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.

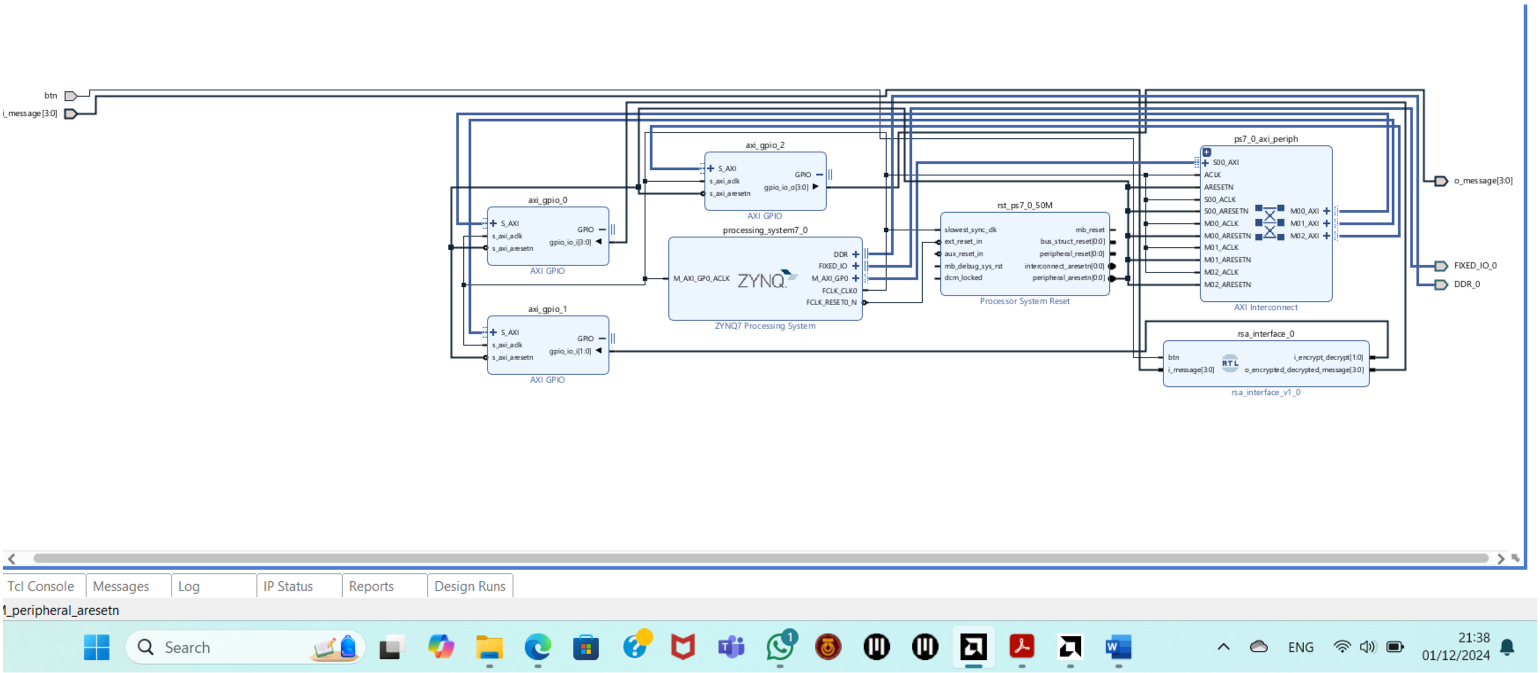
To implement this, the FPGA continuously monitors the push button input. It detects button presses and toggles between the three states (0, 1, and 2) as follows:

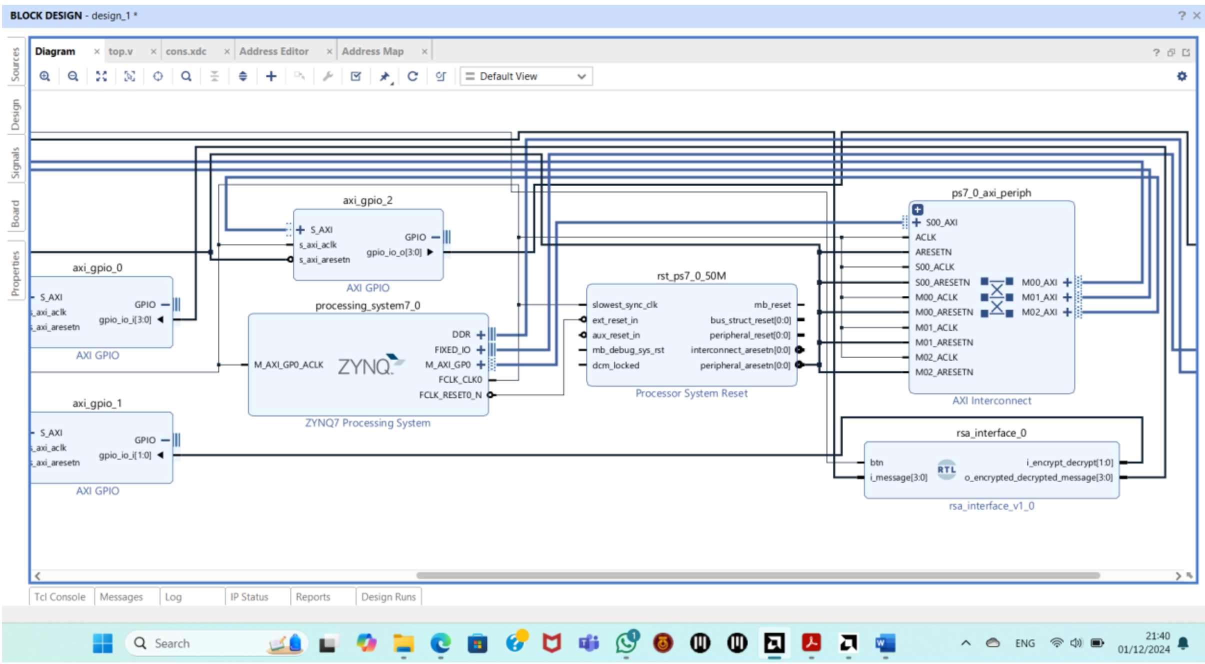
- Initially, the system starts in **State 0**, where no encryption or decryption takes place.
- A push button press increments the state to **State 1**, where the FPGA performs the encryption on the input message from the switches using modular exponentiation with the encryption exponent (7).
- Another 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:





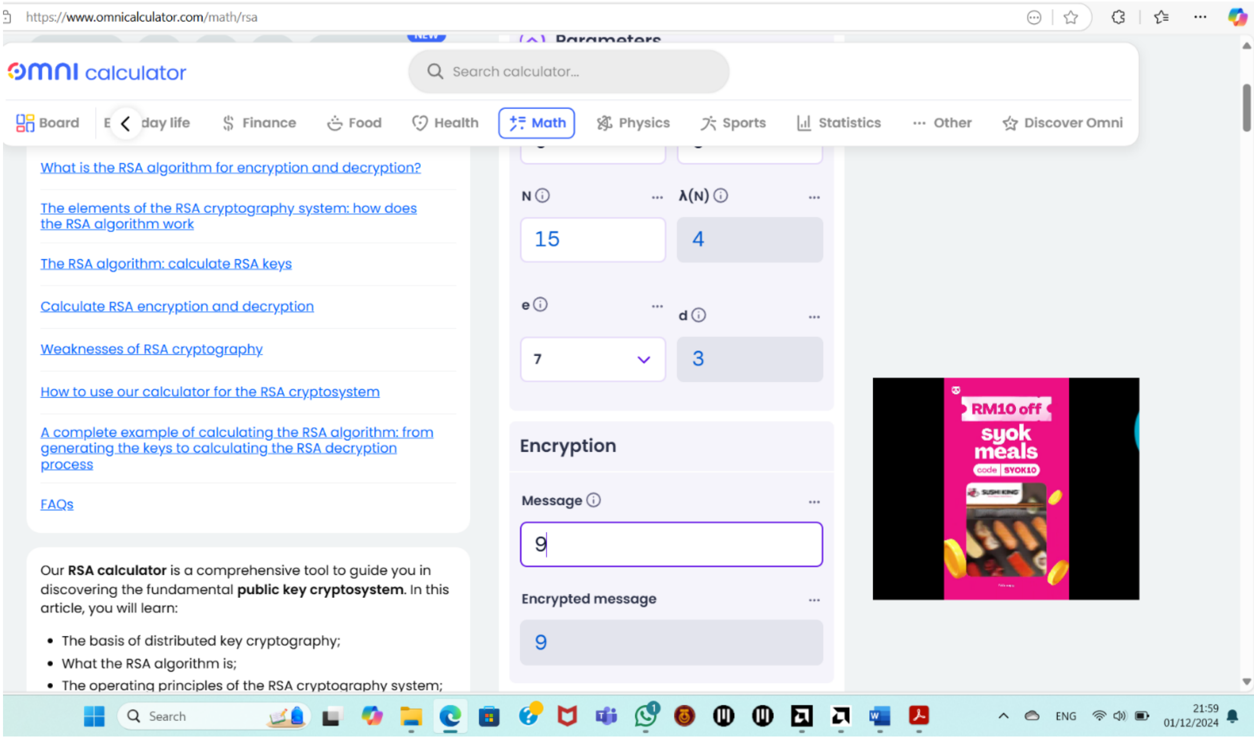
Pure RSA C Code:

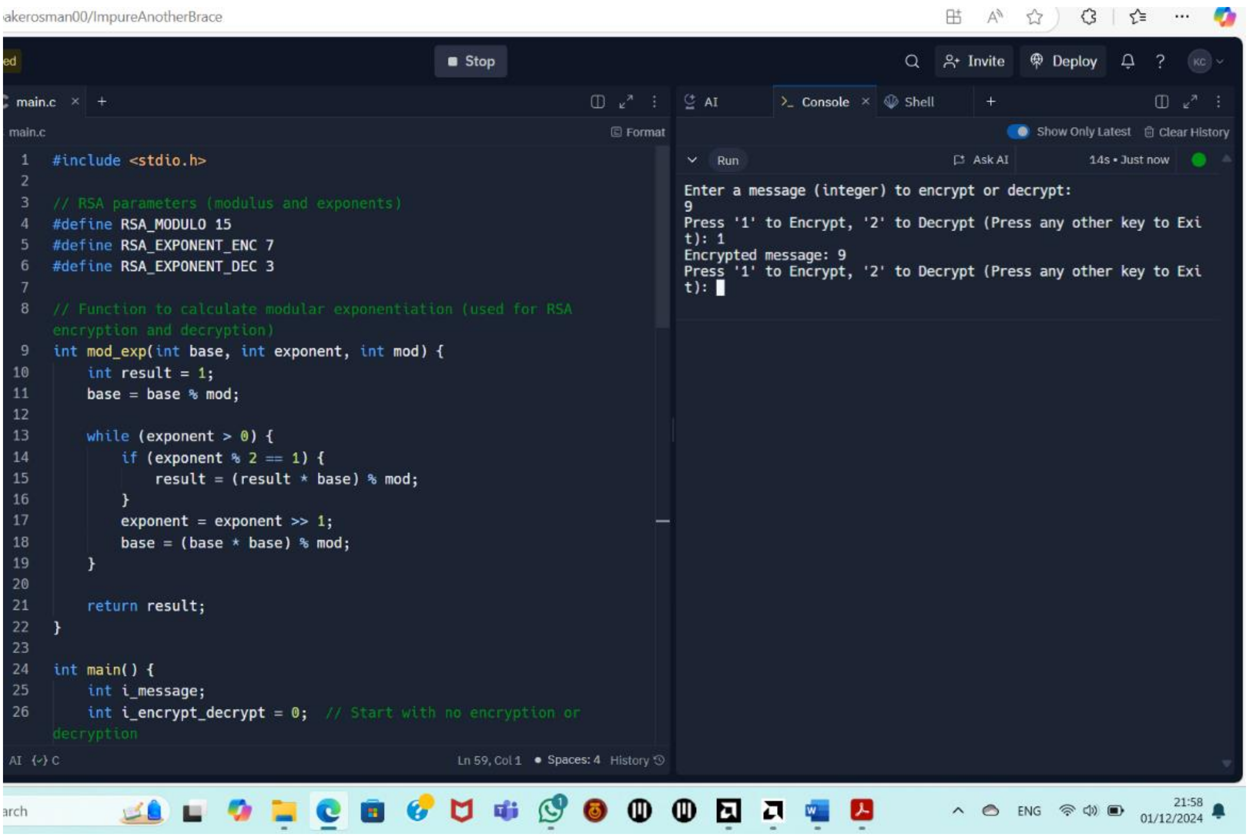
```
#include <stdio.h>



#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;}
```

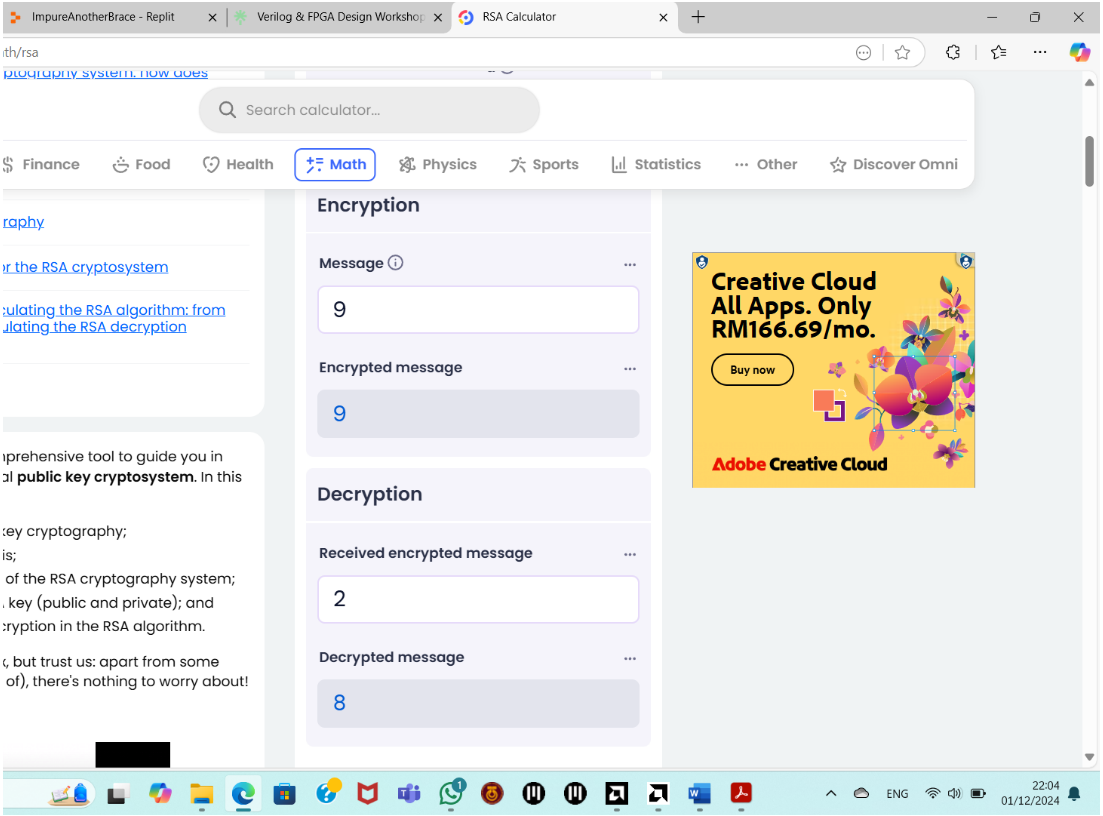
Results:

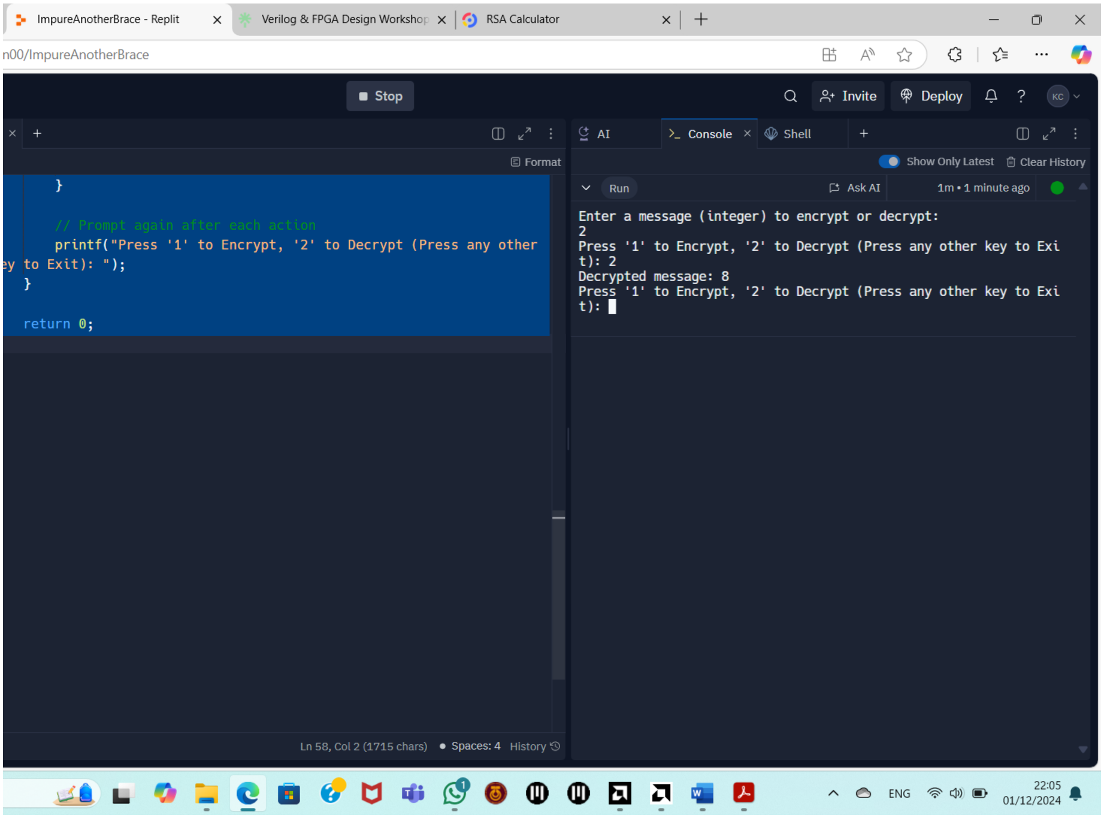
1) Encrypting message = 9



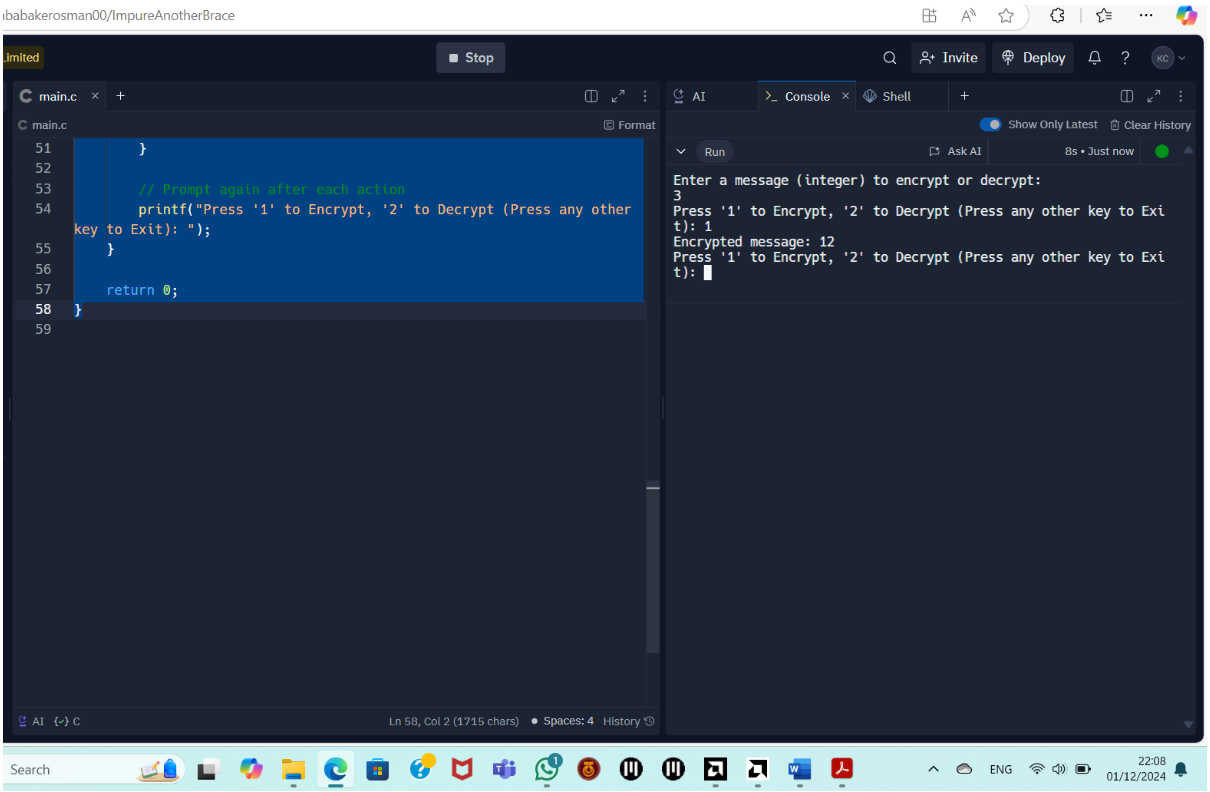


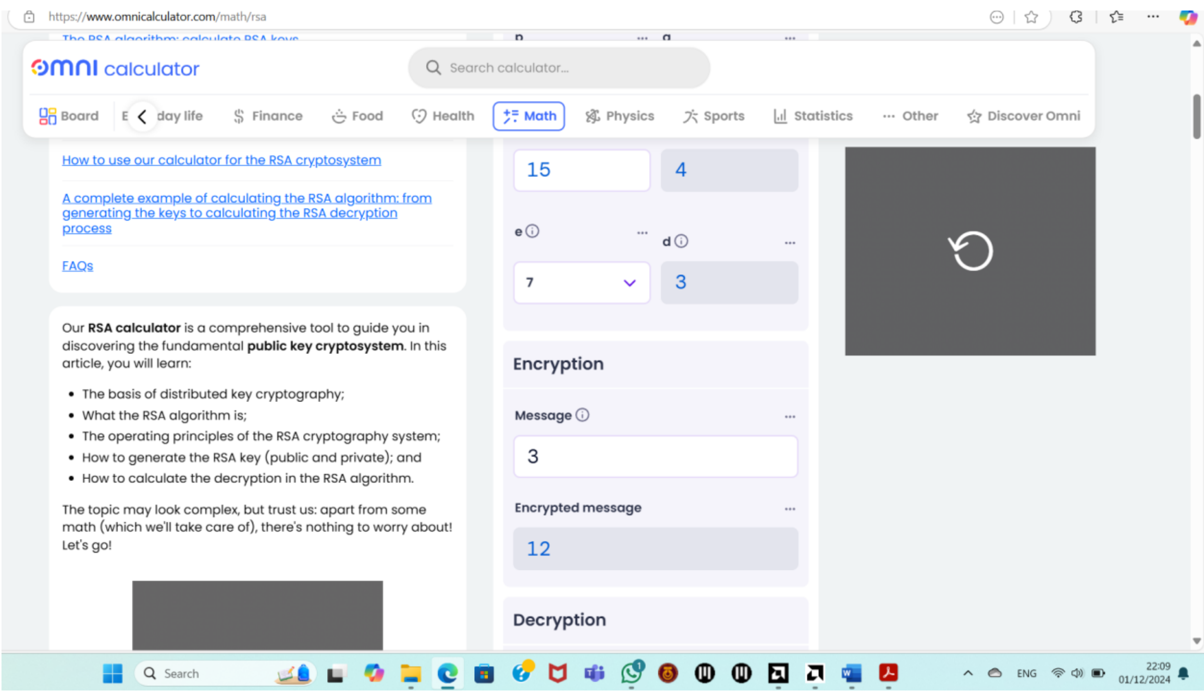
2) Decrypting message = 2





3) Encrypting message = 3





RSA C Code in ZYNQ:

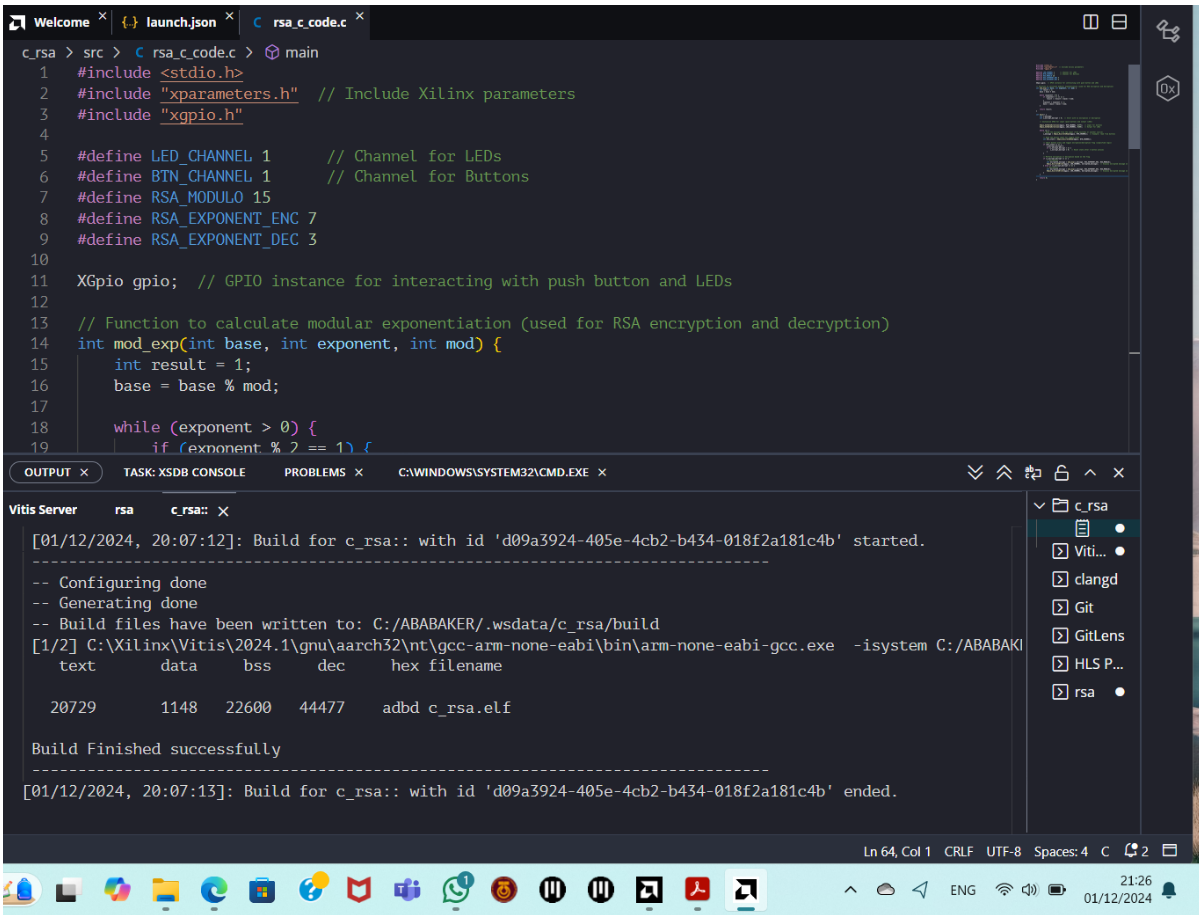


Figure: Build Successful in Vitis

```
#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;
}
```