A GRAPHICAL METHOD FOR AUTOMATIC CODE GENERATION FROM EXTENDED S-SYSTEM PETRI NET MODELS

NG KOK MUN

KOLEJ UNIVERSITI TEKNOLOGI TUN HUSSEIN ONN

KOLEJ UNIVERSITI TEKNOLOGI TUN HUSSEIN ONN

PENGESAHAN STATUS TESIS

A GRAPHICAL METHOD FOR AUTOMATIC CODE GENERATION FROM EXTENDED S-SYSTEM PETRI NET MODELS

SESI PENGAJIAN : 2005/2006

Saya **NG KOK MUN** mengaku membenarkan Tesis Sarjana ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Kolej Universiti Teknologi Tun Hussein Onn.
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. ** Sila tandakan (√)

☐	SULIT	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
☐	TERHAD	(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan
√	TIDAK TERHAD	

Disahkan oleh

(TANDATANGAN PENULIS)

(TANDATANGAN PENYELIA)

Alamat Tetap:

NO 6, LORONG PJS 7/15 B,
BANDAR SUNWAY, 46150
PETALING JAYA, SELANGOR

Tarikh: 24/7/06

PM. DR. ZAINAL ALAM BIN HARON
Nama Penyelia

Tarikh: 24/7/06

CATATAN:

** Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali sebab dan tempoh tesis ini perlu di kelaskan sebagai SULIT atau TERHAD.

A GRAPHICAL METHOD FOR AUTOMATIC CODE GENERATION FROM EXTENDED S-SYSTEM PETRI NET MODELS

NG KOK MUN

A thesis submitted in fulfillment of the requirements for the award of the Degree of

Master of Electrical Engineering

Department of Electrical and Electronics
Faculty of Electrical Engineering
Kolej Universiti Teknologi Tun Hussein Onn

JULY, 2006

"I hereby declare that the work in this thesis in my own except for quotations and summaries which have been duly acknowledged"

Signature : .....

Name of Candidate : NG KOK MUN

Date : 24/7/06

Supervised by :

Supervisor .....

: PM DR. ZAINAL ALAM BIN HARON

"I/We declare that I/We have read through this thesis and in my/our view, this thesis is sufficient in fulfilling the scope and quality for the purpose of awarding the Master of Electrical Engineering"

Examiner I : PROF. DR. MOHAMED KHALIL BIN MOHD. HANI
Unversiti Teknologi Malaysia

Examiner II : PROF. MADYA DR. ROSZIATI BINTE IBRAHIM
Kolej Universiti Teknologi Tun Hussein Onn

Chairman : DR. CHRISTY A/L PATHROSE GOMEZ
Kolej Universiti Teknologi Tun Hussein Onn

Date : 20/6/06

Dedicated to my parents and siblings

ACKNOWLEDGEMENT

I would like to convey my appreciation to my supervisor, Associate Professor Dr. Zainal Alam bin Haron for introducing Petri Net and also providing the necessary guidance in this work. The constructive criticism, feedbacks and advice provided encouraged me to put more effort in this write up.

I am also touched by the encouragement and prayers given by my church friends during this process of doing this research. Not to forget, the financial support and help given when I am in financial need during the initial stage of my course. Special thanks to my siblings Yee Fong, Yee Fun and Kok Kee for being my scholarship's guarantors.

Finally, all praises and glory be to my Lord Jesus Christ for His love and provision of finance, wisdom and strength to complete this work. To God be the glory!

ABSTRACT

This work has introduced a fast and reliable method for graphical modeling of discrete systems control problems using extended S-system Petri Net. By adding new functionalities to the extended S-System Petri Net, dynamic quantities such as microcontroller signals transitions, system timing, interrupts, subroutines and arithmetic operations could now be modeled by software. A graphical-based diagram editor has been developed in this work to handle the model entry, editing and visualization. The diagram editor contains all the basic facilities required for entering, editing, visualization and syntax analysis of the S-System Petri Net model. A compiler has also been built to compile the graphical model and generate the assembly code automatically. Together, the diagram editor and model compiler forms an integrated design and development tool called S-PNGEN. Seamless data binding between the diagram editor and the model compiler is achieved by using a common directed-graph framework to internally represent the model diagrams. Diagram syntax checking was implemented using *attributed graph grammar*. Also introduced in this work is an efficient method for implementing the control solutions on a microcontroller. This involves the development of a procedure for automatically mapping S-System Petri Net models constructed in the diagram editor to control flow graphs. The procedure uses a notion called graph nesting to help the design tool read and understand S-System model diagrams and transform them into control flow graphs. Conversion of an S-System Petri Net model into a control flow graph is an innovative approach introduced in this work for automatic code generation as it guarantees the production of the correct code layout and information for use by the compiler. By applying a syntax-directed translation on the control flow graph constructed, the built-in compiler then automatically generates the assembly code for the target microcontroller.

ABSTRAK

Penyelidikan ini memperkenalkan kaedah yang effisien untuk membentuk model bagi sistem kawalan diskrit secara grafikal dengan menggunakan *extended S-System Petri Net*. Dengan menambahkan fungsi-fungsi baru ke atas suatu *S-System Petri Net*, kuantiti dinamik suatu pengawal mikro seperti isyarat, masa, subrutin, *interrupts* dan operasi aritmetik dapat dimodelkan oleh *software*. Satu *diagram editor* telah dibina untuk membolehkan pelukisan dan pengubahsuaian model. *Diagram editor* ini mempunyai kemudahan asas yang membolehkan pembentukan dan pengubahsuaian model serta melaksanakan analisis sintaks ke atas model *S-System Petri Net* yang dibina. Satu pengkompil telah dibangunkan untuk mengkompil model grafikal yang dibina dan juga untuk menjana kod *assembly* secara otomatik. Kedua-dua *diagram editor* dan pengkompil diintegrasikan sebagai suatu alat rekabentuk model dipanggil S-PNGEN. Kedua-dua *diagram editor* dan pengkompil berkongsi data dengan menggunakan rangka struktur data graf yang sama bagi mewakili model yang dilukis. Sintaks model diimplementasikan melalui *attributed graph grammar*. Hasil kerja ini juga memperkenalkan suatu prosedur yang memetakan model *S-System Petri Net* yang dibina dalam *diagram editor* kepada *control flow graphs*. Prosedur ini menggunakan suatu konsep *graph nesting* yang membolehkan alat rekabentuk kami membaca dan memahami model *S-System Petri Net* dan mengubahnya kepada *control flow graphs*. Pertukaran model kepada *control flow graphs* merupakan satu inovasi di dalam kerja ini untuk menjana kod secara otomatik kerana ia dapat memberikan bentangan kod yang betul dan maklumat untuk kegunaan pengkompil. Dengan mengaplikasikan *syntax-directed translation* ke atas *control flow graphs* yang dibina, pengkompil dapat menjanakan kod *assembly* untuk suatu pengawal mikro secara otomatik.

TABLE OF CONTENTS

CHAPTER ITEMS PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE

LIST OF ABBREVIATIONS

ABBREVIATIONS	MEANING
$B(PN)^2$	Basic Petri Net Programming Notations
CAD	Computer-aided Design
DLL	Doubly Linked-List
EEPROM	Electrically Erasable Programmable Read Only Memory
IDE	Integrated Development Environment
lhs	Left-Hand Side
OOP	Object-Oriented Approach
PLC	Programmable Logic Controllers
PLD	Programmable Logic Devices
PN	Petri Net
rhs	Right-Hand Side
RISC	Reduced Instruction Set Computer
SDK	Standard Development Kit
SFC	Sequential Function Chart
SIPN	Signal Interpreted Petri Net
SLL	Singly Linked-List

LIST OF APPENDIXES

CHAPTER I

INTRODUCTION

1.1 Background

Since its introduction by Carl Petri in 1962, Petri net (PN) has found a number of important applications in the areas of modeling sequential behavior and process concurrency. In the manufacturing environment, the net-theoretic approach of PN has made it especially valuable in the modeling and design of discrete control and manufacturing systems (Desrochers and Al-Jaar, 1995) (Zhou and Venkatesh, 1999). In the area of control system modelling, the ordinary PN of Carl Petri has been subject to numerous simplifications on the one hand, and extensions on the other hand, tailored according to the level of sophistication expected of the formal model. In general, the simplifications have been intended to result in a simple formal model for the complex systems studied, while the extensions have been used to add functionalities to the net which would widen the scope of applications, such as for developing a full model of the hardware and software characteristics of logic and digital controllers. Also, the extended PNs have been used as formal models for developing more systematic and structured controller programs (Frey and Litz, 2000).

Grafcet which is a subset of PN is a notable example of an extended PN. Drawing its inspiration from PN, Grafcet has been used as the basis for the international standard Sequential Function Chart (SFC), a graphical language for specifying programmable logic controllers (PLC) (David, 1995). Another example

of extended PN is Signal Interpreted PN (SIPN), which allows explicit description of input/output in a well defined way; its application in specifying PLC is found in (Frey and Minas, 2000) and (Minas and Frey, 2002).

Encouraging results have been obtained in the areas of specifying digital controllers and automatic code generation by extending the features of PN. (Machado, Fernandes and Proenca, 1997) for example, has used shobi-PN (a PN extension approach based on SIPN) to specify logic control in programmable logic device (PLD). Their work resulted in automated VHDL code for PLDs. Petri Net for Digital Systems (PNDS) proposed by (Oliveira and Marranghello, 2000) contains most of the features needed for a methodical modeling of digital systems.

An Extended Quasi-Static Scheduling (EQSS) method for formally synthesizing and automatically generating code for embedded software using the Complex-Choice Petri Nets (CCPN) models has been proposed in (Su and Hsiung, 2002). Their work resulted in generation of POSIX based multi-threaded embedded software program in the C programming language. The C code generated is applicable for hardware platform such as Application Specific Integrated Circuits (ASICs), Application Specific Instruction Set Processors (ASIPs) and PLDs. Another approach using PN extension called Timed Free Choice Petri Nets (TFCPN) to model embedded real time software (ERTS) is found in (Hsiung, Lee and Su, 2002). The objective of the work is to synthesize complex ERTS to meet up limited embedded memory requirements and to satisfy hard real-time constraints.

In the area of control system design, current design requirements and practices have reached a high degree of complexity that prevents their efficient realization without sophisticated computer-aided specification and implementation tools (Fernandes, Adamski and Proenca, 1997) (Frey and Minas, 2000) (Minas and Viehstaedt, 1995). *Specification* here is concerned with the description of the PN model and its attributes, which can be specified either textually through textual editors or graphically through CAD tools. The word *implementation* in the software context refers to the generation of a piece of code written in some computer programming language. This code should be ready to use when a low-level language

is employed, or directly convertible to a machine understandable one when a high-level programming language is used.

It has been realized from early on, a user-friendly means of controller specifications-writing is a graphical-based CAD tools which provide all the basic functions required by a user to draw the PN model and define its attributes. To this end, a number of customized diagram editors have been developed which allow the user to draw and edit the PN models on the computer screen. A notable example is DIAGEN, the diagram editor in (Frey and Minas, 2000) and (Minas and Frey, 2002). DIAGEN allows the user to specify PLC from SIPN specifications by providing the necessary drawing tools and facilities, such the free-hand editing facility, which thus allows the user to freely create, delete and modify diagram components (places, transitions and tokens for the SIPN). Other examples of PN diagram editor such as SOPHIA is found in (Fernandes, Adamski and Proenca, 1997) and *EnVisAge* (Extended Coloured Petri-Net Based Visual Application Generator Tool) in (Kurdthongmee, 2003). SOPHIA is used to specify shobi-PN while *EnVisAge* is used to construct Color Petri Net (CPN) to specify a microcontroller.

In all the tools reported in the literature, conversion of the PN model into object code for the target system is carried by a customized compiler. In the work reported by (Frey and Minas, 2000) and (Minas and Frey, 2002), for example, a customized compiler was used to generate the PLC's Instruction List (IL) code directly from the SIPN model. In SOPHIA (Fernandes, Adamski and Proenca, 1997), VHDL code for PLD was automatically generated by the compiler from the shobi-PN model. In (Melzer, 1997), a code generator is specifically developed to translate the $B(PN)^2$ notations into C language for a UNIX multiprocessor platform.

The extensions added have all been inspired by the applications intended for the PN. Extended PNs such as SIPN and Grafcet, have been derived from the need to design and specifiy PLC programs and also hybrid systems (Guillemaud and Gueguen, 1999). Both Grafcet and SIPN are naturally suited for modeling and simulating the PLC hardware and control characteristics. In particular, the functionalities available in Grafcet have a very important industrial application in the area of PLC programs specifications and design while the peculiar structure of

SIPN allows it to be directly translated to the instruction list (IL) code of a PLC (Frey, 2000). Another type of extended PN, namely the CPN, has been used to develop a code generator from the given specifications of a security access system (Mortenson, 1999). Similar work adopting CPN such as Color Timed Petri Nets (CTPN) (Gau and Hsiung, 2002) and Extended Color Petri Nets (ECPN) (Kurdthongmee, 2003) had contributed to synthesis of software code for embedded system.

Research works mentioned above have shown a wide applicability of PN and its extensions in specifying controllers. Most of the work carried out aim to provide solutions in programming controllers such as PLCs, ASICs, ASIPs, PLDs and even general microprocessor platform such as UNIX. There is without doubt that the methodologies developed resulted in *specification* and *implementation* tools to generate a piece of code for such controllers. However, the application of PNs in specifying microcontrollers is not widely studied. Motivated by the fact that PN is a useful modeling tool, this work proposes the usage of PN to specify a type of microcontroller.

1.2 Problem statement

In the area of microcontroller modeling and program design, however, extended PNs such as Grafcet and SIPN are simply inapplicable. Microcontrollers, because of their very different architecture from PLCs, are predicated on a platform that strictly executes programs in a sequential manner. Because of this characteristic, Grafcet and SIPN are simply incompatible for microcontroller modeling and program specifications; the main limitation being their tendency to over-describe (or under-describe) the characteristics of a microcontroller's program. For instance, SIPN, which allows explicit description of input/output signals, though having the ability to fully describe a microcontroller's output/input behavior, is incapable of describing characteristics such as subroutines, interrupts, time delays and arithmetic operations.

Approaches such as CCPN in (Hsiung and Su, 2002), TFCPN in (Hsiung, Lee and Su, 2002) and CTPN (Hsiung and Gau, 2002) have proposed methods to generate embedded software with multiple threads, which can be processed for dispatch by a real-time operating system. These PN extensions are also used to solve memory size constraint and concurrent task requirements in embedded systems (e.g. PLDs, ASICs and ASIPs). The methods developed suited devices with "hardwired" architecture and microprocessors that operate with an operating system. This again hinders direct applications of these extensions to specify microcontroller as the microcontroller possesses a single-threaded architecture. Multi-threading activities are not supported in microcontrollers due to the absence of an operating system.

In (Kurdthongmee, 2003), CPN has been used to specify MCS-51 family of microcontrollers. The work had achieved a few envisaged end-points such as the ability to perform model execution analysis and generation of C code for the MCS-51 microcontrollers. Some drawbacks of the work are found in its inability to describe hierarchy functions which makes the model more difficult to read and interpret. Besides, the method introduced does not indicate how interrupts are handled. The author himself stated that the user needs to go through a steep learning curve in using CPN to specify MCS-51. This will somehow affect the user-friendliness of the prototype developed.

In the light of the deficiencies highlighted above, a different type of PN is needed which can satisfactorily address the following requirements:

- It must be able to describe control flow characteristics of a microcontroller program.
- It must be capable in handling routines such as subroutines, timing routines and interrupt handling routines.
- It must be capable of specifying input/output signals of a microcontroller, and
- It must be capable of describing arithmetic operations.

1.3 Research objectives

Based on requirements highlighted in section 1.2, this work has sought to develop a PN model which would satisfactorily address the above-listed modeling requirements. Literature survey carried out at the start of the work by the author has per chance introduced him to the work of (Jorg and Ezparza, 1995) on S-System PN. By imposing the requirement of having only one input and one output arc at every transition, the authors have enabled the PN to model asynchronous control systems.

The objective of the research is to further enhance the S-System PN of (Jorg and Ezparza, 1995) by using some of the extensions introduced by others elsewhere in the literature. We report in this work the details of the enhancements added to the S-System model of (Jorg and Ezparza, 1995) and benefits brought about by the extension in modeling the hardware and software characteristics of the chosen microcontroller.

Another objective of this work is the development of a prototype tool that comprises of a CAD tool (diagram editor) that allows specification of extended S-System PN models and a compiler that implements the specified model to automatically generate the assembly code for the target microcontroller.

1.4 Research scope

The target microcontroller used in this work is PIC 16F84 which is an 8-bit, RISC type, Harvard architecture mid-range microcontroller from the PIC micro family. Sample application of the extended S-System PN model to selected control problems and the method of their specifications and solutions are shown as a guide to its application procedures. These includes modeling of input/output signals, arithmetic operations, interrupts, serial peripheral interface communication and delays in the PIC 16F84.

Towards this end a basic prototype tool comprising of a diagram editor and a compiler has been developed. The design of the diagram editor is on graphs and graph grammars (Minas 1998) (Minas 1999) (Minas 2002) and it allows on-screen model entry and editing. Also provided in this tool are facilities for text parsing with context-free string grammars; diagram parsing with attributed graph grammars and syntax-directed assembly code generation. Also explained in this work is the simulation method used to verify and validate the assembly code generated by the compiler.

1.5 Thesis outline

This thesis comprises of six chapters. Chapter 2 provides reviews of related works to substantiate the details of the functionalities added to the S-System PN for describing a microcontroller program. Besides, it also reviews works related to the development of the prototype tool.

Chapter 3 is dedicated to explain the modeling and design of the prototype tool which comprises of a diagram editor and a compiler in detail. This chapter explains the function of the graphical editor and model editor that helps to construct the PN model of the controller graphically and to define the properties or attributes of the model. Also explained is the design and working principle of the compiler's parsing and code generation operations.

Chapter 4 describes the implementation of the prototype tool. Here, methods on code generation are presented in detail. Chapter 5 serves as a testing platform for the prototype tool. It provides details of the simulation work conducted on samples of assembly code generated by the code generator. Finally, chapter 6 provides the conclusion and discussions on the limitations identified with the model and modeling approaches adopted in this work. Also discussed here are recommendations for future work.

CHAPTER II

LITERATURE REVIEW

2.1 Introduction

This chapter reviews some related work in PN extensions to substantiate additional functionalities to be added to the ordinary S-System PN and discuss the application of these functionalities in microcontroller hardware and software modeling applications. The chapter is divided into a few sections. In section 2.2, we give a brief overview of the S-System PN; this is followed by discussions on various extension properties and their practical applications. A method to correlate control flow graph with S-System PN is found in section 2.3. Some literature reviews on specification and implementation concepts of a prototype tool such as graphs as internal representation of diagrams (Section 2.4), diagram parsing (Section 2.5), textual parsing and code generation technique (Section 2.6) are also presented. Section 2.7 summarizes this review by emphasizing on the concepts formulated or adopted to develop the prototype tool in this work.

2.2 A brief overview of the S-System PN

S-System PN which is discussed in detail in (Jorg and Esparza, 1995) is a bipartite directed graph represented by 3-tuple, PN = (P, T, A, W, M_0) where:

- $P = \{p1, p2, p3, ..., p_n\}$ is a finite set of places
- $T = \{t1, t2, t3, ..., t_n\}$ is a finite set of transitions
- $A \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs
- $W: A \rightarrow 1$ is the weight function attached to each arc
- $M_0: P \rightarrow \{0,1,2,.....\}$ is the initial marking

A defining characteristic of S-System PN is its restriction of allowing only one input arc and one output arc for each transition. Places in an S-System PN, on the other hand, can have more than one input and output arcs. While this property at first sight seem very trivial, it is nevertheless of fundamental consequence since by its inclusion, it allows the S-System PN to express in a formal manner the causality and the sequential attributes of an asynchronous control system. Together with its characteristic structure which allows encapsulation of algebraic assignments and expressions, the sequential nature of S-System PN can be utilized to mirror the sequential execution of a microcontroller program.

2.2.1 Adding extended functionalities to the S-System PN

The first issue to be tackled is identifying the functionalities to be added to allow the ordinary S-System PN to be used for modelling microcontroller signal and timings, and also handle arithmetic operations, interrupts and subroutines. The sub-sections following hereon describe the respective functionalities added and their practical implications.

2.2.1.1 Practical implications of the S-System PN components

Places of a PN can be associated to certain characteristics of a system or controller. A place in Grafcet, for example, corresponds to a component of the state (David, 1995). In manufacturing systems, places are used to model resources in the system (Desrochers and Al-Jaar, 1995). (Oliveira and Marranghello, 2000) have

defined three kinds of places to describe storage components, local states of system components, or functional components (behaviours) of the system being modelled. Their work has demonstrated that places, when used appropriately, can correctly model certain characteristics of a controller.

In this work, we have used places in the S-System PN to describe the state of a system. This approach is similar to Grafcet, where each place (step) represents a particular state of the system. The places also function as avenues for interrupts, read/write operation and serial peripheral interface protocol for the microcontroller. In the S-system PN, these functions are described in terms of algebraic assignments and expressions.

A transition in S-System PN, on the other hand, is associated with a condition. This condition is also constructed as an algebraic assignment or expression. A condition tests the state of input components and also represents arithmetic operations, timing and serial peripheral interface protocol in the microcontroller. In this sense, the S-System PN provides a higher description than either SIPN or Grafcet, wherein their use of Boolean equations only allows them to describe the logic state of the system (David, 1995). The transition enabling condition can therefore be used to pass program control from its input place to an output place when the condition is fulfilled.

It is also possible to have an S-System PN with transitions without any firing conditions. The rules for firing of a transition in the S-System PN are as follows:

- A transition is enabled if all its pre-places are marked and all its post-places unmarked
- A transition fires immediately if it is enabled and its firing condition is fulfilled or executed
- A transition without any firing condition fires immediately if it is enabled.

2.2.1.2 Describing hierarchy and subroutines using macro places

A subroutine is a useful portion of code in a microcontroller program and it can have several individual lines of program code that describe a function that is repeatedly used in the program. In the S-System PN, a subroutine in a microcontroller program is represented by a macro place. A macro place is a useful component for fully specifying a controller with a hierarchical structure. A macro place is known as "macro step" in Grafcet (Michel, 1990), "macro places" in (Amroun and Bolton, 1990), “super places" in (Frey and Minas, 2000) and "supermodes" in (Mirkowski and Yakovlev, 1998). The symbol of a macro place is shown in figure 2-1.

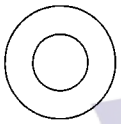


Figure 2-1 Macro place

For each macro place at any given moment there exists for it one hierarchical instance. Figure 2-2(b) shows one instance of the macro place p3 in figure 2-2(a). They are collected into a subnet that further defines a macro place. The subnet in figure 2-2(b) has one input place p3 and one output place p9 and general S-System PN structure in between. The transition following the macro place may not be fired until the output place is active.

2.2.1.3 Describing logic states and arithmetic operations

The use of Boolean equations in some of the PN models only allows description of the signals of a controller. In this work, algebraic assignments and expressions are allowed in places and transitions; this extension draws inspiration from (Wallen, 1995) that extended Grafcet with general statements and expressions to describe complex control algorithms. Extension of Grafcet with algebraic assignments and expressions is also found in (Guillemaud and Gueguen, 1999)

which were used to describe hybrid systems. Besides, their work also incorporated differential equations and Laplace transfer functions in places and transitions.

General algebraic assignments and expressions are used here to describe all functions in the places and conditions in the transitions. The functions in a place include logic states of the controller, read/write operation, interrupts and serial peripheral interface protocol. The condition of a transition may take the form of tasks like testing the logic state of an input or solving arithmetic operations within the microcontroller.

The PN model in figure 2-2 illustrates the usage of assignments and expressions. Figure 2-2 shows an extended S-System PN to specify a switch pressed. When the switch, *pb* is pressed, it will illuminate *led1*. The second press resets *led1* and illuminates *led2*. The *count* variable in the model is used to store the number of presses. Each switch press will increase *count* by 1. The third press on *pb* will reset both *led1*, *led2* and *count*.

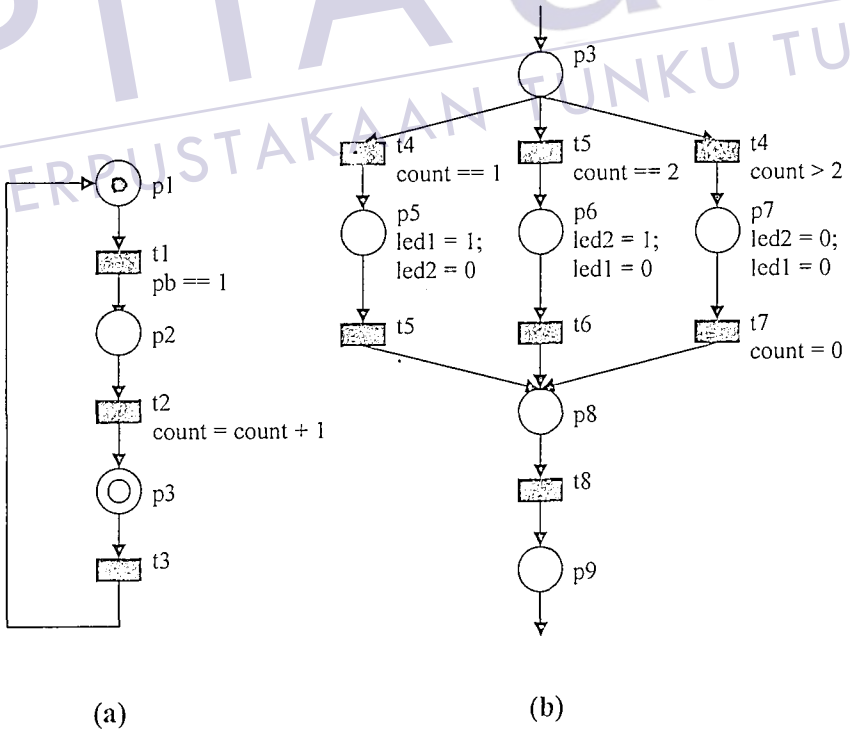


Figure 2-2 S-System PN model for a switch press mechanism

Expression *pb==1* in transition t1 is a condition that describes the pressing of switch *pb*. If the logic state of *pb* is equal to '1', transition t1 fires. Transition t2 is associated with an assignment *count=count+1*. This assignment increases the *count* variable by 1 each time switch *pb* is pressed. Transition t2 fires when its condition is fulfilled. This resulted in the activation of macro place p3. Macro place p3 is further defined by the subnet in figure 2-2(b).

The subnet in figure 2-2(b) has three transitions t4, t5 and t6 that are associated with expressions that evaluate the variable *count*. If *count==1* is true, *led1* illuminates. When *count==2* is true, *led2* illuminates. If *count>2* is true, *led1*, *led2* and the *count* variable will be reset.

It is shown in figure 2-2 that algebraic assignments and expressions can be used to describe logic state of input and output variables as well as solving arithmetic operations. Besides these, assignments are also used to deal with serial peripheral interface protocol and read/write to EEPROM.

2.2.1.4 Handling interrupts in microcontroller

Interrupts can be defined as the suspension of some activity by an event, and the resumption of the same activity later on from the state in which it was suspended. Interrupts in a microcontroller may come from hardware interrupts. For our target mid-range PIC microcontroller, hardware interrupts come from 4 sources:

- An external interrupt from PORT B bit 0.
- A change in state of PORT B bits 4 to 7
- TMR0 overflow from FFh to 00h
- Write completion to an EEPROM location

Hardware interrupts may disturb the normal flow of a microcontroller program to perform some other functions. The interrupt causes the program to jump to another section of code to handle the interrupt.

In handling interrupts in embedded systems, (Mirkowski and Yakovlev, 1998) associate a kind of a working permit to a portion of the net. The work permit can for instance be represented by a variable that guards every action in a given portion of the net. The normal working condition of the net would be when guarding variable is set to true. When there is an interrupt request the variable is set to false blocking usual net processing and allowing the interrupting process to take over. Once the interrupting process finishes, the guard variable is reset to true, and processing of the net resumes from where it was prior to interruption.

A similar approach is adopted in this work whereby a work permit is assigned to a portion of the S-System PN by a variable that guards a specific type of interrupt. Figure 2-3 shows a work permit at place p2 that handles a PORT B bit 0 external interrupt.

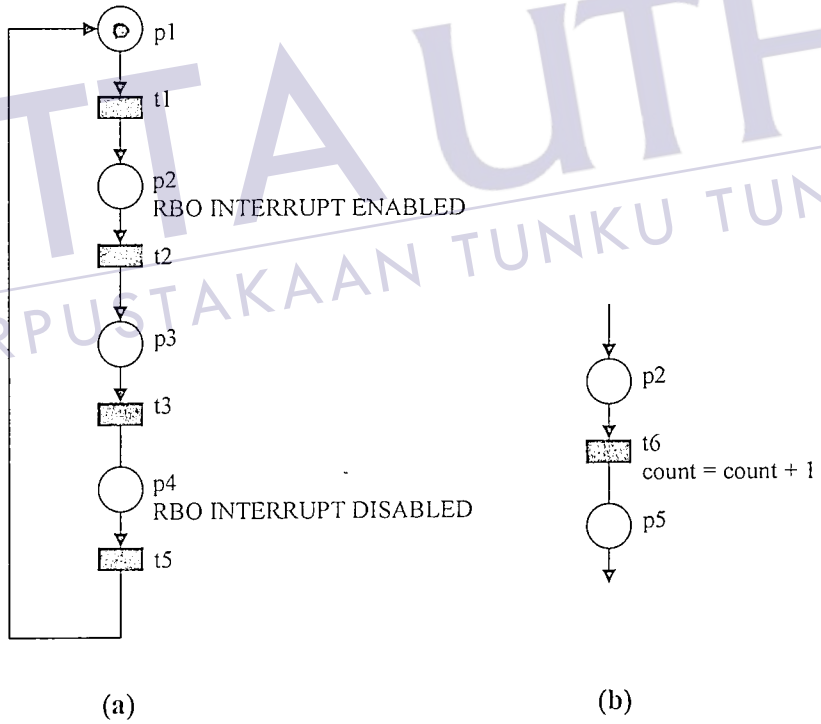


Figure 2-3 Handling PORT B bit 0 interrupt

The work permit in place p2 will guard every action of a portion of the net from p2, t2, p3, t3 and p4. Usual processing of the net will be blocked when a PORT B bit 0 interrupt occurs, allowing an interrupt routine to take over.

Another extension added to the S-System PN is requiring a kind of subnet to represent an interrupt routine. Figure 2-3(b) shows a subnet that represents an interrupt routine handling the PORT B bit 0 interrupt. The subnet shows that the *count* variable will be increased by 1 at every instance of a PORT B bit 0 interrupt. This subnet will be executed in the event of an interrupt occurring in the portion of the net from p2 to p4.

Work permit used in this work also applies to other types of interrupts such as TMR0 overflow, write completion to EEPROM and PORT B bit 4 to 7 state changes. When it is required for each interrupt enabled, a subnet needs to be constructed to handle that interrupt.

2.2.1.5 Describing timing in microcontrollers with timed transitions

In the case of PN, there are different mechanisms dealing with time according to the extension employed. Some extensions assign time mechanisms to transitions while others assign them to places. However, no PN extensions have used both approaches simultaneously. In most research, time is preferred to be associated with transitions. This can be found in (Tanabe, 1997) (Proth and Xie, 1996). In this work, time is assigned to be associated with transitions.

Let us assume that the time associated with a transition is τ, and that the firing of t (transition) starts at time T_0. Then, firing t of an S-System PN consists of:

- removing a token from pre-place of t at time T_0
- adding a token to a post-pace of t at time $T_0 + \tau$

Between instants T_0 and $T_0 + \tau$, the token is supposed to remain in the transition. This represents that the processes taking place in the system modeled would have some length in time, which will be represented as durations of firing transitions of a PN. The timed transition explained above is used to model timing in the microcontroller. Timed transition such as this is analogous to timers in PLC (Zhou and Venkatesh, 1999). In the context of the microcontroller, it represents timing and delay routines that need to be generated to produce the required timing.

Figure 2-4 illustrates the use of a timed transition t2 in a simple operation. The PN model specifies that a switch press at *pb* will illuminate *LED1* at p2. *LED1* will illuminate for 5 seconds when transition t2 fires. After 5 seconds, the token flows into p3 to reset *LED1*.

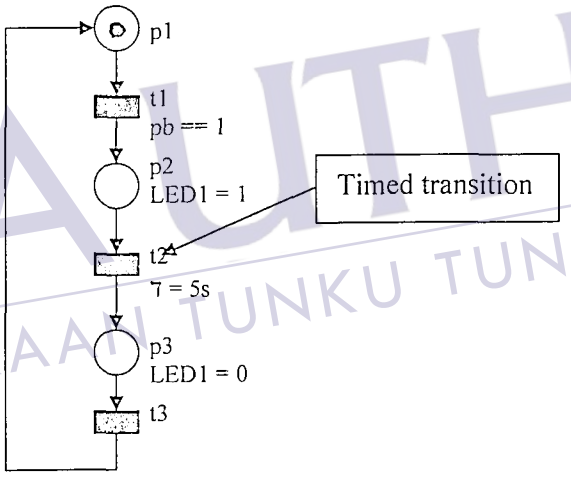


Figure 2-4 Timed transition

2.2.1.6 Token in an S-System PN

In this research, it is restricted that *only* one token can be used at any place in the net. The placement of a token shows the initial marking; it also represents the initial state of a microcontroller program. For example, the token in place p1of figure 2-4.

2.2.2 Application of the extended S-System PN

A mixing operation which is controlled by a microcontroller is shown in figure 2-5. The start button when pressed starts the mixing cycle. This resulted in activation of valve 2 and valve 1 to allow liquid to flow into the tank. When the liquid level reaches sensor 2, valve 1 and 2 are closed to prevent liquid from flowing in. Consequently, the motor starts spinning to stir the mixture for 60 seconds. After 60 seconds, the motor stops and valve 3 is opened to allow the mixture to flow out of the tank. When all the mixture has flown out of the tank, valve 3 is closed and the system is back to its initial state.

The wash cycle starts when the wash button is pressed. Valve 4 will be activated to allow water to flow into the tank. As water level reaches sensor 2, valve 4 is closed; the motor then starts spinning for another 60 seconds. After 60 seconds, valve 3 is opened to dispose water. Valve 3 is then closed and the system returns to its initial state.

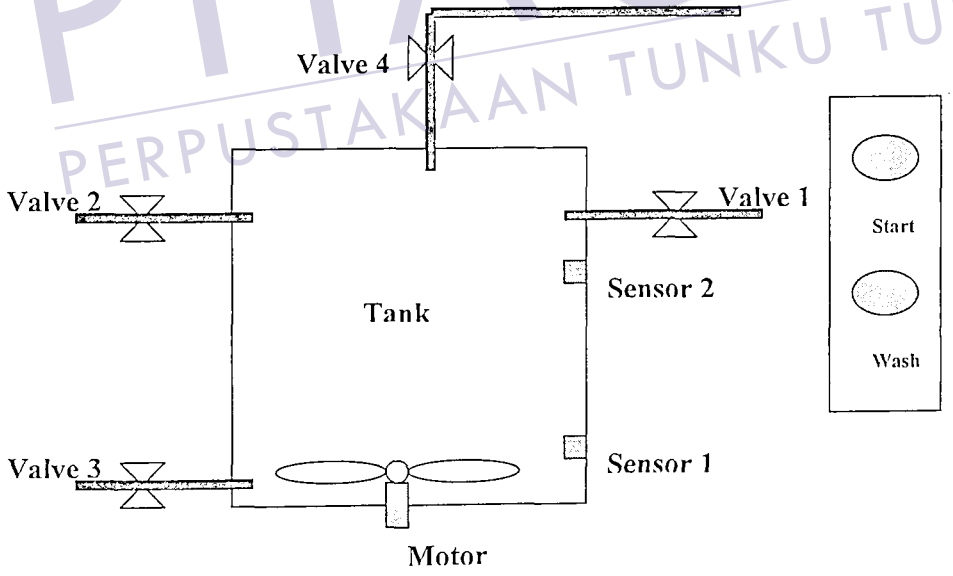


Figure 2-5 A mixing operation

The mixing operation is modeled by the extended S-System PN model shown in figure 2-6. The model consists of a main net and a subnet. The subnet is a collection of places and transitions for macro place p4 which is analogous to a subroutine in the microcontroller program. A subnet is modeled here to demonstrate to the reader how a macro place can be used to model a subroutine.

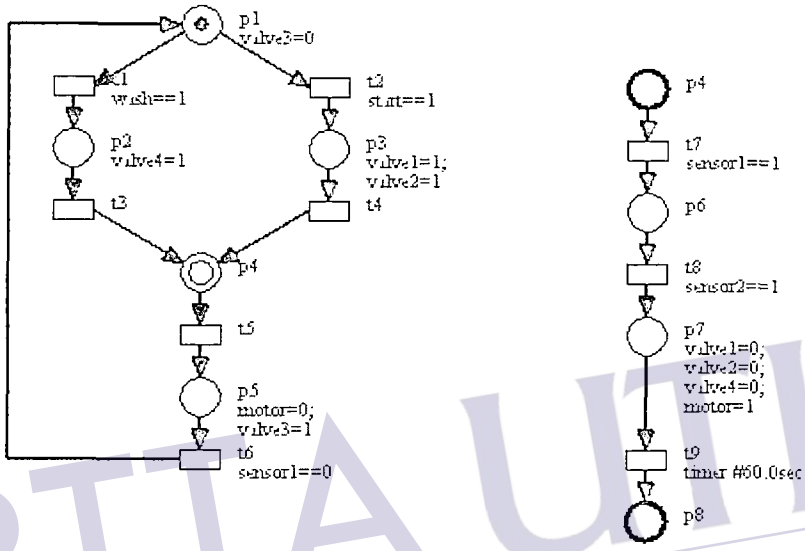


Figure 2-6 S-System PN model for the mixing operation

Each place and transition are associated with algebraic assignments or expressions instead of Boolean equations. The variables such as *wash*, *start*, *motor*, *valve1*, *valve2*, *valve3*, *valve4*, *sensor1* and *sensor2* are related to the microcontroller input and output logic states. For instance, *wash* is a variable that represents the push button that starts the wash cycle. When logic '1' is detected at *wash* (e.g. the push button is pressed), transition t1 fires and the token flows from p1 to p2. At p2, *valve4* is a variable that represents activation or deactivation of valve4. *valve4 = 1* means valve4 is activated.

Table 2.1 and 2.2 list all the places and transitions in the main net with their respective assignments or expressions and tasks. The number '1' and '0' in these assignments or expressions represent the logic states of the devices; '1' represents an ON state while '0' represents an OFF state.

Transitions t3, t4 and t5 in Table 2.2 are transitions that do not have any firing conditions (algebraic assignments or expressions) associated with them. These transitions are non-operational and can fire immediately when they are enabled, e.g. a token at its pre-place.

Table 2.1: Places in the main net

Place	Algebraic assignment / expression	Task
p4	Valve3 = 0	Valve3 OFF
p2	Valve4 = 1	Valve4 ON
p3	Valve1 = 1 valve2 = 1	Valve1 ON Valve2 ON
p4		Call a subroutine
p5	motor = 0 valve3 = 1	Motor OFF Valve3 ON

Table 2.2: Transitions in the main net

Transition	Algebraic assignment / expression	Task
T1	wash == 1	To test whether Push button *Wash* is ON
T2	start == 1	To test whether Push button *Start* is ON
T3		Non-operational
T4		Non-operational
T5		Non-operational
T6	sensor1 ==1	To test whether sensor1 is activated

Table 2.3 and 2.4 list all the places and transitions in the subnet with their respective algebraic assignments or expressions and tasks. A few places such as input place p4 and place p6 have no assignment or expression associated with them. These places are non-operational. Transition t9 in table 2.4 is an example of a timed transition. It creates a 60-second delay in the microcontroller.

Table 2.3: Places in the subnet

Place	Algebraic assignment / expression	Task
P4		Non-operational
P6		Non-operational
P7	valve1 = 0 valve2 = 0 valve 4 = 0 motor = 1	Valve1 OFF Valve2 OFF Valve 4 OFF Motor ON
P8		Return to main routine

Table 2.4: Transitions in the subnet

Transition	Algebraic assignment / expression	Task
t7	sensor1 == 1	To test whether sensor1 is activated
t8	sensor2 == 1	To test whether sensor2 is activated
t9	Timer #60 sec.	Execute a 60 seconds delay

This mixing operation model has illustrated a simple specification of a control system with the extended S-System PN proposed in section 2.2.1. Later parts of this work will demonstrate a few more examples of extended S-System PN models.

2.3 Control flow graphs and the S-System PN

As PN can be compared to ladder logic diagram of a PLC in (Zhou and Venkatesh, 1999)(Lee, Han and Lee, 2004); it is simply inapplicable in the context of a microcontroller as the implementation of a PN for the microcontroller must results in assembly code. The ladder logic diagrams however are generated specifically for the PLC. As the main aim of PN modeling for controllers is a generation of a piece of code; the PN is proven to be useful to produce low-level

coding such as IL in (Frey and Minas, 2000)(Minas and Frey,2002) and high level coding such as C code in (Melzer, 1997).

However, the SIPN model in (Frey and Minas, 2000)(Minas and Frey, 2002) to produce IL code for PLC cannot be adopted directly. Firstly, due to its limitation to over-describe or under-describe the PIC microcontroller as mentioned in chapter 1. Secondly, the proposed framework in their work is suitable for programs that are concurrent. Here, the IL can handle concurrency as it is supported by an operating system in the PLC. However, the program code in a microcontroller is straight-line and sequential where the program counter will only execute each line of program code sequentially. Hence, there arise a need to find a correlation between the PN structure and the PIC microcontroller coding.

A review on the work of (Aho, Sethi and Ullman, 1988) has shown that low-level language code can be segregated or structured with *control flow graphs*. A control flow graph is a representation of a program where contiguous regions of code without branches, known as basic blocks, are represented as nodes in a graph while edges between nodes indicate the possible flow of the program. The basic blocks in the control flow graph contain consecutive statements. The basic blocks have flow of control that enters at the beginning and leaves at the end.

As an illustration, a microcontroller program is shown as a control flow graph in figure 2-7. The microcontroller program can be divided into basic blocks. Figure 2-7 shows how a microcontroller program is segregated into three basic blocks named B1, B2 and B3 where each has its own consecutive statements. The basic blocks are connected by edges that indicate the flow of the program.

The edges in figure 2-7 indicate possibility of conditional and unconditional branching at the end of the basic blocks. For example, block B1 has a conditional jump at the end of the block. The statements *btfsc cut* and *btfsc drill* are evaluated. If *btfsc cut* is true, the flow of control branches to block B2. If *btfsc drill* is true, the flow of control branches to block B3. This is a kind of conditional jump. There are also unconditional jumps in flow graphs. Block B2 and block B3 have unconditional

jumping. There is no evaluation of certain conditions; these block directly branch back to B1 at the end of the block.

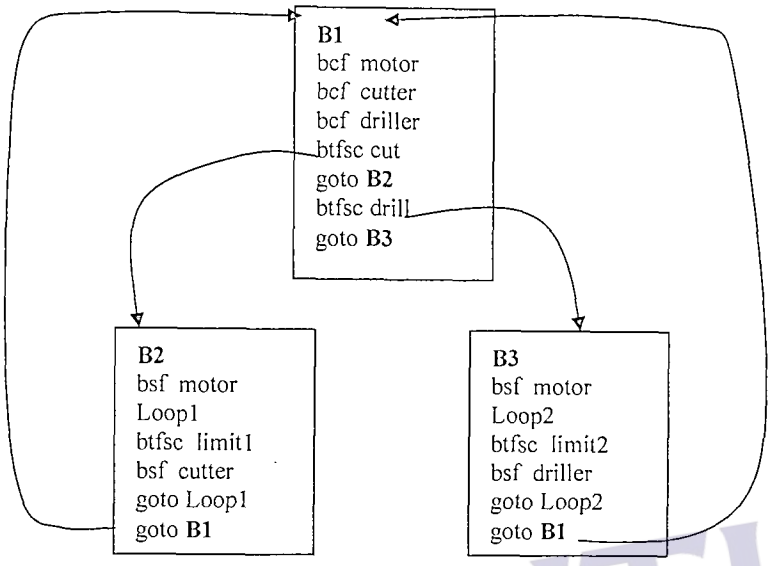


Figure 2-7 Control flow graph of a microcontroller program

Control flow graphs are useful structure for understanding code generation algorithms. They are good vehicles for a compiler to collect information about an intermediate code (Aho, Sethi and Ullman, 1988). They are used as representation of a *trace* for dynamic compilation in (Duesterwald, 2003). A *trace* is a dynamic sequence of consecutively executing basic blocks of a program. Thus traces are likely to offer opportunities to improved code layout and optimization. It is indicated here that control flow graphs can give the compiler a code layout of the program and a basis for further analysis.

We observe that we can categorize the basic blocks of a control flow graph into two types. In this work, we have termed the first type *block* and the second type as *switch*. Figure 2-8(a) shows a *block* that has only one output edge. A *switch* shown in figure 2-8(b) is a basic block with more than one output edges. For instance, block B1 in the control flow graph of figure 2-7 is a *switch* while B2 and B3 are both

blocks. A *block* can only branch to another basic block while a *switch* can branch to different basic blocks.

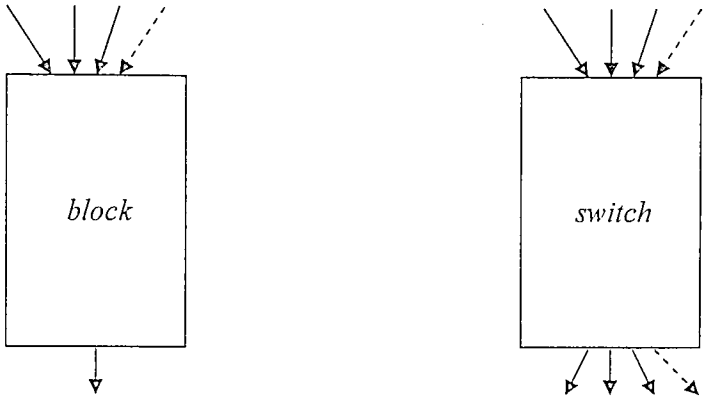


Figure 2-8 Basic blocks terminology

We further suggest in this work to find correlation between the PN net structures with control flow graphs. We found that the S-System PN provide a net structure that can be compared to the control flow graph of a program. Table 2.5 shows an S-System PN structure compared to a *switch* while table 2.6 shows an S-System PN structure compared to a *block*.

Table 2.5: Representations by PN for a *switch* node

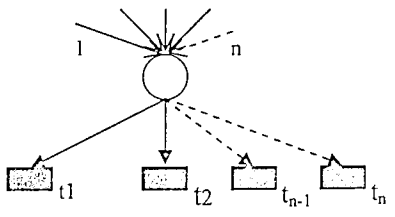
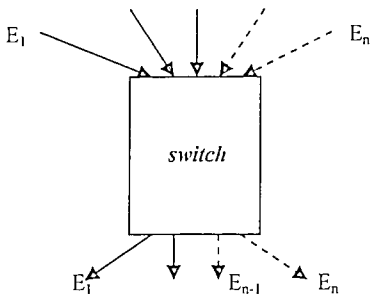
S-System PN	Control Flow Graph nodes
	

Table 2.6: Representations by PN for a *block* node

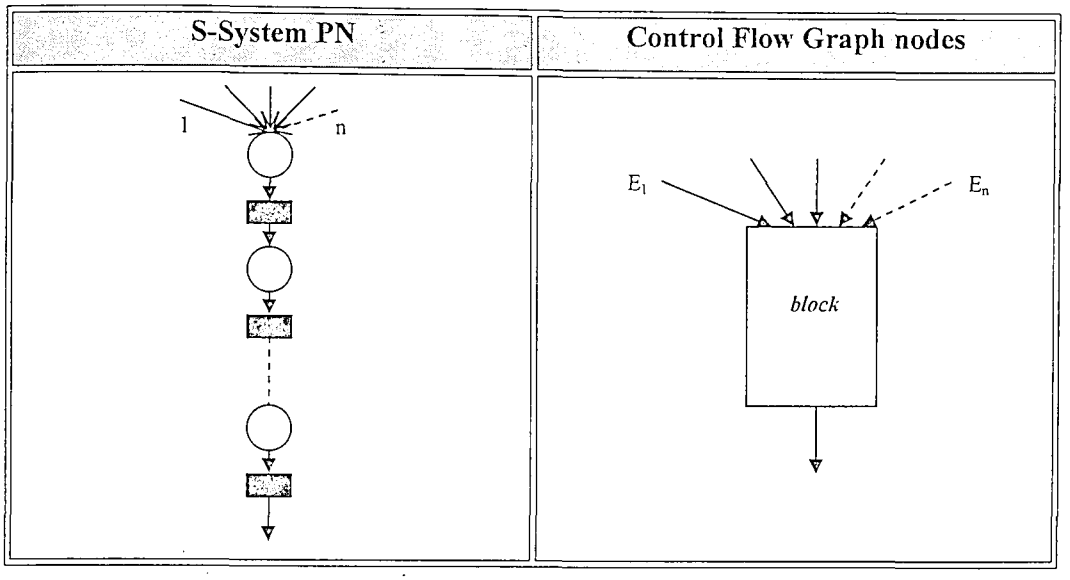


Table 2.5 and 2.6 show two types of S-System PN net structures that can be derived into nodes of a control flow graph. The derivation is an approach of containing graphs within graphs which can provide a more systematic graph layout for easier implementation of the S-System PN.

(Aho, Sethi and Ullman, 1988) presented basic blocks of a control flow graph that hold intermediate programs such as a three-address statement. Thus control flow graph has collections of information about the intermediate program, which is useful to the compiler. In this work, the transformation of a PN into a control flow graph adopts this approach. Here algebraic assignments and expressions are represented as intermediate code within basic blocks of the flow graph. What we need is to construct some kind of "transformer" to transform the PN net structure into control flow graph.

Figure 2-9 illustrates the transformation of a PN into a control flow graph, where we have used the mixing operation example in figure 2-6 to demonstrate transformation of a PN net structure into a control flow graph. The figure shows the corresponding control flow graph of the S-System PN model with its respective *block* and *switch* structure.

REFERENCES

Aho, A.V., Sethi, R. and Ullman, J.D. (1986). "Compilers: Principles, Techniques and Tools." Addison-Wesley.

Ammeraal, L. (1998). "Computer Graphics For Java Programmers." England. John Wiley & Sons Ltd.

Amroun, A. and Bolton, M. (1990). "Synthesis Of Controllers From Petri Net Descriptions And Application Of ELLA." In Claesen, L.J.M. (Ed.). "Formal VLSI Specification And Synthesis." VLSI Design Methods I. North -Holland.Elsevier Science Publishers. pp 291-308.

Bates, M. (2000) "The PIC 16F84 Microcontroller." Arnold.

David, R. (1995) "Grafcet: A Powerful Tool for Specification of Logic Controllers." IEEE Transactions on Control System Technology Vol. 3. No. 3.

Desroches, A.A. and Al-Jaar, R.Y. (1995). "Application of Petri Nets in Manufacturing Systems." IEEE Press marketing.

Drewes, F., Hoffmann, B. and Plump, D. (2000). "Hierarchical graph transformation. "in *Foundation of Software Science and Computation Structures (FOSSACS 2000)*,

Duesterwald, E. (2003). "Dynamic Compilation." In Srikant, Y.N. and Shankar, P.(Eds). " The Compiler Design Handbook: Optimizations and Machine Code Generation." USA. CRC Press LLC. pp. 739-762.

Fernandes, J.M., Adamski, M. and Proenca, A.J. (1997)."VHDL Generation From Hierarchical Petri Net Specifications of Parallel Controllers." IEEE Proceedings-E Computers and Digital Techniques. pp 127-137.

Frey, G. (2000). "Automatic Implementation of Petri Net based Control Algorithms on PLC." Proceedings of the American Control Conference ACC2000, Chicago.

Frey, G. and Litz, L. (2000). "Formal methods in PLC programming." Proceedings of the IEEE Conference on Systems Man and Cybernatics SMC 2000, Nashville.

Frey, G. and Minas, M. (2000) "Editing,Visualizing, and Implementing Signal Interpreted Petri Nets." *Proceedings of the AWPN 2000.* pp 57-62.

Ganapathi , M., Fischer, C.N. and Henessy, J.L. (1982) "Retargetable compiler code generation", *Computing Surveys 14.* pp 573-592.

Gau, C.H. and Hsiung, P.A. (2002) "*Time-memory scheduling and code generation of real-time embedded software*," In Proc. of the 8th International Conference on Real-Time Computing Systems and Applications (RTCSA'2002).

Graham, S.L. and Glanville, R.S. (1978). "A new method for compiler code generation." *Fifth Symposium on Principles of Programming Languages.* 231-240.

Guillemaud, L. and Gueguen, H. (1999). "Extending Grafcet for the specification of control hybrid systems." *IEEE SMC'99*, Tokyo.

Hoffmann, B. and Minas, M. (2000) "A generic model for diagram syntax and semantics." Proceedings of the Satellite Workshops of the 27th International Colloqium on Automata, Languages and Programming. No.8. pp 443-450.

Hsiung, P.A., Lee, T.Y. and Su, F.S. (2002). "Formal Synthesis and Code Generation of Real-Time Embedded Software using Time-Extended Quasi-Static scheduling," *apsec*, p. 395, Ninth Asia-Pacific Software Engineering Conference (APSEC'02).

Jorg, D. and Esparza, J. (1995). "Free Choice Petri nets." Cambridge University Press.

Lee, G.B., Han, Z. and Lee, J.S. (2004)."Automatic generation of ladder diagram with control Petri Net." Journal of Intelligent Manufacturing, Vol. 15, No. 2, pp.245-252.

Machado, R.J., Fernandes, J.M. and Proenca, A.J. (1997) "Specification of Industrial Digital Controllers with Object-Oriented Petri Nets", IEEE International Symposium on Industrial Electronics (ISIE '97). pp 78-83.

Katzen, S. (2003). "The Quintessential PIC Microcontroller." 2nd Edition. London. Springer-Verlag Ltd.

Kurdthongmee, W. (2003). "ertCPN: The adaptations of the coloured Petri-Net theory for real-time embedded system modeling and automatic code generation." Songklanakarin Journal of Science & Technology, 2003, 25(3): pp 381-394

Mak, R. (1996). "Writing Compilers and Interpreters: An Applied Approach Using C++." 2nd Edition. John Wiley & Sons.

Melzer, S. (1997) "Design and Implementation of a C-Code Generator for $B(PN)^2$." Institut fur Informatik, Univesitat Hildesheim.

Michel, G. (1990). "Programmable Logic Controllers: Architecture and Applications." England. John Wiley & Sons Ltd.

Minas, M. (1998). "Hypergraphs as a Uniform Diagram Representation Model", In Proc. 6th International Workshop on Theory and Application of Graph Transformations (TAGT '98), Germany.

Minas, M. (1999). "Creating Semantics Representation of Diagrams." Int. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '99) At Monastery Rolduc, NL.

Minas, M. (2002). "Specifying Graph-like Diagrams with DIAGEN." Electronic Notes in Theoretical Computer Science 72. No. 2.

Minas, M. and Frey, G. (2002). "Visual PLC-Programming using Signal Interpreted Petri Nets." Proceedings of the American Conference 2002 (ACC2002), Anchorage, Alaska. pp. 5019-5024.

Minas, M. and Viehstaedt, G. (1995). "A Generator for Diagram Editor Providing Direct Manipulation and Execution of Diagrams." IEEE Proc.of VL'95.

Mortenson, K.H. (1999). "Automatic Code Generation from Coloured Petri Nets for an Access Control System." Second Workshop on Practical Use of Coloured Petri Nets and Design. pages 41-58.

Mirkowski, J. and Yakovlev, A. (1998). "*A Petri Net Model for Embedded Systems.*" Proceedings of the Workshop on Design and Diagnosis of Electronic Circuits and Systems Szrzyck. Poland.

Muchnick, S.S. (1997) "Advanced Compiler Design and Implementation." USA. Academic Press.

Oliveira, W.L.A. and Marranghello, N. (2000). "A High-level Petri Net for Digital Systems." Proceedings of the XV SBMicro – International Conference on Microelectronics and Packaging. pp.220–225.

Predko, M. (2002). "Programming and Customizing the PIC Microcontroller." McGraw-Hill.

Proth, J.M. and Xie, X. (1996). "Petri Nets: A tool for design and management of manufacturing systems." England. John Wiley & Sons Limited. ()

Qin, W. and Malik, S. (2003). "Architecture Description Languages for Retargetable Compilation." In Srikant, Y.N. and Shankar, P.(Eds). " The Compiler Design Handbook: Optimizations and Machine Code Generation." USA. CRC Press LLC. pp. 535-564.

Rekers, J. and Schurr, A. (1995). "A Graph Grammar Approach to Graphical Parsing." Proc. VL'95 – 11th Int. IEEE Symop. On Visual Languages, Darmstadt, Germany. 195-202. IEEE CS Press, Los Alamitos, USA.

Rekers, J. and Schurr, A. (1996). "A Graph Based Framework for the Implementation of Visual Environments." IEEE Symp. on Visual Languages.

Su, F.S. and Hsiung P.A. (2002), "Extended quasi-static scheduling for formal synthesis and code generation of embedded software." Proc. of the l0th IEEE/ACM International Symposium on Hardware/Software Codesign (CODES'2002).

Tanabe, J.M. (1997). "Timed Petri Nets and Temporal Linear Logic." In Azema, P. and Balbo, G. (Eds). " *Application and Theory of Petri Nets-97"* New York. Springer-Verlag, pp. 156-174.

Wallen, A. (1995). "Using Grafcet To Structure Control Algoritms." Proceedings of The Third European Control Conference, Rome, Italy.

Watt, D.A. and Brown, D.F. (2000). "Programming Language Processors in Java: Compilers and Interpreters." England. Pearsons Education Ltd.

Watt, D.A. and Brown, D.F. (2001). "Java Collections : An Introduction to Abstract Data Types, Data Structures and Algorithms." England. John Wiley & Sons Ltd.

Zhang, K.B., Orgun, M.A. And Zhang, K. (2002). " Visual Language Semantics Specification in the VisPro System." Pan Sydney Area Workshop on Visual Information Processing (VIP 2002).

Zhou, M. and Venkatesh, K. (1999). "Modeling, Simulation and Control of Flexible Manufacturing System." Singapore. World Scientific.

(1998) "MPLAB IDE, Simulator, Editor Users' Guide." Microchip Technology.