

Verilog and FPGA Workshop

Project 3 – RSA with ZYNQ

Name: ABABAKER NAZAR ABDELHAMID OSMAN

Matric No: A23KE0001

Link :

https://drive.google.com/drive/folders/1WISE3SZAgxaMWay2oY_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.

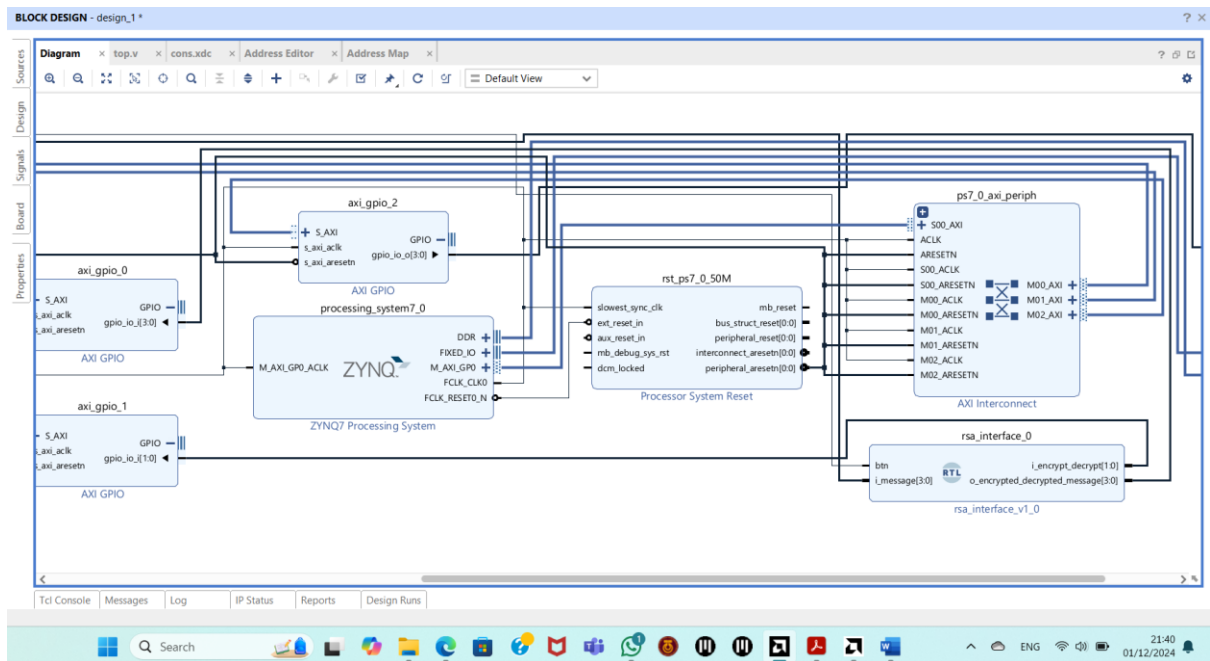
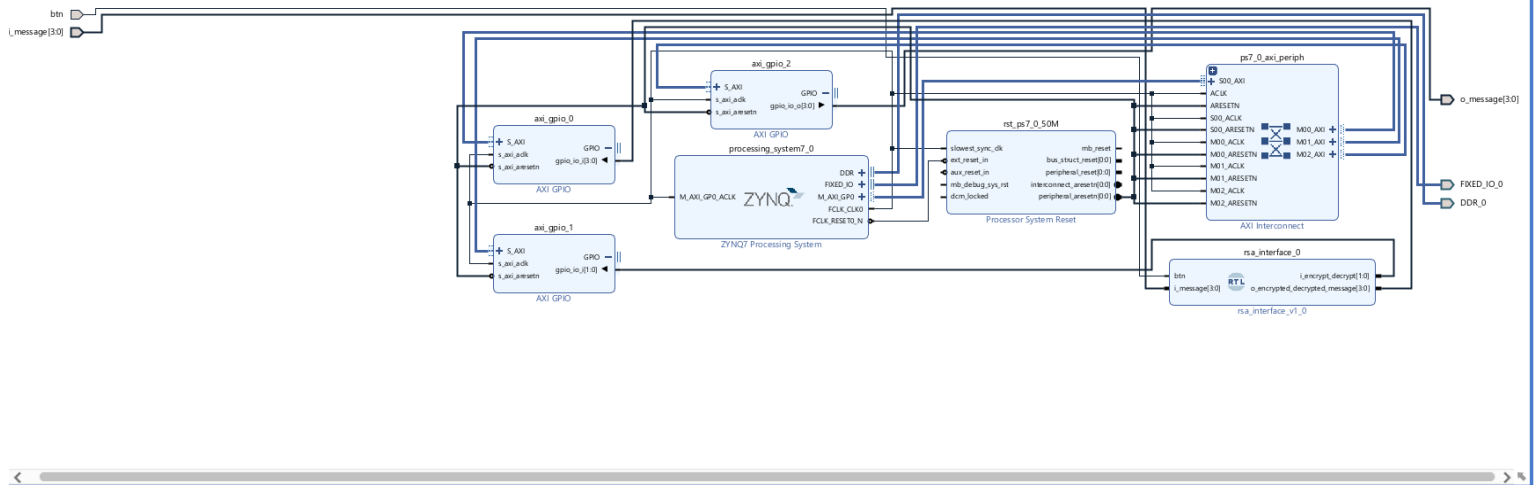
To implement this, the FPGA continuously monitors the push button input. It detects button presses and toggles between the three states (0, 1, and 2) as follows:

- 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 encryption tool. The URL is <https://www.omnicalculator.com/math/rsa>. The interface includes a search bar, navigation tabs (Board, E, day life, Finance, Food, Health, Math, Physics, Sports, Statistics, Other, Discover Omni), and a sidebar with links to RSA-related topics. The main area displays the encryption process with the following parameters:

- N : 15
- $\lambda(N)$: 4
- e : 7
- d : 3

The "Encryption" section shows the "Message" as 9, and the "Encrypted message" is also 9. A sidebar on the right features a promotional banner for "RM10 off syok meals".

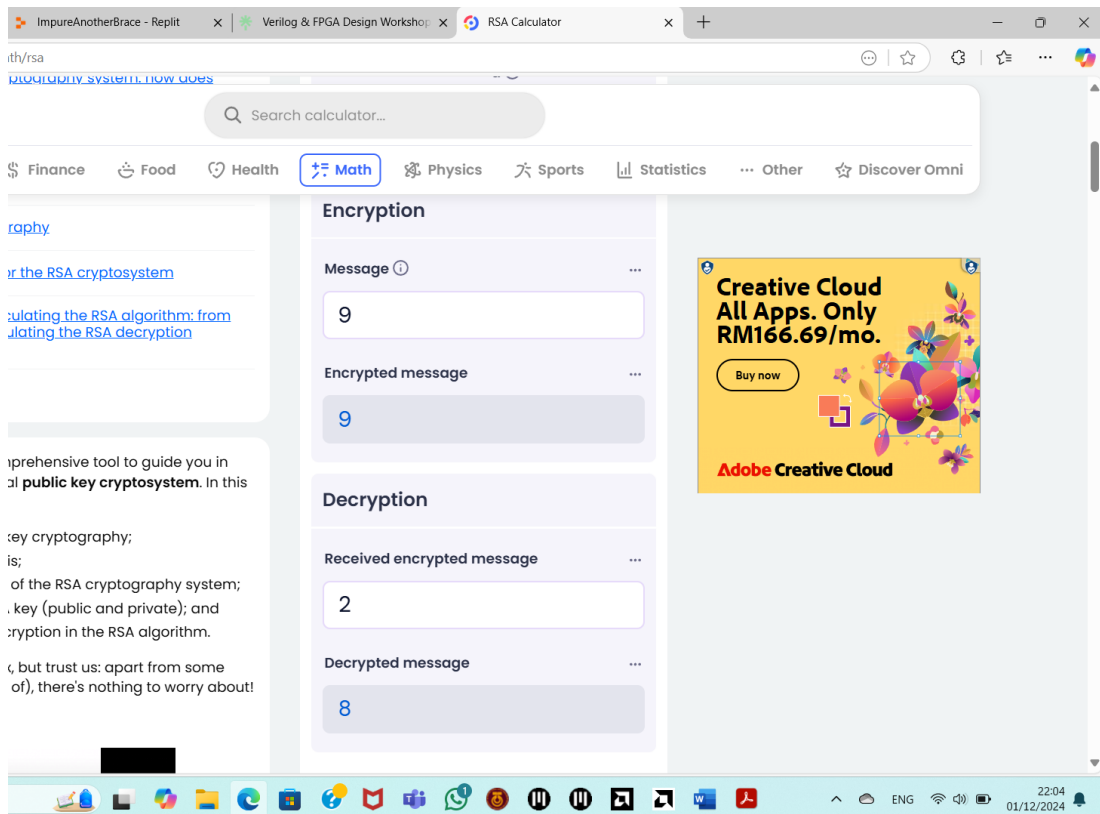
The screenshot shows a C program implementing RSA encryption. The code is written in a dark-themed editor with a console output on the right. The program defines RSA parameters and a modular exponentiation function.

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

The console output shows the program's execution:

```
Enter a message (integer) to encrypt or decrypt:
9
Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit): 1
Encrypted message: 9
Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit):
```

2) Decrypting message = 2



The screenshot shows the RSA Calculator website. The 'Math' tab is selected. The 'Encryption' section shows a message of '9' being encrypted to '9'. The 'Decryption' section shows a received encrypted message of '2' being decrypted to '8'. An advertisement for Creative Cloud is visible on the right side of the page.

Search calculator...

Finance Food Health **Math** Physics Sports Statistics Other Discover Omni

Encryption

Message ① ...

9

Encrypted message ...

9

Decryption

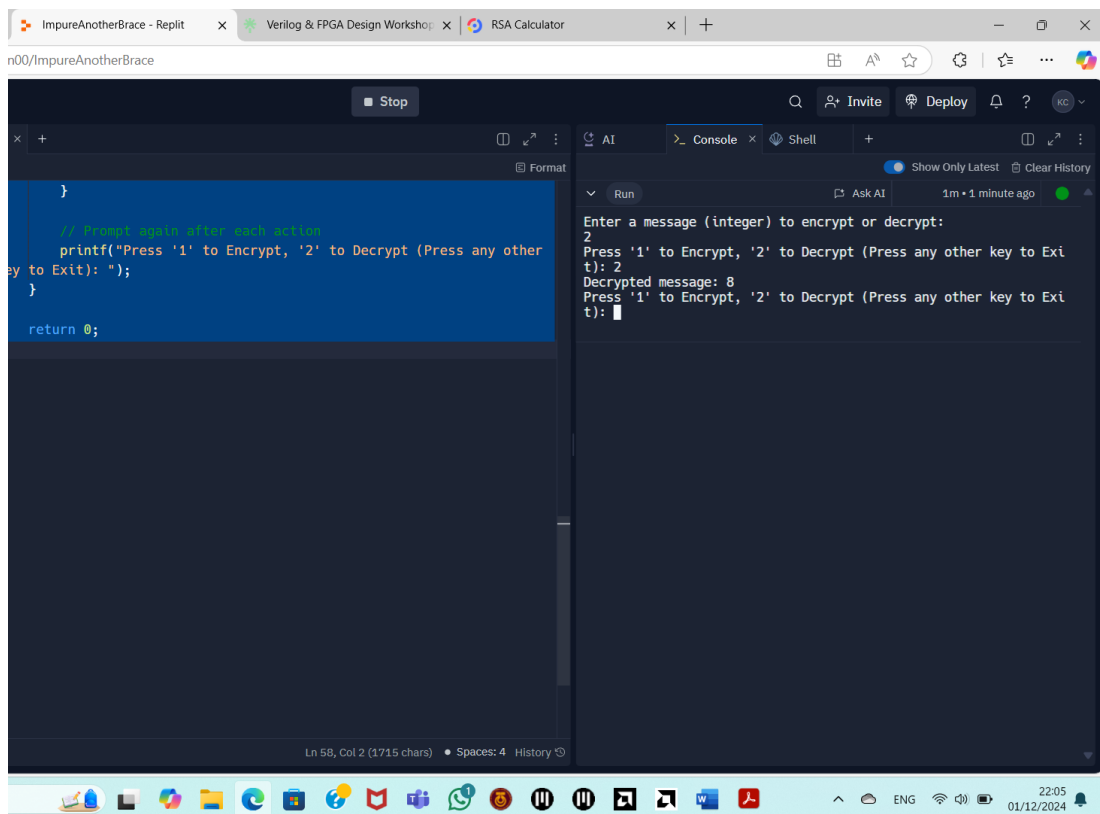
Received encrypted message ...

2

Decrypted message ...

8

Creative Cloud
All Apps. Only
RM166.69/mo.
Buy now
Adobe Creative Cloud



The screenshot shows a Replit IDE with a C program for RSA encryption and decryption. The code prompts the user to enter a message (integer) to encrypt or decrypt. The user enters '2', and the program outputs 'Decrypted message: 8'. The code is as follows:

```
1 // Prompt again after each action
2 printf("Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit): ");
3 }
4 return 0;
```

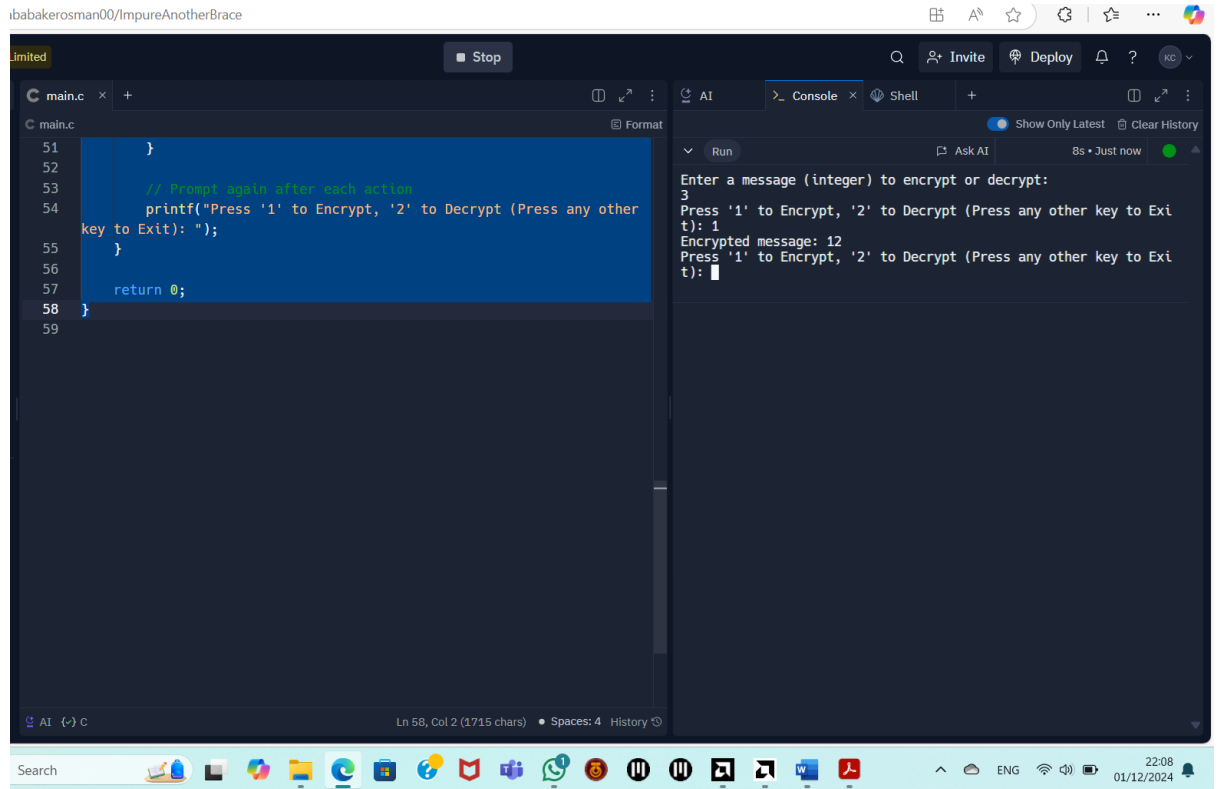
Run Console Shell

Show Only Latest Clear History

Enter a message (integer) to encrypt or decrypt:
2
Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit): 2
Decrypted message: 8
Press '1' to Encrypt, '2' to Decrypt (Press any other key to Exit):

Ln 58, Col 2 (1715 chars) • Spaces: 4 History

3) Encrypting message = 3

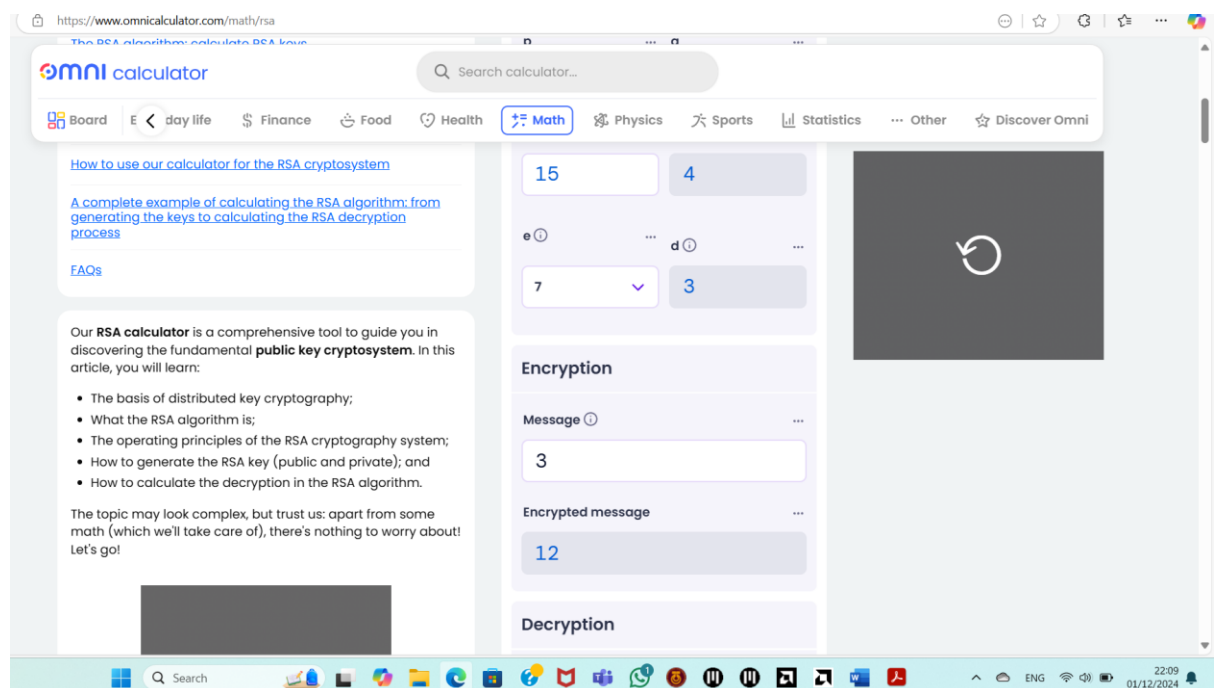


The screenshot shows a C program in a code editor. The program prompts the user to enter a message (integer) to encrypt or decrypt. The user enters 3. The program then prompts the user to press '1' to Encrypt, '2' to Decrypt, or any other key to Exit. The user presses '1'. The program then displays the encrypted message: 12.

```
51 }
52 // Prompt again after each action
53 printf("Press '1' to Encrypt, '2' to Decrypt (Press any other
54 key to Exit): ");
55 }
56 return 0;
57 }
58 }
59 }
```

Console output:

```
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):
```



The screenshot shows the Omni Calculator website. The page is titled "The RSA algorithm: calculate RSA keys". It contains a section titled "How to use our calculator for the RSA cryptosystem" and a section titled "Our RSA calculator is a comprehensive tool to guide you in discovering the fundamental public key cryptosystem. In this article, you will learn:".

The RSA calculator interface is shown on the right. It has input fields for the message (3) and the encrypted message (12). The calculator also has a section for the RSA key (e, d) with values 7 and 3. The calculator is currently set to "Encryption" mode.

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!

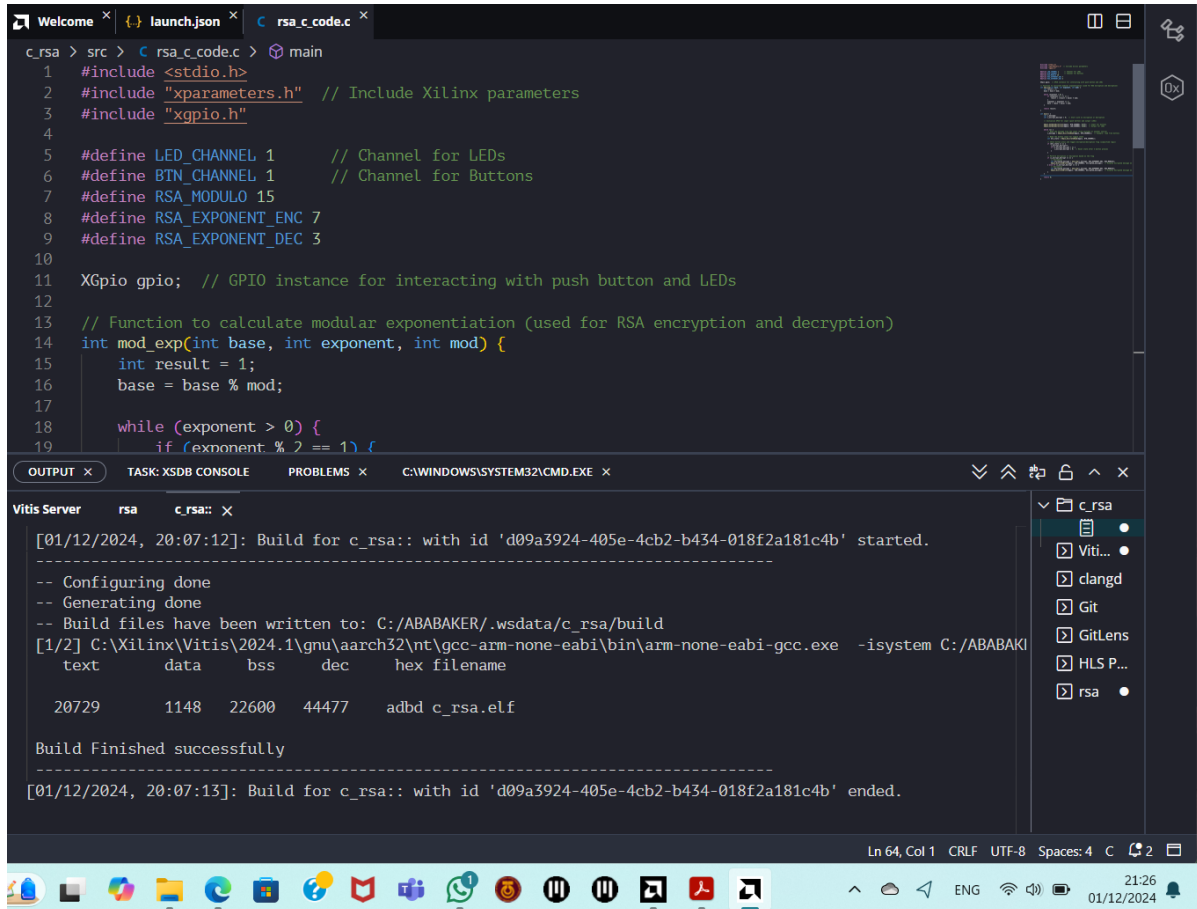
Encryption

Message: 3

Encrypted message: 12

Decryption

RSA C Code in ZYNQ:



```
c_rsa > src > c_rsa_code.c > main
1  #include <stdio.h>
2  #include "xparameters.h" // Include Xilinx parameters
3  #include "xgpio.h"
4
5  #define LED_CHANNEL 1      // Channel for LEDs
6  #define BTN_CHANNEL 1     // Channel for Buttons
7  #define RSA_MODULO 15
8  #define RSA_EXPONENT_ENC 7
9  #define RSA_EXPONENT_DEC 3
10
11 XGpio gpio; // GPIO instance for interacting with push button and LEDs
12
13 // Function to calculate modular exponentiation (used for RSA encryption and decryption)
14 int mod_exp(int base, int exponent, int mod) {
15     int result = 1;
16     base = base % mod;
17
18     while (exponent > 0) {
19         if (exponent % 2 == 1) {
```

Vitis Server **rsa** **c_rsa::** X

```
[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\aaarch32\nt\gcc-arm-none-eabi\bin\arm-none-eabi-gcc.exe -isystem C:/ABABAKER/...
text      data      bss      dec      hex filename
20729     1148     22600   44477   adbd c_rsa.elf

Build Finished successfully
-----
[01/12/2024, 20:07:13]: Build for c_rsa:: with id 'd09a3924-405e-4cb2-b434-018f2a181c4b' ended.
```

Ln 64, Col 1 CRLF UTF-8 Spaces: 4 C 21:26 01/12/2024

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;

}

```