即時嵌入式系統之軟體合成工具設計*
Design of a Software Synthesis Tool for Real-Time Embedded Systems*

李宗演[1] 吳宜穆[2] 熊博安[3] 張劍平[2] 蔡加春[1] 李文達[1]
Trong-Yen Lee[1] I-Mu Wu[2] Pao-Ann Hsiung[3] Chien-Pin Chang[2] Chia-Chun Tsai[1] Wen-Ta Lee[1]
[1]台北科技大學電子工程系、[2]國防大學中正理工學院電機系、[3]中正大學資訊工程學系
[1]Department of Electronic Engineering, National Taipei University of Technology
[2]Department of Electrical Engineering, Chung Cheng Institute of Technology, National Defense University
[3]Department of Computer Science and Information Engineering, National Chung Cheng University
[1]Email: tylee@ntut.edu.tw, [2]Email: cpchang@ccit.edu.tw, [3]Email: pahsiung@cs.ccu.edu.tw

摘 要

本文在於設計一套即時嵌入式系統之軟體合成工具，其中包含提供一個圖形化的使用者介面，以減少軟體之開發的時間，為了能夠處理硬體中斷之事件，我們提出以中斷式時間派翠網路(*Interrupt Time Petri Nets*, ITPN)做為系統模型之描述，可描述出系統中斷處理的動作與每一個任務的即時限制，並提出具中斷處理能力的類似動態排程(*Interrupt-Based Quasi-Dynamic Scheduling*, IQDS)演算法，將中斷動作與即時的行程作有效的排程，當找出有效排程之後利用產生程式碼(*Code Generation*)演算法來產生包含中斷相關指令的單晶片 C 語言程式碼。本工具提供一個視窗化的設計介面，可以直覺的描繪系統模型與便利的參數設定環境。最後我們使用電腦溫度量測器與四相步進馬達控制器為例子來驗證所發展之工具的正確性與方便性。

一、簡介

日常生活中食、衣、住、行、育、樂各方面所看到的、用到的東西，如飛機、電磁爐、數位像機、投影機、血壓計和遊戲機等都為嵌入式系統[1]，在設計方面常以軟硬體共同設計(Hardware/Software Co-design)的方式，藉由軟體和硬體的合成工具將軟體和硬體合成出來，最後再將軟體與硬體結合模擬做最後的整合與調整。

嵌入式軟體合成技術，大致上分成三個步驟，第一步驟為：利用系統模型(System Model)來描述軟體的結構；第二步驟為：根據系統模型的描述來對即時任務作排程(Schedule)，進而獲得有效的系統排程，此系統排程稱之為可排程(Schedulable)，最後一個步驟為：將可排程的即時任務根據系統模型所描述[1]的軟體結構自動化的產生相對應程式碼，以供嵌入式系統使用。

在文獻[2][3][4]中，所使用的系統模型中皆缺少對中斷的表示與描述，因此在系統模型中應該加入對中斷指令的表示與描述，如此一來合成出的程式碼更可直接使用於微控制器中。由於以往嵌入式軟體合成工具皆以文字編輯方式來描述系統模型，故使用者的親和性(User-Friendly)較差[5]。

本文在系統模型方面，將提出中斷式時間派翠網路作為系統模型，並發展出以具中斷處理能力的類似動態排程演算法根據系統模型找出有效的排程，然後利用產生程式碼演算法產生包括中斷指令的程式碼，最後根據這些方法設計出嵌入式軟體合成工具，且設計一個高親和度的圖形之使用者編輯系統的介面。

本文於第一節中簡介嵌入式系統合成工具的發展。第二節中介紹了嵌入式軟體合成工具的相關研究。第三節說明軟體合成工具之設計方法，包含了系統模型、系統的設計的理論及設計的細節。第四節為應用實例驗證，舉出電腦數位溫度量測器與四相步進馬達控制器的兩個實例來說明本工具的實用性。最後提出本研究的結論與未來研究之方向。

二、相關研究

主要嵌入式軟體合成是藉由一個可行的排程方法，將行程依序排列，以滿足設計者所提出的功能要求、即時的限制與記憶體容量的限制。嵌入式軟體結構通常包括資料計算(Data Computations)；和控制結構(Control Structures)，其中控制結構又包含了資料相依控制(Data-Dependent Controls)，如：if-then-else 或 while-do-loops，必須依測試值或一些資料來決定下一個運算稱之；和即時控制(Real-Time Controls)，如優先權、中斷與觸發行為等，這些動作是發生於內部或外部的事件[3]。對於排程而言，只包含資料計算的排程，稱為靜態排程(Static Schedule)，排程中增加了資料相依控制的排程，稱為類似靜態排程(Quasi-Static Schedule)，若包含即時控制的排程，就稱為動態排程(Dynamic Schedule)。

*本研究接受國科會編號：NSC-91-2215-E-002-041 研究計畫之部分補助

在常見的圖形表示法中派翠網路是以直觀的圖形方式描繪系統,更以簡單的數學定義和運算作為分析系統的方法,更減少學習上的困難度[6][7][8],所以在先前的研究當中[2][3][4][5][9],都將派翠網路作為系統模型,並且根據實際需要加以變形。

目前的研究是將派翠網路的模型簡化或是變形,在[4]中作者使用派翠網路的子類別為 FCPN,作為系統模型,並提出類似靜態排程演算法。FCPN 可描繪出軟體架構,並可藉由類似靜態的排程演算法,合成出程式碼,但嵌入式軟體大部分都為即時軟體,且為循環執行的程式,此模型無法對即時行程作排班,所以在[2]中,作者提出以 FCPN 延伸的 TFCPN 作為系統模型,最大的差別於系統模型的轉移上加入最早的激發時間與最晚激發時間,其用意是限制轉移在最晚激發時間前,一定要將權杖轉到下一個位置,以便作循環執行週期的預估。

在[3]中作者提出 TCCPN,此模型移除了先前的限制,結合了[2]中所提出的即時的限制,並利用 TEQSS 的演算法,找出可排程的結果。在產生程式碼方面,在[3]中作者提出以多執行緒的嵌入式軟體取代單執行緒的嵌入式軟體。

以上所介紹的系統模型與排程的演算法,發現所產生的程式碼與 8051 規格的 C 語言的程式碼有些差別,因為 8051 的程式會利用中斷服務程式處理一些動作,所以本文將派翠網路中加入對中斷的描述,提出以中斷式時間派翠網路作為系統的模型,並提出具中斷處理能力的類似動態排程演算法,最後針對中斷描述產生相對應的程式碼。先前的嵌入式軟體合成工具之使用者介面是利用文字編輯方式描繪出所要的系統模型[5],所以本文將針對系統模型,設計一個圖形化的使用者介面。

三、嵌入式軟體合成工具之設計

此節將介紹嵌入式軟體合成工具所使用的系統模型、系統設計,其中系統設計包含了使用者介面設計、排程的方法與程式碼產生的方法。

(一)、系統模型

由於嵌入式系統中需使用即時的中斷服務程式來完成一些動作,所以針對此特性本文中提出中斷式時間派翠網路(*Interrupt Time Petri Nets*, ITPN),如圖 1 所示,其中包含了轉移激發的即時限制,與先前研究最大的差別在於加入了一個中斷處理的表示,相關定義如下:

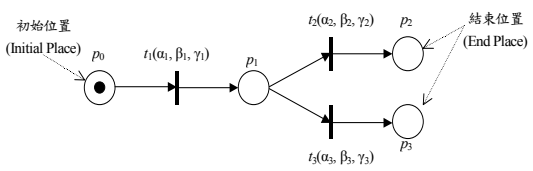

圖 1 中斷式時間派翠網路

(1) 定義一:一中斷式時間派翠網路(*Interrupt Time Petri Nets*, ITPN)之結構由 5 個部分組成的 $ITPN=(P,T,I,O,\Omega)$,其中:
- $P=\{p_1,p_2,\cdots,p_m\}$,為所有位置所構成的有限集合。
- $T=\{t_1,t_2,\cdots,t_n\}$,為所有轉移所構成的有限集合,其中 $P\cup T\neq\phi$ 和 $P\cap T=\phi$。
- $I:T\rightarrow P^{\infty}$,為輸入函數,由一個轉移 t_i 映射至其一群輸入位置的函數,表示為 $I(t_i)$。
- $O:T\rightarrow P^{\infty}$,為輸出函數,由一個轉移 t_j 映射至其一群輸出位置的函數,表示為 $O(t_i)$。
- $\Omega:\Omega(t)=(\alpha,\beta,\gamma)$,這裡的 $t\in T$,α 為最早的激發時間(Earliest Firing Time, EFT);β 最晚的激發時間(Latest Firing Time, LFT);γ 則代表致能(Enable)那一類中斷,以 8051 為例包含了五種中斷源,總共有三十二種組合,故 $\gamma=\{0,1,2,\cdots,31\}$。

(2) 定義二:狀態值(Marking) μ 在派翠網路 $ITPN=(P,T,I,O,\tau,E)$ 中,為位置所構成有限集合的函數,$\mu=(\mu_0,\mu_1,\cdots,\mu_n)$,此 n=|P|且每一個 $\mu_i\in N, i=1,\cdots,n$,也就是說 μ_i 是於穩態時 p_i 上的權杖數目,通常標註表示資源的數目,而 μ_0 為系統的初始狀態。一個加入狀態值的派翠網路表示為 $M=(ITPN,\mu)$。

(3) 定義三:選擇區塊(Choice Block),是由一個 Free Choice 或是一個 Complex Choice 所構成的,包含了一組輸入轉移,如圖 2 中的 t_1、t_4 與 t_5 與一組輸出轉移,如圖 2 中的 t_2、t_3、t_6、t_7 與 t_8。若一個輸出轉移來自不同選擇位置(Choice Place)的方向弧線,此轉移需同時滿足兩種或兩種以上的條件才可被激發,如圖 2 中當 p_2 選擇 no 與 p_3 選擇 yes 時,t_7 才會被激發。

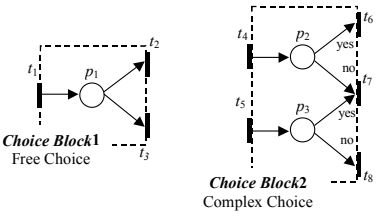

圖 2 選擇區塊(Choice Block, CB)

(4) 定義四:利用 ITPN 的模型只能描繪出程式結構中的循序結構(Sequence Structure)與選擇結構(Selection Structure),無法描繪出反覆結構(Repetition)。

(二)、系統設計

本文中所發展的嵌入式軟體合成工具主要包含了使用者介面設計、排程方法與程式碼產生之方法。為了符合嵌入式軟體的中斷行為與即時的限制，故本文提出以中斷式時間派翠網路作為系統的模型。為了要改善先前嵌入式軟體合成工具以文字編輯系統模型的親和度較差的缺點[5]，因此本文建立一個圖形編輯系統模型的介面，讓使用者可以使用圖形方式建立系統模型。接著提出具中斷處理能力的類似動態排程演算法做系統排程及產生程式碼演算法產生 8051 規格的 C 語言程式，結合圖形化的使用者介面，構成嵌入式軟體合成工具。

(1) 使用者介面設計

為了讓使用者方便且快速的描繪出所要的嵌入式軟體，我們建立了一個圖形編輯的使用者介面，其介面包含了三個視窗如圖 3 示：

- 基本元件編輯視窗：視窗最右邊之工具列中的按鈕其功能依序由上而下為在此視窗產生轉移、位置、方向弧線、刪除元件、復原動作，並可對產生的位置上方輸入名稱，且於轉移上方也可輸入最早的激發時間與最晚的激發時間、允許中斷種類與名稱更可在方向弧線上方寫入權值重(Weight)。
- 檢視排程結果視窗：設計者編輯完後，按下第六個按鈕後，將排程出所有可能排程序列，並可利用此視窗內的時間軸看出是否超出設計者要求程式的執行週期時間。
- 產生程式碼之視窗：排程完後，當每一個排程序列都滿足程式執行的週期時間時，按下第七個按鈕後，將會在產生程式碼視窗中產生設計者所要的程式。

在此編輯工具中也提供讓使用者編輯副程式與中斷服務程式，為了減少了對硬體低階的設定與編輯副程式的時間，故在工具中內建一些常用的副程式供設計者直接使用。圖形編輯的使用者介面也提供讓使用者定義每一個轉移所代表的動作，此動作可能是副程式、時間延遲程式、計算過程與輸入、輸出程式。在欲定義的轉移上方連續點兩下即可。

(2) 排程之方法

本文提出以具中斷處理能力的類似動態排程演算法如表 1 所示，針對 ITPN 找出有效排程。IQDS 演算法是將 ITPN 分成靜態排程與選擇區塊兩部分，也就是程式結構中的選擇結構的部分，如圖 4 示。

表 1 具中斷處理能力的類似動態排程演算法

IQDS_Scheduling (P, T, I, O, Ω) {
int *CountRoute*=1; //*CountRoute* : the number of element in *Rroute*[ ]
bool *Schedulable* = true, *Continue* = false, *Stop* = false;
char *X*, *Y*, *Z*, *Result*;
string *Route*[ ], *StaticRoute*;
Search Initial Place, End Place, Choice Place (1)
Build Extable(*i*) for CB(*i*); //$1 \leq i \leq$ m; where m is the number of choice block; CB:Choice Block (2)
Search CBS_i for *CB*(*i*); //CBS_i : Choice Block Set for CB(*i*) (3)
X = **Static_Scheduling** (t_j); (4)
//t_j is a transition which is after the Initial Place, $t_j \in T$, $1 \leq j \leq n$, *n* is the number of transition
if (*X* = 0) { (5)
Route[*CountRoute*] = *StaticRoute*;} //Only one route (6)
else {
do{
CountRoute = *CountRoute* + $CountCBS_X - 1$; (7)
// $CountCBS_X$: the number of CBS_X for CB(*X*)
Extend (*Route*[]); //Extend route (8)
For (*k* = 1 ; *k* <= *CountRoute* ; *k*++) { (9)
if (*Route*[*k*] has not end yet) { (10)
Y = the end of transition in *Route*[*k*]; (11)
Z = **Static_Scheduling** (*Y*); (12)
if (*Z* = 0) { (13)
stract(*Route*[*k*], *StaticRoute*); (14)
//Combine *Route*[*k*] and StaticRoute
End of *Route*[*k*]; (15)
if (*k* = = *CountRoute*) { (16)
Continue = false;} (17)
else{
X=*Z*; (18)
Continue = true; (19)
Break;}}}} //End of the for (20)
}while(*Continue* = true)}
Real_Time_Check (*Route*[*l*]); //$1 \leq l \leq$ *CountRoute* (21)
if (*Schedulable* = true){
Code_Generation ();}} (22)

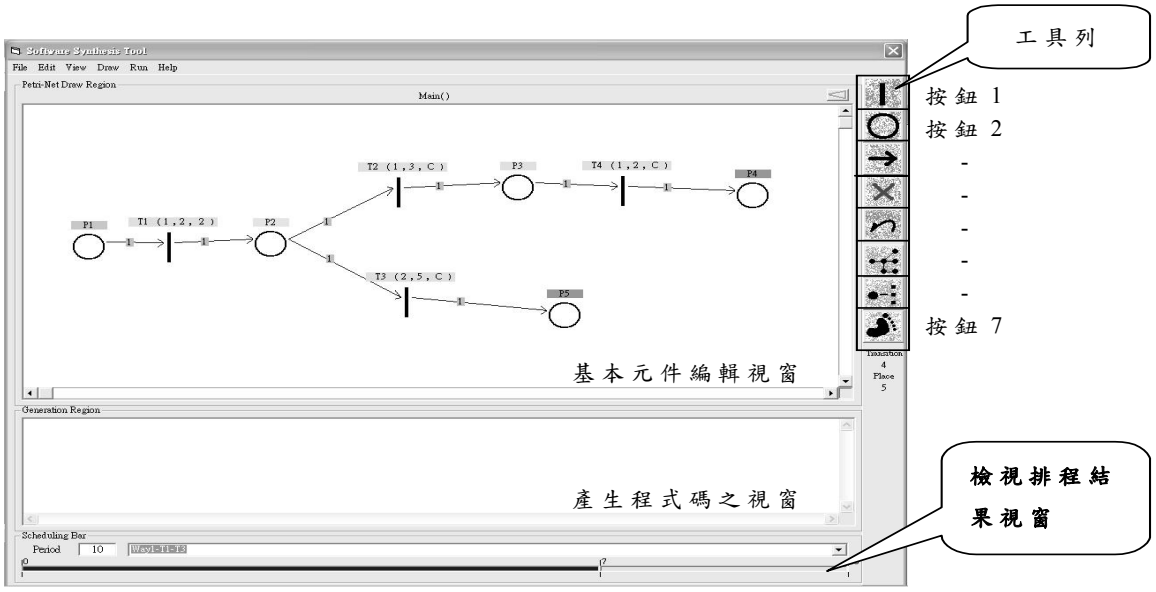

圖 3 嵌入式軟體合成工具之使用者介面

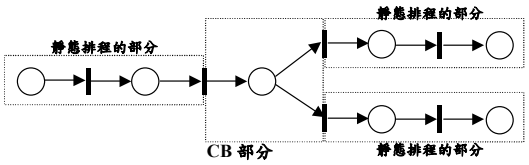

圖 4 將 ITPN 分成兩個排程部分本系統

表 1 中(1)是從系統模型中找出初始位置、結束位置與選擇區塊，(2)為找出每一個選擇區塊之輸出轉移的互斥關係並建立各自的互斥表(ExTable)，(3)是根據互斥表找出每一個選擇區塊路徑(Choice Block Set, CBS)。如圖 5 示為一個選擇區塊找出選擇區塊路徑的步驟。圖 5 的步驟一是根據選擇區塊之輸出轉移間的激發(Fire)關係所建立的，例如選擇區塊一所建立的互斥表是因為當 t_2 激發的時候 t_3 是不會被激發的，同理當 t_3 被激發時 t_2 是不會被激發的；步驟二中之路徑表是根據互斥表所決定，其中"0"代表不激發(Not Fire)，而"1"代表被激發，例如圖 5 的選擇區塊一根據互斥表可知當 t_2 被激發時 t_3 不能被激發，所以在路徑表中只有"01"與"10"項目符合互斥條件，其中"01"代表 t_3 而"10"代表 t_2；步驟三中根據路徑表中符合互斥條件的項目轉換成選擇區塊路徑，所以在圖 5 選擇區塊一總共找出二組選擇區塊路徑，而選擇區塊二則找出四組選擇區塊路徑。

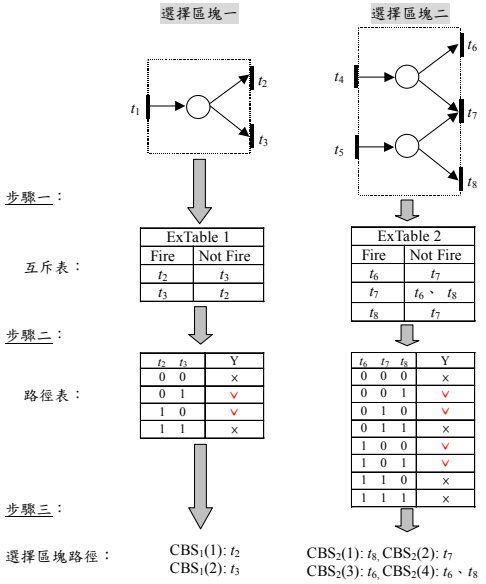

圖 5 找出選擇區塊路徑的過程

如圖 6 的選擇區塊一共可找出三組選擇區塊路徑，所以經第一次擴充後從原本一條路徑擴增為三條路徑，表 1 中(9)至(19)將目前每一條路徑當中最後一個轉移開始尋找靜態排程路徑，(10)當路徑尚未遇到結束位置將繼續尋找路徑的動作，(11)與(12)的動作與(4)相同，(13)至(15)為當回傳值為 0 時表示此路徑最後遇到一個結束位置，且找尋靜態排程路徑與目前的路徑結合，最後將路徑輸出，(16)與(17)是判斷若為最後一條路徑時將結束排程，(18)與(19)為回傳值不為 0 時表示此路徑遇到一個選擇區塊將重複(9)至(20)的動作。

例如圖 6 中當經第一次擴充後，根據每一條路徑之最後一個轉移找出排程路徑，其中第一、二條路徑所找出的為靜態排程路徑，而第三條路徑因遇到選擇區塊，所以必須在做一次擴充的路徑，所以最後有四條排程路徑輸出。當找出所有排程路徑後，由表 1 中(21)檢查每一個轉移與路徑是否滿足即時的限制，若滿足即時限制則由(22)產生對應的程式碼。

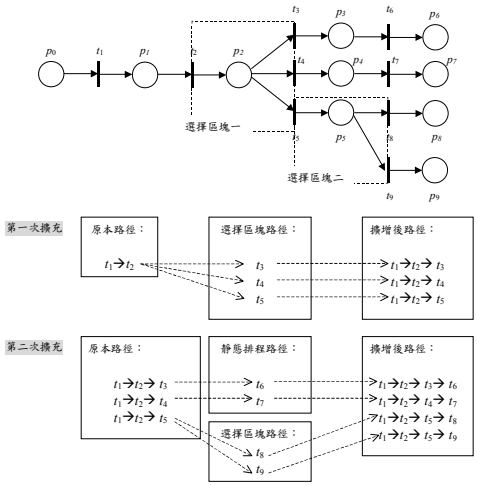

圖 6 路徑擴充的範例

表 1 類似動態排程演算法所找出來的路徑必須再符合即時時間之限制，經修改[2]之演算法並加入具有處理中斷控制之演算法列於表 2。從演算法中可以找出不符合即時時間與中斷產生限制的排程序列，其限制包括每一個排程序列的執行時間是否小於系統的執行週期(Period)與每一個轉移的最晚激發時間內是否足夠讓允許的中斷種類之中斷服務程式執行，若全部排程序列滿足即時限制則稱此系統為可排程(Schedulable)的系統。

表 2 即時限制(Real-Time Constraint)演算法

```
Real_Time_Check(Route[i]){                          //where 1≦i≦CountRoute
    bool Permit = true, Result;
    int RouteTime_j = 0, SystemPeriod;
    for each route W_j ∈ Route[i] {                                          (1)
        for each t_k ∈ W_j {        //where 1≦k≦n; n : the number of transition in W_j  (2)
            Permit = InterruptPermit(t_k);                                   (3)
                if (Permit = true) {                                         (4)
                    RouteTime_j = RouteTime_j + β(t_k)};        // β : LFT   (5)
                else {
                    Schedulable = false;                                     (6)
                    break; }}                                //End of for
        if (RouteTime_j < SystemPeriod) {                                    (7)
            Schedulable = true;}                                             (8)
        else {
            Schedulable = false;                                             (9)
            break;}}                                         //End of for  (10)
    if (Schedulable = true) {
        System is schedulable;}
    else {
        System is not schedulable;}}
//-------------------------------------------------------------------------------//
bool InterruptPermit(char tra) {
    ISRTime = γ(tra) ISR time;                                               (11)
    if (ISRTime <  β(tra) − α(tra)){                 // α : EFT             (12)
        return result = true;}                                               (13)
    else{
        return result = False;}}                                             (14)
```

表 2 中(1)至(10)是檢查每一條路徑與路徑中的每一個轉移是否滿足即時的限制，(3)為呼叫函式的動作，引數為轉移的代號且回傳值為布林值，其功能是檢查每一個轉移是否滿足即時的限制，(4)至(6)當回傳值為真時，將此轉移的最晚的激發時間加於此路徑的時間當中，若為假時，則此模型為不可排程，(7)至(10)則是檢查每一條路徑是否滿足即時的限制，小於系統的值執行週期時間，(11)至(14)是檢查每一個轉移所允許的中斷服務時間是否小於最晚的激發時間與最早的激發時間的差值，若成立則回傳值為真，反之為假。

表 3 為產生程式碼演算法，如表 3 中的(1)至(5)是判斷所產生的程式類別，其中(3)是將整個主程式當成一個反覆的程式結構。(6)為呼叫函式的動作，引數為轉移的代號且回傳值為選擇區塊的代號，其動作為依序列印出靜態排程路徑中每一個轉移之中斷致能暫存器設定與代表的程式碼，(7)當不為 0 代表遇到選擇區塊，(8)之"count"代表目前產生到哪一條路徑，(9)與(10)是將選擇區塊的分歧路徑記錄起來，表 3 中(11)為列印出程式的選擇結構，(12)與(6)是一樣的動作，(13)與(14)假如回傳值為 0 時表示此路徑已完成輸出，(15)至(19)是擴增所要接下去尋找轉移，(20)若處理到最後一條路徑時停止，(21)其將所有的副程式、中斷服務程式與主程式結合起來，完成全部的程式碼的輸出。

表 3 產生程式碼演算法

```
Code_Generation (Route[i]) {                    //1≦i≦s ; s : the number of element in Route[ ]
    int count, CountWaitVisitTra, CountCBS_X; char X, Y ; string WaitVisitTra[ ]; bool Result, Stop;
    switch(classification){                     //Differentiate main-function, sub-function, and ISR   (1)
        case "main": printf "main( ){";                                                              (2)
                     printf "while(1){";                                                             (3)
                     break;
        case "ISR": printf "void ISR(void){";                                                        (4)
                     break;
        case "Sub": printf "void Function_Name(void){";                                              (5)
                     break;}
    X = Print_Static_Route(t_j);                                                                     (6)
    //t_j is a transition which is after the Initial Place, t_j∈T, 1≦j≦u, u is the number of transition
    if (X != 0){                                                                                     (7)
        count = 1;                                                                                   (8)
        CountWaitVisitTra = CountCBS_X;
        //CountCBS_X : the number of CBS_X for CB(X), CB(X) : The X_th choice block
        //CountWaitVisitTra : the number of element in WaitVisitTra[ ]
        for (r = count; r = CountWaitVisitTra; r++){                                                 (9)
            WaitVisitTra[r] = CBS_X for CB(X) ;}                                                     (10)
        do{
            printf "if (condition){" or "else if (condition)";                                       (11)
            Y = Print_Static_Route (WaitVisitTra[count] );                                           (12)
            if (Y = 0){                                                                              (13)
                count = count + 1;}                                                                  (14)
            else{
                CountWaitVisitTra = CountWaitVisitTra + CountCBS_Y − 1;                              (15)
                for (s = count + 1; s = CountWaitVisitTra − CountCBS_Y; s++){                        (16)
                    WaitVisitTra[s + CountCBSY] = WaitVisitTra[s];}                                  (17)
                for (u = count; u = count + CountCBS_Y; u++){                                        (18)
                    WaitVisitTra[u] = CBS_Y for CB(Y);}                                              (19)
        while(count <= CountWaitVisitTra)}                                                           (20)
    Combine main-function, sub-function, and ISR;}                                                   (21)
//-------------------------------------------------------------------------------------------------//
char Print_Static_Route (char tra){
    print "{";
    do{
        printf interrupt enable and content for tra;                          //Ex : IE = 0x81
        search p_k which after tra;     //where P_K ∈ P; 1≦k≦v, v is the number of place
        if (p_k is a choice block) {
            Stop = false;                                                     //continue do loop
            Return Result = CB(q);          //1≦q≦w, w is the number of choice block
        else{
            if (p_k is an End Place){
                printf "}";
                Stop = true;                                                  //stop do loop
                Return Result = 0;}                                           //meet an end place
            else{
                Stop = false;}                                                //continue do loop
    }while(Stop = false)
```

四、應用實例

本文提出電腦數位溫度量測器與四相步進馬達控制器作為本系統之合成工具的應用實例驗證。所合成出的程式是符合 Keil C51 編輯器所要求的 C 語言語法，系統的單晶片是採用 ATMEL 89C51。

(一)、電腦數位溫度量測器

在此實例當中，將利用單晶片、類比數位轉換電路與溫度感測器電路完成溫度量測的功能，並利用單晶片的串列埠傳送至個人電腦，其硬體方塊圖如圖 7 所示，其中溫度感測電路包括溫度感測器與放大電路，類比/數位轉換電路是將類比訊號轉換成數位信號後傳送給單晶片，且利用單晶片於七段顯示器上顯示量測溫度，並利用串列埠傳送給個人電腦。我們將利用外部中斷 0、計時器中斷 0，其中類比/數位轉換電路會利用外部中斷 0 告知單晶片已將類比訊號轉換成數位訊號，而後透過輸出埠 0 將數位訊號傳給單晶片，而利用計時器中斷 0 將數位溫度訊號顯示於四個七段顯示器上，最後再利用串列埠將數位溫度數值透過 RS232 傳送給個人電腦中。

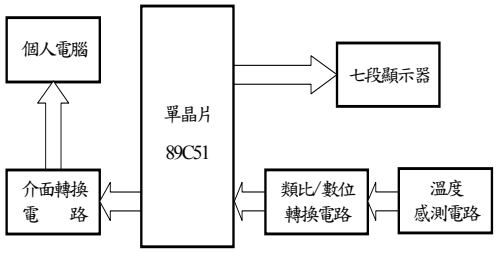

圖 7 電腦數位溫度量測器之硬體方塊圖

圖 8 為電腦數位溫度量測器之系統模型，其中 p_1 為系統開始狀態，t_1 的功能是設定鮑率的步驟，p_2 為等待開始轉換的狀態，t_2 的功能是使類比/數位電路開始轉換，p_3 為等待傳送狀態，t_3 的功能是透過串列埠將偵測到的數值透過 RS232 傳送給電腦。每一個轉移都允許外部中斷 0 與計時器中斷 0 的發生，所以每一個轉移的最晚的激發時間與最早的激發時間的差值時必須大於 34 個單位時間，實例中每一個轉移都符合此要求，且系統執行週期至少為 125 個單位時間，若要求大於 125 個單位時間，則此系統為可排程系統，最後可以按下視窗右邊由上面算起的第七個按鈕便可以產生對應的程式，電腦數位溫度量測器之程式碼產生的結果列於表 4 中。

(二)、四相步進馬達控制器

在此實例當中，利用單晶片、輸入電路、顯示電路、驅動電路與步進馬達以完成步進馬達的控制，其硬體方塊圖如圖 9 所示，以輸入電路控制馬達轉動與停止，並透過外部中斷 0 與外部電路達到

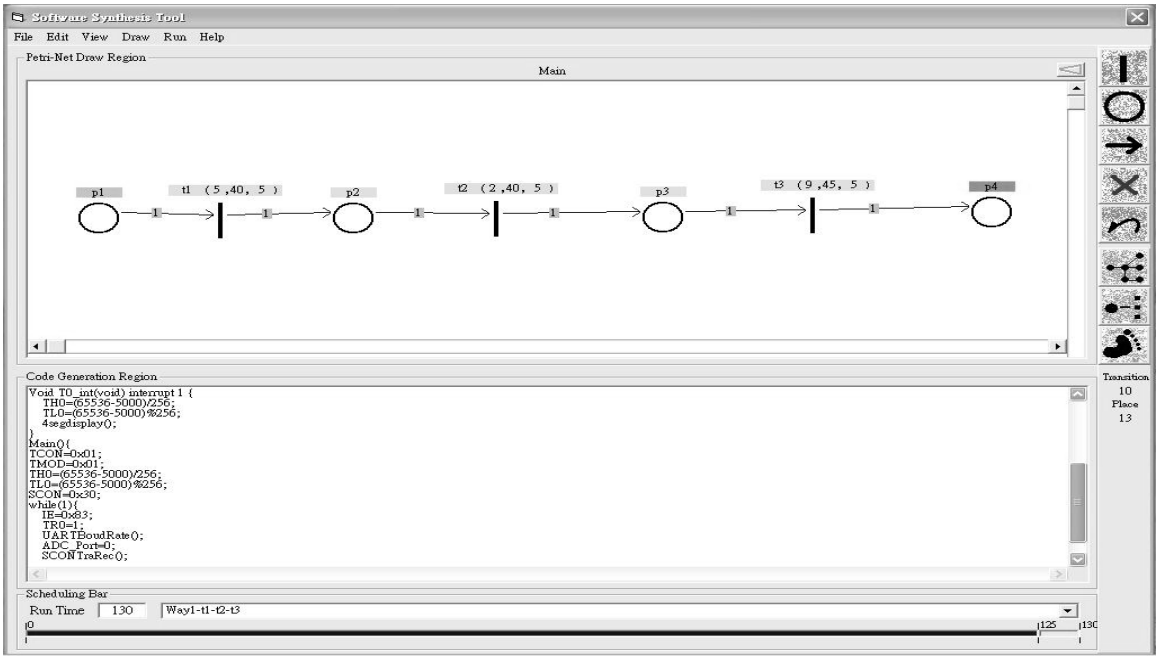

圖 8 電腦數位溫度量測器之系統模型

表 4 電腦數位溫度量測器之程式碼產生結果

```
#include "reg51.h"
Void EX0_int(void) interrupt 0 {
      simple--;
      if (p6)
      {
            simple=4000;
            value=ADC_Port;
            convert3();
      }
}
Void T0_int(void) interrupt 1 {
      TH0=(65536-5000)/256;
      TL0=(65536-5000)%256;
      4segdisplay();
}
Main(){
TCON=0x01;
TMOD=0x01;
TH0=(65536-5000)/256;
TL0=(65536-5000)%256;
SCON=0x30;
while(1){
      IE=0x83;
      TR0=1;
      UARTBoudRate(19600);
      ADC_Port=0;
      SCONTraRec();
}}
```

速度的控制，且利用計時器 0 偵測轉動與暫停的按鍵值與計時器 1 產生脈波控制步進馬達的轉動。

在實例中，在中斷工作模式設定視窗，設定外部中斷 0 為負緣觸發，計時器 0 設定為 2000 個計時脈波，且工作於模式 1，而計時器 1 其計時脈波值為一變數，且工作於模式 1，改變變數可以控制輸出脈波的數量，以控制馬達轉速。圖 10 所示為外部中斷 0 之系統模型，此系統模型的選擇區塊為一 Complex Choice 所構成的，p_{18} 為中斷開始狀態，t_{15} 為讀取按鍵值，t_{16} 為檢查是否為最高速，t_{17}

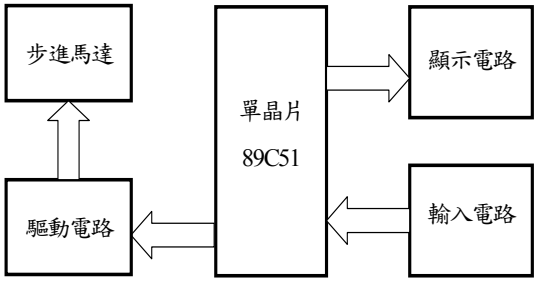

圖 9 四相步進馬達控制器之硬體方塊圖

為檢查是否為最低速，p_{19} 為判斷按鍵值為加速或減速，p_{20} 為判斷是否為最高速，p_{21} 為判斷是否為最低速，若按鍵值為加速且此時不為最高速時 t_{19} 將步進馬達加速一級，若按鍵值為減速且此時不為最高速時 t_{20} 將步進馬達減速一級，p_{22}、p_{23}、p_{24} 與 p_{25} 為結束狀態。在圖 11 為四相步進馬達控制器之系統模型，$t1$ 為讀入按鍵值，p_2 為判斷按鍵值，t_2 與 t_3 為啟動馬達，t_4 為將步進馬達停止，p_3、p_4 為步進馬達為轉動狀態，t_5 使步進馬達正轉，t_6 使步進馬達反轉，p_6、p_7 與 p_8，為結束狀態。在圖 11 中每一個轉移都允許外部中斷 0、計時器中斷 0 與計時器中斷 0 的發生，故每一個轉移的最晚的激發時間與最早的激發時間的差值時必須大於 31 個單位時間，在本實例中每一個轉移都符合此要求，且系統執行週期至少為 105 個單位時間，若要求大於 105 個單位時間，則此系統為可排程系統。將產生的四相步進馬達控制器之程式碼列於表 5 中。

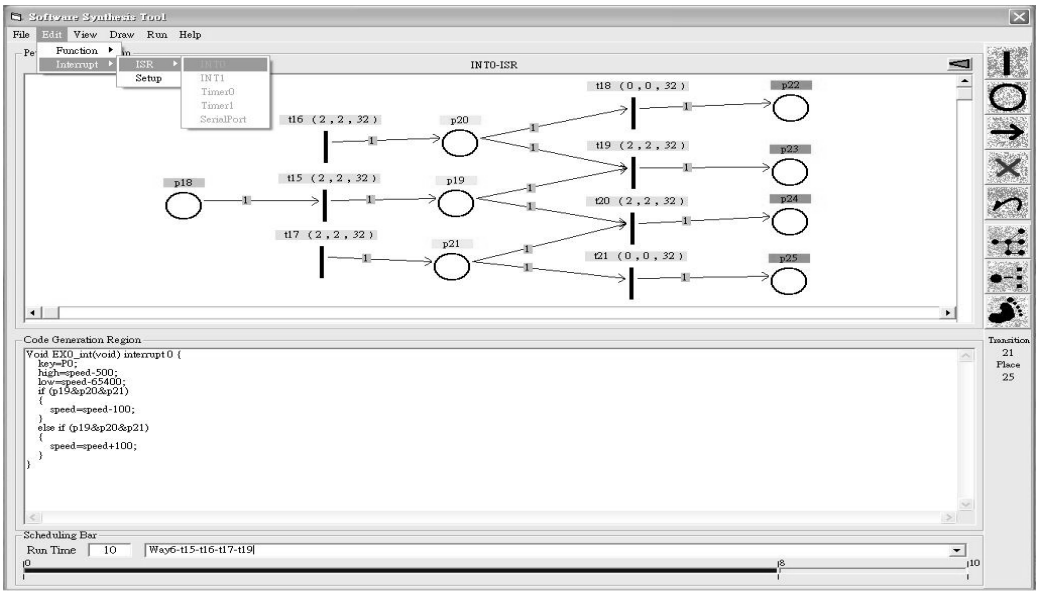

圖 10 外部中斷 0 之系統模型

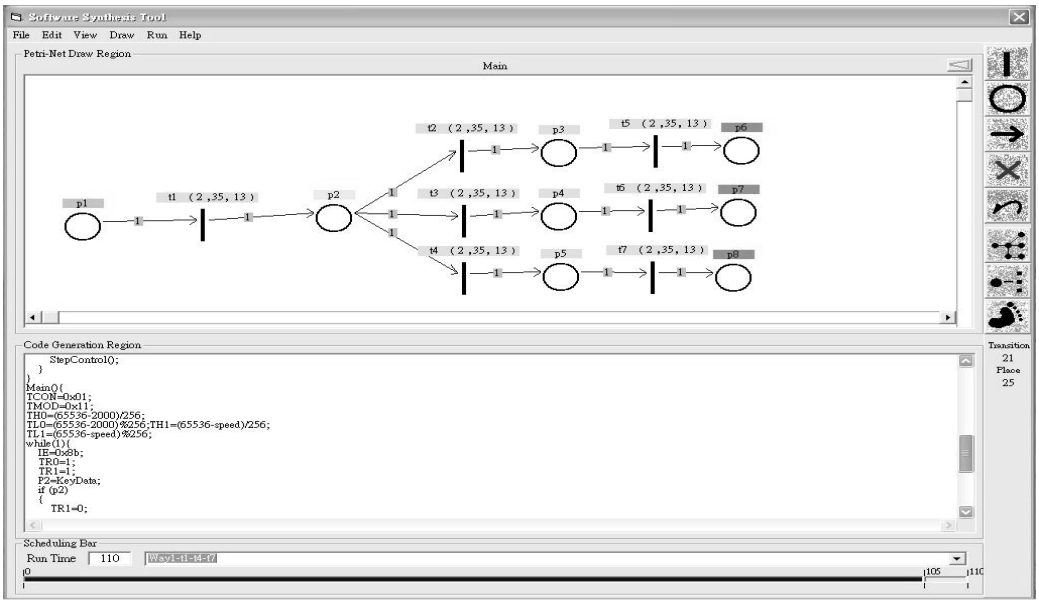

圖 11 四相步進馬達控制器之系統模型

以上兩個例子還須以手動方式加入變數的宣告、巨集的定義和選擇結構中的條件值(如表 4 中的 p_6)等細部程式碼，完成後將程式碼經過 C 編譯器編譯成目的碼在經連結器產生可執行檔，最後將此執行檔燒入[5][9]中所發展的模擬平台中驗證，此平台包括一個微控制器(89C51)提供軟體執行、FPGA 晶片提供硬體模擬與一些輸出入元件等。

五、結論與未來研究方向

為了因應產品能快速上市，所以系統設計時間相對必須縮短，本文針對即時嵌入式系統軟體發展出一個合成工具，以協助系統軟體設計者可以快速設計出所需之軟體。

我們提出以中斷式時間派翠網路(*Interrupt Time Petri Nets*, ITPN)做為系統模型，以滿足系統中之中斷行為的嵌入式系統軟體的要求，並提出具中斷處理能力的類似動態排程(*Interrupt-Based Quasi-Dynamic Scheduling*, IQDS)演算法將包含中斷行為與即時限制的行程找出有效的排程，最後利用產生程式碼(*Code Generation*)演算法產生符合單晶片 C 語言語法的程式碼。為了增加工具使用的親和度，我們建立了一個圖形化的使用者介面。此介面可以讓使用者描繪出所要的系統模型，在嵌入式軟體合成工具中，圖形化的使用者介面，幫助使

用者可以容易的描繪出所要之系統模型，並可設定中斷相關參數，並可以檢視排程與產生程式的結果，且內建一些常用的副程式供使用者使用，以減少系統設計時間，讓使用者可以更快速的發展嵌入式系統軟體，不必專注於硬體低階的設定部份。

中斷式時間派翠網路是為了符合嵌入式系統中的微控制器 8051 會使用中斷動作以完成某些特定之功能，在程式碼中需要有一些中斷相關暫存器被設定，所以程式碼當中必須含有相關的中斷設定程式碼，因此系統模型可以清楚的表示每一個轉移致能那一類中斷的產生、除能中斷與不改變中斷致能，以利於產生中斷設定程式碼。在電腦數位溫度量測器與四相步進馬達控制器實例中其系統模型中皆包括了 Free Choice 和 Complex Choice，說明了我們的軟體合成工具中的合成方法可以處理這兩種結構。

本文所提之類似動態排程演算法中，每一種中斷只能夠限制發生一次，如此將會限制系統的功能，另外本排程演算法只針對循序結構與選擇結構作排程，且把整個系統當作循環結構，本工具並未對程式結構作排程，只利用內建函式的方式來解決此問題。在產生程式碼方面，本工具無法自動產生一些細部的程式碼，因此這些問題將是未來努力的方向。

參考文獻

表 5 四相步進馬達控制器之程式碼產生結果

```
#include “reg51.h”
Void EX0_int(void) interrupt 0 {
      key=P0;
      high=speed-500;
      low=speed-65400;
      if (p19&p20&p21)
            {speed=speed-100;}
      else if (p19&p20&p21)
            {speed=speed+100;}}
Void T0_int(void) interrupt 1 {
      TH0=(65536-2000)/256;
      TL0=(65536-2000)%256;
      4KeyScan1( );}
Void T1_int(void) interrupt 3 {
      TH0=(65536-speed)/256;
      TL0=(65536-speed)%256;
      count--;
      if (p14)
         {count=100;
          StepControl( );}}
Main(){
TCON=0x01;
TMOD=0x11;
TH0=(65536-2000)/256;
TL0=(65536-2000)%256;
TH1=(65536-speed)/256;
TL1=(65536-speed)%256;
while(1){
      IE=0x8b;
      TR0=1;
      TR1=1;
      P2=KeyData;
      if (p2)
         {TR1=0;
          P1_7=1;}
      else if (p2)
         {TR1=1;
          direct=1;}
      else if (p2)
         {TR1=1;
         direct=0;}
}}
```

[1] 蔡仰恩，“嵌入式系統在工業控制上的應用”，http://www.mirl.itri.org.tw/嵌式系統在工業控制上的應用.pdf，2001。

[2] P. -A. Hsiung, “Formal Synthesis and Code Generation of Embedded Real-Time Software,” in *Proceedings of the 9th ACM/IEEE International Symposium on Hardware Software Co-design*(CODES’01, Copenhagen, Denmark), pp. 208-213, April, 2001.

[3] P. -A. Hsiung, T. -Y. Lee, and F. -S. Su, “Formal Synthesis and Code Generation of Real-Time Embedded Software Using Time-Extended Quasi-Static Scheduling,” in *Proceedings of the 9th Asia-Pacific Software Engineering Conference* (APSEC'02), IEEE Computer Society Press, pp. 395-404, December, 2002.

[4] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli, “Synthesis of Embedded Software Using Free-Choice Petri Net,” in *Proceedings Design Automation Conference* (DAC’99), ACM Press, pp. 21-25, June, 1999。

[5] T. -Y. Lee, P. -A. Hsiung, I. -M. Wu, and Su, F. S., “ESSP: An Embedded Software Synthesis and Prototyping Methodology,” in *Proceedings of the International Computer Symposium*, (ICS’2002, NDHU, Taiwan), pp. 150-157, December, 18-21, 2002.

[6] J. L. Peterson, Petri Net Theory and The Modeling of Systems, Central Book Company, 1981.

[7] T. Murata, “Petri Nets: Properties, Analysis and Application,” in *Proceedings of IEEE*, Vol. 77, No. 4, pp. 541-580, 1989.

[8] M. C. Zhou, M. D. Jeng, “Modeling, Analysis, Simulation, Scheduling, and Control of Semiconductor Manufacturing Systems: A Petri-Net Approach,” *IEEE Trans. Semiconduct. Manufact.*, Vol. 11, No. 3, pp. 333-357, Aug, 1998.

[9] T. -Y. Lee, P. -A. Hsiung, I. -M. Wu, and F. -S. Su, “RESS: Real-Time Embedded Software Synthesis and Prototyping Methodology,” in *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications* (RTCSA’2003, Tainan, Taiwan), pp. 143-158, February, 18-20, 2003.

[10] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A, Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, Hardware-software Co-design of Embedded Systems: the POLIS approach, Kluwer Academic Publishers, 1997.