國立成功大學 電機工程學系

畢業專題實作成果報告書

雜湊演算法

MD4/SHA-1/SHA-256

硬體加速器實現

Hardware accelerator Implementation of Hash Algorithms MD4/SHA-1/SHA-256

專題組別：■電路與系統 □電子與材料 □電腦與通訊

指導教授： 邱瀝毅 教授

組員學號&姓名：E24096433 翁梓薰、E24092049 林伯璨

研究期間： 112 年 6 月 至 112 年 10 月底止，計 4 個月

摘要

本專題將採用硬體加速器的方式，以提升雜湊演算法的執行效能和效率。透過分析雜湊演算法的運算原理，將其繪製成資料流程圖。根據資料流程圖，我們能先利用數學定義出優化的目標，再用合理的優化方法實現一個高效且低功耗的硬體架構，能夠加速雜湊演算法 MD4、SHA-1 和 SHA-256 的運算。

在將電路以 0.18umCMOS 製程合成之後，驗證結果的正確性並且進行電路的效能分析與比較。我們所提出的三種雜湊演算法電路架構皆能得到 1Gbps 以上的數據吞吐量，並且與其他論文的硬體架構相比我們的架構在速度上也有優勢。

關鍵字

MD4、SHA-1、SHA-256

Abstract

This project will employ hardware acceleration to enhance the execution efficiency and effectiveness of hashing algorithms. By analyzing the computational principles of hashing algorithms, they will be depicted as data flow diagrams. Based on these diagrams, optimization objectives will be mathematically defined, followed by the implementation of a high-performance and low-power hardware architecture using appropriate optimization methods. This architecture aims to accelerate the computations of hashing algorithms MD4, SHA-1, and SHA-256.

After synthesizing the circuits using the 0.18um CMOS process, the correctness of the verification results will be ensured, and an analysis and comparison of circuit performance will be conducted. The proposed circuit architectures for the three hashing algorithms can all achieve data throughputs exceeding 1 Gbps. Furthermore, when compared to hardware architectures from other research papers, our architectures exhibit superior speed performance.

Keyword

MD4、SHA-1、SHA-256

目錄

一、前言

本專題報告書旨在探討雜湊演算法（MD4、SHA-1、SHA-256）硬體加速器的實現及其在數據安全和效能優化方面的應用。本報告將提供有關這一專題的研究背景、問題說明、研究目的、方法和創新點的詳細信息。

- **研究目的**

近年來，資訊安全的重要性日益突顯，尤其是隨著物聯網時代的來臨，人們對於資料的安全性和完整性的要求也越來越高。在資訊安全中，雜湊演算法扮演著重要的角色，它們能夠將任意大小的輸入資料轉換成固定大小的雜湊值，並且具有碰撞難度大、不可逆等特點。在這次專題中我們希望能透過實現更高速的 MD4、SHA-1 和 SHA-256 雜湊演算法硬體加速器，以提高數據安全應用和效能優化的能力。

- **問題說明**

目前，許多複雜的安全應用和數據處理任務需要高效的雜湊演算法，這對傳統的軟體實現方式提出了挑戰。因此，我們面臨的主要問題是如何設計和實現硬體加速器，以加速 MD4、SHA-1 和 SHA-256 演算法的運算，同時確保數據的安全性。

- **問題解決方法**

本研究會先透過 Retiming、Unfolding transformation 等數位電路優化技術[1]，優化電路架構，再利用 Python、硬體描述語言（HDL）和相關硬體設計工具協助設計並驗證電路正確性。最後透過合成軟體幫助我們實現 MD4、SHA-1 和 SHA-256 的硬體加速器，隨後更進一步進行性能測試和效能分析，以佐證這些硬體加速器的效能。

- **實作結果**

本報告將詳細介紹我們的硬體實現結果，包括設計細節、優化技巧、性能評估、和實驗結果。我們的研究將提供實際證據來支持我們的方法和設計。從合成軟體提供的數據及模擬結果能得知我們的三種雜湊演算法電路設計都實現了每秒 1Gb 以上的數據處理速度，遠遠超過軟體的數據吞吐量。此外，與其他研究中提到的硬體設計相比，我們的架構在速度方面也具有競爭優勢。

我們期望我們的研究結果能夠提高現有數據處理系統的效率和安全性，這對於日益增長的數據處理需求以及數據安全的重要性都具有積極影響。這些高效且安全的雜湊演算法硬體實現為現代數位應用領域帶來了新的可能性，有望改進許多關鍵領域的性能和可靠性。

二、原理分析與系統設計

2.1 原理分析

➢ 雜湊演算法 (Hash Algorithm)

雜湊演算法是一種數學函數，將任意大小的輸入資料轉換成固定長度的雜湊值。這個雜湊值通常是一個固定大小的二進制數字串，稱為雜湊碼。

雜湊演算法的主要功能是將輸入資料轉換為唯一的、不可逆的雜湊碼，並且相同的輸入將始終產生相同的雜湊碼。這使得雜湊演算法在數據存儲、數據庫檢索、數位簽名、密碼學和數據完整性驗證等應用中非常有用。雜湊演算法的目標是快速計算，並確保即使輸入數據稍微變化，雜湊碼也會大不相同。

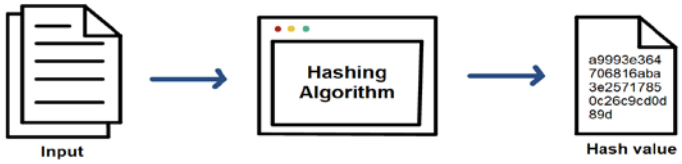


圖 2-1 雜湊示意圖

➢ MD4 雜湊演算法

MD4 演算法於 1990 年由 Ronald Rivest 設計，其運作原理如下[2]：

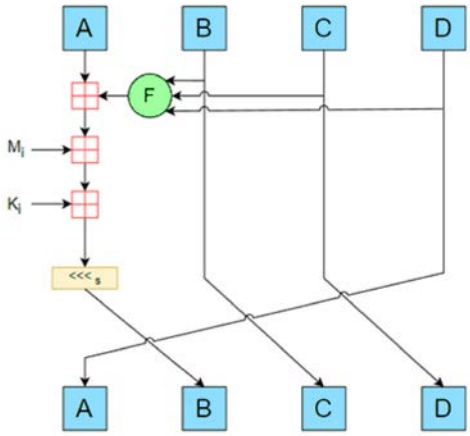


圖 2-2 MD4 演算法

1. 將 512 位元的輸入訊息以每 32 位元切割成 16 組並存入 Mi。

2. 賦予一個 128 位元的初始雜湊值（A、B、C、D）到暫存器中。

$$A = 01234567$$
$$B = 89abcdef$$
$$C = fedcba98$$
$$D = 76543210$$

圖 2-3 初始化數值(hex)

3. 輸入訊息與雜湊值會經由三種類型的運算：加法運算、位元輪轉運算和非線性函數 F。共執行 48 輪的運算，運算過程又以每 16 輪為一組分為以下三大組:

$$F(B,C,D) = (B \wedge C) \vee \big((\sim B) \wedge D\big)$$
$$i = 0, 1, \ldots, 15$$
$$\mathrm{K_i} = 0$$
$$s = 3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19$$

圖 2-4 第一組運算函式與參數

$$F(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$$
$$i = 0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15$$
$$\mathrm{K_i} = 5A827999$$
$$s = 3,5,9,13,3,5,9,13,3,5,9,13,3,5,9,13$$

圖 2-5 第二組運算函式與參數

$$F(B,C,D) = B \oplus C \oplus D$$
$$i = 0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15$$
$$\mathrm{K_i} = 6ED9EBA1$$
$$s = 3,9,11,15,3,9,11,15,3,9,11,15,3,9,11,15$$

圖 2-6 第三組運算函式與參數

4. 最後，將暫存器中的值(A、B、C、D)加上其初始雜湊值，形成最終的 128 位元雜湊值。

➢ SHA-1 雜湊演算法

SHA-1 演算法於 1995 年由美國國家安全局設計，其運作原理如下[3]：

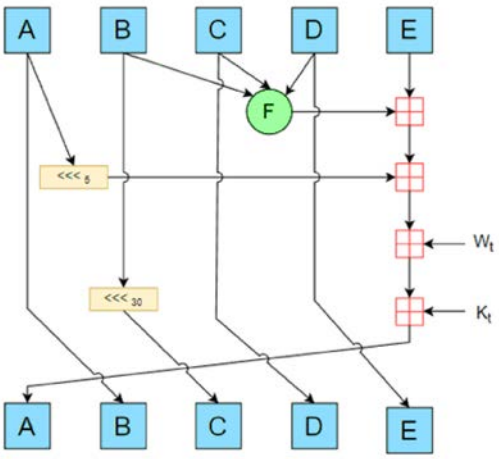


圖 2-7 SHA-1 演算法

1. 將輸入訊息進行填充和分割，形成 512 位元的輸入訊息。再以每 32 位元切割成 16 組並存入 Wt。

2. 賦予一個 160 位元的初始雜湊值（A、B、C、D、E），並存入暫存器。

$$A = 67452301$$
$$B = efcdab89$$
$$C = 98badcfe$$
$$D = 10325476$$
$$E = c3d2e1f0$$

圖 2-8 初始化數值(hex)

3. 輸入訊息與雜湊值會經由三種類型的運算：加法運算、位元輪轉運算和非線性函數 F。共執行 80 輪的運算，運算過程又以每 20 輪為一組分為以下四大組:

$$K_t = 5a827999$$
$$F = Ch(x,y,z) = (x \wedge y) \oplus ((\sim x) \wedge z)$$

圖 2-9 第一組運算函式與參數

$$K_t = 6ed9eba1$$
$$F = Parity(x,y,z) = x \oplus y \oplus z$$

圖 2-10 第二組運算函式與參數

$$K_t = 8f1bbcdc$$
$$F = Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

圖 2-11 第三組運算函式與參數

$$K_t = ca62c1d6$$
$$F = Parity(x,y,z) = x \oplus y \oplus z$$

圖 2-12 第四組運算函式與參數

為減少面積，W 我們採用 16 個暫存器架構，因此每輪都需要更新 W 內的數值

$$W_t = \begin{cases} M_t & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$

圖 2-13 Wt 更新函式

4. 最後，將暫存器中的值(A、B、C、D、E)加上其初始雜湊值，形成最終的 160 位元雜湊值。

➢ SHA-256 雜湊演算法

SHA-256 演算法是後繼於 SHA-1 的演算法，其運作原理如下[3]：

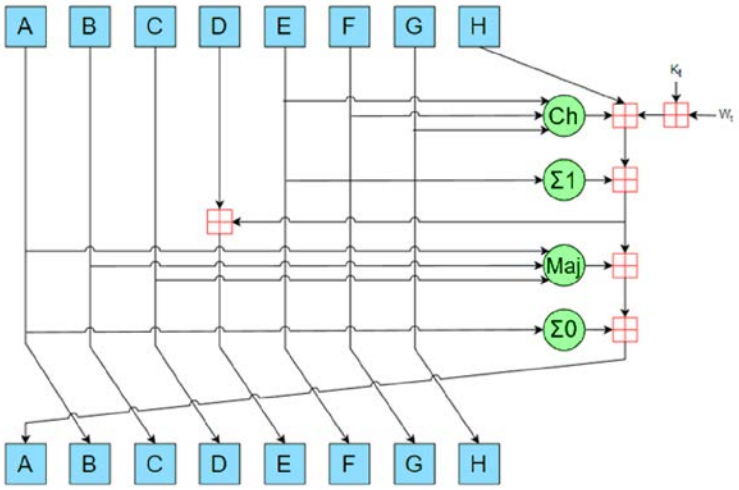


圖 2-14 SHA-256 演算法

1. 將輸入訊息進行填充和分割，形成 512 位元的輸入訊息。再以每 32 位元切割成 16 組並存入 Wt。

2. 賦予一個 256 位元的初始雜湊值（A、B、C、D、E、F、G、H），並存入暫存器。

$$A = 6a09e667$$
$$B = bb67ae85$$
$$C = 3c6ef372$$
$$D = a54ff53a$$
$$E = 510e527f$$
$$F = 9b05688c$$
$$G = 1f83d9ab$$
$$H = 5be0cd19$$

圖 2-15 初始化數值(hex)

3. 輸入訊息與雜湊值會經由三種類型的運算：加法運算、邏輯運算 Σ 和非線性函數 Ch 與 Maj。共執行 64 輪的運算。

$$\mathrm{Ch}(x,y,z) = (x \wedge y) \oplus ((\sim x) \wedge z)$$
$$\mathrm{Maj}(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_{0}^{256}(x) = \mathrm{ROTR}^{2}(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x)$$
$$\sum_{1}^{256}(x) = \mathrm{ROTR}^{6}(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x)$$

圖 2-16 運算函式與參數

為減少面積，W 我們採用 16 個暫存器架構，因此每輪都需要更新 W 內的值

$$W_t = \begin{cases} M_t & 0 \le t \le 15 \\ \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

$$\sigma_0^{256}(x) = \mathrm{ROTR}^{7}(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^{3}(x)$$
$$\sigma_1^{256}(x) = \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x)$$

圖 2-17 Wt 更新函式

4. 最後，將暫存器中的值(A、B、C、D、E、F、G、H)加上其初始雜湊值，形成最終 256 位元雜湊值。

➢ 迭代邊界分析 (Iteration Bound Analysis)

在迭代邊界分析中[1]，我們定義了一個 critical path 長度的最理想邊界值。先找出資料流程圖中的所有迴圈，再個別計算每個迴圈中的所有運算延遲相加後(t_l)除以迴圈中的暫存器個數(w_l)，所有迴圈(L)中的最大值即是欲求得的迭代邊界(T_∞)。

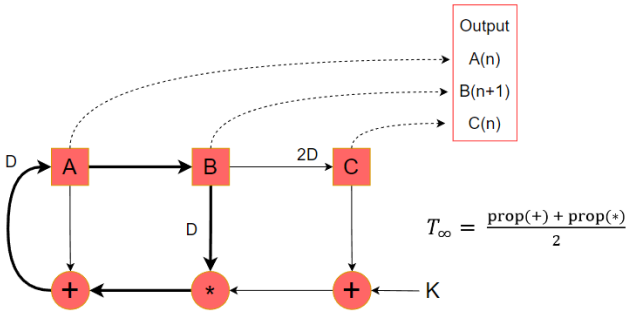
$$T_\infty = \max_{l \in L}\{\frac{t_l}{w_l}\}$$

圖 2-18 迭代邊界分析公式

➢ 展開轉換 (Unfolding Transformation)

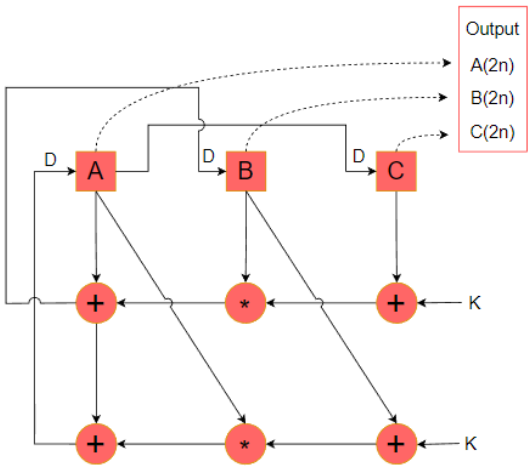
在展開轉換中[1]，我們通常會先定義一個展開係數(k)，作為電路一次需執行的迭代次數。透過建立 n 與 n+k 之間的關係平行化運算，達到減少總執行次數的優化。

在這次的專題實作中，我們都將取迭代邊界的分母為展開係數藉此優化電路的效能。以下以 k=2 為例：



$$A(n+1) = A(n) + B(n)*(C(n) + K)$$
$$B(n+1) = A(n)$$
$$C(n+1) = B(n)$$

圖 2-19 展開轉換前的資料流程圖與數學式



$$A(n+2) = A(n+1) + B(n+1) * (C(n+1) + K)$$
$$= A(n) + B(n) * (C(n) + K) + A(n) * (B(n) + K)$$
$$B(n+2) = A(n+1) = A(n) + B(n) * (C(n) + K)$$
$$C(n+2) = B(n+1) = A(n)$$

圖 2-20 展開轉換後的資料流程圖數學式

➢ 時序調整轉換 (Retiming Transformation)

在時序調整轉換中[1]，我們透過移動暫存器(D)的位置，改變各運算單元之間的時序關係，並且可以在保持電路功能不變的情況下，縮短 critical path 的長度，進而優化電路的最大操作頻率、減少時序延遲。以下是時序調整轉換的例子：

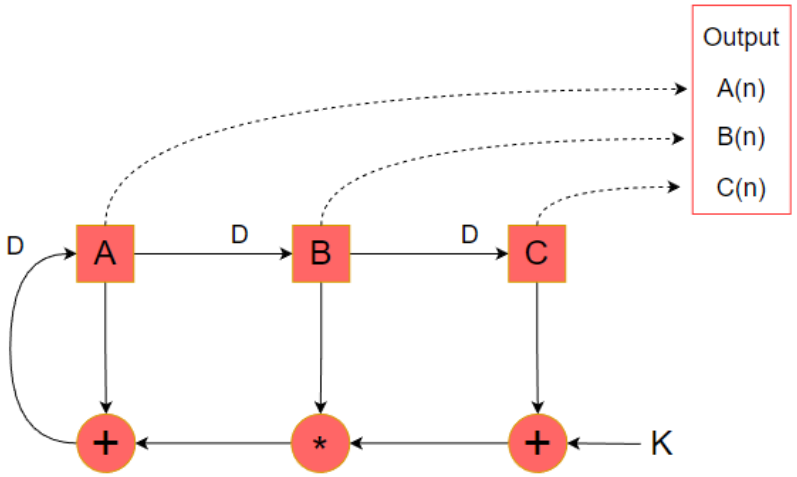


圖 2-21 時序轉換前的資料流程圖

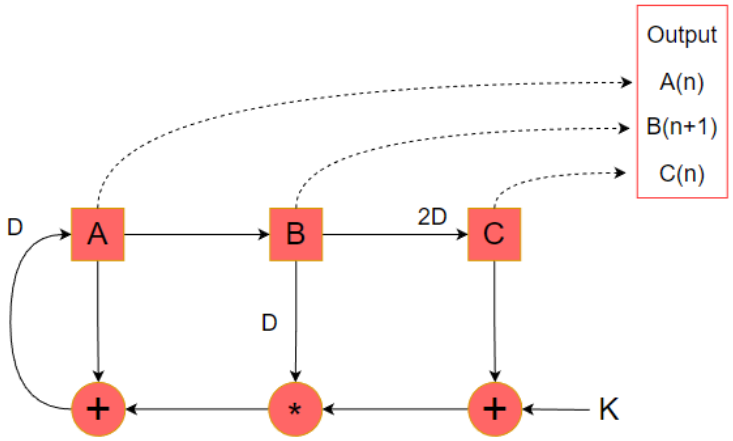


圖 2-22 對 B 點做時序轉換後的資料流程圖

2.2 系統設計

➢ MD4 雜湊演算法

- **資料流程圖表示**

我們先根據原理分析中的 MD4 演算法運算原理圖，寫出了各個暫存器在時序 n 與 n+1 之間的數學關係式，並且畫出 MD4 演算法的資料流程圖。

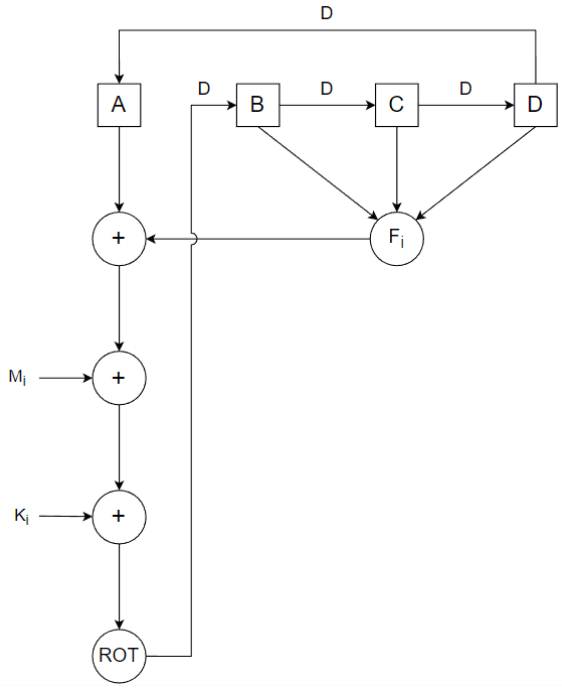


圖 2-23 MD4 演算法資料流程圖

$$A_{n+1} = D_n$$
$$B_{n+1} = ROT(F_i(B_n, C_n, D_n) + A_n + M_i + K_i)$$
$$C_{n+1} = B_n$$
$$D_{n+1} = C_n$$

圖 2-24 MD4 演算法數學關係式

如圖 2-23 所示，圖中的正方形符號代表電路的輸出(A、B、C、D)，圓形符號則是運算函式，其中 F 為非線性函數、ROT 為向左位元輪轉。而在各個符號之間的 D，則是代表目前暫存器的位置。

由於相比與普通加法器，使用 Carry Save Adder（以下簡稱 CSA）僅需通過三層邏輯閘，運算速度較快且面積較小。因此我們將一些加法器適當地改為 CSA，並且 CSA 的運算會有進位與和兩個輸出項，所以在圖中我們以粗體線來表示以作為區別，且在 CSA 的運算後輸入加法器相加其兩個輸出項。

在後面兩種雜湊演算法中我們皆會採用有 CSA 的架構，因此介紹部分便不再贅述。

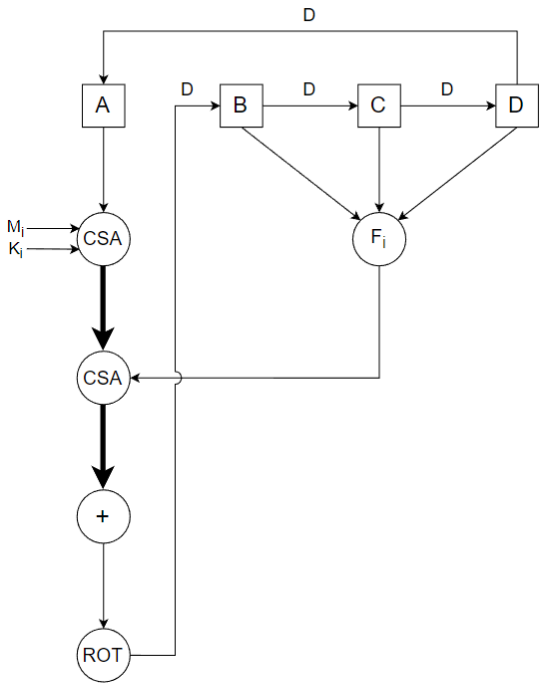


圖 2-25 MD4 演算法資料流程圖(使用 CSA)

- **迭代邊界分析**

由迭代邊界分析，我們可以得到$T_\infty = prop(F_i) + prop(+) + prop(CSA)$。其中 ROT 的運算方式不會延遲不納入計算，$F_i$ 與 CSA 的運算量都大約是加法的三成。

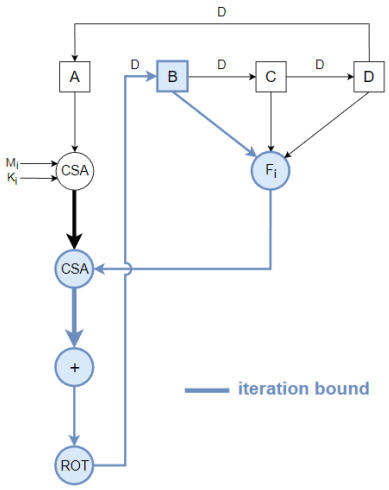


圖 2-26 MD4 演算法迭代邊界

- **Critical Path 優化**

對資料流程圖分析出目前的最長路徑之後，發現大小與迭代邊界最佳解恰好相等，$critical\ path\ =\ prop(F_i) + prop(+) + prop(CSA)$，因此不需再進行其他的轉換，並且架構以此流程圖進行實作。

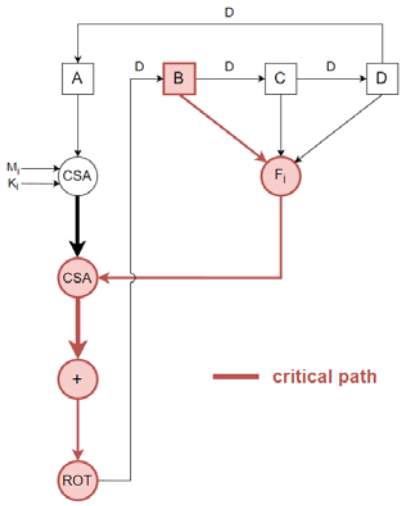


圖 2-27 MD4 演算法 critical path

➢SHA-1 雜湊演算法

- **資料流程圖表示**

我們先根據原理分析中的 SHA-1 演算法運算原理圖，寫出了各個暫存器在時序 n 與 n+1 之間的數學關係式，並畫出 SHA-1 演算法的資料流程圖。

$$TEMP_t = S5(A_t) + F_t(B_t, C_t, D_t) + E_t + W_t + K_t;$$

$$E_{t+1} = D_t; \quad D_{t+1} = C_t; \quad C_{t+1} = S30(B_t); \quad B_{t+1} = A_t; \quad A_{t+1} = TEMP_t;$$

圖 2-28 SHA-1 演算法數學關係式

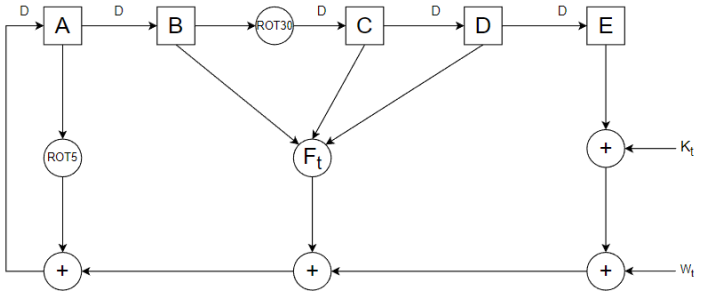


圖 2-29 SHA-1 演算法資料流程圖

- **迭代邊界分析**

由迭代邊界分析，我們可以得到$T_\infty = \frac{2\times prop(+)+prop(F_i)}{2}$。其中 ROT 的運算方式不會延遲不納入計算，$F_t$ 的運算量都大約是加法的三成。原本的資料流程圖中，$critical\ path = 4 \times prop(+)$ 明顯大於迭代邊界，因此需要將架構圖進行優化。

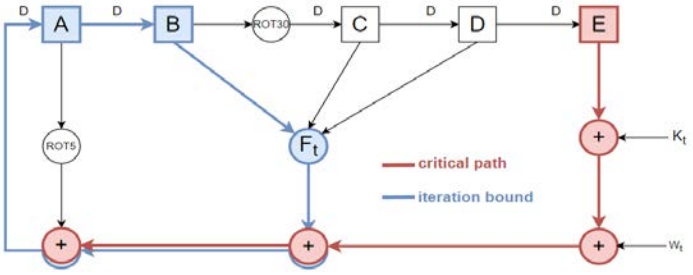


圖 2-30 SHA-1 演算法迭代邊界

- **Critical Path 優化**

由於原本架構的 critical path 無法單純透過時序調整轉換達到理想的迭代邊界，因此我們嘗試**先展開轉換再進行時序調整轉換**。經由這樣的調整後，歸一化的$critical\ path = \frac{2\times prop(+)+prop(F_i)}{2}$，順利地達到迭代邊界。

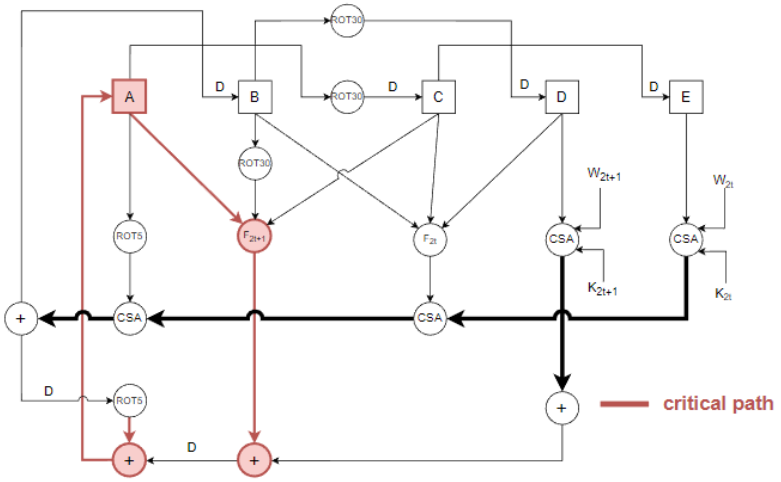


圖 2-31 SHA-1 演算法迭代邊界

➢SHA-256 雜湊演算法

- **資料流程圖表示**

我們先根據原理分析中的 SHA-256 演算法運算原理圖，寫出了各暫存器在時序 n 與 n+1 之間的數學關係，並畫出 SHA-256 演算法的資料流程圖。

$$A_{n+1} = Temp_n + Ma(A_n, B_n, C_n) + \Sigma_0(A_n)$$
$$B_{n+1} = A_n$$
$$C_{n+1} = B_n$$
$$D_{n+1} = C_n$$
$$E_{n+1} = Temp_n + D_n$$
$$F_{n+1} = E_n$$
$$G_{n+1} = F_n$$
$$H_{n+1} = G_n$$
$$Temp_n = W_n + K_n + H_n + Ch(E_n, F_n, G_n) + \Sigma_1(E_n)$$

圖 2-32 SHA-256 演算法數學關係式

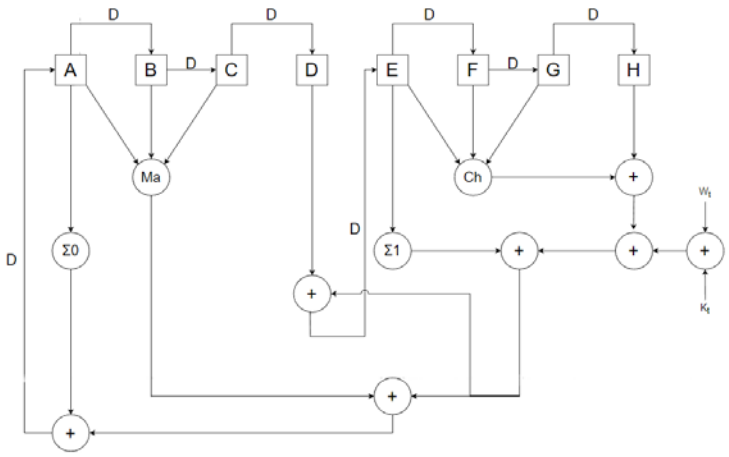


圖 2-33 SHA-256 演算法資料流程圖

- **迭代邊界分析**

在圖 2-33 中，進行迭代邊界分析會得到 $T_\infty = 4 \times prop(+) + prop(Ch)$。然而在經由適當的運算順序及運算元件調整後(圖 2-34)，我們便能得到 $T_\infty = prop(Ch) + prop(CSA) + prop(+)$，遠遠小於原本的迭代邊界。這裡就體現出了迭代邊界的特性：**相同的結果之下，運算架構會影響迭代邊界的大小，從而影響最後的優化結果。**

在目前的優化中，我們有效的將迭代邊界縮小。然而 critical path 長度尚未達到迭代邊界，因此仍需進行優化。

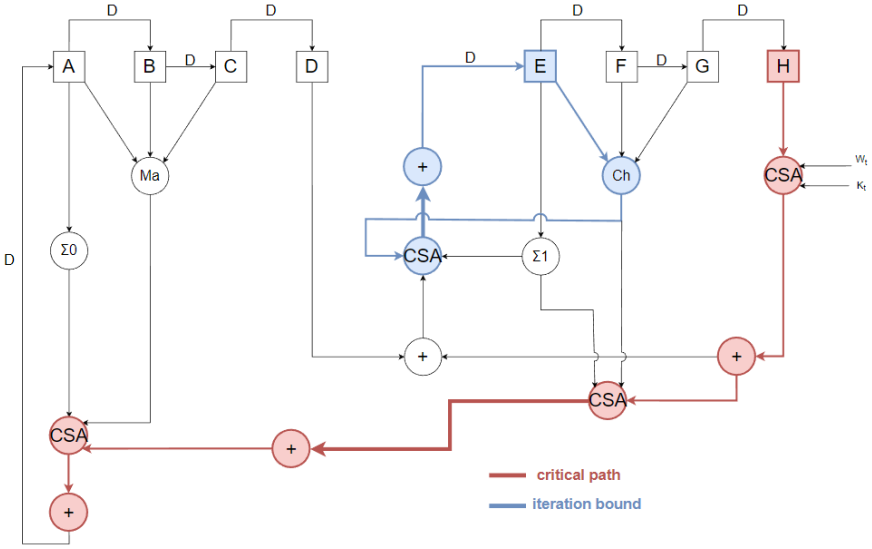


圖 2-34 SHA-256 演算法優化版迭代邊界

- **Critical Path 優化**

為了使 critical path 達到迭代邊界，我們對架構進行適當的時序調整轉換，使 $critical\ path\ = prop(Ch) + prop(CSA) + prop(+)$。

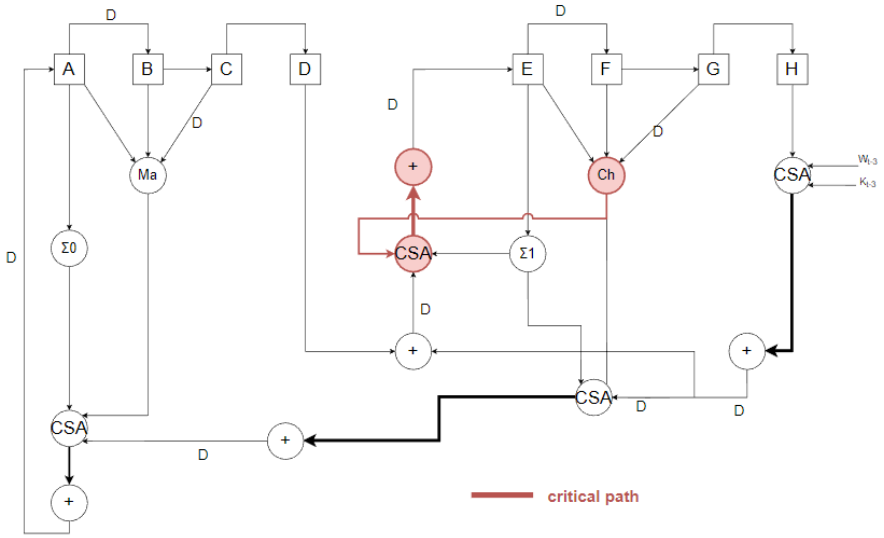


圖 2-35 SHA-256 演算法優化版 critical path

三、實作結果

雜湊演算法效能比較的數據吞吐量皆採用以下公式[1]計算：

$$Throughput = \frac{\text{Frequency}}{\text{number of Cycles}} \times (512bits)$$

圖 3-1 數據吞吐量計算公式

➢MD4 雜湊演算法

- **功能驗證**

```
####################################
*** MD4 Algorithm Verification ***
test pattern number : 12380
match case : 12380/12380
collision case : 0/12380
------------------------------------
$match rate : 100.00%
$collision rate : 0.00%
####################################
```

- **效能比較**

由於 MD4 雜湊演算法的開發較為早期，因此在硬體實現上的參考文獻較少且製程參數與現今差異較大。故 MD4 演算法僅與軟體效能做比較

MD4	Technology (μm)	Area (μm^2)	Frequency (MHz)	Cycles	Throughput (Mbps)
Python code	x	x	x	x	1.039
實作結果	0.18	53320	138.88	48	1481.5

圖 3-2 MD4 演算法效能比較

➢SHA-1 雜湊演算法

- **功能驗證**

```
####################################
*** SHA1 Algorithm Verification ***
test pattern number : 12380
match case : 12380/12380
collision case : 0/12380
------------------------------------
$match rate : 100.00%
$collision rate : 0.00%
####################################
```

- **效能比較**

SHA1	Technology (μm)	Area (μm^2)	Frequency (MHz)	Cycles	Throughput (Mbps)
Python code	x	x	x	x	3.2
[4]	0.18	701700	116	80	824.9
[5]	0.18	230000	290	82	1810
實作結果	0.18	97719	166.67	41	2081.3

圖 3-3 SHA-1 演算法效能比較

➢SHA-256 雜湊演算法

- **功能驗證**

```
####################################
*** SHA256 Algorithm Verification ***
test pattern number : 12380
match case : 12380/12380
collision case : 0/12380
------------------------------------
$match rate : 100.00%
$collision rate : 0.00%
####################################
```

- **效能比較**

SHA256	Technology (μm)	Area (μm^2)	Frequency (MHz)	Cycles	Throughput (Mbps)
Python code	x	x	x	x	2.97
[6]	0.18	178710.84	120	65	944
實作結果	ICC	103982.72	153.8	68	1158

圖 3-4 SHA-256 演算法效能比較

四、結論

雜湊演算法在資訊安全中扮演著重要的角色，因為它們可以將任意大小的輸入資料轉換成固定大小的雜湊值，並確保碰撞難度大且不可逆。這項專題旨在實現高速的 MD4、SHA-1 和 SHA-256 雜湊演算法硬體加速器，以提高數據安全的應用。

經過深入的原理分析和電路架構設計，並且對架構進行多次優化與比較之後，我們成功實現了這些硬體加速器，並進行了性能評估。根據實驗結果，MD4、SHA-1 和 SHA-256 的硬體加速器在每秒處理超過 1 Gb 的數據速度方面表現出色。相較於其他相關研究的硬體設計，在處理速度與電路面積皆具有競爭優勢，效能更是明顯優於軟體的實現。

這項研究能夠為現有的數據處理系統提供了更高的效率和更強的安全性。隨著數據處理需求的不斷增加，以及資訊安全的重要性日益凸顯，高效且安全的雜湊演算法硬體實現將有望改進許多關鍵領域的性能和可靠性。這項研究為數位應用領域帶來了新的可能性，將有助於提高現代系統的效能和安全性。

五、參考文獻

[1] Yong Ki Lee, Herwin Chan, and Ingrid Verbauwhede, Design Methodology for Throughput Optimum Architectures of Hash Algorithms of the MD4-class.

[2] Ronald L. Rivest, The MD4 Message Digest Algorithm.

[3] Federal Information Processing Standards Publication 180-4. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[4] T. S. Ganesh and T. S. B. Sudarshan, ASIC Implementation of a Unified Hardware Architecture for Non-Key Based Cryptographic Hash Primitives. Proceedings of the International Conference on Information Technology: Coding

and Computing (ITCC'05). Pp. 580–585. 2005.

[5] Helion SHA-1 hashing cores. Helion Technology. http://heliontech.com/sha1.htm.

[6] Marsgod. Secure Hash Algorithm. www.opencores.org, 2004.

[7] Ling Bai, Shuguo Li, VLSI Implementation of High-speed SHA-256 , 2009 IEEE 8th International Conference on ASIC, p.133

六、計劃管理與團隊合作方式

計劃管理方面，我們開始時與指導教授進行了深入的討論，明確了專題的目標、範圍和時間表。這成為我們的指導方針。我們將專題以每兩週為一單位分成不同的階段，每個階段有特定的目標和任務，並分配給團隊成員。我們定期與指導教授開會報告進度，並討論可能的變更或挑戰。當遇到困難時，我們立即通知實驗室學長，一起尋找解決方案。指導教授在專題中提供專業指導和監督，幫助我們克服技術和方法上的挑戰。

在團隊合作方面，我們根據彼此行程進行工作量分配，並且每週四天一起至實驗室完成進度、解決困難和規劃下一個目標。另外，我們使用了一些協作工具，如共享文檔和即時通訊平台，以方便團隊之間的即時交流。這些工具有助於快速解決問題，分享資源和協作撰寫報告。

組員	主要工作內容	專題貢獻度
翁梓薰	理論研究、軟體驗證 架構設計、RTL實現	50%
林伯璨	理論研究、軟體驗證 架構設計、RTL實現	50%

圖 6-1 工作分工表