國立清華大學 電機工程學系

實作專題研究成果摘要

A 65,536-Point Radix-4 Number Theoretic Transform Hardware Accelerator for Fully Homomorphic Encryption

應用於全同態加密之65536點 Radix-4數論轉換硬體加速器

專題領域： 系統領域

組別: B564

指導教授： 黃朝宗 教授

組員姓名： 黃義凱、高晧瑄、黃郁程

研究期間： 2025年02月 至 2025年12月 止，共10個月

摘要

本專題提出一套針對全同態加密 FHE 的 65,536 點 Radix-4 NTT 硬體加速器，其核心創新在於：將 CFNTT 的衝突自由映射與 ping-pong 記憶體整合，於高平行度下仍可採用單埠 SRAM 並維持連續資料流；配合 Radix-4 NTT 演算法，在不增加運算量下把記憶體吞吐需求減半，顯著緩解頻寬與面積壓力（有利於成本與可實現性）。

在運算核心方面，我們將蝴蝶運算單元（BFU）管線深化至 18 級，縮短臨界路徑、提升時脈潛力；並提供 1/4/16 BFU 可擴展平行度，搭配併行 I/O 介面以移除資料輸入/輸出瓶頸，使吞吐量可隨平行度近線性放大，分別達成 ~1×、3.64×、13.27× 的加速效益。

為提升系統通用性，我們支援 ≤64-bit 字長並以 twiddle-factor SRAM 動態載入不同模數的旋轉因子，同一硬體即可覆蓋多種 FHE 參數組態，降低系統整合與後續維護成本。整體而論，本設計在面積/頻寬受限的實作條件下，同時提供高吞吐與高度彈性，為 FHE 應用（如隱私保護的雲端運算與模型推論）帶來更佳的效能/成本比。

合成結果顯示，平行度為 1, 4, 16 的設計能在 200–210 MHz 下運行，性能指標 PI 分別為 0.26, 0.86, 2.51，與 FAST[2] 設計的 8.22 尚有差距。分析發現瓶頸主要來自蝴蝶單元臨界路徑與記憶體面積過大（佔比高達 95% 以上）。未來工作將聚焦於：

1. 進一步優化管線化與運算演算法；
2. 改善記憶體架構，降低面積；
3. 提升平行度以接近甚至超越文獻效能。

本專題最終建構出一套可支援彈性資料長度的高平行度 NTT 硬體設計，對於實現實務可行的 FHE 加速器具有重要意義。

1. 全同態加密與數論轉換概覽

1-1. 全同態加密架構介紹

全同態加密(Fully Homomorphic Encryption, FHE）是一種後量子密碼時代的加密技術，其特殊之處在於能直接針對密文進行加法及乘法運算，而非如其他加密結構需將密文解密方才能進行運算。全同態加密技術促進了跨機構之資料合作，如醫院、銀行、政府等機構彼此不必分享原始資料，仍可在加密狀況下進行計算統計、模型訓練等操作。

在日常生活中，判斷新密碼的安全性即為全同態加密的一種應用。當我們設定了一組新密碼，密碼透過全同態加密後，傳至伺服器端進行一系列運算，即可判斷我們的密碼為「安全」、「有風險」或是「非常不安全」。其中的傳輸過程以及伺服器運算階段，皆在同態加密的架構下進行,即伺服器端也無從破譯出明文,以此實現雲端的隱私保護。

全同態加密的安全性來自於晶格問題的計算複雜度。密文與明文的關係如下：

$$c = (A, B);\ B = (A \times s + e + \Delta m)\ mod\ Q$$

其中A為一隨機生成的遮罩，s為秘密金鑰，e為雜訊，m為明文。欲以秘密金鑰還原明文，則按以下關係式：

$$B - A \times s = e + \Delta m, \qquad m = \left\lfloor \frac{B - A \times s}{\Delta} \right\rceil$$

倘若未掌握秘密金鑰的第三方欲直接破譯出明文，需要透過掌握大量密文，在雜訊的干擾下找出B的公因數，利用公因數還原出秘密金鑰。又實務設計中Q為一 700 位元以上的數字，使得破譯成為一複雜度極高的晶格問題，藉此建構出同態加密的安全性。

全同態加密系統中通常使用 RLWE(Ring Learning With Errors)的方式進行運算。會先將複數明文加密進入實數多項式，而後在實數多項式環上進行運算。其中的乘法運算將大量使用多項式環上的捲積運算，實務上為降低運算量，會仿效訊號處理中的傅立葉轉換，將多項式環映射至頻域，並在頻域上進行點對點乘法。數論轉換法（Number Theoretic Transform, NTT）即在此基礎上，以類似快速傅立葉轉換法的運算操作，讓頻域轉換過程的運算複雜度由$O(N^2)$下降至$O(NlogN)$。

1-2. 數論轉換加速器的必要性

傳統數位通訊系統中的快速傅立葉轉換運算點數N 通常為 512 或 1024 點。相比之下，全同態加密架構下的核心運算數論轉換，其運算點數N 暴增至 65536 點。

此外，全態加密下的數論轉換要求每點運算採用 60 位元的加法與乘法，這使得其運算量遠超一般數位訊號處理晶片的負荷能力。同時，為達到最大運算效率，數論轉換對記憶體的要求極高，不僅需要至少 1000 位元的吞吐量，還需 65536x63 SRAM 來進行資料暫存。

總而言之，全同態加密下的數論轉換對運算量、記憶體吞吐量與容量提出了極高的要求，因此，設計專用的數位電路加速器以克服通用數位訊號處理晶片的限制，是實現全同態加密實用化的必要途徑。

2. 問題發想與研究方法

全同態加密雖能在加密條件下進行運算，然而其最大的阻礙為密文的運算量過於龐大，其中大量的乘法運算需仰賴數論轉換單元的操作。在此專題中，我們希望在現行全同態加密的規格下，實作出能支援高點數、高效率、高運算彈性的數論轉換單元，設法優化全同態加密的運算瓶頸。

此研究中，我們將閱讀同態加密系統設計的文獻，了解其中數論轉換單元的硬體規格，並根據該規格定義出性能指標(Performance Index, PI)。接著開始蒐集數論轉換單元的架構資料，以硬體角度分析不同架構在高平行度運作下的吞吐量與硬體需求，尋求最適合實作於硬體的架構。而後利用 C/Python 實作出軟體架構，生成硬體實作對應的 feeding data 與 golden pattern。

完成軟體實作後，我們將開始硬體設計步驟。實做低平行度的數論轉換單元，並比較我們的效能指標與文獻間的差距，而後開始透過拉高平行度使效能指標能逼近甚至超越文獻的水準，並根據我們的研究結果提出不同的設計優化面向供相關研究進行參考。

3. 目標規格訂定

為讓實作的數論轉換單元能符合同態加密系統中的實務需求，我們比較了三篇於 ASIC 實作同態加密的研究論文，分別為 SHARP[1], FAST[2], BTS[3]，並以三篇論文中數論轉換單元的平行度以及吞吐量表現作為實作標準。三篇論中數論轉換單元的硬體規格及吞吐量如表一。

	Frequency (Hz)	NTT Area (um^2)	Word length(bit)	Lane	N	Throughput (#NTT/ms)
SHARP[1]	1.0GHz	unknown	36	1024	65536	1953
FAST[2]	1.0GHz	60.88	60	1024	65536	1953
BTS[3]	1.2GHz	19.46	64	2048	65536	4690

表一、 同態加密文獻中數論轉換單元的各項規格

由表可知，三篇論文皆針對 65536 點數論轉換進行高平行度、高運算頻率的設計，以滿足同態加密系統中的吞吐量需求。此外，我們亦參考了一篇實作於 FPGA 平台的同態加密論文 HEAP[4]，由於實作平台不同，其運算頻率與運算點數皆無法與其他三篇的規格直接比較。但 HEAP 中的硬體設計則支援了 36 至 52 位元資料長度下的彈性運作，而非如其他設計中固定資料長度。鑒於資料長度彈性在同態加密系統中的重要性，亦將論文中的資料長度彈性的功能實作於本研究數論轉換單元。故最終確立本設計之規格為 65536 點，且可支援 64 位元以下的資料長度的彈性運算架構。

為評估本設計與文獻中數論轉換單元的效能差異，本研究選取吞吐量與面積兩項參數，以量化不同設計間單位面積可達成的運算效率。進一步定義性能指標(Performance Index, PI)為：

$$PI = \frac{Throughput(\#NTT/ms)}{Area(um^2)}$$

由於本研究中使用 ADFP N16 製程進行硬體實作，而非文獻中所使用的 N7 製程。為確保比較公平性，我們參考了 TSMC 製程技術間投影資料[5]。可知製程技術由 N16 進步至 N7 時，電路的運作頻率可提升約 30%，而電路面積則會縮小至原來的三分之一。又 SHARP, FAST, BTS 三篇文獻的設計中，SHARP 一文並無提供數論轉換單元的面積資料。BTS 的設計又將數論轉換單元中的運算部分與記憶體部分拆開，無法比較數論轉換單元的面積資料。故將 FAST 的設計經過製程正規化後，得到的 PI 為 8.22，將以此作為基準進行後續專題研究中的數論轉換架構設計。

4. 文獻探討與演算法選用

決定完目標規格後，我們繼續研究關於 NTT 內部架構的相關文獻，其中有三篇文獻的設計方式能應用於我們的架構，分別為 CFNTT[6], Efficient and Flexible NTT Accelerators[7], Optimizing Barrett Modular Multipliers [8]。

4-1. CFNTT

論文中研究者以 Radix-2 架構為基礎，利用旋轉因子(twiddle factor)在質數環上的數學關係，通過合併化簡推導出 Radix-4 架構。比較 Radix-2 與 Radix-4 可知，Radix-4 的

運算架構能在不增加運算量的前提下，減少一半的記憶體吞吐量需求。此特性在高平行度設計中影響尤其明顯，因此本研究參考了 Radix-4 的運算架構進行實作。

此外，該論文亦提供了一種記憶體映射衝突的解決辦法。其演算法能適用於不同 Radix 基底與不同平行度架構，且能確保同一週期中不會對同一個 SRAM 模組進行重複訪問。此特性能讓我們使用 single-port SRAM 進行硬體設計，亦能避免在拉高平行度時需要重新設計記憶體映射的方法。基於以上優點，我們採用了文中提出的記憶體映射方法進行實作。

4-2. Efficient and Flexible NTT Accelerators

此篇論文以 dual-port SRAM 進行設計，並使用了 ping-pong 的記憶體架構。受限於記憶體無法在一周期內同時進行讀與寫的操作，若採用原來的架構，需要在相鄰兩個週期中交錯進行讀與寫，導致硬體使用率僅達 50%。Ping-pong 架構則通過配置兩組對應的記憶體，使資料於一組進行讀、一組進行寫，雖需使用兩倍的記憶體資源，卻能有效提升資料的流動效率。

在本研究中，最終結合了 CFNTT[6]的記憶體映射方式與此篇論文中的 ping-pong 記憶體架構，將 SRAM 分成了 Main SRAM blocks 與 Second SRAM blocks。運作過程中，兩者分別負責讀與寫，並隨運算階段交替角色，實現資料存取的連續性。

4-3. Optimizing Barrett Modular Multipliers

數論轉換演算法的基礎運算單位為蝴蝶運算單元(Butterfly Unit, BFU)，由一模數加減法與一模數乘法構成。其中模數加減法的複雜度較低，即直接計算再透過加減法將結果映射回質數環即可。然而在一質數Q的質數環中，乘法的計算結果可高達Q^2。若對此大數直接取餘數以實現模數乘法，將耗費大量的運算週期與複雜度。

多種模數乘法的優化方式中，我們選用了 Barret 模數乘法進行實作。此篇論文中以數學方式驗證 Barret 模數乘法，對於不同 branch 次數、輸入長度、輸出長度，皆有不同的參數選擇方式，以確保結果能正確的映射回質數環。最終我們參考了這篇論文的做法實作了模數乘法，假設模數Q為一 n 位元的質數，輸入為質數環上的 A, B 二數：

$$CONS = \left\lfloor \frac{2^{n+1}}{Q} \right\rfloor;$$

$$C = A \times B;$$

$$C1 = Rightshift(C, n-2);$$

$$\hat{q} = Rightshift(C1 \times CONS, n+3);$$

$$result\_temp = C - \hat{q} \times Q$$

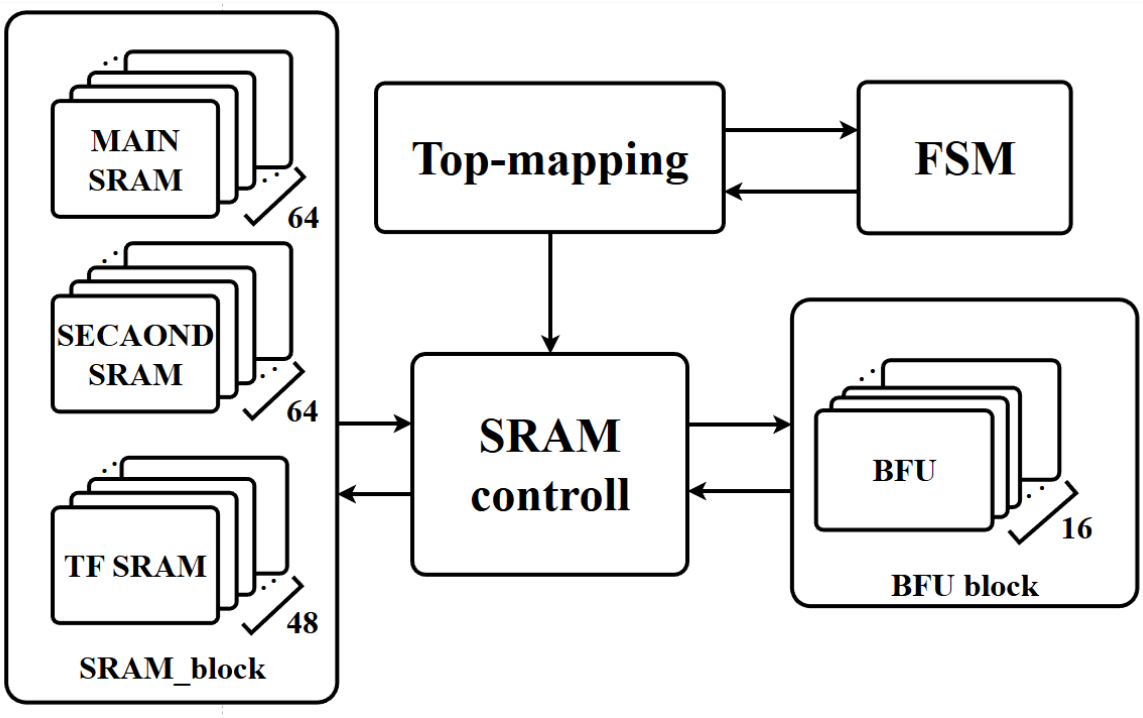
此時仍需進行一次判斷，若$result_temp \geq q$，則模數乘法的正確結果應修正為$result_temp - q$，使結果映射回質數環。

上述幾篇文獻中針對數論轉換單元的設計規格與我們訂定的相差甚遠，如資料點數僅有 1024 點，模數Q僅有 14 位元。故我們將參考文獻中 Radix-4 蝴蝶運算單元以及 ping-pong 的 SRAM 架構設計進行實作，規格則訂為資料點數 65536 點，支援 64 位元以下質數的彈性運算，最後通過拉高平行度以及運算頻率來達成我們訂定的 PI。

5. 硬體實作

5-1. 系統整體架構

為將上述架構實現於硬體，本研究設計了之數論轉換硬體架構。如圖一所示，整體架構由 FSM、Top-Mapping、SRAM-block、BFU-block 以及 SRAM-controller 幾個單元組成。



圖一、 硬體設計架構之 Block Diagram

整體系統架構中，Top-Mapping 單元負責生成每筆運算的資料地址。Top-Mapping 中以三組參數{p, j, k}定義數論轉換中每一次運算需要的各筆資料。當 Top-Mapping 收到 FSM 的啟動訊號後，會根據當前參數進行參數更新。同時，根據參數{p, j, k}的數值，Top-mapping 會產生資料對應資料地址 Bank-index 與 Bank-address 給 SRAM-controller，以決定運算資料對應記憶體位置。

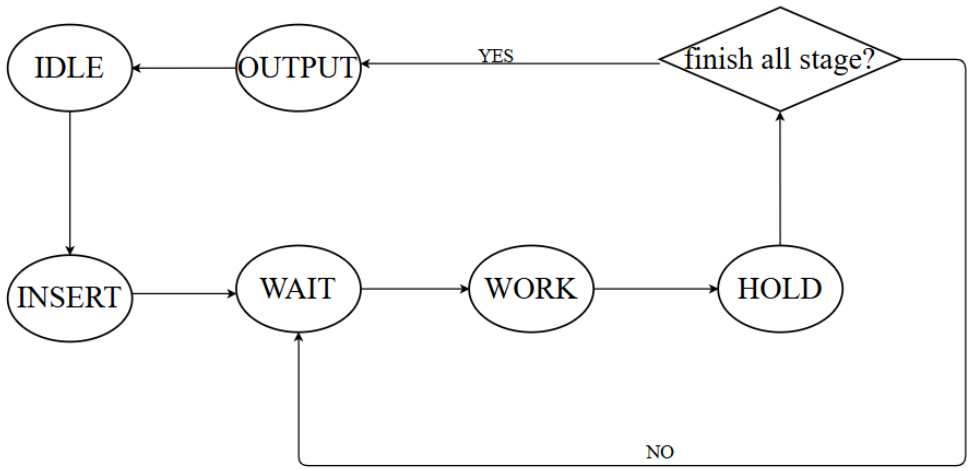
BFU-block 單元為本系統中主要運算核心，負責執行數論轉換過程中的模數加減法與模數乘法運算。BFU-block 由多個蝴蝶單元所組成，並採用 Radix-4 的設計，每個蝴

蝶單元內含有四組模數乘法器以及多組加減法單元，能同時對將四筆輸入資料與四個旋轉因子進行運算，並輸出四組結果。且為進一步提升系統運作頻率與吞吐量，本研究以文獻提出之 14 級的管線架構為基礎進行優化，最終實現了 18 級管線化的蝴蝶運算單元，有效提高操作頻率。

SRAM-controller 為系統中的核心控制單元，負責資料的存取與分配，以及生成所有 SRAM 的控制訊號。SRAM-controller 會根據 Top-mapping 單元產生的 Bank-index 與 Bank-address，以及 ping-pong 系統中的切換狀態，自 SRAM-block 讀取對應的運算資料與旋轉因子，並依序分配至 BFU-block 進行蝴蝶運算。同時，SRAM-controller 會接收 BFU-block 的運算結果，重新分配回 SRAM-block 供記憶體寫入。SRAM-controller 能確保高平行度運算下資料一致性與時序正確性，同時維持整體記憶體通道的高利用效率。

5-2. 控制流程設計

為將上述架構實現於硬體，本研究依據系統的運作流程設計了數論轉換單元的 FSM，整體系統的運作分為 Insert, Wait, Work, Hold 與 Output 五個主要狀態。其狀態轉移方式如圖所示。



圖二、 硬體設計架構中控制訊號之 FSM

- Insert: 在運算開始前，將運算資料載入 Main SRAM，並將質數環對應的旋轉因子載入 twiddle factor SRAM。
- Wait: 資料進入蝴蝶單元開始運算，由於管線設計，蝴蝶單元尚未輸出有效結果。故 ping-pong SRAM 系統中僅進行讀取操作，而不進行任何寫入操作，避免無效資料覆寫 SRAM。
- Work: 此狀態為系統中主要運算階段，此時蝴蝶單元開始輸出有效資料。故 ping-pong 架構中兩組 SRAM 分別進行讀取與寫入的操作。其中一套持續讀取運算所需

資料，另一套則同步將蝴蝶單元的運算結果寫入。

- Hold: 當 SRAM 中資料已讀取完畢，系統暫停新的讀取操作，等待蝴蝶單元管線中剩餘的運算結果全部寫回 SRAM。
- Output: 完成數論轉換 8 個階段的運算後，將最終的數論轉換運算結果從 Main SRAM 輸出，完成一次完整的數論轉換流程。

5-3. 記憶體系統

本研究之記憶體系統主要分為二部分，分別為儲存資料的 ping-pong SRAM，與儲存旋轉因子的 twiddle-factor SRAM。在文獻中，數論轉換系統通常僅能支援單一固定質數的運作，故將旋轉因子以 ROM 形式進行儲存。此架構雖能簡化硬體控制，但無法支援多質數環下的運作，與我們的高彈性的設計目標不相符。為提升系統的通用性，本設計額外以 twiddle-factor SRAM 動態儲存旋轉因子。當系統開始新一輪的數論轉換計算時，能一併載入該質數對應的旋轉因子。

此外，由於此專題使用的 ADFP 製程並沒有提供 Memory Compiler，無法直接產生大容量的 SRAM 單元，因此僅能使用內建的記憶體陣列手動組構出需要的記憶體容量。Main SRAM 為例，Main SRAM 內需要儲存 65536 筆長度為 64 的資料，我們僅能以 128 個 128x64 SRAM 拼湊出所需記憶體，並滿足大 SRAM 與小 SRAM 單元間 read-write, address, data-in, data-out 等訊號間的銜接關係，還原出設計所需的儲存空間與運作需求。

5-4. 高平行度設計

實作了一個蝴蝶運算單元的數論轉換系統後，我們在相同架構下新增了更多的蝴蝶運算單元，以實現高平行度的目標。同時，我們進行了另外三項功能的優化。

一為 SRAM-controller 判斷邏輯的優化。隨著平行度拉高，我們發現原來 SRAM-controller 判斷邏輯的複雜度將以平方關係進行成長，故我們重新優化了判斷邏輯的部分，避免在之後的合成階段因為 fanout 過大需要疊加大量的 buffer。

二為將資料的輸入由串列輸入(serial)改為並列輸入(parallel)。我們發現疊加蝴蝶運算單元固然能減少運算週期，但在高平行度的架構下，運算週期反而受限於資料的輸入與輸出。在 65536 點的數論轉換架構下，共有 65536 筆資料與 65536 筆旋轉因子需要輸入。以 16 個蝴蝶運算單元的高平行度設計為例，僅需花費 8336 個週期進行運算，卻要花費 196608 個週期處理資料的輸入與輸出。故我們重新設計了資料處理的方式與系統的輸入輸出介面，將原先的串列輸入改為並列輸入，避免運算週期受限於資料的傳輸。

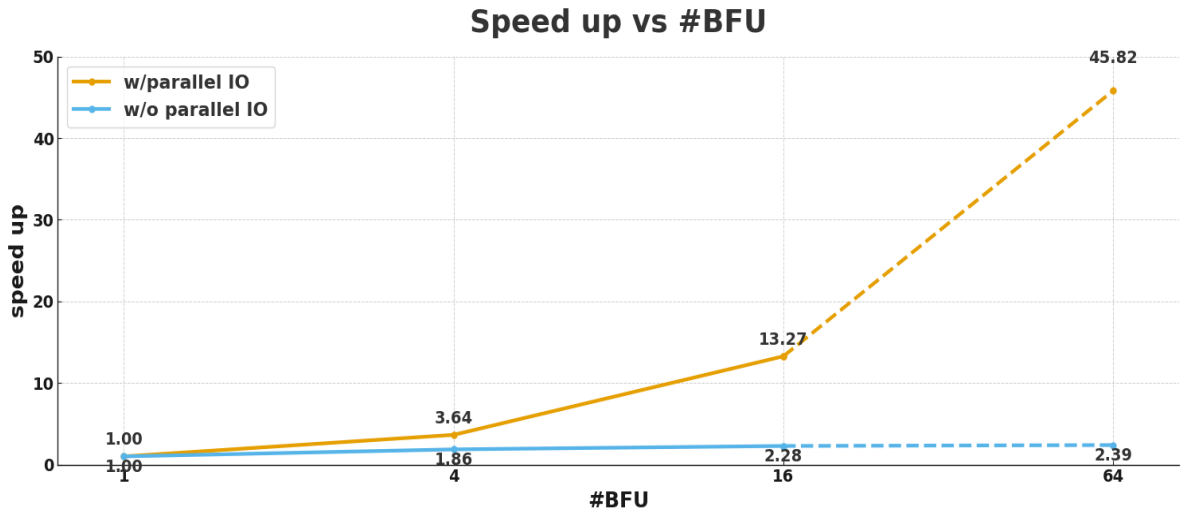
三為旋轉因子 SRAM 之控制電路優化。原先我們只用 3 個 SRAM 存取旋轉因子，使得每個旋轉因子的 life time 並不一致，進而導致在越後期的運算階段會有很多蝴蝶運

算單元閒置，使整體 speed up 在 16 個蝴蝶運算單元時僅提升 4.92 倍，不具效益。因此我們將存取旋轉因子的 SRAM 分割成 3 * 16 個小型 SRAM，使得每個運算單元在每個運算階段都呼叫相對應旋轉因子，不會有任何單元閒置，使整體 speed up 在 16 個蝴蝶運算單元時達到 13.27 倍。

綜合以上設計，我們實現了 1 個、4 個及 16 個蝴蝶運算單元的數論轉換系統，結果如表二。

#BFU	#Total Cycle	SRAM throughput needed	speedup
1	169445	88 byte	x1.0
4	46565	352 byte	x3.64
16	12795	1408 byte	x13.27

表二、不同平行度下數論轉換加速情形



圖三、不同平行度下優化前後的加速差異(虛線部分表理論值、並未實作)

6. Testbench 軟體實作

我們將 Testbench 分為底層運算邏輯、旋轉因子預處理以及數論轉換主體架構三個部分，並透過 C 語言完成實作。

6-1. 底層運算邏輯

由於我們需要在 C 語言中實現 64 位元的數論轉換運算，過程中需要的運算範疇以超出 C 語言中整數 64 位元的長度限制。故另定義一長度為 128 位元的資料型別以實現運算。在這個全新的資料型別下，我們重新建立了比大小、加減法的運算，並以此還原了模數加減法。而後我們參考 Optimizing Barrett Modular Multipliers[8]中的 Barret 模數

乘法步驟，通過拆分位數實現乘法、如 CPU 右移與減法的循環實現除法。最終完成了模數乘法，在這套 128 位元的資料型別下還原了模數加減法與模數乘法的運算。

6-2. 旋轉因子預處理

旋轉因子的生成方式為模數 Q 原根(primitive root)的 0 次方至 65535 次方對 Q 的餘數，其次數之高已無法以正常指數運算進行操作，故透過演算法將指數拆分成多層模數乘法，既能最大幅度的減少運算量，也能確保結果的範圍位於質數環上。

同時，參考 CFNTT[6] Radix-4 的演算法，我們需要將數論轉換的旋轉因子進行順序上的預處理，才可載入記憶體，避免在硬體設計中需要使用額外的邏輯控制旋轉因子輸出順序。最終，我們以一種類似 bit-reversed 的運算方式實現了旋轉因子的預處理。生成的旋轉因子除了提供 Testbench 內部的數論轉換運算以外，同時也會做為硬體系統中 twiddle-factor SRAM 載入的資料。

6-3. 數論轉換主體架構

透過已實現的模數運算以及旋轉因子的預處理，我們在軟體上還原出了蝴蝶運算的架構，實現了數論轉換的主體架構。我們通過兩筆資料測試軟體的正確性，一為僅第一筆資料為 1，其餘資料為 0，由於數論轉換的本質為多項式乘法，這樣的測試資料經過數論轉換後，所有的資料皆會變成 1。第二筆測試資料為第 i 筆資料為 i 的組合，這樣的測試資料經過 NTT 與 INTT 的步驟後，所得到的結果應與輸入相同。

確認軟體功能無誤後，我們將數論轉換的輸入與輸出按記憶體映射方法重新排序，符合硬體中並列的輸入輸出介面，生成 Testbench 所需的 golden pattern。

7. 合成結果分析

以 TSMC 先進製程教育套件 N16 ADFP 的邏輯合成結果如下:

	Frequency	Lane	Total Area (um^2)	PI
d = 1	200MHz	4	5.78	0.26
d = 4	210MHz	16	7.43	0.86
d = 16	200MHz	64	9.71	2.51
FAST[2]	770MHz	1024	182.64	8.22

表三、已完成之低平行度架構與 FAST[2]設計合成結果之比較

根據現有的合成結果，可分析出以下幾點結論以及日後應進行的優化方向：

7-1. 運作頻率優化

第一優化方向為重新修正蝴蝶運算單元的管線化設計。本研究已在 CFNTT[6]的設計基礎上進行優化，進行了 18 級的管線化設計。使得每一個時脈週期僅執行單一加減法或乘法運算，避免出現連續的組合運算。然而，即便在此架構下，合成結果的報告中顯示臨界路徑(critical path)仍落於蝴蝶運算單元。使得整體系統的操作頻率受限於 200MHz，與 FAST[2]的 770MHz 有明顯差距。

為進一步改善此問題，未來工作將考慮使用 Synopsys DesginWare 提供的 pipeline 運算單元，或自行設計多週期運算演算法，以改善 64 位元運算長度帶來的延遲，提升系統的運作頻率。

7-2. 記憶體面積

分析合成結果中記憶體面積與運算單元面積，結果如表四。

#BFU	記憶體面積um^2	運算單元面積um^2	記憶體面積占比
1	5.69	0.09	98.4%
4	7.09	0.34	95.4%
16	9.75	1.11	88.6%

表四、合成結果中運算單元與記憶體面積分佈狀況

觀察合成結果，可發現運算單元的面積部分按預期成長。在平行度為 1 時僅有一蝴蝶運算單元，平行度為 4 時有四個蝴蝶運算單元，運算單元的面積亦成長了將近 4 倍。平行度為 16 的合成結果亦符合此成長趨勢。

然而，受限於此專題使用的 ADFP 製程沒有 Memory Compiler 的功能，我們必須以現有的記憶體陣列手動組構出硬體需要的記憶體容量。以 65536×64 的記憶體為例，我們需要使用 128 個 128×64 的記憶體進行組合，合成時會出現 128 套無關聯的記憶體讀取電路，而非 65536 筆資料共用一套讀取電路，造成不必要的面積浪費。又我們的合成結果中，記憶體的面積佔比高達 98.4%, 95.4%, 88.6%，可想而知這樣的面積浪費將大幅影響我們的比較結果。

在日後進行高平行度硬體系統的合成時，如同先前硬體架構段落的分析，高平行度的硬體系統需要較大的記憶體吞吐量需求。但是整體儲存的資料量並無變化，如同平行度為 1 與 4 的合成結果中，記憶體面積僅因 fanout 有些微的成長，而非如同運算單元因平行度提高了四倍，面積亦提高四倍。

7-3. 平行度分析

在 Radix-4 的蝴蝶運算單元中，每個單元有 4 個模數乘法單元。而 d＝16 為目前已完成合成的最高平行度硬體系統，系統中共有 64 個模數乘法單元，與 FAST 設計中的 1024 個模數乘法單元相差甚遠。

日後我們將進行更高平行度硬體系統的合成工作。已知面積結果主要由記憶體面積決定，且記憶體面積並不會隨著平行度提高成長，可預期在高平行度下面積結果並不會有明顯的改變。又系統的吞吐量與平行度成正比，同時重新優化蝴蝶運算單元管線化設計以提高運作頻率，將使得我們的最終的 PI 達到 FAST 設計的水準，甚至有所提升。

8. 心得感想

在隱私運算的需求驅動下，我們團隊以 65536 點 Radix-4 NTT 為核心設計 FHE 加速器。專題中我們把 CFNTT 的衝突自由映射與 ping-pong 記憶體結合，在維持 single-port SRAM 的前提下做到連續資料流；更同時把 BFU 管線深化到 18 級，並提供 1/4/16 個 BFU 的可調平行度與併行 I/O，讓吞吐接近線性放大。同時也自建 testbench：實作 twiddle factor 的生成與順序預處理，確保硬體/軟體正確性。過程最大的瓶頸，一是頻率受限在約 200 MHz、PI 落後 FAST 論文；二是 ADFP 無 memory compiler，迫使以小 SRAM 拼大記憶體，面積占比逾九成，暴露出「頻寬、面積、平行度」的三邊權衡。我們團隊因此把輸入輸出由串列改為並列，才避免 I/O 成為新瓶頸。團隊協作上，前期共讀建模、中期並行分工、後期流水線整合，讓我們學會把假設量化、用數據收斂設計。回顧整個專題，我們更理解從演算法到實體電路的映射成本：要追上文獻，仍需管線化/多週期單元與記憶體架構再優化，並在更高平行度下重審 PI。這段經歷，讓我把「能做出來」推進到「做得對、做得好」，更希望能將專題進度一步步帶入碩班，繼續往數位 IC 設計的領域前進。