSA_MODULO);

        printf("Decrypted message: %d\n", decrypted_message); } else {

        printf("Exiting...\n");

        break; }

    printf("Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit): ");}

return 0;}
```

## Results:

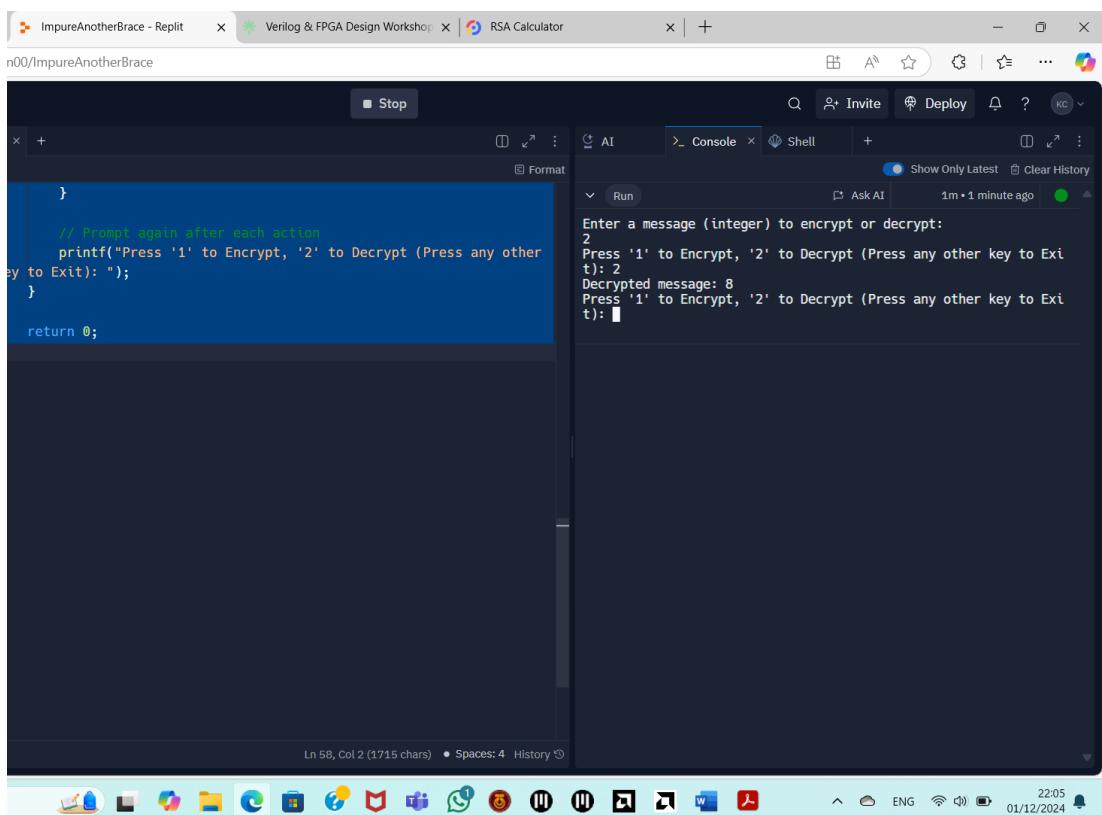
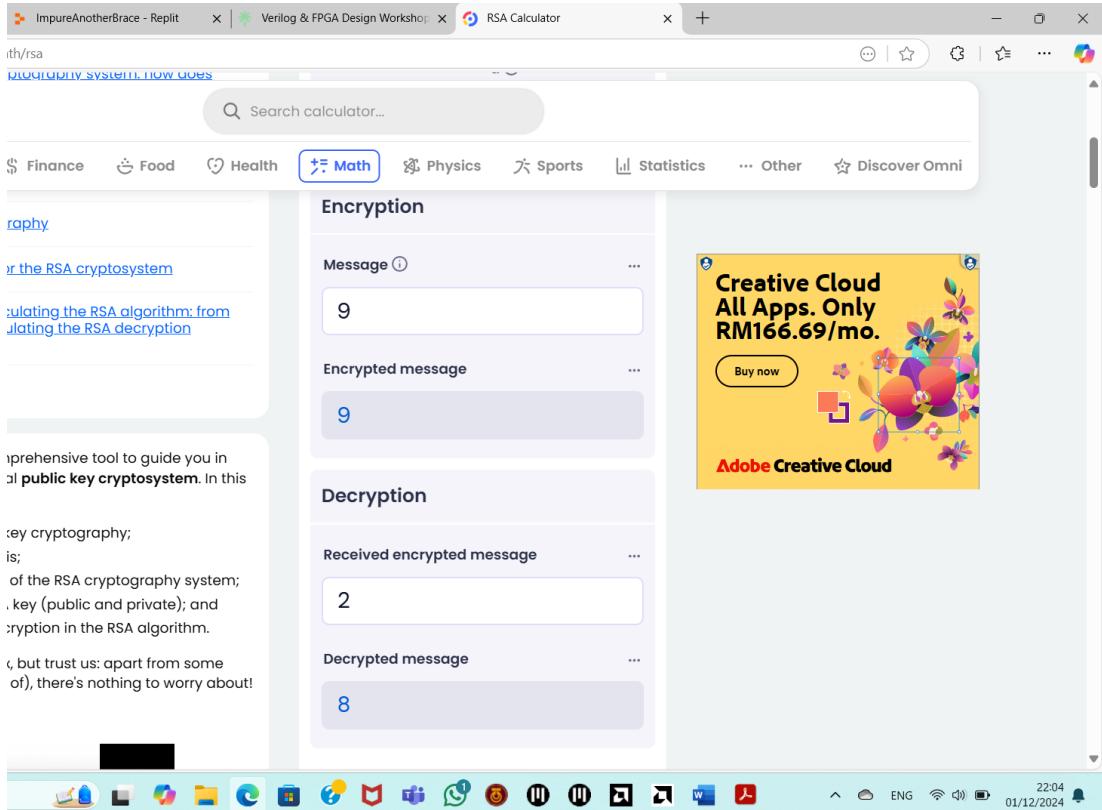
### 1) Encrypting message = 9

The screenshot shows the Omni calculator website's RSA algorithm page. On the left sidebar, there are links for RSA basics, calculating keys, encryption/decryption, weaknesses, and a complete example. The main area displays two input fields for modulus (N) and Euler's totient (λ(N)), both set to 15 and 4 respectively. Below these are fields for public exponent (e) set to 7 and private exponent (d) set to 3. Under the 'Encryption' section, a message field contains the value 9, and the encrypted message field also contains 9, indicating no encryption was applied.

The screenshot shows a code editor window with a dark theme. The main pane displays a C program named 'main.c'. The code defines RSA parameters (modulus 15, public exponent 7, private exponent 3), implements modular exponentiation using the 'mod\_exp' function, and provides a simple command-line interface for encryption and decryption. The 'Console' tab at the bottom shows the output of running the program with the message '9', choosing to encrypt ('1'), and displaying the encrypted message '9'.

```
1 // RSA parameters (modulus and exponents)
2 #define RSA_MODULO 15
3 #define RSA_EXPONENT_ENC 7
4 #define RSA_EXPONENT_DEC 3
5
6 // Function to calculate modular exponentiation (used for RSA
7 // encryption and decryption)
8 int mod_exp(int base, int exponent, int mod) {
9     int result = 1;
10    base = base % mod;
11
12    while (exponent > 0) {
13        if (exponent % 2 == 1) {
14            result = (result * base) % mod;
15        }
16        exponent = exponent >> 1;
17        base = (base * base) % mod;
18    }
19
20    return result;
21 }
22
23
24 int main() {
25     int i_message;
26     int i_encrypt_decrypt = 0; // Start with no encryption or
decryption
27
28     // Your code here
29 }
```

## 2) Decrypting message = 2



### 3) Encrypting message = 3

```
ibabakersman00/ImpureAnotherBrace
main.c
51 }
52
53     // Prompt again after each action
54     printf("Press '1' to Encrypt, '2' to Decrypt (Press any other
55     key to Exit): ");
56
57     return 0;
58 }
```

Enter a message (integer) to encrypt or decrypt:  
3  
Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit): 1  
Encrypted message: 12  
Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit):

https://www.omnicalculator.com/math/rsa

How to use our calculator for the RSA cryptosystem

A complete example of calculating the RSA algorithm: from generating the keys to calculating the RSA decryption process

FAQs

Our RSA calculator is a comprehensive tool to guide you in discovering the fundamental public key cryptosystem. In this article, you will learn:

- The basis of distributed key cryptography;
- What the RSA algorithm is;
- The operating principles of the RSA cryptography system;
- How to generate the RSA key (public and private); and
- How to calculate the decryption in the RSA algorithm.

The topic may look complex, but trust us: apart from some math (which we'll take care of), there's nothing to worry about! Let's go!

15      4

e ⓘ      ...      d ⓘ      ...

7      3

Encryption

Message ⓘ

3

Encrypted message

12

Decryption

## RSA C Code in ZYNQ:

The screenshot shows the Vitis IDE interface with the following details:

- File Explorer:** Shows a project structure with files like `c_rsa`, `Vitis...`, `clangd`, `Git`, `GitLens`, `HLS P...`, and `rsa`.
- Code Editor:** Displays the `rsa_c_code.c` file containing RSA-related code.
- Output Console:** Shows the build log:

```
[01/12/2024, 20:07:12]: Build for c_rsa:: with id 'd09a3924-405e-4cb2-b434-018f2a181c4b' started.  
-- Configuring done  
-- Generating done  
-- Build files have been written to: C:/ABABAKER/.wsdata/c_rsa/build  
[1/2] C:/Xilinx/Vitis/2024.1/gnu/aarch32/nt/gcc-arm-none-eabi/bin/arm-none-eabi-gcc.exe -isystem C:/ABABAKER/  
text      data     bss     dec     hex filename  
20729      1148   22600   44477    abdb c_rsa.elf  
Build Finished successfully  
[01/12/2024, 20:07:13]: Build for c_rsa:: with id 'd09a3924-405e-4cb2-b434-018f2a181c4b' ended.
```
- Bottom Status Bar:** Shows the date (01/12/2024), time (21:26), and battery level.

Figure: Build Successful in Vitis

```
#include <stdio.h>  
  
#include "xparameters.h" // Include Xilinx parameters  
  
#include "xgpio.h"  
  
#define LED_CHANNEL 1 // Channel for LEDs  
  
#define BTN_CHANNEL 1 // Channel for Buttons  
  
#define RSA_MODULO 15  
  
#define RSA_EXPONENT_ENC 7  
  
#define RSA_EXPONENT_DEC 3
```

```
XGpio gpio; // GPIO instance for interacting with push button and LEDs

// Function to calculate modular exponentiation (used for RSA encryption and decryption)

int mod_exp(int base, int exponent, int mod) {

    int result = 1;

    base = base % mod;

    while (exponent > 0) {

        if (exponent % 2 == 1) {

            result = (result * base) % mod;

        }

        exponent = exponent >> 1;

        base = (base * base) % mod; }

    return result; }

int main() {

    int i_message;

    int i_encrypt_decrypt = 0; // Start with no encryption or decryption

    // Initialize GPIO for input (push button) and output (LEDs)

    XGpio_SetDataDirection(&gpio, BTN_CHANNEL, 0xFF); // Input for buttons

    XGpio_SetDataDirection(&gpio, LED_CHANNEL, 0x00); // Output for LEDs
```

```

while (1) {

    // Read the message from the input (from switches or another source)

    i_message = XGpio_DiscreteRead(&gpio, BTN_CHANNEL); // Example: read from
buttons

    // Read the button input for toggle state

    int btn_state = XGpio_DiscreteRead(&gpio, BTN_CHANNEL);

    // Check button press and toggle encryption/decryption flag (simplified logic)

    if (btn_state == 1) {

        i_encrypt_decrypt++;

        if (i_encrypt_decrypt > 2) {

            i_encrypt_decrypt = 0; // Reset state after 3 button presses

        }

    }

    // Perform encryption or decryption based on the flag

    if (i_encrypt_decrypt == 1) {

        // Encryption

        int encrypted_message = mod_exp(i_message, RSA_EXPONENT_ENC,
RSA_MODULO);

        XGpio_DiscreteWrite(&gpio, LED_CHANNEL, encrypted_message); // Display
encrypted message on LEDs
    }
}

```

```
    } else if (i_encrypt_decrypt == 2) {  
  
        // Decryption  
  
        int decrypted_message = mod_exp(i_message, RSA_EXPONENT_DEC,  
RSA_MODULO);  
  
        XGpio_DiscreteWrite(&gpio, LED_CHANNEL, decrypted_message); // Display  
decrypted message on LEDs  
  
    }  
  
    return 0;  
  
}
```