國立成功大學

工程科學系學士班

專題報告

SHA3 安全雜湊演算法與硬體實現

SHA3 Secure Hash Algorithm and Hardware Implementation

鄭智宇

宋方瑜

陳慕丞

許庭毓

指導教授：卿文龍

中華民國 112 年 9 月

摘要

現代社會對於訊息隱私與數據保護越來越重視，密碼學作為保護網路世界的關鍵技術之一，受到學術界的廣泛研究。虛擬貨幣中，為了確保資料的安全性，引入雜湊函數 (Hash Function)，利用 SHA-3 演算法生成雜湊值儲存使用者設定的密碼，由於雜湊值皆有不可回推的特性，使用者不用擔心密碼被反向破解的可能性，因此保障虛擬貨幣資料的安全性。

本專題中，我們以 Verilog 作為主要編譯語言，因為硬體描述語言相較於軟體語言更快速，可同時執行多個操作，也意味著需要的時間更少，效率更高。

SHA-3 演算法中有兩種不同的模式，其一為加密雜湊函數(Cryptographic hash function, Hash)，分別是 SHA3-224、SHA3-256、SHA3-384、SHA3-512，橫槓後的數字為此模式固定的輸出長度，後面數字的兩倍代表此模式所指定的容量長度。其二為可擴張輸出函數(Extendable-output functions, XOF)，分別是 SHAKE128、SHAKE256，可隨意指定輸出長度，後面數字的兩倍同樣代表此模式所指定的容量長度。

本研究中實作四種不同模式與容量長度的演算法，分別為 SHA3-256、SHA3-512、SHAKE128、SHAKE256。

完成的 SHA-3 電路使用 Cadence 的 NC-Verilog 進行 RTL 與 Gate-Level 模擬，利用 Synopsys 的 Verdi 進行除錯，最後利用 Design Compiler 在時脈週期為 6.5 ns 與 TSMC 40nm 的製程下利用 compile\_ultra 指令進行合成，經過多次版本迭代後，面積(cell base area)為 614853.872096 。

**關鍵字：**SHA-3、海綿結構、Pipeline

Abstract

The importance of information privacy and data protection has grown significantly in modern society. Cryptography, as one of the key technologies for safeguarding the digital world, has been extensively researched in academia. Additionally, ensuring data security is significant within the realm of virtual currencies. To achieve this, hash functions, specifically utilizing the SHA-3 algorithm, are employed to generate hash values for storing user-defined passwords. The irreversibility of hash values enhances the security of virtual currency data.

In this research project, Verilog is utilized as the primary compiled language due to its advantages in hardware description over software languages. Hardware description languages(HDL) offer higher speed, concurrent execution of multiple operations, and reduced processing time, resulting in increased efficiency.

The SHA-3 algorithm comprises two distinct modes: the Cryptographic Hash Functions including SHA3-224, SHA3-256, SHA3-384, and SHA3-512, with the appended twice numbers denoting their specified capacity lengths, and the Extendable-Output Functions (XOF) comprising SHAKE128 and SHAKE256, allowing for output lengths customized with the following twice numbers representing the capacity length of each mode.

This study involves the implementation of four different algorithms with varying modes and capacity lengths: SHA3-256, SHA3-512, SHAKE128, and SHAKE256.

The completed SHA-3 circuits undergo RTL and Gate-Level simulations using Cadence's NC-Verilog, and debugging is carried out using Synopsys' Verdi tool. Subsequently, synthesis is performed using Design Compiler with a clock cycle of 6.5 ns and a TSMC 40nm fabrication process, following multiple iterations. The final cell base area is 614853.872096 μm².

**Keywords:** SHA-3, Sponge Structure, Pipeline

目錄

[摘要 I](#_Toc145880122)

[Abstract II](#_Toc145880123)

[目錄 III](#_Toc145880124)

[表目錄 IV](#_Toc145880125)

[圖目錄 V](#_Toc145880126)

[第一章、 研究動機 1](#_Toc145880127)

[第二章、 背景介紹 2](#_Toc145880128)

[第1節、 雜湊函數(Hash Function) 2](#_Toc145880129)

[第2節、 海綿結構(Sponge Construction) 2](#_Toc145880130)

[第3節、 填充規則(Padding Rule) 4](#_Toc145880131)

[第4節、 SHA-3 4](#_Toc145880132)

[第5節、 狀態矩陣(State Array) 5](#_Toc145880133)

[第6節、 迭代函數(Iteration function) 5](#_Toc145880134)

[第三章、 設計分析與考量 12](#_Toc145880135)

[第1節、 輸入資料與輸出資料的條件 12](#_Toc145880136)

[第2節、 Python 產生測資與演算法的實現 13](#_Toc145880137)

[第3節、 硬體設計 14](#_Toc145880138)

[第4節、 設計版本迭代比較 17](#_Toc145880139)

[第四章、 硬體架構 20](#_Toc145880140)

[第1節、 整體架構 20](#_Toc145880141)

[第2節、 細部架構 20](#_Toc145880142)

[第五章、 模擬結果 25](#_Toc145880143)

[第1節、 RTL 模擬波形圖 25](#_Toc145880144)

[第2節、 RTL 模擬 34](#_Toc145880145)

[第3節、 Synthesis 37](#_Toc145880146)

[第4節、 Gate-Level 模擬 38](#_Toc145880147)

[第5節、 Throughput 41](#_Toc145880148)

[第六章、 結論與未來展望 42](#_Toc145880149)

[參考資料 43](#_Toc145880150)

[分工表 44](#_Toc145880151)

表目錄

[表 2‑1 Rho 每個通道的位移量 8](#_Toc145880157)

[表 2‑2 Rho 每個通道的位移量 mod 64 8](#_Toc145880158)

[表 2‑3 Iota(ι) 運算 64 位元回合常數 11](#_Toc145880159)

[表 3‑1 十六進位的 SHA-3 填充原則 15](#_Toc145880160)

[表 3‑2 Iota(ι) 運算 8 位元回合常數 18](#_Toc145880161)

[表 3‑3 三個版本的比較 19](#_Toc145880162)

[表 4‑1 Slice 模組接腳功能定義表 21](#_Toc145880163)

[表 4‑2 Controller模組接腳功能定義表 22](#_Toc145880164)

[表 4‑3 F\_function 模組接腳功能定義表 23](#_Toc145880165)

[表 4‑4 f\_sub 模組之接腳功能定義表 24](#_Toc145880166)

[表 5‑1 PA 數據彙整 37](#_Toc145880167)

[表 5‑2 PA 曲線 37](#_Toc145880168)

[表 5‑3 四種模式 Thoughput 結果 41](#_Toc145880169)

圖目錄

[圖 1‑1 雜湊函數在區塊鏈中的應用 1](#_Toc145880170)

[圖 2‑1 海綿結構示意圖 2](#_Toc145880171)

[圖 2‑2 海綿結構的輸入(Input)與輸出(Output) 3](#_Toc145880172)

[圖 2‑3 海綿結構的吸收階段(Absorbing phase)與擠壓階段(Squeezing phase) 3](#_Toc145880173)

[圖 2‑4 狀態矩陣(State Array)示意圖 5](#_Toc145880174)

[圖 2‑5 Rho位移示意圖 7](#_Toc145880175)

[圖 2‑6 Rho 運算示意圖 8](#_Toc145880176)

[圖 2‑7 Pi 運算示意圖 9](#_Toc145880177)

[圖 2‑8 Chi 運算示意圖 10](#_Toc145880178)

[圖 2‑9 Iota(ι) 運算示意圖 11](#_Toc145880179)

[圖 3‑1 系統初始時序圖 12](#_Toc145880180)

[圖 3‑2 主控端訊號輸出時序圖 13](#_Toc145880181)

[圖 3‑3 SHAKE256 在 Python 中產生的測資結果 14](#_Toc145880182)

[圖 3‑4 Verilog 程式資料預處理 15](#_Toc145880183)

[圖 3‑5 SHA3-512 填充至轉換率長度(r) 16](#_Toc145880184)

[圖 3‑6 SHAKE128 填充至轉換率長度(r) 16](#_Toc145880185)

[圖 3‑7 轉換率(r)填充至 KECCAK-p 寬度位元數(b) 16](#_Toc145880186)

[圖 3‑8 Pipeline 迭代函數 f 設計圖 17](#_Toc145880187)

[圖 3‑9 八位元回合常數擴展過程 17](#_Toc145880188)

[圖 3‑10 版本一、二的 Slice 暫存器示意圖 19](#_Toc145880189)

[圖 3‑11 版本三的 Slice 暫存器示意圖 19](#_Toc145880190)

[圖 4‑1 硬體架構圖 20](#_Toc145880191)

[圖 4‑2 Slice 模組硬體架構圖 21](#_Toc145880192)

[圖 4‑3 Controller模組訊號線圖 22](#_Toc145880193)

[圖 4‑4 F\_function 模組硬體架構圖 23](#_Toc145880194)

[圖 4‑5 f\_sub 模組硬體架構圖 24](#_Toc145880195)

[圖 5‑1 Slice 模擬波形圖之一 25](#_Toc145880196)

[圖 5‑2 Slice 模擬波形圖之二 26](#_Toc145880197)

[圖 5‑3 Slice 模擬波形圖之三 26](#_Toc145880198)

[圖 5‑4 Slice 模擬波形圖之四 26](#_Toc145880199)

[圖 5‑5 Slice 模擬波形圖之五 27](#_Toc145880200)

[圖 5‑6 Slice 模擬波形圖之六 27](#_Toc145880201)

[圖 5‑7 Slice 模擬波形圖之七 27](#_Toc145880202)

[圖 5‑8 Controller 模擬波形圖之一 28](#_Toc145880203)

[圖 5‑9 Controller 模擬波形圖之二 29](#_Toc145880204)

[圖 5‑10 Controller 模擬波形圖之三 30](#_Toc145880205)

[圖 5‑11 Controller 模擬波形圖之四 31](#_Toc145880206)

[圖 5‑12 Controller 模擬波形圖之五 31](#_Toc145880207)

[圖 5‑13 F\_funtion 模擬波形圖 32](#_Toc145880208)

[圖 5‑14 F\_funtion 流程簡圖 33](#_Toc145880209)

[圖 5‑15 f\_sub 模擬波形圖 33](#_Toc145880210)

[圖 5‑16 RTL 模擬之 PIPE 測資 34](#_Toc145880211)

[圖 5‑17 RTL 模擬之 SHA3-256 測資 35](#_Toc145880212)

[圖 5‑18 RTL 模擬之 SHA3-512 測資 35](#_Toc145880213)

[圖 5‑19 RTL 模擬之 SHAKE128 測資 36](#_Toc145880214)

[圖 5‑20 RTL 模擬之 SHAKE256 測資 36](#_Toc145880215)

[圖 5‑21 Gate-Level 模擬之 PIPE 測資 38](#_Toc145880216)

[圖 5‑22 Gate-Level 模擬之 SHA3-256 測資 39](#_Toc145880217)

[圖 5‑23 Gate-Level 模擬之 SHA3-512 測資 39](#_Toc145880218)

[圖 5‑24 Gate-Level 模擬之 SHAKE128 測資 40](#_Toc145880219)

[圖 5‑25 Gate-Level 模擬之 SHAKE256 測資 40](#_Toc145880220)

# 研究動機

現代社會對於訊息隱私與數據保護越來越重視，密碼學作為保護網路世界的關鍵技術之一，受到學術界的廣泛研究，同時也投入到區塊鏈平台與虛擬貨幣的應用。虛擬貨幣中，為了確保資料的安全性，引入雜湊函數(Hash Function)，通過生成雜湊值(Hash Value)儲存使用者設定的密碼，由於雜湊值皆有不可回推的特性，使用者不用擔心密碼被反向破解的可能性，因此保障虛擬貨幣資料的安全性。

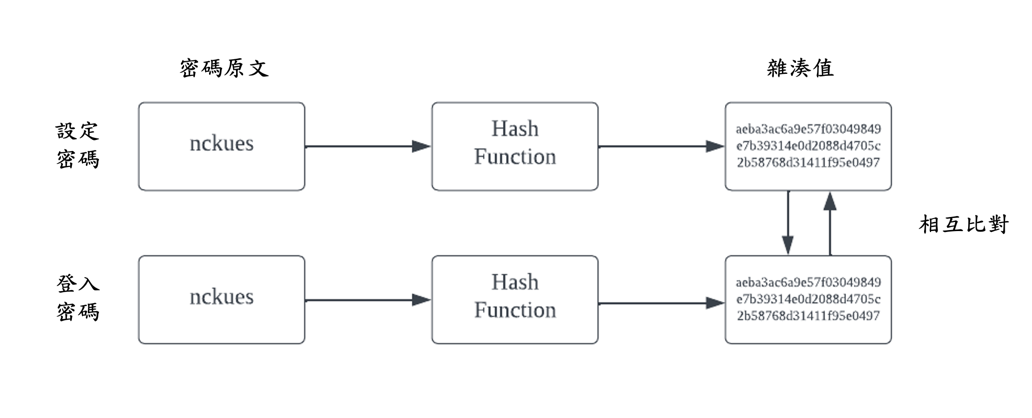


圖 1‑1 雜湊函數在區塊鏈中的應用

其中，我們所研究的 SHA-3 演算法是一種加密雜湊演算法，也是 SHA 家族中最新的演算法，在此之前，SHA-1 已被破解，可能會有不同訊息、相同雜湊值的問題產生，SHA-2 則有訊息長度上的限制，而 SHA-3 不僅對於輸入訊息的最大長度沒有限制，也能夠指定輸出長度，同時也保有雜湊值的安全性。

我們以 Verilog 作為主要編譯語言，因為硬體描述語言(Hardware Description Language, HDL)相較於軟體語言更快速，可同時執行多個操作，也意味著需要的時間更少，效率更高，我們期待能夠使用 FPGA 實現 SHA-3 演算法，達到更佳的效能。

# 背景介紹

## 雜湊函數(Hash Function)

雜湊函數是一種加密方式，用於對訊息進行加密處理。它可為一段明文[[1]](#footnote-1)生成專屬的資料指紋(Data Fingerprint)，稱作雜湊值(Hash Value)。雜湊函數可接受不同長度的輸入，並輸出固定長度的雜湊值，且雜湊值必須符合兩個特性：其一、不能從雜湊值逆向推導出原始訊息，確保數據的安全性；其二、訊息不同，不應該產生相同的雜湊值，若是訊息不同，產生的雜湊值相同，則會被稱為碰撞(Collision)現象，則該雜湊函數會被視為不安全的演算法。

## 海綿結構(Sponge Construction)

SHA-3 演算法中，資料的傳遞是以海綿結構為主要架構，它可以接受任意長度的輸入和生成特定長度的輸出。海綿結構分為兩個階段，明文輸入時為「吸收階段(Absorbing phase)」、輸出雜湊值時為「擠壓階段(Squeezing phase)」。

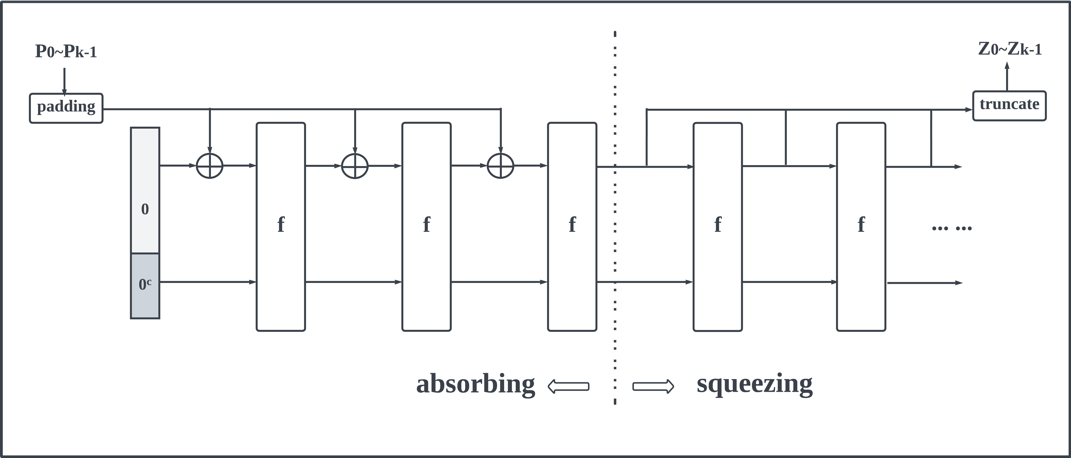


圖 2‑1 海綿結構示意圖

一段明文經過海綿結構的處理流程大致如下，根據輸入明文，海綿結構會先將其切分為多個 r 位元長度的區塊 P ，如圖 2‑2所示，再將小於 r 區塊填充(Padding，見填充規則(Padding Rule))至 r 長度。根據填充規則完成的區塊 P 與區塊 連接後，會與區塊 s 做 XOR 運算後進到迭代函數 ，直到區塊 皆處理完，資料的吸收階段完成，準備進入擠壓階段，而擠壓階段則是不斷地輸出固定長度為 r 的區塊 Z，直到滿足該筆明文指定的輸出長度。

一張含有 文字, 螢幕擷取畫面, 收據, 行 的圖片

自動產生的描述

圖 2‑2 海綿結構的輸入(Input)與輸出(Output)

一張含有 圖表, 方案, 工程製圖, 平行 的圖片

自動產生的描述

圖 2‑3 海綿結構的吸收階段(Absorbing phase)與擠壓階段(Squeezing phase)

## 填充規則(Padding Rule)

迭代構造需要先將分段後的明文在進入函數之前進行填充(Padding)，遇到超過單次可計算長度(r)的明文情況，會先將明文先切為多段 r 位元數長度訊息，再透過 KECCAK-的排列規則並填充至寬度位元數(b)。

若是明文長度為 r 的倍數，則需要在最後多吸收一段長度為 r 的零字串，才可進入擠壓階段。

1. r (Bitrate, 轉換率)：將輸入明文分為多段的位元數長度，r 值代表輸入明文的處理速度。
2. c (Capacity, 容量)：此海綿架構可實現的複雜性與安全程度，若 c 越高則 r 越小，一般會透過 c 的增加或 r 的減少維持安全性，c 越低，安全性也越低。
3. b：KECCAK- 寬度位元數，平衡轉換率與容量的固定位元數()，本研究中 b 固定為1600。

## SHA-3

Kyber 為後量子加密系統架構，由 NTT、INVNTT 等多個單元(Unit)組成，而Keccak f[1600] 為 Kyber 架構(Architecture)中的一個單元，為 Kyber 提供單項雜湊函數的特性(如抗碰撞性、單向性等)，其中 Keccak 又可以分為多個演算法，包含本研究所介紹的 SHA-3 演算法。

SHA-3 演算法中有兩種不同的模式，其一為加密雜湊函數(Cryptographic hash function, Hash)，分別是 SHA3-224、SHA3-256、SHA3-384、SHA3-512，後面數字代表輸出長度，其值為固定長度不可更改，此數字的兩倍也為此模式所指定的容量(c)長度。其二為可擴張輸出函數(Extendable-output functions, XOF)，分別是 SHAKE128、SHAKE256，可隨意指定輸出長度，後面數字的兩倍代表容量(c)長度。需要注意的是，雖然可隨意指定輸出長度，但單次擠壓的輸出長度不能超過容量長度，以確保此運算的安全性。

本研究中，實作四種不同模式與容量長度的演算法，分別為 SHA3-256、SHA3-512、SHAKE128、SHAKE256。

## 狀態矩陣(State Array)

狀態矩陣為 5×5×64 的三維矩陣，存放 1600 位元(bits)的明文，如圖 2‑4 所示，紅色部分為一通道，每一通道為 64 位元(bits)，總共有 25 條通道。

一張含有 寫生, 圖表, 行, 設計 的圖片

自動產生的描述

圖 2‑4 狀態矩陣(State Array)示意圖

## 迭代函數(Iteration function)

迭代函數 的運算順序為 Theta()、Rho()、Pi()、Chi()、Iota()，以下會詳細介紹五個步驟，每一個狀態矩陣都會經過 24 回合的迭代函數。

### Theta(

運算式：

輸入：狀態矩陣 A

輸出：狀態矩陣 E



圖 2-5 Theta()運算示意圖

### Rho()

運算式：

輸入：矩陣 A

輸出：矩陣 E

1. ，通道 輸入和輸出保持不變。
2. 把初始通道位址設定在 。
3. 執行以下步驟，經過 迭代。

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

圖 2‑5 Rho位移示意圖

表 2‑1 Rho 每個通道的位移量

一張含有 文字, 數字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

表 2‑2 Rho 每個通道的位移量 mod 64

一張含有 文字, 數字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 圖表, 行, 設計 的圖片

自動產生的描述

圖 2‑6 Rho 運算示意圖

### Pi()

運算式：

輸入：矩陣 A

輸出：矩陣 A’

將通道移到指定位置，使矩陣重新排列，如圖 2‑7 所示，由灰色(A)移到黃色(A’)位置。

一張含有 正方形, Rectangle, 設計 的圖片

自動產生的描述

圖 2‑7 Pi 運算示意圖

### Chi()

運算式：

輸入：矩陣 A

輸出：矩陣 A’

以「行」作為運算標準，與下一行與下下一行進行邏輯運算，每一行都要進行運算。

一張含有 鮮豔, 設計, 行, 寫生 的圖片

自動產生的描述

圖 2‑8 Chi 運算示意圖[[2]](#footnote-4)

### Iota()

運算式：

輸入：矩陣 A

輸出：矩陣 A’

中心 (0,0) 與回合常數（見回合常數）進行 XOR 運算。

一張含有 圖表, 設計, 行, 方案 的圖片

自動產生的描述

圖 2‑9 Iota(ι) 運算示意圖

**回合常數(Round Constant)**

將 64 位元(bits)的回合常數賦值給 24 個 wire 再傳入對應的 f\_sub 模組；

表 2‑3 Iota(ι) 運算 64 位元回合常數

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述

# 設計分析與考量

## 輸入資料與輸出資料的條件

### 輸入資料

Reset 將維持至少一個時脈週期，待 Reset 輸入為 0 時(T1)，Testbench 開始於 clk 為正緣時輸入明文資料，持續八個時脈週期，輸入完畢後 Testbench 停止所有輸入動作(T2)。

一張含有 文字, 圖表, 螢幕擷取畫面, 行 的圖片

自動產生的描述

圖 3‑1 系統初始時序圖

### 輸出資料

若在 clk 的正緣時 out\_valid 為 1(T3、T4、T5)，則表示當下的輸出訊號為有效指令，Testbench 會將此時的輸出資料與答案進行比對。

若在 clk 的正緣時 finish 為 1 (T5)，則表示此時有一組明文處理完畢，Testbench 會記錄此筆明文已輸出完畢並判斷程式是否需要結束運行。

一張含有 文字, 圖表, 螢幕擷取畫面, 行 的圖片

自動產生的描述

圖 3‑2 主控端訊號輸出時序圖

## Python 產生測資與演算法的實現

為了熟悉此加密雜湊函數的演算法，確保後續在撰寫測資的過程中能快速地根據需求生成足夠多的比對結果，以利後續硬體語言的實現，經過討論決定使用目前較為熟悉的 Python 來進行，將程式大致分為五個區塊處理：

1. 明文處理：將明文由 16 進位轉換成 2 進位，以位元組(byte)為單位進行翻轉。
2. 函式呼叫：翻轉完成後的明文輸入至各式加密雜湊函數，Python 實作 SHA3-128、SHA3-256、SHA3-512、SHAKE128、SHAKE256 五種加密雜湊函數。
3. 吸收：海綿結構中的吸收階段。
4. 擠出：海綿結構中的擠出階段。
5. 迭代函數 (Iteration function)：包含 Theta()、Rho()、Pi()、Chi()、Iota() 五個步驟。

由於在對 RTL 程式進行雜湊值驗證時是以擠出一段、比對一段的方式進行，因此在設計輸出運算結果的模式時，依照擠出的順序分開輸出，而不是一次輸出全部，如此便能直覺地將結果直接輸入至 Testbench 比對正確答案，以圖 3‑3 為例，將總輸出長度定為 3000 位元(bits)，需擠出三次才能完成，依序輸出結果。

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

圖 3‑3 SHAKE256 在 Python 中產生的測資結果

## 硬體設計

Top 模組(module)負責整合所有的模組，包含以下模組：

* Slice 模組：切割、儲存明文。
* Controller 模組：填充(Padding)、輸出控制訊號。
* F\_function 模組：運算迭代函數 。

最後把 F\_function 的輸出再經過翻轉後成為最終的輸出結果。

### Slice

Slice 模組會根據輸入明文的資訊，包含明文、加密演算法模式(SHA3-256、SHA3-512、SHAKE128、SHAKE256)、明文長度，用暫存器(Register)儲存，以輸入的模式得到轉換率(Bitrate)，並依照轉換率把明文切割成所需的長度，將切割好的明文串輸出到下一級模組(Controller)。

已知共有八筆明文，因此以八個時脈週期為一個循環不斷輸出不同明文(Message)的區塊 P。當輸出一次時會把存起來的明文長度減掉轉換率長度，若此刻存的明文長度小於轉換率時，會拉高特定訊號線，代表這筆明文全部吸收完畢，可以進入擠壓階段。

當明文長度恰好為轉換率倍數時，需要再多輸出一段長度為轉換率的零字串，因此，若輸出時儲存的明文長度剛好為轉換率大小，會用暫存器(Register)紀錄需不需要再輸出一次。

### Controller：填充(Padding)

1. 資料預處理：

運算 SHA-3 前，需先將明文翻轉。翻轉資料(Reverse data)的計算以位元組(Byte)為單位，且最左邊為 LSB。



圖 3‑4 Verilog 程式資料預處理

1. 填充至轉換率長度(r)

根據選擇的模式(Mode)進行不同的填充長度，補齊到對應的轉換率(Bitrate)，表 3‑1可知，填充能使用簡單的邏輯運算來實現。

表 3‑1 十六進位的 SHA-3 填充原則[[3]](#footnote-5) [[4]](#footnote-6)



填充規則分為 Hash 與 XOF 兩種，以 SHA3-512 與 SHAKE128 為例。

1. Hash 函數 (SHA3-512)：

* 0x80 進行左移，左移的次數為 SHA3-512 的轉換率(r=576)。
* 0x06 進行左移，左移的次數為輸入資料的長度
* 如圖 3‑5所示，處理後的兩個部分與輸入資料進行或運算(or)，讓 0x80 能位於轉換率的位置、0x06 能位於輸入資料的後方。

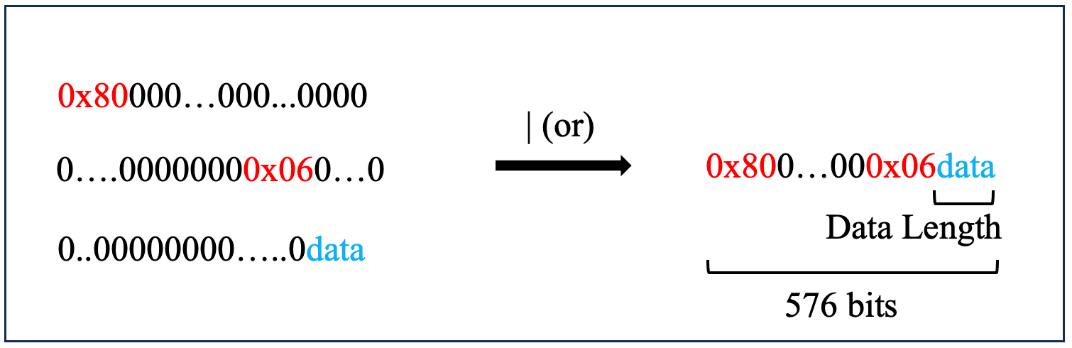


圖 3‑5 SHA3-512 填充至轉換率長度(r)

XOF函數(SHAKE128)

* 0x80進行左移，左移的次數為 SHAKE128 的轉換率(r=1344)。
* 0x1f進行左移，左移的次數為輸入資料的長度。
* 如圖 3‑6所示，處理後的兩個部分與輸入資料進行或運算(or)，讓 0x80 能位於轉換率的位置、0x1f 能位於輸入資料的後方。

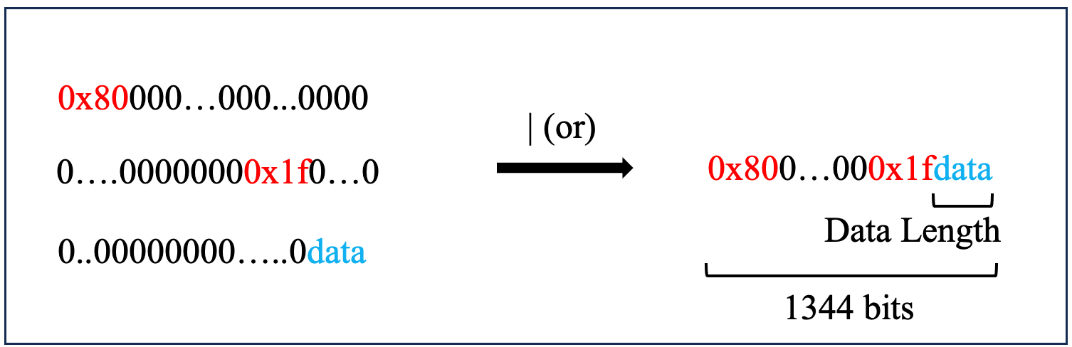


圖 3‑6 SHAKE128 填充至轉換率長度(r)

1. 填充至 KECCAK-寬度位元數(b)：

將結果補零到 1600 位元數(bits)，以符合迭代函數 的輸入條件。

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

圖 3‑7 轉換率(r)填充至 KECCAK-p 寬度位元數(b)

### F\_function： Pipeline 設計

已知共有八筆明文，為了能提高 SHA-3 處理資料的速度，我們將迭代函數 的 24 回合分成 8 次進行運算，每個時脈週期運算三個回合。

一張含有 文字, 行, 收據, 字型 的圖片

自動產生的描述

圖 3‑8 Pipeline 迭代函數 f 設計圖

## 設計版本迭代比較

1. 本研究中使用兩種版本的回合常數：

版本一為最初版本，見 Iota()。

版本二則是以版本一為基底將 64 位元的回合常數轉換至 8 位元，賦值給 24 個 wire 後再傳入對應的 f\_sub 模組，在 f\_sub 模組內，再回復成 64 位元的回合常數。

以回合 6 的常數為例，示範如何將 8 位元回合常數轉換為 64 位元回合常數。

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

圖 3‑9 八位元回合常數擴展過程

表 3‑2 Iota(ι) 運算 8 位元回合常數

一張含有 文字, 數字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

實作出這兩種版本後進行面積比較，發現這項改動對整體面積影響並不大。我們推測是由於電路中面積絕大部分是在處理與運算八筆明文的過程中產生，因此接著改良出第三個版本。

1. 版本三（最終版本）

原先設計中大量使用 input\_counter 當作選擇訊號對八組明文的資訊（如資料長度）進行儲存與輸出，如圖 3‑10 版本一、二的 Slice 暫存器示意圖，產生的面積非常龐大，約 72 萬 um2。

一張含有 文字, 圖表, 螢幕擷取畫面, 行 的圖片

自動產生的描述

圖 3‑10 版本一、二的 Slice 暫存器示意圖

發現問題後思考如何減少面積，最終想出圖 3‑11 的方法進行緩存。將資料在時脈正緣時從第 0 位輸入，根據時脈正緣逐漸傳遞至第 7 位輸出，且第 7 位再傳回第 0 位，如此不斷循環傳遞，且固定輸出第 7 位的資料。此架構大幅降低面積至 61.6 萬 um2。

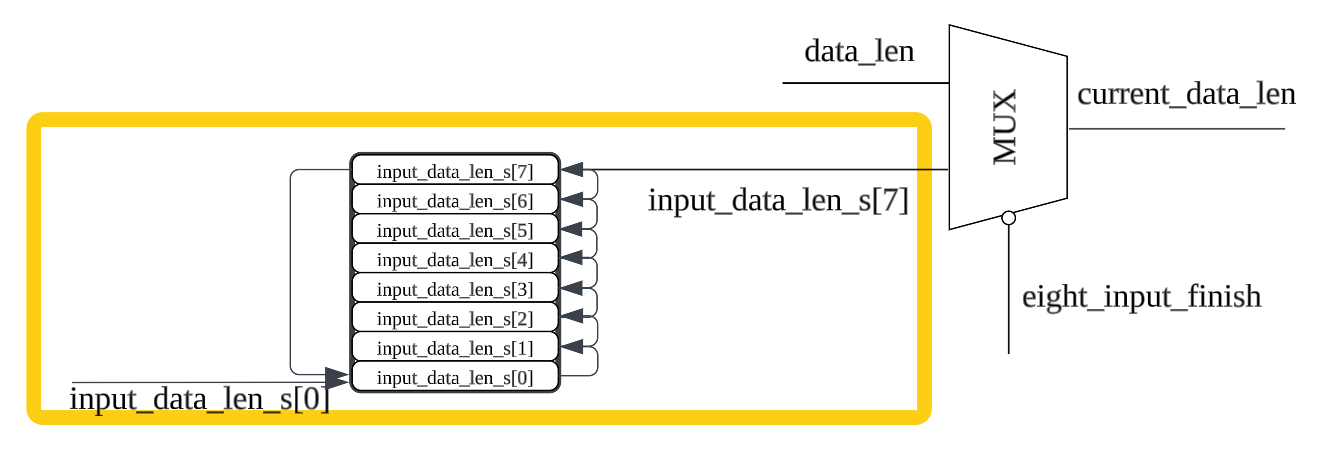


圖 3‑11 版本三的 Slice 暫存器示意圖

表 3‑3 三個版本的比較

|  |  |  |  |
| --- | --- | --- | --- |
|  | 版本一 | 版本二 | 版本三 |
| Total cell area (um2) | 723085.09831 | 722808.62909 | 616538.08892 |
| slack | 0.0 | 0.0 | 0.0 |

# 硬體架構

## 整體架構

本次實現 SHA-3 演算法的整體架構建立在 Top 模組中，如圖 4‑1 所示。首先，明文資料會傳入 Slice 模組，儲存並切割成相應轉換率的區塊 P 後，傳到下一級的 Controller 模組進行填充(Padding)，再送入 F\_function 模組中進行 24 回合的迭代函數運算。

一張含有 文字, 圖表, 行, 方案 的圖片

自動產生的描述

圖 4‑1 硬體架構圖

輸入：從測資輸入八個明文的資料，包括明文內容、加密模式、明文長度以及所求密文長度等等。

輸出：F\_function 採取 Pipeline 的作法，每隔八個時脈週期輸出同一個明文的雜湊值，每個時脈週期會根據明文的模式輸出相應轉換率長度的雜湊值，若全部輸出完畢，finish 訊號線會升起。

## 細部架構

### slice

用於儲存明文並將明文切割成轉換率長度，傳到下一級模組，如圖 4‑1所示。首先會把傳進來的明文存進八個暫存器，並依據傳入的模式來判斷要切割的轉換率長度，把切割好的區塊 P 傳送出去。

輸入：從 Top 模組傳來的各種明文資料，用暫存器把各種資訊儲存起來。

輸出：F\_function 模組採取 Pipeline 的做法，把 24 回合的運算拆成八級 3 回合的運算，所以 Slice 模組會以八個時脈週期為一個循環不斷輸入八筆不同的明文資料，讓八筆明文同時在 F\_function 模組內的 f\_sub 模組進行運算。輸入該筆明文的全部內容之後，part\_data\_in\_finish 訊號線會升起，表示明文已全部輸入完畢，可進行擠壓階段。

一張含有 文字, 圖表, 行, 字型 的圖片

自動產生的描述

圖 4‑2 Slice 模組硬體架構圖

表 4‑1 Slice 模組接腳功能定義表

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | I/O | Bit width | Description |
| clk | I | 1 | Positive edge-triggered clock signal |
| rst | I | 1 | Active high asynchronous reset |
| mode\_in | I | 2 | Input data mode |
| data\_in | I | 6400 | Message |
| data\_len | I | 13 | Message length |
| in\_finish | I | 1 | Current input message is input completely |
| rate\_out | O | 11 | Bitrate of output part data |
| part\_data | O | 1344 | Result of slicing message |
| part\_data\_len | O | 11 | Length of part data |
| mode\_out\_slice | O | 2 | Mode of part data |
| part\_data\_in\_finish | O | 1 | Whether current message finish input |

### controller

將傳進來的區塊 P 進行填補(Padding)並傳到下一級模組(F\_function)，同時輸出控制訊號 absorb\_or\_squeeze(判斷現在是吸收階段還是擠出階段)、zero\_or\_fout(此刻輸入的明文串與零或是 F\_function 輸出的東西做 XOR 運算)。

一張含有 文字, 圖表, 字型, 數字 的圖片

自動產生的描述

圖 4‑3 Controller模組訊號線圖

表 4‑2 Controller模組接腳功能定義表

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | I/O | Bit width | Description |
| clk | I | 1 | Positive edge-triggered clock signal |
| rst | I | 1 | Active high asynchronous reset |
| mode\_in | I | 2 | Input data mode |
| data\_in | I | 1344 | Sliced message without padding |
| data\_len | I | 11 | Input data length |
| length | I | 13 | Output data length (only XOF can be specified) |
| in\_finish | I | 1 | If all slices of current message have been inputted, in\_finish will be one. |
| rate\_in | I | 11 | Input data’s bitrate |
| data | O | 1600 | Sliced message with padding |
| mode\_out\_controller | O | 2 | Output data mode |
| out\_valid | O | 1 | Current sliced message is squeezed and this output is valid |
| finish | O | 1 | Final sliced message is completely squeezed |
| out\_length | O | 11 | Output data length |
| absorb\_or\_squeeze | O | 1 | Current stage (absorb:0, squeeze:1) |
| zero\_or\_fout | O | 1 | Data\_in is XOR 0 string(0) or XOR f\_out(1) |

### F\_function

此模組實作了 24 回合的五個角度函數運算。一個 f\_sub 模組完成一回合的運算，以三個 f\_sub 模組為一級，共分八級來運算 24 回合，每一級中間用一個暫存器來儲存運算值。運算後，會將計算結果輸出到下一級模組。

一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述

圖 4‑4 F\_function 模組硬體架構圖

表 4‑3 F\_function 模組接腳功能定義表

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | I/O | Bit width | Description |
| clk | I | 1 | Positive edge-triggered clock signal |
| rst | I | 1 | Active high asynchronous reset |
| F\_input | I | 1600 | Input data of F function |
| F\_output | O | 1600 | Output data of F function |

### f\_sub

此模組的功能為實作一回合的五個角度函數運算。傳入的 8 位元的回合常數(訊號 rc )會經過補零的方式轉成實際要使用的 64 位元 rc\_wire。

一張含有 文字, 行, 圖表, 字型 的圖片

自動產生的描述

圖 4‑5 f\_sub 模組硬體架構圖

表 4‑4 f\_sub 模組之接腳功能定義表

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | I/O | Bit width | Description |
| absorb\_outcome | I | 1600 | Input data |
| rc | I | 8 | Round constant |
| s\_out | O | 1600 | Output data |

# 模擬結果

本篇專題合成為使用 Design Compiler 在 TSMC 40nm 製程環境下進行，以下為 RTL 模擬（Register-Transfer Level, 暫存器傳輸級）、邏輯合成（Logic Synthesis）兩個階段的模擬資料。

## RTL 模擬波形圖

### Slice

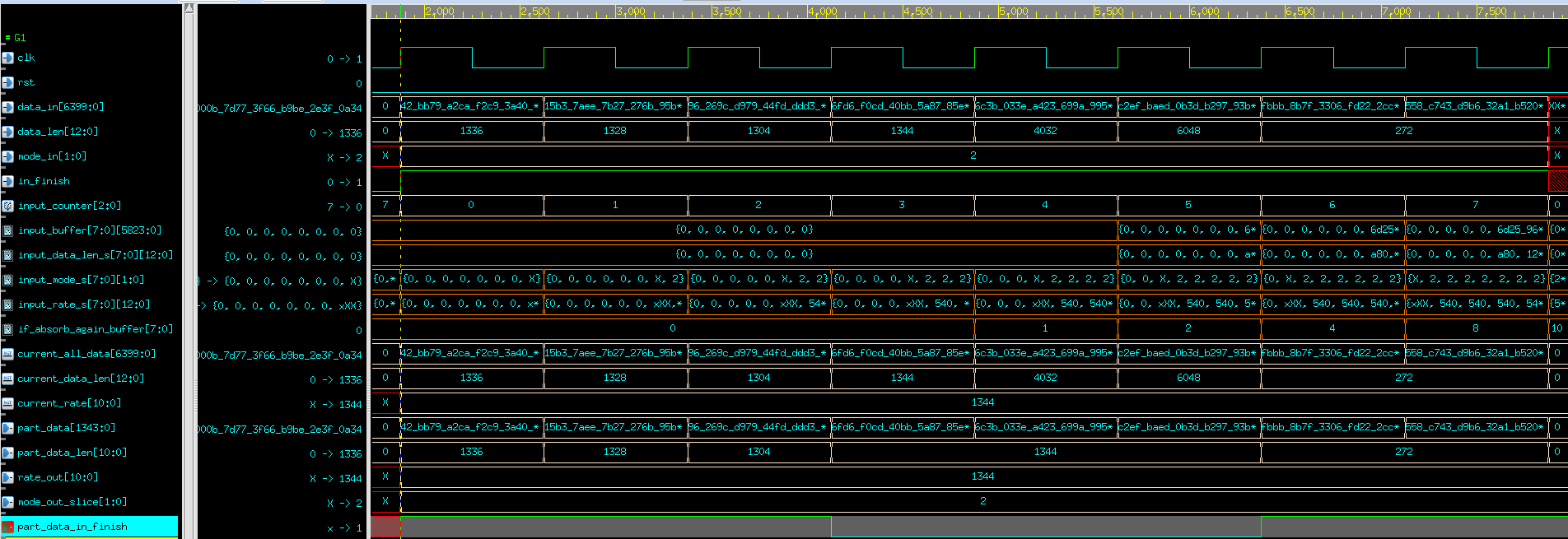


圖 5‑1 Slice 模擬波形圖之一

1. 明文長度 data\_len 小於轉換率(Bitrate)：

如果明文長度 data\_len 小於轉換率(Bitrate)，代表輸入一次明文即可，不用把明文存起來，並且將輸入的明文內容(part\_data)、長度(part\_data\_len)、模式(mode\_out\_slice)都傳到下一級，表示明文傳送完畢的訊號(part\_data\_in\_finish)也會升起來。

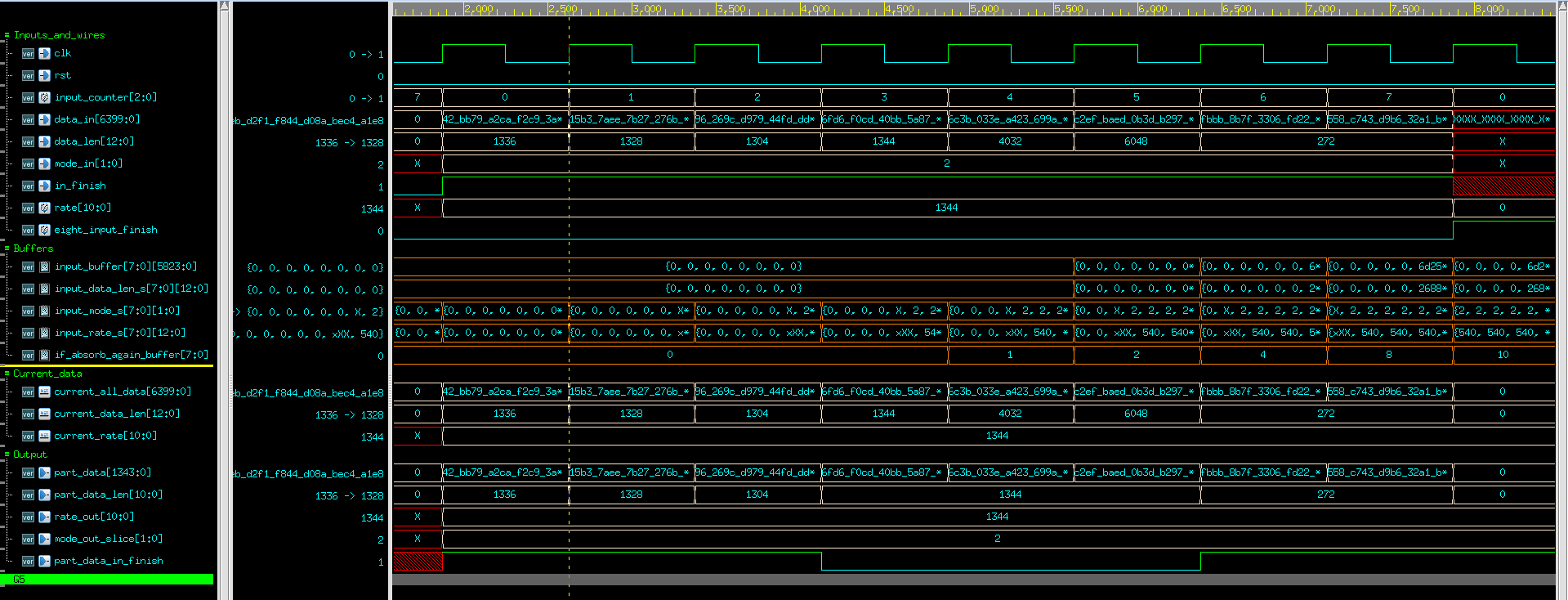


圖 5‑2 Slice 模擬波形圖之二

2. 明文長度 data\_len 與轉換率相等：

如果明文長度 data\_len 與轉換率相等，除了第一次傳送明文內容後，還要多吸收一次全為零的字串，if\_absorb\_again\_buffer[0]會變成 1，提醒這筆明文還需要再吸收一次，傳完明文的訊號(part\_data\_in\_finish)還不會升起來。

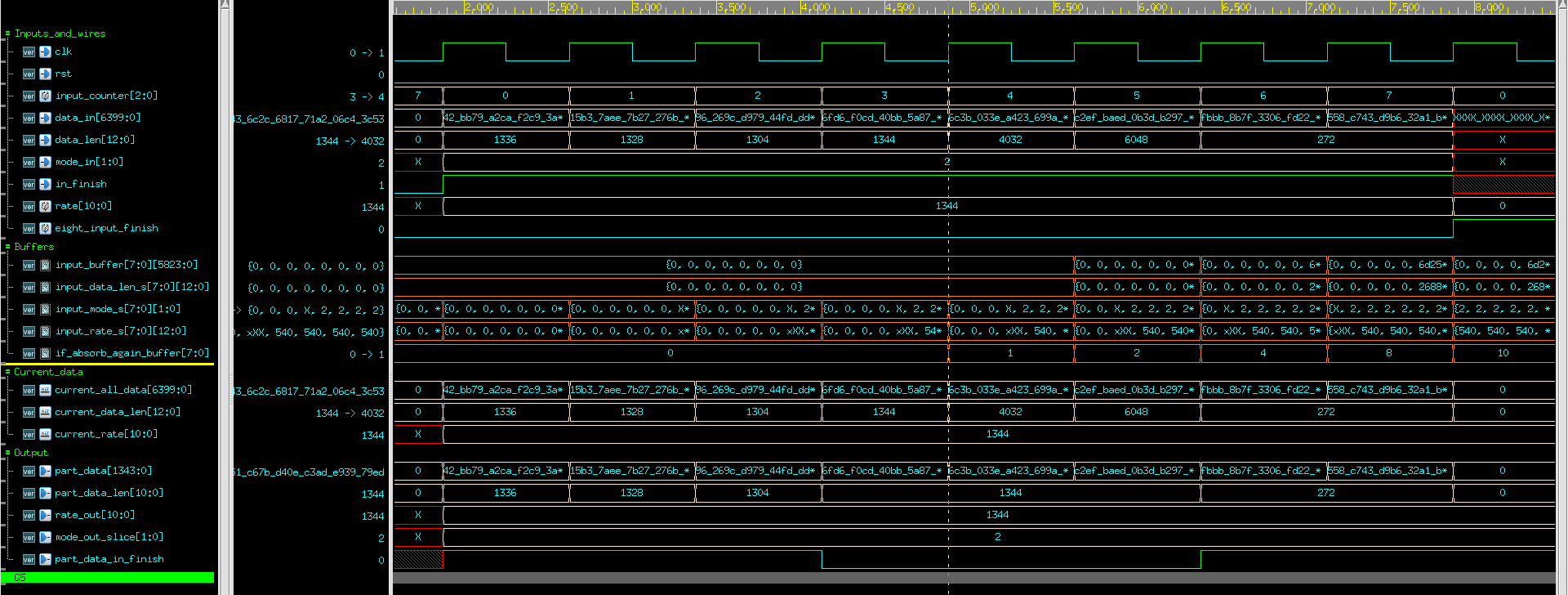


圖 5‑3 Slice 模擬波形圖之三

下一次輸出全部為 0 的字串後，該筆明文的訊號 if\_absorb\_again\_buffer 會變成 0，表示輸出完畢，明文傳送完畢的訊號(part\_data\_in\_finish)會升起來。



圖 5‑4 Slice 模擬波形圖之四

3. 明文長度 data\_len 大於轉換率：

如果明文長度 data\_len 大於轉換率，將剩下沒傳完的明文存起來(input\_buffer)。並把剩下的明文長度[[5]](#footnote-7)存進 data\_len\_s[0]。

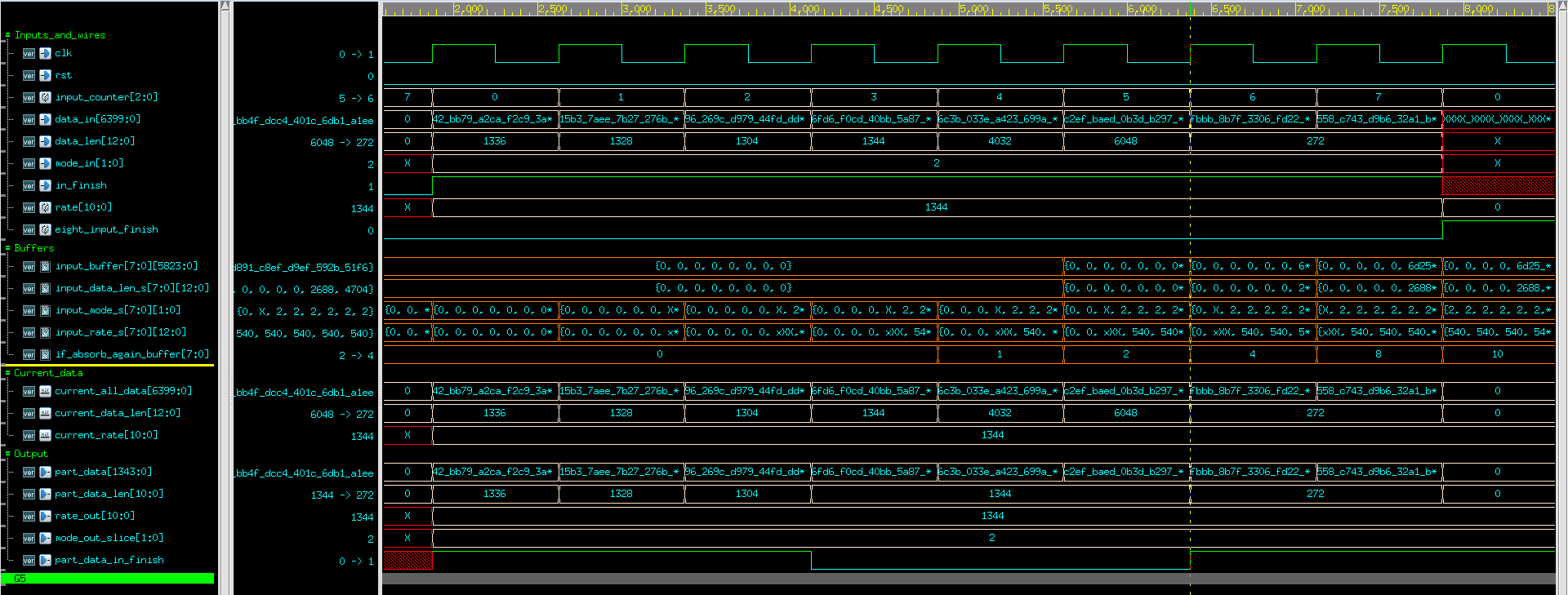


圖 5‑5 Slice 模擬波形圖之五

等傳最後一次時，data\_len\_s (672 bits) 小於轉換率 (1344 bits)，data\_len\_s 會歸零，傳完明文的訊號(part\_data\_in\_finish)也會升起。

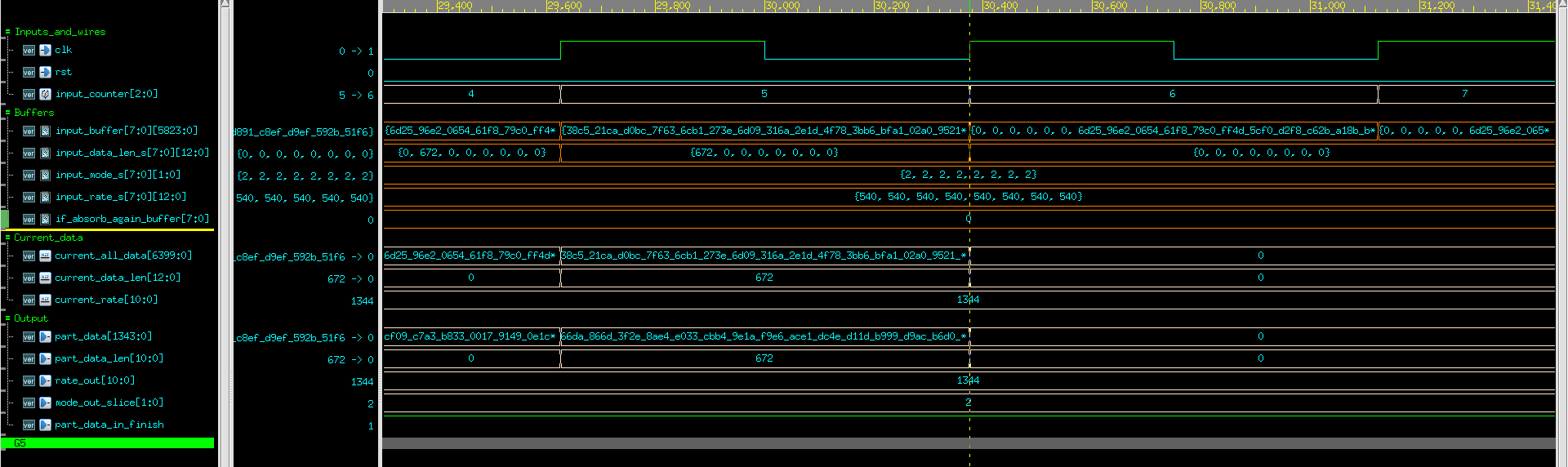


圖 5‑6 Slice 模擬波形圖之六

4. 八筆明文輸入完，訊號 eight\_input\_finish 會升起。

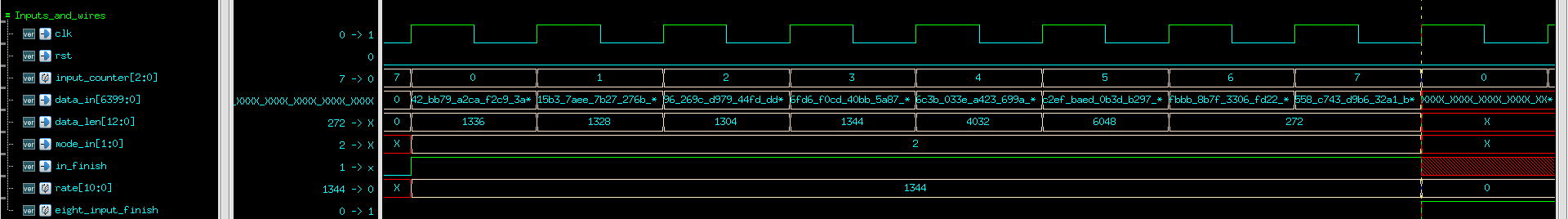


圖 5‑7 Slice 模擬波形圖之七

### Controller

1. 一開始把八個所求的密文輸出長度(length)存起來(out\_length\_buffer)

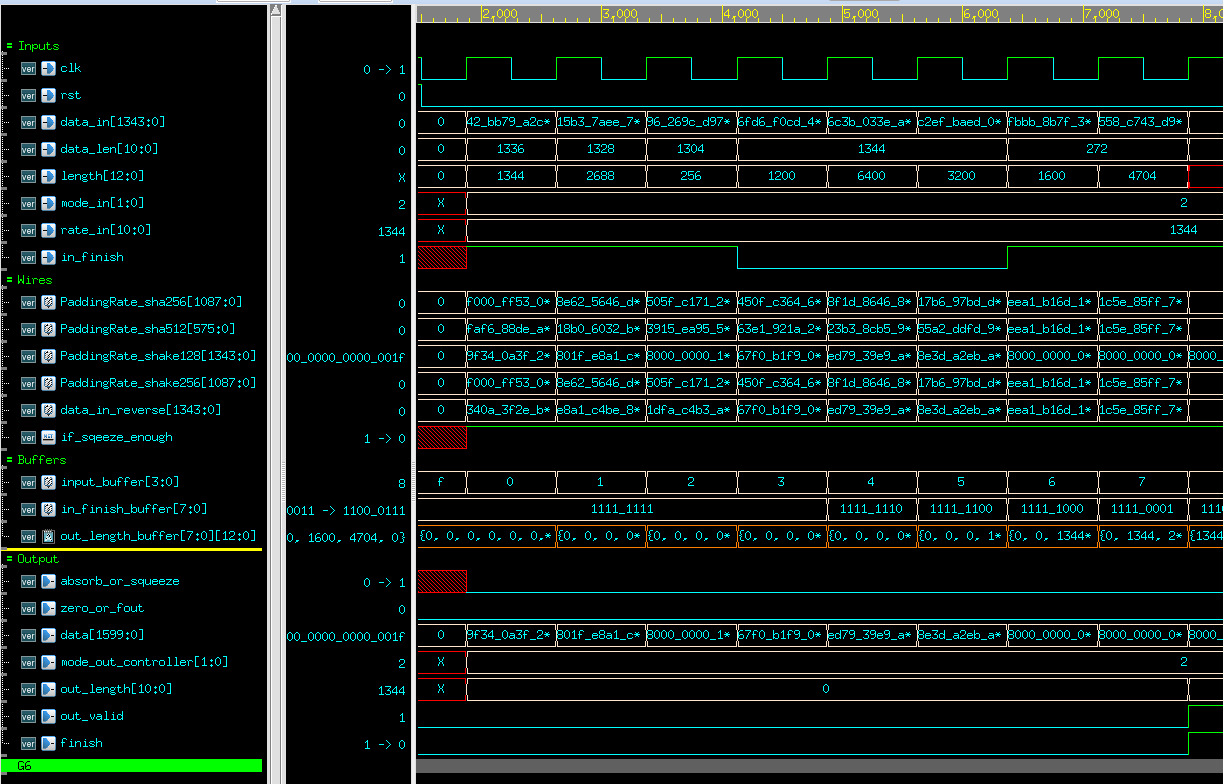


圖 5‑8 Controller 模擬波形圖之一

2. 輸出長度大於轉換率[[6]](#footnote-8)

輸出長度不夠，此刻的 out\_length\_buffer(2688 位元)大於轉換率(1344 位元)，需繼續擠出，表示此次輸出有效的訊號(out\_valid)升起，但代表所有密文輸出完畢的 finish 訊號仍為 0。

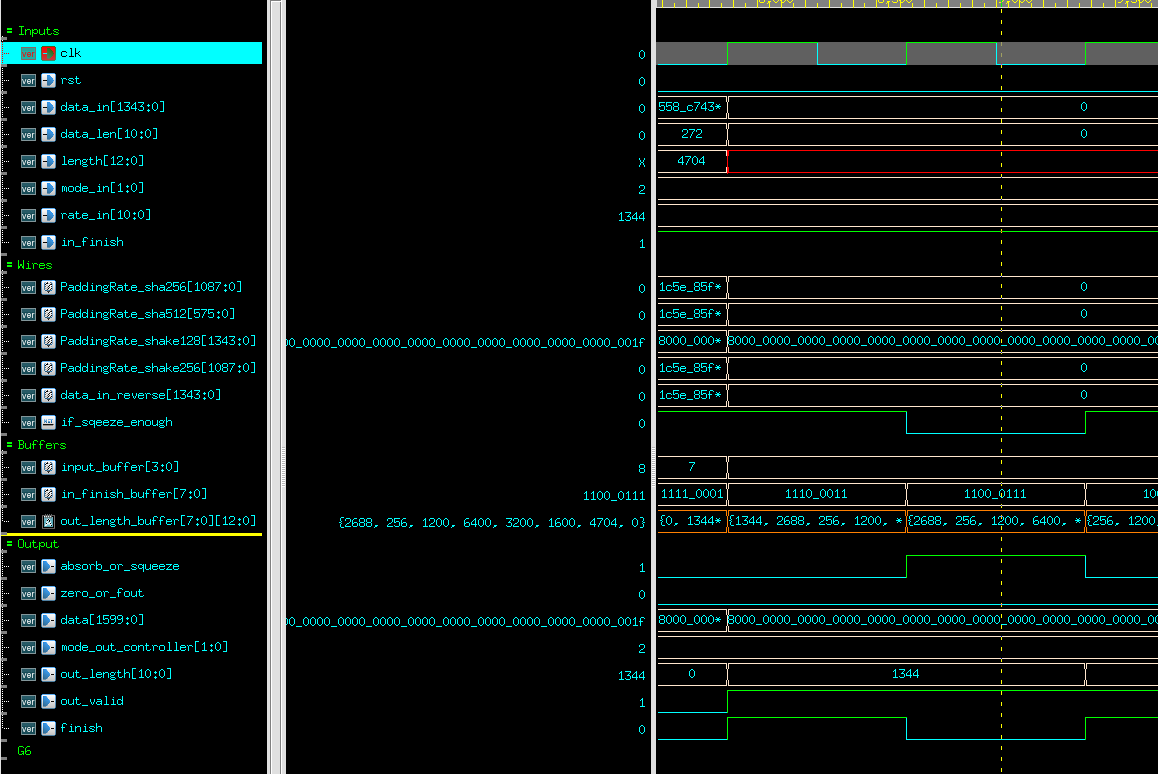


圖 5‑9 Controller 模擬波形圖之二

3. 輸出長度足夠(out\_length\_buffer rate)：

(1)當輸出長度足夠時，out\_valid 和 finish 皆會升起，out\_length\_buffer(1344 位元)等於 rate(1344位元)，代表此次擠出後運算即全部結束。此次輸出有效的訊號(out\_valid)升起，而代表所有密文輸出完畢的 finish 也在此時升起。

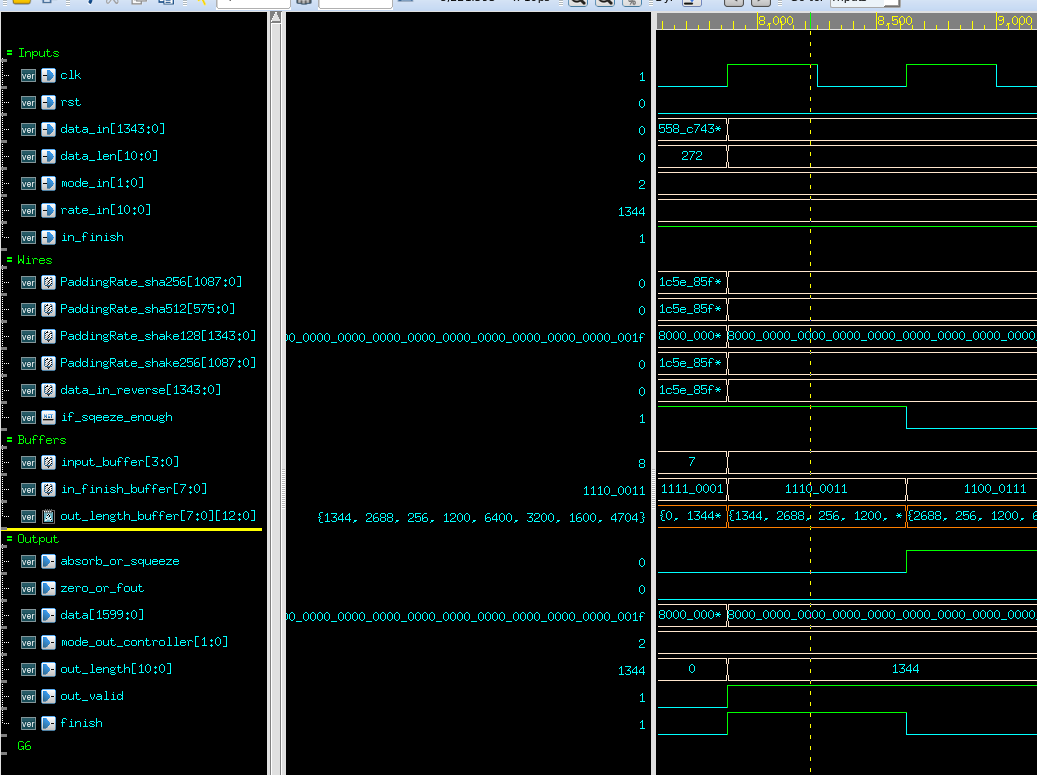


圖 5‑10 Controller 模擬波形圖之三

(2)輸出長度小於轉換率時，out\_length\_buffer(256 位元)小於 rate (1344位元)，代表擠出後運算即全部結束。此次輸出有效的訊號(out\_valid)升起，而表示所有密文輸出完畢的 finish 也在此時升起。

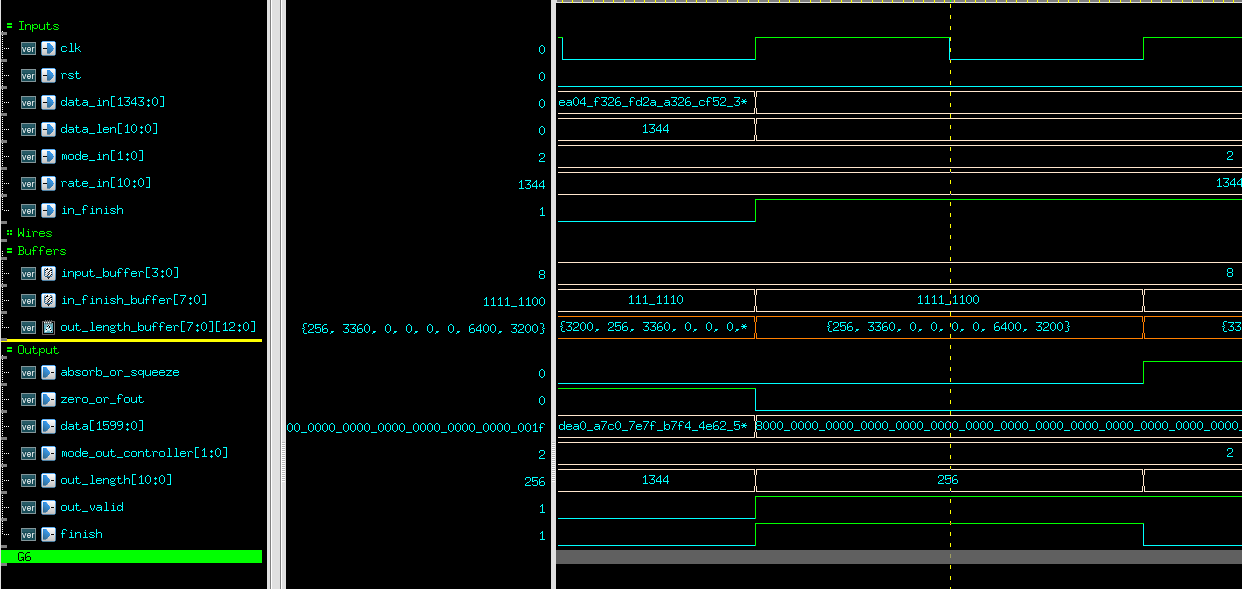


圖 5‑11 Controller 模擬波形圖之四

5. 填充與翻轉

明文內容輸入至 data\_in，而 data\_in\_reverse(wire)為 data\_in 翻轉後的資料，再經過填充後，得到 PaddingRate\_sha256, PaddingRate\_sha512, PaddingRate\_shake128, PaddingRate\_shake256，再依模式來決定要輸出哪一個填充後的值。

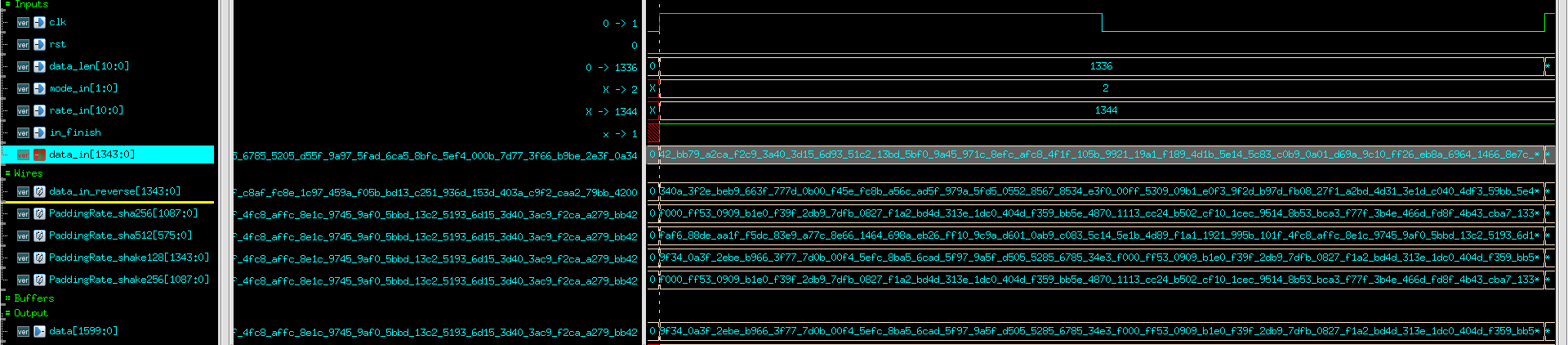


圖 5‑12 Controller 模擬波形圖之五

### F\_function

* F\_input 為 F\_function 模組一個狀態矩陣的輸入，此狀態矩陣已被處理成 1600 位元的一維矩陣。
* const 為 24 個回合常數的二維 wire，將每個回合常數傳入對應的 f\_sub 模組。
* 每一個 f\_wire 都是承接對應 f\_sub 模組的輸出，再傳入下一個 f\_sub 模組。每三個 f\_wire 為一組，在一個時脈週期內運算結束後，會把結果在下一個時脈週期傳入對應的 f\_buffer。
* f\_wire 傳入 f\_buffer 的同一個時脈週期中，f\_buffer 傳入下一組 f\_sub，以此類推，做 8 組後輸出。如圖 5‑13 所示，同樣顏色的 f\_wire 和 f\_buffer 為一組，總共有 8 組，第 8 個時脈週期時，f\_buffer[7]會賦值給 F\_output 後輸出。



圖 5‑13 F\_funtion 模擬波形圖

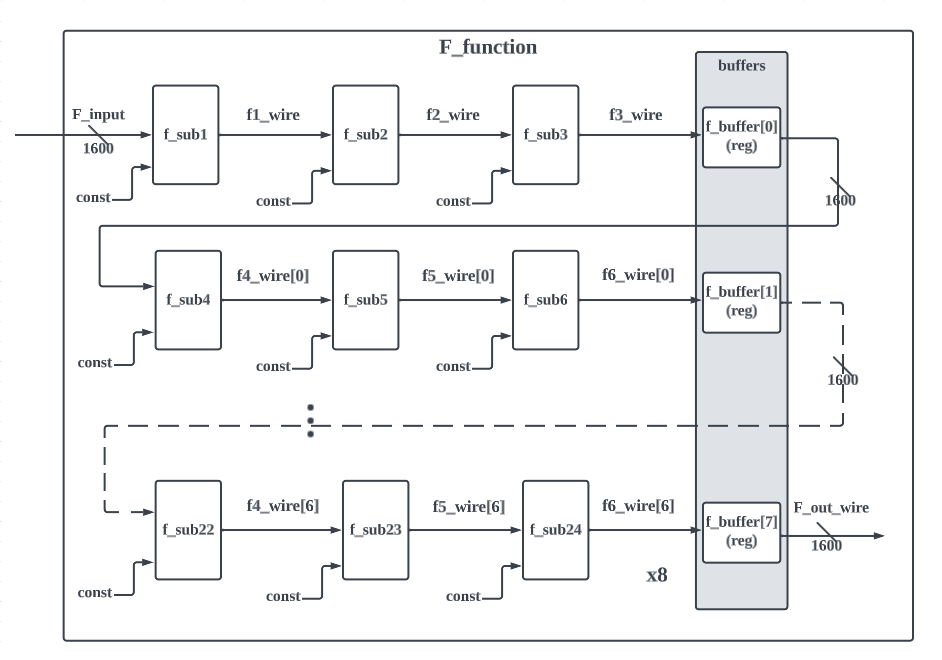


圖 5‑14 F\_funtion 流程簡圖

### f\_sub

* absorb\_outcome 為 f\_sub 模組一個狀態矩陣的輸入，此狀態矩陣已被處理成 1600 位元的一維矩陣。
* rc 為 8 位元回合常數的輸入，會於 f\_sub 模組內轉換成 64 位元的回合常數並賦值給 rc\_wire。
* 之前解釋過 f\_sub 的運算方式，會用到其中的 reg 矩陣[[7]](#footnote-9)，最後再將結果賦值到 out 並接到 s\_out 輸出，以上運算均是在一個時脈週期內完成。

一張含有 螢幕擷取畫面, 多媒體軟體, 軟體, 繪圖軟體 的圖片

自動產生的描述

圖 5‑15 f\_sub 模擬波形圖

## RTL 模擬

RTL 模擬使用 Testbench 於 NC-Verilog 跑的結果，僅作為功能上的模擬。用訊號模擬電路運作情況，並觀察波形圖與結果是否正確，不考慮真實電路的運作情況(例如：合成後遇到的 Timing Violation 問題)。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

圖 5‑16 RTL 模擬之 PIPE 測資

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

圖 5‑17 RTL 模擬之 SHA3-256 測資

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

圖 5‑18 RTL 模擬之 SHA3-512 測資

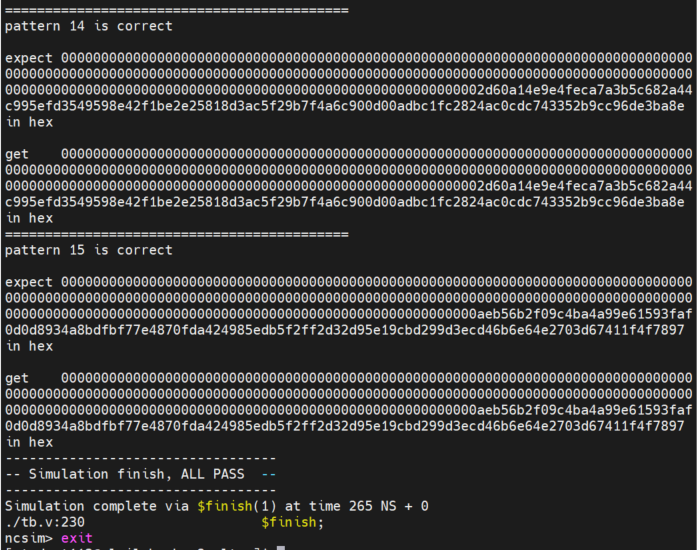


圖 5‑19 RTL 模擬之 SHAKE128 測資

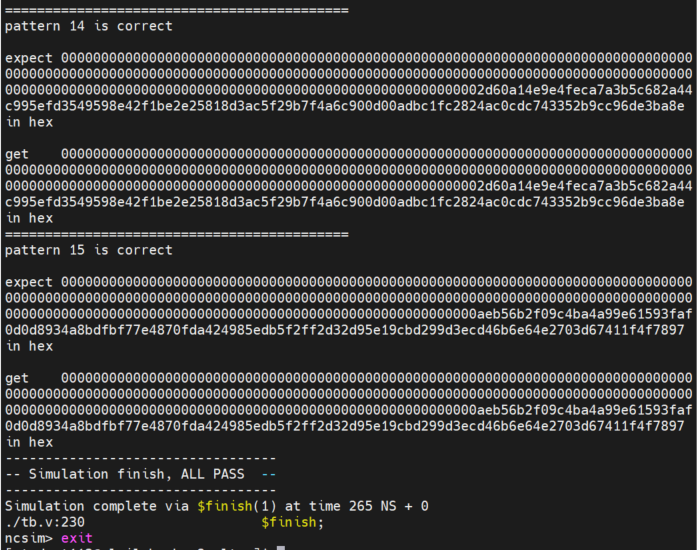


圖 5‑20 RTL 模擬之 SHAKE256 測資

## Synthesis

專題中利用 compile\_ultra 與 compile\_ultra -incremental 指令在 TSMC 40nm 製程的環境下，使用 Design Compiler 將 RTL 的程式碼轉成 Gate-Level 邏輯架構的 Netlist 檔以及產生標準時間延遲檔(.sdf)，同時模擬出電路大致面積。

我們利用繪製 PA[[8]](#footnote-10) 曲線找出合成後可正確執行且 PA 值最低的時脈週期為 6.5 ns，如下表。

表 5‑1 PA 數據彙整

|  |  |  |  |
| --- | --- | --- | --- |
| Clock(ns) | Area(um2) | Post-synthesis time (PS)[[9]](#footnote-11) | PA |
| 6.0[[10]](#footnote-12) | 613562.019319 | 111000 | 68105384140 |
| 6.5 | 615489.365764 | 120250 | 74012596230 |
| 7.0 | 614452.662892 | 129500 | 79571619840 |
| 7.5 | 596560.411749 | 138750 | 82772757130 |
| 8.0 | 614366.705714 | 148000 | 90926272450 |

表 5‑2 PA 曲線

## Gate-Level 模擬

透過 Netlist 檔與標準時間延遲檔(.sdf)進行 Gate-Level 模擬來檢驗合成後的電路是否存在時序問題(Timing Violation)以及功能是否仍運作正確。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

圖 5‑21 Gate-Level 模擬之 PIPE 測資

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

圖 5‑22 Gate-Level 模擬之 SHA3-256 測資

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

圖 5‑23 Gate-Level 模擬之 SHA3-512 測資

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

圖 5‑24 Gate-Level 模擬之 SHAKE128 測資

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

圖 5‑25 Gate-Level 模擬之 SHAKE256 測資

## Throughput

吞吐量( throughput )計算公式如下所示：

r：轉換率 (Bitrate)

f：頻率 (frequency)，最佳時脈週期之倒數。

Cycle：運算一次雜湊函數需要的時脈週期數

表 5‑3 四種模式 Thoughput 結果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | SHA3-256 | SHA3-512 | SHAKE128 | SHAKE256 |
| Bitrate(bit) | 1088 | 576 | 1344 | 1088 |
| Max frequency(MHz) | 153.8 | 153.8 | 153.8 | 153.8 |
| Cycle | 1 | 1 | 1 | 1 |
| Throughput(Gbps) | 167.3344 | 88.5888 | 206.7072 | 167.3344 |

# 結論與未來展望

本研究用 Verilog 硬體語言實作 Keccak 加密演算法，分別有 SHA3-256、SHA3-512、SHAKE128、SHAKE256 ，若是 SHAKE128 與 SHAKE256 可以指定要輸出的雜湊值長度。為了提升吞吐量(Throughput)，利用 Pipeline 的方式設計迭代函數，同時讓八組不同明文進行計算，並持續在每個時脈週期輸出不同明文的雜湊值，讓輸出更有效率。

完成的電路使用 NC-Verilog 進行 RTL 與 Gate-Level 模擬與驗證，利用 Design Compiler 在時脈週期為 6.5 ns 與 TSMC 40nm 的製程下進行合成。最初實現 Keccak 加密演算法功能時面積約 720,000，經過閱讀多篇論文與資料查找，以優化演算法與更改硬體架構的方式將面積降至約 610,000。

我們計畫在接下來幾個月更改目前的電路架構，將 Keccak 加密演算法的輸入資料在每個時脈週期正緣輸入改為事先存在記憶體當中，利用控制模組(controller module) 來決定輸入的明文資料，這種寫法可以避免電路中須利用大量暫存器來儲存輸入資料。完成後透過 memory compiler，在 TSMC 40nm 的製程下進行合成、產出整體的面積，與目前版本比較後選擇出擁有較優良面積與時間的版本。最後，透過 IC compiler 進行晶片實體設計與驗證，完整地進行一次晶片設計流程。

# 參考資料

|  |  |
| --- | --- |
| [1] | M. M. W. *et. al*., "A New High Throughput and Area Efficient SHA-3 Implementation," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, 2018. |
| [2] | A. D. *et. al*., "Hardware architecture for CRYSTALS-Kyber post-quantum cryptographic SHA-3 primitives," in *2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, Valencia, 2023. |
| [3] | G. B. *et. al*., "Team Keccak," keccak.team, 2008-2023. [Online]. Available: https://keccak.team/index.html. [Accessed 2023]. |
| [4] | joostrijneveld, *xmss-reference,* joostrijneveld, 2017. |
| [5] | W. May, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,* Gaithersburg: NIST, 2015. |
| [6] | C.-Y. Hung, *The Design and Verification of an IP for Secure Hash Algorithm SHA-3 with AXI-4 Interface,* Taipei: National Digital Library of Theses and Dissertations in Taiwan, 2021. |

# 分工表

許多部份是一起討論概念後再分工，會互相幫忙優化、除錯或解決困難，以下是主要負責的部分。

|  |  |
| --- | --- |
| 姓名 | 負責工作 |
| 鄭智宇 | 組長、Python、top 模組、F\_function 模組設計及優化、TCL、書面報告 |
| 宋方瑜 | 產生 test pattern、controller 模組、slice 模組設計及優化、書面報告 |
| 陳慕丞 | f\_sub 模組、F\_function 模組設計及優化、寫 f\_sub 的 testbench、合成、書面報告 |
| 許庭毓 | 書面報告撰寫與統整、海報製作、簡報製作 |

1. 需要被加密的訊息 [↑](#footnote-ref-1)
2. 以左一行的運算為例的示意圖 [↑](#footnote-ref-4)
3. q 為填充轉換率(r)時，需要的位元組(Byte)數。 [↑](#footnote-ref-5)
4. || 為連接的意思 [↑](#footnote-ref-6)
5. [↑](#footnote-ref-7)
6. 仍需輸出：out\_valid 升起，finish 沒升起 [↑](#footnote-ref-8)
7. Reg 矩陣: a,a1,b,d,e [↑](#footnote-ref-9)
8. PA: Performance \* Area，此處 Performance 為 Post-synthesis time。 [↑](#footnote-ref-10)
9. Post-synthesis time :我們統一用運行 SHA512 測資的時間。 [↑](#footnote-ref-11)
10. 時脈週期為 6.0 ns 時會發生 Timing Violation，即沒有通過 Gate-Level 模擬 [↑](#footnote-ref-12)