

八十八學年度大學校院
積體電路電腦輔助設計軟體製作競賽
競 賽 報 告

報名編號： A19

組 別： 定題組

<題目>

Subcircuit Identification

中華民國八十九年 6 月 16 日

一、摘要 (Abstract)

從一個超大型積體電路中搜尋辨識(identify)或者擷取(extract)出一個副電路(subcircuit)之重要性持續地提升，此一功能在許多積體電路(IC)的設計製造以及測試驗證上面，均具有相當廣泛程度的應用，例如在積體電路的佈局與電路之對應驗證(LVS)方面、積體電路的測試，甚至在做超大型積體電路之切割(partition)時，副電路之擷取與搜尋，均提供相當程度的協助。

在本報告中，我們提出一個通用的資料結構(data structure)，用以將所輸入之標準CDL格式之主電路(target circuit)，建構轉換(mapping)成為一個類似graph之模組(model)；並且提供一較佳之權重函式(weighting function)，根據此一權重函式，配合一個遞迴式的圖形辨認演算法(recursive graph identification algorithm)，快速地完成副電路辨識以及擷取(identification and extraction)的工作；附帶一提的是，此一方法具有電路設計方法(design-style independent)以及製程(technology independent)皆無關的特性，可適用於各種電路之辨識擷取。

二、簡介 (Introduction)

2.1 問題描述 (Problem Definition)

CDL是一個普遍性之描述電晶體層次(transistor level)之電路的格式，在這樣的一個描述方式中，階層式(hierarchy)的描述方式被廣泛應用，而假使我們所得到的CDL格式檔案採取這樣的階層式的描述，則其實我們可以很容易地做出副電路的擷取，如以下的描述方式：

```
X1 1 2 3 NAND2
X2 2 3 4 NOR2
X3 5 6 7 8 NAND3
X4 8 6 INV
```

我們知道上面的電路中包含一個2-input NAND gate，2-input NOR gate、3-input NAND gate以及一個inverter。

然而，當我們得到一個展開的(flattened circuit)電路時，要看出此電路中所可能包含的副電路，即非一件簡單的工作；而在很多時候我們所能取得的只是無階層描述之電路。

因此，我們的問題即是，從一個展開的電路(flattened circuit)中辨識擷取(identify and extract)出特定的副電路(subcircuit)。我們所持有的輸入資訊是兩個以CDL為描述格式的電路，其一為樣本電路(pattern circuit)，也就是我們所稱的副電路，另一為目標電路(target circuit)，而我們的目標即是從此目標電路(target circuit)中擷取出其中包含之樣本電路(pattern circuit)，並且將目標電路中所對應組成樣本電路的這些電晶體輸出到輸出檔案中。

2.2 軟體功能及特性 (Functions and Features)

此軟體所具備之功能大致如下列數點：

- (1) 如同本專題之題目定義，我們可以從一個CDL格式之flattened circuit中，擷取(extract)出我們所想得到的副電路(subcircuit)。
- (2) 此軟體能夠同時依據使用者之需要與否，將電路對應之連結圖(connection graph)中的各個節點(node)之資訊列印出來，對於瞭解此軟體、在追蹤(trace)程式以及除錯(debug)時皆有相當大的幫助。

此外，此軟體具備以下之特性：

- (1) 使用動態之記憶體宣告方式，能夠節省相當程度之記憶體使用。
- (2) 此軟體能夠適用於任何design style以及technology之使用，而不像一般設計只限於在CMOS電路使用。
- (3) 不採用先對目標電路或者樣本電路作切割的方式，可減少相當多的前置作業時間。
- (4) 藉由使用權重函式(weighting function)，本軟體能夠在搜尋辨識(identify)之前，將那些不可能比對成功的電路節點先篩選過濾掉，因此，節省了相當多的辨識搜尋的時間。
- (5) 此軟體能夠處理相當大的電路結構，不管是對目標電路或是樣本電路皆是如此，並且整個比對搜尋的時間耗時相當短。
- (6) 此軟體能夠在各工作平台(platform)編譯並且執行，達成跨平台的要求，而不限於只能夠在某些平台上執行。

2.3 貢獻及成果簡述 (Contributions and Results)

副電路之辨識擷取(subcircuit identification and extraction)的應用相當廣泛，利用我們所提供的軟體，能夠從一個flattened的大電路中擷取出我們需要的邏輯閘(logic gate)或者小一點的組成電路(subcircuit)之資訊，進而對於超大型積體電路測

試、LVS(Layout Versus Schematic)比對、電路切割等問題提供協助。例如，在超大型積體電路測試方面，利用此軟體得到gate-level之電路描述後，我們可以套用 stuck-at-fault model，對於一個大電路做測試樣本之產生(test pattern generation)以及可測性的分析等；另外，又如在LVS比對上，擷取得到gate-level之電路描述後，可以輕易地與原來以邏輯閘設計之電路做比對，而不須耗費太多時間；或者當我們要對一個電路作邏輯功能檢測(function test)時，經由 subcircuit identification/extraction 得出 gate level 的電路資訊後，可以直接在 gate level 作 function 的驗證，與 transistor level 的驗證相比，能夠節省相當程度的時間。

而經由對於主辦單位所提供之測試檔之驗證、以及我們取用以往設計之實際電路所對應之CDL格式檔案之測試，均展現此軟體程式之正確性，並且如同在前一段我們所提到的，此軟體對於記憶體之使用量、執行速度方面，均有相當突出的成果。

三、演算法 (Algorithms)

此軟體之運作主要包含數個功能，分別為(1)與電路對應之圖形的設置和建立、(2)電路相對應的圖形中各個節點的weight之計算、以及(3)subcircuit與主電路中各個node之 searching/matching 的check。茲分述如下：(可以先參照”4.2 程式流程”，以先瞭解整個比對過程的程序，再回到此部分，可以更清楚本軟體的演算法與程式架構)

(1) Graph Construction :

程式的第一個步驟是從 CDL files 中，將電路本身的資訊收集起來。這樣的動作可分為兩個部份，分別稱做pass1以及pass2：在pass1中，我們建構一個鏈結串列(linked list)，把檔案中第一次出現的節點名稱(circuit node label)放進一個linked list中，同時計算整個list中全部的元素的個數。而在pass2中，我們宣告一個陣列，來儲存全部的節點。並且由檔案中，將電路中各個節點的連結關係建立起來。舉例來說，如下的一列 CDL list：

$$M1 \ A \ B \ C \ vdd \ p$$

表示M1的drain/source neighbor有A、C兩個net node，gate neighbor則只有B一個net node，且M1的type為PMOS。相對應地可知，在A、C的drain/source neighbor中，必須加入M1這個device node，而B的gate neighbor也必須加入M1這個 device node。

若以上述方式，建構出整個電路的結構之後，如果要取M1的neighbor，則只需由M1的 ds_neighbor和g_neighbor即可找到相連的net node。反過來說，也可以由A、C的ds_neighbor或是B的g_neighbor連結到M1這個 device node。因此電路中所有的連結情況都可以以遞

迴的方式來表示。

(2) Weighting Computation :

為了使得比對的時間縮短，我們必須設計一些方法，將一些不可能match的nodes事先刪除，因為一個一個node去做地毯式(exhaustive)的比對必定耗費相當多的時間。此軟體所使用之weighting function計算方式的algorithm如圖一所示：

```
WeightCompute(iteration_num)1{
  for(all net nodes)
    weight_of_net = ds_p_neighbor_num * Cp + ds_n_neighbor_num * Cn;
  for(i = 0; i < iteration_num; i++){
    weight_of_devices = summation of weights of ds_neighbor of this device;
    weight_of_net = summation of weights of ds_neighbor of this net;
  }
}
```

圖一 Weighting Function

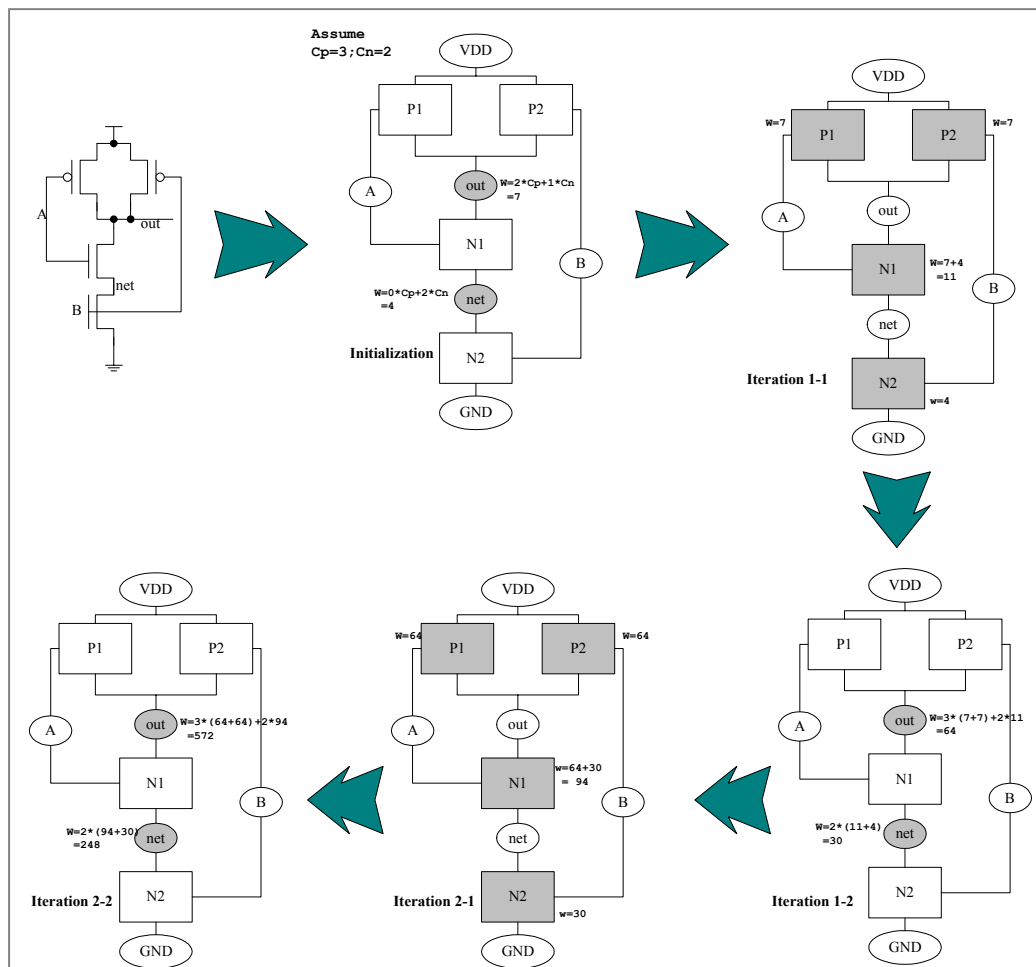
也就是說，我們首先將每一個“net” node之weight設定為其所連接的PMOS的個數乘以 C_p ，加上其所連接的NMOS的個數乘以 C_n 。接著我們計算每一個“device” node的weight，我們將它設定為其所連接的drain以及source的net node的weights的總和(這裡要注意的是，不論是對於net或是device而言，我們都不把一個g_neighbor的weight加進來，這必須配合我們的identification步驟說明，因此，我們將在下一段說明此原因)。之後，可以繼續再計算每一個net node之weight，計算之方式與我們一開始設定其weight之方式相同。

我們可以像這樣反覆計算每一個net以及device node的weights數個iteration，這個iteration的數目可以由user自行設定，但是一般設定大約為3或是4為較佳的狀況，這是因為，我們之所以設定weight，乃是要將每一個node的鄰近之node的資訊蒐集進來，每計算一個iteration的weight，等於從一個node向外一層，將該層的資訊propagate過來這一個node，而一般電路所使用的gates大多不超過4顆MOSFET相串聯，因此將iteration設定為3或4次即可以將一個output node鄰近的information傳遞過來；而如果iteration設得太大，那麼到時候會有太多(target circuit中)的nodes的weight都過大了(與subcircuit中weight最大之node相比)，則candidate set中所含有的node太多，反而又增加identification的時間。下面的實驗結果可以作為說明；以主辦單位所提供之三個測試檔作為例子，以下表格一列出的是，在不同的iteration數目設定之下，最後被取出作為candidate nodes的node個數：

Iteration number	2	3	4	5	6
P8-s1.in vs. p8-c1.in	5	5	5	8	20
P8-s2.in vs. p8-c2.in	35	39	30	40	42
P8-s3.in vs. p8-c3.in	37	38	38	50	72

表格一 Weighting Iteration 與 Candidate number 的關係

為方便解說各個node之weight的計算方法，我們以一個2-input的NAND gate為例，weight計算方法可配合圖二來做瞭解：



圖二 以 NAND gate 為例之 weight 計算方式

(3) Identification(Searching and Checking for Match) algorithm :

此一功能即為利用前兩功能完成後所得到之與電路對應之graph資訊，在target circuit中尋找出subcircuit。在比對的開始，我們必須先找出一個比對的起點，而我們的軟體將subcircuit中具有最大weight值之點，設定為identification步驟之起始點，我們稱之為start_node，這一點通常是subcircuit的primary output；這樣的設定是一個heuristic。

之後我們從主電路(target circuit)中搜尋出一些可能會與此start_node相match的點，而這一個搜尋即利用在各個node的weighting來做篩選。在主電路(target circuit)中，所有 weight_n/weight_p 比 start_node 之 weight_n/weight_p 小的 node，皆不可能與此 start_node 最後得出 match 的結果；換句話說，我們可以在 identification 步驟之開始，只選定那些 weight_n/weight_p 比 start_node 之 weight_n/weight_p 大的 nodes，形成一個”candidate set”，亦即，這些 candidate set 裡的 nodes 最後才可能與 start_node 有 match 的結果，所以 identification 步驟我們只需比對 candidate set 裡頭的 nodes 即可。

整個 identification 步驟之 algorithm 可以由以下之 pseudo code 表示：

```

identify(node_sub, node_target){
  if(node_type equal){
    if(node_type is "net")
      if(node_sub is not a external node)
        if(get_neigh_num == 0) /*pure internal nets*/
          if(ds_p_neigh_num equal and ds_n_neigh_num equal)
            if(identify(node_sub_ds_neigh, node_target_ds_neigh))
              return 1;
            else return 0;
          else return 0;
        else /*the net which is output of a gate*/
          if(ds_neigh_target >= ds_neigh_subcircuit)
            if(identify(node_sub_ds_neigh, node_target_ds_neigh)&&
              identify(node_sub_gate_neigh, node_target_gate_neigh))
              return 1;
            else return 0;
          else return 0;
        else
          if(ds_neigh_num == 0) /*primary output*/
            if(gate_num_target >= gate_num_sub) return 1;
            else return 0;
          else
            if(identify(node_sub_ds_neigh, node_target_ds_neigh)) return 1;
            else return 0;
        else /*for device*/
          if(node_sub and node_target are both flag_vdd or flag_gnd) return 1;
          else
            if(identify(node_sub_ds_neigh, node_target_ds_neigh) &&
              identify(node_sub_gate_neigh, node_target_gate_neigh))
              return 1;
            else return 0;
          };
        else return 0;
      }
    }
  }
}

```

圖 三 Identificaion Algorithm

整個 identify 的過程是採用 recursive 的方式作搜尋比對，從 start_node 出發，然後往與

此start_node相鄰之node一個個作比對，要確認某一個node為比對成功(match)，則必須確認此node之相鄰node皆match，因此每要比對一個node是否match，我們先將subcircuit中此node之所有相鄰node與target circuit中candidate node之所有相鄰node作比對，當然，要比對這些相鄰node是否match，又必須再比對其相鄰的node，如此循環下去。

當然，這樣的recursive function需要一個terminate條件，我們設定這些條件為：

- (1) primary input
- (2) VDD/GND
- (3) 一個node的所有neighbors皆match時

則identify function傳回1，亦即match的意思，在比對過程中，有發現不match的情況，則傳回0，代表subcircuit的此node與target circuit中選定的candidate node比對不成功。例如，當我們選定的subcircuit中之node為NMOS，而target circuit中之candidate node為PMOS，則當然不match，接著我們從target circuit中選定下一個candidate node與subcircuit中的此node再作比對，直到我們發現已經沒有相鄰之node了，代表我們在subcircuit選的start_node與target circuit中所選定之起始比對的node不match，因此我們再繼續選擇下一個(target circuit中的)candidate node作比對。

比對的成功與否，亦即identify傳回值為1或0，取決於一些規則(rule)，例如在前一段中所述，選定的兩個node若為device，而其type(PMOS或是NMOS)不同，則傳回0；這樣的identify函式乃是一個recursive的函式，因此在程式的設計上也不會耗費太多。如圖四所示，我們以一個2-input NAND gate為例，來說明identify函式之比對程序。

此外，由於我們的recursive style之程式在一些特殊情況下可能無法正確處理，因此我們必須再加入一些較為特別的判斷條件，我們歸納如下：

- (1) 電路中有Loop的情況：

我們identify之程序為recursive，亦即，要確認某一個node是否match必須依賴其相鄰(neighbor)之node是否match；然而，若是電路中有loop的情況，因為在recursively traversal的過程中，identify函式的input entry又回到自己，因此會則形成無窮迴圈，導致搜尋辨識的程序永遠跑不完。

因此每當我們traversal過一個node，我們必須在該node設定一個flag，標示說明該node曾被傳入identify函式作為input entry，而稍後如果此node又被傳入函式，則不需再比對此點。但是，這樣做會有一個盲點，那就是：假設我們在

subcircuit中traversal到這一個之前曾被visited過的node，但是，在target circuit中traversal到的這一個對應的node並非先前用以跟subcircuit中的此node做比對確

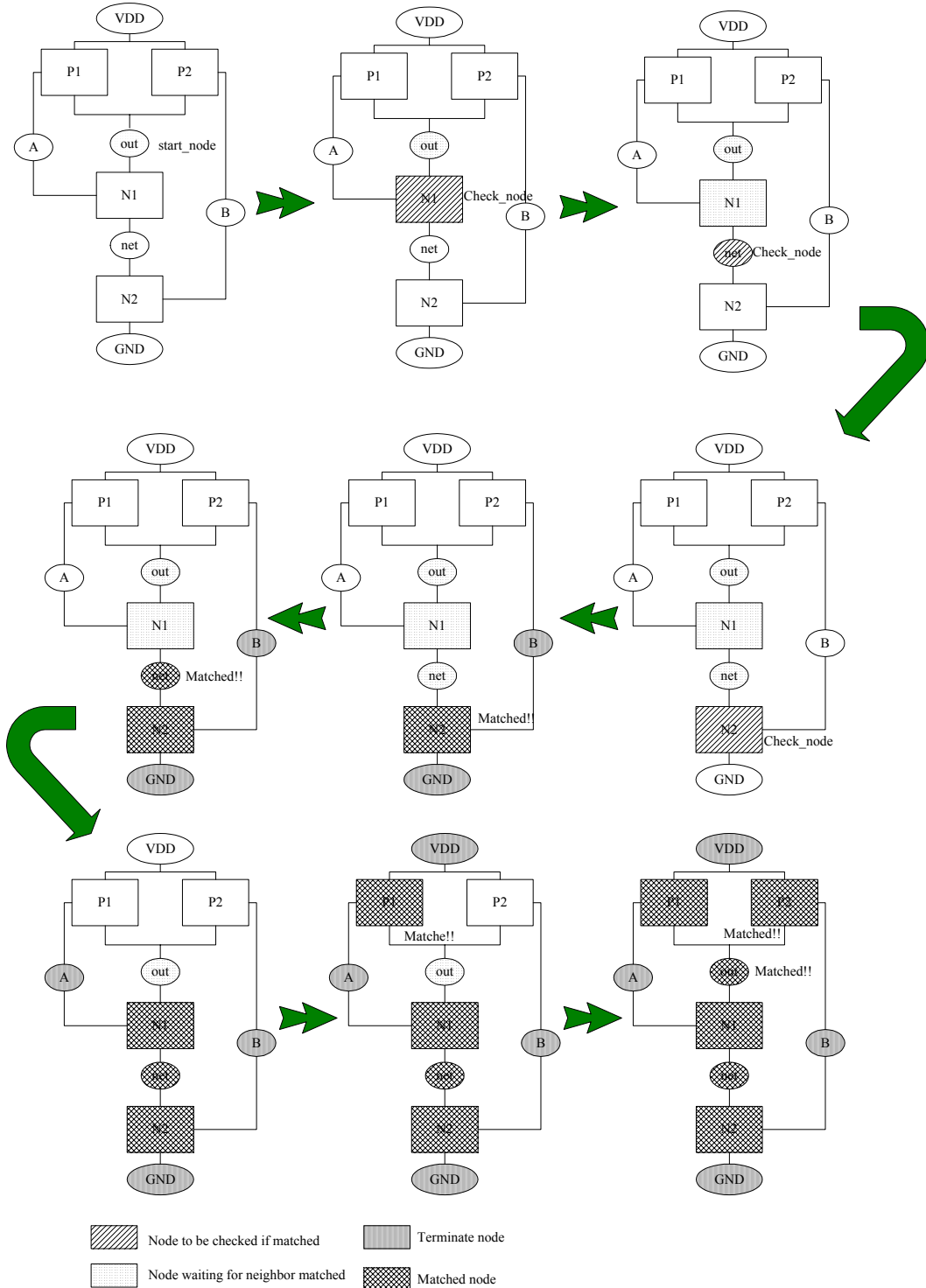


圖 四 2-input NAND gate 之 identify 程式的 traversal 過程

認的node，則函式仍舊應該傳回0，也就是比對失敗的意思。要做到這樣的辨認功能，每一次我們傳入兩個nodes(分別為subcircuit以及target circuit中的node)進入identify函式時，我們分別賦予此兩個node一個整數的tag(此tag乃是用counter計算

出來的值)，而此兩個nodes的tag相同。一旦我們判斷到有loop的情況，也就是當我們traversal subcircuit而發現該node之前被傳入identify函式過，則我們還必須比對這時候此node與target circuit中目前在比對的node是否具有相同的tag，若相同，則函式傳回值為1，亦即比對成功，否則傳回值為0。

(2) 電路中”Multiple-Match”的情況：

我們首先考慮以下的例子：

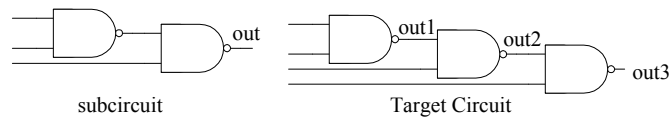


圖 五 Multiple Match Case1

即，若是subcircuit為兩個2-input NAND gate的串聯，而target circuit為三個2-input NAND gate串聯，則照道理，兩個串聯以及後兩個NAND gate串聯的電路應該都被辨認為與subcircuit比對成功，但是若採用以往某些被提出的辦法，例如先對電路作切割的比對方式，則只能夠有一個match成功的結果；但是因為我們所採用的方式是找出candidate set，然後對candidate set中的各點做辨識，而以這個情況來說，”out1”、”out2”以及”out3”三個nodes都將成為candidate nodes(假設subcircuit比對的start node為”out”)，因此都會被用以與subcircuit做比對，而”out2”以及”out3”皆會被比對成功，因此結果將是target circuit中有兩個subcircuit的match。

另一個例子如圖六所示。這是一個比較少見的電路設計，它不是標準的CMOS設計方式，因為並沒有一個NMOS對應到input信號C。假使我們現在用一個2-input NAND gate來與其作identify，則仍舊能夠match出input信號A、B的NAND gate，原因即是，在這裡out信號仍舊會成為candidate node，並且，當我們在subcircuit中traversal過一遍，也就是traversal兩顆NMOS以及兩顆PMOS之後，subcircuit即已比對結束，而target circuit中也有對應的每一顆MOS存在，故仍舊能夠輸出正確的比對結果。

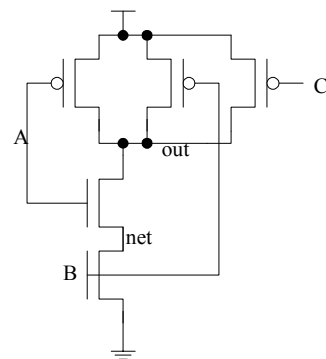


圖 六 Multiple Match Case2

四、實作 (Implementation)

4.1 資料結構 (Data Structures)

我們所採用的資料結構有兩個，分別敘述如下：

(1)CKT_NODE：

單一的結構用來表示電路中所有各種不同形態的節點，存放在一個連續的陣列之中，並且給予每個節點相對應且唯一的serial number。這樣做的優點在於，接下來的比對工作，可以直接由serial number作為memory address來存取該節點的資訊。除此之外並且將previous work中所收集到的資訊，如type、label name、neighbor information以及由 weighting function 中所計算出來的weighting值，存進節點中。CKT_NODE 的定義如下：

```
struct CKT_NODE{
    int sn; //serial number
    int type; //0 for NMOS, 1 for PMOS, 2 for NET, 3 for VDD/GND
    int w_p_num; //weight contributed by neighbor PMOS
    int w_n_num; //weigh contributed by neighbor NMOS
    struct NEIGHBOR_NODE *g_nei; //neighbors connected to Gate
    struct NEIGHBOR_NODE *ds_nei; //neighbors connected to Drain/Source
    char label[LENGTH]; //label name
}
```

(2)NEIGHBOR_NODE：

用來記錄一個CKT_NODE其相鄰的neighbors，由於一個CKT_NODE的neighbor數不固定但往往不甚多，因此將NEIGHBOR_NODE用linked list來表示，避免使用不必要的記憶體。NEIGHBOR_NODE的定義如下：

```
struct NEIGHBOR_NODE{
    int sn; //the serial number of neighbor
    struct NEIGHBOR_NODE *next; //indicate next neighbor
}
```

由於所有的節點數已經在previous work中求出，因此使用以上的資料結構不僅可以最少的記憶體空間來記錄全部的節點，同時可以用陣列的方式來存取所需的節

點的資訊，避免一般用adjacent matrix時所遇到linked list存取不便的問題，對於接下來的比對工作，可減少搜尋的額外負擔，加快比對的速度。

4.2 程式流程 (Flow)

此程式之流程大致如圖七之流程圖所示：

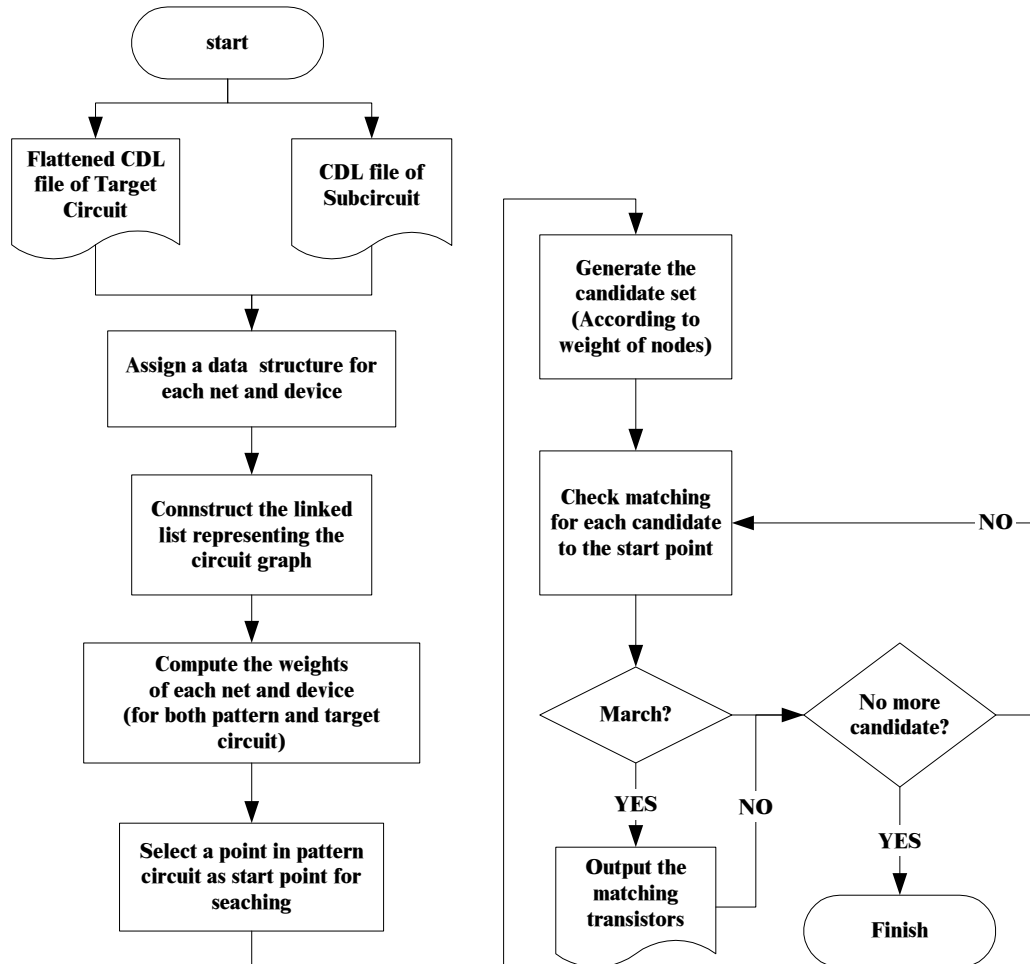


圖 七 程式流程圖

配合圖七之流程圖，茲將程式運作流程再以文字描述如下：

(1) 讀入主電路(target circuit)以及副電路(subckt)之CDL格式檔案，計算二者中各包含多少node(包括device以及net的總數)個數，宣告與此數相同的資料結構用以存放這些node之資訊。

(2) 分別建構出與target circuit以及subcircuit對應之graph, 分別記錄兩個graph中各個node的資訊，包含與其相連接之node個數、名稱等。(可參照”4.1 資料結構”中之描述)。

(3) 以weighting function計算各個node之weights，並且從subcircuit之graph中尋找一點weight為最大的node作為identification的起點(稱之為start_node)。

(4) 搜尋target circuit之graph中所有weight比start_node之weight大的node，蒐集起來成為一個”candidate set”，亦即只有這些nodes最後才可能與start_node比對成功。

(5) 從candidate set中找出每一點與start_node開始做比對(比對之步驟可以參照”三 演算法之 (3) identification algorithm”)

(6) 假設match，則列印出所有比對成功之node。

(7) 如果比對不成功，則看candidate set中是否還有node還沒被比對過，若有，則重複(5)–(6)，否則就結束程式。

五、 實驗結果 (Experimental Results)

5.1 工作平台及程式語言(Platform and Programming Language)

此設計乃是以C語言，在Sun工作站上編譯之程式，我們並且附上程式之make file，可以在數種作業系統上編譯並且執行此程式。

5.2 測試檔輸出 (Test Output)

主辦單位所提供之三個測試檔，經過此軟體之辨識擷取後後，所得之測試檔輸出列表如下：

Subcircuit	p8-s1.in
Target Circuit	p8-c1.in
輸出結果	<pre> nand2 : instance0 M1 20 17 19 1 N M2 1 7 20 1 N M3 2 17 19 2 P M4 19 7 2 2 P nand2 : instance1 M7 22 5 21 1 N M8 1 6 22 1 N M9 2 5 21 2 P M10 21 6 2 2 P nand2 : instance2 M43 32 7 31 1 N M44 1 16 32 1 N M45 2 7 31 2 P M46 31 16 2 2 P nand2 : instance3 </pre>

	M49	34	10	33	1	N
	M50	1	12	34	1	N
	M51	2	10	33	2	P
	M52	33	12	2	2	P
	nand2 : instance4					
	M55	36	5	35	1	N
	M56	1	6	36	1	N
	M57	2	5	35	2	P
	M58	35	6	2	2	P

Subcircuit	P8-s2.in					
Target Circuit	p8-c2.in					
輸出結果	oai : instance0					
	M21	GND	22	43	GND	N
	M22	GND	7	43	GND	N
	M23	43	42	40	GND	N
	M24	40	42	VDD	VDD	P
	M25	40	7	44	VDD	P
	M26	44	22	VDD	VDD	P
	oai : instance1					
	M59	GND	25	58	GND	N
	M60	GND	6	58	GND	N
	M61	58	57	55	GND	N
	M62	55	57	VDD	VDD	P
	M63	55	6	59	VDD	P
	M64	59	25	VDD	VDD	P
	oai : instance2					
	M97	GND	17	73	GND	N
	M98	GND	5	73	GND	N
	M99	73	72	70	GND	N
	M100	70	72	VDD	VDD	P
	M101	70	5	74	VDD	P
	M102	74	17	VDD	VDD	P

Subcircuit	p8-s3.in					
Target Circuit	p8-c3.in					
輸出結果	fd2 : instance0					
	M14	GND	46	57	GND	N
	M15	GND	45	58	GND	N
	M16	GND	46	59	GND	N
	M17	61	60	62	GND	N
	M18	59	24	61	GND	N
	M19	63	62	49	GND	N
	M20	57	64	65	GND	N
	M21	GND	62	66	GND	N
	M22	66	67	60	GND	N
	M23	GND	29	63	GND	N
	M24	65	49	29	GND	N
	M25	68	62	64	GND	N

M26	GND	67	69	GND	N
M27	69	24	68	GND	N
M28	58	46	70	GND	N
M29	70	64	67	GND	N
M30	VDD	46	67	VDD	P
M31	VDD	46	29	VDD	P
M32	VDD	46	62	VDD	P
M33	VDD	24	62	VDD	P
M34	VDD	60	62	VDD	P
M35	VDD	49	29	VDD	P
M36	VDD	29	49	VDD	P
M37	VDD	67	60	VDD	P
M38	VDD	62	60	VDD	P
M39	VDD	62	49	VDD	P
M40	VDD	64	29	VDD	P
M41	VDD	24	64	VDD	P
M42	VDD	62	64	VDD	P
M43	VDD	67	64	VDD	P
M44	VDD	64	67	VDD	P
M45	VDD	45	67	VDD	P
fd2 : instance1					
M46	GND	46	71	GND	N
M47	GND	45	72	GND	N
M48	GND	46	73	GND	N
M49	75	74	76	GND	N
M50	73	25	75	GND	N
M51	77	76	49	GND	N
M52	71	78	79	GND	N
M53	GND	76	80	GND	N
M54	80	81	74	GND	N
M55	GND	30	77	GND	N
M56	79	49	30	GND	N
M57	82	76	78	GND	N
M58	GND	81	83	GND	N
M59	83	25	82	GND	N
M60	72	46	84	GND	N
M61	84	78	81	GND	N
M62	VDD	46	81	VDD	P
M63	VDD	46	30	VDD	P
M64	VDD	46	76	VDD	P
M65	VDD	25	76	VDD	P
M66	VDD	74	76	VDD	P
M67	VDD	49	30	VDD	P
M68	VDD	30	49	VDD	P
M69	VDD	81	74	VDD	P
M70	VDD	76	74	VDD	P
M71	VDD	76	49	VDD	P
M72	VDD	78	30	VDD	P
M73	VDD	25	78	VDD	P
M74	VDD	76	78	VDD	P
M75	VDD	81	78	VDD	P
M76	VDD	78	81	VDD	P
M77	VDD	45	81	VDD	P
fd2 : instance2					
M105	127	126	128	GND	N
M106	GND	86	129	GND	N

M107	GND	91	130	GND	N
M108	132	131	133	GND	N
M109	130	122	132	GND	N
M110	134	133	121	GND	N
M111	GND	91	127	GND	N
M112	GND	133	135	GND	N
M113	135	136	131	GND	N
M114	GND	89	134	GND	N
M115	128	121	89	GND	N
M116	137	133	126	GND	N
M117	GND	136	138	GND	N
M118	138	122	137	GND	N
M119	129	91	139	GND	N
M120	139	126	136	GND	N
M121	VDD	91	136	VDD	P
M122	VDD	91	89	VDD	P
M123	VDD	91	133	VDD	P
M124	VDD	122	133	VDD	P
M125	VDD	131	133	VDD	P
M126	VDD	121	89	VDD	P
M127	VDD	89	121	VDD	P
M128	VDD	136	131	VDD	P
M129	VDD	133	131	VDD	P
M130	VDD	133	121	VDD	P
M131	VDD	126	89	VDD	P
M132	VDD	122	126	VDD	P
M133	VDD	133	126	VDD	P
M134	VDD	136	126	VDD	P
M135	VDD	126	136	VDD	P
M136	VDD	86	136	VDD	P
fd2 : instance3					
M151	148	147	149	GND	N
M152	GND	86	150	GND	N
M153	GND	91	151	GND	N
M154	153	152	154	GND	N
M155	151	143	153	GND	N
M156	155	154	142	GND	N
M157	GND	91	148	GND	N
M158	GND	154	156	GND	N
M159	156	157	152	GND	N
M160	GND	93	155	GND	N
M161	149	142	93	GND	N
M162	158	154	147	GND	N
M163	GND	157	159	GND	N
M164	159	143	158	GND	N
M165	150	91	160	GND	N
M166	160	147	157	GND	N
M167	VDD	91	157	VDD	P
M168	VDD	91	93	VDD	P
M169	VDD	91	154	VDD	P
M170	VDD	143	154	VDD	P
M171	VDD	152	154	VDD	P
M172	VDD	142	93	VDD	P
M173	VDD	93	142	VDD	P
M174	VDD	157	152	VDD	P
M175	VDD	154	152	VDD	P
M176	VDD	154	142	VDD	P

M177	VDD	147	93	VDD	P
M178	VDD	143	147	VDD	P
M179	VDD	154	147	VDD	P
M180	VDD	157	147	VDD	P
M181	VDD	147	157	VDD	P
M182	VDD	86	157	VDD	P
fd2 : instance4					
M229	GND	165	185	GND	N
M230	GND	167	186	GND	N
M231	GND	165	187	GND	N
M232	189	188	190	GND	N
M233	187	166	189	GND	N
M234	191	190	90	GND	N
M235	185	192	193	GND	N
M236	GND	190	194	GND	N
M237	194	195	188	GND	N
M238	GND	163	191	GND	N
M239	193	90	163	GND	N
M240	196	190	192	GND	N
M241	GND	195	197	GND	N
M242	197	166	196	GND	N
M243	186	165	198	GND	N
M244	198	192	195	GND	N
M245	VDD	165	195	VDD	P
M246	VDD	165	163	VDD	P
M247	VDD	165	190	VDD	P
M248	VDD	166	190	VDD	P
M249	VDD	188	190	VDD	P
M250	VDD	90	163	VDD	P
M251	VDD	163	90	VDD	P
M252	VDD	195	188	VDD	P
M253	VDD	190	188	VDD	P
M254	VDD	190	90	VDD	P
M255	VDD	192	163	VDD	P
M256	VDD	166	192	VDD	P
M257	VDD	190	192	VDD	P
M258	VDD	195	192	VDD	P
M259	VDD	192	195	VDD	P
M260	VDD	167	195	VDD	P
fd2 : instance5					
M197	172	171	173	GND	N
M198	GND	87	174	GND	N
M199	GND	91	175	GND	N
M200	177	176	178	GND	N
M201	175	164	177	GND	N
M202	179	178	163	GND	N
M203	GND	91	172	GND	N
M204	GND	178	180	GND	N
M205	180	181	176	GND	N
M206	GND	96	179	GND	N
M207	173	163	96	GND	N
M208	182	178	171	GND	N
M209	GND	181	183	GND	N
M210	183	164	182	GND	N
M211	174	91	184	GND	N
M212	184	171	181	GND	N

M213	VDD	91	181	VDD	P
M214	VDD	91	96	VDD	P
M215	VDD	91	178	VDD	P
M216	VDD	164	178	VDD	P
M217	VDD	176	178	VDD	P
M218	VDD	163	96	VDD	P
M219	VDD	96	163	VDD	P
M220	VDD	181	176	VDD	P
M221	VDD	178	176	VDD	P
M222	VDD	178	163	VDD	P
M223	VDD	171	96	VDD	P
M224	VDD	164	171	VDD	P
M225	VDD	178	171	VDD	P
M226	VDD	181	171	VDD	P
M227	VDD	171	181	VDD	P
M228	VDD	87	181	VDD	P
fd2 : instance6					
M315	GND	204	224	GND	N
M316	GND	206	225	GND	N
M317	GND	204	226	GND	N
M318	228	227	229	GND	N
M319	226	205	228	GND	N
M320	230	229	94	GND	N
M321	224	231	232	GND	N
M322	GND	229	233	GND	N
M323	233	234	227	GND	N
M324	GND	202	230	GND	N
M325	232	94	202	GND	N
M326	235	229	231	GND	N
M327	GND	234	236	GND	N
M328	236	205	235	GND	N
M329	225	204	237	GND	N
M330	237	231	234	GND	N
M331	VDD	204	234	VDD	P
M332	VDD	204	202	VDD	P
M333	VDD	204	229	VDD	P
M334	VDD	205	229	VDD	P
M335	VDD	227	229	VDD	P
M336	VDD	94	202	VDD	P
M337	VDD	202	94	VDD	P
M338	VDD	234	227	VDD	P
M339	VDD	229	227	VDD	P
M340	VDD	229	94	VDD	P
M341	VDD	231	202	VDD	P
M342	VDD	205	231	VDD	P
M343	VDD	229	231	VDD	P
M344	VDD	234	231	VDD	P
M345	VDD	231	234	VDD	P
M346	VDD	206	234	VDD	P
fd2 : instance7					
M283	211	210	212	GND	N
M284	GND	86	213	GND	N
M285	GND	91	214	GND	N
M286	216	215	217	GND	N
M287	214	203	216	GND	N
M288	218	217	202	GND	N

M289	GND	91	211	GND	N
M290	GND	217	219	GND	N
M291	219	220	215	GND	N
M292	GND	99	218	GND	N
M293	212	202	99	GND	N
M294	221	217	210	GND	N
M295	GND	220	222	GND	N
M296	222	203	221	GND	N
M297	213	91	223	GND	N
M298	223	210	220	GND	N
M299	VDD	91	220	VDD	P
M300	VDD	91	99	VDD	P
M301	VDD	91	217	VDD	P
M302	VDD	203	217	VDD	P
M303	VDD	215	217	VDD	P
M304	VDD	202	99	VDD	P
M305	VDD	99	202	VDD	P
M306	VDD	220	215	VDD	P
M307	VDD	217	215	VDD	P
M308	VDD	217	202	VDD	P
M309	VDD	210	99	VDD	P
M310	VDD	203	210	VDD	P
M311	VDD	217	210	VDD	P
M312	VDD	220	210	VDD	P
M313	VDD	210	220	VDD	P
M314	VDD	86	220	VDD	P

5.3 時間及記憶體使用量 (CPU Time and Memory Usage)

由於我們所使用的演算法為遞迴式函式，由所選定的 start node 開始，會一直不斷呼叫自己，直到遇到所設定的終止條件 (VDD、GND 或是 Primary Input) 為止。假設 pattern circuit 中 start node 到 VDD、GND 或是 Primary Input 為止的平均層數為 L，且在不失一般性的前提下，令每個節點有 3 個 neighbour，則我們所提出的 identify function 的總共所需時間為：

$$T(L) = 3 \times T(L-1) = 3 \times 3 \times T(L-2) = 3 \times 3 \times \dots \times 3 \times T(0) \\ = 3^L \times T(0)$$

(T(0)：比對 VDD/GND/Primary Input 所需的時間。)

整個 identification algorithm 的 Big-O 表示式為：O(3^L)，但實際上由於比對的過程中，一旦有一個 neighbor 比對不成功，即使剩下的 neighbor 全部都比對成功，該節點也不會 match，因此只要有一個 neighbor 比對失敗，程式即可終止，不必繼續剩下的比對工作。以這種 branch and bound 的方式可以大幅縮短程式執行的時間，一個 candidate

往往只需一兩個步驟就可以結束。故在速度上的表現較以往一般 graph traversal 的比對方式更佳。

在 Graph Construction 中，每個節點都必須用 CKT_NODE 這個結構體來儲存，除此之外，還必須以 NEIGHBOR_NODE 這個結構體所形成的鏈結串列來表示每個節點的neighbor。而在比對時，為了避免無窮迴圈的狀況發生，因此每個節點得有一個flag_visited來記錄該節點是不是已經被比對過，除此之外，為了解決loop的問題，還必須有一個額外的tag。上述的各結構及變數的大小如下：

```
struct CKT_NODE = 80 bytes;
struct NEIGHBOUR_NODE = 8 bytes;
int flag_matched = 4 bytes;
int flag_tag = 4 bytes;
```

假設pattern circuit中總共的節點數為 N_p ，target中總共的節點數為 N_t ，則整個比對的過程中所需的記憶體空間如下所示：

$$\begin{aligned} \text{Memory Usage} &= (N_p + N_t) \times (80 + 8 \times 3 + 4 + 4) \text{ bytes} \\ &= (N_p + N_t) \times 112 \text{ bytes} \end{aligned}$$

以主辦單位所附之各個測試檔案為例，所需的記憶體大小如下表所示：

檔案名稱	全部節點數(含Device node與Net node)	所需記憶體空間(kb)
P8-s1.in	10	1.093kb
P8-s2.in	14	1.531kb
P8-s3.in	53	5.799kb
P8-c1.in	95	10.391kb
P8-c2.in	288	31.5kb
P8-c3.in	829	90.672kb

由上表可知，所需記憶體空間會隨著電路大小呈線性正相關。

5.4 結果分析 (Data Analysis)

我們將各個測試檔以及我們自行採用之實際電路測試檔案之實驗結果分析歸納出幾項重要指標，如以下表格所列：

檔案名稱	樣本電路 節點數	目標電路 節點數	建構Graph 所需時間 (ms)	電路比對 所需時間 (ms)	執行時間 (ms)
P8-s1.in P8-c1.in	10	95	20ms	10ms	35ms
P8-s2.in P8-c2.in	14	288	35ms	10ms	45ms
P8-s3.in P8-c3.in	95	829	120ms	10ms	130ms
nand5.cdl decoder.cdl	22	10265	16770ms	950ms	17720ms

(註1：所使用的機器為SUN UltraSPARC-IIi(440Mhz), 記憶體大小為1GB)

(註2：上述資料為 gcc 加上 -pg 選項後，由 gprof 程式產生)

有上表可知，由於所提出的演算法具有branch and bound的特性，再加上適當 weighting function的輔助，電路比對時間僅與 candidate 的個數及樣本電路的大小有關。與所參考的文件^{[1],[2],[5]}相較，在電路比對時間上有更佳表現，

Reference

- [1] Miles Ohlrich et al., "SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm", 30th ACM/IEEE Design Automation Conference, pp. 31-37, June 1993.
- [2] N. Vijaykrishnan et al., "SUBGEN: A Genetic Approach for Subcircuit Extraction", 9th International Conference on VLSI Design, p343-345, Jan 1996
- [3] Georg Pelz et al., "Circuit Comparison by Hierarchical Pattern Matching", Proc. Conference of Computer Aided Design, pages 290-293, 1991
- [4] Scott W. Hadley et al., "An Efficient Eigenvector Approach of Finding Netlist Partitions", IEEE Trans. Computer-Aided Design, vol.11, NO. 7, July 1992.
- [5] Zong Ling et al, "An Efficient SubCircuit Extraction Algorithm by Resource Management"
- [6] Georg Pelz, "Pattern Matching and Refinement Hybrid Approach to Circuit Comparison", IEEE Transaction on Computer-Aided Design of Intergrated Circuit and System, vol 13, NO. 7, February 1994.

附錄: 使用手冊(User's Manual)

A. 如何編譯程式(How to compile)

(1)*gzip -d A19.tar.gz*

(2)*tar xvf A19.tar*

前兩個步驟可將壓縮檔解開。

(3)*cd A19*

(4)*make decide*

進入A19的目錄下，下 *make decide* 指令後即可編譯出執行檔。

B. 如何執行程式(How to run)

(1)*decide pattern_circuit target_circuit output_file*

i.e : *decide p8-s1.in p8-c1.in slc1_result*

*slc1_result*一個純文字檔案，即為所產生的結果。用一般的文字編輯器即可觀看結果。