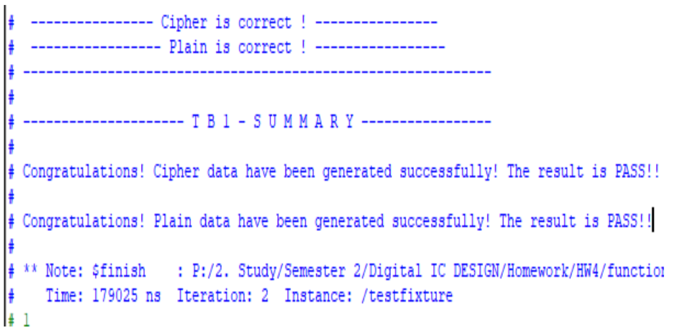
# 2020 Digital IC Design Homework 4: RC4 Encrypt

| | |
|---|---|
| NAME | Nguyen Vu Le Minh |
| Student ID | P76087099 |

**Simulation Result**

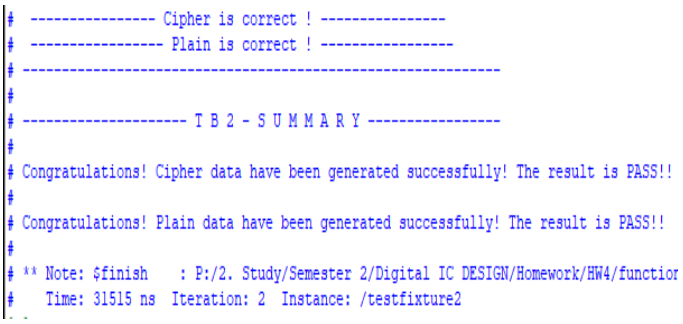
| | | | | | |
|---|---|---|---|---|---|
| Functional simulation | Pass | Gate-level simulation | Pass | Gate-level simulation time | 161132455 (ps)<br>85100455 (ps) |

**(result 1)**



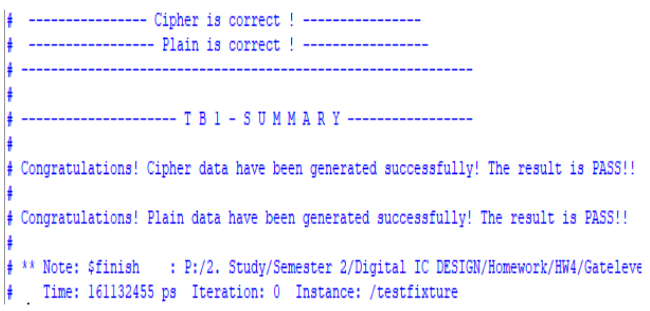
```
#  --------------- Cipher is correct ! ---------------
#  ---------------- Plain is correct ! ----------------
# ------------------------------------------------------------
#
# -------------------- T B 1 - S U M M A R Y ----------------
#
# Congratulations! Cipher data have been generated successfully! The result is PASS!!
#
# Congratulations! Plain data have been generated successfully! The result is PASS!!
#
# ** Note: $finish    : P:/2. Study/Semester 2/Digital IC DESIGN/Homework/HW4/functio
#    Time: 179025 ns  Iteration: 2  Instance: /testfixture
# 1
```

**(result 2)**



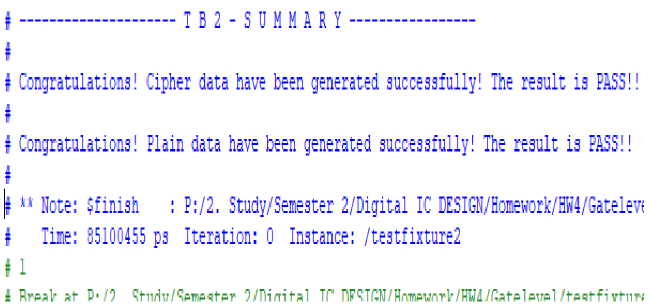
```
#  --------------- Cipher is correct ! ---------------
#  ---------------- Plain is correct ! ----------------
# ------------------------------------------------------------
#
# -------------------- T B 2 - S U M M A R Y ----------------
#
# Congratulations! Cipher data have been generated successfully! The result is PASS!!
#
# Congratulations! Plain data have been generated successfully! The result is PASS!!
#
# ** Note: $finish    : P:/2. Study/Semester 2/Digital IC DESIGN/Homework/HW4/functio
#    Time: 31515 ns  Iteration: 2  Instance: /testfixture2
```

**(result 1)**



```
#  --------------- Cipher is correct ! ---------------
#  ---------------- Plain is correct ! ----------------
# ------------------------------------------------------------
#
# -------------------- T B 1 - S U M M A R Y ----------------
#
# Congratulations! Cipher data have been generated successfully! The result is PASS!!
#
# Congratulations! Plain data have been generated successfully! The result is PASS!!
#
# ** Note: $finish    : P:/2. Study/Semester 2/Digital IC DESIGN/Homework/HW4/Gateleve
#    Time: 161132455 ps  Iteration: 0  Instance: /testfixture
```

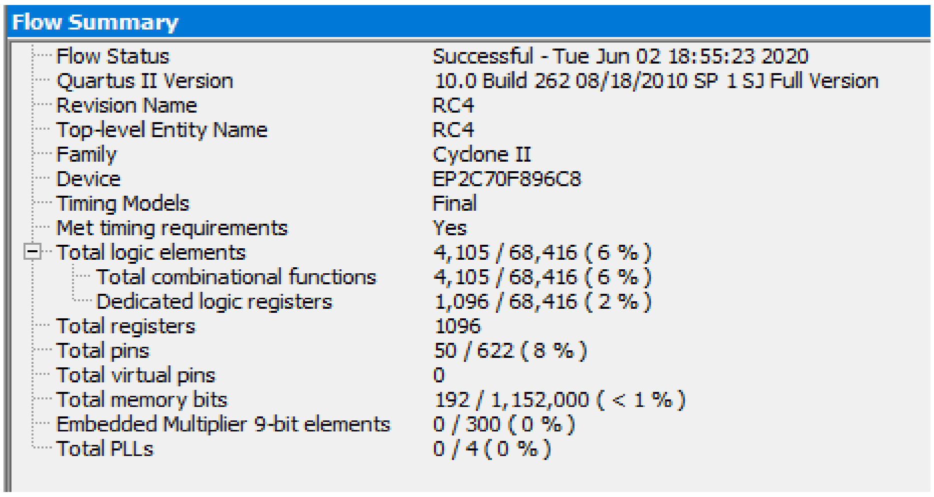
**(result 2)**



```
# -------------------- T B 2 - S U M M A R Y ----------------
#
# Congratulations! Cipher data have been generated successfully! The result is PASS!!
#
# Congratulations! Plain data have been generated successfully! The result is PASS!!
#
# ** Note: $finish    : P:/2. Study/Semester 2/Digital IC DESIGN/Homework/HW4/Gatelev
#    Time: 85100455 ps  Iteration: 0  Instance: /testfixture2
# 1
# Break at P:/2. Study/Semester 2/Digital IC DESIGN/Homework/HW4/Gatelevel/testfixture
```

**Synthesis Result**

| | |
|---|---|
| Total logic elements | 4,105 / 68,416 (6 %) |
| Total memory bit | 192 / 1,152,000 (< 1 %) |
| Embedded multiplier 9-bit element | 0 / 300 (0 %) |



Flow Summary

| | |
|---|---|
| Flow Status | Successful - Tue Jun 02 18:55:23 2020 |
| Quartus II Version | 10.0 Build 262 08/18/2010 SP 1 SJ Full Version |
| Revision Name | RC4 |
| Top-level Entity Name | RC4 |
| Family | Cyclone II |
| Device | EP2C70F896C8 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 4,105 / 68,416 ( 6 % ) |
| Total combinational functions | 4,105 / 68,416 ( 6 % ) |
| Dedicated logic registers | 1,096 / 68,416 ( 2 % ) |
| Total registers | 1096 |
| Total pins | 50 / 622 ( 8 % ) |
| Total virtual pins | 0 |
| Total memory bits | 192 / 1,152,000 ( < 1 % ) |
| Embedded Multiplier 9-bit elements | 0 / 300 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

**Description of your design**

RC4 is a stream cipher and variable length key algorithm. This algorithm encrypts one byte at a time. A key input is pseudorandom bit generator that produces a stream 8-bit number that is unpredictable without knowledge of input key. RC4 is the encryption algorithm used in Wired Equivalent Encryption (WEP) and was previously one of the algorithms used by TLS. The RC4 encryption and decryption use the same set of algorithms as for the coincidence of XOR operation.

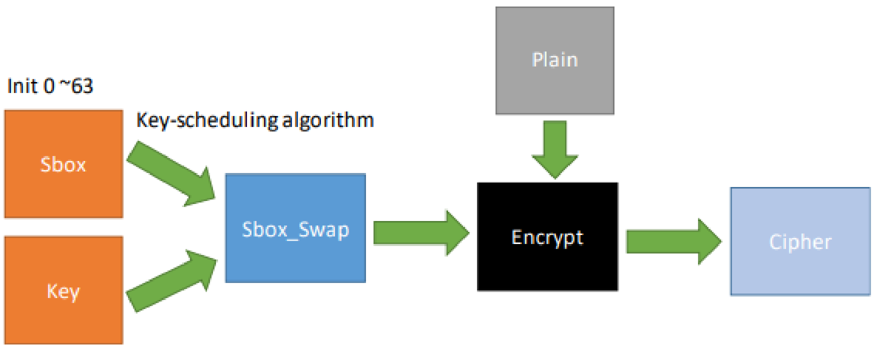
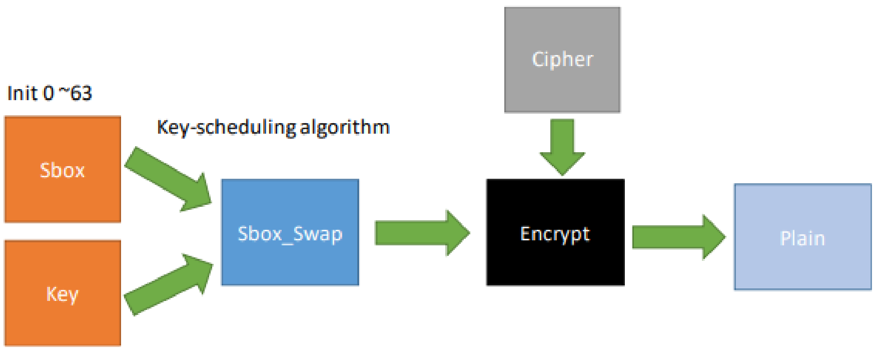
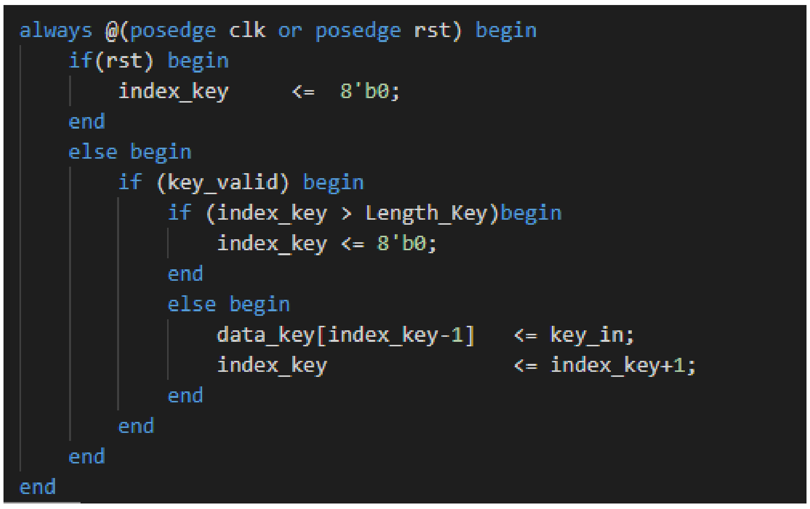


Figure 2 – Encryption flow

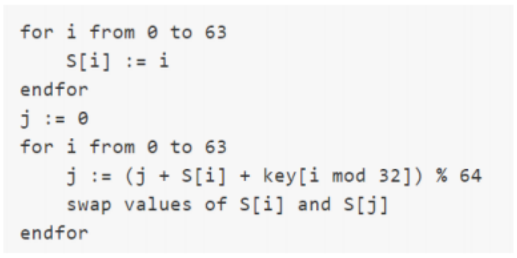


First of all, I will get all data of key and save them in array. The key length of this system is 32, and the key data is stored in the testfixture.



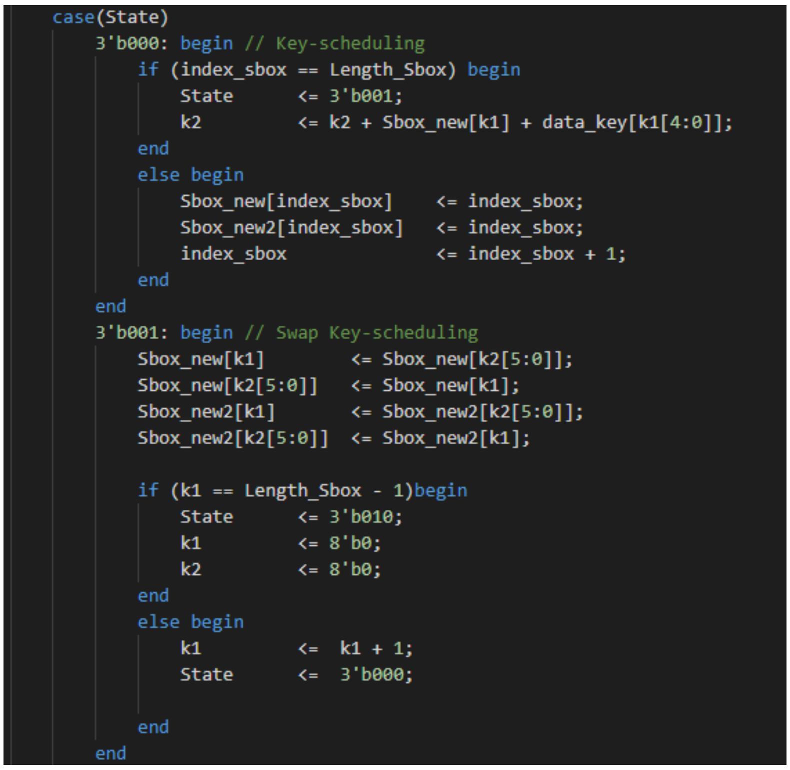
```
always @(posedge clk or posedge rst) begin
    if(rst) begin
        index_key     <=  8'b0;
    end
    else begin
        if (key_valid) begin
            if (index_key > Length_Key)begin
                index_key <= 8'b0;
            end
            else begin
                data_key[index_key-1]    <= key_in;
                index_key                <= index_key+1;
            end
        end
    end
end
```

After the key value is input, shuffle the key and S box. S box is 0 ~ 63 at the beginning, use the following Pseudo codes to shuffle, and then the shuffled S box is used for encryption. I separate many states to process algorithms. The first state, I use for KSA.



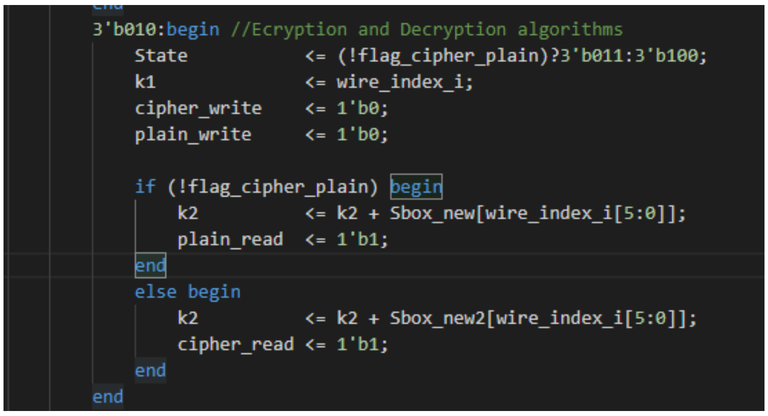
```
for i from 0 to 63
    S[i] := i
endfor
j := 0
for i from 0 to 63
    j := (j + S[i] + key[i mod 32]) % 64
    swap values of S[i] and S[j]
endfor
```

Figure 4 – Key-scheduling algorithm (KSA)



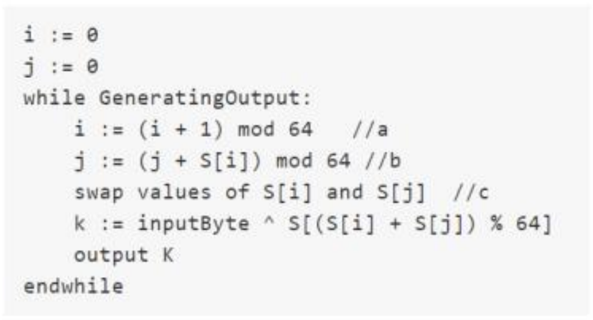
```
case(State)
    3'b000: begin // Key-scheduling
        if (index_sbox == Length_Sbox) begin
            State      <= 3'b001;
            k2         <= k2 + Sbox_new[k1] + data_key[k1[4:0]];
        end
        else begin
            Sbox_new[index_sbox]    <= index_sbox;
            Sbox_new2[index_sbox]   <= index_sbox;
            index_sbox              <= index_sbox + 1;
        end
    end
    3'b001: begin // Swap Key-scheduling
        Sbox_new[k1]        <= Sbox_new[k2[5:0]];
        Sbox_new[k2[5:0]]   <= Sbox_new[k1];
        Sbox_new2[k1]       <= Sbox_new2[k2[5:0]];
        Sbox_new2[k2[5:0]]  <= Sbox_new2[k1];

        if (k1 == Length_Sbox - 1)begin
            State      <= 3'b010;
            k1         <= 8'b0;
            k2         <= 8'b0;
        end
        else begin
            k1         <=  k1 + 1;
            State      <=  3'b000;

        end
    end
```

The next stage, I shuffle elements in Sbox array with together in order to prepare execute encryption and decryption algorithms.



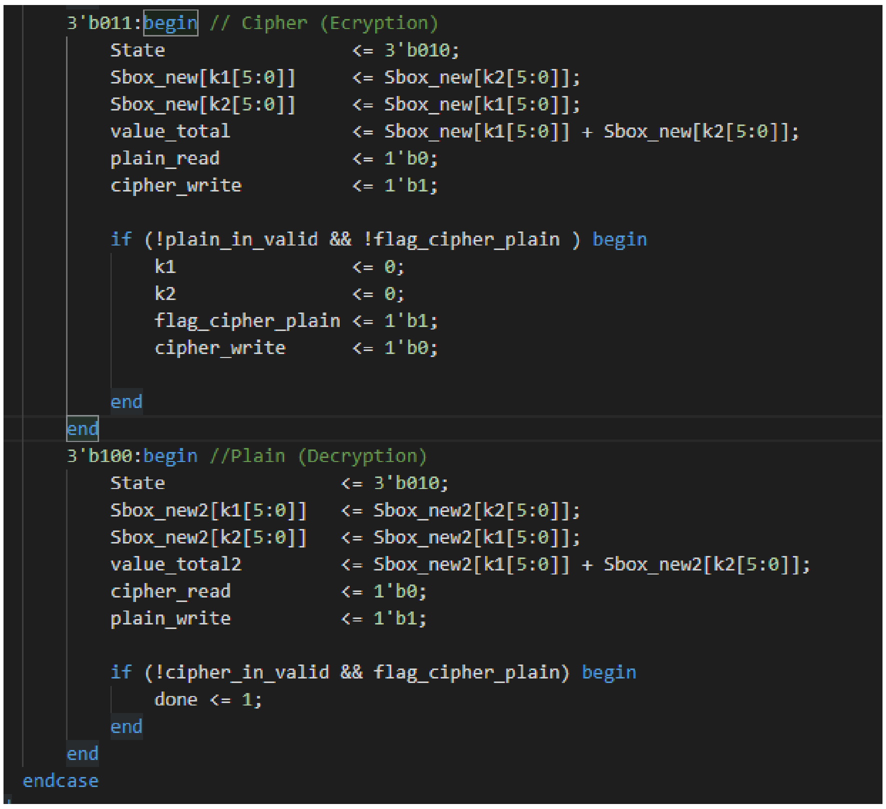
```
    3'b010:begin //Ecryption and Decryption algorithms
        State           <= (!flag_cipher_plain)?3'b011:3'b100;
        k1              <= wire_index_i;
        cipher_write    <= 1'b0;
        plain_write     <= 1'b0;

        if (!flag_cipher_plain) begin
            k2          <= k2 + Sbox_new[wire_index_i[5:0]];
            plain_read  <= 1'b1;
        end
        else begin
            k2          <= k2 + Sbox_new2[wire_index_i[5:0]];
            cipher_read <= 1'b1;
        end
    end
```

Finally, I used a multiplexer to control and execute encryption and decryption algorithms follow in figure 5. The stage 3’b011 that contain all processing to execute Encryption and the result is assigned on cipher_out. On the other hand, the stage 3’b100 that execute the decryption algorithm that is the same as encryption. After finished the encryption and decryption, value done will be set to high.



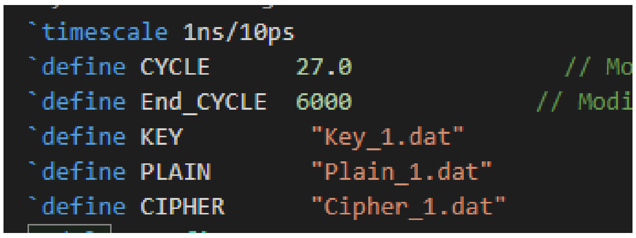
```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 64   //a
    j := (j + S[i]) mod 64 //b
    swap values of S[i] and S[j]  //c
    k := inputByte ^ S[(S[i] + S[j]) % 64]
    output K
endwhile
```

Figure 5 –Flow chart of encryption and decryption algorithm

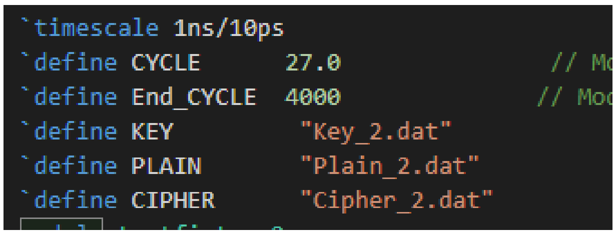


```
    3'b011:begin // Cipher (Ecryption)
        State               <= 3'b010;
        Sbox_new[k1[5:0]]   <= Sbox_new[k2[5:0]];
        Sbox_new[k2[5:0]]   <= Sbox_new[k1[5:0]];
        value_total         <= Sbox_new[k1[5:0]] + Sbox_new[k2[5:0]];
        plain_read          <= 1'b0;
        cipher_write        <= 1'b1;

        if (!plain_in_valid && !flag_cipher_plain ) begin
            k1              <= 0;
            k2              <= 0;
            flag_cipher_plain <= 1'b1;
            cipher_write    <= 1'b0;

        end
    end
    3'b100:begin //Plain (Decryption)
        State              <= 3'b010;
        Sbox_new2[k1[5:0]]  <= Sbox_new2[k2[5:0]];
        Sbox_new2[k2[5:0]]  <= Sbox_new2[k1[5:0]];
        value_total2       <= Sbox_new2[k1[5:0]] + Sbox_new2[k2[5:0]];
        cipher_read        <= 1'b0;
        plain_write        <= 1'b1;

        if (!cipher_in_valid && flag_cipher_plain) begin
            done <= 1;
        end
    end
endcase
```

I also edit cycle and end_cycle in order to get the best result.



```
`timescale 1ns/10ps
`define CYCLE      27.0          // Mo
`define End_CYCLE  6000         // Modi
`define KEY         "Key_1.dat"
`define PLAIN       "Plain_1.dat"
`define CIPHER      "Cipher_1.dat"
```



```
`timescale 1ns/10ps
`define CYCLE      27.0              // Mo
`define End_CYCLE  4000             // Mod
`define KEY         "Key_2.dat"
`define PLAIN       "Plain_2.dat"
`define CIPHER      "Cipher_2.dat"
```

*Scoring = (Total logic elements + total memory bit + 9*embedded multiplier 9-bit element) × (gate-level simulation time in ns)*