國立臺灣大學電機資訊學院電機工程學系(所)

碩士論文

Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

JPEG-LS編碼器之平行管線化架構實作

A Parallel Pipelined Implementation for JPEG-LS Encoders

林家興

Chia-Hsiang Lin

指導教授：王勝德 博士

Advisor: Sheng-De Wang, Ph.D.

中華民國九十八年七月

July, 2009

誌謝

這份論文的完成代表我的碩士求學生涯即將告一段落了，最感謝的是我的指導教授王勝德博士，不論在修課上或者碩士論文上，老師都適時給予建議、關心及指導，老師做事情嚴謹的態度更是我學習的好榜樣，謝謝老師細心的指導讓我的論文可以順利完成。

感謝實驗室的同學勝帆、文暉、學弟博文、博班的宜青、建吉學長、正一學長、慶華學長，謝謝你們在課業上及論文上的幫助與指導，也在工作上分享彼此的經驗，讓我在研究所生涯上過的更加充實難忘；。

還要感謝我的家人，給我的最愛雅容，謝謝妳一直陪伴在我身旁支持我、鼓勵我，陪我熬夜做論文；給志嘉和品愷:「我終於有時間陪你們去小人國了!」給淑英:「謝謝妳，有妳的支援讓我的求學路程更順遂」。給我的爸比、媽咪、及母親大人:「感謝你們在我的求學路程當中，全力支持我，讓我無後顧之憂。」

最後謝謝所有的師長、同學、朋友們，因為有你們讓我的人生更加豐富，祝福你們永遠健康快樂。

中文摘要

隨著醫學電子的蓬勃發展，在醫學影像的處理上勢必越來越受重視，而且畫素也會隨著科技的進步來不斷的提升，造成圖片尺寸越來越大，適當運用無失真壓縮技術不但能方便儲存管理圖片，也可以使用在可攜式的電子醫學儀器上，讓使用者方便儲存以及讀取圖片。

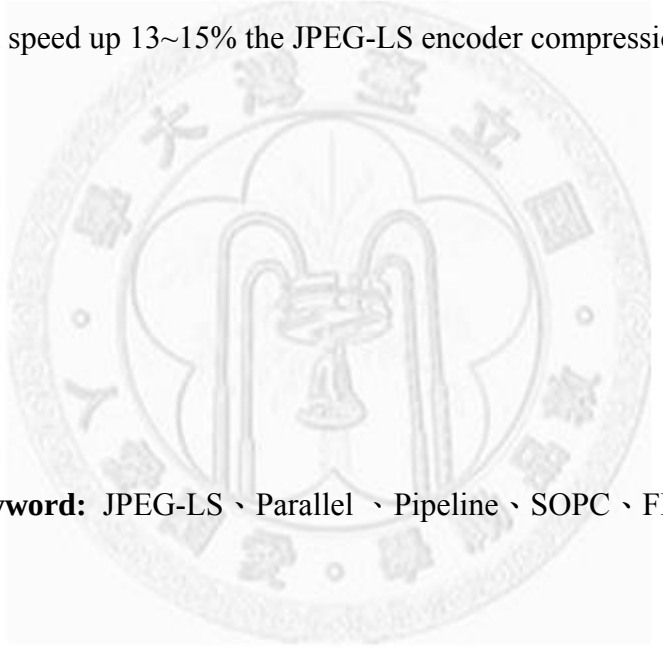
JPEG-LS 是國際標準的壓縮技術之一，在本論文中我們使用 Verilog HDL 來實作 JPEG-LS 編碼器，最後在 FPGA (Field Programmable Gate Array)上面做驗證，在此我們提出在不需要增加額外的硬體電路時，利用平行化、管線化、模組化的特點，改良其電路架構後，就可以加速 JPEG-LS 的編碼電路約 13~15%的時間。

關鍵字：JPEG-LS、平行化、管線化、SOPC、FPGA

Abstract

JPEG-LS is an international standard for lossless and near-lossless image compression. In this paper, a hardware implementation using Verilog HDL is proposed for a JPEG-LS encoder. The hardware design for the JPEG-LS encoder is tested in a Field Programmable Gate Array (FPGA).

JPEG-LS Encoders have two main data compression modules, Regular mode and Run mode. We separate some modules such as Line Buffer, Regular mode, Run mode, Golomb Encoder, and update parameters from each other. We also share the module parameters to reduce the gate number. The architecture developed is a pipelined parallel design, which can speed up 13~15% the JPEG-LS encoder compression performance.



Keyword: JPEG-LS、Parallel 、Pipeline、SOPC、FPGA

目錄

圖目錄

表目錄

第一章 緒論

1.1 研究動機

近年來政府大力推動醫療電子產業，以目前來看，其比例最高為醫療儀器及電子醫療設備，如超音波機、X光機等設備。醫學影像通常區分：超音波影像、內視鏡影像、X光影像，其中以X光片是最為複雜的影像，也需以超高解析度的數位影像來儲存，也因此原始的醫學影像檔案非常大不利於儲存及管理，此時適當的壓縮就的格外的重要。為了避免醫療糾紛，壓縮此類的醫學影像最好不能有失真。

JPEG-LS[1]是ISO/ITU的壓縮標準，由HP實驗室所發展出來的，有無失真(lossless)和趨近無失真(near-lossless)的演算技術[2,3]，主要特色是低成本、低複雜度的壓縮技術[4]，藉其靜態影像壓縮技術可以方便我們儲存管理大量的圖像資訊。

在此本論文先將圖 5-6 Source_Image1 和圖 5-7 Source_Image1 先用軟體運算，得到 Source_Image1 在 JPEG-LS 編碼運算時執行了 Regular mode 有 225687 次，Run mode 有 525 次，Source_Image2 在則執行 Regular mode 有 227826 次，Run mode 有 1260 次(如表 1-1 所示)。

	Regular mode 執行次數	Run mode 執行次數
Source_Image1.bmp	225687	525
Source_Image2.bmp	227826	1260

表 1-1: 軟體執行 JPEG-LS encoder 在 Regular mode 和 Run mode 的次數

我們清楚可以從圖 1-1 和圖 1-2 Regular mode 與 Run mode 的執行比例中看到，在 2 張圖片幾乎 99%都是執行在 Regular mode，若是可以對 Regular mode 做硬體加速時，即可提升整體 JPEG-LS encoder 的速度。

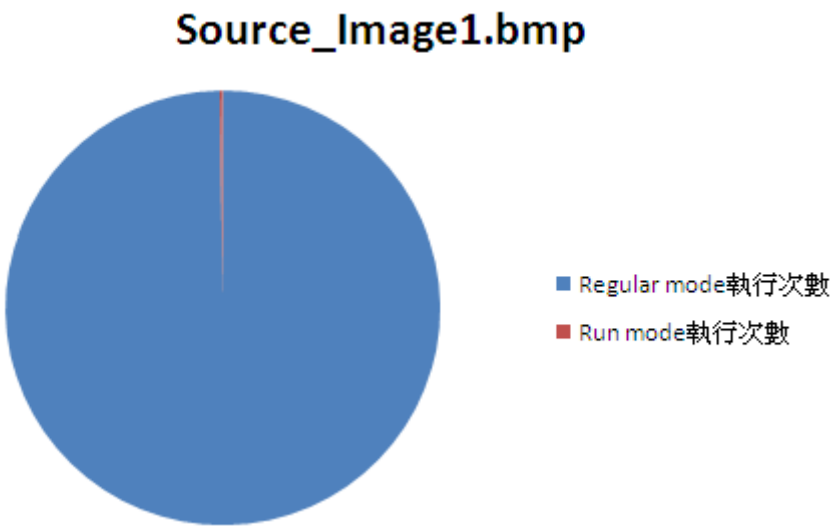


圖 1-1: Regular mode 與 Run mode 的執行比例

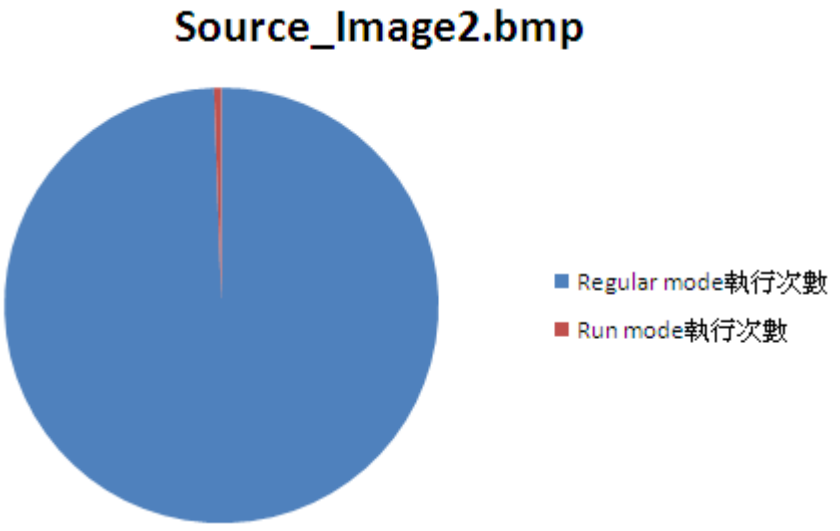


圖 1-2: Regular mode 與 Run mode 的執行比例

JPEG-LS也跟隨著嵌入式系統已經成為主流消費性電子產品的發展重心，為創造出更高效率的ASIC(application-specific integrated circuit) [4,5,6]也被一直提出來討論用於不同領域上。在演算法的部分由於需要做預測值與預測值修正反覆的執行再進行編碼[7]，它所需的時間是比較長的，也因此把硬體架構平行化、管線化[8]及模組化的架構也是本論文努力的目標。

1.2 研究目標與貢獻

在上一章節的結果中我們得到 Regular mode 所花的時間最多，在"A Parallel Pipelined Implementation of LOCO-I for JPEG-LS[8]"論文中。作者一次處理 2 個 Pixel 分別為 X1、X2 如表 3-2 所示，它去取 X1 及 X2 兩個相鄰點的值(C、B、A、D、E)等 5 個 sample 點，原始演算法則只有取(C、B、A、D)四個 sample 點，因此在圖示 3-4 中可以看到三個硬體平行處理，分別為 Regular mode encoder 2 (Encode sample X2)、 Regular mode encoder 1 (Encode sample X1)、Run-Length mode encoder，在這三部份的硬體在處理完資料後一定要經過 Bit stream reorder module 重新組合三部份硬體在不同時間所輸出的資料，最後再將資料透過 Coded bit stream 編碼再輸出到 Output stream buffer，因為在硬體的部份是由(Encode sample X1)和(Encode sample X2)在處理，所以在執行速度上平均提升了約 2 倍。

G1 = D-B
G2 = B-C
G3 = C-A
G4 = E-D
C1 = f(Q1,Q2,Q3)
C2 = f(Q4,Q1,Q3)

Rc	Rb	Rd	Re
Ra	X1	X2	

表 1-2: The extended neighborhood for the parallel computations of pixels X1 and X2[8]

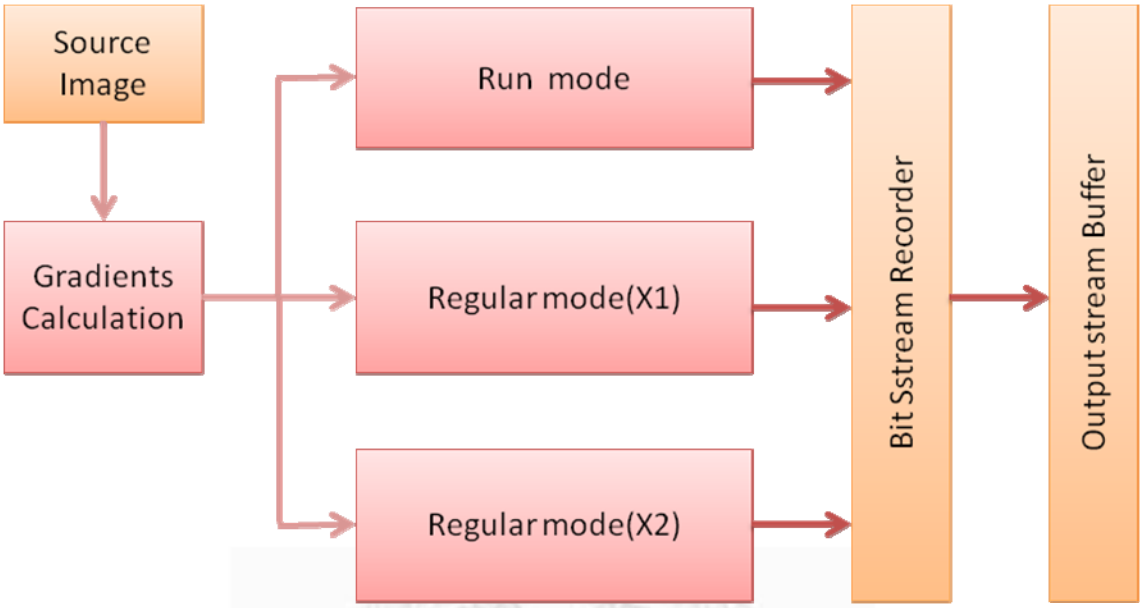


圖 1-3: N2C-EX encoder 電路架構[8]

在 JPEG-LS 的演算法內有許多相同的演算法及參數會被重覆使用，模組化的設計[9]，可增加矽智財(IP)的使用率，也因此本論文將著重在平行化及模組化的探討，將其 JPEG-LS 編碼電路實作出來，並且在 FPGA(Field Programmable Gate Array)[10,11]上面驗證。

本篇論文藉由Altera[14] QuartusII 及DE2-70，來進行整合邏輯電路和設計驗證的功能，分析JPEG-LS架構後，以不增加電路的前提之下，來改善JPEG-LS演算法的系統架構，經平行化與模組化之後，可增加整體JPEG-LS硬體的速度，藉此來開發出高效率的JPEG-LS的硬體壓縮器，增進體積小、速度快的目標。

1.3 章節概述

本篇論文共分為六個章節。第一章為緒論。第二章為相關技術與研究，主要詳細介紹JPEG-LS壓縮的演算法和相關論文的探討。第三章為JPEG-LS硬體壓縮加速器之電路架構，內容詳述本篇論文實驗的系統架構、及圖像檔案格式介紹，改善後的硬體分析。第四章為設計流程，主要詳述整個系統的建構過程。第五章為JPEG-LS壓縮的實驗結果。最後第六章為結論。

第二章　JPEG-LS 編碼演算法

LOCO-I (LOw COmplexity LOssless COmpression for Images)[1] / JPEG-LS 是 ISO/ITU 壓縮標準，JPEG-LS 支援無失真(Lossless)和趨近無失真(Near-lossless)兩種影像壓縮格式，並且根據圖片影像的特性，分為 Run Mode 和 Regular Mode 兩種方式來對影像進行壓縮。其演算法是對於相鄰並且連續相同的影像資料以 Run Mode 的方法將其資料以 32bit 的資料量來編碼壓縮後輸出，影像資料相鄰不相同的會以 Regular Mode 的方式，先將影像資料的預測值與實際值的誤差使用 Golomb Code[7]的編碼方式來輸出。編碼時，就是使用影像資料的預測值與實際值去做 Golomb Code[7]編碼，使其 JPEG-LS 增加壓縮率。

2.1 編碼原理

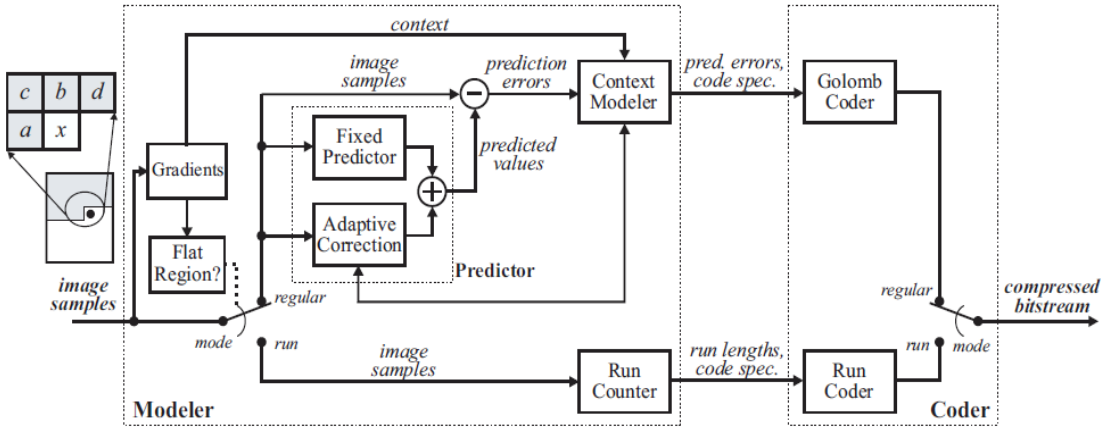


圖 2-1: JPEG-LS Block Diagram

圖2-1是完整的編碼流程，它先將圖檔以一列的方式讀進Line Buffer，然後一次讀取一個byte也就是x,透過Gradients模組來判斷應進入Run mod或者Regular mode模組。

每當進入Run mode時，會計算相鄰並且連續相同的sample個數，直到sample x不相同或者已達到列的最後sample，之後進入Run Coder module來進行編輯。進入Regular Mode會利用x 相鄰的(a,b,c,d)來將預測值(predicted value)，與實際值做資料的比對，在利用Prediction error module來修正，最後再進入Colomb Coder module的編碼方式來輸出。

2.2 壓縮步驟

壓縮的步驟共有 18 個詳細步驟如下：

1. 初始化:

計算 RANGE(參數定義如表 2-1 所示):

lossless => RANGE = MAXVAL + 1

near-lossless => RANGE = ⌊(MAXVAL+2*NEAR)/ (2*NEAR+1) ⌋

計算參數 qbpp = ⌈ log RANGE ⌉

bpp = max(2, ⌈ *MAXVAL* +1 ⌉)

LIMIT = 2*(bpp + max(8, bpp))。

B[Q] = 0; // 365 counters for computing the bias

C[Q] = 0; // 365 counters storing prediction correction values

N[Q] = 1; // 367 counters for negative prediction error for run interruption

A[Q] = 4; // 367 counters for the accumulated prediction error magnitude

RUNindex=0

J[0..31] = {0, 0, 0, 0, 1, 1,1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9,10,11,12,13,14,15}.

Initializes the two run interruption variables *Nn*[365], *Nn*[366] = 0

RANGE: range of prediction error representation
MAXVAL: maximum possible image sample value over all components of a scan
NEAR: difference bound for near-lossless coding
bpp: number of bits needed to represent MAXVAL, with a minimum of 2
qbpp : number of bits needed to represent a mapped error value.
LIMIT: the value of *glimit* for a sample encoded in regular mode.
Q: context determined from Q1, Q2, Q3
RUNindex: index for run mode order
J[0..31] 32 variables indicating order of run-length codes

表 2-1 參數定義

2.計算 Local gradients

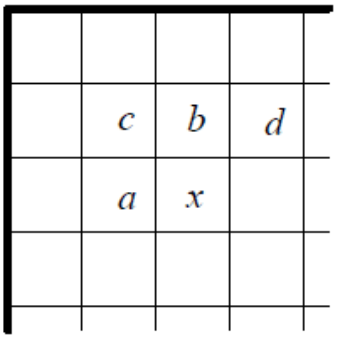


圖 2-2: Causal template used for context modeling and prediction

Index	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	90	74	74
2	0	68	50	43	205	205
3	68	64	145	145	145	145
4	64	100	145	145	145	

圖 2-3: Example image data

D1 = Rd –Rb

D2 = Rb –Rc

D3 = Rc –Ra

3.依據上面式子運算出 D1、D2、D3 結果,如果為 0 進入 Run Mode Processing 則跳到第 17 步驟，否則進入 Regular Mode 往下一步執行。

```
if ( (D1 && D2 == 0) && (D3 == 0))
   go to RunModeProcessing;
else
   go to RegularModeProcessing;
```

4. 將第二步驟求出的 D1、D2、D3 量化(quantize)，得到 vector(Q1,Q2,Q3)來表示 sample x 的 context.

```
if(Di  <=  -T3)
      Qi  =  -4;
else if (Di  <=  -T2)
      Qi  =  -3;
else if(Di  <=  - T1)
      Qi  = -2;
else if(Di  <=  - NEAR)
      Qi  = -1;
else if(Di  <=  NEAR)
      Qi  =  0;
else if(Di  <=  T1)
      Qi  = 1;
else if(Di  <=  T2)
      Qi = 2;
else if(Di  <=  T3)
       Qi = 3;
else
      Qi=4;
```

Qi: one of the three quantized region numbers Q1, Q2, Q3

5. vector (Q1,Q2,Q3)的第一個非零值為負數時，vector 所有元素反相(-Q1,-Q2,-Q3)，SIGN = -1。反之若 vector (Q1,Q2,Q3)的第一個非零值為正數時，則將 SIGN = 1。利用此 vector 來計算出一個整數 Q 值，它將來當存取參數的 Index。

6.使用 Ra，Rb 及 Rc 來計算現在 pixel *x* 的預測值 Px。

```
if (Rc  >=  max(Ra,Rb) )
   Px = min(Ra,Rb);
else{
   if  (Rc <= min(Ra,Rb) )
      Px = max(Ra,Rb);
   else
      Px= Ra+ Rb- Rc;
}
```

7.預測值修正 (Prediction Correction)：

使用步驟 6 所計算出來的 Px 值加上 C[Q]及變數 SIGN 來修正 Px 的範圍，它的範圍介於 [0..MAXVAL] 之間。

```
if(SIGN = +1)
   Px = Px + C[Q];
else
   Px = Px - C[Q];

if(Px > MAXVAL)
   Px=MAXVAL;
else if(Px < 0)
   Px=0;
```

8. 計算預測誤差值 (Computation of prediction error)使用預測值 Px，與目前的 sample x 相減，若 SIGN 為負數，則將 Errval 值反相。

```
Errval = Ix - Px;
if(SIGN = -1)
   Errval = -Errval;
```

9.利用 sample x 、Px、 Errval 來重新建立 Rx 值，演算法如下

```
if(Errval > 0)
   Errval = (Errval + NEAR)/(2*NEAR+1);
else
   Errval = -(NEAR-Errval)/(2*NEAR+1);

   Px = Px + SIGN*Errval*(2*NEAR+1);

if  (Px < 0)
   Rx = 0;
else  if (Rx > MAXVAL)
   Rx = MAXVAL;
```

Errval: prediction error (quantized or unquantized, before and after modulo reduction)

10.縮減預測值誤差值 (prediction error)

它介於 (- ⌊ RANGE / 2 ⌋ ... ⌈ RANGE / 2 ⌉ − 1)

```
if(Errval < 0)
   Errval = Errval + RANGE;
if(Errval >= (RANGE + 1) / 2)
   Errval = Errval + RANGE;
```

11. Golomb coding 變數計算

利用變數 N(Q)、A[Q]來計算 K 供 Golomb encoding 時使用

```
for  (k=0;( (N[Q] << k)  <  A[Q]  ) ;  k++ )
```

12. Error mapping

利用 B[Q]、N[Q]，K 值來將誤差值(Prediction error)轉成正值

```
if( (NEAR == 0) && (k ==0) &&  (2*B[Q] <= -N[Q])){
   if(Errval >= 0)
      MErrval = 2 * Errval + 1;
   else
      MErrval = -2 * Errval + 1;
}else{
   if (Errval >= 0)
      MErrval = 2 * Errval;
   else
      MErrval = -2 * Errval;
}
```

MErrval: Errval mapped to non-negative integers in regular mode

13.得到的 MErrval 配合步驟 11 所得之 k 做 Golomb encoder

14. 更新變數

把目前的 prediction error 值更新到 A[Q]、B[Q]、N[Q]等變數。

```
B[Q] = B[Q] + Errval*(2*NEAR+1)
A[Q] = A[Q] + abs(Errval);
if(N[Q]==RESET)
{
    A[Q]=A[Q]>>1;
    B[Q]=B[Q]>>1;
    N[Q]=N]Q[>>1;
}
N[Q]=N[Q]+1;
```

15.最後是更新變數 C[Q]

```
if(B[Q] <= -N[Q])
{
   B[Q] = B[Q]+ N[Q];
   if(C[Q] > MIN_C)
       C[Q] = C[Q] - 1;
   if(B[Q] <= -N[Q])
       B[Q] = -N[Q] + 1;
}else if (B[Q] > 0){
   B[Q] = B[Q] - N[Q];
   if(C[Q] < MAX_C)
      C[Q] = C[Q] + 1;
   if(B[Q] > 0)
      B[Q] = 0;
}
```

16.重新回到步驟 2 去處理下一個 sample.

17. Run mode encoder

a. 設定 RUNval = Ra. 當(abs(Ix - RUNval) <= NEAR) RUNcnt++ ,

如果不是讀到每條線的最後一個 sample x, 再讀下一個 sample x,

b. 如果 RUNcnt >= 2 ^ J[RUNIndex] 時做下列三步驟, RUNIndex 最大值為 31

i.輸出一個為 1 的 bit stream

ii. RUNcnt = RUNcnt - 2 ^ J[RUNIndex]

iii.假設 RUNindex < 31 , RUNindex++;

c.若讀到列的最後一個 sample x 就停止

i.如果 RUNcnt > 0,輸出一個 bit 1 到 bit stream

ii.跳到步驟 16

附加 0 到 bit stream

RUNcnt: repetitive sample count for run mode

RUNindex: index for run mode order

RUNval: repetitive reconstructed sample value in a run

在編碼的過程中，影像資料的預測值加入 Golomb Code[7]的編碼是使 JPEG-LS 達到良好的壓縮效率原因之一。

第三章 JPEG-LS 硬體壓縮電路之架構分析

在本章節中，會分析 JPEG-LS 硬體編碼電路架構、介紹 PPM(Portable Pixel Map)檔案格式、系統流程圖，各個模組架構、資料流程等。

3.1 PPM(Portable Pixel Map)檔案格式介紹

以彩色 PPM(Portable Pixel Map)檔來說一張圖檔是由 3 個 component 所組成，分別為紅綠藍(RGB)組成，每個 component 的組成為 8 bits，其範圍為 0 ～ 255,若只有 1 個 component 則代表為灰階圖，圖 3-1 為多個 components 的組成，圖 3-2 為每個 sample x 會每條 Line 的讀取從 Top 讀取到 bottom 與 Image 的相對應關係。

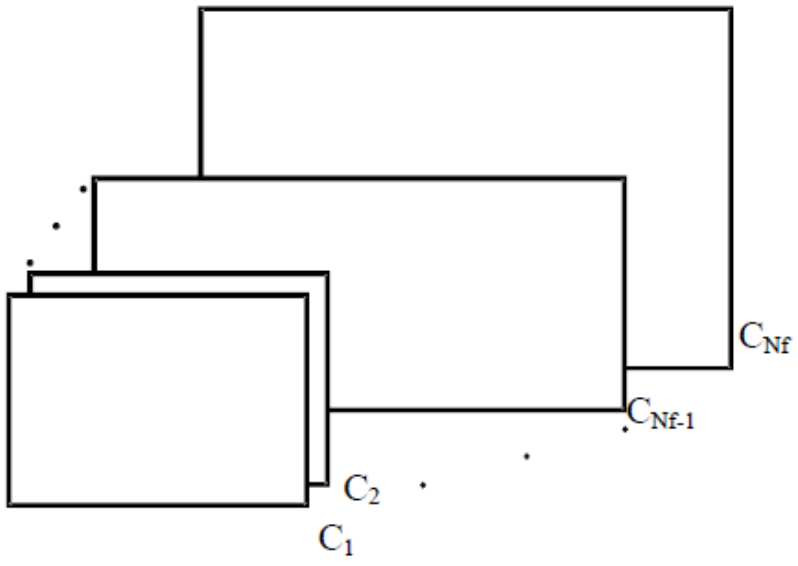


圖 3-1: Source image with multiple components

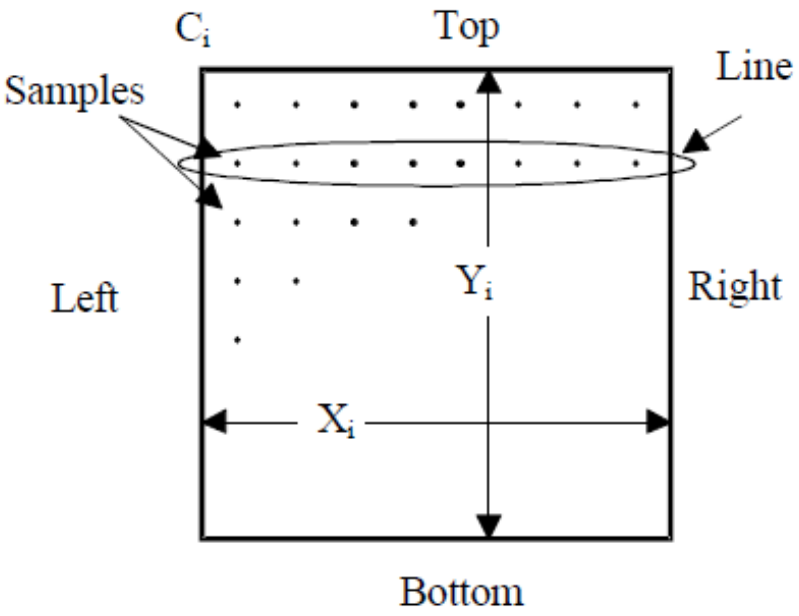


圖 3-2: Characteristics of an image component

ppm 檔案 header 的格式如下圖 3-3 所示。

```
P6
240 320
255
```

圖 3-3: PPM header file

如圖 3-3 中，P6:是表示此張是彩色圖檔,P5:則表示灰階圖檔[13]。

240:寬度為 240 pixels

320:高度為 320 pixels

255:表示最大值是 255

24bit/per pixel	Red	Green	Blue
Red(100% intensity)	11111111	00000000	00000000
Green (100% intensity)	00000000	11111111	00000000
Blue (100% intensity)	00000000	00000000	11111111

表 3-1 RGB 各顏色所代表的值

Total Size: 240 * 3(24bit) * 320 +15(header) = 230415 Bytes (Image data)

3.2 初始化硬體電路

JPEG-LS 編碼電路必須做資料的初始化動作，首先先對硬體做 reset，並且 initial hardware，這部份是把參數預設值寫到 RAM 裡面，待編碼電路開始執行時再去讀取它，每一次會以 1 條 Line 為處理單位，直到讀取最後一條 Line 才結束，圖 3-4 將說明整個硬體讀取的流程。在下一節中我們將開始討論資料輸入的第一個硬體電路 Line Buffer.

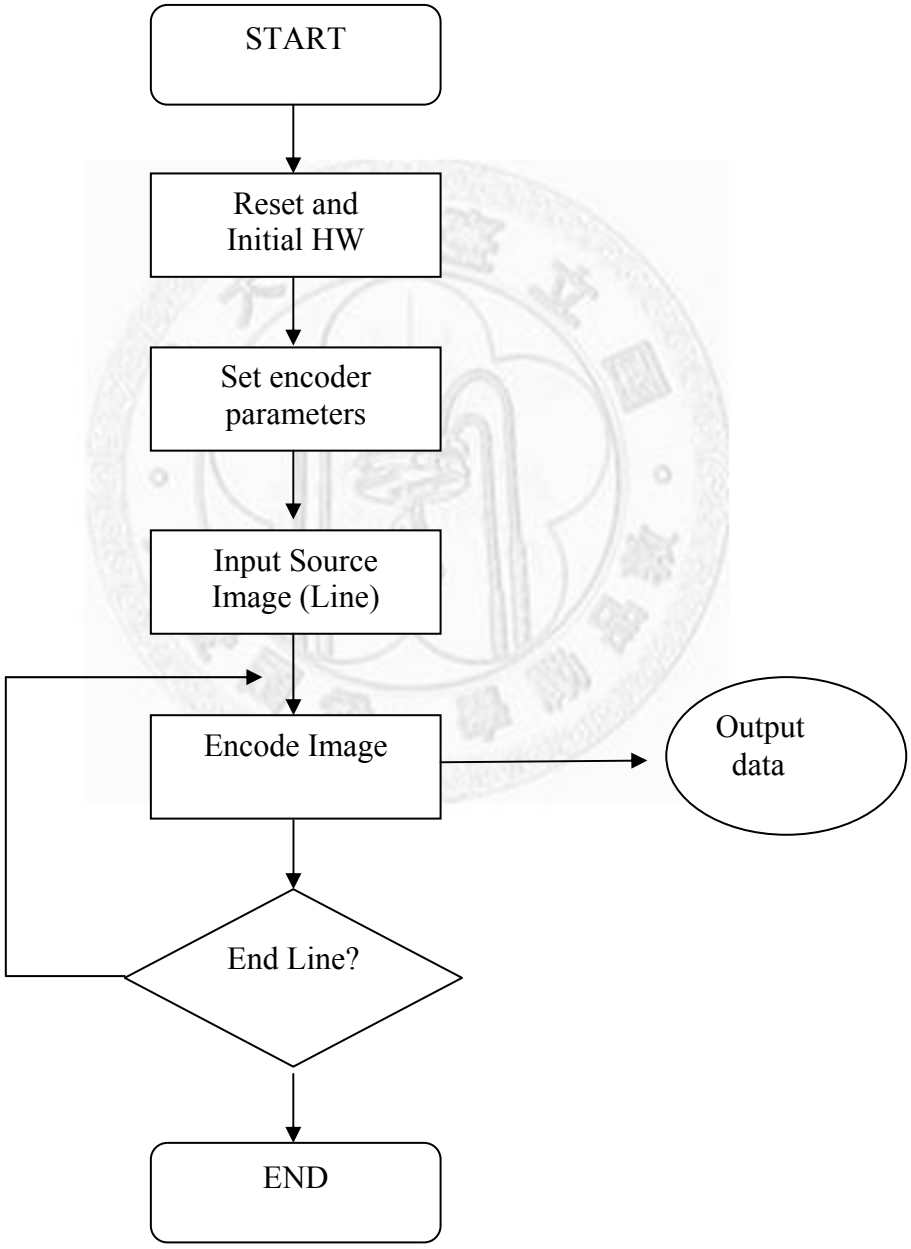


圖 3-4: JPEG-LS 初始化和硬體編碼流程

3.3 輸入行緩衝器(Line Buffer)

在這個小節中我使用輸入行緩衝器(Line Buffer)儲存 2 列的圖像資料，第一行的影像 sample 由 Line1 的尾端輸入如圖 3-5 所示，一次可以讀取 C、B、D、A、X 等等的 sample 點在提供給 Parallel Subtractor 做圖示 3-8 的運算。

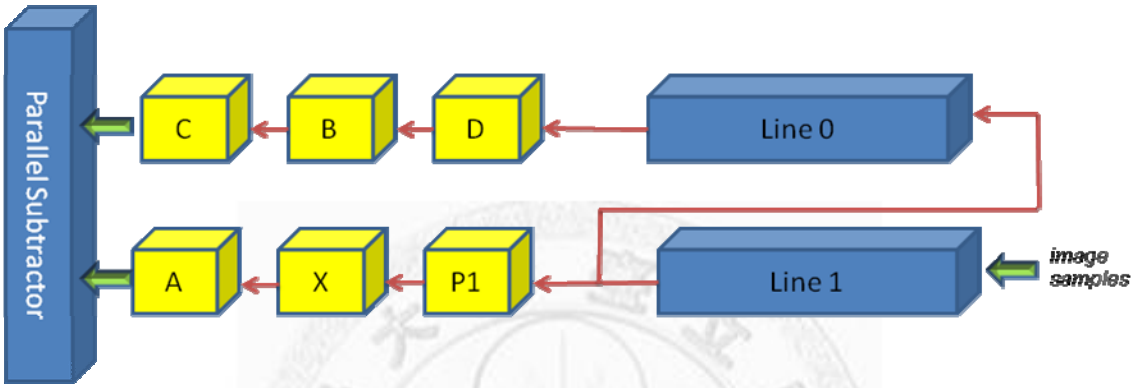
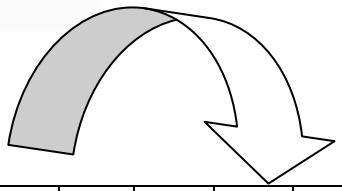


圖 3-5: Source image with multiple components

在每條線的邊緣的處理上，採用圖 3-6 的方式，在每條線處理開始時，將目前的取樣點複製到最前 3 個點[4]也就是如同圖 3-7，bb0、bb1、bb2 複製到 aa1、aa2、aa3 的地方。而線尾邊緣則是 bb0、bb1、bb2 複製到 aa1、aa2、aa3 的位置(圖 3-7)，把這部份的資料處理是交由 Linux Buffer module 來運算的。



0	0	0	XX01	XX02	XX03	…	…	…	X0n-2	X0n-1	X0n	X0n-2	X0n-1	X0n
XX01	XX02	XX03	XX11	XX12	XX13	…	…	…	X1n-2	X1n-1	X1n			
XX21	XX22	XX23	XX21	XX22	XX23				X2n-2	X2n-1	X2n			

圖 3-6 extend the edges

cc0	**cc1**	**cc2**	**bb0**	**bb1**	**bb2**	**dd1**	**dd2**	**dd3**			
aa1	**aa2**	**aa3**	**xx1**	**xx2**	**xx3**						

圖 3-7 extend the edges 與實際 Pixel 對應

3.4 預測及修正值電路架構

圖示 3-8 說明了預測值修正(Prediction Correction)從 Memory 取出 C[Q]並與 SIGN 來修正前一級所計算的 Px 值，並限定其範圍在[0..MAXVAL]內，經預測值修正在重新 mapping B[Q]、A[Q]、N[Q]這些參數後儲存回 Memory 內。可參考第 2 章 JPEG Encode 的 7~14 點。

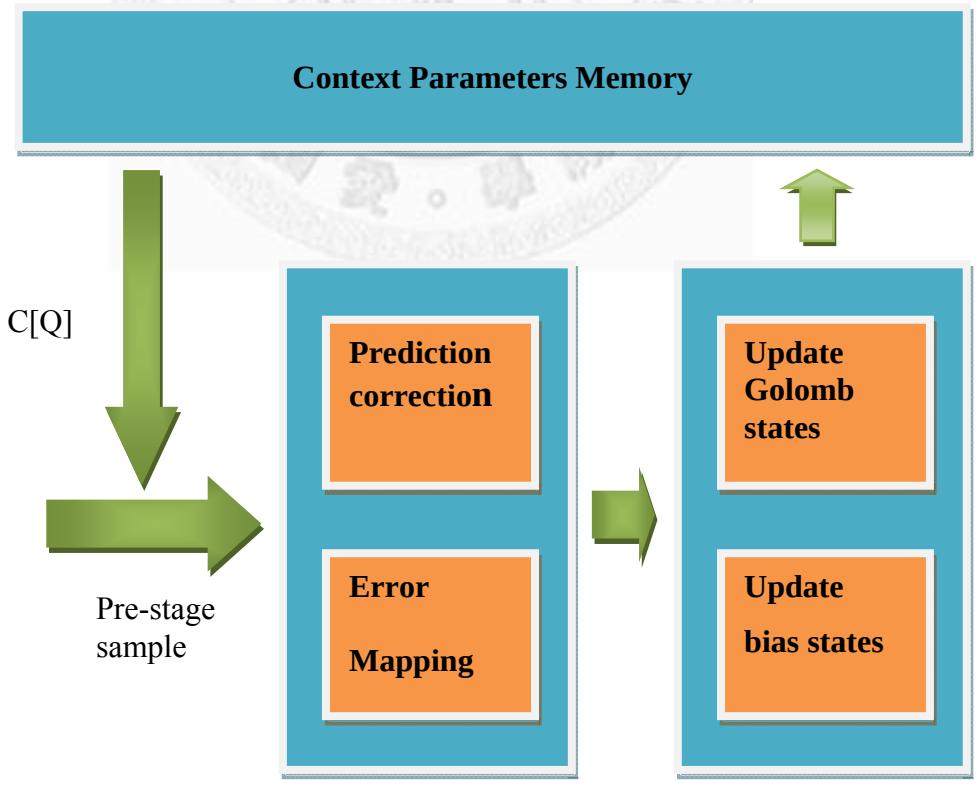


圖 3-8 預測值修正與更新變數電路

3.5 原始硬體架構

原本的影像資料(圖 3-9)每條放入 Line Buffer 後，在去讀取 sample x 後交給 Gradients 去計算去 D1、D2、D3 的值,假設都為 0 進入 Run mode，否則進入 Regular mode.在經過實際的 x 值與預測值比對後做 Prediction error 修正、重新 mapping 更新變數後，進入 Golomb coder 加以編碼，最後再以 bit stream 輸出，此時 Line Buffer 再去取下一個 sample x。

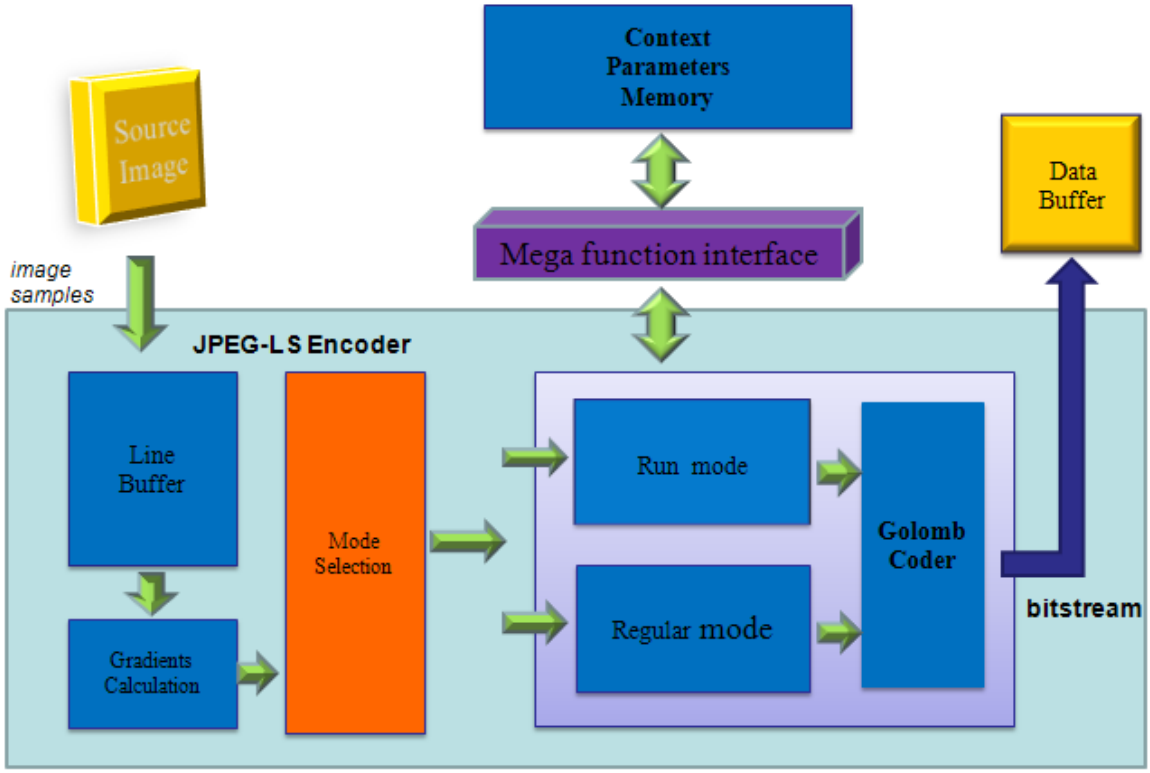


圖 3-9 原始 JPEG-LS 加速器硬體架構方塊圖

3.6 改良的硬體架構

圖 3-10 是 JPEG-LS 加速器基本硬體架構，從 Line Buffer 讀取 sample 點後交給 Gradients 去計算去 D1、D2、D3 的值,假設都為 0 進入 Run mode，否則進入 Regular mode，在經過實際的 x 值與預測值比對後做 Prediction error 修正、重新 mapping 更新變數(update variables)到暫存器和記憶體後，這時候只要 Run mode 或者 Regular mode 輸出一個觸發信號給 Golomb encoder 後，就可以繼續取下一個 sample 點，在這的同時 Golomb coder 平行做編碼，以 bit stream 輸出。完整的硬體電路如圖 3-11 所示。

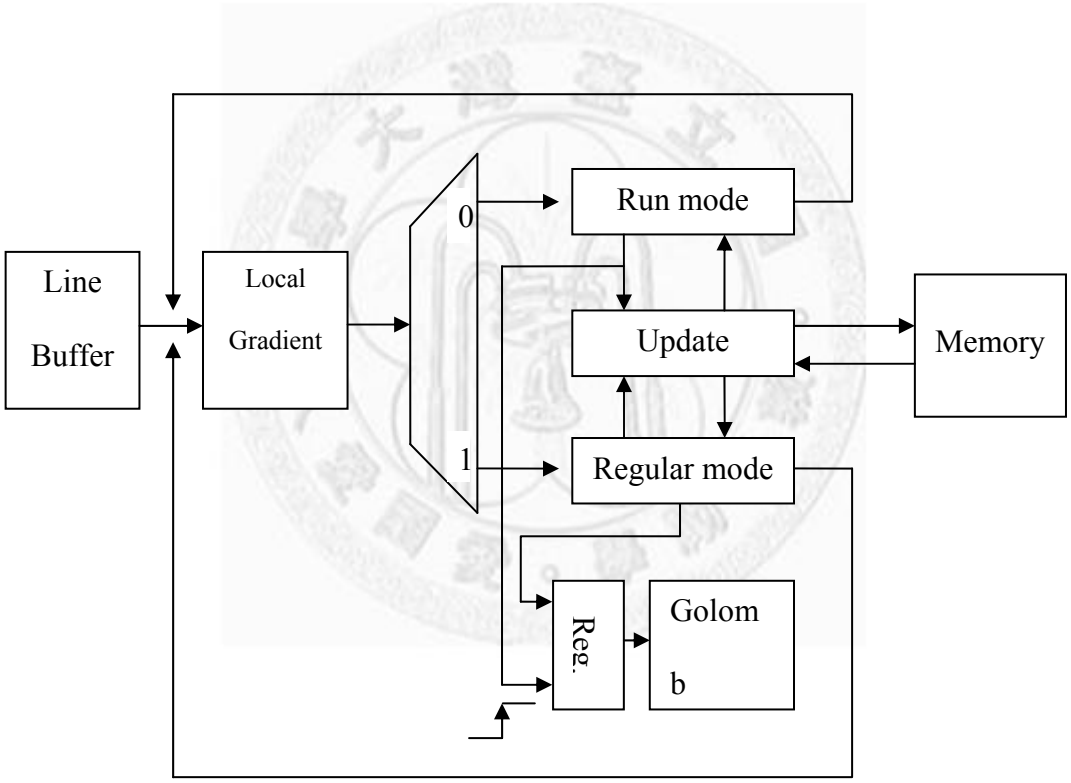


圖 3-10 JPEG-LS 加速器硬體架構

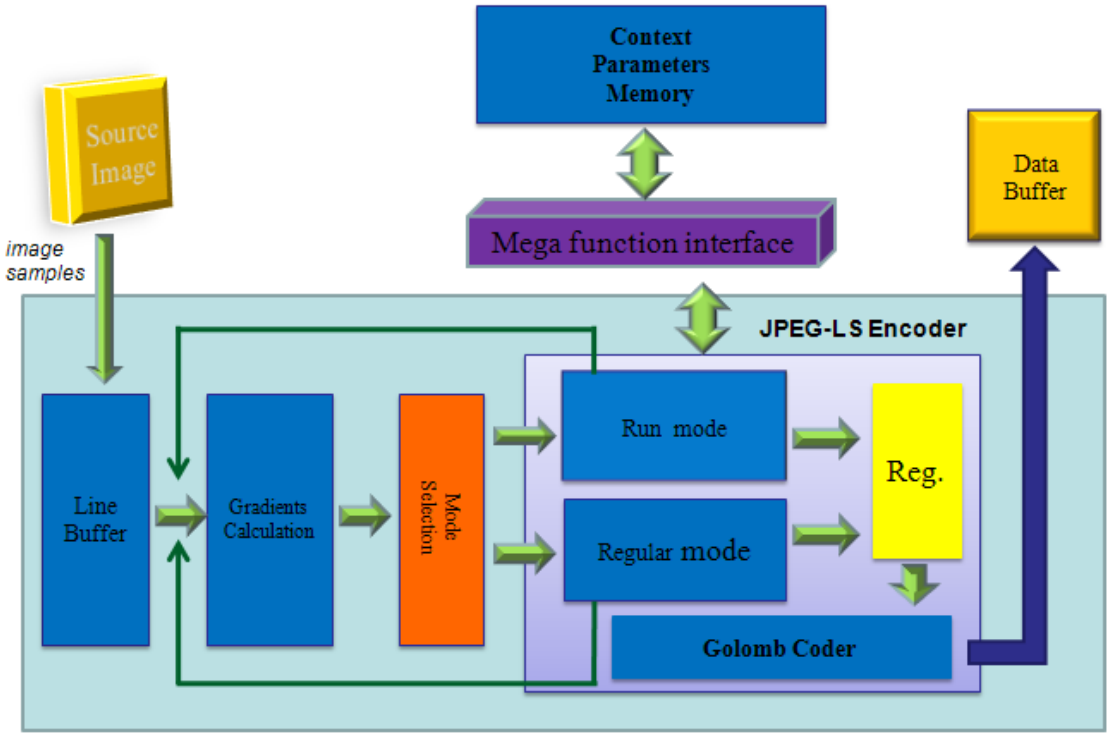


圖 3-11 JPEG-LS 加速器硬體改良架構方塊圖

第四章 系統實作

在本章中，將介紹系統實作過程以 ModelSim 來做各模組與系統的模擬 function 之正確性，並以 Altera QuartusII 搭配 DE2-70 為實作平台加以驗證結果。

4.1 電路模擬

利用為QuartusII 所提供的功能MegaFunction來做為Parameter的Buffer。它是使用內建的Mega Wizard Plug-In Manager來呼叫IP。MegaFunction內的IP有正反器、I/O、算術運算單元、記憶體等等。以下將介紹如何利用MegaFunction來製作Parameter的Buffer:

(1). Tool\Mega Wizard Plug-In Manager…

(2). Create a new custom megafunction variation, next

(3). 選擇Memory Compiler及RAM : 2-PORT。

(4). 8-bit word of Memory選擇1024,data bus 選32,如圖示4-1, next

(5). Create a 'rden'read enable signal 打勾

(6). 直接下一步到圖4-2.

(7). 最後選擇 Finish，這樣就製做出我們要的buffer大小。

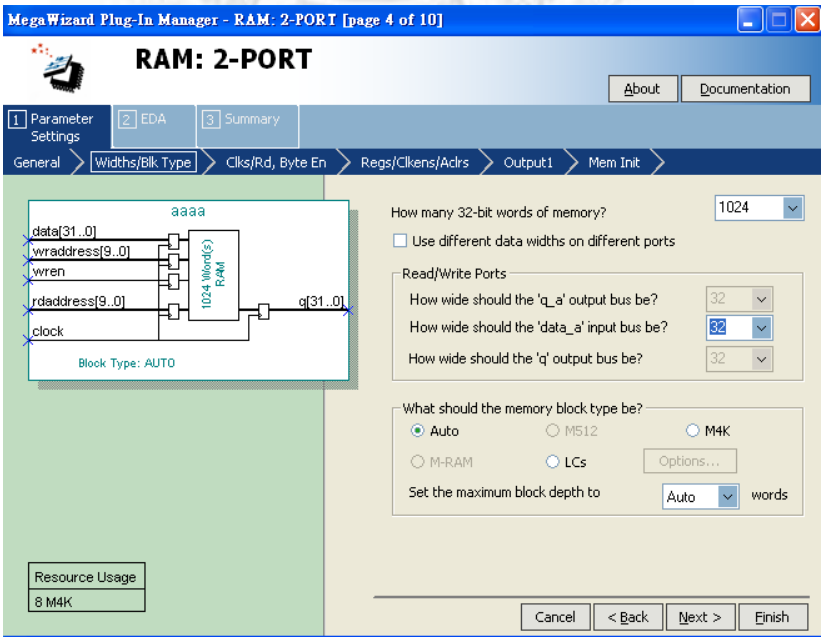


圖 4-1: 調整 Mega RAM size

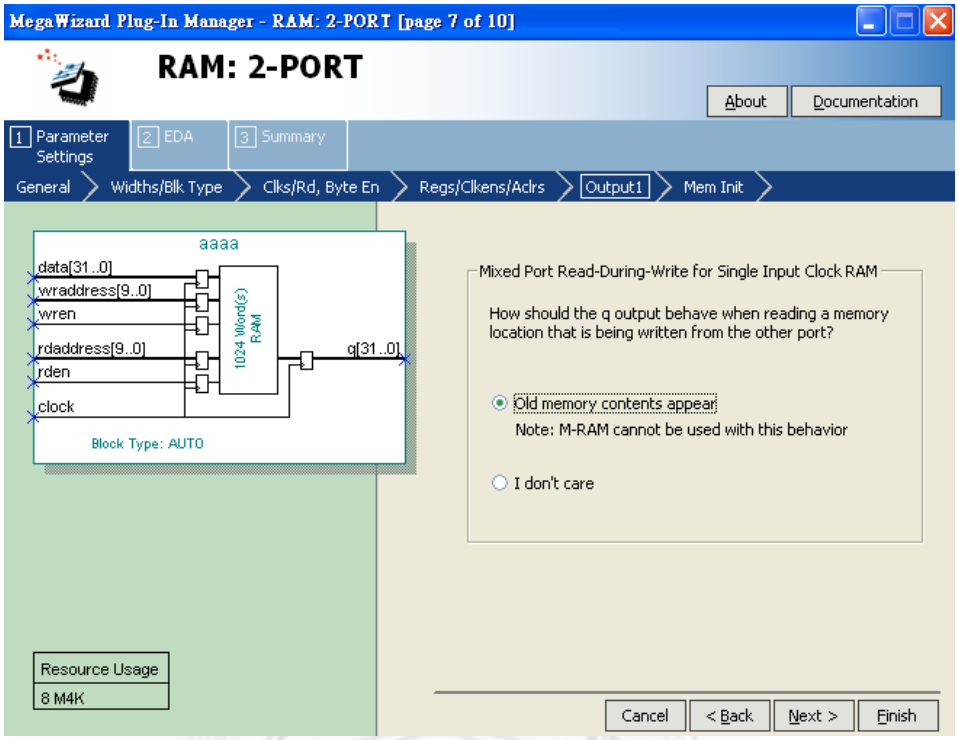


圖 4-2 完成使用的參數儲存的 RAM

4.2 使用 ModelSim Library

在本論文中使用ModleSim來模擬function是否正確。由於資料參數的Buffer使用到Altera的MegaFunction 來產生上一節所提出的 Dual Port RAM，因而在模擬時需要加入QuartusII的模擬Library 。把C:\altera\81\quartus\eda\sim_lib複製到專案當中。結合我們所產生的RAM2Port.v來進行編譯(如圖4-3)，就可以利用ModelSim來驗證所有模組的功能是否正確。

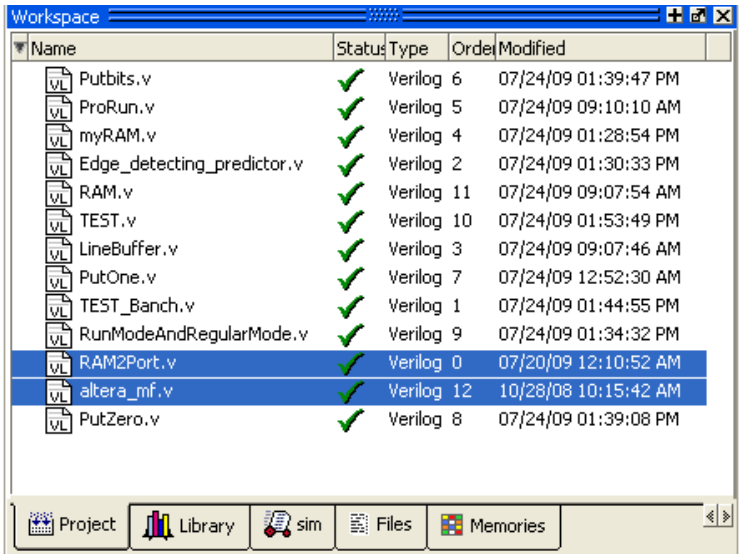


圖 4-3 加入 altera_mf.v 和 RAM2Port.v 來進行模擬

4.3 Altera Quartus II 介紹

Quartus II是Altera[14]針對他們的FPAG(Field Programmable Gate Array) 所開發的電腦輔助設計工具(EDA tool)， Quartus II可將所撰寫的硬體描述語言(Hardware Description Language,HDL)，例如AHDL(Altera HDL)、Verilog HDL、VHDL(Very High Speed Intergrated Circuit HDL)、RTL(Register Transistor Level)、Schematic 等，經由合成(Synthesis)轉換成gate level netlist的電路檔，最後再進行佈置繞線(Place & Route tools)，再將Altera所產生的燒錄檔(*.sof)燒錄到晶片[15,16]。

合成完可先利用QuartusII 提供的NetList Viewers中的RTL Viewer檢查合成後的電路是否符合預期，亦可用內建的邏輯分析儀(SignalTap)，來觀查波形。

4.4 RTL 設計流程

圖示 4-3 為 RTL Design flow，由演算法分析開始，接著 RTL 的設計完後做 simulation，正確後做 RTL 合成(Synthesis)，再做 gate level simulation，直到模擬都正確為止。

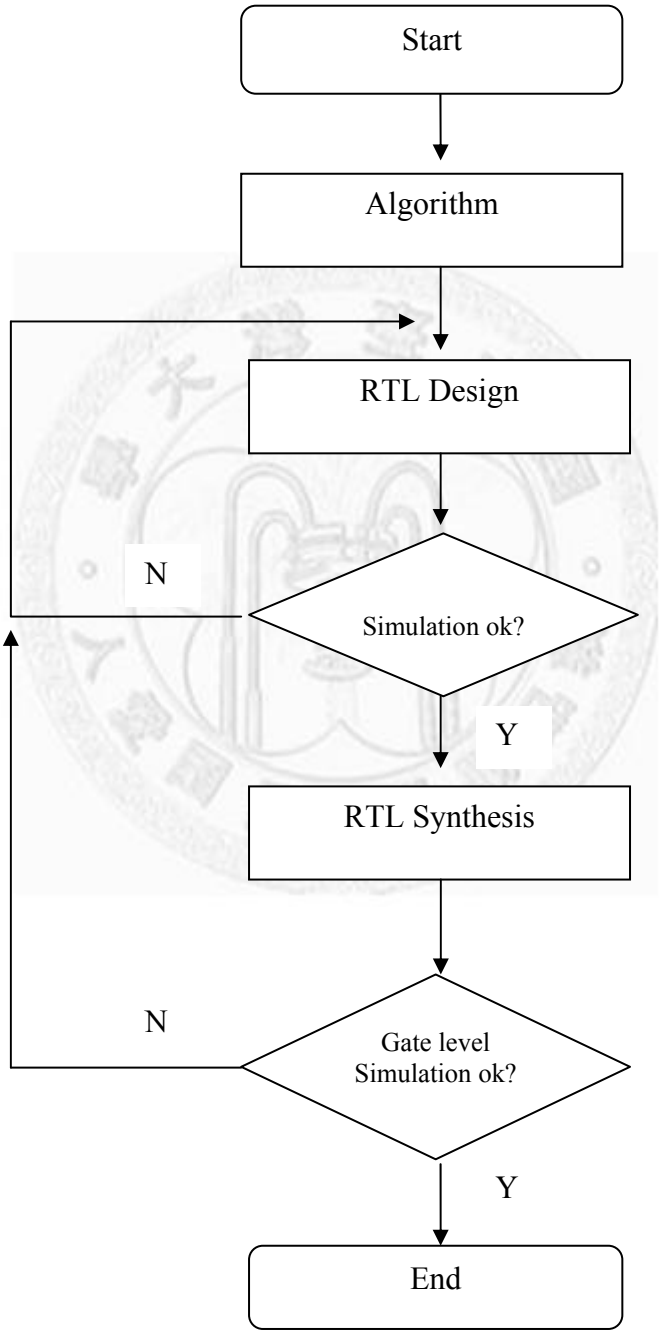


圖 4-4: Design Flow

4.5 RTL 編碼技巧

本論文在RTL設計初期常會有latch 電路的產生，以及不預期的情況出現，為了突顯coding style的重要性，在此章節我加了幾個數據來表達它真的很重要。假設預期我要在第3個CLK輸出資料0x03，如圖4-5，RTL的coding我在圖4-6及4-7列了2種不同的寫法，一樣可以達到輸出資料0x03的目地，但是test1.v和test2.v兩者所成合出來的Total logic elements卻明顯不同，test1.v合成出來是21 Total logic elements，test2.v合成出來是5 Total logic elements，test1.v比test2.v多了3倍的Total logic elements，由此可見好的編碼技巧不但可以保持電路的穩定性，還可以節省硬體空間。

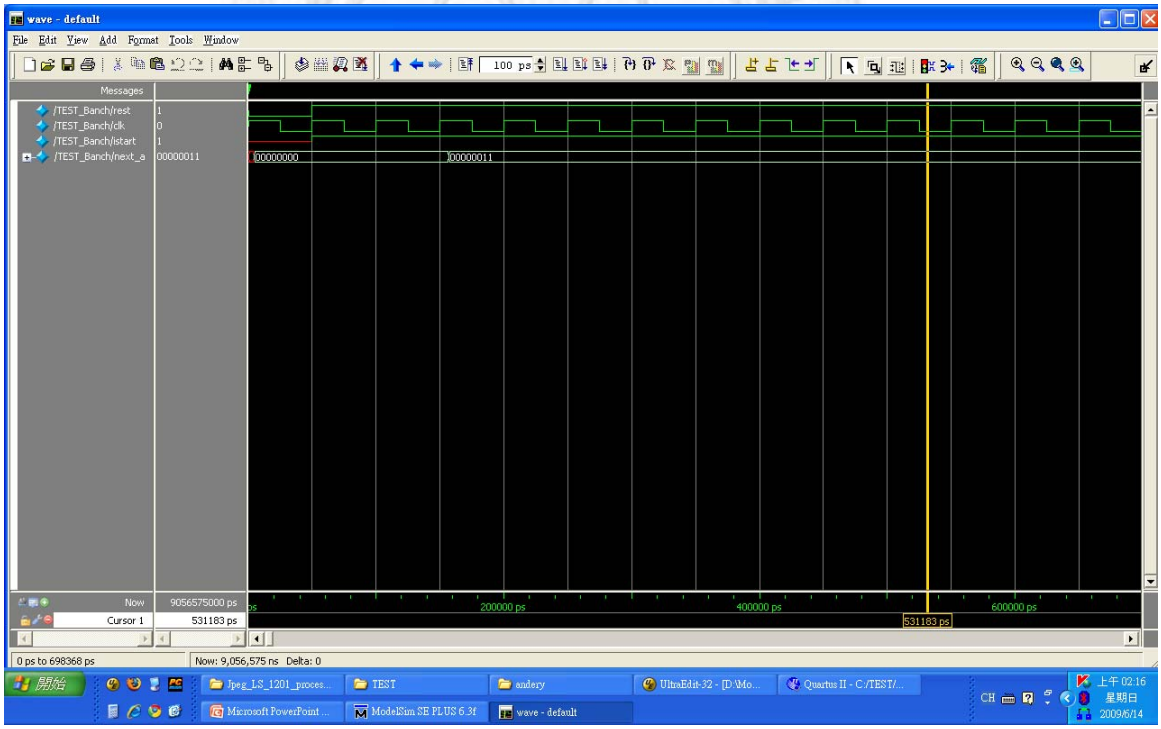


圖 4-5: Source_Image1.bmp 和 Source_Image2.bmp 加快速度

```
always@(negedge rest or posedge clk)
begin
  if(!rest)
  begin
     state <= 0;
     a     <= 0;
  end
  else
  begin
     state  <= next_state;
     a      <= next_a;
  end
end
always@(*)
begin
  case(state)
  8'h00:begin
     next_state= 8'h01;
     next_a    = a; // not change
    end
  8'h01:begin
     next_state= 8'h02;
     next_a    = a; // not change
    end
  8'h02:begin    // change a in this state
     next_state= 8'h03;
     next_a    = 8'h03;
    end
  endcase
end
endmodule
```

圖 4-6: test1.v

```
always@(negedge rest or posedge clk)
begin
  if(!rest)
  begin
     state <= 0;
     a     <= 0;
  end
  else
  begin
     state  <= next_state;
     a      <= next_a;
  end
end
always@(*)
begin
  case(state)
  8'h00:  next_state= 8'h01;
  8'h01:  next_state= 8'h02;
  8'h02:  next_state= 8'h03;
  8'h03:  next_state= 8'h00;
  default:
     next_state = 8'h00;
  endcase
end
wire output_a;
assign output_a =(state == 8'h03)?1:0;
always@(*)
begin
  if(output_a)
    next_a = 8'h03;
  else
    next_a = a;
end
endmodule
```

圖 4-7: test2.v

此張為test1.v所合成後的圖(圖4-8)，合成後共有21 Total logic elements。

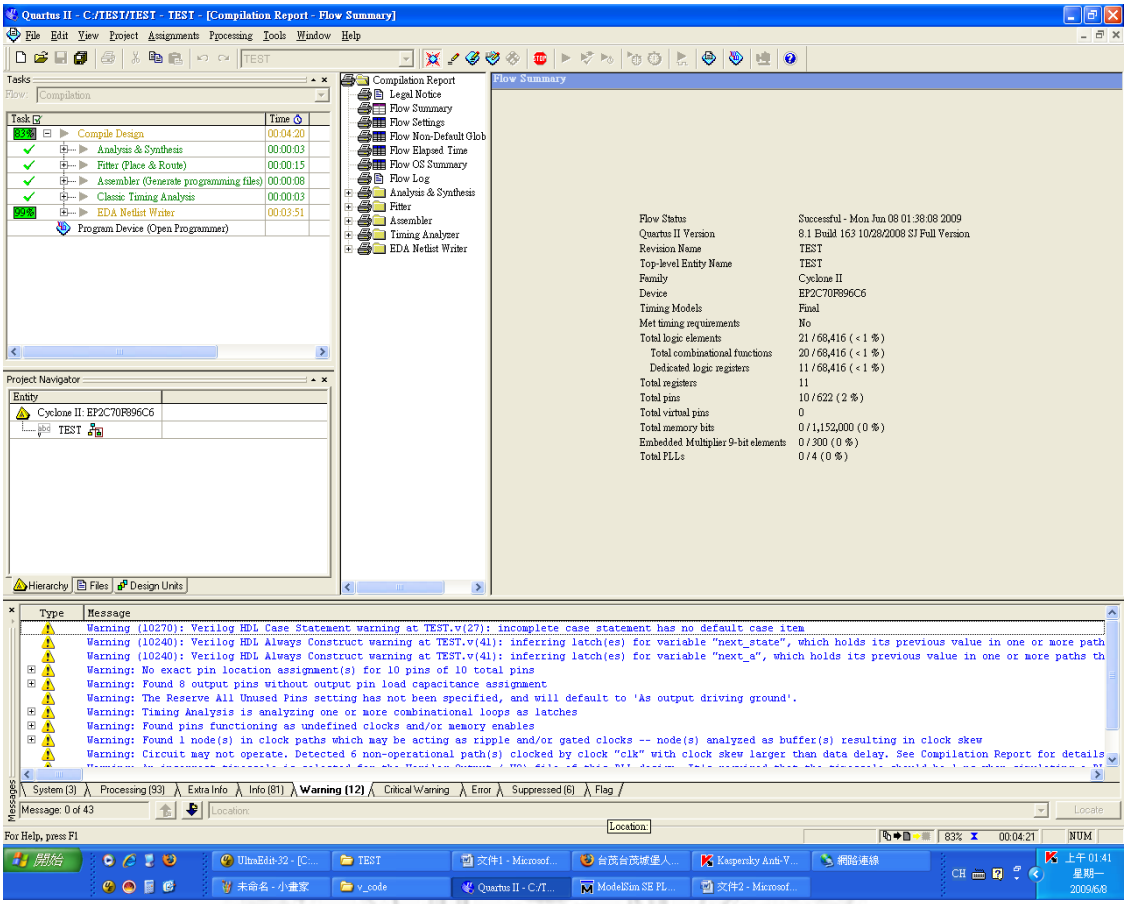


圖4-8: test1.v所合成後的圖

此張為test2.v所合成後的圖(圖4-9)，合成後共有5 Total logic elements。比上一張圖少了16 Total logic elements，是屬於比較好的RTL寫法。

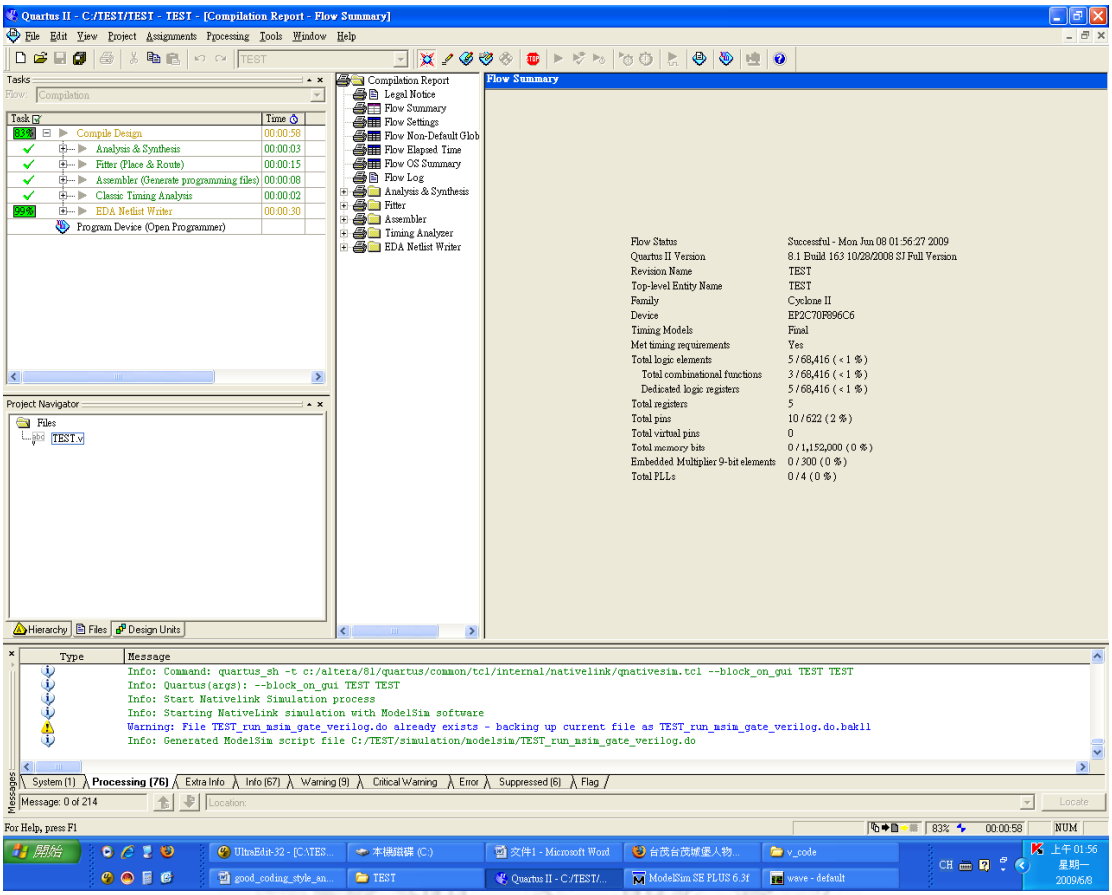


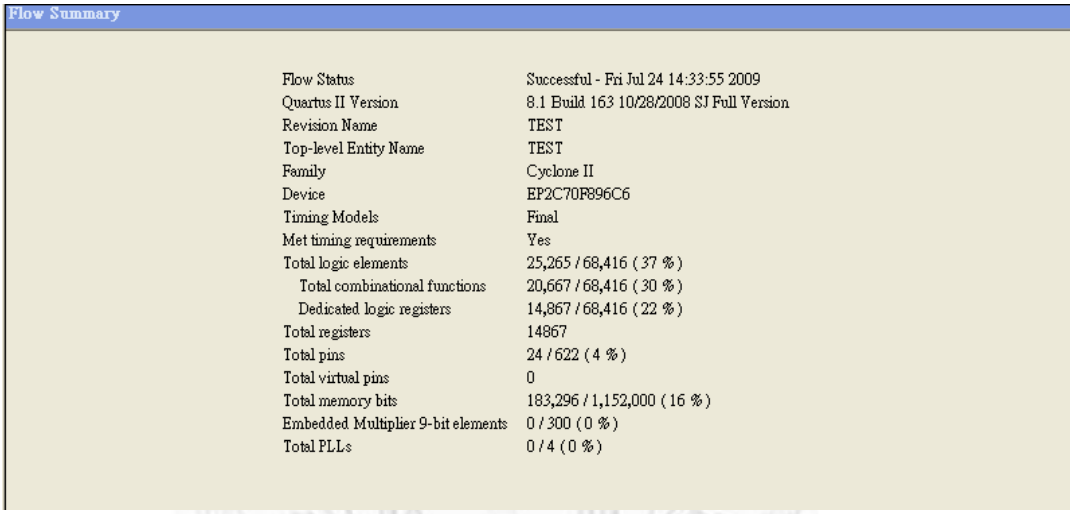
圖4-9: test2.v所合成後的圖

第五章 實驗結果

本章將提出實作 JPEG-LS 硬體加速器的結果，以 Altera[14] QuartusII Development Kit Version 8.1 開發，配合 ModelSim[14]加以驗證。

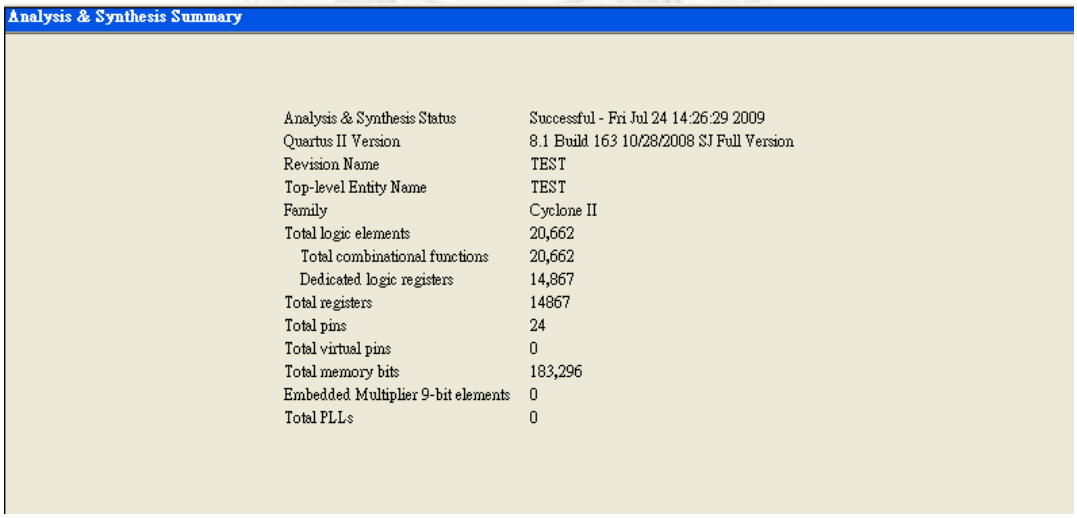
5.1 合成結果

JPEG-LS 硬體加速器經由 Altera QuartusII Synthesis 後的結果如圖(5-1~5-3)所示:


Flow Summary

Flow Status	Successful - Fri Jul 24 14:33:55 2009
Quartus II Version	8.1 Build 163 10/28/2008 SJ Full Version
Revision Name	TEST
Top-level Entity Name	TEST
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	25,265 / 68,416 (37 %)
Total combinational functions	20,667 / 68,416 (30 %)
Dedicated logic registers	14,867 / 68,416 (22 %)
Total registers	14867
Total pins	24 / 622 (4 %)
Total virtual pins	0
Total memory bits	183,296 / 1,152,000 (16 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

圖 5-1: JPEG-LS 硬體加速器的 Flow Summary


Analysis & Synthesis Summary

Analysis & Synthesis Status	Successful - Fri Jul 24 14:26:29 2009
Quartus II Version	8.1 Build 163 10/28/2008 SJ Full Version
Revision Name	TEST
Top-level Entity Name	TEST
Family	Cyclone II
Total logic elements	20,662
Total combinational functions	20,662
Dedicated logic registers	14,867
Total registers	14867
Total pins	24
Total virtual pins	0
Total memory bits	183,296
Embedded Multiplier 9-bit elements	0
Total PLLs	0

圖 5-2: JPEG-LS 硬體加速器的 Analysis & Synthesis Resource Usage summary

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virt...	LUT-Only LCs	Register-Only LCs
Cyclone II: EP2C70F896C6												
TEST	25265 (1009)	14867 (530)	0 (0)	183296	47	0	0	0	24	0	10398 (483)	4598 (77)
Edge_detecting_predictor:Edge_detecting_predictor	66 (66)	30 (30)	0 (0)	0	0	0	0	0	0	0	36 (36)	2 (2)
LineBuffer1:LineBufferA	18099 (18099)	11745 (11745)	0 (0)	0	0	0	0	0	0	0	6350 (6350)	4333 (4333)
ProRun:ProRunA	1418 (435)	671 (262)	0 (0)	0	0	0	0	0	0	0	747 (193)	44 (3)
PutZero:PutZero_A	538 (538)	122 (122)	0 (0)	0	0	0	0	0	0	0	416 (416)	4 (4)
RunModeAndRegularMode:RunModeAndRegularMode	3656 (3602)	1707 (1707)	0 (0)	131072	32	0	0	0	0	0	1948 (1903)	124 (124)
classmap_MyRAM:classmap_MyRAM	399 (399)	10 (10)	0 (0)	0	0	0	0	0	0	0	379 (379)	0 (0)
myRAM_vLUT:myRAM_vLUT_A	91 (91)	52 (52)	0 (0)	52224	15	0	0	0	0	0	39 (39)	14 (14)

Hierarchy | Files | Design Units

圖 5-3: JPEG-LS 硬體加速器的 Resource Utilization

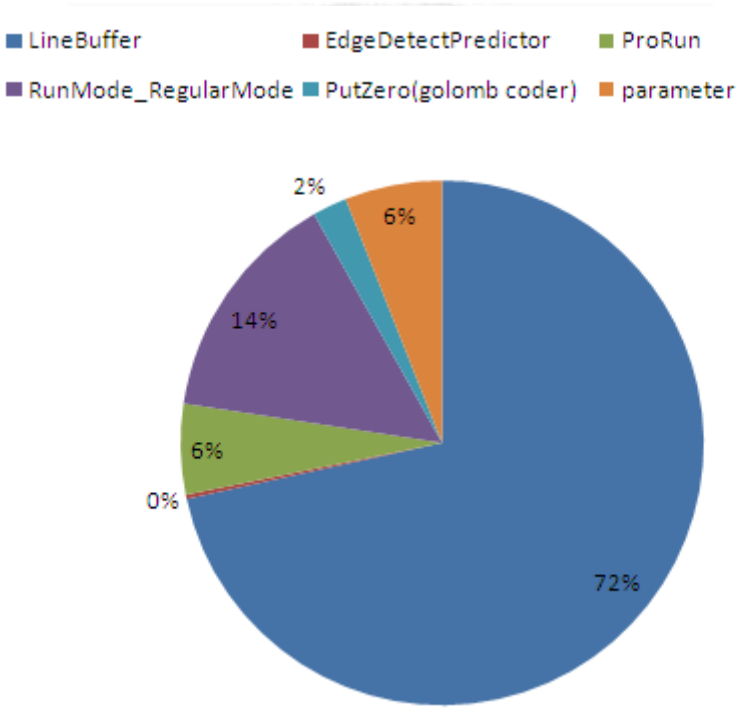


圖 5-4: JPEG-LS Gate count distribution

在圖 5-4 中可以看到 Line Buffer 百分之 72 的 gate count,因為用正反器(Flip-Flop)來儲存輸入的資料。可以提昇資料存取的速度。

在 Tools\Netlist viewers\RTL view(圖 5-5)中可以確認合成後的電路是否符合預期。

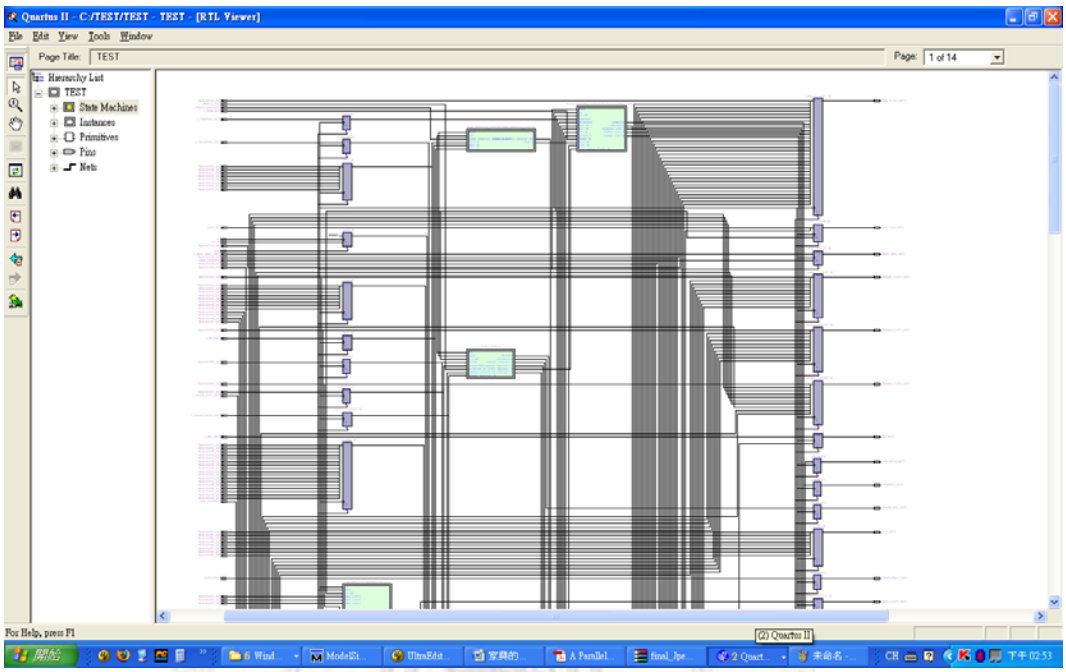


圖 5-5 RTL Viewer

5.2 測試資訊

本實驗使用 2 張圖片來當輸入圖檔分別是圖 5-6 及圖 5-7，皆為 24bit 240*320 的圖片。

模擬速度: 50MHz

圖 5-6 Source_Image1

圖 5-7 Source_Image2

5.3 效能估算

在這章節內，我們驗證方法如圖 5-8，把圖片放入 SD card 中，透過 PIO mode 的方式去讀取 SD card 中的圖片檔案，先放入 buffer 中，1 次讀取 1 Line 丟入 JPEG-LS module 內，而 JPEG-LS 編碼器把壓縮後的 data 先儲存在 Circular buffer 內，CPU 在空閒時再去把 data 從 Circular buffer 讀到 SDRAM 中，我們再 dump 出 data 做比對。

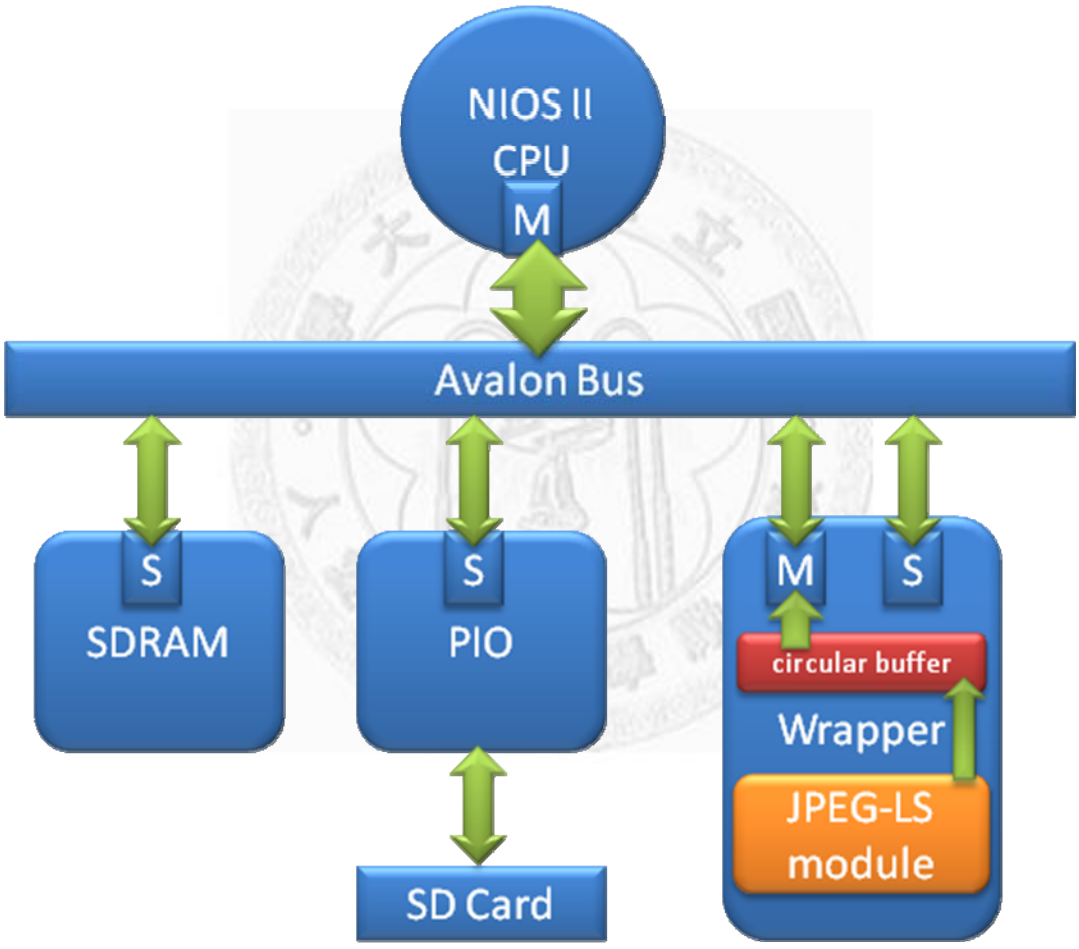


圖 5-8 Test of System Architecture for FPGA

而依據圖示5-9 是分別2張圖片的硬體架構所執行的時間及執行完改良硬體架構所執行的時間，在圖5-10所見，在不增加硬體的情況之下，2張圖像平均加快約1.14倍的速度。

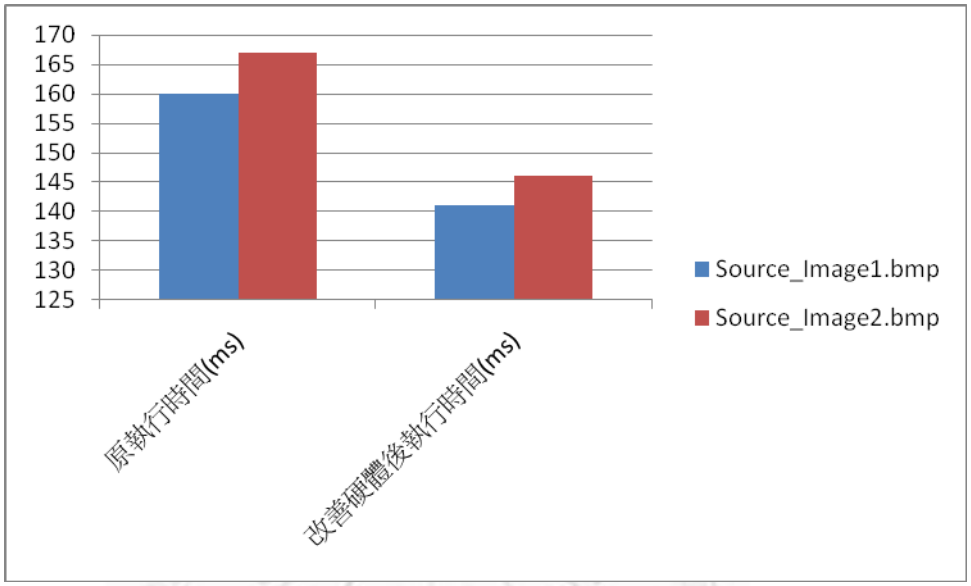


圖 5-9 Source_Image1.bmp 和 Source_Image2.bmp 的執行時間

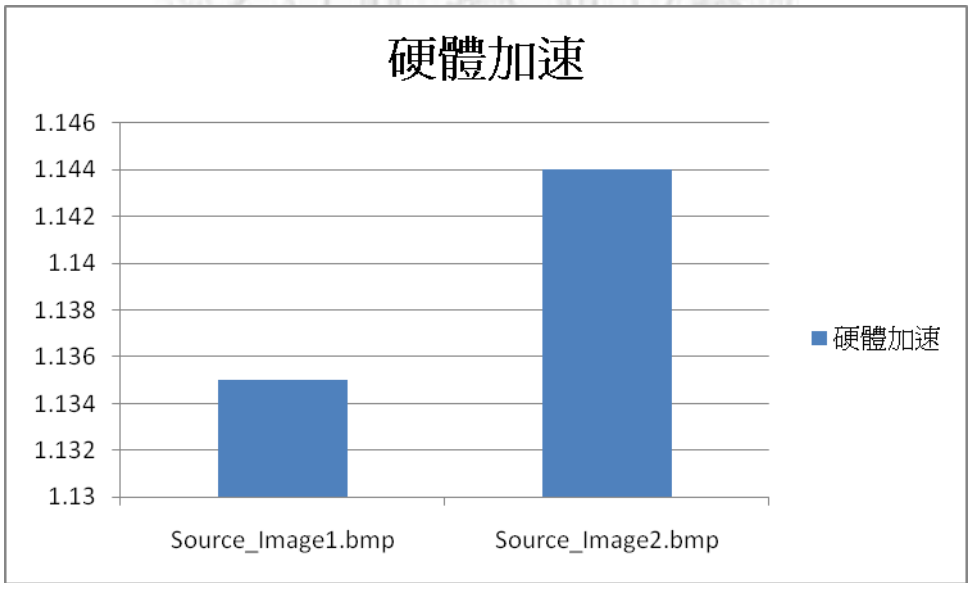


圖 5-10 Source_Image1.bmp 和 Source_Image2.bmp 各別提升速度比

第六章 結論與未來工作

6.1 結論

未來伴隨著醫學電子的蓬勃發展，在醫學影像的處理上勢必越來越受重視，而且畫素也會隨著科技的進步來不斷的提升，若是適當運用無失真壓縮技術不但能方便管理圖片，對於可攜式電子醫學儀器來說也是一大福音。

在此本論文提出在不需要增加額外的硬體電路時，利用平行化、管線化、模組化的特點，改良其電路架構後，就可以加速JPEG-LS的編碼電路約1.15倍的時間，另外本論文提到的，擁有良好RTL coding 技巧是非常重要的，在本論文實作JPEG-LS編碼電路時，只在模擬階段用ModelSim[14]加上QuartusII做模擬，第一次合成出來的檔案直接燒寫在FPGA上即可work，就不需再重新修改IP反覆在FPGA上花費許多時間做驗證。

6.2 未來工作

目前許多論文都以討論JPEG-LS Encoder居多，但是在未來晶片SOC(System On Chip)時應該都會把編碼器和解碼器整合在一起，在"Benchmarking and Hardware Implementation of JPEG-LS[9]"論文中就結合JPEG-LS Decoder and Encoder，並利用其2者會使用到的參數整合在一起，加上模組化的設計，即可縮小整個電路面積，因此若能把本論文的特點結合JPEG-LS編碼解碼器[8]的特點，即能達到面積縮小硬體速度提升的目標。

參考文獻

[1] JPEG/JBIG, FCD 14495, Lossless and near-lossless coding of continuous tone still images, 2000.http://www.jpeg.org/public/fcd14495p.pdf

[2] Xiaolin Chen, Nishan Canagarajah, Jose L. Nunez-Yanez "Hardware Architecture for Lossless Image Compression Based on Context-based Modeling and Arithmetic Coding", SOC Conference, 2007 IEEE International Volume, Issue, 26-29, Sept 2007, pp.251 - 254

[3] Shantanu D. Rane and Guillermo Sapiro, "Evaluation of JPEG-LS, the New Lossless and Controlled-Lossy Still Image Compression Standard,for Compression of High-Resolution Elevation Data", IEEE Trans, Vol.39, issue10, OCTOBER 2001, pp.2298-2306

[4] Jiang,J, Grecos C, "A low cost design of rate controlled JPEG-LS near lossless image Compression", Image & Vision Computing Journal, ELSEVIER, Vol 19, No 3, Feb.2001, pp.153-164

[5] Markos Papadonikolakis, Vasilleios Pantazis and Athanasios P. Kakarountas "Efficient High-Performance ASIC Implementation of JPEG-LS Encoder", Design Automation & Test in Europe Conference & Exhibition, April 2007, pp. 1-6

[6] Xiang Xie, GuoLin Li and ZhiHua Wang, "A Near-lossless Image Compression Algorithm Suitable for Hardware Design in Wireless Endoscopy System", Department of Electronic Engineering, Tsinghua University, Beijing, P. R. China, 100084, ASICON 2005. 6th International Conference On ASIC, Volume 1, 24-27 Oct. 2005 pp. 37 -40

[7] S. W. Golomb, "Run-length encodings" IEEE Trans, Inform.Theory, vol. IT-12, July 1966. pp. 399-401

[8] M Ferretti, M. Boffadossi, "A Parallel Pipelined Implementation for JPEG-LS", International Conference on Pattern Recognition (ICPR'04), vol.1, pp. 769-772.

[9] A. Savakis and M. Pioriun, "Benchmarking and Hardware Implementation of JPEG-LS", International Conference on Image Processing Proceedings(ICIP'02) Vol. 2 Sep.2002, pp.949-952.

[10] S.Hauck, ” The role of FPGAs in Reprogrammable Systems” Proceedings of The IEEE, VOL. 86, NO. 4 Apr.1998, pp.615-639

[11] M. Klimesh, V. Stanton, “Hardware Implementation of a Lossless Image Compression Algorithm Using a Field Programmable Gate Array, ” NASA JPL TMO 2001, Progress Report 42-144

[12] PPM Format Specification, http://netpbm.sourceforge.net/doc/ppm.html

[13] Clunie, D.A. “Lossless compression of grayscale medical images: effectiveness of traditional and state of the art approaches.” Proc. SPIE Medical Imaging, 2000.

[14] http://www.altera.com/

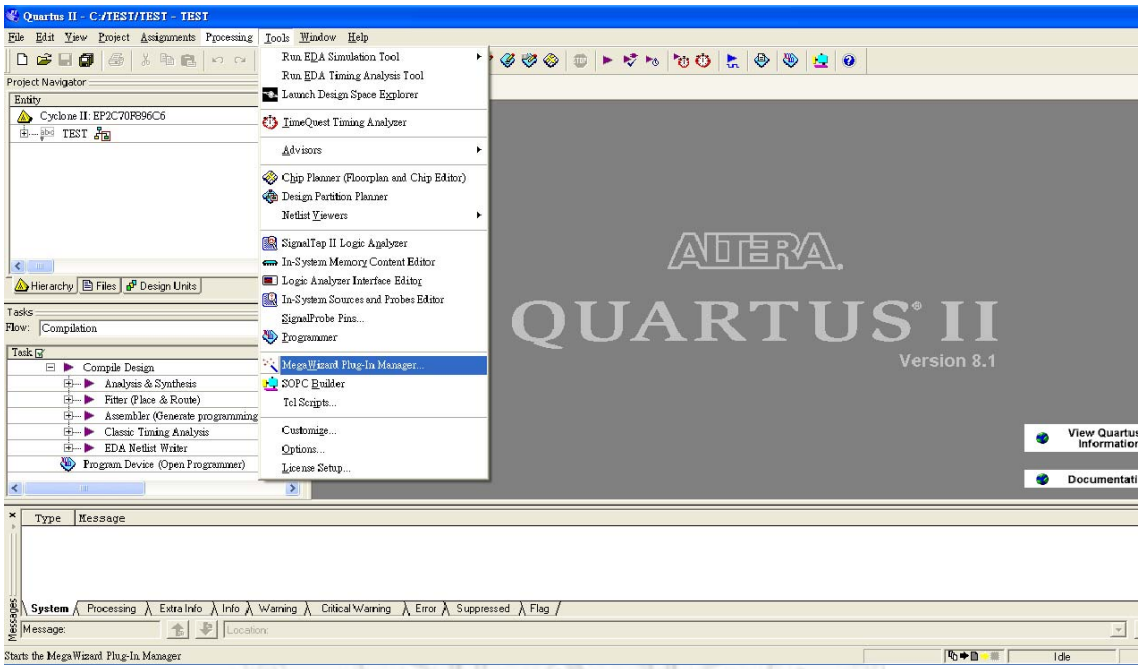
[15] Verilog 數位電路設計, 鄭羽伸 編著

[16] Verilog 硬體描述語言數位電路, 鄭信源 編著

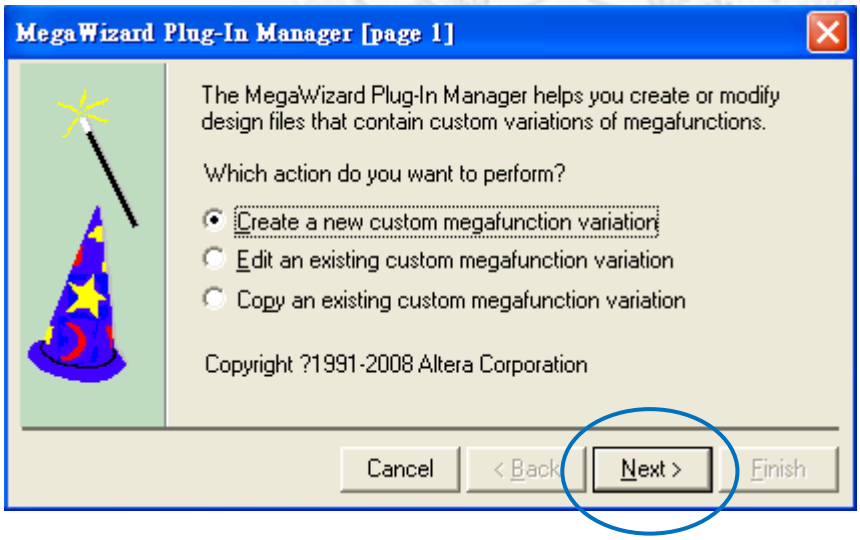
附錄 A.

使用 MegaFunction 來建立 Parameter 的 Buffer.

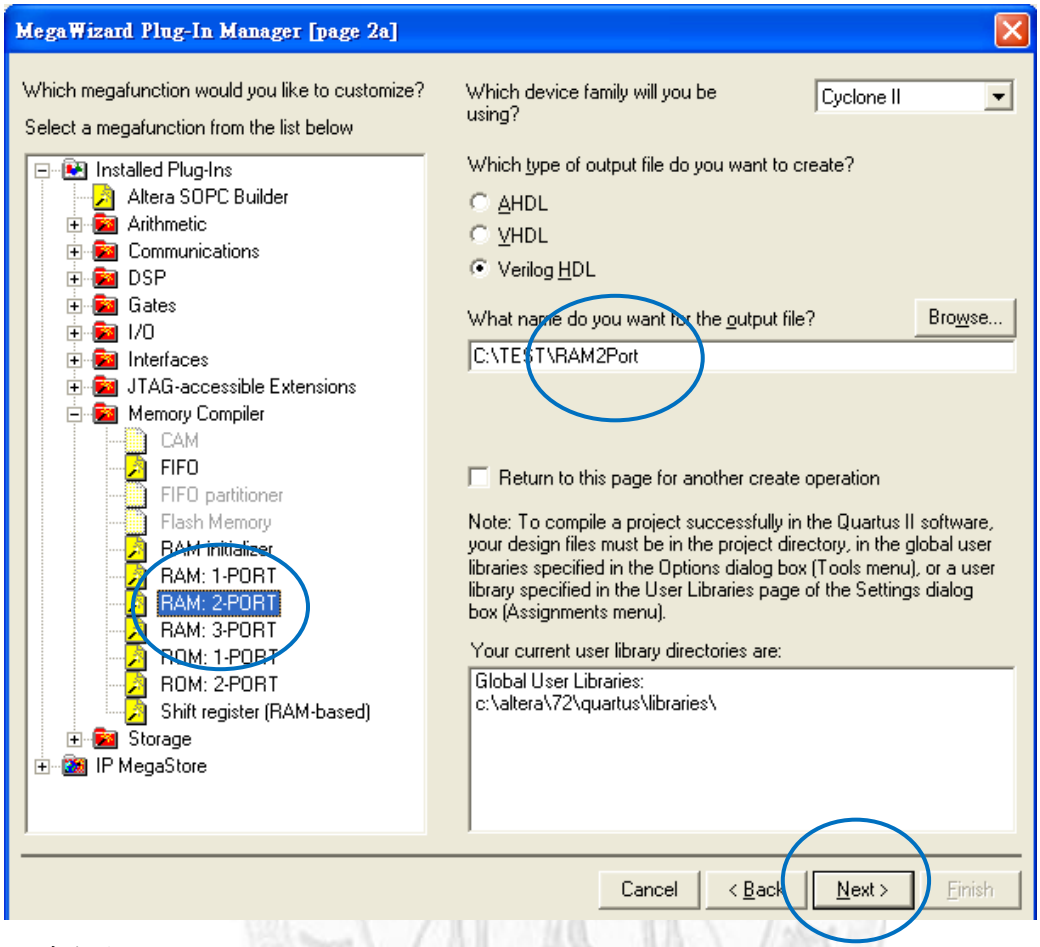
1. 選擇 Tool\Mega Wizard Plug-In Manager…



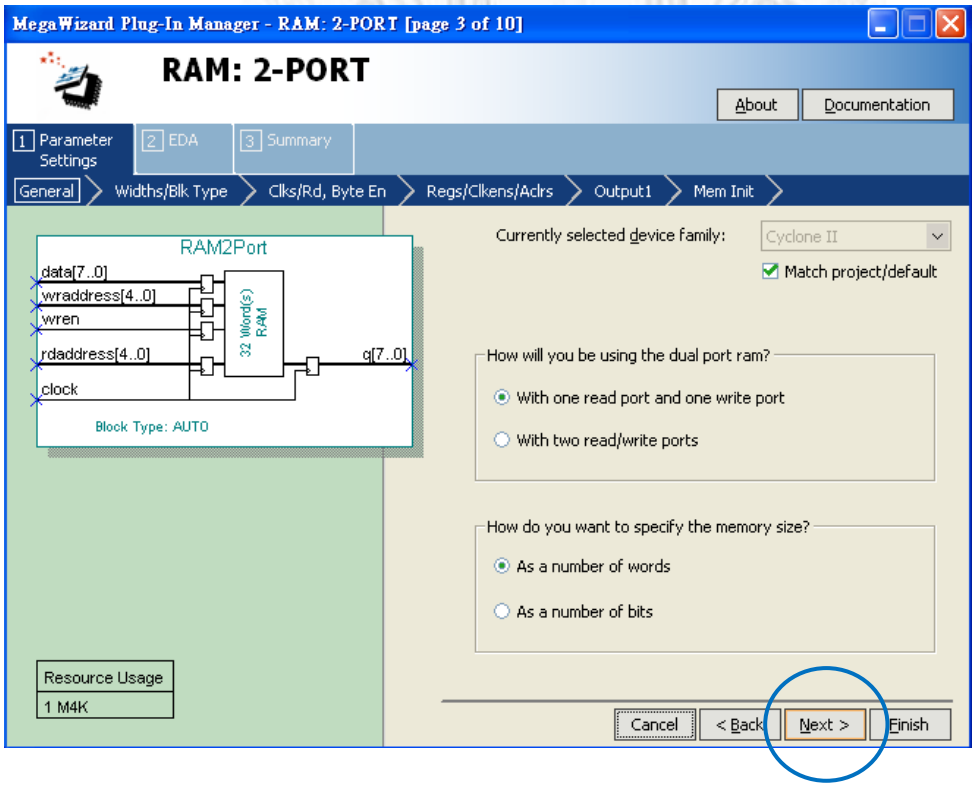
2. 選擇 Create a new custom MegaFunction variation,按 Next



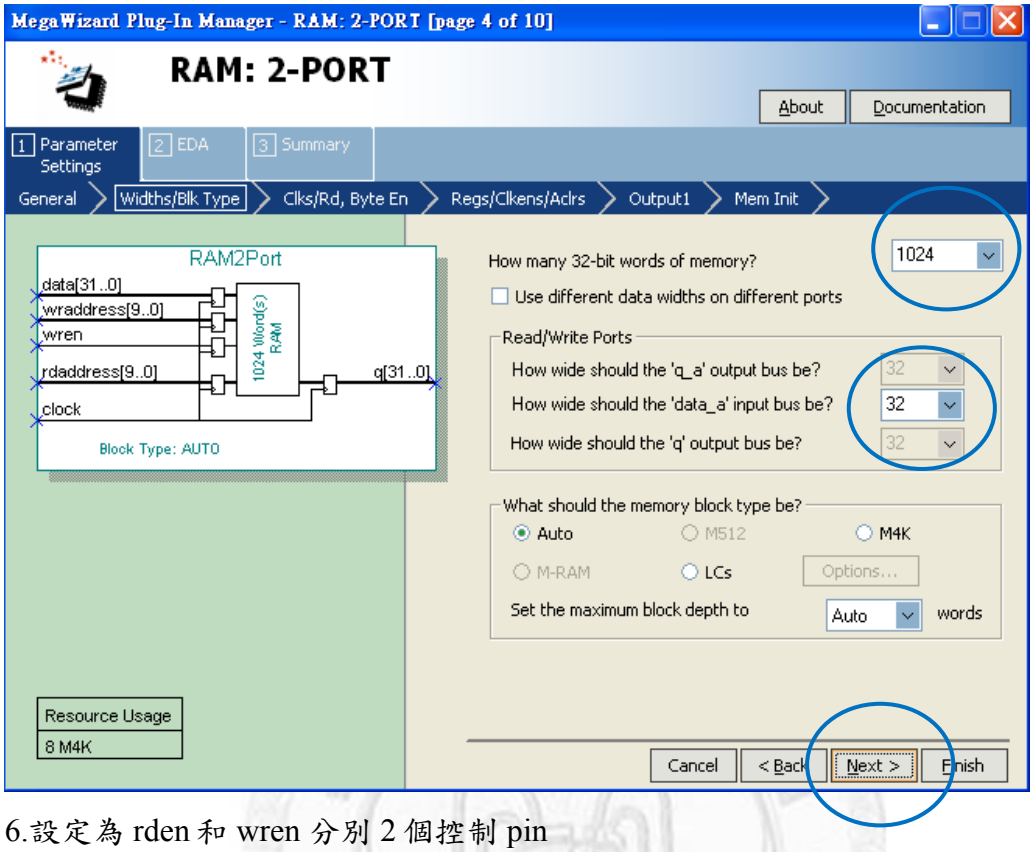
3. 選擇 Memory Compiler 及 RAM : 2-PORT,這裡命名為同名 RAM2Port



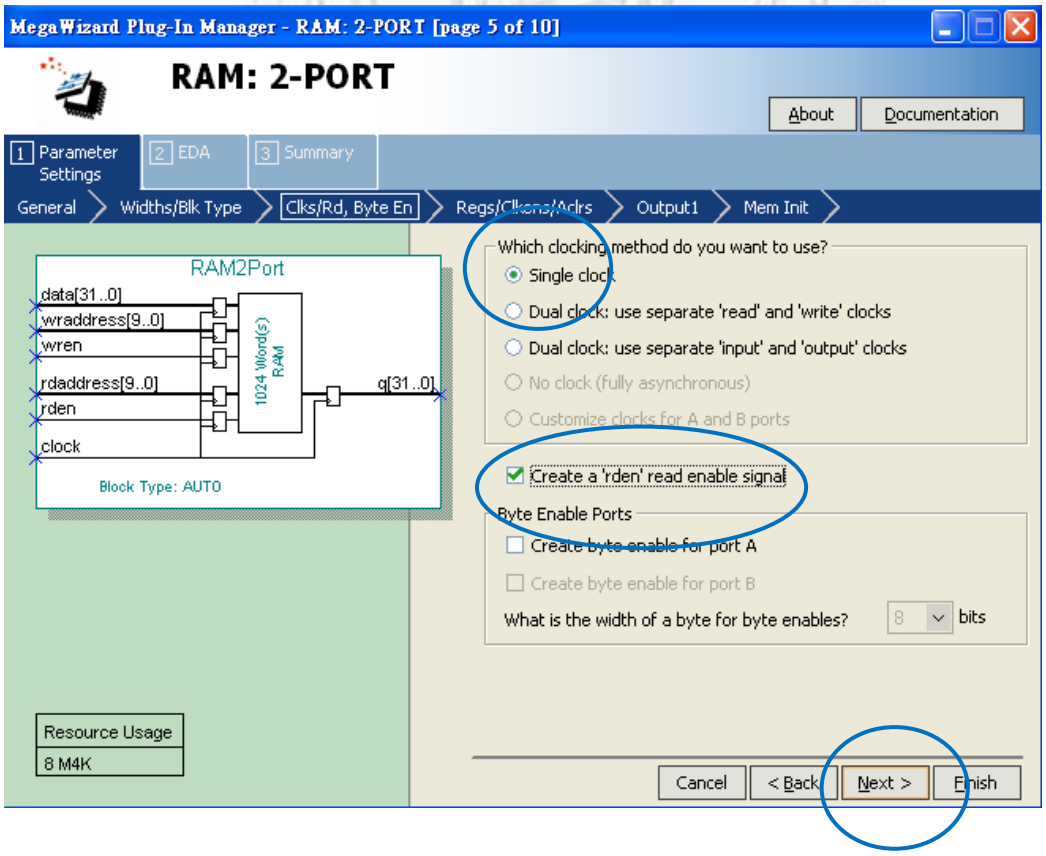
4. 直接按 Next



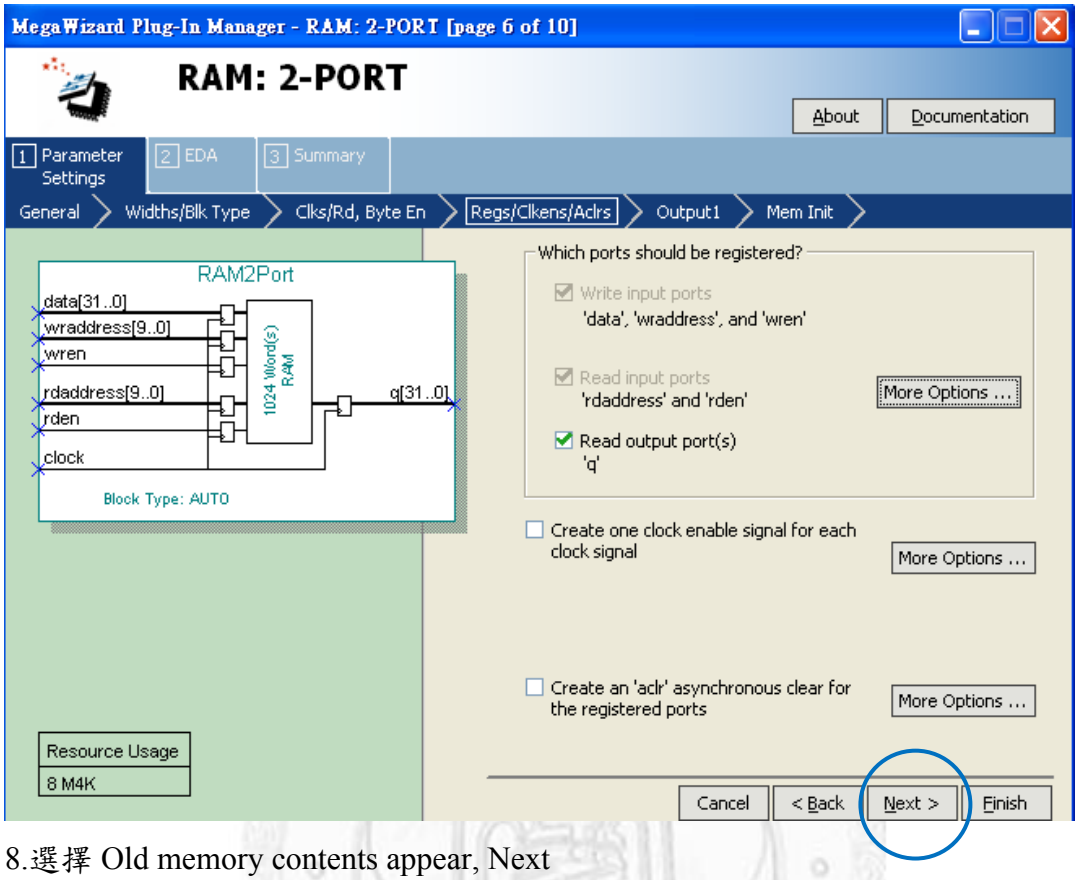
5. 選擇好 memory 大小後按 Next



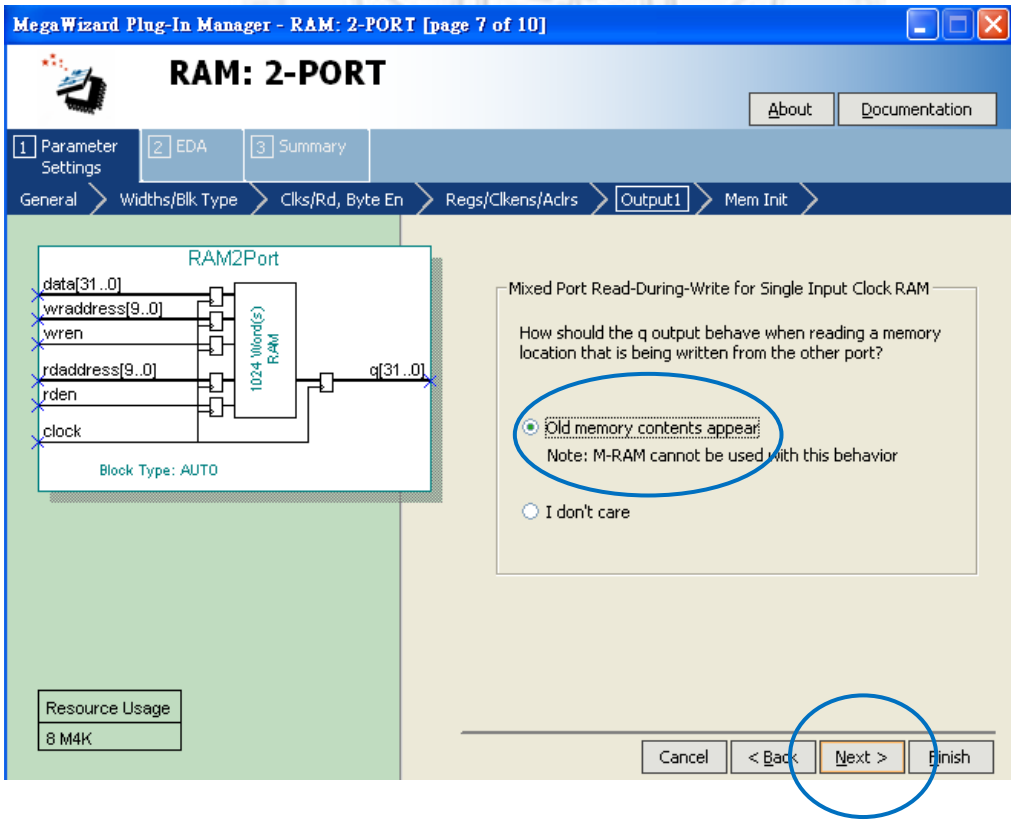
6.設定為 rden 和 wren 分別 2 個控制 pin



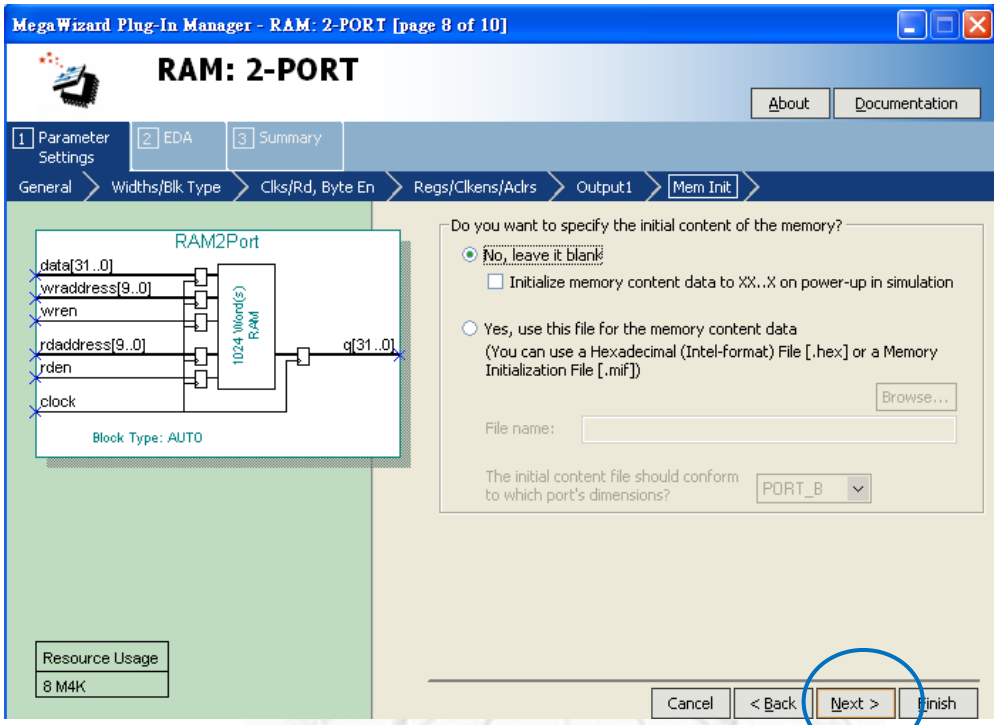
7. 直接按 Next



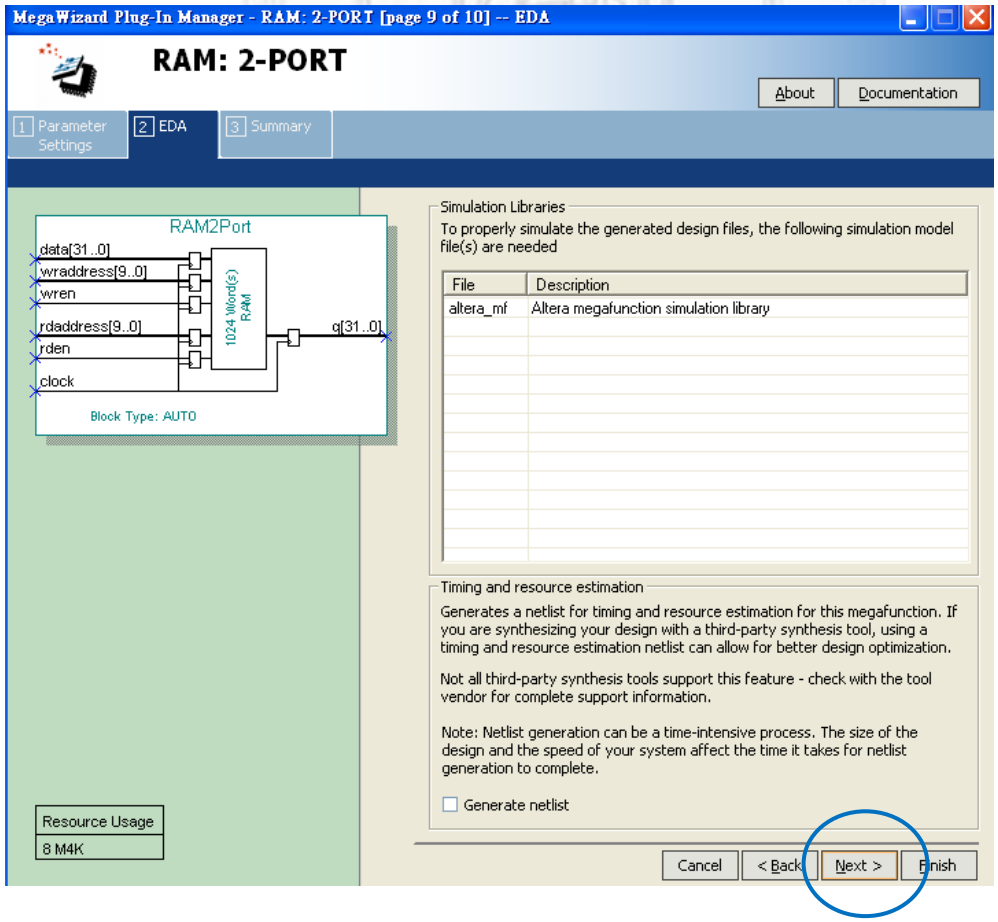
8.選擇 Old memory contents appear, Next



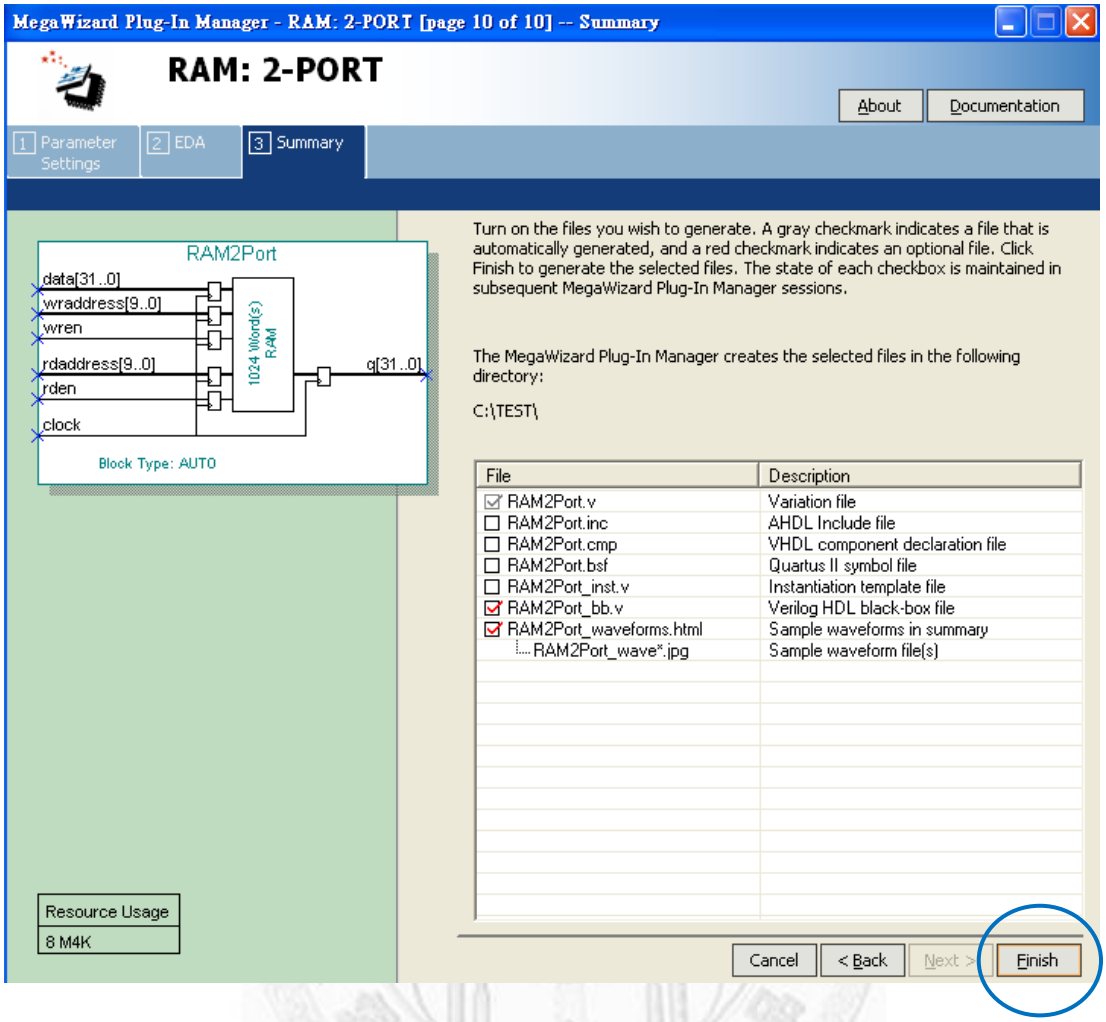
9. Next



10. Next



11.選擇 Finish.



12.資料夾內將多增加 2 個檔案