專案(二)

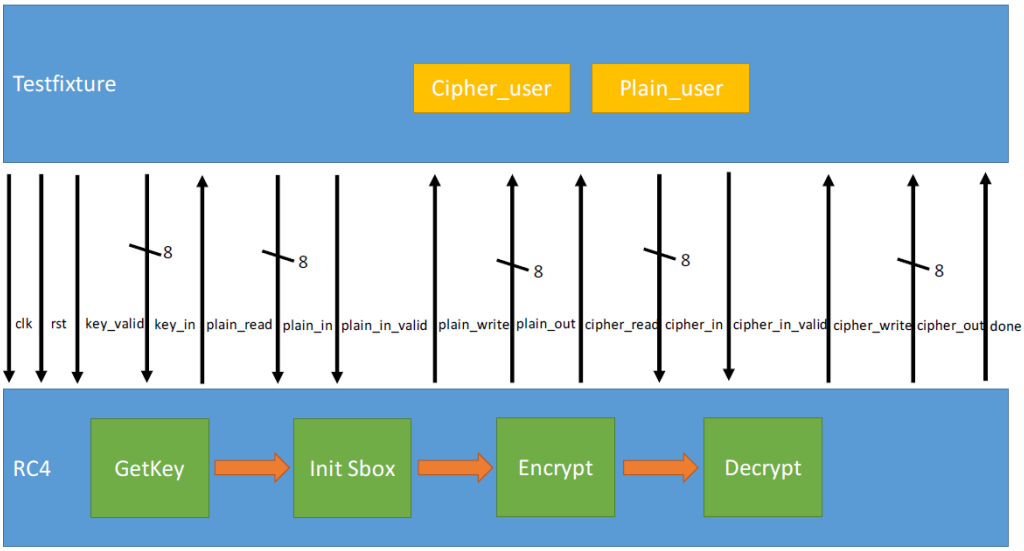
專案名稱 - Rivest Cipher 4 加解密電路實現

簡介

在密碼學中，RC4（來自 Rivest Cipher 4的縮寫）是一種流加密算法，密鑰長度可變。它加解密使用相同的密鑰，因此也屬於對稱加密算法。RC4是有線等效加密（WEP）中採用的加密算法，也曾經是TLS可採用的算法之一。由於 RC4算法存在弱點，2015年 2月所發布的RFC7465規定禁止在TLS中使用 RC4加密算法。RC4由偽隨機數生成器和異或運算組成。~~RC4的密鑰長度可變，範圍是[1,255]~~。RC4一個字節一個字節地加解密。給定一個密鑰，偽隨機數生成器接受密鑰並產生一個Sbox。Sbox用來加密數據，而且在加密過程中Sbox會變化。由於異或運算的對合性，RC4加密解密使用同一套算法。此次作業請實作一個RC4加解密的電路，其中密鑰的長度固定為32bytes，明文的長度為不固定，最長長度不超過2048bytes，Sbox大小為64bytes利用輸入金鑰對明文進行加密，然後將加密完的字元輸出，再將所輸出加密的字元輸入，進行解密，還原出原本的明文。

設計規格

Block overview

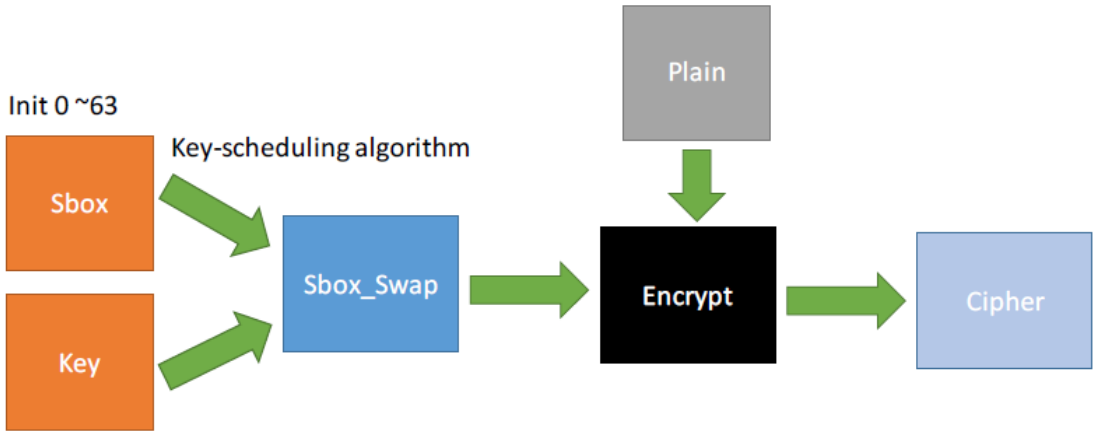


圖一、系統方塊圖

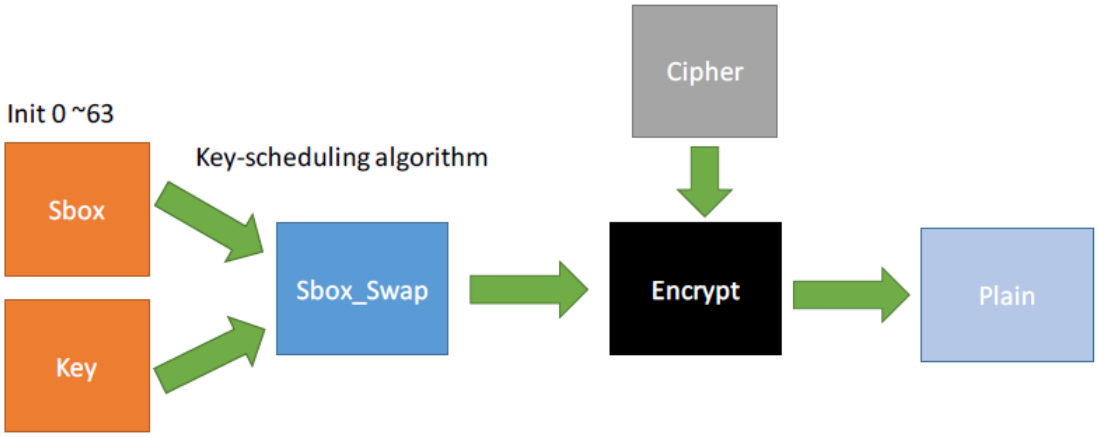
I/O Interface

Name	I/O	Width	Description
clk	I	1	系統時脈訊號。本系統為同步於時脈正緣之同步設計
rst	I	1	高位準"非"同步(active high asynchronous)之系統重置信號
key_valid	I	1	當 key 準備好時，會先將 key_valid 設成 high，然後在下一個負緣輸出 key 值。
key_in	I	8	key data 輸入訊號線，輸入的 data size 為 8 bits，在 key_valid 設成 high 的下一個 cycle 的負緣輸出。
plain_read	O	1	當要索取明文時請將 plain_read 設成 high 且 plain_write 設成 low，再下一個 cycle 後會將其值輸入。
plain_in	I	8	輸入明文資料訊號，由 8bits 整數組成，為無號數
plain_in_valid	I	1	因為明文長度不固定，所以當輸入的明文為有效時，plain_in_valid 為 high，若無效時 plain_in_valid 為 low。
plain_write	O	1	若要將解密後的明文輸出至 testfixture 記憶體，將 plain_write 設為 high，其他不須寫入時務必設成 low。
plain_out	O	8	解密後運算結果記憶體寫出訊號，由 8bits 整數(MSB)組成，為無號數。
cipher_read	O	1	當要索取密文時請將 cipher_read 設成 high 且 cipher_write 設成 low，再下一個 cycle 後會將其值輸入。
cipher_in	I	8	輸入密文資料訊號，由 8bits 整數組成，為無號數
cipher_in_valid	I	1	因為密文長度不固定，所以當輸入的密文為有效時，cipher_in_valid 為 high，若無效時 cipher_in_valid 為 low。
cipher_write	O	1	若要將加密後的密文輸出至 tb 記憶體，將 cipher_write 設為 high，其他不須寫入時務必設成 low。
cipher_out	O	8	加密後運算結果記憶體寫出訊號，由 8bits 整數(MSB)組成，為無號數。
done	O	1	如果系統的運算結束，將 done 訊號輸出

Function Description



圖一、加密流程圖



圖二、解密流程圖

```
for i from 0 to 63:
    sbox[i]=i;

k=0;
for j from 0 to 63:
    k=(k+sbox[j]+key_men[j % 32]) % 64
    swap values of sbox[j] and sbox[k]
```

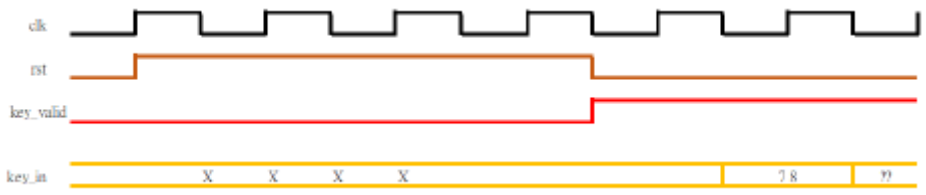
圖三、Key-scheduling algorithm (KSA)

```
a=0
b=0
while GeneratingOutput:
    a = (a+1) % 64
    b = (b+sbox[a])%64
    swap values of sbox[a] and sbox[b]
    c = inputByte^sbox[(sbox[a]+sbox[b])% 64]
    output c
endwhile
```

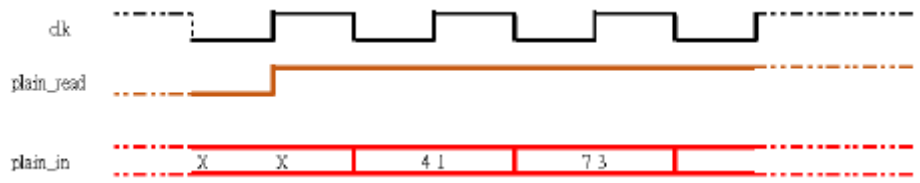
圖三、加解密演算法

本系統的 key 長度為 32bytes，而明文的長度為不固定，最長為 2048bytes，key 的資料儲存於 testfixture 中，在系統進行 reset 之後，下一個 cycle 會先輸出 key_valid= high 然後再過一個 cycle 後輸出 key 的值，當 key_valid 為 high(除了第一個 cycle)時代表 key 值有效，當 key 值輸入完畢後，同學需先將 key 跟 Sbox 進行打亂，Sbox 一開始為 0~63，利用圖四的 Pseudo code 進行打亂後，再利用打亂後的 Sbox 進行加密，加密演算法如圖五的 Pseudo code，當加密完成後的密文請利用 cipher_write 和 cipher_out 將其結果輸出至 testfixture 的記憶體中，當 plain_in_valid 等於 high 時代表明文為有效輸入，當 plain_in_valid 為 low 時代表明文輸入完畢，當明文輸入完畢後，方可藉由 cipher_read 來控制密文的輸入（注意：若明文加密後有錯，輸入的密文也是錯誤的)，當 cipher_in_valid 等於 high 時代表密文為有效輸入，當 cipher_in_valid 為 low 時代表為密文輸入完畢，解密的演算法流程與加密相同，若系統已經將加解密動作完成時，請將 done 設為 high，即可驗證加密。

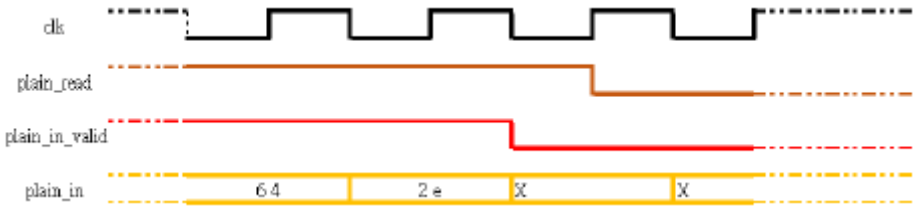
圖六為 key_in 輸入的時序圖，key_valid 為 high 後下一個 clk cycle 便將 key 值輸入。圖七為 cipher data 及 plain data 的讀取時序圖，當 read 設為 high 的下一個 clk 將會把資料輸入。圖八為 cipher data 及 plain data 的資料結束時序圖，當 plain_in_valid 或 cipher_in_valid 由 high 轉 low 時代表資料輸入結束。



圖六、key_in 時序圖



圖七、plain data 及 cipher data 時序圖



圖八、plain data 及 cipher data 結束時序圖

Result

Gate-Level simulation

```
-----------------------------------------------------

 ---------------- 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    : E:/ModelSim projects/DIC_HW4/testfixture.v(244)
   Time: 91704387 ps  Iteration: 0  Instance: /testfixture
```

Synthesis result

Flow Status	Successful - Wed Sep 09 14:41:39 2020
Version	10.0 Build 262 08/18/2010 SP 1 SJ Full Version
Name	RC4
Top-level Entity Name	RC4
Family	Cyclone II
Device	EP2C70F896C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	3,216 / 68,416 (5 %)
Total combinational functions	3,022 / 68,416 (4 %)
Dedicated logic registers	528 / 68,416 (< 1 %)
Total registers	528
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 %)