行政院國家科學委員會專題研究計畫　成果報告

針對 VLIW 處理器提出一有效的容錯設計架構

計畫類別：個別型計畫
計畫編號：NSC94-2213-E-216-016-
執行期間：94 年 08 月 01 日至 95 年 07 月 31 日
執行單位：中華大學資訊工程學系

計畫主持人：陳永源

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 95 年 10 月 27 日

摘　要

在本研究中，我們首先提出一顆可以應用在超長指令集處理器上面的看守狗處理器，來偵測控制流程錯誤。而超長指令集處理器與看守狗處理器整合過程當中會遇到的問題包括: 1. 看守狗處理器要嵌入在超長指令集處理器的那一級; 2. branch 指令處理及如何取得每一個指令區塊的參考標號 (reference signature); 3. 指令區塊內長指令轉換對標號計算的影響; 4. 超長指令集處理器與看守狗處理器之間處理速度如何同步，以確保標號計算的正確性; 5.看守狗處理器的效能。針對以上諸問題提出解決方法，並完成超長指令集處理器與看守狗處理器的整合、驗證與分析。

超長指令集看守狗處理器使用了混合式標號（hybrid signature）技術，包含了垂直標號(vertical signature)與水平標號(horizontal signature)兩種偵測技術，來增加錯誤被查到的機率。利用本研究團隊自行開發的一顆超長指令集處理器，來展示如何將看守狗處理器整合至超長指令集處理器，並經由硬體實踐和灌錯實驗來驗證所提出的標號監督技術在面積、速度、錯誤偵測涵蓋率及錯誤偵測潛伏時間的表現。

另外，我們也提出一個即時錯誤偵測技術架構，來偵測高效能處理器內運算單元的執行錯誤。此偵測技術架構將比較技術、TMR 以及自我檢測技術整合，可以針對不同的硬體複雜度、效能和錯誤偵測能力的需求，來提出適當的錯誤偵測技術。接下來，我們從偵測技術架構中，提出三個不同複雜度的偵測技術，來展現不同偵測技術對硬體複雜度、效能和錯誤偵測能力的影響。並經由超長指令集硬體實踐和灌錯實驗來驗證所提出的技術在面積、效能及錯誤偵測涵蓋率上的表現。此部分的成果請參照 [5-6]。

關鍵詞: 超長指令集處理器 (VLIW processor)，看守狗處理器 (watchdog processor)，控制流程錯誤 (control flow error)，混合式標號監督技術 (hybrid signature monitoring scheme)，錯誤偵測涵蓋率 (error-detection coverage)，錯誤偵測潛伏時間 (error-detection latency)，即時錯誤偵測 (real-time error detection)。

1. 簡介

現代的晶片製程技術已邁向奈米技術層級，所設計出來的晶片，遭受到外來的輻射線、高頻雜訊…等因數，或是晶片本身一些電子物理現象，像是 Crosstalk、IR Drop 等現象，都有可能會造成一些無法預知的錯誤訊號會在我們晶片運算的過程中出現，而錯誤訊號有可能出現在任何地方，所以影響層級就非常廣。假若晶片的指令記憶體受到輻射線干擾，而改變了指令記憶體內部原先儲存的資料，或輻射線影響到程式計數器，就有可能會引起控制流程錯誤（control flow error）進而破壞指令執行順序，而其運算的結果也將不會是原先的運算結果。

目前控制流程錯誤偵測技術主要分為軟體偵測技術(Software Detection Scheme)與硬體偵測技術(Hardware Detection Scheme)兩大主題。無論是哪一類的方法都需要將程式以指令區塊(Instruction Block)的方式切割，每一個指令區塊在執行的過程中並不會有跳躍指令，換句話說，處理器是以循序的方式執行指令區塊裡的指令。

大部分的硬體偵測技術是利用嵌入式標號監督機制 (embedded signature monitoring, ESM)技術來完成控制流程錯誤偵測。嵌入式標號監督機制也可稱為看門狗處理器，一個良好的看門狗處理器的優劣取決於下列五點：1. 錯誤偵測涵蓋率（error-detection coverage）; 2.錯誤偵測潛伏時間（error-detection latency）; 3. 記憶體空間花費（instruction memory space overhead）; 4.看門狗處理器硬體複雜度（watchdog processor complexity）5. 主電腦系統效能下降（main processor performance degradation）。

在過去幾年專家學者對於看門狗技術的研究，大多著重在偵測管線架構處理器的控制流程錯誤，換言之在管線式處理器上面會有控制流程錯誤，相對的在長指令集處理器也會有，目前鮮少有學者探討到

超長指令集處理器的看門狗處理器的設計，所以在本研究將提出完整的超長指令集處理器標號監督機制（VLIW Signature Monitoring）完整架構。

2. 超長指令集處理器技術

圖 1 為與本研究整合的超長指令集處理器的長指令格式（long instruction format）：一道長指令總共包含六個短指令以及六個旗標，I1~I6 均為 32 位元短指令，長指令內的旗標大小皆為 1 位元，旗標是用來決定指令平行運算程度。

圖 1：超長指令集處理器的長指令格式

本研究所引用的超長指令集處理器技術將處理器分為 IF&ID、DRF、EXE、MEM、WB 五級，其硬體架構如圖 2 所示。圖 2 所示的超長指令集處理器架構，為一顆含有容錯技術的超長指令集處理器。超長指令集處理器之容錯技術是針對運算邏輯單元加入容錯保護機制。超長指令集處理器之一道長指令可包含三道運算邏輯指令與三道記憶體存取指令，而超長指令集處理器於 EXE 級，加入一個額外的 ALU 單元，並且利用比較器與 TMR 觀念進行錯誤偵測。當一個時脈週期只有一道運算邏輯短指令時，利用 TMR 觀念來進行錯誤偵測; 二道運算邏輯短指令時，利用比較器觀念; 三道運算邏輯短指令必須分成兩個時脈週期完成三道運算邏輯短指令運算以達到容錯目的。

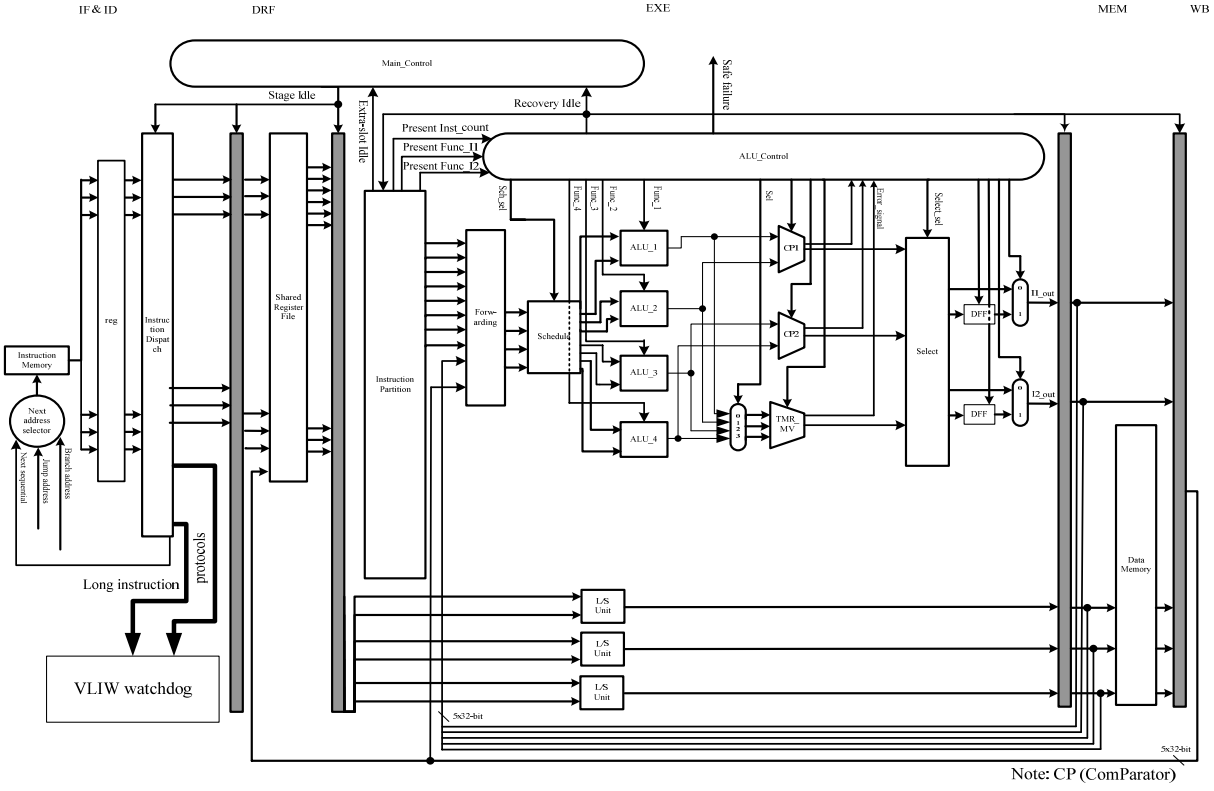


圖 2 超長指令集處理器硬體架構

3. 超長指令集看門狗處理器

3.1 參考標號格式

超長指令集看門狗處理器的參考標號格式（reference signature format）如圖 3 所示，其中 BL 欄位為指令區塊長度（Block Length, BL），其意義是告知看門狗處理器該指令區塊總共有幾道短指令。因為 BL 欄位為 5 位元，所以當指令區塊長度大於 32 道短指令必須進行硬切動作。而垂直參考標號（Vertical Reference Signature）欄位大小為 11 位元，垂直標號的產生是將指令區塊透過垂直標號演算法計算所得。最後一個欄位為 16 位元的水平參考標號（Horizontal Reference Signature），水平標號其目的在於即時錯誤偵測。由於水平參考標號欄位大小有限，所以只對指令區塊前 16 道短指令

進行水平標號的保護。

Block Length BL_B = 5 Bits	Horizontal Signature HS_B = 16 Bits	Vertical Signature VS_B = 11 Bits

圖 3 參考標號格式

3.2 垂直標號演算法

垂直標號演算法最重要的設計觀念是依據兩道短指令位元與位元間的變化情形，在二維座標上來產生屬於這兩道指令的座標。而垂直標號所代表的意義就是，指令區塊的最後座標。首先定義四種狀態，分別是向上（↑）、向下（↓）、向左（←）與向右（→）。此四種狀態的定義如下：

$\begin{matrix}0\\0\end{matrix}$: shift right（→）

$\begin{matrix}0\\1\end{matrix}$: shift left （←）

$\begin{matrix}1\\0\end{matrix}$: shift down（↓）

$\begin{matrix}1\\1\end{matrix}$: shift up （↑）

垂直標號演算法用到的符號定義如下：

BL：Block Length，一個指令區塊長度

W：指令寬度的位元數

I_i ：在block中的第i道短指令，$1 \leqq i \leqq BL$

$I_{i,j}$ ：在block中的第i道短指令，第*j*個位元數 $1 \leqq j \leqq W$

V（I_i ,I_{i+1}）：垂直標號產生函式

定義 1: 假設狀態集合 S = (s_1, s_2, s_3, s_4) $= \left(\begin{pmatrix}0\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}1\\1\end{pmatrix}\right)$。我們定義垂直標號產生函式

$$V(I_i, I_{i+1}) = V\begin{pmatrix} I_{i,1} & \cdots & I_{i,j} & \cdots & I_{i,w} \\ I_{i+1,1} & \cdots & I_{i+1,j} & \cdots & I_{i+1,w} \end{pmatrix}$$

$$= (V_{i+1,1},\cdots,V_{i+1,j},\cdots V_{i+1,W}),$$

$(V_{i+1,1},\cdots,V_{i+1,j},\cdots V_{i+1,W})$ 被稱為一個狀態向量，其中 $V_{i+1,j} \in S$，$1 \le i \le BL-1$，$1 \le j \le W$. □

垂直標號演算法範例：

以 *BL* = 4，*W* = 8 的程式區塊為例，產生該程式區塊垂直標號

I_1 : 1 0 1 0 1 0 1 1

I_2 : 1 0 1 0 1 0 1 1

I_3 : 0 1 0 0 1 1 0 0

I_4 : 1 1 0 1 1 1 1 0

步驟一：

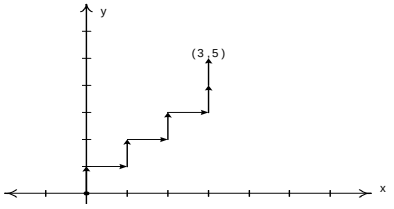
由V（I_1, I_2）產生狀態表。

$$V(I_1 \ , \ I_2) = V\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$= (\uparrow \ \rightarrow \ \uparrow \ \rightarrow \ \uparrow \ \rightarrow \ \uparrow \ \uparrow)$$

步驟二：

根據步驟一產生標號移動圖。



步驟三：

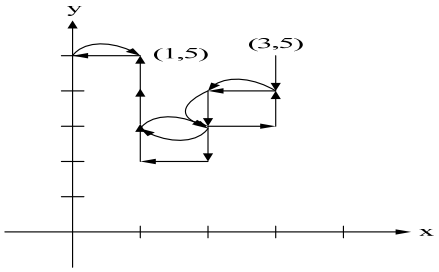
V（$\mathbf{I}_i, \mathbf{I}_{i+1}$）產生狀態表，$2 \leqq i \leqq 3$

(I_2, I_3) (↓ ← ↓ → ↑ ← ↓ ↓)

(I_3, I_4) (← ↑ → ← ↑ ↑ ← →)

步驟四：

根據步驟三所產生的狀態表產生最終標號（1,5）其即為此程式區塊的垂直標號。



3.3 水平標號方法

由3.2所提的垂直標號演算法可得知，垂直標號是在指令區塊結束後才透過標號檢查器來達成

指令記憶體資料保護與控制流程錯誤偵測。但是這對看門狗處理器的錯誤偵測潛伏時間會過長，因此提出水平標號演算法以降低錯誤偵測潛伏時間。水平標號是利用過渡垂直標號（intermediate vertical signature），結合parity來達到同步偵測的觀念。在一個參考標號內，沒有增加記憶體的情況下，大大的縮短了錯誤偵測潛伏時間。

3.4看門狗處理器

在超長指令集看門狗處理器設計上必須特別注意指令平行技術特性、如何取得每一個指令區塊的參考標號、長指令型態所組成的指令區塊與如何將長指令裡面的短指令正確的分配到垂直與水平標號產生器，來計算出垂直與水平的標號並與參考標號作比較來偵測出錯誤的發生。本研究所提出的看門狗處理器為了配合超長指令集處理器的平行高速運算能力，使用了管線技術也使用兩組垂直標號產生器V1,V2，來提升看門狗處理器的運算速度，這裡的V1,V2表示的是3.2節的垂直標號狀態轉換函式。

3.5看門狗處理器短指令分配及標號計算演算法

在說明超長指令集看門狗處理器短指令分配及標號計算演算法之前要先定義以下的符號：

dis_v：所代表的意義為，所要運算的指令區塊，其第一道長指令中有幾道短指令需要做指令分配。

sig_number：所代表的意義為此長指令內含有幾個參考標號，如為一個則 sig_number 設為 0，兩個則設為 1。

fref_sig:所代表的意義為如果長指令中的第六道短指令為參考標號則fref_sig＝1，否則為0。

sbl：所代表的意義為指令區塊還有多少短指令還沒被執行。

$I_{n,p}$：指令區塊中第n道長指令的第p道短指令，$1 \leqq p \leqq 6$。

H（I）：水平標號產生函式。

$I_{n,1}$	Sig	$I_{1,1}$	$I_{1,2}$	$I_{1,3}$	$I_{1,4}$

(a)

$I_{n,1}$	Sig1	$I_{1,1}$	$I_{1,2}$	$I_{1,3}$	Sig2

(b)

圖4：符號說明示意圖

透過圖4說明如何正確使用先前定義之符號。假設圖4（a）為正在運算指令區塊的最後一道長指令，則sbl=1，而且此最後一道短指令一定是跳躍指令。假設其為‘conditional branch’指令，因為VLIW處理器處理‘conditional branch’指令是先猜測不跳，所以其它的符號將被設為：dis_v=4、fref_sig=0、sig_number=0。圖4（b）為一道長指令被嵌入兩個參考標號; 假設圖4（b）為上一個指令區塊的跳躍指令，所跳躍到的長指令。我們會先檢查第一個指令區塊，從$I_{n,3}$開始，符號

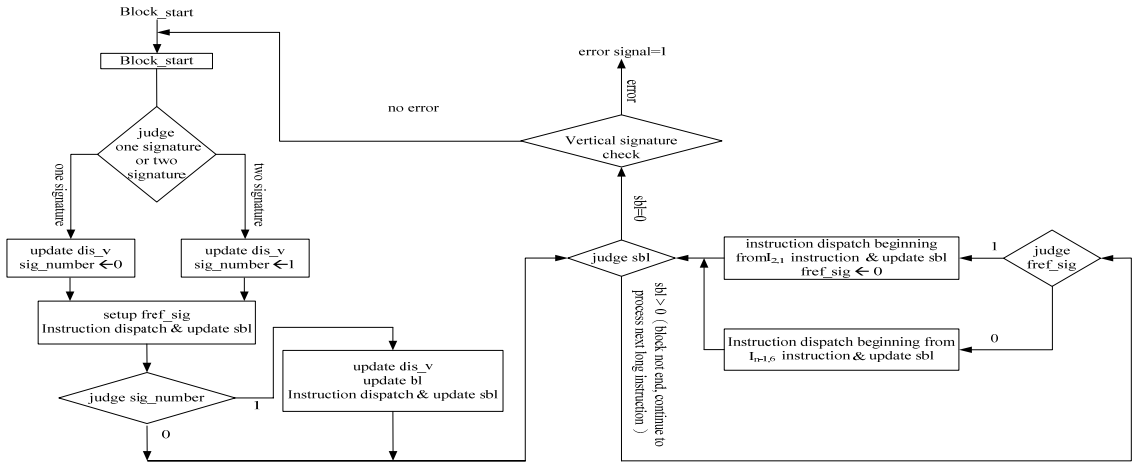


圖 5：指令分配流程圖

將被設為：dis_v=3、fref_sig=1、sig_number=1、bl=3; 第二個指令區塊，其dis_v符號被設為dis_v=0。

圖5為指令分配流程圖，指令分配及標號計算詳細演算法，展示如下:

```
Step1: Block_start
  {if (two signatures in the current long instruction)
   then
      update dis_v;   /*第一個指令區塊需要被
                        分配之指令個數*/
      setup bl;   /*此指令區塊的長度*/
      sig_number ← 1;
   else   /* one signature */
      update dis_v;
      setup bl;
      sig_number ← 0;
   end if；
      setup fref_sig;
      case dis_v is /*短指令分配及標號計算/檢查及修正 sbl */
          when 2
          V1（I1,1，I1,2），V2（idle，idle）;
          H（I1,1），H（I1,2）;
          when 3
          V1（I1,1，I1,2），V2（I1,2，I1,3）;
          H（I1,1），H（I1,2），H（I1,3）;
          when 4
          V1（I1,1，I1,2），V2（I1,2，I1,3）;
          H（I1,1），H（I1,2），H（I1,3）;
          V1（I1,3，I1,4），V2（idle，idle）
          H（I1,4）
          when 5
          V1（I1,1，I1,2），V2（I1,2，I1,3）
          H （I1,1），H（I1,2），H（I1,3）
          V1（I1,3，I1,4），V2（I1,4，I1,5）
          H （I1,4），H（I1,5）
      end case；
       sbl = bl- dis_v；
   If sig_number ＝ 1 then
      update dis_v /*指令區塊二可被指
                     令分配之個數*/
      update bl /*指令區塊二長度*/
      Instruction dispatch/*使用 dis_v 進
                            行下一個指令
                            區塊的指令分
                            配＆modify sbl */
step2： If sbl=0 then
            Jump to step3；
        If fref_sig＝1 then
          case sbl is/*指令分配從I2,1開
                       始＆update sbl */
          when 2
          V1（I2,1，I2,2），V2（idle，idle）
          H（I2,1），H（I2,2）
          sbl = sbl － 2
          when 3
          V1（I2,1，I2,2），V2（I2,2，I2,3）
          H（I2,1），H（I2,2），H（I2,3）
          sbl = sbl － 3
          when 4
          V1（I2,1，I2,2），V2（I2,2，I2,3）
          H（I2,1），H（I2,2），H（I2,3）
          V1（I2,3，I2,4），V2（idle，idle）
          H（I2,4）
          sbl = sbl － 4
          when 5
          V1（I2,1，I2,2），V2（I2,2，I2,3）
          H（I2,1），H（I2,2），H（I2,3）
          V1（I2,3，I2,4），V2（I2,4，I2,5）
          H（I2,4），H（I2,5）
          sbl = sbl － 5
          when others
          V1（I2,1，I2,2），V2（I2,2，I2,3）
          H （I2,1），H（I2,2），H（I2,3）
          V1（I2,3，I2,4），V2（I2,4，I2,5）
          H （I2,4），H（I2,5）
```

V1（$I_{2,5}$，$I_{2,6}$），V2（idle，idle）
H （$I_{2,6}$）
sbl = sbl － 6
end case；
fref_sig ← 0；
Jump to step2；
Else /* fref_sig =0*/
case sbl is/*指令分配從$I_{n-1,6}$開始&update sbl */
when 1
V1（$I_{n-1,6}$，$I_{n,1}$），V2（idle，idle）
H（$I_{n,1}$）
sbl = sbl － 1
when 2
V1（$I_{n-1,6}$，$I_{n,1}$），V2（$I_{n,1}$，$I_{n,2}$）
H（$I_{n,1}$），H（$I_{n,2}$）
sbl = sbl － 2
when 3
V1（$I_{n-1,6}$，$I_{n,1}$），V2（$I_{n,1}$，$I_{n,2}$）
H （$I_{n,1}$），H（$I_{n,2}$）
V1（$I_{n,2}$，$I_{n,3}$），V2（idle，idle）
H （$I_{n,3}$）
sbl = sbl － 3
when 4
V1（$I_{n-1,6}$，$I_{n,1}$），V2（$I_{n,1}$，$I_{n,2}$）
H（$I_{n,1}$），H（$I_{n,2}$）
V1（$I_{n,2}$，$I_{n,3}$），V2（$I_{n,3}$，$I_{n,4}$）
H（$I_{n,3}$），H（$I_{n,4}$）
sbl = sbl － 4
when 5
V1（$I_{n-1,6}$，$I_{n,1}$），V2（$I_{n,1}$，$I_{n,2}$）
H（$I_{n,1}$），H（$I_{n,2}$）
V1（$I_{n,2}$，$I_{n,3}$），V2（$I_{n,3}$，$I_{n,4}$）
H（$I_{n,3}$），H（$I_{n,4}$）
V1（$I_{n,4}$，$I_{n,5}$），V2（idle，idle）
H（$I_{n,5}$）
sbl = sbl － 5
when others
V1（$I_{n-1,6}$，$I_{n,1}$），V2（$I_{n,1}$，$I_{n,2}$）
H（$I_{n,1}$），H（$I_{n,2}$）
V1（$I_{n,2}$，$I_{n,3}$），V2（$I_{n,3}$，$I_{n,4}$）
H（$I_{n,3}$），H（$I_{n,4}$）
V1（$I_{n,4}$，$I_{n,5}$），V2（$I_{n,5}$，$I_{n,6}$）
H（$I_{n,5}$），H（$I_{n,6}$）
sbl = sbl － 6
end case
Jump to step2；
End If；/* fref_sig */
step3：If vertical signature check is ok then
error signal ← 0；
Jump to step1；
Else
error signal ← 1；
End If；
End If；}

為了簡化上面的演算法，我們省略了在每一個H（I）：水平標號檢查後面需加上判斷水平標號的結果，如果有錯則error signal ← 1 的式子。指令分配及標號計算演算法必須得知第一道長指令內含有幾個參考標號（如果含有 2 個參考標號，必須對同一道長指令內不同的指令區塊進行指令分配及檢查）；判斷長指令中的第六道短指令是否為參考標號（假設參考標號在第六道短指令，則下一道長指令分配將從$I_{2,1}$開始分配，假設在其他位置從$I_{n-1,6}$開始分配）；利用dis_v符號進行指令分配與設定sbl符號（假設長指令含有 2 個參考標號則必須進行 2 次指令分配與設定sbl符號）。在完成第一道長指令分配，利用判斷sbl符號來判斷指令區塊是否結束，結束則比對垂直標號，未結束則以fref_sig符號選擇指令分配與更新sbl符號。在完成指令分配後，利用已更新過的sbl符號判斷指令區塊是否結束。因為本研究整合的超長指令集處理器對branch指令是預測不跳，所以當sbl符號歸 0 必須比對標號與利用同一道長指令進行下一個指令區塊的運算，而如果程式區塊未結束，則持續指令分配直到sbl符號變成 0 為止。

3.6超長指令集看門狗處理器硬體架構

超長指令集看門狗處理器，面對到以長指令型態所組成的指令區塊，在處理上會遇到以下幾個問題：

- ◆ 如何知道指令區塊其參考標號儲存的位置
- ◆ VLIW 與看門狗處理器如何同步處理指令，以保證標號計算的正確性
- ◆ VLIW 與超長指令集看門狗處理器之間如何溝通，以達到同步處理指令。

1.參考標號位置（sig_loc）：由超長指令集處理器解碼指令區塊第一道長指令後通知看門狗處理器，參考標號在長指令中的所在位置。

2.長指令轉換訊號（long instruction change, lic）：由於一個指令區塊有可能跨過兩道長指令以上，所以當超長指令集處理器擷取新的長指令需要藉由 lic 訊號告知超長指令集看門狗處理器，長指令已經轉換。

3.起始訊號（v_start, 1 bit）：在程式開始執行或指令區塊結束且 branch 指令跳躍成立，此訊號會被設為 1。此時看門狗處理器會執行清除動作，準備新的指令區塊的檢查。

4.長指令（Long_inst, 192 bits）：與指令匯流排連結，以求得指令訊號。

當看門狗處理器在收到v_start觸發訊號，圖6多工器H會對sig_loc訊號進行解碼，再搭配看門狗處理器主要控制電路（top_control）對多工器A~E發出正確控制訊號，完成第一個垂直時脈週期的短指令分配與擷取參考標號動作。

當 lic=1 時，多工器 H 會分配正確的輸入訊號給多工器 B~D，多工器 A 選擇 feed back 訊號，因為垂直標號產生器所需要的短指令有可能在長指令中的任何一道短指令，同時也有可能需要發出 nop 訊號，再加上垂直標號產生器一共有四端短指令輸入端，所以一共需要四組八對一的短指令分配器，其分別為多工器 A~D。

由於垂直記號硬體架構技術有使用到管線技術，所以圖 6 registerA~D、registerG、Horizontal Signature 與 Vertical Signature Register 為看門狗處理器中的管線暫存器。

圖7垂直標號運算單元及檢查電路使用兩組垂直標號產生器，左半邊為計算X軸座標電路。右半邊電路則是計算Y軸座標電路。當看門狗處理器重新運算一個指令區塊，則必須透過clean mux電路將座標進行歸零的動作。

垂直標號技術只會在指令區塊結束才會進行驗證比對的動作，所以在指令區塊運算過程中，需要有兩個大小為 12 位元的暫存器分別儲存已做完運算的 X 座標與 Y 座標。

在水平標號架構的硬體設計方面，利用資源共享的觀念，從圖 6 register A 與 register B 來完成水平標號的指令分配。但是還是有一些情況沒辦法達到完全地資源共享，必須增加一些硬體電路來完成整個水平標號架構，假設指令區塊最後一道短指令由圖 6 register D，提供給垂直標號產生器，就必須透過多工器 G 完成短指令分配，如果當垂直標號單元進入閒置狀態，而水平標號單元還沒運算完畢，多工器 I 與多工器 J 用來選擇要 register 進入垂直標號運算單元或選擇 NOP 訊號送入垂直標號運算單元，而我們就可以利用多工器 A 與多工器 B 完成水平標號所需要的指令分配。圖 8 的 H_control 輸入訊號由看門狗處理器主要控制電路提供。

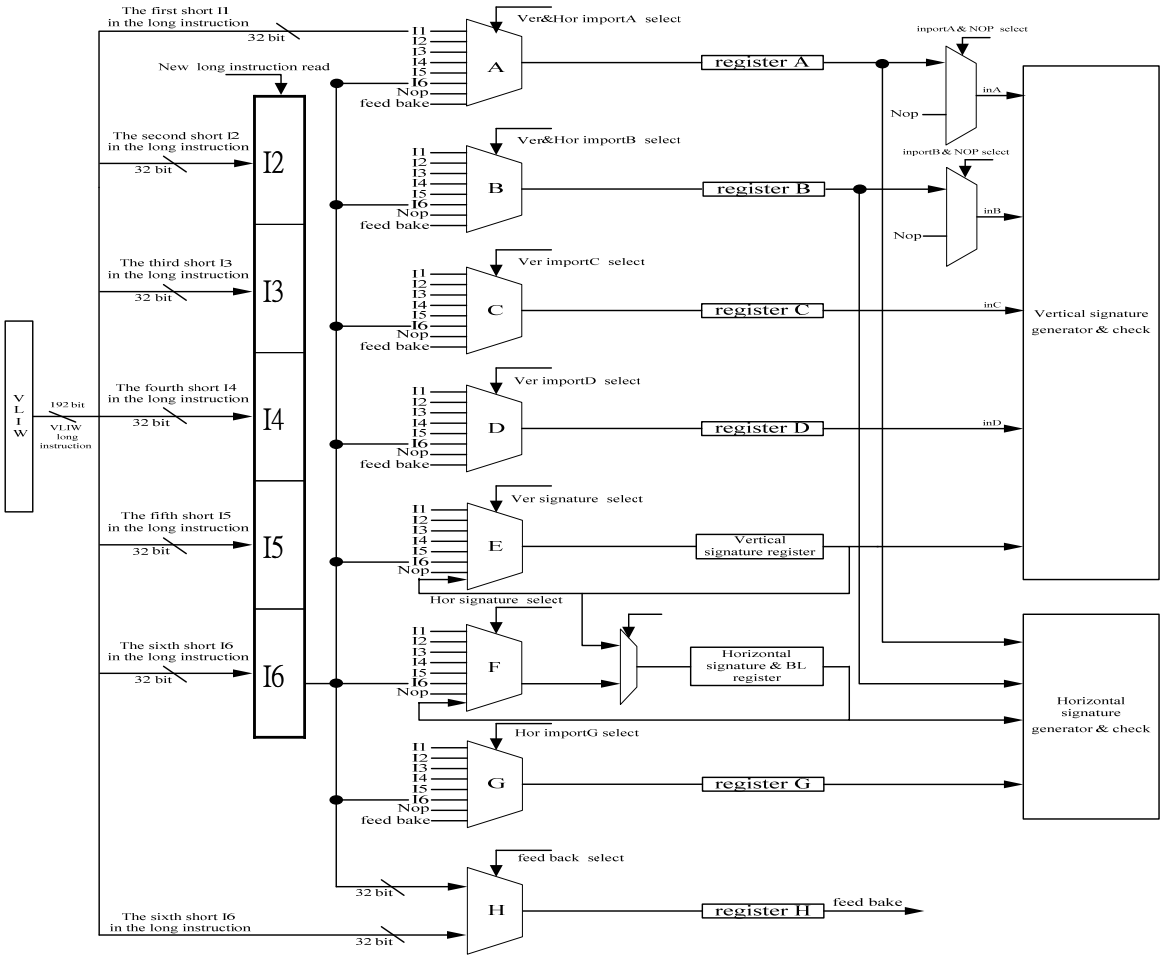


圖 6：看門狗處理器短指令分配電路

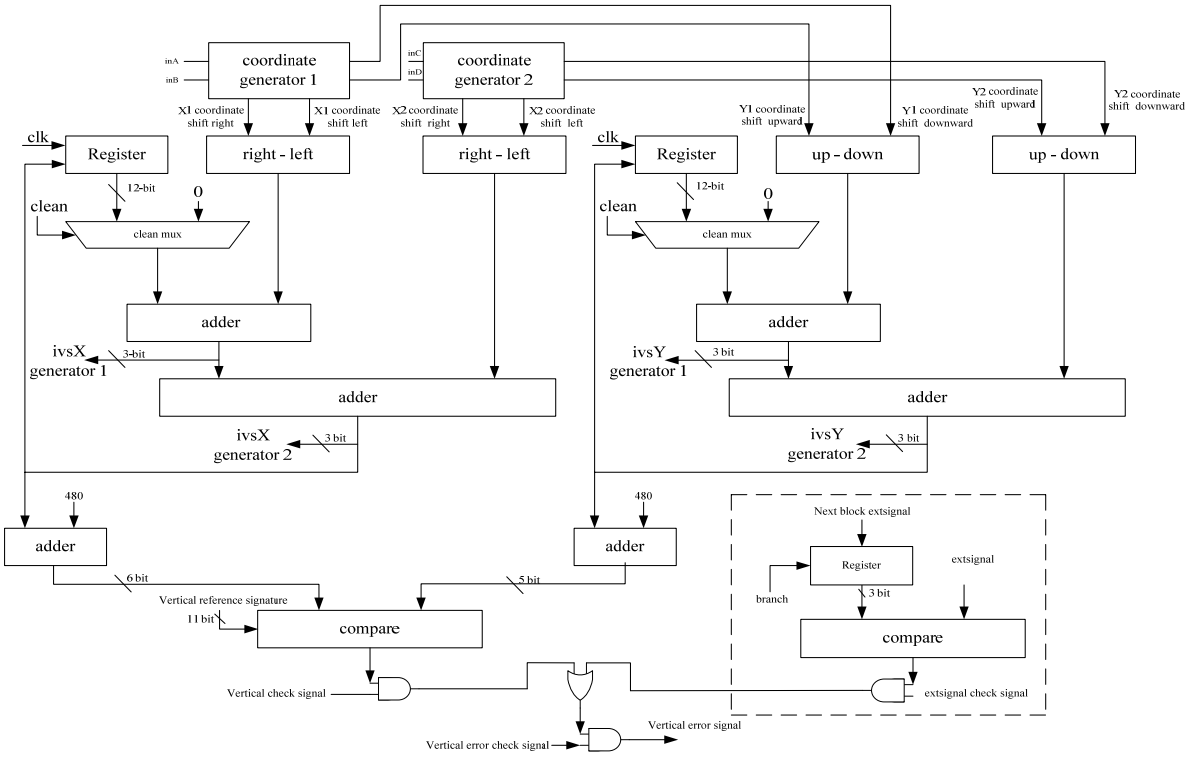


圖 7:垂直標號運算單元及檢查電路

在水平與垂直標號整合過程中，時脈同步問題，我們使用除頻器作為水平與垂直標號硬體的時脈介面。指令區塊長度問題：利用 length 訊號線解決。若垂直標號運算前 16 道短指令，length 訊號將被設為 0，後 16 道短指令即設為 1。當 length 訊號線為 1 時，即不再檢查水平標號。

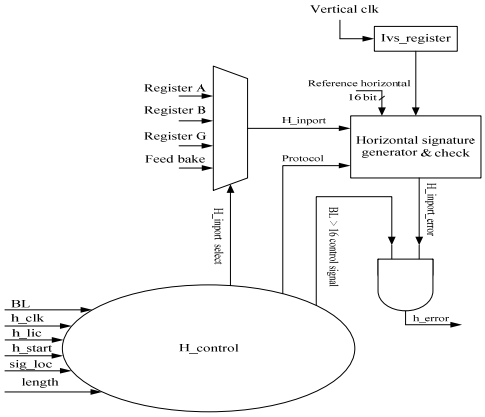


圖 8:水平標號運算及檢查硬體架構圖

4. 超長指令集處理器與看門狗處理器整合

在超長指令集看門狗處理器在設計上面會比傳統管線式看門狗處理器設計上要困難許多，除了需要面對時脈與溝通訊號問題，還有超長指令集看門狗處理器特性問題。

4.1看門狗處理器跳躍至錯誤指令區塊起頭問題

當超長指令集處理器的位址產生電路運算出錯誤指令位址，而此錯誤指令位址，剛好為另一個指令區塊起頭。此類控制流程錯誤是無法偵測到。

此類問題的解決方案:如圖9所示，在指令區塊A的最後一道短指令的跳躍指令的第16~14位元將存放指令區塊C的sig_loc（在組譯階段寫入），即為指令區塊C的參考標號位置是放在指令區塊C第一道長指令裡的那一道短指令。而超長指令集處理器將透過溝通訊號將程式區塊C的sig_loc（來自指令區塊A最後一道短指令），寫入圖7虛線電路暫存器內。假設超長指令集處理器擷取到程式區塊C的第一道長指令，並解碼完後將sig_loc設為指令區塊C第一道長指令的sig_loc（假設位址產生電路出錯，且錯誤位置剛好是另一個指令區塊X的起頭其sig_loc有可能與指令區塊C不同，就可查出錯誤的發生）。

最後在藉由圖 7 虛線內部 compare 元件輸出便可得知，跳躍後的控制流程是否出現錯誤。

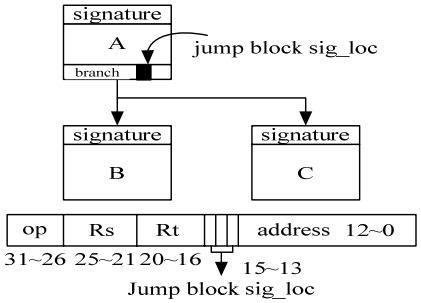


圖 9:指令區塊跳躍範例圖

4.2 超長指令集看門狗處理器嵌入在超長指令集處理器的位置

選擇一個正確位置嵌入超長指令集看門狗處理器，將會對偵錯潛伏時間、記憶體空間花費與看門狗處理器硬體設計複雜度等等有直接影響。我們將會對超長指令集處理器架構上 IF&ID 級與 EXE 級這兩個地方進行分析，並以上述影響參數為依據來決定看門狗處理器的嵌入位置。

若以偵錯潛伏時間而言，嵌入在 EXE 級會比嵌入在 IF&ID 級慢了 2 個 VLIW 時脈週期，因為中間需要經過 2 條管線。而且在長指令的資料流程過程中完整的長指令會被拆散掉，所以記憶體空間花費上面，必須把經過的 2 條管線每一條各別加長 192 位元，來儲存一完整原始的長指令，方能將看門狗處理器整合於 EXE 級。對看門狗處理器硬體設計複雜度而言，嵌入在 EXE 級無須考慮長指令滿載問題（長指令滿載的意思即一個 VLIW 長指令內的六道短指令的運算可以同時執行，也就是說六道短指令可以在一個時脈週期同時執行）。

若將看門狗處理器嵌入於 IF&ID 級，硬體設計複雜度方面，必須在圖 6 付出 5 道短指令大小記憶體空間協助看門狗處理器解決長指令滿載情況。透過以上分析我們決定讓看門狗處理器嵌入於 IF&ID 級以求得更好的效能。

4.3 超長指令集處理器與看門狗處理器之時脈與溝通訊號整合

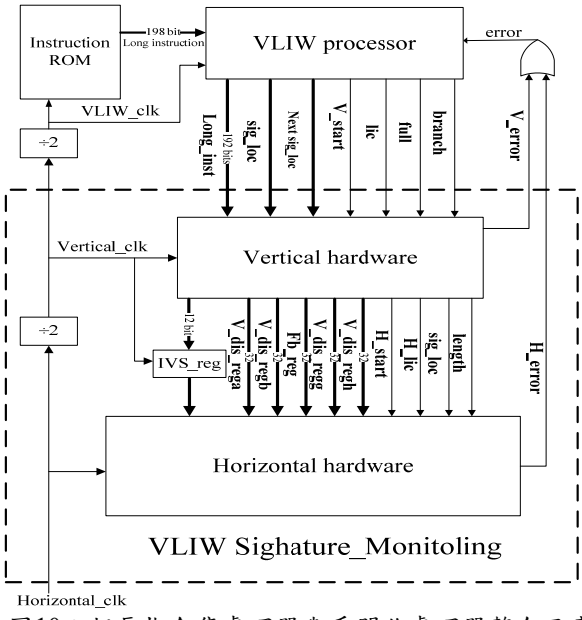


圖10：超長指令集處理器與看門狗處理器整合示意圖

從圖10可以看出利用除頻器的觀念來整合超長指令集處理器與看門狗處理器。由圖11看到超長指令集處理器會送出包含 lic 、sig_loc、V_start、next block extsigna1、full signal、branch、與192bits的長指令，給看門狗處理器，來達成超長指令集處理器與看門狗處理器的溝通與協調。其中full訊號線用來通知看門狗處理器，目前超長指令集處理器現在所運算的長指令為滿載。branch訊號線，為4.1節，圖7虛線電路暫存器的

觸發訊號。

5. 模擬實驗與結果分析

本章節將單獨對不同測試程式做相同灌錯模擬實驗，最主是要探討不同程式特性對偵錯涵蓋率、偵錯潛伏時間與記憶體空間花費的影響。並且提供完整看門狗處理器的面積與速度，作為往後設計參考依據。

5.1程式指令區塊長度分析與測試程式選擇

我們選擇兩種不同型態測試程式：

型態一：所需要的特性為程式指令區塊長度特別短，且 branch 指令跳躍情況非常平凡的程式特性。選用 heapsort 測試程式

型態二：型態二之測試程式所要的特性為指令區塊長度特別長，且branch指令很少出現在測試程式中的程式特性。選用IDCT測試程式

5.2記憶體空間花費

由於系統加入了看門狗處理器，系統必須將參考標號嵌入至記憶體中，進而造成記憶體空間負擔。由表1可知，因為heapsort程式區塊長度都很短，相對地，必須嵌入較多的參考標號。而IDCT測試程式因為程式區塊長度都比較長，所以會嵌入較少的參考標號。

program	Memory overhead
IDCT	4.5%
heapsort	18.18%

表1：記憶體空間負擔結果

5.3硬體空間花費

由於系統加入了標號監督機制，必須負擔看門狗處理器硬體面積，表2為硬體空間花費結果表，表中VLIW數據是沒有容錯系統的超長指令集處理器的面積跟速度。製程為UMC 0.18 μm。表中看門狗處理器數據包括標號產生電路與解決超長指令集標號監督機制的特定電路，共花費7.44%的硬體空間，來達到控制流程錯誤與指令記憶體的容錯機制。

program	Area(um²)	Speed(ns)
VLIW	10948196	7.85
Watchdog	815457	1.87
overhead	7.44%	

表2：硬體空間花費結果表

5.4效能下降

在過去管線式看門狗處理器一遇到參考標號，系統就必須閒置，因此造成系統效能下降。而此種情況並不會出現在超長指令集看門狗處理器，因為超長指令集處理器一次擷取一道長指令，所以系統可以直接分配其他的短指令給超長指令集處理器運算，因此超長指令集看門狗處理器並不會造成電腦系統效能下降。

5.5偵錯涵蓋率、偵錯潛伏時間實驗規劃

在硬體實現部分，是透過硬體描述語言實現，模擬灌錯平台建立在RTL層級。每次的模擬實驗只灌一個錯，一個錯持續5個VLIW時脈週期。超長指令集看門狗處理器能偵錯範圍，包括：偵測控制流程錯誤與指令記憶體資料保護，所以在程式灌錯模擬實驗，細分成為二個不同目標：目標1（target 1）：指令記憶體資料保護模擬實驗，直接透過硬體描述語言將多重位元錯誤（multiple-bit error）注入在指令記憶體的輸出端，模擬指令記憶體內部資料出錯。

目標2（target 2）：偵測控制流程錯誤模擬實驗，將單一位元錯誤（single-bit error）注入在有可能造成超長指令集處理器控制流程錯誤之訊號線或位址產生元件，如程式位址計數器和位址產生器。

5.6不同灌錯地點模擬實驗比較

本節將探討，當錯誤發生在指令記憶體與控制流程以及程式特性，對看門狗處理器偵錯涵蓋率與偵錯潛伏時間的影響與差別，表3表示實驗的模擬結果。

heapsort	target1	target2
EDC	0.992	0.996
EDL	2.23	1.63
NDET	5	3

IDCT	target1	target2
EDC	0.981	1
EDL	4.2	1.97
NDET	13	0

表3:綜合實驗模擬結果

表3的上方是針對heapsort，執行1500次灌錯實驗，所得到的結果。因為heapsort測試程式指令區塊都很短，所以每道短指令可分得較多的水平參考標號位元，進而有較理想的偵錯潛伏時間與偵錯涵蓋率，且別名（aliasing）發生的機會也會較低。但是另一方面發生跳到不正確的指令區塊起頭如4.1節所描述的情況也會比較高。我們先從Target2的EDC(error-detection coverage)欄位開始看起，Target2為模擬控制流程錯誤。在表3中，3次未偵測到的錯誤(Not Detect, NDET)都是發生在超長指令集處理器跳躍發生，且剛好跳到不正確的指令區塊起頭，而此時4.1節所用的技術也恰巧查不到這種錯誤。接下來探討heapsort測試程式的Target 2的偵錯潛伏時間，在EDL（error-detection latency）欄位所顯示數據是以VLIW時脈週期為單位，指令區塊長度短，其水平標號技術分得較多水平參考標號，因此EDL較短。

表3的Target1灌錯模擬實驗裡，灌錯位置位於指令記憶體的輸出端，而在指令記憶體的資料保護模擬實驗注入最多5位元多重位元錯誤，在700次灌錯模擬實驗，結果顯示偵錯涵蓋率EDC為0.992，從表3得知錯誤被漏掉的5次，都是發生在別名情況。

接下來將探討為何Target1偵錯潛伏時間會大於Target2偵錯潛伏時間？Target2的錯誤為控制流程錯誤，所以錯誤的長指令一進來就馬上可以透過水平標號技術來偵測，而在Target1是一定要等到長指令中有錯誤的短指令進到超長指令集標號監督機制才有可能被查到，所以Target1的偵錯潛伏時間會比Target2還要長一點。

表3的下方是對IDCT執行1500次模擬實驗結果。因為IDCT測試程式指令區塊都很長，所以每道短指令所分到的水平參考標號位元數較少。進而有較長的偵錯潛伏時間，且發生別名發生的機會也會較高，所以會得到較低偵錯涵蓋率。但是相對的發生跳到不正確的指令區塊起頭如4.1節所描述的情況也相對變少。我們從Target2 EDC欄位開始看起，表3的下表可以發現並沒有出現跳到錯誤指令區塊起頭的情況，也沒有發生別名情形，實驗結果的EDC為1。至於Target2偵錯潛伏時間，由於水平標號技術只會針對前面16道短指令進行保護，後16道短指令若出錯，只能靠垂直標號技術將錯誤偵測出來。

接著探討表Target 1的EDC欄位，因為在IDCT測試程式分配到的水平參考標號位元數較少，所以水平標號偵測涵蓋率也較低，導致IDCT測試程式的偵錯涵蓋率較低。在EDL方面，因為錯誤出現在第16道短指令之後，會造成需等到垂直標號技術將錯誤偵測出來，所以偵錯潛伏時間變長。

6.結論

在本研究中，提出了一顆通用型，與微小硬體面積的超長指令集看守狗處理器，來幫助超長指令集處理器達到指令記憶體以及控制流程錯誤的偵測。在超長指令集看守狗處理器，利用三種資訊分別是，指令區塊長度（block length），垂直標號（vertical signature），水平標號（horizontal signature），來幫助看守狗處理器，得到了一個很高的偵錯涵蓋率與非常理想的偵錯潛伏時間，進而達到讓錯誤擴散出去的危險降到最低，所以本研究提出來的超長指令集看守狗處理器適合用在及時容錯系統上。

7. Self-Evaluation of Research Results

- The abstract summarizes the results accomplished from this project. It is clear that most of the work has been achieved and published or submitted to be considered for publication. However, the subjects described in our proposal are big and deserve to be further explored. We definitely achieve the goals set in the proposal.
- The VLIW-based embedded systems have found fertile ground in mobile applications. Since more works depend on the portable machines, the reliability issue becomes more important than ever. The results obtained from this research can be applied to the embedded VLIW processors to enhance the overall system dependability. The previous study for the fault-tolerant microprocessors mainly focuses on the superscalar architecture and rarely discusses the fault tolerance in VLIW. We want to fulfill this lack.

8. Publications associated with this research:

[1] Y. Y. Chen and K. F. Chen, "Incorporating Signature-Monitoring Technique in VLIW Processors," *19th IEEE Intl. Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 395-402, 2004. (EI)

[2] **Yung-Yuan Chen,** Kuen-Long Leu and Chao-Sung Yeh, "Fault-Tolerant VLIW Processor Design and Error Coverage Analysis" submitted to *IEEE Trans. on Computers* (full paper), July 2006. (SCI, EI) (NSC 92-2213-E-216-005 and NSC 93-2213-E-216-019)

[3] **Yung-Yuan Chen,** "Concurrent Detection of Control Flow Errors by Hybrid Signature Monitoring," *IEEE Trans. on Computers*, Vol. 54, No. 10, pp. 1298-1313, October 2005. (SCI)

[4] Shu-Seng Lai and Yung-Yuan Chen, "Incorporating Signature-Monitoring Technique in VLIW Processors," submitted to Journal of Information Technology and Applications, August, 2006.

[5] **Yung-Yuan Chen,** Kuen-Long Leu and Li-Wen Lin, "Hybrid Error-Detection Approach with No Detection Latency for High-Performance Microprocessors," International Conference on Computer Designs, pp. 196-202, June 2006.

[6] **Yung-Yuan Chen**, Kuen-Long Leu "Concurrent Error Detection Using Hybrid Approach for High-Performance Microprocessors," submitted to *Journal of Computer Science and Technology* (Full paper), Sept. 2006. (SCI) (NSC 94-2213-E-216-016)