

Integrated Circuit Design

Homework 2

Due: 14:20, April 10, 2025

*No late homework is allowed.

1 RSA Background

In this homework, you need to implement a design to encrypt and decrypt messages using given encryption and decryption keys. In RSA, k_e is the encryption key, and k_d is the decryption key. N is the product of two large, randomly chosen prime numbers p and q .

Let m be the message we want to send. The encryption and decryption processes are $m^{k_e} \bmod N$ and $m^{k_d} \bmod N$, respectively, where k_e and k_d satisfy $k_e k_d \bmod (p-1)(q-1) = 1$.

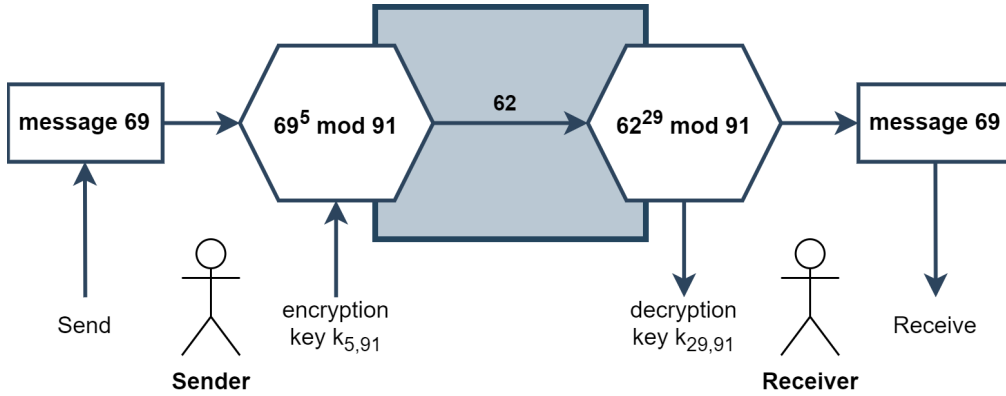


Figure 1: Encryption and decryption using RSA cryptography.

Using Figure 1 as an example, we let $p = 7$ and $q = 13$. We then calculate $N = 7 \times 13 = 91$ and $(p-1)(q-1) = 72$. We next select k_e relatively prime to 72 and < 72 ; Here, we let k_e be 5. Moreover, we calculate k_d such that $k_e k_d \bmod 72 = 1$, yielding 29. At last we have the encryption key, $k_e = 5$, and the decryption key, $k_d = 29$, for $N = 91$. Therefore, the sender can send message 69 by encrypting it and obtain message 62, and the receiver can decrypt it using the decryption key.

2 Problem Description

In this homework, you will receive encrypted instructions from the testbench. You need to decrypt the instruction {OPCODE, IMME} and perform the corresponding operation, where the OPCODE and IMME have 6 and 10 bits, respectively. The instruction set is shown below. Based on the decoded operation, you need to update the stored register.

Instruction	OPCODE	Description
Reset	000001	Register = 0
Set	000010	Register = Next Decrypted Message (See Figure 2 for example)
Add	000100	Register = Register + IMME
Sub	001000	Register = Register - IMME
Set Encryption Key	010000	$k_e = \text{IMME}[7:0]$
Encrypt Message	100000	Encrypt the data in register and output

Note: The register is a 2's complement number with 12 bits, while IMME is a positive number. If overflow occurs, you need to set the value to the maximum positive number or minimum negative number. Assume your decryption key is $k_d = 11$, and we use $N = 52961$ for encrypt and decrypt.

Input	Decrypted	OPCODE	IMME	Description
2	800	000010	don't care	Set register to the next decrypted message
b6e8	476	don't care	don't care	Register = 1142 (16'h476)
9561	20cd	001000	00_1100_1101	Register = Register - IMME (00_1100_1101)
9bd2	22ed	001000	10_1110_1101	Register = Register - IMME (10_1110_1101)
319	40e5	010000	00_1110_0101	ke = IMME[7:0] (00_1110_0101)
c2ac	8000	100000	don't care	Encrypt register and output

Figure 2: An example of input patterns. The first encrypted message, which is 2 in the second line, will tell you that the register needs to be set to the next decrypted message, which is 1142 in the third line.

3 I/O Specification

Signals	I/O	Bit Width	Description
clk	I	1	Positive edge-trigger clock
rst_n	I	1	Asynchronous negative-edge reset
i_valid	I	1	High when the input data is valid
ack	O	1	High if you acknowledge the input
Mi	I	16	Input message
o_valid	O	1	High when the output data is valid
Mo	O	16	Output message

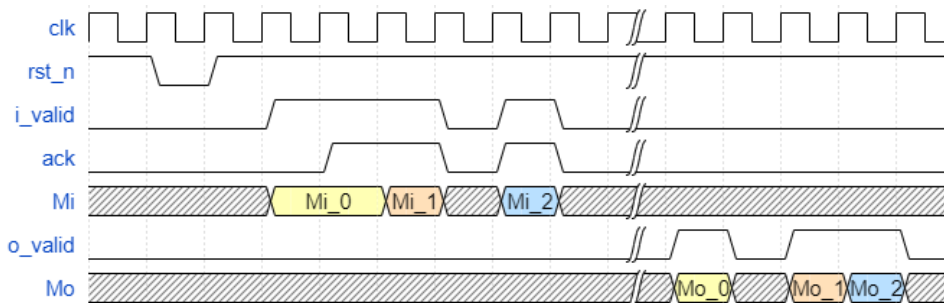


Figure 3: The waveforms of I/O signals.

4 Files

Figure 4 shows the files you will be able to download from NTU Cool.

- **tb_RSA.v**: The testbench for this homework.
- **Mi_pattern*.dat**: The pattern of input Mi.
- **Mo_pattern*.dat**: The golden pattern of output Mo.
- **RSA.v**: The design you need to work with.

```

ICD_hw2
├── 00_TESTBENCH
│   ├── tb_RSA.v
│   └── pattern
│       ├── Mi_pattern*.dat
│       └── Mo_pattern*.dat
└── 01_RTL
    └── RSA.v

```

Figure 4: The files you will get.

5 Simulation

- Use the following code to run Verilog simulations:
`$ vcs ./tb_RSA.v ./RSA.v -full64 -R -debug_access+all +define+pat0`
- Change `pat0` to `pat1`, `pat2`, ... to test other public patterns.

6 Check latch

In your design, all sequential elements must be flip-flops. Use **Design Compiler** to check if there's any latch in your design. The commands are as follows:

```

$ dc_shell
dc_shell > read_verilog RSA.v

```

The output should look like Figure 5.

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
cnt_r_reg	Flip-flop	8	Y	N	Y	N	N	N	N
state_r_reg	Flip-flop	2	Y	N	Y	N	N	N	N
set_message_r_reg	Flip-flop	1	N	N	Y	N	N	N	N
register_r_reg	Flip-flop	12	Y	N	Y	N	N	N	N
ke_r_reg	Flip-flop	8	Y	N	Y	N	N	N	N
Mi_r_reg	Flip-flop	16	Y	N	Y	N	N	N	N
mod_r_reg	Flip-flop	16	Y	N	Y	N	N	N	N

Figure 5: An example of Design Compiler's output.

7 Rules

7.1 What you can do

- Explain a concept to others
- Discuss possible algorithms with others
- Search online for algorithms

7.2 What you cannot do

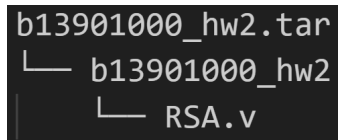
- Use brute force to solve the problem (directly output the golden pattern)
- Use GPT to generate your code
- Share code with others
- Copy or read others' code
- Copy or read online code
- Help your classmate to debug his/her code

8 Grading

- Each pattern is worth 20 points. There are four public and one hidden patterns.
- You will not receive any points if your design has any latch.

9 Submission

- Upload <student_id>_hw2.tar to NTU COOL
 - Wrong file or folder format will get -10 points as a penalty
 - Compress your files with the following instruction:
`$ tar -cvf b13901000_hw2.tar b13901000_hw2/`



```
b13901000_hw2.tar
└─ b13901000_hw2
   └─ RSA.v
```

Figure 6: An example of the file/folder format.

TA

H.-Y. Tseng	r12943119@ntu.edu.tw
P.-S. Yen	r12943009@ntu.edu.tw