Lab 6: Simon Cipher Encryption

EEL 4712 – Fall 2019

Objective:

The objective of this lab is to study the implementation of lightweight cryptographic ciphers using datapath and finite state machine. You will also learn how to instantiate and use memory components.

Required tools and parts:

QuartusII software package, ALTERA DE10-lite board.

IP Used:

An altsyncram IP component will be used in this lab. Also, a memory initialization file (in.mif, key.mif) will be used to initialize the memory values of the ROM and RAM.

Discussion:

Encryption is the process of using an algorithm (*E*) to encode a message (*P*) between two parties using a key (*K*). The output (*C*) should be undecipherable unless the secret key and encryption method are known to reverse the process.

$$E(P,K) \rightarrow C$$

$$E^{-1}(C,K) \rightarrow P$$

Hardware encryption is favored over software implementations due to speed and protections from traditional attacks. Lightweight cryptographic block ciphers are designed to encrypt blocks of data in constrained applications such as embedded processors, internet-of-things (IoT), etc. In this lab, we will be implementing the **Simon32/64 block cipher** developed by the National Security Agency in 2013 [1]. Simon32/64 uses a block size of 32 bits and key size of 64 bits and word size of 16 bits for encryption and decryption. We will be implementing **ENCRYPTION ONLY with 10 rounds**.

Pre-lab requirements:

1. Datapath

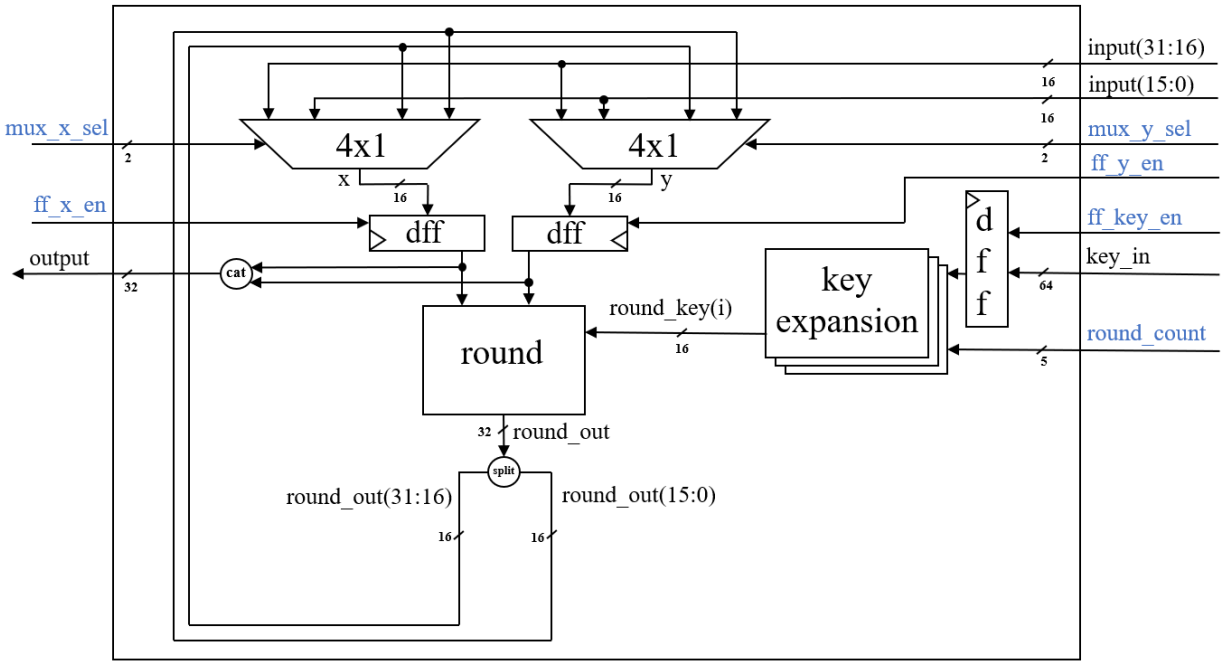

Figure 1. Simon 32/64 Datapath

The datapath and control signals (blue) for Simon32/64 block cipher are shown in Fig. 1. **Please note: The round_count signal keeps track of which round the simon cipher is on.** We will now look into the round and key expansion functions in greater detail.

a. Round

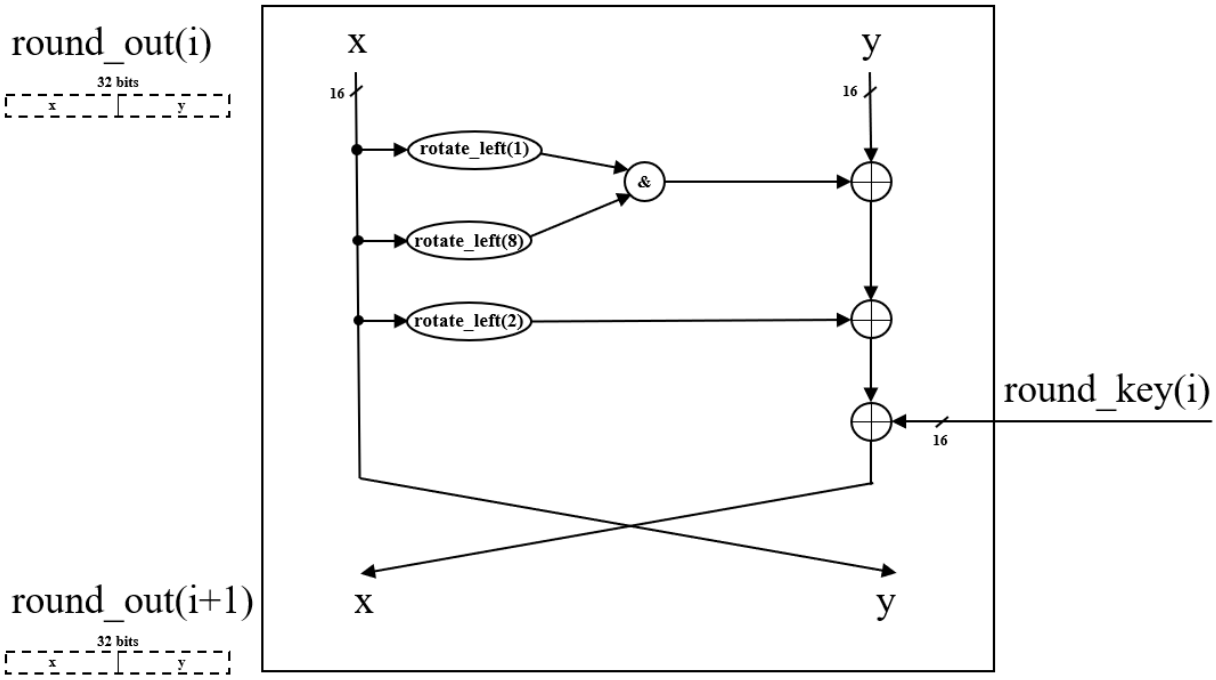

Figure 2. Simon 32/64 Round function.

The round operation applies XOR (linear confusion), AND, and CIRCULAR SHIFTING (ROTATE) (non-linear diffusion) to encrypt a plaintext into ciphertext.

Use the provided file (round.vhd) to complete the behavioral description of the round function as shown in Fig. 2. Pseudocode of the round function is shown in the Appendix for reference. You can test the round function using the provided round_tb.vhd.

b. Key Expansion

The key expansion module generates a unique round key for every round of the Simon32/64. The first 4 round keys use the initial 64-bit key (starting from least significant in 16 bit increments). All subsequent round keys are generated using the datapath shown in Fig. 3. Use the provided file (key_expansion.vhd) to complete the behavioral description of the key expansion function as shown in the figure. Z is defined for you in constants.vhd. Pseudocode of the key expansion is shown in the Appendix for reference. **Please note: in Fig. 3 *i* is the current round_count shown in Fig. 1.**

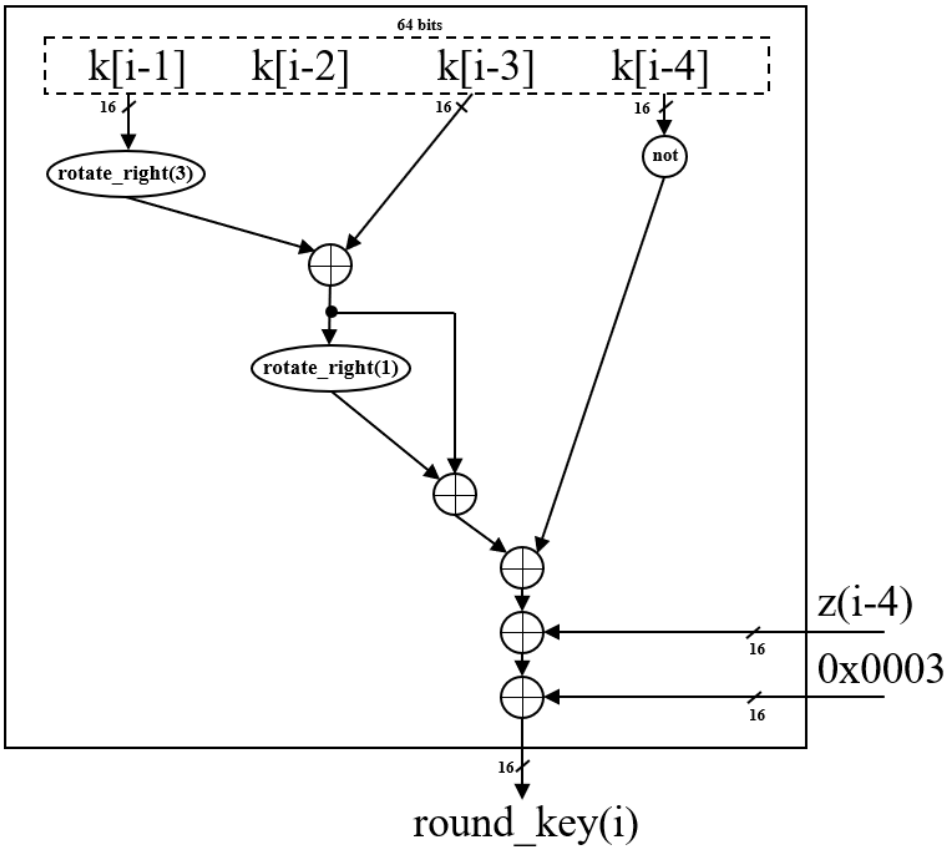

Figure 3. Simon 32/64 Key Expansion.

Use the provided testbench key_expansion_tb.vhd to test the functionality of your key_expansion module.

Using structural VHDL, complete simon.vhd instantiating the newly created round and key expansion components. Create any other necessary components to complete the datapath according to Figure 1. Hint: Key expansion must generate ALL round keys before starting encryption.

2. Memory and Address Generation

We will be using an input ROM for providing the 64-bit key, an input RAM for reading 32-bit blocks of data to be encrypted/decrypted, and an output RAM for writing the 32-bit outputs.

You need to go to Assignments->Device->Device and Pin Options…->Configuration and Set Configuration Mode to "Single Uncompressed Image with Memory Initialization"

The ROM component is made for the alysyncram component provided by Quartus. Create the ROM by:

- Tools -> IP Catalog.
- Search 'ROM' and select 'ROM: 1-PORT'.
- Name: keyrom. IP variation file type: VHDL.
- Click "OK"
- 'q' output bus should be 64 bits.

- There should be 1 word. The option is unavailable, so manually type 1.
- Leave everything else as default.
- Click “Next”
- Uncheck ‘q’ output port under “Which ports should be registered?”
- Leave everything else as default.
- Click “Next”
- Browse to the provided file key.mif
- Check “Allow In-System Memory Content Editor to capture and update content independently of system clock”.
- Leave everything else as default and finish.
- The generated file (keyrom.vhd) can now be used as a component in your design.

The RAM component is made for the alysyncram component provided by Quartus. Create the RAM by:

- Tools -> IP Catalog.
- Search ‘ROM’ and select ‘RAM: 1-PORT’.
- Name: inram. IP variation file type: VHDL.
- Click “OK”
- ‘q’ output bus should be 32 bits.
- There should be 32 words.
- Leave everything else as default.
- Click “Next”
- Uncheck ‘q’ output port under “Which ports should be registered?”
- Leave everything else as default.
- Click “Next”
- Click “Next” again
- Browse to the provided file in.mif
- Check “Allow In-System Memory Content Editor to capture and update content independently of system clock”.
- Leave everything else as default and finish.
- The generated file (inram.vhd) can now be used as a component in your design.

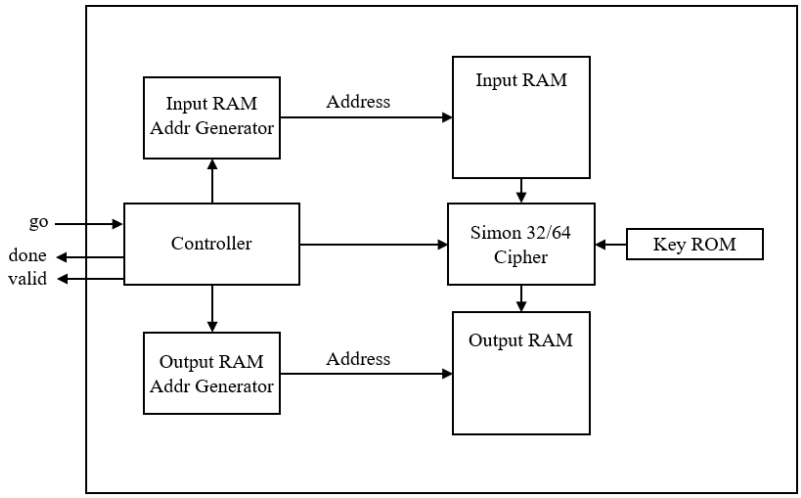

Figure 4. Block Diagram design of Simon 32/64.

Similarly, the output ram is created (outram.vhd) with the above steps except leave the contents blank (DO NOT PROVIDE .mif FILE).

The input and output address generators are both simple counters with enables.

3. Controller

Complete the provided controller.vhd file by creating a 2-process FSM to generate the appropriate signals for controlling the Simon32/64 datapath from Fig. 1 and address generation for reading and writing to memory. The controller should:

- Wait for 'go' signal to be asserted.
- Load key from key ROM to dff_key. This enable should be de-asserted for the remainder of the FSM. Initialize round count. Read input from input RAM.
- Keep track of round_count. Reset round_count after fully encrypting.
- Write output to output RAM and assert 'valid' signal for at least one cycle to signify valid data has been generated.
- Enable input and output address generators. Check current address from input or output RAM to identify final address (done condition is when addr = 31).
- Continually assert 'done' after all inputs are encrypted.

Initial Round

The first round (round_out(0)) uses the original input signal to drive x and y round inputs. All subsequent rounds will use the output of the previous round as input. Use this fact to drive mux_x_sel, and mux_y_sel control signals in your controller.

Memory Delay

The instantiated RAM components have a read delay of 1 cycle. One way to deal with this is to introduce appropriate delay when moving onto the data in InputRAM[address +1]. Alternative solutions are acceptable.

4. Final test setup

Use the provided simon_top.vhd to create a top level simon cipher entity that connects together your controller, simon datapath, input and output address generators, input and output RAMs, and key ROM.

Map 'go', 'rst' to switches (SW1-SW0). Map 'done' and 'valid to LED0, LED1, respectively.

Use the provided testbench simon_top_tb.vhd to test simon_top. Use the provided key.mif for the key ROM and in.mif for the input RAM. For simulation, make sure the init_file generic map points to the correct .mif location on your computer.

Turn in the design files and annotated simulation results of the final circuit for one complete encryption.

In-lab procedure:

1. Compile the final test setup from Prelab step 4 and program it onto your board.
2. Configure your input key ROM to use the lab-provided key.mif and your input RAM to use the lab-provided input.mif file using the In-System Memory Content Editor.

3. Using the In-System Memory Content Editor, observe the contents of the output RAM. Debug if necessary. Once satisfied with the contents, show your content editor and board to the TA. Be prepared to answer any questions regarding the implementation.

Extra Credit:

Implement the decryption feature of the Simon Cipher. Add a 'mode' input in simon_top.vhd and controller.vhd to distinguish between encryption and decryption. You should only need to edit the controller to support decryption. Hint: decryption starts at round 9 and ends at round 0. Turn in all design files and testbenches used to verify decryption.

Appendix:

Simon32/64 Pseudocode [1]

```
--Z is 0 indexed.
Z=[11111010001001010110000111001101111101000100101011000011100110]
----------------------- key expansion -----------------------
for i = 4…9 {
    tmp = circular_shift_right(round_key[i-1], 3)
    tmp = tmp xor round_key[i-3]
    tmp = tmp xor circular_shift_right(tmp, 1)
    round_key[i] = ~(round_key[i-4]) xor tmp xor
                   z[i-4 mod 62] xor 3
}
----------------------- round -----------------------
    tmp = x

    x = y xor
    (circular_shift_left(x,1) and circular_shift_left(x,8)) xor
    circular_shift_left(x,2) xor
    round_key[i]

    y = tmp
----------------------- encryption -----------------------

for i = 0…9
 round(x,y,i)

----------------------- decryption -----------------------

for i = 9 … 0
  round(y,x,i)
```

Table I: Simon 32/64 key expansion for 5 different keys

round	key 0x1918111009080100	key 0x123456789abcdef0	key 0xfeedbeeffeedbeef	key 0x6589412352147750	key 0x1542F58747B562CC
0	0x0100	0xDEF0	0xBEEF	0x7750	0x62CC
1	0x0908	0x9ABC	0xFEED	0x5214	0x47B5
2	0x1110	0x5678	0xBEEF	0x4123	0xF587
3	0x1918	0x1234	0xFEED	0x6589	0x1542
4	0x71C3	0x358A	0x20BA	0x495A	0x1AA2
5	0xB649	0xFDEC	0x8694	0xA1E5	0xD5F2
6	0x56D4	0xE2C8	0x9832	0x87B1	0x6278
7	0xE070	0x50F3	0x4B72	0xDF0E	0x7724
8	0xF15A	0x167C	0x7740	0xF1A1	0x93C2
9	0xC535	0xD214	0xA4DF	0x3B5E	0x020E

Table II: Expanded rounds for Simon 32/64 encryption & decryption

Key = 0x1918111009080100
Plaintext = 0x74657374
Ciphertext = 0xECFB7A9E

encryption				decryption			
round	x (in)	y (in)	round key	round	x (in)	y (in)	round key
0	0x7465	0x7374	0x0100	9	0x7A9E	0xECFB	0xC535
1	0xC3A1	0x7465	0x0908	8	0x578F	0x7A9E	0xF15A
2	0xF2A9	0xC3A1	0x1110	7	0x5AEF	0x578F	0xE070
3	0xB944	0xF2A9	0x1918	6	0x7918	0x5AEF	0x56D4
4	0x4E2A	0xB944	0x71C3	5	0xF86A	0x7918	0xB649
5	0xF86A	0x4E2A	0xB649	4	0x4E2A	0xF86A	0x71C3
6	0x7918	0xF86A	0x56D4	3	0xB944	0x4E2A	0x1918
7	0x5AEF	0x7918	0xE070	2	0xF2A9	0xB944	0x1110
8	0x578F	0x5AEF	0xF15A	1	0xC3A1	0xF2A9	0x0908
9	0x7A9E	0x578F	0xC535	0	0x7465	0xC3A1	0x0100
final	**0xECFB**	**0x7A9E**	--	**final**	**0x7465**	**0x7374**	--

References:

[1] R. Beaulieu, et al., "The SIMON and SPECK Families of Lightweight Block Ciphers." https://eprint.iacr.org/2013/404