國立交通大學

資訊科學與工程研究所

碩 士 論 文

區塊式濾波反投影演算法設計與 FPGA 實作 (II)

Block-based Filtered Backprojection Algorithm Design and Implementation on FPGA (II)

研 究 生：林坤世

指導教授：荊宇泰　教授

中 華 民 國　九 十 五　年 九 月

區塊式濾波反投影演算法設計與 FPGA 實作(II)

Block-based Filtered Backprojection Algorithm Design and Implementation on FPGA (II)

研 究 生：林坤世　　Student：Kun-Shih Lin

指導教授：荊宇泰　　Advisor：Yu-Tai Ching

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年九月

區塊式濾波反投影演算法設計與FPGA實作(II)

學生：林坤世　　指導教授：荊宇泰博士

國立交通大學資訊科學與工程研究所

摘要

濾波反投影(filtered backprojection)演算法在電腦斷層掃描扮演相當重要的角色，其計算繁瑣及複雜。我們目標設計一個可高度平行化處理的演算法，由於硬體資源不足，無法對整張 filtered sinogram 做影像重建。因此採用區塊式重建，逐個區塊做影像重建，即為最後結果。經由軟體模擬，模擬結果有大小不同的區塊，我們都得到預期的結果。此論文需參照吳俊緯同學所撰寫的”區塊式濾波反投影演算法設計與實作(I)”。

Block-based Filtered Backprojection Algorithm Design and Implementation On FPGA (II)

Student：Kun-Shih Lin　　Advisor：Dr. Yu-Tai Ching

Department of Computer and Information Science

National Chiao Tung University

Abstract

The filtered backprojection algorithm plays an important role in computational tomography, its calculations are complicated and tedious.

All we want to do is to design a highly parallel algorithm to solve the limitations of hardware resources. Because of lack of resources, we take the block-based policy, doing reconstruction block by block. After the final block has been reconstructed, the block-based FBP is done. By implementing the algorithm we design, we have the expected consequences in different block size. You should refer to the thesis "Block-based Filtered Backprojection Algorithm Design and Implementation (I)" written by Chun-Wei Wu.

誌謝

很快的兩年就過去了，在這段時間裡很感謝我的指導老師 莊宇泰教授對我的照顧，指導有關論文研究的方向以及知識，幫助我順利的完成論文。另外我要感謝張騰介學長指導我一些有關硬體上的常識，以及黃滄智學長、李秉章同學在學業上提供我意見參考，以及所有陪我順利度過研究所生涯的所有同學、學弟們，感謝他們為實驗室帶來了歡笑，讓實驗室變的更有朝氣，最後我要感謝我的 partner 吳俊緯同學，因為有他跟我一起熬夜寫程式及討論，所以我們才能夠一起順利的完成我們的論文。

目錄

圖目錄

第一章 緒論

醫學往往被稱為經驗的科學，經由文字與圖片記錄方式，累積過去所見所聞的經驗與知識，然後才有能力進行診斷、治療，甚至預防。我們可以說醫學的基本信念是基於了解造成病變的成因，才可以預防、診斷與治療。

綜合而言，疾病的定義可以從幾個方面來看：第一是病徵，經由病原所產生的單一症狀；第二是症候群，意指由病原所引起的多種症狀；第三是已知病原所特定攻擊的組織器官；第四是指組織器官的外形變化。其中前面三種是可以經由觀察、測量與研究得知；但是第四種則非得經由直接透視人體內部不可。過去都是透過人體解剖，才有辦法認識與了解各種組織器官的基本外觀；透過病理切片的觀察得以明白細胞組織的病變。因此長久以來，人類一直有夢想，希望能夠以非侵入方式透視人體解剖結構，而電腦斷層掃描系統正是實現此夢想的重要工具之一。

1.1 簡介

電腦斷層攝影（computerized tomography ，簡稱 CT ），是根據人體截面投影來重建該截面的影像，是近幾年來發展最快，並獲得廣泛應用的輻射診斷技術之一。提到電腦斷層掃描，我們就不得不提到它的發明者，在 1972 年英國的 Godfrey Newbold Hounsfield 利用 X 光成功地作出一個電腦斷層攝影系統。它利用一個非常細的 X 光線從不同角度來照射，在物體的另一端有閃爍偵檢器（ scintillation detection ）來測量穿透的放射線，然後將這些資料輸出電腦，經

由數學運算後便可重建出該物體截面的影像。借由這些影像，人們可以清楚的分辨各個器官組織的形狀和大小及相對的解剖位置。它的出現，使得人們首次可以不必經由開刀手術便可觀察到人體內部結構的細節。

這項劃時代重大發明讓 Hounsfield 與建立電腦斷層掃描系統的線積分基礎的物理學家 Alan Cormack，共同分享一九七九年諾貝爾生理或醫學獎，這對沒有受過傳統大學教育的 Hounsfield 及物理學家 Cormack 無疑是極大的殊榮。

用簡單一句話來加以說明：「從基本原理的觀點， 電腦斷層掃描技術就是測量體表每個角度的能量束通過人體所形成的交互作用量，然後應用電腦系統計算，重建出該橫截面的組織特性影像。 」

1.2 研究動機

時至今日電腦斷層掃描技術已經漸漸的成熟，但是醫學專用的電腦斷層掃描儀器由於功能性強大，因此一般而言體積龐大且造價相當昂貴，所以至今仍無法有效的普及化。有鑑於此，我們希望可以藉由半導體快速發展的優勢，設計出一個具備斷層掃描基本功能的晶片，希望可以達到縮小儀器體積以及造價低廉的目的，使得斷層掃描這項技術能夠在醫學之外有其他更廣泛的應用，譬如在機場、港口等地方的貨物安全檢查，或者是應用在生物領域中對動植物的生長作觀察等，使這項技術可以廣泛的應用在各個層面。

而可程式閘陣列(Field Programmable Gate Array，簡稱 FPGA)，正是可達成此一目的的利器，因為 FPGA 可知針對不同的演算法設計進行配置和重新配置，修改電路較為便利，而且設計週期、投入成本與傳統 ASIC 相比亦較低，因此在實作上我們就是採用 Xilinx 公司所出的 ML310 Development Board 發展平台，搭配同樣是 Xilinx 公司所出的 Embedded Development Kit (EDK)平台發展

工具，希望可以設計出一個我們心目中理想的晶片，除了具備斷層掃描基本的功能之外，還可以有體積小以及造價低廉等優勢。

1.3 論文架構

在本篇論文中主要分成四個章節，第一章提及斷層掃描的發明、來源，以及此篇論文的動機。第二章介紹嵌入式晶片設計的流程以及我們使用的平台，Xilinx 公司的 ML310 Development Board。第三章則介紹我們的演算法如何在嵌入式發展平台上實作出來。最後，第四章對我們實做出來的結果做說明、討論，以及未來的發展性等。此外，此篇論文需要參考吳俊瑋同學所撰寫的區塊式濾波反投影演算法設計與 FPGA 實作(I)，如此將有助於對本論文的了解。

第二章 嵌入式單晶片發展平台介紹

嵌入式系統是一個面向應用、技術密集、資金密集、高度分散、不可壟斷的產業，隨著各個領域應用需求的多樣化，嵌入式設計技術和晶片技術也經歷著一次又一次的革新。雖然 ASIC 本身的成本很低，但設計週期長、投入費用高、風險較大，而可編程邏輯器件（Programmable Logical Device）設計靈活、功能強大，尤其是高密度現場可編程邏輯器件（Field Programmable Gate Array）其設計性能已完全能夠與 ASIC 媲美，而且由於 FPGA 的逐步普及，其性能價格比已經幾乎足以與 ASIC 抗衡。因此，FPGA 在嵌入式系統設計領域中已經佔據著越來越重要的地位。

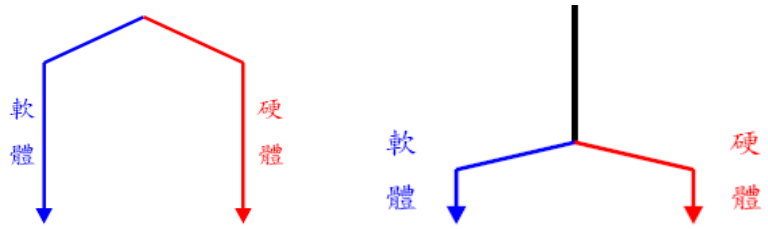
2.1 嵌入式晶片設計演進

FPGA 是一種具有可重配置邏輯閘的晶片，與供應商提供功能定義的 ASIC 晶片不同，FPGA 可根據每個應用的不同需要而進行配置和重新配置。由於 FPGA 允許在硬體中實現自定義算法，因此它具有精確時序和同步、快速決策及平行任務同時執行等優點。

由於硬體製程技術提高，在單一晶片中可容納的電晶體數量亦不斷提昇，因此在考慮晶片功耗，面積及成本上，我們會希望將多個功能完整的電路整合成一顆積體電路，就是系統單晶片(SoC)。整合到系統單晶片的電路可以包括中央處理器、記憶體、傳輸介面，以及使用者自己設計的功能模組等。基本上若可以將上述的所有電路模組完整的放入單晶片中，則該電路就可以滿足人們許多生活上

的需求及應用，因此目前有越來越多的人力陸續投入嵌入式系統的發展。

嵌入式系統SoC採用現場可程式閘陣列(Field Programmable Gate Array ，簡稱FPGA)或專用積體電路(Application Specific Integrated Circuit，ASIC)實現。傳統設計的方法，如圖 2-1(a)所示，會將軟硬體明顯分離獨立設計，最後階段才將軟硬體整合。但是往往由於對彼此設計的東西不甚了解，因此導致修改上的困難度提高，延遲了產品上市的時間。



(a) 傳統設計流程　　(b)軟硬體共同設計流程

圖 2-1 嵌入式系統 SoC 設計流程圖

隨著設計方式的改良，慢慢的出現了軟硬體共同設計(Software/Hardware Co-design)這個名詞。這是一個以系統觀點出發的方法論，強調軟硬體設計時的一致性與整合性，如圖 2-1(b)，對於軟體發展的驗證(Verification)更加容易，且硬體設計的錯誤也能提早發現，降低設計所需的時間與成本，大大縮短系統設計的時間，因應市場的即時性。

2.1 軟硬體共同設計

一般而言，典型的軟硬體共同設計流程，可以用圖 2-2來表示。

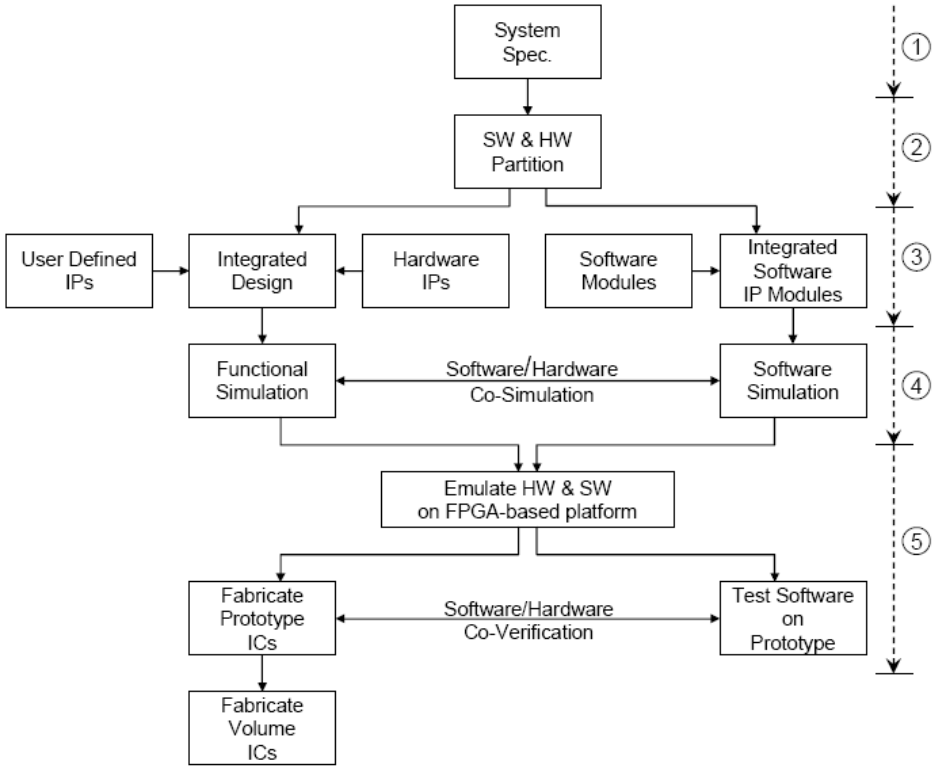


圖 2-2 典型軟硬體設計流程

軟硬體協同設計主要分為5個階段：(1)訂定系統規格、(2)軟硬體分割、(3)軟硬體實作、(4)軟硬體協同模擬，以及(5)軟硬體協同驗證。

- 訂定系統規格(System spec.)：確定系統的目的與可行性，接著訂出此系統需有哪些功能。
- 軟硬體分割(H/W & S/W partition)：依照系統的功能，為了求取最大之效率，可將某些元件作成硬體以產生最快的速度與最多的產出。此時必須評估哪些元件須作成硬體，哪些元件須以軟體方式呈現，為降低整體成本與達到最高效率取得平衡。
- 軟硬體實作：
  - 硬體實作：當我們把硬體規格訂出來後，利用HDL語言描述此硬體並根據板子的規格實作此硬體。
  - 軟體實作：分別撰寫驅動程式與使用此硬體之軟體，整合系統產生之標

頭檔與使用者之檔案。

- 軟硬體協同模擬(Software/Hardware Co-Simulation)：在使用硬體的地方以軟體模擬硬體工作後得到的資訊，並以之作為輸入以驗證軟體的正確性。
- 軟硬體協同驗證(Software/Hardware Co-Verification)：實際將軟硬體放到模擬板上驗證是否可以正確工作。

在本篇論文中會將我們的 FBP 演算法利用可程式閘陣列(FPGA)來實現，設計流程也將模仿上述的軟硬體共同設計的方式來完成我們的 FBP 演算法嵌入式系統的實作。

2.3 嵌入式系統發展平台

在我們的實作中採用的是Xilinx公司生產的嵌入式系統發展平台—ML310 Development Board，如圖 2-3所示：

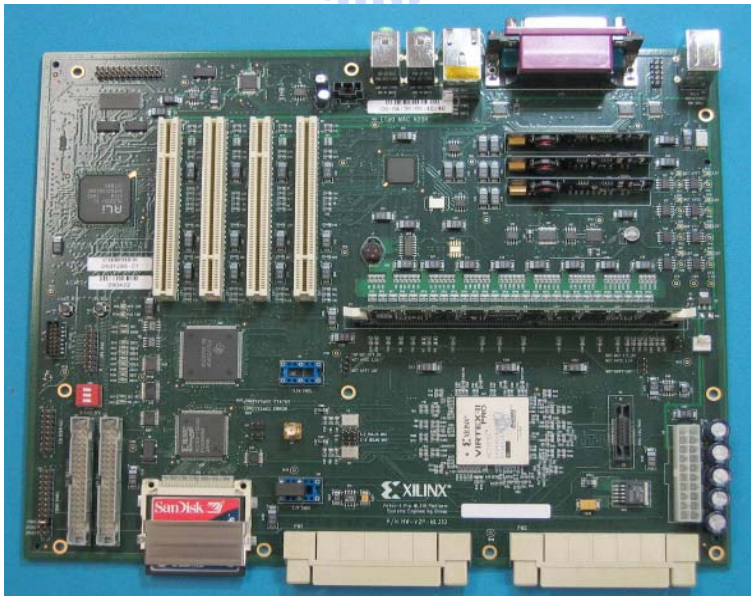


圖 2-3 Xilinx 公司的 ML310 Development Board

平台的核心是一顆 Virtex-Ⅱ pro FPGA(Field Programmable Gate Array)，

型號是 XC2VP30。這顆 FPGA 共有 30,816 Logic Cells 提供邏輯合成的電路資源，且具有 2 顆內建的 PowerPC 405 處理器，可以當作嵌入式系統的核心處理器。此外，FPGA 內尚有內建的 18x18 bits 的乘法器 136 顆及 136 塊 18Kbits 的 Block RAM 提供現成的電路元件。

除了 FPGA 外，發展平台提供一個嵌入式系統相當完整的發展環境，除了電源與一些 I/O 介面，並利用匯流排將核心模組與邏輯模組整合在一起。平台上重要的邏輯模組像是 DDR Memory，System ACE CF Controller，CompactFlash Slot，IDE Drive Connectors，PCI Slots 等。

平台上主要的匯流排是 Process Local Bus(PLB)與 On-chip Peripheral Bus(OPB)，PLB bus 與 OPB bus 是 IBM CoreConnect 的標準匯流排，其中 PLB bus 是高速匯流排，而 OPB bus 是低速匯流排，兩個匯流排中間有個 Bridge 作連接，形成整個嵌入式系統的骨幹。圖 2-4 是以 PowerPC 405 處理器為核心的嵌入式系統架構圖。

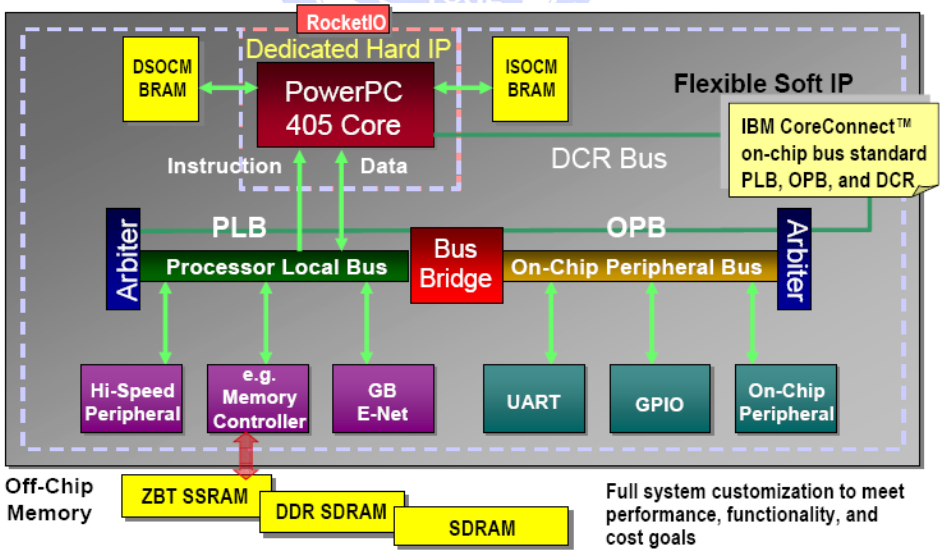


圖 2-5 以 PowerPC 405 為核心的嵌入式系統架構

我們配合 ML310 平台使用的發展工具是 Embedded Development Kit(EDK)及 Integrated Software Enviroment(ISE)。EDK 提供以 PowerPC 405 為核心的嵌入式系統設計者豐富的設計工具及多樣的嵌入式系統週邊設備，如 Memory

Controller 、 UART(Uni-versal Asynchronous Receiver/Transmitter) 、 GPIO(General Purpose Input/Output)等。EDK 發展工具主要包括下面幾部分：

- 嵌入式系統核心及週邊元件的硬體 IP(Intellectual Property)。
- 嵌入式系統軟體所需要的驅動程式(Drivers)，函式庫(Libraries)。
- 具有整合軟硬體研發環境的 Xilinx Platform Studio(XPS)。

其中 Xilinx Platform Studio 中有幾個重要的發展工具，在我們的 FBP 演算法嵌入式系統實作中將使用到，所以下面簡單介紹這些工具：

- The Base System Builder(BSB) wizard：這是一個軟體工具可以幫助使用者快速的建立特定平台上可以執行的基礎嵌入式系統。在執行完 BSB wizard 後，會產生兩個重要的檔案，一個是硬體規格檔(Microprocessor Hardware Specification，MHS)，另一個是軟體規格檔(Microprocessor Software Specification，MSS)。MHS 檔案定義系統的架構及使用到的週邊元件及核心處理器。也定義了系統中週邊與核心元件間匯流排的連接關係，以及各個週邊元件的參數設定。MSS 檔案則是定義週邊元件的驅動程式，系統的標準輸入輸出設備，系統中斷處理常式及相關的軟體設定。
- The Platform Generator Tool(PlatGen)：輸入 MHS 檔案，建立系統的 netlist 檔案，並支援檔案下載到 FPGA 時所需的相關檔案，及新增週邊元件到系統中的相關工作。在執行 PlatGen 工具時，ISE 將自動被呼叫去完成整個硬體平台的邏輯合成，電路佈局到產生 netlist 檔案。
- The Library Generator Tool(LibGen)：輸入 MSS 檔案，建立個人化週邊元件驅動程式、函式庫、系統中斷處理常式及檔案系統。
- The Bitstream Initializer tool：在 FPGA 中對處理器將存取的指令記憶體區塊進行初始化設定，而指令記憶體區塊則存在 Block RAM 中。這個工具將讀入 MHS 檔案，並且呼叫 ISE 的 Data2MEM 工具來初始化 FPGA 中使用到的

Block RAM。

- Create/Import Peripheral Wizard：幫助使用者建立自己的週邊元件並且加入嵌入式系統中。

此外，在實作過程中我們還有用到 Xilinx 公司生產的 ISE 軟體，ISE 是套 IC 設計的整合軟體環境，包含多個軟體套件，可以執行完整的設計流程。從硬體描述語言(Hardware Description Language)程式的編寫(如 VHDL 及 Verilog)，設計的輸入，邏輯模擬，邏輯合成，電路佈局，產生 netlist 檔案，產生可以載入 FPGA 配置電路的 BitStream 檔等工作。

第三章 FBP 演算法嵌入式系統實作

在這個章節中我們要介紹的是整個 FBP 演算法的實作過程，採用的硬體裝備是 ML310 Development Board，搭配 EDK 發展工具來完成我們的實作。圖 3-1 是 EDK 這套軟體的設計流程，採軟硬體共同設計的方式。

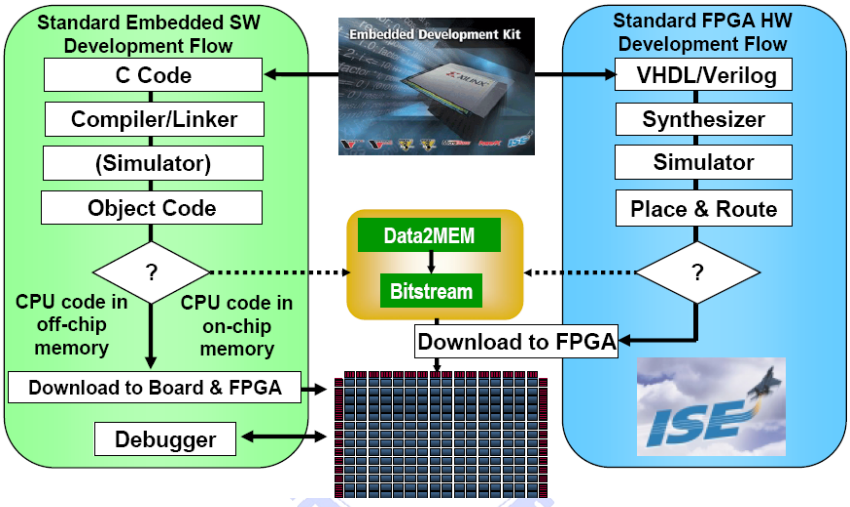


圖 3-2 EDK 設計流程概觀

由此流程圖中可以看出，其實我們也可以直接採用 Standard Embedded SW Development Flow 此種設計方式來得到我們要的結果，也就是直接在 EDK 這套軟體上撰寫 C 語言，經編譯後 Download to FPGA，這樣一樣可以讓整個過程順利運作，並產生正確的結果。但很不幸的 PowerPC 405 處理器的運算頻率並不高，尤其相較於現今的 CPU 頻率，更是毫無優勢可言，因此並不符合我們的期望，所以這樣一來我們勢必要把重複性高、運算次數多的部份利用硬體來實作，充分發揮硬體執行效率較高的優勢。所以接下來將會介紹我們自己的 FBP 演算法如何在硬體上面實作出來。

3.1 FBP 演算法分析

在介紹我們的 FBP 演算法實作過程之前，有一件重要的工作，那就是必須先對將要實作的 FBP 演算法做分析，簡單的說就是先將演算法分成幾塊，再去決定哪些要用軟體而哪些要採硬體實作，圖 3-2 即是將我們演算法做最初步的區分。

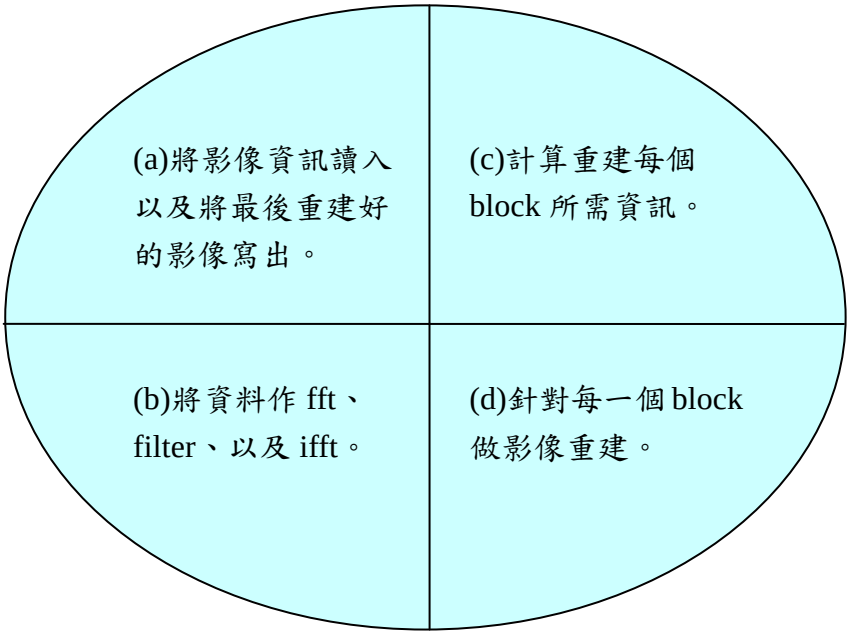


圖 3-2 演算法分析示意圖

圖 3-2 主要將我們的實作過程分成四大部分，而我們的目的是希望將那些運算量高、時間耗費大的部分以硬體來實作，而由於圖 3-2 (a)本身是處理記憶卡(CompacFlash card)與軟體間的資料傳輸，因此自然是直接以軟體來實作。至於圖 3-2 (c)則是計算重建每個 block 所需資訊，這個部分由於運算次數以及所耗時間在可容忍範圍內，因此為增加實作的便利性，我們也將以軟體直接實作。接下來的圖 3-2 (b) 、(d)由於所需的運算量較高且亦較耗時，因此自然成為我們實作成硬體的重點部份。

3.2 FBP 建立及系統設定

在這一小節中，我們將簡單介紹整個系統初步的相關設定，包含 processor、I/O interfaces、軟體及硬體參數設定，以及所用到的 IP(Intellectual Property)。

3.2.1 專案建立

利用 Base System Builder Wizard 選擇 New Design，選定使用版子及版本，然後選擇 processor 為 PowerPC 且維持 processor 的相關設定，接下來 configure I/O interfaces 選擇有用到的部分包括 RS232_Uart、DDR_SDRAM_32Mx64 以及 SysACE_CompacFlash。

3.2.2 建立 FBP 元件及連結其週邊

這裡使用 EDK 軟體所提供的 Create/Import Peripheral Wizard 幫我們建立自己的週邊元件，首先用 Create Peripheral wizard 建立 FBP 元件。關於 FBP 元件的硬體描述，我們將最上層電路的 VHDL 模組，取名為 fbp，然後將 FBP 元件連接上 PLB Bus，如圖 3-3 所示。

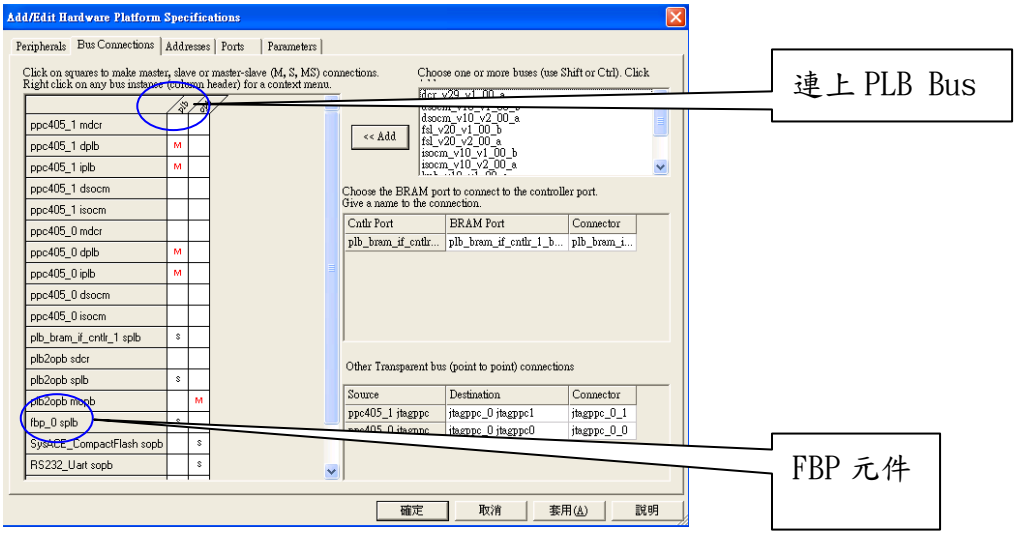


圖 3-3 Add/Edit Hardware Platform Specifications

到目前為止我們訂下了整個系統最初步的架構，包含了 FBP 元件以及各個週邊元件的連結方式，其中 FBP 元件代表的就是要將程式實作成硬體的部份。圖 3-4 是此一架構的示意圖及其說明：

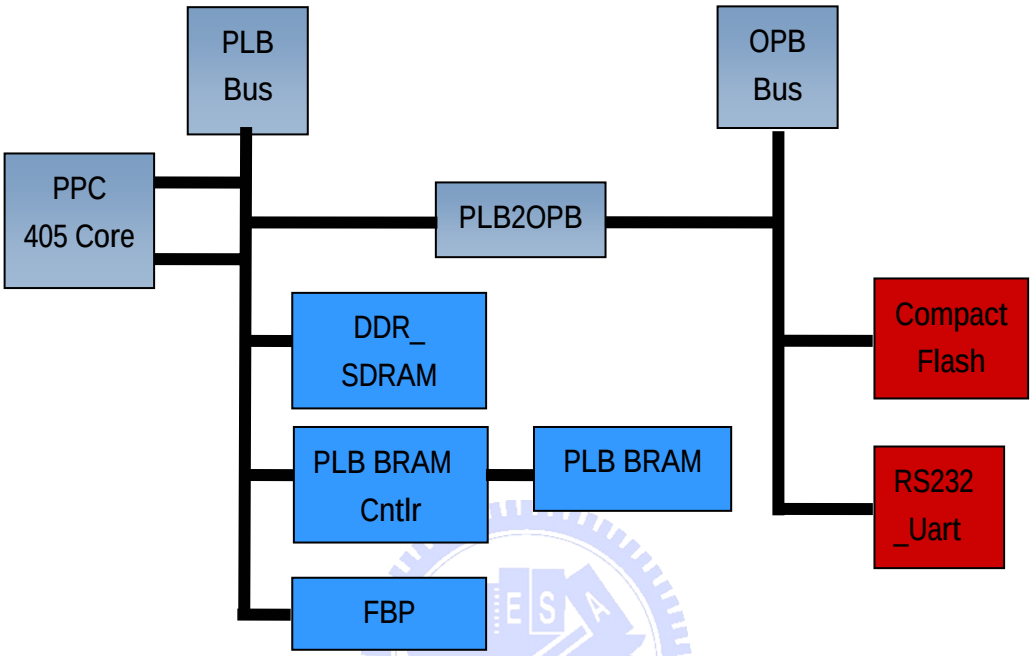


圖 3-4 系統架構圖

- CompactFlash 與 RS232_Uart：

RS232_Uart 的功用是可以讓我們透過 terminal 程式經由 serial port 與系統溝通。而 Compact Flash 則讓我們可以透過 System ACE CF Controller 控制 CompactFlash card，作為 sinogram 影像及完成重建後影像的儲存設備。如下圖所示，其中 Input sonogram image 以及 Output reconstructed image 都是利用此一 CompactFlash card 來完成。

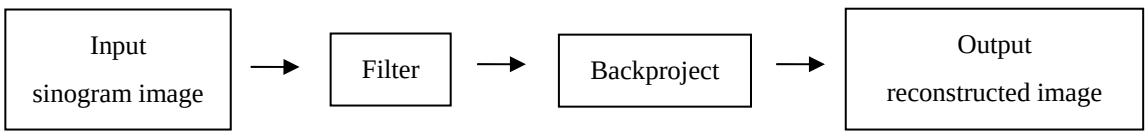


圖 3-5 FBP 簡易流程圖

而由於我們的 FBP 程式要能讀取 CompactFlash card 上的 sinogram 影像以及

將重建結果回存，所以我們需要一個可以讀寫 CompactFlash card 的函式庫。在這部分 Xilinx 公司有提供一個名稱為 xilfatfs 的函式庫可以幫助我們達成此功能，其使用方式只要在 EDK 的 Software Platform Settings 中勾選此函式庫，並在軟體端的程式中包含 xsysace.h 及 sysace_stdio.h 兩個標頭檔，如此一來即可順利讀寫 CompactFlash card。此外，由於 RS232_Uart 及 SysAce_CompactFlash 本身是比較慢的週邊設備，因此將之與 OPB Bus 連接。

- SDRAM、BRAM 以及 FBP

因為 FBP 將是未來實作的主要部分，所以當然希望此元件能獲得較高的效率，因此將之與 PLB 相連接。至於 SDRAM、BRAM 都是 memory 的一部分，本身就屬於高速的設備，因此與 PLB Bus 連接。有關 BRAM 的部分我們會在介紹 IP 時再一併介紹。

當將整個架構定義出來之後我們可以看出 FBP 元件是與 PLB 相連，但是他們之間到底是如何溝通訊息呢，下面我們再針對這部分作一個介紹：

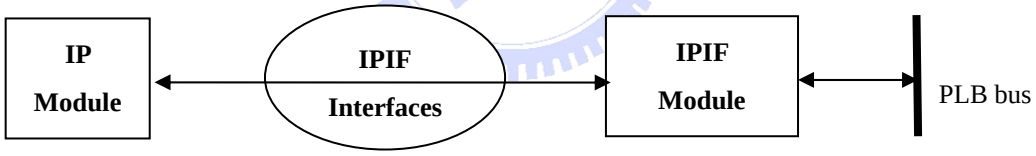


圖 3-6 IPIF 與 IP 模組及 PLB bus 之間的關係

由圖 3-6 我們可以看出，FBP 元件與 PLB bus 之間的溝通方式其實是利用 EDK 所提供的 IP Interface(IPIF)框架，它可以提供使用者建立的 IP 模組與 on-chip buses 之間的溝通介面服務,主要包含兩個介面，分別是對 PLB bus 的介面及對 IP 模組的介面。

利用 IPIF 提供的溝通介面服務，我們可以決定使用者的元件與 Bus 間資料的溝通方式，主要提供了下列幾項：

- S/W Reset and Module Information register(RST/MIR)
- Burst and Cacheline Transaction Support
- DMA
- FIFO
- User Logic Interrupt Support
- User Logic S/W Register Support
- User Logic Master Support
- User Logic Address Range Support

在我們的設計中選擇了 S/W Reset and Module Information register(RST/MIR) 及 User Logic S/W Register Support 這兩項來滿足我們的基本需求，其主要功能就是讓我們定義出 32 個 64bits 的暫存器供我們使用，透過這些暫存器，我們可以將資料從軟體經由寫入暫存器然後傳送至硬體端的 FBP 元件供其內部作運算，相對地也可以將 FBP 元件運算完的結果經由暫存器取送回給軟體，簡單的說就是可以利用此 32 個暫存器來作為軟體跟硬體間溝通的橋樑。

至於實際使用上我們的 fbp 模組需要透過 IPIC 定義的訊號與 IPIF 連接才能夠與 PLB bus 作溝通，在此 IP Interconnect (IPIC)指的是 IPIF 與使用者 IP 模組連接的訊號集合，也就是說在實作上我們必須了解 IPIC 中所定義訊號的意義，如此一來才可以掌握在不同時脈下資料的流動情形，譬如何時該將資料傳入、何時資料開始運算以及將寫出資料等，下面一併附上在我們設計中有使用到的 IPIC 訊號定義：

Signal Name	I/O	Description
Bus2IP_Clk	O	IPIC clock provided to IP. This PLB_CLk
Bus2IP_Reset	O	Active high reset for use by the User IP.

Bus2IP_Data(0:C_IPIF_DWIDTH-1)	O	Write data bus to the User IP. Write data is accepted by the IP during a write operation by assertion of the IP2Bus_WrAck signal and the rising edge of the Bus2IP_Clk.
Bus2IP_BE(0:(C_IPIF_DWIDTH/8)-1)	O	Byte enable qualifiers for the requested read or write operation with the User IP. Bit 0 corresponds to Byte lane 0, Bit 1 to Byte lane 1, and so on.
Bus2IP_RdCE(0: see note 1)	O	Active high chip enable bus. Chip enables are assigned per the User's entries in the C_ARD_NUM_CE_ARRAY. These chip enables are asserted only during active read transaction requests with the target address space and in conjunction with the corresponding sub-address within the space.
Bus2IP_WrCE(0: see note 1)	O	Active high chip enable bus. Chip enables are assigned per the User's entries in the C_ARD_NUM_CE_ARRAY. These chip enables are asserted only during active write transaction requests with the target address space and in conjunction with the corresponding sub-address within the space.
Bus2IP_RdReq	O	Active high signal indicating the initiation of a read operation with the IP. It is asserted for 1 Bus2IP_Clk during single data beat transaction and remains high to completion on burst write operations.
Bus2IP_WrReq	O	Active high signal indicating the initiation of a write operation with the IP. It is asserted for 1 Bus2IP_Clk during single data beat transaction and remains high to completion on burst

		write operations.
IP2Bus_Data(0:C_IPIF_DWIDTH-1)	I	Input Read Data bus from the User IP. Data is qualified with the assertion of IP2Bus_RdAck signal and the rising edge of the Bus2IP_Clk.
IP2Bus_Retry	I	Active high signal indicating the User IP is requesting a retry of an active operation. This signal is currently unused by the PLB IPIF.
IP2Bus_Error	I	Active high signal indicating the User IP has encountered an error with the requested operation. This signal is asserted in conjunction with IP2Bus_RdAck or the IP2Bus_WrAck.
IP2Bus_ToutSup	I	Active high signal requesting suppression of the data phase time-out function in the IPIF for the active read or write operation.
IP2Bus_RdAck	I	Active high read data qualifier. Read data on the IP2Bus_Data Bus is deemed valid at the rising edge of Bus2IP_Clk and the assertion of the IP2Bus_RdAck signal by the User IP.
IP2Bus_WrAck	I	Active high write data qualifier. Write data on the IP2Bus_Data Bus is deemed accepted by the User IP at the rising edge of the Bus2IP_Clk and IP2Bus_WrAck asserted high by the User IP.

表 3-1 PLB IPIF I/O Signals

完成 FBP 元件建立後，會產生兩個重要的檔案，分別是 fbp.vhd 及 user_logic.vhd。其中 fbp.vhd 是 FBP 元件的最上層 VHDL 模組，而 user_logic.vhd 則是我們程式實作成硬體的地方。接下來我們必須把建立出來的 FBP 元件加入整個設計之中，所以利用 EDK 內的 Import Peripheral Wizard，執行後 EDK 軟體會將 FBP 元件的資訊記錄在 MHS 檔案中，也就是前面提到的硬體規格檔(Microprocessor Hardware Specification)。

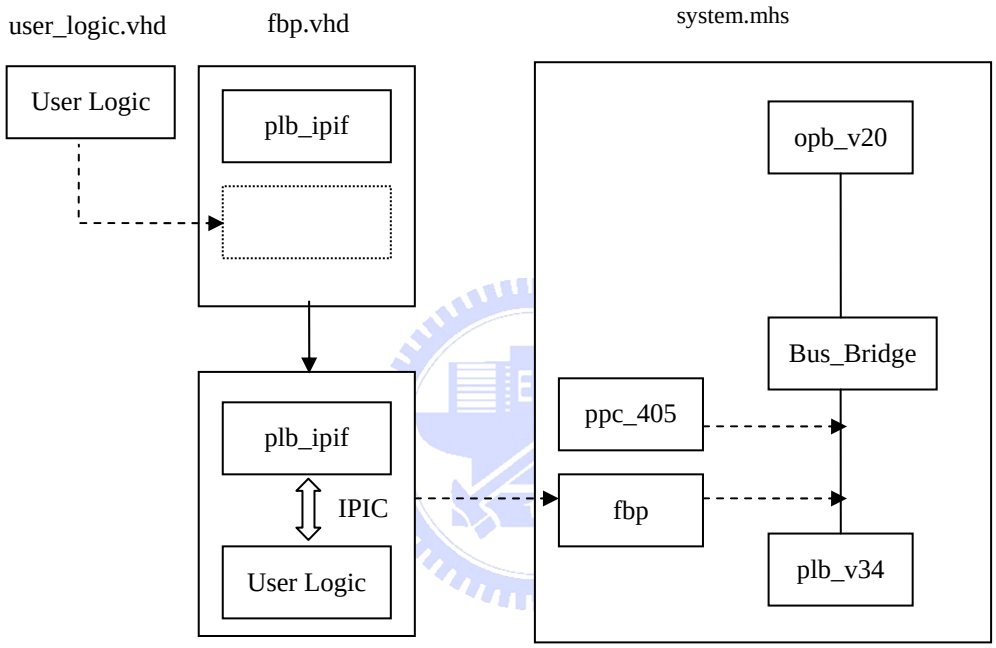


圖 3-7 將 FBP 元件加入設計之示意圖

在將 FBP 元件加入整個設計之中以後，基本上我們就完成了整個架構的建立了，而接下來我們還會繼續介紹有關 IP 的使用以及 FBP 元件的實作等部份。

3.3 使用 IP 介紹

在實作我們的 FBP 演算法過程中，我們將會使用到兩個由 Xilinx 公司所提供的 IP，一個是 Single-Port Block Memory v6.1 IP，另一個則是 Fast Fourier Transform v3.1 IP。其中 Single-Port Block Memory 主要用途是可以幫我們規劃出一塊可以使用的記憶體空間，而 Fast Fourier Transform 本身則是一個可完成 FFT 及 IFFT 功能的 IP，下面我們將依序介紹這兩個 IP。

3.3.1 Single-Port Blcok Memory IP

ML310 Board 中的 FPGA 有 136 塊 18Kbits 的 Block Memory。由於 Block Memory 是 on-chip memory，因此在存取速度上會比 off-chip memory 要快。我們在設計 FBP 元件時需要一次讀進 sinogram 影像中某一列的所有 pixel 灰階值。因此我們需要在我們的設計中增加一個記憶體空間來儲存這些讀入的資料。Signal-Port Block Memory V6.1 IP 可以幫我們使用 FPGA 中的 Block Memory 規劃出一塊可以供我們使用的記憶體空間。

此 Single-Port Block Memory V6.1 IP 的重要特色如下：

- 支援 Read-Only Memory 及 Random-Access Memory 的功能
- 根據選擇的硬體架構，記憶體大小的設定範圍由 2 到 2M words 之間。
- 可以針對 memory 的儲存內容作初始化。
- 記憶體寫入的模式有 Read-after-Write，Read-before-Write 及 No-Read-on-Write 三種模式。

底下是 Signal-Port Block Memory V6.1 IP 的線路符號代碼，其中黑色的部分代表在我們的設計中有使用到的訊號，灰色則是沒有選用到的訊號：

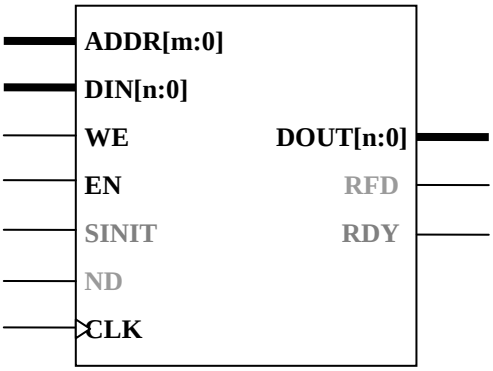


圖 3-8 Signal-Port Block Memory 訊號接腳圖

Port Name	Direction	Description
DIN[n:0] (Optional)	Input	**Data Input**: Data written into memory.
ADDR[m:0]	Input	**Address**: Memory location for data written to/read from.
WE (Optional)	Input	**Write Enable**: Allows data transfer into memory.
EN (Optional)	Input	**Enable**: Enables access to memory via read and write operations.
CLK	Input	**Clock**: All memory operations synchronous with rising or falling edge of clock input, depending on user configuration of the clock pin polarity. When memory is enabled, all control signals, input/output data are registered on the rising or falling edge of clock.
DOUT[n:0]	Output	**Data Output**: Synchronous output of the memory.

表 3-2 Single-Port Block Memory V6.1 IP Core Pinout

表 3-2 是針對我們有用到的接腳做一個說明，而 Single-Port Block Memory V6.1 IP 的功能大致上是這樣，當 Block Memory 的致能訊號 EN 是 1 時，所有記憶體的動作將發生在時脈訊號 CLK 的上升緣(Rising Edge)或是下降緣(Falling Edge)變化時。當致能訊號 EN 為 0 時，則記憶體的設定及輸出訊號 DOUT 將不變。當讀出的動作在致能訊號 EN 為 1 時，輸出訊號 DOUT 將輸出位置訊號 ADDR 所指定的記憶體位置內的值。當寫入訊號 WE 為 1 時，在輸入訊號 DIN 的值將存在位置訊號 ADDR 所指定的記憶體位置。而剛剛提到寫入的模式有三種，

Read-after-Write，Read-before-Write 及 No-Read-on-Write。這是輸出訊號的行為在記憶體為寫入的狀態時，有三種不同的輸出模式。分別敘述如下：

- Read-after-Write 是指在同一個時脈中，在輸入訊號 DIN 中的值除了存到指定的記憶體位置外也直接傳到輸出訊號 DOUT 中當成輸出。
- Read-before-Write 則是同一個時脈中，位置訊號 ADDR 所指定的記憶體位置內的值將傳送到輸出訊號 DOUT 中當成輸出。
- No-Read-on-Write 是指記憶體在寫入的狀態時，輸出訊號 DOUT 維持原狀不作改變。

在我們的設計中，Single-Port Block Memory V6.1 IP 的設定上我們選擇 No-Read-on-Write 的模式。記憶體的讀寫動作設定為發生在脈訊號 CLK 的上升緣。而因為 FFT 及 IFFT 的過程中有 Zero Padding 的動作，所以以 128x128 影像大小為例，我們將記憶體大小設定為 256 words(128*2)，每個 word 是 32bits，且 word 的初始值設定為 0。至於 SINIT、ND、RFD 及 RDY 訊號，我們並不需要因此設定上選擇不用，以簡化設計。我們參考到的時序圖如下：

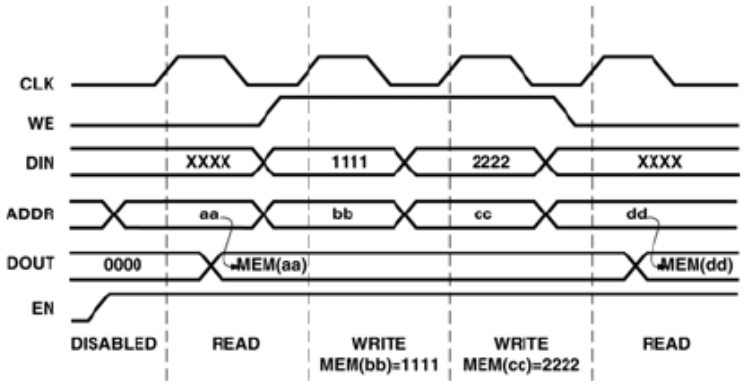


圖 3-9 No-Read-on-Write Mode Waveform

3.3.2 Fast Fourier Transform v3.1 IP

FBP 元件的另一個功能必須包含 FFT 運算及 IFFT 運算。Xilinx 公司也有提

供 FFT 及 IFFT 的 IP 可以供我們使用，我們不需從頭設計 FFT 及 IFFT 的電路。Xilinx 公司提供的 Fast Fourier Transform v3.1 IP 的重要特色如下：

- 此 IP 可以在執行狀態中設定為 FFT 或 IFFT 運算。
- FFT及IFFT可以支援的取樣點數目大小是$N=2^m$，m= 3 到 16。
- 每一筆輸入資料的精確度b_x可以是 8、12、16、20、24 位元寬度。
- FFT運算過程中使用到的phase factor精確度 b_w可以是 8、12、16、20、24 位元寬度。
- 計算過程中計算的形式有：

  -Unscaled (full-precision) fixed point
  -Scaled fixed point
  -Block floating point
- 在 butterfly 運算過程中產生的數值可以使用 Rounding 或是 Truncation 的方式進位到最後一位。
- IP 的硬體架構有三種選擇：

  -Pipelined, Streaming I/O

  -Radix-4, Burst I/O

  -Radix2, Minimum Resource

底下是 Fast Fourier Transform v3.1 IP 的線路符號代碼，黑色的部分表示我們有使用到的訊號，灰色則代表在我們的設計中沒有選用這些訊號：

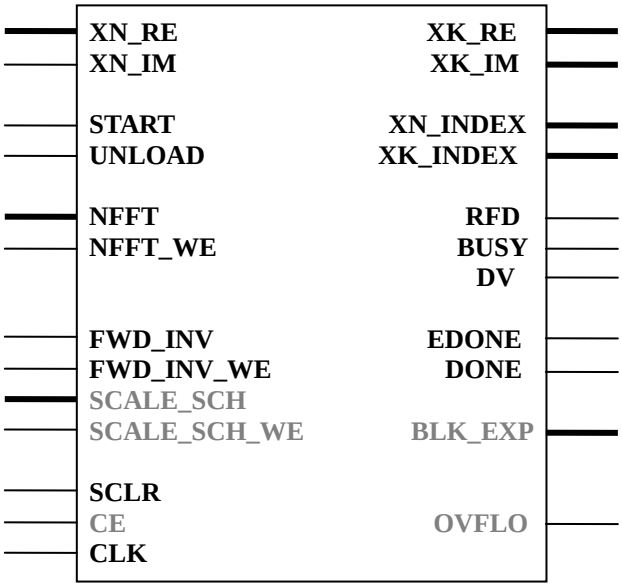


圖 3-10 Core Schematic Symbol

Port Name	Port Width	Direction	Description
XN_RE	**b_{xn}**	Input	Input data bus: Real component (bxn=8,12,16,20,24)
XN_IM	**b_{xn}**	Input	Input data bus: Imaginary component (bxn=8,12,16,20,24)
START	**1**	Input	FFT start signal(Active High): START is asserted to begin the data loading and transform calculation (for the Burst I/O architecture). For continuous data processing, START will begin data loading, which processing directly to transform calculation and the data unloading.
UNLOAD	**1**	Input	Result unloading (Active High): For the Burst I/O architecture, UNLOAD will start the unloading of the results in normal order. The UNLOAD port is not necessary for the Pipelined, Streaming I/O architecture.
NFFT	**5**	Input	Point size of the transform: NFFT can be the size of the transform or

			any smaller point size. For example, a 1024-point FFT can compute point size 1024, 512, 256, and so on. The value of NFFT is $\log_2$ (point size). This port is optional for Pipelined, Streaming I/O architecture.
NFFT_WE	**1**	Input	Write enable for NFFT (Active High): Asserting NFFT_WE will automatically cause the FFT core to stop all processes and to initialize the state of the core. This port is optional for Pipelined, Streaming I/O architecture.
FWD_INV	**1**	Input	Control signal that indicates if a forward FFT or an inverse FFT is performed. When FWD_INV=1, a forward transform is computed. If FWD_INV=0, an inverse transform is performed.
FWD_INV_WE	**1**	Input	Write enable for FWD_INV (Active High).
SCLR	**1**	Input	Master reset (Active High): Optional port.
CLK	**1**	Input	Clock
XK_RE[(B-1):0]	**b_{xk}**	Output	Output data bus: Real component.
XK_IM[(B-1):0]	**b_{xk}**	Output	Output data bus: Imaginary component.
XN_INDEX	**Log_2(point size)**	Output	Index of input data.
XK_INDEX	**Log_2(point size)**	Output	Index of output data.
RFD	**1**	Output	Ready for data (Active High): RFD is High during the load operation.
BUSY	**1**	Output	Core activity indicator (Active High): This signal will go High while the core is computing the transform.

DV	**1**	Output	Data valid (Active High): This signal is High when valid data is presented at the output.
EDONE	**1**	Output	Early done strobe (Active High): EDONE goes High one clock cycle immediately prior to DONE going active.
DONE	**1**	Output	FFT complete strobe (Active High): DONE will transition High for one clock cycle at the end of the transform calculation.

表 3-3 Fast Fourier Transform v3.1 IP Core Pinout

圖 3-10 針對我們有用到的訊號線做一個說明，另外 Xilinx 公司提供的這個 IP，可以設定為執行 FFT 或是 IFFT 運算。此 FFT 是一個計算離散傅立葉轉換 (Discrete Fourier Transform，DFT)高效率的演算法。但是注意的是 FFT 使用的函式是 DFT

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-jnk2\pi/N} \quad k=0\ldots N-1 \tag{3.1}$$

而 Fast Fourier Transform v3.1 IP 執行 IFFT 運算使用的函式是

$$x(n)=\sum_{k=0}^{N-1}X(k)e^{jnk2\pi/N} \quad n=0\ldots N-1 \tag{3.2}$$

並不是和函式(3.1)對應的IDFT(inverse Discrete Fourier Transform)

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)e^{jnk2\pi/N} \quad n=0\ldots N-1 \tag{3.3}$$

換句話說，如果作 128 筆取樣值的 FFT 轉換接著作 IFFT 轉換後，得到的資料將比原取樣值大 128 倍，這是我們使用此 IP 要注意的地方。

此 IP 的內部架構有三種形式可以供我們選擇，說明如下：

- Pipelined, Streaming I/O：此架構可以連續性的處理資料，也就是讀入資料，做轉換及資料的輸出可以同時執行。此架構下執行 FFT 或是 IFFT 運算所需的時間最少，但是此架構下 IP 要合成硬體所需要的硬體資源也是最多的。

此架構使用的傅立葉演算法為 Radix-2 butterfly。

- Radix-4, Burst I/O：此架構下執行 FFT 或是 IFFT 運算有兩個階段，一個是資料的輸入輸出，另一個是資料的轉換運算過程。此架構下所需要的硬體資源會比 Pipelined, Streaming I/O 架構要少，但運算的時間會增加。內部使用的傅立葉演算法為 Radix-4 butterfly。
- Radix2, Minimum Resource：此架構跟 Radix-4, Burst I/O 架構一樣有資料的輸入輸出及轉換運算過程兩個階段，但是因為內部的傅立葉演算法為 Radix-2 butterfly。因此整個運算的時間最長，但所需要的硬體資源也最少。

另外，執行 FFT 或是 IFFT 運算時，在 radix-4 或是 radix-2 butterfly 轉換過程中，數值可能會增大，因此考慮可能發生位元溢位的狀況下。運算的過程中數值的計算有三種形式：

- Unscaled (full-precision) fixed point：整個計算過程，不針對數值作任何比率的縮小，此情形在數值位元寬度不足時可能發生溢位狀況。
- Scaled fixed point：在作 FFT 或是 IFFT 會有多個 radix-4 或是 radix-2 butterfly 轉換過程，每個過程都作比率的縮小，縮小的倍數可能是 1、2、4、8，代表的是將數值作位元右移操作。每個過程要縮小的倍數定義在 SCALE_SCH 訊號中。
- Block floating point：此計算形式是指完成 FFT 或是 IFFT 運算後，由 IP 自動決定要縮小幾倍，而縮小的倍數會儲存在 BLK_EXP 訊號中。

同樣的上述的三種數值計算形式也會影響到此 IP 合成硬體所需要的資源。

在我們的設計中，必須由軟體端將取樣值經由暫存器傳至此 FBP 元件再由 Fast Fourier Transform v3.1 IP 執行 FFT 運算，因此 Pipelined, Streaming I/O 架構無法跟我們的設計配合，因此我們選擇 Radix-4, Burst I/O 架構，使得資料的輸出入跟資料作傅立葉轉換分成兩階段。另外，我們處理的資料是 pixel 的灰階值，

範圍是 0~255，只要 8bits 就可以表達。但是之前我們提到要使用 fix point 的方式來處理 float point 的資料，因此將會乘上 2 冪次方倍數來保留運算中產生的小數位的資訊，再加上考慮計算過程中可能發生的溢位問題，因此在輸入資料的位元寬度及 phase factor 我們設定為 24 位元寬度。資料輸出則由 IP 自動設定為 33 位元寬度。為保留計算中數值所代表的資訊完整性，我們選擇 Unscaled (full-precision) fixed point 計算形式，避免縮小倍數的同時而失去部分資訊。至於我們要執行多少取樣數的 FFT 及 IFFT 運算呢？前面我們提到原本的 128 pixels 的灰階值在執行 FFT 運算前要先作 Zero Padding 的動作，因次我們將取樣數設為 256 筆。這樣我們就完成 Fast Fourier Transform v3.1 IP 的參數設定。

底下是我們參考到的時序圖：

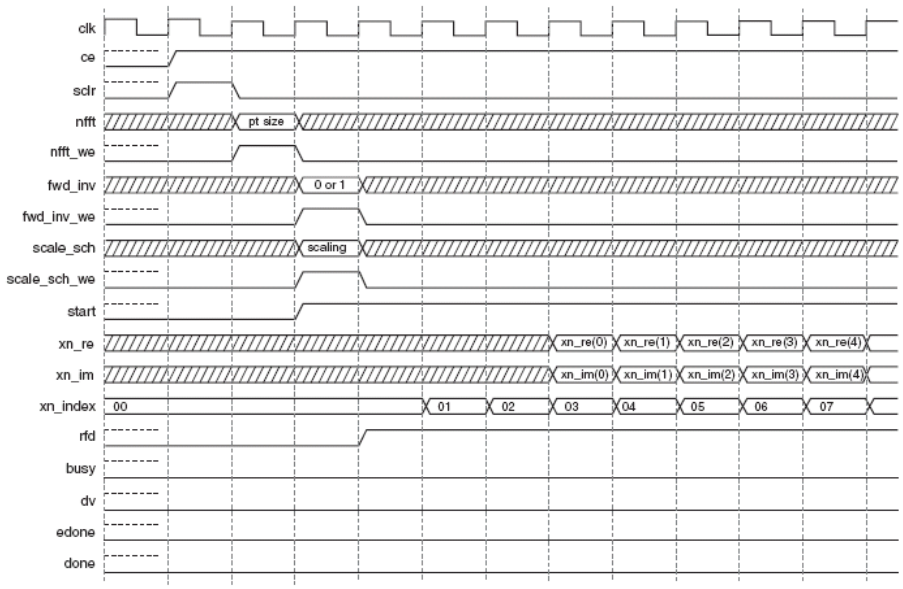


圖 3-11　IP 開始輸入資料的波形圖

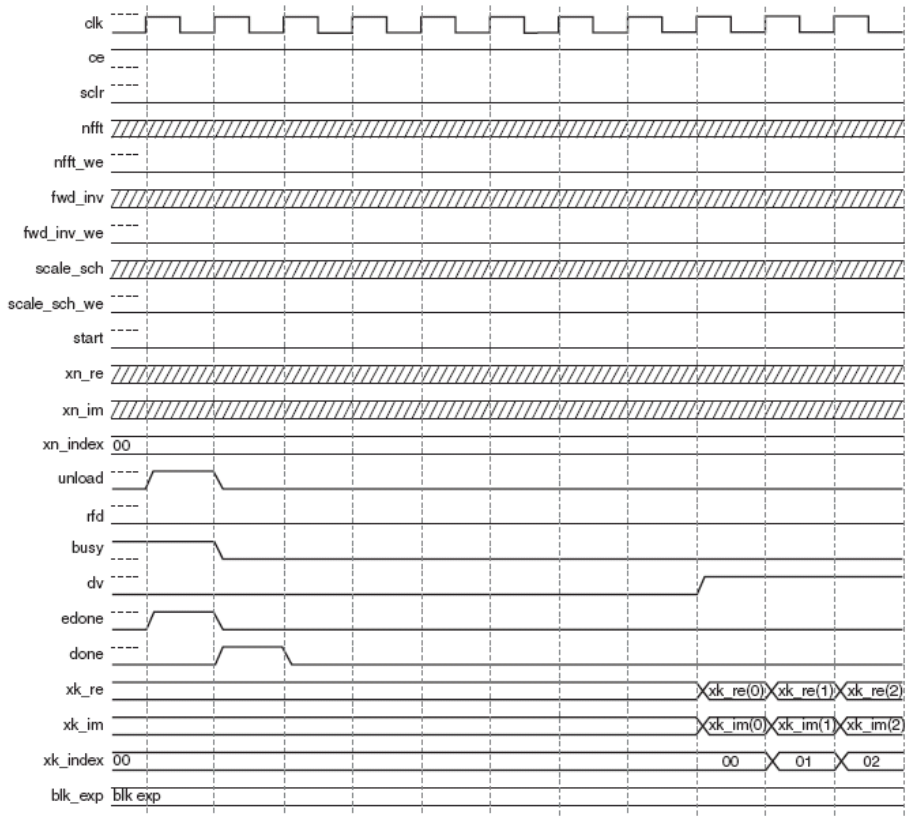


圖 3-12　IP 輸出資料的波形圖

其中要注意的是當 start 訊號拉成 1 時，從下一個 clock 開始的第四個 clock 才能輸入資料到 Fast Fourier Transform v3.1 IP 中。當 dv 訊號為 1 時，則開始輸出傅立葉轉換後的資料。

3.4 FBP 元件實作說明

接下來我們要介紹的是實作的核心部分，也就是 user_logic 模組實作，這個部分主要是由五大部分所組成，如圖 3-13 所示，每個部分各有其不同的作用且意義分述如下：

- IDLE：為剛開始的初始狀態，且之後當程式執行至此狀態時則不做任何事，等待新的命令執行。

- LOAD_IN：主要的功能是藉由 32 個 64bit 的暫存器將軟體端的資料儲存到由 Signal-Port Block Memory 這個 IP 所合成的 Block RAM 中，以便在硬體上作運算。
- FILTER：這個狀態所做的工作包含將讀入的資料作 fft、filter、以及 ifft，完成後的資料及可拿來做反投影(back projection)。
- LOAD_OUT：主要的功能是藉由 32 個 64bit 的暫存器將硬體端的資料搬出到軟體端。
- BP：這部分主要的工作就是將狀態 FILTER 完成後的資料在硬體裡作反投影的工作。

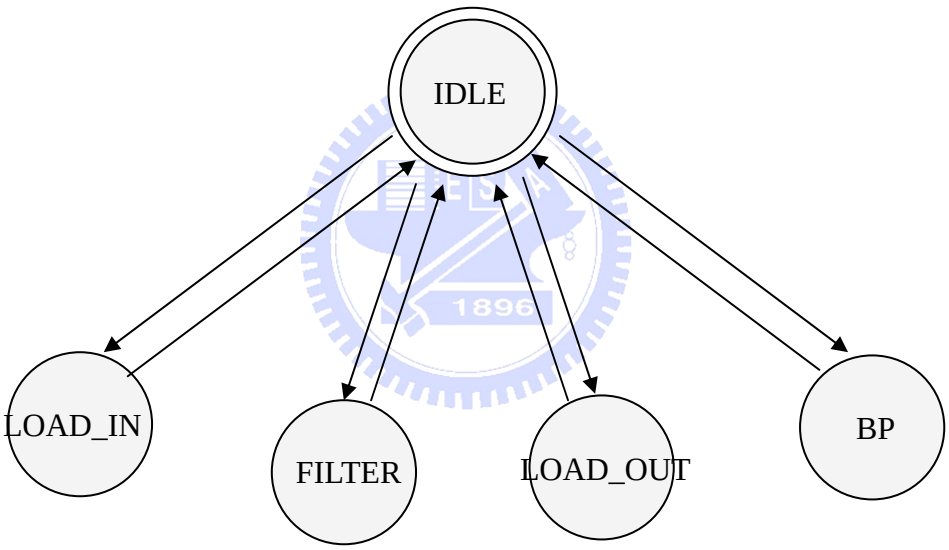


圖 3-13 user_logic 模組狀態圖

繼續我們在針對狀態圖中的 FILTER 以及 BP 兩個比較重要的部分作一個比較詳細的介紹：

圖 3-14 是 FILTER 狀態的一個詳細說明圖，剛開始由於資料尚存在 Block RAM 中，因此我們需要有一個狀態 fft_load 將這些資料一筆筆 load 至 Fast Fourier Transform v3.1 IP 邏輯合成的電路中，之後狀態 fft_transform 開始作傅立葉運算，接下來進入狀態 filter_operate 將傅立葉運算完的值乘與濾波器$|\omega|$，待整個運算

完成後再利用 fft_unload 將運算完結果寫回 Block RAM 中，之所以要先寫回 Block RAM 而不直接繼續做反傅立葉運算的原因是因為當初我們在 IP 的架構設定上是採用 Radix-4, Burst I/O，主要理由是為了減少硬體資源的消耗。接下來同樣的做法利用 ifft_load 狀態將資料 load 進 IP 中，狀態 ifft_transform 作反傅立葉運算，最後 ifft_unload 狀態下將運算完成的資料寫回 Block RAM 中以待後續處理。

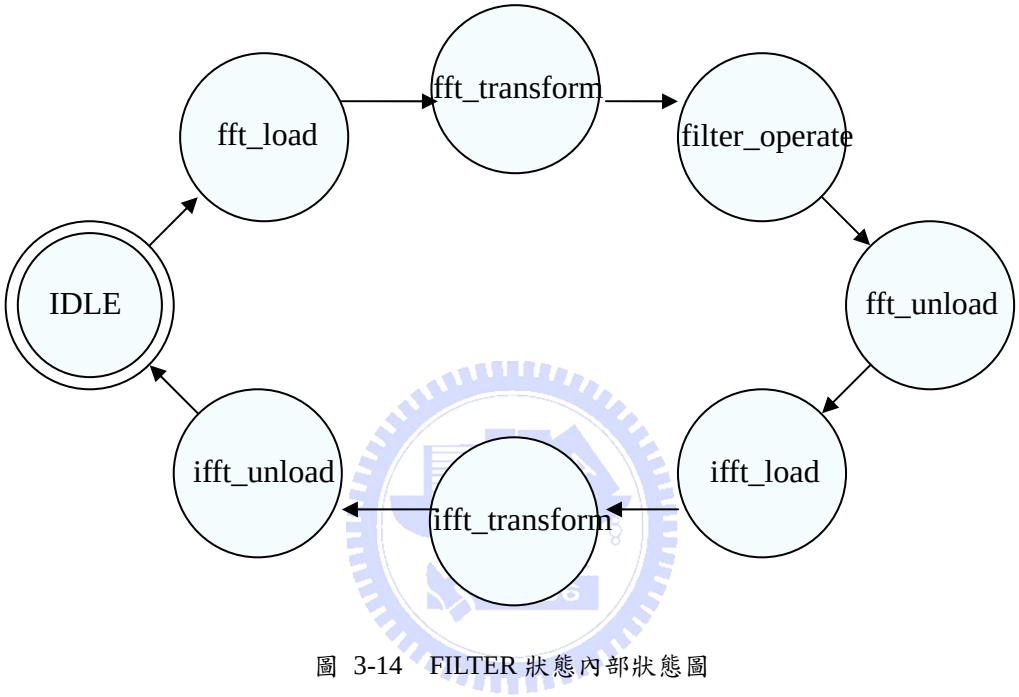


圖 3-14　FILTER 狀態內部狀態圖

而由於我們沒辦法將所有經 fft、filter、ifft 處理過後的資料全部存放在 Block RAM 中，因此在這種情況下我們會將狀態跳至 LOAD_OUT，將處理過後的資料寫回暫存器，由軟體讀回資料。

完成 FILTER 工作後，接下來就是要做反投影，而由於我們是採分塊式來實作，因此必須先於軟體端計算出每一個區塊其對應的 sub_sinogram 區段以便拿來作反投影的動作。以一個 4x4 大小的區塊為例，請參考圖 3-15，依照我們的演算法會在對應的 sub_sinogram 區段上由 block_sino_start 至 block_sino_end 取出七個要反投回去的點，暫時稱之 filtered_real(i，j)，其中 i 指第幾次投影而 j 是 sinogram 上的 index，另外還需求出該點所在之座標 x_coordinate、y_coordinate。

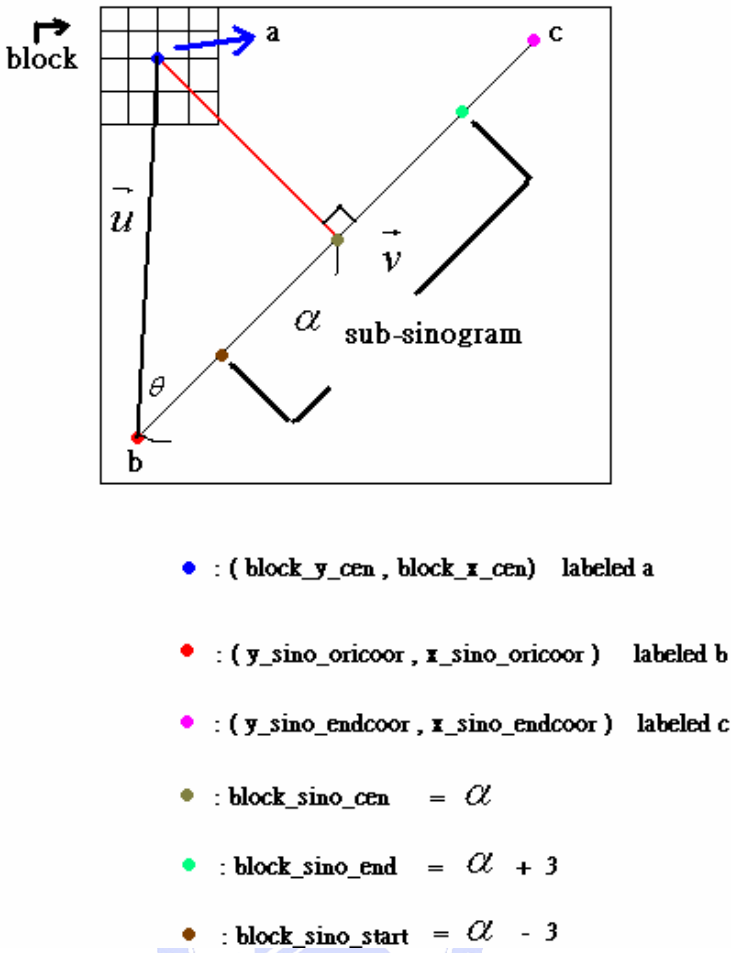


圖 3-15　求出 sub_sinogram 及反投影點示意圖

當所要的資料都求出後，我們就開始編碼動作，將 x_coordinate、y_coordinate 以及 filtered_real(i , j)以 bit 形式編入 Xuint64 此種資料型態之中，Xuint64 可分成 upper 以及 lower 兩部分且分別由兩組 Xuint32 所組成，我們將 filtered_real(i ， j) 置於 Xuint64 中 upper 的部分，而將 x_coordinate、y_coordinates 合併置於 Xuint64 中 lower 的部分，每筆資料之所以要佔這麼多 bits 的原因是因為在硬體中並不支援浮點數的運算，因此我們必須把資料乘以 2 的 10 次方，以盡量保持資料的正確性。圖 3-16 就是編碼的一個示意圖，完成後靠 32 組每組 64bit 的暫存器將 Xuint64 此種資料型態的資料送進硬體端等待重建，如此一來可以一次將三種資料傳送至硬體而不需要每種資料各傳一次，可以減少資料傳送所耗費的時間，此外在作重建的過程中我們還需要知道每個角度的 sin、cos 值，為避免浪費硬體的

資源所以我們不在硬體端使用查表方式來取得，而是在軟體端先取得其值後，同樣使用暫存器將資料送至硬體。

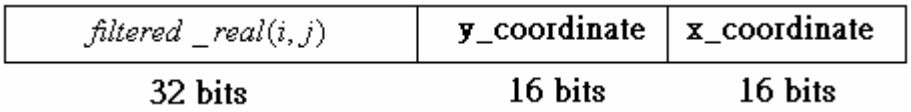


圖 3-16　編碼示意圖

當資料被送達硬體端後，狀態 block_bp_start 就會被啟動並送出一個觸發訊號，同時啟動狀態 even_bp_start 以及 odd_bp_start，如圖 3-17 所示。其中 even 及 odd 是指對於要反投影回去的七個點分成奇數及偶數，然後各自做反投影的動作，所以這裡我們需要兩份相同的系統資源，一旦接受到由 block_bp_start 發送出的觸發訊號就各自進入 decode 狀態，開始解碼動作。而在硬體中考慮到資料正確性及盡量節省資源的原則，我們採用三組 std_logic_vector(0 to 15)來分別儲存由軟體端傳過來的 x_coordinate、y_coordinates 以及 filtered_real(i , j)，在硬體端取出這些值後，解碼動作就算完成了。

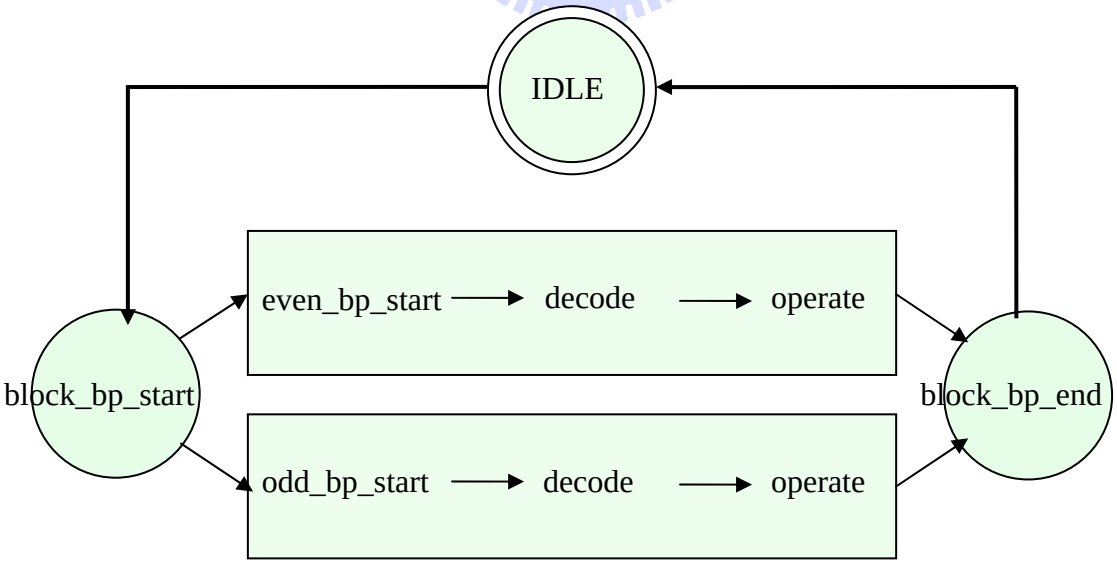


圖 3-17 BP 狀態內部狀態圖

接下來便進入 operate 狀態，而在這之前我們先由圖 3-18 簡單介紹一下硬體中的 block 是如何重建。以單個點的反投影為例，在完成解碼後我們可以得到 sub_sinogram_cen 的 x_coordinate、y_coordinates 以及 filtered_real(i，j)，之後利用加上一個 $\Delta k \times \sin\theta$ 及 $\Delta k \times \cos\theta$ 的偏移量得到下一個位置，如果其範圍仍在(0,0)與(3,3)之間，則我們就取新的 x_coordinate、y_coordinate 之整數座標，並將 filtered_real(i , j)累加至該新座標上。

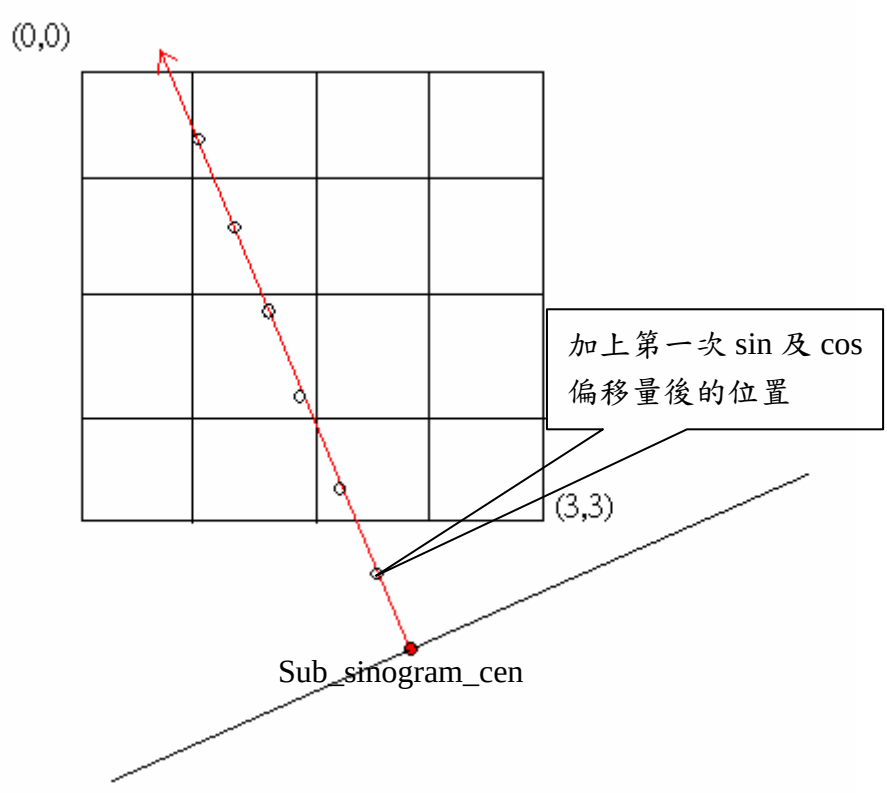


圖 3-18　block 重建示意圖

圖 3-19 說明了硬體中 operate 狀態的整個流程，為了減少 I/O 次數，在重建過程中我們會一次將此 block 所有角度的投影都做完，也就是從 0 度開始做到 127 度，並且在硬體中累加，而不是做完一個角度的投影就將結果回傳至軟體。一旦 even 或 odd 某一邊完成反投影的工作就會送出一個完成訊號給 block_bp_end，等到兩邊都完成重建後我們就將兩者結果相加，並將最後結果回傳至軟體，完成單個 block 之重建。因此每當硬體寫回結果，就是完成了單個 block 整個的重建工作，如此依序做完所有 blocks，就可以得到整張影像重建後的結果。

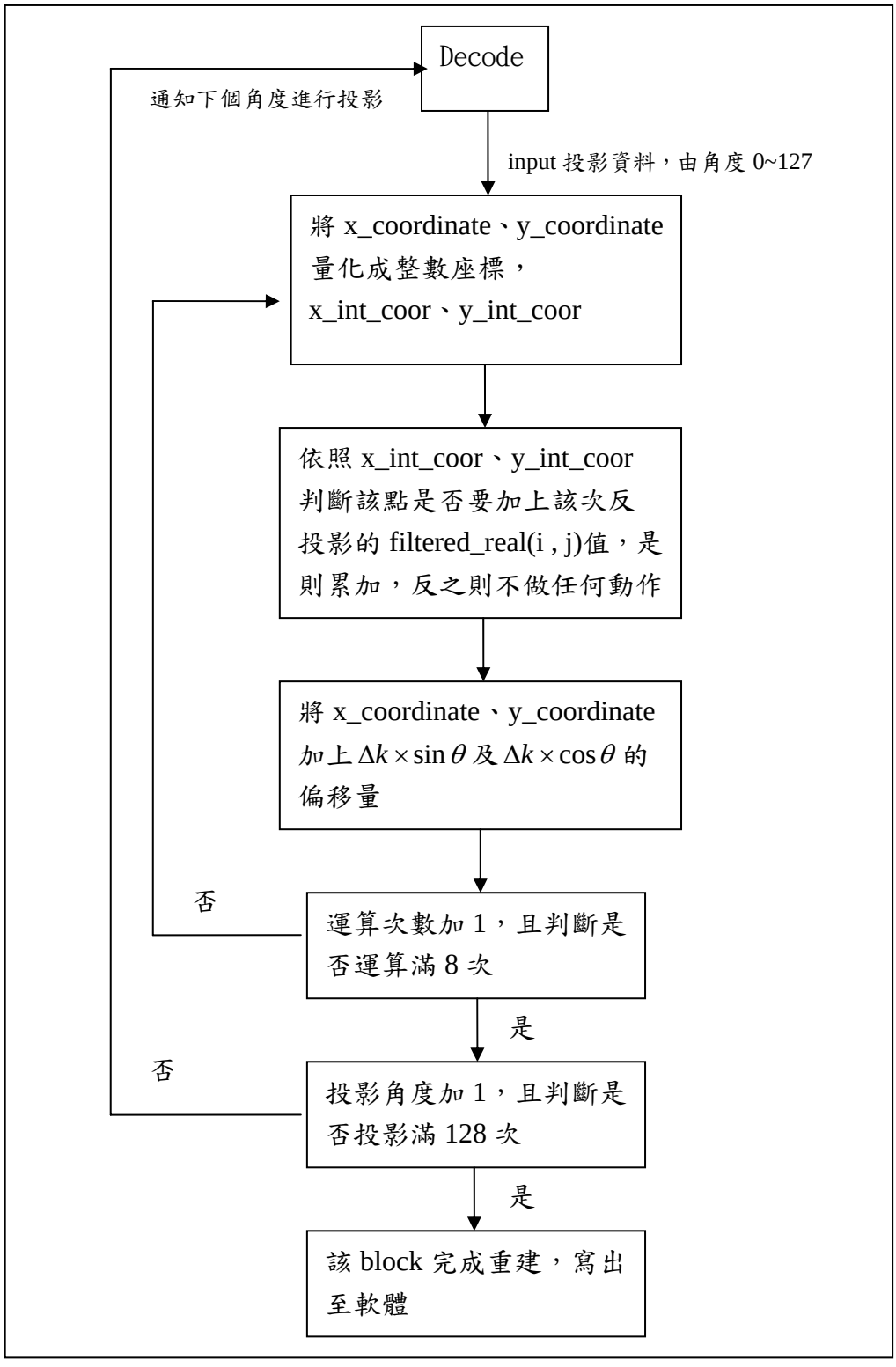


圖 3-19　operate 狀態解說圖

第四章 結果討論與未來展望

在第三章中我們介紹了如何去建立 FBP 的周邊元件、周邊元件的功能與作用、FBP 元件與 PLB bus 之間的溝通方式、IP 的使用，以及最重要的 user_logic 模組實作內容，有了這些東西以後，接下來我們便可以讓程式完整的執行。

4.1 產生實驗結果

圖 4-1 是執行程式的整個流程，首先我們必須先在 EDK 這套軟體上執行軟體端的程式，若執行成功則會產生一個副檔名.elf 的執行檔，接下來則執行硬體端的程式，首先是 Generate Netlist，然後執行 Generate Bitstream，此時會產生一個名為 system.bit 的檔案，最後再執行 Update Bitstream 產生新的 download.bit 檔，配合我們寫的一個批次檔，內容如下：

```
xmd -tcl genace.tcl -hw ../implementation/download.bit -elf ../fbp/executable.elf
-ace ../data/ace/fbp_fin.ace -board ml310
```

表 4-1　合併軟硬體執行檔之批次檔內容

這個批次檔主要的作用就是將.elf 檔以及 download.bit 檔合併，也就是將軟體與硬體分別執行後的檔案做結合，產生最後的可執行檔，我們暫時將之命名為 fbp_fin.ace，此時只要將 fbp_fin.ace 檔案以及用來重建的 sinogram 影像複製至 CompacFlash card 上，則當我們啟動 ML310 Development Board 時，就可以利用電腦的終端機視窗來下指令去執行 CompacFlash card 上的 fbp_fin.ace 檔，如此就

可以順利的跑完整個程式，重建回原來真實的影像。

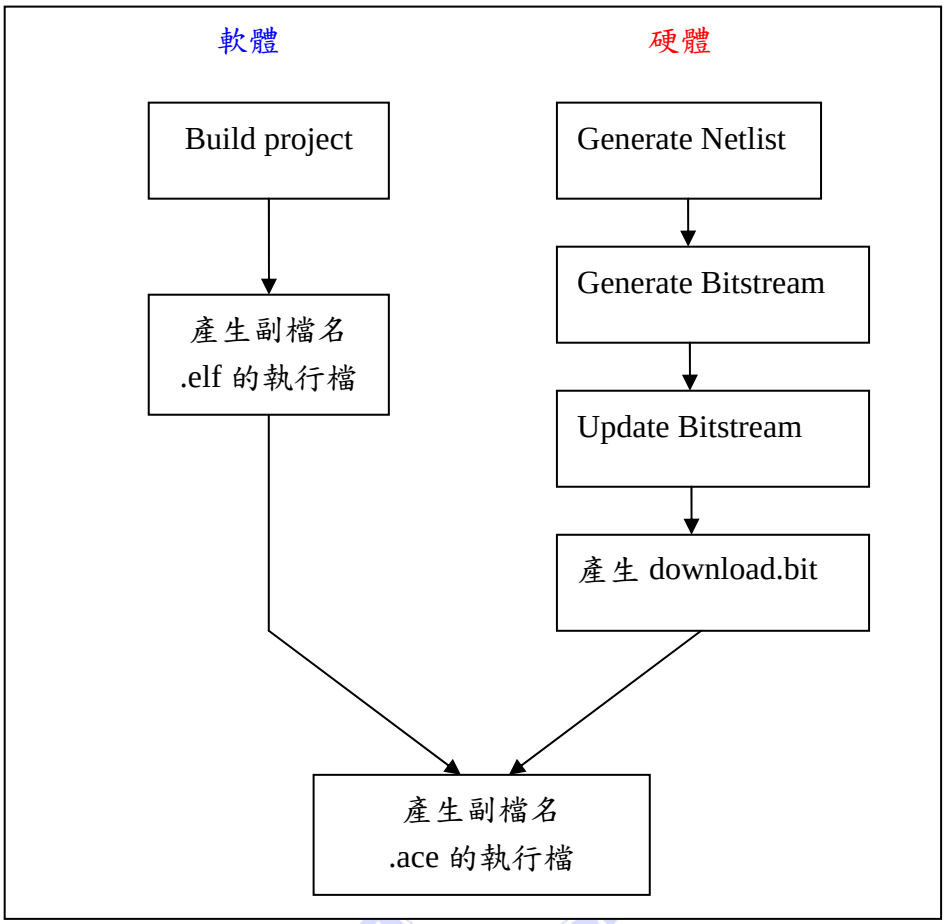


圖 4-1　產生實驗結果之流程

4.2 結果比較與討論

在這個小節裡，我們將會對軟硬體共同設計的方法以及純粹使用軟體實作 FBP 演算法的嵌入式系統做一個比較，表 4-2 是兩者間的比較結果，由表中我們可以看出，利用軟硬體共同設計的方式會比單獨使用軟體實作嵌入式系統來的有效率許多，以 128x128 pixels 大小為例，可以將執行時間由使用軟體實作嵌入式系統的三分鐘左右縮短至不到 5 秒，在效能上算是有不錯的提升。

	比較項目	Filter 時間	Backproject 時間
128x128	軟硬體共同設計	約 1 秒	約 4 秒
	以軟體實作嵌入式系統	約 2 分鐘	約 1 分鐘
256x256	軟硬體共同設計	約 1 秒	約 30 秒
	以軟體實作嵌入式系統	約 10 分鐘	約 8 分鐘

表 4-2　效能比較表

下面我們附上實作過程所佔用的電路資源及系統的時脈等相關資訊。

- 時脈資訊：

  Minimum period：　9.951ns (Maximum frequency: 100.492MHz)

  Maximum path delay from/to any node：2.220ns

- FPGA 資源使用率

  Number of MULT18X18s：　38 out of　136　27%

  Number of Block RAMs：　76 out of　136　55%

  Number of occupied Slices：　11,313 out of　13,696　82%

圖 4-2 是我們以 128x128 pixels 大小的”head phantom”圖重建的結果，其中圖 4-2(a)是原圖，圖 4-2(b)是圖(a)的 sinogram 影像，而圖 4-2(c)是以軟體利用 4x4 block _size 大小重建回來的影像，4-2(d)則是以硬體利用 4x4 block _size 大小重建回來的影像。

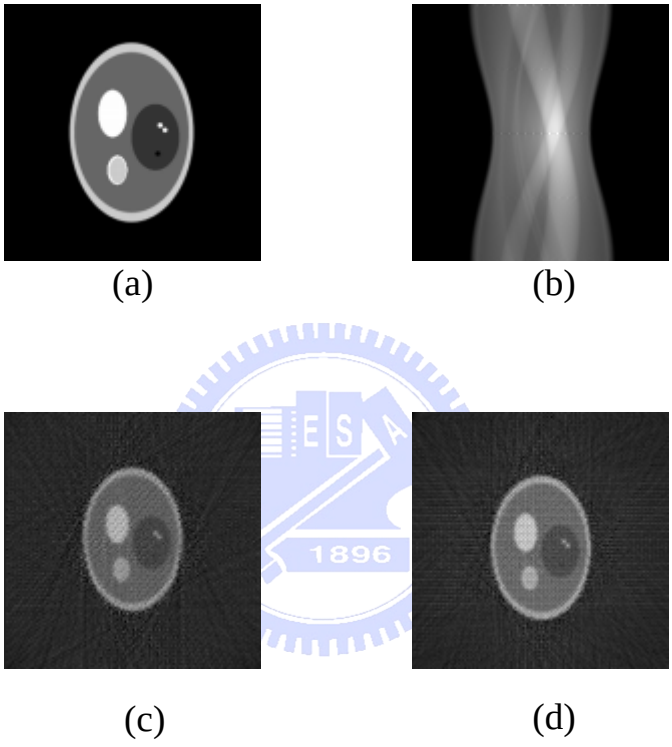


圖 4-2　128x128 原圖以及重建結果

圖 4-3 則是我們以 256x256 pixels 大小的”head phantom”圖重建的結果，同樣的，其中圖 4-2(a)是原圖，圖 4-2(b)是圖(a)的 sinogram 影像，而圖 4-2(c)是以軟體利用 4x4block _size 大小重建回來的影像，4-2(d)則是以硬體利用 4x4 block _size 大小重建回來的影像。

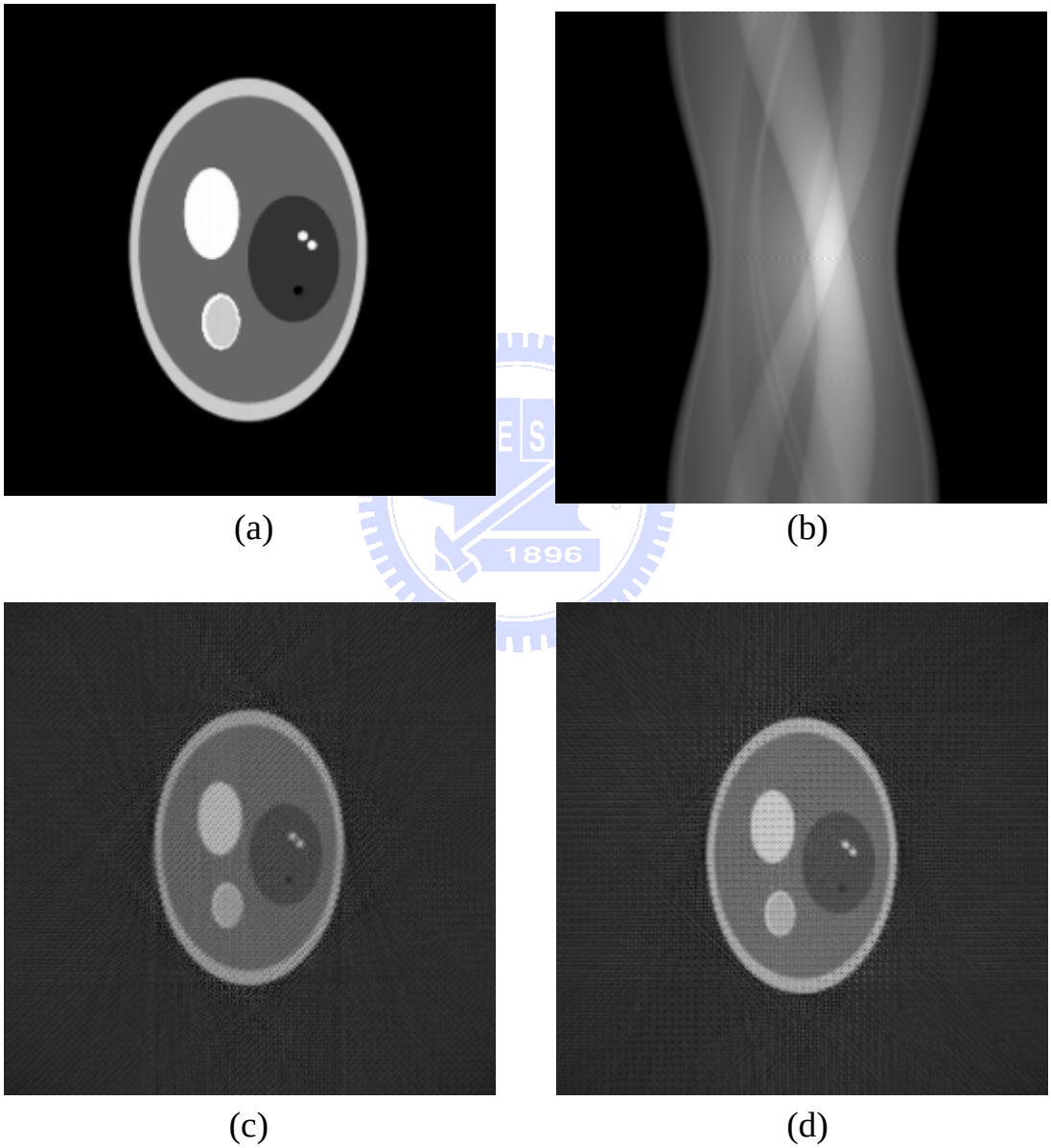


圖 4-3　256x256 原圖以及重建結果

4.3 結論與未來展望

關於實驗的結果，以 128x128 pixels 大小為例，在效能上重建影像所需要的時間大約需要花到 4~5 秒，其中大約有一秒花在軟體計算上，其餘時間皆花費在資料傳輸與硬體計算上，看起來其實是還有改善的空間有，但由於受限於硬體資源所致(主要是 Number of occupied Slices 一項，使用率約 82%)，我們的演算法在硬體實作上只能採用 block size=4x4 的大小，而由軟體的經驗得知，如果我們可以採用較大的 block size，在效能上不管是軟體的運算、資料傳輸與硬體計算上都可以有大幅的減少，更重要的是可以減少重建回來影像上的 artifacts，能夠更接近真實的影像。

因此在未來的方向上，我們可以朝向如何簡化實作上的硬體資源，或者是利用有更多硬體資源可直接使用的版子，如此一來就可以採用較大的 block size 來實作此一演算法，更可提高平行處理的能力，充分發揮其效能及改善重建品質，設計出我們心目中理想的晶片。

參考文獻

[1] Alexandro M.S. Adario, Eduardo L. Roehe, Sergio Bampi, "Dynamically Reconfigurable Architecture for Image Processor Applications", DAC99, New Orleans, Louisiana, ACM, 1999

[2] Didier LATTARD, Guy MAZARE, "Image reconstruction using an original asynchronous cellular array", ISCAS IEEE, 1989

[3] Anup B. Sharma, Keith R. Allen and Roy P. Pargas, "Some new systolic designs for two-dimensional convolution", ACM, 1988

[4] Bei-Chuan Chen, Yu-Tai Ching, "A new antialiased line drawing algorithm", Computers & Graphics, 2001

[5] Bertil Schmidt, Manfred Schimmler, Heiko Schroder, "Tomographic Image Reconstruction on the Instruction Systolic Array", Computers and Artificial Intelligence, 1995

[6] A.V. Lakshminarayanan, "Reconstruction from divergent ray data," tech. rep., Dept.Computer Science, State University of New York at Buffalo, 1975.