The Design of a Synthesis Tool for Interrupt-based Real Time Embedded Software

Trong-Yen Lee[1], Pao-Ann Hsiung[2], I-Mu Wu[3], Chia-Chun Tsai[1], and Wen-Ta Lee[1]
tylee@ntut.edu.tw, pahsiung@cs.ccu.edu.tw, WEMOVE_WU@dbtel.com.tw,
cct@en.ntut.edu.tw, wtlee@en.ntut.edu.tw

[1]Department of Electronic Engineering, National Taipei University of Technology, Taipei, Taiwan, ROC.
[2]Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, ROC.
[3]DBTEL Taiwan Limited, Taipei, Taiwan, ROC.

Abstract

There is a general lack of CAD tools for real-time embedded software, thus we have developed a software synthesis tool with a graphical user interface for real-time embedded systems. We propose an *Interrupt Time Petri Nets* (ITPN) model for real-time embedded software requirements modeling. ITPN can handle both interrupt behavior and real-time constraints on tasks in a real-time embedded system. An *Interrupt-Based Quasi-Dynamic Scheduling* (IQDS) algorithm is proposed to find valid task schedules satisfying interrupt behavior specifications and real-time constraints in a real time embedded system. We use a *Code Generation* algorithm to produce 8051 micro-controller *C* program code. The graphical user interface makes our tool more user-friendly. This tool supports the Windows OS environment and can be used for system model input and easy setting of system parameters. Finally, we use two industrial examples to illustrate the correctness of our methodology and the increase in productivity provided by our real-time embedded software synthesis tool.

Keywords: Embedded System, Software Synthesis, Interrupt-based Quasi Dynamic Schedule, Interrupt Time Petri Nets.

1. Introduction

Due to rapid technology progress, there has been a significant increase in system complexity, decrease in time-to-market, and growing demand for real-time embedded systems. Therefore, the development of a design tool has become indispensably important for the design process of a system. Recently, the methodology for hardware-software co-design of a system has become a major focus in both academia and industries. Though there have been some major breakthroughs related to this field of research, yet to the best of our knowledge there is a general lack of a practical synthesis method for designing software in real time embedded systems. We will propose a method to address the above problem and develop a practical tool for the synthesis of real time embedded software.

A real time embedded system is required to accomplish some dedicated set of periodic tasks within real time deadlines. Some examples include avionics flight control, vehicle cruise control, and network-enabled devices in home appliances. The development time for real time embedded software is crucial to the system design process and can be reduced through various techniques, such as adopting software reuse techniques and seeking for the advancement in software synthesis and verification [1], [2], [3], [4], [5].

The purpose of this work is to not only develop a software synthesis tool for academia but also for industrial use. The proposed real time embedded software development environment will aid in shortening time-to-market. This work will use a Digital Thermometer with Microcontroller (DTM) and a Real-time Stepping Motor Control (RSMC) to demonstrate the benefits of the tool. In this tool, *Interrupt Time Petri Nets* (ITPN) will be proposed as our model for a real-time embedded system. Then, an *Interrupt-based Quasi-Dynamic Scheduling* (IQDS) will be proposed for the synthesis of real-time embedded software. We developed an algorithm for embedded software code generation.

This article is organized as follows. Section 2 gives a brief overview about previous work in real time embedded software framework development. Section 3 describes the design of software synthesis method and graphic interface in the tool. Two embedded system examples are given in Section 4. Section 5 concludes the article and gives directions for future work.

2. Previous Work

Several techniques for software synthesis from a concurrent functional specification have been proposed [7], [8], [9], [10], [11], [15], [16]. Buck and Lee [7] have introduced the *Boolean Data Flow* (BDF) networks model and proposed an algorithm to compute a *quasi-static schedule*. However, the problem of scheduling BDF with bounded memory is undecidable, *i.e.* any algorithm may fail to find a schedule even if the BDF is schedulable. Hence, the algorithm proposed by Buck can find a solution only in special cases. Thoen et al. [8] proposed a technique to exploit static information in the specification and extract from a constraint graph description of the system statically schedulable clusters of threads. The limit of this approach is that it does not rely on a formal model and does not address the problem of checking whether a given specification is schedulable. Lin [9] proposed an algorithm that generates a software program from a concurrent process specification through an intermediate Petri-Nets representation. This approach is based on the strong assumption that the Petri Net is safe, *i.e.* buffers can store at most one data unit. This on one hand guarantees termination of the algorithm, on the other hand it makes impossible to handle multirate specifications, like FFT computations and down-sampling. Safeness implies that the model is always schedulable and therefore also Lin's method does not address the problem of verifying schedulability of the specification. Moreover, safeness excludes the possibility to use Petri Nets where source and sink transitions model the interaction with the environment. This makes impossible to specify inputs with independent rate. Later, Zhu and Lin [10] proposed a compositional synthesis method that reduced the generated code size and thus was more efficient.

Software synthesis method was proposed for a more general Petri-Net framework by Sgroi et al. [11]. A quasi-static scheduling algorithm was proposed for *Free-Choice Petri Nets* (FCPN) [11]. A necessary and sufficient condition was given for a FCPN to be schedulable. Schedulability was first tested for a FCPN and then a valid schedule generated. Decomposing a FCPN into a set of *Conflict-Free* (CF) components which were then individually and statically scheduled. Code was finally generated from the valid schedule.

Balarin et al. [12] proposed a software synthesis produre for reactive embedded systems in the *Codesign Finite State Machine* (CFSM) [13] framework with the POLIS hardware-software codesign tool [13]. This work cannot be easily extended to other more general frameworks.

Recently, Su and Hsiung [15] proposed an *Extended Quasi-Static Scheduling* (EQSS) using *Complex-Choice Petri Nets* (CCPNs) as models to solve the issue of complex choice structures. Gau and Hsiung [16] proposed a *Time-Memory Scheduling* (TMS) method for formally synthesizing and automatically generating code for real-time embedded software, using the *Colored Time Petri Nets* model. Lee et al. [17] proposed a methodology called ESSP (Embedded Software Synthesis and Prototyping) for the automatic design of embedded software.

Later, Lee et al. [17] proposed a RESS (Real-time Embedded Software Synthesis) design methodology which adds real-time constraints to ESSP [17]. In our current work, we will focus on processing interrupts in embedded software synthesis, we use IQDS to synthesize the real-time embedded software and use a code generation procedure to generate the C code for 8051 microcontroller.

Several simulation or emulation boards for single chip micro-controller, such as Intel 8051 or ATMEL 89c51, have been developed. As we know, tools for real-time embedded software synthesis are still lacking. Therefore, we developed a flexible tool for real-time embedded software system.

3. Software Synthesis Tool Design

3.1 Overview of the Tool Design

The framework of our software synthesis tool is shown in Figure 1. For user friendliness, we develop a graphical user interface for easy input of embedded software specification models. Embedded software specification is represented by an *Interrupt Time Petri Net* (ITPN) model which can model the behavior of interrupt events in embedded software. The software specification is represented by ITPN which will be scheduled by the proposed *Interrupt-based Quasi Dynamic Schedule* (IQDS) algorithm. If feasible software schedules cannot be generated then we rollback to the embedded software specification and ask the user to recheck or modify the specification. A valid schedule will be found if all time constraints were met. If feasible software schedules can be generated, then a C code for 8051 microcontroller will be generated by a code generation procedure. The target machine code is finally loaded into the 89C51 or 87C51 microcontroller chip on the platform.

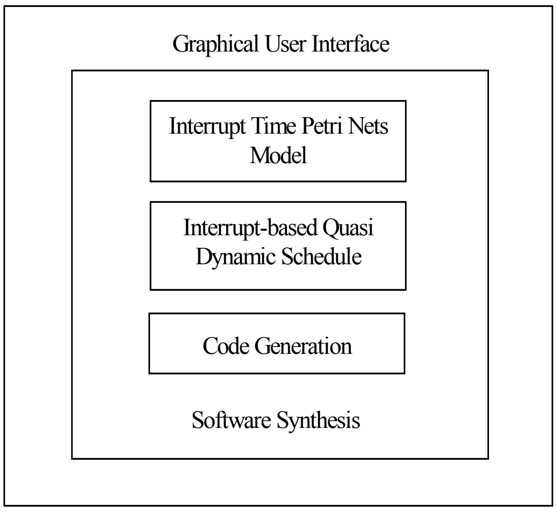

Graphical User Interface
Interrupt Time Petri Nets Model
Interrupt-based Quasi Dynamic Schedule
Code Generation
Software Synthesis

Figure 1 Software Synthesis Tool Framework

3.2 Interrupt Time Petri Nets Model

Interrupt Time Petri Nets (ITPN) are proposed for modeling embedded software specification input. ITPN is defined as follows.

Definition 1. *Interrupt Time Petri Net* (ITPN)

A *Interrupt Time Petri Net* is a 5-tuple (P, T, I, O, Ω),

P: is a non-empty finite set of places, $\{p_0, p_1, \ldots, p_n\}$.

T: is a non-empty finite set of transitions, $\{t_0, t_1, \ldots, t_n\}$.

I: is an input function, $T \rightarrow P$.

O: is a output function, $P \rightarrow T$.

Ω: $\Omega(t)=(\alpha, \beta, \gamma)$, where α: *Earliest Firing Time* (EFT), β: *Latest Firing Time* (LFT), γ: The type of interrupt in 8051 microcontroller. □

An example of the ITPN model is shown in Figure 2.

Definition 2. *Choice Block* (CB)

A *choice block* is a branch from a place P to two or more transitions. A CB includes two situations that are free choice and complex choice. In free choice, the input arc into transition is only one. In complex choice, the input arcs into transition are more than one. □

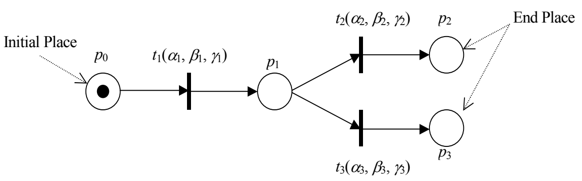

Initial Place
p_0
$t_1(\alpha_1, \beta_1, \gamma_1)$
p_1
$t_2(\alpha_2, \beta_2, \gamma_2)$
p_2
End Place
$t_3(\alpha_3, \beta_3, \gamma_3)$
p_3

Figure 2 The example of IPTN model

3.3 Interrupt-based Quasi Dynamic Scheduling

Software synthesis is a scheduling process whereby feasible software schedules are generated such that they satisfy all user-given functional requirements, timing constraints, and memory constraints. Here, we propose an *Interrupt-based Quasi-Dynamic Scheduling* (IQDS) method for the synthesis of embedded software. IQDS takes a set of *Interrupt Time Petri Nets* (ITPN) as input along with timing and memory constraints such as periods, deadlines, and an upper bound on system memory space. The IQDS algorithm has four steps as shown in the following and the detailed algorithm is shown in Table 1.

Step 1: Find the initial place, end place, and choice block (CB). The procedure is shown in item (1) on Table 1.

Step 2: Decompose the ITPN into two parts:

- Statically schedulable non-choice blocks
- The choice block set (CBS)

The procedure is shown in the item (2) and item (3) on Table 1, respectively.

Step 3: Search the routes for each CB in CBS and derive all routes from Initial Place. The procedure is shown from item (4) to item (21) on Table 1.

Step 4: Check the real-time constraints for all routes. The procedure is shown item (22) in Table 1. The detailed algorithm for checking the system real-time constraints is shown in Table 2. A valid schedule of embedded software was found by the IQDS algorithm if the all time constraints are met.

3.4 Code Generation

Code generation is a procedure which generates the Intel 8051 code from the valid schedules. The procedures of code generation are shown in the following and the detailed algorithm is shown in Table 3.

Step 1: Differentiate main function, sub-function, and Interrupt Service Routine (ISR). The procedures are shown in item (1) to (5) on Table 3.

Step 2: Print transition's content from initial place. The procedure is shown in item (6) on Table 3.

Step 3: Print "if then else". The procedures are shown in item (7) to (20) on Table 3.

Step 4: Combine main function, sub-function, and ISR. The procedure is shown in item (21) on Table 3.

3.5 Graphical User Interface

Our tool has a *Graphical User Interface* (GUI) for embedded software synthesis. This tool is designed under the following environment: Pentium IV 1.4GHz CPU, 256MB DDR RAM, Windows XP OS, Visual Basic 6.0 programming language.

Table 1 Interrupt-based quasi-dynamic scheduling (IQDS)

```
IQDS_Scheduling(P, T, I, O, Ω) {
    int CountRoute=1;                              //CountRoute : the number of element in Route[ ]
    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≦i≦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∈T, 1≦j≦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)}                                                            (21)
    Real_Time_Check (Route[l]);                                  //1≦l≦CountRoute           (22)
```

Table 2 Check real-time constraints algorithm

```
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)
```

The input consists of, graphical ITPN models and the output is Keil *C* code.

The tool window is shown in Figure 3. The GUI functions include (1) model entry: edit, add, and delete components, (2) scheduling results display, and (3) printing Intel 8051 *C* code. The tool can be used for generating real time embedded software through synthesis as shown in the following:

Table 3 Code generation algorithm

```
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)
```

Step 1: Enter the system specification graphically.

Step 2: Enter the sub-functions and ISRs.

Step 3: Choose transition's contents.

Step 4: Scheduling and code generation.

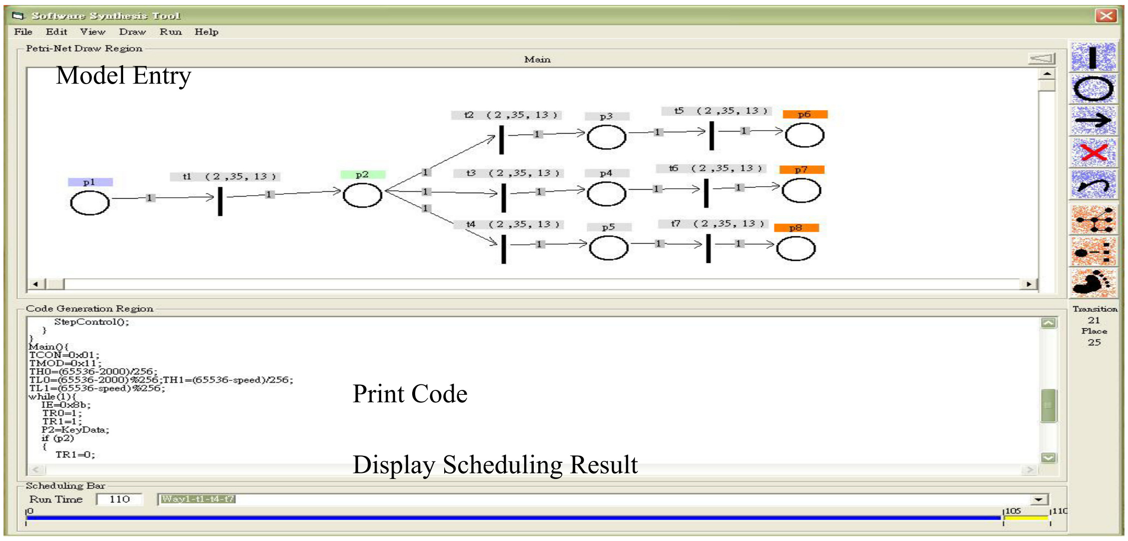
4. Embedded System Examples

In this section, we use two embedded system examples to illustrate our proposed software synthesis tool. First, we show the *Digital Thermometer with Microcontroller* (DTM) example. The function of DTM system is to measure and display the environment temperature in real time using a microcontroller. The system software will be embedded into the memory of the microcontroller. First, the ITPN model of the DTM system is entered using our tool. Then, the transitions in the DTM system are entered for which real time constraints and interrupts are specified. Due to page limits, the final partial code generated for the DTM system is shown in Figure 4.

The other example is a *Real-time Stepping Motor Control* (RSMC). The embedded software specification is to control the speed of stepping motor in real time with speed change including interrupt events. Due to page limits, only partial code synthesis by our tool is shown in Figure 5.

5. Conclusion and Future Work

A graphical software synthesis tool for



```
    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;
```

Figure 3 Graphical user interface of IQDS tool

```
#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();
}}
```

Figure 4 Partial code for the DTM system

```
#include “reg51.h”

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;}
}}
```

Figure 5 Partial code for the RSMC system

real-time embedded system was developed, including a graphical user interface, an interrupt-based quasi-dynamic scheduling, and a code generation procedure. The tool will reduce development time for embedded software.

This version of our embedded software synthesis tool is only supports the 8051 microcontroller. Therefore, we will improve it by adding the code generation of ARM microcontroller in our next version.

6. References

[1] K. Altisen, G. Gössler, A.Pneuli, J. Sifakis, S. Tripakis, and S. Yovine, “A framework for scheduler synthesis,” In *Proceedings of the Real-Time System Symposium* (*RTSS’99*), IEEE Computer Society Press, 1999.

[2] F. Balarin and M. Chiodo. “Software synthesis for complex reactive embedded systems,” In *Proceedings of International Conference on Computer Design (ICCD’99*), IEEE CS Press, October 1999. 634 – 639.

[3] L. A. Cortes, P. Eles, and Z. Peng, “Formal co-verification of embedded systems using model checking,” In *Proceedings of EUROMICRO*, 2000. 106-113.

[4] P. -A. Hsiung, “Formal synthesis and code generation of embedded real-time software,” In *Proceedings of the International Symposium on Hard-ware/Software Codesign* (CODES'01, Copenhagen, Denmark), ACM Press, New York, USA, April 2001. 208 – 213.

[5] P. -A. Hsiung, W.-B. See, T.-Y. Lee, J.-M Fu, and S.-J. Chen, “Formal verification of embedded real-time software in component-based application frameworks,” In *Proceedings of the 8th Asia-Pacific Software Engineering Conference* (APSEC 2001, Macau, China), IEEE CS Press, December 2001.71-78.

[6] T.-Y. Lee, P.-A. Hsiung, and S.-J. Chen, “A case study in codesign of distributed systems — vehicle parking management system,” In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'99, Las Vegas, USA), CSREA Press, June 1999, 2982–2987.

[7] J. Buck, *Scheduling dynamic dataflow graphs with bounded memory using the token flow model*, Ph. D, dissertation, UC Berkeley, 1993.

[8] F. Thoen et al, “Real-time multi-tasking in software synthesis for information processing systems,” In *Proceedings of the International System Synthesis Symposium*, 1995, 48-53.

[9] B. Lin, “Software synthesis of process-based concurrent programs,” In *Proceedings of the IEEE/ACM 35th Design Automation Conference* (DAC’98), June 1998, 502-505.

[10] X. Zhu and B. Lin, “Compositional software synthesis of communicating processes,” In *Proceedings of the IEEE International Conference on Computer Design*, October 1999, 646-651.

[11] M. Sgroi and L. Lavagno, “Synthesis of embedded software using free-choice Petri nets,” In *Proceedings of the IEEE/ACM 36th Design Automation Conference* (DAC’99), June 1999, 805-810.

[12] F. Balarin and M. Chiodo, Software synthesis for complex reactive embedded systems. In *Proceedings of International Conference on Computer Design* (*ICCD’99*), IEEE CS Press, October 1999, 634-639.

[13] F. Balarin et al., *Hardware-software Co-design of Embedded Systems: the POLIS Approach*, Kluwer Academic Publishers, 1997.

[14] J. Zhu and R. Denton, “Timed Petri nets and their application to communication protocol specification,” In *Proceedings of the 21st IEEE Military Communication Conference*, San Diego, CA, 1988, 195-199.

[15] F.-S. Su and P.-A. Hsiung, “Extended quasi-static scheduling for formal synthesis and code generation of embedded software,” In *Proceedings of the 10th IEEE/ACM International Symposium on Hardware/Software Codesign*, (CODES'02, Colorado, USA), IEEE CS Press, May 2002.

[16] C. -H. Gau and P. -A. Hsiung “Time-memory scheduling and code generation of real-time embedded software,” In *Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications* (RTCSA'02, Tokyo, Japan), March 2002, 19-27.

[17] T.-Y. Lee, P.-A. Hsiung, I-Mu Wu, and Feng-Shi Su, “ESSP: An Embedded Software Synthesis and Prototyping Methodology,” In *Proceedings of the International Computer Symposium*, (ICS’2002, NDHU, Taiwan), December 2002, 150-157.

[18] 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), February 2003. 143-158.